ivalDb: REFERENCE DOCUMENTATION

Modal Intervals and Control Engineering group

May 7, 2004

Contents

1 PRESENTATION

This is the documentation for ivalDb software, developed at Girona University by Modal Intervals and Control Engineering group.
This library is intended to provide an easy way to make programs using Modal Intervals. For more information about Modal Intervals see [*].
This library includes the following features:

- Basic operators (sum, substract, multiplication, division and pow)
- Trigonometrical functions (sin, cos, tan, inverse functions)
- Boolean operations
- Proper and Improper Interval operations
- Exception handling

This document is organized as follow:

- Chapter 1: includes an explanation about how to use the library
- Chapter 2: describes available commands and operators to use

COPYRIGHT

IvalDb 0.1 Academic Software License Agreement

The authors hereby grant you non-exclusive and non-transferable license to use, copy and modify the Software and its documentation for non-commercial purposes.

The Software and derived products of the Software shall not be redistributed without specific prior written permission.

LIMITATIONS

IN NO EVENT SHALL THE AUTHORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
THE AUTHORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING,BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

2 ABOUT THE LIBRARY

Object-oriented

IvalDb was made in Borland C++, for that reason uses all the potence of Oriented Object Programming to handle every Interval like an object including many properties and functions.

Numeric guarantee

IvalDb assures the numeric guaranty thanks to the use of FDLIBM [**?**], a C library developed by Sun Microsystems.

FDLIBM is a math C library (like math.h) which assures for an IEEE754 machine, in the worst case, an ULP (Units (Bits) of the Last Place) of error for all the given functions. Moreover, it assures multi-platform compatibility (PC,SUN...).

FDLIBM provides a function which allows to round a floating point number to $+\infty$ or to $-\infty$. Then, knowing that the maximal error that can be committed by FDLIBM functions is one ULP, it is easy to implement a guaranteed interval function by adding an ULP to the upper bound of the solution interval and by resting and ULP to the lower bound of the solution interval.
This method may be numerically conservative respect other techniques used by other libraries to assure the numerical guarantee but it is also more efficient in terms of time consuming because it does not change the rounding mode of the computer.

Use in different environments

The library was adapted to allow programmers to use it on two operating systems: Linux and windows. For Linux the programmers can use g++ to compile programs, but in Windows there are two choices, Borland C++ and Visual C++.

3 INSTALLATION

The library contains two files called ivaldb.cpp and ivaldb.h, and a file called benchmark.cpp which show the operation of many interval functions. It also uses FDLIBM library to guarantee numeric operations.
This library can be used under Borland C++, Visual C++ and g++ compiler. To select the correct environment you must first un-comment the line corresponding to definition describing the environment to use. You can find these lines in ivaldb.h at the beginning of the code (after the license agreement). These lines are:

//Uncomment the following line to use with Borland C++
#define BORLAND
//Uncomment the following line to use with Microsoft Visual C++
//#define VISUAL
//Uncomment the following line to use with g++ compiler for Linux
//#define LINUX

In this case the uncommented line refers to Borland C++ environment.

3.1 Use with Borland C++

1. Start a new project
2. Use 'project->add to project' to insert the file ivaldb.cpp
3. Use 'project->add to project' to insert fdlibm.lib included in the zip file
4. Include the header file in the main unit(#include "ivaldb.h")
5. You can start to use the library

3.2 Use with Microsoft Visual C++

1. Create a new workspace
2. Use 'project->add to project' to insert the file ivaldb.cpp
3. Use 'project->add to project' to insert all FDLIBM C files included in the zip file. There is not library version of FDLIBM to use with Visual C++.
4. Include the header file (#include "ivaldb.h")
5. You can start to use the library

3.3 Use with g++ compiler under linux

1. Create a folder for the new project and copy ivaldb.h and ivaldb.cpp
2. If FDLIBM is not installed on your system, uncompress and install it.
   (a) Use 'rpm -i fdlibm.rpm' to install the file fdlibm.rpm included in zip file.
   (b) Look for the destination directory. In Red Hat, that folder is /usr/src/redhat/SOURCES/fdlibm-5.2.
   (c) Go to destination directory and type ./configure
   (d) Type make
   (e) FDLIBM must crete a file called libm.a.
   (f) Copy this file to your project folder.
3. Go to the folder containing ivaldb files and create a c++ project file to use with the library (you could use benchmark file included in the zip file).
4. type: 'g++ -Wno-deprecated benchmark.cpp ivaldb.cpp libm.a -o program.out'. In this case, the project file is benchmark.cpp and the output file is program.out
5. To run the output file, type ./program.out.

4 AVAILABLE FUNCTIONS AND OPERATORS

4.1 CREATION OF NEW INTERVALS

4.1.1 Completely defined bound interval

Description:
The user must specify both lower and upper bounds to create the new interval.

Input:
Double,Double
Output:
IvalDb

Syntax:
ivalDb A=ivalDb(3.0,8.5);

4.1.2 Only one bound defined

Description:
The user must specify only one value, then it will be assigned to lower and upper bounds.

Input:
Double
Output:
ivalDb

Syntax:
ivalDb A=ivalDb(3.0);

4.1.3 Create new intervals using a previously defined interval

Description:
The user creates the new interval using as parameter another interval. The new interval is exactly the interval used to create it.

Input:
ivalDb
Output:
ivalDb

Syntax:
ivalDb B=ivalDb(A);

4.1.4 Create new interval without parameters

Description:
If it is not specified parameters as input, the new interval will be defined as [-infinite,+infinite].

Input:
None
Output:
ivalDb

Syntax:
ivalDb A;

4.2 ACCESS TO INTERVAL BOUNDS

4.2.1 GetInfimum

Description:
Returns the LOWER bound value of an interval

Syntax:

ivalDb A=(3,5);
Double lb;
lb=A.GetInfimum();

Input:
None
Output:
Double

4.2.2 GetSupremum

Description:
Returns the UPPER bound value of an interval

Syntax:

ivalDb A=(3,5);
Double lb;
lb=A.GetSupremum();

Input:
None
Output:
Double

4.2.3 SetInfimum

Description:
Establishes the LOWER bound value of an interval

Syntax:

ivalDb A;
A.SetInfimum(18.0);

Input:
Double
Output:

None

4.2.4 SetSupremum

Description:
Establishes the UPPER bound value of an interval

Syntax:
ivalDb A;
A.SetSupremum(12.0);

Input:
Double
Output:
None

4.2.5 SetBounds

Description:
Establishes LOWER and UPPER bounds for an interval

Syntax:
ivalDb A;
A.SetBounds(15.0,12.0);

Input:
Double, Double
Output:
None

4.3 OPERATIONS BETWEEN INTERVALS

4.3.1 Assignation operator

Description:
Assigns the value of one interval to another.

Syntax:
ivalDb A,B=ivalDb(8.0,3.0);
A=B;

Input:
ivalDb
Output:
ivalDb

4.3.2 Negation operator

Description:
Return the the inverse value of an interval. The lower bound become upper bound with opposite sign and upper bound become lower bound with opposite sign.

Syntax:
ivalDb A,B=ivalDb(8.0,3.0);
A=-B;

Input:
ivalDb
Output:
ivalDb

4.3.3 Add operator

Description:
Returns the sum between two intervals. The rule used is:
A+B=[a1+b1,a2+b2]

Syntax:
ivalDb C,A=(-10.0,14.0),B=ivalDb(8.0,3.0);
C = A + B;

Input:
ivalDb, ivalDb
Output:

ivalDb

4.3.4 Subtract operator

Description:
Returns the difference between two intervals. The following rule is used:
A-B=[a1-b2,a2-b1]

Syntax:

ivalDb C,A=(-10.0,14.0),B=ivalDb(8.0,3.0);
C = A - B;

Input:
ivalDb, ivalDb
Output:
ivalDb

4.3.5 Multiply operator

Description:
Returns the result of the multiplication between two intervals. There are some exceptions for this operator considered in chapter 3. The following rules are used:

A*B =[a1*b1,a2*b2] if a1 $\geq$ 0, a2 $\geq$ 0, b1 $\geq$ 0, b2 $\geq$ 0
=[a1*b1,a1*b2] if a1 $\geq$ 0, a2 $\geq$ 0, b1 $\geq$ 0, b2 $<$ 0
=[a2*b1,a2*b2] if a1 $\geq$ 0, a2 $\geq$ 0, b1 $<$ 0, b2 $\geq$ 0
=[a2*b1,a1*b2] if a1 $\geq$ 0, a2 $\geq$ 0, b1 $<$ 0, b2 $<$ 0
=[a1*b1,a2*b1] if a1 $\geq$ 0, a2 $<$ 0, b1 $\geq$ 0, b2 $\geq$ 0
=[max(a1*b1,a2*b2),min(a2*b1,a1*b2)] if a1 $\geq$ 0, a2 $\geq$ 0, b1 $<$ 0, b2 $<$ 0
=[0,0] if a1 $\geq$ 0, a2 $<$ 0, b1 $<$ 0, b2 $\geq$ 0
=[a2*b2,a1*b2] if a1 $\geq$ 0, a2 $<$ 0, b1 $<$ 0, b2 $<$ 0
=[a1*b2,a2*b2] if a1 $<$ 0, a2 $\geq$ 0, b1 $\geq$ 0, b2 $\geq$ 0
=[0,0] if a1 $<$ 0, a2 $\geq$ 0, b1 $\geq$ 0, b2 $<$ 0
=[min(a1*b1,a2*b2),max(a1*b1,a2*b2)] if a1 $<$ 0, a2 $\geq$ 0, b1 $<$ 0, b2 $\geq$ 0
=[a2*b1,a1*b1] if a1$<$0, a2 $\geq$ 0, b1 $<$ 0, b2 $<$ 0
=[a1*b2,a2*b1] if a1$<$0, a2 $<$ 0, b1 $\geq$ 0, b2 $\geq$ 0
=[a2*b2,a2*b1] if a1$<$0, a2 $<$ 0, b1 $\geq$ 0, b2 $<$ 0
=[a1*b2,a1*b1] if a1$<$0, a2 $<$ 0, b1 $<$ 0, b2 $\geq$ 0
=[a2*b2,a1*b1] if a1$<$0, a2 $<$ 0, b1 $<$ 0, b2 $<$ 0

Syntax:

ivalDb C,A=(-10.0,14.0),B=ivalDb(8.0,3.0);

*C = A * B;*

Input:
ivalDb, ivalDb
Output:
ivalDb

4.3.6 Division operator

Description:
Returns the result of the division between two intervals. There are some exceptions for this operator considered in chapter 3. The following rules are used:

$$
\begin{aligned}
\text{A/B} &= [a1/b2, a2/b1] && \text{if } a1 \geq 0,\ a2 \geq 0,\ b1 > 0,\ b2 > 0 \\
&= [a2/b2, a1/b1] && \text{if } a1 \geq 0,\ a2 \geq 0,\ b1 < 0,\ b2 < 0 \\
&= [a1/b2, a2/b2] && \text{if } a1 \geq 0,\ a2 < 0,\ b1 > 0,\ b2 > 0 \\
&= [a2/b1, a1/b1] && \text{if } a1 \geq 0,\ a2 < 0,\ b1 < 0,\ b2 < 0 \\
&= [a1/b1, a2/b1] && \text{if } a1 < 0,\ a2 \geq 0,\ b1 > 0,\ b2 > 0 \\
&= [a2/b2, a1/b2] && \text{if } a1 < 0,\ a2 \geq 0,\ b1 < 0,\ b2 < 0 \\
&= [a1/b1, a2/b2] && \text{if } a1 < 0,\ a2 < 0,\ b1 > 0,\ b2 > 0 \\
&= [a2/b1, a1/b2] && \text{if } a1 < 0,\ a2 < 0,\ b1 < 0,\ b2 < 0
\end{aligned}
$$

Syntax:
ivalDb C,A=(-10.0,14.0),B=ivalDb(8.0,3.0);
C = A / B;

Input:
ivalDb, ivalDb
Output:
ivalDb

4.3.7 Exponential operator

Description:
Returns the result of apply exponential value to an interval with any exponent.

Syntax:
ivalDb A,B=(-10.0,14.0);
$A = B\hat{3};$

Input:
ivalDb, integer
Output:

ivalDb

4.3.8 Meet operator

Description:
Apply the meet operation between two interval. The new interval will contain the maximum value of the lower bounds and the minimum value of upper bounds like lower and upper bounds respectively.

Syntax:

ivalDb A=(6.0,3.0),B=(8.0,-10.0);
A = A&&B;

Input:
ivalDb, ivalDb
Output:
ivalDb

4.3.9 Join operator

Description:
Apply the join operator between two interval. The new interval will contain the minimum value of the lower bounds and the maximum value of upper bounds like lower and upper bounds respectively.

Syntax:

ivalDb A=(6.0,3.0),B=(8.0,-10.0);
A = A——B;

Input:
ivalDb, ivalDb
Output:
ivalDb

4.3.10 Relational operators

Description:
This operators allow make comparison between two intervals.
The operations are greater than (>>), greater or equal ($\geq$), least than (<<), least or equal ($\leq$), or equal (==).

Syntax:

ivalDb A=(1.0,3.0),B=(-5.0,20.0);

if (A>>B) cout<<"A is greater than B"<<endl;
if (A>=B) cout<<"A is greater or equal than B"<<endl;
if (A<<B) cout<<"A is least than B"<<endl;
if (A<<B) cout<<"A is least than B"<<endl;
if (A<=B) cout<<"A is least than B"<<endl;

Input:
ivalDb, ivalDb
Output:
unsigned long

4.4 INTERVAL FUNCTIONS

4.4.1 Abs

Description:
Returns absolute value from an interval.

Syntax:

ivalDb A=(1.0,3.0));
A=abs(B);

Input:
ivalDb
Output:
ivalDb

4.4.2 Pow

Description:
This is the function version of exponential operator.

Syntax:

ivalDb A,B=(4.0,-2.0));
A=Pow(B,5);

Input:
ivalDb, long
Output:
ivalDb

4.4.3 sqr

Description:
Return interval elevated to square.

Syntax:

ivalDb A,B=(4.0,-2.0));
A=sqr(B);

Input:
ivalDb
Output:
ivalDb

4.4.4 sqrt

Description:
Return an interval that contain the square root of another.

Syntax:

ivalDb A,B=(4.0,-2.0));
A=sqrt(B);

Input:
ivalDb
Output:
ivalDb

4.4.5 cbrt

Description:
Return the cubic root from an interval.

Syntax:

ivalDb A,B=(4.0,-2.0));
A=cbrt(B);

Input:
ivalDb
Output:
ivalDb

4.4.6 root

Description:
Return the generic root from an interval.

Syntax:

ivalDb A,B=(4.0,-2.0));
A=root(B,2);

Input:
ivalDb, int
Output:
ivalDb

4.4.7 exp

Description:
Calculates exp function from an Interval.

Syntax:

ivalDb A,B=(0,1));
A=exp(B);

Input:
ivalDb, int
Output:
ivalDb

4.4.8 log

Description:
Calculates logarithm from an Interval. There is base 10 logarithm too.

Syntax:

ivalDb A,B,C=(2,5));
A=log10(C);B=log(C);

Input:
ivalDb, int
Output:
ivalDb

4.4.9 Trigonometrical functions

sin: Get sine from an interval into another. The input must be in radians.
cos: Get cosine from an interval into another. Input in radians.
tan: Get tangent from an interval into another.
asin: Calculate arcsine from an interval.
acos: Calculate arccosine from an interval.
sinh: Calculate hyperbolic sine from an interval.
cosh: Calculate hyperbolic cosine from an interval.
tanh: Calculate hyperbolic tangent from an interval.
asinh: Calculate arc hyperbolic sine from an interval.
acosh: Calculate arc hyperbolic cosine from an interval.
atanh: Calculate arc hyperbolic tangent from an interval.

Syntax:

ivalDb A,B=(0,90));

cout<<sin(B)<<endl;

Input:
ivalDb
Output:
ivalDb

4.5 METRIC FUNCTIONS

4.5.1 Width

Description:
Return absolute value of the difference between upper and lower bounds.

Syntax:

ivalDb A=(3,2);
double w;
w=Width(A);

Input:
ivalDb
Output:
double

4.5.2 Center

Description:
Return the half of the sum between upper and lower bounds.

Syntax:

ivalDb A=(-5,8);
double c;
c=Width(A);

Input:
ivalDb
Output:
double

4.5.3 AQuarter

Description:
Return a quarter of the interval width.

Syntax:

ivalDb A=(5.5,1.2);
double q;
q=Quarter(A);

Input:
ivalDb
Output:

double

4.5.4 ThreeQuarters

Description:
Return a three quarters of the interval width.

Syntax:

ivalDb A=(1.5,10);
double tq;
tq=Quarter(A);

Input:
ivalDb
Output:
double

4.6 MODAL FUNCTIONS

4.6.1 Prop

Description:
If the interval is improper then convert it to the proper.

Syntax:

ivalDb A,B=(14,1);
A=Prop(B);

Input:
ivalDb
Output:
ivalDb

4.6.2 Impr

Description:
If the interval is proper then convert it to the improper.

Syntax:

ivalDb B=(1,14);
A=Impr(B);

Input:
ivalDb
Output:
ivalDb

4.6.3 Du

Description:
Return dual from an interval.

Syntax:

ivalDb B=(1,14);
A=Du(B);

Input:
ivalDb
Output:
ivalDb

4.6.4 Dual1

Description:
Return dual from an interval. If the interval is proper an internal flag is activated.

Syntax:

ivalDb B=(1,14);
A=Dual1(B);

Input:
ivalDb
Output:
ivalDb

4.6.5 Dual2

Description:
Return dual from an interval only if the internal flag is active.

Syntax:

ivalDb B=(1,14);
A=Dual2(B);

Input:
ivalDb
Output:
ivalDb

4.7 BOOLEAN FUNCTIONS

4.7.1 IsProper

Description:
Return true if the interval is proper, else return false.

Syntax:

ivalDb A=(14,1);
if (IsProper(A)) cout<<"Is proper"¡¡endl; else cout<<"Is improper"<<endl;
if (A.IsProper()) cout<<"Is proper"¡¡endl; else cout<<"Is improper"<<endl;

Input:
ivalDb -or- None (if it is used as a property of the variable)
Output:
bool

4.7.2 IsImproper

Description:
Return true if the interval is improper, else return false.

Syntax:

ivalDb A=(14,1);
if (IsImproper(A)) cout<<"Is improper"<<endl; else cout<<"Is proper"<<endl;
if (A.IsImproper()) cout<<"Is improper"<<endl; else cout<<"Is proper"<<endl;

Input:
ivalDb -or- None (if it is used as a property of the variable)
Output:
bool

4.7.3 IsInterval

Description:
Return true if lower bound is different than upper bound.

Syntax:

ivalDb A=(1,1);
if (IsInterval(A)) cout<<"Is an interval"<<endl; else cout<<"Is only a point"<<endl;
if (A.IsInterval()) cout<<"Is an interval"<<endl; else cout<<"Is only a point"<<endl;

Input:
ivalDb -or- None (if it is used as a property of the variable)
Output:
bool

4.7.4 IsEmpty

Description:
Return true if either lower or upper bounds are NaNs.

Syntax:

ivalDb A=(NaN(),1.3);
if (IsEmpty(A)) cout<<"Is an empty interval"<<endl;
if (A.IsEmpty()) cout<<"Is an empty interval"<<endl;

Input:
ivalDb -or- None (if it is used as a property of the variable)
Output:
bool

4.7.5 IsInside

Description:
Return true if the first interval is contained in second interval.

Syntax:

ivalDb A=(4,9),B=(0,10);
if (IsInside(A,B)) cout<<"A is contained in B"<<endl;
if (A.IsInside(B)) cout<<"A is contained in B"<<endl;

Input:
ivalDb, ivalDb -or- ivalDb (if it is used as property of the variable)
Output:
bool

4.7.6 IsOutside

Description:
Return true if the first interval is not contained in second interval.

Syntax:

ivalDb A=(0,5),B=(4,9);
if (IsOutside(A,B)) cout<<"A is not contained in B"<<endl;

if (A.IsOutside(B)) cout<<"A is not contained in B"<<endl;

Input:
ivalDb, ivalDb -or- ivalDb (if it is used as a property of the variable)
Output:
bool

4.7.7 IsIntersecting

Description:
Return true if the Intervals intersect.

Syntax:

ivalDb A=(0,5),B=(4,9);
if (IsIntersecting(A,B)) cout<<"A is intersecting B"<<endl;
if (A.IsIntersecting(B)) cout<<"A is not intersecting B"<<endl;

Input:
ivalDb, ivalDb -or- ivalDb (if it is used as a property of the variable)
Output:
bool

4.7.8 IsBigger

Description:
Return true if one interval is completely bigger than another .

Syntax:

ivalDb A=(0,5),B=(6,10);
if (IsBigger(A,B)) cout<<"A is bigger than B"<<endl;
if (A.IsBigger(B)) cout<<"A is bigger than B"<<endl;

Input:
ivalDb, ivalDb -or- ivalDb (if it is used as a property of the variable)
Output:
bool

4.7.9 IsSmaller

Description:
Return true if one interval is completely smaller than another .

Syntax:

ivalDb A=(0,5),B=(6,10);

if (IsSmaller(A,B)) cout<<"A is smaller than B"<<endl;
if (A.IsSmaller(B)) cout<<"A is smaller than B"<<endl;

Input:
ivalDb, ivalDb -or- ivalDb (if it is used as a property of the variable)
Output:
bool

4.8 AUXILIARY FUNCTIONS

These functions return useful intervals to use in some calculation.

PI:	Returns the interval version of PI.
PI2:	Returns interval PI multiplied by 2.
PI05:	Returns interval PI divided by 2.
PI025:	Returns interval PI divided by 4.
LN2:	Returns interval version of LN(2).
Zero:	Returns [0,0].
Infinity:	Returns [-infinity,+infinity].
DualInfinity:	Returns [+infinity,-infinity].

Syntax:

cout<<"PI is"<< PI() <<endl;

Input:
void
Output:
ivalDb

4.9 NOT MEMBER FUNCTIONS

There are some functions that are defined outside ivalDb object. They work only with double values.

IsNaN:	Returns true if the double value is NaN.
NaN:	Returns IEEE754 NaN value.
PlusInfinity:	Returns IEEE754 +infinity value.
MinusInfinity:	Returns IEEE754 -infinity value.
IsPlusInfinity:	Returns true if the double value is +infinity.
IsMinusInfinity:	Returns true if the double value is -infinity.
IsInfinity:	Returns true if the double value is either +infinity or -infinity.
AddULP:	Returns the next representable double-precision floating-point value following the double value entered in the direction of +infinity.
RestULP:	Returns the next representable double-precision floating-point value following the double value entered in the direction of -infinity.

Syntax:

```
double A=NaN();
if (IsNaN(A)) cout<<"A is NaN"<<endl;
double B=PlusInfinity();
if (Isinfinity(B)) cout<<"B is infinity"<<endl;
double C=RestULP(B);
if (!Isinfinity(A)) cout<<"C is not infinity"<<endl;
```

Input:
ivalDb
Output:
bool