TR - H - 143

Mathematica 入門

玉川 浩

1995. 4. 17

ATR人間情報通信研究所

〒619-02 京都府相楽郡精華町光台2-2 ☎ 0774-95-1011

ATR Human Information Processing Research Laboratories

2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Telephone: +81-774-95-1011

Facsimile: +81-774-95-1008

目次

1 はじめに 1

2 起動と終了 3
2.1 環境設定 . . . 3
2.2 起動 . . . 3
2.3 終了 . . . 3
2.4 ヘルプ . . . 3

3 データ処理の基礎 4
3.1 数値演算 . . . 4
3.2 主な数学関数 . . . 6
3.3 配列(リスト)の操作 . . . 6
3.4 行列演算 . . . 8
3.5 ファイル入出力 . . . 9
3.6 プログラミング . . . 10

4 データ処理の応用 13
4.1 信号処理 . . . 13
4.2 統計処理 . . . 15

5 数値データのグラフ表示 17
5.1 データファイルの読み込み . . . 17
5.2 1次元配列データのグラフ表示 . . . 17
5.3 2次元配列データのグラフ表示 . . . 18
5.4 2次元座標点の表示 . . . 20
5.5 3次元座標点の表示 . . . 21

6 関数のグラフ表示 23
6.1 1変数関数のグラフ表示 . . . 23
6.2 2変数関数のグラフ表示 . . . 23

7 複数のグラフ表示およびその他のグラフ表示 25
7.1 複数のグラフ表示 . . . 25
7.2 その他のグラフ表示 . . . 26
7.3 グラフの出力 . . . 27

8 グラフィックスの基礎 27

9 Q & A 28
グラフの目盛りを変えるには? . . . 28
ウィンドウの大きさを変えるには? . . . 29
グラフィックスを TeX に取り込むには? . . . 29
色が出ない時は? . . . 29

1 はじめに

本編は、データの取り込み－加工－視覚化という一連のデータ処理の作業を想定し、Mathematica を最小限の労力で習得するための入門書です。

Mathematica は数値演算、数式処理、強力なグラフィック機能を備えたツールです。本編では、Mathematica の数値データ処理の基本的な使い方を説明します。

使用するデータファイルは、アスキー(ascii)形式のテキストで、以下のように各フィールドがスペースもしくはタブで区切られ、各レコードは、改行で区切られたファイルを対象とします。

右記の例では、65 5657 -3573 が 1 つのレコードになります。またこのレコードの第 2 フィールドは 5657 です。対象となるデータが、複数のファイルにまたがっていたり、フィールドの数が異なる場合は、あらかじめ、UNIX のコマンド(head、tail、paste、nawk 等)で、フィールドの数をそろえ、1 つのファイルに結合してください[1]。

```
% cat wavedata.ascii
65   5657    -3573
47   5611    -2994
57   5547    -2414
21   5472    -1878
     ...
```

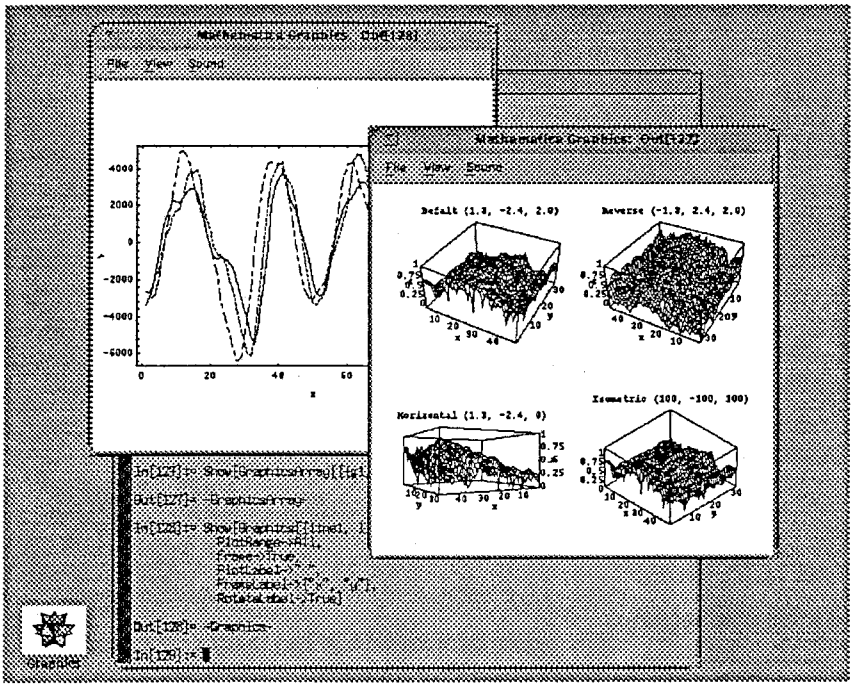


図 1 : Mathematica のメイン画面 (kterm 上) とグラフィック画面 (motifps)

[1]head は、指定された数だけファイルの先頭からレコードを取り出すコマンド、paste は、指定された複数のファイルのフィールドを結合するコマンドです。

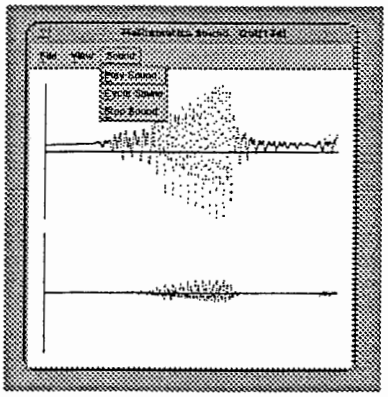


図2: サウンド画面 (motifps)

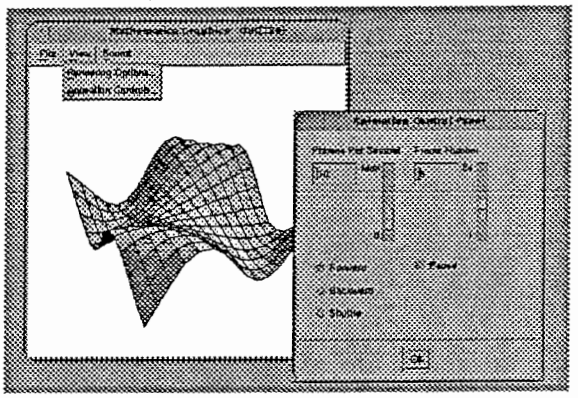


図2: アニメーション画面 (motifps)

*UNIX は UNIX Systems Laboratories,Inc. の登録商標です。
*X ウィンドウは Massachusetts Institute of Technology の登録商標です。
*Sun は Sun Microsystems,Inc. の商標です。
*OpenWindows は日本サン・マイクロシステムズ株式会社の商標です。
*Mathematica は Wolfram Research,Inc. の登録商標です。
*PostScript は Adobe Systems Incorporated の登録商標です。

2 起動と終了

2.1 環境設定

Mathematica を起動する前に、環境変数やパスを設定しておきます。

Mathematica 自身の実行ファイルがあるディレクトリを $path に入れておきます。

```
% set     path=($path /usr/cognition/math/bin)
```

Mathematica のための環境変数の設定は特に必要ありませんが、グラフィックスを描画するために、使用しているウィンドウの種類を、環境変数 $DISPLAY や $OPENWINHOME で識別しています。

グラフィックスが表示されない場合は、これらの環境変数がきちんと設定されているか、特に余分な環境変数が設定されていないか確かめてください。

2.2 起動

Mathematica を起動するには、**math** コマンドを入力します[2]。

Mathematica が起動されて、入力を促すプロンプトが表示されます。右記は Sun のワークステーション上の X ウィンドウから起動した時の表示例です。

Mathematica は、このプロンプトから入力されたコマンドを解釈し実行します。

```
% math
Mathematica 2.2 for SPARC
Copyright 1988-93 Wolfram Research, Inc.
 -- Motif graphics initialized --


In[1]:=
```

2.3 終了

Mathematica を終了するには、入力プロンプトのあとに **Exit** を入力します。

```
In[2] := Exit
%
```

2.4 ヘルプ

演算	Mathematicaでの表現	備考
ヘルプ	*?symbol-name* *??symbol-name*	

□ ヘルプ

? のあとにコマンド名を入力すると、コマンドに関する説明が表示されます。
?? の場合、オプションや属性も表示されます。

```
In[]:= ?Exit
Exit[ ] terminates a Mathematica session.

In[]:= ??Exit
Exit[ ] terminates a Mathematica session.

Attributes[Exit] = {Protected}
```

[2] その他に notebook 型の mathematica というコマンドも用意されています。こちらの方は、式の編集やグラフィックの表示などをマウスを使ってインタラクティブに操作することができます。本稿では、より基本的な math の方をとりあげて説明することにします。

3　データ処理の基礎

3.1　数値演算

演算	Mathematicaでの表現	備考
演算子	+ - / * ^	
式の区切り	;	
関数の呼出し	*func-name*[arg1, arg2, ...]	*ex.* Plus[3, 4]
式の値	N[*expr*]	または、*expr* //N

□ 式の入力と結果

式を入力して改行を行うと、Out[ ]=[3]の後に結果が表示されます。

```
In[]:= 3 + 2

Out[]= 5

In[]:= 3*(5 + 6)

Out[]= 33
```

式どうしを空白で並べてもかけ算を表します。

```
In[]:= 3 (5 + 6)

Out[]= 33
```

式をセミコロン (;) で区切って一行に並べることもできますが、最後の計算の結果だけが表示されます。

```
In[]:= 3 + 2; 3*(5 + 6)

Out[]= 33
```

式をセミコロン (;) で区切ると、結果は表示されません。

```
In[]:= 3 + 2;

In[]:=
```

□ 変数

a に値を代入し、a を用いて計算します。

```
In[]:= a = 6

Out[]= 6

In[]:= a + 7

Out[]= 13
```

"%" で直前の結果を参照できます。

```
In[]:= % + 7

Out[]= 20
```

[3]実際の表示は、[ ] の中に行番号が表示されますが、本編ではすべて省略しています。

□ 関数の呼出し

"+"と"*"は、それぞれPlus と Times という関数でも実行できます。関数の引数は中括弧([ ])で括り、引数と引数の間はカンマ (",") で区切ります。引数の前後には空白や改行を自由に入れることができます。関数は何重にも入れ子にできます。

```
In[]:= Times[3,Plus[5,6]]

Out[]= 33

In[]:= Times [ 3 , Plus [ 5 , 6 ] ]

Out[]= 33

In[]:= Times[3,
            Plus[5, 6]]

Out[]= 33
```

Mathematica の特徴は数式処理にあります。分数やルートの計算など式のままで計算を行います。

```
In[]:= 1 / 3 + 1

       4
Out[]= -
       3
```

このような式の値をみたい場合は、N[ ] 関数を使って数値化してやります。

```
In[]:= N[1 / 3 + 1]

Out[]= 1.33333
```

□ 定数

Mathematica で用意されている定数として、虚数単位 (I)、円周率 (Pi)、無限大 (Infinity) などがあります。

```
In[]:= Pi

Out[]= Pi

In[]:= N[Pi]

Out[]= 3.14159
```

□ 複素数

複素数を表すには I を使用します。

```
In[]:= 3 + 2*I

Out[]= 3 + 2 I
```

Mathematica における式の表現

Mathematica では式を表現するのに、前置式記法、後置式記法、中間式記法 (演算子を使う、関数名を使う) などの複数の方法で表現できます。

前置式記法	関数名 [引数1 , 引数2 ...]
後置式記法	引数 // 関数名
中間式記法 (関数名)	引数1 ~ 関数名 ~ 引数2 ...
中間式記法 (演算子)	引数1 op 引数2 ...
	op := +, -, *, /, ...

例えば、乗算は以下のような表現が可能です。

```
Times[3, 4]
3 ~Times~ 4
3 * 4   または   3 4
```

また1つだけ引数をとる関数、例えば N[ ] も

```
N[3/4]
3/4 //N
```

と表現でき、いずれも結果は同じになります。

本編では、一般の関数については関数名を使用した 前置式記法 で表現し、演算子を使用するのが一般的な演算についてのみ演算子を使用することにします。

3.2　主な数学関数

以下はよく使われる関数の一覧です。

演算	Mathematicaでの表現	備考
平方根	`Sqrt[x]`	$\sqrt{x}$
指数	`Exp[x]`	e^x
対数	`Log[x]`	$\log_e x$
	`Log[b, x]`	$\log_b x$
三角関数	`Sin[x]`, `Cos[x]`, `Tan[x]`, `ArcSin[x]`, `ArcCos[x]`, `ArcTan[x]`	$\sin x, \cos x, \tan x$, $\arcsin x, \arccos x, \arctan x$
階乗	$n!$	$n!$
絶対値	`Abs[x]`	$\|x\|$
実数部	`Re[x]`	$Re\{x\}$
虚数部	`Im[x]`	$Im\{x\}$
位相角	`Arg[x]`	
乱数	`Random[]`	0から1までの実数
	`Random[`*type*`,`*range*`]`	*type* は Integer, Real, Complex のいずれか
四捨五入	`Round[x]`	
剰余	`Mod[n, m]`	

□ 乱数の発生

0から100までの整数値を発生させます。

```
In[]:= Random[Integer, {0, 100}]

Out[] = 4
```

3.3　配列(リスト)の操作

演算	Mathematicaでの表現	備考
1次元配列の作成	`Table[`f`,{`$i, imin, imax, di$`}]`	$\{f_{i_0}, f_{i_1}, ...\}$
2次元配列の作成	`Table[`f`,{`$i, imin, imax$`}, {`$j, jmin, jmax$`}]`	$\{\{f_{i_0,j_0}, f_{i_0,j_1}, \ldots\}, \{f_{i_1,j_0}, \ldots\}\}$
配列の長さ	`Length[`*list*`]`	
配列の大きさ	`Dimensions[`*list*`]`	$\{row, col\}$
配列への要素の追加[4]	`Append[`*list*`,`*element*`]`	
配列への要素の挿入	`Insert[`*list*`,`*element*`,`*pos*`]`	
配列から要素の削除	`Delete[`*list*`,`*pos*`]`	
配列から要素の取り出し	`Take[`*list*`,{`n, m`}]`	$\{n, m\}$ は *list* の範囲を指定
配列の連結	`Join[`$list_1, list_2, ...$`]`	
配列の反転	`Reverse[`*list*`]`	

Mathematicaでは、多次元のデータをあらわすのに、大括弧 ({}) を使います。ベクトル(1次元配列)、行列(2次元配列)、リスト(より自由な構造をもつデータ)ともに扱い方は同じです。

□ 配列の作成

1次元の配列を作成します。

```
In[]:= Table[Sin[x], {x, 0, 2*Pi, 0.4}]

Out[]= {0, 0.389418, 0.717356, 0.932039, 0.999574, 0.909297, 0.675463,
>    0.334988, -0.0583741, -0.44252, -0.756802, -0.951602, -0.996165,
>    -0.883455, -0.631267, -0.279415}
```

関数 $\sin 3x$ の 0 から 2π まで 20 個のデータを作成します。N[ ] は数値化する関数です。

```
In[]:= N[Table[Sin[3*i], {i, 0, 2*Pi, Pi/10}]]

Out[]= {0, 0.809017, 0.951057, 0.309017, -0.587785, -1.,
>  -0.587785, 0.309017, 0.951057, 0.809017, 0, -0.809017,
>  -0.951057, -0.309017, 0.587785, 1., 0.587785, -0.309017,
>  -0.951057, -0.809017, 0}
```

式の表現の中に分数が使われていたり、値の代入がされていない変数が含まれていたりすると、Table[ ] により作成されるデータは、式を含んだデータになります。この式を数値化する(割算を実行する等)時に N[ ] を使います。

2次元の配列を作成します。

```
In[]:= Table[Sin[a*x], {x, 0, 2*Pi, 0.4}, {a, 1, 3}]

Out[]= {{0, 0, 0}, {0.389418, 0.717356, 0.932039},
>  {0.717356, 0.999574, 0.675463}, {0.932039, 0.675463, -0.44252},
...
```

□ 配列の長さ・大きさ

```
In[]:= A = {{2, 3}, {4, 5}, {6, 0}};

In[]:= v = {4, -1};
```

配列の長さ(要素の数)を調べます。

```
In[]:= Length[A]

Out[]= 3

In[]:= Length[v]

Out[]= 2
```

配列の大きさを調べます。
結果は $\{row, col\}$ の配列で表示されます。

```
In[]:= Dimensions[A]

Out[]= {3, 2}

In[]:= Dimensions[v]

Out[]= {2}
```

□ 配列への挿入

v の2行目の前に -2.2 を挿入します。

```
In[]:= Insert[v, -2.2, 2]

Out[]:= {4, -2.2, -1}
```

[4] Append[ ] などの関数は、もとの配列に要素を加えた配列を出力しているだけで、もとの配列を書き換えるのではないことに注意してください。もとの配列を書き換える関数として、AppendTo[ ] があります。

A の 3 行目の前に -1, 2 を挿入します。

```
In[]:= Insert[A, {-1, -2}, 3]

Out[] = {{2, 3}, {4, 5}, {-1, -2}, {6, 0}}
```

□ 配列からの取り出し

配列の 2 行目から 3 行目までを取り出します。

```
In[]:= Take[A, {2, 3}]

Out[] = {{4, 5}, {6, 0}}
```

□ 配列の結合

2 つの配列を結合させます。

```
In[]:= Join[A, A]

Out[] = {{2, 3}, {4, 5}, {6, 0}, {2, 3}, {4, 5}, {6, 0}}
```

同様に配列どうしを結合させていますが、Reverse を使ってデータを反転させて結合します。

```
In[]:= Join[A, Reverse[A]]

Out[] = {{2, 3}, {4, 5}, {6, 0}, {6, 0}, {4, 5}, {2, 3}}
```

Mathematica におけるデータ

Mathematica では、すべてのデータはリストという階層構造をもったデータになっています。そのため以下のように任意の深さと型をもつデータが扱えます。

```
a = {"sin curve", 0.27, {0, Pi}, Sin[x]}
```

3.4　行列演算

演算	Mathematica での表現	備考
ベクトル (行列) 積	a . b	$c_{ij} = \sum_k a_{ik} * b_{kj}$
スカラー (配列) 積	a * b	$c_{ij} = a_{ij} * b_{ij}$
行列式	Det[a]	
転置行列	Transpose[a]	
逆行列	Inverse[a]	a は n×n 行列
固有値	Eigenvalues[a]	n 個の固有値
固有ベクトル	Eigenvectors[a]	n 個の n 次元固有ベクトル

ベクトルや行列は、すべての要素が数値で、各行の要素の数が同じリストとして見ることができます。

□ 配列の代入

3×2 の行列 A に $\begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 0 \end{pmatrix}$、2 次元のベクトル v を入力します。n 次元のベクトルは、$n \times 1$ の行列として扱われます。

```
In[]:= A = {{2, 3}, {4, 5}, {6, 0}}

Out[]= {{2, 3}, {4, 5}, {6, 0}}

In[]:= v = {4, -1}

Out[]= {4, -1}
```

□ 配列の参照

A の 2 行目を参照します。

```
In[]:= A[[2]]

Out[]= {4, 5}
```

a_{21} を参照します。

```
In[]:= A[[2, 1]]

Out[]= 4
```

A の転置行列を求めます。${}^tA = \begin{pmatrix} 2 & 4 & 6 \\ 3 & 5 & 0 \end{pmatrix}$

```
In[]:= Transpose[A]

Out[]= {{2, 4, 6}, {3, 5, 0}}
```

A の 1 列目を参照します。

```
In[]:= Transpose[A][[1]]

Out[]= {2, 4, 6}
```

□ スカラー積

行列 A と 3 の積 (スカラー積) を求めます。

```
In[]:= A * 3

Out[]= {{6, 9}, {12, 15}, {18, 0}}
```

ベクトル (2, 3) と v のスカラー積を求めます。

```
In[]:= {2, 3} * v

Out[] = {8, -3}
```

□ 内積

ベクトル (2, 3) と v の内積を求めます。

```
In[]:= {2, 3} . v

Out[] = 5
```

□ 行列積

行列 A とベクトル v の積を求めます。

```
In[]:= A . v

Out[]= {5, 11, 24}
```

行列 A と A の転置行列との積を求めます。

```
In[]:= A . Transpose[A]

Out[]= {{13, 23, 12}, {23, 41, 24}, {12, 24, 36}}

In[]:= Transpose[A] . A

Out[]= {{56, 26}, {26, 34}}
```

3.5　ファイル入出力

演算	Mathematica での表現	備考
アスキーデータの読み込み	ReadList["*filename*", *type*, *opts*]	
ファイルの内容の表示	!! *filename*	
ファイルの内容の実行	<< *filename*	
実行結果の書きだし	*expr* >> *filename*	
シェルコマンドの実行	! *shell-command*	

Mathematica に式を入力する方法として、直接入力する以外にあらかじめ式をファイルへ入力しておき、そのファイルを読み込ませて実行させることもできます。

また、実行結果は画面に出力されますが、結果をファイルへ書き出すこともできます。

□ データの読み込み

ファイル foo.ascii の内容を表示します。

```
In[]:= !! foo.ascii
-9      -2498   11498
-18     -2559   11584
-42     -2611   11626
          ...
```

ファイル foo.ascii から data1D へ数値データを読み込みます。

```
In[]:= data1D = ReadList["foo.ascii", Number];

Out[]= {-9, -2498, 11498, -18, -2559, 11584, -42, -2611,

>    11626, -14, -2576, 11670, -17, -2518, 11697, -11,

                                  ...
```

RecordLists オプションを True にすると、改行で区切られたデータを 1 レコードとして扱います。

```
In[]:= dataZ = ReadList["foo.ascii", Number, RecordLists->True]

Out[]= {{-9, -2498, 11498}, {-18, -2559, 11584}, {-42, -2611, 11626},

>    {-14, -2576, 11670}, {-17, -2518, 11697}, {-11, -2499, 11704},

>    {-25, -2525, 11644}, {-4, -2545, 11603}, {-1, -2516, 11511},
                                  ...
```

□ バッチファイルの実行

あらかじめファイルにコマンド列をエディタなどで入力しておき、一括して実行させることができます。その場合、ファイルの拡張子は習慣的に *.m にします。
(あくまでも習慣的なものであって、任意のファイル名でもかまいません)。

```
In[]:= !!tbl.m
data = Table[Sin[a*x], {x, 0, 2*Pi, 0.4}, {a, 1, 3}];
Take[data, 3]
```

ファイル tbl.m を読み込み、実行させます。

```
In[]:= <<tbl.m

Out[]= {{-11, 0, 0}, {-4, 0, 0}, {-18, 0, 0}}
```

□ 実行結果の書きだし

3＋4 と 3－4 の結果をファイル read.m へ書き出します。この場合もとのファイルの内容は消されてしまいます。
もとのファイルに追加したい場合は >> の代わりに >>> を使います。

```
In[]:= 3 + 4 >> read.m

In[]:= 3 - 4 >>> read.m
In[]:= !!read.m
7
-1
```

3.6　プログラミング

演算	Mathematica での表現	備考
条件分岐	If[*cond*, *true*, *false*]	
	Switch[$form_1$, $value_1$, $form_2$, $value_2$, ..., $value_d$]	
繰り返し	For[*start*, *test*, *incr*, *body*]	
	While[*cond*, *body*]	
モジュールの作成	Module[{*var*, ...}, *expr*...]	

何回も使う処理を新たに関数として定義したり、値の大小など条件によって処理を変えたりすることができます。

Mathematica には、条件分岐や繰り返しを行うIf[ ], Switch[ ], For[ ], While[ ] などの各関数があり、条件判断も<, >, <=, >=, ==, &&, || など C 言語に近い表現になっています。

□ 制御文

data の中で 3 以上 5 より小さいデータを数えます。

```
In[]:= For[i = 0; pcount = 0, i <= Length[data], i++,
          If[data[[i]] >= 3 && data[[i]] < 5,
            pcount++]
       ]; pcount

Out[]= 2
```

□ 関数の定義

前節で説明したファイルから 2 次元配列データを読み込む式を、新たに ReadArrayList という関数として定義します。
右辺の仮引数 fname のあとには、アンダーバー (_) をつけます[5]。

```
In[]:= ReadArrayList[fname_] :=
            ReadList[fname, Number, RecordLists->True];
```

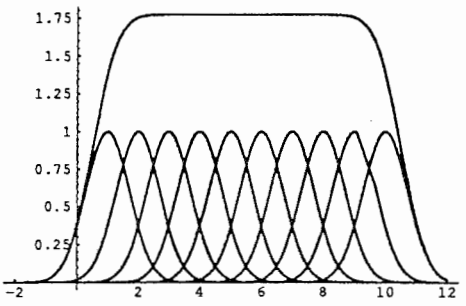
以後 ReadArrayList は他の Mathematica の関数と同様に扱うことができます。
パラメータ a をもつ関数を定義します。次に、パラメータを 1 から 10 まで変えた 10 個の関数のリストを作成します。その後、関数のリストにそれらの関数をすべて加えた関数を追加します。

```
In[]:= data = ReadArrayList["wavedata"];

In[]:= filt[x_] := Exp[-(x-a)^2];

In[]:= funcs = Table[filt[x], {a, 1, 10}];

In[]:= funcs = Append[funcs, Apply[Plus, funcs]]
                        2            2            2            2
               -(-1 + x)    -(-2 + x)    -(-3 + x)    -(-4 + x)
Out[]= {E              , E          , E          , E          ,

          ...
```

Apply という関数は、第 2 引数の配列に第 1 引数の関数を適用させる関数です。
e.x. `Apply[Plus, 2, 3, 4]`
`== Plus[2, 3, 4] = 9`

11 個の関数のリストをグラフ表示します。

```
In[]:= Plot[Evaluate[funcs], {x, -2, 12}, PlotRange->All]
```



[5]実際にはアンダーバー (_) をつけたシンボルはパターン置換されます。

□ モジュールの作成

複数の式や変数を使用する処理をモジュールとして作成することができます。
右記ではデータの間引きを行う処理をモジュールにします。
Module の最初の引数は、モジュール内だけで使用する変数を並べます。この中で変数を初期化することもできます。
2番目の引数には式を1つだけ記述しますが、";"で区切れば複数の式を記述することができます。

ListReduce はデータ *list* と間引き率 *ratio* の2つの引数をとります。
(* *) で囲まれた部分はコメントです。

- 最初に引数のチェックをして、間引き率が0以下の場合は NULL リターンします。
- 次の While は永久ループです。次に取るべきデータの位置を計算して、もとのデータを越えた場合、いままでの結果データのリストを返します。もとのデータを越えない場合、結果データのリストに追加します。

```
In[]:= ListReduce[list_, ratio_] :=
          Module[{i = 1, pos, len = Length[list], ret = {}},
                 (* check argument *)
                 If[ratio <= 0, Return[{}]];
                 (* eternal loop *)
                 While[True,
                       pos = Round[i / ratio];
                       If[pos <= len,
                          AppendTo[ret, list[[pos]]],
                          Return[ret]];
                       i++]];
```

4　データ処理の応用

4.1　信号処理

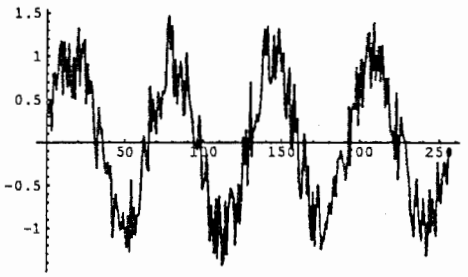
演算	Mathematicaでの表現	備考
数値積分	Integrate[f, x] Integrate[f, {$x, xmin, xmax$}] Integrate[f, {$x, xmin, xmax$}, {$y, ymin, ymax$}]	$\int f dx$ $\int_{xmin}^{xmax} f dx$ $\int_{xmin}^{xmax} dx \int_{ymin}^{ymax} dy f$
フーリエ変換	Fourier[{$a_1, a_2, \ldots$}]	$n = 2^m$ の時 FFT $\frac{1}{\sqrt{n}} \sum_{r=1}^{n} a_r e^{2\pi i(r-1)(s-1)/n}$
逆フーリエ変換	InverseFourier[{$b_1, b_2, \ldots$}]	$\frac{1}{\sqrt{n}} \sum_{s=1}^{n} b_s e^{-2\pi i(r-1)(s-1)/n}$
2次元フーリエ変換	Fourier[{{$a_{11}, a_{12}, \ldots$}, {$a_{21}, a_{22}, \ldots$}, $\ldots$}]	
2次元逆フーリエ変換	InverseFourier[{{$b_{11}, b_{12}, \ldots$}, {$b_{21}, b_{22}, \ldots$}, $\ldots$}]	
畳み込み	InverseFourier[Fourier[f]*Fourier[h]]	
移動平均	MovingAverage[$data, n$]	

□ 畳み込み

入力信号のサンプルとして、ランダムノイズをのせた正弦波から 256 点のデータを作成します。
データをプロットします。

```
In[]:= flist = Table[N[Sin[4*2*Pi*n/256] + (Random[] - 1/2)], {n, 1, 256
```

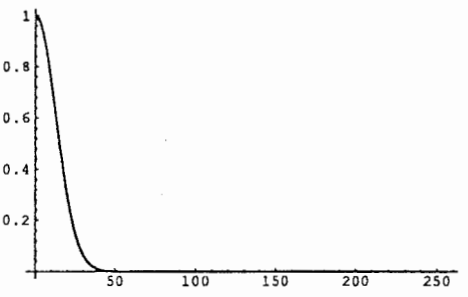
```
In[]:= ListPlot[flist, PlotJoined->True]
```



フィルタ関数として、同様に 256 点のデータを作成します。

```
In[]:= hlist = Table[N[Exp[-200*(n/256)^2]], {n, 256}];
```

```
In[]:= ListPlot[hlist, PlotJoined->True, PlotRange->All]
```



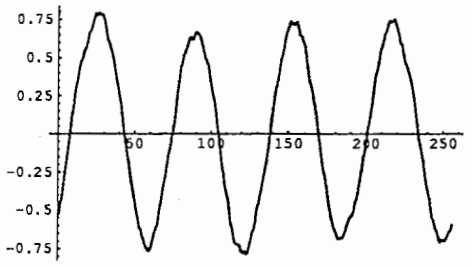
フーリエ変換します。

```
In[]:= fspec = Fourier[flist];

In[]:= hspec = Fourier[hlist];
```

fspec とhspec を逆フーリエ変換すると、flist と hlist の畳み込みになります。

Chop[ ] は微小な数値 (デフォルトでは、10^{-10}) を 0 にする関数です。

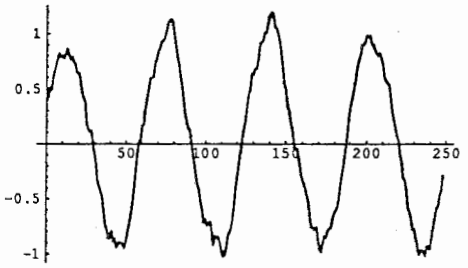
```
In[]:= conv = InverseFourier[fspec*hspec];

In[]:= ListPlot[Chop[conv], PlotJoined->True, PlotRange->All]
```



□ 移動平均

移動平均を求めます。

```
In[]:= ma = MovingAverage[flist, 8];

In[]:= ListPlot[Chop[ma], PlotJoined->True, PlotRange->All]
```



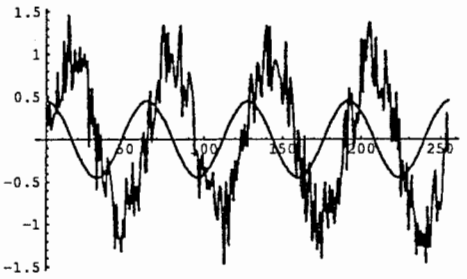
□ 自己相関関数

自己相関関数を定義します。

```
In[]:= AutoCorrelation[list_] :=
          Module[{j, sum, n = Length[list], data = Join[list, list]},
                Table[For[sum = 0; j = 1, j <= n-1, j++,
                      sum = sum+data[[j]]*data[[j+i]]]; sum/n,
                     {i, 0, n-1}]
          ];
```

自己相関関数を計算し、入力信号とともにプロットします。

```
In[]:= result = AutoCorrelation[flist];

In[]:= src = Transpose[{Range[0, 255], flist}];

In[]:= dst = Transpose[{Range[0, 255], result}];

In[]:= Show[Graphics[{Line[src], Line[dst]}, Axes->True]]
```



4.2　統計処理

以下のパッケージを読み込んでください。

```
In[]:= <<Statistics`Master`
```

演算	Mathematicaでの表現	備考
最大値	Max[*data*]	
最小値	Min[*data*]	
総和	Sum[*f*, {*i*, *imin*, *imax*}]	$\sum_{i=imin}^{imax} f$
	Sum[*f*, {*i*, *imin*, *imax*, *di*}]	*di* は増分
	Sum[*f*, {*i*, *imin*, *imax*}, {*j*, *jmin*, *jmax*}]	$\sum_{i=imin}^{imax}\sum_{j=jmin}^{jmax} f$
累積	Product[*f*, {*i*, *imin*, *imax*}]	$\prod_{i=imin}^{imax} f$
	Product[*f*, {*i*, *imin*, *imax*, *di*}]	*di* は増分
	Product[*f*, {*i*, *imin*, *imax*}, {*j*, *jmin*, *jmax*}]	$\prod_{i=imin}^{imax}\prod_{j=jmin}^{jmax} f$
平均	Mean[*data*]	
分散	Variance[*data*]	
標準偏差	StandardDeviation[*data*]	
中央値	Median[*data*]	
正規分布	NormalDistribution[*mu*, *sigma*]	$\frac{1}{\sqrt{2\pi}\,\sigma} exp(\frac{(x-\mu)^2}{2\sigma^2})$
ヒストグラム	BinCounts[*data*, {*xmin*, *xmax*, *dx*}]	

□ 平均・分散

平均を求めます。

```
In[]:= data = {1.3, 5.2, 2.4, 3.3, 8.1, 7.0, 6.9, 4.5, 8.3, 2.1, 2.5}

Out[]= {1.3, 5.2, 2.4, 3.3, 8.1, 7., 6.9, 4.5, 8.3, 2.1, 2.5}

In[]:= Mean[data]

Out[]= 4.69091
```

分散を求めます。

```
In[]:= Variance[data]

Out[]= 6.53491
```

□ 分布・ヒストグラム

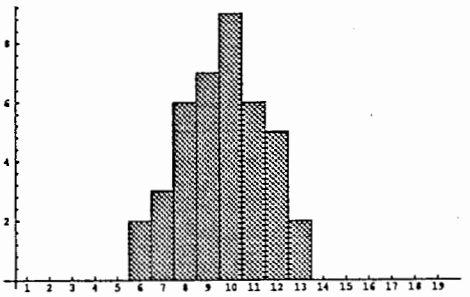
平均10、標準偏差2の正規分布をndistとします。
分布 ndist に沿った乱数データを40個作成します。

Random の引数に分布を指定すると、分布に沿った乱数が得られます。

```
In[]:= ndist = NormalDistribution[10, 2];

In[]:= data = Table[Random[ndist], {n, 40}];

Out[]= {10.0155, 10.4782, 8.26598, 12.6091, 11.4867, 10.0884,

                    ...
```

dataのヒストグラムを求めます。結果は1次元の配列で帰ってきます。

```
In[]:= hist = BinCounts[data, {1, 20, 1}]

Out[]= {0, 0, 0, 0, 0, 2, 3, 6, 7, 9, 6, 5, 2, 0, 0, 0, 0, 0, 0}

In[]:= BarChart[%, BarSpacing->0]
```



5 数値データのグラフ表示

本章以降では数値データをグラフ表示する具体例を説明します。
はじめにデータを以下の4つに分類しておきます。

1次元配列データ　1変数の値
2次元配列データ　2変数の格子上の点の値
2次元座標データ　(x, y) が組になったもの
3次元座標データ　(x, y, z) が組になったもの

2次元座標データも3次元座標データも2次元配列データの一種ですが、座標そのものを表示するためのデータという意味で分けておきます。

5.1 データファイルの読み込み

アスキー形式のファイルのデータをグラフ表示したい場合は、まず、以下の方法でファイルからデータを読み込み配列変数に代入します。

1次元配列データ	data1D = ReadList[*fname*, Number]	$\{x_1, x_2, x_3, \ldots\}$
2次元配列データ	dataZ = ReadList[*fname*, Number, RecordLists->True]	$\{\{z_{11}, z_{12}, \ldots\}, \{z_{21}, z_{22}, \ldots\}, \ldots\}$

□ アスキー形式データの読み込み

ファイル foo.ascii に2次元配列データが以下のようにアスキー形式で格納されているとします。

x_{11}	x_{12}	x_{13}	...	x_{1m}
x_{21}	x_{22}	x_{23}	...	x_{2m}
		⋮		
x_{n1}	x_{n2}	x_{n3}	...	x_{nm}

dataZ に dataZ[[i,j]] = x_{ij} のように代入されます。

```
In[]:= dataZ = ReadList[foo.ascii, Number, RecordList->True]
```

5.2 1次元配列データのグラフ表示

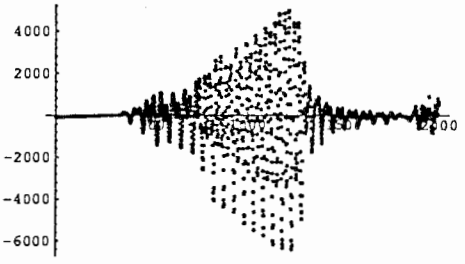
data1D = $\{x_1, x_2, \ldots\}$ のように1次元配列の値が data に代入されているとします。

演算	Mathematicaでの表現	備考
点グラフ	ListPlot[*data1D*]	
線グラフ	ListPlot[*data1D*, PlotJoined->True]	

□ 点グラフ

点グラフ表示させます。
第2引数以降はオプションの指定です。
PlotRange オプション[6]は、値の表示範囲を指定します。

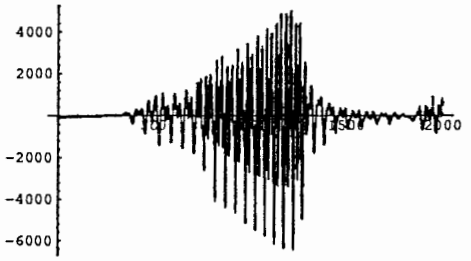
```
In[]:= ListPlot[data1D,
            PlotRange->All]
```



□ 折れ線グラフ

PlotJoined オプションを True にすると折れ線グラフになります。

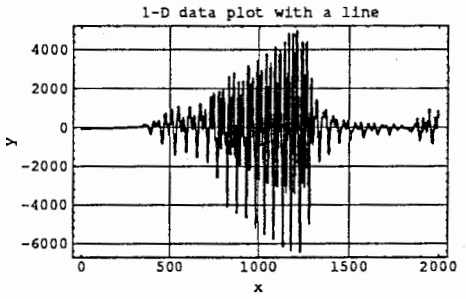
```
In[]:= ListPlot[data1D, PlotJoined->True,
         PlotRange->All]
```



ListPlot 関数のオプションをいろいろ変えて表示します。

オプションの説明
PlotRange: 値の表示範囲
Axes: 座標軸の表示／非表示
Frame: 外枠の表示／非表示
GridLines: グリッド (格子) の描画
FrameLabel: 外枠のラベル
PlotLabel: グラフのタイトル

```
In[]:= ListPlot[data1D, PlotJoined->True,
         PlotRange->All,
         Axes->False,
         Frame->True,
         GridLines->Automatic,
         FrameLabel->{"x", "y"},
         PlotLabel->"1-D data plot with a line"]
```



5.3 2次元配列データのグラフ表示

dataZ = {{$x_{11}, x_{12}, \ldots$}, {x_{21}, x_{22},}, ...} のように2次元配列の値が dataZ に代入されているとします。

演算	Mathematica での表現	備考
サーフェス表示	ListPlot3D[*dataZ*, *opts*]	
等高線表示	ListContourPlot[*dataZ*, *opts*]	
濃淡表示	ListDensityPlot[*dataZ*, *opts*]	

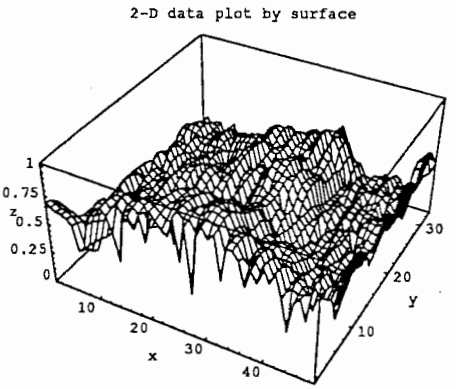
[6]本編では、関数のオプション名は option-name 、関数名は func-name というように書体で表示の区別しています。

□ サーフェス表示

PlotRange: 値の表示範囲
BoxRatios: x, y, z の比
Shading: 陰影をつける／つけない
AxesLabel: 座標軸のラベル
PlotLabel: グラフのタイトル

外枠を表示したくない場合は、Boxed オプションを False に指定します。(デフォルトでは、True)
また、視点の座標位置を変えるには、ViewPoint オプションで指定します。(デフォルトでは、{1.3, -2.4, 2.0})

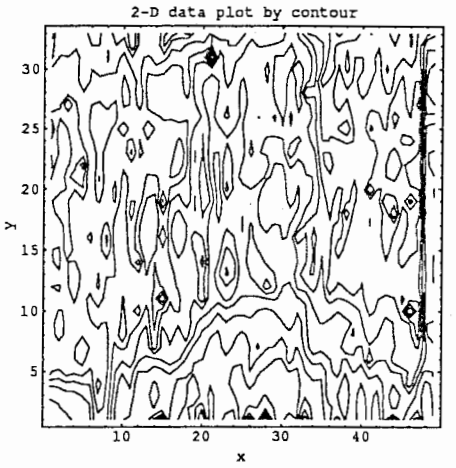
```
In[]:= ListPlot3D[dataZ,
          PlotRange->All,
          BoxRatios->{1, 1, 0.5},
          Shading->False,
          AxesLabel->{"x", "y", "z"},
          PlotLabel->"ListPlot3D with Z data"]
```



□ 等高線表示

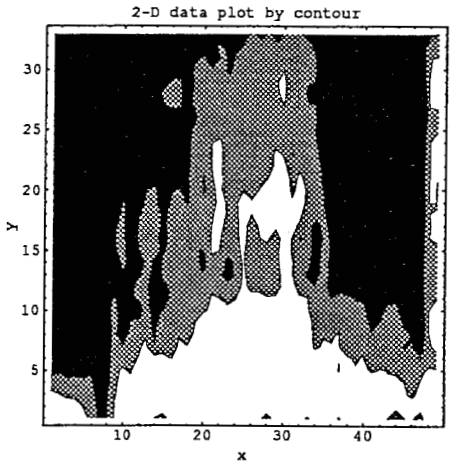
PlotRange: 値の表示範囲
Contours: 等高線のレベル
ContourShading: 塗りつぶしをする／しない
FrameLabel: 外枠のラベル
PlotLabel: グラフのタイトル

```
In[]:= ListContourPlot[dataZ,
          PlotRange->All,
          Contours->40,
          ContourShading->False,
          FrameLabel->{"x", "y"},
          PlotLabel->"2-D data plot by surface"]
```



Contours に配列を与えると、各等高線のレベルの値を指定することができます。
ContourShading を True にして塗りつぶしを行います。

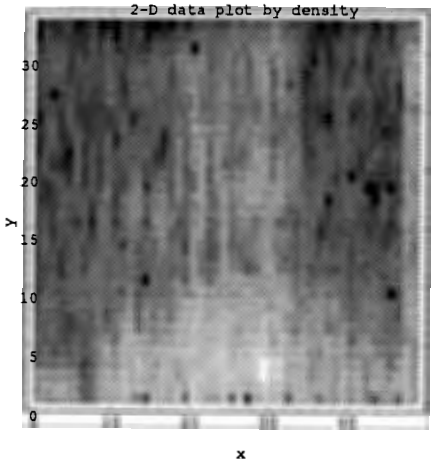
```
In[]:= ListContourPlot[dataZ,
         PlotRange->All,
         Contours->{0, 1, 2},
         FrameLabel->{"x", "y"},
         PlotLabel->"ListContourPlot with Z data",
         ContourShading->True]
```



□ 濃淡表示

PlotRange: 値の表示範囲
Mesh: メッシュの表示／非表示
FrameLabel: 外枠のラベル
PlotLabel: グラフのタイトル

```
In[]:= ListDensityPlot[dataZ,
         PlotRange->All,
         Mesh->False,
         FrameLabel->{"x", "y"},
         PlotLabel->"2-D data plot by density"]
```



5.4　2次元座標点の表示

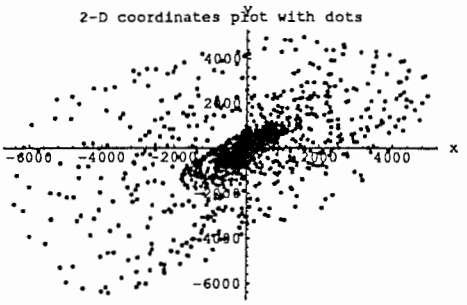
data2D = {{x_1, y_1}, {x_2, y_2},...} のように2次元座標点が data2D に代入されているとします。

演算	Mathematica での表現	備考
2次元座標点の点グラフ	ListPlot[*data2D*]	
	Show[Graphics[Map[Point, *data2D*], *opts*]]	
2次元座標点の線グラフ	ListPlot[*data2D*, PlotJoined->True]	
	Show[Graphics[Line[*data2D*], *opts*]]	

□ 2次元座標点の点グラフ

Axes: 座標軸の表示／非表示
AxesLabel: 座標軸のラベル
PlotLabel: グラフのタイトル

```
In[]:= ListPlot[data2D,
           PlotRange->All,
           AxesLabel->{"x", "y"},
           PlotLabel->"2-D coordinates plot with dots"]
```

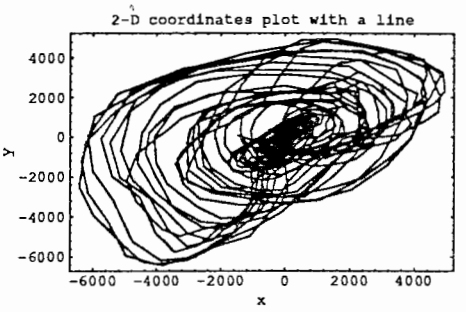


□ 2次元座標点の線グラフ

同じデータを今度は折れ線グラフで表示します。点グラフの時とオプションを少し変えて表示させています。

Axes: 座標軸の表示／非表示
Frame: 外枠の表示／非表示
FrameLabel: 外枠のラベル
PlotLabel: グラフのタイトル

```
In[]:= ListPlot[data2D, PlotJoined->True,
           PlotRange->All,
           Axes->False,
           Frame->True,
           FrameLabel->{"x", "y"},
           PlotLabel->"2-D coordinates plot with a line"]
```



5.5　3次元座標点の表示

data3D = {{x_1, y_1, z_1}, {x_2, y_2, z_2},...} のように3次元座標点が data3D に代入されているとします。

演算	Mathematicaでの表現	備考
3次元座標点の点グラフ	Show[Graphics3D[Map[Point, *data3D*]]]	
3次元座標点の線グラフ	Show[Graphics3D[Line[*data3D*]]]	

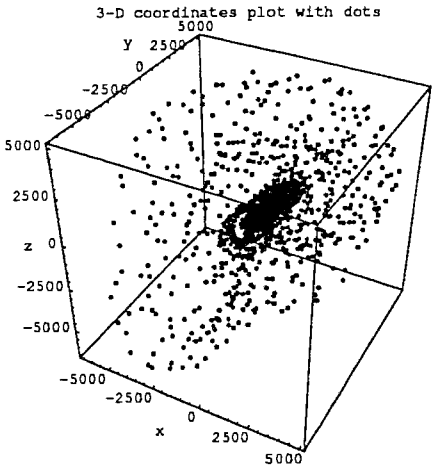
3次元座標点を簡単に表示する関数は用意されていません。したがって、**グラフィック・プリミティブ**から作成することになります。詳しくは、**8 グラフィックスの基礎**を参照してください。

□ 3次元座標点の点グラフ

Map 関数は第1引数の関数を第2引数の個々のデータに適用する関数です。
ここでは、data3D の個々のデータについて点を作成してグラフを表示しています。

Axes: 座標軸を表示するか
AxesLabel: 座標軸のラベル
PlotLabel: グラフのタイトル

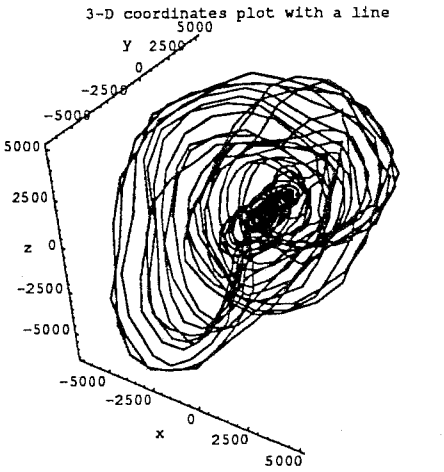
```
In[]:= Show[Graphics3D[Map[Point, data3D],
           PlotRange->All,
           Axes->True,
           AxesLabel->{"x", "y", "z"},
           PlotLabel->"3-D coordinates plot with dots"]]
```



□ 3次元座標点の折れ線グラフ

Axes: 座標軸の表示／非表示
Boxed: 外枠の表示／非表示
BoxRatios: 外枠の縦・横・高さの比
AxesLabel: 座標軸のラベル
PlotLabel: グラフのタイトル

```
In[]:= Show[Graphics3D[Line[data3D],
           Axes->True,
           Boxed->False,
           BoxRatios->{1, 1, 1},
           AxesLabel->{"x", "y", "z"},
           PlotLabel->"3-D coordinates plot with a line"]]
```



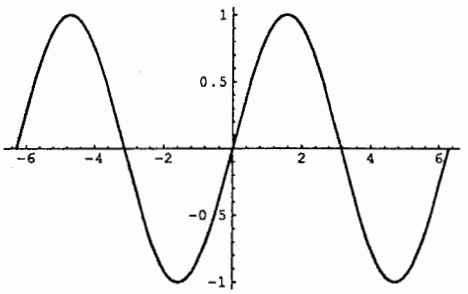
6　関数のグラフ表示

6.1　1変数関数のグラフ表示

□ 1変数関数のグラフ

Plot 関数は、第1引数に式、第2引数に定義域の範囲を指定します。
{x, -2*Pi, 2*Pi} は変数 x の -2π から 2π までの範囲を描画せよという指示です。

```
In[]:= Plot[Sin[x], {x, -2*Pi, 2*Pi}]
```



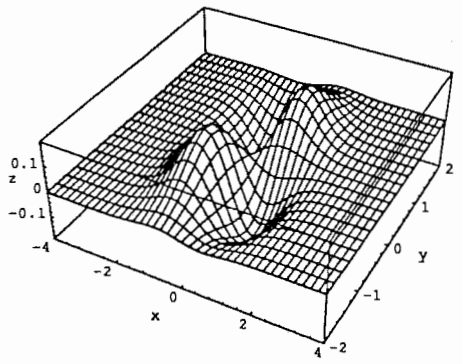
6.2　2変数関数のグラフ表示

□ サーフェス表示

2変数関数 g(x,y) を定義します。
関数 g をサーフェス表示させます。

PlotRange: 関数値の範囲
AxesLabel: 座標軸のラベル
Shading: 陰影
PlotPoints: 格子点の数

```
In[]:= g[x_, y_] := x * y * Exp[-x^2 - y^2];
In[]:= Plot3D[g[x, y], {x, -4, 4}, {y, -2, 2},
          PlotRange->All,
          AxesLabel->{"x", "y", "z"},
          Shading->False,
          PlotPoints->{40, 20}]
```

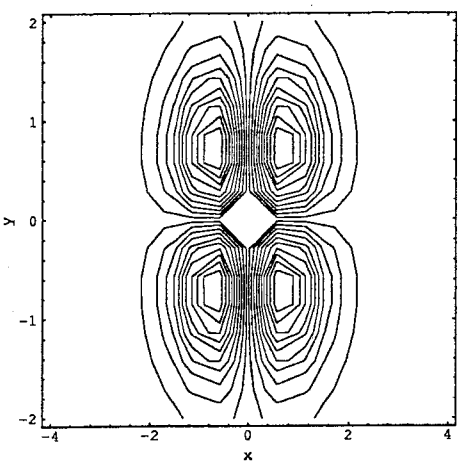


□ 等高線表示

前出の関数 g を等高線表示させます。

PlotRange: 関数値の範囲
FrameLabel: 外枠のラベル
ContourShading: 塗りつぶすか
Contours: 等高線のレベルの数

```
In[]:= ContourPlot[g[x, y], {x, -4, 4}, {y, -2, 2},
        PlotRange->All,
        FrameLabel->{"x", "y"},
        ContourShading->False,
        Contours->20]
```

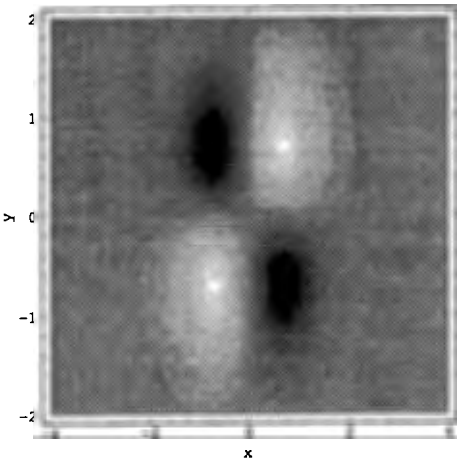


□ 濃淡表示

関数 g を濃淡表示させます。

PlotRange: 関数値の範囲
FrameLabel: 外枠のラベル
Mesh: メッシュを描くかどうか
PlotPoints: 格子点の数

```
In[]:= DensityPlot[g[x, y], {x, -4, 4}, {y, -2, 2},
        PlotRange->All,
        FrameLabel->{"x", "y"},
        Mesh->False,
        PlotPoints->80]
```



7　複数のグラフ表示およびその他のグラフ表示

7.1　複数のグラフ表示

演算	Mathematica での表現	備考
複数のグラフを重ねて表示	Show[$g, g, ...$]	
複数のグラフを並べて表示	GraphicsArray[$g, g, ...$]	
線種の指定	AbsoluteDashing[*pattern*]	

複数のグラフを重ねて表示するには、Show の引数にグラフを並べます。ただし、すべて同じ種類のグラフである必要があります。

□ 複数のグラフを重ねて表示

3つの線種の異なる折れ線グラフを重ねて表示します。

```
In[]:= data1D1 = Transpose[{Table[x, {x, 1, 100}],
          Take[data1D, {1001,1100}]}];
In[]:= data1D2 = Transpose[{Table[x, {x, 1, 100}],
          Take[data1D, {1101, 1200}]}];
In[]:= data1D3 = Transpose[{Table[x, {x, 1, 100}],
          Take[data1D, {1201, 1300}]}];
```

線種を変えてグラフを一つづつ作成します。
PlotStyle の指定を右記のようにすると、g1 は太い実線、g2 は点線、g3 は 1 点鎖線になります。
DisplayFunction を Identity にすると、画面には表示されません。

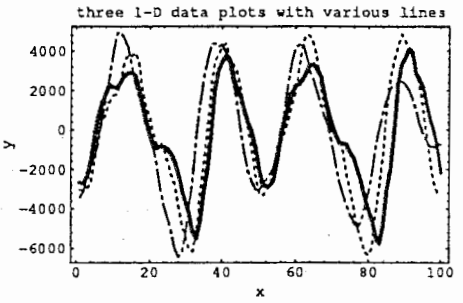
PlotStyle: 線種
DisplayFunction: 描画する関数

```
In[]:= g1 = ListPlot[data1D1, PlotJoined->True,
          PlotStyle->{Thickness[0.01]},
          DisplayFunction->Identity];
In[]:= g2 = ListPlot[data1D2, PlotJoined->True,
          PlotStyle->{AbsoluteDashing[{1, 2}]},
          DisplayFunction->Identity];
In[]:= g3 = ListPlot[data1D3, PlotJoined->True,
          PlotStyle->{AbsoluteDashing[{8, 2, 1, 2}]},
          DisplayFunction->Identity];
```

作成した3つのグラフを重ねて表示します。
DisplayFunction を元の設定 ($DisplayFunction) にもどして画面に表示させます。

PlotRange: 値の範囲
Axes: 座標軸の表示／非表示
Frame: 外枠の表示／非表示
FrameLabel: 外枠のラベル
DisplayFunction: 描画する関数
PlotLabel: グラフのタイトル

```
In[]:= Show[g1, g2, g3,
          PlotRange->All,
          Axes->False,
          Frame->True,
          FrameLabel->{"x", "y"},
          DisplayFunction->$DisplayFunction,
          PlotLabel->"three 1-D data plots with lines"]
```



□ 複数のグラフを並べて表示

4つの視点の位置を変えたサーフェスグラフを並べて表示します。
タイトルと視点の位置を指定してグラフ表示する処理をSurfv と定義しておきます。

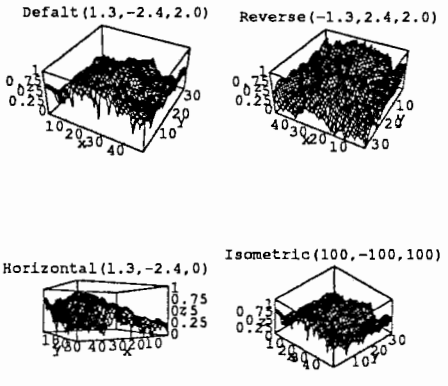
```
In[]:= Surfv[title_, vp_] :=
          ListPlot3D[dataZ,
              PlotRange->All,
              BoxRatios->{1, 1, 0.5},
              Shading->False,
              ViewPoint->vp,
              AxesLabel->{"x", "y", "z"},
              PlotLabel->title];
```

視点の位置を変えて4つのグラフを作成していきます。

```
In[]:= g1=Surfv["Default(1.3,-2.4,2.0)",{1.3,-2.4,2.0}];
In[]:= g2=Surfv["Reverse(-1.3,2.4,2.0)",{-1.3,2.4,2.0}];
In[]:= g3=Surfv["Horizontal(1.3,-2.4,0)",{1.3,-2.4,0}];
In[]:= g4=Surfv["Isometric(100,-100,100)",{100,-100,100}];
```

4つのグラフを1画面に並べて表示させます。

```
In[]:= Show[GraphicsArray[{{g1, g2}, {g3, g4}}]]
```



7.2　その他のグラフ表示

以下のパッケージを読み込んでください。

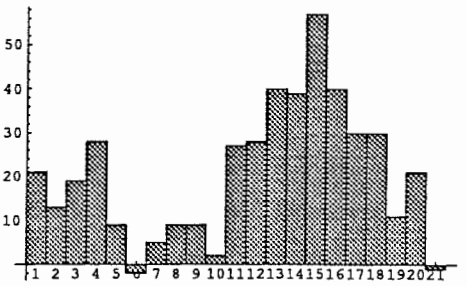
```
In[]:= <<Graphics`Graphics`
```

演算	Mathematicaでの表現	備考
1変数関数の片対数グラフ	LogLinearPlot[f, {x, $xmin$, $xmax$}]	
	または LinearLogPlot[f, {x, $xmin$, $xmax$}]	
1変数関数の両対数グラフ	LogLogPlot[f, {x, $xmin$, $xmax$}]	
1次元配列データの片対数グラフ	LogListPlot[$data1D$]	
1次元配列データの両対数グラフ	LogLogListPlot[$data1D$]	
棒グラフ	BarChart[$data1D$]	
円グラフ	PieChart[$data1D$]	
アニメーション表示	Animation[$plot$, {t, $tmin$, $tmax$}]	<<Graphics`Animation`

□ 棒グラフ

BarSpacing: 棒の間隔

```
In[]:= BarChart[data1D, BarSpacing->0]
```



7.3 グラフの出力

演算	Mathematica での表現	備考
ファイルへの出力	`Display["!psfix > file", g]`	
プリンタへの出力	`PSPrint[g]`	
	`Display["!psfix | lpr -Pprinter", g]`	

グラフィックスを PostScript 形式で出力するには、Display 関数を使います。第1引数は、実行するコマンド文字列です。

最初の '!' は UNIX 上のコマンド実行を意味します。psfix コマンドは、Mathematica の出力するグラフィックス[7]を PostScript に変換するフィルタです。

□ プリンタへの出力

グラフィックスをプリンタ lw へ出力します。

```
In[]:= g = ListPlot[data1D, PlotRange->All];
In[]:= Display["!psfix | lpr -Plw", g]
```

またプリンタへの出力は、**PSPrint** という簡単な関数が用意されています。
この場合 UNIX の環境変数 PRINTER に設定されているプリンタへ出力されます。

```
In[]:= PSPrint[g]
```

□ ファイルへの出力

プリンタではなくファイルに出力する場合もプリンタへの出力と同様です。
Display 関数のコマンド引数を右記のようにしてやればよいでしょう。

```
In[]:= Display["!psfix > graph.ps", g]
```

8 グラフィックスの基礎

ここでは、Mathematica の基本的なグラフィック機能を説明します。

グラフィックスを作成するには、まず数値データから、グラフィック表示用の線や点などのグラフィック・プリミティブとよばれるデータを作成します。

演算	Mathematica での表現	備考
点データを作成する	`Point[{`x_1, y_1`}]`	
線データを作成する	`Line[{{`x_1, y_1`}, {`x_2, y_2`}, ...}]`	

[7] マクロを含んだ PostScript 。

これらの関数によって作成された線や点から、以下のような関数によって2次元や3次元のグラフィック・オブジェクトを作成します。

演算	Mathematicaでの表現	備考
2次元グラフィックスの作成	Graphics[*prim-list*]	
3次元グラフィックスの作成	Graphics3D[*prim-list*]	
サーフェス・グラフィックスの作成	SurfaceGraphics[*data-list*]	
等高線グラフィックスの作成	ContourGraphics[*data-list*]	

なお、Graphics や Graphics3D の引数には、グラフィック・プリミティブのリストを指定しますが、SurfaceGraphics や ContourGraphics の引数には、データを直接指定します。

これらの関数によって作成されたグラフィック・オブジェクトを以下のような関数を使って画面に表示させたり、ファイルやプリンタに出力させます。

演算	Mathematicaでの表現	備考
グラフィックスを画面に表示する	Show[*g*]	
	Show[*g*, *g*, ...]	
グラフィックスを出力する	Display["*command*", *g*]	

以上をまとめると、グラフィックスを作成し表示するには、

表示 ← オブジェクトの作成 ← プリミティブの作成 ← データ
Show [Graphics,Graphics3D など [Line,Point など [{x, y} など]]]

のような手順となります。
前節で説明したグラフを表示する関数は、これらの手順をまとめて簡単にしたものです。

□ グラフィック・プリミティブを使ったグラフ表示の例

ListPlot を使わずに、折れ線グラフを描かせます。

```
In[]:= Show[Graphics[Line[data2D]]]
```

9 Q&A

グラフの目盛りを変えるには？

グラフの目盛りは自動的につきますが、以下のオプションで変更することができます。

Ticks オプション 座標軸の目盛り

FrameTicks オプション 外枠の目盛り

どちらも、指定の仕方は同じです。目盛りをつけたくない場合は、

Ticks->None(またはFrameTicks->None)

と指定します。

また、Ticks->{{0, 1000, 2000}, {-1, 0, 1}} のように目盛りをつける位置を細かく指定することもできます。

ウィンドウの大きさを変えるには？

以下のグローバル変数を変更します。

変数名	説明	デフォルト値
$DisplayWidth	ウィンドウの幅	400
$DisplayHeight	ウィンドウの高さ	400
$DisplayTitle	ウィンドウのタイトル	−

幅 600 、高さ 200 のウィンドウにします。
同時に座標の縦横比も変更します。

```
In[]:= $DisplayWidth = 600;
In[]:= $DisplayHeight = 200;
In[]:= Show[g, AspectRatio->1/4]
```

グラフィックスを TeX に取り込むには？

Mathematica のグラフィックスを PostScript に変換するコマンドととして psfix というコマンドが用意されています。

psfix コマンドで -epsf オプションを指定すると、eps 形式[8]のファイルに変換してくれます。

グラフィックスを eps 形式でファイルへ出力します。

```
In[]:= Display["!psfix -epsf > foo.eps", %]
```

eps 形式のファイルを TeX に取り込む方法は、dvi 形式から PostScript 形式へ変換するドライバによって異なりますが、右記に dvips の場合の例をあげておきます。

```
\documentstyle[epsf,...
        .
        .
\epsfxsize=10cm
\epsfbox{foo.eps}
        .
        .
```

色が出ない時は？

たいていのマシンでは、使える色の数は決まっています。例えば、SPARCStation2 の場合 256 色しか同時に出せません。

Mathematica のグラフや色をたくさん使う他のソフトを表示させたままにしておくと、色が足りなくなり、色が少なくなったり出なくなったりしてしまいます。

色がおかしいと思ったときは、以前に表示させた Mathematica のグラフが残っていないか、または色をたくさん使うソフトを立ち上げていなかを確認してみてください。

謝辞

本編の作成にあたっては、第六研究室の佐藤 雅昭氏に全面的に御協力いただきました。
ここに記して感謝します。

[8] Enhancement PostScript。大きさが変更できる PostScript 形式。