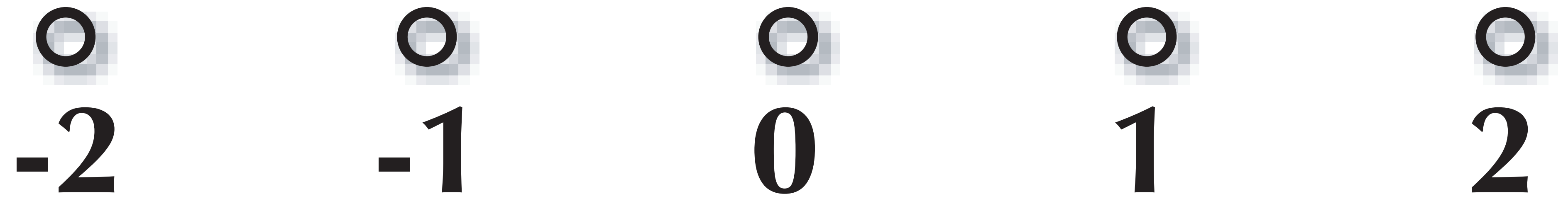
オブジェクト指向
プログラミング

第８回

箕原辰夫

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）$^{\text{指数}}$
- 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
    - 実数が式の中に出てくると実数へ自動的に変換
    - 実数演算（/…実数除算など）が出てくると実数へ自動的に変換
  - 実数から複素数へ
    - 複素数が式の中に出てくると複素数へ自動的に変換

- 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) …実数型へ変換
  - complex(値) … 複素数型へ変換
  - str(値) …文字列型へ変換
- 実数に変換したい場合
  - float(7) ⇒ 7.0
  - float("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
  - 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
  - *epsilon* = 0.0001 #誤差許容範囲
  - **if** 0.1-*epsilon* < x and x < 0.1+ *epsilon*:

mathモジュールの定数

- **import** mathが必要
- math.e…自然対数の底（ネイピア数）
- math.pi…円周率
- math.inf…正の無限大（負の無限大は、-math.infを用いる）
- $e = \lim_{n \to \infty} (1+1/n)^n$
- math.powを使って精度を上げながら求める

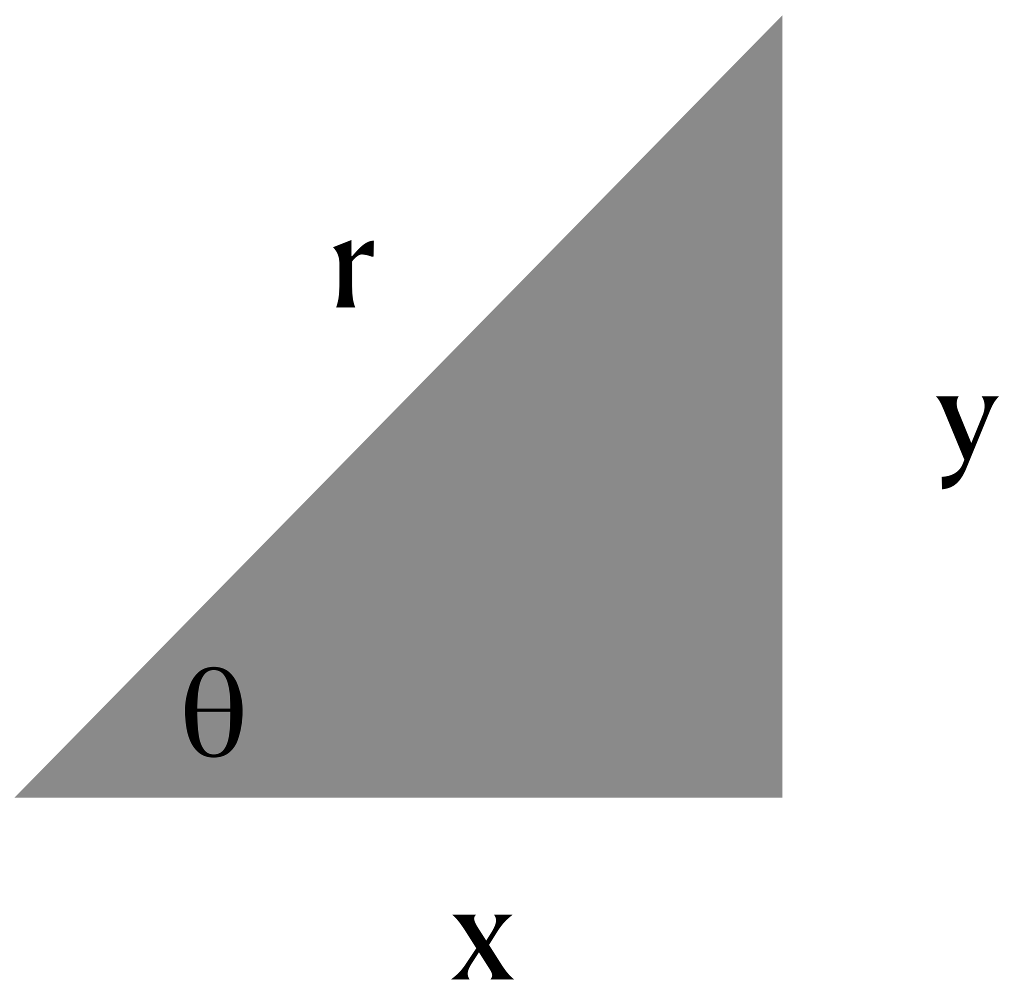
```
n = 1
while n <= 10000000000000:
    e = math.pow( 1+1.0/n, n )
    n *= 10
```

整数との変換

- math.ceil(x)… $\lceil x \rceil$　天井関数：大きいか等しい一番小さな整数
  - 計算結果は整数になる
- math.floor(x)… $\lfloor x \rfloor$　床関数：小さいか等しい一番大きな整数
  - 計算結果は整数になる
- round(x)…統計的四捨五入（ただし、偶数に丸める）、結果は整数になる
- round(x, n)…小数点以下n+1桁で四捨五入
- math.trunc(x)…小数部を切り捨てる、int(x)と等しい

三角関数

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



- math.acos(x / r) → θ (r=1,0〜π)
- math.asin(y / r) → θ (r=1,-π/2〜π/2)
- math.atan(y/x) → θ (-π/2〜π/2)
- math.atan2(y, x) → θ (πを超えるとマイナスで出てくる)
- math.hypot(x, y) → r

Radian体系とDegree体系、分秒

- Radianの角度 = math.radians(Degreeの角度)
- Degreeの角度 = math.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

対数

- $x = \mathrm{a}^p$　aは基数　pは指数　$64 = 8^2$　math.pow(8, 2)
- $p = \log_{\mathrm{a}} x$　xの対数　$2 = \log_8 64$　math.log8(64)　# log8は用意されていない

- 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
  - math.floor(math.log10(value) +1)　# valueの桁数を求める

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

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

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

その他の関数

- math.sqrt(x)
  - xの平方根を求める　　math.pow(x, 0.5)
- abs(x)
  - xの絶対値を求める　　abs(-9) ⇒ 9
- math.fabs(x)
  - xの絶対値を求める　　math.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
- math.gcd(a, b)
  - aとbの最大公約数を求める

Python 3.5以降に加わった関数

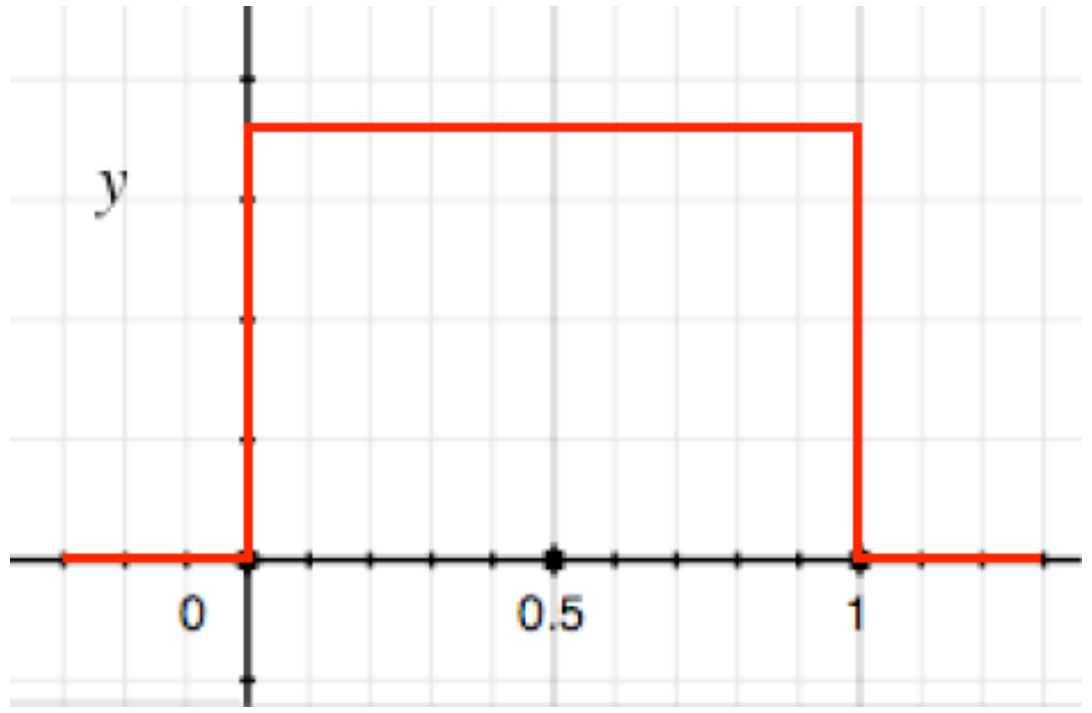
- isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0) … 近似的に等しいかどうか判定する 3.5より
- comb(n, k) … 組み合わせ数を求める 3.8より
- perm(n, k) … 順列数を求める 3.8より
- remainder(x, y) … 精度の高い剰余 3.7より
- dist(p, q) … p=(x, y)とq=(x, y)の間の距離を求める 3.8より
- tau … τ定数（2π）6.283185307179586 3.6より

random関数

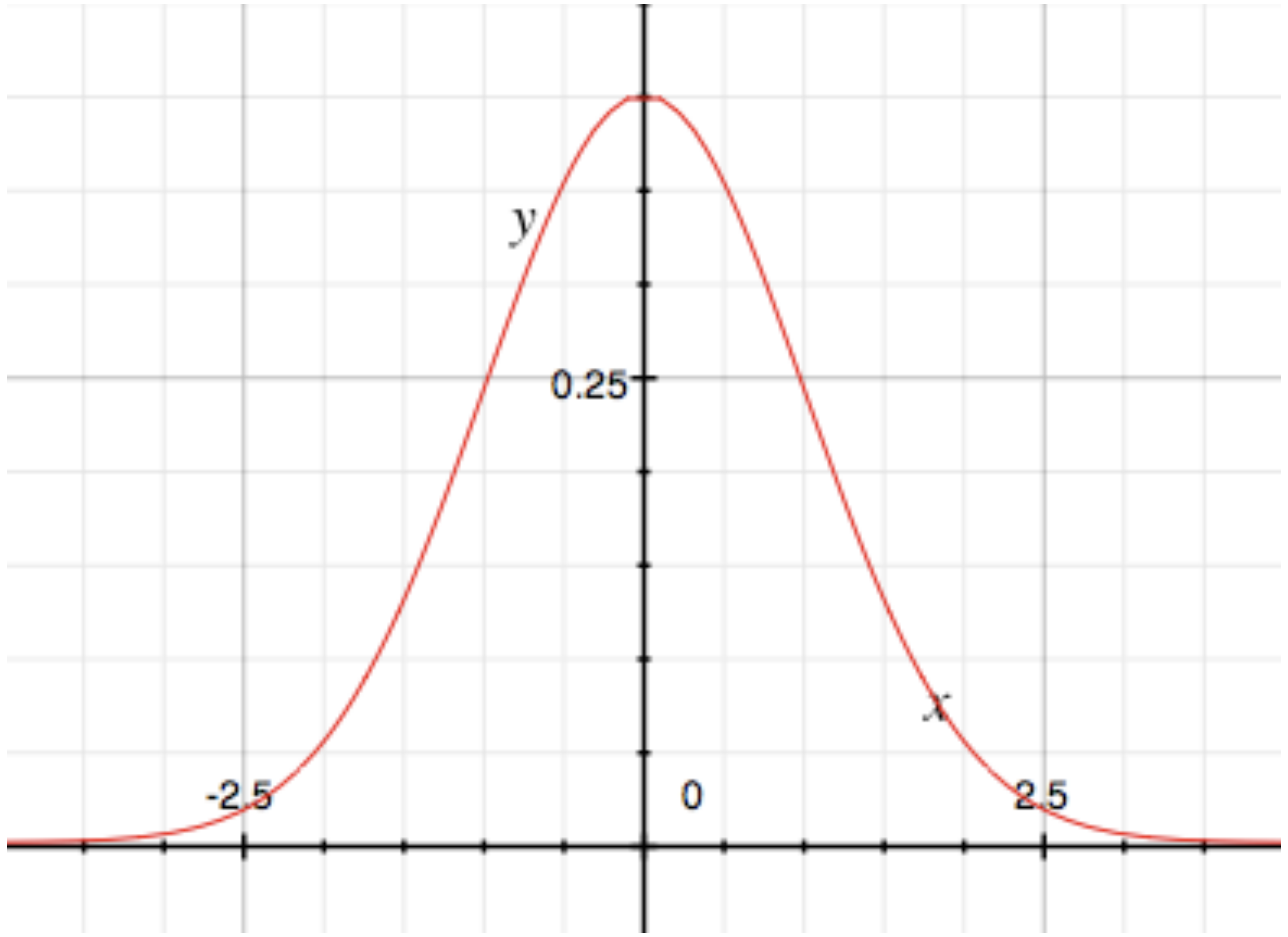
- **import** randomが必要
- random.random( )
  - 0から1未満の乱数（任意の実数）を返す
- random.uniform(a, b)
  - a以上b以下の乱数（任意の実数）を返す
- random.randint(m, n)
  - m以上n以下の乱数（整数）を返す
- random.gauss(m, sigma)
  - gauss分布（正規分布）で平均がm、標準偏差がsigmaの乱数を返す

正規分布の乱数

- 一様乱数



- 正規分布の乱数



乱数

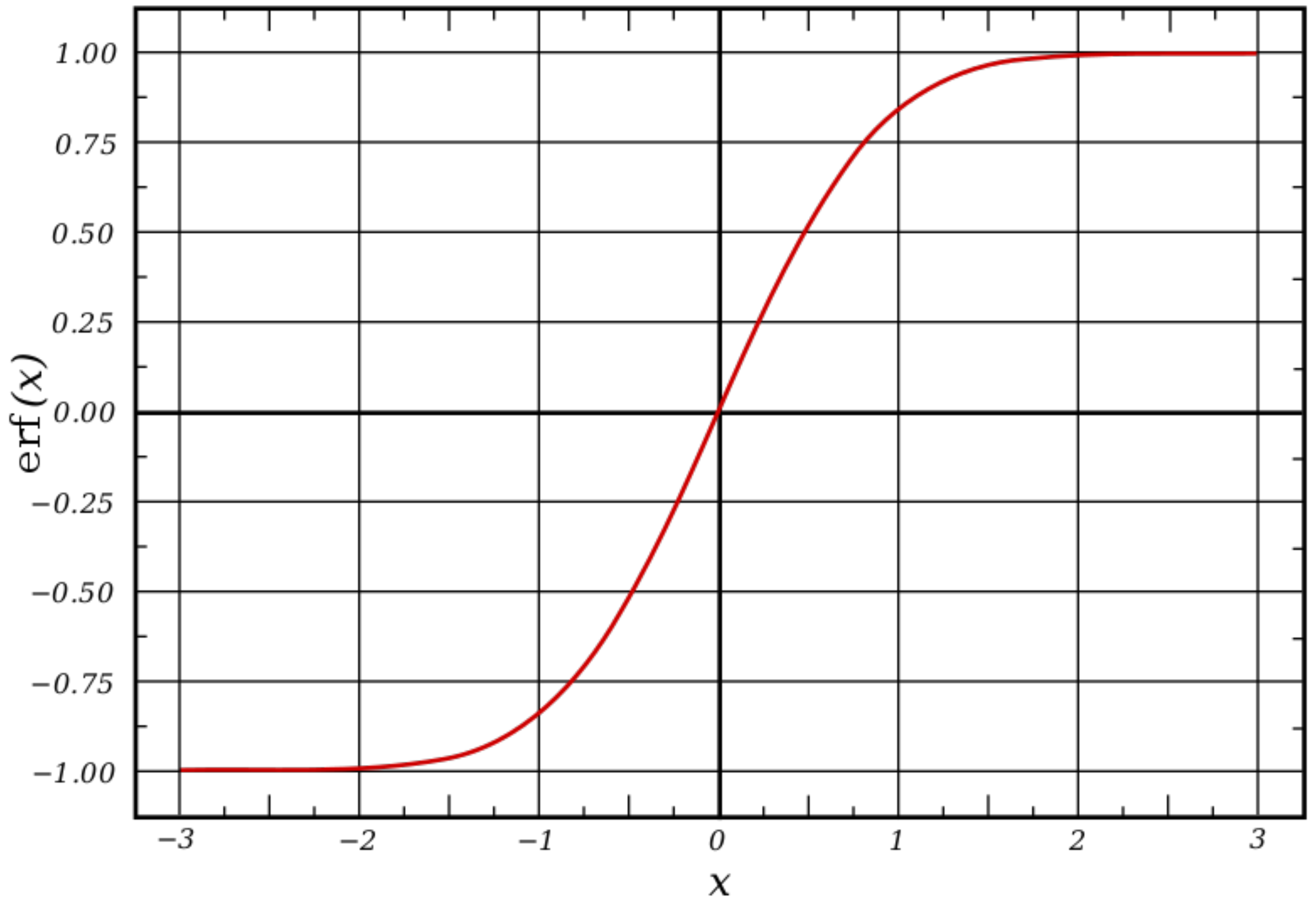
- nからmまでの値の整数の乱数を作りたい場合
- diff = m - n + 1;
  - int(random.random() * diff) + n
- 0から100までの成績を乱数で作る
  - int(random.random() * 101) + 0;
- -10から10までの整数の乱数を作る
  - int(random.random() * 21) - 10

ランダムウォーク

- 現在の位置を憶えておく　cx, cy
- 行く方向を乱数で決める　dir　上下左右
- 行く歩数を乱数で決める　step
- 行った先の位置を x, y とする
- cx, cyからx, yに線を引く
- x, y に cx, cyを代入して繰り返す

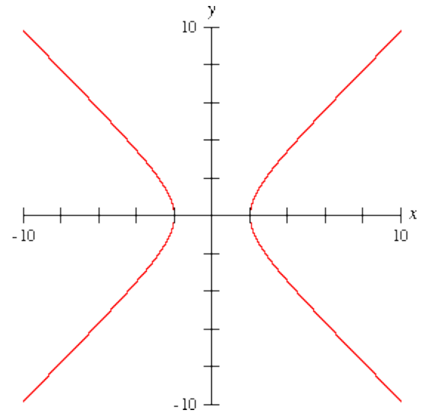
mathモジュールの特殊関数

- 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

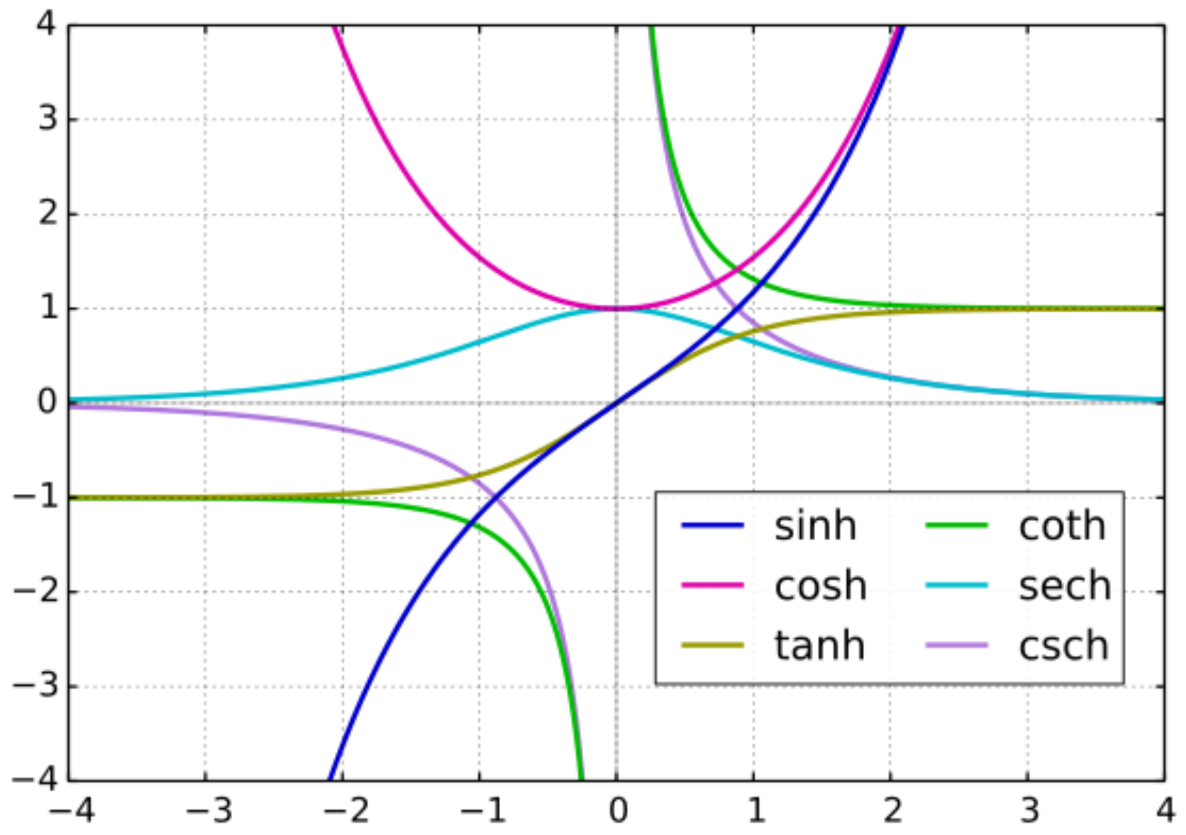


mathモジュールの双曲線関数

- 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

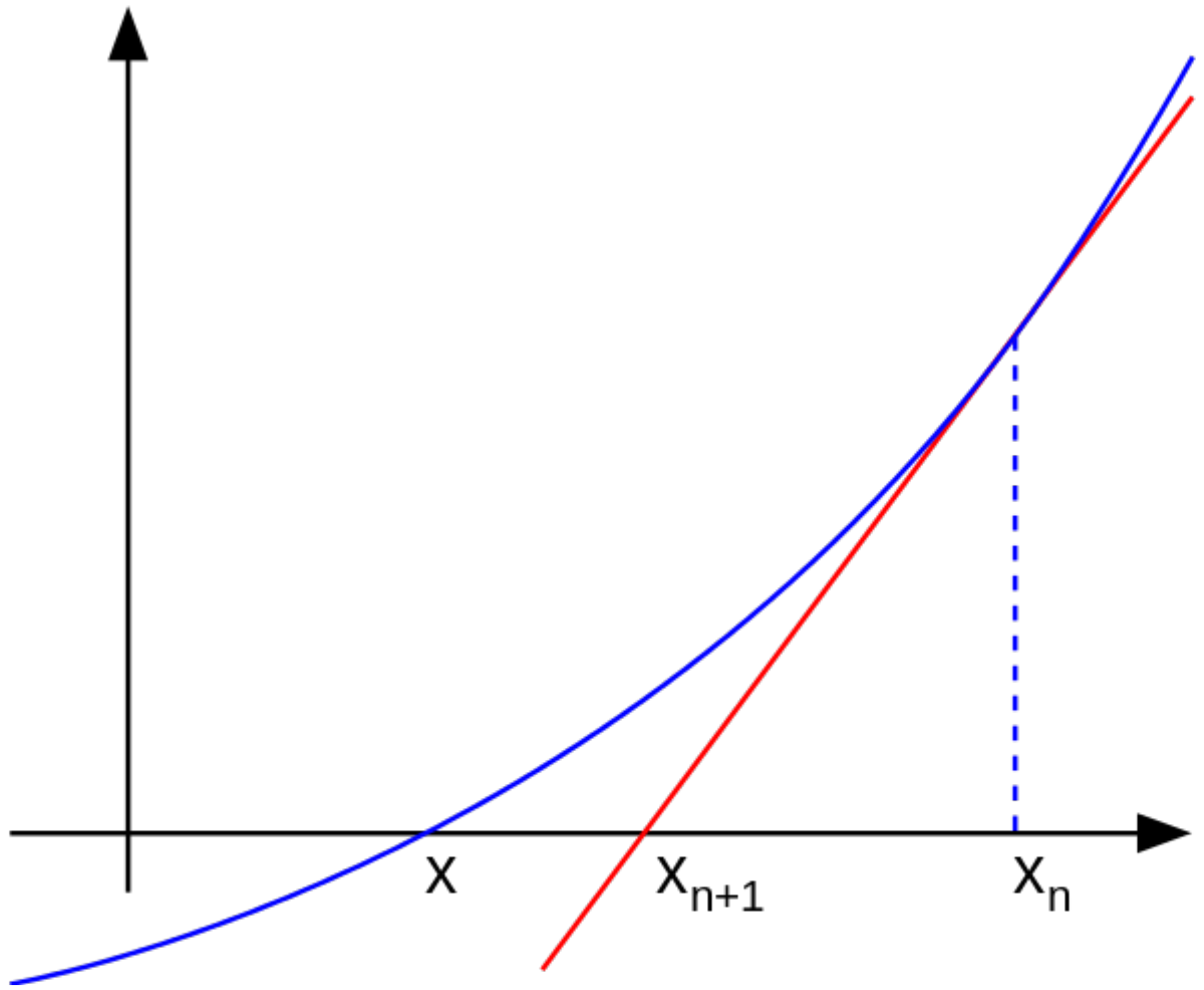


ニュートン法

$$f(x)=0$$

- となるxの値を、以下の数式によって漸次的に求める方法

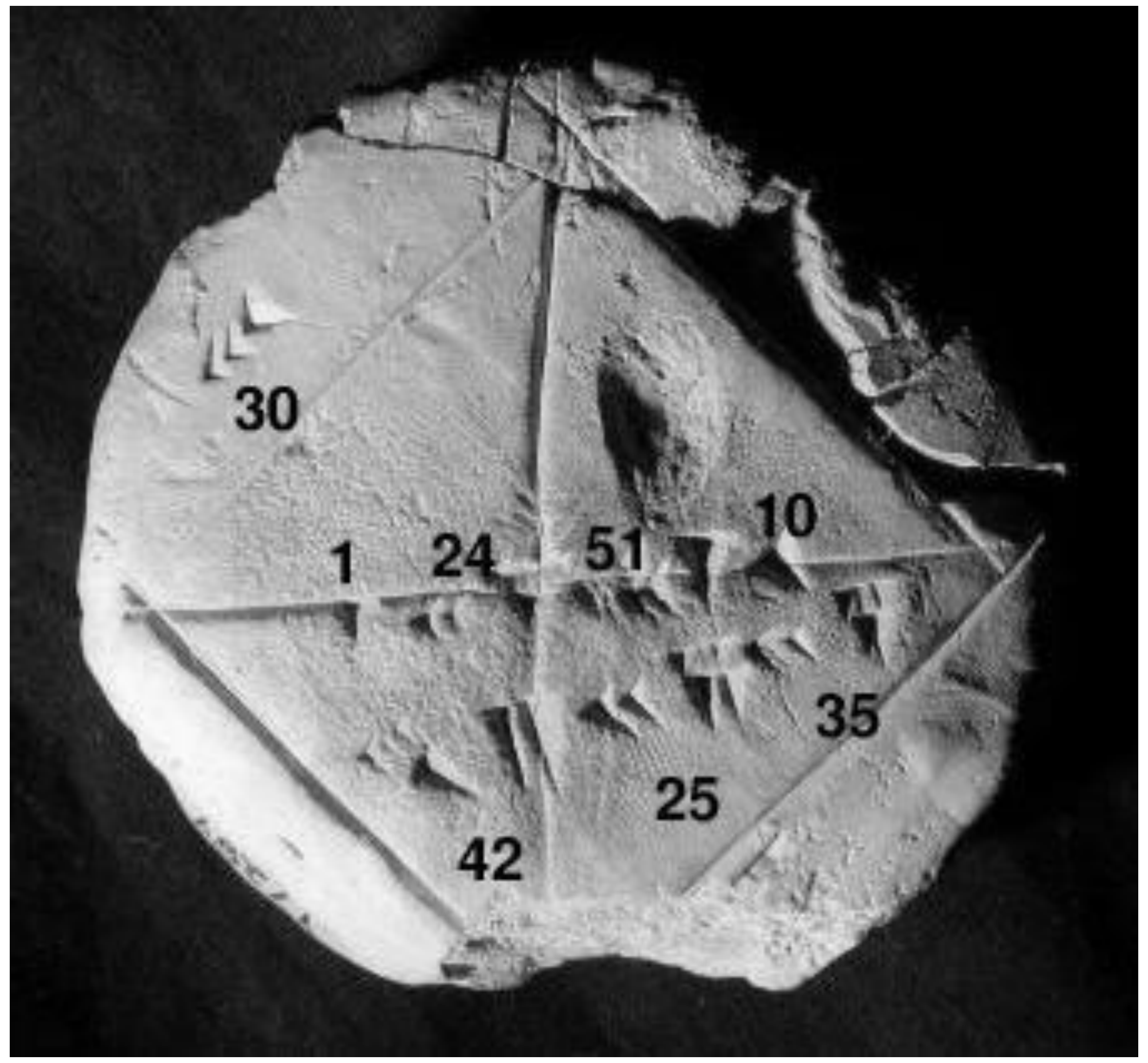
$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$



- x_0は、どのような値でも良い。ただし解が複数あるものは、最初の値によって、どの解が求められるか決定する。

平方根の求め方

- ニュートン法でも求める
  - $x_{n+1} = x_n - f(x_n) / f'(x_n)$
- 平方根の場合
  - $f(x) = x^2 - n$
  - $f'(x) = 2x$
  - $x_{n+1} = x_n - (x_n^2 - n) / 2x_n$
  - $\quad\quad = x_n / 2 + n / 2x_n$



バビロニアの粘土板 YBC 7289 （紀元前1800-1600年頃）
2の平方根の近似値は60進法で4桁、10進法では約6桁に相当する。1 + 24/60 + 51/602 + 10/603 = 1.41421296…。(Image by Bill Casselman)

テイラー展開

- 三角関数のテイラ $\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad \text{for all } x$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad \text{for all } x$$

- 有限回の計算で求める

summm, *factorial* =0.0, 1
for *n* **in** range(100) :
  summ += (1 **if** *n*%2 == 0 **else** -1) / *factorial* * math.pow(*x*, 2*n+1)
  for *f* **in** range(2*(*n*+1)+1, 2*n+1, -1): *factorial* *= *f*

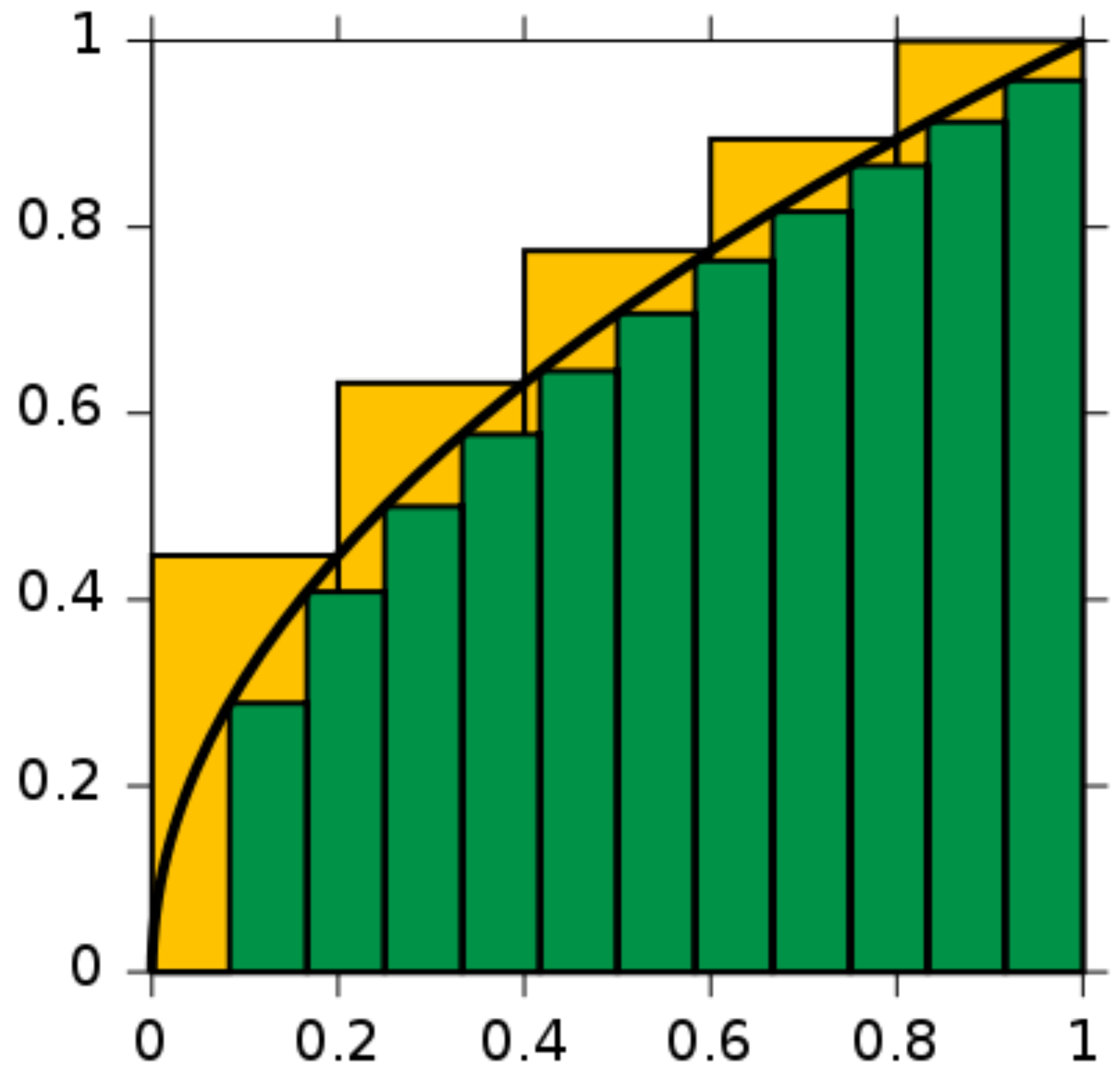
階乗

- $n\,! = 1 \times 2 \times 3 \times \ldots \times n$
- $\Sigma_{x=1}{}^{n}\, x = 1 + 2 + 3 + \ldots + n$
- $\Pi_{x=1}{}^{n}\, x = 1 \times 2 \times 3 \times \ldots \times n$

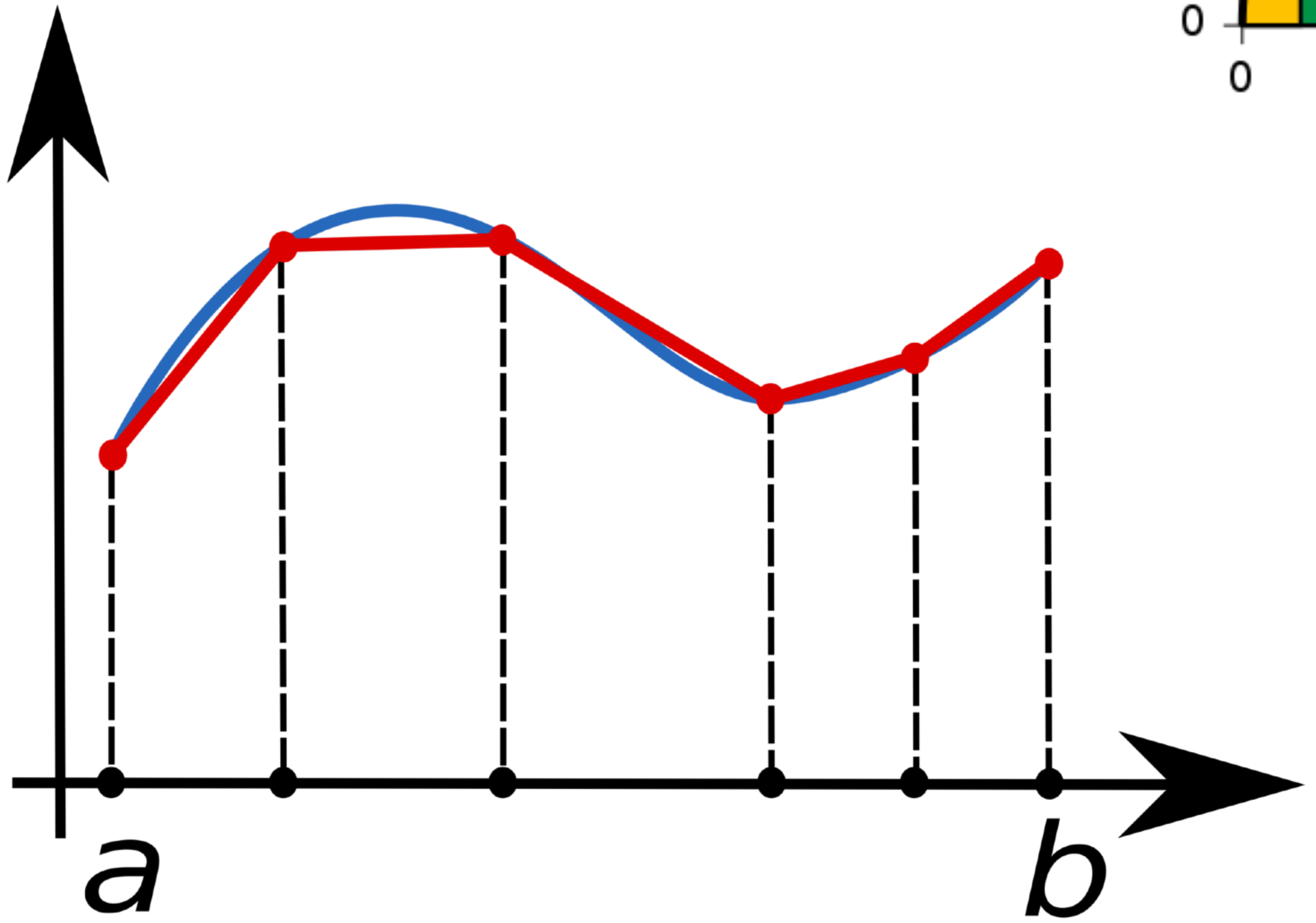
- $\sin x = x - 1/3!\, x^3 + 1/5!\, x^5 - 1/7!\, x^7$

数値積分

- 四角形で近似する

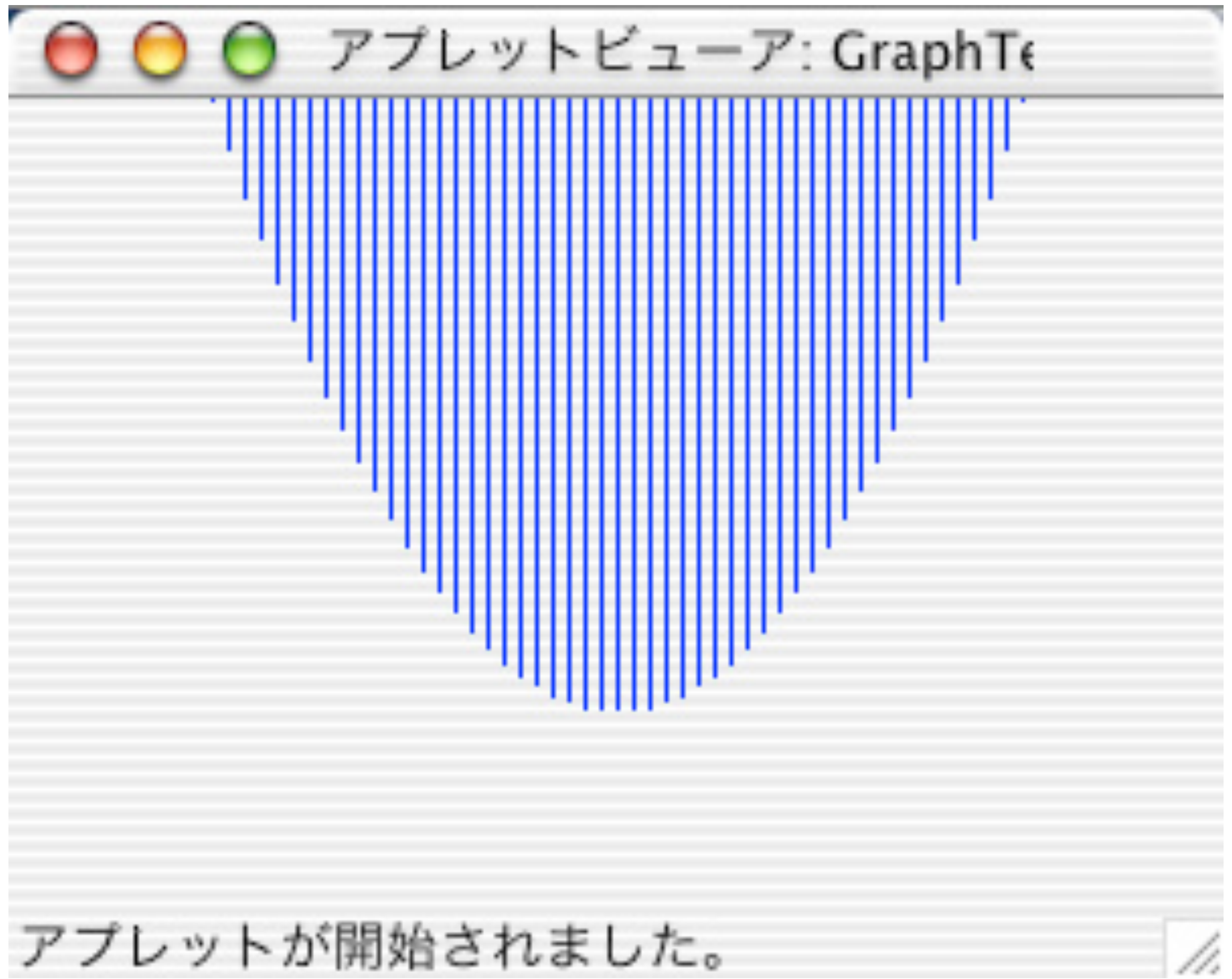


- 台形で近似する



グラフを描く

- 最大値と最小値の位置がどれくらいになるか



拡大・縮小率

- $y = f(x)$のときの計算式
- xの範囲を考える
  - xが動く範囲（定義域）：-100〜100
  - x^3の値の範囲（値域）:-1000000〜1000000
- 拡大・縮小率は、最大値のときの「定義域/値域」になる。
  - 100/1000000 = 0.0001

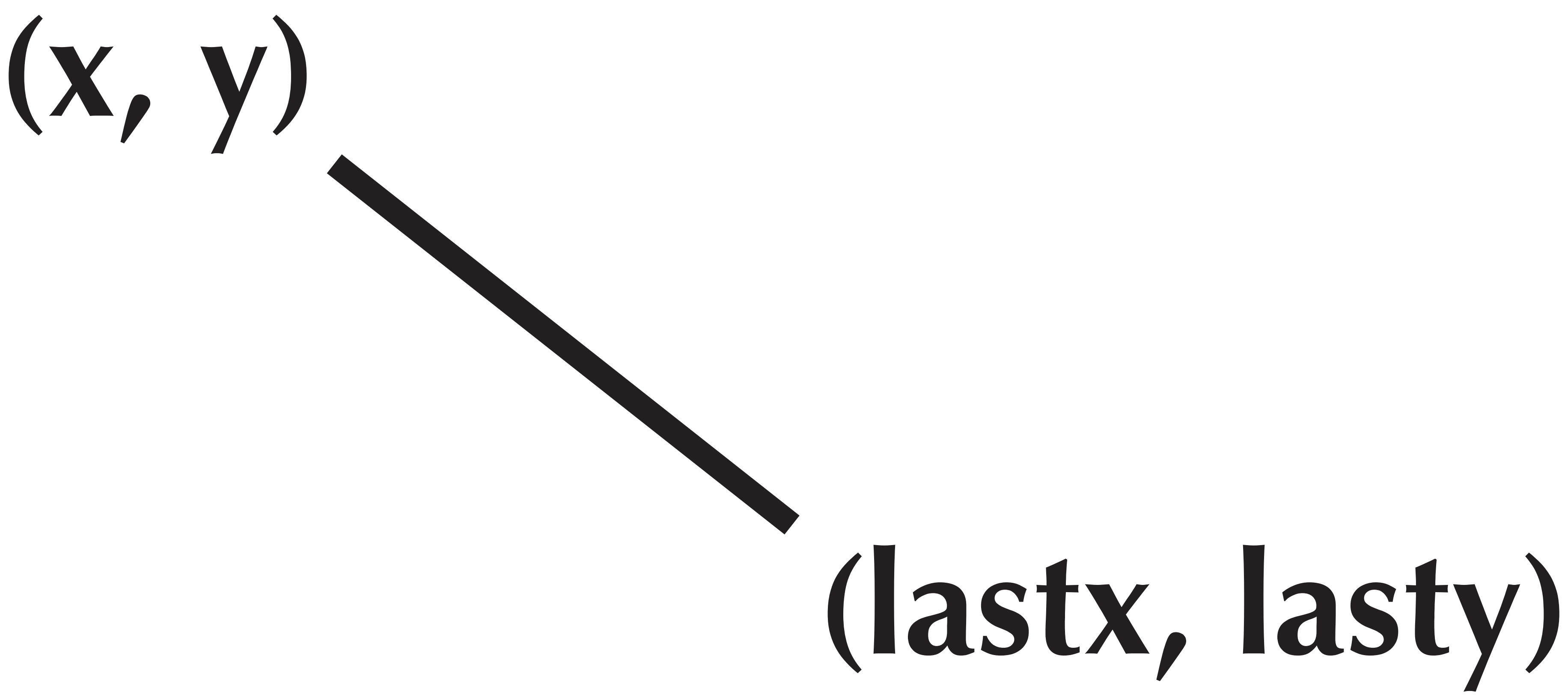
ウィンドウの幅と高さ

- 幅→winfo_width( )で求めることができる
  - *centerx* = win.winfo_width( )/2
- 高さ→winfo_height( )で求めることができる
  - *centery* = win.winfo_height( )/2

- ウィンドウの幅に合せたいとき
  - x方向の拡大率　*centerx* / 最大のx値
  - y方向の拡大率　*cetenry* / 最大のy値

折れ線での近似

- 折れ線近似アルゴリズム
- 前に計算した座標を覚えておく



曲線の表現形式

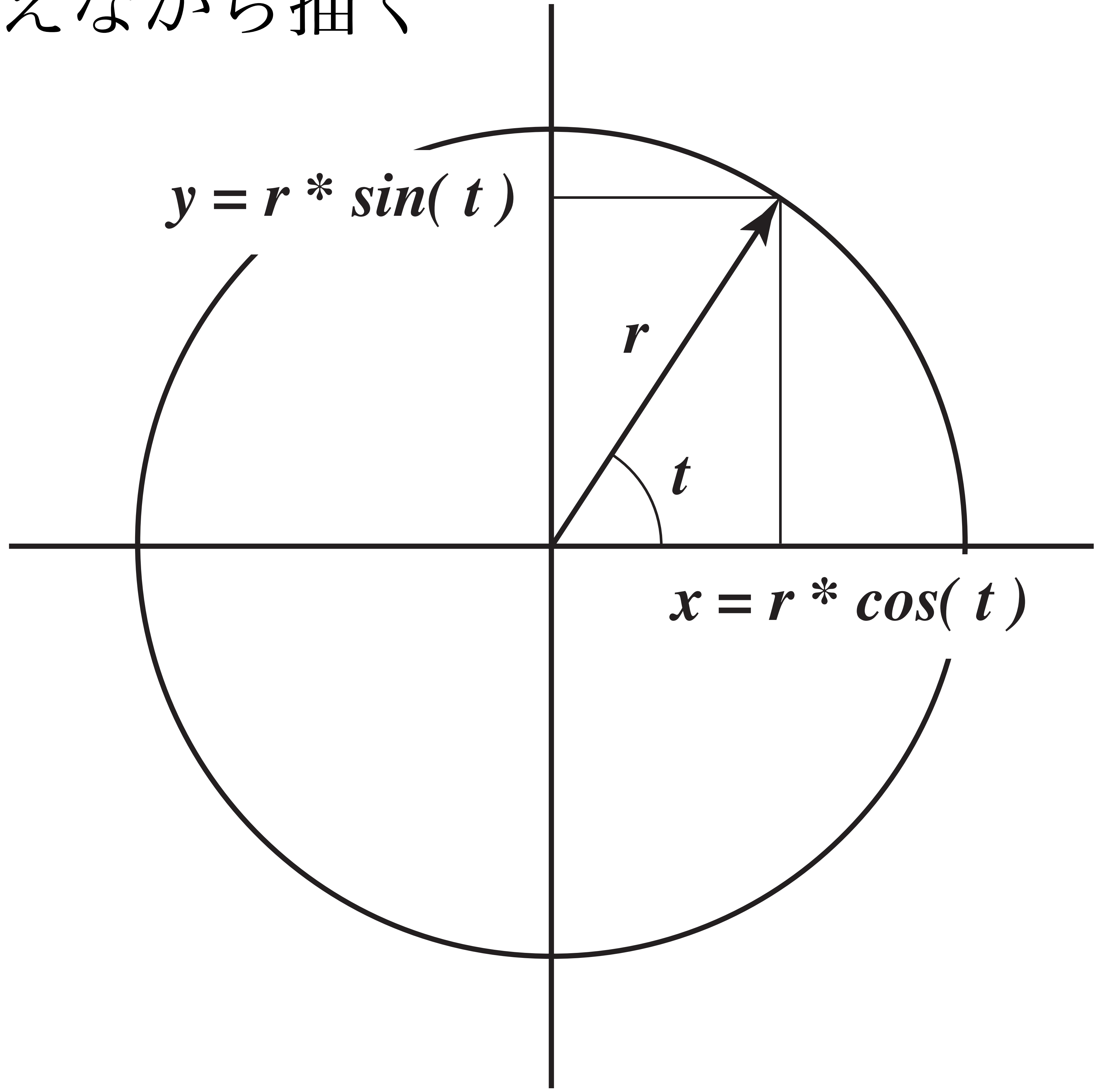
- 陽関数形式
  - $y = f(x)$
  - 例: $y = x^2 + 4x + 3$
- 陰関数形式
  - $f(x, y) = 0$
  - 例：$x^2 + y^2 = r^2$
- パラメトリック形式
  - 媒介変数tを使う（0〜1あるいは0〜2π）
  - $x = f(t)$,　$y = g(t)$
  - 例：$x = r\cos(t)$, $y = r\sin(t)$

パラメトリック曲線の描画

- アルゴリズム
- 媒介変数tを使って、x座標とy座標を求める
- deltaは、１回あたりの進み具合
- n回座標を求めて、線を引く
- 繰返しを始める前に t = 0のところのx, y座標を求めておく
  lastx, lastyに入れておく
- 毎回tをdelta分だけ進めて、x, y座標を求める
- 線をlastx, lastyからx, yに引く
- lastx, lasty = x, y (次回の繰返しに備える)

円の描画

- 三角関数で角度を変えながら描く



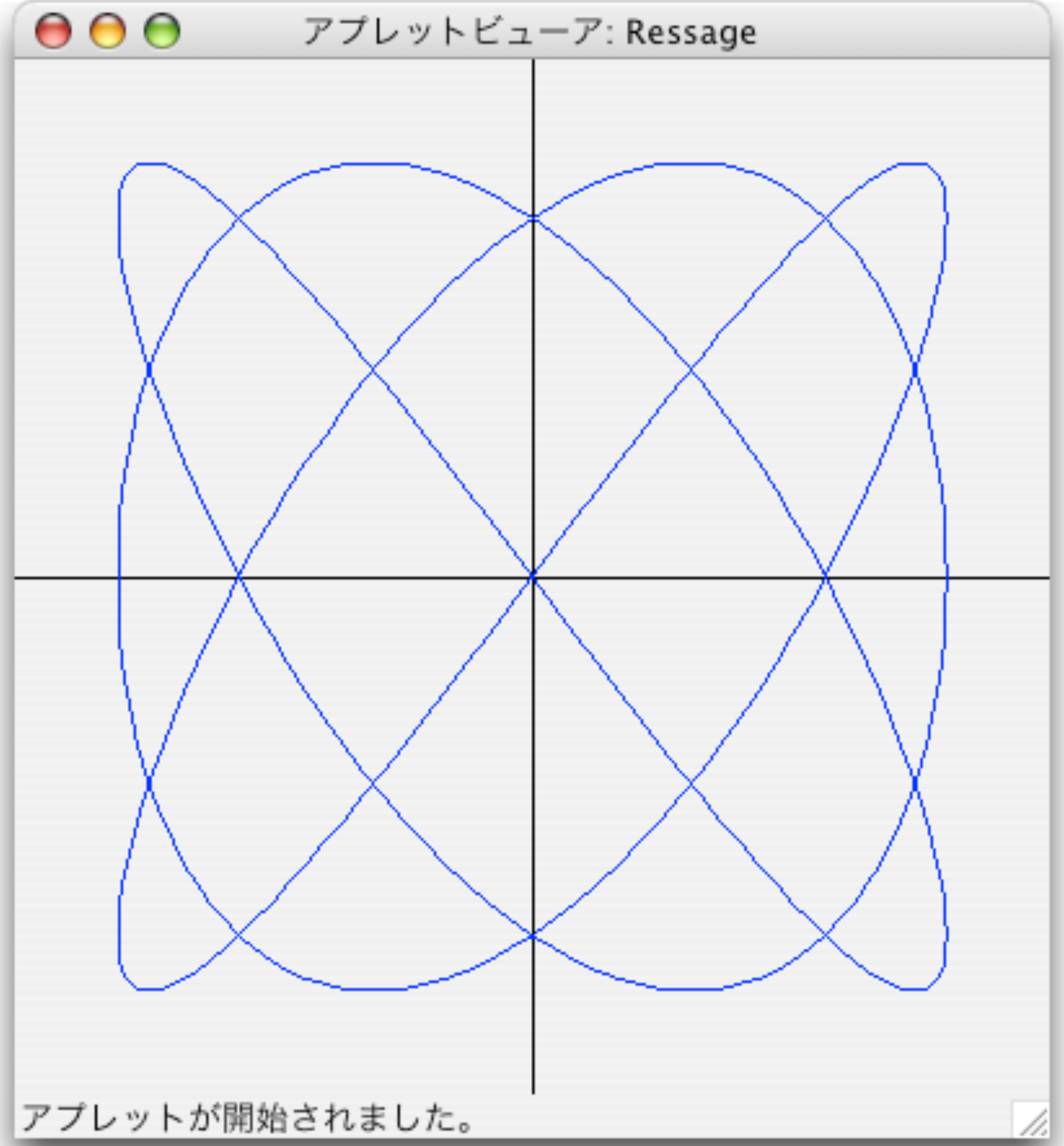
正n角形

- 正n角形と正円
  - 正n角形→(n→∞)→正円

- 正n角形と円周
  - 正n角形の一辺の長さ a
  - 半径 rとする
  - $\lim_{n\to\infty} a * n = 2\pi r$

リサージュ図形

- 1周するだけでなく、sinとcosの角度の変化比率を変えて、何周もさせる

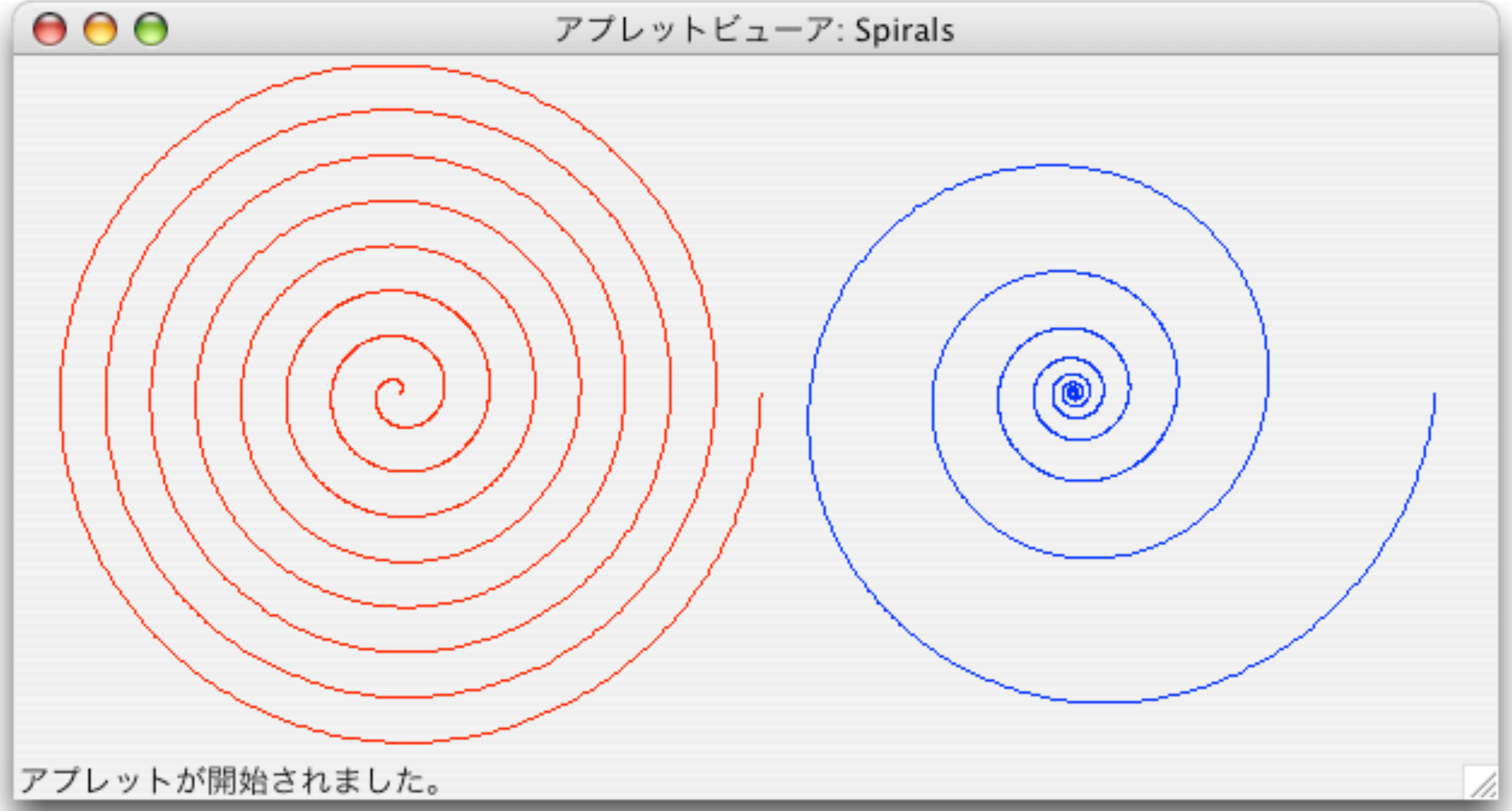


リサージュでのxとｙの動き

- 正円の場合
  - ｘの動き：１周している
  - ｙの動き：１周している
- リサージュの場合
  - xの動き：ｎ回振動している
  - yの動き：m回振動している
- オシロスコープでは、２つの入力の周期（周波数）の比率がリサージュ図形で表現

螺旋

- 角度を変えるときに、半径も変えてゆく

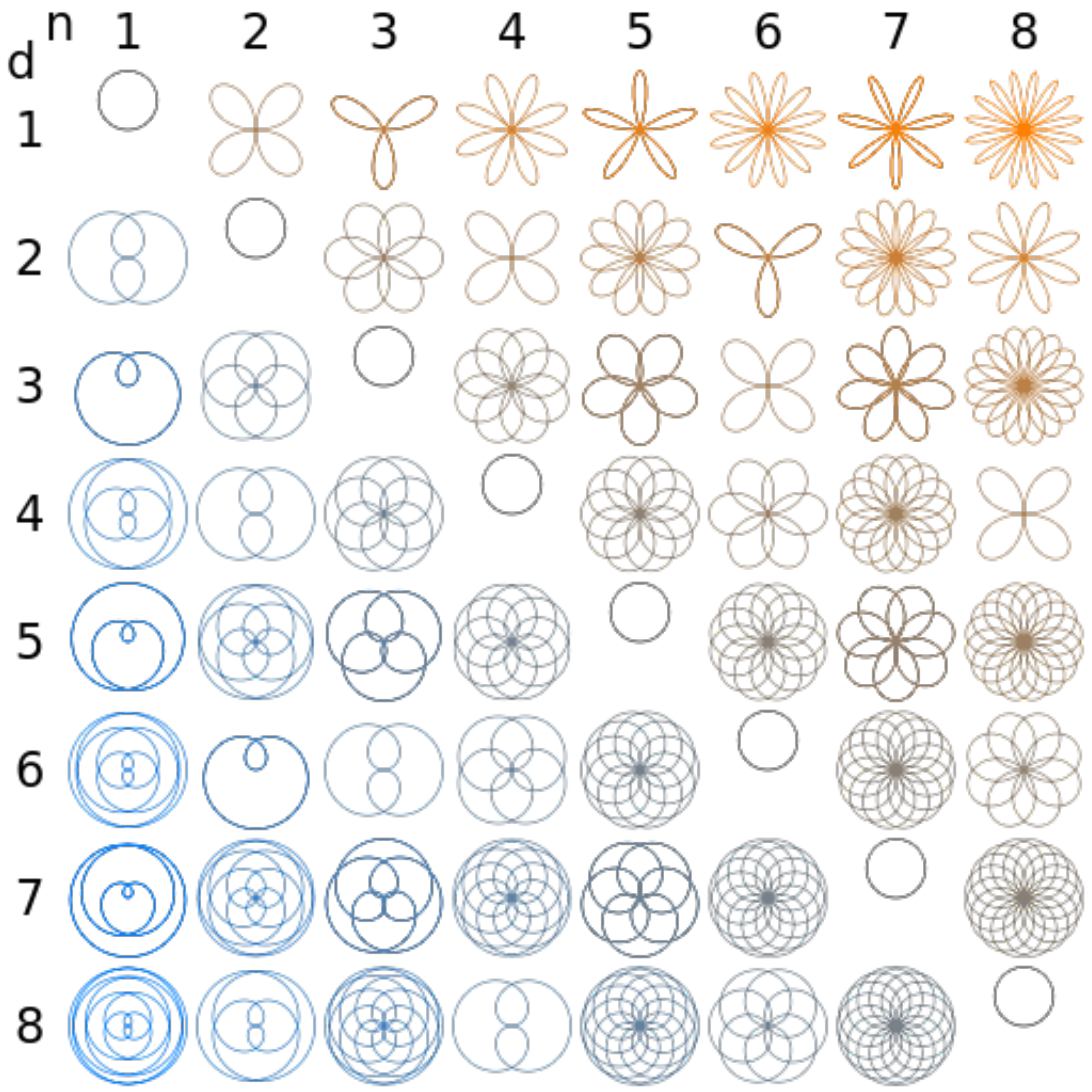


係数の求め方

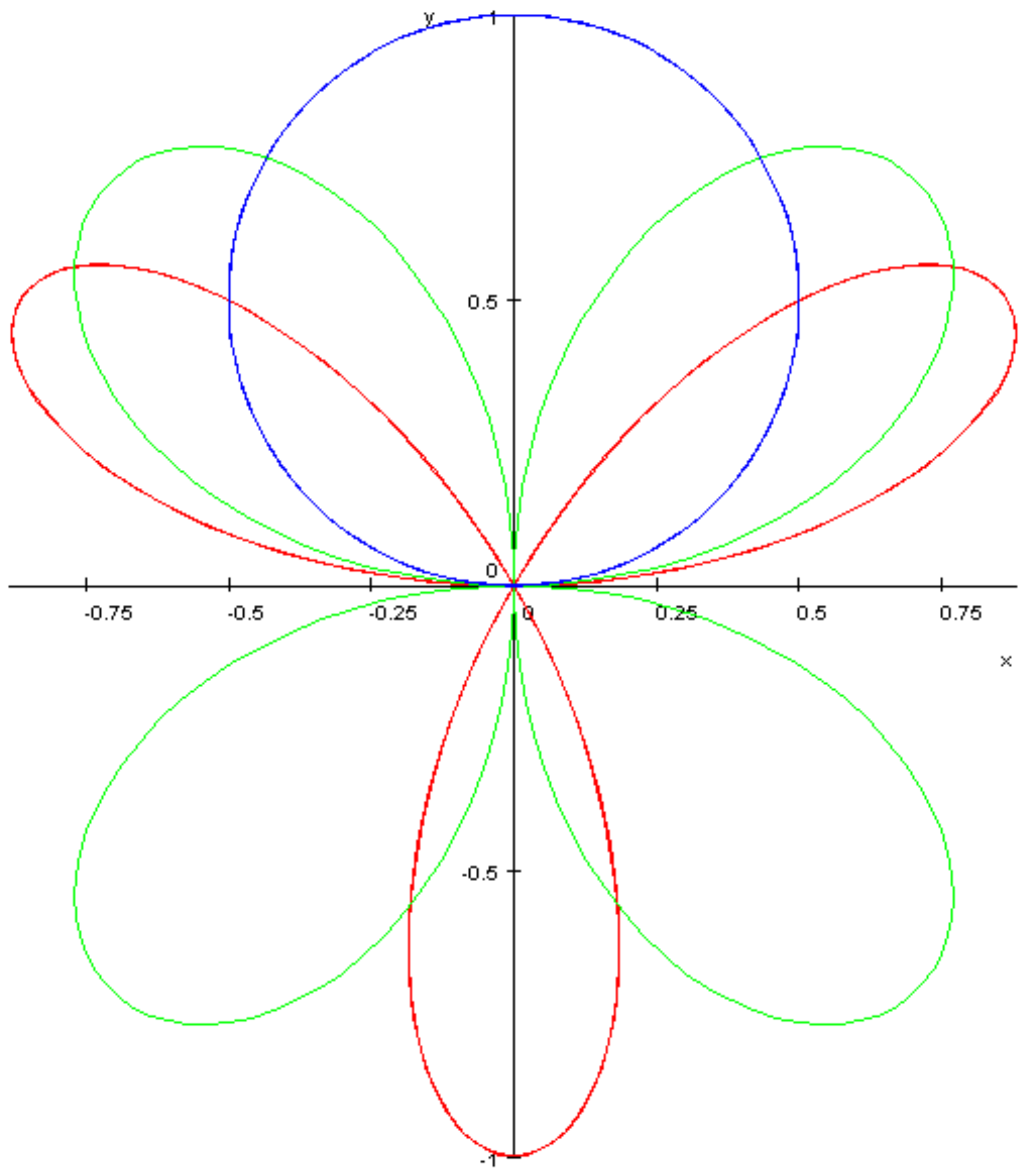
- アルキメデスの螺旋
  - $r = c * \theta$
  - $c = r \ / \ \theta$ = 最終的な半径 / ($2\pi\times$回転数)
- 対数螺旋
  - $r = e^{c\theta}$
  - $c = \log(r) \ / \ \theta = \log($最終的な半径$) \ / \ (2\pi\times$回転数$)$
- Lituus螺旋
  - $r = a / \sqrt{\theta}$ $\theta=0 \rightarrow r=\infty$
  - a = 最終的な半径 $/\sqrt{2\pi}$

薔薇曲線

- r = sin 2θのとき、曲線はXに似た形となる。
- r = sin 3θのとき、曲線はYに似た形となる。
- r = cos 2θのとき、曲線は+に似た形となる。



$r = \sin(\theta \times n / d)$



課題

- 外トロコイド曲線のうち 1 つ
- 内トロコイド曲線のうち 1 つ

- 一つ以上を描画するプログラムを提出のこと
- tkinterのライブラリを使って描くこと
- ファイル名は、Assign02.pyにて

グラフ処理アプリケーション

- Mac OS Xの場合は、Grapherが標準で添付

- Windowsの場合はFunction View, Microsoft Mathematics（旧製品）, あるいは、Word・OneNote用のMicrosoft Mathematics Add-inを使用

- 定積分の計算を数値積分で行なっている
- Grapherは、数式の微分形、積分形を持つことができる