b4m

A free interval arithmetic toolbox for Matlab based on BIAS
version 1.02.004
documentation version 1.00

Jens Zemke
Technische Universität Hamburg-Harburg
Technische Informatik III
Eißendorfer Straße 38
21071 Hamburg
email: `zemke@tu-harburg.de`

`http://www.ti3.tu-harburg.de/zemke/b4m/index.html`

Contents

List of Figures

1 Introduction

In the field of numerical computations the focus has changed during the last decades. For most problem classes standard algorithms are available and (at least) in the case of small sized problems a forward/backward error analysis has been carried out. These standard algorithms deliver in most cases very good results. However, in the case of ill-conditioned problems results may be afflicted with computational errors and worse, there might be no corresponding error message. Floating point numerical researchers currently work on standard algorithms for huge problems. Apart from this important task, for a long time there has been interest in

- the rigorous estimation of the condition of a problem,
- the inclusion of the solution of problems with uncertainties in the data

and/or

- proving possibly existence and uniqueness of solutions (e.g. in the field of global optimization).

For these tasks in most cases forward and backward error analysis is unsatisfactory (sometimes even not possible). One idea is to use perturbation analysis 'on the fly' to derive sharp bounds for the perturbation using the finite precision solution of a given problem. This can be done by means of interval arithmetic.

Interval arithmetic can be implemented in a reliable way (the result of an interval arithmetic computation is guaranteed to include the true solution). References to interval arithmetic include [1, 2, 3, 10, 11, 12, 13, 14, 15].

There exist many implementations of interval arithmetic. Some are extensions of existing languages [4, 5, 7], but even new languages including interval arithmetic have been proposed, see for instance [6]. A very fast implementation of interval arithmetic is given with the ANSI C library BIAS by Knüppel [9]. For a complete introduction to BIAS see [8].

On the other hand, MATLAB is the standard tool for interactive floating point computations. To our knowledge there is only one interval package for MATLAB, namely INTLAB [18]. This toolbox provides a variety of interval functionalities and is, except one routine for switching the rounding mode, entirely written in MATLAB.

Nevertheless we found it useful to provide with `b4m` an interface to PROFIL/BIAS routines. This enables the user to develop algorithms in the convenient MATLAB environment and to move easily to C++ for computing intensive computations. The striking advantage of this approach is that the fast development kit in `b4m` and the fast executable C++ programs in PROFIL/BIAS deliver bitwise identical results.

2 The toolbox b4m

The toolbox `b4m` has been designed

- to use the ANSI C interval arithmetic library BIAS in an interactive way

and

- to add interval arithmetic operations and inclusion algorithms to the standard floating point environment Matlab.

The toolbox `b4m` consists of a new datatype `interval` and of arithmetical operations between arguments of type `double` and `interval`. These elementary operations are performed using BIAS (Basic Interval Arithmetic Subroutines) as proposed and implemented by Knüppel in 1993. Standard functions are part of BIAS and are accessable through `b4m`. Currently standard functions are calculated in floating point and some bits are subtracted/added for infimum/supremum (no rigorous verification). BIAS does not support the use of sparse and complex operations. Therfore, up to now no operations involving arguments of type `sparse` and/or `complex` are possible. Of course, it is possible to extend `b4m` to perform computations in the complex field and to perform sparse computations.

The toolbox `b4m` tries to combine the efficiency of BIAS and the ease of use of Matlab. All interval data are stored in BIAS format, so the computing time splits roughly into the time Matlab needs for the interpretation of the code and the time BIAS needs to perform the operations.

The interface between BIAS and Matlab is implemented Mex-files. Those are stored hidden from the user in private subdirectories. The user interface consists of ordinary M-files with inline help, documentation and error handling. Every M-file calls appropriate Mex-file(s). It is tried to achieve stability and robustness, so many operations could be improved by omitting the error handling. However, this seems to be contrary to the goals of Matlab.

The installation is simple. The whole package is distributed as one packed file which has to be unpacked (Windows) or, additionally the Mex-files have to be compiled using `make` (Unix). It is easier to install the package with precompiled Mex-files because this version does not build upon a previously installed version of BIAS. However, the precompiled version(s) make no use of the underlying architecture and are sometimes much slower compared to a version compiled on the given machine.

The toolbox `b4m` has been implemented by the author in 1998. It is distributed under the TI3 copyright license and can be copied freely. For details see the file

`../matlab/toolbox/b4m/COPYING`

in the `b4m` distribution.

3 Installation of b4m

Currently, two versions of the `b4m` package are available. The original package consists of the source code, makefiles and documentation. It has to be compiled for the given architecture. This presumes the existence of a compiled version of BIAS, and a C compiler that is supported by MATLAB to compile the Mex-files.

The other version of the package includes the source code, makefiles, documentation and precompiled Mex-files. This version is machine dependent. Up to now, a precompiled version for PC running under Windows can be downloaded. This version has to be unpacked (e.g.) in the directory

```
C:\Matlab\Toolbox
```

and will produce a new directory

```
C:\Matlab\Toolbox\b4m
```

including the new toolbox. The path to `b4m` has to be added to the MATLAB search path.

The non-compiled version needs BIAS which has to be compiled with the flags in the file `BiasInt.h` set as indicated in Figure 1.

```
/************************************************************************
*  User definable controls
*************************************************************************/

#undef  __BIASRAISEDIVIDEBYZERO__       /* if defined, division by zero aborts;
                                           otherwise, bounds containing +/-oo
                                           or NaN are computed. */
#define __BIASSETROUNDTONEAREST__       /* if defined, rounding mode is set to
                                           nearest after each operation */
```

Figure 1: Setting of controls in `BiasInt.h`

Setting

```
__BIASRAISEDIVIDEBYZERO__,
```

a division by zero or a similar operation may cause MATLAB or even your whole system to crash. With

```
__BIASSETROUNDTONEAREST__
```

set, extensive floating point computations may be perturbed and be less reliable.

After BIAS has been compiled, the `b4m` package has to be unpacked, e.g. in the directory

```
../matlab/toolbox/
```

and will produce a directory

```
../matlab/toolbox/b4m/
```

with subdirectories

```
../matlab/toolbox/b4m/private/
../matlab/toolbox/b4m/docu/
../matlab/toolbox/b4m/@interval/
../matlab/toolbox/b4m/@interval/private/
```

containing the source code and the documentation. The lines

```
LIBDIR     = /home/ti3/jz/lib  # the directory where you have installed BIAS
                               # BIAS can be downloaded at
                               # http://www.ti3.tu-harburg.de
DLL        = mexrs6
INCLUDEDIR = /home/ti3/jz/include  # the directory with the include files
                                   # Bias0.h Bias1.h Bias2.h BiasF.h ...
```

in the Makefiles in the `private/` directories have to be modified according to your system. Afterwards the Mex-files can be compiled using `make`.

If another directory than

```
../matlab/toolbox/
```

has been chosen, the path to `b4m` has to be added to the search path of MATLAB.

To perform a check whether `b4m` is installed properly just try some of the examples given in this documentation.

4 The class interval

The new datatype (class) `interval` in MATLAB consists of a directory

```
../matlab/toolbox/b4m/@interval/
```

(the class directory) outside the search path (VAX/VMS uses `$` instead of `@`).

Inside the class directory all overloaded functions and operations, the so-called methods of the class `interval`, are stored as a collection of M-files. The most important is the constructor

```
../matlab/toolbox/b4m/@interval/interval.m
```

which is invoked every time a new instance (object) of the class has to be constructed from previous defined ones, and is

```
../matlab/toolbox/b4m/@interval/display.m
```

which serves as command window display.

`@interval/`	class directory
`@interval/interval.m`	class constructor
`@interval/display.m`	command window display

Figure 2: The class `interval` in `b4m`

In `b4m` the constructor can be called with one argument

```
a = rand(10);
y = interval(a);
```

or two arguments

```
a = zeros(10,2); b = ones(10,2);
y = interval(a,b);
```

of type `double`. The floating point input is taken as desired quantity. That means, the interval `interval(0.1)` doesn't contain the true mathematical value $\frac{1}{10}$. The types `complex`

```
>> a = interval(sqrt(-1))
??? Error using ==> interval/interval
Complex intervals are not supported.
```

and `sparse` are currently not supported. Invoked with two arguments, the constructor handles the first one as infimum and the second one as supremum

```
>> a = interval([1 3],[2 4])
a =
[1,2][3,4]
```

and displays a matrix of equal dimensions with brackets around each entry.

If the dimensions of the input arguments mismatch

```
>> a = interval(zeros(2,3),ones(2,4))
??? Error using ==> interval/interval
Upper and lower bound must have the same size.
```

an error message is returned.

If there is an entry, whose infimum is greater than the supremum

```
>> a = interval(1,-1)
??? Error using ==> interval/interval
Upper bound must be greater than lower bound.
```

another error message is returned.

Intervals including NaN's, plus/minus infinity and the empty set

```
>> 1/interval(-1,1)
ans =
[-Inf,Inf]
>> interval(-1,1)/interval(-1,1)
ans =
[NaN,NaN]
>> interval([])
ans =
{}
```

are part of b4m.

Matrices of type interval are stored in b4m as the transpose of corresponding BIAS matrices, and both the constructor and the command window display depend very much on the storage. Therefore these files should not be altered by the user.

5 Overloaded arithmetical operators

Arithmetical operators which are part of MATLAB for the datatype double have been overloaded for the type interval. The behaviour of the routines for the datatype interval is merely the same as the behaviour of the appropriate routine for the datatype double, at least on the logical and error handling level. The implementation issues differ in some cases very much. Some aspects included in MATLAB for arguments of type double are not yet implemented or will never be implemented for arguments of type interval.

$y = a + b$	(almost) similar to matlab routine
$y = a - b$	(almost) similar to matlab routine
$y = a * b$	similar to matlab routine
$y = a/b$	implemented only for scalar b
$y = a\backslash b$	verifies a floating point solution with the Krawczyk-operator, may fail
$y = a^n$	only implemented for scalar arguments
$y = x'$	similar to matlab routine
$y = a.*b$	similar to matlab routine
$y = a./b$	similar to matlab routine
$y = a.\backslash b$	similar to matlab routine
$y = a.^n$	(almost) similar to matlab routine
$y = x.'$	similar to matlab routine
$y = -a$	similar to matlab routine
$y = +a$	similar to matlab routine

Figure 3: The arithmetic operations in b4m

In all figures the lower latin letters $a, b, c\ldots$ can be either of type double or interval unless explicitly indicated otherwise.

The operations plus and minus behave like the appropriate MATLAB routine.

```
>> interval(pi)+3
ans =
[6.141592653589793,6.141592653589793]
>> a = interval(ones(2,1)); b = interval(ones(3,2));
>> a+b
??? Error using ==> interval/plus
Matrix dimensions must agree.
```

Only the sum (subtraction) between a scalar and a vector (matrix) is not implemented. This is not a real disadvantage, because

```
>> x = interval(rand(2,3));
>> z = x + interval(-1,1)*ones(size(x));
```

may be used instead.

The matrix multiplication

```
>> A = interval(rand(10)); R = inv(mid(A));
>> C = eye(size(A))-R*A;
```

and the pointwise multiplication

```
>> A.*R(1:9,1:9)
??? Error using ==> interval/times
Matrix dimensions must agree.
```

involving arguments of type `interval` have all features of the appropriate multiplications involving arguments of type `double`.

For the matrix right division this is no longer true. Only division by a scalar argument is implemented, because the code is more readable and faster for the other operations including elementwise division from the left and from the right.

The matrix left divide is based on an inclusion algorithm

```
>> A = interval(rand(10)); b = ones(10,1);
>> x = A\b;
Inclusion found in one step.
>>
```

using algorithms described in [17] which in turn are based on the Krawczyk-operator. The precision of the solution

```
>> norm(diam(x),inf)
ans =
     7.993605777301127e-15
```

depends mainly on the condition number

```
>> cond(mid(A))
ans =
     4.795779576449495e+01
```

of the matrix and on the machine precision $2^{-53} \approx 10^{-16}$.

This iteration may fail to produce a result if the system is too ill-conditioned or the system matrix is an interval matrix that is not strongly regular. The matrix left divide is also implemented for the least squares problem [16]

```
>> A = rand(3,2); b = ones(3,1);
>> interval(A)\b
Inclusion found in one step.
ans =
[1.258771069839872,1.258771069839872]
[0.8784383329133052,0.8784383329133061]
```

and can compute the minimal 2-norm solution.

```
>> interval(A')\ones(2,1)
Inclusion found in one step.
ans =
[0.4942760521137817,0.4942760521137827]
[0.9507394470495564,0.9507394470495574]
[0.6921939035898382,0.6921939035898388]
>> norm(ans)
ans =
[1.275675002512873,1.275675002512876]
>> norm(A'\ones(2,1))
ans =
     1.538334371494617e+00
```

The matrix power operation has been implemented for scalar arguments. The pointwise power operation is similar to the MATLAB routine for `double` arguments, with the exception, that if called with two arguments of type `interval`, the base interval must be greater than zero, and if called with exponent of type `double`, the exponent must be an integer.

Complex transpose and transpose are the same (nonexistence of complex intervals) and generate output similar to the MATLAB routine for `double` argument. The same holds for the unary operations (unary plus and unary minus).

6 Overloaded functions with one argument

Most mathematical standard functions have been implemented and behave in the same manner as the MATLAB routines for `double` argument. This is true except that `b4m` routines invoked with an argument such that the return argument would be complex generate an error message. All operations are performed elementwise.

The routine `abs.m` is the only overloaded function with return argument of type `double`. It computes the elementwise magnitude of the given argument of type `interval`. The magnitude is defined as

$$\mathrm{mag}([x]) = \max\{|x| : x \in [x]\}.$$

See the routine `iabs.m` for the computation of the interval of absolute values.

Each overloaded function can be used with scalar

$y = \mathrm{abs}(x)$	computes the magnitude
$y = \mathrm{acos}(x)$	similar to matlab routine
$y = \mathrm{acosh}(x)$	similar to matlab routine
$y = \mathrm{acot}(x)$	similar to matlab routine
$y = \mathrm{acoth}(x)$	similar to matlab routine
$y = \mathrm{asin}(x)$	similar to matlab routine
$y = \mathrm{asinh}(x)$	similar to matlab routine
$y = \mathrm{atan}(x)$	similar to matlab routine
$y = \mathrm{atanh}(x)$	similar to matlab routine
$y = \cos(x)$	similar to matlab routine
$y = \cosh(x)$	similar to matlab routine
$y = \cot(x)$	similar to matlab routine
$y = \coth(x)$	similar to matlab routine
$y = \exp(x)$	similar to matlab routine
$y = \log(x)$	similar to matlab routine
$y = \mathrm{log10}(x)$	similar to matlab routine
$y = \sin(x)$	similar to matlab routine
$y = \sinh(x)$	similar to matlab routine
$y = \mathrm{sqrt}(x)$	similar to matlab routine
$y = \tan(x)$	similar to matlab routine
$y = \tanh(x)$	similar to matlab routine

Figure 4: Overloaded functions in `b4m`

```
>> cos(interval(1))
ans =
[0.5403023058681388,0.5403023058681408]
```

or matrix (vector) argument

```
>> A = interval([0 pi;pi 0])
A =
[0,0][3.141592653589793,3.141592653589793]
[3.141592653589793,3.141592653589793][0,0]
>> cos(A)
ans =
[0.9999999999999996,1][-1,-0.9999999999999996]
[-1,-0.9999999999999996][0.9999999999999996,1]
```

and behave like their counterparts for `double` arguments.

It is possible to compute the range of a function for arguments involving $\pm\infty$

```
>> cos(interval(-inf,inf))
ans =
[-1,1]
```

with the correct result. Also the return value of a range evaluation

```
>> tan(interval(0,pi))
ans =
[-Inf,Inf]
```

can yield a result involving $\pm\infty$.

7 Overloaded comparison operations

Right now the comparison operators are used for set comparisons, that means a call to `gt.m`

```
>> a = interval(-1,1); b = interval(-0.5,0);
>> a > b
ans =
     1
```

checks entrywise whether the first argument includes the second argument or not.

`a == b`	set equality
`a ~= b`	set non-equality
`a <= b`	$a \subseteq b$
`a < b`	$a \subset b$
`a >= b`	$b \subseteq a$
`a > b`	$b \subset a$

Figure 5: Overloaded comparison operators in `b4m`

The partial ordering of interval vectors (matrices) was of main interest in the design of BIAS, and for this reason the return argument in every case

```
>> A = interval(rand(2));b = ones(2,1);
>> mid(A)\b < A\b
Inclusion found in one step.
ans =
     1
>> eye(2) <= interval([1 0;0 1],[2 0;0 1])
ans =
     1
```

is a scalar.

8 Miscellaneous overloaded operations

Many other operations on `double` arguments are implemented in `b4m` for arguments of type `interval`. Among them are operations working on matrices

```
>> inv(interval([1 2;3 4]))
Inclusion found in one step.
ans =
[-2.000000000000001,-1.999999999999999][0.9999999999999997,1]
[1.5,1.5][-0.5000000000000002,-0.4999999999999998]
>> A = interval(floor(rand(3)*10)); trace(A)
ans =
[13,13]
```

and/or vectors

```
>> A = diag(interval([1 2 3 4]))
A =
[1,1][0,0][0,0][0,0]
[0,0][2,2][0,0][0,0]
[0,0][0,0][3,3][0,0]
[0,0][0,0][0,0][4,4]
```

of type `interval`. The horizontal

```
>> a = interval(2); b = interval(3,4);
>> A = [a b a]
A =
[2,2][3,4][2,2]
```

and vertical

```
>> B = [a;b;a]
B =
[2,2]
[3,4]
[2,2]
>> C = [A;A]
C =
[2,2][3,4][2,2]
[2,2][3,4][2,2]
```

concatenation are also implemented for type `interval`.

The subscripted reference

```
>> A = interval(ceil(rand(3)*100))
A =
[55,55][38,38][60,60]
[16,16][87,87][50,50]
```

diag	similar to matlab routine
horzcat	similar to matlab routine
inv	only for quadratic matrices
isempty	tests for the empty set
length	similar to matlab routine
max	similar to matlab routine
min	similar to matlab routine
norm	some differences
prod	similar to matlab routine
size	similar to matlab routine
subsasgn	causes trouble with version 5.2
subsref	causes trouble with version 5.2
sum	similar to matlab routine
trace	similar to matlab routine

Figure 6: Miscellaneous overloaded operations in `b4m`

```
[70,70][86,86][90,90]
>> A(2,2)
ans =
[87,87]
>> A(1:2,1:2).inf
ans =
    55    38
    16    87
```

and the subscripted assignment

```
>> A(3) = 200
A =
[55,55][38,38][60,60]
[16,16][87,87][50,50]
[200,200][86,86][90,90]
```

work well as long as they are invoked from the command window display or from an M-file in a directory other than the class directory `@interval/`. In MATLAB 5.2 it is impossible to call this two methods from M-files in the class directory.

The size operator had to be overloaded because of the storing conventions in BIAS.

The sum, prod, max and min operators act on data of type `interval` like their counterparts for `double`.

The inclusion of the norm of a matrix

```
>> A = interval(rand(2,3));
>> norm(A,1)
ans =
```

```
[1.410303816183485,1.410303816183485]
```

is implemented for the Frobenius, 1 and ∞-norm. The inclusion of the norm of a vector

```
>> norm(interval([3 4]))
ans =
[4.999999999999997,5.000000000000003]
```

is implemented for all p-norms with $p \in \mathbb{N} \cup \infty$.

9 Interval specific operations

There are relational operators between intervals and functions with argument of type `interval` which have no counterpart in floating point arithmetic.

diam	computes the diameter
iabs	computes interval of absolute values
in	same as le
in0	same as lt
inf	returns the infimum
infsup	returns infimum and supremum
in_interior	same as lt
intersection	computes intersection of intervals
lss	(l)inear (s)ystem (s)olver (Rump)
mid	computes the midpoint
midrad	computes midpoint and radius
pred	computes the predecessor
rad	computes the radius
sqr	computes square of interval
succ	computes the successor
sup	returns the supremum
TeX	computes string representation

Figure 7: Interval specific operations in `b4m`

Among them are the set comparisons,

```
>> a = interval(-1,1); b = interval(-2,1);
>> a<=b, in(a,b)
ans =
     1
ans =
     1
```

which are also implemented as overloaded comparison operators. Other set operations like the intersection of two intervals

```
>> intersection(interval(-2,2),interval(0,inf))
ans =
[0,2]
```

and predecessor

```
>> pred(interval(0,1))
ans =
[4.940656458412465e-324,0.9999999999999999]
```

and successor

```
>> succ(ans)
ans =
[-0,1]
```

for data of type `interval` are also implemented.

Real quantities describing the quality of an inclusion are the diameter

```
>> A = interval([1 2;3 4]); b = ones(2,1);
>> diam(A\b)
Inclusion found in one step.
ans =
     7.771561172376096e-16
     4.440892098500626e-16
```

and the radius. The midpoint

```
>> A = interval([1 2;3 4]); b = ones(2,1);
>> x = mid(A\b)
Inclusion found in one step.
x =
    -1
     1
>> A*x
ans =
[1,1]
[1,1]
```

of an interval can be of interest as starting point for an iterative method to obtain a new approximate (floating point) solution. The supremum and the infimum

```
>> inf(A*x) == sup(A*x)
ans =
     1
     1
```

return the lower and upper bound of the interval.

The routine iabs computes the elementwise interval of absolute values, and the routine sqr computes the elementwise square.

The routine TeX

```
>> A = interval(rand(3)); b = ones(3,1)/3;
>> x = A\b; A*x
Inclusion found in one step.
ans =
[0.3333333333333319,0.3333333333333346]
[0.3333333333333324,0.3333333333333343]
[0.3333333333333323,0.3333333333333343]
>> TeX(ans)
ans =
\begin{pmatrix}
[0.33333333333333_{19}^{46}] \cr
[0.33333333333333_{24}^{43}] \cr
[0.33333333333333_{23}^{43}]
\end{pmatrix}
>>
```

returns a character string which can be used (e.g. via copy and paste)

$$\begin{pmatrix}[0.33333333333333_{19}^{46}] \\ [0.33333333333333_{24}^{43}] \\ [0.33333333333333_{23}^{43}]\end{pmatrix}$$

to represent intervals in a TEX or LATEX document.

10 Real operations in b4m

To compute the inclusion of the result of an operation with arguments of type `double` it is not necessary to perform a typecast. The four basic arithmetic operations have been implemented as functions of two arguments of type `double`

```
>> div(1,3)
ans =
[0.3333333333333333,0.3333333333333334]
```

with a result of type `interval`. All routines in Figure 8 indicated as dummy routine return the same result as the appropriate `interval` routine invoked with degenerate `interval` argument. The routine midrad for example

```
>> A = rand(2)
A =
     6.191919017459770e-01     3.357822685065588e-01
     7.837094139316327e-01     4.321648717393279e-01
>> [m,r] = midrad(A)
m =
     6.191919017459770e-01     3.357822685065588e-01
     7.837094139316327e-01     4.321648717393279e-01
r =
```

add	computes an inclusion of the sum
diam	dummy routine
div	computes an inclusion of the division
epsilon	same as eps
eta	smallest machine number
lss	(l)inear (s)ystem (s)olver
mid	dummy routine
midrad	dummy routine
mul	computes an inclusion of the multiplication
pred	computes the predecessor
rad	dummy routine
succ	computes the successor
sup	dummy routine

Figure 8: Real operations in `b4m`

```
     0      0
     0      0
```

returns the matrix itself as midpoint and a zero matrix of equal dimensions.

The predecessor

```
>> pred(0)
ans =
   -4.940656458412465e-324
```

of a floating point number x is defined as the largest floating point number $\mathrm{pred}(x)$ with $\mathrm{pred}(x) < x$, the successor

```
>> succ(1)-1
ans =
     2.220446049250313e-16
```

of a floating point number x as the smallest floating point number $\mathrm{succ}(x)$ with $\mathrm{succ}(x) > x$.

The routines eta and epsilon

```
>> eta, epsilon
ans =
    4.940656458412465e-324
ans =
     2.220446049250313e-16
```

compute the smallest positive machine number and the machine precision, respectively.

11 Documentation and help

Apart from this documentation

```
../matlab/toolbox/b4m/docu/b4m.ps
```

every M-file contains its own short documentation. This can be invoked by a call to help

```
>> help interval/mid

 MID (interval) computes the midpoint of a given interval matrix.

 b4m - datatype interval    Version 1.02
                            (c) 26.2.1998 Jens Zemke

    DESCRIPTION:
      'mid' is called

          midpoint = mid(a)

      and computes the midpoint of a given
      interval matrix a.
      The midpoint is defined as
      mid(a) := (inf(a) + sup(a)) / 2.

      The operations on the datatype interval
      are based on BIAS by Olaf Knueppel.

    SEE ALSO:
      interval: rad, midrad, diam.
      double: mid.
```

or by a call to

```
>> helpdesk b4m
```

helpdesk. Currently no help in html format (helpwin) is available and it is not possible to use the 'see also' button. Future editions will support 'see also' and include html pages. All first lines are one line documentations to support lookfor

```
>> lookfor interval
INTERVAL class constructor for the class interval.
DISPLAY (interval) command window display of an interval matrix.
PLUS (interval) overloades plus for interval matrices. a+b.
MINUS (interval) overloades minus for interval matrices, a-b.
MTIMES (interval) overloades mtimes for interval matrices, a*b.
INF (interval) computes the infimum of a given interval matrix.
```

```
SUP (interval) computes the supremum of a given interval matrix.
INFSUP (interval) conversion to double from interval.
VERTCAT (interval) vertical concatenation for interval matrices.
ABS (interval) computes the magnitude of a given interval matrix.
ACOS (interval) implements acos for intervals.
ACOSH (interval) implements acosh for intervals.
ACOT (interval) implements acot for intervals.
ACOTH (interval) implements acoth for intervals.
```

in the search for keywords. In `b4m` every M-file contains the keyword interval.

Actual information about `b4m` will be placed on the internet at

`http://www.ti3.tu-harburg.de/zemke/b4m/index.html`

and

`http://www.ti3.tu-harburg.de/zemke/b4m/docu.html`

For discussions and remarks concerning BIAS/PROFIL, `b4m`, INTLAB, COX and TPX, a mailing list

`validated_numerics@tu-harburg.de`

has been set up. To subscribe send a mail to

`ti3@tu-harburg.de`

with the text

`subscribe validated_numerics`

inside, to unsubscribe send a mail with the text

`unsubscribe validated_numerics`

inside.

12 Download BIAS and b4m

PROFIL/BIAS can be downloaded at

`ftp://ti3sun.ti3.tu-harburg.de`

in the directory

`/pub/profil/`

or from the page

`http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html`

and `b4m` can be found at

`http://www.ti3.tu-harburg.de/zemke/b4m/index.html` .

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.

[2] E.R. Hansen. *Global Optimization using Interval Analysis.* Marcel Dekker, New York, 1992.

[3] R.B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publisher, Dordrecht, 1996.

[4] R.B. Kearfott, M. Dawande, K. Du, and C. Hu. INTLIB: A portable Fortran-77 elementary function library. *Interval Comput.*, 3(5):96–105, 1992.

[5] R.B. Kearfott, M. Dawande, and C. Hu. INTLIB: A portable Fortran-77 interval standard function library. *ACM Trans. Math. Software*, 20:447–459, 1994.

[6] R. Klatte, U. Kulisch, M. Neaga, D. Ratz, and Ch. Ullrich. *PASCAL-XSC: Language reference with examples.* Springer, 1992.

[7] R. Klatte, U. Kulisch, A. Wiethoff, C. Lawo, and M. Rauch. *C-XSC A C++ Class Library for Extended Scientific Computing.* Springer, Berlin, 1993.

[8] O. Knüppel. BIAS — Basic Interval Arithmetic Subroutines. Technical Report 93.3, Berichte des Forschungsschwerpunktes Informations- und Kommunikationstechnik, Inst. f. Informatik III, TU Hamburg-Harburg, 1993.

[9] O. Knüppel. PROFIL / BIAS — A Fast Interval Library. *Computing*, 53:277–287, 1994.

[10] U. Kulisch and W.L. Miranker. *Computer Arithmetic in Theory and Practice.* Academic Press, New York, 1981.

[11] U. Kulisch and W.L. Miranker, editors. *A New Approach to Scientific Computation.* Notes and Reports in Computer Science and Applied Mathematics. Academic Press, 1983.

[12] R.E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1966.

[13] R.E. Moore. *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, 1979.

[14] R.E. Moore (ed.). *Reliability in Computing.* Academic Press, San Diego, 1988.

[15] A. Neumaier. *Interval Methods for Systems of Equations.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1990.

[16] S.M. Rump. Solving Algebraic Problems with High Accuracy. Habilitationsschrift. In U.W. Kulisch and W.L. Miranker, editors, *A New Approach to Scientific Computation*, pages 51–120. Academic Press, New York, 1983.

[17] S.M. Rump. Verification Methods for Dense and Sparse Systems of Equations. In J. Herzberger, editor, *Topics in Validated Computations — Studies in Computational Mathematics*, pages 63–136, Elsevier, Amsterdam, 1994.

[18] S.M. Rump. INTLAB - INTerval LABoratory. submitted for publication, 1999.