Bergische Universität
Wuppertal

A Parallel Solver for (Systems of) Linear Fredholm Integral Equations of the second kind in C-XSC

Markus Grimmer

Preprint
BUW-WRSWT 2007/3

Wissenschaftliches Rechnen/
Softwaretechnologie

Impressum

Herausgeber:	Prof. Dr. W. Krämer, Dr. W. Hofschuster Wissenschaftliches Rechnen/Softwaretechnologie Fachbereich C (Mathematik und Naturwissenschaften) Bergische Universität Wuppertal Gaußstr. 20 42097 Wuppertal, Germany

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über die World Wide Web Seiten

`http://www.math.uni-wuppertal.de/wrswt/literatur.html`

Autoren-Kontaktadresse

Markus Grimmer
Bergische Universität Wuppertal
Gaußstr. 20
42097 Wuppertal, Germany

E-mail: `grimmer@math.uni-wuppertal.de`

A Parallel Solver for (Systems of) Linear Fredholm Integral Equations of the second kind in C-XSC

Markus Grimmer
University of Wuppertal
Department of Mathematics and Natural Sciences
Scientific Computing / Software Engineering
42097 Wuppertal, Germany
markus.grimmer@math.uni-wuppertal.de

Abstract

We present a parallel verified solver for linear Fredholm integral equations of the second kind based on a serial method by Klein. The serial and parallel methods are described, and test results from the new C-XSC implementation on the supercomputer ALiCEnext in Wuppertal, Germany, are given. At the same time, new and enhanced software components in C-XSC as an MPI interface for C-XSC data types and a parallel verified linear system solver, as well as the solution visualization via the Maple package intpakX are presented.

1. Introduction

Linear Fredholm integral equations of the second kind can be solved in the following manner firstly described by W. Klein [7, 8]: The kernel of the equation is split up into a degenerate and a non-degenerate part. First, a solving transformation for the non-degenerate part is computed. This transformation is then applied to the elements of the degenerate part yielding a new integral equation with degenerate kernel. An important part in the computation of the solution for the latter consists in the solution of a linear system.

The solution algorithm can be extended to systems of linear Fredholm integral equations of the second kind by transforming the system into a suitable matrix/vector notation. The possibility to solve systems of integral equations allows to obtain better results for a single equation by splitting it up into a system of integral equations with smaller domains. For growing system sizes, time increase is lower than for growing Taylor order when representing functions as Taylor approximations.

The system approach, unlike the approach for the single equation, is well suited for parallelization. We present a parallel verified solver for the above problem. Based on an old serial Pascal-SC implementation in [7], a new version as well for the single case as for the system case was implemented in C-XSC [6]. A newly developed parallel solver for integral equation systems reduces time effort, making more accurate solutions computable in reasonable time.

The implementation particularly includes a number of new software components which can also be used independently:

- An extended Interval Taylor arithmetic in C-XSC based on works by Bräuer, Krämer, Blomquist and Hofschuster [2, 3]
- A parallel verified linear system solver in C-XSC using "Rump's Device" [10]
- A new package for MPI communication with C-XSC data types [5], which was completed by MPI communication routines for Taylor arithmetic objects from the above Taylor arithmetic, and STL vectors.

Additionally, an interface for the export of the solution to Maple is provided. This allows both the visualization of the range of the solution and further computations with the solution functions, using the *Maple PowerTool Interval Arithmetic/intpakX* [4].

The parallel solver was tested on the parallel supercomputer ALiCEnext [1] in Wuppertal, Germany. Some results of the above implementations are presented.

2. Solving Fredholm Integral Equations of the second kind - An Approach

Firstly, some definitions and notations have to be given.

Let $k : [a, b] \times [a, b] \to \mathbb{R}$, $\alpha, g : [a, b] \to \mathbb{R}$ be continuous and y be the unknown result function.

$$y(s) - \lambda \int_a^b k(s,t)y(t)\,dt \;=\; g(s) \qquad (1)$$

is called *linear Fredholm Integral Equation of the second kind*.

If the kernel has a representation

$$k(s,t) = \sum_{i=1}^{T} a_i(s)b_i(t)$$

with continuous functions $a_i, b_i : [a,b] \to \mathbb{R}, i = 1...T$ it is called *degenerate kernel of order T*. In this case, the integral equation can be written as

$$y(s) - \lambda \sum_{i=1}^{T} a_i(s) \int_a^b b_i(t)y(t)\,dt \;=\; g(s). \qquad (2)$$

For a kernel k of an integral equation, the corresponding *integral operator*

$$K : C[a,b] \to C[a,b]$$

is defined as

$$(Ky)(s) := \int_a^b k(s,t)y(t)\,dt.$$

Then (1) can be written as

$$(I - \lambda K)\, y \;=\; g$$

with the identity I in $C[a,b]$.

For the solution of linear Fredholm integral equations of the second kind with degenerate and non-degenerate kernels, the following two theorems apply.

Theorem 1 *Every solution of a linear Fredholm Integral Equation of the second kind with degenerate kernel* (2) *and* $\lambda \neq 0$ *is of the form*

$$y(s) \;=\; g(s) + \lambda \sum_{i=1}^{T} a_i(s)\,\xi_i \qquad (3)$$

and (3) *is solution of* (2) *if and only if the* ξ_i *solve the following linear system:*

$$\xi_i - \lambda \sum_{i=1}^{T} \int_a^b b_i(t)a_j(t)\,dt\,\xi_j \;=\; \int_a^b b_i(t)g(t)\,dt$$

$i = 1 \dots T$.

The proof is given in [7] and can also be found in a couple of books on functional analysis.

Theorem 2 *For* $K : C[a,b] \to C[a,b]$ *let*

$$\lim_{n\to\infty} \|\lambda^n K^n\|^{\frac{1}{n}} < 1.$$

Then the Fredholm integral equation (1) *has exactly one solution which is given by*

$$y := y(\lambda) = \sum_{n=0}^{\infty} \lambda^n K^n g. \qquad (4)$$

The proof, again, is given in [7] and can also be found in some books on integral equations.

$$q(s,t) := \sum_{n=0}^{\infty} \lambda^n k^n(s,t)$$

is called *solving kernel*, the corresponding operator Q *solving transformation* of the equation.

This theorem induces a *Fixed Point Iteration* which can be applied to solve the integral equation from theorem 2:

$$\begin{aligned} y_0 &:= g, \\ y_{n+1} &:= g + \lambda K y_n,\ n = 0, 1... \end{aligned}$$

In general, the kernel of an integral equation (1) can be represented as

$$K = K_{\mathfrak{E}} + K_{\mathfrak{N}}$$

with a degenerate part $K_{\mathfrak{E}}$ and a non-degenerate part $K_{\mathfrak{N}}$. For instance, Taylor expansion of the kernel function yields this representation, with the Taylor summands forming the degenerate part and the remainder terms forming the non-degenerate part.

In [7], Klein proposes substituting this representation in the original equation. In that way we obtain

$$\begin{aligned} y - \lambda K_{\mathfrak{E}} y - \lambda K_{\mathfrak{N}} y &= g \\ y - \lambda K_{\mathfrak{N}} y &= g + \lambda K_{\mathfrak{E}} y \end{aligned}$$

and thus an integral equation with non-degenerate kernel with modified right-hand side. Applying the *solving transformation* Q of the iterative method yields:

$$\begin{aligned} y &= g + \lambda K_{\mathfrak{E}} y + Q(g + \lambda K_{\mathfrak{E}} y) \\ &= (g + \underbrace{Qg}_{(*)}) + \lambda(K_{\mathfrak{E}} y + \underbrace{QK_{\mathfrak{E}} y}_{(**)}) \end{aligned}$$

The results of the fixed point iterations $(*)$ and $(**)$ yield a new integral equation with degenerate kernel.

This yields a method for the solution of a general integral equation (1).

3. The Approach for Systems of Fredholm Integral Equations of the second kind

We now show how Klein applies the above method to systems of Fredholm Integral Equations of the second kind.

A system of Fredholm Integral Equations of the second kind is given by

$$y^i(s) - \lambda \sum_{j=1}^{N} \int_{\alpha_j}^{\beta_j} k^{ij}(s,t) y^j(t)\, dt = g^i(s) \qquad (5)$$

for $i = 1...N$ with continuous kernel functions k^{ij} on $[\alpha_i, \beta_i] \times [\alpha_j, \beta_j]$ with

$$\alpha_i := a + \frac{i-1}{N}(b-a), \;\; \beta_i := a + \frac{i}{N}(b-a), i = 1...N.$$

With

$$\begin{aligned} \mathcal{Y} &:= (y^1, ..., y^N)^T \\ \mathcal{G} &:= (g^1, ..., g^N)^T \\ \mathcal{K} &:= (k^{ij})_{i,j=1...N} \end{aligned}$$

we get

$$\mathcal{Y}(s) - \lambda \int_a^b \mathcal{K}(s,t)\mathcal{Y}(t)\, dt \;=\; \mathcal{G}(s) \qquad (6)$$

With this denotation and matrices $\mathcal{A}_m$, $\mathcal{B}_m$, $m = 0..T$ for degenerate kernels, a system of integral equations (5) can be written just like a single integral equation (1) and the method for single equations can be applied.

In the next section, we give the general algorithm and describe its parallelization.

4. A Parallel Method

In Method 4.1 we show the general algorithm for Klein's system method using the notations of the above section.

It has to be pointed out that all statements mentioned above are applicable to interval functions and interval counterparts of the defined terms and objects, and Schauder's fixed point theorem is applicable to prove the existence of verified solutions.

The highlighted parts of the algorithm are to be parallelized. The remaining parts are of minor importance since they only have a small share of the method's overall complexity.

The parts (A), (B) and (C) from method 4.1 have been parallelized in the following way.

Iterations

Part (A) can be parallelized in an easy way since the iterations can be carried out independently and thus need no intermediate communication between processes. Hence, iterations can be distributed to the processes, either evenly or using an active distribution strategy. Tests have shown that an even distribution already yields quite acceptable results so that there isn't a great need for active distribution.

Method 4.1 : General System Method

For $j := 1...N, n := 1...2T$:
  Compute integrals of basic monomials $(t - t_0^j)^n$.
Carry out the iteration
  $\mathcal{F}^0 := \mathcal{G}$; $\mathcal{F}^{i+1} := \mathcal{G} + \mathcal{K}\mathfrak{M}\mathcal{F}^i$, $i := 0, 1, ...$
until $\mathcal{F} := \mathcal{F}^{i+1} \subseteq \mathcal{F}^i$ (or abort).

For $m := 0...T$: (A)
  For $j := 1...N$:
    Carry out the iteration
      $\mathcal{C}_m^{j,0} := \mathcal{A}_m^j$; $\mathcal{C}_m^{j,i+1} := \mathcal{A}_m^j + \mathcal{K}\mathfrak{M}\mathcal{C}_m^{j,i}$
      $i := 0, 1, ...$
    until $\mathcal{C}_m^j := \mathcal{C}_m^{j,i+1} \subseteq \mathcal{C}_m^{j,i}$ (or abort).

Compute the entries (B)
- $\mathcal{M} := (\mathcal{M}_{ij})_{i,j=0...T}, \mathcal{M}_{ij} := \int_a^b \mathcal{B}_i(t)\mathcal{C}_j(t)dt$
- $\mathcal{R} := (\mathcal{R}_i)_{i=0...T}, \mathcal{R}_i := \int_a^b \mathcal{B}_i(t)\mathcal{F}(t)dt$

of the interval linear system for the final application of the method for degenerate kernels.

Solve the interval linear system (C)
  $(I - \lambda\mathcal{M})\mathcal{X} = \mathcal{R}$.

Compute the solution function $\mathcal{Y} := \mathcal{G} + \lambda \sum_{m=0}^{T} \mathcal{C}_m \mathcal{X}_m$.

Computation of linear system entries

The matrix of the linear system to be finally solved can be computed elementwise by the processes involved. Elements, again, can be computed independently. For best use in the subsequent solution of the linear system, the desired distribution of data for the linear system solver should already be taken into account in this step so that every process computes the entries of the matrix it will store during the linear system solution.

Parallel linear system solver

The linear system solver is also parallelized. Since it is implemented as a separate component, it is described in the next section convering software components.

5. Software Components

For a full implementation of the described application, a number of software components were newly developed or enhanced. All components were implemented in C-XSC [6], adding new and extending existing modules in the range of XSC software.

The following components are available:

- MPI communication facilities for (NEW)
  - C-XSC data types
  - Taylor arithmetic types
- Parallel verified linear system solver (Rump's method) (NEW)
- 1D and 2D Taylor arithmetic in C-XSC (ENHANCED)
- Output interface for Maple (NEW INTERFACE)

The implementation of these components is described in the next paragraphs.

5.1. MPI communication package for C-XSC

To successfully implement parallel programs with C-XSC data types, you need communication facilities for those types. Communication with user defined data types can be done in different ways:

- Direct application of existing routines, applied to single data elements of objects
- Usage of the MPI data packing/unpacking mechanism
- Definition of the data type in MPI using MPI's data type definition mechanism

These approaches all have advantages and disadvantages.

Direct application, naturally, is a way that always works. Unfortunately, MPI routines can't simply be applied to the data object you use in your application since MPI only knows to handle elements or arrays of elements of a number of basic types as long as you don't do more specification work. This is inconvinient for the programmer of the application and at the same time unnecessarily time consuming, since communication for a single object often needs a number of communication calls.

MPI offers two strategies to make communication of data more convenient. The first of these is the *packing/unpacking* mechanism. It is the more versatile of the two since it allows to pack, then send, receive and finally unpack every kind of data you like. On the debit side there is an extra copying process which also uses additional memory.

The second strategy is the definition of new MPI types. There is a collection of routines for data type definition in MPI making it possible to virtually assemble data by defining a so called *type map* and giving it a name, but it has limitations. Namely, no dynamically allocated data can be incorporated into the new type this way, and lenghts of any data structures of variable size have to be known beforehand to construct the new type.

Hence, the package implemented for C-XSC data types uses the first of the two strategies for types that incorporate dynamic memory allocation and/or array-like structures like vector and matrix types.

It uses data type definition for the basic C-XSC data types `real`, `interval`, `complex` and `cinterval`, and includes packing/unpacking routines and communication routines as template functions for the types derived from these. In general, all C-XSC types are included:

- Basic types: `real`, `interval`, `complex`, `cinterval`
- Vector and Matrix data types for the basic types
- Staggered (= multiple precision) data types
- Dotprecision ("accumulator") data types

Additionally, types from the Taylor package briefly described in paragraph 5.3 are covered including STL vectors of these types.

Regarding MPI communication, the following routines are covered:

- `MPI_Pack`, `MPI_Unpack`
- Point-to-point communication:
  - `MPI_Send`, `MPI_Bsend`, `MPI_Ssend`, `MPI_Rsend`
  - `MPI_Isend`, `MPI_Ibsend`, `MPI_Irsend`, `MPI_Issend`
  - `MPI_Recv`
- Collective Communication:
  - `MPI_Bcast`
- Additional versions of the above routines for submatrices/rows/columns are included as well.

In a general interface or package, it is not possible to offer general versions of further collective communication routines, since data subdivision for gather/scatter processes can be done in various ways, and the decision how to subdivide and distribute data has to be left to the author of the application.

5.2. Parallel verified linear system solver

The parallel verified linear system solver uses Rump's method [10], i.e. a Newton-like iteration for an approximate inverse R of a matrix A followed by a repeated iteration for the approximate inverse S of RA if the first step is not successful.

Method 5.1 : Parallel linear system solver

Compute approximate inverse R of A.

p_0:	$p_1, ..., p_q$: **(*)**
Apply real residual iteration to obtain better initial value. Compute enclosure $[z]$.	Compute enclosure $[C]$.
Apply verified iteration $[x]^{(i+1)} := [z] + [C][x]^{(i)}$, $i = 0, 1, ...$	Compute RA.

If successful: $[x] := \tilde{x} + [x]$. (Approx. solution + residuum)

Else: Compute approx. inverse S of RA, let $R := SR$. Repeat **(*)**.

If successful: $[x] := \tilde{x} + [x]$. Else: failed.

The iteration step is defined as follows:

$$y_{k+1} = \underbrace{R(b - A\tilde{x})}_{=:z} + \underbrace{(I - RA)}_{=:C} y_k, k = 0, 1, ... \qquad (7)$$

Based on this notation, method 5.1 shows the general parallel algorithm of the parallel solver.

The solver is implemented as a module with various interface versions that can be used in all appropriate applications. Matrix inversion and further matrix operations are parallelized and offered in separate units, so that they can be used independently as well.

5.3. Taylor arithemetic in C-XSC

Bräuer, Blomquist, Hofschuster and Krämer implemented a Taylor arithmetic package in in C-XSC in [2, 3]. This implementation was extended to cover

- the Gaussian error function *erf* and complementary error function *erfc* as new functions
- a number of operators for more convenient use of the package

This Taylor arithmetic package is used in the integral equation solver, and MPI communication routines for these types are included in the MPI communication package for C-XSC described above.

5.4. Maple interface

Finally, there is a Maple interface included in the integral equation application. It offers output of solution functions as Maple code and output of the range enclosure of the solution functions for display in Maple.

Interval arithmetic in Maple is provided by the Maple package intpakX [4].

Example

Given the integral equation

$$\begin{aligned} & y(s) - \frac{1}{2}\int_0^1 (s+1)e^{-st}y(t)\,dt \\ = \quad & e^{-x} - \frac{1}{2} + \frac{1}{2}e^{-x+1}, \quad s,t \in [0,1] \end{aligned}$$

which, according to Kress [9], has the analytic solution

$$y : s \to e^{-s},$$

the Taylor coefficients of the first component of the solution (computed with the system method) are computed as

```
[ 0.969232610773108516, 0.969235565177278713]
[-0.969233189716268040,-0.969233131674795078]
[ 0.484616598544399534, 0.484616637435440978]
[-0.161554269639638399,-0.161550720767527217]
[ 0.038892044738204872, 0.041956101920650240]
```

The function output in Maple code is

```
IGLSys_Lsg[0] := x ->
[ 0.969232610773108516,   0.969235565177278713]
&+ ( [-0.969233189716268040,-0.969233131674795078]
&* ( ( x
&- [ 0.031250000000000000, 0.031250000000000000] )
   &intpower 1 ) )
&+ ( [ 0.484616598544399534, 0.484616637435440978]
&* ( ( x
&- [ 0.031250000000000000, 0.031250000000000000] )
   &intpower 2 ) )
&+ ( [-0.161554269639638399,-0.161550720767527217]
&* ( ( x
&- [ 0.031250000000000000, 0.031250000000000000] )
   &intpower 3 ) )
&+ ( [ 0.038892044738204872, 0.041956101920650240]
&* ( ( x
&- [ 0.031250000000000000, 0.031250000000000000] )
   &intpower 4 ) ) ;
```

The solution can be visualized in Maple as shown in figure 1.

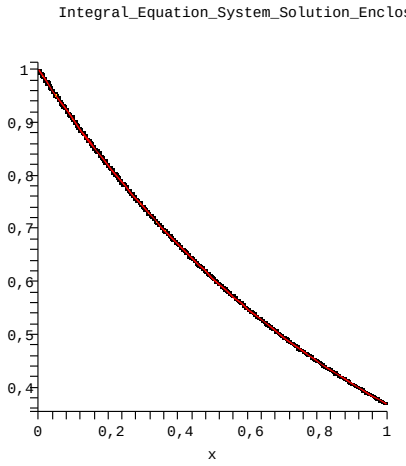


Figure 1. Maple visualization of integral equation solution.

6. Some results

A range of tests with the packages presented above has been done.

These tests include:

- Tests of the performance of the C-XSC/MPI communication package
- Tests of the parallel performance of the integral equation solver
- Analysis of the parameters in the system approach

The test environment was the supercomputer ALiCEnext installed in Wuppertal, Germany since 2004:

- 1024 1.8 GHz AMD Opteron 64 Bit Processors on 512 nodes
- 2004: Top 500 Rank 74
- LINPACK max. performance 2083 GFlops

First, we give some results on the performance of the C-XSC/MPI communication package.

The following table shows communication times for communicating a C-XSC `imatrix` through 16 processors using

- manual elementwise communication
- our package (packing/unpacking)
- manually programmed derived data types for matrices with fixed sizes sending only the matrix data without organisational data like index bounds

Dimension	512	1024	2048	4096
Elementwise	0.97-2.09	4.11-8.35	16.99-22.57	67.8-101.6
Package	0.09-0.11	0.23-0.36	0.80-1.07	4.82-5.42
Derived Type	0.07-0.10	0.25-0.31	0.96-1.10	3.17-3.67

(Time in sec.)

We can see that communicating data elementwise is highly inefficient for objects that consist of a big number of elements. At the same time, our package and manually constructed derived types for fixed matrix sizes (not including communication of index bounds) have much better communication times that are similar to each other.

Next, we have times of the integral equation solver for a random example function. (Of course, times depend on the functions involved.) We show times for varying numbers of processors.

Processors	serial	4	16	64	128
Time	2060-2343	510-540	114-142	31-39	18-23

(Time in sec.)

The numbers show that the integral equation solver parallelizes almost linearly.

We also analyzed time shares of different parts of the program:

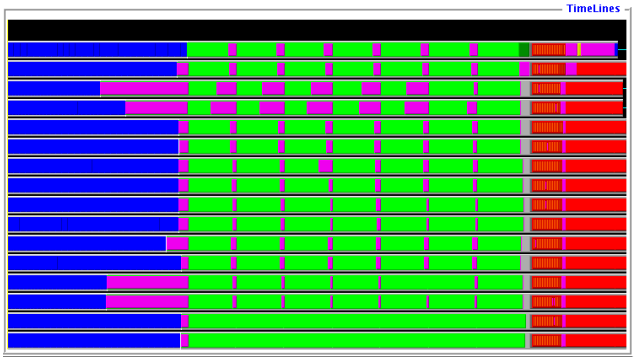
- Iteration phase
- Computation of the entries of the linear system
- Linear system solution (including approximate matrix inversion and further matrix operations, e.g. matrix multiplications)

Figure 2 shows these phases for an example with 16 processors and an integral equation with Taylor order 6 and system order 64 - a medium sized example. The overall computation time was ~ 140 seconds.

In general, the following was observable:

- Idle times resulting from the even and non-adaptive distribution of tasks occur but do not dominate in any way so that there is no urgent need for an adaptive distibution strategy
- The linear system solver had a greater time share for smaller system sizes. This is probably due to communication overhead that dominates computation times for small system sizes.
- The parts of ”minor importance” as for their time share behave like supposed above: Even for the medium sized example shown, time shares of those parts ($O(n^2)$ complexity) are negligible.

Finally, we show a table concerning result accuracy that shows to what extent it is possible to increase the accuracy



Iteration Phase
Communication and idle time
Linear system entry computations
Real matrix inversion
Further matrix operations

Figure 2. Time shares for parallel integral equation solver

of the result when splitting up a single integral equation into an integral equation system.

In the following table, you find the result accuracy of the solution of an example integral equation dependent on the order of the Taylor expansion of the involved functions and dependent on the order of the integral equation system.

System order → ↓ Taylor order	4	8	16	32	64	128
3	0	0	1	1	1	1
4	0	0	1	1	2	2
6	1	3	4	5	4	4
8	5	7	8	10	9	9
12	12	16	18	20	20	-

(Number of correct binary digits)

We find that increasing system order does not always work wonders, especially when Taylor order is fixed to a very small value. Nevertheless result accuracy can be considerably increased in most cases. Tests showed that for many examples increasing the system order further than ~ 64 didn't result in even higher accuracy. Still, there were examples (e.g. functions with denominators near to 0) which could only be solved for a system order of over 500.

Hence, we can conclude that the parallel verified integral system solver gives us the chance to successfully compute the verified solution of linear Fredholm integral equations of the second kind with high accuracy that couldn't previously be solved this accurate or even at all unless you spent a huge amount of computing time.

References

[1] *ALiCEnext information*. http://www.alicenext.uni-wuppertal.de.

[2] F. Blomquist, W. Hofschuster, and W. Krämer. Real and complex Taylor Arithmetic in C-XSC. Preprint BUW-WRSWT 2005/4, University of Wuppertal, 2005. http://www.math.uni-wuppertal.de/wrswt/literatur/lit_wrswt.html.

[3] M. C. Bräuer. Berechnungsmethoden für Ableitungen und Steigungen und deren Realisierung in C-XSC. Master's thesis, University of Karlsruhe, 1999.

[4] M. Grimmer. Interval Arithmetic in Maple with intpakX. In *PAMM - Proceedings in Applied Mathematics and Mechanics*, number Vol. 2, Nr. 1, pages 442–443. Wiley-InterScience, 2003.

[5] M. Grimmer. An MPI Extension for the use of C-XSC in parallel environments. Preprint BUW-WRSWT 2005/3, University of Wuppertal, 2005. http://www.math.uni-wuppertal.de/wrswt/literatur/lit_wrswt.html.

[6] W. Hofschuster and W. Krämer. C-XSC 2.0 – a C++ class library for extended scientific computing. In R. Alt, A. Frommer, B. Kearfott, and W. Luther, editors, *Numerical Software with Result Verification*, pages 15–35. Springer Lecture Notes in Computer Science 2991, 2004.

[7] W. Klein. *Zur Einschließung der Lösung von linearen und nichtlinearen Fredholmschen Integralgleichungssystemen zweiter Art*. PhD thesis, University of Karlsruhe, 1990.

[8] W. Klein. Enclosure methods for linear and nonlinear systems of Fredholm integral equations of the second kind. In E. Adams and U. Kulisch, editors, *Scientific computing with automatic result verification*, Boston, 1993. Academic Press.

[9] R. Kress. *Linear Integral Equations*. Springer, Berlin, Heidelberg, 1989.

[10] S. Rump. *Kleine Fehlerschranken bei Matrixproblemen*. PhD thesis, University of Karlsruhe, 1980.