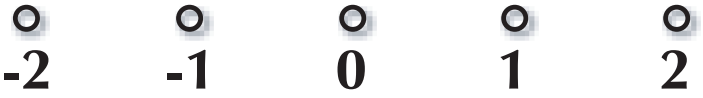
Swift Programming Lecture 9

Mathematical Functions

Tatsuo Minohara

整数と実数の違い

Integer(離散数)



Real(連続数)

実数の指数表現と正規化

- 0.000567 ⇒ 5.67×10^{-4} ⇒5.67e-4
- 1230000.0 ⇒ 1.23×10^{6} ⇒ 1.23e6

- 0を少なくするように小数点を動かして、
- 仮数 × 基数（10）指数
- 正規化（Normalization）
- 小数点が浮動するので、実数は「浮動小数点数」（Floating Point Number）と呼ばれる

型の変換

- 暗黙の型変換
  - ➡整数から実数へ
  - ➡実数が出てくると実数へ自動的に変換
  - ➡実数除算が出てくると実数へ自動的に変換

- 明示的な型変換
  - ➡ 型変換の関数を使う

型変換の関数

- 型変換の関数
  - Int(値) …整数型へ変換
  - Float(値) …単精度実数型へ変換
  - Double(値) …倍精度実数型へ変換
  - String(値) …文字列型へ変換
- 実数に変換したい場合
  - Float(7) ⇒ 7.0f
  - Double("34.5") ⇒ 34.5
- 実数を整数に変換する場合（小数部が切り捨て）
  - Int(56.4e5) ⇒ 5640000
  - Int(43.2) ⇒ 43

実数の誤差

- 丸め誤差
  - ➡有限ビット数で表わすのでどこかで四捨五入や切り捨てが起こる
- 情報落ち（桁落ち）
  - ➡違う大きさの数を足したり、引いたりしたとき
- 打切り誤差
  - ➡指定された桁で表現を打ち切る
  - ➡循環小数
  - ➡無理数：$\sqrt{2}$、e, πなど

実数の比較

- 誤差を考慮して比較しなければならない

➡× **if** x == 0.1 {　// うまくいかない

➡**var** *epsilon* = 0.0001　//誤差許容範囲

➡**if** 0.1-*epsilon* < x and x < 0.1+ *epsilon* { }

実数の内部表現

- 浮動小数点表現
  - IEEEの国際規格になっている
  - IEEE-754規格
    - 単精度（32bit）、倍精度（64bit）
- 表現範囲
  - 単精度（32 bit）

    $1.40129846432481707\times10^{-45}$〜$3.40282346638528860\times10^{+38}$
  - 倍精度（64 bit）

    $4.94065645841246544\times10^{-324}$〜$1.78769313486231570\times10^{+308}$

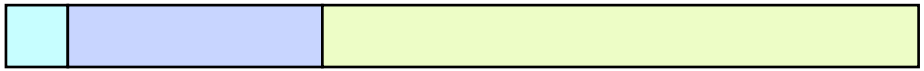
指数表現と正規化

- 指数表現
  - 仮数部 + 基数および指数部
  - $5.67 \times 10^{8} \rightarrow 567000000.0$
- 正規化
  - 仮数部の桁数が限られているので0を少なく
  - $0.000000436 \rightarrow 4.36 \times 10^{-7}$

IEEE-754の浮動小数点数の構造

- 符号と指数部と仮数部から１つの数値が構成される

1.



符号	指数部	仮数部
正…0	単精度…8bit	単精度…23bit
負…1	倍精度…11bit	倍精度…52bit
	バイアス表現	正規化と、けち表現

- 指数部のかさ上げ値（bias）と指数部で表わせる範囲は以下の通り、すべて2を基数とする
  - 32bitの場合　+127　-126〜+127
  - 64bitの場合　+1023　-1022〜+1023
- この範囲を超えると指数部オーバーフロー・指数部アンダーフローとなり信頼性がなくなる

SwiftでのIEEE-754表現

- float.hex(実数)で16進数で表現された内部表現を見ることができる
- 例：float.hex(3740.0)
  - '0x1.d38000000000p+11'
- この表現は、以下の書式になる
  - [sign] ['0x'] integer ['.' fraction] ['p' exponent]
  - sign は必須ではなく、 + と - のどちらか。 integer （整数部）と fraction （小数部）は 16 進数の文字列で、 exponent は 10 進数で符号もつけられる。大文字・小文字は区別されず、最低でも 1 つの 16 進数文字を整数部もしくは小数部に含む必要があります。

IEEE-754 特殊な数の表現

数	単精度	倍精度
不定	FFC00000	FFF8000000000000
非数（-NaN）	FF800001～FFBFFFFF	FFF0000000000001～ FFF7FFFFFFFFFFFF
無限大（-∞）	FF800000	FFF0000000000000
非正規化数（–）	807FFFFF～FF7FFFFF	800FFFFFFFFFFFFF～ FFEFFFFFFFFFFFFF
ゼロ（-0）	80000000	“8000000000000000”
非数（+NaN）	7F800001～7FFFFFFF	7FF0000000000001～ 7FFFFFFFFFFFFFFF
無限大（+∞）	7F800000	7FF0000000000000
非正規化数（＋）	00000001～007FFFFF	0000000000000001～000FFFFFFFFFFFFF
ゼロ（+0）	“00000000”	“0000000000000000”

絶対誤差と相対誤差

- 真値をα、そのコンピュータによる計算結果（近似値）をxとしたとき、誤差と絶対誤差・相対誤差を以下の式で評価する
  - 誤差 $$E = x - \alpha$$
  - 絶対誤差 $$E_A = |E| = |x - \alpha|$$
  - 相対誤差 $$E_R = \frac{E}{\alpha} = \frac{x - \alpha}{\alpha}$$
- 誤差の限界
  - ある正の値εが以下の式を満たす場合

$$x - \varepsilon < \alpha < x + \varepsilon$$

  - このεの値が0に近いほど、近似値xは真値αに近いことになり、このεを誤差の限界という
- 計算機イプシロン
  - 次の式を満たすεの最小値は、コンピュータで1に加減算できる最小値を表わし、誤差の評価や収束判定指数の決定に利用される

数値関係の定数

- Double.e…自然対数の底（ネイピア数）
- Double.pi…円周率
- Double.inf…正の無限大（負の無限大は、-infを用いる）

誤差を少なくするアルゴリズム(3)

- $e = \lim_{n \to \infty} (1+1/n)^n$
- pow（べき乗を求める関数）を使って精度を上げながら求める

```
import Foundation
n = 1
while n <= 10000000000000:
    e = pow( 1+1.0/n, n )
    n *= 10
```

IEEE-754浮動小数点数用の関数

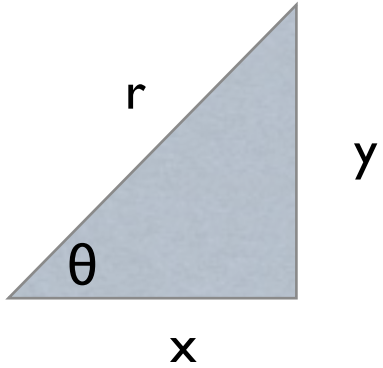
- Foundationライブラリに入っている
  - **import** Foundationが必要
- C/C++からのライブラリがそのまま引き継がれている。
  - C言語のときは、#include <h>が必要
  - Java言語では、標準でMathクラスのライブラリ
  - Python言語では、**import** mathが必要

整数との変換

- ceil(x)…$\lceil x \rceil$　大きいか等しい一番小さな整数
  - ➡計算結果は実数
- floor(x)…$\lfloor x \rfloor$　小さいか等しい一番大きな整数
  - ➡計算結果は実数
- round(x)…四捨五入
- round(x, n)…小数点以下n+1桁で四捨五入
- trunc(x)…小数部を切り捨てる、Int(x)と等しい

三角関数

- cos(θ) → x / r
- sin(θ) → y / r
- tan(θ) → y / x



- acos(x / r) → θ (r=1,0～π)
- asin(y / r) → θ (r=1,-π/2～π/2)
- atan(y/x) → θ (-π/2～π/2)
- atan2(y, x) → θ
- hypot(x, y) → r

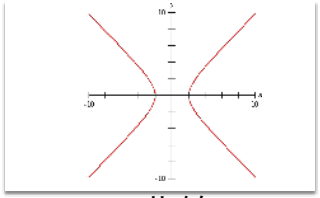
Radian体系とDegree体系、分秒

- Radianの角度 = radians(Degreeの角度)
- Degreeの角度 = degrees(Radianの角度)
- 経度、緯度には、Degree角度以外に分、秒を用いる分…１度の60分の１　　秒…１分の60分の１

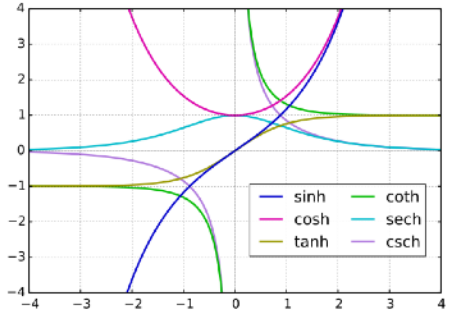
degree	0°	45°	90°	180°	270°	360°
radian	0	π/4	π/2	π	3π/2	2π

mathモジュールの双曲線関数

- 双曲線に対して定義された三角関数
  - sinh(x)…双曲線正弦
  - cosh(x)…双曲線余弦
  - tanh(x)…双曲線正接
  - asinh(x)…逆双曲線正弦
  - acosh(x)…逆双曲線余弦
  - atanh(x)…逆双曲線正接



双曲線



双曲線関数と三角関数の双対性

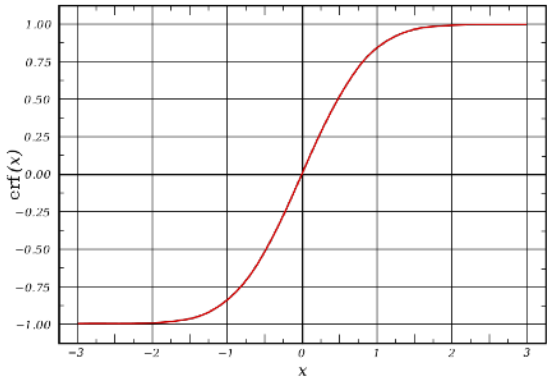
- $e^{i\theta} = \cos\theta + i\sin\theta$
  $e^{-i\theta} = \cos\theta - i\sin\theta$

- $$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

  $$\cosh\theta = \frac{e^{\theta} + e^{-\theta}}{2}$$

mathモジュールの特殊関数

- factorial(n)…nの階乗（正の整数）
- gamma(z)…ガンマ関数（zの階乗：zは複素数）
- gcd(a, b)…aとbの最大公約数（aとbは整数）
- erf(r)…誤差関数、シグモイド形状の特殊関数



$$\varsigma_1(x) = \frac{1}{1+e^{-x}} = \frac{\tanh(x/2)+1}{2}$$

標準シグモイド関数

対数

- $x = a^p$　aは基数　pは指数　$64 = 8^2$　pow(8, 2)
- $p = \log_a x$　xの対数　$2 = \log_8 64$　$\log_8(64)$

- 対数は小さなものから大きなものまで表わせる
- 掛け算を足し算にできる log xy = log x+log y
- 割り算を引き算にできる log x/y = log x - log y
- 桁数を求めることができる

対数の公式

- 基数を変えるためには、

➡$\log_{10}(x) = \log_e(x) / \log_e(10)$

➡$\log_2(x) = \log_e(x) / \log_e(2)$

$$\log_a x = \frac{\log_b x}{\log_b a}$$

$$(\log_a x)\,(\log_b a) = \log_b a^{\log_a x} = \log_b x$$

自然対数と常用対数

- log(値) …　自然対数
- log10(値) …　常用対数
- log2(値) …　2を底とする対数

- pow(x, n) …　x のn乗を求める
- exp(n) … e (自然対数の底）のn乗を求める

その他の関数

- sqrt(x ）
  - xの平方根を求める　　pow(x, 0.5 ）
- abs(x ）
  - xの絶対値を求める　　abs(-9) ⇒ 9
- fabs(x)
  - xの絶対値を求める　　fabs(-9.8) ⇒ 9.8
- min(list), max(list)
  - listの中で最小値、最大値を返す

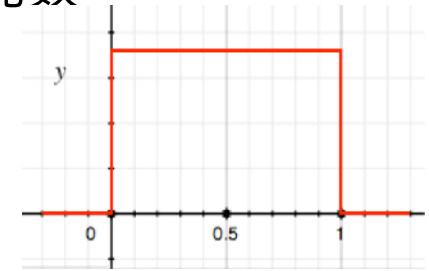
    min(4, 1, 3) ⇒ 1

random関数

- **import** Foundationが必要
- random( )
  - 0から1未満の乱数（任意の数）を返す
- uniform(a, b)
  - a以上b以下の乱数（任意の数）を返す
- randint(m, n)
  - m以上n以下の乱数（整数）を返す
- gauss(m, sigma)
  - gauss分布（正規分布）で平均がm、標準偏差がsigmaの乱数を返す

正規分布の乱数

- 一様乱数



- 正規分布の乱数