Verifizierende Numerik mit PASCAL-XSC BCD-VERSION

NUMERISCHE ANWENDUNGEN

HILFS-MODULE BIBLIOTHEK I

Frithjof Blomquist
Saarbrücken

66346 Püttlingen 3
E-Mail Adressen: Blomquist@math.uni-wuppertal.de
ae29@rz.uni-karlsruhe.de

Ein Postsript- und dvi-File mit dem Inhalt dieses Buches kann über die Web-Seite

`www.math.uni-wuppertal.de/org/WRST/blom.html`

bezogen werden.

Dokumentationen zur Sprache **PASCAL-XSC** findet man im Netz unter

`www.uni-karlsruhe.de/~iam/html/literatur/download.html`

Im Literaturverzeichnis sind die entsprechenden **PASCAL-XSC** Bücher zusammen mit obiger Internet-Adresse angegeben.

Verifizierende Numerik mit PASCAL-XSC BCD-VERSION

NUMERISCHE ANWENDUNGEN

HILFS-MODULE BIBLIOTHEK I

Eine Numerik-Bibliothek für die BCD-Version von **PASCAL-XSC** auf Basis der Toolbox-Bände I,II.

Erste Auflage

Mit 22 Abbildungen

Vorwort

Mit der jetzt im Internet kostenlos zur Verfügung stehenden BCD-Version von **PASCAL-XSC** kann der Anwender eigene Programme und Module entwickeln, die mathematisch gesicherte Aussagen über numerische Ergebnisse ermöglichen. Die bei der IEEE-Version von **PASCAL-XSC** immer wieder notwendigen und lästigen Überlegungen zur Datenkonversion vom dezimalen zum binären Datenformat entfallen bei der BCD-Version, da in dieser Umgebung, wie bei einem Taschenrechner, die z.B. über die Tastatur eingegebenen Daten **exakt** gespeichert werden.

Ausführliche Beschreibungen der **PASCAL-XSC** Sprache mit vielen Beispielen findet man in [14, 15, 17]. Alle zusätzlichen Informationen zur BCD–Version von **PASCAL-XSC** liefert das Benutzer–Handbuch [16].

Zur Lösung vieler Grundaufgaben der Numerik existiert für die BCD–Version von **PASCAL-XSC** eine umfangreiche Modul–Bibliothek I. Im vorliegenden Buch findet der Anwender neben vielen Beispielprogrammen eine ausführliche Beschreibung aller Routinen dieser Bibliothek. Der vollständige Programm-Code zusammen mit allen Quelltexten und die oben genannten Handbücher stehen im Internet über die auf Seite 2 angegebene Adresse zur Verfügung.

Nahezu alle **PASCAL-XSC** Routinen der Numerik–Bibliothek I sind den beiden Toolbox–Bänden I,II [5, 8] entnommen und für die Datenformate `real,Longreal,rrG` der BCD–Version umgeschrieben worden.

Der bisher als Preprint existierende aber noch nicht veröffentlichte Toolbox Band II enthält **PASCAL-XSC** Routinen aus der Habilitationsschrift von Priv.-Doz. Dr. Rudolf Lohner [11, S.1-230] und diverse Beiträge von Prof. Dr. Walter Krämer aus dem Institut für Angewandte Mathematik der Universität Karlsruhe. Die Routinen aus [11] stehen für das IEEE-Format im Netz zur Verfügung.

Eine ausführliche Beschreibung der vielen Routinen zur Matrixinversion findet man in [3], auch dieser Beitrag steht über die auf Seite 2 angegebene Internet-Adresse zur Verfügung.

Die **Numerik–Bibliothek I** behandelt folgende Grundaufgaben:

- Integer–Langzahlarithmetik
- Reelle u. komplexe Staggered Intervall–Arithmetik
- Reelle u. komplexe Staggered Intervall–Arithmetik für Matrizen und Vektoren
- Zufallsgeneratoren
- Reelle u. komplexe Zufalls–Matrizen
- Reelle u. komplexe Testmatrizen
- Maximalgenaue Matrix/Vektor–Ausdrücke zum Typ `Longreal`
- Elementare Matrix/Vektor–Funktionen
- QR- und LU–Zerlegung reeller und komplexer Matrizen
- Inversion reeller und komplexer Matrizen, Pseudoinverse
- Reelle und komplexe lineare Gleichungssysteme
- Auswertung reeller und komplexer Polynome mit Punkt- u. Intervallkoeffizienten
- Skalare und vektorielle Differenzengleichungen
- Automatische Differentiation
- Nichtlineare Gleichungen
- Kurvendiskussion
- Nichtlineare Gleichungssysteme
- Globale Optimierung reeller ein- u. mehrdimensionaler Funktionen
- Nullstellen komplexer Polynome
- Numerische Integration
- Eigenwertproblem: $\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$

Dipl.-Math. Heiner Berlejung, Dr. Axel Facius und Dipl.-Math. Stefan Wedner haben in mühevoller Arbeit dafür gesorgt, daß das Laufzeitsystem der BCD–Version jetzt fehlerfrei funktioniert. Dafür möchte ich mich an dieser Stelle ganz herzlich bedanken. Mein besonderer Dank gilt auch Prof. Dr. U. Kulisch, Prof. Dr. W. Krämer, Dr. R. Klatte und allen Mitarbeitern am Insitut für Angewandte Mathematik der Universität Karlsruhe, ohne deren langjährige Unterstützung der Druck und die Fertigstellung dieses Buches nicht möglich gewesen wäre.

Saarbrücken, im September 2000 F. Blomquist

Inhaltsverzeichnis

Einleitung

Die Bearbeitung numerischer Aufgaben auf einem Rechner ist mit zwei grundsätzlichen Problemen verbunden:

1. Durch i.a. unvermeidbare Rundungsfehler bei den vier Grundoperationen erhält man lediglich Approximationen der exakten Ergebnisse, wobei eine Aussage über eine hinreichend kleine Oberschranke des absoluten oder relativen Fehlers oft nicht vorliegt. In diesen Fällen hat man keinerlei Anhaltspunkte für eine mathematische Beurteilung der Rechnerergebnisse.

2. Da die heutigen Rechenanlagen überwiegend mit dem binären `IEEE`–Datenformat arbeiten, werden fast alle Eingabedaten, wie schon die einfache Dezimalzahl 0.1, zur nächsten Rasterzahl gerundet, so daß ein sonst korrekter Algorithmus auch ohne die in Punkt 1. erwähnten Rundungsfehler nicht das exakte Ergebnis liefern kann.

Das in Punkt 1. geschilderte Problem läßt sich dadurch lösen, daß man die numerischen Aufgaben mit Hilfe der Routinen aus den **XSC**–Sprachen **C–XSC**, **PASCAL-XSC** oder **Fortran–XSC** löst. Diese Routinen liefern zu vielen mathematischen Grundaufgaben **garantierte** Einschließungen der exakten Ergebnisse.

Das 2. Problem wird mit den **XSC**–Sprachen meist dadurch gelöst, daß man die nicht darstellbaren Eingabedaten durch Maschinenintervalle einschließt, deren Randpunkte benachbarte Rasterzahlen sind. Diese Lösung hat jedoch zwei gravierende Nachteile:

- Sind die Eingabedaten Intervalle, so wird die Laufzeit i.a. deutlich vergrößert und die Ergebnisintervalle können sich insbesondere bei schlecht konditionierten Problemen so stark aufblähen, daß vernünftige Aussagen über das exakte Ergebnis nicht mehr möglich sind.

- Der große Nachteil besteht jedoch darin, daß die Einschließung der nicht darstellbaren Eingabedaten in der Praxis oft vergessen oder nicht vorgenommen wird, weil man der Auffassung ist, daß die minimalen Konversionsfehler auf das exakte Ergebnis keinen merklichen Einfluß haben.

Das folgende einfache Beispiel zeigt jedoch, daß die Nichtbeachtung der Konversionsfehler zu gravierenden Fehlern führen kann: Im `IEEE`–Format ist mit $a = 9.000000000000001$ und $b = 1$ die Differenz $a - b = 10^{-15}$ zu berechnen. Da a nicht darstellbar ist, wird a durch eine `read`–Anweisung zur nächsten Rasterzahl $\widetilde{a}$ gerundet, und die anschließende exakte Subtraktion $\widetilde{a} - b$ liefert das Maschinenergebnis $\widetilde{a} - b = 1.776 \ldots \cdot 10^{-15}$, das sich vom exakten Wert 10^{-15} um fast 78% unterscheidet! Es ist einleuchtend, daß weiterführende Rechnungen mit einem solchen Zwischenergebnis völlig unsinnig sind.

Nun weiß man aus Erfahrung, daß eine entsprechende Differenz auf einem Taschenrechner exakt gelöst wird[1]. Der Grund dafür ist, daß alle Taschenrechner mit einem BCD (**B**inary **C**oded **D**ecimal)–System arbeiten, in dem dezimale Eingabedaten bis zur jeweils vorgegebenen Mantissenlänge exakt, d.h. ohne Konversionsfehler gespeichert werden.

Genau diesen Vorteil liefert die BCD–Version von **PASCAL-XSC** , die mit den internen `real`- `Longreal`- und `rrG`–Formaten das Speichern von Eingabedaten mit 13, 21 und 26 Dezimalstellen ohne Konversionsfehler ermöglicht. Da die BCD–Version wegen der aufwendigeren Datenspeicherung deutlich langsamer ist als eine `IEEE`–Version von **PASCAL-XSC** , dafür aber wegen der fehlenden Konversionsfehler gerade für den mathematischen Anfänger sehr viel einfacher zu handhaben ist, empfiehlt sich ihr Einsatz besonders im Ausbildungsbereich an Schulen und Universitäten. Ein weiterer Vorteil liegt in der Sprache **PASCAL-XSC** selbst, mit der man erfahrungsgemäß den weitaus einfachsten Zugang zur **XSC**–Philosophie erhält, nach der exakte Ergebnisse durch Intervalle stets **garantiert** eingeschlossen werden. Im Gegensatz zur Sprache **C** kann nämlich ein in **PASCAL-XSC** geschriebener Algorithmus intuitiv erfaßt werden.

Es ist Aufgabe dieses Buches, den in **PASCAL-XSC** noch ungeübten Leser möglichst schnell in die Lage zu versetzen, die auf Seite 6 aufgelisteten numerischen Grundaufgaben verifiziert lösen zu können. Dazu steht eine Sammlung von 106 meist sehr einfachen Beispielprogrammen im Quelltext zur Verfügung, die nur übersetzt und dann gestartet werden müssen; `Pascal`–Kenntnisse sind also zunächst nicht erforderlich. Die einzelnen Programme sind in den Tabellen ab Seite 691 zusammengestellt.

[1]Kein Mensch würde einen Taschenrechner benutzen, der z.B. die einfache Differenz $1.000000001 - 1 = 10^{-9}$ nicht exakt berechnen könnte!

Will man jedoch eigene Programme entwickeln, so kann man sich an den vielen einfachen Beispielen orientieren und lernt dabei **PASCAL-XSC** praktisch von selbst. Lediglich die Handhabung von Prozeduren, Funktionen und Operatoren sollte man in [12] nachlesen.

Die auf Seite 6 angegebenen Grundaufgaben werden von insgesamt 145 Hilfsmodulen abgedeckt, die in den Tabellen ab Seite 697 kurz beschrieben werden. Eine ausführliche Dokumentation der exportierten Routinen dieser Module findet der Leser in den einzelnen Kapiteln dieses Buches, dabei wird die Bedeutung der jeweiligen Ein- und Ausgangsparameter genau beschrieben, um eine sichere Anwendung der Prozeduren zu gewährleisten. Die exportierten Prozeduren und Funktionen sind zusätzlich in den Tabellen ab Seite 667 bzw. 679 zusammengestellt. Alle zur Verfügung stehenden Datentypen findet man in den Tabellen ab Seite 661.

Der Rahmen dieses Buches wäre deutlich gesprengt worden, wenn man bei jeder Routine zusätzlich auch noch den selbstverifizierenden Algorithmus ausführlich beschrieben hätte. Mathematisch interessierte Leser finden die benutzten Algorithmen in den Toolbox-Bänden [5, 8], zusammen mit einer Fülle weiterer Literaturangaben. In [3] findet man alle Grundlagen zu den vielen Routinen der Matrix–Inversion.

Eine gute Übersicht aller behandelten Grundprobleme liefert sowohl das ausführliche Inhaltsverzeichnis als auch das Stichwortverzeichnis am Ende des Buches, wobei durch die vielen Querverweise zusätzliche Orientierungshilfen gegeben werden.

Der modulare Aufbau von **PASCAL-XSC** ermöglicht es dem Anwender, eigene Routinen in weiteren Hilfsmodulen zusammenzufassen, wodurch das System nach Wunsch beliebig erweitert werden kann. Einen guten Rat möchte ich an dieser Stelle jedem Anwender geben: Investieren Sie unbedingt die erforderliche Zeit, um neu entwickelte Routinen zu testen und vor allem sehr ausführlich zu dokumentieren! Bei umfangreichen Routinen ist es erfahrungsgemäß sehr nervend, die genaue Arbeitsweise eines Programms ohne ausreichende Beschreibung zu rekonstruieren.

Ich wünsche dem Leser nach einer unvermeidbaren Einarbeitungszeit in ein so umfangreiches System viel Spaß bei den numerischen Untersuchungen. Beachten Sie bitte, daß die mit den **PASCAL-XSC** Routinen berechneten Intervalle die exakten Ergebnisse garantiert einschließen!

Im nächsten Abschnitt findet man einige nützliche Informationen über eine mögliche Konfiguration der BCD–Version von **PASCAL-XSC** unter `SuSe Linux`.

Installation

Übersicht

Die Installation und Konfiguration der BCD-Version von **PASCAL-XSC** ist unter SuSe Linux 6.0 denkbar einfach! Die über das Internet bezogenen Dateien readme, install.BCD, pxsc_BCD.tgz kopiert der Anwender[2] z.B. in sein Heimatverzeichnis ~. Mit

```
install.BCD <enter>
```

und nach der interaktiven Eingabe des Verzeichnisses für die Module der Numerik-Bibliothek, die im vorliegenden Buch ausführlich beschrieben wird, erhält man im Unterverzeichnis pxsc_BCD ein lauffähiges BCD-System von **PASCAL-XSC**. Nach dem Start eines neuen xterm-Fensters kann dann in diesem Fenster im Verzeichnis ~/pxsc_BCD durch den Aufruf des interaktiven Managers dxsc mit dxsc test_BCD <enter> das Beispielprogramm test_BCD durch

```
c  <enter>  (Uebersetzen des Quelltextes)
r  <enter>  (Programm-Start             )
   <enter>  (Keine Programm-Parameter   )
```

sofort übersetzt und gestartet werden. Durch die Installation werden Informationen bzgl. der Umgebungsvariablen PATH, PXSC_SYS, PXSC_EDIT nur in die Datei ~.profile geschrieben, die bei jedem Login-Vorgang neu gelesen wird. Zur Programm- und Modul-Entwicklung ist der emacs-Editor voreingestellt, es kann jedoch auch auf den vi-Editor umgeschaltet werden. Hat man im Unterverzeichnis lib1 eines der Module der Numerikbibliothek geändert oder ein neues Modul hinzugefügt, so kann man nach einem entsprechenden Eintrag in Makefile.nms durch make lib1 <enter> alle Module der Numerik-Bibliothek neu übersetzen und damit die Bibliothek nach Bedarf beliebig erweitern.

[2] Bei der Installation sind Aktionen des Systemverwalters nicht erforderlich.

Installation der BCD–Version

Download vom Internet

Über die auf Seite 2 angegebene Adresse meiner Web-Seite sind die drei Dateien

`readme, install.BCD, pxsc_BCD.tgz`

zunächst in ein Verzeichnis des Anwenders zu laden.

`readme` liefert Informationen zur Installation in englischer Sprache. Die Datei `pxsc_BCD.tgz` enthält die vollständige BCD-Version von **PASCAL-XSC** in gepackter Form, und mit dem Shell-Skript `install.BCD` wird die eigentliche Installation gestartet, wobei dann `pxsc_BCD.tgz` automatisch entpackt wird.

Verzeichniswahl

Wir nehmen an, daß sich die obigen drei Dateien im absoluten Verzeichnis `/xyz` befinden, wobei dieses symbolische Verzeichnis das Heimatverzeichnis oder irgendein anderes Verzeichnis des Anwenders bedeuten soll. Startet man dann dieInstallation mit `install.BCD <enter>` , so wird das Unterverzeichnis

`pxsc_BCD`

erzeugt, in das die BCD-Version von **PASCAL-XSC** kopiert wird, d.h. die BCD-Version befindet sich dann im absoluten Verzeichnis: `/xyz/pxsc_BCD`

Durchfürung der Installation

Die Dateien `readme, install.BCD, pxsc_BCD.tgz` befinden sich also im Verzeichnis `/xyz`; die Installation wird dann aus dem gleichen Verzeichnis gestartet durch: `install.BCD <enter>`
Nach etwa zwei Minuten wird man aufgefordert, durch `r <enter>` das Konfigurationsprogramm `pxsccfg` zu starten. Die nachfolgenden Eingaben

```
i                    <enter>
/xyz/pxsc_BCD/lib1/  <enter>
e                    <enter>
```

beenden mit einer entsprechenden Meldung die erfolgreiche Installation.

Die erzeugten Unterverzeichnisse

Durch die Installation werden im Verzeichnis `pxsc_BCD` die folgenden Unterverzeichnisse erzeugt:

- `install` enthält: `readme, install.BCD, pxsc_BCD.tgz`
- `bin` enthält die ausführbaren Programme: `bin2dec, ED_vi, ED_emacs, pxsc, exsc, mxsc, dxsc, psclist, pxsccfg`
- `sys` enthält alle System-Dateien der BCD-Version
- `lib1` enthält alle Module der Numerik-Bibliothek
- `lib1_exap` enthält alle Beispielprogramme zur Numerik-Bibliothek

Installationstest

Mit Hilfe der im Verzeichnis `/xyz/pxsc_BCD` vorhandenen Quell-Datei `test_BCD.p` läßt sich die Installation aus dem gleichen Verzeichnis wie folgt testen:

```
dxsc test_BCD  <enter>  (dxsc:  interaktiven Manager)
c              <enter>  (Uebersetzen der Quell-Datei)
r              <enter>  (Programmstart             )
               <enter>  (Keine Programmparameter    )
```

Bei erfolgreichem Programmablauf erfolgt die Meldung:

```
+++++++++++++++++++++++++++++++++++++++++++
+  Installation and program execution ok  +
+++++++++++++++++++++++++++++++++++++++++++
```

Wechsel des Editors

Nach der Installation ist zur Programm- und Modul-Entwicklung der `emacs`-Editor voreingestellt. Durch einen der folgenden Programmaufrufe kann ein anderer Editor gewählt werden:

```
. ED_vi     <enter>      (Wechselt zum vi-Editor   )
. ED_emacs  <enter>      (Wechselt zum emacs-Editor)
```

Bitte vergessen Sie nicht den Punkt mit dem folgenden Leerzeichen vor dem entsprechenden Programm-Namen!

Die Datei `.profile`

Während der Installation der BCD-Version wird im Heimatverzeichnis des Anwenders an das Ende der Datei `~/.profile` folgendes eingetragen:

```
PATH=/xyz/pxsc_BCD/bin:$PATH
export PATH

unset PXSC_SYS
PXSC_SYS=/xyz/pxsc_BCD/sys/
export PXSC_SYS

unset PXSC_EDIT
PXSC_EDIT=emacs
export PXSC_EDIT
```

dabei ist `/xyz` der symbolische Verzeichnisname, vergl. Seite 28.

Die Umgebungsvariable `PATH` wird zunächst um das obige Verzeichnis `/xyz/pxsc_BCD/bin` erweitert, womit dann die neun Programme aus dem Verzeichnis `bin` von jedem Verzeichnis aus aufgerufen werden können.

Mit Hilfe der Umgebungsvariablen `PXSC_SYS` findet der Pascal-Compiler `pxsc` im Verzeichnis `/xyz/pxsc_BCD/sys/` alle notwendigen System-Module und Hilfsdateien.

Über die Umgebungsvariable `PXSC_EDIT` wird der `emacs`-Editor voreingestellt. Beachten Sie bitte, daß bei jedem Login-Vorgang die Datei `.profile` im Heimatverzeichnis des jeweiligen Anwenders gelesen wird. Wenn also an Stelle des `emacs` der `vi`-Editor voreingestellt werden soll, so muß in `.profile` lediglich die vorletzte Zeile ersetzt werden durch: `PXSC_EDIT=vi`.

Zurückschalten zur BCD-Version

Da die IEEE- und BCD–Version von **PASCAL-XSC** jeweils den gleichen Pascal → C-Compiler `pxsc` benutzen, können beide Versionen auf die gleichen Programme im Verzeichnis `/xyz/pxsc_BCD/bin` zurückgreifen, so daß die Umgebungsvariable `PATH` zum Arbeiten mit der IEEE–Version von **PASCAL-XSC** nicht geändert werden muß!

Beide Versionen benutzen jedoch unterschiedliche System-Module und verschiedene Laufzeitsysteme `rts.a`, so daß beim Arbeiten mit der IEEE–Version die Umgebumgsvariable `PXSC_SYS` ein anderes Verzeichnis enthalten muß. Will man jedoch nach der Arbeit mit der IEEE–Version zur BCD–Version zurückschalten, so muß `PXSC_SYS` wieder das entsprechende Verzeichnis der BCD–Version erhalten, was durch den folgenden Shell-Skript-Aufruf

```
. bin2dec   <enter>
```

realisiert werden kann. Vergessen Sie nicht den Punkt und das nachfolgende Leerzeichen vor dem Programm-Namen `bin2dec`, da sonst die Umgebungsvariable `PXSC_SYS` nur in der Sub-Shell von `bin2dec` geändert wird!

Sicherheitsfragen

Die größte Sicherheit gegen unbeabsichtigten Datenverlust liefert erfahrungsgemäß eine möglichst tägliche Datensicherung auf externe Datenträger wie z.B. Disketten oder CD-ROMs.

Sollten in einem unverhofften Unglücksfall Dateien oder ganze Verzeichnisse der BCD–Version von **PASCAL-XSC** zerstört worden sein, so gibt es für die Restauration die beiden folgenden Möglichkeiten:

1. Restauration mit Hilfe der externen Datensicherung
2. Neuinstallation der BCD–Version.

Beachten Sie aber, daß bei einer Neuinstallation alle zwischenzeitlich entwickelten Programme oder Module nicht mehr bereitgestellt werden können!

Da also Datenverluste stets mit Arbeit und Stress verbunden sind, sollte man sich Strategien überlegen, wie solche Unglücksfälle möglichst vermieden werden können. Dazu kann man von folgender UNIX-Eigenschaft Gebrauch machen:

> Dateien aus einem `root`–Verzeichnis können nur vom Super-User gelöscht werden, vorausgesetzt, daß die Benutzer *others* in diesem `root`–Verzeichnis keine Schreibrechte besitzen.

Danach kann man das unbeabsichtigte Löschen von Dateien in den Verzeichnissen

`bin, install, lib1, lib1_exap, sys`

dadurch verhindern, daß der Systemverwalter die obigen Verzeichnisse als sein Eigentum erklärt und anschließend allen anderen Benutzern die Schreibrechte bzgl. dieser Verzeichnisse entzieht. Bevor wir dazu die entsprechenden Linux–Befehle angeben, werden zunächst die Eigentumsverhältnisse gezeigt, wie sie sich unmittelbar nach der Installation der BCD–Version für den Eigentümer `blom` und seiner Gruppe `users` ergeben:

```
-rw-r--r--   1 blom   users     2072 Nov  2 01:19 Makefile
-rw-r--r--   1 blom   users    11781 Nov  2 01:19 Makefile.nms
drwxr-xr-x   2 blom   users     1024 Nov  6 12:58 bin
drwxr-xr-x   2 blom   users     1024 Nov  3 14:48 install
drwxr-xr-x   2 blom   users    11264 Nov  2 14:49 lib1
drwxr-xr-x   2 blom   users     3072 Oct 23 14:47 lib1_exap
drwxr-xr-x   2 blom   users     2048 Nov  2 14:49 sys
-rw-r--r--   1 blom   users      214 Oct 24 13:10 test_BCD.p
```

Im Verzeichnis `/xyz/pxsc_BCD` erhält man mit den Anweisungen:

```
su                                              <enter>
Password-Eingabe                                <enter>
chown root:root bin install lib1 lib1_exap sys  <enter>
chmod o-w bin install lib1 lib1_exap sys        <enter>
chown root:root Makefile Makefile.nms           <enter>
chmod o=r Makefile Makefile.nms                 <enter>
exit                                            <enter>
```

für die Dateien und Verzeichnisse der BCD–Version die folgenden Eigentumsverhältnisse:

```
-rw-r--r--   1 root   root      2072 Nov  2 01:19 Makefile
-rw-r--r--   1 root   root     11781 Nov  2 01:19 Makefile.nms
drwxr-xr-x   2 root   root      1024 Nov  6 12:58 bin
drwxr-xr-x   2 root   root      1024 Nov  3 14:48 install
drwxr-xr-x   2 root   root     11264 Nov  2 14:49 lib1
drwxr-xr-x   2 root   root      3072 Oct 23 14:47 lib1_exap
drwxr-xr-x   2 root   root      2048 Nov  2 14:49 sys
-rw-r--r--   1 blom   users      214 Oct 24 13:10 test_BCD.p
```

Beachten Sie bitte, daß mit den jetzigen Eigentumsverhältnissen die Durchführung der Anweisungen `make lib1` bzw. `make sys` von Seite 27 bzw. 36 nicht mehr funktionieren kann, da die System- und Hilfsmodule in den Verzeichnissen `sys` und `lib1` vom Anwender `blom` nicht überschrieben werden können. Vor der Durchführung dieser Anweisungen müssen also die Eigentumsverhältnisse vom Anfang dieser Seite wiederhergestellt werden!

Programm- und Modulentwicklung

Wir kommen jetzt zu dem für den Anwender vermutlich wichtigsten Punkt:

Die Programm- und Modulentwicklung

Dazu stehen im Verzeichnis `/xyz/pxsc_BCD/bin/` die Programme `dxsc` und `mxsc` zur Verfügung, und da das obige Verzeichnis in der Umgebungsvariablen `PATH` des jeweiligen Anwenders gespeichert ist, kann die Programmentwicklung aus jedem Verzeichnis heraus erfolgen. Bearbeitet man verschiedene numerische Gebiete, so sollte man aus Gründen der Übersichtlichkeit jeweils geeignete Unterverzeichnisse anlegen.

Die Übersetzung eines **PASCAL-XSC** Quelltextes erfolgt entweder mit dem interaktiven Manager `dxsc` [13, Seite 9,12..19,29] oder mit dem durch Optionen gesteuerten Batch Manager `mxsc` [13, Seite 11,29].

Der interaktive Manager `dxsc`

Mit dem Aufruf[3]

`dxsc test_BCD.p` <enter>

öffnet sich das Hauptmenü des `dxsc` Systems, und durch wiederholte Aufrufe des Editors (`e`) und Compilers (`c`) kann der Quellcode in `test_BCD.p` solange bearbeitet werden, bis durch die Meldung

`linkage complete`

angezeigt wird, daß die Übersetzung erfolgreich war, und mit

`r` <enter>

[3] In **PASCAL-XSC** müssen Quelldateien die Erweiterung `.p` besitzen.

kann anschließend das Programm `test_BCD` gestartet werden. Mit

`q` <enter>

verläßt man den interaktiven Manager `dxsc`. Weitere Einzelheiten auch zu den Programmparametern, d.h. zu den File Variablen und Laufzeitoptionen findet man in [13, Seite 12..19,31..38]. Programmparameter können nach dem Buchstaben `r`, jeweils durch Leerzeichen getrennt, aufgelistet werden:

`r` <Programmparameter> <enter>

Sind jedoch keine Programmparameter erforderlich, so wird man dennoch aufgefordert, solche einzugeben. Einfach mit <enter> läßt sich dann das Programm auch ohne Programmparameter starten.

Compileroptionen [13, Seite 15,20..23] lassen sich ganz analog nach dem Buchstaben `c` angeben.

Der Batch Manager `mxsc`

Im Gegensatz zum interaktiven Manager `dxsc` benötigt der Batch Manager `mxsc` keinerlei spätere Eingaben des Anwenders. Die Arbeitsweise von `mxsc` wird nur durch Compiler- oder(und) Manageroptionen bestimmt, die hinter `mxsc` nach einem Leerzeichen einzugeben sind [13, Seite11]. Diese Optionen werden durch ein Leerzeichen beendet, gefolgt vom Namen des Quelltext-Files `test_BCD.p`, wobei mit jedem `mxsc` Aufruf nur jeweils ein Quelltext-File angegeben werden kann. Das folgende Beispiel

`mxsc _e _x test_BCD.p` <enter>

ruft zunächst den Editor (`_e`) auf, mit dem `test_BCD.p` bearbeitet werden kann. Nach dem Verlassen des Editors erfolgt sofort die Programmübersetzung. Ist diese fehlerfrei, so wird das Programm wegen (`_x`) anschließend direkt ausgeführt, wobei allerdings Programmparameter nicht berücksichtigt werden.

Soll ein Programm jedoch mit Programmparametern (File-Variable oder Laufzeit-Optionen) ausgeführt werden, so sind folgende Schritte erforderlich:

1. `mxsc test.p` <enter>
   Bei fehlerfreier Übersetzung wird durch den obigen Aufruf das ausführbare Programm `test` erzeugt.

2. `test` <Programmparameter> <enter>
   startet `test` mit den angegebenen Programmparametern.

Da der oben beschriebene Batch Manager `mxsc` nur mit Optionen gesteuert wird, können mit seiner Hilfe in Shell-Skripts **PASCAL-XSC** Programme oder Module übersetzt werden.

Das Erweitern der Numerik-Bibliothek

Neben den Systemmodulen, die im Verzeichnis[4]

`/xyz/pxsc_BCD/sys`

gespeichert sind und die Grundstruktur der BCD–Version von **PASCAL-XSC** definieren, existiert noch eine Vielzahl weiterer Hilfsmodule, mit denen die verschiedensten numerischen Gebiete bearbeitet werden können. Diese Hilfsmodule der Numerik-Bibliothek1 befinden sich alle im Verzeichnis

`/xyz/pxsc_BCD/lib1`

Die Entwicklung eines neuen Numerik–Moduls wird man im Verzeichnis

`/xyz/pxsc_BCD`

vornehmen. Nachdem alle Fehler im Quelltex des Moduls `modul1.p` beseitigt wurden, kopiert man die Datei

`modul1.p`

ins Verzeichnis der schon existierenden Hilfsmodule

`/xyz/pxscd_BCD/lib1`

Wir nehmen jetzt an, daß in der `use`-Zeile des Modulquelltextes folgendes eingetragen ist:

```
use ci_ari, nlss, smcf1;
```

Danach importiert `modul1` das System–Modul `ci_ari` und aus dem Verzeichnis `/xyz/pxsc_BCD/lib1` die Numerik–Module `nlss` und `smcf1`.

Um die durch das Modul `modul1` erweiterte Numerik–Bibliothek neu zu übersetzen, sind die folgenden Schritte notwendig:

1. Im Verzeichnis `/xyz/pxsc_BCD` ist mit Hilfe eines Editors die Datei `Makefile.nms` zu öffnen. In Zeile 170 ist hinter `stgr.h` ein backslash \ und in der nächsten Zeile unmittelbar unter `stgr.h\` ist `modul1.h` einzutragen.

[4] `/xyz` ist der symbolische Name eines Anwender-Verzeichnisses, vergl. Seite 28

2. Am End der Datei `Makefile.nms` ist dann noch
   ```
   modul1.h : modul1.p nlss.p smcf1.p
           $(XSC) modul1.p
   ```
   genau in der oben angegebenen Form anzufügen, wobei die letzte Zeile mit einer TAB-Einrückung beginnen muß!

3. `make lib1   <enter>`
   startet dann die Übersetzung aller Module in der Numerik–Bibliothek.

Beachten Sie bitte, daß oben im Punkt 2. die importierten System–Module, hier also `ci_ari`, nicht einzutragen sind!

Abschließend sei noch angemerkt, daß im Verzeichnis `/xyz/pxsc_BCD` durch den Aufruf

```
make sys   <enter>
```

alle System–Module aus dem System–Verzeichnis `/xyz/pxsc_BCD/sys` und alle in `Makefile.nms` eingetragenen Numerik–Module aus dem Verzeichnis `/xyz/pxsc_BCD/lib1` übersetzt werden. Diese letzte Aktion wird aber nur dann notwendig, wenn aus irgendeinem Grunde eines der System–Module geändert werden mußte!

Beachten Sie bitte, daß die Anweisungen `make lib1` und `make sys` nur funktionieren, wenn die Eigentumsverhältnisse der betroffenen Dateien und Verzeichnisse vorher, wie auf Seite 32 oben beschrieben, realisiert wurden. Eigentumsänderungen dürfen nur vom Super-User `root` vorgenommen werden!

Kapitel 1

Zufallsgeneratoren

In diesem Kapitel werden Zufallsgeneratoren behandelt, die auf folgender Rekursionsformel basieren:

$$r_{j+1} = (a \cdot r_j + c) \ \mathbf{mod}\ m \tag{1.1}$$

Solche Generatoren arbeiten nach der linearen Kongruenz–Methode und erzeugen eine Folge von ganzen Zahlen $r_1, r_2, r_3 \ldots$ zwischen 0 und $m-1$. Falls die Rekursion (1.1) eine bereits generierte Zahl r_i neu berechnet, so beginnt eine Periode, die nicht größer sein kann als m selbst. Mit geeigneten Zahlen m, a, c erhält man eine maximale Periode, d.h. alle Zahlen zwischen 0 und $m-1$ werden dann irgendwann einmal angenommen.

1.1 Algorithmus

Im Toolbox-Band II [8, Chapter 8] findet man die Realisierung von drei Zufallsgeneratoren mit den entsprechenden Literaturangaben. Wir benutzen hier die einfachste Version von Park und Miller, die für die meisten Anwendungen völlig ausreichend ist. Die Rekursionsgleichung lautet jetzt:

$$\begin{aligned} r_{j+1} &= (7^5 \cdot r_j) \ \mathbf{mod}\ (2^{31}-1), \quad \text{d.h.} \\ m &:= 2147483647, \quad a := 16807; \quad c := 0; \end{aligned} \tag{1.2}$$

Natürlich darf $r_0 = 0$ nicht als Startwert gewählt werden, aber für jeden anderen positiven, ganzzahligen Wert ist die Periodenlänge gegeben durch: $2^{31} - 2 \approx 2 \cdot 10^9$. Mit der nachfolgenden Prozedur `Seed(seed: integer)` kann für obige Rekursion ein beliebiger, positiver integer–Startwert gewählt werden.

1.2 Realisierungen

In den folgenden Abschnitten findet man alle Routinen zur Berechnung von reellen und komplexen Zufalls–Zahlen und –Intervallen, sowie von reellen und komplexen Zufalls–Matrizen/Vektoren und –Intervall-Matrizen/Vektoren. Die Einrahmungen kennzeichnen die zur Vefügung stehenden Prozeduren und Funktionen, wobei ein geklammerter Name (xxxx) dasjenige Modul kennzeichnet, welches die jeweilige Routine exportiert. Alle diese Routinen werden für die beiden Basistypen *real* und *Longreal* bereitgestellt.

1.2.1 Reelle Zufallszahlen

procedure `Seed` (**seed** : `integer`); (`rand0`)

liefert mit der beliebigen Eingangsgröße `seed` einen von Null verschiedenen, positiven integer–Startwert für die Rekursion (1.2). Für jede der nachfolgenden Zufalls–Funktionen kann durch den vorherigen Aufruf von `Seed` ein beliebiger Startwert gewählt werden, der sich dann vom Standardstartwert aus dem Modul `rand0` i.a. unterscheiden wird.

function `Rand` : `real`; (`rand0`)

function `LRand` : `Longreal`; (`rand0`)

liefern nach (1.2) je eine Zufallszahl aus dem Intervall $(0, 1)$, wobei der von Null verschiedene Startwert mit Hilfe der Prozedur `Seed(seed: integer)` nach Bedarf vorgegeben werden kann. Ohne Aufruf von `Seed` wird im Modul `rand0` ein Standard–Startwert benutzt.

function `Rand` (**a,b** : `real`) : `real`; (`random`)

function `Rand` (**a,b** : `Longreal`) : `Longreal`; (`random`)

liefern je eine Zufallszahl aus dem offenen Intervall (a, b).

function `RandLn` (**x** : `real`) : `real`; (`random`)

function `RandLn` (**x** : `Longreal`) : `Longreal`; (`random`)

liefern je eine Zufallszahl, die logarithmisch über $(1, exp(x))$ verteilt ist. Es gilt: `RandLn := exp(x*Rand);`

function `RandLn` (**a,b** : `real`) : `real`; (`random`)

function `RandLn` (**a,b** : `Longreal`) : `Longreal`; (`random`)

liefern je eine Zufallszahl, die logarithmisch über $(a, b) \not\ni 0$ verteilt ist. Es gilt: `RandLn := a*RandLn( ln(b/a) );`

1.2.2 Reelle Zufallsintervalle

function `Rand` (**x** : `interval`) : `interval`; (`irandom`)

function `Rand` (**x** : `Linterval`) : `Linterval`; (`irandom`)

liefern je ein Zufalls–Teilintervall, das im Innern von `x` liegt.

function `RandLn` (**x** : `interval`) : `interval`; (`irandom`)

function `RandLn` (**x** : `Linterval`) : `Linterval`; (`irandom`)

liefern je ein Zufalls–Teilintervall, das im Innern von `x` liegt, wobei die Teilintervallgrenzen logarithmisch über `x` verteilt sind.

1.2.3 Komplexe Zufallszahlen

function `Rand` (**u,v** : `complex`) : `complex`; (`crandom`)

function `Rand` (**x** : `Lcomplex`) : `Lcomplex`; (`crandom`)

liefern je ein komplexe Zufallszahl z mit den Eigenschaften:
$\Re(u) < \Re(z) < \Re(v)$ und $\Im(u) < \Im(z) < \Im(v)$.

1.2.4 Komplexe Zufallsintervalle

function `Rand` (**z** : `cinterval`) : `cinterval`; (`cirandom`)

function `Rand` (**x** : `Lcinterval`) : `Lcinterval`; (`cirandom`)

liefern je ein komplexe Zufalls–Teilintervall, das im Innern von z liegt.

1.2.5 Reelle Zufallsmatrizen/Vektoren

procedure `Rand_V` (**var** v : `rvector`; (mvran) `x,y : real`);

procedure `Rand_V` (**var** v : `Lrvector`; (mvran) `x,y : Longreal`);

liefern jeweils einen Zufallsvektor mit Elementen aus (x, y).

procedure `Rand_M` (**var** A : `rmatrix`; (mvran) `x,y : real`);

procedure `Rand_M` (**var** A : `Lrmatrix`; (mvran) `x,y : Longreal`);

liefern jeweils eine Zufallsmatrix mit Elementen aus (x, y).

1.2.6 Reelle Zufalls-Intervall-Matrizen/Vektoren

procedure `Rand_V` (**var** v : `ivector`;
(imvran) `x,y,eps : real`);

procedure `Rand_M` (**var** v : `imatrix`;
(imvran) `x,y,eps : real`);

procedure `Rand_V` (**var** v : `Livector`;
(limvran) `x,y : Longreal; eps : real`);

procedure `Rand_M` (**var** v : `Limatrix`;
(limvran) `x,y : Longreal; eps : real`);

liefern jeweils Punktintervall–Matrix/Vektor mit Zufallselementen aus (x, y), die dann bzgl. `eps` aufgebläht werden.

1.2.7 Komplexe Zufallsmatrizen/Vektoren

procedure `Rand_V` (**var** v : `cvector`;
(mvcran) `u,v : complex`);

procedure `Rand_M` (**var** A : `cmatrix`;
(mvcran) `u,v : complex`);

procedure `Rand_V` (**var** v : `Lcvector`;
(mvcran) `u,v : Lcomplex`);

procedure `Rand_M` (**var** A : `Lcmatrix`;
(mvcran) `u,v : Lcomplex`);

liefern Matrix/Vektor mit komplexen Zufallselementen $z[i,j]$, mit den Eigenschaften: $\Re(u) < \Re(z[i,j]) < \Re(v)$ und $\Im(u) < \Im(z[i,j]) < \Im(v)$.

1.2.8 Komplexe Zufalls-Intervall-Matrizen/Vektoren

```
procedure Rand_V ( var v : civector;
(imvcran)          u,v : complex; eps : real );
```

```
procedure Rand_M ( var A : cimatrix;
(imvcran)          u,v : complex; eps : real );
```

```
procedure Rand_V ( var v : Lcivector;
(limvcran)         u,v : Lcomplex; eps : real );
```

```
procedure Rand_M ( var A : Lcimatrix;
(limvcran)         u,v : Lcomplex; eps : real );
```

liefern jeweils Punktintervall–Matrix/Vektor mit komplexen Punktintervall–Elementen, die folgende Eigenschaften haben:

$$\Re(u) < \Re(z[i,j]) < \Re(v) \quad \text{und} \quad \Im(u) < \Im(z[i,j]) < \Im(v).$$

Anschließend werden die komplexen Punktintervalle $z[i,j]$ bzgl. `eps` aufgebläht.

Hinweise:

1. Bei den Zufallsintervall–Matrizen/Vektoren interessieren in der Praxis i.a. nur sehr schmale Intervalle. Daher werden zunächst nur Zufalls–Punktintervalle berechnet, die dann anschließend mit der *real*–Größe `eps` leicht aufgeblählt werden.

2. In den obigen Prozeduren liefern die genannten Aufblähungen auch für die *real*–Größe `eps=0` keine Punktintervalle.

1.3 Programmbeispiel

Das folgende Programm `rantst` erzeugt Zufallszahlen in speziellen Bereichen und mit verschiedenen Startwerten der Rekursion (1.2). Zusätzlich werden berechnet Teilintervalle mit Zufallsintervallgrenzen, komplexe Zufallszahlen aus vorgegebenen Bereichen für Real- und Imaginärteil und ein Intervallvektor mit Zufallskomponenten aus einem gegebenen Intervall.

```
program rantst(input,output);
use type_def, random, crandom, c_ari, i_ari,
    irandom, imvran, mvi_ari;
var k,kmax   : integer;
   u,v       : complex;
   t,Randint : interval;
   x,y,eps   : real;
   iv        : ivector[1..10];
begin { rantst }
   kmax := 10;
   writeln('Gleichmaessig verteilte Zufallszahlen');
   writeln('in (-41,20) mit Standard-Startwert:');
   for k := 1 to kmax do writeln(Rand(-41,20):10:3);
   Seed(135);
   writeln('Gleichmaessig verteilte Zufallszahlen in');
   writeln('(-41,20) mit Startwert:  Seed(135):');
   for k := 1 to kmax do writeln( Rand(-41,20):10:3 );

   writeln;
   writeln('Gleichm. verteilte komplexe Zufallszahlen:');
   u := compl(-1,11);  v := compl(2,12);
   writeln('u = ',u);
   writeln('v = ',v);
   for k := 1 to kmax do writeln( Rand(u,v) );

   writeln;
   writeln('Logarithm. verteilte Zufallszahlen in (1,99999):');
   for k := 1 to kmax do writeln(RandLn(1,99999):10:3);

   writeln;  Seed(4443);
   t := intval(-800,900); { Berechg. v. Teilintervallen von x }
   write('Teilintervalle von [');
```

```
   writeln(t.inf:10:3, ', ',t.sup:10:3, ']:');
   For k := 1 to kmax do
   begin
      Randint := Rand(t);
      writeln('[',inf(Randint):10:3,', ',
                  sup(Randint):10:3, ']');
   end;

   writeln;
   x := 80;   y := 90;  eps := 0;
   { eps=0 --> keine Punktintervalle }
   writeln('Bzgl. eps=0 leicht aufgeblaehter Punktintervall-');
   writeln('Vektor mit Zufalls-Punktintervallen aus (80,90):');
   Rand_V(iv,x,y,eps);  writeln(iv);
end.
```

Es folgt die Ausgabe des Programms `Rantst`:

```
Gleichmaessig verteilte Zufallszahlen
in (-41,20) mit Standard-Startwert:
-27.676
 17.335
  9.600
 -6.737
-15.666
-36.967
-25.228
-34.293
-38.326
 -2.328

Gleichmaessig verteilte Zufallszahlen in
(-41,20) mit Startwert:  Seed(135):
-40.936
  5.214
-40.590
 14.981
 15.338
 -6.871
-19.588
 -1.549
```

```
  2.006
-29.804

Gleichm. verteilte komplexe Zufallszahlen:
u = ( -1.000000000000E+000,  1.100000000000E+001 )
v = (  2.000000000000E+000,  1.200000000000E+001 )
(  1.318341383393E+000,  1.112121022917E+001 )
( -4.590348244918E-001,  1.166723492121E+001 )
( -3.480378069673E-001,  1.150952610025E+001 )
(  8.155005512830E-001,  1.103925513850E+001 )
(  1.283338555732E+000,  1.102370206501E+001 )
(  8.181965541200E-002,  1.171431616867E+001 )
(  5.355403570160E-001,  1.160892678826E+001 )
( -3.024093440278E-001,  1.113538497413E+001 )
(  2.457807568120E-001,  1.127905991035E+001 )
( -5.202604753525E-001,  1.166073025002E+001 )

Logarithm. verteilte Zufallszahlen in (1,99999):
29279.031
30456.002
  143.143
  109.281
   71.659
   36.149
  107.407
    3.776
 8917.268
   40.142

Teilintervalle von  [ -800.000,  900.000]:
[ -740.887,  -81.932 ]
[ -537.441,  530.311 ]
[ -655.779,  620.630 ]
[ -767.309, -466.907 ]
[ -606.317,  628.912 ]
[  503.800,  729.541 ]
[  226.608,  864.356 ]
[  -21.470,  101.076 ]
[ -454.830,  758.413 ]
[   76.752,  866.104 ]
```

```
Bzgl. eps=0 leicht aufgeblaehter Punktintervall-
Vektor mit Zufalls-Punktintervallen aus (80.90):
[  8.890184516035E+001,  8.890184516037E+001 ]
[  8.331161017683E+001,  8.331161017685E+001 ]
[  8.823224212891E+001,  8.823224212893E+001 ]
[  8.929346075713E+001,  8.929346075715E+001 ]
[  8.519494529589E+001,  8.519494529591E+001 ]
[  8.144558815352E+001,  8.144558815354E+001 ]
[  8.600009635835E+001,  8.600009635837E+001 ]
[  8.361949493811E+001,  8.361949493813E+001 ]
[  8.285142488909E+001,  8.285142488911E+001 ]
[  8.389811117383E+001,  8.389811117385E+001 ]
```

Kapitel 2

Integer-Langzahlarithmetik

In **PASCAL-XSC** stehen nur vorzeichenbehaftete *integer*-Zahlen in einer Länge von 32 Bit zur Verfügung. Für n : *integer* gilt damit:

$$-2147483648 \leq n \leq +2147483647 = 2^{31} - 1 \tag{2.1}$$

Wegen des begrenzten Zahlenbereichs können die Operatoren $+, -, *$ einen Überlauf erzeugen, so daß bei besonderen Problemstellungen eine integer-Langzahlarithmetik erforderlich wird. Bekanntlich läßt sich jede nicht-negative ganze Zahl p in einem Zahlensystem mit der Basis $B > 1$ eindeutig darstellen in der Form:

$$p = a_{m-1}B^{m-1} + a_{m-2}B^{m-2} + \ldots + a_1B^1 + a_0B^0, \qquad m \geq 1$$

Für die m benötigten Ziffern a_j gilt dabei:

$$0 \leq a_j \leq B - 1, \qquad j = 0, 1, 2, \ldots, m - 1;$$

Bei einer integer-Langzahlarithmetik speichert man die Ziffern a_j einer vorzeichenbehafteten *integer*-Langzahl z in einem *integer*–Feld, in dem noch zusätzlich das Vorzeichen und die Anzahl m der benötigten Ziffern festzuhalten ist. Als Basis wählen wir $B = 2^{15} = 32768$ und erreichen damit, daß die Multiplikation der a_j in (2.1) ohne Überlauf erfolgen kann.

2.1 Datenstruktur

Eine *integer*-Langzahl z wird als Variable vom Typ

```
type mp = dynamic array[*] of integer;        (type_def)
```

gespeichert. Das Modul `type_def` exportiert auch die integer–Variable

var maxprec : integer;

deren Wert die Länge des arrays festlegt, wenn z.B. die Variable `u` wie folgt vereinbart wird:

```
var u : mp[0..maxprec];
```

Festlegungen:

1. $\mathrm{Sign}(z) := \mathrm{Sign}(\ u[\,0\,]\ )$
2. Benötigte Ziffernzahl $m := |\,u[\,0\,]\,|$
3. $z = \mathrm{Sign}(u[0]) \cdot \sum_{k=0}^{m-1} u[m-k] \cdot B^k$

Beispiel:

Mit `maxprec=4`, $B = 32768$ gilt für $z = -9999999999$ die Darstellung:

$9999999999 = 9 \cdot B^2 + 10263 \cdot B^1 + 25599 \cdot B^0, \quad \rightsquigarrow \quad m = 3$ und damit:

$$u[0] = -3, \quad u[1] = 9, \quad u[2] = 10263, \quad u[3] = 25599, \quad u[4] = 0;$$

Ist die Anzahl m der benötigten Ziffern größer als `maxprec`, so entsteht ein entsprechender Laufzeitfehler. Die nicht benötigten Komponenten (hier $u[4]$) haben keine Bedeutung. Vergleichen Sie bitte das Programm von Seite 55.

Bei den nachfolgende Routinen werden alle Ausgangsgrößen von Typ `mp` nach obigem Schema gespeichert, wenn dies auch für die entsprechenden Eingangsgrößen gilt. Insbesondere die mit `read(u: mp)` eingelesenen *integer*-Langzahlen werden in u genau nach dem oben beschriebenen Muster gespeichert. Ganz entsprechend erwartet auch `write(u: mp)` die Variable u im gleichen Format. Wenn daher alle Zahlenwerte mit `read(u: mp)` eingelesen werden, laufen automatisch alle Routinen korrekt, d.h. der Anwender braucht sich um den oben beschriebenen Speichermechanismus überhaupt nicht zu kümmern.

Es kommen noch die folgende Datentypen zur Anwendung:

type `intvector` = **dynamic array**[*] **of** `integer`; (`type_def`)

type `mpvector` = **dynamic array**[*] **of** `mp`; (`type_def`)

2.2 Präzisionsänderung

Eine Präzisionsänderung, d.h. eine Änderung der Länge des array–Feldes kann mit Hilfe der Prozedur `set_maxprec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_maxprec` (**prec** : `integer`); (`type_def`)

Mit Hilfe der Eingangsgröße `prec` wird durch den Prozeduraufruf

```
set_maxprec( prec )
```

die Variable `maxprec` entsprechend geändert. Der Prozeduraufruf muß **vor** der Vereinbarung der entsprechenden `mp` Variablen erfolgen! Vergleichen Sie dazu bitte das Programm `lint_1` von Seite 55

Der aktuelle Wert der Variablen `maxprec` wird durch folgende Funktion geliefert:

function `get_maxprec` : `integer`; (`type_def`)

2.3 Operatoren

2.3.1 Vergleichsoperatoren

Die Operatoren $<, <=, >, >=, =, <>$ stehen als Vergleichsoperatoren für `mp`–Operanden zur Verfügung.

2.3.2 Arithmetische Operatoren

Neben den monadischen Operatoren $+, -$ stehen die drei arithmetischen Operatoren $+, -, *$ für `mp`–Operanden zur Verfügung und liefern alle Ergebnisse vom Typ `mp`.

Hinweis: Erfolgt bei den arithmetischen Operatoren ein Überlauf, so muß der Wert von `maxprec` entsprechend vergrößert werden.

Die Operatoren `div` und `mod` haben mit `mp`–Operanden genau die gleiche Bedeutung wie für *integer*–Operanden:

operator `div` (**var** u,v : `mp`) res : `mp[0..maxprec]` (`lo_int`) $\mathtt{u} = \mathtt{v} \cdot \mathtt{res} + \mathtt{r}, \qquad 0 \le \mathtt{r} < \lvert\mathtt{v}\rvert$

operator `mod` (**var** u,v : `mp`) res : `mp[0..maxprec]` (`lo_int`)

$$\mathtt{u} = \mathtt{v} \cdot \mathtt{n} + \mathtt{res}, \qquad 0 \le \mathtt{res} < |\mathtt{v}|, \quad \mathtt{n} \in \mathbb{Z}$$

2.3.3 Zuweisungsoperatoren

Durch Überladungen des Zuweisungsoperators stehen zusätzlich noch die folgenden Operatoren zur Verfügung:

`mp`–Variable := *integer/real/Longreal*–Ausdruck;
integer/real/Longreal–Variable := `mp`–Ausdruck;

Können die Wertzuweisungen nicht exakt durchgeführt werden, erfolgt eine entsprechende Fehlermeldung.

2.4 Multiplikation

Neben dem Operator $*$ mit zwei `mp`–Operanden existiert noch die Prozedur `mpmulint`, mit der ein `mp`–Wert mit einer *integer*–Zahl multipliziert wird:

procedure `mpmulint` (**var** u : `mp`; v : `integer`;
var w : `mp`); (`lo_int`)

$$\mathtt{w} = \mathtt{u} \cdot \mathtt{v}, \quad \text{falls} \quad -65535 \le \mathtt{v} \le +65535$$

2.5 Division

Neben dem Operator **div** mit zwei `mp`–Operanden existieren zwei weitere Prozeduren, mit denen auch der ganzzahlige Divisionsrest berechnet wird:

procedure `mpdivint` (**var** u : `mp`; v : `integer`; (`lo_int`)
var q : `mp`; **var** r : `integer`);

$$\mathtt{u} = \mathtt{v} \cdot \mathtt{q} + \mathtt{r}, \quad 0 \le \mathtt{r} < |\mathtt{v}|, \quad \text{falls} \quad -65535 \le \mathtt{v} \le +65535$$

procedure `mpdivint` (**var** u,v : `mp`; **var** q,r : `mp`); (`lo_int`)

$$\mathtt{u} = \mathtt{v} \cdot \mathtt{q} + \mathtt{r}, \quad 0 \le \mathtt{r} < |\mathtt{v}|$$

2.6 Standard–Anwendungen

function `abs` (**var** u : `mp`) : `mp[0..maxprec]`; (`lo_int`)
liefert $|\mathtt{u}|$

function `odd` (**var** u : `mp`) : `boolean`; (`lo_int`)
testet, ob **u** ungerade ist.

function `power` (**var** m : `mp`; ex : `integer`) : `mp[0..maxprec]`;
liefert $\mathtt{m}^{\mathtt{ex}}$, falls $\mathtt{ex} \geq 0$; (`lo_int`)

function `sqrt` (**var** u : `mp`) : `mp[0..maxprec]`; (`lo_int`)
liefert ganzzahligen Anteil von $\sqrt{\mathtt{u}}$, $\mathtt{u} \geq 0$

function `factorial` (n : `integer`) : `mp[0..maxprec]`; (`lo_int`)
liefert $n!$, falls $0 \leq n \leq +65535$

function `factorial` (n : `integer`) : `Longreal`; (`lo_int`)
liefert $n!$, falls $0 \leq n \leq +26$

function `factorial_sum_mp` (n : `integer`) : `mp[0..maxprec]`;
liefert $1! + 2! + \ldots, +n!$, falls $1 \leq \mathtt{n} \leq +65535$; (`lo_int`)

function `factorial_sum_mp` (n : `integer`) : `Longreal`;
liefert $1! + 2! + \ldots, +n!$, falls $1 \leq \mathtt{n} \leq +21$; (`lo_int`)

Die beiden folgenden Funktionen des Moduls `lo_int` liefern

function `binomial_mp` (n,k : `integer`) : `mp[0..maxprec]`;

function `binomial` (n,k : `integer`) : `Longreal`; (`lo_int`)

$$\binom{n}{k} := \frac{n!}{k!\,(n-k)!} = \frac{n(n-1)\cdot\dots\cdot(n-k+1)}{k!} = \binom{n}{n-k}, \quad 0 \le k \le n;$$

Die beiden folgenden Funktionen des Moduls `lo_int`

function `permutation_mp` (n,k : `integer`) : `mp[0..maxprec]`; $0 \le k \le n \le +65535$

function `permutation` (n,k : `integer`) : `Longreal`; (`lo_int`) $0 \le k \le n \le +26$

liefern: $$\frac{n!}{(n-k)!} = (n-k+1)\cdot(n-k+2)\,\cdots\,\cdot n, \quad \text{falls} \quad 0 \le k \le n;$$

function `ggT` (a,b : `mp`) : `mp[0..maxprec]`; (`lo_int`)

function `ggT` (a,b : `integer`) : `integer`; (`lo_int`)

liefern den größten gemeinsamen Teiler: `ggT( |a|,|b| )`;

function `ggT` (v : `intvector`) : `integer`; (`lo_int`)

function `ggT` (v : `mpvector`) : `mp[0..maxprec]`; (`lo_int`)

liefern: `ggT( |v[1]|,|v[2]|,...,|v[n]| )`;

function `kgV` (a,b : `mp`) : `mp[0..maxprec]`; (`lo_int`)

function `kgV` (a,b : `integer`) : `integer`; (`lo_int`)

liefern das kleinste gemeinsame Vielfache: `kgV( |a|,|b| );`

function `kgV` (v : `intvector`) : `integer`; (`lo_int`)

function `kgVmp` (v : `intvector`) : `mp[0..maxprec]`; (`lo_int`)

function `kgV` (v : `mpvector`) : `mp[0..maxprec]`; (`lo_int`)

liefern: `kgV( |v[1]|,|v[2]|,...,|v[n]| );`

Die beiden folgenden Prozeduren des Moduls `lo_int` liefern die Primzahlzerlegung von `n`:

procedure `prime_dec` (n : `integer`; **var** p,v : `intvector`);

procedure `prime_dec` (n : `mp`; **var** p : `mpvector`
var v : `intvector`); (`lo_int`)

`p` enthält die Primzahlen und `v` deren Vielfachheit der Primfaktorzerlegung.

procedure `pollard` (n,a,x1 : `mp`); (`lo_int`)

schreibt mit geeignete Startwerten `a,x1` nach dem Algorithmus von Pollard einige Faktoren von `n` auf den Bildschirm.

2.7 Ein-/Ausgabeprozeduren

Es stehen die gewohnten Prozeduren

procedure `read` (**var** F : `Text`; **var** a : `mp`); (`lo_int`)

procedure `write` (**var** F : `Text`; a : `mp`); (`lo_int`)

mit optionalem Fileparameter, beliebig vielen Ein-/Ausgabeparametern, jedoch ohne Formatspezifikationen zur Verfügung. Die Ein- bzw. Ausgabe von `mp`–Zahlen erfolgt dabei ganz analog zur Ein- bzw. Ausgabe von *integer*–Zahlen.

Zusätzlich stehen die Prozeduren

procedure `read` (**var** F : `Text`; **var** v: `intvector`); (`type_def`)

procedure `write` (**var** F : `Text`; **var** v: `intvector`); (`type_def`)

procedure `read` (**var** F : `Text`; **var** v : `mpvector`); (`lo_int`)

procedure `write` (**var** F : `Text`; **var** v : `mpvector`); (`lo_int`)

mit optionalem Fileparameter, beliebig vielen Ein-/Ausgabeparametern, jedoch ohne Formatspezifikationen zur Verfügung. Die Ein- bzw. Ausgabe eines Vektors erfolgt dabei komponentenweise entsprechend der Ein- bzw. Ausgabe von *integer*- oder `mp`–Zahlen.

2.8 Programmbeispiele

2.8.1 Beispiel zur Datenstruktur

Das nachfolgende Programm `lint_1` demonstriert die auf Seite 48 beschriebene Datenstruktur und zeigt, wie man mit Hilfe einer Zeichenkette `s` eine große Zahl zunächst in eine Textdatei schreibt, um sie danach wieder aus dieser Textdatei zu lesen, ohne im Wiederholungsfall die vielen Ziffern immer wieder über die Tastatur neu eingeben zu müssen.

```
program lint_1(input,output);
use type_def, lo_int;
var p : integer;

procedure main;
var t : Text;
   u  : mp[0..maxprec];
   s  : string;
   k  : integer;
begin
   s := '-9999999999';
   rewrite(t);
   write(t,s);
   reset(t);
   read(t,u);
   For k := 0 to maxprec do
   writeln('u[',k:1,'] = ',u[k]);
end; { main }

begin
   writeln('maxprec = ',get_maxprec);
   writeln('maxprec = ? ');  read(p);
   set_maxprec(p);
   writeln('Neuer Wert = ',get_maxprec);
   main;
end.
```

Mit der Wahl `maxprec=4` liefert das obige Programm die folgende Ausgabe:

```
maxprec = 6
maxprec = ?
4    <enter>
```

```
Neuer Wert = 4
n[0] = -3
n[1] = 9
n[2] = 10263
n[3] = 25599
n[4] = 0
```

2.8.2 $\sum\limits_{k=0}^{n} 2^k \binom{n}{k} = 3^n$

Mit Hilfe des nachfolgenden Programms `lint_2` kann die Beziehung

$$\sum_{k=0}^{n} 2^k \binom{n}{k} = 3^n, \qquad n = 0, 1, 2, \ldots$$

für spezielle n bestätigt werden.

```
program lint_2(input,output);
use type_def, lo_int;
var p : integer;

procedure main;
var u,two,three : mp[0..maxprec];
    n,k         : integer;
begin
   writeln('n = ?');  read(n);
   u := binomial(n,0);  two := 2;  three := 3;
   For k := 1 to n do
       u := u + power(two,k) * binomial_mp(n,k);
   if u = power(three,n) then
       writeln('Beweis fuer n = ',n:4,' gefuehrt.');
end; { main }

begin
   repeat
      writeln('maxprec = ',get_maxprec);
      writeln('maxprec = ? ');  read(p);
      set_maxprec(p);
      writeln('Neuer Wert = ',get_maxprec);
      main;
   until false;
end.
```

Für `maxprec` $= 213$ und $n = 2000$ erhält man die Bildschirmausgabe:

```
Beweis fuer n = 2000 gefuehrt.
```

Bitte beachten Sie, daß zu kleine `maxprec`–Werte wegen Überlauf bei den internen Rechnungen eine Fehlermeldung erzeugen.

2.8.3 Primfaktorzerlegung

Mit Hilfe des nachfolgenden Programms `lint_3` wird zu einer vorgegebenen Zahl n ihre Primfaktorzerlegung berechnet. Besonders aufwendig ist die Primfaktorzerlegung solcher Zahlen n, die als Produkt zweier großer Primzahlen definiert sind. Das Programm `lint_3` verlangt die Eingabe zweier Primzahlen, berechnet deren Produkt n und versucht danach die Primfaktorzerlegung von n mit Hilfe der Prozedur `prime_dec(n : mp; ...)` neu zu bestimmen. Größere Testprimzahlen variabler Länge erhält man z.B. mit den Primzahlen

$$1\,979\,339\,333 \quad \text{und} \quad 1\,979\,339\,339$$

wenn man von rechts nacheinander deren Ziffern streicht.

```
program lint_3(input,output);
{ Primfaktor-Zerlegung einer Testzahl n = p1*p2 }
{ p1,p2 sind Primzahlen und definieren n,       }
{ p: Enthaelt alle Primzahlen von n,            }
{ v: Enthaelt die Vielfachheit der Primzahlen.  }
use type_def, lo_int;
var p : integer;

procedure main;
var n,p1,p2 : mp[0..maxprec];
    p       : mpvector[1..2,0..maxprec];
    v       : intvector[1..2];

begin
   writeln('p1 = ?');  read(p1);
   writeln('p2 = ?');  read(p2);
   n := p1 * p2;
   prime_dec(n,p,v);   { Berechnet p,v }
   writeln; writeln('Ausgabe der Primzahlen:');
   writeln(p);
```

```
   writeln; write('Ausgabe der Vielfachheit:');
   writeln(v);
end; { main }

begin
   repeat
      writeln('maxprec = ',get_maxprec);
      writeln('maxprec = ? ');  read(p);
      set_maxprec(p);
      writeln('Neuer Wert = ',get_maxprec);
      main;
   until false;
end.
```

Bei Eingabe der Primzahlen `p1` = 197933933, `p2` = 1979339339 wird intern zunächst das Produkt `n` = `p1`·`p2` berechnet. `prime_dec(n,p,v)` liefert dann die Primfaktorzerlegung von `n`, und es ergibt sich nach etwa 40 Minuten die folgende Bildschirmausgabe:

```
maxprec = 6
maxprec = ?
30         <enter>
Neuer Wert = 30
p1 = ?
197933933  <enter>
p2 = ?
1979339339 <enter>

Ausgabe der Primzahlen:
197933933
1979339339

Ausgabe der Vielfachheit:
1
1
```

2.8.4 Pollard's Faktorzerlegung

Im vorhergehenden Programm `lint_3` wurde die Primfaktorzerlegung einer Zahl `n` mit Hilfe der Prozedur `prime_dec` berechnet. Bei größeren Zahlen ergeben sich jedoch zu lange Laufzeiten, da obige Prozedur alle Primzahlen systematisch durchprobiert.

Bei größeren Zahlen `n` empfiehlt sich das folgende Programm `lint_4`, mit dem nach Pollard's Algorithmus in vergleichsweise sehr viel kürzerer Laufzeit einige Faktoren von `n` auf den Bildschirm geschrieben werden, wobei diese Faktoren jedoch keine Primzahlen sein müssen.

```
program lint_4(input,output);
{ Faktor-Zerlegung einer Testzahl n nach dem }
{ Algorithmus von Pollard.                   }
{ a,x1:   Startwerte zu Pollards Algorithmus }
use type_def, lo_int;
var p : integer;

procedure main;
var n,a,x1 : mp[0..maxprec];

begin
   writeln('Zu zerlegende Zahl n  = ?');  read(n);
   writeln('Startwert a  = ?');  read(a);
   writeln('Startwert x1 = ?');  read(x1);
   pollard(n,a,x1);   { Pollards Algorithmus schreibt }
end;             { Faktoren von n auf den Bildschirm. }

begin
   repeat
      writeln('maxprec = ',get_maxprec);
      writeln('maxprec = ? ');  read(p);
      set_maxprec(p);
      writeln('Neuer Wert = ',get_maxprec);
      main;
   until false;
end.
```

Bei erfolgreicher Arbeit der Prozedur `pollard` kann `n` dann durch das Produkt der gefundenen Faktor dividiert werden. Auf den kleineren Quotienten läßt sich dann das gleiche Verfahren wiederholt anwenden, bis schließlich alle

Primfaktoren von `n` bekannt sind.

Als Anwendung betrachten wir die Zahl $\mathtt{n} = 391778420109890287$, die das Produkt der beiden Primzahlen `p1,p2` von Seite 58 ist. Mit diesem Wert `n` liefert das Programm `lint_4` in nur wenigen Sekunden die folgende Bildschirmausgabe:

```
maxprec = 6
maxprec = ?
8         <enter>
Neuer Wert = 8
Zu zerlegende Zahl n = ?
391778420109890287  <enter>
Startwert a = ?
3                   <enter>
Startwert x1 = ?
5                   <enter>
Pollard's methode - finding factors of
391778420109890287
For i = 8040    factor = 197933933
Not all factors found in 10000 cycles.
```

Da jetzt ein Faktor `p1=197933933` von `n` bekannt ist, findet man einen zweiten Faktor `p2` durch einfache Division. Mit Hilfe der Prozedur `prime_dec` bestätigt man dann leicht, daß `p1,p2` Primzahlen sind und erhält:

$$\mathtt{n} = \mathtt{p1} \cdot \mathtt{p2}$$

Kapitel 3

Staggered Intervall-Arithmetik

3.1 Einführung

Bei einem schlechtkonditionierten Problem erhält man oftmals nur noch dann vernünftige numerische Ergebnisse, wenn man bestimmte Teile der Rechnung in höherer Präzision durchführt, d.h. wenn man z.B. mit einem doppelt oder dreifach langen Datenformat arbeitet. Um auch dabei garantierte Einschließungen der exakten Ergebnisse gewährleisten zu können, müssen die Operanden der vier Grundoperationen in Intervalle höherer Präzision eingeschlossen werden, und die Grundoperationen sind dann als Intervalloperationen durchzuführen. Es ist bekannt, daß solche Intervallrechnungen unvermeidbare Aufblähungen der Ergebnisintervalle zur Folge haben, die um so stärker ausfallen, je breiter die Operandenintervalle vorgegeben sind. Um diese unerwünschten Überschätzungen nach Möglichkeit zu vermeiden, muß man die Durchmesser der Operandenintervalle hinreichend klein wählen. Bei einer Präzision von 26 Dezimalziffern wird beispielsweise der Wert π durch das folgende Intervall eingeschlossen

$$\pi \in [3.1415926535897932384626433, 3.1415926535897932384626434],$$

dessen relativer Durchmesser gegeben ist durch: $10^{-25}/\pi$. Dieser Durchmesser wird also um so kleiner, je mehr Dezimalziffern verwendet werden. Für die mathematische Darstellung eines solchen Intervalloperanden bietet sich dann z.B. mit dem Datentyp `real` die folgende Summe an:

$$x := x_1 + x_2 + \ \dots \ + x_p + [X.inf, X.sup] \tag{3.1}$$

Dabei sind die Summanden x_i und die Intervallgrenzen $X.inf \leq X.sup$ Größen vom Typ `real`, und der Index p ist der Präzisionsparameter des *staggered Intervall* Formats. Für die Einschließung von π gilt damit:

$$\begin{aligned} x_1 &:= 3.141592653589 \\ [X.inf, X.sup] &:= [7.932384626433 \cdot 10^{-13}, 7.932384626434 \cdot 10^{-13}] \end{aligned}$$

Für die Summanden rechts in (3.1) definieren wir in **PASCAL-XSC** bzgl. des Basistyps `real` den folgenden Datentyp, der vom Hilfs–Modul `type_def` exportiert wird:

```
type Stagg = dynamic array[*] of real;
```

Das gleiche Modul `type_def` exportiert auch die integer–Variable

```
var Stagg_prec : integer;
```

deren Wert den Präzisionsparameter festlegt, wenn z.B. die Variable `x` wie folgt vereinbart wird:

```
var x : Stagg[-1..Stagg_prec];
```

Wenn dann mit den Bezeichnungen aus (3.1) die folgenden Beziehungen

`x[-1]` $= X.inf$; `x[0]` $= X.sup$; `x[i]` $= x_i$, $1 \le i \le$ `Stagg_prec`

erfüllt sind, werden die *staggered* Arithmetik–Routinen mit allen Variablen, die entsprechend `x` vereinbart sind, stets erfolgreich arbeiten.

Nach der Definition des Datentyps `Stagg` muß bei der Vereinbarung einer Variablen dieses Typs der in (3.1) angegebene Index p natürlich nicht mit dem durch die Variable `Stagg_prec` vorgegebenen Wert übereinstimmen. Die vom Compiler her zulässige Vereinbarung

`var a : Stagg[-1..p],` `p` $=$ `ub(a)` $\neq$ `Stagg_prec`

liefert jedoch mit vielen Routinen des Moduls `stagg` Fehlermeldungen, da sich die Indexbereiche der Variablen `x`,`a` unterscheiden. Nur in wenigen Ausnahmefällen wird auch die Variable `a` erfolgreich eingesetzt, wenn mit ihr z.B. die Operanden der vier Grundoperationen $\{+, -, *, /\}$ definiert werden. Im Gegensatz dazu muß jedoch eine Variable, welche das Ergebnis einer solchen Grundoperation aufnimmt, wie die obige Variable `x` nur mit Hilfe von `Stagg_prec` vereinbart werden. Das folgende Programmbeispiel liefert dem Anwender weitere Informationen zur Handhabung der *staggered* Arithmetik:

```
program stagg1(input,output);
use type_def, { exportiert: type Stagg; var Stagg_prec }
    stagg;    { exportiert: operator :=, *;  write(x)  }

  procedure main;
  var a : Stagg[-1..2];
      b : Stagg[-1..1];
      x : Stagg[-1..Stagg_prec];
  begin
     a[-1] := 0;  a[0] := 0;  a[1] := 1;  a[2] := +1e-70;
     b[-1] := -1e-70;   b[0] := -1e-70;   b[1] := 1;
     writeln('Punktintervall:  a = [1+1e-70,1+1e-70]');
     writeln('Punktintervall:  b = [1-1e-70,1-1e-70]');
     x := a * b;
     writeln('a * b =');
     writeln(x);
     writeln('(a * b) / a =');
     writeln(x/a);
  end; { main }

BEGIN
   writeln('Von type_def definierter Stagg_prec Wert:');
   writeln('Stagg_prec = ',Stagg_prec);
   Stagg_prec := 3;
   writeln('Neuer Wert: Stagg_prec = ',Stagg_prec);
   main;
END.
```

Der Aufruf des **PASCAL-XSC** Programms `stagg1` liefert die folgende Bildschirmausgabe:

```
Von type_def definierter Stagg_prec Wert:
Stagg_prec = 1
Neuer Wert:  Stagg_prec = 3
Punktintervall:  a = [1+1e-70, 1+1e-70]
Punktintervall:  b = [1-1e-70, 1-1e-70]
a * b =
   1.000000000000E+000
  -1.000000000000E-140
   0.000000000000E+000
 [-0.000000000000E+000, 0.000000000000E+000]
(a * b) / a =
   1.000000000000E+000
  -1.000000000000E-070
   0.000000000000E+000
 [-0.000000000000E+000, 0.000000000000E+000]
```

Anmerkungen:

1. Die *staggered* Arithmetik liefert mit `(a*b)/a` für das exakte Ergebnis $b = 1 - 10^{-70}$ eine maximalgenaue Einschließung.

2. Der obige Zahlenwert $1 - 10^{-70}$, der in einer Gleitkommadarstellung 69 Dezimalziffern erfordern würde, läßt sich mit der Variablen `b` sehr effektiv speichern, da hierzu im *staggered Intervall* Format nur drei real–Variablen benötigt werden.

3. Bitte beachten Sie, daß im Programm `stagg1` die Präzision von `x` nur **vor** dem Aufruf der Prozedur `main` definiert werden kann. Wie man auch innerhalb einer Prozedur die Präzision einer staggered Variablen ändern kann, zeigt das Programm `stagg3` von Seite 85.

Alle bisherigen Erklärungen zum `staggered Intervall` Format bezogen sich auf den Basistyp `real` mit 13 Dezimalziffern. Für den zweiten Basistyp `Longreal` mit seinen 21 dezimalen Mantissen–Ziffern gelten die Definitionen der Datentypen und die Vereinbarungen der Variablen ganz entsprechend.

Da alle Rechnungen im *staggered* Format mit Hilfe des langen Akkus erfolgen und weil die Longreal–Multiplikationen in diesem Akkumulator nicht uneingeschränkt durchführbar sind, erhält man mit dem Basistyp `Longreal` in Ausnahmefällen Fehlermeldungen, wenn die Exponenten auftretender Longrealprodukte betragsmäßig zu groß werden. In solchen Fällen müssen dann die Rechnungen im *staggered* Format zum Basistyp `real` durchgeführt werden, da real–Produkte den Akku nicht zum Überlauf bringen.

3.2 Reelle staggered Arithmetik

In diesem Abschnitt werden die für das Rechnen mit reellen Zahlen notwendigen Operatoren, Funktionen und Prozeduren im *staggered* Format bzgl. der Basistypen `real` und `Longreal` in jeweils getrennten Unterabschnitten bereitgestellt. Ein geklammerter Name (xxx) in einer der nachfolgenden Einrahmungen bezeichnet das Modul, welches die jeweilige Routine exportiert.

3.2.1 Der Basistyp `real`

Wir betrachten jetzt nur das *staggered* Format zum Basistyp `real`. Bitte beachten Sie auch die Einführung auf Seite 61, die sich genau auf diesen Datentyp bezieht. Das dortige Programm `stagg1` liefert bereits wichtige Informationen für die ersten Programmierschritte. Der notwendige Datentyp sowie die erforderlichen Operatoren, Funktionen und Prozeduren werden von den Hilfs–Modulen `type_def` und `stagg` exportiert.

3.2.1.1 Datenstruktur

Datentyp

type Stagg = **dynamic array**[*] **of** real; (`type_def`)

Präzisionsparameter

var Stagg_prec : integer; (`type_def`)

Variablen-Vereinbarung

```
var x : Stagg[-1..Stagg_prec];
```

3.2.1.2 Präzisionsänderung

Eine Präzisionsänderung kann mit Hilfe der Prozedur `set_Stagg_prec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_Stagg_prec` (**prec** : `integer`); (`type_def`)

Mit Hilfe der Eingangsgröße `prec` wird durch den Prozeduraufruf `set_Stagg_prec(prec)`

die Variable `Stagg_prec` entsprechend geändert. Der Prozeduraufruf muß **vor** der Vereinbarung der entsprechenden *staggered* Variablen erfolgen! Bitte vergleichen Sie dazu auch die Anmerkung 3. von Seite 64.

Der aktuelle Wert der Variablen `Stagg_prec` wird durch die folgende Funktion geliefert:

function `get_Stagg_prec` : `integer`; (`type_def`)

Präzisionsanpassung

Mit der Funktion

function `change_length` (**var** a : `Stagg`) : (stagg) `Stagg[-1..Stagg_prec]`;

kann die Variable `a`, mit `ub(a)` $\neq$ `Stagg_prec` durch die Anweisung

`x := change_length(a);` $\leadsto$ `a` $\subseteq$ `x`

an die aktuelle, durch `Stagg_prec` gegebene Präzision der Variablen `x` angepaßt werden. Das Programm `stagg2` von Seite 83 enthält eine entsprechende Anwendung.

3.2.1.3 Hilfsfunktionen/Operatoren

function `convert_Stagg` (**var** lo,up : `dotprecision`) : (stagg) `Stagg[-1..Stagg_prec]`;

Mit der Anweisung

`x := convert_Stagg(lo,up);` $\leadsto$ [`lo`, `up`] $\subseteq$ `x`

wird das Eingangs–Intervall $[lo, up]$ durch die *staggered* Variable `x` maximalgenau eingeschlossen, wobei die Bedingung `ub(x)` = `Stagg_prec` erfüllt sein muß. Fast alle Hilfsfunktionen und Arithmetikoperatoren werden mit dieser Funktion `convert_Stagg` realisiert.

procedure `normal` (**var a** : `Stagg`); (`stagg`)

Wenn sich bei der Eingangsgröße `a` die `a[i]` für $i \geq 1$ überlappen, was z.B. nach einer Division eintreten kann, so liefert der Aufruf `normal(a)` mit der Ausgangsgröße `a` eine maximalgenaue Einschließung der Eingangsgröße, wobei sich die `a[i]` der Ausgangsgröße dann nicht mehr überlappen. Die Bedingung `ub(a)` = `Stagg_prec` muß erfüllt sein.

function `inf` (**var** a : `Stagg`) : (stagg) `Stagg[-1..Stagg_prec]`;

function `sup` (**var** a : `Stagg`) : (stagg) `Stagg[-1..Stagg_prec]`;

liefern Punkteinschließungen der Unter- bzw. Obergrenze des Intervalls `a`, wobei die Bedingung `ub(a)` = `Stagg_prec` erfüllt sein muß.

function `diam` (**var** a : `Stagg`) : real; (stagg)

liefert mit der Anweisung `r := diam(a);` den aufgerundeten Durchmesser des Intervalls `a` an die real–Variable `r`. `ub(a)` $\neq$ `Stagg_prec` ist erlaubt.

function `mid` (**var** a : `Stagg`) : `Stagg[-1..Stagg_prec]`; (stagg)

liefert mit der Anweisung `m := mid(a)` eine Punkt–Einschließung des gerundeten Mittelpunktes von `a`. Es muß gelten: `ub(a)` = `Stagg_prec`

function `hull` (**var** x,y : `Stagg`) : `Stagg[-1..Stagg_prec]`; (stagg)

liefert Einschließung von $[\inf(x), \sup(y)]$; `ub(x)` $\neq$ `ub(y)` $\neq$ `Stagg_prec` ist erlaubt.

procedure `add_Stagg_times_Stagg` (**var** a,b : `Stagg`;	
(stagg)	**var** d1,d2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter : a,b : `Stagg`; d1,d2 : `dotprecision`
Ausgangsparameter: d1,d2 : `dotprecision`

Bezeichnet man jetzt zur Unterscheidung die Ausgangsparameter d1,d2 mit D1,D2, so gilt nach dem Aufruf der Prozedur: $a \cdot b + [d1, d2] \subseteq [D1, D2]$, wobei folgende Bedingung erlaubt ist: $\texttt{ub}(a) \neq \texttt{ub}(b) \neq \texttt{Stagg\_prec}$.

procedure `add_real_times_Stagg` (r : `real`; **var** a : `Stagg`;	
(stagg)	**var** d1,d2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter : r : `real`; a : `Stagg`; d1,d2 : `dotprecision`
Ausgangsparameter: d1,d2 : `dotprecision`

Bezeichnet man jetzt zur Unterscheidung die Ausgangsparameter d1,d2 mit D1,D2, so gilt nach dem Aufruf der Prozedur: $r \cdot a + [d1, d2] \subseteq [D1, D2]$, wobei folgende Bedingung erlaubt ist: $\texttt{ub(a)} \neq \texttt{Stagg\_prec}$.

procedure `mant_exp` (**var** a,m : `Stagg`; **var** ex : `integer`);
(stagg)

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter : a : `Stagg`;
Ausgangsparameter: m : `Stagg`; ex : `integer`;

Der Aufruf `mant_exp(a,m,ex)` berechnet zu gegebenem, punktförmigem `a` die zugehörige Mantisse `m` und den Zehner–Exponenten `ex`, so daß im Falle $\texttt{a} \neq 0$ gilt: $\texttt{a} \subseteq \texttt{m} \cdot 10^{\texttt{ex}}$, $\texttt{m} \subseteq [0.1, 1)$, wobei der Durchmesser von `m` von Null verschieden sein kann. $\texttt{a} = 0$: $\texttt{m} = 0$ und $ex = -2147483647$.
Bedingungen: $ub(a) \stackrel{!}{=} ub(m)$, aber $ub(a) \neq \texttt{Stagg\_prec}$ ist zulässig. Ist `a` nicht punktförmig, so erfolgt eine entsprechende Fehlermeldung mit Programmabbruch.

operator `M_10HOCH` (x : `Stagg`; k : `integer`) :	
(stagg)	res : `Stagg[-1..ub(x)]`;

`y := x M_10HOCH k` liefert schnelle Einschließung von $x \cdot 10^k \subseteq$ `y`, falls $-267 \leq k \leq 255$. $ub(y) \stackrel{!}{=} ub(x)$, aber `ub(x)` $\neq$ `Stagg_prec` ist zulässig.

function `round` (**var** x : `Stagg`) : `integer`;	(stagg)

Null wird zu x addiert, um mögliche Überlappungen der x[j] zu beseitigen. Dann wird x[1] zur nächsten `integer`-Zahl gerundet. $ub(x) \stackrel{!}{=}$ `Stagg_prec`

procedure `mul1pmeps` (**var** a : `Stagg`; eps : `real`);	(stagg)

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter : a : `Stagg`; eps : `real`;
Ausgangsparameter: a : `Stagg`;

Der Aufruf `mul1pmeps(a,eps)` berechnet zu den gegebenen Eingangsgrößen `a` und `eps` > 0 mit der Ausgangsgröße `a` eine garantierte Einschließung des Intervallprodukts: $a \cdot [1 - eps, 1 + eps] \subseteq a$; $ub(a) \stackrel{!}{=}$ `Stagg_prec`

function `blow` (x : `Stagg`; eps : `real`) :	
(stagg)	`Stagg[-1..ub(x)]`;

`y := blow(x,eps)` liefert die übliche ε–Aufblähung von `x`. $ub(y) \stackrel{!}{=} ub(x)$

3.2.1.4 Transferfunktionen

function `ival` (**var** a : `Stagg`) : interval;	(stagg)

Die Anweisung `x := ival(a)` liefert für `a: Stagg` eine maximalgenaue Einschließung: `a`$\subseteq$ `x` : interval. `ub(a)` $\neq$ `Stagg_prec` ist zulässig.

function `Staggintval` (r : `real`) : `Stagg[-1..Stagg_prec]`; (stagg)

`a := Staggintval(r)` liefert Punkteinschließung für $\texttt{r} \in \texttt{a}$. Die Bedingung `ub(a)` = `Stagg_prec` muß erfüllt sein.

procedure `TRANS` (**var** a : `Stagg`; **var** r : `real`); (stagg)

Eingangsparameter : a : `Stagg`;
Ausgangsparameter: r : `real` ist maximalgenauer Mittelpunkt von `a`

Die Ungleichung `ub(a)` $\neq$ `Stagg_prec` ist zulässig.

procedure `TRANS` (**var** a : `Stagg`; **var** x,y : `real`); (stagg)

Eingangsparameter : a : `Stagg`;
Ausgangsparameter: x,y : `real`

$(x + y)$ ist maximalgenaue Näherung des Mittelpunkts von `a` im doppeltlangen Format. Die Ungleichung `ub(a)` $\neq$ `Stagg_prec` ist zulässig.

function `Staggintval` (**var** a : `interval`) : (stagg) `Stagg[-1..Stagg_prec]`;

`b := Staggintval(a);` $\rightsquigarrow$ Die Intervalle `a`,`b` haben zwar unterschiedliche Datenstrukturen, sind aber mathematisch identisch[1]. Für obige Anweisung muß die Bedingung `ub(b)` = `Stagg_prec` erfüllt sein.

function `Staggintval` (x,y : `real`) : (stagg) `Stagg[-1..Stagg_prec]`;

`b := Staggintval(x,y);` $\rightsquigarrow$ `b = x +* y`, (Punktintervall–Hülle).
Für die Anweisung muß die Bedingung `ub(b)` = `Stagg_prec` erfüllt sein.

[1]Ist `a` kein Punktintervall, so ist der Durchmesser von `b` für eine sinnvolle *staggered* Arithmetik i.a. zu groß!

3.2.1.5 Zuweisungsoperatoren

Zuweisung	Bedeutung	
`rv := S`	`rv`: gerundete Mitte v. `S`;	`ub(S)` $\neq$ `Stagg_prec` zulässig
`iv := S`	`S` $\subseteq$ `iv`;	`ub(S)` $\neq$ `Stagg_prec` zulässig
`Sv := r`	`Sv` = `r`;	`ub(Sv)` $\neq$ `Stagg_prec` zulässig
`Sv := i`	`Sv` = `i`;	`ub(Sv)` $\neq$ `Stagg_prec` zulässig
`Sv := S`	`Sv` = `S`;	`ub(Sv)` = `ub(S)`
`Sv := dot`	`dotprecision` $\in$ `Sv`;	`ub(Sv)` = `Stagg_prec`
`Sv := idot`	`idotprecision` $\subseteq$ `Sv`;	`ub(Sv)` = `Stagg_prec`

rv = `real`–Variable, iv = `interval`–Variable, Sv = `Stagg`–Variable,
S = `Stagg`–Ausdruck, i = `interval`–Ausdruck, r = `real`–Ausdruck,
(i)dot = `(i)dotprecision`–Ausdruck

3.2.1.6 Vergleichs- und Arithmetik-Operatoren

rechter Operand	integer real	Stagg	Bemerkungen
linker Operand			
monadisch		$+, -$	
integer real		**in**, $\diamond$ $=$, $<>$ $\bullet$	$\diamond \in \{+, -, *, /\}$ $\bullet \in \{<, \leq, >, \geq\}$
Stagg	$\diamond$ $=$, $<>$ $\bullet$	**in**, $\diamond$, $\bullet$ $=$, $<>$ $><$, $**$ **lies_in**	

Operatoren des Moduls `stagg`

Anmerkungen:

`Stagg` $<$ `real` bedeutet: `sup(Stagg)` $<$ `real`
`Stagg` $\leq$ `real` bedeutet: `sup(Stagg)` $\leq$ `real`
`Stagg` $>$ `real` bedeutet: `inf(Stagg)` $>$ `real`
`Stagg` $\geq$ `real` bedeutet: `inf(Stagg)` $\geq$ `real`
`real` $<$ `Stagg` bedeutet: `real` $<$ `inf(Stagg)`
`real` $\leq$ `Stagg` bedeutet: `real` $\leq$ `inf(Stagg)`
`real` $>$ `Stagg` bedeutet: `real` $>$ `sup(Stagg)`
`real` $\geq$ `Stagg` bedeutet: `real` $\geq$ `sup(Stagg)`
`Stagg` $=$ `real` bedeutet: `inf(Stagg)` $=$ `sup(Stagg)` $=$ `real`

Der Operator $><$ testet auf Disjunktheit zweier `Stagg`–Operanden.
Der Operator $**$ liefert den Durchschnitt zweier `Stagg`–Operanden, ein leerer Durchschnitt führt zu einem Laufzeitfehler.
Der Operator **in** zwischen zwei `Stagg` Operanden testet die Relation *liegt im Innern.*
Der Operator **in** zwischen einem `real` und einem `Stagg` Operanden testet die Relation *liegt in.*
Der Operator **lies_in** testet die Relation *liegt in* zwischen zwei `Stagg` Operanden, wobei die Präzision beider Operanden durch die Variable `Stagg_prec` gegeben sein muß. `x lies_in y` $\longleftrightarrow$ `x` $\subseteq$ `y`
Die Operatoren $<, \leq, >, \geq$ zwischen zwei punktförmigen `Stagg` Operanden haben die übliche Bedeutung, wobei die Präzision beider Operanden durch die Variable `Stagg_prec` gegeben sein muß. Ist einer der beiden Operanden nicht punktförmig, so erfolgt ein Laufzeitfehler.
Bei den arithmetischen Operatoren kann die Präzision beider Operanden unabhängig vom Wert der Variablen `Stagg_prec` gewählt werden. Die Präzision des Operator–Ergebnisses ist jedoch stets durch den Wert der Variablen `Stagg_prec` gegeben.
Bei den monadischen Operatoren muß die Präzision des rechten Operanden nicht durch den Variablenwert `Stagg_prec` bestimmt sein.

3.2.1.7 Mathematische Standardfunktionen

function `abs` (`var` x : `Stagg`) :   (stagg) `Stagg[-1..Stagg_prec];`

`y := abs(x);` $\leadsto$ `y` $:= \{|r| : r \in$ `x`$\}$
Die Bedingung `ub(y)` $=$ `ub(x)` $=$ `Stagg_prec` muß erfüllt sein.

function `sqrt` (`var` x : `Stagg`) :   (stagg) `Stagg[-1..Stagg_prec];`

Mit $\sqrt{\mathtt{x}} := \{r \mid r = \sqrt{t} \ \wedge \ t \in \mathtt{x}\}$ und der Anweisung `y := sqrt(x)` erhält man: $\sqrt{\mathtt{x}} \subseteq \mathtt{y}$; Die Bedingung `ub(y)` $=$ `Stagg_prec` muß erfüllt sein. `ub(x)` $\neq$ `Stagg_prec` ist zulässig.

function `sqrt3` (`var` x : `Stagg`) :   (s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\sqrt[3]{\mathtt{x}} := \{r \mid r = \sqrt[3]{t} \ \wedge \ t \in \mathtt{x}\}$ und der Anweisung `y := sqrt3(x)` erhält man: $\sqrt[3]{\mathtt{x}} \subseteq \mathtt{y}$; Die Bedingung `ub(y)` $=$ `Stagg_prec` muß erfüllt sein. `ub(x)` $\neq$ `Stagg_prec` ist zulässig.

function `sqrt5` (`var` x : `Stagg`) :   (s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\sqrt[5]{\mathtt{x}} := \{r \mid r = \sqrt[5]{t} \ \wedge \ t \in \mathtt{x}\}$ und der Anweisung `y := sqrt5(x)` erhält man: $\sqrt[5]{\mathtt{x}} \subseteq \mathtt{y}$; Die Bedingung `ub(y)` $=$ `Stagg_prec` muß erfüllt sein. `ub(x)` $\neq$ `Stagg_prec` ist zulässig.

function `power` (`var` x : `Stagg`; n : `integer`) :   (stagg) `Stagg[-1..Stagg_prec];`

Mit $x^n := \{r \mid r = t^n \ \wedge \ t \in \mathtt{x}, n = 0, \pm 1, \ldots\}$ und `y := power(x,n)` erhält man: $\mathtt{x}^n \subseteq \mathtt{y}$; `ub(x)` $=$ `ub(y)` $=$ `Stagg_prec` muß erfüllt sein.

function `arcsin` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec]`;

Mit $\arcsin(\mathtt{x}) := \{r \mid r = \arcsin(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arcsin(x)` erhält man: $\arcsin \mathtt{x} \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `arccos` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec]`;

Mit $\arccos(\mathtt{x}) := \{r \mid r = \arccos(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arccos(x)` erhält man: $\arccos \mathtt{x} \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `arctan` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec]`;

Mit $\arctan(\mathtt{x}) := \{r \mid r = \arctan(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arctan(x)` erhält man: $\arctan \mathtt{x} \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `arccot` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec]`;

Mit $\arctan(\mathtt{x}) := \{r \mid r = \mathrm{arccot}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arccot(x)` erhält man: $\mathrm{arccot}\mathtt{x} \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `log10` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec]`;

Mit $\log_{10}(\mathtt{x}) := \{r \mid r = \log_{10}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := log10(x)` erhält man: $\log_{10}\mathtt{x} \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `ln` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec]`;

Mit $\ln(\mathtt{x}) := \{r \mid r = \ln(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := ln(x)` erhält man: $\ln \mathtt{x} \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `loga` (`var` a,x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\log_a(\mathtt{x}) := \{r \,|\, r = \log_a(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := loga(a,x)` erhält man: $\log_a \mathtt{x} \subseteq \mathtt{y}$; $0 < a \neq 1$, $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `ln1p` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\mathrm{ln1p}(\mathtt{x}) := \{r \,|\, r = \ln(1+t) \;\wedge\; t \in \mathtt{x}\}$ und `y := ln1p(x)` erhält man: $\ln(1+\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `exp` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $e^{\mathtt{x}} := \{r \,|\, r = e^t \;\wedge\; t \in \mathtt{x}\}$ und `y := exp(x)` erhält man: $e^{\mathtt{x}} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `expm1` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $e^{\mathtt{x}} - 1 := \{r \,|\, r = e^t - 1 \;\wedge\; t \in \mathtt{x}\}$ und `y := expm1(x)` erhält man: $e^{\mathtt{x}} - 1 \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `exp10` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $10^{\mathtt{x}} := \{r \,|\, r = 10^t \;\wedge\; t \in \mathtt{x}\}$ und `y := exp10(x)` erhält man: $10^{\mathtt{x}} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `exp2` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $2^{\mathtt{x}} := \{r \,|\, r = 2^t \;\wedge\; t \in \mathtt{x}\}$ und `y := exp2(x)` erhält man: $10^{\mathtt{x}} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `sin` (`var` x : `Stagg`) :	
(s_fu1)	`Stagg[-1..Stagg_prec];`

Mit $\sin(\mathtt{x}) := \{r \,|\, r = \sin(t) \ \wedge\ t \in \mathtt{x}\}$ und `y := sin(x)` erhält man: $\sin(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `cos` (`var` x : `Stagg`) :	
(s_fu1)	`Stagg[-1..Stagg_prec];`

Mit $\cos(\mathtt{x}) := \{r \,|\, r = \cos(t) \ \wedge\ t \in \mathtt{x}\}$ und `y := cos(x)` erhält man: $\cos(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `tan` (`var` x : `Stagg`) :	
(s_fu1)	`Stagg[-1..Stagg_prec];`

Mit $\tan(\mathtt{x}) := \{r \,|\, r = \tan(t) \ \wedge\ t \in \mathtt{x}\}$ und `y := tan(x)` erhält man: $\tan(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `cot` (`var` x : `Stagg`) :	
(s_fu1)	`Stagg[-1..Stagg_prec];`

Mit $\cot(\mathtt{x}) := \{r \,|\, r = \cot(t) \ \wedge\ t \in \mathtt{x}\}$ und `y := cot(x)` erhält man: $\cot(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `sinh` (`var` x : `Stagg`) :	
(s_fu1)	`Stagg[-1..Stagg_prec];`

Mit $\sinh(\mathtt{x}) := \{r \,|\, r = \sinh(t) \ \wedge\ t \in \mathtt{x}\}$ und `y := sinh(x)` erhält man: $\sinh(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `cosh` (`var` x : `Stagg`) :	
(s_fu1)	`Stagg[-1..Stagg_prec];`

Mit $\cosh(\mathtt{x}) := \{r \,|\, r = \cosh(t) \ \wedge\ t \in \mathtt{x}\}$ und `y := cosh(x)` erhält man: $\cosh(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `Stagg_prec` muß erfüllt sein.

function `tanh` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\tanh(\mathtt{x}) := \{r \,|\, r = \tanh(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := tanh(x)` erhält man: $\tanh(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `coth` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\coth(\mathtt{x}) := \{r \,|\, r = \coth(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := coth(x)` erhält man: $\coth(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `arsinh` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\mathrm{arsinh}(\mathtt{x}) := \{r \,|\, r = \mathrm{arsinh}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arsinh(x)` erhält man: $\mathrm{arsinh}(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `arcosh` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\mathrm{arcosh}(\mathtt{x}) := \{r \,|\, r = \mathrm{arcosh}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arcosh(x)` erhält man: $\mathrm{arcosh}(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `artanh` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\mathrm{artanh}(\mathtt{x}) := \{r \,|\, r = \mathrm{artanh}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := artanh(x)` erhält man: $\mathrm{arcosh}(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

function `arcoth` (`var` x : `Stagg`) :
(s_fu1) `Stagg[-1..Stagg_prec];`

Mit $\mathrm{arcoth}(\mathtt{x}) := \{r \,|\, r = \mathrm{arcoth}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arcoth(x)` erhält man: $\mathrm{arcosh}(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{Stagg\_prec}$ muß erfüllt sein.

3.2.1.7.1 Tabelle: Standardfunktionen

Standardfunktionen vom Typ Stagg; x : Stagg			
Funktion	**Aufruf**	**Argumentbereich**	**Hinweise**
$\lvert x \rvert$	abs(x)	Stagg–Bereich	Seite 73, ∘
$\sqrt{x}$	sqrt(x)	$0 \le \inf(x) \le \sup(x) \le 9.9 \cdot 10^{+255}$	Seite 73, ∘
$\sqrt[3]{x}$	sqrt3(x)	$x \subset [-9.9 \cdot 10^{+255}, +9.9 \cdot 10^{+255}]$	Seite 73, ∗
$\sqrt[5]{x}$	sqrt5(x)	$x \subset [-9.9 \cdot 10^{+255}, +9.9 \cdot 10^{+255}]$	Seite 73, ∗
x^n	power(x,n)	Stagg–Bereich, begrenzt durch n	Seite 73, ∘
arcsin(x)	arcsin(x)	$-1 \le \inf(x) \le \sup(x) \le +1$	Seite 74, ∗
arccos(x)	arccos(x)	$-1 \le \inf(x) \le \sup(x) \le +1$	Seite 74, ∗
arctan(x)	arctan(x)	$x \subset [-9.9 \cdot 10^{+255}, +9.9 \cdot 10^{+255}$	Seite 74, ∗
arccot(x)	arccot(x)	$x \subset [-9.9 \cdot 10^{+255}, +9.9 \cdot 10^{+255}$	Seite 74, ∗
$\log_{10}(x)$	log10(x)	$x \subset [10^{-255}, 9.99\ldots 99 \cdot 10^{+255}]$	Seite 74, ∗
ln(x)	ln(x)	$x \subset [10^{-255}, 9.99\ldots 99 \cdot 10^{+255}]$	Seite 74, ∗
$\log_a(x)$	loga(a, x)	$a, x \subset [10^{-255}, 9.99\ldots 99 \cdot 10^{+255}]$	Seite 75, ∗
$\ln(1+x)$	ln1p(x)	$x \subset (-1, +9.99\ldots 99 \cdot 10^{+255}]$	Seite 75, ∗
e^x	exp(x)	$x \subset [-9.9\ldots \cdot 10^{+255}, +589.46\ldots]$	Seite 75, ∗
$e^x - 1$	expm1(x)	$x \subset [-9.9\ldots \cdot 10^{+255}, +589.46\ldots]$	Seite 75, ∗
10^x	exp10(x)	$x \subset [-9.9\ldots \cdot 10^{+255}, +256)$	Seite 75, ∗
2^x	exp2(x)	$x \subset [-9.9\ldots \cdot 10^{+255}, +850.41\ldots)$	Seite 75, ∗
sin(x)	sin(x)		Seite 76, ∗
cos(x)	cos(x)		Seite 76, ∗
tan(x)	tan(x)		Seite 76, ∗
cot(x)	cot(x)		Seite 76, ∗
∘ : exportiert von Modul `stagg`; ∗ : exportiert von Modul `s_fu1`			

3.2.1.7.2 Tabelle: Standardfunktionen

Standardfunktionen vom Typ Stagg; x : Stagg			
Funktion	**Aufruf**	**Argumentbereich**	**Hinweise**
$\sinh(x)$	$\sinh(x)$	$x \subset [-590.15\ldots, +590.15\ldots]$	Seite 76, $*$
$\cosh(x)$	$\cosh(x)$	$x \subset [-590.15\ldots, +590.15\ldots]$	Seite 76, $*$
$\tanh(x)$	$\tanh(x)$	$x \subset [-9.9 \cdot 10^{+255}, +9.9 \cdot 10^{+255}]$	Seite 77, $*$
$\coth(x)$	$\coth(x)$	$0 \notin x \subset [-9.9 \cdot 10^{+255}, +9.9 \cdot 10^{+255}]$	Seite 77, $*$
$\mathrm{arsinh}(x)$	$\mathrm{arsinh}(x)$	$x \subset [-9.9 \cdot 10^{+255}, +9.9 \cdot 10^{+255}]$	Seite 77, $*$
$\mathrm{arcosh}(x)$	$\mathrm{arcosh}(x)$	$x \subset [1, +9.9 \cdot 10^{+255}]$	Seite 77, $*$
$\mathrm{artanh}(x)$	$\mathrm{artanh}(x)$	$x \subset (-1, +1)$	Seite 77, $*$
$\mathrm{arcoth}(x)$	$\mathrm{arcoth}(x)$	$x \not\subset [-1, +1]$	Seite 77, $*$
$\circ$: exportiert von Modul `stagg`; $*$: exportiert von Modul `s_fu1`			

3.2.1.8 Mathematische Konstanten

function `Pi_Stagg` : `Stagg[-1..Stagg_prec];` (stagg)

`x := Pi_Stagg;` liefert mit `x : Stagg` eine garantierte, maximalgenaue Einschließung von π. Die Bedingung `ub(x)` = `Stagg_prec` muß erfüllt sein.

function `Pi_sm` : `Stagg[-1..20];` (stagg)

Obige Funktion liefert für π eine maximalgenaue Einschließung mit größtmöglicher Präzision im `real`–Format. Der relative Fehler dieser Einschließung ist kleiner als $3.2 \cdot 10^{-268}$.

function `Pih_sm` : `Stagg[-1..20];` (stagg)

Obige Funktion liefert für $\pi/2$ eine maximalgenaue Einschließung mit größtmöglicher Präzision im `real`–Format. Der relative Fehler dieser Einschließung ist kleiner als $3.2 \cdot 10^{-268}$.

function `Piv_sm` : `Stagg[-1..20];` (stagg)

Obige Funktion liefert für $\pi/4$ eine maximalgenaue Einschließung mit größtmöglicher Präzision im `real`–Format. Der relative Fehler dieser Einschließung ist kleiner als $3.2 \cdot 10^{-268}$.

3.2.1.9 Ein-/Ausgabeprozeduren

procedure `read` (**var** F : `Text`; **var** a : `Stagg`); (stagg)

Mit dem Aufruf `read(a)` wird der Anwender aufgefordert, ein beliebiges Intervall $a := a_1 + a_2 + \ldots, a_{ub(\mathtt{a})} + [A.inf, A.sup]$ mit z.B. $ub(\mathtt{a}) = 2$ in folgender Form einzugeben:

$$a_1 \sqcup a_2 \sqcup [A.inf, A.sup] \quad < \texttt{enter} >$$

Dabei dürfen sich die a_i sogar überlappen, es muß jedoch $A.inf \leq A.sup$ erfüllt sein. `ub(a)` $\neq$ `Stagg_prec` ist zulässig.

Bitte beachten Sie, daß die staggered Variable `a` ihre Komponenten `a[j]` genau so übernimmt, wie der Anwender die entsprechenden Komponenten eingegeben hat. Daher muß also z.B. `a[1]` nicht der Mittelpunkt oder eine Approximation des Mittelpunktes von `a` sein!

procedure `write` (**var** F : `Text`; **var** a : `Stagg`); (stagg)

Ist `a` das Ergebnis einer Rechnung im *staggered* Format, so entstehen die `a[1],a[2],...,a[ub(a)]` durch das Auslesen des Akkumulators zur jeweils nächsten `real`–Zahl, wodurch die `a[i]` unterschiedliche Vorzeichen erhalten können. So liefern z.B. mit `Stagg_prec=3` die Anweisungen

```
x := 3;   a := sqrt(x);
For k := -1 to UB(a) do writeln(a[k]);
```

den folgenden Bildschirmausdruck

```
 1.280525381037E-038                  { = a[-1]        }
 1.280525381039E-038                  { = a[0 ]        }
 1.732050807569E+000                  { = a[1 ]        }
-1.227064725537E-013                  { = a[2 ]        }
 4.150587236693E-026                  { = a[3 ]        }
```

der den Zahlenwert

$$\sqrt{3} = 1.7320508075688772935274463415058723669428052538103 80\ldots$$

nicht direkt erkennen läßt. Die analogen Anweisungen

```
x := 3;   a := sqrt(x);   write(a);
```

liefern jetzt mit `write(a)` den gut lesbaren Ausdruck für die Einschließung von $\sqrt{3}$

```
  1.732050807568E+000                        { <> a[1]         }
  8.772935274463E-013                        { <> a[2]         }
  4.150587236694E-026                        { <> a[3]         }
[+2.805253810370E-039,+2.805253810390E-039] { <> [a[-1],x[0]] }
```

Die Ausgabe von `write(a)` schließt das Intervall `a` garantiert ein, wobei sich die drei ersten Komponenten nicht überlappen und einerlei Vorzeichen besitzen. Die erste Komponente `1.732050807568E+000` ist eine hochgenaue Approximation des Mittelpunktes von `a`. `ub(a)` $\neq$ `Stagg_prec` ist zulässig.

> Durch die Ausgabe von `write(a)` werden die internen Werte $a[i]$ i.a. nicht korrekt wiedergegeben! $i = -1, 0, 1 \ldots,$ `Stagg_prec`; `write(a)` liefert jedoch eine gut lesbare Einschließung von `a`.

3.2.1.10 Programmbeispiele

3.2.1.10.1 Korrekte Anwendung der Routinen

Das nachfolgende Programm `stagg2` zeigt die verschiedenen Möglichkeiten zur Anwendung der Funktionen `change_length`, `write`, `Pi_Stagg`, `Pi_sm` und der arithmetischen Operatoren:

```
program stagg2(input,output);
use stagg, type_def;
var p : integer;
procedure main;
var a : Stagg[-1..2];                                  {Z 5 }
    x : Stagg[-1..Stagg_prec];                         {Z 6 }
begin { main }                                         {Z 7 }
   a[-1] := 0;  a[0] := 0;  a[1] := 1;  a[2] := 0;     {Z 8 }
   writeln(a);                                         {Z 9 }
   x := change_length(a);     { x = a = 1         }    {Z 10}
   writeln(x);                                         {Z 11}
   x := a + pi_sm - pi_Stagg; { x schliesst 1 ein }    {Z 12}
   writeln(x);                                         {Z 13}
   set_Stagg_prec(4);                                  {Z 14}
   writeln(pi_sm - pi_sm);                             {Z 15}
   writeln(x);                                         {Z 16}
end;  { main }
BEGIN { stagg2 }
   writeln('Der im Modul type_def voreingestellte ');
   writeln('Stagg_prec-Wert lautet: ',get_Stagg_prec);   {Z 20}
   writeln('Neuer Stagg_prec Wert = ?');      read(p);   {Z 21}
   set_Stagg_prec(p);                                  {Z 22}
   main;
END.  { stagg2 }
```

Startet man das Programm `stagg2` und gibt man für die integer-Variable `Stagg_prec` über `read(p)` den neuen Wert 3 ein, so liefert `writeln(a)` nach Zeile 9 (Z 9) den folgenden Ausdruck:

```
 +1.000000000000E+000
 -0.000000000000E+000
[-0.000000000000E+000, 0.000000000000E+000]
```

Beachten Sie bitte, daß jetzt in `writeln(a)` die Obergrenze `ub(a)=2` von `Stagg_prec` verschieden ist, was nach Seite 82 durchaus erlaubt ist.

Mit der Anweisung `x := change_length(a)` (Z 10) wird das Intervall `a` der Präzision 2 auf die Variable `x` mit der Präzision 3 identisch übertragen, und `writeln(x)` in (Z 11) liefert damit den folgenden Bildschirmausdruck:

```
 +1.000000000000E+000
 -0.000000000000E+000
 -0.000000000000E+000
[-0.000000000000E+000, 0.000000000000E+000]
```

In der Anweisung `x := a + pi_sm - pi_Stagg` (Z 12) wird zunächst der `+` Operator mit den Operanden `a`, `pi_sm` aufgerufen, deren Präzision beide von `Stagg_prec` verschieden sind. Nach Tabelle 3.2.1.6 und den Anmerkungen von Seite 71 ist jedoch die Präzision der Summe `a + pi_sm` und damit auch die Präzision der ganzen rechten Seite in (Z 12) durch `Stagg_prec=3` gegeben, so daß die beiden Operanden des Zuweisungsoperators `:=` wieder die gleiche Präzision besitzen, wie es in der unteren Tabelle (drittletzte Zeile) auf Seite 71 verlangt wird. `writeln(x)` von Zeile 13 (Z 13) liefert dann den Ausdruck:

```
 +1.000000000000E+000
 -0.000000000000E+000
 -0.000000000000E+000
[-1.000000000000E-051, 1.000000000000E-051]
```

Erwartungsgemäß wird die Eins eingeschlossen, alledings nicht maximalgenau, da die Intervalle `pi_sm` und `pi_Stagg` von Null verschiedene Durchmesser haben und π in unterschiedlicher Genauigkeit einschließen.

In Zeile 14 wird mit der Prozedur `set_Stagg_prec(4)` die vom Modul `type_def` exportierte Variable `Stagg_prec` und damit die Präzision auf 4 gesetzt. Die Anweisung `writeln(pi_sm - pi_sm)` (Z 15) liefert deshalb den folgenden Ausdruck:

```
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[-1.000000000000E-267, +1.000000000000E-267]
```

Der noch ungeübte Anwender wird als Ausgabe vermutlich das Punktintervall Null erwarten, da identische Intervalle subtrahiert werden. Wegen der von Null verschiedenen Intervalldurchmesser liefert die Intervallrechnung jedoch nur eine Einschließung der Null!

Die letzte Anweisung `writeln(x)` in Zeile 16 liefert den Ausdruck

```
+1.000000000000E+000
-0.000000000000E+000
-0.000000000000E+000
[-1.000000000000E-051, 1.000000000000E-051]
```

womit gezeigt wird, daß sich die Präzision der vorher mit `Stagg_prec=3` in Zeile 6 deklarierten Variablen `x` durch den späteren Prozeduraufruf `set_Stagg_prec(4)` *nicht* geändert hat!

Beachten Sie bitte, daß im Programm `stagg2` die Präzision der Variablen `x` vom Typ `Stagg` vor deren Deklaration durch eine Wertzuweisung an die integer-Variable `Stagg_prec` festlegt wird. Im genannten Programm erfolgt daher die Definition der Präzision über die `read(p)` Anweisung vor dem Aufruf der Prozedur `main`. Damit wird die Programmkonstruktion mit Hilfe dieser Prozedur erst nachträglich verständlich.

3.2.1.10.2 Staggered Größen variabler Präzision

Das nachfolgende Beispiel zeigt den Einsatz der staggered Variablen `x`, deren Präzision innerhalb der Prozedur `main` jetzt beliebig gewählt werden kann.

```
program stagg3(input,output);
use stagg, type_def;
var p : integer;
   procedure main;
   var y : Stagg[-1..Stagg_prec];
       x : Stagg;
       s : integer;
   begin
      s := get_Stagg_prec;  { sichert die alte Praezision  p }
      allocate(x,-1..5);    { x jetzt mit Praezision     5   }
      set_Stagg_prec(5);    { Rechnungen mit Praezision  5   }
      x := 7;     x := sqrt(x);
      set_Stagg_prec(s);    { Realisiert alte Praezision  p  }
      y := x * x;           { y liefert Einschliessung von 7 }
      free(x);              { Gibt den Speicher von x frei.  }
   end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);      { Praezision wird auf p gesetzt. }
   main;
end. { stagg3 }
```

Hinweise:

1. Weitere Informationen zu den Prozeduren `allocate` und `free` findet man in [17, Seite 34].
2. Nach `set_Stagg_prec(5)` und der Anweisung `x := sqrt(x)` liefert `x` eine Einschließung von $\sqrt{7}$ mit der Präzision 5.
3. Durch `set_Stagg_prec(s)` wird die ursprüngliche, mit p gewählte, Präzision wiederhergestellt. Das Produkt `x * x` wird dann in dieser Präzision berechnet und kann der Variablen `y`, die ursprünglich in der gleichen Präzision vereinbart wurde, zugewiesen werden.

3.2.1.10.3 Legendre Polynome der Ordnung n

Das nachfolgende Programm `stagg4` realisiert die Legendre Polynome $P_n(x)$ der Ordnung $n = 0, 1, 2, \ldots$ für reelle *staggered* Argumente x, und demonstriert die starken Auslöschungen beim Auswerten des Polynoms $P_5(x)$ in der Nähe seiner Nullstelle $x_0 = 0.5384\ldots$. Benutzt wird die dreigliedrige Rekursionsformel:

$$\begin{aligned} P_0(x) &:= 1, \\ P_1(x) &:= x, \\ P_n(x) &:= \frac{2n-1}{n}\, x \cdot P_{n-1}(x) - \frac{n-1}{n}\, P_{n-1}(x), \qquad n = 2, 3, \ldots \end{aligned}$$

```
program stagg4(input,output);
use stagg, type_def;
var p : integer;
procedure main;
var x,y : Stagg[-1..Stagg_prec];
   a    : Stagg[-1..6];
function legendreP (n : integer; var x: Stagg):
                        Stagg[-1..Stagg_prec];
{ Legendre-Polynom der Ordnung n;     }
{ ub(x)=Staggprec muss erfuellt sein. }
var h,h1,h2,f1,f2,is : Stagg[-1..Stagg_prec];
    i                : integer;
begin
   if n < 0 then n := 0;
   if n = 0 then legendreP := 1;
   if n = 1 then legendreP := x;
```

```
   if n > 1 then
   begin
      h2 := 1;  h1 := x;
      For i := 2 to n do
      begin
         is := i;
         f2 := 1 - 1/is;   f1 := f2 + 1;
         h := f1*x*h1 - f2*h2;
         h2 := h1;  h1 := h;
      end;
      legendreP := h;
   end;
end; { legendreP }
begin { main }
   { Punktnaeherung der kleinsten positiven Nullstelle }
   { des Legrendre Polynoms P_5(x) 5. Ordnung.         }
   a[-1]:= 0;                    a[0] := 0;
   a[1] := 5.384693101056E-001;  a[2] := 8.309103631442E-014;
   a[3] := 7.002088049672E-028;  a[4] := 8.660690555995E-041;
   a[5] := 6.202231627059E-054;  a[6] := 4.711853677553E-067;
   x := change_length(a); { x mit der Praezision Stagg_prec }
   y := legendreP(5,x);   { enthaelt das Argument a.       }
   writeln('P_5(a) =');   writeln(y);
end;  { main }
BEGIN { stagg4 }
   repeat
      writeln('Neuer Stagg_prec Wert = ?');  read(p);
      set_Stagg_prec(p);
      main;
   until false;
END.  { stagg4 }
```

In der obigen `repeat`-Schleife kann für jede neue Auswertung von $P_5(a)$ eine andere Präzision gewählt werden, die durch `set_Stagg_prec(p)` festgelegt wird. Die Summe

$$a = \sum_{i=1}^{6} a[i] = 0.538469310\ \ldots\ 677553 \approx x_0$$

ist eine maximalgenaue Näherung der kleinsten positiven Nullstelle x_0 von $P_5(x)$ mit 79 Dezimalziffern. Nach der Rekursionsformel von Seite 86 ist $P_5(x)$ gegeben durch: $P_5(x) := \frac{63}{8}x^5 - \frac{35}{4}x^3 + \frac{15}{8}x$
Beachten Sie bitte, daß der Funktionsterm z.B. von $P_{500}(x)$ zur Auswertung dieses Polynoms mit Hilfe der Funktion `LegendreP` nicht bekannt sein muß. In folgender Tabelle sind die Einschließungen der $P_5(a)$ in Abhängigkeit von der mit `Stagg_prec` gewählten Präzision zusammengestellt:

Einschließungen von $P_5(a)$	
`Stagg_prec = 1`	`Stagg_prec = 5`
−1.500000000000 E −026	0.000000000000 E −000
−9.500000000000 E −026	0.000000000000 E −000
+9.500000000000 E −026	0.000000000000 E −000
	0.000000000000 E −000
	0.000000000000 E −000
	−4.000000000000 E −078
	+4.000000000000 E −078
`Stagg_prec = 6`	`Stagg_prec = 8`
−2.174614560285 E −080	−2.174614560286 E −080
0.000000000000 E −000	−1.268983901522 E −093
0.000000000000 E −000	−6.107437003247 E −106
0.000000000000 E −000	0.000000000000 E −000
0.000000000000 E −000	0.000000000000 E −000
0.000000000000 E −000	0.000000000000 E −000
−3.500000000000 E −091	0.000000000000 E −000
+3.500000000000 E −091	0.000000000000 E −000
	−2.800000000000 E −117
	+2.800000000000 E −117

Da das vorgegebene Argument a sehr dicht an der Nullstelle liegt, erhält man bei der Auswertung von $P_5(a)$ so starke Auslöschung, daß mit der Präzision `Stagg_prec=1,...,5` noch nicht einmal das richtige Vorzeichen von $P_5(a)$ bestimmt wird. Erst mit `Stagg_prec=6`, d.h. mit einer internen Präzision von $(6+1)\cdot 13 = 91$ Ziffern, erhält man lediglich 10 verläßliche Dezimalstellen des korrekten Wertes $P_5(a) = -2.1746\ \ldots \cdot 10^{-80}$. Mit der Präzision `Stagg_prec=20` berechnet man eine Einschließung von $P_5(a)$, die sogar 182 korrekte Dezimalziffern aufweist.

3.2.1.10.4 Einschließung rationaler Ausdrücke

Das folgende Programm stagg5 berechnet Einschließungen der Funktion

$$f(x,y,u,v) = \frac{1682xy^4 + 3x^3 + 29xy^2 - 2x^5 + 832}{107751 \cdot (u^4 - (2v)^2 \cdot (1+v^2))}$$

die für $x = 192119201,\ y = 35675640,\ u = 665857,\ v = 470832$ den Wert

$$f(x,y,u,v) = 1783$$

annimmt.

```
program stagg5;
{ Berechnung eines reellen rationalen Ausdrucks:       }
{ Zaehler(x,y) = 1682xy^4 + 3x^3 + 29xy^2 - 2x^5 + 832 }
{ Nenner (u,v) = 10775 * ( u^4 - (2v)^2*(1+v^2) )      }
use type_def, { exportiert: type Stagg; var Stagg_prec }
    stagg,    { exportiert: operator :=, *;  write(x)  }
    s_fu1;    { exportiert: function power,....        }
var prec : integer;    { Zur Festlegung der Praezision }

procedure main;
var x,y,u,v : Stagg[-1..Stagg_prec];

function f(x,y,u,v : Stagg) : Stagg[-1..Stagg_prec];
var Z,N : Stagg[-1..Stagg_prec];
begin
   Z := 1682*x*power(y,4)+3*power(x,3)+29*x*power(y,2)
                                    -2*power(x,5)+832;
   N := 107751*(power(u,4) - power(2*v,2)*(1+v*v));
   f := Z / N;
end; { f }
begin
   writeln('x : Stagg   = ?');  read(x);
   writeln('y : Stagg   = ?');  read(y);
   writeln('u : Stagg   = ?');  read(u);
   writeln('v : Stagg   = ?');  read(v);
   writeln; writeln('f(x,y,u,v) = ');
   writeln(f(x,y,u,v));
end; { main }
```

```
begin
   repeat
      writeln('In type_def definierter Stagg_prec Wert:');
      writeln('Stagg_prec = ',Stagg_prec);
      writeln('gewuenschter Stagg_prec-Wert = ?'); read(prec);
      set_Stagg_prec(prec);
      writeln('Neuer Wert: Stagg_prec = ',Stagg_prec);
      main;
   until false;
end.
```

Der Start des Programms `stagg5` liefert das folgende Ablaufprotokoll:

```
In type_def definierter Stagg_prec Wert:
Stagg_prec = 1
gewuenschter Stagg_prec-Wert = ?
3                    <enter>
Neuer Wert: Stagg_prec = 3
x : Stagg   = ?
--- 3 Stagg components and 1 interval are needed in this sequence --
192119201 0 0 [0,0]  <enter>
y : Stagg   = ?
--- 3 Stagg components and 1 interval are needed in this sequence --
35675640 0 0 [0,0]   <enter>
u : Stagg   = ?
--- 3 Stagg components and 1 interval are needed in this sequence --
665857 0 0 [0,0]     <enter>
v : Stagg   = ?
--- 3 Stagg components and 1 interval are needed in this sequence --
470832 0 0 [0,0]     <enter>

f(x,y,u,v) =
 1.783000000000E+003
-0.000000000000E+000
-0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]
```

Anmerkungen:

1. Mit `Stagg_prec=3`, d.h. mit einer Präzision von $(3+1)\cdot 13 = 52$ Ziffern erhält man sogar eine Punkteinschließung des exakten Wertes 1783.

2. Schreiben Sie ein Programm, das die Funktion $f(x,y,u,v)$ nur in einfacher Gleitkomma–Arithmetik auswertet und vergleichen sie die sehr unterschiedlichen Ergebnisse!

3.2.2 Der Basistyp `Longreal`

Wir betrachten jetzt das *staggered* Format zum Basistyp `Longreal`. Bitte beachten Sie auch die Einführung auf Seite 61, die sich zwar auf den Datentyp `real` bezieht, aber auf den `Longreal`-Typ direkt übertragbar ist. Der notwendige Datentyp sowie die erforderlichen Operatoren, Funktionen und Prozeduren werden von den Hilfs–Modulen `type_def` und `lstagg` exportiert.

3.2.2.1 Datenstruktur

Datentyp

type LStagg = **dynamic array**[∗] **of** Longreal; (`type_def`)

Präzisionsparameter

var LStagg_prec : integer; (`type_def`)

Variablen-Vereinbarung

```
var x : LStagg[-1..LStagg_prec];
```

3.2.2.2 Präzisionsänderung

Eine Präzisionsänderung kann mit Hilfe der Prozedur `set_LStagg_prec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_LStagg_prec` (**prec** : `integer`); (`type_def`)

Mit der Eingangsgröße `prec` wird durch `set_LStagg_prec(prec)` die Variable `LStagg_prec` entsprechend geändert. Der Prozeduraufruf muß **vor** der Vereinbarung der entsprechenden *staggered* Variablen erfolgen! Beachten Sie dazu auch die Anmerkung 3. von Seite 64, die sich direkt auf den Typ `Longreal` übertragen läßt.

Der aktuelle Wert der Variablen `LStagg_prec` wird durch die folgende Funktion geliefert:

function `get_LStagg_prec` : `integer`; (`type_def`)

Präzisionsanpassung

Mit der Funktion

function `change_length` (**var** a : `LStagg`) :	
(lstagg)	`LStagg[-1..LStagg_prec]`;

kann die Variable `a`, mit `ub(a)` $\neq$ `LStagg_prec` durch die Anweisung

`x := change_length(a);` $\rightsquigarrow$ `a` $\subseteq$ `x`

an die aktuelle, durch `LStagg_prec` gegebene Präzision der Variablen `x` angepaßt werden.

3.2.2.3 Hilfsfunktionen

function `convert_LStagg` (**var** lo,up : `dotprecision`) :	
(lstagg)	`LStagg[-1..LStagg_prec]`;

Mit der Anweisung

`x := convert_LStagg(lo,up);` $\rightsquigarrow$ [`lo`, `up`] $\subseteq$ `x`

wird das Eingangs–Intervall $[lo, up]$ durch die *staggered* Variable `x` maximalgenau eingeschlossen, wobei die Bedingung `ub(b)` = `LStagg_prec` erfüllt sein muß. Fast alle Hilfsfunktionen und Arithmetikoperatoren werden mit Hilfe dieser Funktion `convert_LStagg` realisiert.

procedure `normal` (**var a** : `LStagg`);	(`lstagg`)

Wenn sich bei der Eingangsgröße `a` die `a[i]` für $i \geq 1$ überlappen, was z.B. nach einer Division eintreten kann, so liefert der Aufruf `normal(a)` mit der Ausgangsgröße `a` eine maximalgenaue Einschließung der Eingangsgröße, wobei sich die `a[i]` der Ausgangsgröße dann nicht mehr überlappen. Die Bedingung `ub(a)` = `LStagg_prec` muß erfüllt sein.

function `blow` (x : `LStagg`; eps : `real`) :	
(lstagg)	`LStagg[-1..ub(x)]`;

`y := blow(x,eps)` liefert die übliche ε–Aufblähung von `x`. $ub(y) \stackrel{!}{=} ub(x)$

function `inf` (**var** a : `LStagg`) :
(lstagg) `LStagg[-1..LStagg_prec]`;

function `sup` (**var** a : `LStagg`) :
(lstagg) `LStagg[-1..LStagg_prec]`;

liefern Punkteinschließungen der Unter- bzw. Obergrenze des Intervalls `a`, wobei die Bedingung `ub(a)` = `LStagg_prec` erfüllt sein muß.

function `diam` (**var** a : `LStagg`) : Longreal; (lstagg)

liefert mit der Anweisung `r := diam(a);` den aufgerundeten Durchmesser des Intervalls `a` an die Longreal–Variable `r`. `ub(a)` $\neq$ `LStagg_prec` ist erlaubt.

function `mid` (**var** a : `LStagg`) : `LStagg[-1..LStagg_prec]`;
(lstagg)

liefert mit der Anweisung `m := mid(a)` eine Punkt–Einschließung des gerundeten Mittelpunktes von `a`. Es muß gelten: `ub(a)` = `LStagg_prec`

function `hull` (**var** x,y : `LStagg`) : `LStagg[-1..LStagg_prec]`;
(lstagg)

liefert Einschließung von $[\inf(x), \sup(y)]$; `ub(x)` $\neq$ `ub(y)` $\neq$ `LStagg_prec` ist zulässig.

procedure `add_LStagg_times_LStagg`	
(lstagg)	(**var** a,b : `LStagg`; **var** d1,d2 : `dotprecision`)

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter : a,b : `LStagg`; d1,d2 : `dotprecision`

Ausgangsparameter: d1,d2 : `dotprecision`

Bezeichnet man jetzt zur Unterscheidung die Ausgangsparameter d1,d2 mit D1,D2, so gilt nach dem Aufruf der Prozedur: $a \cdot b + [d1, d2] \subseteq [D1, D2]$, wobei folgende Bedingung erlaubt ist: $\texttt{ub}(a) \neq \texttt{ub}(b) \neq \texttt{LStagg\_prec}$.

procedure `add_Longreal_times_LStagg` (r : `Longreal`;	
(lstagg)	**var** a : `LStagg`; **var** d1,d2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangspar. : r : `Longreal`; a : `LStagg`; d1,d2 : `dotprecision`

Ausgangspar.: d1,d2 : `dotprecision`

Bezeichnet man jetzt zur Unterscheidung die Ausgangsparameter d1,d2 mit D1,D2, so gilt nach dem Aufruf der Prozedur: $r \cdot a + [d1, d2] \subseteq [D1, D2]$, wobei folgende Bedingung erlaubt ist: $\texttt{ub(a)} \neq \texttt{Stagg\_prec}$.

procedure `mant_exp` (**var** a,m : `LStagg`; **var** ex : `integer`);
(lstagg)

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter : a : `LStagg`;

Ausgangsparameter: m : `LStagg`; ex : `integer`;

Der Aufruf `mant_exp(a,m,ex)` berechnet zu gegebenem, punktförmigem `a` die zugehörige Mantisse `m` und den Zehner–Exponenten `ex`, so daß im Falle $\texttt{a} \neq 0$ gilt: $\texttt{a} \subseteq \texttt{m} \cdot 10^{\texttt{ex}}$, $\texttt{m} \subseteq [0.1, 1)$, wobei der Durchmesser von `m` von Null verschieden sein kann. $\texttt{a} = 0$: $\texttt{m} = 0$ und $ex = -2147483647$.

Bedingungen: $ub(a) \stackrel{!}{=} ub(m)$, aber $ub(a) \neq$ `LStagg_prec` ist zulässig. Ist `a` nicht punktförmig, so erfolgt eine entsprechende Fehlermeldung mit Programmabbruch.

operator `M_10HOCH` (x : `LStagg`; k : `integer`) :
(lstagg) res : `LStagg[-1..ub(x)]`;

`y := x M_10HOCH k` liefert schnelle Einschließung von $x \cdot 10^k \subseteq$ `y`, falls $-531 \leq k \leq 511$. $ub(y) \stackrel{!}{=} ub(x)$, aber `ub(x)` $\neq$ `LStagg_prec` ist zulässig.

function `round` (**var** x : `LStagg`) : `integer`; (lstagg)

Null wird zu x addiert, um mögliche Überlappungen der x[j] zu beseitigen. Dann wird x[1] zur nächsten `integer`-Zahl gerundet. $ub(x) \stackrel{!}{=}$ `LStagg_prec`

procedure `mul1pmeps` (**var** a : `LStagg`; eps : `Longreal`); (lstagg)

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter : a : `LStagg`; eps : `Longreal`;

Ausgangsparameter: a : `LStagg`;

Der Aufruf `mul1pmeps(a,eps)` berechnet zu den gegebenen Eingangsgrößen `a` und `eps` > 0 mit der Ausgangsgröße `a` eine garantierte Einschließung des Intervallprodukts: $a \cdot [1 - eps, 1 + eps] \subseteq a$; $ub(a) \stackrel{!}{=}$ `LStagg_prec`

3.2.2.4 Transferfunktionen

function `ival` (**var** a : `LStagg`) : Linterval; (lstagg)

Die Anweisung `x := ival(a)` liefert für `a: LStagg` eine maximalgenaue Einschließung: `a`$\subseteq$ `x` : Linterval. `ub(a)` $\neq$ `Stagg_prec` ist zulässig.

function `LStaggintval` (r : `Longreal`) :
(lstagg) `LStagg[-1..LStagg_prec];`

`a := LStaggintval(r)` liefert Punkteinschließung für `r` $\in$ `a`. Die Bedingung `ub(a)` $=$ `LStagg_prec` muß erfüllt sein.

procedure `TRANS` (**var** a : `LStagg`; **var** r : `Longreal`); (lstagg)

besitzt folgende Ein- und Ausgangsparameter:

Eingangspar. : a : `LStagg`;

Ausgangspar.: r : `Longreal` ist maximalgenauer Mittelpunkt von `a`

Die Ungleichung `ub(a)` $\neq$ `LStagg_prec` ist zulässig.

procedure `TRANS` (**var** a : `LStagg`; **var** x,y : `Longreal`); (lstagg)

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter : a : `LStagg`;

Ausgangsparameter: x,y : `Longreal`

$(x + y)$ ist maximalgenaue Näherung des Mittelpunkts von `a` im doppelt-langen Format. Die Ungleichung `ub(a)` $\neq$ `LStagg_prec` ist zulässig.

function `LStaggintval` (**var** a : `Linterval`) :
(lstagg) `LStagg[-1..LStagg_prec];`

`b := LStaggintval(a);` $\rightsquigarrow$ Die Intervalle `a`,`b` haben zwar unterschiedliche Datenstrukturen, sind aber mathematisch identisch[2]. Für obige Anweisung muß die Bedingung `ub(b)` $=$ `LStagg_prec` erfüllt sein.

[2]Ist `a` kein Punktintervall, so ist der Durchmesser von `b` für eine sinnvolle *staggered* Arithmetik i.a. zu groß!

function `LStaggintval` (x,y : `Longreal`) :	
(lstagg)	`LStagg[-1..LStagg_prec]`;

`b := LStaggintval(x,y);` $\rightsquigarrow$ `b = x +* y`, (Punktintervall–Hülle).
Für die Anweisung muß die Bedingung `ub(b)` = `LStagg_prec` erfüllt sein.

3.2.2.5 Zuweisungsoperatoren

Zuweisung	**Bedeutung**	
`lrv := LS`	`lrv`: gerundete Mitte v. `S`;	`ub(LS)` beliebig
`liv := LS`	`S` $\subseteq$ `liv`;	`ub(LS)` beliebig
`LSv := lr`	`LSv` = `lr`;	`ub(LSv)` beliebig
`LSv := ig`	`LSv` = `ig`;	`ub(LSv)` beliebig
`LSv := li`	`LSv` = `li`;	`ub(LSv)` beliebig
`LSv := LS`	`LSv` = `LS`;	`ub(LSv)` = `ub(LS)`
`LSv := dot`	`dotprecision` $\in$ `Sv`;	`ub(LSv)` = `Stagg_prec`
`LSv := idot`	`idotprecision` $\subseteq$ `Sv`;	`ub(LSv)` = `Stagg_prec`

Tabelle 3.1: lrv = `Longreal`–Variable, liv = `Linterval`–Variable, LSv = `LStagg`–Variable, LS = `LStagg`–Ausdruck, li = `Linterval`–Ausdruck, lr = `Longreal`–Ausdruck, ig = `integer`–Ausdruck, dot = `dotprecision`–Ausdruck, idot = `idotprecision`–Ausdruck

3.2.2.6 Vergleichs- und Arithmetik-Operatoren

rechter Operand	integer	Longreal real	LStagg
linker Operand			
monadisch			$+, -$
integer			**in** $=, <>$
Longreal real			**in**, $\diamond$ $=, <>$ $\bullet$
LStagg	$=, <>$	$\diamond$ $=, <>$ $\bullet$	**in**, **lies_in** $\diamond, =, <>$ $\bullet, ><, **$

`LStagg` Operatoren: $\diamond \in \{+, -, *, /\}$, $\bullet \in \{<, \leq, >, \geq\}$

Anmerkungen:				
LStagg	$<$	Longreal	$\longleftrightarrow$	sup(LStagg) $<$ Longreal
LStagg	$\leq$	Longreal	$\longleftrightarrow$	sup(LStagg) $\leq$ Longreal
LStagg	$>$	Longreal	$\longleftrightarrow$	inf(LStagg) $>$ Longreal
LStagg	$\geq$	Longreal	$\longleftrightarrow$	inf(LStagg) $\geq$ Longreal
LStagg	$=$	Longreal	$\longleftrightarrow$	inf(LStagg) $=$ sup(L..) $=$ Longreal
LStagg	$<$	real	$\longleftrightarrow$	sup(LStagg) $<$ real
LStagg	$\leq$	real	$\longleftrightarrow$	sup(LStagg) $\leq$ real
LStagg	$>$	real	$\longleftrightarrow$	inf(LStagg) $>$ real
LStagg	$\geq$	real	$\longleftrightarrow$	inf(LStagg) $\geq$ real
LStagg	$=$	real	$\longleftrightarrow$	inf(LStagg) $=$ sup(LStagg) $=$ real
Der Operator $><$ testet auf Disjunktheit zweier LStagg–Operanden.				
Der Operator $**$ liefert den Durchschnitt zweier LStagg-Operanden, ein leerer Durchschnitt führt zu einem Laufzeitfehler.				
Der Operator **in** zwischen zwei LStagg Operanden testet die Relation *liegt im Innern*.				
Der Operator **in** zwischen einem Longrea/real und einem LStagg Operanden testet die Relation *liegt in*.				
Der Operator **lies_in** testet die Relation *liegt in* zwischen zwei LStagg Operanden, wobei die Präzision beider Operanden durch die Variable LStagg_prec gegeben sein muß. x lies_in y $\longleftrightarrow$ x $\subseteq$ y				
Die Operatoren $<, \leq, >, \geq$ zwischen zwei punktförmigen LStagg Operanden haben die übliche Bedeutung, wobei die Präzision der beiden Operanden durch die Variable LStagg_prec gegeben sein muß. Ist einer der beiden Operanden nicht punktförmig, so erfolgt ein Laufzeitfehler.				
Bei den arithmetischen Operatoren kann die Präzision beider Operanden unabhängig vom Wert der Variablen LStagg_prec gewählt werden. Die Präzision des Operator–Ergebnisses ist jedoch stets durch den Wert der Variablen LStagg_prec gegeben.				
Bei den monadischen Operatoren muß die Präzision des rechten Operanden nicht durch den Variablenwert LStagg_prec bestimmt sein.				

3.2.2.7 Mathematische Standardfunktionen

function `abs` (`var` x : `LStagg`) :	
(lstagg)	`LStagg[-1..LStagg_prec]`;

`y := abs(x);` $\rightsquigarrow$ `y` $:= \{|r| : r \in$ `x`$\}$
Die Bedingung `ub(y)` = `ub(x)` = `LStagg_prec` muß erfüllt sein.

function `sqrt` (`var` x : `LStagg`) :	
(lstagg)	`LStagg[-1..LStagg_prec]`;

Mit $\sqrt{\mathtt{x}} := \{r \,|\, r = \sqrt{t} \;\wedge\; t \in \mathtt{x}\}$ und der Anweisung `y := sqrt(x)` erhält man: $\sqrt{\mathtt{x}} \subseteq \mathtt{y}$; Die Bedingung `ub(y)` = `LStagg_prec` muß erfüllt sein. `ub(x)` $\neq$ `LStagg_prec` ist zulässig.

function `sqrt3` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec]`;

Mit $\sqrt[3]{\mathtt{x}} := \{r \,|\, r = \sqrt[3]{t} \;\wedge\; t \in \mathtt{x}\}$ und der Anweisung `y := sqrt3(x)` erhält man: $\sqrt[3]{\mathtt{x}} \subseteq \mathtt{y}$; Die Bedingung `ub(y)` = `LStagg_prec` muß erfüllt sein. `ub(x)` $\neq$ `LStagg_prec` ist zulässig.

function `sqrt5` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec]`;

Mit $\sqrt[5]{\mathtt{x}} := \{r \,|\, r = \sqrt[5]{t} \;\wedge\; t \in \mathtt{x}\}$ und der Anweisung `y := sqrt5(x)` erhält man: $\sqrt[5]{\mathtt{x}} \subseteq \mathtt{y}$; Die Bedingung `ub(y)` = `LStagg_prec` muß erfüllt sein. `ub(x)` $\neq$ `LStagg_prec` ist zulässig.

function `power` (`var` x : `LStagg`; n : `integer`) :	
(lstagg)	`LStagg[-1..LStagg_prec]`;

Mit $\mathtt{x}^n := \{r \,|\, r = t^n \;\wedge\; t \in \mathtt{x},\, n = 0, \pm 1, \ldots\}$ und `y := power(x,n)` erhält man: $\mathtt{x}^n \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `arcsin` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec]`;

Mit $\arcsin(\mathtt{x}) := \{r \mid r = \arcsin(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arcsin(x)` erhält man: $\arcsin \mathtt{x} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `arccos` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec]`;

Mit $\arccos(\mathtt{x}) := \{r \mid r = \arccos(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arccos(x)` erhält man: $\arccos \mathtt{x} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `arctan` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec]`;

Mit $\arctan(\mathtt{x}) := \{r \mid r = \arctan(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arctan(x)` erhält man: $\arctan \mathtt{x} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `arccot` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec]`;

Mit $\arctan(\mathtt{x}) := \{r \mid r = \mathrm{arccot}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arccot(x)` erhält man: $\mathrm{arccot}\mathtt{x} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `log10` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec]`;

Mit $\log_{10}(\mathtt{x}) := \{r \mid r = \log_{10}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := log10(x)` erhält man: $\log_{10}\mathtt{x} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `ln` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $\ln(\mathtt{x}) := \{r \mid r = \ln(t) \ \wedge\ t \in \mathtt{x}\}$ und `y := ln(x)` erhält man: $\ln \mathtt{x} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `loga` (`var` a,x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $\log_a(\mathtt{x}) := \{r \mid r = \log_a(t) \ \wedge\ t \in \mathtt{x}\}$ und `y := loga(a,x)` erhält man: $\log_a \mathtt{x} \subseteq \mathtt{y}$; $0 < a \neq 1$, $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `ln1p` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $\mathrm{ln1p}(\mathtt{x}) := \{r \mid r = \ln(1+t) \ \wedge\ t \in \mathtt{x}\}$ und `y := ln1p(x)` erhält man: $\ln(1+\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `exp` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $e^{\mathtt{x}} := \{r \mid r = e^t \ \wedge\ t \in \mathtt{x}\}$ und `y := exp(x)` erhält man: $e^{\mathtt{x}} \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `expm1` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $e^{\mathtt{x}} - 1 := \{r \mid r = e^t - 1 \ \wedge\ t \in \mathtt{x}\}$ und `y := expm1(x)` erhält man: $e^{\mathtt{x}} - 1 \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `exp10` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec];`

Mit $10^{\mathtt{x}} := \{r \,|\, r = 10^t \;\wedge\; t \in \mathtt{x}\}$ und `y := exp10(x)` erhält man: $10^{\mathtt{x}} \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `exp2` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec];`

Mit $2^{\mathtt{x}} := \{r \,|\, r = 2^t \;\wedge\; t \in \mathtt{x}\}$ und `y := exp2(x)` erhält man: $10^{\mathtt{x}} \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `sin` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec];`

Mit $\sin(\mathtt{x}) := \{r \,|\, r = \sin(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := sin(x)` erhält man: $\sin(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `cos` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec];`

Mit $\cos(\mathtt{x}) := \{r \,|\, r = \cos(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := cos(x)` erhält man: $\cos(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `tan` (`var` x : `LStagg`) :	
(ls_fu1)	`LStagg[-1..LStagg_prec];`

Mit $\tan(\mathtt{x}) := \{r \,|\, r = \tan(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := tan(x)` erhält man: $\tan(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `cot` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec]`;

Mit $\cot(\mathtt{x}) := \{r \,|\, r = \cot(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := cot(x)` erhält man: $\cot(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `sinh` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec]`;

Mit $\sinh(\mathtt{x}) := \{r \,|\, r = \sinh(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := sinh(x)` erhält man: $\sinh(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `cosh` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec]`;

Mit $\cosh(\mathtt{x}) := \{r \,|\, r = \cosh(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := cosh(x)` erhält man: $\cosh(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `tanh` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec]`;

Mit $\tanh(\mathtt{x}) := \{r \,|\, r = \tanh(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := tanh(x)` erhält man: $\tanh(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `coth` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec]`;

Mit $\coth(\mathtt{x}) := \{r \,|\, r = \coth(t) \;\wedge\; t \in \mathtt{x}\}$ und `y := coth(x)` erhält man: $\coth(\mathtt{x}) \subseteq \mathtt{y}$; `ub(x)` = `ub(y)` = `LStagg_prec` muß erfüllt sein.

function `arsinh` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $\text{arsinh}(\mathtt{x}) := \{r \mid r = \text{arsinh}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arsinh(x)` erhält man: $\text{arsinh}(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `arcosh` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $\text{arcosh}(\mathtt{x}) := \{r \mid r = \text{arcosh}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arcosh(x)` erhält man: $\text{arcosh}(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `artanh` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $\text{artanh}(\mathtt{x}) := \{r \mid r = \text{artanh}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := artanh(x)` erhält man: $\text{arcosh}(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

function `arcoth` (`var` x : `LStagg`) :
(ls_fu1) `LStagg[-1..LStagg_prec];`

Mit $\text{arcoth}(\mathtt{x}) := \{r \mid r = \text{arcoth}(t) \ \wedge \ t \in \mathtt{x}\}$ und `y := arcoth(x)` erhält man: $\text{arcosh}(\mathtt{x}) \subseteq \mathtt{y}$; $\mathtt{ub(x)} = \mathtt{ub(y)} = \mathtt{LStagg\_prec}$ muß erfüllt sein.

3.2.2.7.1 Tabelle: Standardfunktionen

Standardfunktionen vom Typ LStagg; x : LStagg			
Funktion	**Aufruf**	**Argumentbereich**	**Hinweise**
$\|x\|$	abs(x)	LStagg–Bereich	Seite 100, ∘
$\sqrt{x}$	sqrt(x)	$x \subset [0, +9.99\ldots.99 \cdot 10^{+511}$	Seite 100, ∘
$\sqrt[3]{x}$	sqrt3(x)	$x \subset [-3 \cdot 10^{+511}, +3 \cdot 10^{+511}]$	Seite 100, ∗
$\sqrt[5]{x}$	sqrt5(x)	$x \subset [-3 \cdot 10^{+511}, +3 \cdot 10^{+511}]$	Seite 100, ∗
x^n	power(x,n)	LStagg–Bereich, begrenzt durch n	Seite 100, ∘
$\arcsin(x)$	arcsin(x)	$x \subset [-1, +1]$	Seite 101, ∗
$\arccos(x)$	arccos(x)	$x \subset [-1, +1]$	Seite 101, ∗
$\arctan(x)$	arctan(x)	$x \subset [-9.9 \cdot 10^{+511}, +9.9 \cdot 10^{+511}$	Seite 101, ∗
$\mathrm{arccot}(x)$	arccot(x)	$x \subset [-9.9 \cdot 10^{+511}, +9.9 \cdot 10^{+511}$	Seite 101, ∗
$\log_{10}(x)$	log10(x)	$x \subset [10^{-511}, 9 \cdot 10^{+511}]$	Seite 101, ∗
$\ln(x)$	ln(x)	$x \subset [10^{-511}, 9 \cdot 10^{+511}]$	Seite 102, ∗
$\log_a(x)$	loga(a, x)	$a, x \subset [10^{-511}, 9 \cdot 10^{+511}]$	Seite 102, ∗
$\ln(1+x)$	ln1p(x)	$x \subset (-1, +9.99\ldots 99 \cdot 10^{+511}]$	Seite 102, ∗
e^x	exp(x)	$x \subset [-9.9\ldots \cdot 10^{+511}, 1178.92\ldots]$	Seite 102, ∗
$e^x - 1$	expm1(x)	$x \subset [-9.9\ldots \cdot 10^{+255}, 1178.92\ldots]$	Seite 102, ∗
10^x	exp10(x)	$x \subset [-9.9\ldots \cdot 10^{+511}, +512)$	Seite 103, ∗
2^x	exp2(x)	$x \subset [-9.9\ldots \cdot 10^{+511}, 1700.82\ldots)$	Seite 103, ∗
$\sin(x)$	sin(x)		Seite 103, ∗
$\cos(x)$	cos(x)		Seite 103, ∗
$\tan(x)$	tan(x)		Seite 103, ∗
$\cot(x)$	cot(x)		Seite 104, ∗
∘ : exportiert von Modul `lstagg`; ∗ : exportiert von Modul `ls_fu1`			

3.2.2.7.2 Tabelle: Standardfunktionen

Standardfunktionen vom Typ LStagg; x : LStagg			
Funktion	**Aufruf**	**Argumentbereich**	**Hinweise**
$\sinh(x)$	$\sinh(x)$	$x \subset [-1179.61\ldots, +1179.61\ldots]$	Seite 104, $*$
$\cosh(x)$	$\cosh(x)$	$x \subset [-1179.61\ldots, +1179.61\ldots]$	Seite 104, $*$
$\tanh(x)$	$\tanh(x)$	$x \subset [-9.9 \cdot 10^{+511}, +9.9 \cdot 10^{+511}]$	Seite 104, $*$
$\coth(x)$	$\coth(x)$	$0 \notin x \subset [-9.9 \cdot 10^{+511}, 9.9 \cdot 10^{+511}]$	Seite 104, $*$
$\mathrm{arsinh}(x)$	$\mathrm{arsinh}(x)$	$x \subset [-9.9 \cdot 10^{+511}, +9.9 \cdot 10^{+511}]$	Seite 105, $*$
$\mathrm{arcosh}(x)$	$\mathrm{arcosh}(x)$	$x \subset [1, +9.9 \cdot 10^{+511}]$	Seite 105, $*$
$\mathrm{artanh}(x)$	$\mathrm{artanh}(x)$	$x \subset (-1, +1)$	Seite 105, $*$
$\mathrm{arcoth}(x)$	$\mathrm{arcoth}(x)$	$x \not\subset [-1, +1]$	Seite 105, $*$
$\circ$: exportiert von Modul `stagg`; $*$: exportiert von Modul `ls_fu1`			

3.2.2.8 Mathematische Konstanten

function `Pi_LStagg` : `LStagg[-1..LStagg_prec]`; (lstagg)

`x := Pi_LStagg;` liefert mit `x : LStagg` eine garantierte, maximalgenaue Einschließung von π. Die Bedingung `ub(x)` = `LStagg_prec` muß erfüllt sein.

function `Pi_lsm` : `LStagg[-1..25]`; (lstagg)

Obige Funktion liefert für π eine maximalgenaue Einschließung mit größtmöglicher Präzision im `Longreal`–Format. Der relative Fehler dieser Einschließung ist kleiner als 10^{-531}.

function `Pih_lsm` : `LStagg[-1..25]`; (lstagg)

Obige Funktion liefert für $\pi/2$ eine maximalgenaue Einschließung mit größtmöglicher Präzision im `Longreal`–Format. Der relative Fehler dieser Einschließung ist kleiner als 10^{-531}.

function `Piv_lsm` : `LStagg[-1..25]`; (lstagg)

Obige Funktion liefert für $\pi/4$ eine maximalgenaue Einschließung mit größtmöglicher Präzision im `Longreal`–Format. Der relative Fehler dieser Einschließung ist kleiner als $1.3 \cdot 10^{-531}$.

3.2.2.9 Ein-/Ausgabeprozeduren

procedure `read` (**var** F : `Text`; **var** a : `LStagg`); (lstagg)

Mit dem Aufruf `read(a)` wird der Anwender aufgefordert, ein beliebiges Intervall $a := a_1 + a_2 + \ldots, a_{ub(\mathtt{a})} + [A.inf, A.sup]$ mit z.B. $ub(\mathtt{a}) = 2$ in folgender Form einzugeben:

$$a_1 \sqcup a_2 \sqcup [A.inf, A.sup] \quad < \mathtt{enter} >$$

Dabei dürfen sich die a_i sogar überlappen, es muß jedoch $A.inf \le A.sup$ erfüllt sein. `ub(a)` $\neq$ `LStagg_prec` ist zulässig.

Bitte beachten Sie, daß die staggered Variable `a` ihre Komponenten `a[j]` genau so übernimmt, wie der Anwender die entsprechenden Komponenten eingegeben hat. Daher muß also z.B. `a[1]` nicht der Mittelpunkt oder eine Approximation des Mittelpunktes von `a` sein!

procedure `write` (**var** F : `Text`; **var** a : `LStagg`); (lstagg)

Ist `a` das Ergebnis einer Rechnung im *staggered* Format, so entstehen die `a[1]`,`a[2]`,...,`a[ub(a)]` durch das Auslesen des langen Akkumulators zur jeweils nächsten `Longreal`–Zahl, wodurch die `a[i]` mit unterschiedlichen Vorzeichen berechnet werden können. So liefern z.B. mit `LStagg_prec=3` die Anweisungen

```
x := 3;   a := sqrt(x);
For k := -1 to UB(a) do writeln(a[k]);
```

den folgenden Bildschirmausdruck

```
-2.0548066983091199963lE-062        { = a[-1]        }
-2.05480669830911999628E-062        { = a[0 ]        }
 1.7320508075688772935 3E+000       { = a[1 ]        }
-2.5536584941276330571 9E-021       { = a[2 ]        }
-4.7461896193719441930 0E-042       { = a[3 ]        }
```

der den Zahlenwert

$$\sqrt{3} = 1.7320508075688772935274463415058723669428052538103 80\ldots$$

nicht direkt erkennen läßt. Die analogen Anweisungen

```
x := 3;   a := sqrt(x);   write(a);
```

liefern jetzt mit `write(a)` den gut lesbaren Ausdruck für die Einschließung von $\sqrt{3}$

```
 1.73205080756887729352E+000                { <> a[1]          }
 7.44634150587236694280E-021                { <> a[2]          }
 5.25381038062805580695E-042                { <> a[3]          }
[2.94519330169088000369E-062, 2.94519330169088000372E-062]
```

Die Ausgabe von `write(a)` schließt das Intervall `a` garantiert ein, wobei sich die drei ersten Komponenten nicht überlappen und einerlei Vorzeichen besitzen. Die erste Komponente `1.73205...9352E+000` ist eine hochgenaue Approximation des Mittelpunktes von `a`. `ub(a)` $\neq$ `LStagg_prec` ist zulässig.

> Durch die Ausgabe von `write(a)` werden die internen Werte $a[i]$ i.a. nicht korrekt wiedergegeben! $i = -1, 0, 1 \ldots,$ `LStagg_prec`;
> `write(a)` liefert jedoch eine gut lesbare Einschließung von `a`.

3.2.2.10 Programmbeispiele

Im Abschnitt 3.2.1 haben wir zum Basistyp `real` zwei einfache Programme kennengelernt, die den richtigen Umgang der wichtigsten Routinen zeigen. Dem Leser wird empfohlen, die Programme `stagg1` und `stagg2` für den Basistyp `Longreal` entsprechend umzuschreiben.

In diesem Abschnitt stellen wir zwei Programme vor, die den Vorteil einer staggered Arithmetik demonstrieren, wobei nochmals betont wird, daß die exakten Ergebnisse stets *garantiert* eingeschlossen werden.

3.2.2.10.1 Hornerschema

Die Auswertung eines Polynoms

$$p_n(x) := a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0; \qquad n = 0, 1, 2, \ldots$$

kann nach dem Horner–Schema

$$p_n(x) := (\cdots((a_n x + a_{n-1})\, x + a_{n-2})\, x + \cdots + a_1)\, x + a_0$$

erfolgen, wenn die Koeffizienten a_i explizite gegeben sind. Das nachfolgende Programm `lstg1` wertet das Legendre Polynom 5. Ordnung

$$P_5(x) := \frac{63}{8} x^5 - \frac{35}{4} x^3 + \frac{15}{8} x$$

an der Stelle

$$\zeta \;:=\; 0.5384693101056830910363144207002088049672866069055595562022316270594711853677553 \approx x_0; \qquad P_5(x_0) = 0$$

nach dem Hornerschema aus. In der `repeat`–Schleife kann der Anwender jede gewünschte Präzision einstellen.

```
program lstg1(input,output);
{ Auswertung des Legendre-Polynoms 5. Ordnung in der Naehe }
{ der ersten positiven Nullstelle mit staggered Arithmetik }
{ nach dem Hornerschema.                                    }
use type_def, lstagg;
var p : integer;

procedure main;
type Polynom = dynamic array[*] of Lstagg;
```

```
var
   x                    : LStagg[-1..4];
   y                    : LStagg[-1..LStagg_prec];
   pol                  : Polynom[0..5,-1..1];
function Horner( var pol : Polynom; var x: LStagg ) :
                                     Lstagg[-1..LStagg_prec];
var r         : LStagg[-1..LStagg_prec];
    i,degree  : integer;
begin  { Auswertung nach dem Horner-Schema: }
   degree := UB(pol,1);
   r := change_length( pol[degree] );
   For i := degree-1 downto 0 do
       r := r*x + pol[i];
   Horner := r;
end; { Horner }

begin
   { Punktintervalle der Polynomkoeffizienten von P_5(x): }
   pol[0] := 0;       pol[1] :=  Long(15)/8;
   pol[2] := 0;       pol[3] := -Long(35)/4;
   pol[4] := 0;       pol[5] :=  Long(63)/8;
   { Argument x : LStagg[-1..4]; }
   x[-1] := 0;   x[0] := 0;
   x[1]  := LVAL('5.38469310105683091036E-001');
   x[2]  := LVAL('3.14420700208804967286E-022');
   x[3]  := LVAL('6.06905559956202231627E-043');
   x[4]  := LVAL('5.94711853677553000000E-065');
   y := Horner(pol,x);
   writeln(y);
end; { main }

begin
   repeat
      writeln('Neue Praezision = ?');  read(p);
      set_Lstagg_prec(p);
      main;
   until false
end.
```

Da ζ sehr dicht an der ersten positiven Nullstelle x_0 von $P_5(x)$ liegt, ist starke Auslöschung zu erwarten, so daß man aussagekräftige Ergebnisse erst bei Rechnungen in höherer Präzision erhalten wird!

Tatsächlich erhält man bis $p = 2$, d.h. `LStagg_prec=2`, für $P_5(\zeta)$ so grobe Einschließungen, daß noch nichteinmal das richtige Vorzeichen zu erkennen ist. Andererseits erhält man mit `LStagg_prec=15` sogar eine punktförmige Einschließung des exakten Wertes:

$$P_5(\zeta) \quad := \quad -2.17461456028 \ldots 018096430125 \cdot 10^{-80}$$

Aus Platzgründen sind von den insgesamt 319 Dezimalziffern nur die jeweils ersten und letzten 12 Ziffern angegeben. Beachten Sie bitte, daß man mit dem Basistyp `real` wegen des kleineren Exponentenbereichs eine punktförmige Einschließung nicht berechnen kann. Schreiben Sie das Programm `lstg1` auf den Basis–Typ `real` um und bestätigen Sie diese Vermutung.

3.2.2.10.2 Logistik–Gleichung

Wir betrachten ein einfaches dynamisches System, das durch die folgende Logistik–Gleichung gegeben ist:

$$(3.2) \qquad x_{n+1} = a \cdot x_n \cdot (1 - x_n); \quad n \geq 0, \quad a \in [0,4], \quad x_0 \in (0,1);$$

Es ist bekannt, daß man schon nach etwa 100 Schritten i.a. völlig falsche Ergebnisse erhält, wenn die rechte Seite von (3.2) z.B. im Longreal–Format ausgewertet wird. Aber auch mit der normalen Intervallrechnung erhält man ebenfalls sehr schnell viel zu große Überschätzungen, so daß nur die staggered Arithmetik auch für größere n noch brauchbare numerische Ergebnisse der x_n liefern wird. Dabei ist es wichtig, wirklich garantierte Einschließungen der x_n zu berechnen, da sonst auch bei großer Präzision keine gesicherte Aussage über die Genauigkeit der erhaltenen Ergebnisse möglich ist. Bezeichnen wir mit X_n eine Einschließung von $x_n \in X_n$, so lautet (3.2) in Intervallform:

$$(3.3) \quad X_{n+1} = a \cdot X_n \cdot (1 - X_n); \quad n \geq 0, \quad a \in [0,4], \quad x_0 \in X_0 \subset (0,1),$$

wobei die arithmetischen Operatoren in (3.3) jetzt als Intervall–Operatoren zu interpretieren sind.

Wertet man die rechte Seite von (3.3) intervallmäßig aus, so sind die in der Intervallrechnung bekannten Intervallaufblähungen zunächst unvermeidbar. Man kann diese Überschätzung i.a. jedoch wesentlich reduzieren, wenn die rechte Seite von (3.3) auf die *Mittelwertform* [10, S. 107] gebracht wird. Dazu existiert der folgende Satz:

Satz 1.

Vorauss.:

$f : X := [x_1, x_2] \longrightarrow I\!R$ ist differenzierbar in X;
$f'(X)$ entsteht aus $f(x)$, wenn man dort x durch X und die arithmetischen Operatoren durch entsprechende Intervall–Operatoren ersetzt.
Wertebereich von f : $W_f := \{r \mid r = f(x) \ \wedge \ x \in X\}$

Beh.: $W_f \subseteq f(y) + f'(X) \cdot (X - y), \quad \text{mit} \quad y \in X;$

Der Beweis ergibt sich unmittelbar aus dem Mittelwertsatz der Differentialrechnung, wenn man den rechten Randpunkt und den Zwischenpunkt durch das betrachtete Intervall ersetzt. Da $y \in X$ ein beliebiger Wert ist, wird man y in der Praxis oft als Intervallmittelpunkt wählen.

Wendet man nun Satz 1 an auf die Logistik–Gleichung (3.2), so erhält man unmittelbar die folgende Intervalldarstellung:

$$
\begin{aligned}
(3.4) \qquad x_{n+1} \in X_{n+1} := a \cdot (y_n(1 - y_n) + (1 - 2X_n) \cdot (X_n - y_n), \quad n \geq 0 \\
y_n = \text{Mittelpunkt von } X_n, \quad x_0 \in X_0;
\end{aligned}
$$

Die Erfahrung zeigt nun, daß man mit (3.4) deutlich engere Einschließungen für x_{n+1} berechnet als mit der naiven Intervalldarstellung (3.3).

Das folgende **PASCAL-XSC** Programm `lstg2` berechnet daher nach (3.4) garantierte Einschließungen der x_n in staggered Arithmetik, wobei in der `repeat`–Schleife die Präzision nach Bedarf gewählt werden kann. Zum Vergleich werden die x_n nach (3.2) in Gleitpunktarithmetik approximiert und zusätzlich mit Hilfe einer normalen Intervallarithmetik, ebenfalls nach (3.4), eingeschlossen. Die entsprechenden Ergebnisse sind in nachfolgender Tabelle auf Seite 116 zusammengestellt. Zu jeder Iterationsschrittzahl n sind die entsprechenden Gleitpunkt-, Intervall- und staggered– Ergebnisse des Programms `lstg2` in dieser Reihenfolge untereinandergeschrieben.

```
program lstg2(input,output);
use type_def, lstagg, li_ari;
var p :  integer;
procedure main;
var a,x0  : Longreal;
   ym,x   : LStagg[-1..LStagg_prec];
   yim,xi : Linterval;
   n,i    : integer;
begin
   writeln('Auswertung der Logistik-Gleichung');
   writeln('  x(n+1) := a * x(n) * ( 1-x(n) )');
   writeln('in Gleitpunkt-, Intervall- und   ');
   writeln('staggered Arithmetik.            '); writeln;
   writeln('Parameter a  = ?');  read(a);
   writeln('Startwert x0 = ?');  read(x0);
   writeln('Zahl der Iterationen = ?');  read(n);
   xi := x0;  { Startwert bei Intervall Arithmetik }
   x  := x0;  { Startwert bei staggered Arithmetik }
   For i := 1 to n do
   begin
      x0 := a*x0*(1-x0);     { Gleitpunkt-Rechnung: }
      yim := mid(xi);        { Intervall -Rechnung: }
      xi := a*( yim*(1.0-yim) + (1-2*xi)*(xi-yim) );
      ym := mid(x);          { staggered Arithmetik }
      x := a*( ym*(1.0-ym) + (1.0-2.0*x)*(x-ym) );
   end;
   writeln('Einfache Gleitpunkt-Rechnung:');
   writeln(n,' : ',x0);      { Gleitpunkt-Rechnung  } writeln;
   writeln('Einfache Interval-Rechnung:');
   writeln(n,' : ',xi);      { Intervall -Rechnung  } writeln;
   writeln('staggered Arithmetik:');
   writeln(n,' : ',ival(x)); { staggered Arithmetik } writeln;
end; { main }
begin
   repeat
      writeln('Neue Praezision = ?');  read(p);
      set_LStagg_prec(p);
      main;
   until false;
end.
```

n	x_n	LStagg_prec
1	0.93750000000000000000 $0.93750000000000000000^{0}_{0}$ $0.93750000000000000000^{0}_{0}$	4
20	$0.82597097871077541897 7$ $0.825970978710775 4^{34867}_{09550}$ $0.8259709787107754208 3^{9}_{8}$	4
40	$0.41634931695765819532 7$ $0.41634931695 7^{772380647}_{465730156}$ $0.4163493169576356550 3^{9}_{8}$	4
60	$0.79908633407322359588 8$ $0.7990863 3^{6080555299545}_{2882318267576}$ $0.7990863343083094578 8^{5}_{4}$	4
80	$0.85617817023522208409 0$ $0.8561 7^{9049716684935862}_{6687840282034500}$ $0.8561779966293365551 2^{7}_{6}$	4
100	$0.88704211011665259658 6$ $0. 8^{99009321458456586914}_{79405642724533587715}$ $0.8882939922840348567 7^{8}_{7}$	4
500	$0.93234399331994487345 8$ Overflow $0.2767538774320484926 8^{6}_{5}$	4
600	$0.88777668687670260600 3$ Overflow $0.6544 6^{7821383831840447}_{6803636422783831}$	4
700	$0.66230823805678348038 7$ Overflow $0.2027624200919638622^{7}_{6}$	6
1000	$0.66700055528110198089 5$ Overflow $0.7917467409224436376 8^{8}_{6}$	8
1400	$0.64886665773497904916 9$ Overflow $0.93104219807 2^{186276122}_{004213812}$	11

Auswertung der Logistik-Gleichung (3.2) mit $a = 3.75$, $x_0 = 0.5$

Aus der umseitigen Tabelle erkennt man, daß für $n > 100$ die Gleitpunkt–Arithmetik und die Intervallrechnung im einfachen Longrealformat völlig versagen, während die staggered correction Arithmetik noch bis $n = 600$ brauchbare Ergebnisse liefert. Erhöht man die Präzision der Rechnung bis `LStagg_prec=11`, so können sogar bis zu 1400 Iterationsschritte erfolgreich durchgeführt werden.

Die Logistikgleichung (3.2) demonstriert damit in fast idealer Weise die Gefahr einer blinden Gleitpunktrechnung, die für mehr als 110 Iterationen wirklich nur noch numerischen Schrott liefert. Für den naiven Numeriker sollte spätestens dieses sehr einfache Beispiel einen heilsamen Schock auslösen und ihn von der **PASCAL-XSC** Philosophie überzeugen, die stets garantierte Einschließungen der exakten Ergebnisse liefert. Die Tabelle auf Seite 116 zeigt außerdem die unvermeidbaren und mit n stark zunehmenden Intervallaufblähungen bei naiver Anwendung der Intervall–Arithmetik. In selbstverifizierenden Algorithmen sollte man daher Intervalloperationen gezielt nur dort einsetzen, wo diese zur Ergebniseinschließung unvermeidbar sind! So wird man etwa zur Einschließung des Lösungsvektors eines linearen Gleichungssystems eine Approximation der inversen Koeffizientenmatrix nur mit naiver Gleitkomma–Arithmetik durchführen und erst im Verifikationsschritt die Intervallrechnung einsetzen.

3.2.2.10.3 Arcsin Funktion

Bezeichnungen in diesem Abschnitt:

s	staggered Intervall–Argument;	$-1 \le s.inf \le s.sup \le +1$
$y, y1, y2$	staggered Intervalle;	$y, y1, y2$: `LStagg`
x	nicht-negatives Argument;	$0 \le x.inf \le x.sup \le +1$
r	reelles Argument	$r \in I\!R$
$\diamond(\alpha)$	Auswertung des reellen Ausdrucks α im staggered Format	

Am Beispiel der reellen arcsin Funktion wird die grundsätzliche Vorgehensweise erklärt, die bei der Berechnung einer Einschließung des Wertebereichs einer Funktion zur Anwendung kommt, wenn ihr staggered Argument

$$s = [s.inf, s.sup], \qquad \text{mit} \quad -1 \le s.inf \le s.sup \le +1$$

als Intervall vom Typ `LStagg` vorgegeben ist.

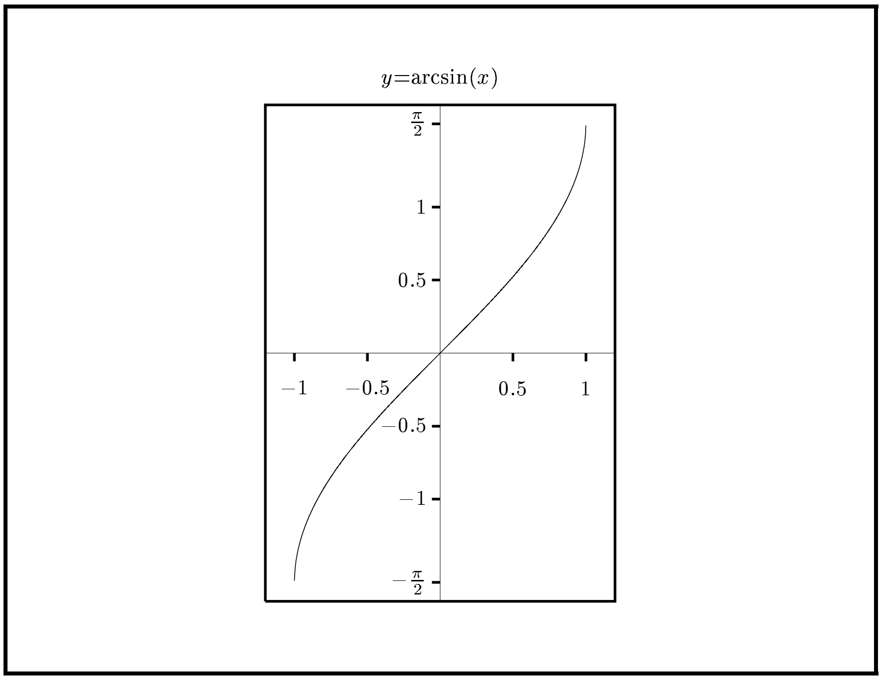


Abbildung 3.1: Graph der arcsin–Funktion

Argumentreduktion

Wegen $\arcsin(r) \equiv -\arcsin(-r)$ kann man sich auf $x \geq 0$ beschränken. Liefert daher `arcsin_ge0`(x) eine Einschließung von $\arcsin(r)$, mit $r \in x$, so erhält man eine Einschließung von $\arcsin(s)$ nach folgendem Algorithmus:

```
if s >= 0 then p := 1 else
  if s <= 0 then p := -1 else p := 0;
case p of
+1: arcsin :=  arcsin_ge0(s);
-1: arcsin := -arcsin_ge0(-s);
0 : begin  { 0 liegt im Innern von s }
      up := hull(0,sup(s));
      up := arcsin_ge(up);
      lo := -hull(inf(s),0);
      lo := -arcsin_ge(lo);
      arcsin := hull( inf(lo),sup(up) );
    end;
end; { case }
```

Die eigentliche Aufgabe besteht also darin, zu einem vorgegebenen, nichtnegativen staggered Argument x

$$0 \leq \inf(x) \leq \sup(x) \leq +1 \tag{3.5}$$

eine möglichst optimale Einschließung von $\arcsin(x)$ zu berechnen. Nach (3.5) ist der Durchmesser des Argumentintervalls im ungünstigsten Fall jedoch gleich 1 und damit für eine Taylor–Approximation viel zu breit, so daß eine weitere Argumentreduktion auf ein staggered Intervall t erforderlich ist, das folgende Eigenschaft haben sollte:

$$0 \leq \inf(t) \leq \sup(t) \ll +1$$

Dabei ist jedoch zu beachten, daß eine solche Argumentreduktion ohne die Speicherung zusätzlicher Daten durchgeführt werden muß, da solche Daten erfahrungsgemäß mit großem Aufwand in der gleichen Präzision bereitgestellt werden müssen, in der die staggered Arithmetik durchgeführt werden soll. Man findet die gewünschte Argumentreduktion auf ein solches Intervall t mit sogar beliebig kleinem Durchmessers nach folgenden Überlegungen:

Mit Hilfe der in $r \in [-1, 1]$ gültigen Beziehungen:

$$\arcsin(r) = \frac{\pi}{2} - \arcsin\sqrt{1-r^2} = \frac{\pi}{2} - 2 \cdot \arcsin\sqrt{\frac{1-r}{2}}$$

erhält man unmittelbar die folgende Identität:

$$\arcsin(r) = 2 \cdot \arcsin\sqrt{0.5 \cdot (1 - \sqrt{1-r^2})}, \quad r \in [0, +1] \tag{3.6}$$

Setzt man anschließend $u(r) := \sqrt{0.5 \cdot (1 - \sqrt{1-r^2})}$ und wendet man in $\arcsin(r) = 2 \cdot \arcsin(u)$ auf $\arcsin(u)$ die Gleichung (3.6) an, so folgt:

$$\arcsin(r) = 4 \cdot \arcsin\sqrt{0.5 - \sqrt{0.125(1 + \sqrt{1-r^2}\ )}}$$

Diese Schritte lassen sich beliebig oft wiederholen und liefern das Ergebnis:

$$g_1(r) := \sqrt{0.125 \cdot (1 + \sqrt{1-r^2})}, \qquad z(n) := 2^{n+1}$$

$$\text{Iteration:} \quad g_{k+1}(r) = \sqrt{0.125 \cdot (1 + \sqrt{0.5 + g_k(r)}\ )}, \ k = 1, 2, \ldots, z(n) - 1$$

$$t_n(r) := \sqrt{\frac{1}{2} - g_{z(n)}(r)} \tag{3.7}$$

$$\arcsin(r) = 4^{z(n)} \cdot \arcsin(\, t_n(r)\,), \quad r \in [0, 1], \quad n = 0, 1, 2, \ldots \tag{3.8}$$

Bezüglich der $g_k(r)$ gilt folgendes:

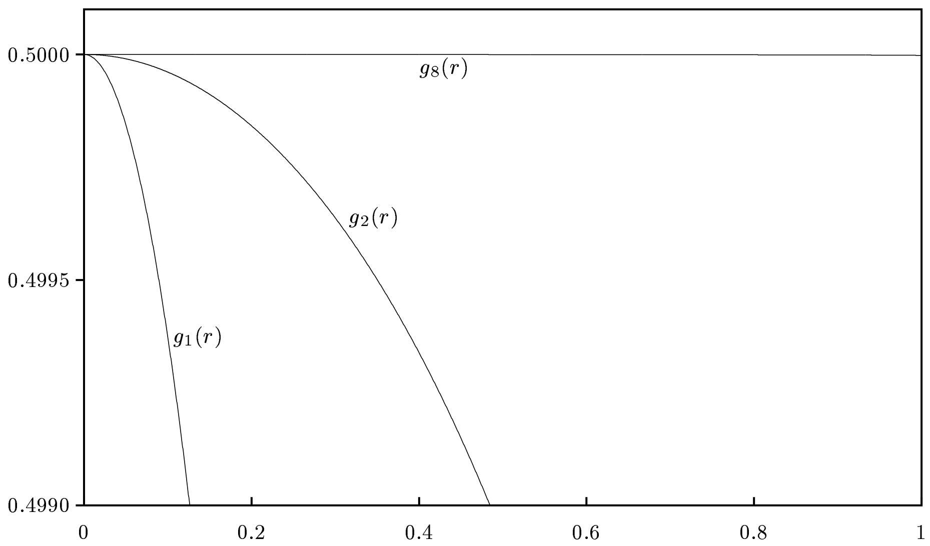
1. Mit $g_1(r)$ sind auch alle $g_k(r)$ monoton fallend in $[0,1]$, da die Wurzel aus einer positiven und monoton fallenden Funktion selbst wieder eine monoton fallende Funktion ist.

2. $g_k(r) \leq g_k(0) = 1/2, \quad r \in [0,+1], \quad k = 1,2,\ldots$

3. $g_{k+1}(r) \geq g_k(r), \quad r \in [0,+1];$ der Beweis folgt aus der der äquivalenten Aussage $\sqrt{1/2 + g_k} \geq 8g_k^2 - 1$ mit Hilfe von $g_k(r) \leq 1/2$.

Die Graphen der Funktionen $g_1(r), g_2(r)$ und $g_8(r)$ findet man in folgender Abbildung:

$g_1(r),\ g_2(r),\ g_8(r)$



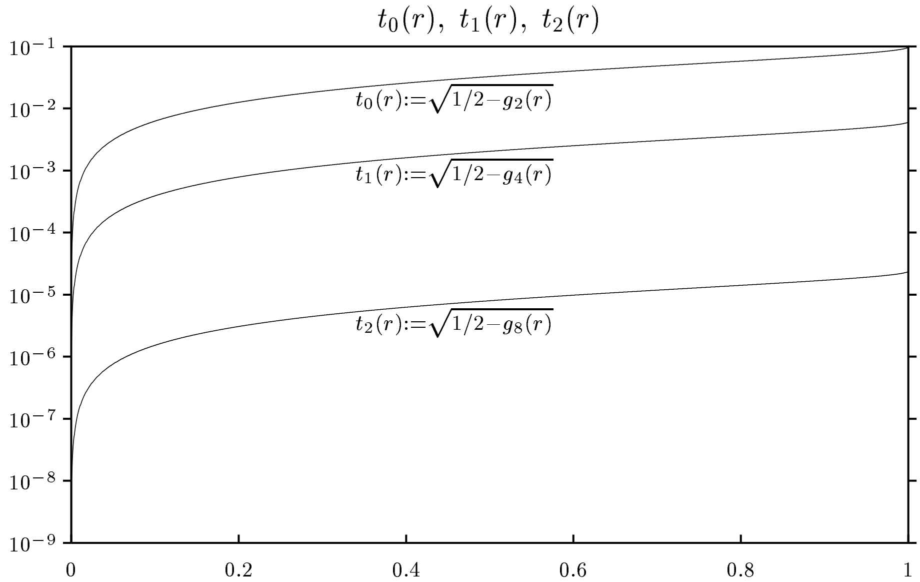
Wegen $g_8(1) \approx 0.5$ erscheint in obiger Abbildung die monoton fallende Funktion $g_8(r)$ als konstante Funktion. Die oben angegebenen Ergebnisse für die $g_k(r)$ lassen sich auf Grund der Beziehung (3.7) direkt auf die $t_n(r)$ übertragen:

1. $t_n(r) := \sqrt{1/2 - g_{z(n)}}$ ist monoton wachsend in $[0,1]$

2. $0 \leq t_n(r) \leq t_n(1), \quad n = 0,1,2,\ldots$

3. $t_{n+1}(r) \leq t_n(r), \quad x \in [0,1]$

Die Graphen von $t_0(r), t_1(r), t_2(r)$ findet man für $r \in [0, +1]$ in folgender Abbildung:



Die monotonen Funktionen $t_n(r)$ liefern nach (3.8) eine Abbildung des Intervalls $x \subseteq [0, 1]$ nach $t_n(x) \subseteq [0, t_n(1)]$, und wegen der Beziehung $t_{n+1}(1) < t_n(1)$ lassen sich damit für hinreichend großes n hinreichend kleine Intervalle $t_n(x)$ angeben, in denen zur Berechnung der arcsin–Funktion nur noch wenige Intervall–Summanden der Taylorreihe zu addieren sind.

In nachfolgender Tabelle sind für `LStagg_prec`=7 in Abhängigkeit von n die Oberschranken von $t_n(1)$, die Anzahl ($\sqrt{\cdots}$) der notwendigen Quadratwurzel–Berechnungen, die Anzahl (Taylor) der benötigten Taylor–Summanden und die jeweilige Laufzeit in Millisekunden zusammengestellt:

`LStagg_prec=7`, d.h. 168 interne Dezimalziffern				
n	$t_n(1) <$	Taylor	$\sqrt{\cdots}$	Laufzeit(ms)
0	$9.802 \cdot 10^{-2}$	82	4	1250
1	$6.136 \cdot 10^{-3}$	38	8	610
2	$2.397 \cdot 10^{-5}$	18	16	370
3	$3.658 \cdot 10^{-10}$	9	32	350
4	$8.516 \cdot 10^{-20}$	5	64	490

Für $n = 2$, d.h. mit $t_2(x) := \sqrt{1/2 - g8(x)}$ erhält man also nach obiger Tabelle fast die günstigste Laufzeit, während für $n = 3$ wegen der dann notwendigen 32 Quadratwurzelberechnungen die Intervallüberschätzungen deutlich größer ausfallen. Wir wählen $n = 2$, und die Argumentreduktion für nicht-negative Argumente ist damit durch folgenden Algorithmus gegeben:

$$
\begin{array}{l}
\texttt{if } x > 10^{-2} \texttt{ then} \\
\texttt{begin} \\
\quad x \;:=\; \sqrt{(1+\sqrt{1-x^2}\;)/8}\texttt{;} \\
\quad \texttt{For } k := 1 \texttt{ to } 7 \texttt{ do } x \;:=\; \sqrt{\left(1+\sqrt{\frac{1}{2}+x}\;\right)/8}\texttt{;} \\
\quad x \;:=\; \sqrt{\frac{1}{2}-x}\texttt{;} \qquad \texttt{\{} \longrightarrow \; x \subseteq [0, 2.397\cdot 10^{-5}] \;\texttt{\}} \\
\texttt{end;} \qquad\qquad\qquad\qquad \texttt{\{} \longrightarrow \; x \subseteq [0, 10^{-2}] \;\texttt{\}}
\end{array}
$$

Im Fall $0 \le x \le 10^{-2}$ wird also die zusätzliche Argumentreduktion mit Hilfe von $t_2(x)$ nicht durchgeführt. Da dann die Wurzelfunktionen nicht ausgewertet werden, erfordert bei `LStagg_prec`=7 die Addition der maximal 42 Taylor-Summanden etwa die gleiche Laufzeit.

Reihenentwicklung: Es gilt: $\arcsin(r) = \sum_{k=0}^{\infty} q_k \cdot r^{2k+1}, \quad |r| \le 1$

Die Koeffizienten q_k sind gegeben durch:

$$
\begin{aligned}
q_0 &:= 1 \\
q_1 &:= \frac{1}{2\cdot 3} \\
q_2 &:= \frac{1\cdot 3}{2\cdot 4\cdot 5} \\
q_3 &:= \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 7} \\
q_k &:= \frac{(2k)!}{(2^k k!)\cdot(2k+1)} = \frac{1}{2k+1}\prod_{j=1}^{k}\frac{2j-1}{2j}, \qquad k = 1, 2, \ldots \\
q_{k+1} &:= q_k \cdot \frac{(2k+1)\cdot(2k+1)}{(2k+2)\cdot(2k+3)}, \qquad k = 0, 1, 2, \ldots
\end{aligned}
$$

Im Bereich $0 \le r \le +1$ gilt:

$$
\begin{aligned}
&(3.9) \quad S_n = S_n(r) := \sum_{k=0}^{n} q_k \cdot r^{2k+1} \le \arcsin(r), \qquad n = 0, 1, \ldots \\
&(3.10) \quad S_n = S_{n-1} + q_n \cdot r^{2n+1}, \qquad n = 1, 2, \ldots
\end{aligned}
$$

Approximationsfehler

Im Bereich $0 < r < +1$ gilt für den relativen Approximationsfehler

$$
\begin{aligned}
\varepsilon_n(r) \quad &:= \quad \left| \frac{\arcsin(r) - r \cdot \sum_{k=0}^{n} q_k \cdot r^{2k}}{\arcsin(r)} \right| = \frac{\arcsin(r) - S_n(r)}{\arcsin(r)} \\
&= \quad \left| \frac{r}{\arcsin(r)} \right| \sum_{k=n+1}^{\infty} q_k \cdot r^{2k} \leq \sum_{k=n+1}^{\infty} q_k \cdot r^{2k} \\
&= \quad q_{n+1} \cdot r^{2n+2} \sum_{k=n+1}^{\infty} \frac{q_k}{q_{n+1}} \cdot \frac{r^{2k}}{r^{2n+2}} \\
&= \quad q_{n+1} \cdot r^{2n+2} \left\{ 1 + \underbrace{\frac{q_{n+2}}{q_{n+1}}}_{\in(0,1)} \cdot r^2 + \underbrace{\frac{q_{n+3}}{q_{n+1}}}_{\in(0,1)} \cdot r^4 + \ \ldots \right\} \\
&\leq \quad q_{n+1} \cdot r^{2n+2} \sum_{k=0}^{\infty} x^2 = q_{n+1} \cdot r^{2n+2} \cdot \frac{1}{1 - r^2}
\end{aligned}
$$

und für den Bereich $0 < r \leq 10^{-2}$ des reduzierten Arguments erhält man:

$$\varepsilon_n(r) \leq 1.00011 \cdot q_{n+1} \cdot r^{2n+2} \ =: \ A_n(r) \tag{3.11}$$

Mit (3.9) erhält man dann für $0 \leq r \leq 10^{-2}$ die Abschätzung:

$$S_n(r) \leq \arcsin(r) = \frac{S_n(r)}{1 - \varepsilon_n(r)} \leq \frac{S_n(r)}{1 - A_n(r)} \ =: \ \alpha_n(r) \tag{3.12}$$

Bevor wir $\alpha_n(r)$ weiter nach oben abschätzen, um die aufwendige Division zu vermeiden, benötigen wir zunächst eine Oberschranke von $A_n(r)$, die für jeden Wert von `LStagg_prec` ≥ 1 gelten soll. Da im staggered Format mit mindestens[3] $21 \cdot ($`LStagg_prec` $+ 1)$ Dezimalstellen gerechnet wird, ist es sinnvoll, für den Approximationsfehler folgende Oberschranke zu verlangen:

$$A_n(r) \leq 10^{-21 \cdot (\texttt{LStagg\_prec}+1)} \leq 10^{-42} \tag{3.13}$$

Bei der Implementierung der Funktion `arcsin_ge0` wird also die Abschätzung $A_n(r) \leq 10^{-42}$ garantiert, d.h. zu gegebenem r wird n in einer `repeat`

[3] Es wird vorausgesetzt, daß sich die staggered Komponenten nicht überlappen.

Schleife solange erhöht, bis die erste Bedingung in (3.13) erfüllt ist. Unter dieser Voraussetzung gilt dann mit $\delta := 1 + 10^{-20}$ die Abschätzung[4]:

$$\frac{1}{1 - A_n(r)} \leq 1 + \frac{A_n(r)}{1 - 10^{-42}} \leq 1 + \delta \cdot A_n(r)$$

Zusammen mit der Abschätzung (3.12) erhält man damit für $\arcsin(r)$ im Bereich $0 \leq r \leq 10^{-2}$ des reduzierten Arguments die folgende Einschließung:

$$\begin{aligned} & 0 \leq r \leq 10^{-2}, \qquad \delta := 1 + 10^{-20} \\ & A_n(r) := 1.00011 \cdot q_{n+1} \cdot r^{2n+2} \leq 10^{-42} \\ \Longrightarrow \quad & S_n(r) \leq \arcsin(r) \leq S_n(r) \cdot [1 + \delta \cdot A_n(r)] \end{aligned} \tag{3.14}$$

Damit haben wir eine Einschließung von $\arcsin(r)$ gefunden, in der die Oberschranke ohne zeitaufwendige Division berechnet werden kann.

Einschließung von $\arcsin(x)$ auf dem Rechner:
Mit Hilfe von (3.14) sind wir nun in der Lage, für ein staggered Intervall–Argument x, das durch eine mögliche Argumentreduktion beschränkt ist auf

$$x \subseteq [0, 10^{-2}]$$

den Wertebereich von $\arcsin(x)$ durch Anwendung der staggered Arithmetik auf dem Rechner einzuschließen. Bezeichnet man mit $\Diamond S_n(x)$ das Ergebnis der Auswertung von $S_n(x)$ nach (3.10) im staggered Format und ist e eine Variable vom Typ `LStagg`, die das Intervall $[1, 1 + \delta \cdot A_n(x)]$ einschließt, so gelten mit dem Multiplikationsoperator $\diamond\!\!\!\cdot$ für staggered Operanden folgende Aussagen:

$$\begin{aligned} & S_n(x) \in (\Diamond S_n(x)) \mathbin{\diamond\!\!\!\cdot} e, \quad S_n(x) \cdot [1 + \delta \cdot A_n(x)] \in (\Diamond S_n(x)) \mathbin{\diamond\!\!\!\cdot} e \\ \Longrightarrow \quad & \arcsin(x) \in (\Diamond S_n(x)) \mathbin{\diamond\!\!\!\cdot} e \end{aligned} \tag{3.15}$$

Damit sind alle Schritte skizziert, die bei der Implementierung der `arcsin` Funktion besonders zu beachten sind. Da die Berechnungen von z.B. $\Diamond S_n(x)$ im staggered Format reine Intervallrechnungen sind, wird die Abschätzung irgendeines Auswertefehlers überflüssig, so daß die Realisierung der arcsin–Funktion im staggered Format sehr viel einfacher ist als für die Datentypen real oder Longreal.

Auf der folgenden Seite findet man den vollständigen **PASCAL-XSC** Quell–Text mit der Implementierung und dem Aufruf der arcsin–Funktion

[4] $(1 + 10^{-20})$ ist im 21-stelligen Longreal–Format exakt darstellbar.

für staggered Intervall–Argumente. In einer `repeat` Schleife kann der Anwender die Präzision der Rechnung und das staggered Argument vorgeben. Bei zu großen `LStagg_prec` Werten kommt es zu einer Fehlermeldung, wenn bei der Addition von Longreal–Produkten der Exponentenbereich des nur für den real–Bereich ausgelegten Akkumulators überschritten wird.

```
program lstg3(input,output);
use lstagg, li_ari, type_def;
var p : integer;

procedure main;
var y,x : LStagg[-1..LStagg_prec];

function arcsin_ge0(var x : Lstagg) :
                           LStagg[-1..LStagg_prec];
{ Berechnet arcsin(x), falls x >= 0; Diese Bedingung }
{ wird in dieser Funktion jedoch nicht ueberprueft!  }

const c = 1.00011;  { 1/(1-x^2) < c, da 0 <= x.sup <= 0.01 }
var t,tt,tpot,s,q_n   : LStagg[-1..LStagg_prec];
    e                 : LStagg[-1..1];
    RelErr,delta      : Longreal;
    reduc             : boolean;
    it,A_n,iq_n,itpot : Linterval;
    n                 : integer;
begin
   RelErr := comp(Long(0.1),-21*(Lstagg_prec+1)+1);
   t := x;  { x bleibt erhalten }
   reduc := SUP(t) > 1e-2;
   if reduc then
   begin  { Argumentreduktion auf: [0,2.397E-5] }
      s := sqrt(0.125 * (1.0 + sqrt(1.0-t*t)));
      for n := 1 to 7 do
               s := sqrt(0.125*(1.0+sqrt(0.5+s)));
      t := sqrt(0.5-s); { --> t in [0,2.397E-5] }
   end;
   { Es gilt stets:   0 <= t.sup <= 0.01 }
   tt := t * t;  tpot := t;
   it := t;
   s := tpot;  { s=q_0*r }
```

```
   n := 1;
   q_n := 1; q_n := q_n/Long(6); { = q_1 = 1/(2*3) }
   delta := Long(1) + 1e-20;

   repeat
      tpot := tpot * tt;
      s := s + q_n * tpot;
      e := (2*n+2)*(2*n+3);
      t := (2*n+1)*(2*n+1);
      q_n := q_n*t/e;
      n := n + 1;
      iq_n := q_n;
      itpot := tpot;
      A_n := c * iq_n * itpot * it;
   until A_n.sup < RelErr;

   if reduc then s := 65536.0 * s;
   e[1] := 1;  e[-1] := 0;  e[0] := delta *> A_n.sup;
   s := s * e;  { ---> arcsin(x) in s }
   arcsin_ge0 := s;
end; { arcsin_ge0 }

function arcsin(x : LStagg) : LStagg[-1..LStagg_prec];
var p : integer;
begin { arcsin }
   if x >= 0.0 then p := 1 else
      if x <= 0.0 then p := -1 else p := 0;
   case p of
     +1: arcsin := arcsin_ge0(x);
     -1: arcsin := -arcsin_ge0(-x);
     0 : arcsin := arcsin( ival(x) );
   end; { case }
end; { arcsin }

begin { main }
   writeln('x = ?');  read(x);
   y := arcsin(x);
   writeln('arcsin(x) = ',y);
end;  { main }
```

```
BEGIN { lstg3 }
  repeat
    writeln('Der im Modul type_def definierte');
    writeln('LStagg_prec-Wert ist: ',get_LStagg_prec);
    writeln('Neuer LStagg_prec Wert = ?');  read(p);
    set_LStagg_prec(p);
    main;
  until false;
END.  { lstg3 }
```

Numerische Ergebnisse:

`LStagg_prec` $= 3$, $x = [0.75, 0.75]$;
$\arcsin(x) \subset 0.848062078981481008052944338998418080073366213263112642860718163570200821 2^{\mathbf{36396915629}}_{\mathbf{13606894272}}$

`LStagg_prec` $= 4$, $x = [0.96875, 0.96875]$;
$\arcsin(x) \subset 1.320140664458765825482187718925582006002289246010208160019544234180921324515416560545293031085 34^{\mathbf{933790103}}_{\mathbf{462044713}}$

Hinweis:

Mit Hilfe der oben definierten arcsin–Funktion kann die arccos–Funktion für staggered Intervall–Argumente $x \subseteq [-1, +1]$ wie folgt realisiert werden:

$$\arccos(x) := \begin{cases} \arcsin\left(\sqrt{1-x^2}\right) & \text{falls } x \geq 0 \\ \frac{\pi}{2} - \arcsin(x) & \text{sonst} \end{cases}$$

3.2.2.10.4 Kubikwurzel

Die Kubikwurzel $\sqrt[3]{a}$ wird definiert als Lösung der Gleichung:

$$x^3 - a = 0, \quad x, a \in I\!R \qquad \Longleftrightarrow \qquad x = \sqrt[3]{a}$$

Die Berechnung von $\sqrt[3]{a}$ läßt sich damit zurückführen auf die Berechnung der Nullstelle der Funktion $f(x) = x^3 - a$. Zur Berechnung einer garantierten Einschließung dieser Nullstelle wenden wir das Intervall–Newton–Verfahren an [10, Seite 48,109].

Das Intervall–Newton–Verfahren:

Ist $[x]^{(0)}$ ein Startintervall, das die gesucht Nullstelle x^* enthält und setzt man voraus, daß $0 \notin f'([x^{(0)}])$, dann lautet das Intervall–Newton–Verfahren:

$$[x]^{(k+1)} := \left(\check{x}^{(k)} - \frac{f(\check{x}^{(k)})}{f'([x]^{(k)})} \right) \cap [x]^{(k)}, \quad k = 0, 1, 2, \ldots$$

und liefert mit $[x]^{(k+1)}$ eine garantierte und deutlich verbesserte Einschließung der gesuchten Nullstelle x^*, wobei der Mittelpunkt $\check{x}^{(k)}$ von $[x]^{(k)}$ auch durch einen anderen Vertreter des Intervalls $[x]^{(k)}$ ersetzt werden kann.

Hinweise:

- Der Beweis folgt unmittelbar aus dem Mittelwertsatz der Differentialrechnung und kann in [10, Seite 110] nachgelesen werden.
- Man kann zeigen, daß für den Fall, daß x^* nicht im Anfangsintervall $[x]^{(0)}$ liegt, irgendwann einmal eine leere Schnittmenge auftreten muß. Damit ist dann die Nichtexistenz einer Nullstelle von $f(x)$ in $[x]^{(0)}$ nachgewiesen.
- Um $\sqrt[3]{a} \in [x]^{(0)}$ garantieren zu können, wird das staggered Argument a zunächst durch `v := ival(a)` eingeschlossen, und mit Hilfe der Anweisung `x := power(v,Lintval(1)/3 )` erhält man $\sqrt[3]{a} \subseteq$ `x`, wobei `x` eine staggered Variable ist, die das Startintervall $[x]^{(0)}$ darstellt.

Das nachfolgende Programm `lstg4` zeigt eine einfache Version zur Berechnung der Kubikwurzel aus einem staggered Argument a, mit $inf(a) \geq 0$. Zur Unterscheidung von der Kubikwurzel `sqrt3`, die das Modul `lstagg` exportiert, wird die Kubikwurzelfunktion in diesem Programm mit `sqrt3_` bezeichnet.

```
program lstg4(input,output);
use lstagg, type_def, li_ari;
var p : integer;
procedure main;
var a : LStagg[-1..LStagg_prec];

function sqrt3_(var a : LStagg) : LStagg[-1..LStagg_prec];
{ Berechnung der Kubikwurzel fuer inf(a) >= 0 }
var v    : Linterval;
   t,m   : LStagg[-1..LStagg_prec];
   k,ex  : integer;
   ready : boolean;
begin
   v := ival(a); { v: Einschliessung des Arguments a }
   if v=0 then t := 0 else
   begin
      v := power(v,Lintval(1)/3);
      { v:  Einschliessung der gesuchten Kubikwurzel }
      t := v;  k := 0;  ready := false;
      repeat { Startintervall t enthaelt Nullstelle. }
         ex := expo(t[-1]);   k := k+1;
         m := mid(t);
         t := (m - (m*m*m-a)/(3.0*t*t)) ** t;
         if k >= 2 then ready := ex=expo(t[-1]);
      until ready;
   end;
   sqrt3_ := t;
end; { sqrt3_ }

begin
   writeln('a = ?');  read(a);
   writeln('sqrt3_(a) = ',sqrt3_(a));
end; { main }
begin
   repeat
      writeln('LStagg_prec = ?');  read(p);
      set_LStagg_prec(p);
      main;
   until false;
end.
```

Anmerkungen zum Programm `lstg4`:

- Eine deutliche Verbesserung der Laufzeit wird erreicht, wenn man die ersten Newton–Schritte in entsprechend kleinerer Präzision und erst den letzten Schritt in der durch `LStagg_prec` vorgegebenen Präzision berechnet. Vergleichen Sie bitte dazu das Programm `stagg3` von Seite 85.

- Die Laufzeit läßt sich auch noch dadurch reduzieren, daß man in der `repeat` Schleife durch eine andere Abbruchbedingung den letzten Newton–Schritt vermeidet, der nur zur Konvergenzkontrolle benötigt wird aber keine zusätzliche Genauigkeit liefert.

- Bei der verwendeten staggered Arithmetik werden die Ergebniskomponenten durch sukzessives Auslesen des Akkumulatorinhaltes gewonnen. Es ist daher sinnvoll, diesen Akkuinhalt im oberen Exponentenbereich des benutzten Datenformats anzusiedeln. Bei der Kubikwurzelberechnung wird dies dadurch erreicht, daß man das staggered Argument a mit einem möglichst großen Faktor 10^{ex} skaliert, wobei ex durch 3 teilbar sein muß. Mit Hilfe des Operators `M_10HOCH` wird die Multiplikation mit 10^{ex} schnell und rundungsfehlerfrei durchgeführt. Lediglich die anschließende Multiplikation mit $10^{-ex/3}$ muß als staggered Multiplikation intervallmäßig erfolgen, da der Operator `M_10HOCH` die Multiplikation mit $10^{-ex/3}$ im denormalisierten Zahlenbereich i.a. nicht rundungsfehlerfrei ausführt.

- Mit der Anweisung `v := power(v,Lintval(1)/3)` wird eine Einschließung der gesuchten Kubikwurzel berechnet. Da das Intervall `Lintval(1)/3` nicht punktförmig sein kann, wird die Einschließung `v` nicht maximalgenau berechnet, wobei die Überschätzung um so stärker ausfällt, je mehr die Größenordnung des Arguments a von Eins verschieden ist. Um optimale Einschließungen der Kubikwurzel zu erhalten, sollte man daher bei der Berechnung von `v` eine entsprechende zweite Skalierung vornehmen. Die dafür benötigte Laufzeit ist unerheblich, da die entsprechenden Rechnungen alle im Longreal–Format durchzuführen sind.

Die oben genannten Verbesserungsvorschläge sind in der Funktion `sqrt3` realisiert, die vom Modul `ls_fu1` exportiert wird.

3.2.2.10.5 Arithmetische Ausdrücke

Eine reellwertige Funktion $f(x, y, z)$ sei für $x, y, z \in I\!R$ durch den folgenden arithmetischen Ausdruck definiert:

$$f(x,y,z) := \arccos(x) - \pi \cdot \frac{1682yz^4 + 3y^3 + 29yz^2 - 2y^5 + 832}{192120033} \tag{3.16}$$

Wertet man f dann für beliebige Gleitkommazahlen x, y, z aus, so können durch meist nicht vorhersehbare Auslöschungseffekte die Rechner–Ergebnisse völlig unbrauchbar werden. Niemand wird z.B. auf Anhieb erkennen, daß der Faktor von π für $y_0 = 192119201$ und $z_0 = 35675640$ den Wert 1 hat und damit $f(-1, y_0, z_0)$ verschwindet!

Um auch noch bei starker Auslöschung zuverlässige Funktionswerte zu erhalten, muß die rechte Seite in (3.16) mit Hilfe der Staggered–Arithmetik in hoher Präzision ausgewertet werden.

Das nachfolgende Programm `lstg5` liefert Einschließungen von $f(x, y, z)$, wobei die Argumente x, y, z jeweils als punktförmige Staggered–Intervalle einzugeben sind. Zum Vergleich wird dann der Funktionsterm auch noch im `Longreal`–Format ausgewertet. In der Funktion `power(y,n,rnd)` bestimmt der Parameter `rnd`$= \pm 1$ die Rundung der berechneten y^n-Näherung.

```
program lstg5;
{ Einschliessung eines reellen Ausdrucks:                             }
{ f(x,y,z) = arccos(x)                                                }
{            - Pi * (1682yz^4+3y^3+29yz^2-2y^5+832)/192120033 }
{ f(-1,192119201,35675640) = 0                                        }
use type_def,      { exportiert: type LStagg; var LStagg_prec }
    lstagg,        { exportiert: operator :=,   *;  write(x)  }
    ls_fu1;        { exportiert: arccos,power,...             }
var prec : integer;           { Zur Festlegung der Praezision }
   rnd   : integer;           { rnd = +1,-1 bestimmt Rundung  }
procedure main;
var x,y,z : LStagg[-1..LStagg_prec];
   u,v,w  : Longreal;

function f(x,y,z : LStagg) : LStagg[-1..LStagg_prec];
begin
   f := arccos(x) - Pi_lsm * (1682.0*y*power(z,4)
        + 3.0*power(y,3) + 29.0*y*power(z,2) - 2.0*power(y,5)
        + 832.0) / 192120033.0;
end; { f }
```

```
function g(u,v,w : Longreal) : Longreal;
begin
   g := arccos(u) - Pi_Down * (1682*v*power(w,4,rnd)
        + 3*power(v,3,rnd) + 29*v*power(w,2,rnd)
        - 2*power(v,5,rnd) + 832) / 192120033;
end; { f }

begin
   writeln('x : LStagg   = ?');  read(x);
   writeln('y : LStagg   = ?');  read(y);
   writeln('z : LStagg   = ?');  read(z);
   writeln;
   writeln('f(x,y,z) = ');
   writeln(f(x,y,z));

   writeln('Rundung (+1,-1) bei power-Fkt = ?');
   read(rnd);
   writeln('u : Longreal = ?');  read(u);
   writeln('v : Longreal = ?');  read(v);
   writeln('w : Longreal = ?');  read(w);
   writeln;
   writeln('g(u,v,w) = ');
   writeln(g(u,v,w));
   writeln;
end; { main }

begin
   repeat
      writeln('In type_def definierter LStagg_prec Wert:');
      writeln('LStagg_prec = ',LStagg_prec);
      writeln('gewuenschter LStagg_prec-Wert = ?'); read(prec);
      set_LStagg_prec(prec);
      writeln('Neuer Wert: LStagg_prec = ',LStagg_prec);
      main;
   until false;
end.
```

Der Start des obigen Programms liefert das folgende Ablaufprotokoll:

```
In type_def definierter LStagg_prec Wert:
LStagg_prec = 1
gewuenschter LStagg_prec-Wert = ?
3                    <enter>
Neuer Wert: LStagg_prec = 3
x : LStagg   = ?
--- 3 LStagg components and 1 Linterval are needed in this sequence
-1 0 0 [0,0]         <enter>
y : LStagg   = ?
--- 3 LStagg components and 1 Linterval are needed in this sequence
192119201 0 0 [0,0]  <enter>
z : LStagg   = ?
--- 3 LStagg components and 1 Linterval are needed in this sequence
35675640 0 0 [0,0]   <enter>

f(x,y,z) =
-3.2557333550000000000E-076
-0.0000000000000000000E+000
-0.0000000000000000000E+000
[ -2.7342611905000000000E-075,  2.7342611905000000000E-075 ]

Rundung (+1,-1) bei power-Fkt = ?
1            <enter>
u : Longreal = ?
-1           <enter>
v : Longreal = ?
192119201    <enter>
w : Longreal = ?
35675640     <enter>

g(u,v,w) =  4.9056716333012410401 1E+013
```

Anmerkungen:

1. Mit Hilfe der Staggered–Arithmetik wird der exakte Funktionswert 0 in hoher Genauigkeit eingeschlossen. Dagegen liefert die Auswertung des gleichen Funktionsterms im `Longreal`–Format das absurde Ergebnis $4.9\ldots \cdot 10^{13}$.

2. Zeigen Sie, daß man mit der Gleitkomma–Arithmetik im `Longreal`–Format wieder ein ganz anderes aber immer noch falsches Ergebnis erhält, wenn man an der gleichen Stelle nur den Rundungsparameter $rnd = -1$ wählt.

3.3 Komplexe staggered Arithmetik

In diesem Abschnitt werden die für das Rechnen mit komplexen Zahlen notwendigen Operatoren, Funktionen und Prozeduren im *staggered* Format bzgl. der Basistypen `real` und `Longreal` in getrennten Unterabschnitten bereitgestellt. Ein geklammerter Name (xxxxx) in einer der nachfolgenden Einrahmungen bezeichnet dasjenige Modul, welches die jeweilige Routine exportiert.

3.3.1 Der Basistyp `real`

Wir betrachten jetzt das komplexe *staggered* Format nur zum Basistyp `real`. Bitte beachten Sie auch die Einführung auf Seite 61, die sich auf den reellen Datentyp `Stagg` bezieht. Das dortige Programm `stagg1` liefert bereits wichtige Informationen für die ersten Programmierschritte. Der Datentyp für die komplexen Rechnungen sowie die erforderlichen Operatoren, Funktionen und Prozeduren werden von den Hilfs–Modulen `type_def` und `cstagg` exportiert.

3.3.1.1 Datenstruktur

Datentyp

type Staggcomplex = **dynamic array**[∗] **of** Stagg; (`type_def`)

Präzisionsparameter

var Stagg_prec : integer; (`type_def`)

Variablen-Vereinbarung

```
var z : Staggcomplex[1..2,-1..Stagg_prec];
```

Nach Definition gilt: $z[1] = \Re(z)$ und $z[2] = \Im(z)$.

3.3.1.2 Präzisionsänderung

Eine Präzisionsänderung kann mit Hilfe der Prozedur `set_Stagg_prec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_Stagg_prec` (**prec** : `integer`);	(`type_def`)

Mit Hilfe der Eingangsgröße `prec` wird durch den Prozeduraufruf

```
set_Stagg_prec(prec)
```

die Variable `Stagg_prec` entsprechend geändert. Der Prozeduraufruf muß **vor** der Vereinbarung der entsprechenden *staggered* Variablen erfolgen! Bitte vergleichen Sie dazu auch die Anmerkung 3. von Seite 64.

Der aktuelle Wert der Variablen `Stagg_prec` wird durch die folgende Funktion geliefert:

function `get_Stagg_prec` : `integer`;	(`type_def`)

Präzisionsanpassung

Mit Hilfe der Funktion

function `change_length` (**var** `w` : `Staggcomplex`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec];`

kann die Variable `w`, mit: $\mathtt{ub}(\mathtt{w}[1]) = \mathtt{ub}(\mathtt{w}[2]) \neq \mathtt{Stagg\_prec}$ durch die Anweisung

$$\mathtt{z := change\_length(w);} \quad \rightsquigarrow \quad \mathtt{w} \subseteq \mathtt{z}$$

an die aktuelle, durch `Stagg_prec` gegebene Präzision der Variablen `z` angepaßt werden.

3.3.1.3 Hilfsfunktionen

function `convert` (**var** dr1,dr2,di1,di2 : `dotprecision`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec];`

Mit der Anweiung

$$\mathtt{z := convert(dr1,dr2,di1,di2);} \quad \rightsquigarrow \quad [\mathtt{dr1},\mathtt{dr2}] + i\cdot[\mathtt{di1},\mathtt{di2}] \subseteq \mathtt{z}$$

wird das komplexe Eingangs–Intervall $[dr1, dr2] + i \cdot [di1, di2]$ durch die *staggered* Variable `z` maximalgenau eingeschlossen, wobei die Bedingung `ub(z[1])` = `ub(z[2])` = `Stagg_prec` erfüllt sein muß.

procedure `normal` (**var a** : `Staggcomplex`); (`cstagg`)

Wenn sich bei der Eingangsgröße `a` die `a[i,k]` für $k \geq 1$ überlappen, was z.B. nach einer Division eintreten kann, so liefert der Aufruf `normal(a)` mit der Ausgangsgröße `a` eine maximalgenaue Einschließung der Eingangsgröße, wobei sich die `a[i,k]` der Ausgangsgröße dann bzgl. `k` nicht mehr überlappen. Die Bedingung `ub(a)` = `Stagg_prec` muß erfüllt sein.

function `inf` (**var** z : `Staggcomplex`) : (cstagg) `Staggcomplex[1..2,-1..Stagg_prec]`;

function `sup` (**var** z : `Staggcomplex`) : (cstagg) `Staggcomplex[1..2,-1..Stagg_prec]`;

liefern Punkteinschließungen der komplexen Unter- bzw. Obergrenze von `z` unter der Bedingung: `ub(z[1])` = `ub(z[2])` = `Stagg_prec`.

function `diam` (**var** z : `Staggcomplex`) : real; (cstagg)

liefert mit der Anweisung `r := diam(z);` den aufgerundeten Durchmesser von `z` an die real–Variable `r`. `ub(z[1])` = `ub(z[2])` $\neq$ `Stagg_prec` ist zulässig.

function `mid` (**var** z : `Staggcomplex`) : (cstagg) `Staggcomplex[1..2,-1..Stagg_prec]`;

liefert mit der Anweisung `m := mid(z)` eine Punkt–Einschließung des komplexen, gerundeten Mittelpunktes von `z`. Die nachfolgende Bedingung `ub(z[1])` = `ub(z[2])` = `Stagg_prec` muß erfüllt sein.

procedure `add_Staggc_times_Staggc` (**var** z,w : `Staggcomplex`;
(cstagg) **var** dr1,dr2,di1,di2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter:	z,w : `Staggcomplex`;
	dr1,dr2,di1,di2 : `dotprecision`
Ausgangsparameter:	dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$z \cdot w + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

$\mathtt{ub}(z[1]) = \mathtt{ub}(z[2]) \neq \mathtt{ub}(w[1]) = \mathtt{ub}(w[2]) \neq \mathtt{Stagg\_prec}$ ist zulässig.

procedure `add_Stagg_times_Staggc` (**var** x : `Stagg`;
var z : `Staggcomplex`; **var** dr1,dr2,di1,di2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter:	x : `Stagg`; z : `Staggcomplex`;
	dr1,dr2,di1,di2 : `dotprecision`
Ausgangsparameter:	dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$x \cdot z + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

$\mathtt{ub}(x) \neq \mathtt{ub}(z[1]) = \mathtt{ub}(z[2]) \neq \mathtt{Stagg\_prec}$ ist zulässig.

procedure `add_real_times_Staggc` (**var** r : `real`;
var z : `Staggcomplex`; **var** dr1,dr2,di1,di2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter: x : `real`; z : `Staggcomplex`;
dr1,dr2,di1,di2 : `dotprecision`
Ausgangsparameter: dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$r \cdot z + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

`ub`$(z[1]) =$ `ub`$(z[2]) \neq$ `Stagg_prec` ist zulässig.

> **procedure** `add_complex_times_Staggc` (**var** c : `complex`;
> **var** z : `Staggcomplex`; **var** dr1,dr2,di1,di2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter: c : `complex`; z : `Staggcomplex`;
dr1,dr2,di1,di2 : `dotprecision`
Ausgangsparameter: dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$c \cdot z + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

`ub`$(z[1]) =$ `ub`$(z[2]) \neq$ `Stagg_prec` ist zulässig.

> **procedure** `add_complex_times_Stagg` (**var** c : `complex`;
> **var** s : `Stagg`; **var** dr1,dr2,di1,di2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter: c : `complex`; s : `Stagg`;
dr1,dr2,di1,di2 : `dotprecision`
Ausgangsparameter: dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$c \cdot s + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

`ub`(s) $\neq$ `Stagg_prec` ist zulässig.

Mit Hilfe der letzten fünf Prozeduren lassen sich maximalgenaue Skalarprodukte vom Ergebnistyp `Staggcomplex` realisieren, die z.B. bei der maximalgenauen Matrixmultiplikation zur Anwendung kommen. Beachten Sie jedoch, daß die hier definierte staggered Arithmetik eine Intervall–Arithmetik ist, so daß die dabei auftretenden Intervallüberschätzungen auch durch die Benutzung des Akkumulators nicht verhindert werden können. Der Akkumulator vermeidet lediglich die bekannten Auslöschungseffekte, nicht aber die prinzipiellen Intervallaufblähungen, die jedoch umso kleiner ausfallen, je höher die Präzision der staggered Arithmetik gewählt wird.

function `blow` (z : `Staggcomplex`; eps : `real`) :
(cstagg) `Staggcomplex[1..2,-1..ub(z,2)]`;

`w := blow(z,eps)` liefert die ε–Aufblähung von `z`. $ub(w,2) \stackrel{!}{=} ub(z,2)$.

3.3.1.4 Transferfunktionen

function `Re` (**var** z : `Staggcomplex`) :
(cstagg) `Stagg[-1..Stagg_prec]`;

function `Im` (**var** z : `Staggcomplex`) :
(cstagg) `Stagg[-1..Stagg_prec]`;

liefern den Real– bzw. Imaginärteil von `z`, wobei `ub(z[1])` = `Stagg_prec` bzw. `ub(z[2])` = `Stagg_prec` erfüllt sein muß.

function `cival` (**var** z : `Staggcomplex`) : cinterval; (cstagg)

Die Anweisung `ci := cival(z)` liefert für `z` eine maximalgenaue Einschließung: `z`$\subseteq$ `ci` : cinterval. `ub(z[1])` = `ub(z[2])` $\neq$ `Stagg_prec` ist zulässig.

procedure `TRANS` (**var** z : `Staggcomplex`; **var** c : `complex`);

Eingangsparam. : z : `Staggcomplex`;
Ausgangsparam.: c : `complex` ist maximalgenauer Mittelpunkt von `z`
`ub(z[1])` $=$ `ub(z[2])` $\neq$ `Stagg_prec` ist zulässig.

procedure `TRANS` (**var** z : `Staggcomplex`; **var** c1,c2 : `complex`);

Eingangsparameter : z : `Staggcomplex`;
Ausgangsparameter: c1,c2 : `complex`;
$(c1 + c2)$ ist maximalgenaue Näherung des Mittelpunktes von `z` im doppelt-langen `real`–Format. `ub(z[1])` $=$ `ub(z[2])` $\neq$ `Stagg_prec` ist zulässig.

function `Staggcintval` (**var** ci : `cinterval`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec]`;

`z := Staggcintval(ci);` $\rightsquigarrow$ Die Intervalle `z`,`ci` haben zwar unterschiedliche Datenstrukturen, sind aber mathematisch identisch[5].

function `compl` (**var** x,y : `Stagg`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec]`;

liefert unter der Bedingung `ub(x)` $=$ `ub(y)` $=$ `Stagg_prec` das komplexe staggered Intervall: $x1 + i \cdot x2$.

function `compl` (**var** x : `Stagg`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec]`;

liefert unter der Bedingung `ub(x)` $=$ `Stagg_prec` das reellwertige staggered Intervall: $x + i \cdot [0, 0]$.

[5]Ist `ci` kein Punktintervall, so ist der Durchmesser von `z` für eine sinnvolle *staggered* Arithmetik i.a. zu groß!

function `Staggcompl` (**var** c : `complex`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec];`

`z := Staggcompl(c)` liefert Punkteinschließung für $\texttt{c} \in \texttt{z}$.

function `Staggcompl` (**var** r1,r2 : `real`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec];`

liefert das komplexe staggered Punktintervall: $[r1, r1] + i \cdot [r2, r2]$.

function `Staggcompl` (**var** r : `real`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec];`

liefert das reellwertige staggered Punktintervall: $[r, r] + i \cdot [0, 0]$.

3.3.1.5 Zuweisungsoperatoren

Zuweisung	**Bedeutung**	
`Scv := Sc`	$\texttt{Scv} = \texttt{Sc}[1] + i \cdot \texttt{Sc}[2]$;	$\texttt{ub}(\texttt{Scv}[j]) = \texttt{ub}(\texttt{Sc}[j]),\ j = 1, 2$
`Scv := S`	$\texttt{Scv} = \texttt{S} + i \cdot [0, 0]$;	`ub(Scv[1]) = ub(S)`
`Scv := r`	$\texttt{Scv} = \texttt{r} + i \cdot [0, 0]$;	
`Scv := c`	$\texttt{Scv} = \texttt{c.re} + i \cdot \texttt{c.im}$;	
`Scv := cdot`	$\texttt{cdot} \in \texttt{Scv}$;	maximalgenaue Einschließung
`cv := Sc`	`cv`: Ins real–Format gerundete Mitte von `Sc`	
`civ := Sc`	$\texttt{Sc} \subseteq \texttt{civ}$;	maximalgenaue Einschließung

Tabelle 3.2: Scv = `Staggcomplex`–Variable, cv = `complex`–Variable, civ = `cinterval`–Variable, S = `Stagg`–Ausdruck, c = `complex`–Ausdruck, Sc = `Staggcomplex`–Ausdruck, r = `real`–Ausdruck, cdot = `cdotprecision`–Ausdruck;

3.3.1.6 Vergleichs- und Arithmetik-Operatoren

rechter Operand	integer real	complex	Stagg	Staggcomplex
linker Operand				
monadisch				$+, -$
integer real				**in**, $\diamond$ $=, <>$
complex			$\diamond$	**in**, $\diamond$ $=, <>$
Stagg		$\diamond$		$\diamond$
Staggcomplex	$\diamond$ $=, <>$	$\diamond$ $=, <>$	$\diamond$	**in**, **lies_in** $\diamond, =, <>$

Tabelle 3.3: Die Operatoren des Moduls `cstagg`, $\diamond \in \{+, -, *, /\}$

Weitere Anmerkungen:
In obiger Tabelle haben alle arithmetischen Operatoren $\{+, -, *, /\}$ den Ergebnistyp `Staggcomplex`.
in, **lies_in**, $=$, $<>$ sind Operatoren vom Ergebnistyp `boolean`.
`z lies_in w` $\longleftrightarrow$ `z` $\subseteq$ `w`; `ub(z[j])=ub(w[j])=Stagg_prec, j=1,2;`
Der Operator **in** testet die Relation *liegt in* z.B. zwischen einem `complex`– und einem `Staggcomplex`–Operanden oder die Relation *liegt im Innern* zwischen zwei `Staggcomplex`–Operanden.
Bei einer Variablen `z : Staggcomplex` können die Präzisionen von Real- und Imaginärteil nicht unabhängig voneinander gewählt werden.
Bei den arithmetischen Operatoren kann die Präzision beider Operanden unabhängig vom Wert der Variablen `Stagg_prec` gewählt werden. Die Präzision des Operator–Ergebnisses ist jedoch stets durch den Wert der Variablen `Stagg_prec` gegeben.
Bei den monadischen Operatoren muß die Präzision des rechten Operanden nicht durch den Variablenwert `Stagg_prec` bestimmt sein.

3.3.1.7 Mathematische Standardfunktionen

function `abs` (`var` z : `Staggcomplex`) :	
(cstagg)	`Stagg[-1..Stagg_prec];`

Mit $c \in \mathbb{C}$, $r \in \mathbb{R}$ gilt: $|\mathtt{z}| := \{r \,|\, r = |c| \wedge c \in \mathtt{z}\} \subseteq \mathtt{abs(z)}$; $\mathtt{ub(z[1])} = \mathtt{ub(z[2])} \neq$ `Stagg_prec` ist zulässig.

function `conj` (`var` z : `Staggcomplex`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec];`

Unter der Bedingung $\mathtt{ub(z[1])} = \mathtt{ub(z[2])} =$ `Stagg_prec` erhält man mit `w := conj(z)` das Ergebnis: `w[1] := z[1], w[2] := −z[2].`

function `sqrt` (`var` z : `Staggcomplex`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec];`

Mit $c \in \mathbb{C}$ gilt: $\sqrt{c} \in \mathtt{sqrt(z)}$ für alle $c \in \mathtt{z}$. Die folgende Bedingung $\mathtt{ub(z[1])} = \mathtt{ub(z[2])} =$ `Stagg_prec` muß erfüllt sein. Der Verzweigungsschnitt ist wie üblich die negative, reelle Achse, wobei diese Achse der oberen Halbebene zugeordnet wird. $0 \in \mathtt{z} \neq 0$ führt zu einem Laufzeitfehler.

function `power` (`var` z : `Staggcomplex`; n : `integer`) :	
(cstagg)	`Staggcomplex[1..2,-1..Stagg_prec];`

Mit $c \in \mathbb{C}$ gilt: $c^n \in \mathtt{power(z,n)}$ für alle $c \in \mathtt{z}$. Die folgende Bedingung $\mathtt{ub(z[1])} = \mathtt{ub(z[2])} =$ `Stagg_prec` muß erfüllt sein. $0 \in \mathtt{z}$ und $n < 0$ führt zu einem Laufzeitfehler.

3.3.1.7.1 Tabelle: Standardfunktionen

Standardfunktionen vom Typ Staggcomplex			
Funktion	**Aufruf**	**z : Staggcomplex**	**Hinweise**
$\|z\|$	abs(z)	$\mathbf{z}$ beschränkt durch real–Format	Seite 143
$\sqrt{z}$	sqrt(z)	$0 \in \mathbf{z} \neq 0 \;\rightarrow\;$ Laufzeitfehler	Seite 143
z^n	power(z,n)	z–Bereich, begrenzt durch n	Seite 143
conj(z)	conj(z)	$\mathbf{z}$ beschränkt durch real–Format	Seite 143
diam(z)	diam(z)	$\mathbf{z}$ beschränkt durch real–Format	Seite 136
mid(z)	mid(z)	$\mathbf{z}$ beschränkt durch real–Format	Seite 136

3.3.1.8 Ein-/Ausgabeprozeduren

procedure `read` (**var** F : `Text`; **var** z : `Staggcomplex`); (cstagg)

Mit dem Aufruf `read(z)` wird der Anwender zunächst aufgefordert, für ein rein reelles `z` ($\mathtt{z}[2] = 0$) ein `r` und für ein komplexwertiges `z` ($\mathtt{z}[2] \neq 0$) ein `c` einzugeben. Mit z.B. `ub(z[1])=ub(z[2])=2` erfolgt dann das Einlesen von Real- bzw. Imaginärteil in der Form:

$$\mathtt{z}[1,1] \sqcup \mathtt{z}[1,2] \sqcup [\ \mathtt{z}[1,-1], \mathtt{z}[1,0]\ ] \qquad <\mathbf{enter}> \qquad \text{(Realteil)}$$
$$\mathtt{z}[2,1] \sqcup \mathtt{z}[2,2] \sqcup [\ \mathtt{z}[2,-1], \mathtt{z}[2,0]\ ] \qquad <\mathbf{enter}> \qquad \text{(Imaginärteil)}$$

Dabei dürfen sich die `z[j,1]`,`z[j,2]` sogar überlappen, es muß jedoch $\mathtt{z}[\mathtt{j},-1] \leq \mathtt{z}[\mathtt{j},0]$ erfüllt sein. $\mathtt{ub(z[1])} = \mathtt{ub(z[2])} \neq \text{Stagg\_prec}$ ist zulässig.

Bitte beachten Sie, daß die staggered Variable `z` ihre Komponenten `z[i,j]` genau so übernimmt, wie der Anwender die entsprechenden Komponenten eingegeben hat. Daher muß also z.B. `z[1,1]` nicht der Mittelpunkt oder eine Approximation des Mittelpunktes vom Realteil `z[1]` sein!

procedure `write` (**var** F : `Text`; **var** z : `Staggcomplex`); (cstagg)

Die Ausgabe von `write(z)` liefert eine optimale Einschließung des komplexen staggered Intervalls `z`, wobei sich die ausgegebenen Komponenten nicht überlappen und einerlei Vorzeichen besitzen. Durch `write(z)` bleibt der Wert von `z` ungeändert. $\mathtt{ub(z[1])} = \mathtt{ub(z[2])} \neq \text{Stagg\_prec}$ ist zulässig. Die Ausgabe von `write(z)` erfolgt in der Form

$$(\ \mathtt{w}[1] \sqcup \mathtt{w}[2]\ )$$

wobei die `w[j]` für $j = 1, 2$ folgendermaßen dargestellt werden:

```
          w[j,1]
          w[j,2]
            .
            .
      w[j,ub(z[j])]
  [ w[j,-1], w[j,0] ]
```

$j = 1$ liefert den Realteil und $j = 2$ den Imaginärteil von `w`. Es gilt:

$$\mathtt{z}[1] + i \cdot \mathtt{z}[2] \subseteq \mathtt{w}[1] + i \cdot \mathtt{w}[2].$$

3.3.1.9 Programmbeispiele

3.3.1.9.1 Test der komplexen Quadratwurzel

Das nachfolgende Programm `cstg1` zeigt u.a. die Vereinbarung komplexer staggered Variablen z.B. mit Hilfe von `allocate(...)` und deren Ein- bzw. Ausgabe.

Unter der Voraussetzung $\Re(z) > 0$ gilt für die komplexe Quadratwurzel die Beziehung:

$$z = \sqrt{z^2}$$

Wertet man nun die obige rechte Seite für ein komplexes staggered Intervall `z` unter der Voraussetzung `inf(z[1])` > 0 mit Hilfe der staggered Intervall–Arithmetik aus, so erhält man automatisch eine Einschließung von `z`, die im Programm mit dem Operator **lies_in** überprüft wird.

```
program cstg1(input,output);
{ Test der komplexen Quadratwurzel }
use cstagg, type_def;
var p : integer;
procedure main;
var z : Staggcomplex;
    w,zz : Staggcomplex[1..2,-1..Stagg_prec];
begin
   allocate(z,1..2,-1..1);
   read(z);
   writeln('z-Ausgabe:');   writeln(z);
   w := sqrt(z*z);
   writeln('w-Ausgabe:');   writeln(w);
   zz := change_length(z);
   if zz lies_in w then writeln('z liegt in sqrt(z*z)')
   else writeln('z liegt NICHT in sqrt(z*z)');
   free(z);
end; { main }
begin
   repeat
       writeln('Stagg_prec = ?');   read(p);
       set_Stagg_prec(p);
       main;
   until false;
end.
```

Mit den Eingaben

```
Stagg_prec = 3     <enter>
c                  <enter>

1 [0,0]            <enter>
1 [1e-200,1e-200]  <enter>
```

erhält man die Programmausgabe:

```
z-Ausgabe:
(  1.000000000000E+000
[ -0.000000000000E+000, 0.000000000000E+000 ]

   1.000000000000E+000
[  1.000000000000E-200, 1.000000000000E-200 ]
)
w-Ausgabe:
(  1.000000000000E+000
  -0.000000000000E+000
  -0.000000000000E+000
[ -1.000000000000E-267, 1.000000000000E-267 ]

   1.000000000000E+000
   1.000000000000E-200
  -0.000000000000E+000
[ -1.000000000000E-267, 1.000000000000E-267 ]
)
z liegt in sqrt(z*z)
```

Hinweise:

- Bitte beachten Sie, daß `z` durch `w` maximalgenau eingeschlossen wird.
- Wegen `allocate(z,1..2,-1..1)` erfolgt die Berechnung von `z*z` in der minimalen Präzision `1`. Nach den Anmerkungen von Seite 142 wird das Produkt–Ergebnis jedoch in der durch `Stagg_prec=3` gegebenen Präzision geliefert, so daß die Quadratwurzelberechnung genau in dieser Präzision erfolgt.
- Mit `zz := change_length(z)` wird der Wert von `z` auf die Variable `zz` mit der aktuellen Präzision 3 übertragen. Damit haben die Operanden von `lies_in` die durch `Stagg_prec` vorgegebene Präzision, wie es auf Seite 142 verlangt wird.

3.3.2 Der Basistyp `Longreal`

Wir betrachten in den folgenden Abschnitten das komplexe *staggered* Format zum Basistyp `Longreal`. Bitte beachten Sie auch die Einführung auf Seite 61, die sich allerdings auf den reellen Datentyp `Stagg` bezieht. Das dortige Programm `stagg1` liefert bereits wichtige Informationen für die ersten Programmierschritte, die sich auf das Longreal–Format direkt übertragen lassen. Der Datentyp für die komplexen Rechnungen sowie die erforderlichen Operatoren, Funktionen und Prozeduren werden von den Hilfs–Modulen `type_def` und `lcstagg` exportiert.

3.3.2.1 Datenstruktur

Datentyp

type LStaggcomplex = **dynamic array**[*] **of** LStagg; (`type_def`)

Präzisionsparameter

var LStagg_prec : integer; (`type_def`)

Variablen-Vereinbarung

```
var z : LStaggcomplex[1..2,-1..LStagg_prec];
```

Nach Definition gilt: $z[1] = \Re(z)$ und $z[2] = \Im(z)$.

3.3.2.2 Präzisionsänderung

Eine Präzisionsänderung kann mit Hilfe der Prozedur `set_LStagg_prec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_LStagg_prec` (**prec** : `integer`); (`type_def`)

Mit Hilfe der Eingangsgröße `prec` wird durch den Prozeduraufruf

```
set_LStagg_prec(prec)
```

die Variable `LStagg_prec` entsprechend geändert. Der Prozeduraufruf muß **vor** der Vereinbarung der entsprechenden *staggered* Variablen erfolgen! Bitte vergleichen Sie dazu auch die Anmerkung 3. von Seite 64.

Der aktuelle Wert der Variablen `LStagg_prec` wird durch die folgende Funktion geliefert:

function `get_LStagg_prec` : `integer`; (`type_def`)

Präzisionsanpassung

Mit Hilfe der Funktion

function `change_length` (**var** `w` : `LStaggcomplex`) :	
(lcstagg)	`LStaggcomplex[1..2,-1..LStagg_prec]`;

kann die Variable `w`, mit: $\texttt{ub}(\texttt{w}[1]) = \texttt{ub}(\texttt{w}[2]) \neq \texttt{LStagg\_prec}$ durch die Anweisung

$$\texttt{z := change\_length(w);} \qquad \rightsquigarrow \qquad \texttt{w} \subseteq \texttt{z}$$

an die aktuelle, durch `LStagg_prec` gegebene Präzision der Variablen `z` angepaßt werden.

3.3.2.3 Hilfsfunktionen

function `lconvert` (**var** dr1,dr2,di1,di2 : `dotprecision`) :	
(lcstagg)	`LStaggcomplex[1..2,-1..LStagg_prec]`;

Mit der Anweiung

$$\texttt{z := lconvert(dr1,dr2,di1,di2);} \qquad \rightsquigarrow \qquad [\texttt{dr1},\texttt{dr2}] + i\cdot[\texttt{di1},\texttt{di2}] \subseteq \texttt{z}$$

wird das komplexe Eingangs–Intervall $[dr1, dr2] + i \cdot [di1, di2]$ durch die *staggered* Variable `z` maximalgenau eingeschlossen, wobei die Bedingung $\texttt{ub(z[1])} = \texttt{ub(z[2])} = \texttt{LStagg\_prec}$ erfüllt sein muß.

procedure `normal` (**var a** : `LStaggcomplex`); (`lcstagg`)

Wenn sich bei der Eingangsgröße `a` die `a[i,k]` für $k \geq 1$ überlappen, was z.B. nach einer Division eintreten kann, so liefert der Aufruf `normal(a)` mit der Ausgangsgröße `a` eine maximalgenaue Einschließung der Eingangsgröße, wobei sich die `a[i,k]` der Ausgangsgröße dann bzgl. `k` nicht mehr überlappen. Die Bedingung `ub(a)` = `Stagg_prec` muß erfüllt sein.

function `inf` (**var** z : `LStaggcomplex`) : (lcstagg) `LStaggcomplex[1..2,-1..LStagg_prec]`;

function `sup` (**var** z : `LStaggcomplex`) : (lcstagg) `LStaggcomplex[1..2,-1..LStagg_prec]`;

liefern Punkteinschließungen der komplexen Unter- bzw. Obergrenze von `z` unter der Bedingung: `ub(z[1])` = `ub(z[2])` = `LStagg_prec`.

function `diam` (**var** z : `LStaggcomplex`) : Longreal; (lcstagg)

liefert mit der Anweisung `r := diam(z);` den aufgerundeten Durchmesser von `z` an die Longreal–Variable `r`. Die folgende Ungleichung `ub(z[1])` = `ub(z[2])` $\neq$ `LStagg_prec` ist zulässig.

function `mid` (**var** z : `LStaggcomplex`) : (lcstagg) `LStaggcomplex[1..2,-1..LStagg_prec]`;

liefert mit der Anweisung `m := mid(z)` eine Punkt–Einschließung des komplexen, gerundeten Mittelpunktes von `z`. Die nachfolgende Bedingung `ub(z[1])` = `ub(z[2])` = `LStagg_prec` muß erfüllt sein.

procedure `add_LStaggc_times_LStaggc` (lcstagg)
(**var** z,w : `LStaggcomplex`; **var** dr1,dr2,di1,di2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter: z,w : `LStaggcomplex`;
dr1,dr2,di1,di2 : `dotprecision`

Ausgangsparameter: dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$z \cdot w + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

$\texttt{ub}(z[1]) = \texttt{ub}(z[2]) \neq \texttt{ub}(w[1]) = \texttt{ub}(w[2]) \neq \texttt{LStagg\_prec}$ ist zulässig.

procedure `add_LStagg_times_LStaggc` (**var** x : `LStagg`;
var z : `LStaggcomplex`; **var** dr1,dr2,di1,di2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter: x : `LStagg`; z : `LStaggcomplex`;
dr1,dr2,di1,di2 : `dotprecision`

Ausgangsparameter: dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$x \cdot z + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

$\texttt{ub}(x) \neq \texttt{ub}(z[1]) = \texttt{ub}(z[2]) \neq \texttt{LStagg\_prec}$ ist zulässig.

procedure `add_Longreal_times_LStaggc` (**var** r : `Longreal`;
var z : `LStaggcomplex`; **var** dr1,dr2,di1,di2 : `dotprecision`);

besitzt folgende Ein- und Ausgangsparameter:

Eingangsparameter: r : `Longreal`; z : `LStaggcomplex`;
dr1,dr2,di1,di2 : `dotprecision`

Ausgangsparameter: dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$r \cdot z + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

`ub`$(z[1]) =$ `ub`$(z[2]) \neq$ `LStagg_prec` ist zulässig.

procedure `add_Lcomplex_times_LStaggc` (**var** c : `Lcomplex`;
var z : `LStaggcomplex`; **var** dr1,dr2,di1,di2 : `dotprecision`);

Eingangsparameter: c : `Lcomplex`; z : `LStaggcomplex`;
dr1,dr2,di1,di2 : `dotprecision`

Ausgangsparameter: dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$c \cdot z + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

`ub`$(z[1]) =$ `ub`$(z[2]) \neq$ `LStagg_prec` ist zulässig.

procedure `add_Lcomplex_times_LStagg` (**var** c : `Lcomplex`;
var s : `LStagg`; **var** dr1,dr2,di1,di2 : `dotprecision`);

Eingangsparameter: c : `Lcomplex`; s : `LStagg`;
dr1,dr2,di1,di2 : `dotprecision`

Ausgangsparameter: dr1,dr2,di1,di2 : `dotprecision`

Bezeichnet man zur Unterscheidung die Ausgangsparameter dr1,dr2,di1,di2 mit Dr1,Dr2,Di1,Di2, so gilt nach dem Aufruf der Prozedur:

$$c \cdot s + ([dr1, dr2] + i \cdot [di1, di2]) \subseteq [Dr1, Dr2] + i \cdot [Di1, Di2]$$

`ub`$(s) \neq$ `LStagg_prec` ist zulässig.

function `blow` (z : `LStaggcomplex`; eps : `real`) :
(lcstagg) `LStaggcomplex[1..2,-1..ub(z,2)]`;

`w := blow(z,eps)` liefert die ε–Aufblähung von `z`. $ub(w,2) \overset{!}{=} ub(z,2)$.

3.3.2.4 Transferfunktionen

function `Re` (**var** z : `LStaggcomplex`) :
(lcstagg) `LStagg[-1..LStagg_prec]`;

function `Im` (**var** z : `LStaggcomplex`) :
(lcstagg) `LStagg[-1..LStagg_prec]`;

liefern den Real– bzw. Imaginärteil von `z`, wobei `ub(z[1])` = `LStagg_prec` bzw. `ub(z[2])` = `LStagg_prec` erfüllt sein muß.

function `cival` (**var** z : `LStaggcomplex`) : Lcinterval; (lcstagg)

Die Anweisung `ci := cival(z)` liefert für `z` eine maximalgenaue Einschließung: `z`$\subseteq$ `ci` : Lcinterval. `ub(z[1])` $\neq$ `LStagg_prec` ist zulässig.

procedure `TRANS` (**var** z : `LStaggcomplex`; **var** c : `Lcomplex`);

Eingangsparam. : z : `LStaggcomplex`;

Ausgangsparam.: c : `Lcomplex` ist maximalgenauer Mittelpunkt v. `z`
`ub(z[1])` = `ub(z[2])` $\neq$ `Stagg_prec` ist zulässig.

procedure `TRANS` (**var** z : `LStaggcomplex`; **var** c1,c2: `Lcomplex`);

Eingangsparameter : z : `LStaggcomplex`;

Ausgangsparameter: c1,c2 : `Lcomplex`;

$(c1+c2)$ ist maximalgenaue Näherung des Mittelpunktes von `z` im doppelt-langen `Longreal`–Format. `ub(z[1])` $=$ `ub(z[2])` $\neq$ `Stagg_prec` ist zulässig.

function `LStaggcintval` (**var** ci : `Lcinterval`) :	
(lcstagg)	`LStaggcomplex[1..2,-1..LStagg_prec]`;

`z := LStaggcintval(ci);` $\rightsquigarrow$ Die Intervalle `z`,`ci` haben zwar unterschiedliche Datenstrukturen, sind aber mathematisch identisch[6].

function `compl` (**var** x,y : `LStagg`) :	
(lcstagg)	`LStaggcomplex[1..2,-1..LStagg_prec]`;

liefert unter der Bedingung `ub(x)` $=$ `ub(y)` $=$ `LStagg_prec` das komplexe staggered Intervall: $x1 + i \cdot x2$.

function `compl` (**var** x : `LStagg`) :	
(lcstagg)	`LStaggcomplex[1..2,-1..LStagg_prec]`;

liefert unter der Voraussetzung `ub(x)` $=$ `LStagg_prec` das reellwertige staggered Intervall: $x + i \cdot [0,0]$.

function `LStaggcompl` (**var** c : `Lcomplex`) :	
(lcstagg)	`LStaggcomplex[1..2,-1..LStagg_prec]`;

`z := LStaggcompl(c)` liefert Punkteinschließung für `c` $\in$ `z`.

[6]Ist `ci` kein Punktintervall, so ist der Durchmesser von `z` für eine sinnvolle *staggered* Arithmetik i.a. zu groß!

function `LStaggcompl` (**var** r1,r2 : `Longreal`) :
(lcstagg) `LStaggcomplex[1..2,-1..LStagg_prec]`;

liefert das komplexe staggered Punktintervall: $[r1, r1] + i \cdot [r2, r2]$.

function `LStaggcompl` (**var** r : `Longreal`) :
(lcstagg) `LStaggcomplex[1..2,-1..LStagg_prec]`;

liefert das reellwertige staggered Punktintervall: $[r, r] + i \cdot [0, 0]$.

3.3.2.5 Zuweisungsoperatoren

Zuweisung	**Bedeutung**	
`Scv := Sc`	$\texttt{Scv} = \texttt{Sc}[1] + i \cdot \texttt{Sc}[2]$;	$\texttt{ub}(\texttt{Scv}[j]) = \texttt{ub}(\texttt{Sc}[j]),\ j = 1, 2$
`Scv := S`	$\texttt{Scv} = \texttt{S} + i \cdot [0, 0]$;	`ub(Scv[1]) = ub(S)`
`Scv := r`	$\texttt{Scv} = \texttt{r} + i \cdot [0, 0]$;	
`Scv := c`	$\texttt{Scv} = \texttt{c.re} + i \cdot \texttt{c.im}$;	
`Scv := cdot`	$\texttt{cdot} \in \texttt{Scv}$;	maximalgenaue Einschließung
`cv := Sc`	`cv`: Ins Longreal–Format gerundete Mitte von `Sc`	
`civ := Sc`	$\texttt{Sc} \subseteq \texttt{civ}$;	maximalgenaue Einschließung

Tabelle 3.4: Scv = `LStaggcomplex`–Variable, cv = `Lcomplex`–Variable, civ = `Lcinterval`–Variable, S = `LStagg`–Ausdruck, c = `Lcomplex`–Ausdruck, Sc = `LStaggcomplex`–Ausdruck, r = `Longreal/real/integer`–Ausdruck, cdot = `cdotprecision`–Ausdruck;

3.3.2.6 Vergleichs- und Arithmetik-Operatoren

rechter Operand	real Longreal	Lcomplex	LStagg	LStaggcomplex
linker Operand				
monadisch				$+, -$
real Longreal				**in**, $\diamond$ $=, <>$
Lcomplex			$\diamond$	**in**, $\diamond$ $=, <>$
LStagg		$\diamond$		$\diamond$
LStaggcomplex	$\diamond$ $=, <>$	$\diamond$ $=, <>$	$\diamond$	**in**, **lies_in** $\diamond, =, <>$

Tabelle 3.5: Die Operatoren des Moduls `lcstagg`, $\diamond \in \{+, -, *, /\}$

Weitere Anmerkungen:
In obiger Tabelle haben alle arithmetischen Operatoren $\{+, -, *, /\}$ den Ergebnistyp `LStaggcomplex`.
in, **lies_in**, $=$, $<>$ sind Operatoren vom Ergebnistyp `boolean`.
`z lies_in w` $\longleftrightarrow$ `z` $\subseteq$ `w`; `ub(z[j])=ub(w[j])=LStagg_prec, j=1,2;`
Der Operator **in** testet die Relation *liegt in* z.B. zwischen einem `Lcomplex`– und einem `LStaggcomplex`–Operanden oder die Relation *liegt im Innern* zwischen zwei `LStaggcomplex`–Operanden.
Bei einer Variablen `z : LStaggcomplex` können die Präzisionen von Real- und Imaginärteil nicht unabhängig voneinander gewählt werden.
Bei den arithmetischen Operatoren kann die Präzision beider Operanden unabhängig vom Wert der Variablen `LStagg_prec` gewählt werden. Die Präzision des Operator–Ergebnisses ist jedoch stets durch den Wert der Variablen `LStagg_prec` gegeben.
Bei den monadischen Operatoren muß die Präzision des rechten Operanden nicht durch den Variablenwert `LStagg_prec` bestimmt sein.

3.3.2.7 Mathematische Standardfunktionen

function `abs` (`var` z : `LStaggcomplex`) :
(lcstagg) `LStagg[-1..LStagg_prec];`

Mit $c \in \mathbb{C}$, $r \in \mathbb{R}$ gilt: $|\mathtt{z}| := \{r \mid r = |c| \wedge c \in \mathtt{z}\} \subseteq \mathtt{abs(z)}$;
$\mathtt{ub}(\mathtt{z}[1]) = \mathtt{ub}(\mathtt{z}[2]) \neq$ `LStagg_prec` ist zulässig.

function `conj` (`var` z : `LStaggcomplex`) :
(lcstagg) `LStaggcomplex[1..2,-1..LStagg_prec];`

Unter der Bedingung $\mathtt{ub}(\mathtt{z}[1]) = \mathtt{ub}(\mathtt{z}[2]) =$ `LStagg_prec` erhält man mit `w := conj(z)` das Ergebnis: `w[1] := z[1]`, `w[2] :=` $-$`z[2]`.

function `sqrt` (`var` z : `LStaggcomplex`) :
(lcstagg) `LStaggcomplex[1..2,-1..LStagg_prec];`

Mit $c \in \mathbb{C}$ gilt: $\sqrt{c} \in \mathtt{sqrt(z)}$ für alle $c \in \mathtt{z}$. Die folgende Bedingung $\mathtt{ub}(\mathtt{z}[1]) = \mathtt{ub}(\mathtt{z}[2]) =$ `LStagg_prec` muß erfüllt sein. Der Verzweigungsschnitt ist wie üblich die negative, reelle Achse, wobei diese Achse der oberen Halbebene zugeordnet wird. $0 \in \mathtt{z} \neq 0$ führt zu einem Laufzeitfehler.

function `power` (`var` z : `LStaggcomplex`; n : `integer`) :
(lcstagg) `LStaggcomplex[1..2,-1..LStagg_prec];`

Mit $c \in \mathbb{C}$ gilt: $c^n \in \mathtt{power(z,n)}$ für alle $c \in \mathtt{z}$. Die folgende Bedingung $\mathtt{ub}(\mathtt{z}[1]) = \mathtt{ub}(\mathtt{z}[2]) =$ `LStagg_prec` muß erfüllt sein. $0 \in \mathtt{z}$ und $n < 0$ führt zu einem Laufzeitfehler.

3.3.2.7.1 Tabelle: Standardfunktionen

Standardfunktionen vom Typ LStaggcomplex			
Funktion	**Aufruf**	**z : LStaggcomplex**	**Hinweise**
$\|z\|$	abs(z)	z beschränkt durch Longreal–Format	Seite 157
$\sqrt{z}$	sqrt(z)	$0 \in \mathbf{z} \neq 0 \;\rightarrow\;$ Laufzeitfehler	Seite 157
z^n	power(z,n)	z–Bereich, begrenzt durch n	Seite 157
conj(z)	conj(z)	z beschränkt durch Longreal–Format	Seite 157
diam(z)	diam(z)	z beschränkt durch Longreal–Format	Seite 150
mid(z)	mid(z)	z beschränkt durch Longreal–Format	Seite 150

3.3.2.8 Ein-/Ausgabeprozeduren

procedure `read` (**var** F : `Text`; **var** z: `LStaggcomplex`); (lcstagg)

Mit dem Aufruf `read(z)` wird der Anwender zunächst aufgefordert, für ein rein reelles `z` ($\mathtt{z}[2] = 0$) ein `r` und für ein komplexwertiges `z` ($\mathtt{z}[2] \neq 0$) ein `c` einzugeben. Mit z.B. `ub(z[1])=ub(z[2])=2` erfolgt dann das Einlesen von Real- bzw. Imaginärteil in der Form:

$$\mathtt{z}[1,1] \sqcup \mathtt{z}[1,2] \sqcup [\, \mathtt{z}[1,-1], \mathtt{z}[1,0] \,] \qquad < \mathtt{enter} > \qquad \text{(Realteil)}$$
$$\mathtt{z}[2,1] \sqcup \mathtt{z}[2,2] \sqcup [\, \mathtt{z}[2,-1], \mathtt{z}[2,0] \,] \qquad < \mathtt{enter} > \qquad \text{(Imaginärteil)}$$

Dabei dürfen sich die `z[j,1],z[j,2]` sogar überlappen, es muß jedoch $\mathtt{z}[\mathtt{j},-1] \leq \mathtt{z}[\mathtt{j},0]$ erfüllt sein. `ub(z[1])` = `ub(z[2])` ≠ Stagg_prec ist zulässig.

Bitte beachten Sie, daß die staggered Variable `z` ihre Komponenten `z[i,j]` genau so übernimmt, wie der Anwender die entsprechenden Komponenten eingegeben hat. Daher muß also z.B. `z[1,1]` nicht der Mittelpunkt oder eine Approximation des Mittelpunktes vom Realteil `z[1]` sein!

procedure `write` (**var** F: `Text`; **var** z: `LStaggcomplex`); (lcstagg)

Die Ausgabe von `write(z)` liefert eine optimale Einschließung des komplexen staggered Intervalls `z`, wobei sich die ausgegebenen Komponenten nicht überlappen und einerlei Vorzeichen besitzen. Durch `write(z)` bleibt der Wert von `z` ungeändert. `ub(z[1])` = `ub(z[2])` ≠ LStagg_prec ist zulässig. Die Ausgabe von `write(z)` erfolgt in der Form

$$(\ \mathtt{w}[1] \sqcup \mathtt{w}[2]\ )$$

wobei die `w[j]` für $j = 1, 2$ folgendermaßen dargestellt werden:

```
        w[j,1]
        w[j,2]
          .
          .
     w[j,ub(z[j])]
  [ w[j,-1], w[j,0] ]
```

$j = 1$ liefert den Realteil und $j = 2$ den Imaginärteil von `w`. Es gilt:

$$\mathtt{z}[1] + i \cdot \mathtt{z}[2] \subseteq \mathtt{w}[1] + i \cdot \mathtt{w}[2].$$

3.3.2.9 Programmbeispiele

3.3.2.9.1 Die komplexe Division

Mit der Darstellung

$$z_1 = x_1 + i \cdot y_1, \qquad z_2 = x_2 + i \cdot y_2, \quad i^2 = -1$$

gilt für die komplexe Division im Falle $z_2 \neq 0$:

$$\frac{z_1}{z_2} = \frac{x_1 x_2 + y_1 y_2}{x_2^2 + y_2^2} + i \cdot \frac{y_1 x_2 - x_1 y_2}{x_2^2 + y_2^2}$$

Bei Auswertung der beiden Zähler in naiver Gleitkomma–Arithmetik erhält man für den Real- oder Imaginärteil von z_1/z_2 in konstruierten Fällen so starke Auslöschung, daß sich das exakte Ergebnis vom Rechnerergebnis um viele Größenordnungen unterscheiden kann. Berechnet man die Zähler jedoch mit Hilfe des Akkumulators, so liefern die entsprechenden Operatoren für die Punkt- und Intervall–Auswertung von z_1/z_2 hochgenaue Ergebnisse. Noch sehr viel genauere Einschließungen des Quotienten erhält man bei Anwendung des Divisionsoperators aus dem Modul `lcstagg`. Das nachfolgende Programm `lcstg1` liefert mit: $z_1 = 1 + i \cdot (1 + 10^{-20}),\ z_2 = (1 - 10^{-20}) + i$

```
program lcstg1(input,output);
{ Test der komplexen Division }
use lcstagg, type_def, lc_ari, lci_ari;
var p : integer;

procedure main;
var z1s,z2s    : LStaggcomplex[1..2,-1..LStagg_prec];
   x1,x2,y1,y2 : Longreal;
   z1,z2       : Lcomplex;
   z1i,z2i     : Lcinterval;
priority CDIV = *;  { Naive komplexe Gleitkomma-Division }
operator CDIV(var z,w : Lcomplex) res: Lcomplex;
var Ne : Longreal;
begin
   Ne := w.re*w.re + w.im*w.im;
   res.re := (z.re*w.re + z.im*w.im) / Ne;
   res.im := (z.im*w.re - z.re*w.im) / Ne;
end;  { CDIV }
begin
   x1 := 1; y1 := lval('1.00000000000000000001');
```

```
      z1 := Lcompl(x1,y1);   { z1 = 1+i*(1+10^-20) }
      x2 := lval('0.99999999999999999999'); y2 := 1;
      z2 := Lcompl(x2,y2);   { z2 = (1-10^-20) + i }
      writeln(z1 CDIV z2);   { Naiver Operator     }
      writeln('---------------------------------');
      writeln(z1/z2);     {  Operator / aus lc_ari }
      writeln('---------------------------------');
      z1i := Lintval( Lcompl(x1,y1) );
      z2i := Lintval( Lcompl(x2,y2) );
      writeln(z1i/z2i);   { Operator / aus lci_ari }
      writeln('---------------------------------');
      z1s := LStaggcompl(x1,y1);
      z2s := LStaggcompl(x2,y2);
      writeln(z1s / z2s); { Operator / aus lcstagg }
      writeln('---------------------------------');
   end;  { main }
   begin { lcstg1 }
      repeat
          writeln('LStagg_prec = ?');   read(p);
          set_LStagg_prec(p);
          main;
      until false;
   end.
```

und `LStagg_prec=2` die Ergebnisse:

```
(  1.00000000000000000001E+000,  0.00000000000000000000E+000 )
---------------------------------------------------------------
(  1.00000000000000000001E+000, -5.00000000000000000005E-041 )
---------------------------------------------------------------
([ 1.00000000000000000000E+000,  1.00000000000000000002E+000 ],
 [-5.00000000000000000006E-041, -5.00000000000000000004E-041 ])
---------------------------------------------------------------
(  1.00000000000000000001E+000
   4.99999999999999999999E-041
 [ 9.99999999999999999974E-062,  9.99999999999999999975E-062 ]

  -5.00000000000000000005E-041
  -2.49999999999999999999E-081
 [-9.99999999999999999988E-102, -9.99999999999999999987E-102 ])
---------------------------------------------------------------
```

Kapitel 4

Matrizen

4.1 Testmatrizen

In diesem Abschnitt werden Testmatrizen vorgestellt, deren Inverse zwar analytisch darstellbar, in Gleitkomma–Rechnung jedoch wegen starker Auslöschungseffekte nur schwer zu realisieren sind. Es ist dabei zu beachten, daß die Elemente dieser Testmatrizen in den zur Verfügung stehenden Datenformaten *real* und *Longreal* **exakt** darstellbar sind, da sonst schon minimale Rundungsfehler beim Speichern der Matrixelemente die Eigenschaften dieser Testmatrizen i.a. stark beeinflussen.

4.1.1 Hilbertmatrizen

Hilbertmatrizen $\boldsymbol{h}$ der Dimension n sind definiert durch:

$$h_{ij} := \frac{1}{i+j-1}, \qquad i,j = 1,2,\dots,n. \tag{4.1}$$

Die Inverse $\boldsymbol{h}^{-1}$ besitzt die Elemente

$$(h^{-1})_{ij} := \frac{(-1)^{i+j}}{i+j-1} \cdot \frac{(n+i-1)! \cdot (n+j-1)!}{[(i-1)! \cdot (j-1)!]^2 \cdot (n-i)! \cdot (n-j)!}$$

und die Determinanten der ersten $n = 1, 2, ...$ Hilbertmatrizen sind gegeben durch die reziproken Werte der Folge:

$$1, 12, 2160, 6048000, 266716800000, ...$$

woraus sich die schlechte Konditioniertheit der Hilbertmatizen direkt ergibt. Nach (4.1) sind die h_{ij} für $n > 1$ rationale Zahlen, die im *real*- und *Longreal*–Format i.a. nicht exakt darstellbar sind. Multipliziert man daher $\boldsymbol{h}$ mit dem

Hauptnenner seiner Elemente, so besitzt die neue Matrix $\boldsymbol{H}$ ganzzahlige Elemente, die bei nicht zu großer Dimension n im jeweiligen Datenformat exakt gespeichert werden können. Entsprechend findet man die Inverse $\boldsymbol{H}^{-1}$, indem man die $(h^{-1})_{ij}$ durch den gleichen Hauptnenner dividiert, wodurch die schlechte Konditioniertheit auch bei den Matrizen $\boldsymbol{H}$ gegeben ist. Für $n = 2$ erhält man mit dem Hauptnenner 6 die Matrizen:

$$\boldsymbol{H}_2 = \begin{pmatrix} 6 & 3 \\ 3 & 2 \end{pmatrix}, \qquad (\boldsymbol{H}_2)^{-1} = \begin{pmatrix} 2/3 & -1 \\ -1 & 2 \end{pmatrix}$$

procedure `HILBERT` (**var** H : `rmatrix`); (mv_exap)

liefert bis $n = 18$ die mit dem Hauptnenner multiplizierte Hilbertmatrix $\boldsymbol{H}$ für das *real*–Format. Für $n > 18$ erfolgt eine entsprechende Fehlermeldung.

procedure `HILBERT` (**var** H : `Lrmatrix`); (lmv_exap)

liefert bis $n = 26$ die mit dem Hauptnenner multiplizierte Hilbertmatrix $\boldsymbol{H}$ für das *Longreal*–Format. Für $n > 26$ erfolgt eine Fehlermeldung.

4.1.2 Boothroyd/Dekker–Matrizen

Boothroyd/Dekker–Matrizen $\boldsymbol{B}$ der Dimension n sind definiert durch:

$$\begin{aligned} B_{ij} &:= \frac{n}{i+j-1} \cdot \binom{n+i-1}{i-1} \cdot \binom{n-1}{n-j}, \qquad i,j = 1,2,\ldots,n. \\ (B_{ij})^{-1} &= (-1)^{i+j} B_{ij} \end{aligned}$$

Die B_{ij} sind positive ganze Zahlen, wobei sich die Elemente der Inversen einfach durch Multiplikation mit $(-1)^{i+j}$ ergeben. Für $n = 3$ gilt beispielsweise:

$$\boldsymbol{B}_3 = \begin{pmatrix} 3 & 3 & 1 \\ 6 & 8 & 3 \\ 10 & 15 & 6 \end{pmatrix}, \qquad (\boldsymbol{B}_3)^{-1} = \begin{pmatrix} 3 & -3 & 1 \\ -6 & 8 & -3 \\ 10 & -15 & 6 \end{pmatrix}$$

Die Zeilensummen der Inversen sind abwechselnd $+1$ und -1. Die Kondition von $\boldsymbol{B}_n$ verschlechtert sich rasch mit wachsendem n. Die Eigenwerte von $\boldsymbol{B}_n$ sind reell, positiv und voneinander verschieden

$$0 < \lambda_1 < \lambda_2 < \cdots < \lambda_{n-1} < \lambda_n,$$

und es gilt für gerade n

$$\lambda_1\lambda_n = \lambda_2\lambda_{n-1} = \cdots \lambda_\nu\lambda_\nu = 1 \quad \text{mit} \quad \nu = \frac{n}{2}.$$

Für ungerade n gilt: $\quad \lambda_\mu = 1 \quad \text{mit} \quad \mu = \frac{n}{2} + 1.$
Da die Determinante das Produkt aller Eigenwerte ist, folgt daraus noch:

$$\det\ \boldsymbol{B}_n = \det\ \boldsymbol{B}_n^{-1} = 1.$$

procedure `DEKKER` (**var** B : `rmatrix` (mv_exap)
var v : `rvector`);

liefert bis $n = 16$ die Boothroyd/Dekker–Matrix $\boldsymbol{B}$ für das *real*–Format. Im Fall $n > 16$ erfolgt eine entsprechende Fehlermeldung. Der Vektor `v` erhält die Komponenten $1, 2, \ldots, n$.

procedure `DEKKER` (**var** B : `Lrmatrix` (lmv_exap)
var v : `Lrvector`);

liefert bis $n = 27$ die Boothroyd/Dekker–Matrix $\boldsymbol{B}$ für das *Longreal*–Format. Im Fall $n > 27$ erfolgt eine entsprechende Fehlermeldung. Der Vektor `v` erhält die Komponenten $1, 2, \ldots, n$.

procedure `DEKKER_INV` (**var** I : `rmatrix`); (mv_exap)

liefert bis $n = 16$ die Inverse der Boothroyd/Dekker–Matrix für das *real*–Format. Im Fall $n > 16$ erfolgt eine entsprechende Fehlermeldung.

procedure `DEKKER_INV` (**var** I : `Lrmatrix`); (lmv_exap)

liefert bis zur Dimension $n = 27$ die Inverse der Boothroyd/Dekker–Matrix für das *Longreal*–Format. Im Fall $n > 27$ erfolgt eine entsprechende Fehlermeldung. Im Programm `tstdet1` auf Seite 241 findet man eine Anwendung der Boothroyd/Dekker–Matrix.

4.1.3 Zielke–Matrizen

Wir betrachten die modifizierten, reellen Zielke–Matrizen der Form:

$$\boldsymbol{Z} := \begin{pmatrix} a+b & a+b & \cdot & \cdot & a+b & a \\ a+b & \cdot & \cdot & a+b & a & \cdot \\ \cdot & \cdot & \cdot & a & \cdot & \cdot \\ \cdot & a+b & \cdot & \cdot & \cdot & \cdot \\ a+b & a & \cdot & \cdot & \cdot & a \\ a & \cdot & \cdot & \cdot & a & a-c \end{pmatrix}, \qquad a,b,c \in I\!R$$

Die symmetrischen Matrizen Z sind unter den Bedingungen $|a| \gg |b|, |c|$ sehr schlecht konditioniert und eignen sich u.a. als Testmatrizen zur Eigenwertberechnung. Mit Hilfe der Eingangsparameter a, b, c liefert die von den Hilfsmodulen `(l)mv_exap` exportierte

procedure `ZIELKE` (**var** `Z : (L)rmatrix`; `a,b,c : (Long)real`);

mit der Ausgangsgröße Z die gewünschte Zielke–Matrix. Es ist darauf zu achten, daß $a, b, c \in I\!R$ so gewählt werden, daß $a + b$ und $a - c$ im `real`- bzw `Longreal`–Format exakt darstellbar sind, da sonst eine entsprechende Fehlermeldung erfolgt.

4.2 Blockmatrizen

4.2.1 Definitionen

Blockmatrizen $\boldsymbol{B}$ werden aufgebaut aus quadratischen Matrizen $\boldsymbol{U}, \boldsymbol{V}$ gleicher Dimension und haben folgende Gestalt:

$$\boldsymbol{B} = \begin{pmatrix} \boldsymbol{U} & \boldsymbol{V} \\ \boldsymbol{V} & \boldsymbol{U} \end{pmatrix} \qquad \text{oder} \qquad \boldsymbol{B}1 = \begin{pmatrix} \boldsymbol{U} & -V \\ \boldsymbol{V} & +U \end{pmatrix}$$

Eine Blockmatrix $\boldsymbol{B}$ ist somit quadratisch, und es gilt:

$$\mathrm{Dim}(\boldsymbol{B}) = 2 \cdot \mathrm{Dim}(\boldsymbol{U}) = 2 \cdot \mathrm{Dim}(\boldsymbol{V})$$

4.2.2 Realisierung

Mit Hilfe der folgenden Prozeduren lassen sich zu gegebenen Matrizen $\boldsymbol{U}, \boldsymbol{V}$ die Blockmatrizen $\boldsymbol{B}$, $\boldsymbol{B1}$ wie folgt realisieren:

procedure `block` (**var** U,V,B : ■);

procedure `block_1` (**var** U,V,B1 : ■);

Die Zuordnung des Symbols ■ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von ■ zu den erlaubten Datentypen		
■	`(L)rmatrix`	`(L)cmatrix`
exportierendes Modul	block	
■	`(L)imatrix`	`(L)cimatrix`
exportierendes Modul	iblock	

4.2.3 Programmbeispiel

Mit Hilfe der obigen Prozeduren `block` bzw. `block_1` können viele Blockmatrizen konstruiert werden. Das nachfolgende Programm `tstblock` liefert ein Beispiel für eine Blockmatrix $\boldsymbol{B}$, die nur schwer invertiert werden kann. Spezielle Blockmatrizen sind daher zum Test spezieller Matrix–Funktionen besonders gut geeignet.

```
program tstblock(input,output);
use block , { Exportiert  block(U,V,B) }
    mv_ari; { Matrix/Vektor-Arithmetik }
var U,V    : rmatrix[0..2,1..3];
   B       : rmatrix[1..6,1..6];
begin
   U := 1;       { Identische Matrix mit U[i,j] = 1 }
   V := 1e-30 * id(V);     { 1e-30 * Einheitsmatrix }
   block(U,V,B); { Realisiert mit U,V Blockmatrix B }
   writeln(B);
end.
```

Das obige Programm `tstblock` realisiert mit Hilfe der beiden Matrizen $\boldsymbol{U} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$, $\boldsymbol{V} = \begin{pmatrix} 10^{-30} & 0 & 0 \\ 0 & 10^{-30} & 0 \\ 0 & 0 & 10^{-30} \end{pmatrix}$ die Blockmatrix $\boldsymbol{B}$ der Dimension 6:

$$\boldsymbol{B} = \begin{pmatrix} 1 & 1 & 1 & 10^{-30} & 0 & 0 \\ 1 & 1 & 1 & 0 & 10^{-30} & 0 \\ 1 & 1 & 1 & 0 & 0 & 10^{-30} \\ 10^{-30} & 0 & 0 & 1 & 1 & 1 \\ 0 & 10^{-30} & 0 & 1 & 1 & 1 \\ 0 & 0 & 10^{-30} & 1 & 1 & 1 \end{pmatrix}$$

Zur Kontrolle werden mit `writeln(B)` die Elemente von $\boldsymbol{B}$ auf dem Bildschirm zeilenweise ausgegeben.

4.3 Matrix/Vektor–Ausdrücke zum Typ `Longreal`

Mit Hilfe der auch in der BCD–Version von **PASCAL-XSC** zur Verfügung stehenden Lattenkreuzausdrücke (#-Ausdrücke) können beliebige Matrix- und Vektor–Operationen bzgl. der Standardtypen *rvector, ivector, cvector, civector, rmatrix, imatrix, cmatrix* und *cimatrix* im Akkumulator rundungsfehlerfrei berechnet und anschließend mit nur einer einzigen Rundung ins *real*-Format transformiert werden.

Für den Basistyp *Longreal* sind die entsprechenden Lattenkreuzausdrücke nicht vorgesehen. Im Bedarfsfall lassen sich jedoch beliebige Matrix- und Vektor–Operationen auch bzgl. der analogen Standardtypen *Lrvector, Livector, Lcvector, Lcivector, Lrmatrix, Limatrix, Lcmatrix* und *Lcimatrix* mit Hilfe des Datentyps *dotprecision* ebenfalls rundungsfehlerfrei berechnen und anschließend wieder mit nur einer einzigen Rundung ins *Longreal*-Format zurücktransformieren.

In den beiden folgenden Abschnitten sind die Routinen zur maximalgenauen Auswertung der wichtigsten Matrix/Vektor–Ausdrücke für reelle und komplexe Rechnungen zum Basistyp `Longreal` zusammengestellt.

4.3.1 Reelle Matrix/Vektor–Ausdrücke

Die folgenden Prozeduren des Moduls `mv_comp` liefern maximalgenaue Auswertungen von Matrix/Vektor–Ausdrücken zu den Datentypen:

Lrmatrix, Limatrix, Lrvector, Livector;

Zu jeder Routine wird eine Beziehung zwischen den Ein- und Ausgangsgrößen angegeben, wobei die Eingangsgrößen stets auf der linken und die Ausgangsgrößen stets auf der rechten Seite stehen.

procedure `mul` (**var** A,B,R1,R2 : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{B} \approx \boldsymbol{R1} + \boldsymbol{R2}$$

procedure `mul` (**var** A,B : `Lrmatrix`; (mv_comp)
var R : `Lrmatrix`; **var** I : `Limatrix`);

$$\boldsymbol{A} \cdot \boldsymbol{B} \in \boldsymbol{R} + \boldsymbol{I}$$

procedure `m_comp` (**var** A,B : `Lrmatrix`; x : `Longreal`;
var R : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} + x \cdot \boldsymbol{B} \approx \boldsymbol{R}$$

procedure `m_comp` (**var** A,B : `Lrmatrix`; x : `Longreal`;
var R : `Limatrix`); (mv_comp)

$$\boldsymbol{A} + x \cdot \boldsymbol{B} \in \boldsymbol{R}$$

procedure `m_comp` (**var** A,B : `Limatrix`; x : `Longreal`;
var R : `Limatrix`); (mv_comp)

$$\boldsymbol{A} + x \cdot \boldsymbol{B} \subseteq \boldsymbol{R}$$

procedure `ab_plus_c` (**var** A,B,C,R : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R}$$

procedure `ab_plus_c` (**var** A,B,C : `Lrmatrix`; (mv_comp)
var I : `Limatrix`);

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \in \boldsymbol{I}$$

procedure `ab_plus_c` (**var** A : `Lrmatrix`; **var** B : `Limatrix`;
var C : `Lrmatrix`; **var** I : `Limatrix`);

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \subseteq \boldsymbol{I}$$

procedure `ab_plus_c` (**var** A,B,C,R1,R2 : `Lrmatrix`);

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R}1 + \boldsymbol{R}2$$

procedure `ab_plus_c` (**var** A : `Limatrix`; **var** B,C : `Lrmatrix`;
var I : `Limatrix`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \subseteq \boldsymbol{I}$$

procedure `m_comp2` (**var** A,B,C,R : `Lrmatrix`); (mv_comp)

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} \approx \boldsymbol{R}$$

procedure `m_comp2` (**var** A,B,C : `Lrmatrix`; (mv_comp)
var I : `Limatrix`);

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} \in \boldsymbol{I}$$

procedure `m_comp2` (**var** A,B,C,D,R : `Lrmatrix`); (mv_comp)

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \approx \boldsymbol{R}$$

procedure `m_comp2` (**var** A,B,C,D : `Lrmatrix`; (mv_comp)
var I : `Limatrix`);

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \in \boldsymbol{I}$$

procedure `m_comp2` (**var** A,B : `Lrmatrix`; **var** C : `Limatrix`;
var D : `Lrmatrix`; **var** I : `Limatrix`);

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \subseteq \boldsymbol{I}$$

procedure `m_comp2_m` (**var** A,B,C,R1,R2 : `Lrmatrix`);

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + (\boldsymbol{A} + \boldsymbol{B}) \approx \boldsymbol{R1} + \boldsymbol{R2}$$

procedure `m_comp1` (**var** A,B,C,R : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) \approx \boldsymbol{R}$$

procedure `m_comp1` (**var** A,B,C,D,R : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \approx \boldsymbol{R}$$

procedure `m_comp1` (**var** A,B,C,D : `Lrmatrix`; (mv_comp)
var I : `Limatrix`);

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \in \boldsymbol{I}$$

procedure `m_comp1` (**var** A : `Limatrix`; **var** B,C,D : `Lrmatrix`;
var I : `Limatrix`); (mv_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \subseteq \boldsymbol{I}$$

procedure `m_comp` (**var** A,B,C,R : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{B} \cdot \boldsymbol{C} \approx \boldsymbol{R}$$

procedure `m_comp` (**var** A,B,C,D,R : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + (\boldsymbol{B} + \boldsymbol{C}) \cdot \boldsymbol{D} \approx \boldsymbol{R}$$

Notwendiger Indexbereich aller Matrizen: `[1..ub(A),1..ub(A)]`

procedure `m_comp3` (**var** A,B,C,R1,R2 : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + (\boldsymbol{B} + \boldsymbol{C}) \approx \boldsymbol{R1} + \boldsymbol{I\!R2}$$

procedure `m_comp5` (**var** A,B,C,D,R : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C} + \boldsymbol{D}) \approx \boldsymbol{R}$$

procedure `m_comp4` (**var** A,B,C,R1,R2 : `Lrmatrix`); (mv_comp)

$$\boldsymbol{A} + \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R1} + \boldsymbol{R2}$$

procedure `m_comp1` (**var** A,B : `Lrmatrix`; (mv_comp)
var C,I : `Limatrix`);

$$\boldsymbol{A} + \boldsymbol{B} + \boldsymbol{C} \subseteq \boldsymbol{I}$$

procedure `mv_comp1` (**var** A : `Lrmatrix`; **var** b,c : `Lrvector`;
var r : `Lrvector`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{b} + \boldsymbol{c} \approx \boldsymbol{r}$$

procedure `mv_comp1a` (**var** A : `Lrmatrix`; **var** b,c : `Lrvector`;
t : `Longreal`;
var r : `Lrvector`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{b} + t \cdot \boldsymbol{c} \approx \boldsymbol{r}$$

procedure `mv_comp1b` (**var** A : `Lrmatrix`; **var** b,c,d : `Lrvector`;
t : `Longreal`;
var i : `Livector`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{b} + t \cdot \boldsymbol{c} + \boldsymbol{d} \in \boldsymbol{i}$$

procedure `mv_comp2` (**var** A : `Lrmatrix`; **var** b,c,d : `Lrvector`;
var i : `Livector`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d} \in \boldsymbol{i}$$

procedure `mv_comp2a` (**var** A : `Limatrix`; **var** b,c : `Lrvector`;
t : `Longreal`;
var i : `Livector`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{b} + t \cdot \boldsymbol{c} \subseteq \boldsymbol{i}$$

procedure `mv_comp2b` (**var** A : `Limatrix`; **var** b,c,d : `Lrvector`;
t : `Longreal`;
var i : `Livector`); (mv_comp)

$$\boldsymbol{A} \cdot \boldsymbol{b} + t \cdot \boldsymbol{c} + \boldsymbol{d} \subseteq \boldsymbol{i}$$

procedure `mv_comp3` (**var** A : `Lrmatrix`; **var** b : `Lrvector`;
var c,i : `Livector`); (mv_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{b} + \boldsymbol{c}) \subseteq \boldsymbol{i}$$

procedure `mv_comp3` (**var** A : `Limatrix`; **var** b : `Lrvector`;
var c,i : `Livector`); (mv_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{b} + \boldsymbol{c}) \subseteq \boldsymbol{i}$$

procedure `mv_comp3` (**var** A : `Lrmatrix`; **var** b,c : `Lrvector`;
x : `Longreal`; **var** r : `Lrvector`);

$$\boldsymbol{A} \cdot (\boldsymbol{b} + \boldsymbol{c}) + x \cdot (\boldsymbol{b} + \boldsymbol{c}) \approx r$$ (mv_comp)

procedure `mv_comp3` (**var** A : `Lrmatrix`; **var** b,c : `Lrvector`;
x : `Longreal`; **var** r : `Livector`);

$$\boldsymbol{A} \cdot (\boldsymbol{b} + \boldsymbol{c}) + x \cdot (\boldsymbol{b} + \boldsymbol{c}) \in r$$ (mv_comp)

procedure `mv_comp4` (**var** A,B : `Lrmatrix`; **var** v,r : `Lrvector`);

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{v} \approx \boldsymbol{r}$$ (mv_comp)

procedure `mv_comp4` (**var** A,B : `Lrmatrix`; **var** v,w,r : `Lrvector`);

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{v} + \boldsymbol{w} \approx \boldsymbol{r}$$ (mv_comp)

procedure `mv_comp4` (**var** A,B : `Lrmatrix`;
var u,v,w,r : `Lrvector`);

$$\boldsymbol{A}\boldsymbol{u} + \boldsymbol{B}(\boldsymbol{v} + \boldsymbol{w}) \approx \boldsymbol{r} \qquad \text{(mv\_comp)}$$

procedure `vv_comp` (**var** a,b : `Lrvector`; x : `Longreal`;
var r : `Lrvector`); (vv_comp)

$$\boldsymbol{a} \cdot x + \boldsymbol{b} \approx \boldsymbol{r}$$

procedure `vv_comp` (**var** u : `Lrvector`; x,y : `Longreal`;
var v,w,r : `Lrvector`); (vv_comp)

$$(x + y) \cdot \boldsymbol{u} + \boldsymbol{v} + \boldsymbol{w} \approx \boldsymbol{r}$$

procedure `vv_comp` (**var** a,u,v,b : `Lrvector`; x : `Longreal`;
var r : `Longreal`); (vv_comp)

$$\boldsymbol{a} \cdot (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{b} \cdot (\boldsymbol{u} + \boldsymbol{v}) + x \approx r$$

procedure `vv_comp` (**var** a,u,v,b : `Lrvector`; x : `Longreal`;
var r : `Linterval`); (vv_comp)

$$\boldsymbol{a} \cdot (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{b} \cdot (\boldsymbol{u} + \boldsymbol{v}) + x \in r$$

4.3.2 Komplexe Matrix/Vektor–Ausdrücke

Die folgenden Prozeduren des Moduls `mvc_comp` liefern maximalgenaue Auswertungen von Matrix/Vektor–Ausdrücken zu den Datentypen:

Lcmatrix, Lcimatrix, Lrmatrix, Lcvector;

Zu jeder Routine wird eine Beziehung zwischen den Ein- und Ausgangsgrößen angegeben, wobei die Eingangsgrößen stets auf der linken und die Ausgangsgrößen stets auf der rechten Seite stehen.

procedure `mul` (**var** A,B,R1,R2 : `Lcmatrix`); (mvc_comp)

$$\boldsymbol{A} \cdot \boldsymbol{B} \approx \boldsymbol{R1} + \boldsymbol{R2}$$

procedure `mul` (**var** A,B : `Lcmatrix`; (mvc_comp)
var R : `Lcmatrix`; **var** I : `Lcimatrix`);

$$\boldsymbol{A} \cdot \boldsymbol{B} \in \boldsymbol{R} + \boldsymbol{I}$$

procedure `ab_plus_c` (**var** A,B,C,R : `Lcmatrix`); (mvc_comp)

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R}$$

procedure `ab_plus_c` (**var** A,B : `Lcmatrix`; **var** C : `Lrmatrix`;
var R : `Lcmatrix`); (mvc_comp)

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R}$$

procedure `ab_plus_c` (**var** A,B,C : `Lcmatrix`; (mvc_comp)
var I : `Lcimatrix`);

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \in \boldsymbol{I}$$

procedure `ab_plus_c` (**var** A,B : `Lcmatrix`; **var** C : `Lrmatrix`;
var I : `Lcimatrix`); (mvc_comp)

$$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \in \boldsymbol{I}$$

procedure `m_comp1` (**var** A,B,C,D,R : `Lcmatrix`); (mvc_comp)

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \approx \boldsymbol{R}$$

procedure `m_comp1` (**var** A,B,C : `Lcmatrix`; (mvc_comp)
var D : `Lrmatrix`; **var** R : `Lcmatrix`);

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \approx \boldsymbol{R}$$

procedure `m_comp1` (**var** A,B,C,D : `Lcmatrix`; (mvc_comp)
var R : `Lcimatrix`);

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \in \boldsymbol{R}$$

procedure `m_comp1` (**var** A,B,C : `Lcmatrix`; (mvc_comp)
var D : `Lrmatrix`; **var** R : `Lcimatrix`);

$$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \in \boldsymbol{R}$$

procedure `m_comp2` (**var** A,B,C,D,R : `Lcmatrix`); (mvc_comp)

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \approx \boldsymbol{R}$$

procedure `m_comp2` (**var** A,B,C : `Lcmatrix`; (mvc_comp)
var D : `Lrmatrix`; **var** R : `Lcmatrix`);

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \approx \boldsymbol{R}$$

procedure `m_comp2` (**var** A,B,C,D : `Lcmatrix`; (mvc_comp)
var R : `Lcimatrix`);

$$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \in \boldsymbol{R}$$

procedure `m_comp2` (**var** A,B,C : `Lcmatrix`; (mvc_comp)
var D : `Lrmatrix`; **var** R : `Lcimatrix`);

$$(\boldsymbol{A}+\boldsymbol{B})\cdot\boldsymbol{C}+\boldsymbol{D}\in\boldsymbol{R}$$

procedure `m_comp2_m` (**var** A,B,C,R1,R2 : `Lcmatrix`);

$$(\boldsymbol{A}+\boldsymbol{B})\cdot\boldsymbol{C}+(\boldsymbol{A}+\boldsymbol{B})\approx\boldsymbol{R1}+\boldsymbol{R2}$$ (mvc_comp)

procedure `m_comp3` (**var** A,B,C,R1,R2 : `Lcmatrix`);

$$\boldsymbol{A}\cdot(\boldsymbol{B}+\boldsymbol{C})+(\boldsymbol{B}+\boldsymbol{C})\approx\boldsymbol{R1}+\boldsymbol{R2}$$ (mvc_comp)

procedure `m_comp1` (**var** A,B : `Lcmatrix`; (mvc_comp)
var C,R : `Lcimatrix`);

$$\boldsymbol{A}+\boldsymbol{B}+\boldsymbol{C}\subseteq\boldsymbol{R}$$

procedure `mv_comp1` (**var** A : `Lcmatrix`; (mvc_comp)
var b,c,r : `Lcvector`);

$$\boldsymbol{A}\cdot\boldsymbol{b}+\boldsymbol{c}\approx\boldsymbol{r}$$

4.4 Reelle staggered Arithmetik

Mit Hilfe der auch in der BCD–Version von **PASCAL-XSC** zur Verfügung stehenden Lattenkreuzausdrücke (#-Ausdrücke) können beliebige Matrix- und Vektor–Operationen bzgl. der Standardtypen *rvector, ivector, cvector, civector, rmatrix, imatrix, cmatrix* und *cimatrix* im Akkumulator rundungsfehlerfrei berechnet und anschließend mit nur einer einzigen Rundung ins *real*-Format transformiert werden.

Für den Basistyp *Longreal* sind die entsprechenden Lattenkreuzausdrücke nicht vorgesehen. Im Bedarfsfall lassen sich jedoch beliebige Matrix- und Vektor–Operationen auch bzgl. der analogen Standardtypen *Lrvector, Livector, Lcvector, Lcivector, Lrmatrix, Limatrix, Lcmatrix* und *Lcimatrix* mit Hilfe des Datentyps *dotprecision* ebenfalls rundungsfehlerfrei berechnen und anschließend wieder mit nur einer einzigen Rundung ins *Longreal*-Format zurücktransformieren. Die für die Praxis wichtigsten Matrix/Vektor–Ausdrücke lassen sich mit Hilfe entsprechender Routinen auswerten, die von den Hilfsmodulen `mv_comp` bzw. `mvc_comp` exportiert werden. Zur näheren Information findet man auf den Seiten 169 bzw. 176 jeweils eine Zusammenstellung der zur Verfügung stehenden Matrix-Vektor-Ausdrücke.

Damit besteht die Möglichkeit, für die beiden Basistypen *real* und *Longreal* beliebige Matrix/Vektor–Ausdrücke maximalgenau auszuwerten. Als einfaches Beispiel betrachten wir das nachfolgende Programm `stgm1`, in dem mit `i` = [9.999999999999E-001, 1.000000000000E+000] eine maximalgenaue Einschließung des exakten Skalarproduktes

$$\mathtt{v1} \cdot \mathtt{v2} = 1 - 10^{-24} + 10^{-100} \in \mathtt{i}$$

berechnet wird.

```
program stgm1(input,output);
use i_ari;   { Intervall-Arithmetik zum Basistyp real }
var v1,v2 : rvector[1..2];
   i      : interval;
begin
   v1[1] := 1+1e-12;  v1[2] := 1;
   v2[1] := 1-1e-12;  v2[2] := 1e-100;
   i := ##(v1*v2); { Maximalgenaue Einschliessung von v1*v2 }
   writeln('Maximalgenaue Einschliessung von v1*v2:');
   write(i);
end.
```

Eine viel genauere Einschließung von `v1` · `v2` erhält man[1], wenn das obige Skalarprodukt mit Hilfe einer staggered Matrix/Vektor–Arithmetik in höherer Präzision berechnet wird. In den folgenden Abschnitten findet man dazu für die beiden Basistypen *real* und *Longreal* eine Zusammenstellung aller benötigten Funktionen, Prozeduren und Operatoren. Im Kapitel 3 ab Seite 61 sind die allgemeinen Grundbegriffe einer staggered Arithmetik kurz zusammengestellt.

4.4.1 Der Basistyp `real`

Wir betrachten jetzt das *staggered* Format zum Basistyp *real*. Beachten Sie bitte dazu auch die Einführung auf Seite 61. Das dortige Programm `stagg1` liefert bereits wichtige Informationen für die ersten Programmierschritte. Die für die staggered Matrix/Vektor–Arithmetik notwendigen Datentypen sowie die erforderlichen Operatoren, Funktionen und Prozeduren werden von den Hilfs–Modulen `type_def` und `stgmv` exportiert. Ein geklammerter Name (xxx) in einer der nachfolgenden Einrahmungen bezeichnet das Hilfsmodul, welches die jeweilige Routine exportiert.

4.4.1.1 Datenstruktur

Datentyp

type Staggrvector = **dynamic array**[∗] **of** Stagg;	(`type_def`)

type Staggrmatrix = **dynamic array**[∗] **of** Staggrvector;	(s.o.)

Präzisionsparameter

var Stagg_prec : integer;	(`type_def`)

Variablen-Vereinbarung

```
var v : Staggrvector[1..2,-1..Stagg_prec];
```

`v` ist ein Vektor mit zwei Komponenten vom Typ `Stagg`, deren Präzision durch die Variable `Stagg_prec` vorgegeben ist.

[1]Vergleiche das Programm `stgm2` von Seite 189

Die Vereinbarung eines Vektors vom Typ `Staggrvector` ist auch innerhalb einer Prozedur ohne die Variable `Stagg_prec` wie folgt möglich:

```
procedure beispiel( n : integer );
var v : Staggrvector;
begin
   allocate(v,1..2,-1..2*n);
   .....  Anweisungen  .....
   free(v);
end;  { beispiel }
```

```
var m : Staggrmatrix[1..2,1..3,-1..Stagg_prec];
```

`m` ist eine Matrix mit zwei Zeilen und drei Spalten. Ihre Elemente sind vom Typ `Stagg` mit einer Präzision, die durch die Variable `Stagg_prec` aus dem Hilfsmodul `type_def` vorgegeben ist. Die Matrix `m` kann ganz analog zur oberen Prozedur `beispiel` mit Hilfe von `allocate(m,1..2,1..3,-1..n)` auch ohne die Variable `Stagg_prec` vereinbart werden.

4.4.1.2 Präzisionsänderung

Eine Präzisionsänderung kann mit Hilfe der Prozedur `set_Stagg_prec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_Stagg_prec` (**prec** : `integer`); (`type_def`)

Mit Hilfe der Eingangsgröße `prec` wird durch den Prozeduraufruf

```
set_Stagg_prec(prec)
```

die Variable `Stagg_prec` aus dem Modul `type_def` entsprechend geändert. Der aktuelle Wert dieser Variablen `Stagg_prec` wird durch die folgende Funktion geliefert:

function `get_Stagg_prec` : `integer`; (`type_def`)

Präzisionsanpassung

Mit Hilfe der Funktion

function `change_length` (**var** v : `Staggrvector`) :
(stgmv) `Staggrvector[lb(v)..ub(v),-1..Stagg_prec]`;

kann der Vektor `v`, mit `ub(v,2)` $\neq$ `Stagg_prec` durch die Anweisung

`x := change_length(v);` $\rightsquigarrow$ `v` $\subseteq$ `x` : `Staggrvector`

an die aktuelle, durch `Stagg_prec` gegebene Präzision des Vektors `x` angepaßt werden, wobei die Dimensionen beider Vektoren übereinstimmen müssen.

Mit Hilfe der Funktion

function `change_length` (**var** m : `Staggrmatrix`) : Staggrmatrix
`[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec]`; (stgmv)

kann die Matrix `m`, mit `ub(m,3)` $\neq$ `Stagg_prec` durch die Anweisung

`x := change_length(m);` $\rightsquigarrow$ `m` $\subseteq$ `x` : `Staggrmatrix`

an die aktuelle, durch `Stagg_prec` gegebene Präzision der Matrix `x` angepaßt werden, wobei die Dimensionen beider Matrizen übereinstimmen müssen.

4.4.1.3 Hilfsfunktionen

procedure `mul` (**var** A,B,C : `rmatrix`;
(stgmv) **var** R : `Staggrmatrix`;

Mit den Eingangsmatrizen `A,B,C` liefert die Ausgangsgröße `R` eine Einschließung des Produktes $\boldsymbol{A} \cdot \boldsymbol{B} \cdot \boldsymbol{C} \in \boldsymbol{R}$.

4.4.1.4 Standardfunktionen

function `inf` (**var** v : `Staggrvector`) :
(stgmv) `Staggrvector[lb(v)..ub(v),-1..Stagg_prec];`

function `sup` (**var** v : `Staggrvector`) :
(stgmv) `Staggrvector[lb(v)..ub(v),-1..Stagg_prec];`

liefern Punkteinschließungen der Unter- bzw. Obergrenze des Vektors `v`; die Bedingung `ub(v,2)` = `Stagg_prec` muß erfüllt sein.

function `inf` (**var** m : `Staggrmatrix`) : `Staggrmatrix`
(stgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec];`

function `sup` (**var** m : `Staggrmatrix`) : `Staggrmatrix`
(stgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec];`

liefern Punkteinschließungen der Unter- bzw. Obergrenze der Matrix `m`, wobei die Bedingung `ub(m,3)` = `Stagg_prec` erfüllt sein muß.

function `mid` (**var** v : `Staggrvector`) :
(stgmv) `Staggrvector[lb(v)..ub(v),-1..Stagg_prec];`

function `mid` (**var** m : `Staggrmatrix`) : `Staggrmatrix`
(stgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec];`

liefern eine Punkteinschließung des gerundeten Mittelpunktes von `v` bzw. `m`, wobei gelten muß: $\texttt{ub}(\texttt{v},2) = \texttt{Stagg\_prec} = \texttt{ub}(\texttt{m},3)$.

function `null` (**var** v : `Staggrvector`) :
(stgmv) `Staggrvector[lb(v)..ub(v),-1..Stagg_prec];`

function `null` (**var** m : `Staggrmatrix`) : `Staggrmatrix`
(stgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec];`

liefern einen staggered Nullvektor bzw. eine staggered Nullmatrix mit dem aktuellen Indexbereich von `v` bzw. `m`, wobei jedoch die Präzision der Null–Elemente durch `Stagg_prec` bestimmt wird. Für die Funktions–Argumente `v,m` ist $\texttt{ub}(\texttt{v},2) \neq \texttt{Stagg\_prec} \neq \texttt{ub}(\texttt{m},3)$ zulässig. `ub(v,2)` ist dabei die Präzision der staggered Elemente des Vektors `v`, und `ub(m,3)` ist die Präzision der staggered Elemente der Matrix `m`.

function `id` (**var** m : `Staggrmatrix`) : `Staggrmatrix`
(stgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec];`

liefert eine staggered Einheitsmatrix mit dem aktuellen Indexbereich von `m`, wobei jedoch die Präzision der gelieferten staggered Matrix–Elemente durch `Stagg_prec` bestimmt wird. $\texttt{ub}(\texttt{m},3) \neq \texttt{Stagg\_prec}$ ist zulässig.

function `transp` (**var** m : `Staggrmatrix`) : `Staggrmatrix`
(stgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec];`

liefert die transponierte Matrix (Zeilen- und Spalten-Vertauschung), wobei die Bedingung $\texttt{ub}(\texttt{m},3) = \texttt{Stagg\_prec}$ erfüllt sein muß. `ub(m,3)` ist die Präzision der staggered Elemente von `m`.

4.4.1.5 Transferfunktionen

procedure `TRANS` (**var** sv : `Staggrvector`;	
(stgmv) **var** v : `rvector`);	

Zum Eingangsvektor `sv` liefert die Ausgangsgröße `v` den maximalgenauen Mittelpunkts–Vektor von `sv`.

procedure `TRANS` (**var** sv : `Staggrvector`;	
(stgmv) **var** v1,v2 : `rvector`);	

Zum Eingangsvektor `sv` liefern die Ausgangsgrößen `v1,v2` mit `v1` + `v2` ∈ `sv` den maximalgenauen Mittelpunkts–Vektor von `sv` in doppelter Länge.

procedure `TRANS` (**var** sm : `Staggrmatrix`;	
(stgmv) **var** m : `rmatrix`);	

Zur Eingangsmatrix `sm` liefert die Ausgangsgröße `m` die maximalgenaue Mittelpunkts–Matrix von `sm`.

procedure `TRANS` (**var** sm : `Staggrmatrix`;	
(stgmv) **var** m1,m2 : `rmatrix`);	

Zur Eingangsmatrix `sm` liefern die Ausgangsgrößen `m1,m2` mit `m1` + `m2` ∈ `sm` die maximalgenaue Mittelpunkts–Matrix von `sm` in doppelter Länge.

4.4.1.6 Zuweisungsoperatoren

Zuweisung	Bedeutung		
`Svv := Sv`	`Svv = Sv`;	falls `ub(Svv,2) = ub(Sv,2)`	
`Svv := r`	`Svv = r`;	`ub(Svv,2)` ≠ `Stagg_prec`	ist zulässig
`Svv := S`	`Svv = S`;	falls `ub(Svv,2) = ub(S)`	
`Svv := rv`	`Svv = rv`;	`ub(Svv,2)` ≠ `Stagg_prec`	ist zulässig
`ivv := Sv`	`Sv` ⊆ `ivv`	`ub(Sv,2)` ≠ `Stagg_prec`	ist zulässig
`rvv := Sv`	`rvv`: Mitte v. `Sv`;	`ub(Sv,2)` ≠ `Stagg_prec`	ist zulässig
`Smv := Sm`	`Smv = Sm`;	falls `ub(Smv,3) = ub(Sm,3)`	
`Smv := r`	`Smv = r`;	`ub(Smv,3)` ≠ `Stagg_prec`	ist zulässig
`Smv := S`	`Smv = S`;	falls `ub(Smv,3) = ub(S)`	
`Smv := rm`	`Smv = rm`;	`ub(Smv,3)` ≠ `Stagg_prec`	ist zulässig
`imv := Sm`	`Sm` ⊆ `imv`	`ub(Sm,3)` ≠ `Stagg_prec`	ist zulässig
`rmv := Sm`	`rmv`: Mitte v. `Sm`;	`ub(Sm,3)` ≠ `Stagg_prec`	ist zulässig

Tabelle 4.1: rv = **`rvector`**–Ausdruck, rvv = **`rvector`**–Variable,
Svv = **`Staggrvector`**–Variable, Sv = **`Staggrvector`**–Ausdruck,
r = **`real/integer`**–Ausdruck, S = **`Stagg`**–Ausdruck,
ivv = **`ivector`**–Variable, imv = **`imatrix`**–Variable,
rm = **`rmatrix`**–Ausdruck, rmv = **`rmatrix`**–Variable,
Smv = **`Staggrmatrix`**–Variable, Sm = **`Staggrmatrix`**–Ausdruck

Anmerkungen:

- `ub(Svv,2)` bedeutet die Obergrenze des zweiten Indexbereiches von `Svv` und ist damit der Präzisionsparameter dieser `Staggrvector`–Variablen.
- `ub(Smv,3)` bedeutet die Obergrenze des dritten Indexbereiches von `Smv` und ist damit der Präzisionsparameter dieser `Staggrmatrix`–Variablen.
- Mit **`rvv`** := `Sv` liefert **`rvv`** natürlich nur die ins *real*–Format gerundete Mitte vom `Staggrvector`–Ausdruck. Analoges gilt für **`rmv`** := `Sm`.

4.4.1.7 Vergleichs- und Arithmetik-Operatoren

rechter Operand	integer real	Stagg	Stagg- rvector	Stagg- rmatrix	rvector	rmatrix
linker Operand						
monadisch			$+, -$	$+, -$		
integer real			$*$	$*$		
Stagg			$*$	$*$	$*$	$*$
Stagg- rvector	$*, /$	$*, /$	**lies_in** **in**, =, <> $+, -, *$		$+, -, *$ =, <>	
Stagg- rmatrix	$*, /$	$*, /$	$*$	**lies_in** **in**, =, <> $+, -, *$	$*$	$+, -, *$ =, <>
rvector		$*, /$	**in**, =, <> $+, -, *$			
rmatrix		$*, /$	$*$	**in**, =, <> $+, -, *$		

Tabelle 4.2: Die Operatoren des Moduls `stgmv`

Weitere Anmerkungen:
In obiger Tabelle ist der Ergebnistyp der arithmetischen Operatoren $\{+, -, *, /\}$ durch die üblichen Matrix/Vektor–Operationen bestimmt.
Bei den arithmetischen Operatoren kann die Präzision beider Operanden unabhängig vom Wert der Variablen `Stagg_prec` gewählt werden. Die Präzision des Operator–Ergebnisses ist jedoch stets durch den Wert der Variablen `Stagg_prec` gegeben.
in, **lies_in**, =, <> sind Operatoren vom Ergebnistyp `boolean`.
`u lies_in v` $\longleftrightarrow$ `u` $\subseteq$ `v;` `ub(u,2)=ub(v,2)=Stagg_prec;`
Der Operator **in** testet die Relation *liegt in* z.B. zwischen einem `rvector`– und einem `Staggrvector`–Operanden oder die Relation *liegt im Innern* zwischen zwei `Staggrvector`–Operanden. Für Matrizen gelten ganz entsprechende Aussagen.

4.4.1.8 Ein-/Ausgabeprozeduren

Es stehen die Prozeduren

procedure `read` (**var** F : `Text`; **var** v: `Staggrvector`); (stgmv)

procedure `read` (**var** F : `Text`; **var** m: `Staggrmatrix`); (stgmv)

procedure `write` (**var** F : `Text`; **var** v: `Staggrvector`); (stgmv)

procedure `write` (**var** F : `Text`; **var** m: `Staggrmatrix`); (stgmv)

mit einem optionalem Fileparameter, beliebig vielen Ein- bzw. Ausgabe-Parametern, jedoch ohne Formatspezifikationen zur Verfügung.

Die Eingabe eines staggered Vektors oder einer staggered Matrix erfolgt im interaktiven Modus komponentenweise, wobei die Präzision dieser Komponenten durch die Vereinbarung der entsprechenden staggered Variablen vorgegeben ist. Die Eingabe einer Matrix erfolgt dabei wie üblich zeilenweise. Bitte beachten Sie, daß bei der Eingabe eines einzelnen staggered Elements dessen Komponenten und das anschließende Intervall genau so übernommen werden, wie der Anwender diese Werte eingegeben hat. Vergleichen Sie dazu auch die entsprechenden Anmerkungen auf Seite 81.

Die Ausgabe eines staggered Vektors oder einer staggered Matrix erfolgt wie bei der Eingabe komponentenweise in dem auf Seite 82 beschriebenen Format, wobei sich die staggered Komponenten eines einzelnen Elements nicht überlappen und einerlei Vorzeichen besitzen. Die ausgegebenen Elemente schließen die berechneten Matrix- oder Vektor-Komponenten garantiert ein, wobei die berechneten Elemente durch die Ausgabe nicht verändert werden.

4.4.1.9 Programmbeispiele

4.4.1.9.1 Skalarprodukt in hoher Präzision

Das folgende **PASCAL-XSC** Programm `stgm2` berechnet für eine Präzision `Stagg_prec` ≥ 2 maximalgenau Punkteinschließungen des exakten Skalarproduktes $\mathtt{v1} \cdot \mathtt{v2} = 1 - 10^{-24} + 10^{-100}$, das mit dem Programm `stgm1` von Seite 179 im einfachen *real*–Format nur maximalgenau eingeschlossen werden konnte.

```
program stgm2(input,output);
use type_def, stagg, stgmv;
var p : integer;
procedure main;
var v1 : Staggrvector[1..2,-1..Stagg_prec];
    v2 : Staggrvector;
begin
   v1[1] := 1+1e-12; v1[2] := 1; { Definiert Vektor v1}
   allocate(v2,2..3,-1..4); { Definiert Indexbereiche }
   v2[2] := 1-1e-12; v2[3] := 1e-100;  { Definiert v2 }
   write('Skalaprodukt v1*v2 mit Praezision  ');
   writeln(get_Stagg_prec);
   writeln(v1*v2);
   free(v2);       { Gibt Speicher von v2 wieder frei }
end; { main }
begin { stgm2 }
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

Ergebnisausgabe für `Stagg_prec = 2`:

```
Skalarprodukt v1*v2 mit Praezision  2
  9.999999999999E-001
  9.999999999900E-014
[ 1.000000000000E-100, 1.000000000000E-100 ]
```

Beachten Sie bitte, daß die Operanden `v1,v2` bei der Berechnung von `v1*v2` unterschiedliche Präzisionen besitzen, was nach den Anmerkungen von Seite 187 durchaus zulässig ist.

4.4.2 Der Basistyp `Longreal`

Wir betrachten jetzt das *staggered* Format zum Basistyp *Longreal*. Beachten Sie bitte dazu auch die Einführung auf Seite 61. Das dortige Programm `stagg1` liefert bereits wichtige Informationen für die ersten Programmierschritte. Die für die staggered Matrix/Vektor–Arithmetik notwendigen Datentypen sowie die erforderlichen Operatoren, Funktionen und Prozeduren werden von den Hilfs–Modulen `type_def` und `lstgmv` exportiert. Ein geklammerter Name (xxx) in einer der nachfolgenden Einrahmungen bezeichnet das Hilfsmodul, welches die jeweilige Routine exportiert.

4.4.2.1 Datenstruktur

Datentyp

type LStaggrvector = **dynamic array**[$*$] **of** LStagg; (`type_def`)

type LStaggrmatrix = **dynamic array**[$*$] **of** LStaggrvector; (s.o.)

Präzisionsparameter

var LStagg_prec : integer; (`type_def`)

Variablen-Vereinbarung

```
var v : LStaggrvector[1..2,-1..LStagg_prec];
```

`v` ist ein Vektor mit zwei staggered Komponenten vom Typ `LStagg`, deren Präzision durch die Variable `LStagg_prec` vorgegeben ist. Die Vereinbarung eines Vektors vom Typ `Staggrvector` ist auch innerhalb einer Prozedur ohne die Variable `LStagg_prec` wie folgt möglich:

```
procedure beispiel( n : integer );
var v : LStaggrvector;
begin
   allocate(v,1..2,-1..2*n);
   .....  Anweisungen  .....
   free(v);
end;  { beispiel }
```

```
var m : LStaggrmatrix[1..2,1..3,-1..LStagg_prec];
```

`m` ist eine Matrix mit zwei Zeilen und drei Spalten. Ihre Elemente sind vom Typ `LStagg` mit einer Präzision, die durch die Variable `LStagg_prec` aus dem Hilfsmodul `type_def` vorgegeben ist. Die Matrix `m` kann ganz analog zur oberen Prozedur `beispiel` mit Hilfe von `allocate(m,1..2,1..3,-1..n)` auch ohne die Variable `LStagg_prec` vereinbart werden.

4.4.2.2 Präzisionsänderung

Eine Präzisionsänderung kann mit Hilfe der Prozedur `set_LStagg_prec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_LStagg_prec` (**prec** : `integer`); (`type_def`)

Mit Hilfe der Eingangsgröße `prec` wird durch den Prozeduraufruf

`set_LStagg_prec(prec)`

die Variable `LStagg_prec` aus dem Modul `type_def` entsprechend geändert. Der aktuelle Wert dieser Variablen `LStagg_prec` wird durch die folgende Funktion geliefert:

function `get_LStagg_prec` : `integer`; (`type_def`)

Präzisionsanpassung
Mit Hilfe der Funktion

function `change_length` (**var** v : `LStaggrvector`) :
(lstgmv) `LStaggrvector[lb(v)..ub(v),-1..LStagg_prec];`

kann der Vektor `v`, mit `ub(v,2)` $\neq$ `LStagg_prec` durch die Anweisung

`x := change_length(v);` $\rightsquigarrow$ `v` $\subseteq$ `x` : `LStaggrvector`

an die aktuelle, durch `LStagg_prec` gegebene Präzision des Vektors `x` angepaßt werden, wobei die Dimensionen der beider Vektoren übereinstimmen müssen.

Mit Hilfe der Funktion

function `change_length` (**var** m: `LStaggrmatrix`): Lstaggrmatrix
`[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`; (lstgmv)

kann die Matrix `m`, mit `ub(m,3)` $\neq$ `LStagg_prec` durch die Anweisung

`x := change_length(m);` $\rightsquigarrow$ `m` $\subseteq$ `x` : `LStaggrmatrix`

an die aktuelle, durch `LStagg_prec` gegebene Präzision der Matrix `x` angepaßt werden, wobei die Dimensionen der beider Matrizen übereinstimmen müssen.

4.4.2.3 Hilfsfunktionen

procedure `mul` (**var** A,B,C : `Lrmatrix`;
(stgmv) **var** R : `LStaggrmatrix`;

Mit den Eingangsmatrizen `A,B,C` liefert die Ausgangsgröße `R` eine Einschießung des Produktes `A` $\cdot$ `B` $\cdot$ `C` $\in$ `R`.

4.4.2.4 Standardfunktionen

function `inf` (**var** v : `LStaggrvector`) :
(lstgmv) `LStaggrvector[lb(v)..ub(v),-1..LStagg_prec]`;

function `sup` (**var** v : `LStaggrvector`) :
(lstgmv) `LStaggrvector[lb(v)..ub(v),-1..LStagg_prec]`;

liefern Punkteinschließungen der Unter- bzw. Obergrenze des Vektors `v`; die Bedingung `ub(v,2)` = `Stagg_prec` muß erfüllt sein.

function `inf` (**var** m : `LStaggrmatrix`) : `LStaggrmatrix`
(lstgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

function `sup` (**var** m : `LStaggrmatrix`) : `LStaggrmatrix`
(lstgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

liefern Punkteinschließungen der Unter- bzw. Obergrenze der Matrix `m`, wobei die Bedingung `ub(m,3)` = `LStagg_prec` erfüllt sein muß.

function `mid` (**var** v : `LStaggrvector`) :
(lstgmv) `LStaggrvector[lb(v)..ub(v),-1..LStagg_prec]`;

function `mid` (**var** m : `LStaggrmatrix`) : `LStaggrmatrix`
(lstgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

liefern eine Punkteinschließung des gerundeten Mittelpunktes von `v` bzw. `m`, wobei gelten muß: $\mathtt{ub}(\mathtt{v},2) = \mathtt{Stagg\_prec} = \mathtt{ub}(\mathtt{m},3)$.

function `null` (**var** v : `LStaggrvector`) :
(lstgmv) `LStaggrvector[lb(v)..ub(v),-1..LStagg_prec]`;

function `null` (**var** m : `LStaggrmatrix`) : `LStaggrmatrix`
(lstgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

liefern einen staggered Nullvektor bzw. eine staggered Nullmatrix mit dem aktuellen Indexbereich von `v` bzw. `m`, wobei jedoch die Präzision der Null–Elemente durch `LStagg_prec` bestimmt wird. Für die Funktions–Argumente `v,m` ist $\mathtt{ub}(\mathtt{v},2) \neq \mathtt{LStagg\_prec} \neq \mathtt{ub}(\mathtt{m},3)$ zulässig. `ub(v,2)` ist dabei die Präzision der staggered Elemente des Vektors `v`, und `ub(m,3)` ist die Präzision der staggered Elemente der Matrix `m`.

function `id` (**var** m : `LStaggrmatrix`) : `LStaggrmatrix`
(lstgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

liefert eine staggered Einheitsmatrix mit dem aktuellen Indexbereich von `m`, wobei jedoch die Präzision der gelieferten staggered Matrix–Elemente durch `LStagg_prec` bestimmt wird. $\mathtt{ub}(\mathtt{m},3) \neq \mathtt{LStagg\_prec}$ ist zulässig.

function `transp` (**var** m : `LStaggrmatrix`) : `LStaggrmatrix`
(lstgmv) `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

liefert die transponierte Matrix (Zeilen- und Spalten-Vertauschung), wobei die Bedingung $\mathtt{ub}(\mathtt{m},3) = \mathtt{LStagg\_prec}$ erfüllt sein muß. `ub(m,3)` ist die Präzision der staggered Elemente von `m`.

4.4.2.5 Transferfunktionen

```
procedure TRANS ( var sv : LStaggrvector;
(lstgmv)          var v  : Lrvector);
```

Zum Eingangsvektor `sv` liefert die Ausgangsgröße `v` den maximalgenauen Mittelpunkts–Vektor von `sv`.

```
procedure TRANS ( var sv    : LStaggrvector;
(lstgmv)          var v1,v2 : Lrvector);
```

Zum Eingangsvektor `sv` liefern die Ausgangsgrößen `v1,v2` mit `v1` + `v2` ∈ `sv` den maximalgenauen Mittelpunkts–Vektor von `sv` in doppelter Länge.

```
procedure TRANS ( var sm : LStaggrmatrix;
(lstgmv)          var m  : Lrmatrix);
```

Zur Eingangsmatrix `sm` liefert die Ausgangsgröße `m` die maximalgenaue Mittelpunkts–Matrix von `sm`.

```
procedure TRANS ( var sm    : LStaggrmatrix;
(lstgmv)          var m1,m2 : Lrmatrix);
```

Zur Eingangsmatrix `sm` liefern die Ausgangsgrößen `m1,m2` mit `m1` + `m2` ∈ `sm` die maximalgenaue Mittelpunkts–Matrix von `sm` in doppelter Länge.

4.4.2.6 Zuweisungsoperatoren

Zuweisung	Bedeutung	
`Lsvv := Lsv`	`Lsvv` = `Lsv`;	falls `ub(Lsvv,2)` = `ub(Lsv,2)`
`Lsvv := r`	`Lsvv` = `r`;	`ub(Lsvv,2)` $\neq$ `LStagg_prec` ist zulässig
`Lsvv := Ls`	`Lsvv` = `Ls`;	falls `ub(Lsvv,2)` = `ub(Ls)`
`Lsvv := Lrv`	`Lsvv` = `Lrv`;	`ub(Lsvv,2)` $\neq$ `LStagg_prec` ist zulässig
`Livv := Lsv`	`Lsv` $\subseteq$ `Livv`	`ub(Lsv,2)` $\neq$ `LStagg_prec` ist zulässig
`Lrvv := Lsv`	Mitte von `Lsv`;	`ub(Lsv,2)` $\neq$ `LStagg_prec` ist zulässig
`Lsmv := Lsm`	`Lsmv` = `Lsm`;	falls `ub(Lsmv,3)` = `ub(Lsm,3)`
`Lsmv := r`	`Lsmv` = `r`;	`ub(Lsmv,3)` $\neq$ `LStagg_prec` ist zulässig
`Lsmv := Ls`	`Lsmv` = `Ls`;	falls `ub(Lsmv,3)` = `ub(Ls)`
`Lsmv := Lrm`	`Lsmv` = `Lrm`;	`ub(Lsmv,3)` $\neq$ `LStagg_prec` ist zulässig
`Limv := Lsm`	`Lsm` $\subseteq$ `Limv`	`ub(Lsm,3)` $\neq$ `LStagg_prec` ist zulässig
`Lrmv := Lsm`	Mitte von `Lsm`;	`ub(Lsm,3)` $\neq$ `LStagg_prec` ist zulässig

Tabelle 4.3: Lrv = `Lrvector`–Ausdruck, Lrvv = `Lrvector`–Variable, Lsvv = `LStaggrvector`–Variable, Lsv = `LStaggrvector`–Ausdruck, r = `Longreal/real/integer`–Ausdruck, Ls = `LStagg`–Ausdruck, Livv = `Livector`–Variable, Limv = `Limatrix`–Variable, Lrm = `Lrmatrix`–Ausdruck, Lrmv = `Lrmatrix`–Variable, Lsmv = `LStaggrmatrix`–Variable, Lsm = `LStaggrmatrix`–Ausdruck

Anmerkungen:

- `ub(Lsvv,2)` bedeutet die Obergrenze des zweiten Indexbereiches von `Lsvv` und ist damit der Präzisionsparameter dieser `LStaggrvector`–Variablen.
- `ub(Lsmv,3)` bedeutet die Obergrenze des dritten Indexbereiches von `Lsmv` und ist damit der Präzisionsparameter dieser `LStaggrmatrix`–Variablen.
- Mit `Lrvv` := `Lsv` liefert `Lrvv` nur die ins *real*–Format gerundete Mitte vom `LStaggrvector`–Ausdruck. Analoges gilt für `Lrmv` := `Lsm`.

4.4.2.7 Vergleichs- und Arithmetik-Operatoren

rechter Operand / **linker Operand**	real Longreal	LStagg	LStagg-rvector	LStagg-rmatrix	Lrvector	Lrmatrix
monadisch			$+, -$	$+, -$		
real Longreal			$*$	$*$		
LStagg			$*$	$*$	$*$	$*$
LStagg-rvector	$*, /$	$*, /$	**lies_in** **in**, $=, <>$ $+, -, *$		$+, -, *$ $=, <>$	
LStagg-rmatrix	$*, /$	$*, /$	$*$	**lies_in** **in**, $=, <>$ $+, -, *$	$*$	$+, -, *$ $=, <>$
Lrvector		$*, /$	**in**, $=, <>$ $+, -, *$			
Lrmatrix		$*, /$	$*$	**in**, $=, <>$ $+, -, *$		

Tabelle 4.4: Die Operatoren des Moduls `lstgmv`

Weitere Anmerkungen:
In obiger Tabelle ist der Ergebnistyp der arithmetischen Operatoren $\{+, -, *, /\}$ durch die üblichen Matrix/Vektor–Operationen bestimmt.
Bei den arithmetischen Operatoren kann die Präzision beider Operanden unabhängig vom Wert der Variablen `LStagg_prec` gewählt werden. Die Präzision des Operator–Ergebnisses ist jedoch stets durch den Wert der Variablen `LStagg_prec` gegeben.
in, **lies_in**, $=$, $<>$ sind Operatoren vom Ergebnistyp `boolean`.
`u` **lies_in** `v` $\longleftrightarrow$ `u` $\subseteq$ `v`; `ub(u,2)=ub(v,2)=LStagg_prec;`
Der Operator **in** testet die Relation *liegt in* z.B. zwischen einem `Lrvector`– und einem `LStaggrvector`–Operanden oder die Relation *liegt im Innern* zwischen zwei `LStaggrvector`–Operanden. Für Matrizen gelten bezüglich **in** ganz entsprechende Aussagen.

4.4.2.8 Ein-/Ausgabeprozeduren

Das Modul `lstgmv` exportiert die folgenden Ein-/Ausgabe–Prozeduren:

procedure `read` (**var** F : `Text`; **var** v: `LStaggrvector`);

procedure `read` (**var** F : `Text`; **var** m: `LStaggrmatrix`);

procedure `write` (**var** F : `Text`; **var** v: `LStaggrvector`);

procedure `write` (**var** F : `Text`; **var** m: `LStaggrmatrix`);

Die obigen Routinen stehen dem Anwender mit optionalem Fileparameter, beliebig vielen Ein- bzw. Ausgabe-Parametern, jedoch ohne Formatspezifikationen zur Verfügung.

Die Eingabe eines staggered Vektors oder einer staggered Matrix erfolgt im interaktiven Modus komponentenweise, wobei die Präzision dieser Komponenten durch die Vereinbarung der entsprechenden staggered Variablen vorgegeben ist. Die Eingabe einer Matrix erfolgt dabei wie üblich zeilenweise. Bitte beachten Sie, daß bei der Eingabe eines einzelnen staggered Elements dessen Komponenten und das anschließende Intervall genau so übernommen werden, wie der Anwender diese Werte eingegeben hat. Vergleichen Sie dazu auch die entsprechenden Anmerkungen auf Seite 109.

Die Ausgabe eines staggered Vektors oder einer staggered Matrix erfolgt wie bei der Eingabe komponentenweise in dem auf Seite 110 beschriebenen Format, wobei sich die staggered Komponenten eines einzelnen Elements nicht überlappen und einerlei Vorzeichen besitzen. Die ausgegebenen Elemente schließen die berechneten Matrix- oder Vektor-Komponenten garantiert ein, wobei die berechneten Elemente durch die Ausgabe natürlich nicht verändert werden.

4.5 Komplexe staggered Arithmetik

Mit Hilfe der auch in der BCD–Version von **PASCAL-XSC** zur Verfügung stehenden Lattenkreuzausdrücke (#-Ausdrücke) können beliebige Matrix- und Vektor–Operationen bzgl. der Standardtypen *rvector, ivector, cvector, civector, rmatrix, imatrix, cmatrix* und *cimatrix* im Akkumulator rundungsfehlerfrei berechnet und anschließend mit nur einer einzigen Rundung ins *real*-Format transformiert werden.

Für den Basistyp *Longreal* sind die entsprechenden Lattenkreuzausdrücke nicht vorgesehen. Im Bedarfsfall lassen sich jedoch beliebige Matrix- und Vektor–Operationen auch bzgl. der analogen Standardtypen *Lrvector, Livector, Lcvector, Lcivector, Lrmatrix, Limatrix, Lcmatrix* und *Lcimatrix* mit Hilfe des Datentyps *dotprecision* ebenfalls rundungsfehlerfrei berechnen und anschließend wieder mit nur einer einzigen Rundung ins *Longreal*-Format zurücktransformieren. Die für die Praxis wichtigsten Matrix/Vektor–Ausdrücke lassen sich mit Hilfe entsprechender Routinen auswerten, die von den Hilfsmodulen `mv_comp` bzw. `mvc_comp` exportiert werden. Zur näheren Information findet man auf den Seiten 169 bzw. 176 jeweils eine Zusammenstellung der zur Verfügung stehenden Matrix-Vektor-Ausdrücke.

Damit beseht die Möglichkeit, für die beiden Basistypen *real* und *Longreal* beliebige komplexe Matrix/Vektor–Ausdrücke maximalgenau auszuwerten. In der numerischen Praxis reichen die maximalgenauen Rechnungen in einfacher Länge oft jedoch nicht aus, so daß z.B. Matrixinversionen in schlecht-konditionierten Fällen in doppelter oder sogar dreifacher Länge des benutzten Datenformats durchgeführt werden müssen.

In den nachfolgenden Abschnitten werden für eine komplexe staggered Arithmetik alle notwendigen Funktionen, Prozeduren und Operatoren bezüglich der Basistypen *real* und *Longreal* zusammengestell. Da die komplexen Matrix/Vektor–Operatoren für Intervall–Operanden definiert werden, lassen sich mit Hilfe der komplexen staggered Arithmetik auch Einschließungs–Algorithmen sehr einfach realisieren. Im Kapitel 3 ab Seite 61 sind die allgemeinen Grundbegriffe einer staggered Arithmetik kurz zusammengestellt.

4.5.1 Der Basistyp `real`

Wir betrachten jetzt das *staggered* Format zum Basistyp *real*. Beachten Sie bitte dazu auch die Einführung auf Seite 61. Das dortige Programm `stagg1` liefert bereits wichtige Informationen für die ersten Programmierschritte. Die für die staggered Matrix/Vektor–Arithmetik notwendigen Datentypen sowie die erforderlichen Operatoren, Funktionen und Prozeduren werden von den

Hilfs–Modulen `type_def` und `smvc` exportiert. Ein geklammerter Name (xxx) in einer der nachfolgenden Einrahmungen bezeichnet das Hilfsmodul, welches die jeweilige Routine exportiert.

4.5.1.1 Datenstruktur

Datentyp

type Staggcvector = **dynamic array**[*] **of** Staggcomplex; (`type_def`)

type Staggcmatrix = **dynamic array**[*] **of** Staggcvector; (`type_def`)

Präzisionsparameter

var Stagg_prec : integer; (`type_def`)

Variablen-Vereinbarung

```
var v : Staggcvector[1..3,1..2,-1..Stagg_prec];
```

`v` ist ein Vektor vom Typ `Staggcvector` mit drei komplexen Komponenten vom Typ `Staggcomplex`, deren Präzision durch die Variable `Stagg_prec` vorgegeben ist. Der Indexbereich `1..2` definiert den Realteil(`1`) und den Imaginärteil(`2`) der komplexen Elemente.

```
var m : Staggcmatrix[1..4,1..3,1..2,-1..Stagg_prec];
```

`m` ist eine komplexe Matrix vom Typ `Staggcmatrix` mit vier Zeilen und drei Spalten . Die Präzision der Elemente ist für Real- und Imaginärteil durch die Variable `Stagg_prec` vorgegeben. Der Indexbereich `1..2` definiert den Realteil(`1`) und den Imaginärteil(`2`) der komplexen Elemente. Beachten Sie bitte, daß der Anwender den gewünschten Wert von `Stagg_prec` vor der Vereinbarung der obigen Variablen `v,m` festlegen muß.

Die Vereinbarung eines Vektors vom Typ `Staggcvector` und einer Matrix vom Typ `Staggcmatrix` ist jedoch auch innerhalb einer Prozedur ohne die Variable `Stagg_prec` wie folgt möglich:

```
procedure beispiel( n : integer );
var m : Staggcmatrix;
    v : Staggcvector;
begin
   allocate(m,1..4,1..3,1..2,-1..2*n);
   allocate(v,1..3,1..2,-1..n);
   .....  Anweisungen  .....
   free(m);  free(v);
end;  { beispiel }
```

4.5.1.2 Präzisionsänderung

Eine Präzisionsänderung kann mit Hilfe der Prozedur `set_Stagg_prec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_Stagg_prec` (**prec** : `integer`); (`type_def`)

Mit Hilfe der Eingangsgröße `prec` wird durch den Prozeduraufruf

```
set_Stagg_prec(prec)
```

die Variable `Stagg_prec` aus dem Modul `type_def` entsprechend geändert. Der aktuelle Wert dieser Variablen `Stagg_prec` wird durch die folgende Funktion geliefert:

function `get_Stagg_prec` : `integer`; (`type_def`)

Präzisionsanpassung
Mit Hilfe der Funktion

function `change_length` (**var** v : `Staggcvector`) : (smvc) `Staggcvector[lb(v)..ub(v),1..2,-1..Stagg_prec]`;

kann der Vektor `v`, mit `ub(v,3)` $\neq$ `Stagg_prec` durch die Anweisung

`x := change_length(v);` $\rightsquigarrow$ `v` $\subseteq$ `x` : `Staggcvector` an die aktuelle, durch `Stagg_prec` gegebene Präzision des Vektors `x` angepaßt werden, wobei die Dimensionen beider Vektoren übereinstimmen müssen.

Mit Hilfe der Funktion

> **function** `change_length` (**var** m : `Staggcmatrix`) : Staggcmatrix
> `[lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..Stagg_prec]`;

kann die Matrix `m`, mit `ub(m,4)` $\neq$ `Stagg_prec` durch die Anweisung

`x := change_length(m);` $\rightsquigarrow$ `m` $\subseteq$ `x` : `Staggcmatrix`

an die aktuelle, durch `Stagg_prec` gegebene Präzision der Matrix `x` angepaßt werden, wobei die Dimensionen beider Matrizen übereinstimmen müssen.

4.5.1.3 Hilfsfunktionen

> **procedure** `mul` (**var** A,B,C : `cmatrix`;
> (smvc) **var** R : `Staggcmatrix`;

Mit den Eingangsmatrizen `A,B,C` liefert die Ausgangsgröße `R` eine Einschießung des Produktes `A` $\cdot$ `B` $\cdot$ `C` $\in$ `R`.

4.5.1.4 Transferfunktionen

function `re` (**var** v : `Staggcvector`) : `Staggrvector[lb(v)..ub(v),-1..Stagg_prec]`; (smvc)

function `im` (**var** v : `Staggcvector`) : `Staggrvector[lb(v)..ub(v),-1..Stagg_prec]`; (smvc)

function `re` (**var** m : `Staggcmatrix`) : `Staggrmatrix [lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec]`; (smvc)

function `im` (**var** m : `Staggcmatrix`) : `Staggrmatrix [lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec]`; (smvc)

liefern Real- bzw. Imaginärteil von `v,m`, wobei die folgenden Bedingungen `ub(m,4)` = `Stagg_prec` = `ub(v,3)` erfüllt sein müssen.

function `null` (**var** v : `Staggcvector`) : `Staggrvector[lb(v)..ub(v),-1..Stagg_prec]`; (smvc)

function `null` (**var** m : `Staggcmatrix`) : `Staggrmatrix [lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec]`; (smvc)

liefern Nullvektor bzw. Nullmatrix mit dem aktuellen Indexbereich von `v` bzw. `m`. `ub(m,4)` $\neq$ `Stagg_prec` $\neq$ `ub(v,3)` ist zulässig. Die Präzision der Funktionswerte ist jedoch durch `Stagg_prec` bestimmt.

```
function id ( var m : Staggcmatrix ) : Staggrmatrix        (smvc)
        [lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec];
```

liefert die Einheitsmatrix mit dem aktuellen Indexbereich der Matrix `m`. `ub(m,4)` $\neq$ `Stagg_prec` ist zulässig. Die Präzision der Funktionswerte ist jedoch durch `Stagg_prec` bestimmt.

Bei den folgenden vier Prozeduren `TRANS` muß die Präzision der Elemente von `sv,sm` nicht durch `Stagg_prec` vorgegeben sein.

```
procedure TRANS ( var sv : Staggcvector;
(smvc)            var v  : cvector);
```

Zum Eingangsvektor `sv` liefert die Ausgangsgröße `v` den maximalgenauen Mittelpunkts–Vektor von `sv`.

```
procedure TRANS ( var sv    : Staggcvector;
(smvc)            var v1,v2 : cvector);
```

Zum Eingangsvektor `sv` liefern die Ausgangsgrößen `v1,v2` mit `v1` + `v2` $\in$ `sv` den maximalgenauen Mittelpunkts–Vektor von `sv` in doppelter Länge.

```
procedure TRANS ( var sm : Staggcmatrix;
(smvc)            var m  : cmatrix);
```

Zur Eingangsmatrix `sm` liefert die Ausgangsgröße `m` die maximalgenaue Mittelpunkts–Matrix von `sm`.

```
procedure TRANS ( var sm    : Staggcmatrix;
(smvc)            var m1,m2 : cmatrix);
```

Zur Eingangsmatrix `sm` liefern die Ausgangsgrößen `m1,m2` mit `m1` + `m2` $\in$ `sm` die maximalgenaue Mittelpunkts–Matrix von `sm` in doppelter Länge.

4.5.1.5 Standardfunktionen

function `inf` (**var** v : `Staggcvector`) : (smvc)
`Staggcvector[lb(v)..ub(v),1..2,-1..Stagg_prec]`;

function `sup` (**var** v : `Staggcvector`) : (smvc)
`Staggcvector[lb(v)..ub(v),1..2,-1..Stagg_prec]`;

liefern Punkteinschließungen der Unter- bzw. Obergrenze des Vektors `v`; die Bedingung `ub(v,3)` = `Stagg_prec` muß erfüllt sein.

function `inf` (**var** m : `Staggcmatrix`) : `Staggcmatrix` (smvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..Stagg_prec]`;

function `sup` (**var** m : `Staggcmatrix`) : `Staggcmatrix` (smvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..Stagg_prec]`;

liefern Punkteinschließungen der Unter- bzw. Obergrenze der Matrix `m`, wobei die Bedingung `ub(m,4)` = `Stagg_prec` erfüllt sein muß.

function `mid` (**var** v : `Staggcvector`) : (smvc)
`Staggcvector[lb(v)..ub(v),1..2,-1..Stagg_prec]`;

function `mid` (**var** m : `Staggcmatrix`) : `Staggcmatrix` (smvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..Stagg_prec]`;

liefern Punkteinschließungen der gerundeten Mittelpunkte der Elemente von `v,m`, wobei `ub(m,4)` = `Stagg_prec` = `ub(v,3)` erfüllt sein muß.

function `conj` (**var** v : `Staggcvector`) : `Staggcvector[lb(v)..ub(v),1..2,-1..Stagg_prec];` (smvc)

function `conj` (**var** m : `Staggcmatrix`) : `Staggcmatrix [lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..Stagg_prec];` (smvc)

liefert die konjugiert komplexen Elemente von `v,m`, wobei die Bedingung `ub(m,4)` = `Stagg_prec` = `ub(v,3)` erfüllt sein muß.

function `transp` (**var** m : `Staggcmatrix`) : `Staggcmatrix [lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..Stagg_prec];`

liefert bzgl. `m` die transponierte Matrix durch Zeilen- und Spaltenvertauschung, wobei die Bedingung `ub(m,4)` = `Stagg_prec` erfüllt sein muß.

function `herm` (**var** m : `Staggcmatrix`) : `Staggcmatrix [lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..Stagg_prec];`

liefert das Ergebnis des Hermiteschen Matrix-Operators H, falls folgende Bedingung `ub(m,4)` = `Stagg_prec` erfüllt ist.

4.5.1.6 Zuweisungsoperatoren

Zuweisung	**Bedeutung**
Scvv := Scv	Scvv = Scv; falls ub(Scvv,3) = ub(Scv,3)
Scvv := r	Scvv = r; ub(Scvv,3) $\neq$ Stagg_prec zulässig
Scvv := c	Scvv = c; ub(Scvv,3) $\neq$ Stagg_prec zulässig
Scvv := S	Scvv = S; falls ub(Scvv,3) = ub(S)
Scvv := Sc	Scvv = Sc; falls ub(Scvv,3) = ub(Sc,2)
Scvv := rv	Scvv = rv; ub(Scvv,3) $\neq$ Stagg_prec zulässig
Scvv := cv	Scvv = cv; ub(Scvv,3) $\neq$ Stagg_prec zulässig
Scvv := Srv	Scvv = Srv; falls ub(Scvv,3) = ub(Srv,2)
cvv := Scv	Mitte v. Scv ub(Scv,3) $\neq$ Stagg_prec zulässig
civ := Scv	Scv $\subseteq$ civ ub(Scv,3) $\neq$ Stagg_prec zulässig
Scmv := Scm	Scmv = Scm; falls ub(Scmv,4) = ub(Scm,4)
Scmv := r	Scmv = r; ub(Scmv,4) $\neq$ Stagg_prec zulässig
Scmv := c	Scmv = c; ub(Scmv,4) $\neq$ Stagg_prec zulässig
Scmv := S	Scmv = S; falls ub(Scmv,4) = ub(S)
Scmv := Sc	Scmv = Sc; falls ub(Scmv,4) = ub(Sc,2)
Scmv := rm	Scmv = rm; ub(Scmv,4) $\neq$ Stagg_prec zulässig
Scmv := cm	Scmv = cm; ub(Scmv,4) $\neq$ Stagg_prec zulässig
Scmv := Srm	Scmv = Srm; falls ub(Scmv,4) = ub(Srm,3)
cmv := Scm	Mitte v. Scm ub(Scm,4) $\neq$ Stagg_prec zulässig
cimv := Scm	Scm $\subseteq$ cimv ub(Scm,4) $\neq$ Stagg_prec zulässig

Zuweisungsoperatoren des Moduls smvc. Es bedeuten:

Scvv: Staggcvector–Variable, Scmv: Staggcmatrix–Variable, cvv: cvector–Variable, civ: civector–Variable, cmv: cmatrix–Variable, cimv: cimatrix–Variable, Srv: Staggrvector–Ausdruck, Scv: Staggcvector–Ausdruck, Srm: Staggrmatrix–Ausdruck, Scm: Staggcmatrix–Ausdruck, Sc: Staggcomplex–Ausdruck, S: Stagg–Ausdruck, c: complex–Ausdruck, r: real/integer–Ausdruck, rv: rvector–Ausdruck, cv: cvector–Ausdruck, rm: rmatrix–Ausdruck, cm: cmatrix–Ausdruck;

4.5.1.7 Vergleichs- und Arithmetik-Operatoren

rechter Operand	integer real	Stagg	complex	Stagg-complex	rvector	cvector
linker Operand						
monadisch						
integer real						
Stagg						$*$
complex						
Stagg-complex					$*$	$*$
rvector				$*, /$		
cvector		$*, /$		$*, /$		
rmatrix				$*, /$		
cmatrix		$*, /$		$*, /$		
Stagg-rvector			$*, /$	$*, /$		$+, -, *$
Stagg-cvector	$*, /$	$*, /$	$*, /$	$*, /$	$+, -, *$	$+, -, *$
Stagg-rmatrix			$*, /$	$*, /$		$*$
Stagg-cmatrix	$*, /$	$*, /$	$*, /$	$*, /$	$*$	$*$

Operatoren des Moduls `smvc`, Teil 1.

Anmerkungen:

- Die Präzisionen der arithmetischen Operanden können unabhängig voneinander gewählt werden, die Präzision des Operatorergebnisses wird jedoch von der Variablen `Stagg_prec` bestimmt.
- Der Operator **in** testet die Relation *liegt im Innern* zwischen zwei Intervall–Operanden, und **lies_in** testet die Relation *ist enthalten in*.

Vergleichs- und Arithmetik-Operatoren, Teil 2.

rechter Operand	rmatrix	cmatrix	Stagg-rvector	Stagg-cvector	Stagg-rmatrix	Stagg-cmatrix
linker Operand						
monadisch				$+, -$		$+, -$
integer real				$*$		$*$
Stagg		$*$		$*$		$*$
complex			$*$	$*$	$*$	$*$
Stagg-complex	$*$	$*$	$*$	$*$	$*$	$*$
rvector				**in**, $\diamond$		
cvector			$\diamond$	**in**, $\diamond$		
rmatrix				$*$		**in**, $\diamond$
cmatrix			$*$	$*$	$*$	**in**, $\diamond$
Stagg-rvector				**lies_in** $=, <>$ **in**, $\diamond$		
Stagg-cvector			$\diamond$ $=, <>$	**lies_in** $=, <>$ **in**, $\diamond$		
Stagg-rmatrix		$*$		$*$		**lies_in** $=, <>$ **in**, $\diamond$
Stagg-cmatrix	$\diamond$	$\diamond$	$*$	$*$	$*$ $=, <>$	**lies_in** $=, <>$ **in**, $\diamond$

Operatoren des Moduls `smvc`, Teil 2. $\diamond \in \{+, -, *\}$

4.5.1.8 Ein-/Ausgabeprozeduren

Das Modul `smvc` exportiert die folgenden Ein-/Ausgabe–Prozeduren:

procedure `read` (**var** F : `Text`; **var** v: `Staggcvector`);

procedure `read` (**var** F : `Text`; **var** m: `Staggcmatrix`);

procedure `write` (**var** F : `Text`; **var** v: `Staggcvector`);

procedure `write` (**var** F : `Text`; **var** m: `Staggcmatrix`);

Die obigen Routinen stehen dem Anwender mit optionalem Fileparameter, beliebig vielen Ein- bzw. Ausgabe-Parametern, jedoch ohne Formatspezifikationen zur Verfügung.

Die Eingabe eines staggered Vektors oder einer staggered Matrix erfolgt im interaktiven Modus komponentenweise, wobei die Präzision dieser Komponenten durch die Vereinbarung der entsprechenden staggered Variablen vorgegeben ist. Die Eingabe einer Matrix erfolgt dabei wie üblich zeilenweise. Bitte beachten Sie, daß bei der Eingabe eines einzelnen staggered Elements dessen Komponenten und das anschließende Intervall genau so übernommen werden, wie der Anwender diese Werte eingegeben hat. Vergleichen Sie dazu auch die entsprechenden Anmerkungen auf Seite 145.

Die Ausgabe eines staggered Vektors oder einer staggered Matrix erfolgt wie bei der Eingabe komponentenweise in dem auf Seite 145 beschriebenen Format, wobei sich die staggered Komponenten eines einzelnen Elements nicht überlappen und einerlei Vorzeichen besitzen. Die ausgegebenen Elemente schließen die berechneten Matrix- oder Vektor-Komponenten garantiert ein, wobei die berechneten Elemente durch die Ausgabe natürlich nicht verändert werden.

4.5.2 Der Basistyp `Longreal`

Wir betrachten jetzt das *staggered* Format zum Basistyp *Longreal*. Beachten Sie bitte dazu auch die Einführung auf Seite 61. Das dortige Programm `stagg1` liefert bereits wichtige Informationen für die ersten Programmierschritte. Die für die staggered Matrix/Vektor–Arithmetik notwendigen Datentypen sowie die erforderlichen Operatoren, Funktionen und Prozeduren werden von den Hilfs–Modulen `type_def` und `lsmvc` exportiert. Ein geklammerter Name (xxx) in einer der nachfolgenden Einrahmungen bezeichnet das Hilfsmodul, welches die jeweilige Routine exportiert.

4.5.2.1 Datenstruktur

Datentyp

type LStaggcvector = **dynamic array**[∗] **of** LStaggcomplex;

type LStaggcmatrix = **dynamic array**[∗] **of** LStaggcvector;

Präzisionsparameter

var LStagg_prec : integer; (`type_def`)

Variablen-Vereinbarung

```
var v : LStaggcvector[1..3,1..2,-1..Stagg_prec];
```

`v` ist ein Vektor vom Typ `LStaggcvector` mit drei komplexen Komponenten vom Typ `LStaggcomplex`, deren Präzision durch die Variable `LStagg_prec` vorgegeben ist. Der Indexbereich `1..2` definiert den Realteil(`1`) und den Imaginärteil(`2`) der komplexen Elemente.

```
var m : LStaggcmatrix[1..4,1..3,1..2,-1..LStagg_prec];
```

`m` ist eine komplexe Matrix vom Typ `LStaggcmatrix` mit vier Zeilen und drei Spalten . Die Präzision der Elemente ist für Real- und Imaginärteil durch die Variable `LStagg_prec` vorgegeben. Der Indexbereich `1..2` definiert den

Realteil(1) und den Imaginärteil(2) der komplexen Elemente. Beachten Sie bitte, daß der Anwender den gewünschten Wert von `LStagg_prec` vor der Vereinbarung der obigen Variablen `v,m` festlegen muß.

Die Vereinbarung einer Variablen vom Typ `LStaggcvector` oder vom Typ `LStaggcmatrix` ist jedoch auch innerhalb einer Prozedur ohne die Variable `LStagg_prec` wie folgt möglich:

```
procedure beispiel( n : integer );
var m : LStaggcmatrix;
    v : LStaggcvector;
begin
   allocate(m,1..4,1..3,1..2,-1..2*n);
   allocate(v,1..3,1..2,-1..n);
   .....  Anweisungen  .....
   free(m);  free(v);
end;  { beispiel }
```

4.5.2.2 Präzisionsänderung

Eine Präzisionsänderung kann mit Hilfe der Prozedur `set_LStagg_prec` vorgenommen werden. Prozedurvereinbarung:

procedure `set_LStagg_prec` (**prec** : `integer`); (`type_def`)

Mit Hilfe der Eingangsgröße `prec` wird durch den Prozeduraufruf

`set_LStagg_prec(prec)`

die Variable `LStagg_prec` aus dem Modul `type_def` entsprechend geändert. Der aktuelle Wert dieser Variablen `LStagg_prec` wird durch die folgende Funktion geliefert:

function `get_LStagg_prec` : `integer`; (`type_def`)

Präzisionsanpassung

Mit Hilfe der Funktion

function `change_length` (**var** v : `LStaggcvector`) : (lsmvc) `LStaggcvector[lb(v)..ub(v),1..2,-1..LStagg_prec]`;

kann der Vektor `v`, mit `ub(v,3)` $\neq$ `LStagg_prec` durch die Anweisung

`x := change_length(v);` $\rightsquigarrow$ `v` $\subseteq$ `x` : `LStaggcvector`

an die aktuelle, durch `LStagg_prec` gegebene Präzision des Vektors `x` angepaßt werden, wobei die Dimensionen der Vektoren übereinstimmen müssen.

Mit Hilfe der Funktion

function `change_length`	(lsmvc)
(**var** m : `LStaggcmatrix`) : `LStaggcmatrix`	
`[lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..LStagg_prec];`	

kann die Matrix `m`, mit `ub(m,4)` $\neq$ `LStagg_prec` durch die Anweisung

`x := change_length(m);` $\rightsquigarrow$ `m` $\subseteq$ `x` : `LStaggcmatrix`

an die aktuelle, durch `LStagg_prec` gegebene Präzision der Matrix `x` angepaßt werden, wobei die Dimensionen der Matrizen übereinstimmen müssen.

4.5.2.3 Hilfsfunktionen

procedure `mul`	(**var** A,B,C : `Lcmatrix`;
(lsmvc)	**var** R : `LStaggcmatrix`;

Mit den Eingangsmatrizen `A,B,C` liefert die Ausgangsgröße `R` eine Einschießung des Produktes `A` $\cdot$ `B` $\cdot$ `C` $\in$ `R`.

4.5.2.4 Transferfunktionen

function `re` (**var** v : `LStaggcvector`) : (lsmvc)
`LStaggrvector[lb(v)..ub(v),-1..LStagg_prec]`;

function `im` (**var** v : `LStaggcvector`) : (lsmvc)
`LStaggrvector[lb(v)..ub(v),-1..Stagg_prec]`;

function `re` (**var** m : `LStaggcmatrix`) : `LStaggrmatrix` (lsmvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

function `im` (**var** m : `LStaggcmatrix`) : `LStaggrmatrix` (lsmvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..Stagg_prec]`;

liefern Real- bzw. Imaginärteil von `v,m`, wobei die folgenden Bedingungen `ub(m,4)` = `LStagg_prec` = `ub(v,3)` erfüllt sein müssen.

function `null` (**var** v : `LStaggcvector`) : (lsmvc)
`LStaggrvector[lb(v)..ub(v),-1..LStagg_prec]`;

function `null` (**var** m : `LStaggcmatrix`) : `LStaggrmatrix`
`(lsmvc)` `[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

liefern Nullvektor bzw. Nullmatrix mit dem aktuellen Indexbereich von `v` bzw. `m`. `ub(m,4)` $\neq$ `LStagg_prec` $\neq$ `ub(v,3)` ist zulässig. Die Präzision der Funktionswerte ist jedoch durch `LStagg_prec` bestimmt.

function `id` (**var** m : `LStaggcmatrix`) : `LStaggrmatrix` (lsmvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),-1..LStagg_prec]`;

liefert die Einheitsmatrix mit dem aktuellen Indexbereich der Matrix `m`. `ub(m,4)` $\neq$ `LStagg_prec` ist zulässig. Die Präzision der Funktionswerte ist jedoch durch `LStagg_prec` bestimmt.

Bei den folgenden vier Prozeduren `TRANS` muß die Präzision der Elemente von `sv,sm` nicht durch `LStagg_prec` vorgegeben sein.

procedure `TRANS` (**var** sv : `LStaggcvector`;
(lsmvc) **var** v : `Lcvector`);

Zum Eingangsvektor `sv` liefert die Ausgangsgröße `v` den maximalgenauen Mittelpunkts–Vektor von `sv`.

procedure `TRANS` (**var** sv : `LStaggcvector`;
(lsmvc) **var** v1,v2 : `Lcvector`);

Zum Eingangsvektor `sv` liefern die Ausgangsgrößen `v1,v2` mit `v1` + `v2` $\in$ `sv` den maximalgenauen Mittelpunkts–Vektor von `sv` in doppelter Länge.

procedure `TRANS` (**var** sm : `LStaggcmatrix`;
(lsmvc) **var** m : `Lcmatrix`);

Zur Eingangsmatrix `sm` liefert die Ausgangsgröße `m` die maximalgenaue Mittelpunkts–Matrix von `sm`.

procedure `TRANS` (**var** sm : `LStaggcmatrix`;
(lsmvc) **var** m1,m2 : `Lcmatrix`);

Zur Eingangsmatrix `sm` liefern die Ausgangsgrößen `m1,m2` mit `m1` + `m2` $\in$ `sm` die maximalgenaue Mittelpunkts–Matrix von `sm` in doppelter Länge.

4.5.2.5 Standardfunktionen

function `inf` (**var** v : `LStaggcvector`) : (lsmvc)
`LStaggcvector[lb(v)..ub(v),1..2,-1..LStagg_prec]`;

function `sup` (**var** v : `LStaggcvector`) : (lsmvc)
`LStaggcvector[lb(v)..ub(v),1..2,-1..LStagg_prec]`;

liefern Punkteinschließungen der Unter- bzw. Obergrenze des Vektors `v`; die Bedingung `ub(v,3)` = `LStagg_prec` muß erfüllt sein.

function `inf` (**var** m : `LStaggcmatrix`) : `Staggcmatrix` (lsmvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..LStagg_prec]`;

function `sup` (**var** m : `LStaggcmatrix`) : `Staggcmatrix` (lsmvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..LStagg_prec]`;

liefern Punkteinschließungen der Unter- bzw. Obergrenze der Matrix `m`, wobei die Bedingung `ub(m,4)` = `LStagg_prec` erfüllt sein muß.

function `mid` (**var** v : `LStaggcvector`) : (lsmvc)
`LStaggcvector[lb(v)..ub(v),1..2,-1..LStagg_prec]`;

function `mid` (**var** m : `LStaggcmatrix`) : `Staggcmatrix` (lsmvc)
`[lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..LStagg_prec]`;

liefern Punkteinschließungen der gerundeten Mittelpunkte der Elemente von `v,m`, wobei `ub(m,4)` = `LStagg_prec` = `ub(v,3)` erfüllt sein muß.

```
function conj ( var v : LStaggcvector ) :                    (lsmvc)
        LStaggcvector[lb(v)..ub(v),1..2,-1..LStagg_prec];
```

```
function conj ( var m: LStaggcmatrix ): LStaggcmatrix (lsmvc)
     [lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..Stagg_prec];
```

liefert die konjugiert komplexen Elemente von `v,m`, wobei die Bedingung `ub(m,4)` = `LStagg_prec` = `ub(v,3)` erfüllt sein muß.

```
function transp ( var m : LStaggcmatrix ) : LStaggcmatrix
    [lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..LStagg_prec];
```

liefert bzgl. `m` die transponierte Matrix durch Zeilen- und Spaltenvertauschung, wobei die Bedingung `ub(m,4)` = `LStagg_prec` erfüllt sein muß.

```
function herm ( var m : LStaggcmatrix ) : LStaggcmatrix
    [lb(m)..ub(m),lb(m,2)..ub(m,2),1..2,-1..LStagg_prec];
```

liefert das Ergebnis des Hermiteschen Matrix-Operators H, falls folgende Bedingung `ub(m,4)` = `LStagg_prec` erfüllt ist.

4.5.2.6 Zuweisungsoperatoren

Zuweisung	**Bedeutung**		
Scvv := Scv	Scvv = Scv;	falls ub(Scvv,3) = ub(Scv,3)	
Scvv := r	Scvv = r;	ub(Scvv,3) $\neq$ LStagg_prec	zulässig
Scvv := Lc	Scvv = Lc;	ub(Scvv,3) $\neq$ LStagg_prec	zulässig
Scvv := LS	Scvv = LS;	falls ub(Scvv,3) = ub(LS)	
Scvv := LSc	Scvv = LSc;	falls ub(Scvv,3) = ub(LSc,2)	
Scvv := Lrv	Scvv = Lrv;	ub(Scvv,3) $\neq$ LStagg_prec	zulässig
Scvv := Lcv	Scvv = Lcv;	ub(Scvv,3) $\neq$ LStagg_prec	zulässig
Scvv := LSrv	Scvv = LSrv;	falls ub(Scvv,3) = ub(LSrv,2)	
cvv := LScv	Mitte v. LScv	ub(LScv,3) $\neq$ LStagg_prec	zulässig
civ := LScv	LScv $\subseteq$ civ	ub(LScv,3) $\neq$ LStagg_prec	zulässig
Scmv := LScm	LScmv = Scm;	falls ub(Scmv,4) = ub(LScm,4)	
Scmv := r	Scmv = r;	ub(Scmv,4) $\neq$ LStagg_prec	zulässig
Scmv := Lc	Scmv = Lc;	ub(Scmv,4) $\neq$ LStagg_prec	zulässig
Scmv := LS	Scmv = LS;	falls ub(Scmv,4) = ub(LS)	
Scmv := LSc	Scmv = LSc;	falls ub(Scmv,4) = ub(LSc,2)	
Scmv := Lrm	Scmv = Lrm;	ub(Scmv,4) $\neq$ LStagg_prec	zulässig
Scmv := Lcm	Scmv = Lcm;	ub(Scmv,4) $\neq$ LStagg_prec	zulässig
Scmv := LSrm	Scmv = LSrm;	falls ub(Scmv,4) = ub(LSrm,3)	
cmv := LScm	Mitte v. LScm	ub(LScm,4) $\neq$ LStagg_prec	zulässig
cimv := LScm	LScm $\subseteq$ cimv	ub(LScm,4) $\neq$ LStagg_prec	zulässig

Zuweisungsoperatoren des Moduls `lsmvc`. Es bedeuten:

`Scvv`: LStaggcvector–Variable, `Scmv`: LStaggcmatrix–Variable, `cvv`: Lcvector–Variable, `civ`: Lcivector–Variable, `cmv`: Lcmatrix–Variable, `cimv`: Lcimatrix–Variable, `LSrv`: LStaggrvector–Ausdr., `LScv`: LStaggcvector–Ausdr., `LSrm`: LStaggrmatrix–Ausdr., `LScm`: LStaggcmatrix–Ausdruck, `LSc`: LStaggcomplex–Ausdruck, `LS`: LStagg–Ausdruck, `Lc`: Lcomplex–Ausdruck, `r`: integer/real/Longreal–Ausdruck, `Lrv`: Lrvector–Ausdruck, `Lcv`: Lcvector–Ausdruck, `Lrm`: Lrmatrix–Ausdruck, `Lcm`: Lcmatrix–Ausdruck;

4.5.2.7 Vergleichs- und Arithmetik-Operatoren

rechter Operand	real Longreal	LStagg	Lcomplex	LStagg-complex	Lrvector	Lcvector
linker Operand						
monadisch						
real Longreal						
LStagg						$*$
Lcomplex						
LStagg-complex					$*$	$*$
Lrvector				$*, /$		
Lcvector		$*, /$		$*, /$		
Lrmatrix				$*, /$		
Lcmatrix		$*, /$		$*, /$		
LStagg-rvector			$*, /$	$*, /$		$+, -, *$
LStagg-cvector	$*, /$	$*, /$	$*, /$	$*, /$	$+, -, *$	$+, -, *$
LStagg-rmatrix			$*, /$	$*, /$		$*$
LStagg-cmatrix	$*, /$	$*, /$	$*, /$	$*, /$	$*$	$*$

Operatoren des Moduls `lsmvc`, Teil 1.

Anmerkungen:

- Die Präzisionen der arithmetischen Operanden können unabhängig voneinander gewählt werden, die Präzision des Operatorergebnisses wird jedoch von der Variablen `LStagg_prec` bestimmt.
- Der Operator **in** testet die Relation *liegt im Innern* zwischen zwei Intervall–Operanden, und **lies_in** testet die Relation *ist enthalten in*.

Vergleichs- und Arithmetik-Operatoren, Teil 2.

rechter Operand	Lrmatrix	Lcmatrix	LStagg-rvector	LStagg-cvector	LStagg-rmatrix	LStagg-cmatrix
linker Operand						
monadisch				$+, -$		$+, -$
real Longreal				$*$		$*$
LStagg		$*$		$*$		$*$
Lcomplex			$*$	$*$	$*$	$*$
LStagg-complex	$*$	$*$	$*$	$*$	$*$	$*$
Lrvector				**in**, $\diamond$		
Lcvector			$\diamond$	**in**, $\diamond$		
Lrmatrix				$*$		**in**, $\diamond$
Lcmatrix			$*$	$*$	$*$	**in**, $\diamond$
LStagg-rvector				**lies_in** =, <> **in**, $\diamond$		
LStagg-cvector			$\diamond$ =, <>	**lies_in** =, <> **in**, $\diamond$		
LStagg-rmatrix		$*$		$*$		**lies_in** =, <> **in**, $\diamond$
LStagg-cmatrix	$\diamond$	$\diamond$	$*$	$*$	$*$ =, <>	**lies_in** =, <> **in**, $\diamond$

Operatoren des Moduls `lsmvc`, Teil 2. $\diamond \in \{+, -, *\}$

4.5.2.8 Ein-/Ausgabeprozeduren

Das Modul `lsmvc` exportiert die folgenden Ein-/Ausgabe–Prozeduren:

procedure `read` (**var** F : `Text`; **var** v: `LStaggcvector`);

procedure `read` (**var** F : `Text`; **var** m: `LStaggcmatrix`);

procedure `write` (**var** F : `Text`; **var** v: `LStaggcvector`);

procedure `write` (**var** F : `Text`; **var** m: `LStaggcmatrix`);

Die obigen Routinen stehen dem Anwender mit optionalem Fileparameter, beliebig vielen Ein- bzw. Ausgabe-Parametern, jedoch ohne Formatspezifikationen zur Verfügung.

Die Eingabe eines staggered Vektors oder einer staggered Matrix erfolgt im interaktiven Modus komponentenweise, wobei die Präzision dieser Komponenten durch die Vereinbarung der entsprechenden staggered Variablen vorgegeben ist. Die Eingabe einer Matrix erfolgt dabei wie üblich zeilenweise. Bitte beachten Sie, daß bei der Eingabe eines einzelnen staggered Elements dessen Komponenten und das anschließende Intervall genau so übernommen werden, wie der Anwender diese Werte eingegeben hat. Vergleichen Sie dazu auch die entsprechenden Anmerkungen auf Seite 159.

Die Ausgabe eines staggered Vektors oder einer staggered Matrix erfolgt wie bei der Eingabe komponentenweise in dem auf Seite 159 beschriebenen Format, wobei sich die staggered Komponenten eines einzelnen Elements nicht überlappen und einerlei Vorzeichen besitzen. Die ausgegebenen Elemente schließen die berechneten Matrix- oder Vektor-Komponenten garantiert ein, wobei die berechneten Elemente durch die Ausgabe natürlich nicht verändert werden.

4.6 Zufallsmatrizen

Die Berechnung reeller und komplexer Punkt- und Intervall–Matrizen und Vektoren wurde bereits im Kapitel 1 ab Seite 40 beschrieben.

4.7 Elementare Matrix/Vektor–Umformungen

4.7.1 Tauschoperationen

Zur Verfügung stehen Routinen zum Vertauschen zweier Komponenten bzw. zweier Zeilen oder Spalten von Vektoren bzw. Matrizen. Darüber hinaus existieren Routinen zum Permutieren von Vektorkomponenten bzw. Zeilen oder Spalten von Matrizen. Es werden folgende Datentypen berücksichtigt:

`intvector, (L)rvector, (L)rmatrix, (L)ivector, (L)imatrix,`
`(L)cvector, (L)cmatrix, (L)civector, (L)cimatrix,`
`(L)Staggrvector, (L)Staggrmatrix, (L)Staggcvector,`
`(L)Staggcmatrix,`

Die folgenden Prozeduren ermöglichen das Vertauschen zweier Vektorkomponenten oder das Vertauschen zweier Matrix-Zeilen bzw. Matrix-Spalten:

procedure `swap` (**var** V : ◇ ; i,j : `integer`);
tauscht im Vektor V die i-te mit der j-ten Komponente

procedure `swap_r` (**var** M : ■ ; i,j : `integer`);
vertauscht in der Matrix M die i–te mit der j–ten Zeile

procedure `swap_c` (**var** M : ■ ; i,j : `integer`);
vertauscht in der Matrix M die i-te mit der j-ten Spalte

Beachten Sie bitte, daß z.B. die dritte ($i = 3$) Komponente des Vektors $\boldsymbol{V}$ nicht mit $\boldsymbol{V}[3]$ übereinstimmen muß, da der Indexbereich von $\boldsymbol{V}$ beliebig gewählt werden kann. Entsprechendes gilt für die Zeilen und Spalten der Matrizen M.

Die Zuordnung der Symbole ◇,■ zu den zulässigen Datentypen und die Angabe der exportierenden Module findet man in der nachfolgenden Tabelle.

Zuordnung der Symbole ◇,■ zu den erlaubten Datentypen bzgl. der Prozeduren dieses Abschnitts			
◇	`(L)rvector` `intvector`	`(L)ivector`	`(L)Staggrvector`
■	`(L)rmatrix`	`(L)imatrix`	`(L)Staggrmatrix`
exportrnd. Module	(l)swap		(l)stgswap
◇	`(L)cvector`	`(L)civector`	`(L)Staggcvector`
■	`(L)cmatrix`	`(L)cimatrix`	`(L)Staggcmatrix`
exportrnd. Module	(l)cswap		(l)scswap

Die folgenden Prozeduren ermöglichen mit Hilfe des Vektors `p : intvector` die Permutation der Komponenten eines Vektors oder die Permutation der Zeilen oder Spalten einer Matrix beliebiger Form:

> **procedure** `Pv` (**var** p : `intvector`; **var** V,PV : ◇);
> permutiert den Vektor V mit Hilfe von p zum Vektor PV

> **procedure** `PA` (**var** p : `intvector`; **var** M,PM : ■);
> permutiert die Zeilen von M mit Hilfe von p zur Matrix PM

> **procedure** `AP` (**var** M,MP : ■ ; **var** p : `intvector`);
> permutiert die Spalten von M mit Hilfe von p zur Matrix MP

Die Zuordnung der Symbole ◇,■ zu den jeweils erlaubten Datentypen und die exportierenden Module der jeweiligen Prozeduren können ebenfalls in obiger Tabelle abgelesen werden.

Die genaue Bedeutung des Permutations–Vektors `p` wird jetzt bzgl. der Prozedur `PA ( var p : intvector; var M,PM : rmatrix )` beschrieben, bei der die Ausgangsmatrix PM durch Permutation der Zeilen der Eingangsmatrix M bestimmt wird.

Der *integer*-Wert $p[k] = i$ definiert in der Ausgangsmatrix PM deren k-te Zeile durch die i-te Zeile der Eingangsmatrix M.

Mit: $\boldsymbol{p}^T := (2, 3, 1, 4)$ und $\boldsymbol{M} := \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}$ liefert `PA( p,M,PM )`

die zeilenpermutierte Ausgangsmatrix: $\texttt{PM} := \begin{pmatrix} 2 & 2 & 2 \\ 3 & 3 & 3 \\ 1 & 1 & 1 \\ 4 & 4 & 4 \end{pmatrix}$

Mit Hilfe des Permutations–Vektors `p` läßt sich eine Zeilen–Permutations–Matrix $\boldsymbol{P}_z$ wie folgt definieren:

Der *integer*-Wert $p[k] = i$ definiert in der Zeilen-Permutations–Matrix $\boldsymbol{P}_z$ in der k-ten Zeile den i-ten Einheitsvektor.

Demnach definiert obiger Permutations–Vektor $\boldsymbol{p}^T := (2, 3, 1, 4)$ die Zeilen–Permutations–Matrix

$$\boldsymbol{P}_z = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{und es gilt:} \quad \boxed{\texttt{PM} = \boldsymbol{P}_z \cdot \boldsymbol{M}}$$

Ganz analog wird jetzt die Bedeutung des Permutations–Vektors `p` bzgl. der Prozedur `AP ( var M,MP : rmatrix; var p : intvector )` beschrieben, bei der die Ausgangsmatrix MP durch Permutation der Spalten der Eingangsmatrix M bestimmt wird.

Der *integer*-Wert $p[k] = i$ definiert in der Ausgangsmatrix MP deren k-te Spalte durch die i-te Spalte der Eingangsmatrix M.

Mit: $\boldsymbol{p}^T := (2,3,1,4)$, $\quad \boldsymbol{M} := \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$ liefert `AP( M,MP,p )`

die spaltenpermutierte Ausgangsmatrix: $\mathtt{MP} := \begin{pmatrix} 2 & 3 & 1 & 4 \\ 2 & 3 & 1 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}$

Mit Hilfe des Permutations–Vektors `p` läßt sich ganz analog auch eine Spalten–Permutations–Matrix $\boldsymbol{P}_s$ wie folgt definieren:

> Der *integer*-Wert $p[k] = i$ definiert in der Spalten-Permutations–Matrix $\boldsymbol{P}_s$ in der k-ten Spalte den i-ten Einheitsvektor.

Demnach definiert obiger Permutations–Vektor $\boldsymbol{p}^T := (2,3,1,4)$ die Spalten–Permutations–Matrix

$$\boldsymbol{P}_s = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{und es gilt:} \quad \boxed{\mathtt{MP} = \boldsymbol{M} \cdot \boldsymbol{P}_s}$$

Bei gleichem Permutations–Vektor $\boldsymbol{p}$ gilt die Beziehung: $\boxed{\boldsymbol{P}_z^T = \boldsymbol{P}_s}$

Hinweise:

- Beachten Sie bitte, daß die Komponenten des Permutations–Vektors $\boldsymbol{p}$ der Dimension n, nur durch eine Permutation der Zahlen $1, 2, \ldots, n$ definiert sind.
- Aus Speicherplatzgründen wird man in Programmen natürlich nicht die Permutationsmatrizen $\boldsymbol{P}$ selbst, sondern nur die sie definierenden Permutations–Vektoren $\boldsymbol{p}$ benutzen.

4.7.2 Umwandlungen in Dreiecksmatrizen $\boldsymbol{L}, \boldsymbol{U}$

Die beiden folgenden Prozeduren erzeugen aus einer vorgegebenen quadratischen Matrix `Q` aus den Elementen unterhalb ihrer Hauptdiagonalen eine normierte `L`–Matrix bzw. aus den Elementen auf und oberhalb der Hauptdiagonalen eine `U`–Matrix. Es gilt also i.a. $\mathtt{Q} \neq \mathtt{L} \cdot \mathtt{U}$ und $\mathtt{Q} \neq \mathtt{L} + \mathtt{U}$.

> **procedure** `MLU_to_L` (**var** Q,L : ■);
>
> `Q` (quadratisch) → `L` (Diagonalelemente auf 1 normiert)

> **procedure** `MLU_to_U` (**var** Q,U : ■);
>
> `Q` (quadratisch) → `U` (mit den Elementen der Hauptdiagonalen)

Die Zuordnung des Symbols ■ zu den erlaubten Datentypen und das jeweils exportierende Modul findet man wieder in der Tabelle von Seite 223.

Mit $\quad \mathtt{Q} = \begin{pmatrix} 3 & 7 & 8 & 1 \\ 2 & 5 & 7 & 2 \\ 1 & 4 & 6 & 3 \\ 4 & 3 & 5 & 4 \end{pmatrix}$ erhält man z.B. die folgenden Ergebnisse:

`MLU_to_L(Q,L)` liefert: $\mathtt{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 4 & 3 & 5 & 1 \end{pmatrix}$

`MLU_to_U(Q,U)` liefert: $\mathtt{U} = \begin{pmatrix} 3 & 7 & 8 & 1 \\ 0 & 5 & 7 & 2 \\ 0 & 0 & 6 & 3 \\ 0 & 0 & 0 & 4 \end{pmatrix}$

Die beiden Prozeduren liefern also nicht die $\boldsymbol{LU}$–Zerlegung von $\boldsymbol{Q}$, sondern produzieren die Dreiecksmatrizen $\boldsymbol{L}$ oder $\boldsymbol{U}$ aus $\boldsymbol{Q}$, wenn $\boldsymbol{Q}$ das Ergebnis einer $\boldsymbol{LU}$–Zerlegung nach dem Crout'schen Algorithmus ist; vergleichen Sie dazu bitte den Abschnitt 4.9 ab Seite 249.

4.8 Elementare Matrix/Vektor–Funktionen

In den folgenden Unterabschnitten findet man Routinen zur Berechnung elementarer Matrix/Vektor–Funktionen, die in den System–Modulen der **BCD**–Version von **PASCAL-XSC** nicht vorhanden sind.

4.8.1 Betrags–Vektoren/Matrizen

Die folgenden Funktionen liefern zu jedem Vektor oder Matrix den Vektor oder die Matrix seiner Betrags–Elemente, dabei ist der Betrag eines komplexen Intervalls `ci` ein reelles Gleitpunkt–Intervall, das alle Beträge der komplexen Zahlen aus `ci` optimal einschließt, und der Betrag eines reellen Intervalls `i` ist wieder ein reelles Intervall, welches alle Beträge der reellen Zahlen aus `i` optimal einschließt.

4.8.1.1 Reelle Matrizen/Vektoren

function `abs` (**var** v : ▼) : ▼ `[lb(v)..ub(v)];`

function `abs` (**var** M : ■) :
■ `[lb(M)..ub(M),lb(M,2)..ub(M,2)];`

Die Zuordnung der Symbole ▼,■ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von ▼,■ zu den Datentypen		
`v` : ▼	`(L)rvector`	`(L)ivector`
`M` : ■	`(L)rmatrix`	`(L)imatrix`
exportrnd. Module	(l)mvf0	(l)mvif0

Die Funktion `abs` mit `v : Livector` wird vom Modul lmvif0 exportiert, und die Funktion `abs` mit `M : rmatrix` nach obiger Tabelle vom Modul mvf0.

4.8.1.2 Komplexe Matrizen/Vektoren

function `abs` (**var** v : ▼) : ▽ `[lb(v)..ub(v)];`

function `abs` (**var** M : ■) :
□ `[lb(M)..ub(M),lb(M,2)..ub(M,2)];`

Die Zuordnung der Symbole ▼, ▽, ■, □ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von ▼, ▽, ■, □ zu den Datentypen		
`v` : ▼	`(L)cvector`	`(L)civector`
▽	`(L)ivector`	
exportierende Module	(l)mvcf0	(l)mcif0
`M` : ■	`(L)cmatrix`	`(L)cimatrix`
□	`(L)imatrix`	
exportierende Module	(l)mvcf0	(l)mcif0

Die Symbole ■, □ einer Funktion enthalten entweder nur mit ( ) versehene Namen oder nur Namen ohne ( ) Klammern. Danach ist z.B. der Funktionsaufruf `y := abs(M);` mit `y : imatrix` und `M : Lcimatrix` unzulässig. Die analoge Regelung gilt auch für die Namen der exportierenden Module aus obiger Tabelle. Die Funktion `abs(M)` mit `M : Lcimatrix` wird daher vom Modul `lmcif0` exportiert.

4.8.2 Matrix/Vektor–Normen

Matrix- oder Vektornormen sind Funktionen, die einer Matrix oder einem Vektor eine reelle Zahl zuordnen, wobei diese Funktionen mehrere Bedingungen zu erfüllen haben, die z.B. in [19, Seite 29] beschrieben werden.

4.8.2.1 Die wichtigsten Normen

Für reelle Vektoren $\boldsymbol{x}$ lauten die wichtigsten Normen:

$$\begin{array}{lll}
\|\boldsymbol{x}\|_\infty := \max\limits_k |\boldsymbol{x}_k|, & \text{Maximumnorm} & \texttt{(MaxNorm)} \\
\|\boldsymbol{x}\|_1 := \sum\limits_{k=1}^{n} |\boldsymbol{x}_k|, & L_1\text{–Norm} & \texttt{(L1Norm)} \\
\|\boldsymbol{x}\|_2 := \left[\sum\limits_{k=1}^{n} |\boldsymbol{x}_k|^2\right]^{1/2}, & \text{euklidische Norm.} & \texttt{(EkNorm)}
\end{array}$$

Für reelle, quadratische Matrizen $\boldsymbol{A}$ lauten die wichtigsten Normen:

$$\begin{array}{lll}
\|\boldsymbol{A}\|_G := n \cdot \max\limits_{i,k} |\boldsymbol{a}_{ik}|, & \text{Gesamtnorm} & \texttt{(NormG)} \\
\|\boldsymbol{A}\|_Z := \max\limits_i \sum\limits_{k=1}^{n} |a_{ik}|, & \text{Zeilensummen–Norm} & \texttt{(NormR)} \\
\|\boldsymbol{A}\|_S := \max\limits_k \sum\limits_{i=1}^{n} |a_{ik}|, & \text{Spaltensummen–Norm} & \texttt{(NormC)} \\
\|\boldsymbol{A}\|_F := \left[\sum\limits_{i,k=1}^{n} |\boldsymbol{a}_{ik}|^2\right]^{1/2}, & \text{Frobenius–Norm.} & \texttt{(NormF)}
\end{array}$$

In den jeweils letzten Spalten rechts stehen die geklammerten Namen der **PASCAL-XSC** Funktionen, mit denen garantierte Einschließungen der entsprechenden Normen berechnet werden können.

Ist $\boldsymbol{x}$ ein komplexer Vektor, so definiert man $|\boldsymbol{x}_k|$ wie üblich als Betrag einer komplexen Zahl: $|\boldsymbol{x}_k| := \sqrt{\Re(\boldsymbol{x}_k) \cdot \Re(\boldsymbol{x}_k) + \Im(\boldsymbol{x}_k) \cdot \Im(\boldsymbol{x}_k)}$. Damit kann man die obigen Normen auch auf komplexe Vektoren und Matrizen übertragen.

Normen für reelle oder komplexe Intervall–Vektoren oder Intervall–Matrizen werden nicht behandelt.

4.8.2.2 Berechnung der Vektor–Normen

function `MaxNorm` (**var** x : $\blacktriangledown$) : $\triangledown$;	Maximum–Norm
function `L1Norm` (**var** x : $\blacktriangledown$) : $\triangledown$;	L1–Norm
function `EkNorm` (**var** x : $\blacktriangledown$) : $\triangledown$;	euklidische–Norm

Die Zuordnung der Symbole $\blacktriangledown, \triangledown$ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von $\blacktriangledown, \triangledown$ zu den Vektordatentypen		
`x` : $\blacktriangledown$	`(L)rvector`	`(L)cvector`
$\triangledown$	`(L)interval`	
exportierende Module	(l)mvf0	(l)mvcf0

Die beiden Symbole $\blacktriangledown, \triangledown$ einer Funktion enthalten entweder nur mit ( ) versehene Namen oder nur Namen ohne ( ) Klammern. Danach ist z.B. der Funktionsaufruf

```
y := MaxNorm(x);
```

mit `y : interval` und `x : Lcvector` unzulässig. Die analoge Regelung gilt auch für die Namen der exportierenden Module aus obiger Tabelle. Die obige Funktion `MaxNorm(x)` mit `x : Lcvector` und dem Ergebnistyp `Linterval` wird daher vom Modul `lmvcf0` exportiert und nicht etwa vom Modul `mvcf0`.

Bitte beachten Sie, daß die obigen drei Funktionen zu jedem Vektor eine garantierte **Einschließung** der entsprechenden Vektor–Norm berechnen.

4.8.2.3 Berechnung der Matrix–Normen

function `NormG` (**var** A : ■) : □;	Gesamtnorm
function `NormR` (**var** A : ■) : □;	Zeilensummen–Norm
function `NormC` (**var** A : ■) : □;	Spaltensummen–Norm
function `NormF` (**var** A : ■) : □;	Frobenius–Norm

Die Zuordnung der Symbole ■, □ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von ■, □ zu den Matrixdatentypen		
`A` : ■	`(L)rmatrix`	`(L)cmatrix`
□	`(L)interval`	
exportierende Module	(l)mvf0	(l)mvcf0

Die beiden Symbole ■, □ einer Funktion enthalten entweder nur mit ( ) versehene Namen oder nur Namen ohne ( ) Klammern. Danach ist z.B. der Funktionsaufruf

```
y := NormF(x);
```

mit `y : interval` und `x : Lcmatrix` unzulässig. Die analoge Regelung gilt auch für die Namen der exportierenden Module aus obiger Tabelle. Die obige Funktion `NormF(x)` mit `x : Lcmatrix` wird daher vom Modul `lmvcf0` exportiert und nicht etwa vom Modul `mvcf0`.

Bitte beachten Sie, daß die obigen vier Funktionen zu jeder Matrix eine garantierte **Einschließung** der entsprechenden Matrix–Norm berechnen.

4.8.2.4 Programmbeispiele

4.8.2.4.1 Verträglichkeit von Matrix/Vektor–Normen

Nach [19, Seite 31] ist die Gesamtnorm einer beliebigen Matrix $\boldsymbol{A}$ mit der Maximum–Norm eines beliebigen Vektors $\boldsymbol{x}$ verträglich, d.h. es gilt:

$$\|\boldsymbol{A} \cdot \boldsymbol{x}\|_\infty \leq \|\boldsymbol{A}\|_G \cdot \|\boldsymbol{x}\|_\infty \qquad \text{für alle } \boldsymbol{A} \text{ und } \boldsymbol{x}.$$

Berechnet man im nachfolgenden Programm `tstnorm1` mit Hilfe der Funktionen `MaxNorm` und `NormG` Einschließungen von $\|\boldsymbol{A} \cdot \boldsymbol{x}\|_\infty$ und $\|\boldsymbol{A}\|_G \cdot \|\boldsymbol{x}\|_\infty$ und ist die Oberschranke der Einschließung der linken Seite kleiner oder gleich der Unterschranke der Einschließung der rechten Seite, so erfolgt die Bildschirmausgabe:

```
MaxNorm(A*x) <= NormG(A) * MaxNorm(x)
```

Der Programmabbruch erfolgt mit der Tastenkombination [Strg]+[C].

```
program tstnorm1(input,output);
use mvcf0   , { exportiert MaxNorm, NormG         }
    mvcran  , { exportiert Rand_V, Rand_M         }
    c_ari   , { komplexe Arithmetik               }
    i_ari   , { Intervall-Arithmetik              }
    mvc_ari ; { Komplexe Matrix/Vektor Arithmetik }
var x         : cvector[1..100];
   A          : cmatrix[1..100,1..100];
   c1,c2      : complex;
   intl,intr  : interval;
procedure exit(retcode : integer);        external  a_exit;
begin { tstnorm1 }
   c1 := compl(-60,-1e2);  c2 := compl(+60,+1e2);
   repeat
      Rand_V(x,c1,c2); { erzeugt komplexen Zufalls-Vektor }
      Rand_M(A,c2,c2); { erzeugt komplexe  Zufalls-Matrix }
      intl := MaxNorm(A*x);  intr := NormG(A) * MaxNorm(x);
      if intl.sup <= intr.inf
      then writeln('MaxNorm(A*x) <= NormG(A) * MaxNorm(x)')
      else
      begin
         writeln('! Keine Aussage moeglich !');  exit(-1);
      end;
   until false;
end.
```

4.8.2.4.2 Winkel zwischen Vektoren

Für den Winkel φ zwischen zwei reellen, dreidimensionalen, punktförmigen Vektoren $\boldsymbol{u}, \boldsymbol{v}$ gilt die Beziehung:

$$\cos(\varphi) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\|} , \tag{4.2}$$

wobei $\|\boldsymbol{u}\| = \sqrt{u_1^2 + u_2^2 + u_3^2}$ die euklidische Norm des Vektors $\boldsymbol{u}$ bedeutet.

Das nachfolgende Programm `tstnorm2` berechnet für die eingegebenen Vektoren $\boldsymbol{u}, \boldsymbol{v}$ mit Hilfe der Funktionen `winkel` und `angle` Einschließungen des Winkels φ im Grad–Maß. Dabei wertet die Funktion `winkel` die rechte Seite von (4.2) direkt aus, wobei im Nenner die Wurzel zweimal zu berechnen ist. Dies ist nicht nur zeitaufwendig sondern führt auch zu unnötigen Intervallüberschätzungen, da im Nenner dann zwei Intervalle zu multiplizieren sind.

Dies wird in der ersten Funktion `angle` vermieden, indem die beiden Quadratsummen in je einem Akkumulator berechnet und beide Akkumulatoren dann rundungsfehlerfrei multipliziert werden. Danach wird das exakte Produkt durch `##(dot)` in ein Intervall ausgelesen und erst dann erfolgt die Quadratwurzelberechnung, die jetzt nur einmal nötig ist.

In der zweiten Funktion `angle` wird der beschriebene Algorithmus auf Intervallvektoren $\boldsymbol{u}, \boldsymbol{v}$ übertragen. Dadurch lassen sich dann auch die Winkel zwischen dreidimensionalen Intervallvektoren einschließen.

```
program tstnorm2(input,output);
{ Einschliessungen der Winkel (im Gradmass) zwischen }
{ Punkt-Vektoren im dreidimensionalen Raum:          }
use mvf0  , { exportiert EkNorm(Vektor)  }
    mv_ari, { Matrix/Vektor-Arithmetik   }
    i_ari , { Intervall-Arithmetik       }
    li_ari; { Longreal-Intervall-Arithm. }
const c = 5.729577951308E+001; { Unterschr. von 180/pi }
var u,v            : rvector[1..3];

function winkel(u,v : rvector) : interval;
var x : interval;
begin
   x := ##(u*v) / (EkNorm(u)*EkNorm(v));
   if x.inf < -1 then x.inf := -1; { Verhindert Fehlermeldung }
   if x.sup > +1 then x.sup := +1; { bei der arccos-Funktion  }
   winkel := arccos(x)*45/arctan(intval(1));
```

```
end; { angle }

function angle(u,v : rvector) : interval;
{ Genauere Einschliessung des Winkels zwischen u,v }
var x,y : interval;
   dot  : dotprecision;
   i    : integer;
begin
   x := ##(u*v); { x: Einschliessung von u*v }
   if x = 0 then angle := intval(90) else
   begin
      dot := #(for i:= lb(u) to ub(u) SUM(u[i]*u[i]));
      dot := dot * #(for i:= lb(v) to ub(v) SUM(v[i]*v[i]));
      x := x/sqrt(##(dot));
      if x.inf < -1 then x.inf := -1;
      if x.sup > +1 then x.sup := +1;
      x := arccos(x);
      y.inf := c;  y.sup := succ(c);
      { y: Einschliessung von 180/pi }
      angle := x*y; { angle_ im Gradmass }
   end;
end; { angle }

function angle(u,v : ivector) : interval;
{ Einschliessung des Winkels zwischen }
{ reellen Intervallvektoren u,v       }
var x,y : interval;
   dot  : idotprecision;
   i    : integer;
begin
   x := ##(u*v); { x: Einschliessung von u*v }
   if x = 0 then angle := intval(90) else
   begin
      dot := #(for i:= lb(u) to ub(u) SUM(u[i]*u[i]));
      dot := dot * #(for i:= lb(v) to ub(v) SUM(v[i]*v[i]));
      x := x/sqrt(##(dot));
      if x.inf < -1 then x.inf := -1;
      if x.sup > +1 then x.sup := +1;
      x := arccos(x);
      y.inf := c;  y.sup := succ(c);
```

```
      { y: Einschliessung von 180/pi }
      angle := x*y; { angle_ im Gradmass }
   end;
end; { angle }

begin
   repeat
   writeln('Dreidimensionaler Vektor u = ?'); read(u);
   writeln('Dreidimensionaler Vektor v = ?'); read(v);
   writeln('Der Winkel(Grad) zwischen u,v liegt in:');
   writeln(winkel(u,v));
   writeln;
   writeln('Genauere Einschliessung des Winkels:');
   writeln('Der Winkel(Grad) zwischen u,v liegt in:');
   writeln(angle(u,v));  writeln;
   until false;
end.
```

Mit der Eingabe der Vektoren $\boldsymbol{u} = \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$, $\boldsymbol{v} = \begin{pmatrix} 3 \\ 2 \\ -4 \end{pmatrix}$ liefert das Programm `tstnorm2` die Bildschirmausgabe:

```
Der Winkel(Grad) zwischen u,v liegt in:
[  8.999999999994E+001,  9.000000000002E+001 ]

Genauere Einschliessung des Winkels:
Der Winkel(Grad) zwischen u,v liegt in:
[  9.000000000000E+001,  9.000000000000E+001 ]
```

4.8.3 Spur einer Matrix

Die Spur einer quadratischen[2] Matrix $\boldsymbol{A}^{(n\times n)}$ ist definiert als Summe ihrer Diagonalelemente. Mit Hilfe der Funktionen `trace` bzw. `trace_i` wird eine maximalgenaue Näherung bzw. Einschließung der Spur einer reellen oder komplexen Matrix oder eines Matrix–Produktes berechnet.

4.8.3.1 Maximalgenaue Näherung der Spur

Die beiden folgenden Funktionen berechnen eine maximalgenaue Näherung der Spur einer Matrix oder eines Matrix–Produktes.

function `trace` (**var** A : ■) : ⊙ ;

function `trace` (**var** A,B : ■) : ⊙ ;

Die Zuordnung der Symbole ■, ⊙ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von ■, ⊙ zu den Datentypen		
`A` : ■	`(L)rmatrix`	`(L)cmatrix`
⊙	`(Longreal)` `real`	`(L)complex`
exportierende Module	(l)mvf1	(l)mvcf1

Die Symbole ■, ⊙ in einer Funktion enthalten entweder nur mit ( ) versehene Namen oder nur Namen ohne ( ) Klammern. Danach ist z.B. der Funktionsaufruf `y := trace(A);` mit `y : real` und `A : Lrmatrix` unzulässig. Die analoge Regelung gilt auch für die Namen der exportierenden Module aus obiger Tabelle. Die Funktion `trace(A)` mit `A : Lrmatrix` wird daher vom Modul `lmvf1` exportiert.

[2] Für nicht-quadratische Matrizen ist die Spur nicht definiert.

4.8.3.2 Maximalgenaue Einschließung der Spur

Die folgenden Funktionen berechnen eine maximalgenaue Einschließung der Spur einer Matrix oder eines Matrix–Produktes.

function `trace_i` (**var** A : ■) : ♦ ;

function `trace_i` (**var** A,B : ■) : ♦ ;

Die Zuordnung der Symbole ■, ♦ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von ■, ♦ zu den Datentypen		
`A` : ■	`(L)rmatrix`	`cmatrix`
♦	`(L)interval`	`cinterval`
exportierende Module	(l)mvf1	(l)mvcf1

Die Symbole ■, ♦ in einer der beiden Funktionen enthalten entweder nur mit ( ) versehene Namen oder nur Namen ohne ( ) Klammern. Danach ist z.B. `y := trace(A);` mit `y : interval` und `A : Lrmatrix` unzulässig. Die analoge Regelung gilt auch für die Namen der exportierenden Module aus obiger Tabelle. Die Funktion `trace_i(A)` mit `A : Lrmatrix` wird daher vom Modul `lmvf1` exportiert und nicht etwa vom Modul `mvf1`.

4.8.3.3 Programmbeispiel

Nach [22, Seite 37,(34)] besteht zwischen der Frobenius–Norm[3] $\|\boldsymbol{A}\|_F$ und der Spur einer quadratischen Matrix $\boldsymbol{A}$ der Zusammenhang:

$$\|\boldsymbol{A}\|_F = \sqrt{\mathrm{Spur}(\boldsymbol{A}^* \cdot \boldsymbol{A})}, \tag{4.3}$$

wobei $\boldsymbol{A}^*$ die hermitesche Matrix bedeutet. Das nachfolgende Programm `tsttrace` berechnet mit Hilfe der Funktion `NormF` eine Einschließung der Frobenius–Norm $\|\boldsymbol{A}\|_F$ und mit `S` ein komplexes Intervall, das die rechte Seite der obigen Gleichung garantiert einschließt.

[3]vergl. Seite 229

```
program tsttrace(input,output);
use mvcran,  { exportiert Rand_M             }
    c_ari,   { komplexe Arithmetik           }
    ci_ari,  { komplexe Intervall-Arithmetik }
    mvcf0,   { exportiert NormF              }
    mvcf1,   { exportiert trace_i            }
    i_ari,   { Intervall-Arithmetik          }
    mvc_ari; { exportiert herm(A)            }
var A     : cmatrix[1..100,1..100];  { kompl. Zufallsmatrix }
   c1,c2 : complex;        { Bestimmt kompl. Bereich von A }
   OS    : interval;
   S     : cinterval;
begin
   c1 := compl(-30,-50);  c2 := compl(+30,+50);
   Rand_M(A,c1,c2);
   OS := NormF(A); { OS: Einschliessung der Frobenius-Norm }
   writeln('Einschliessung der Frobenius-Norm = ');
   writeln('  ',OS);
   S  := sqrt( trace_i(herm(A),A) );
   writeln('Wurzel( Spur(herm(A)*A) ) = ');
   writeln(S);
end.
```

Das obige Programm liefert die folgende Bildschirmausgabe:

```
Einschliessung der Frobenius-Norm =
  [  3.354877249723E+003,  3.354877249725E+003 ]
Wurzel( Spur(herm(A)*A) ) =
( [  3.354877249723E+003,  3.354877249725E+003 ],
  [  0.000000000000E+000,  0.000000000000E+000 ] )
```

Man erkennt, daß beide Seiten der Gleichung (4.3) durch das gleiche reelle Intervall eingeschlossen werden. Man erkennt außerdem, daß der Imaginärteil von $\sqrt{\mathrm{Spur}(\boldsymbol{A}^* \cdot \boldsymbol{A})}$ tatsächlich verschwindet und auf der Maschine optimal eingeschlossen wird.

4.8.4 Determinanten

Determinanten von quadratischen Matrizen $\boldsymbol{A}$ der Dimension n bezeichnen wir mit $\det(\boldsymbol{A})$ oder $|\boldsymbol{A}|$. Auf die Definition der Determinante wird hier nicht näher eingegangen, sie kann z.B. in [2, Seite 89,...] nachgelesen werden.

4.8.4.1 Berechnungsmethoden

Die Berechnung der Determinante einer Matrix $\boldsymbol{A} = (a_{ij})$ erfolgt für die Dimensionen $n = 2, 3$ mit Hilfe ihrer Definitionsgleichungen:

$$\begin{array}{rcll} \det(\boldsymbol{A}) & := & a_{11}a_{22} - a_{21}a_{12}, & \text{für } n = 2; \\ \det(\boldsymbol{A}) & := & +a_{11}(a_{22}a_{33} - a_{23}a_{32}) + & \\ & & +a_{12}(a_{23}a_{31} - a_{21}a_{33}) + & \\ & & +a_{13}(a_{21}a_{32} - a_{22}a_{31}), & \text{für } n = 3; \end{array}$$

Die beiden obigen Gleichungen haben den Vorteil, daß sie wegen fehlender Divisionen mit Hilfe des Akkumulators exakt[4] ausgewertet werden können, womit dann eine maximalgenaue Einschließung der Determinanten möglich ist. Bei einer singulären Matrix $\boldsymbol{A}$ wird daher $\det(\boldsymbol{A}) = 0$ durch ein **Punkt**-Intervall optimal eingeschlossen, so daß aus dieser Punkteinschließung auf die Sigularität von $\boldsymbol{A}$ geschlossen werden kann.

Für größere Dimensionen n kann $\det(\boldsymbol{A})$ mit ihrer Definitionsgleichung jedoch nicht mehr ausgewertet werden, da eine 10 x 10 Determinante schon die Berechnung von $10! = 3\,628\,800$ Teilprodukten erforderlich macht, was zu nicht vertretbaren Laufzeiten führt. In diesen Fällen ($n \geq 4$) benutzt man bei regulären Matrizen $\boldsymbol{A}$ deren existierende LU–Zerlegung[5]:

$$\boldsymbol{P} \cdot \boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U}$$

$\boldsymbol{P}$ ist eine Permutationsmatrix, die aus der Einheitsmatrix $\boldsymbol{I}$ durch geeignete Zeilenvertauschungen hervorgeht, d.h. es gilt: $\det(\boldsymbol{P}) = \pm 1$. Die Matrix $\boldsymbol{L}$ ist eine untere Dreiecksmatrix, deren Diagonalelemente auf Eins normiert werden, d.h. es gilt: $\det(\boldsymbol{L}) = 1$. Die Matrix $\boldsymbol{U}$ ist eine obere Dreiecksmatrix, deren Determinante durch das Produkt ihrer Diagonalelemente gegeben ist.

Für eine reguläre Matrix $\boldsymbol{A}$ gilt damit nach dem Determinanten–Multiplikationssatz:

$$\det(\boldsymbol{A}) = \pm \prod_{i=1}^{n} u_{ii}$$

[4]Bei Über- oder Unterlauf entstehen automatische Fehlermeldungen!

[5]Vergleichen Sie bitte dazu den Abschnitt 4.9 ab Seite 249.

Ist die Anzahl der Zeilenvertauschungen in $\boldsymbol{P}$ gerade, so gilt das + Zeichen, sonst gilt das Minuszeichen.

4.8.4.2 Routinen zur Determinanten–Einschließung

Mit Hilfe der folgenden Prozedur wird im Fall `Err=0` mit dem Intervall `D` eine Einschließung von det($\boldsymbol{A}$) berechnet:

> **procedure** `Det` (**var** A : ■ ; **var** D : ⧫ ; **var** Err : `integer`);

Die Zuordnung der Symbole ■, ⧫ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von ■, ⧫ zu den erlaubten Datentypen		
`A` : ■	`(L)rmatrix`	
`D` : ⧫	`(L)interval`	`(L)Stagg`
exportierende Module	(l)mvf1	(l)sgmf1
`A` : ■	`(L)imatrix`	`(L)Staggrmatrix`
`D` : ⧫	`(L)interval`	`(L)Stagg`
exportierende Module	(l)mvif1	(l)sgmf1
`A` : ■	`(L)cmatrix`	
`D` : ⧫	`(L)cinterval`	
exportierende Module	(l)mvcf1	
`A` : ■	`(L)cmatrix`	`(L)Staggcmatrix`
`D` : ⧫	`(L)Staggcomplex`	
exportierende Module	(l)smcf1	

Die Symbole ■, ⧫ in der Prozedur `Det` enthalten entweder nur mit ( ) versehene Namen oder nur Namen ohne ( ) Klammern. Danach ist z.B. `Det(A,D,Err)` mit `A: Lcmatrix` und `D: cinterval` unzulässig. Die analoge Regelung gilt auch für die Namen der exportierenden Module aus obiger Tabelle. Die Prozedur `Det(A,D,Err)` mit `A: Lcmatrix` und `D: Lcinterval` wird daher vom Modul `lmvcf1` exportiert und nicht etwa vom Modul `mvcf1`.

4.8.4.3 Programmbeispiele

4.8.4.3.1 Boothroyd/Dekker–Matrizen

Die Boothroyd/Dekker–Matrizen $\boldsymbol{B}$ sind definiert im Abschnitt 4.1.2 auf Seite 164 und können für `B: Lrmatrix` bis zur Dimension $n = 27$ mit der Prozedur `Dekker(B,v)` realisiert werden. Mit der Dimension $n = 27$ ist die Boothroyd/Dekker–Matrix jedoch schon so schlecht konditioniert, daß die Einschließung seiner Determinante $\det(\boldsymbol{B}) = 1$ nur noch in staggered Arithmetic mit Hilfe des folgenden Programms `tstdet1` zu berechnen ist:

```
program tstdet1(input,output);
use type_def, { exportiert Datentypen und LStagg_prec }
   lmv_exap , { exportiert Dekker(B,v);              }
   lsgmf1   , { exportiert Det(B,D,Err);             }
   lstagg;    { Staggered Arithmetik bzgl.  Longreal }
var p : integer;
procedure main;
var B  : Lrmatrix[1..27,1..27];   { Dekker-Matrix mit n = 27  }
   v   : Lrvector[1..27];         { v[i] = i;   i=1,2,...,27  }
   Err : integer;          { Err=0: Keine Fehler aufgetreten  }
   D   : LStagg[-1..LStagg_prec]; { Einschliessung von Det(B) }
begin
   set_maxprec(7); { Genauigkeit zur Berechnung der }
   Dekker(B,v);    { Boothroyd/Dekker-Matrix   B.   }
   Det(B,D,Err);   { Berechnung von D und Err;      }
   if Err = 0 then
      begin
         writeln(' Det(B) enthalten in ');
         writeln(D);
      end     else writeln('Err = ',Err); { Err: Fehlerkode }
end; { main }
begin
   writeln('LStagg_prec = ',get_LStagg_prec);
   writeln('LStagg_prec = ?');  read(p);
   set_LStagg_prec(p);            { p: Neue LStagg-Praezision }
   main;       { Aufruf der Prozedur main nach Festlegung der }
               { Variablen LStagg_prec im Modul type_def.     }
end.
```

Der Start von `tstdet1` liefert die folgende Bildschirmausgabe:

```
LStagg_prec = 1
LStagg_prec = ?
4          <enter>
Det(B)=1 enthalten in D =
9.9999999999999999999E-001
9.9999999999999999996E-022
4.2823917327473353400E-043
0.0000000000000000000E+000
[ -3.7992258140261284230E-040,  3.7992258140261284230E-040]
```

Damit ist das Intervall $\boldsymbol{D}$ eine garantierte Einschließung von $\det(\boldsymbol{B}) = 1$.

4.8.4.3.2 Singuläre Matrizen

Eine Matrix $\boldsymbol{A}$ heißt singulär, wenn gilt:

$$\det(\boldsymbol{A}) = 0$$

In der Praxis entsteht oft die Fragestellung, ob eine vorgelegte quadratische Matrix $\boldsymbol{A}$ singulär ist. Für die Dimensionen $n = 2, 3$ kann diese Frage mit Hilfe der Prozedur `Det(A,D,Err)` bzgl. jeder Matrix $\boldsymbol{A}$ beantwortet werden, da für alle singulären Matrizen $\boldsymbol{A}$ mit `D` eine punktförmige Einschließung der Null berechnet wird, und eine reguläre Matrix ein Intervall `D` liefert, welches die Zahl Null garantiert nicht enthält.

Für die Dimensionen $n \geq 4$ ist die Sachlage jedoch komplizierter, da zur Einschließung der Determinante eine Intervallrechnung benutzt wird, die unvermeidbare Intervallüberschätzungen zur Folge hat. Bei regulären Matrizen kann dies dazu führen, daß im Algorithmus durch ein Intervall zu dividieren ist, das die Zahl Null nur wegen der erwähnten Aufblähungseffekte enthält. Die damit verbundene Fehlermeldung (Division durch Null) kann jedoch bei regulären Matrizen vermieden werden, wenn die Determinantenberechnung in staggered Arithmetik, d.h. in höherer Präzision wie im Programm `tstdet1` durchgeführt wird. Liefert also `Det(A,D,Err)` mit `Err=0` ein Intervall `D`, das die Null nicht enthält, so ist die Matrix $\boldsymbol{A}$ garantiert regulär.

Ist $\boldsymbol{A}$ jedoch singulär, so können bei der Berechnung der Einschließung $< u_{jj} >$ eines verschwindenden Diagonalelements u_{jj} der oberen Dreiecksmatrix $\boldsymbol{U}$ zwei Fälle auftreten:

1. $< u_{jj} >$ ist ein Punktintervall, das die Null enthält.
2. $< u_{jj} >$ ist ein Intervall mit positivem Durchmesser, das die Zahl Null enthält.

Im ersten Fall liefert `Det(A,D,Err)` mit dem Fehlerkode `Err=0` das Intervall `D = [0,0]`, woraus sich $\mathrm{Det}(\boldsymbol{A}) = 0$ direkt ergibt.

Im zweiten Fall liefert `Det(A,D,Err)` den Fehlerkode `Err=1`, d.h. eine Einschließung der Determinante kann nicht berechnet werden. Aus `Err=1` darf jedoch nicht geschlossen werden, daß $\boldsymbol{A}$ singulär ist, da der zweite Fall theoretisch auch bei einer regulären Matrix bei zu starken Intervallüberschätzungen eintreten kann. Läßt sich der Fehlerkode `Err=1` auch bei Verwendung der staggered Arithmetik in hoher Präzision nicht beseitigen, so kann man lediglich **vermuten**, daß $\boldsymbol{A}$ singulär ist.

In seltenen Fällen erhält man bei singulären Matrizen durch geeignete Zeilen- oder Spalten-Vertauschungen den gewünschten Fehlerkode `Err=0` und `D = [0,0]`, woraus dann die Singularität von $\boldsymbol{A}$ wieder unmittelbar folgt. Für $n \geq 4$ ist damit der Nachweis der Singularität einer Matrix eine nichttriviale Aufgabe!

Im nachfolgenden Programm `tstdet2` wird der Anwender aufgefordert, eine dreidimensionale, quadratische, reelle Matrix $\boldsymbol{A}$: `rmatrix` einzugeben. Das Programm liefert mit `D : interval` eine maximalgenau Einschließung der Determinante $\mathrm{Det}(\boldsymbol{A})$.

```
program tstdet2(input,output);
use mvf1    , { exportiert Prozedur  Det(B,D,Err);    }
    mv_ari  , { Matrix/Vektor-Arithmetik; read(A);    }
    i_ari   ; { Intervall-Arithmetik;     writeln(D); }

var A  : rmatrix[1..3,1..3];  { Matrix der Dimension 3       }
   Err : integer;        { Err=0:  Keine Fehler aufgetreten  }
   D   : interval;       { liefert Einschliessung von Det(A) }
begin
   writeln('Geben Sie eine dreidim. Matrix zeilenweise ein:');
   read(A);  { Eingabe der Matrix A durch den Anwender }
   Det(A,D,Err); { Berechnung von D mit Fehlerkode Err }
   if Err = 0 then
      begin
         writeln(' Det(A) enthalten in D =');
         writeln(D);               { Bildschirmausgabe von D }
      end     else writeln('Err = ',Err); { Err: Fehlerkode }
end.
```

Die Eingabe der Matrix $\boldsymbol{A} = \begin{pmatrix} 1 & 3 & -2 \\ 2 & 6 & -4 \\ 3 & 9 & 1 \end{pmatrix}$ liefert die Bildschirmausgabe:

```
Det(A) enthalten in D =
[ 0.000000000000E+000,  0.000000000000E+000 ]
```

woraus folgt, daß die Vektoren $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix} \begin{pmatrix} -2 \\ -4 \\ 1 \end{pmatrix}$ bzw. $\begin{pmatrix} 1 \\ 3 \\ -2 \end{pmatrix} \begin{pmatrix} 2 \\ 6 \\ -4 \end{pmatrix} \begin{pmatrix} 3 \\ 9 \\ 1 \end{pmatrix}$ linear abhängig sind.

Im nachfolgenden Programm `tstdet3` liefert die Prozedur `Det(A,D,Err)` zu einer singulären Matrix $\boldsymbol{A}$ der Dimension $n = 5$ zunächst den Fehlerkode `Err=1`, d.h. eine Einschließung von $\text{Det}(\boldsymbol{A}) = 0$ kann nicht berechnet werden. Erst nach dem Vertauschen der Spalten 1 und 2 von $\boldsymbol{A}$ wird mit `D = [0,0]` die Determinante maximalgenau eingeschlossen, woraus sich die Singularität von $\boldsymbol{A}$ dann automatisch ergibt.

```
program tstdet3(input,output);
use mvf1  , { exportiert Prozedur  Det(B,D,Err);    }
    mv_ari, { Matrix/Vektor-Arithmetik; read(A);    }
    i_ari , { Intervall-Arithmetik;     writeln(D); }
    swap  ; { exportiert Prozedur    swap_C(A,1,2); }

var A   : rmatrix[1..5,1..5];  { Matrix der Dimension 4       }
   Err : integer;          { Err=0:  Keine Fehler aufgetreten  }
   D   : interval;         { liefert Einschliessung von Det(A) }
begin
   A[1,1] := 1.16;  A[1,2] := 3;  A[1,3] := -2;  A[1,4] :=  4;
                                                  A[1,5] := 11;
   A[2,1] := 2.32;  A[2,2] := 6;  A[2,3] := -4;  A[2,4] :=  8;
                                                  A[2,5] := 22;
   A[3,1] := 2;     A[3,2] := 3;  A[3,3] :=  2;  A[3,4] :=  4;
                                                  A[3,5] := 11;
   A[4,1] := 2;     A[4,2] := 3;  A[4,3] :=  2;  A[4,4] :=  4;
                                                  A[4,5] := 11;
   A[5,1] := 1;     A[5,2] := 1;  A[5,3] :=  4;  A[5,4] :=  7;
                                                  A[5,5] :=  9;
   Det(A,D,Err); { Berechnung von D mit Fehlerkode Err }
```

```
   if Err = 0 then
      begin
         writeln(' Det(A) enthalten in D =');
         writeln(D);              { Bildschirmausgabe von D }
      end     else writeln(' Err = ',Err); { Err: Fehlerkode }
   writeln(' Nach Vertauschung der Spalten 1,2 in Matrix A:');
   swap_C(A,1,2);     { Vertauscht die 1. mit der 2. Spalte }
   Det(A,D,Err);      { Berechnung von D mit Fehlerkode Err }
   if Err = 0 then
      begin
         writeln(' Det(A) enthalten in D =');
         writeln(D);              { Bildschirmausgabe von D }
      end     else writeln('Err = ',Err); { Err: Fehlerkode }
end.
```

Der Start des Programms `tstdet3` liefert den folgenden Bildschirmausdruck:

```
Err = 1
Nach Vertauschung der Spalten 1,2 in Matrix A:
Det(A) enthalten in D =
[ -0.000000000000E+000, -0.000000000000E+000 ]
```

4.8.5 Matrix–Potenzen

4.8.5.1 Funktionen zur Berechnung der Potenzen

Die nachfolgende Funktion liefert mit `P := Power(A,m)` zu einer gegebenen quadratischen Matrix $\boldsymbol{A}$ mit `P` eine Näherung oder Einschließung der Matrix–Potenz $\boldsymbol{A}^m$, je nachdem ob `P` eine Punkt- oder Intervall–Matrix ist. Der ganzzahlige Exponent m darf nicht negativ sein.

function `Power` (**var** A : ■ ; m : `integer`) : □ ;

Die Zuordnung der Symbole ■, □ zu den jeweils erlaubten Datentypen und die exportierenden Module findet man in nachfolgender Tabelle:

Zuordnung von ■, □ zu den erlaubten Datentypen		
`A` : ■	`(L)rmatrix`	`(L)imatrix`
Ergebnistyp □	`(L)rmatrix`	`(L)imatrix`
exportierende Module	(l)mvf1	(l)mvif1
`A` : ■	`(L)rmatrix`	`(L)Staggrmatrix`
Ergebnistyp □	`(L)Staggrmatrix`	
exportierende Module	(l)sgmf1	
`A` : ■	`(L)cmatrix`	`cimatrix`
Ergebnistyp □	`(L)cmatrix`	`cimatrix`
exportierende Module	(l)mvcf1	mcif1
`A` : ■	`(L)cmatrix`	`(L)Staggcmatrix`
Ergebnistyp □	`(L)Staggcmatrix`	
exportierende Module	(l)smcf1	

Die Symbole ■, □ in der Funktion `Power` enthalten entweder nur mit ( ) versehene Namen oder nur Namen ohne ( ) Klammern. Danach ist z.B. die Anweisung `P := Power(A,m)` mit `A: Lrmatrix` und `P: rmatrix` unzulässig. Die analoge Regelung gilt auch für die Namen der exportierenden Module aus obiger Tabelle. Die Funktion `Power(A,m)` mit `A : Lrmatrix` und `P : Lrmatrix` wird daher vom Modul `lmvf1` exportiert und nicht etwa vom Modul `mvf1`.

4.8.5.2 Programmbeispiele

Das folgende Programm `tstpower1` demonstriert eine einfache Anwendung der Potenzfunktion `Power(A,m)`. Wählt man $\boldsymbol{A} = 2 \cdot \boldsymbol{I}$, so gilt $\boldsymbol{A}^{33} = 2^{33} \cdot \boldsymbol{I}$, und die Näherung `P := Power(A,m)` $\approx \boldsymbol{A}^m$ liefert bei diesem Beispiel sogar den exakten Wert `P[50,50]` $= 2^{33} = 8589934592$.

```
program tstpower1(input,output);
{ Programm zur Berechnung von Matrix-Potenzen }
use lmvf1  , { exportiert Power-Funktion }
    lmv_ari; { Exportiert     writeln(P) }
var A,P : Lrmatrix[1..50,1..50];
begin
   A := 2.0 * Lid(A); { A = 2    * Einheitsmatrix   }
   P := Power(A,33);  { P = 2^33 * Einheitsmatrix   }
   writeln(P[50,50]); { Ausgabe von P[50,50] = 2^33 }
end.
```

Hinweis:

Falls bei komplizierteren Matrizen $\boldsymbol{A}$ die Genauigkeit der Näherung `P` $\approx \boldsymbol{A}^m$ nicht mehr ausreicht, sollte man mit `P := Power(A,m)` eine Einschließung `P : LStaggrmatrix` von $\boldsymbol{A}^m$ in höherer Präzision berechnen. Die Funktion `Power(A,m)` wird vom Modul `lsgmf1` exportiert. Das folgende Programm `tstpower2` berechnet mit `P` eine Einschließung von $\boldsymbol{A}^{33}$, wobei $\boldsymbol{A}$ als Zufallsmatrix mit `Rand_M(A,-30.0,30.0)` berechnet wird.

```
program tstpower2(input,output);
{ Programm zur Berechnung von Matrix-Potenzen }
{ in staggered Arithmetik in hoher Praezision }
use type_def, { exportiert Datentypen     }
    lsgmf1  , { exportiert Power-Funktion }
    lstagg  , { exportiert     writeln(P) }
    mvran   ; { exportiert Rand_M         }
var p : integer;
procedure main;
var A : Lrmatrix[1..20,1..20];
   P  : LStaggrmatrix[1..20,1..20,-1..LStagg_prec];
```

```
begin
   Rand_M(A,-30.0,30.0);       { A: Zufallsmatrix }
   P := Power(A,33); { P: Einschliessung von A^33 }
   writeln('P[20,20] =');
   writeln(P[20,20]);      { Ausgabe von P[20,20] }
end; { main }
begin
   writeln('LStagg_prec = ',get_LStagg_prec);
   writeln('LStagg_prec = ?');       read(p);
   set_LStagg_prec(p); { Setzt LStagg_prec fest }
   main;
end.
```

Das Programm `tstpower2` liefert die folgende Bildschirmausgabe:

```
LStagg_prec = 1
LStagg_prec = ?
2       <enter>
P[20,20] =
-2.51944509067859264345E+060
-7.07726546438221619625E+039
[ -3.01792531173450820058E+019, -3.01792531173450819881E+019 ]
```

4.9 *LU*–Zerlegung quadratischer Matrizen

4.9.1 Definitionen

Nach [19, Seite 18] gilt für die $\boldsymbol{LU}$–Zerlegung einer Matrix $\boldsymbol{A}$ der folgende

> **Satz 2**
>
> **Vorauss**: $\boldsymbol{A}$ ist regulär;
>
> **Beh.**: Es gibt eine Permutationsmatrix $\boldsymbol{P}$, eine normierte untere Dreiecksmatrix $\boldsymbol{L}$ und eine obere Dreiecksmatrix $\boldsymbol{U}$ mit: $\boldsymbol{P} \cdot \boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U}$.

Anmerkungen:

- Die Permutationsmatrix $\boldsymbol{P}$ entsteht aus der Einheitsmatrix $\boldsymbol{I}$ durch geeignete Vertauschung der Zeilen–Einheitsvektoren und kann mit dem Permutationsvektor $\boldsymbol{p}$ wie folgt definiert werden:
  Der Zahlenwert $\boldsymbol{p}[k] = i$ schreibt in die k-te Zeile von $\boldsymbol{P}$ den i-ten Einheitsvektor. Demnach definiert der Vektor $\boldsymbol{p}^T = (1, 3, 2)$ die Matrix
  $$\boldsymbol{P} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad \text{und mit} \quad \boldsymbol{A} = \begin{pmatrix} 3 & -1 & 0 \\ 3 & -1 & 1 \\ 0 & 2 & 1 \end{pmatrix} \quad \text{gilt:}$$
  $$\boldsymbol{P} \cdot \boldsymbol{A} = \begin{pmatrix} 3 & -1 & 0 \\ 0 & 2 & 1 \\ 3 & -1 & 1 \end{pmatrix}, \text{ d.h. zweite und dritte Zeile werden vertauscht.}$$
  Nach obigem Satz gilt mit $\boldsymbol{L} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ und $\boldsymbol{U} = \begin{pmatrix} 3 & -1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$
  die Zerlegung: $\boldsymbol{P} \cdot \boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U}$.

- Die Diagonalelemente von $\boldsymbol{L}$ haben den Wert 1. Durch diese Normierung ist die $\boldsymbol{LU}$–Zerlegung $\boldsymbol{P} \cdot \boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U}$ sogar eindeutig.

4.9.2 Routinen zur *LU*–Zerlegung

Die folgenden Prozeduren berechnen die $\boldsymbol{LU}$–Zerlegung nach Crout's Algorithmus, der die Dreiecksmatrizen $\boldsymbol{L}, \boldsymbol{U}$ zu einer Matrix $\boldsymbol{Z}$ zusammenfaßt. Die Diagonalelemente 1 von $\boldsymbol{L}$ werden dabei durch die Diagonalelemente von $\boldsymbol{U}$ überschrieben.

procedure `LU` (**var** A,Z : ■ ;
var p : `intvector`; **var** Err : `integer`);

procedure `LU` (**var** A : ◇ ; **var** Z : □ ;
var p : `intvector`; **var** Err : `integer`);

Die Zuordnung der Symbole ■ bzw. ◇, □ zu den jeweils erlaubten Datentypen von `A,Z` und die jeweils exportierenden Module findet man in nachfolgender Tabelle:

Zuordung der Symbole ■, ◇, □ zu den erlaubten Datentypen bzgl. der Prozeduren zur *LU* Zerlegung			
`A,Z` : ■	`(L)rmatrix`	`(L)imatrix`	`(L)cmatrix`
exportrnd. Module	(l)lu	(l)ilu	(l)clu
`A,Z` : ■	`cimatrix`	`(L)Staggcmatrix`	
exportrnd. Module	cilu	(l)sclu	
`A` : ◇	`(L)rmatrix`	`rmatrix`	`Lrmatrix`
`Z` : □	`(L)imatrix`	`Staggrmatrix`	`Staggrmatrix`
exportrnd. Module	(l)lu	stglu	lstglu
`A` : ◇	`cmatrix`	`(L)cmatrix`	
`Z` : □	`cimatrix`	`(L)Staggcmatrix`	
exportrnd. Module	clu	(l)sclu	

Anmerkungen:

- Gemischte Typen, wie z.B. `Lrmatrix,rmatrix` dürfen innerhalb einer Prozedur nicht auftreten.
- Punktförmige Matrizen `Z` liefern nur Näherungen, während intervallmäßige `Z` die $\boldsymbol{LU}$–Zerlegung von `A` garantiert einschließen.

Beispiel:

Mit der regulären Matrix $\boldsymbol{A} = \begin{pmatrix} 3 & -1 & 0 \\ 3 & -1 & 1 \\ 0 & 2 & 1 \end{pmatrix}$ liefert der Prozeduraufruf `LU(A,Z,p,Err)` mit `A,Z : rmatrix` und dem Fehler–Kode `Err=0` folgende Ergebnisse:

$$\boldsymbol{p} = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}, \; \boldsymbol{Z} = \begin{pmatrix} 3 & -1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 1 \end{pmatrix}, \text{ d.h. } \boldsymbol{L} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \; \boldsymbol{U} = \begin{pmatrix} 3 & -1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix},$$

wobei die Dreiecksmatrizen $\boldsymbol{L}, \boldsymbol{U}$ sogar exakt berechnet werden.

4.9.3 Programmbeispiele

Im nachfolgenden Programm `lu1` wird zunächst mit `Rand_M(A,c1,c2)` eine komplexe Zufallsmatrix `A` berechnet, in den Bereichen $10 < \Re(A[i,j]) < 500$ und $1 < \Im(A[i,j]) < 20$. Die Prozedur `LU(A,Z,pv,Err)` liefert mit der komplexen Intervallmatrix `Z` eine Einschließung der Dreiecksmatrizen `L,U`, die mit den Prozeduren `MLU_to_L(Z,L)` und `MLU_to_U(Z,U)` realisiert werden. Nach der Berechnung der Permutationsmatrix `P` aus dem Permutationsvektor `pv` wird diese Einschließung überprüft, indem gezeigt wird, daß `P*A` in `L*U` enthalten ist.

```
program lu1(input,output);
use type_def,  { exportiert Datentypen                 }
    clu     ,  { exportiert Prozedur LU(A,Z,p,Err)     }
    c_ari   ,  { komplexe Arithmetik                   }
    ci_ari  ,  { komplexe Intervall-Arithmetik         }
    mvcran  ,  { exportiert Prozedur Rand_M(A,c1,c2)   }
    mvc_ari ,  { komplexe Matrix/Vektor-Arithmetik     }
```

```
    mvci_ari,  { komplexe Matrix/Vektor-Interv.-Arithm. }
    cswap   ;  { exportiert MLU_to_L(Z,L),MLU_to_U(Z,U) }
var A     : cmatrix[1..30,1..30];
   P      : rmatrix[1..30,1..30];   { Permutationsmatrix }
   Z,L,U  : cimatrix[1..30,1..30];
   pv     : intvector[1..30];       { Permutationsvektor }
   Err,k  : integer;
   c1,c2  : complex;  { Kompl. Zahlen fuer Zufallsmatrix }

begin
   c1 := compl(10,1);  c2 := compl(500,20);
   Rand_M(A,c1,c2); { A: komplexe Zufallsmatrix     }
   LU(A,Z,pv,Err);  { Z: Einschlg. der LU-Zerlegung }
   if Err = 0 then
      begin
         { Berechnung der Permutationsmatrix P mit dem }
         { Permutationsvektor pv:                      }
         P := NULL(A); { erzeugt zunaechst Null-Matrix }
         For k := 1 to 30 do P[k,pv[k]] := 1;
         { Umformung von Z nach L und U:          }
         MLU_to_L(Z,L); { L: untere Dreiecksmatrix }
         MLU_to_U(Z,U); { U: obere  Dreiecksmatrix }
         if P*A in L*U then writeln('P*A in L*U enthalten')
                       else writeln('!! trouble !!');
      end else writeln('Err = ',Err);
end.
```

Der Programmstart liefert mit dem Fehler–Kode `Err=0` die Bildschirmausgabe: `P*A in L*U enthalten`.

Bei höheren Dimensionen von `A` erhält man Fehlermeldungen, da dann wegen der immer stärkeren Intervallüberschätzungen irgendwann einmal durch ein Intervall zu dividieren ist, welches nur wegen dieser Überschätzungen die Zahl Null enthält. Abhilfe schafft dann nur noch die Berechnung der ***LU***–Zerlegung mit `Z` vom Typ `Staggcmatrix`. Der Anwender findet das entsprechende Programm `lu2` auf der folgenden Seite:

```
program lu2(input,output);
{ Berechnung einer komplexen LU-Zerlegung in staggered  }
{ Arithmetik:                                           }
use type_def,  { exportiert benoetigte Datentypen       }
    sclu    ,  { exportiert Prozedur LU(A,Z,p,Err)      }
    c_ari   ,  { komplexe Arithmetik                    }
    mvcran  ,  { exportiert Prozedur Rand_M(A,c1,c2)    }
    mvc_ari ,  { komplexe Matrix/Vektor-Arithmetik      }
    smvc    ,  { komplexe Matrix/Vektor-Staggd.-Arithm. }
    scswap  ;  { exportiert MLU_to_L(Z,L),MLU_to_U(Z,U) }
var p : integer;
procedure main;
var A     : cmatrix[1..60,1..60];
   P      : rmatrix[1..60,1..60];   { Permutationsmatrix }
   Z,L,U : Staggcmatrix[1..60,1..60,1..2,-1..Stagg_prec];
   pv     : intvector[1..60];       { Permutationsvektor }
   Err,k : integer;
   c1,c2 : complex;  { Kompl. Zahlen fuer Zufallsmatrix }

begin
   writeln('Bitte ein wenig Geduld ...');
   c1 := compl(10,1);  c2 := compl(500,20);
   Rand_M(A,c1,c2); { A: komplexe Zufallsmatrix     }
   LU(A,Z,pv,Err);  { Z: Einschlg. der LU-Zerlegung }
   if Err = 0 then
      begin
         { Berechnung der Permutationsmatrix P mit dem }
         { Permutationsvektor pv:                      }
         P := NULL(A); { erzeugt zunaechst Null-Matrix }
         For k := 1 to 60 do P[k,pv[k]] := 1;
         { Umformung von Z nach L und U:           }
         MLU_to_L(Z,L); { L: untere Dreiecksmatrix }
         MLU_to_U(Z,U); { U: obere  Dreiecksmatrix }
         if P*A in L*U then writeln('P*A in L*U enthalten')
                       else writeln('!! trouble !!');
      end else writeln('Err = ',Err);
end; { main }

begin
   writeln('Stagg_prec = ',get_Stagg_prec);
```

```
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

Nach dem Programmstart von `lu2` mit einer komplexen Zufallsmatrix `A` der Dimension $n = 60$ erhält man folgende Bildschirmausgabe:

```
Stagg_prec = 1
Stagg_prec = ?
2      <enter>
Bitte ein wenig Geduld ...
P*A in L*U enthalten
```

4.10 QR–Zerlegung quadratischer Matrizen

4.10.1 Definitionen

Nach [19, Seite 306] gilt für die $\boldsymbol{QR}$–Zerlegung einer Matrix $\boldsymbol{A}$ der folgende

Satz 3

Vorauss: $\boldsymbol{A}, \boldsymbol{Q}, \boldsymbol{R} \in \mathbb{C}^{n\times n}$, $\boldsymbol{A}$ sei regulär;

Beh.: Es gibt eine unitäre Matrix $\boldsymbol{Q}$ und eine obere Dreiecksmatrix $\boldsymbol{R}$ mit: $\boldsymbol{A} = \boldsymbol{Q} \cdot \boldsymbol{R}$.

Anmerkungen:

1. Eine quadratische Matrix $\boldsymbol{Q}$ mit komplexen Elementen heißt *unitär*, wenn gilt: $\boldsymbol{Q}^* = \boldsymbol{Q}^{-1}$, $\boldsymbol{Q}^* := \overline{\boldsymbol{Q}}^T$, dabei ist $\overline{\boldsymbol{Q}}$ die zu $\boldsymbol{Q}$ konjugiert komplexe Matrix. Es gilt also nach obigem Satz: $\boldsymbol{QQ}^* = \boldsymbol{Q}^*\boldsymbol{Q} = \boldsymbol{I}$. Statt $\boldsymbol{Q}^*$ benutzt man oft auch die Symbolik $\boldsymbol{Q}^H$, wobei dann H der Hermite'sche Operator ist.

2. Bei reellem $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ gilt: $\boldsymbol{QQ}^T = \boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$, d.h. $\boldsymbol{Q}$ ist orthogonal.

3. Die Zerlegung $\boldsymbol{A} = \boldsymbol{QR}$ ist nicht eindeutig, denn mit der beliebigen unitären Diagonalmatrix $S := \operatorname{diag}(e^{i\alpha_1}, e^{i\alpha_2}, \ldots, e^{i\alpha_n})$, $\alpha_i \in \mathbb{R}$, ist $(\boldsymbol{QS})$ wieder unitär, und mit der oberen Dreiecksmatrix $(\boldsymbol{S}^H\boldsymbol{R})$ gilt die Beziehung: $(\boldsymbol{QS})(\boldsymbol{S}^H\boldsymbol{R}) = \boldsymbol{QR} = \boldsymbol{A}$.

4. Mit $\boldsymbol{A}' := \boldsymbol{RQ}$ gilt: $\boldsymbol{A}' = \boldsymbol{QR} = \boldsymbol{Q}^{-1}\boldsymbol{AQ}$, d.h. $\boldsymbol{A}'$ ist orthogonal ähnlich zu $\boldsymbol{A}$; vergl. z.B. [19, Seite 308].

5. Die Spalten einer regulären Matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ bilden eine Basis des $\mathbb{C}^n$. Ist $\boldsymbol{Q}$ die unitäre Matrix der $\boldsymbol{QR}$–Zerlegung von $\boldsymbol{A}$, so bilden die Spaltenvektoren von $\boldsymbol{Q}$ eine orthonormierte Basis des $\mathbb{C}^n$.

6. Die $\boldsymbol{QR}$–Zerlegung spielt insbesondere bei der Eigenwertberechnung eine zentrale Rolle, da die Faktorzerlegung $\boldsymbol{A} = \boldsymbol{QR}$ mit Hilfe von Householder–Matrizen numerisch stabil durchgeführt werden kann, vergl. dazu z.B. [21],[20],[6],[19].

4.10.2 Routinen zur QR–Zerlegung

Die folgenden Routinen liefern zu der punktförmigen, quadratischen Eingangsmatrix `A` mit den punktförmigen Ausgangsmatrizen `Q,R` wirklich gute Approximationen der entsprechenden exakten Matrizen $\boldsymbol{Q}, \boldsymbol{R}$. Sind jedoch die Elemente der Ausgangsmatrizen `Q,R` vom Typ `Stagg` oder `LStagg`, so erhält man garantierte Einschließungen der entsprechenden exakten Elemente.

function `QR` (**var** A : ■) : ■ `[1..ub(A),1..ub(A)]`;

`Q := QR(A)` liefert mit `Q` eine Approximation der unitären oder orthogonalen Matrix $\boldsymbol{Q}$, wobei `A` verändert wird. Notwendiger Indexbereich der Matrix `A: [1..ub(A),1..ub(A)]`, sonst erfolgt eine entsprechende Fehlermeldung.

procedure `QR_DEC` (**var** A,Q,R : ■);

`QR_DEC(A,Q,R)` liefert zur Eingangsgröße `A` mit `Q,R` Approximationen zur Zerlegung $\boldsymbol{A} = \boldsymbol{QR}$, wobei `A` unverändert bleibt. Notwendiger Indexbereich der quadratischen Matrizen `A,Q,R` ist `[1..ub(A),1..ub(A)]`, sonst erfolgt eine entsprechende Fehlermeldung. Die folgende Tabelle liefert für ■ die erlaubten Datentypen und die jeweils exportierenden Module:

Zuordung von ■ zu den erlaubten Datentypen		
`A,Q,R` : ■	`(L)rmatrix`	`(L)cmatrix`
exportierende Module	(l)qr	(l)qr_c

Die folgenden Routinen liefern Einschließungen der exakten Matrizen $\boldsymbol{Q}, \boldsymbol{R}$:

function `QR` (**var** `A : □`) : `(L)Staggrmatrix`
`[1..ub(A),1..ub(A),-1..(L)Stagg_prec]`;

procedure `QR_DEC` (**var** `A : □`;
var `Q,R : (L)Staggrmatrix`);

Die beiden ersten Indexbereiche von `A : [1..ub(A),1..ub(A)]`;
Indexbereich von `Q,R: [1..ub(A),1..ub(A),-1..(L)Stagg_prec]`;

Zuordnung von $\square$ zu den erlaubten Datentypen		
`A` : $\square$	`(L)rmatrix`	`(L)Staggrmatrix`
`Q,R` : $\square$	`(L)Staggrmatrix`	
exportierende Module	(l)stqr	

Im Gegensatz zur Prozedur `QR_DEC(A,Q,R)` wird die Eingangsgröße `A` durch den Funktionsaufruf `Q := QR(A)` verändert; Es gilt: $\boldsymbol{Q} \in$ `Q`, $\boldsymbol{R} \in$ `R`, $\boldsymbol{A} = \boldsymbol{QR}$.

4.10.3 Programmbeispiele

4.10.3.1 QR–Zerlegung

Für $\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ gilt die Zerlegung $\boldsymbol{A} = \boldsymbol{QR}$, mit:

$$\boldsymbol{Q} = \begin{pmatrix} -1/\sqrt{3} & 2/\sqrt{6} & 0 \\ -1/\sqrt{3} & -1/\sqrt{6} & -1/\sqrt{2} \\ -1/\sqrt{3} & -1/\sqrt{6} & +1/\sqrt{2} \end{pmatrix}, \qquad \boldsymbol{R} = \begin{pmatrix} -3/\sqrt{3} & -2/\sqrt{3} & -1/\sqrt{3} \\ 0 & -2/\sqrt{6} & -1/\sqrt{6} \\ 0 & 0 & +1/\sqrt{2} \end{pmatrix}$$

$\boldsymbol{Q}$ ist orthogonal, d.h. $\boldsymbol{QQ}^T = \boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$; Das nachfolgende Programm `tstqr`

```
program tstqr(input,output);
{ A = Q*R; Q: orthogonal; R: obere Dreiecks-Matrix }
use qr, mv_ari;
var A,Q,R,T : rmatrix[1..3,1..3];
begin
   A[1,1] := 1;  A[1,2] := 0;  A[1,3] := 0;
   A[2,1] := 1;  A[2,2] := 1;  A[2,3] := 0;
   A[3,1] := 1;  A[3,2] := 1;  A[3,3] := 1;
   QR_DEC(A,Q,R);     { A bleibt erhalten;   }
   writeln('Q = ',Q); { Approximationen der  }
   writeln('R = ',R); { exakten Matrizen Q,R }
end.
```

liefert mit der obigen Eingangmatrix $\boldsymbol{A}$ für die exakten Matrizen $\boldsymbol{Q}, \boldsymbol{R}$ die folgenden Approximationen:

$$\mathtt{Q} = \begin{pmatrix} -5.773502691900{\cdot}10^{-1} & +8.164965809283{\cdot}10^{-1} & +5.000000000000{\cdot}10^{+0} \\ -5.773502691897{\cdot}10^{-1} & -4.082482904642{\cdot}10^{-1} & -7.071067811867{\cdot}10^{-1} \\ -5.773502691897{\cdot}10^{-1} & -4.082482904639{\cdot}10^{-1} & +7.071067811865{\cdot}10^{-1} \end{pmatrix} \approx \boldsymbol{Q};$$

$$\mathtt{R} = \begin{pmatrix} -1.732050807569{\cdot}10^{+0} & -1.154700538379{\cdot}10^{+0} & -5.773502691897{\cdot}10^{-1} \\ 0 & -8.164965809277{\cdot}10^{-1} & -4.082482904639{\cdot}10^{-1} \\ 0 & 0 & +7.071067811865{\cdot}10^{-1} \end{pmatrix} \approx \boldsymbol{R};$$

Damit gilt also: $\boldsymbol{A} = \boldsymbol{QR} \approx \mathtt{QR}$ und $\mathtt{QQ}^{\mathrm{T}} \approx \boldsymbol{I}$. Für viele Anwendungen sind diese Approximationen völlig ausreichend.

Das nachfolgende Programm `tststqr`

```
program tststqr(input,output);
{ Einschliessungen von Q,R bzgl. A = Q*R }
use type_def, stqr, stgmv, mvi_ari, mv_ari;
var p            : integer;
procedure main(p : integer);
var A    : rmatrix[1..3,1..3];
   QS,RS : Staggrmatrix[1..3,1..3,-1..p];
   Q,R   : imatrix[1..3,1..3];
begin
   A[1,1] := 1;  A[1,2] := 0;  A[1,3] := 0;
   A[2,1] := 1;  A[2,2] := 1;  A[2,3] := 0;
   A[3,1] := 1;  A[3,2] := 1;  A[3,3] := 1;
   QR_DEC(A,QS,RS);
   Q := QS;  R := RS; { Rundung ins real-Format }
   writeln('Q = ',Q);  writeln('R = ',R);
   if A in Q*R then writeln('A in QR enthalten');
   if ID(A) in TRANSP(Q)*Q then
      writeln('I in TRANSP(Q)*Q enthalten');
end; { main }
begin
   writeln('Genauigkeitsparameter = ',get_stagg_prec);
   writeln('p = ?'); read(p); set_Stagg_prec(p); main(p);
end.
```

liefert mit dem Genauigkeitsparameter `p=1` für die exakten Matrizen $\boldsymbol{Q}, \boldsymbol{R}$ die folgenden Einschließungen:

$$\boldsymbol{Q} \in \begin{pmatrix} -5.773502691189^{6}_{7}\cdot 10^{-1} & +8.164965809927^{8}_{7}\cdot 10^{-1} & [-5.4\cdot 10^{-25}, +5.8\cdot 10^{-25}] \\ -5.773502691189^{6}_{7}\cdot 10^{-1} & -4.08248290463^{8}_{9}\cdot 10^{-1} & -7.071067811186^{5}_{6}\cdot 10^{-1} \\ -5.773502691189^{6}_{7}\cdot 10^{-1} & -4.08248290463^{8}_{9}\cdot 10^{-1} & +7.071067811186^{6}_{5}\cdot 10^{-1} \end{pmatrix} = \mathtt{Q};$$

$$\boldsymbol{R} \in \begin{pmatrix} -1.732050807756^{8}_{9}\cdot 10^{+0} & -1.1547005383^{79}_{80}\cdot 10^{+0} & -5.773502691189^{6}_{7}\cdot 10^{-1} \\ 0 & -8.164965809927^{7}_{8}\cdot 10^{-1} & -4.08248290463^{8}_{9}\cdot 10^{-1} \\ 0 & 0 & +7.071067811186^{6}_{5}\cdot 10^{-1} \end{pmatrix} = \mathtt{R};$$

und die Ausgabe: `A in QR enthalten, I in TRANSP(Q)*Q enthalten.`

4.10.3.2 Orthonormalbasis

Die drei linear unabhängigen Spaltenvektoren der Matrix $\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ bilden eine Basis des $\mathbb{R}^3$. Gesucht ist eine Einschließung einer Orthonormalbasis $\{\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3\}$ des $\mathbb{R}^3$, die aus $\boldsymbol{A}$ mit Hilfe des Gram-Schmidt-Verfahrens zu berechnen ist. Eine Koordinatendarstellung des Vektors $\boldsymbol{b} = (1, 1, 0)^T$ ist dann bzgl. dieser Orthonormalbasis zu bestimmen.

Lösung:

Die Spaltenvektoren der Intervall–Matrix

$$\mathtt{Q} = \begin{pmatrix} -5.773502691189^{6}_{7}\cdot 10^{-1} & +8.164965809927^{8}_{7}\cdot 10^{-1} & [-3.9\cdot 10^{-25}, +4.3\cdot 10^{-25}] \\ -5.773502691189^{6}_{7}\cdot 10^{-1} & -4.08248290463^{8}_{9}\cdot 10^{-1} & -7.071067811186^{5}_{6}\cdot 10^{-1} \\ -5.773502691189^{6}_{7}\cdot 10^{-1} & -4.08248290463^{8}_{9}\cdot 10^{-1} & +7.071067811186^{6}_{5}\cdot 10^{-1} \end{pmatrix}$$

sind die gesuchten Einschließungen der Orthonormalbasis $\{\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3\}$, mit:

$$\boldsymbol{q}_1 = \begin{pmatrix} -1/\sqrt{3} \\ -1/\sqrt{3} \\ -1/\sqrt{3} \end{pmatrix}, \quad \boldsymbol{q}_2 = \begin{pmatrix} 2/\sqrt{6} \\ -1/\sqrt{6} \\ -1/\sqrt{6} \end{pmatrix}, \quad \boldsymbol{q}_3 = \begin{pmatrix} 0 \\ -1/\sqrt{2} \\ +1/\sqrt{2} \end{pmatrix},$$

und die Intervall–Matrix `Q` ergibt sich im nachfolgenden Programm `tststqr1` mit `p=1` durch die Anweisungen: `Q := QR(A);` `writeln('Q = ',Q);`
Die gesuchte Koordinatendarstellung $x_1 \cdot \boldsymbol{q}_1 + x_2 \cdot \boldsymbol{q}_2 + x_3 \cdot \boldsymbol{q}_3 = \boldsymbol{b}$ des Vektors $\boldsymbol{b}$ bzgl. der Orthonormalbasis $\{\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3\}$ ist ein lineares Gleichungssystem, dessen Lösungsvektor $\boldsymbol{x}$ durch den Intervall-Lösungsvektor $\boldsymbol{z}$ des Systems $\boldsymbol{Qz} = \boldsymbol{b}$ eingeschlossen wird. Die Anweisung `lss(Q,false,b,z,Err)` im Programm `tststqr1` liefert das Ergebnis:

$$\begin{aligned} x_1 &\in z_1 = -1.1547005383^{79}_{80} \cdot 10^{+0} \\ x_2 &\in z_2 = +4.0824829046^{41}_{37} \cdot 10^{-1} \\ x_3 &\in z_3 = -7.07106781186^{3}_{8} \cdot 10^{-1} \end{aligned}$$

Die obigen Ergebnisse wurden mit dem Programm `tststqr1` berechnet:

```
program tststqr1(input,output);
{ Berechnung einer Orthonormierten Basis }
use type_def, stqr, stgmv, mvi_ari, mv_ari, ilss, i_ari;
var p            : integer;
procedure main(p : integer);
var A  : rmatrix[1..3,1..3];
   QS  : Staggrmatrix[1..3,1..3,-1..p];
   Q   : imatrix[1..3,1..3];
   b,z : ivector[1..3];
   Err : integer;
begin
   A[1,1] := 1;  A[1,2] := 0;   A[1,3] := 0;
   A[2,1] := 1;  A[2,2] := 1;   A[2,3] := 0;
   A[3,1] := 1;  A[3,2] := 1;   A[3,3] := 1;
   b[1] := 1;  b[2] := 1;  b[3] := 0;
   QS := QR(A);  { QS: Orthonormierte Basis }
   Q := QS;      { Rundung ins real-Format  }
   writeln('Q = ',Q);
   lss(Q,false,b,z,Err);
   if Err = 0 then writeln('z = ',z);
end; { main }
begin
   writeln('Genauigkeitsparameter = ',get_stagg_prec);
   writeln('p = ?'); read(p);
   set_Stagg_prec(p);
   main(p);
end.
```

4.11 Inversion quadratischer Matrizen

4.11.1 Übersicht

Ist $\boldsymbol{A}$ eine quadratische, reelle Matrix und ist $\boldsymbol{I}$ die Einheitsmatrix, so hat die Gleichung $\boldsymbol{AX} = \boldsymbol{I}$ für reguläres $\boldsymbol{A}$ die eindeutige Lösung $\boldsymbol{X} = \boldsymbol{A}^{-1}$. Mit der Inversen $\boldsymbol{A}^{-1}$ gelten die Beziehungen:

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}, \qquad (\boldsymbol{A}^{-1})^{-1} = \boldsymbol{A}; \tag{4.4}$$

Da sich die Elemente von $\boldsymbol{A}^{-1}$ bei schlecht-konditioniertem $\boldsymbol{A}$ um viele Größenordnungen ändern können, wenn nur ein a_{ij} lediglich auf der letzten Dezimalstelle variiert wird, spielen unvermeidbare Rundungsfehler in den Gleitkommarechnungen eine große Rolle und liefern für die Inverse i.a. völlig unbrauchbare Ergebnisse. In diesen Fällen sind gerade diejenigen **PASCAL-XSC** Prozeduren des übernächsten Abschnittes von großem Nutzen, die mit dem Fehlerkode Err=0 garantierte Einschließungen von $\boldsymbol{A}^{-1}$ realisieren. Sind diese Einschließungen zu grob, so erhält man i.a. bessere Ergebnisse, wenn man entweder in doppelter Länge rechnet oder die staggered Arithmetik in hoher Präzision anwendet. In jedem Fall enthält die berechnete Einschließung die gesuchte Inverse. Der Fehlerkode Err$\neq$0 signalisiert, daß die Matrix $\boldsymbol{A}$ entweder singulär ist oder daß $\boldsymbol{A}$ so schlecht konditioniert ist, daß eine Einschließung der Inversen nicht mehr realisiert werden konnte.

Neben diesen Einschließungs–Routinen stehen weitere Prozeduren zur Verfügung, mit denen man Punkt–Approximationen der Inversen $\boldsymbol{A}^{-1}$ in einfacher oder doppelter Länge berechnen kann. Diese Approximationen werden u.a. benötigt, wenn man für den Lösungsvektor $\boldsymbol{x}$ eines linearen Gleichungssystems $\boldsymbol{Ax} = \boldsymbol{b}$ eine garantierte Einschließung berechnen will.
Für die Matrix $\boldsymbol{A}$ stehen die folgenden Datentypen zur Verfügung:

```
rmatrix, Lrmatrix, imatrix, Limatrix, cmatrix, cimatrix
```

Damit lassen sich also neben reellen auch komplexe Matrizen invertieren, und die Intervallmatrizen werden dann benötigt, wenn z.B. $a_{ij} = \sqrt{2}$ als Gleitpunkzahl nicht darstellbar ist und durch ein Intervall maximalgenau einzuschließen ist.

Der mathematisch interessierte Anwender findet im folgenden Abschnitt eine kurze Zusammenfassung der verwendeten Algorithmen. Eine sehr ausführliche Beschreibung kann in [3] nachgelesen werden. Alle zur Verfügung stehenden Routinen zur Matrixinversion sind im darauf folgenden Abschnitt zusammengestellt.

4.11.2 Inversions–Algorithmen

In diesem Abschnit findet man lediglich eine grobe Zusammenstellung der verwendeten Inversions–Algorithmen, wobei wir uns zunächst auf reelle Punktmatrizen beschränken können. In [3] werden die Algorithmen an Hand vieler numerischer Beispiele ausführlich beschrieben.

Wir betrachten also eine reelle, reguläre Matrix $\boldsymbol{A}$, für die zunächst eine punktförmige Näherung der Inversen $\boldsymbol{A}^{-1}$ zu berechnen ist.

Grobe Näherungsinverse:
Eine erste Näherung findet man nach folgendem Schema:

1. Berechne LU–Zerlegung von $\boldsymbol{A}$: $\quad \boldsymbol{PA} = LU$;
2. Berechne $\boldsymbol{Y}$ in $\quad \boldsymbol{LY} = PI \quad$ spaltenweise durch Vorwärtseinsetzen
3. Berechne $\boldsymbol{R}$ in $\quad \boldsymbol{UR} = Y \quad$ spaltenweise durch Rückwärtseinsetzen
4. $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$, falls $\boldsymbol{A}$ nicht zu schlecht konditioniert ist.

Falls $\boldsymbol{R}$ zu grob ist, kann diese Näherung weiter verbessert werden, wobei zwischen Rechnungen in einfacher bzw. doppelter Länge zu unterscheiden ist.

Verbesserung der Näherung in einfacher Länge:

1. Berechne die Defekte: $\quad {}^{l}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{RA} \quad$ bzw. $\quad {}^{r}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{AR}$;
2. Wähle diejenige Defektdefinition mit der kleineren Defektnorm;
3. Führe mit der unter 2. gewählten Defektdefinition die entsprechende Defektiteration durch:

$$\begin{array}{rcl} {}^{l}\boldsymbol{D}^{(n)} & := & \boldsymbol{I} - \boldsymbol{R}^{(n)}\boldsymbol{A}; \quad \boldsymbol{R}^{(n+1)} := \boldsymbol{R}^{(n)} + {}^{l}\boldsymbol{D}^{(n)} \cdot \boldsymbol{R}^{(n)}, \quad \boldsymbol{R}^{(0)} = \boldsymbol{R}; \\ {}^{r}\boldsymbol{D}^{(n)} & := & \boldsymbol{I} - \boldsymbol{A}\boldsymbol{R}^{(n)}; \quad \boldsymbol{R}^{(n+1)} := \boldsymbol{R}^{(n)} + \boldsymbol{R}^{(n)} \cdot {}^{r}\boldsymbol{D}^{(n)}, \quad \boldsymbol{R}^{(0)} = \boldsymbol{R}; \end{array}$$

Beachten Sie bitte, daß bei schlecht-konditionierten Matrizen die Konvergenz der Defektiteration ganz wesentlich von der richtigen Wahl der entsprechenden Defektdefinition in 2. abhängt! Beispiele findet man in [3, Seite 41..]

Falls die Defektdefinition nicht konvergiert, kann die Verbesserung der ersten groben Näherung $\boldsymbol{R}$ oft durch entsprechende Rechnungen in doppelter Länge erfolgreich durchgeführt werden.

Verbesserung der Näherung in doppelter Länge:

1. Berechne die Defekte: $\,^l\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{RA}$ bzw. $\,^r\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{AR}$;
2. Wähle diejenige Defektdefinition mit der kleineren Defektnorm;
3. Eine ganz erhebliche Verbesserung wird erreicht, wenn man vor der Defektiteration das Verfahren von Rump anwendet und dabei diejenige Version wählt, die durch die in 2. getroffene Wahl der Defektdefinition bestimmt ist [3, Seite 25..]. Wir formulieren hier die Version, die zur Defektdefinition $\,^r\boldsymbol{D} = \boldsymbol{I} - \boldsymbol{AR}$ gehört:
   - Berechnung von $\boldsymbol{A} \odot \boldsymbol{R}$; $\odot$: maximalgenau Maschinenauswertung
   - Berechnung von $\boldsymbol{S} \approx (\boldsymbol{A} \odot \boldsymbol{R})^{-1}$ mit Hilfe einer LU–Zerlegung von $\boldsymbol{A} \odot \boldsymbol{R}$
   - Exakte Auswertung von $\boldsymbol{R} \cdot \boldsymbol{S} \approx \boldsymbol{A}^{-1}$ im Akkumulator
   - $\boldsymbol{R}_1 := \square(\boldsymbol{R} \cdot \boldsymbol{S})$; $\quad \boldsymbol{R}_2 := \square(\boldsymbol{R} \cdot \boldsymbol{S} - \boldsymbol{R}_1)$;
     $\square$: Rundung zur nächsten Gleitkommazahl;
4. Defektiteration in doppelter Länge; wir formulieren wieder diejenige Version, die zur Defektdefinition $\,^r\boldsymbol{D} = \boldsymbol{I} - \boldsymbol{AR}$ gehört:
   - $\boldsymbol{R}_1^{(0)} := \boldsymbol{R}_1; \qquad \boldsymbol{R}_2^{(0)} := \boldsymbol{R}_2;$
   - $\,^r\boldsymbol{D}^{(n)} := \square(\boldsymbol{I} - \boldsymbol{A}(\boldsymbol{R}_1^{(n)} + \boldsymbol{R}_2^{(n)})); \quad n = 0, 1, \ldots$
   - $\text{Akku} := (\boldsymbol{R}_1^{(n)} + \boldsymbol{R}_2^{(n)}) + (\boldsymbol{R}_1^{(n)} + \boldsymbol{R}_2^{(n)}) \cdot \,^r\boldsymbol{D}^{(n)}$
   - $R_1^{(n+1)} := \square(\text{Akku}); \quad R_2^{(n+1)} := \square(\text{Akku} - R_1^{(n+1)})$
   - Tritt auf der Maschine bei $n = N$ Konvergenz ein, so ist die Summe $\boldsymbol{R}_1^{(N+1)} + \boldsymbol{R}_2^{(N+1)} \approx \boldsymbol{A}^{-1}$ i.a. eine maximalgenaue Näherungsinverse.

Anwendung der staggered Arithmetik:

Falls bei sehr schlecht konditionierten Matrizen $\boldsymbol{A}$ die Rechnungen auch in doppelter Länge nicht zum Ziel führen, kann die LU–Zerlegung mit Hilfe der staggered Arithmetik durchgeführt werden. Da diese Arithmetik eine Intervallrechnung in hoher Präzision realisiert, erhält man für die Inverse $\boldsymbol{A}^{-1}$ sogar eine garantierte Einschließung. Wegen der hohen Laufzeiten sollte dieses Verfahren jedoch nur in wirklichen Notfällen eingesetzt werden.

Einschließung der Inversen:

Wir skizzieren jetzt einen Algorithmus, der zu einer gegebenen Punktnäherung $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$ eine garantierte Einschließung der Inversen liefert und betrachten nur Rechnungen in einfacher Länge des benutzten Datenformats. Das Verfahren beruht auf dem Fixpunktsatz von Brouwer [5, Seite 51] und auf den Ergebnissen von Rump [18],[10, Seite 119]. Benutzt wird diejenige Defekdefinition, die nach Durchführung der LU–Zerlegung und Berechnung der ersten groben Näherung $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$ die kleinere Defektnorm liefert. Die numerische Erfahrung zeigt nämlich, daß der Einschließungs–Algorithmus deutlich besser funktioniert, wenn ganau diese Defektdefinition gewählt wird. Der nachfolgende Algorithmus setzt wieder voraus, daß die Defektdefinition ${}^{r}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{A}\boldsymbol{R}$ die kleinere Defektnorm liefert. Für die Inverse einer regulären, reellen Matrix $\boldsymbol{A}$ machen wir den Ansatz

$$\boldsymbol{A}^{-1} = \boldsymbol{R} + \boldsymbol{Y}$$

und nach elementaren Umformungen findet man die Fixpunktgleichung:

$$\boldsymbol{Y} = f(\boldsymbol{Y}) \quad \text{mit} \quad f(\boldsymbol{Y}) = (\boldsymbol{R} + \boldsymbol{Y}) \cdot {}^{r}\boldsymbol{D}$$

Wenn man mit dem Startwert $[\boldsymbol{Y}]^{(0)} := \boldsymbol{R} \mathbin{\Diamond\!\!\!\!\cdot\,} [{}^{r}\boldsymbol{D}]$ die folgende Iteration

$$[\boldsymbol{Y}]^{(\nu+1)} := (\boldsymbol{R} + [\boldsymbol{Y}]^{(\nu)}) \mathbin{\Diamond\!\!\!\!\cdot\,} [{}^{r}\boldsymbol{D}], \quad \nu = 0, 1, 2, \ldots$$

so lange durchführt, bis für einen Index k die Inklusionsbedingung

$$[\boldsymbol{Y}]^{(k+1)} \overset{\circ}{\subset} [\boldsymbol{Y}]^{(k)}$$

erfüllt ist, so gelten die folgenden Aussagen:

$$\boldsymbol{Y} \in [\boldsymbol{Y}]^{(k+1)} \quad \text{und damit:} \quad \boldsymbol{A}^{-1} \in \boldsymbol{R} \mathbin{\Diamond\!\!\!\!+\,} [\boldsymbol{Y}]^{(k+1)}.$$

$\Diamond\!\!\!\!\cdot\,$, $\Diamond\!\!\!\!+\,$ sind Maschinen–Operatoren für Intervall–Matrizen im Gleitkomma–Format, die mit $[\boldsymbol{Y}]$ oder $[{}^{r}\boldsymbol{D}]$ bezeichnet werden, und das Symbol $\overset{\circ}{\subset}$ verlangt, daß $[\boldsymbol{Y}]^{(k+1)}$ ganz im Innern von $[\boldsymbol{Y}]^{(k)}$ enthalten ist. Weitere Einzelheiten findet man in [18],[5, Seite 175],[3, Seite 73..].

Alle oben beschriebenen Algorithmen lassen sich ohne weiteres auch auf komplexe Matrizen übertragen, wobei die bei reellen Matrizen benutzte Gesamtnorm $\|\boldsymbol{D}\| = n \cdot \max_{ij} |d_{ij}|$ durch den im Komplexen viel einfacher zu berechnenden Term $\max_{ij}(|d_{ij}.Re| + |d_{ij}.Im|)$ ersetzt werden kann. Auch für Intervallmatrizen $[\boldsymbol{A}]$ können die Algorithmen direkt übernommen werden, wenn man zur Berechnung der Näherungsinversen die Matrix $[\boldsymbol{A}]$ durch ihre Mittelpunktsmatrix ersetzt [3, Seite 93..].

4.11.3 Prozeduren zur Matrixinversion

4.11.3.1 Berechnung einer Näherungsinversen nur mit *LU*–Zerlegung

Prozedurvereinbarung:

procedure `inv_LU` (**var** `A,R` : ■; **var** `Err : integer`);

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Matrix
Ausgangsparameter	
R	Näherungsinverse $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit dem Prozeduraufruf `inv_LU(A,R,Err)` wird mit Hilfe der LU–Zerlegung von **A** eine brauchbare Näherungsinverse $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$ berechnet, wenn **A** nicht zu schlecht konditioniert ist. Die quadratischen Matrizen **A**, **R** müssen die gleiche Dimension besitzen.

Erlaubte Datentypen:

Erlaubte Datentypen in `inv_LU`		
$\boldsymbol{A}, \boldsymbol{R}$: ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

Berechnet werden damit punktförmige Näherungen $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$ der Inversen $\boldsymbol{A}^{-1}$ für reelle oder komplexe Matrizen $\boldsymbol{A}$.

Weitere Informationen:
Vergl. Seite 262 und [3, Seite 16].

4.11.3.2 Berechng. einer Näherungsinversen mit *LU*-Zerlegung oder mit Verfahren von Rump

Prozedurvereinbarung:

```
procedure inv_Rump ( var A : ■; illcd, factor : integer;
                     var R : ■; var Err : integer );
```

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Matrix
illcd	Steuert den internen Algorithmus
factor	Bestimmt die Versionen des Verfahrens von Rump
Ausgangsparameter	
R	Näherungsinverse $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit `inv_Rump(A,illcd,factor,R,Err)` wird $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$ berechnet.
`illcd=0`: Es wird nur die LU–Zerlegung benutzt.
`illcd=1`: Es wird das Verfahren von Rump angewandt, wobei der integer–Parameter `factor` die Version links (0) bzw. rechts (1) bestimmt.
Die quadratischen Matrizen **A**, **R** müssen die gleiche Dimension besitzen.

Erlaubte Datentypen:

Erlaubte Datentypen in `inv_Rump`		
$\boldsymbol{A}, \boldsymbol{R}$: ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

Berechnet werden damit punktförmige Näherungen $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$ der Inversen $\boldsymbol{A}^{-1}$ für reelle oder komplexe Matrizen $\boldsymbol{A}$.

Weitere Informationen: Seite 25 in [3].

4.11.3.3 Näherungsinverse nach Rump in doppelter Länge

Prozedurvereinbarung:

procedure `inv_Rump` (**var** `A` : ■; `factor : integer`;
var `R1,R2` : ■; **var** `Err : integer`);

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Matrix
factor	Bestimmt die Versionen des Verfahrens von Rump
Ausgangsparameter	
R1,R2	Näherungsinverse $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit `inv_Rump(A,factor,R1,R2,Err)` wird $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$ berechnet.
`factor=0`: Es wird die Version *links* gewählt.
`factor=1`: Es wird die Version *rechts* gewählt.
Die quadratischen Matrizen **A**,**R1**,**R2** müssen gleiche Dimensionen besitzen.

Erlaubte Datentypen:

Erlaubte Datentypen in `inv_Rump`		
$\boldsymbol{A}, \boldsymbol{R}$: ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

Berechnet werden punktförmige Näherungen $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$ der Inversen $\boldsymbol{A}^{-1}$ für reelle oder komplexe Matrizen $\boldsymbol{A}$.

Weitere Informationen:
Seite 30 in [3].

4.11.3.4 Berechnung einer Näherungsinversen nach Rump und Wahl der Version mit minimaler Defektnorm

Prozedurvereinbarung:

```
procedure inv_Rump ( var A,R1,R2 : ■;
                     var factor,Err : integer );
```

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Matrix
Ausgangsparameter	
R1,R2	Näherungsinverse $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$
factor	Liefert Rump–Version mit kleinster Defektnorm
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit `inv_Rump(A,R1,R2,factor,Err)` wird nach dem Verfahren von Rump eine Näherungsinverse $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$ so berechnet, daß die zugehörige Defektnorm minimal wird.
`factor=0`: Die Defektnorm bzgl. der Rump–Version *links* ist minimal.
`factor=1`: Die Defektnorm bzgl. der Rump–Version *rechts* ist minimal.

Erlaubte Datentypen:

Erlaubte Datentypen in `inv_Rump`		
$\boldsymbol{A}, \boldsymbol{R1}, \boldsymbol{R2}$: ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

Berechnet werden punktförmige Näherungen $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$ der Inversen $\boldsymbol{A}^{-1}$ für reelle oder komplexe Matrizen $\boldsymbol{A}$.

Weitere Informationen: Vergleiche die Seiten 30,39 in [3].

4.11.3.5 Korrektur von $R \approx A^{-1}$ mit Defektiteration und Vorgabe einer Defektdefinition

procedure `inv_Def` (**var** `A` : ■ ; `factor : integer;`
var `R` : ■ ; **var** `Err : integer`);

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Matrix
factor	Legt eine der beiden Defektdefinitionen fest.
R	Erste Näherung der Inversen: $R \approx A^{-1}$
Ausgangsparameter	
R	i.a. maximalgenaue Näherungsinverse $R \approx A^{-1}$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit `inv_Def(A,factor,R,Err)` wird zu gegebener quadratischer Matrix **A** und einer gegebenen Näherung $R \approx A^{-1}$ mit Hilfe einer Defektiteration eine i.a. maximalgenaue Näherung $R \approx A^{-1}$ berechnet. Der integer–Parameter `factor` bestimmt dabei die zu benutzende Defektdefinition:
`factor=0`: Defektdefinition *links*: $^{l}D := I - RA$
`factor=1`: Defektdefinition *rechts*: $^{r}D := I - AR$

Erlaubte Datentypen:

Erlaubte Datentypen in `inv_Def`		
A, R : ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

Berechnet werden punktförmige Näherungen $R \approx A^{-1}$ der Inversen A^{-1} für reelle oder komplexe Matrizen A, wobei die Eingangsnäherung R durch die Ausgangsgröße R überschrieben wird.

Weitere Informationen: Vergl. Seite 262 und [3, Seite 41,43].

4.11.3.6 Näherungsinverse mit Defektiteration, optimale Defektdefinition wird bestimmt

Prozedurvereinbarung:

procedure `inv` (**var** `A,R` : ■ ; **var** `factor,Err : integer`);

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Matrix
Ausgangsparameter	
R	i.a. maximalgenaue Näherungsinverse $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$
factor	Definiert Defektdefinition mit minimaler Defektnorm.
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit `inv(A,R,factor,Err)` wird zu gegebener quadratischer Matrix **A** eine i.a. maximalgenaue Näherung $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$ wie folgt berechnet:
Zunächst LU–Zerlegung, dann Suche nach der Defektdefinition mit minimaler Defektnorm mit anschließender Defektiteration. Die berechnete Näherung besitzt eine minimale Defektnorm. Der integer–Parameter `factor` definiert die benutzte Defektdefinition:

`factor=0`: Defektdefinition *links*: ${}^{l}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{R}\boldsymbol{A}$;
`factor=1`: Defektdefinition *rechts*: ${}^{r}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{A}\boldsymbol{R}$

Erlaubte Datentypen:

Erlaubte Datentypen in `inv`		
$\boldsymbol{A}, \boldsymbol{R}$: ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

Berechnet werden punktförmige Näherungen $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$ der Inversen $\boldsymbol{A}^{-1}$ für reelle oder komplexe Matrizen $\boldsymbol{A}$.

Weitere Informationen: Vergl. Seite 48 in [3].

4.11.3.7 Korrektur von $\boldsymbol{R}_1 + \boldsymbol{R}_2 \approx \boldsymbol{A}^{-1}$ mit Defektiteration und gegebener Defektdefinition

```
procedure inv_Def ( var A : ■; factor : integer;
                    var R1,R2 : ■; var Err : integer );
```

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Matrix
factor	Bestimmt die vorgegebene Defektdefinition.
R1,R2	Grobe Näherungsinverse $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$
Ausgangsparameter	
R1,R2	i.a. maximalgenaue Näherungsinverse $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit dem Aufruf `inv_Def(A,factor,R1,R2,Err)` wird zu gegebener quadratischer Matrix **A** und zu einer vorgegebenen Näherung $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$ mit Hilfe einer Defekt–Iteration eine doppeltlange Näherung $\boldsymbol{R1}+\boldsymbol{R2} \approx \boldsymbol{A}^{-1}$ i.a. maximalgenaue berechnet. Mit dem integer–Parameter `factor` bestimmt man die gewünschte Defektdefinition:

`factor=0`: Defektdefinition *links*: ${}^{l}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{RA}$;
`factor=1`: Defektdefinition *rechts*: ${}^{r}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{AR}$

Erlaubte Datentypen:

Erlaubte Datentypen in `inv_Def`		
$\boldsymbol{A}, \boldsymbol{R}$: ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

Verbessert werden punktförmige Näherungen $\boldsymbol{R}_1 + \boldsymbol{R}_2 \approx \boldsymbol{A}^{-1}$ der Inversen $\boldsymbol{A}^{-1}$ für reelle oder komplexe Matrizen $\boldsymbol{A}$.

Weitere Informationen: Vergl. Seite 59 in [3].

4.11.3.8 $\boldsymbol{R}_1 + \boldsymbol{R}_2 \approx \boldsymbol{A}^{-1}$ mit automatischer Wahl der günstigsten Defektdefinition

Prozedurvereinbarung:

procedure `inv` (**var** `A,R1,R2` : ■; **var** `factor,Err : integer`);

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Matrix
Ausgangsparameter	
R1,R2	i.a. maximalgenaue Näherungsinverse $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$
factor	Liefert die benutzte, günstigste Defektdefinition.
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit `inv(A,R1,R2,factor,Err)` wird zu gegebener quadratischer Matrix **A** mit Hilfe des Rump'schen Verfahrens und anschließender Defekt–Iteration eine doppeltlange Näherung $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$ i.a. maximalgenaue berechnet. Der Parameter `factor` liefert die benutzte, günstigste Defektdefinition:

`factor=0`: Defektdefinition *links*: ${}^{l}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{RA}$;
`factor=1`: Defektdefinition *rechts*: ${}^{r}\boldsymbol{D} := \boldsymbol{I} - \boldsymbol{AR}$

Erlaubte Datentypen:

Erlaubte Datentypen in `inv`		
$\boldsymbol{A}, \boldsymbol{R}$: ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

Verbessert werden punktförmige Näherungen $\boldsymbol{R}_1 + \boldsymbol{R}_2 \approx \boldsymbol{A}^{-1}$ der Inversen $\boldsymbol{A}^{-1}$ für reelle oder komplexe Matrizen $\boldsymbol{A}$.

Weitere Informationen:
Vergl. Seite 263 und [3, Seite 60].

4.11.3.9 $A^{-1} \in S$ für Punktmatrix A in staggered Arithmetik

Prozedurvereinbarung:

```
procedure inv ( var A : ■ ; var S : ◇ ;
                var Err : integer );
```

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Punkt–Matrix
Ausgangsparameter	
S	Einschließung der Inversen: $A^{-1} \in S$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit `inv(A,S,Err)` wird zu gegebener Matrix **A** mit **S** eine Einschließung der Inversen $A^{-1} \in S$ in staggered Arithmetik berechnet. Um den Genauigkeitsparameter p festzulegen, muß vorher die Prozedur `set_(L)Stagg_prec(p)` aufgerufen werden, womit dann $(p+1) \cdot (21)13$ Dezimalziffern für die interne Rechnung zur Verfügung stehen. Die Prozedur `set_(L)Stagg_prec(...)` wird vom Modul `type_def` exportiert.

Erlaubte Datentypen:

Erlaubte Datentypen in `inv`		
A : ■	`(L)rmatrix`	`cmatrix`
S : ◇	`(L)Staggrmatrix`	`Staggcmatrix`
exportierende Module	(l)inv	invc

Weitere Informationen:
Vergl. Seite 263 und [3, Seite 66].

4.11.3.10 $\boldsymbol{A}^{-1} \subseteq \boldsymbol{S}$ für Intervallmatrix $\boldsymbol{A}$ in staggered Arithmetik

Prozedurvereinbarung:

procedure `inv` (**var** `A,S` : ■ ; **var** `Err : integer`);

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Intervall–Matrix
Ausgangsparameter	
S	Einschließung der Inversen: $\boldsymbol{A}^{-1} \subseteq \boldsymbol{S}$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit dem Prozeduraufruf `inv(A,S,Err)` erhält man zu gegebener Intervall–Matrix **A** eine Einschließung ihrer Inversen $\boldsymbol{A}^{-1} \subseteq \boldsymbol{S}$ in staggered Arithmetik. Um für die staggered Arithmetik den Genauigkeitsparameter p nach Bedarf festzulegen, muß vorher die Prozedur `set_(L)Stagg_prec(p)` aufgerufen werden, mit der dann $(p + 1) \cdot (21)13$ Dezimalziffern für die interne Rechnung zur Verfügung stehen. Die Prozedur `set_(L)Stagg_prec(...)` wird vom Modul `type_def` exportiert.

Erlaubte Datentypen:

Erlaubte Datentypen in `inv`		
$\boldsymbol{A}, S$: ■	`(L)Staggrmatrix`	`Staggcmatrix`
exportierende Module	inv(l)stg	invcstg

Weitere Informationen:
Vergl. Seite 263 und [3, Seite 66].

4.11.3.11 Einschließung der Inversen $A^{-1} \in X$ bei vorgegebener Punkt–Matrix A

Prozedurvereinbarung:

```
procedure inv ( var A : ■; illcd,Alefeld : integer;
                var X : ◇; var Err : integer );
```

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Quadratische Matrix, deren Inverse einzuschließen ist
illcd	Steuert den internen Algorithmus
Alefeld	Anzahl der Nachiterationen zur Verbesserung von **X**
Ausgangsparameter	
X	i.a. maximalgenaue Einschließung der Inversen $A^{-1} \in X$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit dem Aufruf `inv(A,illcd,Alefeld,X,Err)` wird zu gegebener quadratischer Matrix **A** mit **X** eine Einschließung der Inversen $A^{-1} \in X$ berechnet.
`illcd=0`: LU–Zerlegung von **A**, Defektiteration und dann Verifikation;
`illcd=1`: Verfahren von Rump, Defektiteration und dann Verifikation; alle Rechnungen in doppelter Länge;
`illcd=2`: Berechnung von **X** in *staggered Arithmetik*

Erlaubte Datentypen:

Erlaubte Datentypen in `inv`		
A : ■	`(L)rmatrix`	`cmatrix`
X : ◇	`(L)imatrix`	`cimatrix`
exportierende Module	(l)inv	invc

Weitere Informationen: Vergl. Seite 81 in [3].

4.11.3.12 Einschließung der Inversen $\boldsymbol{A}^{-1} \subseteq \boldsymbol{X}$ bei vorgegebener Intervall–Matrix $\boldsymbol{A}$

Prozedurvereinbarung:

```
procedure inv ( var A : ■; illcd : integer;
                var X : ◇; var Err : integer );
```

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Quadratische Intervall–Matrix, deren Inverse einzuschließen ist
illcd	Steuert den internen Algorithmus
Ausgangsparameter	
X	i.a. maximalgenaue Einschließung der Inversen $\boldsymbol{A}^{-1} \in \boldsymbol{X}$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit dem Aufruf `inv(A,illcd,X,Err)` wird zu gegebener quadratischer Intervall–Matrix **A** eine Einschließung **X** der Inversen $\boldsymbol{A}^{-1} \subseteq \boldsymbol{X}$ berechnet.
`illcd=0`: LU–Zerlegung von **A**, Defektiteration und dann Verifikation;
`illcd=1`: Verfahren von Rump, Defektiteration und danach Verifikation, alle Rechnungen in doppelter Länge;

Erlaubte Datentypen:

Erlaubte Datentypen in `inv`		
$\boldsymbol{A}$: ■	`(L)imatrix`	`cimatrix`
$\boldsymbol{X}$: ◇	`(L)imatrix`	`cimatrix`
exportierende Module	(l)inv	invc

Weitere Informationen:
Siehe Seite 264 und [3, Seite 95].

4.11.3.13 $R_1 + R_2 \approx A^{-1}$ für punktförmige Matrix A in staggered Arithmetik

Prozedurvereinbarung:

procedure `inv_ST` (**var** `A,R1,R2` : ■; **var** `Err : integer`);

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende quadratische Punkt–Matrix
Ausgangsparameter	
R1,R2	i.a. maximalgenaue Näherungsinversen $\boldsymbol{R1} + \boldsymbol{R2} \approx \boldsymbol{A}^{-1}$
Err	Fehlerkode; Err=0: Keine Fehler aufgetreten

Arbeitsweise:
Mit dem Aufruf `inv_ST(A,R1,R2,Err)` wird zu gegebener quadratischer Punkt–Matrix **A** eine doppeltlange Näherung der Inversen $\boldsymbol{A}^{-1}$ wie folgt berechnet:
Berechnung einer ersten Näherung in *staggered arithmetic* und Rundung dieser Näherung ins doppeltlange Format. Die Genauigkeit hängt ab vom Parameter `p`, der vorher mit Hilfe der Prozedur `set_(L)stagg_prec(p)` aus dem Modul `type_def` zu wählen ist.
Die quadratischen Matrizen **A**,**R1**,**R2** müssen gleiche Dimensionen besitzen.

Erlaubte Datentypen:

Erlaubte Datentypen in `inv_ST`		
$\boldsymbol{A}, \boldsymbol{R1}, \boldsymbol{R2}$: ■	`(L)rmatrix`	`cmatrix`
exportierende Module	(l)inv	invc

4.11.3.14 Verbesserung einer Einschließung der Inversen nach Alefeld/Herzberger

Prozedurvereinbarung:

procedure `Alefeld_IT` (**var** `A` : ■; **var** `X` : ◇);

Eingangs-/Ausgangs–Parameter:

Eingangsparameter	
A	Zu invertierende reguläre, quadratische Punkt–Matrix
X	Einschließung der Inversen $\boldsymbol{A}^{-1}$
Ausgangsparameter	
X	Verbesserte Einschließung der Inversen $\boldsymbol{A}^{-1}$
Anmerkung:	
Die Eingangsgröße **X** wird durch die Ausgangsgröße **X** überschrieben	

Arbeitsweise:
Mit dem Aufruf `Alefeld_IT(A,X)` wird eine schon existierende Einschließung **X** der Inversen $\boldsymbol{A}^{-1}$ i.a. verbessert. Die quadratischen Matrizen **A**, **X** müssen gleiche Dimensionen besitzen. $\boldsymbol{A}$ muß eine Punktmatrix sein.

Erlaubte Datentypen:

Erlaubte Datentypen in `Alefeld_IT`		
$\boldsymbol{A}$: ■	`(L)rmatrix`	`cmatrix`
$\boldsymbol{X}$: ◇	`(L)imatrix`	`cimatrix`
exportierende Module	(l)inv	invc

Weitere Informationen:
[3, Seite 84],[1, Abschnitt 18].

4.11.4 Programmbeispiele

4.11.4.1 Komplexe Blockmatrizen

```
program tstinv1(input,output);
{ Einschliessung der Inversen einer komplexen Blockmatrix: }
use invc, { Exportiert inv(...)              }
ci_ari  , { Komplexe Intervall-Arithmetik     }
c_ari   , { Komplexe Arithmetik              }
mvc_ari , { Komplexe Matrix/Vektor-Arithmetik }
block   ; { Exportiert  block(U,V,B)         }
var U,V      : cmatrix [1..10,1..10];
   B         : cmatrix [1..20,1..20];
   X         : cimatrix[1..20,1..20];
   Err,illcd : integer;
   dp        : real;

begin
   U := compl(1,0);  { Identische Matrix U, mit: u_(ij) = 1 }
   repeat
      writeln;
      writeln('Diagonal-Parameter von V = ?');  read(dp);
      V := compl(dp,0)*id(V); { Diagonalmatrix: v_(jj) = dp }
      block(U,V,B);
      writeln('illcd=0: Rechnung in einfacher Laenge');
      writeln('illcd=1: Rechnung in doppelter Laenge');
      writeln('illcd=2: Rechnung in staggered Arithmetik');
      writeln('illcd = ?');  read(illcd);
      inv(B,illcd,0,X,Err);  { X: Einschliessung von B^(-1) }
      if Err=0 then
      begin
      writeln('B^(-1)[',ub(B):2,',',ub(B,2),'] enthalten in:');
      writeln(X[ub(X,1),ub(X,2)])
      end      else writeln('Err = ',Err);
   until false;
end.
```

Das obige Programm `tstinv1` berechnet die Inverse einer Blockmatrix $\boldsymbol{B}$, vergl. Seite 167, der Dimension $n = 20$ mit Hilfe der Prozedur `inv` von Seite 275. Das Diagonalelement $v_{jj} =$ `dp` der Diagonalmatrix $\boldsymbol{V}$ kann vom Anwender frei gewählt werden:

```
Diagonalelement von V = ?
1e-20      <enter>
illcd=0: Rechnung in einfacher Laenge
illcd=1: Rechnung in doppelter Laenge
illcd=2: Rechnung in staggered Arithmetic
illcd = ?
1          <enter>
B^(-1)[20,20] enthalten in:
( [  1.000000000000E-002,  1.000000000001E-002 ],
  [ -1.000000000000E-267,  1.000000000000E-267 ] )
```

4.11.4.2 Boothroyd/Dekker–Matrizen

```
program tstinv2(input,output);
use type_def, lmv_exap, linv, lmvi_ari;
var n : integer;
 procedure main(n: integer);
 var A         : Lrmatrix[1..n,1..n];
     v         : Lrvector[1..n];
     X         : Limatrix[1..n,1..n];
     Err,illcd : integer;
 begin { main }
    set_maxprec(7); { Genauigkeit zur Berechnung von: }
    Dekker(A,v);    { A: Boothroyd/Dekker-Matrix      }
    writeln('illcd=0: Rechnung in einfacher Laenge');
    writeln('illcd=1: Rechnung in doppelter Laenge');
    writeln('illcd=2: Rechnung in staggered Arithmetik');
    writeln('illcd = ?');    read(illcd);
    inv(A,illcd,0,X,Err);
    if Err = 0 then
    begin
       writeln('Einschliessung der inversen Dekker-Matrix:');
       writeln(X);
    end        else writeln('Err = ',Err);
 end;  { main }
begin
   writeln('Dimension der Matrix = ?'); read(n);  main(n);
end.
```

Das obige Programm ermöglicht die Inversion der schlecht-konditionierten Boothroyd/Dekker–Matrizen bis zur Dimension $n = 27$.

4.11.5 Inversion eines Matrixproduktes

4.11.5.1 Das benutzte Verfahren

Wir betrachten jetzt die Aufgabe, nicht eine einzelne Matrix $\boldsymbol{A}$ sondern das Matrixprodukt $\boldsymbol{AB} \in I\!R^{m\times m}$ zu invertieren, wobei $(\boldsymbol{AB})^{-1}$ durch die quadratische Intervallmatrix $\boldsymbol{X}$ maximalgenau einzuschließen ist:

$$(4.5) \quad (\boldsymbol{AB})^{-1} \in \boldsymbol{X} = ?, \qquad \boldsymbol{A} \in I\!R^{m\times n},\ \boldsymbol{B} \in I\!R^{n\times m},\ (\boldsymbol{AB})^{-1} \in I\!R^{m\times m}.$$

In einem ersten Lösungsversuch könnte man $\boldsymbol{X}$ dadurch erhalten, daß man zunächst das Produkt $\boldsymbol{AB}$ durch $\boldsymbol{P} := \#\#(\boldsymbol{A}\cdot\boldsymbol{B})$ maximalgenau einschließt und danach zu $\boldsymbol{P}$ die Einschließung $\boldsymbol{X}$ der Inversen $\boldsymbol{P}^{-1} \in \boldsymbol{X}$ berechnet. Da jedoch $\boldsymbol{P}$ schon als Intervallmatrix vorliegt, kann $\boldsymbol{X}$ die gesuchte Inverse $(\boldsymbol{A}B)^{-1}$ nicht maximalgenau einschließen; es kann sogar vorkommen, daß bei schlecht konditioniertem $\boldsymbol{A}$ oder $\boldsymbol{B}$ die Einschließung von $(\boldsymbol{AB})^{-1}$ nicht mehr gelingt.

Im Spezialfall, daß $\boldsymbol{A}, \boldsymbol{B}$ beide quadratisch sind, könnte man zunächst $(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$ benutzen, um mit $\boldsymbol{X}$ und $\boldsymbol{Y}$ Einschließungen von B^{-1} bzw. A^{-1} zu berechnen. Aber auch das Matrix–Produkt $\boldsymbol{XY}$ wäre dann keine maximalgenaue Einschließung von $(\boldsymbol{AB})^{-1}$, da $\boldsymbol{X}$ und $\boldsymbol{Y}$ schon als Intervallmatrizen vorliegen.

Um $(\boldsymbol{AB})^{-1}$ durch $\boldsymbol{X}$ wirklich maximalgenau einzuschließen, betrachten wir die $(m+n)\times(m+n)$ Matrixgleichung:

$$(4.6) \qquad \begin{pmatrix} \boldsymbol{B} & -\boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{A} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{O} \\ \boldsymbol{I} \end{pmatrix} \quad \Longleftrightarrow \quad \begin{pmatrix} \boldsymbol{BX} - \boldsymbol{Y} & = \boldsymbol{O} \\ \boldsymbol{AY} & = \boldsymbol{I} \end{pmatrix}$$

Multipliziert man jetzt $\boldsymbol{BX} - \boldsymbol{Y} = \boldsymbol{O}$ von links mit $\boldsymbol{A}$ und benutzt $\boldsymbol{AY} = \boldsymbol{I}$, so erhält man die Matrixgleichung

$$(4.7) \qquad (\boldsymbol{AB})\boldsymbol{X} = \boldsymbol{I} \quad \Longleftrightarrow \quad \boldsymbol{X} = (\boldsymbol{AB})^{-1}$$

Löst man daher die Gleichung in (4.6) mit der $(m+n)\times(m+n)$ Koeffizientenmatrix $\begin{pmatrix} \boldsymbol{B} & -\boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{A} \end{pmatrix}$ spaltenweise und schließt den Lösungsvektor $\begin{pmatrix} \boldsymbol{x}_j \\ \boldsymbol{y}_j \end{pmatrix}$ des jeweiligen linearen Gleichungssystems mit den bekannten selbstverifizierenden Methoden ein, so erhält man mit der oberen Intervallmatrix $\boldsymbol{X}$ die gesuchte Einschließung für $(\boldsymbol{AB})^{-1} \in \boldsymbol{X}$, wobei $\boldsymbol{X}$ jetzt wirklich **maximalgenau** berechnet werden kann.

4.11.5.2 Routinen zur Einschließung der Inversen

Gegeben seien die Matrizen $\boldsymbol{A}^{(m\times n)}$, $\boldsymbol{B}^{(n\times m)}$, so daß das Produkt $\boldsymbol{AB}$ als quadratische $(m \times m)$ Matrix berechnet werden kann. Die Prozedur

procedure `AB_inv` (**var** `A,B` : ■; `illcd : boolean`;
var `X` : ◇; **var** `Err : integer`);

liefert eine i.a. maximalgenaue Einschließungen $\boldsymbol{X}$ der Inversen $(\boldsymbol{AB})^{-1}$.

Die nachfolgende Tabelle enthält die erlaubten Datentypen bzgl. $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{X}$ und den Namen des Moduls, das die jeweilige Prozedur `AB_inv` exportiert:

Erlaubte Datentypen in Prozedur `AB_inv`				
$\boldsymbol{A}, \boldsymbol{B}$: ■	`(L)rmatrix`	`(L)imatrix`	`cmatrix`	`cimatrix`
$\boldsymbol{X}$: ◇	`(L)imatrix`		`cimatrix`	
exportiernd. Module	ab(l)inv	ab(l)iinv	abcinv	abciinv

Die Bedeutung von `illcd : boolean` findet man in nachfolgender Tabelle. Der Anwender sollte `AB_inv` zunächst mit `illcd=False` aufrufen, da mit `illcd=True` die zur Lösung linearer Gleichungssysteme notwendige Matrixinversion sehr zeitaufwendig in staggered Arithmetik durchgeführt wird.

Bedeutung des Parameters `illcd : boolean`
`illcd = False` Die Berechnung der $(m+n)\times(m+n)$ Inversen und der Verifikationsschritt werden in einfacher Länge durchgeführt. Falls die Verifikation nicht gelingt, erfolgen alle Rechnungen im doppeltlangen Datenformat.
`illcd = True` Nur die Berechnung der $(m+n)\times(m+n)$ Inversen zur Lösung des lin. Gleichungssystems erfolgt in staggered Arithmetik. Falls dann der nachfolgende Verifikationsschritt in einfacher Länge nicht erfolgreich ist, werden alle Rechnungen in doppelter Länge durchgeführt.

Bedeutung des Fehlerkodes `Err`	
`Err = 0`	Keine Fehler aufgetreten
`Err = 1`	Verifikationsschritt nicht möglich
`Err = 2`	Matrix-Inversion ist nicht möglich
`Err = 3`	Falsche Zeilenzahl der Matrix $\boldsymbol{X}$
`Err = 4`	Falsche Spaltenzahl der Matrix $\boldsymbol{X}$
`Err = 5`	Falsche Zeilenzahl der Matrix $\boldsymbol{B}$
`Err = 6`	Falsche Spaltenzahl der Matrix $\boldsymbol{B}$

4.11.5.3 Programmbeispiele

4.11.5.3.1 Inverse der Dekker-Produkt-Matrix

Das nachfolgende Programm `tstABinv1` berechnet die i.a. maximalgenaue Einschließung des inversen Produktes $(\boldsymbol{AA})^{-1}$, wobei $\boldsymbol{A}$ die quadratische Boothroyd/Dekker Matrix ist. Zum Vergleich werden Einschließungen der Konditionszahlen $\varkappa(\boldsymbol{A})$ und $\varkappa(\boldsymbol{AA})$ ausgegeben; es gilt die Definition:

$$\varkappa(\boldsymbol{A}) := \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^{-1}\|$$

```
program tstABinv1(input,output);
{ Berechnung einer Einschliessung der Produktinversen  }
{ (A*A)^(-1), wobei A eine Boothroyd/Dekker Matrix ist }
use type_def,     { exportiert benoetigte Datentypen   }
    li_ari   ,    { Intervall-Arithmetik               }
    ablinv   ,    { exportiert AB_inv(...)             }
    lmvi_ari,     { Intervall-Matrix/Vektor-Arithmetik }
    lmv_ari ,     { Matrix/Vektor-Arithmetik           }
    lmv_exap,     { exportiert Dekker, Dekker_INV      }
    lmvf0   ;     { exportiert Gesamtnorm NormG(...)   }
var n : integer;  { Dimension der Matrizen:     A,B,X  }
procedure main(n : integer) ;
var A,B   : Lrmatrix[1..n,1..n];
    X     : Limatrix[1..n,1..n];
    v     : Lrvector[1..n];
    Err   : integer;
    illcd : boolean;
begin
```

```
   writeln('illcd = False(0),True(1) = ?');  read(Err);
   illcd := Err=1;
   Dekker(A,v);       { Berechnung der Dekker-Matrizen }
   writeln('Die Konditionszahl der (',n:2,'x',n:2,')');
   writeln('Dekker-Matrix A ist enthalten in:');
   Dekker_INV(B);
   writeln( NormG(A)*NormG(B) );      { Konditionszahl }
   AB_inv(A,A,illcd,X,Err);
   if Err = 0 then
    begin
       writeln('(AA)^(-1)[',n:2,',',n:2,'] ist enthalten in:');
       writeln(X[n,n]);
       writeln('Die Konditionszahl der (',n:2,'x',n:2,')');
       writeln('Produkt-Matrix (A*A) ist enthalten in:');
       writeln( NormG(A*A)*NormG(Mid(X)) );
       writeln;
    end     else writeln('Err = ',Err);
end; { main }
begin
   repeat
      writeln('Dimension der Dekker-Matrix A = ?');  read(n);
      main(n);
   until false
end.
```

Die nachfolgende Bildschirmausgabe eines Programmablaufs bestätigt die Ungleichung $\varkappa(\boldsymbol{AA}) \leq \varkappa^2(\boldsymbol{A})$

```
Dimension der Dekker-Matrix A = ?
26       <enter>
illcd = False(0),True(1) = ?
0        <enter>
Die Konditionszahl der (26x26)
Dekker-Matrix A ist enthalten in:
[  5.26191321821796413215E+044,  5.26191321821796413216E+044]
(AA)^(-1)[26,26] ist enthalten in:
[  2.32288513431722896684E+032,  2.32288513431722896685E+032]
Die Konditionszahl der (26x26)
Produktmatrix (A*A) ist enthalten in:
[  2.01029991379164338513E+081,  2.01029991379164338515E+081]
```

4.11.5.3.2 Konditionszahl

Das nachfolgende Programm tstkond1 berechnet zu einer vorgegebenen Boothroyd/Dekker Matrix $\boldsymbol{A} \in I\!R^{n\times n}$ die Konditionszahl $\varkappa(\boldsymbol{M})$ der Matrix

$$\boldsymbol{M} := \begin{pmatrix} \boldsymbol{A} & -\boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{A} \end{pmatrix} \in I\!R^{2n\times 2n}, \qquad \varkappa(\boldsymbol{M}) = \|\boldsymbol{M}\| \cdot \|\boldsymbol{M}^{-1}\|$$

Die doppeltlange Approximation $\boldsymbol{R}_1 + \boldsymbol{R}_2 \approx \boldsymbol{M}^{-1}$ erfolgt mit Hilfe der Prozedur inv(M,R1,R2,factor,Err) von Seite 272.

```
program tstkond1(input,output);
{ Berechnung der Konditionszahl kappa(M) der Matrix }
{                   | A -I |                         }
{                   | O  A |                         }
{ wobei A eine Boothroyd/Dekker Matrix ist.          }
use li_ari   ,  { Intervall-Arithmetik               }
    linv     ,  { exportiert inv(A,R1,R2,factor,Err) }
    lmv_exap,  { exportiert Prozedur Dekker(A,v)    }
    lmvf0    ;  { exportiert Gesamtnorm NormG(...)   }

var n : integer;    { Dimension der Dekker Matrix A }
procedure main(n : integer) ;
var M,R1,R2         : Lrmatrix[1..2*n,1..2*n];
    A               : Lrmatrix[1..n  ,1..n  ];
    v               : Lrvector[1..n];
    Err,factor,i,j : integer;

begin
   Dekker(A,v);    { Berechnung der Dekker-Matrizen }
   M[1..n,1..n] := A;
   For i := 1 to n do
      for j := n+1 to 2*n do
         if j=n+i then M[i,j] := -1 else M[i,j] := 0;
   For i := n+1 to 2*n do
      for j := 1 to n do M[i,j] := 0.0;
   M[n+1..2*n,n+1..2*n] := A;

   inv(M,R1,R2,factor,Err);      { R1+R2 '=' M^(-1) }
   if Err = 0 then
   begin
```

```
      writeln('Die Konditionszahl der (',2*n:2,'x',2*n:2,')');
      writeln('Matrix M ist enthalten in:');
      writeln( NormG(M)*NormG(R1) );   writeln;
   end         else writeln('Err = ',Err);
end; { main }
begin
   repeat
      writeln('Dimension der Dekker-Matrix A = ?');  read(n);
      main(n);
   until false
end.
```

Für die Dimensionen $n = 26, 27$ der Boothroyd/Dekker Matrizen erhält man die folgenden Bildschirmausgaben:

```
Dimension der Dekker-Matrix A = ?
26       <enter>
Die Konditionszahl der (52x52)
Matrix M ist enthalten in:
[  4.1139807853635256573 5E+063,  4.11398078536352565736E+063]

Dimension der Dekker-Matrix A = ?
27       <enter>
Die Konditionszahl der (54x54)
Matrix M ist enthalten in:
[  1.51679368377473233573E+066,  1.51679368377473233575E+066]
```

Hinweis:

Bis $n = 26$ gelingt die Einschließung von $(\boldsymbol{AA})^{-1}$ bei Rechnung in doppelter Länge ohne staggered Arithmetik, vergl. Seite 284. Dabei wird jedoch $\boldsymbol{AA}$ nicht wirklich berechnet und anschließend invertiert, sondern man benutzt die Matrixgleichung (4.6) und invertiert die größere Matrix $\boldsymbol{M} := \begin{pmatrix} \boldsymbol{A} & -\boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{A} \end{pmatrix} \in I\!R^{2n\times 2n}$, deren Konditionszahl $\varkappa(\boldsymbol{M}) = 4.11 \cdot 10^{+63}$ jedoch um ganze 18 Größenordungen kleiner ist als $\varkappa(\boldsymbol{AA}) = 2.01 \cdot 10^{+81}$, d.h. der Umweg über $\boldsymbol{M}$ ist zwar wegen der doppelten Dimension aufwendiger, aber wegen der kleineren Konditionszahl erscheint die Inversion von $\boldsymbol{M}$ dennoch einfacher zu sein als die Inversion von $\boldsymbol{AA}$, zudem dieses Produkt durch eine Intervallmatrix einzuschließen wäre.

4.11.5.3.3 Nichtquadratische Faktoren

Wir nehmen jetzt $\boldsymbol{A} \in I\!R^{m\times n}$ und $\boldsymbol{B} \in I\!R^{n\times m}$ als nichtquadratisch an. Für das zu invertierende Produkt $\boldsymbol{AB} \in I\!R^{m\times m}$ gilt damit $\mathrm{Dim}(\boldsymbol{AB}) = m$ und $\mathrm{Rang}(\boldsymbol{AB}) \leq \min(m,n)$.

Betrachten wir zunächst den Fall $m > n$, so gilt

$$\mathrm{Rang}(\boldsymbol{AB}) \leq n < m = \mathrm{Dim}(\boldsymbol{AB}),$$

d.h. $\boldsymbol{AB}$ ist singulär und kann damit nicht invertiert werden. Die Einschließung von $(\boldsymbol{AB})^{-1}$ ist also nur möglich, wenn folgende Bedingung erfüllt ist:

$$\mathrm{Rang}(\boldsymbol{AB}) = m \leq n, \qquad \boldsymbol{A} \in I\!R^{m\times n},\ \boldsymbol{B} \in I\!R^{n\times m}; \tag{4.8}$$

Das nachfolgende Programm `tstABinv2` ermöglicht die Eingabe der rechteckigen Matrizen $\boldsymbol{A}$, $\boldsymbol{B}$ und berechnet die Einschließung von $(\boldsymbol{AB})^{-1}$.

```
program tstABinv2(input,output);
{ Berechnung der Einschliessung von (A*B)^(-1) }
{ A aus R^(mxn), B aus R^(nxm)                 }

use ablinv  ,  { exportiert AB_inv(A,B,illcd,X,Err) }
    lmvi_ari,  { Intervall-Matrix/Vektor-Arithmetik }
    lmv_ari ;  { Matrix/Vektor-Arithmetik           }

var m,n : integer;  { Dimension der Dekker Matrix A }
procedure main(m,n : integer) ;
var A     : Lrmatrix[1..m,1..n];
    B     : Lrmatrix[1..n,1..m];
    X     : Limatrix[1..m,1..m]; { X: Einschliessung }
    Err   : integer;      { Err: liefert Fehlerkode }
    illcd : boolean;
begin
   writeln('Eingabe von A zeilenweise,');
   writeln(m:2,' Zeilen zu je ',n:2,' Spalten:'); read(A);
   writeln('Eingabe von B zeilenweise,');
   writeln(n:2,' Zeilen zu je ',m:2,' Spalten:'); read(B);
   writeln('illcd = False(0),True(1) = ?'); read(Err);
   illcd := Err=1;
   AB_inv(A,B,illcd,X,Err); { X: Einschlg. v. (AB)^(-1) }
   if Err = 0 then
   begin
```

```
      writeln('Einschliessung von (AB)^(-1):');
      writeln(X);
      writeln;
   end        else writeln('Err = ',Err);
end; { main }
begin
   repeat
      writeln('Zeilenzahl  der Matrix A = ?');  read(m);
      writeln('Spaltenzahl der Matrix A = ?');  read(n);
      main(m,n);
   until false
end.
```

Ergebnisse:

- Im Falle $m > n$ erhält man stets `Err=2`, d.h. eine Matrixinversion ist nicht möglich, da (4.8) nicht erfüllt ist.

- Mit $\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix}$ und $\boldsymbol{B} = \begin{pmatrix} -2 & 3 \\ 2 & -7 \\ -1 & 5 \end{pmatrix}$ gilt zunächst:

  $\boldsymbol{AB} = \begin{pmatrix} 1 & 4 \\ -2 & 5 \end{pmatrix}$ und $(\boldsymbol{AB})^{-1} = \begin{pmatrix} \frac{5}{3} & -\frac{4}{3} \\ \frac{2}{3} & -\frac{1}{3} \end{pmatrix}$ und das Programm `tstABinv2` liefert nach der entsprechenden Eingabe von $\boldsymbol{A}$ und $\boldsymbol{B}$ mit `illcd=false` die maximalgenaue Einschließung $\boldsymbol{X}$ von $(\boldsymbol{AB})^{-1}$:

```
[  1.666666666666666666666E+000,  1.666666666666666666667E+000]
[ -1.333333333333333333334E+000, -1.333333333333333333333E+000]

[  6.666666666666666666666E-001,  6.666666666666666666667E-001]
[ -3.333333333333333333334E-001, -3.333333333333333333333E-001]
```

Anmerkung:

Natürlich könnte man im letzten Beispiel das berechnete Produkt $\boldsymbol{AB}$ auch direkt maximalgenau invertieren, da $\boldsymbol{AB}$ als Punktmatrix exakt darstellbar ist. Erst wenn $\boldsymbol{AB}$ durch eine Intervallmatrix eingeschlossen werden muß, wird der Einsatz der Prozedur `AB_inv(A,B,illcd,X,Err)` notwendig, falls eine maximalgenaue Einschließung von $(\boldsymbol{AB})^{-1}$ erforderlich ist.

4.12 Pseudoinverse

4.12.1 Definition

Wir betrachten die Matrix $\boldsymbol{A} \in I\!R^{m\times n}$ und verlangen:

$$\mathrm{Rang}(\boldsymbol{A}) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{falls } m \geq n \end{cases}$$

wobei Rang($\boldsymbol{A}$) die Maximalzahl liner unabhängiger Zeilen- oder Spaltenvektoren von $\boldsymbol{A}$ bedeutet. Unter dieser Voraussetzung ist die Moore/Penrose Pseudoinverse $\boldsymbol{A}^+$ wie folgt definiert:

$$\boldsymbol{A}^+ = \begin{cases} (\boldsymbol{A}^H\boldsymbol{A})^{-1}\boldsymbol{A}^H & \text{falls } m \geq n \\ \boldsymbol{A}^H(\boldsymbol{A}\boldsymbol{A}^H)^{-1} & \text{falls } n \geq m \end{cases} \qquad \text{mit:} \quad \boldsymbol{A}^+ \in I\!R^{n\times m} \tag{4.9}$$

Bildet man zu $\boldsymbol{A}$ die Transponierte und dazu die konjugiert komplexe Matrix, so erhält man nach Definition $\boldsymbol{A}^H$, und mit der Pseudoinversen $\boldsymbol{A}^+$ gelten die Beziehungen:

$$\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{I} \in I\!R^{n\times n}, \quad \text{falls } m \geq n \tag{4.10}$$

$$\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{I} \in I\!R^{m\times m}, \quad \text{falls } n \geq m \tag{4.11}$$

4.12.2 Algorithmen zur Berechnung der Pseudoinversen

Wir betrachten zunächst den Fall: $m > n$

Dann ist also $\boldsymbol{X} = \boldsymbol{A}^+$ nach (4.9) die eindeutige Lösung der Matrixgleichung

$$(\boldsymbol{A}^H\boldsymbol{A})\boldsymbol{X} = \boldsymbol{A}^H, \tag{4.12}$$

die wir jedoch nicht direkt lösen werden, da $\boldsymbol{A}^H\boldsymbol{A}$ zum einen sehr schlecht konditioniert ist, und weil die Produktmatrix auf dem Rechner durch eine Intervall–Matrix einzuschließen ist. Zur Berechnung von $\boldsymbol{X}$ betrachten wir vielmehr mit $\boldsymbol{I} \in I\!R^{m\times m}$ die folgende Matrixgleichung:

$$\begin{pmatrix} \boldsymbol{A} & -\boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{A}^H \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{O} \end{pmatrix} \iff \begin{pmatrix} \boldsymbol{A}\boldsymbol{X} - \boldsymbol{Y} & = \boldsymbol{I} \\ \boldsymbol{A}^H\boldsymbol{Y} & = \boldsymbol{O} \end{pmatrix} \tag{4.13}$$

Multipliziert man $\boldsymbol{A}\boldsymbol{X} - \boldsymbol{Y} = \boldsymbol{I}$ von links mit $\boldsymbol{A}^H$ und benutzt $\boldsymbol{A}^H\boldsymbol{Y} = \boldsymbol{O}$, so erhält man die Matrixgleichung (4.12), d.h. löst man (4.13) spaltenweise mit den Einschließungsmethoden für lineare Gleichungssysteme, so erhält man für die Pseudoinverse $\boldsymbol{X} = \boldsymbol{A}^+$ i.a. eine maximalgenaue Einschließung.

Beachten Sie bitte, daß in der quadratischen $(n+m) \times (n+m)$ Matrix $\begin{pmatrix} \boldsymbol{A} & -\boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{A}^H \end{pmatrix}$ aus (4.13) keine Produkte enthalten sind, während im Gegensatz dazu das Matrixprodukt $(\boldsymbol{A}^H\boldsymbol{A})$ in Gleichung (4.12) eine maximalgenaue Einschließung der Pseudoinversen gerade verhindert.

Wir betrachten jetzt den Fall: $n > m$

Dann gilt nach (4.9) für $\boldsymbol{X} = \boldsymbol{A}^+$ die Beziehung $\boldsymbol{X}(\boldsymbol{A}\boldsymbol{A}^H) = \boldsymbol{A}^H$. Wendet man hierauf den Hermite–Operator H an, so erhält man die Matrixgleichung:

$$(\boldsymbol{A}\boldsymbol{A}^H) \cdot \boldsymbol{X}^H = \boldsymbol{A}, \tag{4.14}$$

die wir aus den bekannten Gründen wieder nicht direkt lösen. Betrachtet man jedoch die Matrixgleichung:

$$\begin{pmatrix} \boldsymbol{A}^H & -\boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{A} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{X}^H \\ \boldsymbol{Y}^H \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{O} \end{pmatrix} \quad \Longleftrightarrow \quad \begin{pmatrix} \boldsymbol{A}^H\boldsymbol{X}^H - \boldsymbol{Y}^H & = \boldsymbol{I} \\ \boldsymbol{A}\boldsymbol{Y}^H & = \boldsymbol{O} \end{pmatrix} \tag{4.15}$$

und multipliziert man $\boldsymbol{A}^H\boldsymbol{X}^H - \boldsymbol{Y}^H = \boldsymbol{I}$ von links mit $\boldsymbol{A}$, so erhält man genau die Matrixgleichung (4.14), d.h. löst man (4.15) spaltenweise mit den Einschließungsmethoden für lineare Gleichungssysteme, so erhält man für die Pseudoinverse $\boldsymbol{X} = \boldsymbol{A}^+$ i.a. eine maximalgenaue Einschließung, wenn man auf die Einschließung $[\boldsymbol{X}^H]$ des Lösungsblocks $\boldsymbol{X}^H$ abschließend noch den Hermite–Operator H anwendet.

Abschließend noch einige Bemerkungen zum Fall $m = n$. Zunächst gilt nach (4.9) die Beziehung $\boldsymbol{A}^+ = \boldsymbol{A}^{-1}$ und

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I} \tag{4.16}$$

Eine Einschließung der Inversen $\boldsymbol{A}^{-1}$ kann man daher nach Gleichung (4.16) auch dadurch gewinnen, daß man die Matrixgleichung (4.16) spaltenweise mit den Einschließungsmethoden für lineare Gleichungssysteme löst. Da diese Vorgehensweise jedoch zeitaufwendiger ist als die Methoden aus Abschnitt 4.11.2 auf Seite 262, sollten Einschließung der Inversen $\boldsymbol{A}^{-1}$ nur mit Hilfe dieser letzteren Verfahren berechnet werden. Die Prozedur `Ps_inv` ermöglicht jedoch, im Falle $m = n$ die Einschließung der Inversen nach (4.16) auch durch das Lösen von n linearen Gleichungssystemen unter größerem Zeitaufwand zu berechnen.

4.12.3 Routinen zur Berechnung der Pseudoinversen

Gegeben sei eine nicht notwendig quadratische Matrix

$$A \in I\!R^{m \times n} \tag{4.17}$$

Die Prozedur

```
procedure Ps_inv ( var A : ■; illcd : boolean;
                   var X : ◇; var Err : integer );
```

liefert eine i.a. maximalgenaue Einschließungen $\boldsymbol{X}$ der Pseudoinversen $\boldsymbol{A}^+$, wobei die Bedeutung von $\boldsymbol{A}^+$ bzgl. der drei Fälle $m = n$, $m > n$ und $n > m$ zu unterscheiden ist:

Bedeutung der Pseudoinversen A^+ im Falle Err=0		
$m = n$	$m > n$	$n > m$
$\boldsymbol{A}^+ = \boldsymbol{A}^{-1} \in \boldsymbol{X}$	$\boldsymbol{A}^+ = (\boldsymbol{A}^H\boldsymbol{A})^{-1}\boldsymbol{A}^H \in \boldsymbol{X}$	$\boldsymbol{A}^+ = \boldsymbol{A}^H(\boldsymbol{A}\boldsymbol{A}^H)^{-1} \in \boldsymbol{X}$

Die Bedeutung von `illcd : boolean` findet man in nachfolgender Tabelle. Der Anwender sollte `Ps_inv` zunächst mit `illcd=False` aufrufen, da mit `illcd=True` die zur Lösung linearer Gleichungssysteme notwendige Matrixinversion sehr zeitaufwendig in staggered Arithmetik durchgeführt wird.

Bedeutung des Parameters `illcd : boolean`
`illcd = False` Die Berechnung der $(n + m) \times (n + m)$ Inversen und der Verifikationsschritt werden in einfacher Länge durchgeführt. Falls die Verifikation nicht gelingt, erfolgen alle Rechnungen im doppeltlangen Datenformat.
`illcd = True` Nur die Berechnung der $(n + m) \times (n + m)$ Inversen zur Lösung des lin. Gleichungssystems erfolgt in staggered Arithmetik. Falls dann der nachfolgende Verifikationsschritt in einfacher Länge nicht erfolgreich ist, werden alle Rechnungen in doppelter Länge durchgeführt.

Bedeutung des Fehlerkodes `Err`	
`Err = 0`	Keine Fehler aufgetreten
`Err = 1`	Verifikationsschritt nicht möglich
`Err = 2`	Matrix-Inversion ist nicht möglich
`Err = 3`	Falsche Zeilenzahl der Matrix $\boldsymbol{X}$
`Err = 4`	Falsche Spaltenzahl der Matrix $\boldsymbol{X}$

Die nachfolgende Tabelle enthält die erlaubten Datentypen bzgl. $\boldsymbol{A}, \boldsymbol{X}$ und den Namen des Moduls, das die jeweilige Prozedur `Ps_inv` exportiert.

Erlaubte Datentypen in Prozedur `Ps_inv`				
$\boldsymbol{A} : \blacksquare$	`(L)rmatrix`	`(L)imatrix`	`cmatrix`	`cimatrix`
$\boldsymbol{X} : \diamond$	`(L)imatrix`		`cimatrix`	
exportier. Module	(l)lss	(l)ilss	clss	cilss

4.12.4 Programmbeispiel

Das folgende Programm `tstPsinv` ermöglicht die Eingabe einer Rechteck–Matrix $\boldsymbol{A} \in I\!R^{m\times n}$ und die Berechnung einer maximalgenauen Einschließung $\boldsymbol{X}$ seiner Pseudo–Inversen $\boldsymbol{A}^+ \in \boldsymbol{X}$, die nach (4.9) definiert ist. Nach (4.10),(4.11) muß dann die Einheitsmatrix $\boldsymbol{I}$ in $\boldsymbol{XA}$, $(m \geq n)$ bzw. in $\boldsymbol{AX}$, $(n \geq m)$ enthalten sein, was vom Programm durch eine entsprechende Meldung bestätigt wird.

```
program tstPsinv1(input,output);
{ Berechnung der Einschliessung einer Pseudo-Inversen }

use lss    ,  { exportiert Ps_inv(A,illcd,X,Err)   }
    mvi_ari,  { Intervall-Matrix/Vektor-Arithmetik }
    mv_ari ;  { Matrix/Vektor-Arithmetik           }

var m,n : integer; { Dimension der Dekker Matrix A }
procedure main(m,n : integer) ;
var A     : rmatrix[1..m,1..n];
    X     : imatrix[1..n,1..m]; { X: Einschliessung }
    Err   : integer;     { Err: liefert Fehlerkode }
    illcd : boolean;

begin
   writeln('Eingabe von A zeilenweise,');
   writeln(m:2,' Zeilen zu je ',n:2,' Spalten:'); read(A);
   writeln('illcd = False(0),True(1) = ?'); read(Err);
   illcd := Err=1;
   Ps_inv(A,illcd,X,Err); { X: Einschlg. d. Pseudo-Inv. }
   if Err = 0 then
   begin
      writeln('Einschliessung X der Pseudo-Inversen:');
      writeln(X);
      if m>n then
         if id(n) in X*A then writeln('I in X*A enthalten')
         else writeln('trouble')  else
         if id(m) in A*X then writeln('I in A*X enthalten')
         else writeln('trouble');
      writeln;
   end        else writeln('Err = ',Err);
end; { main }
```

```
begin
   repeat
      writeln('Zeilen-Zahl der Matrix A = ?');  read(m);
      writeln('Spaltenzahl der Matrix A = ?');  read(n);
      main(m,n);
   until false
end.
```

Mit der Matrix $\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix}$ ergibt sich die folgende Bildschirmausgabe:

```
Zeilen-Zahl der Matrix A = ?
2           <enter>
Spaltenzahl der Matrix A = ?
3           <enter>
Eingabe von A zeilenweise,
 2 Zeilen zu je 3 Spalten:
1 2 3       <enter>
2 3 4       <enter>
illcd = False(0),True(1) = ?
0           <enter>
Einschliessung X der Pseudo-Inversen:
[ -1.833333333334E+000, -1.833333333333E+000]
[  1.333333333333E+000,  1.333333333334E+000]

[ -3.333333333334E-001, -3.333333333333E-001]
[  3.333333333333E-001,  3.333333333334E-001]

[  1.166666666666E+000,  1.166666666667E+000]
[ -6.666666666667E-001, -6.666666666666E-001]

I in A*X enthalten
```

Die exakte Pseudo–Inverse ist gegeben durch $\boldsymbol{A}^+ := \begin{pmatrix} -\frac{11}{6} & \frac{4}{3} \\ -\frac{1}{3} & \frac{1}{3} \\ \frac{7}{6} & -\frac{2}{3} \end{pmatrix}$, da $\boldsymbol{AA}^+ = \boldsymbol{I}$. Damit ist dann $\boldsymbol{X}$ tatsächlich die maximalgenaue Einschließung der Pseudo–Inversen $\boldsymbol{A}^+ \in \boldsymbol{X}$. Beachten Sie bitte, daß in unserem Fall $(n > m)$ nach (4.10) die Gleichung $\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{I}$ nicht erfüllt ist.

Kapitel 5

Lineare Gleichungssysteme

In der numerischen Praxis ist die Lösung eines linearen Gleichungssystems

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

eine oft gestellte Aufgabe, die auch bei kleineren Dimensionen nicht einfach zu bewältigen ist, wenn die Koeffizientenmatrix $\boldsymbol{A}$ schlecht konditioniert ist. In diesem Fall können die bekannten Rundungsfehler bei den Gleitkommarechnungen so starke Auslöschungen verursachen, daß die exakte Lösung $\boldsymbol{x}$ durch das Maschinenergebnis $\widetilde{\boldsymbol{x}}$ nur noch sehr schlecht approximiert wird. Dabei zeigt sich, daß auch Rechnungen im doppeltlangen Format diese Auslöschungen nicht immer beseitigen. Wenn man jedoch nach Rump [10, Lineare Probleme] mit selbstverifizierenden Algorithmen eine garantierte Einschließung $[\boldsymbol{x}]$ des Lösungsvektors $\boldsymbol{x}$ berechnet, so liefern erst die Intervalldurchmesser der einzelnen Komponenten dieser Einschließung eine sichere Basis zur Beurteilung der Güte der erhaltenen Maschinenlösung.

Im ersten Abschnitt betrachten wir dicht besetzte Matrizen $\boldsymbol{A}$, die auch nichtquadratisch sein können. Der interessierte Anwender findet dort eine Zusammenfassung des benutzten Algorithmus, der sich sowohl auf Intervallmatrizen als auch auf komplexwertige Matrizen erweitern läßt.

Im darauffolgenden Abschnitt werden Bandmatrizen behandelt, die in der Praxis ebenfalls eine große Rolle spielen. Die Lösungsvektoren der zugehörigen Gleichungssysteme lassen sich dann auch noch für größere Dimensionen erfolgreich einschließen.

5.1 Dicht besetzte Matrizen

Wir betrachten das lineare Gleichungssystem

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \qquad \boldsymbol{A} \in I\!R^{n\times n} \text{ und } \boldsymbol{x}, \boldsymbol{b} \in I\!R^n,$$

wobei vorausgesetzt wird, daß seine Koeffizientenmatrix $\boldsymbol{A}$ dicht besetzt ist.

5.1.1 Einschließung des Lösungsvektors

Die Einschließung $[\boldsymbol{x}]$ des Lösungsvektors $\boldsymbol{x} \in I\!R^n$ erfolgt mit folgendem Algorithmus, [10, Lineare Probleme], [5, Chapter 10], [8, Chapter 2]:

1. Approximation der Inversen: $\boldsymbol{R} \approx \boldsymbol{A}^{-1}$
2. Berechnung einer ersten Näherung: $\widetilde{\boldsymbol{x}} := \boldsymbol{R}\boldsymbol{b}$
   - Der Ansatz $\boldsymbol{x} = \widetilde{\boldsymbol{x}} + \boldsymbol{y}$ führt mit dem Residuum $\boldsymbol{d} := \boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}}$ auf das lineare Gleichungssystem: $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{d}$ und eine Einschließung $[\boldsymbol{y}]$ der Lösung $\boldsymbol{y}$ erhält man durch die beiden folgenden Punkte:
3. $[\boldsymbol{y}]^{(\nu+1)} := [\boldsymbol{R}(\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}})] \mathbin{\diamond\!\!\!\!+} [\boldsymbol{I} - \boldsymbol{R}\boldsymbol{A}] \diamond [\boldsymbol{y}]^{(\nu)}, \qquad \nu = 0, 1, 2, \ldots$
4. Gilt für $\nu = k$: $[\boldsymbol{y}]^{(k+1)} \overset{\circ}{\subset} [\boldsymbol{y}]^{(k)}$, so folgt $\boldsymbol{y} \in [\boldsymbol{y}]^{(k+1)} =: [\boldsymbol{y}]$.
5. Die Lösung $\boldsymbol{x} \in \widetilde{\boldsymbol{x}} \mathbin{\diamond\!\!\!\!+} [\boldsymbol{y}]^{(k+1)} =: [\boldsymbol{x}]$ ist dann eindeutig.

Anmerkungen:

- Die Operatoren $\mathbin{\diamond\!\!\!\!+}$, $\diamond$ sind Intervalloperatoren, die in der **PASCAL-XSC** Umgebung zur Verfügung stehen.
- Mit $[\boldsymbol{I} - \boldsymbol{R}\boldsymbol{A}] := \#\#(\boldsymbol{I} - \boldsymbol{R}\cdot\boldsymbol{A})$ kann der Defekt $\boldsymbol{I} - \boldsymbol{R}\boldsymbol{A}$ maximalgenau eingeschlossen werden.
- Die Einschließung $[\boldsymbol{R}(\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}})]$ von $\boldsymbol{R}(\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}})$ wird wie folgt berechnet:
  1. $\widetilde{\boldsymbol{d}} := \# * (\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}})$; $[\boldsymbol{z}] := \#\#(\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}} - \widetilde{\boldsymbol{d}})$; $\rightsquigarrow$ $\boldsymbol{d} \in \widetilde{\boldsymbol{d}} + [\boldsymbol{z}]$;
  2. $[\boldsymbol{R}(\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}})] := \#\#(\boldsymbol{R}\cdot\widetilde{\boldsymbol{d}} + \boldsymbol{R}\cdot[\boldsymbol{z}])$; $\rightsquigarrow$ $\boldsymbol{R}(\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}}) \in [\boldsymbol{R}(\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}})]$;
- $[\boldsymbol{y}]^{(0)} := [\boldsymbol{R}(\boldsymbol{b} - \boldsymbol{A}\widetilde{\boldsymbol{x}})]$ ist der Startvektor in 3.
- $\overset{\circ}{\subset}$ verlangt, daß $[\boldsymbol{y}]^{(k+1)}$ ganz im Innern von $[\boldsymbol{y}]^{(k)}$ enthalten ist und wird in der **PASCAL-XSC** Umgebung durch den Operator **in** realisiert. Damit diese Einschließung gelingt, muß im Punkt 3. vor jedem Iterationsschritt der Vektor $[\boldsymbol{y}]^{(\nu)}$ mit Hilfe der `blow`–Funktion leicht aufgebläht werden [10, Seite 125].

5.1.2 Nichtquadratische Koeffizientenmatrix

Wir betrachten das lineare Gleichungssystem

$$(5.1) \qquad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \qquad \boldsymbol{A} \in I\!R^{m\times n} \ \text{ und } \ \boldsymbol{x} \in I\!R^{n}, \ \boldsymbol{b} \in I\!R^{m}$$

und setzen voraus:

$$\text{Rang}(\boldsymbol{A}) = \begin{cases} n & \text{falls } \ m > n \ \text{ bei einem überbestimmten System} \\ m & \text{falls } \ n > m \ \text{ bei einem unterbestimmten System} \end{cases}$$

wobei Rang($\boldsymbol{A}$) die Maximalzahl linear unabhängiger Zeilen- oder Spaltenvektoren ist.

5.1.2.1 Überbestimmte Systeme

Bei einem überbestimmten System ($m > n$) gibt es mehr Gleichungen als Unbekannte, so daß (5.1) i.a. keine Lösung $\boldsymbol{x}$ besitzt, d.h. $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \neq 0$. Wir suchen daher ein $\boldsymbol{x} \in I\!R^{n}$, so daß gilt:

$$(5.2) \qquad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 = \min_{\boldsymbol{x}\in I\!R^n}, \qquad \|\boldsymbol{r}\|^2 := \sum_i |\boldsymbol{r}_i|^2;$$

Die im Falle Rang($\boldsymbol{A}$) $= n$ eindeutig bestimmte Lösung $\boldsymbol{x}$ ist Lösung von

$$(5.3) \qquad \boldsymbol{A}^H\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^H\boldsymbol{b},$$

wobei $\boldsymbol{A}^H$ die hermitesche Matrix ist, welche im reellen Fall durch die Transponierte von $\boldsymbol{A}$ gegeben ist. Da $\boldsymbol{A}^H\boldsymbol{A}$ i.a. schlecht konditioniert ist und auf der Maschine nicht exakt berechnet werden kann, löst man nicht (5.3), sondern mit $\boldsymbol{y} \in I\!R^{m}$ das quadratische $(n+m)\,\text{x}\,(m+n)$ System

$$(5.4) \qquad \begin{pmatrix} \boldsymbol{A} & -\boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{A}^H \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{pmatrix} \quad \Longleftrightarrow \quad \begin{pmatrix} \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} & = \boldsymbol{b} \\ \boldsymbol{A}^H\boldsymbol{y} & = \boldsymbol{0} \end{pmatrix}$$

woraus sich das Gleichungssystem (5.3) unmittelbar ergibt, wenn man die erste Gleichung $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} = \boldsymbol{b}$ von links mit $\boldsymbol{A}^H$ multipliziert. Die gesuchte Lösung $\boldsymbol{x}$ von (5.3) ist damit automatisch der Vektor $\boldsymbol{x}$ in (5.4), wobei der Lösungsvektor dieses größeren linearen Glg.-Systems mit dem Algorithmus aus Abschnitt 5.1.1 eingeschlossen werden kann.

5.1.2.2 Unterbestimmte Systeme

Bei einem unterbestimmten System

$$(5.5) \qquad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \qquad \boldsymbol{A} \in I\!R^{m\times n},\ \boldsymbol{x} \in I\!R^n,\ \boldsymbol{b} \in I\!R^m, \quad n > m$$

gibt es wegen $n > m$ mehr Unbekannte als Gleichungen, so daß (5.5) i.a. unendlich viele Lösungen $\boldsymbol{x}$ besitzt. Ist dann $\mathbb{L} = \{\boldsymbol{x} \,|\, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ die Menge aller Lösungsvektoren $\boldsymbol{x} \in I\!R^n$, so suchen wir jetzt denjenigen Vektor $\boldsymbol{y} \in \mathbb{L}$, für den $\|\boldsymbol{y}\|^2$ minimal wird:

$$(5.6) \qquad \boldsymbol{y} := \{\boldsymbol{x} \,|\, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \ \wedge\ \|\boldsymbol{x}\|^2 = \min\}, \quad \boldsymbol{x}, \boldsymbol{y} \in I\!R^n, \ \ \boldsymbol{b} \in I\!R^m;$$

Unter der Voraussetzung $\text{Rang}(\boldsymbol{A}) = m$ ist $\boldsymbol{y} \in I\!R^n$ eindeutig bestimmt und berechnet sich aus

$$(5.7) \qquad \boldsymbol{y} = \boldsymbol{A}^H\boldsymbol{u}, \quad \text{mit} \quad \boldsymbol{A}\boldsymbol{A}^H\boldsymbol{u} = \boldsymbol{b}, \ \ \boldsymbol{u} \in I\!R^m;$$

Auch jetzt lösen wir nicht das lineare System $\boldsymbol{A}\boldsymbol{A}^H\boldsymbol{u} = \boldsymbol{b}$, da $\boldsymbol{A}\boldsymbol{A}^H$ schlecht konditioniert ist und die Multiplikation nicht exakt ausgewertet werden kann. Wir betrachten vielmehr das größere quadratische $(n+m) \times (n+m)$ System

$$(5.8) \qquad \begin{pmatrix} \boldsymbol{A}^H & -\boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{A} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{b} \end{pmatrix} \quad \Longleftrightarrow \quad \begin{pmatrix} \boldsymbol{A}^H\boldsymbol{u} - \boldsymbol{y} & = \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{y} & = \boldsymbol{b} \end{pmatrix}$$

aus dem durch Einsetzen von $\boldsymbol{y} = \boldsymbol{A}^H\boldsymbol{u}$ in die zweite Gleichung $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$ das lineare System $\boldsymbol{A}\boldsymbol{A}^H\boldsymbol{u} = \boldsymbol{b}$ in (5.7) direkt folgt. Die gesuchte Lösung $\boldsymbol{y}$ von (5.7) ist damit automatisch der Vektor $\boldsymbol{y}$ in (5.8), wobei der Lösungsvektor dieses größeren linearen Glg.-Systems mit dem Algorithmus aus Abschnitt 5.1.1 eingeschlossen werden kann.

5.1.3 Routinen zur Einschließung des Lösungsvektors

Gegeben sei das lineare Gleichungssystem

$$(5.9) \qquad \boldsymbol{Ax} = \boldsymbol{b}, \qquad \boldsymbol{A} \in I\!R^{m \times n},\ \boldsymbol{x} \in I\!R^{n},\ \boldsymbol{b} \in I\!R^{m};$$

Die Prozedur

procedure `lss` (**var** `A` : ■; `illcd : boolean`; **var** `b` : □; **var** `z` : ◇; **var** `Err : integer`);

liefert Einschließungen $\boldsymbol{z}$ des Lösungsvektors eines linearen Systems, wobei die Bedeutung von $\boldsymbol{z}$ bzgl. der drei Fälle $m = n$, $m > n$ und $n > m$ zu unterscheiden ist:

Bedeutung des Intervallvektors $\boldsymbol{z}$ im Falle Err=0		
$m = n$ $\boldsymbol{x} := \boldsymbol{A}^{-1}\boldsymbol{b} \in \boldsymbol{z}$	$m > n$ $\boldsymbol{x} := (\boldsymbol{A}^H\boldsymbol{A})^{-1}\boldsymbol{A}^H \cdot \boldsymbol{b} \in \boldsymbol{z}$	$n > m$ $\boldsymbol{y} := \boldsymbol{A}^H(\boldsymbol{AA}^H)^{-1} \cdot \boldsymbol{b} \in \boldsymbol{z}$
$\boldsymbol{x} \in I\!R^n$ ist Lösung v. (5.9)	Bedeutung von $\boldsymbol{x} \in I\!R^n$ vergleiche (5.2)	Bedeutung von $\boldsymbol{y} \in I\!R^n$ vergleiche (5.6)

Die Bedeutung des Parameters `illcd : boolean` findet man in nachfolgender Tabelle. Der Anwender sollte die Prozedur `lss` zunächst mit `illcd=False` aufrufen, da mit `illcd=True` die Matrixinversion sehr zeitaufwendig in staggered Arithmetik durchgeführt wird.

Bedeutung des Parameters `illcd : boolean`
`illcd = False` Die Berechnung der Inversen und der Verifikationsschritt werden in einfacher Länge durchgeführt. Falls die Verifikation nicht gelingt, erfolgen alle Rechnungen im doppeltlangen Datenformat.
`illcd = True` Nur die Berechnung der Inversen erfolgt jetzt in staggered Arithmetik. Falls dann der Verifikationsschritt in einfacher Länge nicht erfolgreich ist, werden alle Rechnungen in doppelter Länge durchgeführt.

Bedeutung des Fehlerkodes `Err`	
`Err = 0`	Keine Fehler aufgetreten
`Err = 1`	Verifikationsschritt nicht möglich
`Err = 2`	Matrix-Inversion ist nicht möglich
`Err = 3`	Falsche Dimension des Vektors $\boldsymbol{b}$
`Err = 4`	Falsche Dimension des Vektors $\boldsymbol{z}$

Die nachfolgende Tabelle enthält die erlaubten Datentypen bzgl. $\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{z}$ und den Namen des Moduls, das die jeweilige Prozedur `lss` exportiert.

Erlaubte Datentypen in Prozedur `lss`				
$\boldsymbol{A} : \blacksquare$	`(L)rmatrix`	`(L)imatrix`	`cmatrix`	`cimatrix`
$\boldsymbol{b} : \square$	`(L)rvector`	`(L)ivector`	`cvector`	`civector`
$\boldsymbol{z} : \diamond$	`(L)ivector`		`civector`	
exportier. Module	(l)lss	(l)ilss	clss	cilss

5.1.4 Programmbeispiele

5.1.4.1 Boothroyd/Dekker–Matrizen

Wir betrachten das lineare System $\boldsymbol{Ax} = \boldsymbol{b}$, wobei die Koeffizientenmatrix $\boldsymbol{A}$ als Boothroyd/Dekker Matrix und der Vektor $\boldsymbol{b}$ durch $\boldsymbol{b}[i] = i$ gegeben sind. Das nachfolgende Programm `tstllss1` berechnet mit `z` eine maximalgenau Einschließung des exakten Lösungsvektors $\boldsymbol{z}$, mit $\boldsymbol{z}[i] = (-1)^i \cdot (i-1)$.

```
program tstllss1(input,output);
{ Einschliessung z des Loesungsvektors x des lin.  }
{ Systems:       Ax = b;         x enthalten in z; }
use llss    , { Exportiert lss(A,illcd,b,z,Err)    }
    lmv_exap, { Exportiert Dekker(A,b)             }
    lmvi_ari; { Intervall-Matrix/Vektor Arithmetik }
var n            : integer;
procedure main(n : integer);
var A     : Lrmatrix[1..n,1..n];
    b     : Lrvector[1..n];
    z     : Livector[1..n];
    illcd : boolean;
    Err   : integer;
begin
   Dekker(A,b); { A: Dekker-Matrix, b[i] := i, i = 1..n; }
   writeln('illcd True(1),False(0) = ?');  read(Err);
   illcd := Err=1;
   lss(A,illcd,b,z,Err); { z: Einschliessung der Loesung }
   if Err = 0 then
   writeln(z) else writeln('Err = ',Err);
end; { main }

BEGIN
   writeln('Matrixdimension = ?');  read(n);
   main(n);
END.
```

Das Programm liefert bis zur Dimension $n = 27$ ohne staggered Arithmetik (`illcd=0`) die maximalgenaue Einschließung $\boldsymbol{z}$ des Lösungsvektors $\boldsymbol{x}$. Die Definition der Boothroyd/Dekker–Matrizen findet man im Abschnitt 4.1.2 auf Seite 164.

Bildschirmausgabe des Programms `tstllss1`:

```
Matrixdimension = ?
27          <enter>
illcd True(1),False(0) = ?
0           <enter>

[  0.00000000000000000000E+000,  0.00000000000000000000E+000]
[  1.00000000000000000000E+000,  1.00000000000000000000E+000]
[ -2.00000000000000000000E+000, -2.00000000000000000000E+000]
                                *
                                *
[  2.50000000000000000000E+001,  2.50000000000000000000E+001]
[ -2.60000000000000000000E+001, -2.60000000000000000000E+001]
```

5.1.4.2 Boothroyd/Dekker–Blockmatrizen

Mit den bereits in Abschnitt 4.1.2 auf Seite 164 definierten BoothroydDekker Matrizen $\boldsymbol{D} \in I\!R^{n\times n}$ bilden wir die Blockmatrix $\boldsymbol{A} := \begin{pmatrix} \boldsymbol{D} & -\boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{D} \end{pmatrix} \in I\!R^{2n\times 2n}$ und betrachten das lineare Gleichungssystem

$$(5.10) \qquad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \qquad \boldsymbol{I}:\ \text{Einheitsmatrix}, \quad \boldsymbol{b}[i] := i, \quad i = 1, 2, \ldots, 2n;$$

Bedeutet $\widetilde{\boldsymbol{x}} \approx \boldsymbol{x}$ eine Näherung des Lösungsvektors $\boldsymbol{x}$ und sind $\|\boldsymbol{x}\|$ und $\|\boldsymbol{A}\|$ verträgliche Vektor- und Matrix–Normen, so gilt mit dem Defekt $\boldsymbol{r} := \boldsymbol{A}\widetilde{\boldsymbol{x}} - \boldsymbol{b}$ nach [19] für den relativen Fehler die Abschätzung:

$$\frac{\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \le \varkappa(\boldsymbol{A}) \cdot \frac{\|\boldsymbol{r}\|}{\|\boldsymbol{b}\|}, \qquad \varkappa(\boldsymbol{A}) := \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^{-1}\|$$

Nur bei einer kleinen Konditionszahl $\varkappa(\boldsymbol{A})$ kann man daher aus einem relativ kleinen Defektvektor $\boldsymbol{r}$ auf einen kleinen relativen Fehler des Lösungsvektors $\boldsymbol{x}$ schließen.

Das nachfolgende Programm `tstllss2` liefert bis zur Dimension $n = 26$ der Dekker–Matrix $\boldsymbol{D}$ ohne Anwendung der staggered Arithmetik (`illcd=0`) eine Einschließung der Lösung $\boldsymbol{x}$ aus (5.10). Erst mit der Dimension $n = 27$ und `illcd=0` scheitert wegen `Err=1` der Verifikationsschritt in der Prozedur lss(A,illcd,b,z,Err).

```
program tstllss2(input,output);
{ Berechnung einer Einschliessung der Loesung x eines   }
{ lin. Systems  Ax = b  mit der Koeffizientenmatrix A = }
{                                                       }
{                     | D  -I |                         }
{                     | 0   D |                         }
{                                                       }
{ wobei  D  eine Boothroyd/Dekker Matrix ist.           }
use llss    ,      { exportiert  lss(M,illcd,b,z,Err)   }
    lmvi_ari,      { Intervall-Matrix/Vektor-Arithmetik }
    lmv_exap;      { exportiert Prozedur  Dekker(D,v)   }
var n : integer;   { Dimension der Dekker Matrix  D     }
procedure main(n : integer) ;
var A       : Lrmatrix[1..2*n,1..2*n];
    D       : Lrmatrix[1..n,1..n];
    z       : Livector[1..2*n];
    v       : Lrvector[1..n];
    b       : Lrvector[1..2*n];
    Err,i,j : integer;
    illcd   : boolean;
begin
   writeln('illcd = False(0),True(1) = ?');  read(Err);
   illcd := Err=1;
   Dekker(D,v);       { Berechnung der Dekker-Matrizen }
   { Aufbau der Blockmatrix A: }
   for j := 1 to 2*n do b[j] := j;
   A[1..n    ,1..n    ] := D;
   A[n+1..2*n,n+1..2*n] := D;
   for i := n+1 to 2*n do
      for j := 1 to n do A[i,j] := 0.0;
   for i := 1 to n do
      for j := n+1 to 2*n do
         if j=n+i then A[i,j] := -1e0 else A[i,j] := 0.0;
   lss(A,illcd,b,z,Err); { liefert Einschliessung z von x }
   if Err = 0 then
      begin
         writeln('Einschliessung des Loesungsvektors:');
         writeln(z);
         writeln;
      end     else writeln('Err = ',Err);
```

```
end; { main }
begin
   repeat
      writeln('Dimension der Dekker-Matrix = ?');  read(n);
      main(n);
   until false
end.
```

Mit der Boothroyd/Dekker Matrix der Dimension $n = 26$ ergibt sich für die Einschließung z der Lösung $\boldsymbol{x} \in I\!R^{52}$ der folgende Bildschirmausdruck:

```
Dimension der Dekker-Matrix = ?
26    <enter>
illcd = False(0),True(1) = ?
0     <enter>
Einschliessung des Loesungsvektors:

[  9.39524068999999999999E+008,  9.39524069000000000001E+008]
[ -2.34209935630000000001E+010, -2.34209935629999999999E+010]
                                *
                                *
[  2.00000000000000000000E+000,  2.00000000000000000000E+000]
[ -1.00000000000000000000E+000, -1.00000000000000000000E+000]
```

Zum Vergleich werden noch die Konditionszahlen $\varkappa(D)$, $\boldsymbol{D} \in I\!R^{26\times 26}$ und $\varkappa(A)$, $\boldsymbol{A} \in I\!R^{52\times 52}$ angegeben:

$$\varkappa(D) = 5.26 \cdot 10^{+44}, \qquad \varkappa(A) = 4.11 \cdot 10^{+63}$$

Die Konditionszahl der (52×52) Blockmatrix $\boldsymbol{A}$ ist damit um 19 Größenordnungen größer als $\varkappa(\boldsymbol{D})$.

Die Konditionszahl $\varkappa(\boldsymbol{A})$ kann noch erheblich vergrößert werden, wenn man in der Blockmatrix $\boldsymbol{A}$ die Einheitsmatrix $\boldsymbol{I}$ mit dem Faktor $10^m, m = 1, 2, \ldots$ multipliziert. Die Koeffizientenmatrix $\boldsymbol{A}$ hat demnach die folgende Gestalt:

$$\boldsymbol{A} = \boldsymbol{A}(m) := \begin{pmatrix} \boldsymbol{D} & -10^m \cdot \boldsymbol{I} \\ \boldsymbol{O} & \boldsymbol{D} \end{pmatrix} \in I\!R^{2n\times 2n}, \qquad m = 1, 2, \ldots \tag{5.11}$$

In nachfolgender Tabelle sind die Konditionszahlen der Matrix $\boldsymbol{A}$ für die Dimension $n = 2 * 26 = 52$ in Abhängigkeit von m zusammengestellt:

Die Dimension der Matrix A ist: $n = 2 * 26 = 52$ **Konditionszahl $\varkappa(A)$ in Abhängigkeit von m:**				
m	0	20	40	60
$\varkappa(A)$	$4.11 \cdot 10^{+63}$	$4.11 \cdot 10^{+83}$	$4.66 \cdot 10^{+122}$	$4.66 \cdot 10^{+162}$
m	80	100	120	140
$\varkappa(A)$	$4.66 \cdot 10^{+202}$	$4.66 \cdot 10^{+242}$	$4.66 \cdot 10^{+282}$	$4.66 \cdot 10^{+322}$
m	170	190	210	230
$\varkappa(A)$	$4.66 \cdot 10^{+382}$	$4.66 \cdot 10^{+422}$	$4.66 \cdot 10^{+462}$	$4.66 \cdot 10^{+502}$

Für alle m-Werte aus obiger Tabelle kann der gesuchte Lösungsvektor $\boldsymbol{x}$ aus (5.10) bzgl. der Koeffizientenmatrix $\boldsymbol{A}$ aus (5.11) mit Hilfe der Prozedur `lss(A,illcd,b,z,Err)` bis zur Dimension $n = 2*26 = 52$ durch `z` erfolgreich (`Err=0`) eingeschlossen werden. Dabei kann man `illcd=False` wählen, d.h. alle Rechnungen erfolgen ohne staggered Arithmetik. Für $n = 2 * 27 = 54$ muß zur Einschließung von $\boldsymbol{x}$ jedoch `illcd=True` gewählt werden, d.h. die Approximations–Inverse von $\boldsymbol{A}$ wird dann intern in staggered Arithmetik berechnet. Die erhaltene Einschließung `z` ist dann allerdings bei den ersten Komponenten so grob[1], daß eine sinnvolle Aussage über die entsprechenden Lösungskomponenten $\boldsymbol{x}[i]$ nicht mehr möglich ist.

[1] Untergrenze und Obergrenze der Intervalle $\boldsymbol{z}[i]$ haben unterschiedliche Vorzeichen.

5.1.4.3 Überbestimmte Systeme

Bei einem überbestimmten System

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{A} \in I\!R^{m\times n},\ \boldsymbol{x} \in I\!R^n,\ \boldsymbol{b} \in I\!R^m$$

existiert wegen $m > n$ i.a. kein Lösungsvektor $\boldsymbol{x}$. Als Beispiel betrachten wir mit $m = 3$ und $n = 2$ das lineare System:

$$\begin{aligned} 3x + 4y - 1 &= 0 \\ 5x + 6y - 2 &= 0 \\ 8x + 7y - 3 &= 0 \end{aligned}$$

das keine Lösung (x, y) besitzt. Die entsprechende Quadratsumme

$$F(x,y) := (3x + 4y - 1)^2 + (5x + 6y - 2)^2 + (8x + 7y - 3)^2$$

ist daher positiv und gesucht ist (x, y), so daß gilt:

$$F(x,y) \overset{!}{=} \min_{x,y\in I\!R}, \qquad \text{(Methode der kleinsten Quadrate)}$$

Für das Minimum lauten die notwendigen Bedingungen:

$$\begin{pmatrix} \dfrac{\partial F(x,y)}{\partial x} & = 0 \\ \dfrac{\partial F(x,y)}{\partial y} & = 0 \end{pmatrix} \iff \begin{pmatrix} x + y & = \frac{37}{98} \\ x + \frac{101}{98}\cdot y & = \frac{37}{98} \end{pmatrix} \iff \begin{pmatrix} x & = \frac{37}{98} \\ y & = 0 \end{pmatrix}$$

Das nachfolgende Programm `tstllss3` liefert mit

$$\boldsymbol{A} = \begin{pmatrix} 3 & 4 \\ 5 & 6 \\ 8 & 7 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

`illcd=False` und dem Prozeduraufruf

```
lss(A,illcd,b,z,Err)
```

mit $\boldsymbol{z}$ eine Einschließung für x, y:

$$x = \frac{37}{98} \in \boldsymbol{z}[1], \qquad y = 0 \in \boldsymbol{z}[2];$$

```
program tstllss3(input,output);
{ Ueber- oder unterbestimmte lin. Gleichung-Systeme:    }
{  Ax = b;   A aus R^(mxn),   b[i] := i,  i=1,2,...,m;  }
{ 1. Fall:   m>n, ueberbestimmtes System:               }
{ Berechnung einer Einschliessung des Vektors  x  fuer  }
{ den bei einem ueberbestimmten linearen Systems Ax = b }
{ die Norm    || Ax - b ||^2   minimal wird.            }
{ 2. Fall:   n>m, unterbestimmtes System:               }
{ Berechnung einer Einschliessung derjenigen Loesung x  }
{ mit minimaler Norm:  || x ||^2 = min;                 }

use llss    ,      { exportiert  lss(A,illcd,b,z,Err)   }
    lmv_ari ,      { Matrix/Vektor-Arithmetik           }
    lmvi_ari;      { Intervall-Matrix/Vektor-Arithmetik }
var m,n : integer; { Zeilen- bzw. Spalten-Zahl von  A   }
procedure main(m,n : integer) ;
var A       : Lrmatrix[1..m,1..n];
    z       : Livector[1..n];
    b       : Lrvector[1..m];
    Err,i,j : integer;
    illcd   : boolean;
begin
   writeln('Eingabe von A zeilenweise');  read(A);
   For i := 1 to m do b[i] := i;
   writeln('illcd = False(0),True(1) = ?');  read(Err);
   illcd := Err=1;
   lss(A,illcd,b,z,Err); { liefert Einschliessung z von x }
   if Err = 0 then
      begin
         writeln('Einschliessung z des gesuchten Vektors:');
         writeln(z);  writeln;
      end     else writeln('Err = ',Err);
end; { main }
begin
   repeat
      writeln('Zeilen-Zahl der Matrix = ?');  read(m);
      writeln('Spaltenzahl der Matrix = ?');  read(n);
      main(m,n);
   until false
end.
```

Das Programm `tstllss3` von Seite 307 liefert mit der Matrix $\boldsymbol{A}$ von Seite 306 den folgenden Bildschirmausdruck:

```
Zeilen-Zahl der Matrix = ?
3     <enter>
Spaltenzahl der Matrix = ?
2     <enter>
Eingabe von A zeilenweise
3 4   <enter>
5 6   <enter>
8 7   <enter>
illcd = False(0),True(1) = ?
0     <enter>
Einschliessung z des gesuchten Vektors:

[  3.77551020408163265306E-001,  3.77551020408163265307E-001]
[ -4.80000000000000000002E-297,  4.80000000000000000002E-297]
```

5.1.4.4 Unterbestimmte Systeme

Bei einem unterbestimmten System

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{A} \in \mathbb{R}^{m\times n},\ \boldsymbol{x} \in \mathbb{R}^n,\ \boldsymbol{b} \in \mathbb{R}^m$$

existieren wegen $n > m$ i.a. unendlich viele Lösungsvektoren $\boldsymbol{x}$. Als Beispiel betrachten wir mit $m = 2$ und $n = 3$ das lineare System:

$$\begin{aligned} 2x_1 + 3x_2 + 4x_3 &= 1 \\ 3x_1 + 4x_2 + 5x_3 &= 2 \end{aligned}$$

das beliebig viele Lösungen $\boldsymbol{x}$ besitzt. Einfache Rechnungen liefern für den Vektor $\boldsymbol{x} = \boldsymbol{x}(t)$ die Parameterdarstellung:

$$\boldsymbol{x}_1(t) = t + 2, \quad \boldsymbol{x}_2(t) = -2t - 1, \quad \boldsymbol{x}_3(t) = t;$$

Wir suchen diejenige Lösung $\boldsymbol{x}$, für die gilt:

$$\|\boldsymbol{x}(t)\|^2 \stackrel{!}{=} \min_{t\in\mathbb{R}}$$

Mit $F(t) := \|\boldsymbol{x}(t)\|^2 = 6t^2 + 8t + 5$ lautet dann die notwendige Bedingung für das gesuchte Minimum: $F'(t) = 12t + 8 = 0 \iff t = -2/3$.

$$\rightsquigarrow \quad \boldsymbol{x} = \begin{pmatrix} \frac{4}{3} \\ \frac{1}{3} \\ -\frac{2}{3} \end{pmatrix} \text{ ist der Lösungsvektor mit minimaler Norm } \|\boldsymbol{x}\|.$$

Das Programm `tstllss3` von Seite 307 liefert mit der Matrix

$$\boldsymbol{A} = \begin{pmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix} \quad \text{und} \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

den folgenden Bildschirmausdruck:

```
Zeilen-Zahl der Matrix = ?
2       <enter>
Spaltenzahl der Matrix = ?
3       <enter>
Eingabe von A zeilenweise
2 3 4   <enter>
3 4 5   <enter>
illcd = False(0),True(1) = ?
0     <enter>
Einschliessung z des gesuchten Vektors:

[  1.33333333333333333333E+000,  1.33333333333333333334E+000]
[  3.33333333333333333333E-001,  3.33333333333333333334E-001]
[ -6.66666666666666666667E-001, -6.66666666666666666666E-001]
```

Die Komponenten des Lösungsvektors $\boldsymbol{x}$ mit minimaler Norm $\|\boldsymbol{x}\|$ werden also von $\boldsymbol{z}$ maximalgenau eingeschlossen.

5.2 Band-Matrizen

5.2.1 Definitionen

Wir betrachten das lineare Gleichungssystem

$$\boldsymbol{Bx} = \boldsymbol{b}, \quad \boldsymbol{B} \in I\!R^{n\times n}, \quad \boldsymbol{x}, \boldsymbol{b} \in I\!R^n \tag{5.12}$$

unter der Voraussetzung, daß die Matrix $\boldsymbol{B}$ von Null verschiedene Elemente nur auf Parallelen zur Hauptdiagonalen besitzt. Solche Matrizen werden als Bandmatrizen bezeichnet.

Die Zahl $l = 0, 1, 2, \ldots$ beschreibt dasjenige Band unterhalb der Hauptdiagonalen, bis zu dem von Null verschiedene Elemente anzutreffen sind. Ganz analog beschreibt $k = 0, 1, 2, \ldots$ das entsprechende Band oberhalb der Hauptdiagonalen. Für die folgende Bandmatrix der Dimension $n = 5$ gilt $l = 2$ und $k = 1$:

$$\boldsymbol{B} = \begin{pmatrix} 3 & \mathbf{4} & 0 & 0 & 0 \\ \mathbf{2} & 3 & \mathbf{4} & 0 & 0 \\ \mathbf{1} & \mathbf{2} & 3 & \mathbf{4} & 0 \\ 0 & \mathbf{1} & \mathbf{2} & 3 & \mathbf{4} \\ 0 & 0 & \mathbf{1} & \mathbf{2} & 3 \end{pmatrix} \quad \longleftrightarrow \quad \boldsymbol{A} = \begin{pmatrix} 0 & 0 & 3 & 4 \\ 0 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 0 \end{pmatrix}$$

Demnach haben alle Elemente auf dem $l = 2$-ten Band unterhalb der Hauptdiagonalen den konstanten Wert 1. Die Elemente auf den einzelnen Bändern dürfen aber durchaus verschieden sein. Bezüglich l, k müssen lediglich die folgenden Bedingungen erfüllt sein:

$$0 \le l, k \le n-1 \quad \text{und} \quad l + k > 0$$

Für $k = 0$ und $l = n-1$ ist demnach $\boldsymbol{B}$ eine untere Dreiecksmatrix. Aus ökonomischen Gründen speichert man $\boldsymbol{B}$ nicht in seiner ursprünglichen Form sondern als äquivalente Matrix $\boldsymbol{A}$, was im Falle $n \gg 1$ und $l, k \ll n$ besonders effektiv ist. Nach dem obigen Beispiel ist die Bandmatrix $\boldsymbol{B}$ als Matrix $\boldsymbol{A}$ zusammen mit den Vektoren $\boldsymbol{b}, \boldsymbol{z}$ wie folgt zu vereinbaren:

```
var A : Lrmatrix[1..n,-l..k];
    b : Lrvector[1..n];
    z : Livector[1..n];
```

Der Intervall–Vektor $\boldsymbol{z}$ ist dabei die gesuchte, garantierte Einschließung des Lösungsvektors $\boldsymbol{x}$.

5.2.2 Anwendungsroutinen

Zur Eingabe der Matrix $\boldsymbol{A}$ und des Vektors $\boldsymbol{b}$ steht folgende Prozedur zur Verfügung:

procedure `read_Ab` (**var** A : ■; **var** b □);

Die folgende Tabelle liefert für ■, □ die erlaubten Datentypen und die jeweils exportierenden Module:

Zuordnung von ■, □ zu den erlaubten Datentypen		
`A` : ■	`(L)rmatrix`	`(L)imatrix`
`b` : □	`(L)rvector`	`(L)ivector`
exportierende Module	(l)band	

Nach dem Prozeduraufruf `read_Ab(A,b)` ist zunächst die Dimension `n` der quadratischen Matrix $\boldsymbol{B}$ einzugeben; danach muß die jeweilige Anzahl `l,k` der Bänder unter bzw. oberhalb der Hauptdiagonalen bereitgestellt werden. Dann sind die $(l + k + 1)$ konstanten Bandwerte einzugeben, wobei mit dem Wert des untersten l-ten Bandes zu beginnen ist. Verlangt die Anwendung, daß die zunächst konstanten Bandwerte an einigen Positionen zu ändern sind, so ist die entsprechende Frage mit `y` zu beantworten. Danach ist unter Angabe der Zeilen- und Spaltenzahl[2] der jeweils neue Wert einzugeben. Die Eingabe der Änderungen wird dann durch `0 0 <enter>` beendet. Die anschließende Eingabe des Vektors $\boldsymbol{b}$ kann entweder dadurch erfolgen, daß man die verschiedenen $b[i]$ nacheinander eingibt, oder daß man mit `= 3.6` z.B. den konstanten Wert 3.6 vorgibt, der dann wieder nach Bedarf an den gewünschten Positionen geändert werden kann, wobei diese Änderungen mit `0 <enter>` abgeschlossen werden. Danach stehen sowohl der Vektor $\boldsymbol{b}$ als auch die Matrix $\boldsymbol{A}$ zur Verfügung, wobei die Rechteckmatrix $\boldsymbol{A}$ durch die vorgegebene quadratische Matrix $\boldsymbol{B}$ nach dem Beispiel von Seite 310 bestimmt ist. Alle durch den Prozeduraufruf `read_Ab(A,b)` notwendigen Eingaben erfolgen mit dem Anwender interaktiv.

[2] Die Zeilen- und Spaltenzahlen beziehen sich auf die quadratische Matrix $\boldsymbol{B}$.

Nach Bereitstellung von $\boldsymbol{A}, \boldsymbol{b}$ erfolgt die Einschließung des Lösungsvektors $\boldsymbol{x}$ durch den Intervall–Vektor $\boldsymbol{z}$ mit Hilfe der Prozedur

procedure `lss` (**var** A: $\blacksquare$; **var** b: $\square$; **var** z: $\diamond$; **var** Err: integer);

Die folgende Tabelle liefert für $\blacksquare, \square, \diamond$ die erlaubten Datentypen und die jeweils exportierenden Module:

Zuordnung von $\blacksquare, \square, \diamond$ zu den Datentypen		
`A` : $\blacksquare$	`(L)rmatrix`	`(L)imatrix`
`b` : $\square$	`(L)rvector`	`(L)ivector`
`z` : $\diamond$	`(L)ivector`	
exportierende Module	(l)band	

5.2.3 Programmbeispiele

5.2.3.1 Tschebyscheff–Polynome

Die Auswertung eines Polynoms der Ordnung n kann stets zurückgeführt werden auf die Lösung eines linearen Gleichungssystems $\boldsymbol{Bx} = \boldsymbol{b}$, dessen quadratische Koeffizientenmatrix $\boldsymbol{B} \in I\!R^{(n+1)\times(n+1)}$ eine Bandmatrix mit $l = 1$ und $k = 0$ ist, [5, Seite 58]. Das schnelle iterative Lösungsverfahren liefert eine Einschließung des Lösungsvektors $\boldsymbol{x}$, wobei allerdings die $n+1$ Polynomkoeffizienten explizit angegeben werden müssen. Die lästige Koeffizienteneingabe läßt sich jedoch vermeiden, wenn das Polynom durch eine Rekursionsformel gegeben ist. Für die Tschebyscheff–Polynome $T_n(\zeta)$ gilt beispielsweise:

$$T_0(\zeta) = 1,\ T_1(\zeta) = 1, \qquad T_{n+1}(\zeta) - 2\zeta T_n(\zeta) + T_{n-1}(\zeta) = 0$$

und die Auswertung von $T_n(\zeta)$ an der Stelle $\zeta = \zeta_0$ wird realisiert durch die Lösung des linearen Band-Systems $\boldsymbol{Bx} = \boldsymbol{b}$:

$$\begin{pmatrix} 1 & & & & \\ 0 & 1 & & & \\ 1 & -2\zeta_0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2\zeta_0 & 1 \end{pmatrix} \begin{pmatrix} T_0(\zeta_0) \\ T_1(\zeta_0) \\ T_2(\zeta_0) \\ \vdots \\ T_n(\zeta_0) \end{pmatrix} = \begin{pmatrix} 1 \\ \zeta_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \text{mit: } l = 2,\ k = 0,$$

wobei die letzte Komponente des Lösungsvektors

$$\boldsymbol{x} = (T_0(\zeta_0), T_1(\zeta_0), T_2(\zeta_0), \ldots, T_n(\zeta_0))^T$$

den gesuchten Funktionswert $T_n(\zeta_0)$ liefert und die Eingabe der Polynomkoeffizienten jetzt nicht notwendig ist.

Das nachfolgende Programm `tstlband`

```
program tstlband(input,output);
{ Ax = b; A: reelle Bandmatrix (Longreal) }
{ Berechnet wird Einschliessung von  x    }
use lband   ,  { exportiert lss(A,b,z,Err)          }
    li_ari  ,  { Intervall-Arithmetik               }
    lmvi_ari,  { Intervall-Matrix/Vektor-Arithmetik }
    lmv_ari ;  { Matrix/Vektor-Arithmetik           }
var n,l,k : integer; { n: Zeilenzahl der Matrix A }
procedure main(n,l,k : integer) ;
var A   : Lrmatrix[1..n,-l..k];
    b   : Lrvector[1..n];
    z   : Livector[1..n]; { Einschliessung von x  }
    Err : integer;      { Err: liefert Fehlerkode }
begin
   writeln('Input of the band values of A:');
   read_Ab(A,b);
   lss(A,b,z,Err);
   if Err = 0 then
   begin
      writeln('Inclusion of the component x[',n:2,'] by:');
      writeln(z[n]);
      writeln;
   end        else writeln('Err = ',Err);
end; { main }
begin
   repeat
      writeln('Dimension of the quadratic matrix A = ?');
      read(n);
      writeln('Number l of the lower bands = ?');  read(l);
      writeln('Number k of the upper bands = ?');  read(k);
      main(n,l,k);
   until false
end.
```

liefert mit der Eingabe

```
Dimension of the quadratic matrix A = ?
51  <enter>
Number l of the lower bands = ?
2   <enter>
Number k of the upper bands = ?
0   <enter>
Input of the band values of A:
 3 band values (start with value of lowest band) of A = ?
1 -1.98 1  <enter>
change individual band elements of A ? (y/n) y    <enter>
row = 0, col = 0 <enter> finish the change of individual
                                                elements
row, col, new values: 2 1 0  <enter>
row = 0, col = 0 <enter> finish the change of individual
                                                elements
row, col, new values: 0 0  <enter>
 b = ?
 input of  = REAL-value   <enter>  ===>  b[i] = const.
 input of  value1 value2..<enter>  ===>  different b[i]
=0  <enter>
change individual elements of b ?  (y/n) y  <enter>
row = 0 <enter> finishs the change of individual elements
row, new value 1 1  <enter>
row = 0 <enter> finishs the change of individual elements
row, new value 2 0.99  <enter>
row = 0 <enter> finishs the change of individual elements
row, new value 0  <enter>
Inclusion of the component x[51] by
[7.01149198176931383221E-001, 7.01149198176931383222E-001]
```

die maximalgenaue Einschließung

$$T_{50}(0.99) \in 7.0114919817693138322^{2}_{1} \cdot 10^{-1}$$

Das Programm `tstlband` liefert mit den analogen Eingaben die maximalgenaue Einschließung: $T_{900}(0.99) \in -0.15045008257509453945^{8}_{9}$.
Ändert man in den Programmzeilen 10,11 die Datentypen in `Limatrix` und in `Livector`, so erhält man das Ergebnis:

$$T_{50}([0.99, 0.99 + 10^{-20}]) \subset 0.7011491981769313^{99472}_{69498}$$

Wenn in Gleichung (5.12) auf Seite 310 die Bandmatrix $\boldsymbol{B}$ und die rechte Seite $\boldsymbol{b}$ Intervallelemente enthalten, so treten mit wachsendem Polynomgrad n sehr schnell die bekannten Intervallüberschätzungen auf.

5.2.3.2 Toeplitz–Matrix

In (5.12) betrachten wir als Bandmatrix $\boldsymbol{B}$ die symmetrische Toeplitz–Matrix mit $l = k = 2$ und den jeweils konstanten Bandwerten 1,2,4,2,1. Auf der rechten Seite werden alle $b[i] = 1$ gesetzt. Mit der Dimension $n = 120000$ liefert das Programm `tstlband` mit den entsprechenden Eingaben

```
Dimension of the quadratic matrix A = ?
120000  <enter>
Number l of the lower bands = ?
2   <enter>
Number k of the upper bands = ?
2   <enter>
Input of the band values of A:
 5 band values (start with value of lowest band) of A = ?
1 2 4 2 1  <enter>
change individual band elements of A ? (y/n) n    <enter>
 b = ?
 input of  = REAL-value   <enter>  ===>  b[i] = const.
 input of  value1 value2..<enter>  ===>  different b[i]
=1  <enter>
change individual elements of b ?  (y/n) n  <enter>
```

die maximalgenaue Einschließung der letzten Lösungskomponente

$$x[120000] \in 0.186014606747918020057_{6}^{7}$$

Der in [8, Seite 36] beschriebene Algorithmus zur Lösung von (5.12) liefert damit auch noch bei großen Matrixdimensionen n hervorragende Ergebnisse!

Kapitel 6

Differenzengleichungen

Differenzengleichungen kommen z.B. bei der numerischen Integration und bei Randwertproblemen häufig zur Anwendung. In diesem Kapitel werden nur solche Routinen angegeben, mit denen man garantierte Einschließungen der entsprechenden Lösungen berechnen kann. Dabei wird zwischen skalaren und vektoriellen, linearen Differenzengleichungen unterschieden, obwohl erstere stest auf spezielle vektorielle Differenzengleichungen zurückgeführt werden können, [8, Seite 46].

6.1 Skalare Differenzengleichungen

6.1.1 Definitionen

Bei einer skalaren Differenzengleichung der Ordnung m sind zunächst m reelle Startwerte $x_0, x_1, \ldots, x_{m-1}$ gegeben. Mit Hilfe der Gleichung

$$a_{0,m}x_m + a_{0,m-1}x_m + \ldots + a_{0,0}x_0 = b_0, \quad a_{0,m} \neq 0$$

kann die Unbekannte x_m direkt berechnet werden:

$$x_m = \frac{b_0}{a_{0,m}} - \frac{a_{0,m-1}}{a_{0,m}}x_{m-1} - \ldots - \frac{a_{0,0}}{a_{0,m}}x_0; \qquad \text{und die nächste Glg.}$$

$$a_{1,m}x_{m+1} + a_{1,m-1}x_m + \ldots + a_{1,0}x_1 = b_1, \quad a_{1,m} \neq 0 \quad \text{liefert:}$$

$$x_{m+1} = \frac{b_1}{a_{1,m}} - \frac{a_{1,m-1}}{a_{1,m}}x_m - \ldots - \frac{a_{1,0}}{a_{1,m}}x_1; \qquad \text{usw.}$$

Mit den Startwerten $x_0, x_1, \ldots, x_{m-1}$ läßt sich damit eine Folge von Werten $x_m, x_{m+1}, x_{m+2}, \ldots$ rekursiv berechnen, wenn die b_i und die $a_{i,j}$ gegeben sind; $i = 0, 1, \ldots;$ $j = 0, 1, \ldots, m;$

Zusammenfassung:

Skalare Differenzengleichung:

$x_0, x_1, \ldots, x_{m-1}$ (Startwerte)

$a_{i,m}x_{m+i} + a_{i,m-1}x_{m+i-1} + \ldots + a_{i,0}x_i = b_i, \quad i = 0, 1, \ldots$

Lösungen:

$x_{m+i} = \frac{b_i}{a_{i,m}} - \frac{a_{i,m-1}}{a_{i,m}}x_{m+i-1} - \ldots - \frac{a_{i,0}}{a_{i,m}}x_i; \qquad i = 0, 1, \ldots$

Falls die $a_{i,j},\ j = 0, 1, \ldots, m$ von i unabhängig sind, spricht man von einer linearen Differenzengleichung der Ordnung m mit **konstanten** Koeffizienten.

6.1.2 Routinen zur Berechnung der Lösungen

Prinzipiell lassen sich mit Hilfe der Intervallarithmetik Einschließungen der Lösungen x_{m+i} sukzessive berechnen. Dabei ergeben sich jedoch schon ab etwa $i = 100$ so große Intervallüberschätzungen, daß weitere Rechnungen sinnlos werden. Die folgende Prozedur `diff_iter` benutzt jedoch einen in [8, Seite 41] beschriebenen Algorithmus, der brauchbare Einschließungen der Lösungen x_{m+i} auch noch für $i = 100\,000$ liefert.

Die folgende Prozedur `diff_sc`

```
procedure diff_sc (var a   : ivector;  var b  : interval;
(differ)           var B0  : rmatrix;
                   var z0  : ivector;  var y0 : rvector;
                   var B1  : rmatrix;
                   var z1  : ivector;  var y1 : rvector;
                   var Err : integer );
```

berechnet mit Hilfe der m Werte $x_i, x_{i+1}, \ldots, x_{m+i-1}$, welche durch die Komponenten des Intervallvektors $\texttt{y0} + \texttt{B0} \cdot \texttt{z0}$ einzuschließen sind, mit dem Vektor $\texttt{y1} + \texttt{B1} \cdot \texttt{z1}$ eine Einschließung der m Werte $x_{i+1}, x_{i+2}, \ldots, x_{m+i}$, wobei also der neue Wert x_{m+i} in der letzten Komponente von $\texttt{y1} + \texttt{B1} \cdot \texttt{z1}$ enthalten ist und x_{m+i} durch

$$x_{m+i} := \frac{b}{a[m]} - \frac{a[m-1]}{a[m]} \cdot x_{m+i-1} - \ldots - \frac{a[0]}{a[m]} \cdot x_i$$

definiert ist. Enthalten die Komponenten des Vektors `x0 : ivector[1..m]` die m Werte $x_i, x_{i+1}, \ldots, x_{i+m-1}$, so werden diese mit Hilfe der Anweisungen

```
var B0    : rmatrix[1..m,1..m];
    y0    : rvector[1..m];
    x0,z0 : ivector[1..m];
begin
   B0 := ID(B0);
   y0 := mid(x0);
   z0 := x0 - y0;
end;
```

durch den obigen Intervallvektor `y0` + `B0` · `z0` tatsächlich eingeschlosssen.

Bei der Prozedur **`diff_sc`** müssen alle Indexbereiche der benutzten Parameter mit 1 beginnen; nur der Indexbereich von **`a`** muß mit 0 starten, d.h. `var a : ivector[0..m];`

Anwendungsbeispiele für den Basistyp **`real`** findet man ab Seite 320.

Für den Basistyp `Longreal` steht die gleiche Prozedur mit den gleichen Eigenschaften zur Verfügung:

procedure `diff_sc` (**var** `a` : `Livector`; **var** `b` : `Linterval`;
(ldiffer)
var `B0` : `Lrmatrix`;
var `z0` : `Livector`; **var** `y0` : `Lrvector`;
var `B1` : `Lrmatrix`;
var `z1` : `Livector`; **var** `y1` : `Lrvector`;
var `Err` : `integer`);

Anwendungsbeispiele für den Basistyp `Longreal` findet man ab Seite 322.

6.1.3 Programmbeispiele

6.1.3.1 Rechnungen zum Basistyp real

Das folgende Programm tstdiff

```
program tstdiff(input,output);
{ Lineare Differenzengleichung der Ordnung m mit }
{ konstanten, reellen, Intervall-Koeffizienten;  }
{ Basistyp REAL;           Blomquist, 05.11.1999 }
use i_ari, mv_ari, mvi_ari, differ;
var m            : integer;
procedure main(m : integer);
var k,n,err : integer;
   a        : ivector[0..m]; { Konstante Koeffizienten }
   B0,B1    : rmatrix[1..m,1..m];
   y0,y1    : rvector[1..m];
   z0,z1,x0 : ivector[1..m];
   b        : interval;       { Konstante rechte Seite }
begin
   writeln(m+1:2,' Intervalle a[0],a[1],...,a[',m,'] = ?');
   read(a);
   writeln('Rechtes Intervall b = ?');  read(b);
   repeat
      writeln(m:2,' Startintervalle x[i] = ?');  read(x0);
      writeln('Iterations-Zahl = ?');  read(n);
      B0 := ID(B0) ;  { Der Intervall-Vektor y0+B0*z0 }
      y0 := mid(x0);  { ist eine Einschliessung der m }
      z0 := x0 - y0;  { Startwerte x[k] aus x0[k+1],  }
                      { k = 0,1,...,m-1;              }
      For k := 1 to n do
      begin
         diff_sc(a,b,B0,z0,y0,B1,z1,y1,err);
         if err <> 0 then differErrMsg(1);
         B0 := B1;  z0 := z1;  y0 := y1;
         { Der Intervall-Vektor  y1+B1*z1  enthaelt die m }
         { Werte x[i+1],x[i+2],...,x[i+m], wobei der neue }
         { Wert x[i+m] durch die m-te Komponente des      }
         { Intervall-Vektors eingeschlossen wird.         }
      end;
      write('Von den ',n,' neu berechneten Loesungen lauten');
```

```
      writeln(' die letzten ',m,' Einschliessungen:');
      writeln(y1 + B1*z1);
   until n <= 0;
end; { main }

begin
   repeat
      writeln('Ordnung m = ?');  read(m);
      if m > 0 then main(m);
   until m <= 0;
end.
```

liefert mit der punktförmigen Eingabe

```
Ordnung m = ?
12   <enter>
13 Intervalle a[0],a[1],...,a[12] = ?
1 2 3 4 5 6 7 8 9 10 11 12 13    <enter>
Rechtes Intervall b = ?
0.1  <enter>
12 Startintervalle x[i] = ?
1 2 3 4 5 6 7 8 9 10 11 12       <enter>
Iterations-Zahl = ?
300  <enter>
```

das Ergebnis:

```
Von den 300 neu berechneten Loesungen lauten die
letzten 12 Einschliessungen:

[ 1.098901072906E-003,  1.098901124897E-003 ]
[ 1.098901081681E-003,  1.098901116120E-003 ]
[ 1.098901083917E-003,  1.098901113886E-003 ]
[ 1.098901081083E-003,  1.098901116719E-003 ]
[ 1.098901081069E-003,  1.098901116734E-003 ]
[ 1.098901085302E-003,  1.098901112500E-003 ]
[ 1.098901088773E-003,  1.098901109029E-003 ]
[ 1.098901091414E-003,  1.098901106388E-003 ]
[ 1.098901090521E-003,  1.098901107282E-003 ]
[ 1.098901090167E-003,  1.098901107634E-003 ]
[ 1.098901091679E-003,  1.098901106124E-003 ]
[ 1.098901093061E-003,  1.098901104741E-003 ]
```

Das gleiche Programm `tstdiff` liefert mit der Intervall–Eingabe

```
Ordnung m = ?
12   <enter>
13 Intervalle a[0],a[1],...,a[12] = ?
[1,1.000000000001] 2 3 4 5 6 7 8 9 10 11 12 13    <enter>
Rechtes Intervall b = ?
0.1  <enter>
12 Startintervalle x[i] = ?
1 2 3 4 5 6 7 8 9 10 11 12       <enter>
Iterations-Zahl = ?
300  <enter>
```

das Ergebnis:

```
Von den 300 neu berechneten Loesungen lauten die
letzten 12 Einschliessungen:
[ 1.098901065177E-003,  1.098901132626E-003 ]
[ 1.098901065904E-003,  1.098901131897E-003 ]
[ 1.098901067650E-003,  1.098901130153E-003 ]
[ 1.098901070618E-003,  1.098901127184E-003 ]
[ 1.098901079290E-003,  1.098901118513E-003 ]
[ 1.098901083956E-003,  1.098901113846E-003 ]
[ 1.098901080567E-003,  1.098901117235E-003 ]
[ 1.098901083170E-003,  1.098901114631E-003 ]
[ 1.098901084117E-003,  1.098901113686E-003 ]
[ 1.098901086194E-003,  1.098901111607E-003 ]
[ 1.098901090431E-003,  1.098901107372E-003 ]
[ 1.098901091136E-003,  1.098901106666E-003 ]
```

6.1.3.2 Rechnungen zum Basistyp `Longreal`

Das folgende Programm `tstldiff`

```
program tstldiff(input,output);
{ Lineare Differenzengleichung der Ordnung m mit }
{ konstanten, reellen, Intervall-Koeffizienten;  }
{ Basistyp Longreal,       Blomquist, 05.11.1999 }
use li_ari, lmv_ari, lmvi_ari, ldiffer;
var m           : integer;
procedure main(m : integer);
var k,n,err : integer;
```

```
   a        : Livector[0..m]; { Konstante Koeffizienten }
   B0,B1    : Lrmatrix[1..m,1..m];
   y0,y1    : Lrvector[1..m];
   z0,z1,x0 : Livector[1..m];
   b        : Linterval;       { Konstante rechte Seite }
begin
   writeln(m+1,' Intervalle a[0],a[1],...,a[',m,'] = ?');
   read(a);
   writeln('Rechtes Intervall b = ?');  read(b);
   repeat
      writeln(m,' Startintervalle x[i] = ?');  read(x0);
      writeln('Iterations-Zahl = ?');  read(n);
      B0 := LID(B0) ; { Der Intervall-Vektor y0+B0*z0 }
      y0 := mid(x0);  { ist eine Einschliessung der m }
      z0 := x0 - y0;  { Startwerte x[k] aus x0[k+1],  }
                      { k = 0,1,...,m-1;              }
      For k := 1 to n do
      begin
         diff_sc(a,b,B0,z0,y0,B1,z1,y1,err);
         if err <> 0 then ldifferErrMsg(1);
         B0 := B1;  z0 := z1;  y0 := y1;
         { Der Intervall-Vektor  y1+B1*z1  enthaelt die m }
         { Werte x[i+1],x[i+2],...,x[i+m], wobei der neue }
         { Wert x[i+m] durch die m-te Komponente des      }
         { Intervall-Vektors eingeschlossen wird.         }
      end;
      write('Von den ',n,' neu berechneten Loesungen lauten');
      writeln(' die letzten ',m,' Einschliessungen:');
      writeln(y1 + B1*z1);
   until n <= 0;
end; { main }

begin
   repeat
      writeln('Ordnung m = ?');  read(m);
      if m > 0 then main(m);
   until m <= 0;
end.
```

liefert mit der Eingabe

```
Ordnung m = ?
2           <enter>
3 Intervalle a[0],a[1],...,a[2] = ?
1 -2 1      <enter>
Rechtes Intervall b = ?
0.5         <enter>
2 Startintervalle x[i] = ?
0 2         <enter>
Iterations-Zahl = ?
50000       <enter>
```

das Ergebnis:

```
  Von den 50000 neu berechneten Loesungen lauten
  die letzten 2 Einschliessungen:

[ 6.25087500000000000000E+008, 6.25087500000000000000E+008 ]
[ 6.25112502000000000000E+008, 6.25112502000000000000E+008 ]
```

Das gleiche Programm `tstldiff` liefert mit der Intervall–Eingabe

```
Ordnung m = ?
2           <enter>
3 Koeffizienten a[0],a[1],...,a[2] = ?
[1,1.00000000000000000001] -2
[1,1.00000000000000000001]     <enter>
Rechte Seite b = ?
[0.5,0.500000000000000000001]  <enter>
2 Startwerte x[i] = ?
0 2         <enter>
Iterations-Zahl = ?
50000       <enter>
```

das Ergebnis:

```
  Von den 50000 neu berechneten Loesungen lauten
  die letzten 2 Einschliessungen:

[ 6.25087499996841756870E+008, 6.25087500000502667814E+008 ]
[ 6.25112501996841615187E+008, 6.25112502000502599348E+008 ]
```

Dieses letzte Beispiel zeigt den Vorteil des Basistyps `Longreal` mit seinen 21 Dezimalstellen, denn selbst bei 50 000 Iterationen mit Intervallkoeffizienten bleiben für die Einschließungen der beiden letzten Lösungen noch 11 korrekte Ziffern!

6.2 Vektorielle Differenzengleichungen

6.2.1 Definition

Wir betrachten eine Vektor–Differenzengleichung 1. Ordnung, die wie folgt definiert ist:

Vektorielle Differenzengleichung:

Startvektor: $\boldsymbol{x}_0$

$\boldsymbol{x}_{i+1} = \boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{d}_i, \quad i = 0, 1, 2, \ldots$

Zu gegebem Startvektor $\boldsymbol{x}_0$ berechnet man mit den quadratischen Matrizen $\boldsymbol{A}_i$ und dem Summanden $\boldsymbol{d}_i$ zukzessive die Vektoren $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ bis zum maximalen Index $i = n$, der vom Anwender festgelegt wird.

6.2.2 Routinen zur Berechnung der Lösungen

Prinzipiell lassen sich mit Hilfe der Intervallarithmetik Einschließungen der Lösungen $\boldsymbol{x}_{i+1}$ sukzessive berechnen. Dabei ergeben sich jedoch schon bei kleinerer Iterationszahl so große Intervallüberschätzungen, daß dann weitere Rechnungen sinnlos werden. Die folgende Prozedur `diff_mv` benutzt jedoch einen in [8, Seite 41] beschriebenen Algorithmus, der auch noch für große n brauchbare Einschließungen der Lösung $\boldsymbol{x}_n$ liefert.
Die folgende Prozedur `diff_mv`

```
procedure diff_mv (var A   : imatrix; var d  : ivector;
(differ)           var B0  : rmatrix;
                   var z0  : ivector; var y0 : rvector;
                   var B1  : rmatrix;
                   var z1  : ivector; var y1 : rvector;
                   var Err : integer );
```

berechnet mit Hilfe des alten Vektors $\boldsymbol{x}_i \in \mathtt{y0} + \mathtt{B0} \cdot \mathtt{z0}$ den neuen Vektor $\boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{d}$, der in $\mathtt{y1} + \mathtt{B1} \cdot \mathtt{z1}$ enthalten ist.

Enthält der Intervallvektor `x0 : ivector[1..m]` z.B. nach `read(x0)` den Startvektor $\boldsymbol{x}_0$, so wird dieser mit Hilfe der Anweisungen

```
var B0    : rmatrix[1..m,1..m];
    y0    : rvector[1..m];
    x0,z0 : ivector[1..m];
begin
   B0 := ID(B0);  { Einheitsmatrix }
   y0 := mid(x0);
   z0 := x0 - y0;
end;
```

durch den obigen Intervallvektor $\mathtt{y0}+\mathtt{B0}\cdot\mathtt{z0} \ni \boldsymbol{x}_0$ tatsächlich eingeschlosssen.

Bei der Prozedur `diff_mv` müssen alle Indexbereiche der benutzten Parameter mit 1 beginnen; z.B. `var A : imatrix[1..m,1..m];`

Ein Anwendungsbeispiel für den Basistyp `real` findet man auf Seite 327.

Für den Basistyp `Longreal` steht die gleiche Prozedur mit den gleichen Eigenschaften zur Verfügung:

```
procedure diff_mv (var A   : Limatrix; var d  : Livector;
(ldiffer)          var B0  : Lrmatrix;
                   var z0  : Livector; var y0 : Lrvector;
                   var B1  : Lrmatrix;
                   var z1  : Livector; var y1 : Lrvector;
                   var Err : integer );
```

Ein Anwendungsbeispiel für den Basistyp `Longreal` findet man auf Seite 329.

6.2.3 Programmbeispiele

In den beiden folgenden Anwendungen setzen wir voraus, daß die $\boldsymbol{A}_i = \boldsymbol{A}$ konstant sind und daß die $\boldsymbol{d}_i$ alle verschwinden. Nimmt man zusätzlich an, daß $\boldsymbol{A}$ orthogonal ist mit $\det(\boldsymbol{A}) = 1$, so bedeutet jeder Iterationsschritt $\boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i$ eine reine Drehung des Vektors $\boldsymbol{x}_i$ nach $\boldsymbol{x}_{i+1}$.

6.2.3.1 Drehung um die y-Achse

Die folgende Matrix

$$\boldsymbol{A} = \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{pmatrix} \quad \text{mit} \quad \boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I} \quad \text{und} \quad \det(\boldsymbol{A}) = 1$$

bewirkt eine Drehung alle Punkte des $I\!R^3$ um die y-Achse um den Winkel φ im Uhrzeigersinn. Mit $\cos\varphi = 0.6$ und $\sin\varphi = 0.8$, d.h. $\varphi \in 0.927295218001^7_6$, liefert das Programm `tstdiffv`

```
program tstdiffv(input,output);
{ Vektor-Differenzengleichung der Ordnung 1 mit }
{ konstanten, reellen, Intervall-Koeffizienten; }
{ Basistyp REAL;          Blomquist, 05.11.1999 }
use i_ari, mv_ari, mvi_ari, differ;
var m            : integer;
procedure main(m : integer);
var k,n,err : integer;
   A        : imatrix[1..m,1..m];
   B0,B1    : rmatrix[1..m,1..m];
   y0,y1    : rvector[1..m];
   z0,z1,x0 : ivector[1..m];
   d        : ivector[1..m];
begin
   write('Eingabe der ',m,'x',m,' Intervall-Matrix ');
   writeln('zeilenweise:');
   read(A);
   writeln(m,' dimensionaler Intervall-Vektor d = ?');
   read(d);
   repeat
      writeln(m,' dimensionaler Intervall-Startvektor = ?');
      read(x0);
      writeln('Iterations-Zahl = ?');  read(n);
      B0 := ID(B0) ;  { Der Intervall-Vektor y0+B0*z0 }
      y0 := mid(x0);  { ist eine Einschliessung des   }
      z0 := x0 - y0;  { Startvektors x_0              }
      For k := 1 to n do
      begin
```

```
         diff_mv(A,d,B0,z0,y0,B1,z1,y1,err);
         if err <> 0 then differErrMsg(1);
         B0 := B1;  z0 := z1;  y0 := y1;
         { Der Intervall-Vektor  y1+B1*z1  enthaelt den }
         { iterierten Vektor.                           }
      end;
      writeln('Der ',n,'-te Loesungsvektor ist enthalten in:');
      writeln(y1 + B1*z1);
   until n <= 0;
end; { main }

begin
   repeat
      writeln('Matrixdimension m = ?');  read(m);
      if m > 0 then main(m);
   until m <= 0;
end.
```

mit der Eingabe

```
Matrixdimension m = ?
3           <enter>
Eingabe der 3x3 Intervall-Matrix zeilenweise:
0.6  0 0.8  <enter>
0    1   0  <enter>
-0.8 0 0.6  <enter>
3 dimensionaler Intervall-Vektor d = ?
0 0 0       <enter>
3 dimensionaler Intervall-Startvektor = ?
1 2 0       <enter>
Iterations-Zahl = ?
1000        <enter>
```

das Ergebnis:

```
Der 1000-te Loesungsvektor ist enthalten in:

[ -8.651308139488E-001, -8.651308138120E-001 ]
[  1.999999999999E+000,  2.000000000001E+000 ]
[  5.015462838153E-001,  5.015462839487E-001 ]
```

6.2.3.2 Drehung in der x, y-Ebene

Die folgende Matrix

$$\boldsymbol{A} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \quad \text{mit} \quad \boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I} \quad \text{und} \quad \det(\boldsymbol{A}) = 1$$

bewirkt eine Drehung alle Punkte der x, y-Ebene um den Winkel φ im Uhrzeigersinn. Mit $\cos 30^0 = \sqrt{3}/2$ und $\sin 30^0 = 1/2$ wird durch $x_{i+1} = \boldsymbol{A}\boldsymbol{x}_i$ jeder Vektor $\boldsymbol{x}_i$ in der x, y-Ebene um 30^0 im Uhrzeigersinn gedreht. Das folgende Programm `tstldiffv`

```
program tstldiffv(input,output);
{ Vektor-Differenzengleichung der Ordnung 1 mit }
{ konstanten, reellen, Intervall-Koeffizienten; }
{ Basistyp LONGREAL;      Blomquist, 05.11.1999 }
use li_ari, lmv_ari, lmvi_ari, ldiffer;
var m            : integer;
procedure main(m : integer);
var k,n,err   : integer;
   A          : Limatrix[1..m,1..m];
   B0,B1      : Lrmatrix[1..m,1..m];
   y0,y1      : Lrvector[1..m];
   z0,z1,x0,d : Livector[1..m];
begin
   write('Eingabe der ',m,'x',m,' Intervall-Matrix ');
   writeln('zeilenweise:');
   read(A);
   writeln(m,' dimensionaler Intervall-Vektor d = ?');
   read(d);
   repeat
      writeln(m,' dimensionaler Intervall-Startvektor = ?');
      read(x0);
      writeln('Iterations-Zahl = ?');  read(n);
      B0 := LID(B0) ;  { Der Intervall-Vektor y0+B0*z0 }
      y0 := mid(x0);   { ist eine Einschliessung des   }
      z0 := x0 - y0;   { Startvektors x_0              }
      For k := 1 to n do
      begin
         diff_mv(A,d,B0,z0,y0,B1,z1,y1,err);
         if err <> 0 then ldifferErrMsg(1);
```

```
         B0 := B1;  z0 := z1;  y0 := y1;
         { Der Intervall-Vektor  y1+B1*z1  enthaelt den }
         { neuen Vektor.                                }
      end;
      writeln('Der ',n,'-te Loesungsvektor ist enthalten in:');
      writeln(y1 + B1*z1);
   until n <= 0;
end; { main }

begin
   repeat
      writeln('Matrixdimension m = ?');  read(m);
      if m > 0 then main(m);
   until m <= 0;
end.
```

liefert mit der Eingabe:

```
 Matrixdimension m = ?
 2       <enter>
 Eingabe der 2x2 Intervall-Matrix zeilenweise:
 [  8.66025403784438646763E-001,  8.66025403784438646764E-001 ]
 0.5     <enter>
 -0.5
 [ -8.66025403784438646764E-001, -8.66025403784438646763E-001 ]
         <enter>
 2 dimensionaler Intervall-Vektor d = ?
 0 0     <enter>
 2 dimensionaler Intervall-Startvektor = ?
 1 0     <enter>
 Iterations-Zahl = ?
 120000  <enter>
```

das Ergebnis:

```
Der 120000-te Loesungsvektor ist enthalten in:

[  9.99999999999999870717E-001,  1.00000000000000042001E+000 ]
[ -7.46410161513777785824E-017,  5.46410161513777456250E-017 ]
```

120 000 Drehungen um je 30^0 liefern also erwartungsgemäß eine Einschließung des Startvektors $(1,0)^T$.

Kapitel 7

Polynomauswertung

7.1 Überblick

Die Auswertung von Polynomen

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0, \quad a_n \neq 0$$

der Ordnung n ist in der Numerik eine häufig gestellte Aufgabe, z.B. wenn Standardfunktionen durch solche Polynome zu approximieren sind. In der **PASCAL-XSC** Umgebung besteht nun die eigentliche Aufgabe darin, nicht nur Approximationen $\widetilde{P}_n(x) \approx P_n(x)$, sondern garantierte **Einschließungen** der Polynomwerte $P_n(x)$ zu berechnen.

Das mit Abstand schnellste Verfahren besteht darin, für punktförmige Argumente $x \in [\alpha, \beta]$ einen Näherungswert $\widetilde{P}_n(x) \approx P_n(x)$ nach dem Hornerschema zu berechnen und mit einer a priori Fehlerabschätzung relative Fehlerschranken zu bestimmen, mit denen dann eine garantierte Einschließung des exakten Polynomwertes $P_n(x)$ angegeben werden kann. Da die genannten Fehlerschranken für jedes Polynom und für jedes Intervall $[\alpha, \beta]$ mit speziellen Programmen vorher jeweils einzeln zu berechnen sind und weil die damit gewonnenen Einschließungen i.a. nicht maximalgenau ausfallen, wird auf dieses Verfahren hier nicht näher eingegangen, [9].

In diesem Kapitel betrachten wir nur selbsverifizierende Algorithmen, die für jedes Polynomargument x automatisch eine garantierte Einschließung von $P_n(x)$ liefern. Bei der Realisierung unterscheidet man im Wesentlichen zwei Methoden:

1. $P_n(x)$ wird intervallmäßig nach dem Hornerschema ausgewertet, wobei x als Intervallargument betrachtet wird. Die Inklusionsmonotonie der

Intervallrechnung liefert dann automatisch eine garantierte Einschließung des Polynomwertes $P_n(x)$, [7]. Insbesondere in der Nähe einer Nullstelle liefert dieses Verfahren wegen der bekannten Intervallüberschätzungen zu grobe Einschließungen, so daß man in diesen Fällen auf die etwas aufwendigere nächste Methode zurückgreifen muß. Die Funktionen `Poly_Horn(...)` verwenden das beschriebenen intervallmäßige Hornerschema.

2. Bei der zweiten Methode transformiert man die Polynomauswertung auf die Lösung eines äquivalenten linearen Gleichungssystems. Da die Koeffizientenmatrix eine spezielle Toeplitzmatrix ist, erfolgt die Lösung des linearen Systems mit Hilfe einer schnellen iterativen Residuum-Methode, bei der eine Näherungsinverse der Koeffizientenmatrix nicht mehr berechnet werden muß. Das beschriebenen Verfahren liefert i.a. eine maximalgenaue Einschließung des gesuchten Polynomwertes $P_n(x)$. Ausführliche Informationen mit zusätzlichen Literaturangaben findet man in [5, Seite 57], [10, Seite 157].

Falls das Polynomargument x auf der Maschine nicht darstellbar ist, muß es durch ein Maschinenintervall $[x]$ maximalgenau eingeschlossen werden. Wenn man dann nachweisen kann, daß das reelle Polynom $P_n(x)$ auf diesem Intervall monoton ist, so kann man durch Auswertung des Polynoms auf den Randpunkten des Intervalls $[x]$ nach der zweiten Methode eine maximalgenaue Einschließung von $P_n([x])$ berechnen. Eine Beschreibung des von W. Krämer angegebenen Verfahrens findet man z.B. in [10, Seite 167]. Einschließungen, die nach der zweiten Methode oder bei Intervallargumenten nach dem Verfahren von Krämer berechnet werden, lassen sich mit Hilfe der Prozeduren `Polynom(...)` realisieren.

Bei komplexwertigen Polynomen kann die zweite Methode direkt übernommen werden, falls das Argument x als Maschinenzahl vorliegt. Müssen die Polynomkoeffizienten oder das Argument jedoch durch Intervalle eingeschlossen werden, so kann eine Inklusion der Polynomwerte nach der ersten Methode erfolgen, wenn das entsprechende Hornerschema in der zur Verfügung stehenden Intervall-Staggered Arithmetik durchgeführt wird. Zur Vermeidung unnötiger Intervallüberschätzungen müssen dann in der Nähe von Polynom-Nullstellen die Koeffizienten und das Polynomargument in möglichst hoher Genauigkeit eingeschlossen werden.

7.2 Routinen zur Polynomauswertung

7.2.1 Datentypen

Durch das Hilfsmodul `type_def` werden die folgenden Datentypen zur Verfügung gestellt:

type		(rpeval)
RPolynomial	=	**dynamic array** [*] **of** real;
RIPolynomial	=	**dynamic array** [*] **of** interval;
RSPolynomial	=	**dynamic array** [*] **of** Stagg;
type		(rlpeval)
RLPolynomial	=	**dynamic array** [*] **of** Longreal;
RLIPolynomial	=	**dynamic array** [*] **of** Linterval;
RLSPolynomial	=	**dynamic array** [*] **of** LStagg;
type		(cpeval)
CPolynomial	=	**dynamic array** [*] **of** complex;
CIPolynomial	=	**dynamic array** [*] **of** cinterval;
CSPolynomial	=	**dynamic array** [*] **of** Staggcomplex;

7.2.2 Hornerschema

Die folgenden Funktionen `Poly_Horn` liefern Einschließungen des Polynoms

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$$

durch Anwendung des Horner–Schemas für reelle oder komplexe Intervall-Argumente x. Die Koeffizienten a_i können als Maschinenzahlen, Maschinenintervalle oder als Staggered–Intervalle gewählt werden. Wegen der unvermeidbaren Intervallüberschätzungen sind die Einschließungen der $P_n(x)$ auch bei Punktintervallen i.a. nicht maximalgenau.

function `Poly_Horn` (**var** a: ■; x : ◇) : ◇;

Der Indexbereich der Polynomkoeffizienten $a[i]$ muß mit 0 beginnen, sonst erfolgt eine entsprechende Fehlermeldung.

Die folgende Tabelle liefert für ■, ◇ die erlaubten Datentypen und die jeweils exportierenden Module:

Zuordnung von ■, ◇ zu den Datentypen		
`a` : ■	◇	exportierendes Modul

`RPolynomial`
`RIPolynomial`
`RSPolynomial`
	`interval`	`rpeval`

`RLPolynomial`
`RLIPolynomial`
`RLSPolynomial`
	`Linterval`	`rlpeval`

`CPolynomial`
`CIPolynomial`
`CSPolynomial`
	`cinterval`	`cpeval`

7.2.3 Transformation auf ein lineares Problem

Die folgenden Prozeduren `Polynom(...)` liefern mit der Intervallvariablen `z` i.a. eine maximalgenaue Einschließungen des Polynoms

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$$

Der Indexbereich der Polynomkoeffizienten $a[i]$ muß mit 0 beginnen, sonst erfolgt eine entsprechende Fehlermeldung.

Die folgende Prozedur liefert zum Maschinenargument `x` mit der Intervall–Variablen `z` in fast allen Fällen eine maximalgenaue Einschließung von $P_n(x)$:

procedure `Polynom` (**var** `a` : ■; `x` : □;
var `z` : ◇; **var** `Err : integer`);

Zuordnung von $\blacksquare, \square, \diamond$ zu den Datentypen			
`a` : $\blacksquare$	`x` : $\square$	`z` : $\diamond$	exportierendes Modul
`RPolynomial`	`real`	`interval`	`rpeval`
`RLPolynomial`	`Longreal`	`Linterval`	`rlpeval`
`CPolynomial`	`complex`	`cinterval`	`cpeval`

Die folgende Prozedur liefert zu einem Maschinenintervall `x` mit Hilfe der Intervallvariablen `z` bei geeigneter Wahl des `integer`-Parameters `illcd` eine maximalgenaue Einschließung von $P_n(x)$.

procedure `Polynom` (**var** `a` : $\blacksquare$; `illcd : integer`; `x` : $\square$;
var `z` : $\diamond$; **var** `Err : integer`);

Zuordnung von $\blacksquare, \square, \diamond$ zu den Datentypen			
`a` : $\blacksquare$	`x` : $\square$	`z` : $\diamond$	exportierendes Modul
`RPolynomial`	`interval`	`interval`	`rpeval`
`RLPolynomial`	`Linterval`	`Linterval`	`rlpeval`

Mit `illcd=0` wird in einfacher Länge des jeweils benutzten Datenformats gerechnet. Erhält man damit keine maximalgenaue Einschließung, so werden mit `illcd=1` interne Rechnungen in staggered Arithmetik durchgeführt, wobei der Anwender den Genauigkeitsparameter interaktiv bestimmen kann.

Die folgenden Prozeduren ermöglichen auch die Einschließung von $P_n(x)$, wenn die Polynomkoeffizienten `a[i]` Intervalle sind, um beispielsweise den Koeffizienten $a_n = \sqrt{2}$ durch `a[n]` maximalgenau einschließen zu können.

procedure `Polynom` (**var** `a` : $\blacksquare$; `x` : $\square$; **var** `z` : $\diamond$);

Ist der Polynomkoeffizient `a[n]` ein Intervall, so darf `a[n]` die Null nicht enthalten, sonst erfolgt eine entsprechende Fehlermeldung.

Zuordnung von ■,□,◇ zu den Datentypen			
a : ■	x : □	z : ◇	exportiert v. Modul
RIPolynomial	real	interval	rpeval
RIPolynomial	interval	interval	rpeval
RSPolynomial	real	Stagg	rpeval
RSPolynomial	interval	interval	rpeval
RLIPolynomial	Longreal	Linterval	rlpeval
RLIPolynomial	Linterval	Linterval	rlpeval
RLSPolynomial	Longreal	LStagg	rlpeval
RLSPolynomial	Linterval	Linterval	rlpeval
CIPolynomial	complex	cinterval	cpeval
CSPolynomial	complex	Staggcomplex	cpeval
CSPolynomial	Staggcomplex	Staggcomplex	cpeval

7.3 Ein-/Ausgabeprozeduren

Es stehen die Prozeduren

```
procedure read  ( var t : text; var a : ■ );
procedure write ( var t : text; var a : ■ );
```

mit optionalem Fileparameter, beliebig vielen Ein-/Ausgabeparametern, jedoch ohne Formatspezifikationen zur Verfügung.

Zuordnung von ■ zu den erlaubten Datentypen			
a : ■	export. Modul	a : ■	export. Modul
R(L)Polynomial R(L)IPolynomial R(L)SPolynomial	r(l)peval	CPolynomial CIPolynomial CSPolynomial	cpeval

Die Eingabe eines reellen oder komplexen staggered Intervall–Koeffizienten erfolgt dabei interaktiv, wobei der Genauigkeitsparameter `p : integer` vor Vereinbarung der Variablen `a` mit Hilfe der Funktion `set_(L)Stagg_prec(p)` festgelegt werden muß.

7.4 Programmbeispiele

7.4.1 Reelle Polynome mit Punktkoeffizienten

7.4.1.1 Punktargumente

Das nachfolgende Programm `tstrlpev` berechnet für das streng monoton wachsende Polynom

$$P_{13}(x) = \frac{1}{100} \cdot (3x-1)^{13}$$

mit der 13-fachen Nullstelle $x_0 = 1/3$ garantierte Einschließungen der Polynomwerte $P_{13}(x)$, wobei die Punktargumente x vom Anwender einzugeben sind. Zum Vergleich werden dabei die Einschließungen berechnet, die mit dem iterativen Residuum–Verfahren (`procedure Polynom(...)`) bzw. mit dem Intervall–Hornerschema (`function Poly_Horn(..)`) bestimmt wurden.

```
program tstrlpev(input,output);
{ Auswertung des Polynoms  P(x) := 0.01*(3x-1)^13    }
{ x: Longreal wird vom Anwender beliebig vorgegeben. }
use type_def,rlpeval, { Polynom(a,x,z,Err); Poly_Horn(a,z) }
    li_ari; { Intervall-Arithmetik zum Basistyp Longreal;  }

procedure main(n : integer);
var a    : RLPolynomial[0..n];
   x     : Longreal;
   z     : Linterval;
   Err,j : integer;
begin
   { P(x) = a[13]*x^13 +...+a[0] = 0.01*(3x-1)^13 }
   a[0]  := -0.01;     a[1]  := 0.39;       a[2]  := -7.02;
   a[3]  := 77.22;     a[4]  := -579.15;    a[5]  := 3127.41;
   a[6]  := -12509.64; a[7]  := 37528.92;   a[8]  := -84440.07;
   a[9]  := 140733.45; a[10] := -168880.14; a[11] := 138174.66;
   a[12] := -69087.33; a[13] := 15943.23;
   repeat
      writeln('Polynomargument x = ?');  read(x);
      Polynom(a,x,z,Err);
      if Err = 0 then
      begin
```

```
    writeln('Optimale Einschliessung mit Residuum-Iteration:');
    writeln(z);
      end else writeln('Err = ',Err);
      writeln('Polynomauswertung mit Intervall-Hornerschema:');
      z := x;
      writeln(Poly_Horn(a,z));  writeln;
   until false;
end; { main }
begin
   main(13);
end.
```

Mit Hilfe der Eingabe von $x = 0.3$ erhält man den folgenden Programmablauf:

```
Polynomargument x = ?
0.3   <enter>
Optimale Einschliessung mit Residuum-Iteration:
[ -1.00000000000000000000E-015, -1.00000000000000000000E-015 ]
Polynomauswertung mit Intervall-Hornerschema:
[ -1.00000000000000000000E-015, -1.00000000000000000000E-015 ]
```

In folgender Tabelle sind die Ergebnisse für verschiedene Argumente `x : Longreal` zusammengestellt. Beachten Sie bitte, daß die Residuum–Iteration stets die maximalgenaue Einschließung liefert.

Vergleich der mit dem Programm `tstrlpev` berechneten Polynomeinschließungen	
`x = 0.3`	
$P_{13}(x) \in [-1.0 \cdot 10^{-15}, -1.0 \cdot 10^{-15}]$	Residuum–Iteration
$P_{13}(x) \in [-1.0 \cdot 10^{-15}, -1.0 \cdot 10^{-15}]$	Intervall–Hornerschema
`x = 0.33`	
$P_{13}(x) \in [-1.0 \cdot 10^{-28}, -1.0 \cdot 10^{-28}]$	Residuum–Iteration
$P_{13}(x) \in [-1.528 \cdot 10^{-20}, +3.62 \cdot 10^{-20}]$	Intervall–Hornerschema
`x = 0.333`	
$P_{13}(x) \in [-1.0 \cdot 10^{-41}, -1.0 \cdot 10^{-41}]$	Residuum–Iteration
$P_{13}(x) \in [-7.327 \cdot 10^{-20}, +6.56 \cdot 10^{-20}]$	Intervall–Hornerschema
`x = 0.33333333333333333333`	
$P_{13}(x) \in [-1.0 \cdot 10^{-275}, -1.0 \cdot 10^{-275}]$	Residuum–Iteration
$P_{13}(x) \in [-1.0101 \cdot 10^{-19}, +8.94 \cdot 10^{-20}]$	Intervall–Hornerschema

7.4.1.2 Intervallargumente

Das nachfolgende Programm `tstrlpev1` berechnet mit Hilfe der Prozedur `Polynom(a,illcd,x,z,Err)` aus dem Hilfsmodul `rlpeval` bei geeigneter Wahl des integer-Parameters `illcd` i.a. maximalgenaue Einschließungen des Polynoms

$$Q_{13}(x) = 0.01 \cdot (1 + 10^{-13}) \cdot (3x - 1)^{13} = a_{13}x^{13} + \ldots + a_0$$

für Intervallargumente `x : Linterval`. Benutzt wird der von Walter Krämer angegebene Algorithmus [10, Seite 167], der mit Hilfe der Polynomableitungen die Monotonie von $Q_{13}(x)$ im Intervall `x` untersucht. Alle Koeffizienten a_i von Q_{13} sind im Longreal–Format exakt darstellbar; dies gilt jedoch nicht mehr für die Koeffizienten der benötigten Ableitungs–Polynome. In diesem Fall kann man `illcd=1` wählen und damit Teile des Algorithmus in staggered Arithmetik durchführen, wobei der Genauigkeitsparameter vom Anwender interaktiv festgelegt wird. Damit erreicht man u.a., daß die Koeffizienten der Ableitungspolynome optimal durch staggered **Punkt**–Intervalle eingeschlossen werden, was eine **maximalgenaue** Einschließung der Polynomwerte erst ermöglicht. Mit `illcd=0` werden alle Rechnungen nur im `Longreal`–Format durchgeführt, und die Koeffizienten der Ableitungspolynome können dann nicht mehr durch Punktintervalle eingeschlossen werden. Für Argumentintervalle `x` in der Nähe der Nullstelle $x_0 = 1/3$ kann deshalb wegen `Err<>0` eine Einschließung der Polynomwerte nicht mehr berechnet werden. Zum Vergleich wird noch eine sehr grobe Einschließung der gleichen Polynomwerte mit Hilfe des Intervall–Hornerschemas durch die Funktion `Poly_Horn(a,x)` angegeben.

```
program tstrlpev1(input,output);
{ Einschliessung von Q(x) := 0.01*(1+10^(-13))*(3x-1)^(13) }
{ x: Linterval wird vom Anwender beliebig vorgegeben.      }
use type_def, lstagg,
    rlpeval , { Polynom(a,illcd,x,z,Err);  Poly_Horn(a,x); }
    li_ari  ; { Intervall-Arithmetik bzgl. type  Longreal; }

procedure main(n : integer);
var a    : RLPolynomial [0..n];     { Polynom-Koeffizienten }
   x     : Linterval;               { Intervall-Argument    }
   z     : Linterval;               { Einschliessg. v. Q(x) }
   Err   : integer;
```

```
begin    { Q(x) = a[13]*x^13 + a[12]*x^12 + ... + a[0] }
   a[0]  := Long(-0.01)      - 1e-15;
   a[1]  := Long( 0.39)      + 3.9e-14;
   a[2]  := Long(-7.02)      - 7.02e-13;
   a[3]  := Long( 77.22)     + 7.722e-12;
   a[4]  := Long(-579.15)    - 5.7915e-11;
   a[5]  := Long( 3127.41)   + 3.12741e-10;
   a[6]  := Long(-12509.64)  - 1.250964e-9;
   a[7]  := Long( 37528.92)  + 3.752892e-9;
   a[8]  := Long(-84440.07)  - 8.444007e-9;
   a[9]  := Long( 140733.45) + 1.4073345e-8;
   a[10] := Long(-168880.14) - 1.6888014e-8;
   a[11] := Long( 138174.66) + 1.3817466e-8;
   a[12] := Long(-69087.33)  - 6.908733e-9;
   a[13] := Long( 15943.23)  + 1.594323e-9;

   repeat
     writeln('Intervall-Argument x = ?');  read(x);
     Polynom(a,0,x,z,Err);
     write('Polynom-Einschliessung mit Polynom(a,0,x,z,Err),');
     writeln(' d.h. illcd=0 :');
     if Err = 0 then writeln(z) else writeln('Err = ',Err);
     writeln;
     write('Polynom-Einschliessung mit Polynom(a,1,x,z,Err),');
     writeln(' d.h. illcd=1 :');
     Polynom(a,1,x,z,Err);
     if Err = 0 then writeln(z) else writeln('Err = ',Err);
     writeln;
     writeln('Einschliessung mit Intervall-Hornerschema:');
     writeln(Poly_Horn(a,x));
     writeln('---------------------------------------------');
   until false;

end; { main }

begin
   main(13);
end.
```

Mit dem Intervallargument

$$[\ 0.33333333333333333332, 0.33333333333333333333\ ]$$

ganz in der Nähe der Nullstelle $x_0 = 1/3$ liefert das Programm `tstlrpev1` für `illcd=1` die folgende Bildschirmausgabe:

```
Intervall-Argument x = ?
[ 0.33333333333333333332, 0.33333333333333333333 ]    <enter>
Polynom-Einschliessung mit Polynom(a,0,x,z,Err), d.h. illcd=0 :
Err = 1

Polynom-Einschliessung mit Polynom(a,1,x,z,Err), d.h. illcd=1 :
     ***********************************************
     *  Wie lautet der Genauigkeitsparameter p = ?  *
     *  Zu kleine  p >= 1  liefern keine optimalen  *
     *  Einschliessungen der Polynomwerte!          *
     ***********************************************
11   <enter>
[ -6.7108864000067108864E-268, -1.0000000000010000000E-275 ]

Einschliessung mit Intervall-Hornerschema:
[ -1.5110000000000000000E-019,  1.3400000000000000000E-019 ]
--------------------------------------------------------------
```

Mit `illcd=1` erhält man durch interne Intervall–Staggered–Arithmetik für das obige Intervallargument `x` in unmittelbarer Nähe der Nullstelle $x_0 = 1/3$ sogar eine **maximalgenaue** und **garantierte** Einschließung der Polynomwerte $Q_{13}(x)$.

7.4.2 Reelle Polynome mit Intervallkoeffizienten

Ist ein Koeffizient, z.B. $a_0 = -1/7$, des Polynoms

$$P_{13}(x) = \frac{1}{7} \cdot (3x-1)^{13} = a_{13}x^{13} + a_{12}x^{12} + \ldots + a_0$$

im benutzten Longreal–Format nicht darstellbar, so müssen a_0 und alle anderen Koeffizienten jeweils durch ein Intervall maximalgenau eingeschlossen werden, wenn $P_{13}(x)$ für ein Punkt- oder Intervall–Argument erfolgreich eingeschlossen werden soll. Wir wählen:

$a_i \in$ `b[i]` : `RLIPolynomial[0..13]`;
$a_i \in$ `c[i]` : `RLSPolynomial[0..13, -1..LSTagg_prec]`;

und benutzen die Prozeduren

`Polynom(b,x,z);`	vergl. Seite 335
`Polynom(c,x,z);`	vergl. Seite 335

wobei die Ausgangsgröße `z : Linterval` die reellen Polynomwerte $P_{13}(x)$ garantiert einschließt. Das Eingangsintervall `x : Linterval` enthält die reellen Argumente x und wird vom Anwender im nachfolgenden Programm `tstrlpev2` beliebig vorgegeben. Es gilt somit:

$$x \in \texttt{x : Linterval} \quad \Longrightarrow \quad P_{13}(x) \in \texttt{z : Linterval}$$

Die obigen Prozeduren `Polynom(..)` benutzen den von W. Krämer angegebenen Algorithmus, mit dem die Monotonie des Polynoms über dem Eingangsintervall `x` untersucht wird und die Einschließung der Werte $P_{13}(x)$ durch Intervall–Auswertung des Polynoms an den Randpunkten `x.inf` und `x.sup` erfolgt. Ist die Monotonie nicht nachweisbar, so wird das grobe Intervall–Hornerschema angewandt. Da die Einschließung der Polynomwerte $P_{13}(x)$ in der Umgebung der 13-fachen Nullstelle $x_0 = 1/3$ besonders problematisch ist, können die auf der Maschine nicht darstellbaren Polynomkoeffizienten a_i durch die Staggered Intervalle `c[i]` um so enger eingeschlossen werde, je größer der Präzisionsparameter `LStagg_prec` vom Anwender gewählt wird.

```
program tstrlpev2(input,output);
{ Auswertung des Polynoms  P(x) := (1/7)*(3x-1)^(13) }
{ x: Linterval wird vom Anwender beliebig vorgegeben }
use rlpeval, li_ari, type_def, lstagg;
var n,           { Polynomgrad;    In diesem Beispiel: n = 13 }
    p : integer; { Genauigkeitsparameter f. Stagg. Arithmetik }
procedure main(n,p : integer);
var b    : RLIPolynomial[0..n];
   c     : RLSPolynomial[0..n,-1..p];
   x,z   : Linterval;
   Err,k : integer;
begin    { Einschliessung der Koeffizienten von (3x-1)^(13) }
         { zunaechst mit dem Daten-Typ:  RLIPolynomial:     }
   b[0]  := -Long(1);         b[1]  :=  Long(39);
   b[2]  := -Long(702);       b[3]  :=  Long(7722);
   b[4]  := -Long(57915);     b[5]  :=  Long(312741);
   b[6]  := -Long(1250964);   b[7]  :=  Long(3752892);
   b[8]  := -Long(8444007);   b[9]  :=  Long(14073345);
```

```
   b[10] := -Long(16888014);  b[11] :=  Long(13817466);
   b[12] := -Long(6908733);   b[13] :=  Long(1594323);
   For k := 0 to n do
   begin    { Einschliessung der Koeffz. durch RLSPolynomial: }
      c[k] := b[k];
      c[k] := c[k] / LONG(7);
   end;
   For k:=0 to n do b[k] := b[k]/7;
   repeat
      writeln('Intervall-Argument x = ?');
      read(x);
      write  ('Koeffizienten-Einschliessung mit dem Typ ');
      writeln('RLIPolynomial;');
      writeln('P(x) enthalten in:');
      Polynom(b,x,z);
      writeln(z);   writeln;
      write  ('Koeffizienten-Einschliessung mit dem Typ ');
      writeln('RLSPolynomial;');
      writeln('P(x) enthalten in:');
      Polynom(c,x,z);
      writeln(z);
      writeln('---------------------------------------------');
   until false;
end; { main }

begin
   writeln('Polynom-Beispiel:      P(x) = (1/7)*(3x-1)^13');
   write  ('Zu gegebenem x : Linterval werden ');
   writeln('Einschliessungen fuer P(x) berechnet.');
   write  ('Wie lautet zur Staggered Arithmetik der ');
   writeln('Genauigkeitsparameter p >= 1 ?');     read(p);
   set_LStagg_prec(p); { Legt LStagg_prec in type_def fest }
   main(13,p);
end.
```

Das Programms `tstrlpev2` liefert mit den durch `<enter>` abgeschlossenen Eingaben die folgenden Ergebnisse:

```
Polynom-Beispiel:    P(x) = (1/7)*(3x-1)^13
Zu gegebenem x : Linterval werden Einschliessungen fuer P(x)
```

```
berechnet.
Wie lautet zur Staggered Arithmetik der
Genauigkeitsparameter p >= 1 ?
14    <enter>
Intervall-Argument x = ?
0.33333333333333333332   <enter>
Koeffizienteneinschliessung mit dem Typ RLIPolynomial;
P(x) enthalten in:
[ -3.66100000000000000000E-018,  3.80500000000000000000E-018 ]
Koeffizienteneinschliessung mit dem Typ RLSPolynomial;
P(x) enthalten in:
[ -9.58698057142857142858E-267, -9.58698057142857142857E-267 ]
---------------------------------------------------------------
Intervall-Argument x = ?
0.33333333333333333333   <enter>
Koeffizienteneinschliessung mit dem Typ RLIPolynomial;
P(x) enthalten in:
[ -3.48700000000000000000E-018,  3.83200000000000000000E-018 ]
Koeffizienteneinschliessung mit dem Typ RLSPolynomial;
P(x) enthalten in:
[ -1.42857142857142857143E-274, -1.42857142857142857142E-274 ]
---------------------------------------------------------------
Intervall-Argument x = ?
[0.33333333333333333332, 0.33333333333333333333]     <enter>
Koeffizienteneinschliessung mit dem Typ RLIPolynomial;
P(x) enthalten in:
[ -4.14400000000000000000E-018,  4.79500000000000000000E-018 ]
Koeffizienteneinschliessung mit dem Typ RLSPolynomial;
P(x) enthalten in:
[ -9.58698057142857142858E-267, -1.42857142857142857142E-274 ]
---------------------------------------------------------------
Intervall-Argument x = ?
[12, 12.0000000000000000001]                         <enter>
Koeffizienteneinschliessung mit dem Typ RLIPolynomial;
P(x) enthalten in:
[  1.68961025402832031249E+019,  1.68961025402832031270E+019 ]
Koeffizienteneinschliessung mit dem Typ RLSPolynomial;
P(x) enthalten in:
[  1.68961025402832031249E+019,  1.68961025402832031269E+019 ]
---------------------------------------------------------------
```

Die drei ersten Beispiele zeigen, daß eine maximalgenaue Einschließung der Polynomwerte für Intervallargumente `x` in der unmittelbaren Umgebung der Nullstelle $x_0 = 1/3$ nur möglich ist, wenn die Polynomkoeffizienten a_i durch Staggered Intervalle in hoher Präzision (`LStagg_prec=14`) eingeschlossen werden, während im letzten Beispiel die viel gröberen Einschließungen der Koeffizienten durch `b[i] : RLIPolynomial[1..13]` für Intervallargumente `x` mit größerem Nullstellenabstand schon optimale Ergebnisse liefern.

Hinweis:
Müssen die Polynomkoeffizienten wie im obigen Beispiel durch Intervalle eingeschlossen werden und ist das Polynomargument x eine Maschinenzahl, so sollte man aus Laufzeitgründen nicht mit Punktintervall–Argumenten rechnen, sondern die entsprechenden Prozeduren `Polynom(...)` mit `real`- bzw. `Longreal`–Argumenten benutzen, vergl. Seite 335.

7.4.3 Komplexe Polynome mit Punktkoeffizienten

Das folgende Programm `tstcpev1` liefert garantierte und i.a. maximalgenaue Einschließungen des komplexen Polynoms

$$\begin{aligned} P_8(x) &= (x-i)^8 \\ &= x^8 - 8ix^7 - 28x^6 + 56ix^5 + 70x^4 - 56ix^3 - 28x^2 + 8ix + 1 \end{aligned}$$

für punktförmige, komplexe Maschinenargumente x.

```
program tstcpev1(input,output);
{ Auswertung des Polynoms  P(x) := (x-i)^8           }
{ x : complex wird vom Anwender beliebig vorgegeben.  }
use type_def,cpeval, { Prozedur Polynom(a,x,z,Err)    }
    c_ari , { Komplexe Arithmetik,           read(x)  }
    ci_ari; { Kompl. Intervall-Arithmetik, writeln(z) }
procedure main(n : integer);
var a  : CPolynomial [0..n];     { n: Polynomgrad = 8 }
   x   : complex;           { komplexes Punktargument }
   z   : cinterval;         { Einschliessg. von  P(x) }
   Err : integer;           { Err:         Fehler-Kode }
begin  { Polynomkoeffizienten von:  P(x) = (x-i)^8    }
   a[0].re := 1;   a[0].im := 0;  a[1].re := 0; a[1].im := 8;
   a[2].re := -28; a[2].im := 0;  a[3].re := 0; a[3].im := -56;
   a[4].re := 70;  a[4].im := 0;  a[5].re := 0; a[5].im := +56;
   a[6].re := -28; a[6].im := 0;  a[7].re := 0; a[7].im := -8;
```

```
   a[8].re := 1;   a[8].im := 0;
   repeat
      writeln('Komplexes Polynomargument x = ?');  read(x);
      Polynom(a,x,z,Err);
      if Err = 0 then
      begin
         writeln('P(x) ist enthalten in:');
         writeln(z);
      end        else writeln('Err = ',Err);
   until false;
end; { main }
begin
   main(8);
end.
```

Mit den entsprechenden Eingaben, die mit `<enter>` abgeschlossen werden, liefert das obige Programm die folgenden Ergebnisse:

```
Komplexes Polynomargument x = ?
(1e-170,1.000000000001)   <enter>
P(x) ist enthalten in:
( [  9.999999999999E-097,  1.000000000001E-096 ],
  [ -8.000000000001E-254, -7.999999999999E-254 ] )

Komplexes Polynomargument x = ?
(1e-33,1)                 <enter>
P(x) ist enthalten in:
( [  1.000000000000E-264,  1.000000000000E-264 ],
  [ -0.000000000000E+000,  0.000000000000E+000 ] )

Komplexes Polynomargument x = ?
(2,3)                     <enter>
P(x) ist enthalten in:
( [  4.096000000000E+003,  4.096000000000E+003 ],
  [ -0.000000000000E+000,  0.000000000000E+000 ] )
```

Auch in unmittelbarer Nähe der 8-fachen Nullstelle $x_0 = i$ erhält man noch maximalgenaue Einschließungen der komplexen Polynomwerte.

7.4.4 Komplexe Polynome mit Intervallkoeffizienten

Die Koeffizienten des komplexen Polynoms

$$\begin{aligned} Q_8(x) &= \frac{1}{3}\cdot(x-i)^8 \\ &= \frac{1}{3}\cdot x^8 - \frac{8i}{3}\cdot x^7 - \frac{28}{3}\cdot x^6 + \frac{56i}{3}\cdot x^5 + \frac{70}{3}\cdot x^4 - \frac{56i}{3}\cdot x^3 \\ &\quad -\frac{28}{3}\cdot x^2 + \frac{8i}{3}\cdot x + \frac{1}{3} \end{aligned}$$

sind auf der Maschine nicht darstellbar und müssen daher durch Intervalle möglichst eng eingeschlossen werden. Zur Verfügung steht dafür die Prozedur `Polynom(...)` von Seite 335, mit der man komplexe Staggered Intervalle für $Q_8(x)$ erhält, wenn man sowohl die komplexen Polynomkoeffizienten als auch das komplexe Argument x durch entsprechende komplexe Staggered Intervalle einschließt. $Q_8(x)$ soll für das Argument $x_1 = (10^{-30}\sqrt{2} + i)$ ausgewertet werden, wobei x_1 auf der Maschine nicht darstellbar ist und daher im folgenden Programm `tstcpev2` durch das komplexe Staggered Intervall `s` eingeschlossen wird. Um den Wert $Q_8(x_1) = \frac{16}{3}\cdot 10^{-240} + 0\cdot i$ möglichst gut einzuschließen, kann der Anwender den Präzisionsparameter `p = Stagg_prec` hinreichend groß wählen.

```
program tstcpev2(input,output);
{ Auswertung des Polynoms   Q_8(x) := (1/3)*(x-i)^8  }
{ an der komplexen Stelle: x = (1e-30 * sqrt(2) + i) }
use cpeval, type_def, ci_ari, stagg, cstagg;
var p               : integer;
procedure main(n,p : integer);
var a   : CPolynomial [0..n];
    b   : CSPolynomial[0..n,1..2,-1..p];
    zz  : cinterval;
    k   : integer;
    s,z : Staggcomplex[1..2,-1..p];
begin
   a[0].re := 1;   a[0].im := 0;  a[1].re := 0; a[1].im := 8;
   a[2].re := -28; a[2].im := 0;  a[3].re := 0; a[3].im := -56;
   a[4].re := 70;  a[4].im := 0;  a[5].re := 0; a[5].im := +56;
   a[6].re := -28; a[6].im := 0;  a[7].re := 0; a[7].im := -8;
   a[8].re := 1;   a[8].im := 0;
   For k := 0 to n do
   begin
```

```
      b[k] := a[k];  b[k] := b[k]/3;
   end;
   s[1] := 2; s[1] := sqrt(s[1])*1e-30;
   s[2] := 1;
   { s: Einschliessung von  (1e-30 * sqrt(2) + i) }
   Polynom(b,s,z); { z: Einschliessung von Q_8(x) }
   zz := z;        { Rundg. von z ins real-Format }
   writeln(zz);
end; { main }
begin
   repeat
      write  ('Welcher Praezisionsparameter p >= 1 ');
      writeln('wird gewuenscht ?');
      read(p);  set_Stagg_prec(p);
      main(8,p);  writeln;
   until false;
end.
```

Für die Präzisionsparameter `p = 17,18,19` liefert das obige Programm die folgende Bildschirmausgabe:

```
Welcher Praezisionsparameter p >= 1 wird gewuenscht ?
17      <enter>
Q_8(x_1) wird eingeschlossen durch:
( [ -1.981800000007E-232,  2.458300000001E-232 ],
  [ -6.536941500000E-260,  6.500192600000E-260 ] )

Welcher Praezisionsparameter p >= 1 wird gewuenscht ?
18      <enter>
Q_8(x_1) wird eingeschlossen durch:
( [  5.333313481999E-240,  5.333357883001E-240 ],
  [ -3.100000000000E-266,  4.600000000000E-266 ] )

Welcher Praezisionsparameter p >= 1 wird gewuenscht ?
19      <enter>
Q_8(x_1) wird eingeschlossen durch:
( [  5.333333333333E-240,  5.333333333334E-240 ],
  [ -3.100000000000E-266,  4.600000000000E-266 ] )
```

Die Einschließung des Realteils von $Q_8(x_1) = \frac{16}{3} \cdot 10^{-240} + 0 \cdot i$ erfolgt mit `p=19` also maximalgenau, während der verschwindende Imaginärteil etwas gröber eingeschlossen wird.

Kapitel 8

Automatische Differentiation

Die Berechnung von Ableitungen reeller oder komplexer Funktionen in einer oder mehreren Variablen ist ein zentrale Aufgabe der Numerik. Dabei ist es z.B. bei Restgliedabschätzungen von großem Interesse, Ableitungen auch höherer Ordnung intervallmäßig einschließen zu können.

Bei symbolischer Differentiation berechnet man die gewünschte Ableitung zunächst formelmäßig, um danach den entsprechenden Funktionsterm numerisch auszuwerten. Zuverlässige Ergebnisse erhält man dabei jedoch nur, wenn die Auswertung der Ableitungsterme **intervallmäßig** durchgeführt wird. In **PASCAL-XSC** lassen sich bei der Automatischen Differentiation beide Schritte sehr effektiv kombinieren, d.h. der Wert einer Ableitung wird garantiert eingeschlossen, indem die mit den Ableitungsregeln (z.B. Produkt- oder Quotientenregel) gelieferten Terme intervallmäßig ausgewertet werden. Nach Bereitstellung der notwendigen Intervall–Standardfunktionen, die in der BCD-Version von **PASCAL-XSC** in großer Zahl zur Verfügung stehen, lassen sich dann höhere Ableitungen auch stark verschachtelter Funktionen bequem und vor allem zuverlässig berechnen. Die eigentliche Auswertung der durch die Ableitungsregeln gegebenen Terme erfolgt mit Hilfe des Datentyps `rrGinterval`, so daß die Ableitungswerte mit etwa 24 richtigen Dezimalziffern berechnet und deshalb i.a. maximalgenau nach `Linterval` oder `interval` gerundet werden können.

Wenn in Ausnahmefällen die gewünschten Ableitungswerte wegen der bekannten Intervallüberschätzungen zu grob eingeschlossen werden, bringt Abhilfe oft nur der Einsatz einer Staggered Arithmetik, die alledings die entsprechenden Intervall-Standardfunktionen im Staggered–Format benötigt.

Ausführliche Informationen und weitere Literaturangaben zur Automatischen Differentiation und zur Taylor–Arithmetik findet man in [5],[8].

8.1 Reelle Funktionen mit einer Variablen

8.1.1 Ableitungen bis zur zweiten Ordnung

Bei vielen numerischen Anwendungen ist es völlig ausreichend, Ableitungen einer differenzierbaren Funktion nur bis zur zweiten Ordnung zu berechnen. Benötigt wird dazu der vom Modul `ddf_rrg` exportierte Datentyp

```
type DerivType = record
                   f, df, ddf : rrGinterval;
                 end;                          (ddf_rrg)
```

Eine zu differenzierende Funktion $f(x)$ ist wie folgt zu vereinbaren:

```
function f(x : DerivType) : DerivType;
begin
   f := DerivType-Ausdruck;
end;
```

Der `DerivType-Ausdruck` wird realisiert mit Hilfe der vom Modul `ddf_rrg` exportierten Operatoren $\boxed{+, -, *, /}$ mit dem Ergebnistyp `DerivType`, wobei zunächst beide Operanden vom Typ `DerivType` sein dürfen. Ist jedoch nur ein Operand vom Typ `DerivType`, so darf der andere Operand die folgenden Datentypen annehmen:

```
integer, real, Longreal, rrG, rrGinterval
```

Zusammen mit den auf der folgenden Seite aufgelisteten Standardfunktionen vom Typ `DerivType` lassen sich damit praktisch alle Funktionen realisieren, die aus endlich vielen Grundoperationen und Standardfunktionen aufgebaut sind. Ist z.B. $f(x) = \sqrt{5} \cdot e^{1/(1+x^2)}$ die zu differenzierende Funktion, so kann diese in einem **PASCAL-XSC** Programm in der folgenden sehr gut lesbaren Art vereinbart werden:

```
function f (x : DerivType) : DerivType;
begin
   f := sqrt( rrGintval(5) ) * exp( 1/(1+sqr(x)) )
end;
```

Beachten Sie bitte, daß durch `sqrt( rrGintval(5) )` der Wert $\sqrt{5}$ wirklich eingeschlossen wird, während z.B. `sqrt(5)` den Wurzelwert nur im real–Format approximieren würde!

Von `ddf_rrg` exportierte Funktionen vom Typ `DerivType`			
`x,y : DerivType; a : rrGinterval; n : integer; n = 0,1,...`			
Funktion	**Aufruf**	**Funktion**	**Aufruf**
x^2	`sqr(x)`	$\cos(x)$	`cos(x)`
x^n	`power(x,n)`	$\tan(x)$	`tan(x)`
x^x	`power(x)`	$\cot(x)$	`cot(x)`
x^y	`power(x,y)`	$\sin(\pi \cdot x)$	`sinpi(x)`
$\sqrt{x}$	`sqrt(x)`	$\cos(\pi \cdot x)$	`cospi(x)`
$\sqrt{1+x}-1$	`sqrt1pm1(x)`	$\tan(\pi \cdot x)$	`tanpi(x)`
$\sqrt{1-x^2}$	`sqrt1mx2(x)`	$\cot(\pi \cdot x)$	`cotpi(x)`
$\sqrt{x^2-1}$	`sqrtx2m1(x)`	$\arcsin(x)$	`arcsin(x)`
$\sqrt{1+x^2}$	`sqrt1px2(x)`	$\arccos(x)$	`arccos(x)`
e^x	`exp(x)`	$\arctan(x)$	`arctan(x)`
e^x-1	`expm1(x)`	$\mathrm{arccot}(x)$	`arccot(x)`
e^{-x^2}	`exp_x2(x)`	$\sinh(x)$	`sinh(x)`
2^x	`exp2(x)`	$\cosh(x)$	`cosh(x)`
10^x	`exp10(x)`	$\tanh(x)$	`tanh(x)`
$\ln(x)$	`ln(x)`	$\coth(x)$	`coth(x)`
$\ln(e \cdot x)$	`ln_ex(x)`	$\mathrm{arsinh}(x)$	`arsinh(x)`
$\ln(1+x)$	`ln1p(x)`	$\mathrm{arcosh}(x)$	`arcosh(x)`
$\log_a(x)$	`loga(a,x)`	$\mathrm{arcosh}(1+x)$	`arcosh1p(x)`
$\log_2(x)$	`log2(x)`	$\mathrm{artanh}(x)$	`artanh(x)`
$\log_{10}(x)$	`log10(x)`	$\mathrm{arcoth}(x)$	`arcoth(x)`
$\log_{10}(1+x)$	`log1p(x)`		
$\sin(x)$	`sin(x)`		

Nach der Vereinbarung der zu differenzierenden Funktion $f(x)$ gibt es für die Datentypen der zu berechnenden Funktionswerte $f(x)$, $f'(x)$, $f''(x)$ drei Möglichkeiten:

1. Ergebnis–Typ von $f(x)$, $f'(x)$, $f''(x)$: `rrGinterval`
2. Ergebnis–Typ von $f(x)$, $f'(x)$, $f''(x)$: `Linterval`
3. Ergebnis–Typ von $f(x)$, $f'(x)$, $f''(x)$: `interval`

8.1.1.1 $f(x)$, $f'(x)$, $f''(x)$ vom Typ `rrGinterval`

Das Modul `ddf_rrg` stellt zur Berechnung von $f(x)$, $f'(x)$, $f''(x)$ die folgenden Funktionen zur Verfügung:

Berechnung von $f(x)$ mit:

```
function fValue( y : DerivType ) : rrGinterval;
```

Berechnung von $f'(x)$ mit:

```
function dfValue( y : DerivType ) : rrGinterval;
```

Berechnung von $f''(x)$ mit:

```
function ddfValue( y : DerivType ) : rrGinterval;
```

Als Eingangsparameter `y : DerivType` ist dann jeweils die vorher, wie auf Seite 350 vereinbarte Funktion `f(u)` zu wählen, wobei das Argument `u` vom Typ `DerivType` sein muß. Zur Umwandlung des Datentyps des vom Anwender vorgegebenen Intervalls `x` in den Datentyp `DerivType` steht die folgende Funktion zur Verfügung:

```
function DerivVar( v : ■ ) : DerivType;   (ddf_ rrg)
```

wobei ■ durch die Datentypen `rrG, rrGinterval, Linterval, interval` ersetzt werden kann.

Das nachfolgende Programm `tstddf_rrg1` berechnet zum vorgegebenen Intervall `x : Linterval` Einschließungen von $f(x)$, $f'(x)$, $f''(x)$, wobei $f(x)$ definiert ist durch $f(x) = \sqrt{5} \cdot e^{1/(1+x^2)}$ und die Einschließungen vom Typ `rrGinterval` geliefert werden:

```
program tstddf_rrg1(input,output);
{ Einschliessung der 0. 1. 2. Ableitung von f(x) }
{ mit dem Typ rrGinterval;        x : Linterval; }
use li_ari  , { exportiert read(x)              }
    ddf_rrg , { DerivVar(x); fValue(f(u)); ....  }
    rrgi_ari; { writeln(fValue(f(u))); ......... }

function f(x : DerivType): DerivType;
begin
   f := sqrt( rrGintval(5) )*exp( 1/(1+sqr(x)) );
end; { f }

var x : Linterval;
   u  : DerivType;
begin
   writeln('f(x) = sqrt(5) * exp( 1/(1+x^2) )');
   writeln('Berechnung von f(x) mit 1. u. 2. Ableitung:');
   repeat
      writeln('x : Linterval = ?');  read(x);
      u := DerivVar(x); { Linterval ---> DerivType }
      writeln('Einschliessung von f(x):');
      writeln(fValue(f(u)));
      writeln('Einschliessung der 1. Ableitung:');
      writeln(dfValue(f(u)));
      writeln('Einschliessung der 2. Ableitung:');
      writeln(ddfValue(f(u)));  writeln;
   until false;
end.
```

Mit dem Punktintervall $x = [0, 0]$ erhält man die Ergebnisse:

```
f(x) = sqrt(5) * exp( 1/(1+x^2) )
Berechnung von f(x) mit 1. u. 2. Ableitung:
x : Linterval = ?
0   <enter>
```

```
Einschliessung von f(x):
[ {  6.078262950437E-001,  -1.524434163389E-014,    1 },
  {  6.078262950437E-001,  -1.524434162893E-014,    1 } ]
Einschliessung der 1. Ableitung:
[ {  0.000000000000E+000,   0.000000000000E+000, -2147483647 },
  {  0.000000000000E+000,   0.000000000000E+000, -2147483647 }]
Einschliessung der 2. Ableitung:
[ { -1.215652590087E-001,  -3.695113167481E-014,    2 },
  { -1.215652590087E-001,  -3.695113167262E-014,    2 } ]
```

8.1.1.2 $f(x)$, $f'(x)$, $f''(x)$ vom Typ `Linterval`

Wenn man $f(x)$, $f'(x)$, $f''(x)$ wie im vorigen Beispiel durch Intervalle vom Typ `rrGintervall` einschließt, so kann man diese Ergebnisse i.a. maximalgenau ins gröbere `Linterval`–Format runden. Genau diese Vorgehensweise wird realisiert beim Aufruf der Prozeduren:

Einschließung von $f(x)$:

```
procedure fEval ( function f(x:DerivType) : DerivType;
(ddf_rrg)           x : Linterval; var fx : Linterval );
```

Einschließung von $f(x)$, $f'(x)$:

```
procedure dfEval ( function f(x:DerivType) : DerivType;
                   x : Linterval;
(ddf_rrg)          var fx, dfx : Linterval );
```

Einschließung von $f(x)$, $f'(x)$, $f''(x)$:

```
procedure ddfEval ( function f(x:DerivType) : DerivType;
                    x : Linterval;
(ddf_rrg)           var fx, dfx, ddfx : Linterval );
```

Es gilt: $f(x) \in$ `fx`, $f'(x) \in$ `dfx`, $f''(x) \in$ `ddfx`.
function `f(x : DerivType) : DerivType` ist dabei so zu deklarieren, wie

es auf Seite 350 beschrieben wurde. Bitte beachten Sie, daß das Intervall–Argument `x : Linterval` hier vom Anwender nicht in den Typ `DerivType` umzuwandeln ist, sondern daß dies in der Prozedur intern erledigt wird.

8.1.1.2.1 Wahl des richtigen Funktionsterms

Als Anwendung betrachten wir die Funktionen

$$f(x) = x - \ln(1 + e^{x}) \tag{8.1}$$

$$g(x) = -\ln(1 + e^{-x}) \equiv f(x), \tag{8.2}$$

die wegen $x \equiv \ln(e^x)$ für $x \in I\!R$ mathematisch identisch sind. Wendet man jedoch auf f und g die automatische Differentiation an, so erhält man mit den gleichen Argument–Intervallen `x` für die Einschließungen der Funktionswerte und Ableitungen von f bzw. g extrem unterschiedliche Intervalldurchmesser:

- Mit $x = +10^3$ erhält man für $f(x)$, $f'(x)$, $f''(x)$ so grobe Einschließungen, daß noch nicht einmal das richtige Vorzeichen bestimmt ist, während für $g(x)$, $g'(x)$, $g''(x)$ sogar maximalgenaue Einschließungen berechnet werden.

- Andererseits erhält man mit $x = -10^3$ für $f(x)$, $f'(x)$, $f''(x)$ sogar maximalgenaue Einschließungen, während die Einschließung für $g''(x)$ völlig unbrauchbar ausfällt.

Die entsprechenden Rechenergebnisse findet man ab Seite 358.

> Es gibt Funktionen, bei denen man für die automatische Differentiation in verschiedenen Argumentbereichen auch verschiedene, aber mathematisch identische Funktionsterme benutzen sollte.

Um nun $f(x)$, $f'(x)$, $f''(x)$ für möglichst alle Intervall–Argumente `x` maximalgenau einzuschließen, definieren wir die Funktion $h(x) \equiv f(x)$, indem Funktionswert und Ableitungen **direkt** berechnet werden und nicht durch die Automatische Differentiation. Dazu schreiben wir f als Funktion von u, wobei $u = u(x)$ zweimal nach x differenzierbar sein soll:

$$\begin{aligned}
f(u(x)) &= -\ln(1 + e^{-u(x)}) \\
\frac{d}{dx} f(u) &= \frac{u'(x)}{1 + e^{u(x)}} \\
\frac{d^2}{dx^2} f(u) &= \frac{1}{1 + e^{u}} \left[u'' - \frac{(u')^2}{1 + e^{-u}} \right]
\end{aligned}$$

Für die Funktion $h(u) \equiv f(u)$ ergibt sich dann aus den obigen Formeln die folgende Deklaration:

```
function h(u : DerivType): DerivType;
var y : rrGinterval;
begin
   h.f := -ln1p( exp(-u.f) );      { u.f   = u(x)   }
   y := 1/(1+exp(u.f));
   h.df := u.df * y;               { u.df  = u'(x)  }
   h.ddf := y * ( u.ddf - sqr(u.df)/(1+exp(-u.f)) );
end; { h }                         { u.ddf = u''(x) }
```

Nach diesem Beispiel sind alle in der Tabelle von Seite 351 angegebenen Standardfunktionen vom Typ `DerivType` im Modul `ddf_rrg` realisiert. Dabei ist darauf zu achten, daß die Formeln für die Ableitungen möglichst einfach dargestellt werden und dabei die Variable `u` mit ihren Ableitungen möglichst wenig benutzt wird, um unnötige Intervallüberschätzungen zu vermeiden.

Das nachfolgende Programm `tstddf_rrg2` ermöglicht die Eingabe eines Intervalls `x : Linterval` und berechnet Einschließungen des Funktionswertes und der beiden ersten Ableitungen für die beiden identischen Funktionen $f(x), g(x)$ nach (8.1),(8.2) und für die mit $f(x)$ identischen Funktion $h(x)$, wobei $h(x)$ als Funktion vom Typ `DerivType` nach obigem Beispiel extra deklariert wird.

```
program tstddf_rrg2(input,output);
{ Zu gegebenem Argumentintervall x werden berechnet:      }
{ Einschliessungen fuer Funktionswert, 1. u. 2. Ableitung }
{ der identischen Funktionen:      f(x)=g(x)=h(x);        }
{ f(x) = x - ln[1+exp(x)];     g(x) = -ln[1+exp(-x)]      }
{ f(x) = h(x): DerivType   wird eigens deklariert, um     }
{ unnoetige Intervall-Ueberschaetzungen zu vermeiden.     }

use li_ari, ddf_rrg, rrgi_util, rrgi_ari;

function f(x : DerivType): DerivType;
begin
   f := x - ln1p( exp(x) );
end; { f }
function g(x : DerivType): DerivType;
begin
   g := -ln1p( exp(-x) );
```

```
end; { g }
function h(x : DerivType): DerivType;
var y : rrGinterval;
begin
   h.f := -ln1p( exp(-x.f) );
   y := 1/(1+exp(x.f));
   h.df := x.df * y;
   h.ddf := y * ( x.ddf - sqr(x.df)/(1+exp(-x.f)) );
end; { h }
var x,              { Argumentintervall                 }
   fx,              { Einschliessung des Funktionswertes }
   dfx,             { Einschliessung der 1. Ableitung   }
   ddfx : Linterval; { Einschliessung der 2. Ableitung   }
begin
   repeat
      writeln('x : Linterval = ?');  read(x);
      ddfEval(f,x,fx,dfx,ddfx);
      writeln('Einschliessung von f(x):');
      writeln(fx);
      writeln('Einschliessung der 1. Ableitung von f(x):');
      writeln(dfx);
      writeln('Einschliessung der 2. Ableitung von f(x):');
      writeln(ddfx);  writeln;
      ddfEval(g,x,fx,dfx,ddfx);
      writeln('Einschliessung von g(x) = f(x):');
      writeln(fx);
      writeln('Einschliessung der 1. Ableitung von g(x):');
      writeln(dfx);
      writeln('Einschliessung der 2. Ableitung von g(x):');
      writeln(ddfx);  writeln;
      ddfEval(h,x,fx,dfx,ddfx);
      writeln('Einschliessung von h(x) = f(x):');
      writeln(fx);
      writeln('Einschliessung der 1. Ableitung von h(x):');
      writeln(dfx);
      writeln('Einschliessung der 2. Ableitung von h(x):');
      writeln(ddfx);
      writeln('----------------------------------------');
   until false;
end.
```

Mit der Eingabe der Intervalle $x = [-10^{+3}, -10^{+3}]$ und $x = [10^{+3}, 10^{+3}]$ liefert das Programm `tstddf_rrg2` die folgende Bildschirmausgabe:

```
x : Linterval = ?
-1e3      <enter>
Einschliessung von f(x):
[ -1.00000000000000000001E+003, -9.99999999999999999999E+002 ]
Einschliessung der 1. Ableitung von f(x):
[  9.99999999999999999999E-001,  1.00000000000000000001E+000 ]
Einschliessung der 2. Ableitung von f(x):
[ -5.07595889754945676530E-435, -5.07595889754945676529E-435 ]

Einschliessung von g(x) = f(x):
[ -1.00000000000000000001E+003, -9.99999999999999999999E+002 ]
Einschliessung der 1. Ableitung von g(x):
[  9.99999999999999999999E-001,  1.00000000000000000001E+000 ]
Einschliessung der 2. Ableitung von g(x):
[ -4.42116019976557684257E-024, +4.42116019976557684257E-024 ]

Einschliessung von h(x) = f(x):
[ -1.00000000000000000001E+003, -9.99999999999999999999E+002 ]
Einschliessung der 1. Ableitung von h(x):
[  9.99999999999999999999E-001,  1.00000000000000000001E+000 ]
Einschliessung der 2. Ableitung von h(x):
[ -5.07595889754945676530E-435, -5.07595889754945676529E-435 ]
--------------------------------------------------------------

x : Linterval = ?
1e3       <enter>
Einschliessung von f(x):
[ -2.78513457818000000001E-021, +2.78519942182700000001E-021 ]
Einschliessung der 1. Ableitung von f(x):
[ -3.28599561510300000001E-024, +3.28599561510300000001E-024 ]
Einschliessung der 2. Ableitung von f(x):
[ -4.42116019976557684257E-024, +4.42116019976557684257E-024 ]

Einschliessung von g(x) = f(x):
[ -5.07595889754945676530E-435, -5.07595889754945676529E-435 ]
Einschliessung der 1. Ableitung von g(x):
[  5.07595889754945676529E-435,  5.07595889754945676530E-435 ]
```

```
Einschliessung der 2. Ableitung von g(x):
[ -5.07595889754945676530E-435, -5.07595889754945676529E-435 ]

Einschliessung von h(x) = f(x):
[ -5.07595889754945676530E-435, -5.07595889754945676529E-435 ]
Einschliessung der 1. Ableitung von h(x):
[  5.07595889754945676529E-435,  5.07595889754945676530E-435 ]
Einschliessung der 2. Ableitung von h(x):
[ -5.07595889754945676530E-435, -5.07595889754945676529E-435 ]
```

Mit der Eingabe des Intervalls $x = [0, 0]$ liefert das Programm `tstddf_rrg2` die folgende Bildschirmausgabe:

```
x : Linterval = ?
0         <enter>
Einschliessung von f(x):
[ -6.93147180559945309418E-001, -6.93147180559945309417E-001 ]
Einschliessung der 1. Ableitung von f(x):
[  4.99999999999999999999E-001,  5.00000000000000000001E+000 ]
Einschliessung der 2. Ableitung von f(x):
[ -2.50000000000000000001E-001, -2.49999999999999999999E-001 ]

Einschliessung von g(x) = f(x):
[ -6.93147180559945309418E-001, -6.93147180559945309417E-001 ]
Einschliessung der 1. Ableitung von g(x):
[  4.99999999999999999999E-001,  5.00000000000000000001E+000 ]
Einschliessung der 2. Ableitung von g(x):
[ -2.50000000000000000001E-001, -2.49999999999999999999E-001 ]

Einschliessung von h(x) = f(x):
[ -6.93147180559945309418E-001, -6.93147180559945309417E-001 ]
Einschliessung der 1. Ableitung von h(x):
[  4.99999999999999999999E-001,  5.00000000000000000001E+000 ]
Einschliessung der 2. Ableitung von h(x):
[ -2.50000000000000000001E-001, -2.49999999999999999999E-001 ]
```

Nur mit der eigens deklarierten Funktion $h(x) \equiv f(x)$ erhält man also für die drei Punktintervall–Argumente $x = -10^3, +10^3, 0$ maximalgenaue Einschließungen des Funktionswertes und der beiden ersten Ableitungen.

8.1.1.3 $f(x)$, $f'(x)$, $f''(x)$ vom Typ `interval`

Wenn man $f(x)$, $f'(x)$, $f''(x)$ durch Intervalle vom Typ `rrGintervall` einschließt, so kann man diese Ergebnisse i.a. maximalgenau ins gröbere `interval`–Format runden. Genau diese Vorgehensweise wird realisiert beim Aufruf der Prozeduren:

Einschließung von $f(x)$:

```
procedure fEval ( function f(x:DerivType) : DerivType;
(ddf_rrg)         x : interval; var fx : interval );
```

Einschließung von $f(x)$, $f'(x)$:

```
procedure dfEval ( function f(x:DerivType) : DerivType;
                   x : interval;
(ddf_rrg)          var fx, dfx : interval );
```

Einschließung von $f(x)$, $f'(x)$, $f''(x)$:

```
procedure ddfEval ( function f(x:DerivType) : DerivType;
                    x : interval;
(ddf_rrg)           var fx, dfx, ddfx : interval );
```

Es gilt: $f(x) \in$ `fx`, $f'(x) \in$ `dfx`, $f''(x) \in$ `ddfx`.
function `f(x : DerivType) : DerivType` ist dabei so zu deklarieren, wie es auf Seite 350 beschrieben wurde. Bitte beachten Sie, daß das Intervall–Argument `x : interval` hier vom Anwender nicht in den Typ `DerivType` umzuwandeln ist, sondern daß dies in der Prozedur jeweils intern erledigt wird.

8.1.1.3.1 Programmbeispiel

Als Anwendung betrachten wir die Funktionen

$$f(x) = (x+1) \cdot [\ln(1+x) - 1], \quad x > -1;$$

Bei Anwendung der Automatischen Differentiation erhält man für $f'(x)$ in der Umgebung von $x = 0$ wegen Intervallüberschätzungen so grobe Einschließungen, daß noch nicht einmal das richtige Vorzeichen von $f'(x)$ bestimmt werden kann. Mit den Formeln

$$\begin{aligned}
f(u(x)) &= (u+1)\cdot[\ln(1+u)-1] \\
\frac{d}{dx}f(u) &= \ln(1+u)\cdot u'(x) \\
\frac{d^2}{dx^2}f(u) &= \frac{(u')^2}{1+u} + \ln(1+u)\cdot u''(x)
\end{aligned}$$

deklarieren wir daher die mit $f(x)$ identische Funktion

```
function h(u : DerivType): DerivType;
var t : rrGinterval;
begin
   t := ln1p(u.f);
   h.f   := (u.f + 1) * (t - 1);          { Funktionswert }
   h.df  := u.df * t;                     {  1. Ableitung }
   h.ddf := sqr(u.df)/(1+u.f) + t*u.ddf;  {  2. Ableitung }
end; { h }
```

um die genannten Intervallüberschätzungen zu vermeiden. Das nachfolgende Programm `tstddf_rrg3` ermöglicht die Eingabe eines Intervall–Arguments `x : interval` und liefert die Ergebnisse bzgl. $f(x)$ und $h(x)$.

```
program tstddf_rrg3(input,output);
{ Zu gegebenem Argumentintervall x werden berechnet:      }
{ Einschliessungen fuer Funktionswert, 1. u. 2. Ableitung }
{ der identischen Funktionen:        f(x)=h(x);           }
{ f(x) = (x+1) * [ln(1+x)-1];                             }
{ f(u) = h(u): DerivType   wird eigens deklariert, um     }
{ unnoetige Intervall-Ueberschaetzungen zu vermeiden.     }

use i_ari, ddf_rrg, rrgi_util, rrgi_ari;

function f(u : DerivType): DerivType;
begin
   f := (u+1) * ( ln1p(u)-1 );
end; { f }

function h(u : DerivType): DerivType;
```

```
var t : rrGinterval;
begin
   t := ln1p(u.f);
   h.f   := (u.f + 1) * (t - 1);          { Funktionswert }
   h.df  := u.df * t;                     {  1. Ableitung }
   h.ddf := sqr(u.df)/(1+u.f) + t*u.ddf;  {  2. Ableitung }
end; { h }

var x,               { Argumentintervall                 }
   fx,               { Einschliessung des Funktionswertes }
   dfx,              { Einschliessung der 1. Ableitung    }
   ddfx : interval;  { Einschliessung der 2. Ableitung    }
begin
   repeat
      writeln('x : interval = ?');  read(x);
      ddfEval(f,x,fx,dfx,ddfx);
      writeln('Einschliessung von f(x):');
      writeln(fx);
      writeln('Einschliessung der 1. Ableitung von f(x):');
      writeln(dfx);
      writeln('Einschliessung der 2. Ableitung von f(x):');
      writeln(ddfx);  writeln;

      ddfEval(h,x,fx,dfx,ddfx);
      writeln('Einschliessung von h(x) = f(x):');
      writeln(fx);
      writeln('Einschliessung der 1. Ableitung von h(x):');
      writeln(dfx);
      writeln('Einschliessung der 2. Ableitung von h(x):');
      writeln(ddfx);
      writeln('-----------------------------------------');
   until false;
end.
```

Mit der Eingabe der Intervalle $x = [0, 0]$ und $x = [10^{-30}, 10^{-30}]$ liefert das Programm `tstddf_rrg3` die folgende Bildschirmausgabe:

```
x : interval = ?
0        <enter>
Einschliessung von f(x):
[ -1.000000000001E+000, -9.999999999999E-001 ]
```

```
Einschliessung der 1. Ableitung von f(x):
[ -2.500505000006E-024,  2.500505000006E-024 ]
Einschliessung der 2. Ableitung von f(x):
[  9.999999999999E-001,  1.000000000001E+000 ]

Einschliessung von h(x) = f(x):
[ -1.000000000001E+000, -9.999999999999E-001 ]
Einschliessung der 1. Ableitung von h(x):
[  0.000000000000E+000,  0.000000000000E+000 ]
Einschliessung der 2. Ableitung von h(x):
[  9.999999999999E-001,  1.000000000001E+000 ]
----------------------------------------------
x : interval = ?
1e-30    <enter>
Einschliessung von f(x):
[ -1.000000000001E+000, -9.999999999999E-001 ]
Einschliessung der 1. Ableitung von f(x):
[ -2.500504000006E-024,  2.500506000006E-024 ]
Einschliessung der 2. Ableitung von f(x):
[  9.999999999999E-001,  1.000000000001E+000 ]

Einschliessung von h(x) = f(x):
[ -1.000000000001E+000, -9.999999999999E-001 ]
Einschliessung der 1. Ableitung von h(x):
[  9.999999999999E-031,  1.000000000001E-030 ]
Einschliessung der 2. Ableitung von h(x):
[  9.999999999999E-001,  1.000000000001E+000 ]
----------------------------------------------
```

Die eigens deklarierte Funktion `function h(u : DerivType): DerivType` liefert im Gegensatz zur direkten Anwendung der Automatischen Differentiation mittels `function f(u : DerivType): DerivType` in der Umgebung von Null sogar maximalgenaue Einschließungen der 1. Ableitung.

> Mögliche Intervallüberschätzungen können bei der Automatischen Differentiation i.a. vermieden werden, wenn man die zu differenzierende Funktion mit Hilfe der Ableitungsregeln als Funktion vom Typ `DerivType` neu deklariert, vergl. Seite 355 und 361.

8.1.1.4 Weitere Hilfsfunktionen des Moduls `ddf_rrg`

Neben den schon auf Seite 352 beschriebenen Hilfsfunktionen exportiert das Modul `ddf_rrg` die folgende Funktion

function `DerivConst( c : ■ ) : DerivType;` (ddf_ rrg)

wobei ■ durch die Datentypen `real, Longreal, rrG` ersetzt werden kann. Zur Konstruktion von `DerivType`-Ausdrücken lassen sich mit Hilfe dieser Funktionen die Datentypen `real, Longreal, rrG` in den Typ `DerivType` umwandeln.

8.1.2 Ableitungen höherer Ordnung, Taylor-Arithmetik

Wir betrachten jetzt wieder reelle, aber beliebig oft differenzierbare Funktionen f, deren Ableitungen bzw. Taylorkoeffizienten bis zu einer Ordnung $p \geq 0$ einzuschließen sind, wobei p vom Anwender vorgegeben wird.

Als Funktionsargumente lassen wir dabei nicht nur einen einzelnen Punkt des Definitionsintervalls von $f(x)$ zu, sondern wir betrachten vielmehr ganze Argumentintervalle x_0, die im Bedarfsfall natürlich auch als Punktintervalle benutzt werden können. Damit sind wir dann z.B. in der Lage, das Restglied einer abgebrochenen Taylorreihe der Ordnung p für das ganze betrachtete Entwicklungsintervall x_0 abzuschätzen, wenn in der $(p+1)$-ten Ableitung des Restgliedes der bekannte Zwischenwert durch dieses Intervall ersetzt wird. Bei Anfangswertproblemen und bei der numerischen Integration sind solche Abschätzungen von großer Bedeutung, wenn garantierte Einschließungen der Lösungen zu berechnen sind.

Der naive Anwender wird sich nun fragen, wie es überhaupt möglich ist, z.B. die 50. Ableitung einer stark verschachtelten Funktion in vergleichsweise kurzen Laufzeiten in hoher Genauigkeit einzuschließen, denn in der Praxis wird man oft schon kapitulieren, wenn man nur die 3. Ableitung mit Papier und Bleistift zu berechnen hat! Die Zauberei der Automatischen Differentiation wird jedoch durchschaubar, wenn man erkennt, daß sich die Taylorkoeffizienten der Standardfunktionen **rekursiv** berechnen lassen. Da die Auswertungen intervallmäßig erfolgen, können in Ausnahmefällen unerwünschte Überschätzungen auftreten, die sich manchmal vermeiden lassen, wenn man die auftretenden Terme entsprechend umformt. Die dazu notwendigen Informationen findet der Anwender im folgenden Abschnitt, der jedoch übersprungen werden kann, wenn die mit Hilfe der Routinen des übernächsten Abschnitts berechneten Einschließungen hinreichend genau sind.

8.1.2.1 Theoretische Grundlagen

Grundlage aller Überlegungen zur Intervall–Taylorarithmetik ist die Taylorreihe[1] einer beliebig oft differenzierbaren Funktion $f : I\!R \to I\!R$

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \tag{8.3}$$

Für den Taylorkoeffizienten $\dfrac{f^{(k)}(x_0)}{k!}$ der Funktion f im Entwicklungspunkt x_0 wählen wir folgende Bezeichnung:

$$(f)_k := \frac{f^{(k)}(x_0)}{k!} = \frac{1}{k!}\frac{d^k f(x_0)}{dt^k} \tag{8.4}$$

Damit läßt sich (8.3) schreiben als

$$f(x) = \sum_{k=0}^{\infty} (f)_k (x - x_0)^k \tag{8.5}$$

Wir betrachten jetzt Funktionen f, die sich aus Standardfunktionen mit Hilfe der vier Grundoperationen konstruieren lassen. Die Taylorkoeffizienten solcher Funktionen lassen sich dann rekursiv bestimmen.

Für die konstante Funktion $f = c$ gilt zunächst:

$$(c)_0 = c, \quad (c)_k = 0, \ \text{falls } k \geq 1 \tag{8.6}$$

und für die unabhängige Variable $f = x$:

$$(x)_0 = x, \quad (x)_1 = 1, \quad (x)_k = 0, \ \text{falls } k \geq 2; \tag{8.7}$$

Wegen der Eindeutigkeit der Potenzreihen erhält man für die beiden Funktionen $u(x)$ und $v(x)$ die beiden folgenden Gleichungen:

$$(u \pm v)_k = (u)_k \pm (v)_k \tag{8.8}$$

$$(u \cdot v)_k = \sum_{j=0}^{k} (u)_j (v)_{k-j} = \sum_{j=0}^{k} (u)_{k-j} (v)_j \tag{8.9}$$

Die Quotientenregel

$$(u/v)_k = \frac{1}{(v)_0}\left[(u)_k - \sum_{j=1}^{k} (v)_j (u/v)_{k-j}\right] \tag{8.10}$$

[1] Wenn man z.B. voraussetzt, daß $f(x)$ analytisch ist, so existiert die dann eindeutig bestimmte Taylorreihe.

ergibt sich wie folgt: mit $f \equiv u/v$ gilt: $u = v \cdot f$, und die Produktregel (8.9) liefert:

$$(u)_k = \sum_{j=0}^{k} (v)_j (f)_{k-j} = (v)_0 (f)_k + \sum_{j=1}^{k} (v)_j (f)_{k-j}$$

Löst man nach $(f)_k = (u/v)_k$ auf, so erhält man gerade (8.10).

Wir betrachten jetzt Funktionen $f(u)$, wobei $u = u(x)$ eine beliebig oft differenzierbare Funktion von x sein soll. Mit diesen Bezeichnungen gilt dann:

$$\text{(8.11)} \qquad w(x) \equiv f(u(x))$$

$$\text{(8.12)} \qquad \frac{d}{dx} w = w'(x) = \left.\frac{d\,f(u)}{du}\right|_{u=u(x)} \cdot u'(x) = f'(u)\Big|_{u=u(x)} \cdot u'(x);$$

Als erste Anwendung betrachten wir die Quadratwurzel $f(u) = \sqrt{u}$ und schreiben $f \cdot f = u$. Die Anwendung der Produktregel (8.9) auf $f \cdot f$ liefert dann nach einfachen Umformungen für $k \geq 1$:

$$\begin{aligned} (u)_k &= (f \cdot f)_k = \sum_{j=0}^{k} (f)_j (f)_{k-j} \\ &= (f)_0 (f)_k + \sum_{j=1}^{k-1} (f)_j (f)_{k-j} + (f)_k (f)_0 \\ &= 2(f)_0 (f)_k + \sum_{j=1}^{k-1} (f)_j (f)_{k-j} \end{aligned}$$

Löst man die letzte Gleichung nach $(f)_k$ auf, so folgt:

$$(f)_k = \frac{1}{2(f)_0} \left[(u)_k - \sum_{j=1}^{k-1} (f)_j (f)_{k-j} \right]$$

Beachten Sie bitte, daß $(f)_j$ den Taylorkoeffizienten bzgl. x bedeutet, d.h.

$$(f)_j := \frac{1}{j!} \frac{d^j}{d\,x^j} f\,(u(x)) \Bigg|_{u=u(x_0)}$$

Ersetzt man jetzt noch f durch $\sqrt{u}$, so erhält man für die Quadratwurzel:

$$\text{(8.13)} \qquad (\sqrt{u})_k = \frac{1}{2(\sqrt{u})_0} \left[(u)_k - \sum_{j=1}^{k-1} (\sqrt{u})_j (\sqrt{u})_{k-j} \right], \quad k \geq 1;$$

Man erkennt deutlich den rekursiven Aufbau in (8.13), d.h. zur Bestimmung von $(\sqrt{u})_k$ müssen vorher alle Taylorkoeffizienten $(\sqrt{u})_j$ der Ordnung $j < k$ bereits berechnet worden sein. Da die Summe in (8.13) jeweils zwei gleiche Summanden besitzt, ergibt sich für $k \geq 1$ die günstigere Form:

$$(\sqrt{u})_k = \begin{cases} \dfrac{1}{2(\sqrt{u})_0} \left[(u)_k - 2 \displaystyle\sum_{j=1}^{(k-1)/2} (\sqrt{u})_j (\sqrt{u})_{k-j} \right] & k \text{ ungerade} \\ \dfrac{1}{2(\sqrt{u})_0} \left[(u)_k - 2 \displaystyle\sum_{j=1}^{(k-2)/2} (\sqrt{u})_j (\sqrt{u})_{k-j} - (\sqrt{u})_{k/2}^2 \right] & k \text{ gerade} \end{cases}$$

Wegen $(f(u))_0 = f((u)_0)$ gilt natürlich: $(\sqrt{u})_0 = \sqrt{(u)_0}$.

Ganz analog erhält man für die quadratische Funktion $f(u) = u^2$ die Taylorkoeffizienten im Falle $k \geq 1$:

$$(u^2)_k = \begin{cases} 2 \displaystyle\sum_{j=0}^{(k-1)/2} (u)_j (u)_{k-j} & k \text{ ungerade} \\ 2 \displaystyle\sum_{j=0}^{(k-2)/2} (u)_j (u)_{k-j} + (u)_{k/2}^2 & k \text{ gerade} \end{cases}$$

Um Rekursionsformeln für die Taylorkoeffizienten der Standardfunktionen herleiten zu können, muß noch eine grundlegende Beziehung zwischen den Taylorkoeffizienten von f und seiner Ableitung f' gefunden werden. Dazu berechnen wir zunächst die Ableitung von (8.5) nach x und erhalten:

$$f'(x) = \sum_{k=1}^{\infty} k(f)_k (x - x_0)^{k-1} = \sum_{k=0}^{\infty} (k+1)(f)_{k+1} (x - x_0)^k$$

Andererseits kann man in (8.5) f durch f' ersetzen:

$$f'(x) = \sum_{k=0}^{\infty} (f')_k (x - x_0)^k$$

und Koeffizientenvergleich liefert die wichtige Beziehung:

$$\boxed{(f')_k = (k+1) \cdot (f)_{k+1}} \tag{8.14}$$

die auch dann richtig bleibt, wenn f als $f = f(u(x))$ gegeben ist. f' ist dann definiert durch:

$$f' := \left. \frac{d\,f(u)}{du} \right|_{u=u(x)} \cdot u'(x)$$

Beachten Sie bitte weiter, daß sich die Ordnungen k der Taylorkoeffizienten stets auf die unabhängige Variable x beziehen, auch wenn f durch $f(u(x))$ gegeben ist.

Als erste Anwendung von (8.14) betrachten wir die Exponentialfunktion $f(u) = e^u$. Bildet man dann die 1. Ableitung von $w(x) := f(u(x)) = e^{u(x)}$ nach x, so folgt: $w' = w \cdot u'$, und die Produktregel (8.9) liefert:

$$(w')_{k-1} = (w \cdot u')_{k-1} = \sum_{j=0}^{k-1} (w)_j (u')_{k-1-j}, \quad k \geq 1$$

Nach (8.14) gilt nun: $(w')_{k-1} = k(w)_k$ bzw. $(u')_{k-1-j} = (k-j)(u)_{k-j}$, und wir erhalten:

$$k(w)_k = \sum_{j=0}^{k-1} (w)_j (k-j)(u)_{k-j}, \quad k \geq 1$$

Löst man nach $(w)_k$ auf und setzt $w = e^u$, so ergibt sich die Rekursionsformel für die Taylorkoeffizienten der Exponentialfunktion:

$$\begin{aligned} (e^u)_k &= \frac{1}{k} \sum_{j=0}^{k-1} (k-j)(e^u)_j (u)_{k-j}, \quad u = u(x), \quad k \geq 1 \\ (e^u)_0 &= e^{(u)_0}, \quad k = 0. \end{aligned}$$

Ganz analog findet man die entsprechenden Rekursionsformeln für $\sin(u)$ und $\cos(u)$:

$$\begin{aligned} (\sin u)_k &= \frac{1}{k} \sum_{j=0}^{k-1} (k-j)(\cos u)_j (u)_{k-j} \\ (\cos u)_k &= -\frac{1}{k} \sum_{j=0}^{k-1} (k-j)(\sin u)_j (u)_{k-j} \end{aligned}$$

und für $\sinh(u)$ und $\cosh(u)$:

$$\begin{aligned}
(\sinh u)_k &= \frac{1}{k}\sum_{j=0}^{k-1}(k-j)(\cosh u)_j(u)_{k-j} \\
(\cosh u)_k &= \frac{1}{k}\sum_{j=0}^{k-1}(k-j)(\sinh u)_j(u)_{k-j}
\end{aligned}$$

Die Potenzfunktion $f(u) = u^a$ mit konstantem Exponenten $a \in I\!R$ läßt sich analog behandeln. Bildet man die 1. Ableitung von $w(x) := f(u(x)) = u(x)^a$ nach x, so folgt: $w'u = aw \cdot u'$, und die Produktregel (8.9) liefert:

$$(u^a)_k = \frac{1}{k\cdot(u)_0}\sum_{j=0}^{k-1}[a\cdot(k-j)-j]\cdot(u^a)_j\cdot(u)_{k-j}, \quad k\geq 1$$

Fast alle anderen Standardfunktionen lassen sich wie folgt behandeln: Gegeben sei $w(x) = f(u(x))$; für die 1. Ableitung nach x gilt:

$$\frac{d}{dx}w = w'(x) = \left.\frac{d\,f(u)}{du}\right|_{u=u(x)}\cdot u'(x) = f'(u)\,|_{u=u(x)}\cdot u'(x);$$

Schreibt man $f'(u)\,|_{u=u(x)} = \left.\frac{1}{g(u)}\right|_{u=u(x)}$ oder kürzer: $f'(u) = \frac{1}{g(u)}$, so folgt

$$\boxed{u' = g\cdot w'} \tag{8.15}$$

Die Produktregel (8.9) zusammen mit (8.14) ergibt:

$$\begin{aligned}
k(u)_k &= (u')_{k-1} = (g\cdot w')_{k-1} = \sum_{j=0}^{k-1}(g)_{k-j-1}(w')_j \\
&= \sum_{j=0}^{k-1}(g)_{k-j-1}(j+1)(w)_{j+1} = \sum_{j=1}^{k}j\cdot(g)_{k-j}\cdot(w)_j \\
&= \sum_{j=1}^{k-1}j\cdot(w)_j\cdot(g)_{k-j} + k\cdot(w)_k\cdot(g)_0 \quad\longrightarrow
\end{aligned}$$

$$(w)_k = \frac{1}{(g)_0}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j\cdot(w)_j\cdot(g)_{k-j}\right], \quad k\ge 1$$

Da sich die Taylorkoeffizienten der Ordnung k stets auf die unabhängige Variable x beziehen, gilt $(w)_k = (f(u))_k = (f)_k$ und man erhält:

(8.16)
$$\boxed{\begin{gathered}(f)_k = \frac{1}{(g)_0}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j\cdot(f)_j\cdot(g)_{k-j}\right], \quad k\ge 1\\ f(u) = f(u(x)); \quad g(u) = \left[\frac{d\,f(u)}{du}\right]^{-1}, \quad \frac{d\,f(u)}{du}\neq 0\end{gathered}}$$

Die folgenden Tabelle enthält die Rekursionsformeln der Taylorkoeffizienten $(f)_k$. Bei Angabe von $g(u)$ wurde (8.16) benutzt.

Rekursionsformeln der Taylorkoeffizienten $(f)_k$, (Teil 1)		
$f(u)$	$g(u)$	$(f)_k,\ k\ge 1$
$\ln u$	u	$\frac{1}{(u)_0}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j\cdot(\ln u)_j(u)_{k-j}\right]$
$\tan u$	$\cos^2 u$	$\frac{1}{\cos^2(u)_0}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j\cdot(\tan u)_j(\cos^2 u)_{k-j}\right]$
$\cot u$	$-\sin^2 u$	$\frac{-1}{\sin^2(u)_0}\left[(u)_k + \frac{1}{k}\sum_{j=1}^{k-1} j\cdot(\cot u)_j(\sin^2 u)_{k-j}\right]$
$\arcsin u$	$\sqrt{1-u^2}$	$\frac{1}{\sqrt{1-(u)_0^2}}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j(\arcsin u)_j(\sqrt{1-u^2})_{k-j}\right]$
$\arccos u$	$-\sqrt{1-u^2}$	$\frac{-1}{\sqrt{1-(u)_0^2}}\left[(u)_k + \frac{1}{k}\sum_{j=1}^{k-1} j(\arccos u)_j(\sqrt{1-u^2})_{k-j}\right]$

Rekursionsformeln der Taylorkoeffizienten $(f)_k$, (Teil 2)		
$f(u)$	$g(u)$	$(f)_k,\ k \geq 1$
$\arctan u$	$1+u^2$	$\frac{1}{1+(u)_0^2}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j(\arctan u)_j(1+u^2)_{k-j}\right]$
$\operatorname{arccot} u$	$-(1+u^2)$	$\frac{-1}{1+(u)_0^2}\left[(u)_k + \frac{1}{k}\sum_{j=1}^{k-1} j(\operatorname{arccot} u)_j(1+u^2)_{k-j}\right]$
$\tanh u$	$\cosh^2 u$	$\frac{1}{\cosh^2(u)_0}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j(\tanh u)_j(\cosh^2 u)_{k-j}\right]$
$\coth u$	$-\sinh^2 u$	$\frac{-1}{\sinh^2(u)_0}\left[(u)_k + \frac{1}{k}\sum_{j=1}^{k-1} j(\coth u)_j(\sinh^2 u)_{k-j}\right]$
$\operatorname{arsinh} u$	$\sqrt{u^2+1}$	$\frac{1}{\sqrt{(u)_0^2+1}}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j(\operatorname{arsinh} u)_j(\sqrt{u^2+1})_{k-j}\right]$
$\operatorname{arcosh} u$	$\sqrt{u^2-1}$	$\frac{1}{\sqrt{(u)_0^2-1}}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j(\operatorname{arsinh} u)_j(\sqrt{u^2-1})_{k-j}\right]$
$\operatorname{artanh} u$	$1-u^2$	$\frac{1}{1-(u)_0^2}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j(\operatorname{artanh} u)_j(1-u^2)_{k-j}\right]$
$\operatorname{arcoth} u$	$1-u^2$	$\frac{1}{1-(u)_0^2}\left[(u)_k - \frac{1}{k}\sum_{j=1}^{k-1} j(\operatorname{arcoth} u)_j(1-u^2)_{k-j}\right]$
u^2	$(u^2)_k = \sum_{j=0}^{k}(u)_j(u)_{k-j}$	

Rekursionsformeln der Taylorkoeffizienten $(f)_k$, (Teil 3)	
$f(u)$	$(f)_k,\ k \geq 1$
$\sqrt{u}$	$(\sqrt{u})_k = \dfrac{1}{2(\sqrt{u})_0}\left[(u)_k - \sum_{j=1}^{k-1}(\sqrt{u})_j(\sqrt{u})_{k-j}\right]$
$\sin u$	$(\sin u)_k = \dfrac{1}{k}\sum_{j=0}^{k-1}(k-j)(\cos u)_j(u)_{k-j}$
$\cos u$	$(\cos u)_k = -\dfrac{1}{k}\sum_{j=0}^{k-1}(k-j)(\sin u)_j(u)_{k-j}$
$\sinh u$	$(\sinh u)_k = \dfrac{1}{k}\sum_{j=0}^{k-1}(k-j)(\cosh u)_j(u)_{k-j}$
$\cosh u$	$(\cosh u)_k = \dfrac{1}{k}\sum_{j=0}^{k-1}(k-j)(\sinh u)_j(u)_{k-j}$
e^u	$(e^u)_k = \dfrac{1}{k}\sum_{j=0}^{k-1}(k-j)(e^u)_j(u)_{k-j}$
2^u	$(2^u)_k = \dfrac{\ln 2}{k}\sum_{j=0}^{k-1}(k-j)(2^u)_j(u)_{k-j}$
10^u	$(10^u)_k = \dfrac{\ln 10}{k}\sum_{j=0}^{k-1}(k-j)(10^u)_j(u)_{k-j}$
$\log_2 u$	$(\log_2 u)_k = \dfrac{1}{\ln 2}\cdot(\ln u)_k$
$\log_{10} u$	$(\log_{10} u)_k = \dfrac{1}{\ln 10}\cdot(\ln u)_k$

Hinweise:

- Für $(u^2)_k$ und $(\sqrt{u})_k$ gibt es bzgl. der Laufzeiten bessere Rekursionsformeln, die auf Seite 367 angegeben sind.
- Die Auswertung der Rekursionsformeln für $(\sin u)_k$ und $(\cos u)_k$ muß paarweise erfolgen. Entsprechendes gilt für $(\sinh u)_k$ und $(\cosh u)_k$.

Da die Rekursionsformeln intervallmäßig auszuwerten sind, können sich die unvermeidbaren Überschätzungen besonders dann störend auswirken, wenn die Taylorkoeffizienten an Stellen x_0 auszuwerten sind, die in der Nähe einer Nullstelle der jeweiligen Ableitungsfunktion liegt. In solchen Fällen ist es dann sinnvoll, die Rekursionsformeln wenn möglich so umzuschreiben, daß die genannten Intervallüberschätzungen vermieden werden.

Als erstes Beispiel betrachten wir dazu die Funktion

$$\begin{aligned} w(x) &= f(u(x)) = u \cdot \ln u \quad \text{mit} \quad u = u(x); \\ w'(x) &= (1 + \ln u) \cdot u'(x) \end{aligned}$$

Für $u(x_0) = x_0 = 1/e$ besitzt jedoch $(1 + \ln u)$ eine Nullstelle, in deren Umgebung $w'(x)$ nur mit starken Auslöschungseffekten ausgewertet wird. Benutzt man jedoch die schon durch `ln_ex(u)` realisierte Funktion $\ln(e\cdot u)$, so lassen sich die genannten Auslöschungseffekte so weit reduzieren, daß $w'(x)$ auch in der Umgebung von $x_0 = 1/e$ in hoher Genauigkeit eingeschlossen werden kann. Wir schreiben also geschickter

$$w'(x) = \ln(e \cdot u) \cdot u'(x),$$

wobei die Funktion[2] $\ln(e\cdot u)$ durch `ln_ex(u)` realisiert wird. Die Anwendung von (8.14) liefert dann zusammen mit der Produktregel (8.9):

$$\begin{aligned} k \cdot (w)_k &= (w')_{k-1} = (\ln(e \cdot u) \cdot u')_{k-1} \\ &= \sum_{j=0}^{k-1} (\ln(e \cdot u))_j (k-j)(u)_{k-j} \end{aligned}$$

Mit $(w)_k = (u \cdot \ln u)_k$ folgt dann:

$$(u \cdot \ln u)_k = \frac{1}{k} \sum_{j=0}^{k-1} (k-j) \cdot (\ln(e \cdot u))_j \cdot (u)_{k-j}, \quad k \geq 1$$

[2]Die Rekursionsformel für $(\ln(e \cdot u))_k$ findet man in der Tabelle auf Seite 376.

Als nächstes Beispiel betrachten wir die Potenzfunktion

$$\begin{aligned} w(x) &= f(u(x)) = u^u = e^{u\cdot\ln(u)} \quad \text{mit} \quad u = u(x) > 0; \\ w'(x) &= \ln(eu)\cdot e^{u\cdot\ln(u)}\cdot u'(x) \end{aligned}$$

Die Anwendung von (8.14) liefert zusammen mit der Produktregel (8.9):

$$\begin{aligned} k\cdot(w)_k &= (w')_{k-1} = (\ \ln(eu)\cdot e^{u\cdot\ln(u)}\cdot u'\ )_{k-1} \\ &= \sum_{j=0}^{k-1}(\ \ln(eu)\cdot e^{u\cdot\ln(u)}\ )_j(k-j)(u)_{k-j} \end{aligned}$$

Mit $(w)_k = (u^u)_k$ folgt dann:

$$(u^u)_k = \frac{1}{k}\sum_{j=0}^{k-1}(k-j)\cdot(\ \ln(eu)\cdot e^{u\cdot\ln(u)}\ )_j\cdot(u)_{k-j}, \quad u(x) > 0, \quad k \geq 1$$

Als nächstes Beispiel betrachten wir die Funktion:

$$w(x) = f(u(x)) = \sqrt{1+u(x)} - 1, \quad w + 1 = \sqrt{1+u}$$
$$w\cdot w = u - 2w$$

und die Anwendung der Produktregel liefert:

$$(w\cdot w)_k = \sum_{j=0}^{k}(w)_j\cdot(w)_{k-j} = (u)_k - 2(w)_k$$

$$(w)_0\cdot(w)_k + \sum_{j=1}^{k-1}(w)_j\cdot(w)_{k-j} + (w)_k\cdot(w)_0 = (u)_k - 2(w)_k$$

$$2(w)_0\cdot(w)_k + 2(w)_k = (u)_k - \sum_{j=1}^{k-1}(w)_j\cdot(w)_{k-j} \quad \longrightarrow$$

$$(f)_k = \frac{1}{2\sqrt{1+(u)_0}}\left[(u)_k - \sum_{j=1}^{k-1}(f)_j(f)_{k-j}\right], \quad (f)_j = (\sqrt{1+u}-1)_j$$

Als nächstes Beispiel betrachten wir die Funktion:

$$w(x) = f(u(x)) = \sqrt{1-u^2(x)}, \qquad u^2(x) < 1$$
$$w\cdot w = 1 - u^2$$

und die Anwendung der Produktregel liefert:

$$(w \cdot w)_k = (1-u^2)_k = -(u^2)_k, \quad k \geq 1$$

$$\sum_{j=0}^{k} (w)_j \cdot (w)_{k-j} = -(u^2)_k$$

$$2(w)_0 \cdot (w)_k + \sum_{j=1}^{k-1} (w)_j \cdot (w)_{k-j} = -(u^2)_k \qquad \longrightarrow$$

$$(\sqrt{1-u^2})_k = \frac{-1}{2\sqrt{1-(u^2)_0}} \left[(u^2)_k + \sum_{j=1}^{k-1} (\sqrt{1-u^2})_j (\sqrt{1-u^2})_{k-j} \right], \; k \geq 1$$

Für die Funktion $w(x) = \sqrt{u^2(x)-1}$ findet man ganz analog die folgende Rekursionsformel:

$$(\sqrt{u^2-1})_k = \frac{1}{2\sqrt{(u^2)_0 - 1}} \left[(u^2)_k - \sum_{j=1}^{k-1} (\sqrt{u^2-1})_j (\sqrt{u^2-1})_{k-j} \right], \; k \geq 1$$

Die Rekursionsformeln der letzten Beispiele sind in der nachfolgenden Tabelle auf Seite 376 zusammengestellt. Beachten Sie bitte, daß die bei den Wurzelfunktionen auftretenden Summen $\sum_{j=1}^{k-1} (f)_j \cdot (f)_{k-j}$ nach dem Beispiel von Seite 367 bzgl. der Laufzeit noch einfacher dargestellt werden können:

$$\sum_{j=0(1)}^{k-1} (f)_j \cdot (f)_{k-j} = \begin{cases} 2 \displaystyle\sum_{j=0(1)}^{(k-1)/2} (f)_j (f)_{k-j} & k \text{ ungerade} \\ 2 \displaystyle\sum_{j=0(1)}^{(k-2)/2} (f)_j (f)_{k-j} + (u)_{k/2}^2 & k \text{ gerade} \end{cases}$$

Rekursionsformeln der Taylorkoeffizienten $(f)_k$, (Teil 4)	
$f(u)$	$(f)_k,\ k \geq 1$
$\ln(e \cdot u)$	$(\ln(e \cdot u))_k = \dfrac{1}{(u)_0} \left[(u)_k - \dfrac{1}{k} \sum_{j=1}^{k-1} j \cdot (\ln(e \cdot u))_j \cdot (u)_{k-j} \right]$
$u \cdot \ln(u)$	$(u \cdot \ln(u))_k = \dfrac{1}{k} \sum_{j=0}^{k-1} (k-j) \cdot (\ln(e \cdot u))_j \cdot (u)_{k-j}$
$e^u - 1$	$(e^u - 1)_k = \dfrac{1}{k} \sum_{j=0}^{k-1} (k-j) \cdot (e^u)_j \cdot (u)_{k-j}$
u^u	$(u^u)_k = \dfrac{1}{k} \sum_{j=0}^{k-1} (k-j) \cdot (\ \ln(eu) \cdot e^{u \cdot \ln(u)}\ )_j \cdot (u)_{k-j}$
$\sqrt{1+u} - 1$	$(f)_k = \dfrac{1}{2\sqrt{1+(u)_0}} \left[(u)_k - \sum_{j=1}^{k-1} (f)_j \cdot (f)_{k-j} \right]$
$\sqrt{1-u^2}$	$(f)_k = \dfrac{-1}{2(f)_0} \left[(u^2)_k + \sum_{j=1}^{k-1} (f)_j \cdot (f)_{k-j} \right]$
$\sqrt{u^2-1}$	$(f)_k = \dfrac{1}{2(f)_0} \left[(u^2)_k - \sum_{j=1}^{k-1} (f)_j \cdot (f)_{k-j} \right]$

8.1.2.2 Taylorkoeffizienten zum Typ `rrGinterval`

Die Einschließung der Taylorkoeffizienten einer gegebenen Funktion $f(u(x))$ erfolgt mit Hilfe des vom Modul `itaylor` exportierten Datentyps[3]

type `itaylor` = **dynamic array** [*] **of** `rrGinterval`; (itaylor)

und einer Funktion `f`, die wie folgt zu deklarieren ist:

```
function f(u : itaylor) : itaylor;
begin
    f := itaylor-Ausdruck;
end;
```

Ist `x0` der vorgegebene Entwicklungspunkt oder das vorgegebene Intervall der Entwicklungspunkte und ist `p` die gewünschte Ordnung, bis zu der die Taylorkoeffizienten zu berechnen sind, so liefert die Prozedur

procedure `expand ( `**var** `u : itaylor; x0 : ■ );` (itaylor)

■ = `interval, Linterval, rrGinterval, rrG`

mit der Ausgangsgröße `u : itaylor[0..p]` einen Vektor mit den $(p+1)$ Intervall-Komponenten:

$$\begin{aligned} \texttt{u[0]} &= \texttt{x0} \\ \texttt{u[1]} &= 1 \\ \texttt{u[j]} &= 0 \quad \text{für } 2 \le j \le \texttt{ub(u)} = p\,, \end{aligned}$$

d.h. `u : itaylor[0..p]` liefert die Taylorkoeffizienten der Ordnung 0 bis zur Ordnung `p = ub(u)` für die Funktion $u(x) \equiv x$.

Der `itaylor-Ausdruck` im Anweisungsteil der obigen Funktion `f` läßt sich realisieren mit Hilfe der vom Modul `itaylor` exportierten Operatoren $\boxed{+,-,*,/}$ mit dem Ergebnistyp `itaylor`, wobei zunächst beide Operanden vom Typ `itaylor` sein dürfen. Ist jedoch nur ein Operand vom Typ `itaylor`, so darf der andere Operand die folgenden Datentypen annehmen:

[3]Die Routinen zur Berechnung der Taylorkoeffizienten zum Typ `Linterval` finden Sie ab Seite 397.

```
integer, real, Longreal, rrG, rrGinterval
```

Die beiden Anweisungen `expand(u,x0);   y := f(u)` liefern dann mit `y : itaylor[0..p]` Einschließungen der Taylorkoeffizienten bis zur Ordnung `p` der Funktion `f` für alle Entwicklungspunkte aus dem Intervall `x0`.

Die Einschließung der Taylorkoeffizienten der Funktion $f(x) = x/(1+x^2)$ liefert das nachfolgende Programm `tstitayl1`:

```
program tstitayl1(input,output);
{ Einschliessung der Taylorkoeffizienten von f=x/(1+x*x) }
use itaylor, rrgi_ari, li_ari;

var p,k : integer;
   x0   : rrGinterval; { Entwicklungspunkte-Intervall    }
   h    : Linterval;   { Ausgabe der Tayl.-Koeffizienten }
   y,u  : itaylor;

function f(u : itaylor) : itaylor[0..ub(u)];
begin
   f := u / (1 + u*u);
end; { f }

begin
   repeat
      writeln('Order of derivative = ?');   read(p);
      allocate(u,0..p); { Speicherplatz der dynamischen  }
      allocate(y,0..p); { Variablen   u,y   anlegen.     }
      writeln('point of evaluation = ?');  read(x0);
      expand(u,x0);     { Eingangsvariable u realisieren }
      y := f(u);
      for k := 0 to p do
      begin
         h := y[k];     { Ausgabe der Einschliessung der }
         writeln(k:3,' : ',h);     { Taylorkoeffizienten }
      end;
      free(u); free(y); { Speicher fuer  u,y  freigeben. }
   until p<0;
end.
```

Für den Entwicklungspunkt `x0 = 0` liefert das Programm `tstitayl1` bzgl. der Ordnung `p = 5` die folgende Bildschirmausgabe:

```
Order of derivative = ?
5     <enter>
point of evaluation = ?
0     <enter>
0 : [ 0.00000000000000000000E+000, 0.00000000000000000000E+000]
1 : [ 9.99999999999999999999E-001, 1.00000000000000000001E+000]
2 : [ 0.00000000000000000000E+000, 0.00000000000000000000E+000]
3 : [-1.00000000000000000001E+000,-9.99999999999999999999E-001]
4 : [ 0.00000000000000000000E+000, 0.00000000000000000000E+000]
5 : [ 9.99999999999999999999E-001, 1.00000000000000000001E+000]
```

Hinweise:

- Beachten Sie bitte, daß obige Funktion Einschließungen der Taylorkoeffizienten im 26-stelligen `rrG`–Format liefert, so daß mit solchen Zwischenergebnissen weitere Intervallrechnungen in hoher Genauigkeit durchgeführt werden können.

- In der Regel werden verschwindende Taylorkoeffizienten wie hier nicht maximalgenau eingeschlossen. Vergleichen Sie dazu beispielsweise die entsprechenden Ergebnisse für den Entwicklungspunkt `x0 = 1`.

- Im Anweisungsteil einer Funktion

  ```
  function f(u : itaylor) : itaylor[0..ub(u)]
  ```

  kann der Ausdruck für `f` auch Standardfunktionen vom Typ `itaylor` enthalten, die in der Tabelle auf der folgenden Seite zusammengestellt sind und vom Modul `itaylor` exportiert werden. Dadurch lassen sich Einschließungen der Taylorkoeffizienten auch für praktisch beliebig verschachtelte Funktionen berechnen.

- Daß die oben genannten Verschachtelungen jedoch nicht blind gewählt werden sollten, zeigt schon das Beispiel von Seite 381. Der ungünstig gewählte Funktionsterm für $h(x)$ liefert nämlich für betragsmäßig große Argumente viel zu grobe Einschließungen der Ableitungswerte.

- Die Einschließungen der Taylorkoeffizienten fallen i.a. um so gröber aus, je höher die Ordnung `p` gewählt wird.

Standardfunktionen vom Typ `itaylor` (itaylor) `u : itaylor;` `a : rrGinterval;`			
Funktion	**Aufruf**	**Funktion**	**Aufruf**
u^2	`sqr(u)`	$\tan(u)$	`tan(u)`
$\sqrt{u}$	`sqrt(u)`	$\cot(u)$	`cot(u)`
$\sqrt{1+u^2}$	`sqrt1px2(u)`	$\sin(\pi u)$	`sinpi(u)`
$\sqrt{1-u^2}$	`sqrt1mx2(u)`	$\cos(\pi u)$	`cospi(u)`
$\sqrt{u^2-1}$	`sqrtx2m1(u)`	$\tan(\pi u)$	`tanpi(u)`
$\sqrt{1+u}-1$	`sqrt1pm1(u)`	$\cot(\pi u)$	`cotpi(u)`
e^u	`exp(u)`	$\arcsin(u)$	`arcsin(u)`
e^u-1	`expm1(u)`	$\arccos(u)$	`arccos(u)`
2^u	`exp2(u)`	$\arctan(u)$	`arctan(u)`
10^u	`exp10(u)`	$\mathrm{arccot}(u)$	`arccot(u)`
u^a	`power(u,a)`	$\sinh(u)$	`sinh(u)`
u^u	`power(u)`	$\cosh(u)$	`cosh(u)`
$\ln(u)$	`ln(u)`	$\tanh(u)$	`tanh(u)`
$\ln(e\cdot u)$	`ln_ex(u)`	$\coth(u)$	`coth(u)`
$\ln(1+u)$	`ln1p(u)`	$\mathrm{arsinh}(u)$	`arsinh(u)`
$\ln(1+e^u)$	`lnexp1(u)`	$\mathrm{arcosh}(u)$	`arcosh(u)`
$u\cdot\ln(u)$	`xlnx(u)`	$\mathrm{arcosh}(1+u)$	`arcosh1p(u)`
$\log_2(u)$	`log2(u)`	$\mathrm{artanh}(u)$	`artanh(u)`
$\log_{10}(u)$	`log10(u)`	$\mathrm{arcoth}(u)$	`arcoth(u)`
$\log_{10}(1+u)$	`log1p(u)`		
$\log_a(u)$	`loga(a,u)`		
$\sin(u)$	`sin(u)`		
$\cos(u)$	`cos(u)`		

Mit Hilfe der nachfolgenden Prozedur berechnet man zu gegebener Funktion `f` vom Ergebnistyp `itaylor` garantierte Einschließungen seiner Taylorkoeffizienten zu Entwicklungspunkten aus dem Intervall `x0`.

procedure `taylor_K` (`function f(t : itaylor) : itaylor;`
(itaylor) `x0 :` ■; **var** `taylor : itaylor`);
■ = `rrGinterval, Linterval, interval;`

Die Ausgangsgröße `taylor : itaylor[0..p]` liefert die Einschließungen der Taylorkoeffizienten von `f` bis zur Ordnung `p = ub(taylor)`.

Mit Hilfe der folgenden Prozedur berechnet man zu gegebener Funktion `f` vom Ergebnistyp `itaylor` garantierte Einschließungen seiner Ableitungen in allen Punkten des Intervalls `x0`.

procedure `derivative` (`function f(t: itaylor) : itaylor;`
(itaylor) `x0 :` ■; **var** `deriv : itaylor`);
■ = `rrGinterval, Linterval, interval;`

Die Ausgangsgröße `deriv : itaylor[0..p]` liefert die Einschließungen der Ableitungen von `f` bis zur Ordnung `p = ub(deriv)`.

8.1.2.3 Programmbeispiele

8.1.2.3.1 Wahl des richtigen Funktionsterms

Die beiden folgenden Programmbeispiele zeigen, daß bei einer vorgegebenen Funktion f die Wahl des richtigen Funktionsterms zur optimalen Einschließung seiner Ableitungen ganz entscheidend sein kann. Dazu betrachten wir zunächst die beiden identischen Funktionen $f(u) \equiv h(u)$ mit $u \in \mathbb{R}$.

$$h(u) = u + \ln \frac{1 + e^u}{e^u} \tag{8.17}$$

$$f(u) = \ln(1 + e^u) \tag{8.18}$$

Im Programm `tstitayl2` wird $h(u)$ direkt durch die rechte Seite von (8.17) realisiert, während $f(u)$ durch die in der Tabelle von Seite 380 angegebene Standardfunktion `ln1p( exp(u) )` definiert ist.

```
program tstitayl2(input,output);
{ Einschliessungen der Ableitungen der Ordg. p fuer: h und f }
use itaylor, rrgi_ari, li_ari;
var p,k  : integer;
   x0    : rrGinterval;                 { Argument-Intervall }
   y     : Linterval;  { Zum Einschliessen der Ableitungen   }
   deriv : itaylor;    { Liefert Einschlg. aller Ableitungen }

function h(u : itaylor): itaylor[0..ub(u)];
begin
   h := u + ln( (1+exp(u))/exp(u) );
end; { h }
function f(u : itaylor): itaylor[0..ub(u)];
begin
   f := ln1p( exp(u) );                 { f = ln( 1+exp(u) ) }
end; { f }
begin
   repeat
      writeln('Order of derivative = ?');  read(p);
      allocate(deriv,0..p);    { Speicherplatz bereitstellen }
      writeln('point of evaluation = ?');  read(x0);

      derivative(h,x0,deriv);
      writeln('Function h:  derivatives of order');
      for k := 0 to p do
      begin      { Liefert Einschliessung der Ableitungen im }
         y := deriv[k];                    { Longreal-Format }
         writeln(k:3,' : ',y);
      end;   writeln;

      derivative(f,x0,deriv);
      writeln('Function f:  derivatives of order');
      for k := 0 to p do
      begin
         y := deriv[k];
         writeln(k:3,' : ',y);
      end;
      free(deriv);          { Speicherplatz wieder freigeben }
   until p<0;
end.
```

Bis zur Ordnung $p = 4$ erhält man für die Ableitungen der Funktionen $h(u), f(u)$ an den Stellen $u_0 = -10^3$ und $u_0 = +10^3$ die folgenden Einschließungen:

```
Order of derivative = ?
4     <enter>
point of evaluation = ?
-1e3  <enter>
Function h:   derivatives of order
0: [-2.73574.............01E-021, 2.73581.............01E-021 ]
1: [-1.36003.............01E-023, 1.36003.............01E-023 ]
2: [-5.85968.............86E-023, 5.84953.............33E-023 ]
3: [-3.33341.............51E-022, 3.33575.............54E-022 ]
4: [-2.35408.............39E-021, 2.35355.............45E-021 ]

Function f:   derivatives of order
0: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
1: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
2: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
3: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
4: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]


Order of derivative = ?
4     <enter>
point of evaluation = ?
1e3   <enter>
Function h:   derivatives of order
0: [ 9.99999999999999999999E+002, 1.00000000000000000001E+003 ]
1: [ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
2: [-3.19582.............43E-023, 3.19582.............43E-023 ]
3: [-1.41719.............96E-022, 1.41719.............96E-022 ]
4: [-8.18431.............75E-022, 8.18431.............75E-022 ]

Function f:   derivatives of order
0: [ 9.99999999999999999999E+002, 1.00000000000000000001E+003 ]
1: [ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
2: [-1.62532.............25E-023, 1.62430.............90E-023 ]
3: [-5.35960.............34E-023, 5.36173.............18E-023 ]
4: [-2.35544.............37E-022, 2.35569.............33E-022 ]
```

Mit dem ungünstig gewählten Funktionsterm der Funktion $h(u)$ erhält man an den Stellen $u_0 = \pm 10^3$ für alle Ableitungen so grobe Einschließungen, daß noch nicht einmal das richtige Vorzeichen bestimmt wird, während der Funktionsterm für $f(u)$ wenigstens für $u_0 = +10^3$ maximalgenaue Einschließungen liefert.

Um auch an der Stelle $u_0 = +10^3$ möglichst maximalgenaue Einschließungen zu erhalten, wird jetzt gezeigt, wie man $f(u) = \ln(1 + e^u)$ unter dem Namen `f1` als Funktion mit dem Ergebnistyp `itaylor` noch besser definieren kann. Dabei greifen wir zurück auf die Rekursionsformel (8.16) von Seite 370 und erhalten mit der Funktion

$$g(u) := \frac{1}{\dfrac{d\,f(u)}{du}} = 1 + e^{-u}, \qquad f(u(x)) = \ln(1 + e^{u(x)})$$

für die Taylorkoeffizienten $(f)_k$ die Rekursionsformel:

$$(f)_k = \frac{1}{(g)_0} \left[(u)_k - \frac{1}{k} \sum_{j=1}^{k-1} j \cdot (f)_j \cdot (g)_{k-j} \right], \quad k \geq 1 \tag{8.19}$$

Das folgende Programm `tstitayl3` zeigt, wie mit Hilfe der Rekursionsformel (8.19) die Funktion `f1` zu deklarieren ist:

```
program tstitayl3(input,output);
{ Einschliessungen der Ableitungen der Ordnung p fuer:    f1 }
use itaylor, rrgi_ari, li_ari;
var p,k  : integer;
   x0    : rrGinterval;                  { Argument-Intervall }
   y     : Linterval;  { Zum Einschliessen der Ableitungen   }
   deriv : itaylor;    { Liefert Einschlg. aller Ableitungen }

function f1 (u : itaylor) : itaylor[0..ub(u)];
{ f1(u) = ln( 1+exp(u) )}
var k,j : integer;
   g,w  : itaylor[0..ub(u)];
begin
   g := 1+exp(-u);
   w[0] := ln1p( exp(u[0]) );
   for k := 1 to ub(u) do
   begin
      w[k] := 0;
```

```
      for j := 1 to k-1 do w[k] := w[k] + j * w[j] * g[k-j];
      w[k] := (u[k]-w[k]/k) / g[0];
   end;
   f1 := w;
end; { f1 }

begin
   repeat
      writeln('Order of derivative = ?');  read(p);
      allocate(deriv,0..p);    { Speicherplatz bereitstellen }
      writeln('point of evaluation = ?');  read(x0);

      derivative(f1,x0,deriv);
      writeln('Function f1:  derivatives of order');
      for k := 0 to p do
      begin
         y := deriv[k];
         writeln(k:3,' : ',y);
      end;

      free(deriv);          { Speicherplatz wieder freigeben }
   until p<0;
end.
```

An den Stellen $u_0 = \pm 10^3$ erhält man jetzt mit der neu deklarierten Funktion `f1` sogar maximalgenaue Einschließungen der Ableitungen von

$$f(u) = \ln(1 + e^u)$$

```
Order of derivative = ?
4     <enter>
point of evaluation = ?
-1e3  <enter>

Function f1:   derivatives of order
0: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
1: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
2: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
3: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
4: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
```

```
Order of derivative = ?
4     <enter>
point of evaluation = ?
1e3   <enter>

Function f1:   derivatives of order
0: [ 9.99999999999999999999E+002, 1.00000000000000000001E+003 ]
1: [ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
2: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
3: [-5.07595889754945676530E-435,-5.07595889754945676529E-435 ]
4: [ 5.07595889754945676529E-435, 5.07595889754945676530E-435 ]
```

Die obige Funktion `f1` wird unter dem Namen `lnexp1` vom Modul `itaylor` exportiert und ist in der Tabelle auf Seite 380 zu finden.

Zusammenfassung:

Funktionsterme sind möglichst einfach zu wählen, wobei die unabhängige Variable so selten wie möglich zu benutzen ist. Es kann sinnvoll sein, bei der Deklaration einer zusammengesetzten Funktion direkt auf die Rekursionsformel ihrer Taylorkoeffizienten zurückzugreifen.

8.1.2.3.2 $\sqrt{1+x^2}$

Realisiert man die Funktion $f(u) = \sqrt{1+u^2}$ als Funktion vom Ergebnistyp `itaylor` direkt mit dem Term $\sqrt{1+u^2}$, so erhält man an der Stelle $u_0 = 10^{13}$ für die Ableitungen der Ordnung $p \geq 2$ schon so grobe Einschließungen, daß noch nicht einmal das richtige Vorzeichen bestimmt werden kann. Benutzt man dagegen die mit $f(u)$ identische Funktion $g(u)$

$$f(u) \equiv g(u) := \begin{cases} +u \cdot \sqrt{1 + \dfrac{1}{u^2}} & \text{falls } u.inf > +80 \\ -u \cdot \sqrt{1 + \dfrac{1}{u^2}} & \text{falls } u.sup < -80 \\ \sqrt{1+u^2} & \text{sonst,} \end{cases}$$

so liefert diese Funktion auch für $|u| \to \infty$ maximalgenaue Einschließungen der entsprechenden Ableitungen.

Das folgende Programm `tstitayl4` liefert für die Funktionen $f(u)$ und $g(u)$ an der Stelle $u_0 = 10^{+13}$ Einschließungen der entsprechenden Ableitungen bis zur Ordnung $p = 4$.

```
program tstitayl4(input,output);
{ Einschliessung der Ableitungen von:    f(u) und g(u)       }
use itaylor, rrgi_ari, li_ari;
var p,k  : integer;
   x0    : rrGinterval;                 { Argument-Intervall }
   y     : Linterval;  { Zum Einschliessen der Ableitungen   }
   deriv : itaylor;    { Liefert Einschlg. aller Ableitungen }

function f( u : itaylor ) : itaylor[0..ub(u)];
begin
   f := sqrt( 1+sqr(u) );
end; { f }
function g( u : itaylor ) : itaylor[0..ub(u)];
const c = 8e1;
var w : itaylor[0..ub(u)];
begin
   if u[0].inf.r1 > c then w := u*sqrt(1+1/sqr(u)) else
      if u[0].sup.r1 <-c then w := -u*sqrt(1+1/sqr(u)) else
   w := sqrt( 1+sqr(u) );
   g := w;
end; { g }

begin
   repeat
      writeln('Order of derivative = ?');  read(p);
      allocate(deriv,0..p);    { Speicherplatz bereitstellen }
      writeln('point of evaluation = ?');  read(x0);

      derivative(f,x0,deriv);
      writeln('Function f:  derivatives of order');
      for k := 0 to p do
      begin
         y := deriv[k];
         writeln(k:3,' : ',y);
      end;
```

```
      derivative(g,x0,deriv);
      writeln('Function g:  derivatives of order');
      for k := 0 to p do
      begin
         y := deriv[k];
         writeln(k:3,' : ',y);
      end;

      free(deriv);          { Speicherplatz wieder freigeben }
   until p<0;
end.
```

Das Programm `tstitayl4` liefert mit den nachfolgenden Eingaben die Bildschirmausgabe:

```
Order of derivative = ?
4     <enter>
point of evaluation = ?
1e13  <enter>

Function f:   derivatives of order
0: [ 9.99999999999999999999E+012, 1.00000000000000000001E+013 ]
1: [ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
2: [-7.54181.............01E-037, 7.56181.............01E-037 ]
3: [-2.26854.............01E-049, 2.26254.............01E-049 ]
4: [-9.05017.............01E-062, 9.04717.............01E-062 ]

Function g:   derivatives of order
0: [ 9.99999999999999999999E+012, 1.00000000000000000001E+013 ]
1: [ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
2: [ 9.99999999999999999999E-040, 1.00000000000000000001E-039 ]
3: [-3.00000000000000000001E-052,-2.99999999999999999999E-052 ]
4: [ 1.19999999999999999999E-064, 1.20000000000000000001E-064 ]
```

Die obige Funktion `g` wird unter dem Namen `sqrt1px2` vom Modul `itaylor` exportiert und ist in der Tabelle auf Seite 380 zu finden.

Zur optimalen Einschließung der Ableitungen einer vorgegebenen Funktion kann es ausreichend sein, nur den Funktionsterm geeignet zu ändern. Falls notwendig, muß die richtige Vorgehensweise von Fall zu Fall entschieden werden!

8.1.2.3.3 Restgliedabschätzungen

Wir betrachten eine beliebig oft differenzierbare Funktion $f : I\!R \to I\!R$, die durch ihre abgebrochene Taylorreihe[4]der Ordnung n in einer Umgebung der Stelle x_0 zu approximieren ist:

$$(8.20) \qquad f(x_0 + h) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} \cdot h^k + R_{n+1}(x_0 + h), \qquad x_0, h \in I\!R$$

$$(8.21) \qquad R_{n+1}(x_0 + h) := \frac{f^{(n+1)}(x_0 + \delta h)}{(n+1)!} \cdot h^{n+1}, \qquad 0 < \delta(h) < 1;$$

Ist $\quad P_n(x_0, h) := \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} \cdot h^k \quad$ das Taylorpolynom der Ordnung n, so gilt für den absoluten Fehler der Approximation:

$$(8.22) \qquad |f(x_0 + h) - P_n(x_0, h)| = |R_{n+1}(x_0 + h)|$$

Betrachtet man für $|h| \le r$ das Approximationsintervall $z := [x_0 - r, x_0 + r]$ mit dem Mittelpunkt x_0, so benötigen wir zur Abschätzung des absoluten Fehlers zunächst eine Einschließung von $f^{(n+1)}(z)/(n+1)!$

$$(8.23) \qquad \frac{f^{(n+1)}(z)}{(n+1)!} \subseteq y,$$

wobei y eine Einschließung des Taylorkoeffizienten der Ordnung $n+1$ von f über dem Intervall z ist. Dieses Intervall y kann nun mit Hilfe der Intervall–Taylorarithmetik auch für größere n direkt berechnet werden.

Bedeutet dann $|y|$ das Intervall aller Betragselemente aus y, so gilt für den absoluten Approximationsfehlers die einfache Abschätzung:

$$(8.24) \qquad |R_{n+1}(x_0 + h)| \le |y|.sup \cdot r^{n+1}$$

und günstige Oberschranken wird man nur dann erhalten, wenn es gelingt, das Intervall y in (8.23) optimal zu berechnen. Wegen der bekannten Intervallüberschätzungen wird dies jedoch nicht erreicht, wenn mit r der Intervalldurchmesser $2r$ von z zu groß gewählt wird. Andererseits ist man natürlich auch an einem nicht zu kleinen Approximationsintervall z interessiert. Die genannten Intervallüberschätzungen bei der Berechnung von y lassen sich jedoch vollständig vermeiden, wenn es gelingt nachzuweisen, daß $f^{(n+1)}(z)$

[4]Die dann eindeutig bestimmte Taylorreihe existiert z.B., wenn f analytisch ist.

über z monoton ist. In diesem Fall berechnet man Einschließungen y_1, y_2 des Taylorkoeffizienten an den Randpunkten $x_0 - r$ und $x_0 + r$:

$$\frac{f^{(n+1)}(x_0 - r)}{(n+1)!} \subseteq y_1, \qquad \frac{f^{(n+1)}(x_0 + r)}{(n+1)!} \subseteq y_2,$$

und bildet die Intervall–Hülle $y_1 +\!* \, y_2$. Daraus folgt dann:

$$\frac{f^{(n+1)}(z)}{(n+1)!} \subseteq y := y_1 +\!* \, y_2,$$

wobei y jetzt optimal ist, da zur Berechnung von y_1, y_2 nur noch die Randpunkte und nicht mehr das ganze Intervall z benutzt wird.

Die vorausgesetzte Monotonie von $f^{(n+1)}(z)$ über z ist dabei garantiert, wenn man nachgewiesen hat, daß in der Einschließung von $f^{(n+2)}(z)$ die Null nicht enthalten ist. Bei zu breitem z kann diese Null jedoch eingefangen werden, ohne daß $f^{(n+2)}(z)$ über dem Intervall z tatsächlich verschwindet. In diesem Fall muß man dann z hinreichend fein unterteilen und den Nachweis $0 \notin f^{(n+2)}(z_j)$ in jedem Teilintervall z_j führen.

Falls jedoch die Monotonie von $f^{(n+1)}(z)$ trotz mehrfacher Intervallunterteilungen nicht nachgewiesen werden kann oder wenn $f^{(n+1)}(z)$ über z tatsächlich nicht monoton ist, so erhält man bei zu breitem Intervall z für den Betrag des Restglieds so große Intervallüberschätzungen, daß die entsprechende Oberschranke viel zu groß ausfällt. Um diese Überschätzung zu vermeiden, muß man das Intervall z in j Intervalle unterteilen und y aus (8.23) für jedes Teilintervall sehr zeitaufwendig einzeln berechnen und von all diesen Einschließungen die Intervallhülle bilden. Die Oberschranke des Restgliedbetrages läßt sich auf ihren optimalen Wert oft nur mit einer sehr feinen Intervallunterteilung ($j > 1000$) drücken.

Mit Hilfe der Prozedur `restglied_S(...)` aus dem folgenden Programm `tstitayl5` kann die Restgliedabschätzung mit dem Werkzeug der Intervall–Taylorarithmetik in der oben beschriebenen Weise vorgenommen werden, wobei `f : itaylor` eine aus den Standardfunktion der Tabelle von Seite 380 beliebig zusammengesetzte Funktion sein kann.

```
program tstitayl5(input,output);
{ Restgliedabschaetzung und Berechnung der          }
{ Taylorkoeffizienten fuer   f = exp(sin(x^2))      }
use itaylor, rrgi_ari, rrgi_util, li_ari;
var s    : Longreal;    { Oberschranke von  R_(n+1) }
   x0,r  : rrG;         { Mittelpunkt und Radius    }
```

```
   n,j,k : integer;
   zz    : rrGinterval; { Approximationsintervall   }
   tayc  : itaylor;     { Taylorkoeffizienten um x0 }

function f(u : itaylor): itaylor[0..ub(u)];
begin
   f := exp( sin(sqr(u)) );
end; { f }

procedure
   restglied_S (function f(u : itaylor) : itaylor;
                 x0,r : rrG;          { Intervall-Mittelpunkt }
                                      { und Intervall-Radius  }
                 n    : integer;      { Taylorpolynomgrad     }
                 j    : integer;      { Teilintervall-Zahl    }
             var zz   : rrGinterval;{ Approximationsintervall }
             var s    : Longreal;   { Oberschranke des        }
                                    { Restgliedes R_(n+1)(x)  }
             var tayc : itaylor );  { Taylorkoeffizienten an  }
                                    { der Stelle x0           }
const kmax =  12; { Maximale Anzahl der Intervallhalbierungen }
                  { zum Monotonienachweis von R_(n+1)         }
var x1,x2,dm        : rrG;
   k                : integer;
   tayl             : itaylor;
   y,y1,y2,zj,z1,z2 : rrGinterval;
   zeroin,fertig    : boolean;
   si               : Linterval;

begin
   writeln('Bitte ein wenig Geduld  ...');
   if n <= 0 then n := 0;
   if j <= 1 then j := 1;
   x1 := x0 -< r;  x2 := x0 +> r;       { Intervallrandpunkte }
   zz := RRGINTVAL(x1,x2);          { liefern z : rrGinterval }
   k := 0;
   dm := diam(zz);
   allocate(tayl,0..n+2);      { Speicherplatz bereitstellen  }
   taylor_K(f,zz,tayl);
   y := tayl[n];               { Fuer den nichtmonotonen Fall }
```

```
zeroin := 0 in tayl[n+2];
zj.inf := x1;

while zeroin and (k<kmax) do
begin       { Untersuchung, ob Restglied in z monoton ist: }
   dm := 0.5 * dm; { Halbierung des Intervall-Durchmessers }
   fertig := false;
   repeat
      if not zeroin then zj.inf := zj.sup;
      zj.sup := zj.inf + dm;
      if zj.sup > x2 then
      begin
         zj.sup := x2;  fertig := true;
      end;
      taylor_K(f,zj,tayl);
      zeroin := 0 in tayl[n+2];
   until zeroin or fertig;
   k := k+1;
end;

if zeroin then
begin { Das Restglied R_(n+1) ist vermutlich nicht monoton }
   dm := (x2-x1) / j;   { Durchmesser der j Teilintervalle }
   zj.sup := x1;
   z1 := RRGINTVAL(x1); { z1 ist Punktintervall zum Start. }
   taylor_K(f,z1,tayl);
   y := tayl[n+1];  fertig := false;
   repeat
      zj.inf := zj.sup;
      zj.sup := zj.inf + dm;  { zj ist neues Teilintervall }
      if zj.sup > x2 then
      begin        { x2 darf nicht ueberschritten werden! }
         zj.sup := x2;  fertig := true;
      end;
      taylor_K(f,zj,tayl);
      y1 := tayl[n+1];
      y := y +* y1; { Intervall-Huelle mit altem Intervall }
   until fertig;
   y := abs(y);
end
```

```
   else
   begin   { Das Restglied ist monoton in z }
      z1 := RRGINTVAL(x1);  { rechter Randpunkt }
      taylor_K(f,z1,tayl);  y1 := tayl[n+1];
      z2 := RRGINTVAL(x2);  { linker  Randpunkt }
      taylor_K(f,z2,tayl);  y2 := tayl[n+1];
      y := abs( y1 +* y2 ); { Intervall-Huelle  }
   end;

   free(tayl);               { Speicherplatz wieder freigeben }
   x1 := 0.5; z1 := x1 *> diam(zz);
   si := y.sup * power(z1,n+1);
   s := si.sup;

   taylor_K(f,RRGINTVAL(x0),tayc);{ Taylorkoeffizienten um x0 }

   if zeroin then
   begin
      write('  Die Oberschranke s kann durch groesseres j ');
      writeln('noch verkleinert werden!');
   end       else
   writeln('  Die Oberschranke s ist optimal!');
end; { restglied_S }

begin
   repeat
      writeln('Entwicklungspunkt x0 = ?');  read(x0);
      writeln('Intervall-Radius  r  = ?');  read(r);
      writeln('Ordnung des Taylorpolynoms = ?');  read(n);
      writeln('Teilintervallzahl j im nichtmonotonen Fall:');
      read(j);
      allocate(tayc,0..n);
      restglied_S(f,x0,r,n,j,zz,s,tayc);
      writeln('Approximationsintervall:');
      writeln(zz);
      writeln('|Restglied| <= ',s);
      write('Gerundete Taylor-Koeffizienten am ');
      writeln('Entwicklungspunkt x0:');
      For k := 0 to n do
      begin
```

```
         r := mid(tayc[k]);
         writeln(k:2,': ',r);
      end;
      free(tayc);
      writeln;
   until n<0;
end.
```

Als Beispiel betrachten wir die Funktion

$$f(x) = e^{\sin(x^2)},$$

die im Intervall $[-0.05, +0.05]$ mit dem Mittelpunkt `x0=0` und dem Radius `r=0.05` durch das Taylorpolynom $P_{14}(x)$ zu approximieren ist:

$$f(x) = e^{\sin(x^2)} = P_{14}(x) + R_{15}(x), \qquad x \in [-0.05, +0.05]$$

$$P_{14}(x) := \sum_{k=0}^{14} \frac{f^{(k)}(0)}{k!} \cdot x^k, \qquad R_{15}(x) := \frac{f^{(15)}(\delta \cdot x)}{15!} \cdot x^{15}, \quad 0 < \delta(x) < 1;$$

Der Aufruf des Programms `tstitayl5` liefert mit den entsprechenden Eingaben die folgende Bildschirmausgabe:

```
Entwicklungspunkt x0 = ?
0        <enter>
Intervall-Radius  r  = ?
0.05     <enter>
Ordnung des Taylorpolynoms = ?
14       <enter>
Teilintervallzahl j im nichtmonotonen Fall:
500      <enter>
Bitte ein wenig Geduld  ...
  Die Oberschranke ist optimal!
Approximationsintervall:
[{ -5.000000000000E-001, -2.500050000001E-026,   -1 },
 {  5.000000000000E-001,  2.500050000001E-026,   -1 } ]
|Restglied| <= 1.31822591423195687572E-022
Gerundete Taylor-Koeffizienten am Entwicklungspunkt x0:
 0: { 1.000000000000E+000,  0.000000000000E+000,  0 }
 1: { 0.000000000000E+000,  0.000000000000E+000,  -2147483647 }
 2: { 1.000000000000E-001,  0.000000000000E+000,  1 }
 3: { 0.000000000000E+000,  0.000000000000E+000,  -2147483647 }
```

```
 4: { 5.000000000000E-001,  0.000000000000E+000,  0 }
 5: { 0.000000000000E+000,  0.000000000000E+000,  -2147483647 }
 6: { 1.666666666667E-004, -3.333333333000E-017,  -23 }
 7: { 0.000000000000E+000,  0.000000000000E+000,  -2147483647 }
 8: {-1.250000000000E-001,  1.666666667000E-027,  0 }
 9: { 0.000000000000E+000,  0.000000000000E+000,  -2147483647 }
10: {-6.666666666667E-001,  3.333333333334E-014,  -1 }
11: { 0.000000000000E+000,  0.000000000000E+000,  -2147483647 }
12: {-4.166666666667E-001,  3.333333333334E-014,  -2 }
13: { 0.000000000000E+000,  0.000000000000E+000,  -2147483647 }
14: { 1.111111111111E-001,  1.111111111112E-014,  -1 }
```

Anmerkungen:

- Das Intervall $z = [x_0 - r, x_0 + r]$ wird durch das Maschinenintervall `zz = [x0 -< r,x0 +> r]` fast optimal eingeschlossen.
- Die Teilintervallzahl `j` wird nur benötigt, wenn eine Monotonie der Ableitung $f^{(n+1)}(x_0 + h)$ für $|h| \leq r$ nicht nachgewiesen werden kann. Bei diesem Beispiel ist $f^{(n+1)}(x_0 + h)$ jedoch monoton, weshalb die Meldung `Die Oberschranke ist optimal!` ausgegeben wird.
- Beachten Sie bitte, daß die Einschließung des Restgliedes nach (8.21) intervallmäßig erfolgt, indem man für den Zwischenwert $\delta(h)$ nicht den festen Wert zwischen 0 und 1 sondern das ganze Intervall $[0, 1]$ zu betrachten hat. Dadurch wird die Oberschranke für den Absolutbetrag des Restgliedes i.a. immer noch größer als notwendig ausfallen, selbst wenn man nachweisen kann, daß $f^{(n+1)}(x)$ für $x \in$ `zz` monoton ist.
- Beim Prozeduraufruf `restglied_S(f,x0,r,n,j,zz,s,tayc)` liefert die Ausgangsgröße `tayc : itaylor` garantierte Einschließungen der Taylorkoeffizienten der Funktion `f` an der Stelle `x0`. Lediglich aus Platzgründen werden oben nur Näherungen dieser Taylorkoeffizienten ausgegeben, die mit Hilfe der Funktion `mid(tayc[k])` berechnet werden, vergl. dazu auch Seite 653.
- Im Intervall `zz = [x0 -< r,x0 +> r]` kann man die Funktion `f` durch das Taylorpolynom der Ordnung 14 approximieren. Dabei ist zu beachten, daß neben der Oberschranke des Restgliedbetrages auch diejenigen Fehler zu berücksichtigen sind, die durch das notwendige Runden der Taylorkoeffizienten und durch das Auswerten des Polynoms entstehen. Auf eine vollständige Fehlerabschätzung wird hier nicht eingegangen.

Die im Programm `tstitayl5` angegebene Prozedur `restglied_S` wird vom Modul `itayl1` exportiert:

```
procedure restglied_S( function f(x : itaylor) : itaylor;
        x0,r : rrG;              n,j : integer;
        var zz : rrGinterval;    var s : Longreal;
        var tayc : itaylor )                          (itayl1)
```

Eingangsgrößen:

- `function f(x : itaylor) : itaylor` ist diejenige Funktion, die im Intervall $[x_0 - r, x_0 + r]$ durch ein Taylorpolynom der Ordnung n mit dem Entwicklungspunkt x_0 zu approximieren ist.
- `x0`: Entwicklungspunkt; `r`: Intervall–Radius;
- `n`: Polynomgrad;
- `j`: Anzahl der Intervalle, in die das Maschinenintervall `zz` unterteilt wird, wenn eine Monotonie von $f^{(n+1)}(x)$ in `zz` nicht nachgewiesen werden kann.

Ausgangsgrößen:

- `zz` Maschinenintervall, welches das Intervall $[x_0 - r, x_0 + r]$ fast optimal einschließt.
- `s`: Garantierte Oberschranke des Restgliedbetrags der Taylorapproximation bzgl. aller Punkte aus `zz`. Damit ist `s` eine Oberschranke des absoluten Approximationsfehlers, wenn `f` in `zz` durch das Taylorpolynom vom Grade `n` approximiert wird. Falls die Monotonie von $f^{(n+1)}(x)$ in `zz` nicht nachgewiesen werden kann, läßt sich `s` durch eine hinreichend große Teilintervallzahl `j` auf sein Minimum reduzieren.
- `tayc`: Vektor vom Typ `itaylor`, dessen Intervall–Komponenten die Taylorkoeffizienten bis zur Ordnung `n` bzgl. des Entwicklungspunktes x_0 einschließen.

Betrachten Sie bitte als Anwendung das Beispiel von Seite 394.

8.1.2.4 Taylorkoeffizienten zum Typ `Linterval`

Die Einschließung der Taylorkoeffizienten einer gegebenen Funktion $f(u(x))$ erfolgt mit Hilfe des vom Modul `litaylor` exportierten Datentyps

```
type litaylor = dynamic array [*] of Linterval;   (litaylor)
```

und einer Funktion `f`, die wie folgt zu deklarieren ist:

```
function f(u : litaylor) : litaylor;
begin
    f := litaylor-Ausdruck;
end;
```

Ist `x0` der vorgegebene Entwicklungspunkt oder das vorgegebene Intervall der Entwicklungspunkte und ist `p` die gewünschte Ordnung, bis zu der die Taylorkoeffizienten zu berechnen sind, so liefert die Prozedur

```
procedure expand ( var u : litaylor; x0 : ■ );   (litaylor)
    ■ = Linterval, interval, Longreal, real
```

mit der Ausgangsgröße `u : litaylor[0..p]` einen Vektor mit den $(p+1)$ Intervall-Komponenten:

$$\begin{array}{lcl} \texttt{u[0]} & = & \texttt{x0} \\ \texttt{u[1]} & = & 1 \\ \texttt{u[j]} & = & 0 \quad \text{für } 2 \le j \le \texttt{ub(u)} = p\,, \end{array}$$

d.h. `u : litaylor[0..p]` liefert die Taylorkoeffizienten der Ordnung 0 bis zur Ordnung `p = ub(u)` für die Funktion $u(x) \equiv x$.

Der `litaylor-Ausdruck` im Anweisungsteil der obigen Funktion `f` läßt sich realisieren mit Hilfe der vom Modul `litaylor` exportierten Operatoren $\boxed{+,-,*,/}$ mit dem Ergebnistyp `litaylor`, wobei zunächst beide Operanden vom Typ `litaylor` sein dürfen. Ist jedoch nur ein Operand vom Typ `litaylor`, so darf der andere Operand die folgenden Datentypen annehmen:

```
integer, real, Longreal, Linterval
```

Die beiden Anweisungen `expand(u,x0);` `y := f(u)` liefern dann mit `y : litaylor[0..p]` Einschließungen der Taylorkoeffizienten bis zur Ordnung `p` der Funktion `f` für alle Entwicklungspunkte aus dem Intervall `x0`.

Mit Hilfe der folgenden Prozedur berechnet man zu gegebener Funktion `f` vom Ergebnistyp `litaylor` garantierte Einschließungen seiner Taylorkoeffizienten zu Entwicklungspunkten aus dem Intervall `x0`.

```
procedure  taylor_K (function f(t : litaylor): litaylor;
(litaylor)              x0 : ■; var taylor : litaylor );
                  ■ = Linterval, interval;
```

Die Ausgangsgröße `taylor : litaylor[0..p]` liefert die Einschließungen der Taylorkoeffizienten von `f` bis zur Ordnung `p = ub(taylor)`.

Mit Hilfe der folgenden Prozedur berechnet man zu gegebener Funktion `f` vom Ergebnistyp `litaylor` garantierte Einschließungen seiner Ableitungen in allen Punkten des Intervalls `x0`.

```
procedure  derivative (function f(t: litaylor): litaylor;
(litaylor)                x0 : ■; var deriv : litaylor );
                  ■ = Linterval, interval;
```

Die Ausgangsgröße `deriv : litaylor[0..p]` liefert die Einschließungen der Ableitungen von `f` bis zur Ordnung `p = ub(deriv)`.

Sind z.B. für die Funktion $f(x) = x/(1+x^2)$ Taylorkoeffizienten oder Ableitungen höherer Ordnung zu berechnen, so ist f wie folgt zu deklarieren:

```
function f(u : itaylor) : itaylor[0..ub(u)];
begin
   f := u / (1 + u*u);
end; { f }
```

Mit Hilfe der vier Grundoperationen kann f aus den Standardfunktionen der nachfolgenden Tabelle zusammengesetzt werden.

Standardfunktionen vom Typ `litaylor` (litaylor) `u : litaylor;` `a : Linterval;`			
Funktion	**Aufruf**	**Funktion**	**Aufruf**
u^2	`sqr(u)`	$\tan(u)$	`tan(u)`
$\sqrt{u}$	`sqrt(u)`	$\cot(u)$	`cot(u)`
$\sqrt{1+u^2}$	`sqrt1px2(u)`	$\sin(\pi u)$	`sinpi(u)`
$\sqrt{1-u^2}$	`sqrt1mx2(u)`	$\cos(\pi u)$	`cospi(u)`
$\sqrt{u^2-1}$	`sqrtx2m1(u)`	$\tan(\pi u)$	`tanpi(u)`
$\sqrt{1+u}-1$	`sqrt1pm1(u)`	$\cot(\pi u)$	`cotpi(u)`
e^u	`exp(u)`	$\arcsin(u)$	`arcsin(u)`
e^u-1	`expm1(u)`	$\arccos(u)$	`arccos(u)`
2^u	`exp2(u)`	$\arctan(u)$	`arctan(u)`
10^u	`exp10(u)`	$\mathrm{arccot}(u)$	`arccot(u)`
u^a	`power(u,a)`	$\sinh(u)$	`sinh(u)`
u^u	`power(u)`	$\cosh(u)$	`cosh(u)`
$\ln(u)$	`ln(u)`	$\tanh(u)$	`tanh(u)`
$\ln(e \cdot u)$	`ln_ex(u)`	$\coth(u)$	`coth(u)`
$\ln(1+u)$	`ln1p(u)`	$\mathrm{arsinh}(u)$	`arsinh(u)`
$\ln(1+e^u)$	`lnexp1(u)`	$\mathrm{arcosh}(u)$	`arcosh(u)`
$u \cdot \ln(u)$	`xlnx(u)`	$\mathrm{arcosh}(1+u)$	`arcosh1p(u)`
$\log_2(u)$	`log2(u)`	$\mathrm{artanh}(u)$	`artanh(u)`
$\log_{10}(u)$	`log10(u)`	$\mathrm{arcoth}(u)$	`arcoth(u)`
$\log_{10}(1+u)$	`log1p(u)`		
$\log_a(u)$	`loga(a,u)`		
$\sin(u)$	`sin(u)`		
$\cos(u)$	`cos(u)`		

8.1.2.4.1 Programmbeispiel

Mit Hilfe des nachfolgenden Programms `tstlitayl1` werden zur Funktion $f(x) = x/(1+x^2)$ die Taylorkoeffizienten an der Stelle `x0` bis zur Ordnung p eingeschlossen:

```
program tstlitayl1(input,output);
{ Einschliessung der Taylorkoeffizienten von f=x/(1+x*x) }
use litaylor, li_ari;

var p,k : integer;
   x0   : Linterval;   { Entwicklungspunkte-Intervall    }
   h    : Linterval;   { Ausgabe der Tayl.-Koeffizienten }
   y,u  : litaylor;
function f(u : litaylor) : litaylor[0..ub(u)];
begin
   f := u / (1 + u*u);
end; { f }

begin
   repeat
      writeln('Order of derivative = ?');   read(p);
      allocate(u,0..p); { Speicherplatz der dynamischen  }
      allocate(y,0..p); { Variablen   u,y   anlegen.     }
      writeln('point of evaluation = ?');  read(x0);
      expand(u,x0);     { Eingangsvariable u realisieren }
      y := f(u);
      writeln('Ausgabe der Taylor-Koeffizienten:');
      for k := 0 to p do
      begin
         h := y[k];     { Ausgabe der Einschliessung der }
         writeln(k:3,' :   ',h);   { Taylorkoeffizienten }
      end;
      free(u); free(y); { Speicher fuer u,y freigeben.   }
      writeln;
   until p<0;
end.
```

Mit den entsprechenden Eingabedaten für $p = 40$ und $x0 = 1$ erhält man für das Programm `tstlitayl1` das folgende Ablaufprotokoll:

```
Order of derivative = ?
40
point of evaluation = ?
1
Ausgabe der Taylor-Koeffizienten:

  0 :   [  5.0000000000000000000E-001,  5.0000000000000000000E-001 ]
  1 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
  2 :   [ -2.5000000000000000000E-001, -2.5000000000000000000E-001 ]
  3 :   [  2.5000000000000000000E-001,  2.5000000000000000000E-001 ]
  4 :   [ -1.2500000000000000000E-001, -1.2500000000000000000E-001 ]
  5 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
  6 :   [  6.2500000000000000000E-002,  6.2500000000000000000E-002 ]
  7 :   [ -6.2500000000000000000E-002, -6.2500000000000000000E-002 ]
  8 :   [  3.1250000000000000000E-002,  3.1250000000000000000E-002 ]
  9 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
 10 :   [ -1.5625000000000000000E-002, -1.5625000000000000000E-002 ]
 11 :   [  1.5625000000000000000E-002,  1.5625000000000000000E-002 ]
 12 :   [ -7.8125000000000000000E-003, -7.8125000000000000000E-003 ]
 13 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
 14 :   [  3.9062500000000000000E-003,  3.9062500000000000000E-003 ]
 15 :   [ -3.9062500000000000000E-003, -3.9062500000000000000E-003 ]
 16 :   [  1.9531250000000000000E-003,  1.9531250000000000000E-003 ]
 17 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
 18 :   [ -9.7656250000000000000E-004, -9.7656250000000000000E-004 ]
 19 :   [  9.7656250000000000000E-004,  9.7656250000000000000E-004 ]
 20 :   [ -4.8828125000000000000E-004, -4.8828125000000000000E-004 ]
 21 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
 22 :   [  2.4414062500000000000E-004,  2.4414062500000000000E-004 ]
 23 :   [ -2.4414062500000000000E-004, -2.4414062500000000000E-004 ]
 24 :   [  1.2207031250000000000E-004,  1.2207031250000000000E-004 ]
 25 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
 26 :   [ -6.1035156250000000000E-005, -6.1035156250000000000E-005 ]
 27 :   [  6.1035156250000000000E-005,  6.1035156250000000000E-005 ]
 28 :   [ -3.0517578125000000000E-005, -3.0517578125000000000E-005 ]
 29 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
 30 :   [  1.5258789062500000000E-005,  1.5258789062500000000E-005 ]
 31 :   [ -1.5258789062500000000E-005, -1.5258789062500000000E-005 ]
 32 :   [  7.6293945312500000000E-006,  7.6293945312500000000E-006 ]
 33 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
 34 :   [ -3.8146972656250000000E-006, -3.8146972656250000000E-006 ]
 35 :   [  3.8146972656250000000E-006,  3.8146972656250000000E-006 ]
 36 :   [ -1.9073486328125000000E-006, -1.9073486328125000000E-006 ]
 37 :   [ -0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
 38 :   [  9.5367431640625000000E-007,  9.5367431640625000000E-007 ]
 39 :   [ -9.5367431640625000000E-007, -9.5367431640625000000E-007 ]
 40 :   [  4.7683715820312500000E-007,  4.7683715820312500000E-007 ]
```

8.2 Reelle Funktionen mit mehreren Variablen

In diesem Abschnitt werden Routinen angegeben, mit denen der Gradient einer einzelnen Funktion mehrerer Veränderlicher an einer einzelnen Stelle garaniert eingeschlossen werden kann. Sind jedoch n Funktionen gegeben, die jeweils von n unabhängigen Variablen abhängen, so läßt sich zu dieser vektorwertigen Funktion ihre $(n \times n)$-Jacobimatrix ebenfalls an einzelnen Stellen garantiert einschließen. Vorausgesetzt wird dabei lediglich, daß die partiellen Ableitungen alle existieren und stetig sind. Die Funktionen lassen sich mit Hilfe der Grundoperationen und den Intervallstandardfunktionen vom Typ `rrGinterval` zusammensetzen, so daß die partiellen Ableitungen mit sehr engen Intervallen eingeschlossen werden, die i.a. maximalgenau in Intervalle vom Typ `Linterval` gerundet werden können. Gradienten und Jacobi–Matrizen werden für die beiden genannten Datentypen berechnet.

Die gesuchten Einschließungen der partiellen Ableitungen liefert eine in [5, Chapter 12] ausführlich beschriebene Differentiations–Arithmetik, deren Kenntnis für die praktische Anwendung jedoch nicht erforderlich ist. Anhand mehrerer Beispiele wird die wirklich einfache Anwendung der entsprechenden Routinen demonstriert.

8.2.1 Gradientenberechnung

Gegeben sei eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ mit n unabhängigen Variablen $x_i,\ \ i = 1, 2, ..., n$. Alle n partiellen Ableitungen $\dfrac{\partial f}{\partial x_i}$ seien stetig in einem Definitionsbereich $D_f \subset \mathbb{R}^n$. Der Gradient $\nabla f(x)$ ist definiert durch:

$$\nabla f(x_1, x_2, \ldots, x_n) \equiv \nabla f(x) \equiv \nabla f := \begin{pmatrix} \dfrac{\partial f}{\partial x_1}(x) \\ \dfrac{\partial f}{\partial x_2}(x) \\ \vdots \\ \dfrac{\partial f}{\partial x_n}(x) \end{pmatrix}$$

Im nächsten Abschnitt wird beschrieben, wie man in einem **PASCAL-XSC** Programm die Funktion $f(x)$ darzustellen hat und wie der Vektor ∇f einzuschließen ist.

8.2.1.1 Routinen zur Gradientenberechnung

Zur Darstellung der Funktion $f(x)$ werden bzgl. des Intervall–Datentyps `rrGinterval` vom Hilfsmodul `igrad` die folgenden Datentypen exportiert:

```
type GradType = dynamic array [*] of rrGinterval;
```

```
type GTvector = dynamic array [*] of GradType;
```

Zur Einschließung von $f(x)$ oder $\nabla f(x)$ durch Intervalle vom Typ `Linterval` oder `rrGinterval` ist z.B.

$$f(x_1, x_2) = 2x_1^2 \cdot \cos(\pi \cdot x_1 x_2)$$

als Funktion zweier Variabler in **PASCAL-XSC** wie folgt zu deklarieren:

```
function f(x : GTvector) : GradType[0..2];
begin
    f := 2 * sqr(x[1]) * cospi( x[1]*x[2] );
end;
```

Die Darstellung von f ist also denkbar einfach und entspricht genau der mathematischen Notation!

Im Ausdruck für f sind zulässig die vier Grundoperationen $\{+, -, *, /\}$ mit Operanden vom Typ

```
integer, real, Longreal, rrG, rrGinterval
```

wobei für die Operanden auch alle Standardfunktionen der folgenden Tabelle gewählt werden können.

Standardfunktionen vom Typ `GradType` (igrad) **zur Berechnung von Gradienten und Jacobi-Matrizen** `u : GradType; a : Longreal; k : rrGinterval; n : integer;`			
Funktionsterm	**Aufruf**	**Funktionsterm**	**Aufruf**
u^2	`sqr(u)`	$\sin(u)$	`sin(u)`
$\sqrt{u}$	`sqrt(u)`	$\cos(u)$	`cos(u)`
$\sqrt{1+u}-1$	`sqrt1pm1(u)`	$\tan(u)$	`tan(u)`
$\sqrt{1-u^2}$	`sqrt1mx2(u)`	$\cot(u)$	`cot(u)`
$\sqrt{u^2-1}$	`sqrtx2m1(u)`	$\sin(\pi u)$	`sinpi(u)`
$\sqrt{1+u^2}$	`sqrt1px2(u)`	$\cos(\pi u)$	`cospi(u)`
e^u	`exp(u)`	$\tan(\pi u)$	`tanpi(u)`
e^u-1	`expm1(u)`	$\cot(\pi u)$	`cotpi(u)`
e^{-u^2}	`exp_x2(u)`	$\arcsin(u)$	`arcsin(u)`
2^u	`exp2(u)`	$\arccos(u)$	`arccos(u)`
10^u	`exp10(u)`	$\arctan(u)$	`arctan(u)`
u^u	`power(u)`	$\operatorname{arccot}(u)$	`arccot(u)`
$u^n,\ n=0,1,...$	`power(u,n)`	$\sinh(u)$	`sinh(u)`
u^k	`power(u,k)`	$\cosh(u)$	`cosh(u)`
$\ln(u)$	`ln(u)`	$\tanh(u)$	`tanh(u)`
$\ln(1+u)$	`ln1p(u)`	$\coth(u)$	`coth(u)`
$\ln(e\cdot u)$	`ln_ex(u)`	$\operatorname{arsinh}(u)$	`arsinh(u)`
$u\cdot\ln(u)$	`xlnx(u)`	$\operatorname{arcosh}(u)$	`arcosh(u)`
$\log_2(u)$	`log2(u)`	$\operatorname{arcosh}(1+u)$	`arcosh1p(u)`
$\log_{10}(u)$	`log10(u)`	$\operatorname{artanh}(u)$	`artanh(u)`
$\log_{10}(1+u)$	`log1p(u)`	$\operatorname{arcoth}(u)$	`arcoth(u)`
$\log_a(u)$	`loga(a,u)`		

Zur Einschließung nur des Funktionswertes $f(a) \in$ `fa` an der festen Stelle $a = (a_1, a_2, \ldots, a_n)$ werden vom Modul `igrad` die beiden folgenden Prozeduren exportiert:

```
procedure fEvalG(
  function f(x : GTvector) : GradType[lb(x,2)..ub(x,2)];
  a : Livector;
  var fa : Linterval );                                  (igrad)
```

```
procedure fEvalG_rrG(
  function f(x : GTvector) : GradType[lb(x,2)..ub(x,2)];
  a : rrGivector;
  var fa : rrGinterval );                                (igrad)
```

Dabei liefert die Prozedur `fEvalG_rrG` mit `fa` eine garantierte Einschließung von $f(a)$ im feineren `rrG`–Format, während die Prozedur `fEvalG` mit `fa` eine i.a. maximalgenaue Einschließung von $f(a)$ im gröberen `Longreal`–Format berechnet. Beachten Sie bitte, daß in beiden Prozeduren die interne Funktionsauswertung im 26-stelligen `rrG`–Format erfolgt und daß `fEvalG` die Ergebnisse ins `Longreal`–Format rundet, in dem praktisch alle späteren Anwendungen durchgeführt werden.

Zur Einschließung des Funktionswertes $f(a) \in$ `fa` und des Gradienten $\nabla f(a) \in$ `ga` an der festen Stelle $a = (a_1, a_2, \ldots, a_n)$ werden vom Modul `igrad` die beiden folgenden Prozeduren exportiert:

```
procedure fgEvalG(
  function f(x : GTvector) : GradType[lb(x,2)..ub(x,2)];
  a : Livector;
  var fa : Linterval;  var ga : Livector );              (igrad)
```

```
procedure fgEvalG_rrG(
  function f(x : GTvector) : GradType[lb(x,2)..ub(x,2)];
  a : rrGivector;
  var fa : rrGinterval; var ga : rrGivector );           (igrad)
```

Dabei liefert `fgEvalG_rrG` mit `fa` und `ga` garantierte Einschließungen von $f(a)$ und $\nabla f(a)$ im feineren `rrG`–Format, während die Prozedur `fgEvalG` mit `fa` und `ga` i.a. maximalgenaue Einschließungen von $f(a)$ und $\nabla f(a)$ im gröberen `Longreal`–Format berechnet.

Bei der Entwicklung eigener **PASCAL-XSC** Programme ist bezüglich der Indexbereiche der Datentypen `GradType` und `GTvector` folgende Regel zu beachten:

> Ist n die Anzahl der unabhängigen Variablen, so müssen die Indexbereiche der Datentypen wie folgt gewählt werden:
>
> `GTvector [1..n,0..n];`   `GradType [0..n];`

Die nachfolgenden Beispiele zeigen, wie einfach in einer **PASCAL-XSC** Umgebung die Gradientenberechnung durchgeführt werden kann.

8.2.1.2 Programmbeispiele

8.2.1.2.1 $f(x_1, x_2) = 2x_1^2 \cdot \cos(\pi \cdot x_1 x_2)$

Die Funktion

$$f(x) = f(x_1, x_2) = 2x_1^2 \cdot \cos(\pi \cdot x_1 x_2)$$

soll an der Stelle $x_0 = (1, 2)$ ausgewertet und der Gradient $\nabla f(x)$ an der gleichen Stelle berechnet werden. Es gilt:

$$f(x_0) = 2; \qquad \nabla f(x_0) = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$$

Die folgenden Programme `tstigrad1` und `tstigrad2` liefern Einschließungen dieser Werte, wobei das erste Programm den Term $\cos(\pi \cdot x_1 x_2)$ mit Hilfe der Funktion `cospi(x[1]*x[2])` aus der Tabelle von Seite 404 auswertet, während das Programm `tstigrad2` die Funktion `cos(Pi_rrGi*x[1]*x[2])` benutzt. Der auf der Maschine nicht darstellbare Wert π wird dabei durch das Intervall `Pi_rrGi` vom Typ `rrGinterval` eingeschlossen, vergl. dazu die Tabelle auf Seite 652.

```
program tstigrad1(input,output);
{ Berechnet fuer die Formel    f = 2*x1^2*cos(Pi*x1*x2)    }
{ f(x) und gradient[f(x)] an der Stelle:  x = (a[1],a[2]) }
```

```
use igrad,          { exportiert u.a. fgEvalG(f,a,fa,ga)   }
    li_ari;         { Longreal-Intervall-Arithmetik         }
const n      = 2; { Anzahl der unabhaengigen Variablen   }

var a  : Livector[1..n];      { Argument-Intervallvektor  }
    fa : Linterval;           { Ergebnis-Intervall fuer f }
    ga : Livector[1..n];      { Gradient-Einschliessung   }
    k  : integer;
function f(x : GTvector) : GradType[0..n];
begin
   f := 2 * sqr(x[1]) * cospi(x[2]*x[1]);
end;
begin
   writeln('    f(x1,x2) = 2*x1^2*cos(Pi*x1*x2)');
   repeat
      writeln('Eingabe der Auswertestelle:');
      for k := 1 to n do read(a[k]);
      fgEvalG(f,a,fa,ga);
      writeln('Einschliessung des Funktionswertes:');
      writeln(fa); writeln;
      writeln('Einschliessung des Gradienten:');
      for k := 1 to n do writeln(ga[k]);
      writeln;
   until false;
end.
```

Mit der Eingabe des zweidimensionalen Punktintervalls $([1,1],[2,2])$ durch `1 2 <enter>` liefert das obige Programm die folgende Bildschirmausgabe:

```
    f(x1,x2) = 2*x1^2*cos(Pi*x1*x2)
Eingabe der Auswertestelle:
1 2     <enter>
Einschliessung des Funktionswertes:
[  2.0000000000000000000E+000,  2.0000000000000000000E+000 ]

Einschliessung des Gradienten:
[  3.9999999999999999999E+000,  4.0000000000000000001E+000 ]
[  0.0000000000000000000E+000,  0.0000000000000000000E+000 ]
```

Mit der Eingabe des gleichen, zweidimensionalen Punktintervall-Vektors $([1,1],[2,2])$ durch `1 2 <enter>` liefert das folgende Programm `tstigrad2`

```
program tstigrad2(input,output);
{ Berechnet fuer die Formel g = 2*x1^2*cos(Pi*x1*x2)      }
{ g(x) und gradient[g(x)] an der Stelle:  a = (a[1],a[2]) }
use type_def,      { exportiert u.a. Type rrGivector       }
    igrad,         { exportiert u.a. fgEvalG(g,a,ga,gra)  }
    rrgi_ari,      { Intervall-Arithmetik mit dem Typ rrG }
    rrgi_util,     { Exportiert PI_rrGi: Einschliessung   }
                   { von Pi im rrG-Format.                }
    li_ari;        { Longreal-Intervall-Arithmetik        }

const n = 2;       { Anzahl der unabhaengigen Variablen   }

var a   : rrGivector[1..n];   { Argument-Intervallvektor  }
    ga  : rrGinterval;        { Ergebnis-Intervall fuer g }
    gra : rrGivector[1..n];   { Gradient-Einschliessung   }
    k   : integer;

function g(x : GTvector) : GradType[0..n];
begin
   g := 2 * sqr(x[1]) * cos( PI_rrGi*x[1]*x[2] );
end;

begin
   writeln('    f(x1,x2) = 2*x1^2*cos(Pi*x1*x2)');
   repeat
      writeln('Eingabe der Auswertestelle:');
      for k := 1 to n do read(a[k]);
      fgEvalG_rrG(g,a,ga,gra);
      writeln('Einschliessung des Funktionswertes:');
      writeln(ga); writeln;
      writeln('Einschliessung des Gradienten:');
      for k := 1 to n do writeln(gra[k]);
      writeln;
   until false;
end.
```

die folgende Bildschirmausgabe:

```
    f(x1,x2) = 2*x1^2*cos(Pi*x1*x2)
Eingabe der Auswertestelle:
1 2     <enter>
Einschliessung des Funktionswertes:
[ {  2.000000000000E-001, -2.648772746553E-025,    1 },
  {  2.000000000000E-002,  0.000000000000E+000,    2 } ]

Einschliessung des Gradienten:
[ {  4.000000000000E-001, -1.597087791183E-024,    1 },
  {  4.000000000000E-001,  1.234614268714E-024,    1 } ]
[ { -5.236664209357E-001, -1.711773568830E-014,    -23 },
  {  6.073069343566E-001,  8.448110865038E-015,    -23 } ]
```

Anmerkungen:

- Beachten Sie bitte, daß in beiden Programmen sowohl der Funktionswert als auch der Gradient von $f(x)$ an der Stelle $(1,2)$ im `rrG`–Format berechnet werden und daß nur im ersten Programm `tstigrad1` die Ergebnisintervalle maximalgenau ins gröbere `Longreal`–Format gerundet werden.

- Die Wahl der Funktion `cospi(x[1]*x[2])` liefert Punktintervalle für die Einschließung der exakten Werte 2 und 0. Wählt man jedoch im Programm `tstigrad2` die Funktion `cos(Pi_rrGi*x[1]*x[2])`, so erhält man wegen des von Null verschiedenen Durchmessers von `Pi_rrGi` $\ni \pi$ für die Werte 2 und 0 keine Punktintervalle mehr.

Das folgende Programmbeispiel zeigt noch deutlicher, daß die geeignete Wahl der Funktionen aus der Tabelle von Seite 404 für die Güte der gesuchten Einschließungen oft von entscheidender Bedeutung ist.

8.2.1.2.2 Wahl des richtigen Funktionsterms

Wir betrachten jetzt die Funktionen

$$\begin{aligned}
g(x) &:= \frac{2}{3}\cdot(1+x)^{3/2} - x \qquad \text{und} \\
f(x_1,x_2) &:= x_1\cdot g(x_2) \equiv x_1\left\{\frac{2}{3}\cdot(1+x_2)^{3/2} - x_2\right\} \qquad \text{mit:} \\
\frac{\partial f}{\partial x_1} &= g(x_2); \qquad \frac{\partial f}{\partial x_2} = x_1\cdot\frac{d\,g(x_2)}{d\,x_2} = x_1\cdot\left\{\sqrt{1+x_2}-1\right\}
\end{aligned}$$

und erkennen, daß $\frac{\partial f}{\partial x_2}$ für $x_2 \to 0$ wegen zu starker Auslöschungseffekte im Ableitungsterm $\frac{d\,g(x_2)}{d\,x_2} = \sqrt{1+x_2}-1$ nur mit erheblichen Überschätzungen auszuwerten ist. Diese Überschätzungen kann man jedoch ganz vermeiden, wenn man die Funktion $g(x)$ analog zu den Standardfunktionen aus der Tabelle von Seite 404 wie folgt deklariert und dabei die Ableitung $\frac{d\,g(x_2)}{d\,x_2}$ mit Hilfe der schon vordefinierten Funktion `sqrt1pm1(x[2])` darstellt:

$$\frac{d\,g(x_2)}{d\,x_2} = \left\{\sqrt{1+x_2}-1\right\} \in \texttt{sqrt1pm1}(\texttt{x}[2])$$

```
function g(var u : rrG_GradType) : rrG_GradType[0..ub(u)];
var h,ex : rrGinterval;
   i     : integer;
begin
   ex := rrGintval(1.5);
   g[0] := 2*power(1+u[0],ex)/3 - u[0];      { Funktionswert }
   h := sqrt1pm1(u[0]);  { h: 1. Ableitung durch sqrt1pm1(u) }
   For i := 1 to ub(u) do g[i] := h*u[i];
end;
```

Vergleichen Sie dazu bitte auch die Elementarfunktionen in [5, ab Seite 250]. Das nachfolgende Programm `tstigrad3` berechnet mit Hilfe dieser Funktion `g` jetzt sogar maximalgenaue Einschließungen des exakten Funktionswertes $f(x)$ und des Gradienten $\nabla f(x)$ an der Stelle $x_0 = (3, 10^{-60})$:

$$\begin{aligned}
f(3,10^{-60}) &= 2(1+10^{-60})^{3/2} - 3\cdot 10^{-60} \approx 2 \\
\frac{\partial f}{\partial x_1}(3,10^{-60}) &= \frac{2}{3}(1+10^{-60})^{3/2} - 10^{-60} \approx \frac{2}{3} \\
\frac{\partial f}{\partial x_2}(3,10^{-60}) &= 3\left[\sqrt{1+10^{-60}}-1\right] \approx 1.5\cdot 10^{-60}
\end{aligned}$$

```
program tstigrad3(input,output);
{ Berechnet fuer die Formel:                                  }
{          f = x1*( (2/3)*power(1+x2,1.5)-x2 )               }
{ f(x) und gradient[f(x)] an der Stelle:  x = (a[1],a[2]) }
use igrad,         { exportiert u.a. fgEvalG(f,a,fa,ga)   }
    rrgi_ari,      { Intervall-Arithmetik mit dem Typ rrG }
    rrgi_util,     { exportiert zusaetzliche Operatoren   }
    li_ari;        { Longreal-Intervall-Arithmetik        }

const n = 2;       { Anzahl der unabhaengigen Variablen   }

var a  : Livector[1..n];      { Argument-Intervallvektor  }
    fa : Linterval;           { Ergebnis-Intervall fuer f }
    ga : Livector[1..n];      { Gradient-Einschliessung   }
    k  : integer;

function g(u : GradType) : GradType[0..ub(u)];
{ Hilfsfunktion g analog zum Toolbox Band I ab Seite 250. }
{ g(u) = (2/3)*power(1+u,1.5)-u;                           }
var h,ex : rrGinterval;
   i     : integer;
begin
   ex := rrGintval(1.5);
   g[0] := 2*power(1+u[0],ex)/3 - u[0];      { Funktionswert }
   h := sqrt1pm1(u[0]);  { h: 1. Ableitung durch sqrt1pm1(u) }
   For i := 1 to ub(u) do g[i] := h*u[i];
end;

function f(x : GTvector) : GradType[0..n];
begin
   f := x[1] * g(x[2]);
end;

begin
   writeln('   f(x1,x2) = x1*{ (2/3)*power(1+x_2,1.5)-x_2 }');
   repeat
      writeln('Eingabe der Auswertestelle:');
      for k := 1 to n do read(a[k]);
      fgEvalG(f,a,fa,ga);
      writeln('Einschliessung des Funktionswertes:');
```

```
      writeln(fa); writeln;
      writeln('Einschliessung des Gradienten:');
      for k := 1 to n do writeln(ga[k]);     writeln;
   until false;
end.
```

Die Eingabe der Werte `3 1e-60 <enter>` liefert die Bildschirmausgabe:

```
    f(x1,x2) = x1*{ (2/3)*power(1+x2,1.5)-x2 }
Eingabe der Auswertestelle:
3 1e-60     <enter>
Einschliessung des Funktionswertes:
[  1.9999999999999999999E+000,  2.0000000000000000001E+000 ]

Einschliessung des Gradienten:
[  6.6666666666666666666E-001,  6.6666666666666666667E-001 ]
[  1.4999999999999999999E-060,  1.5000000000000000001E-060 ]
```

Anmerkungen:

- Schreiben Sie in Anlehnung an `tstigrad3` ein Programm, in dem die Funktion

$$f(x_1, x_2) = x_1 \left\{ \frac{2}{3} \cdot (1 + x_2)^{2/3} - x_2 \right\}$$

wie folgt ohne die Hilfsfunktion $g(x)$ deklariert wird:

```
function f(x : GTvector) : GradType[0..n];
var ex : rrGinterval;
begin
   ex := rrGintval(1.5);
   f  := x[1] * ( 2*power(1+x[2],ex)/3 - x[2] );
end;
```

und berechnen Sie damit $f(x)$ und $\nabla f(x)$ an der gleichen Stelle $x_0 = (3, 10^{-60})$. Die Einschließung der zweiten Gradientenkomponente fällt mit der obigen Deklaration von $f(x)$ so grob aus, daß noch nicht einmal das richtige Vorzeichen des exakten Wertes $3(\sqrt{1 + 10^{-60}} - 1)$ bestimmt wird.

- Falls Sie eigene Hilfsfunktionen wie $g(x)$ von Seite 411 deklarieren, beachten Sie bitte vorher die Beispiele in [5, ab Seite 250] und testen Sie Ihre Funktionen recht sorgfältig, denn mit falsch deklarierten Hilfsfunktionen erhalten Sie natürlich keine garantierten Einschließungen Ihrer Gradientenwerte!

- Intervallüberschätzungen werden in der Praxis immer dann auftreten, wenn Gradientenkomponenten in der Nähe der Zahl Null durch Addition bzw. Subtraktion zustandekommen. In solchen Fällen erhält man ausreichende Genauigkeiten nur durch Rechnungen mit hinreichend hoher Stellenzahl. Daher ist es auch sinnvoll, die Gradientenberechnungen im 26-stelligen `rrG`–Format vorzunehmen und die Ergebnisse danach ins 21-stellige `Longreal`–Format zu runden, in dem dann die eigentlichen praktischen Anwendungen durchgeführt werden.

8.2.2 Berechnung der Jacobi–Matrix

Gegeben seien n reelle Funktionen $f_i(x) : I\!R^n \to I\!R,\ i = 1, 2, \ldots, n$ mit n unabhängigen Variablen $x = (x_1, x_2, \ldots, x_n)$:

$$\begin{pmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{pmatrix} = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix} = \boldsymbol{f}(x) : I\!R^n \to I\!R^n$$

Setzt man dabei voraus, daß alle Funktionen $f_i(x)$ partielle Ableitungen der Ordnung 1 besitzen, so wird die Jacobi–Matrix $\boldsymbol{J_f}(x)$ definiert durch:

$$\boldsymbol{J_f}(x) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \ldots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \ldots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \ldots & \frac{\partial f_n}{\partial x_n}(x) \end{pmatrix}$$

8.2.2.1 Routinen zur Jacobimatrix–Berechnung

Zur Darstellung der Funktion $\boldsymbol{f}(x)$ werden bzgl. der Intervall–Datentypen `rrGinterval` und `Linterval` vom Modul `igrad` die folgenden Datentypen exportiert:

```
type GradType = dynamic array [*] of rrGinterval;
```

```
type GTvector = dynamic array [*] of GradType;
```

Um $\boldsymbol{f}(x)$ oder $\boldsymbol{J_f}(x)$ durch Intervalle vom Typ `rrGinterval` oder `Linterval`[5] einzuschließen, sind z.B. die Funktionen f_1, f_2

$$\boldsymbol{f}(x) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} x_1^2 + \sin(x_1 x_2) \\ x_1 \cdot \arcsin(x_2) \end{pmatrix}$$

als Funktion zweier Variabler in **PASCAL-XSC** wie folgt zu deklarieren:

```
function f(x : GTvector) : GradType[0..2];
begin
    f[1] := sqr(x[1]) + sin( x[1]*x[2] );
    f[2] := x[1] * arcsin(x[2]);
end;
```

Die Darstellung von $\boldsymbol{f}(x)$ ist also denkbar einfach und entspricht genau der sonst üblichen mathematischen Notation!

In den Ausdrücken für die Funktionen $f_1(x), f_2(x)$ sind zulässig die vier Grundoperationen $\{+, -, *, /\}$ mit Operanden vom Typ

```
integer, real, Longreal, rrG, rrGinterval
```

[5]Beachten Sie bitte, daß die interne Auswertung der Funktion f ebenfalls im `rrG`–Format erfolgt, daß aber die berechneten Einschließungen danach maximalgenau ins gröbere `Longreal`–Format gerundet werden, da praktisch alle Anwendungen im `Longreal`- oder `real`–Format gerechnet werden.

wobei für die Operanden auch alle Standardfunktionen aus der Tabelle von Seite 404 gewählt werden können.

Zur Einschließung nur des Funktionswertes $\boldsymbol{f}(a) \in$ `fa` an der festen Stelle $a = (a_1, a_2, \dots, a_n)$ werden vom Hilfsmodul `igrad` die beiden folgenden Prozeduren exportiert:

```
procedure fEvalJ(
  function f(x : GTvector) : GTvector[1..n,0..n];
  a : LIvector;
  var fa : LIvector );                         (igrad)
```

```
procedure fEvalJ_rrG(
  function f(x: GTvector) : GTvector[1..n,0..n];
  a : rrGivector;
  var fa : rrGivector );                       (igrad)
```

Dabei liefert die Prozedur `fEvalJ_rrG` mit `fa` eine garantierte Einschließung des Vektors $\boldsymbol{f}(a)$ im feineren `rrG`–Format, während die Prozedur `fEvalJ` mit `fa` eine i.a. maximalgenaue Einschließung des Vektors $\boldsymbol{f}(a)$ im gröberen `Longreal`–Format berechnet.

Bei der Entwicklung eigener **PASCAL-XSC** Programme ist bezüglich der Indexbereiche der Datentypen `GTvector` und `GradType` folgende Regel zu beachten:

> Ist n die Anzahl der unabhängigen Variablen, so müssen die Indexbereiche der Datentypen wie folgt gewählt werden:
>
> `GTvector [1..n,0..n];`   `GradType [0..n];`

Zur Einschließung des Funktionswertes $\boldsymbol{f}(a) \in$ `fa` und der Jacobimatrix $\boldsymbol{J_f}(a) \in$ `Jac` an der festen Stelle $a = (a_1, a_2, \dots, a_n)$ werden vom Modul `igrad` die beiden folgenden Prozeduren exportiert:

```
procedure fJEvalJ(
  function f(x : GTvector) : GTvector[1..n,0..n];
  a : Livector;
  var fa : Livector;  var Jac : Limatrix ); (igrad)
```

```
procedure fJEvalJ_rrG(
  function f(x : GTvector) : GTvector[1..n,0..n];
  a : rrGivector;
  var fa : rrGivector; var Jac : rrGimatrix ); (igrad)
```

Dabei liefert `fJEvalJ_rrG` mit `fa` und `Jac` garantierte Einschließungen von $f(a)$ und $\boldsymbol{J_f}(a)$ im feineren `rrG`–Format, während die Prozedur `fJEvalJ` mit `fa` und `Jac` i.a. maximalgenaue Einschließungen von $f(a)$ und $\boldsymbol{J_f}(a)$ im gröberen `Longreal`–Format berechnet.

8.2.2.2 Programmbeispiel

$\boldsymbol{f}(x) : I\!R^2 \to I\!R2$ sei definiert durch:

$$\boldsymbol{f}(x) = \begin{array}{ll} f_1(x_1, x_2) & = x_1^2 + \sin(x_1 x_2) \\ f_2(x_1, x_2) & = x_1 \cdot \arcsin(x_2), \qquad x_1 \in I\!R,\ x_2 \in]-1, +1[\end{array}$$

Die Jacobi–Matrix

$$\boldsymbol{J_f}(x) = \begin{pmatrix} 2x_1 + x_2 \cdot \cos(x_1 x_2) & x_1 \cdot \cos(x_1 x_2) \\ \arcsin(x_2) & \dfrac{x_1}{\sqrt{1 - x_2^2}} \end{pmatrix}$$

besitzt an der Stelle $x_0 = (3, 0)$ den Matrix–Wert:

$$\boldsymbol{J_f}(3, 0) = \begin{pmatrix} 6 & 3 \\ 0 & 3 \end{pmatrix}$$

Das folgende Programm `tstigrad4` berechnet für die obige Funktion $\boldsymbol{f}(x)$ Einschließungen des Funktionswertes $\boldsymbol{f}(x_0)$ und der Jacobimatrix $\boldsymbol{J_f}(x_0)$ an der vom Anwender einzugebenden Auswertestelle $x_0 = (a_1, a_2)$:

```
program tstigrad4(input,output);
{ Berechnet fuer die Funktionen:                          }
{ f1(x1,x2) = x1^2 + sin(x1*x2)                           }
{ f2(x1,x2) = x1 * arcsin(x2)                             }
{ f(x) und die Jacobimatrix an der Stelle: a = (a[1],a[2]) }
use igrad,          { exportiert u.a. fJEvalJ(f,a,fa,Ja)   }
    li_ari;         { Longreal-Intervall-Arithmetik        }

const n = 2;        { Anzahl der unabhaengigen Variablen   }

var a  : Livector[1..n];        { Argument-Intervallvektor }
   fa  : Livector[1..n]; { Einschliessung der Funktion  f  }
   Jac : Limatrix[1..n,1..n]; { Jacobimatrix-Einschliessg. }
   i,j : integer;

function f(x : GTvector) : GTvector[1..n,0..n];
begin
   f[1] := sqr(x[1]) + sin(x[1]*x[2]);
   f[2] := x[1] * arcsin(x[2]);
end;

begin
   writeln('   f1(x1,x2) = x1^2 + sin(x1*x2)');
   writeln('   f1(x1,x2) = x1 * arcsin(x2); -1 <= x2 <= +1');
   repeat
      writeln('Eingabe der Auswertestelle:');
      for i := 1 to n do read(a[i]);
      fJEvalJ(f,a,fa,Jac);
      writeln('Einschliessung des Funktionswertes:');
      For i := 1 to n do writeln(fa[i]);     writeln;
      writeln('Einschliessung der Jacobimatrix:');
      for i := 1 to n do
         for j := 1 to n do writeln(Jac[i,j]);
      writeln;
   until false;
end.
```

Nach Eingabe einer Einschließung `(a[1],a[2])` der Auswertestelle

$$x_0 = (3, 0) \in (\mathtt{a}[1], \mathtt{a}[2])$$

liefert das Programm `tstigrad4` die folgende Bildschirmausgabe:

```
   f1(x1,x2) = x1^2 + sin(x1*x2)
   f2(x1,x2) = x1 * arcsin(x2);  -1 < x2 < +1
Eingabe der Auswertestelle:
3 0     <enter>
Einschliessung des Funktionswertes:
[  8.99999999999999999999E+000,  9.00000000000000000001E+000 ]
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]

Einschliessung der Jacobimatrix:
[  5.99999999999999999999E+000,  6.00000000000000000001E+000 ]
[  2.99999999999999999999E+000,  3.00000000000000000001E+000 ]
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
[  2.99999999999999999999E+000,  3.00000000000000000001E+000 ]
```

Hinweise:

- Da die Ableitung der $\arcsin(x)$–Funktion an den Randpunkten ± 1 nicht definiert ist, führen die Auswertestellen $x_0 = (x_1, \pm 1)$ zu einem Laufzeitfehler.

- Wählt man $x_1 = 10^{-20}$, $x_2 = -2 \cdot 10^{-20}$, so erhält man mit dem Programm `tstigrad4` wegen starker Auslöschung für den Wert

$$2x_1 + x_2 \cdot \cos(x_1 x_2) \approx 4 \cdot 10^{-100}$$

  eine so grobe Einschließung, daß noch nicht einmal das richtige Vorzeichen des Jacobi–Elements $2x_1 + x_2 \cdot \cos(x_1 x_2)$ bestimmt wird. Um auch für diese x_1, x_2–Werte z.B. im `Longreal`–Format eine maximalgenaue Einschließung des Jacobi–Elements zu erhalten, müßte man die Jacobi–Matrix in hoher Präzision mit Hilfe einer staggered Arithmetik auswerten.

8.2.3 Berechnung der Hesse–Matrix

Gegeben sei eine zweimal stetig differenzierbare Funktion $f : I\!R^n \to I\!R$ mit n unabhängigen Variablen $x = (x_1, x_2, \ldots, x_n)$. Die Hesse–Matrix $\nabla^2 f(x)$ ist definiert durch:

$$\nabla^2 f(x) := \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_2^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \frac{\partial^2 f}{\partial x_n \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{pmatrix}$$

Im nächsten Abschnitt wird beschrieben, wie man in einem **PASCAL-XSC** Programm die Funktion $f(x)$ darzustellen hat und wie die Hesse–Matrix $\nabla^2 f$ an einer bestimmten Stelle x_0 einzuschließen ist.

8.2.3.1 Routinen zur Hesse–Matrix–Berechnung

Zur Darstellung der Funktion $f(x)$ werden bzgl. der Intervall–Datentypen `rrGinterval` und `Linterval` vom Modul `hess_ari` die folgenden Datentypen exportiert:

```
type HessType = dynamic array [*,*] of rrGinterval;
```

```
type HTvector = dynamic array [*] of HessType;
```

Zur Einschließung von $f(x)$ oder der Hesse–Matrix $\nabla^2 f(x)$ durch Intervalle vom Typ `Linterval` oder `rrGinterval` ist z.B. die Funktion

$$f(x) = f(x_1, x_2) = x_1^2 + \sin(x_1 \cdot x_2)$$

mit zwei Variablen in **PASCAL-XSC** wie folgt zu deklarieren:

```
function f(x : HTvector) : HessType[0..2,0..2];
begin
    f := sqr(x[1]) + sin( x[1]*x[2] );
end;
```

Die Darstellung von $f(x)$ ist also denkbar einfach und entspricht genau der sonst üblichen mathematischen Notation!
Im Ausdruck für die Funktion $f(x)$ sind zulässig die vier Grundoperationen $\{+, -, *, /\}$ mit Operanden vom Typ

```
integer, real, Longreal, rrG, rrGinterval
```

wobei für die Operanden auch alle Standardfunktionen aus der nachfolgenden Tabelle gewählt werden können.

Bei der Entwicklung eigener **PASCAL-XSC** Programme ist bezüglich der Indexbereiche der Datentypen `HTvector` und `HessType` folgende Regel zu beachten:

Ist n die Anzahl der unabhängigen Variablen von $f(x)$, so müssen die Indexbereiche der Datentypen wie folgt gewählt werden:

```
HTvector [1..n,0..n,0..n];  HessType [0..n,0..n];
```

Standardfunktionen vom Typ `HessType` (`hess_ari`) **zur Berechnung von Gradienten und Hesse-Matrizen** `u : HessType; a : Longreal; k : rrGinterval; n : integer;`			
Funktionsterm	**Aufruf**	**Funktionsterm**	**Aufruf**
u^2	`sqr(u)`	$\sin(u)$	`sin(u)`
$\sqrt{u}$	`sqrt(u)`	$\cos(u)$	`cos(u)`
$\sqrt{1+u}-1$	`sqrt1pm1(u)`	$\tan(u)$	`tan(u)`
$\sqrt{1-u^2}$	`sqrt1mx2(u)`	$\cot(u)$	`cot(u)`
$\sqrt{u^2-1}$	`sqrtx2m1(u)`	$\sin(\pi u)$	`sinpi(u)`
$\sqrt{1+u^2}$	`sqrt1px2(u)`	$\cos(\pi u)$	`cospi(u)`
e^u	`exp(u)`	$\tan(\pi u)$	`tanpi(u)`
e^u-1	`expm1(u)`	$\cot(\pi u)$	`cotpi(u)`
e^{-u^2}	`exp_x2(u)`	$\arcsin(u)$	`arcsin(u)`
2^u	`exp2(u)`	$\arccos(u)$	`arccos(u)`
10^u	`exp10(u)`	$\arctan(u)$	`arctan(u)`
u^u	`power(u)`	$\mathrm{arccot}(u)$	`arccot(u)`
$u^n,\ n=0,1,...$	`power(u,n)`	$\sinh(u)$	`sinh(u)`
u^k	`power(u,k)`	$\cosh(u)$	`cosh(u)`
$\ln(u)$	`ln(u)`	$\tanh(u)$	`tanh(u)`
$\ln(1+u)$	`ln1p(u)`	$\coth(u)$	`coth(u)`
$\ln(e\cdot u)$	`ln_ex(u)`	$\mathrm{arsinh}(u)$	`arsinh(u)`
$u\cdot\ln(u)$	`xlnx(u)`	$\mathrm{arcosh}(u)$	`arcosh(u)`
$\log_2(u)$	`log2(u)`	$\mathrm{arcosh}(1+u)$	`arcosh1p(u)`
$\log_{10}(u)$	`log10(u)`	$\mathrm{artanh}(u)$	`artanh(u)`
$\log_{10}(1+u)$	`log1p(u)`	$\mathrm{arcoth}(u)$	`arcoth(u)`
$\log_a(u)$	`loga(a,u)`		

Zur Einschließung des Funktionswertes $f(a) \in$ `fa`, des Gradienten $\nabla f(a) \in$ `ga` und der Hesse–Matrix $\nabla^2 f(a) \in$ `Ha` an der vom Anwender vorgegebenen Stelle $a = (a_1, a_2, \ldots, a_n)$ werden vom Modul `hess_ari` die beiden folgenden Prozeduren exportiert:

```
procedure fghEvalH(
  function f(x : HTvector) : HessType[lb(x,2)..ub(x,2),
                                      lb(x,3)..ub(x,3)];
  a : Livector;
  var fa : Linterval;  var ga : Livector;
  var Ha : Limatrix )                              (hess_ari)
```

```
procedure fghEvalH(
  function f(x : HTvector) : HessType[lb(x,2)..ub(x,2),
                                      lb(x,3)..ub(x,3)];
  a : rrGivector;
  var fa : rrGinterval;  var ga : rrGivector;
  var Ha : rrGimatrix )                            (hess_ari)
```

Dabei liefert die zweite Prozedur `fghEvalH` mit `fa,ga,Ha` garantierte Einschließungen von $f(a), \nabla f(a), \nabla^2 f(a)$ im feineren `rrG`–Format, während die erste Prozedur `fghEvalH` mit `fa,ga,Ha` i.a. maximalgenaue Einschließungen von $f(a), \nabla f(a), \nabla^2 f(a)$ im gröberen `Longreal`–Format berechnet.

Anmerkung:

Die beiden obigen Prozeduren liefern neben der Einschließung der Hessematrix $\nabla^2 f(a) \in$ `Ha` automatisch auch die Einschließung des Gradienten $\nabla f(a) \in$ `ga`. Will man jedoch nur den Funktionswert zusammen mit dem Gradienten berechnen, so ist es laufzeitmäßig günstiger, die beiden folgenden Prozeduren zu benutzen:

```
procedure fghEvalH(
  function f(x : HTvector) : HessType[lb(x,2)..ub(x,2),
                                      lb(x,3)..ub(x,3)];
  a : Livector;
  var fa : Linterval;  var ga : Livector )      (hess_ari)
```

```
procedure fghEvalH(
  function f(x : HTvector) : HessType[lb(x,2)..ub(x,2),
                                      lb(x,3)..ub(x,3)];
  a : rrGivector;
  var fa : rrGinterval;  var ga : rrGivector ) (hess_ari)
```

Zur Einschließung nur der Funktionswerte sind die beiden nachfolgenden Prozeduren am schnellsten, wobei letztere die Einschließung bei gleicher Laufzeit im feineren `rrG`-Format liefert.:

```
procedure fhEvalH(
  function f(x : HTvector) : HessType[lb(x,2)..ub(x,2),
                                      lb(x,3)..ub(x,3)];
  a : Livector;   var fa : Linterval )          (hess_ari)
```

```
procedure fhEvalH(
  function f(x : HTvector) : HessType[lb(x,2)..ub(x,2),
                                      lb(x,3)..ub(x,3)];
  a : rrGivector;   var fa : rrGinterval )     (hess_ari)
```

8.2.3.2 Approximation stationärer Punkte

Gegeben sei eine zweimal stetig differenzierbare Funktion $f : I\!R^n \to I\!R$. Der Definitionsbereich D_f sei ein n-dimensionales Intervall. Ein Innenpunkt x^* von D_f heißt *stationärer* Punkt von f, wenn gilt:

$$\nabla f(x^*) = 0$$

Nur an solchen stationären Punkten kann f bekanntlich lokale Extrema besitzen. Mit einem einfachen Iterationsverfahren kann man ausgehend von einer Näherung $x^{(0)}$ in sehr vielen Fällen eine Einschließung von x^* berechnen. Der benutzte Algorithmus liefert jedoch keine *garantierte* Einschließung eines stationären Punktes, d.h. selbst dann, wenn der Algorithmus mit `Err=0` erfolgreich beendet wird, muß der iterierte Intervallvektor einen stationären Punkt nicht enthalten!

Mit einem Startvektor $x^{(0)} \in D_f$ lautet das bekannt Newton–Verfahren:

$$\begin{array}{rcll} R^{(k)} & := & \{\, \nabla^2 f(x^{(k)})\,\}^{-1}, & \text{(Inversion der Hesse-Matrix)} \\ x^{(k+1)} & := & x^{(k)} - R^{(k)} \cdot \nabla f(x^{(k)}), & k = 0, 1, 2, \ldots \end{array}$$

Zur Anwendung der Intervall–Differentiationsarithmetik schreiben wir das Iterationsverfahren in Intervallform:

$$\begin{array}{rcll} R^{(k)} & := & \{\, m(\nabla^2 f([x]^{(k)}))\,\}^{-1} & \\ [x]^{(k+1)} & := & [x]^{(k)} - R^{(k)} \cdot \nabla f([x]^{(k)}), & k = 0, 1, 2, \ldots \end{array}$$

dabei bedeutet $m(\nabla^2 f([x]^{(k)}))$ die Mittelpunktsmatrix der intervallmäßigen Hesse–Matrix $\nabla^2 f([x]^{(k)})$.

Die Iteration wird erfolgreich abgebrochen, falls die Bedingung

$$0 \in \nabla f([x]^{(k+1)}) \tag{8.25}$$

erfüllt ist. Wegen der unvermeidbaren Intervallüberschätzungen kann jedoch der Fall eintreten, daß (8.25) auf dem Rechner erfüllt wird, ohne daß der Gradient $\nabla f(x)$ über dem Intervallvektor $[x]^{(k+1)}$ tatsächlich verschwindet. Die Iteration wird dann scheinbar erfolgreich abgebrochen, ohne daß $[x]^{(k+1)}$ einen stationären Punkt enthält.

Da die unvermeidbaren Intervallüberschätzungen um so stärker ausfallen, je breiter $[x]^{(k+1)}$ geworden ist[6], läßt sich der obige Algorithmus erheblich

[6] Tatsächlich wachsen die Intervalldurchmesser von $[x]^{(k+1)}$ mit k deutlich an!

verbessern, wenn man nach Abbruch durch (8.25) $[x]_1^{(0)} := \mathrm{mid}([x]^{(k+1)})$ wählt und das obige Iterationsverfahren mit diesem verbesserten Punkt–Intervallvektor neu startet, usw. Das Verfahren wird abgebrochen, wenn nach p Schritten der Intervallvektor $[x]_p^{(k+1)}$ so schmal geworden ist, daß sich die Ober- und Unterschranken aller seiner Intervallkomponenten um höchstens 3 ulp unterscheiden. 1 ulp ist dabei eine Einheit auf der letzten Mantissenstelle.

Der beschriebene Algorithmus wird im folgenden Programm `tsthesse1` in der äußeren `repeat`–Schleife realisiert. Bitte beachten Sie, daß auch dieser stark verbesserte Algorithmus mit $[x]_p^{(k+1)}$ einen stationären Punkt x^* aus den genannten Gründen nicht garantiert einschließen muß!

Als Anwendungsbeispiel betrachten wir eine Verallgemeinerung der zweidimensionalen Rosenbrock–Funktion:

$$f(x) = \sum_{i=2}^{n} 100 \cdot (x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$$

Im nachfolgenden Programm `tsthesse1` wird die Anzahl n der unabhängigen Variablen in der Konstanten `n` festgehalten.

```
program tsthesse1(input,output);
{ Berechnet mit stp i.a. eine Einschliessung eines         }
{ stationaeren Punktes der verallgemeinerten Rosenbrock-   }
{ Funktion f.   stp ist keine garantierte Einschliessung!! }
use hess_ari, { exportiert:    fghEvalH(f, x, fx, gx, Hfx) }
    lmvi_ari, { Matrix/Vektor-Arithmetik im Longrealformat }
    li_ari  , { Intervall-Arithmetik im Longreal-Format    }
    linv    , { exportiert INV(mid(Hfx),InvHfx,factor,Err) }
    mv_util ; { exportiert UlpAcc(x,3) und Operator  in    }
const n = 7;  { Anzahl der unabhaengigen Variablen von  f  }
var fx      : Linterval;
   x,stp,gx : Livector[1..n];
   Err      : integer;

function f(x : HTvector) : HessType[0..n,0..n];
{ Rosenbrocks verallgemeinerte Funktion mit n = 7 Variablen. }
var h : HessType[0..n,0..n];
    i : integer;
begin
   h := 0;
```

```
   for i := 2 to n do
       h := h + 100*sqr(x[i]-sqr(x[i-1])) + sqr(x[i-1]-1);
   f := h;
end;

procedure statpoint(function f(u : HTvector) : HessType;
                    n       : integer;   { Variablen-Anzahl   }
                    x       : Livector;  { Iterationsvektor   }
                    var stp : Livector;  { Stationaerer Punkt }
                    var fx  : Linterval; { Funktionswert      }
                    var gx  : Livector;  { Gradient           }
                    var Err : integer);  { Fehler-Kode        }
const kmax = 100;
      pmax = 20;
var p,k,factor  : integer;
    ok,zeroinGr : boolean;
    Hfx         : Limatrix[1..n,1..n]; { Hesse-Matrix         }
    InvHfx      : Lrmatrix[1..n,1..n]; { Invertierte H-Matrix }
begin
   p := 0;
   repeat
      p := p + 1;   ok := false;
      fghEvalH(f, x, fx, gx, Hfx);
      k := 0;
      repeat
         k := k + 1;   zeroinGr := false;
         INV(mid(Hfx),InvHfx,factor,Err);  { Matrix-Inversion }
         if Err = 0 then
         begin
            x := x - InvHfx * gx;       { Iterationsgleichung }
            fghEvalH(f, x, fx, gx, Hfx);
            zeroinGr := 0 in gx;
         end;
      until zeroinGr or (k >= kmax) or (Err <> 0);
      If zeroinGr and (Err=0) then
      begin
         ok := UlpAcc(x,3);  stp := x;  x := mid(x);
      end  else Err := 3;
   until ok or (p >= pmax) or (Err<>0);
   if not ok then Err := 4;
```

```
end; { statpoint }

begin
   write('Newton''s methode for finding stationary points ');
   writeln('of generalized Rosenbrock''s');
   writeln('function:');
   write('    f(x) = SUM_(i=2)(n): 100*sqr(x[i]-sqr(x[i-1]))');
   writeln(' + sqr(x[i-1]-1)');
   repeat
      writeln;
      writeln('Starting vector x with ',n,' components:');
      read(x);
      statpoint(f,n,x,stp,fx,gx,Err);
      if Err = 0 then
      begin
      writeln('Stationary point: ',stp,'Gradient value: ',gx);
      writeln('Function value:');  writeln(fx);
      end        else writeln('Err = ',Err);
   until false;
end.
```

Der Start des obigen Programms `tsthesse1` liefert mit den entsprechenden Eingaben für den Startvektor `x` die folgende Bildschirmausgabe:

```
Newton's methode for finding stationary points of generalized
Rosenbrock's function:
 f(x) = SUM_(i=2)(n): 100*sqr(x[i]-sqr(x[i-1])) + sqr(x[i-1]-1)

Starting vector x with 7 components:
3 3 3 3 3 3 3   <enter>
Stationary point:
[  9.9999999999999999999E-001,  1.0000000000000000001E+000 ]
[  9.9999999999999999999E-001,  1.0000000000000000001E+000 ]
[  9.9999999999999999999E-001,  1.0000000000000000001E+000 ]
[  9.9999999999999999999E-001,  1.0000000000000000001E+000 ]
[  9.9999999999999999999E-001,  1.0000000000000000001E+000 ]
[  9.9999999999999999999E-001,  1.0000000000000000001E+000 ]
[  9.9999999999999999999E-001,  1.0000000000000000001E+000 ]
Gradient value:
[ -4.8020200004004000005E-018,  8.4200200004040000009E-018 ]
```

```
[ -9.00203000060240000005E-018,  1.08200300006042000001E-017 ]
[ -9.00203000060240000005E-018,  1.08200300006042000001E-017 ]
[ -9.00203000060240000005E-018,  1.08200300006042000001E-017 ]
[ -9.00203000060240000005E-018,  1.08200300006042000001E-017 ]
[ -9.00203000060240000005E-018,  1.08200300006042000001E-017 ]
[ -4.20001000020200000001E-018,  2.40001000020020000001E-018 ]

Function value:
[  0.00000000000000000000E+000,  2.65201260026952060601E-037 ]

Starting vector x with 7 components:
2 2 2 2 2 2 2   <enter>
Stationary point:
[  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
Gradient value:
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]

Function value:
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]

Starting vector x with 7 components:
-2 2 2 2 2 2 2  <enter>
Stationary point:
[ -9.91722572561405540275E-001, -9.91722572561405540274E-001 ]
[  9.93555393503371264466E-001,  9.93555393503371264467E-001 ]
[  9.92173321594692115637E-001,  9.92173321594692115638E-001 ]
[  9.86898762690313916967E-001,  9.86898762690313916968E-001 ]
[  9.75164756608873041320E-001,  9.75164756608873041321E-001 ]
```

```
[  9.51431982704992671207E-001,  9.51431982704992671209E-001 ]
[  9.05222817713953473422E-001,  9.05222817713953473423E-001 ]
Gradient value:
[ -1.70833500000000000001E-019,  1.01872990000000000001E-018 ]
[ -4.72308904000000000001E-019,  1.31559730400000000001E-018 ]
[ -1.04596835400000000001E-018,  7.38908844000000000001E-019 ]
[ -8.47258038000000000001E-019,  9.26086308000000000001E-019 ]
[ -1.38413789500000000001E-018,  7.53764775000000000001E-019 ]
[ -1.33020159000000000001E-018,  1.59003410100000000001E-018 ]
[ -5.44226000000000000001E-019,  4.16940000000000000001E-019 ]

Function value:
[  3.98360053642485395624E+000,  3.98360053642485395626E+000 ]

Starting vector x with 7 components:
3 3 3 3 4 5 6   <enter>
Stationary point:
[  1.00000000000000000000E+000,  1.00000000000000000001E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000001E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000001E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000001E+000 ]
[  1.00000000000000000000E+000,  1.00000000000000000001E+000 ]
[  1.00000000000000000001E+000,  1.00000000000000000002E+000 ]
[  1.00000000000000000003E+000,  1.00000000000000000004E+000 ]
Gradient value:
[ -4.00000000000000000005E-018,  8.02002000040400000009E-018 ]
[ -8.00001000020200000005E-018,  1.00200200004040000001E-017 ]
[ -8.00001000020200000005E-018,  1.00200200004040000001E-017 ]
[ -8.00001000020200000005E-018,  1.00200200004040000001E-017 ]
[ -1.20000100002020000001E-017,  6.02002000040400000005E-018 ]
[ -9.98003000060600000017E-018,  8.04002000040400000009E-018 ]
[ -2.00001000020200000001E-018,  4.00001000020200000001E-018 ]

Function value:
[  9.99999999999999999999E-041,  2.40901000021450050501E-037 ]
```

Anmerkungen:

1. Beachten Sie, daß die Prozedur `statpoint(f,n,x,stp,fx,gx,Err)` mit dem Intervallvektor `stp` i.a. keine garantierte Einschließung eines stationären Punktes x^* liefert. Vergleichen Sie dazu die Anmerkungen von Seite 424.

2. Nur wenn der *punktförmige* Intervallvektor `gx` als Gradient der Nullvektor ist, liefert `stp` tatsächlich eine garantierte Einschließung eines stationären Punktes x^* von f. Vergleichen Sie dazu die Ergebnisse bzgl. des Startvektors `x = (2,2,2,2,2,2,2)`. Der punktförmige Intervall–Vektor `stp = (1,1,1,1,1,1,1)` ist damit tatsächlich die garantierte Einschließung des stationären Punktes $x_1^* = (1, 1, 1, 1, 1, 1, 1)$.

3. Nach Punkt 2 liefert also der Startvektor `x = (3,3,3,3,3,3,3)` mit `stp` eine garantierte Einschließung des stationären Punktes $x_1^* = (1, 1, 1, 1, 1, 1, 1)$. Es ist damit jedoch keinesfalls sicher, daß x_1^* in `stp` der einzige stationäre Punkt ist!

4. Mit dem Startvektor `x = (-2,2,2,2,2,2,2)` erhält man vermutlich die Einschließung eines anderen stationären Punktes.

5. Mit dem neuen Startvektor `x = (3,3,3,3,4,5,6)` wird das Iterationsverfahren mit `Err=0` zwar erfolgreich abgeschlossen, aber `stp` liefert vermutlich keine garantierte Einschließung eines stationären Punktes.

Das Beispiel zeigt sehr eindrucksvoll, wie viele Fragen offen bleiben, wenn der benutzte Algorithmus nicht die Existenz und Eindeutigkeit der berechneten Ergebnisse garantieren kann.

Im Abschnitt ab Seite 486 werden die Einschließungen der gesuchten stationären Punkte berechnet durch garantierte Einschließungen der Lösungen eines entsprechenden nichtlinearen Gleichungssystems. Vergleicht man die so gewonnenen Ergebnisse mit den entsprechenden Einschließungen dieses Abschnitts, so zeigt sich, daß nur die Approximation der stationären Punkte ohne einen nachfolgenden Eindeutigkeitsnachweis zu Einschließungen führen kann, die in Wirklichkeit keine stationären Punkte enthalten.

8.3 Komplexe Funktionen mit einer Variablen

8.3.1 Theoretische Grundlagen

Zur Berechnung der Taylorkoeffizienten einer vorgegebenen, analytischen Funktion $f(z), z \in \mathbb{C}$ gelten die gleichen Überlegungen wie bei reellen Funktionen, die durch ihre Taylorreihe darstellbar sind. Da insbesondere die Rekursionsformeln der Taylorkoeffizienten aus dem Reellen direkt übernommen werden können, lassen sich alle vorhandenen Routinen des Moduls `itaylor` praktisch direkt ins Komplexe übertragen, wenn man den Datentyp `itaylor` in `citaylor` umbenennt und die Grundoperatoren auch für die benötigten komplexe Operanden definiert. Der Leser findet ausführliche Informationen zur Taylorarithmetik ab Seite 365.

8.3.2 Routinen zur Berechnung der Taylorkoeffizienten

Die Einschließung der Taylorkoeffizienten einer gegebenen Funktion $f(u(z))$ erfolgt mit Hilfe des vom Modul `citaylor` exportierten Datentyps

```
type citaylor = dynamic array [*] of rrGcinterval;
```

und einer Funktion `f`, die wie folgt zu deklarieren ist:

```
function f(u : citaylor) : citaylor;
begin
    f := citaylor-Ausdruck;
end;
```

Ist `z0` der vorgegebene Entwicklungspunkt oder das vorgegebene Intervall der Entwicklungspunkte und ist `p` die gewünschte Ordnung, bis zu der die Taylorkoeffizienten einzuschließen sind, so liefert die Prozedur

```
procedure expand ( var u : citaylor; z0 : ■ );   (citaylor)
■ = (L)complex, rrGcomplex, (L)cinterval, rrGcinterval
```

mit der Ausgangsgröße `u : citaylor[0..p]` einen Vektor mit den $(p+1)$ Intervall-Komponenten:

$$\begin{aligned} \mathtt{u}[0] &= \mathtt{z0} \\ \mathtt{u}[1] &= 1 \\ \mathtt{u}[\mathtt{j}] &= 0 \quad \text{für } 2 \le j \le \mathtt{ub(u)} = p\,, \end{aligned}$$

d.h. `u : citaylor[0..p]` liefert die Taylorkoeffizienten der Ordnung 0 bis zur Ordnung `p = ub(u)` für die Funktion $u(z) \equiv z$.

Der `citaylor-Ausdruck` im Anweisungsteil der obigen Funktion `f` läßt sich realisieren mit Hilfe der vom Modul `citaylor` exportierten Operatoren $\boxed{+,-,*,/}$ mit dem Ergebnistyp `citaylor`, wobei zunächst beide Operanden vom Typ `citaylor` sein dürfen. Ist jedoch nur ein Operand vom Typ `citaylor`, so darf der andere Operand die folgenden Datentypen annehmen:

```
integer, real, Longreal, rrG, rrGinterval,
complex, Lcomplex, rrGcomplex, rrGcinterval
```

Die Anweisungen

```
expand(u,z0);
y := f(u);
```

liefern dann mit `y : citaylor[0..p]` Einschließungen der Taylorkoeffizienten bis zur Ordnung `p` der Funktion `f` für alle Entwicklungspunkte aus dem Intervall `z0`.

Die Einschließung der Taylorkoeffizienten der Funktion $f(z) = z/(1+z^2)$ an der Stelle $z_0 = \pi + i \cdot \sqrt{2}$ liefert das nachfolgende Programm `tstcitayl1`

```
program tstcitayl1(input,output);
{ Einschliessung des Taylorkoeffizienten der Ordnung p }
{ der komplexen Funktion:         f=z/(1+z*z)          }
{ an der Stelle        z0 = Pi + i*sqrt(2);            }
use citaylor, rrgi_util, lci_ari;
var p,k : integer;
    z0  : rrGcinterval; { Entwicklungspunkte-Intervall    }
    h   : Lcinterval;   { Ausgabe der Tayl.-Koeffizienten }
    y,u : citaylor;
function f(u : citaylor): citaylor[0..ub(u)];
```

```
begin
   f := u / (1 + u*u);
end; { f }
begin
   writeln('    Komplexe Funktion:  f(u) = u / (1 + u*u)');
   repeat
      writeln('Ordnung des Taylor-Koeffizienten = ?');
      read(p);
      allocate(u,0..p);  { Speicherplatz der dynamischen  }
      allocate(y,0..p);  { Variablen   u,y   anlegen.     }
      z0.re := Pi_rrGi;  { Realteil:  Einschlssg. von Pi  }
      z0.im := SQRT2_rrGi; { Img.-T:  Einschl. v. sqrt(2) }
      expand(u,z0);      { Eingangsvariable u realisieren }
      y := f(u); { Einschliessung d. Taylor-Koeffizienten }
      h := y[p];         { Ausgabe der Einschliessung des }
      write('Einschliessung des Taylor-Koeffizienten ');
      writeln('der Ordnung ',p,':');  writeln(h);  writeln;
                         { Taylorkoeffizienten d. Ordg. p }
      free(u); free(y);  { Speicher fuer u,y freigeben.   }
   until p < 0;
end.
```

Das obige Programm `tstcitayl1` liefert zu den vorgegebenen Ordnungen die folgenden Einschließungen der Taylorkoeffizienten:

```
    Komplexe Funktion:  f(u) = u / (1 + u*u)
Ordnung des Taylorkoeffizienten = ?
0       <enter>
Einschliessung des Taylor-Koeffizienten der Ordnung 0:
( [ 2.564987414416022270 44E-001, 2.5649874144160222 7045E-001 ],
  [-9.752117033998123137 24E-002,-9.7521170339981231 3723E-002 ])
Ordnung des Taylorkoeffizienten = ?
5       <enter>
Einschliessung des Taylor-Koeffizienten der Ordnung 5:
( [-2.578182559230590579 36E-004,-2.5781825592305905 7932E-004 ],
  [ 2.578374534169657869 34E-004, 2.5783745341696578 6938E-004 ])
Ordnung des Taylorkoeffizienten = ?
20      <enter>
Einschliessung des Taylor-Koeffizienten der Ordnung 20:
( [-1.396303642349399059 72E-011,-1.3963036423477224 3811E-011 ],
  [-5.867729958996817962 37E-012,-5.8677299589800517 4637E-012 ])
```

Hinweise:

- Der nicht darstellbare Entwicklungspunkt $z_0 = \pi + i \cdot \sqrt{2}$ muß durch das komplexe Intervall `z0` eingeschlossen werden, um die Taylorkoeffizienten korrekt einschließen zu können.
- Da der Durchmesser von `z0` von Null verschieden ist, erhält man mit der Ordnung $p = 20$ schon merkbare Intervallüberschätzungen, die bei der Einschließung des Taylorkoeffizienten nur noch 11 verläßliche Stellen belassen.
- Im Anweisungsteil einer Funktion

  ```
  function f(u : citaylor) : citaylor[0..ub(u)]
  ```

  kann der Ausdruck für `f` auch Standardfunktionen vom Typ `citaylor` enthalten, die in folgender Tabelle zusammengestellt sind und vom Modul `citaylor` exportiert werden. Dadurch lassen sich Einschließungen der Taylorkoeffizienten auch für praktisch beliebig verschachtelte Funktionen berechnen.

Standardfunktionen vom Typ `citaylor` (citaylor) `u : citaylor;` `a : rrGcinterval;`			
Funktion	**Aufruf**	**Funktion**	**Aufruf**
u^2	`sqr(u)`	$\ln(1+u)$	`ln1p(u)`
$\sqrt{u}$	`sqrt(u)`	$\ln(1+e^u)$	`lnexp1(u)`
u^a	`power(u,a)`	$\sin(u)$	`sin(u)`
u^u	`power(u)`	$\cos(u)$	`cos(u)`
e^u	`exp(u)`	$\sinh(u)$	`sinh(u)`
$\ln(u)$	`ln(u)`	$\cosh(u)$	`cosh(u)`

Mit Hilfe der nachfolgenden Prozedur berechnet man zu gegebener Funktion `f` vom Ergebnistyp `citaylor` garantierte Einschließungen seiner Taylorkoeffizienten zu Entwicklungspunkten aus dem Intervall `z0`.

procedure `taylor_K` (`function f(t : citaylor): citaylor;`
(citaylor) `z0 : ■`; **var** `taylor : citaylor`);
■ = `rrGcinterval, Lcinterval, cinterval;`

Die Ausgangsgröße `taylor : citaylor[0..p]` liefert die Einschließungen der Taylorkoeffizienten von `f` bis zur Ordnung `p = ub(taylor)`.

Mit Hilfe der folgenden Prozedur berechnet man zu gegebener Funktion `f` vom Ergebnistyp `citaylor` garantierte Einschließungen seiner Ableitungen an allen Punkten des Intervalls `z0`.

procedure `derivative` (`function f(t: citaylor): citaylor;`
(citaylor) `z0 : ■`; **var** `deriv : citaylor`);
■ = `rrGcinterval, Lcinterval, cinterval;`

Die Ausgangsgröße `deriv : citaylor[0..p]` liefert die Einschließungen der Ableitungen von `f` bis zur Ordnung `p = ub(deriv)`.

8.3.3 Programmbeispiele

8.3.3.1 Ableitungen von $f(z) = \ln(1 + e^z)$

Das nachfolgende Programm `tstcitayl2` berechnet für $f(z) = \ln(1 + e^z)$ Einschließungen der Ableitung p-ter Ordnung an einer vorgegebenen Stelle z_0, die im Programm durch das komplexe Intervall `z0` eingeschlossen wird.

```
program tstcitayl2(input,output);
{ Einschliessungen der Ableitungen der Ordnung p fuer:      }
{                  f(u) = ln( 1+exp(u) )                    }
use citaylor, lci_ari;
var p    : integer;
   z0    : rrGcinterval;               { Argument-Intervall }
   y     : Lcinterval; { Zum Einschliessen der Ableitungen   }
```

```
   deriv : citaylor;   { Liefert Einschlg. aller Ableitungen }

function f (u : citaylor) : citaylor[0..ub(u)];
{ f(u) = ln( 1+exp(u) )}
begin
   f := lnexp1(u);
end; { f }

begin
   writeln('     Function f(u) = ln( 1+exp(u) );');
   repeat
      writeln('Order of derivative = ?');  read(p);
      allocate(deriv,0..p);    { Speicherplatz bereitstellen }
      writeln('point of evaluation = ?');  read(z0);

      derivative(f,z0,deriv);  { deriv: Einschliessungen der }
      y := deriv[p];           { ersten p Ableitungen.      }
      writeln('Inclusion of the ',p,'. derivative:');
      writeln(y);  writeln;
      free(deriv);          { Speicherplatz wieder freigeben }
   until p<0;
end.
```

Nach Eingabe der Ordnung `p` und des Intervalls `z0`, das den Entwicklungspunkt $z_0 \in$ `z0` enthalten muß, liefert obiges Programm die Ergebnisse:

```
     Function f(u) = ln( 1+exp(u) )
Order of derivative = ?
10          <enter>
point of evaluation = ?
(1,0)       <enter>
Inclusion of the 10. derivative:
( [-2.261602898428369477783E+001,-2.261602898428369477766E+001],
  [ 6.291944864710165761755E+001, 6.291944864710165761933E+001] )
Order of derivative = ?
20          <enter>
point of evaluation = ?
(1e3,1e3)   <enter>
Inclusion of the 20. derivative:
( [ 2.854613076093438594077E-435, 2.854613076093438594088E-435],
  [-4.197206560965025066598E-435,-4.197206560965025066588E-435] )
```

Kapitel 9

Nichtlineare Gleichungen

9.1 Problemstellung

Gegeben sei eine stetig differenzierbare Funktion $f : \mathbb{R} \to \mathbb{R}$ und ein Intervall $[a, b]$ aus dem Definitionsbereich $D_f \subseteq \mathbb{R}$, wobei zusätzlich vorausgesetzt werden soll, daß f in $[a, b]$ nur endlich viele Nullstellen besitzt. Gesucht ist die Menge aller dieser Nullstellen $x_i \in [a, b]$, mit $f(x_i) = 0,\ i = 1, 2, \ldots n,\ a < b$.

Die Berechnung aller $x_i \in [a, b]$ ist eine der wichtigsten Grundaufgaben der Numerik und gilt als prinzipiell nicht lösbar, wenn die x_i mit Hilfe eines Computerprogramms zu bestimmen sind. Es existieren u.a. die folgenden, i.a. nicht gleichzeitig lösbaren Teilprobleme:

1. Zwischen zwei benachbarten Rasterzahlen liegen mehrere Nullstellen, die daher nicht mehr getrennt darstellbar sind.

2. Eine existierende, mehrfache Nullstelle gerader Ordnung wird wegen der unvermeidbaren Rundungsfehler vom Algorithmus nicht mehr mit Sicherheit erkannt.

3. Wegen der gleichen Rundungsfehler wird eine Nullstelle vorgetäuscht, die in Wirklichkeit jedoch nicht existiert.

Nach Ablauf eines Nullstellenprogramms gibt es damit i.a. keine Sicherheit, daß genau die in $[a, b]$ existierenden Nullstellen x_i mit den berechneten Nullstellen übereinstimmen. Außerdem erhält man i.a. keine gesicherten Informationen, mit welchen Fehlern die berechneten Nullstellen–Approximationen behaftet sind.

9.2 Nullstelleneinschließung in PASCAL-XSC

Eine wirklich grundlegende Verbesserung der Nullstellenberechnung erreicht man in der **PASCAL-XSC** Umgebung durch den Einsatz der Routine `AllZeros`, mit deren Hilfe **garantierte** Existenz- und Eindeutigkeitsaussagen für die Nullstellen–Einschließungen geliefert werden.

Für ein vorgegebenes Intervall $[a, b]$ liefert `AllZeros` Teilintervalle $y_i, [x_i]$ mit folgenden Informationen:

1. $[x_i] \subset [a, b]$ enthält genau eine Nullstelle x_i der Funktion $f(x)$

2. $y_i \subset [a, b]$ kann eine Nullstelle von $f(x)$ enthalten.

Bildet man dann die Differenzmenge $[a, b] \backslash \{[x_i], y_i\}$ aus $[a, b]$ und allen Teilintervallen $[x_i], y_i$, so erhält man Teilintervalle, die bewiesenermaßen keine Nullstellen enthalten. Damit bleiben vom ganzen Intervall $[a, b]$ lediglich die Teilintervalle y_i übrig, für die eine Aussage über die Existenz von Nullstellen nicht möglich ist. Es bleibt dann die Aufgabe, durch geeignete Umformungen der Funktion f Existenz- und Eindeutigkeitsaussagen auch für diese Teilintervalle y_i zu gewinnen. Ab Seite 441 wird anhand vieler Beispiele gezeigt, wie man in speziellen Fällen vorzugehen hat, um auch für die restlichen y_i die gewünschten Existenz- und Eindeutigkeitsaussagen zu gewinnen.

Ein ganz wesentlicher Vorteil der Prozedur `AllZeros` besteht außerdem darin, daß mit Hilfe der berechneten Einschließungen $[x_i] \ni x_i$ der relative oder absolute Fehler der jeweiligen Nullstellenapproximation direkt angegeben werden kann.

Durch Anwendung der Prozedur `AllZeros` ist die stetig differenzierbare Funktion $f : I\!R \to I\!R$ wie folgt einzuschränken:

> Im Ausdruck für die Funktion f sind die vier Grundoperationen $+, -, *, /$ zulässig, wobei die Operanden entweder vom Typ `integer,real,Longreal,rrG,rrGinterval` sein können oder als Standardfunktionen aus der Tabelle von Seite 351 zu wählen sind.

In [5, Seite 87,..] findet der interessierte Leser nähere Informationen zum benutzten Intervall–Newtonverfahren und zur extended Intervallarithmetik, mit deren Hilfe die Nullstellen x_i eingeschlossen werden.

9.2.1 Routinen zur Nullstelleneinschließung

Zur Deklaration einer stetig differenzierbaren Funktion $f : I\!R \to I\!R$, deren Nullstellen $x_i \in [a, b]$ einzuschließen sind, benötigt man in **PASCAL-XSC** den Typ `DerivType`, der auf Seite 350 definiert ist und vom Hilfsmodul `ddf_rrg` exportiert wird. Die Funktion

$$f(x) = e^{-3x} - \sin^3(x)$$

wird beispielsweise wie folgt deklariert:

```
function f(x : DerivType) : DerivType;
begin
    f := exp(-3*x) - power(sin(x),3);
end;
```

Der `DerivType`–Ausdruck für f entspricht also genau der sonst üblichen mathematischen Notation und ist damit sehr gut lesbar.

Die Einschließungen der gesuchten Nullstellen $x_i \in [a, b]$ der Funktion f liefert die nachfolgende Prozedur `AllZeros`, die vom Hilfsmodul `nlfzero` exportiert wird.

```
procedure AllZeros( function f(x : DerivType) : DerivType;
  ab  : Linterval;                         { Startintervall }
  eps : Longreal;                   { Genauigkeitsparameter }
  var Zerovector   : Livector;       { liefert x_i und y_i }
  var Infovector   : bvector;    { Existenz/Eindeutigkeit }
  var Numberofzeros : integer;  { Zahl der Einschliessgn. }
  var Err          : integer; )      { Err=0: Kein Fehler }
```

Eingangsgrößen:

- `f` : Funktion, deren Nullstellen einzuschließen sind.
- `ab` : Alle Nullstellen aus diesem Startintervall sind einzuschließen.
- `eps` : Relative Fehlerschranke der Nullstelleneinschließungen.

Ausgangsgrößen:

- `Zerovector` : Speichert alle gefundenen Einschliessungen.
- `Infovector` : Existenz- u. Eindeutigkeitsaussagen der x_i.
- `Numberofzeros` : Zahl der gefundenen Einschließungen.
- `Err` : Fehler-Kode; `Err=0:` Keine Fehler.

Anmerkungen:

1. Im Intervall `ab = [a,b]` sind alle Nullstellen der stetig differenzierbaren Funktion $f : I\!R \to I\!R$ einzuschließen.

2. Die Einschließungen $[x_i], y_i$ werden solange verbessert, bis die relativen Durchmesser der Intervalle kleiner `eps` geworden sind. Ein zu klein gewählter `eps`–Wert wird intern auf $5 \cdot 10^{-20}$ gesetzt.

3. Im Intervallvektor `Zerovector` werden alle gefundenen Einschließungen $[x_i], y_i$ gespeichert. Ist die Anzahl der gefundenen Einschließungen $[x_i], y_i$ größer als die Dimension von `Zerovector`, so erfolgt eine Fehlermeldung und `ub(Zerovector)` ist im aufrufenden Programm entsprechend zu vergrößern.

4. `Infovector[i]` liefert zur gefundenen Einschließung `Zerovector[i]` folgende Informationen:

   - `Infovector[i] = true` : $[x_i]$ encloses a locally unique zero!
   - `Infovector[i] = false` : y_i may contain a zero

   Im ersten Fall enthält die Intervall–Komponente `Zerovector`$[\mathtt{i}] = [x_i]$ genau eine einfache Nullstelle x_i von $f(x)$. Im zweiten Fall `(false)` kann `Zerovector`$[\mathtt{i}] = y_i$ mehrere einfache oder auch mehrfache Nullstellen enthalten. Es kann aber auch sein, daß `Zerovector`$[\mathtt{i}] = y_i$ keine Nullstelle enthält. Die Vektoren `Zerovector` und `Infovector` müssen den gleichen Indexbereich besitzen.

5. `Numberofzeros` liefert die Anzahl aller berechneten Einschließungen $[x_i], y_i$. Nur wenn `Infovector[i] = true` für alle i erfüllt ist, liefert `Numberofzeros` die Anzahl der Nullstellen von $f(x)$ in `ab = [a,b]`.

9.2.2 Programmbeispiele

9.2.2.1 Mehrfache Nullstellen

Eine Funktion $f : I\!R \to I\!R$ besitzt in x_0 eine n-fache Nullstelle, wenn gilt:

$$f(x) = (x - x_0)^n \cdot g(x), \qquad n = 1, 2, \dots, \quad g(x_0) \neq 0$$

Für $n \geq 2$ liefert die Prozedur `AllZeros` zwar Einschließungen y_i dieser mehrfachen Nullstellen, aber Existenz- und Eindeutigkeitsaussagen werden mit dem Vektor `Infovector` nicht geliefert, wie folgendes Programm mit der Funktion $f(x) = (x - 1)^3$ zeigt:

```
program tstnlfz1;
use
  type_def,   { Type Definitionen          }
  li_ari,     { Interval arithmetic        }
  ddf_rrg,    { Differentiation arithmetic }
  nlfzero;    { Nonlinear function zeros   }
const n = 20; { Maximum number of zeros to be computed }

function f (x: DerivType) : DerivType;
begin
   f := power(x-1,3);
end;

{-------------------------------------------------------------}
{ Procedure for prompting and reading information to call the }
{ procedure  'AllZeros'.  This procedure must be called with  }
{ the function 'f' and a string 'Name' containing a textual   }
{ description of that function.                               }
{-------------------------------------------------------------}
procedure compute (function f(x:DerivType): DerivType;
                                       Name : string);
var
  SearchInterval            : Linterval;
  Tolerance                 : Longreal;
  Zero                      : Livector[1..n];
  Unique                    : bvector[1..n];
  NumberOfZeros, i, ErrCode : integer;
begin
```

```
   writeln('Computing all zeros of the function  ', Name);
   write('Search interval      : ');  read(SearchInterval);
   write('Tolerance (relative) : ');  read(Tolerance);
   writeln;
   AllZeros(f,SearchInterval,Tolerance,Zero,Unique,
                                          NumberOfZeros,ErrCode);
   for i:=1 to NumberOfZeros do
   begin
      writeln(Zero[i]);
      if unique[i] then
         writeln(' encloses a locally unique zero!')
                  else
         writeln(' may contain a zero (not verified unique)!');
   end;
   if ErrCode <> 0 then writeln(AllZerosErrMsg(ErrCode));
   writeln;
   writeln(NumberOfZeros:1, ' interval enclosure(s)'); writeln;
end;

begin
  compute(f, 'f(x) = (x-1)^3');
end.
```

Der Start des Programms `tstnlfz1` liefert mit den entsprechenden Startintervallen und Toleranzwerten die folgende Bildschirmausgabe:

```
Computing all zeros of the function  f(x) = (x-1)^3
search interval      : [0,5]  <enter>
Tolerance (relative) : 5e-20  <enter>

[  9.99999999999999999976E-001, 1.00000000000000000001E+000 ]
may contain a zero (not verified unique)!

search interval      : [0.5,2]  <enter>
Tolerance (relative) : 5e-20    <enter>

[  9.99999999999999999961E-001, 1.00000000000000000000E+000 ]
may contain a zero (not verified unique)!
[  1.00000000000000000000E-001, 1.00000000000000000002E+000 ]
may contain a zero (not verified unique)!
```

Anmerkungen:

- Das Programm zeigt, daß die Prozedur `AllZeros` eine mehrfache Nullstelle zwar stets einschließt[1], daß aber eine Existenz- und Eindeutigkeitsaussage nicht geliefert wird. Das Programm auf Seite 445 gibt ein Beispiel dafür, daß ein Intervall angegeben wird mit der Information *'may contain a zero (not verified unique)!'*, das eine Nullstelle jedoch nicht wirklich einschließt.

- Mit dem Startintervall $[0.5, 2]$ erhält man zwei Intervalle mit gemeinsamer Ober- und Untergrenze jeweils mit der gleichen Information *'may contain a zero (not verified unique)!'*, so daß man beide Intervalle zu einem einzigen Intervall vereinigen kann.

Da die Prozedur `AllZeros` bei einer mehrfachen Nullstelle x_0 keine Existenz- und Eindeutigkeitsaussagen liefert, wird man bemüht sein, die gegebene Funktion $f(x)$ in eine andere Funktion $q(x)$ so umzuformen, daß $q(x)$ in x_0 eine einfache Nullstelle besitzt, die dann durch `AllZeros` als solche eindeutig erkannt wird. Eine solche Umformung ist prinzipiell nach folgendem Satz immer möglich:

Gegeben sei $f(x) = (x - x_0)^n \cdot g(x)$ mit stetig differenzierbarer Funktion $g(x)$. Unter der Voraussetzung $g(x_0) \neq 0$ besitzt dann der Quotient

$$\frac{f(x)}{f'(x)} = q(x) := \frac{(x - x_0)^1 \cdot g(x)}{n \cdot g(x) + (x - x_0) \cdot g'(x)}, \qquad n \geq 1 \tag{9.1}$$

in x_0 eine **einfache** Nullstelle, und es gilt:

$$\lim_{x \to x_0} \frac{f(x)}{f'(x)} = q(x_0) = 0$$

In einer Umgebung von x_0, in der der Nenner von $q(x)$ nicht verschwindet, haben also $f(x)$ und $q(x)$ identische Nullstellen, wobei jedoch die Nullstelle x_0 die gesuchte einfache Nullstelle von $q(x)$ ist.

Bei der praktischen Anwendung besteht die Schwierigkeit jedoch darin, daß der Ausdruck $f(x)/f'(x)$ für $x \to x_0$ zunächst die Form $0/0$ besitzt und deshalb bei der Intervallauswertung auf dem Rechner eine Fehlermeldung *'Division durch Null'* erzeugt. $f(x)/f'(x)$ muß daher so umgeformt werden, daß der unbestimmte Ausdruck $0/0$ verschwindet und die Nullstellen dabei unverändert bleiben.

[1] Eine mehrfache Nullstelle wird also von der Prozedur `AllZeros` niemals übersehen!

Als Beispiel betrachten wir die Funktion $f(x) = 1 + \sin(x-2)$, die im Intervall $[6,7]$ die doppelte Nullstelle $x_0 = 2 + 1.5 \cdot \pi$ besitzt. Der Quotient $f/f' = (1+\sin(x-2))/\cos(x-2)$ ist für x_0 nicht definiert und würde auf dem Rechner einen Programmabbruch verursachen. Im Intervall $[6,7]$ gilt jedoch die identische Umformung

$$\frac{f(x)}{f'(x)} = \frac{1+\sin(x-2)}{\cos(x-2)} \equiv \frac{\cos(x-2)}{1-\sin(x-2)} =: q(x),$$

wobei $q(x)$ jetzt auch an der Stelle $x_0 = 2+1.5\cdot\pi$ definiert ist und $f(x)/f'(x)$ stetig ergänzt. Definiert man daher im Programm `tstnlfz1` die Funktion $q(x)$ als:

```
function q (x: DerivType) : DerivType;
begin
   q := cos(x-2) / (1-sin(x-2));
end;
```

und ändert den Funktionsnamen von `f` nach `q`, so erhält man mit dem Startintervall `[6,7]` und dem Toleranzwert $5 \cdot 10^{-20}$ die folgende maximalgenaue Einschließung der doppelten Nullstelle $x_0 = 2 + 1.5 \cdot \pi$ von $f(x)$:

```
[  6.71238898038468985769E+000, 6.71238898038468985770E+000 ]
encloses a locally unique zero!
```

Analog zum obigen Beispiel behandelt man z.B. auch

$$f(x) = \cosh(x) - 1 \quad \text{mit} \quad f/f' = 2\sinh(x)/\cosh^2(x/2)$$

an der Stelle $x_0 = 0$, an der f/f' eine einfache Nullstelle besitzt, die deshalb von der Prozedur `AllZeros` erfolgreich eingeschlossen werden kann.

Ist f jedoch durch einen komplizierteren Funktionsterm definiert und besitzt f an einer unbekannten Stelle x_0 eine mehrfache Nullstelle, so ist deren Existenz und Eindeutigkeit mit Hilfe der Funktion f/f' i.a. nicht nachweisbar, da f/f' an der einfachen Nullstelle x_0 zunächst die Form $0/0$ besitzt. Diese für die Intervallauswertung ungeeignete Form läßt sich jedoch selbst mit Algebrapaketen nur in Ausnahmefällen so umformen, daß der Nenner des Terms für f/f' nicht mehr verschwindet.

Mehrfache Nullstellen von f werden von der Prozedur `AllZeros` jedoch nicht übersehen, sondern stets eingeschlossen mit dem Kommentar

```
may contain a zero (not verified unique)!
```

Allerdings darf man aus dieser Meldung nicht rückwärts auf die Existenz einer mehrfachen Nullstelle schließen!

9.2.2.2 Wahl des geeigneten Funktionsterms

9.2.2.2.1 $f(x) = \cosh(x) - \sinh(x)$

Im ersten Beispiel betrachten wir die Funktion

$$f(x) = \cosh(x) - \sinh(x) \equiv e^{-x} > 0,$$

Das Programm `tstnlfz1` von Seite 441 liefert mit der Deklaration

```
function f (x: DerivType) : DerivType;
begin
   f := cosh(x) - sinh(x);
end;
```

und dem Startintervall $[9, 10]$ zusammen mit der Toleranz 10^{-6} mehr als 20 Einschließungen, alle mit der Information:

```
may contain a zero (not verified unique)
```

Benutzt man dagegen die einfachere Darstellung mit Hilfe der Exponentialfunktion

```
function f (x: DerivType) : DerivType;
begin
   f := exp(-x);
end;
```

so erhält man mit dem Startintervall $[-10^2, +10^2]$ und der Toleranz $5 \cdot 10^{-20}$ das folgende Ergebnis:

```
0 interval enclosure(s)
```

womit die triviale Aussage **bewiesen** ist, daß e^{-x} in $x \in [-10^2, +10^2]$ keine Nullstellen besitzt.

> Zur Nullstelleneinschließung sollte man bei der Deklaration der zu untersuchenden Funktion stets den einfachsten Funktionsterm wählen!

9.2.2.2.2 $f(x) = e^{-3x} - \sin^3(x)$

Im zweiten Beispiel betrachten wir die Funktion

$$f(x) = e^{-3x} - \sin^3(x)$$

Wegen $f(x) \approx \sin^3(x)$ für $x \to +\infty$ sind die Nullstellen von $f(x)$ praktisch die dreifachen Nullstellen von $\sin^3(x)$ an den Stellen $x_n = n\pi$, die von der Prozedur `AllZeros` zwar eingeschlossen werden, für die aber keine Existenz- und Eindeutigkeitsaussagen geliefert werden. Mit der Deklaration

```
function f (x: DerivType) : DerivType;
begin
   f := exp(-3*x) - power(sin(x),3);
end;
```

liefert das **PASCAL-XSC** Programm `tstnlfz1` von Seite 441 mit dem Startintervall $[100, 101]$ und der Toleranz $5 \cdot 10^{-20}$ tatsächlich das Ergebnis:

```
[  1.00530964914873383628E+002,  1.00530964914873383628E+002 ]
 may contain a zero  (not verified unique)
```

Um auch für dieses Startintervall Existenz- und Eindeutigkeitsaussagen zu erhalten, betrachten wir die Umformung:

$$f(x) = e^{-3x} - \sin^3(x) \equiv e^{-3x} \left[1 - e^{3x} \cdot \sin^3(x)\right]$$

$f(x)$ und $g(x) = 1 - e^{3x} \cdot \sin^3(x)$ besitzen damit identische Nullstellen, und mit $u(x) := e^x \cdot \sin(x)$ gilt $1 - e^{3x} \cdot \sin^3(x) = 1 - u^3$, d.h. die Nullstellen von $f(x)$ sind auch die Nullstellen von $u - 1 = 0$ bzw. von

$$h(x) = e^x \cdot \sin(x) - 1$$

wobei jetzt aber alle Nullstellen von $h(x)$ **einfache** Nullstellen sind, für die `AllZeros` die gewünschten Existenz- und Eindeutigkeitsaussagen liefert. Mit der Deklaration

```
function f (x: DerivType) : DerivType;
begin
   f := exp(x)*sin(x) - 1;
end;
```

dem Startintervall $[100, 101]$ und der Toleranz $5 \cdot 10^{-20}$ erhält man in der Tat die maximalgenaue Einschließung:

```
[  1.00530964914873383630E+002,  1.00530964914873383631E+002 ]
 encloses a locally unique zero!
```

9.2.2.2.3 $f(x) = e^{-x^2} - \sin^3(x)$

Im dritten Beispiel betrachten wir die Funktion

$$f(x) = e^{-x^2} - \sin^3(x)$$

Da die Exponentialfunktion positiv ist, liegen alle Nullstellen x_i von $f(x)$ nur in den offenen Intervallen

$$x_i \in \boldsymbol{I}_n = \,]\,2n\pi, (2n+1)\pi\,[, \quad n = 0, \pm 1, \pm 2, \ldots$$

Wir wählen $n = 2$ und suchen damit alle Nullstellen x_i von $f(x)$ aus dem offenen Intervall $\boldsymbol{I}_2 = \,]\,4\pi, 5\pi\,[$. Mit der Deklaration[2]

```
function f (x: DerivType) : DerivType;
begin
   f := exp_x2(x) - power(sin(x),3);
end;
```

dem Startintervall $[12.566, 15.708] \supset \boldsymbol{I}_2$ und der Toleranz $5 \cdot 10^{-20}$ liefert das Programm `tstnlfz1` die Einschließungen

```
[  1.25663706143591729536E+001, 1.25663706143591729542E+001 ]
 may contain a zero  (not verified unique)
[  1.57079632679489661920E+001, 1.57079632679489661926E+001 ]
 may contain a zero  (not verified unique)
```

Um für obige Einschließungen die gewünschten Existenz- und Eindeutigkeitsaussagen zu gewinnen, betrachten wir die äquivalenten Umformungen:

$$\begin{aligned} f(x) &= e^{-x^2} - \sin^3(x) = 0 \Longleftrightarrow e^{-x^2} = \sin^3(x) \Longleftrightarrow e^{-x^2} = e^{3\ln(\sin(x))} \\ g(x) &= x^2 + 3 \cdot \ln(\sin(x)) = 0 \end{aligned}$$

Die Funktionen $f(x), g(x)$ besitzen also identische Nullstellen. Versucht man nun, mit den beiden obigen Einschließungen die Nullstellen von $g(x)$ zu berechnen, so erhält man Fehlermeldungen, da beide Einschließungen die Werte 4π bzw. 5π enthalten. Die ln–Funktion müßte damit für negative Argumente ausgewertet werden.

Um diese Fehlermeldungen zu vermeiden, benutzen wir die folgende Transformation:

$$\sin(x) = u, \qquad \text{mit:} \quad 0 < u < 1;$$

[2]Bzgl. `exp_x2(x)` vergl. die Tabelle auf Seite 351.

Wir müssen jetzt x als Funktion von u so ausdrücken, daß für $0 < u < 1$ die Aussage $x \in \boldsymbol{I}_2 = \,]\,4\pi, 5\pi\,[$ erfüllt ist. Dafür gibt es genau zwei Möglichkeiten:

$$x_1(u) = \arcsin(u) + 4\pi, \qquad 0 < u < 1 \;\rightarrow\; x_1 \in \,]\,4\pi, 5\pi\,[\tag{9.2}$$

$$x_2(u) = 5\pi - \arcsin(u), \qquad 0 < u < 1 \;\rightarrow\; x_2 \in \,]\,4\pi, 5\pi\,[\tag{9.3}$$

Damit ergeben sich die beiden Funktionen

$$\begin{aligned} g_1(u) &= \{4\pi + \arcsin(u)\}^2 + 3 \cdot \ln(u) \\ g_2(u) &= \{5\pi - \arcsin(u)\}^2 + 3 \cdot \ln(u), \end{aligned}$$

deren Nullstellen im Bereich $0 < u < 1$ einzuschließen sind. Mit der Deklaration[3]

```
function g1 (u: DerivType) : DerivType;
begin
   g1 := sqr( 4*PI_rrGi+arcsin(u) ) + 3*ln(u);
end;
```

und dem Startintervall $[10^{-511}, 0.9999]$ liefert das Programm `tstnlfz1` mit der Toleranz $5 \cdot 10^{-20}$ für die Nullstelle von $g_1(u)$ die folgende Einschließung:

```
[  1.37928741480072715705E-023, 1.37928741480072715706E-023 ]
 encloses a locally unique zero!
```

Bezeichnet man diese Einschließung mit $u1$, so liefert die folgende Anweisung `y1 := arcsin(u1)` das Ergebnis:

```
y1 = [1.37928741480072715705E-023, 1.37928741480072715707E-023]
```

Mit (9.2) erhält man für die erste Nullstelle x_1 von $f(x)$ die Einschließung:

$$x_1 \in [x_1] := 4\pi + y_1 \subset \boldsymbol{I}_2 = \,]\,4\pi, 5\pi\,[;$$

Ganz analog erhält man mit den Funktionen $g_2(u), x_2(u)$ das Intervall

```
y2 = [1.90858094534521008547E-036, 1.90858094534521008549E-036]
```

und damit nach (9.3) die Einschließung der zweiten Nullstelle x_2 von $f(x)$

$$x_2 \in [x_2] := 5\pi - y_2 \subset \boldsymbol{I}_2 = \,]\,4\pi, 5\pi\,[;$$

[3] Im Programm `tstnlfz1` ist der ursprüngliche Funktionsname f natürlich überall in g1 abzuändern, und die Module `rrgi_ari, rrgi_util` müssen zusätzlich importiert werden!

9.2.2.2.4 $f(x) = e^{-x} - x^{-200}$

Im nächsten Beispiel betrachten wir die Funktion

$$f(x) = e^{-x} - x^{-200}$$

deren Nullstellen im Intervall $[1, 10^5]$ einzuschließen sind. Mit der Deklaration

```
function f (x: DerivType) : DerivType;
begin
   f := exp(-x) - power(x,-200);
end;
```

und dem Startintervall $[1, 10^5]$ liefert das Programm `tstnlfz1` von Seite 441 mit der Toleranz $5 \cdot 10^{-20}$ nur Einschließungen mit der Information

```
may contain a zero (not verified unique)!
```

so daß eine Existenz- oder Eindeutigkeitsaussage nicht getroffen werden kann. Wir betrachten daher die Äquivalenzumformungen:

$$\begin{aligned} f(x) &= e^{-x} - x^{-200} = 0 \Longleftrightarrow e^{-x} = e^{-200 \cdot \ln(x)} \Longleftrightarrow \\ g(x) &= 200 \cdot \ln(x) - x = 0 \end{aligned}$$

Die Funktionen $f(x), g(x)$ besitzen damit in $[1, 10^5]$ identische Nullstellen, und mit der Deklaration

```
function f (x: DerivType) : DerivType;
begin
   f := 200*ln(x) - x;
end;
```

und dem Startintervall $[1, 10^5]$ liefert das **PASCAL-XSC** Programm `tstnlfz1` mit der Toleranz $5 \cdot 10^{-20}$ die beiden Nullstelleneinschließungen:

```
[  1.00503783662266051228E+000,  1.00503783662266051229E+000 ]
encloses a locally unique zero!
[  1.45679942701981632938E+003,  1.45679942701981632939E+003 ]
encloses a locally unique zero!
```

Erst mit der Funktion $g(x)$ erhält man also die gewünschten Existenz- und Eindeutigkeitsaussagen für die beiden einzigen Nullstellen der Funktion $f(x)$ im Intervall $[1, 10^5]$.

9.2.2.3 Eng benachbarte, einfache Nullstellen

9.2.2.3.1 $f(x) = 2x^2 - 0.03 \cdot e^{-[\,200\cdot(x-0.0675)\,]^2}$

Im ersten Beispiel betrachten wir die Funktion

$$f(x) = 2x^2 - 0.03 \cdot e^{-[\,200 \cdot (x - 0.0675)\,]^2} \tag{9.4}$$

Da der zweite Summand in (9.4) nur negativ ist und in der Umgebung des Ursprungs sehr kleine Werte annimmt, besitzt f in der Umgebung von $x = 0$ zwei sehr dicht zusammenliegende einfache Nullstellen, die einzuschließen sind. Mit der Deklaration

```
function f (x: DerivType) : DerivType;
begin
   f := 2*sqr(x) - 0.03*exp(-sqr(200*(x-0.0675)));
end;
```

und dem Startintervall $[-0.05, 0.06]$ liefert das Programm `tstnlfz1` von Seite 441 mit der Toleranz $5 \cdot 10^{-20}$ das Ergebis

```
[ -2.035385860091841705I5E-020,  2.731953128209638I4945E-020 ]
 may contain a zero (not verified unique)!
```

Die beiden dicht zusammenliegenden Nullstellen werden also nicht getrennt, sondern durch das obige Intervall $\boldsymbol{I}$ nur sehr grob eingeschlossen. Zur Trennung der Nullstellen nehmen wir die folgende Koordinatentransformation vor:

$$u = a \cdot (x - \mathrm{mid}(\boldsymbol{I})) \tag{9.5}$$

Ist $\boldsymbol{I}$ eine Einschließung der Nullstellen, so werden diese durch $x - \mathrm{mid}(\boldsymbol{I})$ zunächst in die unmittelbare Nähe des Ursprungs abgebildet, und der Faktor $a \gg 1$ spreizt dann die Nullstellen bzgl. der Koordinate u so stark, daß diese wirklich getrennt eingeschlossen werden können.

Für unser Beispiel wählen wir $\mathrm{mid}(\boldsymbol{I}) = 0,\ a = 10^{+60}$ und erhalten:

$$x(u) = 10^{-60} \cdot u \tag{9.6}$$

$$f(x(u)) = g(u) = 2 \cdot (10^{-60}u)^2 - 0.03 \cdot e^{-(200\cdot(10^{-60}u-0.0675))^2} \tag{9.7}$$

$$x \in [-0.05, 0.06] \quad \longrightarrow \quad u \in [-5 \cdot 10^{+58}, 6 \cdot 10^{+58}] \tag{9.8}$$

Mit der Deklaration

```
function g (u: DerivType) : DerivType;
begin
   g := 2e-120*sqr(u) - 0.03*exp(-sqr(200*(1e-60*u-0.0675)));
end;
```

und dem Startintervall $[-5 \cdot 10^{+58}, 6 \cdot 10^{+58}]$ liefert das Programm `tstnlfz1` von Seite 441 mit der Toleranz $5 \cdot 10^{-20}$ die Nullstelleneinschließungen der Funktion $g(u)$:

```
[ -3.25807416477039770283E+019, -3.25807416477039770282E+019 ]
 encloses a locally unique zero!
[  3.25807416477039770282E+019,  3.25807416477039770283E+019 ]
 encloses a locally unique zero!
```

und nach (9.6) lauten die maximalgenauen Einschließungen der beiden Nullstellen von $f(x)$ im Intervall $[-0.05, 0.06]$

```
[ -3.25807416477039770283E-041, -3.25807416477039770282E-041 ]
 encloses a locally unique zero!
[  3.25807416477039770282E-041,  3.25807416477039770283E-041 ]
 encloses a locally unique zero!
```

Beachten Sie bitte, daß wir damit den mathematischen Beweis geführt haben, daß $f(x)$ in $[-0.05, 0.06]$ genau zwei Nullstellen besitzt. Ein solcher Beweis ist natürlich nur dann schlüssig, wenn es gelingt, alle Nullstellen im Intervall $[-0.05, 0.06]$ so einzuschließen, daß das Programm `tstnlfz1` mit Hilfe der Prozedur `AllZeros` für jede Einschließung den Kommentar

```
encloses a locally unique zero!
```

liefert. Beachten Sie bitte weiter, daß die Nullstelleneinschließungen nur mit der in **PASCAL-XSC** implementierten Intervallrechnung möglich ist, mit deren Hilfe die intervallmäßige Differentiationsarithmetik im Hilfsmodul `ddf_rrg` realisiert wurde.

9.2.2.3.2 $f(x) = 2x^2 - 0.03 \cdot e^{-[\,10^{20}\cdot(x-0.0675)\,]^2}$

Wir betrachten jetzt die Funktion

$$f(x) = 2x^2 - 0.03 \cdot e^{-[\,10^{20} \cdot (x - 0.0675)\,]^2} \tag{9.9}$$

Der große Faktor 10^{20} bewirkt jetzt, daß die obige Exponentialfunktion in einer sehr kleinen Umgebung von 0.0675 fast alle Werte zwischen 0 und 1 annimmt und außerhalb dieser Umgebung praktisch verschwindet. $f(x)$ wird daher im Intervall $[10^{-2}, 10^{-1}]$ zwei Nullstellen besitzen, die möglichst optimal einzuschließen sind.

Da beide Nullstellen mit dem Programm `tstnlfz1` von Seite 441 nicht getrennt werden, benutzen wir die in (9.5) angegebene Transformation und wählen $a = 10^{+20}$ und $\mathrm{mid}(\boldsymbol{I}) = 0.0675$. Wir erhalten damit die Beziehungen:

$$u(x) = 10^{+20} \cdot (x - 0.0675) \tag{9.10}$$

$$x(u) = 10^{-20} \cdot u + 0.0675 \tag{9.11}$$

$$f(x(u)) = g(u) = 2 \cdot (10^{-20}u + 0.0675)^2 - 0.03 \cdot e^{-u^2} \tag{9.12}$$

$$x \in [10^{-2}, 10^{-1}] \quad \longrightarrow \quad u \in \boldsymbol{I}_u = [-5.75 \cdot 10^{+18}, 3.25 \cdot 10^{+18}] \tag{9.13}$$

Mit der Deklaration[4]

```
function g (u: DerivType) : DerivType;
begin
   g := 2*sqr(1e-20*u + 0.0675) - 0.03*exp_x2(u);
end;
```

dem Startintervall $[-5.75\cdot10^{18}, +3.25\cdot10^{18}]$ und der Toleranz $5\cdot10^{-20}$ erhält man die maximalgenauen Einschließungen der beiden Nullstellen von $g(u)$:

```
I1=[-1.09158155184456046195E+000,-1.09158155184456046194E+000]
 encloses a locally unique zero!
I2=[ 1.09158155184456046164E+000, 1.09158155184456046165E+000]
 encloses a locally unique zero!
```

Nach (9.11) werden dann die beiden einzigen Nullstellen von $f(x)$ im Intervall $[10^{-2}, 10^{-1}]$ eingeschlossen durch:

$$\begin{aligned} \boldsymbol{I}_{x1} &:= 0.0675 + 10^{-20} \cdot \mathtt{I1} \\ \boldsymbol{I}_{x2} &:= 0.0675 + 10^{-20} \cdot \mathtt{I2} \end{aligned}$$

[4] Im Quelltext des Programms `tstnlfz1` muß der ursprüngliche Funktionsname `f` natürlich in `g` umgewandelt werden!

Wegen des großen Faktors 10^{20} in (9.9) besitzt $f(x)$ in einer sehr kleinen Umgebung des Ursprungs sehr eng benachbarte Nullstellen, die im Intervall $[-10^{-500}, +10^{-500}]$ einzuschließen sind. Da $f(0) < 0$ ist, beschränken wir uns zunächst auf das halboffene Intervall $(0, 10^{-500}]$.

Die Transformation $u = a \cdot x$ von Seite 450 führt jetzt nicht zum Ziel, da $a > 0$ so groß gewählt werden müßte, daß a selbst im `rrG`–Format nicht mehr darstellbar wäre!

Wir wählen deshalb jetzt die Transformation:

$$x = e^u > 0 \tag{9.14}$$

$$x \in (0, 10^{-500}] \quad \longrightarrow \quad u \in (\,-\infty, -500 \cdot \ln(10)\,] \tag{9.15}$$

$$f(x(u)) = g(u) := 2 \cdot e^{2u} - 0.03 \cdot e^{-[10^{20} \cdot (e^u - 0.0675)]^2} \tag{9.16}$$

Da das Intervall rechts in (9.15) auf einem Rechner nicht realisierbar ist, wählen wir $u \in \boldsymbol{I} = [-10^{+500}, -10^{+3}]$ und suchen eine Einschließung der Nullstelle von $g(u)$ in $\boldsymbol{I}$. Dabei zeigt sich jedoch, daß bei der Intervallauswertung der Funktion e^{2u} in (9.16) die Intervalluntergrenzen auf Null gerundet werden, wodurch eine Nullstelleneinschließung verhindert wird. Abhilfe schafft die äquivalente Umformung:

$$\begin{array}{rcl} g(u) & = & 0 \quad \Longleftrightarrow \\ h(u) & := & \ln(\dfrac{200}{3}) + 2u + [\,10^{20} \cdot (e^u - 0.0675)\,]^2 = 0 \end{array}$$

$g(u)$ und $h(u)$ besitzen daher in $\boldsymbol{I} = [-10^{+500}, -10^{+3}]$ identische Nullstellen. Mit der Deklaration[5]

```
function h (u: DerivType) : DerivType;
begin
   h := ln(rrGintval(200)/3) + 2*u + sqr(1e20*(exp(u)-0.0675));
end;
```

und dem Startintervall $\boldsymbol{I} = [-10^{+500}, -10^{+3}]$ liefert das Programm `tstnlfz1` von Seite 441 mit der Toleranz $5 \cdot 10^{-20}$ das Ergebnis:

```
Iu = [-2.278125000...001E+037,-2.278124999...999E+037]
 encloses a locally unique zero!
```

Nach (9.14) besitzt dann $f(x)$ in $[e^{-10^{+500}}, 10^{-500}]$ genau eine Nullstelle, die durch $e^{\mathtt{Iu}}$ eingeschlossen wird.

[5]Im Quelltext des Programms `tstnlfz1` muß der ursprüngliche Funktionsname `f` natürlich in `h` umgewandelt werden!

Anmerkungen:

- Beachten Sie bitte, daß z.B. `rrGintval(200/3)` den Bruch 200/3 nicht wirklich einschließt, da jetzt zunächst `200/3` mit einem Rundungsfehler berechnet wird und erst danach eine Einschließung dieser Approximation erfolgt.

- Beachten Sie bitte, daß $e^{\mathtt{Iu}}$ eine Nullstelle von der Größenordnung $10^{-9.858\ldots\cdot 10^{+36}}$ einschließt. Da diese sehr kleine Zahl noch nicht einmal im `rrG`–Format darstellbar ist, konnte die Transformation (9.5) natürlich nicht erfolgreich sein, weil der Faktor $a > 0$ in diesem Datenformat nicht groß genug gewählt werden kann.

- Die Einschließung der negativen Nullstelle von $f(x)$ in $[-10^{-500}, 0)$ erfolgt ganz analog mit Hilfe der Transformation $x = -e^u$ und bleibt dem Leser überlassen.

- Die letzten Beispiele sollten zeigen, daß man mit Hilfe der Prozedur `AllZeros` einfache Nullstellen von $f(x)$ auch noch in ganz extremen Fällen einschließen kann, wenn man nach einer geeigneten Koordinatentransformation die Nullstellen von $f(x(u)) = g(u)$ einschließt. Die Tranformations-Funktionen $x = x(u)$ müssen dabei natürlich streng monoton sein, damit keine Nullstellen verloren gehen oder zusätzlich erzeugt werden.

- Bei einem zu klein gewählten Toleranz–Wert kann es sein, daß die Laufzeit erheblich ansteigt. In diesem Fall sollte man dann zuerst mit einem größeren Toleranz–Wert (10^{-8}) eine zunächst noch grobe (aber eindeutige!) Einschließung $\boldsymbol{I}_1$ der gesuchten Nullstelle berechnen, um dann mit $\boldsymbol{I}_1$ als Startintervall und einem viel kleineren Toleranzwert (10^{-17}) eine zweite genauere Einschließung $\boldsymbol{I}_2 \subset \boldsymbol{I}_1$ zu erhalten. Diese Vorgehensweise ist besonders dann zu empfehlen, wenn die erste grobe Einschließung $\boldsymbol{I}_1$ die Zahl Null enthält. Ist dann $x_0 = 0$ tatsächlich eine Nullstelle der zu untersuchenden Funktion, so kann man oft schon nach ein oder zwei Schritten $x_0 = 0$ mit seinem Punktintervall $[0, 0]$ maximalgenau einschließen. Testen Sie diese Vorgehensweise mit der Funktion $f(x) = \sin(x)$ und den Startintervallen $[-0.5, 1]$, $[-1, 1]$.

9.2.2.4 Existenz- und Eindeutigkeitsaussagen

Wenn die Prozedur `AllZeros` von Seite 439 mit dem Fehlerkode `Err=0` erfolgreich beendet wurde und wenn die Meldung

```
may contain a zero (not verified unique)!'
```

nicht ausgegeben wurde, dann ist damit der Beweis geführt worden, daß die untersuchte Funktion f im angegebenen Startintervall genau so viele Nullstellen besitzt, wie Intervall–Einschließungen mit dem Kommentar

```
encloses a locally unique zero!
```

aufgelistet werden. Jedes angegebene Intervall enthält also genau eine Nullstelle von f, und außerhalb dieser Einschließungen wird garantiert, daß f keine weiteren Nullstellen besitzt.

Die Meldung

```
may contain a zero (not verified unique)!'
```

wird in folgenden Fällen ausgegeben:

1. Der Toleranzwert wurde zu groß gewählt.
2. Im denormalisierten Zahlenbereich werden sehr kleine Funktionswerte von f bei der Intervallauswertung zu grob eingeschlossen.
3. Verschiedene Nullstellen von f liegen zu dicht nebeneinander.
4. Die Funktion f besitzt mehrfache Nullstellen.
5. Der mathematische Term, der die Funktion f definiert, wurde ungeschickt gewählt.
6. f besitzt in x_0 eine einfache Nullstelle mit $0 < |f'(x_0)| \ll 1$.

Im nächsten Beispiel beweisen wir, daß die Funktion

$$f(x) = \tan(x) \cdot \tanh(x) - 1$$

im Intervall $[0, 1.5]$ genau eine Nullstelle besitzt. Mit der Deklaration

```
function f (x: DerivType) : DerivType;
begin
   f := tan(x)*tanh(x)-1;
end;
```

und dem Startintervall $[0, 1.5]$ liefert das Programm `tstnlfz1` von Seite 441 mit der Toleranz $5 \cdot 10^{-20}$ das Ergebnis

```
[  9.37552034355980583222E-001,  9.37552034355980583223E-001 ]
 encloses a locally unique zero!

1 interval enclosure(s)
```

Mit Hilfe numerischer Methoden haben wir damit **bewiesen**, daß obige Funktion im Intervall $[0, 1.5]$ genau eine Nullstelle besitzt.

Ganz analog beweisen wir im nächsten Beispiel, daß die Funktion

$$f(x) = e^x - x - 1.01$$

im Intervall $[-1, +1]$ genau zwei Nullstellen besitzt. Mit der Deklaration

```
function f (x: DerivType) : DerivType;
begin
   f := exp(x) - x - 1.01;
end;
```

und dem Startintervall $[0, 1.5]$ liefert das Programm `tstnlfz1` von Seite 441 mit der Toleranz $5 \cdot 10^{-20}$ das Ergebnis

```
[ -1.44834751066834685866E-001, -1.44834751066834685865E-001 ]
 encloses a locally unique zero!
[  1.38165122379394735870E-001,  1.38165122379394735871E-001 ]
 encloses a locally unique zero!

2 interval enclosure(s)
```

9.2.3 Kurvendiskussion

Zur Diskussion einer in einem Intervall $[a, b] \in I\!I\!R$ mindestens dreimal stetig differenzierbaren Funktion $f : I\!R \to I\!R$ benötigt man u.a Informationen über

1. Nullstellen von f, f', f''
2. Monotonie–Verhalten[6]
3. Lokale Extrema
4. Krümmungs–Verhalten
5. Wendestellen

Mit Hilfe der Prozedur `Function_Analyser(...)` aus dem Hilfsmodul `nlfz3` lassen sich diese Informationen automatisch und vor allem mathematisch zuverlässig gewinnen, wenn die Nullstellen von f, f', f'' alle einfach sind. Dabei wird jede Nullstelle durch ein Intervall i.a. maximalgenau eingeschlossen, und es ist dabei sichergestellt, daß außer dieser einfachen Nullstelle keine weitere Nullstelle in dem Intervall enthalten ist.

Liegen mehrere einfache Nullstellen so dicht nebeneinander[7], daß ihre Eindeutigkeit vom Algorithmus nicht mehr erkannt wird, so kann man mit der auf Seite 450 beschriebenen Transformation (9.5) diese Nullstellen so weit auseinanderziehen, daß der Eindeutigkeitsnachweis wieder gelingt.

Ein weiterer, ganz wesentlicher Vorteil bei der Anwendung der Prozedur `Function_Analyser` besteht auch darin, daß die Ableitungen von f nicht von Hand berechnet werden müssen. Lediglich der Funktionsterm von f ist im Quelltext des aufrufenden Programms einzugeben!

Mehrfache Nullstellen werden stets durch Intervalle eingeschlossen mit der Anmerkung, daß die Existenz und Eindeutigkeit etwaiger Nullstellen in diesen Intervallen nicht garantiert werden kann. Eine Kurvendiskussion wird nur dann durchgeführt, wenn das Startintervall solche Einschließungen nicht enthält.

Mehrfache Nullstellen der Funktion f lassen sich jedoch eindeutig nachweisen, wenn es gelingt, die einfache Nullstelle von f/f' mit Hilfe von `Function_Analyser` einzuschließen. Dazu muß jedoch f' explizit berechnet werden, und es muß gelingen, den Funktionsterm von f/f' so umzuformen,

[6] f ist (streng) monoton wachsend $\iff x_2 > x_1 \Longrightarrow f(x_2)\ (>)\ \geq f(x_1)$

[7] Dieser Fall wird in der Praxis zwar nur selten eintreten, der Algorithmus wird solche Nullstellen jedoch niemals übersehen!

daß sein Nenner in einer Umgebung dieser einfachen Nullstelle nicht mehr verschwindet. Beispiele für solche Umformungen findet man ab Seite 441.

9.2.3.1 Eine Routine zur Kurvendiskussion

Die Kurvendiskussion wird mit Hilfe der Prozedur `Function_Analyser(..)` durchgeführt, die vom Hilfsmodul `nlfz3` exportiert wird. Die dazu benötigte Intervall–Differentiationsarithmetik liefert das Hilfsmodul `d3f_rrg`, das den folgenden Datentyp exportiert:

```
type Grad3Type = record
                     f, df, ddf, dddf : rrGinterval;
                 end;                                  (d3f_rrg)
```

Ist die zu untersuchende Funktion f z.B. definiert durch

$$f(x) = x \cdot \sin[\sin(2.2x) + x/2] - 0.55$$

so ist f im Quelltext eines aufrufenden Programms zu deklarieren als:

```
function f(x : Deriv3Type) : Deriv3Type;
begin
   f := x*sin( sin(2.2*x) + 0.5*x ) - 0.55;
end;
```

Die Darstellung von f ist also denkbar einfach und entspricht genau der mathematischen Notation!

Die Prozedur `Function_Analyser` ist im Hilfsmodul `nlfz3` wie folgt deklariert:

```
procedure Function_Analyser (
    function f ( x : Deriv3Type) : Deriv3Type;
    DerivOrder, Zero_max  : integer;
    SearchInterval        : Linterval;
    Tolerance             : integer );
```

Alle Ergebnisse der Kurvendiskussion werden auf dem Monitor ausgegeben. Die Eingangsgrößen haben folgende Bedeutung:

1. `f` ist die zu untersuchende Funktion vom Typ `deriv3Type`.

2. `Zero_max` ist die Maximalzahl von Nullstellen, die intern gespeichert werden können. Werden mehr Nullstellen gefunden, so erfolgt eine entsprechende Fehlermeldung mit Programmabbruch. `Zero_max` ist dann vom Anwender im aufrufenden Programm geeignet zu vergrößern.

3. `SearchInterval` ist das Intervall, in dem die Kurvendiskussion durchzuführen ist und das keine mehrfachen Nullstellen von `f` enthalten darf.

4. `Tolerance` ist eine ungefähre relative Fehlerschranke, mit der die Nullstellen einzuschließen sind. Wird `Tolerance` vom Anwender kleiner als $5 \cdot 10^{-20}$ gewählt, so wird der Parameter intern auf $5 \cdot 10^{-20}$ gesetzt. Bei zu klein gewählten Werten kann in besonderen Situationen die Laufzeit so stark anwachsen, daß das Programm mit der Tastenkombination `[Strg]+[C]` abgebrochen werden muß. Mit `Tolerance` $= 10^{-14}$ erhält man in fast allen Fällen brauchbare Ergebnisse.

5. `DerivOrder` bestimmt die Ordnung der Ableitung von f und muß mit 0,1,2 vorgegeben werden. Die entsprechende Arbeitsweise der Prozedur `Function_Analyser` wird in den folgenden Punkten ausführlich beschrieben.

DerivOrder = 0

Alle Bildschirmausgaben beziehen sich auf die Funktion f.

1. Alle Nullstellen–Einschließungen von f werden mit Angabe des jeweiligen Vorzeichenwechsels ausgegeben. Das Symbol $(-,+)$ bedeutet dabei, daß der Graph von f die x–Achse von negativen zu positiven Funktionswerten durchläuft.
2. Innerhalb des Startintervalls werden alle die Intervalle ausgegeben, in denen f jeweils identische Vorzeichen besitzt. $(-)$ bedeutet dabei, daß $f(x)$ im entsprechenden Intervall stets negativ ist.

DerivOrder = 1

Alle Bildschirmausgaben beziehen sich auf die 1. Ableitung f'.

1. Alle Nullstellen–Einschließungen von f' werden mit Angabe des jeweiligen Vorzeichenwechsels ausgegeben. Das Symbol $(+,-)$ bedeutet dabei, daß der Graph von f' die x–Achse von positiven zu negativen Funktionswerten durchläuft. Die Funktion f besitzt damit im entsprechenden Intervall eine lokale Maximumstelle.
2. Für jede Nullstelleneinschließung von f' wird eine Einschließung aller entsprechenden Funktionswerte $f(x)$ berechnet, so daß damit eine Einschließung der jeweiligen Ordinate des Extremwerts geliefert wird.
3. Innerhalb des Startintervalls werden alle die Intervalle ausgegeben, in denen f' jeweils identische Vorzeichen besitzt. $(-)$ bedeutet dabei, daß $f'(x)$ im entsprechenden Intervall stets negativ ist. Im gleichen Intervall ist damit f streng monoton fallend.

DerivOrder = 2

Alle Bildschirmausgaben beziehen sich auf die 2. Ableitung f''.

1. Alle Nullstellen–Einschließungen von f'' werden mit Angabe des jeweiligen Vorzeichenwechsels ausgegeben. Das Symbol $(+,-)$ bedeutet dabei, daß der Graph von f'' die x–Achse von positiven zu negativen Funktionswerten durchläuft. Die Funktion f besitzt damit im entsprechenden Intervall eine Wendepunkt–Abszisse.
2. Für jede Nullstelleneinschließung von f'' wird eine Einschließung aller entsprechenden Funktionswerte $f(x)$ berechnet, so daß damit eine Einschließung der jeweiligen Wendepunkt–Ordinate geliefert wird.
3. Innerhalb des Startintervalls werden alle die Intervalle ausgegeben, in denen f'' jeweils identische Vorzeichen besitzt. $(-)$ bedeutet dabei, daß $f''(x)$ im entsprechenden Intervall stets negativ ist. Im gleichen Intervall ist damit der Graph von f rechtsgekrümmt.

Wenn also im Startintervall die Funktionen f, f', f'' nur einfache Nullstellen besitzen, kann man nach Aufruf der Prozedur `Function_Analyser` mit den Parameterwerten `DerivOrder = 0,1,2` alle für die Diskussion der Funktion f notwendigen Informationen wie Nullstellen, Extremwerte, Wendepunkte, Monotonie- und Krümmungsintervalle vom Bildschirm ablesen.

Die nachfolgende Tabelle enthält alle Funktionen vom Typ `Deriv3Type`, aus denen die zu untersuchende Funktion f zusammengesetzt werden kann:

Von `d3f_rrg` exportierte Funktionen vom Typ `Deriv3Type`			
`x : DerivType; a : rrGinterval; n : integer; n = 0,1,...`			
Funktion	**Aufruf**	**Funktion**	**Aufruf**
x^2	`sqr(x)`	$\cos(x)$	`cos(x)`
x^n	`power(x,n)`	$\tan(x)$	`tan(x)`
x^x	`power(x)`	$\cot(x)$	`cot(x)`
x^a	`power(x,a)`	$\sin(\pi \cdot x)$	`sinpi(x)`
$\sqrt{x}$	`sqrt(x)`	$\cos(\pi \cdot x)$	`cospi(x)`
$\sqrt{1+x}-1$	`sqrt1pm1(x)`	$\tan(\pi \cdot x)$	`tanpi(x)`
$\sqrt{1-x^2}$	`sqrt1mx2(x)`	$\cot(\pi \cdot x)$	`cotpi(x)`
$\sqrt{x^2-1}$	`sqrtx2m1(x)`	$\arcsin(x)$	`arcsin(x)`
$\sqrt{1+x^2}$	`sqrt1px2(x)`	$\arccos(x)$	`arccos(x)`
e^x	`exp(x)`	$\arctan(x)$	`arctan(x)`
e^x-1	`expm1(x)`	$\mathrm{arccot}(x)$	`arccot(x)`
e^{-x^2}	`exp_x2(x)`	$\sinh(x)$	`sinh(x)`
2^x	`exp2(x)`	$\cosh(x)$	`cosh(x)`
10^x	`exp10(x)`	$\tanh(x)$	`tanh(x)`
$\ln(x)$	`ln(x)`	$\coth(x)$	`coth(x)`
$\ln(e \cdot x)$	`ln_ex(x)`	$\mathrm{arsinh}(x)$	`arsinh(x)`
$\ln(1+x)$	`ln1p(x)`	$\mathrm{arcosh}(x)$	`arcosh(x)`
$\log_a(x)$	`loga(a,x)`	$\mathrm{arcosh}(1+x)$	`arcosh1p(x)`
$\log_2(x)$	`log2(x)`	$\mathrm{artanh}(x)$	`artanh(x)`
$\log_{10}(x)$	`log10(x)`	$\mathrm{arcoth}(x)$	`arcoth(x)`
$\log_{10}(1+x)$	`log1p(x)`		
$\sin(x)$	`sin(x)`		

9.2.3.2 Programmbeispiele

9.2.3.2.1 $f = x \cdot \sin[\sin(2.2x) + x/2] - 0.55$

Im ersten Beispiel betrachten wir die Funktion

$$f(x) = x \cdot \sin[\sin(2.2x) + x/2] - 0.55$$

die im Intervall $[-1, 1.6]$ zu diskutieren ist. Das nachfolgende Programm

```
program nlfz3_1;
use
  li_ari,            { Interval arithmetic         }
  d3f_rrg,           { Differentiation arithmetic  }
  nlfz3;             { procedure Function_Analyser }

const Zero_max = 50; { Maximum number of zeros to be computed }

var DerivOrder     : integer;  { Desired Order of derivative   }
   SearchInterval : Linterval;
   Tolerance      : Longreal; { Tolerance for zero inclusions }

function f (x: Deriv3Type) : Deriv3Type;
begin
   f := x*sin( sin(2.2*x) + 0.5*x ) - 0.55;
end;

begin
   repeat
      writeln('Order (0,1,2) of the derivative = ?');
      read(DerivOrder);
   until (DerivOrder >= 0) and (DerivOrder <=2);
   write('Search interval      : ');  read(SearchInterval);
   write('Tolerance (relative) : ');  read(Tolerance);
   Function_Analyser(f,DerivOrder,Zero_max,SearchInterval,
                                                  Tolerance);
end.
```

liefert mit den jeweiligen Eingaben die entsprechenden Ergebnisse:

```
Order (0,1,2) of the derivative = ?
0                               <enter>
Search interval      : [-1,1.6]  <enter>
Tolerance (relative) : 1e-20     <enter>

   *****************************************************
   *   The following results refer to the function f(x)   *
   *****************************************************

   The search interval
        [ -1.00000000000000000000E+000,  1.60000000000000000000E+000 ]
   contains exactly 2 inclusion(s), each with a single and unique zero:

   [ -5.80143019596673568787E-001,  -5.80143019596673568786E-001 ],  (+1,-1)
   [  5.80143019596673568786E-001,   5.80143019596673568787E-001 ],  (-1,+1)

   Open interval(s) with f(x) values of identical sign are:
   ( -1.00000000000000000000E+000,  -5.80143019596673568787E-001 )     (+)
   ( -5.80143019596673568786E-001,   5.80143019596673568786E-001 )     (-)
   (  5.80143019596673568787E-001,   1.60000000000000000000E+000 )     (+)


Order (0,1,2) of the derivative = ?
1                               <enter>
Search interval      : [-1,1.6]  <enter>
Tolerance (relative) : 1e-20     <enter>

   ****************************************************************
   *   The following results refer to the first derivative f'(x)   *
   ****************************************************************

   The search interval
        [ -1.00000000000000000000E+000,  1.60000000000000000000E+000 ]
   contains exactly 2 inclusion(s), each with a single and unique zero:

   [ -1.00000000000000000000E-032,  1.00000000000000000000E-032 ],  (-1,+1)
   [  1.21533590237459010301E+000,  1.21533590237459010302E+000 ],  (+1,-1)

   The inclusions of the corresponding function values f(x) are:
   [ -5.50000000000000000001E-001, -5.49999999999999999999E-001 ]
   [  5.09395679524838402181E-001,  5.09395679524838402205E-001 ]

   Open interval(s) with f'(x) values of identical sign are:
   ( -1.00000000000000000000E+000, -1.00000000000000000000E-032 )     (-)
   (  1.00000000000000000000E-032,  1.21533590237459010301E+000 )     (+)
   (  1.21533590237459010302E+000,  1.60000000000000000000E+000 )     (-)
```

```
Order (0,1,2) of the derivative = ?
2                               <enter>
Search interval      : [-1,1.6]  <enter>
Tolerance (relative) : 1e-20     <enter>
```

```
*****************************************************************
*   The following results refer to the second derivative f''(x)   *
*****************************************************************

The search interval
     [ -1.00000000000000000000E+000,  1.60000000000000000000E+000 ]
contains exactly 2 inclusion(s), each with a single and unique zero:

[ -4.10132658840226079072E-001, -4.10132658840226079067E-001 ],  (-1,+1)
[  4.10132658840226079069E-001,  4.10132658840226079070E-001 ],  (+1,-1)

The inclusions of the corresponding function values f(x) are:
[ -2.07159896085191323452E-001, -2.07159896085191323445E-001 ]
[ -2.07159896085191323450E-001, -2.07159896085191323447E-001 ]

Open interval(s) with f''(x) values of identical sign are:
( -1.00000000000000000000E+000, -4.10132658840226079072E-001 )     (-)
( -4.10132658840226079067E-001,  4.10132658840226079069E-001 )     (+)
(  4.10132658840226079070E-001,  1.60000000000000000000E+000 )     (-)
```

Mit diesen Ergebnissen erhalten wir für die Funktion

$$f(x) = x \cdot \sin[\,\sin(2.2x) + x/2\,] - 0.55$$

im Intervall $[-1, 1.6]$ folgende Informationen:

Nullstelleneinschließungen:

```
[ -5.80143019596673568787E-001,  -5.80143019596673568786E-001 ],  (+1,-1)
[  5.80143019596673568786E-001,   5.80143019596673568787E-001 ],  (-1,+1)
```

Einschließungen relativer Extrema:

```
Minimum-Abszisse enthalten in:
[ -1.00000000000000000000E-032,  1.00000000000000000000E-032 ]
Relatives Minimum enthalten in:
[ -5.50000000000000000001E-001, -5.49999999999999999999E-001 ]

Maximum-Abszisse enthalten in:
[  1.21533590237459010301E+000,  1.21533590237459010302E+000 ]
Relatives Maximum enthalten in:
[  5.09395679524838402181E-001,  5.09395679524838402205E-001 ]
```

Monotonie-Intervalle

```
( -1.00000000000000000000E+000, -1.00000000000000000000E-032 )  (fallend )
(  1.00000000000000000000E-032,  1.21533590237459010301E+000 )  (wachsend)
(  1.21533590237459010302E+000,  1.60000000000000000000E+000 )  (fallend )
```

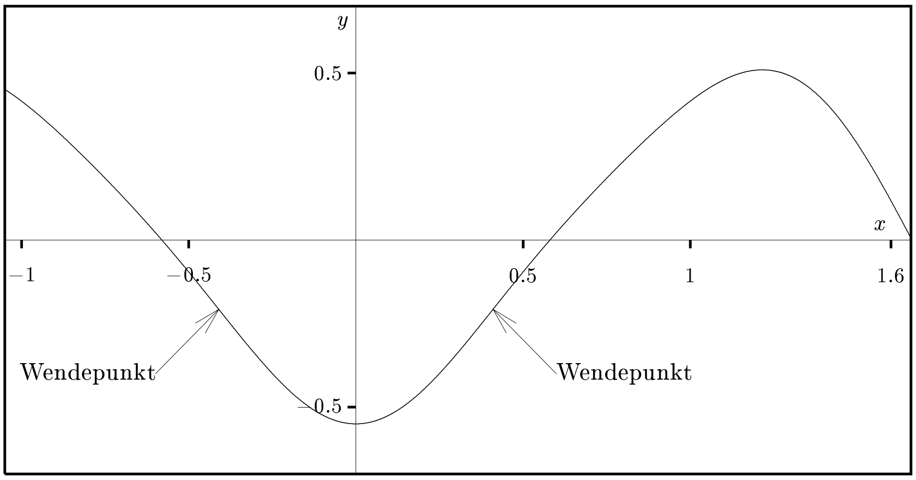
Wendepunkt-Einschließungen:

```
1. Wendepunkt-Abszisse enthalten in:
[ -4.10132658840226079072E-001, -4.10132658840226079067E-001 ]
1. Wendepunkt-Ordinate enthalten in:
[ -2.07159896085191323452E-001, -2.07159896085191323445E-001 ]

2. Wendepunkt-Abszisse enthalten in:
[  4.10132658840226079069E-001,  4.10132658840226079070E-001 ]
2. Wendepunkt-Ordinate enthalten in:
[ -2.07159896085191323450E-001, -2.07159896085191323447E-001 ]
```

Krümmungs-Intervalle

```
( -1.00000000000000000000E+000, -4.10132658840226079072E-001 )  (rechts)
( -4.10132658840226079067E-001,  4.10132658840226079069E-001 )  (links )
(  4.10132658840226079070E-001,  1.60000000000000000000E+000 )  (rechts)
```

$$f(x) = x \cdot \sin[\sin(2.2x) + x/2] - 0.55$$



Hinweis:

- Die Einschließung $[-10^{-32}, +10^{-32}]$ der ersten Nullstelle von f' kann auf das Punktintervall $[0, 0]$ verbessert werden, wenn man die Prozedur `Function_Analyser` aufruft mit den Parametern: `DerivOrd=1, SearchInterval=[-1E-32,1E-32], Tolerance=1E-20`.
- Fast alle nicht-maximalgenauen Nullstellen–Einschließungen kann man nach obiger Methode verbessern.

9.2.3.2.2 $f(x) = 2 \cdot \sin(x) + \sin(2x)$

Im nächsten Beispiel betrachten wir die 2π-periodische Funktion

$$f(x) = 2 \cdot \sin(x) + \sin(2x)$$

die im Intervall $[0, 6.3]$ mit Hilfe der Prozedur `Function_Analyser` aus dem Hilfsmodul `nlfz3` zu diskutieren ist.

Ändert man im Programm `nlfz3_1` von Seite 462 die zu untersuchende Funktion `f (x : Deriv3Type) : Deriv3Type` entsprechend ab, so erhält man mit den Eingangsparametern

```
DerivOrder = 0,  SearchInterval = [0,6.3],  Tolerance = 2e-10
```

die folgende Bildschirmausgabe:

```
   ********************************************************
   *   The following results refer to the function f(x)   *
   ********************************************************

****************************************************************************
Sorry
The following 1 interval may contain a zero (not verified unique!)
        [  3.14159264615326942287E+000,  3.14159266052998223083E+000 ]

Remarks:

   The number of this interval(s) can probably be reduced by choosing
   tolerance values different from  2.00000000000000000000E-010  TRY IT !!

   The number of the above interval(s) can probably be reduced by
   choosing small search intervals containing only one of the above
   interval(s),  TRY IT !!

   An interval containing a multiple zero cannot be removed!

   For the following search use only start intervals, which contain none of
   the unremovable interval(s)!

****************************************************************************
```

Da sich das obige Intervall nicht weiter reduzieren läßt und die Transformation (9.5) von Seite 450 auch keine getrennte, einfache Nullstellen liefert, kann man vermuten[8], daß das obige Intervall eine mehrfache Nullstelle einschließt, die von der Prozedur `Function_Analyser` zwar stets eingeschlossen, jedoch nicht als eindeutig erkannt wird.

[8]Eine mathematisch gesicherte Aussage ist das natürlich nicht!

Für den Eingangsparameter `DerivOrder=0` diskutieren wir die Funktion f daher in den beiden getrennten Intervallen:

$$S_1 = [0, 3.14159264], \qquad S_2 = [3.14159267, 6.3]$$

und überspringen dabei im ursprünglichen Startintervall $0, 6.3$ das Intervall $S_0 = [3.14159264, 3.14159267]$. Das nachfolgende Programm

```
program nlfz3_2;
use
  li_ari,            { Interval arithmetic         }
  d3f_rrg,           { Differentiation arithmetic  }
  nlfz3;             { procedure Function_Analyser }

const Zero_max = 50; { Maximum number of zeros to be computed }

var DerivOrder    : integer;  { Desired Order of derivative   }
   SearchInterval : Linterval;
   Tolerance      : Longreal; { Tolerance for zero inclusions }

function f (x: Deriv3Type) : Deriv3Type;
begin
   f := 2*sin(x) + sin(2*x);
end;

begin
   repeat
      writeln('Order (0,1,2) of the derivative = ?');
      read(DerivOrder);
   until (DerivOrder >= 0) and (DerivOrder <=2);
   write('Search interval      : ');  read(SearchInterval);
   write('Tolerance (relative) : ');  read(Tolerance);
   Function_Analyser(f,DerivOrder,Zero_max,SearchInterval,
                     Tolerance);
end.
```

liefert mit den Eingabewerten

`DerivOrder=0, SearchInterval=[0,3.14159264], Tolerance=1E-20`

die Bildschirmausgabe:

```
*******************************************************
*   The following results refer to the function f(x)   *
*******************************************************

The search interval
     [  0.00000000000000000000E+000,  3.14159264000000000000E+000 ]
contains exactly 1 inclusion(s), each with a single and unique zero:
[ -1.00000000000000000000E-057,  1.00000000000000000000E-057 ],  (-1,+1)

Open interval(s) with f(x) values of identical sign are:
(  1.00000000000000000000E-057,  3.14159264000000000000E+000 )     (+)
```

Die Nullstellen–Einschließung läßt sich noch weiter verbessern, wenn man das Programm `nlfz3_2` mit dem Startintervall `[-1E-57,+1E-57]` aufruft; man erhält dann für die einfache Nullstelle die exakte Einschließung `[0,0]`.

Für f ergeben sich daher in $S_1 = [0, 3.14159264]$ die Informationen:

Nullstelleneinschließungen:

```
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ],  (-1,+1)
```

Intervalle mit Funktionswerten gleichen Vorzeichens:

```
(  1.00000000000000000000E-057,  3.14159264000000000000E+000 ),    (+)
```

Wir untersuchen die Funktion f jetzt im Intervall $S_2 = [3.14159267, 6.3]$. Das Programm `nlfz3_2` liefert mit den Eingabewerten

```
DerivOrder=0, SearchInterval=[3.14159267,6.3], Tolerance=1E-20
```

die Bildschirmausgabe:

```
*******************************************************
*   The following results refer to the function f(x)   *
*******************************************************

The search interval
     [  3.14159267000000000000E+000,  6.30000000000000000000E+000 ]
contains exactly 1 inclusion(s), each with a single and unique zero:
[  6.28318530717958647692E+000,  6.28318530717958647693E+000 ],  (-1,+1)

Open interval(s) with f(x) values of identical sign are:
(  3.14159267000000000000E+000,  6.28318530717958647692E+000 )     (-)
(  6.28318530717958647693E+000,  6.30000000000000000000E+000 )     (+)
```

Für die Funktion f ergeben sich damit im Intervall $S_2 = [3.14159267, 6.3]$ die Informationen:

Nullstelleneinschließungen:

```
[  6.28318530717958647692E+000,  6.28318530717958647693E+000 ],  (-1,+1)
```

Intervalle mit Funktionswerten gleichen Vorzeichens:

```
(  3.14159267000000000000E+000,  6.28318530717958647692E+000 )     (-)
(  6.28318530717958647693E+000,  6.30000000000000000000E+000 )     (+)
```

Zur Einschließung der relativen Extrema von f untersuchen wir jetzt f' auf einfache Nullstellen, indem wir die Prozedur `Function_Analyser` im Programm `nlfz3_2` mit `DerivOrder=1` aufrufen. Die Eingabewerte

```
DerivOrder = 1,  SearchInterval = [0,6.3],  Tolerance=1E-10
```

liefern die folgende Bildschirmausgabe:

```
   *****************************************************************
   *   The following results refer to the first derivative f'(x)   *
   *****************************************************************

****************************************************************************
Sorry
The following 2 intervals may contain a zero (not verified unique!)
        [  3.14159265332456796050E+000,  3.14159265352507929851E+000 ]
        [  3.14159265352937200509E+000,  3.14159265376809391967E+000 ]

Remarks:

   The number of this interval(s) can probably be reduced by choosing
   tolerance values different from  2.00000000000000000000E-010  TRY IT !!

   The number of the above interval(s) can probably be reduced by
   choosing small search intervals containing only one of the above
   interval(s),  TRY IT !!

   An interval containing a multiple zero cannot be removed!

   For the following search use only start intervals, which contain none of
   the unremovable interval(s)!

****************************************************************************
```

Wählt man das erste dieser beiden Intervalle als Startintervall, so erhält man mit den Eingabewerten:

```
DerivOrder=0, SearchInterval=[3.14...050,3.14...851], Tolerance=1E-13
```

die Bildschirmausgabe:

```
****************************************************************
*   The following results refer to the first derivative f'(x)   *
****************************************************************

The search interval
     [  3.14159265332456796050E+000,  3.14159265352507929851E+000 ]
contains only negative function values
```

Damit enthält nur das Intervall

```
[  3.14159265352937200509E+000,  3.14159265376809391967E+000 ]
```

von Seite 469 eine mögliche Nullstelle von f'. Da auch mit Hilfe der Transformation (9.5) von Seite 450 nicht nachgewiesen werden kann, daß dieses Intervall einfache Nullstellen enthält, kann man wieder nur vermuten, daß besagtes Intervall ein Einschließung einer mehrfachen Nullstelle ist.

Mit dem gleichen Eingangsparameter `DerivOrder=1` untersuchen wir die Funktion f' daher in den beiden getrennten Intervallen

$$S_3 = [0, 3.1415926535], \qquad S_4 = [3.1415926538, 6.3]$$

und überspringen dabei im vorgegebenen Startintervall $[0, 6.3]$ das Intervall $S_5 = [3.1415926535, 3.1415926538]$, in dem wirklich gesicherte Aussagen über existierende Nullstellen von f' nicht möglich sind.

Mit den Eingabedaten

```
DerivOrder=1, SearchInterval=[0,3.1415926535], Tolerance=1E-20
```

liefert das Programm `nlfz3_2` die Bildschirmausgabe:

```
****************************************************************
*   The following results refer to the first derivative f'(x)   *
****************************************************************

The search interval
     [  0.00000000000000000000E+000,  3.14159265350000000000E+000 ]
contains exactly 1 inclusion(s), each with a single and unique zero:
[  1.04719755119659774615E+000,  1.04719755119659774616E+000 ],  (+1,-1)

The inclusions of the corresponding function values f(x) are:
[  2.59807621135331594028E+000,  2.59807621135331594031E+000 ]

Open interval(s) with f'(x) values of identical sign are:
(  0.00000000000000000000E+000,  1.04719755119659774615E+000 )     (+)
(  1.04719755119659774616E+000,  3.14159265350000000000E+000 )     (-)
```

Damit erhalten wir für f im Intervall $S_3 = [0, 3.1415926535]$ die folgenden Informationen:

Einschließungen relativer Extrema:

```
Maximum-Abszisse enthalten in:
[  1.04719755119659774615E+000,  1.04719755119659774616E+000 ]
Relatives Maximum enthalten in:
[  2.59807621135331594028E+000,  2.59807621135331594031E+000 ]
```

Monotonie-Intervalle

```
(  0.00000000000000000000E+000,  1.04719755119659774615E+000 )  (wachsend)
(  1.04719755119659774616E+000,  3.14159265350000000000E+000 )  (fallend )
```

Ganz entsprechend erhalten wir im Intervall $S_4 = [3.1415926538, 6.3]$ für f die Informationen:

Einschließungen relativer Extrema:

```
Minimum-Abszisse enthalten in:
[  5.23598775598298873076E+000,  5.23598775598298873078E+000 ]
Relatives Minimum enthalten in:
[ -2.59807621135331594032E+000, -2.59807621135331594027E+000 ]
```

Monotonie-Intervalle

```
(  3.14159265380000000000E+000,  5.23598775598298873076E+000 )  (fallend )
(  5.23598775598298873078E+000,  6.30000000000000000000E+000 )  (wachsend)
```

Zur Berechnung der Wendepunkte und Krümmungsintervalle untersuchen wir die Funktion f'' auf einfache Nullstellen. Mit den Eingabedaten

```
DerivOrder = 2,  SearchInterval = [0,6.3],  Tolerance = 1E-20
```

erhält man mit dem Programm `nlfz3_2` die folgende Bildschirmausgabe:

```
*******************************************************************
*   The following results refer to the second derivative f''(x)   *
*******************************************************************

The search interval
     [  0.00000000000000000000E+000,  6.30000000000000000000E+000 ]
contains exactly 5 inclusion(s), each with a single and unique zero:
[ -2.00000000000000000000E-052,  3.00000000000000000000E-052 ],  (+1,-1)
[  1.82347658193697527271E+000,  1.82347658193697527272E+000 ],  (-1,+1)
[  3.14159265358979323845E+000,  3.14159265358979323847E+000 ],  (+1,-1)
[  4.45970872524261120420E+000,  4.45970872524261120422E+000 ],  (-1,+1)
[  6.28318530717958647692E+000,  6.28318530717958647693E+000 ],  (+1,-1)
```

```
The inclusions of the corresponding function values f(x) are:
[ -8.00000000000000000001E-052,  1.20000000000000000001E-051 ]
[  1.45236875482778133193E+000,  1.45236875482778133196E+000 ]
[ -4.00004000000000000001E-020,  4.00004000000000000001E-020 ]
[ -1.45236875482778133197E+000, -1.45236875482778133192E+000 ]
[ -2.11477662360230735774E-020,  1.88536337639769264227E-020 ]

Open interval(s) with f''(x) values of identical sign are:
(  3.00000000000000000000E-052,  1.82347658193697527271E+000 )     (-)
(  1.82347658193697527272E+000,  3.14159265358979323845E+000 )     (+)
(  3.14159265358979323847E+000,  4.45970872524261120420E+000 )     (-)
(  4.45970872524261120422E+000,  6.28318530717958647692E+000 )     (+)
(  6.28318530717958647693E+000,  6.30000000000000000000E+000 )     (-)
```

Aus den obigen Ergebnissen lassen sich die Wendepunkt–Einschließungen und die Krümmungsintervalle direkt ablesen, vergl. Seite 473.

Für die zu untersuchende Funktion $f(x) = 2 \cdot \sin(x) + \sin(2x)$ fassen wir alle im Intervall $[0, 6.3]$ gefundenen Informationen zusammen:

Nullstelleneinschließungen:

```
[  0.00000000000000000000E+000,  0.00000000000000000000E+000 ],  (-1,+1)
[  6.28318530717958647692E+000,  6.28318530717958647693E+000 ],  (-1,+1)
```

Intervalle mit Funktionswerten gleichen Vorzeichens:

```
(  1.00000000000000000000E-057,  3.14159264000000000000E+000 ),    (+)
(  3.14159267000000000000E+000,  6.28318530717958647692E+000 )     (-)
(  6.28318530717958647693E+000,  6.30000000000000000000E+000 )     (+)
```

Einschließungen relativer Extrema:

```
Maximum-Abszisse enthalten in:
[  1.04719755119659774615E+000,  1.04719755119659774616E+000 ]
Relatives Maximum enthalten in:
[  2.59807621135331594028E+000,  2.59807621135331594031E+000 ]

Minimum-Abszisse enthalten in:
[  5.23598775598298873076E+000,  5.23598775598298873078E+000 ]
Relatives Minimum enthalten in:
[ -2.59807621135331594032E+000, -2.59807621135331594027E+000 ]
```

Monotonie-Intervalle

```
(  0.00000000000000000000E+000,  1.04719755119659774615E+000 )  (wachsend)
(  1.04719755119659774616E+000,  3.14159265350000000000E+000 )  (fallend )
(  3.14159265380000000000E+000,  5.23598775598298873076E+000 )  (fallend )
(  5.23598775598298873078E+000,  6.30000000000000000000E+000 )  (wachsend)
```

Wendepunkt-Einschließungen:

```
1. Wendepunkt-Abszisse enthalten in:
[ -2.00000000000000000000E-052,  3.00000000000000000000E-052 ]
1. Wendepunkt-Ordinate enthalten in:
[ -8.00000000000000000001E-052,  1.20000000000000000001E-051 ]

2. Wendepunkt-Abszisse enthalten in:
[  1.82347658193697527271E+000,  1.82347658193697527272E+000 ]
2. Wendepunkt-Ordinate enthalten in:
[  1.45236875482778133193E+000,  1.45236875482778133196E+000 ]

3. Wendepunkt-Abszisse enthalten in:
[  3.14159265358979323845E+000,  3.14159265358979323847E+000 ]
3. Wendepunkt-Ordinate enthalten in:
[ -4.00004000000000000001E-020,  4.00004000000000000001E-020 ]

4. Wendepunkt-Abszisse enthalten in:
[  4.45970872524261120420E+000,  4.45970872524261120422E+000 ]
4. Wendepunkt-Ordinate enthalten in:
[ -1.45236875482778133197E+000, -1.45236875482778133192E+000 ]

5. Wendepunkt-Abszisse enthalten in:
[  6.28318530717958647692E+000,  6.28318530717958647693E+000 ]
5. Wendepunkt-Ordinate enthalten in:
[ -2.11477662360230735774E-020,  1.88536337639769264227E-020 ]
```

Krümmungs-Intervalle

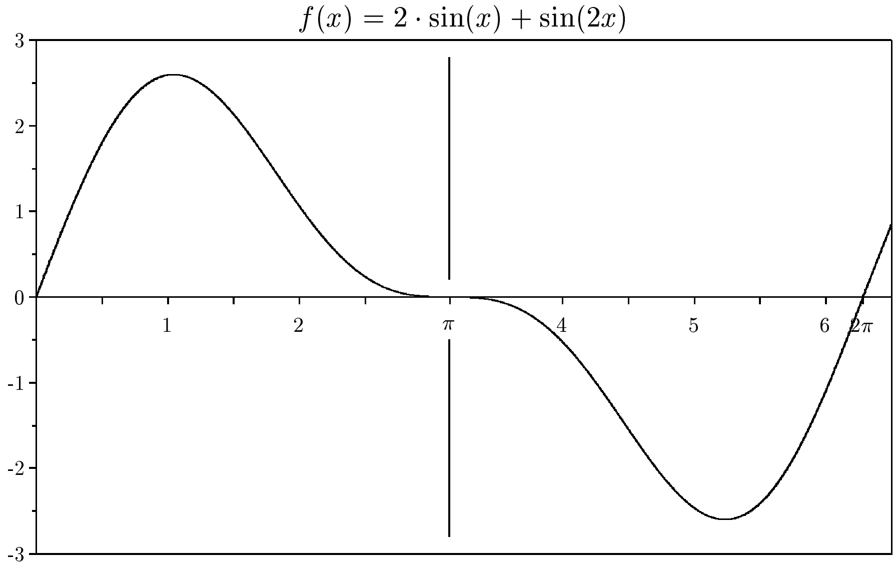
```
(  3.00000000000000000000E-052,  1.82347658193697527271E+000 )  (rechts)
(  1.82347658193697527272E+000,  3.14159265358979323845E+000 )  (links )
(  3.14159265358979323847E+000,  4.45970872524261120420E+000 )  (rechts)
(  4.45970872524261120422E+000,  6.28318530717958647692E+000 )  (links )
(  6.28318530717958647693E+000,  6.30000000000000000000E+000 )  (rechts)
```

Anmerkungen:

1. Die Einschließung $[-2 \cdot 10^{-52}, +3 \cdot 10^{-52}]$ der ersten Wendestelle läßt vermuten, daß diese im Ursprung liegt. Mit Hilfe der Prozedur `Function_Analyser` kann das sogar bewiesen werden, wenn man das Programm `nlfz3_2` mit dem zum Ursprung symmetrischen Intervall $[-3 \cdot 10^{-52}, +3 \cdot 10^{-52}]$ und dem Parameter `DerivOrder=2` startet. Die existierende, einfache Nullstelle wird dann tatsächlich durch das Punktintervall $[0,0]$ exakt eingeschlossen. Dieses Verfahren liefert das Punktintervall $[0,0]$ allerdings nur dann, wenn das Startintervall wirklich symmetrisch zum Ursprung gewählt wird!

2. Beachten Sie bitte, daß die gelieferten Informationen zur Funktion f in den entsprechenden Teilintervallen mathematisch wirklich abgesichert sind, da die Prozedur `Function_Analyser(..)` neben der Intervallrechnung nur selbstverifizierende Algorithmen benutzt.

3. Bei der Diskussion der beiden Funktionen f und f' mußten die sehr schmalen Intervalle S_0 bzw. S_5 ausgeklammert werden, da die Existenz und Eindeutigkeit einer Nullstelle von f bzw. f' in diesen Intervallen nicht nachgewiesen werden konnte. Da jedoch die Intervalle links bzw. rechts von S_0 nur positive bzw. negative Funktionswerte von f enthalten, muß die stetige Funktion f in S_0 selbst mindestens eine Nullstelle besitzen. Der nachfolgende Graph von f läßt dann vermuten, daß dies eine dreifache Nullstelle ist. Die bei der Diskussion nicht beachteten Intervalle S_0, S_5 sind durch die Parallele zur y-Achse symbolisiert.



Daß die aus dem Graph der Funktion f gewonnene Vermutung einer in S_0 eingeschlossenen dreifachen Nullstelle sogar richtig ist, zeigt die äquivalente Umformung des Funktionsterms im offenen Intervall $(0, 2\pi)$:

$$f(x) := 2 \cdot \sin(x) + \sin(2x) \equiv 2 \cdot \frac{\sin^3(x)}{1 - \cos(x)}, \qquad x \in (0, 2\pi); \tag{9.17}$$

Ganz entsprechend liefert die Darstellung der ersten Ableitung

$$f'(x) := 2\,\sin^2(x) \cdot \frac{2\,\cos(x) - 1}{1 - \cos(x)}, \qquad x \in (0, 2\pi),$$

daß f' in $\pi \in S_5$ eine doppelte Nullstelle besitzt. Daß jedoch ohne Kenntnis der Produkt–Terme für f bzw. f' die obigen Aussagen über die Nullstellen von f und f' nicht schlüssig sein müssen, zeigt der folgende Abschnitt.

9.2.3.3 Mehrfache Nullstellen

Wir kommen noch einmal auf das letzte Beispiel der Funktion

$$f(x) = 2 \cdot \sin(x) + \sin(2x)$$

zurück. Bei Durchführung der Kurvendiskussion mit Hilfe der Prozedur `Function_Analyser` hatten wir das Intervall

$$S_0 = [3.14159264, 3.14159267]$$

gefunden, in dem die Existenz und Eindeutigkeit einer Nullstelle von f zunächst nicht nachgewiesen werden konnte. Da die Funktionswerte auf den Randpunkten von S_0 jedoch positiv bzw. negativ sind, besitzt die stetige Funktion f in S_0 damit wenigstens eine Nullstelle. Bezüglich des Intervalls S_0 hatten wir folgende Aussagen gefunden:

1. f hat in S_0 mindestens eine Nullstelle mit Vorzeichenwechsel

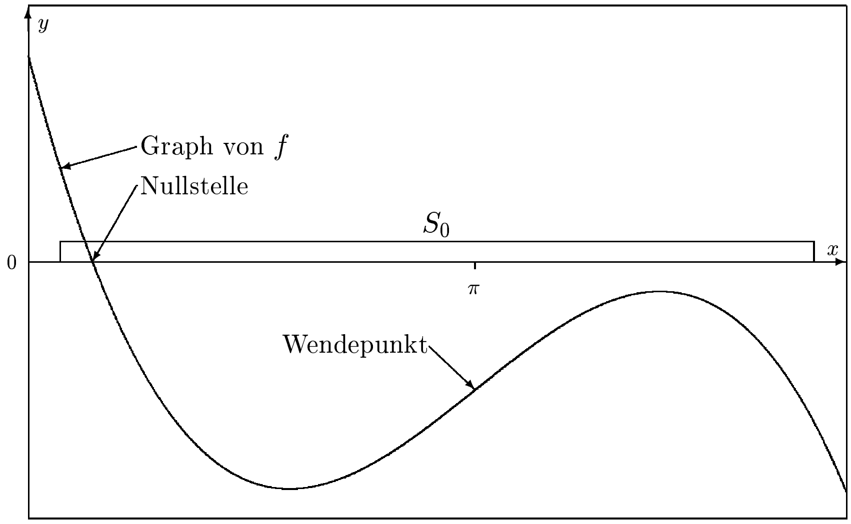
2. f hat in S_0 genau einen Wendepunkt

3. $2 < x < S_0.inf \quad \Longrightarrow \quad f(x) > 0$ ist linksgekrümmt
   $S_0.sup < x < 4 \quad \Longrightarrow \quad f(x) < 0$ ist rechtsgekrümmt

Aus der Produktdarstellung (9.17) des Funktionsterms ergab sich für obige Funktion f, daß die dreifache Nullstelle und der Wendepunkt im Intervall S_0 identisch sind, und dieses Ergebnis wird durch den Funktionsgraphen auf der vorhergehenden Seite auch überzeugend bestätigt.

Mit Hilfe des folgenden Funktionsgraphen soll jedoch gezeigt werden, daß dieses Ergebnis keinesfalls zwingend ist, wenn eine dreimal differenzierbare Funktion in einem, wenn auch noch so kleinen, Intervall die obigen Punkte 1,2 erfüllt und außerhalb dieses Intervalls auch das in Punkt 3 beschriebene Krümmungsverhalten besitzt.

Eine Funktion f erfüllt in S_0 die Bedingungen 1,2,3



Man erkennt sehr deutlich, daß bei einer beliebigen, dreimal differenzierbaren Funktion,welche die Bedingungen 1,2,3 von Seite 475 erfüllt, Nullstelle und Wendepunkt nicht identisch sein müssen, wie es beim letzten Beispiel der Fall gewesen ist.

Liegt also ein i.a. sehr schmales Intervall S_0 vor, in dem die Existenz und Eindeutigkeit einer Nullstelle nicht geklärt werden konnte, so darf man das Verhalten der Funktion in diesem Intervall nicht einfach aus einem plot des Funktionsgraphen ablesen, da schon wegen der viel zu geringen Auflösung der exakte Funktionsverlauf nicht richtig wiedergegeben wird.

Noch ein letzter Hinweis auf mehrfache Nullstellen: Ist S_0 wieder ein Intervall in dem die Existenz und Eindeutigkeit einer p-fachen Nullstelle $x_0 \in S_0$ nicht geklärt werden konnte, so hat bekanntlich $f^{(p-1)}$ an dieser Stelle eine einfache Nullstelle, deren Einschließung I_0 bis zur Ordnung $p = 3$ mit Hilfe der Prozedur `Function_Analyser` problemlos berechnet wird. Gilt dann $I_0 \subseteq S_0$, so wird dadurch der Verdacht auf eine mehrfache Nullstelle weiter erhärtet. Ein mathematischer Beweis für die Existenz einer p-fachen Nullstelle ist damit natürlich **nicht** geführt!

Abschließend noch einige Anmerkungen zu **Algebra–Systemen:** Vor allem bei polynomialen oder auch einfachen Funktionstermen werden die Nullstellen exakt berechnet. Aber schon bei etwas komplizierteren Termen werden existierende Nullstellen in exakter Form nicht mehr gefunden, ohne daß eine Meldung erfolgt, daß Nullstellen existieren könnten.

Wählt man numerische Gleichungslöser, so wird in einem vorgegebenen Intervall i.a. nur eine von mehreren Nullstellen berechnet, wobei trotz hoher Stellenzahl durchaus nicht alle ausgegebenen Ziffern korrekt sein müssen. Bei mehrfachen Nullstellen kann es sogar vorkommen, daß diese überhaupt nicht gefunden werden, ohne daß eine entsprechende Warnung ausgegeben wird.

Im Gegensatz dazu ist die Prozedur `Function_Analyser` ein absolut zuverlässiges Werkzeug:

- Werden in einem vorgegebenem Intervall n Nullstelleneinschließungen mit Existenz- und Eindeutigkeitsaussage gefunden, so ist mathematisch bewiesen, daß die zu untersuchende Funktion in diesem Intervall genau n einfache Nullstellen besitzt, die durch die n i.a. maximalgenauen Intervalle garantiert eingeschlossen werden.

- Werden in einem vorgegebenem Intervall Teilintervalle $[x_i]$ gefunden, für die Existenz- und Eindeutigkeitsaussagen nicht möglich sind und läßt sich deren Anzahl durch mathematische Überlegungen nicht weiter reduzieren, so erhält man gesicherte Aussagen über existierende Nullstellen nur in den Intervallen, die kein $[x_i]$ enthalten. Besitzt die zu untersuchende Funktion mehrfache Nullstellen, so werden diese durch die Intervalle $[x_i]$ eingeschlossen.

Übungen:

Diskutieren Sie mit Hilfe des **PASCAL-XSC** Programms `nlfz3_2`, in dem die Prozedur `Function_Analyser` aufgerufen wird, die folgenden Funktionen und vergleichen Sie Ihre Ergebnisse mit den entsprechenden Werten eines Algebra–Systems:

1. $f(x) = 2 \cdot \sin(x) + \sin(2x) + e^{-10x}/200, \quad x \in [-1, 7];$
2. $f(x) = \sin^2(x), \quad x \in [0, 4];$
3. $f(x) = \sin(2x) - e^{-x^2} + 0.5, \quad x \in [-2, 2];$

Bitte beachten Sie dabei, daß das Programm `nlfz3_2` nach jeder Änderung der zu untersuchenden Funktion neu übersetzt werden muß!

Kapitel 10

Nichtlineare Gleichungssysteme

Wir betrachten die reelle Funktion

$$\boldsymbol{f}(\boldsymbol{x}) \equiv \begin{pmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{pmatrix} \equiv \begin{pmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_n(\boldsymbol{x}) \end{pmatrix} : I\!R^n \to I\!R^n$$

und setzen voraus, daß der Definitionsbereich $D_{\boldsymbol{f}} \subset I\!R^n$ der Funktion $\boldsymbol{f}$ ein n-dimensionales Intervall ist, in dem die nichtlinearen Funktionen $f_i(\boldsymbol{x})$ alle stetig differenzierbar sind.

Die Aufgabe, in $D_{\boldsymbol{f}}$ alle Lösungen des nichtlinearen Gleichungssystems

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0} \tag{10.1}$$

einzuschließen, wird in **PASCAL-XSC** mit Hilfe der Prozedur `Allnlss` erledigt, die vom Hilfsmodul `nlss` exportiert wird.

Existenz- und Eindeutigkeitsaussagen werden nur für solche Nullstellen $\boldsymbol{x}^* \in D_{\boldsymbol{f}}$ geliefert, die folgende Bedingungen erfüllen:

1. $\boldsymbol{x}^*$ ist eine isolierte Nullstelle, die nicht zu dicht neben einer anderen Nullstelle liegen darf.

2. $\boldsymbol{x}^*$ ist keine mehrfache Nullstelle.

Bei geeigneter Wahl der Eingangsparameter von `Allnlss` erhält man jedoch für jede Nullstelle $\boldsymbol{x}^* \in D_{\boldsymbol{f}}$ eine Einschließung.

10.1 Nullstelleneinschließung mit `Allnlss`

Als erstes Beispiel betrachten wir Hansen's Funktion $f : I\!R^n \to I\!R^n$ mit n Variablen:

$$f_i(\boldsymbol{x}) := 0.6 \cdot x_i - 2 + 0.49 \cdot x_i \cdot \sum_{j=1}^{n} x_j^2, \qquad i = 1, 2, \ldots, n \tag{10.2}$$

deren einzige Nullstelle $\boldsymbol{x}^* = (x_i)$ in geschlossener Form darstellbar ist:

$$x_i := \frac{4\sqrt{5}}{7\sqrt{n}} \cdot \sinh\left(\frac{1}{3} \cdot \operatorname{arsinh}\left(\frac{7\sqrt{5n}}{2}\right)\right), \quad i = 1, 2, \ldots, n \tag{10.3}$$

In **PASCAL-XSC** erfolgt die Deklaration der nichtlinearen Funktionen $f_i(\boldsymbol{x})$ nach obigem Beispiel durch:

```
function f(x : GTvector) : GTvector[1..n,0..n];
var sqrsum : GradType[0..n];
    i      : integer;
begin
    sqrsum := 0;
    for i := 1 to n do ;
      sqrsum := sqrsum + sqr(x[i]);
    for i := 1 to n do ;
      f[i] := 0.6*x[i] - 2 + 0.49*x[i]*sqrsum;
end;
```

wobei die Datentypen `GTvector` und `GradType` auf Seite 414 definiert sind und vom Hilfsmodul `igrad` exportiert werden.

Die Darstellung der f_i ist also denkbar einfach und entspricht genau der mathematischen Notation!

Im Ausdruck für die nichtlinearen Funktionen $f_i(\boldsymbol{x})$ sind zulässig die vier Grundoperationen $\{+, -, *, /\}$, wobei ein Operand vom Typ `GradType` sein muß, während der andere Operand vom Typ

```
integer, real, Longreal, rrG, rrGinterval, GradType
```

gewählt werden kann. Als Operand vom Typ `GradType` sind zulässig auch alle Standardfunktionen der Tabelle von Seite 404, womit dann praktisch beliebig verschachtelte Funktionen realisierbar sind.

Die Einschließungen der gesuchten Nullstellen $\boldsymbol{x}^*$ der Funktion $\boldsymbol{f}(\boldsymbol{x})$ liefert die nachfolgende Prozedur `Allnlss`, die vom Hilfsmodul `nlss` exportiert wird.

```
procedure Allnlss( function f(x : GTvector) :
                         GTvector[1..ub(x,1),0..ub(x,2)];
  Start : Livector;                  { Startintervall-Vektor }
  eps   : real;                      { Genauigkeitsparameter }
  var Solumatrix    : Limatrix;  { liefert N.ST.-Einschlg. }
  var Infovector    : bvector;   { Existenz u. Eindeutgkt. }
  var NumberOfSolus : integer;   { Zahl der Einschliessgn. }
  var Err           : integer; ) { Err=0:      Kein Fehler }
```

Eingangsgrößen:

- `f` : Funktion, deren Nullstellen einzuschließen sind.
- `Start` : Alle Null-Stn. aus diesem Intervallvektor sind einzuschließen.
- `eps` : Relative Fehlerschranke der Nullstelleneinschließungen.

Ausgangsgrößen:

- `Solumatrix` : Speichert alle gefundenen Einschliessungen.
- `InfoVector` : Liefert Existenz- u. Eindeutigkeitsaussagen.
- `NumberOfSolus` : Liefert Anzahl der gefundenen Einschließungen.
- `Err` : Fehler-Kode; `Err=0`: Keine Fehler aufgetreten.

Anmerkungen:

1. Der Intervallvektor `Start` ist das n-dimensionale Intervall, in dem alle Nullstellen der stetig differenzierbaren Funktion $\boldsymbol{f} : I\!R^n \to I\!R^n$ einzuschließen sind. Die Eingabe von `Start` erfolgt in der Form:

   $$[a_1, b_1] \;\; [a_2, b_2] \;\; \ldots \;\; [a_n, b_n] \quad < \texttt{enter} > \qquad (\, a_i, b_i : \texttt{Longreal}\,)$$

   Es ist zu empfehlen, in dem n-dimensionale Intervall `Start` zunächst möglichst viele n-dimensionale Teilintervalle T_i zu finden, für die nachgewiesen werden kann, daß sie keine Nullstellen von $\boldsymbol{f}$ enthalten. Der Nachweis ist dabei für ein T_i dann geführt, wenn die Ausgangsgröße `NumberOfSolus` den Wert 0 liefert. $R := \texttt{Start}\backslash(T_1 \cup T_2 \cup \ldots)$ enthält dann alle gesuchten Nullstellen und sollte so klein wie möglich ausfallen. Es kann notwendig sein, auch R noch weiter zu unterteilen!

2. Die berechneten Einschließungen werden solange verbessert, bis die relativen Durchmesser der Intervalle kleiner `eps` geworden sind. Ein zu klein gewählter `eps`–Wert wird intern auf 10^{-20} gesetzt. Bei sehr kleinen (`eps` $\approx 10^{-19}$) Werten kann die Laufzeit extrem lang sein. In diesen Fällen sollte man z.B. mit `eps` $= 10^{-6}$ beginnen und diesen Wert dann nach Bedarf verkleinern.

3. In der Intervallmatrix `Solumatrix` werden alle gefundenen Einschließungen zeilenweise gespeichert, d.h. jede Zeile von `Solumatrix` enthält die n Intervall–Komponenten einer einzelnen Einschließung. Ist die Anzahl der gefundenen Einschließungen größer als die Zeilenzahl von `Solumatrix`, so erfolgt eine Fehlermeldung und `ub(Solumatrix,1)` ist im aufrufenden Programm entsprechend zu vergrößern.

4. `InfoVector[i]` liefert zur gefundenen Einschließung `Solumatrix[i] = S[i]` folgende Informationen:

   - `InfoVector[i] = true  :` `S[i]` encloses a locally unique zero!
   - `InfoVector[i] = false :` `S[i]` may contain a zero (not verified unique!)

   Im ersten Fall enthält `S[i]` $= [\boldsymbol{x}_i]$ genau eine einfache Nullstelle $\boldsymbol{x}_i$ von $\boldsymbol{f}(\boldsymbol{x})$. Im zweiten Fall `(false)` kann `S[i]` mehrere[1] einfache oder auch mehrfache Nullstellen enthalten. Es kann aber auch sein, daß `S[i]` keine Nullstelle enthält. Der erste Indexbereich von `Solumatrix` muß mit dem Indexbereich von `InfoVector` identisch sein.

5. Die Ausgangsgröße `NumberOfSolus` liefert die Zahl aller berechneten Einschließungen. Wenn `InfoVector[i] = true` für alle i erfüllt ist, liefert `NumberOfSolus` bewiesenermaßen die Anzahl aller Nullstellen von $\boldsymbol{f}$ in `Sart`. Erhält man z.B. `NumberOfSolus=0`, so hat man den Beweis geführt, daß die Funktion $\boldsymbol{f}$ im n-dimensionalen Eingangs–Intervall `Start` keine Nullstellen besitzt.

6. Beachten Sie bitte, daß die im letzten Punkt beschriebenen Existenz- und Eindeutigkeitsaussagen nicht geliefert werden, wenn die Funktion $\boldsymbol{f}$ in `Start` mehrfache oder unendlich viele Nullstellen besitzt.

[1] Es können auch unendlich viele Nullstellen eingeschlossen werden.

10.2 Programmbeispiele

10.2.1 Hansen's Funktion

Wir betrachten die in (10.2) definierte Funktion von Hansen. Für $n = 7$ sollen im 7-dimensionalen Intervall mit den jeweils gleichen Komponenten $[-1, +1]$ alle Nullstellen von $\boldsymbol{f} = (f_i)$ berechnet werden. Mit dem folgenden Programm `nlss_ex`

```
program nlss_ex;

use
  type_def, { Type Definitionen                  }
  li_ari,   { Interval arithmetic                }
  lmvi_ari, { Interval matrix/vector arithmetic  }
  igrad,    { Gradient differentiation arithmetic }
  nlss;     { Nonlinear System Solver            }

const
  m    = 20;    { Maximum number of solutions to be computed }
  name = 'Hansen''s Function';

var
  n : integer;    { Number of variables in Hansen's function }

{-------------------------------------------------------------}
{ Function to prompt for the desired value of 'n'.      It is }
{ called only once within the declaration part of the first   }
{ dynamic variable. This enables full dynamics within the     }
{ main program.                                               }
{-------------------------------------------------------------}
function set_fDim : integer;
begin
  write('Desired dimension    : ');  read(n);
  set_fDim := n;
end;

var
  SearchInterval   : Livector[1..set_fDim]; { using a call of }
  Tolerance        : real;              { function 'set_fDim' }
  Solu             : Limatrix[1..m,1..n];
```

```
  Unique           : bvector[1..m];
  NumberOfSolus, i : integer;
  ErrCode          : integer;

{--------------------------------------------------------------}
{ Definition of Hansen's function in 'n' variables.            }
{--------------------------------------------------------------}
function f (x: GTvector) : GTvector[1..n,0..n];
var
  sqrsum : GradType[0..n];
  i      : integer;
begin
   sqrsum := 0;
   for i := 1 to n do
     sqrsum := sqrsum + sqr(x[i]);
   for i := 1 to n do
      f[i] := 0.6*x[i] - 2 + 0.49 * x[i] * sqrsum;
end;

{ Main program }
{--------------}
begin
   writeln;
   write('Computing all solutions for ', Name, ' in ', n);
   writeln(' variables');   writeln;
   write('Search interval      : ');  read(SearchInterval);
   write('Tolerance (relative) : ');  read(Tolerance);
   writeln;
   AllNLSS(f, SearchInterval, Tolerance, Solu, Unique,
                                    NumberOfSolus, ErrCode);
   for i:=1 to NumberOfSolus do
   begin
      write(Solu[i]);
      if unique[i] then
        writeln('encloses a locally unique zero!')
                 else
      writeln('may contain a zero (not verified unique)!')
   end;
   if ErrCode <> 0 then writeln(AllNLSSErrMsg(ErrCode));
   writeln;
```

```
   writeln(NumberOfSolus:1, ' interval enclosure(s)'); writeln;
   if (NumberOfSolus = 1) and (unique[1]) then
   begin
      write('We have validated that there is a globally ');
      writeln('unique zero!');
   end;
end.
```

erhält man das Ablaufprotokoll:

```
Desired dimension     : 7  <enter>

Computing all solutions for Hansen's function in 7 variables

Search interval      :
[-1,1] [-1,1] [-1,1] [-1,1] [-1,1] [-1,1] [-1,1]     <enter>

Tolerance (relative) : 1e-20                         <enter>

[ 7.658145069986342420800E-001, 7.658145069986342420810E-001 ]
[ 7.658145069986342420800E-001, 7.658145069986342420810E-001 ]
[ 7.658145069986342420800E-001, 7.658145069986342420810E-001 ]
[ 7.658145069986342420800E-001, 7.658145069986342420810E-001 ]
[ 7.658145069986342420800E-001, 7.658145069986342420810E-001 ]
[ 7.658145069986342420800E-001, 7.658145069986342420810E-001 ]
[ 7.658145069986342420800E-001, 7.658145069986342420810E-001 ]
encloses a locally unique zero!

1 interval enclosure(s)

We have validated that there is a globally unique zero!
```

Anmerkungen:

- Beachten Sie, daß die sieben Einschließungen für die nach (10.3) von i unabhängigen Nullstellenkomponenten x_i identisch sind und maximalgenau berechnet wurden.
- Beweisen Sie mit Hilfe des Programms `nlss_ex`, daß außerhalb des obigen 7−dimenisonalen Startintervalls keine weiteren Nullstellen der Hansen'schen Funktion existieren.

10.2.2 Einschließung stationärer Punkte

Gegeben sei eine Funktion $f : I\!R^n \to I\!R$ mit stetigen partiellen Ableitungen in einem n-dimensionalen Intervall $D_f \subset I\!R^n$. Wir betrachten das System der n nichtlinearer Gleichungen in n Variablen x_i, mit: $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$

$$\begin{pmatrix} \dfrac{\partial f(\boldsymbol{x})}{\partial x_1} = 0 \\ \dfrac{\partial f(\boldsymbol{x})}{\partial x_2} = 0 \\ \vdots \\ \dfrac{\partial f(\boldsymbol{x})}{\partial x_n} = 0 \end{pmatrix} \iff \nabla f(\boldsymbol{x}) = 0 \tag{10.4}$$

Ein $\boldsymbol{x}^* \in D_f$ heißt stationärer Punkt von f, wenn $\boldsymbol{x}^*$ Lösung des nichtlinearen Systems (10.4) ist.

Die Aufgabenstellung lautet, garantierte Einschließungen aller $\boldsymbol{x}^* \in D_f$ zu berechnen. Eine Lösungsmethode besteht darin, zu gegebener Funktion f das System (10.4) aufzustellen und danach die stationären Punkte mit Hilfe der Prozedur `Allnlss` von Seite 481 einzuschließen. Daß die partiellen Ableitungen dabei von Hand oder mit einem Computeralgebrapaket vorher zu berechnen sind, ist natürlich ein Nachteil dieser Methode, allerdings hat man dabei die Möglichkeit, durch geeignete äquivalente Umformungen das System (10.4) weiter zu vereinfachen.

Da wir die Berechnung der stationären Punkte von f auf die Lösung eines nichtlinearen Gleichungssystem zurückführen[2], gelten alle Hinweise von Seite 481. Existenz- und Eindeutigkeitsaussagen werden also nicht geliefert, wenn die $\boldsymbol{x}^*$ mehrfache Nullstellen von (10.4) sind oder wenn die $\boldsymbol{x}^*$ zu dicht nebeneinander liegen oder ein Kontinuum bilden. Beachten Sie jedoch, daß solche $\boldsymbol{x}^*$ vom Algorithmus stets eingeschlossen werden, wenn man die zu untersuchenden Teilintervalle nicht zu groß und den Toleranzwert nicht zu klein wählt.

Beachten Sie bitte weiter, daß dieses Verfahren alle $\boldsymbol{x}^* \in D_f$ garantiert einschließt. Es gehen also keine stationären Punkte von f verloren und es werden auch keine Punkte als vermeintliche stationäre Punkte erfunden, die durch Scheinkonvergenz auf einem Rechner entstehen können. Auf Seite 424 haben wir einen Algorithmus vorgestellt, mit dem wir stationäre Punkte lediglich approximieren konnten, ohne Existenz- und Eindeutigkeitsaussagen

[2] Jede partielle Ableitung $\partial f / \partial x_i$ ist eine eigenständige, nichtlineare Funktion $f_i(\boldsymbol{x})$.

machen zu können. Das folgende Beispiel zeigt, daß erst die Einschließungsmethoden von **PASCAL-XSC** gesicherte Aussagen bzgl. der stationären Punkte liefern und daß vermeintliche stationäre Punkte in dem Beispiel von Seite 425 in Wirklichkeit keine stationären Punkte sind!

10.2.2.1 Stationäre Punkte der Rosenbrock-Funktion

Die verallgemeinerte Rosenbrock–Funktion lautet mit $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$:

$$f(\boldsymbol{x}) = \sum_{i=2}^{n} 100 \cdot (x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2 \tag{10.5}$$

Für $n = 7$ sollen im 7-dimensionalen Intervall

$$I := (\,[-2, +2], [0,2], [0,2], [0,2], [0,2], [0,2], [0,2]\,)$$

alle stationären Punkte $\boldsymbol{x}^* \in I$ von $f : I\!R^7 \to I\!R$ eingeschlossen werden. Die n partiellen Ableitungen von f sind gegeben durch:

$$\begin{aligned}
\frac{\partial f(\boldsymbol{x})}{\partial x_1} &= -400 \cdot x_1 \cdot (x_2 - x_1^2) - 2 \cdot (1 - x_1) \\
\frac{\partial f(\boldsymbol{x})}{\partial x_i} &= 200 \cdot (x_i - x_{i-1}^2) - 400 \cdot x_i \cdot (x_{i+1} - x_i^2) - 2 \cdot (1 - x_i) \\
&\qquad\qquad\qquad\qquad i = 2, 3, \dots, n-1 \\
\frac{\partial f(\boldsymbol{x})}{\partial x_n} &= 200 \cdot (x_n - x_{n-1}^2)
\end{aligned}$$

Das zu lösende nichtlineare Gleichungssystem lautet dann nach geeigneten Umformungen:

$$\begin{aligned}
f_1(\boldsymbol{x}) &:= 200 \cdot x_1 \cdot (x_2 - x_1^2) - x_1 + 1 = 0 \\
f_i(\boldsymbol{x}) &:= -100 \cdot (x_i - x_{i-1}^2) + 200 \cdot x_i \cdot (x_{i+1} - x_i^2) - x_i + 1 = 0 \\
&\qquad\qquad\qquad\qquad i = 2, 3, \dots, n-1 \\
f_n(\boldsymbol{x}) &:= x_n - x_{n-1}^2 = 0
\end{aligned}$$

Für die Prozedur `Allnlss` von Seite 480 ist damit die Funktion[3] `f` vom Typ `GTvector` wie folgt zu deklarieren:

[3] Mit $\mathtt{f} = (f_1, f_2, \dots, f_7)^T$ ist also nicht die Rosenbrock-Funktion f gemeint!

```
function f (x: GTvector) : GTvector[1..n,0..n];
var i : integer;
begin
   f[1] := 200*x[1]*(x[2]-sqr(x[1])) - x[1] + 1;
   For i := 2 to n-1 do
      f[i] := -100*(x[i]-sqr(x[i-1])) +
               200*x[i]*(x[i+1]-sqr(x[i])) - x[i] + 1;
   f[n] := x[n] - sqr(x[i-1]);
end;
```

Das nachfolgende Programm `nlss_ex1`

```
program nlss_ex1;

use
  type_def,  { Type Definitionen                    }
  li_ari,    { Interval arithmetic                  }
  lmvi_ari,  { Interval matrix/vector arithmetic    }
  igrad,     { Gradient differentiation arithmetic }
  nlss;      { Nonlinear System Solver              }

const
  m    = 30;     { Maximum number of solutions to be computed }
  name = 'stationary points of Rosenbrock''s function';

var
  n : integer;     { Number of variables in Hansen's function }

{-----------------------------------------------------------}
{ Function to prompt for the desired value of 'n'.    It is }
{ called only once within the declaration part of the first }
{ dynamic variable. This enables full dynamics within the   }
{ main program.                                             }
{-----------------------------------------------------------}
function set_fDim : integer;
begin
  write('Desired dimension    : ');  read(n);
  set_fDim := n;
end;
```

```
var
  SearchInterval   : Livector[1..set_fDim]; { using a call of }
  Tolerance        : real;              { function 'set_fDim' }
  Solu             : Limatrix[1..m,1..n];
  Unique           : bvector[1..m];
  NumberOfSolus, i : integer;
  ErrCode          : integer;

{-------------------------------------------------------------}
{ The n partial derivatives in 'n' variables of Rosenbrock's  }
{ function after some divisions.                              }
{-------------------------------------------------------------}
function f (x: GTvector) : GTvector[1..n,0..n];
var i : integer;
begin
   f[1] := 200*x[1]*(x[2]-sqr(x[1])) - x[1] + 1;
   For i := 2 to n-1 do
      f[i] := -100*(x[i]-sqr(x[i-1])) +
               200*x[i]*(x[i+1]-sqr(x[i])) - x[i] + 1;
   f[n] := x[n] - sqr(x[i-1]);
end;

{ Main program }
{--------------}
begin
   writeln;
   write('Computing all ', Name);
   writeln(' in ',n,' variables');   writeln;
   write('Search interval      : ');  read(SearchInterval);
   write('Tolerance (relative) : ');  read(Tolerance);
   writeln;
   AllNLSS(f, SearchInterval, Tolerance, Solu, Unique,
                                    NumberOfSolus, ErrCode);
   for i:=1 to NumberOfSolus do
   begin
      write(Solu[i]);
      if unique[i] then
        writeln('encloses a locally unique zero!')
                 else
      writeln('may contain a zero (not verified unique)!')
```

```
   end;
   if ErrCode <> 0 then writeln(AllNLSSErrMsg(ErrCode));
   writeln;
   writeln(NumberOfSolus:1, ' interval enclosure(s)'); writeln;
   if (NumberOfSolus = 1) and (unique[1]) then
   begin
      write('We have validated that there is a globally ');
      writeln('unique zero!');
   end;
end.
```

liefert das Ausgabe–Protokoll:

```
Desired dimension     : 7  <enter>

Computing all stationary points of Rosenbrock's function
in 7 variables

Search interval      :
[0.9,1.1] [0.9,1.1] [0.9,1.1] [0.9,1.1] [0.9,1.1]
[0.9,1.1] [0.9,1.1]                        <enter>

Tolerance (relative) : 1e-16               <enter>

[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
encloses a locally unique zero!

1 interval enclosure(s)

We have validated that there is a globally unique zero!
```

Damit ist die Einschießung des stationären Punktes bestätigt, der auf Seite 427 mit dem Startvektor `(3,3,3,3,3,3,3)` erfolgreich approximiert wurde. Beachten Sie bitte, daß das Programm mit dem zu kleinen Wert `Tolerance = 1e-20` nicht erfolgreich beendet wird.

Das gleiche Programm `nlss_ex1` liefert das folgende Ausgabe–Protokoll:

```
Desired dimension     : 7  <enter>

Computing all stationary points of Rosenbrock's function
in 7 variables

Search interval      :
[-1,-0.9] [0.9,1] [0.9,1] [0.9,1] [0.9,1]
[0.9,1] [0.9,1]                   <enter>

Tolerance (relative) : 1e-16      <enter>

[ -9.932633329552317079 58E-001, -9.932633329552317079 57E-001 ]
[  9.966059603809815892 31E-001,  9.966059603809815892 32E-001 ]
[  9.982404539056858054 75E-001,  9.982404539056858054 76E-001 ]
[  9.989881190088627698 93E-001,  9.989881190088627698 94E-001 ]
[  9.992255232052923335 92E-001,  9.992255232052923335 93E-001 ]
[  9.990723852310858223 86E-001,  9.990723852310858223 87E-001 ]
[  9.984516462248902078 46E-001,  9.984516462248902078 47E-001 ]
encloses a locally unique zero!

1 interval enclosure(s)

We have validated that there is a globally unique zero!
```

Damit haben wir nachgewiesen, daß die ab Seite 427 mit dem Startvektor `(-2,2,2,2,2,2,2)` gefundene Einschließung nicht die Einschließung eines stationären Punktes der Rosenbrock-Funktion f sein kann.

Wir zeigen jetzt noch, daß die ab Seite 427 mit dem dortigen Startvektor `(3,3,3,3,4,5,6)` gefundene Einschließung nicht die Einschließung eines stationären Punktes der Rosenbrock-Funktion f sein kann. Dazu liefert das Programm `nlss_ex1` das entsprechende Ausgabe–Protokoll:

```
Desired dimension     : 7  <enter>

Computing all stationary points of Rosenbrock's function
in 7 variables

Search interval      :
[1,1.5] [1,1.5] [1,1.5] [1,1.5] [1,1.5]
```

```
[1,1.5] [1,1.5]                         <enter>

Tolerance (relative) : 1e-16            <enter>

[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
[ 9.99999999999999999999E-001, 1.00000000000000000001E+000 ]
encloses a locally unique zero!

1 interval enclosure(s)

We have validated that there is a globally unique zero!
```

Die obige Existenz- und Eindeutigkeitsaussage schließt damit die Existenz eines weiteren stationären Punktes aus, so daß die mit dem Startvektor `(3,3,3,3,4,5,6)` auf Seite 427 gefundene Einschließung keinen stationären Punkt der Rosenbrock–Funktion enthalten kann.

Das letzte Beispiel zeigt deutlich, daß nur die Verifikationsmethoden in Verbindung mit der in **PASCAL-XSC** implementierten Intervallrechnung garantierte Einschließungen der stationären Punkte von f liefern.

Wir haben bis jetzt gezeigt, daß in den beiden Bereichen

$$\begin{aligned} I_1 &:= ([0.9,1.5],[0.9,1.5],[0.9,1.5],[0.9,1.5],[0.9,1.5],[0.9,1.5],[0.9,1.5]) \\ I_2 &:= ([-1,-0.9],[0.9,1],[0.9,1],[0.9,1],[0.9,1],[0.9,1],[0.9,1]) \end{aligned}$$

jeweils genau ein stationärer Punkt der Rosenbrock–Funktion f mit sieben Variablen liegt. Beweisen Sie, daß in

$$I := ([-2,+2],[0,2],[0,2],[0,2],[0,2],[0,2],[0,2])$$

genau zwei stationäre Punkte von f liegen, indem Sie zeigen, daß in der Restmenge $R := I \backslash (I_1 \cup I_2)$ keine weiteren stationären Punkte existieren. Skizzieren Sie dazu die Restmenge R zunächst im zweidimensionalen Fall und übertragen Sie dann die Vorgehensweise auf den 7-dimensionalen Fall. In der Umgebung der beiden schon gefundenen stationären Punkte ist R hinreichend fein zu unterteilen, damit vom Programm die Meldung geliefert werden kann, daß im entsprechenden Teilintervall keine Nullstelle existiert.

10.2.3 $e^{x_1^2+x_2^2} - 3 = 0; \quad x_1 + x_2 - \sin[3 \cdot (x_1 - 2x_2)] = 0$

Wir betrachten das nichtlineare Gleichungssystem aus 2 Gleichungen mit 2 Unbekannten:

$$\begin{aligned} f_1(x_1, x_2) &:= e^{x_1^2+x_2^2} - 3 &= 0; \\ f_2(x_1, x_2) &:= x_1 + x_2 - \sin[3 \cdot (x_1 - 2x_2)] &= 0 \end{aligned}$$

und wollen beweisen, daß obiges System in der x_1, x_2–Ebene genau zwei Nullstellen besitzt. Zunächst erkennt man aus $f_1(x_1, x_2) = 0$, daß alle Nullstellen auf einem Kreis um den Ursprung mit dem Radius $r = \sqrt{\ln 3} = 1.048\ldots$ liegt, so daß die Nullstellensuche eingeschränkt werden kann auf den Bereich: $-1.1 \leq x_1, x_2 \leq +1.1$; Wegen der Beziehungen $f_1(-x_1, -x_2) = f_1(x_1, x_2)$ und $f_2(-x_1, -x_2) = -f_1(x_1, x_2)$ kann die Nullstellensuche weiter reduziert werden auf den Quader in der rechten Halbebene:

$$0 \leq x_1 \leq +1.1; \qquad -1.1 \leq x_2 \leq +1.1 \tag{10.6}$$

Die restlichen Nullstellen sind dann die punktsymmetrischen Spiegelbilder aller Nullstellen dieses Quaders.

Mit Hilfe des nachfolgenden Programms `nlss_ex2`

```
{---------------------------------------------------------------}
{ This program uses module 'nlss' to compute the solution of  }
{ a system of nonlinear equations given by:                    }
{ f1(x1,x2) = exp(x1^2+x2^2) - 3        = 0                    }
{ f2(x1,x2) = x1 + x2 - sin[3(x1-2*x2)] = 0                    }
{ Two starting intervals and a tolerance must be entered.     }
{---------------------------------------------------------------}
program nlss_ex2;

use
  type_def,  { Type Definitionen                   }
  li_ari,    { Interval arithmetic                 }
  lmvi_ari,  { Interval matrix/vector arithmetic   }
  igrad,     { Gradient differentiation arithmetic }
  nlss;      { Nonlinear System Solver             }

const
   n    = 20;  { Maximum number of solutions to be computed }
   name = 'two nonlinear equations in two variables';
```

```
   fDim =  2;  { Two variables }

var
  SearchInterval   : Livector[1..fDim];
  Tolerance        : real;
  Solu             : Limatrix[1..n,1..fDim];
  Unique           : bvector[1..n];
  NumberOfSolus, i : integer;
  ErrCode          : integer;

{-------------------------------------------------------------}
{ Definition of the two equatons:                             }
{-------------------------------------------------------------}
function f (x: GTvector) : GTvector[1..fDim,0..fDim];
begin
   f[1] := exp( sqr(x[1])+sqr(x[2]) ) - 3;
   f[2] := x[1] + x[2] - sin( 3*(x[1]-2*x[2]) );
end;

{ Main program }
{--------------}
begin
  writeln;
  writeln('Computing all solutions for ', Name);
  writeln;
  write('Search interval      : ');  read(SearchInterval);
  write('Tolerance (relative) : ');  read(Tolerance);
  writeln;
  AllNLSS(f, SearchInterval, Tolerance, Solu, Unique,
                                NumberOfSolus, ErrCode);
  for i:=1 to NumberOfSolus do
  begin
    write(Solu[i]);
    if unique[i] then
      writeln('encloses a locally unique zero!')
    else
      writeln('may contain a zero (not verified unique)!')
  end;
  if ErrCode<>0 then writeln(AllNLSSErrMsg(ErrCode)); writeln;
  writeln(NumberOfSolus:1, ' interval enclosure(s)'); writeln;
```

```
  if (NumberOfSolus = 1) and (unique[1]) then
  begin
     write('We have validated that there is a globally ');
     writeln('unique zero!');
  end;
end.
```

zeigt man zunächst, daß in den beiden Bereichen

$$\begin{aligned} I_1 &:= ([0,+1.1],[0,+1.1]) \\ I_2 &:= ([0,+1.1],[-0.5,0]) \end{aligned}$$

keine Nullstellen existieren.

Für den Restbereich $I_3 := ([0,+1.1],[-1.1,-0.5])$ liefert dann das obige Programm das folgende Ablauf–Protokoll:

```
Search interval      : [0,+1.1] [-1.1,-0.5]  <enter>

Tolerance (relative) : 1e-20                 <enter>

[  8.08900224658822254366E-001,  8.08900224658822254367E-001 ]
[ -6.66552860030632961873E-001, -6.66552860030632961872E-001 ]
encloses a locally unique zero!

1 interval enclosure(s)

We have validated that there is a globally unique zero!
```

Mit diesem Ergebnis existiert also in der rechten Halbebene genau eine Nullstelle des nichtlinearen Gleichungssystems, und die zweite Nullstelle entsteht durch Spiegelung am Ursprung:

```
[ -8.08900224658822254367E-001, -8.08900224658822254366E-001 ]
[ +6.66552860030632961872E-001, +6.66552860030632961873E-001 ]
```

Damit ist bewiesen, daß das gegebene nichtlineare Gleichungssystem in der ganzen (x_1, x_2)–Ebene genau zwei Nullstellen besitzt, die durch die obigen zweidimensionalen Intervalle garantiert eingeschlossen werden.

Kapitel 11

Globale Optimierung

Gegeben sei eine reelle, zweimal stetig differenzierbare Funktion

$$f : D \to I\!R, \qquad [x] \subseteq D \subseteq I\!R^n, \quad n = 1, 2, \ldots$$
$$[x] := ([x]_1, \ldots, [x]_n) \qquad \text{mit} \quad [x]_i := [\,\underline{x}_i, \overline{x}_i\,], \quad \underline{x}_i, \overline{x}_i \in I\!R;$$

$[x]$ ist ein n-dimensionaler Quader, der auch als Intervallvektor bezeichnet wird[1].

Gesucht sind sowohl das *globale Minimum* f^* aller Funktionswerte $f(x)$ mit $x = (x_1, x_2, \ldots, x_n) \in [x]$:

$$f^* := \min_{x \in [x]} f(x)$$

als auch alle *globalen Minimumstellen* $x^* \in [x]$, mit $f(x^*) = f^*$. Da f auf der kompakten Menge $[x]$ stetig ist, existiert das Minimum f^* und wird an wenigstens einer Stelle $x^* \in [x]$ angenommen. f^* ist also eindeutig bestimmt, während die Stellen (Minimizer) x^* in $[x]$ sogar kontinuierlich verteilt sein können. Bei der Funktion $f(x_1, x_2) := \sin^2(x_1^2 + x_2^2)$ wird $f^* = 0$ in der x_1, x_2-Ebene u.a. auf einem Kreis um den Ursprung mit dem Radius $r = \pi$ angenommen.

Ein Innenpunkt $x_l \in [x]$ heißt *local minimizer*, wenn es eine Umgebung $U(x_l) \subset [x]$ gibt, in der für $x \neq x_l$ gilt: $\quad f(x) > f(x_l)$. Bei unseren stetig differenzierbaren Funktionen ist x_l damit ein stationärer Punkt, und wir bezeichnen $f(x_l)$ als *lokales* Minimum.

In den folgenden Abschnitten wird gezeigt, wie man in den Fällen $n = 1$ und $n > 1$ das globale Minimum und die globalen Minimizer einschließen kann.

[1] Auf dem Rechner sind die Intervallgrenzen $\underline{x}_i, \overline{x}_i$ Rasterzahlen des Longreal–Formats.

11.1 Reelle eindimensionale Funktionen

Gegeben sei eine reelle, zweimal stetig differenzierbare Funktion

$$f : D \to I\!R, \qquad [x] \subseteq D \subseteq I\!R, \quad [x] := [\,\underline{x}, \overline{x}\,], \quad \underline{x}, \overline{x} \in I\!R,$$

Gesucht sind sowohl die Einschließung des globalen Minimums f^* aller Funktionswerte $f(x)$ mit $x \in [x]$:

$$f^* := \min_{x \in [x]} f(x)$$

als auch alle Einschließungen der globalen Minimumstellen $x^* \in [x]$, mit $f(x^*) = f^*$. Soll z.B. in einem vorgegebenen Intervall $[x]$ das globale Minimum f^* der Funktion

$$f(x) = (x + \sin x) \cdot e^{-x^2}$$

eingeschlossen werden, so ist im Quelltext eines entsprechenden **PASCAL-XSC** Programms die Funktion `f` wie folgt zu deklarieren:

```
function f(x : DerivType) : DerivType;
begin
    f := (x+sin(x)) * exp(-sqr(x));
end;
```

wobei der Datentyp `DerivType` auf Seite 350 definiert und vom Hilfsmodul `ddf_rrg` exportiert wird. Eine Tabelle der zur Verfügung stehenden Funktionen vom Typ `DerivType` findet man auf Seite 351.

Die Einschließungen von f^* und der möglicherweise verschiedenen x^* und deren Bildschirmausgabe liefert die Prozedur `Min_g1`, die vom Hilfsmodul `gop1` exportiert wird:

```
procedure Min_g1( function f(x : DerivType) : DerivType
  n              : integer;  { Maximalzahl der Minimizer }
  SearchInterval : Linterval;           { Startintervall }
  Tolerance      : real );   { Fehlerschr. der Minimizer }
```

Eingangsparameter:

1. `function f(x : DerivType) : DerivType;` ist die Funktion, deren globales Minimum und deren globale Minimizer einzuschließen sind.

2. `n : integer` ist die Maximalzahl möglicher Minimizer. Dieser Wert sollte im aufrufenden Programm als Konstante vereinbart werden.

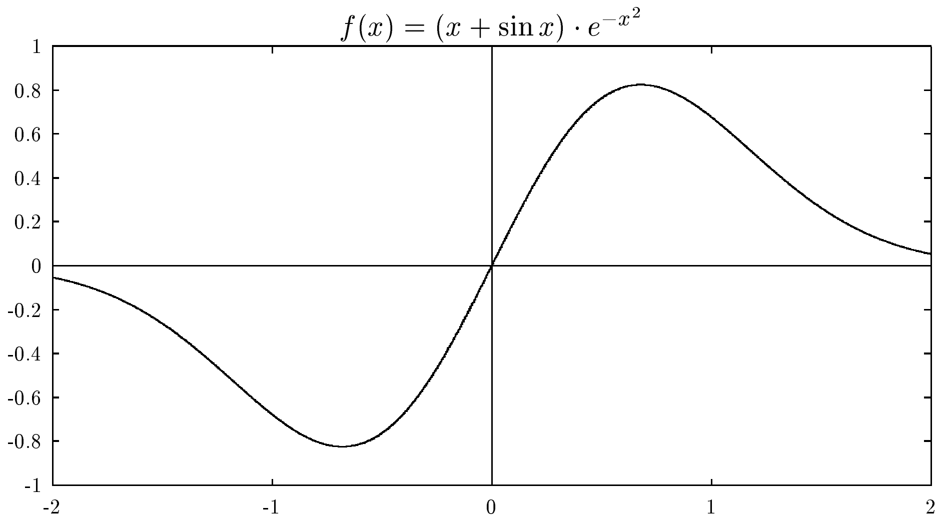
3. `SearchInterval : Linterval` ist das Intervall, in dem die Funktion auf Minimizer zu untersuchen ist.

4. `Tolerance : real` ist die relative Fehlerschranke, mit der die gesuchten Minimizer einzuschließen sind. Werte, die der Anwender kleiner als $2 \cdot 10^{-20}$ wählt, werden intern automatisch auf diese Schranke gesetzt.

11.1.1 Programmbeispiele

11.1.1.1 $f(x) = (x + \sin x) \cdot e^{-x^2}$

Im ersten Beispiel soll das globale Minimum f^* und der globale Minimizer x^* der Funktion



im Intervall $[-10, 10]$ eingeschlossen werden.

Das nachfolgende Programm `gop1_ex` schreibt mit den entsprechenden Eingaben für das Startintervall und die Toleranz die berechneten Einschließungen des globalen Minimums und des globalen Minimizers auf den Bildschirm.

```
program gop1_ex;
use
  li_ari,   { Interval arithmetic                }
  ddf_rrg,  { Differentiation arithmetic         }
  gop1;     { Global optimization one-dimensional }
const
  n = 20;   { Maximum number of optimizers to be computed }

var SearchInterval : Linterval;
    Tolerance      : real;

{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}
function f (x : DerivType) : DerivType;
begin
   f := (x + sin(x)) * exp(-sqr(x));
end;
{&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&}

begin { Main program }
  writeln;
  write('   ***********************************************');
  writeln('***************');
  write('   *   Enclosing all global minimizers and the ');
  writeln('global minimum   *');
  write('   *                  of a given function:');
  writeln('                     *');
  write('   ***********************************************');
  writeln('***************');
  writeln;
  write('Search interval      : '); read(SearchInterval);
  write('Tolerance (relative) : '); read(Tolerance); writeln;
  Min_g1(f,n,SearchInterval,Tolerance); writeln;
end.
```

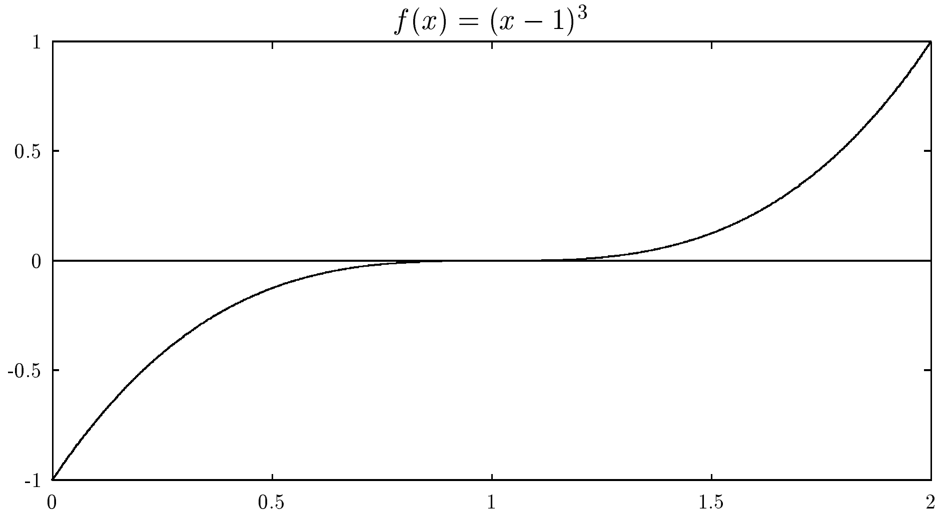
Übersetzt man dieses Programm, so erhält man das folgende Ablaufprotokoll:

```
   **********************************************************
   *   Enclosing all global minimizers and the global minimum   *
   *                  of a given function:                       *
   **********************************************************

Search interval      : [-10,10]    <enter>
Tolerance (relative) : 2e-20       <enter>

   [ -6.7957866001988153972 9E-001, -6.79578660019881539728E-001 ]
   encloses a locally unique candidate for a global minimizer!

   [ -8.24239398476076654249E-001, -8.24239398476076654247E-001 ]
   encloses the global minimum value!

   1 enclosure(s) for possible minimizers found.

   We have validated that there is a unique global minimizer!
```

Das durch $[-0.824...249, -0.824...247]$ eingeschlossene globale Minimum ist also gleichzeitig auch ein lokales Minimum[2], das nur an der Stelle $x^* \in [-0.679...729, -0.679...728]$ angenommen wird, in Übereinstimmung mit der Abbildung auf Seite 499.

11.1.1.2 $f(x) = (x-1)^3$

Das nächste Beispiel



[2]Beachten Sie dazu die Definition des lokalen Minimizers auf Seite 497.

zeigt, daß ein globales Minimum, das auf dem Rand des Intervalls $[0, 2]$ angenommen wird und nicht durch einen stationären Punkt gegeben ist, vom Programm korrekt eingeschlossen wird. Ändert man nämlich im Quelltext des Programms `prog1_ex` von Seite 500 den Funktionsterm durch die Zeile

```
f := power(x-1,3);
```

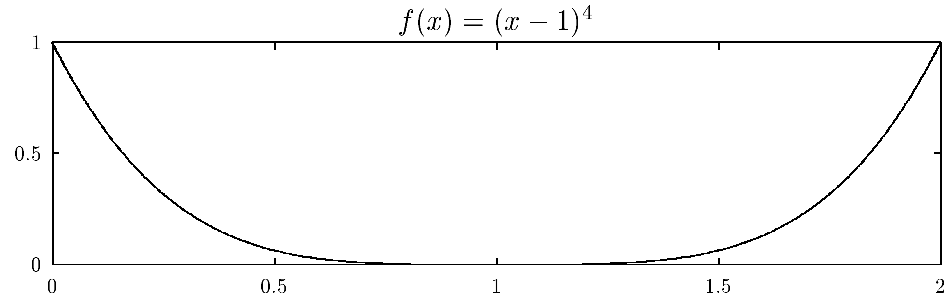
und startet das neu übersetzte Programm, so erhält man das nachfolgende Ablaufprotokoll:

```
   ***************************************************************
   *   Enclosing all global minimizers and the global minimum   *
   *                  of a given function:                       *
   ***************************************************************

Search interval      : [0,2]
Tolerance (relative) : 2e-20

   [  0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
   encloses a locally unique candidate for a global minimizer!

   [ -1.00000000000000000001E+000, -9.99999999999999999999E-001 ]
   encloses the global minimum value!

   1 enclosure(s) for possible minimizers found.

   We have validated that there is a unique global minimizer!
```

Das globale Minimum wird auch dann korrekt eingeschlossen, wenn es auf dem Intervallrand durch einen stationären Punkt gegeben ist. Wählen Sie dazu $[1, 2]$ als Startintervall und den Toleranzwert $2 \cdot 10^{-20}$, dann wird das globale Minimum 0 durch das Punktintervall $[0, 0]$ und der eindeutige globale Minimizer $x^* = 1$ durch $[1, 1]$ exakt eingeschlossen.

11.1.1.3 $f(x) = (x-1)^4$

Das nächste Beispiel



zeigt, daß ein globales Minimum, das im Innern des Intervalls $[0, 2]$ an der Stelle $x^* = 1$ angenommen wird und durch einen stationären Punkt gegeben ist, in dem f' eine mehrfache Nullstelle besitzt, nicht mehr eindeutig erkannt wird. Ändert man nämlich im Quelltext des Programms `prog1_ex` von Seite 500 den Funktionsterm durch die Zeile

```
f := power(x-1,4);
```

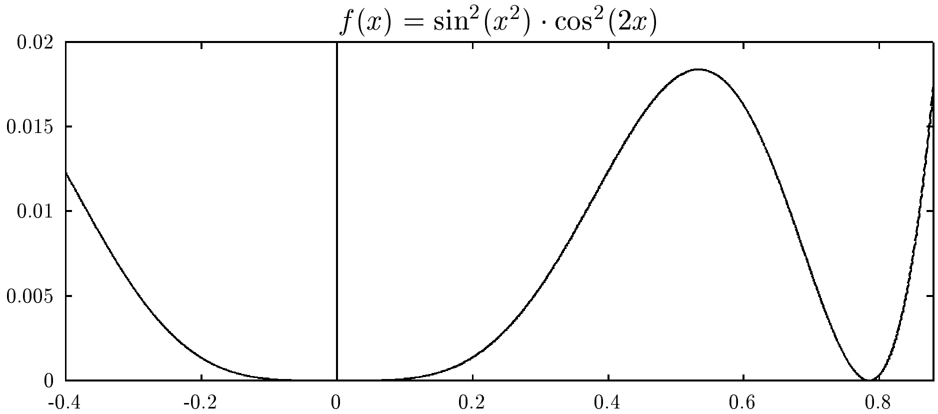
und startet das neu übersetzte Programm, so erhält man das nachfolgende Ablaufprotokoll:

```
   ************************************************************
   *   Enclosing all global minimizers and the global minimum   *
   *                  of a given function:                      *
   ************************************************************

Search interval      : [0,2]
Tolerance (relative) : 1e-20

   [  5.416666666666666666666E-001,  1.000000000000000000000E+000 ]
   may contain a local or global minimizer (not verified unique!)
   [  1.000000000000000000000E+000,  1.458333333333333333334E+000 ]
   may contain a local or global minimizer (not verified unique!)

   [  0.000000000000000000000E+000,  0.000000000000000000000E+000 ]
   encloses the global minimum value!

   2 enclosure(s) for possible minimizers found.
```

Obwohl der globale Minimizer $x^* = 1$ nicht eindeutig erkannt wird, erhält man für des globalen Minimums $f^* = 0$ sogar die exakte Punkteinschließung.

11.1.1.4 $f(x) = \sin^2(x^2) \cdot \cos^2(2x)$

Im nächsten Beispiel



besitzt die Funktion f im Intervall $[-0.4, 0.88]$ zwei verschiedene globale Minimizer $x_1^* = \pi/4$ und $x_2^* = 0$. Da x_2^* eine mehrfache Nullstelle von f' ist, kann dieser globale Minimizer vom Programm wieder nicht eindeutig erkannt werden, während im globalen Minimizer $x_1^* = \pi/4$ eine einfache Nullstelle von f' vorliegt, weshalb die maximalgenauen Einschließung von $\pi/4$ einen eindeutigen Kandidaten für ein globales Minimum enthält. Ändert man im Quelltext des Programms `prog1_ex` von Seite 500 den Funktionsterm durch die Zeile

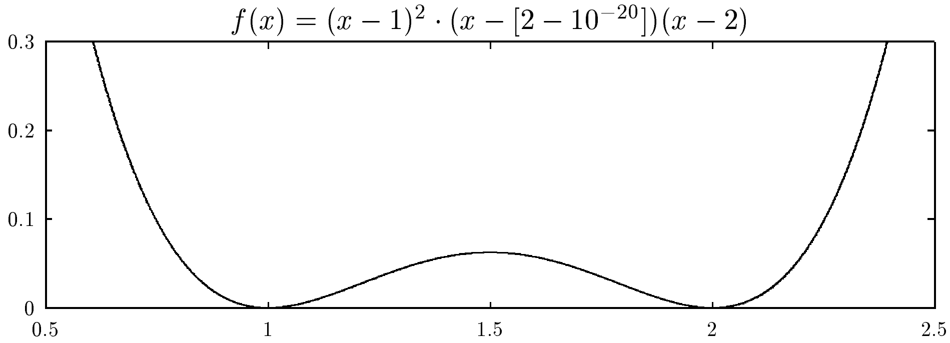
```
f := sqr( sin(sqr(x))*cos(2*x) );
```

und startet das neu übersetzte Programm, so erhält man das nachfolgende Ablaufprotokoll, das die obigen Aussagen bestätigt.

```
Search interval      : [-0.4,0.88]
Tolerance (relative) : 1e-20

   [  7.85398163397448309615E-001,  7.85398163397448309616E-001 ]
   encloses a locally unique candidate for a global minimizer!
   [ -1.64332393190105243540E-005,  2.27122411952590288298E-004 ]
   may contain a local or global minimizer (not verified unique!)

   [  0.00000000000000000000E+000,  1.30756550298756072888E-024 ]
   encloses the global minimum value!

   2 enclosure(s) for possible minimizers found.
```

11.1.1.5 $f(x) = (x-1)^2 \cdot (x - 1.99...99) \cdot (x-2)$

Im nächsten Beispiel



besitzt f neben dem globalen Minimizer $x^* \in [2 - 10^{-20}, 2]$ auch ein lokales Minimum bei $x_l = 1$, wobei $f(x_l) = 0$ und $f^* = f(x^*) < 0$ sich nur ganz minimal unterscheiden[3]. Ändert man im Quelltext des Programms `prog1_ex` von Seite 500 den Funktionsterm durch die Zeile

```
f := sqr(x-1)*(x-(Long(2)-1e-20))*(x-2);
```

und startet das neu übersetzte Programm, so erhält man das nachfolgende Ablaufprotokoll:

```
Search interval      : [0.5,2.5]
Tolerance (relative) : 1e-20

   [  1.99999999999999999999E+000,  2.00000000000000000000E+000 ]
   encloses a locally unique candidate for a global minimizer!
   [  5.00000000000000000000E-001,  1.00000000000000000000E+000 ]
   may contain a local or global minimizer (not verified unique!)
   [  1.00000000000000000000E+000,  1.50000000000000000000E+000 ]
   may contain a local or global minimizer (not verified unique!)

   [ -9.19900000000000000001E-037,  2.11553999999999980538E-033 ]
   encloses the global minimum value!

   3 enclosure(s) for possible minimizers found.
```

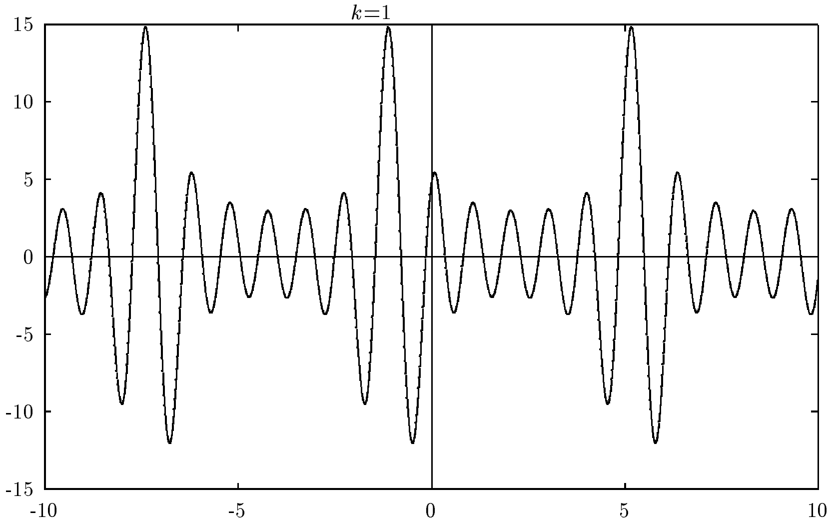
Das Programm vermutet also in der Umgebung von 1 weitere globale Minimizer, die in Wirklichkeit aber nicht existieren!

[3]Beachten Sie, daß $f(x)$ in der Umgebung von 2 zwei eng benachbarte, einfache Nullstellen besitzt und damit die x-Achse bei $x = 2$ nicht berührt!

11.1.1.6 $f(x) = -\sum_{k=1}^{5} k \cdot \sin((k+1)x + k)$

Als nächstes Beispiel betrachten wir Shubert's Funktion

$$f(x) = -\sum_{k=1}^{5} k \sin((k+1)x + k)$$



mit dem globalen Minimum $f^* = -12.031\ldots$, das im Intervall $[-10, +10]$ an den drei Stellen $-6.77457\ldots$, $-0.49139\ldots$, $5.79179\ldots$ angenommen wird. Ändert man im Quelltext des Programms `prog1_ex` von Seite 500 die Deklaration der Funktion durch die folgenden Zeilen

```
function f (x : DerivType) : DerivType;
var s : DerivType;
    k : integer;
begin
   s := Derivconst(0);
   For k := 1 to 5 do
      s := s + k*sin( (k+1)*x + k );
   f := -s;
end;
```

und startet das neu übersetzte Programm, so erhält man das nachfolgende Ablaufprotokoll:

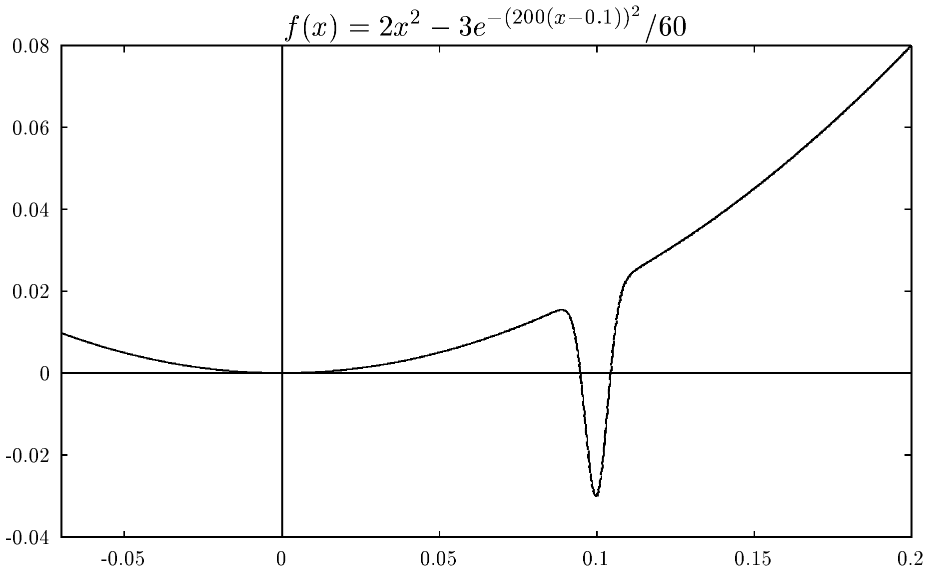
```
   ***********************************************************
   *   Enclosing all global minimizers and the global minimum   *
   *                  of a given function:                      *
   ***********************************************************

Search interval      : [-10,10]
Tolerance (relative) : 1e-20

   [ -4.91390836259314554061E-001, -4.91390836259314554060E-001 ]
   encloses a locally unique candidate for a global minimizer!
   [ -6.77457614343890103099E+000, -6.77457614343890103098E+000 ]
   encloses a locally unique candidate for a global minimizer!
   [  5.79179447092027192286E+000,  5.79179447092027192287E+000 ]
   encloses a locally unique candidate for a global minimizer!

   [ -1.20312494421671389482E+001, -1.20312494421671389480E+001 ]
   encloses the global minimum value!

   3 enclosure(s) for possible minimizers found.
```

Hinweise:

- Wegen der vielen relativen Extrema ist Shubert's Funktion ein geeignetes Testbeispiel zur Einschließung des globalen Minimums.
- Erweitert man das Startintervall $[-10, 10]$ z.B. auf $[-50, 50]$, so bleibt das globale Minimum $f^* = -12.0312494\ldots$ zwar erhalten, aber die Anzahl der globalen Minimizer wächst in diesem Fall von 3 auf 16.
- Nur die relativen Extrema mit den wirklich kleinsten Minima sind die Kandidaten für das existierende globale Minimum der Funktion.
- Das Programm nimmt bzgl. des Startintervalls $[-10, +10]$ an, daß der globale Minimizer in wenigstens einem der drei oben ausgegebenen Intervalle liegt.
- Wegen der von Null verschiedenen Intervalldurchmesser kann das Programm natürlich nicht erkennen, daß ein globaler Minimizer in **jedem** der drei oben angegebenen Intervalle enthalten ist.

11.1.1.7 $f_{a,b,c}(x) = 2x^2 - 3e^{-a(x-b)^2}/c$

Wählt man $a = 200,\ b = 0.1,\ c = 60$, so besitzt $f(x)$ ein sehr schmales und tiefes Tal, auf dessen Boden sich das globale Minimum befindet, das von vielen Programmen schon nicht mehr erkannt wird. Da unser Programm die Intervallrechnung anwendet und damit f nicht nur an einzelnen Punkten, sondern über einem ganzen Kontinuum auswertet, wird das globale Minimum stets mit Sicherheit eingeschlossen!



Ändert man im Quelltext des Programms `prog1_ex` von Seite 500 den Funktionsterm durch die Zeile

```
f := 2*sqr(x) - 3*exp(-sqr(200*(x-0.1)))/60;
```

und startet das neu übersetzte Programm, so erhält man das nachfolgende Ablaufprotokoll

```
  ***************************************************************
  *   Enclosing all global minimizers and the global minimum    *
  *                  of a given function:                       *
  ***************************************************************

Search interval      : [-0.05,0.2]
Tolerance (relative) : 1e-20
```

```
[  9.99000600199357258219E-002,  9.99000600199357258220E-002 ]
encloses a locally unique candidate for a global minimizer!

[ -3.00199840066700742019E-002, -3.00199840066700742018E-002 ]
encloses the global minimum value!

1 enclosure(s) for possible minimizers found.

We have validated that there is a unique global minimizer!
```

Sowohl das globale Minimum von f als auch der eindeutig bestimmte globale Minimizer werden von unserem Programm `prog1_ex` sogar maximalgenau eingeschlossen.

Das Tal fällt umso tiefer und schmaler aus, je kleiner $c > 0$ und je größer a gewählt werden. Zeigen Sie mit Hilfe des Programms `prog1_ex`, in dem die Prozedur `Min_g1` aufgerufen wird, daß das globale Minimum und der globale Minimizer auch noch für die Parameterwerte $a = 10^{+10}, c = 10^{-10}$ hochgenau eingeschlossen werden.

11.1.1.8 Nützliche Hinweise

1. Alle Standardfunktionen vom Typ `DerivType`, aus denen die zu untersuchende Funktion f mit Hilfe der vier Grundoperationen aufgebaut werden kann, findet man in der Tabelle auf Seite 351.

2. Falls die Maximalzahl n der zu speichernden Einschließungen möglicher Minimizer zu klein ist, muß diese Konstante im Programm `gop1_ex` von Seite 500 entsprechend vergrößert werden.

3. Die Prozedur `Min_g1` im Programm `gop1_ex` kann Fehlermeldungen erzeugen, wenn durch die unvermeidbaren Intervallüberschätzungen der jeweils zulässige Definitionsbereich einer der benutzten Standardfunktionen überschritten wird. Diese Fehlermeldungen lassen sich oft schon dadurch vermeiden, daß man das Startintervall geeignet unterteilt.

4. Das globale Minimum f^* wird von der Prozedur `Min_g1` auch dann noch korrekt eingeschlossen, wenn bei allen Einschließungen möglicher Minimizer–Kandidaten diese als nicht eindeutig erkannt werden, d.h. wenn jeder Minimizer–Kandidat durch

   ```
   may contain a local or global minimizer (not verified unique!)
   ```

   kommentiert wird, vergl. Seite 503.

5. Eine Verbesserung der Einschließung eines Minimizer–Kandidaten kann oft dadurch erreicht werden, daß man diese Einschließung als Startintervall einer neuen Rechnung benutzt.

6. Eine zusätzliche Verbesserung der Einschließung eines Minimizer–Kandidaten oder des globalen Minimums kann man oft auch dadurch erreichen, daß man mit Hilfe der Prozedur `Function_Analyser` von Seite 458 das Minimum der Funktion in der genannten Einschließung des Minimizer–Kandidaten noch deutlich enger einschließt. Aber bitte beachten Sie dabei, daß die Prozedur `Function_Analyser` nur erfolgreich arbeitet, wenn f' an der Minimumstelle keine mehrfache Nullstelle besitzt, während die globale Optimierung den Minimizer–Kandidaten zwar nicht als eindeutig erkennt, das globale Minimum der Funktion jedoch garantiert einschließt.

11.2 Reelle mehrdimensionale Funktionen

Gegeben sei eine reelle, zweimal stetig differenzierbare Funktion

$$f : D \to I\!R, \qquad [x] \subseteq D \subseteq I\!R^n, \quad n = 2, 3, \ldots$$
$$[x] := ([x]_1, \ldots, [x]_n) \qquad \text{mit} \quad [x]_i := [\underline{x}_i, \overline{x}_i], \quad \underline{x}_i, \overline{x}_i \in I\!R;$$

Gesucht wird eine garantierte Einschließung des kleinsten Wertes aller im n-dimensionalen Startintervall $[x]$ existierenden lokalen Minima von f. Ist z.B. die Funktion zweier Variabler

$$f(x_1, x_2) = -e^{-(x_1^2 + x_2^2)}$$

auf lokale Minima zu untersuchen, so ist f wie folgt zu deklarieren:

```
function f(x : HTvector) : HessType[0..2,0..2];
begin
    f := -exp(-sqr(x[1])-sqr(x[2]));
end;
```

wobei der Datentypen `HessType` und `HTvector` auf Seite 419 definiert und vom Hilfsmodul `hess_ari` exportiert werden. Eine Tabelle der zur Verfügung stehenden Funktionen vom Typ `HessType` findet man auf Seite 421.

Die Einschließung des kleinsten Wertes aller lokalen Minima von f liefert die Prozedur `Local_Min`, die vom Hilfsmodul `gop` exportiert wird und alle Ergebnisse auf dem Bildschirm ausgibt:

```
procedure Local_Min( function f(x : HTvector)
                        : HessType[0..ub(x),0..ub(x)];
  n              : integer;  { Maximalzahl der Minimizer }
  fDim           : integer;        {Anzahl der Variablen}
  SearchInterval : Livector;           { Startintervall }
  Tolerance      : real );   { Fehlerschr. der Minimizer }
```

Erfolgen keine Fehlermeldungen, so haben die auf dem Bildschirm ausgegebenen Ergebnisse der Prozedur `Local_Min` folgende Bedeutung:

Erfolgt keine Fehlermeldung, so gelten die folgenden Aussagen:

1. Es existiert wenigstens ein lokales Minimum von f, das auch auf dem Rand des Startintervalls $[x]$ liegen kann.
2. Durch ein Intervall I_0 wird der kleinste Wert $f^\vee \in I_0$ aller existierenden lokalen Minima eingeschlossen[4].
3. Es gibt keine stationären Punkte ($\nabla f = 0$) mit Funktionswerten, die kleiner sind als $I_0.inf$.
4. Das kleinste lokale Minimum $f^\vee \in I_0$ kann an mehreren isolierten Stellen $\zeta_i \in [x]$ angenommen werden, $f^\vee = f(\zeta_i)$.
5. Existiert nur ein $\zeta \in [x]$ mit $f^\vee = f(\zeta) \in I_0$, so ist ζ zwar die einzige lokale Minimumstelle, es kann aber durchaus sein, daß es auf dem Intervallrand Funktionswerte gibt, die kleiner sind als $f^\vee = f(\zeta)$.

Außer I_0 werden noch weitere Intervalle auf dem Bildschirm ausgegeben, die mit einem der beiden Kommentare versehen sind:

```
encloses a locally unique candidate for the smallest local minimum!

may contain a local or global minimizer!
```

- Intervalle mit dem ersten Kommentar enthalten zwar jeweils genau eine lokale Minimumstelle von f, aber nicht jedes dieser Intervalle muß ein ζ_i enthalten. Jedoch wird jedes der existierenden ζ_i von mindestens einem der Intervalle eingeschlossen!
- Intervalle mit dem zweiten Kommentar können zwar einen lokalen oder auch globalen Minimizer einschließen, es kann aber auch sein, daß ein solches Intervall keine lokale Minimumstelle einschließt.

Es ist am einfachsten, die möglichen Fälle anhand entsprechender Beispiele zu erläutern. Mit Hilfe des nachfolgenden Programms `gop_ex` können die gewählten Funktionen in vorgegebenen Startintervallen auf lokale Minima untersucht werden, wobei nach einer notwendigen Übersetzung und anschließendem Start des Programms die Ergebnisse auf dem Bildschirm dargestellt werden.

[4]Beachten Sie bitte, daß $f^\vee$ mit dem auf Seite 497 definierten globalen Minimum f^* nicht übereinstimmen muß!

11.2.1 Programmbeispiele

Mit Hilfe des nachfolgenden Programms `gop_ex` wird die Funktion

$$f(x_1, x_2) = -e^{-(x_1^2+x_2^2)}$$

in einem vorgegebenen Startintervall auf lokale Minima untersucht, wie es auf der vorhergehenden Seite beschrieben wurde. Eine andere Funktion ist entsprechend zu deklarieren, und nach dem anschließenden Übersetzen kann das Programm dann neu gestartet werden. Das Programm läßt sich mit der Tastenkombination `<Strg>+<C>` abbrechen.

```
program gop_ex;
use rrgi_ari, { Interval arithmetic with rrG-Type }
  li_ari,     { Interval arithmetic with Longreal }
  lmvi_ari,   { Interval matrix/vector arithmetic }
  hess_ari,   { Differentiation arithmetic       }
  gop;        { exports Local_Min(...)           }

const
  n    = 40;   { Maximum number of optimizers to be computed }
  fDim = 2;    { Number of function variables                }

var SearchInterval : Livector[1..fDim];
   Tolerance       : real;

function f (x: HTvector) : HessType[0..fDim,0..fDim];
begin
   f := -exp( -sqr(x[1])-sqr(x[2]) );
end;

begin { Main program }
   repeat
      writeln;
      write('   ',fDim,' Search intervals   : ');
      read(SearchInterval);
      write('   Tolerance (relative) : '); read(Tolerance);
      writeln;
      Local_Min(f,n,fDim,SearchInterval,Tolerance);
   until false;
end.
```

11.2.1.1 $f(x_1, x_2) = -e^{-(x_1^2+x_2^2)}$

Im ersten Beispiel betrachten wir die Funktion

$$f(x_1, x_2) = -e^{-(x_1^2+x_2^2)}$$

die nur im Ursprung ein lokales Minimum besitzt, das in einem Startintervall, welches den Ursprung enthält, sogar ein globales Minimum ist.

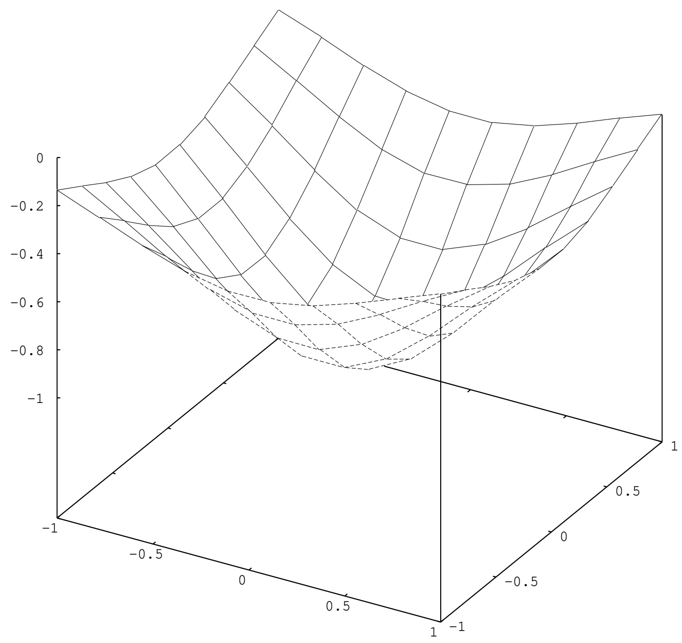


Abbildung 11.1: $f(x_1, x_2) = -e^{-(x_1^2+x_2^2)}$

Das Programm `gop_ex` von Seite 513 liefert mit der Deklaration von f

```
function f (x: HTvector) : HessType[0..fDim,0..fDim];
begin
   f := -exp( -sqr(x[1]) - sqr(x[2]) );
end;
```

nach seinem Start das folgende Ablaufprotokoll:

```
2 Search intervals   : [-1,1.01] [-1,1.01]
Tolerance (relative) : 1e-20


*****************************************************************
*   Enclosing the smallest value of all existing local minima   *
*   of a twice continuously differentiable function. Dim =  2   *
*****************************************************************
```

```
[ -4.00000000000000000000E-038,  5.00000000000000000000E-038 ]
[ -5.00000000000000000000E-038,  4.00000000000000000000E-038 ]
encloses a locally unique candidate for the smallest local minimum!

   [ -1.00000000000000000000E+000, -9.99999999999999999999E-001 ]
   encloses the unique local minimum

   We have validated that there is a unique local minimizer!
   There are probably smaller function values on the interval bounds

   1 interval enclosure(s) found
```

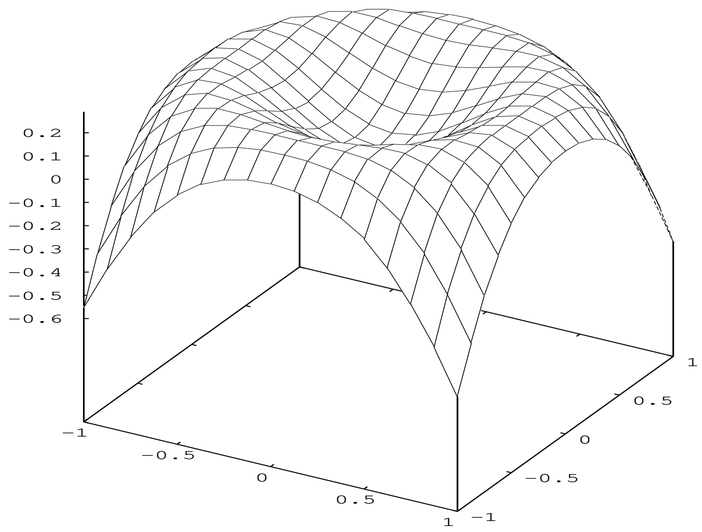
Anmerkungen:

- Die Procedure `Local_Min` erkennt also nicht, daß der lokale Minimizer $\zeta = (0,0)$ auch gleichzeitig der globale Minimizer ist, da auf mögliche kleinere Funktionswerte auf dem Intervallrand hingewiesen wird.
- Das Startintervall $([-1, 1.01], [-1, 1.01])$ wurde zum Ursprung absichtlich unsymmetrisch gewählt, damit für den eindeutigen Minimizer $(0,0)$ auch nur eine Einschließung ausgegeben wird. Es ist nämlich möglich, daß die vom Hilfsmodul `gop` exportierte Prozedur `Local_Min(...)` für einen lokalen Minimizer mehrere Einschließungen berechnet. Wählen Sie zur Bestätigung das zum Minimizer $\zeta = (0,0)$ symmetrische Startintervall $([-1,1], [-1,1])$.
- Zeigen Sie, daß mit den Startintervallen $[0,1]$ $[0,1]$ das lokale Minimum auch auf einem Eckpunkt des Startintervalls erkannt wird.

11.2.1.2 $f(x_1, x_2) = \cos(x_1^2 + x_2^2) - e^{-(x_1^2+x_2^2)}$, (Teil 1)

Im nächsten Beispiel betrachten wir die Funktion

$$f(x_1, x_2) = \cos(x_1^2 + x_2^2) - e^{-(x_1^2+x_2^2)}$$



Die obige Graphik von $f(x_1, x_2)$ zeigt deutlich, daß z.B. der Funktionswert $f(+1, -1) = -0.551482\ldots$ im vorderen Eckpunkt $(+1, -1)$ deutlich kleiner ist als das lokale Minimum $f^\vee = f(0, 0) = 0$, das damit natürlich kein globales Minimum im zweidimensionalen Intervall $([-1, 1], [-1, 1])$ ist. Bitte vergleichen Sie dazu auch den Punkt 5. von Seite 512.

Das Programm `gop_ex` von Seite 513 liefert mit der entsprechenden Deklaration von f

```
function f (x: HTvector) : HessType[0..fDim,0..fDim];
begin
   f := -exp( -sqr(x[1]) - sqr(x[2]) )
        +cos(  sqr(x[1]) + sqr(x[2]) );
end;
```

nach dem Start das folgende Ablaufprotokoll:

```
2 Search intervals   : [-1.01,1] [-1.01,1]  <enter>
Tolerance (relative) : 1e-20                <enter>


*****************************************************************
*   Enclosing the smallest value of all existing local minima   *
*   of a twice continuously differentiable function. Dim =  2   *
*****************************************************************


[ -1.10000000000000000000E-039,  1.30000000000000000000E-039 ]
[ -4.00000000000000000000E-039,  3.00000000000000000000E-039 ]
encloses a locally unique candidate for the smallest local minimum!

   [ -1.15570000000200000001E-024,  4.62140000000500000001E-025 ]
   encloses the unique local minimum

   We have validated that there is a unique local minimizer!
   There are probably smaller function values on the interval bounds

   1 interval enclosure(s) found
```

Anmerkungen:

1. Das Startintervall $([-1.01, 1], [-1.01, 1])$ wurde zum Ursprung absichtlich unsymmetrisch gewählt, damit für den eindeutigen Minimizer $(0, 0)$ auch nur eine Einschließung ausgegeben wird. Es ist nämlich möglich, daß die Prozedur `Local_Min(..)` für einen lokalen Minimizer mehrere Einschließungen berechnet.

2. Welche Ergebnisse erhält man mit dem Startintervall $([-1, 1], [-1, 1])$?

3. Man kann die obige Einschließung des gefundenen Minimizers weiter verbessern, wenn man $([-1.1\mathrm{E}-039, 1.3\mathrm{E}-039], [-4\mathrm{E}-039, 3\mathrm{E}-039])$ als neues Startintervall wählt und den Minimizer erneut einschließt. Das Ergebnis

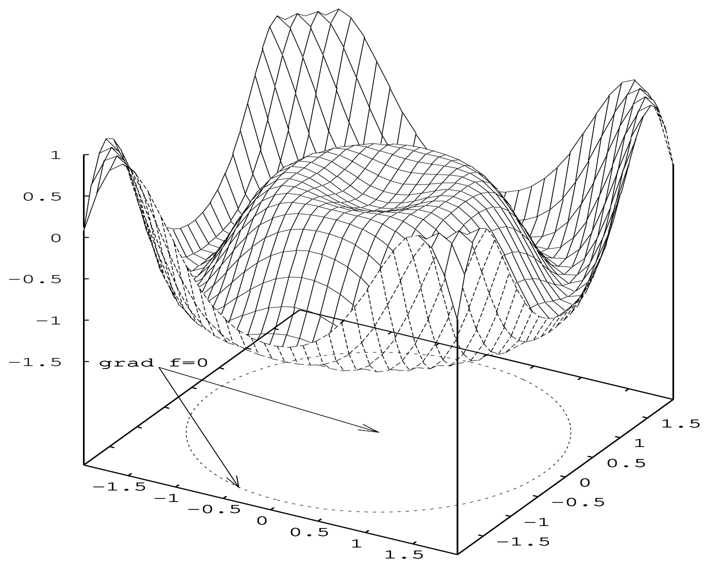
   ```
   [ -1.30000000000000000000E-080,  1.20000000000000000000E-080 ]
   [ -1.10000000000000000000E-079,  7.00000000000000000000E-080 ]
   ```

   läßt sich dann im nächsten Schritt ganz analog verbessern!

11.2.1.3 $f(x_1, x_2) = \cos(x_1^2 + x_2^2) - e^{-(x_1^2+x_2^2)}$, (Teil 2)

Wir betrachten noch einmal die gleiche Funktion

$$f(x_1, x_2) = \cos(x_1^2 + x_2^2) - e^{-(x_1^2+x_2^2)}$$



jetzt aber im etwas breiteren Intervall $([-1.97, +1.97], [-1.97, +1.97])$. Man erkennt deutlich, daß $f(x_1, x_2)$ im Ursprung und auf dem Kreis um $(0, 0)$ mit dem Radius $r = 1.759\ldots$ stationäre Punkte besitzt. Da auf diesem Kreis keine *lokalen* Minima liegen und die Funktionswerte auf diesem Kreis deutlich kleiner sind als der lokale Minimumwert 0 im Ursprung, kann die Prozedur `Local_Min` im obigen Startintervall bei der Suche nach dem kleinsten lokalen Minimum nicht erfolgreich sein, da es bei einem fehlerfreien Programmablauf nach Punkt 3. von Seite 512 neben diesem kleinsten lokalen Minimum keine stationären Punkte mit noch kleineren Funktionswerten geben darf!

> Die Prozedur `Local_Min(..)` findet nur dann den kleinsten Wert aller lokalen Minima einer zweimal stetig differenzierbaren Funktion, wenn es im vorgegebenen Intervall keine stationären Punkte mit noch kleineren Funktionswerten gibt!

Mit der gleichen Deklaration der Funktion f von Seite 516 liefert das Programm `gop_ex` von Seite 513 das folgende Ablaufprotokoll:

```
   2 Search intervals   : [-1.97,+1.97] [-1.97,+1.97]
   Tolerance (relative) : 1e-20

   ***************************************************************
   *   Enclosing the smallest value of all existing local minima   *
   *   of a twice continuously differentiable function. Dim =  2   *
   ***************************************************************

Error: Internal list too small.
       Increase "ListSize" and recompile "gop.p"!!

./gop_ex returned with return code 65280.
```

Auch eine Vergrößerung von ListSize bringt keine Abhilfe, da die stationären Punkte auf dem Kreis um den Ursprung **kontinuierlich** verteilt sind. Auf diesem Kreis kann es daher keine **lokalen** Extrema geben!

Abschließend noch einige Bemerkungen zum Kreis um $(0, 0)$ mit dem Radius $r = 1.759\ldots$, auf dem der Gradient von f überall verschwindet. Da die Funktion $f(x_1, x_2)$ auf jedem Kreis um $(0, 0)$ konstante Funktionswerte besitzt, wählen wir die Transformation $t = x_1^2 + x_2^2$ und berechnen mit Hilfe der Prozedur `Function_Analyser` von Seite 458 das relative Minimum der Funktion

$$f(x_1, x_2) \equiv g(t) = \cos(t) - e^{-t}, \quad t = x_1^2 + x_2^2 \in [0, 4]$$

Mit der Deklaration

```
function f (t: Deriv3Type) : Deriv3Type;
begin
   f := -exp(-t) + cos(t);
end;
```

liefert das Programm `nlfz3_2` von Seite 467 mit den Eingaben

```
Order (0,1,2) of the derivative = ?
1                                  <enter>
Search interval      : [0,4]       <enter>
Tolerance (relative) : 1e-20       <enter>
```

für die Minimumstelle t_0 die Einschließung:

```
[  3.09636393241064611562E+000,  3.09636393241064611563E+000 ],
```

woraus sich der gesuchte Kreisradius $r = \sqrt{3.096363\ldots} = 1.75964880\ldots$ direkt berechnet. Für den Funktionswert an der Minimumstelle liefert das Programm zusätzlich: $g(t_0) = -1.044190\ldots$

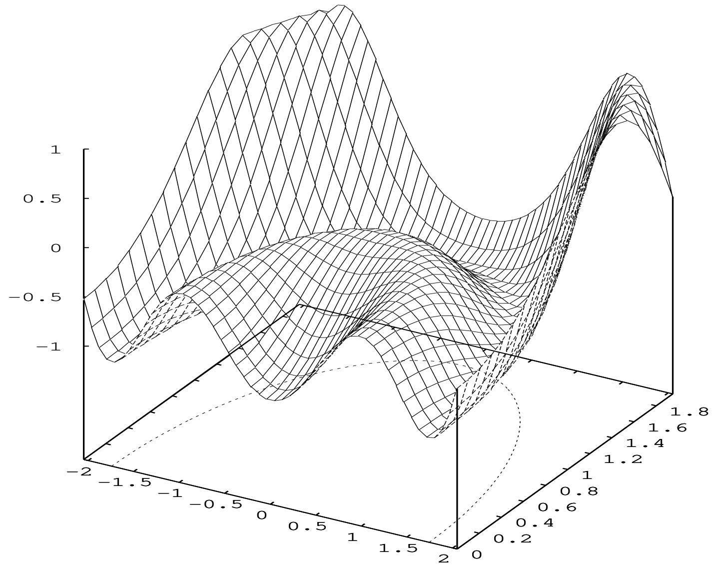
11.2.1.4 $f(x_1, x_2) = \cos(x_1^2 + x_2^2) - 2.0955 \cdot e^{-(x_1^2+x_2^2)}$

Nach den bisherigen Ergebnissen fassen wir noch einmal der Arbeitsweise der Prozedur `Local_Min` zusammen[5]:

> Liefert die Prozedur `Local_Min` keine Fehlermeldungen, so gilt folgendes:
>
> Das Minimum aller Funktionswerte auf stationären Punkten ist ein lokales Minimum, dessen Wert $f^\vee$ und dessen lokale Minimumstelle(n) $\zeta_i \in [x]$ von obiger Prozedur garantiert eingeschlossen werden. Es gilt: $f^\vee = f(\zeta_i),\ i = 1, 2, \dots\ ;$

Wir betrachten jetzt die Funktion

$$f(x_1, x_2) = \cos(x_1^2 + x_2^2) - 2.0955 \cdot e^{-(x_1^2+x_2^2)}$$



Der gezeichnete Halbkreis um $(0,0)$ mit dem Radius $r = 1.743\dots$ ist die Projektion der stationären Punkte mit der z-Koordinate $-1.09508\dots$. Man erkennt deutlich, daß die Funktionswerte in diesen stationären Punkten mit dem lokalen Minimum $f(0,0) = -1.0955$ praktisch übereinstimmen.

[5]Vergleichen Sie bitte auch die fünf Punkte auf Seite 512 oben.

Da das lokale Minimum jedoch tatsächlich etwas kleiner ist als die konstanten Funktionswerte in den beschriebenen stationären Punkten, wird man erwarten, daß die Prozedur `Local_Min` den Wert $f^\vee = -1.0955$ als kleinsten lokalen Minimumwert einschließt.

In der Tat liefert das Programm `gop_ex` von Seite 513 mit der entsprechenden Deklaration von f

```
function f (x: HTvector) : HessType[0..fDim,0..fDim];
begin
   f := -2.0955*exp( -sqr(x[1]) - sqr(x[2]) )
         + cos( sqr(x[1]) + sqr(x[2]) );
end;
```

nach dem Start das folgende Ablaufprotokoll:

```
   2 Search intervals   : [-5,5] [-5,5]   <enter>
   Tolerance (relative) : 1e-20           <enter>

[ -3.71395502000000000000E-019,  1.16337890000000000000E-019 ]
[ -6.35283026000000000000E-019,  1.25267081000000000000E-019 ]
encloses a locally unique candidate for the smallest local minimum!

[ -3.71395502000000000000E-019,  1.16337890000000000000E-019 ]
[ -1.25267081000000000000E-019,  6.35283026000000000000E-019 ]
encloses a locally unique candidate for the smallest local minimum!

[ -4.00000000000000000000E-034,  3.00000000000000000000E-034 ]
[ -4.00000000000000000000E-034,  3.00000000000000000000E-034 ]
encloses a locally unique candidate for the smallest local minimum!

[ -4.00000000000000000000E-034,  3.00000000000000000000E-034 ]
[ -3.00000000000000000000E-034,  4.00000000000000000000E-034 ]
encloses a locally unique candidate for the smallest local minimum!

   [ -1.09550000000000000001E+000, -1.09549999999999999999E+000 ]
   encloses the smallest value of all local minima!

   There is no stationary point with a function value smaller than:
                -1.09550000000000000001E+000
   There are probably smaller function values on the interval bounds

   4 interval enclosure(s) found
```

Das lokale Minimum $f^\vee = -1.0955$ wird also hochgenau eingeschlossen, und jedes der ersten vier Intervalle enthält den lokalen Minimizer $\zeta = (0, 0)$.

11.2.1.5 $f = -1.0..01 \cdot e^{-(x_1^2+x_2^2)} - e^{-(x_1-3)^2-(x_2-3)^2}$

Wir betrachten jetzt die Funktion

$$f(x_1, x_2) = -(1 + 10^{-20}) \cdot e^{-(x_1^2+x_2^2)} - e^{-((x_1-3)^2+(x_2-3)^2)}$$

die an ihren beiden einzigen lokalen Minimumstellen ζ_1, ζ_2 nahezu identische Funktionswerte besitzt. Dies erkennt man am einfachsten, wenn man unten die Graphik der Funktion $-f(x_1, x_2)$ betrachtet, die an den Stellen ζ_1, ζ_2 relative Maxima besitzt, die sich optisch wesentlich besser darstellen lassen.

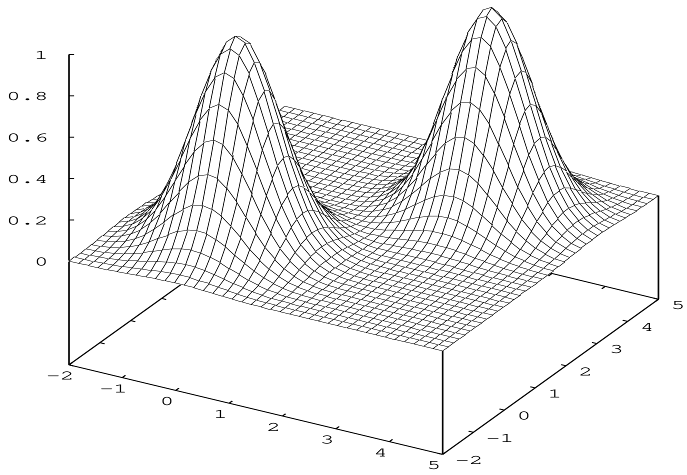


Abbildung 11.2: 3D–Graphik der Funktion $-f(x_1, x_2)$

Es soll jetzt untersucht werden, welche der beiden verschiedenen Punkte ζ_1, ζ_2 von der Prozedur `Local_Min` als die Stelle mit dem kleinsten relativen Minimum erkannt wird.

Das Programm `gop_ex` von Seite 513 liefert dazu mit der Deklaration von f

```
function f (x: HTvector) : HessType[0..fDim,0..fDim];
begin
   f := -(Long(1)+1e-20) * exp( -sqr(x[1])-sqr(x[2]) )
        -exp( -sqr(x[1]-3)-sqr(x[2]-3) );
end;
```

nach seinem Start das folgende Ablaufprotokoll:

```
   2 Search intervals   : [-5,5] [-5,5]
   Tolerance (relative) : 1e-20

   *****************************************************************
   *   Enclosing the smallest value of all existing local minima   *
   *   of a twice continuously differentiable function. Dim =  2   *
   *****************************************************************

[  4.5689963589147071199E-008,  4.5689963589147071200E-008 ]
[  4.5689963589147071199E-008,  4.5689963589147071200E-008 ]
encloses a locally unique candidate for the smallest local minimum!

[  2.9999999543100364108 5E+000,  2.9999999543100364108 6E+000 ]
[  2.9999999543100364108 5E+000,  2.9999999543100364108 6E+000 ]
encloses a locally unique candidate for the smallest local minimum!

   [ -1.0000000152299839198 7E+000, -1.0000000152299839198 6E+000 ]
   encloses the smallest value of all local minima!

   There is no stationary point with a function value smaller than:
                   -1.0000000152299839198 7E+000
   There are probably smaller function values on the interval bounds

   2 interval enclosure(s) found
```

Anmerkungen:

1. Das Programm findet also nicht die Stelle mit dem kleinsten lokalen Minimum. Es werden lediglich Einschließungen der beiden Kandidaten ζ_1, ζ_2 geliefert, wobei jedoch sichergestellt ist, daß in einer der beiden Einschließungen diese Stelle wirklich liegt!

2. Auch wenn bei diesem Beispiel die Stelle mit dem kleinsten lokalen Minimum nicht eindeutig gefunden wird, erhält man dennoch stets eine garantierte und hier sogar maximalgenaue Einschließung des kleinsten lokalen Minimumwertes.

3. Mit Hilfe der Intervallrechnung im `rrG`–Format kann man recht einfach nachweisen, daß die obige erste Einschließung in der Nähe des Ursprungs die Stelle mit dem kleinsten lokalen Minimum enthält.

4. Die Prozedur `Local_Min` kann grundsätzlich nicht entscheiden, ob es auf dem zweidimensionalen Intervallrand Funktionswerte gibt, die noch kleiner sind als das kleinste lokale Minimum, vergl. dazu Seite 516.

11.2.1.6 Die Funktion von Branin

Nachdem wir die Arbeitsweise der Prozedur `Local_Min(..)` von Seite 511 anhand mehrerer Beispiele studiert haben[6], betrachten wir jetzt die Funktion von Branin, deren kleinstes lokales Minimum jetzt nicht mehr so einfach zu erraten ist.

$$f_B(x) = \left(\frac{5}{\pi} \cdot x_1 - \frac{5.1}{4\pi^2} \cdot x_1^2 + x_2 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

$f_B(x_1, x_2)$ nimmt das kleinste lokale Minimum $\frac{5}{4\pi}$ an den folgenden Stellen an:

$$\zeta_1 = (-\pi, \frac{491}{40}), \quad \zeta_2 = (\pi, \frac{91}{40}), \quad \zeta_3 = (3\pi, \frac{99}{40}), \qquad f_B(\zeta_i) = \frac{5}{4\pi}$$

Die drei lokalen Minima werden in der unteren Graphik durch je einen Pfeil gekennzeichnet.

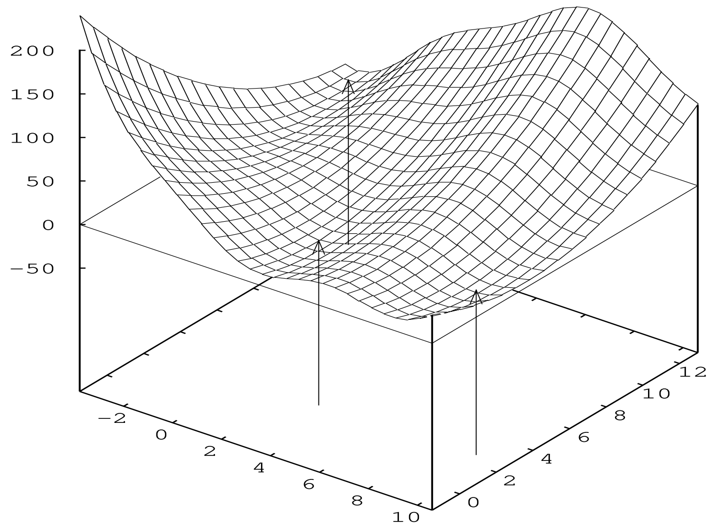


Abbildung 11.3: Funktion von Branin

Mit der nachfolgenden Deklaration der Funktion f_B liefert das Programm `gop_ex` von Seite 513 das umseitige Ablaufprotokoll:

[6] Vergleichen Sie die Zusammenfassungen auf den Seiten 512,518.

```
function f (x: HTvector) : HessType[0..fDim,0..fDim];
{ Deklaration der Funktion von Branin:                         }
{ Pi_rrGi schliesst pi ein und wird von rrgi_util exportiert }
begin
   f := sqr(5/Pi_rrGi*x[1] - 5.1/(4*sqr(Pi_rrGi))*sqr(x[1])
        + x[2] - 6) + 10*(1-0.125/Pi_rrGi)*cos(x[1]) + 10;
end;
```

```
   2 Search intervals   : [-5,10] [0,15]    <enter>
   Tolerance (relative) : 1e-20             <enter>

[  9.42477796076937971538E+000,  9.42477796076937971539E+000 ]
[  2.47499999999999999999E+000,  2.47500000000000000001E+000 ]
encloses a locally unique candidate for the smallest local minimum!

[  3.14159265358979323846E+000,  3.14159265358979323847E+000 ]
[  2.27499999999999999999E+000,  2.27500000000000000001E+000 ]
encloses a locally unique candidate for the smallest local minimum!

[ -3.14159265358979323847E+000, -3.14159265358979323846E+000 ]
[  1.22749999999999999999E+001,  1.22750000000000000001E+001 ]
encloses a locally unique candidate for the smallest local minimum!

   [  3.97887357729738339422E-001,  3.97887357729738339423E-001 ]
   encloses the smallest value of all local minima!

   There is no stationary point with a function value smaller than:
                    3.97887357729738339422E-001
   There are probably smaller function values on the interval bounds

   3 interval enclosure(s) found
```

Anmerkungen:

1. Die obigen drei ersten zweidimensionalen Intervalle sind hochgenaue Einschließungen der lokalen Minimizer $\zeta_3, \zeta_2, \zeta_1$.

2. Die Funktion von Branin besitzt auf den Intervallrändern keine Funktionswerte, die kleiner sind als $f_B(\zeta_i) = 5/4\pi$. Der Nachweis muß dazu aber gesondert geführt werden!

3. Da `Pi_rrGi` $\ni \pi$ benötigt wird, ist im Programm `gop_ex` zusätzlich das Modul `rrgi_util` zu importieren.

11.2.1.7 Die Funktion von Levy

Die Funktion von Levy ist als Funktion zweier Variabler definiert durch:

$$\begin{aligned} f_L(x_1, x_2) &= \sum_{i=1}^{5} i \cos((i-1)x_1 + i) \sum_{j=1}^{5} j \cos((j+1)x_2 + j) \\ &= +(x_1 + 1.42513)^2 + (x_2 + 0.80032)^2 \end{aligned}$$

f_L besitzt in $I\!R^2$ etwa 700 lokale Minimizer, so daß es für herkömmliche Approximationsmethoden praktisch unmöglich ist, ohne zusätzliche Informationen über den Funktionsverlauf sowohl den globalen Minimizer $\zeta = (-1.306\ldots, -1.424\ldots)$ als auch das globale Minimum $f_L(\zeta) = -176.13\ldots$ zu finden. Der untere Graph zeigt den Funktionsverlauf in einem nur sehr kleinen Bereich um den Ursprung:

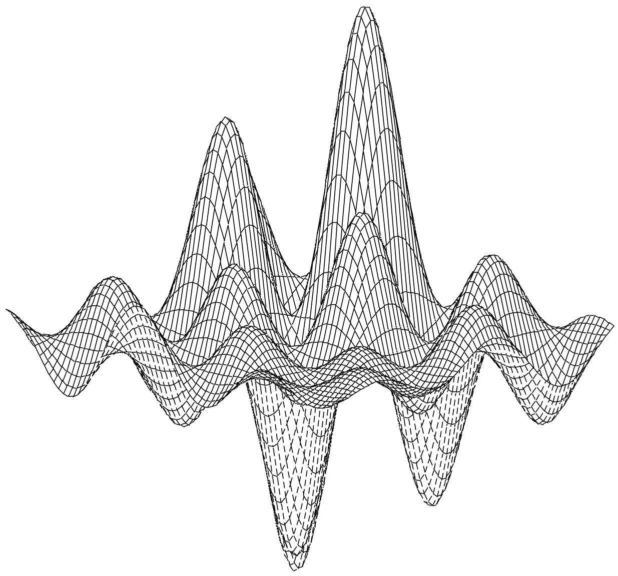


Abbildung 11.4: Funktion von Levy

Mit Hilfe des Programms `gop_ex` von Seite 513 werden die Werte ζ und $f_L(\zeta)$ problemlos eingeschlossen. Dazu ist zunächst die auf der folgenden Seite angegebene Deklaration von f_L zu realisieren und das Programm neu zu übersetzen.

```
function f (x: HTvector) : HessType[0..fDim,0..fDim];
{ Deklaration der Funktion von Levy: }
var
  isum, jsum : HessType[0..fDim,0..fDim];
  i          : integer;
begin
  isum := 0; jsum := 0;
  for i:=1 to 5 do
  begin
    isum := isum + i*cos((i-1)*x[1] + i);
    jsum := jsum + i*cos((i+1)*x[2] + i);
  end;
   f := isum * jsum + sqr(x[1] + rrGintval(1.42513))
                        + sqr(x[2] + rrGintval(0.80032));
end;
```

Nach dem Start des neu übersetzten Programms `gop_ex` erhält man das folgende Ablaufprotokoll:

```
  2 Search intervals   : [-20,20] [-20,20]
   Tolerance (relative) : 1e-20

   ****************************************************************
   *   Enclosing the smallest value of all existing local minima   *
   *   of a twice continuously differentiable function. Dim =  2   *
   ****************************************************************

[ -1.306853009753572145 31E+000, -1.30685300975357214530E+000 ]
[ -1.42484504156068114951E+000, -1.42484504156068114950E+000 ]
encloses a locally unique candidate for the smallest local minimum!

   [ -1.76137578001629392221E+002, -1.76137578001629392219E+002 ]
   encloses the unique local minimum

   We have validated that there is a unique local minimizer!
   There are probably smaller function values on the interval bounds

   1 interval enclosure(s) found
```

Bestätigen Sie, daß ζ nicht nur ein lokaler sondern auch der globale Minimizer von f_L ist. Zeigen Sie dazu mit Hilfe des Programms `gop1_ex` von Seite 500, daß das Minimum der Funktion f_L auf den Intervallrändern größer ist als $f_L(\zeta) = -176.13\ldots$.

Kapitel 12

Komplexe Polynom-Nullstellen

Das komplexe Polynom $P_n : \mathbb{C} \to \mathbb{C}$ der Ordnung n sei definiert durch:

$$P_n(z) = \sum_{i=0}^{n} p_i \cdot z^i, \qquad p_i, z \in \mathbb{C}, \quad i = 0, \ldots, n, \quad p_n \neq 0. \tag{12.1}$$

Nach dem Fundamentalsatz der Algebra besitzt $P_n(z)$ genau n Nullstellen z_i, wenn man die Vielfachheit der einzelnen Nullstellen entsprechend mitzählt. Die Berechnung der z_i ist ein nicht-triviales Problem, und die meisten Algorithmen liefern nur Approximationen der z_i, wobei über die Genauigkeit kaum brauchbare Aussagen gemacht werden.

In diesem Kapitel werden **PASCAL-XSC** Routinen vorgestellt, die bei komplexen Polynomen mit nur einfachen Nullstellen alle diese Nullstellen i.a. maximalgenau einschließen, wobei die punktförmigen Polynomkoeffizienten p_i auf der Maschine exakt darstellbar sein müssen.

Bezüglich der Präzision[1] einer Rechnung gibt es zwei Möglichkeiten:

1. Bei der Rechnung im `real`–Format müssen die p_i mit 13 BCD–Stellen darstellbar sein, und die Ergebnisse werden durch Intervalle gleicher Präzision eingeschlossen.

2. Bei der Rechnung in einem festen `staggered`–Format müssen die p_i mit 42 BCD–Stellen darstellbar sein, und die Ergebnisse werden durch Intervalle in doppelter Präzision, also mit 84 Stellen eingeschlossen.

[1] Die Präzision gibt Auskunft über die in der Rechnung benutzte Stellenzahl und ist nicht mit der Genauigkeit zu verwechseln, mit der ein z_i berechnet wird. Ist $[z_i]$ eine Einschließung von z_i, so ist der Durchmesser von $[z_i]$ eine garantierte Oberschranke für den absoluten Fehler. Bei höherer Präzision wird man diese Oberschranke i.a. entsprechend verkleinern.

12.1 Routinen zur Nullstelleneinschließung

Zur Einschießung aller einfachen Nullstellen z_i eines in (12.1) definierten komplexen Polynoms $P_n(z)$ werden die folgenden Datentypen benötigt, die von den in Klammern gesetzten Modulen exportiert werden:

type `CPolynomial  = dynamic array[*] of complex`	(type_def)
type `CIPolynomial = dynamic array[*] of cinterval`	(type_def)
type `SCPolynomial = dynamic array[*] of LStaggcomplex`	(type_def)

Die Einschließung aller einfachen Nullstellen eines Polynoms $P_n(z)$ erfolgt mit Hilfe der Prozedur

procedure `AllPZeros( p :` ■; **var** `zz :` ◇;
var `Err : integer`)

Eingangsgröße:

`p :` $(n+1)$ punktförmige Polynomkoeffizienten

Ausgangsgrößen:

`zz :` i.a. maximalgenaue Einschließung der n einfachen Nullstellen z_i
`Err:` Fehlerkode; `Err=0`: Keine Fehler aufgetreten.

Zuordnung der Symbole ■, ◇ zu den erlaubten Datentypen der Prozedur AllPZeros		
`p :` ■	`CPolynomial`	`SCPolynomial`
`zz :` ◇	`CIPolynomial`	`SCPolynomial`
exportierendes Modul	cpzero	scpzero
Präzision	13 BCD–Stellen	84 BCD–Stellen
Beispielprogramm auf	Seite 532	Seite 546

Mit Hilfe der Prozedur `CPolyZero` wird folgende Aufgabe bearbeitet:

Gegeben ist ein nach (12.1) definiertes komplexes Polynom $P_n(z)$ mit punktförmigen Koeffizienten und mit wenigstens einer einfachen Nullstelle z_i. Gegeben sei ferner eine hinreichend gute Approximation dieser Nullstelle. Gesucht ist eine optimale Einschließung von z_i und der Koeffizienten des reduzierten Polynoms

$$Q_{n-1}(z) := \frac{P_n(z)}{z - z_i} = \sum_{i=0}^{n-1} q_i \cdot z^i \tag{12.2}$$

$$\boxed{\begin{array}{l} \textbf{procedure}\ \ \texttt{CPolyZeros(p : } \blacksquare\texttt{; z : } \boxdot\texttt{; } \textbf{var}\ \texttt{qq : } \diamond_q\texttt{;} \\ \qquad\qquad\qquad\qquad \textbf{var}\ \texttt{zz : } \diamond_{z_i}\texttt{; } \textbf{var}\ \texttt{Err : integer)} \end{array}}$$

Eingangsgrößen:

`p` : $(n+1)$ punktförmige Polynomkoeffizienten von $P_n(z)$

`z` : Hinreichend gute Approximation einer einfachen Nullstelle z_i

Ausgangsgrößen:

`qq` : i.a. maximalgenaue Einschließung der $n-1$ Koeffizienten q_i
`zz` : i.a. maximalgenaue Einschließung der einfachen Nullstelle z_i
`Err`: Fehlerkode; `Err=0`: Keine Fehler aufgetreten.

Zuordnung der Symbole $\blacksquare, \boxdot, \diamond_q, \diamond_{z_i}$ zu den erlaubten Datentypen der Prozedur CPolyZero		
`p` : $\blacksquare$	`CPolynomial`	`SCPolynomial`
`z` : $\boxdot$	`complex`	`LStaggcomplex`
`qq` : $\diamond_q$	`CIPolynomial`	`SCPolynomial`
`zz` : $\diamond_{z_i}$	`cinterval`	`LStaggcomplex`
exportierendes Modul	cpzero	scpzero
Präzision	13 BCD–Stellen	84 BCD–Stellen
Beispielprogramm auf	Seite 535	Seite 537

12.2 Programmbeispiele

12.2.1 $z^4 - (5+3i)z^3 + (6+7i)z^2 - (54-22i)z + 120 + 90i$

Gesucht sind alle Nullstellen des komplexen Polynoms

$$P_4(z) = z^4 - (5+3i)z^3 + (6+7i)z^2 - (54-22i)z + (120+90i)$$

Exakte Nullstellen: $z_{1,2} = \pm\sqrt{2} \cdot (1+2i), \quad z_3 = 5, \quad z_4 = 3i;$

Das nachfolgende Programm `cproot1`

```
program cproot1(input,output);
{ Enclosing all zeros of a complex polynomial with only }
{ simple zeros                                          }
{ Precision of all enclosures:      13 BCD-digits.      }
use
  type_def,
  c_ari,
  ci_ari,
  cpeval,
  cpzero;

procedure main(n : integer);
var
   ErrCode,i : integer;
   z         : complex;
   zz        : cinterval;
   p         : CPolynomial[0..n];
   qq        : CIPolynomial[0..n-1];
   zzi       : CIPolynomial[1..n];
begin

   writeln('Enter the coefficients in increasing order: ');
   write('a <enter> for real,   (a,b) <enter> for ');
   writeln('complex values');
   writeln;
   read(p);  writeln;
   AllPZeros(p,zzi,ErrCode);
   if ErrCode = 0 then
      For i := 1 to ub(p) do
      begin
```

```
         writeln(i,'. enclosure of a simple zero');
         writeln(zzi[i]);
         write('--------------------------------------');
         writeln('----------');
      end         else  writeln(CPolyZeroErrMsg(ErrCode));
end; { main }

var n : integer;
begin
   write('   ************************************');
   writeln('************************************');
   write('   *   Enclosing all zeros of a complex ');
   writeln('polynomial with only simple zeros   *');
   write('   *         Zero enclosures with mantissa length ');
   writeln('of 13 BCD-digits.        *');
   write('   ************************************');
   writeln('************************************');
   writeln;
   n := -1;
   while n<0 do
   begin
      write('Enter the degree of the polynomial (>=0): ');
      read(n);
      writeln;
   end;
   main(n);
end.
```

liefert mit den entsprechenden Eingaben ohne die Programmüberschrift das folgende Ablaufprotokoll:

```
Enter the degree of the polynomial (>=0): 4

Enter the coefficients in increasing order:
a <enter> for real,   (a,b) <enter> for complex values

  x^0 * (120,90)    <enter>
+ x^1 * (-54,22)    <enter>
+ x^2 * (6,7)       <enter>
+ x^3 * (-5,-3)     <enter>
+ x^4 * 1           <enter>
```

```
1. enclosure of a simple zero
( [  1.414213562373E+000,  1.414213562374E+000 ],
  [  2.828427124746E+000,  2.828427124747E+000 ] )
------------------------------------------------
2. enclosure of a simple zero
( [ -1.000000000000E-267,  1.000000000000E-267 ],
  [  2.999999999999E+000,  3.000000000001E+000 ] )
------------------------------------------------
3. enclosure of a simple zero
( [  4.999999999999E+000,  5.000000000001E+000 ],
  [ -1.000000000000E-267,  1.000000000000E-267 ] )
------------------------------------------------
4. enclosure of a simple zero
( [ -1.414213562374E+000, -1.414213562373E+000 ],
  [ -2.828427124747E+000, -2.828427124746E+000 ] )
------------------------------------------------
```

Anmerkungen:

1. Wenn die einfachen Nullstellen z_i des Polynoms $P_n(z)$ nicht zu dicht nebeneinander liegen und wenn die Präzision von 13 BCD–Stellen ausreicht, so liefert `AllPZeros` bei kurzen Laufzeiten i.a. maximalgenaue Einschließungen aller einfachen Nullstellen.

2. Schließen Sie mit dem Programm `cproot1` alle einfachen Nullstellen des Wilkinson Polynoms $P_n(z) = \prod_{i=1}^{n}(z-i)$ mit $n = 5$ ein, indem Sie $P_5(z)$ vorher ausmultiplizieren.

3. Bei mehrfachen Nullstellen von $P_n(z)$ liefert die Prozedur `AllPZeros` eine entsprechende Fehlermeldung.

4. Zeigen Sie, daß die doppelte Nullstelle $z_{1,2} = i$ des Polynoms $Q_2(z) = z^2 - 2iz - 1$ mit dem Programm `cproot1` nicht eingeschlossen werden kann.

5. Das Polynom

   $$P_2(z) := (z-1)(z-[1+10^{-10}]) \equiv z^2 - (2+10^{-10})z + (1+10^{-10})$$

   besitzt die beiden einfachen Nullstellen $z_1 = 1,\ z_2 = 1 + 10^{-10}$. Zeigen Sie, daß diese zu dicht liegenden Nullstellen mit Hilfe des Programms `cproot1` nicht mehr erfolgreich eingeschlossen werden.

12.2.2 $z^3 - (10+2i)z^2 - (1-20i)z + 10$

Das komplexe Polynom

$$P_3(z) = z^3 - (10+2i)z^2 - (1-20i)z + 10 \tag{12.3}$$

besitzt die doppelte Nullstelle $z_{1,2} = i$ und die einfache Nullstelle $z_3 = 10$. Mit der recht groben Approximation $z_3 = 10 \approx 7$ liefert das Programm

```
program cproot2(input,output);
{ Enclosing a simple zero of a complex polynomial with a    }
{ given zero approximation. Enclosures for the coefficients }
{ of the deflated polynomial are computed too.              }
{ Enclosure precision:         13 BCD-digits.               }
use
  c_ari,
  ci_ari,
  cpeval,
  cpzero;

procedure main(n : integer);
var
   ErrCode,i : integer;
   z         : complex;
   zz        : cinterval;
   p         : CPolynomial[0..n];
   qq        : CIPolynomial[0..n-1];

begin
   writeln('Enter the coefficients in increasing order: ');
   write('a <enter> for real,   (a,b) <enter> for complex ');
   writeln('values');
   writeln;
   read(p);  writeln;
   write('Enter the starting approximation: ');  read(z);
   writeln;
   CPolyZero(p,z,qq,zz,ErrCode);
   if ErrCode=0 then
   begin
      writeln('Polynomial:');  writeln(p);
      writeln('Simple zero found in:');  writeln(zz);  writeln;
```

```
      writeln('Coefficients of the reduced polynomial:');
      writeln(qq);
   end       else writeln(CPolyZeroErrMsg(ErrCode));
end; { main }

var n : integer;
begin
   write('   **************************************');
   writeln('************************************');
   write('   *   Enclosing a simple zero of a complex ');
   writeln('polynomial with a given zero   *');
   write('   *   approximation.    Enclosures for the ');
   writeln('coefficients of the deflated   *');
   write('   *   polynomial are computed too. ');
   writeln('                                       *   ');
   write('   *         All enclosures with mantissa ');
   writeln('length of 13 BCD-digits.        *');
   write('   **************************************');
   writeln('************************************');
   n := -1;  writeln;
   while n<0 do
   begin
      write('Enter the degree of the polynomial (>=0): ');
      read(n);  writeln;
   end;
   main(n);
end.
```

mit den entsprechenden Eingaben ohne die Programmüberschrift das nachfolgende Ablaufprotokoll:

```
Enter the degree of the polynomial (>=0): 3   <enter>

Enter the coefficients in increasing order:
a <enter> for real,   (a,b) <enter> for complex values

  x^0 * 10           <enter>
+ x^1 * (-1,20)      <enter>
+ x^2 * (-10,-2)     <enter>
+ x^3 * 1            <enter>

Enter the starting approximation: 7   <enter>
```

```
Polynomial:
  (  1.000000000000E+001,  0.000000000000E+000 ) * x^0
+ ( -1.000000000000E+000,  2.000000000000E+001 ) * x^1
+ ( -1.000000000000E+001, -2.000000000000E+000 ) * x^2
+ (  1.000000000000E+000,  0.000000000000E+000 ) * x^3

Simple zero found in:
( [  9.999999999999E+000,  1.000000000001E+001 ],
  [ -3.000000000000E-033,  2.000000000000E-033 ] )

Coefficients of the reduced polynomial:
  ([ -1.000000000001E+000, -9.999999999999E-001 ],
   [ -2.318031565537E-032,  2.111989020682E-032 ] ) * x^0
+ ([ -1.000000000000E-033,  2.000000000000E-033 ],
   [ -2.000000000001E+000, -1.999999999999E+000 ] ) * x^1
+ ([  1.000000000000E+000,  1.000000000000E+000 ],
   [  0.000000000000E+000,  0.000000000000E+000 ] ) * x^2
```

Anmerkungen:

1. Die obigen drei Intervalle schließen die Koeffizienten des reduzierten Polynoms $P_2(z) = z^2 - 2iz - 1$ ein, das nur die doppelte Nullstelle $z_{1,2} = i$ besitzt.

2. Besitzt das reduzierte Polynom $Q_{n-1}(z)$ eine weitere einfache Nullstelle z_i, so läßt sich mit $Q_{n-1}(z)$ eine Approximation $\zeta \approx z_i$ dieser Nullstelle berechnen. Mit diesem ζ und dem ursprünglichen Polynom $P_n(z)$ kann dann mit Hilfe des Programms `cproot2` eine garantierte Einschließung der neuen Nullstelle z_i von $P_n(z)$ gewonnen werden. In dieser Weise lassen sich damit alle einfachen, nicht zu dicht liegenden Nullstellen des Polynoms $P_n(z)$ erfolgreich einschließen.

Die gleichen Rechnungen für das in (12.3) definierte komplexe Polynom $P_3(z)$ lassen sich in höherer Präzision mit Hilfe des folgenden Programms `scproot2` durchführen:

```
program scproot2(input,output);
{ Mit Hilfe einer Naeherung wird eine einfache Nullstelle z0  }
{ eines komplexen Polynoms Pn(z) der Ordnung n eingeschlossen }
{ Zusaetzlich werden auch die Koeffizienten des reduzierten   }
{ Polynoms  Pn(z)/(z-z0)  eingeschlossen.                     }
{ Liegt in der Naehe der Naeherung eine mehrfache Nullstelle  }
```

```
{ vor, so erfolgt eine entsprechende Fehlermeldung.           }
{ Die Ordnung n und die Polynomkoeffizienten sind einzugeben. }
{ Alle Einschliessungen erfolgen mit einer Praezision von  84 }
{ Stellen.                                                    }
{ Blomquist, 06.04.2000                                       }
use
  type_def,
  lcstagg,
  scpoly,
  scpzero;

procedure main(n : integer);
var
   ErrCode : integer;
   p       : SCPolynomial[0..n  ,1..2,-1..LStagg_prec];
   z       : LStaggcomplex[1..2,-1..LStagg_prec];
   qq      : SCPolynomial[0..n-1,1..2,-1..LStagg_prec+2];
   zz      : LStaggcomplex[1..2,-1..LStagg_prec+2];

begin
   writeln('Enter the coefficients in increasing order: ');
   writeln;
   read(p);  writeln;
   writeln('Enter the starting approximation: ');  read(z);
   writeln;
   CPolyZero(p,z,qq,zz,ErrCode);
   if ErrCode=0 then
   begin
      writeln('Polynomial:');  writeln(p);
      writeln('Simple zero found in:');  writeln(zz);  writeln;
      writeln('Coefficients of the reduced polynomial:');
      writeln(qq);
   end          else writeln(SCPolyZeroErrMsg(ErrCode));
end; { main }

var n : integer;
begin
   set_LStagg_prec(1);
   n := -1;
   while n<0 do
```

```
   begin
      write('   ************************************');
      writeln('**********************************');
      write('   *   Enclosing a simple zero of a complex ');
      writeln('polynomial with a given zero   *');
      write('   *   approximation.    Enclosures for the ');
      writeln('coefficients of the deflated   *');
      write('   *   polynomial are computed too. ');
      writeln('                                      *   ');
      write('   *         All enclosures with mantissa ');
      writeln('length of 84 BCD-digits.        *');
      write('   ************************************');
      writeln('**********************************');
      writeln;
      write('Enter the degree of the polynomial (>=0): ');
      read(n);   writeln;
   end;
   main(n);
end.
```

Die Eingabe der Polynomkoeffizienten und der Nullstellen–Näherung erfolgt in einer Präzision von 42 BCD–Stellen, und die Ausgabe der Nullstellen–Einschließung und die Einschließung der Koeffizienten des reduzierten Polynoms $P_2(z)$ erfolgt mit 84 BCD–Stellen. Mit den entsprechenden Eingaben für $P_3(z)$ liefert das Programm `scproot2` das folgende Ablaufprotokoll:

```
 ********************************************************************
 *   Enclosing a simple zero of a complex polynomial with a given zero   *
 *   approximation.    Enclosures for the coefficients of the deflated   *
 *   polynomial are computed too.                                        *
 *         All enclosures with mantissa length of 84 BCD-digits.         *
 ********************************************************************

Enter the degree of the polynomial (>=0): 3   <enter>

Enter the coefficients in increasing order:

-----------------------------------------------------------------------
coefficient of order 0:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r       <enter>
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
10 0    <enter>
-----------------------------------------------------------------------
```

```
coefficient of order 1:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
c       <enter>
--- Realteil     :
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-1 0
--- Imaginaerteil:
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
20 0    <enter>
----------------------------------------------------------------------------
coefficient of order 2:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
c       <enter>
--- Realteil     :
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-10 0   <enter>
--- Imaginaerteil:
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-2 0
----------------------------------------------------------------------------
coefficient of order 3:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r       <enter>
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
1 0     <enter>

Enter the starting approximation:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r       <enter>
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
7 0     <enter>

Polynomial:
  (  1.00000000000000000000E+001
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]

 0.00000000000000000000E+000
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
 ) * x^0
+ ( -1.00000000000000000000E+000
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]

 2.00000000000000000000E+001
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
 ) * x^1
```

```
+ ( -1.00000000000000000000E+001
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]

-2.00000000000000000000E+000
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
 ) * x^2
+ (  1.00000000000000000000E+000
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]

 0.00000000000000000000E+000
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
 ) * x^3

Simple zero found in:
(  1.00000000000000000000E+001
 5.14947281231015057363E-190
-0.00000000000000000000E+000
[ -9.75394569159886285687E-190,  9.75394569159886285687E-190 ]

-7.42168169214376247915E-190
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -2.57327712969316733660E-189,  2.57327712969316733660E-189 ]
 )

Coefficients of the reduced polynomial:
  ( -1.00000000000000000000E+000
-2.78032271044783567970E-189
-0.00000000000000000000E+000
[ -2.50000000000000000001E-188,  2.50000000000000000001E-188 ]

-1.51847268023329836233E-188
-5.00000000000000000000E-209
-0.00000000000000000000E+000
[ -5.00000000000000000001E-188,  5.00000000000000000001E-188 ]
 ) * x^0
+ (  3.69639395405137873205E-190
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -1.50000000000000000001E-189,  1.50000000000000000001E-189 ]

-2.00000000000000000000E+000
-7.44544801152270787695E-190
-0.00000000000000000000E+000
[ -3.00000000000000000001E-189,  3.00000000000000000001E-189 ]
 ) * x^1
```

```
+ (  1.00000000000000000000E+000
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]

 0.00000000000000000000E+000
 0.00000000000000000000E+000
 0.00000000000000000000E+000
[ -0.00000000000000000000E+000,  0.00000000000000000000E+000 ]
 ) * x^2
```

12.2.3 $(z-1)(z-[1+10^{-39}])(z-31)$

Auf Seite 534 haben wir bereits ein Polynom $P_2(z)$ kennen gelernt, dessen einfache Nullstellen so dicht nebeneinander liegen, daß deren Einschließung im 13–stelligen `real`–Format nicht mehr möglich war. Wir betrachten jetzt das Polynom

$$\begin{aligned} P_3(z) &= (z-1)(z-[1+10^{-39}])(z-31) \\ &= z^3-(33+10^{-39})z^2+(63+32\cdot 10^{-39})z-31\cdot(1+10^{-39}) \end{aligned}$$

mit den einfachen, reellen Nullstellen: $z_1=1,\ z_2=1+10^{-39},\ z_3=31.$

Um auch die sehr dicht liegenden Nullstellen $z_{1,2}$ noch erfolgreich einzuschließen, können im nachfolgenden Programm `scproot1` die punktförmigen Koeffizienten mit einer Präzision von 42 BCD–Stellen eingegeben werden. Die Einschließung der Nullstellen erfolgt dann sogar mit einer Präzision von 84 BCD–Stellen. Da die Einschließungen i.a. maximalgenau erfolgen, lassen sich damit praktisch alle Genauigkeitsanforderungen erfüllen!

```
program scproot1(input,output);
{ Schliesst alle Nullstellen eines komplexen Polynoms der    }
{ Ordnung n ein, falls diese NStn. alle einfach sind.        }
{ Bei mehrfachen Nullstellen erfolgt eine Fehlermeldung.     }
{ Die Ordnung n und die Polynomkoeffizienten sind einzugeben }
{ Die Einschliessungen erfolgen i.a. auf 84 Stellen genau!   }
use
  type_def,
  lcstagg,
  scpoly,
  scpzero;
```

```
procedure main(n : integer);
var
   ErrCode,i : integer;
   p         : SCPolynomial[0..n  ,1..2,-1..LStagg_prec];
   zz        : SCPolynomial[1..n  ,1..2,-1..LStagg_prec+2];
begin
   writeln('Enter the coefficients in increasing order: ');
   read(p);   writeln;
   AllPZeros(p,zz,ErrCode);
   if ErrCode = 0 then
      For i := 1 to ub(p) do
      begin
         writeln(i:2,'. enclosure of a simple zero:');
         writeln(zz[i]);
         write('------------------------------------');
         writeln('--------------------------');
      end   else  writeln(SCPolyZeroErrMsg(ErrCode));
end; { main }

var n : integer;
begin
   set_LStagg_prec(1);  n := -1;
   while n<0 do
   begin
      write('   **************************************');
      writeln('************************************');
      write('   *   Enclosing all zeros of a complex ');
      writeln('polynomial with only simple zeros   *');
      write('   *         Zero enclosures with mantissa ');
      writeln('length of 84 BCD-digits.         *');
      write('   **************************************');
      writeln('************************************');
      writeln;
      write('Enter the degree of the polynomial (>=0): ');
      read(n);   writeln;
   end;
   main(n);
end.
```

Die Eingabe der entsprechenden Polynom–Koeffizienten liefert für $P_3(z)$ das folgende Ablaufprotokoll:

```
 ************************************************************************
 *   Enclosing all zeros of a complex polynomial with only simple zeros   *
 *         Zero enclosures with mantissa length of 84 BCD-digits.         *
 ************************************************************************

Enter the degree of the polynomial (>=0): 3   <enter>

Enter the coefficients in increasing order:

--------------------------------------------------------------------------
coefficient of order 0:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r               <enter>
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-31 -31e-39     <enter>
--------------------------------------------------------------------------
coefficient of order 1:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r               <enter>
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
63 32e-39       <enter>
--------------------------------------------------------------------------
coefficient of order 2:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r               <enter>
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-33 -1e-39      <enter>
--------------------------------------------------------------------------
coefficient of order 3:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r               <enter>
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
1 0             <enter>

 1. enclosure of a simple zero:
(  1.00000000000000000000E+000
 1.00000000000000000000E-039
 1.48381756538160139128E-063
[ -5.87223463199933161328E-063,  5.87223463199933161329E-063 ]

 2.93489050579686914500E-065
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -5.06848847928447159780E-064,  5.06848847928447159780E-064 ]
 )
--------------------------------------------------------------
```

```
 2. enclosure of a simple zero:
(  9.99999999999999999999E-001
 9.99999999999999999999E-022
 9.99999999999999999990E-043
[ -3.00000000000000000000E-062,  3.00000000000000000000E-062 ]

 8.81163503716392634850E-065
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -1.44246513057126287970E-063,  1.44246513057126287970E-063 ]
 )
--------------------------------------------------------------
 3. enclosure of a simple zero:
(  3.10000000000000000000E+001
 1.72222222222222222123E-110
 8.00000000000000000000E-131
[ -1.83333333333333333336E-110,  1.83333333333333333336E-110 ]

-1.11111117270679012385E-211
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -3.00000001321412037038E-210,  3.00000001321412037038E-210 ]
 )
--------------------------------------------------------------
```

Es gelingt also, die beiden sehr dicht zusammenliegenden, einfachen Nullstellen $z_1 = 1,\ z_2 = 1 + 10^{-39}$ von $P_3(z)$ jeweils erfolgreich einzuschließen.

12.2.4 Nullstellen der Legendre–Polynome

Als nächstes Beispiel betrachten wir das Legendre–Polynom der Ordnung 4:

$$\begin{aligned} P_4(z) &:= \frac{1}{8}(35z^4 - 30z^2 + 3) \\ &= 4.375z^4 - 3.75z^2 + 0.375 \end{aligned}$$

Das Programm `scproot1` von Seite 542 liefert mit den entsprechenden Eingabedaten das folgende Ablaufprotokoll:

```
*************************************************************************
*   Enclosing all zeros of a complex polynomial with only simple zeros   *
*         Zero enclosures with mantissa length of 84 BCD-digits.         *
*************************************************************************

Enter the degree of the polynomial (>=0): 4
Enter the coefficients in increasing order:
---------------------------------------------------------------------------
coefficient of order 0:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
0.375 0
---------------------------------------------------------------------------
coefficient of order 1:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
0 0
---------------------------------------------------------------------------
coefficient of order 2:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-3.75 0
---------------------------------------------------------------------------
coefficient of order 3:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
0 0
---------------------------------------------------------------------------
coefficient of order 4:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
4.375 0
```

```
 1. enclosure of a simple zero:
(  3.39981043584856264802E-001
 6.65759103244687200575E-022
 8.69770914352592953975E-043
[  1.82102003046323703447E-063,  1.82102003046323703448E-063 ]

-4.52430640397799705825E-232
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -9.05000000000000000001E-231,  9.05000000000000000001E-231 ] )
----------------------------------------------------------------
 2. enclosure of a simple zero:
(  8.61136311594052575223E-001
 9.46488892809505095725E-022
 3.79629717637615721920E-043
[  9.06529471495048865703E-064,  9.06529471495048865705E-064 ]

 7.20124715038028179480E-218
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -3.59865356752823765331E-217,  3.59865356752823765331E-217 ] )
----------------------------------------------------------------
 3. enclosure of a simple zero:
( -3.39981043584856264802E-001
-6.65759103244687200575E-022
-8.69770914352592953975E-043
[ -1.82102003046323703448E-063, -1.82102003046323703447E-063 ]

 6.32730499129581188050E-217
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -2.27700000000000000001E-215,  2.27700000000000000001E-215 ] )
----------------------------------------------------------------
 4. enclosure of a simple zero:
( -8.61136311594052575223E-001
-9.46488892809505095725E-022
-3.79629717637615721920E-043
[ -9.06529471495048865705E-064, -9.06529471495048865704E-064 ]

 2.55072263081147275885E-216
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -2.54916770697947491311E-215,  2.54916770697947491311E-215 ] )
----------------------------------------------------------------
```

Anmerkungen:

1. Bis auf eine Ausnahme werden also alle Nullstellen von $P_4(z)$ sogar maximalgenau eingeschlossen.
2. Da die Nullstellen der Legendre Polynome alle rein reell sind, können die obigen Einschließungen des Imaginärteils der Nullstellen durch das Punktintervall $[0, 0]$ ersetzt werden.
3. Schließen Sie alle Nullstellen des Legendre Polynoms $P_{11}(z)$ mit einer Präzision von 84 BCD–Dezimalstellen ein.

$$P_{11}(z)=\frac{1}{256}(88179z^{11}-230945z^{9}+218790z^{7}-90090z^{5}+15015z^{3}-693z)$$

12.2.5 $(z-i)(z-[10^{-35}+i])(z^2-1)$

Wir betrachten das Polynom

$$\begin{aligned} P_3(z) &= (z-i)(z-[10^{-35}+i])(z^2-1) \\ &= z^4-(10^{-35}+2i)\cdot z^3+(-2+10^{-35}i)\cdot z^2+ \\ &= +(10^{-35}+2i)\cdot z+(1-10^{-35}i) \end{aligned}$$

dessen Nullstellen $z_1 = i,\ z_2 = 10^{-35} + i,\ z_{3,4} = \pm 1$ einzuschließen sind. Mit Hilfe des Programms `scproot1` von Seite 542 erhält man jedoch die Fehlermeldung

```
Exponent range restricted ... ,
```

da die intern benutzte staggered Arithmetik den langen Akkumulator benutzt, der jedoch nur für das `real`–Format ausgelegt ist, während die Nullstelleneinschließung im `Longreal`–Format mit einem etwa doppelt breiten Exponentenbereich erfolgt.

Um die bei den Iterationen auftretenden zu kleinen Zehnerexponenten zu vermeiden, multipliziert man $P_3(z)$ einfach mit 10^{+35} und erhält das zu $P_3(z)$ äquivalente Polynom

$$\begin{aligned} Q_3(z) := {} & 10^{+35}z^4-(1+2\cdot 10^{+35}i)\cdot z^3+(-2\cdot 10^{+35}+i)\cdot z^2+ \\ & +(1+2\cdot 10^{+35}i)\cdot z+(10^{+35}-i) \quad\Longleftrightarrow\quad P_3(z) \end{aligned}$$

Das Programm `scproot1` von Seite 542 liefert dann mit den Eingabedaten des Polynoms $Q_3(z)$ das folgende Ablaufprotokoll, nach dem alle Nullstellen von $P_3(z)$ jetzt ohne Fehlermeldung erfolgreich eingeschlossen werden:

```
 ************************************************************************
 *   Enclosing all zeros of a complex polynomial with only simple zeros   *
 *        Zero enclosures with mantissa length of 84 BCD-digits.          *
 ************************************************************************

Enter the degree of the polynomial (>=0): 4   <enter>

Enter the coefficients in increasing order:
---------------------------------------------------------------------------
coefficient of order 0:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
c   <enter>
--- Realteil     :
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
1e35 0    <enter>
--- Imaginaerteil:
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-1 0      <enter>
---------------------------------------------------------------------------
coefficient of order 1:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
c   <enter>
--- Realteil     :
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
1 0       <enter>
--- Imaginaerteil:
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
2e35 0    <enter>
---------------------------------------------------------------------------
coefficient of order 2:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
c   <enter>
--- Realteil     :
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-2e35 0   <enter>
--- Imaginaerteil:
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
1 0       <enter>
---------------------------------------------------------------------------
coefficient of order 3:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
c   <enter>
--- Realteil     :
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-1 0      <enter>
--- Imaginaerteil:
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
-2e35 0   <enter>
```

```
coefficient of order 4:
** Fuer rein reelle bzw. komplexe staggered Variable r bzw. c eingegeben: **
r   <enter>
--- 1 LStagg components and 1 Linterval are needed in this sequence ---
1e35 0    <enter>
 1. enclosure of a simple zero:
(  1.00000000000000000000E-035
 3.55793390978880792270E-113
-0.00000000000000000000E+000
[ -1.87044999999999999970E-112,  1.87044999999999999970E-112 ]

 9.99999999999999999999E-001
 9.99999999999999999999E-022
 9.99999999999999999995E-043
[  4.99999999999999999999E-063,  5.00000000000000000001E-063 ] )
----------------------------------------------------------------
 2. enclosure of a simple zero:
(  1.14347515473858846630E-113
-0.00000000000000000000E+000
-0.00000000000000000000E+000
[ -2.37119999999999999803E-112,  2.37119999999999999803E-112 ]

 1.00000000000000000000E+000
 1.94980007997856887555E-096
 5.00000000000000000000E-117
[ -5.00000000125000000315E-096,  5.00000000125000000315E-096 ] )
----------------------------------------------------------------
 3. enclosure of a simple zero:
(  9.99999999999999999999E-001
 9.99999999999999999999E-022
 9.99999999999999999995E-043
[  4.99999999999999999999E-063,  5.00000000000000000001E-063 ]

 1.27375000000004299268E-163
 5.20000000000000000000E-184
-0.00000000000000000000E+000
[ -1.28500000000000000001E-163,  1.28500000000000000001E-163 ] )
----------------------------------------------------------------
 4. enclosure of a simple zero:
( -1.00000000000000000000E+000
-9.45312499999897404250E-165
-0.00000000000000000000E+000
[ -1.45000000000000000001E-163,  1.45000000000000000001E-163 ]

 2.49999999952193981093E-148
 5.00000000000000000000E-169
-0.00000000000000000000E+000
[ -3.50000000000000000001E-148,  3.50000000000000000001E-148 ] )
```

Kapitel 13

Numerische Integration

13.1 Einschließung bestimmter Integrale

In diesem Kapitel wird eine Routine angegeben, mit der eine garantierte Einschließung des Integrals

$$I := \int_a^b f(x)\,dx, \qquad a, b \in \mathbb{R} \tag{13.1}$$

einer hinreichend oft differenzierbaren Funktion $f : \mathbb{R} \to \mathbb{R}$ berechnet werden kann. Benutzt wird dabei ein adaptives Romberg–Verfahren, wobei zu berechnende höhere Ableitungen mit Hilfe einer Differentiations–Arithmetik eingeschlossen werden. Die notwendigen Routinen und Datentypen exportiert das auf Seite 397 beschriebene Modul `litaylor`.

Zur Einschließung von I exportiert das Modul `iromberg` die folgende Funktion:

```
function  INTEGRAL( function f(x : litaylor) :
(iromberg)                          litaylor[0..ub(x)];
                    a,b     : Linterval;
                    abs_err : Linterval;
                    var Err : integer ) : Linterval;
```

Ist beispielsweise

$$f(x) = \frac{100}{1 + 100 \cdot x^2}$$

die zu integrierende Funktion, so ist f im Quelltext eines **PASCAL-XSC** Programms wie folgt zu deklarieren:

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   f := 100 / ( 1 + 100*sqr(x) );
end;
```

Mit Hilfe der vier Grundoperationen kann f aus allen Standardfunktionen der Tabelle von Seite 399 zusammengesetzt werden.

Die Eingangsparameter `a,b : Linterval` sind Einschließungen der Integrationsgrenzen $a, b \in I\!R$. Ist a im 21-stelligen `Longreal`–Format exakt darstellbar, so ist `a : Linterval` als Punktintervall zu wählen. Die nicht darstellbare Zahl $a = \sqrt{2}$ ist dagegen durch `a : Linterval` maximalgenau einzuschließen.

Die Eingangsgröße `abs_err : Longreal` gibt den absoluten Fehler vor, mit dem der Integralwert einzuschließen ist. Die Laufzeit wächst dabei mit kleiner werdenden `abs_err`–Werten deutlich an. Ein zu kleiner Wert führt zu einer Fehlermeldung (`Err=1`), wenn die internen Intervallüberschätzungen den vorgegebenen absoluten Fehler übersteigen. Mit geeignetem `abs_err`–Wert lassen sich bei nicht zu komplizierten Integranden $f(x)$ die Integrale I i.a. auf bis zu 18 dezimale Stellen genau einschließen.

Das Programm `romberg_ex1` liefert Einschließungen des Integrals

$$\begin{aligned} I &= \int_{-1}^{+1} \frac{100}{1 + 100 \cdot x^2}\, dx = 20 \cdot \arctan(10) \\ &= 29.4225534860746918370575114 35\ldots \end{aligned}$$

```
program romberg_ex1;
{ Berechnet Einschliessung eines Integrals einer hinreichend  }
{ oft differenzierbaren Funktion f.                          }
{ a,b : Linterval:  Einschliessungen der Integrationsgrenzen  }
{ abs_err        :  Vorgegebener absoluter Fehler            }
use li_ari,    { Intervall-Arithmetik zum Typ Longreal       }
    litaylor,  { Intervall-Taylor-Arithmetik zum Typ Longreal }
    iromberg;  { Exportiert function INTEGRAL(..) : Linterval }

function f(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   f := 100 / (1 + 100*sqr(x));
```

```
end; { f }

var a,b     : Linterval;
   abs_err : Longreal;
   Err     : integer;
   int     : Linterval;

begin
   write('Enclosure of the lower bound a = ?  ');  read(a);
   write('Enclosure of the upper bound b = ?  ');  read(b);
   repeat
      write('abs_err = ?  ');  read(abs_err);
      if abs_err > 0 then
      begin
         int := INTEGRAL(f,a,b,abs_err,Err);
         if Err=0 then
         begin
            writeln;
            writeln('   Enclosure of the integral:');
            writeln('   ',int);
         end      else writeln('Err = ',Err);
         writeln;
      end;
   until abs_err <= 0;
end.
```

Der Start des Programms `romberg_ex1` liefert mit den entsprechenden Eingaben das folgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  -1    <enter>
Enclosure of the upper bound b = ?  +1    <enter>
abs_err = ?  1e-7                         <enter>

   Enclosure of the integral:
   [  2.94225534828352472998E+001,  2.94225534890479094822E+001 ]

abs_err = ?  1e-17

   Enclosure of the integral:
   [  2.94225534860746918350E+001,  2.94225534860746918390E+001 ]

abs_err = ?  1e-18
Err = 1
```

13.2 Zwei kritische Integranden

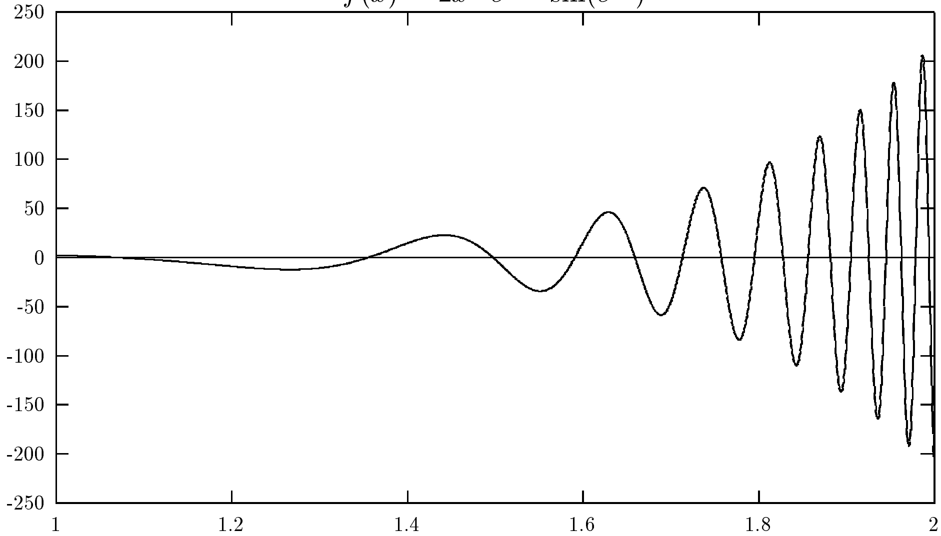
13.2.1 $\int_0^2 2x \cdot e^{x^2} \sin(e^{x^2})\, dx$

Einzuschließen ist das folgende Integral:

$$\begin{aligned} I &= \int_0^2 2x \cdot e^{x^2} \sin(e^{x^2})\, dx = \cos(1) - \cos(e^4) \\ &= 0.910964039265932\ldots \end{aligned}$$

Bei diesem Beispiel ist die numerische Integration besonders schwierig, weil der Integrand $f(x) = 2x \cdot e^{x^2} \sin(e^{x^2})$ zunächst bis $x = 1$ nahezu konstant ist, um dann mit immer stärkerer Frequenz und Amplitude zu oszillieren:

$f(x) = 2x \cdot e^{x^2} \cdot \sin(e^{x^2})$



Mit der Deklaration des Integranden

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
var ex : litaylor[lb(x)..ub(x)];
begin
   ex := exp( sqr(x) );
   f := 2*x*ex*sin(ex);
end;
```

liefert das Programm `romberg_ex1` mit den entsprechenden Eingaben das folgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0    <enter>
Enclosure of the upper bound b = ?  2    <enter>

abs_err = ?  1      <enter>

   Enclosure of the integral:
   [  8.35433599256593891616E-001,  9.82787621716232463250E-001 ]

abs_err = ?  1e-4   <enter>

   Enclosure of the integral:
   [  9.10954651182514259796E-001,  9.10974114537870807473E-001 ]

abs_err = ?  1e-7   <enter>

   Enclosure of the integral:
   [  9.10964024320257792792E-001,  9.10964053312297897370E-001 ]

abs_err = ?  1e-10  <enter>

   Enclosure of the integral:
   [  9.10964039251688759506E-001,  9.10964039277707197677E-001 ]

abs_err = ?  1e-12  <enter>

   Enclosure of the integral:
   [  9.10964039265855310540E-001,  9.10964039266007887053E-001 ]

abs_err = ?  1e-15  <enter>

   Enclosure of the integral:
   [  9.10964039265932797175E-001,  9.10964039265932862856E-001 ]

abs_err = ?  5e-17  <enter>

   Enclosure of the integral:
   [  9.10964039265932817099E-001,  9.10964039265932843878E-001 ]

abs_err = ?  1e-17  <enter>
Err = 1
```

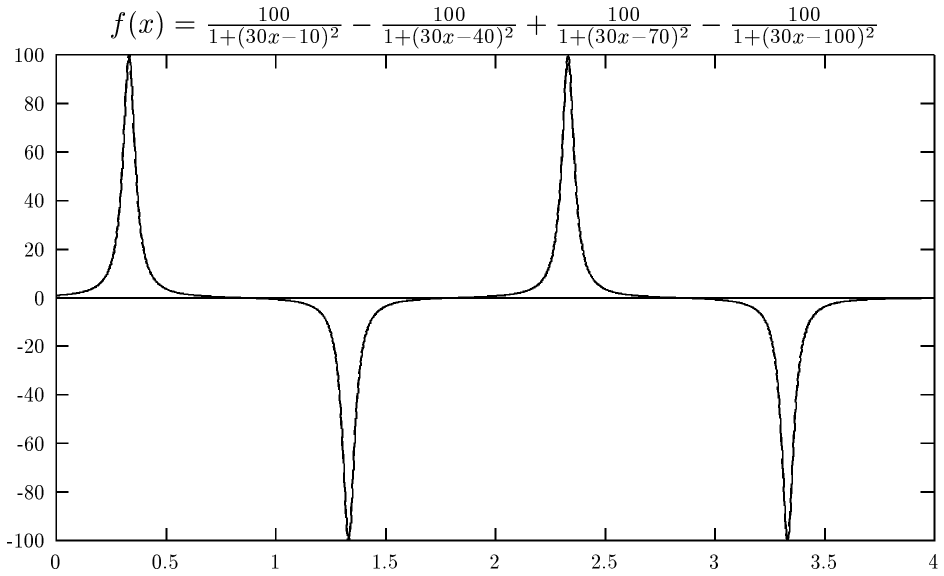
Der absolute Fehler `abs_err=1e-17` kann wegen `Err=1` nicht unterschritten werden. Das Integral läßt sich trotz der starken Oszillationen noch auf 16 Dezimalstellen genau einschließen!

13.2.2 Integration einer rationalen Funktion

Einzuschließen ist das Integral:

$$\begin{aligned} I &= \int_0^4 \left(\frac{100}{1+(30x-10)^2} - \frac{100}{1+(30x-40)^2} + \frac{100}{1+(30x-70)^2} - \frac{100}{1+(30x-100)^2}\right) dx \\ &= -0.151963942232930\ldots \end{aligned}$$

Die numerische Integration ist jetzt aufwendig, da der Integrand $f(x)$ vier scharfe Spitzen mit den ungefähren Werten ± 100 besitzt und dazwischen nur sehr kleine Werte annimmt:



Mit der Deklaration des Integranden

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   f := 100 * (  1/(1+sqr(30*x-10)) - 1/(1+sqr(30*x-40))
                + 1/(1+sqr(30*x-70)) - 1/(1+sqr(30*x-100)) );
end;
```

liefert das Programm `romberg_ex1` von Seite 552 mit den entsprechenden Eingaben das folgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0   <enter>
Enclosure of the upper bound b = ?  4   <enter>

abs_err = ?  1                          <enter>

   Enclosure of the integral:
   [ -2.81892156369937205810E-001,  6.52358838145221898000E-003 ]

abs_err = ?  1e-2   <enter>

   Enclosure of the integral:
   [ -1.53685354909239757550E-001, -1.50033774830206084940E-001 ]

abs_err = ?  1e-6   <enter>

   Enclosure of the integral:
   [ -1.51963960368937846360E-001, -1.51963923991103150030E-001 ]

abs_err = ?  1e-10  <enter>

   Enclosure of the integral:
   [ -1.51963942250340221080E-001, -1.51963942215430214080E-001 ]

abs_err = ?  1e-12  <enter>

   Enclosure of the integral:
   [ -1.51963942232962049220E-001, -1.51963942232899078630E-001 ]

abs_err = ?  1e-15  <enter>

   Enclosure of the integral:
   [ -1.51963942232930614990E-001, -1.51963942232930521120E-001 ]

abs_err = ?  1e-17  <enter>

   Enclosure of the integral:
   [ -1.51963942232930570470E-001, -1.51963942232930565900E-001 ]

abs_err = ?  1e-18  <enter>
Err = 1
```

Der absolute Fehler `abs_err=1e-18` kann wegen `Err=1` nicht unterschritten werden. Das Integral läßt sich trotz des extremen Funktionsverlaufes noch auf 16 Dezimalstellen genau einschließen!

13.3 Integrale mit Wurzelfunktionen

13.3.1 Problemstellung

Enthält ein Integrand $f(x)$ eine Wurzelfunktion, die über einem gegebenem Integrationsintervall mit dem Argument 0 auszuwerten ist, so treten zwei grundsätzliche Probleme auf:

1. Durch unvermeidbare Intervallüberschätzungen wird der Definitionsbereich der Wurzelfunktion überschritten, was zu einer entsprechenden Fehlermeldung führt.
2. Da die Wurzelfunktion im Ursprung zwar stetig aber nicht differenzierbar ist, kommt es bei der internen Restgliedabschätzung ebenfalls zu einer Fehlermeldung.

In vielen Fällen lassen sich diese Fehlermeldungen jedoch vermeiden, wenn man im Integral die entsprechende Wurzelfunktion geeignet substituiert. Mit $t(x) = \sqrt{x}$ erhält man z.B. die Beziehung:

$$\boxed{\int_0^b f(x, \sqrt{x})\, dx = 2 \int_0^{\sqrt{b}} t \cdot f(t^2, t)\, dt, \quad b > 0} \tag{13.2}$$

Die rechte Seite in (13.2) läßt sich dann problemlos integrieren, wenn $f(t^2, t)$ bzgl. t hinreichend oft differenzierbar ist.

13.3.2 Beispiele

13.3.2.1 $\int_0^2 e^{\sqrt{x}(1-x^2)}$

Nach (13.2) gilt: $\displaystyle\int_0^2 e^{\sqrt{x}(1-x^2)} = 2\int_0^{\sqrt{2}} x \cdot e^{x(1-x^4)} dx$ und mit der Deklaration des rechten Integranden

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
var v : litaylor[lb(x)..ub(x)];
begin
   v := sqr(x);
   f := 2*x*exp( x*(1-sqr(v)) );
end;
```

liefert das Programm `romberg_ex1` von Seite 552 das nachfolgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0                                  <enter>
Enclosure of the upper bound b = ?
                  [1.41421356237309504880,1.41421356237309504881]  <enter>
abs_err = ?  1e-18                                                 <enter>

   Enclosure of the integral:
   [  1.79935544216179925800E+000,  1.79935544216179925839E+000 ]

abs_err = ?  1e-19                                                 <enter>
Err = 1
```

Anmerkungen:

1. Die maximalgenaue Einschließung von $\sqrt{2}$ wird vom Modul `i_util` durch die Funktion `SQRT2_Li` im 21-stelligen `Longreal`–Format exportiert, vergl. die Tabelle auf Seite 640.
2. Das Integral I wird mit einer maximalen Genauigkeit von 19 Dezimalstellen eingeschlossen.

13.3.2.2 $\int_0^{100} x^{-1/2} e^{-x/(1+x^2)} dx$

Nach (13.2) gilt: $I = \int_0^{100} x^{-1/2} e^{-x/(1+x^2)} dx = 2 \int_0^{10} e^{-x^2/(1+x^4)} dx$ und mit der Deklaration des rechten, jetzt nicht mehr singulären Integranden

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
var v : litaylor[lb(x)..ub(x)];
begin
   v := sqr(x);
   f := 2*exp( -v/(1+sqr(v)) );
end;
```

liefert das Programm `romberg_ex1` von Seite 552 das nachfolgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0
Enclosure of the upper bound b = ?  10
abs_err = ?  1e-17

   Enclosure of the integral:
   [  1.82245135720656147662E+001,  1.82245135720656147695E+001 ]
```

13.3.2.3 $\int_0^1 x^{-2/3} e^{\sqrt{x}}\, dx$

Einzuschließen ist das uneigentliche Integral

$$I = \int\limits_0^1 \frac{e^{\sqrt{x}}}{x^{2/3}}\, dx$$

Für $x = 0$ verschwindet der Nenner $\sqrt[3]{x^2}$ und erzeugt damit eine Singularität des Integranden. Zusätzlich ist der Exponent $\sqrt{x}$ im Ursprung zwar stetig aber nicht differenzierbar, so daß die direkte numerische Integration mit Hilfe der Funktion `INTEGRAL` von Seite 551 zu einer entsprechenden Fehlermeldung führt. Die Substitution $u = x^{1/6}$ liefert jedoch die Beziehung

$$I = \int\limits_0^1 \frac{e^{\sqrt{x}}}{x^{2/3}}\, dx = 6 \int\limits_0^1 u \cdot e^{u^3}\, du$$

wobei der Integrand $f(u) = u \cdot e^{u^3}$ beliebig oft differenzierbar ist, so daß das Integral I jetzt mit `INTEGRAL(...)` problemlos eingeschlossen werden kann. Mit der Deklaration

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   f := 6*x*exp( x*sqr(x) );
end;
```

liefert das Programm `romberg_ex1` von Seite 552 das nachfolgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0    <enter>
Enclosure of the upper bound b = ?  1    <enter>
abs_err = ?  1e-4                        <enter>

   Enclosure of the integral:
   [  4.68718150752582962156E+000,  4.68718219266125792396E+000 ]

abs_err = ?  1e-18                       <enter>

   Enclosure of the integral:
   [  4.68718218665193549044E+000,  4.68718218665193549083E+000 ]

abs_err = ?  1e-19                       <enter>
Err = 1
```

Das Integral I wird also mit einer maximalen Genauigkeit von 19 Dezimalstellen erfolgreich eingeschlossen.

13.3.2.4 $\int_0^1 \sqrt{1-x^4}\,dx$

Einzuschließen ist das Integral

$$I = \int_0^1 \sqrt{1-x^4}\,dx$$

Wegen $f(x) = \sqrt{1-x^4} = \sqrt{(1-x)(1+x)(1+x^2)}$ liefert im Integranden der Faktor $\sqrt{1-x}$ bei der Integration eine Fehlermeldung, da $f(x)$ an der Stelle $x_0 = 1$ nicht differenzierbar ist. Mit der Substitution $u = \sqrt{1-x}$ erhält man das Ergebnis:

$$I = \int_0^1 \sqrt{1-x^4}\,dx = 2\int_0^1 x^2 \cdot \sqrt{(2-x^2)(1+(1-x^2)^2)}\,dx$$

Der Integrand des rechten Integrals ist jetzt über dem ganzen Intervall $[0,1]$ hinreichend oft differenzierbar, so daß die Integration jetzt mit Hilfe der Funktion `INTEGRAL(...)` von Seite 551 problemlos erfolgen kann. Mit der Deklaration

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
var v : litaylor[lb(x)..ub(x)];
begin
   v := sqr(x);
   f := 2*v*sqrt((2-v)*(1+sqr(1-v)));
end;
```

liefert das Programm `romberg_ex1` von Seite 552 das nachfolgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0    <enter>
Enclosure of the upper bound b = ?  1    <enter>
abs_err = ?  1e-19

   Enclosure of the integral:
   [  8.74019184764039936772E-001,  8.74019184764039936872E-001 ]

abs_err = ?  1e-20                       <enter>
Err = 1
```

Das Integral I wird also mit einer maximalen Genauigkeit von 18 Dezimalstellen eingeschlossen.

13.3.2.5 $\int_0^\pi e^{\sqrt{1+\cos(x)}}\,dx$

Einzuschließen ist das Integral

$$I = \int_0^\pi e^{\sqrt{1+\cos(x)}}\,dx = 8.42934815865039883764242 0\ldots$$

Da der Exponent $\sqrt{1+\cos(x)}$ an der oberen Integrationsgrenze $x = \pi$ verschwindet, ist der Integrand im ganzen Integrationsgebiet nicht hinreichend oft differenzierbar, so daß die direkte numerische Integration mit Hilfe der Funktion `INTEGRAL(...)` von Seite 551 nicht funktionieren kann. Mit der Substitution $u = \cos(x)$ erhält man zunächst:

$$I = \int_0^\pi e^{\sqrt{1+\cos(x)}}\,dx = \int_{-1}^{+1} e^{\sqrt{1+u}} \cdot \frac{du}{\sqrt{1-u^2}}$$

wodurch der neue Integrand jetzt an beiden Integrationsgrenzen ± 1 sogar singulär geworden ist. Diese Singularitäten lassen sich jedoch beseitigen, wenn man das Integral I zunächst in zwei Summanden aufspaltet und für jeden Summanden eine geeignete Substitution durchführt:

$$I_1 := \int_{-1}^{0} e^{\sqrt{1+u}} \cdot \frac{du}{\sqrt{1-u^2}}, \quad I_2 := \int_{0}^{+1} e^{\sqrt{1+u}} \cdot \frac{du}{\sqrt{1-u^2}}, \qquad I = I_1 + I_2;$$

$$t = \sqrt{1+u}: \quad I_1 = 2\int_0^1 \frac{e^t dt}{\sqrt{2-t^2}}; \qquad t = \sqrt{1-u}: \quad I_2 = 2\int_0^1 \frac{e^{\sqrt{2-t^2}}}{\sqrt{2-t^2}}\,dt,$$

$$\longrightarrow \quad I = \int_0^\pi e^{\sqrt{1+\cos(x)}}\,dx = 2\int_0^1 \frac{e^x + e^{\sqrt{2-x^2}}}{\sqrt{2-x^2}}\,dx,$$

wobei der Integrand rechts in $[0,1]$ beliebig oft differenzierbar ist, so daß das ursprüngliche Integral I jetzt mit Hilfe der Funktion `INTEGRAL(...)` problemlos eingeschlossen werden kann. Mit der Deklaration

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
var v : litaylor[lb(x)..ub(x)];
begin
   v := sqrt( 2-sqr(x) );
   f := 2*( exp(x)+exp(v) ) / v;
end;
```

liefert das Programm `romberg_ex1` von Seite 552 das nachfolgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0
Enclosure of the upper bound b = ?  1
abs_err = ?  1e-4

   Enclosure of the integral:
   [  8.4293202780105216996 2E+000,  8.4293588430956151279 3E+000 ]

abs_err = ?  1e-10

   Enclosure of the integral:
   [  8.4293481586456123284 9E+000,  8.4293481586516778610 3E+000 ]

abs_err = ?  1e-18

   Enclosure of the integral:
   [  8.4293481586503988370 8E+000,  8.4293481586503988380 5E+000 ]

abs_err = ?  1e-19
Err = 1
```

Anmerkungen:

1. Das Integral I wird also bei Vorgabe eines absoluten Höchstfehlers von 10^{-18} mit einer maximalen Genauigkeit von 18 Dezimalstellen erfolgreich eingeschlossen.

2. Der relative Höchstfehler berechnet sich aus der obigen Einschließung zu: $\varepsilon_{rel} = 1.153 \cdot 10^{-19}$.

3. Das obige Beispiel zeigt, daß u.U. auch mehrere gezielte Transformationen notwendig sind, wenn im Integrationsbereich die Voraussetzung der Differenzierbar des Integranden nicht von vornherein gegeben ist. Beachten Sie aber, daß dann bei einem hinreichend oft differenzierbaren Integranden die mit Hilfe der Funktion `INTEGRAL` berechnete Einschließung den gesuchten Integralwert I **garantiert** enthält! Der interne Algorithmus wertet nämlich den Integranden nicht nur an einzelnen Stützstellen aus sondern erfaßt mit Hilfe der benutzten Intervall–Arithmetik die Funktionswerte über dem ganzen Integrationsintervall.

13.4 Integrale mit Logarithmus-Funktionen

13.4.1 Problemstellung

Enthält ein Integrand $f(x)$ eine Logarithmus-Funktion, die über einem gegebenem Integrationsintervall mit dem Argument 0 auszuwerten ist, so treten zwei grundsätzliche Probleme auf:

1. Der Integrand $f(x)$ besitzt eine Singularität, so daß eine direkte Auswertung des Integrals mit Hilfe der Funktion `INTEGRAL` von Seite 551 zu einer Fehlermeldung führt.
2. Wenn die Singularität durch eine geeignete Substitution beseitigt werden kann, so wird dabei i.a. ein ursprünglich endliches Integrationsintervall in einen unendlichen Integrationsbereich transformiert.

Ein solches uneigentliches Integral $\int_0^\infty g(x)\,dx$ mit hinreichend oft differenzierbarem Integranden $g(x)$ ist dann in zwei Summanden aufzuspalten:

$$\int_0^\infty g(x)\,dx = I_1 + I_2, \quad \text{mit:} \quad I_1 := \int_0^N g(x)\,dx, \quad I_2 := \int_N^\infty g(x)\,dx, \quad N \in \mathbb{N},$$

wobei I_1 den Hauptteil des uneigentlichen Integrals darstellt und I_2 durch geeignete Wahl von N hinreichend klein zu wählen ist. Mit Hilfe der Funktion `INTEGRAL` von Seite 551 läßt sich dann eine Einschließung E_1 für $I_1 \in E_1$ problemlos berechnen. Um eine entsprechende Einschließung E_2 auch für $I_2 \in E_2$ zu erhalten, muß das Integral I_2 nach oben und unten abgeschätzt werden:

$$S_1 \leq \int_N^\infty g(x)\,dx \leq S_2, \qquad \longrightarrow \qquad I_2 \in E_2 := [S_1, S_2]$$

Für das uneigentliche Integral $\int_0^\infty g(x)\,dx$ ergibt sich daraus die gesuchte Einschließung:

$$\int_0^\infty g(x)\,dx \in E_1 + E_2,$$

wobei die Addition rechts natürlich als Intervall–Addition zu interpretieren ist. Entsprechende Beispiele findet man in den nachfolgenden Abschnitten.

13.4.2 Beispiele

13.4.2.1 $\int_0^{\pi/4} \ln(\tan(x))\, dx$

Einzuschließen ist das uneigentliche Integral

$$I = \int_0^{\pi/4} \ln(\tan(x))\, dx$$

dessen Integrand im Randpunkt 0 singulär ist, so daß eine Einschließung von I mit Hilfe der Funktion `INTEGRAL` von Seite 551 auf direktem Wege nicht möglich ist. Zur Beseitigung der Singularität führen wir die folgende Substitution durch:

$$u(x) = -\ln(\tan(x)) \quad \longleftrightarrow \quad e^{-u} = \tan(x)$$

$$\frac{du}{dx} = \frac{-1}{\tan(x)} \cdot \frac{1}{\cos^2(x)} = \frac{-1}{\tan(x)} \cdot \left(1 + \frac{1}{\tan^2(x)}\right) = -\frac{\tan^2(x)+1}{\tan^3(x)}$$

$$\longrightarrow \qquad I = \int_0^{\pi/4} \ln(\tan(x))\, dx = -\int_0^{\infty} \frac{x \cdot e^{-x}}{1+e^{2x}}\, dx,$$

wobei jetzt der rechte Integrand $g(x) = x \cdot e^{-x}/(1+e^{2x})$ im ganzen Integrationsbereich beliebig oft differenzierbar ist. Wir führen jetzt die auf Seite 564 beschriebene Aufspaltung des Integrals I durch:

$$I_1 := -\int_0^{17} \frac{x \cdot e^{-x}}{1+e^{2x}}\, dx, \quad I_2 := -\int_{17}^{\infty} \frac{x \cdot e^{-x}}{1+e^{2x}}\, dx, \quad \int_0^{\pi/4} \ln(\tan(x))\, dx = I_1 + I_2$$

Das Integral I_1 kann jetzt mit Hilfe der Funktion `INTEGRAL` von Seite 551 durch ein Intervall $E_1 \ni I_1$ problemlos eingeschlossen werden. Um eine Einschließung E_2 auch für das Integral I_2 zu erhalten, wird jetzt das Integral $\int_N^{\infty} x \cdot e^{-x}/(1+e^{2x})\, dx$ für $N > 1$ abgeschätzt:

$$\int_N^{\infty} x \cdot \frac{e^{-x}}{1+e^{2x}}\, dx \;<\; \int_N^{\infty} x \cdot e^{-3x}\, dx = \frac{e^{-3N}}{3}\left[N + \frac{1}{3}\right]$$

$$\int_N^{\infty} x \cdot \frac{e^{-x}}{1+e^{2x}}\, dx \;>\; \int_N^{\infty} \frac{1}{2} \cdot e^{-3x}\, dx = \frac{1}{6} \cdot e^{-3N}$$

Daraus ergibt sich für das Integral I_2 die Einschließung:

$$I_2 = -\int_{17}^{\infty} \frac{x \cdot e^{-x}}{1+e^{2x}}\,dx \in -e^{-51}\left[\frac{1}{6}, \frac{52}{9}\right] =: E_2 \qquad \longrightarrow$$

$$I = \int_0^{\pi/4} \ln(\tan(x))\,dx = -\int_0^{\infty} \frac{x \cdot e^{-x}}{1+e^{2x}}\,dx = I_1 + I_2 \in E_1 + E_2$$

Das nachfolgende Programm `romberg_ex2` berechnet zu vorgegebenem absoluten Fehler `abs_err` die Intervallsumme $E_1 + E_2$ und damit eine garantierte Einschließung des gesuchten Integrals I.

```
program romberg_ex2;
{ Berechnung einer Einschliessung des Integrals              }
{             I = INT_0^(Pi/4)ln(tan(x))dx                   }
{ abs_err      :  Einzugebener absoluter Fehler              }
use li_ari,    { Intervall-Arithmetik zum Typ Longreal       }
    litaylor,  { Intervall-Taylor-Arithmetik zum Typ Longreal }
    iromberg;  { Exportiert function INTEGRAL(..) : Linterval }

const stars = '*****************';

function f(x : litaylor) : litaylor[lb(x)..ub(x)];
var v : litaylor[lb(x)..ub(x)];
begin
   v := exp(x);
   f := -x/( v*(1+sqr(v)) );
end; { f }

var a,b    : Linterval;
   abs_err : Longreal;
   Err     : integer;
   E1,E2   : Linterval;

begin
   a := 0;  b := 17;                   { Integrationsgrenzen }
   write  ('  ***********************************************');
   writeln(stars);
   write  ('  *  Einschliessung des Integrals');
```

```
   writeln('  I = INT_0^(Pi/4)ln(tan(x))dx  *');
   write  ('  ********************************************');
   writeln(stars);
   writeln;
   repeat
      write('abs_err = ?  ');  read(abs_err);
      if abs_err > 0 then
      begin
         E1 := INTEGRAL(f,a,b,abs_err,Err); { Einschlg. v. I1 }
         if Err=0 then
         begin  { Einschliessung von I2: }
            E2 := -exp(Lintval(-51))*Lintval(1/<6,52/>9);
            writeln;
            write('   Enclosure of the integral ');
            writeln('I = INT_0^(Pi/4) ln( tan(x) )dx:');
            writeln('   ',E1+E2);
         end      else writeln('Err = ',Err);
         writeln;
      end;
   until abs_err <= 0;
end.
```

Bei Eingabe des jeweiligen absoluten Fehlers abs_err liefert das obige Programm das nachfolgende Ablaufprotokoll:

```
      *************************************************************
      *  Einschliessung des Integrals  I = INT_0^(Pi/4)ln(tan(x))dx  *
      *************************************************************

    abs_err = ?  1e-10   <enter>

       Enclosure of the integral I = INT_0^(Pi/4) ln( tan(x) )dx:
       [ -8.403440583144322171 42E-002, -8.40344058079267017870E-002 ]

    abs_err = ?  5e-20   <enter>

       Enclosure of the integral I = INT_0^(Pi/4) ln( tan(x) )dx:
       [ -8.40344058227809849558E-002, -8.40344058227809849349E-002 ]

    abs_err = ?  1e-20   <enter>
    Err = 1
```

Die Einschließung des Integrals $I \in E_1 + E_2$ wird also mit einem relativen Höchstfehler $\varepsilon_{rel} = 2.393 \cdot 10^{-19}$ in hoher Genauigkeit berechnet.

13.4.2.2 $\int_0^1 e^x \cdot \ln(x)\, dx$

Einzuschließen ist das uneigentliche Integral

$$I = \int_0^1 e^x \cdot \ln(x)\, dx$$

dessen Integrand im Randpunkt 0 singulär ist, so daß eine Einschließung von I mit Hilfe der Funktion `INTEGRAL` von Seite 551 auf direktem Wege nicht möglich ist. Zur Beseitigung der Singularität führen wir die folgende Substitution durch:

$$u(x) = -\ln(x) \quad \longleftrightarrow \quad e^{-u} = x, \quad \frac{du}{dx} = \frac{-1}{x}$$

$$\longrightarrow \qquad I = \int_0^1 e^x \cdot \ln(x)\, dx = -\int_0^\infty x \cdot e^{-x+e^{-x}}\, dx,$$

wobei jetzt der rechte Integrand $g(x) = x \cdot e^{-x+e^{-x}}$ im ganzen Integrationsbereich beliebig oft differenzierbar ist. Wir führen jetzt die auf Seite 564 beschriebene Aufspaltung des Integrals I durch:

$$I_1 := -\int_0^{55} x \cdot e^{-x+e^{-x}}\, dx, \; I_2 := -\int_{55}^\infty x \cdot e^{-x+e^{-x}}\, dx, \; \int_0^1 e^x \cdot \ln(x)\, dx = I_1 + I_2$$

Das Integral I_1 kann jetzt mit Hilfe der Funktion `INTEGRAL` von Seite 551 durch ein Intervall $E_1 \ni I_1$ problemlos eingeschlossen werden. Um eine Einschließung E_2 auch für das Integral I_2 zu erhalten, wird im nächsten Schritt das Integral $\int_N^\infty x \cdot e^{-x+e^{-x}}\, dx$ für $N > 1$ abgeschätzt:

$$\begin{aligned}
\int_N^\infty x \cdot e^{-x+e^{-x}}\, dx &\leq e^{e^{-N}} \int_N^\infty x \cdot e^{-x}\, dx = e^{e^{-N}} \cdot (N+1) \cdot e^{-N} \\
\int_N^\infty x \cdot e^{-x+e^{-x}}\, dx &\geq \int_N^\infty x \cdot e^{-x}\, dx = (N+1) \cdot e^{-N}
\end{aligned}$$

Daraus ergibt sich für das Integral I_2 die Einschließung:

$$I_2 = -\int_{55}^{\infty} x \cdot e^{-x+e^{-x}}\,dx \in -56 \cdot e^{-55}\left[1, e^{e^{-55}}\right] =: E_2 \qquad \longrightarrow$$

$$I = \int_0^1 e^x \cdot \ln(x)\,dx = -\int_0^{\infty} x \cdot e^{-x+e^{-x}}\,dx = I_1 + I_2 \in E_1 + E_2$$

Das nachfolgende Programm `romberg_ex3` berechnet zu vorgegebenem absoluten Fehler `abs_err` die Intervallsumme $E_1 + E_2$ und damit eine garantierte Einschließung des gesuchten Integrals I.

```
program romberg_ex3;
{ Berechnung einer Einschliessung des Integrals              }
{                 I = INT_0^1 e^x*ln(x)dx                    }
{ abs_err      :  Einzugebener absoluter Fehler              }
use li_ari,    { Intervall-Arithmetik zum Typ Longreal       }
    litaylor,  { Intervall-Taylor-Arithmetik zum Typ Longreal }
    iromberg;  { Exportiert function INTEGRAL(..) : Linterval }

const stars = '****************';

function f(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   f := -x*exp( -x+exp(-x) );
end; { f }

var a,b    : Linterval;
   abs_err : Longreal;
   Err     : integer;
   E1,E2,u : Linterval;

begin
   a := 0;  b := 55;                   { Integrationsgrenzen }
   write  ('  *********************************************');
   writeln(stars);
   write  ('  *  Einschliessung des Integrals');
   writeln('  I = INT_0^1 exp(x)*ln(x)dx  *');
   write  ('  *********************************************');
   writeln(stars);
```

```
   writeln;
   repeat
      write('abs_err = ?  ');  read(abs_err);
      if abs_err > 0 then
      begin
         E1 := INTEGRAL(f,a,b,abs_err,Err); { Einschlg. v. I1 }
         if Err=0 then
         begin  { Einschliessung von I2: }
            u := exp( exp(Lintval(-55)) );
            E2 := -56 * exp( Lintval(-55) ) * Lintval(1,u.sup);
            writeln;
            write('   Enclosure of the integral ');
            writeln('I = INT_0^1 exp(x)*ln(x)dx:');
            writeln('    ',E1+E2);
         end      else writeln('Err = ',Err);
         writeln;
      end;
   until abs_err <= 0;
end.
```

Bei Eingabe des jeweiligen absoluten Fehlers abs_err liefert das obige Programm das nachfolgende Ablaufprotokoll:

```
      ***********************************************************
      *  Einschliessung des Integrals  I = INT_0^1 exp(x)*ln(x)dx  *
      ***********************************************************

    abs_err = ?  1e-10   <enter>

       Enclosure of the integral I = INT_0^1 exp(x)*ln(x)dx:
       [ -1.3179021514619478369lE+000, -1.3179021514392796066 4E+000 ]

    abs_err = ?  1e-18   <enter>

       Enclosure of the integral I = INT_0^1 exp(x)*ln(x)dx:
       [ -1.3179021514544038949 8E+000, -1.3179021514544038947 3E+000 ]

    abs_err = ?  1e-19   <enter>
    Err = 1
```

Die Einschließung des Integrals $I \in E_1 + E_2$ wird also mit einem relativen Höchstfehler $\varepsilon_{rel} = 1.897 \cdot 10^{-19}$ in hoher Genauigkeit berechnet.

13.5 Integrale mit Arcussinus-Funktionen

13.5.1 Problemstellung

Enthält ein Integrand $f(x)$ eine Arcussinus-Funktion, die auf einem der Randpunkte ± 1 ihres Definitionsbereiches auszuwerten ist, so ist $f(x)$ in diesen Randpunkten i.a. nicht differenzierbar, so daß eine direkte Auswertung des Integrals mit Hilfe der Funktion `INTEGRAL` von Seite 551 zu einer Fehlermeldung führt.

In einigen Fällen kann man mit Hilfe einer geeigneten Substitution dafür sorgen, daß der neue Integrand $g(x)$ im ganzen Integrationsbereich hinreichend oft differenzierbar ist, womit dann der Integralwert wieder problemlos eingeschlossen werden kann.

Wenn dies jedoch nicht gelingt, so läßt sich das Integral I oft in zwei Summanden $I = I_1 + I_2$ so zerlegen, daß I_1 den Hauptteil von I darstellt und mit Hilfe der Funktion `INTEGRAL` garantiert eingeschlossen werden kann, wenn der Integrand $f(x)$ über dem Integrationsbereich von I_1 hinreichend oft differenzierbar ist. Der Betrag des zweiten Integrals I_2, das den nicht differenzierbaren Punkt enthält, läßt sich dann mit einer um so kleineren Oberschranke abschätzen, je schmaler der Integrationsbereich gewählt wird. Sind dann E_1 und E_2 die Einschließungen von I_1 und I_2, so gilt $I \in E_1 + E_2$.

In den folgenden Abschnitten findet man entsprechende Beispiele zu den oben diskutierten Fällen.

13.5.2 Beispiele

13.5.2.1 $\int_{-1}^{1} \arcsin(x) \cdot \frac{x}{1+x^2}\, dx$

Einzuschließen ist das Integral

$$I = \int_{-1}^{1} \frac{x}{1+x^2} \cdot \arcsin(x)\, dx$$

dessen Integrand $f(x) = \frac{x}{1+x^2} \cdot \arcsin(x)$ in den Randpunkten $x = \pm 1$ nicht differenzierbar ist. Mit Hilfe der Substitution $u = \arcsin(x)$ erhält man

$$I = \int_{-1}^{1} \frac{x}{1+x^2} \cdot \arcsin(x)\, dx = \int_{0}^{+\pi/2} \frac{u \cdot \sin(2u)}{1+\sin^2(u)}\, du$$

wobei jetzt der Integrand $g(u)$ des zweiten Integrals über dem ganzen Integrationsbereich beliebig oft differenzierbar ist.

Mit der Deklaration

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   f := x*sin(2*x)/(1+sqr(sin(x)));
end;
```

liefert das Programm `romberg_ex1` von Seite 552 das nachfolgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0                          <enter>
Enclosure of the upper bound b = ?
        [1.57079632679489661923,1.57079632679489661924]    <enter>

abs_err = ?  1e-19                                         <enter>

   Enclosure of the integral:
   [  4.97462349406722478032E-001,  4.97462349406722478126E-001 ]

abs_err = ?  1e-20                                         <enter>
Err = 1
```

Danach wird das Integral I bei Vorgabe des kleinstmöglichen absoluten Fehlers `abs_err`$=10^{-19}$ mit einem relativen Höchstfehler $\varepsilon_{rel} = 1.89 \cdot 10^{-19}$ in hoher Genauigkeit eingeschlossen.

13.5.2.2 $\int_0^1 \sqrt{1-x} \cdot e^{x \cdot \arcsin(x)}\, dx$

Einzuschließen ist das Integral

$$I = \int\limits_0^1 \sqrt{1-x} \cdot e^{x \cdot \arcsin(x)}\, dx$$

dessen Integrand $f(x) = \sqrt{1-x} \cdot e^{x \cdot \arcsin(x)}$ im Randpunkt $x = 1$ nicht differenzierbar ist. Mit Hilfe der Substitution $u = \arcsin(x)$ erhält man

$$I = \int\limits_0^1 \frac{\sqrt{1-x^2}}{\sqrt{1+x}} \cdot e^{x \cdot \arcsin(x)}\, dx = \int\limits_0^{\pi/2} \frac{\cos^2(u)}{\sqrt{1+\sin(u)}} \cdot e^{u \cdot \sin(u)}\, du$$

Da der Integrand $g(u)$ des rechten Integrals jetzt im ganzen Integrationsbereich beliebig oft differenzierbar ist, kann I mit Hilfe der Funktion `INTEGRAL` von Seite 551 problemlos eingeschlossen werden.

Mit der Deklaration

```
function f(x : litaylor) : litaylor[lb(x)..ub(x)];
var v : litaylor[lb(x)..ub(x)];
begin
   v := sin(x);
   f := sqr(cos(x))*exp(x*v) / sqrt(1+v);
end;
```

liefert das Programm `romberg_ex1` von Seite 552 das nachfolgende Ablaufprotokoll:

```
Enclosure of the lower bound a = ?  0                          <enter>
Enclosure of the upper bound b = ?
       [1.57079632679489661923,1.57079632679489661924]  <enter>
abs_err = ?  1e-18                                             <enter>

   Enclosure of the integral:
   [  9.06992221489140030162E-001,  9.06992221489140030313E-001 ]

abs_err = ?  1e-19                                             <enter>
Err = 1
```

Danach wird das Integral I bei Vorgabe des kleinstmöglichen absoluten Fehlers `abs_err`$=10^{-18}$ mit einem relativen Höchstfehler $\varepsilon_{rel} = 1.67 \cdot 10^{-19}$ in hoher Genauigkeit eingeschlossen.

13.5.2.3 $\int_0^1 e^{x \cdot \arcsin(x)} \cdot \sqrt[3]{1-x}\, dx$

Einzuschließen ist das Integral

$$I = \int_0^1 \sqrt[3]{1-x} \cdot e^{x \cdot \arcsin(x)}\, dx = 1.06741737032606046674027104\ldots$$

dessen Integrand $f(x) = \sqrt[3]{1-x} \cdot e^{x \cdot \arcsin(x)}$ im Randpunkt $x = 1$ nicht differenzierbar ist, so daß eine direkte Einschließung von I mit Hilfe der Funktion `INTEGRAL` von Seite 551 nicht direkt möglich ist. Bei den beiden letzten Beispielen war es in dieser Situation möglich, durch eine geeignete Substitution den neuen Integranden $g(u)$ im ganzen Integrationsbereich hinreichend oft differenzierbar zu machen. Dies gelingt bei dem jetzt vorliegenden Beispiel jedoch nicht, da sowohl $\arcsin(x)$ als auch $\sqrt[3]{1-x}$ an der gleichen Stelle

$x = 1$ nicht differenzierbar sind. Mit der Substitution $u = \sqrt[3]{1-x}$ erreicht man lediglich, daß die Wurzelfunktion verschwindet:

$$I = \int_0^1 \sqrt[3]{1-x} \cdot e^{x\cdot\arcsin(x)}\, dx = 3\int_0^1 u^3 e^{(1-u^3)\cdot\arcsin(1-u^3)}\, du$$

Im letzten Integranden ist jetzt nur noch die $\arcsin(1-u^3)$–Funktion für $u = 0$ nicht differenzierbar. Nach der auf Seite 571 beschriebenen Integralaufspaltung

$$I_2 := 3\int_0^{\varepsilon} u^3 e^{(1-u^3)\cdot\arcsin(1-u^3)}\, du, \quad I_1 := 3\int_{\varepsilon}^1 u^3 e^{(1-u^3)\cdot\arcsin(1-u^3)}\, du$$

$$I = \int_0^1 \sqrt[3]{1-x} \cdot e^{x\cdot\arcsin(x)}\, dx = I_1 + I_2$$

läßt sich jetzt I_1 mit Hilfe der Funktion `INTEGRAL` für $\varepsilon = 10^{-6}$ durch $E_1 \ni I_1$ problemlos einschließen und I_2 kann wie folgt abgeschätzt werden:

$$I_2(\varepsilon) = 3\int_0^{\varepsilon} u^3 e^{(1-u^3)\cdot\arcsin(1-u^3)}\, du \le 3\cdot e^{\pi/2}\cdot \int_0^{\varepsilon} u^3\, du = \frac{3}{4}e^{\pi/2}\cdot\varepsilon^4$$

$$\varepsilon = 10^{-6} \quad \longrightarrow \quad I_2 \in E_2 := [0, 3.61\cdot 10^{-24}];$$

Das nachfolgende Programm `romberg_ex4` berechnet zu vorgegebenem absoluten Fehler `abs_err` die Intervallsumme $E_1 + E_2$ und damit eine garantierte Einschließung des gesuchten Integrals I.

```
program romberg_ex4;
{ Berechnung einer Einschliessung des Integrals             }
{        I = INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx         }
{ abs_err      :  Einzugebener absoluter Fehler             }
use li_ari,    { Intervall-Arithmetik zum Typ Longreal       }
    litaylor,  { Intervall-Taylor-Arithmetik zum Typ Longreal }
    iromberg;  { Exportiert function INTEGRAL(..) : Linterval }

const stars = '**********************';

function f(x : litaylor) : litaylor[lb(x)..ub(x)];
```

```
var v1,v2 : litaylor[lb(x)..ub(x)];
begin
   v1 := x*sqr(x);  v2 := 1-v1;
   f := 3*v1*exp(v2*arcsin(v2));
end; { f }

var a,b     : Linterval;
   abs_err : Longreal;
   Err     : integer;
   E1,E2   : Linterval;

begin
   a := 1e-6;  b := 1;                  { Integrationsgrenzen }
   write  ('  ********************************************');
   writeln(stars);
   write  ('  *  Einschliessung von ');
   writeln('  I = INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx  *');
   write  ('  ********************************************');
   writeln(stars);
   writeln;
   repeat
      write('abs_err = ?  ');  read(abs_err);
      if abs_err > 0 then
      begin
         E1 := INTEGRAL(f,a,b,abs_err,Err); { Einschlg. v. I1 }
         if Err=0 then
         begin  { Einschliessung von I2: }
            E2 := Lintval(0,3.61e-24);
            writeln;
            write('   Enclosure of the integral I = ');
            writeln('INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx:');
            writeln('   ',E1+E2);
         end      else writeln('Err = ',Err);
         writeln;
      end;
   until abs_err <= 0;
end.
```

Mit den entsprechenden Eingaben für den absoluten Fehler abs_err liefert obiges Programm das nachfolgende Ablaufprotokoll:

```
 ********************************************************************
 *  Einschliessung von  I = INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx  *
 ********************************************************************

abs_err = ?  1e-2   <enter>

   Enclosure of the integral I = INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx:
   [  1.06378199058607372320E+000,  1.07009947680934751295E+000 ]

abs_err = ?  1e-8   <enter>

   Enclosure of the integral I = INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx:
   [  1.06741736965680781382E+000,  1.06741737080229196546E+000 ]

abs_err = ?  1e-14  <enter>

   Enclosure of the integral I = INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx:
   [  1.06741737032606023207E+000,  1.06741737032606067211E+000 ]

abs_err = ?  1e-18  <enter>

   Enclosure of the integral I = INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx:
   [  1.06741737032606046663E+000,  1.06741737032606046686E+000 ]

abs_err = ?  2e-19  <enter>

   Enclosure of the integral I = INT_0^1 sqrt[3](1-x)*exp(x*arcsin(x)dx:
   [  1.06741737032606046666E+000,  1.06741737032606046682E+000 ]

abs_err = ?  1e-19  <enter>
Err = 1
```

Danach wird das Integral I bei Vorgabe des kleinstmöglichen absoluten Fehlers `abs_err=`$2 \cdot 10^{-19}$ mit einem relativen Höchstfehler $\varepsilon_{rel} = 1.5 \cdot 10^{-19}$ in hoher Genauigkeit eingeschlossen.

13.6 Einschließung der Fresnel-Integrale

Die Fresnel–Integrale sind für $x \in I\!R$ definiert durch:

$$(13.3) \qquad C(x) := \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\cos(t)}{\sqrt{t}}\, dt = \sqrt{\frac{2}{\pi}} \int_0^{\sqrt{x}} \cos(t^2)\, dt$$

$$(13.4) \qquad S(x) := \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\sin(t)}{\sqrt{t}}\, dt = \sqrt{\frac{2}{\pi}} \int_0^{\sqrt{x}} \sin(t^2)\, dt$$

Da die Integranden $\cos(t^2)$, $\sin(t^2)$ beliebig oft differenzierbar sind, können die Funktionswerte $C(x)$, $S(x)$ mit Hilfe der Funktion `INTEGRAL` von Seite 551 direkt eingeschlossen werden. Das nachfolgende Programm `romberg_ex5` liefert zu gegebenem Funktionsargument x und nach Vorgabe eines absoluten Höchstfehlers `abs_err` garantierte Einschließungen von $C(x)$ und $S(x)$.

```
program romberg_ex5;
{ Berechnung einer Einschliessung der Fresnel-Integrale       }
{        C(x) = (2pi)-0.5 INT_0^x cos(t)/sqrt(t)dt            }
{        S(x) = (2pi)-0.5 INT_0^x sin(t)/sqrt(t)dt            }
{ x            :  Einzugebenes reelles Funktionsargument      }
{ abs_err      :  Einzugebener absoluter Fehler               }
use li_ari,    { Intervall-Arithmetik zum Typ Longreal        }
    i_util,    { exportiert  R_SQRT2PI_Li = 1/sqrt(2Pi)       }
    litaylor,  { Intervall-Taylor-Arithmetik zum Typ Longreal }
    iromberg;  { Exportiert function INTEGRAL(..) : Linterval }
const stars = '*********';
function c(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   c := 2*R_SQRT2Pi_Li * cos( sqr(x) );
end; { c }
function s(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   s := 2*R_SQRT2Pi_Li * sin( sqr(x) );
end; { s }

var a,b     : Linterval;
   abs_err : Longreal;
   Err     : integer;
```

```
   C1,S1   : Linterval;

begin
   write  ('  *******************************************');
   writeln(stars);
   write  ('  *  Enclosures of the Fresnel integrals ');
   writeln('C(x) und S(x)  *');
   write  ('  *******************************************');
   writeln(stars);
   writeln;
   a := 0;                        { Untere Integrationsgrenze }
   repeat
      writeln('  Function argument x = ?');  read(b);
      b := sqrt(b);
      repeat
         write('abs_err = ?  ');  read(abs_err);
         if abs_err > 0 then
         begin
            C1 := INTEGRAL(c,a,b,abs_err,Err);
            if Err=0 then
            begin  { Enclosure of  C(x): }
               writeln;
               write  ('   Enclosure of the Fresnel ');
               writeln('integral C(x):');
               writeln('   ',C1);
            end      else writeln('Err = ',Err);
            writeln;
            S1 := INTEGRAL(s,a,b,abs_err,Err);
            if Err=0 then
            begin  { Enclosure of S(x): }
               writeln;
               write  ('   Enclosure of the Fresnel ');
               writeln('integral S(x):');
               writeln('   ',S1);
            end      else writeln('Err = ',Err);
            writeln;
         end;
      until abs_err <= 0;
   until false;
end.
```

Mit den Eingaben von x und `abs_err` liefert obiges Programm das folgende Ablaufprotokoll:

```
    *******************************************************
    *  Enclosures of the Fresnel integrals C(x) und S(x)  *
    *******************************************************

    Function argument x = ?
  0.5                       <enter>
  abs_err = ?  3e-20        <enter>

     Enclosure of the Fresnel integral C(x):
     [  5.50247154645006372065E-001,  5.50247154645006372094E-001 ]


     Enclosure of the Fresnel integral S(x):
     [  9.23657602098141562335E-002,  9.23657602098141562428E-002 ]

  abs_err = ?  0            <enter>
    Function argument x = ?
  1                         <enter>
  abs_err = ?  4e-20        <enter>

     Enclosure of the Fresnel integral C(x):
     [  7.21705924292605087752E-001,  7.21705924292605087786E-001 ]


     Enclosure of the Fresnel integral S(x):
     [  2.47558287651610842597E-001,  2.47558287651610842620E-001 ]
```

Beachten Sie bitte, daß die Auswertung der Integrale rechts in (13.3,13.4) durch numerische Integration natürlich extrem aufwendig ist und z.B. mit Hilfe der Eulerschen Summenformel sehr viel schneller erfolgen kann. Zu Vergleichszwecken sind in nachfolgender Tabelle weitere Ergebnisse des Programms `romberg_ex5` zusammengestellt:

x	$C(x) \in$	$S(x) \in$	`abs_err`
1.5	$0.779083738503963709^{711}_{642}$	$0.415483318265655425^{828}_{783}$	$7 \cdot 10^{-20}$
2.0	$0.753302375467891165^{620}_{539}$	$0.562848906230056479^{327}_{274}$	$8 \cdot 10^{-20}$
2.5	$0.670985872509503474^{899}_{759}$	$0.665786889578172555^{266}_{138}$	$2 \cdot 10^{-19}$
3.0	$0.561020328978138669^{374}_{232}$	$0.711685021607530032^{579}_{463}$	$2 \cdot 10^{-19}$
3.5	$0.452047147373449482^{590}_{443}$	$0.700180326217104049^{935}_{794}$	$2 \cdot 10^{-19}$
4.0	$0.368192976280974796^{334}_{282}$	$0.6421187357445146953^{51}_{08}$	$6 \cdot 10^{-20}$
4.5	$0.325249229409973829^{388}_{249}$	$0.556489304501275895^{927}_{800}$	$2 \cdot 10^{-19}$

13.7 Einschließung uneigentlicher Integrale

Wir betrachten Integrale der Form

$$\int_a^{\infty} f(x)\,dx \qquad \text{bzw.} \qquad \int_{-\infty}^{+\infty} f(x)\,dx$$

mit $a \in I\!R$ und stetigem Integranden $f : I\!R \to I\!R$.

Die grundsätzliche Vorgehensweise zur Einschließung solcher Integrale wurde schon auf Seite 564 ausführlich beschrieben und an Hand mehrerer Beispiele auf den nachfolgenden Seiten erklärt, so daß wir uns in diesem Abschnitt auf weitere Beispiele beschränken können.

13.7.1 Einschließung der Eulerschen Konstanten C

Die Eulersche Konstante C ist definiert durch:

$$\begin{aligned} -C \quad &:= \int_0^{\infty} e^{-x}\ln(x)\,dx = -0.57721566490153286060651209\ldots \\ &= \int_{-\infty}^{+\infty} x \cdot e^{x-e^x}\,dx =: I \end{aligned}$$

wobei der Integrand des letzten Integrals beliebig oft differenzierbar ist. Mit

$$I_1 := \int_{-\infty}^{-M} x \cdot e^{x-e^x}\,dx, \quad I_2 := \int_{-M}^{N} x \cdot e^{x-e^x}\,dx, \quad I_3 := \int_{N}^{+\infty} x \cdot e^{x-e^x}\,dx$$

$$\text{folgt:} \qquad -C = \int_{-\infty}^{+\infty} x \cdot e^{x-e^x}\,dx = I = I_1 + I_2 + I_3, \quad M, N \in I\!N$$

Bei hinreichend großen M, N gilt dann $-C \approx I_2$, wobei I_2 mit Hilfe der Funktion `INTEGRAL` von Seite 551 durch $-E_2 \ni I_2$ problemlos eingeschlossen werden kann, da der Integrand $g(x) = x \cdot e^{x-e^x}$ beliebig oft differenzierbar ist. Für die Integrale I_1, I_3 ergeben sich die folgenden Einschließungen:

$$0 < -\int_{-\infty}^{-M} x \cdot e^{x-e^x} dx < -\int_{-\infty}^{-M} x \cdot e^{x} dx < (M+1)e^{-M}$$

$$E_1 := [0, (M+1)e^{-M}] \quad \longrightarrow \quad I_1 \in -E_1$$

Mit Hilfe der Ungleichung[1] $x - e^x < -x^2, x \geq 0$ findet man die Abschätzung:

$$0 < \int_N^\infty x \cdot e^{x-e^x} dx < \int_N^\infty x \cdot e^{-x^2} dx = \frac{1}{2} \int_{N^2}^\infty e^{-t} dt = \frac{1}{2} \cdot e^{-N^2}$$

$$E_3 := \left[0, \frac{1}{2} \cdot e^{-N^2}\right] \quad \longrightarrow \quad I_3 \in -E_3$$

Für die Eulersche Konstante C erhält man dann die Einschließung:

$$C = \int_0^\infty e^{-x} \ln(x)\, dx \in E_1 + E_2 + E_3,$$

wobei die letzten Additionen als Intervall–Addition zu verstehen sind. Mit den Werten $N = 7, M = 55$ liefert das nachfolgende Programm `romberg_ex6`

```
program romberg_ex6;
{ Einschliessung der Eulerschen Konstanten  C=0.577215...     }
{ abs_err      :  Einzugebener absoluter Fehler               }
use li_ari,    { Intervall-Arithmetik zum Typ Longreal        }
    litaylor,  { Intervall-Taylor-Arithmetik zum Typ Longreal }
    iromberg;  { Exportiert function INTEGRAL(..) : Linterval }

function f(x : litaylor) : litaylor[lb(x)..ub(x)];
begin
   f := -x*exp( x-exp(x) );
end; { f }
var a,b        : Linterval;
   abs_err     : Longreal;
   Err         : integer;
   E1,E2,E3,u  : Linterval;
begin
   writeln('  ****************************************');
   writeln('  *    Enclosure of the Euler Constant C   *');
   writeln('  ****************************************');
   writeln;
```

[1] Diese Ungleichung beweist man z.B., indem man mit Hilfe der Prozedur `AllZeros` von Seite 439 zeigt, daß $h(x) := x(x+1) - e^x$ für $x \geq 0$ keine Nullstellen besitzt und $h(0) = -1$ negativ ist.

```
   a := -55;  b := 7;                    { Integrationsgrenzen }
   repeat
      write('abs_err = ?  ');  read(abs_err);
      if abs_err > 0 then
      begin
         E2 := INTEGRAL(f,a,b,abs_err,Err);     { I_2 in -E_2 }
         if Err=0 then
         begin  { Enclosure of  C: }
            writeln;
            u := (-a+1)*exp(a);
            E1 := Lintval(0,u.sup);
            u := 0.5*exp(-b*b);
            E3 := Lintval(0,u.sup);
            writeln('   Enclosure of the Euler Constant C:');
            writeln('   ',E1+E2+E3);
         end      else writeln('Err = ',Err);
         writeln;
      end;
   until abs_err <= 0;
end.
```

das folgende Ablaufprotokoll:

```
      *****************************************
      *    Enclosure of the Euler Constant C   *
      *****************************************

    abs_err = ?  2e-19   <enter>

       Enclosure of the Euler Constant C:
       [  5.77215664901532860541E-001,  5.77215664901532860671E-001 ]
```

Anmerkungen:

1. C wird also mit einem relativen Höchstfehler $\varepsilon_{rel} = 2.26 \cdot 10^{-19}$ in hoher Präzision eingeschlossen.
2. Neben der zeitaufwendigen numerischen Integration läßt sich C z.B. mit Hilfe der Eulerschen Summenformel sehr effektiv berechnen.
3. Die beiden Hilfsmodule `rrgi_util` und `i_util` liefern unter den jeweiligen Namen `GAMMA_C_rrGi` bzw. `GAMMA_C_Li` Einschließungen von C im `rrG`- bzw. im `Longreal`–Format, vergl. die Seiten 652, 640.

13.7.2 $\int_0^1 \frac{\ln^{16}(x)}{\sqrt{1-x^2}}\,dx$

Einzuschließen ist das Integral

$$I := \int_0^1 \frac{\ln^{16}(x)}{\sqrt{1-x^2}}\,dx$$

dessen Integrand $f(x) = \ln^{16}(x)/\sqrt{1-x^2}$ an den Randpunkten $x = 0$ und $x = 1$ nicht differenzierbar ist, so daß eine Einschließung des Intergrals I mit Hilfe der Funktion `INTEGRAL` von Seite 551 zunächst nicht möglich ist. Die beiden Substitutionen $u = \sqrt{1-x}$ und $t = -\ln(1-u)$ liefern nach einfachen Rechnungen:

$$I = \int_0^1 \frac{\ln^{16}(x)}{\sqrt{1-x^2}}\,dx = 2\int_0^\infty \frac{[t-\ln(2-e^{-t})]^{16}\cdot e^{-t}}{\sqrt{1+2\cdot e^{-t}-e^{-2t}}}\,dt$$

wobei der neue Integrand $g(x) := [t-\ln(2-e^{-t})]^{16}\cdot e^{-t}/\sqrt{1+2\cdot e^{-t}-e^{-2t}}$ jetzt im ganzen Integrationsbereich beliebig oft differenzierbar ist. Mit

$$I_1 := 2\int_0^{90} g(t)\,dt, \quad I_2 := 2\int_{90}^\infty g(t)\,dt \quad \text{folgt:} \qquad I = I_1 + I_2$$

wobei $I_1 \approx I$ jetzt mit Hilfe der Funktion `INTEGRAL` von Seite 551 durch das Intervall $E_1 \ni I_1$ problemlos eingeschlossen werden kann. Für eine Einschließung von I_2 beweist man zunächst:

$$\sqrt{1+2\cdot e^{-t}-e^{-2t}} \geq 1 \quad \Longrightarrow \quad g(t) \leq [t-\ln(2-e^{-t})]^{16}\cdot e^{-t}$$

Für $t \geq 0$ gilt zusätzlich die Doppelungleichung[2]

$$0 \leq t-\ln(2-e^{-t}) \leq t \quad \Longrightarrow \quad g(t) \leq t^{16}\cdot e^{-t}$$

und daraus ergibt sich für I_2 die Abschätzung:

$$0 < I_2 < 2\int_{90}^\infty t^{16}\cdot e^{-t}\,dt \tag{13.5}$$

[2] Mit $h(t) := t-\ln(2-e^{-t}), h(0) = 0$ und $h'(0) = 0$ beweist man die erste Ungleichung, indem man für $t > 0$ den Nachweis $h'(t) > 0$ führt; die zweite Ungleichung ist trivial!

Mit dem Algebrasystem Maple findet man $2\int_{90}^{\infty} t^{16}\cdot e^{-t}\,dt < 3.683\cdot 10^{-8}$ und damit für I_2 die Einschließung: $I_2 \in E_1 := [0, 3.683\cdot 10^{-8}]$. Für das Integral I erhält man damit die gesuchte Einschließung

$$I \in E_1 \oplus E_2$$

wobei die Berechnung der Intervalle E_1, E_2 und deren Addition $E_1 \oplus E_2$ im nachfolgenden **PASCAL-XSC** Programm `romberg_ex7` ausgeführt wird.

```
program romberg_ex7;
{ Einschliessung des Integrals:                              }
{  I = int_0^1 ln(x)^16/sqrt(1-x^2)dx                        }
{ abs_err      :  Einzugebener absoluter Fehler              }
use li_ari,    { Intervall-Arithmetik zum Typ Longreal       }
    litaylor,  { Intervall-Taylor-Arithmetik zum Typ Longreal }
    iromberg;  { Exportiert function INTEGRAL(..) : Linterval }

const stars = '*********************************';

function f(x : litaylor) : litaylor[lb(x)..ub(x)];
var v1,v2,v3 : litaylor[lb(x)..ub(x)];
begin
   v1 := exp(-x);  v2 := 2-v1;
   v3 := sqr(x-ln(v2)); v3 := sqr(v3);
   v3 := sqr(v3);       v3 := sqr(v3);
   f := 2*v1*v3 / sqrt(1+v1*v2);
end; { f }

var a,b    : Linterval;
   abs_err : Longreal;
   Err     : integer;
   E1,E2   : Linterval;

begin
   writeln('  ************************************'+stars);
   write  ('  *    Enclosure of the integral I = ');
   writeln('int_0^1 (ln(x))^16/sqrt(1-x^2)dx    *');
   writeln('  ************************************'+stars);
   writeln;
   a := 0;  b := 90;                  { Integrationsgrenzen }
```

```
   repeat
      write('abs_err = ?  ');  read(abs_err);
      if abs_err > 0 then
      begin
         E1 := INTEGRAL(f,a,b,abs_err,Err);     { I_1 in E_1 }
         if Err=0 then
         begin  { Enclosure of  I: }
            writeln;
            E2 := Lintval(0,3.683e-8);    { Enclosure of I_2 }
            writeln('   Enclosure of the integral I:');
            writeln('   ',E1+E2);
         end      else writeln('Err = ',Err);
         writeln;
      end;
   until abs_err <= 0;
end.
```

Das obige Programm `romberg_ex7` liefert mit den entsprechenden Eingabedaten das folgende Ablaufprotokoll:

```
  ***********************************************************************
  *    Enclosure of the integral I = int_0^1 (ln(x))^16/sqrt(1-x^2)dx    *
  ***********************************************************************

abs_err = ?  1e-4

   Enclosure of the integral I:
   [  2.09227899690183791621E+013,  2.09227899690183791910E+013 ]

abs_err = ?  1e-5
Err = 1
```

Anmerkungen:

- Das Integral I kann z.B. mit Mathematica in geschlossener Form dargestellt werden. Da der entsprechende Ausdruck jedoch eine ganze Seite füllt, bleibt zur Integralauswertung nur die numerische Integration.
- Das Integral $I_2 = 2\int\limits_{90}^{\infty} t^{16} e^{-t}\,dt$ wird mit Maple zunächst exakt berechnet. Der exakte Ausdruck wird danach numerisch ausgewertet und erst dann nach oben abgeschätzt. I_2 wird also mit Maple nicht durch dessen mathematisch unzuverlässige numerischer Integration abgeschätzt!

Kapitel 14

Eigenwerte

14.1 Übersicht

Wir betrachten das gewöhnlichen Eigenwertproblemen:

$$(14.1) \qquad \boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}, \qquad \boldsymbol{A} \in \mathbb{C}^{(n,n)},\ \boldsymbol{x} \in \mathbb{C}^n,\ \lambda \in \mathbb{C},\ n = 2, 3, \ldots ;$$

Zu gegebener Matrix $\boldsymbol{A}$ sind von Null verschiedene Eigenvektoren $\boldsymbol{x}_j$ und zugehörige Eigenwerte λ_j gesucht, so daß die obige Vektorgleichung erfüllt ist.

Es ist nicht Aufgabe dieses Kapitels, die vielen verschiedenen Lösungsverfahren zu diskutieren, dazu muß auf die umfangreiche Spezialliteratur verwiesen werden. Hier werden nur zu einzelnen Teilproblemen die entsprechenden **PASCAL-XSC** Routinen angegeben, mit denen man bei einfachen Eigenwerten Einschließungen der entsprechenden Eigenpaare berechnen kann. Mehrfache Eigenwerte lassen sich nur bei reellen, symmetrischen oder bei hermiteschen Matrizen einschließen. In allen anderen Fällen können mehrfache Eigenwerte mit ihren Eigenvektoren nur approximiert werden.

Beachten Sie bitte, daß mit einer berechneten Einschließung eines Eigenpaares, die exakten Werte $\boldsymbol{x}_j$ und λ_j bewiesenermaßen und meist hochgenau durch Intervalle eingeschlossen werden. Da bei sehr schlecht konditionierten Matrizen $\boldsymbol{A}$ die auf der Maschine berechneten Eigenwerte von den unvermeidbaren Rundungsfehlern oft sehr stark verfälscht werden, liefern erst die garantierten Einschließungen der gesuchten Eigenpaare eine wirklich hinreichende mathematische Sicherheit bezüglich der erhaltenen numerischen Ergebnisse.
Bezüglich der **gewöhnlichen** Eigenwertaufgabe $\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$ werden folgende Teilprobleme behandelt:

1. Für eine beliebige, reelle, im 13-stelligen BCD `real`-Format darstellbare Matrix $\boldsymbol{A} \in I\!R^{(n,n)}, 2 \leq n \leq 50$, werden Approximationen für alle Eigenpaare berechnet.

2. Ist eine hinreichend gute Approximation eines Eigenpaares mit einem einfachen Eigenwert bekannt, so lassen sich garantierte Einschließungen dieses Eigenpaares berechnen, wenn $\boldsymbol{A}$ als reelle oder komplexe Punkt- oder Intervall-Matrix im BCD `real`-Format gegeben ist. Wenn $\boldsymbol{A}$ jedoch nur als reelle Punkt- oder Intervall-Matrix vorliegt, so kann ein einfaches Eigenpaar mit Hilfe seiner Approximation auch im 26-stelligen `Longreal`-Format eingeschlossen werden.

3. Für eine beliebige reelle oder hermitesche Matrix $\boldsymbol{A}$ werden Einschließungen aller einfachen Eigenpaare berechnet. Bei mehrfachen Eigenwerten werden nur Approximationen der Eigenpaare angegeben. Die Ausgabe aller Ergebnisse im BCD `real`-Format erfolgt auf dem Bildschirm.

4. Für eine reelle, symmetrische $(n \times n)$ Matrix $\boldsymbol{A}$ werden im `real`- und `Longreal`–Format alle Eigenwerte eingeschlossen, wobei $\boldsymbol{A}$ auch mehrfache Eigenwerte besitzen darf.

5. Für eine hermitesche $(n \times n)$ Matrix $\boldsymbol{A}$ werden im `real`- und `Longreal`–Format alle Eigenwerte eingeschlossen, wobei $\boldsymbol{A}$ auch mehrfache Eigenwerte besitzen darf.

14.2 Das gewöhnliche Eigenwertproblem

Gesucht sind nichttriviale Eigenvektoren $\boldsymbol{x}_j$ und die zugehörigen Eigenwerte λ_j, mit denen die Eigenwertaufgabe

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}, \qquad \boldsymbol{A} \in \mathbb{C}^{(n,n)},\ \boldsymbol{x} \in \mathbb{C}^{n},\ \lambda \in \mathbb{C},\ n = 2, 3, \ldots$$

gelöst wird. Beachten Sie bitte, daß im Gegensatz zu den Eigenwerten λ_j die Eigenvektoren $\boldsymbol{x}_j$ nur bis auf einen von Null verschiedenen Faktor bestimmt sind. Gehören dagegen mehrere linear unabhängige Eigenvektoren $\boldsymbol{x}_i$ zu einem mehrfachen Eigenwert λ_j, so ist auch jede nichttriviale Linearkombination der x_i wieder Eigenvektor zu λ_j.

14.2.1 Approximation aller Eigenpaare, $A \in \mathbb{R}^{(n,n)}$

Mit Hilfe der folgenden Prozedur `eigen(...)`, die vom Hilfsmodul `eig_app` exportiert wird, lassen sich zu einer gegebenen reellen Matrix $\boldsymbol{A} \in \mathbb{R}^{(n,n)}$ Approximationen aller Eigenpaare berechnen. Benutzt wird ein Verfahren von Martin, Parlett, Peters, Reinsch und Wilkinson; der Algorithmus ist der Formelsammlung [4] entnommen.

```
procedure eigen (     vec      : boolean;
                  var A,EVec   : rmatrix;
                  var cew      : cvector;
                  var cnt      : intvector;
(eig_app)         var Err      : integer )
```

Eingangsgrößen:

- Mit `vec=false` erhält man Approximationen nur der Eigenwerte λ_j, mit `vec=true` werden zusätzlich auch Approximationen der Eigenvektoren berechnet.
- A ist die beliebige reelle, quadratische $(n \times n)$ Matrix.

Ausgangsgrößen:

- Die Matrix `EVec` liefert mit seiner j-ten Spalte den Eigenvektor zum reellen Eigenwert $\lambda_j,\ j = 0, 1, \ldots, n-1$; Ist jedoch der j-te Eigenwert von $\boldsymbol{A}$ komplex, so liefert die j-te Spalte von `EVec` den Realteil und die $(j+1)$-te Spalte den Imaginärteil des zugehörigen Eigenvektors. Es gilt zusätzlich $\lambda_{j+1} = \overline{\lambda_j}$ und $\boldsymbol{x}_{j+1} = \overline{\boldsymbol{x}_j}$.

- Der komplexe Vektor `cew[0..n-1]` enthält alle Eigenwerte λ_j, die im Fall komplexer Eigenwerte stets als konjugiert komplexe Eigenwert-Paare auftreten.

- Der `integer`–Vektor `cnt[0..n-1]` liefert die jeweils benötigte Iterationszahl zur Approximation des jeweiligen Eigenwerts λ_j. Bei konjugiert komplexen Eigenwert-Paaren haben die betragsgleichen `integer`–Werte `cnt[j],cnt[j+1]` entgegengesetzte Vorzeichen.

- `Err` liefert den Fehlerkode; `Err=0` bedeutet: Alle Eigenpaare wurden erfolgreich approximiert.

> Alle Indexbereiche der Matrizen und Vektoren, mit denen die Prozedur `eigen` aufgerufen wird, müssen die folgende Struktur besitzen: `0..n-1`, wobei $2 \leq n \leq 50$ die Dimension von $\boldsymbol{A}$ ist.

14.2.1.1 Programmbeispiele

Die reelle Matrix $\boldsymbol{A} = \begin{pmatrix} -2 & 0 & 1 \\ -6 & -2 & 0 \\ 19 & 5 & -4 \end{pmatrix}$ besitzt die folgenden Eigenpaare:

$$\lambda_1 = -8, \quad \boldsymbol{x}_1 = \begin{pmatrix} -\frac{1}{6} \\ -\frac{1}{6} \\ 1 \end{pmatrix}, \quad \text{und das konjugiert komplexe Eigenpaar:}$$

$$\lambda_2 = +i, \quad \boldsymbol{x}_2 = \begin{pmatrix} -\frac{1}{3} - \frac{1}{6} \cdot i \\ 1 \\ -\frac{1}{2} - \frac{2}{3} \cdot i \end{pmatrix}; \qquad \lambda_3 = -i, \quad \boldsymbol{x}_2 = \begin{pmatrix} -\frac{1}{3} + \frac{1}{6} \cdot i \\ 1 \\ -\frac{1}{2} + \frac{2}{3} \cdot i \end{pmatrix},$$

wobei die betragsmäßig größte Komponente jedes Eigenvektors auf 1 normiert wurde. Beachten Sie bitte, daß jeder Eigenvektor nur bis auf einen von Null verschiedenen, komplexen Faktor bestimmt ist.

Das nachfolgende Programm `test_eigen` liefert nach Eingabe der Matrix $\boldsymbol{A}$ die gewünschten Approximationen der drei obigen Eigenpaare:

```
program test_eigen;
{ Berechnet werden Approximationen aller Eigenpaare einer }
{ beliebigen reellen (nxn)-Matrix nach dem Verfahren von: }
{ Martin, Parlett, Peters, Reinsch und Wilkinson.         }

use type_def, eig_app, mv_ari;
var n : integer;

procedure main(n : integer);
var A,EVec : rmatrix[0..n-1,0..n-1];
   vec     : boolean;
   i,j,res : integer;
   cew     : cvector[0..n-1];
   cnt     : IntVector[0..n-1];

begin { main }
   vec := TRUE; { Auch die Eigenvektoren werden berechnet }
   writeln('Matrix A (der Dimension ',n,') = ?');
   read(A);
   writeln; writeln;
   eigen(vec,A,EVec,cew,cnt,res);
   if res <> 0 then
   begin
      writeln('Fehler:  Approximation nicht moeglich!',res);
      for i := 0 to n-1 do write('  ',cnt[i]);
   end         else
   begin
      if vec then
      begin
         writeln;
         write  ('Die normierten, approximierten ');
         writeln('Eigenvektoren (spaltenweise) sind:');
         writeln;
         For i := 0 to n-1 do
         begin
            For j := 0 to n-1 do write(' ',EVec[i,j],' ');
            writeln;
         end;
      end;
      writeln;
```

```
      write  ('Die approximierten Eigenwerte lauten:');
      writeln('         Iterationszahlen: ');
      writeln('  Realteil:             Imaginaerteil:');
      for i := 0 to n-1 do
      writeln(' ',cew[i].re,'  ', cew[i].im, '   ', cnt[i]:10);
      writeln;
   end;
end; { main }

begin { test_eigen }
   repeat
      writeln('Dimension(A) (>1) = ?');  read(n);
      if n < 2 then n := 2;
      main(n);
   until false;
end.
```

Mit den entsprechenden Eingabedaten für die Matrix $\boldsymbol{A}$ liefert das obige Programm das folgende Ablaufprotokoll:

```
Dimension(A) (>1) = ?
3           <enter>
Matrix A (der Dimension 3) = ?
-2  0  1     <enter>
-6 -2  0     <enter>
19  5 -4     <enter>

Die normierten, approximierten Eigenvektoren (spaltenweise) sind:

 -1.666666666667E-001  -3.333333333329E-001  -1.666666666666E-001
 -1.666666666667E-001   1.000000000000E+000   0.000000000000E+000
  1.000000000000E+000  -5.000000000061E-001  -6.666666666691E-001

Die approximierten Eigenwerte lauten:          Iterationszahlen:
  Realteil:             Imaginaerteil:
 -8.000000000003E+000   0.000000000000E+000           0
  0.000000000000E+000   9.999999999994E-001           5
  0.000000000000E+000  -9.999999999994E-001          -5
```

Vergleicht man mit den exakten Eigenpaaren von Seite 590, so erkennt man deren sehr gute Approximation durch die Prozedur `eigen` aus dem Modul `eig_app`. Es ergibt sich außerdem, daß die Approximationen der konjugiert komplexen Eigenwerte selbst wieder konjugiert komplex sind.

Das gleiche Programm `test_eigen` liefert für die Matrix $\boldsymbol{A} = \begin{pmatrix} 0 & 0 & -2 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix}$ mit den Eigenpaaren

$$\lambda_1 = 2,\ \boldsymbol{x}_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}; \qquad \lambda_2 = 1,\ \boldsymbol{x}_2 = \begin{pmatrix} 1 \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}; \qquad \lambda_3 = 2,\ \boldsymbol{x}_3 = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

die folgenden Approximationen:

```
Dimension(A) (>1) = ?
3         <enter>
Matrix A (der Dimension 3) = ?
0 0 -2    <enter>
1 2  1    <enter>
1 0  3    <enter>

Die normierten, approximierten Eigenvektoren (spaltenweise) sind:

  0.000000000000E+000   1.000000000000E+000   1.000000000000E+000
  1.000000000000E+000  -5.000000000000E-001   6.827439976317E-002
  0.000000000000E+000  -5.000000000000E-001  -1.000000000000E+000

Die approximierten Eigenwerte lauten:          Iterationszahlen:
  Realteil:             Imaginaerteil:
  2.000000000000E+000   0.000000000000E+000            0
  1.000000000000E+000   0.000000000000E+000            0
  2.000000000000E+000   0.000000000000E+000            0
```

Anmerkungen:

- Die Eigenwerte werden jetzt sogar exakt approximiert, wobei $\lambda_{1,3} = 2$ mehrfache Eigenwerte sind.
- Auch die Eigenvektoren werden exakt approximiert, da der 3. Eigenvektor mit der 2-ten Komponente `6.827...17E-002` aus den exakten Eigenvektoren $\boldsymbol{x}_1$, $\boldsymbol{x}_3$ linear kombiniert werden kann!

14.2.2 Einschließung eines einfachen Eigenpaares mit Hilfe einer gegebenen Eigenpaar–Approximation

In den letzten Abschnitten wurde ab Seite 589 die Prozedur `eigen` beschrieben, mit der man zu einer gegebenen, reellen Matrix $\boldsymbol{A}$ Approximationen aller Eigenpaare berechnen kann. In diesem Abschnitt wird die Prozedur `EIG` vorgestellt, mit der man mit einer solchen Eigenpaar–Approximation das entsprechende einfache Eigenpaar garantiert einschließen kann, [8]:

procedure `EIG` (**var** $\boldsymbol{A}$: ■; `LAM` : ⊙; **var** `RX` : **⊙**;
var `ILAM` : ◇; **var** `IX` : **◇**; **var** `Err : integer`)

Eingangsgrößen:

- $\boldsymbol{A}$: Reelle oder komplexe Punkt- oder Intervall–Matrix des Eigenwertproblems,
- `LAM`: Approximation eines reellen oder komplexen Eigenwerts,
- `RX`: Approximation eines zugehörigen Eigenvektors.

Ausgangsgrößen:

- `ILAM` liefert Einschließung des approximierten Eigenwerts,
- `IX` liefert Einschließung eines Eigenvektors,
- `Err` liefert den Fehlerkode; `Err=0`: Das approximierte Eigenpaar wurde erfolgreich eingeschlossen.

Erlaubte Datentypen in Prozedur `EIG`				
$\boldsymbol{A}$: ■	`(L)rmatrix`	`(L)imatrix`	`cmatrix`	`cimatrix`
`LAM` : ⊙	`(Long)real`	`(Long)real`	`complex`	`complex`
`RX` : **⊙**	`(L)rvector`	`(L)rvector`	`cvector`	`cvector`
`ILAM` : ◇	`(L)interval`	`(L)interval`	`cinterval`	`cinterval`
`IX` : **◇**	`(L)ivector`	`(L)ivector`	`civector`	`civector`
expt. Module	(l)eig	(l)ieig	ceig	cieig

14.2.2.1 Programmbeispiele

1. Beispiel: $\boldsymbol{A}$: `rmatrix`

Wir betrachten wie auf Seite 593 die gleiche Matrix $\boldsymbol{A} = \begin{pmatrix} 0 & 0 & -2 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix}$ mit den Eigenpaaren

$$\lambda_1 = 2,\ \boldsymbol{x}_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}; \qquad \lambda_2 = 1,\ \boldsymbol{x}_2 = \begin{pmatrix} 1 \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}; \qquad \lambda_3 = 2,\ \boldsymbol{x}_3 = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

Das folgende Programm `eig_ex1`

```
program eig_ex1;
{ Einschliessung eines einfachen Eigenpaares mit Hilfe }
{ einer Eigenpaar-Approximation;           A : rmatrix }
use i_ari,mv_ari,mvi_ari,eig;

procedure MAIN( N : integer );

VAR Err : integer;
    A   : rmatrix[1..N,1..N];
    EV  : rvector[1..N];
    EW  : real;
    IEV : ivector[1..N];
    IEW : interval;
    C   : char;

procedure write_EW;
var J : integer ;
begin
   writeln;
   writeln('Eigenvalue  enclosure = ', IEW );
   writeln('Eigenvector enclosure = ', IEV );
end { write_EW };

begin { MAIN }
   writeln('Matrix A = ?'); read(A);
```

```
   repeat
      write('Eigenvalue  approximation : ');  read(EW);
      write('Eigenvector approximation : ');  read(EV);
      EIG( A,EW,EV,IEW,IEV,Err );
      if Err=0 then write_EW  else
      begin
         writeln;
         writeln(EigErrMsg(Err));
      end;
      writeln;
      write( 'another eigenvalue ? (y/n) ' );
      readln; read(C);
   until C in ['N','n'];
end; { MAIN }

var N : integer ;

begin
   writeln('  *****************************************');
   writeln('  *  Enclosure of an  Eigen-pair using  an  *');
   writeln('  *  Eigen-pair approximation,  A: rmatrix  *');
   writeln('  *****************************************');
   writeln;
   repeat
      write('Matrix dimension = ? (End = 0) '); read(N);
      if N>0 then MAIN( N );
   until N<=0;
end.
```

liefert mit den entsprechenden Eigenpaar–Approximationen das umseitige Ablaufprotokoll. Bitte beachte Sie dabei, daß die Prozedur `EIG(...)` von Seite 594 nur Einschließungen eines einfachen Eigenpaares berechnen kann. Da im vorliegenden Beispiel $\lambda_{1,3} = 2$ ein doppelter Eigenwert ist, erfolgt sogar mit den jeweils **exakten** Eigenpaar–Approximationen eine entsprechende Fehlermeldung.

Wenn eine reelle Matrix konjugiert komplexe Eigenwerte besitzt, können die entsprechenden Eigenwertpaare nicht mit Hilfe der Prozedur `EIG` aus dem Testprogramm `eig_ex1` eingeschlossen werden. Vergleichen Sie bitte in einem solchen Fall das Beispiel auf Seite 605.

```
   ******************************************
   *  Enclosure of an  Eigen-pair using  an  *
   *  Eigen-pair approximation,  A: rmatrix  *
   ******************************************

Matrix dimension = ? (End = 0) 3            <enter>
Matrix A = ?
0 0 -2    <enter>
1 2  1    <enter>
1 0  3    <enter>
Eigenvalue  approximation : 2               <enter>
Eigenvector approximation : 0 1 0           <enter>

Error: Inversion failed, matrix singular (?)!

another eigenvalue ? (y/n) y
Eigenvalue  approximation : 0.9             <enter>
Eigenvector approximation : 1.1 -0.45 -0.6  <enter>

Eigenvalue  enclosure = [  9.999999999999E-001,  1.000000000001E+000 ]
Eigenvector enclosure =
[  1.000000000000E+000,  1.000000000000E+000 ]
[ -5.000000000001E-001, -4.999999999999E-001 ]
[ -5.000000000001E-001, -4.999999999999E-001 ]


another eigenvalue ? (y/n) y
Eigenvalue  approximation : 2               <enter>
Eigenvector approximation : -1 0 1          <enter>

Error: Inversion failed, matrix singular (?)!
```

2. Beispiel: $\boldsymbol{A}$: `Lrmatrix`

Wir betrachten die gleiche Matrix wie im 1. Beispiel: $\boldsymbol{A} = \begin{pmatrix} 0 & 0 & -2 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix}$ mit den Eigenpaaren

$$\lambda_1 = 2,\ \boldsymbol{x}_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}; \qquad \lambda_2 = 1,\ \boldsymbol{x}_2 = \begin{pmatrix} 1 \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}; \qquad \lambda_3 = 2,\ \boldsymbol{x}_3 = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

Das folgende Programm `leig_ex1`

```
program leig_ex1;
{ Einschliessung eines einfachen Eigenpaares mit Hilfe }
{ einer Eigenpaar-Approximation;          A : Lrmatrix }
use li_ari,lmv_ari,lmvi_ari,leig;

procedure MAIN( N : integer );

VAR Err : integer;
    A   : Lrmatrix[1..N,1..N];
    EV  : Lrvector[1..N];
    EW  : Longreal;
    IEV : Livector[1..N];
    IEW : Linterval;
    C   : char;

procedure write_EW;
var J : integer ;
begin
   writeln;
   writeln('EVal. enclosure = ', IEW );
   writeln('Eigenvector enclosure = ', IEV );
end { write_EW };

begin { MAIN }
   writeln('Matrix A = ?'); read(A);
   repeat
      write('Eigenvalue  approximation : ');  read(EW);
      write('Eigenvector approximation : ');  read(EV);
      EIG( A,EW,EV,IEW,IEV,Err );
      if Err=0 then write_EW  else
      begin
         writeln;
         writeln(LEigErrMsg(Err));
      end;
      writeln;
      write( 'another eigenvalue ? (y/n) ' );
      readln; read(C);
   until C in ['N','n'];
end; { MAIN }
```

```
var N : integer ;

begin
   writeln('  *****************************************');
   writeln('  *  Enclosure of an  Eigen-pair using  an  *');
   writeln('  *  Eigen-pair approximation, A: Lrmatrix  *');
   writeln('  *****************************************');
   writeln;
   repeat
      write('Matrix dimension = ? (End = 0) '); read(N);
      if N>0 then MAIN( N );
   until N<=0;
end.
```

liefert mit der Approximation des einfachen Eigenpaares das nachfolgende Ablaufprotokoll:

```
    *****************************************
    *  Enclosure of an  Eigen-pair using  an  *
    *  Eigen-pair approximation, A: Lrmatrix  *
    *****************************************

  Matrix dimension = ? (End = 0) 3              <enter>
  Matrix A = ?
  0 0 -2     <enter>
  1 2  1     <enter>
  1 0  3     <enter>
  Eigenvalue  approximation : 1.01              <enter>
  Eigenvector approximation : 0.98 -0.49 -0.48   <enter>

  EVal. enclosure =
  [  9.99999999999999999999E-001,  1.00000000000000000001E+000 ]
  Eigenvector enclosure =
  [  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
  [ -5.00000000000000000001E-001, -4.99999999999999999999E-001 ]
  [ -5.00000000000000000001E-001, -4.99999999999999999999E-001 ]


  another eigenvalue ? (y/n) n
  Matrix dimension = ? (End = 0) 0
```

Das einfache Eigenpaar wird also jetzt im 21-stelligen `Longreal`–Format hochgenau eingeschlossen.

3. Beispiel: $\boldsymbol{A}$: `imatrix`

Wir betrachten jetzt die Matrix $\boldsymbol{A} = \begin{pmatrix} 0 & \sqrt{2} & 0 \\ \sqrt{2} & 0 & 1 \\ 1 & 3 & 0 \end{pmatrix}$ mit den Eigenpaaren

$$\lambda_1 = 0,\ \boldsymbol{x}_1 = \begin{pmatrix} \frac{-1}{\sqrt{2}} \\ 0 \\ 1 \end{pmatrix};\quad \lambda_2 = \sqrt{5},\ \boldsymbol{x}_2 = \begin{pmatrix} \frac{\sqrt{2}}{3} \\ \frac{\sqrt{5}}{3} \\ 1 \end{pmatrix};\quad \lambda_3 = -\sqrt{5},\ \boldsymbol{x}_3 = \begin{pmatrix} \frac{\sqrt{2}}{3} \\ -\frac{\sqrt{5}}{3} \\ 1 \end{pmatrix}$$

Da $\sqrt{2}$ als Gleitpunktzahl nicht exakt darstellbar ist, muß die gegebene Matrix $\boldsymbol{A}$: `imatrix` als Intervallmatrix gewählt werden, um die irrationalen Koeffizienten maximalgenau einschließen zu können. Das **PASCAL-XSC** Programm `ieig_ex1`

```
program ieig_ex1;
{ Einschliessung eines einfachen Eigenpaares mit Hilfe }
{ einer Eigenpaar-Approximation;           A : imatrix }
use i_ari,mv_ari,mvi_ari,ieig;

procedure MAIN( N : integer );

VAR Err : integer;
    A   : imatrix[1..N,1..N];
    EV  : rvector[1..N];
    EW  : real;
    IEV : ivector[1..N];
    IEW : interval;
    C   : char;

procedure write_EW;
var J : integer ;
begin
   writeln;
   writeln('Eigenvalue  enclosure = ', IEW );
   writeln('Eigenvector enclosure = ', IEV );
end { write_EW };

begin { MAIN }
```

```
   writeln('Matrix A = ?'); read(A);
   repeat
      write('Eigenvalue  approximation : ');  read(EW);
      write('Eigenvector approximation : ');  read(EV);
      EIG( A,EW,EV,IEW,IEV,Err );
      if Err=0 then write_EW  else
      begin
         writeln;
         writeln(IEigErrMsg(Err));
      end;
      writeln;
      write( 'another eigenvalue ? (y/n) ' );
      readln; read(C);
   until C in ['N','n'];
end; { MAIN }

var N : integer ;

begin
   writeln('  *****************************************');
   writeln('  *  Enclosure of an  Eigen-pair using  an  *');
   writeln('  *  Eigen-pair approximation,  A: imatrix  *');
   writeln('  *****************************************');
   writeln;
   repeat
      write('Matrix dimension = ? (End = 0) '); read(N);
      if N>0 then MAIN( N );
   until N<=0;
end.
```

liefert mit einer Approximation des zweiten Eigenpaares

$$\lambda_2 = \sqrt{5} \approx 2.2, \qquad \boldsymbol{x}_2 = \begin{pmatrix} \frac{\sqrt{2}}{3} \\ \frac{\sqrt{5}}{3} \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.47 \\ 0.74 \\ 0.99 \end{pmatrix}$$

das folgende Ablaufprotokoll:

```
  ******************************************
  *  Enclosure of an  Eigen-pair using  an  *
  *  Eigen-pair approximation,  A: imatrix  *
  ******************************************

Matrix dimension = ? (End = 0) 3               <enter>
Matrix A = ?
0 [1.414213562373,1.414213562374] 0            <enter>
[1.414213562373,1.414213562374] 0 1            <enter>
0 3 0                                          <enter>
Eigenvalue  approximation : 2.2                <enter>
Eigenvector approximation : 0.47 0.74 0.99     <enter>

Eigenvalue  enclosure = [  2.236067977499E+000,  2.236067977501E+000 ]
Eigenvector enclosure =
[  4.714045207909E-001,  4.714045207914E-001 ]
[  7.453559924999E-001,  7.453559925002E-001 ]
[  1.000000000000E+000,  1.000000000000E+000 ]
```

4. Beispiel: $\boldsymbol{A}$: `Limatrix`

Wir betrachten wieder, wie im vorhergehenden Beispiel die Matrix

$\boldsymbol{A} = \begin{pmatrix} 0 & \sqrt{2} & 0 \\ \sqrt{2} & 0 & 1 \\ 1 & 3 & 0 \end{pmatrix}$ mit den Eigenpaaren

$$\lambda_1 = 0,\ \boldsymbol{x}_1 = \begin{pmatrix} \frac{-1}{\sqrt{2}} \\ 0 \\ 1 \end{pmatrix};\quad \lambda_2 = \sqrt{5},\ \boldsymbol{x}_2 = \begin{pmatrix} \frac{\sqrt{2}}{3} \\ \frac{\sqrt{5}}{3} \\ 1 \end{pmatrix};\quad \lambda_3 = -\sqrt{5},\ \boldsymbol{x}_3 = \begin{pmatrix} \frac{\sqrt{2}}{3} \\ -\frac{\sqrt{5}}{3} \\ 1 \end{pmatrix}$$

Für $\boldsymbol{A}$ wählen wir jetzt den Datentyp `Limatrix`, um $a_{1,2} = \sqrt{2}$ im 26-stelligen `Longreal`–Format mit noch höherer Präzision einschließen zu können. Mit der Approximation des ersten Eigenpaares

$$\lambda_1 = 0 \approx 10^{-40}, \qquad \boldsymbol{x}_1 = \begin{pmatrix} \frac{-1}{\sqrt{2}} \\ 0 \\ 1 \end{pmatrix} \approx \begin{pmatrix} -0.7 \\ 10^{-40} \\ 0.99 \end{pmatrix}$$

liefert das nachfolgende Programm

```
program lieig_ex1;
{ Einschliessung eines einfachen Eigenpaares mit Hilfe }
{ einer Eigenpaar-Approximation;          A : Limatrix }
use li_ari,lmv_ari,lmvi_ari,lieig;

procedure MAIN( N : integer );

VAR Err : integer;
    A   : Limatrix[1..N,1..N];
    EV  : Lrvector[1..N];
    EW  : Longreal;
    IEV : Livector[1..N];
    IEW : Linterval;
    C   : char;

procedure write_EW;
var J : integer ;
begin
   writeln;
   writeln('EVal. enclosure = ', IEW );
   writeln('Eigenvector enclosure = ', IEV );
end { write_EW };

begin { MAIN }
   writeln('Matrix A = ?'); read(A);
   repeat
      write('Eigenvalue  approximation : ');  read(EW);
      write('Eigenvector approximation : ');  read(EV);
      EIG( A,EW,EV,IEW,IEV,Err );
      if Err=0 then write_EW  else
      begin
         writeln;
         writeln(LIEigErrMsg(Err));
      end;
      writeln;
      write( 'another eigenvalue ? (y/n) ' );
      readln; read(C);
   until C in ['N','n'];
end; { MAIN }
```

```
var N : integer ;
begin
   writeln('  *****************************************');
   writeln('  *  Enclosure of an  Eigen-pair using  an  *');
   writeln('  *  Eigen-pair approximation, A: Limatrix  *');
   writeln('  *****************************************');
   writeln;
   repeat
      write('Matrix dimension = ? (End = 0) '); read(N);
      if N>0 then MAIN( N );
   until N<=0;
end.
```

folgendes Ablaufprotokoll:

```
    *******************************************
    *  Enclosure of an  Eigen-pair using  an  *
    *  Eigen-pair approximation, A: Limatrix  *
    *******************************************

  Matrix dimension = ? (End = 0) 3                        <enter>
  Matrix A = ?
  0 [1.41421356237309504880,1.41421356237309504881] 0     <enter>
  [1.41421356237309504880,1.41421356237309504881] 0 1     <enter>
  0 3 0

  Eigenvalue  approximation : 1e-40                       <enter>
  Eigenvector approximation : -0.7 1e-40 0.99             <enter>

  EVal. enclosure =
  [ -4.45505827800000000000E-061,  4.45505992300000000000E-061 ]
  Eigenvector enclosure =
  [ -7.07106781186547524402E-001, -7.07106781186547524396E-001 ]
  [ -1.48501942700000000000E-061,  1.48501997500000000000E-061 ]
  [  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
```

Hinweise:

- Beachten Sie, daß bei einer Intervallmatrix die Eigenpaare nicht mehr wie bei einer Punktmatrix hochgenau eingeschlossen werden können.
- Zeigen Sie, daß die Approximation eines Eigenvektors mit einem beliebigen, von Null verschiedenen Faktor multipliziert werden kann, d.h. auch mit $\boldsymbol{x}_1 \approx (-0.98, 1.4 \cdot 10^{-40}, 1.39)^T$ erhält man eine garantierte Einschließung des ersten Eigenpaares.

5. Beispiel: $\boldsymbol{A}$: `cmatrix`; $\Im(\boldsymbol{A}) = 0$;

Wir betrachten zunächst die reelle Matrix $\boldsymbol{A} = \begin{pmatrix} 2 & 2 & 0 & -3 \\ 4 & 5 & 2 & -5 \\ -6 & -4 & -1 & 6 \\ 4 & 3 & 2 & -1 \end{pmatrix}$ mit den beiden konjugiert komplexen Eigenpaaren

$$\lambda_1 = 2 + \sqrt{6} \cdot i,\ \boldsymbol{x}_1 = \begin{pmatrix} -\frac{1}{2} \\ -\frac{9}{11} - \frac{\sqrt{6}}{22} \cdot i \\ 1 \\ -\frac{6}{11} + \frac{3}{22}\sqrt{6} \cdot i \end{pmatrix}; \qquad \lambda_2 = \overline{\lambda}_1,\ \boldsymbol{x}_2 = \overline{\boldsymbol{x}}_1$$

dabei wurde die dritte Komponente als betragsmäßig größte Komponente auf 1 normiert. Das nachfolgende Programm

```
program ceig_ex1;
{ Einschliessung eines einfachen Eigenpaares mit Hilfe }
{ einer Eigenpaar-Approximation;           A : cmatrix }
use c_ari,ci_ari,mvc_ari,mvci_ari,ceig;

procedure MAIN( N : integer );

VAR Err : integer;
    A   : cmatrix[1..N,1..N];
    EV  : cvector[1..N];
    EW  : complex;
    IEV : civector[1..N];
    IEW : cinterval;
    C   : char;

procedure write_EW;
var J : integer ;
begin
   writeln;
   writeln('Eigenvalue  enclosure = ', IEW );
   writeln('Eigenvector enclosure = ', IEV );
end { write_EW };
```

```
begin { MAIN }
   writeln('Matrix A = ?'); read(A);
   repeat
      write('Eigenvalue  approximation : ');  read(EW);
      write('Eigenvector approximation : ');  read(EV);
      EIG( A,EW,EV,IEW,IEV,Err );
      if Err=0 then write_EW  else
      begin
         writeln;
         writeln(CEigErrMsg(Err));
      end;
      writeln;
      write( 'another eigenvalue ? (y/n) ' );
      readln; read(C);
   until C in ['N','n'];
end; { MAIN }

var N : integer ;

begin
   writeln('  ******************************************');
   writeln('  *  Enclosure of an  Eigen-pair using  an  *');
   writeln('  *  Eigen-pair approximation,  A: cmatrix  *');
   writeln('  ******************************************');
   writeln;
   repeat
      write('Matrix dimension = ? (End = 0) '); read(N);
      if N>0 then MAIN( N );
   until N<=0;
end.
```

liefert mit den Approximationen

$$\lambda_1 = 2 + \sqrt{6}i \approx 1.9 + 2.4i, \; \boldsymbol{x}_1 = \begin{pmatrix} -\frac{1}{2} \\ -\frac{9}{11} - \frac{\sqrt{6}}{22} \cdot i \\ 1 \\ -\frac{6}{11} + \frac{3}{22}\sqrt{6} \cdot i \end{pmatrix} \approx \begin{pmatrix} -0.49 \\ -0.81 - 0.11 \cdot i \\ 0.99 \\ -0.54 + 0.33 \cdot i \end{pmatrix}$$

eine garantierte Einschließung des ersten Eigenpaares durch das folgende Ablaufprotokoll:

```
   *****************************************
   *  Enclosure of an  Eigen-pair using  an  *
   *  Eigen-pair approximation,  A: cmatrix  *
   *****************************************

 Matrix dimension = ? (End = 0) 4       <enter>
 Matrix A = ?
  2   2   0  -3                         <enter>
  4   5   2  -5                         <enter>
 -6  -4  -1   6                         <enter>
  4   3   2  -1                         <enter>
 Eigenvalue  approximation : (1.9,2.4)
 Eigenvector approximation : -0.49 (-0.81,-0.11) 0.99 (-0.54,0.33)

 Eigenvalue  enclosure = ( [  1.999999999999E+000,  2.000000000001E+000 ],
   [  2.449489742783E+000,  2.449489742784E+000 ] )
 Eigenvector enclosure =
 ( [ -5.000000000001E-001, -4.999999999999E-001 ],
   [ -5.082300000000E-024,  5.087400000000E-024 ] )
 ( [ -8.181818181819E-001, -8.181818181818E-001 ],
   [ -1.113404428538E-001, -1.113404428537E-001 ] )
 ( [  1.000000000000E+000,  1.000000000000E+000 ],
   [  0.000000000000E+000,  0.000000000000E+000 ] )
 ( [ -5.454545454546E-001, -5.454545454545E-001 ],
   [  3.340213285613E-001,  3.340213285614E-001 ] )
```

6. Beispiel: $\boldsymbol{A}$: `cmatrix`, $\Im(\boldsymbol{A}) \neq 0$

Wir betrachten jetzt die komplexe Matrix $\boldsymbol{A} = \begin{pmatrix} i & 1+i \\ 1+2i & 2i \end{pmatrix}$ mit den beiden Eigenpaaren

$$\lambda_1 = -1,\ \boldsymbol{x}_1 = \begin{pmatrix} -1 \\ +1 \end{pmatrix}; \qquad \lambda_2 = 1+3i,\ \boldsymbol{x}_2 = \begin{pmatrix} \frac{1}{5}(3-i) \\ 1 \end{pmatrix};$$

Mit der Approximation des zweiten Eigenpaares

$$\lambda_2 = 1+3i \approx 0.91 + 3.1i, \qquad \boldsymbol{x}_2 = \begin{pmatrix} \frac{1}{5}(3-i) \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.61 - 0.19i \\ 1.05 \end{pmatrix}$$

liefert das Programm `ceig_ex1` von Seite 605 das folgende Ablaufprotokoll:

```
  ****************************************
  *  Enclosure of an  Eigen-pair using  an  *
  *  Eigen-pair approximation,  A: cmatrix  *
  ****************************************

Matrix dimension = ? (End = 0) 2
Matrix A = ?
(0,1) (1,1)
(1,2) (0,2)
Eigenvalue  approximation : (0.91,3.1)
Eigenvector approximation : (0.61,-0.19) 1.05

Eigenvalue  enclosure = ( [  9.999999999999E-001,  1.000000000001E+000 ],
  [  2.999999999999E+000,  3.000000000001E+000 ] )
Eigenvector enclosure =
( [  5.999999999999E-001,  6.000000000001E-001 ],
  [ -2.000000000001E-001, -1.999999999999E-001 ] )
( [  1.000000000000E+000,  1.000000000000E+000 ],
  [  0.000000000000E+000,  0.000000000000E+000 ] )
```

Das zweite Eigenpaar wird also hochgenau eingeschlossen. Auch hier wird der Eigenvektor so normiert, das seine betragsmäßig größte zweite Komponente auf 1 gesetzt wird.

7. Beispiel: $\boldsymbol{A}$: `cimatrix`

Die folgende Matrix $\boldsymbol{A} = \begin{pmatrix} i & 2 \\ \sqrt{2} & i \end{pmatrix}$ mit den beiden Eigenpaaren

$$\lambda_1 = \sqrt[4]{8} + i,\ \boldsymbol{x}_1 = \begin{pmatrix} 1 \\ \frac{1}{\sqrt[4]{2}} \end{pmatrix}; \qquad \lambda_2 = -\sqrt[4]{8} + i,\ \boldsymbol{x}_2 = \begin{pmatrix} 1 \\ \frac{-1}{\sqrt[4]{2}} \end{pmatrix}$$

muß als komplexe Intervallmatrix vom Typ `cimatrix` deklariert werden, damit die irrationale Komponente $a_{2,1} = \sqrt{2}$ maximalgenau eingeschlossen werden kann. Das folgende Programm

```
program cieig_ex1;
{ Einschliessung eines einfachen Eigenpaares mit Hilfe }
{ einer Eigenpaar-Approximation;          A : cimatrix }
use c_ari,ci_ari,mvc_ari,mvci_ari,cieig;

procedure MAIN( N : integer );

VAR Err : integer;
```

```
    A   : cimatrix[1..N,1..N];
    EV  : cvector[1..N];
    EW  : complex;
    IEV : civector[1..N];
    IEW : cinterval;
    C   : char;

procedure write_EW;
var J : integer ;
begin
   writeln;
   writeln('Eigenvalue  enclosure = ', IEW );
   writeln('Eigenvector enclosure = ', IEV );
end { write_EW };

begin { MAIN }
   writeln('Matrix A = ?'); read(A);
   repeat
      write('Eigenvalue  approximation : ');  read(EW);
      write('Eigenvector approximation : ');  read(EV);
      EIG( A,EW,EV,IEW,IEV,Err );
      if Err=0 then write_EW  else
      begin
         writeln;
         writeln(CIEigErrMsg(Err));
      end;
      writeln;
      write( 'another eigenvalue ? (y/n) ' );
      readln; read(C);
   until C in ['N','n'];
end; { MAIN }

var N : integer ;

begin
   writeln('  *****************************************');
   writeln('  *  Enclosure of an  Eigen-pair using  an  *');
   writeln('  *  Eigen-pair approximation, A: cimatrix  *');
   writeln('  *****************************************');
   writeln;
```

```
   repeat
      write('Matrix dimension = ? (End = 0) '); read(N);
      if N>0 then MAIN( N );
   until N<=0;
end.
```

liefert mit Hilfe der Approximation des ersten Eigenpaares

$$\lambda_1 = \sqrt[4]{8} + i \approx 1.6 + 0.9i, \quad \boldsymbol{x}_1 = \begin{pmatrix} 1 \\ \frac{1}{\sqrt[4]{2}} \end{pmatrix} \approx \begin{pmatrix} 1.1 \\ 0.85 \end{pmatrix}$$

mit dem folgenden Ablaufprotokoll eine garantierte Einschließung des ersten Eigenpaares:

```
   ******************************************
   *  Enclosure of an  Eigen-pair using  an  *
   *  Eigen-pair approximation, A: cimatrix  *
   ******************************************

 Matrix dimension = ? (End = 0) 2          <enter>
 Matrix A = ?
 (0,1) 2                                   <enter>
 [1.414213562373,1.414213562374] (0,1)     <enter>
 Eigenvalue  approximation : (1.6,0.9)     <enter>
 Eigenvector approximation : 1.1 0.85      <enter>

 Eigenvalue  enclosure = ( [  1.681792830507E+000,  1.681792830509E+000 ],
   [  9.999999999999E-001,  1.000000000001E+000 ] )
 Eigenvector enclosure =
 ( [  1.000000000000E+000,  1.000000000000E+000 ],
   [  0.000000000000E+000,  0.000000000000E+000 ] )
 ( [  8.408964152536E-001,  8.408964152541E-001 ],
   [ -1.239365436629E-014,  1.240440530360E-014 ] )
```

Auch hier erkennt man, daß die Einschließung des Eigenpaares wegen unvermeidbarer Intervallüberschätzungen nicht mehr hochgenau erfolgen kann, wenn irrationale Matrixkomponenten durch Intervalle eingeschlossen werden müssen.

14.2.3 Einschließung bzw. Approximation aller Eigenpaare

Im Abschnitt 14.2.1 haben wir zunächst die Prozedur `eigen(...)` kennengelernt, mit der man zu einer vorgegebenen, reellen Matrix $\boldsymbol{A}$ Approximationen aller Eigenpaare berechnen konnte. Mit Hilfe der Prozedur `EIG(...)` ließ sich dann im Abschnitt 14.2.2 zu einer vorgegebenen Eigenpaar–Approximation das zugehörige, einfache Eigenpaar einer reellen oder komplexen Punkt- oder Intervallmatrix garantiert einschließen.

In diesem Abschnitt werden wir mit Hilfe der Prozedur `eig_all(A)` alle einfachen Eigenpaare garantiert einschließen und alle mehrfachen Eigenpaare i.a. gut approximieren. Obige Prozedur besitzt als einzigen Parameter nur die Eingangsgröße $\boldsymbol{A}$ des Eigenwertproblems

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$$

Für eine quadratische Matrix $\boldsymbol{A}$ gibt

procedure `eig_all` (**var** A : ■)

die Einschließungen aller einfachen Eigenpaare und die Approximationen aller mehrfachen Eigenpaare auf dem Bildschirm aus.

Erlaubte Datentypen in Prozedur `eig_all`				
$\boldsymbol{A}$: ■	`rmatrix`	`imatrix`	`cmatrix`	`cimatrix`
exportierende Module	eig_reell	ieig_reell	eig_herm	ieig_herm

Für den Datentyp `cmatrix` muß die Matrix $\boldsymbol{A}$ hermitesch sein, und für den Typ `cimatrix` muß die intern berechnete Mittelpunktsmatrix der komplexen Intervallmatrix $\boldsymbol{A}$ hermitesch sein, sonst erfolgt eine entsprechende Fehlermeldung.

Beachten Sie bitte, daß in der Praxis Eigenwertprobleme mit komplexer Matrix fast immer hermitesch sind. Für nicht hermitesche Matrizen läßt sich jedoch nach dem Beispiel von Seite 607 auch dann noch eine Einschließung eines einfachen Eigenpaares berechnen, wenn eine entsprechende Approximation diese Eigenpaares bekannt ist. Beachten Sie ferner, daß eine komplexe Intervallmatrix $\boldsymbol{A}$ nicht nur aus hermiteschen Matrizen bestehen kann. Ist jedoch ihre Mittelpunktsmatrix hermitesch, so können alle einfachen Eigenpaare von $\boldsymbol{A}$ eingeschlossen werden, vergl. dazu das Beispiel von Seite 618.

14.2.3.1 Programmbeispiele

1. Beispiel: $\boldsymbol{A}$: `rmatrix`

Die Matrix $\boldsymbol{A} = \begin{pmatrix} 5 & -2 & -4 \\ -2 & 2 & 2 \\ -4 & 2 & 5 \end{pmatrix}$ mit den beiden Eigenpaaren

$$\lambda_1 = 1,\ \boldsymbol{x}_1 = \begin{pmatrix} 0 \\ 1 \\ -0.5 \end{pmatrix};\quad \lambda_2 = 10,\ \boldsymbol{x}_2 = \begin{pmatrix} 1 \\ -0.5 \\ -1 \end{pmatrix};\quad \lambda_3 = 1,\ \boldsymbol{x}_3 = \begin{pmatrix} 0.5 \\ 1 \\ 0 \end{pmatrix}$$

besitzt mit $\lambda_{1,3} = 1$ einen doppelten Eigenwert, so daß nur eine Approximation der entsprechenden Eigenpaare möglich ist. Das folgende Programm

```
program eig_all1;
{ Einschliessung bzw. Approximation aller einfachen bzw. }
{ mehrfachen Eigenpaare;      A : rmatrix[0..n-1,0..n-1] }

use mv_ari, eig_reell;
var n : integer;

procedure main(n : integer);
var A : rmatrix[0..n-1,0..n-1];
begin
   writeln('Reelle Matrix (Dim(A)=',n,') = ?');
   read(A);
   eig_all(A); { Alle Einschliessungen bzw. Approximationen }
end; { main }

begin
   repeat
      writeln('Dimension(A) = ?');
      read(n);
      main(n);
   until false;
end.
```

liefert mit den entsprechenden Eingabedaten für die Matrix $\boldsymbol{A}$ das folgende Ablaufprotokoll:

```
Dimension(A) = ?
3            <enter>
Reelle Matrix (Dim(A)=3) = ?
 5 -2 -4      <enter>
-2  2  2      <enter>
-4  2  5      <enter>
  ***********************************************************
  *  Berechnung aller Eigenpaare einer reellen (n x n)Matrix  *
  *  Mehrfache Eigenwerte u. -Vektoren werden approximiert,   *
  *  Einfache  Eigenwerte u. -Vektoren werden eingeschlossen  *
  *  Indexbereich der (n x n)-Matrix:    [0..n-1,0..n-1]      *
  ***********************************************************

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
Keine Einschliessung eines Eigenpaares moeglich:
Approximation des 1-ten reellen Eigenwerts:
 1.000000000000E+000
Approximation des zugehoerigen Eigenvektors:

 1.000000000000E+000
 4.000000000001E-001
 8.000000000002E-001

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
2-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[  9.999999999999E+000,  1.000000000001E+001 ]
Einschliessung des zugehoerigen Eigenvektors:

[  1.000000000000E+000,  1.000000000000E+000 ]
[ -5.000000000001E-001, -4.999999999999E-001 ]
[ -1.000000000001E+000, -9.999999999999E-001 ]

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
Keine Einschliessung eines Eigenpaares moeglich:
Approximation des 3-ten reellen Eigenwerts:
 1.000000000000E+000
Approximation des zugehoerigen Eigenvektors:

 2.349043742616E-001
 9.999999999997E-001
-2.650956257384E-001
```

Bitte beachten Sie, daß sich die beiden Eigenvektor–Approximationen im Rahmen der Approximationsungenauigkeit jeweils durch die exakten Eigenvektoren $\boldsymbol{x}_1$ und $\boldsymbol{x}_3$ linear kombinieren lassen. Das einfache Eigenpaar mit dem Eigenwert $\lambda_2 = 10$ wird hochgenau eingeschlossen.

2. Beispiel: $\boldsymbol{A}$: imatrix

Die Matrix $\boldsymbol{A} = \begin{pmatrix} 0 & \sqrt{2} & 0 \\ \sqrt{2} & 0 & 1 \\ 1 & 3 & 0 \end{pmatrix}$ mit den Eigenpaaren

$$\lambda_1 = 0,\ \boldsymbol{x}_1 = \begin{pmatrix} \frac{-1}{\sqrt{2}} \\ 0 \\ 1 \end{pmatrix};\quad \lambda_2 = \sqrt{5},\ \boldsymbol{x}_2 = \begin{pmatrix} \frac{\sqrt{2}}{3} \\ \frac{\sqrt{5}}{3} \\ 1 \end{pmatrix};\quad \lambda_3 = -\sqrt{5},\ \boldsymbol{x}_3 = \begin{pmatrix} \frac{\sqrt{2}}{3} \\ -\frac{\sqrt{5}}{3} \\ 1 \end{pmatrix}$$

muß wie auf Seite 600 als Intervallmatrix deklariert werden, damit z.B. die Komponente $a_{1,2} = \sqrt{2}$ als irrationale Zahl maximalgenau durch ein Intervall eingeschlossen werden kann[1]. Das nachfolgende

```
program eig_all2;
{ Einschliessung bzw. Approximation aller einfachen bzw. }
{ mehrfachen Eigenpaare;      A : imatrix[0..n-1,0..n-1] }
use mvi_ari, ieig_reell;
var n : integer;
procedure main(n : integer);
var A : imatrix[0..n-1,0..n-1];
begin
   writeln('Reelle Matrix (Dim(A)=',n,') = ?');
   read(A);
   eig_all(A); { Alle Einschliessungen bzw. Approximationen }
end; { main }

begin
   repeat
      writeln('Dimension(A) = ?');
      read(n);
      main(n);
   until false;
end.
```

[1]Wenn $a_{1,2} = \sqrt{2}$ im `real`-Format nur approximiert wird, kann man eine korrekte Einschließung der Eigenpaare natürlich nicht erwarten!

liefert mit den entsprechenden Eingabedaten für die Matrix $\boldsymbol{A}$ das folgende Ablaufprotokoll mit den garantierten Einschließungen der drei Eigenpaare:

```
Dimension(A) = ?
3                                       <enter>
Reelle Matrix (Dim(A)=3) = ?
0 [1.414213562373,1.414213562374] 0   <enter>
[1.414213562373,1.414213562374] 0 1   <enter>
0 3 0                                   <enter>
  ************************************************************
  *  Berechnung der Eigenpaare einer reellen Intervallmatrix  *
  *  Mehrfache Eigenwerte u. -Vektoren werden approximiert,   *
  *  Einfache  Eigenwerte u. -Vektoren werden eingeschlossen  *
  *  Indexbereich der (n x n)-Matrix:    [0..n-1,0..n-1]      *
  ************************************************************

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
1-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[ -1.000000000000E-267,  1.000000000000E-267 ]
Einschliessung des zugehoerigen Eigenvektors:

[ -7.071067811866E-001, -7.071067811860E-001 ]
[ -1.000000000000E-267,  1.000000000000E-267 ]
[  1.000000000000E+000,  1.000000000000E+000 ]

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
2-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[  2.236067977499E+000,  2.236067977501E+000 ]
Einschliessung des zugehoerigen Eigenvektors:

[  4.714045207909E-001,  4.714045207914E-001 ]
[  7.453559924999E-001,  7.453559925002E-001 ]
[  1.000000000000E+000,  1.000000000000E+000 ]

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
3-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[ -2.236067977501E+000, -2.236067977499E+000 ]
Einschliessung des zugehoerigen Eigenvektors:

[  4.714045207909E-001,  4.714045207914E-001 ]
[ -7.453559925002E-001, -7.453559924999E-001 ]
[  1.000000000000E+000,  1.000000000000E+000 ]
```

3. Beispiel: $\boldsymbol{A}$: `cmatrix`

Gegeben sei die hermitesche Matrix $\boldsymbol{A} = \begin{pmatrix} 1 & i & 1 \\ -i & 2 & -i \\ 1 & i & 1 \end{pmatrix}$ mit den Eigenpaaren

$$\lambda_{1,3} = 2 \pm \sqrt{2}, \quad \boldsymbol{x}_{1,3} = \begin{pmatrix} \pm\frac{\sqrt{2}}{2} \\ 1 \\ \pm\frac{\sqrt{2}}{2} \end{pmatrix}; \qquad \lambda_2 = 0, \quad \boldsymbol{x}_2 = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

Das folgende Programm

```
program eig_all3;
{ Einschliessung bzw. Approximation aller einfachen bzw. }
{ mehrfachen Eigenpaare einer hermiteschen Matrix A;     }
{ Indexbereich von A : imatrix[0..n-1,0..n-1];           }

use mvc_ari, eig_herm;
var n : integer;

procedure main(n : integer);
var A : cmatrix[0..n-1,0..n-1];

begin
   writeln('Hermitesche Matrix (Dim(A)=',n,') = ?');
   read(A);
   eig_all(A); { Alle Einschliessungen bzw. Approximationen }
end; { main }

begin
   repeat
      writeln('Dimension(A) = ?');
      read(n);
      main(n);
   until false;
end.
```

liefert mit den Eingabedaten der Matrix $\boldsymbol{A}$ das folgende Ablaufprotokoll mit den garantierten Einschließungen für die reellen, einfachen Eigenwerte und die komplexen Eigenvektoren:

```
Dimension(A) = ?
3                                 <enter>
Hermitesche Matrix (Dim(A)=3) = ?
   1   (0,1)   1                  <enter>
(0,-1)   2   (0,-1)               <enter>
   1   (0,1)   1                  <enter>
  *********************************************************
  *  Berechnung aller Eigenpaare einer hermiteschen Matrix.  *
  *  Mehrfache Eigenwerte u. -Vektoren werden approximiert,  *
  *  Einfache Eigenwerte u. -Vektoren werden eingeschlossen  *
  *  Indexbereich der (n x n)-Matrix:    [0..n-1,0..n-1]     *
  *********************************************************

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
1-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[  3.414213562373E+000,  3.414213562374E+000 ]
Einschliessung des zugehoerigen Eigenvektors:

( [ -1.000000000000E-267,  1.000000000000E-267 ],
  [  7.071067811865E-001,  7.071067811866E-001 ] )
( [  1.000000000000E+000,  1.000000000000E+000 ],
  [  0.000000000000E+000,  0.000000000000E+000 ] )
( [ -1.000000000000E-267,  1.000000000000E-267 ],
  [  7.071067811865E-001,  7.071067811866E-001 ] )

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
2-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[ -1.000000000000E-049,  3.000000000000E-049 ]
Einschliessung des zugehoerigen Eigenvektors:

( [ -1.000000000001E+000, -9.999999999999E-001 ],
  [ -1.000000000000E-267,  1.000000000000E-267 ] )
( [ -1.000000000000E-267,  1.000000000000E-267 ],
  [ -4.000000000000E-050,  2.700000000000E-049 ] )
( [  1.000000000000E+000,  1.000000000000E+000 ],
  [  0.000000000000E+000,  0.000000000000E+000 ] )

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
3-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[  5.857864376269E-001,  5.857864376270E-001 ]
```

```
Einschliessung des zugehoerigen Eigenvektors:

( [ -1.000000000000E-267,  1.000000000000E-267 ],
  [ -7.071067811866E-001, -7.071067811865E-001 ] )
( [  1.000000000000E+000,  1.000000000000E+000 ],
  [  0.000000000000E+000,  0.000000000000E+000 ] )
( [ -1.000000000000E-267,  1.000000000000E-267 ],
  [ -7.071067811866E-001, -7.071067811865E-001 ] )
```

4. Beispiel: $\boldsymbol{A}$: `cimatrix`

Wir betrachten jetzt die hermitesche Matrix $\boldsymbol{A} = \begin{pmatrix} \sqrt{2} & i & 1 \\ -i & \sqrt{2} & -i \\ 1 & i & \sqrt{2} \end{pmatrix}$ mit dem einfachen Eigenpaar $\lambda_2 = \sqrt{2} + 2,\ \boldsymbol{x}_2 = (i, 1, i)^T,$ und dem doppelten Eigenpaar

$$\lambda_1 = \sqrt{2} - 1,\ \ \boldsymbol{x}_1 = \begin{pmatrix} 1 \\ \frac{i}{2} \\ -\frac{1}{2} \end{pmatrix}, \qquad \lambda_3 = \sqrt{2} - 1,\ \ \boldsymbol{x}_3 = \begin{pmatrix} -\frac{i}{2} \\ 1 \\ -\frac{i}{2} \end{pmatrix};$$

Wegen $a_{1,1} = \sqrt{2}$ muß die Matrix $\boldsymbol{A}$ als komplexe Intervallmatrix deklariert werden, um $\sqrt{2}$ maximalgenau einschließen zu können; beachten Sie bitte in diesem Zusammenhang auch die Bemerkungen zu hermiteschen Matrizen von Seite 611 unten. Das folgende

```
program eig_all4;
{ Einschliessung bzw. Approximation aller einfachen bzw.  }
{ mehrfachen Eigenpaare einer Matrix A, wobei die Mittel- }
{ punktsmatrix von A hermitesch sein muss!                }
{ Indexbereich von A : cimatrix[0..n-1,0..n-1];           }

use mvci_ari, ieig_herm;
var n : integer;

procedure main(n : integer);
var A : cimatrix[0..n-1,0..n-1];

begin
   writeln('Komplexe Intervall-Matrix (Dim(A)=',n,') = ?');
   read(A);
```

```
   eig_all(A); { Alle Einschliessungen bzw. Approximationen }
end; { main }

begin
   repeat
      writeln('Dimension(A) = ?');
      read(n);
      main(n);
   until false;
end.
```

liefert mit den Daten der komplexen aber nicht-hermiteschen Intervall–Matrix `A : cimatrix` das folgende Ablaufprotokoll:

```
Dimension(A) = ?
3                                                   <enter>
Komplexe Intervall-Matrix (Dim(A)=3) = ?
[1.414213562373,1.414213562374]    (0,1)    1       <enter>
(0,-1) [1.414213562373,1.414213562374] (0,-1)       <enter>
1    (0,1)    [1.414213562373,1.414213562374]       <enter>

  **********************************************************
  *  Berechnung aller Eigenpaare einer komplexen Intervall-  *
  *  Matrix, deren Mittelpunktsmatrix hermitesch sein muss.  *
  *  Mehrfache Eigenwerte u. -Vektoren werden approximiert,  *
  *  Einfache Eigenwerte u. -Vektoren werden eingeschlossen  *
  *  Indexbereich der (n x n)-Matrix:    [0..n-1,0..n-1]     *
  **********************************************************

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
Keine Einschliessung eines Eigenpaares moeglich:
Approximation des 1-ten reellen Eigenwerts:
 4.142135623740E-001
Approximation des zugehoerigen Eigenvektors:

(  1.000000000000E+000, -0.000000000000E+000 )
( -0.000000000000E+000,  5.000000000000E-001 )
( -5.000000000000E-001, -0.000000000000E+000 )

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
2-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[  3.414213562372E+000,  3.414213562375E+000 ]
```

```
Einschliessung des zugehoerigen Eigenvektors:

( [ -1.000000000000E-267,  1.000000000000E-267 ],
  [  9.999999999996E-001,  1.000000000001E+000 ] )
( [  1.000000000000E+000,  1.000000000000E+000 ],
  [  0.000000000000E+000,  0.000000000000E+000 ] )
( [ -1.000000000000E-267,  1.000000000000E-267 ],
  [  9.999999999996E-001,  1.000000000001E+000 ] )

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
Keine Einschliessung eines Eigenpaares moeglich:
Approximation des 3-ten reellen Eigenwerts:
 4.142135623740E-001
Approximation des zugehoerigen Eigenvektors:

( -0.000000000000E+000, -5.000000000000E-001 )
(  1.000000000000E+000, -0.000000000000E+000 )
( -0.000000000000E+000, -5.000000000000E-001 )
```

Auch dieses Beispiel zeigt, daß mehrfache Eigenpaare nicht eingeschlossen sondern nur approximiert werden können, vergl. auch das Beispiel von Seite 612.

5. Beispiel: ***A*** : `rmatrix`, Boothroyd/Dekker–Matrizen

Alle bisherigen Beispiele wurden bewußt einfach gewählt, um die Einschließungen oder Approximationen der exakten Eigenpaare möglichst übersichtlich darstellen zu können. Dieses Beispiel zeigt, daß alle einfachen Eigenpaare der sehr schlecht konditionierten Boothroyd/Dekker–Matrizen im 13-stelligen `real`–Format sogar bis zur Dimension $n = 10$ noch erfolgreich eingeschlossen werden. Informationen zu den Dekker–Matrizen findet man auf Seite 164. Das folgende

```
program eig_all5;
{ Einschliessung der einfachen Eigenpaare einer Boothroyd/ }
{ Dekker-Matrix bis zur maximalen Dimension n = 16;        }
{ Indexbereich von B : rmatrix[0..n-1,0..n-1];             }

use mvci_ari, eig_reell, mv_exap;
var n : integer;

procedure main(n : integer);
var B : rmatrix[0..n-1,0..n-1];
   v  : rvector[0..n-1];
```

```
begin
   Dekker(B,v); { v wird nicht benoetigt! }
   eig_all(B);  { Berechnung alle Einschliessungen }
end; { main }

begin
   repeat
      writeln('Dimension der Dekker-Matrix = ?');
      read(n);
      main(n);
   until false;
end.
```

liefert mit dem folgenden Ablaufprotokoll zur Dimension $n = 10$ die Einschließungen des ersten und zehnten Eigenpaares:

```
Dimension der Dekker-Matrix = ?
10
  ***********************************************************
  *  Berechnung aller Eigenpaare einer reellen (n x n)Matrix  *
  *  Mehrfache Eigenwerte u. -Vektoren werden approximiert,   *
  *  Einfache  Eigenwerte u. -Vektoren werden eingeschlossen  *
  *  Indexbereich der (n x n)-Matrix:    [0..n-1,0..n-1]      *
  ***********************************************************

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
1-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[  1.768842240173E+006,  1.768842240174E+006 ]
Einschliessung des zugehoerigen Eigenvektors:

[  2.482472850131E-005,  2.482472850132E-005 ]
[  2.382156989212E-004,  2.382156989213E-004 ]
[  1.268748856330E-003,  1.268748856331E-003 ]
[  4.945521307513E-003,  4.945521307514E-003 ]
[  1.573456255322E-002,  1.573456255323E-002 ]
[  4.327747602275E-002,  4.327747602276E-002 ]
[  1.065619232605E-001,  1.065619232606E-001 ]
[  2.403992231369E-001,  2.403992231370E-001 ]
[  5.050475522265E-001,  5.050475522266E-001 ]
[  1.000000000000E+000,  1.000000000000E+000 ]

Ausgabe des naechsten Eigen-Paares (y,n) ?
n
Ausgabe des naechsten Eigen-Paares (y,n) ?
```

```
n
Ausgabe des naechsten Eigen-Paares (y,n) ?
n
Ausgabe des naechsten Eigen-Paares (y,n) ?
n
Ausgabe des naechsten Eigen-Paares (y,n) ?
n
Ausgabe des naechsten Eigen-Paares (y,n) ?
n
Ausgabe des naechsten Eigen-Paares (y,n) ?
n
Ausgabe des naechsten Eigen-Paares (y,n) ?
n
Ausgabe des naechsten Eigen-Paares (y,n) ?
y
10-ter Eigenwert(EW):
Einschliessung eines reellen EW:
[  5.653415422180E-007,  5.653415422181E-007 ]
Einschliessung des zugehoerigen Eigenvektors:

[ -2.482472850132E-005, -2.482472850131E-005 ]
[  2.382156989212E-004,  2.382156989213E-004 ]
[ -1.268748856331E-003, -1.268748856330E-003 ]
[  4.945521307513E-003,  4.945521307514E-003 ]
[ -1.573456255323E-002, -1.573456255322E-002 ]
[  4.327747602275E-002,  4.327747602276E-002 ]
[ -1.065619232606E-001, -1.065619232605E-001 ]
[  2.403992231369E-001,  2.403992231370E-001 ]
[ -5.050475522266E-001, -5.050475522265E-001 ]
[  1.000000000000E+000,  1.000000000000E+000 ]
```

Nach Seite 164 gilt $\lambda_1 \cdot \lambda_{10} = 1$, so daß das Produkt der beiden Eigenwerteinschließungen den Wert 1 enthalten muß, was sich leicht bestätigen läßt.

6. Beispiel: $\boldsymbol{A}$: `rmatrix`, Zufalls–Matrizen

Im nächsten Beispiel betrachten wir eine reelle Zufallsmatrix $\boldsymbol{A}$ der Dimension $n = 50$, die mit Hilfe der Prozedur `Rand_M` von Seite 40 berechnet wird. Damit soll gezeigt werden, daß in den bisherigen Beispielen die Einschließung einfacher Eigenpaare oder die Approximation mehrfacher Eigenpaare auch noch für größere Dimensionen möglich ist, falls die Matrizen $\boldsymbol{A}$ nicht zu schlecht konditioniert sind.

Das nachfolgende Programm berechnet zu einer Zufallsmatrix, mit Elementen aus dem offenen Intervall $(-10^3, +10^3)$, Einschließungen bzw. Approximationen aller einfachen bzw. mehrfachen Eigenpaare. Aus Platzgründen wird nur das erste Eigenpaar am Bildschirm ausgegeben, wobei auch nur die ersten und letzten Komponenten des Eigenvektors gedruckt werden.

```
program eig_all6;
{ Einschliessung der einfachen Eigenpaare einer reellen }
{ Zufalls-Matrix A der Dimension n;                     }
{ Indexbereich von A : rmatrix[0..n-1,0..n-1];          }

use mvci_ari, eig_reell, mvran;
var n : integer;

procedure main(n : integer);
var A : rmatrix[0..n-1,0..n-1];
begin
   Rand_M(A,-1e3,+1e3);
   eig_all(A); { Berechnung alle Einschliessungen }
end; { main }

begin
   repeat
      writeln('Dimension der reellen Zufallsmatrix = ?');
      read(n);
      main(n);
   until false;
end.
```

Mit der Eingabe der Dimension $n = 50$ erhält man das folgende Ablaufprotokoll:

```
Dimension der reellen Zufallsmatrix = ?
50   <enter>
  ************************************************************
  *  Berechnung aller Eigenpaare einer reellen (n x n)Matrix  *
  *  Mehrfache Eigenwerte u. -Vektoren werden approximiert,   *
  *  Einfache  Eigenwerte u. -Vektoren werden eingeschlossen  *
  *  Indexbereich der (n x n)-Matrix:    [0..n-1,0..n-1]      *
  ************************************************************

Ausgabe des naechsten Eigen-Paares (y,n) ?
y
1-ter Eigenwert(EW):
Einschliessung eines komplexen EW:
( [  4.935861392405E+002,  4.935861392406E+002 ],
  [  4.178821852143E+003,  4.178821852144E+003 ] )
```

```
Einschliessung des zugehoerigen Eigenvektors:

( [  2.267726877325E-001,  2.267726877326E-001 ],
  [  1.796195753020E-001,  1.796195753021E-001 ] )
( [ -2.915106996114E-002, -2.915106996113E-002 ],
  [  5.457078122581E-001,  5.457078122582E-001 ] )
( [  2.079980603835E-001,  2.079980603836E-001 ],
  [ -1.986837118674E-001, -1.986837118673E-001 ] )
( [ -5.634183774693E-001, -5.634183774692E-001 ],
  [  2.410980686991E-001,  2.410980686992E-001 ] )
( [ -1.496604540063E-001, -1.496604540062E-001 ],
  [  4.454697201545E-001,  4.454697201546E-001 ] )
( [  2.217574511571E-001,  2.217574511572E-001 ],
  [ -1.451620809794E-001, -1.451620809793E-001 ] )
( [ -7.296897313383E-002, -7.296897313382E-002 ],
  [ -3.578569957090E-002, -3.578569957089E-002 ] )
( [ -8.141461956548E-002, -8.141461956547E-002 ],
  [ -4.132507571044E-001, -4.132507571043E-001 ] )

                         .
                         .
                         .

( [ -2.841992287959E-001, -2.841992287958E-001 ],
  [ -5.518582500156E-002, -5.518582500155E-002 ] )
( [  3.314557437327E-001,  3.314557437328E-001 ],
  [ -1.642638765459E-001, -1.642638765458E-001 ] )
( [  4.267554877528E-001,  4.267554877529E-001 ],
  [  1.094560743801E-001,  1.094560743802E-001 ] )
( [ -2.977201091123E-001, -2.977201091122E-001 ],
  [ -5.057961723804E-001, -5.057961723803E-001 ] )
( [  2.995107296597E-004,  2.995107296598E-004 ],
  [  5.474182586217E-001,  5.474182586218E-001 ] )
( [  1.874770542145E-001,  1.874770542146E-001 ],
  [ -3.784339656565E-001, -3.784339656564E-001 ] )
( [ -3.417139183013E-001, -3.417139183012E-001 ],
  [  1.628825665757E-001,  1.628825665758E-001 ] )
( [  7.224501476130E-002,  7.224501476131E-002 ],
  [  1.904976652583E-001,  1.904976652584E-001 ] )

Ausgabe des naechsten Eigen-Paares (y,n) ?
```

Man erkennt, daß neben dem Eigenwert auch die Komponenten des zugehörigen Eigenvektors sogar maximalgenau eingeschlossen werden.

Bis zur maximalen Dimension $n = 50$ liefert die Prozedur `eig_all(A)` damit sehr zuverlässige Einschließungen bzw. Approximationen der einfachen bzw. mehrfachen Eigenpaare, wenn $\boldsymbol{A}$ nicht zu schlecht konditioniert ist.

14.2.4 Eigenwerte reeller, symmetrischer Matrizen

Für das symmetrische Eigenwertproblem

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}, \quad \boldsymbol{A}^T = \boldsymbol{A} \qquad \text{liefert}$$

procedure `seig`(**var** A : ■ ; **var** EW : ◇);

zur $(n \times n)$ Eingangsmatrix `A` mit dem Ausgangsvektor `EW` Einschließungen aller reellen Eigenwerte $\lambda_i \in$ `EW[i]` von $\boldsymbol{A}$; $\quad i = 1, 2, \ldots, n = \mathrm{Dim}(A)$.

Erlaubte Datentypen in Prozedur `seig`				
`A : ■`	`rmatrix`	`Lrmatrix`	`imatrix`	`Limatrix`
`EW : ◇`	`ivector`	`Livector`	`ivector`	`Livector`
exportierende Module	seig	lseig	iseig	liseig

Die Indexbereiche von `A` und `EW` müssen stets mit 1 beginnen, sonst erfolgt eine entsprechende Fehlermeldung mit Programmabbruch.

Beachten Sie bitte, daß eine Intervallmatrix nicht symmetrisch sein kann; für eine Matrix $\boldsymbol{A}$ vom Typ `(L)imatrix` werden daher Einschließungen der Eigenwerte λ_i nur dann erfolgreich berechnet, wenn die Mittelpunktsmatrix von `A` symmetrisch ist. Eine solche symmetrische Mittelpunktsmatrix erhält man z.B. dann, wenn irrationale Elemente $a_{i,j}$ einer symmetrischen Matrix $\boldsymbol{A}$ bei der Eingabe durch maximalgenaue Intervalle eingeschlossen werden, und zu dieser Intervallmatrix im Rechner die Mittelpunktsmatrix bestimmt wird[2]. Ein Symmetrietest wird in der Prozedur `seig` für die entsprechenden Punktmatrizen stets durchgeführt. Ein Beispiel zur Behandlung irrationaler Matrixelemente findet man auf Seite 630.

Beachten Sie ferner, daß jetzt mit Hilfe der obigen Prozedur `seig` auch mehrfache Eigenwerte erfolgreich eingeschlossen werden! Betrachten Sie dazu das Beispiel auf Seite 626. Hinweise zum verwendeten Algorithmus zur Einschließung der Eigenwerte einer reellen, symmetrischen Matrix findet man in [8].

[2] Beachten Sie bitte, daß der intern berechnete Mittelpunkt eines Intervalls, welches eine irrationale Zahl maximalgenau einschließt, entweder der rechte oder linke Intervall–Randpunkt ist.

14.2.4.1 Programmbeispiele

1. Beispiel: Mehrfache Eigenwerte

Wir betrachten die symmetrische Matrix $\boldsymbol{A} = \begin{pmatrix} 5 & -2 & -4 \\ -2 & 2 & 2 \\ -4 & 2 & 5 \end{pmatrix}$ mit dem Eigenwert $\lambda_1 = 10$ und den doppelten Eigenwerten $\lambda_{2,3} = 1$. Das folgende

```
program lseig_ex2;
{ Enclosures of all Eigenvalues of a real, symmetric matrix }
use li_ari, lmv_ari, lmvi_ari, lseig;
const stars = '******************';
var n : integer;

procedure main(n : integer);
var i   : integer;
   A   : Lrmatrix[1..N,1..N];
   IEW : Livector[1..N];

begin
   writeln('Matrix (dimension=',n,') = ?');  read(A);
   seig(A,IEW);  writeln;
   write('All Eigenvalue enclosures of a ');
   writeln('symmetric matrix:');
   for i := 1 to n do
      writeln(i:3, ': ',IEW[i]);
end; { main }

begin
   writeln('  ***************************************'+stars);
   write  ('  * Enclosing all Eigenvalues of a real, ');
   writeln('symmetric matrix *');
   writeln('  ***************************************'+stars);
   repeat
      writeln;
      write('Matrix dimension = ?  ');  read(n);
      if n > 1 then main(n);
   until n=0;
end.
```

liefert mit den Eingabedaten der Matrix $\boldsymbol{A}$ das Ablaufprotokoll

```
   *******************************************************
   * Enclosing all Eigenvalues of a real, symmetric matrix *
   *******************************************************

 Matrix dimension = ?  3    <enter>
 Matrix (dimension=3) = ?
  5 -2 -4                   <enter>
 -2  2  2                   <enter>
 -4  2  5                   <enter>

 All Eigenvalue enclosures of a symmetric matrix:
   1: [  9.99999999999999999999E+000,  1.00000000000000000001E+001 ]
   2: [  9.99999999999999999999E-001,  1.00000000000000000001E+000 ]
   3: [  1.00000000000000000000E+000,  1.00000000000000000000E+000 ]
```

mit den garantierten Einschließungen der Eigenwerte $\lambda_1 = 10,\ \lambda_{2,3} = 1$.

2. Beispiel: Zielke Matrizen

Wir betrachten die modifizierten, reellen Zielke–Matrizen der Form:

$$\boldsymbol{Z}(a,b,c) := \begin{pmatrix} a+b & a+b & \cdot & \cdot & a+b & a \\ a+b & \cdot & \cdot & a+b & a & \cdot \\ \cdot & \cdot & \cdot & a & \cdot & \cdot \\ \cdot & a+b & \cdot & \cdot & \cdot & \cdot \\ a+b & a & \cdot & \cdot & \cdot & a \\ a & \cdot & \cdot & \cdot & a & a-c \end{pmatrix}, \qquad a,b,c \in I\!R,$$

die für $|a| \gg |b|, |c|$ schlecht konditioniert sind, vergl. Seite 166. Das nachfolgende Programm `lseig_ex1` berechnet Einschließungen der Eigenwerte von $\boldsymbol{Z} \in I\!R^{(n,n)}$ zu vorgegebenen Werten $a, b, c \in I\!R$:

```
program lseig_ex1;
{ Einschliessgn. aller reellen Eigenwerte einer Zielke-Matrix }

use li_ari, lmv_ari, lmvi_ari, lseig, lmv_exap;

var n : integer;

procedure main(n : integer);
var i    : integer;
```

```
   a,b,c : Longreal;
   Z     : Lrmatrix[1..N,1..N];
   IEW   : Livector[1..N];

begin
   write('a,b,c = ?  ');  read(a,b,c);
   Zielke(Z,a,b,c);           { Berechnet Zielke-Matrix }
   seig(Z,IEW);       { Einschliessung aller Eigenwerte }
   writeln('Eigenvalues of a symmetric Zielke matrix:');
   for i := 1 to n do
      writeln(i:3, ': ',IEW[i]);
end; { main }

begin
   writeln('Enclosures of all Eigenvalues of Zielke matrices');
   repeat
      write('Dimension of Zielke Matrix = ?  ');  read(n);
      if n > 1 then main(n);
   until n=0;
end.
```

Mit den Eingabewerten $n = 12, a = 10^{20}, b = c = 20$ liefert das folgende Ablaufprotokoll Einschließungen der 12 reellen Eigenwerte:

```
Enclosures of all Eigenvalues of Zielke matrices
Dimension of Zielke Matrix = ?  12
a,b,c = ?  1e20 20 20
Eigenvalue enclosures of the Zielke matrix:
  1: [  1.20000000000000000010E+021,  1.20000000000000000011E+021 ]
  2: [  3.73386176590640860909E+001,  3.73386176590640860910E+001 ]
  3: [  1.01076938603239353990E+001,  1.01076938603239353992E+001 ]
  4: [  1.34038903687599319968E+001,  1.34038903687599319969E+001 ]
  5: [  1.90097403538172133340E+001,  1.90097403538172133341E+001 ]
  6: [  1.10422709956830960516E+001,  1.10422709956830960518E+001 ]
  7: [ -4.05248892979193139283E+001, -4.05248892979193139282E+001 ]
  8: [ -1.04204943407622417510E+001, -1.04204943407622417509E+001 ]
  9: [ -2.15055050585602423352E+001, -2.15055050585602423351E+001 ]
 10: [ -1.49347234588783574450E+001, -1.49347234588783574449E+001 ]
 11: [ -1.18499344148614407491E+001, -1.18499344148614407490E+001 ]
 12: [ -2.00289582727466599991E-018, -1.99710417272533399989E-018 ]
```

Auffällig ist der betragsmäßig sehr kleine 12-te Eigenwert, der jedoch um ganze 19 Zehnerexponenten vergrößert wird, wenn man $c = 0$ wählt. Das folgende Ablaufprotokoll liefert mit den entsprechenden Eingabedaten die Einschließungen der zum Teil sehr stark geänderten Eigenwerte von $\boldsymbol{Z}$:

```
Enclosures of all Eigenvalues of Zielke matrices
Dimension of Zielke Matrix = ?  12
a,b,c = ?  1e20 20 0
Eigenvalue enclosures of the Zielke matrix:
  1: [  1.2000000000000000011E+021,  1.2000000000000000012E+021 ]
  2: [  3.86370330515627314684E+001,  3.86370330515627314685E+001 ]
  3: [  1.03527618041008304937E+001,  1.03527618041008304938E+001 ]
  4: [  1.15470053837925152899E+001,  1.15470053837925152900E+001 ]
  5: [  1.99999999999999999994E+001,  1.99999999999999999996E+001 ]
  6: [  9.9999999999999999991E+000,  9.9999999999999999992E+000 ]
  7: [ -3.86370330515627314710E+001, -3.86370330515627314709E+001 ]
  8: [ -1.03527618041008304940E+001, -1.03527618041008304939E+001 ]
  9: [ -2.00000000000000000002E+001, -2.00000000000000000001E+001 ]
 10: [ -1.41421356237309504881E+001, -1.41421356237309504880E+001 ]
 11: [ -1.15470053837925152903E+001, -1.15470053837925152901E+001 ]
 12: [  1.41421356237309504877E+001,  1.41421356237309504878E+001 ]
```

Beachten Sie bitte, daß die mit $c = 0$ ganz minimale Änderung nur des einen Elements $z_{12,12}$ den 12-ten Eigenwert um 19 Größenordnungen verändert! Das Beispiel zeigt sehr eindrucksvoll die bekannte starke Sensibilität der Eigenwerte gegenüber nur minimalen Änderungen der Matrixelemente bei sehr schlecht konditionierten Eigenwertproblemen.

Besonders in diesen Fällen werden die unvermeidbaren Rundungsfehler den Wert der berechneten Eigenwerte stark beeinflussen, so daß erst die in der **PASCAL-XSC** Umgebung bestimmten Eigenwert–Einschließungen ausreichende Informationen über die numerische Stabilität der Eigenwertberechnungen liefern. Beachten Sie bitte, daß der 12-te Eigenwert der viel schlechter konditionierten Zielke Matrix $\boldsymbol{Z}$ mit $a = 10^{20}, b = c = 20$ sehr viel gröber eingeschlossen wird als der 12-te Eigenwert bzgl. der Parameter $a = 10^{20}, b = 20, c = 0$.

Abschließend sollen noch die Konditionszahlen $\varkappa(\boldsymbol{Z}) = \frac{\max|\lambda_i|}{\min|\lambda_i|}$ der beiden symmetrischen Zielke-Matrizen bzgl. der Spektralnorm angegeben werden [19, Seite 34,35]:

$$\begin{aligned}\varkappa(\,\boldsymbol{Z}(10^{20},20,20)\,) &\approx \frac{1.2\cdot 10^{21}}{2\cdot 10^{-18}} = 6\cdot 10^{+38}\\ \varkappa(\,\boldsymbol{Z}(10^{20},20,0)\,) &\approx \frac{1.2\cdot 10^{21}}{1} = 1.2\cdot 10^{+21}\end{aligned}$$

Man erkennt, daß $\boldsymbol{Z}(10^{20}, 20, 20)$ wegen der größeren Konditionszahl deutlich schlechter konditioniert ist als $\boldsymbol{Z}(10^{20}, 20, 0)$.

3. Beispiel: Irrationale Matrix–Elemente $a_{i,j}$

Wir betrachten jetzt die symmetrische Matrix $\boldsymbol{A} = \begin{pmatrix} \sqrt{2} & 1 & 1 \\ 1 & \sqrt{2} & 0 \\ 1 & 0 & \sqrt{2} \end{pmatrix}$ mit den Eigenwerten $\lambda_1 = 2\sqrt{2}$, $\lambda_2 = 0$, $\lambda_3 = \sqrt{2}$. Um diese Eigenwerte korrekt einzuschließen, müssen die Diagonalelemente $a_{ii} = \sqrt{2}$ durch Intervalle maximalgenau eingeschlossen werden, d.h. die gegebene Matrix $\boldsymbol{A}$ muß im Programm als Intervallmatrix `A : Limatrix` deklariert werden. Beachten Sie bitte, daß diese Intervallmatrix auch unsymmetrische Matrizen enthält, daß aber die intern zu berechnende Mittelpunktsmatrix selbst wieder symmetrisch ist, vergleichen Sie dazu auch die Bemerkungen von Seit 625. Das folgende Programm

```
program tstliseig1;
{ Einschliessung aller Eigenwerte einer symmetrischen Matrix }
{ mit irrationalen Matrix-Elementen.                         }

use li_ari, lmv_ari, lmvi_ari, liseig;
const stars = '**********************';
var n : integer;

procedure main(n : integer);
var i    : integer;
   A     : Limatrix[1..n,1..n];
   IEW   : Livector[1..n];

begin
   writeln('Interval matrix = ?');  read(A);
   seig(A,IEW);
   writeln;
   writeln('Enclosures of all Eigenvalues:');
   for i := 1 to n do writeln(i:3, ': ',IEW[i]);
end; { main }

begin
   writeln('  ****************************************'+stars);
   write  ('  * Eigenvalues of symmetric matrices with ');
   writeln(' irrational elements *');
   writeln('  ****************************************'+stars);
```

```
   repeat
      write('Matrix dimension = ?  ');  read(n);
      if n > 1 then main(n);
   until n=0;
end.
```

liefert mit den entsprechenden Eingabedaten für die Intervallmatrix `A` das Ablaufprotokoll

```
     ************************************************************
     * Eigenvalues of symmetric matrices with  irrational elements *
     ************************************************************

   Matrix dimension = ?  3  <enter>
   Interval matrix = ?
   [1.41421356237309504880,1.41421356237309504881] 1 1   <enter>
   1 [1.41421356237309504880,1.41421356237309504881] 0   <enter>
   1 0 [1.41421356237309504880,1.41421356237309504881]   <enter>

   Enclosures of all Eigenvalues:
     1: [  2.82842712474619009759E+000,  2.82842712474619009763E+000 ]
     2: [ -1.02242581156310785697E-020,  1.68468096962349214305E-020 ]
     3: [  1.41421356237309504879E+000,  1.41421356237309504882E+000 ]
```

mit den Einschließungen der Eigenwerte $\lambda_1 = 2\sqrt{2}$, $\lambda_2 = 0$, $\lambda_3 = \sqrt{2}$.

14.2.5 Eigenwerte hermitescher Matrizen

Für das Eigenwertproblem

$$\boldsymbol{Ax} = \lambda \boldsymbol{x}, \quad \boldsymbol{A}^H = \boldsymbol{A} \qquad \text{einer hermiteschen Matrix } \boldsymbol{A} \text{ liefert}$$

$$\boxed{\textbf{procedure } \texttt{heig}(\ \textbf{var}\ \mathrm{A} : \blacksquare;\ \textbf{var}\ \mathrm{EW} : \diamond\ );}$$

zur $(n \times n)$ Eingangsmatrix `A` mit dem Ausgangsvektor `EW` Einschließungen aller reellen Eigenwerte $\lambda_i \in$ `EW[i]` von $\boldsymbol{A}$; $\quad i = 1, 2, \ldots, n = \mathrm{Dim}(A)$.

Erlaubte Datentypen in Prozedur `heig`				
`A` : ■	`cmatrix`	`Lcmatrix`	`cimatrix`	`Lcimatrix`
`EW` : ◇	`ivector`	`Livector`	`ivector`	`Livector`
exportierende Module	heig	lheig	iheig	liheig

Die Indexbereiche von `A` und `EW` müssen stets mit 1 beginnen, sonst erfolgt eine entsprechende Fehlermeldung mit Programmabbruch.

Beachten Sie bitte, daß eine komplexe Intervallmatrix nicht hermitesch sein kann; für eine Matrix `A` vom Typ `(L)cimatrix` werden daher Einschließungen der Eigenwerte λ_i nur dann erfolgreich berechnet, wenn die Mittelpunktsmatrix von `A` hermitesch ist. Eine solche hermitesche Mittelpunktsmatrix erhält man z.B. dann, wenn irrationale Elemente $a_{i,j}$ einer hermiteschen Matrix $\boldsymbol{A}$ bei der Eingabe durch maximalgenaue Intervalle eingeschlossen werden, und zu dieser Intervallmatrix im Rechner die Mittelpunktsmatrix bestimmt wird.

Intern wird die Berechnung der reellen Eigenwerte von $\boldsymbol{Ax} = \lambda\boldsymbol{x}$ zurückgeführt auf das reelle $(2n \times 2n)$ Eigenwertproblem

$$\begin{pmatrix} \Re(\boldsymbol{A}) & -\Im(\boldsymbol{A}) \\ \Im(\boldsymbol{A}) & \Re(\boldsymbol{A}) \end{pmatrix} \cdot \boldsymbol{y} = \lambda \cdot \boldsymbol{y}, \qquad \boldsymbol{y} := \begin{pmatrix} \Re(\boldsymbol{x}) \\ \Im(\boldsymbol{x}) \end{pmatrix}$$

Die reelle $(2n \times 2n)$ Matrix ist dabei symmetrisch, und die jeweils doppelt auftretenden Eigenwerte λ_i sind die gesuchten Eigenwerte des hermiteschen Eigenwertproblems. In der Prozedur `heig(...)` wird ein Symmetrietest der $(2n \times 2n)$ Matrix stets durchgeführt. Hinweise zum verwendeten Algorithmus zur Einschließung der Eigenwerte einer reellen, symmetrischen Matrix findet man in [8].

Beachten Sie bitte, daß jetzt mit der obigen Prozedur `heig` auch mehrfache Eigenwerte des hermiteschen Eigenwertproblems erfolgreich eingeschlossen werden! Ein Beispiel zur Behandlung irrationaler Matrixelemente findet man auf Seite 635.

14.2.5.1 Programmbeispiele

1. Beispiel: `A : Lcmatrix;`

Wir betrachten die hermitesche Matrix $\boldsymbol{A} = \begin{pmatrix} 1 & i & i & 2+i \\ -i & 1 & i & i \\ -i & -i & 1 & i \\ 2-i & -i & -i & 1 \end{pmatrix}$ mit den reellen Eigenwerten: $\lambda_1 = 2+\sqrt{2}$, $\lambda_2 = \sqrt{6}$, $\lambda_3 = -\sqrt{6}$, $\lambda_4 = 2-\sqrt{2}$.
Mit dem **PASCAL-XSC** Programm

```
program lheig_ex1;
{ Einschliessung aller Eigenwerte einer hermiteschen Matrix A }
{ Indexbereich von A : Lcmatrix[1..n,1..n];                   }

use lmvc_ari, lheig, li_ari;

const stars = '**************************';

var n : integer;

procedure main(n : integer);
var A   : Lcmatrix[1..n,1..n];
    IEW : Livector[1..n];
    i   : integer;

begin
   writeln('Hermitesche Matrix (Dim(A)=',n,') = ?');
   read(A);
   heig(A,IEW); { Alle Eigenwert-Einschliessungen }
   writeln;
   writeln('Einschliessung aller Eigenwerte:');
   for i := 1 to n do writeln(i:3,'.  ',IEW[i]);
end; { main }
```

```
begin
   writeln('  ***********************************'+stars);
   write  ('  *  Einschliessung aller Eigenwerte einer ');
   writeln('hermiteschen Matrix  *');
   writeln('  ***********************************'+stars);
   writeln;
   repeat
      writeln('Dimension(A) = ?');
      read(n);
      main(n);
   until false;
end.
```

erhält man mit den entsprechenden Eingabedaten bzgl. der Matrix $\boldsymbol{A}$ das folgende Ablaufprotokoll

```
     ***************************************************************
     *  Einschliessung aller Eigenwerte einer hermiteschen Matrix  *
     ***************************************************************

   Dimension(A) = ?
   4   <enter>
   Hermitesche Matrix (Dim(A)=4) = ?
      1   (0,1)  (0,1) (2,1)       <enter>
   (0,-1)   1    (0,1) (0,1)       <enter>
   (0,-1) (0,-1)   1   (0,1)       <enter>
   (2,-1) (0,-1) (0,-1)  1         <enter>

   Einschliessung aller reellen Eigenwerte:
     1.  [  3.41421356237309504880E+000,  3.41421356237309504881E+000 ]
     2.  [  2.44948974278317809819E+000,  2.44948974278317809820E+000 ]
     3.  [ -2.44948974278317809820E+000, -2.44948974278317809819E+000 ]
     4.  [  5.85786437626904951198E-001,  5.85786437626904951199E-001 ]
```

mit den maximalgenauen Einschließungen der Eigenwerte: $\lambda_1 = 2 + \sqrt{2}$, $\lambda_2 = \sqrt{6}$, $\lambda_3 = -\sqrt{6}$, $\lambda_4 = 2 - \sqrt{2}$.

2. Beispiel: `A : Lcimatrix;` Irrationale Matrixelemente;

Wir betrachten die hermitesche Matrix $\boldsymbol{A} = \begin{pmatrix} \sqrt{2} & i & i & 2+i \\ -i & \sqrt{2} & i & i \\ -i & -i & \sqrt{2} & i \\ 2-i & -i & -i & \sqrt{2} \end{pmatrix}$ mit den Eigenwerten: $\lambda_1 = 1 + 2\sqrt{2}$, $\lambda_2 = -1 + \sqrt{2} + \sqrt{6}$, $\lambda_3 = -1 + \sqrt{2} - \sqrt{6}$, $\lambda_4 = 1$. Um diese Eigenwerte korrekt einzuschließen, müssen die Elemente $a_{ii} = \sqrt{2}$ durch Intervalle maximalgenau eingeschlossen werden, d.h. die gegebene Matrix $\boldsymbol{A}$ muß im Programm als Intervallmatrix `A : Lcimatrix` deklariert werden. Beachten Sie bitte, daß diese Intervallmatrix auch nicht-hermitesche Matrizen enthält, daß aber die intern zu berechnende Mittelpunktsmatrix selbst wieder hermitesch ist. Das folgende Programm

```
program tstliheig1;
{ Einschliessung aller Eigenwerte einer hermiteschen Matrix A }
{ mit irrationalen Matrixelementen.                           }
{ Indexbereich von A : Lcimatrix[1..n,1..n];                  }

use lmvci_ari, liheig, li_ari;

const stars = '**************************';

var n : integer;

procedure main(n : integer);
var A   : Lcimatrix[1..n,1..n];
    IEW : Livector[1..n];
    i   : integer;

begin
   writeln('Hermitesche Matrix (Dim(A)=',n,') = ?');
   read(A);
   heig(A,IEW); { Alle Eigenwert-Einschliessungen }
   writeln;
   writeln('Einschliessung aller reellen Eigenwerte:');
   for i := 1 to n do writeln(i:3,'.  ',IEW[i]);
end; { main }
```

```
begin
   writeln('  **********************************'+stars);
   write  ('  *  Einschliessung aller Eigenwerte einer ');
   writeln('hermiteschen Matrix  *');
   writeln('  **********************************'+stars);
   writeln;
   repeat
      writeln('Dimension(A) = ?');
      read(n);
      main(n);
   until false;
end.
```

liefert mit den entsprechenden Eingabedaten bzgl. der Matrix $\boldsymbol{A}$ das folgende Ablaufprotokoll:

```
  ************************************************************
  *  Einschliessung aller Eigenwerte einer hermiteschen Matrix  *
  ************************************************************

Dimension(A) = ?
4   <enter>
Hermitesche Matrix (Dim(A)=4) = ?
[1.41421356237309504880,1.41421356237309504881] (0,1)  (0,1)  (2,1)  <enter>
(0,-1) [1.41421356237309504880,1.41421356237309504881] (0,1)  (0,1)  <enter>
(0,-1) (0,-1) [1.41421356237309504880,1.41421356237309504881] (0,1)  <enter>
(2,-1) (0,-1) (0,-1) [1.41421356237309504880,1.41421356237309504881] <enter>

Einschliessung aller reellen Eigenwerte:
  1.  [  3.82842712474619009757E+000,  3.82842712474619009764E+000 ]
  2.  [  2.86370330515627314697E+000,  2.86370330515627314704E+000 ]
  3.  [ -2.03527618041008304943E+000, -2.03527618041008304935E+000 ]
  4.  [  9.99999999999999999974E-001,  1.00000000000000000004E+000 ]
```

mit den garantierten Einschließungen aller reellen Eigenwerte:

$$\lambda_1 = 1 + 2\sqrt{2}, \quad \lambda_2 = -1 + \sqrt{2} + \sqrt{6}, \quad \lambda_3 = -1 + \sqrt{2} - \sqrt{6}, \quad \lambda_4 = 1.$$

Anhang A

Spezielle Hilfsmodule

A.1 Modul `type_def`

Das Hilfsmodul `type_def` exportiert

1. alle Datentypen der beiden Tabellen ab Seite 663,
2. alle Variablen der Tabelle auf Seite 666,
3. Routinen zur Kontrolle der Präzisionsparameter:
   `maxprec, Stagg_prec, LStagg_prec`
4. Ein-/Ausgabe Prozeduren für den Typ `intvector`

A.1.1 Exportierte Datentypen

Die vom Hilfsmodul `type_def` exportierten Datentypen findet man in den beiden Tabellen ab Seite 663.

A.1.2 Ein-/Ausgabe Prozeduren für den Typ `intvector`

```
procedure read ( var F : Text; var A : intvector );
```

```
procedure write ( var F : Text; var A : intvector );
```

Zur Definition des Typs `intvector` siehe Seite 663.

A.1.3 Routinen zur Kontrolle der Präzisionsparameter

Mit Hilfe der folgenden Routinen, die vom Hilfsmodul `type_def` exportiert werden, lassen sich die `integer`–Werte der Präzisionsparameter `maxprec`, `Stagg_prec`, `LStagg_prec` darstellen bzw. verändern:

function `get_maxprec : integer;`

Vergleiche Seite 49

procedure `set_maxprec ( prec : integer );`

Vergleiche Seite 49

function `get_Stagg_prec : integer;`

Vergleiche Seite 66

procedure `set_Stagg_prec ( prec : integer );`

Vergleiche Seite 65

function `get_LStagg_prec : integer;`

Vergleiche Seite 91

procedure `set_LStagg_prec ( prec : integer );`

Vergleiche Seite 91

A.1.4 Exportierte Variable

Die vom Modul `type_def` exportierten Variablen `maxprec`, `Stagg_prec`, `LStagg_prec` findet man in der Tabelle auf Seite 666.

A.2 Modul `i_util`

Das Hilfsmodul `i_util` liefert Ergänzungen zu den Systemmodulen `i_ari`, `li_ari` und exportiert:

1. die folgenden Funktionen `AbsMin`, `AbsMax`[1], `RelDiam` vom Ergebnistyp `real|Longreal`
2. die Funktion `UlpAcc` vom Ergebnistyp `boolean`
3. und Einschließungen von 24 mathematischen Konstanten.

A.2.1 Funktionen vom Ergebnistyp `real | Longreal`

function `AbsMin (x : ■) : □;` (`i_util`)

■ = `interval` und □ = `real` oder

■ = `Linterval` und □ = `Longreal`

$$\texttt{AbsMin} = \mathrm{Min}\{|r| : r \in \mathbf{x}\}$$

function `AbsMax (x : interval) : real;` (`i_util`)

$$\texttt{AbsMax} = \mathrm{Max}\{|r| : r \in \mathbf{x}\}$$

[1] `function AbsMax(x : Linterval) : Longreal` wird vom Modul `li_ari` exportiert.

function `RelDiam (x : ■) : □;` (`i_util`)

■ = `interval` und □ = `real` oder

■ = `Linterval` und □ = `Longreal`

$$\texttt{RelDiam} = \begin{cases} \operatorname{diam}(x) & \text{falls } 0 \in \mathbf{x} \\ \operatorname{diam}(x) \mathbin{/>} \operatorname{AbsMin}(x) & \text{sonst} \end{cases}$$

A.2.2 Einschließung mathematischer Konstanten

Einschließung mathematischer Konstanten mit dem Datentyp `Linterval`			
Konstante	**Name**	**Konstante**	**Name**
π	`PI_Li`	2π	`PI2_Li`
$\pi/2$	`PI_HALF_Li`	$\pi/4$	`PI_DIV4_Li`
$1/\pi$	`R_PI_Li`	$1/(2\pi)$	`R_2PI_Li`
$\sqrt{\pi}$	`SQRT_PI_Li`	$\sqrt{\pi}/2$	`SQRT_PI_D2_Li`
$1/\sqrt{\pi}$	`R_SQRTPI_Li`	$2/\sqrt{\pi}$	`R_SQRTPI2_Li`
$1/\sqrt{2\pi}$	`R_SQRT2PI_Li`	e	`EUL_Li`
$\sqrt{e}$	`SQRT_E_Li`	e^{π}	`E_PI_Li`
$\ln(2)$	`LN_2_Li`	$1/\ln(2)$	`R_LN_2_Li`
$\ln(10)$	`LN_10_Li`	$1/\ln(10)$	`R_LN_10_Li`
$\sqrt{2}$	`SQRT2_Li`	$1/\sqrt{2}$	`R_SQRT2_Li`
$\sqrt{3}$	`SQRT3_Li`	$\sqrt{5}$	`SQRT5_Li`
$\sqrt{10}$	`SQRT10_Li`	$\ln(\pi)$	`LN_PI_Li`
C	`GAMMA_C_Li`	$\ln(C)$	`LN_GAM_C_Li`

A.2.3 Funktionen vom Ergebnistyp `boolean`

function `UlpAcc (x : ■; n : integer) : boolean;` (`i_util`)

■ = `interval | Linterval`

`UlpAcc(x,n)` testet, ob sich Unter- und Oberschranke von `x` um höchstens n ulp unterscheiden.

1 ulp: Eine Einheit auf der letzten Mantissenstelle.

Das Hilfsmodul `i_util` importiert die folgenden Module:

`i_ari, li_ari`

A.3 Modul `int_mv`

Das Hilfsmodul `int_mv` exportiert:

procedure `read ( var F : Text; var p : intvector );`

procedure `write ( var F : Text; p : intvector );`

A.4 Modul `cpoly`

Das Hilfsmodul `cpoly` exportiert:

operator `:= (`**var** `p : CPolynomial; r : real);`

A.5 Modul `cipoly`

Das Hilfsmodul `cipoly` exportiert:

operator `:=` (**var** `p : CIPolynomial; r : real);`

operator `in ( p,q : CIPolynomial ) res : boolean;`

function `blow ( p : CIPolynomial; eps : real ) :`	
(cipoly)	`CIPolynomial[0..ub(p)];`

A.6 Modul scpoly

Das Hilfsmodul scpoly exportiert:

```
operator := (var p : SCPolynomial; r : real);
```

```
operator := (var p : SCPolynomial; r : Longreal);
```

```
operator in ( p,q : SCPolynomial ) res : boolean;
```

```
function blow ( p : SCPolynomial; eps : real ) :
(scpoly) SCPolynomial[0..ub(p),1..2,-1..LStagg_prec];
```

```
procedure read ( var F : Text; var p : SCPolynomial );
```

```
procedure write ( var F : Text; p : SCPolynomial );
```

A.7 Modul `lmv_exap`

Das Hilfsmodul `lmv_exap` exportiert:

procedure `Hilbert( var H : Lrmatrix );`
$\dim(\boldsymbol{H}) \leq 26$, vergl. Seite 163

procedure `Dekker( var D : Lrmatrix; var b : Lrvector );`
$\dim(\boldsymbol{D}) \leq 27$, vergl. Seite 164

procedure `Dekker_INV( var D : Lrmatrix );`
$\dim(\boldsymbol{D}) \leq 27$, vergl. Seite 165

procedure `Zielke( var Z : Lrmatrix; a,b,c : Longreal );`
vergl. Seite 166

A.8 Modul mv_exap

Das Hilfsmodul mv_exap exportiert:

procedure `Hilbert( var H : rmatrix );`
$\dim(\boldsymbol{H}) \leq 18$, vergl. Seite 163

procedure `Dekker( var D : rmatrix; var b : rvector );`
$\dim(\boldsymbol{D}) \leq 16$, vergl. Seite 164

procedure `Dekker_INV( var D : rmatrix );`
$\dim(\boldsymbol{D}) \leq 16$, vergl. Seite 165

procedure `Zielke( var Z : rmatrix; a,b,c : real );`
vergl. Seite 166

A.9 Modul `lst1_ari`

Das Hilfsmodul `lst1_ari` exportiert die folgenden Routinen zur Listenverwaltung von Paaren aus je einem Intervall und einer reellen Zahl[2]:

Globale Datentypen des Hilfsmoduls `lst1_ari`		
Type		**Definition**
`Pair`	=	`record int : Linterval; fyi : Longreal; end;`
`PairPtr`	=	`^PairElmt;`
`PairElmt`	=	`record P : Pair; N : PairPtr; end;`

procedure `FreeAll( var List : PairPtr );`

function `MakePair(int: Linterval; fyi: Longreal): Pair;`

function `GetInt ( P : Pair ) : Linterval;`

function `GetFyi ( P : Pair ) : Longreal;`

function `EmptyList : PairPtr;`

operator `+ (List : PairPtr; P : Pair) Enter : PairPtr;`

procedure `MultiDelete(var List: PairPtr;fmax: Longreal);`

---

[2]Eine kurze Dokumentation aller Routinen findet man im Quelltext des Hilfsmoduls `lst1_ari.p`

function `Next ( List : PairPtr ) : PairPtr;`

function `Head ( List : PairPtr ) : Pair;`

procedure `DelHead ( var List : PairPtr );`

function `Length ( List : PairPtr ) : integer;`

A.10 Modul lst_ari

Das Hilfsmodul **lst_ari** exportiert die folgenden Routinen zur Listenverwaltung von Paaren aus je einem Intervall-Vektor und einer reellen Zahl[3]:

Globale Datentypen des Hilfsmoduls lst_ari	
Type	**Definition**
PairPtr =	^PairElmt;
PairElmt =	record P : integer; N : PairPtr; end;

procedure `FreeAll( var List : PairPtr );`

function `MakePair(int: Livector; fyi: Longreal): Pair;`

function `Int(var Pair: Livector): Livector[1..ub(Pair)];`

[3]Eine kurze Dokumentation aller Routinen findet man im Quelltext des Hilfsmoduls lst_ari.p

function `Fyi ( var Pair : Livector ) : Longreal;`

function `EmptyList : PairPtr;`

```
procedure Enter( var List: PairPtr; var VecMat: Limatrix;
                 var Pair: Livector; var Full: boolean);
```

```
procedure MultiDelete( var List: PairPtr;
                       var M: Limatrix; fmax: Longreal);
```

function `Next ( List : PairPtr ) : PairPtr;`

```
function Head  (List : PairPtr; var MV : Limatrix):
               Livector[0..ub(MV,2)];
```

procedure `DelHead ( var List : PairPtr );`

function `Length ( List : PairPtr ) : integer;`

A.11 Modul `mv_util`

Das Hilfsmodul `mv_util` liefert Ergänzungen zu den beiden Systemmodulen `mvi_ari`, `lmvi_ari` und exportiert:

1. die Funktionen `MaxRelDiam` vom Ergebnistyp `real | Longreal`
2. die Funktion `UlpAcc` vom Ergebnistyp `boolean` und
3. zusätzliche Operatoren `in` vom Ergebnistyp `boolean`.

Das Hilfsmodul `mv_util` importiert die folgenden Module:

`i_util, i_ari, li_ari`

A.11.1 Funktionen vom Ergebnistyp `real | Longreal`

function `MaxRelDiam (v : ■) : □;` (`mv_util`)

■ = `ivector` und □ = `real` oder

■ = `Livector` und □ = `Longreal`

`MaxRelDiam(v)` liefert das Maximum des relativen Durchmessers aller Komponenten des Intervallvektors `v`; vergl. dazu Seite 640.

A.11.2 Funktionen vom Ergebnistyp `boolean`

function `UlpAcc (v : ■; n : integer) : boolean;` (`mv_util`)

■ = `ivector | Livector`

Funktion `UlpAcc(v,n)` testet, ob sich Unter- und Oberschranke aller Komponenten von `v` um höchstens n ulp unterscheiden.

1 ulp: Eine Einheit auf der letzten Mantissenstelle.

A.11.3 Operatoren vom Ergebnistyp `boolean`

Die folgenden Operatoren testen, ob `r` in allen Komponenten des Intervallvektors `v` enthalten ist.

```
operator in ( r : real; v : (L)ivector ) res : boolean;
```

```
operator in ( r : Longreal; v : Livector ) res : boolean;
```

A.12 Modul `blow_z`

Das Hilfsmodul `blow_z` exportiert die Funktionen `Blow_z`, mit denen komplexe Intervall-Matrizen und -Vektoren aufgebläht werden, wobei die Null nach $\pm 10^{+255}$ gerundet wird.

```
function Blow_z ( var A : Lcivector; eps : real ) :
(blow_z)                    Lcivector[lb(A)..ub(A)];
```

```
function Blow_z ( var A : Lcimatrix; eps : real ) :
(blow_z)  Lcimatrix[lb(A)..ub(A),lb(A,2)..ub(A,2)];
```

A.13 Modul rrgi_util

Das Hilfsmodul rrgi_util liefert Ergänzungen zum Systemmodul rrgi_ari und exportiert:

1. Operatoren $+, -, /$ mit gemischten Operanden
2. Einschließung mathematischer Konstanten mit dem Typ rrGinterval
3. Hilfsfunktionen
4. Approximation mathematischer Konstanten mit dem Typ rrG
5. Mathematische Funktionen

Das Hilfsmodul rrgi_util importiert das folgende System–Modul:

rrgi_ari

A.13.1 Operatoren $+, -, /$ mit gemischten Operanden

Ist bei der Addition oder Subtraktion ein Operand vom Typ rrGinterval, so darf der andere Operand die Typen integer, real, Longreal, rrG annehmen.

Ist bei der Division der zweite Operand vom Typ rrGinterval, so darf der erste Operand die Typen integer, real, Longreal, rrG annehmen.

A.13.2 Einschließung mathematischer Konstanten

Einschließung mathematischer Konstanten mit dem Datentyp `rrGinterval`			
Konstante	**Name**	**Konstante**	**Name**
$\sqrt{2}$	`SQRT2_rrGi`	e	`EUL_rrGi`
$1/\sqrt{2}$	`R_SQRT2_rrGi`	$1/e$	`R_EUL_rrGi`
$\ln(2)$	`LN_2_rrGi`	$\sqrt{e}$	`SQRTEUL_rrGi`
$1/\ln(2)$	`R_LN_2_rrGi`	C	`GAMMA_C_rrGi`
$\ln(10)$	`LN_10_rrGi`	$\ln(C)$	`LN_GAM_C_rrGi`
$1/\ln(10)$	`R_LN_10_rrGi`	$Catalan$	`CATALAN_rrGi`
π	`PI_rrGi`		
2π	`PI2_rrGi`		
$\pi/2$	`PI_HALF_rrGi`		
$\pi/4$	`PI_DIV4_rrGi`		
$1/\pi$	`R_PI_rrGi`		
$1/(2\pi)$	`R_2PI_rrGi`		
$\sqrt{\pi}$	`SQRT_PI_rrGi`		
$\sqrt{\pi}/2$	`SQRT_PI_D2_rrGi`		
$1/\sqrt{\pi}$	`R_SQRTPI_rrGi`		
$2/\sqrt{\pi}$	`R_SQRTPI2_rrGi`		
$1/\sqrt{2\pi}$	`R_SQRT2PI_rrGi`		
$\ln(\pi)$	`LN_PI_rrGi`		

A.13.3 Hilfsfunktionen

function `AbsMax( x : rrGinterval) : rrG;`

liefert das Maximum der Beträge aller Elemente des Intervalls `x`.

```
function AbsMin( x : rrGinterval) : rrG;
```

liefert das Minimum der Beträge aller Elemente des Intervalls `x`.

```
function mid( x : rrGinterval) : rrG;
```

liefert i.a. eine maximalgenaue Approximation des Mittelpunktes von `x`.

A.13.4 Approximation mathematischer Konstanten

Approximation mathematischer Konstanten mit dem Datentyp `rrG`			
Konstante	**Name**	**Rel. Fehlerschr.**	**Vergleich**
$1/\ln 2$	`R_LN_2_rrG`	$6.95 \cdot 10^{-28}$	`R_LN_2_rrG` $< 1/\ln(2)$

Eine Vielzahl weiterer Konstanten wird durch das Standardmodul `stdmod` automatisch zur Verfügung gestellt. Eine Liste dieser Konstanten zusammen mit den entsprechenden relativen Fehlerschranken findet man im Handbuch der **PASCAL-XSC** BCD–Version.

A.13.5 Mathematische Funktionen

```
function factorial_rrGi( n : integer ) : rrGinterval;
```

liefert für $n = 0, 1, 2, \ldots$ eine Einschließung des ganzzahligen Wertes $n!$

```
function power( x : rrG; n : integer ) : rrG;
```

`y := power(x,n)` liefert für $n = 0, 1, 2, \ldots$ eine Approximation der Potenz $x^n \approx$ `y`, wobei $n < 0$ zu einer entsprechenden Fehlermeldung führt. Weitere Standardfunktionen findet man im Handbuch der **PASCAL-XSC** BCD–Version.

A.14 Modul `lci_util`

Das Hilfsmodul `lci_util` liefert Ergänzungen zum Systemmodul `lci_ari`.

A.14.1 Operatoren vom Ergebnistyp `rrGcinterval`

Das System–Modul `lci_ari` exportiert die wichtigsten in der Tabelle des BCD–Benutzerhandbuchs [16, S.154] angegebenen Operatoren mit dem Ergebnistyp `rrGcinterval`.

Zusammen mit den vom Hilfs–Modul `lci_util` zusätzlich bereitgestellten Operatoren $+\ -\ *\ /$, ebenfalls vom Ergebnistyp `rrGcinterval`, stehen dann die folgenden Operandenkombinationen zur Verfügung: Ein Operand vom Typ `rrGcinterval`, der andere Operand vom Typ:

```
integer, real, Longreal, rrG, rrGinterval,
complex, Lcomplex, rrGcomplex, rrGcinterval
```

Damit lassen sich alle möglichen Operandenkombinationen realisieren. Ein Operand vom Typ `Lcinterval` kann beispielsweise nach der unteren Tabelle in [16, S.154] als Operand vom Typ `rrGcinterval` dargestellt werden.

A.14.2 Weitere komplexe Standardfunktionen

A.14.2.1 $f(z) = z^z$

Zur Realisierung der komplexwertigen Funktion $f(z) = z^z \equiv e^{z \cdot \ln(z)}, z \in \mathbb{C}$ werden vom Hilfs–Modul `lci_util` die folgenden Routinen exportiert:

function `power (z : ■) : ■;` (lci_util)

■ = `rrGcinterval, Lcinterval`

function `power_rrG (z : Lcinterval) : rrGcinterval;`

Die obigen Routinen liefern zu einem Intervallargument $\mathbf{z}$ Einschließungen der Funktionswerte $f(z) = z^z$ für $z \in \mathbf{z} \not\ni 0$. Wie bei der $\ln(z)$-Funktion ist der Verzweigungsschnitt die negative reelle Achse.

A.15 Modul xi_ari

Das Hilfsmodul `xi_ari` exportiert Funktionen und Operatoren zum Basistyp `Longreal` für eine reelle *extended interval arithmetic*. In dieser Intervall–Arithmetik werden die Datentypen `KindType` und `xinterval` definiert, mit deren Hilfe auch die Division durch ein Intervall möglich wird, das die Zahl Null enthält.

```
type
KindType  = (Finite, PlusInfty, MinusInfty, Double);
xinterval = record
              kind     : KindType;
              inf, sup : Longreal;
(xi_ari)    end;
```

Neben der Funktion `EmptyIntval`, welche die Leere Menge darstellt, werden nur noch drei arithmetische Operatoren exportiert, die in Kurzform lauten:

Longreal	−	xinterval	⟶	xinterval
Linterval	**div**	Linterval	⟶	xinterval
Linterval	**	xinterval	⟶	Livector[1..2]

Diese Operatoren werden benötigt beim Intervall–Newtonverfahren zur Nullstelleneinschließung nichtlinearer Gleichungen. Weitere Einzelheiten und Beispiele findet man in [5, Seite 40,41]. Es folgt der Quelltext des Moduls `xi_ari`:

```
module xi_ari;

use li_ari;     { Interval arithmetic with base type Longreal }

{-------------------------------------------------------------}
{ Global type definitions                                     }
{-------------------------------------------------------------}
{ An extended interval 'x', represented by type 'xinterval',  }
{ is defined according to the following rules (a <= b):       }
{                                                             }
```

```
{ x = [a, b]             :                               }
{            x.kind = Finite,     x.inf = a, x.sup = b      }
{ x = [a, +oo]           :                               }
{            x.kind = PlusInfty,  x.inf = a, x.sup undefined }
{ x = [-oo, a]           :                               }
{            x.kind = MinusInfty, x.sup = a, x.inf undefined }
{ x = [-oo, a] v [b, +oo] :                              }
{            x.kind = Double,     x.inf = b, x.sup = a      }
{ x = [-oo, +oo]         :                               }
{            x.kind = Double,     x.inf = a, x.sup = a      }
{                                                          }
{ In this definition, 'v' stands for the set union and     }
{ 'oo' for infinity.                                       }
{----------------------------------------------------------}
global type
 KindType  = (Finite, PlusInfty, MinusInfty, Double);
 xinterval = global record             { Extended intervals  }
                     kind     : KindType; { according to the }
                     inf, sup : Longreal; { definition above }
             end;                        {------------------}

{----------------------------------------------------------}
{ Function 'EmptyIntval' delivers an empty interval(empty set)}
{----------------------------------------------------------}
global function EmptyIntval : Linterval;
begin
  EmptyIntval.inf :=  999999999;        { irregular interval }
  EmptyIntval.sup := -999999999;
end;

{----------------------------------------------------------}
{ Operators -, 'div', and ** for extended intervals         }
{----------------------------------------------------------}
{ Subtraction of an extended interval 'B' from a Longreal   }
{ value 'a'.                                               }
{----------------------------------------------------------}
global operator - (a : Longreal; B : xinterval) di : xinterval;
  var
    D : xinterval;
  begin
```

```
    case B.kind of
      Finite     : begin                  { D = [D.inf, D.sup] }
                      D.kind:= Finite;
                      D.inf := a -< B.sup;
                      D.sup := a -> B.inf;
                    end;
      PlusInfty  : begin                  { D = [inf, +oo]     }
                      D.kind:= MinusInfty;
                      D.sup := a -> B.inf;
                    end;
      MinusInfty : begin                  { D = [-oo, sup]     }
                      D.kind:= PlusInfty;
                      D.inf := a -< B.sup;
                    end;
      Double     : begin    { D = [-oo, D.sup] v [D.inf, +oo] }
                      D.kind:= Double;
                      D.inf := a -< B.sup;
                      D.sup := a -> B.inf;
                      if (D.inf < D.sup) then
                        D.inf := D.sup;
                    end;
    end;
    di := D;
  end;

{-------------------------------------------------------------}
{ Extended interval division 'A/B', where 0 in 'B' is allowed }
{-------------------------------------------------------------}
global operator div (A, B : Linterval) quotient : xinterval;
  var
    C : Linterval;
    Q : xinterval;
  begin
    if (0 in B) then
      begin
        if ((inf(A)<0) and (0<sup(A))) or (A=0) or (B=0) then
          begin        { Q = [-oo, +oo] = [-oo, 0] v [0, +oo] }
            Q.kind:= Double;
            Q.sup := 0;
            Q.inf := 0;
```

```
      end
    else if ((sup(A) <= 0) and (sup(B) = 0)) then
      begin                           { Q = [Q.inf, +oo] }
        Q.kind:= PlusInfty;
        Q.inf := sup(A) /< inf(B);
      end
    else if (sup(A)<=0) and (inf(B)<0) and (sup(B)>0)  then
      begin             { Q = [-oo, Q.sup] v [Q.inf, +oo] }
        Q.kind:= Double;
        Q.sup := sup(A) /> sup(B);
        Q.inf := sup(A) /< inf(B);
      end
    else if ((sup(A) <= 0) and (inf(B) = 0)) then
      begin                           { Q = [-oo, Q.sup] }
        Q.kind:= MinusInfty;
        Q.sup := sup(a) /> sup(b);
      end
    else if ((inf(A) >= 0) and (sup(B) = 0)) then
      begin                           { Q = [-oo, Q.sup] }
        Q.kind:= MinusInfty;
        Q.sup := inf(A) /> inf(B);
      end
    else if (inf(A)>=0) and (inf(B)<0) and (sup(B)>0) then
      begin             { Q = [-oo, Q.sup] v [Q.inf, +oo] }
        Q.kind:= Double;
        Q.sup := inf(A) /> inf(B);
        Q.inf := inf(A) /< sup(B);
      end
    else if ((inf(A) >= 0) and (inf(B) = 0)) then
      begin                           { Q = [Q.inf, +oo] }
        Q.kind:= PlusInfty;
        Q.inf := inf(A) /< sup(B);
      end
    else ;
  end { 0 in B }
else  { not (0 in B) }
  begin                             { Q = [C.inf, C.sup] }
    C     := A / B;          Q.kind:= Finite;
    Q.inf := C.inf;
    Q.sup := C.sup;
```

```
      end;
    quotient := Q;
  end;

{-------------------------------------------------------------}
{ Intersection of an interval 'X' and an extended interval 'Y'}
{ The result is given as a pair (vector) of intervals, where  }
{ one or both of them can be empty intervals ('EmptyIntval'). }
{-------------------------------------------------------------}
global operator ** (X: Linterval; Y: xinterval) Intersect :
                                                Livector[1..2];
  var
    H : Linterval;
  begin
    Intersect[1]:= EmptyIntval;
    Intersect[2]:= EmptyIntval;

    case Y.kind of
      Finite    : begin        { [X.inf,X.sup]**[Y.inf,Y.sup] }
                    H := Lintval(Y.inf,Y.sup);
                    if not (X >< H) then
                      Intersect[1]:= X ** H;
                  end;
      PlusInfty : if (X.sup >= Y.inf) then
                  begin       { [X.inf,X.sup] ** [Y.inf,+oo] }
                    if (X.inf > Y.inf) then
                      Intersect[1]:= X
                    else
                      Intersect[1]:= Lintval(Y.inf, X.sup);
                  end;
      MinusInfty: if (Y.sup >= X.inf) then
                  begin       { [X.inf,X.sup] ** [-oo,Y.sup] }
                    if (X.sup < Y.sup) then
                      Intersect[1]:= X
                    else
                      Intersect[1]:= Lintval(X.inf, Y.sup)
                  end;
      Double    : if ((X.inf<=Y.sup) and (Y.inf<=X.sup)) then
                    begin
      Intersect[1]:= Lintval(X.inf, Y.sup);  { X**[-oo,Y.sup] }
```

```
      Intersect[2]:= Lintval(Y.inf, X.sup);  { X**[Y.inf,+oo] }
                    end
                  else if (Y.inf <= X.sup) then
                    begin        { [X.inf,X.sup]**[Y.inf,+oo] }
                      if (X.inf >= Y.inf) then
                        Intersect[1]:= X
                      else
                        Intersect[1]:= Lintval(Y.inf, X.sup);
                    end
                  else if (X.inf <= Y.sup) then
                    begin        { [X.inf,X.sup]**[-oo,Y.sup] }
                      if (X.sup <= Y.sup) then
                        Intersect[1]:=  X
                      else
                        Intersect[1]:=  Lintval(X.inf, Y.sup);
                    end;
    end; { CASE kind OF ... }
  end; { OPERATOR ** ... }

{---------------------------------------------------------------}
{ Module initialization part                                    }
{---------------------------------------------------------------}
begin
  { Nothing to initialize }
end.
```

Anhang B

Globale Datentypen/Variable

In der BCD–Version von **PASCAL-XSC** stehen neben den Datentypen

`integer, real, Longreal, boolean, char, string`

noch folgende Typen zur Verfügung[1]:

`dotprecision, idotprecision, cdotprecision`

Der Typ `dotprecision` ist die Grundlage für alle #-Ausdrücke, die in [12] ausführlich beschrieben werden.

Die Typen `idotprecision, cdotprecision` sind `records` mit Komponenten vom Typ `dotprecision` und ermöglichen folgende Konstruktionen[2]:

```
type idotprecision = record
                       Inf,Sup : dotprecision
                     end;
var x : idotprecision;
begin ...  x.Inf := #(0);      x.Sup := #(1);  ... end.


type cdotprecision = record
                       Re,Im : dotprecision
                     end;
var z : cdotprecision;
begin ...  z.Inf := #(5*2.1);  z.Sup := #(2);  ... end.
```

[1]Es existiert zusätzlich der Typ `cidotprecision`, der jedoch vom Compiler fehlerhaft behandelt wird.

[2]Die Typen `idotprecision, cdotprecision` sind vordefiniert und müssen daher nicht eigens deklariert werden!

Die folgende Tabelle enthält alle im Systemmodul `stdmod`[3] vordefinierten, globalen Datentypen, die damit bei jeder Programmentwicklung automatisch zur Verfügung stehen:

Globale Datentypen des System-Moduls `stdmod`		
Type		**Definition**
`Linterval`	=	global record inf,sup : `Longreal` end;
`Lcinterval`	=	global record Re,Im : `Linterval` end;
`Lcomplex`	=	global record Re,Im : `Longreal` end;
`Lrvector`	=	global dynamic array [∗] of `Longreal`;
`Lrmatrix`	=	global dynamic array [∗] of `Lrvector`;
`Lcvector`	=	global dynamic array [∗] of `Lcomplex`;
`Lcmatrix`	=	global dynamic array [∗] of `Lcvector`;
`Livector`	=	global dynamic array [∗] of `Linterval`;
`Limatrix`	=	global dynamic array [∗] of `Livector`;
`Lcivector`	=	global dynamic array [∗] of `Lcinterval`;
`Lcimatrix`	=	global dynamic array [∗] of `Lcivector`;
`rrG`	=	global record r1,r2 : `real`; G : `integer` end;
`lrG`	=	global record l : `Longreal`; r : `real`; G : `integer` end;
`rrGinterval`	=	global record inf,sup : `rrG` end;
`rrGcinterval`	=	global record Re,Im : `rrGinterval` end;
`rrGcomplex`	=	global record Re,Im : `rrG` end;

Das System–Modul `stdmod` exportiert zusätzlich die globalen Variablen

```
global var FW_exakt, interval_calculation : boolean;
```

die jedoch nur bei der Deklaration mathematischer Funktionen zu verwenden sind.

[3]Das Systemmodul `stdmod` muß über die `use`-Klausel nicht extra eingebunden werden!

In den beiden folgenden Tabellen sind alle Datentypen zusammengestellt, die vom Hilfsmodul `type_def` exportiert werden:

Globale Datentypen des Hilfsmoduls `type_def` (Teil 1)		
Type		**Definition**
`intvector`	=	global dynamic array [*] of `integer`;
`bvector`	=	global dynamic array [*] of `boolean`;
`mp`	=	global dynamic array [*] of `integer`;
`mpvector`	=	global dynamic array [*] of `mp`;
`rrGvector`	=	global dynamic array [*] of `rrG`;
`rrGmatrix`	=	global dynamic array [*] of `rrGvector`;
`rrGivector`	=	global dynamic array [*] of `rrGinterval`;
`rrGimatrix`	=	global dynamic array [*] of `rrGivector`;
`Stagg`	=	global dynamic array [*] of `real`;
`Staggcomplex`	=	global dynamic array [*] of `Stagg`;
`Staggrvector`	=	global dynamic array [*] of `Stagg`;
`Staggrmatrix`	=	global dynamic array [*] of `Staggrvector`;
`Staggcvector`	=	global dynamic array [*] of `Staggcomplex`;
`Staggcmatrix`	=	global dynamic array [*] of `Staggcvector`;
`LStagg`	=	global dynamic array [*] of `Longreal`;
`LStaggcomplex`	=	global dynamic array [*] of `LStagg`;
`LStaggrvector`	=	global dynamic array [*] of `LStagg`;
`LStaggrmatrix`	=	global dynamic array [*] of `LStaggrvector`;
`LStaggcvector`	=	global dynamic array [*] of `LStaggcomplex`;
`LStaggcmatrix`	=	global dynamic array [*] of `LStaggcvector`;

Globale Datentypen des Hilfsmoduls type_def (Teil 2)		
Type		**Definition**
RPolynomial	=	global dynamic array [*] of real;
RIPolynomial	=	global dynamic array [*] of interval;
RSPolynomial	=	global dynamic array [*] of Stagg;
RLPolynomial	=	global dynamic array [*] of Longreal;
RLIPolynomial	=	global dynamic array [*] of Linterval;
RLSPolynomial	=	global dynamic array [*] of LStagg;
CPolynomial	=	global dynamic array [*] of complex;
CIPolynomial	=	global dynamic array [*] of cinterval;
CSPolynomial	=	global dynamic array [*] of Staggcomplex;
SCPolynomial	=	global dynamic array [*] of LStaggcomplex;

Globaler Datentyp des Hilfsmoduls d3f_rrg		
Type		**Definition**
Grad3Type	=	global record f,df,ddf,dddf : rrGinterval end;

Globaler Datentyp des Hilfsmoduls ddf_rrg		
Type		**Definition**
DerivType	=	global record f,df,ddf : rrGinterval end;

Globale Datentypen des Hilfsmoduls igrad		
Type		**Definition**
GradType	=	global dynamic array [*] of rrGinterval;
GTvector	=	global dynamic array [*] of GradType;

Globaler Datentyp des Hilfsmoduls `itaylor`		
Type		**Definition**
`itaylor`	=	`global dynamic array [*] of rrGinterval;`

Globaler Datentyp des Hilfsmoduls `litaylor`		
Type		**Definition**
`litaylor`	=	`global dynamic array [*] of Linterval;`

Globaler Datentyp des Hilfsmoduls `citaylor`		
Type		**Definition**
`citaylor`	=	`global dynamic array [*] of rrGcinterval;`

Globale Datentypen des Hilfsmoduls `hess_ari`		
Type		**Definition**
`HessType`	=	`dynamic array [*,*] of rrGinterval;`
`HTvector`	=	`global dynamic array [*] of HessType;`

Globale Datentypen des Hilfsmoduls `lst1_ari`		
Type		**Definition**
`Pair`	=	`record int : Linterval; fyi : Longreal; end;`
`PairPtr`	=	`^PairElmt;`
`PairElmt`	=	`record P : Pair; N : PairPtr; end;`

Globale Datentypen des Hilfsmoduls `lst_ari`		
Type		**Definition**
`PairPtr`	=	`^PairElmt;`
`PairElmt`	=	`record P : integer; N : PairPtr; end;`

Globale Datentypen des Hilfsmoduls xi_ari		
Type		**Definition**
KindType	=	(Finite,PlusInfty,Minusinfty,Double);
PairElmt	=	record
	=	kind : KindType; inf,sup : Longreal;
	=	end;

Das Hilfsmodul `type_def` exportiert die folgenden Variablen:

Globale Variable des Hilfsmoduls type_def		
Variable	**Präzisionsparameter**	**Default**
`Stagg_prec : integer`	für Ausdrücke vom Typ `Stagg`	1
`LStagg_prec : integer`	für Ausdrücke vom Typ `LStagg`	1
`maxprec : integer`	für Ausdrücke vom Typ `mp`	7

Routinen zur Darstellung oder Veränderung der `integer`–Werte `maxprec`, `Stagg_prec`, `LStagg_prec` findet man auf Seite 638.

Anhang C

Globale Prozeduren der Hilfsmodule

Mit (...) geklammerte Namen sind die Namen der jeweils exportierenden Hilfsmodule. Das Symbol $(***)$ bedeutet, daß verschiedene Prozeduren mit gleichem Namen von mehreren Modulen exportiert werden.

Globale Prozeduren der Hilfsmodule		
Name	**Bearbeitung**	**Seite**
`AB_inv`	Einschließung von $(\boldsymbol{AB})^{-1}$ $(***)$	282
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R}$ (`mv_comp`)	170
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \in \boldsymbol{I}$ (`mv_comp`)	170
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \subseteq \boldsymbol{I}$ (`mv_comp`)	170
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R1} + \boldsymbol{R2}$ (`mv_comp`)	170
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \subseteq \boldsymbol{I}$ (`mv_comp`)	171
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R}$ (`mvc_comp`)	176
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R}$ (`mvc_comp`)	176
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \in \boldsymbol{I}$ (`mvc_comp`)	176
`ab_plus_c`	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} \in \boldsymbol{I}$ (`mvc_comp`)	176

Globale Prozeduren der Hilfsmodule Teil 2		
Name	**Bearbeitung**	**Seite**
add_complex_times_Staggc	complex * Staggc+cd (cstagg)	138
add_complex_times_Stagg	complex * Stagg+cd (cstagg)	138
add_Lcomplex_times_LStagg	Lcomplex*LStagg+cdot (lcstagg)	152
add_Lcomplex_times_LStaggc	Lcomplex*LStaggc+cdot (lcstagg)	152
add_Longreal_times_LStagg	Lgreal*LStagg+[d1, d2] (lstagg)	94
add_Longreal_times_LStaggc	Lgreal*LStaggc+cdot (lcstagg)	151
add_LStagg_times_LStagg	LStagg*LStagg+[$d1, d2$] (lstagg)	94
add_LStagg_times_LStaggc	LStagg*LStaggc+cd (lcstagg)	151
add_LStaggc_times_LStaggc	LStaggc*LStaggc+cd (lcstagg)	151
add_real_times_Stagg	real * Stagg+[d1, d2] (stagg)	68
add_real_times_Staggc	real * Staggc+cdot (cstagg)	137
add_Stagg_times_Stagg	Stagg * Stagg+[d1, d2] (stagg)	68
add_Stagg_times_Staggc	Stagg * Staggc+cdot (cstagg)	137
add_Staggc_times_Staggc	Staggc * Staggc+cd (cstagg)	137
Alefeld_IT	Inv.: Einschlgs.-Verbesserung (***)	278
Allnlss	Nichtlin. Gleichungssysteme (nlss)	481
AllZeros	Nichtlineare Gleichungen (nlfzero)	439
AP	Matrix: Spaltenpermutationen (***)	223
block	liefert Block-Matrix (***)	167
block_1	liefert weitere Block-Matrix (***)	167
DEKKER	liefert Dekker-Matrix (mv_exap)	165
DEKKER	liefert Dekker-Matrix (lmv_exap)	165
DEKKER_INV	inverse Dekker-Matrix (mv_exap)	165
DEKKER_INV	inverse Dekker-Matrix (lmv_exap)	165
DelHead	Listen, (Intervall,Zahl) (lst1_ari)	647
DelHead	Listen, (Vektor,Zahl) (lst_ari)	648

Globale Prozeduren der Hilfsmodule Teil 3		
Name	**Bearbeitung**	**Seite**
derivative (citaylor)	Ableitungen komplexer Funktionen	435
derivative (litaylor)	Ableitungen reeller Funktionen	398
derivative (itaylor)	Ableitungen reeller Funktionen	381
DET	Determinanten-Einschließung (∗∗∗)	240
ddfEval (ddf_rrg)	Einschließung von $f''(x)$, Linterval	354
ddfEval (ddf_rrg)	Einschließung von $f''(x)$, interval	360
dfEval (ddf_rrg)	Einschließung von $f'(x)$, Linterval	354
dfEval (ddf_rrg)	Einschließung von $f'(x)$, interval	360
diff_mv	Vektorielle Differenzenglg. (differ)	325
diff_mv	Vektorielle Differenzenglg. (ldiffer)	326
diff_sc	Skalare Differenzengleichung (differ)	318
diff_sc	Skalare Differenzenglg. (ldiffer)	319
EIG	Einschlg. einfacher Eigenpaare (∗∗∗)	594
eig_all	Berechnung aller Eigenpaare (∗∗∗)	611
eigen (eig_app)	Approximation aller Eigenpaare	589
Enter	Listen, (Vektor,Zahl) (lst_ari)	648
expand	Reelle Taylorkoeffizienten (itaylor)	377
expand	Reelle Taylorkoeffizienten (litaylor)	397
expand (citaylor)	Komplexe Taylor-Koeffizienten	431
fEval (ddf_rrg)	Einschließung v. $f(x)$, Linterval	354
fEval (ddf_rrg)	Einschließung v. $f(x)$, interval	360
fEvalG (igrad)	Nur Funktionswert-Einschließung	405
fEvalG_rrG (igrad)	Nur Funktionswert-Einschließung	405

Globale Prozeduren der Hilfsmodule Teil 4			
Name	**Bearbeitung**		**Seite**
fEvalJ	Nur Funktionswert-Einschließung	(igrad)	415
fEvalJ_rrG	Nur Funktionswert-Einschließung	(igrad)	415
fgEvalG	Gradienten-Einschließung	(igrad)	405
fgEvalG_rrG	Gradienten-Einschließung	(igrad)	405
fghEvalH	Fkt.-Wert, Gradient, Hesse-Matrix	(hess_ari)	422
fghEvalH	wie oben, aber im rrG-Format	(hess_ari)	422
fghEvalH	Funktionswert, Gradient	(hess_ari)	423
fghEvalH	wie oben, aber im rrG-Format	(hess_ari)	423
fghEvalH	Nur Funktionswert-Einschließung	(hess_ari)	423
fghEvalH	wie oben, aber im rrG-Format	(hess_ari)	423
fJEvalJ	Jacobi-Matr.-Einschließung	(igrad)	416
fJEvalJ_rrG	Jacobi-Matr.-Einschließung	(igrad)	416
FreeAll	Listen, (Intervall,Zahl)	(lst1_ari)	646
FreeAll	Listen, (Vektor,Zahl)	(lst_ari)	647
Function_Analyser	Kurvendiskussion	(nlfz3)	458
heig	Hermitesches EW-Problem	(∗∗∗)	632
HILBERT	liefert Hilbert-Matrix	(mv_exap)	164
HILBERT	liefert Hilbert-Matrix	(lmv_exap)	164
inv	Näherungsinverse Defektiteration	(∗∗∗)	270
inv	Näherungsinverse (Rump, Def.-Iter.)	(∗∗∗)	272
inv	Inverse: Einschließg. in stagg. Arithm.	(∗∗∗)	273
inv	Inverse: Einschließg. in stagg. Arithm.	(∗∗∗)	274
inv	Inverse: Einschließg., verschiedene Verf.	(∗∗∗)	275

Globale Prozeduren der Hilfsmodule Teil 5		
Name	**Bearbeitung**	**Seite**
inv	Inverse einer Intervall-Matrix (***)	276
inv_Def	Näherungsinv. Defekt-Iter., -Vorgabe (***)	269
inv_Def	Wie oben, aber doppelte Länge (***)	271
inv_LU	Näherungsinverse nur mit $\boldsymbol{LU}$-Zerlg. (***)	265
inv_Rump	Näherungsinverse, ($\boldsymbol{LU}$ oder Rump) (***)	266
inv_Rump	Näherungsinv., (Rump, Defektvorgabe) (***)	267
inv_Rump	Näherungsinv., (Rump, Defektausgabe) (***)	268
inv_ST	Näherungsinverse in doppelter Länge (***)	277
Local_Min	Minimum aller lokalen Minima (gop)	511
lss	$\boldsymbol{Ax} = \boldsymbol{b}$; Einschließung von $\boldsymbol{x}$ (***)	299
lss	$\boldsymbol{Ax} = \boldsymbol{b}$; $\boldsymbol{A}$: Bandmatrix (***)	312
LU	$\boldsymbol{LU}$-Zerlegung einer Matrix (***)	250
m_comp	$\boldsymbol{A} + x \cdot \boldsymbol{B} \approx \boldsymbol{R}$ (mv_comp)	170
m_comp	$\boldsymbol{A} + x \cdot \boldsymbol{B} \in \boldsymbol{R}$ (mv_comp)	170
m_comp	$\boldsymbol{A} + x \cdot \boldsymbol{B} \subseteq \boldsymbol{R}$ (mv_comp)	170
m_comp	$\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{B} \cdot \boldsymbol{C} \approx \boldsymbol{R}$ (mv_comp)	172
m_comp	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + (\boldsymbol{B} + \boldsymbol{C}) \cdot \boldsymbol{D} \approx \boldsymbol{R}$ (mv_comp)	172
m_comp1	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) \approx \boldsymbol{R}$ (mv_comp)	171
m_comp1	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \approx \boldsymbol{R}$ (mv_comp)	172
m_comp1	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \in \boldsymbol{I}$ (mv_comp)	172
m_comp1	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \subseteq \boldsymbol{I}$ (mv_comp)	172
m_comp1	$\boldsymbol{A} + \boldsymbol{B} + \boldsymbol{C} \subseteq \boldsymbol{I}$ (mv_comp)	173
m_comp1	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \approx \boldsymbol{R}$ (mvc_comp)	177

Globale Prozeduren der Hilfsmodule Teil 6		
Name	**Bearbeitung**	**Seite**
m_comp1	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \approx \boldsymbol{R}$ (mvc_comp)	177
m_comp1	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \in \boldsymbol{R}$ (mvc_comp)	177
m_comp1	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + \boldsymbol{D} \in \boldsymbol{R}$ (mvc_comp)	177
m_comp1	$\boldsymbol{A} + \boldsymbol{B} + \boldsymbol{C} \subseteq \boldsymbol{R}$ (mvc_comp)	178
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \approx \boldsymbol{R}$ (mvc_comp)	177
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \approx \boldsymbol{R}$ (mvc_comp)	177
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \in \boldsymbol{R}$ (mvc_comp)	177
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \in \boldsymbol{R}$ (mvc_comp)	178
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + (\boldsymbol{A} + \boldsymbol{B}) \approx \boldsymbol{R1} + \boldsymbol{R2}$ (mvc_comp)	178
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} \approx \boldsymbol{R}$ (mv_comp)	171
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} \in \boldsymbol{I}$ (mv_comp)	171
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \approx \boldsymbol{R}$ (mv_comp)	171
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \in \boldsymbol{I}$ (mv_comp)	171
m_comp2	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + \boldsymbol{D} \subseteq \boldsymbol{I}$ (mv_comp)	171
m_comp2_m	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{C} + (\boldsymbol{A} + \boldsymbol{B}) \approx \boldsymbol{R1} + \boldsymbol{R2}$ (mv_comp)	171
m_comp3	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + (\boldsymbol{B} + \boldsymbol{C}) \approx \boldsymbol{R1} + \boldsymbol{R2}$ (mvc_comp)	178
m_comp3	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C}) + (\boldsymbol{B} + \boldsymbol{C}) \approx \boldsymbol{R1} + \boldsymbol{I\!R2}$ (mv_comp)	172
m_comp4	$\boldsymbol{A} + \boldsymbol{B} + \boldsymbol{C} \approx \boldsymbol{R1} + \boldsymbol{R2}$ (mv_comp)	172
m_comp5	$\boldsymbol{A} \cdot (\boldsymbol{B} + \boldsymbol{C} + \boldsymbol{D}) \approx \boldsymbol{R}$ (mv_comp)	172
mant_exp	Mantisse, Exponent (stagg)	68
mant_exp	Mantisse, Exponent (lstagg)	94
Min_g1	Einschließung des glob. Minimums (gop1)	498
MLU_to_L	liefert normiertes $\boldsymbol{L}$ (***)	226

Globale Prozeduren der Hilfsmodule Teil 7		
Name	**Bearbeitung**	**Seite**
MLU_to_U	liefert obere Dreiecks-Matrix $\boldsymbol{U}$ (***)	226
mpdivint	mp-Division mit Rest (lo_int)	50
mpmulint	mp-Wert * integer (lo_int)	50
mul	$\boldsymbol{A} \cdot \boldsymbol{B} \approx \boldsymbol{R}1 + \boldsymbol{R}2$ (mv_comp)	169
mul	$\boldsymbol{A} \cdot \boldsymbol{B} \in \boldsymbol{R} + \boldsymbol{I}$ (mv_comp)	169
mul	$\boldsymbol{A} \cdot \boldsymbol{B} \approx \boldsymbol{R}1 + \boldsymbol{R}2$ (mvc_comp)	176
mul	$\boldsymbol{A} \cdot \boldsymbol{B} \in \boldsymbol{R} + \boldsymbol{I}$ (mvc_comp)	176
mul	$\boldsymbol{A} \cdot \boldsymbol{B} \cdot \boldsymbol{C} \in \boldsymbol{R}$ (stgmv)	182
mul	$\mathtt{A} \cdot \mathtt{B} \cdot \mathtt{C} \in \mathtt{R}$ (stgmv)	192
mul	$\mathtt{A} \cdot \mathtt{B} \cdot \mathtt{C} \in \mathtt{R}$ (smvc)	202
mul	$\mathtt{A} \cdot \mathtt{B} \cdot \mathtt{C} \in \mathtt{R}$ (lsmvc)	213
mul1pmeps	Stagg $* [1-\varepsilon, 1+\varepsilon]$ (stagg)	69
mul1pmeps	LStagg $* [1-\varepsilon, 1+\varepsilon]$ (lstagg)	95
MultiDelete	Listen, (Intervall,Zahl) (lst1_ari)	646
MultiDelete	Listen, (Vektor,Zahl) (lst_ari)	648
mv_comp1	$\boldsymbol{A} \cdot \boldsymbol{b} + \boldsymbol{c} \approx \boldsymbol{r}$ (mv_comp)	173
mv_comp1	$\boldsymbol{A} \cdot \boldsymbol{b} + \boldsymbol{c} \approx \boldsymbol{r}$ (mvc_comp)	178
mv_comp1a	$\boldsymbol{A} \cdot \boldsymbol{b} + t \cdot \boldsymbol{c} \approx \boldsymbol{r}$ (mv_comp)	173
mv_comp1b	$\boldsymbol{A} \cdot \boldsymbol{b} + t \cdot \boldsymbol{c} + \boldsymbol{d} \in \boldsymbol{i}$ (mv_comp)	173
mv_comp2	$\boldsymbol{A} \cdot \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d} \in \boldsymbol{i}$ (mv_comp)	173
mv_comp2a	$\boldsymbol{A} \cdot \boldsymbol{b} + t \cdot \boldsymbol{c} \subseteq \boldsymbol{i}$ (mv_comp)	173
mv_comp2b	$\boldsymbol{A} \cdot \boldsymbol{b} + t \cdot \boldsymbol{c} + \boldsymbol{d} \subseteq \boldsymbol{i}$ (mv_comp)	174
mv_comp3	$\boldsymbol{A} \cdot (\boldsymbol{b} + \boldsymbol{c}) \subseteq \boldsymbol{i}$ (mv_comp)	174

Globale Prozeduren der Hilfsmodule Teil 8			
Name	**Bearbeitung**		**Seite**
`mv_comp3`	$\boldsymbol{A} \cdot (\boldsymbol{b} + \boldsymbol{c}) \subseteq \boldsymbol{i}$	(`mv_comp`)	174
`mv_comp3`	$\boldsymbol{A} \cdot (\boldsymbol{b} + \boldsymbol{c}) + x \cdot (\boldsymbol{b} + \boldsymbol{c}) \approx r$	(`mv_comp`)	174
`mv_comp3`	$\boldsymbol{A} \cdot (\boldsymbol{b} + \boldsymbol{c}) + x \cdot (\boldsymbol{b} + \boldsymbol{c}) \in r$	(`mv_comp`)	174
`mv_comp4`	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{v} \approx \boldsymbol{r}$	(`mv_comp`)	174
`mv_comp4`	$(\boldsymbol{A} + \boldsymbol{B}) \cdot \boldsymbol{v} + \boldsymbol{w} \approx \boldsymbol{r}$	(`mv_comp`)	174
`mv_comp4`	$\boldsymbol{Au} + \boldsymbol{B}(\boldsymbol{v} + \boldsymbol{w}) \approx \boldsymbol{r}$	(`mv_comp`)	175
`normal`	Keine Überlappungen	(`stagg`)	67
`normal`	Keine Überlappungen	(`lstagg`)	92
`normal`	Keine Überlappungen	(`cstagg`)	136
`normal`	Keine Überlappungen	(`lcstagg`)	150
`PA`	Matrix: Zeilen-Permutation	(∗∗∗)	223
`pollard`	`mp`-Faktorzerlegung	(`lo_int`)	53
`Polynom`	Polynomauswertung, Punkt-Argumente	(∗∗∗)	334
`Polynom`	Polynomauswertung, Interv.-Argumente	(∗∗∗)	335
`Polynom`	Polynomauswertung, Interv.-Koeffztn.	(∗∗∗)	335
`prime_dec`	Primfaktorzerlegung, `integer`	(`lo_int`)	53
`prime_dec`	Primfaktorzerlegung, `mp`	(`lo_int`)	53
`Ps_inv`	Einschließung einer Pseudoinversen	(∗∗∗)	291
`Pv`	Vektor: Komponenten-Permutation	(∗∗∗)	223
`QR_DEC`	$\boldsymbol{QR}$-Zerlegung einer Matrix	(∗∗∗)	256
`Rand_M`	Reelle Zufallsmatrix	(`mvran`)	40
`Rand_M`	Reelle Zufalls-Intervall-Matrix	(`imvran`)	41
`Rand_M`	Reelle Zufalls-Intervall-Matrix	(`limvran`)	41

Globale Prozeduren der Hilfsmodule Teil 9		
Name	**Bearbeitung**	**Seite**
`Rand_M`	Komplexe Zufallsmatrix (`mvcran`)	41
`Rand_M`	Komplexe Zufalls-Intervall-Matrix (`imvcran`)	42
`Rand_M`	Komplexe Zufalls-Intervall-Matrix (`limvcran`)	42
`Rand_V`	Reelle Zufallsvektoren (`mvran`)	40
`Rand_V`	Reelle Zufalls-Intervallvektoren (`imvran`)	41
`Rand_V`	Reelle Zufalls-Intervallvektoren (`limvran`)	41
`Rand_V`	Komplexe Zufallsvektoren (`mvcran`)	41
`Rand_V`	Komplexe Zufalls-Intervallvektoren (`imvcran`)	42
`Rand_V`	Komplexe Zufalls-Intervallvektoren (`limvcran`)	42
`read`	`mp`-Eingabe (`lo_int`)	54
`read`	`integer`-Vektor-Eingabe (`type_def`)	54
`read`	`mp`-Vektor-Eingabe (`lo_int`)	54
`read`	`Stagg`-Eingabe (`stagg`)	81
`read`	`LStagg`-Eingabe (`lstagg`)	109
`read`	`Staggcomplex`-Eingabe (`cstagg`)	145
`read`	`LStaggcomplex`-Eingabe (`lcstagg`)	159
`read`	`Staggrvector`-Eingabe (`stgmv`)	188
`read`	`Staggrmatrix`-Eingabe (`stgmv`)	188
`read`	`LStaggrvector`-Eingabe (`lstgmv`)	198
`read`	`LStaggrmatrix`-Eingabe (`lstgmv`)	198
`read`	`Staggcvector`-Eingabe (`smvc`)	210
`read`	`Staggcmatrix`-Eingabe (`smvc`)	210
`read`	`LStaggcvector`-Eingabe (`lsmvc`)	221

Globale Prozeduren der Hilfsmodule Teil 10		
Name	**Bearbeitung**	**Seite**
`read`	`LStaggcmatrix`-Eingabe (`lsmvc`)	221
`read`	Polynomkoeffizienten-Eingabe (∗∗∗)	336
`read`	`SCPolynomial`-Eingabe (`scpoly`)	643
`read_Ab`	Bandmatrix: Eingabe von $\boldsymbol{A}$, $\boldsymbol{b}$ (∗∗∗)	311
`restglied_S`	Restgliedabschätzung (`itayl1`)	396
`Seed`	`integer`-Startwert f. Zufallsgrößen (`rand0`)	38
`seig`	Reelles, symmetr. EW-Problem (∗∗∗)	625
`set_LStagg_prec`	Präzision f. `LStagg`-Arithm. (`type_def`)	91
`set_maxprec`	Präzision f. `integer`-LZ-Arithm. (`type_def`)	49
`set_Stagg_prec`	Präzision f. `Stagg`-Arithm. (`type_def`)	65
`swap`	Vertauscht Vektorkomponenten (∗∗∗)	222
`swap_r`	Vertauscht Matrix-Zeilen (∗∗∗)	222
`swap_c`	Vertauscht Matrix-Spalten (∗∗∗)	222
`taylor_K`	Taylorkoeff. kompl. Funktionen (`citaylor`)	435
`taylor_K`	Taylorkoeff. reeller Funktionen (`litaylor`)	398
`taylor_K`	Taylorkoeff. reeller Funktionen (`itaylor`)	381
`TRANS`	`real`-Mittelpunkt (`stagg`)	70
`TRANS`	`real,real`-Mittelpunkt (`stagg`)	70
`TRANS`	`Longreal`-Mittelpunkt (`lstagg`)	96
`TRANS`	`Longreal,Longreal`-Mittelpunkt (`lstagg`)	96
`TRANS`	`complex`-Mittelpunkt (`cstagg`)	140
`TRANS`	`complex,complex`-Mittelpunkt (`cstagg`)	140

Globale Prozeduren der Hilfsmodule Teil 11

Name	Bearbeitung		Seite
TRANS	Lcomplex-Mittelpunkt	(lcstagg)	153
TRANS	Lcomplex,Lcomplex-Mittelpunkt	(lcstagg)	154
TRANS	Mittelpunktsvektor	(stgmv)	185
TRANS	Mittelpunktsvektor, dopp. Länge	(stgmv)	185
TRANS	Mittelpunktsmatrix	(stgmv)	185
TRANS	Mittelpunktsmatrix, dopp. Länge	(stgmv)	185
TRANS	Mittelpunktsvektor	(lstgmv)	195
TRANS	Mittelpunktsvektor, dopp. Länge	(lstgmv)	195
TRANS	Mittelpunktsmatrix	(lstgmv)	195
TRANS	Mittelpunktsmatrix, dopp. Länge	(lstgmv)	195
TRANS	Mittelpunktsvektor	(smvc)	204
TRANS	Mittelpunktsvektor, dopp. Länge	(smvc)	204
TRANS	Mittelpunktsmatrix	(smvc)	204
TRANS	Mittelpunktsmatrix, dopp. Länge	(smvc)	204
TRANS	Mittelpunktsvektor	(lsmvc)	215
TRANS	Mittelpunktsvektor, dopp. Länge	(lsmvc)	215
TRANS	Mittelpunktsmatrix	(lsmvc)	215
TRANS	Mittelpunktsmatrix, dopp. Länge	(lsmvc)	215
vv_comp	$\boldsymbol{a} \cdot x + \boldsymbol{b} \approx \boldsymbol{r}$	(vv_comp)	175
vv_comp	$(x + y) \cdot \boldsymbol{u} + \boldsymbol{v} + \boldsymbol{w} \approx \boldsymbol{r}$	(vv_comp)	175
vv_comp	$\boldsymbol{a} \cdot (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{b} \cdot (\boldsymbol{u} + \boldsymbol{v}) + x \approx r$	(vv_comp)	175
vv_comp	$\boldsymbol{a} \cdot (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{b} \cdot (\boldsymbol{u} + \boldsymbol{v}) + x \in r$	(vv_comp)	175
write	mp-Ausgabe	(lo_int)	54

Globale Prozeduren der Hilfsmodule Teil 12		
Name	**Bearbeitung**	**Seite**
`write`	`integer`-Vektor-Ausgabe (`type_def`)	54
`write`	`mp`-Vektor-Ausgabe (`lo_int`)	54
`write`	`Stagg`-Ausgabe (`stagg`)	81
`write`	`LStagg`-Ausgabe (`lstagg`)	109
`write`	`Staggcomplex`-Ausgabe (`cstagg`)	145
`write`	`LStaggcomplex`-Ausgabe (`lcstagg`)	159
`write`	`Staggrvector`-Ausgabe (`stgmv`)	188
`write`	`Staggrmatrix`-Ausgabe (`stgmv`)	188
`write`	`LStaggrvector`-Ausgabe (`lstgmv`)	198
`write`	`LStaggrmatrix`-Ausgabe (`lstgmv`)	198
`write`	`Staggcvector`-Ausgabe (`smvc`)	210
`write`	`Staggcmatrix`-Ausgabe (`smvc`)	210
`write`	`LStaggcvector`-Ausgabe (`lsmvc`)	221
`write`	`LStaggcmatrix`-Ausgabe (`lsmvc`)	221
`write`	Ausgabe der Polynomkoeffizienten (∗∗∗)	336
`write`	`SCPolynomial`-Ausgabe (`scpoly`)	643
`ZIELKE`	liefert Zielke-Matrix (`(l)mv_exap`)	166

Anhang D

Globale Funktionen der Hilfsmodule

Die nachfolgenden Tabellen enthalten, mit Ausnahme der mathematischen Standardfunktionen, in alphabetischer Reihenfolge alle von den Hilfsmodulen exportierten Funktionen.

Mit (...) geklammerte Namen sind die Namen der jeweils exportierenden Hilfsmodule. Das Symbol ($***$) bedeutet, daß verschiedene Funktionen mit gleichem Namen von mehreren Modulen exportiert werden. Weitere Informationen liefert die in der rechten Spalte angegebene Seite dieses Buches. `Lgreal` ist eine Abkürzung für den Typ `Longreal`.

Die mathematischen Standardfunktionen der Hilfsmodule findet man in den Tabellen auf den Seiten: 78, 79, 106, 107, 144, 158, 351, 380, 399, 404, 421, 434, 461, 654.

Mathematische Konstanten stehen in Form von Pascal–Funktionen ohne Funktionsargumente zur Verfügung. Nach Datentypen getrennte Konstanten findet man auf den Seiten: 80, 108, 640, 652.

Globale Funktionen der Hilfsmodule		
Name	**Bearbeitung**	**Seite**
`abs`	Betrag einer `mp`-Zahl (`lo_int`)	51
`abs`	Beträge reeller Vektoren oder Matrizen ($***$)	227
`abs`	Beträge komplexer Vektoren oder Matrizen ($***$)	228

Globale Funktionen der Hilfsmodule Teil 2		
Name	**Bearbeitung**	**Seite**
Abs_Min	AbsMin $= \mathrm{Min}\{|r| : r \in [a, b]\}$ (i_util)	639
Abs_Min	AbsMin $= \mathrm{Min}\{|r| : r \in [a, b]\}$ (rrgi_util)	653
Abs_Max	AbsMax $= \mathrm{Max}\{|r| : r \in [a, b]\}$ (i_util)	639
Abs_Max	AbsMax $= \mathrm{Max}\{|r| : r \in [a, b]\}$ (rrgi_util)	652
binomial	Binomialkoeffizienten, Longreal (lo_int)	52
binomial_mp	Binomialkoeffizienten, mp (lo_int)	52
blow	ε-Aufblähung von Stagg (stagg)	69
blow	ε-Aufblähung von LStagg (lstagg)	92
blow	ε-Aufblähung von Staggcomplex (cstagg)	139
blow	ε-Aufblähung von LStaggcomplex (lcstagg)	153
blow	ε-Aufblähung von CIPolynomial (cipoly)	642
blow	ε-Aufblähung von SCPolynomial (scpoly)	643
blow_z	ε-Aufblähung von Lcivector (blow_z)	650
blow_z	ε-Aufblähung von Lcimatrix (blow_z)	650
change_length	Präzisionsanpassung, Stagg (stagg)	66
change_length	Präzisionsanpassung, LStagg (lstagg)	92
change_length	Präzis.-Anpassg., Staggcomplex (cstagg)	135
change_length	Präzis.-Anpassg., LStaggcomplex (lcstagg)	149
change_length	Präzis.-Anpassg., Staggrvector (stgmv)	182
change_length	Präzis.-Anpassg., Staggrmatrix (stgmv)	182
change_length	Präzis.-Anpassg., LStaggrvector (lstgmv)	191
change_length	Präzis.-Anpassg., LStaggrmatrix (lstgmv)	192
change_length	Präzis.-Anpassg., Staggcvector (smvc)	201

Globale Funktionen der Hilfsmodule Teil 3		
Name	**Bearbeitung**	**Seite**
change_length	Präzis.-Anpassg., Staggcmatrix (smvc)	202
change_length	Präzis.-Anpassg., LStaggcvector (lsmvc)	212
change_length	Präzis.-Anpassg., LStaggcmatrix (lsmvc)	213
cival	Staggcomplex ⊆ cinterval (cstagg)	139
cival	LStaggcomplex ⊆ Lcinterval (lcstagg)	153
compl (cstagg)	Staggcomplex := compl(Stagg,Stagg)	140
compl (cstagg)	Staggcomplex := compl(Stagg), reell!	140
compl (lcstagg)	LStaggcomplex := compl(LStagg,LStagg)	154
compl (lcstagg)	LStaggcomplex := compl(LStagg), reell!	154
conj (smvc)	Konjug. Vektor vom Typ Staggcvector	206
conj (smvc)	Konjug. Matrix vom Typ Staggcmatrix	206
conj (lsmvc)	Konjug. Vektor vom Typ LStaggcvector	217
conj (lsmvc)	Konjug. Matrix vom Typ LStaggcmatrix	217
convert	[kompl.dotpr.] ⊆ Staggcomplex (cstagg)	135
convert_LStagg	[dotprec1,dotprec2] ⊆ LStagg (lstagg)	92
convert_Stagg	[dotprec1,dotprec2] ⊆ Stagg (stagg)	66
dfValue	Einschließung von $f'(x)$, rrG (ddf_rrg)	352
ddfValue	Einschließung von $f''(x)$, rrG (ddf_rrg)	352
DerivConst	Konstante vom Typ DerivType (ddf_rrg)	364
DerivVar	Variable vom Typ DerivType (ddf_rrg)	352
diam	Oberschr. Intervall-Durchmesser (stagg)	67
diam	Oberschr. Intervall-Durchmesser (lstagg)	93
diam	Oberschr. Intervall-Durchmesser (cstagg)	136

Globale Funktionen der Hilfsmodule Teil 4		
Name	**Bearbeitung**	**Seite**
`diam`	Oberschr. Intervall-Durchmesser (`lcstagg`)	150
`EkNorm`	Euklidische Normen (∗∗∗)	230
`EmptyList`	Listen, (Intervall,Zahl) (`lst1_ari`)	646
`EmptyList`	Listen, (Vektor,Zahl) (`lst_ari`)	648
`factorial`	$n!$, $1 \leq n \leq 65535$ (`lo_int`)	51
`factorial`	$n!$, $1 \leq n \leq 26$ (`lo_int`)	51
`factorial_rrGi`	$n!$, Ergebnistyp: `rrGinterval` (`rrgi_util`)	653
`factorial_sum_mp`	$1! + \ldots + n!$, $1 \leq n \leq 65535$ (`lo_int`)	51
`factorial_sum_mp`	$1! + \ldots + n!$, $1 \leq n \leq 21$ (`lo_int`)	51
`fValue`	Einschließung v. $f(x)$, `rrG` (`ddf_rrg`)	352
`Fyi`	Listen, (Vektor,Zahl) (`lst_ari`)	648
`get_LStagg_prec`	Präzision f. `LStagg`-Arithm. (`type_def`)	91
`get_maxprec`	Präzision, `integer`-Arithm. (`type_def`)	49
`get_Stagg_prec`	Präzision f. `Stagg`-Arithm. (`type_def`)	66
`GetFyi`	Listen, (Intervall,Zahl) (`lst1_ari`)	646
`GetInt`	Listen, (Intervall,Zahl) (`lst1_ari`)	646
`ggT`	`ggT(mp,mp): mp` (`lo_int`)	52
`ggT`	`ggT(integer,integer):integer` (`lo_int`)	52
`ggT`	`ggT(intvector): integer` (`lo_int`)	52
`ggT`	`ggT(mpvector): mp` (`lo_int`)	52
`Head`	Listen, (Intervall,Zahl) (`lst1_ari`)	647
`Head`	Listen, (Vektor,Zahl) (`lst_ari`)	648
`herm`	Hermite Operator: `Staggcmatrix` (`smvc`)	206

Globale Funktionen der Hilfsmodule Teil 5		
Name	**Bearbeitung**	**Seite**
`herm`	Hermitescher Operator: `LStaggcmatrix` (`lsmvc`)	217
`hull`	`Stagg` +* `Stagg`, konvexe Hülle (`stagg`)	67
`hull`	`LStagg` +* `LStagg`, konvexe Hülle (`lstagg`)	93
`id`	Einheitsmatrix vom Typ `Staggrmatrix` (`stgmv`)	184
`id`	Einheitsmatrix vom Typ `LStaggrmatrix` (`lstgmv`)	194
`id`	Einheitsmatrix vom Typ `Staggcmatrix` (`smvc`)	204
`id`	Einheitsmatrix vom Typ `LStaggcmatrix` (`lsmvc`)	215
`Im`	$\Im$(`Staggcomplex`) : `Stagg` (`cstagg`)	139
`Im`	$\Im$(`LStaggcomplex`) : `LStagg` (`lcstagg`)	153
`Im`	$\Im$(`Staggcvector`) : `Staggrvector` (`smvc`)	203
`Im`	$\Im$(`Staggcmatrix`) : `Staggrmatrix` (`smvc`)	203
`Im`	$\Im$(`LStaggcvector`) : `LStaggrvector` (`lsmvc`)	214
`Im`	$\Im$(`LStaggcmatrix`) : `LStaggrmatrix` (`lsmvc`)	214
`inf`	Intervall-Untergrenze, `Stagg` (`stagg`)	67
`inf`	Intervall-Untergrenze, `LStagg` (`lstagg`)	93
`inf`	kompl. Interv.-Untergr., `Staggcomplex` (`cstagg`)	136
`inf`	kompl. Interv.-Untergr., `LStaggcomplex` (`lcstagg`)	150
`inf`	Intervall-Untergrenzen, `Staggrvector` (`stgmv`)	183
`inf`	Intervall-Untergrenzen, `Staggrmatrix` (`stgmv`)	183
`inf`	Intervall-Untergrenzen, `LStaggrvector` (`lstgmv`)	193
`inf`	Intervall-Untergrenzen, `LStaggrmatrix` (`lstgmv`)	193
`inf`	kompl. Interv.-Untergrenzen, `Staggcvector` (`smvc`)	205
`inf`	kompl. Interv.-Untergrenzen, `Staggcmatrix` (`smvc`)	205

Globale Funktionen der Hilfsmodule Teil 6		
Name	**Bearbeitung**	**Seite**
inf (lsmvc)	kompl. Intervall-Untergrenzen, LStaggcvector	216
inf (lsmvc)	kompl. Intervall-Untergrenzen, LStaggcmatrix	216
Int	Listen, (Vektor,Zahl) (lst_ari)	647
INTEGRAL	Integraleinschließung (iromberg)	551
ival	Stagg $\subseteq$ interval (stagg)	69
ival	LStagg $\subseteq$ Linterval (lstagg)	95
kgV	kgV(mp,mp): mp (lo_int)	53
kgV	kgV(integer,integer):integer (lo_int)	53
kgV	kgV(intvector): integer (lo_int)	53
kgV	kgV(mpvector): mp (lo_int)	53
kgVmp	kgV(intvector): mp (lo_int)	53
L1Norm	L1–Normen (∗∗∗)	230
lconvert	[kompl.dotpr.] $\subseteq$ LStaggcomplex (lcstagg)	149
Length	Listen, (Intervall,Zahl) (lst1_ari)	647
Length	Listen, (Vektor,Zahl) (lst_ari)	648
LStaggcintval	LStaggcomplex := Lcinterval; (lcstagg)	154
LStaggcompl	LStaggcomplex := Lcomplex; (lcstagg)	154
LStaggcompl	LStaggcomplex := (Lgreal, Lgreal); (lcstagg)	155
LStaggcompl	LStaggcomplex := (Longreal, 0); (lcstagg)	155
LStaggintval	LStagg := Longreal; Pkt.-Intervall (lstagg)	96
LStaggintval	LStagg := Linterval; (lstagg)	96
LStaggintval	LStagg := Longreal +∗ Lomgreal; (lstagg)	97
LRand	Zufallszahl aus (0, 1), Longreal (rand0)	38

Globale Funktionen der Hilfsmodule Teil 7		
Name	**Bearbeitung**	**Seite**
MakePair	Listen, (Intervall,Zahl) (lst1_ari)	646
MakePair	Listen, (Vektor,Zahl) (lst_ari)	647
MaxNorm	Maximum–Normen (∗∗∗)	230
MaxRelDiam	Maximum aller rel. Durchmesser (mv_util)	649
mid	Stagg-Mittelpunkt (stagg)	67
mid	LStagg-Mittelpunkt (lstagg)	93
mid	Staggcomplex-Mittelpunkt (cstagg)	136
mid	LStaggcomplex-Mittelpunkt (lcstagg)	150
mid	Staggrvector-Mittelpunkte (stgmv)	183
mid	Staggrmatrix-Mittelpunkte (stgmv)	183
mid	LStaggrvector-Mittelpunkte (lstgmv)	193
mid	LStaggrmatrix-Mittelpunkte (lstgmv)	193
mid	Staggcvector-Mittelpunkte (smvc)	205
mid	Staggcmatrix-Mittelpunkte (smvc)	205
mid	LStaggcvector-Mittelpunkte (lsmvc)	216
mid	LStaggcmatrix-Mittelpunkte (lsmvc)	216
mid	rrGinterval-Mittelpunkt (rrgi_util)	653
Next	Listen, (Intervall,Zahl) (lst1_ari)	647
Next	Listen, (Vektor,Zahl) (lst_ari)	648
NormC	Spaltensummen–Norm (∗∗∗)	231
NormF	Frobenius–Norm (∗∗∗)	231
NormG	Gesamtnorm (∗∗∗)	231
NormR	Zeilensummen–Norm (∗∗∗)	231

Globale Funktionen der Hilfsmodule Teil 8		
Name	**Bearbeitung**	**Seite**
null	Nullvektor vom Typ Staggrvector (stgmv)	184
null	Nullmatrix vom Typ Staggrmatrix (stgmv)	184
null	Nullvektor vom Typ LStaggrvector (lstgmv)	194
null	Nullmatrix vom Typ LStaggrmatrix (lstgmv)	194
null	Nullvektor vom Typ Staggcvector (smvc)	203
null	Nullmatrix vom Typ Staggcmatrix (smvc)	203
null	Nullvektor vom Typ LStaggcvector (lsmvc)	214
null	Nullmatrix vom Typ LStaggcmatrix (lsmvc)	214
odd	odd-Test einer mp-Zahl (lo_int)	51
Pi_lsm	π in maximaler LStagg-Genauigkeit (lstagg)	108
Pi_LStagg	π in aktueller LStagg-Genauigkeit (lstagg)	108
Pih_lsm	$\pi/2$ in maxim. LStagg-Genauigkeit (lstagg)	108
Piv_lsm	$\pi/4$ in maxim. LStagg-Genauigkeit (lstagg)	108
Pi_sm	π in maximaler Stagg-Genauigkeit (stagg)	80
Pi_Stagg	π in aktueller Stagg-Genauigkeit (stagg)	80
Pih_sm	$\pi/2$ in maximaler Stagg-Genauigkeit (stagg)	80
Piv_sm	$\pi/4$ in maximaler Stagg-Genauigkeit (stagg)	80
Poly_Horn	Polynom-Einschließung nach Horner $(***)$	333
power	(mp-Zahl)^n, $n = 0, 1, 2, \ldots$ (lo_int)	51
power	$\boldsymbol{A}^m$, $m = 0, 1, 2, \ldots$ $(***)$	246
power	(rrG)^n, $n = 0, 1, 2, \ldots$ (rrgi_util)	653
power	z^z, rrGcinterval,Lcinterval (lci_util)	654
power_rrG	z^z, Ergebn.-Typ: rrGcinterval (lci_util)	654

Globale Funktionen der Hilfsmodule Teil 9		
Name	**Bearbeitung**	**Seite**
permutation	Permutation, Longreal (lo_int)	52
permutation_mp	Permutation, mp (lo_int)	52
QR	$\boldsymbol{A} = \boldsymbol{QR}$, Approximation von $\boldsymbol{Q}$ (∗∗∗)	256
QR	$\boldsymbol{A} = \boldsymbol{QR}$, Einschließung von $\boldsymbol{Q}$ (∗∗∗)	256
RelDiam	Oberschr. für rel. Intervalldurchm. (i_util)	640
Rand	Zufallszahl aus $(0, 1)$, real (rand0)	38
Rand	Zufallszahl aus (a, b), real (random)	38
Rand	Zufallszahl aus (a, b), Longreal (random)	38
Rand	reelle Zufallsintervalle, interval (irandom)	39
Rand	reelle Zufallsintervalle, Linterval (irandom)	39
Rand	komplexe Zufallszahlen, complex (crandom)	40
Rand	komplexe Zufallszahlen, Lcomplex (crandom)	40
Rand	kompl. Zuf.-Interv., cinterval (cirandom)	40
Rand	kompl. Zuf.-Interv., Lcinterval (cirandom)	40
RandLn	Logarithm. vert. Zuf.-Zahlen, real (random)	39
RandLn	Logar. vert. Zufallszahlen, Lgreal (random)	39
RandLn	Logar. vert. Zufallszahlen, real (random)	39
RandLn	Logar. vert. Zufallszahlen, Lgreal (random)	39
RandLn	Logar. vert. Intervalle, interval (irandom)	39
RandLn	Logar. vert. Intervalle, Linterval (irandom)	39
Re	$\Re$(Staggcomplex) : Stagg (cstagg)	139
Re	$\Re$(LStaggcomplex) : LStagg (lcstagg)	153
Re	$\Re$(Staggcvector) : Staggrvector (smvc)	203

Globale Funktionen der Hilfsmodule Teil 10		
Name	**Bearbeitung**	**Seite**
Re	$\Re$(Staggcmatrix) : Staggrmatrix (smvc)	203
Re	$\Re$(LStaggcvector) : Staggrvector (lsmvc)	214
Re	$\Re$(LStaggcmatrix) : LStaggrmatrix (lsmvc)	214
round	Stagg + 0; $x[1] \rightarrow$ integer-Zahl (stagg)	69
round	LStagg + 0; $x[1] \rightarrow$ integer-Zahl (lstagg)	95
sqrt	Ganzzahliger Anteil von $\sqrt{\texttt{mp-Zahl}}$ (lo_int)	51
sup	Intervall-Obergrenze, Stagg (stagg)	67
sup	Intervall-Obergrenze, LStagg (lstagg)	93
sup	kompl. Interv.-Obergr., Staggcomplex (cstagg)	136
sup (lcstagg)	komplexe Intervall-Obergrenze, LStaggcomplex	150
sup	Intervall-Obergrenzen, Staggrvector (stgmv)	183
sup	Intervall-Obergrenzen, Staggrmatrix (stgmv)	183
sup	Intervall-Obergrenzen, LStaggrvector (lstgmv)	193
sup	Intervall-Obergrenzen, LStaggrmatrix (lstgmv)	193
sup	kompl. Interv.-Obergr., Staggcvector (smvc)	205
sup	kompl. Interv.-Obergr., Staggcmatrix (smvc)	205
sup	kompl. Interv.-Obergr., LStaggcvector (lsmvc)	216
sup	kompl. Interv.-Obergr., LStaggcmatrix (lsmvc)	216
Staggcintval	Staggcomplex := cinterval; (cstagg)	140
Staggcompl	Staggcomplex := complex; (cstagg)	141
Staggcompl	Staggcomplex := (real,real); (cstagg)	141
Staggcompl	Staggcomplex := (real, 0); (cstagg)	141
Staggintval	Stagg := real; Punktintervall (stagg)	70

Globale Funktionen der Hilfsmodule Teil 11		
Name	**Bearbeitung**	**Seite**
`Staggintval`	`Stagg` := `interval`; (`stagg`)	70
`Staggintval`	`Stagg` := `real` +* `real`; (`stagg`)	70
`trace`	Spur($\boldsymbol{A}$), Spur($\boldsymbol{AB}$), Approximationen (***)	236
`trace_i`	Spur($\boldsymbol{A}$), Spur($\boldsymbol{AB}$), Einschließungen (***)	237
`transp`	Transponierte Matrix: `Staggrmatrix` (`stgmv`)	184
`transp`	Transponierte Matrix: `LStaggrmatrix` (`lstgmv`)	194
`transp`	Transponierte Matrix: `Staggcmatrix` (`smvc`)	206
`transp`	Transponierte Matrix: `LStaggcmatrix` (`lsmvc`)	217
`UlpAcc`	Testet auf maximal n ulp Unterschied (`i_util`)	641
`UlpAcc`	Testet auf max. n ulp Unterschied (`mv_util`)	649

Anhang E

Programmbeispiele

Die folgenden Tabellen enthalten meist kurze **PASCAL-XSC** Programme der BCD–Version zu den jeweils angegebenen numerischen Grundaufgaben.

Automatische Differentiation	**Programm**
Einschließungen von $f(x), f'(x), f''(x)$	`tstddf_rrg1`, S.353
Wahl des geeigneten Funktionsterms	`tstddf_rrg2`, S.356
Vermeidung von Intervallüberschätzungen	`tstddf_rrg3`, S.361
Einschließung der Taylorkoeffizienten, `real`	`tstitayl1`, S.378
Einschließung der Taylorkoeffizienten, `Longreal`	`tstlitayl1`, S.400
Wahl des geeigneten Funktionsterms	`tstitayl2`, S.382
Wahl des geeigneten Funktionsterms	`tstitayl3`, S.384
Wahl des geeigneten Funktionsterms	`tstitayl4`, S.387
Restgliedabschätzung	`tstitayl5`, S.390
Gradienten-Einschließung	`tstigrad1`, S.406
Gradienten-Einschließung	`tstigrad2`, S.408
Grad.-Einschließung, geeigneter Funktionsterm	`tstigrad3`, S.411
Einschließung der Jacobi-Matrix	`tstigrad4`, S.417
Nur Approximation stationärer Punkte	`tsthesse1`, S.425

Automatische Differentiation	**Programm**
Taylorkoeffizienten komplexer Funktionen	`tstcitayl1`, S.432
Ableitungen komplexer Funktionen	`tstcitayl2`, S.435

Differenzen-Gleichungen	**Programm**
Skalare Differenzen-Glg. zum Basistyp `real`	`tstdiff`, S.320
Skalare Differenzen-Glg. zum Basistyp `Longreal`	`tstldiff`, S.322
Vektor. Differenzen-Glg.: Drehung um y-Achse	`tstdiffv`, S.327
Vektor. Differenzen-Glg.: Drehung in x, y-Ebene	`tstldiffv`, S.329

Eigenwerte	**Programm**
Approximation aller Eigenpaare	`test_eigen`, S.591
EigenP.-Einschlg. mit gegebener Approximation	`eig_ex1`, S.595
wie oben, aber zum Basistyp `Longreal`	`leig_ex1`, S.598
EP.-Einschlg. mit gegeb. Approx., Intervall-Matrix	`ieig_ex1`, S.600
wie oben, aber zum Basistyp `Longreal`	`lieig_ex1`, S.603
EP-Einschlg. mit Approximation, `cmatrix`	`ceig_ex1`, S.605
EP-Einschlg. mit Approximation, `cimatrix`	`cieig_ex1`, S.608
Einschließung bzw. Approxim. aller Eigenpaare	`eig_all1`, S.612
wie oben, aber für Intervallmatrizen	`eig_all2`, S.614
Einschlg./Approxim. aller EPre herm. Matrizen	`eig_all3`, S.616
wie oben, aber für Intervallmatrizen	`eig_all4`, S.618
Eigenpaare einer Boothroyd/Dekker Matrix	`eig_all5`, S.620
Eigenpaare reeller Zufallsmatrizen	`eig_all6`, S.623
Eigenwerte einer reellen, symmetrischen Matrix	`lseig_ex2`, S.626
Eigenwerte einer symmetrischen Zielke-Matrix	`lseig_ex1`, S.627
Symmetrische Matrix mit irrationalen Elementen	`tstliseig1`, S.630

Eigenwerte	Programm
Einschlg. aller EW einer hermiteschen Matrix	`lheig_ex1`, S.633
wie oben, aber irrationale Matrix-Elemente	`tstliheig1`, S.635

Globale Optimierung	Programm
Reelle eindimensionale Funktionen	`gop1_ex`, S.500
Reelle mehrdimensionale Funktionen	`gop_ex`, S.513

Integer-Langzahlarithmetik	Programm
Eingabe mit einem string, Beisp. zur Datenstruktur	`lint_1`, S.55
$\sum_{k=0}^{n} 2^k \binom{n}{k} = 3^n$	`lint_2`, S.56
Primfaktorzerlegung	`lint_3`, S.57
Pollard's Faktorzerlegung	`lint_3`, S.59

Intervall Staggered-Arithmetik, Typ `Stagg`	Programm
$(a \cdot b)/a = b$; Einschließung von b	`stagg1`, S.63
`change_length, write, Pi_Stagg, Pi_sm`	`stagg2`, S.83
Staggered Größen variabler Präzision	`stagg3`, S.85
Legendre Polynome der Ordnung n	`stagg4`, S.86
Einschließung rationaler Ausdrücke	`stagg5`, S.89

Intervall Staggered-Arithmetik, Typ `LStagg`	Programm
Legendre Polynom der Ordnung 5, Hornerschema	`lstg1`, S.111
Logistik-Gleichung	`lstg2`, S.115
Einschließung von $\arcsin(x)$	`lstg3`, S.125
Einschließung der Kubikwurzel	`lstg4`, S.129
Einschließung reeller, arithmetischer Ausdrücke	`lstg5`, S.131

Komplexe Intervall Staggered-Arithmetik	**Programm**
Test der komplexen Quadratwurzel: $z = \sqrt{z^2}$	`cstg1`, S.146
Test der komplexen Division, Typ `LStaggcomplex`	`lcstg1`, S.160

Komplexe Polynomnullstellen	**Programm**
$z^4 - (5+3i)z^3 + \ldots + 120 + 90i$	`cproot1`, S.532
NStn.-Einschließung mit gegebener Approximation	`cproot2`, S.535
Nullstellen-Einschließung in hoher Präzision	`scproot2`, S.537
Einschließung dicht benachbarter Nullstellen	`scproot1`, S.542

Lineare Gleichungssysteme	**Programm**
$\boldsymbol{Ax} = \boldsymbol{b}$; $\boldsymbol{A}$: Boothroyd/Dekker-Matrix	`tstllss1`, S.301
$\boldsymbol{Ax} = \boldsymbol{b}$; $\boldsymbol{A}$: Boothroyd/Dekker-Block-Matrix	`tstllss2`, S.303
$\boldsymbol{Ax} = \boldsymbol{b}$; Über- oder unterbestimmte Systeme	`tstllss3`, S.307

Lineare Gleichungssysteme, Bandmatrizen	**Programm**
Einschließung der Tschebyscheff-Polynomwerte	`tstlband`, S.313
$\boldsymbol{Bx} = \boldsymbol{b}$; $\boldsymbol{B}$: symmetrische Toeplitz-Matrix	`tstlband`, S.315

Matrizen/Vektoren	**Programm**
Realisierung von Blockmatrizen	`tstblock`, S.168
Skalarprodukt–Einschließung	`stgm1`, S.179
Skalarprodukt–Einschließung in hoher Präzision	`stgm2`, S.189
Verträglichkeit von Matrix/Vektor-Normen	`tstnorm1`, S.232
Winkel zwischen Vektoren	`tstnorm2`, S.233
Frobenius-Norm $\|\boldsymbol{A}\|_F = \sqrt{\mathrm{Spur}(\boldsymbol{A}^* \cdot \boldsymbol{A})}$	`tsttrace`, S.238
Determinanten der Boothroyd/Dekker-Matrizen	`tstdet1`, S.241
Determinante einer singulären Matrix, $n = 3$	`tstdet2`, S.243
Determinante einer singulären Matrix, $n = 5$	`tstdet3`, S.244

Matrizen/Vektoren	Programm
Matrix-Potenzen, Typ `Lrmatrix`	`tstpower1`, S.247
Matrix-Potenzen, Typ `LStaggrmatrix`	`tstpower2`, S.247
LU–Zerlegung: $\boldsymbol{P} \cdot \boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U}$	`lu1`, S.251
LU–Zerlegung in Staggered Arithmetik	`lu2`, S.253
QR–Zerlegung einer Matrix, Approxim. von $\boldsymbol{Q}, \boldsymbol{R}$	`tstqr`, S.257
QR–Zerlegung einer Matrix, Einschließung von $\boldsymbol{Q}, \boldsymbol{R}$	`tststqr`, S.258
Einschließung einer Orthonormalbasis	`tststqr1`, S.260
Einschließung der Konditionszahl einer Matrix	`tstkond1`, S.285

Matrix–Inversion	Programm
Einschlg. der Inversen einer komplexen Blockmatrix	`tstinv1`, S.279
Einschließung der Inversen einer Dekker-Matrix	`tstinv2`, S.280
Einschlg. der Inversen einer Dekker-Produkt-Matrix	`tstABinv1`, S.283
Inversion eines Produkts aus nichtquadrt. Faktoren	`tstABinv2`, S.287
Einschließung einer Pseudoinversen	`tstPsinv1`, S.293

Nichtlineare Gleichungen	Programm
Beispiel für mehrfache Nullstellen	`tstnlfz1`, S.441
Kurvendiskussion	`nlfz3_1`, S.462
Kurvendiskussion	`nlfz3_2`, S.467

Nichtlineare Gleichungssysteme	Programm
Nullstellen der Funktion von Hansen	`nlss_ex`, S.483
Stationäre Punkte der Rosenbrock-Funktion	`nlss_ex1`, S.488
$e^{x_1^2+x_2^2} - 3 = 0; \quad x_1 + x_2 - \sin[3 \cdot (x_1 - 2x_2)] = 0$	`nlss_ex2`, S.493

Numerische Integration	**Programm**
Beispiele auch mit kritischen Integranden	`romberg_ex1`, S.552
Einschließung von $\int_0^{\pi/4} \ln(\tan(x))\,dx$	`romberg_ex2`, S.566
Einschließung von $\int_0^1 e^x \ln(x)\,dx$	`romberg_ex3`, S.569
Einschließung von $\int_0^1 e^{x\cdot\arcsin(x)} \cdot \sqrt[3]{1-x}\,dx$	`romberg_ex4`, S.574
Einschließung der Fresnel-Integrale	`romberg_ex5`, S.577
Einschließung der Eulerschen Konstanten C	`romberg_ex6`, S.581
Einschließung von $\int_0^1 \ln^{16}(x)/\sqrt{1-x^2}\,dx$	`romberg_ex7`, S.584

Polynomauswertung	**Programm**
Iteratives Residuum-Verfahr./Intervall-Hornersch.	`tstrlpev`, S.337
Polynom-Einschließung für Intervallargumente	`tstrlpev1`, S.339
Reelle Polynome mit Intervall-Koeffizienten	`tstrlpev2`, S.342
Komplexe Polynome mit Punkt-Koeffizienten	`tstcpev1`, S.345
Komplexe Polynome mit Intervall-Koeffizienten	`tstcpev2`, S.347

Zufallsgeneratoren	**Programm**
Reelle und kompl. Zufalls-Zahlen u. Zuf.-Intervalle	`rantst`, S.43

Anhang F

Hilfsmodule

Tabelle der Hilfsmodule der BCD-Version Teil 1		
Modul	**Anwendungen**	**Seite**
`abciinv`	Einschließung von $(\boldsymbol{AB})^{-1}$, $\boldsymbol{A},\boldsymbol{B}$: `cimatrix`	282
`abcinv`	Einschließung von $(\boldsymbol{AB})^{-1}$, $\boldsymbol{A},\boldsymbol{B}$: `cmatrix`	282
`abiinv`	Einschließung von $(\boldsymbol{AB})^{-1}$, $\boldsymbol{A},\boldsymbol{B}$: `imatrix`	282
`abinv`	Einschließung von $(\boldsymbol{AB})^{-1}$, $\boldsymbol{A},\boldsymbol{B}$: `rmatrix`	282
`abliinv`	Einschließung von $(\boldsymbol{AB})^{-1}$, $\boldsymbol{A},\boldsymbol{B}$: `Limatrix`	282
`ablinv`	Einschließung von $(\boldsymbol{AB})^{-1}$, $\boldsymbol{A},\boldsymbol{B}$: `Lrmatrix`	282
`band`	$\boldsymbol{Bx} = \boldsymbol{b}$, Bandmatrix $\boldsymbol{B}$: `rmatrix`, `imatrix`	310
`block`	Blockmatrizen $\boldsymbol{B}$: `(L)rmatrix, (L)cmatrix`	167
`blow_z`	Aufblähung kompl. Intervallmatrizen/Vektoren	650
`ceig`	$\boldsymbol{Ax} = \lambda \cdot \boldsymbol{x}$, $\boldsymbol{A}$: `cmatrix`;	594
`cieig`	$\boldsymbol{Ax} = \lambda \cdot \boldsymbol{x}$, $\boldsymbol{A}$: `cimatrix`;	594
`cilss`	Lin. GlgS.: $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{A}$: `cimatrix`;	299
`cilu`	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: `cimatrix`;	250
`cipoly`	Routinen zum Typ `CiPolynomial`	642
`cirandom`	Komplexe Zufallsintervalle	40

Tabelle der Hilfsmodule der BCD-Version Teil 2		
Modul	**Anwendungen**	**Seite**
`citaylor`	Komplexe Taylor-Koeffizienten, `rrGcinterval`	431
`clss`	Lin. GlgS.: $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{A}$: `cmatrix`;	299
`clu`	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: `cmatrix`;	250
`cpeval`	Einschießung komplexer Polynomwerte	333
`cpoly`	Routinen zum Typ `CPolynomial`	641
`cpzero`	Nullstelleneinschließung komplexer Polynome	530
`crandom`	Komplexe Zufallszahlen	40
`cstagg`	Komplexe Staggered Arithmetik, `Staggcomplex`	134
`cswap`	Matrix/Vektor-Tauschoperationen, `complex`	222
`d3f_rrg`	Intervall–Differentiationsarithmetik, bis f'''	458
`ddf_rrg`	Intervall–Differentiationsarithmetik, bis f''	350
`differ`	Differenzengleichungen zum Typ `real`	317
`eig`	$\boldsymbol{Ax} = \lambda \cdot \boldsymbol{x}$, $\boldsymbol{A}$: `rmatrix`;	594
`eig_app`	Approximation aller Eigenpaare	589
`eig_herm`	Eigenpaare hermitescher Matrizen, `cmatrix`	611
`eig_reell`	Eigenpaare reeller Matrizen zum Typ `rmatrix`	611
`gop`	Globale Optimierung mehrdim. Funktionen	511
`gop1`	Globale Optimierung eindim. Funktionen	498
`heig`	Eigenwerte hermitescher Matrizen, `cmatrix`	632
`hess_ari`	Funktionswert, Gradient, Hessematrix	419
`i_util`	Ergänzungen zum Systemmodul `i_ari`	639
`iblock`	Blockmatrizen $\boldsymbol{B}$: `(L)imatrix, (L)cimatrix`	167
`ieig`	$\boldsymbol{Ax} = \lambda \cdot \boldsymbol{x}$, $\boldsymbol{A}$: `imatrix`;	594

Tabelle der Hilfsmodule der BCD-Version Teil 3		
Modul	**Anwendungen**	**Seite**
`ieig_herm`	Eigenpaare hermitescher Matrizen, `cimatrix`	611
`ieig_reell`	Eigenpaare reeller Matrizen zum Typ `imatrix`	611
`igrad`	Gradienten-Einschließung	403
`iheig`	Eigenwerte hermitescher Matrizen, `cimatrix`	632
`ilss`	Lin. GlgS.: $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{A}$: `imatrix`;	299
`ilu`	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: `imatrix`;	250
`imvcran`	Komplexe Zufalls-Intervallmatizen/Vektoren	42
`imvran`	Reelle Zufalls-Intervall-Matrizen/Vektoren	41
`int_mv`	Ein-/Ausgaberoutinen zum Typ `intvector`	641
`inv`	Inversion reeller Matrizen zum Typ `rmatrix`	265
`invc`	Inversion komplexer Matrizen, `cmatrix`	265
`invcstg`	Matrix–Inversion zum Typ `Staggcmatrix`	274
`invlstg`	Matrix–Inversion zum Typ `LStaggrmatrix`	274
`invstg`	Matrix–Inversion zum Typ `Staggrmatrix`	274
`irandom`	Reelle Zufallsintervalle	39
`iromberg`	Einschließung bestimmter Integrale	551
`iseig`	Eigenwerte symmetrischer Matrizen, `imatrix`	625
`itayl1`	Taylorreihe: Restgliedabschätzung	396
`itaylor`	Reelle Taylor-Koeffizienten, `rrGinterval`	377
`lband`	$\boldsymbol{Bx} = \boldsymbol{b}$, Bandmatrix $\boldsymbol{B}$: `Lrmatrix`,`Limatrix`	310
`lci_util`	Ergänzungen zum Systemmodul `lci_ari`	654
`lclu`	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: `Lcmatrix`;	250
`lcstagg`	Komplexe Intervall-Staggered-Arithmetik	148

Tabelle der Hilfsmodule der BCD-Version Teil 4		
Modul	**Anwendungen**	**Seite**
`lcswap`	Matrix/Vektor-Tauschoperationen, `Lcomplex`	222
`ldiffer`	Differenzengleichungen zum Typ `Longreal`	317
`leig`	$\boldsymbol{Ax} = \lambda \cdot \boldsymbol{x},\ \boldsymbol{A}$: `Lrmatrix`;	594
`lheig`	Eigenwerte hermitescher Matrizen, `Lcmatrix`	632
`lieig`	$\boldsymbol{Ax} = \lambda \cdot \boldsymbol{x},\ \boldsymbol{A}$: `Limatrix`;	594
`liheig`	Eigenwerte hermitescher Matrizen, `Lcimatrix`	632
`lilss`	Lin. GlgS.: $\boldsymbol{Ax} = \boldsymbol{b},\ \boldsymbol{A}$: `Limatrix`;	299
`lilu`	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU},\ \boldsymbol{A}$: `Limatrix`;	250
`limvcran`	Komplexe Zufalls-Intervall-Matrizen/Vektoren	42
`limvran`	Reelle Zufalls-Intervallmatizen/Vektoren	41
`linv`	Inversion reeller Matrizen zum Typ `Lrmatrix`	265
`liseig`	Eigenwerte symmetrischer Matrizen, `Limatrix`	625
`litaylor`	Reelle Taylor-Koeffizienten zum Typ `Linterval`	397
`llss`	Lin. GlgS.: $\boldsymbol{Ax} = \boldsymbol{b},\ \boldsymbol{A}$: `Lrmatrix`;	299
`llu`	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU},\ \boldsymbol{A}$: `Lrmatrix`;	250
`lmcif0`	Matrix bzw. Vektor der Absolutbeträge	228
`lmv_exap`	Matrizen: Hilbert, Dekker, Zielke, `Lrmatrix`	644
`lmvcf0`	Matrix/Vektor Hilfsfunktionen, `Lcomplex`	228
`lmvcf1`	Funktionen für Power, Det, Trace, `Lcmatrix`	236
`lmvf0`	Matrix/Vektor Hilfsfunktionen, `Longreal`	227
`lmvf1`	Funktionen für Power, Det, Trace, `Lrmatrix`	240
`lmvif0`	Matrix bzw. Vektor der Absolutbeträge	227
`lmvif1`	Funktionen Power, Det zum Typ `Limatrix`	240

Tabelle der Hilfsmodule der BCD-Version Teil 5		
Modul	**Anwendungen**	**Seite**
lo_int	Integer–Langzahlarithmetik	49
lqr	$\boldsymbol{A} = \boldsymbol{Q} \cdot \boldsymbol{R}$, A,Q,R : Lrmatrix	256
lqr_c	$\boldsymbol{A} = \boldsymbol{Q} \cdot \boldsymbol{R}$, A,Q,R : Lcmatrix	256
ls_fu1	Standardfunktionen zum Typ LStagg	106
lsclu	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: LStaggcmatrix;	250
lscswap	Matrix/Vektor-Tauschoperationen, LStaggcomplex	222
lseig	Eigenwerte symmetrischer Matrizen, Lrmatrix	625
lsgmf1	Funktionen Power, Det zum Typ LStaggrmatrix	240
lsmcf1	Funktionen Power, Det zum Typ LStaggcmatrix	240
lsmvc	Komplexe Staggered Arithmetik, Matrizen/Vektoren	211
lss	Lin. GlgS.: $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{A}$: rmatrix;	299
lst1_ari	Listenverwaltung, (Intervall,Zahl)	646
lst_ari	Listenverwaltung, (Vektor,Zahl)	647
lstagg	Intervall-Staggered-Arithmetik zum Typ LStagg	91
lstglu	Staggered LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: Lrmatrix	250
lstgmv	Staggered Arithmetik f. Matrizen/Vektoren, LStagg	190
lstgswap	Matrix/Vektor-Tauschoperationen zum Typ LStagg	222
lstqr	$\boldsymbol{A} = \boldsymbol{Q} \cdot \boldsymbol{R}$, A,Q,R : LStaggrmatrix	257
lswap	Matrix/Vektor-Tauschoperationen, Longreal	222
lu	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: rmatrix;	250
mcif0	Matrix bzw. Vektor der Absolutbeträge	228
mcif1	Power-Funktion zum Typ cimatrix	246
mv_comp	Maximalgenaue, reelle Matrix/Vektor-Ausdrücke	169

Tabelle der Hilfsmodule der BCD-Version Teil 6		
Modul	**Anwendungen**	**Seite**
mv_exap	Matrizen: Hilbert, Dekker, Zielke; Typ: rmatrix	645
mv_util	Ergänzungen zu den Modulen: mvi_ari, lmvi_ari	649
mvc_comp	Maximalgenaue, kompl. Matrix/Vektor-Ausdrücke	176
mvcf0	Matrix/Vektor Hilfsfunktionen, complex	228
mvcf1	Funktionen für Power, Det, Trace, cmatrix	236
mvcran	Komplexe Zufallsmatrizen/Vektoren	41
mvf0	Matrix/Vektor Hilfsfunktionen zum Typ real	227
mvf1	Funktionen für Power, Det, Trace, rmatrix	240
mvif0	Matrix bzw. Vektor der Absolutbeträge	227
mvif1	Funktionen Power, Det zum Typ imatrix	240
mvran	Reelle Zufallsmatrizen/Vektoren	40
nlfz3	Kurvendiskussion mit Function_Analyser	457
nlfzero	Einfache Nullstellen nichtlinearer Gleichungen	439
nlss	Nullstellen nichtlinearer Gleichungssysteme	479
qr	$\boldsymbol{A} = \boldsymbol{Q} \cdot \boldsymbol{R}$, A,Q,R : rmatrix	256
qr_c	$\boldsymbol{A} = \boldsymbol{Q} \cdot \boldsymbol{R}$, A,Q,R : cmatrix	256
rand0	Reelle Zufallszahlen	38
random	Reelle Zufallszahlen	38
rlpeval	Einschießung reeller Polynomwerte, Longreal	333
rpeval	Einschießung reeller Polynomwerte, real	333
rrgi_util	Ergänzungen zum Systemmodul rrgi_ari	651
s_fu1	Standardfunktionen zum Typ Stagg	78
sclu	LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: Staggcmatrix;	250

Tabelle der Hilfsmodule der BCD-Version Teil 7		
Modul	**Anwendungen**	**Seite**
`scpoly`	Routinen zum Typ `SCPolynomial`	643
`scpzero`	Nullstellen komplexer Polynome, `SCPolynomial`	530
`scswap`	Matrix/Vektor-Tauschoperationen, `Staggcomplex`	222
`seig`	Eigenwerte symmetrischer Matrizen, `rmatrix`	625
`sgmf1`	Funktionen für Power, Det, `Staggrmatrix`	240
`smcf1`	Funktionen für Power, Det, `Staggcmatrix`	240
`smvc`	Staggered Arithmetik f. Matrizen/Vektoren, `Stagg`	200
`stagg`	Intervall-Staggered-Arithmetik zum Typ `Stagg`	65
`stglu`	Staggered LU-Zerlegung: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{A}$: `rmatrix`	250
`stgmv`	Staggered Arithmetik f. Matrizen/Vektoren, `Stagg`	180
`stgswap`	Matrix/Vektor-Tauschoperationen zum Typ `Stagg`	222
`stqr`	$\boldsymbol{A} = \boldsymbol{Q} \cdot \boldsymbol{R}$, $\boldsymbol{A}, \boldsymbol{Q}, \boldsymbol{R}$: `Staggrmatrix`	257
`swap`	Matrix/Vektor-Tauschoperationen zum Typ `real`	222
`type_def`	Datentypen, Variable, Präzisionsänderung	637
`vv_comp`	Maximalgenaue, reelle Vektoroperationen	175
`xi_ari`	Extended interval arithmetic zum Typ `Linterval`	655

Index

Literaturverzeichnis

[1] G. Alefeld and J. Herzberger. *Einführung in die Intervallrechnung.* Reihe Informatik 12. **BI** Bibliographisches Institut Mannheim Wien Zürich, 1974. ISBN 3-411-01466-0.

[2] Howard Anton. *Lineare Algebra – Einführung, Grundlagen, Übungen.* Spektrum Akademischer Verlag Heidelberg Berlin, 1998. ISBN 3-8274-0324-3.

[3] Frithjof Blomquist. Inverse Matrizen. Inst. f. Angewandte Mathematik, Universität Karlsruhe, 1999. www.math.uni-wuppertal.de/org/WRST/blom.html

[4] G. Engeln-Müllges and F. Reutter. *Formelsammlung zur Numerischen Mathematik mit Turbo-Pascal-Programmen.* **BI** Wissenschaftsverlag Mannheim Wien Zürich, 1987. ISBN 3-411-03156-5.

[5] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *Numerical Toolbox for Verified Computing, Basic Numerical Problems*, volume 1. Springer-Verlag Berlin Heidelberg, 1993. Springer Series in Computational Mathematics **21**
www.uni-karlsruhe.de/~iam/html/literatur/download.html

[6] Hämmerlin and Hoffmann. *Numerische Mathematik.* Springer-Verlag, Berlin Heidelberg New York, fourth edition, 1994. ISBN 3-540-58033-6.

[7] Kiessling, Lowes, and Paulik. *Genaue Rechnerarithmetik – Intervallrechnung und Programmieren mit PASCAL-XSC.* Teubner-Studienskripten; 114: Mathematik, Informatik. B.G. Teubner, Stuttgart, 1988. ISBN 3-519-00114-4.

[8] W. Krämer, U. Kulisch, and R. Lohner. Numerical Toolbox for Verified Computing II. Veröffentlichung geplant im Springer Verlag.

[9] Walter Krämer. Inverse Standardfunktionen für reelle und komplexe Intervallargumente mit a priori Fehlerabschätzungen für beliebige Datenformate. Dissertation, Universität Karlsruhe, 1987.

[10] Ulrich Kulisch. *Wissenschaftliches Rechnen mit Ergebnisverifikation - Eine Einführung -*. Friedr. Vieweg & Sohn, Braunschweig/Wiesbaden, 1989. ISBN 3-528-08943-1.

[11] Rudolf Lohner. *Verified Computing and Programs in PASCAL-XSC*. Habilitationsschrift, Institut für Angewandte Mathematik, Universität Karlsruhe (TH), 1994.

[12] PASCAL-XSC. *Sprachbeschreibung mit Beispielen*. R.Klatte, U.Kulisch, M.Neaga, D.Ratz and Ch.Ullrich, Inst. f. Angewandte Mathematik, Universität Karlsruhe, 1990.
www.uni-karlsruhe.de/~iam/html/literatur/download.html

[13] PASCAL-XSC. *A PASCAL Extension for Scientific Computation and Numerical Data Processing, User's Guide*. Numeric Software GmbH, P.O. Box 2232, D-7570 Baden-Baden, Germany, 1991.

[14] PASCAL-XSC. *Sprachbeschreibung mit Beispielen*. R.Klatte, U.Kulisch, M.Neaga, D.Ratz and Ch.Ullrich, Springer-Verlag Heidelberg, 1991.
www.uni-karlsruhe.de/~iam/html/literatur/download.html

[15] PASCAL-XSC. *Language Reference with Examples*. R.Klatte, U.Kulisch, M.Neaga, D.Ratz and Ch.Ullrich, Springer-Verlag New York, 1992. www.uni-karlsruhe.de/~iam/html/literatur/download.html

[16] PASCAL-XSC. *BCD-Version 1.0, Benutzerhandbuch für das dezimale Laufzeitsystem*. Frithjof Blomquist, Inst. f. Angewandte Mathematik, Universität Karlsruhe, 1997.
Forschungsschwerpunkt **C**omputerarithmetic, **I**ntervallrechnung und **N**umerische Algorithmen mit **E**rgebnisverifikation
www.uni-karlsruhe.de/ ~iam/html/literatur/download.html

[17] PASCAL-XSC. *A Survey of PASCAL-XSC and a Language Reference Supplement on Dynamic and Flexible Arrays*. P. Januschke, D. Ratz, Institut für Angewandte Mathematik, Universität Karlsruhe, Kaiserstr. 12, D-76128 Karlsruhe, Bericht 1 aus 1998.
Forschungsschwerpunkt **C**omputerarithmetic, **I**ntervallrechnung und **N**umerische Algorithmen mit **E**rgebnisverifikation
www.uni-karlsruhe.de/~iam/html/literatur/download.html

[18] S.M. Rump. Kleine Fehlerschranken bei Matrixproblemen. Dissertation, Universität Karlsruhe, 1980.

[19] H.R. Schwarz. *Numerische Mathematik*. B.G. Teubner, Stuttgart, 1997. ISBN 3-519-32960-3.

[20] Stoer. *Numerische Mathematik 1*. Springer-Verlag, Berlin Heidelberg New York, eighth edition, 1999. ISBN 3-540-66154-9.

[21] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, Berlin Heidelberg New York, second edition, 1993. ISBN 3-540-97878-X.

[22] Rudolf Zurmühl and Sigurd Falk. *Matrizen und ihre Anwendungen - für Angewandte Mathematiker, Physiker und Ingenieure*, volume 2 Numerische Methoden. Springer-Verlag, Berlin Heidelberg New York Tokyo, 1986. ISBN 3-540-15474-4.