情報科学

数値計算

数値計算とは？

Mathematica を
実行してみよう

- 代数演算
  - 等式を一定の規則のもとに変形して解析的に求める
- 数値計算
  - 反復計算によって”近似的に”解を求める
  - 解析的に求めることが困難な方程式を解く場合に用いる

例） 物理学、化学、天文学などにおける数値積分や微分方程式の解法など

数値計算トピック

- 数値積分
  - モンテカルロ法を含む。
- 数値計算における誤差
- 連立方程式

乱数とモンテカルロ法

1. モンテカルロ法
2. 乱数とは？
   - 一様乱数
   - 正規乱数
3. 疑似乱数

モンテカルロ法

- 数学的問題の解法に乱数（疑似乱数）を用いる方法の総称をモンテカルロ法という

  モンテカルロとはモナコにある賭博で有名な都市（賭博場が国営）であり、多くの賭博は乱数を利用して行われるためこのように呼ばれる

- 一般に以下の２種類の問題を扱う
  - 決定論的問題
    - 乱数を用いる多次元積分

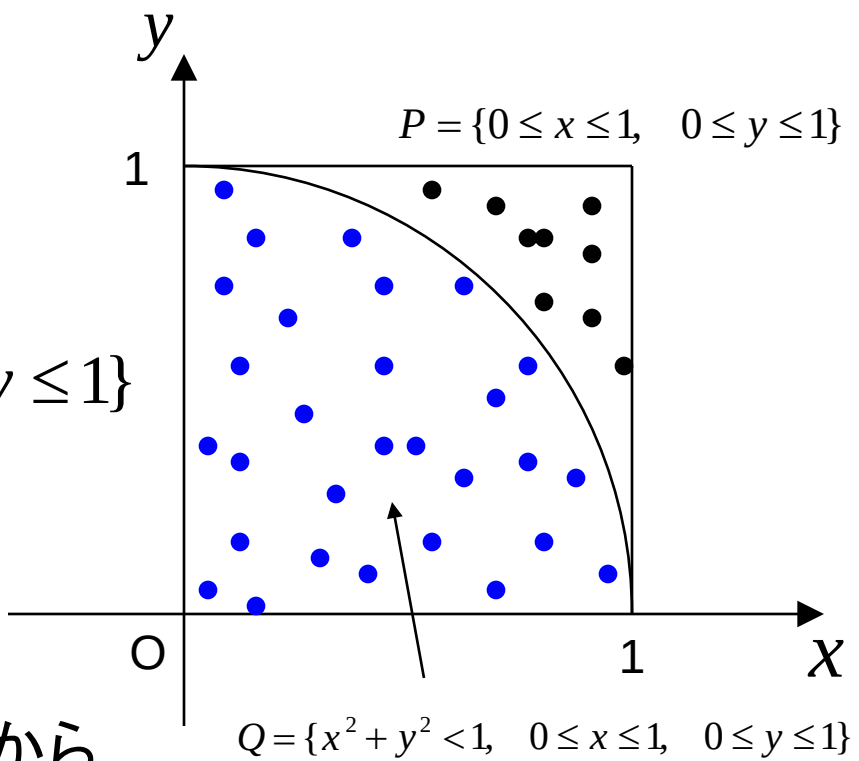
      例えば、最も簡単な例では円周率の近似計算などがある。
  - 非決定論的（確率的変動を含む）問題
    - 自然科学・社会科学におけるシミュレーション

      （実際に実験が困難であったり多大な費用がかかる場合）

      例えば、ランダムウォーク（ブラウン運動）

円周率の近似計算

- 平面上の正方領域
  $P=\{0\le x\le 1,\quad 0\le y\le 1\}$
  にランダムに n 個の点を配置し、
  その中で、四半円領域
  $Q=\{x^2+y^2<1,\quad 0\le x\le 1,\quad 0\le y\le 1\}$
  にある点の数 m を求める。
- 一様乱数を用いる場合、
  配置される点の数は
  領域の面積に比例するはずであるから、
  $m/n\approx Q/P=\pi/4$
- つまり円周率は $\pi\approx 4m/n$ で
  近似できる。



$x_i\in[0,1)\quad y_i\in[0,1)\quad i=1...n$ のうち

- $x_i^2+y_i^2\ge 1$ を満たす点
- $x_i^2+y_i^2<1$ を満たす点

ただし、精度を上げるためには、
非常に多くの点を用いる必要がある

```
def montecarlo(n)
    m = 0
    for i in 1..n
        x = rand() # random number in [0,1)
        y = rand()
        if x*x + y*y < 1.0
            m = m + 1
        end
    end
    m*1.0/n
end
```

montecarlo.rb

```
def mcplot(n)
    a = make2d(500,500)
    for i in 1..n
        x = rand() # random number in [0,1)
        y = rand()
        if x*x + y*y < 1.0
            a[y*500][x*500] = 1.0
        else
            a[y*500][x*500] = 0.5
        end
    end
    a
end
```

練習

- まだ make1d と make2d ができていない人は、ダウンロードしてください。
- show(mcplot(n)) をいろいろな n で実行して、点がばらまかれる様子を観察せよ。
  - mcplot はカット&ペーストしてよい。
  - 配列の添字に浮動小数点数を与えると切り捨てられる。
- 4*montecarlo(n) を色々な n で実行し、円周率が近似できることを確かめよ。
  - montecarlo.rb は共通資料にある。

進捗状況の確認

1. 4*montecarlo(n) で円周率が近似できることを確認した。
2. show(mcplot(n)) がうまく表示された。
3. 困った。

乱数とは？

- 乱数とは、ある一つの数が現れたとき、それに続く数が前の数と全く関係なく現れるような列（乱数列）の要素

もう少し丁寧に定義すると…

- 乱数列とは、ある分布に従う（互いに独立な事象を表す）確率変数の実現値の列をいう
- 乱数とは、乱数列の各要素である

ある確率変数 x の実現値の列である乱数列

$\{x_1, x_2, \ldots, x_n, x_{n+1}, \ldots\}$ において

x が x_{n+1} の値をとることは $x_1, x_2, \ldots, x_n$ からは予測できない

確率分布によって、一様乱数、正規乱数などがある。

一様乱数

- 出現確率が値によらず一定である乱数を一様乱数という（確率分布が一様である場合）

例えば、

サイコロの振って出た数字を並べたものは、続けて出現する二つの数の間には因果関係がないため乱数列である

また、（サイコロに細工がしてなければ）1から6までの数字が1/6の（一定の）確率で出現するため一様乱数である

年末ジャンボ宝くじの当選番号（数字部分）を毎年記録したものは一様乱数だろうか？

矢を射ることによって全く公平に行われるとするとこれも一様乱数となる。

自然現象における乱数列

放射性原子は放射線を出して崩壊する
（原子核が崩壊して別の核種になるか、励起状態の原子核
がより低いエネルギー状態へ遷移する）
放射性物質の単位時間あたりの崩壊数を計測する

- 個々の原子核は他と無関係に崩壊するとすると乱数列
- 同一の核種は平均として同じ割合で崩壊するが、
  崩壊数は同じ確率で出現するわけではない
  （平均からのずれがある）

このように、一般には、一様ではない乱数列を扱う必要もある

正規乱数

- 出現確率が正規分布に従うような乱数を正規乱数という
- 確率変数 x が ξ よりも小さい値をとる確率 $P(x)$ が以下の正規分布に従う場合

$$P(x<\xi)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\xi}\exp\left(-\frac{x^2}{2}\right)dx$$

自然現象を扱うシミュレーション（計算機で模擬実験を行うこと）では、一様乱数と共に正規乱数をよく用いる

疑似乱数
(pseudorandom numbers)

- 乱数列のように見えるが、計算機である計算を行うことによって求めることができる数列をいう
- 区間[0,1)の一様乱数を近似する数列をさすことが多い

疑似乱数は、*n*個目までの*m*個の既知要素 $x_n, x_{n-1}, \ldots, x_{n-m+1}$ を用いて以下の漸化式により*n+1*番目の要素を計算する

$$x_{n+1} = f(x_n, x_{n-1}, \ldots, x_{n-m+1})$$

疑似乱数列 $\{x_n\}$ は周期が大きいことが望ましい

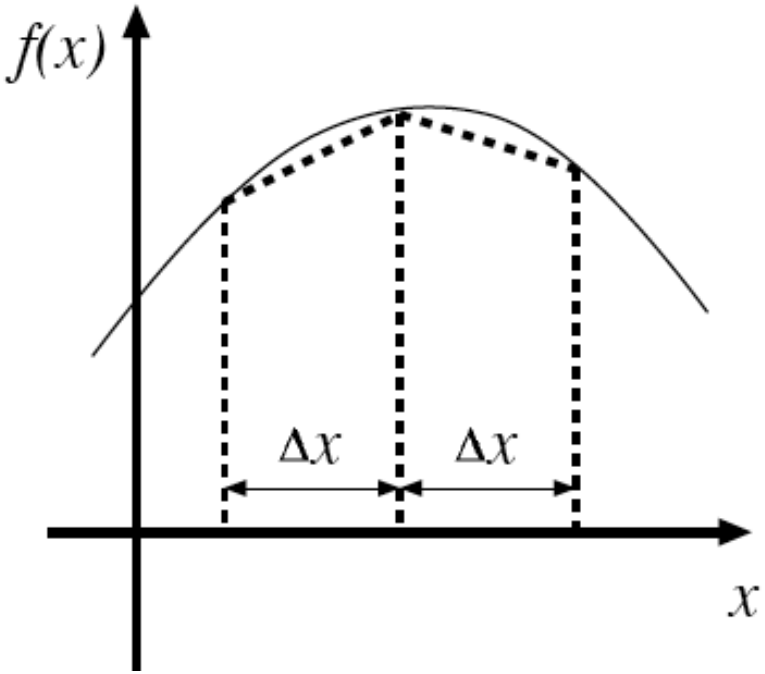
（実際には、一様分布に関する適合度、統計的独立性などを様々な乱数検定法で検定し疑似乱数としての資格を与える）

数値積分

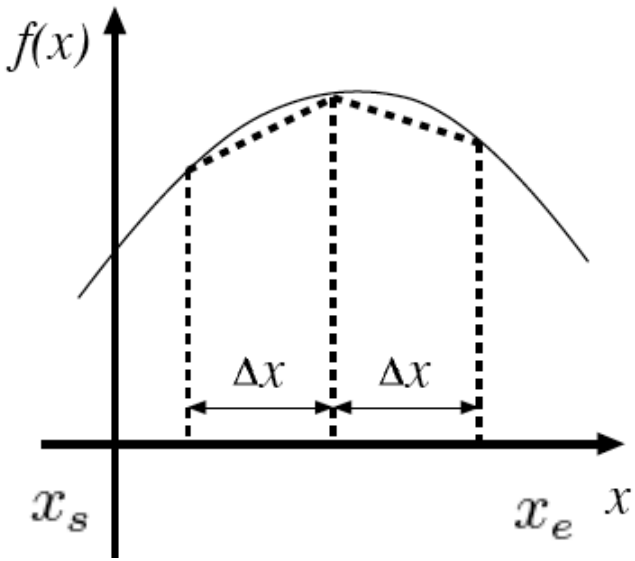
1. 台形公式
2. シンプソン公式
3. モンテカルロ法

台形公式

- 積分を区分線形(piecewise linear)で近似する方法
- 全積分区間をn等分し各部分区間 Δx
  の積分を台形の面積として計算し総和をとる



台形公式（続き）



ある関数 $f(x)$ の x_s から x_e までの積分の近似値を求めるには、

各部分区間 $\Delta x = \dfrac{x_e - x_s}{n}$

における台形の総和で近似する

$$\int_{x_s}^{x_e} f(x)dx \approx \sum_{i=0}^{n-1} \frac{1}{2}\{f(x_s + i\Delta x) + f(x_s + (i+1)\Delta x)\}\Delta x$$

$$= \Delta x \left\{ \frac{f(x_s) + f(x_e)}{2} + \sum_{i=1}^{n-1} f(x_s + i\Delta x) \right\}$$

```
def f(x)
    x/((x+1.0)*(x+2.0))
end

def trapezoid(xs,xe,n)
    deltax = (xe-xs)*1.0/n
    sum = (f(xs)+f(xe))/2.0
    for i in 1..(n-1)
        sum = sum + f(xs+i*deltax)
    end
    deltax * sum
end
```

trapezoid.rb

練習

- 以下の積分を台形公式で近似して円周率を求めよ。n を引数として円周率を返す関数を定義せよ。

$$\pi = 4\int_0^1 \frac{1}{1+x^2}dx$$

trapezoid.rb の f を定義しなおし、

```
def pi(n)
  4.0 * trapezoid(....)
end
```

注意！

- f(x) を 1/(1+x*x) と定義して、しかも上限と下限として 0 と 1 を指定すると、下限は f(0)=1 となって正しいが、上限は f(1)=0 となってしまう。
- 大きな誤差を生み、誤差の評価は無意味になってしまう。
- 1.0/(1.0+x*x) などとすればよい。

進捗状況の確認

1. π の計算ができた時点で投票してください。
2. 値がおかしい。
3. プログラムが動かない。

正しく動いたら、n の値を変えて、真の値に近づく様子を観察しよう。

さらに時間があれば次のシンプソン法。

収束に関するコメント

$$f(x+t) = f(x) + f'(x)t + \frac{f''(x)}{2}t^2 + \ldots$$

$$f'(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} - \frac{f''(x)}{2}\Delta x - \ldots$$

$$\int_x^{x+\Delta x} f(x)dx = \int_0^{\Delta x} f(x+t)dt = f(x)\Delta x + f'(x)\frac{\Delta x^2}{2} + \frac{f''(x)}{2}\frac{\Delta x^3}{3} + \ldots$$

$$= f(x)\Delta x + \left(\frac{f(x+\Delta x) - f(x)}{\Delta x} - \frac{f''(x)}{2}\Delta x - \ldots \right)\frac{\Delta x^2}{2} + \frac{f''(x)}{2}\frac{\Delta x^3}{3} + \ldots$$

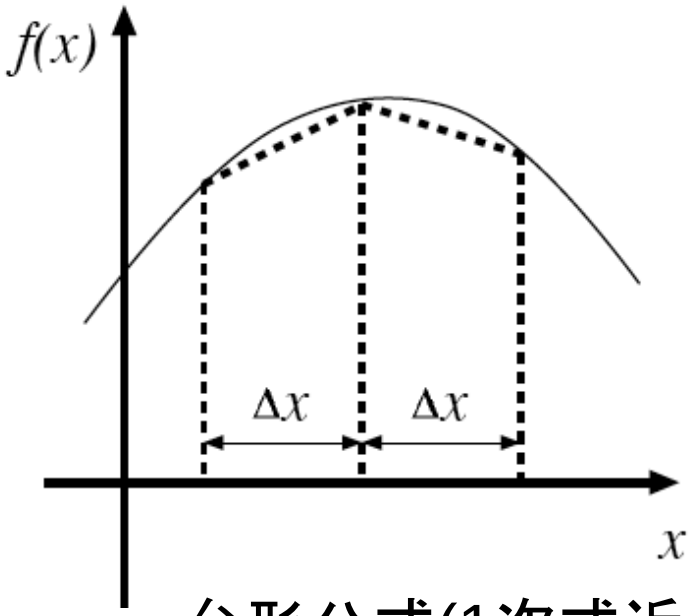
$$= \frac{f(x) + f(x+\Delta x)}{2}\Delta x + O(\Delta x^3)$$

Δx^3 に比例した項

- 刻み一つごとに Δx^3 に比例した誤差
- これが積もると Δx^2 に比例した誤差になる

シンプソン公式

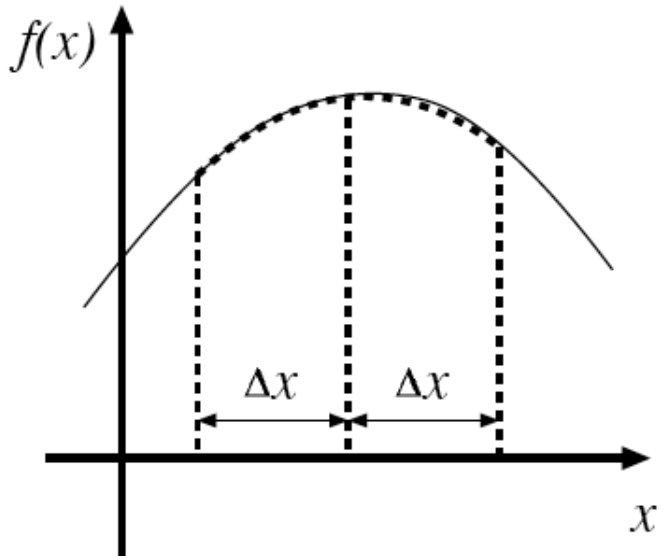
- 積分を1次式ではなく2次式で近似する方法



台形公式(1次式近似)

$$\frac{1}{2}\{f(x)+f(x+\Delta x)\}\Delta x$$

$$\Delta x = \frac{x_e - x_s}{n}$$

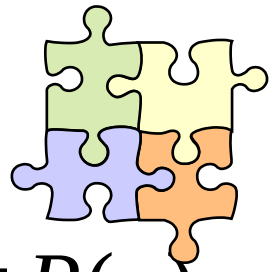


シンプソン公式(2次式近似)

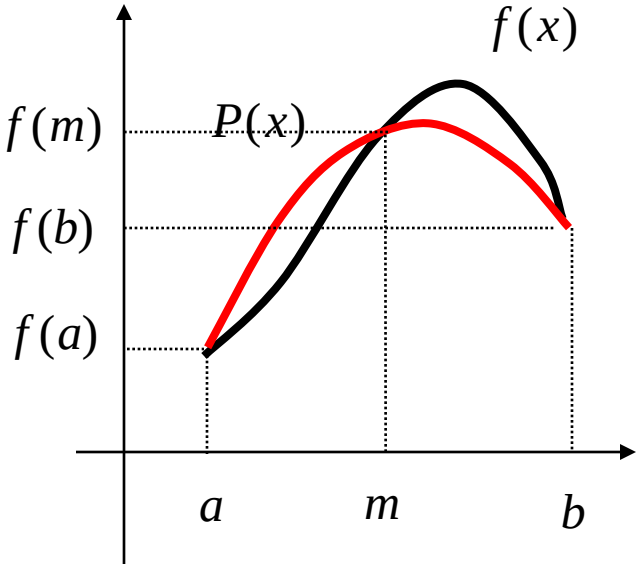
$$\frac{1}{3}\{f(x)+4f(x+\Delta x)+f(x+2\Delta x)\}\Delta x$$

$$\Delta x = \frac{x_e - x_s}{2n}$$

ラグランジュ補間



- ある関数 $f(x)$を3点a,m,bで補間する2次関数 $P(x)$

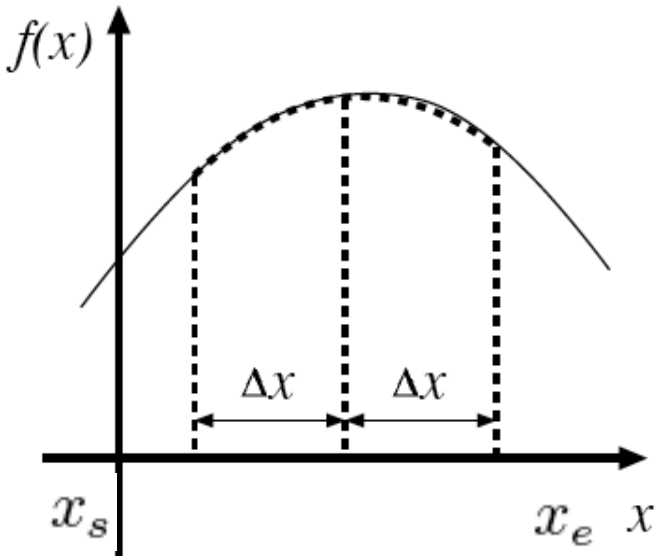


$$P(x)=\frac{(x-b)(x-m)}{(a-b)(a-m)}f(a)+\frac{(x-a)(x-b)}{(m-a)(m-b)}f(m)+\frac{(x-a)(x-m)}{(b-a)(b-m)}f(b)$$

$$m=\frac{(b+a)}{2} \Rightarrow \int_a^b P(x)dx=\frac{(b-a)}{6}[f(a)+4f(m)+f(b)]$$

$$a\to x,\quad b\to x+2\Delta x,\quad m\to x+\Delta x \Rightarrow =\frac{1}{3}\{f(x)+4f(x+\Delta x)+f(x+2\Delta x)\}\Delta x$$

シンプソン公式（続き）



部分区間 $\Delta x = \dfrac{x_e - x_s}{2n}$ とし

区間 $2\Delta x$ の部分の積分近似値は

$$\frac{1}{3}\left\{f(x) + 4f(x+\Delta x) + f(x+2\Delta x)\right\}\Delta x$$

となるため、x_s から x_e までの

積分近似値の総和は

$$\begin{aligned}\int_{x_s}^{x_e} f(x)dx &\approx \sum_{i=0}^{n-1}\frac{1}{3}\left\{f(x_s+2i\Delta x) + 4f(x_s+(2i+1)\Delta x) + f(x_s+(2i+2)\Delta x)\right\}\Delta x \\ &= \frac{\Delta x}{3}\left\{f(x_s) + 4f(x_s+\Delta x) + f(x_e) + \sum_{i=1}^{n-1}\left(2f(x_s+2i\Delta x) + 4f(x_s+(2i+1)\Delta x)\right)\right\}\end{aligned}$$

練習6.2

- 前のスライドの式に従って積分を行う関数 simpson(xs,xe,n) を定義せよ（注意：台形公式の場合と違って Δx が積分範囲の 1/2n になっている）。

練習

- 以下の積分をシンプソン公式で近似して円周率を求めよ。n を引数として円周率を返す関数を定義せよ。

$$\pi = 4\int_0^1 \frac{1}{1+x^2}dx$$

練習に関する補足

- シンプソン公式は、区間ごとに

$$\Delta x^5 f^{(4)}(\xi)$$

の誤差を持つ。ξ は区間内の点。

- この誤差が累積すると、$f^{(4)}$ を積分区間で積分するのと同様の状況になり、積分区間を $[a,b]$ とすると、誤差は

$$\Delta x^4 (f^{(3)}(b) - f^{(3)}(a))$$

に比例。

- $f^{(3)}(b) = f^{(3)}(a)$ の場合、この項は 0 に。
- 誤差の次の項は Δx^6 に比例。

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

$$(-1)^{s_{(2)}} 1.d_1 d_2 \cdots d_{m(2)} \times 2^{d'_1 d'_2 \cdots d'_{c(2)} - b} \quad (s, d_i, d'_i \in \{0, 1\})$$

$s_{(2)}$　符号

$1.d_1 d_2 \cdots d_{m(2)}$　仮数

$d'_1 d'_2 \cdots d'_{c(2)} - b$　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）

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

次の数は 10 進数で何？

0 01111111 10000000000000000000000
(単精度表現)

1. 0.0
2. 0.1
3. 0.5
4. 1.0
5. 1.1
6. 1.5

次の数は 10 進数で何？

0 10000000 10000000000000000000000
(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 00000000000000000000000
(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 00000000 00000000000000000000000

(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

$1.00\ldots0_{(2)} = 1$ から $1.11\ldots1_{(2)} = 2 - 2^{-23} \approx 2 - 1.2 \times 10^{-7} \approx 2$

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

$1.00\ldots0_{(2)} = 1$ から $1.11\ldots1_{(2)} = 2 - 2^{-52} \approx 2 - 2.2 \times 10^{-16} \approx 2$

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

$2^{000\ldots01\ (2)\ -1023} = 2^{-1022} \approx 2.2 \times 10^{-308}$

$2^{111\ldots10\ (2)\ -1023} = 2^{1023} \approx 9.0 \times 10^{307}$

$2^{000\ldots000\ (2)\ -127}$ と $2^{111\ldots111\ (2)\ -127}$ 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

- 10進数を2進数で表現する場合、有限桁では近似的にしか表せないことに起因する誤差

例）

$$0.1_{(10)} = 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + \ldots$$
$$= 0.000110011\ldots_{(2)} \times 2^{0}$$
$$= 1.10011\ldots_{(2)} \times 2^{-4}$$

$0.1_{(10)}$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…
1/7 = 0.142857…

2進小数展開

$$\frac{1}{10} = \frac{1}{16}\left(1+\frac{3}{5}\right)$$

$$= \frac{1}{16}\left(1+\frac{1}{2}\left(1+\frac{1}{5}\right)\right)$$

$$= \frac{1}{16}\left(1+\frac{1}{2}\left(1+\frac{1}{8}\left(1+\frac{3}{5}\right)\right)\right)$$

$$= \frac{1}{16}\left(1+\frac{1}{2}\left(1+\frac{1}{8}\left(1+\frac{1}{2}\left(1+\frac{1}{5}\right)\right)\right)\right)$$

$$= 2^{-4}+2^{-5}+2^{-8}+2^{-9}\left(1+\frac{1}{5}\right)$$

```
      0.000110011
   ------------------
1010) 1.0000
        1010
         1100
         1010
           10000
            1010
             1100
             1010
               10
```

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

$$0.1_{(10)} = 0.000\dot{1}\dot{1}0011\ldots_{(2)} = 1.\dot{1}0011_{(2)} \times 2^{-4} = 1.\dot{1}0011_{(2)} \times 2^{01111111011_{(2)} - 1023}$$

倍精度表現では ["0", "01111111011", "10011001100…110011010"] 0捨1入

符号部　指数部　仮数部

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

$$0.3_{(10)} = 1.\dot{0}0\dot{1}1\ldots_{(2)} \times 2^{-2} = 1.\dot{0}0\dot{1}1_{(2)} \times 2^{01111111101_{(2)} - 1023}$$

誤差

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

(検証)irbで以下を試してみよ
0.1*3-0.3
2**-54

丸め誤差は $\underbrace{0.0\ldots01}_{0\text{が}52\text{個}}{}_{(2)} \times 2^{-2} = 2^{-54}$

$0.1_{(10)} = 1.\dot{1}0\dot{0}11_{(2)} \times 2^{-4}$ ["0", "01111111011", "10011001100…11001101**0**"]

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

$$
\begin{array}{rl}
 & 1.10011001100\ldots11001101\mathbf{0} \\
\times) & \qquad\qquad\qquad\qquad\qquad\qquad 11 \\
\hline
 & 1.10011001100\ldots11001101\mathbf{0} \\
+) & \quad 1.10011001100\ldots11001101\mathbf{0} \\
\hline
 & 10.01100110011\ldots01100\mathbf{1}1\mathbf{10} \\
 & 1.0011001100110\ldots0110\mathbf{100} \times 2^{1}
\end{array}
$$

桁あふれで0捨1入

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だそう

52桁（倍精度の仮数部の桁数）

$$(-1)^{s_{(2)}} 1.d_1 d_2 \cdots d_{m(2)} \times 2^{d'_1 d'_2 \cdots d'_{c(2)} - b} \quad (s, d_i, d'_i \in \{0,1\})$$

0.1x3の計算結果の倍数度表現は以下のようになる

["0", "01111111101", "00110011001**1**…00110100"]

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

```
   0.124
 - 0.123
 -------
   0.001
```

- 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

```
   0.124
 + 0.0000000123
 --------------
   0.124
```

打切り誤差

- ある関数の値を無限級数を用いて数値計算する場合、有限の項数で打切って近似することにより生じる誤差（たとえ，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

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

無限回の計算を行うことはできないので、有限回（n回）で打切って近似を行う

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

誤差の主要成分は $\dfrac{x^{n+1}}{(n+1)!}$

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

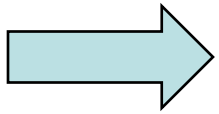
消去法による解法

- 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

$$\begin{aligned} x + y - z &= 2 \\ 3x + 5y - 7z &= 0 \\ 2x - 3y + z &= 5 \end{aligned}$$



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

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 ○ 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 ○ 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

今度は ○ 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	$-r_3/7$

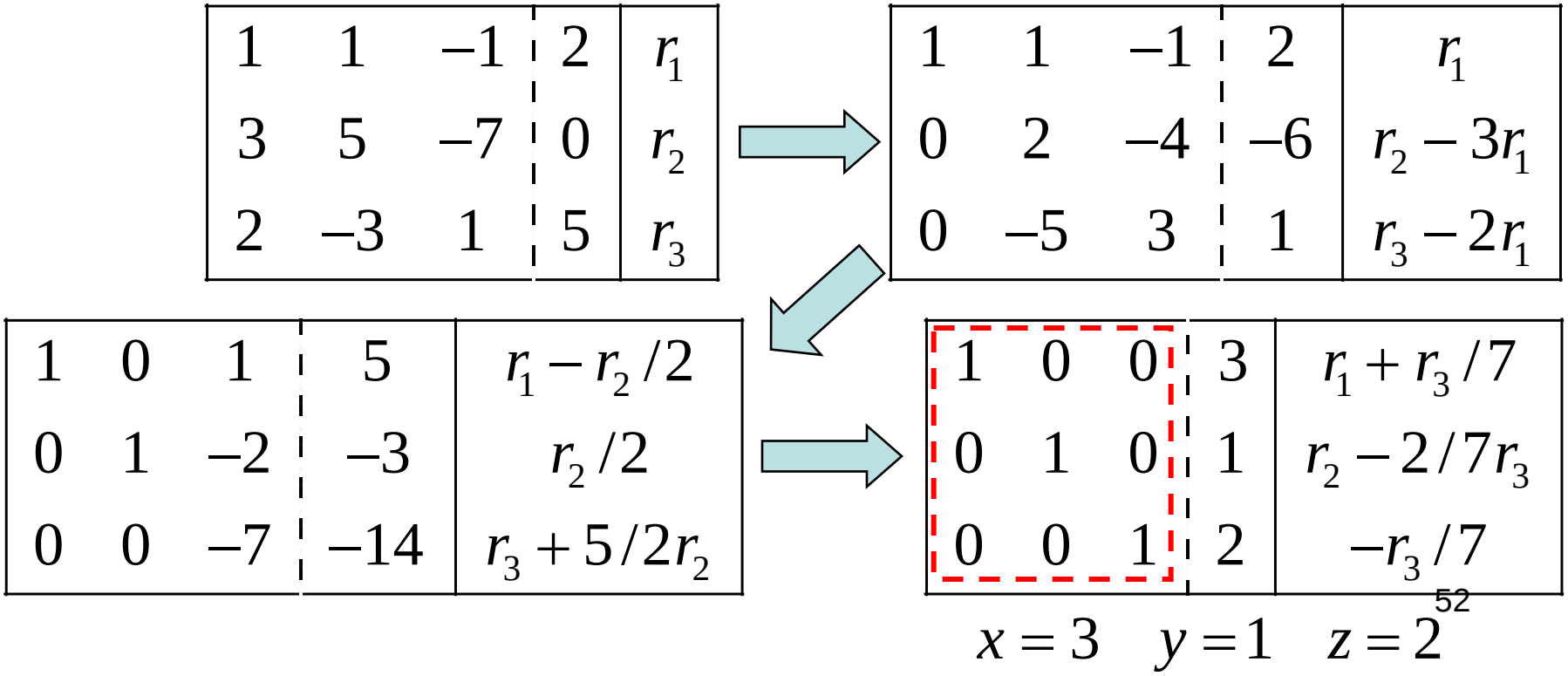
最終的に係数行列が上三角行列に変形されたら、後退置換により、各変数の値が求まる

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

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



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

1行ずつ消去する
(k行目の処理をする)

k行目を対角成分a[k][k]で
割り算し正規化

a[i][k]を消去する処理
（ただしiはk以外）

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）

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

$$
\begin{array}{cc}
 & \begin{array}{ccc} & \vdots & \vdots \end{array} \\
 & \begin{array}{ccc} 1 & \vdots & \vdots \end{array} \\
k\text{ 行} & \begin{array}{ccc} \cdots\ 0 & a_{kk} & a_{k,k+1} \end{array} \\
 & \begin{array}{ccc} \cdots\ 0 & a_{k+1,k} & a_{k+1,k+1} \end{array} \\
 & \begin{array}{ccc} \vdots & \vdots & \vdots \end{array} \\
i\text{ 行} & \begin{array}{ccc} \cdots\ 0 & a_{i,k} & a_{i,k+1} \end{array} \\
 & \begin{array}{ccc} \vdots & \vdots & \vdots \end{array}
\end{array}
$$

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)
        max = maxrow(a,k)
            # find absolute maximal coeff .
        swap(a,k,max)
            # swap rows
        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
```

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

gjp.rb

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
```

gjp.rb の中で定義。

abs(x)

```
def abs(x)
    if x<0
        ??
    else
        x
    end
end
```

abs.rb に格納。

swap(a,k,m)

```
def swap(a,k,m)
    w = a[?]
    a[m] = a[?]
    a[?] = w
end
```

swap.rb に格納。

練習6.9

- 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. swap が定義できない。
4. maxrow が定義できない。
5. abs が定義できない。
6. gj は動いた。
7. gj が動かない。
8. gj への入力が作れない。

練習問題確認プログラム

- ex06.rb（練習6.2と6.9-6.16）の結果を以下のアドレスに送れ。

  is-komaba@lyon.is.s.u-tokyo.ac.jp

- Subject:欄は、

  is-komaba

  としてください。

- ✍切: 11月XX日
  - 練習問題確認プログラムの結果は出席点の一部とします。結果の内容は問いません。（習得状況を知りたい。）

配列の引数・返り値

```
def  inc1(b)
   n = b.length()

   for i in 0..n-1
      b[i] = b[i]+1
   end
   b
end
```

```
def plus1(b)
   n = b.length()
   c = Array.new(n)
   for i in 0..n-1
      c[i] = b[i]+1
   end
   c
end
```

配列の引数・返り値

```
>> a = [1,2]
=> [1, 2]
>>  inc1(a)
=> [2, 3]
>> a
=> [2, 3]
```

```
>> plus1(a)
=> [3, 4]
>> a
=> [2, 3]
```

次の最後の結果は？

```
>> d = [2,3]
>> e = d
>> inc1(e)
>> d = plus1(d)
>> d[0]*e[1]
```

1. 9
2. 12
3. 15
4. 16
5. 20
6. 25

練習4.13b)

- 配列 a の i 番目と i+1 番目の数値の大きさを比べ、前者が後者より大きいときに両者を入れ替える関数 swap_ascending(a,i) を作れ。

swap_ascending(a, i)

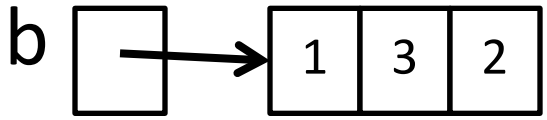
swap_ascending(a, i)

```
def swap_ascending(a, i)
  if a[i] > a[i+1]
    w = a[i]
    a[i] = a[i+1]
    a[i+1] = w
  end
end
```

swap_ascending(a, i)

```
def swap_ascending(a, i)
  if a[i] > a[i+1]
    w = a[i]
    a[i] = a[i+1]
    a[i+1] = w
  end
end
```

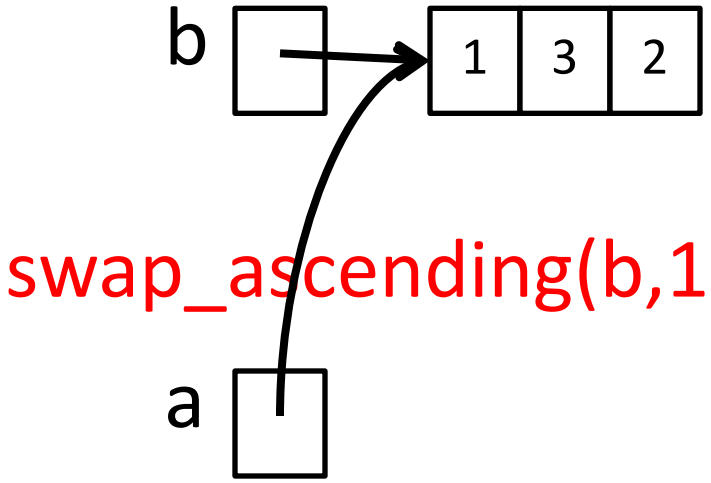
b = [1,3,2]



swap_ascending(a, i)

```
def swap_ascending(a, i)
  if a[i] > a[i+1]
    w = a[i]
    a[i] = a[i+1]
    a[i+1] = w
  end
end
```

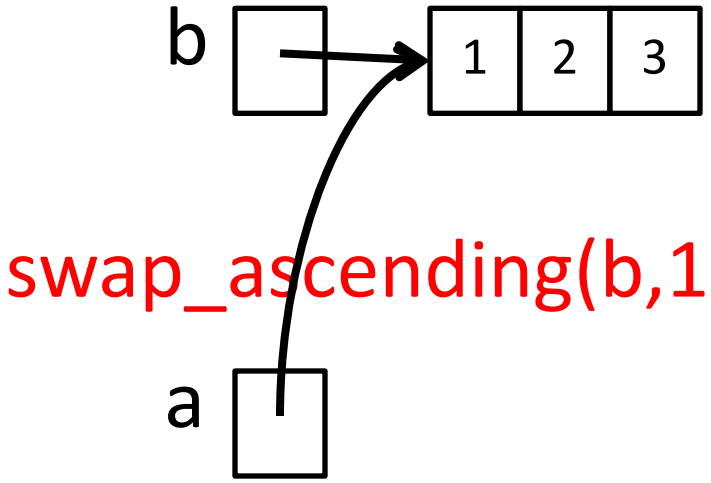
b = [1,3,2]



swap_ascending(a, i)

```
def swap_ascending(a, i)
  if a[i] > a[i+1]
    w = a[i]
    a[i] = a[i+1]
    a[i+1] = w
  end
end
```

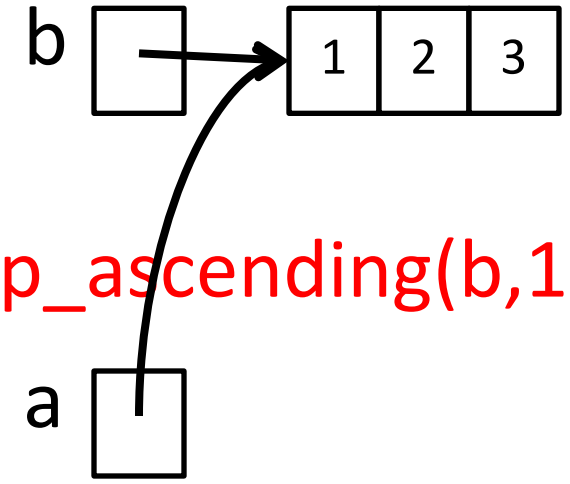
b = [1,3,2]



swap_ascending(a, i)

```
def swap_ascending(a, i)
  if a[i] > a[i+1]
    w = a[i]
    a[i] = a[i+1]
    a[i+1] = w
  end
end
```

b = [1,3,2]



swap_ascending(b,1)

b

=> [1, 2, 3]

レポート課題

- 課題
  - 数値計算を行うプログラムを書き、
    その実行結果と考察を提出してください。
    - 連立方程式（教科書のままでなく何か工夫を）
    - 積分（教科書のままでなく何か工夫を）
    - 常微分方程式
      - 質点の運動・・・
    - モンテカルロ法
      - ランダムウォーク・・・

もちろん、
プログラムも

- 提出先: is-komaba@lyon.is.s.u-tokyo.ac.jp
- Subject: is-komaba report 2
- 〆切: 12月XX日（過ぎても受け付けます）

レポート課題

- 実行結果はグラフや画像の形で表示しよう。
  - 質点の軌跡
  - 二次元のランダムウォーク（軌跡や分布）
  - グラフ

```
def plot(xmin,xmax,ymin,ymax,n)
    graph = make2d(n+1,n+1)
    for i in 0..n
        y = f(xmin+1.0*i*(xmax-xmin)/n)
        graph[n-1.0*(y-ymin)/(ymax-ymin)*n][i] = 1
    end
    graph
end
```

```
load("./max.rb")
load("./abs.rb")

def draw_line(x0,y0,x1,y1,a)
  n=max(abs(x1-x0), abs(y1-y0))
  for i in 0..n
    x = x0+(x1-x0)*(i*1.0/n)
    y = y0+(y1-y0)*(i*1.0/n)
    if 0 <= y+0.5 && y+0.5 < a.   length() &&
       0 <= x+0.5 && x+0.5 < a[0].length()
      a[y+0.5][x+0.5]=1
    end
  end
end
```

draw_line.rb

レポートの例

- 二次元のランダムウォーク
  - 軌跡（[0,1]の乱数から移動方向を求める）
  - たとえば、100歩目にたどり着いた点の分布
- 積分
  - 楕円関数のグラフ
  - ガンマ関数のグラフ
- 微分方程式
  - 質点の軌跡（たとえば、太陽と地球と月）
  - Lotka-Volterraのグラフ
  - カオス力学系の軌跡

常微分方程式の解法(参考)

1. Euler Method
2. Runge-Kutta Method

常微分方程式の数値解法

- 高階(*n*階)の常微分方程式は一般に *n*組の連立1階常微分方程式に帰着される
- 1階の常微分方程式の解法が基本となる
- (1階)常微分方程式の数値解法にはTaylor展開を利用する場合が多い

常微分方程式 $\frac{dy}{dx} = f(x,y)$ を初期条件 (x_0, y_0) で解く場合

初期条件近傍の解 $(x_0 + h, y(x_0 + h))$ はTaylor展開で求まる

$$y(x_0 + h) = y_0 + h \cdot \left.\frac{dy}{dx}\right|_{x_0} + \frac{1}{2} h^2 \cdot \left.\frac{d^2 y}{dx^2}\right|_{x_0} + \ldots$$

$$= y_0 + h \cdot f(x_0, y_0) + O(h^2)$$

Euler法

常微分方程式 $\frac{dy}{dx} = f(x, y)$ を初期条件 (x_0, y_0) で解く場合

(x_i, y_i) を起点としてその近傍 (x_{i+1}, y_{i+1}) を以下のように逐次求める

$$y_{i+1} \cong y_i + (x_{i+1} - x_i) \cdot f(x_i, y_i)$$

(Eulerの公式)

任意の値 (x_n, y_n) はこの漸化式を数値計算することにより

数列として求めることができる

$$(x_0, y_0) \Rightarrow (x_1, y_1) \Rightarrow (x_2, y_2) \Rightarrow \ldots \Rightarrow (x_n, y_n)$$

問題点：精度が良くない・誤差が蓄積する

簡単な例

$$x = y^2$$

$$dx = 2ydy$$

$$\frac{dy}{dx} = \frac{1}{2y}$$

$$y(1) = 1$$

$$y(2) = ?$$

```
def euler(n)
   y = 1.0
   h = 1.0/n
   for i in 0..n-1
      y = y + (1.0/(2*y))*h
   end
   y
end
```

Taylor展開・数値解法の問題

- アイデア：Euler法（1次）よりも高次の微分係数を用いる方法が考えられる
- 問題点：計算機では高次の導関数を求める数式処理が一般に容易ではない

例）

$$\frac{dy}{dx} = f(x,y) = \sin x + \sin y$$

$$y(x_0 + h) = y_0 + h \cdot \left.\frac{dy}{dx}\right|_{x_0} + \frac{1}{2}h^2 \cdot \left.\frac{d^2y}{dx^2}\right|_{x_0} + \ldots$$

$$\frac{d^2y}{dx^2} = \cos x + y'\cos y = \cos x + (\sin x + \sin y)\cos y,$$

$$\frac{d^3y}{dx^3} = -\sin x + (\cos x + y'\cos y)\cos y - y'\sin y(\sin x + \sin y)$$

ではどうするか？

Runge-Kutta法

- 別の方法でTaylor級数と同値なものを計算する方法を考える
- アイデア：高次導関数を求めるのではなく、高次導関数の値を差分で近似する

$$k_1 = h \cdot f(x_0, y_0)$$
$$k_2 = h \cdot f(x_0 + h, y_0 + \alpha k_1)$$
$$y(x_0 + h) = y(x_0) + w_1 k_1 + w_2 k_2$$

※高次導関数が現れていないことに注意

上の式で $y(x_0 + h)$ の値が、h の2次の項までは、

Taylor級数（下の式）と一致するように α, w_1, w_2 を決めたい

$$y(x_0 + h) = y_0 + h \cdot \left.\frac{dy}{dx}\right|_{x_0} + \frac{1}{2} h^2 \cdot \left.\frac{d^2 y}{dx^2}\right|_{x_0} + O(h^3)$$

Runge-Kutta法（続き）

$k_1 = h \cdot f(x_0, y_0) = O(h)$ に注意すると $O(k_1^2) = O(h^2)$

$k_2 = h \cdot f(x_0 + h, y_0 + \alpha k_1)$

$= h \cdot \left[f(x_0, y_0) + \frac{\partial f}{\partial x} h + \frac{\partial f}{\partial y} \alpha k_1 + O(h^2) \right]$

$O(k_1^2)$ の項は $O(h^2)$ に含まれる

そこで

$w_1 k_1 + w_2 k_2 = h \cdot \left[w_1 f + w_2 f + (w_2 \frac{\partial f}{\partial x} + w_2 \alpha \frac{\partial f}{\partial y} f) h + O(h^2) \right]$

と

$h \cdot \left. \frac{dy}{dx} \right|_{x_0} + \frac{1}{2} h^2 \cdot \left. \frac{d^2 y}{dx^2} \right|_{x_0} = hf + \frac{h^2}{2} \left(\frac{\partial f}{\partial x} + f \frac{\partial f}{\partial y} \right)$ (Taylor展開 2次まで)

の両式が $f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$ に関係なく等しくなるためには

$w_1 + w_2 = 1 \quad w_2 = 1/2 \quad \alpha w_2 = 1/2$ 従って $w_1 = w_2 = 1/2 \quad \alpha = 1$

Runge-Kutta法（続き）

- 従って、以下の方法での数値解法は、h の2次の項まではTaylor級数と一致する

2次の
Runge-Kutta法

$$k_1 = h \cdot f(x_0, y_0)$$
$$k_2 = h \cdot f(x_0 + h, y_0 + k_1)$$
$$y(x_0 + h) = y(x_0) + \frac{1}{2}(k_1 + k_2)$$

これを2次のRunge-Kutta法と呼ぶ

同様の手法用いることにより、
高次のRunge-Kutta法を導くこともできる

簡単な例

$$x = y^2$$

$$dx = 2ydy$$

$$\frac{dy}{dx} = \frac{1}{2y}$$

$$y(1) = 1$$

$$y(2) = ?$$

```
def runge_kutta(n)
   y = 1.0
   h = 1.0/n
   for i in 0..n-1
      k1 = ...
      k2 = ...
      y = y + (k1+k2)/2.0
   end
   y
end
```

Runge-Kutta法（4次）

- Runge-Kutta法は原理的に高次のものを用いることができる
- 4次の場合は、 k_1, k_2, k_3, k_4 の項を計算するが、このうち k_1, k_2, k_3 と h にかかる係数に任意性がある（Taylor級数と一致するという要件では一意に決まらない）

4次の
Runge-Kutta法

$$k_1 = h \cdot f(x_0, y_0)$$

（ここでは結果だけ示すが、自分で構築して見よ）

$$k_2 = h \cdot f(x_0 + h/2, y_0 + k_1/2)$$

$$k_3 = h \cdot f(x_0 + h/2, y_0 + k_2/2)$$

$$k_4 = h \cdot f(x_0 + h, y_0 + k_3)$$

$$y(x_0 + h) = y(x_0) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

連立微分方程式

- 物理現象の多くは2階の微分方程式で記述できる

一般に 2階微分方程式 $F(\frac{d^2y}{dx^2}, \frac{dy}{dx}, x, y) = 0$ は

変数変換をすることにより、連立１次微分方程式
（各変数に関する1次微分方程式の集合）に帰着できる

$$\frac{dy}{dx} = f_1(x,y,z) \qquad \frac{dz}{dx} = f_2(x,y,z)$$

例）調和振動子の運動方程式 $m\frac{d^2x}{dt^2} = -kx$ において

$\frac{dx}{dt} = y$ とおくと $\frac{dx}{dt} = y$ $\frac{dy}{dt} = -\frac{k}{m}x$ となる

同様に高階の微分方程式も連立1次微分方程式に帰着できる

連立微分方程式の数値解法

- 連立微分方程式の場合も同様にRunge-Kutta法を適用できる

$$\frac{dy}{dx} = f_1(x,y,z) \qquad \frac{dz}{dx} = f_2(x,y,z)$$

$$k_1 = h \cdot f_1(x_0, y_0, z_0)$$
$$k_2 = h \cdot f_1(x_0 + h/2, y_0 + k_1/2, z_0 + m_1/2)$$
$$k_3 = h \cdot f_1(x_0 + h/2, y_0 + k_2/2, z_0 + m_2/2)$$
$$k_4 = h \cdot f_1(x_0 + h, y_0 + k_3, z_0 + m_3)$$

$$y(x_0 + h) = y(x_0) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$m_1 = h \cdot f_2(x_0, y_0, z_0)$$
$$m_2 = h \cdot f_2(x_0 + h/2, y_0 + k_1/2, z_0 + m_1/2)$$
$$m_3 = h \cdot f_2(x_0 + h/2, y_0 + k_2/2, z_0 + m_2/2)$$
$$m_4 = h \cdot f_2(x_0 + h, y_0 + k_3, z_0 + m_3)$$

$$z(x_0 + h) = z(x_0) + \frac{1}{6}(m_1 + 2m_2 + 2m_3 + m_4)$$

Runge-Kutta法の例

- 平面上で質点*M*の周りを運動する質点*m*の軌跡

質点*M*の座標系で質点*m*の座標を (x, y) とする
このとき運動方程式は、

$$\frac{d^2x}{dt^2} = -\frac{GMmx}{r^3} \quad \frac{d^2y}{dt^2} = -\frac{GMmy}{r^3} \quad r = \sqrt{x^2 + y^2}$$ となる

ここで変数変換を行い、 $x \Rightarrow u_0 \quad y \Rightarrow u_1 \quad \frac{dx}{dt} \Rightarrow u_2 \quad \frac{dy}{dt} \Rightarrow u_3$

$$\frac{du_0}{dt} = u_2 \qquad \frac{du_1}{dt} = u_3 \qquad r = \sqrt{u_0^2 + u_1^2}$$

$$\frac{du_2}{dt} = -\alpha\frac{u_0}{r^3} \qquad \frac{du_3}{dt} = -\alpha\frac{u_1}{r^3} \qquad \alpha = GMm$$

4連1次微分方程式が得られるので、これを4次のRunge-Kutta法で解いてみよう

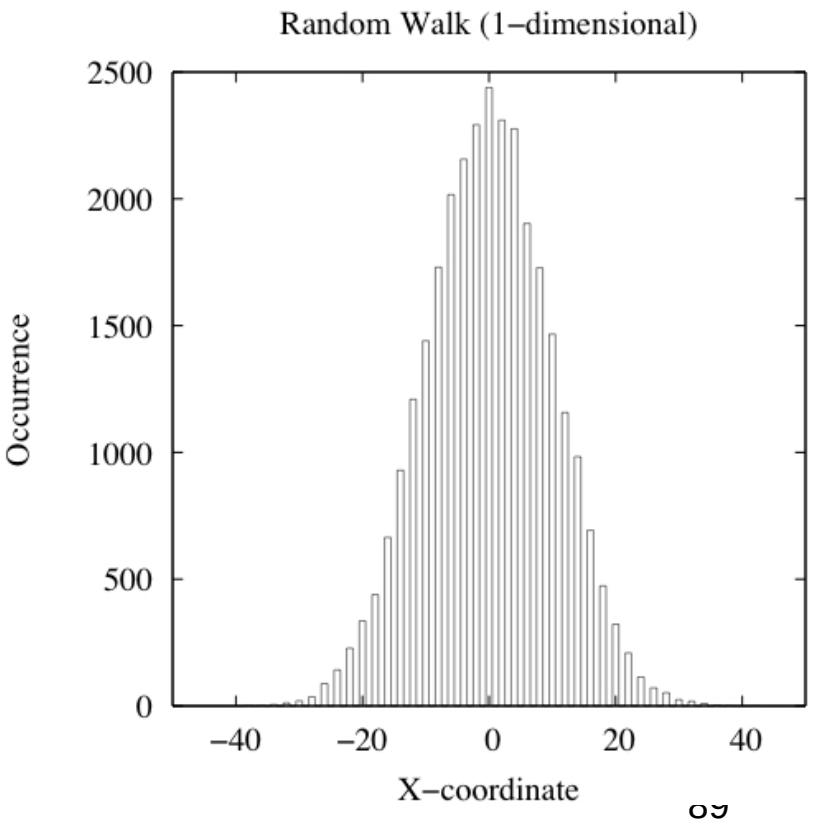
非決定論的問題

Random Walk

1次元格子上をm個の粒子が原点を起点とし、互いに独立に確率1/2で+1あるいは-1移動する。各粒子がn回移動した後、格子点上のどの点にどれだけいるか統計をとり、粒子の位置の確率分布をしらべよう。

```
m = 30000    #30000個の粒子
n = 100      #粒子の移動の回数
hist = make1d(2*n)
for i in 1..m
  x=0
  for j in 1..n
    p = rand()  #一様乱数を確率過程に用いる
    if p>0.5
      x = x+1    # 確率1/2で+1移動
    else
      x = x-1    # 確率1/2で-1移動
    end
  end
  hist[n+x] = hist[n+x]+1
end
```

実行結果



この確率過程は正規分布に従う？