toc

Index

[第1回：▶ 簡単なグラフを描く](@id ch01)

▪ 対話形式で使う

本文では、対話形式で、Julia を利用する。

Juliaをコマンドラインから利用している場合は、
プロンプト `julia>` が印字され、利用者の入力を待っている。
命令を打ち込み、ENTERキーを押すと、
その命令を評価した結果が出力される。

```
1
```

上では、1 という文字の並びから、
1 という数を内部で作成し、
それを計算の結果として印字したのである。

計算機側から見ると、
利用者の入力を読み込み(Read)、
入力された命令を評価し(Eval)、
その結果を印字する(Print)ことを、繰り返す(Loop)。
この４つの頭文字をとって、
対話型利用のことを REPL とも呼ぶ。

Jupyter notebookを用いる場合には、
Code cellが表示されている。
ここに命令を打ち込み、SHIFT + ENTER キーを押すと、
その命令を評価した結果が出力される。

▪ 電卓として使う

数と数との加減乗除をしてみよう。

加算は + 、
減算は - の文字を使う。
乗算は * (アスタリスク astarisk と読む)、
除算は (% ではなくて) / (スラッシュ slash と読む) の文字を使う。

```
1 + 2
3 * 4
```

数式と同じように、乗算と除算は、加算・減算に優先する。
計算の順序を変えるには、括弧 (と) との組を用いる

```
2 + 3 * 4
(2 + 3) * 4
```

除算の結果は、小数となる。

```
2 / 2
1 / 3
5 / 2
```

▪ 変数に値を代入する

値には、名前（名札、ラベル）をつけることができる。
この名前を変数といい、名前をつける操作を「値を変数に代入する」という。
変数には、色々な種類の値を代入できる。

変数を評価すると、変数の値となる。

```
# 変数 x に 値 2 を代入する
x = 2
# 変数 x の値を用いる
x + 1
# 変数 x に 別の値 3 を再代入する
x = 3
```

はコメントである。
から行末までの文字は全て無視される。

▪ ベクトル

角括弧 [と] との間に、カンマ , で区切って
数を並べたものを、(数の)ベクトルという。
ベクトルは、縦に印字される (縦ベクトル)。

```
[1,3,2]
```

変数には、数ベクトルも代入できる。

```
xs = [1,2,2,1]
ys = [1,1,3,1]
```

ベクトルのスカラー倍やベクトル同士の和を計算できる。
演算子 .+ の最初のピリオド . は、各要素に対する演算を意味する。

```
xs = [1,2,2,1]; nothing #hide
ys = [1,1,3,1]; nothing #hide
xs * 2
xs .+ ys
```

▶ 三角形を描く

上のベクトル xs, ys で一つづつ数を取り出して、
それらを xy座標とする点を結んで、図形を描いてみる。

グラフを描画するのに、PyPlot パッケージを導入する。
PyPlotを導入されると使えるようになる plot関数は、
点を結んで、図を描く命令である。

```
# PyPlot パッケージの導入
using PyPlot
clf() #hide
xs = [1,2,2,1]
ys = [1,1,3,1]
# プロット
plot(xs,ys)
savefig("ch01-tri-plot.svg"); nothing # hide
```

練習01-1 xs, ys の値を変えて、別の図形を表示させてみよ、

▶ Range型

Base.colon (https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.colon)

Rangeは、等差数列である。
コロン：演算子を用いると、Range型の量を作ることができる。

二つの数をコロンで区切った量 a:b は、
aから 1づつ増やして、bを超えるまでの数からなる等差数列である。
三つの数をコロンで区切った量 a:b:c は、
aから b づつ増やして、cを超えるまでの数からなる等差数列である。

```
1:5
xs=0:0.1:1
```

collect関数を用いルト、
Range型から、各要素を取り出したベクトルに変換できる。

```
xs=0:0.1:1; nothing #hide
collect(xs)
```

Rangeの型を保ったまま、
各要素のスカラー倍、同じ数の和差を一斉に適用できる。

```
xs=0:0.1:1 #hide
# 各要素を 2倍する
xs * 2
# 各要素に 1を加える
xs+1
```

plot関数に対して、二つのコレクション xs、ysを渡すと、
xs、ys から一つづつ要素を取り出し、
これらを x, y 座標とする点を結んで、図形が描かれるのであった。

グラフを描くには、ysとして、 xs から計算した値を与えればよい。

二つの直線 $y=-x$ と $y=2x-1$ のグラフを描いてみよう。

```
# PyPlot パッケージの導入
using PyPlot
clf() #hide
xs=-1:0.1:1
# プロット
plot(xs,-1*xs)
plot(xs,2*xs-1)
savefig("ch01-graph-plot.svg"); nothing # hide
```

== ◀ 練習
別の直線を描いてみよ。

★ 今回のまとめ

- 対話形式の使い方
- 数の四則演算
- PyPlotパッケージを用いた図形とグラフの描画
- ベクトルと等差数列

[第2回：▶ 複数のグラフを描く](@id ch02)

▪ リテラル

リテラル (literal) とは、
「文字の並び」の通りに解釈される量をいう。

1や1.1はリテラルである。
それぞれ、整数 1, 小数1.1 という値として評価されるからである。

▪ 文字列

「文字の並び」として表される量が、文字列である。
文字列のリテラルは、ダブルクォート " で囲まれた文字の並びである。

```
"Hello world"
```

文字列を連結するには、演算子 * を用いる。

```
h="Hello"
w="world"
h*w
h*" "*w
```

数字を表す文字列を作るには、string関数を用いる。

```
string(0)
string(1)
string(1.1)
```

▪ for文

[Repeated Evaluation: Loops (https://docs.julialang.org/en/v0.6.4/manual/control-flow/#man-loops-1)](https://docs.julialang.org/en/v0.6.4/manual/control-flow/#man-loops-1)

一つずつ要素を取り出すことができる量を、コレクションという。
ベクトルやRangeは、コレクションである。
for文を用いると、コレクションから要素を一つづつ取り出して、
繰り返して、計算を行うことができる。

次の例では、変数 iに、ベクトルの各要素を入れて、end文までの計算を繰り返す。
ここで、@show i は、式 iの値を表示する関数(マクロ)である。

```
for i in [1,3,2]
  @show i
end
```

Rangeでも、上と同様である。

string関数の結果を表示する。
println 関数は、印字してから、改行する命令である。

```
for i in 1:5
  println( string(i) )
end
```

▪ リスト内包表記

[Comprehensions (https://docs.julialang.org/en/v0.6.4/manual/arrays/#Comprehensions-1)](https://docs.julialang.org/en/v0.6.4/manual/arrays/#Comprehensions-1)

[2x for x in [1:2]]

▶ グラフに凡例を加える

グラフの凡例(lengend)とは、グラフに描かれた曲線を区別するための説明である。
PyPlotパッケージで書かれたグラフに凡例を追加するには、以下のようにする。

まず、plot関数に label=文字列 の形式で、
その曲線に付与する文字列を指定する。
全ての曲線を描いた後に、legend関数を付与する。

```
using PyPlot
clf() #hide
xs=-1:0.1:1
plot(xs,-1*xs,  label="y=-x")
plot(xs,2*xs-1, label="y=2x-1")
legend()
savefig("ch02-leg1-plot.svg"); nothing # hide
```

for文を用いて、直線群 y=ax のグラフに凡例を加えてみる。

```
using PyPlot
clf() #hide
xs=-1:0.1:1
for a in 1:5
  plot(xs, a*xs, label="y="*string(a)*"x" )
end
legend()
savefig("ch02-leg2-plot.svg"); nothing # hide
```

▶ 冪乗関数を描く

Base.:^ (https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.:^-Tuple{Number,Number})

x^yは、冪（べき, power）ないし冪乗（べきじょう）x^yを表す。
xを底 (base)、yを冪指数 (exponent)という。

```
2^2
2^3
2^4
```

冪指数が負の整数のとき、冪の計算は失敗する (DomainError：定義域外)。
代わりに、負の小数を指定する。

```
2^-2 # DomainError
2^-2.0
```

ベクトルや等差級数の各要素で、
冪乗を計算するには　演算子 ^の前にピリオド . を加える。

```
2.^[2,3,4]
(2:4).^2
2:4.^2    # `^`は `:` よりも優先度が高い
```

冪乗 $y=x^a, (a=2,3,4,5)$ のグラフを描こう。

plt[:axes]()[:set_aspect]("equal") は、
グラフの縦横比 (アスペクト比 aspect ratio) を等しくする命令である。

```
using PyPlot
clf() #hide
xs=0:0.1:1
plot(xs, xs.^2 )
plot(xs, xs.^3 )
plot(xs, xs.^4 )
plot(xs, xs.^5 )
plt[:axes]()[:set_aspect]("equal")
savefig("ch02-po1-plot.svg"); nothing # hide
```

for文を使って、繰り返しの処理をまとめる。
xlim, ylim を使うと、グラフの描画範囲を調整できる。

xlim(a,b) は、x軸の描画を a から b の範囲に限定する命令である。

ylim(a,b) は、y軸の描画を a から b の範囲に限定する命令である。

さらに、凡例を追加する。

```
using PyPlot
clf() #hide
# 間隔を狭めた
xs=0:0.05:2
for a in 2:5
  plot(xs, xs.^a , label="y=x^"*string(a) )
end
legend()
# 描画範囲を設定
xlim(-0.05,2)
ylim(-0.05,2)
plt[:axes]()[:set_aspect]("equal")
savefig("ch02-po2-plot.svg"); nothing # hide
```

▶ ローレンツ関数を描く

以下の曲線を、
ローレンツ関数 (Lorentz function)という。

$y = \frac{\frac{\gamma}{2}}{(x-x_0)^2+\left(\frac{\gamma}{2}\right)^2}$
$\gamma > 0$

パラメータを $x_0 = 0, \gamma = 2$ のように選ぶと、
以下のように簡単な形となる。

$y = \frac{1}{x^2+1}$

まず、この曲線を描いてみる。
各要素に対して除算を行うため、/の前にも、ピリオド.を付与する。

```
xs=-3:0.5:3
1 ./ (xs.^2+1)
```

上のグラフを描こう。

```
using PyPlot
clf() #hide
xs=-3:0.1:3
plot(xs, 1 ./ (xs.^2+1) )
savefig("ch02-lo1-plot.svg"); nothing # hide
```

以下のように、パラメータ γ を導入する。

$y = \frac{\frac{\gamma}{2}}{x^2+\left(\frac{\gamma}{2}\right)^2}$

$\gamma=0.5,1,2$ について、この曲線を描く。

```
using PyPlot
clf() #hide
xs=-3:0.05:3
gamma=0.5
plot(xs, (gamma/2) ./ (xs.^2+(gamma/2)^2), label=gamma )
gamma=1.0
plot(xs, (gamma/2) ./ (xs.^2+(gamma/2)^2), label=gamma )
gamma=2.0
plot(xs, (gamma/2) ./ (xs.^2+(gamma/2)^2), label=gamma )
legend()
plt[:axes]()[:set_aspect]("equal")
savefig("ch02-lo2-plot.svg"); nothing # hide
```

for文を用いて、簡潔に書こう。（結果のグラフは同じである)

```
using PyPlot
clf() #hide
xs=-3:0.05:3
for gamma in [0.5,1.0,2.0]
  plot(xs, (gamma/2) ./ (xs.^2+(gamma/2)^2), label=gamma )
end
legend()
plt[:axes]()[:set_aspect]("equal")
savefig("ch02-lo3-plot.svg"); nothing # hide
```

次の性質がある。

- 点 $x=0$ で 最大値 $y = \frac{2}{\gamma}$
- 点 $x=\pm\frac{2}{\gamma}$ で、$y = \frac{1}{2}\frac{2}{\gamma}$ となること。

２つ目の性質を観察するため、最大値に対する比を描いてみる。

```
using PyPlot
clf() #hide
xs=-3:0.05:3
for gamma in [0.5,1.0,2.0]
  plot(xs,
    (gamma/2) ./ (xs.^2+(gamma/2)^2) / (2/gamma),
    label=gamma )
end
legend()
axhline(0, color="k", lw=0.5)
savefig("ch02-lo4-plot.svg"); nothing # hide
```

ここで、γを非常に小さくすると、
Diracのデルタ関数 (Dirac delta function)の近似となる。

なお、$x_0 \neq 0$ の場合は、読者の検討の課題とする。

★ 今回のまとめ

- 文字列
- for文
- グラフに凡例を加える
- 冪乗関数
- ローレンツ関数

第3回：▶ 連続な曲線を描く

▶ 関数が連続とは

$\lim_{x \longrightarrow a} f(x)$
が存在して、その値が $f(a)$ に等しいとき、
「関数 $f(x)$ は $x=a$ で連続という」

▶ 正弦関数・余弦関数を描く

- 正弦 $y = \sin{x}$
- 余弦 $y = \cos{x}$

ラジアン単位
sin, cos

```
using PyPlot
clf() #hide
xs=-2pi:pi/360:2pi
plot(xs, cos.(xs), label="cos")
plot(xs, sin.(xs), label="sin")
xlabel("radian")
legend()
savefig("ch03-sin1-plot.svg"); nothing # hide
```

円周率単位
sinpi, cospi

```
using PyPlot
clf() #hide
xs=-2:1/360:2
plot(xs, cospi.(xs), label="cospi")
plot(xs, sinpi.(xs), label="sinpi")
xlabel("pi")
legend()
savefig("ch03-sin2-plot.svg"); nothing # hide
```

角度単位

sind, cosd

```
using PyPlot
clf() #hide
xs=-360:1:360
plot(xs, cosd.(xs), label="cosd")
plot(xs, sind.(xs), label="sind")
xlabel("degree")
legend()
savefig("ch03-sin3-plot.svg"); nothing # hide
```

▶ 楕円を描く

楕円
$ \left(\frac{x}{a}\right)^2+\left(\frac{x}{b}\right)^2 = 1$

媒介変数表示(パラメータ曲線)

$\begin{align}x &= a \cos \theta \\ y &= b \sin \theta \end{align}$

```
using PyPlot
clf() #hide
ts=0:pi/18:2pi
xs=2*cos.(ts)
ys=sin.(ts)
plot(xs,ys)
xlim(-3,3)
ylim(-3,3)
plt[:axes]()[:set_aspect]("equal")
savefig("ch03-ell1-plot.svg"); nothing # hide
```

▶ アルキメデスの渦を描く

平面座標上の点 $（x,y)$は、
極座標 (r, θ) でも表示できる。
x,yと r, θ との対応は
$\begin{align}x &= r \cos \theta \\ y &= r \sin \theta \end{align}$
である。

次の関係で結ばれた曲線を、アルキメデスの渦という。
$r = \theta$

```
using PyPlot
clf() # hide
ts=0:pi/1800:2pi
xs=ts .* cos.(ts)
ys=ts .* sin.(ts)
plot(xs, ys)
plt[:axes]()[:set_aspect]("equal")
savefig("ch03-arch1-plot.svg"); nothing # hide
```

▶ 花曲線を描く

flower curve

$r = cos(n \theta)$

```
using PyPlot
clf() # hide
n=3
ts=0:pi/1800:2pi
rs=cos.(n*ts)
xs=rs .* cos.(ts)
ys=rs .* sin.(ts)
plot(xs, ys)
plt[:axes]()[:set_aspect]("equal")
savefig("ch03-flo1-plot.svg"); nothing # hide
```

▶ 指数関数を描く

$y=a^x \quad a > 0$

```
using PyPlot
clf() #hide
xs=-10:0.01:10
plot(xs, 2.^xs)
savefig("ch03-exp1-plot.svg"); nothing # hide
```

```
clf() #hide
plot(xs, 2.^xs)
plot(xs, 3.^xs)
plot(xs, 4.^xs)
plot(xs, 5.^xs)
xlim(0,3)
ylim(0,100)
savefig("ch03-exp2-plot.svg"); nothing # hide
```

```
clf() #hide
plot(xs, 2.^xs, label="a="*string(2))
plot(xs, 3.^xs, label="a="*string(3))
plot(xs, 4.^xs, label="a="*string(4))
plot(xs, 5.^xs, label="a="*string(5))
plot(xs, 6.^xs, label="a="*string(6))
legend()
xlim(0,3)
ylim(0,100)
savefig("ch03-exp3-plot.svg"); nothing # hide
```

```
e
```

```
using PyPlot #hide
clf() #hide
for a in [2,e,3,4,5,6]
  plot(xs, a.^xs, label="a="*string(a))
end
yscale("log")
xlim(-1,3)
ylim(1e-1,1e3)
legend()
savefig("ch03-exp4-plot.svg"); nothing # hide
```

```
exp2
exp
exp10
```

```
clf() #hide
plot(xs, exp2.(xs), label="exp2")
plot(xs, exp.(xs), label="exp")
plot(xs, exp10.(xs), label="exp10")
yscale("log")
xlim(-1,3)
ylim(1e-1,1e3)
legend()
savefig("ch03-exp7-plot.svg"); nothing # hide
```

```
clf() #hide
for f in [exp2,exp,exp10]
  plot(xs, f.(xs), label=string(f))
end
yscale("log")
xlim(-1,3)
ylim(1e-1,1e3)
legend()
savefig("ch03-exp8-plot.svg"); nothing # hide
```

▶ 冪乗根を描く

$y = \sqrt_{n}{x} = x^{\frac{1}{n}}, \quad n\textrm{は整数}$

(正数を定義域とする関数)

Base.sqrt (https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.sqrt)

Base.Math.cbrt (https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.Math.cbrt)

```
sqrt(2)
sqrt(3)
sqrt(-1) # DomainError
sqrt(complex(-1,0))  # 複素数を引数に与える
```

```
using PyPlot
clf() #hide
xs=0:0.01:3
plot(xs, sqrt.(xs), label="sqrt")
plot(xs, cbrt.(xs), label="cbrt")
legend()
xscale("log")
yscale("log")
xlim(1/2,2)
ylim(1/2,2)
plt[:axes]()[:set_aspect]("equal")
savefig("ch03-pr1-plot.svg"); nothing # hide
```

```
using PyPlot
clf() #hide
xs=0:0.01:3
for f in [sqrt,cbrt]
  plot(xs, f.(xs), label=string(f))
end
legend()
xscale("log")
yscale("log")
xlim(1/2,2)
ylim(1/2,2)
plt[:axes]()[:set_aspect]("equal")
savefig("ch03-pr1L-plot.svg"); nothing # hide
```

```
using PyPlot
clf() #hide
xs=0:0.1:3
plot(xs, xs.^(1/2), label=2)
plot(xs, xs.^(1/3), label=3)
plot(xs, xs.^(1/4), label=4)
plot(xs, xs.^(1/5), label=5)
xlim(0,3)
ylim(0,3)
plt[:axes]()[:set_aspect]("equal")
savefig("ch03-pr2-plot.svg"); nothing # hide
```

```
using PyPlot
clf() #hide
xs=0:0.1:3
for n in 2:5
  plot(xs, xs.^(1/n), label=n)
end
xlim(1/3,3)
ylim(1/3,3)
xscale("log")
yscale("log")
plt[:axes]()[:set_aspect]("equal")
savefig("ch03-pr2L-plot.svg"); nothing # hide
```

▶ 対数関数

(正数を定義域とする関数)

```
log(1)
log(e)
log(e^2)
log(-1) # DomainError
```

```
using PyPlot
clf() #hide
using PyPlot
xs=0.1:0.01:100
0.1:0.01:100.0
plot(xs, log.(xs))
savefig("ch03-log1-plot.svg"); nothing # hide
```

x軸を対数で表示すると、直線で表示される。

```
plot(xs, log.(xs))
xscale("log")
savefig("ch03-log2-plot.svg"); nothing # hide
```

```
clf() #hide
plot(xs, log.(2,xs), label=string(2))
plot(xs, log.(xs), label=string(e))
plot(xs, log.(3,xs), label=string(3))
plot(xs, log.(10,xs), label=string(10))
xscale("log")
legend()
savefig("ch03-log3-plot.svg"); nothing # hide
```

```
clf() #hide
for a in [2, e, 3, 10]
plot(xs, log.(a,xs), label=string(a))
end
xscale("log")
legend()
savefig("ch03-log4-plot.svg"); nothing # hide
```

```
clf() #hide
plot(xs, log2.(xs), label="log2")
plot(xs, log.(xs), label="log")
plot(xs, log10.(xs), label="log10")
xscale("log")
legend()
savefig("ch03-log5-plot.svg"); nothing # hide
```

```
clf() #hide
for f in [log2, log, log10]
plot(xs, f.(xs), label=string(f))
end
xscale("log")
legend()
savefig("ch03-log6-plot.svg"); nothing # hide
```

▶ ダブルYグラフを描く

```
using PyPlot
xs=-2:0.1:2
fig=plt[:figure]()
ax1=fig[:add_subplot](111)
ax1[:plot](xs, -xs)
ax2=ax1[:twinx]()
ax2[:plot](xs, xs.^2)
savefig("ch04-tw1-plot.svg"); nothing # hide
```

▶ 自由落下運動を描く

鉛直上向きに投げられた球が、重力のみを感じて自由落下するとする。
時刻 $t=0$において、高さ $y=0$, 鉛直上向きの速度 v_0 とすると、
時刻 tにおける、高さ y, 鉛直上向きの速度 v は、以下のように表される。

$v =v_0 - gt$,
$y = v_0 t - \frac{1}{2}gt^2$

```
using PyPlot
clf() #hide
v0=10 # m/s
g=9.8 # m/s^2

ts=0:0.1:3 # s
vs=v0-g*ts
plot(ts,vs)
savefig("ch04-gy1-plot.svg"); nothing # hide
```

```
ys=v0*ts-g*ts.^2/2
plot(ts,ys)
savefig("ch04-gy2-plot.svg"); nothing # hide
```

```
clf() #hide
ax1=plt[:subplot]()
ax2=ax1[:twinx]()
ax1[:plot](ts,vs)
ax2[:plot](ts,ys)
savefig("ch04-gy3-plot.svg"); nothing # hide
```

続けて

```
ax1[:set_xlabel]("time /s")
ax1[:set_xlim](-0.3,2.3)
ax1[:set_ylim](-12,12)
ax2[:set_ylim](-6,6)
ax1[:set_ylabel]("velocity / m s^-1")
ax2[:set_ylabel]("height / m")

savefig("ch04-gy4-plot.svg"); nothing # hide
```

続けて

```
ax1[:axvline](10/9.8, color="k", lw=0.5)
ax1[:axvline](0, color="k", lw=0.5)
ax1[:axhline](0, color="k", lw=0.5)
savefig("ch04-gy5-plot.svg"); nothing # hide
```

▶ 複数のグラフを描く

花曲線の例で nを変えてみる。

```
using PyPlot
clf() #hide
ts=0:pi/1800:2pi
for i=1:6
  n=i+2
  ax=plt[:subplot](330+i)
  rs=cos.(n*ts)
  xs=rs .* cos.(ts)
  ys=rs .* sin.(ts)
  ax[:plot](xs,ys)
  ax[:set_xlim](-1,1)
  ax[:set_ylim](-1,1)
  ax[:set_aspect]("equal")
end
savefig("ch03-flo2-plot.svg"); nothing # hide
```

★ 今回のまとめ

- 全域で定義された関数
  ** 正弦・余弦関数
  ** 楕円
  ** アルキメデスの渦
  ** 花曲線
  ** 指数関数
- 正数を定義域とする関数
  ** 対数関数
- 複数のグラフを描く方法

第4回：▶ 不連続な曲線を描く

▶ 不連続

▶ 逆数関数を描く

```
using PyPlot
clf() #hide
xs=-3:0.1:3
plot(xs,1./xs)
xlim(-3,3)
ylim(-3,3)
plt[:axes]()[:set_aspect]("equal")
savefig("recipro1-plot.svg"); nothing # hide
```

▶ 不連続な有理式を描く

$y = \dfrac{x^3+8}{x^3+3x^2-4x-12}$

```
using PyPlot
clf() #hide
xs=-10:0.1:10
plot(xs, xs.^3+3xs.^2-4*xs-12)
ylim(-20,20)
xlim(-4,4)
axhline(0, color="k", lw=0.5)
axvline(-3, color="k", lw=0.5)
axvline(-2, color="k", lw=0.5)
axvline(2, color="k", lw=0.5)
savefig("ch03-dis1-plot.svg"); nothing # hide
```

```
clf() #hide
xs=-20:0.05:20
plot(xs, (xs.^3+8)./(xs.^3+3xs.^2-4xs-12))
ylim(-10,10)
axhline(0, color="k", lw=0.5)
axvline(-3, color="k", lw=0.5)
axvline(2, color="k", lw=0.5)
savefig("ch03-dis2-plot.svg"); nothing # hide
```

```
clf() #hide
xs=-30:0.05:30
plot(xs, (xs.^3+8)./(xs.^3+3xs.^2-4xs-12))
ylim(-10,10)
xlim(-3,3)
axhline(0, color="k", lw=0.5)
axvline(-3, color="k", lw=0.5)
axvline(-2, color="k", lw=0.5)
axvline(2, color="k", lw=0.5)
savefig("ch03-dis3-plot.svg"); nothing # hide
```

▶ 正接関数・余接関数を描く

- 正接 $y = \tan{x}$
- 余接 $y = \cot{x}$

ラジアン単位
tan, cot

```
using PyPlot
clf() #hide
xs=-2pi:pi/360:2pi
plot(xs, tan.(xs), label="tan")
plot(xs, cot.(xs), label="cot")
ylim(-1e1, 1e1)
xlabel("radian")
legend()
savefig("ch03-tan1-plot.svg"); nothing # hide
```

角度単位
tand, cotd

```
using PyPlot
clf() #hide
xs=-360:1:360
plot(xs, tand.(xs), label="tand")
plot(xs, cotd.(xs), label="cotd")
xlabel("degree")
ylim(-1e1, 1e1)
legend()
savefig("ch03-tan2-plot.svg"); nothing # hide
```

▶ 符号関数を描く

[Base.sign (https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.sign)](https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.sign)

```
using PyPlot
clf() #hide
using PyPlot
xs=-6.4:0.1:6.4
-6.4:0.1:6.4
plot(xs, sign.(xs), ".")
savefig("sign1-plot.svg"); nothing # hide
```

```
clf() #hide
plot(xs, sign.( cos.(xs)), ".")
savefig("sign2-plot.svg"); nothing # hide
```

```
clf() #hide
plot(xs, sign.( cos.(xs*2/pi)), ".")
savefig("sign3-plot.svg"); nothing # hide
```

▶ 絶対値関数

```
using PyPlot
clf() #hide
xs=-1:0.1:1
plot(xs,abs.(xs))
plt[:axes]()[:set_aspect]("equal")
savefig("abs1-plot.svg"); nothing # hide
```

★ 今回のまとめ

第5回：■ 条件式・■ 条件分岐

■ 論理演算

!
&&
||

■ 数の大小比較

Numeric Comparisons (https://docs.julialang.org/en/v0.6.4/manual/mathematical-operations/#Numeric-Comparisons-1)

演算子==は、a == bのように用いて、値が等しいか否か判定する。
演算子!=は、値が異なるか否か判定する。
成立すれば(真ならば) trueが、
成立しなければ(偽ならば) false が結果となる。

```
1 == 1
1 == 2
1 != 1
1 != 2
```

演算子>は、a > b のように用いて、aの値がbの値よりも大きいか否か。
演算子>=は、a >= b のように用いて、aの値がbの値以上であるか否か判定する。

```
2 > 1
2 >= 1
2 >= 2
```

演算子<は、a < b のように用いて、aの値がbの値よりも小さいか否か。
演算子>=は、a <= b のように用いて、aの値がbの値以下であるか否か判定する。

```
2 < 1
2 <= 1
2 <= 2
```

数の比較演算子は、連続して記述できる。

```
1 < 2 < 3
1 < 2 && 2 < 3
```

a>b, >(a,b)

■ if文

Conditional Evaluation (https://docs.julialang.org/en/v0.6.4/manual/control-flow/#man-conditional-evaluation-1)

if文は、条件式をとる。
条件式の値が true なら、if文の次の文から、end, else, elseif が出現するまでの文を実行する。
条件が成り立つときだけに実行される部分をブロック(block)という。
ブロックは、字下げで表記される。

```
x=1
y=2
if x < y
  println("x は y より小さい")
end
```

else文が続く場合がある。
条件式の値が false なら、else文の次の文から、end が出現するまでの文 (elseブロック)を実行する。

```
x=1
y=2
if x < y
  println("x は y より小さい")
else
  println("x は y より小さくない")
end
```

else文の前に、elseif文文が続く場合がある。
最初の if文の条件式が false なら、
elseif文の条件式を計算し、それが true なら、elseif 文の次の文から、elseifまたはend が出現するまでの文 (elseifブロック)を実行する。

```
x=1
y=2
if x < y
  println("x は y より小さい")
elseif x > y
  println("x は y より大きい")
else
  println("x は y と等しい")
end
```

```
x=40
m=if     x >= 100
  println("x は 100 以上である")
elseif x >= 50
  println("x は 50 以上である")
elseif x >= 20
  println("x は 20 以上である")
else
  println("x は 20 よりも小さい")
end
@show m
```

▪ if式

```
x=40
m=if x >= 100
  "x は 100 以上である"
elseif x >= 50
  "x は 50 以上である"
elseif x >= 20
  "x は 20 以上である"
else
  "x は 20 よりも小さい"
end
@show m
```

▪ 3項演算子

a ? b : c

```
m= 2 > 1 ? "yes" : "no"
@show m
```

▪ 短絡評価

Short-Circuit Evaluation (https://docs.julialang.org/en/v0.6.4/manual/control-flow/#Short-Circuit-Evaluation-1)

▶ 擬似乱数

Random Numbers (https://docs.julialang.org/en/v0.6.4/stdlib/numbers/#Random-Numbers-1)

Base.Random.rand (https://docs.julialang.org/en/v0.6.4/stdlib/numbers/#Base.Random.rand)

rand()
0から1未満の小数の乱数を出す擬似乱数

Base.Random.srand (https://docs.julialang.org/en/v0.6.4/stdlib/numbers/#Base.Random.srand)

Base.Random.srand (https://docs.julialang.org/en/v0.6.4/stdlib/numbers/#Base.Random.srand)

```
rand()
rand()
rand()
```

rand(コレクション)
とすると、コレクションのうち、
一つの要素を選ぶ擬似乱数。

▶ モンテカルロ法

$x^2+y^2 \lt 1$

半円の面積 $\frac{pi}{4}$

```
n=2^10
s=0
for i=1:n
  x=rand()
  y=rand()
  if x*x + y*y <= 1
    s += 1
  end
end
@show s/n; #hide
```

```
using PyPlot
clf() #hide
for m in 1:14
  n=2^m
  s=0
  for i=1:n
    x=rand()
    y=rand()
    if x*x + y*y <= 1
      s += 1
    end
  end
  plot(n, s/n, ".")
end
ylim(0.9*pi/4, 1.1*pi/4)
xscale("log")
axhline(pi/4, color="k", lw=0.5)
savefig("ch05-qc1-plot.svg"); nothing # hide
```

★ 今回のまとめ

第6回：■ 型・■ 整数型

■ 更新演算子

Updating operators (https://docs.julialang.org/en/v0.6.4/manual/mathematical-operations/#Updating-operators-1)

■ 型

```
typeof(1)
typeof(1.0)
```

Float64

Primitive types
Composite Types

▪ 整数

Integers (https://docs.julialang.org/en/v0.6.4/manual/integers-and-floating-point-numbers/#Integers-1)

```
typemax(Int64)
typemin(Int64)
```

Overflow behavior (https://docs.julialang.org/en/v0.6.4/manual/integers-and-floating-point-numbers/#Overflow-behavior-1)

2の補数とは、負の数 $-n$ を $2^{64}-n$ で表す。

```
typemax(Int64)+1
```

▪ 整数を 0 で割る

Division erros (https://docs.julialang.org/en/v0.6.4/manual/integers-and-floating-point-numbers/#Division-errors-1)

▪ 整数と浮動小数点数の演算

```
1 / 2
1.0 / 2.0
```

どちらも浮動小数点数

▶ 浮動小数点数から整数への変換

```
Int64(1.0)
Int64(1.1)  # エラー
Int64(floor(1.1))
```

▶ 床関数・天井関数

[Base.ceil]
(https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.ceil)
Base.floor (https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.floor))

```
using PyPlot
clf() #hide
xs=-2.4:0.1:2.4
plot(xs, ceil.(xs), "o", label="ceil")
plot(xs, floor.(xs), ".", label="floor")
for x in -2:2
  axvline(x, color="k", lw=0.5)
end
legend()
plt[:axes]()[:set_aspect]("equal")
savefig("floorceil1-plot.svg"); nothing # hide
```

▶ ユークリッドの互除法

2 つの自然数 a, b (a ≧ b) について、a の b による剰余を r とすると、 a と b との最大公約数は b と r との最大公約数に等しいという性質が成り立つ。この性質を利用して、 b を r で割った剰余、 除数 r をその剰余で割った剰余、と剰余を求める計算を逐次繰り返すと、剰余が 0 になった時の除数が a と b との最大公約数となる。

Base.rem (https://docs.julialang.org/en/v0.6.4/stdlib/math/#Base.rem)

```
a=1071
b=1029
@show a,b
while b != 0
  t = a
  b = rem(a, b)
  a = t
  @show a,b
end
@show a,b
```

```
a=3355
b=2379
@show a,b
while b != 0
  t = a
  b = rem(a, b)
  a = t
  @show a,b
end
@show a,b
```

▶ 商・剰余関数

```
using PyPlot
clf() #hide
xs=-6.8:0.2:6.8
d=3
plot(xs,fld.(xs,d), ".", label="fld")
plot(xs,mod.(xs,d), ".", label="mod")
legend()
xlim(-6.2,6.2 )
legend()
plt[:axes]()[:set_aspect]("equal")
savefig("fldmod1-plot.svg"); nothing # hide
```

同じことは床関数でも書ける。

```
using PyPlot
clf() #hide
xs=-6.8:0.2:6.8
d=3
qs=floor.(xs/d)
rs=xs-qs*d
plot(xs, qs, ".", label="quotient")
plot(xs, rs,".", label="remainder")
for x in -9:d:9
axvline(x, color="k", lw=0.5)
end
xlim(-6.2,6.2 )
legend()
plt[:axes]()[:set_aspect]("equal")
savefig("floorceil2-plot.svg"); nothing # hide
```

◀ 切り捨てるには？

- 10の位で
- 100の位で
- 1000の位で
- 0.1の位で
- 一般に10^{n}の位で

◀ 四捨五入するには？

- 1の位で
- 10の位で
- 100の位で
- 1000の位で
- 0.1の位で
- 一般に10^{n}の位で

★ 今回のまとめ

第7回：▪ 浮動小数点数

▪ 浮動小数点数

正規化数、副正規化数

浮動小数とは、0.12 の代わりに 1.2×10^{-1} のように表示することである。

10進数の浮動小数は

$\pm{\left(d_0.d_1d_2\cdots \right)}_{10}\times 10^{e}$

のように表される。$\times$の前までの${\left(d_0.d_1d_2\cdots \right)}*{10}$
の部分は仮数部と呼ばれる。添字の10は10進数を意味し、$d_0, d{1}, \cdots $ は $0,1,\ldots,9$ までの数字である。$\times$ の後ろの 10^{e} は指数部と呼ばれる。

2進数の浮動小数は

$\pm{\left(b_0.b_1b_2\cdots \right)}_{2}\times 2^{e}$

のように表される。
ここで、$\times$の前までの
${\left(b_0.b_1b_2\cdots \right)}*{2}$
の部分は仮数部と呼ばれる。添字の2は2進数を意味し、$b_0, b{1}, \cdots $ は 0 または 1 の数字である。$\times$ の後ろの 2^{e} は指数部と呼ばれる。

bit (binary digit)とは、2進数の一桁のことである。

本文で用いる浮動小数点数は Float64 型である。

```
typeof(1.0)
```

Float64 型は、「IEEE754標準倍精度浮動小数点数」に基づき、
符号部 1 bit、
指数部 11 bit
仮数部 53 bit から構成される。
ただし、以下のように先頭の 1 bitを固定し、仮数部の 52 bit のみをデータとして採用するため、
2進数の並びは 1+11+52 = 64 bit である。

Float64は、正規化数、副正規化数、数でない数の３種類からなりたっている。

正規化数は、$b_{0} = 1$として、
$\pm{\left(1.b_1b_2\cdots b_{52}\right)}*{2}|times 2^{e}$
*のように表すものである。
ただし、指数は $-1022 |le e |le 1023$ の範囲である。
仮数 ${|left(1.b_1b_2|cdots b*{52}\right)}_2$ は
1以上で2を超えない範囲の小数となる。

正規化数で表すことができない、絶対値が小さい浮動小数は副正規化数で表わされる。

副正規化数は、$b_{0} = 0$, $e=-1023$ として、
$\pm{\left(0.b_1b_2\cdots b_{52}\right)}*{2}|times 2^{e}$
*のように表すものである。
仮数部 ${|left(1.b_1b_2|cdots b*{52}\right)}_{2}$ は
0以上で1を超えない範囲の小数となる。

「数でない数」はあとに述べる。

Float64で表すことができる、絶対値が最も大きい数は、
正規化数の $2^{1024}≒1.798\times10^{308}$ である。
絶対値が最も小さい数は副正規化数の $2^{-1022}≒2.225\times10^{-308}$ である。

これらは、realmax, realmin関数で得られる。

```
realmax(Float64)
realmin(Float64)
```

丸め

小数 0.2 は
$0.2 = \frac{1}{5} = \frac{1}{{101}*{2}}$
となるが、1 を ${101}*{2}$ で割り切ることはできない。0.2 を2進数で表すと

${0.00110011001100\cdots}_{2}$

のようになる。すなわち、1100 の並びが無限に続く循環小数となる。

また、小数 0.1 は
$0.1 = \frac{1}{5\times 2} = \frac{1}{{101}_{2}} \times 2^{-1}$ であるから、0.1 を2進数で表すと
（上を1桁ずらして）

${0.000110011001100\cdots}_{2}$

のようになる。これも、1100 の並びが無限に続く循環小数となる。

「循環小数」は「有限桁の小数」では表すことができないが、
Float64型で表現するため、仮数の下位の桁を修正する操作を行う場合がある。
この「仮数の下位の桁を修正する」ことを「丸める」という。

「丸め」られた浮動小数の計算は、筆算とは違う結果となる場合がある。
例えば、

```
0.1+0.2
0.1+0.2 == 0.3
```

筆算の結果は 0.3であるが、
計算結果は `0.30000000000000004` と異なってしまう。

別の例として、0.1を 10回足した結果は

```
s=0
for i in 1:10
  s+= 0.1
end
@show s
s == 1.0
```

0.9999999999999999
となり、1.0 にはならない。

このような、「丸め」が原因の、正しい値からの「ずれ」を「丸め誤差」と呼んでいる。

2次方程式

$x^2-bx+c=0$
解の公式
$x_1=\frac{b-\sqrt{b^2-4c}}{2}$
$x_2=\frac{b+\sqrt{b^2-4c}}{2}$

x_2は、桁落ちしやすい。
そこで、
$(b-\sqrt{b^2-4c})$ を分母分子に掛けて

$x_{21} = \frac{2c}{b+\sqrt{b^2-4c}}=\frac{c}{x_1}$$

解と係数の関係 $x_1x_2=c$ である。

▶ 小数を2進数へ変換する

1100 が循環する。

```
x=0.2
for i=1:50
  q=floor(x/2)
  print(Int64(q))
  x -= q*2
  x *= 2
end
```

00000110011001100110011001100110011001100110011001

1100 循環小数

```
x=0.2
for i=1:50
  q=floor(x/2)
  print(Int64(q))
  x -= q*2
  x *= 2
end
```

00001100110011001100110011001100110011001100110011

桁落ち、情報落ち

桁落ち、情報落ち

1.7976931348623157e308
加算

結合則を満たさない

```
x= 1e20
y=-1e20
z= 1.0
(x + y) + z
x + (y + z)
```

0.0
丸め誤差

isapprox

数でない数

Numeric Comparisons (https://docs.julialang.org/en/v0.6.4/manual/mathematical-operations/#Numeric-Comparisons-1)

数でない数か、確かめる。
Base.isfinite (https://docs.julialang.org/en/v0.6.4/stdlib/numbers/#Base.isfinite)

Base.isinf (https://docs.julialang.org/en/v0.6.4/stdlib/numbers/#Base.isinf)

Base.isnan (https://docs.julialang.org/en/v0.6.4/stdlib/numbers/#Base.isnan)

▶ 数値微分

```
logspace(1,2,5)
```

$\frac{df(x_0)}{dx} = \lim_{h \longrightarrow 0}\frac{f(x_0+h)-f(x_0)}{h}$

関数 $y=x$ の $x=1$における微分係数を、
上の定義により求めよう。
求まるべき値は 1 であるが、h を小さくすると 1 の上下に暴れてしまう。

```
using PyPlot
clf() #hide
h=logspace(-18,-8,100)
d=( (1+h).^2 - 1) ./ h
plot(h,d, ".")
ylim(5e-1,3e0)
yscale("log")
xscale("log")
savefig("ch07-df1-plot.svg"); nothing # hide
```

関数 $y=x^n$, ($n=1,2,3$) の $x=1$における微分係数を、
上の定義により求めよう。
求まるべき値は n であるが、h を小さくすると n の上下に暴れてしまう。

```
using PyPlot
clf() #hide
h=logspace(-18,-8,100)
for n=1:3
    d=( (1+h).^n - 1) ./ h
    plot(h,d, ".", label="y=x^"*string(n))
end
yscale("log")
xscale("log")
legend()
savefig("ch07-df2-plot.svg"); nothing # hide
```

★今回のまとめ