スクリプト言語プログラミング
Pythonによる数値解析

第8回講義資料

箕原辰夫

実数の計算とmathモジュール

- 指数表現と正規化

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 \Rightarrow 5.67 \times 10^{-4} \Rightarrow$5.67e-4
- $1230000.0 \Rightarrow 1.23 \times 10^{6} \Rightarrow$ 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) …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（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

- 誤差許容範囲を設定する
  - *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）

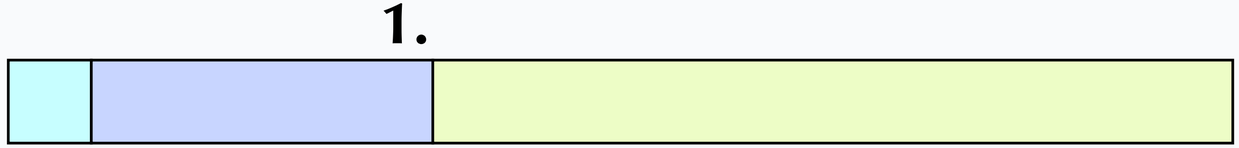
    $4.94065645841246544\times10^{-324}$〜$1.78769313486231570\times10^{+308}$

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



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

Python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ε
  - 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

    $$1 + \varepsilon > 1$$

浮動小数点型数値の精度

- 機械イプシロン（machine epsilon）εを求める
- b進法でp桁の浮動小数点数の場合、計算機イプシロンはb^{1-p}となる。
- IEEE 754のbinary32（単精度）では、b = 2、p = 23なので、$\varepsilon=2^{1-23}=2.384\times10^{-7}$となる。
- IEEE 754のbinary64（倍精度）では　$\varepsilon=2^{1-53}=2.220\times10^{-16}$となる
- binary128（四倍精度）では　$\varepsilon=2^{1-113}=1.926\times10^{-34}$となる。

誤差解析

- 補助資料を使って説明
- 誤差を少なくするアルゴリズム
  - 2次方程式の解を求める場合
  - 無理数の極限値を求める場合

ライブラリ（パッケージ・モジュール）を使うとき

- いくつかの記述法がある
- **import** [パッケージ名.]モジュール名
  - プログラム中で「モジュール名.名前」で利用可能
  - 例：**import** numpy

    numpy.array([ ])
- **import** [パッケージ名.]モジュール名 **as** 省略名
  - 「省略名.名前」で利用可能
  - 例：**import** numpy **as** np

    np.array([])
- **from** [パッケージ名.]モジュール名 **import** クラスあるいは関数名
  - 「クラス名あるいは関数名」で利用可能
  - 例：**from** numpy **import** array

    array([])
- **from** [パッケージ名.]モジュール名 **import** *
  - モジュールで定義されているすべての「クラス名あるいは関数名」で利用可能
  - 例：**from** numpy **import** *

    array([])
- **from** パッケージ名 **import** モジュール名
  - 「モジュール名.名前」で利用可能
  - 例：**from** numpy **import** linalg

    linalg.det([[1]])

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）3.5より
- math.nan…Not a Number（浮動小数の非数）3.5より

誤差を少なくするアルゴリズム(3)

- 極限値の計算
  - ネイピア数（自然対数の底）を求める

    $e = \lim_{n\to\infty} (1+1/n)^n$
  - マクローリン展開によって求める（次の回のスライドで）

    $$e^x = 1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots+\frac{x^n}{n!}+\cdots$$
- math.powを使って定義に従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
```

- 丸め誤差と打ち切り誤差によって、誤差を最小にするnが存在する

整数との変換

- math.ceil(x)…$\lceil x \rceil$　大きいか等しい一番小さな整数
  - 計算結果は整数
- math.floor(x)…$\lfloor x \rfloor$　小さいか等しい一番大きな整数
  - 計算結果は整数
- round(x)…四捨五入（境界値は偶数に丸めるので注意！）
- round(x, n)…小数点以下n+1桁で四捨五入（偶数注意！）
- math.trunc(x)…小数部を切り捨てる、int(x)と等しい

Decimalクラスを利用する四捨五入

- Pythonでは、四捨五入は、偶数にまとめるのが標準になっている（ROUND_HALF_EVEN）
- 四捨五入を正確に行ないたい場合には、DecimalクラスとDecimalクラスのquantize(rounding=ROUND_HALF_UP)メソッドを用いる
- Decimalを利用した例

```
from decimal import *
a = Decimal( "4.5" )
print( a )
print( a.quantize( Decimal( "1." ), rounding=ROUND_HALF_UP ) )
```

通常の四捨五入をする関数の定義

- 正の数について、小数点digit桁で四捨五入する関数は、次のように定義できる

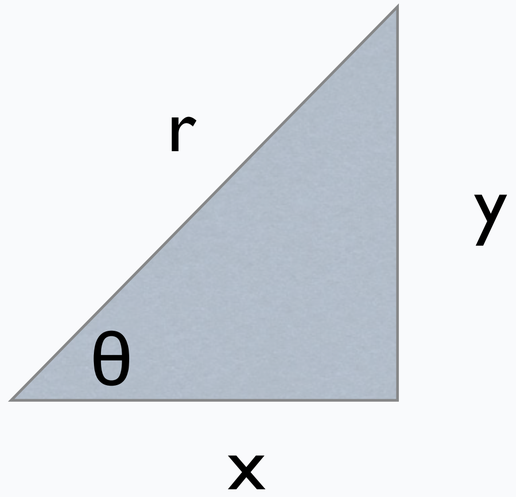
```
def round(val, digit=0):
    p = 10 ** digit
    return (val * p * 2 + 1) // 2 / p
```

- また、負の数にも対応するには、次のように定義する

```
def allround(val, digit=0):
    p = 10 ** digit
    s = math.copysign(1, val)  # -1.0 if val < 0 else 1.0
    value = (s * val * p * 2 + 1) // 2 / p * s
    return int( value ) if digit == 0 else value
```

- 参照： https://note.nkmk.me/python-round-decimal-quantize/

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
- math.hypot(x, y) → r

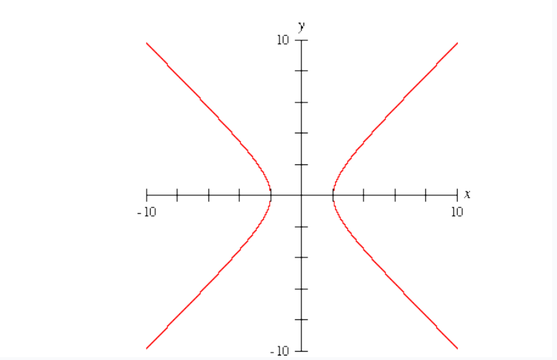
Radian体系とDegree体系、分秒

- 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

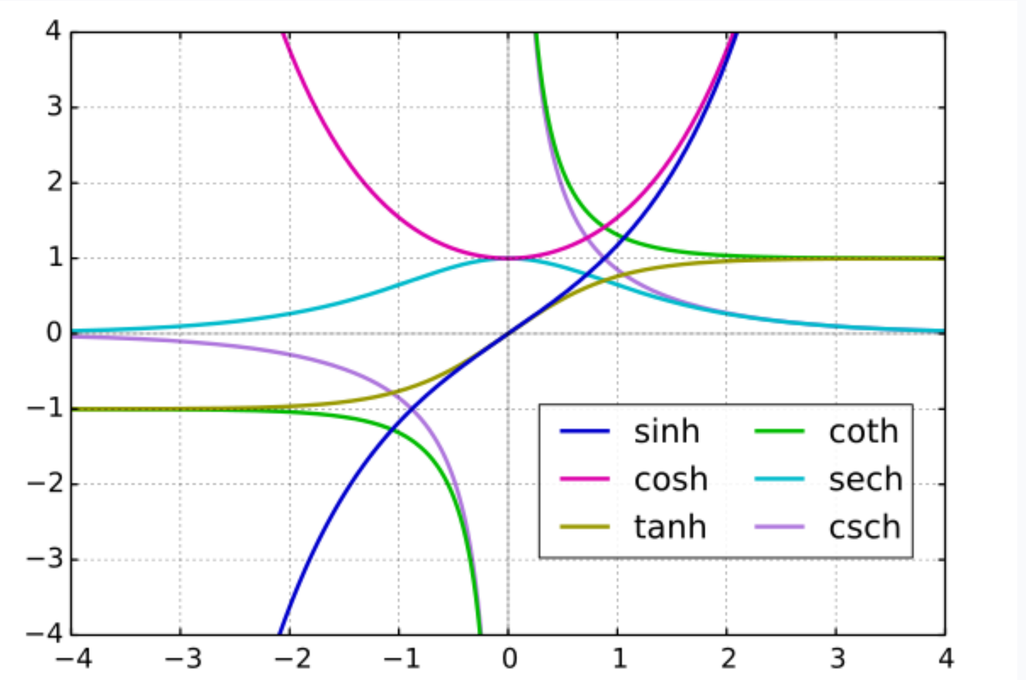
degree	0°	45°	90°	180°	270°	360°
radian	0	π/4	π/2	π	3π/2	2π

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
  - math.sinh(x)…双曲線正弦
  - math.cosh(x)…双曲線余弦
  - math.tanh(x)…双曲線正接
  - math.asinh(x)…逆双曲線正弦
  - math.acosh(x)…逆双曲線余弦
  - math.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

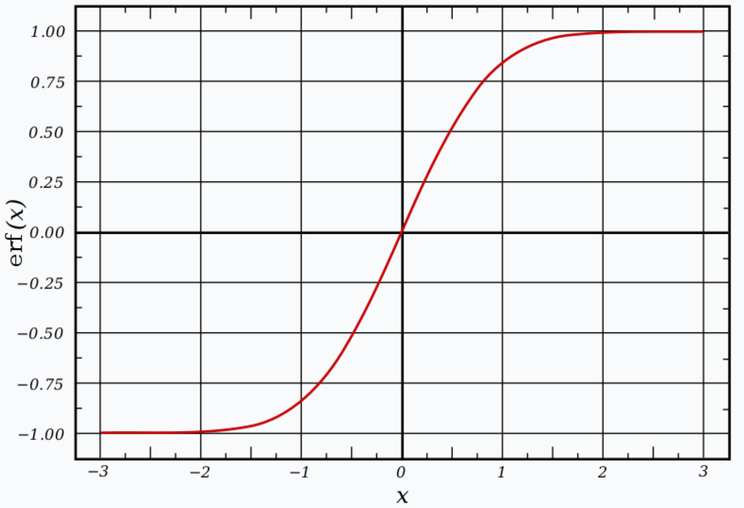
- math.factorial(n)…nの階乗（正の整数）
- math.gamma(z)…ガンマ関数（zの階乗：本来はzは複素数まで受け入れる筈だが、Pythonでは、実数まで）なお、自然数nについては、math.gamma(n+1) = n!
- math.gcd(a, b)…aとbの最大公約数（aとbは整数）
- math.erf(r)…誤差関数、
  シグモイド形状の特殊関数

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$$

誤差関数

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

標準シグモイド関数



対数

- $x = a^p$　aは基数　pは指数　$64 = 8^2$　power(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、
  - $\log_{10}(x) = \log_e(x) / \log_e(10)$
  - $\log_2(x) = \log_e(x) / \log_e(2)$

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

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

対数関数と指数関数

- math.log(値) … 自然対数
- math.log1p(値) … 1 + 値 の自然対数
- 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
- math.sqrt(x) … xの平方根を求めるmath.pow(x, 0.5)
- math.expm1(x) … eのx乗から1を引いた数を求める

log1pとexpm1

- math.log1pおよびmath.expm1は、0に近い値の対数を用いるときに利用できる。
- と言っても、かなり小さい数の場合には、0になってしまうので、使用範囲は限られる
- 例：

```
val = 1.2e-200
math.exp( math.log( val ) )
    # 1.1999999999999867e-200
math.expm1( math.log1p( val ) )
    # 1.2e-200
```

その他の関数

- abs(x)
  - xの絶対値を求める　　abs(-9) ⇒ 9
- math.copysign(x, y)
  - x の大きさ（絶対値）で y と同じ符号の浮動小数点数を返す。
- 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(4.7, 1.3, 3.9) ⇒ 1.3
- math.fsum(リストなど)
  - 精度の高い総和を求める

Python 3.5以降に加わった関数

- math.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
- math.comb(n, k) … 組み合わせ数を求める 3.8より
- math.perm(n, k) … 順列数を求める 3.8より
- math.remainder(x, y) … 近い方の値からマイナスで表示される剰余 3.7より
- math.dist(p, q) … p=(x, y)とq=(x, y)の間の距離を求める 3.8より
- math.tau … τ定数（2π）6.283185307179586 3.6より

IEEE 754を補佐するmathモジュール関数

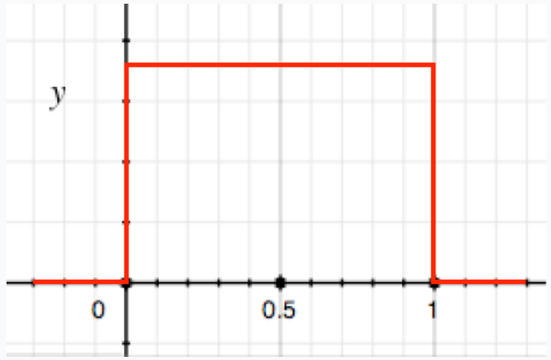
- math.frexp(x) … xを表現する(m, e)を返す x == m* (2**e)となるようなm（実数型）とe（整数型）
- math.ldexp(m, e) … m * (2**e)となるような実数を返す
- math.modf(x) … xの小数部（実数型）と整数部（実数型）を返す

random関数

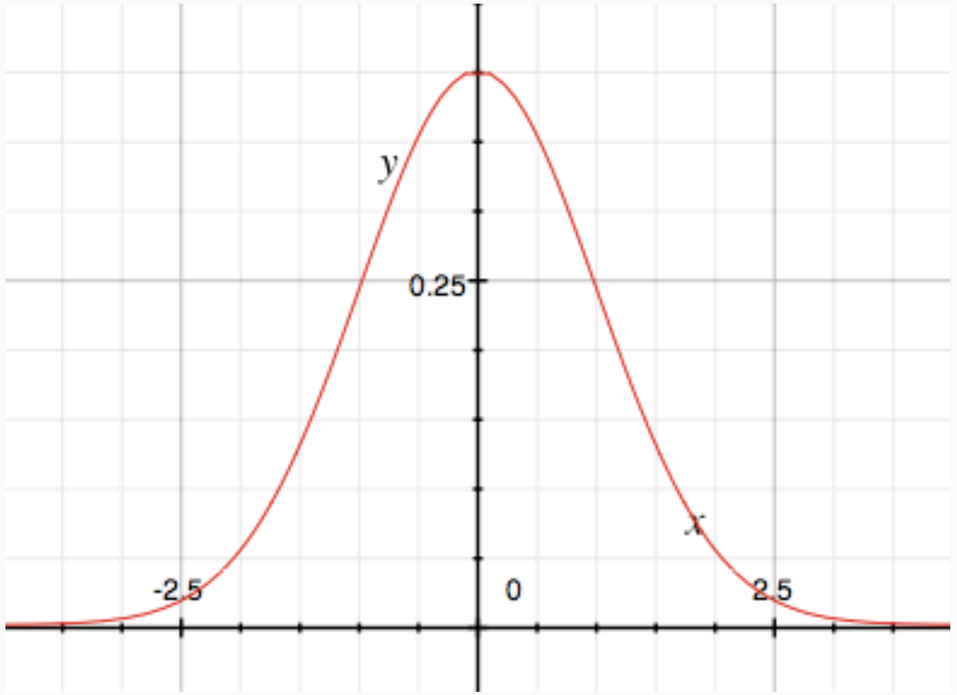
- **import** randomが必要
- random.random( )
  - 0から1未満の乱数（任意の数）を返す
- random.uniform(a, b)
  - a以上b以下の乱数（任意の数）を返す
- random.randint(m, n)
  - m以上n以下の乱数（整数）を返す
- random.gauss(m, sigma)
  - gauss分布（正規分布）で平均がm、標準偏差がsigmaの乱数を返す

正規分布の乱数

- 一様乱数



- 正規分布の乱数



matplotlibでグラフ表示

- **import** numpy **as** np
- **import** matplotlib.pyplot **as** plt

- X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
- C, S = np.cos(X), np.sin(X)

- plt.plot(X, C)
- plt.plot(X, S)

- plt.show()

- グラフィックバックエンドを自動に設定すると（別の回答：ツール>>設定>> IPythonコンソール>>グラフィック->自動に示されているように）

mac OS の Grapherでの関数プロット

- アプリケーション >>> ユーティリティ >>> Grapher
- ^ べき乗
- Option + v … 平方根 √
- ギリシャ文字は、綴りを入力
  - 例：θ…theta, π…pi
- rとθで、極座標で入力することも可能
- 新規方程式、テンプレートで媒介変数の式を出せる（式入力のパネルの右端のところがメニューになっている）
- 1...10のように、下限値と上限値を指定した範囲を指定できる
- { 1...10 }のように、for文のように繰返しを指定できる
- erfなどいくつかの特殊関数が用意されている

WindowsはMicrosoft Mathematics

- アプリケーション
  - https://microsoft-mathematics.jp.uptodown.com/windows
  - あたりからダウンロードしてください。もうサポートが終わった製品なので、公式にはサポートされていません。

- Microsoft Mathematics Add-In for Word and OneNote
  - 後継の製品で、WordやOneNoteで同じようなことができる製品になっています。
  - https://www.microsoft.com/ja-jp/download/details.aspx?id=17786