情報処理 II No.9

Mathematica 入門 (1)

桂田 祐史

1995 年 6 月 23 日

今日から 3 回かけて、代表的な**数式処理系** footnote 他の数式処理系としては REDUCE, Maple などが有名です。である Mathematica を体験してもらいます。

今後の予定　3 回あるわけですが、大体

1. 高級電卓としての利用, グラフ描画機能の利用
2. プログラミング入門
3. プログラミングの実際 (課題へのチャレンジ)

という感じでやります。どういうことが出来るのか雰囲気をつかんでもらい、今後の学習・研究のヒントにしてもらいたい、と思っています。

1　数式処理について

計算機言語の中には、数値や文字だけでなく、数式をデータとして扱うことの出来る「数式処理言語」と呼ばれるものがあります。

Fortran のような言語は、プログラムの中では「数式」を書けますが、 `read` 文や `write` 文で入出力可能なデータは数や文字列だけで、例えば `-2/5` のような分数式の入力は出来ません。また (気がついてくれていたら嬉しいのですが)、グラフを描くプログラムを作る場合に、範囲や、分割数の指定等は実行時に入力出来ても、グラフを描こうとしている関数自体は (普通の方法では) 入力できず、プログラムの中に `function` 副プログラム等として埋め込むしかなかったわけです。そういう意味では Fortran は不自由な言語であると言えます[1]。

(原理的には一つの計算機言語があれば、どんな計算でも出来るはずなのですが、実際的な意味で万能の計算機言語と呼べるものは存在せず、適材適所を心がけることが重要です。みなさんも、あまり一つの言語、一つのシステムにこだわらずに、機会があったら色々なものを勉強してみましょう。)

2　Mathematica 実行例

以下の例は数学科の計算機である “*hostname*” (仮名[2]) にログインして実行してみたものです。情報処理 II 受講中は（こちらで登録した人は）情報科学センターのワークステーションからは、“`xhost` *hostname* `; rlogin` *hostname* `-l math`” とすれば *hostname* にログインすることが

[1] もちろん不自由さを補って余りある大きな利点があるから現在でも盛んに使われているわけです。

[2] ある理由から名前は後で知らせます。

出来ます。以下ではプログラムなどは書かずに、順次式を入力して計算結果を表示させています (BASIC でいうならダイレクト・モードでの実行に相当します)。

```
hostname% mkdir ee480??
hostname% cd ee480??
hostname% math
Mathematica 2.0 for SPARC
Copyright 1988-91 Wolfram Research, Inc.
 -- X11 windows graphics initialized --

In[1]:= 1/2 + 1/3                    ← 分数計算

        5
Out[1]= -                            → ちょっと見難いですけどね
        6

In[2]:= a={{0,1},{6,1}}              ← 行列の入力

Out[2]= {{0, 1}, {6, 1}}

In[3]:= Eigenvalues[a]               ← 行列の固有値の計算

Out[3]= {-2, 3}

In[4]:= Eigenvectors[a]              ← 行列の固有ベクトルの計算

            1        1
Out[4]= {{-(-), 1}, {-, 1}}
            2        3

In[5]:= Expand[(x+y)^6]              ← 式の展開

         6      5         4  2       3  3       2  4        5    6
Out[5]= x  + 6 x  y + 15 x  y  + 20 x  y  + 15 x  y  + 6 x y  + y

In[6]:= N[Pi,50]                     ← 円周率 50 桁

Out[6]= 3.14159265358979323846264338327950288419716939937510

In[7]:= Integrate[Log[x],x]          ← 不定積分

Out[7]= -x + x Log[x]

In[8]:= Plot3D[x^2 - y^2, {x,-1,1}, {y,-1,1}]  ← グラフ

Out[8]= -Graphics-                   → ここで画面に図が表示されます
```

```
In[9]:= Quit                              ← 終了
hostname%
```

3 基本的な使い方

3.1 Mathmatica の起動・終了

1. センターからは “`xhost` *hostname* ; `rlogin` *hostname* -l math” で *hostname* にログインする[3]。

2. 情報処理 II 受講中は

   ```
   mkdir ee48000
   cd ee48000
   ```

   のように、自分のユーザー名のディレクトリィを作って、そこで作業して下さい。

3. 起動は `math` コマンド。

4. 終了するには `Quit` コマンド（先頭が大文字であることに注意）。

NEmacs の shell buffer の中で使うのがお勧めです。でも授業中はマシンへの負荷が心配なので遠慮して下さい。

3.2 操作の説明

“`In[番号]:=` ” というプロンプトに対してコマンドを入力。最後にリターン。
“`Out[番号]=` ” に続いて結果が出る。
入力に関して便利な記号として

- `%`　直前の結果
- `%%%···%`（k 回）　k 回前の結果
- `%` 数　Out[数]

“;” でマルチステートメントを作れる。結果を表示しない。
NEmacs の中で実行しているのでない場合は、以下のコマンドが便利でしょう。

- `Edit[]`（環境変数 EDITOR で指定したコマンドを呼び出す）
- `Edit[式]`（式を変更するためにエディターを呼び出す）
- `EditIn[]`（直前の `In[]` の編集）, `EditIn[番号]`（`In[数]` の編集）、
- `EditDefinition[f]` 関数の `f` の定義の編集

[3]情報処理 II が終了したら、”`-l math`” は不要です。

3.3 文法

かっこ

- ( ) は結合順位をグループ化する普通のかっこ。
- [ ] は関数の引数を示す。
- { } はリスト。
- [[ ]] は要素のアクセス。

変数（名前）の宣言、変数への代入

- 組み込みの名前は先頭が大文字。ユーザーはなるべく小文字を使おう（衝突を回避するため）。
- a= 式, a=b= 式, 等で代入。
- a=. または Clear[a] で解放（内容を除く）。
- Remove[a] 名前まで込めて完璧に除く。

演算子

- 四則演算は普通の記号で OK。 a+b, a-b, a*b または a b （ab は一つの名前。 5a は 5 かける a。）, a/b.
- べき乗は a^b.

組み込み関数 Sqrt[x], Exp[x], Log[x], Log[b,x], Sin[x], Cos[x], Tan[x], ArcSin[x], ArcCos[x], ArcTan[x], Abs[x], Round[x], Random[], Max[x,y,...,z], Min[x,y,...,z], 等々。 [] であることに注意。またシステム組み込みのものは頭文字が大文字。 “? 文字列 *” で一覧表が出る。さらに “? 関数名”、 “?? 関数名”、で説明が出る。

無限多倍長整数（小数も任意精度まで） 特に意識せずに使うことが出来る。

```
100!, 2^100
```

などで試してみよう。

分数

```
1/2+1/3
```

のように分数が入力でき、分数のまま計算できる。

文字式（多項式、分数式、その他）

```
p1=Expand[(1+x)^10], 2 f[x] + 3 f[x], p2 = (1+x)^3,
PolynomialQuotient[p1,p2,x], PolynomialRemainder[p1,p2,x],
PolynomialGCD[p1,p2]
```

定数　Pi(円周率 π), E(自然対数の底 e), Degree($=180/\pi$), I(虚数単位 $i=\sqrt{-1}$), Infinity (無限大 $+\infty$), ComplexInfinity(複素平面の無限大) などの定数があらかじめ定義されている。

ルートなど　ルートなどのシンボリックな数も使える。(1+Sqrt[3])^2 など。

近似値の計算

```
N[%,100], N[Pi,100], N[E,100], 2^100 //N, N[Sin[30 Degree]]
```

整数（素因数分解、素数判定）

```
FactorInteger[12345], Mod[n,m], PrimeQ[21]
```

複素数

```
(3 + 4I) (1 + 2I), Re[%], Im[%%], Conjugate[%%%], Abs[%], Arg[%%]
```

リスト　集合、ベクトルや行列として扱える。それだけではないが。

```
{1,2,3}+{a,b,c}, 2 {1 3 5}, {1 3 5} / 3, {1 2 3} . {3 4 5},
{1 2 3} {2 3 4}, {1 2 3} / {2 3 4},
list = {1 2 3},
Part[list,i] または list[[i]]  ... これらは左辺値にもなることに注意
Log[{a,b,c}], N[{1/2,1/3,1/4}],
First[], Rest[] ... Lisp 風
Union[],Intersection[],Complement[] ... 集合算
A={{a11,a12,a13},{a21,a22,a23},{a31,a32,a33}}, y={y1,y2,y3}, A.y
Transpose[],Inverse[],Eigenvalues[],Eigenvectors[]
Length[{a,b,c,d,e}], MemberQ[{a,b,c,d,e,f},a], Count[{a,b,a,b,a,b},a],
Reverse[], Sort[], RotateLeft[], RotateRight[],
```

代入（置き換え）　“/. 名前 -> 値”, “/. {名前 -> 値, 名前 -> 値,...}” で置き変えをすることが出来る。

例. (x + y + z + w)^2 の後 % /. {y->1,z->1}

ユーザー定義関数　(詳しくは次週)

名前 [変数名 _]:= 式
名前 [変数名 _]:= 式 /; 条件式
名前 [変数名 _ 型名]:= 式

方程式

```
Solve[式 == 式, 未知数]
```

繰り返し　(詳しくは次週)

```
Do[Print[CoefficientList[Expand[(1+b)^n],b]],{n,10}]
```

微分

```
D[x^n,x]
```

不定積分、定積分

```
Integrate[式、変数]
Integrate[x^2,{x,0,1}]
Integrate[E^(-x^2),{x,0,Infinity}]
NIntegrate[Sqrt[Sin[x]], {x,1,2}]
NIntegrate[Sqrt[Sin[x]], {x,1,2}, WorkingPrecision->50,AccuracyGoal->40]
```

級数

```
Series[Exp[x],{x,0,10}]

Sum[n,{n,1,5}]
```

ユーザーが関数を定義する方法や、繰り返し、条件判断等いわゆるプログラミング用の機能は次回に詳しく説明する。

3.4 ファイル入出力について

ファイルからの入力“ << ファイル名” ファイルの内容をキーボードからの打鍵の代わりに入力

ファイルへの出力“ 式 >> ファイル名” 式の値をファイルに出力

ファイルへの追加出力“ 式 >>> ファイル名” 式の値をファイルに追加出力（(2) の場合は古い内容は上書きされてしまう。）

ファイルの内容表示“ !! ファイル名”

ファイルへの出力“ Save["ファイル名 ", f,g,...]”

ファイルは数学科のマシンに保存されることになります。後の始末のことを考えて、不必要なファイルは残さないで下さい。

4 Mathematica のグラフィックス

Mathematica の特徴の一つであるグラフィックス機能を使ってみる。

グラフィックスは結構メモリーを消費するので、人数が多い場合は不要なウィンドウは早目に消すなどを心掛ける。

4.1 グラフィックス用の関数について概説

2 次元グラフ Plot（関数のグラフ）
ParametricPlot（パラメーター曲線の描画）
ListPlot（リスト・データのプロット）

3 次元グラフ　　Plot3D（2 変数関数のグラフ）
ParametricPlot3D（パラメーター曲線・曲面の描画）
ListPlot3D

等高線グラフ　　ContourPlot（2 変数関数の等高線）
ListContourPlot

濃淡グラフ　　DensityPlot（2 変数関数の濃淡図）
ListDensityPlot

これらの関数では、対応するグラフィックス・オブジェクトを生成して、画面に表示する。グラフィックス・オブジェクトは変数に代入可能である。オブジェクトを表示するには関数 Show[] を用いる。またオブジェクトがどう表現されているかは InputForm[] で読むことが出来る。

例. 代入しておいて、後から再表示

```
g1 = Plot[Sin[x],{x,0,2Pi}]
g2 = Plot[Cos[x],{x,0,2Pi}]
Show[g1]
Show[g1,g2]
InputForm[g1]
```

引数として取るものは、順に

1. 関数あるいは関数のリスト or 数値データによるリスト
2. 描画する範囲
3. 描画を制御するオプション

このうちオプションはその名前の通り省略可能である。
（オプションとは？実際にどうやって描画するかについて、自由度があり、それを一々指定するのはとても面倒なので、普段はそれを適当に決めているが、そのデフォールト値で満足できない場合に、ユーザーが自分の求めるものを指定することが出来るようにしたもの。）
オプションの指定の仕方は、”オプション名 -> 値” である。例えば

```
AspectRatio -> 数値              デフォールト値は 1/GoldenRatio
Axes -> 真偽値
AxesLabel -> {"x", "y=f(x)"}
Compiled -> False                デフォールト値は True
Frame -> True                    デフォールト値は False
GridLines -> Automatic
PlotRange -> {zmin,zmax}
```

関数がどういうオプションを持っているか、見たい場合は Options[] を用いる。

```
Options[Plot]                    Plot のオプション全部を表示
Options[Plot, PlotRange]         Plot の PlotRange オプションを表示
```

オプションを変更するには “SetOptions[]” を用いる。
プリンターに印刷するには、

```
Display["!psfix > ファイル名 ", オブジェクト]
```

として、 PostScript 形式のデータとしてセーブしてから、 lpr コマンドを使う。

２次元グラフ

Plot[]

```
Plot[関数, 描画範囲を示すリスト]
Plot[{関数 1, 関数 2,..., 関数 n}, 描画範囲を示すリスト]
```

描画範囲を示すリストから、関数値を計算するための変数の適当な値を選びだして、そこでの関数値を評価して、そうして得た点を順に結んでグラフを描く。

```
Plot[Sin[x], {x,0,2Pi}]
Plot[{Sin[x], Sin[2x], Sin[3x]}, {x,0,2Pi}]
```

ホントの関数でなく、関数のテーブル等を生成するコマンド等の場合は、先に Evaluate するのが良い。

```
Plot[Evaluate[Table[BesselJ[n,x], {n,4}]], {x,0,10}]
```

この Table[] については後述。

```
NDSolve[{y'[x]==Sin[y[x]],y[0]==1}, y, {x,0,4}]
Plot[Evaluate[y[x] /. %], {x,0,4}]
```

(この NDSolve[] は数値的に微分方程式を解くコマンドであり、結果は離散的な点での関数値を近似的に求めたものである。計算していない点での値は補間により簡略計算する。こういうものを InterpolatingFunction という。)

```
y[1.5] /. %%
```

ParametricPlot[関数 x, 関数 y, t,tmin,tmax]　いわゆるパラメーター表示された平面曲線を描く

```
ParametricPlot[{Sin[t], Sin[2t]}, {t,0,2Pi}]
```

ListPlot[]　数値データのリストから曲線を描く。これを用いると、外部のプログラムで計算したデータを表示できる。

```
fp = Table[{t,N[Sin[Pi t]]}, {t,0,0.5,0.025}]
ListPlot[fp]
ListPlot[fp, PlotJoined -> True]
```

２変数関数の等高線、濃淡図

```
ContourPlot[f, {x,xmin,xmax}, {y,ymin,ymax}]
DensityPlot[f, {x,xmin,xmax}, {y,ymin,ymax}]

ContourPlot[Sin[x]Sin[y], {x,-2,2}, {y,-2,2}]
```

オプションとして

```
Contours -> 数
PlotRange -> {zmin,zmax} または Automatic
PlotPoints -> 数 （デフォールトが 15。小さい！非力さが出て来る。）
ContourShading -> False （デフォールトは True）
```

特にモノクロ・ディスプレイでは、 ContourShading を False にした方が良い。関数 ContourPlot 使用時に常に False にしたい場合は

```
SetOptions[ContourPlot, ContourShading->False]
```

３次元グラフィックス

```
Plot3D[f, {x,xmin,xmax}, {y,ymin,ymax}]
Plot3D[Sin[x y], {x,0,3}, {y,0,3}]
```

オプション

```
HiddenSurface -> False          隠面消去をしない
PlotPoints -> 個数
ViewPoint -> {x,y,z}            視点の指定
```

ParametricPlot3D[], ListPlot3D[] 等もある。以下の例のうち曲面を描くするものはモノクロのディスプレイではうまく表示できない。

```
ParametricPlot3D[{Sin[t], Cos[t], t/3}, {t, 0, 15}]
ParametricPlot3D[{t,u,Sin[t u]}, {t,0,3}, {u,0,3}]
ParametricPlot3D[{Sin[t],Cos[t],u]}, {t,0,2Pi}, {u,0,4}]
ParametricPlot3D[{Cos[t](3+Cos[u]),Sin[t](3+Cos[u]),Sin[u]},
                 {t,0,2Pi}, {u,0,2Pi}]
ParametricPlot3D[{Cos[t]Cos[u],Sin[t]Cos[u],Sin[u]},
                 {t,0,2Pi}, {u,-Pi/2,Pi/2}]
```

5 第二回レポートについて

No.5 〜 No.8 までの各回の問題から最低一問ずつ、なるべくたくさん解きましょう（目安としては、平均一回二問程度）。内容的に関係あれば、自分で考えた問題を解いたもの受け付けます。

〆切は 6 月 30 日 (金) の午後 5 時です。提出先は、まず桂田 (普段は 6716B や 6701 号室にいることが多い) を探し、不在の場合は数学科資料室 (6606 号室) の野町さん or 中里さんに預けて下さい。

最終回 (7 月 7 日) の授業中に、第一回、第二回のレポートを返却する予定です。

レポート作成上の注意 以前書いたことの繰り返しです。

- プログラム・リストに名前を添えただけのものではなく、きちんとした報告書になっていることを要求します。
- レポートには、学年、番号、氏名、ユーザー名、解いた問題の番号を明記して下さい。

- 自分でプログラムを書いた場合はプログラムのリスト、実験した場合は実行結果のプリント・アウト、また実行結果を得るのに用いた入力データ (もしあれば) を添えて下さい。他人がレポートを読んで、結果を再現出来るだけの情報をまとめて下さい。
- 何を解いたのか、プログラムではなく文章で、はっきり説明して下さい。例えば関数のグラフを描いた場合は、どの関数の、どこからどこまでの範囲のグラフを描いたのかを明記して下さい。レポートを読む人にプログラムを解読することを強制しないように。
- どうやって解いたのかを説明して下さい。自分でプログラムを一から書きあげた場合はもちろん、例題プログラムを修正した場合も、どこをどのように修正したのか書いて下さい。
- 必要な場合は実行結果の分析もきちんと書いて下さい。

6 参考書

1. Wolfram, Mathematica （これはマニュアル）, Addison Wesley
2. 小池慎一、 Mathematica 数式処理入門、技術評論社
3. I. ヴァルディ、 Mathematica：計算の愉しみ、トッパン
4. R.E. クランダール、 Mathematica：理工系ツールとしての、トッパン
5. R. ローダー、 Mathematica プログラミング技法、トッパン
6. S. スキエナ、 Mathematica: 離散数学とグラフ理論、トッパン

それぞれ目的としているところが違いますし、出来も玉石混交です（特に翻訳のマズイのもある）。 [2] は簡単な日本語マニュアル代わりになるかもしれませんが、最近では [1] の翻訳も出ていた。 [5] はプログラムの書き方をしっかり説明してあります（良い本だと思います）。 [6] は応用ですが、信頼すべき筋によればきちんと書けていて面白いそうです（専門外なので私には判定出来ません）。