

1 数値に関する命令

▶ 小数と分数

ふつう電卓などの割り算の計算 $1 \div 3$ は、0.333333 などと数値表現されることが多いのですが、Mathematica では、

$$1 \div 3$$

$$\frac{1}{3}$$

のように分数で返されます（ $\div$ は、`[Esc] div [Esc]` で入力、またはファイルメニューのパレットから BasicInput（基礎的な入力）を選択し、パレットからも入力できます）（図1.1）。

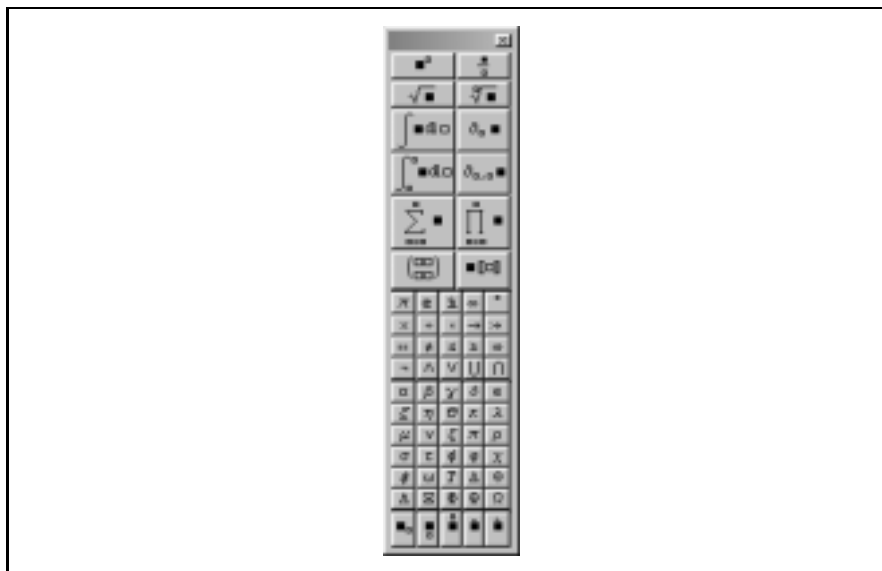
また、分数の入力は、`[Ctrl] + [/]`（「`[Ctrl] + [/]`」は、`[Ctrl]` キーを押しながら、`/` キーを押すことを意味します）で可能です。では、分数ではなく数値を得るにはどうすればよいのでしょうか。Mathematica には、数値を得る関数が用意されています。

関数 `N`（数値表現）は、桁数の指定も可能です。例えば、 π を 100 桁表示させてみましょう。

```
N[Pi, 100]
3.141592653589793238462643383279502884197169399\
3751058209749445923078164062862089986280348253\
42117068
```

次に、逆の操作を説明しましょう。例えば、1.23456 を分数で表してみます。

図 1.1 パレット BasicInput



Rationalize[1.23456]

$\frac{3858}{3125}$

ほかに、整数部分だけを取り出す関数などがあります。

IntegerPart[Pi]

3

▶ 円周率と自然対数の底

円周率は、Pi または π という形で用意されています。Pi を利用する場合は、必ず最初の文字 P は大文字にしてください。 π は、先ほどの BasicInput パレット

第 1 章 Mathematica の基礎

からも入力可能ですが、`[Esc] pi [Esc]` または `[Esc] p [Esc]` で入力できます。

```
N[ $\pi$ ]  
3.14159
```

自然対数の底の場合は、E または e で入力します。Pi のときと同様 E は大文字で、e は `[Esc] ee [Esc]` で入力します。当然、BasicInput パレットからの入力も可能です。

```
N[E]  
2.71828  
  
N[e]  
2.71828
```

▶ 偶数・奇数

偶数や奇数を判断するには、EvenQ や OddQ という関数を使います。これは、偶数・奇数によって処理を分けるときに便利な関数です。

偶数をチェックするには、次のように入力します。

```
EvenQ[3]  
False
```

奇数をチェックするには、次のように入力します。

```
OddQ[3]  
True
```

▶ 素数

BASICなどで素数を探すには、若干のプログラミングを必要としますが、*Mathematica*では、素数に関して n 番目の素数を見つける `Prime` や、素数か否かを答える `PrimeQ` が用意されています。さて、`Prime` は次のように使います。例えば、10 番目の素数を求めるには、

```
Prime[10]  
  
29
```

とします。また、ある数字が素数かどうか調べるには、`PrimeQ` を使います。

```
PrimeQ[123]  
  
False
```

次のように `Map` (または `/@`, p.40 を参照) と `Range` (p.34 を参照) を上手に使うと、素数を 10 個取り出すということなども簡単にできます。

```
Prime /@ Range[10]  
  
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
```

それでは、10 番目から 100 番目の素数を取り出してみましょう。

第 1 章 Mathematica の基礎

```
Prime/@ Range[10, 100]
```

```
{29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
139, 149, 151, 157, 163, 167, 173, 179, 181, 191,
193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
251, 257, 263, 269, 271, 277, 281, 283, 293, 307,
311, 313, 317, 331, 337, 347, 349, 353, 359,
367, 373, 379, 383, 389, 397, 401, 409, 419,
421, 431, 433, 439, 443, 449, 457, 461, 463,
467, 479, 487, 491, 499, 503, 509, 521, 523, 541}
```

▶ 素因数分解

素因数分解は、FactorIntegerを使います。それでは、20 を素因数分解してみましょう。

```
FactorInteger[20]
```

```
{{2, 2}, {5, 1}}
```

計算の結果 $\{\{2, 2\}, \{5, 1\}\}$ は、 $2^2 \times 5^1$ を表しています。

▶ 約数・倍数

約数を求めるには、Divisorsという関数を使います。例えば、6 の約数を求めてみましょう。

```
Divisors[6]
```

```
{1, 2, 3, 6}
```

そのほかにも、最大公約数や最小公倍数を求める関数が用意されています。例えば、42 と 30 と 12 の最大公約数は、GCDを使って求めます。

```
GCD[42, 30, 12]
```

```
6
```

次に「2, 53, 21」の最小公倍数を求めてみましょう。最小公倍数は, LCMを使って,

```
LCM[2, 53, 21]
```

```
2226
```

となります。

▶ 乱数

確率などのシミュレーションを行うときに便利な、乱数の発生方法について説明しましょう。乱数の発生には、Randomという関数を使います。

```
Random[]
```

```
0.270094
```

ここで注意したいことは、BASIC などでは時間などを利用して乱数を発生させていましたが、*Mathematica* ではその必要がありません。また、同系列の乱数を発生させるためには、SeedRandomを使います。

```
SeedRandom[4];
```

```
Random[]
```

```
0.672212
```

```
Random[]
```

```
0.381237
```

第 1 章 Mathematica の基礎

```
SeedRandom[4];
```

```
Random[]
```

```
0.672212
```

```
Random[]
```

```
0.381237
```

同系列の乱数を発生させることができました。

さて、もう少し詳しく Random をみることにしましょう。-10 から 10 までの間の整数の乱数を発生させたいときには、

```
Random[Integer, {-10, 10}]
```

```
10
```

また、-1 から 1 の間の実数の乱数を発生させるには、

```
Random[Real, {-1, 1}]
```

```
0.328031
```

のようにします。また、以下のように複素数の乱数も可能です。

```
Random[Complex, {0, 1 + I}]
```

```
0.669248 + 0.156987 I
```

ここで、虚数単位 i は `[Esc] ii [Esc]` で入力可能です。当然のことながら、BasicInput パレットからも可能です。虚数単位の出力結果が I になっていますが、 i と同じ意味です。

▶ 桁数字の操作

数値の各桁を取り出す関数を紹介しましょう。まず、「523」から「5」、「2」、「3」を取り出してみましょう。整数値から各桁の数を得るには、IntegerDigitsを使います。

```
IntegerDigits[523]  
  
{5, 2, 3}
```

では、実数の場合はどうすればよいでしょうか？。先に説明した IntegerPart という関数を使って、 π の値から 10 桁分を取り出してみましょう。まず、 π を 10^9 倍してから IntegerPart を使って整数化します。その後、IntegerDigits を施します。

```
IntegerDigits[IntegerPart[109  $\pi$ ]]  
  
{3, 1, 4, 1, 5, 9, 2, 6, 5, 3}
```

上記の通り π の値の 10 桁を得ることができました。このように工夫してもよいのですが、実は RealDigits という関数が用意されています。

```
RealDigits[N[ $\pi$ , 10], 10, 10]  
  
{{3, 1, 4, 1, 5, 9, 2, 6, 5, 4}, 1}
```

上の結果と表示が少し違ってしまいました。これは、ご想像のとおり四捨五入による誤差です（Ver 4.0 からは修正されています）。そこで、一桁多めにとっておくと同じ結果となります。

```
RealDigits[N[ $\pi$ , 11], 10, 10]  
  
{{3, 1, 4, 1, 5, 9, 2, 6, 5, 3}, 1}
```

次に、数字のリストからもとの数値に戻す逆の操作を説明しましょう。数値の

第 1 章 Mathematica の基礎

リストから数値を作るには , `FromDigits` という関数を使います。

```
FromDigits[{5, 2, 3}]  
  
523
```

▶ 進法の変換

今まで , 10 進数で考えてきましたが , 2 進数でも同様に操作することが可能です。例えば , $10!$ を 2 進数で表現し , それをバラバラにしてみましょう。まず , $10!$ を `BaseForm` を使って 2 進数で表現すると ,

```
BaseForm[10!, 2]  
  
11011101011111000000002
```

となります。 `IntegerDigits` を使って , 2 進数の各桁を取り出すことも可能です。

```
IntegerDigits[10!, 2]  
  
{1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,  
 0, 0, 0, 0, 0, 0}
```

2 番目の引数で基数を指定しています。また , 2 進数表現された数値リストから 10 進数の数値へ変換するには , `FromDigits` を用いて ,

```
FromDigits[{1, 0, 0, 1}, 2]  
  
9
```

とします。実数についても同じように入力します。

```
RealDigits[N[ $\pi$ , 21], 10, 20]
```

```
{ {3, 1, 4, 1, 5, 9, 2,  
  6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4},  
  1 }
```

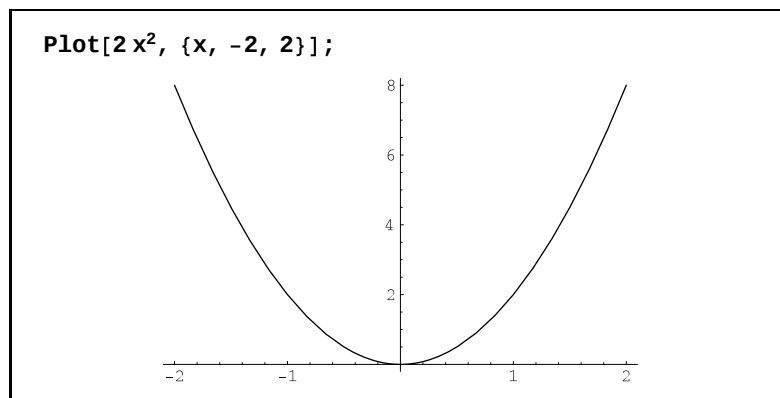
2 番目の引数が基数を表し, 3 番目の引数が桁数を表しています。

2 グラフの描画

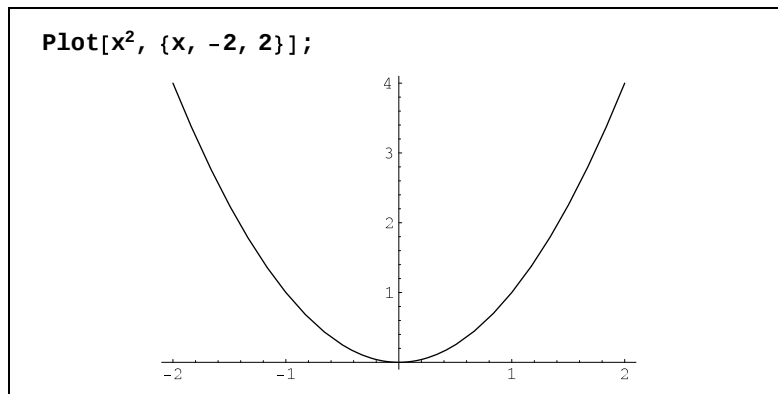
BASIC などではグラフを描くにはプログラムを組まなくてはなりません。Mathematica では、グラフを描くために Plot という関数が用意されています。

▶ グラフを描く

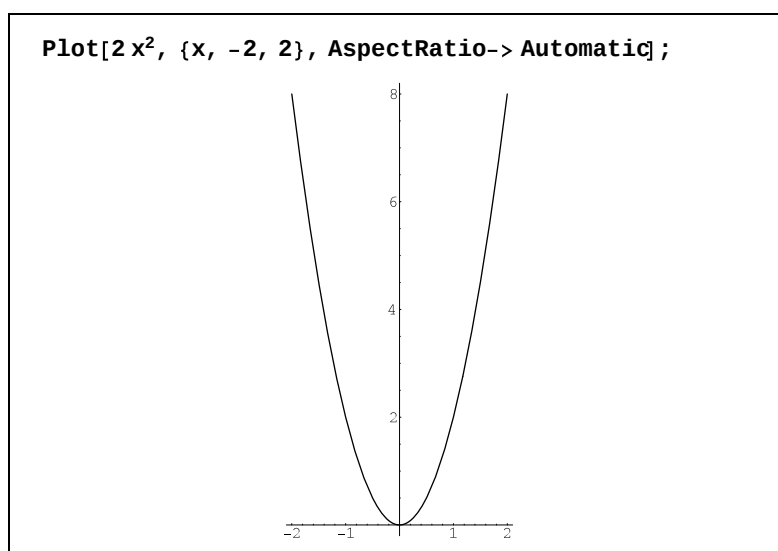
中学校では、一次関数と原点を通る二次関数を学習し、高校では、そこに、三角関数、指数関数、対数関数に加わり、その平行移動を考えるようになります。Mathematica でそれらのグラフを描かせてみましょう。例えば、 $y = 2x^2$ のグラフは、



となります (“ x^2 ” はパレット BasicInput から入力できます。また、指数は **Ctrl** + **6** または **Ctrl** + **^** で入力できます)。さて、次の $y = x^2$ のグラフを見てください。グラフの開き具合が上のグラフと同じになってしまいました。



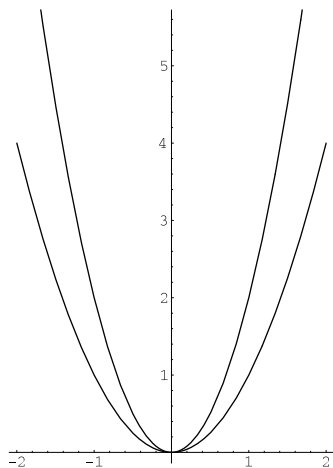
そこで , 次のように AspectRatio -> Automatic というオプションを付け加えてみましょう。



思い通りにグラフを描くことができました。では次に , 別々に描かれた 2 つのグラフを重ねてみましょう。

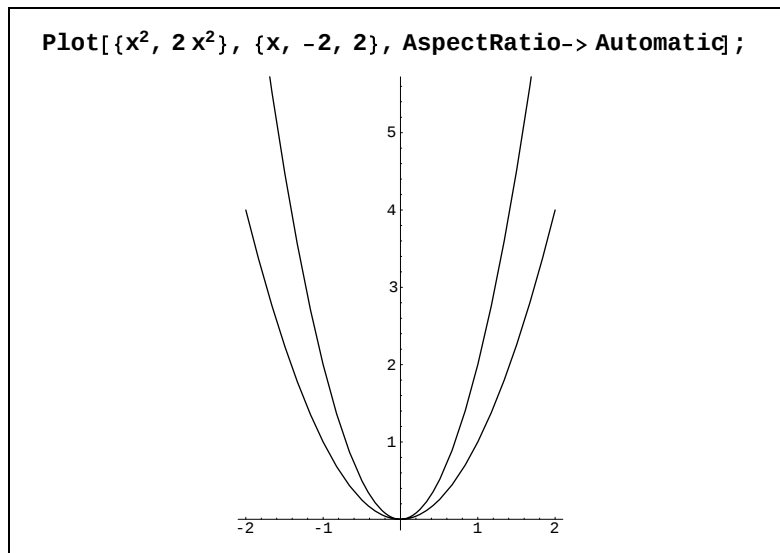
第 1 章 Mathematica の基礎

```
Show[%, %%];
```



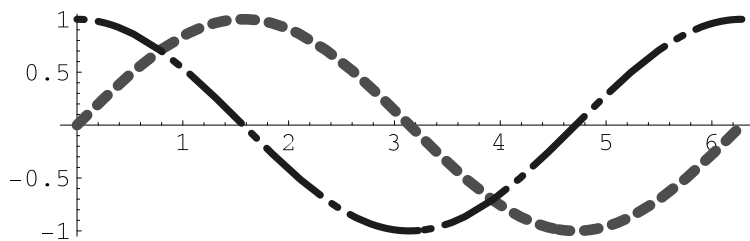
%は、直前の出力結果を、%%は、2 つ前の出力結果を表します。また、「%5」と記述すれば、Out [5] の結果を表します。

同時に 2 つのグラフを重ねて表示させたいときには、通常は次のように、 x^2 のところを $\{x^2, 2x^2\}$ と $\{\}$ で描きたいグラフの式をくくります。当然、3 つでも 4 つでも可能です。では、実際に入力してみます。



一度に描くことができました。でも、グラフが多くなると見にくくなります。そこで、グラフを破線 (点線) や鎖線にしたり、色をつけたりしてみましょう。それには `PlotStyle` を使います。

```
Plot[{Sin[x], Cos[x]}, {x, 0, 2  $\pi$ },  
  AspectRatio-> Automatic,  
  PlotStyle-> {  
    {RGBColor[1, 0, 0],  
     Thickness[0.015],  
     Dashing[{0.02, 0.02}]},  
    {RGBColor[0, 0, 1],  
     Thickness[0.01],  
     Dashing[{0.01, 0.02, 0.1, 0.02]}}];
```



```
PlotStyle{{ 第 1 式のグラフのオプション },  
          { 第 2 式のグラフオプション }}
```

という指定になります。オプションの中身については、

RGBColor[r,g,b]: 色指定r,g,b はそれぞれ、0 ~ 1 の値

Thickness [t]: 線の太さを指定t は、0 ~ 1 の値

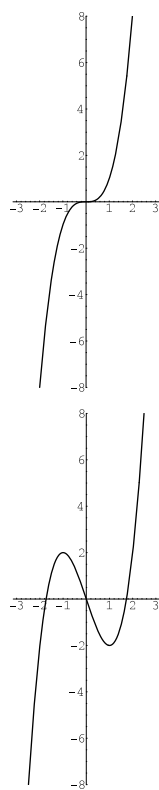
Dashing [{a,b,...}]: 点線の指定a,b はそれぞれ、0 ~ 1 の値

を表しています。

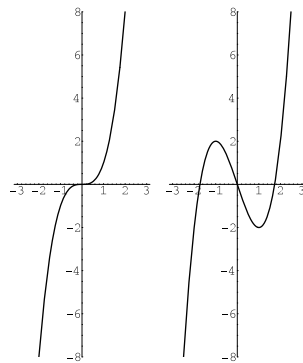
▶ グラフを並べて描く

次に、グラフを重ねるのではなく、2つのグラフを横に並べて比較してみましょう。GraphicsArrayという関数を使います。

```
g1 = Plot[x3, {x, -3, 3},
  PlotRange-> {-8, 8},
  AspectRatio-> Automatic ];
g2 = Plot[x3 - 3 x, {x, -3, 3},
  PlotRange-> {-8, 8},
  AspectRatio-> Automatic ];
Show[GraphicsArray[{g1, g2}]];
```



第 1 章 Mathematica の基礎



▶ アニメーション

二次関数のグラフの平行移動や三角関数の平行移動などを観察するとき、アニメーションで連続的に動きを見るとわかりやすくなります。例えば、 $y = \sin x$ のグラフを x 軸方向に平行移動させて、 $y = \cos x$ のグラフと重なる様子を観察したいときには、次のように入力します。すると、複数のグラフが表示されます。

```
Table[Plot[{Sin[x - t], Cos[x]}, {x, 0, 2  $\pi$ },  
  AspectRatio -> Automatic,  
  PlotStyle -> {  
    {RGBColor[1, 0, 0],  
     Thickness[0.015],  
     Dashing[{0.02, 0.02}]},  
    {RGBColor[0, 0, 1],  
     Thickness[0.01],  
     Dashing[{0.01, 0.02, 0.1, 0.02]}}},  
  {t, 0, 2  $\pi$ , 0.1}];
```

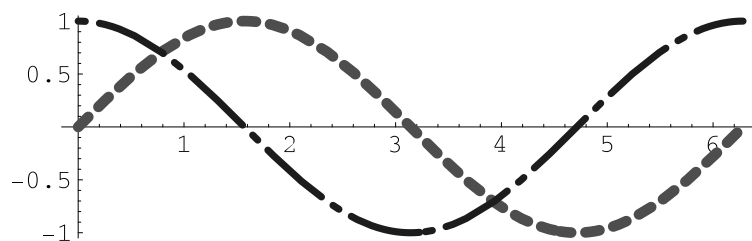
描かれたグラフの一つをダブルクリックすると、グラフが動き始めるはずです(ここで使った関数 Table については、p.34 で説明します)。

次に、グラフを描き終わると自動的にグラフをアニメートする方法も紹介しておきましょう。

```
<< Graphics`Animation`
<< ProgrammingInMathematica`AutoAnimation`
```

というパッケージを使います（パッケージは、上記のように入力して数式と同様に実行すると、`Out[**]` は表示されませんが、パッケージが適用された状態になります）。`Animate` という関数を使って、次のように入力します。

```
Animate[Plot[{Sin[x - t], Cos[x]}, {x, 0, 2  $\pi$ },
  AspectRatio -> Automatic,
  PlotStyle -> {
    {RGBColor[1, 0, 0],
     Thickness[0.015],
     Dashing[{0.02, 0.02}]},
    {RGBColor[0, 0, 1],
     Thickness[0.01],
     Dashing[{0.01, 0.02, 0.1, 0.02]}]}],
  {t, 0, 2  $\pi$ , 0.1}];
```



上図のように自動的にアニメートされます。区間によって関数を変えてグラフを描くには p.123 の `Which` を参照してください。

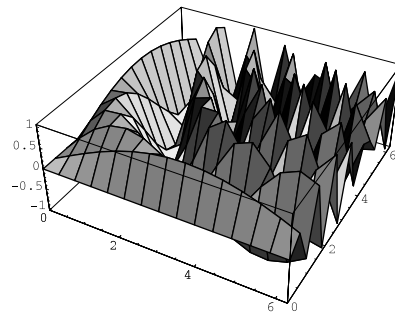
▶ 3次元のグラフ

中学や高校では直接3次元のグラフを扱うことはありませんが、*Mathematica* で

第 1 章 Mathematica の基礎

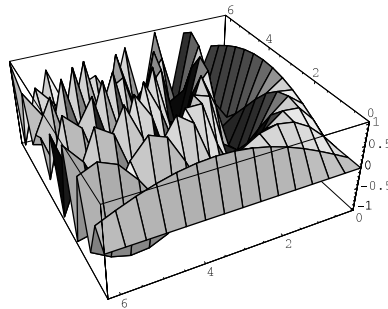
は簡単に描くことができます。

```
Plot3D[Sin[x y], {x, 0, 2  $\pi$ }, {y, 0, 2  $\pi$ ];
```



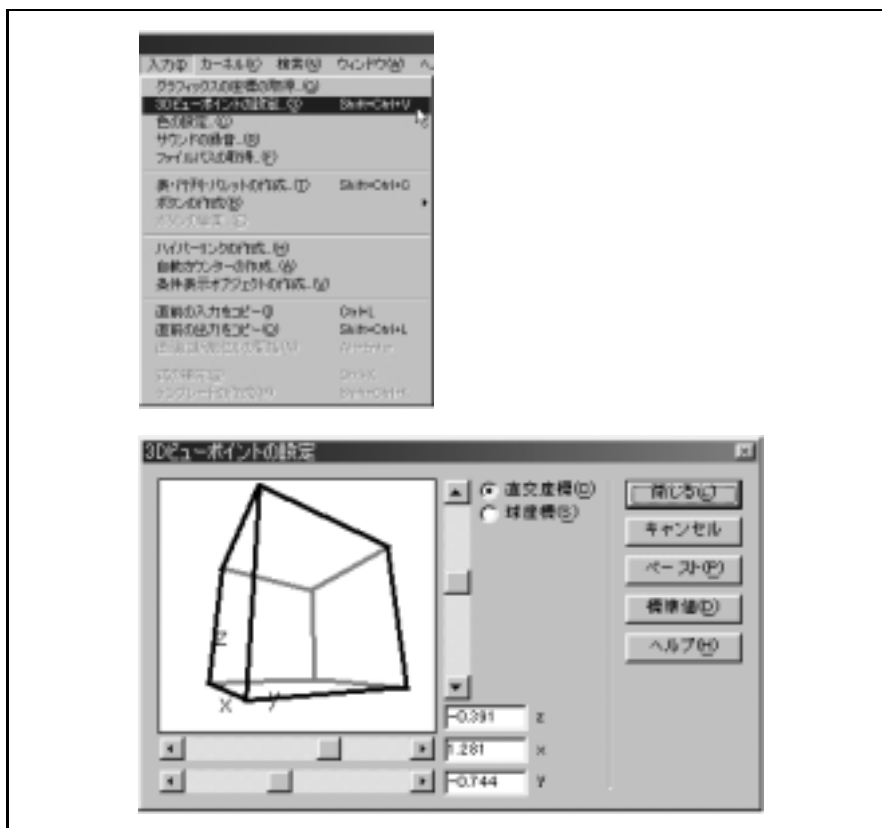
グラフを見る位置を変えることも可能です。では、視点を変えてみましょう。

```
Plot3D[Sin[x y], {x, 0, 2  $\pi$ }, {y, 0, 2  $\pi$ },  
ViewPoint->{-2, 1, 1.5}];
```



視点を変えるには、ViewPoint -> {-2, 1, 1.5}のように、見る位置の座標を与えます。また、座標で与えるのではなく、単にグラフを別の角度から見たいだけであれば、図1.2のように入力メニューから 3D ビューポイントの設定を選びます。ダイアログボックス内の座標系を直交座標にし、座標軸をマウスでドラッグして好きな角度にしてペーストボタンを押せば、ViewPoint が挿入されます。

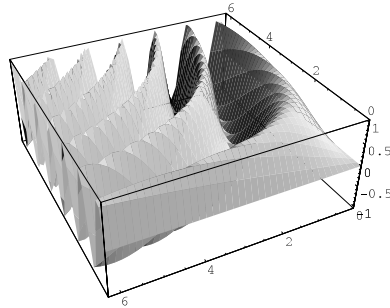
図 1.2 3D ビューポイントの設定



次に、グラフの網目をとって、もう少し滑らかに描いてみましょう。網目をとるには Mesh -> Falseというオプションを指定し、滑らかにするためには PlotPoints -> 40というオプションを指定します。PlotPoints の値を 40 より大きくすればさらに滑らかになりますが、グラフを表示するための時間とメモリをより多く必要とします。

第 1 章 Mathematica の基礎

```
Plot3D[Sin[x y], {x, 0, 2  $\pi$ }, {y, 0, 2  $\pi$ },  
PlotPoints->40,  
Mesh->False,  
ViewPoint->{-2, 1, 1.5}];
```

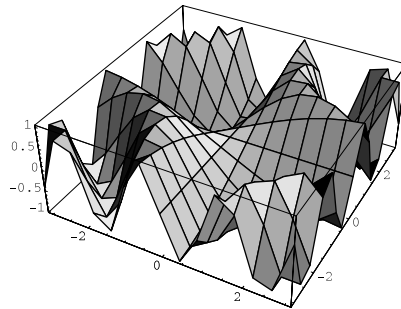


また, Ver 4.0 から, マウスで視点を変えることができるようになっています。
例えば, RealTime3D というパッケージを読み込みます。

```
<< RealTime3D'
```

Plot3D を使って, 3 次元のグラフを表示します。

```
Plot3D[ Sin[x y], {x, - $\pi$ ,  $\pi$ }, {y, - $\pi$ ,  $\pi$ }];
```



グラフをクリックして, マウスの左ボタンを押しながらポインタを動かしてく

ださい。それにとまって、グラフが回転するはず。また、`Ctrl` キーとマウスの左ボタンを押しながらマウスを動かすと、拡大・縮小できます。

注意：RealTime3D パッケージを読み込んだ後の 3D グラフは、印刷できないので注意が必要です。デフォルトの 3D パッケージに戻すには、Default3D というパッケージを読み込みます。

```
<< Default3D'
```

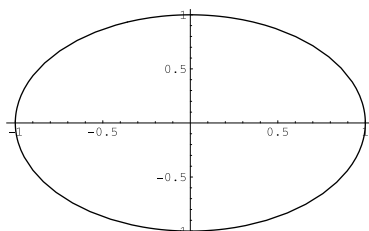
▶ 媒介変数表示のグラフ

媒介変数表示された曲線の方程式をグラフにしてみましょう。まず、お馴染みの円のグラフを描いてみましょう。中心が原点で半径 1 の円は、

$$\begin{cases} x = \cos t \\ y = \sin t \end{cases}$$

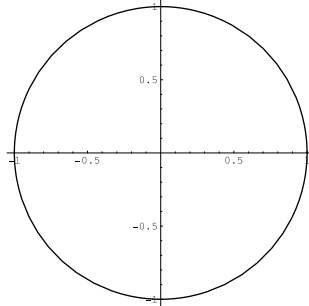
と表されます。円の表示には ParametricPlot という関数を使います。

```
ParametricPlot[Cos[t], Sin[t]], {t, 0, 2 π}];
```



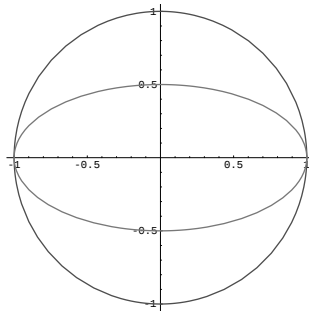
これは、円の方程式のはずですが、楕円になってしまいます。Plot のときと同様に、AspectRatio -> Automatic をオプションとして付け加えてください。

```
ParametricPlot[Cos[t], Sin[t]], {t, 0, 2  $\pi$ },  
AspectRatio-> Automatic];
```



円になりました。では, いろいろなグラフを描いてみましょう。Plot のときと同様引数を $\{\{x_1(t), y_1(t)\}, \{x_2(t), y_2(t)\}\}$ のように与えると, 複数のグラフを描くことができます。色の付け方も Plot のときと同様 PlotStyle を利用します。

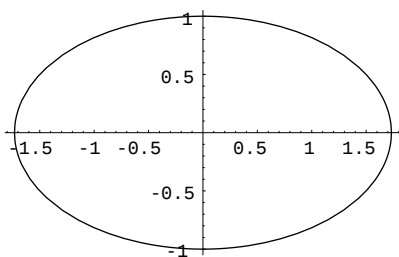
```
ParametricPlot[{Cos[t], Sin[t]],  
{Cos[t],  $\frac{1}{2}$  Sin[t]}], {t, 0, 2  $\pi$ },  
AspectRatio-> Automatic,  
PlotStyle-> {RGBColor[1, 0, 0],  
RGBColor[0, 0.8, 0]}];
```



▶ 陰関数のグラフ

例えば, $x^2 + 3y^2 = 3$ というグラフを描こうとすると, $y = \pm \frac{\sqrt{3-x^2}}{\sqrt{3}}$ として,

```
Plot[{- $\frac{\sqrt{3-x^2}}{\sqrt{3}}$ ,  $\frac{\sqrt{3-x^2}}{\sqrt{3}}$ }, {x, - $\sqrt{3}$ ,  $\sqrt{3}$ }]
```

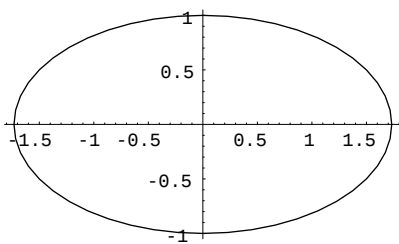


グラフを描くことができます。けれども, できれば $x^2 + 3y^2 = 3$ を与えて, グラフを描いてもらいたいものです。ImplicitPlotというパッケージを使えば可能です。

```
<< Graphics`ImplicitPlot`
```

では, $x^2 + 3y^2 = 3$ というグラフを描いてみましょう。

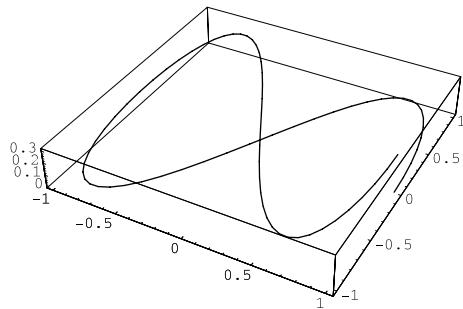
```
ImplicitPlot[x2 + 3y2 == 3, {x, -2, 2}]
```



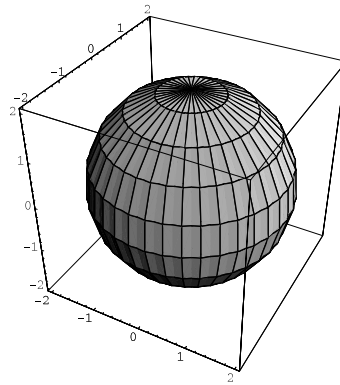
▶ 3 次元のグラフ

媒介変数表示された 3 次元の曲線や曲面の方程式のグラフも，中学校や高校で直接扱うことはありませんが，簡単に描くことができるので，いくつか例を挙げておくことにします。

```
ParametricPlot3D[{Cos[t], Sin[2 t], 0.05 t},  
  {t, 0, 2  $\pi$ }] ;
```



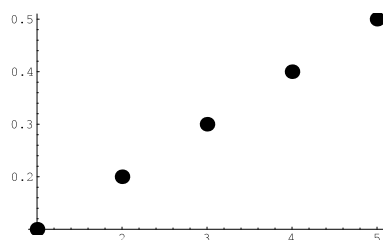
```
ParametricPlot3D[{2 Sin[t] Cos[s],  
  2 Sin[t] Sin[s], 2 Cos[t]},  
  {t, 0, 2  $\pi$ }, {s, 0, 2  $\pi$ }] ;
```



▶ 点列のプロット

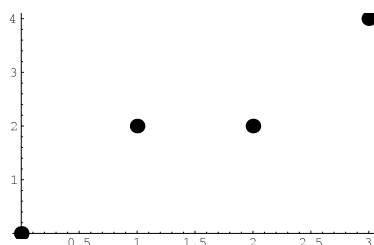
点列のプロットは、後に出てくる数列や統計データのプロットなどを行うときに便利です。また、複素数平面上の点をプロットするときにも使うことが可能です。ここでは簡単な使い方を挙げておきましょう。

```
ListPlot[{0.1, 0.2, 0.3, 0.4, 0.5},  
PlotStyle -> PointSize[0.02]];
```



プロットする点を $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ で与えると、プロットする座標は、 $(1, 0.1), (2, 0.2), (3, 0.3), (4, 0.4), (5, 0.5)$ となります。当然、 (x, y) の座標で与えることもできます。次の例は、 $(0, 0), (1, 2), (2, 2), (3, 4)$ という座標を与えた場合です。

```
ListPlot[{{0, 0}, {1, 2}, {2, 2}, {3, 4}},  
PlotStyle -> PointSize[0.02]];
```



3 リストの操作

▶ Append

リストに要素を付け加えるには、Appendを使います。

```
lst = {a, b, c};  
Append[lst, d]  
  
{a, b, c, d}
```

Append を実行しても、lst の内容は変更されません。つまり、lst に「d」を Append しても

```
lst  
  
{a, b, c}
```

のように、 $\{a, b, c, d\}$ とはなりません。lst の内容を $\{a, b, c\}$ から、 $\{a, b, c, d\}$ に変更するには、AppendToを使います。

```
AppendTo[lst, d]  
  
{a, b, c, d}  
  
lst  
  
{a, b, c, d}
```

▶ Sort

Sortにリストだけを与えると、数値の場合は昇順、文字列の場合は辞書順にリストを並べ替えます（ただし、現在のところ日本語は正しく処理できません）。

```
Sort[{3, 2, 6, 4, 7, 1}]  
  
{1, 2, 3, 4, 6, 7}  
  
Sort[{b, a, d, z, ab, ca, acb}]  
  
{a, ab, acb, b, ca, d, z}
```

Sort にはオプションを指定することができます。降順指定には Greater を用い、次のように表現することができます。

```
Sort[{3, 2, 6, 4, 7, 1}, Greater]  
  
{7, 6, 4, 3, 2, 1}
```

昇順指定には Less を用います。

```
Sort[{3, 2, 6, 4, 7, 1}, Less]  
  
{1, 2, 3, 4, 6, 7}
```

► Flatten

Flatten はリストの階層を変えたいときに用います。例えば、 $\{\{a, b\}, \{c, d\}, \{\{e\}\}\}$ のように $\{\}$ が二重や三重になっている場合、内側の $\{\}$ をすべて取るには、

```
Flatten[{{a, b}, {c, d}, {{e}}}]  
  
{a, b, c, d, e}
```

とします。また、1 階層だけ $\{\}$ をはずすには、

```
Flatten[{{a, b}, {c, d}, {{e}}}, 1]  
  
{a, b, c, d, {e}}
```

のように 2 番目の引数に階層数を指定します。

► Take

Take は、リストの先頭から指定された個数だけ要素を取り出すときに使います。

```
Take[{a, b, c, d}, 3]  
  
{a, b, c}
```

リストの末尾から順番に要素を取り出したい場合には、指定する個数に “ - ” をつけて実行します。

```
Take[{a, b, c, d}, -2]  
  
{c, d}
```

► Select

Select は、リストの中から型（パターン：p.38 参照）の一致した要素を取り出すときに使います。例えば、与えられたリストの中から偶数だけを取り出すには、次のように入力します。

```
Select[{a, 23, 12, 0, 3.5}, EvenQ]  
  
{12, 0}
```

► Drop

Drop は、リストの先頭から指定された個数だけ要素を取り除くときに使います。

```
Drop[{a, b, c, d}, 2]  
  
{c, d}
```

Take 同様，リストの末尾から指定された個数分だけ要素を取り除くには，指定する個数に “ - ” をつけます。

```
Drop[{a, b, c, d}, -2]  
  
{a, b}
```

► Delete

Delete は，リストの中から指定された場所の要素を取り除くときに使います。最初から 2 番目の要素を削除するには，次のように入力します。

```
Delete[{a, b, c, d}, 2]  
  
{a, c, d}
```

最後から 2 番目の要素を削除するには，次のように入力します。

```
Delete[{a, b, c, d}, -2]  
  
{a, b, d}
```

► First

関数 First は，リストから先頭の要素を取り出します。

```
First[{a, b, c, d}]  
  
a
```

► Last

関数 Last は，リストから末尾の要素を取り出します。

第 1 章 Mathematica の基礎

```
Last[{a, b, c, d}]
```

```
d
```

▶ MemberQ

リストに、ある要素が含まれているかどうかを調べるときに用います。例えば、リスト $\{a, b, c, d\}$ の中に a が含まれているかどうかを調べるには、次のように入力します。

```
MemberQ[{a, b, c, d}, a]
```

```
True
```

また、 $\{a, b, c, d\}$ の中にリスト形式の $\{a\}$ が含まれているかどうかを調べるには、次のように入力します。

```
MemberQ[{a, b, c, d}, {a}]
```

```
False
```

▶ Position

指定した文字や数値、パターン (p.38 を参照) が、リストのどの位置に存在するかどうかを知りたいときに Position を用います。例えば、「 a 」がリストの何番目の位置にあるかを調べるには、次のように入力します。

```
Position[{a, b, a, c, d}, a]
```

```
{{1}, {3}}
```

また、「□□」のような型のものに対しては、“ $_$ ”(アンダーバー)を用いて、“ $_$ ”や“ $_$ ”を使って、次のように指定することも可能です。

```
Position[{x, x2, a5, y}, _^_]
```

```
{{2}, {3}}
```

4 繰り返し

▶ Table

繰り返しの操作といえば、For、Do、Whileなどを思い浮かべますが、操作の結果がリスト形式で返されるので、MathematicaではTableがよく使われるようです。Table[操作,{カウンタ,初期値,終了値}]の形で使います。

```
Table[n, {n, 1, 10}]  
  
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

カウンタのステップを変えることも可能です。次の例は、ステップを2に設定した例です。

```
Table[n, {n, 1, 10, 2}]  
  
{1, 3, 5, 7, 9}
```

この例では単なる数値の生成にTableを使いましたが、数値のリストを生成するときには、Rangeがよく使われます。

```
Range[10, 20, 3]  
  
{10, 13, 16, 19}
```

Tableを使った一連の操作を繰り返す例もあげておきましょう。

```
l = {};  
Table[AppendTo[l, Prime[n]];  
Print[l];  
 , {n, 1, 5}];  
  
{2}  
  
{2, 3}  
  
{2, 3, 5}  
  
{2, 3, 5, 7}  
  
{2, 3, 5, 7, 11}
```

► For

Table と同様、繰り返す回数がわかっているときにもっとも使われるのが、この For だと思います。

```
For[i = 1, i < 5, Print[i]; i++]  
  
1  
  
2  
  
3  
  
4
```

Table と同じ出力を得るには、Print[i] の部分にリストの操作を記述します。