数理解析学特論III講義資料

高安亮紀　(筑波大学)*

2018 年 11 月 16 日

1. 浮動小数点数と誤差

計算機で扱える数「整数」と「浮動小数点数」について解説し、「丸め誤差」などの数値計算の誤差と、それを制御する「区間演算」を紹介する。

そもそも数値計算とは？

- 数値計算（数値解析）とは…数学の問題を有限桁の浮動小数点数を使って「数値的」に解く．数式処理とは違う
- 数値計算でできること，できないこと
- 数値計算は誤差との戦い
- 整数（integer）と浮動小数点数（binary32/binary64）

1.1. 整数型

Matlab 上で整数は

型	機械内表現
int8	8bit 整数
int16	16bit 整数
int32	32bit 整数
int64	64bit 整数

のような数値がある．

```
a = int32(10); disp(a)
```

```
   10
```

```
b = int32(2147483647)+int32(1); disp(b)
```

```
   2147483647
```

```
c = intmax('int32'); disp(c)
```

```
   2147483647
```

これから int32 の最大値が 2147483647 であることがわかる。実際 $2^{31}-1 = 2147483647$ である．これは int32 整数が次のようなビットパターンで表現されていることによる。

* 〒 1305-8573 茨城県つくば市天王台 1-1-1
e-mail: takitoshi@risk.tsukuba.ac.jp
web: http://www.risk.tsukuba.ac.jp/~takitoshi/

ビットパターン	数値
01111111111111111111111111111111	2147483647
00000000000000000000000000000010	2
00000000000000000000000000000001	1
00000000000000000000000000000000	0
11111111111111111111111111111111	-1
11111111111111111111111111111110	-2
10000000000000000000000000000000	-2147483648

このような負の数の表現形式を「**2の補数形式**」と呼ぶ．32個の各bitが次のような重みをもっていると考えられる．

$$\boxed{-2^{31}}\boxed{2^{30}}\boxed{2^{29}}\cdots\boxed{2^{1}}\boxed{2^{0}}$$

2の補数形式の場合，nビットで-2^{n-1}〜$2^{n-1}-1$の範囲の数を表現できる．つまり

型	表現範囲
int8	-128〜127
int16	-32768〜32767
int32	-2147483648〜2147483647
int64	-9223372036854775808〜9223372036854775807

の範囲の整数が表せる．

```
b = int64(2147483647)+1; disp(b)
```

```
   2147483648
```

```
b = int64(9223372036854775807)+1; disp(b)
```

```
   9223372036854775807
```

1.2. 浮動小数点数

浮動小数点数（binary64, binary32）は,「浮動小数点形式」と呼ばれる形式で表現できる．例えば,「1234.5」を「1.2345×10^3」のように，小数点の位置を1番左の数値と左から2番目の数値の間に移動（「正規化」と呼ぶ）し，それに指数を掛けた形式で数を表現する．この「1.2345」の部分を「**仮数部**」,「10^3」の部分（厳密には ^{3}）を「**指数部**」という．

浮動小数点数は仮数部の長さ，指数部の長さ，指数が2，10，16など，多様な規格が考えられる．そこで1985年にWilliam Kahanが中心となって

IEEE 754: Standard for Binary Floating-Point Arithmetic

という標準規格が制定された．最近では世に出るハードウェアのほぼ全てがこの規格に従っている．

1.2.1. 倍精度 (binary64)

倍精度は，符号（±）に1ビット．指数部に11ビット，仮数部に52ビットを使う． 全部で64ビット=8バイトである．

1（符号）	11（指数部）	52（仮数部）

- 符号は，0なら正，1なら負
- 指数部は「$\times 2^{\text{指数}}$」の指数の部分に1023を加えたものが11ビット符号無しの整数の形で格納されている
- 仮数部は，実際の仮数部の先頭の「1」を取り除いた残りが格納されている．仮数部の先頭は必ず1にしてメモリに格納しないことで1ビット分精度を稼いでいる

数値xは

$$x = \pm\, 1.d_1 d_2 \cdots d_{52} \times 2^m = \pm \left(\frac{1}{2^0} + \frac{d_1}{2^1} + \frac{d_2}{2^2} + \cdots + \frac{d_{52}}{2^{52}} \right) 2^e_{(10)}$$

と書ける（$-1022 \leq e \leq 1023$，$m : e + 1023$の2進表現）．

例えば，5.25は2進数で書くと

$$101.01_{(2)} = \left(\frac{1}{2^0} + \frac{0}{2^1} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{1}{2^4} \right) \times 2^2_{(10)}$$

であるから，計算機内では

0	10000000001	0101000000000000000000000000000000000000000000000000

のように格納されている。指数部の「1000000001」は、「2+1023=1025」を2進数にしたもの．これは**正規化数**と呼ばれる数の表現．

```
x = 5.25;
disp(float2bin(x))
disp(float2bin(5.24))
```

```
0100000000010101000000000000000000000000000000000000000000000000
0100000000010100111101011100001010001111010111000010100011110110
```

```
binx = float2bin(x);
sign = binx(1); disp(sign)
exp  = binx(2:12); disp(exp)
frac = binx(13:end); disp(frac)
```

```
0
10000000001
0101000000000000000000000000000000000000000000000000
```

符号はsign $= 0$で正の数, 指数部はexp-1023=1025-1023=2，仮数部は
frac=$(1010000000000000000000000000000000000000000000000000)_2$
で2進数表示され，実際の値は1.3125. 元の浮動小数点数に戻すと

$$(-1)^{\text{sign}} * \text{dfrac} * 2^{\text{exp}-1023} = 5.25$$

で確かに元の数に戻ることがわかる．

```
dfrac = 1 + sum((frac-'0')./(2.^(1:52))); disp(dfrac)
dexp  = sum((exp-'0').*(2.^(10:-1:0)))-1023;
x = (-1)^(str2double(sign))*dfrac*2^(dexp); disp(x)
```

```
   1.312500000000000
   5.250000000000000
```

次に e の範囲 $-1022 \leq e \leq 1023$ に注目する．$e+1023$ が 11 ビット符号無しの整数なら範囲は 0〜2047 であるが，e 範囲から $e+1023=0$ のときと $e+1023=2047$ のときが使われていない．これらは**特殊な数を表す**のに使われる．それらは

- 零
- 無限大 (Inf)
- NaN (Not a Number)
- 非正規化数

と呼ばれる．

零

指数部が $e+1023=0$ かつ仮数部が 0 のとき．

$$\pm\, 0.00\cdots0 \times 2^0 = \pm\left(\frac{0}{2^0}+\frac{0}{2^1}+\frac{0}{2^2}+\cdots+\frac{0}{2^{52}}\right)2^{-1023}{}_{(10)}.$$

```
x = 0.0; disp(x), disp(float2bin(x))
x = -0.0; disp(x), disp(float2bin(x))
```

```
     0
0000000000000000000000000000000000000000000000000000000000000000
     0
1000000000000000000000000000000000000000000000000000000000000000
```

無限大

$e+1023=2047$ かつ仮数部が 0 のとき，$\pm\infty$ を表す．

$$\pm\, 0.00\cdots0 \times 2^m = \pm\left(\frac{0}{2^0}+\frac{0}{2^1}+\frac{0}{2^2}+\cdots+\frac{0}{2^{52}}\right)2^{1024}{}_{(10)},$$

m: 2047 の 2 進表現．このとき**オーバーフロー**が起こるという．

```
x = inf; disp(x), disp(float2bin(x))
disp(-x), disp(float2bin(-x))
```

```
   Inf
0111111111110000000000000000000000000000000000000000000000000000
   -Inf
1111111111110000000000000000000000000000000000000000000000000000
```

NaN (Not a Number)

$e + 1023 = 2047$ かつ仮数部が 0 でないとき．負数の平方根など，不可能な演算の結果を表すのに使われる．

```
x = inf; disp(float2bin(x))
x = nan; disp(x), disp(float2bin(x))
x = inf-inf; disp(x), disp(float2bin(x))
x = inf/inf; disp(x), disp(float2bin(x))
```

```
0111111111110000000000000000000000000000000000000000000000000000
   NaN
1111111111111000000000000000000000000000000000000000000000000000
   NaN
1111111111111000000000000000000000000000000000000000000000000000
   NaN
1111111111111000000000000000000000000000000000000000000000000000
```

正規化数の最大最小

正の正規化数の最大の数は，$e + 1023 = 2046$, かつ仮数部のビットが全て 1 のとき．

$$\left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \cdots + \frac{1}{2^{52}}\right) 2^{1023}{}_{(10)} = 2^{1024} - 2^{971} \approx 10^{308.25}$$

であり，これを 1 ビットでも超えたら無限大になる．これを**オーバーフロー**という．

```
x = (2^53-1)*2^971; disp(x), disp(float2bin(x))
x = (2^53)*2^971; disp(x), disp(float2bin(x))
```

```
    1.797693134862316e+308
0111111111101111111111111111111111111111111111111111111111111111
    Inf
0111111111110000000000000000000000000000000000000000000000000000
```

正の正規化数の最小の数は，$e + 1023 = 1$, かつ仮数部のビットが全て 0 のとき．

$$\left(\frac{1}{2^0} + \frac{0}{2^1} + \frac{0}{2^2} + \cdots + \frac{0}{2^{52}}\right) 2^{-1022}{}_{(10)} = 2^{-1022} \approx 10^{-307.65}$$

であり，これを下回ると**アンダーフロー**というが，IEEE 754 では，ここでアンダーフローさせないで「悪あがき」をする．

非正規化数

指数部が$e+1023=0$かつ仮数部が0でないとき，仮数部の最初の桁を0にして

$$\pm\, 0.d_1d_2\cdots d_{52}\times 2^0 = \pm\left(\frac{0}{2^0}+\frac{d_1}{2^1}+\frac{d_2}{2^2}+\cdots+\frac{d_{52}}{2^{52}}\right)2^{-1022}{}_{(10)}.$$

という数の表現をする． 非正規化数は文字通り「正規化していない」数．

```
x = 2^(-1022); disp(x), disp(float2bin(x))
disp(float2bin(x-2^(-1074)))
```

```
    2.225073858507201e-308
0000000000010000000000000000000000000000000000000000000000000000
0000000000001111111111111111111111111111111111111111111111111111
```

漸近アンダーフロー

正規化数の最小数の最終bitを1だけ減じると

$$\left(\frac{0}{2^0}+\frac{1}{2^1}+\frac{1}{2^2}+\cdots+\frac{1}{2^{52}}\right)2^{-1022}{}_{(10)}$$

となり，これを正規化すると

$$\left(\frac{1}{2^0}+\frac{1}{2^1}+\frac{1}{2^2}+\cdots+\frac{0}{2^{52}}\right)2^{-1023}{}_{(10)}$$

となって，指数部の下限$-1022\le e$を超えてしまう．そこで，「2^{-1022}を下回ったら正規化をやめて指数部を2^{-1022}に固定して仮数部の最初の桁を0としてみて格納する」とする．これが非正規化数であり，

$$\left(\frac{0}{2^0}+\frac{1}{2^1}+\frac{1}{2^2}+\cdots+\frac{1}{2^{52}}\right)2^{-1022}{}_{(10)}$$

$$\left(\frac{0}{2^0}+\frac{1}{2^1}+\frac{1}{2^2}+\cdots+\frac{0}{2^{52}}\right)2^{-1022}{}_{(10)}$$

$$\vdots$$

$$\left(\frac{0}{2^0}+\frac{1}{2^1}+\frac{0}{2^2}+\cdots+\frac{0}{2^{52}}\right)2^{-1022}$$

$$\left(\frac{0}{2^0}+\frac{0}{2^1}+\frac{1}{2^2}+\cdots+\frac{1}{2^{52}}\right)2^{-1022}{}_{(10)}$$

$$\vdots$$

$$\left(\frac{0}{2^0}+\frac{0}{2^1}+\frac{0}{2^2}+\cdots+\frac{1}{2^{52}}\right)2^{-1022}{}_{(10)}=2^{-1074}\approx 10^{-323.31}$$

のような数が表現できる．ただし，2^{-1022}と2^{-1074}の間の数は，本来53ビットあるべき仮数部の長さが52ビット～1ビットまで減ってしまっており，精度が低下していることに注意．

```
x = 2^(-1074); disp(x), disp(float2bin(x))
disp(x/2), disp(float2bin(x/2))
```

```
    4.940656458412465e-324
0000000000000000000000000000000000000000000000000000000000000001
    0
0000000000000000000000000000000000000000000000000000000000000000
```

1.2.2. 浮動小数点数まとめ

倍精度　(binary64) は

	仮数部が 0	仮数部が 0 でない
$e + 1023 = 0$	± 0	非正規化数
$1 \leq e + 1023 \leq 2046$	正規化数	正規化数
$e + 1023 = 2047$	$\pm\infty$	NaN

単精度 (binary32) は

	仮数部が 0	仮数部が 0 でない
$e + 127 = 0$	± 0	非正規化数
$1 \leq e + 127 \leq 254$	正規化数	正規化数
$e + 127 = 255$	$\pm\infty$	NaN

今後，浮動小数点数全体の集合を $\mathbb{F}$ と表すことにする．特に断りがなければ，浮動小数点数は倍精度浮動小数点数（64bit）とする．IEEE754 では **binary64** と呼ばれている．

1.3. 丸め誤差

浮動小数点数を使用した演算に混入する誤差について述べる。浮動小数点数同士の演算（加減乗除など）の結果は，浮動小数点数で表せるとは限らない．例えば，10 進数で仮数部 3 桁の浮動小数点演算を考え，2/3 を計算すると，

$$2.00 \times 10^0 / 3.00 \times 10^0 = 0.66666666... \times 10^0$$

となり，仮数部 3 桁に収まらない．仮数部の 4 桁目で四捨五入を行うと

$$6.67 \times 10^{-1}$$

となる．このときの計算値と真値との差

$$6.67 \times 10^{-1} - 6.6666666... \times 10^{-1} = 3.3333333 \times 10^{-4}$$

が丸め誤差である．

IEEE 754 は 2 進数浮動小数点数なので，基本的には 0 捨 1 入で丸められる．例えば，10 進数の「0.1」を IEEE 754 の double に変換してみると

$$\begin{aligned} 0.1_{(10)} &= 0.0001100110011001100110011\ldots_{(2)} \\ &= 1.10011001100110011\ldots \times 2^{m}, \end{aligned}$$

m は $-4+1023=1019$ の 2 進表現，つまり $m=01111111011$. 仮数部は無限小数になっているのでそのまま格納出来ない． 小数点以下を 52bit 以内とそれ以降で区切って表示すると

1001100110011001100110011001100110011001100110011001	10011001100....

となり，はみ出た部分の先頭が「1」なので、0 捨 1 入で繰り上げる．最終的には， 10 進数の「0.1」は

0	01111111011	1001100110011001100110011001100110011001100110011010

のように格納されている．

```
x = 0.1; disp(float2bin(x))
```

```
0011111110111001100110011001100110011001100110011001100110011010
```

10 進数の 0.1 は計算機には正確に格納できず，少しだけ 0.1 より大きい値で格納されており，**丸め誤差**が含まれている

$$\frac{1}{10} \approx 0.1000000000000000055511151231257827021181583404541015625$$

```
x = 0.1;
fprintf('%.55f\n',x)
for i=2:100
  x = x+0.1;
end
fprintf('%.55f\n',x)
```

```
0.1000000000000000055511151231257827021181583404541015625
9.9999999999999804600747665972448885440826416015625000000
```

1.4. 丸めモード

IEEE754 では，4 つの丸めモードを用意している．$\tilde{x}$ を実数（$\tilde{x} \in \mathbb{R}$）とする．いま実数 $\tilde{x}$ が

$$\tilde{x} = \pm\left(\frac{1}{2^0} + \frac{d_1}{2^1} + \frac{d_2}{2^2} + \ldots\right) 2^{e}_{(10)}$$

であるときに，その近似

$$x = \pm\left(\frac{1}{2^0} + \frac{d_1}{2^1} + \frac{d_2}{2^2} + \cdots + \frac{d_{52}}{2^{52}}\right) 2^{e}_{(10)} \in \mathbb{F}$$

を採用する丸めは**原点方向への丸め**（切り捨て）と呼ばれている．

また $\tilde{x}$ の近似として誤差 $|x-\tilde{x}|$ の値が最小になる $x \in \mathbb{F}$，すなわち

$$|x-\tilde{x}| = \min_{y \in \mathbb{F}} |y-\tilde{x}|$$

をみたす $x \in \mathbb{F}$ を採用する方法を**最近点への丸め**という．しかし $\tilde{x}$ が 2 つの浮動小数点数

$$x_1 = \left(\frac{1}{2^0} + \frac{d_1}{2^1} + \frac{d_2}{2^2} + \cdots + \frac{d_{52}}{2^{52}}\right) 2^e_{(10)}, \quad x_2 = \left(\frac{1}{2^0} + \frac{d_1}{2^1} + \frac{d_2}{2^2} + \cdots + \frac{d_{52}+1}{2^{52}}\right) 2^e_{(10)}$$

の中点となると $|x_1-\tilde{x}| = |x_2-\tilde{x}|$ が成り立ち，x は一つに決まらない．このときは**仮数部の最後のビットが 0** になる方に丸めが実行される．これを**最近偶数への丸め**という．

```
eps = 2^(-52);
x = 1 + eps/4;   fprintf('%.50f\n',x), disp(float2bin(x))
y = 1 + 3*eps/4; fprintf('%.50f\n',y), disp(float2bin(y))
z = 1 + eps/2;   fprintf('%.50f\n',z), disp(float2bin(z))
```

```
1.00000000000000000000000000000000000000000000000000
0011111111110000000000000000000000000000000000000000000000000000
1.00000000000000022204460492503130808472633361816406
0011111111110000000000000000000000000000000000000000000000000001
1.00000000000000000000000000000000000000000000000000
0011111111110000000000000000000000000000000000000000000000000000
```

```
x = 2;         fprintf('%.50f\n',x), disp(float2bin(x))
y = 2 - eps;   fprintf('%.50f\n',y), disp(float2bin(y))
z = 2 - eps/2; fprintf('%.50f\n',z), disp(float2bin(z))
```

```
2.00000000000000000000000000000000000000000000000000
0100000000000000000000000000000000000000000000000000000000000000
1.99999999999999977795539507496869191527366638183594
0011111111111111111111111111111111111111111111111111111111111111
2.00000000000000000000000000000000000000000000000000
0100000000000000000000000000000000000000000000000000000000000000
```

次は $\tilde{x}$ よりも必ず大きな $a \in \mathbb{F}$，すなわち

$$a = \min\{x \in \mathbb{F} : x \geq \tilde{x}\}$$

をみたす $a \in \mathbb{F}$ を採用する方法を $+\infty$ **方向への丸め**（上向き丸め）という．逆に $\tilde{x}$ よりも必ず小さな $b \in \mathbb{F}$，すなわち

$$b = \max\{x \in \mathbb{F} : x \leq \tilde{x}\}$$

をみたす $b \in \mathbb{F}$ を採用する方法を $-\infty$ **方向への丸め**（下向き丸め）という．これらより

$$b \leq \tilde{x} \leq a$$

が常に成立する．特に精度保証付き数値計算ではこの 2 つの丸めモードを利用することで，厳密な包含を得ることができるようになる．まとめると

- 最近点への丸め（デフォルト）: $\tilde{x}$ に最も近い浮動小数点数に丸める．もし2点あるならば仮数部の最後のビットが0である浮動小数点数に丸める．
- $+\infty$ 方向への丸め：$\tilde{x}$ 以上の浮動小数点数の中で最も小さい浮動小数点数に丸める．
- $-\infty$ 方向への丸め：$\tilde{x}$ 以下の浮動小数点数の中で最も大きい浮動小数点数に丸める．
- 原点方向への丸め：絶対値が $\tilde{x}$ 以下の浮動小数点数の中で，$\tilde{x}$ に最も近いものに丸める．

1.5. その他の誤差いろいろ

丸め誤差は実数を浮動小数点数で近似する際の不正確さであった．ここでは浮動小数点数を用いた演算の問題点を紹介する．

1.5.1. 桁落ち

極めて近い数どうしの減算によって，誤差が著しく大きくなってしまう現象．2つの浮動小数点数

$$x = \left(\frac{1}{2^0} + \frac{d_1}{2^1} + \cdots + \frac{d_p}{2^p} + \frac{1}{2^{p+1}} + \cdots + \frac{b_{p+2}}{2^{p+2}} + \cdots + \frac{b_{52}}{2^{52}} \right) 2^e_{(10)},$$

$$y = \left(\frac{1}{2^0} + \frac{d_1}{2^1} + \cdots + \frac{d_p}{2^p} + \frac{0}{2^{p+1}} + \cdots + \frac{c_{p+2}}{2^{p+2}} + \cdots + \frac{c_{52}}{2^{52}} \right) 2^e_{(10)}$$

が $x > y$ とし，仮数部の最初から p ビットが等しいとする．このとき

$$x - y = \left(\frac{1}{2^0} + \frac{b_{p+2} - c_{p+2}}{2^1} + \cdots + \frac{b_{52} - c_{52}}{2^{52-p-1}} \right) 2^{e-p-1}_{(10)}$$

これよりもともと 52 個あった仮数部の情報が，$52 - p$ 個に減っている．例を挙げよう．$b > 0$ とし，2次方程式 $x^2 + bx + c = 0$ の解の公式

$$x_1 = \frac{-b + \sqrt{b^2 - 4c}}{2},\ x_2 = \frac{-b - \sqrt{b^2 - 4c}}{2}$$

を考える．いまもしも $b^2 \gg c$ となるならば，b と $\sqrt{b^2 - 4c}$ が近い数になるので，x_1 の分子の計算で，桁落ちが起こる．

```
b = 1e15; c = 1e14;
x1 = (-b+sqrt(b^2-4*c))/2; disp(x1), disp(x1^2+b*x1+c)
x2 = 2*c/(-b-sqrt(b^2-4*c)); disp(x2), disp(x2^2+b*x2+c)

-0.125000000000000
-2.499999999999998e+13
-0.100000000000000
0
```

1.5.2. 情報落ち

絶対値の大きさが極端に違う 2 数の加減算を行った時，小さいほうの数値の下位の桁が失われてしまう現象.

```
disp((3.14159265358979+1e10)-1e10)
disp(3.14159265358979 + (1e10-1e10))
```

```
3.141592025756836
3.141592653589790
```

```
disp(1e48+543.2-1e48-1e36+123.4+1e36)
disp(1e48-1e48-1e36+1e36+543.2+123.4)
disp(1e48-1e36-1e48+1e36+543.2+123.4)
```

```
0
6.666000000000000e+02
-3.923098441916162e+30
```

1.5.3. 打ち切り誤差

無限回行うべき計算を有限回の計算で置き換えることにより生じる誤差．計算機は有限回の四則演算しかできない．そのため，無限級数や収束列のような値を求めるためには，有限項で打ち切った近似値を用いる．その際に誤差が生じる.（例）Taylor 展開の打ち切り誤差，Newton 法の打ち切り誤差，数値積分の打ち切り誤差

```
disp(exp_taylor(20))
disp(exp(20))
disp(exp_taylor(-20))
disp(exp(-20))

function s = exp_taylor(x)
  s = 0; t = 1; i = 1;
  while 1
    s = s + t;
    if abs(t) < abs(s) * 1e-15, break; end
    t = (x / i)*t;
    i = i+1;
  end
end
```

```
4.851651954097903e+08
4.851651954097903e+08
6.147561828914626e-09
2.061153622438558e-09
```

2. 区間演算

区間の表現，区間演算について述べる．精度保証付き数値計算の基本的な原理は，実数値で与えられる真の値の上限と下限を浮動小数点数により浮動小数点演算を用いて包み込むことである．

$\mathbb{R}$ 上の区間を

$$\boldsymbol{a} := \{x \in \mathbb{R} : \underline{a} \le x \le \overline{a},\ \underline{a}, \overline{a} \in \mathbb{R}\} = [\underline{a}, \overline{a}]$$

と表し，区間の全体を $\mathbb{IR}$ とする． この時 $\underline{a}$ を区間の下端，$\overline{a}$ を区間の上端という． さらに区間の

直径（diameter）：	$\mathrm{d}(\boldsymbol{a}) = \overline{a} - \underline{a} \in \mathbb{R}$
半径（radious）：	$\mathrm{rad}(\boldsymbol{a}) = \frac{\overline{a}-\underline{a}}{2} \in \mathbb{R}$
中点（center）：	$\mathrm{mid}(\boldsymbol{a}) = \frac{\overline{a}+\underline{a}}{2} \in \mathbb{R}$
最小絶対値:	$\mathrm{mig}(\boldsymbol{a}) = \min\{\lvert a\rvert : a \in \boldsymbol{a}\} \in \mathbb{R}$
最大絶対値:	$\mathrm{mag}(\boldsymbol{a}) = \max\{\lvert a\rvert : a \in \boldsymbol{a}\} \in \mathbb{R}$

をそれぞれ表すとする．

```
a = infsup(-1,2); disp(a)
disp(['d(a) = ', num2str(diam(a))])
disp(['rad(a) = ', num2str(rad(a))])
disp(['mid(a) = ', num2str(mid(a))])
disp(['mig(a) = ', num2str(mig(a))])
disp(['mag(a) = ', num2str(mag(a))])
```

```
[  -1.00000000000000,   2.00000000000000]
d(a) = 3
rad(a) = 1.5
mid(a) = 0.5
mig(a) = 0
mag(a) = 2
```

2.1. 区間演算（上端下端型）

区間 $\boldsymbol{X} = [a, b]$, $\boldsymbol{Y} = [c, d]$ に対して，四則演算を定義する：

$$\begin{aligned}
\boldsymbol{X} + \boldsymbol{Y} &= [a + c, b + d] \\
\boldsymbol{X} - \boldsymbol{Y} &= [a - d, b - c] \\
\boldsymbol{X} \times \boldsymbol{Y} &= [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}] \\
\boldsymbol{X} \div \boldsymbol{Y} &= [\min\{a/c, a/d, b/c, b/d\}, \max\{a/c, a/d, b/c, b/d\}]
\end{aligned}$$

$\times$, $\div$ は場合分けをする．

	$c > 0$	$0 \in \boldsymbol{Y}$	$d < 0$
$a > 0$	$[ac, bd]$	$[bc, bd]$	$[bc, ad]$
$0 \in \boldsymbol{X}$	$[ad, bd]$	A	$[bc, ac]$
$b < 0$	$[ad, bc]$	$[ad, ad]$	$[bd, ac]$

ただし A$=[\min\{ad,bc\},\max\{ad,bc\}]$.

注意 2.1. 区間内全ての要素について演算を行うため無限回の計算が必要のように思えるが，これをまとめて実現するのが区間演算．さらに除算で割る区間$\boldsymbol{Y}$ に0が含まれる（$0\in\boldsymbol{Y}$）と演算結果は無限大を含むことになる．

```
X = infsup(-2,1);
Y = infsup(1,3);
disp(X+Y)
disp(X-Y)
disp(X*Y)
disp(X/Y)
Z = Y/X; disp(Z)
```

```
[  -1.00000000000000,   4.00000000000000]
[  -5.00000000000000,   0.00000000000000]
[  -6.00000000000000,   3.00000000000000]
[  -2.00000000000000,   1.00000000000000]
+/-Inf
```

最後の+/-Inf は区間$[-\infty,\infty]$を表している。

包含関係の単調性が成立する．すなわち$\boldsymbol{X}_1\subseteq\boldsymbol{X}_2$ かつ$\boldsymbol{Y}_1\subseteq\boldsymbol{Y}_2$ ならば$\boldsymbol{X}_1\circ\boldsymbol{Y}_1\subseteq\boldsymbol{X}_2\circ\boldsymbol{Y}_2$ が成立する．ここで，$\circ\in\{+,-,\times,\div\}$とする．

さらに $+,\times$ に関しては，交換則と結合則が成立する．

$$\boldsymbol{X}\circ\boldsymbol{Y}=\boldsymbol{Y}\circ\boldsymbol{X},\ \boldsymbol{X}\circ(\boldsymbol{Y}\circ\boldsymbol{Z})=(\boldsymbol{X}\circ\boldsymbol{Y})\circ\boldsymbol{Z},\ \circ\in\{+,\times\}.$$

しかし，加法と乗法の逆元は存在しない．すなわち分配則が成立しない．劣分配則のみ成立する．

$$\boldsymbol{X}(\boldsymbol{Y}+\boldsymbol{Z})\subseteq\boldsymbol{X}\boldsymbol{Y}+\boldsymbol{X}\boldsymbol{Z}.$$

```
X = infsup(-1,1);
Y = infsup(1,2);
Z = infsup(-2,1);
disp(X*(Y+Z))
disp(X*Y+X*Z)
disp(in(X*(Y+Z),X*Y+X*Z))
```

```
[  -3.00000000000000,   3.00000000000000]
[  -4.00000000000000,   4.00000000000000]
1
```

2.2. 中心半径型区間演算

上端下端型区間の他に区間を中心と半径で表す型もある．それを**中心半径型区間**という．中心半径型（mid-rad 型）区間は実数 $x \in \mathbb{R}$ に対して，区間の中心を x_c，半径を x_r としたとき

$$\boldsymbol{x} = \langle x_c, x_r \rangle = \{x : x_c - x_r \leq x \leq x_c + x_r\}$$

と表される．中心半径型区間と上端下端型区間の間には次のような関係が成り立つ：

$$\boldsymbol{x} = [\underline{x}, \overline{x}] = [x_c - x_r, x_c + x_r] = \left\langle \frac{\underline{x} + \overline{x}}{2}, \frac{\underline{x} - \overline{x}}{2} \right\rangle = \langle x_c, x_r \rangle = \boldsymbol{x}$$

中心半径型区間 $\boldsymbol{x} = \langle x_c, x_r \rangle$, $\boldsymbol{y} = \langle y_c, y_r \rangle$ に対して，四則演算を定義する．

$$\begin{aligned}
\boldsymbol{x} + \boldsymbol{y} &= \langle x_c + y_c, x_r + y_r \rangle \\
\boldsymbol{x} - \boldsymbol{y} &= \langle x_c - y_c, x_r + y_r \rangle \\
\boldsymbol{x} \times \boldsymbol{y} &\subseteq \langle x_c y_c, |x_c| y_r + |y_c| x_r + x_r y_r \rangle \\
\boldsymbol{x} \div \boldsymbol{y} &= \frac{\boldsymbol{x} \times \boldsymbol{y}}{\underline{y}\overline{y}} = \frac{\boldsymbol{x} \times \boldsymbol{y}}{y_c^2 - y_r^2}
\end{aligned}$$

ちなみに乗算は $\mathrm{sgn}(\dot{})$ を括弧内の符号を返す関数とすると

$$\boldsymbol{x} \times \boldsymbol{y} = \langle x_c y_c + \delta_1, \delta_2 \rangle$$

と書ける．ここで

$$\begin{aligned}
\delta_1 &= \mathrm{sgn}(x_c y_c) \min \{x_r |y_c|, |x_c| y_r, x_r y_r\}, \\
\delta_2 &= \max \{x_r (|y_c| + y_r), (|x_c| + x_r) y_r, x_r |y_c| + |x_c| y_r\}
\end{aligned}$$

である．

```
X = midrad(0,1);
Y = midrad(1.5,0.5);
Z = midrad(-0.5,1.5);
format midrad
disp(X+Y)
disp(X-Y)
disp(X*Y)
disp(X/Y)
```

```
<   1.50000000000000,  1.50000000000000>
<  -1.50000000000000,  1.50000000000000>
<   0.00000000000000,  2.00000000000000>
<   0.00000000000000,  1.00000000000000>
```

2.3. 関数の値域評価

$\boldsymbol{I} \in \mathbb{IR}$ をある区間とし，$f : D \subset \mathbb{R} \to \mathbb{R}$ を領域 D で連続な関数とする． このとき関数 f は $\mathbb{IR}$ 上の関数として拡張できる．

$$f(\boldsymbol{I}) = \{f(x) : x \in \boldsymbol{I}\}$$

この $f(\boldsymbol{I})$ は関数の値域で，これを厳密に計算することは非線形関数の場合不可能である．よって

$$f(\boldsymbol{I}) \subseteq [a, b]$$

となる区間 $[a, b]$ で関数の値域を包含する．これを f の区間 $\boldsymbol{I}$ における**区間拡張**（区間拡張は一意的でないことに注意！）といい，$f_{[]}(\boldsymbol{I}) = [a, b]$ と表す．

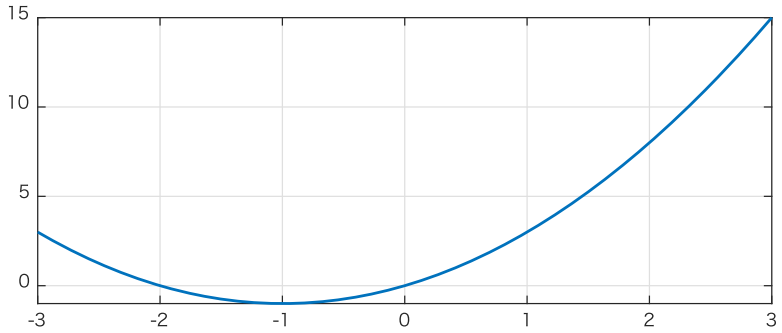
注意 2.2. IEEE 754 標準規格における丸めに従っているのは「**四則演算・平方根**」だけ．それ以外の関数（例えば，sin，cos などは）丸めの向きの変更が不可能（**自作するか，丸めに対応し作成されたものを使う**こと）．ただし，デフォルトの関数も精度が十分に高く作られているため，精度保証などを考えない場合はそれで十分．

2.3.1. 例

例えば，関数 $f(x) = x^2 + 2x$ を考える．関数に区間を単純に代入すると

$$\begin{aligned} x &\in [0.9, 1.1] \\ x^2 &\in [0.81, 1.21] \\ 2x &\in [1.8, 2.2] \\ x^2 + 2x &\in [2.61, 3.41] \end{aligned}$$

```
format infsup
f = @(x) x.^2+2*x;
LW = 'linewidth'; lw = 1.6;
fplot(f,[-3,3],LW,lw)
f(infsup(0.9,1.1))
```



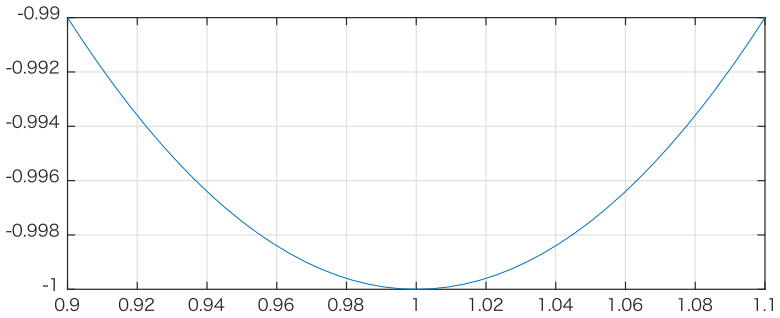

```
intval ans =
[   2.60999999999999,   3.41000000000001]

disp(f(0.9))
disp(f(1.1))
```

```
2.610000000000000
3.410000000000000
```

$f(x) = x^2 + 2x$ のときは値域評価が区間拡張を利用しても精度が良い．一方で $f(x) = x^2 - 2x$ のときは

$$\begin{aligned} x &\in [0.9, 1.1] \\ x^2 &\in [0.81, 1.21] \\ 2x &\in [1.8, 2.2] \\ x^2 - 2x &\in [-1.39, -0.59] \text{（幅 } 0.8!\text{）} \end{aligned}$$

```
f = @(x) x.^2-2*x;
fplot(f,[0.9,1.1])
disp(f(0.9))
disp(f(1))
disp(f(1.1))
```




```
-0.990000000000000
-1
-0.990000000000000
```

これから値域は $[-1, -0.99]$，一方で区間演算による区間拡張は $[-1.39, -0.59]$．だいぶ過大評価になってしまう．そこで区間幅を改良したい。

アイディア 1. $f(x) = x(x-2)$;

$$\begin{aligned} &x \in [0.9, 1.1] \\ &x(x-2) = [0.9, 1.1] \times [-1.1, -0.9] = [-1.21, -0.81] \text{（幅 } 0.4\text{）} \end{aligned}$$

大分改善された．もう一声！

アイディア 2. $f(x) = (x-1)^2 - 1$;

$$\begin{aligned} x &\in [0.9, 1.1] \\ (x-1)^2 - 1 &= [-0.1, 0.1]^2 - 1 \\ &= [0, 0.01] - 1 \\ &= [-1, -0.99] \text{（幅 } 0.01\text{）} \end{aligned}$$

```
x = infsup(0.9,1.1);
disp(x^2-2*x)
disp(x*(x-2))
disp((x-1)^2-1)
```

```
[  -1.39000000000001,  -0.58999999999999]
[  -1.21000000000001,  -0.80999999999999]
[  -1.00000000000000,  -0.98999999999999]
```

2.3.2. 区間幅の抑制のための工夫

1. 区間分割

$$\boldsymbol{I} = \boldsymbol{I}_1 \cup \boldsymbol{I}_2$$

と考えて，$f_{[]}(\boldsymbol{I}_1) \cup f_{[]}(\boldsymbol{I}_2)$ を計算する．

2. 平均値形式
区間拡張 $f_{[]}(\boldsymbol{I})$ を

$$f_{[]}(\boldsymbol{I}) = f(c) + f'_{[]}(\boldsymbol{I})(\boldsymbol{I} - c),\ c = \mathrm{mid}(\boldsymbol{I})$$

によって得る．ただし $f'_{[]}(\boldsymbol{I})$ は区間 $\boldsymbol{I}$ における f の 1 階微分の区間拡張．

2.4. 機械区間演算

区間演算をコンピュータで実現するには $\mathbb{R}$ の代わりに $\mathbb{F}$ を使った区間が必要．そのような区間全体を

$$\mathbb{IF} := \{\boldsymbol{x} \in \mathbb{IR} : \underline{x},\ \overline{x} \in \mathbb{F}\}$$

と定義する．IEEE754 規格に準拠したシステム上では演算後の丸めの向きを制御することができる．演算結果が浮動小数点数でない場合，丸めの向きを制御して計算する．いま $a, b \in \mathbb{F}$ に対して，$\circ \in \{+, -, \times, \div\}$ として

$$\mathtt{fl}_{\triangledown}(a \circ b) := \max\{x \in \mathbb{F} : x \le a \circ b\}\ (\text{下向き丸め})$$
$$\mathtt{fl}_{\triangle}(a \circ b) := \min\{x \in \mathbb{F} : x \ge a \circ b\}\ (\text{上向き丸め})$$

とすると

$$\mathtt{fl}_{\triangledown}(a \circ b) \le a \circ b \le \mathtt{fl}_{\triangle}(a \circ b)$$

が成立する．$\boldsymbol{X} = [a, b]$, $\boldsymbol{Y} = [c, d]$ $(a, b, c, d \in \mathbb{F})$ に対して，機械区間演算は次のように実現できる．

$$\boldsymbol{X} + \boldsymbol{Y} = [\mathtt{fl}_{\triangledown}(a + c)\,, \mathtt{fl}_{\triangle}(b + d)]$$
$$\boldsymbol{X} - \boldsymbol{Y} = [\mathtt{fl}_{\triangledown}(a - d)\,, \mathtt{fl}_{\triangle}(b - c)]$$
$$\boldsymbol{X} \times \boldsymbol{Y} = [\mathtt{fl}_{\triangledown}(\min\{ac, ad, bc, bd\})\,, \mathtt{fl}_{\triangle}(\max\{ac, ad, bc, bd\})]$$
$$\boldsymbol{X} \div \boldsymbol{Y} = [\mathtt{fl}_{\triangledown}(\min\{a/c, a/d, b/c, b/d\})\,, \mathtt{fl}_{\triangle}(\max\{a/c, a/d, b/c, b/d\})]$$

$\boldsymbol{X} \times \boldsymbol{Y}$	$c > 0$	$0 \in \boldsymbol{Y}$	$d < 0$
$a > 0$	$[\mathtt{fl}_{\bigtriangledown}(ac)\,, \mathtt{fl}_{\bigtriangleup}(bd)]$	$[\mathtt{fl}_{\bigtriangledown}(bc)\,, \mathtt{fl}_{\bigtriangleup}(bd)]$	$[\mathtt{fl}_{\bigtriangledown}(bc)\,, \mathtt{fl}_{\bigtriangleup}(ad)]$
$0 \in \boldsymbol{X}$	$[\mathtt{fl}_{\bigtriangledown}(ad)\,, \mathtt{fl}_{\bigtriangleup}(bd)]$	B	$[\mathtt{fl}_{\bigtriangledown}(bc)\,, \mathtt{fl}_{\bigtriangleup}(ac)]$
$b < 0$	$[\mathtt{fl}_{\bigtriangledown}(ad)\,, \mathtt{fl}_{\bigtriangleup}(bc)]$	$[\mathtt{fl}_{\bigtriangledown}(ad)\,, \mathtt{fl}_{\bigtriangleup}(ad)]$	$[\mathtt{fl}_{\bigtriangledown}(bd)\,, \mathtt{fl}_{\bigtriangleup}(ac)]$

ただし $\mathrm{B} = [\min\{\mathtt{fl}_{\bigtriangledown}(ad)\,, \mathtt{fl}_{\bigtriangledown}(bc)\}, \max\{\mathtt{fl}_{\bigtriangleup}(ad)\,, \mathtt{fl}_{\bigtriangleup}(bc)\}]$.

2.5. ベクトル・行列の区間演算

上で述べた丸めの向きを制御することにより，ベクトル $x, y \in \mathbb{F}^n$ の内積 $x^T y$，行列 $A, B \in \mathbb{F}^{n\times n}$ の積，あるいは，ベクトル行列積 Ax の結果を区間で厳密に包含することができる．

$$\mathtt{fl}_{\bigtriangledown}\big(x^T y\big) \le x^T y \le \mathtt{fl}_{\bigtriangleup}\big(x^T y\big)$$

$$\mathtt{fl}_{\bigtriangledown}(Ax) \le Ax \le \mathtt{fl}_{\bigtriangleup}(Ax)$$

$$\mathtt{fl}_{\bigtriangledown}(AB) \le AB \le \mathtt{fl}_{\bigtriangleup}(AB)$$

このようにすると丸め方向の制御で区間演算が容易にできる．しかし，行列ベクトル積，行列積を高速に実装することは職人芸のレベルの難しさである（例えば，キャシュサイズをみて最適なブロック分割などを行う）．そのため通常は数値計算ライブラリを利用するのが主流である．

演習問題

3つの浮動小数点数 $a, b, c \in \mathbb{F}$ に対して，次が正しいかどうかを調べよ．正しくない場合は理由を述べ，できたらその反例をあげよ．

1. $\mathtt{fl}_{\bigtriangleup}(a + b) \ge a + b$
2. $\mathtt{fl}_{\bigtriangleup}(a - b) \ge a - b$
3. $\mathtt{fl}_{\bigtriangleup}(ab) \ge ab$
4. $\mathtt{fl}_{\bigtriangleup}(a/b) \ge a/b$
5. $\mathtt{fl}_{\bigtriangleup}((a + b) + c) \ge (a + b) + c$
6. $\mathtt{fl}_{\bigtriangleup}((ab)c) \ge (ab)c$
7. $\mathtt{fl}_{\bigtriangleup}((a - b) - c) \ge (a - b) - c$
8. $\mathtt{fl}_{\bigtriangleup}(a - (b + c)) \ge a - (b + c)$
9. $\mathtt{fl}_{\bigtriangleup}(-a + b) \ge (-a + b)$
10. $\mathtt{fl}_{\bigtriangleup}(-((-a) + (-b))) \ge -((-a) + (-b))$
11. $\mathtt{fl}_{\bigtriangleup}(ab + cd) \ge ab + cd$
12. $\mathtt{fl}_{\bigtriangleup}((a + b)(c + d)) \ge (a + b)(c + d)$
13. $\mathtt{fl}_{\bigtriangleup}(a/(b + c)) \ge a/(b + c)$
14. $\mathtt{fl}_{\bigtriangleup}(a - bc) \ge a - bc$
15. $\mathtt{fl}_{\bigtriangleup}(a + (-b)c) \ge a + (-b)c$

16. $\texttt{fl}_{\triangle}(sqrt(a)) \geq sqrt(a)$