公開コピー誌
Mathematicaのメモ

暗黒通信団

■はじめに

Mathematica（マセマティカ）は数学全般を扱うためのアプリケーションだ。全般というのは本当に全般で、方程式を解いたり、グラフを描いたり、微分や積分や級数の計算をしたり、順列組み合わせを計算したり、その他諸々のことができる[*1]。実際は数学以外にも様々なことができるのだが、とりあえず数学に絞らないと何も解説できないので本稿では数学に限って記述する。

非常に悲しいことに Mathematica はフリーではなく売りものであるから、使うためにはまず買ってこなくてはいけない。そして Mathematica は学生版以外では強烈に高いから、学生になるか、会社や研究室で買ってもらうのが正しい。ただし本稿はある程度コンピュータに精通した人を対象にするので、インストールの方法はこまごまと書かない。インストールのサポートは売り主（ウルフラム・リサーチ社）の仕事だ。インストールが無事にできたとして、次にやることは Mathematica に与える命令の列を書くことだ。とりあえずテキストエディタ[*2]で「`Export["output.txt",Solve[x^2-3x+2==0,x]]`」という 1 行の中身のファイルを作ろう。ファイル名は `sample.m` とでもしておく。

そしてコマンドラインを開き「`math < sample.m`」と打ち込む[*3]。もしかしたら `math` というコマンドがないと怒られてしまうかもしれない。Mathematica をインストールすると自動的に`/usr/local/bin/math`(Linux,Mac) か `C:/Program Files/Wolfram Research/Mathematica/10.0/math.exe` というコマンドができているのだが、そこにパスが通っていない可能性がある。そういう場合には、例えば (Linux であれば)「`/usr/local/bin/math < sample.m`」というふうに絶対位置を指定してやる必要がある。するとたぶん以下のように表示されて終わるだろう[*4]。

```
Mathematica 10.0 for Linux x86 (64-bit)
Copyright 1988-2014 Wolfram Research, Inc.
In[1]:=
Out[1]= output.txt
In[2]:=
```

無事に `output.txt` というファイルができていたら中身を見る。

```
{x -> 1}
{x -> 2}
```

と書いてあれば成功だ。もし上記がうまくいかない場合、ウルフラム・リサーチ社に電話をかけるべきだ。この例はつまり $x^2-3x+2=0$ を解いたものである。4 次方程式まではきっちり解いてくれるが 5 次以上は一般には無理だ[*5]。

❶ 方針

Mathematica は基本的にプロユースであるから、以下では「はじめに」のような手取り足取りの書き方はしない。そして、もともと自分のために作ったものなので、話題は偏っているかもしれない。本稿の基本的な考え方は次の通りである。

1. GUI[*6]を前提にしない。Mathematica の入門書は GUI 操作を前提にしたものがほとんどだが、Mathematica の GUI 環境は正直使いにくく、ネットワークからログインしてバックエンドで実行することも難しく、実はおすすめできない。また、通常のセーブデータにはパスや機種などの情報が入っていて、異なる環境では動かないことが多い。さらに Mathematica は高額なので学生版以外を個人で買うことは難しく、共用のサーバで動かすことが実際的であるからだ[*7]。このため本稿では、関数を羅列したスクリプトファイルを作り、`math` カーネルに直接入力する方法を前提とする[*8]。ちょうど「はじめに」でやったことがそれだ。この方法は知る限り最も可搬性があり効率的である。
2. 数学的な処理と数学的な図形の生成以外には極力ふれない。現代の Mathematica には多様な関数があり、例えば地図解析や化学構造解析や機械学習や音声解析や経済データの分析さえも可能だが、それらは数学的な処理をベースにした応用であり、数学の部分が分からないと少しのことでつまずく。個人的な感想ではあるが、Mathematica は（開発元には申し訳ないが）あくまで数学のためのツールとして割り切った方がよく、Mathematica でしかできないことも大概は数学的なことである。そしてまた、数学に関しては最強のツールである[*9]。
3. 極力簡潔に書く。厳密な定義、細かい挙動はヘルプで補完

[*1] ただし定理の証明は難しいかもしれない。
[*2] ノートパッドとか秀丸エディタや sakura エディタや jed や emacs や vi や…色々だ。ただし MS WORD とか Open office はダメだ。ワープロはエディタじゃない。
[*3] コマンドラインが何か分からない人は、「ターミナル」とか「コマンドプロンプト」とか「CUI」といった単語で検索しよう。
[*4] “`Linux`” の部分は “`Windows`” になってるかもしれない。
[*5] 一般の 5 次方程式に対するベキ根による解の公式は存在しない（アーベルの定理またはガロア理論の応用）。ただし、超越関数を駆使すれば、5 次方程式の解の公式は存在するようだ。
[*6] グラフィカルユーザインタフェースという。マウスでダブルクリックして起動するようなシステムは GUI だ。
[*7] そのようなサーバでは GUI に伴う無駄な負荷は嫌われるだろう。
[*8] UNIX 系 OS であれば、スクリプトファイルの先頭に「`#!/usr/local/bin/math -script`」などと書いて実行ビットを立ててもよい。
[*9] もちろん僕は Maple や maxima や、古くは Derive 等も使ったことがあるが、Mathematica にはどうしても及ばない。フリーソフトには頑張ってほしいのだけど。

することを前提にしている。そもそも Mathematica の関数は膨大過ぎて、そのほとんどは紹介すらできない。本稿で書けるのは本当に基本中の基本であり、本稿をすべて理解しても初心者の入り口（の手前）に立っただけである。加えて Mathematica は様々な書き方が許容されているので、読者の流儀とは異なるかもしれない。例えば APL 的記法とかが好きな方々[*10]には不満であろうと思うが、そこは許してほしい。本稿で前提としているバージョンは一応 10 である。それ以前でも大概は動くだろうが、確認はしていない。特に（グラフ理論を含む）拡張パッケージの周辺はバージョンによって大きく異なるので留意してほしい。

なお、Mathematica はウルフラム・リサーチ社の商標である。本稿は Mathematica の販促に供すると思うのでいちいち許可は取っていないが、何か問題が起きればさっさと回収して絶版になるだろう。また、初版の頒布が終わったら増刷せずに PDF で無償公開する予定である。その代わり定価は極力安く設定しているので怒らないでほしい。

❷ 文法

Mathematica は計算機言語としての基本機能を備えるが、一般的な計算機言語に対して特筆すべき点が二点ある。

- 行列も配列も複数の解も、変数や定数をまとめて扱う場合には、すべて「リスト」という共通の概念で実現されている。リストの中身は何でもよい。リスト自身もリストに入れることができる。「リスト」の扱いを理解することは Mathematica の理解において必須である。
- Mathematica ではすべてが「関数」である[*11]。`If[]` 文や `For[]` ループですら関数で実現されている。関数は `A[B]` という形で統一されている。`A` はヘッドと呼ばれる文字列で関数名（関数の機能）を示し[*12]、`B` は引数で、一般にリストをとる。形式が統一されているため、リストを引数に変換することもできる。それが `Apply[]` である。例えば `Apply[f,{a,b,c}]` は `f[a,b,c]` と展開される。`Map[]` は、引数の各要素（リスト）に関数を作用させる効果がある。つまり `Map[f,{a,b,c}]` は `{f[a], f[b], f[c]}`と展開される。リストと `Map[]` を組み合わせることは実質的に `For[]` 構文に相当し、ループで回すよりもリストを作用させるほうが一般的に高速だ。

❶ 基本文法と基本的な関数

以下は基本的な事項である。細かい説明は不要だと思うので箇条書きにする。

- 終了するには、`Exit[]` 関数か `Quit[]` 関数を使う。引数なしで終了できる。バッチ的に処理するときは書かなくてもスクリプトが終われば終了する。
- 大文字小文字は区別される。組み込みの関数はすべて大文字から始まる。例えば「`ListPlot[]`」のように、複数の単語から成る関数名では、各単語の先頭も英大文字である。変数は自由に名付けて構わない。
- 「`%`」は直前の計算結果を意味する。実は Mathematica は実行したすべての結果を保持しているので、「`%`n」とすると n 番目の結果を参照できる。
- `math` カーネルではなく対話式に使っているとき、計算を中断するには ALT と「.」を同時に押す。
- 画面出力行末の「`\`」は結果が次の行に続くことを示す。`Export[]` 関数でファイルに出力した場合、この記号はつかない。
- コメントは `(*`と`*)` で囲む。
- 進数の表現は、例えば 2 進数であれば `2^^101110011` などと書く。逆に 10 進数 n の 2 進表記文字列を得たければ `IntegerString[n, 2]` とすればよい。`2` のところを `"Roman"`にすればローマ字表記も出てくる。
- 行末にセミコロンをつけると行の結果は出力されない[*13]。
- ヘルプは調べたい命令の前に「`?`」をつける。例えば「`?N`」であれば「`N[expr] gives the numerical value of expr.`」といったヘルプが表示される。しかし現実的には関数名をネットで検索すれば日本語ヘルプが出てくるので、その方が効率的だろう。ある関数のオプションを調べるには `Options[]` 関数を使う。`Options[`*Plot*`]` とすると `Plot[]` 関数のオプションとデフォルト値がずらずらと出てくる。
- 外部コマンドの実行は `Run[`*COMMAND*`]` とする。Linux なら例えば `Run[`*ls*`]` でファイル一覧が出る。実際には `RunThrough[`*COMMAND, input*`]` のほうが使い勝手がよい。これは `COMMAND` に入力を与え、その出力を取り込む。
- 非常に長い時間がかかる場合、処理が終わったらメールを送ることもできる。携帯電話などに送ると便利である。スクリプトの最後に、
  `SendMail[`*"To"->"*宛先メールアドレス*", "Subject"->"*題名*", "Body"->"*本文*", "Server"->"SMTP* サーバ*", "From"->"*送信元*"]` と書いておけば良い。
- 乱数の種を設定は例えば `SeedRandom[`*123*`]` などとする。種を設定すれば乱数を使ったプログラムでも状況を再現できる。
- 変数 x の中身を初期化するには `Clear[x]` とする。関数 `f[x_]` をクリアするには `Clear[f]` とする。似た関数に `Remove[]` というのがあるが、`Remove[x]` はその変数名 `x` ごとすべて消し去る。
- 外部コマンドファイルの読み込みは例えば`<< "sample.m"`とする。サブルーチンをファイルごとに分けて書き、`<<`でつなぐなどの使い方がある。
- 算術演算子は、`+`, `-`, `*`, `/`, `^`など。通常の数学演算式と同じに

[*10] Wikipedia の Mathematica ページに書いてあるがマニアックすぎないか？
[*11] 実は上のリストも `List[]` という関数である。
[*12] `Head[]` という関数に式を与えるとヘッドが得られる。
[*13] 実はセミコロンさえも実装としては、前後の式の後ろだけを評価して返す演算子である。
[*14] ただし内部ではこれらもすべて関数に展開されている。例えば `1+2` は「`Plus[`*1,2*`]`」といった具合である。

記述できる[*14]。

- 予約された定義値は次である[*15]。Pi（円周率）, E（自然対数の底）, I（虚数単位）, Infinity（無限大）, True（真）, False（偽）, Degree（度数を Rad に直す定数）, GoldenRatio（黄金比）, EulerGamma（オイラー定数）, Catalan（カタラン定数）, Khinchin（ヒンチンの定数）, Glaisher（グレシャーの定数)。
- 複素数から実部と虚部を取り出すには Re[] と Im[] を使う。
- 数の型としては Integer, Rational, Real, Complex の 4 つがある。 Rational は Integer 同士の分数で定義される型（有理数）である。前者 3 つにはそれぞれ対応する集合が定義されており、Integers, Rationals, Reals である。素数の集合は Primes である。単位円周は Disk[*{centerx, centery}*]、単位球は Ball[*{centerx,centery,centerz}*] である。これらは例えば Element[*Pi, Reals*] といった集合判定関数で使う。円周上に点があるかどうかなどの判定にも使う。乱数を発生させる関数は Random[] であるが、この引数に種類と範囲を与えるときも型を使う。例えば Random[*Integer, {1,10}*] といった具合である。

❷ 定義・代入・条件構文

構文について述べる。

- 代入
  代入には単純代入（記述位置での変数の内容を反映して一度だけ代入する）と遅延代入（実際に変数 A を使う時点で、そのときの B の値を随時参照して代入する）がある。前者は単に A=B と書けばよい。後者は A:=B と書く。なお方程式を与える場合には論理等号==を使う[*16]。
- 関数定義
  関数を定義することは重要で頻出する。例えば f[x_]:=x^2 は $f(x) = x^2$ の定義である。定義関数名の中で変数 x の次に _をつける点に注意する。
- 条件分岐
  基本は、If[*condition,then,else*] という関数である。condition が True なら then を, False なら式 else を実行する。例えば「If[*a>1*,Print[*a*],Print[*0*]]」などである。これを簡略して書く記法があり、「もし B ならば A を実行」を A /; B と書く。
  Which[*[condition1,return1,condition2,return2,…]* という関数は、condition1 を評価して True だったら return1 を実行して終わる。condition1 が False なら次に condition2 を評価し、True だったら return2 を実行して終わる。以下同様である。
  いわゆる Switch-Case 文もある、Switch[*A,B1,C1,B2,C2,…*] は、A=B1 なら C1 を返し、A=B2 なら C2 を返す。引数の最後に書いた値はデフォルト値で、どれにもマッチしなかった場合に返される値である。
- 繰り返し
  for 文と while 文がある。C 言語の for(A;B;C){execute} は For[*A,B,C,execute*] と書かれる。あるいは While[*condition,execute*] と書けば condition が真であるあいだ execute が実行される。
- 置換
  これは特に重要である。解を求める関数では、解の集合が置換構文で与えられるので、これを知らないと方程式の解も扱えない。基本となる関数は ReplaceAll[*original,rule*] である。例えば ReplaceAll[*x-x^2,x->*N[*Pi*]] といった書き方である。これは簡略した記法（スラッシュドット）が存在し、x-x^2 /. x->N[*Pi*] などと書く。

❸ 入出力

- 多桁出力は N[*equation, digits*] とする。例えば、「N[*Pi,1000000*]」は円周率を 100 万桁精度で出す。
- ファイルからのデータ入力
  Import[*"filename","format"*] という関数を使う。
  例えば、L=Import[*"sample.csv","CSV"*] で L に sample.csv を読み込む。
- ファイルへの出力
  Export[*"filename",data,"format"*] という関数を使う。例えば、Export[*"sample.csv",data,"CSV"*] と書けば data を sample.csv に CSV フォーマットで書き出す。フォーマット指示部（"format"のところ）には Table、EPS, PDF, SVG, PICT, WMF, TIFF, GIF, JPEG, PNG, BMP, PCX, XBM, PBM, PPM, PGM, PNM, DICOM, AVI, DXF, STL, WAV, AU, SND, AIFF などが入る[*17]。何も書かなければファイル名の拡張子をもとに自動判断される。不要な改行マークなどが入らないので、出力は Export を使うことがおすすめだ。画像用のオプションとして、*ImageSize*->72 x(x インチ) と *ImageResolution*->d(dpi 単位) がある。3D イメージの場合は POV によって PovRay というレイトレーシング用のファイルを出力することもできる。
- データではなくスクリプト自体の C 言語コード生成は CForm[*equation*] とする。関数名の先頭が大文字になっているため、コンパイルするには mdefs.h という関数名変換のためのヘッダが必要だ。このヘッダはインストールパッケージに含まれている[*18]。Compile[*variables, equation*] は式全体を評価して、コンパイル可能な言語に変換する。Export[*"sample.c", Compile[{x}, x+1]*] とすると main() がないだけのコンパイル可能な C

[*15] http://reference.wolfram.com/language/tutorial/MathematicalConstants.html
[*16] これは実は Equal[] という関数である。
[*17] フォーマットを C にすると、data を計算する C 言語の関数を出力することになっているが、実際はよく失敗するので使えない。
[*18] find などで探せばすぐ見つかる。

言語関数群ができあがる。ヘッダの WolframLibrary.h や WolframRTL.h はインストールディレクトリの下の SystemFiles/IncludeFiles/C/にある。ただしこの機能は単なる式では使えるが Mathematica の固有関数を含む場合うまくいかないことが多い。例えば Export["*sample.c*", Compile[{*x*}, N[*Pi*,*1000000*]]] はうまくいかない。

表示形式を変えて人に見やすく（あるいは機械にとって見やすく）するいくつかの記法がある。

- 演算子をすべて関数形式で表示する：実用的ではないかもしれないが、「// FullForm」をつける。例えば「x-y*z-x/z // FullForm」は「Plus[*x*, Times[-*1*, *x*, Power[*z*, -*1*]], Times[-*1*, *y*, *z*]]」と展開される。現実問題としてこれは Mathematica スクリプトを別の言語で機械生成する場合に有効で、その他の使い方は思いつかない。
- 行列的表現：Mathematica は複数データを 1 次元のリスト表現で格納しているが見やすさのため行列にして表示してほしいことがある。その際には「//TableFrom」とか「//MatrixForm」をつける。
- 短縮表現：// Short は多項式の途中をスキップして 1 行以内に抑えた出力をする。

❸ リスト

リストとは、Mathematica が複数のデータをまとめて扱うときの共通形式である。C 言語の配列と同じような概念だが、Mathematica のリストの場合インデックスは 1 から始まる。C 言語のように 0 から始まるわけではない。リスト L の n 番目の要素へのアクセスは L[[n]] である。例えば、Print[*{3,1,4,1,5}[[3]]*] などとする。n に負の値を入れると末尾からのカウントになる。リストのアクセスは Part[] 関数を使って例えば Part[*{3,1,4,1,5,},3*] としても良い。

❶ リストの代表的な使われかた

- Solve[] など解が複数ある場合の結果は「置換のルールのリスト」で与えられる。Solve[*x^2-4==0,x*] と書けば、{{x -> -2}, {x -> 2}}という答えが得られる。解のリストにしたければ最初から x /. Solve[*x^2-4==0,x*] と書いておくと良い。
- 複数の関数もリストである。Plot[] で複数の関数を同時に表示する場合には、例えば Plot[*{x^2,x^3},{x,-2,2}*] などと書く。後半の{x,-2,2}は表示範囲を与える指定だが、これもリストになっている。
- リスト全部に対して四則演算 (For[] 文の代用) することもできる。例えば、{1,2,3,4}*5 は各要素を 5 倍する。
- リストのそれぞれの要素に対しての演算することもできる。例えば{1,2,3,4}+{5,4,3,2}の結果はそれぞれの要素が 6 になる。
- 座標列、行列もリストである。例えば、{{1,2},{3,4}}は 2×2 行列である。

❷ リストの生成

- 最も簡単なリスト生成は{, , , }と書く。例えば、{3,1,4,1,5}などとすればリストが作れる。空のリストは{}である。
- Table[] を使って半自動的に作ることもできる。例えば、「Table[*k*2, {k,1,6}*]」で「{2, 4, 6, 8, 10, 12}」が生成される。ステップも与える場合には、Table[*k^2, {k,1,6,1/2}*] などとする。次のようにすると二重ループも作れる。Table[*k*a, {k,1,6},{a,1,4}*]。
- その他のリスト作成関数として、Range[a,b]（a から b までのリストを生成）, Array[f,n]（f_i を要素とする長さ n のリストを生成）, ConstantArray[c,n]（c を n 個並べたリストを生成）, RandomInteger[*{1,10},n*]（範囲が 1 から 10 までの n 個の乱数からなるリストを生成）などがある。

❸ リストの情報

- リスト L の長さを調べるには Length[*L*] 関数を使う。
- リスト L の次元を調べるには Dimensions[*L*] を使う。
- リスト L の中に指定の要素 x があるかどうかを調べるには MemberQ[*L,x*] を使う。
- リスト L から指定要素（例えば整数）の個数を得るには Count[*L, _Integer*] とする。

❹ リストの加工

- リスト中の置換は例えば ReplacePart[*L, {1->3, 4->6}*] とする。
- リストを合成する関数としては、Append[*L,tsuika*]（後方追加[*19]）, Prepend[*L,tsuika*]（前方追加）, Insert[*L,tsuika,n*](途中の位置 n に追加), Join[*L1,L2*]（複数のリスト同士を単に結合）, Riffle[*L1,L2*]（複数のリストを交互に挿入）などがある。集合関数を用いてリストを加工することもできる。Union[*L1,L2*]（和集合[*20]）, Intersection[*L1,L2*]（積集合）, Complement[*L,l1,l2,l3*]（補集合）, Subsets[*L*]（全ての部分集合のリスト）などである。
- 逆にリストを分解する関数としては、前述の Part[] の他、First[*L*]（第一要素）, Last[*L*]（最終要素）, Rest[*L*]（最初の要素以外）, Most[*L*]（最後の要素以外）, Take[*L,n*]（最初の n 個の要素）, Drop[*L,n*]（最初の n 個を削除）,

[*19] Append[] はリストだけでなく、式に項を付け加えるときにも使える。
[*20] Join[] との違いは、重複を許さない点である。

Select[*L,conditions*]（条件に合致する要素の抽出）, Cases[*L,pettern*]（パターンにマッチする要素の抽出)、Partition[*L,n*]（指定した長さの部分リストに分解）などがある。

Cases[] は特に奥が深い。例えばリストから整数だけを抜き出すときには Cases[*L, _Integer*] とする。リストから整数以外を抜き出す場合には Except[] 関数を使い、Cases[*L,* Except[*_Integer*]] とする。Cases[] には条件もかける。例えば Cases[*L, _Integer :> x^2*] などとする。逆にリストから整数を削除するには DeleteCases[*list, _Integer*] とする。

- その他のリスト操作関数としては、Reverse[]（反転)、Sort[]（ソート)、RotateLeft[]（左回転）, RotateRight[]（右回転）, Flatten[]（ネストしたリストの平坦化）などがある。Permutations[] は順列リストを生成する。面白いものとして、IntegerDigits[] は数値を数字のリストに分解する関数で、例えば IntegerDigits[*1234*] とすれば{1,2,3,4}というリストが得られる。ちなみに IntegerDigits[*1234,3*] だと 3 進数の数値列でリストを作る。

❺ 行列操作

行列はリストのリストによって表現される。以下はリストを行列としてみた場合の諸関数である。

- 内積は「.」をつかう。例えば「M.M」である。外積は Cross[] 関数を使う。
- Det[] は行列式、Transpose[] は転置、Inverse[] は逆行列、Eigenvalues[] は固有値、Eigenvectors[] は固有ベクトルを返す。
- 単位行列、対角行列の生成としては IdentityMatrix[*n*]（$n \times n$ の単位行列を生成)、DiagonalMatrix[*l*]（対角成分がリスト l になる対角行列を生成）がある。
- 固有値を根とする多項式の生成は CharacteristicPolynomial[*M, x*] とする。

❻ グラフ構造

グラフとはプロットのことではなく、グラフ理論のグラフである。グラフは基本的には隣接行列という行列の一種で表現されるので、リストの応用と見なせる。グラフの生成、加工としては以下の関数がある。

- グラフの作成は Graph[] 関数を使う。無向グラフであれば Graph[*{1 <-> 2, 2 <-> 3, 3 <-> 1}*] などと指定する。有向グラフなら<->が->になる。Export[] 関数で画像にすると、無向グラフだとエッジが矢印で表示されている。エッジにウェイトを設定するには、*EdgeWeight->*オプションでリストを与える。
- CompleteGraph[] は完全グラフを簡単に作る関数で CompleteGraph[*6*] とすれば頂点 6 で辺が 15 の完全無向グラフができる。
- AdjacencyGraph[] 関数は隣接行列からグラフを作る関数で、例えば AdjacencyGraph[*{{0, 1, 0}, {0, 0, 1}, {1, 0, 0}}*] とすれば 3 頂点の有向グラフができる。
- ランダムグラフの生成は RandomGraph[*n,m*] で n 個の頂点と m 本の辺を持つランダムグラフが生成される。
- 木の生成は KaryTree[] 関数を使う。KaryTree[*n,k*] で n 個の頂点がある k 分木を作る。
- 最小全域木はグラフ g を使って FindSpanningTree[*g*] とする。
- グラフ g からエッジ e を削除するには EdgeDelete[*g,e*] とする。
- グラフ同士を集合演算によって結合したり削除したりできる。それらの関数は、GraphUnion[]、GraphIntersection[]、GraphDifference[] である。

生成されたグラフがどのような性質を持っているのかを判定する関数がいくつかある。

- g にハミルトン閉路が存在するか否かの判定は HamiltonianGraphQ[*g*] 関数を使う。一筆書き可能かどうかの判定は EulerianGraphQ[*g*] 関数である。
- その他色々な判定関数がある。EmptyGraphQ[]（グラフが空かどうか)、SymmetricMatrixQ[*g*]（対称行列かどうか)、ReflexiveQ[*g*]（隣接行列が反射的な 2 項関係にあるかどうか)、UndirectedGraphQ[*g*]（無向グラフかどうか)、LoopFreeGraphQ[*g*]（自己ループがあるか否か)、AcyclicGraphQ[*g*]（非巡回グラフかどうか）などである。

❹ 式を解く、式の変形

❶ 式の基本処理

Mathematica の最もよく使う用途である。

- 線形多項式を解く場合、例えば、Solve[*x^2-1==0,x*] などとする。結果は「置換のルールのリスト」、例えば x->5 などと返ってくるので。これを x に作用させれば実際の解のリストになる。そのためには/. 演算子（置換演算子）を使い、「x /. x->5」とする。式をリストにして複数与えれば連立方程式も解ける。
- 式を最初から数値的に解くのは NSolve[] である。
  例えば、NSolve[*x^3/Pi-1==0,x*] とする。ちなみに平方根（$\sqrt{\ }$）は Sqrt[] で入力する。e^x は Exp[x] だ。$\log_e x$ は Ln[x]ではなくLog[x] である。
- 条件付きの式は Reduce[*x>0 && x^2-2*x-3==0,x,Reals*] などと書く。すると実数解の中で $x > 0$ に合致する x==3 が得られる。ちなみに Solve[] にも&&で条件指定をすることはできる。
- ニュートン法で解を見つけるためには、FindRoot[*equation,{valiable,init}*] を使う。例えば FindRoot[*x^5-x^4+x^3-x^2+x^1+4==0,{x,2}*] とすると、{x -> -0.926519}という数値解が得られる。この結果は機械精度に依存するので N[] 関数で多桁を出そうとしてもうまくいかない。機械精度自体は Print[*$MachinePrecision*]

で調べることができる[*21]。

- 展開にはいくつかの種類がある。基本的な展開の関数は Expand[] である。特殊関数も展開したいなら FunctionExpand[] を使う。入れ子の内部まで展開したいときは ExpandAll[] である。三角関数を含んだ式を展開するためには TrigExpand[] や TrigExpandAll[] を使う。
- 因数分解にもいくつかの種類がある。基本的な因数分解は Factor[] 関数を使う。オプションをつけて複素数まで許した因数分解をするには例えば Factor[*x*x+9,GaussianIntegers->*True] とする。三角関数の因数分解は TrigFactor[] である。因数分解とまでいかなくても、特定の変数に依存しない因子を外に括り出すには FactorTerms[]、FactorSquareFree[] を使う。
- 実際によく使う式変形としては「簡約化」がある。簡約化の基本は先の Reduce[] であるが、特に記号を減らす方向の簡約化は Simplify[]、FullSimplify[] であり、三角関数の簡約化は TrigReduce[] である。
- その他の式加工関数は以下の通りである。
  MinimalPloynomial[*numbers,variable*]（与えられた数値を根とするような最小次数の多項式を求める。エラーになる場合、数値は Rationalize[] で囲む）、CoefficientList[]（係数だけを抜いてリストにする）、MonomialList[]（単項式のリストにする）、Variables[]（式に含まれる変数をリストで返す）、PolynomialQuotient[*y,z,x*]（x についての多項式 y を z で除算したときの商を求める）、PolynomialRemainder[]（同様に剰余を求める）、Maximize[*y,x*]（x を変数としたときの式 y の最大値を求める）、Together[]（分母をまとめる）、Apart[]（有理数を部分分数に展開）、Collect[*equation, term*]（同類項をまとめる）、ContinuedFraction[*equation,n*]（連分数展開する。n は展開数）等。

❷ 式の近似

数式を近似することは結構高度な作業で、他のソフトではなかなかやりにくい。

- 小数を近い有理数に変換するには Rationalize[*number*] を使う。
- RootApproximant[*number*] は数 number を最も簡単な代数的数の 1 つに近似する。RootApproximant[*number,n*] とすると、最大 n 次で近似する。
- n 項までの連分数展開をするには ContinuedFraction[*number,n*] とする。結果はリストで返る。
- FindSequenceFunction[*numbers*] は整数のリスト numbers を与えると、それを数列と見なし、数列を生成する関数を返す。

❸ 関数の近似− FindFit −

（主に）最小二乗法でフィッティング関数を求める。色々なグラフソフトにある機能だが、Mathematica は非常に強力である。最も簡単な場合は、Fit[] 関数で線形フィッティングができる。以下は Mathematica のヘルプにある例だが、data = {{0, 1}, {1, 0}, {3, 2}, {5, 4}}という二次元の点列があったとき、Fit[*data, {1, x, x^2}, x*] とすると、3 つの関数 $1, x, x^2$ の組み合わせで得られる x についての関数を返す。

FindFit[] はこれが強力になったものである。書式は FindFit[*data, {equation, conditions},parameters, variables*] で、data のところには{{x1, y1}, {x2, y2}, {x3, y3}}という形でデータ点を与える。3 次元データであれば、{{x1, y1, z1}, {x2, y2, z2}, {x3, y3, z3}}となる。この場合、variables は{x,y,z}となる。

- 返値は「式のパラメータ」のリストである。
- パラメータの初期値を設定する場合は、「{{p1, p1init},{p2, p2init}}」などとする。
- デフォルトでは距離として最小二乗法を使う。
- conditions は不等式を「,」で区切って列挙する。
- 例：Print[FindFit[*data,{y=(1+Exp[b*Log[x]]) /(2+Exp[-a*Log[x]]+Exp[b*Log[x]])},a,b,x*]]

❹ 級数

関数を示すだけで良いに違いない。

- 級数和：Sum[*x,{i,imin,imax}*]
- 級数積：Product[*x,{i,imin,imax}*]

❺ グラフ

❶ 二次元グラフ

- グラフの基本的なプロットは Plot[] 関数を使う[*22]。例えば、Plot[*Sin[3*x]*Cos[5*x], {x, -Pi, Pi}*] といった具合である。プロット引数は Plot[] の引数の中で与える。以下はプロット引数（オプション）である。
  - ▷ 色の変更は *PlotStype->*で指定する。青色にするには例えば *PlotStyle->*Hue[*2/3*] とする。Hue[*h,s,b*] 関数は色調 (hue)、彩度 (saturation)、輝度 (brightness) で指定する色指定関数である。RGB で指定する場合は RGBColor[*0,0,1*]（青色）などとする。
  - ▷ 破線にするには *PlotStyle->*Dashing[*0.05*] と指定する。
  - ▷ ラベルを付けるには *PlotLabel->*"string"と指定する。

[*21] Linux では 15.9546 と出た。

[*22] Linux 等でネットワーク越しに使っている場合、端末に X サーバが動いている必要がある。Windows 端末の場合 Xming あたりを入れておくことが良い。

- ▷ 軸のラベルは *AxisLabel*->{"x","sin x"}。
- ▷ 枠をつけるには *Frame*->True と指定する。つけなければ False である。
- ▷ グリッドは例えば *GridLines*->Autometic などとしておく。
- ▷ フォント変更は *DefaultFont*->{"Helvetica",18}などと指定する。
- ▷ 縦横比は *AspectRatio*->0.2 など。

- コンター図（等高線）は、ContourPlot[*equation,{xrange},{yrange}*] とする。オプションとしては、*Contours*->n で分割数を指定。*PlotPoints*->で点数。例えば、ContourPlot[Cos[*x*] + Cos[*y*], *{x, 0, 4 Pi}, {y, 0, 4 Pi}*] はヘルプに載っている例である。
- 密度図は DensityPlot[*equation,{xrange},{yrange}*] とする。オプションは、*PlotPoints*->で点の数（例えば {100,50}と指定）、*Mesh*->False でメッシュを消してなめらかに表現する。*PlotRange*->–0,4″は範囲指定。*AspectRation*->Automatic とすれば比率を自動調整する。
- リストのプロットは ListPlot[*list, PlotJoined*->True] などとする。
- 媒介変数表示のプロットは ParametricPlot[*equations,ranges*] である。
- 離散プロットは DiscretePlot[*list,ranges*] である。

グラフの表示方法としては、以下の 3 つをよく使う。

- グラフを並べて表示する場合、GraphicsGrid[*plots*]
- グラフを横に並べて表示するときは GraphicsRow[*plots*]
- グラフを縦に並べる場合には、Column[*plots*]

他に、グラフを組み合わせて一つのグラフとして表示する Show[*plots*] という関数もある。例えば（ヘルプに書いてある例だが）Show[Plot[*x^2, {x, 0, 3.5}*], ListPlot[*{1, 4, 9}*]] は曲線と点列を一つのグラフに表示する。

❷ 3次元グラフ

- 三次元プロットの基本は Plot3D[] である。結果は画面に表示するだけでも画像にしてセーブするのでも良いが、POV 形式で Export[] し、PovRay というレイトレーシングソフトで描かせると格段に綺麗な画像が得られる。ただし POV ファイルを若干修正しないといけないことがある。
- 3 次元版の媒介変数表示プロットとしては ParametricPlot3D[*equations, ranges*] がある, *PlotPoints*->{40,80}や *BoxRarios*->{1,1,0.3}というオプションが使える。
- リストの 3 次元プロットは ListPlot3D[*list, PlotJoined*->True] などとする。
- その他、球面座標上へのプロットは SphericalPlot3D[]、円筒座標系へのプロットは RevolutionPlot3D[]、3D 版のコンター図は ContourPlot3D[] がある。それらに式ではなくリスト形式でデータを与える関数もあり、表面を記述するリストで与える 3D プロットは ListSurfacePlot3D[]、等高線リストで与える 3D コンター図は ListContourPlot3D[] である。

❸ その他

- 定義された幾何形状として、Cuboid（直方体）、Sphere（球）、Cylinder （円筒）、Point （点）、Polygon （ポリゴン）等がある。
- 複雑な形状をネットからダウンロードする方法もある。ExampleData/@Take[ExampleData[*"Geometry3D"*],*16*] とすると、データセンターからサンプルをダウンロードできる。直接 PNG にセーブする場合は、Export[*"exampledata.png"*,Column[*ExampleData/@*Take[ExampleData[*"Geometry3D"*],*2*]],*"PNG"*] などとする。
- その他形状変換として例えば GeometricTransformation[]（幾何学変換を施す）や、ShearingTransform[]（図形を剪断（せんだん）する）等があるが詳細は割愛する。

❹ 3D グラフ指定関数

指定についてのいくつかの関数がある。

- Specularity[] は鏡面性を指定する。例えば Specularity[{White,10}] などとする。
- Glow[]：物体の表面色を指定する。
- Scale[]：拡大率を指定する
- Translate[] ：平行移動を指定する
- Rotate[] ：回転を指定する

❺ 3D グラフオプション

3D グラフには様々なオプションがあるので以下に別項立ててまとめる。

- *Axes*->：軸を表示するかどうか。None とするとなめらかになる。
- *AxesEdge*->：座標軸の出現場所 {1,-1}などのペア 3 つで指定する。
- *BoundaryStyle*->：縁や境界のスタイルを決める。Directive[*Black*, Thickness[*0.0123*]] とすれば境界を強調することになる。
- *Boxed*-> ：枠の有無を指定する。True か False で与える。
- *BoxRatios*->：枠の比率を指定する。例えば{1,2,1}など。
- *BoxStyle*->：縁のスタイルを指定する。
- *ColorFunction*->：色についての指定を行う。よく使うのは（本誌表紙でも使ったが）Function[*{x, y, z}*, Glow[RGBColor[*z, z, z*]]] 等である。
- *PlotLabel*->：ラベルを指定する。
- *Imagesize*->：画像のサイズを指定。{500,500}で 500×500

のサイズの画像が出てくる。印刷媒体に画像を出す場合にはこれをかなり大きくする必要がある。

- *InterpolationOrder->*：リスト系のプロット関数で点の間を補完する次数の指定。0 は補完なし、1 は直線、2 は曲線で補完。
- *FaceGrids->*：対象を囲む直方体表面上に書かれる格子線の有無を指定。
- *FaceGridsStyle->*：対象を囲む直方体表面上に書かれる格子線のスタイルを指定。
- *Lighting->*:光源色を指定する。Automatic、Neutral、None という値をとれる。具体的には{"Point", Red, {0,0,2}} とか、{"Ambient", Green}とか{"Directional", Blue, {0,0,1},{-1,1,1}}と指定する。
- *Mesh->* ：None にすればメッシュを表示しない。
- *Opacity->* ：不透明度を指定する
- *RegionFunction->*：描画領域を制限する。例えば Function[*{T,F}*,Sin[*T F*]<*0.3*] と設定する。
- *ViewAngle->*：視点角度を指定。
- *ViewCenter->* 視点中心を指定。0.5,0.5,0.5 がデフォルトである。
- *ViewPoint->*：視点を指定。三次元リストでも指定できるが、Top や Front といった記号でも指定できる。
- *ViewRange->*：視点からの距離の範囲を指定する。
- *ViewVector->*：視点の位置と方向をまとめて指定する。
- *ViewVertical->*：どの方向が上になるかを 3 次元座標で指定する。

❻ 解析学

- 極限は Limit[] 関数を使う。
  例えば Limit[*(x-2)/(x-1),x->0*]。
- ベキ級数展開は Series[] である。Series[Sin[*x*],*{x,0,10}*] などとする。結果はオーダー項を含むので、Normal[] をかけてオーダー項をとれば普通の式にできる。
- 与えられた数列がベキ展開したときの係数になるような関数を求める関数もある。FindGenerationgFunction[*{1,3,5,7,9},x*] などとする。
- 与えられた数列を生成するような関数を求める関数もある。FindSequenceFunction[*{1,3,5,7,9},x*] などとするが、うまくいかない場合は多い。カオス時系列などは無理である。
- 逆関数の展開は InverseSeries[*equation*] とする。
- 微分は D[] 関数を使う。D[Sin[*x*],*x*]、あるいは Sin'[x] でも動作する。ちなみに二階微分は D[Sin[*x*],*{x,2}*] と指定すればよい。
- 積分は Integrate[*equation,x*] である。これは不定積分を求めるが、特に定積分は変数のところに範囲を指定し、例えば Integrate[*1/x,{x,1,Infinity}*] とする。記号的処理をスキップして最初から数値積分するには NIntegrate[] 関数を使う。詳細な条件を明示的に与えるには、定義域の後に *Assumptions->*オプションを加える。
- 微分方程式を解かせるには例えば DSolve[*m y''[t]+k y[t]==0, y[t],t*] などとする。
- いくつかの積分変換関数がある。
  - ▷ ラプラス変換は LaplaceTransform[*equation,t,s*]。
  - ▷ その逆変換は InverseLaplaceTransform[*equation,s,t*]。
  - ▷ フーリエ変換は FourierTransform[*equation,t,s*]。
  - ▷ その逆変換は InverseFourierTransform[*equation,s,t*]。

  どれも多次元の場合は s や t のところがリストになる。
  - ▷ Z 変換は ZTransform[*equation,n,z*]。
  - ▷ その逆変換は InverseZTransform[*equation,z,n*]。

  ガボール変換、ヒルベルト変換などはこれらを組み合わせて頑張ることになる。

❼ 整数論

整数論については非常に多くの関数がありとても列挙しきれない。以下は作者の興味で選んであるので文句が多々あるかもしれない。

❶ 初等整数論

以下は基本的な整数論の関数である。

- 自然数 n を素因数分解するには FactorInteger[*n*] を使う。結果は基数とべきの組み合わせのリストで返る。
- 剰余は Mod[*k,n*] で k/n のあまりを返す。商は Quotient[*k,n*] で k/n の商が求められる。
- Divisible[*k,n*] は k が n で割り切れるかどうかを判定する。
- Divisors[*n*] は n の約数をリストで返す。
- CoprimeQ[*n1, n2, n3*] 与えた n_1, n_2, n_3 が互いに素であるかどうかを判定。何個の n を与えても良い。
- GCD[*n1, n2, n3*] は最大公約数を返す。LCM[*n1, n2, n3*] は最小公倍数を返す。
- PrimitiveRoot[*n*] は n の最小の原始根を返す。原始根のリストは、PrimitiveRootList[*n*] が与える。

❷ 素数

- n 番目の素数を出す関数は Prime[*n*] である。これは非常に速い。Prime[*99999999999*] が 2 秒程度で 2760727302497 と出てくる。
- n 以下の素数の数をカウントする関数は PrimePi[] である。素数定理に関係する。これも驚異的に速く、PrimePi[*99999999999*] は 1 秒もかからず 4118054813 と出てくる。
- 素数かどうかを判定するには PrimQ[*number*] を使う。素数のべき乗かどうかは PrimePowerQ[*number*] で調べる。
- 自然数 n を素因数分解して、含まれる素数の種類を返すのが PrimeNu[*n*] であり、重複を数えた素因数の個数を計算するのが PrimeOmega[*n*] である。

❸ 数論関数

いくつかの有名な数論関数が使える。

- メビウス関数 $\mu(n)$：MoebiusMu[*n*]
- リウヴィル関数 $\lambda(n)$：LiouvilleLambda[*n*]
- ヤコビの記号 (n/m)：JacobiSymbol[*n,m*]
- クロネッカーのデルタ：KroneckerDelta[*n1,n2,n3*]
- カーマイケル関数 $\lambda(n)$：CarmichaelLambda[*n*]
- オイラーの $\phi(n)$ 関数：EulerPhi[*n*]

❹ 再帰

再帰を数論に入れるべきかは悩ましいが、他に入れようがないのでここに書く。

- a_n についての漸化式の一般項を出すのは RSolve[*{equation,init},a[n],n*] を使う。実例を見たほうがよい。
  RSolve[*{a[n+1]==2*a[n]+1, a[0]==1},a[n],n*] を実行すると、
  a[n]->-1+2^(1+n) という答えが出る。これが一般項である。
- 一般項のでないカオス時系列などは RecurrenceTable[] 関数を使って点列を生成させることもできる。例えば、
  RecurrenceTable[*{a[n+1]==3*a[n],a[1]==2},a,{n, 1, 10}*] とすると結果のリストが得られる。
- その他、Fibonacci[*n*] は n 番目のフィボナッチ数を出力する[*23]。LucasL[*n*] は n 番目のリュカ数を計算する。

❽ 拡張

❶ 拡張パッケージ

Mathematica はパッケージによって機能を拡張できる[*24]。拡張のためには、Needs[**パッケージ名** ‘] と打ち込む。どのようなパッケージがあるのかをネットのヘルプで探すのは難しいが「Wolfram システム標準パッケージ」というワードで検索すると出てくる。以下は Mathematica10 で内部に取り込まれていないパッケージとロードの方法である。

分散分析 .....Needs[*"ANOVA‘"*]
階層的クラスタ .....Needs[*"HierarchicalClustering‘"*]
仮説検定 .....Needs[*"HypothesisTesting‘"*]
多変量統計 .....Needs[*"MultivariateStatistics‘"*]
統計プロット .....Needs[*"StatisticalPlots‘"*]
エラーバープロット .....Needs[*"ErrorBarPlots‘"*]
方程式トレッカー .....Needs[*"EquationTrekker‘"*]
フーリエ級数 .....Needs[*"FourierSeries‘"*]
関数近似 .....Needs[*"FunctionApproximations‘"*]
数値計算 .....Needs[*"NumericalCalculus‘"*]
微分方程式の数値解析 Needs[*"NumericalDifferentialEquationAnalysis‘"*]
変分法 .....Needs[*"VariationalMethods‘"*]
有限体 .....Needs[*"FiniteFields‘"*]
四元数 .....Needs[*"Quaternions‘"*]
多面体操作 .....Needs[*"PolyhedronOperations‘"*]
多胞体 .....Needs[*"Polytopes‘"*]
コンピュータ演算 .....Needs[*"ComputerArithmetic‘"*]
素数証明 .....Needs[*"PrimalityProving‘"*]
オーディオ .....Needs[*"Audio‘"*]
ミュージック .....Needs[*"Music‘"*]
黒体輻射 .....Needs[*"BlackBodyRadiation‘"*]
共鳴吸収線 .....Needs[*"ResonanceAbsorptionLines‘"*]
標準大気 .....Needs[*"StandardAtmosphere‘"*]
ベンチマーク .....Needs[*"Benchmarking‘"*]
開発者ユーティリティ .....Needs[*"Developer‘"*]
試験的関数 .....Needs[*"Experimental‘"*]
表記法 .....Needs[*"Notation‘"*]
XML .....Needs[*"XML‘"*]

各パッケージの中にどのような関数があるかはヘルプを参照のこと。ベンチマークなどは特に簡単だ。ベンチマークパッケージをロードしてから Benchmark[] を実行するだけでマシンスコアが表示される。素数証明も結構面白い。パッケージをロードしてから、例えば PrimeQCertificate[*3837523*] とすると（これは合成数である）、{2, 3837522, 3837523}というリストで答えが返る。素数ならもっと複雑なリストが返ってくる。これを引数にしてそのまま PowerMod[] 関数に渡して 1 になれば素数である。具体的には Apply[*PowerMod,%*] とすると 3134697 になり 1 ではないので合成数だ。PowerMod[*2,p-1,p*] は p を法として 2^{p-1} を与えるが、p が素数ならフェルマーの小定理 ($2^{p-1} \equiv 1 \mod p$) が成り立つので結果が 1 にならなくてはいけない。

❷ データベース

いくつかの関数は、ウルフラム・リサーチ社のデータベースをネット越しに自動検索して表示する。どのような種類のデータがあるかは、
./SystemFiles/FrontEnd/SystemResources/FunctionalFrequency/*trie にバイナリ形式で格納されている[*25]。少なくともファイル名を見れば、AstronomicalData、CalendarData、ChemicalData、CityData、ColorData、CountryData、ElementData、FinancialData、GalaxyData、GraphData、ImportFormats、KnotData、StarData、ThermodynamicData、WeatherData、WordData 等があることが分かる。ここでは使い方の例をいくつか示す。これらは実際に使うときに自動的にデータベースに接続するが、ダウンロードの時間があるので完了までに時間がかかる。

例えば C_2H_6O の構造が知りたいとする。そのためにはまず、ChemicalData[*"C2H6O"*] と実行する。するとデータベースに問い合わせに行き、{Entity[*Chemical, Ethanol*],

[*23] Prime[] に比べてこちらはそれほど速くない。
[*24] 起動時にもっているパッケージは Print[*$Packages*] によって表示できる。パッケージはユーザが自分で作ることもできるが詳細は割愛する。
[*25] そのフォーマットは明らかではない。

Entity[*Chemical, DimethylEther*]}というリストが返ってくる。この化学式から推定される物質が二つあるからだ。ここではエタノールを見ることにすると、それは要素の一番目だから、ChemicalData[*"C2H6O"*][[1]] がエタノールの Entity 構造体を指すことになる。この構造体の中には様々な情報が格納されているが、もしここで構造図を知りたいのなら、構造体の要素を取り出す EntityValue[] 関数に MoleculePlot を指定する。画像として取り出すことを考えると、結局、

```
Export["C2H6O.png",EntityValue[ChemicalData["C2H6O"][[1]],"MoleculePlot"]]
```

でエタノールの構造が画像で保存できる。

CityData も似たようなものである。CityData[*"Tokyo","Population"*] と打つと、速やかに 8476919 people という答えが返ってくる。データベースにはかなり細かい都市データが入っており、Kashiwa とか Asahikawa も出てくる。引数に何も与えなければ一覧が出てくる。FinancialData[*"JPY/USD"*] とすると、為替レートが出るが、これを cron などで自動起動すると時系列が取得できるだろう。他にも組み込み多面体の一覧を出すには PolyhedronData[*"Properties"*]、組み込みグラフは GraphData[]、組み込みの結び目データは KnotData[]、組み込みの格子データは LatticeData[] を使うと得られる。これらは例えば Export[PolyhedronData[*"Cube","NetImage"*]]（展開図）とか、PolyhedronData[*"Cube","SkeletonImage"*]（連結グラフの生成）などとして使う。データベースに入っているグラフを隣接行列には GraphData[*"CubicalGraph", "AdjacencyMatrix"*] などとする。

■関数名索引

マセマティカ
Mathematicaのメモ

2014 年 11 月 15 日 初版発行
2015 年 7 月 12 日 PDF 版公開
2016 年 6 月 25 日 PDF 版修正

著 者 シ (し)
発行者 星野 香奈 (ほしの かな)
発行所 同人集合 暗黒通信団 (http://www.mikaka.org/~kana/)

初版頒布完了したので PDF 公開です。