A Computational Environment for Interval Matrices in C++

Michael Nooner and Chenyi Hu*
Computer Science Department, University of Central Arkansas

Abstract. It is often required to manipulate interval matrices in reliable scientific computing. A portable computational environment for basic interval linear algebra subroutines (interval BLAS) is needed. We report such a environment recently developed based on the interval BLAS standard included in (Blackford and et al, 2001) and (Dongarra, J. et al, 2002).

The computational environment is object-oriented in ISO/ANSI standard C++. It consists of arithmetic, fundamental and utility functions, and set operations among intervals, interval vectors, and interval matrices. These operations are implemented as member functions of three classes: *Interval, IntervalVector*, and *IntervalMatrix*. This package is portable and robust with built-in error handling features. Instructions on package installation, testing, and usages are included. A sample application program is attached as an appendix.

Keywords: Interval arithmetic, Interval BLAS, C++

1. Introduction

Ever since Ray Moore introduced interval arithmetic (Moore, 1979) in 1950's, it has achieved numerous successful applications in scientific computing (Hansen, 1992; Coliss and Kearfott, 1999; de Korvin and Hu, 2004) and many of others. Similar to traditional floating point arithmetic, interval linear algebra is fundamental to most calculations and often the computationally intense part in applications of interval computing. Designers of computer programs involving linear algebraic operations have frequently chosen to implement certain low level operations, such as the dot product or the matrix vector product, as separate subprograms. This may be observed both in many published codes and in codes written for specific applications at many computer installations (Kågström, 1998; Duff and Vömel, 2002; Duff and Heroux, 2002; Li and et al., 2002; Sun Studio, 2005). With the same motivation we develop the computing environment for interval matrices.

2. The Interval BLAS Standard

In the Basic Linear Algebra Subprograms Technical (BLAST) Forum (Blackford and et al, 2001), we proposed an interval BLAS standard. Based on the standard, Sun Microsystems made its Fortran 95 implementation in its Sun Studio (Sun Studio, 2005; Walster, 2002). This software package is mostly based on the interval BLAS standard but implemented in C++. We need to briefly review some basic definitions of the standard here.

* Partially supported by NSF/CISE/CCR grant 0202042

A nonempty *mathematical interval* $[a, b]$ is the set $\{x \in \Re | a \leq x \leq b\}$ where $a \leq b$. A *machine interval* $[a^*, b^*]$ is a mathematical interval whose endpoints are machine representable numbers. We say that $[a^*, b^*]$ is a machine representation of $[a, b]$ if $[a^*, b^*]$ *contains* $[a, b]$ i.e. $a^* \leq a$ and $b \leq b^*$. We say that the machine interval $[a^*, b^*]$ is a *tight representation* of a mathematical interval $[a, b]$ if and only if a^* is the greatest machine representable number which is less than or equal to a, and b^* is the least machine representable number which is greater than or equal to b. The *empty interval* $\emptyset$, which does not contain any real number, is required in interval BLAS. In our implementation, we use [1, -1] to represent the empty interval.

Interval vectors and *interval matrices* are vectors and matrices whose entries are intervals. Both scalar (floating point number) and interval arguments are used for the specifications of routines in this paper. We use **boldface letters** to specify interval arguments. We also use overline and underline to specify the greatest lower bound and the least upper bound of an interval variable, respectively. For example, if $\mathbf{x}$ is an interval vector, then $\mathbf{x} = [\underline{x}, \overline{x}]$.

Interval arithmetic on mathematical intervals is defined as follows.

Let $\mathbf{a}$ *and* $\mathbf{b}$ *be two mathematical intervals. Let* `op` *be one of the arithmetic operations* $+, -, \times, \div$. *Then* $\mathbf{a}$ `op` $\mathbf{b} \equiv \{a$ `op` $b : a \in \mathbf{a}, b \in \mathbf{b}\}$, *provided that* $0 \notin \mathbf{b}$ *if* `op` *represents* $\div$.

Table I gives explicit definitions of these four basic interval arithmetic operations and other operations on mathematical intervals used in this package. All operations inside a computer are performed on machine intervals. Arithmetic on machine intervals must satisfy the following property:

Containment Condition: Let $\mathbf{a} = [\underline{a}, \overline{a}]$ and $\mathbf{b} = [\underline{b}, \overline{b}]$ be intervals. Let $\mathbf{c} = [\underline{c}, \overline{c}]$ be the interval result of computing $\mathbf{a}$ `op` $\mathbf{b}$ where `op` is defined in Table I. If $\mathbf{c}$ is nonempty, then $\mathbf{c}$ must contain the exact mathematical interval $\mathbf{a}$ `op` $\mathbf{b}$.

In other words, interval arithmetic on nonempty machine intervals requires that we round down the lower bound and round up the upper bound to guarantee that the machine interval result contains the true mathematical interval result. This is needed to propagate guaranteed error bounds. To ensure our implementation satisfies the containment condition, we make use of the constants DBL_EPSILON and DBL_MIN provided in $< cfloat >$ within the ISO/ANSI standard C++. As the default, our package uses double precision. Our testing shows that the rounding process we used effects the sixteenth to eighteenth significant digits on machines with 64 bits representation for the type double.

3. Functionality

This section reports the functionality (mathematical operations) and operators involving interval vectors and interval matrices. We group the functionalities into two tables. Table II lists the functionalities involving interval vectors. It includes basic algebraic operations, set operations, interval matrix-vector operations, and utility operations (including data movement) for interval vectors. Table III lists functionalities for interval matrix operations that include $O(n^2)$ and $O(n^3)$ algebraic operations, set operations, and utility operations (including data movement) for interval matrices. The LaTeXputs the tables at the end of this paper.

Table I. Elementary interval operations

Operation	$\mathbf{a} \neq \emptyset$ and $\mathbf{b} \neq \emptyset$	Operator
Addition $\mathbf{a} + \mathbf{b}$	$[\underline{a} + \underline{b}, \overline{a} + \overline{b}]$	+
Subtraction $\mathbf{a} - \mathbf{b}$	$[\underline{a} - \overline{b}, \overline{a} - \underline{b}]$	-
Multiplication $\mathbf{a} * \mathbf{b}$	$[\min\{\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}\}, \max\{\underline{a}\underline{b}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}\}]$	*
Division $\frac{\mathbf{a}}{\mathbf{b}}, (0 \notin \mathbf{b})$	$[\min\{\underline{a}/\underline{b}, \underline{a}/\overline{b}, \overline{a}/\underline{b}, \overline{a}/\overline{b}\}, \max\{\underline{a}/\underline{b}, \underline{a}/\overline{b}, \overline{a}/\underline{b}, \overline{a}/\overline{b}\}]$	/
Intersection $\mathbf{a} \cap \mathbf{b}$	$[\max\{\underline{a}, \underline{b}\}, \min\{\overline{a}, \overline{b}\}]$ if $\max\{\underline{a}, \underline{b}\} \leq \min\{\overline{a}, \overline{b}\}$; Otherwise, $\emptyset$	&
Union $\mathbf{a} \cup \mathbf{b}$	$[\min\{\underline{a}, \underline{b}\}, \max\{\overline{a}, \overline{b}\}]$ if $\mathbf{a}$ and $\mathbf{b}$ are not disjoint;	|
Cancellation $\mathbf{a} \ominus \mathbf{b}$	$[\underline{a} - \underline{b}, \overline{a} - \overline{b}]$ if $(\underline{a} - \underline{b}) \leq (\overline{a} - \overline{b})$; Otherwise, $\emptyset$	cancel()
Convex hull $\mathbf{a}, \mathbf{b}$	$[\min\{\underline{a}, \underline{b}\}, \max\{\overline{a}, \overline{b}\}]$	hull()
Square root	$\sqrt{\mathbf{a}}$	isqrt()
Exponential	$e^{\mathbf{a}}$	iexp()
Logarithm	$log_{10}\mathbf{a}$	ilog()
Power	$\mathbf{a}^{\mathbf{b}}$	ipow()
Absolute value $\|\mathbf{a}\|$	$\max\{\|\underline{a}\|, \|\overline{a}\|\}$	iabs()
Trig functions		isin() icos() itan() iasin() iacos() iatan()
Disjoint test	*True* if $\mathbf{a} \cap \mathbf{b} = \emptyset$; *False*, otherwise	disjoint()
Enclosure test	*True* if $\underline{a} \leq \underline{b}$ and $\overline{b} \leq \overline{a}$; *False*, otherwise	encloses()
Interior test	*True* if $\underline{a} < \underline{b}$ and $\overline{b} < \overline{a}$; *False*, otherwise	interior()
Midpoint $\mathbf{a}$	$(\underline{a} + \overline{a})/2$	midpoint()
Width $\mathbf{a}$	$\overline{a} - \underline{a}$	width()
Assignment	$\mathbf{a} \leftarrow \mathbf{b}$	=
Insertion operator	for output	<<
Extraction operator	for input	>>
Equality test	*True* if $\underline{a} \equiv \underline{b}$ and $\overline{b} \equiv \overline{a}$; *False*, otherwise	==

4. Package Contents

This package contains three main classes (*Interval, IntervalVector,* and *IntervalMatrix*), and auxiliary classes such as (*IntervalVectorT*, *IntervalMatrixT,* and *INTERVAL_EXCEPTION*) for error handling and performance enhancement.

4.1. The *Interval* Class

This class implements intervals and the fundamental operations among them. It is required by operations among the interval vectors and matrices. The private data members of the *Interval* class consists of the lower and upper bounds of a real interval with two double precision variables and a string for error handling. There are three built in constructors for the *Interval* class. They initiate

an interval object with zero to two double precision floating-point parameters (the interval [0, 0] for zero parameter as default; directed rounding for single and double parameters).

The lower and upper bounds are accessed through the [ ] operator with index 1 or 2 respectively. For example, $x[1]$ and $x[2]$ return the lower and upper bounds of an *Interval* object x. Arithmetic operations of *Interval* objects are overloaded as arithmetic operators $(+, -, *, /)$. Interval fundamental functions (Hu and Kearfott, 1993; Kearfott et al, 1994) are defined as friend functions (*isqrt, iexp, isin, icos, iatan,* etc). The insertion and extraction operators in C++ are also overloaded for the class. Table I lists the operators of this class.

4.2. The *IntervalVector* and *IntervalMatrix* classes

An instance of *IntervalVector*, as per the name, is a vector whose elements are *Interval* objects. The *IntervalVector* class contains two private data members: an pointer of *Interval* type and an integer that indicates the dimension of an instance. The vector's dimension is set when the vector is created and it is not resizable.

An *IntervalMatrix* object, as per the name, is a matrix whose elements are *Interval* objects. As with the *IntervalVector* and *Interval* classes, the standard arithmetic operators are overloaded. The dimensions of an interval matrix are set when it is created with the *IntervalMatrix* constructors and is not resizable.

4.3. Error Handling and Auxiliary Classes

Error handling is done through exceptions. When an error occurs an *INTERVAL_EXCEPTION* is thrown. This *INTERVAL_EXCEPTION* type is a structure with two fields. The first is a numeric code that corresponds to the generated error type that allows for ease of programmatic error handling. The other field is a string message that corresponds to that error. The last error can be retrieved by calling the static *getLastError()* method of the *Interval* class. Similarly, the global error can be set using *Interval*'s static method *setError()*.

The *setError()* method takes two arguments. The first is the numeric code of the error that is to be set. There are twenty-six predefined error codes. Each of them has a default error message. If you send *setError()* an unknown code with no message, the global error's message is set to "Unknown Error!" A complete listing of all defined error codes and their associated messages can be found in the library's documentation for *INTERVAL_EXCEPTION*. The second argument is an optional string to use as a message. This will override the default message.

There are also two additional auxiliary template classes *IntervalVectorT* and *IntervalMatrixT* in the package for type checking and declaration. The template classes check dimensions strongly before assignment. For software robustness, the assignment operators for *IntervalMatrix* and *IntervalVector* classes do not check whether the source dimension matches the destination dimension or not. Instead, the vector or matrix is simply resized. The template versions enable checking. They require that the source and destination objects be of the same dimensions before allowing assignment.

We have also made efforts to effectively manage the memory especially for intermediary results in our implementation. Although this is opaque to general users, readers who are interested may refer the source code for the details.

5. Installation and Usage

5.1. Installation

The computational environment is available at www.geocities.com/mike_nooner as a single zip file containing the documentation, source code, test, a sample application program, a README and a LICENSE file. After unzipping the downloaded package, one may install it by following the installation instructions in the package. The package is ready to install directly on machines with GCC or Microsoft Visual Studio .NET 2003. It can be installed on other platforms with C++ compilers in compliance with ISO/ANSI standard. However, some minor modifications may be needed.

Included in the package, there are four sets of tests. The first three test whether the *Interval*, *IntervalVector*, and *IntervalMatrix* classes function properly. With the `-r` option, the containment condition is tested for matrix-matrix multiplication, matrix-vector multiplication, matrix scaled accumulation, and vector scaled accumulation with randomly generated numbers. It is recommended to run the standard test cases for the library using the packaged tester application. It is recommended to pass the test output results in a file. Otherwise, the test outputs will be written to the screen.

5.2. Using the Package

To use the library in your applications, there are four required steps.

1. Include the file **IntBLAS.h**
2. All the classes, functions, and global variables are in the *intblas* namespace. So, make the appropriate **using** declaration(s).
3. You need to call the function *INIT_INTERVAL()* before making use of any of the classes, functions, or global variables. This function should ideally be called even before any declarations.
4. Finally, you will need to link to the **intblas.lib** static library.

A simple sample program, that performs level 0-3 interval basic linear algebra operations, is attached as the Appendix with I/O data.

6. Conclusions and future work

The computational environment for interval matrices reported in this paper has various commonly used functionalities in interval software development. It can be conveniently embedded into a standard C++ environment. We plan to further test the package and enhance it with more features. Using this package as a kernel, we are working on building applications for decision making systems based on fuzzy logic, interval valued databases, and interval matrices (Collins and Hu, 2005; de

Korvin et al, 2000; de Korvin and Hu, 2004; de Korvin et al, 2002). Suggestions, comments, and error reports are very appreciated for further improvements of the package.

References

Blackford, G., Demmel. J., Dongarra, J., Duff, and et al. Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard. *High Performance Computing Applications*, No.16, pp. 1-199, 2001. Also available at http://www.netlib.org/blas/blast-forum/blast-forum.html

Collins, D. and Hu, C. Fuzzily Determined Interval Matrix Games. *Proc. 2005 Berkeley Initiative in Soft Computing*, Forging the Frontiers, in press, 2005.

Dongarra, J. et al. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*,28(2), 2002.

Corliss, G., and Kearfott, B. Rigorous Global Search: Industrial Applications. *Reliable Computing*, 1-16, 1999.

de Korvin, A., Hu, C., and Sirisaengtaksin, O. On Firing Rules of Fuzzy Sets of Type II. *Int. J. of Applied Mathematics*, 3(2), 151-159, 2000.

de Korvin, A., Hu, C., and Chen, P. Association Analysis with Interval Valued Fuzzy Sets and Body of Evidence. *Proc. 2002 IEEE Int. Conf. on Fuzzy Systems*, 518-523, 2002.

de Korvin, A., Hu, C., and Chen, P. Generating and Applying Rule for Interval Valued Fuzzy Observations. *Lecture Notes in Computer Science* 3177, 279-284, 2004.

Duff, I. S., Vömel, C. Algorithm 818: A reference model implementation of the sparse BLAS in fortran 95. *ACM Transactions on Mathematical Software*, 28(2), 2002.

Duff, I. S., Heroux, M. A., Pozo, R. An overview of the sparse basic linear algebra subprograms: The new standard from the BLAS technical forum. *ACM Transactions on Mathematical Software*, 28(2), 2002.

Hansen, E. Global optimization using interval analysis. Marcel Dekker, New York, 1992.

Hu, C., Kearfott B., and Awad, A. On Bounding the Range of Some Elementary Functions in FORTRAN-77. *Interval Computations*, No.(3), 29-40, 1993.

Hu, C., Xu, S., and Yang, X. An Introduction to Interval Computation. *Theory and Practice in System Science*, 23(4), 59-62, 2003.

Kågström, B., van Loan, C. .
Algorithm 784: GEMM-based level 3 BLAS: portability and optimization. *ACM Transactions on Mathematical Software*, 24(3), 1998.

Kågström, B., Ling, P., van Loan, C. GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark. *ACM Transactions on Mathematical Software*, 24(3), 1998.

Kearfott, B., Dawande,M., Du, K., and Hu, C. Algorithm 737: INTLIB: a Portable Fortran-77 Interval Standard Function Library. *ACM, Trans. on Math. Software*, 20, 447-459, 1994.

Li, X., Demmel, J., and et al. Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software*, 28(2), 2002.

Moore, R. E. Methods and Applications of Interval Analysis. Society for Industrial and Applied Mathematics, 1979.

Sun Studio, Sun Microsystems, Sun Performance Library User's Guide. http://docs.sun.com/source/819-0498/, 2005.

Walster, W. Interval Version of the Basic Linear Algebra Subprogram Standard (BLAS). http://www.sun.com/software/sundev/whitepapers/blas.pdf, 2002.

Appendix: A simple application with I/O

```
#include <iostream>
#include <iomanip>
#include <IntBLAS.h> //Include IntBLAS

using namespace std;
using namespace intblas; //The IntBLAS namspace

int main()
{
  INIT_INTERVAL(); //Must initialize the library
  Interval a( 10, 11), b ( 5.5 ), int_r;

  //The output operators use the streams format
  cout << setprecision( 25 ) << setiosflags( ios::scientific );

  int_r = a * b;
  cout << "\nLevel 0 example:[10,11]*[5.5, 5.5] =\n" << int_r << endl;

  IntervalVector m( 3 ), n( 3 ), vec_r;
  m[0] = 1.1; m[1] = 2.3; m[2] = 4;
  n[0] = a;   n[1] = b;   n[2] = a+b;

  //vec_r = 2*m + 3*n
  vec_r = scaledAccumulation( m, n, 2, 3 );

  cout << "\nLevel 1 example: 2*{1,2,4} + 3*{5,6,7} =\n" << vec_r << endl;

  IntervalMatrix x( 3, 3 );
  x[0][0] = 1, x[0][1] = 2, x[0][2] = b;
  x[1][0] = a, x[1][1] = 4, x[1][2] = 6;
  x[2][0] = 3, x[2][1] = 6, x[2][2] = Interval( 9.5 );

  vec_r = x * m;
  cout << "\nLevel 2 example: x * {1, 2, 4} =\n" << mat_r << endl;

  IntervalMatrix y( 3, 3 ), mat_r;
  y[0][0] = b, y[0][1] = 2, y[0][2] = 3;
  y[1][0] = 2, y[1][1] = a, y[1][2] = 6;
  y[2][0] = b, y[2][1] = 6, y[2][2] = int_r;

  mat_r = x * y;
```

```
  cout << "\nLevel 3 example: x * y =\n" << mat_r << endl;

  return 0;
}
```

The output of the above sample program:

```
Level 0 example:[10,11]*[5.5, 5.5] =
[5.4999999999999978683717927e+01, 6.0500000000000021316282073e+01]

Level 1 example: 2*{1,2,4} + 3*{5,6,7} =
{ [3.2199999999999981525888870e+01, 3.5200000000000017053025658e+01]
  [2.1099999999999987210230756e+01, 2.1100000000000012079226508e+01]
  [5.4499999999999957367435854e+01, 5.7500000000000042632564146e+01] }


Level 2 example: x * {1, 2, 4} =
{ [2.7699999999999977973175191e+01, 2.7700000000000020605739337e+01]
  [4.4199999999999967315034155e+01, 4.5300000000000046895820560e+01]
  [5.5099999999999951683093968e+01, 5.5100000000000051159076975e+01] }


Level 3 example: x * y =
|  [3.9749999999999964472863212e+01, 3.9750000000000035527136788e+01]
[5.4999999999999957367435854e+01, 5.7000000000000049737991503e+01]
[3.1749999999999971578290570e+02, 3.4775000000000028421709430e+02]  |
|  [9.5999999999999928945726424e+01, 1.0150000000000008526512829e+02]
[9.5999999999999928945726424e+01, 1.0200000000000009947598301e+02]
[3.8399999999999960209606797e+02, 4.2000000000000045474735089e+02]  |
|  [8.0749999999999928945726424e+01, 8.0750000000000071054273576e+01]
[1.2299999999999988631316228e+02, 1.2900000000000011368683772e+02]
[5.6749999999999943156581139e+02, 6.1975000000000056843418861e+02]  |
```

Table II. Functionality Involving Interval Vectors:

Algebraic Operation	Mathematical Definition	Operator
Dot product	$\mathbf{r} \leftarrow \beta\mathbf{r} + \alpha\mathbf{x}^T\mathbf{y}$	scaledDot()
Vector norms	$r \leftarrow \|\|\mathbf{x}\|\|_1, r \leftarrow \|\|\mathbf{x}\|\|_2$	
	$r \leftarrow \|\|\mathbf{x}\|\|_\infty$	norm()
Sum	$\mathbf{r} \leftarrow \sum_i \mathbf{x}_i$	vectorSum()
Max magnitude & location	$k, \mathbf{x}_k$; $k = \arg\ \max_i\{\|\underline{x}_i\|, \|\overline{x}_i\|\}$	max()
Min absolute value & location	$k, \mathbf{x}_k$; $k = \arg\ \min_i\{\|\underline{x}_i\|, \|\overline{x}_i\|\}$	min()
Sum of squares	$(\mathbf{a}, \mathbf{b}) \leftarrow \sum_i \mathbf{x}_i^2$, $\mathbf{a} \cdot \mathbf{b}^2 = \sum_i \mathbf{x}_i^2$	sumOfSquares()
Reciprocal scale	$\mathbf{x} \leftarrow \mathbf{x}/\alpha$	reciprocalScale()
Scaled interval vector accumulation	$\mathbf{y} \leftarrow \alpha\mathbf{x} + \beta\mathbf{y}$	scaledAccumulation()
Scaled interval vector accumulation	$\mathbf{w} \leftarrow \alpha\mathbf{x} + \beta\mathbf{y}$	scaledAccumulation()
Scaled interval vector cancellation	$\mathbf{y} \leftarrow \alpha\mathbf{x} \ominus \beta\mathbf{y}$	scaledCancelation()
Scaled interval vector cancellation	$\mathbf{w} \leftarrow \alpha\mathbf{x} \ominus \beta\mathbf{y}$	scaledCancelation()
Set Operation		
Enclosed	$\mathbf{x}$ is enclosed in $\mathbf{y}$ if $\mathbf{x} \subseteq \mathbf{y}$	encloses()
Interior	$\mathbf{x}$ is enclosed in the interior of $\mathbf{y}$	interior()
Disjoint	$\mathbf{x}$ and $\mathbf{y}$ are disjoint if $\mathbf{x} \cap \mathbf{y} = \emptyset$	disjoint()
Intersection	$\mathbf{y} \leftarrow \mathbf{x} \cap \mathbf{y}, \mathbf{z} \leftarrow \mathbf{x} \cap \mathbf{y}$	operator &
Hull	the convex hull of $\mathbf{x}$ and $\mathbf{y}$	intervalHull()
Matrix-vector Operation		
Matrix vector product	$\mathbf{y} \leftarrow \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$	scaledVectorMult()
	$\mathbf{y} \leftarrow \alpha\mathbf{A}^T\mathbf{x} + \beta\mathbf{y}$	
Triangular solve	$\mathbf{x} \leftarrow \mathbf{T}\mathbf{x}, \mathbf{x} \leftarrow \mathbf{T}^T\mathbf{x}$	triangularMult()
	$\mathbf{x} \leftarrow \alpha\mathbf{T}^{-1}\mathbf{x}, \mathbf{x} \leftarrow \alpha\mathbf{T}^{-T}\mathbf{x}$	
Rank one updates	$\mathbf{A} \leftarrow \alpha\mathbf{x}\mathbf{y}^T + \beta\mathbf{A}$	rankeOneUpdate()
Utility Operation		
Vector copy	$\mathbf{x} \leftarrow \mathbf{y}$	=
Insertion operator	for output	<<
Extraction operator	for input	>>
Swap	$\mathbf{y} \leftrightarrow \mathbf{x}$	swap()
Permute vector	$\mathbf{x} \leftarrow P\mathbf{x}$	permute()
Empty element	k if $\mathbf{x}_k = \emptyset$; or -1	containsEmpty()
Left endpoint	$v \leftarrow \underline{x}$	lowerBounds()
Right endpoint	$v \leftarrow \overline{x}$	upperBounds()
Midpoint	$v \leftarrow (\underline{x} + \overline{x})/2$	midpoint()
Width	$v \leftarrow \overline{x} - \underline{x}$	width()
Construct	$\mathbf{x} \leftarrow u, v$	constructor

Table III. Functionality for Interval Matrices

Algebraic Operation	Mathematical Definition	Operator
Matrix norms	$r \leftarrow \|\|\mathbf{A}\|\|_1, r \leftarrow \|\|\mathbf{A}\|\|_F,$	norm()
	$r \leftarrow \|\|\mathbf{A}\|\|_\infty, r \leftarrow \|\|\mathbf{A}\|\|_{\max}$	
Diagonal scaling	$\mathbf{A} \leftarrow \mathbf{DA}, \mathbf{A} \leftarrow \mathbf{AD}$	diagonalScale()
Two sided diagonal scaling	$\mathbf{A} \leftarrow \mathbf{D}_1\mathbf{AD}_2$	diagonalScale2()
Two sided diagonal scaling	$\mathbf{A} \leftarrow \mathbf{DAD}$	diagonalScale2()
	$\mathbf{A} \leftarrow \mathbf{A} + \mathbf{BD}$	
Matrix acc and scale	$\mathbf{B} \leftarrow \alpha\mathbf{A} + \beta\mathbf{B},$	scaledAccumulation()
	$\mathbf{B} \leftarrow \alpha\mathbf{A}^T + \beta\mathbf{B}$	accTranspose()
Matrix add and scale	$\mathbf{C} \leftarrow \alpha\mathbf{A} + \beta\mathbf{B}$	scaledAccumulation()
Matrix matrix product	$\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$, $\mathbf{C} \leftarrow \alpha\mathbf{A}^T\mathbf{B} + \beta\mathbf{C}$,	operator *
	$\mathbf{C} \leftarrow \alpha\mathbf{AB}^T + \beta\mathbf{C}$, $\mathbf{C} \leftarrow \alpha\mathbf{A}^T\mathbf{B}^T + \beta\mathbf{C}$	
	$\mathbf{C} \leftarrow \alpha\mathbf{BA} + \beta\mathbf{C}$, $\mathbf{C} \leftarrow \alpha\mathbf{B}^T\mathbf{A} + \beta\mathbf{C}$,	operator *
	$\mathbf{C} \leftarrow \alpha\mathbf{BA}^T + \beta\mathbf{C}$, $\mathbf{C} \leftarrow \alpha\mathbf{B}^T\mathbf{A}^T + \beta\mathbf{C}$	
Set Operation		
Enclosed	$\mathbf{A}$ is enclosed in $\mathbf{B}$ if $\mathbf{A} \subseteq \mathbf{B}$	encloses()
Interior	$\mathbf{A}$ is enclosed in the interior of $\mathbf{B}$	interior()
Disjoint	$\mathbf{A}$ and $\mathbf{B}$ are disjoint if $\mathbf{A} \cap \mathbf{B} = \emptyset$	disjoint()
Intersection	$\mathbf{B} \leftarrow \mathbf{A} \cap \mathbf{B}$, $\mathbf{C} \leftarrow \mathbf{A} \cap \mathbf{B}$	operator &
Hull	the hull of $\mathbf{A}$ and $\mathbf{B}$	intervalHull()
Utility Operations		
Matrix copy	$\mathbf{B} \leftarrow \mathbf{A}$	operator =
	$\mathbf{B} \leftarrow \mathbf{A}^T$	operator =
Matrix transpose	$\mathbf{A} \leftarrow \mathbf{A}^T$	transpose()
Permute matrix	$\mathbf{A} \leftarrow \mathbf{PA}, \mathbf{A} \leftarrow \mathbf{AP}$	permute()
Empty element	if $\mathbf{A}$ has an empty interval element	containsEmpty()
Insertion operator	for output	<<
Extraction operator	for input	>>
Left endpoint	$C \leftarrow \underline{A}$	lowerBounds()
Right endpoint	$C \leftarrow \overline{A}$	upperBounds()
Midpoint	$C \leftarrow (\underline{A} + \overline{A})/2$	midpoint()
Width	$C \leftarrow \overline{A} - \underline{A}$	width()
Construct	$\mathbf{A} \leftarrow B, C$	constructor