学習院大学理学部数学科

ステップ･バイ･ステップ式

はじめての Maple

基礎編

数学講話 1

▼目次

- 変数の扱い方
    - 変数名の作り方
        - 命名規則の注意事項
        - 数式入力におけるアンダースコア（ _ ）の入力
    - 変数への割り当て（代入）
    - 変数の初期化（リセット）
    - 結果の再利用
- 厳密解と数値解
- オンラインヘルプの活用
    - Maple 13 のヘルプメニュー
    - コマンドからヘルプを起動する方法（例題の実行）
- 応用事例
    - Maplesoft Application Center
    - Maple デモ＆ギャラリー

▼ 運動を見る！

物体をただ落としてみる（自由落下運動）

等加速度直線運動

自由落下運動

こんどは物体を上に投げてみる（投げ上げ運動）

式のパラメータ化

v0 = 10 のとき

v0 = 20 のとき

v0 = 30 のとき

結果のプロット

アニメーションの作成

投げ上げ運動のアニメーション作成

初速度 v0 = 10 の場合

初速度 v0 = 20 の場合

初速度 v0 = 30 の場合

同時アニメーションの作成

ついでに斜めにも投げてみる（斜め投げ上げ運動）

▼ ワークシートを操る！

ツールバー（Windows 及び Mac OS 共通）

ファイル操作と編集

グループの入力とセクション

計算の実行と中断，デバッグなど

その他

ワークシート向けのショートカット（Windows）

入力とグループ向けのショートカット（Windows）

セクションとサブセクション向けのショートカット（Windows）

典型的なセクション内の構成

セクション－サブセクション－サブサブセクション

セクション

サブセクション

サブサブセクション

セクション

サブセクション

サブサブセクション

セクション

サブセクション

サブサブセクション

スタイルの設定

変更手順

設定するスタイル

ヘッダとフッタ

▼ Maple の仲間たち

Maple の予約語

数学定数

グローバル変数（大域的変数）

円周率 π とギリシャ文字 π

複素数

複素数の定義と四則演算（１）

複素数の定義と四則演算（２）

複素平面へのプロット

式の性質を確認

（参考）三角関数の指数関数表記

多項式

１変数多項式の操作

２変数の多項式を生成

方程式

１変数の代数方程式

１次方程式

２次方程式

３次方程式

４次方程式

RootOf() の解

５次方程式

高次方程式の数値解（近似解）

円の方程式

楕円の方程式

双曲線の方程式

連立代数方程式の解法とプロット

放物線と楕円の交点

（復習）RootOf の解

数値解の求め方

要素の操作

関数

三角関数

三角関数の基本公式

（参考）第１式から第２式を導く手順

加法定理

双曲線関数

指数と指数関数

指数の定義

指数関数

対数と対数関数

対数の底

常用対数

対数関数

自然対数

区分関数

文字列

配列

１次元配列

２次元配列

３次元配列

２次元配列から４次元配列への拡張

シーケンス（数列，式列）

シーケンスの定義

要素の指定

その他の数列，式列，データの定義例

リスト

リストの定義

要素の抽出

シーケンスのリスト

行列

ベクトル

集合（セット）

ブーリアン

ベン図

論理演算

型（type）

Maple オブジェクトの型（type）

変数型の仮定

主な仮定

仮定の解除（キャンセル）

ステップ･バイ･ステップ式 **はじめての Maple** 基礎編

Maple を使う前に

Maple の基本的な操作方法を習得します.

目次

Maple の起動

Mapleには 3 つのインターフェイスが用意されています.

• スタンダードインターフェイス（推奨）

Maple のすべての機能を利用可能な Java ベースのインターフェイスです.

• クラシックインターフェイス

基本機能のみのインターフェイスです.

• コマンドラインインターフェイス

コマンドプロンプトから利用する計算機能のみのインターフェイスです.

（注意） OS によっては用意されていないインターフェイスもあります.

スタンダードインターフェイスの起動方法は下の表を参考にして下さい.

Windows	デスクトップのアイコンをダブルクリック	
Macintosh	Finder で /Applications/Maple 12 へ移動しアイコンをダブルクリック	
Linux/UNIX	ターミナルを起動し，$Maple/bin の xmaple を実行	なし

編集モード

- ２つの編集モードがあります．
- ワークシートモードとドキュメントモードがあります．
- 数式処理・ドキュメント作成はどちらでも可能です．

ワークシートモード

- 主にコマンド入力・プログラミング向けの編集モードです．
- プロンプト（＞）から処理を実行します．
- 割り算（$\frac{a}{b}$）はスラッシュ（ / ）を使用します（ a/b ）．
- 累乗（x^2など）はキャレット（ ^ ）を使用します（ x^2 ）．

> $\frac{\sin(x)}{x}$

$$\frac{\sin(x)}{x} \qquad \textbf{(2.1.1)}$$

> $a x^2 + b x + c = 0$

$$a x^2 + b x + c = 0 \qquad \textbf{(2.1.2)}$$

入力方法を切り替えることができます（ただし，処理内容及び結果は同じです）．

入力前に，F5 キーを押します．

```
> sin(x)/x;
```

$$\frac{\sin(x)}{x} \qquad \textbf{(2.1.3)}$$

```
> a*x^2 + b*x + c = 0;
```

$$a x^2 + b x + c = 0 \qquad \textbf{(2.1.4)}$$

ドキュメントモード

- 主にレポートや技術文書作成向けの編集モードです．
- ワークシートモードの文書作成機能を拡張した編集モードです．
- プロンプト（＞）は非表示です（設定によりプロンプト表示も可能です）．
- 数式入力が基本になります．

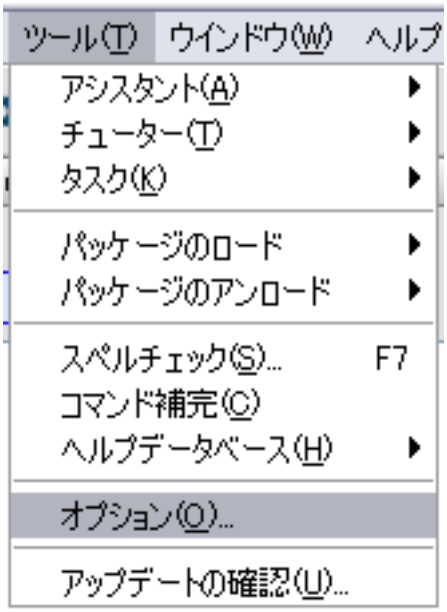
$$\frac{\sin(x)}{x}$$

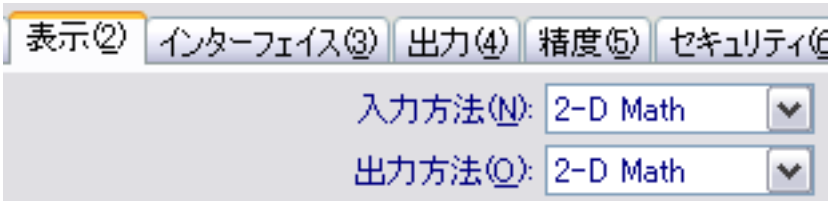
$$a\,x^2 + b\,x + c = 0$$

入力方法と編集モードの設定

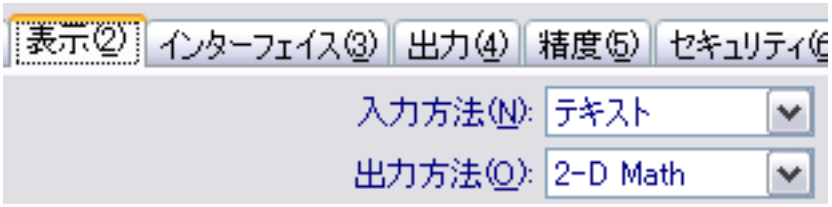
通常の入力方法を設定します（デフォルトの入力方法を設定します）

［ツール］メニューから［オプション］を選択し，［表示(2)］タブ内から［入力方法］を設定します.





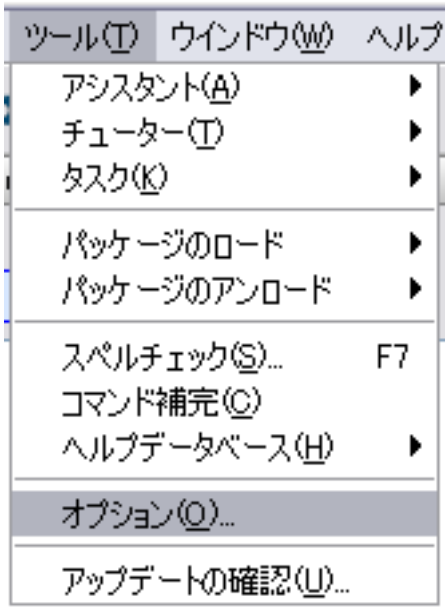
ここでは，［入力方法］を<テキスト>に設定します.

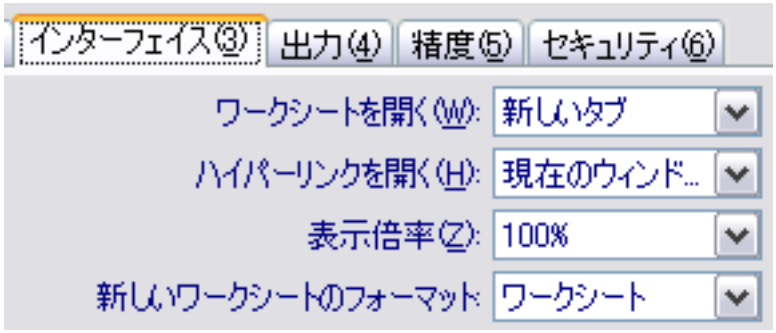


必要に応じて，続けて編集モードの設定をします．通常の編集モードを設定します（デフォルトの編集モードを設定します）

［ツールメニュー］から［オプション］を選択し，［インターフェース(3)］タブ内から［新しいワ

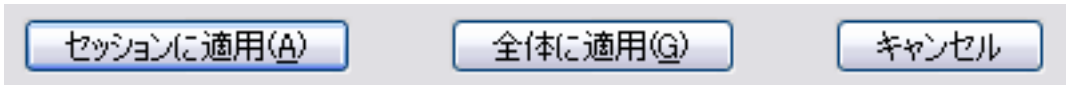
ークシートのフォーマット］を［ワークシート］に設定します．





すでに［ワークシート］になっている場合は，そのままにします．

設定変更後，［全体に適用］をクリックし，これ以降の作業に設定を反映させます．



インターフェースの構成

一般的なインターフェースの種類

引用元 IT用語事典・e-Words （ http://e-words.jp/ ）

グラフィカルユーザインターフェース

- Graphical User Interface，GUI.
- グラフィカルユーザインターフェースとは，ユーザに対する情報の表示にグラフィックを多用し，大半の基礎的な操作をマウスなどのポインティングデバイスによって行なうことができるユーザインターフェースのこと.
- 最近では GUI を利用するための基本的なプログラムをOSが提供することにより，アプリケーションソフトの操作感の統一や，開発負担の軽減などが図れている.
- GUI を実装した OS には，Microsoft 社の Windows シリーズや Apple 社の Mac OS などがある.
- UNIX 系の OS は，OS 自体は GUI 機能を持っていないことが多いが，X Window System というソフトウェアを組み込んで GUI 機能を追加する場合がある.

コマンドインターフェース

- Command User Interface, CUI
- コマンドユーザインターフェースとは，ユーザに対する情報の表示を文字によって行ない，すべての操作をキーボードを用いて行なうユーザインターフェースのこと.
- UNIX 系 OS の操作方式として一般的なほか，Windows なども CUI 画面を呼び出して操作することができる.
- 画面上に命令の入力を促すプロンプトと呼ばれる文字列が表示され，ユーザがそれに続けてキーボードからコマンド(命令)を入力し，コンピュータに指示を与える.
- コンピュータはユーザの入力の次の行から処理過程や結果を出力し，再び入力が可能な状態になると改行して行頭にプロンプトを表示する.
- この繰り返しで対話式に作業を進めていくのが CUI である.

Maple のユーザインターフェース

5つに区分されます.

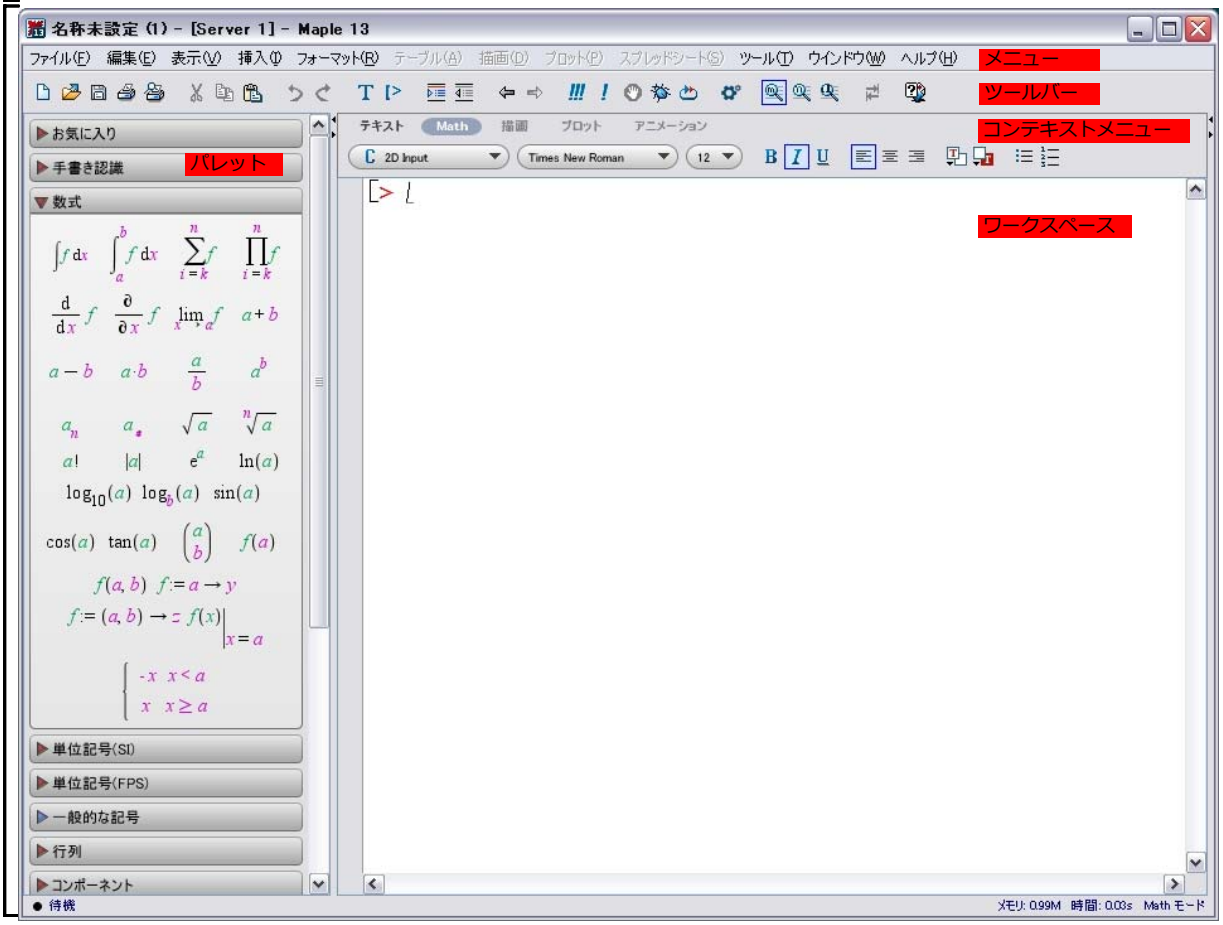
1. メニュー
ファイル操作，シート編集，あるいはオプションなどを設定します.

2. ツールバー
主なメニュー機能をアイコンから使用できます.

3. コンテキストメニュー
編集・選択されている対象によって表示が変わります.

4. パレット
煩雑な数式などの記号入力をサポートします.

5. ワークスペース
処理を入力するスペースです．結果も表示されます.



入力方法

グループ

ある処理（コマンド）や文章（テキスト）のまとまりを表します（コマンドやテキストが混在したグループも入力可能です）．左側の角括弧（ [）がひとつのグループになります．グループを用いて，実行グループ（処理の単位）やテキストグループ（文章の単位）などを作成します．左側の角括弧（ [）は，F9 キーで表示・非表示を切り替えることができます．

実行グループ

>

テキストグループ

（文章）

コマンドと文章が混在したグループ

（文章＝パラグラフ）

（文章＝パラグラフ）

（文章＝パラグラフ）

（参考）出力（結果）は，パラグラフの後に表示されます．

結果の表示・非表示

計算を実行する際に，結果の表示・非表示を指定することができます．

表示の場合は，セミコロン（ ; ）を入力の最後に追加します．

非表示の場合は，コロン（ : ）を追加します．

基本的に，入力は，セミコロン（ ; ）もしくはコロン（ : ）で閉じなければなりません．

コマンドの実行と入力

処理（コマンド）の入力は，プロンプト（>）から行います．文章（テキスト）の入力は，「入力方法の混在」を参照してください．コマンドは，ある特定の処理を Maple に実行させるための命令になります．通常，丸括弧 ( ) の中に必要な値や変数名を指定し，それらを引数と呼びます．引数は，コマンド（処理の内容）によって，ことなります．

```
>  コマンド名( 引数 );
```

コマンドを実行するには Enter キーを押します．実行後，カーソルは次の実行グループへ移動します．例えば，sin(x) を x について微分するコマンドは，以下のように入力します．

```
> sin(x);
```

$$\sin(x) \tag{4.3.1}$$

```
> diff( sin(x), x );
```

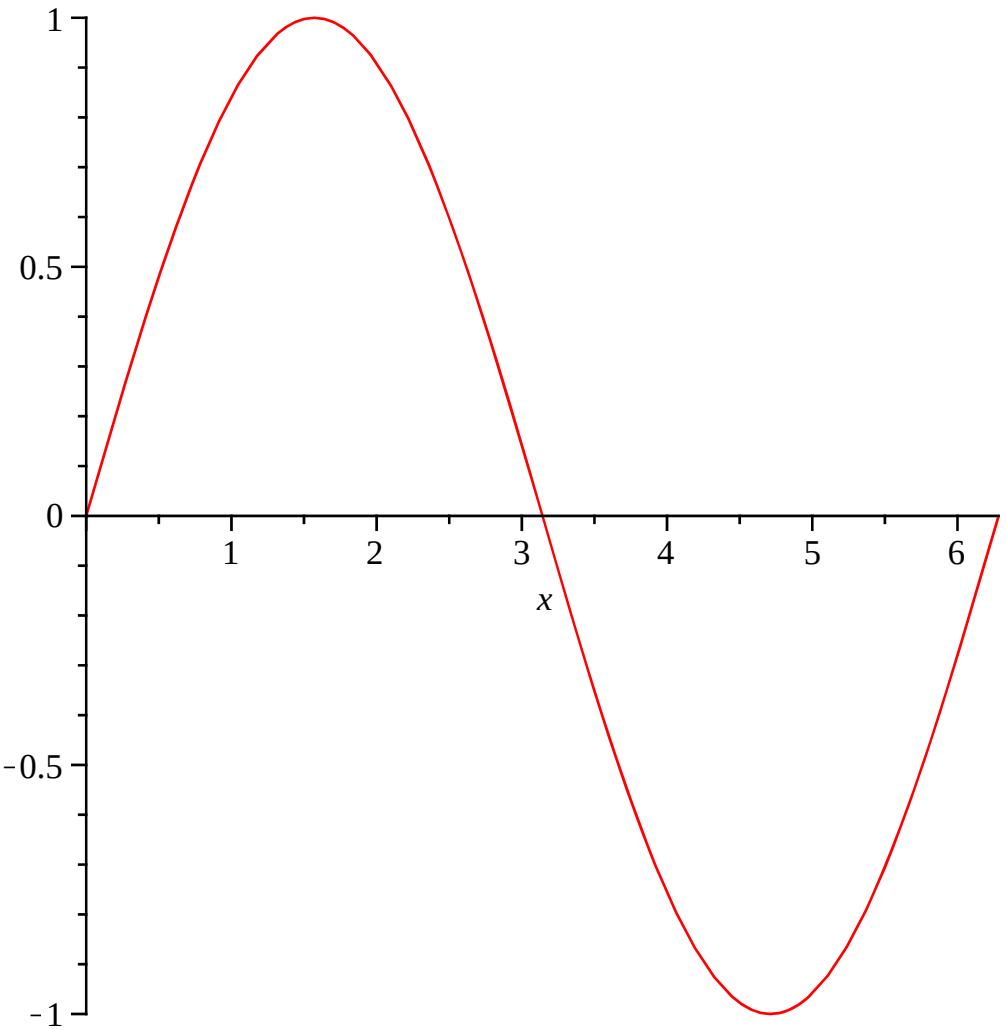
$$\cos(x) \tag{4.3.2}$$

引数は，Maple が解釈可能な式や値などを取り，コマンドと併せて処理に引き渡されます．必要に応じて，処理を拡張するためにオプションを用いるときがあります．

```
>  コマンド名( 引数, オプション );
```

例えば，sin(x) を $0 \leq x \leq 2\pi$ の範囲でグラフ表示する場合（通常，プロットする，と表現します），以下のように入力します．

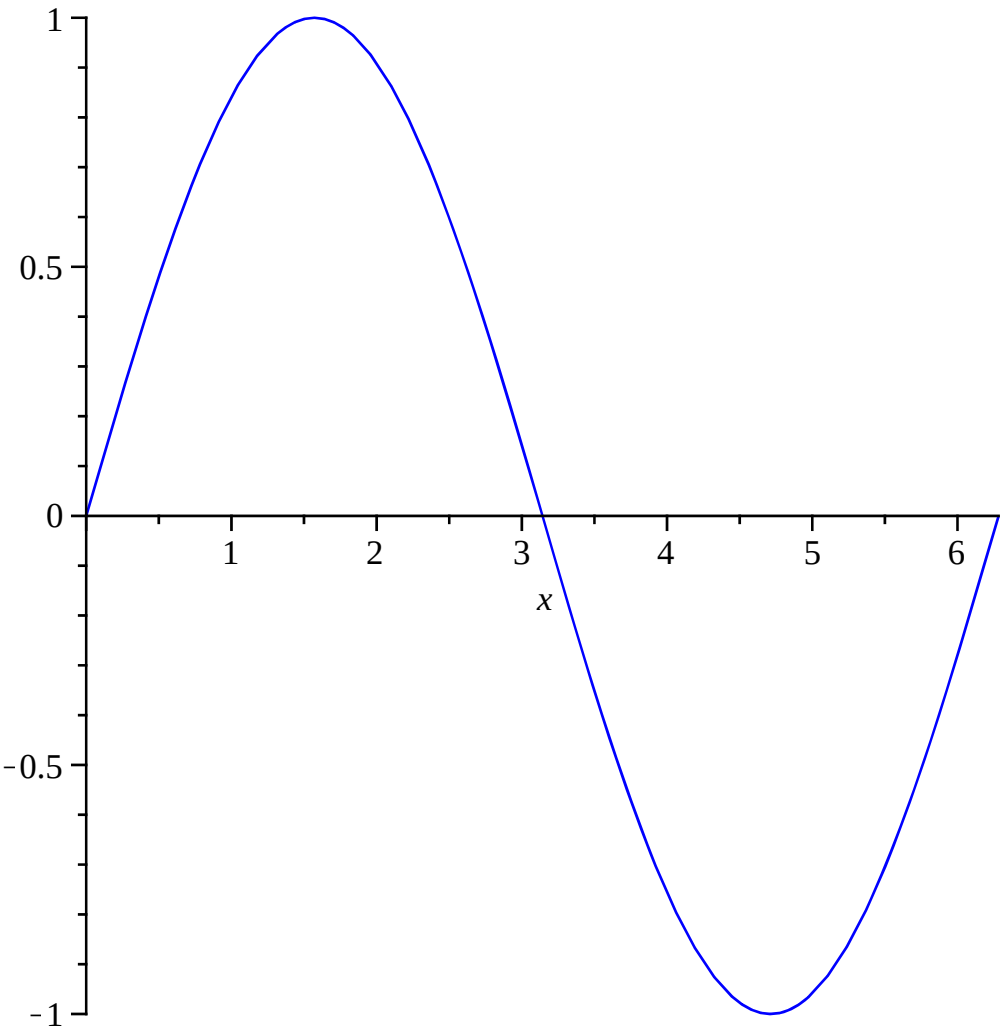
```
> plot( sin(x), x=0..2*Pi);
```



プロットの変更や操作は，以下の操作パネルから行うことができます．
操作パネルは，プロットされたグラフをクリックするとツールバーの下に表示されます．

ただし，コンマ・コンマ（ `..` ）は範囲，`Pi` は円周率 π （3.141592654.....），及びアスタリスク（ `*` ）は掛け算を表します．さらに，線の色を青に変更する場合（オプション）は，以下のように入力します．

```
> plot( sin(x), x=0..2*Pi, color=blue );
```



以下は，Maple 上での計算例になります．

足し算

```
> 1+2;
```

$$3 \qquad (4.3.3)$$

平方根

```
> sqrt(2);
```

$$\sqrt{2} \qquad (4.3.4)$$

円の面積

```
> Pi*r^2;
```

$$\pi r^2 \qquad (4.3.5)$$

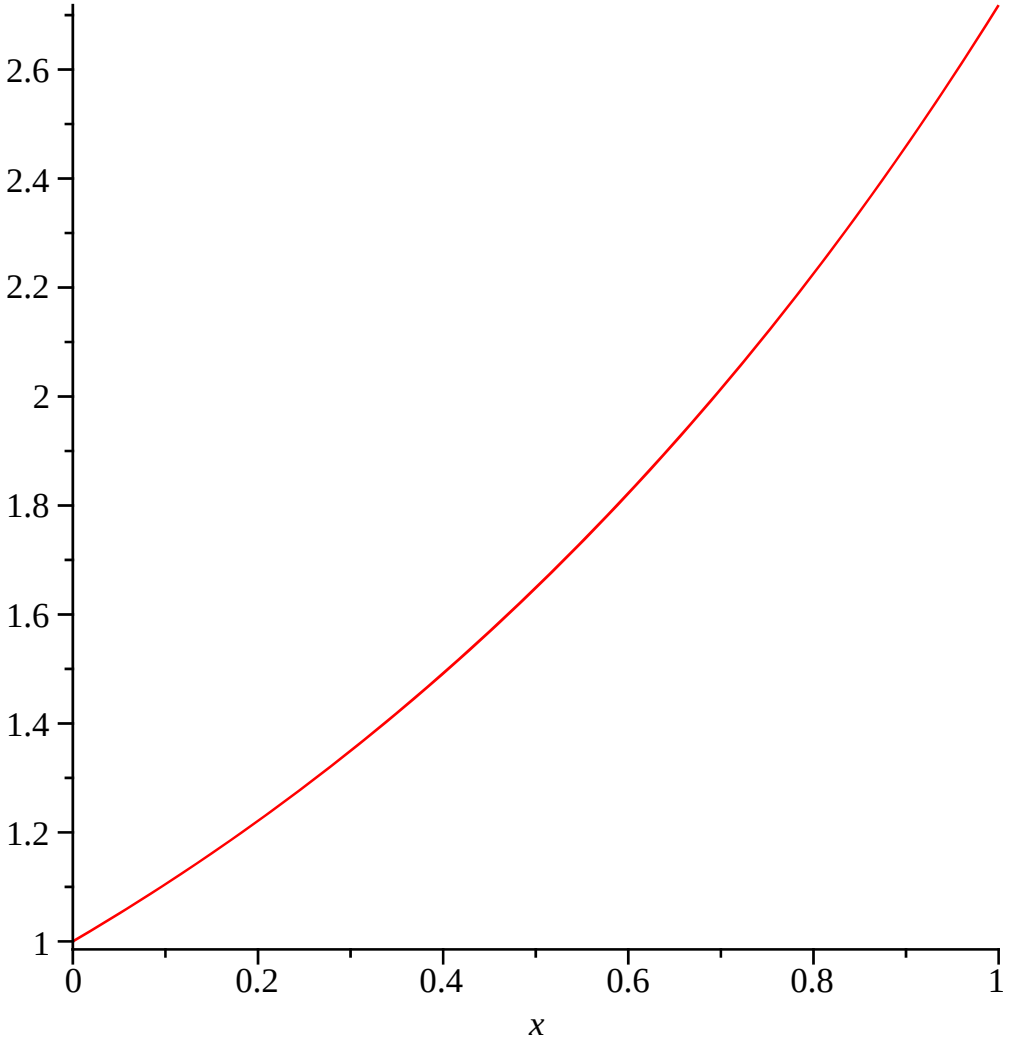
指数関数

```
> exp(x);
```

$$e^{x} \qquad \textbf{(4.3.6)}$$

指数関数のプロット

```
> plot(exp(x), x=0..1);
```



指数関数の極限

```
> limit(exp(x), x=infinity);
```

$$\infty \qquad \textbf{(4.3.7)}$$

テキスト入力（Maple Input, 1D Math）

「テキスト入力」はテキスト形式で（１次元的に，あるいは直線的に）入力を行い，赤文字で表示されます．また，結果は，数式入力の形式（青文字）で表示されます．すべてのコマンドを１行で入力します．以下は $\sin(x)$ の定積分になります．**int** は積分（ integration ）のコマンドです．

```
> int( sin(x), x = a..b );
```

$$\cos(a) - \cos(b) \qquad \textbf{(4.3.1.1)}$$

ラベル（ **(4.3.1.1)** ）が結果とともに出力されます．ラベルの丸括弧 **( )** とコマンド内の丸括弧 **( )** は異なります．

また，ラベルは再計算に用いることができます．

以下は，式 **(4.3.1.1)** に $a=0$，$b=\pi$ を代入し，評価する処理になります．引数やオプションを複数まとめて与える場合，角括弧 [ ] あるいは波括弧 { } を用います．並べる値や式は，コンマで , 区切ります．

eval は式を代入・評価するコマンドです．

```
> eval( (4.3.1.1), [a=0, b=Pi] );
```

$$2 \qquad \textbf{(4.3.1.2)}$$

数式入力（2D Math Input）

「数式入力」は数式の形を（２次元的に，あるいは平面的に）組み立てながら入力を行い，黒文字で表示されます．また，結果は，テキスト入力同様，数式入力の形式（青文字）で表示されます．

（参考）入力には［数式］パレットのテンプレート $\int_a^b f\,dx$ を使用すると便利です．

```
> int(sin(x), x=a..b);
```

$$\cos(a) - \cos(b) \qquad \textbf{(4.3.2.1)}$$

パッケージのロード

- Maple では，大きく ［ 基本コマンド ］ と ［ 応用コマンド ］ に分けられています
- 基本コマンドは，Maple の起動といっしょに読み込まれます（ロードされます）．
- 例えば，プロット（plot）や微積分（diff，int）などのコマンドになります．

一方，応用コマンドは，機能ごとにグループ化され，パッケージとしてその都度ロードされます（ユーザが明示的に読み込みます）．例えば，アニメーション(animate)や連立 1 次方程式(LinearSolve)などのコマンドになります．これは，計算機のメモリ使用を効率化するための仕組みになります．

パッケージをロードするためには，with コマンドを使用します．例えば，アニメーションを作成するための animate コマンドを使用する場合，プロットの拡張機能が集められた plots パッケージを以下のように Maple にロードします．

初期化します（後述）．

```
> restart;
```

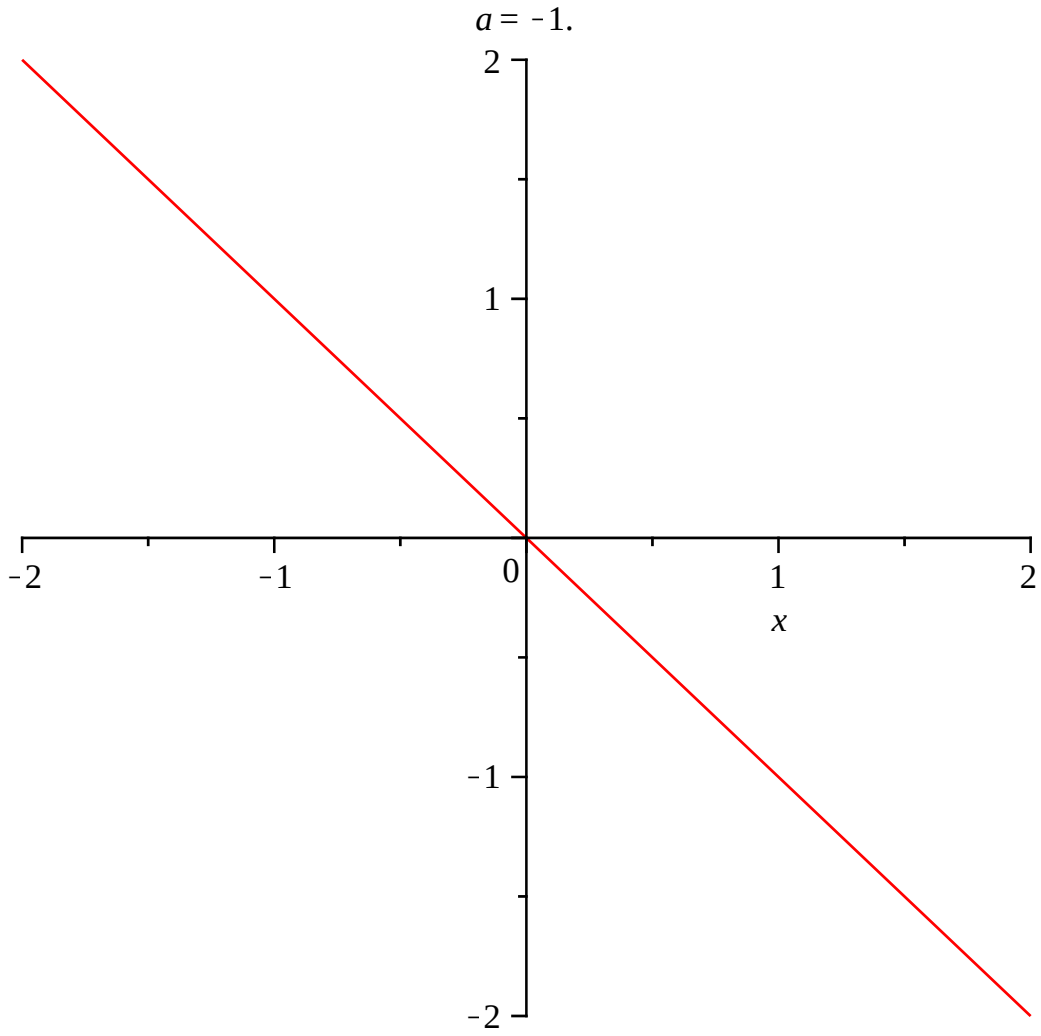
plots パッケージをロードします．

```
> with(plots);
```

[*animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d, densityplot, display, dualaxisplot, fieldplot, fieldplot3d, gradplot, gradplot3d, graphplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams, intersectplot, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot, rootlocus, semilogplot, setcolors, setoptions, setoptions3d, spacecurve, sparsematrixplot, surfdata, textplot, textplot3d, tubeplot*] **(4.4.1)**

ここで，簡単なアニメーションを作成します．式 $a\,x$ の x を -2 から 2 の範囲でプロットし， a を -1 から 1 の範囲で変動させたアニメーションです．

```
> animate( plot, [ a*x, x=-2..2 ], a=-1..1 );
```



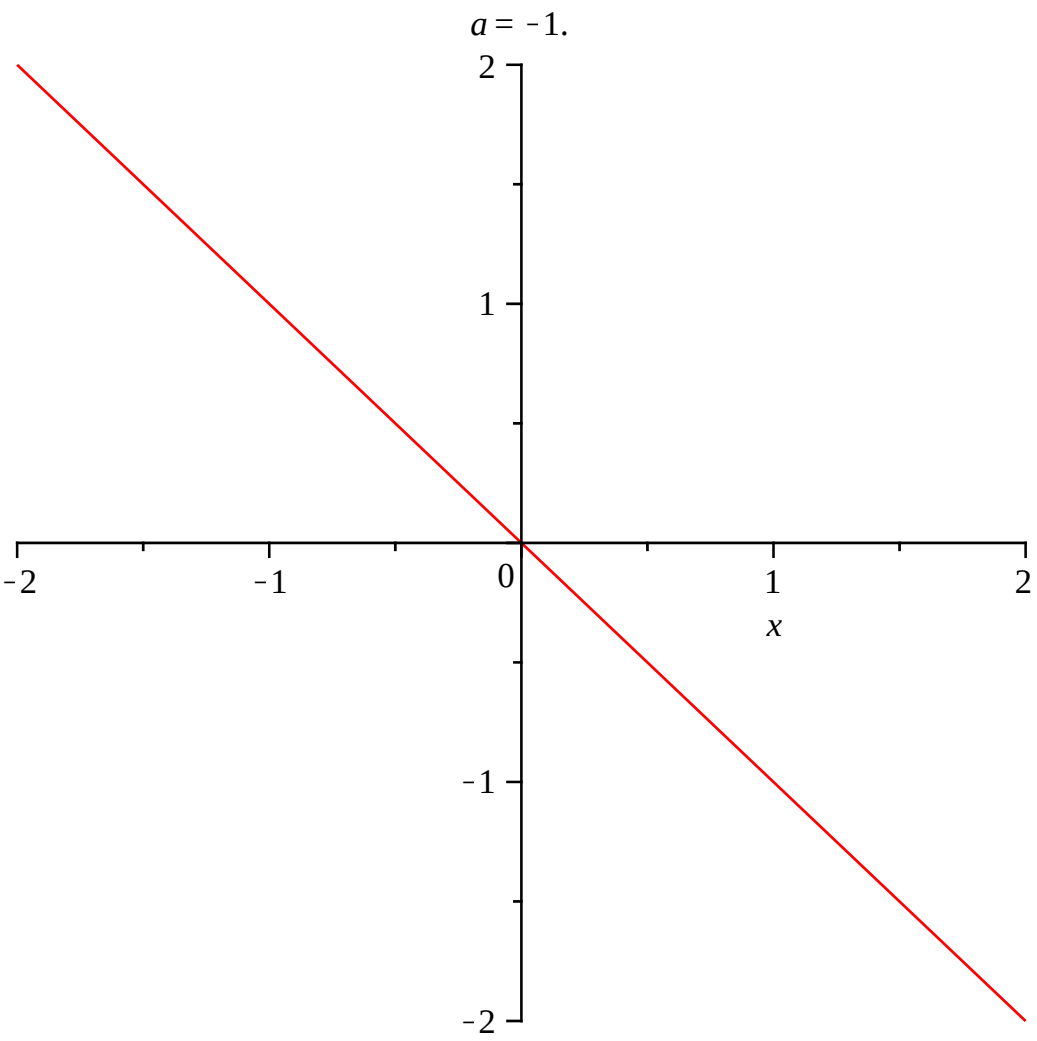
アニメーションの操作は，以下の操作パネルから行うことができます．
操作パネルは，プロットされたグラフをクリックするとツールバーの下に表示されます．

ロードしたパッケージを無効にする場合（ロードをキャンセルする場合），unwith コマンドを使用します．

```
> unwith(plots);
```

再度，animate コマンドを実行します．

```
> animate( plot, [ a*x, x=-2..2 ], a=-1..1 );
```



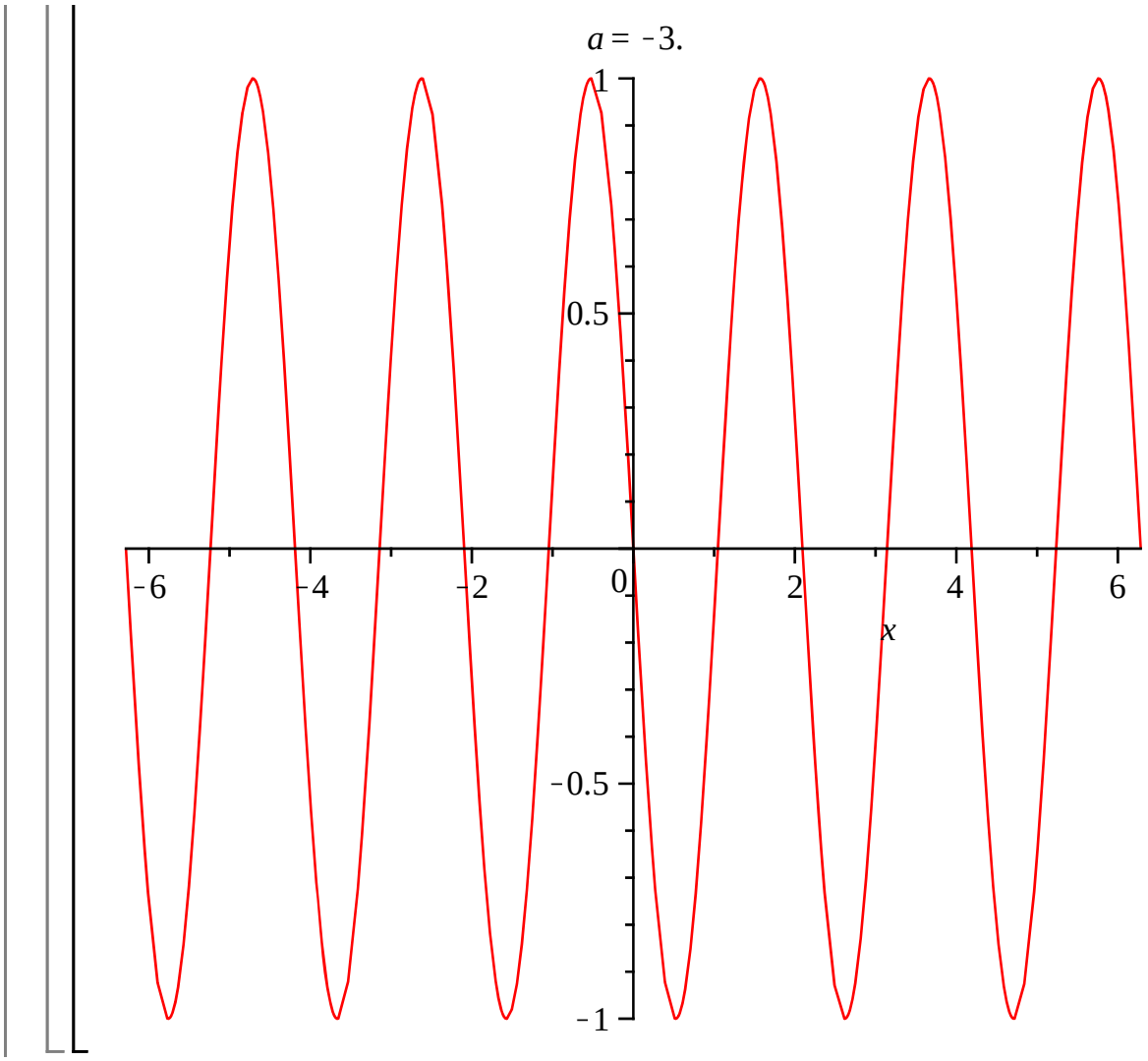
Maple は，animate コマンドを解釈できず，入力をそのまま返します（意味のないオウム返し）．

一方，パッケージ内のコマンドは，そのパッケージをロードしなくても

パッケージ名 [コマンド名] (引数, オプション)

の組合せで使用することができます．以下は，$\sin(a\,x)$ の x を -2π から 2π の範囲でプロットし，を -3 から 3 の範囲で変動させたアニメーションです．

```
> plots[animate]( plot, [ sin(a*x), x=-2*Pi..2*Pi ], a=-3..3 );
```


a = -3.
1
0.5
-0.5
-1
-6
-4
-2
0
2
4
6
x

入力方法の混在

テキスト入力と数式入力を切り替え

一つのワークシート内に 2 つの入力方法を混在させることができます.

テキスト入力と数式入力を切り替えるには，キーボードの F5 キーを押します． また，いつでもテキスト入力と数式入力を切り替えて，編集するることができます.

（例１） 2 次方程式 $a x^2 + b x + c = 0$ の解を求めなさい.

（例２） 式 $\frac{\sin(x)}{a}$ を x について 微分しなさい（ただし，$a \neq 0$）.

入力形式とカーソル

テキスト入力の場合，カーソルはバーティカルバー（ | ）の状態で点滅します.
数式入力の場合，カーソルはスラッシュ（ / ）の状態で点滅します.

```
>
> a*x^2 + b*x + c = 0;
```

F5 キーを押すと，カーソルの形がバーティカルバー（ | ）からスラッシュ（／）に変わります．F5 キーを押すごとに切り替わります.

```
>  /
> a x^2 + b x + c = 0
```

デバッギング

もしコマンドの入力や処理の実行に誤りがある場合，エラーメッセージ（Error）が表示されます.

```
> a*x^2 + b*x + c = ;
Error, `;` unexpected
```

また，コマンドの入力や処理の実行に不正確さがある場合，コマンドは実行されますが，警告メッセージ（Warning）が表示される場合があります.

```
> a*x^2 + b*x + c = 0
Warning, inserted missing semicolon at end of statement
```

$$a x^2 + b x + c = 0 \qquad \textbf{(4.6.1)}$$

メッセージ内容に沿って，コマンドを修正します.

（参考） Warning, inserted missing semicolon at end of statement
セミコロン（ ; ）が足りないことを警告しています．ただし，実行は正常に終了されます.

セミコロン（；）を追加して実行します．

```
> a*x^2 + b*x + c = 0;
```

$$a x^2 + b x + c = 0 \qquad \textbf{(4.6.2)}$$

警告メッセージ（Warning）が出力されなくなります．

（注意）
エラーメッセージ（Error）や警告メッセージ（Warning）の行（表示）をクリックすると，Web ヘルプへ移動します（インターネットブラウザ上に関連するヘルプページが表示されます）．

コメントの追加

シャープ（#）に続く内容（コマンドやテキスト）は実行に含まれません．

（例）

```
> a*x^2 + b*x + c = 0; #  2次方程式（a ≠ 0）
```

$$a x^2 + b x + c = 0 \qquad \textbf{(4.7.1)}$$

コマンド内の改行

改行を入れたい箇所で Shift キーを押しながら，Enter キーを押します．主に，プログラミングで使用します．

（例１）

```
> #  2次方程式（←ここで  Shift  +  Enter  を押します）
  a*x^2 + b*x + c = 0;
```

$$a x^2 + b x + c = 0 \qquad \textbf{(4.8.1)}$$

（例２）

```
> #  以下のような入力も可能です．
    a*x^2 #  2次の項
  + b*x   #  1次の項
  + c     #  0次の項
  = 0;    #  セミコロン（：）かコロン（；）で閉じます
```

$$a x^2 + b x + c = 0 \qquad \textbf{(4.8.2)}$$

コマンド入力の最後を明確にするために，必ずセミコロン（：）かコロン（；）で閉じなければなりません．コマンド入力が閉じられていない場合，処理が正しく実行されない場合があります．

テキスト入力・セミコロン（；）なし

```
> 1/x
```

Warning, inserted missing semicolon at end of statement

$$\frac{1}{x} \quad (4.9.1)$$

```
> 1+2
```

Warning, inserted missing semicolon at end of statement

$$3 \quad (4.9.2)$$

セミコロン（；）を使用せず実行した場合，計算結果は表示されますが，自動的にセミコロンが補完されたことを現す警告メッセージ（Warning）が表示されます．

テキスト入力・セミコロン（；）あり

```
> 1/x;
```

$$\frac{1}{x} \quad (4.10.1)$$

```
> 1+2;
```

$$3 \quad (4.10.2)$$

テキスト入力・コロン（：）あり

```
> 1/x:
```

```
> 1+2:
```

出力は表示されません．

数式入力・セミコロン（；）なし

> $\frac{1}{x}$

$$\frac{1}{x} \quad (4.12.1)$$

> $3+4$

$$7 \quad (4.12.2)$$

数式入力・セミコロン（；）あり

> $\frac{1}{x};$

$$\frac{1}{x} \quad (4.13.1)$$

> $3+4;$

$$7 \quad (4.13.2)$$

「数式入力」では，最後に何も指定しなかった場合と；(セミコロン)を指定した場合に計算結果が表示されます．

数式入力・コロン（：）あり

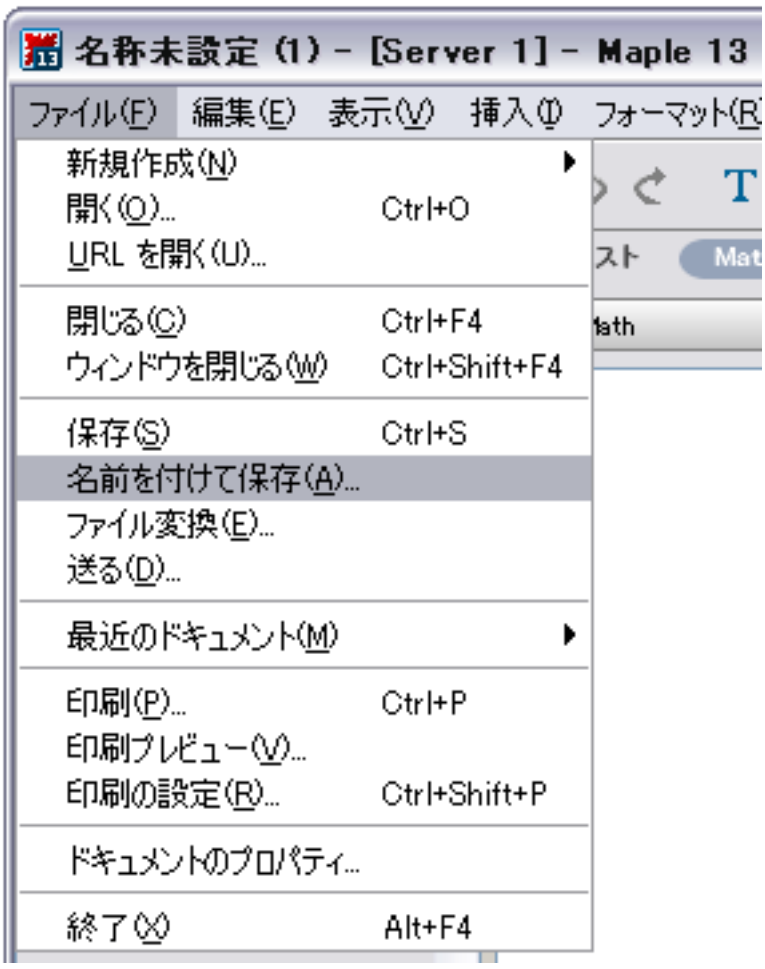
```
> 1/x :
```

```
> 3 + 4 :
```

コロン（：）を入力した場合，テキスト入力と同様に結果は非表示となります．

ファイルの「保存」と「開く」

- ファイルを保存するには，メニューバーの「ファイル」より「名前を付けて保存」を選択します.
- 保存されているワークシートを開くには，ワークシートのアイコンをダブルクリックするか，メニューバーの「ファイル」より「開く」を選択しダイアログから開きたいワークシートを選択します.
- 上書き保存のショートカット（ある特定の機能をキーボードに割り当てたもの）： Ctrl キー + S キー



Maple の四則演算と剰余演算

Maple の四則演算は次のように入力します.

足し算

```
> 12 + 23;
```

$$35 \tag{6.1}$$

```
> a + b;
```

$$a + b \tag{6.2}$$

引き算

```
> 25 - 15;
```

$$10 \tag{6.3}$$

```
> a - b;
```

$$a - b \tag{6.4}$$

掛け算

```
> 11 * 8;
```

$$88 \tag{6.5}$$

```
> a * b;
```

$$a\,b \tag{6.6}$$

割り算

```
> 48 / 12;
```

$$4 \tag{6.7}$$

```
> a / b;
```

$$\frac{a}{b} \tag{6.8}$$

余りの計算（剰余演算）

```
> 21 mod 6;
```

$$3 \tag{6.9}$$

変数の扱い方

変数名の作り方

初期化します（後述）．

```
> restart;
```

最も簡単な形は，1 つの文字です．

```
> a, b, c, x, y, z;
```

複数の文字，数字，アンダースコア (_) を用いて作成することもできます．

```
> abc1, abc2, xy_1, yz_2, zx_3;
```

小文字と大文字の区別を行います．

```
> A, a, X, x, Xy, xY;
```

添え字は，変数名に [ ] を用います．

```
> a[1], a[2], a[3], a[n], a[n+1];
```

$$a, b, c, x, y, z$$
$$abc1, abc2, xy\_1, yz\_2, zx\_3$$
$$A, a, X, x, Xy, xY$$
$$a_1, a_2, a_3, a_n, a_{n+1} \qquad \textbf{(7.1.1)}$$

命名規則の注意事項

（注意）テキスト入力において，変数名の先頭 1 文字目に数字を使用することはできません．

```
> 3x;
```

```
Error, missing operator or `;`
```

ただし，数式入力においては，3*x の演算と等価になります．

```
> 3 x;
  3*x;
```

$$3\,x$$
$$3\,x \qquad \textbf{(7.1.1.1)}$$

（注意）Maple が内部変数として予め持っている変数名（予約語）はユーザ定義の変数名に使用することができません.

```
> D := 1;
```
Error, attempting to assign to `D` which is protected

D は微分演算子として，Maple で使用されます．さらに，以下の数学定数もユーザ定義の変数名に使用することができません.

```
> constants;
```
$$false, \gamma, \infty, true, Catalan, FAIL, \pi \qquad \textbf{(7.1.1.2)}$$

```
> false := 1;
  gamma := 2;
  infinity := 3;
  true := 4;
  Catalan := 5;
  FAIL := 6;
  Pi := 7;
```
Error, attempting to assign to `false` which is protected
Error, attempting to assign to `gamma` which is protected
Error, attempting to assign to `infinity` which is protected
Error, attempting to assign to `true` which is protected
Error, attempting to assign to `Catalan` which is protected
Error, attempting to assign to `FAIL` which is protected
Error, attempting to assign to `Pi` which is protected

また，Digits などの環境変数名もユーザ定義の変数名に使用できない予約語になります.

（参考）環境変数のリスト

%	%%	%%%	Digits	UseHardwareFloats
index/newtable	mod	Normalizer	NumericEventHandlers	Order
printlevel	Rounding	Testzero		

数式入力におけるアンダースコア（ _ ）の入力

数式入力において，アンダースコア（ _ ）を変数名に使用する場合，特別な入力方法があります.

a_1

数式入力の場合，アンダースコア（ _ ）を入力すると，字下げになり，添え字の入力に切り替わります．この字下げを回避するために，アンダースコア（ _ ）を入力する前に，円記号 ¥（もしくは，バックスラッシュ \）を入力します.

$a \backslash\_ 1$

数式入力において，a _ 1 を続けて入力すると以下のようになります．アンダースコア（ _ ）の入力は，Shift キー + バックスラッシュ（ \ ）です（スラッシュ / の隣にあります）．

```
> a_1
```

$$a_1 \qquad \textbf{(7.1.2.1)}$$

数式入力において，a _ 1 を続けて入力すると以下のようになります．

```
> a_1
```

$$a\_1 \qquad \textbf{(7.1.2.2)}$$

つまり，添え字入力がキャンセルされます．

添え字つきの変数をテキスト入力で記述する場合，以下のように入力します．

```
> a[1];
```

$$a_1 \qquad \textbf{(7.1.2.3)}$$

（注意）添え字つきの変数を使用する場合，正しく処理がなされない場合があります．必要に応じて，a1，a2 などの変数名を使用します．

変数への割り当て（代入）

初期化します（後述）．

```
> restart;
```

- 変数（文字や数字の組合せ）に値や数式等を割り当てておく（代入しておく）ことができます．
- 割り当て（代入）をすることで，毎回式を入力する必要なく変数名を使用して計算することができます（再利用することが可能になります）．
- 割り当て（代入）をするには := (コロン+イコール) を使用します．
- 変数名には，大文字と小文字の区別があります．

例えば，a という変数（文字）に 2 という値を割り当てる（格納する）場合，

```
> a := 2;
```

$$a := 2 \qquad \textbf{(7.2.1)}$$

b という変数に $\frac{\sin(x)}{x^2}$ を割り当てる（格納する）場合も同様に := を使用します．

```
> b := sin(x)/x^2;
```

$$b := \frac{\sin(x)}{x^2} \quad \textbf{(7.2.2)}$$

- 変数に割り当てられているもの（格納されているもの）を確認したい場合は，変数名 ; (セミコロン)と入力し Enter キーを押します．
- Maple は変数に割り当てられている（格納されている）中身を出力として返します．
- 変数に何も割り当てられていない（格納されていない）場合は変数名がそのまま返されます．

```
> a;
```

$$2 \quad \textbf{(7.2.3)}$$

```
> b;
```

$$\frac{\sin(x)}{x^2} \quad \textbf{(7.2.4)}$$

```
> c;
```

$$c \quad \textbf{(7.2.5)}$$

- 変数 c には，何も割り当てられていません（格納されていません）ので，入力された文字がそのまま出力されます．また，変数どうしを用いて計算することができます．

```
> a * b;
```

$$\frac{2\sin(x)}{x^2} \quad \textbf{(7.2.6)}$$

```
> a * c;
```

$$2c \quad \textbf{(7.2.7)}$$

変数は大文字と小文字で区別されます．

```
> A := 3;
```

$$A := 3 \quad \textbf{(7.2.8)}$$

```
> A;
```

$$3 \quad \textbf{(7.2.9)}$$

```
> a;
```

$$2 \quad \textbf{(7.2.10)}$$

同じ変数名に，新たに値を割り当てると，上書きされます（格納されている値が入れ替わります）．

```
> b;
```

$$\frac{\sin(x)}{x^2} \quad \textbf{(7.2.11)}$$

```
> b := cos(x)/x;
```

$$b := \frac{\cos(x)}{x} \quad \textbf{(7.2.12)}$$

```
> b;
```

$$\frac{\cos(x)}{x} \quad \textbf{(7.2.13)}$$

同時に複数の変数に別々の値や式を割り当てる（格納する）ことができます．

```
> a, b, c := x+1, sin(x), cos(x);
```

$$a, b, c := x + 1, \sin(x), \cos(x) \quad \textbf{(7.2.14)}$$

(　) を使用するとこともできます．

```
> (p, q, r) := 1, 2, 3;
```

$$p, q, r := 1, 2, 3 \quad \textbf{(7.2.15)}$$

```
> a;
```

$$x + 1 \quad \textbf{(7.2.16)}$$

```
> b;
```

$$\sin(x) \quad \textbf{(7.2.17)}$$

```
> c;
```

$$\cos(x) \quad \textbf{(7.2.18)}$$

```
> p; r; q;
```

$$1$$
$$3$$
$$2 \quad \textbf{(7.2.19)}$$

変数の初期化（リセット）

変数を初期化するには（変数の中身を消去するには），以下の 2 つのコマンドを使用します．

- unassign コマンド
- restart コマンド

unassign コマンドは特定の変数を初期化するコマンドです．変数名を引数としてコマンドに与えます．一方，restart コマンドのように引数を持たないコマンドもあります．

コマンドには様々な形式があります．例えば，a1，a2，a3 という 3 つの変数名にそれぞれ式などが割り当てられている場合（格納されている場合），

12 を *a1* に割り当てます（格納します）．

```
> a1 := 12;
```

$$a1 := 12 \quad \textbf{(7.3.1)}$$

多項式 $x^2 + 2 \cdot x + 1$ を *a2* に割り当てます（格納します）．

```
> a2 := x^2 + 2*x +1;
```

$$a2 := x^2 + 2\,x + 1 \quad \textbf{(7.3.2)}$$

式 $\dfrac{\sin(x)^2}{\sqrt{x}}$ を *a3* に割り当てます（格納します）．

三角関数 sin は，そのまま sin( )を使用します．

平方根 $\sqrt{}$ は， を使用します．

```
> a3 := sin(x)^2/sqrt(x);
```

$$a3 := \frac{\sin(x)^2}{\sqrt{x}} \quad \textbf{(7.3.3)}$$

```
> a1;
```

$$12 \quad \textbf{(7.3.4)}$$

```
> a2;
```

$$x^2 + 2x + 1 \quad \textbf{(7.3.5)}$$

```
> a3;
```

$$\frac{\sin(x)^2}{\sqrt{x}} \quad \textbf{(7.3.6)}$$

ここで a1 を初期化する場合，以下のように unassign を実行します．

```
> unassign('a1');
```

'a1' の ' ' はシングルクォーテーションです．入力は，Shift キー + 7 キーです．

```
> a1;
```

$$a1 \quad \textbf{(7.3.7)}$$

a1 の中身が初期化され， a1 という変数には何も割り当てられていない（格納されていない）状態に戻りました．

```
> a2;
```

$$x^2 + 2x + 1 \quad \textbf{(7.3.8)}$$

```
> a3;
```

$$\frac{\sin(x)^2}{\sqrt{x}} \quad \textbf{(7.3.9)}$$

複数の変数を初期化したい時には，

unassign('*a2*', '*a3*')

と　，(カンマ)　で変数名を区切り入力します．

```
> unassign('a2', 'a3');
> a1;
```

$$a1 \quad \textbf{(7.3.10)}$$

```
> a2;
```

$$a2 \quad \textbf{(7.3.11)}$$

```
> a3;
```

$$a3 \quad \textbf{(7.3.12)}$$

restart コマンドは Maple を起動した時と同じ状態に戻す働きがあります．

全ての変数を一度に初期化できるコマンドです.

ここで， a1， a2，および a3 を再定義します.

```
> a1 := 12;
```

$$a1 := 12 \tag{7.3.13}$$

```
> a2 := x^2 + 2*x +1;
```

$$a2 := x^2 + 2\,x + 1 \tag{7.3.14}$$

```
> a3 := sin(x)^2/sqrt(x);
```

$$a3 := \frac{\sin(x)^2}{\sqrt{x}} \tag{7.3.15}$$

```
> a1;
```

$$12 \tag{7.3.16}$$

```
> a2;
```

$$x^2 + 2\,x + 1 \tag{7.3.17}$$

```
> a3;
```

$$\frac{\sin(x)^2}{\sqrt{x}} \tag{7.3.18}$$

a1，a2，および a3 にそれぞれ値や式が割り当てられています（格納されています）.

restart を実行します.

```
> restart;
```

a1， a2，および a3 を確認します.

```
> a1;
```

$$a1 \tag{7.3.19}$$

```
> a2;
```

$$a2 \tag{7.3.20}$$

```
> a3;
```

$$a3 \tag{7.3.21}$$

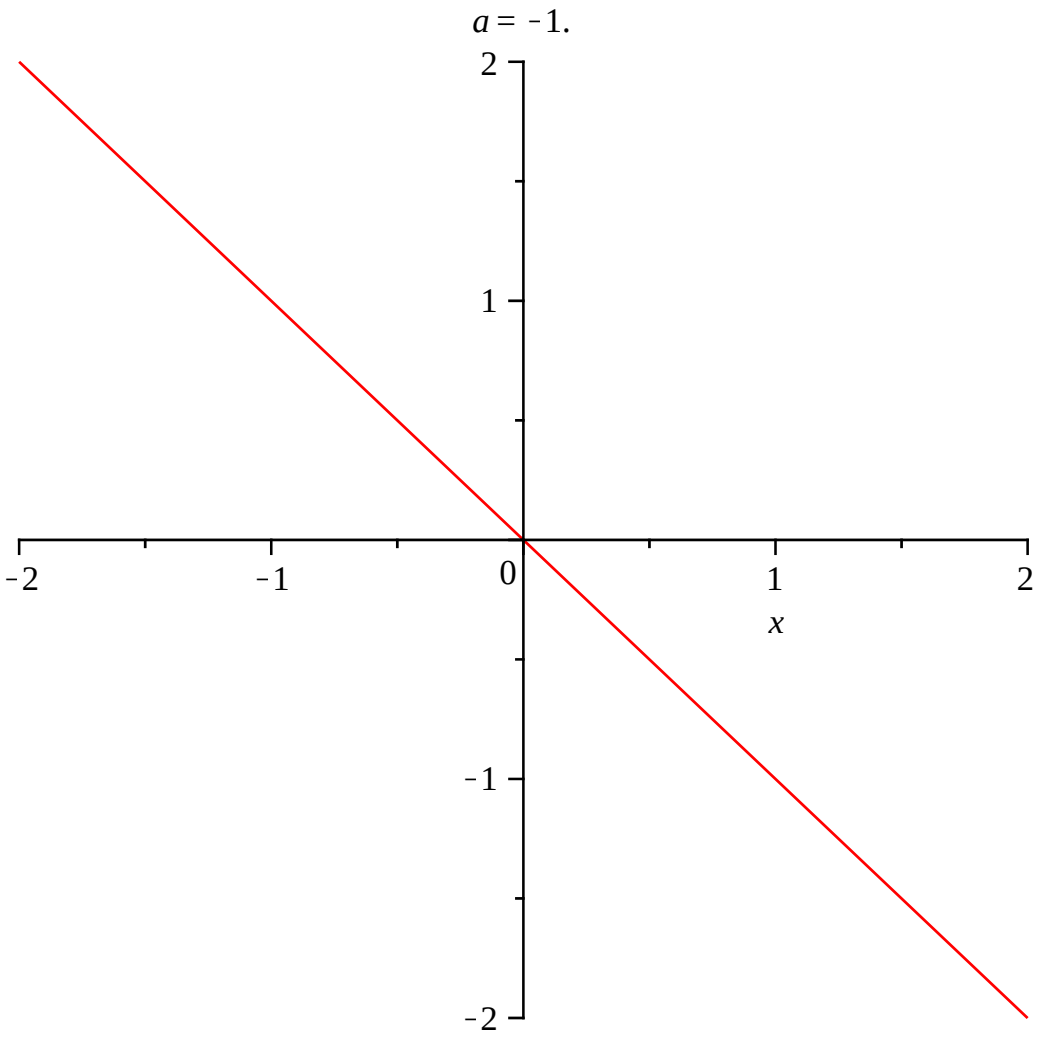
すべての変数が初期化されました.

また，restart コマンドは，ロードされたパッケージ（応用コマンド）もリセット（キャンセル）することができます．再度，アニメーションを作成するため，plots パッケージをロードします.

```
> with(plots):
```

式 $a\,x$ の x を -2 から 2 の範囲でプロットし， a を -1 から 1 の範囲で変動させた場合のアニメーションです.

```
> animate( plot, [a*x, x=-2..2], a=-1..1 );
```



restart コマンドを実行します.

```
> restart:
```

再び，animate コマンドを使用します.

```
> animate( plot, [a*x, x=-2..2], a=-1..1 );
```

$$animate(plot, [a\,x, x = -2..2], a = -1..1) \qquad \textbf{(7.3.22)}$$

restart（初期化）コマンドの実行により，animate コマンドが正しく動作しません.

結果の再利用

初期化します.

```
> restart;
```

ラベルの利用　（出力結果の利用１）

```
> sin(x);
```

$$\sin(x) \tag{7.4.1}$$

ラベルの入力

Ctrl キー ＋ L キーにより次図のラベル入力ウィンドウが立ち上がります．適当なラベル（例えば，7.4.1）を入力し，OK を押します．


```
> (7.4.1);
```

$$\sin(x) \tag{7.4.2}$$

```
> int( (7.4.1), x );
```

$$-\cos(x) \tag{7.4.3}$$

ラベルの丸括弧 ( ) とコマンド内の丸括弧 (　) は異なります．なお，以下のような入力は，正しく処理されません．

```
> int( (8.4.1), x);
```

$$8.400000000\,x \tag{7.4.4}$$

別の計算処理を途中に追加したことにより，ラベルの値が変わったとしても，ラベルの値は自動的に更新されます．

ditto 演算子の利用　（出力結果の利用２）

%（ひとつ前の結果），%%（ふたつ前の結果），%%%（３つ前の結果） など

```
> a, b, c := x+1, sin(x), cos(x);
```

$$a, b, c := x+1, \sin(x), \cos(x) \tag{7.4.5}$$

```
> a;
```

$$x+1 \tag{7.4.6}$$

```
> b;
```

$$\sin(x) \tag{7.4.7}$$

```
> c;
```

$$\cos(x) \tag{7.4.8}$$

直前（ひとつ前）の結果 **(7.4.8)** を表示

```
> %;
```

$$\cos(x) \qquad \textbf{(7.4.9)}$$

3 つ前の結果 **(7.4.7)** $\sin(x)$ を利用

```
> int(%%%, x);
```

$$-\cos(x) \qquad \textbf{(7.4.10)}$$

厳密解と数値解

Maple では，有理数や整数で値を入力した場合は，厳密な値を求める計算をします．一方，値に小数点が含まれる場合は，近似値（数値）を求める計算をします．一般的に，近似値（数値）を求める計算を数値計算と呼びます．

```
> 1/sqrt(2);
```

$$\frac{\sqrt{2}}{2} \tag{8.1}$$

```
> 1/sqrt(2.);
```

$$0.7071067814 \tag{8.2}$$

精度は，Digits で操作できます．

（参考）この Digits は Maple の予約語（あらかじめ Maple が持っている変数名）になります．そのため，ユーザが予約語を変数名として使用することはできません．その他，Pi （π = 3.141592654・・・）のような数学定数の名前も変数名として使用することはできません．

Maple の起動時に Digits に割り当てられている値は 10 です（通常，あらかじめ割り当てられている値をデフォルト値と呼びます）．以下は，その値を 50 に変更した場合の結果になります．Digits の値を大きくすると，その分計算時間が掛かります．

```
> Digits := 50;
```

$$Digits := 50 \tag{8.3}$$

```
> 1/sqrt(2.);
```

$$0.70710678118654752440084436210484903928483593768850 \tag{8.4}$$

厳密値は，誤差のない値を求めることを可能にします．

```
> sqrt(2)*sqrt(3);
```

$$\sqrt{2}\,\sqrt{3} \tag{8.5}$$

厳密値は誤差のない値を表しますが，その結果を他のツールでそのまま使用することはできません．そのため，数値化（近似）しなければならない場合があります．そのような場合，evalf コマンドを使用します．

```
> ans := sqrt(2)*sqrt(3);
```

$$ans := \sqrt{2}\,\sqrt{3} \tag{8.6}$$

```
> evalf(ans);
```

$$2.4494897427831780981972840747058913919659474806566 \tag{8.7}$$

ここで，Digits をデフォルト値 10 に再設定します．

```
> Digits := 10;
```

$$Digits := 10 \tag{8.8}$$

```
> evalf(ans);
```

$$2.449489743 \quad \textbf{(8.9)}$$

Maple は，数値計算において，ハードウェアに依存しない実数の近似値計算を行っています．従って，Digits で設定されている精度の範囲内で正確な値を計算します．以下のような単純な計算でも，他の数値計算ソフトウェアでは，計算結果がことなります．

```
> 0.3 - 0.2 - 0.1;
```

$$0. \quad \textbf{(8.10)}$$

例えば，典型的な倍精度演算において，以下のような計算結果が返されます．

```
> 0.3 - 0.2 - 0.1
```

```
-2.7756e-017
```

ソフトウェアによっては多少対処がされているものもありますが，多くの数値計算ソフトウェアは倍精度演算（浮動小数点演算＝実数近似値の表現方式のひとつ）を使用する為，0以外の結果を返します．

倍精度演算は値を浮動小数点演算のひとつであり，64bit 2 進数で計算します．0.3 等のほとんどの小数点数は 2 進数にすると無限に続き，長さが 64 bit 以下については切り捨てられます．そのため，倍精度演算は誤差が発生しやすくなります．

Maple でも evalhf コマンドを使用することにより倍精度演算ができます．

```
> a := evalhf(0.3);
```

$$a := 0.29999999999999988 \quad \textbf{(8.11)}$$

```
> b := evalhf(0.2);
```

$$b := 0.20000000000000010 \quad \textbf{(8.12)}$$

```
> c := evalhf(0.1);
```

$$c := 0.10000000000000004 \quad \textbf{(8.13)}$$

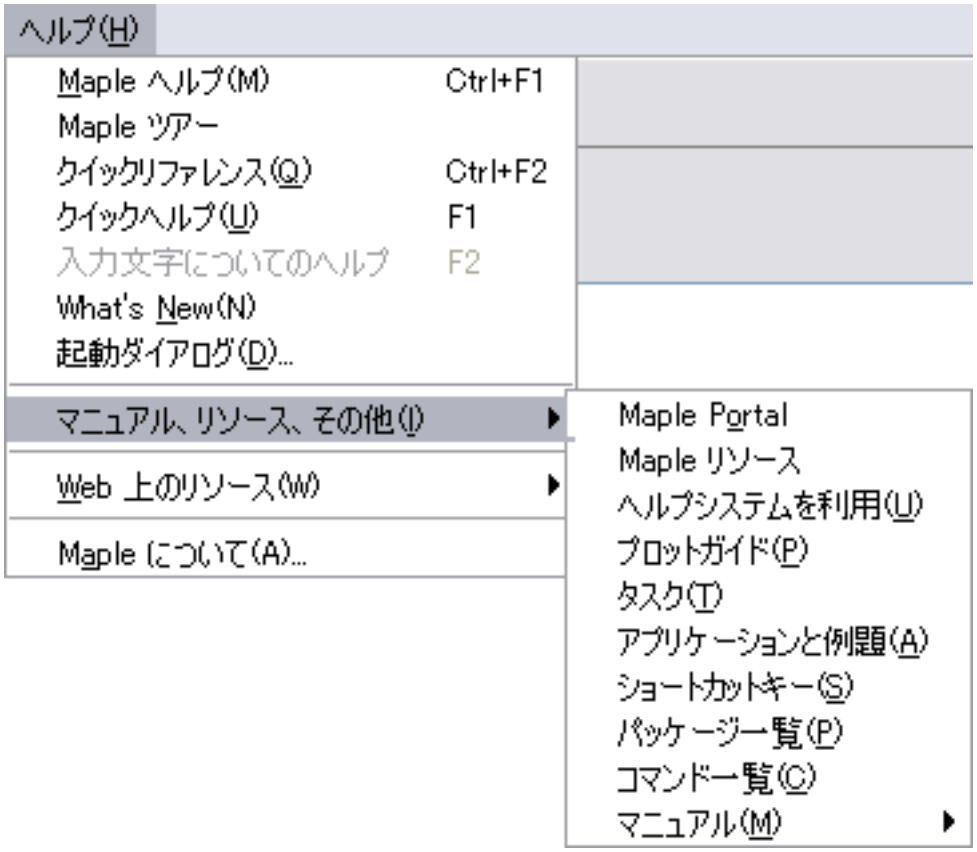
ここで，0.3, 0.2, 0.1 を確認すると，返される値には誤差が含まれていることが確認できます．倍精度演算では，このような誤差が演算ごとに累積されていき，結果の精度が落ちる原因になります．倍精度演算での計算を Maple で再現します．

```
> evalhf( a - b - c );
```

$$-2.77555756156289136\ 10^{-17} \quad \textbf{(8.14)}$$

オンラインヘルプの活用

Maple 13 のヘルプメニュー



- Maple ヘルプ
- Maple ツアー
- Maple Portal （英語版のみ）
- プロットガイド

コマンドからヘルプを起動する方法（例題の実行）

Maple ヘルプ起動後，ツールバー  から  （ ワークシートに現在のヘルプを開く ）をクリックすると，ヘルプ画面が Maple のワークスペースに読み込まれます．そこで，例題などをそのまま実行することができます．例えば，積分コマンドのヘルプを呼び出します．

```
> ?int
```

また，int コマンド（の上）にカーソルを移動させて，F2 キーを押すと，同じように Maple ヘルプ

が起動します.

（参考）コマンドは，すべて英名か，その短縮形などが使用されています.

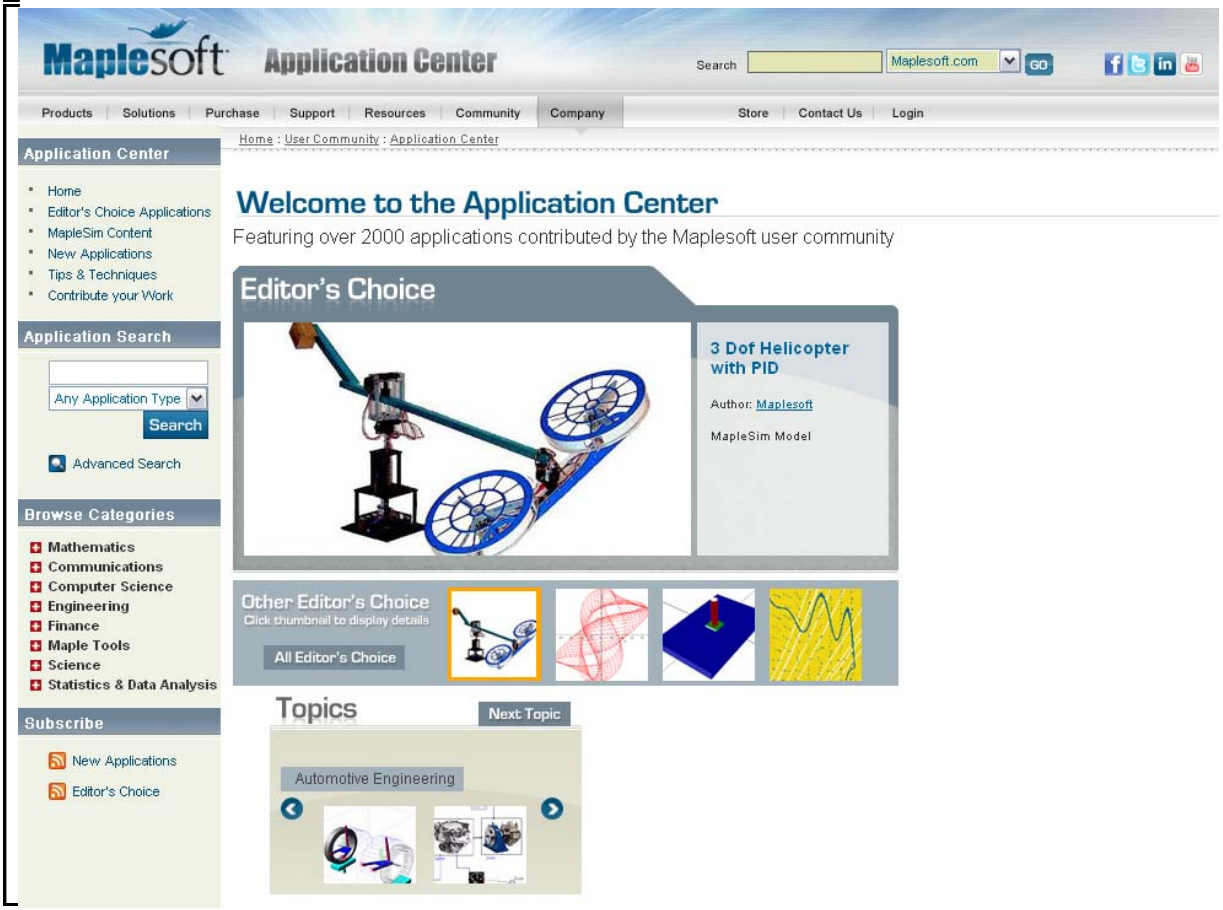
そのほか，Maple ツアー，Maple Portal，あるいはプロットガイドなどは，<やりたいこと> から Maple の使い方を学習できるツールになっています.

応用事例

Maplesoft Application Center

Maple を利用した（学生向け，教師向け，研究者向け，およびエンジニア向けの）様々な事例が公開されています（随時更新）．

http://www.maplesoft.com/applications/index.aspx（英語）



Maple デモ&ギャラリー

日本語による事例サイトになります．基礎から応用まで広く紹介しています．

http://www.cybernet.co.jp/maple/product/maple/demo/index.shtml

ステップ･バイ･ステップ式 **はじめての Maple** 基礎編

運動を見る！

Maple の数式処理を概観します.

目次

物体をただ落としてみる（自由落下運動）

自由落下運動をシミュレーションします．

初期化します．

```
> restart;
```

等加速度直線運動

はじめに等加速度直線運動（同じ加速度で直線上を動く運動）を考えます．等加速度直線運動の速度と位置の式は，初速度を v0 $\left[\left[\frac{m}{s}\right]\right]$ とし，加速度を a $\left[\left[\frac{m}{s^2}\right]\right]$，また時間 t のときの位置を s $[[m]]$，速度を v $\left[\left[\frac{m}{s}\right]\right]$ とすると以下のように表されます．

$$v = a\,t + v0$$

$$s = \frac{1}{2}a\,t^2 + v0\,t$$

まずは，上の２式をそのまま入力してみます．

```
> v = a*t + v0;
```

$$v = a\,t + v0 \qquad \textbf{(1.1.1)}$$

```
> s = (1/2)*a*t^2 + v0*t;
```

$$s = \frac{1}{2}\,a\,t^2 + v0\,t \qquad \textbf{(1.1.2)}$$

ここで，位置 s を時間 t で微分します．

```
> diff(s, t);
```

$$0 \qquad \textbf{(1.1.3)}$$

期待される結果は，速度の式

$$a\,t + v0$$

ですが，ここでは結果が 0 になります．これは，式を割り当てていない（格納していない）ためです（ 割り当てには := を使用します ）．確認のために，s を表示します．

```
> s;
```

$$s \qquad \textbf{(1.1.4)}$$

$\frac{1}{2} a t^2 + v0\, t$ が表示されません．割り当て（変数への格納） **:=** を用いて式を再定義します．

```
> s := (1/2)*a*t^2 + v0*t;
```

$$s := \frac{1}{2} a t^2 + v0\, t \qquad \textbf{(1.1.5)}$$

```
> s;
```

$$\frac{1}{2} a t^2 + v0\, t \qquad \textbf{(1.1.6)}$$

今度は，ただしく割り当てられました．再度，位置 s を時間 t で微分します．

```
> diff(s, t);
```

$$a t + v0 \qquad \textbf{(1.1.7)}$$

同様に，速度の式も定義しなおします．

```
> v := a*t + v0;
```

$$v := a t + v0 \qquad \textbf{(1.1.8)}$$

```
v := diff(s, t);
```

を実行しても，同様の定義が可能です．

自由落下運動

これから調べる運動を自由落下運動とします．また，このとき鉛直方向下向きをプラス（正の方向）とします．自由落下運動は，初速度 v0=0，加速度 a=+g（下向きを正にした） から，位置と速度の式を以下のように変形します．**eval** は式を代入・評価するコマンドです．式 s に，a=g, v0=0 を代入します．

```
> eval(s, [a=g, v0=0]);
```

$$\frac{g t^2}{2} \qquad \textbf{(1.2.1)}$$

ここで，代入する式 [a=g, v0=0] を別の変数 param に割り当てます．

```
> param := [a=g, v0=0];
```

$$param := [a = g, v0 = 0] \qquad \textbf{(1.2.2)}$$

今度は，変数 param を用いて位置の式に代入（評価）してみます． 入力がすっきりします．

```
> eval(s, param);
```

$$\frac{g t^2}{2} \qquad \textbf{(1.2.3)}$$

同様に速度の式にも param を代入します．

```
> eval(v, param);
```

$$g\,t \tag{1.2.4}$$

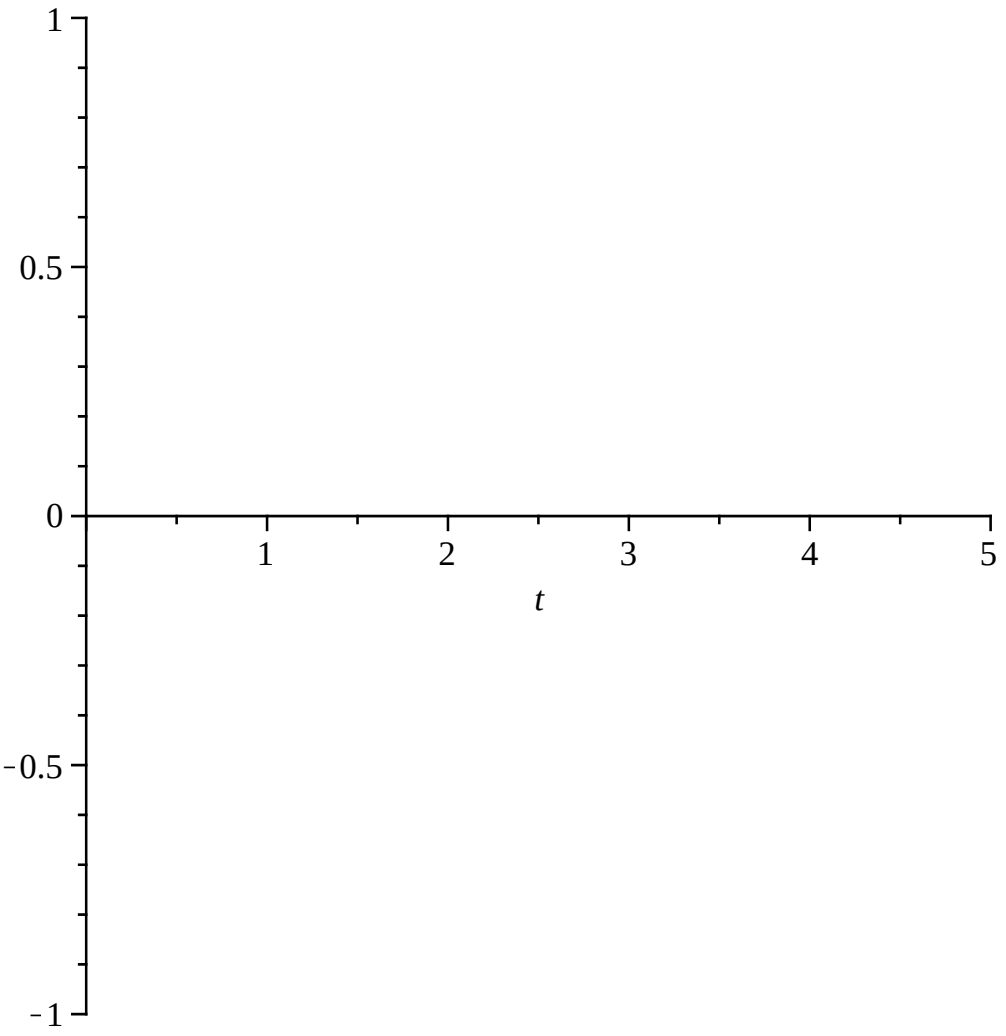
結果を，次の処理で使用するために eval(s, param) の結果を別の変数（大文字） S に割り当てます．

```
> S := eval(s, param);
```

$$S := \frac{g\,t^2}{2} \tag{1.2.5}$$

結果（自由落下運動の位置） S を時間 0 から 5 [[s]] の範囲でプロットします（ t = 0..5 ）．

```
> plot(S, t=0..5);
```



なぜか，プロットされません．S の中身をじっと見てみます．

```
> S;
```

$$\frac{g\,t^2}{2} \tag{1.2.6}$$

何かが足りません．実は，重力加速度 g の値が未定義のままです．つまり，プロットするには g が

数値になっていなければなりません（t はそのままです）.

$g = 9.807 \left[\!\left[\frac{m}{s^2} \right]\!\right]$

そこで，代入する式（値）を再定義し，もとの位置の式に代入･評価します．上と区別するために param1 を使用します.

```
> param1 := [a=9.807, v0=0];
```

$$param1 := [a = 9.807, v0 = 0] \quad \textbf{(1.2.7)}$$

s の内容を確認します.

```
> s;
```

$$\frac{1}{2} a t^2 + v0\, t \quad \textbf{(1.2.8)}$$

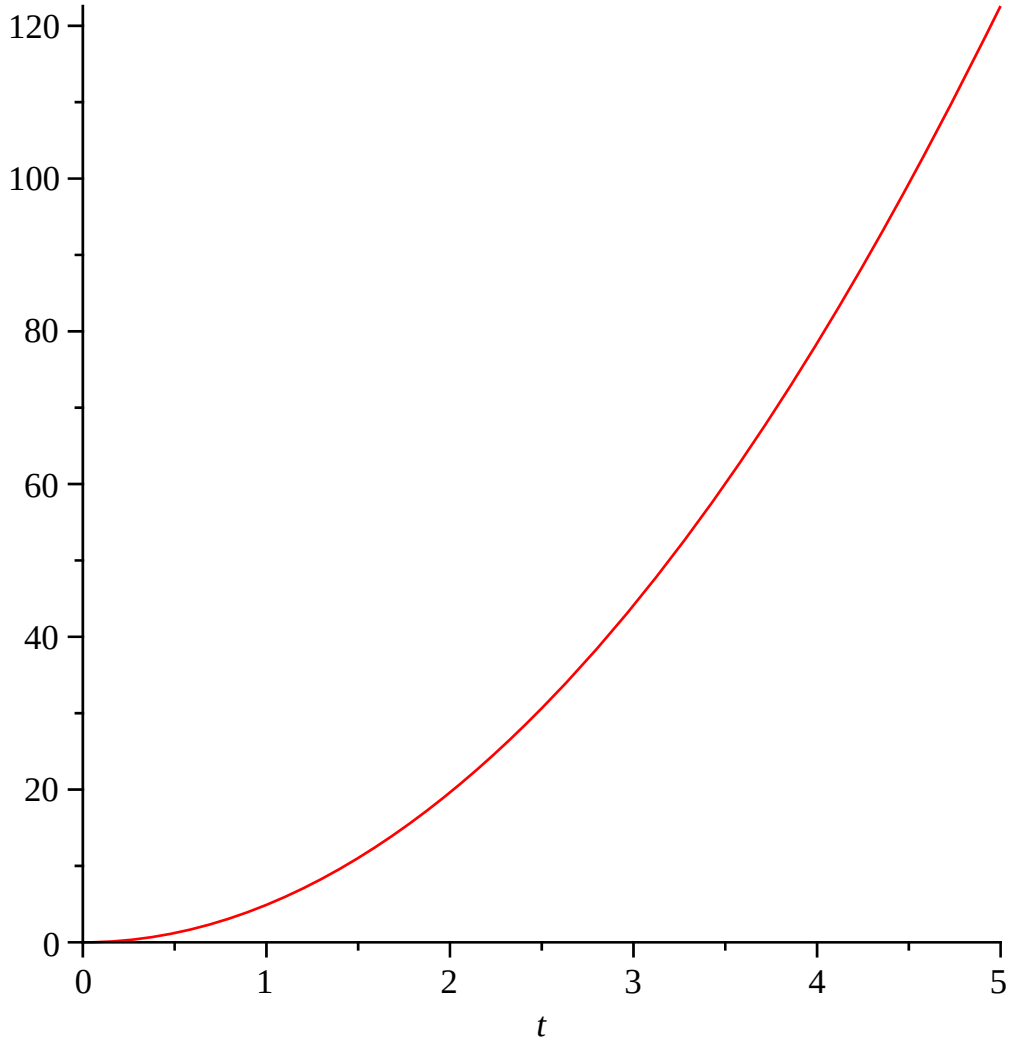
S と区別するために，S1 を使用します.

```
> S1 := eval(s, param1);
```

$$S1 := 4.903500000\, t^2 \quad \textbf{(1.2.9)}$$

再度，時間 0 から 5 ⟦s⟧ の範囲でプロットを試します.

```
> plot(S1, t=0..5);
```



今度は，プロットされました．次に，速度の式に param1 を代入・評価し，同様にプロットします（時間は 0 から 5 〚s〛 間）．確認のため内容を表示します．

```
> v; param1;
```

$$a\,t + v0$$

$$[a = 9.807,\ v0 = 0] \qquad \textbf{(1.2.10)}$$

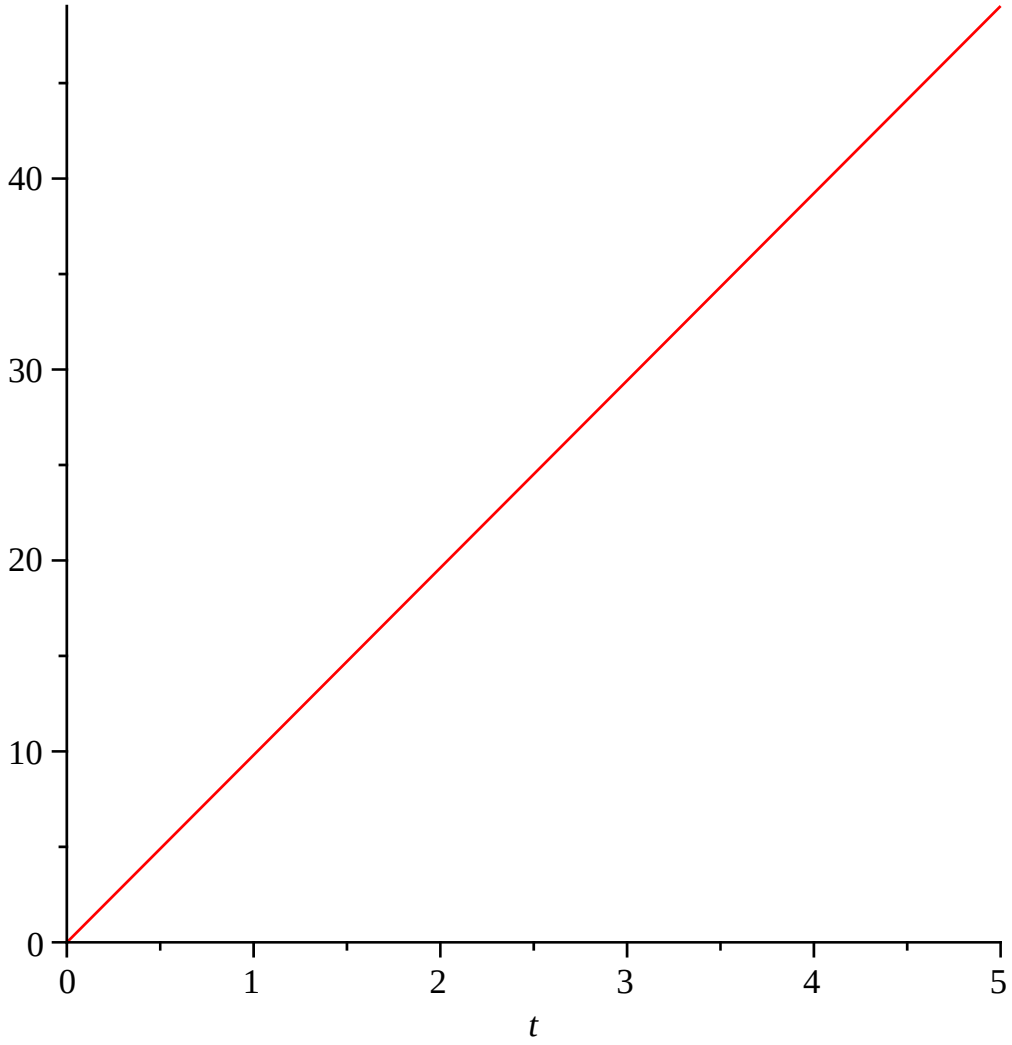
評価後の式を V1 とします．

```
> V1 := eval(v, param1);
```

$$V1 := 9.807\,t \qquad \textbf{(1.2.11)}$$

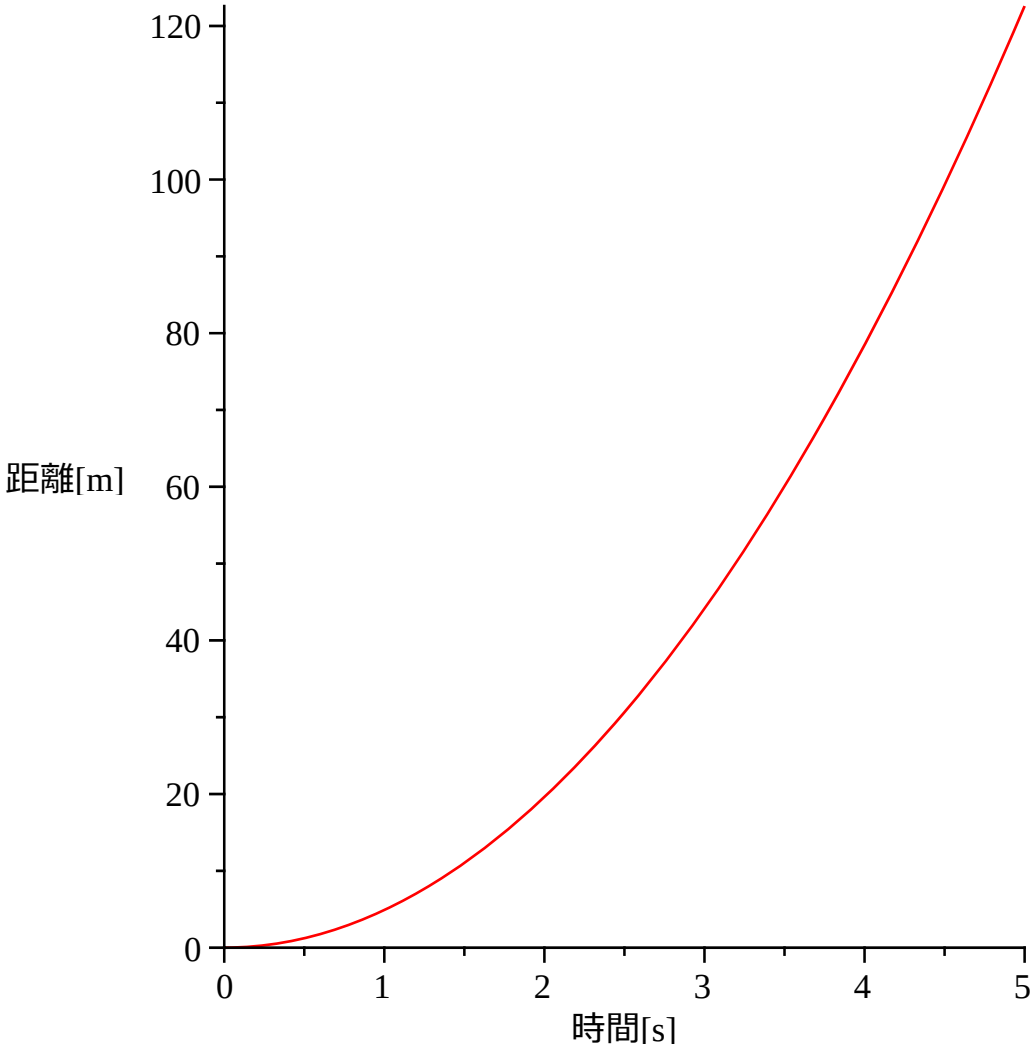
速度の式 V1 をプロットします（時間は 0 から 5 〚s〛 間）．

```
> plot(V1, t=0..5);
```



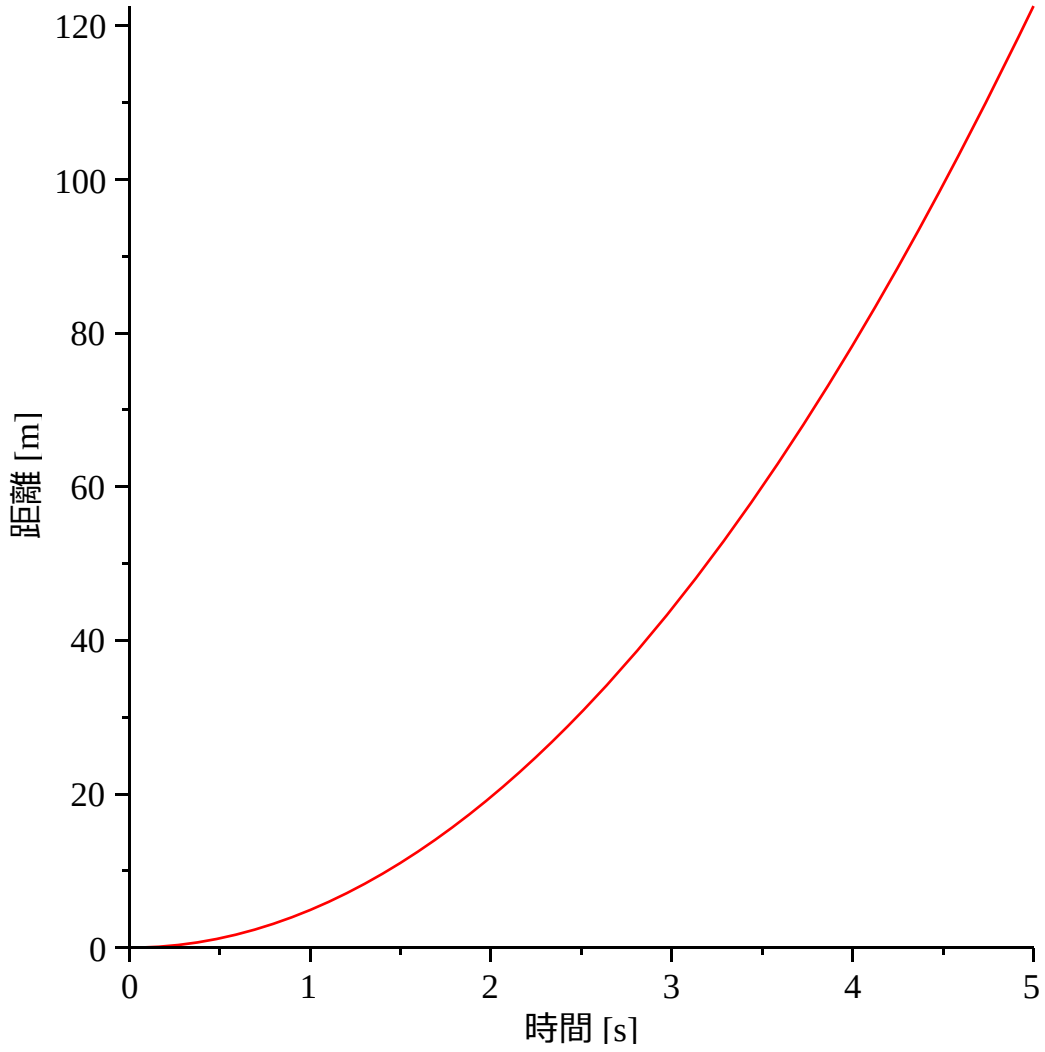
位置（落下距離）をプロットします．横軸と縦軸それぞれにラベルを追加します（ plot コマンドのオプション labels を使用します）．

```
> plot(S1, t=0..5, labels=["時間[s]", "距離[m]"]);
```



縦軸のラベルの向きを調整します.

```
> plot(S1, t=0..5, labels=["時間 [s]", "距離 [m]"], labeldirections=
  [default, vertical]);
```



プロットにタイトルをつけます.

はじめに，ラベル，タイトルを変数に割り当てます.

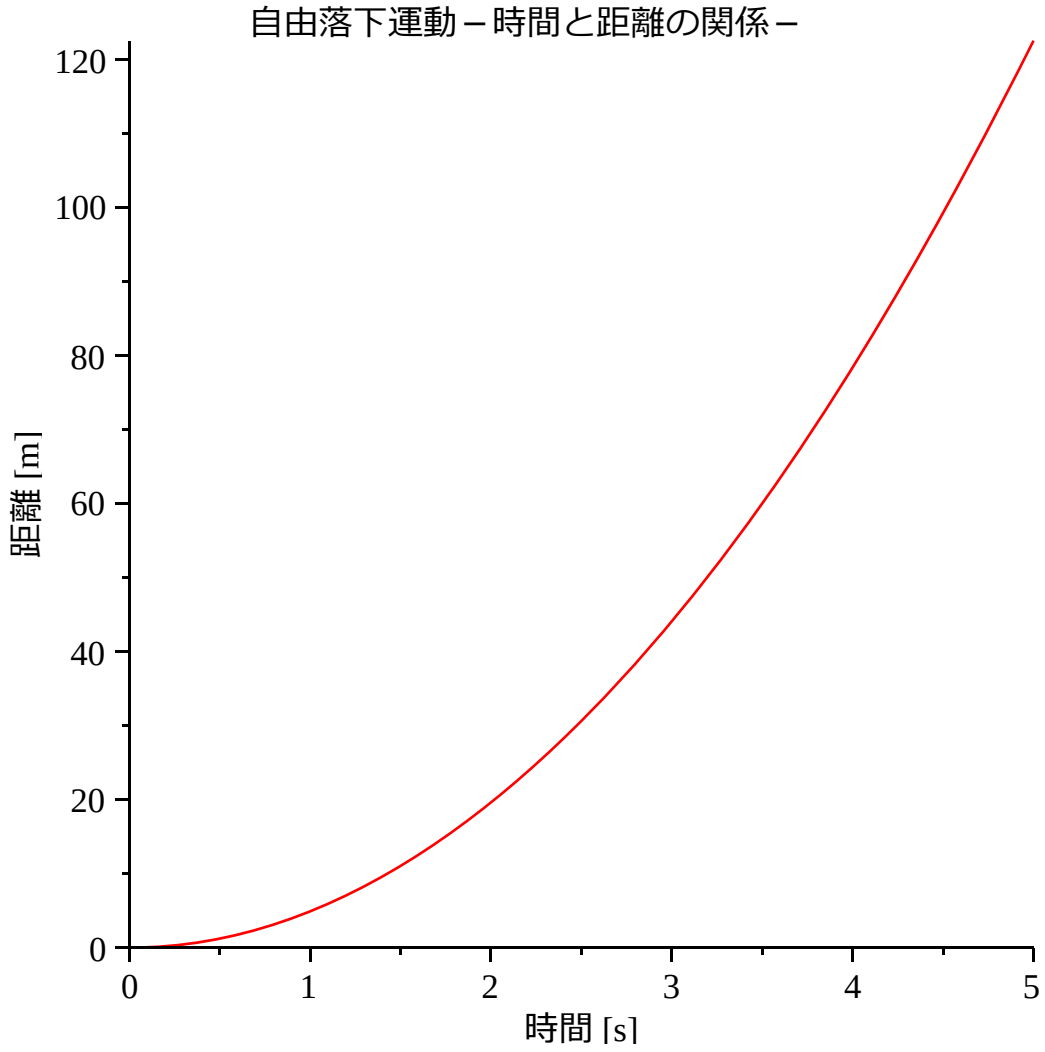
```
> myLabel := ["時間 [s]", "距離 [m]"];
```

$$myLabel := [\text{"時間 [s]"}, \text{"距離 [m]"}] \quad \textbf{(1.2.12)}$$

```
> myTitle := "自由落下運動－時間と距離の関係－";
```

$$myTitle := \text{"自由落下運動－時間と距離の関係－"} \quad \textbf{(1.2.13)}$$

```
> plot(S1, t=0..5, labels=myLabel, labeldirections=[default, vertical],
  title = myTitle);
```



（参考）入力の途中に改行を入れて整理すると見やすくなります．

改行は，Shift + Enter になります．以下は，入力例です．

```
> A := 1;
  B := 2;
  C := A + B;
```

$$A := 1$$
$$B := 2$$
$$C := 3 \qquad \textbf{(1.2.14)}$$

オプションの入力をわかりやすくした入力例になります．

```
> plot(S1, t=0..5, labels=myLabel,
                   labeldirections=[default, vertical],
                   title = myTitle):
```

今度は，上向きをプラスとします．そのため，重力加速度 a=-g（ -9.807 ）となります．

代入する式を param2 として定義し，位置の式 **(1.1.5)** に代入します．

（復習）ラベルは，Ctrl キー ＋ L キーで使用します．

```
> param2 := [a=-9.807, v0=0];
```

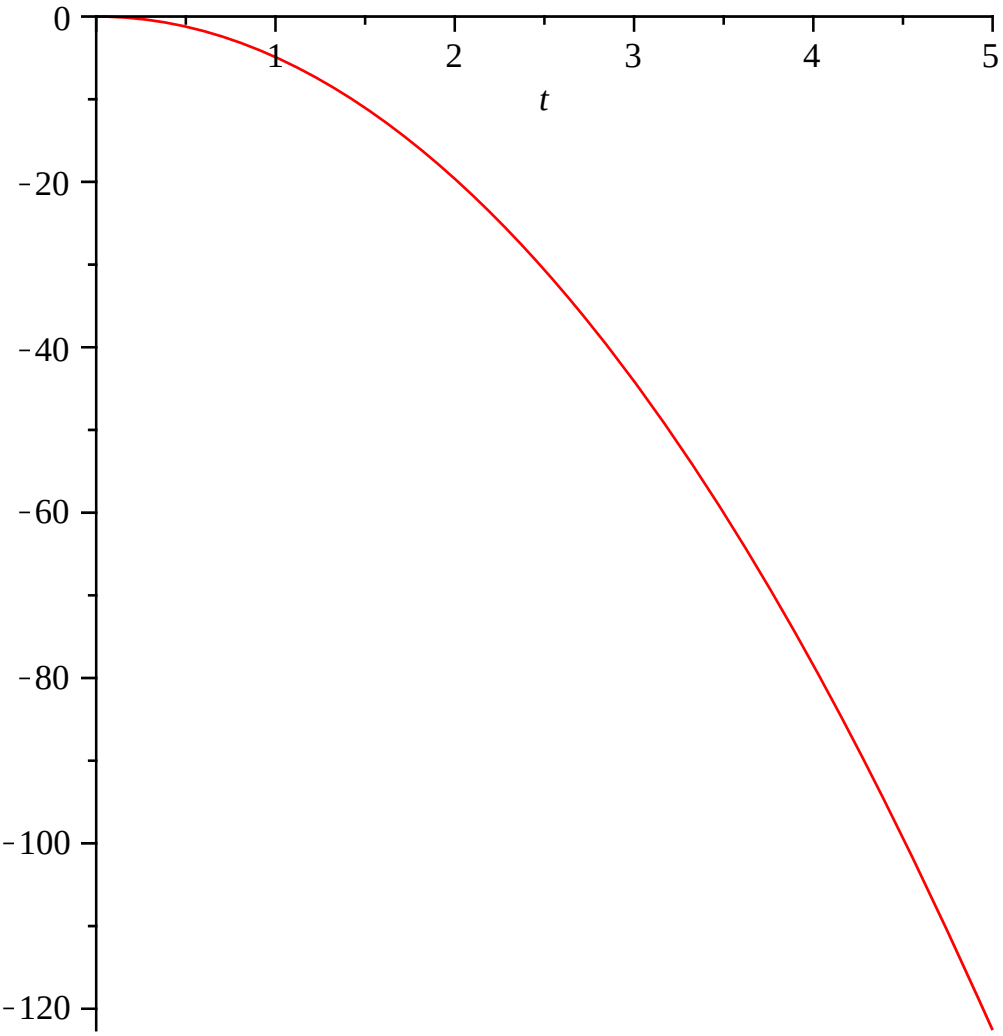
$$param2 := [a = -9.807, v0 = 0] \quad \textbf{(1.2.15)}$$

```
> S2 := eval(s, param2);
```

$$S2 := -4.903500000\, t^2 \quad \textbf{(1.2.16)}$$

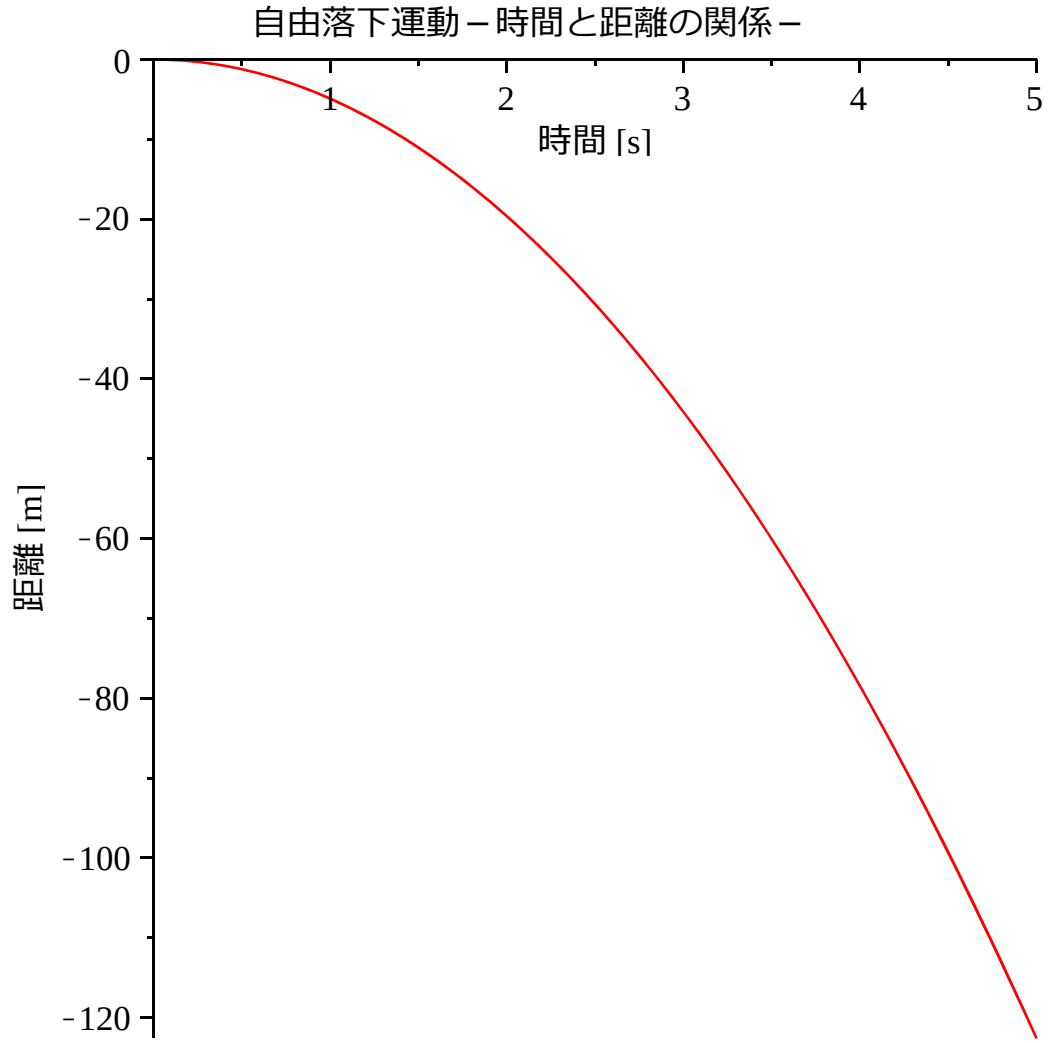
式 **(1.2.16)** をプロットします（時間は 0 から 5 ⟦s⟧ 間）．

```
> plot(S2, t=0..5);
```



ラベルとタイトルを追加します（時間は 0 から 5 ⟦s⟧ 間）．

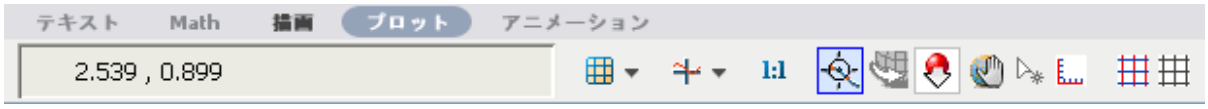
```
> plot(S2, t=0..5, labels=["時間 [s]", "距離 [m]"],
                   labeldirections=[default, vertical],
                   title = myTitle);
```



（復習）

プロットの変更や操作は，以下の操作パネルから行うことができます．

操作パネルは，プロットされたグラフをクリックするとツールバーの下に表示されます．



こんどは物体を上に投げてみる（投げ上げ運動）

投げ上げ運動をシミュレーションします．

変数を初期化します．

```
> restart;
```

（参考）S1 や V1 などの内容が消去されます．

```
> S1; V1;
```

$$S1$$
$$V1 \qquad \textbf{(2.1)}$$

（参考）ただし，出力結果は，ラベルを使用することで，処理の中で再利用することが出来ます．

```
> (1.2.9); (1.2.11);
```

$$4.903500000\,t^2$$
$$9.807\,t \qquad \textbf{(2.2)}$$

それでは，等加速度直線運動の速度と加速度を再定義します．

```
> v := a*t + v0;
  s := (1/2)*a*t^2 + v0*t;
```

$$v := a\,t + v0$$
$$s := \frac{1}{2}\,a\,t^2 + v0\,t \qquad \textbf{(2.3)}$$

運動の方向（鉛直方向上向き）をプラスとした場合，重力加速度の向きは下向きになります（ a=-g になります）．

代入する式を定義し，速度と加速度の式に代入します．

```
> param1 := [a=-g];
```

$$param1 := [a = -g] \qquad \textbf{(2.4)}$$

```
> v1 := eval(v, param1);
```

$$v1 := -g\,t + v0 \qquad \textbf{(2.5)}$$

```
> s1 := eval(s, param1);
```

$$s1 := -\frac{1}{2}\,g\,t^2 + v0\,t \qquad \textbf{(2.6)}$$

重力加速度 g の値を定義し，param2 に割り当てます．続けて，式 **(2.5)**，**(2.6)** に代入・評価します．

（復習）ラベルは，Ctrl キー ＋ L キーで使用します．

```
> param2 := [g=9.807];
```

$$param2 := [g = 9.807] \tag{2.7}$$

```
> V1 := eval((2.5), param2);
```

$$V1 := -9.807\,t + v0 \tag{2.8}$$

```
> S1 := eval((2.6), param2);
```

$$S1 := -4.903500000\,t^2 + v0\,t \tag{2.9}$$

ここで，打ち上げの速度を v0 = 10 $\left[\!\left[\frac{m}{s}\right]\!\right]$ として，param3 に割り当てます．param3 を 式 **(2.8)**, **(2.9)** に代入・評価します．

```
> param3 := [v0=10];
```

$$param3 := [v0 = 10] \tag{2.10}$$

```
> V2 := eval((2.8), param3);
```

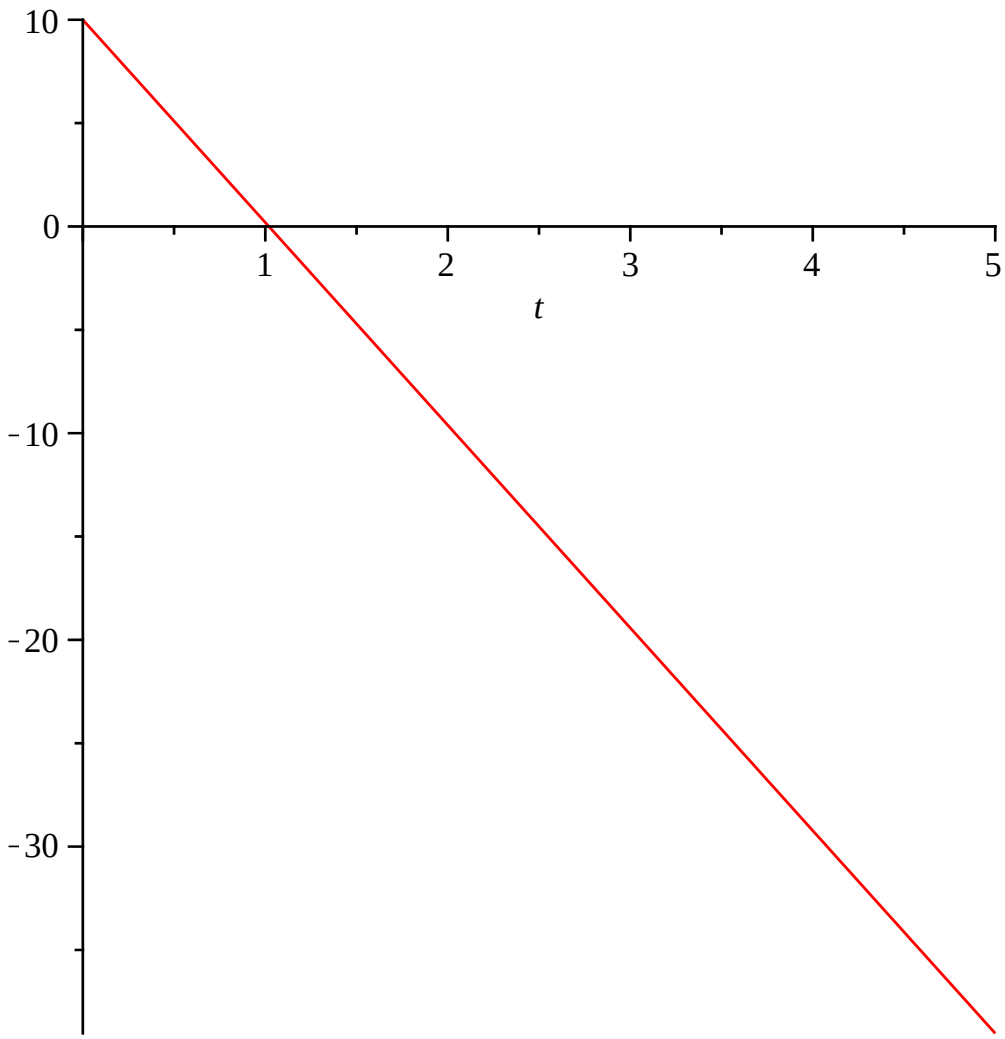
$$V2 := -9.807\,t + 10 \tag{2.11}$$

```
> S2 := eval((2.9), param3);
```

$$S2 := -4.903500000\,t^2 + 10\,t \tag{2.12}$$

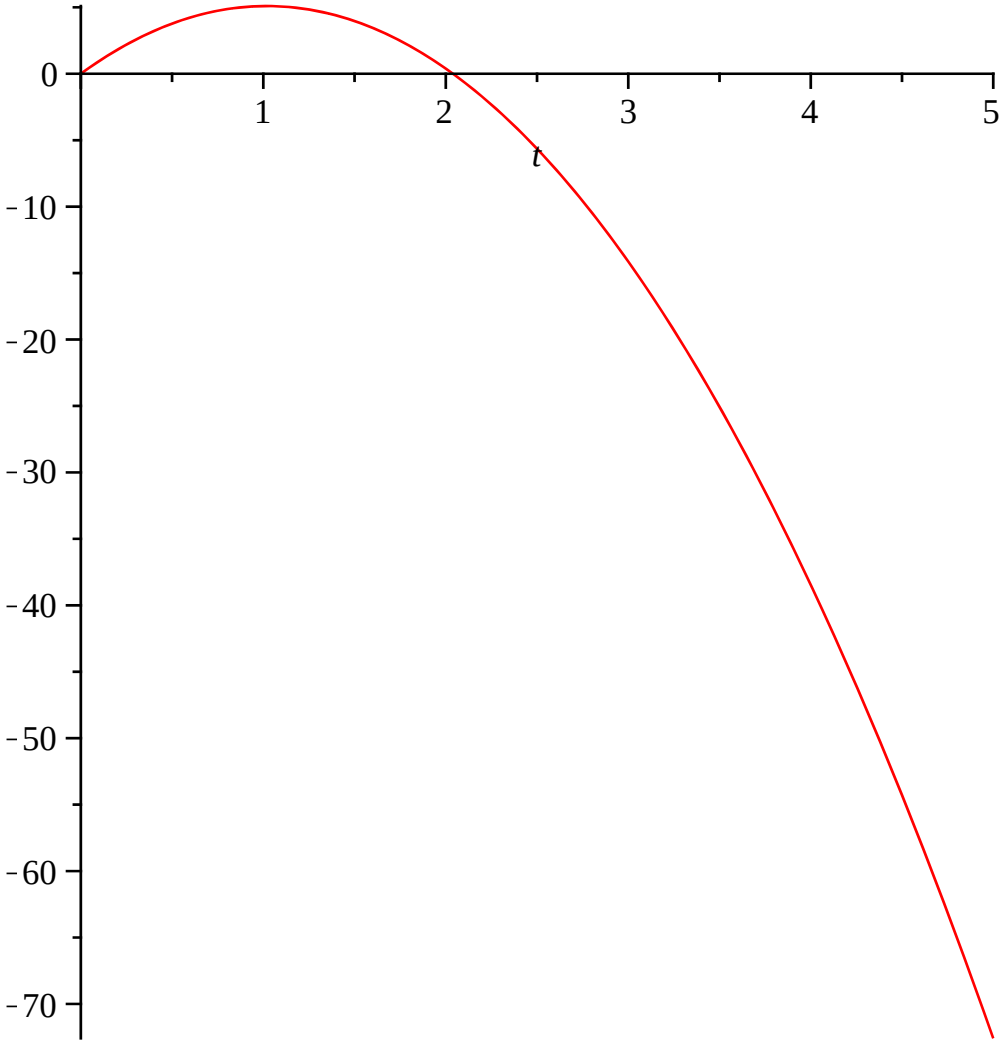
速度 V2 をプロットします（時間は 0 から 5 $[\![s]\!]$ 間）．

```
> plot(V2, t=0..5);
```



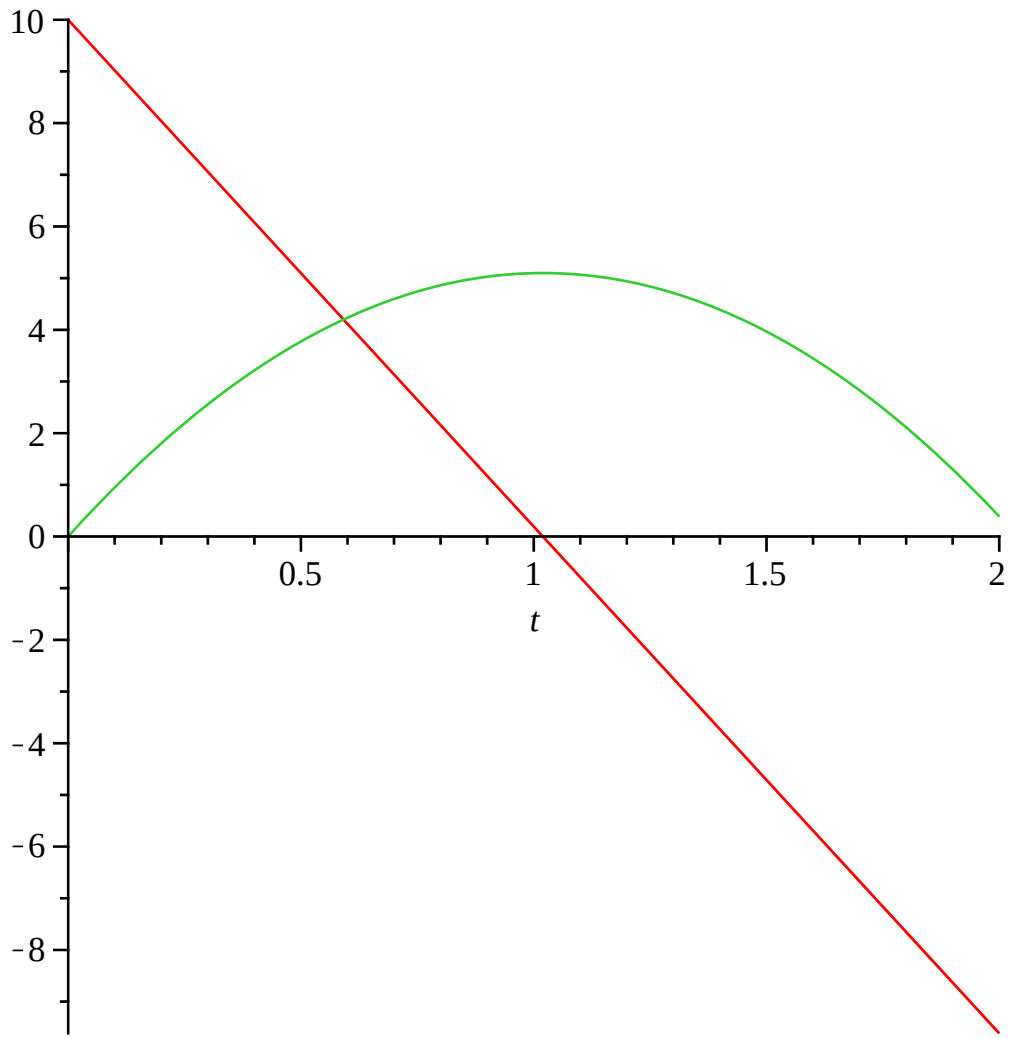
加速度 S2 をプロットします（時間は 0 から 5 ⟦s⟧ 間）.

```
> plot(S2, t=0..5);
```



速度 V2 と位置 S2 を同時にプロットします（時間は 0 から 2 〚s〛 間）．

```
> plot([V2, S2], t=0..2);
```



式のパラメータ化

式の一部をパラメータ化します.

はじめに，変数を初期化します.

```
> restart;
```

v0 をパラメータとします（変数に何も値を割り当てず，変数として式に残しておきます）．以下のように，等加速度直線運動の速度と加速度の式をそれぞれ再度定義します.

速度の式 v を定義します.

```
> v := a*t + v0;
```

$$v := a\,t + v0 \qquad \textbf{(2.1.1)}$$

加速度の式 s を定義します.

```
> s := (1/2)*a*t^2 + v0*t;
```

$$s := \frac{1}{2} a t^2 + v0 t \qquad \textbf{(2.1.2)}$$

v0 = 10 のとき

はじめに，重力加速度と初速度 v0=10 を定義し，p1 に割り当てます．

```
> p1 := [a=-9.807, v0=10];
```

$$p1 := [a = -9.807, v0 = 10] \qquad \textbf{(2.1.1.1)}$$

位置の式 s に p1 を代入し，結果を s1 に割り当てます．

```
> s1 := eval(s, p1);
```

$$s1 := -4.903500000\, t^2 + 10\, t \qquad \textbf{(2.1.1.2)}$$

以下，v0 = 20，v0 = 30 として，同様に定義します．

v0 = 20 のとき

```
> p2 := [a=-9.807, v0=20];
```

$$p2 := [a = -9.807, v0 = 20] \qquad \textbf{(2.1.2.1)}$$

```
> s2 := eval(s, p2);
```

$$s2 := -4.903500000\, t^2 + 20\, t \qquad \textbf{(2.1.2.2)}$$

v0 = 30 のとき

```
> p3 :=  [a=-9.807, v0=30];
```

$$p3 := [a = -9.807, v0 = 30] \qquad \textbf{(2.1.3.1)}$$

```
> s3 := eval(s, p3);
```

$$s3 := -4.903500000\, t^2 + 30\, t \qquad \textbf{(2.1.3.2)}$$

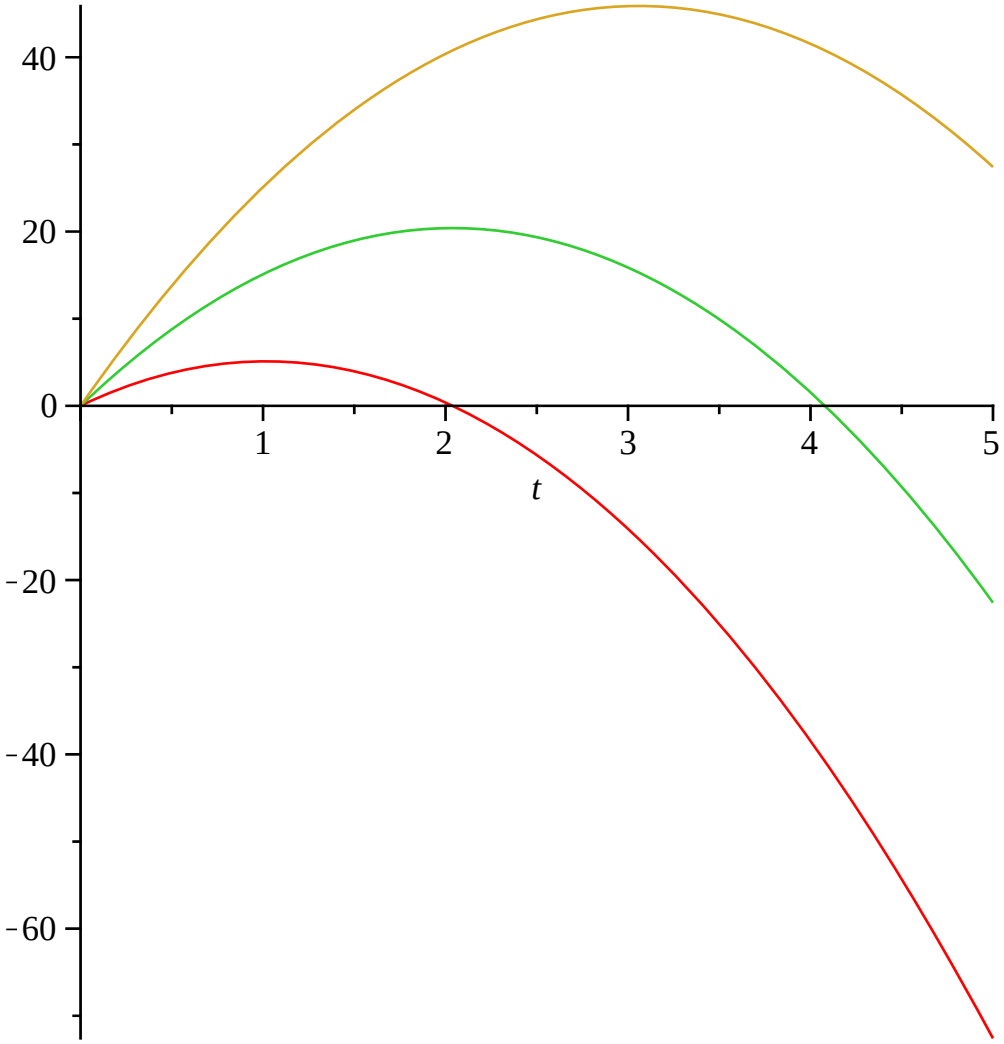
結果のプロット

結果をひとつにまとめ，プロットします（時間は 0 から 5 ⟦s⟧ 間）．

```
> sList := [s1, s2, s3];
```

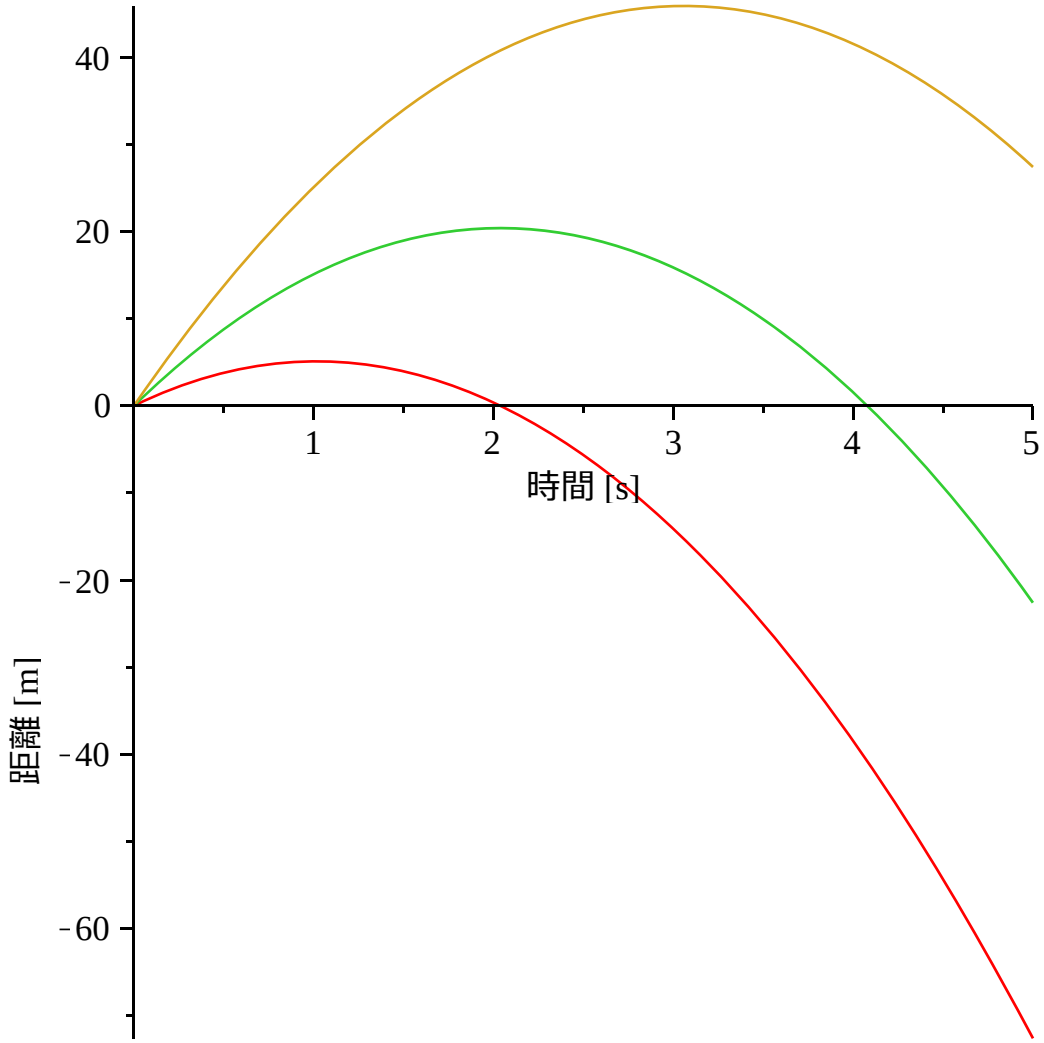
$$sList := [-4.903500000\, t^2 + 10\, t,\ -4.903500000\, t^2 + 20\, t,\ -4.903500000\, t^2 + 30\, t] \qquad \textbf{(2.1.4.1)}$$

```
> plot(sList, t=0..5);
```



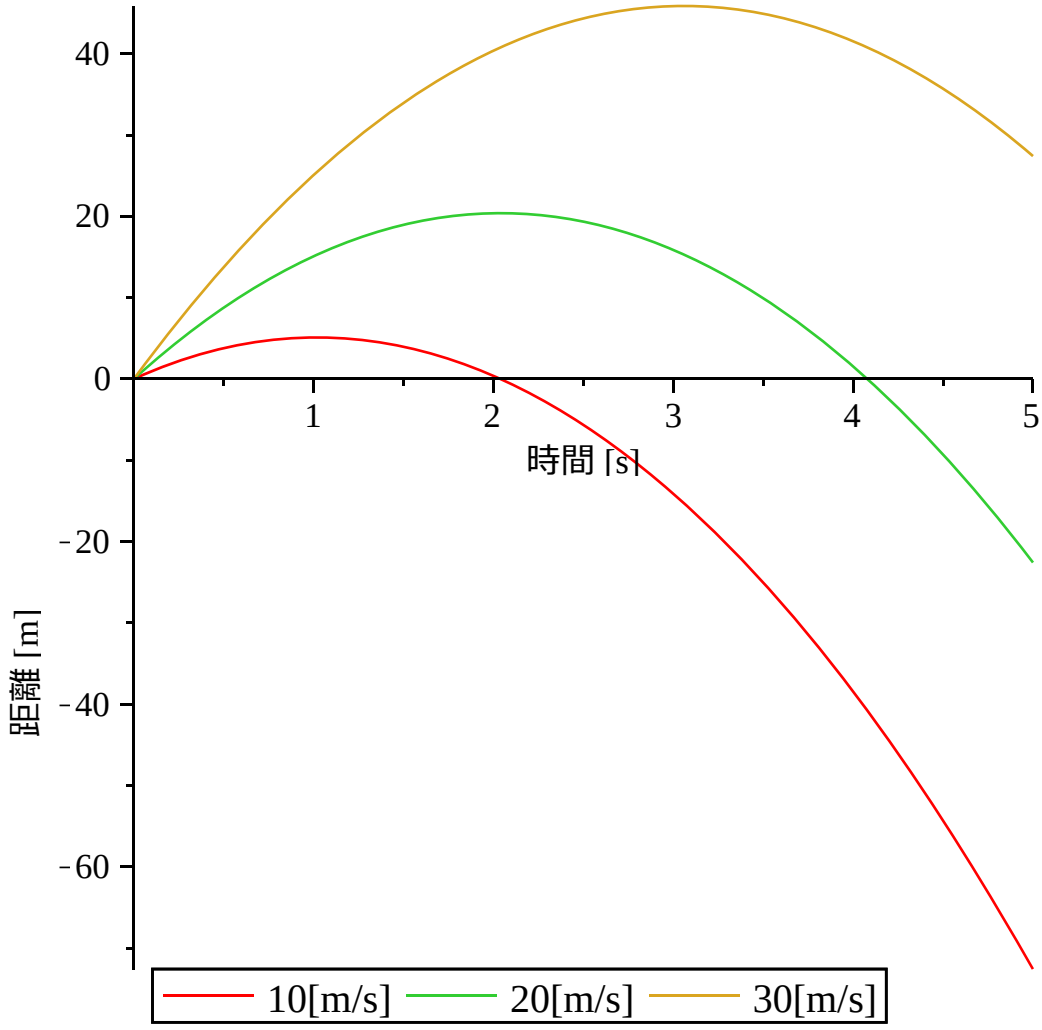
ラベルを追加します.

```
> plot(sList, t=0..5, labels = ["時間 [s]", "距離 [m]"],
                      labeldirections = [default, vertical]);
```



レジェンドを追加します.

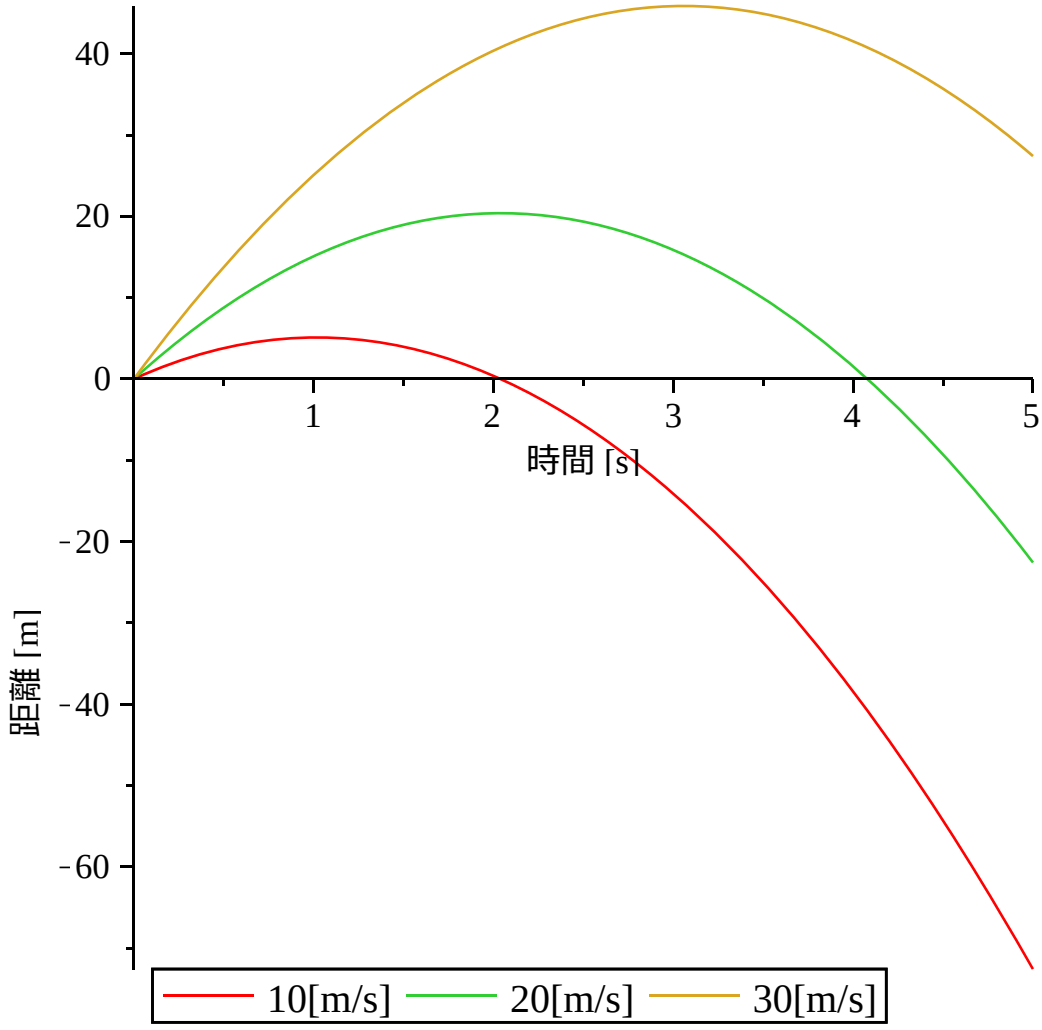
```
> plot(sList, t=0..5, labels = ["時間 [s]", "距離 [m]"],
                      labeldirections = [default, vertical],
                      legend = ["10[m/s]","20[m/s]","30[m/s]"]);
```



コマンドを整理し，コメントを追加します．

（参考）行の先頭に # を追加すると，# の後ろはコメントとして扱われます（処理に含まれません）．

```
> # プロット
  plot(sList,
       t=0..5,                                    # プロット範囲（時間）
       labels=["時間 [s]", "距離 [m]"],           # 軸ラベル
       labeldirections=[default, vertical],       # 軸ラベルの方向
       legend = ["10[m/s]","20[m/s]","30[m/s]"] # レジェンド
  );
```



オプションの値を変数として定義します.

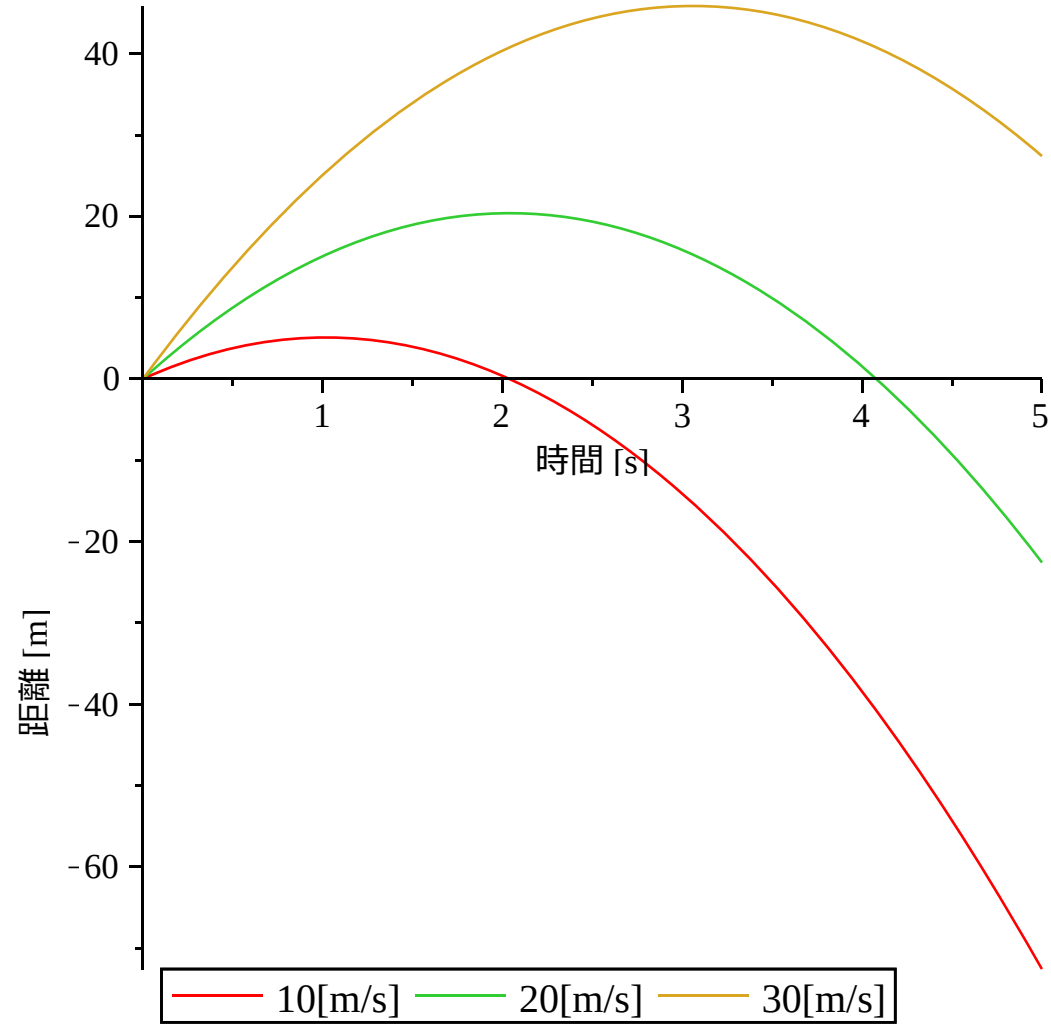
```
> myTime     := 5;
  myLabel    := ["時間 [s]", "距離 [m]"];
  myLabelDir := [default, vertical];
  myLegend   := ["10[m/s]","20[m/s]","30[m/s]"];

  plot(sList,
       t               = 0..myTime,
       labels          = myLabel,
       labeldirections = myLabelDir,
       legend          = myLegend
  );
```

$$myTime := 5$$

$$myLabel := [\text{"時間 [s]"}, \text{"距離 [m]"}]$$

$$myLabelDir := [default, vertical]$$

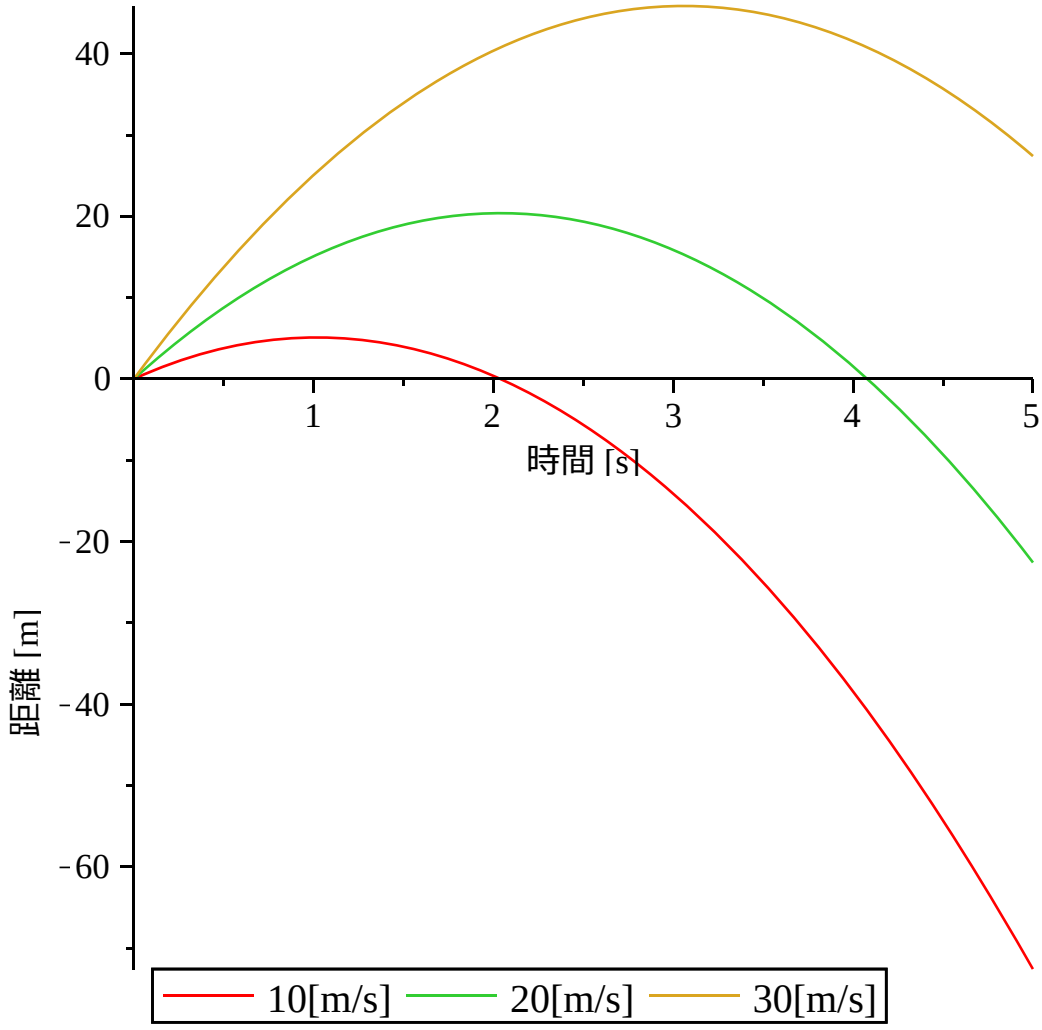
$$myLegend := [\text{"10[m/s]"}, \text{"20[m/s]"}, \text{"30[m/s]"}]$$



オプション群をひとつの変数にまとめます.

```
> myOptions := t = 0..5,
               labels = ["時間 [s]", "距離 [m]"],
               labeldirections = [default, vertical],
               legend = [ "10[m/s]", "20[m/s]", "30[m/s]" ];

   plot(sList, myOptions);
```

myOptions := *t* = 0 ..5, *labels* = ["時間 [s]", "距離 [m]"], *labeldirections* = [*default*, *vertical*], *legend* = ["10[m/s]", "20[m/s]", "30[m/s]"]

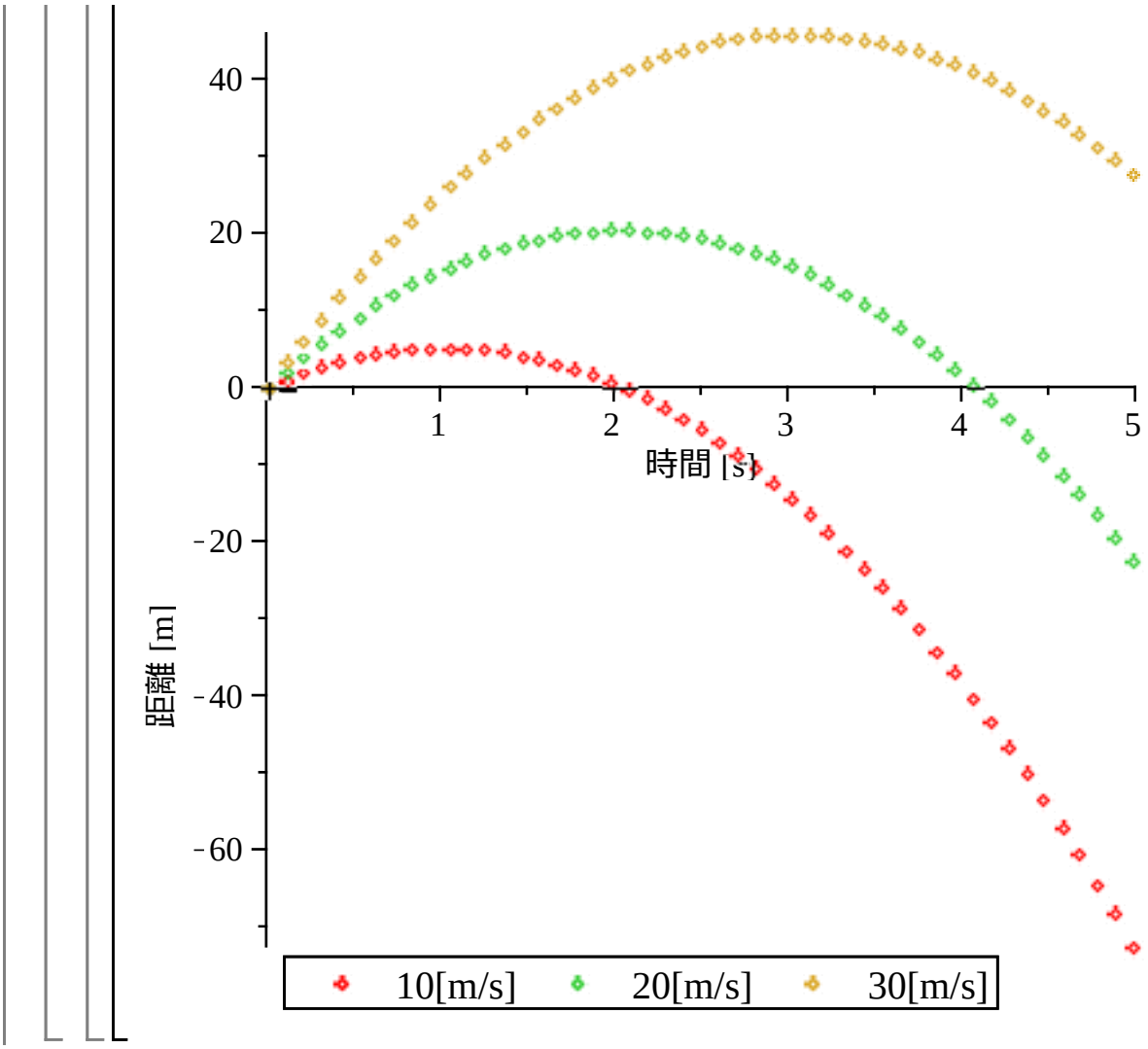


プロットの線を点で描画しなおします.

（参考）ツールバーからも様々修正が可能です.

```
> myOptions := t = 0..5,
               labels = ["時間 [s]", "距離 [m]"],
               labeldirections = [default, vertical],
               legend = [ "10[m/s]", "20[m/s]", "30[m/s]" ],
               style = point;

   plot(sList, myOptions);
```

myOptions := *t* = 0 ..5, *labels* = ["時間 [s]", "距離 [m]"], *labeldirections* = [*default*, *vertical*], *legend* = ["10[m/s]", "20[m/s]", "30[m/s]"], *style* = *point*


40
20
0
-20
-40
-60
1
2
3
4
5
時間 [s]
距離 [m]
10[m/s]
20[m/s]
30[m/s]

アニメーションの作成

投げ上げ運動のアニメーションを作成します．

はじめに初期化します．

```
> restart;
```

等加速度直線運動の速度と位置の式を定義します．ここでは，v0 の変化によって，プロットが連続的に変化する様子をアニメーションとして作成します．

```
> v := a*t + v0;
  s := (1/2)*a*t^2 + v0*t;
```

$$v := a\,t + v0$$

$$s := \frac{1}{2}\,a\,t^2 + v0\,t \qquad \textbf{(2.2.1)}$$

plots パッケージをロードします．

```
> with(plots);
```

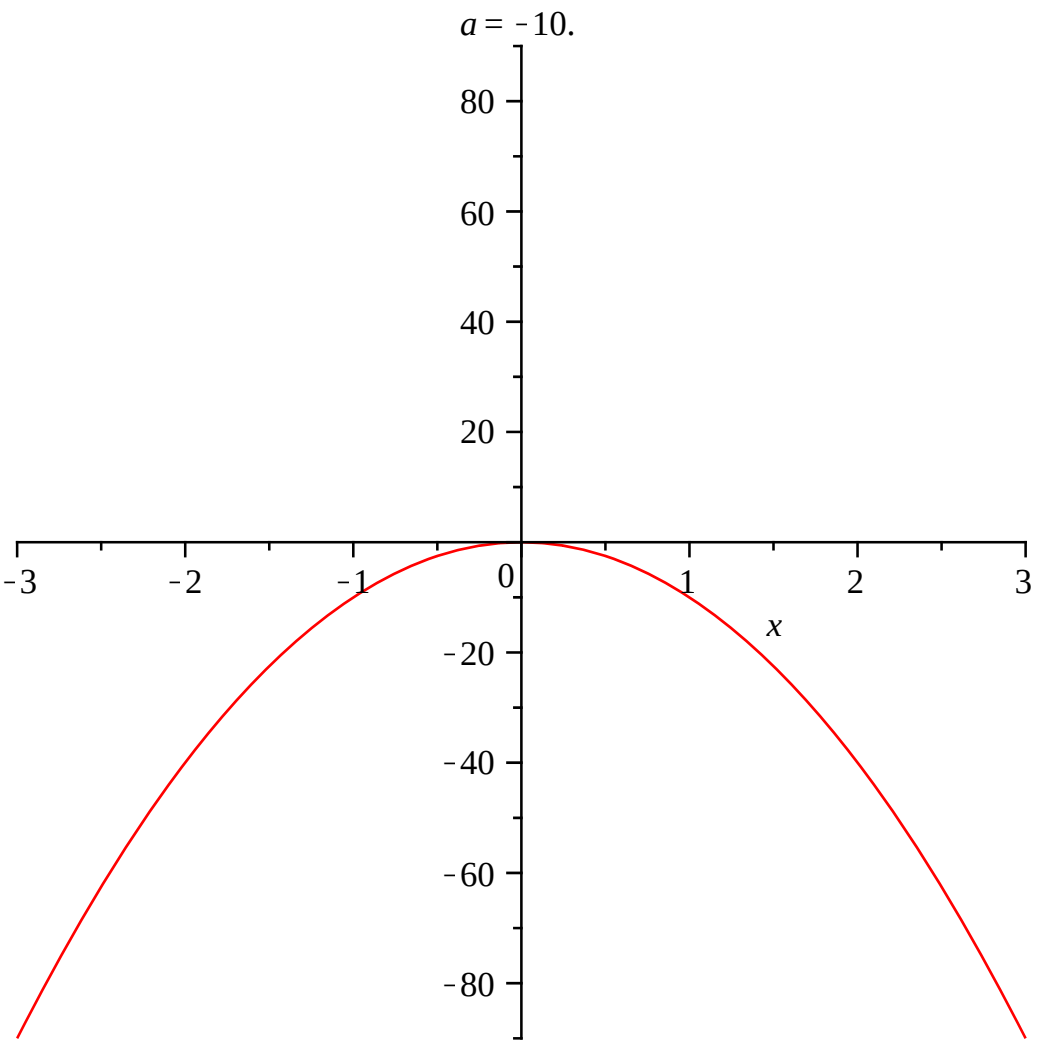
[*animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d, densityplot, display, dualaxisplot, fieldplot, fieldplot3d, gradplot, gradplot3d, graphplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams, intersectplot, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot, rootlocus, semilogplot, setcolors, setoptions, setoptions3d, spacecurve, sparsematrixplot, surfdata, textplot, textplot3d, tubeplot*] **(2.2.2)**

以下は，アニメーションの例題です．プロットのアニメーションには，plots パッケージの animate コマンドを使用します．

（参考） 2 次曲線のアニメーション

$a\,x^2$ を x = -3 から x = 3 までの範囲でプロットします（範囲は固定）．ただし，式中の a を -10 から 10 までの範囲で変動させます（a はパラメータ）．アニメーションの操作は，下のプロット図を選択しコンテキストバーから行います．

```
> animate(plot, [a*x^2, x=-3..3], a=-10..10);
```



（復習）
アニメーションの操作は，以下の操作パネルから行うことができます．
操作パネルは，プロットされたグラフをクリックするとツールバーの下に表示されます．

投げ上げ運動として，加速度 a を -9.807 と定義します（上向きがプラス．重力加速度は下向き）．

```
> param := [a=-9.807];
```

$$param := [a = -9.807] \qquad \textbf{(2.2.3)}$$

位置の式に param を代入し評価します．

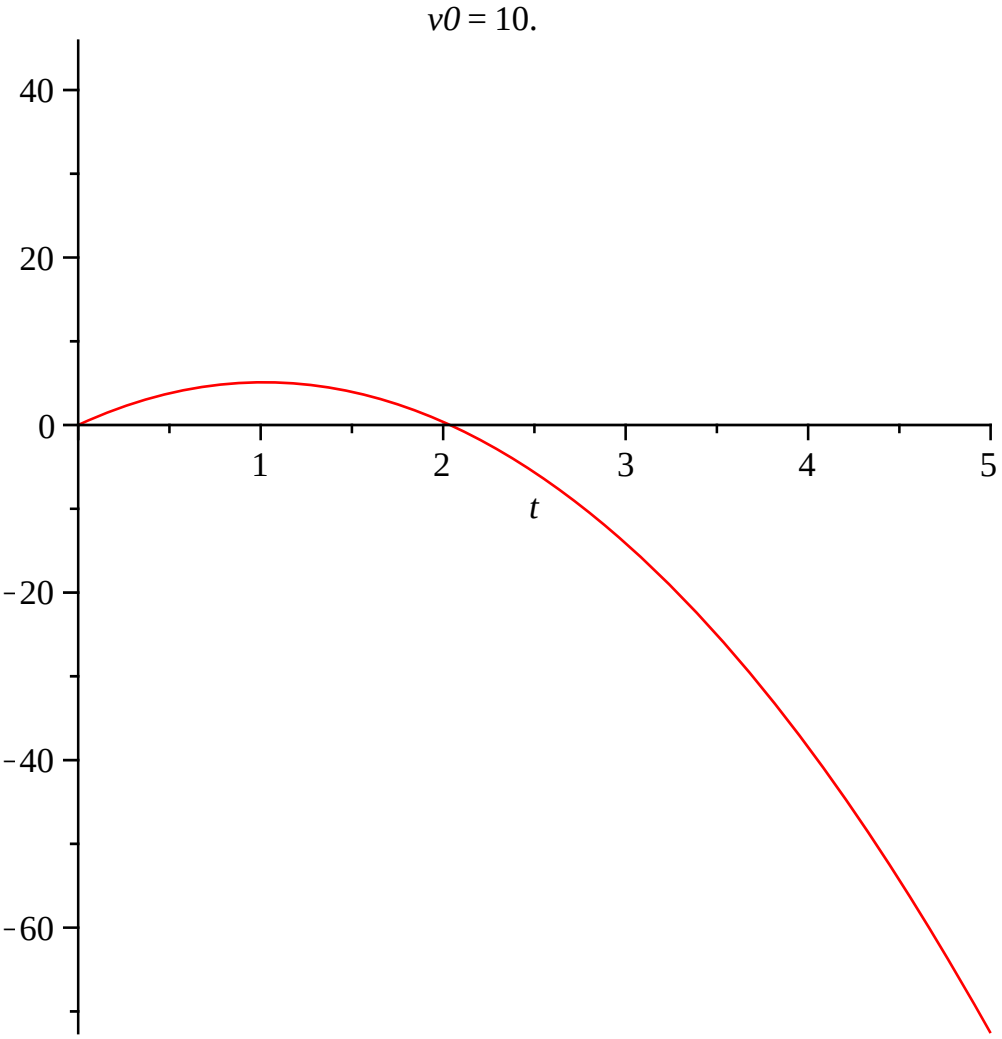
```
> S := eval(s, param);
```

$$S := -4.903500000\,t^2 + v0\,t \qquad \textbf{(2.2.4)}$$

位置の式 $-4.9035\,t^2 + v0\,t$ を t = 0 から x = 5 までの範囲でプロットします（範囲は固定）．ただ

し，式中の初速度 *v0* を 10 から 30 までの範囲で変動させます（*v0* はパラメータ）．

```
> animate(plot, [S, t=0..5], v0=10..30);
```



投げ上げ運動の初速度に対する位置の変化を，もう少し分りやすくアニメーション化します．

ここで，点（ポイント）プロットを使用します．

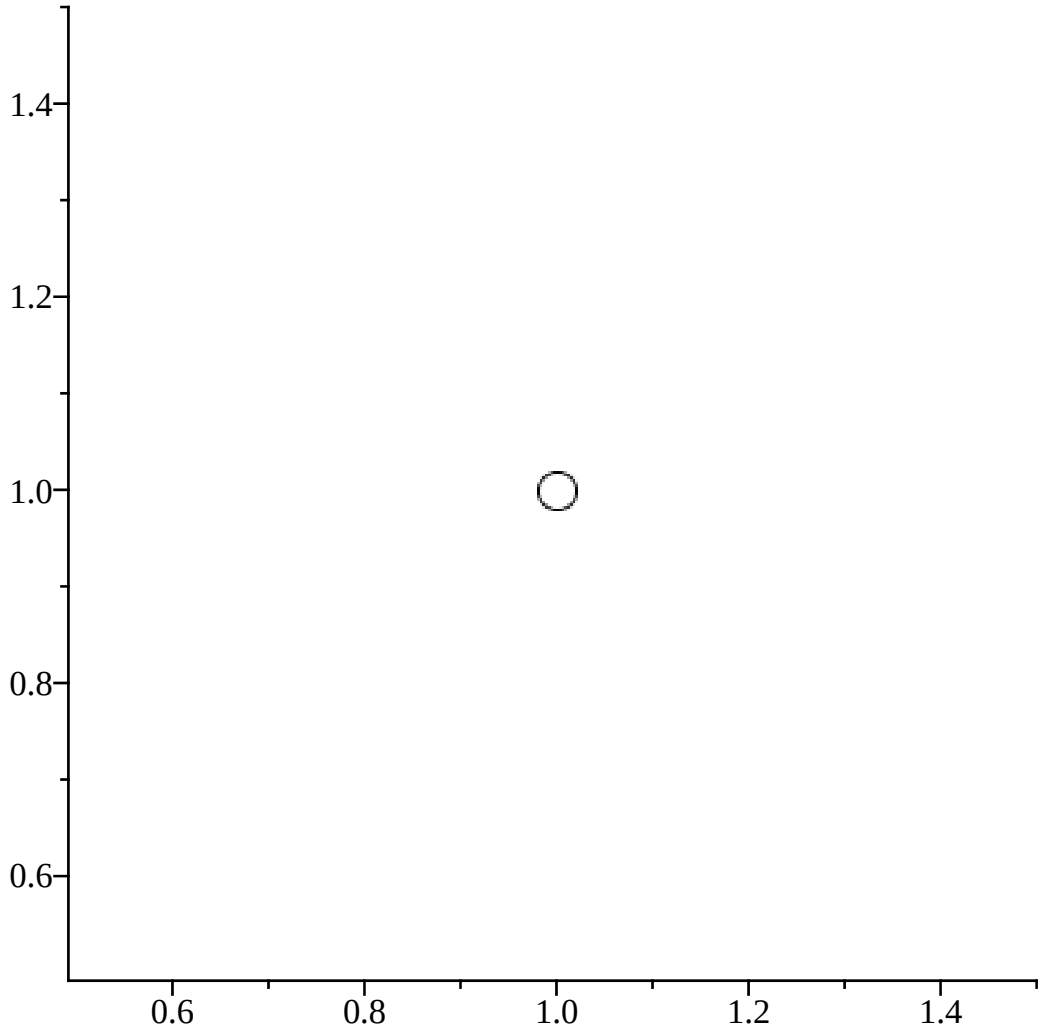
以下は，点プロットの例題です．点［１，１］の１点をプロットします．

オプションとして，

- symbol=circle（点を円として表現），
- symbolsize=30（点のサイズを30ポイントとしてスケーリング）

を追加します．

```
> pointplot([1,1], symbol=circle, symbolsize=30);
```



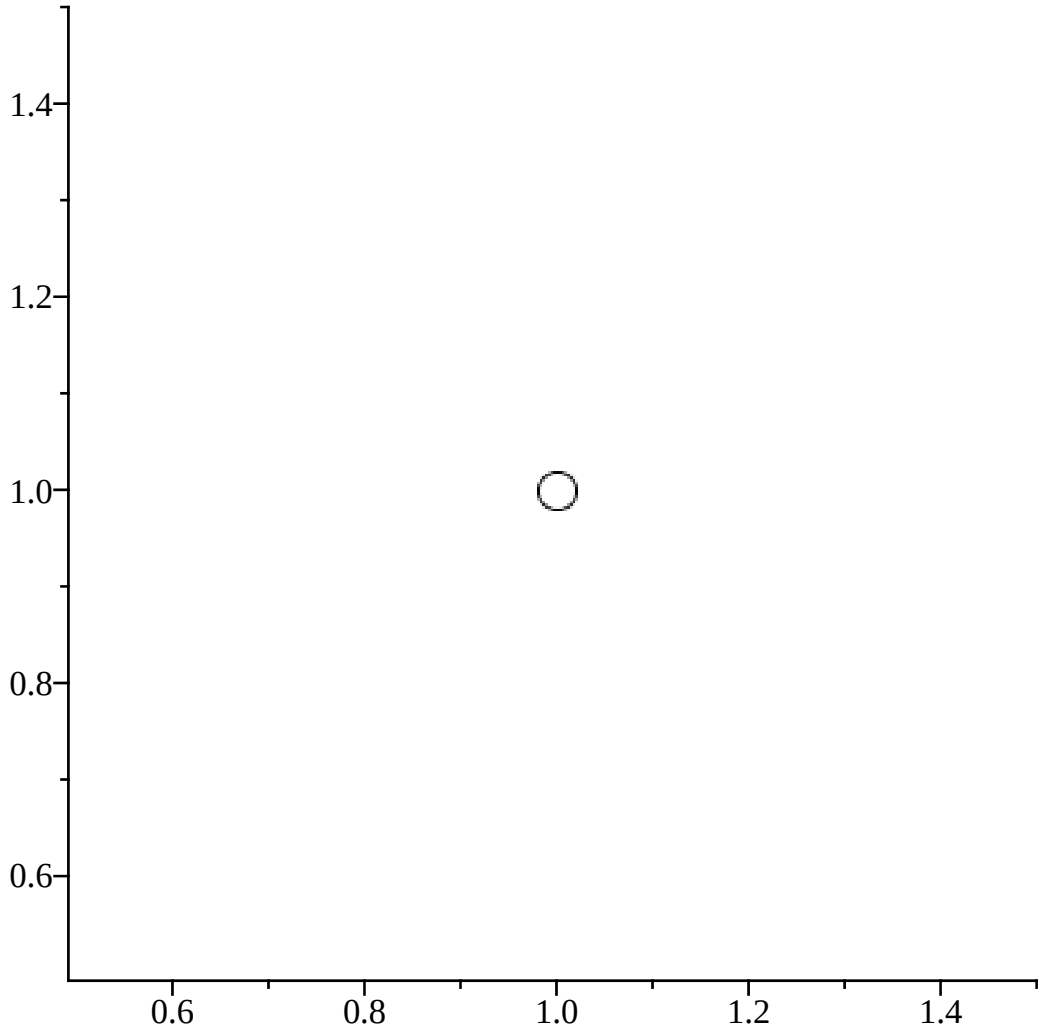
点を変数 points に割り当てます.

```
> points := [1,1];
```

$$points := [1, 1] \quad \textbf{(2.2.5)}$$

再度，点をプロットします.

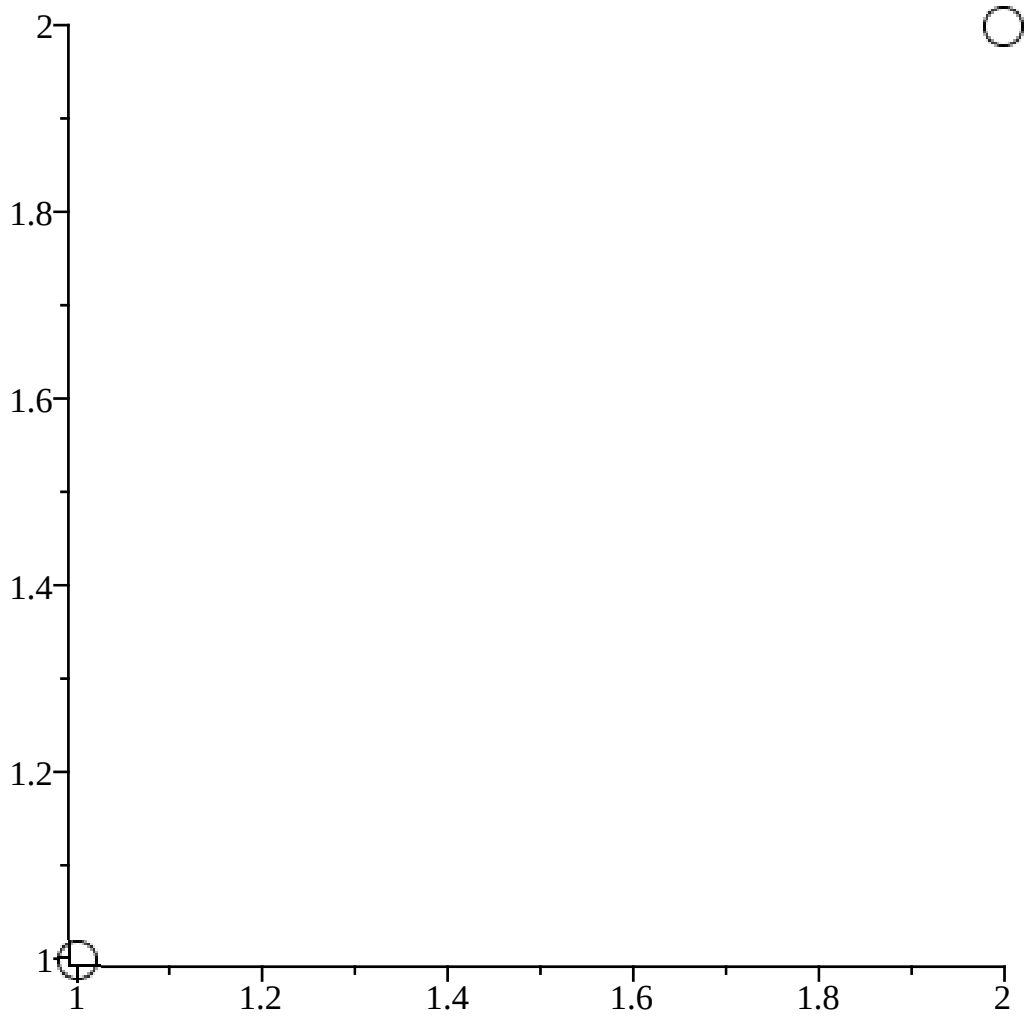
```
> pointplot(points, symbol=circle, symbolsize=30);
```



今度は，２点定義し，プロットします．

```
> points := [ [1,1], [2,2] ];
  pointplot(points, symbol=circle, symbolsize=30);
```

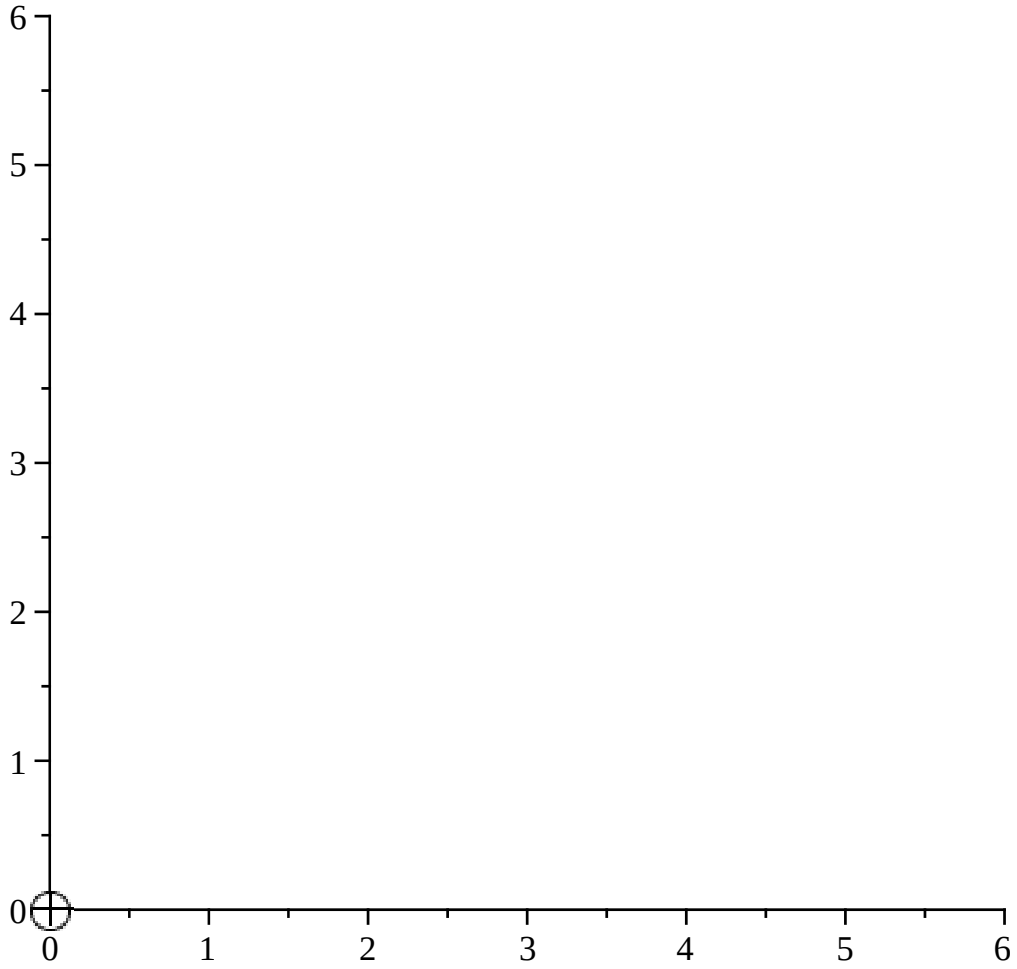
$$points := [[1, 1], [2, 2]]$$



点プロットを用いてアニメーションを作成します．点 [t, t] ○ を t = 0..6 の範囲で変動させます．

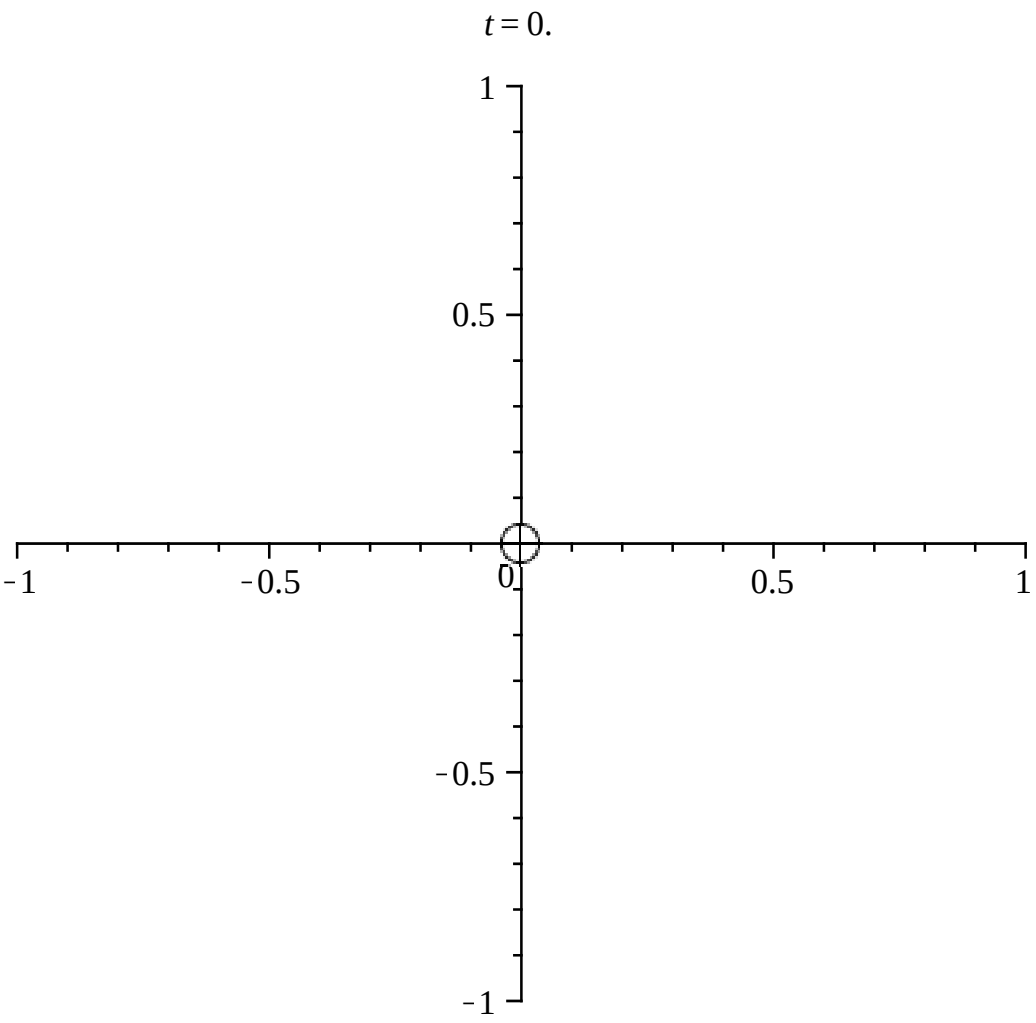
```
> animate(pointplot, [ [t,t], symbol=circle, symbolsize=30 ], t=0..6 );
```

$t = 0.$



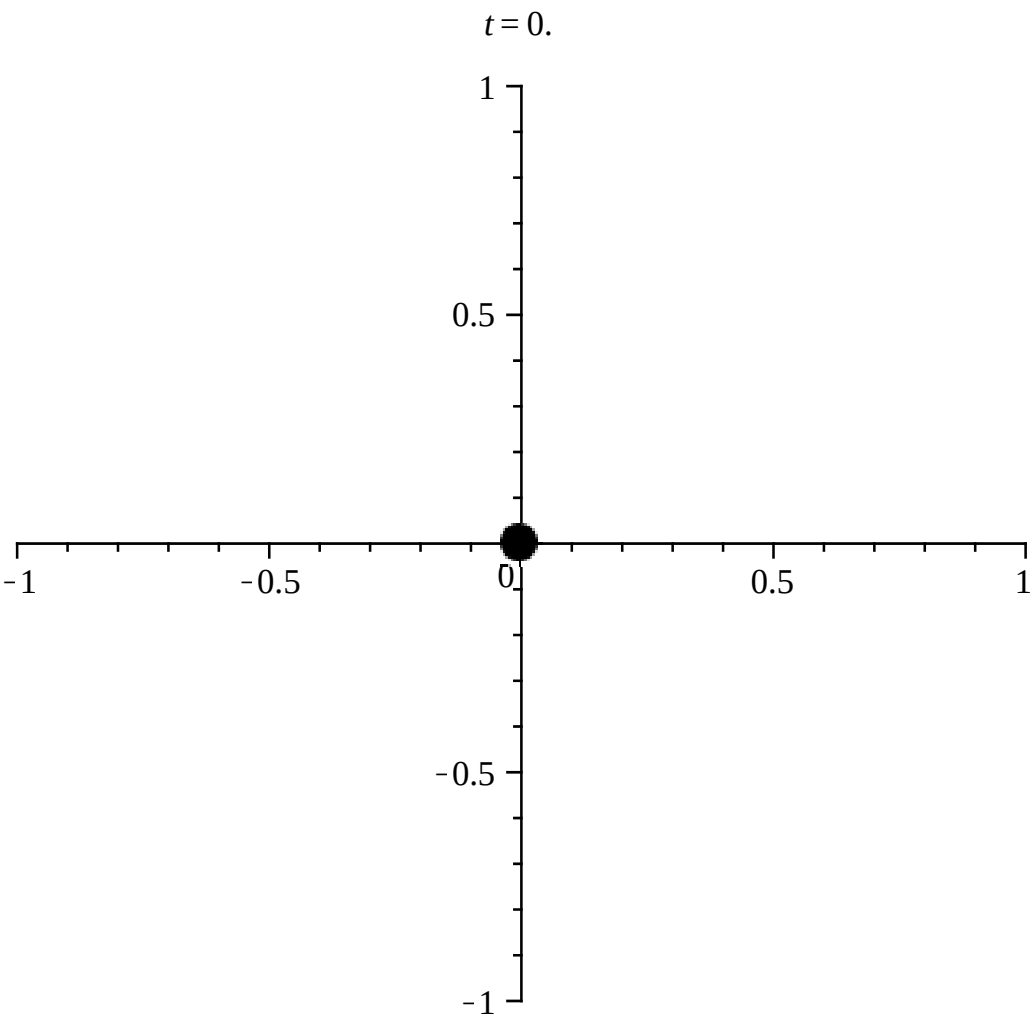
点を [0,sin(t)] として，t=0..2*Pi の範囲で変動させます.

```
> animate(pointplot, [[0,sin(t)], symbol=circle, symbolsize=30], t=0.
  .2*Pi);
```



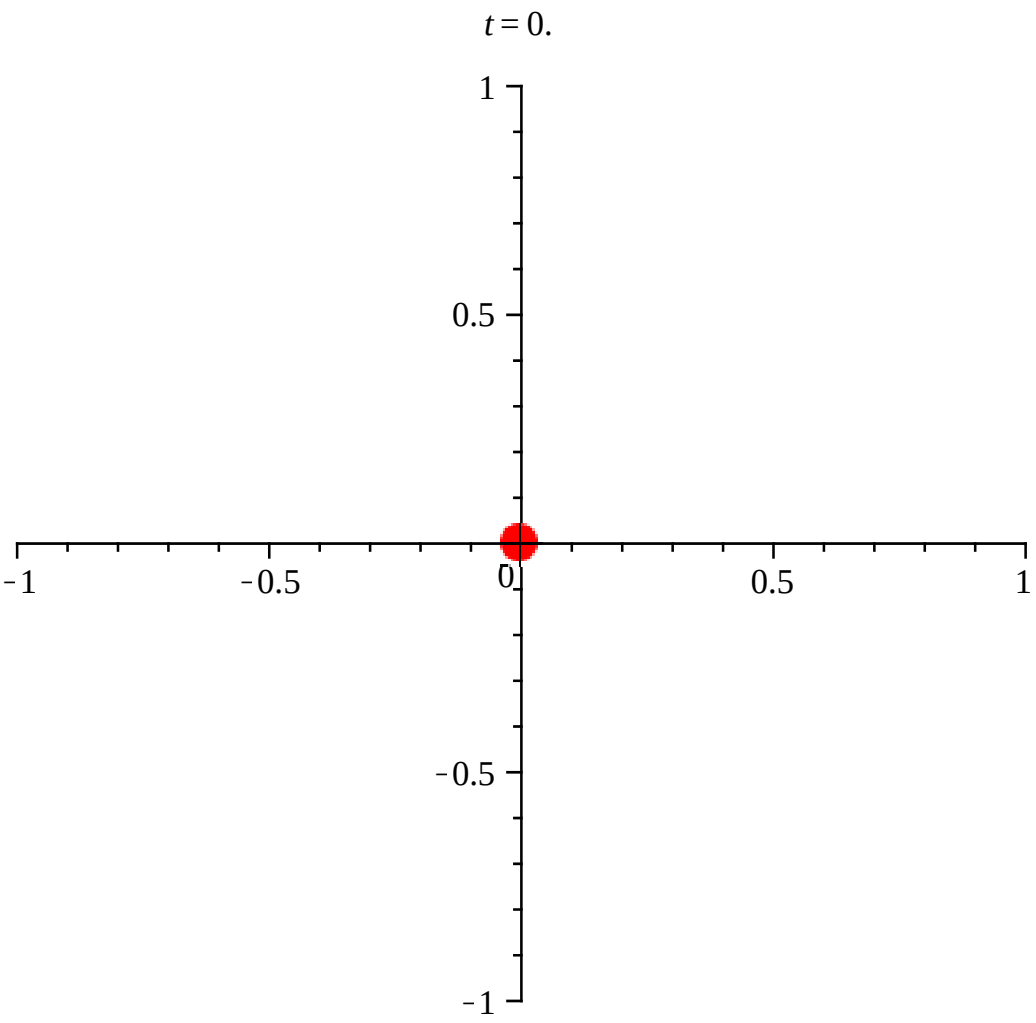
わかりやすいように点 ○ を ● に変更します．symbol=solidcircle とします．

```
> animate(pointplot, [[0,sin(t)], symbol=solidcircle, symbolsize=30],
  t=0..2*Pi);
```



● の色を赤に変更します.

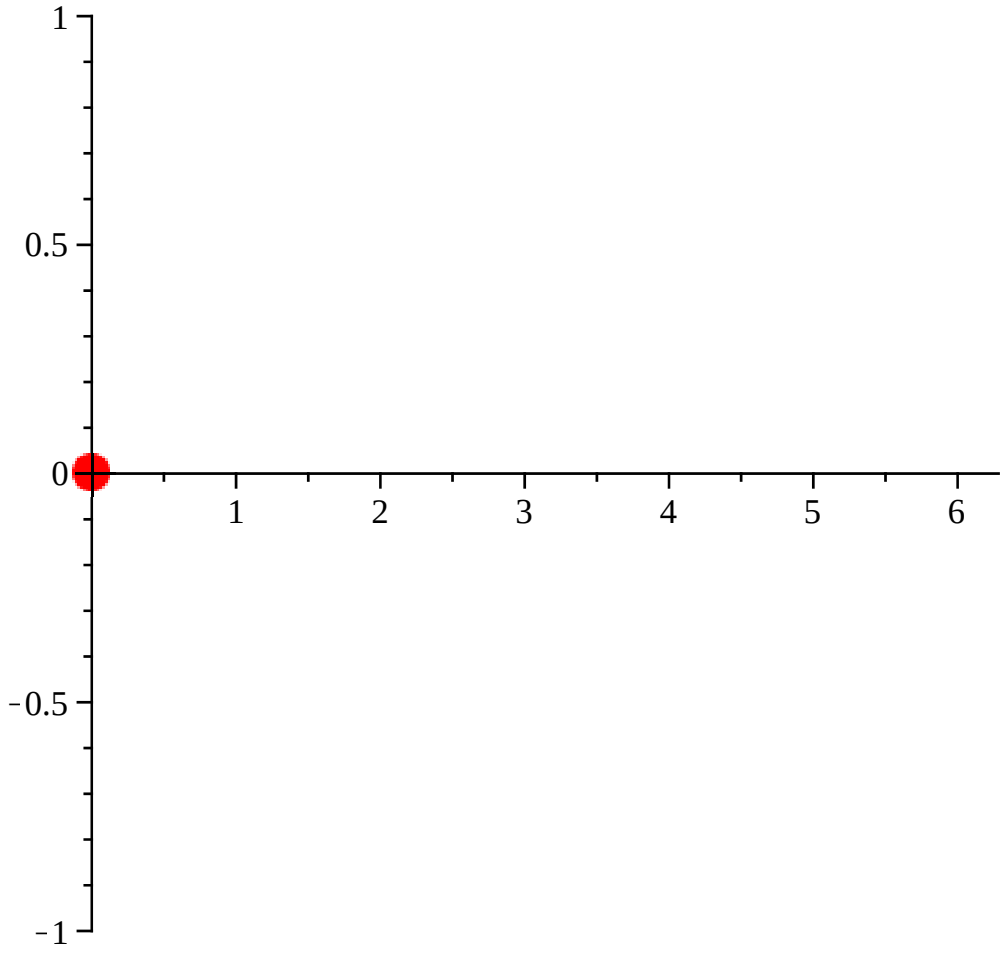
```
> animate(pointplot, [[0,sin(t)], symbol=solidcircle, symbolsize=30,
  color=red], t=0..2*Pi);
```



[0,sin(t)] を [t,sin(t)] に変更します.

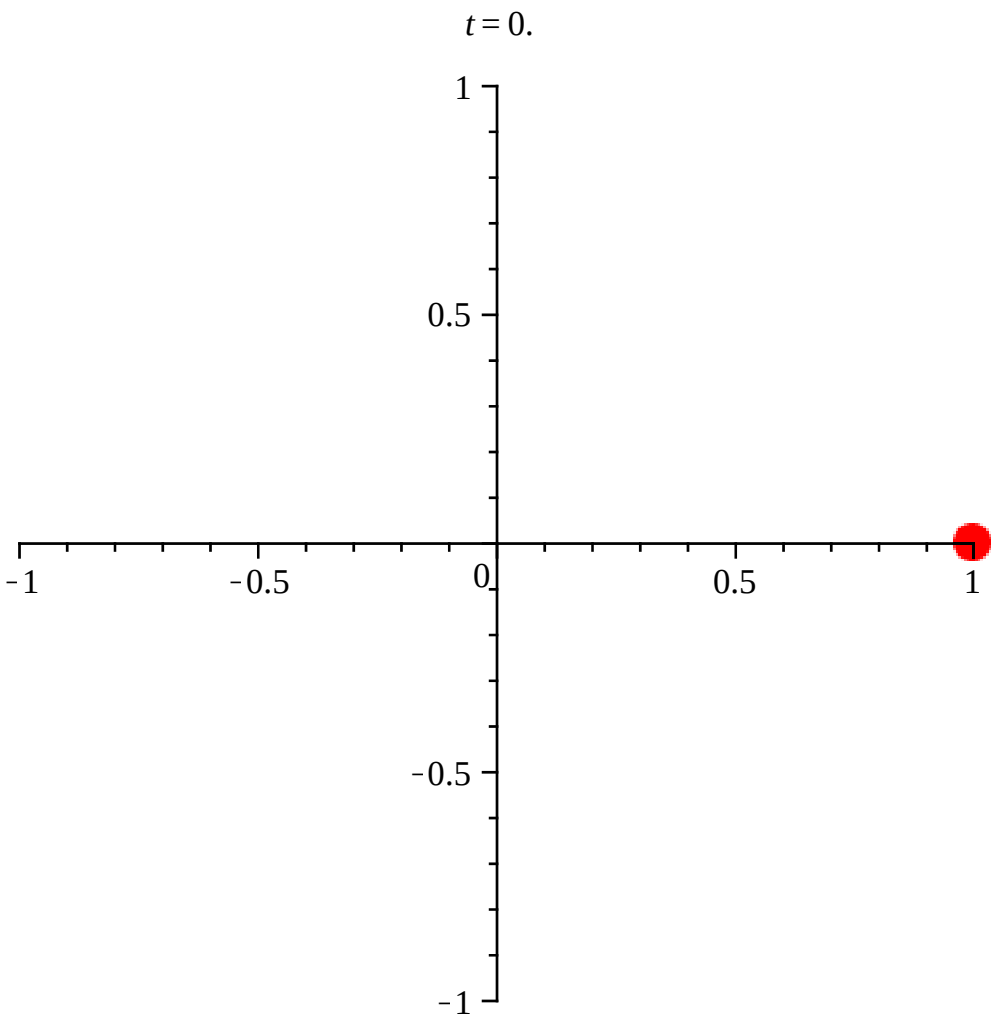
```
> animate(pointplot, [[t,sin(t)], symbol=solidcircle, symbolsize=30,
  color=red], t=0..2*Pi);
```

$t = 0.$



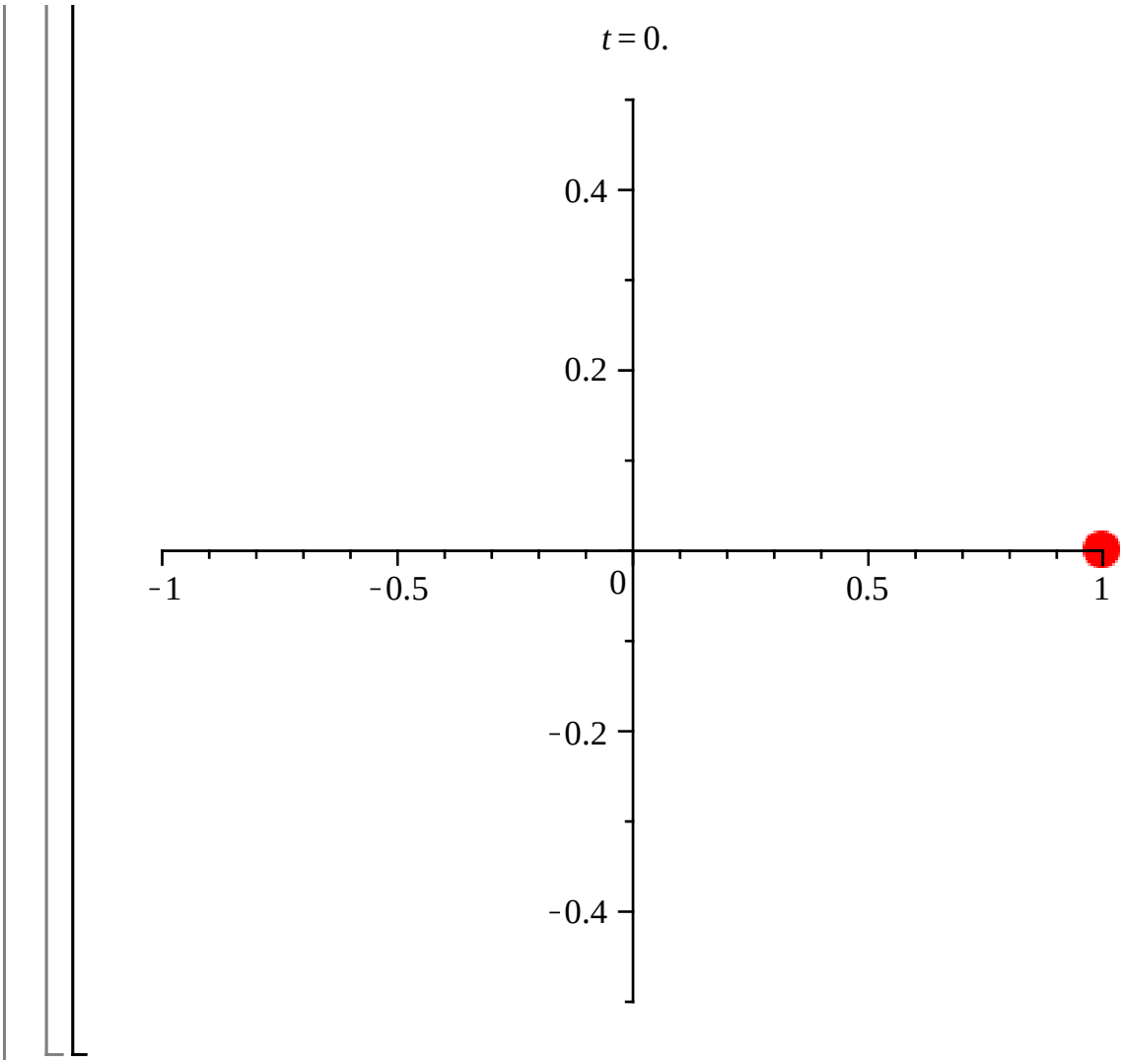
[cos(t),sin(t)] を試してみます.

```
> animate(pointplot, [[cos(t),sin(t)], symbol=solidcircle, symbolsize=
  30, color=red], t=0..2*Pi);
```



さらに，[cos(t), sin(t)*cos(t)] を確認します.

```
> animate(pointplot, [[cos(t),sin(t)*cos(t)], symbol=solidcircle,
  symbolsize=30, color=red], t=0..2*Pi);
```


$t = 0.$
0.4
0.2
-1
-0.5
0
0.5
1
-0.2
-0.4

投げ上げ運動のアニメーション作成

初速度 v0 = 10 の場合
初速度 v0 = 20 の場合
初速度 v0 = 30 の場合

について（投げ上げ運動）アニメーションを作成します.

式 **(2.2.4)** にそれぞれの初速度を代入して，式を定義します.

```
> (2.2.4);
```

$$-4.903500000\,t^2 + v0\,t \qquad \textbf{(2.3.1)}$$

```
> S10 := eval(S, [v0=10]);
```

$$S10 := -4.903500000\,t^2 + 10\,t \qquad \textbf{(2.3.2)}$$

```
> S20 := eval(S, [v0=20]);
```

$$S20 := -4.903500000\,t^2 + 20\,t \qquad \textbf{(2.3.3)}$$

```
> S30 := eval(S, [v0=30]);
```

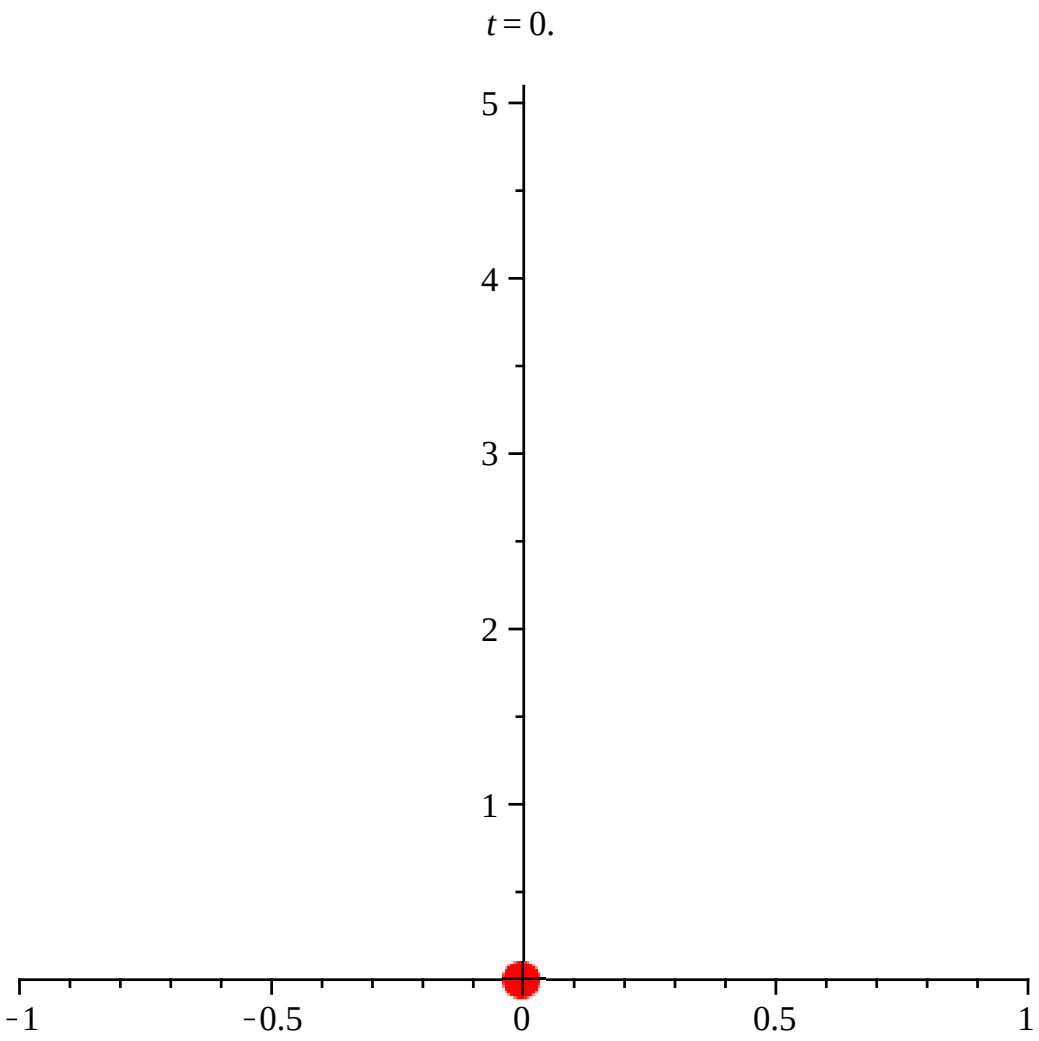
$$S30 := -4.903500000\,t^2 + 30\,t \qquad \textbf{(2.3.4)}$$

初速度 v0 = 10 の場合

点 [0, S10]（実際は，$[0, -4.903500000\,t^2 + 10\,t]$）を t=0..endTime で変化させます．ただし，endTime = 2 とします.

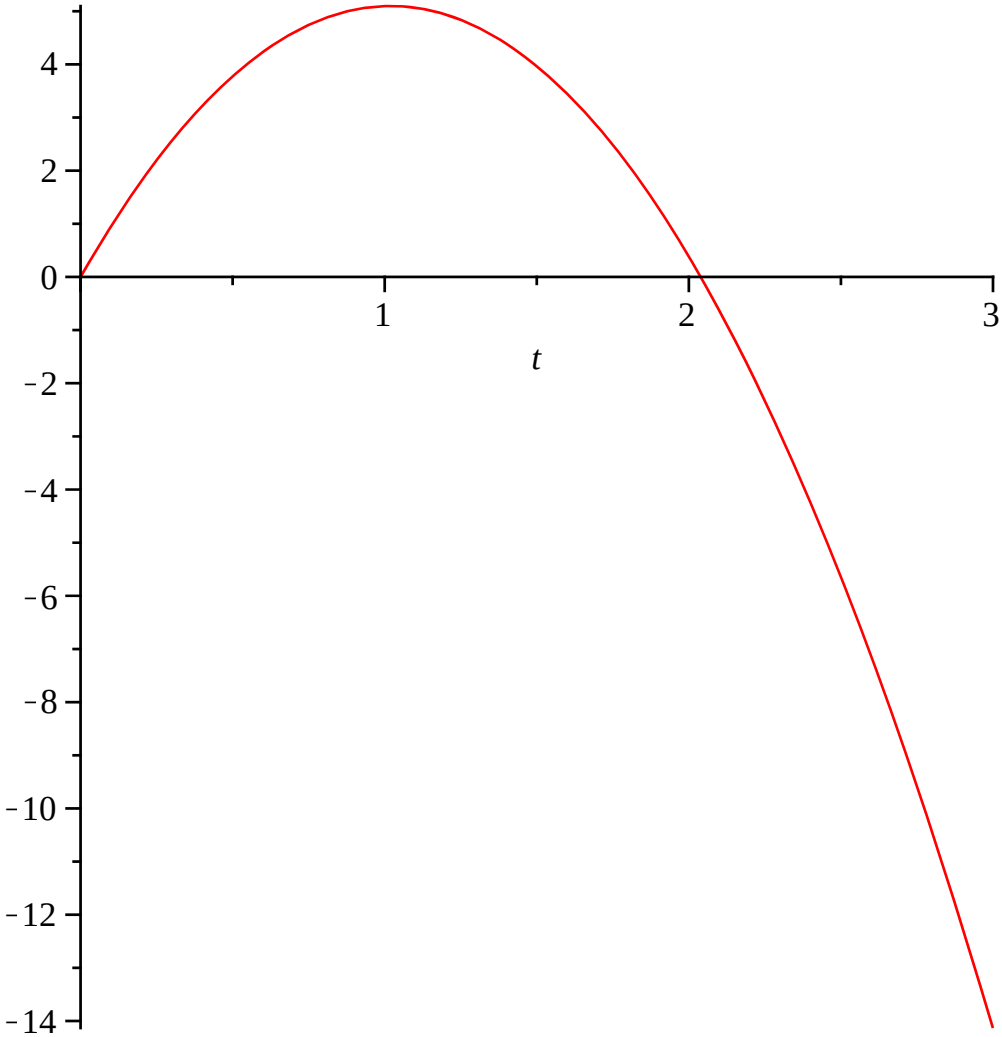
```
> endTime := 2;
  animate( pointplot, [ [0, S10], symbol=solidcircle, symbolsize =
  30, color = red ], t=0..endTime);
```

$$endTime := 2$$



再度，S10 をプロットします．

```
> plot(S10, t=0..3);
```



物体が投げ上げられて，もとの位置に戻ってくるまでの時間は，t=0 を除く，赤い 2 次曲線と横軸の交点になります．その交点を式 **(2.3.2)** を t について解くことで求めます（solve コマンドを使用します）．

```
> sol1 := solve(S10, t);
```

$$sol1 := 0., 2.039359641 \qquad \textbf{(2.3.1.1)}$$

2.039359641 がその交点になります．

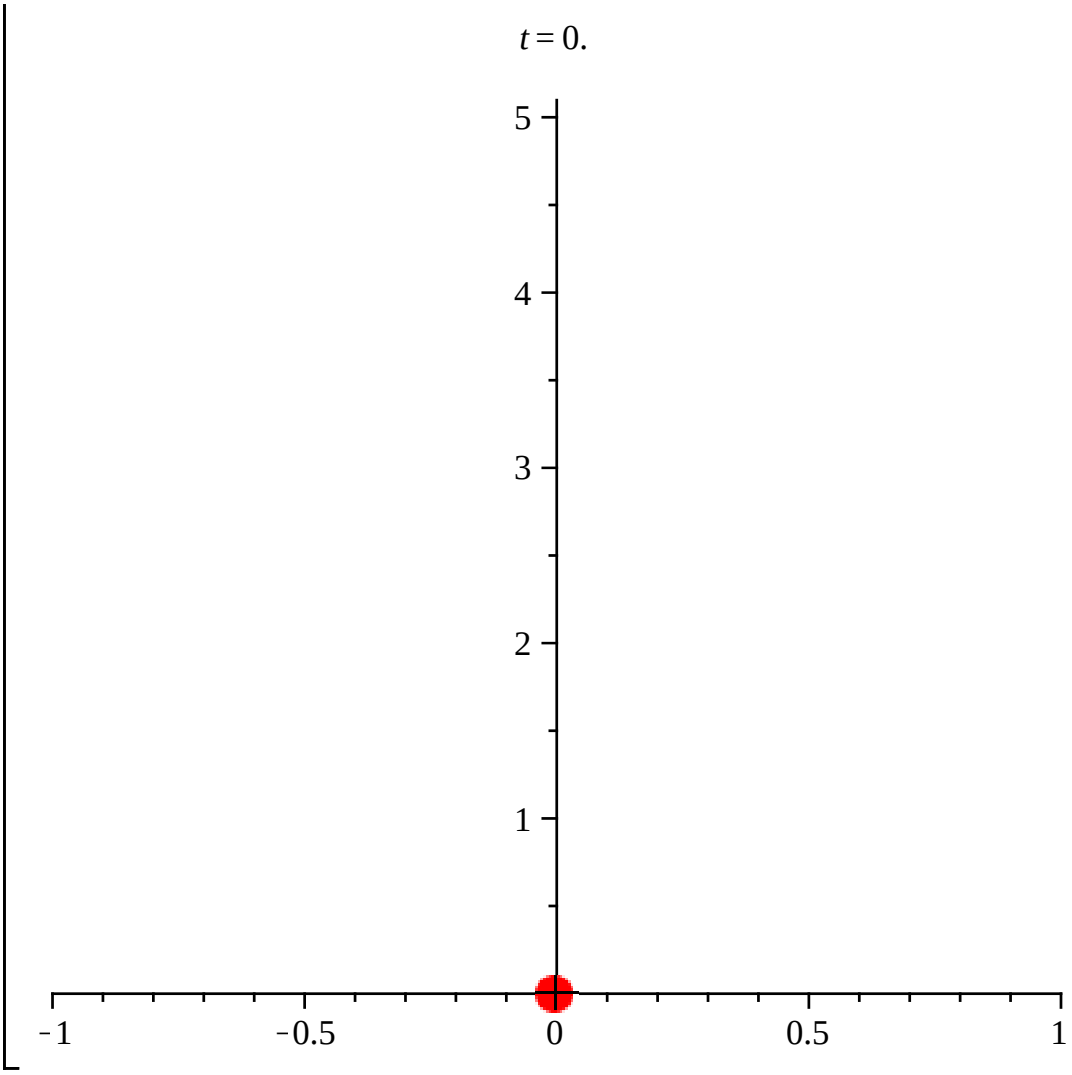
式 **(2.3.1.1)** の 2 つめの値を endTime として定義します．

sol1[2] により，2 つめの値を sol1 から抽出することができます．

再度，endTime := sol1[2] として，アニメーションを作成します．

```
> endTime := sol1[2];
  animate(pointplot, [[0, S10], symbol=solidcircle, symbolsize = 30,
  color = red], t=0..endTime);
```

$$endTime := 2.039359641$$



初速度 v0 = 20 の場合

２次方程式の２根を，一度で（１行で）別々の変数に割り当てます．

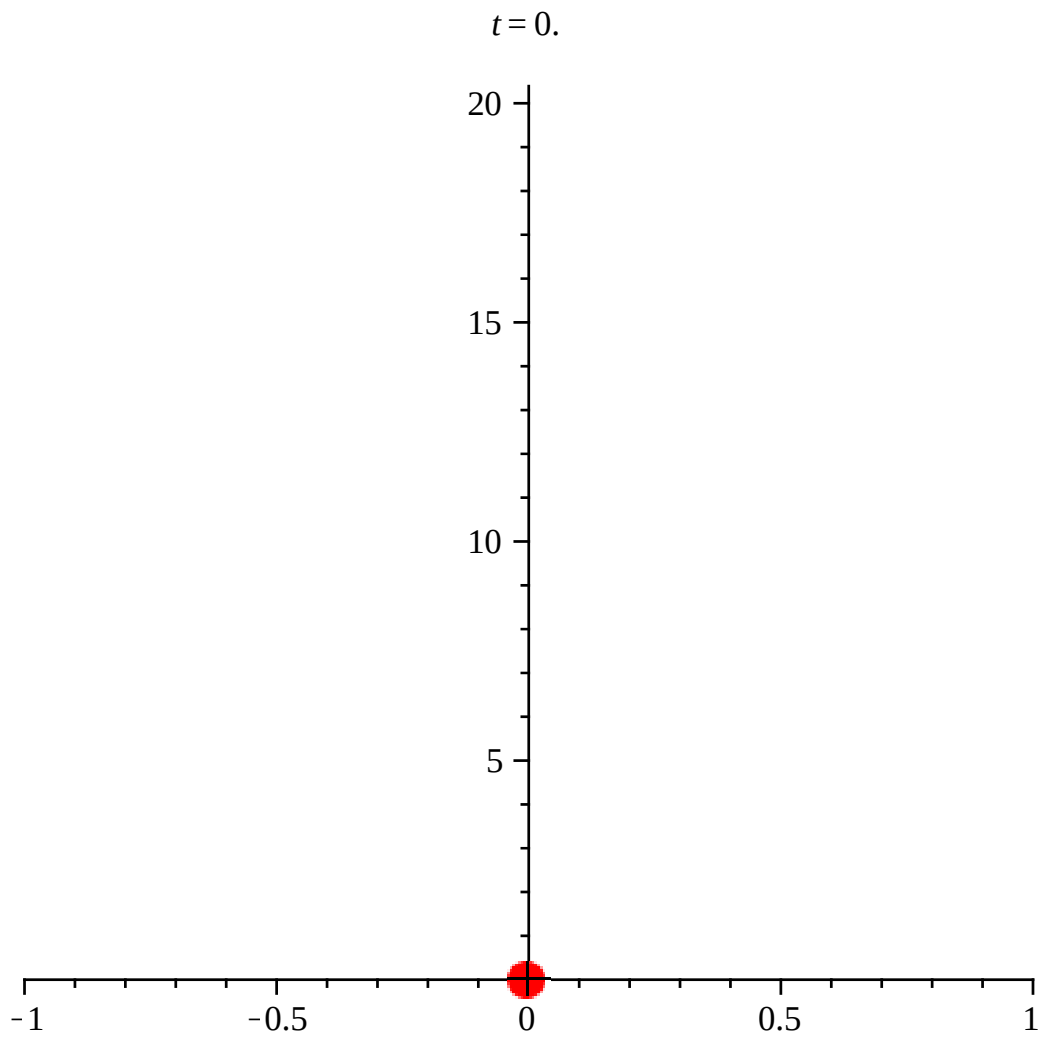
```
> ( sol2[1], sol2[2] ) := solve(S20, t);
```

$$sol2_1, sol2_2 := 0., 4.078719282 \qquad \textbf{(2.3.2.1)}$$

endTime := sol2[2] として，アニメーションを作成します．

```
> endTime := sol2[2];
  animate(pointplot, [[0, S20], symbol=solidcircle, symbolsize = 30,
  color = red], t=0..endTime);
```

$$endTime := 4.078719282$$



初速度 v0 = 30 の場合

コマンドの結果から直接２つの根を取り出せます.
solve(S30, t) の後ろに，[1] または [2] を指定します.

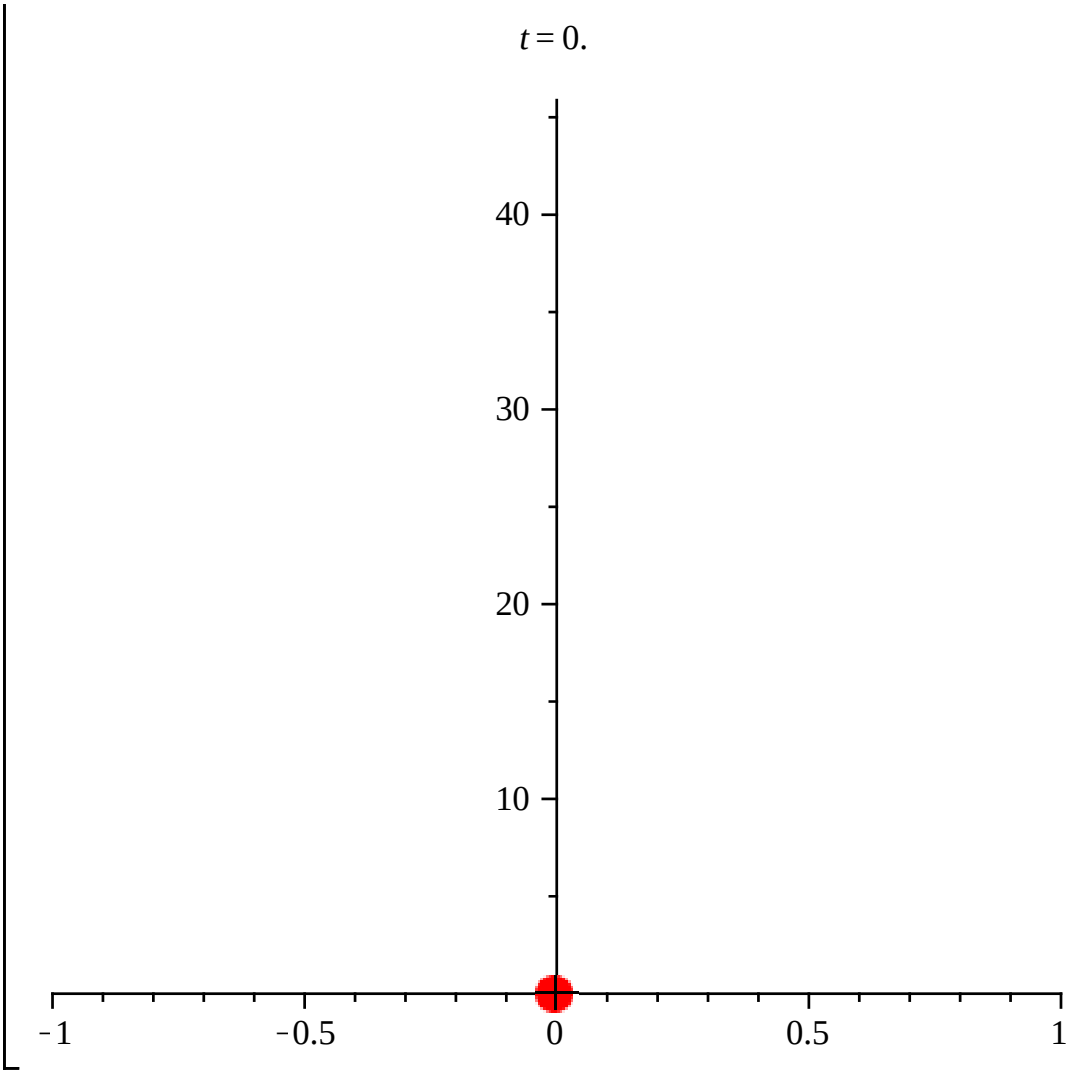
```
> solve(S30, t)[1];
  solve(S30, t)[2];
```

$$0.$$

$$6.118078923 \qquad \textbf{(2.3.3.1)}$$

```
> endTime := solve(S30, t)[2];
  animate(pointplot, [[0, S30], symbol=solidcircle, symbolsize = 30,
  color = red], t=0..endTime);
```

$$endTime := 6.118078923$$



同時アニメーションの作成

3点をひとつのアニメーションにまとめます.

```
> sPoints := [[0, S10], [0, S20], [0, S30]];
```

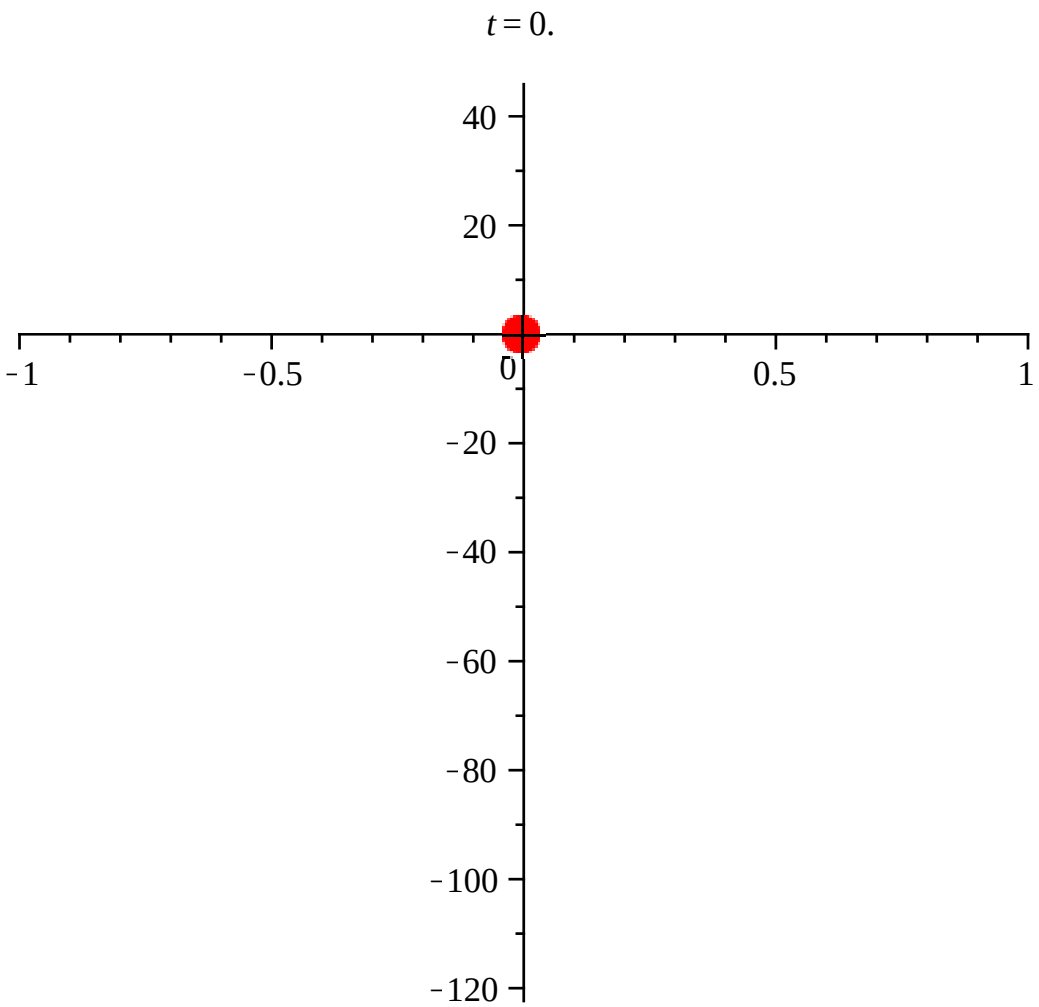
$$sPoints := [[0, -4.903500000\,t^2 + 10\,t], [0, -4.903500000\,t^2 + 20\,t], [0, -4.903500000\,t^2 + 30\,t]] \quad \textbf{(2.3.4.1)}$$

初速度30［m/s］がグラウンドに到達したときにアニメーションを止めるようにします．すなわち，solve(S30, t) の2番目の根 [2] を使用します.

```
> endTime := solve(S30, t)[2];
```

$$endTime := 6.118078923 \quad \textbf{(2.3.4.2)}$$

```
> animate(pointplot, [sPoints, symbol=solidcircle, symbolsize = 30,
  color = red], t=0..endTime);
```

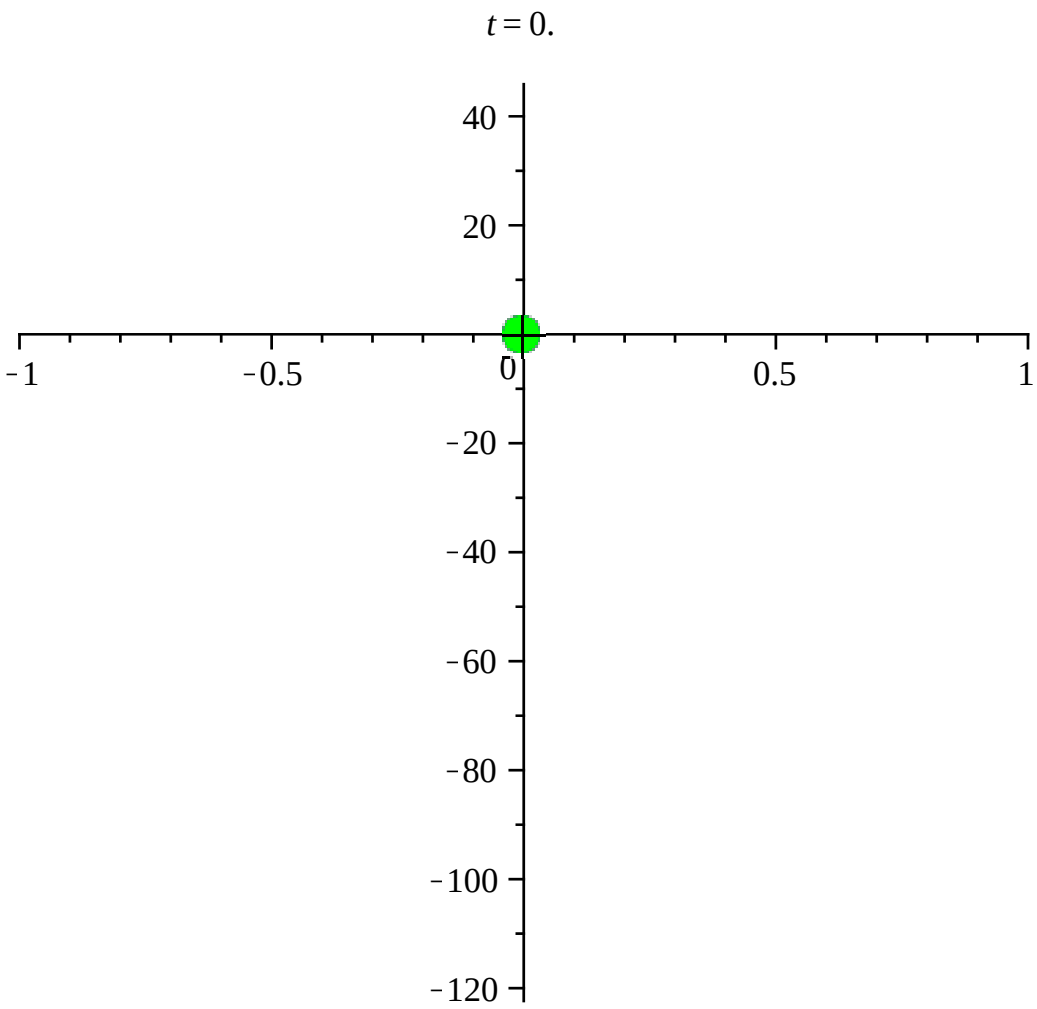


点の色を分けます.

初速度 v0 = 10 の場合： ● (赤)
初速度 v0 = 20 の場合： ● (青)
初速度 v0 = 30 の場合： ● (緑)

color = [red, blue, green] のようにオプションを指定します.

```
> animate(pointplot, [sPoints, symbol=solidcircle, symbolsize = 30,
  color = [red, blue, green]], t=0..endTime);
```



位置を横にずらします（座標で点の位置を指定します）.

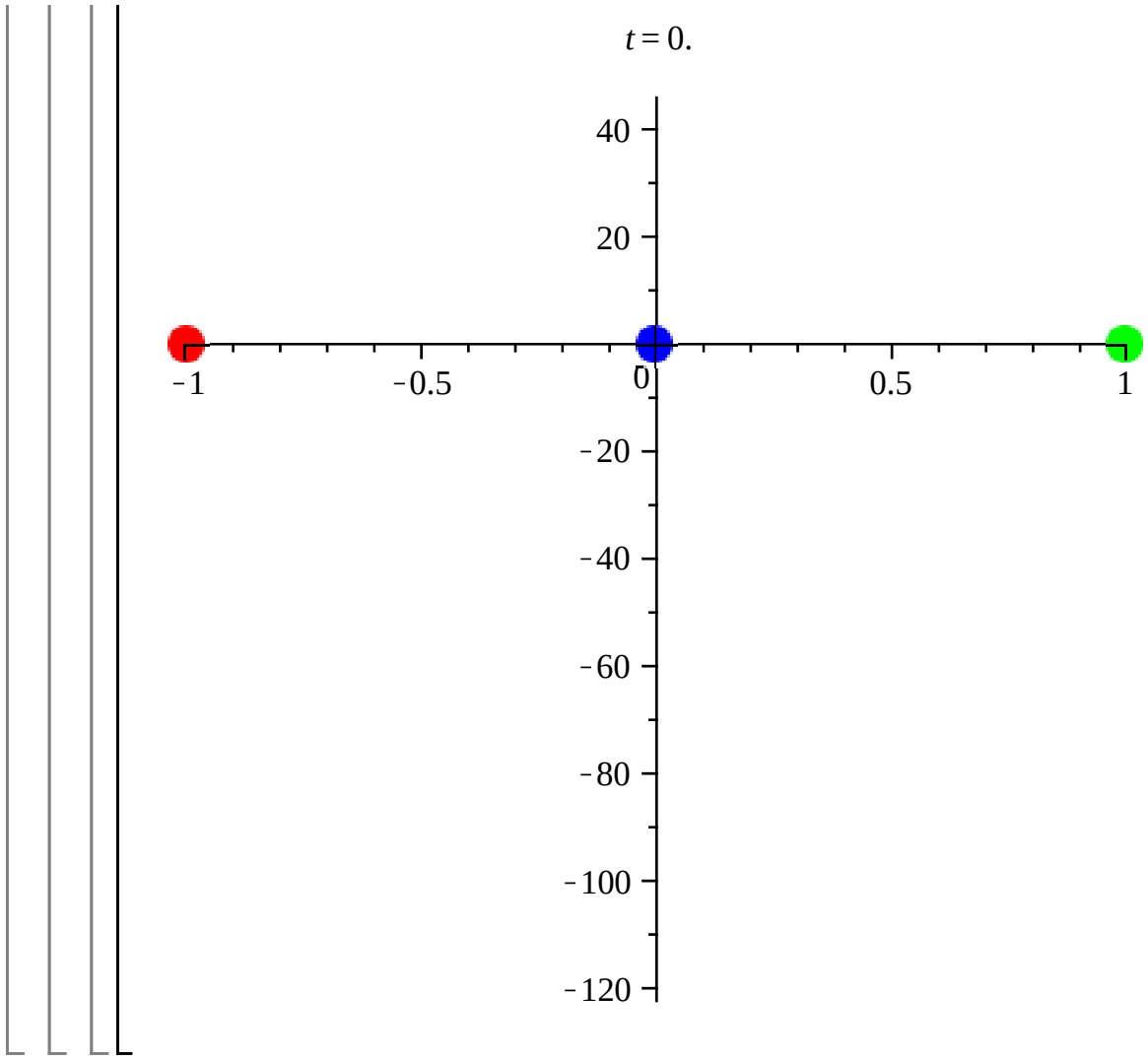
sPoints := [[-1, S10], [0, S20], [1, S30]]; を

sPoints := [[0, S10], [0, S20], [0, S30]]; のように変更します.

```
> sPoints := [[-1, S10], [0, S20], [1, S30]];
```

$$sPoints := [[-1, -4.903500000\,t^2 + 10\,t], [0, -4.903500000\,t^2 + 20\,t], [1, -4.903500000\,t^2 + 30\,t]] \quad \textbf{(2.3.4.3)}$$

```
> animate(pointplot, [sPoints, symbol=solidcircle, symbolsize = 30,
  color = [red, blue, green]], t=0..endTime);
```


t = 0.
40
20
-1
-0.5
0
0.5
1
-20
-40
-60
-80
-100
-120

ついでに斜めにも投げてみる（斜め投げ上げ運動）

斜め投げ上げ運動（斜方投射）を確認します．

水平方向に対し θ の方向に初速度 v0 で投げ上げる運動になります．x（水平），y（垂直）方向の運動を別々に考えます．位置 y は，上向きを正とします．

はじめに，初期化します．

```
> restart:
```

次に，運動の式を定義します．

x（水平） 方向

```
> x := v0 * cos(theta) * t;
```

$$x := v0 \cos(\theta)\, t \tag{3.1}$$

y（垂直） 方向　（t^2の項の符号に注意）

```
> y := v0 * sin(theta) * t - ( g*t^2 )/2;
```

$$y := v0 \sin(\theta)\, t - \frac{g\, t^2}{2} \tag{3.2}$$

初速度 v0 =10 [m/s] で角度 θ= Pi/4 [rad] で投げ上げたとします（重力加速度 g = 9.807 ）．ここで，パラメータを定義します．

```
> params := [v0=10, theta = Pi/4, g=9.807];
```

$$params := \left[v0 = 10, \theta = \frac{\pi}{4}, g = 9.807\right] \tag{3.3}$$

値を式に代入します．

x（水平）方向

```
> X1 := eval( x, params );
```

$$X1 := 5\sqrt{2}\, t \tag{3.4}$$

y（垂直）方向

```
> Y1 := eval( y, params );
```

$$Y1 := 5\sqrt{2}\,t - 4.903500000\,t^2 \quad \textbf{(3.5)}$$

ここで，アニメーションを作成します．

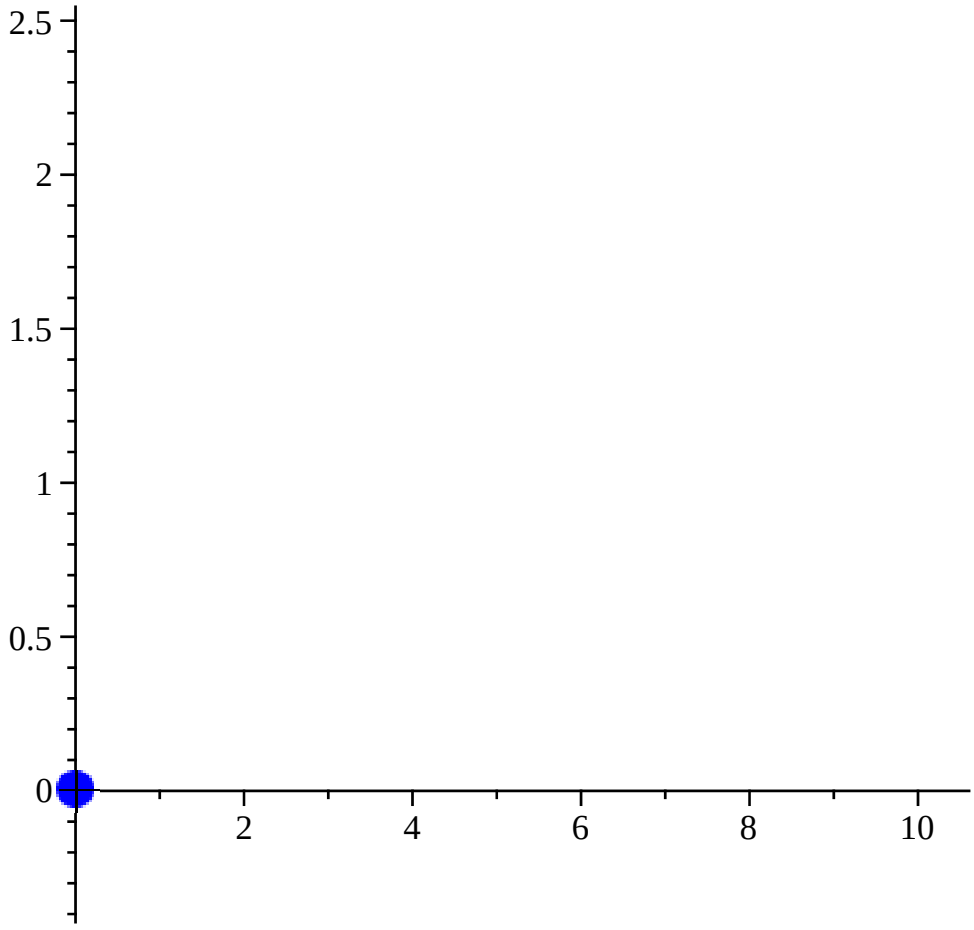
plots パッケージをロードします．

```
> with(plots):
```

animate コマンドと pointplot コマンドを使用してアニメーションを作成します．

```
> animate(pointplot, [ [X1, Y1], symbol=solidcircle, symbolsize=30, color=
  blue], t=0..1.5 );
```

$t = 0.$



ステップ･バイ･ステップ式 **はじめての Maple** 基礎編

ワークシートを操る！

ワークシートの操作方法の基礎を習得します.

目次

ツールバー（Windows 及び Mac OS 共通）

ファイル操作と編集

左のボタンから順に以下のような機能が割り当てられています.

- ファイルの新規作成（デフォルトのファイルモード設定によります）
- ファイルを開く
- 現在開いているファイルを保存する
- 現在開いているファイルを印刷する
- ファイル印刷前にプレビューする
- 選択した部分を切り取る
- 選択した部分をクリップボードにコピーする
- クリップボードの内容を現在のワークシートに貼り付ける
- 直前の操作を元に戻す（アンドゥ）
- 直前の操作を取り消す

グループの入力とセクション

左のボタンから順に以下のような機能が割り当てられています.

- 現在の実行グループの次にテキストを挿入する
- 現在の実行グループの次に新しい実行グループを挿入する
- 選択した部分をひとつのセクション（章立て）にする
- 章立て部分を解除する
- ハイパーリンクの履歴を戻る
- ハイパーリンクをひとつ進む

計算の実行と中断，デバッグなど

左のボタンから順に以下のような機能が割り当てられています．

- ワークシート中のコマンドを最初からすべて実行する
- 現在カーソルのある実行グループのコマンドを実行する
- 実行中の計算を中断する
- 現在の処理をデバッグする
- 現在の計算カーネルを初期化する
- スタートアップコードを編集する
- 100%ズームで表示
- 150%ズームで表示
- 200%ズームで表示

その他

左のボタンから順に以下のような機能が割り当てられています．

- テキストグループでのタブキーの挙動の変更
- ヘルプを起動する

ワークシート向けのショートカット（Windows）

ワークシート全体に影響のあるショートカットになります.

機能・作業	操作
ワークスペースの拡大縮小（50%）	Ctrl + 0
ワークスペースの拡大縮小（75%）	Ctrl + 1
ワークスペースの拡大縮小（100%）	Ctrl + 2
ワークスペースの拡大縮小（125%）	Ctrl + 3
ワークスペースの拡大縮小（150%）	Ctrl + 4
ワークスペースの拡大縮小（200%）	Ctrl + 5
ワークスペースの拡大縮小（300%）	Ctrl + 6
ワークスペースの拡大縮小（400%）	Ctrl + 7
グループのレンジ（範囲）の表示切り替え	F9
セクションのレンジ（範囲）の表示切り替え	Shift + F9
スペルチェック	F7

入力とグループ向けのショートカット（Windows）

コマンド入力やグループの操作に使用するショートカットになります．ただし，(W) は，Window のアプリケーション（ソフトウェア）において，一般的なショートカットになります．

機能・作業	操作
コマンドの実行	Enter
コマンド内の改行	Shift + Enter
行先頭へ移動	Home
行末尾へ移動	End
グループをカーソルの前に追加	Ctrl + K
グループをカーソルの後に追加	Ctrl + J
パラグラフをカーソルの前に追加	Ctrl + Shift + K
パラグラフをカーソルの後に追加	Ctrl + Shift + J
カーソルある行の削除	Ctrl + Delete
テキスト入力と数式入力の切り替え	F5
カーソルの位置で実行グループの分割	F3
カーソルの位置で実行グループの結合	F4
テキスト入力への変更（プロンプトの表示）	Ctrl + M
数式入力への変更	Ctrl + R
テキストの入力	Ctrl + T
コマンド補間	Esc
コマンドヘルプの起動	コマンドにカーソルを置いて F2
ギリシャ文字の入力	Ctrl + Shift + G（例えば，$a > \alpha$）
(W) コピー	Ctrl + C
(W) 貼り付け（ペースト）	Ctrl + V
(W) 切り取り（カット）	Ctrl + X
(W) アンドゥー（作業のキャンセル）	Ctrl + Z
(W) 上書き保存	Ctrl + S
(W) すべて選択	Ctrl + A

セクションとサブセクション向けのショートカット（Windows）

セクション（サブセクション）の追加・削除に使用するショートカットになります.

機能・作業	操作
セクション（サブセクション）の追加（挿入）	Ctrl + >
セクション（サブセクション）の削除	Ctrl + <

典型的なセクション内の構成

（文章）

（グループ）

（文章）

（パラグラフ－文章）

（パラグラフ－文章）

（パラグラフ－文章）

（パラグラフ－文章）

セクション－サブセクション－サブサブセクション

以下は，典型的な論文の構成例になります．

タイトル

セクション

サブセクション

サブサブセクション

セクション

サブセクション

サブサブセクション

セクション

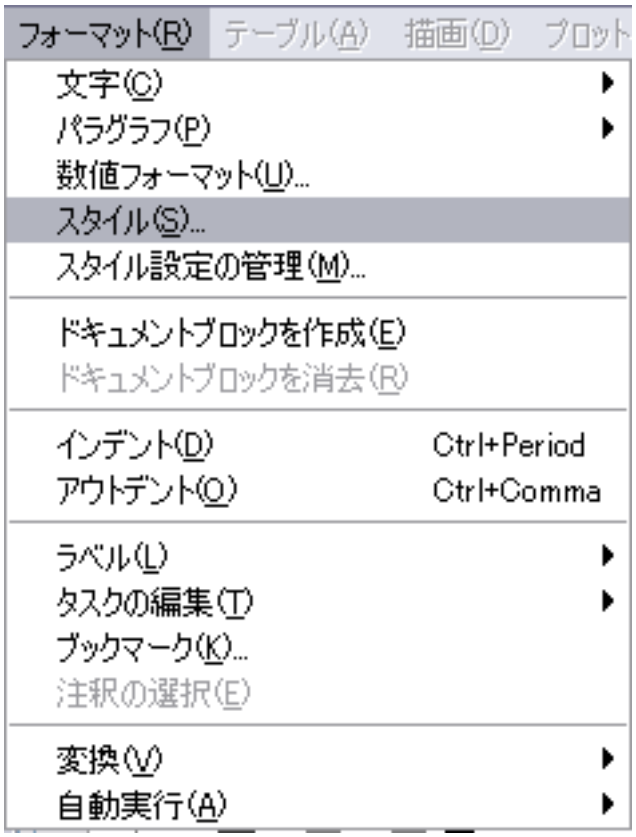
サブセクション

サブサブセクション

スタイルの設定

変更手順

1. **［フォーマット］メニューから［スタイル］を選択します.**



［スタイル管理］ウィンドウが開きます.

2. ここで変更したい（C）文字スタイルもしくは（P）パラグラフスタイルをクリックします.

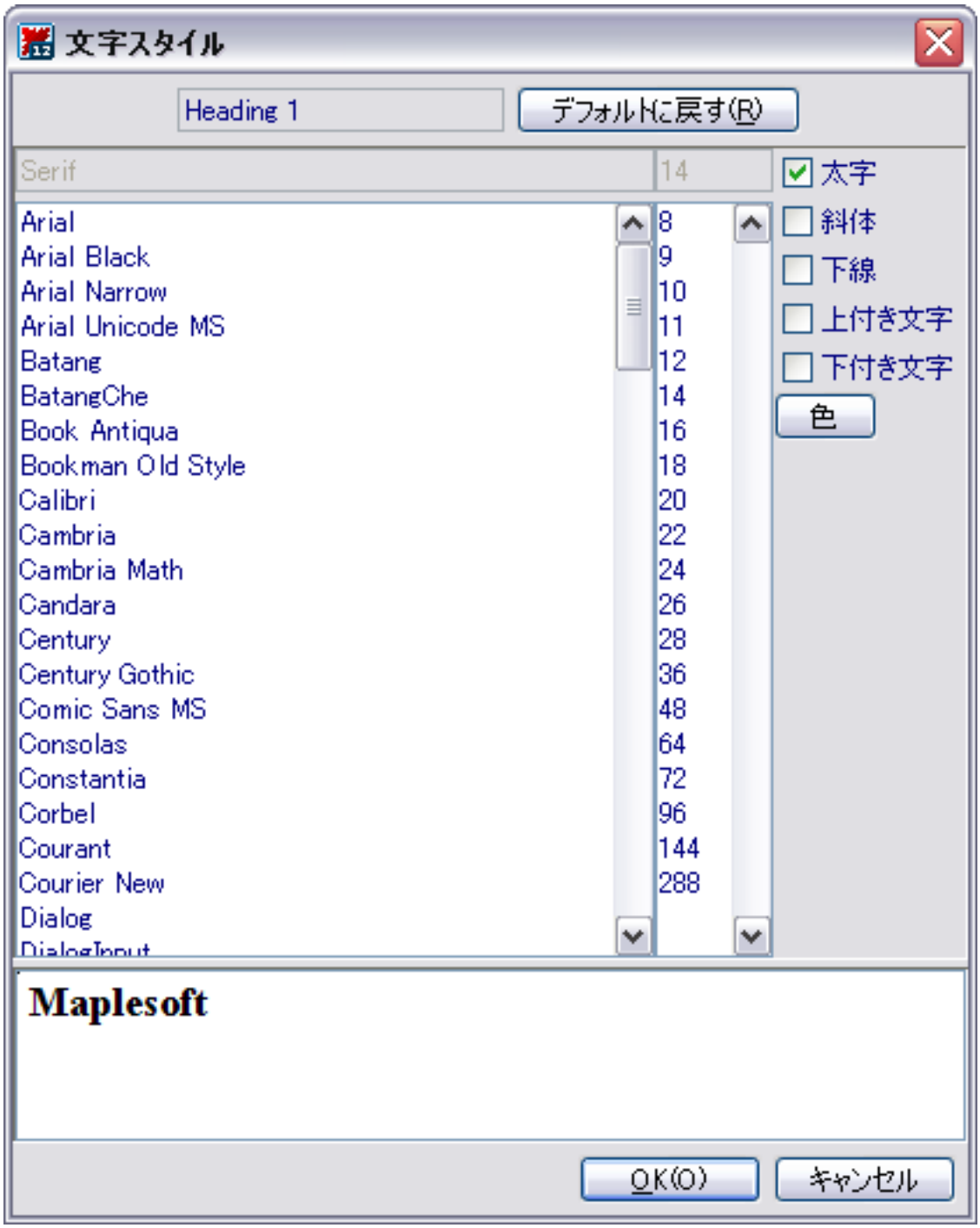
例えば，［Heading 1］（見出し 1 ＝セクションに相当します）をクリックします.

［Heading 1］の［パラグラフスタイル］ウィンドウが開きます.

3. **ここで，［フォント］ボタンをクリックします.**

［パラグラフスタイル］から［文字スタイル］変更ウィンドウが開きます.

4. **ここから，［Heading 1］（見出し１）に関するフォントの種類，サイズ，スタイル（太文字や斜体など）あるいは色を設定します.**



5. **設定完了後，すべてのウィンドウにおいて，［OK］ボタンをクリックすると，設定がすぐに反映されます．**

設定するスタイル

P : パラグラフスタイル
C : 文字スタイル（キャラクタスタイル）

(**P**) **Title**（タイトル）

サイズ： 18pt

(**P**) **Author**（著者）

サイズ： 16pt

(**P**) **Heading 1**（見出し 1 ＝セクション）

種類： MS Pゴシック
サイズ： 12pt
スタイル： なし（チェックをはずす）

(**P**) **Heading 2**（見出し 2 ＝サブセクション）

種類： MS Pゴシック
サイズ： 12pt
スタイル： なし（チェックをはずす）

(**P**) **Heading 3**（見出し 3 ＝サブサブセクション）

種類： MS Pゴシック

サイズ： 10pt
スタイル： なし（チェックをはずす）

(**C**) **Text**（文章＝テキスト部分）

種類： MS P明朝
サイズ： 10pt
スタイル： なし（チェックをはずす）

(**C**) **Header and Footer**（ヘッダー・フッター）

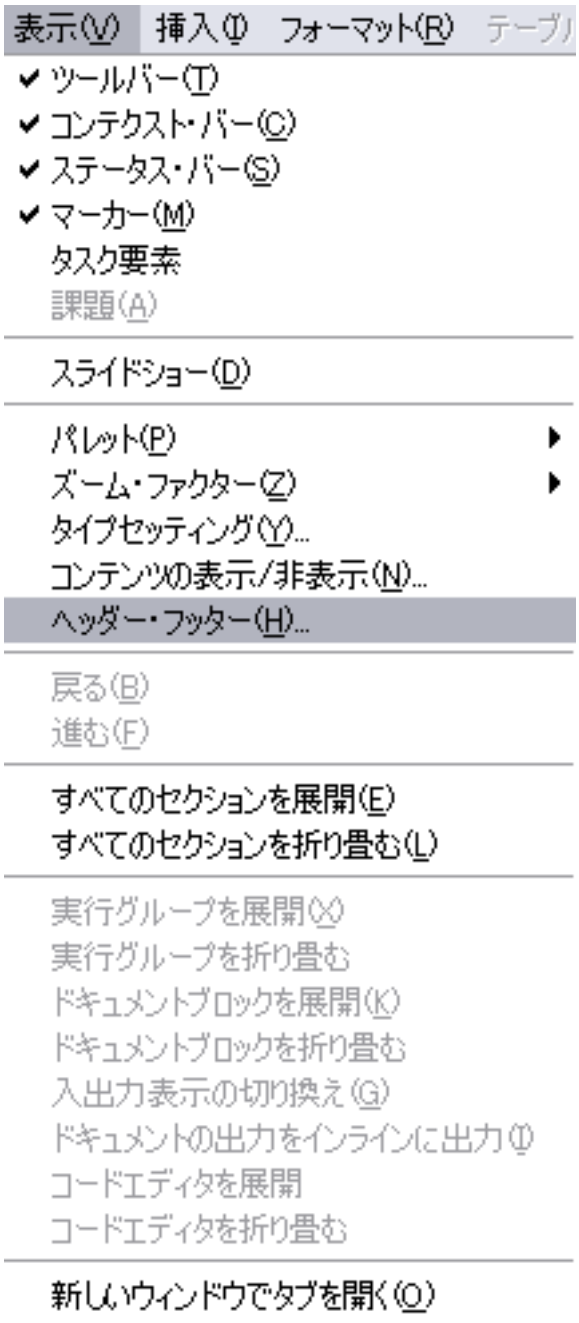
種類： MS Pゴシック
サイズ： 10pt
スタイル： なし

(**C**) **Maple Input**（テキスト入力）

種類： Courier New
サイズ： 10pt
スタイル： なし
色： 赤

ヘッダとフッタ

［表示］メニューから［ヘッダー・フッター］を選択します.



［標準のヘッダー・フッター］タブから. ［Header］および［Footer］を［カスタム］に変更します.

［カスタムヘッダー］タブをクリックします．任意の文字あるいはページ数などを設定します．

［カスタムフッター］タブをクリックします．任意の文字あるいはページ数などを設定します．

ステップ･バイ･ステップ式 **はじめての Maple** 基礎編

Maple の仲間たち

Maple で定義あるいは作成できる様々なオブジェクトを紹介します.

基礎編

目次

Maple の予約語

数学定数

初期化します.

```
> restart;
```

Maple で使用される数学定数を表示します.

```
> constants;
```

$$false, \gamma, \infty, true, Catalan, FAIL, \pi \quad (1.1.1)$$

円周率 π

```
> Pi;
```

$$\pi \quad (1.1.2)$$

Pi の浮動小数点化します（近似値を求めます）.

```
> evalf(Pi);
```

$$3.141592654 \quad (1.1.3)$$

小数点以下 100 桁にします.

```
> Digits := 101;
```

$$Digits := 101 \quad (1.1.4)$$

```
> evalf(Pi);
```

$$3.14159265358979323846264338327950288419716939937510582097494459230781\backslash 64062862089986280348253421170680 \quad (1.1.5)$$

オイラーの定数 γ （オイラー・マスケローニ定数，オイラーの γ）

```
> evalf(gamma);
```

$$0.57721566490153286060651209008240243104215933593992359880576723488486\backslash 772677766467093694706329174674951 \quad (1.1.6)$$

カタランの定数 G

```
> G := evalf(Catalan);
```

$$G := 0.91596559417721901505460351493238411077414937428167213426649811\backslash 962176301977625476947935651292611510625 \quad (1.1.7)$$

グローバル変数（大域的変数）

グローバル変数は，すべてのスコープ（変数の参照範囲）から参照可能な変数になります．詳細は，プログラミングの項で説明します．

虚数 i（もしくは j）

（参考）電気・電子工学などでは，電流 i と区別するために，虚数に j を使用します．

```
> I;
```

$$I \tag{1.2.1}$$

無限大 ∞

```
> infinity;
```

$$\infty \tag{1.2.2}$$

ブール論理（真・偽）

```
> true;
```

$$true \tag{1.2.3}$$

```
> false;
```

$$false \tag{1.2.4}$$

円周率 π とギリシャ文字 π

円周率 π

```
> Pi;
```

$$\pi \tag{1.3.1}$$

ギリシャ文字 π

```
> pi;
```

$$\pi \tag{1.3.2}$$

円周率 π を数値化します．

```
> evalf(Pi);
```

$$3.14159265358979323846264338327950288419716939937510582097494459230781\backslash 640628620899862803482534211706800 \tag{1.3.3}$$

ギリシャ文字 π を評価してみます．

```
> eval(pi);
```

(1.3.4)

$$\pi \qquad \textbf{(1.3.4)}$$

大文字で始まる Pi は数学定数としての円周率になります．小文字で始まる pi は文字として認識されます．

複素数

複素数の定義と四則演算（１）

初期化します．

```
> restart;
```

以下のような複素数を定義します．ただし，Iは虚数，a、b、c、dは実数．

```
> z1 := a + I*b;
  z2 := c + I*d;
```

$$z1 := a + \mathrm{I}\,b$$
$$z2 := c + \mathrm{I}\,d \qquad (2.1.1)$$

足し算

```
> z1 + z2;
```

$$a + \mathrm{I}\,b + c + \mathrm{I}\,d \qquad (2.1.2)$$

複素数の評価には，evalc を使用します（evalc の c は complex の c です）．

```
> evalc( z1 + z2 );
```

$$a + c + \mathrm{I}\,(b + d) \qquad (2.1.3)$$

引き算

```
> z1 - z2;
```

$$a + \mathrm{I}\,b - c - \mathrm{I}\,d \qquad (2.1.4)$$

```
> evalc( z1 - z2 );
```

$$a - c + \mathrm{I}\,(b - d) \qquad (2.1.5)$$

掛け算

```
> z1*z2;
```

$$(a + \mathrm{I}\,b)\,(c + \mathrm{I}\,d) \qquad (2.1.6)$$

```
> evalc( z1*z2 );
```

$$a\,c - b\,d + \mathrm{I}\,(a\,d + b\,c) \qquad (2.1.7)$$

割り算

```
> z1/z2;
```

$$\frac{a + \mathrm{I}\,b}{c + \mathrm{I}\,d} \qquad (2.1.8)$$

```
> evalc( z1/z2 );
```

$$\frac{a\,c}{c^2 + d^2} + \frac{b\,d}{c^2 + d^2} + \mathrm{I}\left(\frac{b\,c}{c^2 + d^2} - \frac{a\,d}{c^2 + d^2}\right) \qquad (2.1.9)$$

複素数の定義と四則演算（２）

Complex コマンドによって，同様に複素数を定義できます．

```
> z3 := Complex(a, b);
```

$$z3 := Complex(a, b) \quad \textbf{(2.2.1)}$$

evalc で評価します．

```
> z3 := evalc(z3);
```

$$z3 := a + \mathrm{I}\, b \quad \textbf{(2.2.2)}$$

2 - 3*I を定義します．

```
> z4 := Complex(2, -3);
```

$$z4 := 2 - 3\,\mathrm{I} \quad \textbf{(2.2.3)}$$

z3 と z4 の四則演算を evalc で確認します．

```
> evalc( z3 + z4 );
```

$$a + 2 + \mathrm{I}\,(b - 3) \quad \textbf{(2.2.4)}$$

```
> evalc( z3 - z4 );
```

$$a - 2 + \mathrm{I}\,(b + 3) \quad \textbf{(2.2.5)}$$

```
> evalc( z3 * z4 );
```

$$2\,a + 3\,b + \mathrm{I}\,(-3\,a + 2\,b) \quad \textbf{(2.2.6)}$$

```
> evalc( z3 / z4 );
```

$$\frac{2\,a}{13} - \frac{3\,b}{13} + \mathrm{I}\left(\frac{3\,a}{13} + \frac{2\,b}{13}\right) \quad \textbf{(2.2.7)}$$

複素平面へのプロット

初期化します．

```
> restart;
```

複素平面へのプロットコマンド complexplot を読み込みます．complexplot コマンドは，plots パッケージに含まれています．

plots パッケージをロードします．

```
> with(plots);
```

[*animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d, densityplot, display, dualaxisplot, fieldplot, fieldplot3d, gradplot, gradplot3d, graphplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams, intersectplot, listcontplot, listcontplot3d, listdensityplot,* **(2.3.1)**

listplot, listplot3d, loglogplot, logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot, rootlocus, semilogplot, setcolors, setoptions, setoptions3d, spacecurve, sparsematrixplot, surfdata, textplot, textplot3d, tubeplot]

複素数を定義します.

（参考）コマンド内の改行は，Shift キー + Enter キーになります.

```
> z1 := 1 + 2*I;
  z2 := 3 + 4*I;
  z3 := 5 - 1*I;
  z4 := 7 - 8*I;
```

$$z1 := 1 + 2\,I$$
$$z2 := 3 + 4\,I$$
$$z3 := 5 - I$$
$$z4 := 7 - 8\,I \qquad \textbf{(2.3.2)}$$

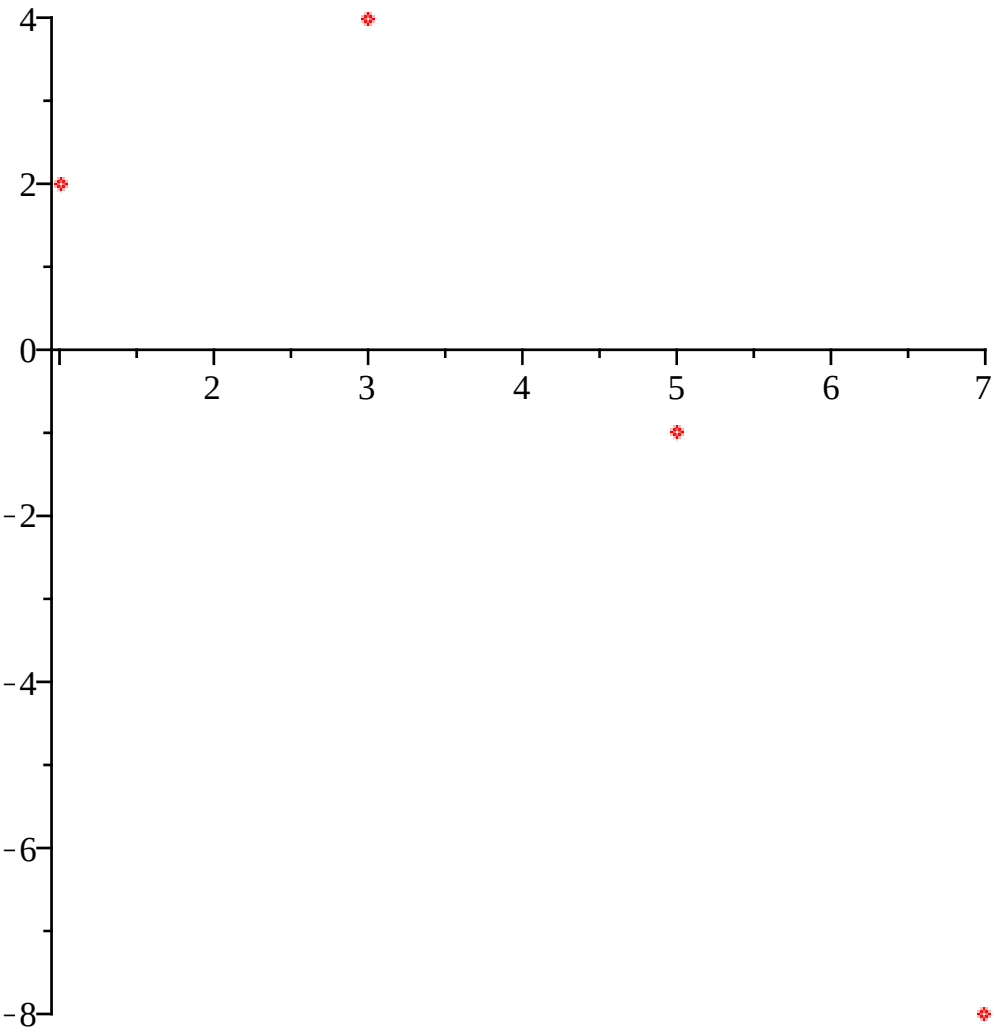
角括弧 [ ] は，値や式（要素）を並べて，ひとつの処理のかたまりを定義します（通常，リストと呼びます）.

```
> clist := [z1, z2, z3, z4];
```

$$clist := [1 + 2\,I, 3 + 4\,I, 5 - I, 7 - 8\,I] \qquad \textbf{(2.3.3)}$$

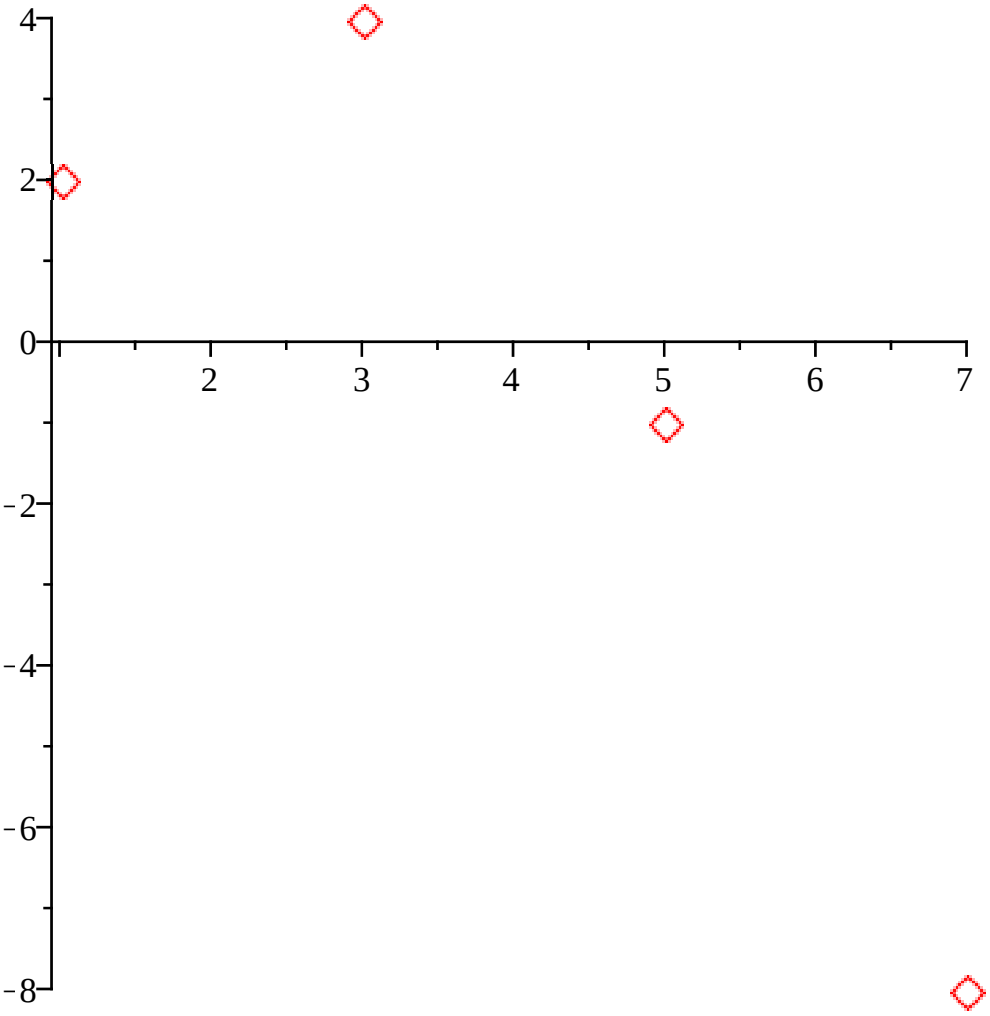
複素平面に clist をプロットします．style=point は，点としてプロットします.

```
> complexplot( clist, style=point );
```



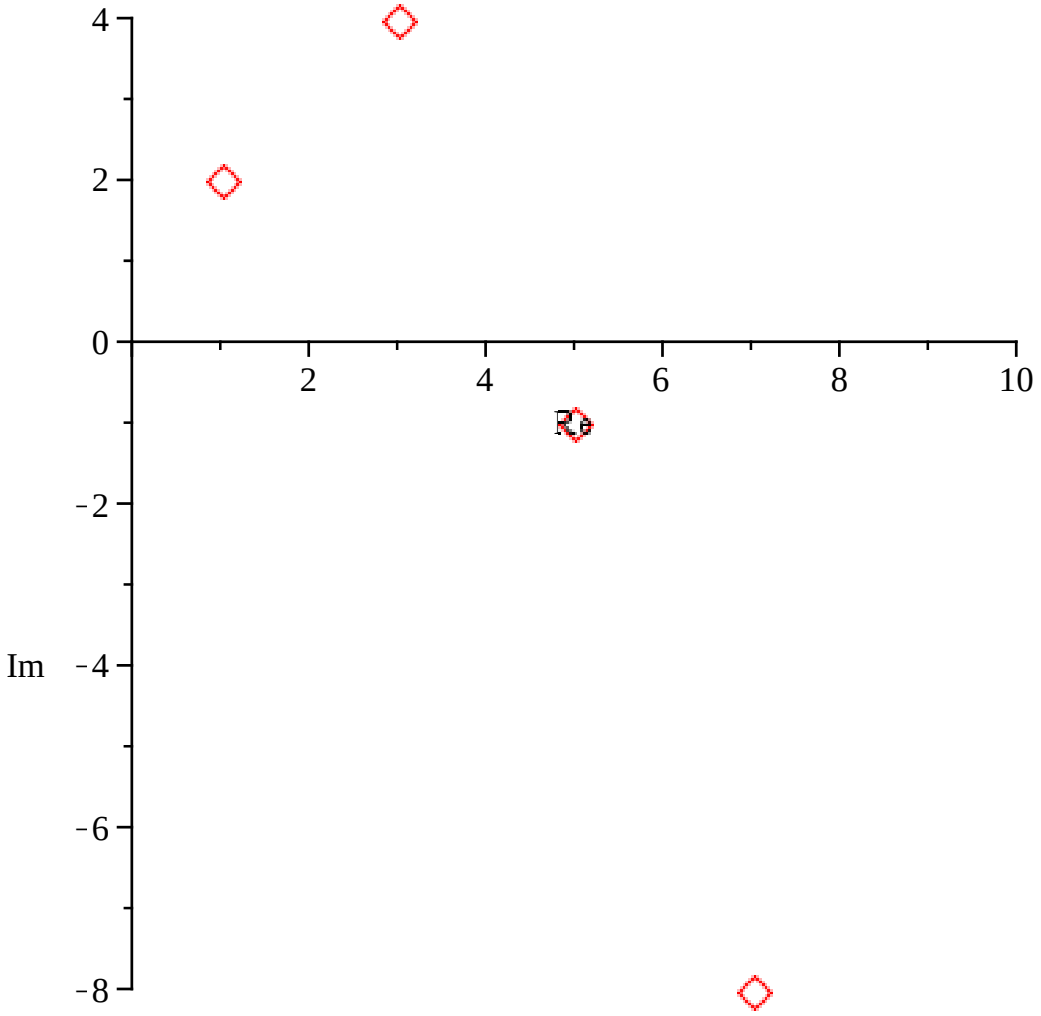
ポイントのサイズを変更します.

```
> complexplot( clist, style=point, symbolsize=24 );
```



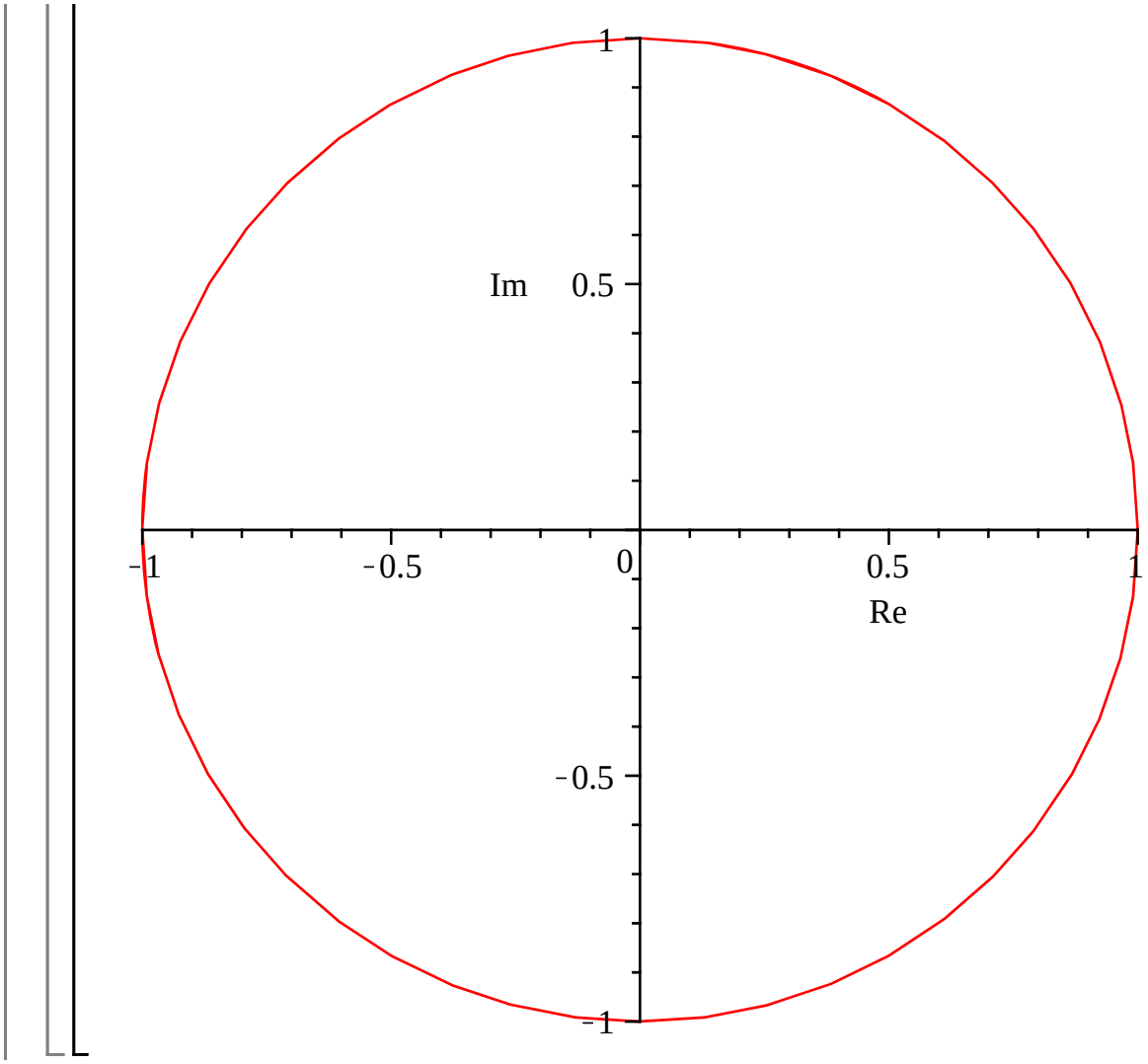
実部の範囲を 0..10 でプロットし直します．また，ラベル labels=["Re", "Im"] を追加します．

```
> complexplot( clist, 0..10, style=point, symbolsize=24, labels=["Re",
  "Im"] );
```



cos + I*sin （オイラーの公式） を，プロットしてみます.

```
> complexplot( cos + I*sin, 0..2*Pi, labels=["Re", "Im"] );
```


1
Im
0.5
-1
-0.5
0
0.5
1
Re
-0.5
-1

式の性質を確認

オイラーの公式

$$e^{I\theta} = \cos(\theta) + I\sin(\theta)$$
$$e^{-I\theta} = \cos(\theta) - I\sin(\theta)$$

の右辺を変数 treq1，treq2 にそれぞれ代入します．

```
> treq1 := cos(theta) + I*sin(theta);
  treq2 := cos(theta) - I*sin(theta);
```

$$treq1 := \cos(\theta) + I\sin(\theta)$$
$$treq2 := \cos(\theta) - I\sin(\theta) \quad \textbf{(2.4.1)}$$

式 treq1，treq2 を指数関数に変換し，exeq1，exeq2 にそれぞれ代入します．

```
> exeq1 := convert(treq1, exp);
  exeq2 := convert(treq2, exp);
```

$$exeq1 := e^{I\theta}$$
$$exeq2 := e^{-I\theta} \quad \textbf{(2.4.2)}$$

以上のように，Maple は，オイラーの公式を用いて式を変換します．

（参考）三角関数の指数関数表記

ここで，式 exeq1 と exeq2 を足し合わせたあと三角関数に変換します．

```
> exeq1 + exeq2;
```

$$e^{I\theta} + e^{-I\theta} \quad \textbf{(2.4.1.1)}$$

```
> convert((2.4.1.1), trig);
```

$$2\cos(\theta) \quad \textbf{(2.4.1.2)}$$

式 $e^{I\theta} + e^{-I\theta}$ と式 $2\cos(\theta)$ は等価なことから以下の式 eq1 が定義されます．

```
> eq1 := (2.4.1.1) = (2.4.1.2);
```

$$eq1 := e^{I\theta} + e^{-I\theta} = 2\cos(\theta) \quad \textbf{(2.4.1.3)}$$

式 $eq1 := e^{I\theta} + e^{-I\theta} = 2\cos(\theta)$ を cos(θ) について整理します．

```
> isolate((2.4.1.3), cos(theta));
```

$$\cos(\theta) = \frac{e^{I\theta}}{2} + \frac{e^{-I\theta}}{2} \quad \textbf{(2.4.1.4)}$$

次に，exeq1 から exeq2 を引いたあと三角関数に変換します．

```
> exeq1 - exeq2;
```

$$e^{I\theta} - e^{-I\theta} \qquad \textbf{(2.4.1.5)}$$

```
> convert((2.4.1.5), trig);
```

$$2\,I\sin(\theta) \qquad \textbf{(2.4.1.6)}$$

式 $e^{I\theta} - e^{-I\theta}$ と式 $2\,I\sin(\theta)$ は等価なことから以下の式 eq2 が定義されます.

```
> eq2 := (2.4.1.5) = (2.4.1.6);
```

$$eq2 := e^{I\theta} - e^{-I\theta} = 2\,I\sin(\theta) \qquad \textbf{(2.4.1.7)}$$

式 $eq2 := e^{I\theta} - e^{-I\theta} = 2\,I\sin(\theta)$ を sin(θ) について整理します.

```
> isolate((2.4.1.7), sin(theta));
```

$$\sin(\theta) = \frac{I}{2}\left(-e^{I\theta} + e^{-I\theta}\right) \qquad \textbf{(2.4.1.8)}$$

以上から，以下の関係が導かれます.

$$\cos(\theta) = \frac{e^{I\theta}}{2} + \frac{e^{-I\theta}}{2}$$

$$\sin(\theta) = \frac{I}{2}\left(-e^{I\theta} + e^{-I\theta}\right)$$

多項式

初期化します.

```
> restart;
```

1変数多項式の操作

1変数 x の多項式 p を定義します.

```
> p := x^3 + 5*x^2 + 11*x + 15;
```

$$p := x^3 + 5\,x^2 + 11\,x + 15 \qquad (3.1.1)$$

次数の確認

```
> degree(p, x);
```

$$3 \qquad (3.1.2)$$

指定次数の係数抽出

```
> coeff(p, x, 1);
```

$$11 \qquad (3.1.3)$$

すべての係数を抽出

```
> coeffs(p, x);
```

$$15,\ 11,\ 1,\ 5 \qquad (3.1.4)$$

値の代入

```
> subs( x=0, p );
```

$$15 \qquad (3.1.5)$$

値の代入・評価

```
> eval( p, x=0 );
```

$$15 \qquad (3.1.6)$$

（参考）subs と eval の比較

```
> subs( x=0, sin(x) );
  eval( sin(x), x=0 );
```

$$\sin(0)$$
$$0 \qquad (3.1.7)$$

被演算子（オペランド）の数（項の数）

```
> nops(p);
```

$$4 \qquad \textbf{(3.1.8)}$$

指定された次数の被演算子（項）

```
> op(2, p);
```

$$5\,x^2 \qquad \textbf{(3.1.9)}$$

すべての被演算子（項）を抽出

```
> op(p);
```

$$x^3,\ 5\,x^2,\ 11\,x,\ 15 \qquad \textbf{(3.1.10)}$$

因数分解

```
> factor(p);
```

$$(x+3)\left(x^2+2\,x+5\right) \qquad \textbf{(3.1.11)}$$

x で 1 階微分

```
> diff(p, x);
```

$$3\,x^2+10\,x+11 \qquad \textbf{(3.1.12)}$$

x について積分

```
> int(p, x);
```

$$\frac{1}{4}\,x^4+\frac{5}{3}\,x^3+\frac{11}{2}\,x^2+15\,x \qquad \textbf{(3.1.13)}$$

２変数の多項式を生成

２変数 x, y の多項式 q を定義します.

```
> q := (x + y + alpha)^2;
```

$$q := (x+y+\alpha)^2 \tag{3.2.1}$$

ランダムに２変数の多項式を生成

```
> r := randpoly([x,y], terms=6);
```

$$r := -62x + 97x^2 - 73y^2 - 4x^3y - 83x^2y^2 - 10y^4 \tag{3.2.2}$$

展開

```
> expand(q);
```

$$x^2 + 2xy + 2x\alpha + y^2 + 2y\alpha + \alpha^2 \tag{3.2.3}$$

同類項

```
> collect(q, x);
```

$$x^2 + (2y + 2\alpha)x + (y+\alpha)^2 \tag{3.2.4}$$

```
> collect(q, y);
```

$$y^2 + (2x + 2\alpha)y + (x+\alpha)^2 \tag{3.2.5}$$

項のソート（並べ替え）

```
> sort(r, x);
```

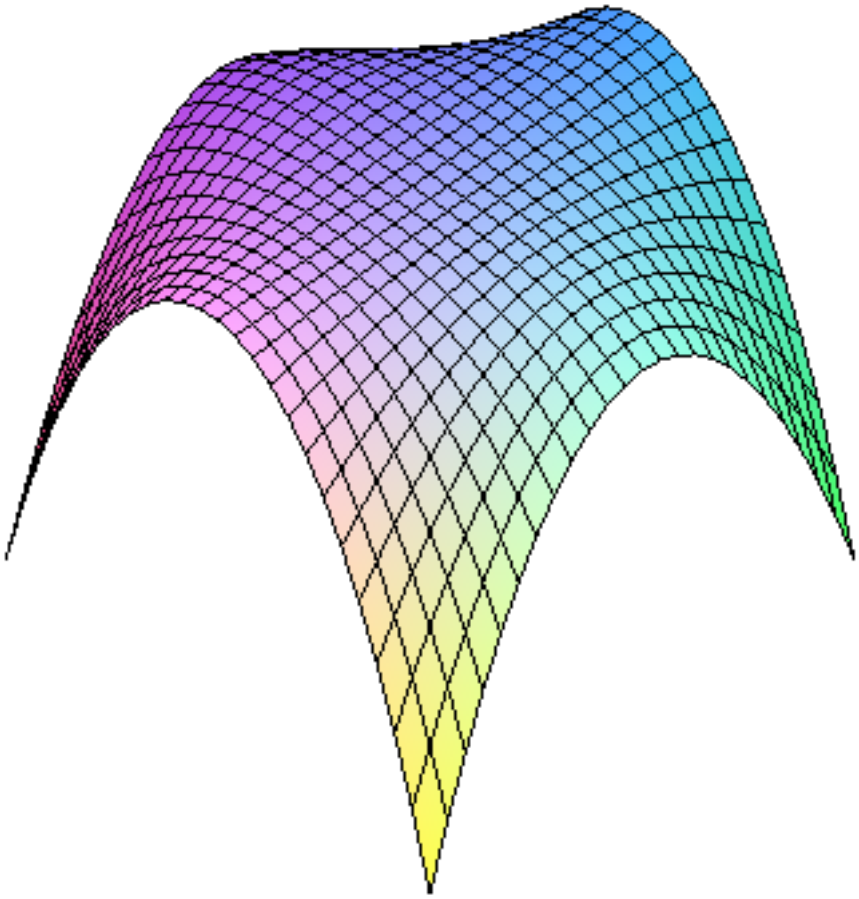
$$-4yx^3 - 83y^2x^2 + 97x^2 - 62x - 73y^2 - 10y^4 \tag{3.2.6}$$

```
> sort(r, y);
```

$$-10y^4 - 73y^2 - 83x^2y^2 - 4x^3y + 97x^2 - 62x \tag{3.2.7}$$

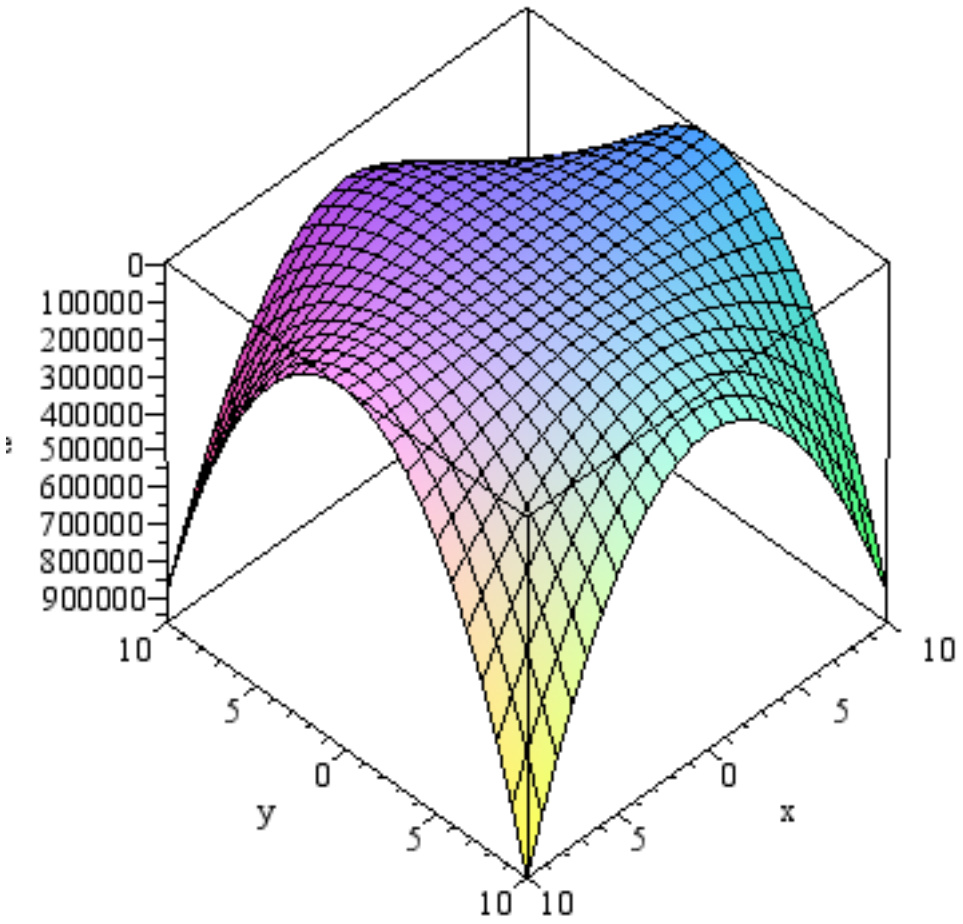
式 **(3.2.2)** を３次元プロットします． ３次元プロットには，plot3d コマンドを使用します.

```
> plot3d(r, x=-10..10, y=-10..10);
```



ボックス（目盛り）とラベルを追加します．

```
> plot3d(r, x=-10..10, y=-10..10, axes=boxed, labels=["x", "y",
  "Value"]);
```


0
100000
200000
300000
400000
500000
600000
700000
800000
900000
10
5
0
y
5
10
10
5
0
x
5
10

方程式

１変数の代数方程式

初期化します．

```
> restart;
```

１次方程式

１次方程式を p1 として定義します．

```
> p1 := a*x + b*x^0 = 0;
```

$$p1 := a\,x + b = 0 \qquad (4.1.1.1)$$

p1 を x について解きます．

```
> solve(p1, x);
```

$$-\frac{b}{a} \qquad (4.1.1.2)$$

２次方程式

２次方程式を p2 として定義します．

```
> p2 := a*x^2 + b*x^1 + c*x^0 = 0;
```

$$p2 := a\,x^2 + b\,x + c = 0 \qquad (4.1.2.1)$$

p2 を x について解きます．

```
> solve(p2, x);
```

$$\frac{-b + \sqrt{b^2 - 4\,a\,c}}{2\,a},\ -\frac{b + \sqrt{b^2 - 4\,a\,c}}{2\,a} \qquad (4.1.2.2)$$

３次方程式

３次方程式を p3 として定義します．

```
> p3 := a*x^3 + b*x^2 + c*x^1 + d*x^0 = 0;
```

$$p3 := a\,x^3 + b\,x^2 + c\,x + d = 0 \qquad (4.1.3.1)$$

p3 を x について解きます．

```
> solve(p3, x):
```

（参考）コロン（：）を用いて，結果を非表示にしています．

4次方程式

4 次方程式を p4 として定義します.

```
> p4 := a*x^4 + b*x^3 + c*x^2 + d*x^1 + e*x^0 = 0;
```

$$p4 := a\,x^4 + b\,x^3 + c\,x^2 + d\,x + e = 0 \qquad \textbf{(4.1.4.1)}$$

p4 を x について解きます.

```
> solve(p4, x);
```

$$RootOf\left(a\,\_Z^4 + b\,\_Z^3 + c\,\_Z^2 + d\,\_Z + e\right) \qquad \textbf{(4.1.4.2)}$$

RootOf() の解

$RootOf(\ )$ は，方程式の解が代数的に表現不可能なとき，あるいは解の式が長くなるようなときに，解の別表現として使用されます．$RootOf(\ )$ の解表現を，可能な限り四則演算と根号で記述する allvalues コマンドがあります.

```
> sol4 := [ allvalues((4.1.4.2)) ]: # 式が長いため非表示にします
```

解の数を nops コマンドで調査します．nops コマンドは，被演算子（オペランド）の数を返します.

```
> nops(sol4);
```

$$4 \qquad \textbf{(4.1.4.1.1)}$$

根が 4 つあることがわかります.

5次方程式

5 次方程式を p5 として定義します.

```
> p5 := a*x^5 + b*x^4 + c*x^3 + d*x^2 + e*x^1 + f*x^0 = 0;
```

$$p5 := a\,x^5 + b\,x^4 + c\,x^3 + d\,x^2 + e\,x + f = 0 \qquad \textbf{(4.1.5.1)}$$

p5 を x について解きます.

```
> solve(p5, x);
```

$$RootOf\left(a\,\_Z^5 + b\,\_Z^4 + c\,\_Z^3 + d\,\_Z^2 + e\,\_Z + f\right) \qquad \textbf{(4.1.5.2)}$$

一般的に，5 次以上の代数方程式は，その解を四則演算と根号で記述できないことが証明されています（アーベルの定理＝5 次以上の場合，解の公式は存在しない）．その場合，数値的に解を求めることになります（当然，1 次から 4 次までの方程式においても解を数値的に求めることはできます）.

高次方程式の数値解（近似解）

係数を定義します.

```
> param := [ a =    1.2,
             b =   10.1,
             c =   11.2,
             d =   -5.8,
             e =   38.1,
             f = -100.6 ];
```

$$param := [a = 1.2, b = 10.1, c = 11.2, d = -5.8, e = 38.1, f = -100.6] \quad \textbf{(4.1.6.1)}$$

p5 に param を代入します.

```
> np5 := eval(p5, param);
```

$$np5 := 1.2\,x^5 + 10.1\,x^4 + 11.2\,x^3 - 5.8\,x^2 + 38.1\,x - 100.6 = 0 \quad \textbf{(4.1.6.2)}$$

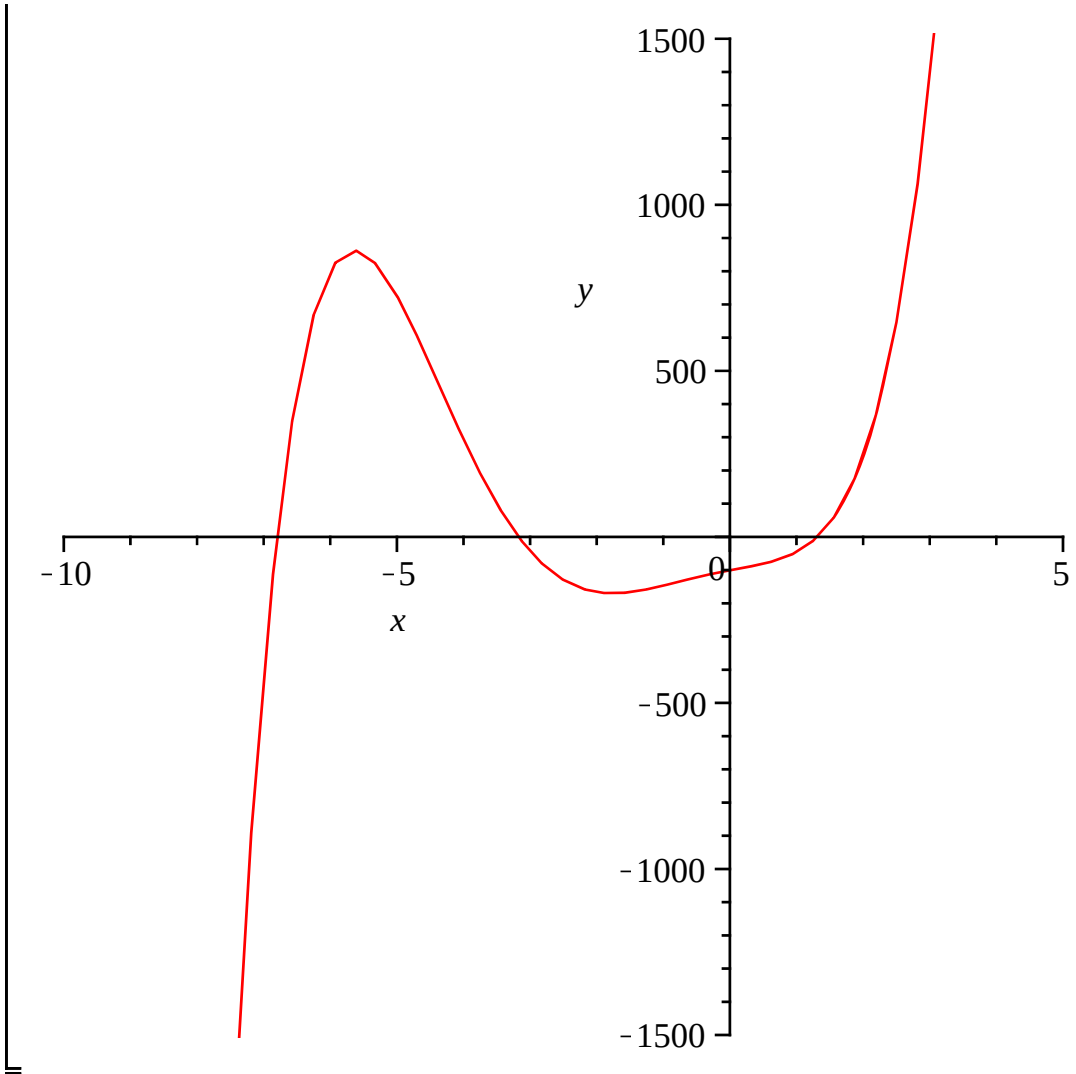
プロットするために，lhs コマンドを用いて左辺を抽出します.

```
> np51 := lhs(np5);
```

$$np51 := 1.2\,x^5 + 10.1\,x^4 + 11.2\,x^3 - 5.8\,x^2 + 38.1\,x - 100.6 \quad \textbf{(4.1.6.3)}$$

np51 を x=-10..5 および y=-1500..1500 の範囲でプロットします.

```
> plot( np51, x=-10..5, y=-1500..1500);
```



np5 を x について数値的に解きます.

```
> sol5 := fsolve(np5, x);
```

$$sol5 := -6.799151464, -3.172470811, 1.315734491 \quad \textbf{(4.1.6.4)}$$

根の数を調べるために fsolve(np5, x) を [ ] で括ります.

```
> sol5 := [ fsolve(np5, x) ];
```

$$sol5 := [-6.799151464, -3.172470811, 1.315734491] \quad \textbf{(4.1.6.5)}$$

根の数を調べます.

```
> nops(sol5);
```

$$3 \quad \textbf{(4.1.6.6)}$$

n 次の方程式は, （複素数平面上において） n 個の根を持ちます.

```
> csol5 := [ fsolve(np5, x, complex) ];
```

$$csol5 := [-6.799151464, -3.172470811, 0.1196105590 - 1.714524889\,I, \quad \textbf{(4.1.6.7)}$$
$$0.1196105590 + 1.714524889\,I, 1.315734491]$$

根の数を調べます．

```
> nops(csol5);
```

$$5 \qquad \textbf{(4.1.6.8)}$$

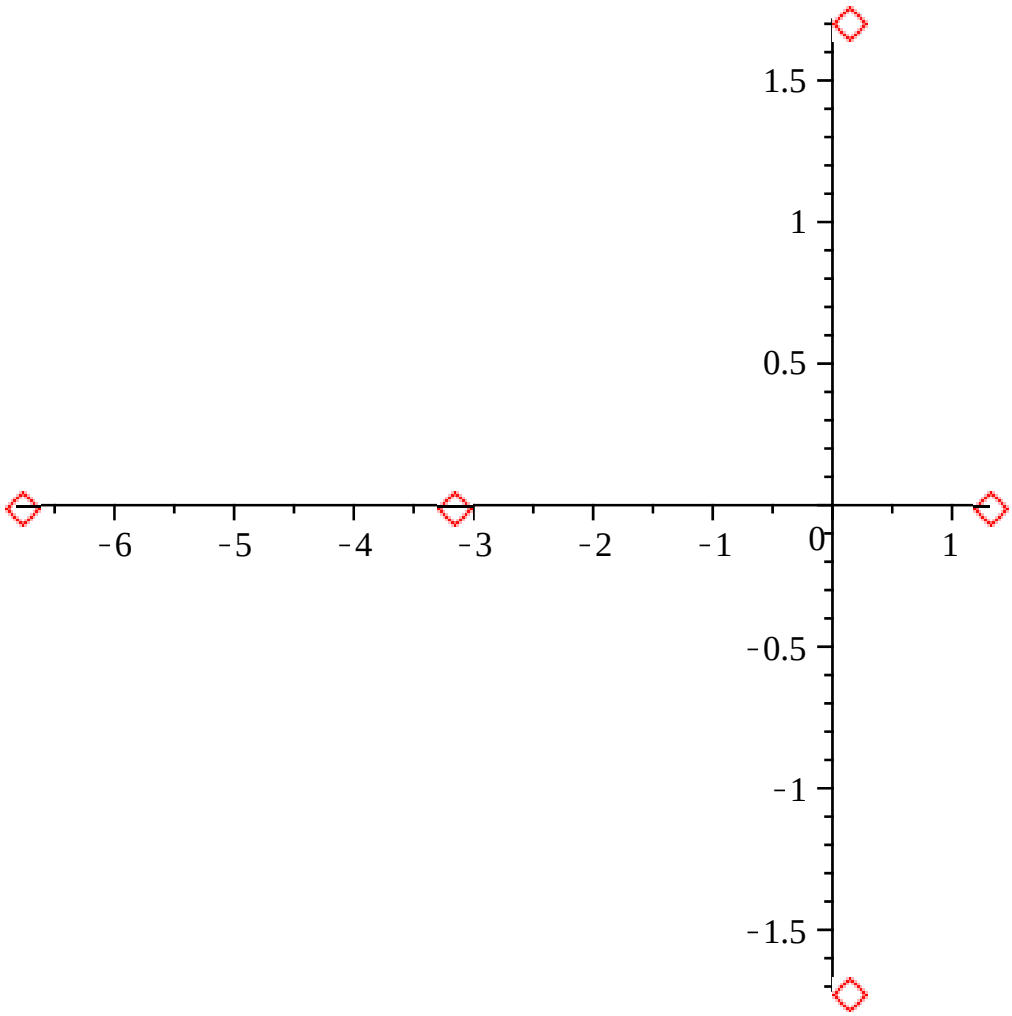
根が 5 つあることを示します．それぞれの根を表示します．

```
> csol5[1];
  csol5[2];
  csol5[3];
  csol5[4];
  csol5[5];
```

$$-6.799151464$$
$$-3.172470811$$
$$0.1196105590 - 1.714524889\,I$$
$$0.1196105590 + 1.714524889\,I$$
$$1.315734491 \qquad \textbf{(4.1.6.9)}$$

複素数の根を含む解 csol5 を複素平面上にプロットします．ここでは，plots パッケージをロードせずに complexplot コマンドを使用します．

```
> plots[complexplot]( csol5, style=point, symbolsize=24 );
```



円の方程式

初期化します.

```
> restart;
```

円の方程式を定義します.

```
> ceq := (x-a)^2 + (y-b)^2 = r^2;
```

$$ceq := (x-a)^2 + (y-b)^2 = r^2 \quad \textbf{(4.2.1)}$$

中心が［a=0,b=0］，半径が r=1 として，円の方程式を定義しなおします.

```
> param := [a=0, b=0, r=1];
```

$$param := [a = 0, b = 0, r = 1] \quad \textbf{(4.2.2)}$$

```
> ceq := eval(ceq, param);
```

$$ceq := x^2 + y^2 = 1 \quad \textbf{(4.2.3)}$$

半径 1 の円を表します.

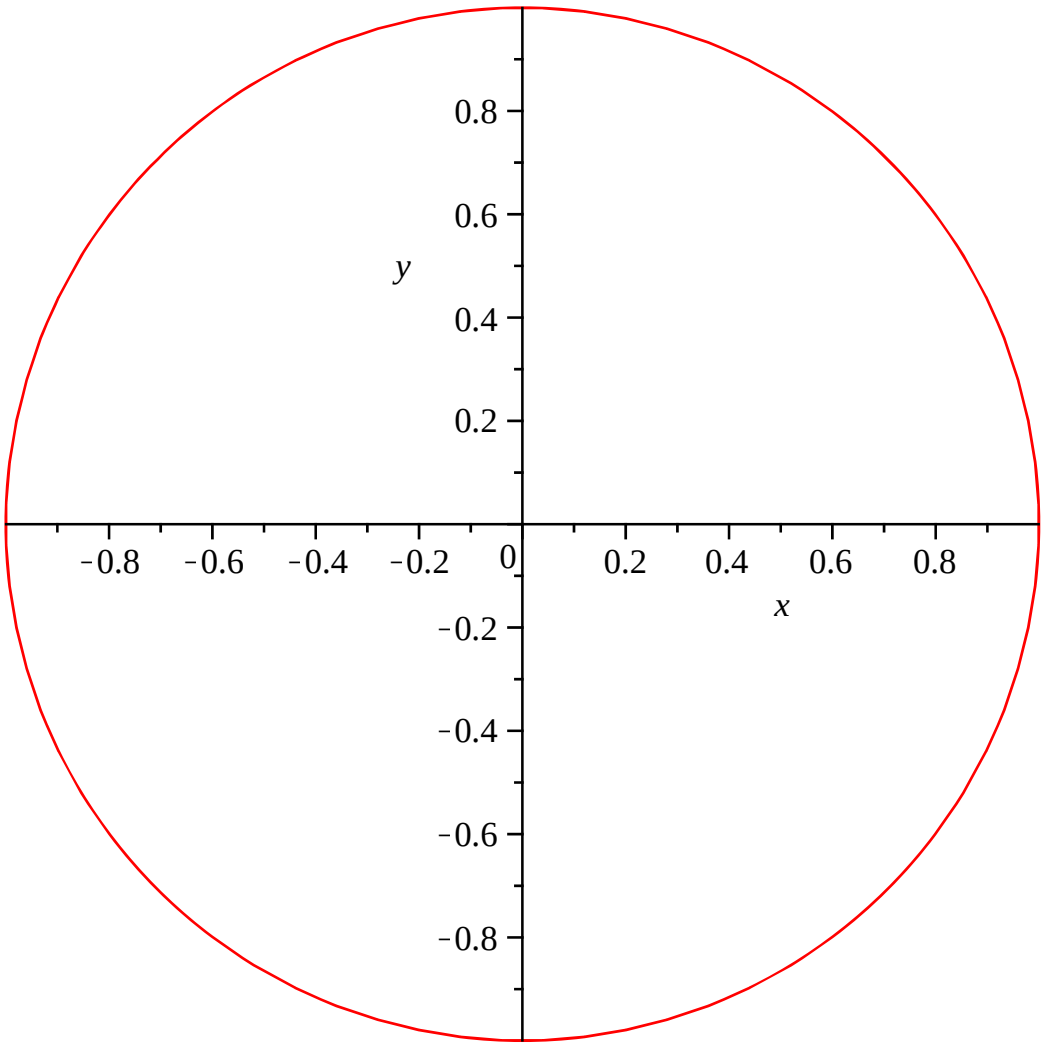
ここで，円の式を（陰関数をプロットする） implicitplot コマンドを用いてプロットします．ここでは，implicitplot コマンドを含む plots パッケージをロードします.

```
> with(plots):
```

x=-1..1， y=-1..1 の範囲で，ceq をプロットします.

$$ceq := x^2 + y^2 = 1$$

```
> implicitplot(ceq, x=-1..1, y=-1..1);
```


0.8
0.6
y
0.4
0.2
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
x
-0.2
-0.4
-0.6
-0.8

楕円の方程式

初期化します.

```
> restart;
```

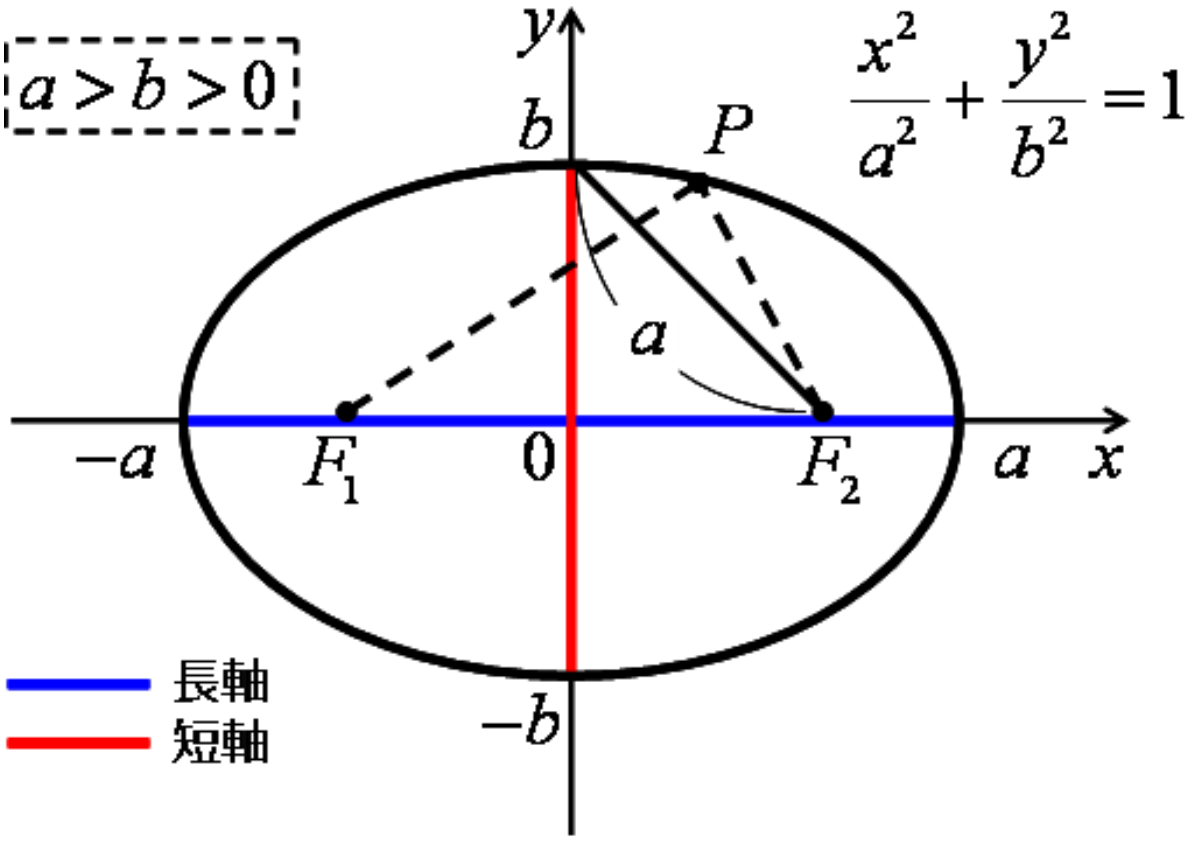
ここで，plots パッケージをロードします.

```
> with(plots):
```

楕円の方程式を定義します.

```
> eeq := x^2/a^2 + y^2/b^2 = 1;
```

$$eeq := \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \qquad \textbf{(4.3.1)}$$



（例）

2 点 $(5, 0)$，$(-5, 0)$ を焦点とした場合，長軸の長さが 16 の楕円の方程式は

$$\frac{x^2}{64} + \frac{y^2}{39} = 1$$

となります.

a, b を定義します.

```
> param := [a = 16/2, b = sqrt( (16/2)^2 - 5^2 ) ];
```

$$param := \left[a = 8, b = \sqrt{39}\right] \qquad \textbf{(4.3.2)}$$

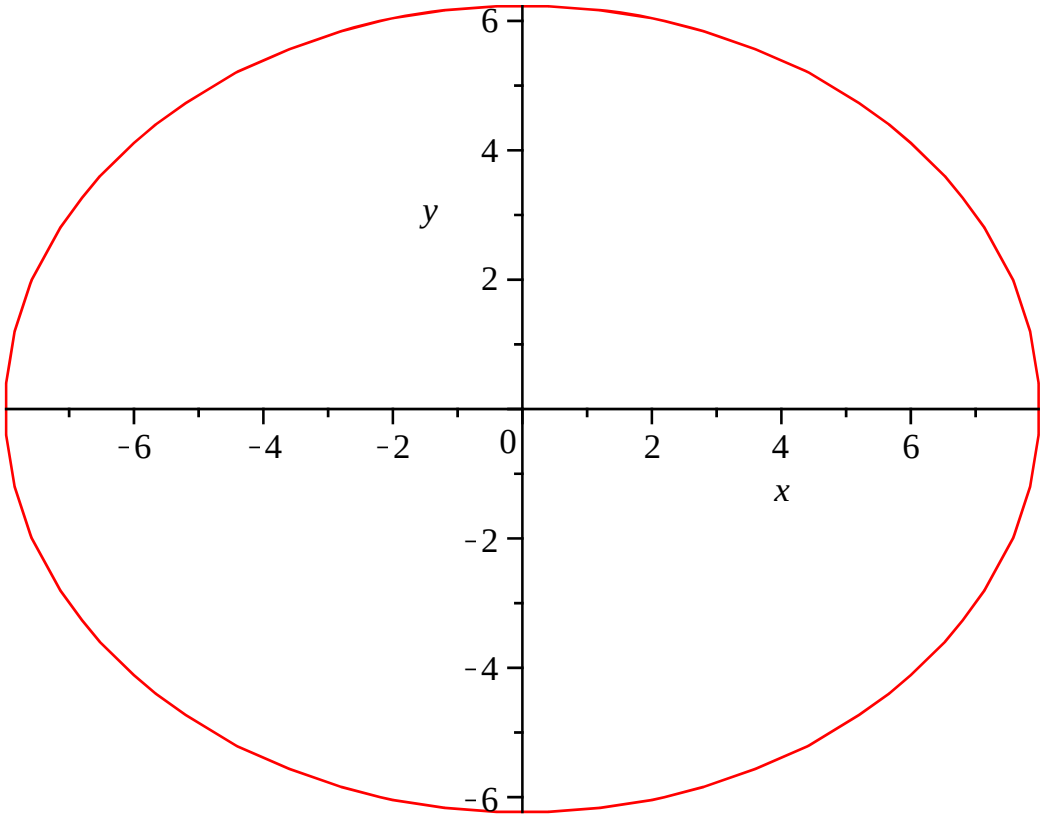
eeq を param を代入して評価します（計算します）.

```
> eeq := eval(eeq, param);
```

$$eeq := \frac{x^2}{64} + \frac{y^2}{39} = 1 \qquad \textbf{(4.3.3)}$$

implicitplot コマンドを用いて eeq をプロットします. オプション scaling=constrained は, プロットの縦横比を１：１に設定します.

```
> implicitplot(eeq, x=-10..10, y=-10..10, scaling=constrained);
```



双曲線の方程式

初期化します.

```
> restart;
```

双曲線の方程式を定義します.

```
> heq := x^2/a^2 - y^2/b^2 = 1;
```

$$heq := \frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \qquad \textbf{(4.4.1)}$$

漸近線の方程式

```
> yp :=  b*x/a;
  yn := -b*x/a;
```

$$yp := \frac{b\,x}{a}$$

$$yn := -\frac{b\,x}{a} \qquad \textbf{(4.4.2)}$$

（例）

2 点 $(4, 0)$, $(-4, 0)$ を焦点とした場合，主軸の長さが $2\sqrt{3}$ の双曲線の方程式，およびその漸近線の方程式は，それぞれ

$$\frac{x^2}{3} - \frac{y^2}{13} = 1$$

$$y = \pm\sqrt{\frac{13}{3}}\cdot x$$

となります.

a, b を定義します.

```
> param := [ a=sqrt(3), b=sqrt(13) ];
```

$$param := [a = \sqrt{3}, b = \sqrt{13}] \qquad \textbf{(4.4.3)}$$

双曲線の方程式 heq に代入します.

```
> heq := eval(heq, param);
```

$$heq := \frac{x^2}{3} - \frac{y^2}{13} = 1 \qquad \textbf{(4.4.4)}$$

漸近線の式 yp，yn に代入します．

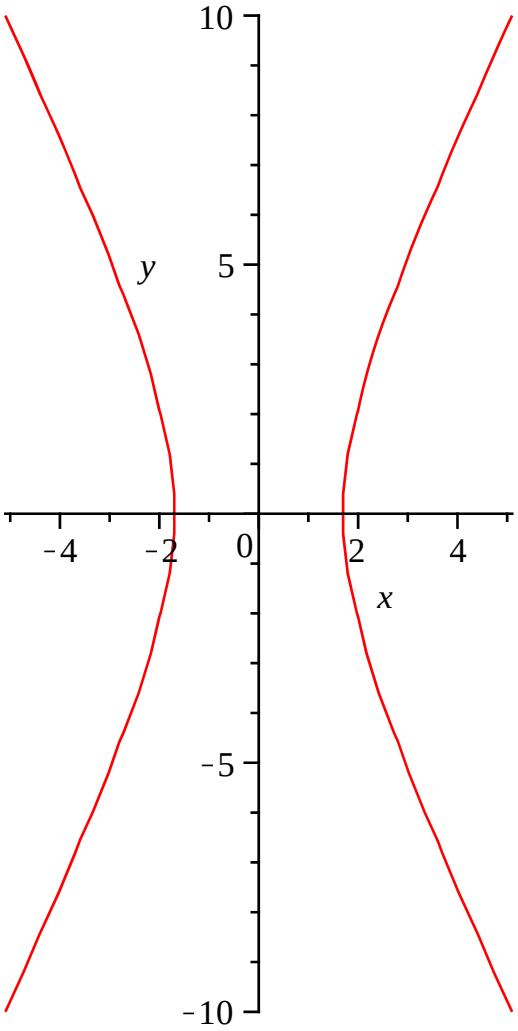
```
> yp := eval(yp, param);
  yn := eval(yn, param);
```

$$yp := \frac{\sqrt{13}\, x \sqrt{3}}{3}$$

$$yn := -\frac{\sqrt{13}\, x \sqrt{3}}{3} \qquad \textbf{(4.4.5)}$$

plots パッケージをロードせずに，双曲線 heq を implicitplot コマンドを使用します．オプション scaling=constrained は，プロットの縦横比を１：１に設定します．

```
> plots[implicitplot](heq, x=-10..10, y=-10..10, scaling=constrained);
```



プロットイメージを plt1 に割り当てます（格納します）．

```
> plt1 := plots[implicitplot](heq, x=-10..10, y=-10..10, scaling=
  constrained, color=red);
```

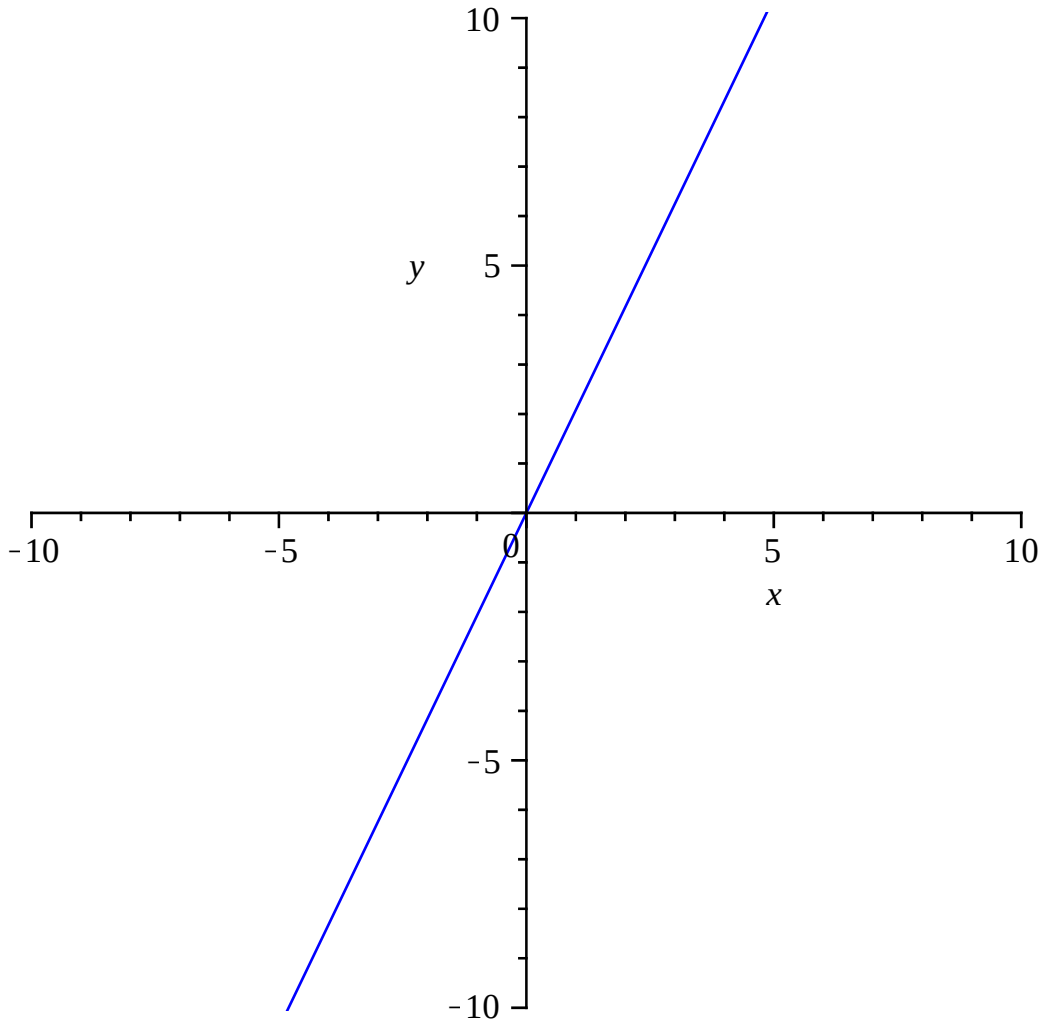
$$plt1 := PLOT(\ldots) \qquad \textbf{(4.4.6)}$$

PLOT(...) は，プロットコマンド（ plot や plot3d ）から作成される描画イメージを記述したデータ

です．また，PLOT(...) を変数に割り当てることで，その描画イメージを再利用することができます．

漸近線 yp をプロットします．オプションを scaling=constrained, color=blue とします．

```
> plot(yp, x=-10..10, y=-10..10, scaling=constrained, color=blue);
```



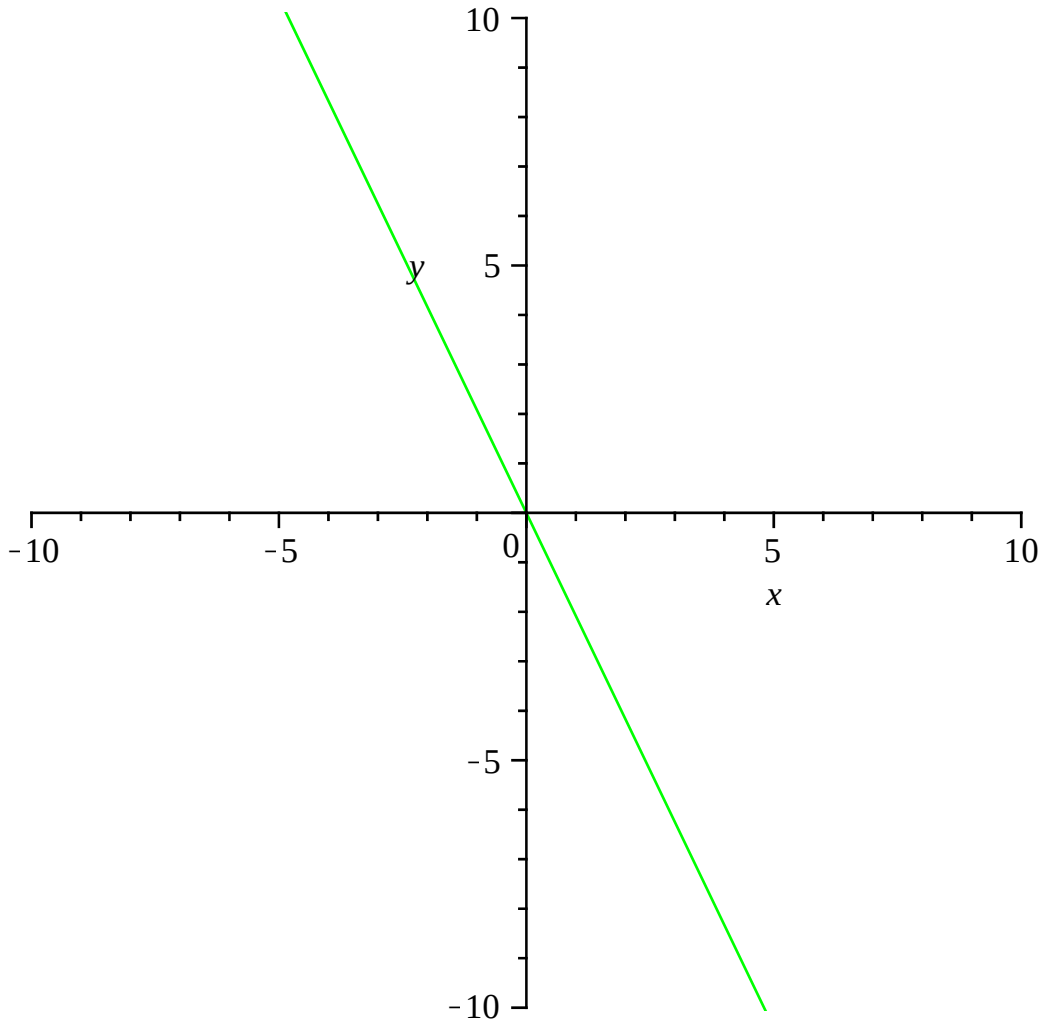
漸近線 yp のプロットイメージを plt2 に割り当てます．

```
> plt2 := plot(yp, x=-10..10, y=-10..10, scaling=constrained, color=
  blue);
```

$$plt2 := PLOT(\ldots) \tag{4.4.7}$$

漸近線 yn をプロットします．オプションを scaling=constrained, color=green とします．

```
> plot(yn, x=-10..10, y=-10..10, scaling=constrained, color=green);
```



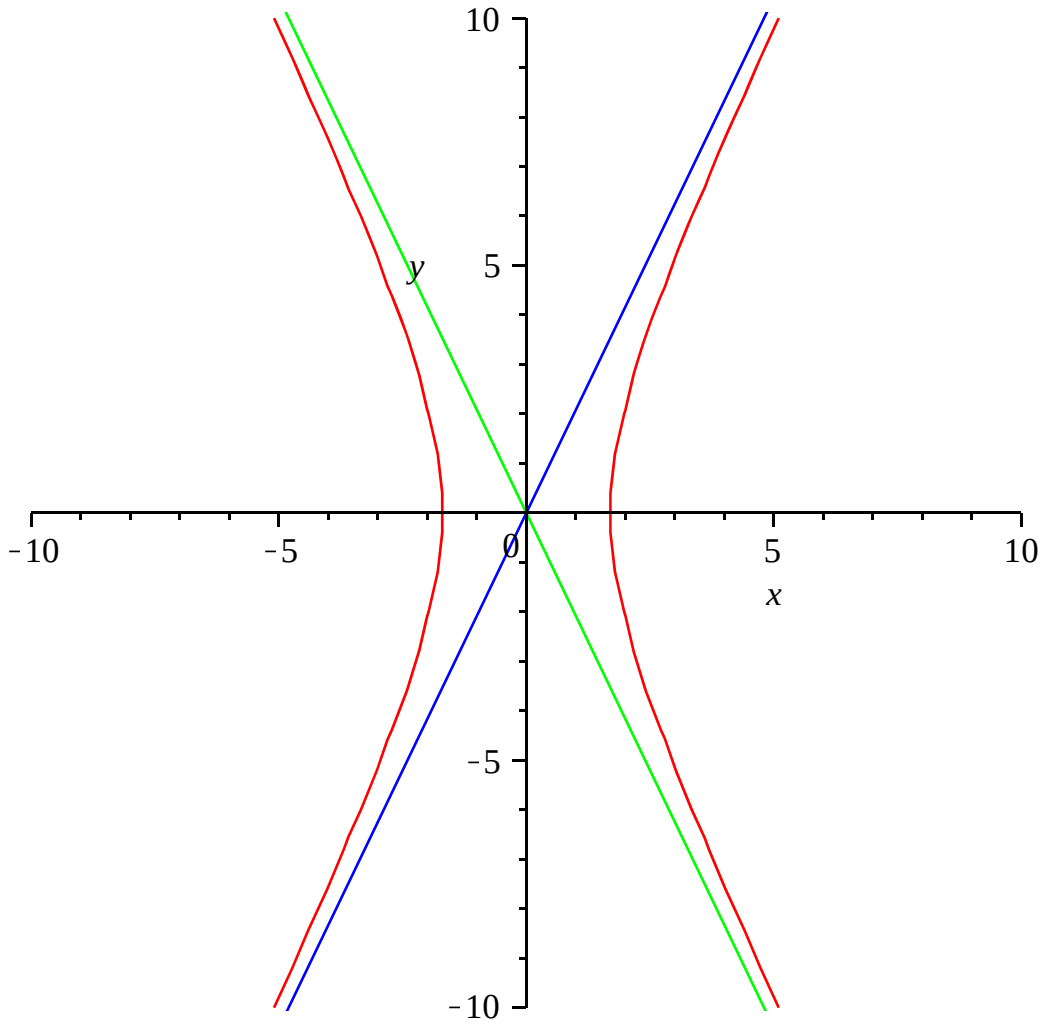
漸近線 yn のプロットイメージを plt3 に割り当てます.

```
> plt3 := plot(yn, x=-10..10, y=-10..10, scaling=constrained, color=
  green);
```

$$plt3 := PLOT(\ldots) \qquad \textbf{(4.4.8)}$$

plt1, plt2, plt3 を同時にプロットします. plots パッケージの display コマンドを使用します.

```
> plots[display]([plt1, plt2, plt3]);
```



display コマンドは PLOT データ（プロットイメージ）を表示するコマンドです.

連立代数方程式の解法とプロット

2 つの 2 次方程式を定義します.

```
> eq1 := y =  x^2 +   x - 3;
```

$$eq1 := y = x^2 + x - 3 \quad \textbf{(4.5.1)}$$

```
> eq2 := y = -x^2 - 3*x + 3;
```

$$eq2 := y = -x^2 - 3x + 3 \quad \textbf{(4.5.2)}$$

solve コマンドを用いて，[eq1, eq2] を連立させて，[x, y] について解きます.

```
> sol6 := solve([eq1, eq2], [x, y]);
```

$$sol6 := [[x = -3, y = 3], [x = 1, y = -1]] \quad \textbf{(4.5.3)}$$

2 つの 2 次方程式をプロットするために，rhs コマンドを用いて，それぞれの方程式の右辺を抽出します.

```
> aeq1 := rhs(eq1);
```

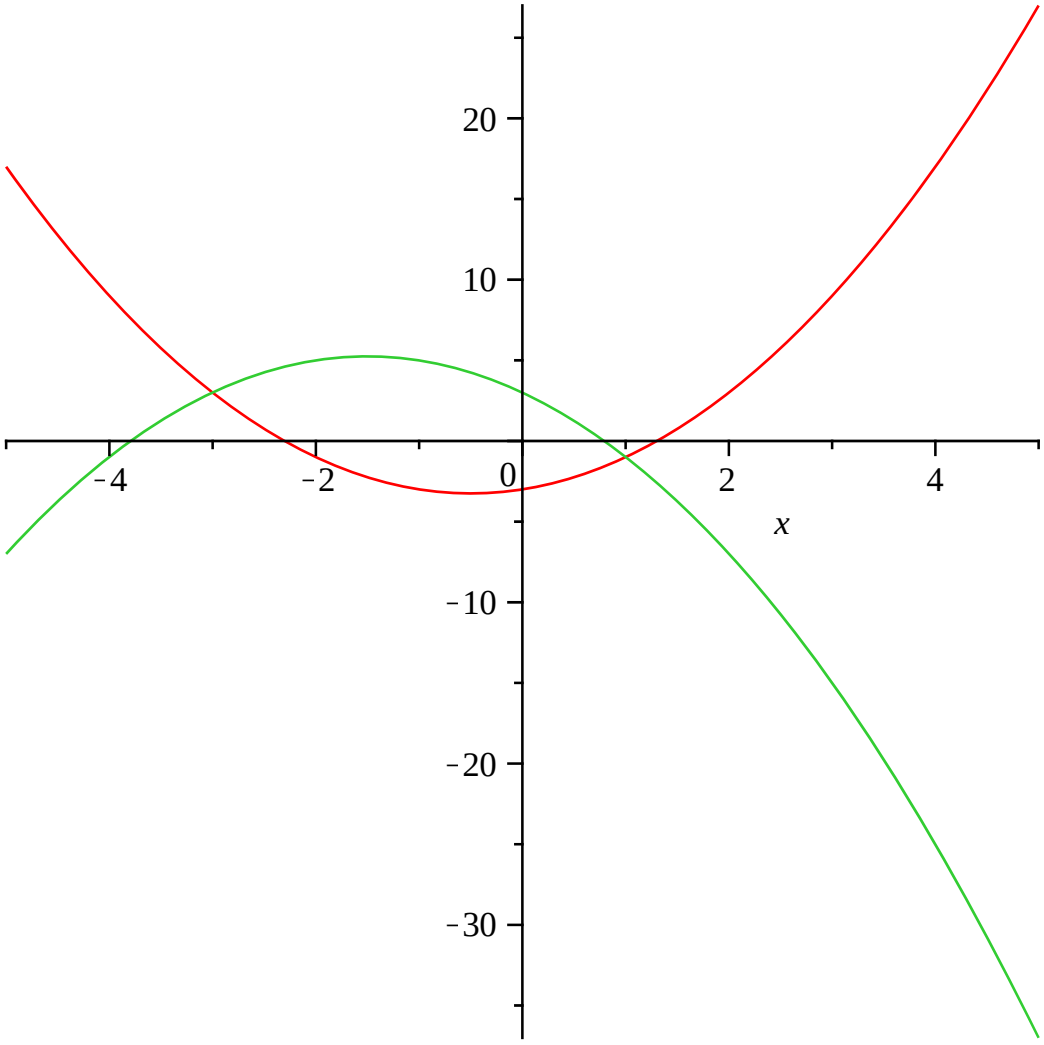
$$aeq1 := x^2 + x - 3 \quad \textbf{(4.5.4)}$$

```
> aeq2 := rhs(eq2);
```

$$aeq2 := -x^2 - 3\,x + 3 \quad \textbf{(4.5.5)}$$

x=-5..5 の範囲で，２つの（多項）式をプロットします．

```
> plot( [ aeq1, aeq2 ], x=-5..5 );
```



交点が２つあります．

プロットイメージを p1 に割り当てます．

```
> p1 := plot( [ aeq1, aeq2 ], x=-5..5 );
```

$$p1 := PLOT(\ldots) \quad \textbf{(4.5.6)}$$

解 $sol6 := [[x = -3, y = 3], [x = 1, y = -1]]$ を点として定義します．

```
> points := [[-3,3], [1,-1]];
```

$$points := [[-3, 3], [1, -1]] \quad \textbf{(4.5.7)}$$

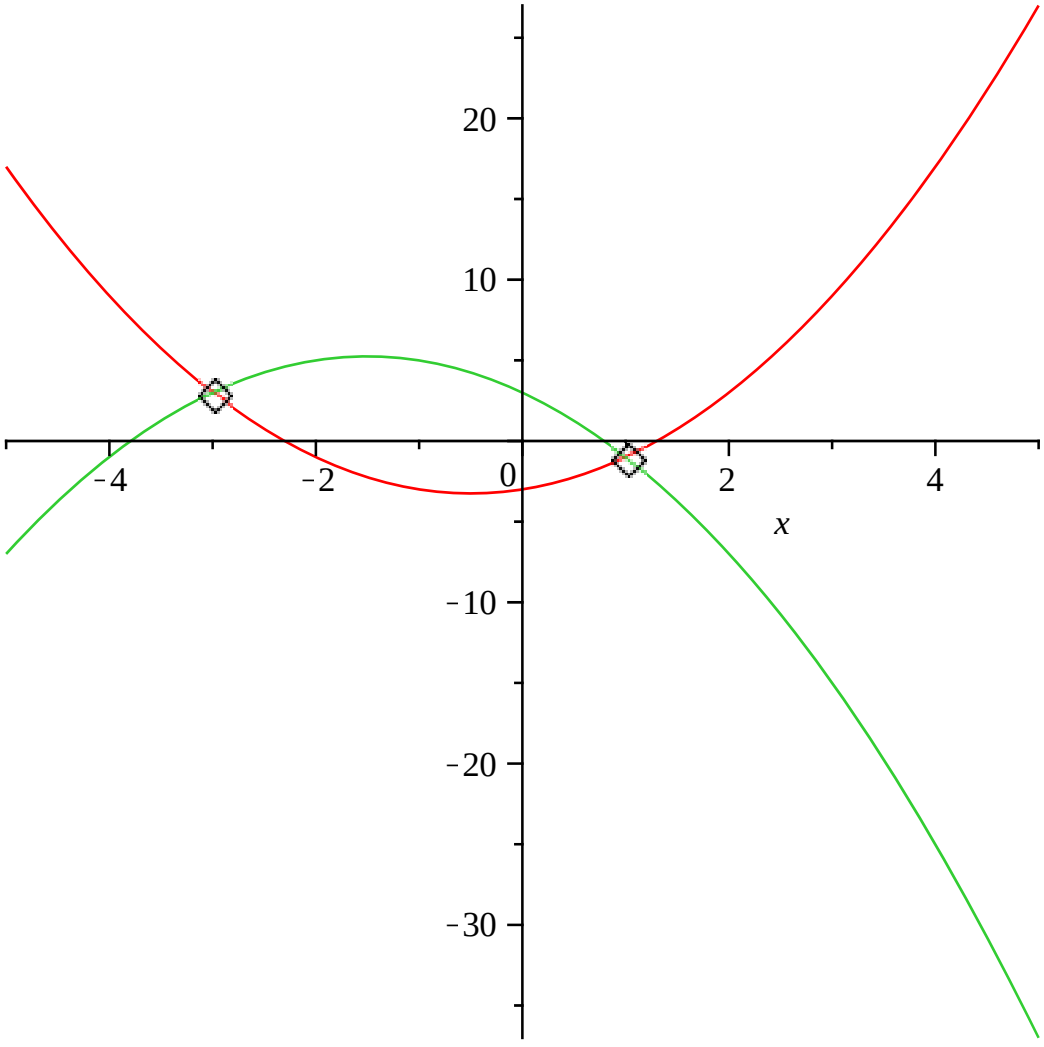
plots パッケージの pointplot コマンドを使用して，points をプロットします．ただし，そのままプロットイメージを p2 に割り当てます．

```
> p2 := plots[pointplot]( points, symbolsize=24 );
```

$$p2 := PLOT(\ldots) \qquad \textbf{(4.5.8)}$$

plots パッケージの display コマンドを用いて，p1 および p2 をプロットします．

```
> plots[display]([p1,p2]);
```



放物線と楕円の交点

初期化します.

```
> restart;
```

2 次方程式を定義します.

```
> eq3 := y = x^2 + 5*x - 3;
```

$$eq3 := y = x^2 + 5\,x - 3 \qquad \textbf{(4.6.1)}$$

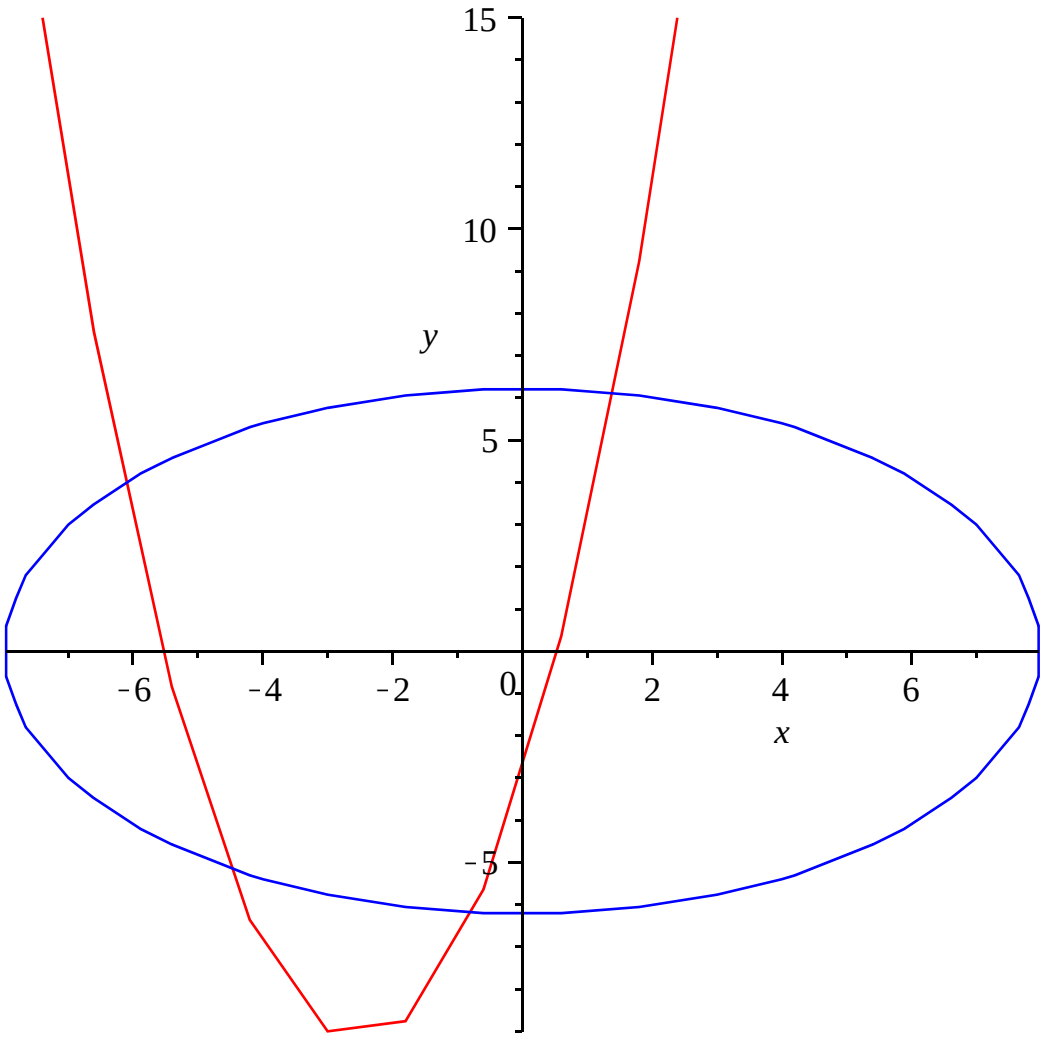
楕円の式を定義します.

```
> eq4 := (1/64)*x^2+(1/39)*y^2 = 1;
```

$$eq4 := \frac{x^2}{64} + \frac{y^2}{39} = 1 \qquad \textbf{(4.6.2)}$$

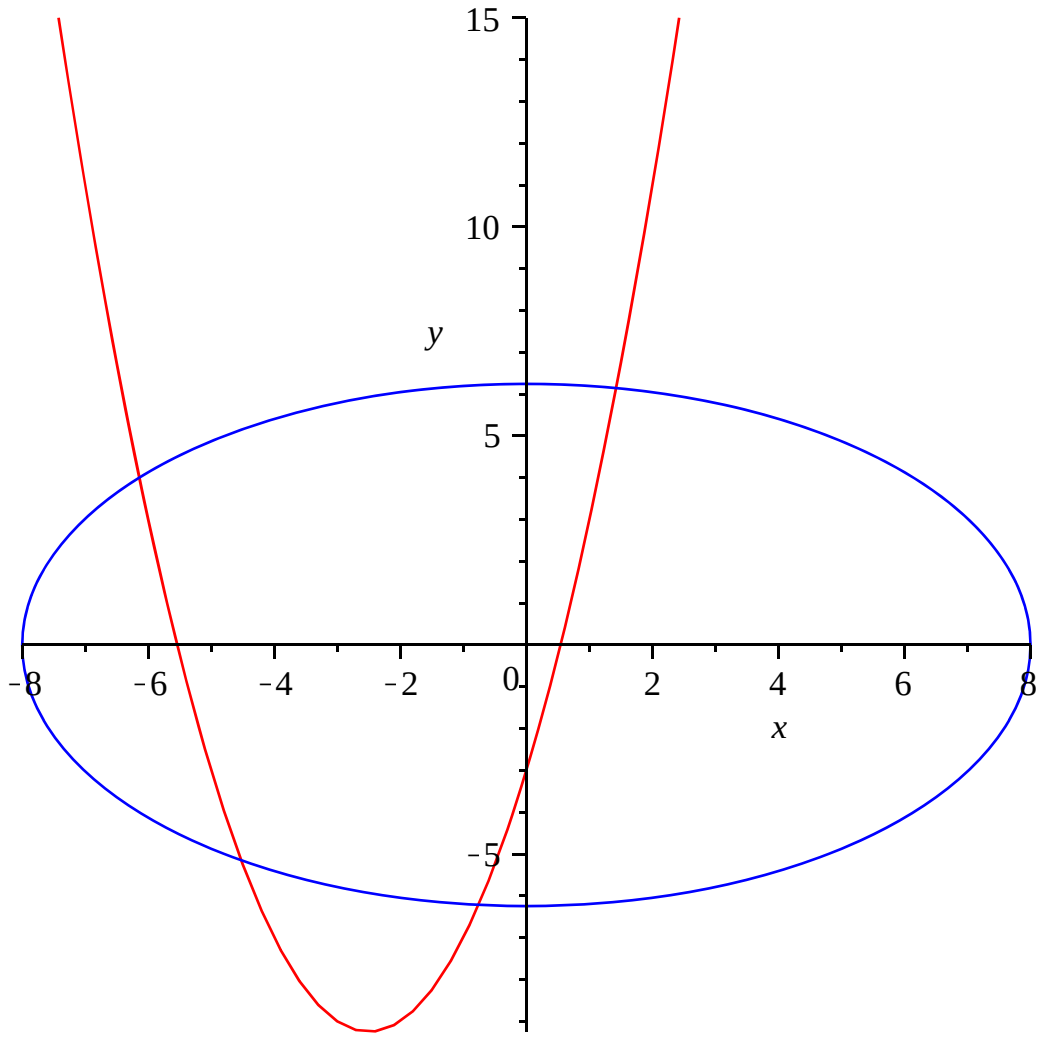
plots パッケージの implicitplot コマンドを用いて，eq3 と eq4 をプロットします．オプションを color=[red, blue] として，放物線を赤，楕円を青に設定します.

```
> plots[implicitplot]([eq3, eq4], x=-15..15, y=-15..15, color=[red,
  blue]);
```



線を滑らかにするために，numpoints=10000 を追加します．

```
> plots[implicitplot]([eq3, eq4], x=-15..15, y=-15..15, color=[red,
  blue], numpoints=10000 );
```



式 [eq3, eq4] を連立させて， [x, y] について解きます．ここで，solve コマンドを用います．

```
> sol7 := solve([eq3, eq4], [x, y]);
```

$$sol7 := \left[\left[x = RootOf\left(1255\,\_Z^2 + 64\,\_Z^4 + 640\,\_Z^3 - 1920\,\_Z - 1920,\ label = \_L1\right),\ y = RootOf\left(1255\,\_Z^2 + 64\,\_Z^4 + 640\,\_Z^3 - 1920\,\_Z - 1920,\ label = \_L1\right)^2 + 5\,RootOf\left(1255\,\_Z^2 + 64\,\_Z^4 + 640\,\_Z^3 - 1920\,\_Z - 1920,\ label = \_L1\right) - 3\right]\right] \quad \textbf{(4.6.3)}$$

（復習）RootOf の解

$RootOf(\ )$ は，方程式の解が代数的に表現不可能なとき，あるいは解の式が長くなるようなときに，解の別表現として使用されます．

$RootOf(\ )$ の解表現を，可能な限り四則演算と根号で記述する allvalues コマンドがあります．

```
> allvalues(sol7): # 結果が長いため非表示にします．
```

式の内容を確認します．

```
> eq3; eq4;
```

$$y = x^2 + 5x - 3$$

$$\frac{x^2}{64} + \frac{y^2}{39} = 1 \qquad \textbf{(4.6.4)}$$

数値解の求め方

```
> fsolve([eq3, eq4], {x,y});
```

$$\{x = -0.7584121437, y = -6.216871739\} \qquad \textbf{(4.6.2.1)}$$

ただし，fsolve([eq3, eq4], {x, y}) では，ひとつしか解が求まりません．これは，（おおまかには）数値的な解の探索において，他の解に到達できないために起こります．そこで，ここでは，ある１点［１， １］を決めて，その周りにある解を探索します．

```
> s81 := fsolve([eq3, eq4], {x=1,y=1});
```

$$s81 := \{x = 1.423686117, y = 6.145312741\} \qquad \textbf{(4.6.2.2)}$$

オプション avoid={s81} を追加して，既に見つけた解を避けるように次の解を探索します．

```
> s82 := fsolve([eq3, eq4], {x=1,y=1}, avoid={s81});
```

$$s82 := \{x = -0.7584121437, y = -6.216871739\} \qquad \textbf{(4.6.2.3)}$$

{s81, s82} を省いて解を探します．

```
> s83 := fsolve([eq3, eq4], {x=1,y=1}, avoid={s81, s82});
```

$$s83 := \{x = -6.140456398, y = 4.002922787\} \qquad \textbf{(4.6.2.4)}$$

{s81, s82, s83} を省いて解を探します．

```
> s84 := fsolve([eq3, eq4], {x=1,y=1}, avoid={s81, s82, s83});
```

$$s84 := \{x = -4.524817575, y = -5.150113789\} \qquad \textbf{(4.6.2.5)}$$

{s81, s82, s83, s84} を省いて解を探します．

```
> s85 := fsolve([eq3, eq4], {x=1,y=1}, avoid={s81, s82, s83, s84});
```

$$s85 := fsolve\left(\left[y = x^2 + 5x - 3, \frac{x^2}{64} + \frac{y^2}{39} = 1\right], \{x = 1, y = 1\}, avoid = \{\{x = -6.140456398, y = 4.002922787\}, \{x = -4.524817575, y = -5.150113789\}, \{x = -0.7584121437, y = -6.216871739\}, \{x = 1.423686117, y = 6.145312741\}\}\right) \qquad \textbf{(4.6.2.6)}$$

ただし，解の数はプロットから４個になります．そのため，解 s85 は見つかりません（ある解に収束しません）．

要素の操作

解 s81 は {　} で表現されます．以下の方法で，[　] に変換することができます．

```
> convert( s81, list );
```

$$[x = 1.423686117, y = 6.145312741] \quad \textbf{(4.6.3.1)}$$

さらに，x および y の値を取り出すには，以下のような処理を実行します．

ひとつめの要素を表示

```
> s81[1];
```

$$x = 1.423686117 \quad \textbf{(4.6.3.2)}$$

ふたつめの要素を表示

```
> s81[2];
```

$$y = 6.145312741 \quad \textbf{(4.6.3.3)}$$

右辺を抽出

```
> rhs( s81[1] );
```

$$1.423686117 \quad \textbf{(4.6.3.4)}$$

右辺を抽出

```
> rhs( s81[2] );
```

$$6.145312741 \quad \textbf{(4.6.3.5)}$$

[ ] で整理

```
> [rhs( s81[1] ), rhs( s81[2] )];
```

$$[1.423686117, 6.145312741] \quad \textbf{(4.6.3.6)}$$

（参考）以下は，上記手順と等価な処理になります．

```
> map( rhs, convert( s81, list ) );
```

$$[1.423686117, 6.145312741] \quad \textbf{(4.6.3.7)}$$

map コマンドは，ひとつめの引数（ rhs コマンド）をその後ろにある引数（ 実際は，$[x = 1.423686117, y = 6.145312741]$ ）の各要素に作用させます．すなわち，以下のような処理を行っています（rhs コマンドは，式の右辺を抽出するコマンドです）．

```
> [ rhs(x = 1.423686117), rhs(y = 6.145312741) ];
```

$$[1.423686117, 6.145312741] \quad \textbf{(4.6.3.8)}$$

関数

三角関数

初期化します.

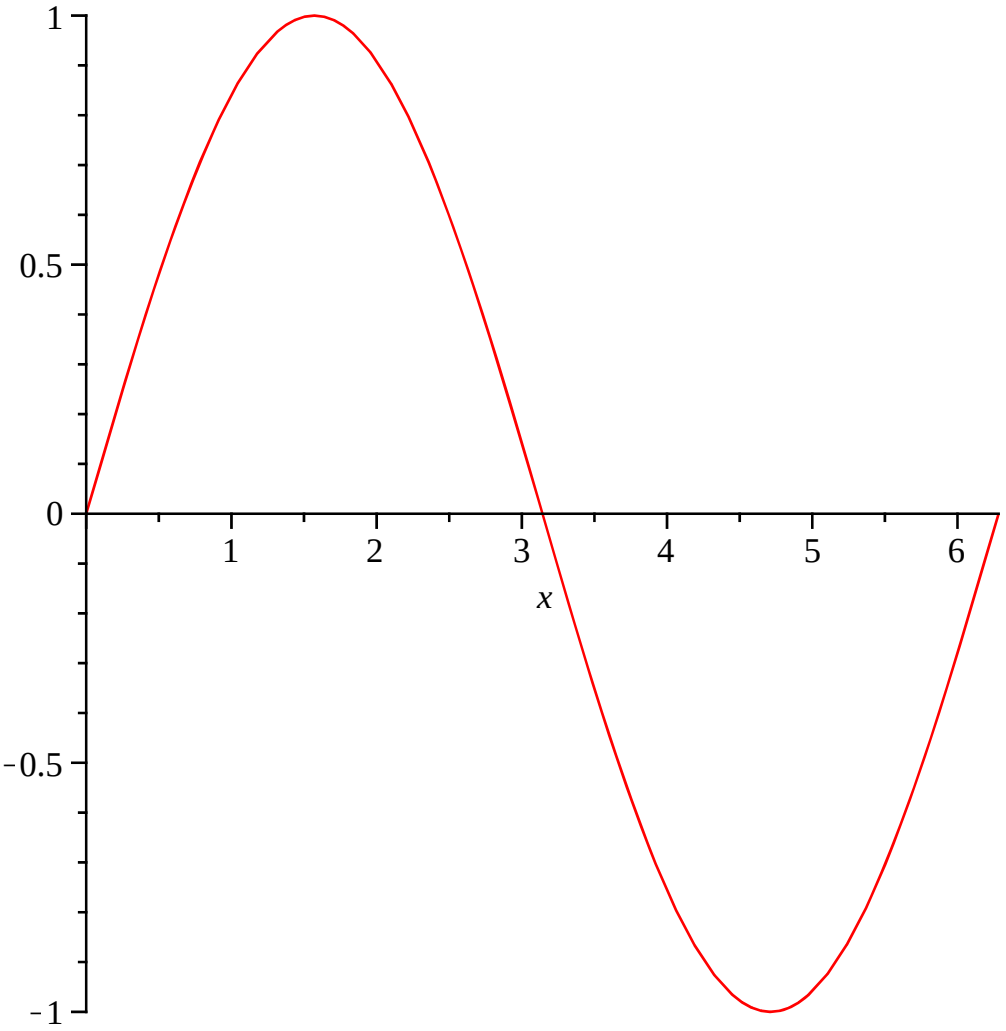
```
> restart;
```

sin(x) を定義しプロットします.

```
> tr1 := sin(x);
```

$$tr1 := \sin(x) \quad \textbf{(5.1.1)}$$

```
> plot(tr1, x=0..2*Pi);
```

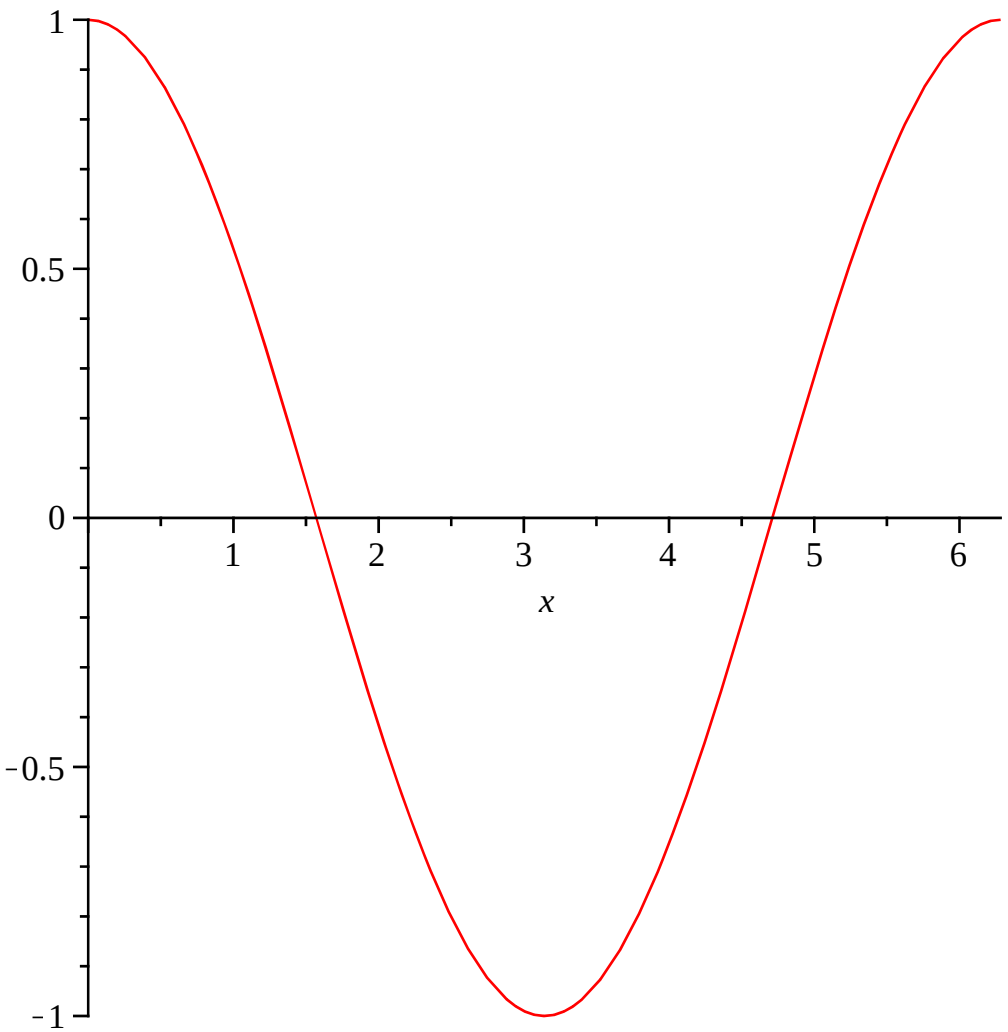


cos(x) を定義しプロットします.

```
> tr2 := cos(x);
```

$$tr2 := \cos(x) \quad \textbf{(5.1.2)}$$

```
> plot(tr2, x=0..2*Pi);
```

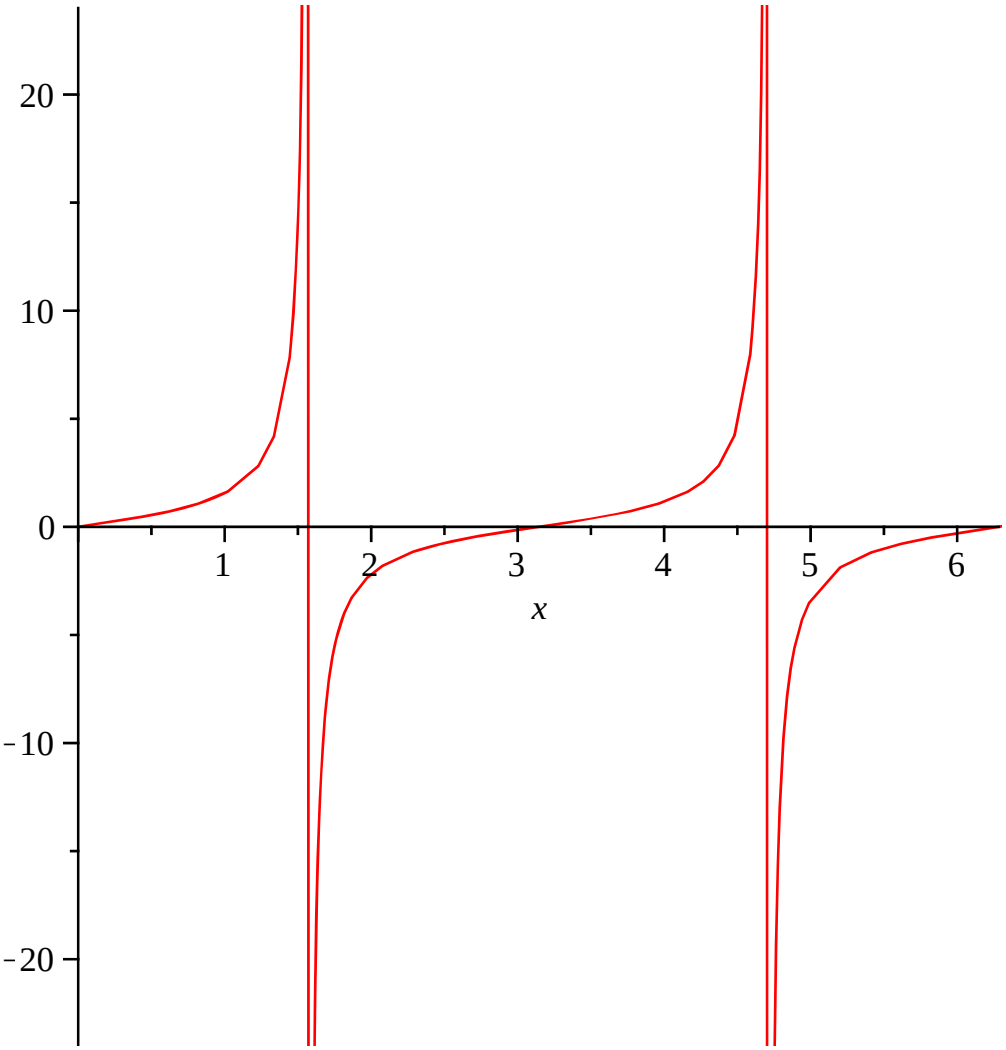


tan(x) を定義し，プロットします．

```
> tr3 := tan(x);
```

$$tr3 := \tan(x) \quad \textbf{(5.1.3)}$$

```
> plot(tr3, view=[0..2*Pi, -24..24]);
```



三角関数の基本公式

（第１式） $\sin(\theta)^2 + \cos(\theta)^2 = 1$

（第２式） $\tan(\theta)^2 + 1 = \frac{1}{\cos(\theta)^2}$

第１式の関係を確認します．

はじめに，第１式の左辺を定義します．

```
> treq1 := sin(theta)^2 + cos(theta)^2;
```

$$treq1 := \sin(\theta)^2 + \cos(\theta)^2 \qquad \textbf{(5.1.1.1)}$$

式 **(5.1.1.1)** を簡単化します．

```
> simplify(treq1);
```

$$1 \qquad \textbf{(5.1.1.2)}$$

以上から，$\sin(\theta)^2+\cos(\theta)^2=1$ の関係が示されます．

次に，第２式の左辺を定義して，第２式の関係を確認します．

```
> treq2 := tan(theta)^2 + 1;
```

$$treq2 := \tan(\theta)^2+1 \qquad \textbf{(5.1.1.3)}$$

式 **(5.1.1.3)** を簡単化します．

```
> treq3 := simplify( treq2 );
```

$$treq3 := \frac{1}{\cos(\theta)^2} \qquad \textbf{(5.1.1.4)}$$

以上から，$\tan(\theta)^2+1=\frac{1}{\cos(\theta)^2}$ の関係が示されます．

（参考）第１式から第２式を導く手順

なお，以下の手順で第１式から第２式を導けます．

１番目の式を定義します。

```
> treq1a := sin(theta)^2 + cos(theta)^2 = 1;
```

$$treq1a := \sin(\theta)^2+\cos(\theta)^2=1 \qquad \textbf{(5.1.1.1.1)}$$

式 **(5.1.1.1.1)** の両辺を $\cos(\theta)^2$ で割ります．

```
> treq2a := treq1a/cos(theta)^2;
```

$$treq2a := \frac{\sin(\theta)^2+\cos(\theta)^2}{\cos(\theta)^2}=\frac{1}{\cos(\theta)^2} \qquad \textbf{(5.1.1.1.2)}$$

式 **(5.1.1.1.2)** を展開します．

```
> treq3a := expand( treq2a );
```

$$treq3a := \frac{\sin(\theta)^2}{\cos(\theta)^2}+1=\frac{1}{\cos(\theta)^2} \qquad \textbf{(5.1.1.1.3)}$$

式 **(5.1.1.1.3)** の左辺のみを変換して等式を変形します．ここで，$\tan(\theta)=\frac{\sin(\theta)}{\cos(\theta)}$ の関係を用います．

```
> treq4a := convert( lhs(treq4a), tan) = rhs(treq4a);
```

$$treq4a := \tan(\theta)^2 + 1 = \frac{1}{\cos(\theta)^2} \qquad \textbf{(5.1.1.1.4)}$$

以上の手順によって，第１式 $\sin(\theta)^2 + \cos(\theta)^2 = 1$ から第２式 $\tan(\theta)^2 + 1 = \frac{1}{\cos(\theta)^2}$ が導かれます．

加法定理

式を定義します．

```
> add1 := sin(A+B);
```

$$add1 := \sin(A + B) \qquad \textbf{(5.1.2.1)}$$

展開します．

```
> add2 := expand(add1);
```

$$add2 := \sin(A)\cos(B) + \cos(A)\sin(B) \qquad \textbf{(5.1.2.2)}$$

式を定義します．

```
> add3 := sin(A-B);
```

$$add3 := \sin(A - B) \qquad \textbf{(5.1.2.3)}$$

展開します．

```
> add4 := expand(add3);
```

$$add4 := \sin(A)\cos(B) - \cos(A)\sin(B) \qquad \textbf{(5.1.2.4)}$$

結合します．

```
> combine(add2);
```

$$\sin(A + B) \qquad \textbf{(5.1.2.5)}$$

結合します．

```
> combine(add4);
```

$$\sin(A - B) \qquad \textbf{(5.1.2.6)}$$

双曲線関数

初期化します.

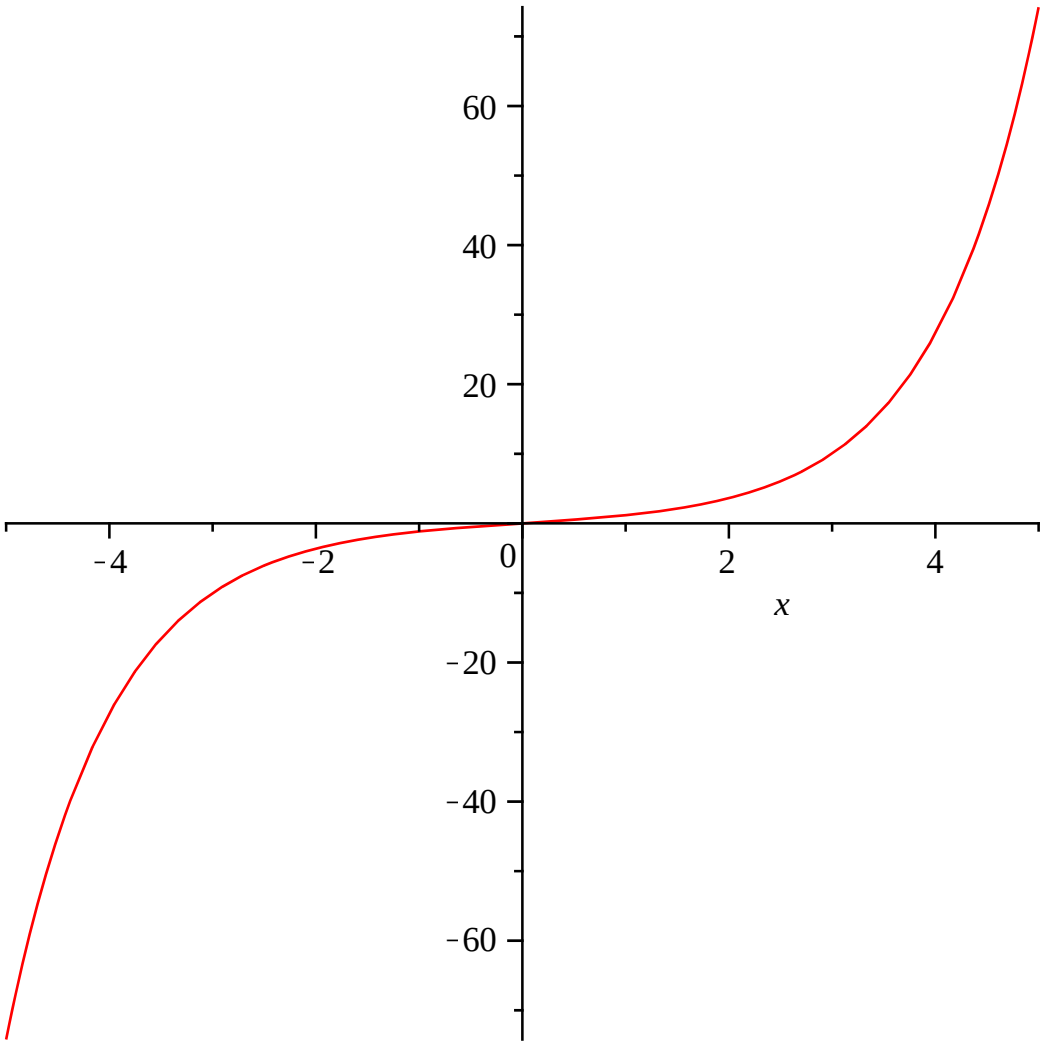
```
> restart;
```

双曲線正弦関数 (hyperbolic sine; ハイパボリックサイン)を定義し， x=-5..5 の範囲でプロットします.

```
> hyp1 := sinh(x);
```

$$hyp1 := \sinh(x) \tag{5.2.1}$$

```
> plot(hyp1, x=-5..5);
```



指数関数の形に変換します.

```
> convert(hyp1, exp);
```

$$\frac{e^{x}}{2} - \frac{e^{-x}}{2} \tag{5.2.2}$$

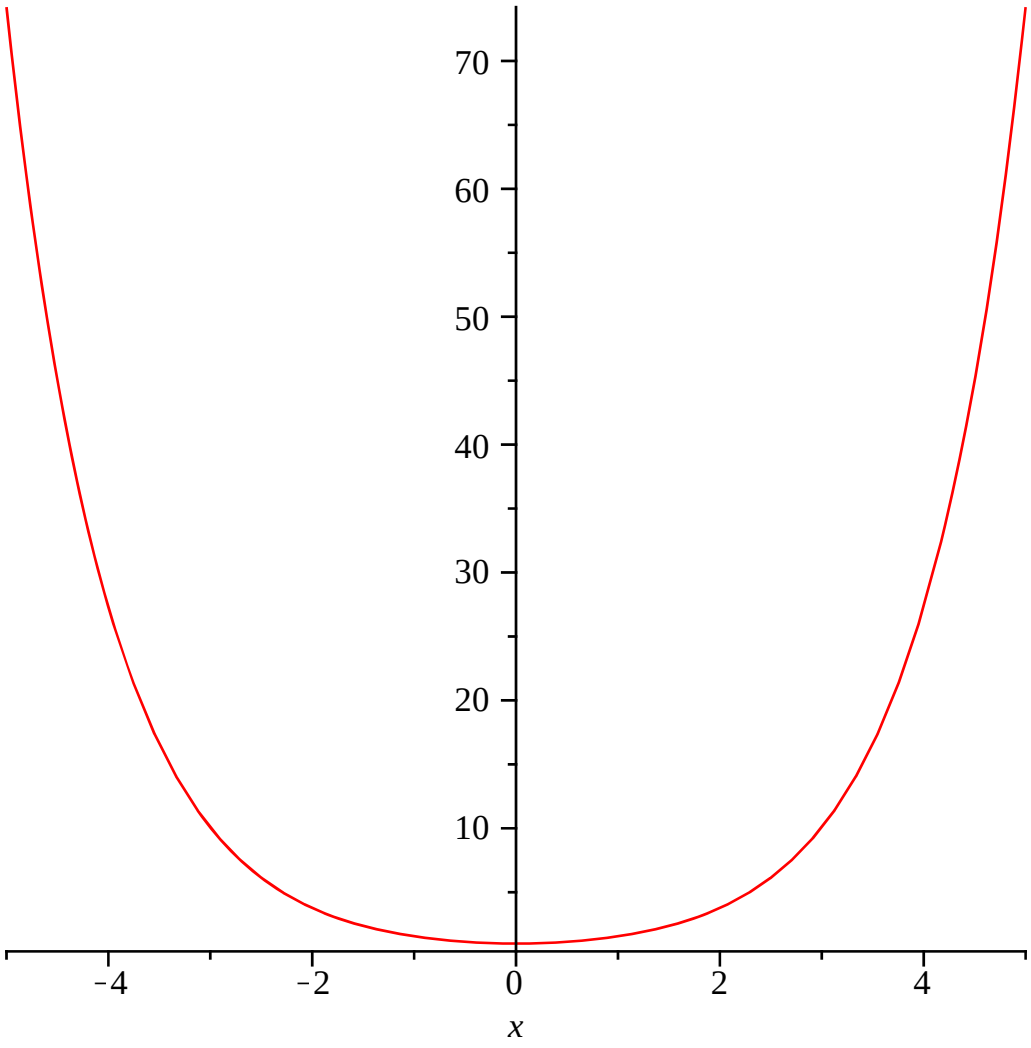
双曲線余弦関数 (hyperbolic cosine; ハイパボリックコサイン)を定義し， x=-5..5 の範囲でプロッ

トします.

```
> hyp2 := cosh(x);
```

$$hyp2 := \cosh(x) \tag{5.2.3}$$

```
> plot(hyp2, x=-5..5);
```



指数関数の形に変換します.

```
> convert(hyp2, exp);
```

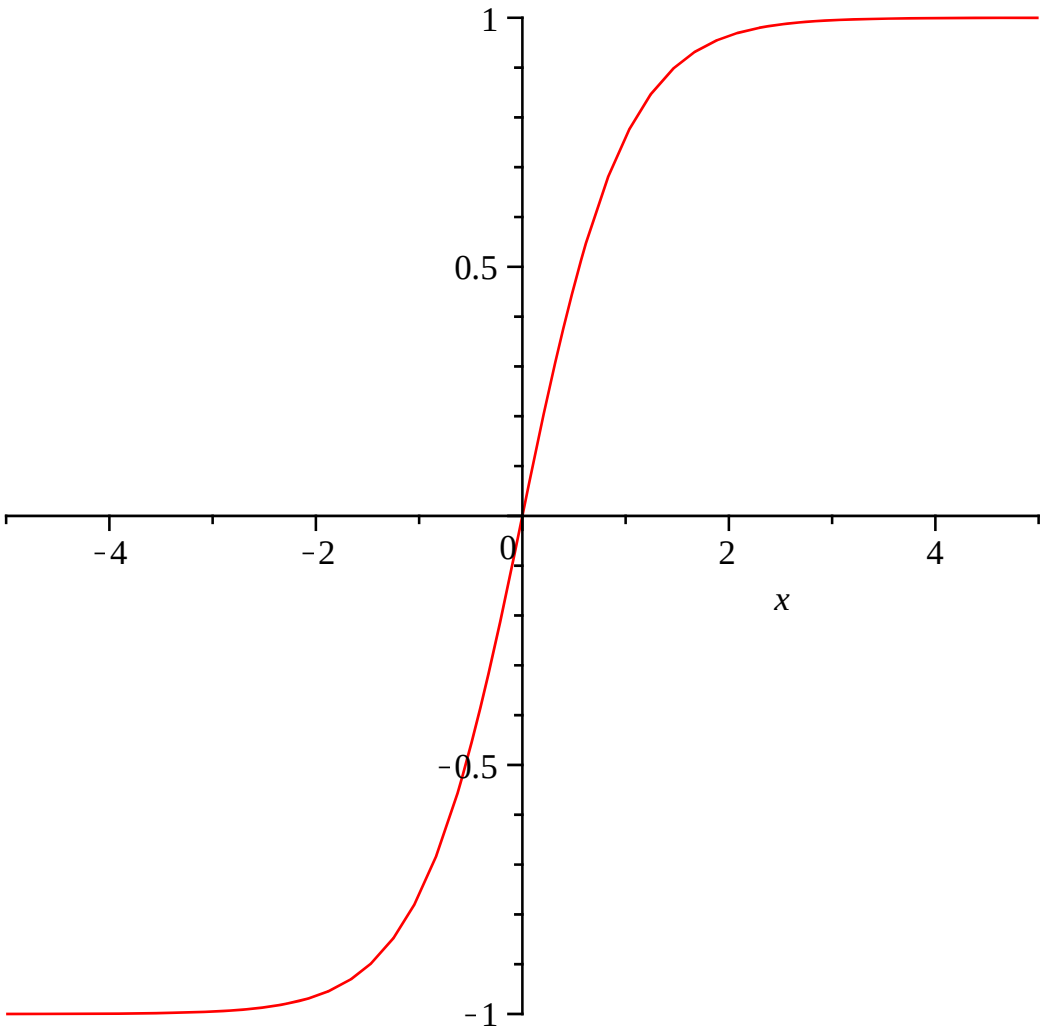
$$\frac{e^{x}}{2} + \frac{e^{-x}}{2} \tag{5.2.4}$$

双曲線正接・余接関数を定義し，x=-5..5 の範囲でプロットします.

```
> hyp3 := tanh(x);
```

$$hyp3 := \tanh(x) \tag{5.2.5}$$

```
> plot(hyp3, x=-5..5);
```



指数関数の形に変換します.

```
> convert(hyp3, exp);
```

$$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad \textbf{(5.2.6)}$$

指数と指数関数

指数の定義

初期化します.

```
> retart;
```

$$retart \qquad \textbf{(5.3.1.1)}$$

```
> exp1 := a^(0);
```

$$exp1 := 1 \qquad \textbf{(5.3.1.2)}$$

```
> exp2 := a^n;
```

$$exp2 := a^{n} \qquad \textbf{(5.3.1.3)}$$

```
> exp3 := a^(-n);
```

$$exp3 := a^{-n} \qquad \textbf{(5.3.1.4)}$$

```
> expand(exp3);
```

$$\frac{1}{a^n} \quad (5.3.1.5)$$

$a > 0$で，m が整数，n が 2 以上の整数のとき，

```
> a^(1/n);
```

$$a^{\frac{1}{n}} \quad (5.3.1.6)$$

初期化します.

```
> restart;
> a^p * a^q;
```

$$a^p a^q \quad (5.3.1.7)$$

```
> simplify(a^p * a^q);
```

$$a^{p+q} \quad (5.3.1.8)$$

```
> (a^p)^q;
```

$$(a^p)^q \quad (5.3.1.9)$$

```
> simplify((a^p)^q);
```

$$(a^p)^q \quad (5.3.1.10)$$

指数関数

a を底とする指数関数（ただし，$a \neq 1$ かつ$a > 0$）

```
> eq := y = a^x;
```

$$eq := y = a^x \quad (5.3.2.1)$$

対数と対数関数

対数の底

```
> eq := a^m = N;
```

$$eq := a^m = N \quad (5.4.1.1)$$

```
> solve(eq, m);
```

$$\frac{\ln(N)}{\ln(a)} \quad (5.4.1.2)$$

```
> lg := m = log[a](N);
```

$$lg := m = \frac{\ln(N)}{\ln(a)} \quad (5.4.1.3)$$

常用対数

```
> lg10 := log[10](N);
```

$$lg10 := \frac{\ln(N)}{\ln(10)} \quad (5.4.2.1)$$

対数関数

```
> y := log[a](x);
```

(5.4.3.1)

$$y := \frac{\ln(x)}{\ln(a)} \quad \textbf{(5.4.3.1)}$$

```
> y1 := eval(y, [a=2]);
```

$$y1 := \frac{\ln(x)}{\ln(2)} \quad \textbf{(5.4.3.2)}$$

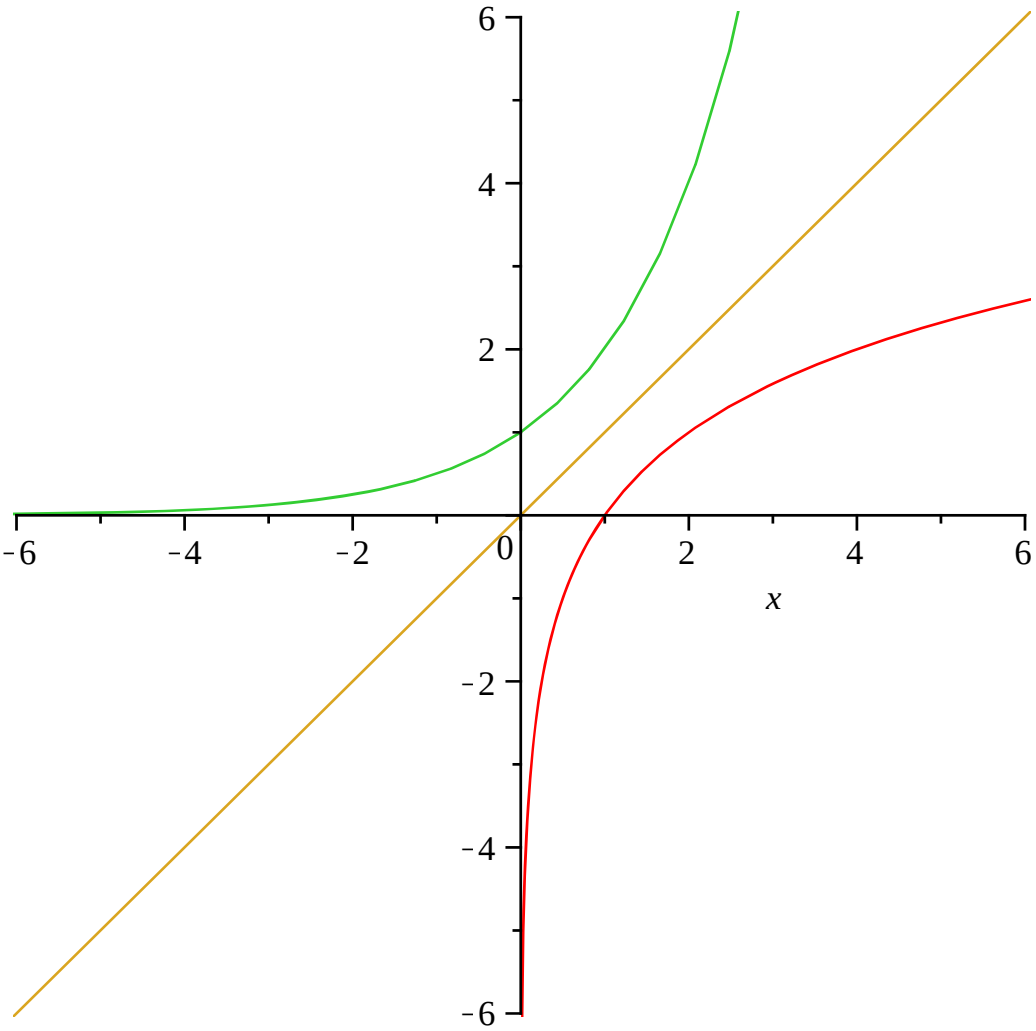
```
> y2 := 2^x;
```

$$y2 := 2^x \quad \textbf{(5.4.3.3)}$$

```
> y3 := x;
```

$$y3 := x \quad \textbf{(5.4.3.4)}$$

```
> plot( [y1, y2, y3], view=[-6..6, -6..6]);
```



自然対数

```
> Limit( (1+1/x)^x, x=+infinity );
```

$$\lim_{x \to \infty} \left(1 + \frac{1}{x}\right)^x \quad \textbf{(5.4.4.1)}$$

$\lim_{x \to \infty} \left(1 + \frac{1}{x}\right)^x$ は，ネイピア数 e になります．

自然対数の底として用いられます．

```
> limit( (1+1/x)^x, x=+infinity );
```

$$e \qquad \textbf{(5.4.4.2)}$$

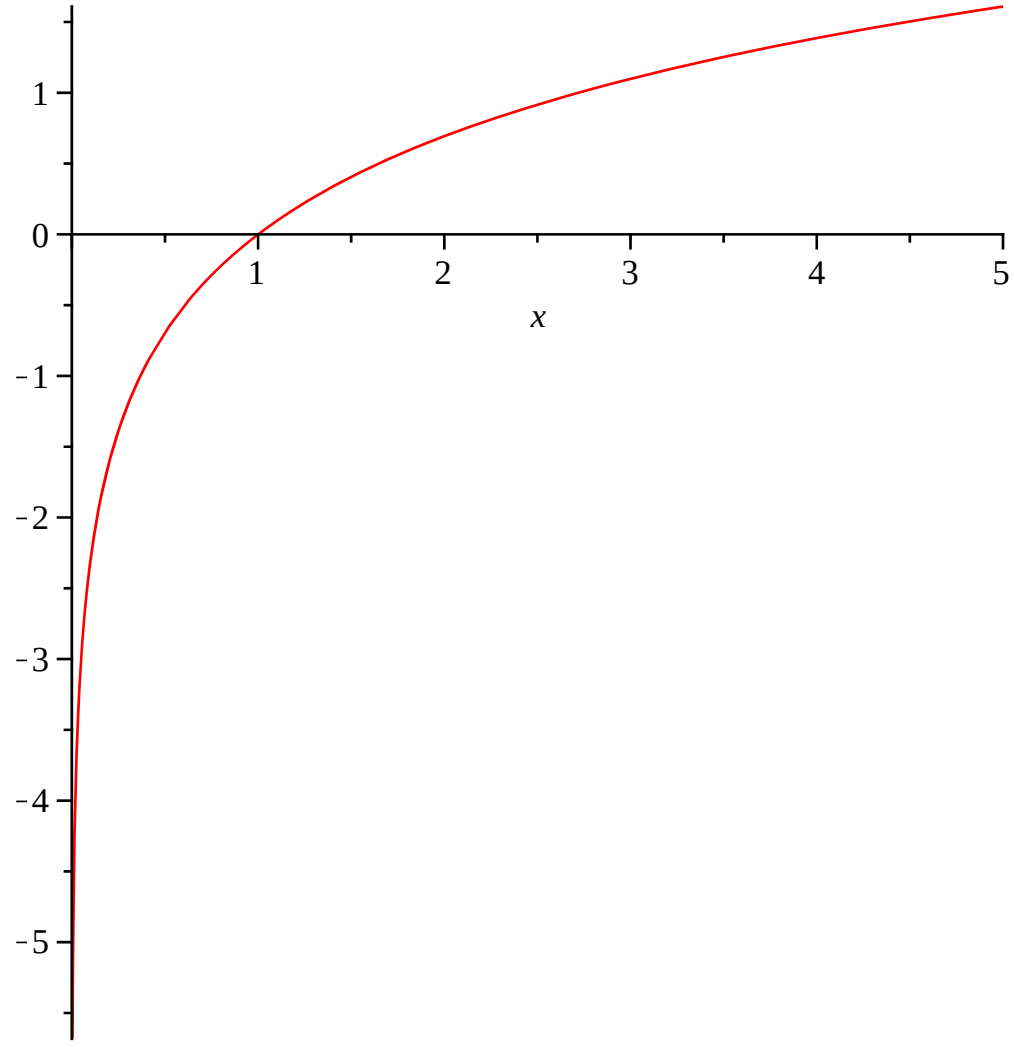
自然対数の定義

```
> ln(x);
```

$$\ln(x) \qquad \textbf{(5.4.4.3)}$$

自然対数を x=0..5 の範囲でプロットします.

```
> plot(ln(x), x=0..5);
```



ここで，x で 1 階微分します.

```
> diff( ln(x), x);
```

$$\frac{1}{x} \qquad \textbf{(5.4.4.4)}$$

さらに，1/x を x で積分してみます.

```
> int( 1/x, x );
```

$$\ln(x) \qquad \textbf{(5.4.4.5)}$$

区分関数

初期化します.

```
> restart;
```

以下のような変数によって異なる関数を持つ区分関数を作成します.

$$f := \begin{cases} -x & x \le -1 \\ x^2 & x \le 1 \\ \dfrac{\sin(x-1)}{x-1} & 1 < x \end{cases}$$

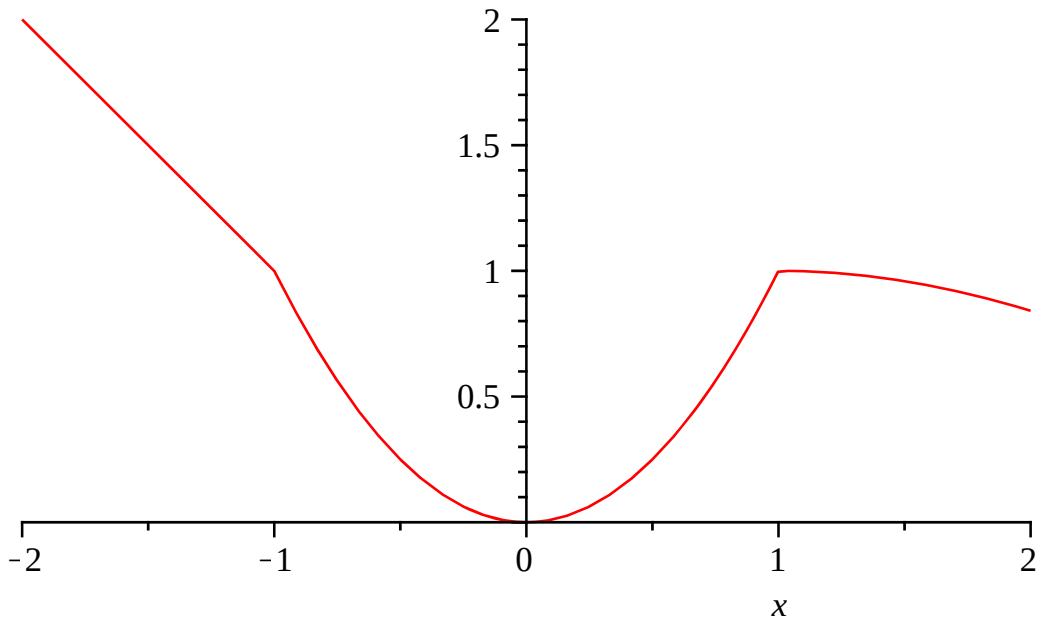
区分関数を作成するには，piecewise コマンドを使用します.

```
> f := piecewise( x <= -1, -x,
                  x <=  1, x*x,
                  1 <   x, sin(x-1)/(x-1) );
```

$$f := \begin{cases} -x & x \le -1 \\ x^2 & x \le 1 \\ \dfrac{\sin(x-1)}{x-1} & 1 < x \end{cases} \qquad \textbf{(5.5.1)}$$

区分関数をプロットします.

```
> plot(f, x = -2 .. 2, scaling=constrained);
```



x=-2 のときの関数の値になります.

```
> eval(f, x = -2);
```

$$2 \tag{5.5.2}$$

x=3 のときの関数の値になります.

```
> eval(f, x = 3);
```

$$\frac{\sin(2)}{2} \tag{5.5.3}$$

近似値を計算する場合，x = 3. と3 の後ろに小数点（ドット）を置きます.

```
> eval(f, x = 3.);
```

$$0.4546487134 \tag{5.5.4}$$

あるいは，x=3.0 を指定します.

```
> eval(f, x = 3.0);
```

$$0.4546487134 \tag{5.5.5}$$

極限を計算します.

```
> limit(f, x = 1);
```

$$1 \tag{5.5.6}$$

x で微分します.

```
> diff(f, x);
```

$$\begin{cases} -1 & x < -1 \\ undefined & x = -1 \\ 2\,x & x < 1 \\ undefined & x = 1 \\ \dfrac{\cos(x-1)}{x-1} - \dfrac{\sin(x-1)}{(x-1)^2} & 1 < x \end{cases} \tag{5.5.7}$$

undefined は，Maple の予約語のひとつになります．計算において，値が定まらなかったときに出力されます.

x について積分します（不定積分）.

```
> int(f, x);
```

$$\begin{cases} -\dfrac{x^2}{2} & x \le -1 \\ \dfrac{x^3}{3} - \dfrac{1}{6} & x \le 1 \\ \mathrm{Si}(x-1) + \dfrac{1}{6} & 1 < x \end{cases} \tag{5.5.8}$$

（参考）Si(x) は正弦積分を表します.

この積分は以下のようにすべての複素数 x に対して定義されます:

Si(x)　= int(sin(t)/t, t=0..x)

x について積分します（定積分）.

```
> int(f, x = -2 .. 3);
```

$$\frac{13}{6} + \mathrm{Si}(2) \tag{5.5.9}$$

文字列

通常，文字列は変数名などを作成するときに使用しますが，Maple では高度な処理を記述する際にも（そのほとんどはプログラミングの際に）様々な文字列の定義あるいは操作を行います．

ここでは，簡単な文字列の定義と操作を紹介します．

初期化します．

```
> restart;
```

文字列を定義するには，ダブルクォーテーション（ " " ）で文字列を括ります．

```
> st1 := "Hello World!";
```

$$st1 := \text{"Hello World!"} \tag{6.1}$$

文字列の結合には，cat コマンドを使用します．

```
> st2 := cat( "abc", "def" );
```

$$st2 := \text{"abcdef"} \tag{6.2}$$

変数名（名前）の定義にも cat コマンドを使用します．

```
> st3 := cat( f, n );
```

$$st3 := fn \tag{6.3}$$

文字列に含まれる部分文字列（文字列に一部）をパラメータにできます．

```
> i := 6;
```

$$i := 6 \tag{6.4}$$

```
> cat( "答えは ", i, " です．" );
```

$$\text{"答えは 6 です．"} \tag{6.5}$$

printf コマンドによって，文字列の出力を制御できます．%s の場所に文字列が入ります．¥n（もしくは \n）は改行になります．

```
> printf( "%s World!\n", "Hello" );
Hello World!
> printf( "%s World!\n\n", "Hello" );
Hello World!
```

配列

配列は，データ構造のひとつになります．また，多次元の配列を定義することができます．

```
> restart;
```

１次元配列

はじめに，１次元の配列を定義します．ひとつのインデックス（index）だけで，配列（データ構造）内の要素を指定することができます．インデックスは，データ構造内の要素を指定するための整数値です．

```
> Arr1 := Array([a, b, c, d, e, f]);
```

$$Arr1 := \left[\begin{array}{cccccc} a & b & c & d & e & f \end{array}\right] \quad (7.1.1)$$

インデックス１を指定します．

```
> Arr1[1];
```

$$a \quad (7.1.2)$$

インデックス６を指定します．

```
> Arr1[6];
```

$$f \quad (7.1.3)$$

インデックス７を指定します．

```
> Arr1[7];
```

```
Error, Array index out of range
```

Arr1 には６個までの要素しかないため，指定したインデックスでは範囲外のエラーになっていまいます．

２次元配列

次に，２次元の配列を定義します．２つのインデックス（index）で，配列（データ構造）内の要素を指定します．

```
> Arr2 := Array( [ [a, b, x], [c, d, y], [e, f, z] ] );
```

$$Arr2 := \begin{bmatrix} a & b & x \\ c & d & y \\ e & f & z \end{bmatrix} \quad \textbf{(7.2.1)}$$

	1	2	3
1	a	b	x
2	c	d	y
3	e	f	z

インデックス［１，１］の要素を指定します．

```
> Arr2[1,1];
```

$$a \quad \textbf{(7.2.2)}$$

インデックス［２，３］の要素を指定します．

```
> Arr2[2,3];
```

$$y \quad \textbf{(7.2.3)}$$

インデックス［３，４］の要素を指定します．

```
> Arr2[3, 4];
Error, Array index out of range
```

Arr2 は３×３のサイズを持つ２次元配列のため，［３，４］の要素はありません．

３次元配列

次に，３次元の配列を定義します．３つのインデックス（index）で，配列（データ構造）内の要素を指定します．

```
> Arr3 := Array( [ [ [a1,a2,a3], [b1,b2,b3], [x1,x2,x3] ],
                   [ [c1,c2,c3], [d1,d2,d3], [y1,y2,y3] ],
                   [ [e1,e2,e3], [f1,f2,f3], [z1,z2,z3] ] ] );
```

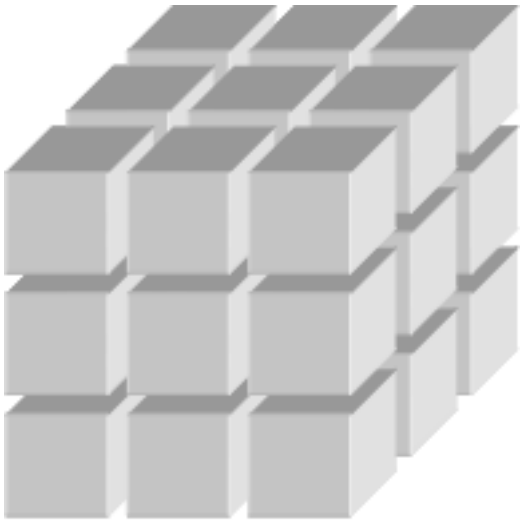
(7.3.1)

$$Arr3 := \begin{bmatrix} 1..3\ x\ 1..3\ x\ 1..3\ Array \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \quad \textbf{(7.3.1)}$$

（参考）
配列の構造や大きさによって，以下のような表示になる場合があります．内容を確認したい場合は，表示をダブルクリックします．詳細は，rtable コマンドのヘルプを参照してください．

$$\begin{bmatrix} 1..3\ x\ 1..3\ x\ 1..3\ Array \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix}$$

（参考） ３次元配列のイメージ



インデックス［１，１，１］の要素を指定します．

```
> Arr3[1,1,1];
```

$$a1 \quad \textbf{(7.3.2)}$$

インデックス［２，３，２］の要素を指定します．

```
> Arr3[2,3,2];
```

$$y2 \quad \textbf{(7.3.3)}$$

２次元配列から４次元配列への拡張

２次元配列

```
> Arr2a := Array([[a,b],[c,d]]);
```

$$Arr2a := \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \textbf{(7.4.1)}$$

```
> Arr2a[1,2];
```

$$b \quad \textbf{(7.4.2)}$$

3 次元配列

```
> Arr3a := Array([[[a1,a2],[b1,b2]],[[c1,c2],[d1,d2]]]);
```

$$Arr3a := \begin{bmatrix} 1..2\ x\ 1..2\ x\ 1..2\ Array \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \quad \textbf{(7.4.3)}$$

```
> Arr3a[2, 1, 2];
```

$$c2 \quad \textbf{(7.4.4)}$$

4 次元配列

```
> Arr4a:=Array([[[[a11,a12], [a21,a22]], [[b11,b12],[b21,b22]]], [[
  [c11,c12],[c21,c22]], [[d11,d12],[d21,d22]]]]);
```

$$Arr4a := \begin{bmatrix} 1..2\ x\ 1..2\ x\ 1..2\ x\ 1..2\ Array \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \quad \textbf{(7.4.5)}$$

```
> Arr4 := [1,2,2,1];
```

$$Arr4 := [1, 2, 2, 1] \quad \textbf{(7.4.6)}$$

シーケンス（数列，式列）

初期化します．

```
> restart;
```

シーケンスの定義

数列

```
> a := 1, 2, 3, 4, 5, 6;
```

$$a := 1, 2, 3, 4, 5, 6 \quad \textbf{(8.1.1)}$$

```
> b := seq( n, n=1..6 );
```

$$b := 1, 2, 3, 4, 5, 6 \quad \textbf{(8.1.2)}$$

式列

```
> c := x=1.0, y=1.3, z=-0.6;
```

$$c := x = 1.0, y = 1.3, z = -0.6 \quad \textbf{(8.1.3)}$$

```
> d := seq( eq[i]=(x+y)^i, i=1..3 );
```

$$d := eq_1 = x + y, eq_2 = (x + y)^2, eq_3 = (x + y)^3 \quad \textbf{(8.1.4)}$$

要素の指定

数列を定義します．

```
> sq1 := seq( n^2, n=1..5 );
```

$$sq1 := 1, 4, 9, 16, 25 \quad \textbf{(8.2.1)}$$

要素を角括弧 [ ] とインデックスを用いて指定します．

```
> sq1[1];
```

$$1 \quad \textbf{(8.2.2)}$$

```
> sq1[3];
```

$$9 \quad \textbf{(8.2.3)}$$

```
> sq1[5];
```

$$25 \quad \textbf{(8.2.4)}$$

その他の数列，式列，データの定義例

```
> sq2 := seq( sin( Pi*i/6 ), i=0..6 );
```

$$sq2 := 0, \frac{1}{2}, \frac{\sqrt{3}}{2}, 1, \frac{\sqrt{3}}{2}, \frac{1}{2}, 0 \quad \textbf{(8.3.1)}$$

```
> sq3 := seq( (x+y)^i, i=1..3 );
```

$$sq3 := x + y, (x + y)^2, (x + y)^3 \quad \textbf{(8.3.2)}$$

expand コマンドの追加

```
> sq4 := seq( expand( (x+y)^i ), i=1..3 );
```

$$sq4 := x + y, x^2 + 2\,x\,y + y^2, x^3 + 3\,x^2\,y + 3\,x\,y^2 + y^3 \quad \textbf{(8.3.3)}$$

範囲（ 0..1 ）と刻み幅（ 0.1 ）の指定方法

```
> sq5 := seq( t, t=0..1, 0.1 );
```

$$sq5 := 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 \quad \textbf{(8.3.4)}$$

リスト

初期化します．

```
> restart;
```

リストの定義

シーケンスを各括弧 [　] で括ると，リストになります．リストは，様々な場面で利用されます．

```
> lst := [1, 2, 3, 2, 1, 4, 1, 1, 2, 5, 6, 2, 3, 4];
```

$$lst := [1, 2, 3, 2, 1, 4, 1, 1, 2, 5, 6, 2, 3, 4] \quad \textbf{(9.1.1)}$$

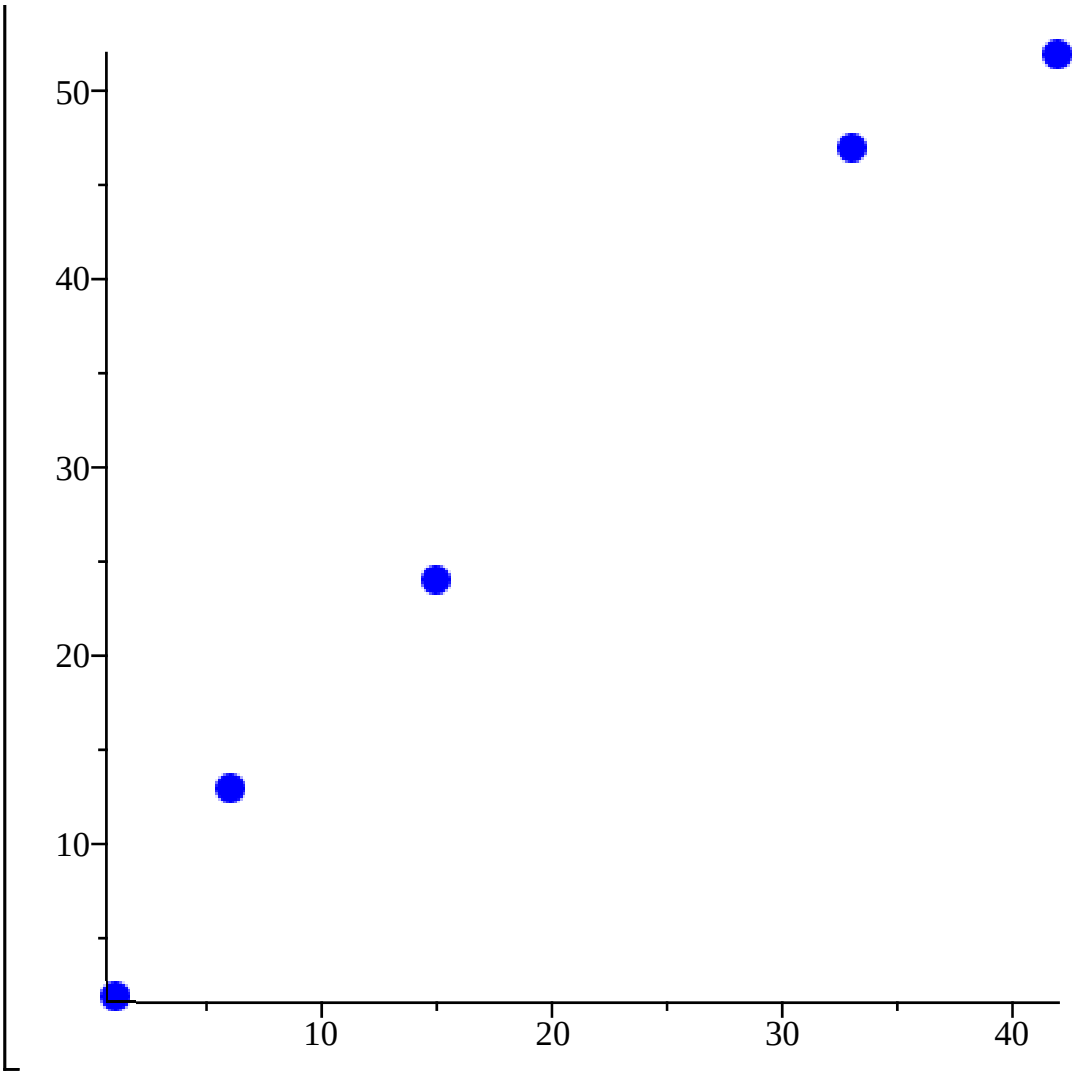
要素の順番が保持されます．また，重複する要素も省略されません．

リストの中にリストを要素として定義することが出来ます．通常，リスト・リストを呼びます．

```
> lstlst := [[1,2], [6,13], [15, 24], [33,47], [42,52]];
```

$$lstlst := [[1, 2], [6, 13], [15, 24], [33, 47], [42, 52]] \quad \textbf{(9.1.2)}$$

```
> plots[pointplot]( lstlst, symbol=solidcircle, symbolsize=24, color=
  blue );
```



要素の抽出

インデックスで指定します.

```
> lst[2];
```

2 **(9.2.1)**

```
> lstlst[2];
```

[6, 13] **(9.2.2)**

2 つのインデックスで指定します.

```
> lstlst[2][1];
  lstlst[2][2];
```

6
13 **(9.2.3)**

シーケンスのリスト

seq コマンドを用いて，擬似データを作成します.

```
> dat1 := seq( [t, sin(t)], t=0..5, 0.1 );
```

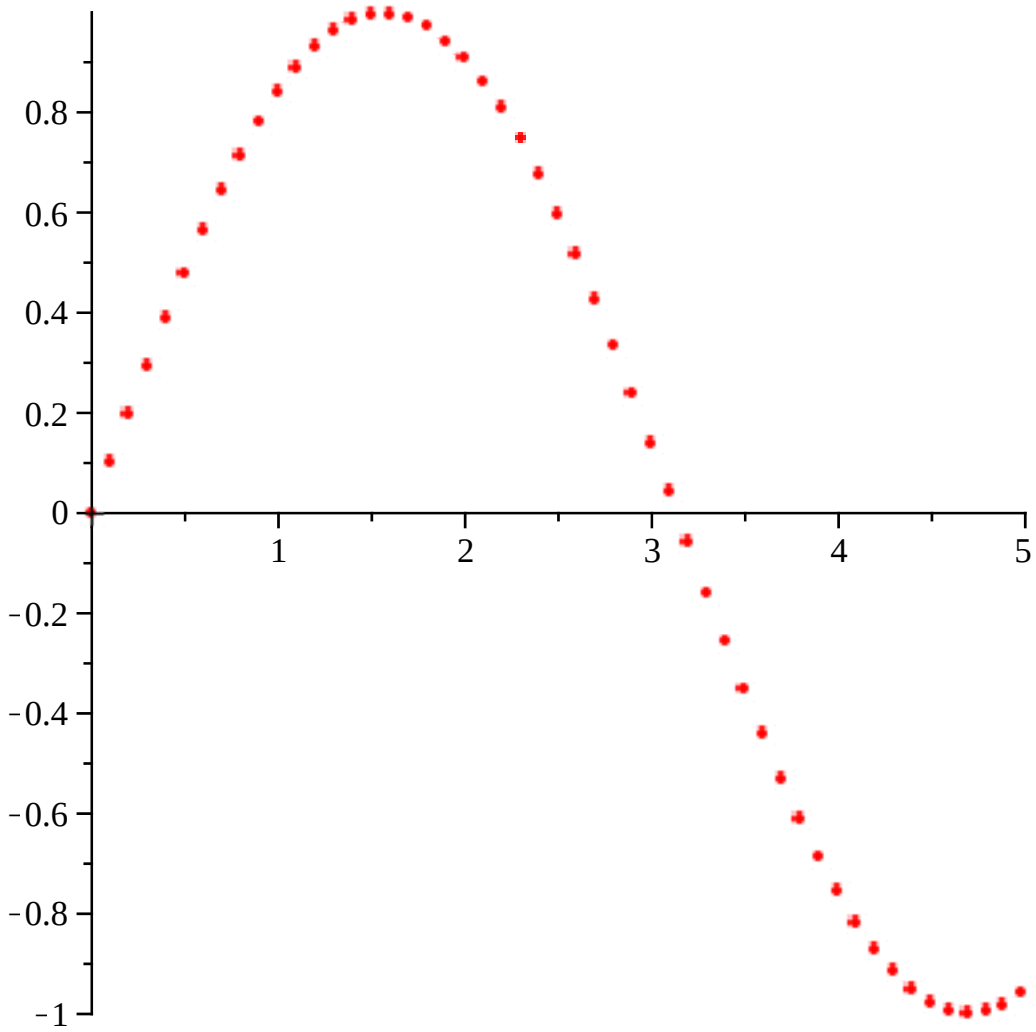
$$dat1 := [0, 0], [0.1, 0.09983341665], [0.2, 0.1986693308], [0.3, 0.2955202067], [0.4, 0.3894183423], [0.5, 0.4794255386], [0.6, 0.5646424734], [0.7, 0.6442176872], [0.8, 0.7173560909], [0.9, 0.7833269096], [1.0, 0.8414709848], [1.1, 0.8912073601], [1.2, 0.9320390860], [1.3, 0.9635581854], [1.4, 0.9854497300], [1.5, 0.9974949866], [1.6, 0.9995736030], [1.7, 0.9916648105], [1.8, 0.9738476309], [1.9, 0.9463000877], [2.0, 0.9092974268], [2.1, 0.8632093666], [2.2, 0.8084964038], [2.3, 0.7457052122], [2.4, 0.6754631806], [2.5, 0.5984721441], [2.6, 0.5155013718], [2.7, 0.4273798802], [2.8, 0.3349881502], [2.9, 0.2392493292], [3.0, 0.1411200081], [3.1, 0.04158066243], [3.2, −0.05837414343], [3.3, −0.1577456941], [3.4, −0.2555411020], [3.5, −0.3507832277], [3.6, −0.4425204433], [3.7, −0.5298361409], [3.8, −0.6118578909], [3.9, −0.6877661592], [4.0, −0.7568024953], [4.1, −0.8182771111], [4.2, −0.8715757724], [4.3, −0.9161659367], [4.4, −0.9516020739], [4.5, −0.9775301177], [4.6, −0.9936910036], [4.7, −0.9999232576], [4.8, −0.9961646088], [4.9, −0.9824526126], [5.0, −0.9589242747] \quad \textbf{(9.3.1)}$$

作成したデータをプロットします.

```
> plots[pointplot]( dat1, symbol=solidcircle, color=red );
Error, (in plot/options2d) unexpected options: [[.3, .2955202067],
[.4, .3894183423], [.5, .4794255386], [.6, .5646424734], [.7,
.6442176872], [.8, .7173560909], [.9, .7833269096], [1.0,
.8414709848], [1.1, .8912073601], [1.2, .9320390860], [1.3,
.9635581854], [1.4, .9854497300], [1.5, .9974949866], [1.6,
.9995736030], [1.7, .9916648105], [1.8, .9738476309], [1.9,
.9463000877], [2.0, .9092974268], [2.1, .8632093666], [2.2,
.8084964038], [2.3, .7457052122], [2.4, .6754631806], [2.5,
.5984721441], [2.6, .5155013718], [2.7, .4273798802], [2.8,
.3349881502], [2.9, .2392493292], [3.0, .1411200081], [3.1,
0.4158066243e-1], [3.2, -0.5837414343e-1],...
```

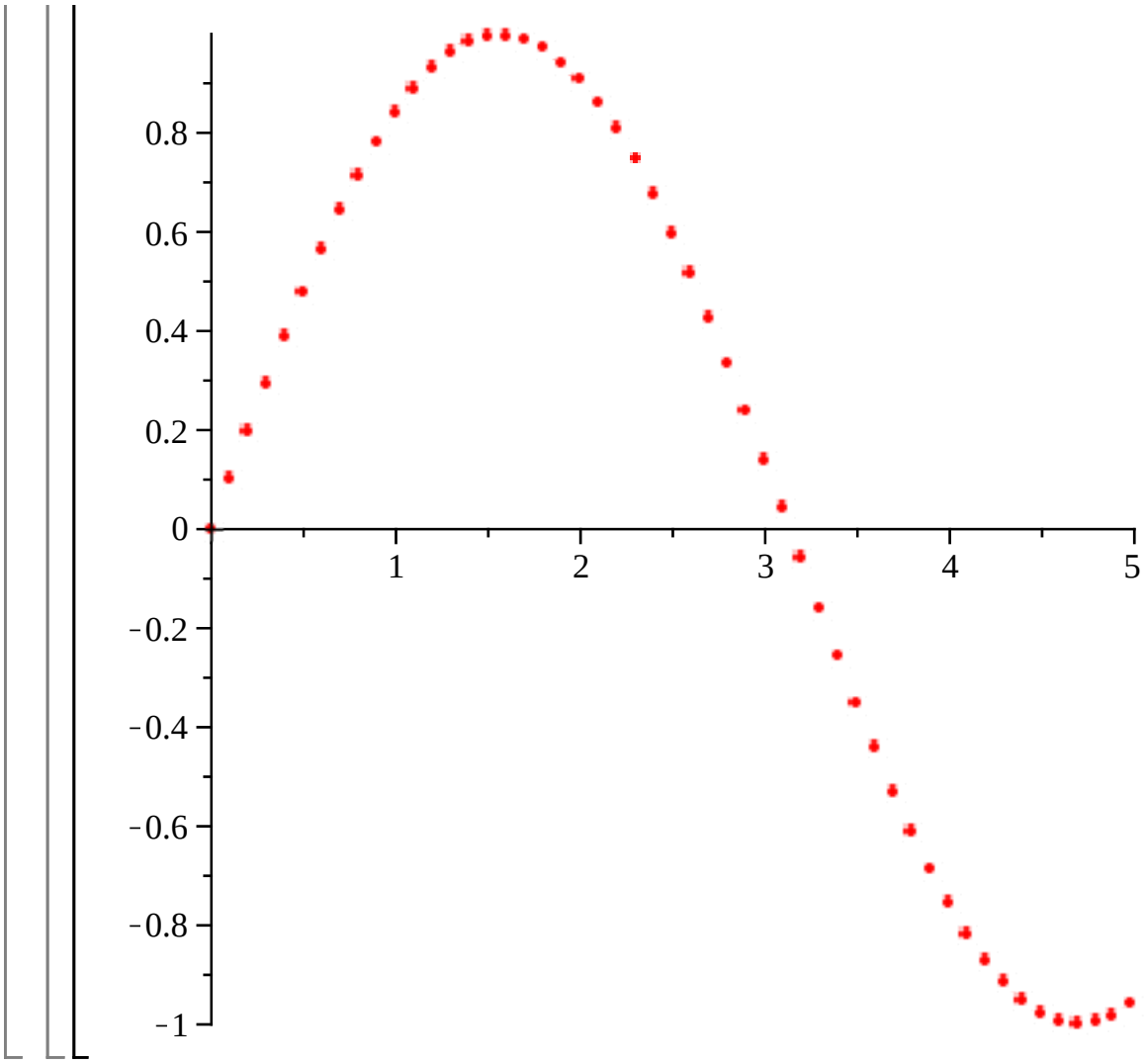
引数はリスト・リストである必要があります.

```
> plots[pointplot]( [ dat1 ], symbol=solidcircle, color=red );
```



あらかじめ，シーケンスを [　] で括ります．

```
> dat2 := [seq( [t, sin(t)], t=0..5, 0.1 )]: # 非表示
> plots[pointplot]( dat2, symbol=solidcircle, color=red );
```


0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
1
2
3
4
5

行列

初期化します.

```
> restart;
```

通常，行列の定義には Matrix コマンドを使用します.

```
> M1 := Matrix( [ [a,b],[c,d] ] );
```

$$M1 := \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{10.1}$$

< > を用いて，行列を定義することもできます.

```
> M2 := < <a,b> | <c,d> >;
```

$$M2 := \begin{bmatrix} a & c \\ b & d \end{bmatrix} \tag{10.2}$$

行列のサイズと要素を指定することで，行列を定義します．n×n の行列の場合，n^2 個の要素が必要になります．以下のように，３×３ の行列の場合，９の要素が必要になります.

```
> M3 := Matrix( 3,3, [1, 2, 3, 4, 5, 6, 7, 8, 9]  );
```

$$M3 := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{10.3}$$

ベクトル

通常，ベクトルの定義には Vector コマンドを使用します．以下の定義は，列ベクトルになります．

```
> V1 := Vector( [a, b, c] );
```

$$V1 := \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \textbf{(11.1)}$$

行ベクトルを定義する場合，Vector**[row]**( ) を使用します．

```
> V2 := Vector[row]([ x, y, z ]);
```

$$V2 := \begin{bmatrix} x & y & z \end{bmatrix} \quad \textbf{(11.2)}$$

行列と同様に < > でベクトルも定義できます．要素を横に並べて，行ベクトルを定義する場合，バーティカルバー（|）で要素を区切ります．

```
> V3 := < i|j|k >;
```

$$V3 := \begin{bmatrix} i & j & k \end{bmatrix} \quad \textbf{(11.3)}$$

要素を縦に並べて，列ベクトルを定義する場合，カンマ（, ）で要素を区切ります．

```
> V4 := < s,t,u >;
```

$$V4 := \begin{bmatrix} s \\ t \\ u \end{bmatrix} \quad \textbf{(11.4)}$$

（参考）内積には，ドット（. ）を使用します．

```
> V5 := V3.V4;
```

$$V5 := s\,i + t\,j + u\,k \quad \textbf{(11.5)}$$

集合（セット）

シーケンスを波括弧 { } で括ると，集合になります．集合は，様々な場面で利用されます．

（注意）集合は，重複する要素を省き，順番を保持しません．

初期化します．

```
> restart;
> st := {1, 2, 3, 2, 1, 4, 1, 1, 2, 5, 6, 2, 3, 4};
```

$$st := \{1, 2, 3, 4, 5, 6\} \quad \textbf{(12.1)}$$

そのため，要素を指定する場合，要素数に気をつける必要があります．

```
> st[6];
```

$$6 \quad \textbf{(12.2)}$$

```
> st[7];
Error, invalid subscript selector
```

集合 $st := \{1, 2, 3, 4, 5, 6\}$ に７番目の要素はありません．

リストの場合，重複する要素は省かれず，順番は保持されます．

```
> lt := [1, 2, 3, 2, 1, 4, 1, 1, 2, 5, 6, 2, 3, 4];
```

$$lt := [1, 2, 3, 2, 1, 4, 1, 1, 2, 5, 6, 2, 3, 4] \quad \textbf{(12.3)}$$

当然，７番目の要素は存在します．

```
> lt[7];
```

$$1 \quad \textbf{(12.4)}$$

ただし，存在しない要素数を指定した場合，集合と同様にエラーメッセージが表示されます．

```
> lt[16];
Error, invalid subscript selector
```

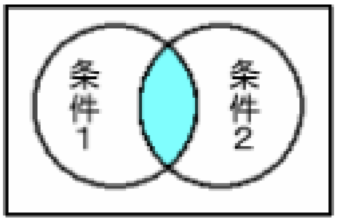
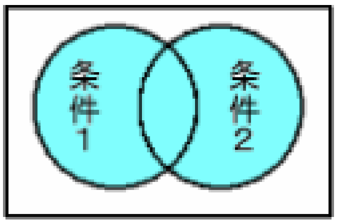
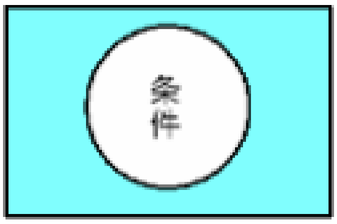
ブーリアン

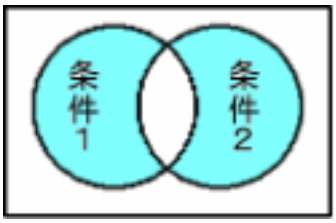
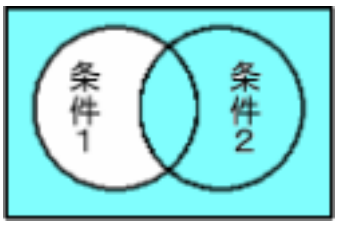
ブーリアン演算をします．true（真） と false（偽） の2値で演算をします．Mapleは第3の値としてFAILを持っています．これは，true および false のどちらにも定まらないときの値になります．

ベン図

```
                 and                            or              not
              (論理積)                        (論理和)          (論理否定)
       true   false   FAIL         true   false   FAIL
       -----  -----   -----        -----  -----   -----
true   true   false   FAIL         true   true    true     false
false  false  false   false        true   false   FAIL     true
FAIL   FAIL   false   FAIL         true   FAIL    FAIL     FAIL

                 xor                          implies
            (排他的論理和)                     (含意)
       true   false   FAIL         true   false   FAIL
       -----  -----   -----        -----  -----   -----
true   false  true    FAIL         true   false   FAIL
false  true   false   FAIL         true   true    true
FAIL   FAIL   FAIL    FAIL         true   FAIL    FAIL
```

（Maple 13 - boolean のヘルプより）

and （論理積） 条件1を満たし，かつ条件2を満たす	
or （論理和） 条件1を満たすか，または条件2を満たす	
not （論理否定） 条件でない	
xor （排他的論理和） 条件1を満たし，かつ条件2を満たさない または，	

条件２を満たし，かつ条件１を満たさない	
implies （論理含意，論理包含演算） 条件１を満たさない，または条件２を満たす	

論理演算

c1 を条件１，c2 を条件２として，演算をします．

```
> c1tr := true; # 条件１を満たす
```

$$c1tr := true \tag{13.2.1}$$

```
> c2tr := true; # 条件２を満たす
```

$$c2tr := true \tag{13.2.2}$$

```
> c1fl := false; # 条件１を満たさない
```

$$c1fl := false \tag{13.2.3}$$

```
> c2fl := false; # 条件２を満たさない
```

$$c2fl := false \tag{13.2.4}$$

論理積

条件１を満たし，かつ条件２を満たす

```
> c1tr and c2tr;
```

$$true \tag{13.2.5}$$

論理和

条件１を満たすか，または条件２を満たす

```
> c1tr or c2tr;
```

$$true \tag{13.2.6}$$

論理和と論理積

```
> ( c1tr or c2tr ) and c1fl;
```

$$false \tag{13.2.7}$$

論理和と論理否定の組合せ

```
> c1tr and c2tr and c1fl;
```

$$\mathit{false} \qquad (13.2.8)$$

```
> c1tr and (not c1fl) and c2tr;
```

$$\mathit{true} \qquad (13.2.9)$$

不等式 1 < 2 を c として定義します.

```
> c := 1 < 2;
```

$$c := 1 < 2 \qquad (13.2.10)$$

evalb を用いて不等式の真・偽を評価します.

```
> evalb( c );
```

$$\mathit{true} \qquad (13.2.11)$$

evalb で以下のような評価が可能です.

```
> evalb( 1=1 );
```

$$\mathit{true} \qquad (13.2.12)$$

```
> evalb( 1=2 );
```

$$\mathit{false} \qquad (13.2.13)$$

```
> x := 2;
  y := 3;
  evalb( x<y );
```

$$x := 2$$

$$y := 3$$

$$\mathit{true} \qquad (13.2.14)$$

型（type）

定義されるオブジェクトには型があります．
コマンド（あるいはコマンドの引数に）によって，要求される型が異なります．

型を調べるには，whattype コマンドを使用します．

Maple オブジェクトの型（type）

整数型

```
> whattype( 1 );
```

integer **(14.1.1)**

浮動小数点型

```
> whattype( 1.0 );
```

float **(14.1.2)**

シンボル（記号）型

```
> whattype( a );
```

symbol **(14.1.3)**

複素数型

```
> whattype( Complex(1,2) );
```

complex(*extended_numeric*) **(14.1.4)**

和型

```
> whattype( x^2 + x + 1 );
```

integer **(14.1.5)**

積型

```
> whattype( a*x );
```

`*` **(14.1.6)**

累乗型

```
> whattype( x^2 );
```

integer **(14.1.7)**

```
> whattype( sqrt(2) );
```

(14.1.8)

`^` **(14.1.8)**

方程式型

```
> whattype( x^2 + x + 1 = 0);
```

`=` **(14.1.9)**

関数型

```
> whattype( sin(x) );
```

function **(14.1.10)**

```
> whattype( f(x) );
```

function **(14.1.11)**

文字型

```
> whattype( "文字" );
```

string **(14.1.12)**

配列型

```
> whattype( Array(1..2,1..2) );
```

Array **(14.1.13)**

シーケンス型

```
> whattype( seq(i, i=1..3) );
```

exprseq **(14.1.14)**

リスト型

```
> whattype( [seq(i, i=1..3)] );
```

list **(14.1.15)**

行列型

```
> whattype( Matrix(3,3) );
```

Matrix **(14.1.16)**

ベクトル型

```
> whattype( Vector([1,2,3]) );
```

$Vector_{column}$ **(14.1.17)**

集合型

```
> whattype( {seq(i, i=1..3)} );
```

set **(14.1.18)**

変数型の仮定

初期化します.

```
> restart;
```

変数 a の型を正の整数（positive integer, posint）と仮定する場合，assume コマンドを使用します.

```
> assume(a, posint);
> a;
```

$$a\sim \tag{14.2.1}$$

```
> sqrt(a);
```

$$\sqrt{a\sim} \tag{14.2.2}$$

変数の仮定を行うと，変数名にチルダ（～）が追加されます．変数名の仮定内容を調べる場合，about コマンドを使用します.

```
> about(a);
```

```
Originally a, renamed a~:
  is assumed to be: AndProp(integer,RealRange(1,infinity))
```

型や正負が仮定されていない x^2 の平方根を計算します.

```
> sqrt(x^2);
```

$$\sqrt{x^2} \tag{14.2.3}$$

もし x が正の実数と仮定した場合，以下のような結果になります.

（参考） assuming によって，一時的に仮定を設けることが出来ます.

```
> sqrt(x^2) assuming x>0;
```

$$x \tag{14.2.4}$$

もし x が負の実数と仮定した場合，以下のような結果になります.

```
> sqrt(x^2) assuming x<0;
```

$$-x \tag{14.2.5}$$

何も仮定をせず，平方根をはずそうとすると，以下のような結果になります.

```
> simplify(sqrt(x^2));
```

$$\mathrm{csgn}(x)\, x \tag{14.2.6}$$

csgn コマンドは，符号関数のひとつです．入力（複素数も含む）の正負を判断し，正の場合は 1

を，負の場合は－１を返します．

$$\mathrm{csgn}(x) = \begin{cases} 1 & 0 < \Re(x) \text{ or } \Re(x) = 0 \text{ and } 0 < \Im(x) \\ -1 & \Re(x) < 0 \text{ or } \Re(x) = 0 \text{ and } \Im(x) < 0 \end{cases}$$

（Maple 13 のヘルプから）

```
> csgn(5);
```

$$1 \qquad (14.2.7)$$

```
> csgn(-2.1);
```

$$-1 \qquad (14.2.8)$$

入力として，複素数も使用できます．

```
> csgn( +1 +2*I ); # 0 < Re(x)   and   0 < Im(x)
```

$$1 \qquad (14.2.9)$$

```
> csgn( +1 -2*I ); # 0 < Re(x)   and   0 > Im(x)
```

$$1 \qquad (14.2.10)$$

```
> csgn( -1 +2*I ); # 0 > Re(x)   and   0 < Im(x)
```

$$-1 \qquad (14.2.11)$$

```
> csgn( -1 -2*I ); # 0 > Re(x)   and   0 > Im(x)
```

$$-1 \qquad (14.2.12)$$

（参考）restart コマンドにより，仮定も初期化されます．

主な仮定

初期化します.

```
> restart;
```

偶数

```
> assume(a, even); about(a);
Originally a, renamed a~:
  is assumed to be: LinearProp(2,integer,0)
```

```
> sqrt(a) assuming a::even;
```

$$\sqrt{a\sim} \tag{14.3.1}$$

奇数

```
> assume(b, odd); about(b);
Originally b, renamed b~:
  is assumed to be: LinearProp(2,integer,1)
```

```
> sqrt(b) assuming b::odd;
```

$$\sqrt{b\sim} \tag{14.3.2}$$

実数

```
> assume(c, real); about(c);
Originally c, renamed c~:
  is assumed to be: real
```

```
> sqrt(c) assuming c::real;
```

$$\sqrt{c\sim} \tag{14.3.3}$$

正の実数

```
> assume(d, positive); about(d);
Originally d, renamed d~:
  is assumed to be: RealRange(Open(0),infinity)
```

```
> sqrt(d) assuming d::positive;
```

$$\sqrt{d\sim} \tag{14.3.4}$$

負の実数

```
> assume(e, negative); about(e);
Originally e, renamed e~:
  is assumed to be: RealRange(-infinity,Open(0))
```

```
> sqrt(e) assuming e::negative;
```

(14.3.5)

$$I\sqrt{-e\sim} \tag{14.3.5}$$

```
> %^2;
```

$$e\sim \tag{14.3.6}$$

整数

```
> assume(f, integer); about(f);
Originally f, renamed f~:
  is assumed to be: integer
```

```
> sqrt(f) assuming f::integer;
```

$$\sqrt{f\sim} \tag{14.3.7}$$

正の整数

```
> assume(g, posint); about(g); # POSitive INTegr
Originally g, renamed g~:
  is assumed to be: AndProp(integer,RealRange(1,infinity))
```

```
> sqrt(g) assuming g::posint;
```

$$\sqrt{g\sim} \tag{14.3.8}$$

負の整数

```
> assume(h, negint); about(h); # NEGative INTegr
Originally h, renamed h~:
  is assumed to be: AndProp(integer,RealRange(-infinity,-1))
```

```
> sqrt(i) assuming i::negint;
```

$$I\sqrt{-i} \tag{14.3.9}$$

一つ前の結果 **(14.3.9)** を二乗します.

```
> %^2;
```

$$i \tag{14.3.10}$$

（参考） e を負の実数として，sqrt(e) を計算し，その二乗を取ります.

```
> sqrt(e) assuming e::negative;
> %^2;
```

$$I\sqrt{-e\sim}$$
$$e\sim \tag{14.3.11}$$

仮定の解除（キャンセル）

restart コマンドを使用する以外に，unassign コマンドを使用する方法があります.

変数 var を負の実数として仮定し平方根を計算します.

```
> assume(var, negative);
> sqrt(var);
```

$$I\sqrt{-var\sim} \tag{14.4.1}$$

restart コマンドを実行します.

```
> restart;
```

確認のため，sqrt(var) を実行します.

```
> sqrt(var);
```

$$\sqrt{var} \tag{14.4.2}$$

再度，var を 負の実数として仮定し平方根を計算します.

```
> assume(var, negative);
> sqrt(var);
```

$$I\sqrt{-var\sim} \tag{14.4.3}$$

unassign コマンドを用いて，仮定を解除します.

```
> unassign('var');
```

確認のため，sqrt(var) を実行します.

```
> sqrt(var);
```

$$\sqrt{var} \tag{14.4.4}$$

ステップ・バイ・ステップ式

はじめての Maple

基礎編

Keywords & Key commands

Maple を使う前に

Maple の起動

スタンダードインターフェイス
クラシックインターフェイス
コマンドラインインターフェイス

編集モード

ワークシートモード
ドキュメントモード

ワークシートモード

`>` プロンプト
`/` スラッシュ
`^` キャレット
`F5` キー

ドキュメントモード

入力方法と編集モードの設定

インターフェースの構成

一般的なインターフェースの種類

▼ グラフィカルユーザインターフェース

▼ コマンドインターフェース

▼ Maple のユーザインターフェース

コンテキストメニュー
パレット
ワークスペース

▼ 入力方法

▼ グループ

コマンド
テキスト
左側の角括弧　[
実行グループ
テキストグループ
F9 キー

▼ 結果の表示・非表示

;　セミコロン
:　コロン

▼ コマンドの実行と入力

>　プロンプト
( )　丸括弧
引数
コマンド名(引数);
Enter キー
sin()
diff()
オプション
コマンド名(引数, オプション);
プロット
plot()
..　コンマ・コンマ（範囲の指定）
Pi
円周率 π （3.141592654.....）
*　アスタリスク
plot(color)
sqrt()
exp()
limit()
infinity

▼ テキスト入力（Maple Input, 1D Math）

int()
ラベル
ラベルの丸括弧 ( )
() コマンド内の丸括弧
[] 角括弧
{} 波括弧
, コンマ
eval()

▼ 数式入力（2D Math Input）

パレット
テンプレート

▼ パッケージのロード

基本コマンド
応用コマンド
パッケージロード
初期化
`restart`
`with()`
`plots` パッケージ
アニメーション
`animate()`
`unwith()`
パッケージ名［コマンド名］（引数，オプション）
`plots[animate]()`

▼ 入力方法の混在

`F5` キー

▼ テキスト入力と数式入力を切り替え

▼ 入力形式とカーソル

テキスト入力
カーソル
`|`　バーティカルバー
数式入力
`/`　スラッシュ

デバッギング

エラーメッセージ（Error）
警告メッセージ（Warning）
;　セミコロン
Web ヘルプ

コメントの追加

#　シャープ

コマンド内の改行

改行
Shift キー + Enter キー

テキスト入力・セミコロン（;）なし

テキスト入力・セミコロン（;）あり

テキスト入力・コロン（:）あり

数式入力・セミコロン（;）なし

数式入力・セミコロン（;）あり

数式入力・コロン（:）あり

▼ ファイルの「保存」と「開く」

名前を付けて保存
上書き保存
Ctrl キー + S キー

▼ Maple の四則演算と剰余演算

足し算
引き算
掛け算
割り算
余りの計算（剰余演算）
mod

▼ 変数の扱い方

▼ 変数名の作り方

_　アンダースコア
添え字

▼ 命名規則の注意事項

内部変数
予約語
D
constants
Digits
環境変数

▼ 数式入力におけるアンダースコア（ _ ）の入力

_　アンダースコア
字下げ
添え字
字下げの回避

¥　円記号

\　バックスラッシュ

添え字入力のキャンセル

▼ 変数への割り当て（代入）

割り当て（代入）

:=　コロン + イコール

上書き

同時割り当て

▼ 変数の初期化（リセット）

unassign()

restart

初期化

引数

' '　シングルクォーテーション

Shift キー + 7 キー

,　カンマanimate()

▼ 結果の再利用

restart

Ctrl キー + L キー

int()

ラベルの丸括弧 (　)

() コマンド内の丸括弧

ditto 演算子

%

%%

%%%

▼ 厳密解と数値解

有理数

整数

近似値（数値）

厳密値

数値計算
精度
切り捨て
誤差
Maple の予約語
Digits
数学定数
Pi
倍精度演算
浮動小数点演算
evalf()
evalhf()

▼ オンラインヘルプの活用

▼ Maple 13 のヘルプメニュー

Maple ヘルプ
Maple ツアー
Maple Portal （英語版のみ）
プロットガイド

▼ コマンドからヘルプを起動する方法（例題の実行）

? クエスチョン
F2 キー

▼ 応用事例

▼運動を見る！

▼物体をただ落としてみる（自由落下運動）

▼等加速度直線運動

```
diff()
```

▼自由落下運動

```
eval()
plot()
plot( labels )
plot( labeldirections )
plot( title )
```

▼こんどは物体を上に投げてみる（投げ上げ運動）

```
eval()
plot()
```

▼式のパラメータ化

▼v0 = 10 のとき

```
eval()
```

▼v0 = 20 のとき

```
eval()
```

▼v0 = 30 のとき

```
eval()
```

▼ 結果のプロット

```
plot()
plot( labels )
plot( labeldirections )
plot( legend )
```

▼ アニメーションの作成

```
with( plots )
animate( plot() )
eval()
pointplot( symbol )
pointplot( symbolsize )
animate( pointplot() )
```

▼ 投げ上げ運動のアニメーション作成

```
eval()
```

▼ 初速度 v0 = 10 の場合

```
animate( pointplot() )
plot()
solve()
```

▼ 初速度 v0 = 20 の場合

```
solve()
animate( pointplot() )
```

▼ 初速度 v0 = 30 の場合

```
solve()[]
```

```
animate( pointplot() )
```

▼ 同時アニメーションの作成

```
solve()[]
animate( pointplot( symbol ) )
animate( pointplot( symbolsize ) )
animate( pointplot( color ) )
```

▼ ついでに斜めにも投げてみる（斜め投げ上げ運動）

```
eval()
animate( pointplot( symbol ) )
animate( pointplot( symbolsize ) )
animate( pointplot( color ) )
```

ワークシートを操る！

ツールバー（Windows 及び Mac OS 共通）

ファイル操作と編集

グループの入力とセクション

計算の実行と中断，デバッグなど

その他

▼ ワークシート向けのショートカット（Windows）

機能・作業	操作
ワークスペースの拡大縮小（50%）	Ctrl + 0
ワークスペースの拡大縮小（75%）	Ctrl + 1
ワークスペースの拡大縮小（100%）	Ctrl + 2
ワークスペースの拡大縮小（125%）	Ctrl + 3
ワークスペースの拡大縮小（150%）	Ctrl + 4
ワークスペースの拡大縮小（200%）	Ctrl + 5
ワークスペースの拡大縮小（300%）	Ctrl + 6
ワークスペースの拡大縮小（400%）	Ctrl + 7
グループのレンジ（範囲）の表示切り替え	F9
セクションのレンジ（範囲）の表示切り替え	Shift + F9
スペルチェック	F7

▼ 入力とグループ向けのショートカット（Windows）

機能・作業	操作
コマンドの実行	Enter
コマンド内の改行	Shift + Enter
行先頭へ移動	Home
行末尾へ移動	End
グループをカーソルの前に追加	Ctrl + K
グループをカーソルの後に追加	Ctrl + J
パラグラフをカーソルの前に追加	Ctrl + Shift + K
パラグラフをカーソルの後に追加	Ctrl + Shift + J
カーソルある行の削除	Ctrl + Delete
テキスト入力と数式入力の切り替え	F5
カーソルの位置で実行グループの分割	F3
カーソルの位置で実行グループの結合	F4
テキスト入力への変更（プロンプトの表示）	Ctrl + M
数式入力への変更	Ctrl + R
テキストの入力	Ctrl + T
コマンド補間	Esc
コマンドヘルプの起動	コマンドにカーソルを置いて F2
ギリシャ文字の入力	Ctrl + Shift + G（例えば，$a > \alpha$）
(W) コピー	Ctrl + C
(W) 貼り付け（ペースト）	Ctrl + V
(W) 切り取り（カット）	Ctrl + X
(W) アンドゥー（作業のキャンセル）	Ctrl + Z
(W) 上書き保存	Ctrl + S
(W) すべて選択	Ctrl + A

▼ セクションとサブセクション向けのショートカット（Windows）

機能・作業	操作
セクション（サブセクション）の追加（挿入）	Ctrl + >
セクション（サブセクション）の削除	Ctrl + <

▼ 典型的なセクション内の構成

▼ セクション－サブセクション－サブサブセクション

▼ スタイルの設定

▼ 変更手順

▼ 設定するスタイル

▼ ヘッダとフッタ

▼ Maple の仲間たち

▼ Maple の予約語

▼ 数学定数

```
constants
Pi
evalf()
Digits
gamma
Catalan
```

▼ グローバル変数（大域的変数）

```
I
infinity
true
false
```

▼ 円周率 π とギリシャ文字 π

```
Pi （大文字 P）
pi （小文字 p）
eval()
evalf()
```

▼ 複素数

▼ 複素数の定義と四則演算（１）

```
I
evalc( )
```

▼ 複素数の定義と四則演算（２）

```
Complex()
evalc()
```

▼ 複素平面へのプロット

```
with( plots )
I
complexplot( style )
complexplot( symbolsize )
complexplot( labels )
```

▼ 式の性質を確認

```
I
convert( exp )
```

▼ （参考）三角関数の指数関数表記

```
convert( trig )
isolate()
```

多項式

1変数多項式の操作

```
^
degree()
coeff()
coeffs()
subs()
eval()
subs()
nops()
op()
factor()
diff()
int()
```

2変数の多項式を生成

```
^
randpoly( terms )
expand()
collect()
sort()
plot3d()
plot3d( axes )
plot3d( labels )
```

方程式

1変数の代数方程式

1次方程式

```
solve()
```

2次方程式

```
solve()
```

▼ 3 次方程式

```
solve()
```

▼ 4 次方程式

```
solve()
```

▼ RootOf() の解

```
solve()
nops
```

▼ 5 次方程式

```
solve()
```

▼ 高次方程式の数値解（近似解）

```
eval()
lhs()
plot()
fsolve()
nops()
fsolve( complex )
plots[complexplot]( style )
plots[complexplot]( symbolsize )
```

▼ 円の方程式

```
eval()
with( plots )
implicitplot()
```

▼ 楕円の方程式

```
with( plots )
sqrt()
eval()
implicitplot( scaling )
```

▼ 双曲線の方程式

```
eval()
plots[implicitplot]( scaling )
plots[implicitplot]( color )
plot( scaling )
plot( color )
plots[display]()
```

▼ 連立代数方程式の解法とプロット

```
solve()
rhs()
plot()
plots[pointplot]( symbolsize )
plots[display]()
```

▼ 放物線と楕円の交点

```
plots[implicitplot]( color )
plots[implicitplot]( numpoints )
solve()
```

▼ 数値解の求め方

```
fsolve()
fsolve( avoid )
```

▼ 要素の操作

```
convert( list )
```

```
rhs()
map()
```

▼関数

▼三角関数

```
sin()
cos()
tan()
plot()
plot( view )
```

▼三角関数の基本公式

```
simplify()
```

▼（参考）第１式から第２式を導く手順

```
expand()
lhs()
rhs()
convert( tan )
```

▼加法定理

```
expand()
combine()
```

▼双曲線関数

```
sinh()
cosh()
tanh()
convert( exp )
plot()
```

▼ 指数と指数関数

▼ 指数の定義

^

expand()

simplify()

▼ 指数関数

▼ 対数と対数関数

▼ 対数の底

solve()

log[]()

▼ 常用対数

log[10]()

▼ 対数関数

log[]()

eval()

plot(view)

▼ 自然対数

Limit()　(大文字 L)

limit()　(大文字 l)

infinity

ln()

plot()

diff()

int()

▼区分関数

```
piecewise()
plot( scaling )
eval()
limit()
diff()
int()
```

▼文字列

```
" "
cat()
printf(), %s
printf(), \n
```

▼配列

▼１次元配列

```
Array()
```

▼２次元配列

```
Array()
```

▼３次元配列

```
Array()
```

▼２次元配列から４次元配列への拡張

```
Array()
```

▼シーケンス（数列，式列）

▼シーケンスの定義

```
,
..
seq()
```

▼要素の指定

```
..
[]
seq()
```

▼その他の数列，式列，データの定義例

```
..
seq()
expand()
```

▼リスト

▼リストの定義

```
[]
[ [ ] ]
plots[pointplot]( symbol )
plots[pointplot]( symbolsize )
plots[pointplot]( color )
```

▼要素の抽出

```
[]
[][]
```

▼シーケンスのリスト

```
seq( [] )
plots[pointplot]( symbol )
plots[pointplot]( color )
```

▼行列

```
Matrix()
<>
|
```

▼ベクトル

```
Vector()
Vector[row]()
<>
|
.
```

▼集合（セット）

{}
[]

▼ブーリアン

▼ベン図

and（論理積）
or（論理輪）
not 論理否定）
xor（排他的論理輪）
implies（含意）
true
false
FAIL
boolean

▼論理演算

true
false
and
or
not
<
evalb()

▼ 型（type）

▼ Maple オブジェクトの型（type）

```
whattype()
```

▼ 変数型の仮定

```
about()
assuming
assume(  )
assume( posint )
sqrt()
simplify()
csgn()
```

▼ 主な仮定

```
assume( even )
assuming even
assume( odd )
assuming odd
assume( real )
assuming real
assume( positive )
assuming positive
assume( negative )
assuming negative
assume( integer)
assuming integer
assume( posint )
assuming posint
assume( negint )
assuming negint
assuming negative
```

▼ 仮定の解除（キャンセル）

```
assume()
unassign( ' ' )
sqrt()
```