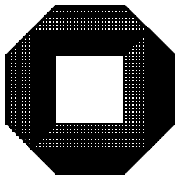


UNIVERSITÄT KARLSRUHE

Intervallrechnung in Maple – Die Erweiterung intpakX zum Paket intpak der Share-Library

I. Geulig und W. Krämer

Preprint Nr. 99/2

Institut für Wissenschaftliches Rechnen
und Mathematische Modellbildung

76128 Karlsruhe

Anschrift der Verfasser:

Ilse Geulig

Walter Krämer
Institut für Wissenschaftliches Rechnen und
Mathematische Modellbildung (IWRMM)

Universität Karlsruhe
Postfach 6980
76128 Karlsruhe

Bundesrepublik Deutschland

Das Postscript-File dieses Preprints ist über FTP unter der Adresse

`iamk4515.mathematik.uni-karlsruhe.de`

im Verzeichnis

`/pub/iwrmm/preprints`

abrufbar.

Inhaltsverzeichnis

1 Zusammenfassung

Das Paket `intpak` der Share-Library von Maple stellt ein erstes experimentelles Intervallarithmetikpaket dem Benutzer zur Verfügung. Es umfaßt unter anderem den neuen Datentyp `interval` (Langzahlintervalle), die zugehörigen arithmetischen Grundoperationen, Verbandsoperationen wie Durchschnitt und Vereinigung, einige elementare (Langzahl-)Intervallfunktionen sowie ein Kommando, welches gegebene Ausdrücke automatisch in Intervallausdrücke umformt.

Die Erweiterung `intpakX` ergänzt das Paket `intpak` in wesentlichen Punkten. Z. B. wird die sogenannte erweiterte Intervalldivision zur Verfügung gestellt, es findet sich die Realisierung eines Intervallnewtonverfahrens, eine komplexe Kreisscheibenarithmetik, eine Erweiterung der Exponentialfunktion auf Kreisscheibenintervalle sowie die Realisierung verschiedener Algorithmen zur Einschließung des Wertebereiches eines komplexen Polynoms (mit zentrierter Multiplikation, mit flächenoptimaler Multiplikation). Darüber hinaus zielt die Erweiterung durch die Bereitstellung geeigneter Prozeduren auf die graphische Veranschaulichung von Verifikationsalgorithmen (z. B. Nullstellensuche, lineare, quadratische Wertebereichseinschließungen, Berechnungen mit Kreisscheibenintervallen). Die Bildschirmausgabe der Graphikroutinen ist (im Gegensatz zu den in diesem Bericht wiedergegebenen Abbildungen) farbig.

Die aktuellen Quellen (ca. 2000 Zeilen Maple-Code) der Erweiterung `intpakX` sind frei verfügbar und können via ftp vom Server

`iamk4515.mathematik.uni-karlsruhe.de`

im Verzeichnis `/pub/iwrmm/maple/software` abgerufen werden.

Key Words: MapleV, Interval Analysis, Validated Computations, Disk Arithmetic, Visualization of Self Validating Numerical Algorithms

MSC: 65G05, 65G10, 68M15

2 Zum Programmpaket `intpak`

2.1 Funktionsumfang

Das in der Share-Library von Maple enthaltene Intervallpaket `intpak` umfaßt

- den neuen Datentyp `interval` (Langzahlintervalle),
- die zugehörigen arithmetischen Grundoperationen,
- elementare Intervallfunktionen und
- einige Hilfsfunktionen.

Der Datentyp `interval`

Eine Variable x ist vom Typ `interval`, falls x eine leere Liste ist, oder falls $x = [x_1, x_2]$ eine Liste mit zwei Elementen ist. Die beiden Intervallgrenzen $x_i, i = 1, 2$ müssen eine der folgenden Bedingungen erfüllen:

- x_i ist eine reelle Zahl vom Typ `float` oder x_i ist gleich 0.
- $x_i \in \{-$`infinity`, `infinity`$\}$.
- x_i ist eine in Maple vordefinierte Konstante (dazu gehören `Pi, gamma, Catalan, FAIL, false, true und infinity`).

Bemerkung: Zahlen vom Typ `integer` oder vom Typ `fraction` sind als Intervallgrenzen nicht zugelassen!

Arithmetische Grundoperationen

Die implementierten Grundoperationen sind &+, &−, &∗, &/ und `inv`. Ihre Eingabeparameter müssen entweder vom Typ `interval` oder vom Typ `num_or_FAIL` sein. Ausgegeben wird ein Intervall.

Eine Variable ist vom Typ `num_or_FAIL`, falls ihr Wert eine Zahl (d. h. die Variable ist vom Typ `numeric`), −`infinity` oder eine in Maple vordefinierte Konstante (siehe oben) ist.

Bemerkung: Die Prioritäten für die sogenannten 'inerten' Operatoren &+, &−, &∗, &/ sind in Maple ärgerlicherweise so festgelegt, daß &+ und &− höhere Priorität haben als &∗ und &/. Bei Ausdrücken muß deshalb häufig geklammert werden!

Elementare Intervallfunktionen

Die implementierten Intervallfunktionen sind &sqr, &sqrt, &ln, &exp, &**, &intpower, &sin, &cos, &tan, &arcsin, &arccos, &arctan, &sinh, &cosh und &tanh. Es wird davon ausgegangen, daß die entsprechenden mathematischen Funktionen in Maple maximal um 0.6 ulp von den exakten Ergebnissen abweichen. Diese Annahme wird im Maple-Handbuch jedoch nirgends bestätigt!

Die meisten Bezeichnungen sind selbsterklärend. Die Operation &intpower entspricht der Funktion x^n, wobei n eine natürliche Zahl ist. Der Operator &** entspricht der Funktion x^α, wobei α ein Intervall, eine ganze Zahl oder eine Zahl vom Typ `float` bedeuten kann.

Hilfsfunktionen

Die Funktion `construct` erzeugt aus einer Zahl oder einem Zahlenpaar ein Element vom Typ `interval`. Als optionaler Parameter kann der String 'rounded' angegeben werden. In diesem Fall werden die Intervallgrenzen um ein ulp nach unten bzw. nach oben gerundet. Dies wird mit Hilfe der Funktionen `ru` (`Interval_Round_Up`) und `rd` (`Interval_Round_Down`) realisiert. Zu beachten ist, daß die Eingabeparameter von `ru` und `rd` vom Typ `float` sein müssen.

Ebenfalls vorhanden sind die Funktionen `midpoint`, `width`, &intersect, &union und `is_in`. Die Funktion `width` berechnet den Durchmesser und `midpoint` eine Einschließung für den Mittelpunkt eines Intervalls.

Für die Umformung eines Ausdruck in einen Intervallausdruck bzw. in eine Intervallfunktion stehen die Kommandos '`convert/interval`' und `inapply` zur Verfügung.

2.2 Erste Berechnungsbeispiele

Das Intervallpaket `intpak` kann folgendermaßen eingelesen werden:

```
> with(share):
See ?share and ?share,contents for information about the share library
> with(intpak);
```

$$[init]$$

Man beachte, daß in Maple Groß- und Kleinschreibung signifikant ist.

Beispiel 1: Der Datentyp `interval`
Intervallgrenzen vom Typ integer sind *nicht* zugelassen:

```
> x:=[1,2];
```
$$x := [1, 2]$$
```
> type(x,interval);
```
$$false$$
```
> x:=construct(1,2); type(x,interval);
```
$$x := [1., 2.]$$
$$true$$

Einschliessendes Intervall generieren:

```
> construct(1,rounded);
```
$$[.999999999, 1.000000001]$$
```
> y:=construct(1,infinity,rounded);
```
$$y := [.999999999, \infty]$$
```
> type(y,interval);
```
$$true$$

Durchmesser und Einschließung des Mittelpunktes eines Intervalls:

```
> width(x); width(y);
```
$$1.$$
$$\infty$$
```
> midpoint(x);
```
$$[1.499999999, 1.500000001]$$

Beispiel 2: Mengenoperationen

```
> x:=[1.,3.]; y:=[2.,infinity]; z:=[4.,5.];
```
$$x := [1., 3.]$$
$$y := [2., \infty]$$
$$z := [4., 5.]$$
```
> x &union y;
```
$$[1., \infty]$$
```
> x &union z;
```
$$[1., 3.], [4., 5.]$$

Die Ergebnismenge besteht aus zwei Intervallen. &union berechnet also **nicht** die Intervallhülle. Dazu steht in intpakX die Prozedur &Convex_Hull zur Verfügung.

```
> x &intersect y;
```
$$[2., 3.]$$
```
> x[1];
```
$$1.$$
```
> is_in(x[1],y);
```
$$false$$
```
> is_in(z,y);
```
$$true$$

Beispiel 3: Arithmetische Operationen und Intervallfunktionen

```
> [1.,2.] &+ 3 &* 0; # Falsche Prioritaeten der Operatoren
```
$$[0, 0]$$
```
> [1.,2.] &+ (3 &* 0);
```
$$[.999999999, 2.000000001]$$
```
> [-1.,2.] &intpower 3;
```
$$[-1.000000001, 8.000000001]$$
```
> &sqr(&cosh(1)) &- &sqr(&sinh(1));
```
$$[.9999999919, 1.000000008]$$

Beispiel 4: Wertebereicheinschließungen
Einschließung des Wertebereichs von $f(x) := x^3 - x^2 - x + 1$ über dem Intervall $[0, 0.5]$ durch intervallmäßige Auswertung, mit Hilfe der Mittelwertform und unter Beachtung der Monotonieeigenschaften von f.

```
> x:='x'; # Variable freigeben
```
$$x := x$$
```
> f:=x^3-x^2-x+1;
```
$$f := x^3 - x^2 - x + 1$$
```
> F:=inapply(f,x); # Umwandlung von f in eine Intervallfunktion
```
$$F := x \rightarrow (x\ `\&\mathrm{intpower}`\ 3)\ `\&+`\ (((-1)\ `\&*`\ (x\ `\&\mathrm{intpower}`\ 2))\ `\&+`\ (((-1)\ `\&*`\ x)\ `\&+`\ 1))$$

Umwandlung der Ableitung f' von f in eine Intervallfunktion

```
> dF:=inapply(diff(f,x),x);
```
$$dF := x \rightarrow (3\ `\&*`\ (x\ `\&\mathrm{intpower}`\ 2))\ `\&+`\ (((-2)\ `\&*`\ x)\ `\&+`\ (-1))$$

Einschliessung des Wertebereichs von f über dem Intervall [0,0.5]

```
> X:=[0,0.5];
```

$$X := [0,\ .5]$$

```
> mid_X:=midpoint(X);
```

$$mid\_X := [.2499999999,\ .2500000002]$$

```
> r_i:=F(X); # intervallmaessige Auswertung
```

$$r\_i := [.2499999994,\ 1.125000003]$$

```
> r_m:=F(mid_X) &- ( dF(X) &* (X &- mid_X) ); # Mittelwertform
```

$$r\_m := [.2031249977,\ 1.203125003]$$

Die intervallmäßige Auswertung von f' über dem Intervall $X = [0, 0.5]$ zeigt, daß f monoton fallend ist in X.

```
> dF([0,0.5]);
```

$$[-2.000000003,\ -.2499999985]$$

Der exakte Wertebereich von f über X ist das Intervall $[0.375, 1]$. Eine sehr enge Wertebereicheinschließung kann man folgendermaßen berechnen:

```
> r_e:=construct(F(X[2])[1],F(X[1])[2]);
```

$$r\_e := [.3749999993,\ 1.000000003]$$

`F(X[2])` berechnet die intervallmäßige Auswertung von f an der Stelle `X[2]`$= 0.5$. `F(X[2])[1]` liefert die linke Intervallgrenze von `F(X[2])`, also eine **gesicherte** untere Schranke für das Minimum von f über dem Intervall $[0, 0.5]$.

Um nachzuprüfen, ob die 'exakte' Wertebereicheinschließung `r_e` im Durchschnitt der oben berechneten Einschließungen `r_i` und `r_m` enthalten ist, kann man z. B. die Prozedur `is_in` verwenden.

```
> is_in(r_e, r_i &intersect r_m);
```

true

2.3 Notwendige Korrekturen und Änderungen

In diesem Abschnitt werden einige Fehler in der Implementierung der `intpak`-Prozedur '`convert/interval`' angesprochen, die durch die Erweiterung `intpakX` behoben worden sind. Außerdem wird auf einige im Hinblick auf `intpakX` vorgenommene Änderungen in der Definition des Typs `interval_comp` und der Implementierung der Prozeduren `is_in`, `Interval_power` und `construct` hingewiesen.

Die Prozeduren `convert/interval` und `inapply`

Das Kommando `inapply` soll(!) einen Ausdruck in eine Intervallfunktion umformen und ist daher für die Erstellung von Prozeduren, die auf das Package `intpak` aufbauen, sehr hilfreich (siehe auch Beispiel 4, Abschnitt 2.2). Die Transformation erfolgt in zwei Schritten. Zuerst wird der eingegebene Ausdruck mit Hilfe des Kommandos `convert/interval` in einen Intervallausdruck umgeformt. Anschließend wird mit dem Maple-Kommando `unapply` aus dem Intervallausdruck eine Intervallfunktion erzeugt.

Leider haben sich in die Implementierung des Kommandos `convert/interval` einige Fehler eingeschlichen, die sich beim Aufruf von `inapply` bemerkbar machen:

1. Der Aufruf

```
> inapply(0.5*t,t);
```

```
Error, (in type/interval\_comp) too many levels of recursion
```

führt zu einer Fehlermeldung. Die Prozedur `convert/interval` gelangt nämlich in eine Endlosschleife, wenn in einem Ausdruck ein Produkt auftritt, bei dem einer der Faktoren vom Typ `float` ist.

2. Umwandlung von $f(t) := \sqrt{t}$ in eine Intervallfunktion:

```
> f:=inapply(sqrt(t),t);
```

$$f := t \rightarrow t \text{ `\&\^{}`} \frac{1}{2}$$

Eigentlich hätte man erwartet, daß f :=&$sqrt$ zurückgegeben wird. Aber Maple wandelt den Ausdruck $\sqrt{t}$ in $t^{1/2}$ um, noch ehe die Prozedur `inapply` zum Zuge kommt. Wenn man nun z. B. $f([4.,9.])$ berechnen möchte, so wird der Ausdruck unausgewertet zurückgegeben, da der Operator &\^{} in `intpak` nicht definiert ist:

```
> f([4.,9.]);
```

$$[4.,\ 9.] \,\&\hat{}\, \frac{1}{2}$$

Die korrekte Operatornotation für die in `intpak` definierte Prozedur `Interval_power` wäre &**. Allerdings liefert der Aufruf

```
> [4.,9.] &** (1/2);
```

$$[4.,\ 9.] \,\&**\, \frac{1}{2}$$

ebenfalls nicht das gewünschte Ergebnis, da die Prozedur `Interval_power` als zweites Argument keine rationalen Zahlen zuläßt.

```
> [4.,9.] &** 0.5;
```

$$[1.999999999,\ 3.000000001]$$

liefert endlich das gewünschte Ergebnis.

3. Bei der Übertragung des Intervallpakets von Release 4 nach Release 5 hat sich zusätzlich noch ein Fehler eingeschlichen. Die globale Variable `Interval_fnlist`, die für die Transformation der Standardfunktionen in die entsprechenden Intervallfunktionen verantwortlich ist, ist dabei verlorengegangen.

```
> inapply(sin(x),x);

Error, (in convert/interval) wrong number (or type) of parameters in
function subs
```

Um mit `inapply` auch Ausdrücke, die Standardfunktionen enthalten, verarbeiten zu können, muß man daher die Erweiterung `intpakX` verwenden. Die Variable `Interval_fnlist` wird dabei durch das Laden von `intpakX` automatisch initialisiert.

Mit der im Package `intpakX` enthaltenen verbesserten Version der Prozedur ‘`convert/interval`‘ erhält man z. B.:

```
> f:=inapply(0.5*t,t);
```

$$f := t \rightarrow .5\ \text{`\&}*\text{`}\ t$$

```
> f(2);
```

$$[.999999999,\ 1.000000001]$$

```
> f:=inapply(sqrt(t),t);
```

$$f := \&sqrt$$

```
> f([4.,9.]);
```

$$[1.999999999,\ 3.000000001]$$

```
> f:=inapply(sin(x)+x,x);
```

$$f := x \rightarrow \text{`\&sin`}(x)\ \text{`\&}+\text{`}\ x$$

```
> f(0);
```

$$[0,\ 0]$$

Die Prozedur Interval_power

Um die Auswertung von z. B. der zweiten Ableitung von $f(x) := \sqrt{x}$

$$f''(x) = -\frac{1}{4}x^{-\frac{3}{2}}$$

intervallmäßig zu ermöglichen, wurde die Prozedur `Interval_power` (alias &**) so ergänzt, daß als zweiter Parameter auch rationale Zahlen zulässig sind.

Berechnung von $f''(4)$ mit Gleitkommaarithmetik

```
> evalf(-1/4*4^(-3/2));
```

$$-.03125000000$$

Intervallmäßige Auswertung von $f''(4)$

```
> df2:=inapply(diff(sqrt(x),x$2),x);
```

$$df2 := x \rightarrow (\frac{-1}{4}) \text{`\&*`} (x \text{`\&**`} (\frac{-3}{2}))$$

```
> df2(4);
```

$$[-.03125000004,\ -.03124999997]$$

Die Datentypen interval_comp und interval

Laut Definition des Typs `interval_comp` sind die in Maple vordefinierten Konstanten `Pi, gamma, Catalan, false` und `true` in `intpak` als Intervallgrenzen zugelassen. Allerdings wird das bei der Implementierung des Typs `interval` sowie bei der Implementierung der Grundoperationen und der elementaren Intervallfunktionen nicht berücksichtigt und führt daher zu unerwarteten Ergebnissen oder Fehlermeldungen. Beispiele:

```
> type([1.,Pi],interval);

Error, (in intpak/max) cannot evaluate boolean

> &sinh([-infinity,Pi]);

Error, (in Interval\_ulp) improper op or subscript selector

> 1. &+ [Pi,infinity];
```

$$[1. + \pi - \mathrm{Float}(1,\ -8 + \pi),\ \infty]$$

Im letzten Fall erfolgt keine Fehlermeldung, aber das Ergebnis ist nicht vom Typ `interval`. Ein ähnliches Verhalten tritt auch bei den anderen oben genannten Konstanten auf. Daher enthält das Intervallpaket `intpakX` eine geänderte Version des Datentyps `interval_comp`, bei der die Maple-Konstanten `Pi, gamma, Catalan, false` und `true` als Intervallkomponenten (d. h. als Intervallgrenzen) nicht mehr zugelassen sind.

Dies bedeutet keine wesentliche Einschränkung, da bei der Durchführung von Intervalloperationen die Konstanten `Pi, gamma` und `Catalan` sowieso zuerst in Zahlen vom Typ `float` umgewandelt werden. Beispiel:

```
> &sin(Pi);
```

$$[-.979323846265\,10^{-11}, .102067615375\,10^{-10}]$$

Die Prozedur `is_in`

Die `intpak`-Prozedur `is_in` hat zwei Eingabeparameter und prüft nach, ob der erste Parameter in dem zweiten Parameter enthalten ist (im Sinne der Mengentheorie). Als Eingabeparameter sind Variablen vom Typ `interval`, Zahlen vom Typ `numeric` und die Werte `FAIL`, `infinity` und `-infinity` zugelassen. Insbesondere sind also auch Zahlen vom Typ `rational`, welche bisher auf **falsche** Ergebnisse führten, erlaubt. Beispiel

```
> Digits; # Rechengenauigkeit
```

$$10$$

```
> is_in(1/3,[0.3333333332,0.3333333333]);
```

$$true$$

Das Ergebnis ist offensichtlich **falsch**, denn

$$1/3 \notin [0.3333333332, 0.3333333333].$$

Zu fehlerhaften Ergebnissen kommt es natürlich auch, falls als Eingabeparameter Zahlen vom Typ `float` verwendet werden, deren Länge den aktuellen Wert der Variablen `Digits` überschreitet.

```
> is_in(1.9999999999,[2.,2.]);
```

$$true$$

Auch das Maple-Kommando `evalb` liefert in beiden Fällen **falsche** Ergebnisse, da in dem logischen Ausdruck Gleitkommazahlen auftreten.

```
> evalb((0.3333333332 <= 1/3) and (1/3 <= 0.3333333333));
```

$$true$$

```
> evalb(2. <= 1.9999999999);
```

$$true$$

Sind alle Eingabeparameter jedoch rationale Zahlen, so ist das Ergebnis bei Verwendung von `evalb` korrekt, während `is_in` auch in diesem Fall ein **falsches** Ergebnis liefert.

```
> is_in(1/3,3333333333/10^10);
```

$$true$$

```
> evalb(1/3 = 3333333333/10^10);
```

$$false$$

Um also die Prozedur `is_in` so zu verbessern, daß sie auch bei Verwendung von rationalen oder 'zu langen' Gleitkommazahlen ein korrektes Ergebnis liefert, muß die exakte rationale Langzahlarithmetik von Maple verwendet werden.

Um dies zu erreichen muß man eine Zahl vom Typ `float` ohne 'Konversionsfehler' in eine Zahl vom Typ `rational` umwandeln können. Das Maple-Kommando `convert/rational` kann dazu leider nicht verwendet werden.

```
> convert(0.3333333333,rational);
```

$$\frac{1}{3}$$

Mit Hilfe des Maple-Kommandos `op` scheint es jedoch möglich, fehlerfreie Transformationen durchführen zu können.

```
> x:=0.3333333333;
```

$$x := .3333333333$$

Umwandlung von `x` in eine rationale Zahl:

```
> x_rational:=op(1,x) * 10^op(2,x);
```

$$x\_rational := \frac{3333333333}{10000000000}$$

Ist `x = x_rational` ?

```
> evalb((x <= x_rational) and (x_rational <= x));
```

$$true$$

Ist `x = 1/3` ?

```
> evalb((x <= (1/3)) and ((1/3) <= x));
```

$$true$$

Ist `x_rational = 1/3` ?

```
> evalb((x_rational <= (1/3)) and ((1/3) <= x_rational));
```

false

Die Erweiterung `intpakX` enthält das Kommando '`intpakX/greater`', das mit Hilfe der oben gezeigten Transformation und der Langzahlarithmetik von Maple nachprüft, ob der erste Eingabeparameter größer gleich dem zweiten Eingabeparameter ist. Unter Verwendung dieses Kommandos wurde die Prozedur `is_in` so abgeändert, daß sie auch dann ein korrektes Ergebnis liefert, wenn eine der Zahlen rational ist oder wenn die Länge der eingegebenen Zahlen den Wert der Variablen Digits überschreitet. Beispiel:

```
> Digits;
```

10

```
> is_in(1.9999999999999999,[2.,2.]);
```

false

```
> is_in(1/3,[0.3333333332,0.33333333333333333]);
```

false

Die Prozedur `construct`

Als Parameter einer `intpak`-Intervalloperation sind auch Zahlen vom Typ `numeric` sowie Maple-Konstanten zugelassen. Vor der Durchführung der entsprechenden Operation werden diese jedoch mit Hilfe der Prozedur `construct` in Intervalle umgewandelt. Allerdings sind diese Intervalle nicht immer eine Einschließung des tatsächlichen Wertes der eingegebenen Zahl, was zu fehlerhaften Ergebnissen führt. Beispiele:

```
> (1/3) &- 0.3333333333;
```

$[0, 0]$

```
> 1.0000000001 &- 1.;
```

$[0, 0]$

In beiden Fällen ist $[0,0]$ offensichtlich **keine** Einschließung des exakten Ergebnisses. Die Ursache des Fehlers liegt in der Prozedur `construct`. Sie erzeugt bei Eingabe einer rationalen Zahl oder einer 'zu langen' Zahl, ohne zusätzliche Angabe des optionalen Parameters `rounded`, **kein** einschließendes Intervall.

```
> construct(1/3);
```

[.3333333333, .3333333333]

```
> construct(1.0000000001);
```

[1.000000000, 1.000000000]

Ähnlich zum Vorgehen bei der Prozedur `is_in` kann auch in diesem Fall, unter Verwendung der exakten rationalen Arithmetik von Maple, die Prozedur korrigiert werden. Mit der verbesserten `intpakX`-Version von `construct` erhält man dann

```
> construct(1/3);
```

$$[.3333333333,\ .3333333334]$$

```
> (1/3) &- 0.3333333333;
```

$$[0,\ .1000000001\,10^{-9}]$$

```
> construct(1.0000000001);
```

$$[1.000000000,\ 1.000000001]$$

```
> 1.0000000001 &- 1;
```

$$[0,\ .1000000001\,10^{-8}]$$

3 Die Erweiterung intpakX

3.1 Die Installation von intpakX

Die Kommandos der Erweiterung `intpakX` sind in der Datei `intpakX.m` gespeichert. Sie kann via ftp vom Server `iamk4515.mathematik.uni-karlsruhe.de` im Verzeichnis `/pub/iwrmm/maple/software` abgerufen werden. Die Datei wurde mit Maple V Release 5 erzeugt.

Die Datei kann mit `with` eingelesen werden, falls der Pfad zum Verzeichnis in dem sie abgelegt ist in der Systemvariablen `libname` gespeichert ist. Die Variable `libname` enthält in der Regel nur den Pfad zur Maple-library. Sie wird automatisch beim Starten von Maple initialisiert.

```
> restart;
> libname;
```

"C:\\PROGRAMME\ \MAPLE V RELEASE 5/lib"

Wird die `share`-library von Maple mit `with` geladen, so wird auch der Pfad zum Verzeichnis in dem die `share`-Pakete liegen in `libname` gespeichert.

```
> with(share);

See ?share and ?share,contents for information about the share library
> libname;
```

"C:\\PROGRAMME\ \MAPLE V RELEASE 5/lib",
"C:\\PROGRAMME\ \MAPLE V RELEASE 5/share"

Wird die Datei `intpakX.m` in diesem `share`-Verzeichnis abgelegt, so kann sie genau wie alle anderen `share`-Pakete mit `with` eingelesen werden. Da `intpakX` auf das Intervallpaket `intpak` aufbaut und einige geänderte Kommandos enthält, muß das Package `intpak` vorher geladen worden sein. (Falls das Einlesen von `intpakX` beim ersten Mal nicht funktioniert, so sollte das System vorher mit `restart` nochmals neu gestartet werden.)

```
> with(intpak):

Share Library:  intpak

Authors: Connell, Amanda E. and Corless, Robert.

Description:  Interval Arithmetic Package
> with(intpakX);

Authors: Geulig, Ilse and Kraemer, Walter (supervisor),
University of Karlsruhe, IWRMM

Description: Extension of the Interval Arithmetic Package intpak
```

[*&cadd*, *&cdiv*, *&cdiv_opt*, *&cmult*, *&cmult_opt*, *&csub*, *centred_form_eval*, *cexp*, *complex_disc_plot*, *compute_all_zeros*, *compute_all_zeros_with_plot*, *compute_combined_range*, *compute_mean_value_range*, *compute_monotonic_range*, *compute_naive_interval_range*, *compute_range*, *compute_range3d*, *compute_taylor_form_range*, *ext_int_div*, *horner_eval_cent*, *horner_eval_opt*, *init*, *interval_list_plot*, *interval_list_plot3d*, *is_in*, *mid*, *rel_diam*, *subdivide_adaptive*, *subdivide_equidistant*]

Leider funktioniert diese Vorgehensweise in der Regel nur unter Windows, da unter Linux die einzelnen Benutzer im `share`-Verzeichnis keine eigenen Dateien ablegen dürfen. Die Datei muß also in einem benutzereigenen Verzeichnis abgelegt werden. Sei nun `/users/maple/software` das Verzeichnis in dem die Datei `intpakX.m` abgelegt ist. Erweitert man (in einer laufenden Maple-Session) die Systemvariable `libname` um den Pfad zu diesem Verzeichnis, so kann das Package `intpakX` wieder mit `with` eingelesen werden.

Beispiel zum Laden des Intervallpakets `intpak` und der Erweiterung `intpakX`:

```
> restart;
> with(share):

See ?share and ?share,contents for information about the share library
> with(intpak);
```

[*init*]

Erweiterung der Systemvariablen `libname` um den Pfad zu dem Verzeichnis in dem `intpakX.m` gespeichert ist.

```
> libname:=libname, "/users/maple/software"; # unter Linux
```

libname := "/usr/local/maple/lib", "/usr/local/maple/share", "/users/maple/software"

```
> libname:=libname,"C:\\users\\maple\\software": # unter Windows
> with(intpakX);
```

Bei den Prozeduren mit graphischer Ausgabe (siehe z. B. Abschnitte 4 und 5.3) werden Routinen aus dem `plots`-Package sowie dem `geometry`-Package verwendet. Deshalb werden diese beiden Pakete generell beim Einbinden von `intpakX` (ohne weiteres Zutun des Benutzers) mit eingebunden.

3.2 Funktionsumfang der Erweiterung `intpakX`

Das Paket `intpakX` stellt dem Benutzer folgende Erweiterungen zum Intervallpaket `intpak` zur Verfügung:

1. Verifizierte Nullstellenberechnung
   - die erweiterte Intervalldivision `ext_int_div`,
   - die Realisierung eines erweiterten Intervall-Newton-Verfahrens, (`compute_all_zeros, compute_all_zeros_with_plot`)
2. Komplexe Kreisscheibenarithmetik
   - den neuen Datentyp `complex_disc` (komplexes Kreisscheibenintervall),
   - die Prozedur `mid` zur Bestimmung des Mittelpunktes eines Intervalls (bzw. einer Näherung des Mittelpunktes die mit Sicherheit im eingegebenen Intervall liegt),
   - arithmetische Grundoperationen für Kreisscheibenintervalle (übliche 'zentrierte' Defintionen) (&`cadd`, &`csub`, &`cmult`, &`cdiv`)
   - flächenoptimale Multiplikation und Division zweier Kreisscheibenintervalle, (&`cmult_opt`, &`cdiv_opt`)
   - die Exponentialfunktion für Kreisscheibenintervalle, (`cexp`)
   - verschiedene Algorithmen zur gesicherten Einschließung des Wertebereichs eines komplexen Polynoms, (`horner_eval_cent, horner_eval_opt, centred_form_eval`)
3. Wertebereicheinschließungen mit graphischer Ausgabe

- Einschließung des Wertebereichs einer reellwertigen Funktion einer reellen Variablen durch sukzessive Zerlegung des Startintervalls in Teilintervalle (`compute_range`)
- Einschließung des Wertebereichs einer reellwertigen Funktion zweier reeller Variablen durch sukzessive Zerlegung des Startintervalls in Teilintervalle (`compute_range3d`).

4. Eine Reihe von Prozeduren zur graphischen Veranschaulichung der oben genannten Verfahren.

4 Wertebereiche mit graphischer Ausgabe

4.1 Funktionen einer Variablen

Zwei einfache Möglichkeiten, den Wertebereich einer Funktion $f : D \subseteq \mathbb{R} \to \mathbb{R}$ über einem Intervall $[x] \subseteq D$ einzuschließen, wurden bereits im Beispiel 4, Abschnitt 2.2, vorgestellt. Nämlich die intervallmäßige Auswertung von f (falls sie existiert) und die Mittelwertform (falls die intervallmäßige Auswertung von f' über $[x]$ existiert).

Eine verbesserte Einschließung erhält man, wenn man das Intervall $[x]$ zerlegt und über jedem Teilintervall eine Wertebereicheinschließung berechnet. Die Intervallhülle dieser Teilwertebereicheinschließungen ist dann eine Einschließung des Wertebereichs von f über $[x]$. Wird die Zerlegung iterativ fortgesetzt, so kann die Startwertebereicheinschließung sukzessive verbessert werden.

Dies realisiert die Prozedur `compute_range`. Die Prozedur fordert drei Eingabeparameter

- eine Funktion `f`,
- das Startintervall `xstart` (kann entweder als Intervall oder als Bereich (`range`) eingegeben werden)
- die Anzahl der durchzuführenden Iterationsschritte `iterationsteps`. Sie dient als Abbruchkriterium.

Die Reihenfolge der obengenannten drei Eingabeparameter ist zwingend. Zusätzlich ist die Übergabe von vier optionalen Parametern möglich (in beliebiger Reihenfolge)

- die Übergabe eines Parameters `Nx = n`, wobei `n` eine ganze Zahl größer gleich 1 bedeutet, bewirkt, daß das Startintervall schon vorab in `n` Intervalle zerlegt wird.

- der optionale Parameter `linear` (`quadratic`) bewirkt, daß das Verfahren linear (quadratisch) konvergiert. Bei Eingabe von `linear` wird nämlich zur Bestimmung der Teilwertebereicheinschließungen die 'naive' intervallmäßige Auswertung verwendet. Wird `quadratic` als Parameter übergeben, so wird zur Bestimmung der Teilwertebereicheinschließungen die Prozedur `compute_combined_range` verwendet, die intervallmäßige Auswertung, Mittelwertform und Monotonieuntersuchung kombiniert. Wird keiner dieser beiden Parameter eingegeben, so wird in den ersten drei Iterationsschritten die intervallmäßige Auswertung zur Bestimmung der Wertebereicheinschließungen verwendet und ab Schritt 4 die Prozedur `compute_combined_range`.

- der optionale Parameter `adaptive` bewirkt, daß eine adaptive Zerlegung der aktuellen Intervalliste vorgenommen wird und führt daher in der Regel zu einer Verringerung der Rechenzeit.

- der optionale Parameter `colorlist = [farbe1,farbe2,...]` bestimmt die Farben die zur graphischen Darstellung der einzelnen Iterationsschritte verwendet werden. `farbe1, farbe2`, usw. müssen dabei in Maple vordefinierte Farben sein, also z. B. `blue, red, green, magenta, coral, brown` usw. Das beeinflußt allerdings nicht die Darstellung des letzten Iterationsschrittes. Dieser wird jeweils gelb dargestellt.

Aus Übersichtlichkeitsgründen wird nur die graphische Darstellung der letzten drei Iterationsschritte und der Funktion f ausgegeben. Die graphische Darstellung aller Iterationsschritte wird jedoch in der globalen Variablen `q` gespeichert. Die Variable `q` ist eine Tabelle. Wurden 3 Iterationsschritte durchgeführt, so enthalten die Einträge `q[1], q[2]` und `q[3]` die Darstellung der einzelnen Iterationsschritte, allerdings ohne den Graphen der Funktion. Dieser ist im Tabelleneintrag `q[4]` gespeichert.

Auch die berechneten Wertebereicheinschließungen werden gespeichert und zwar in der globalen Variablen `r`. Sie ist ebenfalls eine Tabelle und `r[i]` enthält die im `i`-ten Schritt berechnete Wertebereicheinschließung.

Die aktuelle Zerlegung des Startintervalls wird in der globalen Variablen `list_of_intervals` gespeichert. Die entsprechenden Teilwertebereicheinschließungen sind in der globalen Variablen `list_of_ranges` abgelegt.

Beispiele:

Einschließung des Wertebereichs der Funktion

```
> f:=x->exp(-x^2)*sin(Pi*x^3);
```

$$f := x \rightarrow e^{(-x^2)} \sin(\pi\, x^3)$$

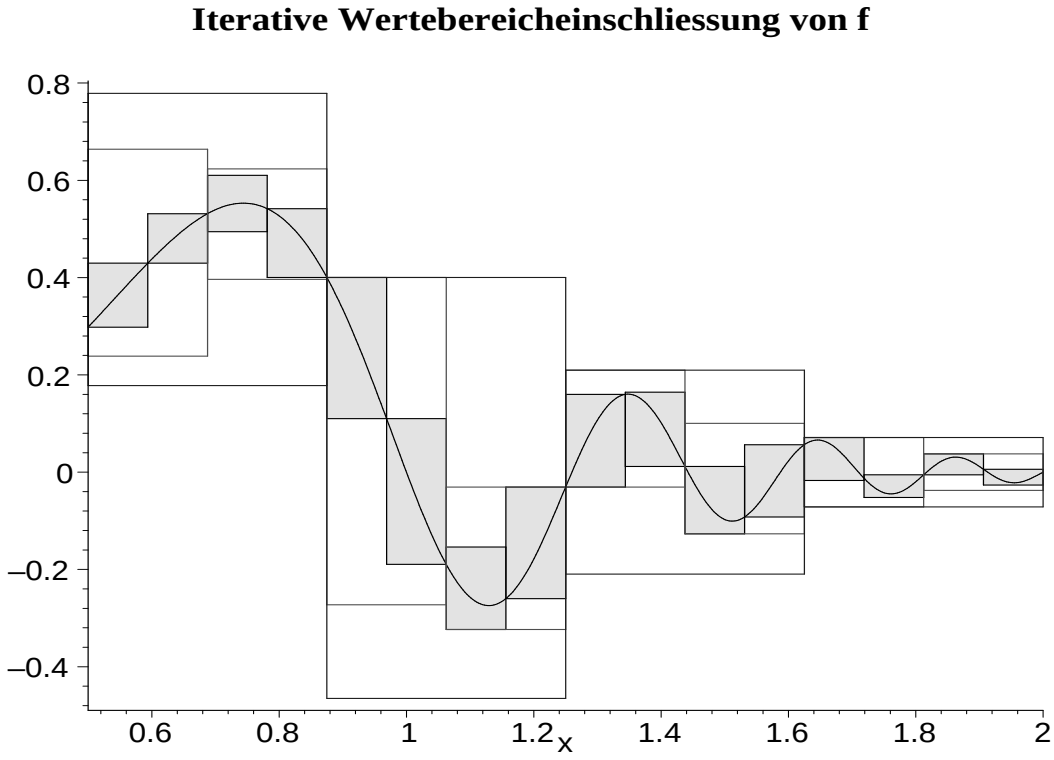
über dem Intervall $X := [0.5, 2.]$ mit Hilfe der Prozedur `compute_range`

```
> X:=[0.5,2.];
```

$$X := [.5,\ 2.]$$

```
> compute_range(f,X,4);

Startwertebereicheinschliessung =     [-.7788007834, .7788007834]

Wertebereicheinschliessung nach Iterationsschritt 4 =
[-.3233867682,.6103317518]
```

Die Startwertebereicheinschließung ist $\approx [-0.78, 0.78]$. Die Wertebereicheinschließung nach 4 Iterationsschritten, also nach Zerlegung in $2^4 = 16$ Teilintervalle, ist $\approx [-0.32, 0.61]$. Die graphische Ausgabe findet man in Abbildung 1 auf S. 19.



Die gleiche Wertebereicheinschließung erhält man bereits nach einem 1 Iterationsschritt, wenn das Startintervall schon vorab in 2^3 Intervalle zerlegt und der optionale Parameter `quadratic` angegeben wird:

```
> compute_range(f,X,1,Nx = 2^3,quadratic);

Startwertebereicheinschliessung =     [-.7788007834, .7788007834]

Wertebereicheinschliessung nach Zerlegung in 8 Teilintervalle =
```

```
[-.3233867682,.6639743998]

Wertebereicheinschliessung nach Iterationsschritt 1 =
[-.3233867682,.6103317518]
```

Abbildung 1: Verfeinerung einer Wertebereicheinschließung durch Zerlegung des Argumentbereichs in Teilintervalle

Durch Verwendung des Parameters `adaptive` kann die Anzahl der Teilintervalle in der Regel deutlich verringert werden:

```
> compute_range(f,[0.5,2.],6,adaptive);

Startwertebereicheinschliessung =     [-.7788007834, .7788007834]

Wertebereicheinschliessung nach Iterationsschritt 6 =
[-.2834388814,.5563221618]
```

Die aktuelle Zerlegung des Startintervalls ist in der Variablen `list_of_intervals` gespeichert. Daher kann die Gesamtzahl der Teilintervalle jederzeit auf einfache Weise bestimmt werden. Im obigem Beispiel bestimmt man sie folgendermaßen (nach 6 Iterationsschritten):

```
> nops(list_of_intervals);
```

24

Ohne Angabe des Parameters `adaptive` wäre die Anzahl der Teilintervalle nach 6 Iterationsschritten gleich $2^6 = 64$.

Um nur den letzten Iterationsschritt und die Funktion f darzustellen kann das `plots`-Kommando `display` verwendet werden. Das Ergebnis des folgenden Aufrufs findet man in Abbildung 2.

```
> display([q[7],q[6]],title='Adaptive Zerlegung nach 6
Iterationsschritten',titlefont=[TIMES,BOLD,12]);
```

4.2 Funktionen zweier Variablen

Die Prozedur `compute_range3d` berechnet Wertebereicheinschließungen für reellwertige Funktionen zweier reeller Variablen über einem zweidimensionalen Intervall $X \times Y$.

Ihre Eingabeparameter sind (analog zu `compute_range`, allerdings **ohne** die optionalen Parameter `linear/quadratic` und `adaptive`)

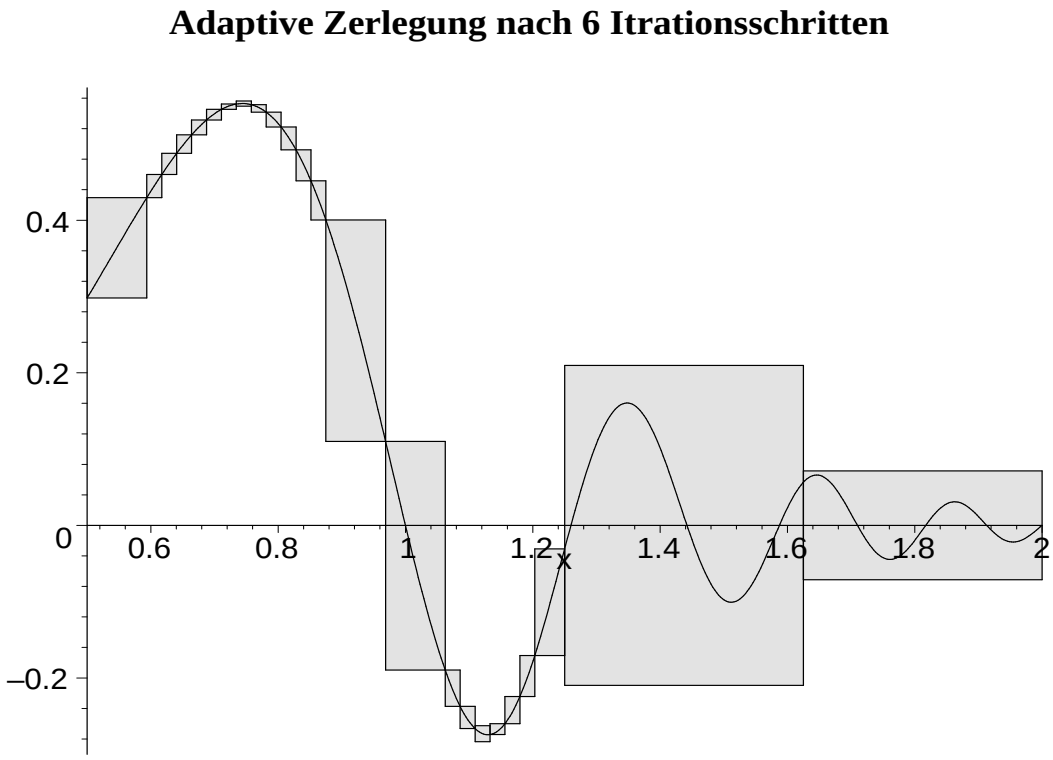


Abbildung 2: Adaptive Zerlegung

- die Funktion `f`,
- ein reelles Intervall, oder ein Bereich `X`,
- ein reelles Intervall, oder ein Bereich `Y`,
- die Anzahl der durchzuführenden Iterationsschritte.

Die Reihenfolge dieser Parameter ist zwingend. Auch bei dieser Prozedur können eine Reihe von optionalen Parametern angegeben werden

- die Parameter `Nx = n` und `Ny = m` bewirken entsprechende Zerlegungen des Startintervalls (achsenparalleles Rechteck) in `x`- und in `y`-Richtung,
- der Parameter `colorlist` hat dieselbe Bedeutung wie bei `compute_range`,
- der Parameter `cutout = r` bestimmt die Stärke der Linien bei der Darstellung der berechneten Einschließungen. Dabei sollte `r` gleich 0, 1 oder ein Bruch mit $0 < r < 1$ sein.

Zusätzlich können noch beliebige Optionen des `plot3d`-Kommandos verwendet werden.

Für die Bestimmung der Teilwertebereicheinschließungen wird bei `compute_range3d` die 'naive' intervallmäßige Auswertung verwendet. In jedem Iterationsschritt werden die Teilintervalle jeweils nur in eine Richtung zerlegt. Werden also z. B. zwei Iterationsschritte durchgeführt, so wird im ersten Schritt in `x`-Richtung zerlegt und im zweiten in `y`-Richtung.

Beispiel: Bestimmung einer Wertebereicheinschließung für die Funktion

```
> f:=(x,y)->exp(-x*y)*sin(Pi*x^2*y^2);
```

$$f := (x,\, y) \to e^{(-x\,y)} \sin(\pi\, x^2\, y^2)$$

über dem Intervall $X \times Y = [\pi/8, \pi/2] \times [\pi/8, \pi/2]$.

```
> X:=[evalf(Pi)/8,evalf(Pi)/2]; Y:=X;
```

$$X := [.3926990818,\ 1.570796327]$$
$$Y := [.3926990818,\ 1.570796327]$$

```
> compute_range3d(f,X,Y,4);

Startwertebereicheinschliessung =    [-.8570898115, .8570898115]

Wertebereicheinschliessung nach Iterationsschritt 1 =
[-.8570898115, .8570898115]

Wertebereicheinschliessung nach Iterationsschritt 2 =
[-.6800891261, .8570898115]

Wertebereicheinschliessung nach Iterationsschritt 3 =
[-.6800891261, .8486122905]

Wertebereicheinschliessung nach Iterationsschritt 4 =
[-.5093193828, .7559256232]
```

Nach jedem Iterationsschritt wird die berechnete Wertebereicheinschließung ausgegeben. Nur die Darstellung des letzten Iterationsschritts und die graphische Darstellung der Funktion f werden ausgegeben. Auch in diesem Fall wird die graphische Darstellung der anderen Iterationsschritte in der globalen Variablen `q` gespeichert.

Die graphische Ausgabe der Prozedur kann wie jede andere 3d-Graphik in Maple anschließend noch mit den Kommandos aus dem Graphik-Menü bearbeitet werden. Die gewünschten Graphikoptionen können aber auch direkt als Parameter übergeben werden. Z. B. erzeugt der Aufruf

```
> compute_range3d(f,X,Y,3,cutout=9/10,color=yellow,
> lightmodel=light2,axes=framed,titlefont=[TIMES,BOLD,12],
  title='Wertebereicheinschliessung durch Zerlegung in
  Teilintervalle',);
```

die Graphik aus Abbildung 3, S. 23.

Wertebereicheinschliessung durch Zerlegung in Teilintervalle

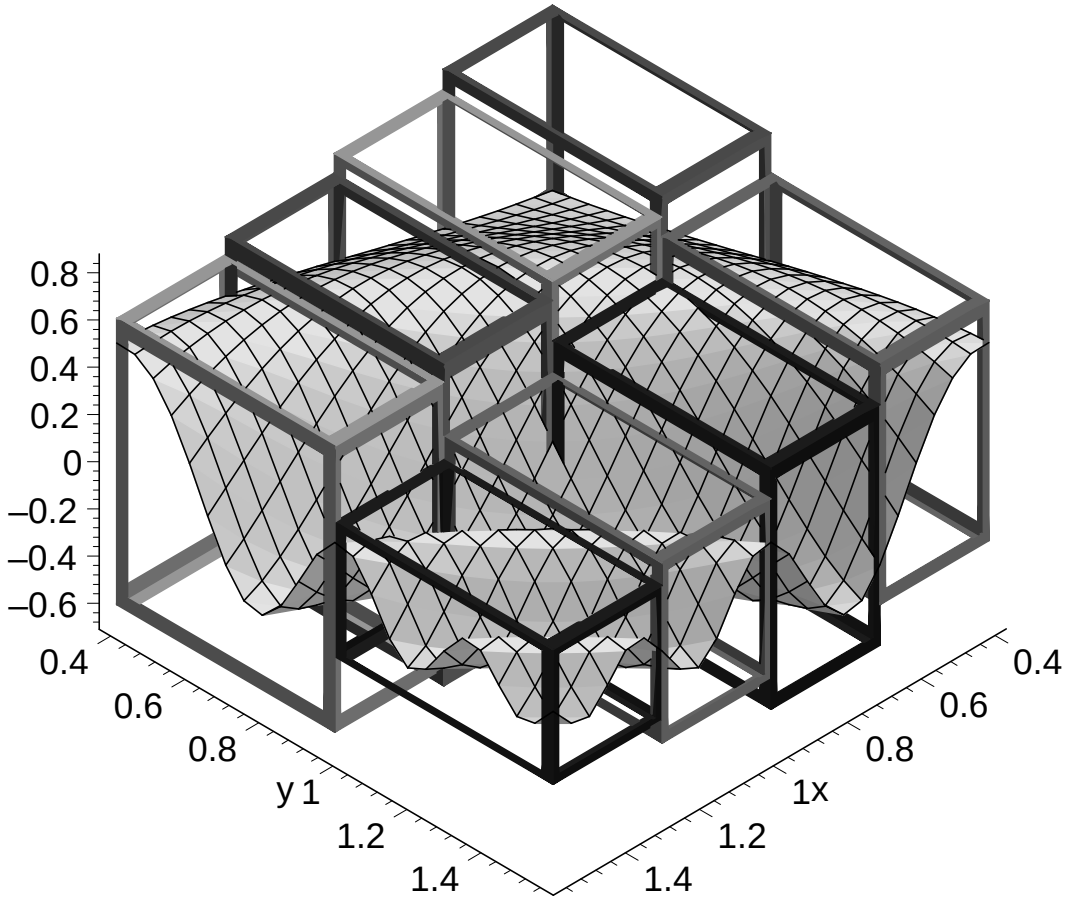


Abbildung 3: Wertebereicheinschliessung einer Funktion zweier Variablen

5 Verifizierte Nullstellenberechnung

Die Erweiterung `intpakX` enthält eine Realisierung der erweiterten Intervall-Newton-Iteration

$$\begin{cases} [x]^0 & , \quad \text{reelles Startintervall} \\ [x]^{k+1} := N([x]^k) \cap [x]^k, & k = 0, 1, 2, \ldots, \end{cases}$$

wobei

$$N([x]) := m([x]) - \frac{f(m([x]))}{f'([x])}$$

den Intervall-Newton-Operator bezeichnet und $m([x])$ i. a. den Mittelpunkt des

Intervalls $[x]$ darstellt.

Ist $f : D \subset \mathbb{R} \to \mathbb{R}$ eine auf D stetig differenzierbare Funktion und $[x]^0 \subset D$ ein reelles Intervall für das die intervallmäßige Auswertung $f'([x]^0)$ existiert, dann berechnet das erweiterte Intervall-Newton-Verfahren Einschließungen aller in $[x]^0$ enthaltenen Nullstellen der Funktion f. Zusätzlich kann mit Hilfe dieses Verfahrens die Existenz und Eindeutigkeit einfacher Nullstellen von f im angegebenen Startintervall nachgewiesen werden. Eine genauere Beschreibung des Verfahrens findet man in [8].

Der Fall $0 \in f'([x]^0)$ ist zugelassen! Daher benötigt man zur Durchführung des Verfahrens die erweiterte Intervalldivision und die Subtraktion eines erweiterten Intervalls von einer reellen Zahl (erweiterte Intervallsubtraktion, siehe Seite 25).

5.1 Erweiterte Intervalldivision und erweiterte Intervallsubtraktion

Sei $\mathbb{IR}$ die Menge der reellen Intervalle und

$$\mathbb{IR}^* := \mathbb{IR} \cup \{[-\infty, r] \,|\, r \in \mathbb{R}\} \cup \{[l, +\infty] \,|\, l \in \mathbb{R}\} \cup \{[-\infty, +\infty]\}$$

die Menge der erweiterten Intervalle.

Die in `intpak` bzw. `intpakX` verwendeten Definitionen der erweiterten Intervalldivision und der erweiterten Intervallsubtraktion entsprechen den in [13] verwendeten Definitionen. Die so definierten Intervalloperationen sind inklusionsisoton.

Für zwei reelle Intervalle $[x] = [\underline{x}, \overline{x}]$ und $[y] = [\underline{y}, \overline{y}]$ wird die erweiterte Intervalldivision folgendermaßen definiert

$$[x]/[y] := \begin{cases} [x] \cdot [1/\overline{y}, 1/\underline{y}], & \text{falls} \quad 0 \notin [y] \\ [-\infty, +\infty], & \text{falls} \quad 0 \in [x] \quad \text{und} \quad 0 \in [y] \\ [\overline{x}/\underline{y}, +\infty], & \text{falls} \quad \overline{x} < 0 \quad \text{und} \quad \underline{y} < \overline{y} = 0 \\ [-\infty, \overline{x}/\overline{y}] \cup [\overline{x}/\underline{y}, +\infty], & \text{falls} \quad \overline{x} < 0 \quad \text{und} \quad \underline{y} < 0 < \overline{y} \\ [-\infty, \overline{x}/\overline{y}], & \text{falls} \quad \overline{x} < 0 \quad \text{und} \quad 0 = \underline{y} < \overline{y} \\ [-\infty, \underline{x}/\underline{y}], & \text{falls} \quad 0 < \underline{x} \quad \text{und} \quad \underline{y} < \overline{y} = 0 \\ [-\infty, \underline{x}/\underline{y}] \cup [\underline{x}/\overline{y}, +\infty], & \text{falls} \quad 0 < \underline{x} \quad \text{und} \quad \underline{y} < 0 < \overline{y} \\ [\underline{x}/\overline{y}, +\infty], & \text{falls} \quad 0 < \underline{x} \quad \text{und} \quad 0 = \underline{y} < \overline{y} \\ [\,], & \text{falls} \quad 0 \notin [x] \quad \text{und} \quad 0 = [y]. \end{cases}$$

Da der Datentyp `interval` die Punkte $-$`infinity` und `infinity` als Intervallgrenzen zuläßt, kann die erweiterte Intervalldivision ohne Schwierigkeiten in das Intervallpaket eingebunden werden. Das entsprechende Kommando in `intpakX` heißt `ext_int_div`. Beispiele:

```
> ext_int_div([1.,2.],[-1.,1.]);
```
$$[-\infty, -.999999999],\ [.999999999,\ \infty]$$
```
> ext_int_div([-2.,-1.],[0,2.]);
```
$$[-\infty,\ -.4999999999]$$

Für $r \in \mathbb{R}$ und ein Intervall $[y] \in \mathbb{IR}^* \cup \{[\,]\}$ wird die erweiterte Intervallsubtraktion definiert durch

$$r - [y] := \begin{cases} [r - \overline{y}, r - \underline{y}], & \text{falls} \quad [y] = [\underline{y}, \overline{y}] \in \mathbb{IR} \\ [-\infty, +\infty], & \text{falls} \quad [y] = [-\infty, +\infty] \\ [r - \overline{y}, +\infty], & \text{falls} \quad [y] = [-\infty, \overline{y}] \\ [-\infty, r - \underline{y}], & \text{falls} \quad [y] = [\underline{y}, +\infty] \\ [\,], & \text{falls} \quad [y] = [\,]. \end{cases}$$

Diese realisiert bereits der in `intpak` enthaltene Subtraktionsoperator &$-$. Beispiele:

```
> 1 &- [1.,infinity];
```
$$[-\infty,\ 0]$$
```
> 1 &- [-infinity,1.];
```
$$[0,\ \infty]$$
```
> 1 &- [];
```
$$[\,]$$
```
> 1 &- [-infinity,infinity];
```
$$[-\infty,\ \infty]$$

5.2 Erweitertes Intervall-Newton-Verfahren

Die Prozedur `compute_all_zeros` berechnet mit Hilfe des Intervall-Newton-Verfahrens Einschließungen aller Nullstellen einer stetig differenzierbaren Funktion in einem eingegebenen Startintervall.

Die Eingabeparameter der Prozedur `compute_all_zeros` sind

- die Funktion `f`, deren Nullstellen berechnet werden sollen,
- das Startintervall `xstart` der Iteration und

– der gewünschte relative Durchmesser `eps` der zu berechnenden Nullstelleneinschließungen.

Der relative Durchmesser eines reellen Intervalls wird folgendermaßen definiert

$$d_{rel}([x]) := \begin{cases} \frac{d([x])}{\langle [x] \rangle}, & \text{falls } 0 \notin [x] \\ d([x]), & \text{sonst.} \end{cases}$$

Dabei bezeichnet $d([x])$ den Durchmesser und $\langle [x] \rangle$ das Betragsminimum des Intervalls $[x]$.

Die Reihenfolge der oben genannten Parameter ist zwingend. Als optionaler vierter Parameter kann die gewünschte Rechengenauigkeit, d. h. der Wert der Systemvariablen `Digits` innerhalb der Prozedur, eingegeben werden. Der vierte Parameter sollte also eine positive ganze Zahl größer gleich 10 sein. Fehlt dieser vierte Parameter, so wird die Rechengenauigkeit der geforderten relativen Genauigkeit und der Länge der Eingabeparameter angepaßt, ist aber auf jeden Fall größer gleich dem aktuellen Wert der Variablen `Digits`.

Ausgegeben werden die verwendete Rechengenauigkeit, die berechneten Nullstelleneinschließungen und zu jeder Nullstelleneinschließung die Information, ob die Existenz und Eindeutigkeit einer Nullstelle im angegebenen Intervall nachgewiesen werden konnte.

Ist das berechnete Intervall nur eine *potentielle Nullstelleneinschließung*, so kann es entweder eine, mehrere oder gar keine Nullstelle von f enthalten.

Die berechneten Nullstelleneinschließungen werden in der globalen Variablen `zeros` gespeichert und können daher beliebig weiter verarbeitet werden. `zeros` ist eine Tabelle und der Zugriff auf einen Eintrag in der Tabelle erfolgt in der üblichen Weise, also z. B. mittels `zeros[2]`.

Weitere in der Prozedur initialisierte globale Variablen sind die Tabelle `infos`, die die Zusatzinformationen enthält, die Anzahl der berechneten Nullstelleneinschließungen `N` und der Iterationszähler `iter_counter`.

Beispiel 1: Berechnung aller Nullstellen von

```
> f:=x->2*exp(tan(cos(x))) - sin(x) + cos(2*x);
```

$$f := x \rightarrow 2\, e^{\tan(\cos(x))} - \sin(x) + \cos(2\, x)$$

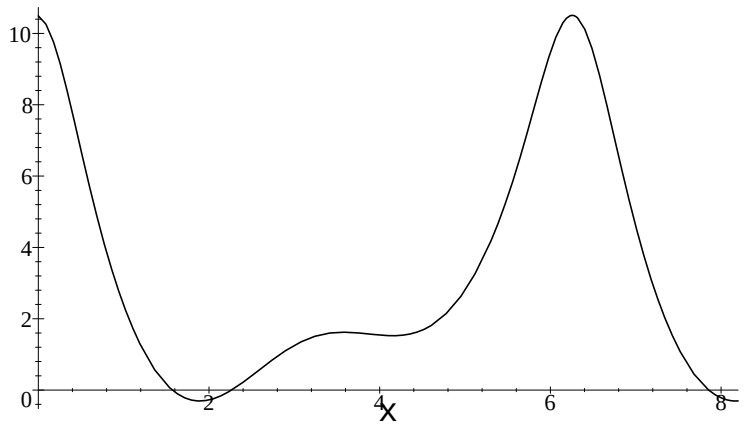


Abbildung 4: Darstellung der Funktion $f(x) := 2e^{\tan(\cos(x))} - \sin(x) + \cos(2x)$ im Intervall $[0, 8]$.

Die Funktion hat drei einfache Nullstellen im Intervall $[0, 8]$ (siehe Abbildung 4, S. 27). Zwei davon kann man exakt angeben, nämlich $\frac{\pi}{2}$ und $\frac{5\pi}{2}$. Einen Näherungswert für die dritte Nullstelle kann man mit Hilfe des Kommandos `fsolve` bestimmen.

Nachprüfen ob $\pi/2$ und $5\pi/2$ Nullstellen von f sind:

```
> f(Pi/2);
```

$$0$$

```
> f(5*Pi/2);
```

$$0$$

Berechnung der dritten Nullstelle mit `fsolve`, `Digits=30`:

```
> Digits:=30: zero3:=fsolve(f(x),x=2..2.5); Digits:=10:
```

$$zero3 := 2.26480074200004996505814286126$$

Berechnung der Nullstelleneinschliessungen, `Digits=20` (vierter Eingabeparameter):

```
> compute_all_zeros(f,[0,8.],10^(-10),20);
Digits =    20
```

$$[7.8539816339705772082,\ 7.8539816339775304044]$$

enthaelt genau eine Nullstelle

$$[2.2648007419999768034,\ 2.2648007420001249007]$$

enthaelt genau eine Nullstelle

$$[1.5707963267948966180,\ 1.5707963267948966204]$$
enthaelt genau eine Nullstelle

```
Anzahl der Nullstelleneinschliessungen:    3

Anzahl der Iterationsschritte:    21
```

Nachprüfen, ob die Nullstellen von f in den berechneten Nullstelleneinschließungen enthalten sind:

```
> is_in(evalf(Pi/2,30),zeros[3]);
```
$$true$$
```
> is_in(evalf(5*Pi/2,30),zeros[1]);
```
$$true$$
```
> is_in(zero3,zeros[2]);
```
$$true$$

Beispiel 2: Einschließung der Nullstelle von

```
> f:=x->(x-1)^3;
```
$$f := x \to (x-1)^3$$

mit Startintervall $[-3., 4.]$. Der relative Durchmesser der Einschließung soll $\leq 10^{-50}$ sein. (Die verwendete Rechengenauigkeit wird automatisch angepasst.)

```
> compute_all_zeros(f,[-3.,4.],10^(-50));

Digits =    55
```

$$[.99999999999999999999999999999999999999999999997208,$$
$$1.00000000000000000000000000000000000000000000005551]$$
potentielle Nullstelleneinschliessung

```
Anzahl der Nullstelleneinschliessungen:    1

Anzahl der Iterationsschritte:    163
```

Durchmesser und relativer Durchmesser der berechneten Nullstelleneinschliessung:

```
> Digits:=55:
> diam:=width(zeros[1]);
```
$$diam := .58302\,10^{-50}$$

```
> `relative_diam`:=rel_diam(zeros[1]);
```

$$relative\_diam := .583020000000000000000000000000000000000000000000001628\,10^{-50}$$

```
> Digits:=10:
```

5.3 Graphische Veranschaulichung

Zur graphischen Veranschaulichung des Intervall-Newton-Verfahrens steht die Prozedur `compute_all_zeros_with_plot` zur Verfügung. Sie berechnet analog zur Prozedur `compute_all_zeros` Nullstelleneinschließungen mit Hilfe des Intervall-Newton-Verfahrens. Zusätzlich wird jedoch jeder Iterationsschritt graphisch dargestellt.

Auch hier kann als optionaler vierter Parameter der Wert der Variablen `Digits` angegeben werden. Zusätzlich ist die Eingabe eines fünften optionalen Parameters möglich, der angibt, wieviele Iterationsschritte maximal durchgeführt werden sollen. Fehlt dieser fünfte Parameter, so muß die maximale Anzahl von Iterationsschritten interaktiv eingegeben werden. Die Eingabe muß mit einem Doppelpunkt oder einem Semikolon abgeschlossen werden.

Da die Prozedur `compute_all_zeros_with_plot` Kommandos aus dem `plots`-Package und aus dem `geometry`-Package enthält müssen diese vor dem Aufruf der Prozedur mit `with` geladen werden.

Beispiel: Berechnung einer Nullstelleneinschließung für die Funktion

```
> f:=x->exp(sin(x-1))-1;
```

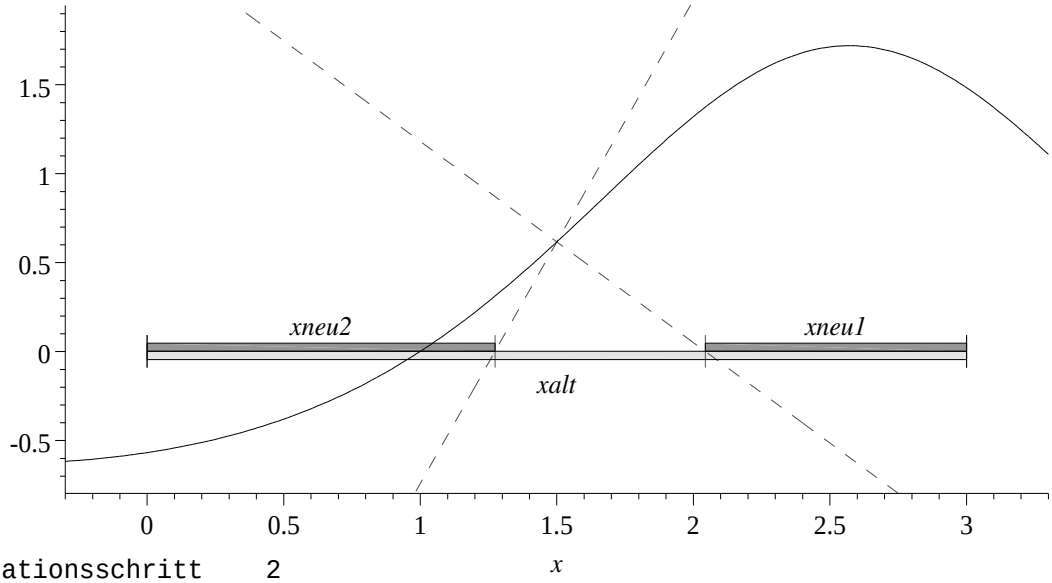
$$f := x \to e^{\sin(x-1)} - 1$$

im Intervall $[0, 3.]$:

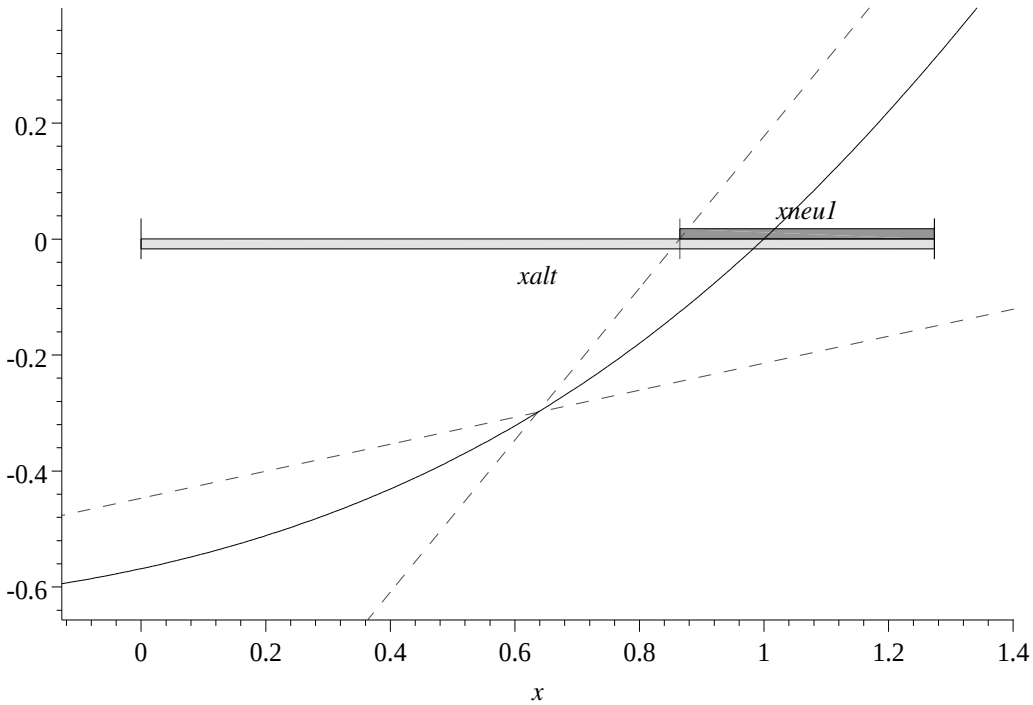
```
> compute_all_zeros_with_plot(f,[0.,3.],10^(-3));
> 10;
```

Die Ausgabe des Prozeduraufrufs findet man auf den Seiten 30 und 31. Die Steigungen der gestrichelt gezeichneten Geraden sind gegeben durch die kleinste bzw. größte Steigung aller Tangenten an den Graphen der Funktion im aktuellen Argumentbereich `xalt`. Die Geraden schneiden sich im Punkt (Entwicklungstelle, f(Entwicklungsstelle)). Die Schnittpunkte dieser Geraden mit der `x`-Achse sind wichtige Hilfsgrößen zur Bestimmung der neuen Iterierten des erweiterten Intervall-Newton-Verfahrens (siehe Seite 23).

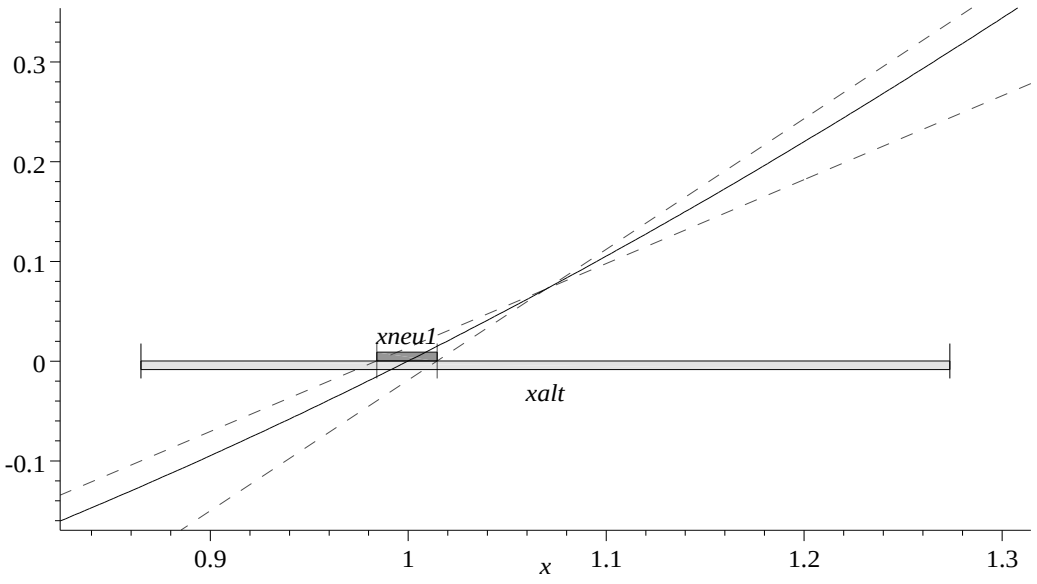
```
Digits =    10

Iterationsschritt    1
xalt=   [0, 3.]
xneu1=   [2.043797652, 3.]
xneu2=   [0, 1.273700327]
```



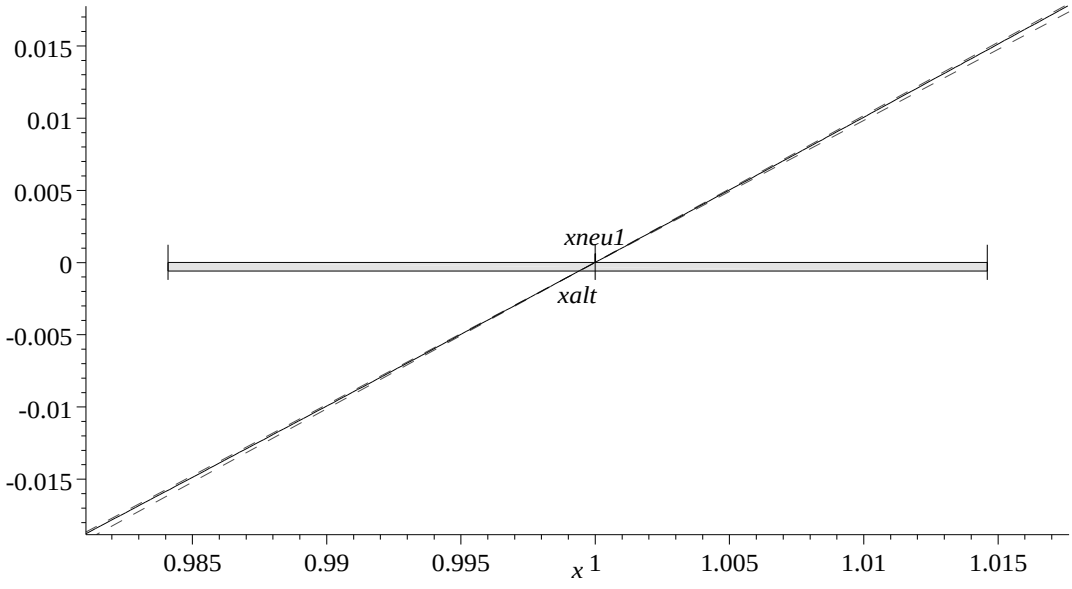

```
Iterationsschritt    2
xalt=   [0, 1.273700327]
xneu1=   [.8650186701, 1.273700327]
```



```
Iterationsschritt    3
xalt=   [.8650186701, 1.273700327]
xneu1=   [.9840864652, 1.014594170]
```




```
Iterationsschritt    4
xalt=   [.9840864652, 1.014594170]
xneu1=   [.9999902275, 1.000010447]
```



[.9999902275, 1.000010447]

enthaelt genau eine Nullstelle

```
Anzahl der Nullstelleneinschliessungen:    1
Anzahl der Iterationsschritte:    4
```

6 Kreisscheibenarithmetik

Aufbauend auf den reellen Intervalloperationen kann auch eine komplexe Intervallarithmetik definiert werden. In der Erweiterung `intpakX` wird eine solche Arithmetik für Kreisscheibenintervalle realisiert.

Ein Kreisscheibenintervall mit Mittelpunkt $z_0 \in \mathbb{C}$ und Radius $r > 0$

$$Z = \langle z_0,\, r\rangle := \{z \in \mathbb{C} \,|\, |z - z_0| \leq r\}$$

wird in `intpakX` als Liste mit drei Einträgen dargestellt: Realteil des Mittelpunkts z_0 von Z, Imaginärteil von z_0 und Radius r.

Die Bezeichnung des neuen Datentyps ist `complex_disc`. Als Komponenten des neuen Typs sind Zahlen vom Typ `numeric` zugelassen, d. h. insbesondere auch Zahlen vom Typ `integer` und vom Typ `fraction`.

Zur graphischen Darstellung eines Kreisscheibenintervalls kann die Prozedur `complex_disc_plot` verwendet werden. Sie hat als Eingabeparameter eine Variable vom Typ `complex_disc`. Als weitere optionale Parameter können die üblichen Darstellungsoptionen des Maple `plot`-Befehls eingegeben werden.

6.1 Arithmetische (Kreis-)Operationen

Die arithmetischen Grundoperationen für Kreisscheibenintervalle werden üblicherweise (siehe z. B. [1]) folgendermaßen definiert.

Seien $A = \langle a,\, r_a\rangle$ und $B = \langle b,\, r_b\rangle$ zwei Kreisscheibenintervalle. Dann ist

$$\begin{aligned}
A + B &:= \langle a + b,\, r_a + r_b\rangle \\
A - B &:= \langle a - b,\, r_a + r_b\rangle \\
A \cdot B &:= \langle a \cdot b,\, |a| r_b + |b| r_a + r_a r_b\rangle \\
1 \,/\, B &:= \left\langle \frac{\overline{b}}{b\overline{b} - r_b^2},\, \frac{r_b}{b\overline{b} - r_b^2} \right\rangle, \quad 0 \notin B \\
A \,/\, B &:= A \cdot (1 \,/\, B), \quad 0 \notin B
\end{aligned}$$

wobei $|a| = \sqrt{a_1^2 + a_2^2}$ den Betrag der komplexen Zahl $a = a_1 + i\, a_2$ und $\overline{b} = b_1 - i\, b_2$ die zu $b = b_1 + i\, b_2$ konjugiert komplexe Zahl darstellen.

Für die so definierten Operationen gilt mit den Bezeichnungen von oben

$$\begin{array}{lcl} A \pm B & = & \{a \pm b \,|\, a \in A,\, b \in B\} \\ A \cdot B & \overset{\text{i. a.}\neq}{\supseteq} & \{a \cdot b \,|\, a \in A,\, b \in B\} \\ 1 \,/\, B & = & \{1/b \,|\, b \in B\} \\ A \,/\, B & \overset{\text{i. a.}\neq}{\supseteq} & \{a/b \,|\, a \in A,\, b \in B\} \end{array}$$

Die oben definierten Operationen für Kreisscheibenintervalle werden in der Erweiterung `intpakX` durch die Operatoren &`cadd`, &`csub`, &`cmult` und &`cdiv` realisiert. Für die Inversion steht kein eigener Operator zur Verfügung.

Flächenoptimale Multiplikation und Division

Die oben definierte (sogenannte **zentrierte**) Multiplikation zweier Kreisscheibenintervalle $A = \langle a,\, r_a \rangle$ und $B = \langle b,\, r_b \rangle$ liefert bei vorgegebenem Mittelpunkt $a \cdot b$ eine optimale Einschließung des Punktergebniskomplexes $\{\alpha \cdot \beta \,|\, \alpha \in A,\, \beta \in B\}$. Diese Einschließung ist jedoch nicht flächenoptimal.

Die Bestimmung einer flächenoptimalen Einschließung der Punktergebnismenge bei der Multiplikation zweier Kreisscheibenintervalle ist aufwendiger und führt auf das Lösen einer Gleichung dritten Grades (siehe [10]).

Für $A = \langle a,\, r_a \rangle$ und $B = \langle b,\, r_b \rangle$ wird die **flächenoptimale** Multiplikation definiert durch

$$\begin{array}{rcll} A \cdot_{opt} B & := & \langle ab\,(1 + x_0), & (3\,|ab|^2 {x_0}^2 \\ & & & +2\,(|ab|^2 + |r_a b|^2 + |r_b a|^2)\, x_0 \\ & & & +|r_a b|^2 + |r_b a|^2 + (r_a r_b)^2\,)^{\frac{1}{2}} \rangle \end{array}$$

wobei x_0 die nichtnegative Nullstelle des Polynoms

$$P(x) \;=\; 2\,|ab|^2 x^3 + (|ab|^2 + |r_a b|^2 + |r_b a|^2) x^2 - r_a^2 r_b^2$$

ist, falls $\mathtt{grad}(P) \geq 2$ gilt (ansonsten wird $x_0 = 0$ gesetzt).

Die flächenoptimale Division zweier Kreisscheibenintervalle wird dann definiert durch

$$A /_{opt} B := A \cdot_{opt} (1/\,B)$$

Das Package `intpakX` enthält eine Realisierung der flächenoptimalen Multiplikation (&`cmult_opt`) und der flächenoptimalen Division (&`cdiv_opt`) zweier Kreisscheibenintervalle.

Generelles Vorgehen bei der Implementierung der Grundoperationen

Seien A und B zwei Kreisscheibenintervalle, die auf dem Rechner exakt darstellbar sind und sei $* \in \{+, -, \cdot, /\}$. Um auf der Maschine eine gesicherte Einschließung C des exakten Ergebniskomplexes $A * B$ zu erhalten, wurde bei der Implementierung folgendermaßen vorgegangen:

1. Berechne ein reelles Maschinenintervall cx, welches den Realteil des Mittelpunktes des Ergebnisintervalls einschließt, und ein reelles Maschinenintervall cy, welches den Imaginärteil des Mittelpunktes einschließt.

2. Berechne den Radius r des Ergebniskreises gemäß:

$$\begin{aligned} r1 &:= \sup(\text{Formel für den Radius intervallmäßig ausgewertet}) \\ r2 &:= \triangle(r1 + d(cx)) \\ r &:= \triangle(r2 + d(cy)) \end{aligned}$$

   wobei $\triangle$ die Rundung nach oben bezeichnet und $d(cx)$, $d(cy)$ den Durchmesser von cx bzw. von cy darstellen.

3. Setze $C = \langle m(cx) + i \cdot m(cy), r \rangle$. Wobei $m(cx)$ und $m(cy)$ den Mittelpunkt von cx bzw. von cy bezeichnen.

Um den Mittelpunkt eines Intervalls zu bestimmen, wird die Prozedur `mid` verwendet. Sie berechnet zum Unterschied von der `intpak`-Prozedur `midpoint` keine Einschließung des Mittelpunktes eines Intervalls sondern eine Zahl (Näherung des Mittelpunktes des Intervalls), die mit Sicherheit im eingegebenen Intervall liegt.

Ein Rechenbeispiel

Für $A = \langle 1, 1 \rangle$, $B = \langle -1 + i, 1 \rangle$ ist

$$\begin{aligned} A + B &= \langle i, 2 \rangle \\ A - B &= \langle 2 - i, 2 \rangle \\ 1 \,/\, B &= \langle -1 - i, 1 \rangle \end{aligned}$$

Berechnung mit Maple

```
> A:=[1,0,1]: B:=[-1,1,1]:
> A &cadd B;
```

$$[0,\ 1.000000000,\ 2.000000005]$$

```
> A &csub B;
```
$$[2.000000000,\ -1.000000000,\ 2.000000007]$$
```
> 1 &cdiv B;
```
$$[-1.000000001,\ -1.000000001,\ 1.000000051]$$

Bei Verwendung der zentrierten Multiplikation (siehe S. 32) erhält man

$$A \cdot B = \langle -1+i, 2+\sqrt{2}\rangle \approx \langle -1+i, 3.414213562\rangle$$
$$A \,/\, B = \langle -1-i, 2+\sqrt{2}\rangle$$

```
> A &cmult B;
```
$$[-1.000000000,\ 1.000000000,\ 3.414213579]$$
```
> A &cdiv B;
```
$$[-1.000000001,\ -1.000000001,\ 3.414213685]$$

und bei Verwendung der flächenoptimalen Multiplikation ergibt sich

```
> A &cmult_opt B;
```
$$[-1.390388204,\ 1.390388204,\ 2.969562256]$$
```
> A &cdiv_opt B;
```
$$[-1.390388219,\ -1.390388219,\ 2.969562345]$$

In Abbildung 5 sind die Punktergebnismenge bei der Multiplikation von A und B, die zentrierte Einschließung und die flächenoptimale Einschließung dieser Menge simultan dargestellt.

6.2 Wertebereiche komplexer Polynome

Eine erste Anwendungsmöglichkeit der in `intpakX` definierten Kreisarithmetik ist die Bestimmung von gesicherten Einschließungen für den Wertebereich eines Polynoms mit komplexen Koeffizienten über einem Kreisintervall.

Dazu stehen drei Prozeduren zur Verfügung

1. `horner_eval_cent` (Hornerschema unter Verwendung der zentrierten Multiplikation &`cmult`),
2. `horner_eval_opt` (Hornerschema unter Verwendung der flächenoptimalen Multiplikation &`cmult_opt`),
3. `centred_form_eval` (zentrierte Form für komplexe Polynome).

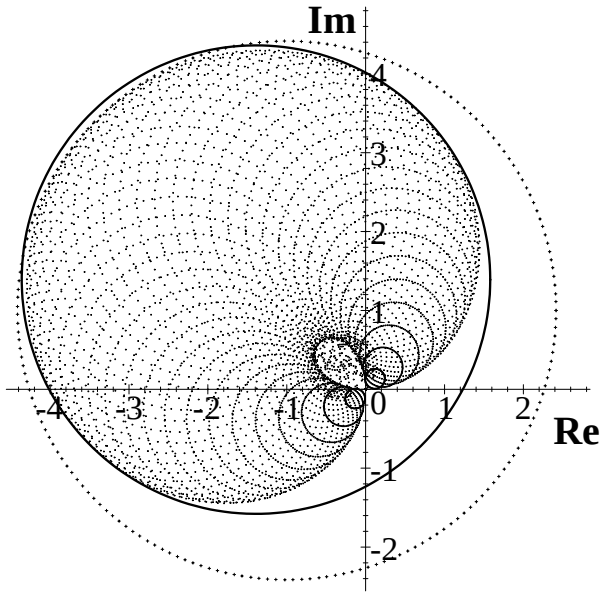


Abbildung 5: Zentrierte und flächenoptimale Einschließung von $\langle 1,1\rangle \cdot \langle -1+i,1\rangle$

Die Prozeduren `horner_eval_opt` und `centred_form_eval` liefern in der Regel deutlich bessere Einschließungen als die Prozedur `horner_eval_cent`. Sie haben allerdings auch einen wesentlich höheren Zeit- und Speicherplatzbedarf.

Der erste Eingabeparameter jeder dieser Prozeduren ist ein (komplexes) Polynom in der Variablen z. Die Bezeichnung für die Variable ist zwingend! Als zweiter Parameter muß eine Zahl oder eine Variable vom Typ `complex_disc` eingegeben werden.

Beispiel 1: Einschließung des Wertebereichs von

$$\begin{aligned} p(z) \;:=\; & (0.15-0.1i)+(0.15-0.12i)z+(-0.2-0.2i)z^2 \\ & +(0.1+0.3i)z^3+(0.1-0.2i)z^4+(0.1-0.2i)z^5 \\ & +(0.2-0.2i)z^6+(0.1-0.2i)z^7+(0.2-0.1i)z^8 \\ & +(0.1-0.1i)z^9 \end{aligned}$$

über dem Intervall $Z = \langle -0.1+0.2i,\, 0.9\rangle$.

```
> p_H:=horner_eval_cent(p,Z);
```

$$p\_H := [.1590115281,\ -.04050517670,\ 3.058832329]$$

```
> p_Hopt:=horner_eval_opt(p,Z);
```

$$p\_Hopt := [.2219721872,\ .2917174855,\ 2.243412729]$$

```
> p_C:=centred_form_eval(p,Z);
```

$$p\_C := [.1590115281,\ -.04050517670,\ 1.717944237]$$

Bei diesem Beispiel liefert die Prozedur `centred_form_eval` die beste Einschließung. Die graphische Darstellung des Wertebereichs von p über Z und der berechneten Einschließungen findet man in Abbildung 6 auf S. 37.

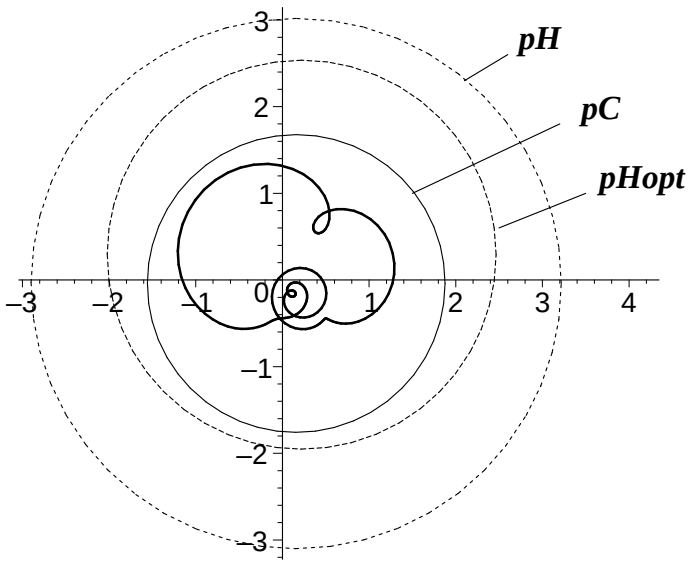


Abbildung 6: Einschließungen des Wertebereichs eines komplexen Polynoms

Erzeugung der Graphik

Zur Darstellung des Wertebereichs von p über Z kann das Kommando `complexplot` aus dem `plots`-Package verwendet werden. Die einzelnen Befehle sollten mit einem Doppelpunkt beendet werden (ansonsten wird sehr viel i. a. unnötige Information ausgegeben).

```
> with(plots):
> c1:=complexplot(subs(z=Z[1]+I*Z[2]+Z[3]*(cos(t)+I*sin(t)),p),
                  t=0..2*Pi,color=black,thickness=3,numpoints=200):
```

Zur graphischen Darstellung der berechneten Einschließungen wurde das Kommando `complex_disc_plot` verwendet.

```
> c2:=complex_disc_plot(p_H,color=black,thickness=2,linestyle=4):
> c3:=complex_disc_plot(p_Hopt,color=black,thickness=2,linestyle=3):
```

```
> c4:=complex_disc_plot(p_C,color=black,thickness=2):
```

Die Beschriftung der Graphik wurde mit einer Reihe von `plot-` und `textplot`-Befehlen erzeugt. `textplot` ist ein Kommando aus dem `plots`-Package.

```
> s1:=plot([[2.1,2.3],[2.6,2.6]],color=black,thickness=2):
> t1:=textplot([3,2.8,pH],font=[TIMES,BOLDITALIC,12]):
> s2:=plot([[1.5,1],[3.2,1.8]],color=black,thickness=2):
> t2:=textplot([3.7,2,pC],font=[TIMES,BOLDITALIC,12]):
> s3:=plot([[2.5,0.6],[3.5,1]],color=black,thickness=2):
> t3:=textplot([4.2,1.2,pHopt],font=[TIMES,BOLDITALIC,12]):
```

Zur simultanen Ausgabe der oben erzeugten Graphiken, kann das Kommando `display` aus dem `plots`-Package verwendet werden.

```
> display([c1,c2,c3,c4,s1,s2,s3,t1,t2,t3],scaling=constrained);
```

Beispiel 2: Einschließung des Wertes von

$$p(z) := (z-i)^4(z-1-i)^5$$

and der Stelle $z = 1$. Der exakte Wert $p(1) = 4i$.

```
> subs(z=1,p);
```

$$4\,I$$

Berechnung von Einschließungen mit Hilfe der Kreisintervallarithmetik:

```
> Digits:=60:
> horner_eval_cent(p,1);
```

$$[0,\ 4.000000000000000000000000000000000000000000000000000000000000,\ .281310000000000000000000000000000000000000000000000000000045\,10^{-55}]$$

```
> horner_eval_opt(p,1);
```

$$[0,\ 4.000000000000000000000000000000000000000000000000000000000000,\ .281310000000000000000000000000000000000000000000000000000045\,10^{-55}]$$

```
> centred_form_eval(p,1);
```

$$[0,\ 4.000000000000000000000000000000000000000000000000000000000000,\ .112542000000000000000000000000000000000000000000000000000010\,10^{-54}]$$

Alle drei Prozeduren liefern als Einschließung von $p(1)$ ein Kreisintervall mit Mittelpunkt $4i$ und die Radien der berechneten Einschließungen liegen im Bereich von 10^{-55}, falls mit 60 Stellen gerechnet wird (`Digits = 60`).

6.3 Die Exponentialfunktion für Kreisscheibenintervalle

Das Bild eines Kreisintervalls $Z = \langle c, r \rangle$ unter der Exponentialfunktion ist i. a. keine Kreisscheibe. Wird als Mittelpunkt des Ergebnisintervalls der Punkt $\exp(c)$ vorgegeben, so wird durch

$$\exp(Z) := \langle \mathsf{e}^c, |\mathsf{e}^c|(\mathsf{e}^r - 1) \rangle$$

eine optimale Einschließung (bei vorgegebenem Mittelpunkt $\exp(c)$) des Punktergebniskomplexes $\{\exp(z) \,|\, z \in Z\}$ definiert.

Eine ausführliche Diskussion über Bilder von Kreisintervallen unter der Exponentialfunktion findet man in [4].

Die Realisierung der Exponentialfunktion für Kreisscheibenintervalle `cexp` aus der Erweiterung `intpakX` hat als Eingabeparameter eine Variable vom Typ `complex_disc` oder eine komplexe Zahl und liefert eine gesicherte Einschließung des Punktergebniskomplexes.

Beispiel: Einschließung des Bildes von $Z = \langle 0, \pi + 1 \rangle$ unter der Exponentialfunktion

```
> Cexp:=cexp([0,0,evalf(Pi+1)]);
```

$$Cexp := [1.000000000,\ 0,\ 61.90292461]$$

Graphische Darstellung der berechneten Kreisintervalleinschließung:

```
> c1:=complex_disc_plot(Cexp,color=black,thickness=3,numpoints=400):
```

Bild des Randes von $\langle 0, \pi + 1 \rangle$ unter der Exponentialfunktion (Achtung: `plots` muß vorher mit `with` geladen werden!):

```
> c2:=complexplot(exp(polar(Pi+1,phi)),phi=0..2*Pi,
                  color=black,thickness=3,numpoints=400):
```

Um innere Punkte des Bildbereichs darzustellen, wird bei fest gewähltem Winkel der Radius r von 0 bis $\pi + 1$ variiert. Beispiel:

```
> c2:=complexplot(exp(polar(r,0.5)),r=0..Pi+1,
                  color=black,thickness=3,numpoints=400):
```

Die einzelnen Graphikkommandos werden anschließend mit `display` zusammengefaßt. Bei der Ausgabe sollte als zusätzliche Option `scaling=constrained` eingegeben werden. Um einen Teilausschnitt darzustellen wurde die `plot`-Option `view` verwendet.

Die graphische Darstellung in Abbildung 7 zeigt, daß die zentrierte Einschließung des Bildes von $\langle 0, \pi + 1\rangle$ unter der Exponentialfunktion die bei vorgegebenem Ergebniskreis-Mittelpunkt $\exp(0) = 1$ optimale Einschließung ist. Sie ist aber bei weitem nicht flächenoptimal. Abbildung 8 auf S. 41 zeigt einen Teilausschnitt um den Nullpunkt.

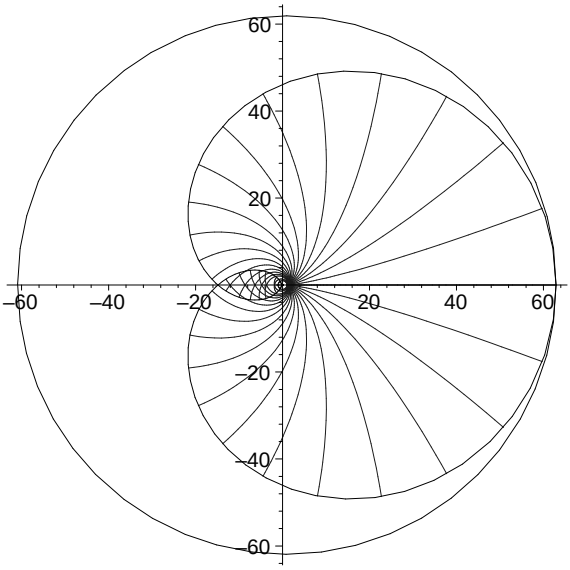


Abbildung 7: Bild des Kreisintervalls $\langle 0, \pi + 1\rangle$ unter der Exponentialfunktion und zentrierte Kreisintervalleinschliessung von $\exp(\langle 0, \pi + 1\rangle)$

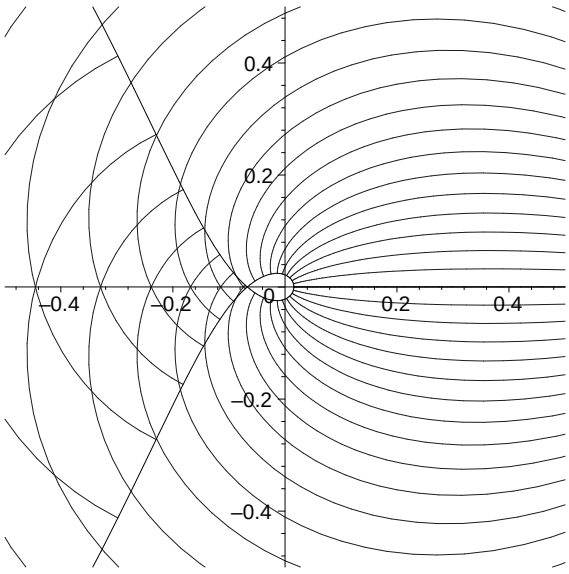


Abbildung 8: Darstellung eines Teilausschnittes um den Nullpunkt

Die Exponentialfunktion ist $2\pi\mathtt{i}$ periodisch und da der Durchmesser des Intervalls $Z = \langle 0,\ \pi + 1\rangle$ größer ist als 2π, gibt es im Bild von Z unter der Exponentialfunktion einen Bereich in dem jeder Punkt genau zwei Urbildpunkte besitzt. Dieser Bereich ist an der doppelten Schraffierung zu erkennen. Außerdem ist bei der Darstellung des Teilausschnittes der tropfenförmige bildpunktfreie Bereich um die Null zu erkennen.

7 Bewertung und Ausblick

Mit Hilfe von Verifikationsalgorithmen ist man gegebenenfalls in der Lage mit dem Rechner automatisch die Existenz und die Eindeutigkeit der Lösung eines Problems nachweisen zu können sowie eine (enge) Eischließung der exakten Lösung zu berechnen. Die auf diese Weise erzielten Aussagen haben die gleiche mathematische Qualität wie Ergebnisse, die z. B. durch den Einsatz von Computeralgebrasystemen, d. h. durch automatische Formelmanipulationen erzielt werden. Dabei stellt es sich als großer Vorteil heraus, daß Verifikationsalgorithmen als Eingaben auch toleranzbehaftete numerische Daten sicher behandeln können. In solchen Fällen werden simultan unendlich viele Probleme gelöst. Für eine ganze Familie von Problemen wird z. B. simultan nachgewiesen, daß jedes einzelne eine eindeutige Lösung besitzt.

Immer dann, wenn Computeralgebrapakete auf numerische Routinen zurückgreifen (z. B. bei der Berechnung bestimmter Integrale), sollte, wenn möglich, ein Verifikationsalgorithmus herangezogen werden. Die damit erzielten Aussagen sind dann mathematisch sicher (eine Eigenschaft, die man üblicherweise erwartet, wenn man mit einem CA-System arbeitet). Vorgetäuschte Lösungen bzw. unerkannte grob falsche Näherungen sind dann ausgeschlossen.

Auch können durch den Einsatz von Intervallverfahren die graphischen Fähigkeiten von Computeralgebrasystemen verbessert bzw. abgesichert werden. Daß dies notwendig ist, zeigt eindrucksvoll Beispiel 3 im Anhang.

Computeralgebra und Verifikationsnumerik ergänzen sich in idealer Weise. Sie machen den Rechner für den Mathematiker, aber auch für den Ingenieur zu einem sicheren mathematischen Werkzeug. Gerade im Hinblick auf die immer schneller und leistungsfähiger werdenden Prozessoren sollte die Symbiose von symbolischem Rechnen und sicheren numerischen Routinen massiv vorangetrieben werden.

8 Anhang: Fragwürdige Maple-Ergebnisse

Die folgenden Beispiele machen deutlich, daß auch mit Computeralgebrasystemen berechnete Ergebnisse sorgfältig geprüft werden müssen. Um die Verläßlichkeit zu steigern und um weitere Anwendungsgebiete zu erschließen, sollten ergänzend Verifikationsalgorithmen in solche Systeme integriert werden.

Beispiel 1: Falsche Berechnung von Minimum und Maximum

```
> f:=x->x^2+sin(x)+cos(2*x);
```

$$f := x \to x^2 + \sin(x) + \cos(2\,x)$$

Versuch der Berechnung des Wertebereichs von f über dem Intervall [-2.,0] mit Hilfe der Maple-Funktionen `minimize` und `maximize`:

```
> minimize(f(x),x,-2..0);
```

$$1$$

```
> r1:=evalf(%);
```

$$r1 := 1.$$

```
> maximize(f(x),x,-2..0);
```

$$4 - \sin(2) + \cos(4)$$

```
> r2:=evalf(%);
```

$$r2 := 2.437058952$$

```
> range_f:=[r1,r2];
```

$$range\_f := [1., 2.437058952]$$

Abbildung 9 zeigt, das `range_f` offensichtlich **keine** Einschliessung des Wertebereichs von f über dem Intervall $[-2., 0]$ ist!

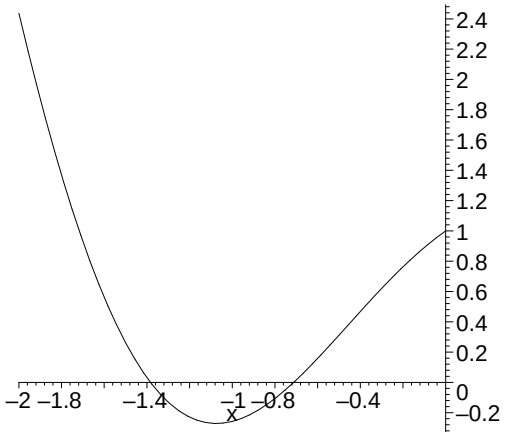


Abbildung 9: Darstellung der Funktion $f(x) := x^2 + \sin(x) + \cos(2x)$

Wertebereicheinschliessung mit Hilfe von `compute_range`:

```
> compute_range(f,[-2.,0],3,adaptive,Nx=4,quadratic):
> range_f:=r[3];
```

$$range\_f := [-.2786237965,\ 2.437058957]$$

Grobe Abschätzung des maximalen Fehlers bei der Bestimmung des Minimums:

```
> max(seq(width(list_of_ranges[i]),i=8..11));
```

$$.0499309122$$

Graphische Darstellung des letzten Iterationsschrittes:

```
> display([q[3],q[4]],view=[-1.8..-0.51,-0.28..1.5]);
```

Dieser Aufruf erzeugt die Abbildung 10.

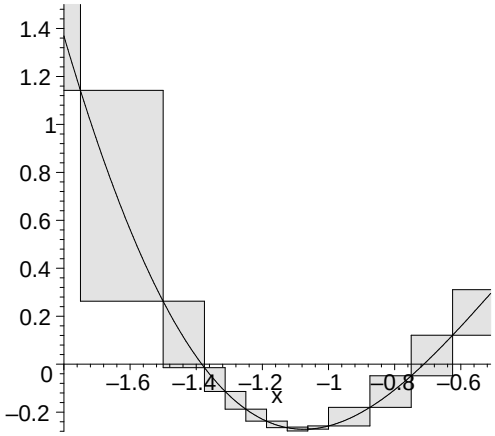


Abbildung 10: Verfizierte Wertebereicheinschließung

Beispiel 2: Fragwürdiges Integrationsergebnis und Grenzwertberechnung

```
> int(x/cosh(x), x=1..2);
```

$$-2\,I\ln(1+I\,e^2)-\frac{1}{2}\ln(1+I\,e^2)\,\pi-I\,\mathrm{dilog}(1+I\,e^2)+\frac{1}{2}\ln(e-I\,e^{(-1)})\,\pi-I\,\mathrm{dilog}(I\,e^2)$$
$$+\frac{1}{2}\ln(e+I\,e^{(-1)})\,\pi+\frac{1}{2}\,\pi+I\ln(1+I\,e)+\frac{1}{2}\ln(1+I\,e)\,\pi+I\,\mathrm{dilog}(1+I\,e)$$
$$-\frac{1}{2}\ln(e^{(1/2)}-I\,e^{(-1/2)})\,\pi+I\,\mathrm{dilog}(I\,e)-\frac{1}{2}\ln(e^{(1/2)}+I\,e^{(-1/2)})\,\pi$$

Ist diese Formel richtig?

```
> for k from 2 to 5 do evalf(Int(x/cosh(x), x=0..10^k)); od;
```

$$1.831931188$$
$$1.831931188$$
$$.2300979673\,10^{-28}$$
$$0$$

Konvergiert denn

$$\int_0^t \frac{x}{\cosh(x)} dx$$

mit wachsendem t gegen den Wert 0?

Beispiel 3: Unzuverlässige Graphik

Das folgende Maple-Kommando sollte einen Kreis erzeugen, tatsächlich ergibt sich jedoch die Graphik der Abbildung 11:

```
> implicitplot(x^2+y^2 = 1, x=-1..1, y=-15..50, numpoints=10000,
scaling=constrained);
```

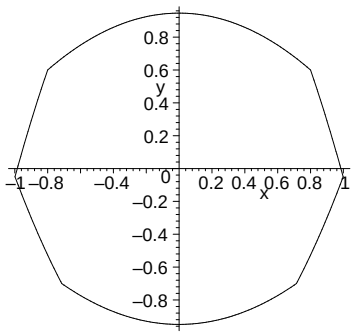


Abbildung 11: Ein Kreis?

Nun wird versucht, ein Gänseblümchen mit 90 Blütenblättern zu generieren (das Ergebnis gibt Abbildung 12 wieder):

```
> r:= 1/2*sin(90*t): plot( [(1+r)*cos(t), (1+r)*sin(t), t=0..2*Pi]);
```

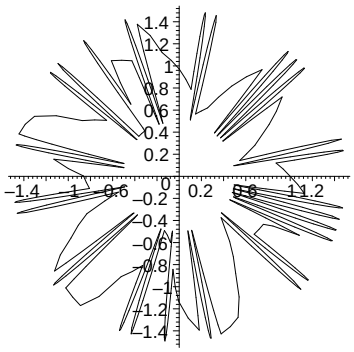


Abbildung 12: Ein Gänseblümchen mit 90 Blütenblättern?

Unterstützt man die Graphikroutinen durch geeignete Verifikationsschritte [3], so ergeben sich erwartungsgemäß in den Abbildungen ein Kreis bzw. eine stilisierte Blüte.

Literatur

[1] Götz Alefeld, Jürgen Herzberger: *Introduction to Interval Computations*, New York: Academic Press, 1983.

[2] Thomas Bauknecht: *Verifizierte numerische Quadratur in Maple*, Diplomarbeit (Betreuer: W. Krämer), Univ. Karlsruhe, 1997.

[3] Ulrich Bolz: *Verifizierte graphische Darstellung reeller Funktionen*, Diplomarbeit (Betreuer R. Lohner), Univ. Karlsruhe, 1996.

[4] Norbert C. Börsken: *Komplexe Kreis-Standardfunktionen*, Diplomarbeit, Univ. Freiburg, 1978.

[5] Amanda E. Connell, Robert M. Corless: *An Experimental Interval Arithmetic Package in Maple*, Tex-Dokument in Zusammenhang mit der Maple Share Library, 1993.

[6] George F. Corliss: *INTPAK for Interval Arithmetic in Maple: Introduction and Applications*, J. Symbolic Computation 11, 1994.

[7] Ilse Geulig: *Computeralgebra und Verifikationsalgorithmen*, Diplomarbeit (Betreuer: W. Krämer), Univ. Karlsruhe, 1998.

[8] R. Hammer, M. Hocks, U. Kulisch, D. Ratz: *Numerical Toolbox for Verified Computing I*, Berlin, Heidelberg: Springer-Verlag, 1993.

[9] Walter Krämer: *Computeralgebra und Verifikationsalgorithmen I und II*, Vorlesungen im WS 96/97 bzw. SS 97, Univ. Karlsruhe.

[10] Norbert Krier: *Komplexe Kreisarithmetik*, Dissertation, Univ. Karlsruhe, 1973.

[11] Arnold Neumaier: *Interval Methods for Systems of Equations*, Cambridge: Cambridge University Press, 1990.

[12] H. Ratschek, J. Rokne: *Computer Methods for the Range of Functions*, Chichester, West Sussex, England: Ellis Horwood Limited, 1984.

[13] Dietmar Ratz: *Inclusion Isotone Extended Interval Arithmetic*, Bericht 5/96, Institut für Angewandte Mathematik, Univ. Karlsruhe, 1996.

[14] A. Steins: *Verifizierte Formelauswertung in Computer-Algebra-Systemen*, Dissertation, Universität Wuppertal, 1996.

Seit Anfang letzten Jahres sind folgende Arbeiten in der Preprintreihe des IWRMM erschienen:

Nr. 98/1: S. Doll, R. Hauptmann, K. Schweizerhof, C. Freischläger: Selective Reduced Integration and Volumetric Locking in Finite Deformation Elastoviscoplasticity

Nr. 98/2: P. Vielsack, H. Kammerer: Finite Element Formulierung nichtglatter Schwingungen eines Balkens mit Reibglied

Nr. 98/3: H. Prautzsch: How Smooth are Subdividable Surfaces at Extraordinary Points?

Nr. 98/4: W. Wu, W. Rodi, Th. Wenka: 3D Numerical Modeling of Flow and Sediment Transport in Open Channels

Nr. 98/5: A. Bantle, W. Krämer: Ein Kalkül für verläßliche absolute und relative Fehlerabschätzungen

Nr. 98/6: R. Hauptmann, K. Schweizerhof, S. Doll: Extension of the "solid-shell" concept for application to large elastic and large elastoplastic deformations

Nr. 98/7: W. Hofschuster, W. Krämer: Eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-Double-Format

Nr. 98/8: U. W. Kulisch: Advanced Arithmetic for the Digital Computer – Design of Arithmetic Units

Nr. 98/9: B. Breuer, M. Plum: Lösungseinschließungen bei einem nichtlinearen Randwertproblem mittels eines Fourierreihenansatzes

Nr. 99/1: A. Kværnø, P. Rentrop: Low Order Multirate Runge-Kutta Methods in Electric Circuit Simulation

Nr. 99/2: I. Geulig, W. Krämer: Intervallrechnung in Maple – Die Erweiterung `intpakX` zum Paket `intpak` der Share-Library

Weitere Arbeiten sind in Vorbereitung.