

数値計算における誤差

- 実数データ表現
 - 浮動小数点表現
- 丸め誤差
 - 0.1の2進数表記・単精度表現・倍精度表現
- 桁落ち
- 情報落ち
- 打ち切り誤差
 - Taylor展開・級数展開の高次項の影響

実数データ表現

- 計算機内の数値データは、有限長のビット列で表現される
- 大きな数や小さな数を表現するためには、**符号部**と**指数部**、**仮数部**をもった実数表現(浮動小数点表現)が利用される

IEEE 754 実数表現に関する規格

$$(-1)^{s(2)} 1.d_1d_2 \cdots d_{m(2)} \times 2^{d'_1d'_2 \cdots d'_{c(2)} - b} \quad (s, d_i, d'_i \in \{0, 1\})$$

$s(2)$ 符号

$1.d_1d_2 \cdots d_{m(2)}$ 仮数

$d'_1d'_2 \cdots d'_{c(2)} - b$ 指数

この表記は、例えば、1.0101...010を2進数として解釈するという意味である

b バイアス(指数が負の値をとれるようにするもの) ¹⁶

単精度と倍精度

IEEE 754 実数表現に関する規格

$$(-1)^{s(2)} 1.d_1d_2 \cdots d_{m(2)} \times 2^{d'_1d'_2 \cdots d'_c(2)-b} \quad (s, d_i, d'_i \in \{0, 1\})$$

- 単精度 **32**ビットで実数を表現
- 倍精度 **64**ビットで実数を表現

	符号部	指数部	仮数部	m	c	b
単精度 (32 ビット)	第0ビット	第1~8ビット	第9~31ビット	23	8	127
倍精度 (64 ビット)	第0ビット	第1~11ビット	第12~63ビット	52	11	1023

指数部と仮数部がすべて0の実数は0(±0)を意味する
このほかにも±無限大などの表現が可能
整数の大小比較回路がそのまま使える

127は2進数で何桁？

(単精度表現)

1. 1
2. 11
3. 111
4. 1111
5. 11111
6. 111111
7. 1111111
8. 11111111
9. 111111111

次の数は 10 進数で何？

0 00000000 000000000000000000000000000000

(単精度表現)

1. 0.0
2. 0.1
3. 0.2
4. 0.3
5. 0.5
6. 1.0
7. 2.0
8. 3.0
9. 5.0

次の数は 10 進数で何？

0 01111111 000000000000000000000000

(単精度表現)

1. 0.0
2. 0.1
3. 0.2
4. 0.3
5. 0.5
6. 1.0
7. 2.0
8. 3.0
9. 5.0

次の数は 10 進数で何？

0 10000000 10000000000000000000000000000000

(単精度表現)

1. 0.0
2. 0.1
3. 0.2
4. 0.3
5. 0.5
6. 1.0
7. 2.0
8. 3.0
9. 5.0

次の数は 10 進数で何？

0 01111110 000000000000000000000000

(単精度表現)

1. 0.0
2. 0.1
3. 0.2
4. 0.3
5. 0.5
6. 1.0
7. 2.0
8. 3.0
9. 5.0

次の数は 10 進数で何？

0 10000001 01000000000000000000000000000000

(単精度表現)

1. 0.0
2. 0.1
3. 0.2
4. 0.3
5. 0.5
6. 1.0
7. 2.0
8. 3.0
9. 5.0

単精度表現

- 仮数部23ビット 指数部8ビット

仮数の表現範囲は有効桁数は24ビット

これは10進数で7桁程度の精度しかないことを意味する

$$1.00\dots0_{(2)} = 1 \text{ から } 1.11\dots1_{(2)} = 2 - 2^{-23} \approx 2 - 1.2 \times 10^{-7} \approx 2$$

指数の表現範囲は 1.2×10^{-38} から 1.7×10^{38}

$$2^{00000001_{(2)} - 127} = 2^{-126} \approx 1.2 \times 10^{-38}$$

$$2^{11111110_{(2)} - 127} = 2^{127} \approx 1.7 \times 10^{38}$$

$2^{00000000_{(2)} - 127}$ と、
 $2^{11111111_{(2)} - 127}$ は
特別な意味を持つので
除外してある

最大値の限界は $\approx 2 \times 2^{127} = 2^{128} \approx 3.4 \times 10^{38}$ となる

倍精度表現

- 仮数部52ビット 指数部11ビット

仮数の表現範囲は有効桁数は53ビット
これは10進数で16桁程度の精度

$$1.00\dots0_{(2)} = 1 \text{ から } 1.11\dots1_{(2)} = 2 - 2^{-52} \approx 2 - 2.2 \times 10^{-16} \approx 2$$

指数の表現範囲は 2.2×10^{-308} から 9.0×10^{307}

$$2^{000\dots01_{(2)} - 1023} = 2^{-1022} \approx 2.2 \times 10^{-308}$$

$$2^{111\dots10_{(2)} - 1023} = 2^{1023} \approx 9.0 \times 10^{307}$$

$2^{000\dots000_{(2)} - 127}$ と
 $2^{111\dots111_{(2)} - 127}$ は
特別な意味を持つので
除外してある

最大値の限界は $2 \times 2^{1023} = 2^{1024} \approx 1.7 \times 10^{308}$ となる

丸め誤差

- 有限桁2進数で表現する場合には、近似的にしか表せないことに起因する誤差

例)

$$\begin{aligned} 0.1_{(10)} &= 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + \dots \\ &= 0.000110011\dots_{(2)} \times 2^0 \\ &= 1.10011\dots_{(2)} \times 2^{-4} \end{aligned}$$

$0.1_{(10)}$ は有限桁では近似的にしか表現できない
(10進数の小数で $1/3$ や $1/7$ を表す場合に相当)

$$1/3 = 0.333333\dots$$

$$1/7 = 0.142857\dots$$

2進小数展開

$$\begin{aligned}\frac{1}{10} &= \frac{1}{2} \frac{1}{5} \\ &= \frac{1}{2} \frac{1}{8} \left(1 + \frac{3}{5}\right) \\ &= \frac{1}{2} \frac{1}{8} \left(1 + \frac{1}{2} \left(1 + \frac{1}{5}\right)\right) \\ &= \frac{1}{2} \frac{1}{8} \left(1 + \frac{1}{2} \left(1 + \frac{1}{8} \left(1 + \frac{1}{2} \left(1 + \frac{1}{5}\right)\right)\right)\right) \\ &= 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} \left(1 + \frac{1}{5}\right)\end{aligned}$$

$$\begin{array}{r}
 0.000110011 \\
 \hline
 1010 \overline{) 1.0000} \\
 1010 \\
 1100 \\
 1010 \\
 \\
 10000 \\
 1010 \\
 \\
 1010 \\
 \\
 10
 \end{array}$$

1/3 は 2 進数では？

1. 0.01
2. 0.011111111111...
3. 0.010101010101...
4. 0.010010010010...
5. 0.011011011011...

丸め誤差の例

irbで以下を試してみる
(0.1*3==0.3)

これは0.1の3倍が0.3に等しいかどうかを真偽で評価するための表現で
数学的には真(true)という結果になるはずであるが、実際は偽(false)となる

$$0.1_{(10)} = 0.000110011\dot{\dots}_{(2)} = 1.10011\dot{\dots}_{(2)} \times 2^{-4} = 1.10011\dot{\dots}_{(2)} \times 2^{01111111011_{(2)}-1023}$$

倍精度表現では 符号部 ["0", 指数部 "01111111011", 仮数部 "10011001100...110011010"] 0捨1入

$\times 3_{(10)} (\times 11_{(2)})$ ["0", "01111111101", "001100110011...00110100"] 0捨1入

$$0.3_{(10)} = 1.0011\dot{\dots}_{(2)} \times 2^{-2} = 1.0011\dot{\dots}_{(2)} \times 2^{01111111101_{(2)}-1023}$$

["0", "01111111101", "0011001100...1100110011"]

誤差

(検証) irbで以下を試してみよ
0.1*3-0.3
2**-54

丸め誤差は $\underbrace{0.0\dots01}_{(2)} \times 2^{-2} = 2^{-54}$
0が52個 31

$$0.1_{(10)} = 1.\overset{\cdot}{1}\overset{\cdot}{0}0\overset{\cdot}{1}1_{(2)} \times 2^{-4} \quad [\text{"0", "01111111011", "10011001100...110011010"}]$$

符号部
指数部
仮数部

$$3_{(10)} = 11_{(2)} = 1.1_{(2)} \times 2^1$$

$0.1_{(10)} \times 3_{(10)}$ はどのように計算される？

循環小数の
打ち切りで
0捨1入

$$\begin{array}{r}
 1.10011001100\dots110011010 \\
 1.1 \times 2^1 \\
 \hline
 110011001100\dots110011010 \\
 +) 110011001100\dots110011010 \\
 \hline
 10011001100110\dots011001110 \\
 1.0011001100110\dots0110100 \times 2^1
 \end{array}$$

最初の1は
仮数部には入れない
倍精度浮動小数点数
の定義を思い出そう

52桁(倍精度の仮数部の桁数)

桁あふれで
0捨1入

$$(-1)^{s_{(2)}} \overset{\circ}{1}.d_1 d_2 \dots d_{m_{(2)}} \times 2^{d'_1 d'_2 \dots d'_{c_{(2)}} - b} \quad (s, d_i, d'_i \in \{0, 1\})$$

0.1x3の計算結果の倍数度表現は以下のようなになる

$$[\text{"0", "01111111101", "001100110011...00110100"}]$$

桁落ちと情報落ち

- 桁落ち誤差

- ほぼ同じような数値の差をとると有効桁数が減少する

$$\begin{array}{r} 0.124 \\ - 0.123 \\ \hline 0.001 \end{array}$$

- 情報落ち誤差

- 大きさの異なる数値の加減算では、小さな数値は大きな数値の有効桁範囲外になり無視されてしまう

$$\begin{array}{r} 0.124 \\ + 0.0000000123 \\ \hline 0.124 \end{array}$$

桁落ち

- 桁落ち誤差

- ほぼ同じような数値の差をとると有効桁数が減少する

$$\begin{array}{r} 0.124 \\ - 0.123 \\ \hline 0.001 \end{array}$$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x) = \log x \quad f'(x) = \frac{1}{x}$$

```
def f(x)
  log(x)
```

```
end
```

```
def cancellations(x,h_digits)
```

```
  errors=Array.new(h_digits+1)
```

```
  for i in 0..h_digits
```

```
    h = 0.1**i
```

```
    df=(f(x+h)-f(x))/h
```

```
    errors[i] = abs(df-1.0/x)
```

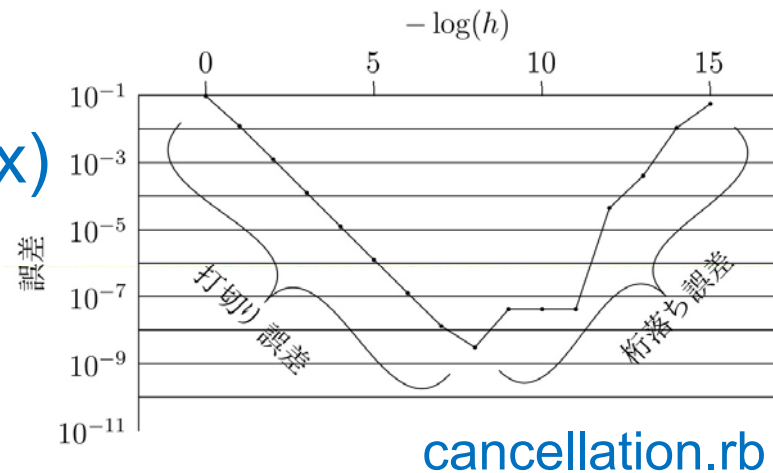
```
  end
```

```
  errors
```

```
end
```

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x) = \log x \quad f'(x) = \frac{1}{x}$$



情報落ち

- 情報落ち誤差

- 大きさの異なる数値の加減算では、小さな数値は大きな数値の有効桁範囲外になり無視されてしまう

$$\begin{array}{r} 0.124 \\ + 0.0000000123 \\ \hline 0.124 \end{array}$$

$$1 = \sum_{i=1}^n \frac{1}{n} = \frac{n-1}{n} + \frac{1}{n}$$

```
def coerce32(f)
  [f].pack("f").unpack("f")[0]
end

def sum(digits)
  n=10**digits
  sum = 0.0
  d = coerce32(1.0 / n)
  for i in 1..n
    sum = coerce32(sum + d)
  end
  sum
end
```

```
irb(main):004:0> sum(6)
=> 1.0090389251709
irb(main):005:0> sum(7)
=> 1.06476747989655
irb(main):006:0> sum(8)
=> 0.25
```

lossofinformation.rb

打ち切り誤差

- ある関数の値を無限級数を用いて数値計算する場合、有限の項数で打ち切って近似することにより生じる誤差（たとえば、実数が無限精度で表現できても生じる）

例) 指数関数の原点の周りのテイラー展開

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2!} + \dots$$

無限回の計算を行うことはできないので、有限回 (n 回) で打ち切って近似を行う

$$e^x = \sum_{i=0}^{\overset{n}{\circ}} \frac{x^i}{i!}$$

誤差の主要成分は $\frac{x^{n+1}}{(n+1)!}$

連立1次方程式の数値解法

- 小規模な連立1次方程式の解法
 - 消去法
 - Gauss消去法
 - Gauss-Jordan法
- (大規模な連立1次方程式の解法)
 - (反復法)
 - (Jacobi法) 講義では扱わない

消去法による解法

- 小規模な連立1次方程式は消去法によって解くことができる
- 消去法とは以下の演算を何回か行うことによって未知数を消去し、方程式をより簡単な方程式に変形して解を求める方法(筆算と同様)
 - 1つの方程式に(0でない)ある数を掛ける
 - 1つの方程式にある数を掛けて他の方程式に加える
- 多くの消去法では、未知数を1回に1個ずつ消去することによって係数行列を三角行列に変形し、後退置換によって未知数の値を逐次求める

Gauss消去法

- 与えられた連立1次方程式を行列演算で表現し、係数行列を定義する

$$\begin{array}{l} x + y - z = 2 \\ 3x + 5y - 7z = 0 \\ 2x - 3y + z = 5 \end{array} \quad \longrightarrow \quad \underbrace{\begin{pmatrix} 1 & 1 & -1 \\ 3 & 5 & -7 \\ 2 & -3 & 1 \end{pmatrix}}_{\text{係数行列}} \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{\text{定数ベクトル}} = \underbrace{\begin{pmatrix} 2 \\ 0 \\ 5 \end{pmatrix}}_{\text{定数ベクトル}}$$

連立1次方程式

消去法の説明を簡略化するため
右のような表記を導入する

筆算で式1に3を掛けて式2から
減算する操作を $r_2 - 3r_1$ で表す

1	1	-1	2	r_1
3	5	-7	0	r_2
2	-3	1	5	r_3

Gauss消去法(続き)

係数行列を三角行列に変形できるように各行に演算操作を行う

変形前

1	1	-1	2	r_1
3	5	-7	0	r_2
2	-3	1	5	r_3

例えば \bigcirc で示された係数を0にするため
行1に3を掛けて行2から減算する $r_2 - 3r_1$

同様に \bigcirc で示された係数を消去
するために $r_3 - 2r_1$ という操作を行う

変形後

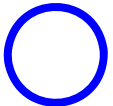
1	1	-1	2	r_1
0	2	-4	-6	$r_2 - 3r_1$
0	-5	3	1	$r_3 - 2r_1$

この操作を左のように表記する

一番右の列は、変形における
行の演算操作を表す

Gauss消去法(続き)

1	1	-1	2	r_1
0	2	-4	-6	r_2
0	-5	3	1	r_3

今回は  で示された係数を調整する

対角成分は1に、非対角成分は0
になるように行の演算操作を行う

1	1	-1	2	r_1
0	1	-2	-3	$r_2/2$
0	0	-7	-14	$r_3 + 5/2r_2$

r_1, r_2, r_3 はその都
度あらわす値が変わっ
ていることに注意

1	1	-1	2	r_1
0	1	-2	-3	r_2
0	0	1	2	$-r_3/7$

最終的に係数行列が上三角行列
に変形されたら、後退置換により、
各変数の値が求まる

$$x = 3 \quad y = 1 \quad z = 2$$

Gauss-Jordan法

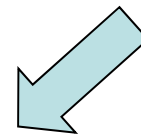
- Gauss消去法において、係数行列を単位行列に変形する方法
- 後退置換が不必要である

例) 前出の連立方程式の解法

1	1	-1	2	r_1
3	5	-7	0	r_2
2	-3	1	5	r_3



1	1	-1	2	r_1
0	2	-4	-6	$r_2 - 3r_1$
0	-5	3	1	$r_3 - 2r_1$



1	0	1	5	$r_1 - r_2/2$
0	1	-2	-3	$r_2/2$
0	0	-7	-14	$r_3 + 5/2r_2$



1	0	0	3	$r_1 + r_3/7$
0	1	0	1	$r_2 - 2/7r_3$
0	0	1	2	$-r_3/7$

$$x = 3 \quad y = 1 \quad z = 2$$

```
def gj(a) # Gauss - Jordan method without pivoting
```

```
  row = a.length()
```

```
  col = a [0].length()
```

```
  for k in 0..(col-2)
```

1行ずつ消去する
(k行目の処理をする)

```
    akk = a[k][k]
```

```
    for i in 0..(col-1) # normalize row k
```

k行目を対角成分 $a[k][k]$ で
割り算し正規化

```
      a[k][i]=a[k][i ]*1.0/akk
```

```
    end
```

```
  for i in 0..(row-1) # eliminate column k
```

```
    if i != k # of all rows but k
```

$a[i][k]$ を消去する処理
(ただしiはk以外)

```
      aik = a[i][k]
```

```
      for j in k..(col-1)
```

```
        a[i][j] = a[i][j] - aik * a[k][j]
```

k行に $aik=a[i][k]$ をかけて
i行から引くことで
 $a[i][k]$ を消去(0にする)

```
      end
```

```
    end
```

```
  end
```

```
end
```

```
  a
```

```
end
```

gj.rb

Pivoting

- 消去法では、係数行列の要素での割算における問題がある
 - 割算の分母が0になる場合 (係数行列の対角要素が0)
 - ⇒ 行または変数を入れ替える必要がある
 - 割算の分母が0に近い値になる場合
 - ⇒ 割算の結果非常に大きな数字が結果として表れると数値計算の精度が失われる (前回の講義参照)
 - ⇒ この場合も行または変数を入れ替える必要がある

この問題を解決するため以下のPivotingを行う

$$\begin{array}{r} \dots \\ k \text{ 行} \dots \\ \dots \\ i \text{ 行} \dots \\ \dots \end{array} \begin{array}{ccc} 1 & \vdots & \vdots \\ \dots & 0 & a_{k,k+1} \\ \dots & 0 & a_{k+1,k+1} \\ \vdots & \vdots & \vdots \\ \dots & 0 & a_{i,k} \\ \vdots & \vdots & \vdots \end{array}$$

k 番目の行の処理で a_{kk} が0に近い場合、 a_{ik}/a_{kk} の絶対値が大きくなり、 k 行に a_{ik}/a_{kk} をかけて i 行から引いた時に情報落ち誤差が生じるので良くない
そこで、 a_{ik} の中で絶対値が最大の a_{pk} を選び、 k 行と p 行を入れ替える
元の連立方程式では式の順序を入れ替えることに相当する

```
def gjp(a) # Gauss - Jordan method without WITH pivoting
```

```
  row = a.length()
```

```
  col = a [0].length()
```

```
  for k in 0..(col-2)
```

```
    akk = a[k][k]
```

```
    for i in 0..(col-1) # normalize row k
```

```
      a[k][i]=a[k][i ]*1.0/akk
```

```
    end
```

```
    for i in 0..(row-1) # eliminate column k
```

```
      if i != k # of all rows but k
```

```
        aik = a[i][k]
```

```
        for j in k..(col-1)
```

```
          a[i][j] = a[i][j] - aik * a[k][j]
```

```
        end
```

```
      end
```

```
    end
```

```
  end
```

```
  a
```

```
end
```

```
max = maxrow(a,k)
```

```
# find absolute maximal coeff .
```

```
swap(a,k,max)
```

```
# swap rows
```

k列目の係数の絶対値が
最大となる行を
k行目以降から探す

maxrow(a,k)

maxrow([[1,2,3,0],[-3,0,1,2],[2,1,0,3]],0) → 1

maxrow([[1,2,3,0],[-3,0,1,2],[2,1,0,3]],1) → 2

maxrow([[1,2,3,0],[-3,0,1,2],[2,1,0,3]],2) → 2

```
def maxrow(a,k)
  m = k
  for i in ??..??
    if abs(a[i][k]) > abs(a[m][k])
      ???
    end
  end
  m
end
```

練習

- 以下の連立1次方程式をGauss-Jordan法で解いてみよ。Pivotingするかしないかで、結果が変わるか観察せよ。

$$x - 50y - 3z = -90$$

$$-85x + 2y - 25z = -6$$

$$79x + 5y + 30z = -1$$

- 共通資料から gj.rb と gjp.rb をダウンロード。

進捗状況の確認

1. gj も gjp も動いた時点で投票してください。
2. maxrow (および abs) と swap を定義したが gjp が動かない。
3. maxrow (abs を使う) または swap が定義できない。
4. gj は動いた。
5. gj が動かない。
6. gj への入力が作れない。