Validated computation by INTLAB

INTLAB (INTerval LABoratory)

→ MATLAB Toolbox for reliable computing

http://www.ti3.tu-harburg.de/rump/intlab/

- Changing of a rounding mode
- Interval arithmetic (real, complex)
- Functions for enclosures

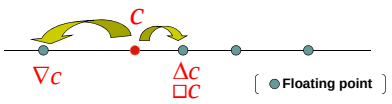
1

Changing of a rounding mode

Floating point system based on IEEE Standard 754

$F \subset \mathbb{R}$: set of floating points $(c \in \mathbb{R})$

- **Rounding upwards** $\Delta : \mathbb{R} \to F$
  Rounds to the smallest floating point which is equal to or larger than c.
- **Ronuding downwards** $\nabla : \mathbb{R} \to F$
  Rounds to the largest floating point which is equal to or smaller than c.
- **Rounding to nearest** $\square : \mathbb{R} \to F$
  Rounds to the floating point which is the nearest to c.



2

Verified computation by changing a rounding mode

【Example】

Compute the inner product Z
of $x = (x_1, \cdots, x_N),\ y = (y_1, \cdots, y_N)$

Here setround (up) and setround (down) stand for the rounding upwards and downwards respectively.

setround (down);
$\underline{z} = \sum_{k=1}^{N} x_k y_k$;
setround (up);
$\overline{z} = \sum_{k=1}^{N} x_k y_k$;

→ $\underline{z} \le z \le \overline{z}$

is assured in mathematically rigorous sense.

3

setround function in INTLAB

→ Changes the rounding mode

setround(1) ・・・・・・Rounding upwards

setround(-1) ・・・・・・Rounding downwards

setround(0) ・・・・・・Rounding to nearest

```
>> getround
ans =
   0
```

→ We can check the current rounding mode by "getround".
(The default mode is Rounding to nearest.)

Remark: If you once changed the rounding mode by the "setround" then the mode is maintained until you change it again.

4

e.g. Compute $\sum_{i=1}^{100000} 0.1 = 10^4$ with different rounding modes

```
>> setround(0); x=0;
>> for i=1:100000, x=x+0.1; end
>> disp(x)
   1.000000000001885e+004

>> setround(1); x=0;
>> for i=1:100000, x=x+0.1; end
>> disp(x)
   1.000000000003054e+004

>> setround(-1); x=0;
>> for i=1:100000, x=x+0.1; end
>> disp(x)
   9.999999999947979e+003
```

Rounding to nearest

Rounding upwards

Rounding downwards

5

Input of an interval

■ intval

```
>> a=intval(0.1)
intval a =
   0.10000000000000
>> rad(a)
ans =
     0
>> b=intval('0.1')
intval b =
   0.10000000000000
>> rad(b)
ans =
   1.387778780781446e-017
```

generates a point interval a

generates a real interval b which has a width

※ "rad" means the radius of an interval. (= half of width)

6

Interval vector and Interval matrix

```
>> A=intval([0.1 0.2 0.3 0.4])
intval A =
   0.10000000000000   0.20000000000000   0.30000000000000   0.40000000000000
>> rad(A(1,1))
ans =
     0
>> B=intval(['0.1 0.2 0.3 0.4'])
intval B =
   0.10000000000000
   0.20000000000000
   0.30000000000000
   0.40000000000000
>> B=reshape(B,2,2)
intval B =
   0.10000000000000   0.30000000000000
   0.20000000000000   0.40000000000000
>> rad(B(1,1))
ans =
   1.387778780781446e-017
```


■ **infsup(a,b)** ··· generates an interval which includes [a,b]

```
>> a=infsup(0.1,0.2)
intval a =
[    0.1000,    0.2001]        interval a
>> A=[infsup(0.1,0.11), infsup(0.2,0.21); infsup(0.3,0.31), infsup(0.4,0.41)]
intval A =
[    0.1000,    0.1101] [    0.2000,    0.2101]
[    0.2999,    0.3101] [    0.4000,    0.4101]        interval matrix A
```

■ **midrad(c,w)** ··· generates an interval whose midpoint is c, and radius is w

```
>> a=midrad(0.1, 1e-3)
intval a =
[    0.0989,    0.1011]        interval a
>> A=[midrad(0.1,1e-3); midrad(0.2,1e-3)]
intval A =
[    0.0989,    0.1011]
[    0.1989,    0.2011]        interval vector A
>> B=[midrad(0.1,1e-3), midrad(0.2,1e-3); midrad(0.3,1e-3), midrad(0.4,1e-3)]
intval B =
[    0.0989,    0.1011] [    0.1989,    0.2011]        interval matrix B
[    0.2989,    0.3011] [    0.3989,    0.4011]
```


Changing of displaying for an interval

In INTLAB there are three ways of output for an interval :

1. add _ e.g: 2.00__
   (_ means uncertainties)
2. [a, b] e.g: [0.1, 0.1001]
   (lower bound is “a”, upper bound is “b”)
3. <a, r> e.g: <1.0, 1e-3>
   (midpoint is “a”, radius is “r”)

The default output is 1.
“intvalinit” can change the way of displaying

1	⟶	intvalinit('display_')
2	⟶	intvalinit('displayinfsup')
3	⟶	intvalinit('displaymidrad')

Example

```
>> intvalinit('display_')
===> Default display of intervals with uncertainty (e.g. 3.14_)
>> pi=acos(intval('-1'))
intval pi =
   3.1415_

>> intvalinit('displayinfsup')
===> Default display of intervals by infimum/supremum (e.g. [ 3.14 , 3.15 ])
>> pi
intval pi =
[ 3.14159265358979,   3.14159265358980]

>> intvalinit('displaymidrad')
===> Default display of intervals by midpoint/radius (e.g. < 3.14 , 0.01 >)
>> pi
intval pi =
< 3.14159265358979,  0.00000000000001>
```


A complex interval is set by the midpoint/radius notation with the midpoint as a complex number.

```
>> intvalinit('display_')
===> Default display of intervals with uncertainty (e.g. 3.14_)
>> c=midrad(1+2i,0.01)
intval c =
   1.00_____________ + 2.00_____________i

>> intvalinit('displaymidrad')
===> Default display of intervals by midpoint/radius (e.g. < 3.14 , 0.01 >)
>> c
intval c =
<   1.00000000000000 + 2.00000000000000i,  0.01000000000001>

>> intvalinit('displayinfsup')
===> Default display of intervals by infimum/supremum (e.g. [ 3.14 , 3.15 ])
>> c
intval c =
[   0.98999999999998 + 1.98999999999999i,   1.01000000000001 + 2.01000000000001i]
```


midrad(a, r) generates a real interval if “a” is real, so you should use “cintval” if you want to generate a complex interval which has a real midpoint.

```
>> intvalinit('displaymidrad')
===> Default display of intervals by midpoint/radius (e.g. < 3.14 , 0.01 >)
>> c=midrad(0.0, 1e-3)
intval c =
<   0.00000000000000,  0.00100000000001>
>> c=cintval(0.0, 1e-3)
intval c =
  1.0e-002 *
<   0.00000000000000 +  0.00000000000000i,  0.10000000000001>
```

※ **A complex interval is always treated in the form of**

$$\{z \in \mathbb{C}; |z-a| \le r\}$$

Interval Arithmetic a, b : intervals

formula	expression in INTLAB
$a+b$	a + b
$a-b$	a - b
ab	a * b
a/b	a / b

※Elementwise operation is same as MATLAB （ a .* b etc.）

13

comparison operation ⟶ true: 1 false: 0

formula	expression in INTLAB
$a < b$	a < b
$a \leq b$	a <= b
$a > b$	a > b
$a \geq b$	a >= b
$a = b$	a == b
$a \neq b$	a ~= b
$a \subseteq b$	in(a,b)
$a \subset b$	in0(a,b)

14

Example for “in” and “in0”

```
>> a=infsup(1,3)
intval a =
[   1.00000000000000,   3.00000000000000]
>> b=infsup(0,3)
intval b =
[   0.00000000000000,   3.00000000000000]
>> c=infsup(0,4)
intval c =
[   0.00000000000000,   4.00000000000000]
>> in(a,b)
ans =
     1
>> in0(a,b)
ans =
     0
>> in0(a,c)
ans =
     1
```

15

Interval trigonometric functions etc.（cf. “help intval”）

```
>> a
intval a =
[   1.00000000000000,   3.00000000000000]
>> sin(a)
intval ans =
[   0.14112000805986,   1.00000000000000]
>> a=infsup(0,pi/2)
intval a =
[   0.00000000000000,   1.57079632679490]
>> sin(a)
intval ans =
[   0.00000000000000,   1.00000000000001]
>> exp(a)
intval ans =
[   1.00000000000000,   4.81047738096536]
>> cosh(a)
intval ans =
[   1.00000000000000,   2.50917847865806]
>> log(a)
intval ans =
[            - Inf,   0.45158270528946]
```

16

There are several functions for enclosures.

Example 1: verifylss

Enclose the solution of $Ax = b$

```
>> A=rand(5); b=rand(5,1);
>> verifylss(A,b)
intval ans =
[   0.83144502264404,   0.83144502264405]
[   0.70563459485949,   0.70563459485952]
[   1.15370741392371,   1.15370741392373]
[  -1.22407501349193,  -1.22407501349192]
[  -0.82811808977386,  -0.82811808977385]
```

17

If A is an interval matrix then

A¥b

is also available.（¥ means back slash.）

```
>> A=midrad(A,1e-4);
>> A¥b
intval ans =
[   0.82933211341759,   0.83355793187051]
[   0.70286710112650,   0.70840208859252]
[   1.15008203307291,   1.15733279477453]
[  -1.22949917932440,  -1.21865084765946]
[  -0.83160470085303,  -0.82463147869468]
```

18

Example 2: verifyeig

Compute a verified eigenpair of $Ax = \lambda x$

```
>> A=wilkinson(9)
A =
     4     1     0     0     0     0     0     0     0
     1     3     1     0     0     0     0     0     0
     0     1     2     1     0     0     0     0     0
     0     0     1     1     1     0     0     0     0
     0     0     0     1     0     1     0     0     0
     0     0     0     0     1     1     1     0     0
     0     0     0     0     0     1     2     1     0
     0     0     0     0     0     0     1     3     1
     0     0     0     0     0     0     0     1     4
```

```
>> [V,D]=eig(A);
>> for i=1:9 disp(D(i,i))
end
  -1.12542241567332
   0.25471875982586
   0.95258421907521
   1.82271708088711
   2.17828473954998
   3.17728291911289
   3.24739647257898
   4.74528124017414
   4.74715698446914
```

computation of approximate eigenpairs

In case that you want to enclose the first eigenpair:

```
>> [L,X]=verifyeig(A,D(1,1),V(:,1))
intval L =
[ -1.12542241567332,  -1.12542241567331]
intval X =
[ -0.00742897533695,  -0.00742897533694]
[  0.03807663671744,   0.03807663671745]
[ -0.14965323529066,  -0.14965323529065]
[  0.42965293943800,   0.42965293943801]
[ -0.76354075315081,  -0.76354075315080]
[  0.42965293943800,   0.42965293943801]
[ -0.14965323529066,  -0.14965323529065]
[  0.03807663671744,   0.03807663671745]
[ -0.00742897533695,  -0.00742897533694]
```

Enclosure of the eigenvalue

Enclosure of the eigenvector

Exercise (Matrix Eigenvalue Problem)

$p, q : (0,1) \to \mathbb{R}$ periodic, positive

$$\begin{cases} -\big(p(x)u'(x)\big)' + q(x)u(x) = \lambda u(x) & \text{in } (0,1) \\ u(1) = e^{i\mu}u(0),\ u'(1) = e^{i\mu}u'(0) & (\mu \in \mathbb{R}) \end{cases}$$

Weak Formulation

$$a(u,v) := \int_0^1 p(x)u'(x)\overline{v'(x)}\,dx + \int_0^1 q(x)u(x)\overline{v(x)}\,dx, \quad u, v \in H^1_{per}(0,1)$$

$$H^1_{per}(0,1) = \left\{ u \in H^1(0,1);\ u(1) = e^{i\mu}u(0) \right\}$$

$$a(u,v) = \lambda \langle u, v \rangle_{L^2}$$

Approximate eigenfunction

$$\phi_n(x) = e^{i(2\pi n + \mu)x}$$

$$u_N(x) = \sum_{n=-N}^{N} u_n \phi_n(x), \quad u_n \in \mathbb{C}$$

$$A = \left(A_{nm}\right)_{-N \le n,m \le N}$$

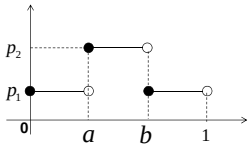
$$A_{nm} := \int_0^1 p(x)\phi_n'(x)\overline{\phi_m'(x)}\,dx + \int_0^1 q(x)\phi_n(x)\overline{\phi_m(x)}\,dx$$

$$\mathbf{x} = \left(u_n\right)_{-N \le n \le N}$$

$$A\mathbf{x} = \lambda_N \mathbf{x}$$

Example

p : piecewise constant, q : constant



$$A_{nm} := \int_0^1 p(x)\phi_n'(x)\overline{\phi_m'(x)}\,dx + \int_0^1 q\phi_n(x)\overline{\phi_m(x)}\,dx$$

$$\phi_n(x) = e^{i(2\pi n + \mu)x}$$

$$-N \le n, m \le N$$

$$A_{nn} = p_1(2\pi n + \mu)^2(a + 1 - b) + p_2(2\pi n + \mu)^2(b - a) + q$$

$$A_{nm} = \frac{p_1(2\pi n + \mu)(2\pi m + \mu)}{2\pi i(n - m)}\left\{e^{2\pi i(n-m)a} - 1 + e^{2\pi i(n-m)} - e^{2\pi i(n-m)b}\right\}$$

$$+ \frac{p_2(2\pi n + \mu)(2\pi m + \mu)}{2\pi i(n - m)}\left\{e^{2\pi i(n-m)b} - e^{2\pi i(n-m)a}\right\} \quad (n \ne m)$$

prog1.m
Compute an approximate eigenpair of $Ax = \lambda x$
Plot the eigenfunction corresponding to the minimum eigenvalue

prog2.m
Enclose an eigenpair of $Ax = \lambda x$

prog3.m
Construct a matrix

$B = (B_{nm})_{1 \le n,m \le K}, \quad B_{nm} = a(u_n, u_m), \quad C = (C_{nm})_{1 \le n,m \le K}, \quad C_{nm} = \langle u_n, u_m \rangle_{L^2}$

u_i : approximate eigenfunctions

Enclose an eigenpair of $Bx = \lambda Cx \quad (\text{e.g. } K = 5)$

$\text{prog1} \subset \text{prog2} \subset \text{prog3}$

25

```
%Eigenvalue computation for 1D interface problem
%-(pu')'+qu=lambda*u
%p=p1 on (0,a) and (b,1),   p=p2 on (a,b)
%q: constant

clear
p1=1;
p2=2;
a=0.3;
b=0.6;
mu=1;
N=100;
q=1;

A=zeros(2*N+1,2*N+1);

for n=-N:N
   A(n+N+1,n+N+1)=p1*(2*pi*n+mu)^2*(a+1-b)+p2*(2*pi*n+mu)^2*(b-a)+q;
end
  ⋮
```

26

Computation of eigenvalues:

[V,E]=eig(A); $\quad Ax = \lambda x$

[V,E]=eig(B,C); $\quad Bx = \lambda Cx$

$V = (v_1\ v_2 \cdots v_n), \quad E = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}$

Sort of eigenpairs:

```
>> [V,E]=eig(A);
>> Evec=diag(E);
>>[EP,I]=sort(Evec);
```

v1=V(:,I(1));

⟶ **v1=The eigenvector of A which corresponds to the minimum eigenvalue of A**

"sort" and "index vector"

```
>> X=[40 50 20]; [Y,I]=sort(X)
Y =
   20   40   50
I =
    3    1    2
```

27

Plot of an eigenfunction:

```
M=1000;
u=zeros(M,1);
x=linspace(0,1,M);
v1=V(:,1);

for k=-N:N
   for t=1:M
      u(t)=u(t)+v1(k+N+1)*exp((2*pi*k+mu)*i*x(t));
   end
end

figure(1);
plot(x,real(u));
xlim([0,1])

figure(2);
plot(x,imag(u));
xlim([0,1])
```

28

Enclosure of the 1st eigenvalue:

```
>> [V,E]=eig(A);
>> Evec=diag(E);
>> [EP,I]=sort(Evec);
>> [L,X]=verifyeig(A,EP(1),V(:,I(1)));
```

$Ax = \lambda x$

```
>> [V,E]=eig(B,C);
>> Evec=diag(E);
>> [EP,I]=sort(Evec);
>> [L,X]=verifyeig(B,EP(1),V(:,I(1)),C);
```

$Bx = \lambda Cx$

```
p1=intval('1');
p2=intval('2');
…
pi=acos(intval('-1'))
 …
A=intval(zeros(2*N+1,2*N+1));
…
[V,E]=eig(mid(A));
…
```

29