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
応用編

数学講話 1

▼やっぱり微積分

- **数列と和**
- **極限**
- **微分法**
  - １変数関数の微分
  - ２変数関数の微分
    - 微分と可視化（アニメーション）
    - （参考） 連続微分
- **級数展開**
- **積分法**
  - 不定積分
  - 定積分
- **積分の応用**
  - 線の積分
  - 回転面の面積
    - （例１） $y = x^3 \quad (0 \le x \le 1)$
    - （例２） $y = 2\sqrt{x} \quad (0 \le x \le 1)$
    - （参考）半径 r の球の表面積 S
  - ２重積分の計算

▼いわゆる線形代数

- **行列の演算**
  - 行列の定義
  - 行列の定義と演算
  - 行列の操作
- **ベクトル**
  - ベクトルの定義
  - ベクトルの操作
  - 足し算
  - ベクトルの角度
  - 内積（スカラー積）と外積（ベクトル積）
  - ノルム（ベクトルの大きさ）
- **固有値と固有ベクトル**
  - 固有値の計算
  - 固有ベクトルの計算
- **１次変換（線形写像）のプロット**
- **連立１次方程式の解法**

▼微分方程式に挑む！

- **数学モデルの作り方**
- **常微分方程式（ODE）の定義と解析解**

少しだけ常微分方程式のこと

１階常微分方程式

（例１） $\frac{\mathrm{d}}{\mathrm{d}x}y(x)=\left(1-y(x)^2\right)\tan(x)$

（例２） $x\left(\frac{\mathrm{d}}{\mathrm{d}x}y(x)\right)=y(x)+\sqrt{x^2+y(x)^2}$

（例３） $\frac{\mathrm{d}}{\mathrm{d}x}y(x)+2y(x)\tan(x)=\sin(x)$

２階線形常微分方程式の解

１自由度粘性減衰振動系の解

指数関数の特性

１） 異なる２つの実根を持つ場合

a) ともに正の場合

b) 正と負の場合

c) ともに負の場合

2） 異なる２つの虚根を持つ場合

d) 実部が正の場合

e) 実部が負の場合

少しだけ考察…

２階線形常微分方程式の１階化

（参考）連立微分方程式の解法

初期値問題

ラプラス変換による初期値問題の解法

- **常微分方程式（ODE）の数値解法**

【参考】物体の運動の可視化

【参考】位相平面のプロット

▼プログラミング前夜

- **処理の再利用**
  - プロシージャ化（処理を１行で記述可能な場合）
  - プロシージャ化（処理を複数行で記述する場合）
  - グローバル変数とローカル変数
    - （例）グローバル変数とローカル変数の違い
  - 引数の型・変数の型
- **制御構文**
  - 条件分岐（if-then-else-end if）
    - 数字の大小を判別する条件分岐
    - 条件分岐を含むプロシージャの作成手順
    - （参考）引数を比較するプロシージャに変更
  - 繰り返し（for-do-end do）
    - 足し算の実行
      - 繰り返し文の前に
      - 繰り返し文の利用
      - プロシージャへの拡張
        - （参考）sum コマンドを用いて実行します
    - （参考）２つのインデックス操作
- **それは振動しますか？**
  - Maple による２階常微分方程式の解法と解のプロット
  - 「それは振動しますか？」プロシージャの作成手順
    - ２階常微分方程式の解法をプロシージャ化
    - プロシージャのパラメータ化
    - 固有値の計算
    - 固有値の実部・虚部抽出
    - 実部の符号と条件分岐
    - 虚部の符号と条件分岐
    - 固有値のプロット

▼みんなの Maple

- Maple コンポーネントの基礎
  - Maple コンポーネントの基本的な利用方法
    - プロットのためのプロシージャ
  - Maple コンポーネントの種類
- グラフィカル・ユーザー・インターフェース（GUI）の構築
  - 「それは振動しますか？」プロシージャの確認
  - 「それは振動しますか？」の GUI 化
    - mySim7 プロシージャの変更
    - （参考）スライダーコンポーネントの設定例

ステップ･バイ･ステップ式 はじめての Maple 応用編

やっぱり微積分

微積分に関する基本的な操作手順を習得します.

目次

数列と和

初期化します.

```
> restart;
```

第 n 項が与えられている数列

```
> srn := (1-n^3)/(3*n+1);
```

$$srn := \frac{1-n^3}{3n+1} \tag{1}$$

（有限）数列の作成（seq コマンドの利用）

```
> seq( srn, n=0..10 );
```

$$1, 0, -1, -\frac{13}{5}, -\frac{63}{13}, -\frac{31}{4}, -\frac{215}{19}, -\frac{171}{11}, -\frac{511}{25}, -26, -\frac{999}{31} \tag{2}$$

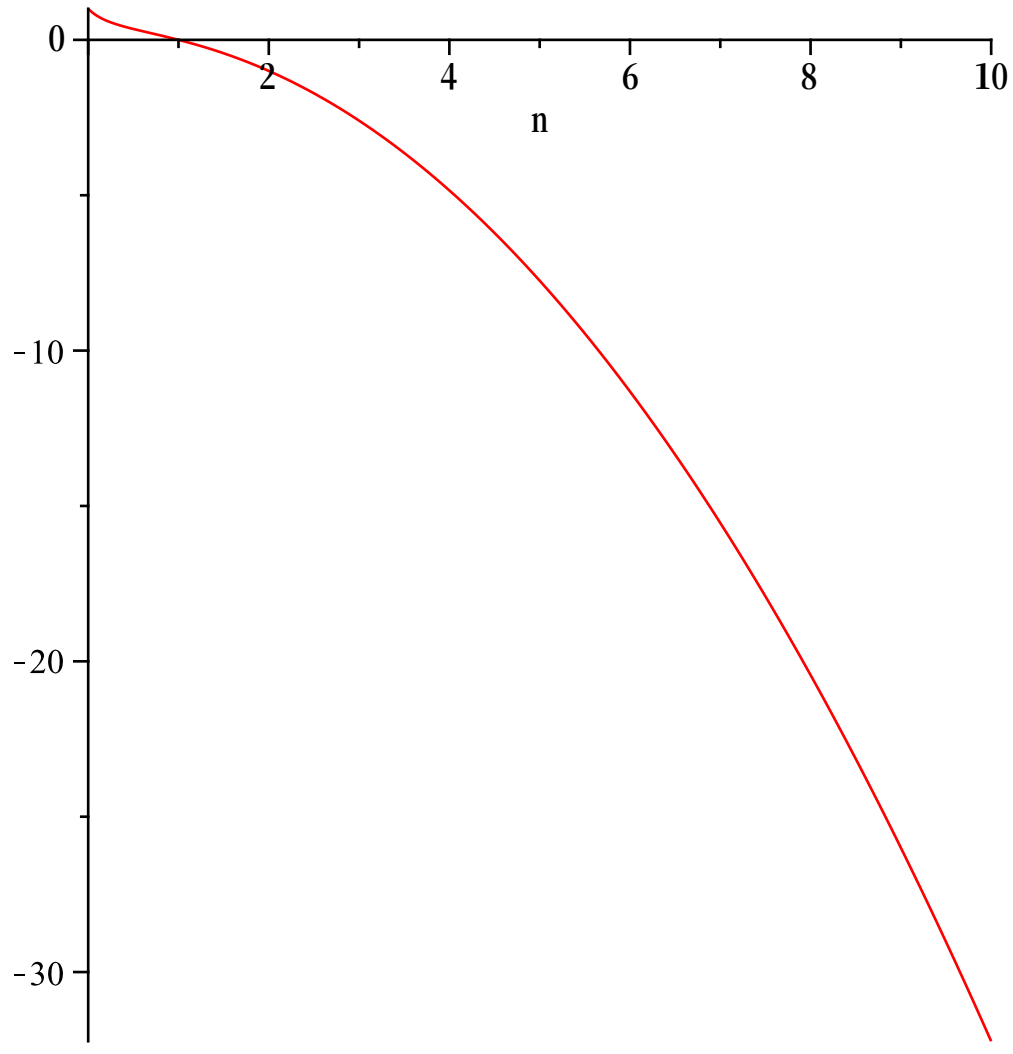
n=0..10 の和

```
> sum( srn, n=0..10 );
```

$$-\frac{1016815333}{8422700} \tag{3}$$

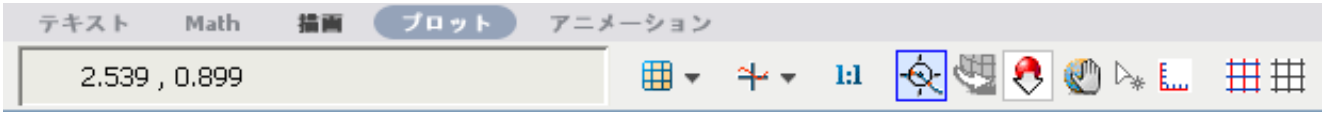
（有限範囲での）プロット

```
> plot( srn, n=0..10 );
```



（参考）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.



第 n 項の再表示

```
> srn;
```

$$\frac{1-n^3}{3n+1} \tag{4}$$

＋無限大N（infinity）までの和

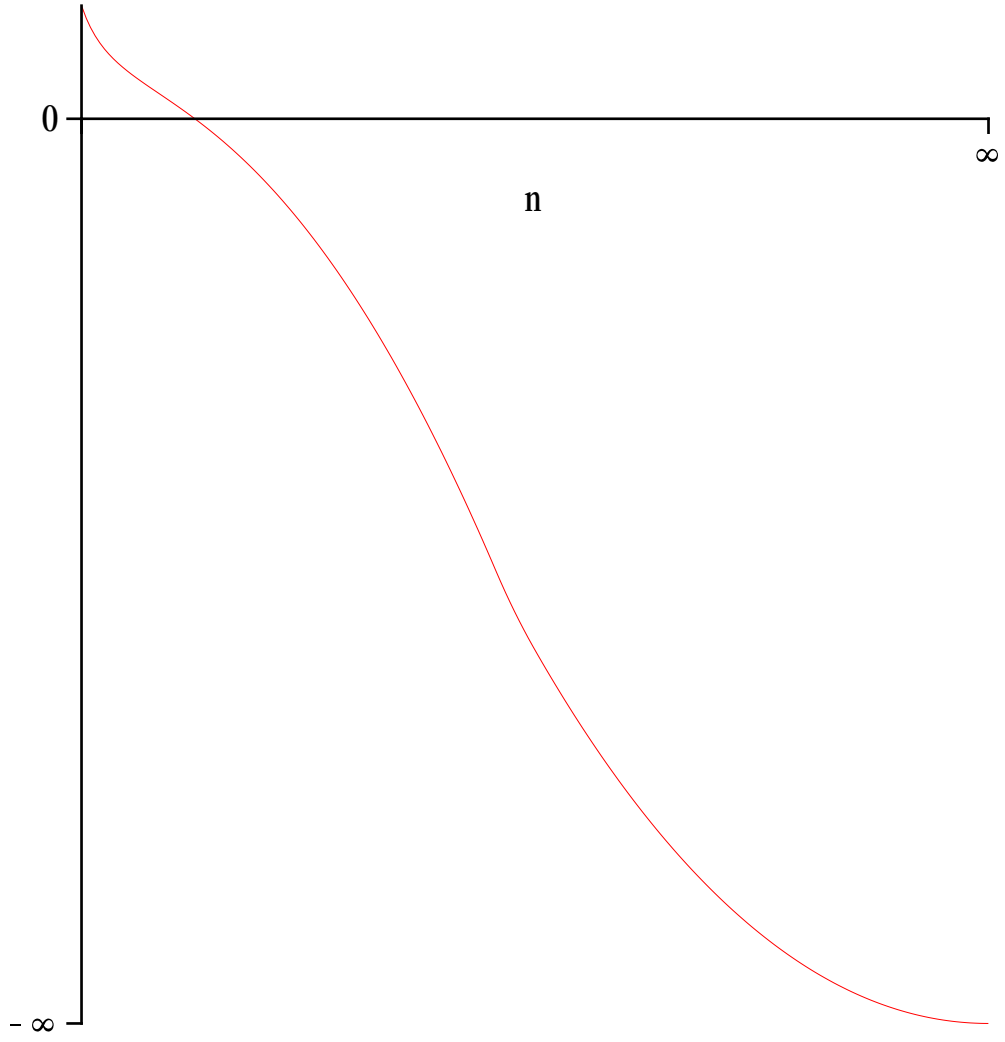
```
> sum( srn, n=0..infinity );
```

$$-\infty \tag{5}$$

無限プロット

（注意）
数学定数infinity（無限大，N）を用いたプロットは，対象，例えば数列の第 n 項や関数などの定性的な挙動を示したものであり，定量的あるいは数値的に正しい挙動をプロットしたものではありません.

```
> plot( srn, n=0..infinity );
```



第n項の再表示

```
> srn;
```

$$\frac{1-n^3}{3n+1} \tag{6}$$

（参考）極限（

limitコマンド)

```
> limit( srn, n=infinity );
```

$$-\infty \tag{7}$$

極限

初期化します.

```
> restart;
```

関数 $\frac{1}{x-1}$ の極限を調べます.

関数 $\frac{1}{x-1}$ の定義

```
> fn := 1/(x-1);
```

$$fn := \frac{1}{x-1} \tag{8}$$

$x=1$ の極限

```
> limit( fn, x=1 );
```

$$\text{undefined} \tag{9}$$

極限値が定まりません. ここで, 左側（left）から $x=1$ のときの極限を調べます.

```
> limit( fn, x=1, left );
```

$$-\infty \tag{10}$$

左側（right）から $x=1$ のときの極限を調べます.

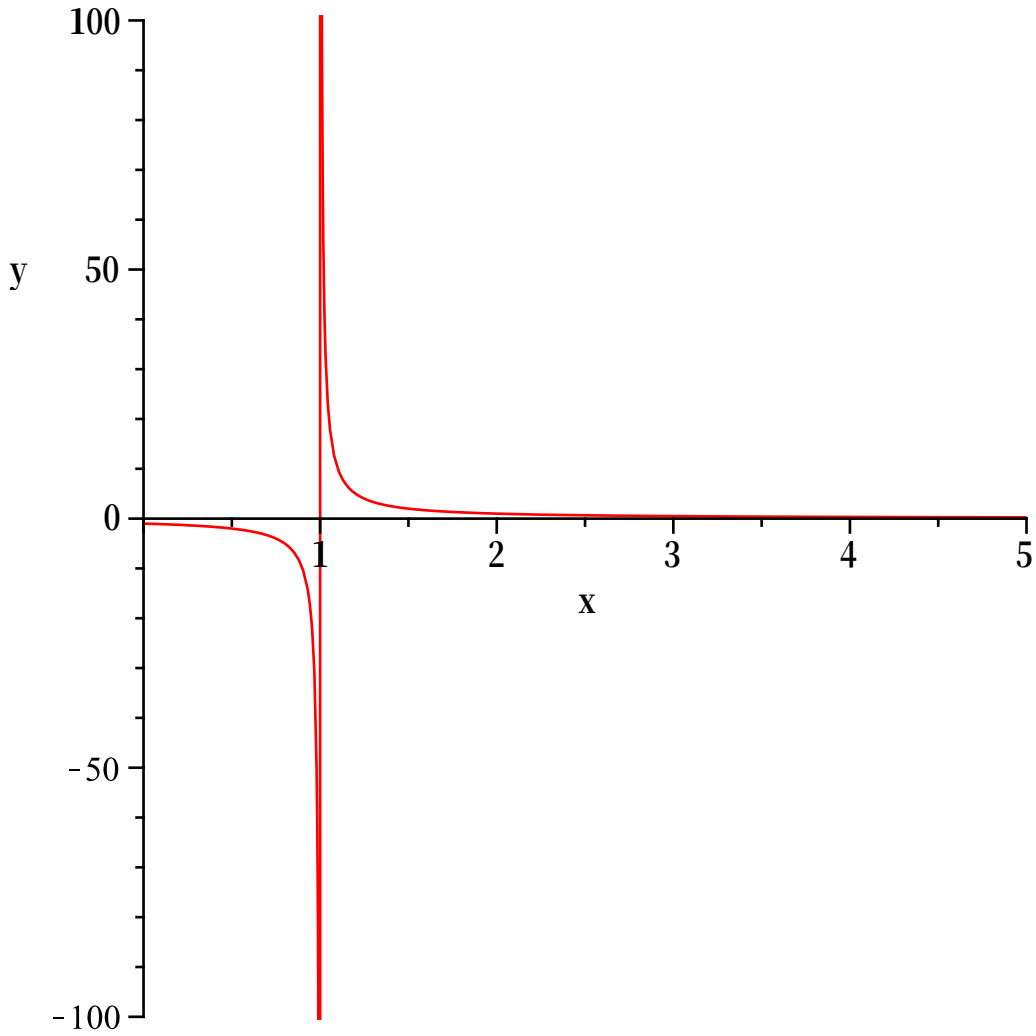
```
> limit( fn, x=1, right );
```

$$\infty \tag{11}$$

プロット範囲を指定して, 関数 $\frac{1}{x-1}$ をプロットします.
□オプションdiscont=trueは不連続点を削除します.

関数 $\frac{1}{x-1}$ をプロット（1）

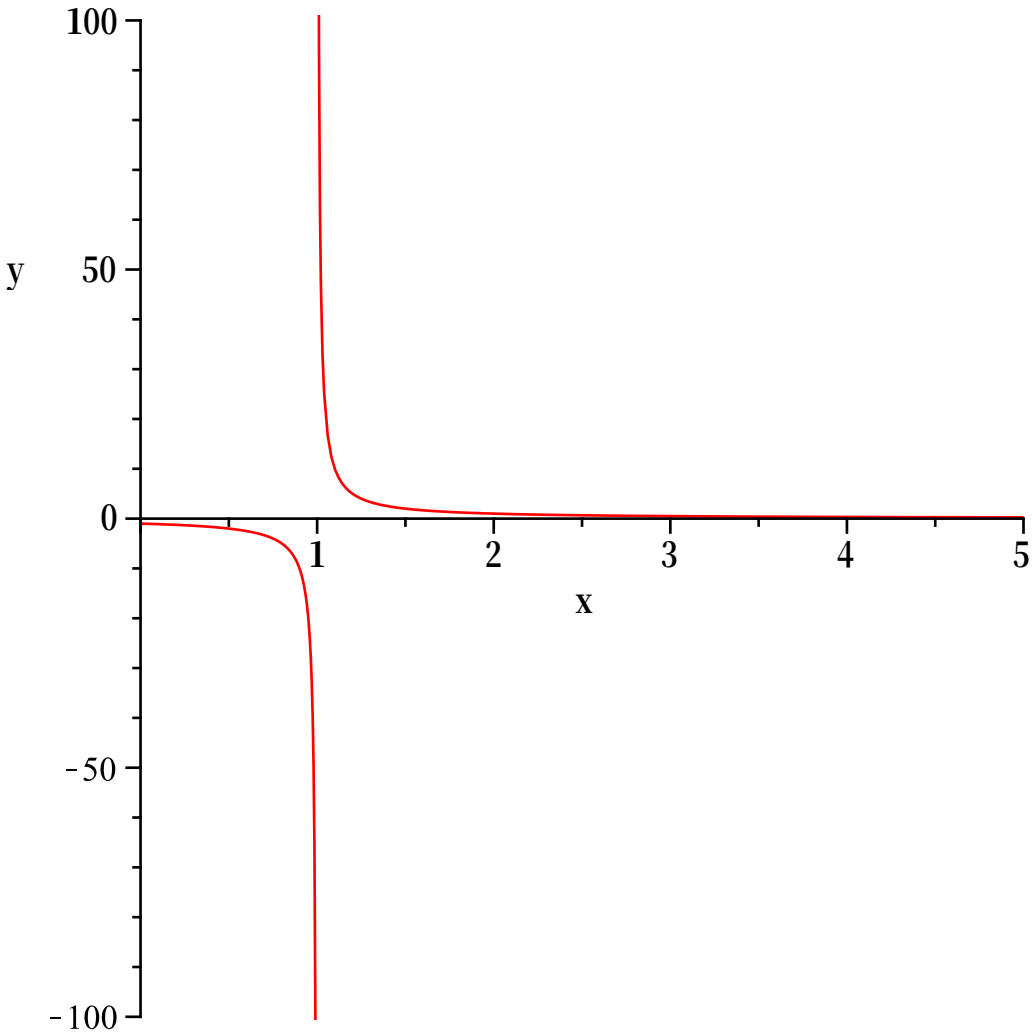
```
> plot( fn, x=0..5, y=-100..100 );
```



関数 $\frac{1}{x-1}$ をプロット（2）

□オプションdiscont=trueを用いて不連続点を削除します.

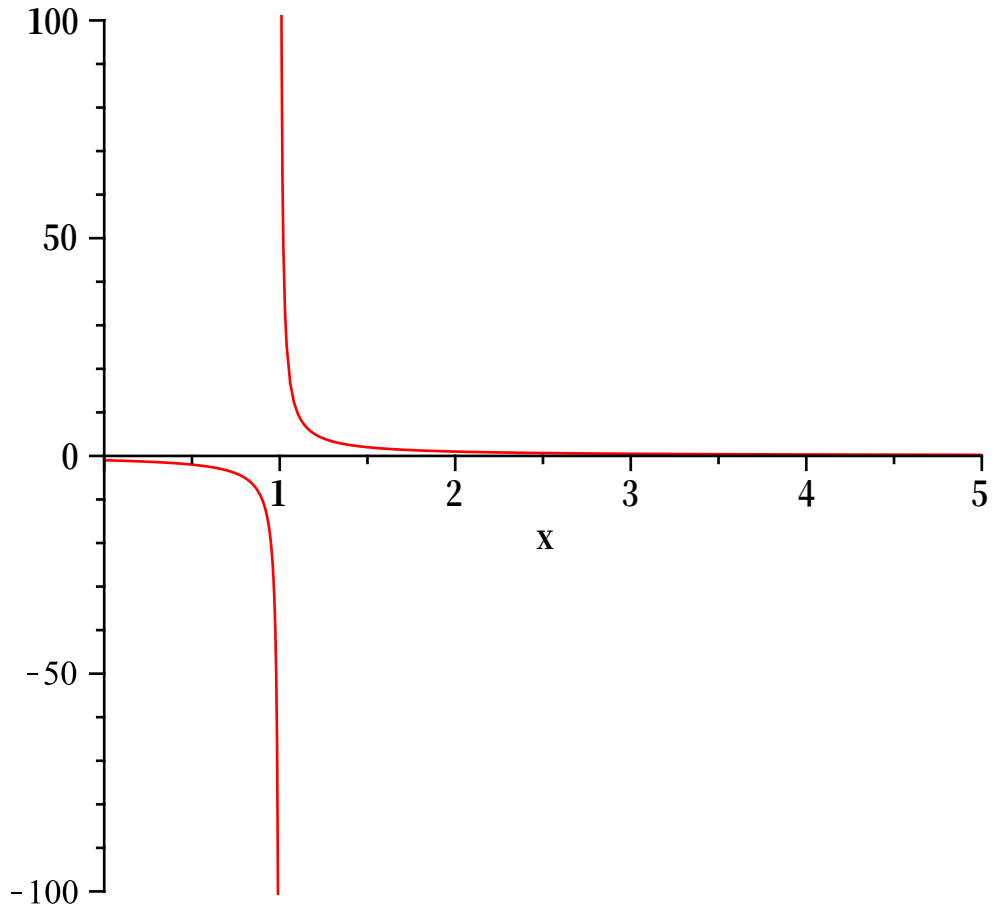
```
> plot( fn, x=0..5, y=-100..100, discont=true );
```



（参考１）極限プロットのアニメーション化

極限に向かう点の挙動をアニメーションで確認します．ここでは，Studentパッケージに含まれているサブパッケージPrecalculusのLimitPlotコマンドを使用します．

```
> Student[Precalculus][LimitPlot]( fn, x=1, view=[ 0..5, -100..100 ] );
```



The limit of $f(x) = \frac{1}{x-1}$ at $x = 1$ depends on the values $f(x)$ assumes near $x = 1$.

（参考）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．


```
> #terminator
```

微分法

1変数関数の微分

はじめに，初期化します．

```
> restart;
```

xの関数としてfを定義します．

```
> f := sin( a*x )/cos( a*x );
```

$$f := \frac{\sin(a\,x)}{\cos(a\,x)} \tag{12}$$

関数fをxで**1階微分**します．

```
> df := diff( f, x );
```

$$df := a + \frac{\sin(a\,x)^2\,a}{\cos(a\,x)^2} \tag{13}$$

結果を簡単な形に整理します．

```
> simplify( df );
```

$$\frac{a}{\cos(a\,x)^2} \tag{14}$$

三角関数$\sin(a\,x)$を定義し，xで**2階微分**します．

```
> f2 := sin( a*x );
```

$$f2 := \sin(a\,x) \tag{15}$$

```
> diff( f2, x, x );
```

$$-\sin(a\,x)\,a^2 \tag{16}$$

指定方法を一般化します（ダラー演算子 $ の利用）．

```
> diff( f2, x$2 );
```

$$-\sin(a\,x)\,a^2 \tag{17}$$

xのn**階微分**になります．

```
> diff( f2, x$n );
```

$$\sin\left(a\,x + \frac{n\,\pi}{2}\right)a^n \tag{18}$$

（参考）大文字Dで始まる微分コマンドDiffを用いると，評価前の形式で結果が表示されます（イナート形式あるいは不活性形式と呼ばれます）．評価を進める場合は，valueコマンドを使用します．

```
> Diff( f2, x$2 );
```

$$\frac{\partial^2}{\partial x^2}\,\sin(a\,x) \tag{19}$$

```
> value( (19) );
```

$$-\sin(a\,x)\,a^2 \tag{20}$$

（参考）ダラー演算子$はseqコマンドとほぼ同等の働きを持っています．同じ処理を実行させる場合，タイピングの回数（文字を入力する回数）を減らすことができます．

（例1）シーケンス，x, x, x, x, xの定義

ダラー演算子 $ を用いた場合

```
> x$5;
```

$$x, x, x, x, x \tag{21}$$

seq コマンドを用いた場合

```
> seq( x, i=1..5 );
```

$$x, x, x, x, x \tag{22}$$

（例２） 一般項が $\frac{1-n^3}{3n+1}$ で定義される数列（n は正の整数）

```
> (1-n^3)/(3*n+1) $ n=0..10;
```

$$1, 0, -1, -\frac{13}{5}, -\frac{63}{13}, -\frac{31}{4}, -\frac{215}{19}, -\frac{171}{11}, -\frac{511}{25}, -26, -\frac{999}{31} \tag{23}$$

```
> seq( (1-n^3)/(3*n+1), n=0..10 );
```

$$1, 0, -1, -\frac{13}{5}, -\frac{63}{13}, -\frac{31}{4}, -\frac{215}{19}, -\frac{171}{11}, -\frac{511}{25}, -26, -\frac{999}{31} \tag{24}$$

２変数関数の微分

はじめに，初期化します．

```
> restart;
```

微分と可視化（アニメーション）

はじめに，パラメータ a を sin 関数の位相として，x と y の関数 $g(x, y)$ を定義します．

```
> g := x^2*sin( y + a );
```

$$g := x^2 \sin(y + a) \tag{25}$$

次に，パラメータ a（sin 関数の位相）を変動させて関数 g の変化をアニメーションで確認します（可視化します）．ここでは plots パッケージの animate コマンドを使用します．

x 軸の上下限値

```
> xmin := -(3/2)*Pi;
  xmax :=  (3/2)*Pi;
```

$$xmin := -\frac{3\pi}{2}$$

$$xmax := \frac{3\pi}{2} \tag{26}$$

y 軸の上下限値

```
> ymin := -Pi;
  ymax :=  Pi;
```

$$ymin := -\pi$$

$$ymax := \pi \tag{27}$$

パラメータ a の変動幅

□パラメータ a が -3 π から 3 π に変動して，アニメーションが生成されます．

```
> amin := -3*Pi;
  amax :=  3*Pi;
```

$$amin := -3\pi$$

(28)

$$amax := 3\pi \tag{28}$$

（参考）アニメーションの生成

・アニメーションを生成するanimateコマンドの基本的な入力方法になります．

```
plots[animate]( プロットコマンド, [ プロットコマンドの引数 ], アニメーションパラメータ );

あるいは,

with( plots );
animate( プロットコマンド, [ プロットコマンドの引数 ], アニメーションパラメータ);
```

・animateコマンドのオプションframesは，アニメーションのフレーム数を正の整数で指定します．フレーム数が大きくなるに従って，アニメーションも滑らかになります．ただし，計算量は増大します．

・plot3dコマンドのオプションaxesは，軸の種類を指定します．ここではボックス型（boxed）を指定しています．その他の種類として，frame（フレーム型），none（なし），normal（標準）が用意されています．

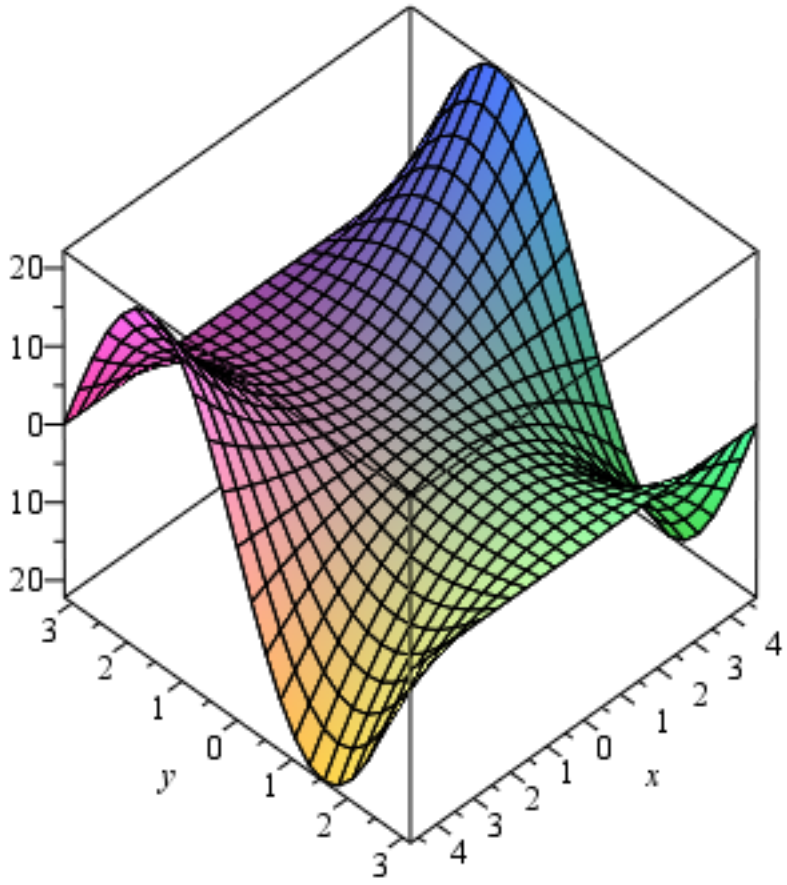
```
> g;                  # 関数表示のみ（処理とは関係ありません）
  x = xmin..xmax; # 範囲表示のみ（処理とは関係ありません）
  y = ymin..ymax; # 範囲表示のみ（処理とは関係ありません）

  plots[animate]( plot3d, [ g,                  # 関数の指定( plot3d )
                            x = xmin..xmax, # x 軸の範囲指定( plot3d )
                            y = ymin..ymax, # y 軸の範囲指定( plot3d )
                             axes = boxed ], # 軸の設定( plot3d )
                  a = amin..amax,             # パラメータの範囲指定( animate
  )
                   frames = 50 );             # フレーム数の指定( animate )
```

$$x^2 \sin(y + a)$$

$$x = -\frac{3\pi}{2} .. \frac{3\pi}{2}$$

$$y = -\pi..\pi$$

$$a = -9.4248$$



y軸方向にx^2の関数（放物線）が$\sin(y+a)$を係数として連続的に変化していることがわかります．

（参考）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．

はじめに，関数gをxで**1階微分**し，得られた導関数をアニメーションで確認します．

```
> dgx := diff( g, x );
```

$$dgx := 2x\sin(y+a) \tag{29}$$

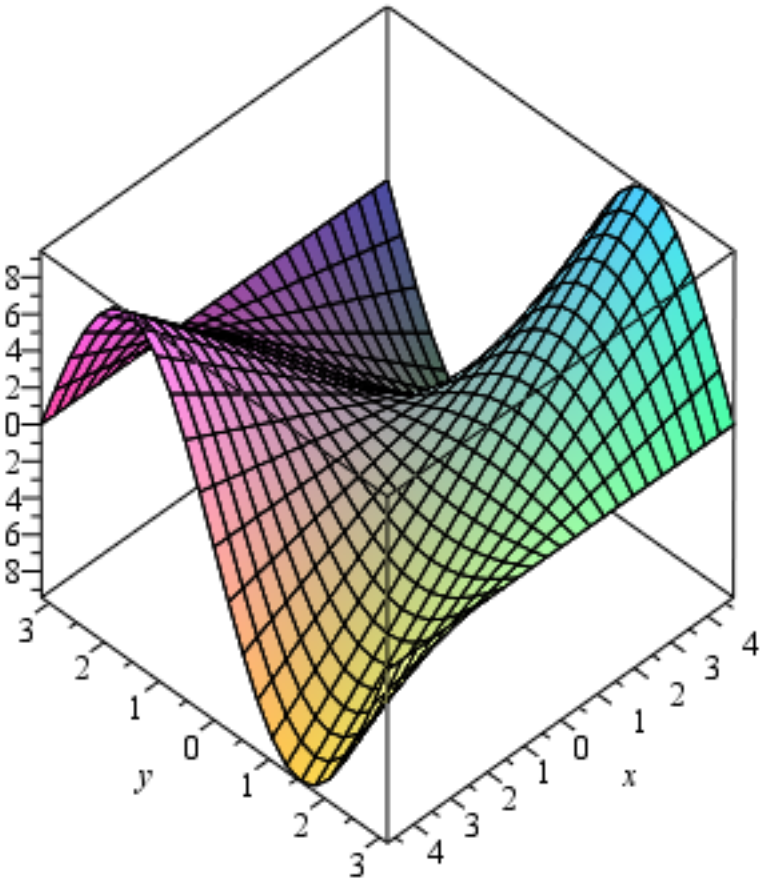
```
> dgx;              # 関数表示のみ（処理とは関係ありません）
  x = xmin..xmax; # 範囲表示のみ（処理とは関係ありません）
  y = ymin..ymax; # 範囲表示のみ（処理とは関係ありません）

  plots[animate]( plot3d, [ dgx,
                            x = xmin..xmax,
                            y = ymin..ymax,
                            axes = boxed ],
                  a = amin..amax,
                  frames = 50 );
```

$$2x\sin(y+a)$$

$$x = -\frac{3\pi}{2}..\frac{3\pi}{2}$$

$$y = -\pi..\pi$$

$$a = -9.4248$$



y軸方向にxの関数（直線）が$\sin(y+a)$を係数として連続的に変化していることがわかります.

（参考）連続微分

再度，関数gをxで1階微分し，同じ変数名dgxに割り当てます.

```
> dgx := diff( g, x );
```

$$dgx := 2x\sin(y+a) \tag{30}$$

さらに，yで1階微分し，ddgxyに割り当てます.

```
> ddgxy := diff( dgx, y );
```

$$ddgxy := 2x\cos(y+a) \tag{31}$$

ここで，関数gをx，yの順番で連続して微分を行います.

```
> diff( g, x, y );
```

$$2x\cos(y+a) \tag{32}$$

ddgxyと同じ結果が得られます.

（参考1）大文字Dで始まる微分コマンドDiffを用いると，評価前の形式で結果が表示されます（イナート形式あるいは不活性形式と呼ばれます）．評価を進める場合は，valueコマンドを使用します.

```
> DDgxy := Diff( g, x, y );
```

$$DDgxy := \frac{\partial^2}{\partial x \partial y}\left(x^2\sin(y+a)\right) \tag{33}$$

```
> value( DDgxy );
```

$$2x\cos(y+a) \tag{34}$$

（参考２）以下の二つは同値です.

x, x, y の順番で微分します.

```
> DDDgxxy := Diff( g, x,x,y );
```

$$DDDgxxy := \frac{\partial^3}{\partial x^2 \partial y}\left(x^2\sin(y+a)\right) \tag{35}$$

y, x, x の順番で微分します.

```
> DDDgyxx := Diff( g, y,x,x );
```

$$DDDgyxx := \frac{\partial^3}{\partial y \partial x^2}\left(x^2\sin(y+a)\right) \tag{36}$$

DDDgxxy, DDDgyxx をそれぞれ評価します.

$\frac{\partial^3}{\partial x^2 \partial y}\left(x^2\sin(y+a)\right)$ **を評価**

```
> value( DDDgxxy );
```

$$2\cos(y+a) \tag{37}$$

$\frac{\partial^3}{\partial y \partial x^2}\left(x^2\sin(y+a)\right)$ **を評価**

```
> value( DDDgyxx );
```

$$2\cos(y+a) \tag{38}$$

同じ値 $2\cos(y+a)$ **が得られます.**

級数展開

初期化します．

```
> restart;
```

級数は，数列の無限和になり，数列の和は１変数（例えば，x など）の多項式になります．

級数展開では，ある無限回微分可能な関数（例えば，$e^{ax}, \sin(ax), \cos(ax)$ など）が対象になります．足される項の数が増えると，すなわち多項式の次数が上がると，級数展開される前の関数に近づいていきます（これを，近似と呼びます．正確には，関数の多項式近似になります）．

近似は，形や値がよく似ているだけで，必ず真の形や値との間に差が生じます（これを，誤差と呼びます）．ただし，関数を多項式に近似することで，その後の解析がとても容易になります．

関数 $\sin(x)$ を，$x=0$ で（多項式の）次数を 5 として級数展開します．

```
> ser1 := sin(x) = series( sin(x), x=0, 5 );
```

$$ser1 := \sin(x) = x - \frac{1}{6}x^3 + O(x^5) \tag{39}$$

$O(x^5)$ を剰余項と呼びます．多項式近似において，誤差の項を意味します．

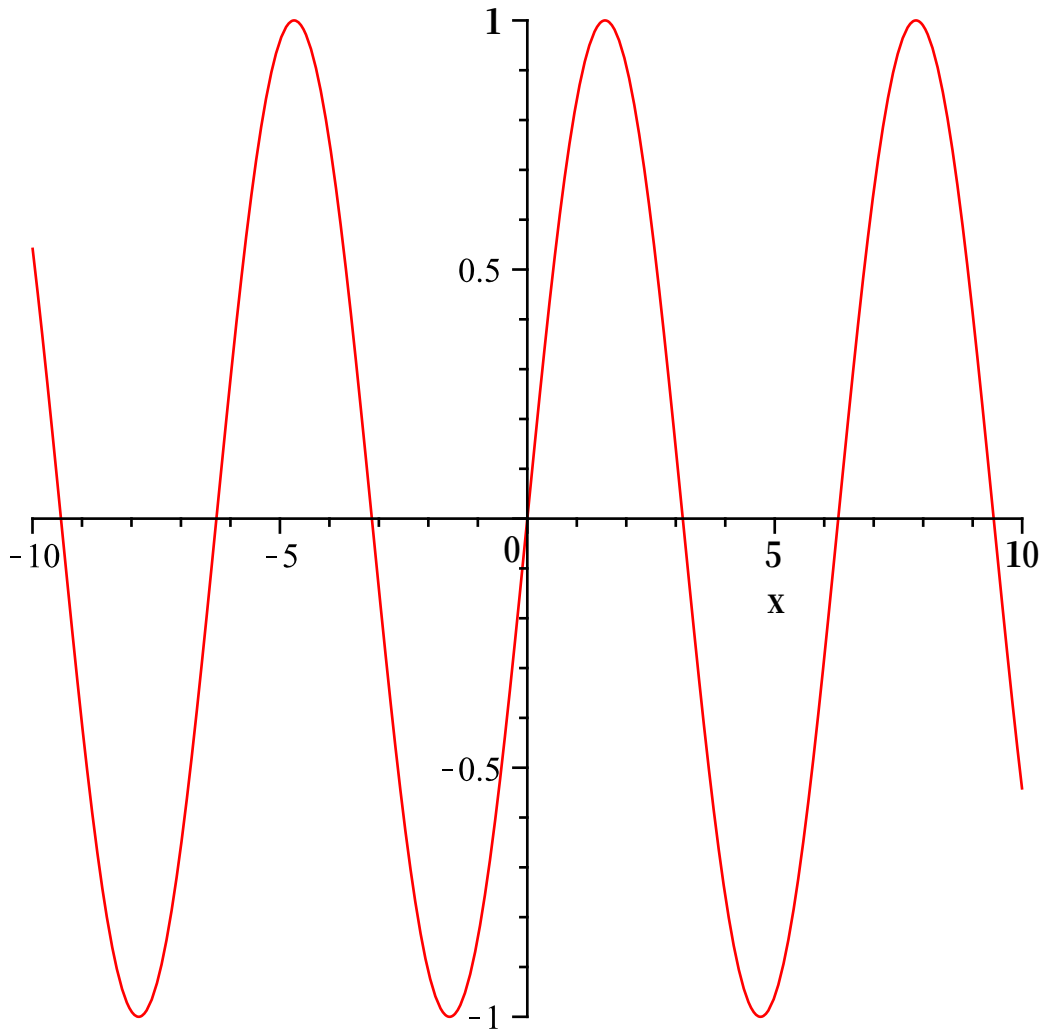
（参考） $O(x^5) = \sin(x) - \left(x - \frac{1}{6}x^3\right)$

展開前の関数（$\sin(x)$）と展開後の級数（$x - \frac{1}{6}x^3 + O(x^5)$）をプロットします．lhs コマンドは式の左辺を抽出し，rhs コマンドは式の右辺を抽出します．

```
> lhs( ser1 );
```

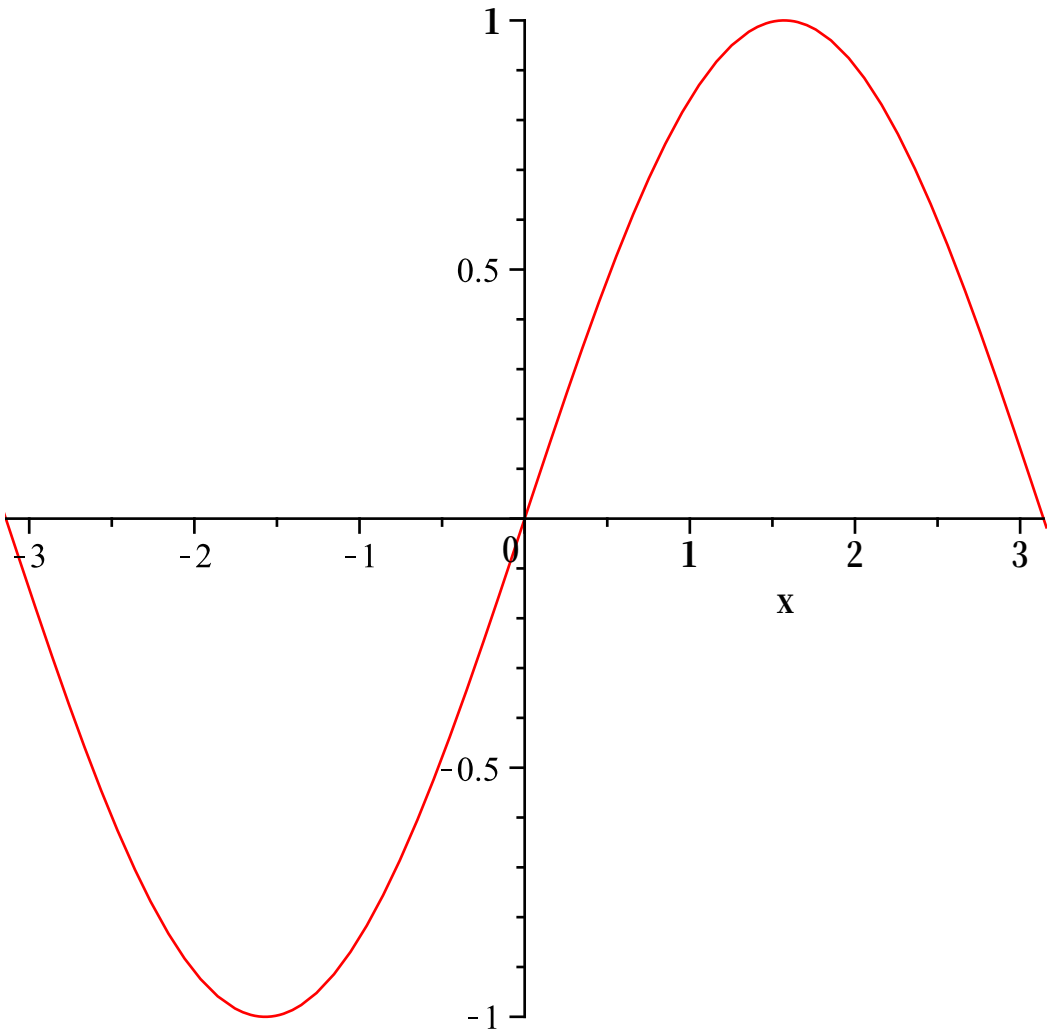
$$\sin(x) \tag{40}$$

```
> plot( lhs(ser1) );
```



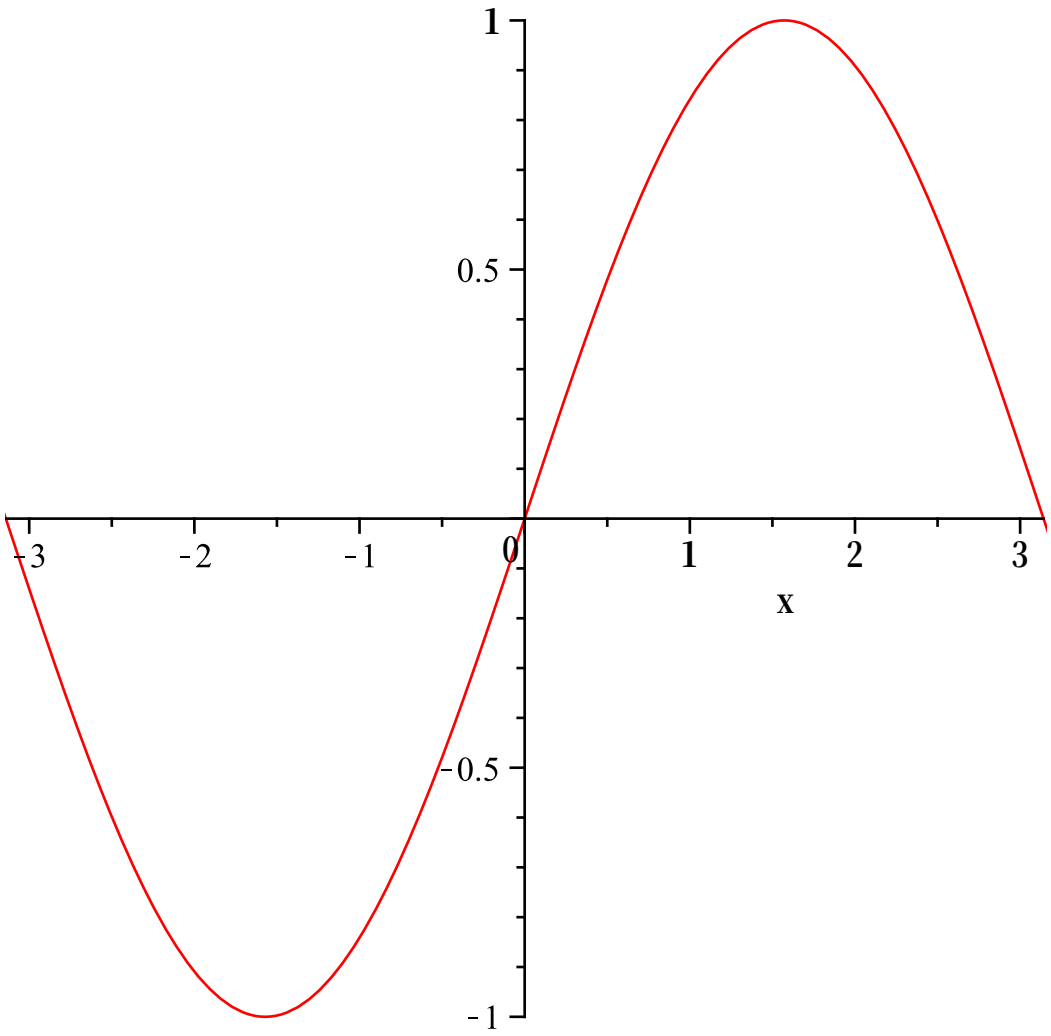
プロットの描画範囲を指定
横軸： −π～π
縦軸： −1.0～1.0

```
> plot( lhs(ser1), view = [ -Pi..Pi, -1.0..1.0 ] );
```



プロットの点数を指定

```
> plot( lhs(ser1), view = [ -Pi..Pi, -1.0..1.0 ], numpoints=300 );
```



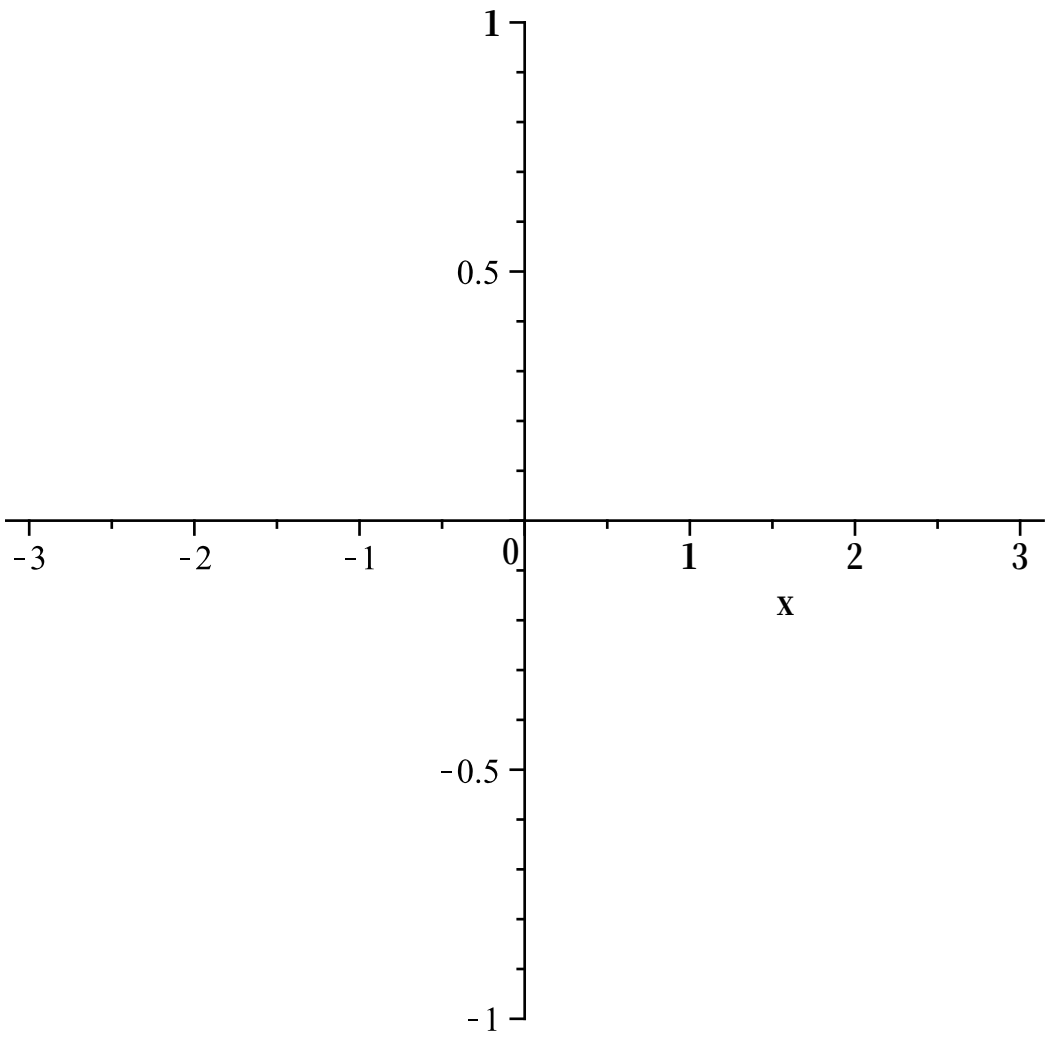
続けて，展開後の級数（$x-\frac{1}{6}x^3+O(x^5)$）をプロットします.

```
> rhs( ser1 );
```

$$x-\frac{1}{6}x^3+O(x^5) \tag{41}$$

```
> plot( rhs(ser1), view = [ -Pi..Pi, -1.0..1.0 ], numpoints=300 );
```

Warning, unable to evaluate the function to numeric values in the region; see the plotting command's help page to ensure the calling sequence is correct



展開後の級数がプロットされません.

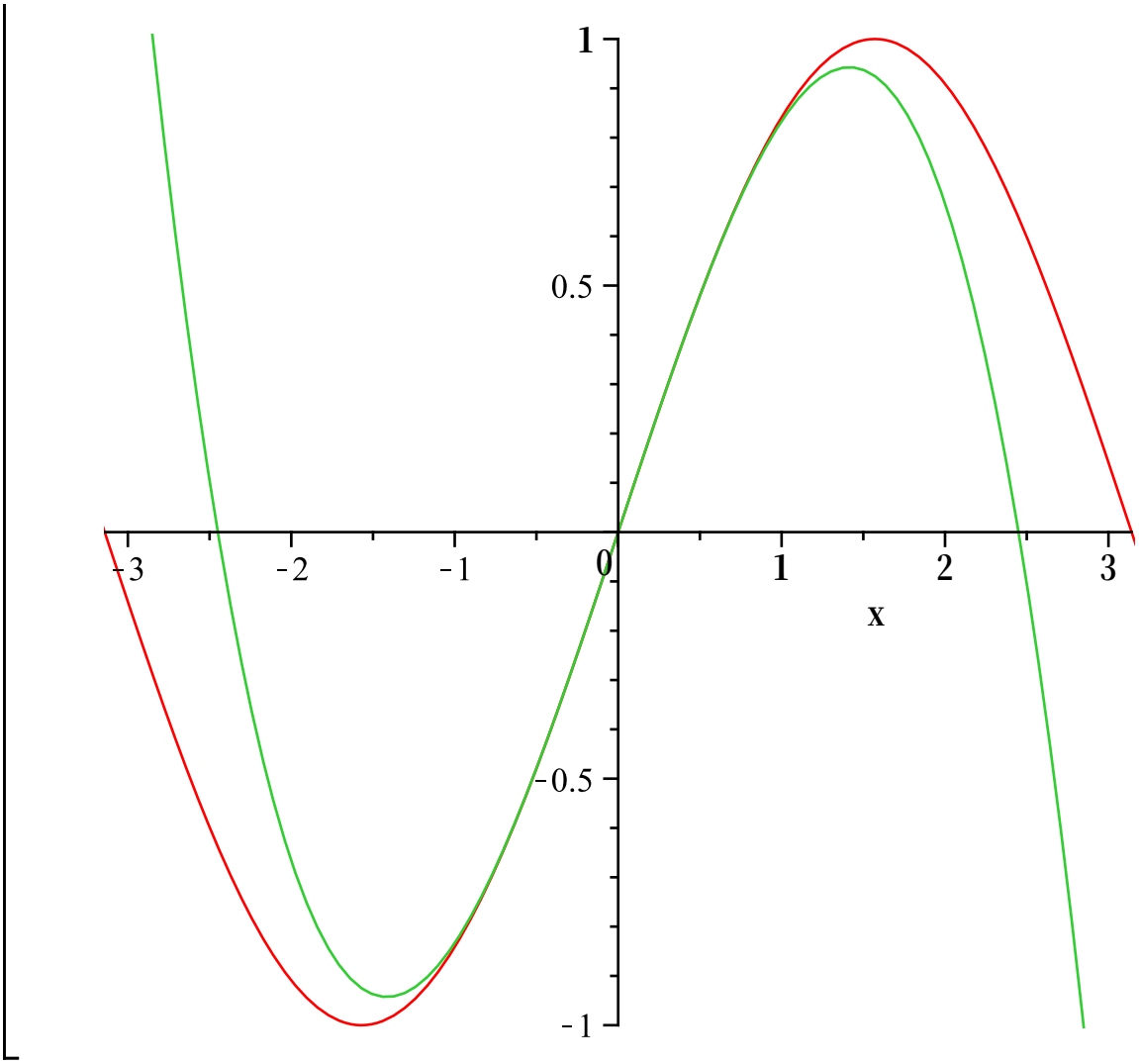
これは. 剰余項 $O(x^5)$ が数値でないためにプロットできないためです. ここで, 級数を純粋な多項式に変換します. すなわち, 剰余項 $O(x^5)$ を打ち切ります. このとき発生する誤差を打切り誤差といいます.

```
> pol1 := convert( rhs(ser1), polynom );
```

$$pol1 := x - \frac{1}{6} x^3 \tag{42}$$

剰余項 $O(x^5)$ がなくなります（打ち切られます）. 再度, 展開前の関数と展開後, 級数から多項式に変換した式を同時にプロットします.

```
> plot( [ lhs(ser1), pol1 ], view = [ -Pi..Pi, -1.0..1.0 ], numpoints=300 );
```



展開点 $x=0$ 近傍で，非常によく似た形を示していますが，展開点から離れるに従って差が大きくなっていきます．再度，次数を上げて，近似の度合を調査します．

$\sin(x)$ を，$x=0$ の点において，24 次で級数展開します．

```
> ser2 := sin(x) = series( sin(x), x=0, 24 );
```

$$ser2 := \sin(x) = x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9 - \frac{1}{39916800}x^{11} + \frac{1}{6227020800}x^{13} - \frac{1}{1307674368000}x^{15} + \frac{1}{355687428096000}x^{17} - \frac{1}{121645100408832000}x^{19} + \frac{1}{51090942171709440000}x^{21} - \frac{1}{25852016738884976640000}x^{23} + O(x^{24}) \quad (43)$$

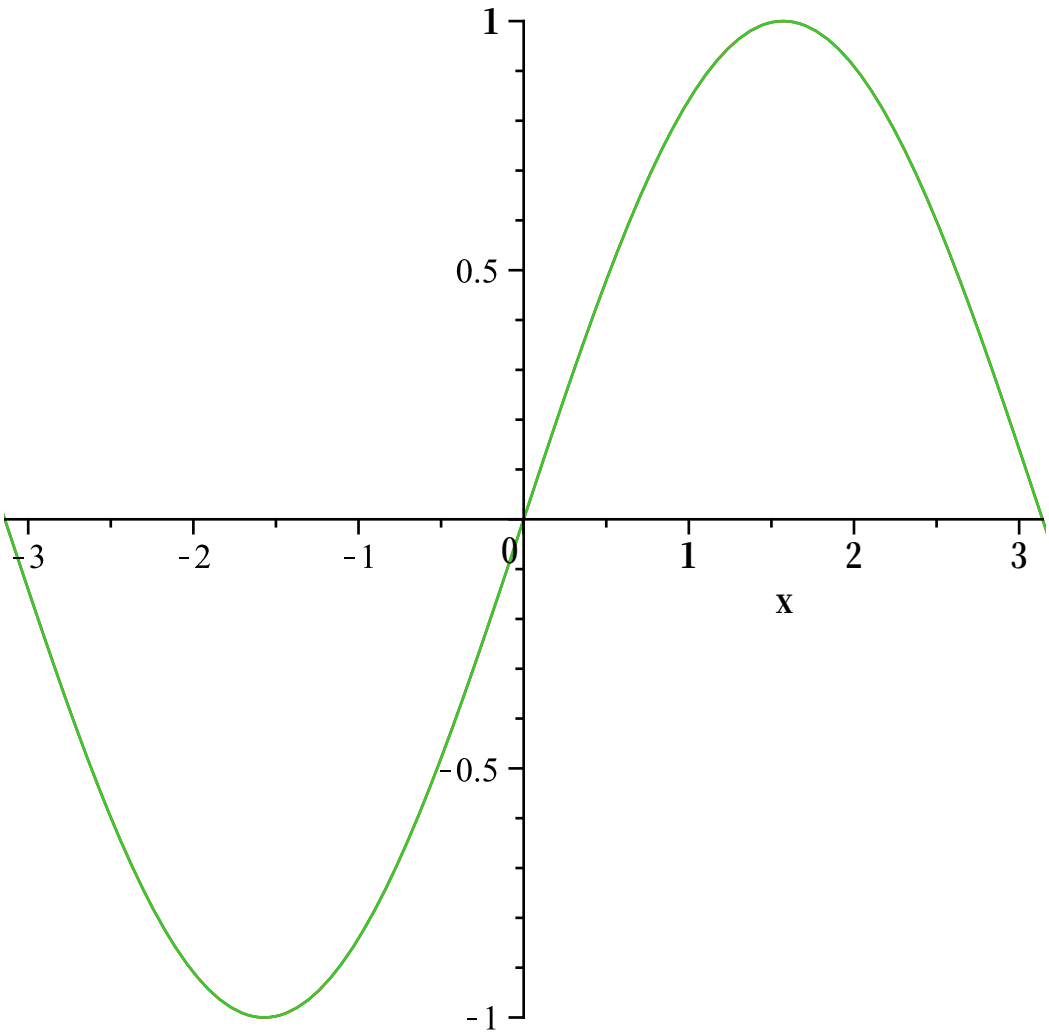
展開後の級数を多項式に変換します．

```
> pol2 := convert( rhs(ser2), polynom );
```

$$pol2 := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9 - \frac{1}{39916800}x^{11} + \frac{1}{6227020800}x^{13} - \frac{1}{1307674368000}x^{15} + \frac{1}{355687428096000}x^{17} - \frac{1}{121645100408832000}x^{19} + \frac{1}{51090942171709440000}x^{21} - \frac{1}{25852016738884976640000}x^{23} \quad (44)$$

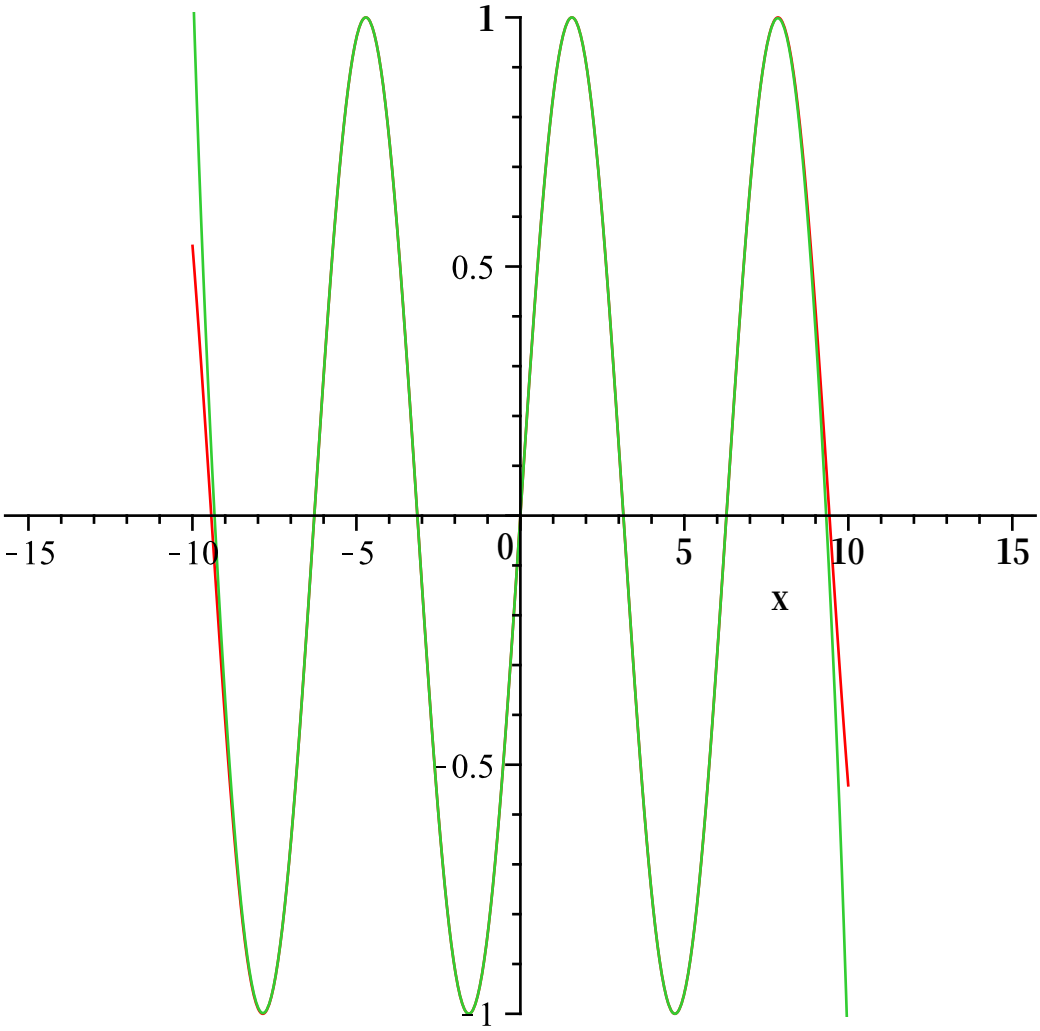
展開前の関数と展開後の多項式に変換した式を同時にプロットします．

```
> plot( [ lhs(ser2), pol2 ], view = [ -Pi..Pi, -1.0..1.0 ], numpoints=300 );
```



ほぼ重なります．プロットの範囲を広げます．

```
> plot( [ lhs(ser2), pol2], view = [ -5*Pi..5*Pi, -1.0..1.0 ], numpoints=300 )
  ;
```



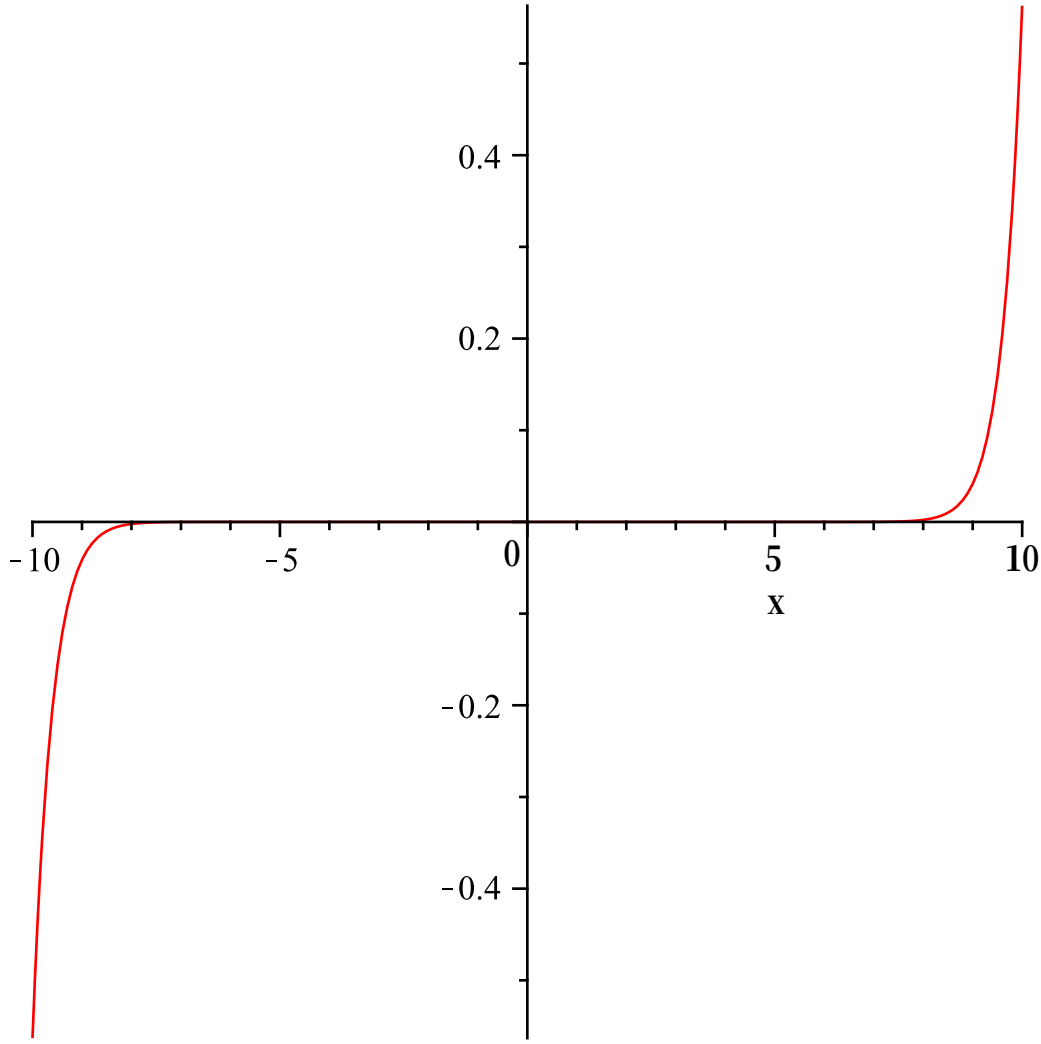
やはり，プロットの両端で差が見られます．（打ち切り）誤差をプロットします．

関数（sin(x)）と多項式（pol2）の差（err2）を取ります．

```
> err2 := sin(x) - pol2;
```

$$err2 := \sin(x) - x + \frac{x^3}{6} - \frac{x^5}{120} + \frac{x^7}{5040} - \frac{x^9}{362880} + \frac{x^{11}}{39916800} - \frac{x^{13}}{6227020800} + \frac{x^{15}}{1307674368000} - \frac{x^{17}}{355687428096000} + \frac{x^{19}}{121645100408832000} - \frac{x^{21}}{51090942171709440000} + \frac{x^{23}}{25852016738884976640000} \quad (45)$$

```
> plot( err2 );
```



次に，関数 $\sin(x)+\cos(x)$ を，展開点を $x=0$ として 6 次で級数展開します．

```
> ser3 := sin(x) + cos(x) = series( sin(x) + cos(x), x=0, 6 );
```

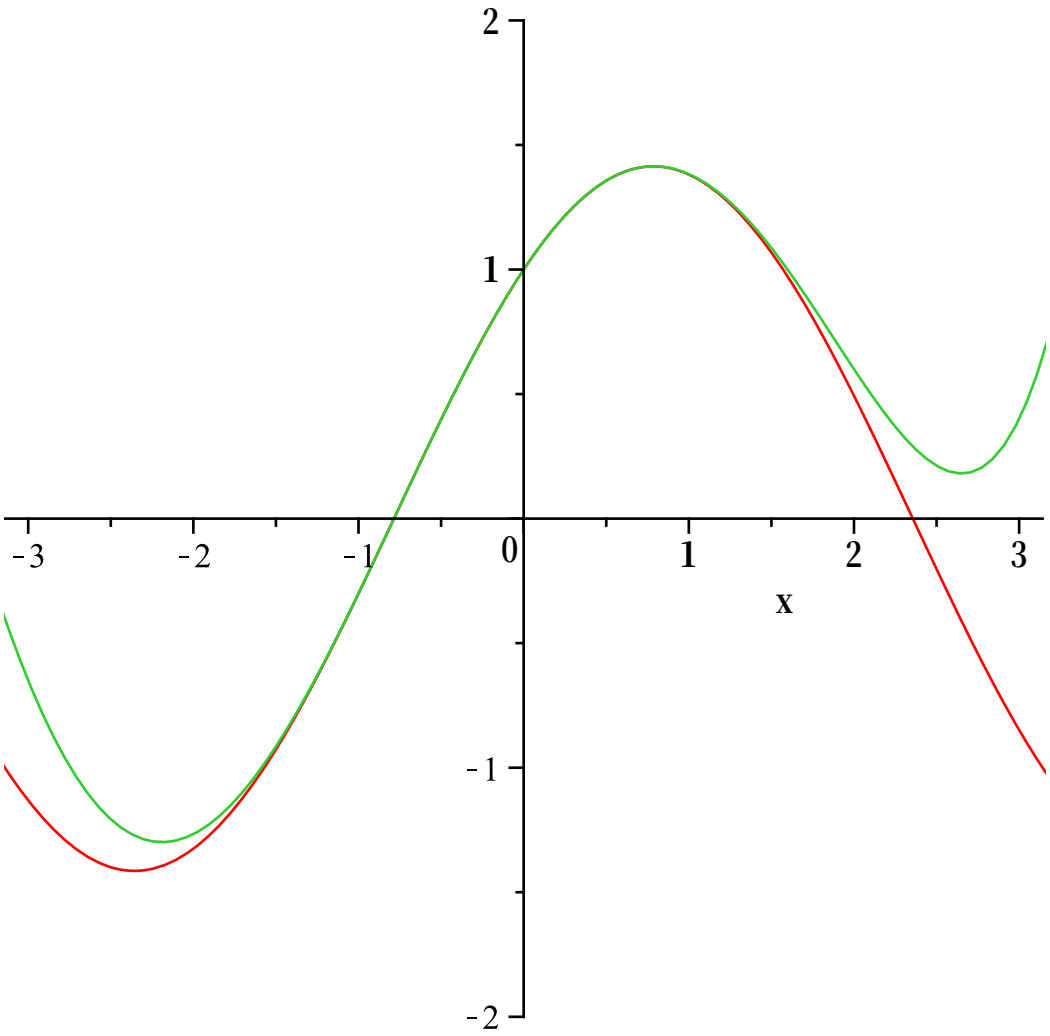
$$ser3 := \sin(x)+\cos(x) = 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6) \tag{46}$$

多項式に変換します．

```
> pol3 := convert( rhs(ser3), polynom );
```

$$pol3 := 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 \tag{47}$$

```
> plot( [ lhs(ser3), pol3 ], view = [ -Pi..Pi, -2..2 ], numpoints=300 );
```



以下のように入れ子構造（ネスト構造）で処理を１行にまとめます．

（注意）以下は，テキストによる記述です．実行はできません．

```
> pol3 := convert( rhs(ser3), polynom );
```

rhs(ser3)

$$= 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6)$$

```
> pol3 := convert( sin(x) + cos(x), polynom );
```

sin(x) + cos(x)

$$= 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6)$$

```
> pol3 := convert( series( sin(x) + cos(x), x=0, 6 ), polynom );
```

series(sin(x) + cos(x), x=0, 6)

$$= 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6)$$

最終的に，コマンドのネスト構造が作成されます．

rhs(ser3) = series(sin(x) + cos(x), x=0, 6)**として，コマンドを以下のように書き換えます．**

```
> pol3 := convert( series( sin(x) + cos(x), x=0, 6 ), polynom );
```

$$pol3 := 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 \tag{48}$$

一度で級数展開から多項式への変換が実行されます．さらに，以下のようにパラメータ化します．

関数

```
> f := sin(x) + cos(x);
```

$$f := \sin(x) + \cos(x) \tag{49}$$

展開点

```
> p := 0;
```

$$p := 0 \tag{50}$$

次数

```
> k := 6;
```

$$k := 6 \tag{51}$$

```
> pol3 := convert( series( f, x=p, k ), polynom );
```

$$pol3 := 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 \tag{52}$$

いったん，以下のような処理にまとめます．

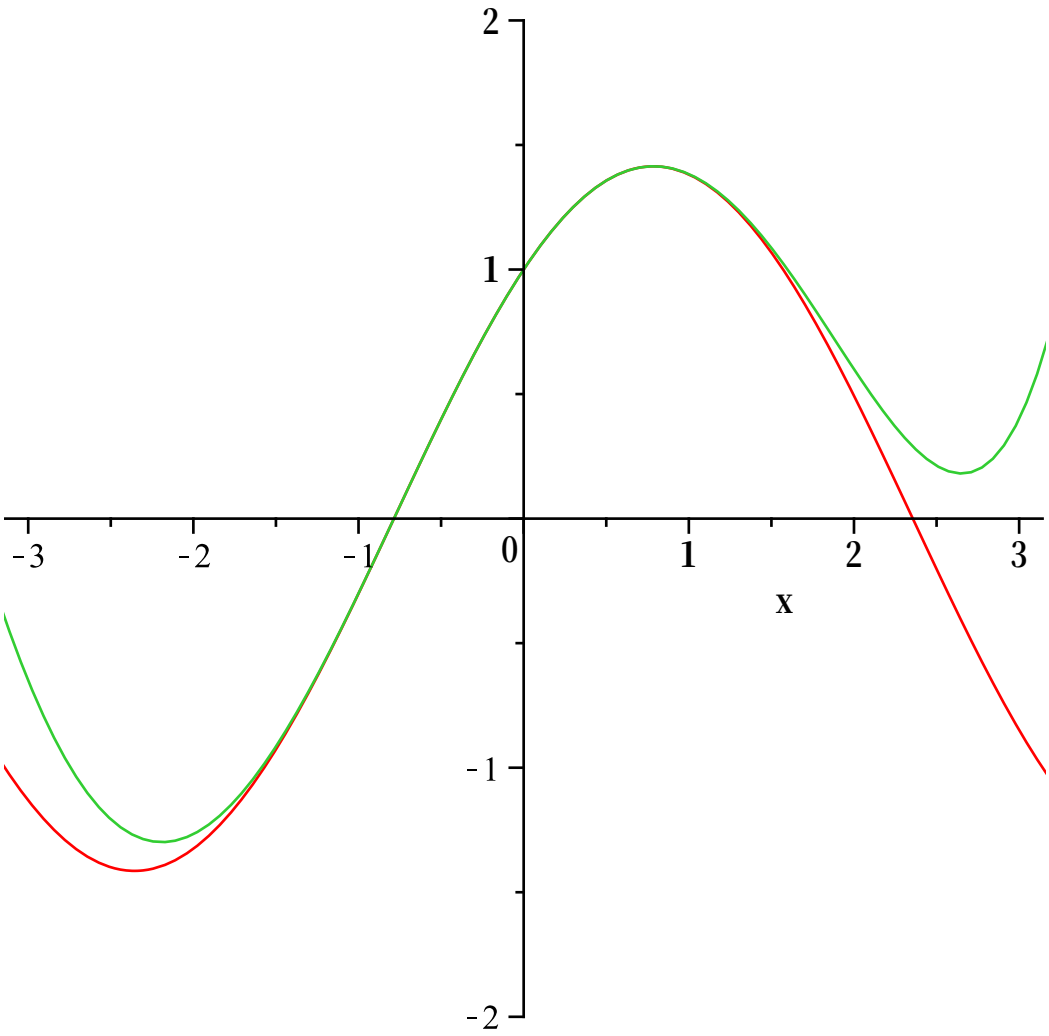
```
> f := sin(x) + cos(x);
  p := 0;
  k := 6;
  pol3 := convert( series( f, x=p, k ), polynom );
```

$$f := \sin(x) + \cos(x)$$
$$p := 0$$
$$k := 6$$
$$pol3 := 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 \tag{53}$$

次に，プロットコマンドと組み合わせます．

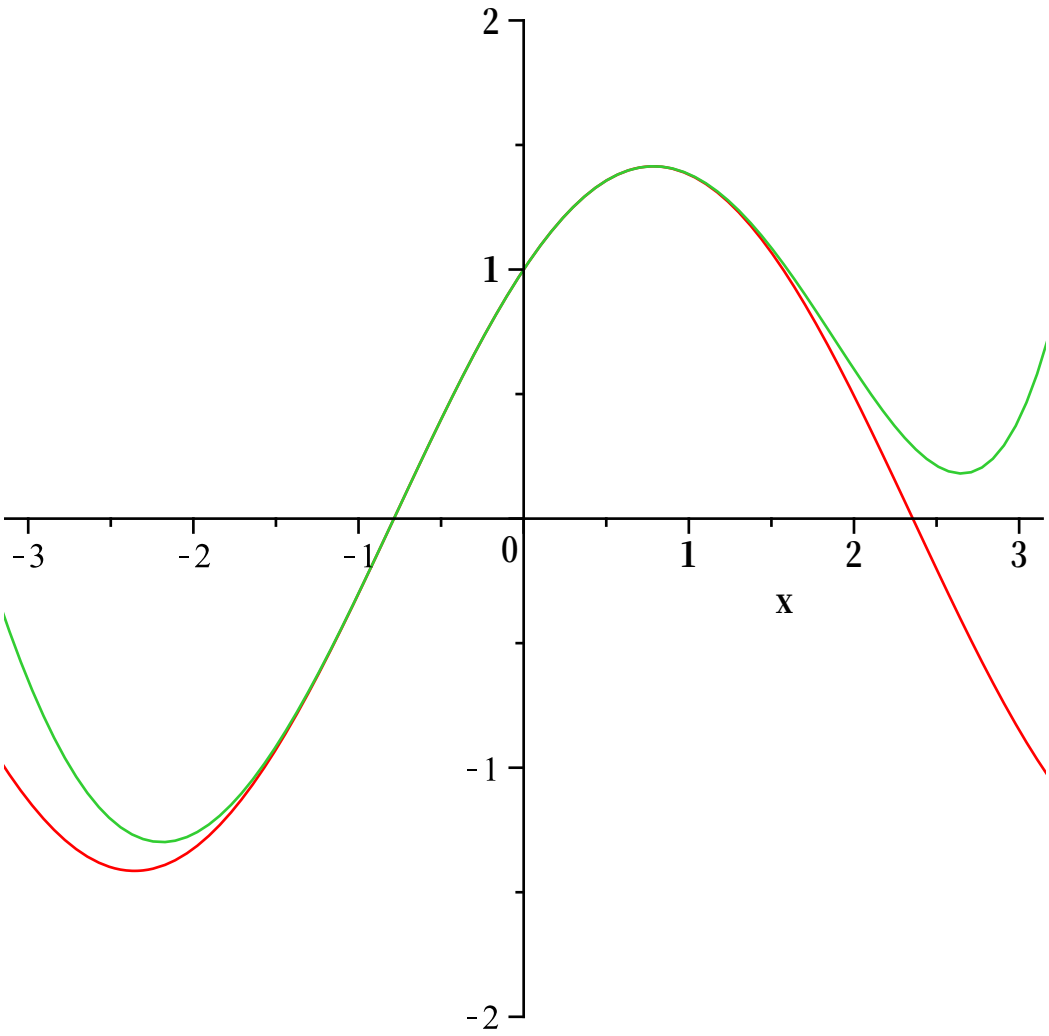
オプション numpoints=300 はプロットのポイント数を指定します（ポイント数が増えると線が滑らかになります）．

```
> plot( [ lhs(ser3), pol3 ], view = [ -Pi..Pi, -2..2 ], numpoints=300 );
```



ただし，rhs(ser3)は，f そのものになりますので，以下のように書き換えます．

```
> plot( [ f, pol3 ], view = [ -Pi..Pi, -2..2 ], numpoints=300 );
```



ここで，以下のように整理します．

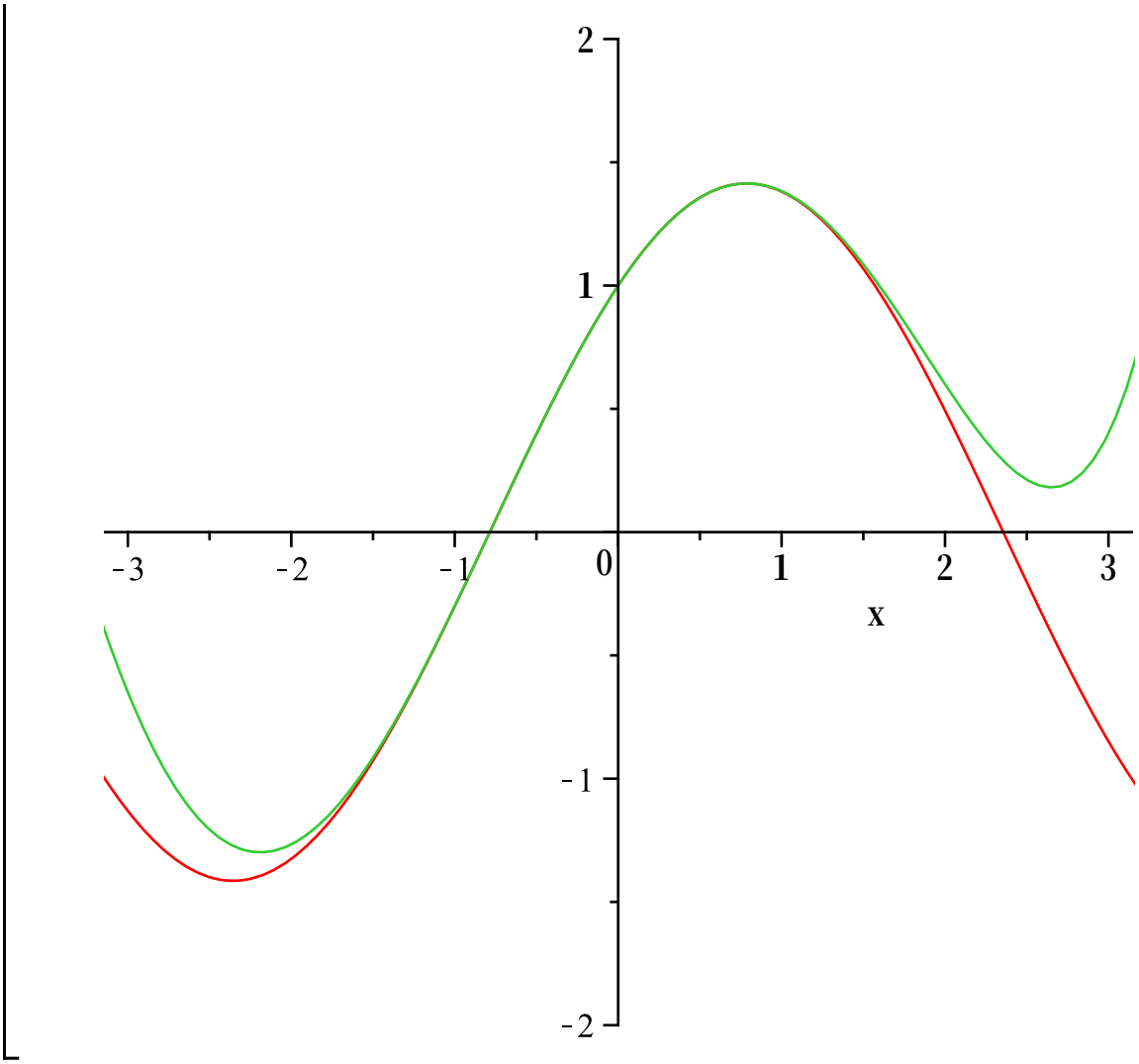
```
> f := sin(x) + cos(x);
  p := 0;
  k := 6;
  pol3 := convert( series( f, x=p, k ), polynom );

  plot( [ f, pol3 ], view = [ -Pi..Pi, -2..2 ], numpoints=300 );
```

$$f := \sin(x) + \cos(x)$$

$$p := 0$$

$$k := 6$$

$$pol3 := 1 + x - \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5$$



ここで，次数 k を 1 から 3 まで変化させ（k=1..3），展開後の多項式を式列で作成します．

```
> seq3 := seq( convert( series( f, x=p, k ), polynom ), k=1..3 );
```

$$seq3 := 1, 1+x, 1+x-\frac{1}{2}x^2 \tag{54}$$

さらに，1 から 24 まで（k=1..24）の多項式列を作成します（数が多いので，コロン : で入力を閉じます）．

```
> seq24 := seq( convert( series( f, x=p, k ), polynom), k=1..24 ):
```

同じ要領で，各次数に対するプロットを作成します．

```
> plt1 := seq( plot( [ f, convert( series( f, x=p, k ), polynom ) ], view = [
  -Pi..Pi, -2..2 ], numpoints=300 ), k=1..3 );
```

$$plt1 := PLOT(\ldots), PLOT(\ldots), PLOT(\ldots) \tag{55}$$

```
> plt2 := [ seq( plot( [ f, convert( series( f, x=p, k ), polynom ) ], view =
  [ -Pi..Pi, -2..2 ], numpoints=300 ), k=1..3 ) ];
```

$$plt2 := [PLOT(\ldots), PLOT(\ldots), PLOT(\ldots)] \tag{56}$$

（復習） 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．PLOT(...) をプロットする場合，plots パッケージの display コマンドを使用します．また，display コマンドに複数の PLOT(...) を渡す場合リストにする必要があります．角括弧 [　] で複数の PLOT(...) を括るとリストになります．

display コマンドを用いて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．

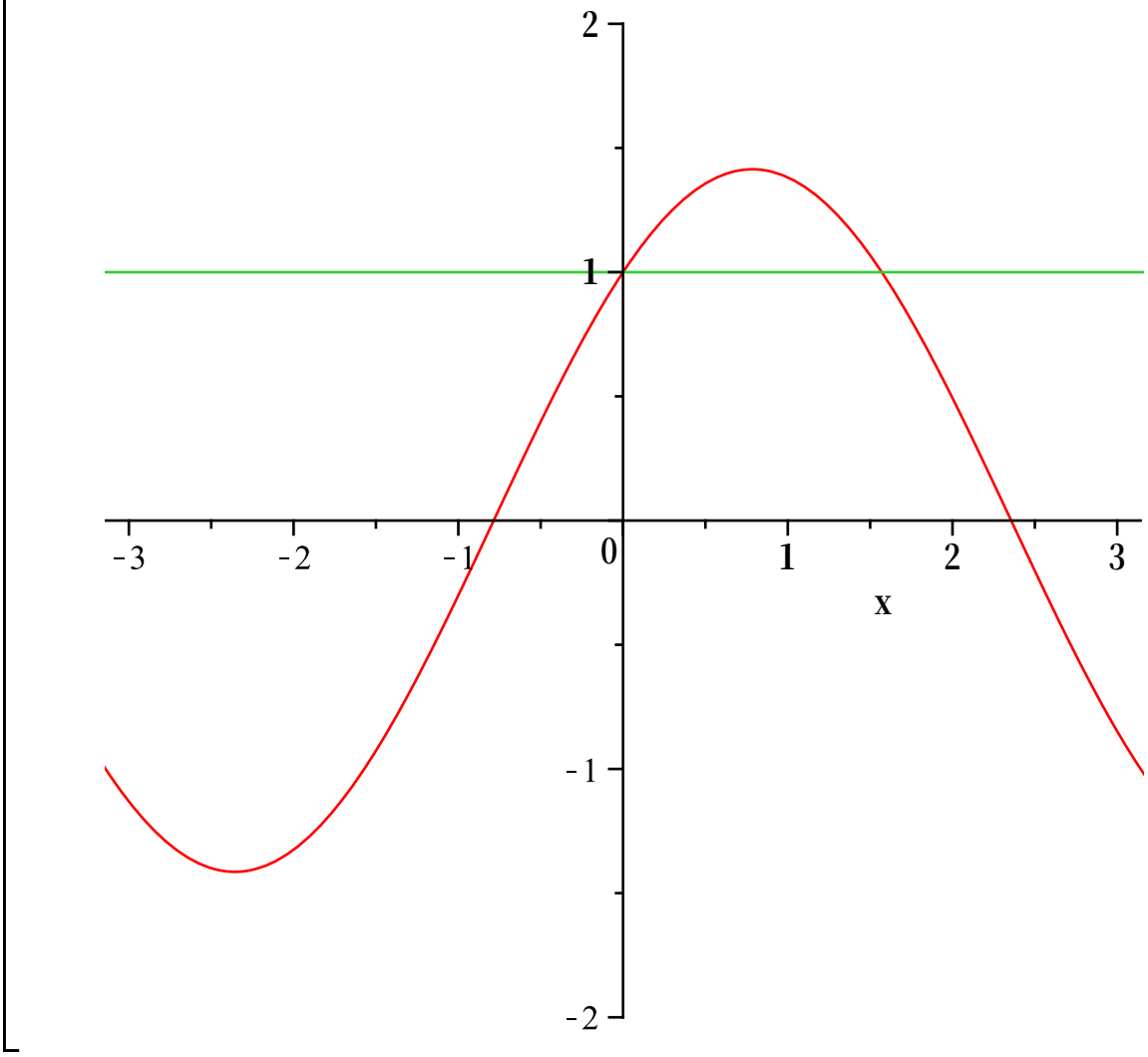
```
> with( plots ):
```

insequence = trueを指定すると，

$plt2 := [PLOT(\ldots), PLOT(\ldots), PLOT(\ldots)]$

の順番でプロットイメージをアニメーションに変換します．

```
> display( plt2, insequence=true );
```



（参考）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．

次数を 6 にします．

```
> N := 6;
```

$$N := 6 \tag{57}$$

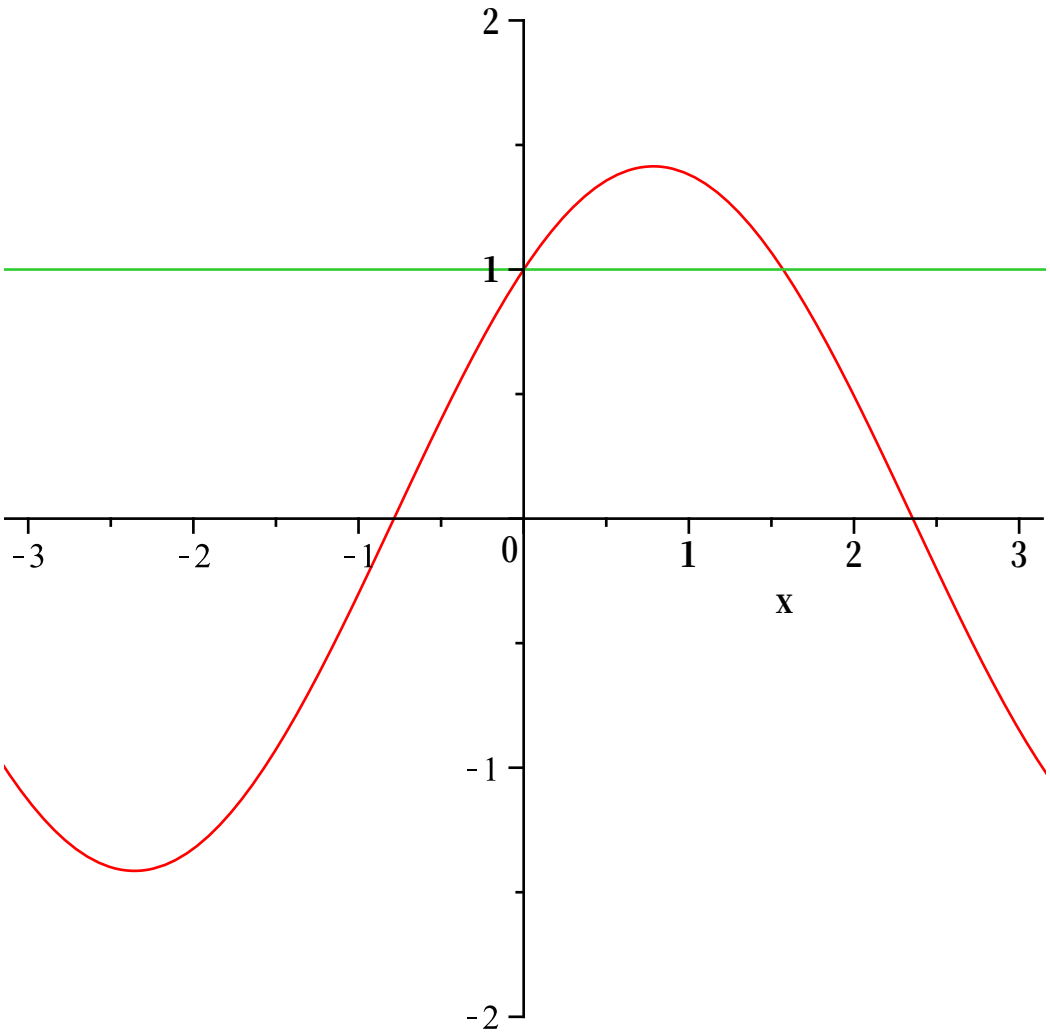
k=1..Nとして，プロットイメージを作成します．

```
> plt2 := [ seq( plot( [ f, convert( series( f, x=p, k ), polynom ) ], view =
  [ -Pi..Pi, -2..2 ], numpoints=300 ), k=1..N ) ];
```

$$plt2 := [PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots)] \tag{58}$$

displayコマンドでアニメーションを作成します．

```
> display( plt2, insequence=true );
```



以上の処理を整理します.

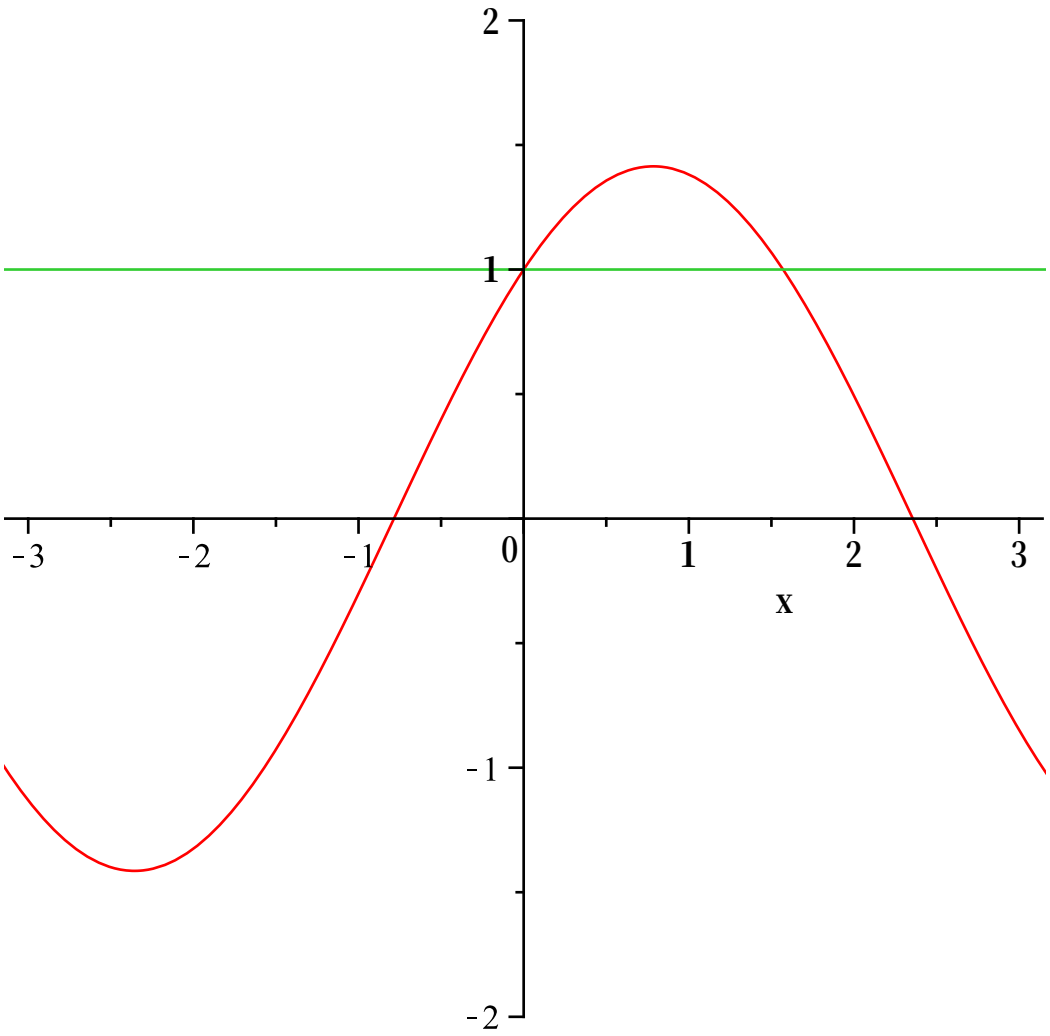
```
> f := sin(x) + cos(x);
  p := 0;
  N := 6;

  plt2 := [ seq( plot( [ f, convert( series( f, x=p, k ), polynom ) ], view =
  [ -Pi..Pi, -2..2 ], numpoints=300 ), k=1..N ) ];

  display(plt2,insequence=true);
```

$$f := \sin(x) + \cos(x)$$

$$p := 0$$

$$N := 6$$

$$plt2 := [PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots)]$$



最後に，x の範囲もパラメータ化します．

```
> f := sin(x) + cos(x);
  p := 0;
  N := 15;
  xmin := -2*Pi;
  xmax :=  2*Pi;

  plt2 := [ seq( plot( [ f, convert( series( f, x=p, k ), polynom ) ], view =
  [ xmin. .xmax, -2..2 ], numpoints=300 ), k=1..N ) ];

  display( plt2, insequence=true );
```

$$f := \sin(x) + \cos(x)$$

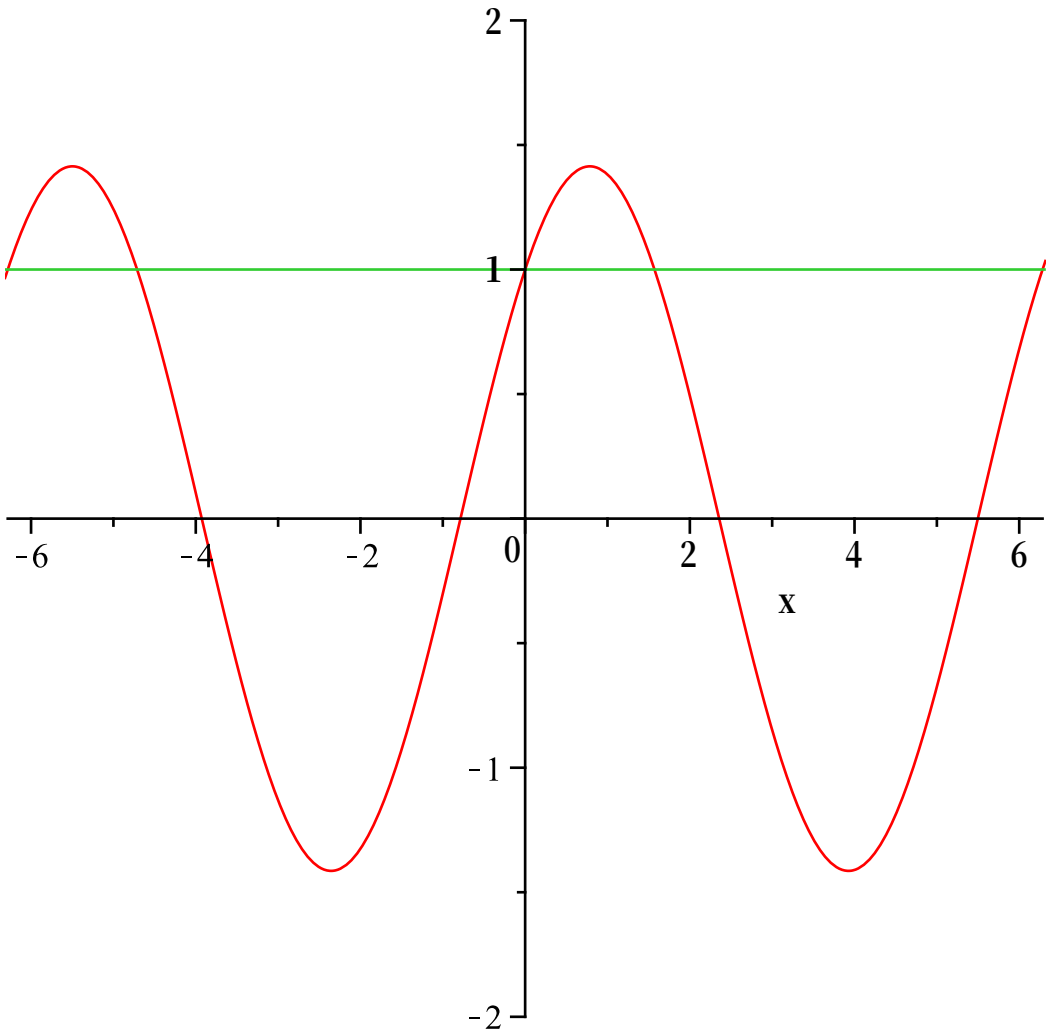
$$p := 0$$

$$N := 15$$

$$xmin := -2\pi$$

$$xmax := 2\pi$$

$$plt2 := [PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots)]$$



例えば，f に設定した関数 $\sin(x)+\cos(x)$ を $\sin(x)+\cos(x)^2$ に変更します.

```
> f := sin(x) + cos(x)^2;
  p := 0;
  N := 24;
  xmin := -2*Pi;
  xmax :=  2*Pi;

  plt2 := [ seq( plot( [ f, convert( series( f, x=p, k ), polynom ) ],
                    view = [ xmin..xmax, -2..2 ],
                    numpoints=300 ),
                k=1..N ) ];

  display( plt2, insequence=true );
```

$$f := \sin(x) + \cos(x)^2$$

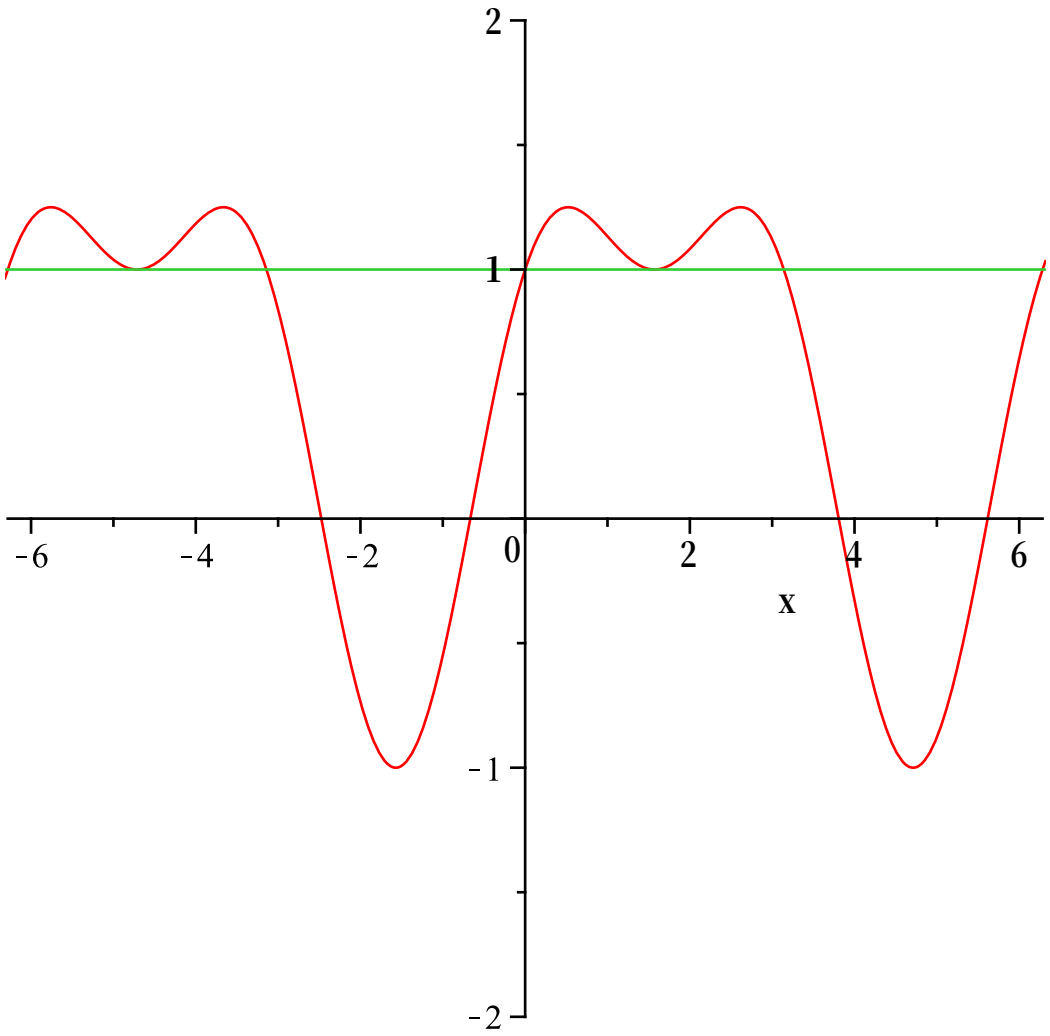
$$p := 0$$

$$N := 24$$

$$xmin := -2\pi$$

$$xmax := 2\pi$$

$$plt2 := [PLOT(\ldots), PLOT(\ldots)]$$



例えば，次数を 6 次に固定して，展開点だけを x = -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6 のように変化させます．
これは，次数を 6 次に固定したまま展開点を移動させるようなアニメーションになります．

```
> f := sin(x) + cos(x);
  p := 0;
  k := 6; # 次数
  Pmin := -6; # 展開点の最小値
  Pmax :=  6; # 展開点の最大値
  xmin := -3*Pi;
  xmax :=  3*Pi;

  plt2 := [ seq( plot( [ f, convert( series( f, x=p, k ), polynom )],
                       view = [ xmin..xmax, -2..2 ],
                       numpoints=300 ),
            p=Pmin. .Pmax ) ];

  display( plt2, insequence=true );
```

$$f := \sin(x) + \cos(x)$$

$$p := 0$$

$$k := 6$$

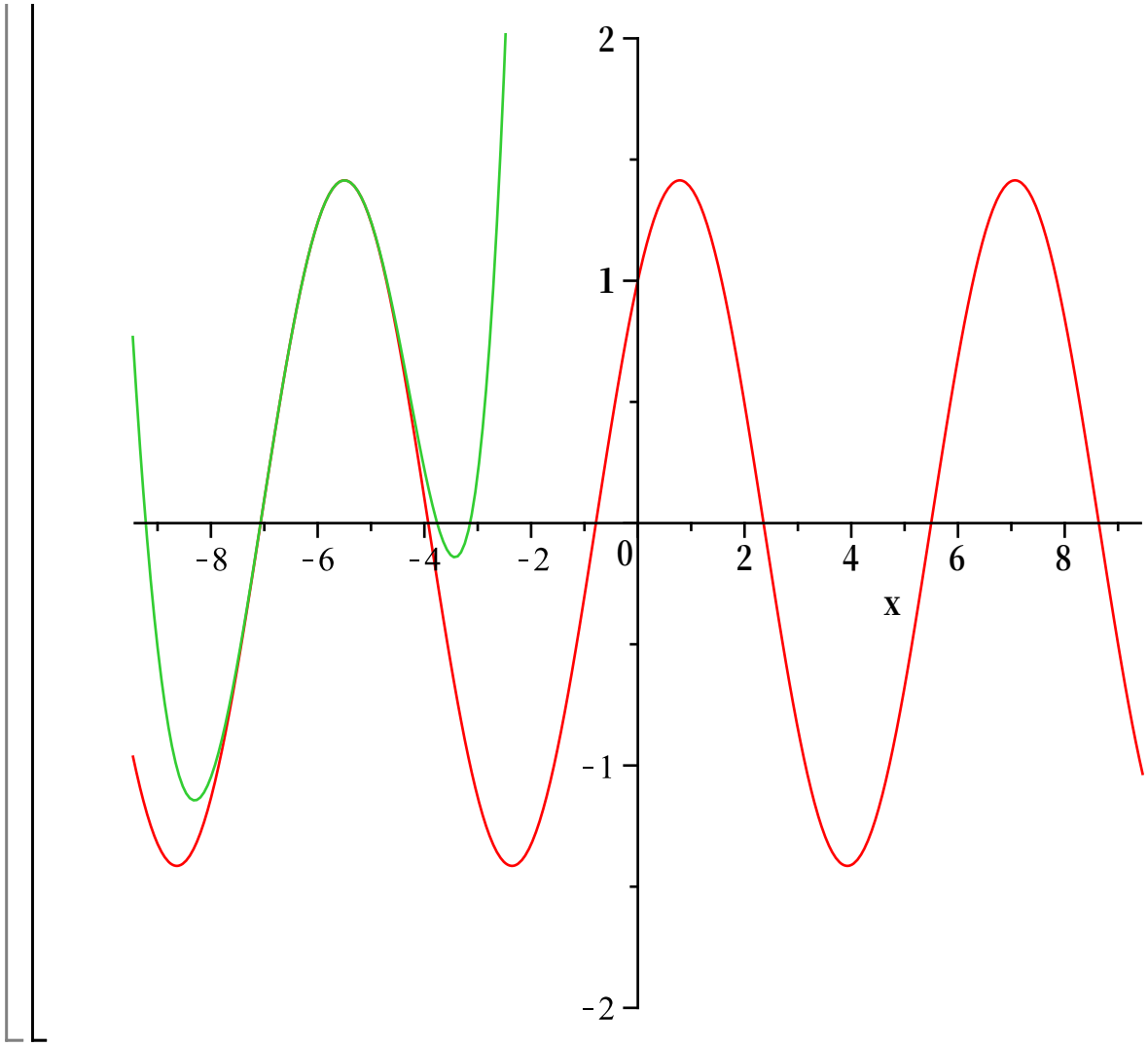
$$Pmin := -6$$

$$Pmax := 6$$

$$xmin := -3\pi$$

$$xmax := 3\pi$$

$$plt2 := [PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots), PLOT(\ldots)]$$


2
1
-1
-2
-8
-6
-4
-2
0
2
4
6
8
x

積分法

不定積分

初期化します.

```
> restart;
```

(例) $af(x)$ の不定積分（a は定数）

```
> int( a*f(x), x );
```

$$\int af(x)\,dx \tag{59}$$

```
> simplify( (59) );
```

$$a\left(\int f(x)\,dx\right) \tag{60}$$

(例) 関数の和$f(x)+g(x)$ の（不定）積分

```
> int( f(x)+g(x), x );
```

$$\int (f(x)+g(x))\,dx \tag{61}$$

```
> expand( (61) );
```

$$\int f(x)\,dx+\int g(x)\,dx \tag{62}$$

(例) 1 の不定積分
（通常，$x+C$のように積分定数（C）が記号としてつきますが， Maple の場合，表示されません．）

```
> int( 1, x );
```

$$x \tag{63}$$

(例) x^aの不定積分

```
> int( x^a, x );
```

$$\frac{x^{a+1}}{a+1} \tag{64}$$

(例) $\frac{1}{x}$の不定積分

```
> int( 1/x, x );
```

$$\ln(x) \tag{65}$$

(例) $\frac{f'(x)}{f(x)}$の不定積分

```
> int( ( diff( f(x), x )/f(x) ), x );
```

$$\ln(f(x)) \tag{66}$$

(例) e^{ax}の不定積分

```
> int( exp( a*x ), x );
```

$$\frac{e^{ax}}{a} \tag{67}$$

(例) $\sin(ax)$ の不定積分

```
> int( sin( a*x ), x );
```

(68)

$$-\frac{\cos(a\,x)}{a} \tag{68}$$

（例） $\cos(a\,x)$ **の不定積分**

```
> int( cos( a*x ), x );
```

$$\frac{\sin(a\,x)}{a} \tag{69}$$

（例） $\frac{1}{\cos(x)^2}$ **の不定積分**

```
> int( 1/cos(a*x)^2, x );
```

$$\frac{\sin(a\,x)}{\cos(a\,x)\,a} \tag{70}$$

式 (70) を tan **の形に変換します．**

```
> convert( (70), tan );
```

$$\frac{\tan(a\,x)}{a} \tag{71}$$

（例）通常，置換積分法によって積分する問題

```
> f1 := sin(x)*cos(x)^3;
```

$$f1 := \sin(x)\cos(x)^3 \tag{72}$$

```
> int( f1, x );
```

$$-\frac{\cos(x)^4}{4} \tag{73}$$

（例）通常，部分積分法によって積分する問題

```
> f2 := x*sin(x);
```

$$f2 := x\sin(x) \tag{74}$$

```
> int( f2, x );
```

$$\sin(x) - x\cos(x) \tag{75}$$

（例）通常，有理関数（分子･分母が多項式の関数）の問題例
（部分分数の形に分解して積分を行います．）

```
> f3 := x/(x^2+x-2);
```

$$f3 := \frac{x}{x^2+x-2} \tag{76}$$

```
> int( f3, x );
```

$$\frac{2\ln(x+2)}{3} + \frac{\ln(x-1)}{3} \tag{77}$$

（参考）部分分数に変換後，積分を実行します．

```
> f3a := convert( f3, parfrac );
```

$$f3a := \frac{1}{3\,(x-1)} + \frac{2}{3\,(x+2)} \tag{78}$$

```
> int( f3a, x );
```

$$\frac{2\ln(x+2)}{3} + \frac{\ln(x-1)}{3} \tag{79}$$

定積分

初期化します．

```
> restart;
```

定積分の記述は以下のように，積分範囲を指定します．

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

$$\int_a^b f(x)\,dx \tag{80}$$

1 の定積分になります．

```
> int( 1, x=1..3 );
```

$$2 \tag{81}$$

指数関数の定積分になります．

```
> int( exp(x), x=0..2 );
```

$$-1+e^2 \tag{82}$$

（参考）大文字 I で始まる積分コマンド Int を用いると，評価前の形式で結果が表示されます（イナート形式あるいは不活性形式と呼ばれます）．評価を進める場合は，value コマンドを使用します．

```
> Int( exp(x), x=0..2 );
```

$$\int_0^2 e^x\,dx \tag{83}$$

```
> value( (83) );
```

$$-1+e^2 \tag{84}$$

積分の応用

線の積分

初期化します.

```
> restart;
```

曲線の長さ

関数 $y=f(x)$ は区間 $[a,b]$ で連続とする．このときの曲線の長さ L は

$$L=\int_a^b \sqrt{1+\{f'(x)\}^2}\,dx$$

である.

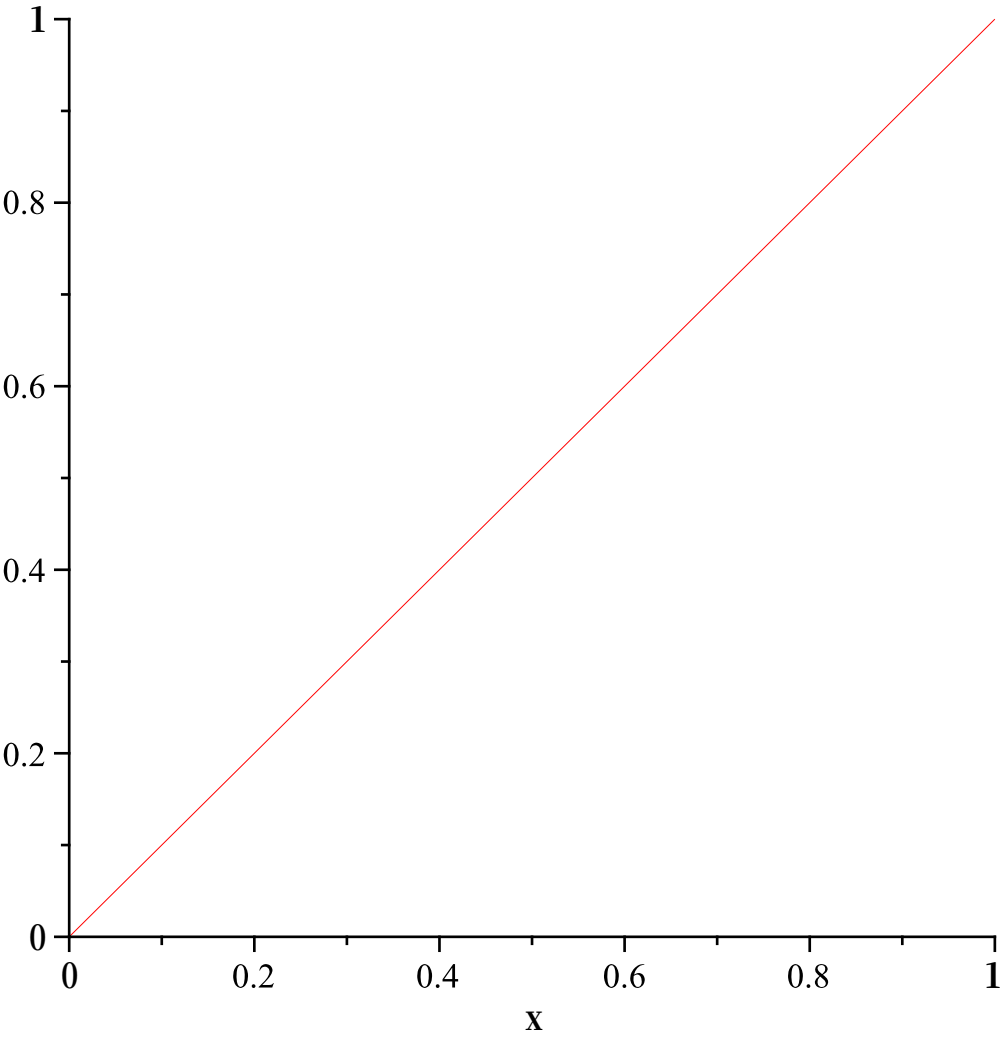
（例1） 直線 $y=x\ (0\le x\le 1)$ の長さ L を求めます.

```
> y1 := x;
```

$$y1 := x \tag{85}$$

y1を確認します（プロットします）.

```
> plot( y1, x=0..1 );
```



$L=\int_a^b \sqrt{1+\{f'(x)\}^2}\,dx$ に代入し，評価します．

$f'(x)$ に代入する dy1 を計算します．

```
> dy1 := diff( y1, x );
```

$$dy1 := 1 \tag{86}$$

$\sqrt{1+\{f'(x)\}^2}$ を iy1 として計算します．

```
> iy1 := sqrt( 1 + dy1^2 );
```

$$iy1 := \sqrt{2} \tag{87}$$

最後に $L=\int_a^b \sqrt{1+\{f'(x)\}^2}\,dx$ を計算します．範囲 x = a..b は x = 0..1 になります．

```
> L1 := int( iy1, x=0..1 );
```

$$L1 := \sqrt{2} \tag{88}$$

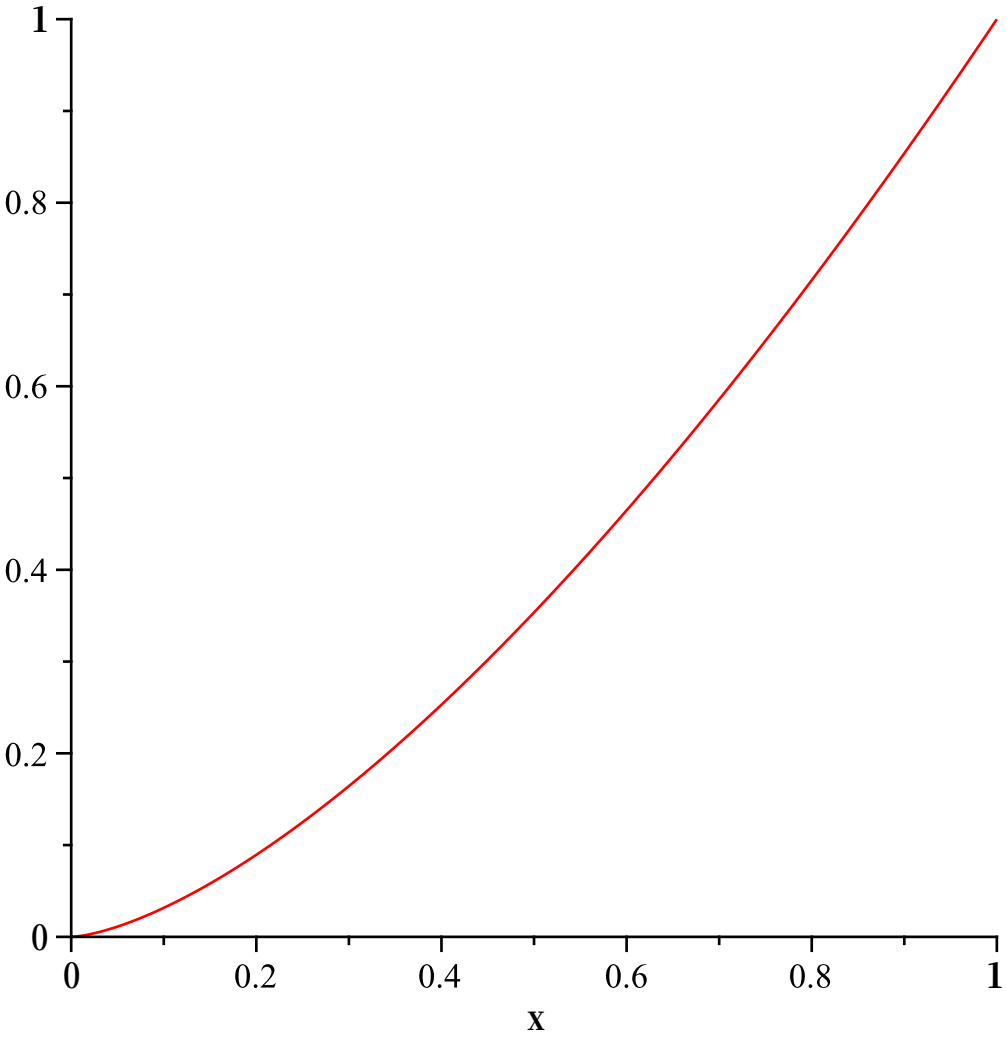
（例2）次に，曲線 $y=x\sqrt{x}$ $(0\le x\le 1)$ の長さ L を求めます．

```
> y2 := x*sqrt( x );
```

$$y2 := x^{3/2} \tag{89}$$

y2 をプロットします．

```
> plot( y2, x=0..1 );
```



$\sqrt{1+\{f'(x)\}^2}$ を iy2 として計算します.

```
> iy2 := sqrt( 1 + ( diff(y2, x) )^2 );
```

$$iy2 := \frac{\sqrt{4+9x}}{2} \tag{90}$$

最後に $L=\int_a^b \sqrt{1+\{f'(x)\}^2}\,dx$ を計算します. 範囲 x=a..b は x=0..1 になります.

```
> L2 := int( iy2, x=0..1 );
```

$$L2 := -\frac{8}{27} + \frac{13\sqrt{13}}{27} \tag{91}$$

式(91)を簡単化します.

```
> L2 := simplify( L2 );
```

$$L2 := -\frac{8}{27} + \frac{13\sqrt{13}}{27} \tag{92}$$

(参考) 定義式のファンクション化

-> はアロー演算子になります(詳細は『プログラミング前夜』で説明します).

```
> L := ( y, a, b ) -> int( sqrt( 1 + ( diff(y, x) )^2 ), x = a .. b );
```

$$L := (y,a,b) \to \int_a^b \sqrt{1+\left(\frac{\partial}{\partial x}y\right)^2}\,dx \tag{93}$$

はじめに, y1 および y2 を確認します.

(例1)の曲線(直線)

```
> y1;
```

$$x \tag{94}$$

(例2)の曲線

```
> y2;
```

$$x^{3/2} \tag{95}$$

定義したファンクション L を x=0..1 の範囲で y1 および y2 それぞれに適用します.

(例1)の曲線(直線)の長さ

```
> L( y1, 0, 1 );
```

$$\sqrt{2} \tag{96}$$

(例2)の曲線の長さ

```
> L( y2, 0, 1 );
```

$$-\frac{8}{27} + \frac{13\sqrt{13}}{27} \tag{97}$$

(例1)式 $L1 := \sqrt{2}$ を数値化

```
> evalf( (88) );
```

$$1.414213562 \tag{98}$$

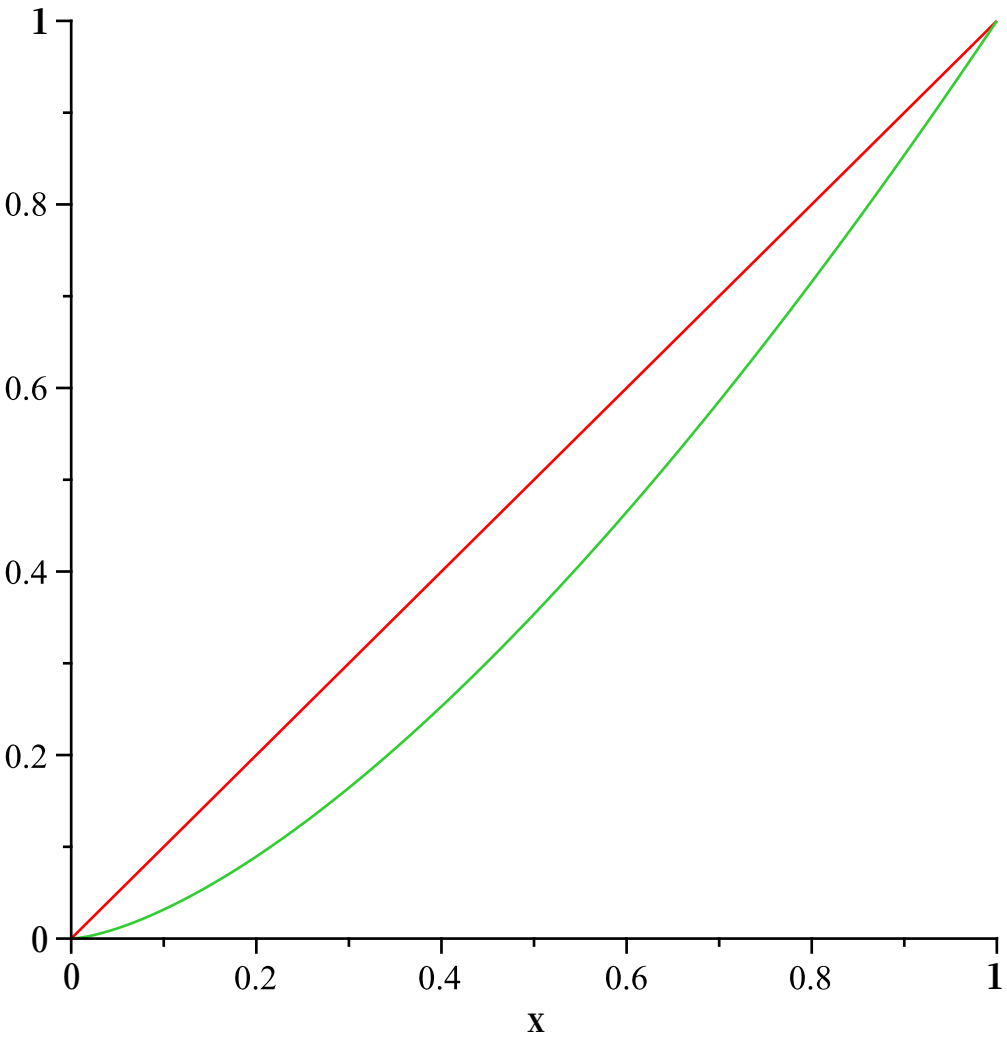
（例２）式 $L2 := -\frac{8}{27} + \frac{13\sqrt{13}}{27}$ を数値化

```
> evalf( (92) );
```

$$1.439709874 \tag{99}$$

式 $y1 := x$ 及び式 $y2 := x^{3/2}$ を x = 0..1 の範囲でプロットします．

```
> plot( [ y1, y2 ], x=0..1 );
```



回転面の面積

初期化します．

```
> restart;
```

回転面の面積

関数 $y=f(x)$ は区間 $[a,b]$ で連続で $f(x) \geq 0$ とする．このとき，$y=f(x)$ を x 軸のまわりに回転してできる回転面の面積 S は，

$$S = 2\pi \left(\int_a^b f(x) \sqrt{1 + f'(x)^2}\, dx \right)$$

である．

回転面の面積を求めるファンクション S を定義します．

```
> S := ( fx, a, b ) -> 2*Pi*( int( fx * sqrt( 1 + ( diff( fx, x ) )^2 ), x
```

```
= a .. b) );
```

$$S := (fx, a, b) \to 2\pi \left(\int_a^b fx \sqrt{1 + \left(\frac{\partial}{\partial x} fx \right)^2} \, dx \right) \tag{100}$$

次の曲線を x 軸のまわりに回転してできる回転面の面積を求めます.

（例１） $y = x^3 \quad (0 \le x \le 1)$

（例２） $y = 2\sqrt{x} \quad (0 \le x \le 1)$

（参考） 半径 r の球の表面積 S

（例１） $y = x^3 \quad (0 \le x \le 1)$

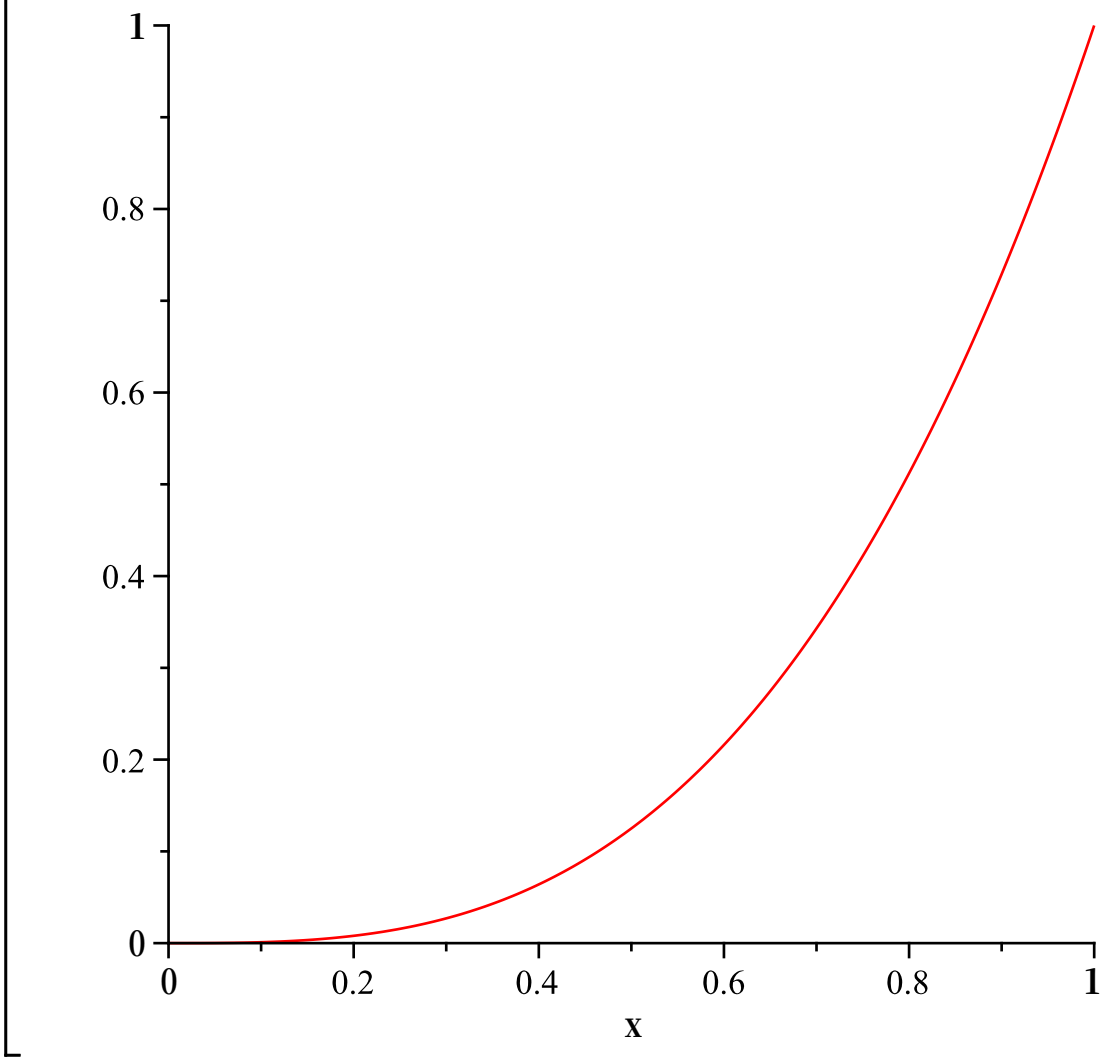
式を y1 として定義します.

```
> y1 := x^3;
```

$$y1 := x^3 \tag{101}$$

y1 を x=0..1 の範囲（区間）でプロットします.

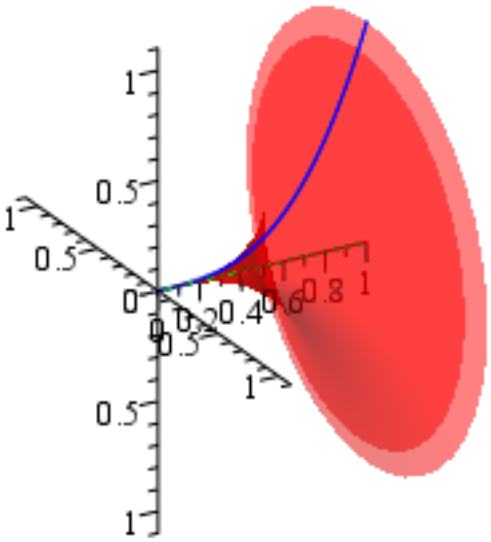
```
> plot( y1, x=0..1 );
```



Student パッケージの Calculus1 サブパッケージに含まれる SurfaceOfRevolution コマンドを用いて，x 軸回りで回転させた場合の表面を３次元プロットします．オプション scaling=constrained は，プロットの縦横比を１：１にします.

```
> plt1 := Student[Calculus1][SurfaceOfRevolution](
              y1,                          # 関数
              x = 0..1,                    # プロットの範囲
             output = plot,                # 出力の指定（プロット）
            transparency = 0.5,            # 不透明 0.0 ～ 1.0 透明
            scaling = constrained ):

  plots[display]( plt1 );
```

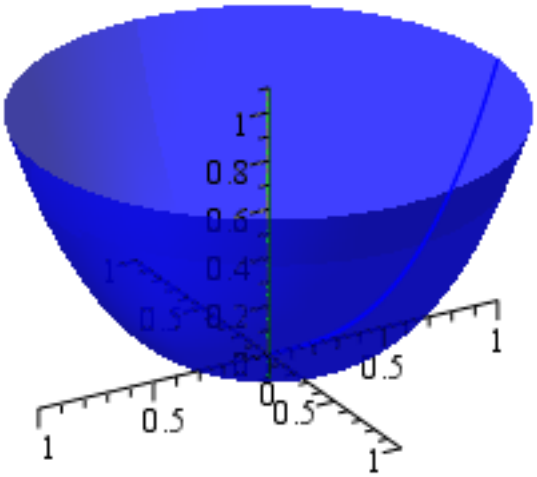
The Surface of Revolution Around the Horizontal Axis of
f(x) = x^3
on the Interval [0, 1]



（参考） *y* **軸回りで回転させた場合**

```
> plt2 := Student[Calculus1][SurfaceOfRevolution](
              y1,                          # 関数
              x = 0..1,                    # プロットの範囲
             output = plot,                # 出力の指定（プロット）
            transparency = 0.5,            # 不透明 0.0 ～ 1.0 透明
             axis = vertical,              # 軸の指定
            scaling = constrained,
              surfaceoptions = [ color=blue ] ): # 色の指定

  plots[display]( plt2 );
```

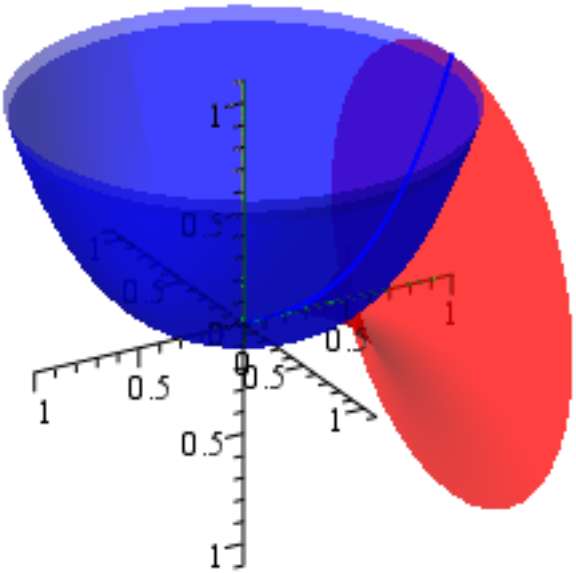
The Surface of Revolution Around the Vertical Axis of
f(x) = x^3
on the Interval [0, 1]



（参考）同時プロット

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

The Surface of Revolution Around the Horizontal Axis of
f(x) = x^3
on the Interval [0, 1]



ファンクションSを用いて，区間［0，1］として関数y1の回転面の面積を求めます．結果をS1に格納します．

```
> S1 := S( y1, 0, 1 );
```

$$S1 := 2\pi\left(\frac{5\sqrt{10}}{27} - \frac{1}{54}\right) \tag{102}$$

式をS1を簡単化します．

```
> simplify( S1 );
```

$$\frac{\pi\left(10\sqrt{10}-1\right)}{27} \tag{103}$$

（参考）Student[Calculus1][SurfaceOfRevolution]コマンドを用いて回転面の面積を求めます．

```
> Student[Calculus1][SurfaceOfRevolution]( y1, x = 0..1 );
```

$$\frac{\pi\left(10\sqrt{10}-1\right)}{27} \tag{104}$$

（例2） $y = 2\sqrt{x}\ \ (0 \le x \le 1)$

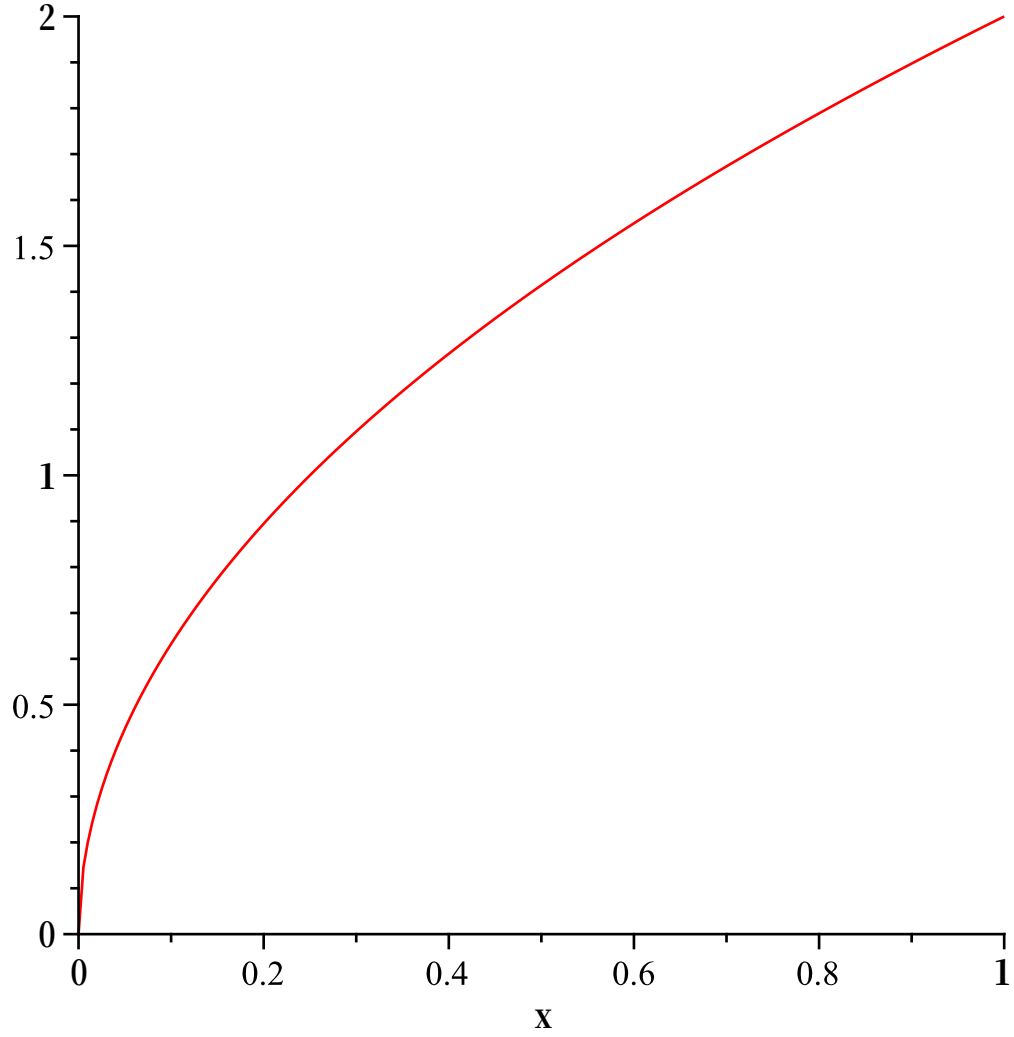
式をy2として定義します．

```
> y2 := 2*sqrt( x );
```

$$y2 := 2\sqrt{x} \tag{105}$$

y2をx=0..1の範囲（区間）でプロットします.

```
> plot( y2, x=0..1 );
```



Studentパッケージの Calculus1 サブパッケージに含まれる SurfaceOfRevolution コマンドを用いて，x 軸回りで回転させた場合の表面（側面）を３次元でプロットします．オプション scaling=constrained は，プロットの縦横比を１：１にします.

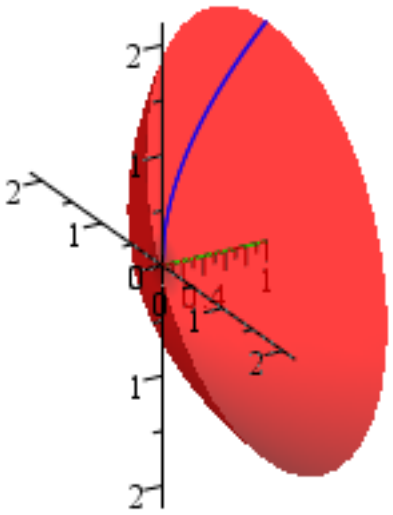
ここで，macro コマンドを用いて，Student[Calculus1][SurfaceOfRevolution] コマンドの短縮形を surfr として定義します．この操作により長いコマンド名を短縮して使用することができるようになります．なお，短縮形の名前は Maple の予約語（例えば，数学定数の Pi や infinity など）以外であれば任意に定義することができます.

```
> macro( surfr = Student[Calculus1][SurfaceOfRevolution] );
```

surfr (106)

```
> plt3 := surfr( y2,                    # 関数
                 x = 0..1,              # プロットの範囲
                 output = plot,         # 出力の指定（プロット）
                 transparency = 0.5,    # 不透明 0.0 ～ 1.0 透明
                 scaling = constrained ):

  plots[display]( plt3 );
```

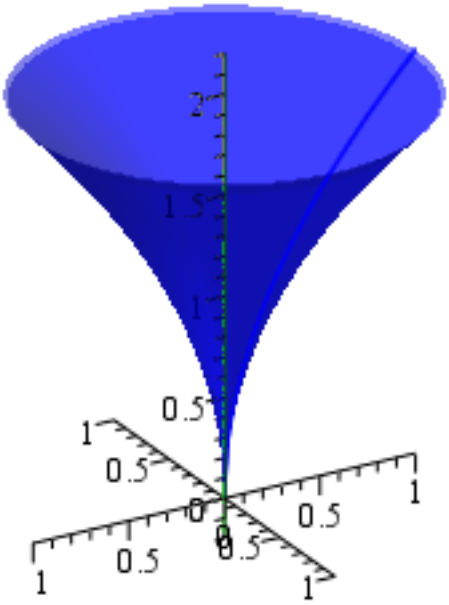
The Surface of Revolution Around the Horizontal Axis of
f(x) = 2*x^(1/2)
on the Interval [0, 1]



（参考）*y*軸回りで回転させた場合（axis = vertical）

```
> plt4 := surfr( y2,                    # 関数
                 x = 0..1,              # プロットの範囲
                output = plot,          # 出力の指定（プロット）
              transparency = 0.5,       # 不透明 0.0 ～ 1.0 透明
                axis = vertical,        # 軸の指定
              scaling = constrained,
                surfaceoptions = [ color=blue ] ): # 色の指定

  plots[display]( plt4 );
```

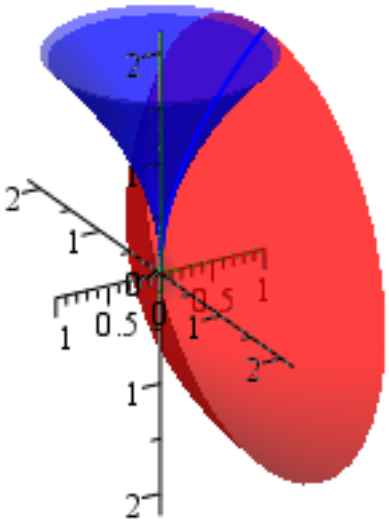
The Surface of Revolution Around the Vertical Axis of
f(x) = 2*x^(1/2)
on the Interval [0, 1]



（参考）同時プロット

```
> plots[display]( [ plt3, plt4 ] );
```

The Surface of Revolution Around the Horizontal Axis of
f(x) = 2*x^(1/2)
on the Interval [0, 1]



ファンクション S を用いて，区間［0， 1］として関数 y2 の回転面の面積を求めます．結果を S2 に格納します．

```
> S2 := S( y2, 0, 1 );
```

$$S2 := 2\pi\left(-\frac{4}{3}+\frac{8\sqrt{2}}{3}\right) \tag{107}$$

式を S2 を簡単化します．

```
> simplify( S2 );
```

$$\frac{8\pi\left(-1+2\sqrt{2}\right)}{3} \tag{108}$$

（参考）Student[Calculus1][SurfaceOfRevolution] コマンドの短縮形 surfr を用いて回転面の面積を求めます．

```
> surfr( y2, x = 0..1 );
```

$$-\frac{8\pi}{3}+\frac{16\sqrt{2}\,\pi}{3} \tag{109}$$

（参考）半径 r の球の表面積 S

半円 $y=\sqrt{r^2-x^2}\ (-r\le x\le r)$ を x 軸のまわりに回転させると球になります．

式を y3 として定義します．

```
> y3 := sqrt( r^2 - x^2 );
```

$$y3 := \sqrt{r^2 - x^2} \tag{110}$$

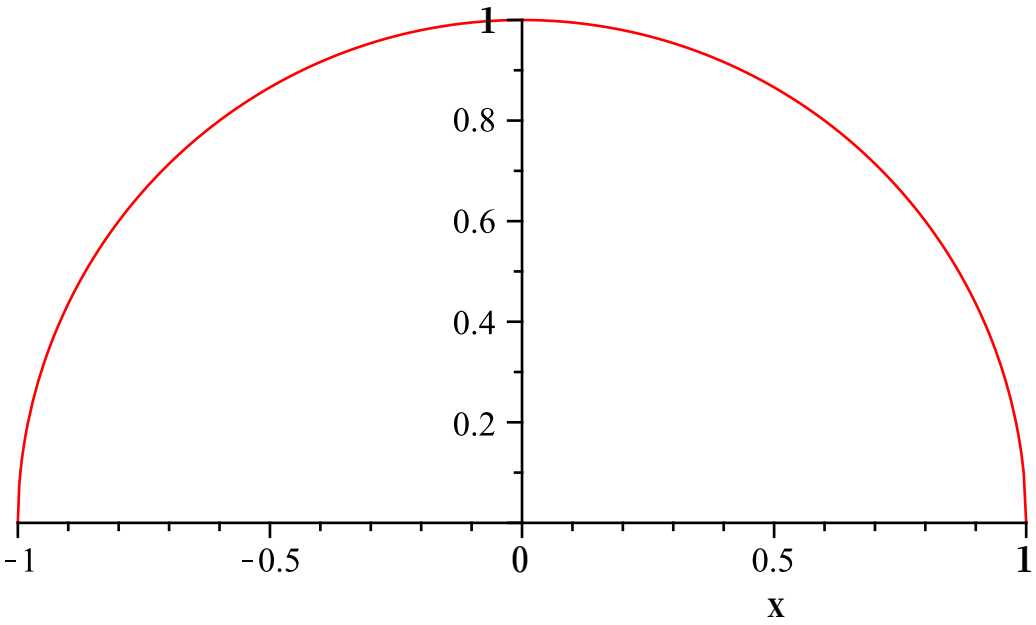
式y3にr=1を代入してy3aに格納します.

```
> y3a := eval( y3, [ r = 1 ] );
```

$$y3a := \sqrt{1 - x^2} \tag{111}$$

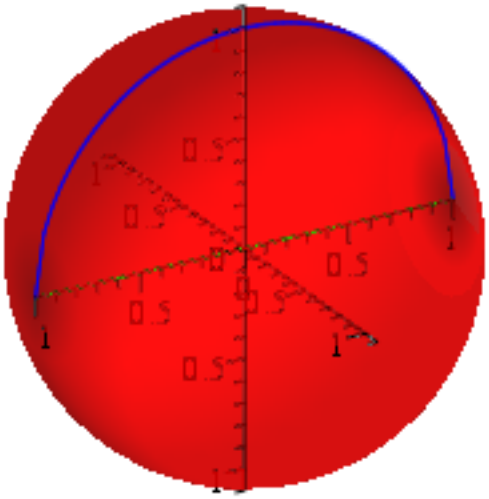
y3aをx=-1..1の範囲でプロットします. オプションscaling=constrainedは, プロットの縦横比を1：1にします.

```
> plot( y3a, x=-1..1, scaling = constrained );
```




```
> Student[Calculus1][SurfaceOfRevolution]( y3a,                    # 関数
                                           x = -1..1,               # プロッ
トの範囲
                                           output = plot,           # 出力の
指定（プロット）
                                           transparency = 0.5,      # 不透明
0.0 ～ 1.0 透明
                                           scaling = constrained );
```

The Surface of Revolution Around the Horizontal Axis of
f(x) = (1-x^2)^(1/2)
on the Interval [-1, 1]



ファンクションSを用いて，区間 [-1, 1] として関数y3aの回転面の面積（球の表面積）を求めます．

```
> S( y3a, -1, 1 );
```

$$4\pi \tag{112}$$

（参考）球の表面積を公式 $4\pi r^2$ に $r = 1$ を代入して求めます．

```
> eval( 4*Pi*r^2, [ r = 1 ] );
```

$$4\pi \tag{113}$$

（参考）Student[Calculus1][SurfaceOfRevolution]コマンドを用いて回転面の面積を求めます．

```
> Student[Calculus1][SurfaceOfRevolution]( y3a, x = -1..1 );
```

$$4\pi \tag{114}$$

２重積分の計算

初期化します．

```
> restart;
```

２重積分

$z = f(x, y)$ のＤ：$a \le x \le b, c \le y \le d$ における２重積分は，

$$\iint_D f(x,y)\,dxdy = \int_a^b \int_c^d f(x,y)\ \mathrm{d}y\,\mathrm{d}x = \int_c^d \int_a^b b(x,y)\ \mathrm{d}y\,\mathrm{d}x$$

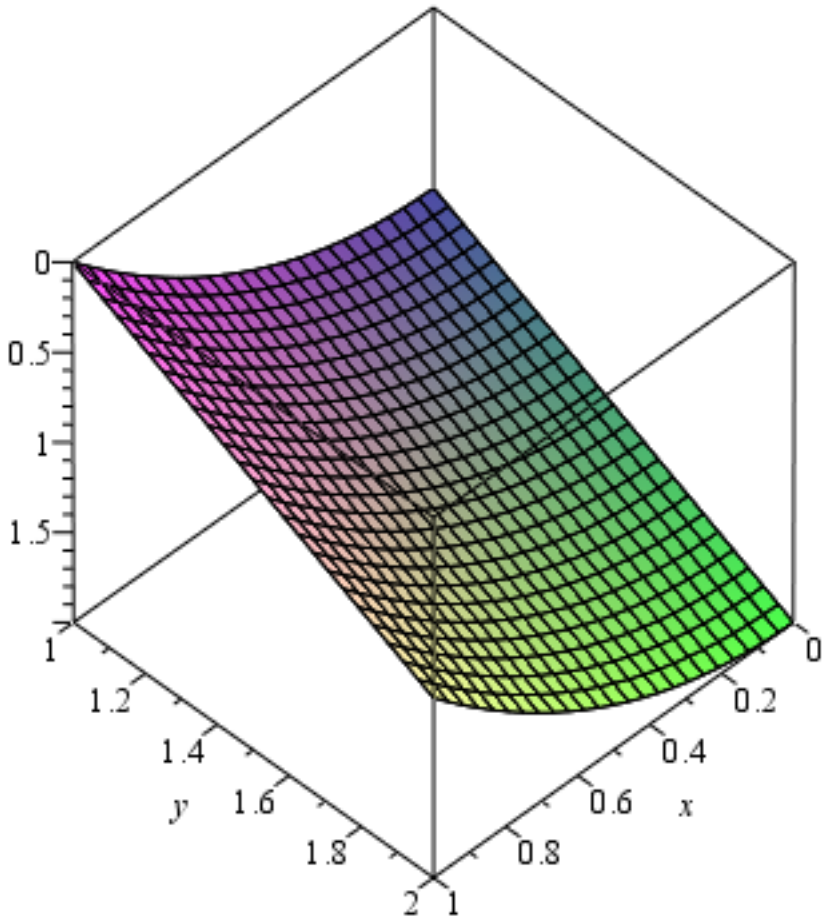
（例） 2重積分 $\iint_D (x^2-y)\,dxdy$ $(\mathrm{D}: 0 \le x \le 1, 1 \le y \le 2)$ **の値を求めます.**

```
> fxy := x^2 - y;
```

$$fxy := x^2 - y \tag{115}$$

（参考）二つの変数 x, y を持つ関数（fxy）は，plot3d コマンドを用いてプロットします．オプションとして，ボックス目盛りを指定します.

```
> plot3d( fxy, x=0..1, y=1..2, axes=boxed );
```



int コマンド用いて，x, y についてそれぞれ定積分を実行します.

```
> V1 := int( int( fxy, x=0..1 ), y=1..2 );
```

$$V1 := -\frac{7}{6} \tag{116}$$

（参考）コマンドの先頭が大文字 I の場合，評価前の形式で結果が表示されます（イナート形式あるいは不活性形式と呼ばれます）.

```
> V2 := Int( Int( fxy, x=0..1 ), y=1..2 );
```

$$V2 := \int_1^2 \int_0^1 (x^2 - y)\ \mathrm{d}x\ \mathrm{d}y \tag{117}$$

valueコマンドを用いて，不活性形式を評価します．

```
> value( V2 );
```

$$-\frac{7}{6} \tag{118}$$

ステップ･バイ･ステップ式 **はじめての Maple 応用編**

いわゆる線形代数

線形代数に関する基礎的な操作手順を習得します.

目次

行列の演算

初期化します.

```
> restart;
```

はじめに線形代数（LinearAlgebra）パッケージをロードします.

```
> with(LinearAlgebra);
```

(1)

```
[&x, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm,
  BilinearForm, CARE, CharacteristicMatrix, CharacteristicPolynomial, Column,
  ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix,
  ConditionNumber, ConstantMatrix, ConstantVector, Copy, CreatePermutation,
  CrossProduct, DARE, DeleteColumn, DeleteRow, Determinant, Diagonal,
  DiagonalMatrix, Dimension, Dimensions, DotProduct, EigenConditionNumbers,
  Eigenvalues, Eigenvectors, Equal, ForwardSubstitute, FrobeniusForm,
  GaussianElimination, GenerateEquations, GenerateMatrix, Generic, GetResultDataType,
  GetResultShape, GivensRotationMatrix, GramSchmidt, HankelMatrix, HermiteForm,
  HermitianTranspose, HessenbergForm, HilbertMatrix, HouseholderMatrix,
  IdentityMatrix, IntersectionBasis, IsDefinite, IsOrthogonal, IsSimilar, IsUnitary,
  JordanBlockMatrix, JordanForm, KroneckerProduct, LA_Main, LUDecomposition,
  LeastSquares, LinearSolve, LyapunovSolve, Map, Map2, MatrixAdd, MatrixExponential,
  MatrixFunction, MatrixInverse, MatrixMatrixMultiply, MatrixNorm, MatrixPower,
  MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial, Minor, Modular,
  Multiply, NoUserValue, Norm, Normalize, NullSpace, OuterProductMatrix, Permanent,
  Pivot, PopovForm, QRDecomposition, RandomMatrix, RandomVector, Rank,
  RationalCanonicalForm, ReducedRowEchelonForm, Row, RowDimension,
  RowOperation, RowSpace, ScalarMatrix, ScalarMultiply, ScalarVector, SchurForm,
  SingularValues, SmithForm, StronglyConnectedBlocks, SubMatrix, SubVector, SumBasis,
  SylvesterMatrix, SylvesterSolve, ToeplitzMatrix, Trace, Transpose, TridiagonalForm,
  UnitVector, VandermondeMatrix, VectorAdd, VectorAngle, VectorMatrixMultiply,
  VectorNorm, VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip]
```

行列の定義

行列はMatrixコマンドを用いて定義します.

行列（$r \times c$）のサイズのみで，要素が行列に与えられない場合，すべての値は0（デフォルト値）で埋め尽くされます.

```
> M22 := Matrix( 2, 2 );
```

$$M22 := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{2}$$

```
> M33 := Matrix( 3, 3 );
```

$$M33 := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3}$$

```
> M23 := Matrix( 2, 3 );
```

$$M23 := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4}$$

（参考）零行列を明示的に定義する場合は，ZeroMatrix コマンドを使用します．

```
> Z33 := ZeroMatrix( 3, 3 );
```

$$Z33 := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5}$$

要素（5）を指定した場合，すべての値がその要素で埋め尽くされます．

```
> M33a := Matrix( 3,3, 5 );
```

$$M33a := \begin{bmatrix} 5 & 5 & 5 \\ 5 & 5 & 5 \\ 5 & 5 & 5 \end{bmatrix} \tag{6}$$

各要素を指定する場合，以下のような定義方法があります．行列の大きさを宣言し，要素をリストとしてコマンドに与えます．

```
> M33b := Matrix( 3,3, [ a, b, c, d, e, f, g, h, i ] );
```

$$M33b := \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{7}$$

変数（記号）で要素を定義する場合，以下のような定義方法があります．

```
> M33c := Matrix( 3,3, symbol=a );
```

$$M33c := \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \tag{8}$$

（参考）< >を用いた行列の定義方法があります．

```
> M33d := < < 1, 2, 3 > | < 4, 5, 6 > | < 7, 8, 9 > >;
```

$$M33d := \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \tag{9}$$

単位行列の定義は IdentityMatrix コマンドを使用します．

```
> E33 := IdentityMatrix( 3 );
```

$$E33 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

対角行列の定義は DiagonalMatrix コマンドを使用します．

```
> D33 := DiagonalMatrix( [ a, b, c ] );
```

$$D33 := \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \tag{11}$$

ランダムに行列の要素を定義する RandomMatrix コマンドがあります．要素は実行するたびに異なります．

```
> M33e := RandomMatrix( 3,3 );
```

$$M33e := \begin{bmatrix} 27 & 99 & 92 \\ 8 & 29 & -31 \\ 69 & 44 & 67 \end{bmatrix} \tag{12}$$

多項式や関数などの式を要素として，行列を定義することができます．

```
> M22f := Matrix( 2,2, [ sin(a*x), cos(b*x), exp(c*x), f(x)] );
```

$$M22f := \begin{bmatrix} \sin(ax) & \cos(bx) \\ e^{cx} & f(x) \end{bmatrix} \tag{13}$$

（参考）全要素にｄｉｆｆコマンドを適用します（ｘについて全要素を２階微分）．

map コマンドの利用

```
> map( diff, M22f, x );
```

$$\begin{bmatrix} \cos(ax)\,a & -\sin(bx)\,b \\ c\,e^{cx} & f'(x) \end{bmatrix} \tag{14}$$

~ チルダ演算子の利用

```
> diff~( M22f, x );
```

$$\begin{bmatrix} \cos(ax)\,a & -\sin(bx)\,b \\ c\,e^{cx} & f'(x) \end{bmatrix} \tag{15}$$

（参考）全要素にｉｎｔコマンドを適用します（ｘについて全要素を積分）．

map コマンドの利用

```
> map( int, M22f, x );
```

$$\begin{bmatrix} -\frac{\cos(ax)}{a} & \frac{\sin(bx)}{b} \\ \frac{e^{cx}}{c} & \int f(x)\,dx \end{bmatrix} \tag{16}$$

~ チルダ演算子の利用

```
> int~( M22f, x );
```

$$\begin{bmatrix} -\frac{\cos(ax)}{a} & \frac{\sin(bx)}{b} \\ \frac{e^{cx}}{c} & \int f(x)\,dx \end{bmatrix} \tag{17}$$

（参考）map コマンドは，行列の各要素に同じ処理を作用させることができます．

行列の定義と演算

行列の定義方法になります．

Matrix コマンドを用いた 2# 2の行列 Aの定義

```
> A := Matrix( 2,2, symbol=a );
```

$$A := \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \tag{18}$$

（参考）演算子< | >を用いた 2# 2の行列の定義

```
> AA := < < a[1,1] | a[1,2] >, < a[2,1] | a[2,2] > >;
```

$$AA := \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \tag{19}$$

Matrixコマンドを用いた 2# 2の行列Bの定義

```
> B := Matrix( 2,2, symbol=b );
```

$$B := \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \tag{20}$$

2# 2の行列C

```
> C := Matrix( 2,2, symbol=c );
```

$$C := \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} \tag{21}$$

行列の四則演算になります.

足し算

```
> A + B;
```

$$\begin{bmatrix} a_{1,1}+b_{1,1} & a_{1,2}+b_{1,2} \\ a_{2,1}+b_{2,1} & a_{2,2}+b_{2,2} \end{bmatrix} \tag{22}$$

（参考）MatrixAddコマンドを用いた足し算

```
> MatrixAdd( A, B );
```

$$\begin{bmatrix} a_{1,1}+b_{1,1} & a_{1,2}+b_{1,2} \\ a_{2,1}+b_{2,1} & a_{2,2}+b_{2,2} \end{bmatrix} \tag{23}$$

引き算

```
> A - B;
```

$$\begin{bmatrix} a_{1,1}-b_{1,1} & a_{1,2}-b_{1,2} \\ a_{2,1}-b_{2,1} & a_{2,2}-b_{2,2} \end{bmatrix} \tag{24}$$

掛け算

（参考）掛け算にはドット（. ）を使用します.

```
> A.B;
```

$$\begin{bmatrix} a_{1,1}b_{1,1}+a_{1,2}b_{2,1} & a_{1,1}b_{1,2}+a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1}+a_{2,2}b_{2,1} & a_{2,1}b_{1,2}+a_{2,2}b_{2,2} \end{bmatrix} \tag{25}$$

逆行列の掛け算

```
> A.B^(-1);
```

$$\left[\left[\frac{a_{1,1}b_{2,2}}{b_{1,1}b_{2,2}-b_{1,2}b_{2,1}}-\frac{a_{1,2}b_{2,1}}{b_{1,1}b_{2,2}-b_{1,2}b_{2,1}},-\frac{a_{1,1}b_{1,2}}{b_{1,1}b_{2,2}-b_{1,2}b_{2,1}}+\frac{a_{1,2}b_{1,1}}{b_{1,1}b_{2,2}-b_{1,2}b_{2,1}}\right],\left[\frac{a_{2,1}b_{2,2}}{b_{1,1}b_{2,2}-b_{1,2}b_{2,1}}-\frac{a_{2,2}b_{2,1}}{b_{1,1}b_{2,2}-b_{1,2}b_{2,1}},-\frac{a_{2,1}b_{1,2}}{b_{1,1}b_{2,2}-b_{1,2}b_{2,1}}+\frac{a_{2,2}b_{1,1}}{b_{1,1}b_{2,2}-b_{1,2}b_{2,1}}\right]\right] \tag{26}$$

$\frac{A}{B}$ を $A\cdot B^{-1}$ として計算します．数学では行列の割り算が定義されていません．

MatrixInverse コマンドを用いた逆行列の演算

```
> MatrixInverse( A );
```

$$\begin{bmatrix}\frac{a_{2,2}}{a_{1,1}a_{2,2}-a_{1,2}a_{2,1}} & -\frac{a_{1,2}}{a_{1,1}a_{2,2}-a_{1,2}a_{2,1}}\\ -\frac{a_{2,1}}{a_{1,1}a_{2,2}-a_{1,2}a_{2,1}} & \frac{a_{1,1}}{a_{1,1}a_{2,2}-a_{1,2}a_{2,1}}\end{bmatrix} \tag{27}$$

A^{-1} も A の逆行列を求めることができます．

```
> A^(-1);
```

$$\begin{bmatrix}\frac{a_{2,2}}{a_{1,1}a_{2,2}-a_{1,2}a_{2,1}} & -\frac{a_{1,2}}{a_{1,1}a_{2,2}-a_{1,2}a_{2,1}}\\ -\frac{a_{2,1}}{a_{1,1}a_{2,2}-a_{1,2}a_{2,1}} & \frac{a_{1,1}}{a_{1,1}a_{2,2}-a_{1,2}a_{2,1}}\end{bmatrix} \tag{28}$$

Determinant コマンドを用いた行列式の計算

```
> Determinant( A );
```

$$a_{1,1}a_{2,2}-a_{1,2}a_{2,1} \tag{29}$$

Rank コマンドを用いたランクの計算

```
> A1 := eval( A, [ a[1,1]=1, a[1,2]=0,
                   a[2,1]=1, a[2,2]=1 ] );
```

$$A1 := \begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix} \tag{30}$$

```
> Rank( A1 );
```

$$2 \tag{31}$$

```
> A2 := eval( A, [ a[1,1]=1, a[1,2]=2,
                   a[2,1]=3, a[2,2]=6 ] );
```

$$A2 := \begin{bmatrix}1 & 2\\ 3 & 6\end{bmatrix} \tag{32}$$

```
> Rank( A2 );
```

(33)

$$1 \tag{33}$$

行列のノルムを計算します．はじめに，３# ３の行列 N1 を定義します．

```
> N1 := Matrix( 3,3, [ a, b, c, d, e, f, g, h, i ] );
```

$$N1 := \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{34}$$

MatrixNorm コマンドを用いたノルムの計算

```
> MatrixNorm( N1, 1 );
```

$$\max(|a| + |d| + |g|, |b| + |e| + |h|, |c| + |f| + |i|) \tag{35}$$

（参考）max コマンドは最大値を求めるコマンドになります．$|\ |$ は絶対値を表します．

（例）行列のノルムを実際に求めてみます． ３# ３の行列 N1 を定義します．

```
> N2 := Matrix( 3,3, [ 1, -2, 3, -4, -5, 6, -7, 8, -9 ] );
```

$$N2 := \begin{bmatrix} 1 & -2 & 3 \\ -4 & -5 & 6 \\ -7 & 8 & -9 \end{bmatrix} \tag{36}$$

行列 N2 のノルムを計算します．

```
> MatrixNorm( N2, 1 );
```

$$18 \tag{37}$$

（参考）式 $\max(|a| + |d| + |g|, |b| + |e| + |h|, |c| + |f| + |i|)$ に実際の値を代入して確認します．

```
> max( abs(  1 ) + abs( -4 ) + abs( -7 ),
       abs( -2 ) + abs( -5 ) + abs(  8 ),
       abs(  3 ) + abs(  6 ) + abs( -9 ) );
```

$$18 \tag{38}$$

行列の操作

初期化します．

```
> restart;
```

はじめに，線形代数パッケージをロードします．

```
> with( LinearAlgebra ):
```

（参考）コロン（:）で，結果を非表示にしています．

行列 M の転置

```
> M := Matrix( 6,6, symbol=m );
```

(39)

$$M := \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} & m_{2,5} & m_{2,6} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} & m_{3,5} & m_{3,6} \\ m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} & m_{4,5} & m_{4,6} \\ m_{5,1} & m_{5,2} & m_{5,3} & m_{5,4} & m_{5,5} & m_{5,6} \\ m_{6,1} & m_{6,2} & m_{6,3} & m_{6,4} & m_{6,5} & m_{6,6} \end{bmatrix} \tag{39}$$

Transposeコマンドを用いた行列の転置

```
> Transpose( M );
```

$$\begin{bmatrix} m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} & m_{5,1} & m_{6,1} \\ m_{1,2} & m_{2,2} & m_{3,2} & m_{4,2} & m_{5,2} & m_{6,2} \\ m_{1,3} & m_{2,3} & m_{3,3} & m_{4,3} & m_{5,3} & m_{6,3} \\ m_{1,4} & m_{2,4} & m_{3,4} & m_{4,4} & m_{5,4} & m_{6,4} \\ m_{1,5} & m_{2,5} & m_{3,5} & m_{4,5} & m_{5,5} & m_{6,5} \\ m_{1,6} & m_{2,6} & m_{3,6} & m_{4,6} & m_{5,6} & m_{6,6} \end{bmatrix} \tag{40}$$

（参考）$A^{\%T}$によっても転置を行うことが出来ます．

```
> M^%T;
```

$$\begin{bmatrix} m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} & m_{5,1} & m_{6,1} \\ m_{1,2} & m_{2,2} & m_{3,2} & m_{4,2} & m_{5,2} & m_{6,2} \\ m_{1,3} & m_{2,3} & m_{3,3} & m_{4,3} & m_{5,3} & m_{6,3} \\ m_{1,4} & m_{2,4} & m_{3,4} & m_{4,4} & m_{5,4} & m_{6,4} \\ m_{1,5} & m_{2,5} & m_{3,5} & m_{4,5} & m_{5,5} & m_{6,5} \\ m_{1,6} & m_{2,6} & m_{3,6} & m_{4,6} & m_{5,6} & m_{6,6} \end{bmatrix} \tag{41}$$

要素の抽出

```
> M[ 1, 1 ];
```

$$m_{1,1} \tag{42}$$

```
> M[ 2, 3 ];
```

$$m_{2,3} \tag{43}$$

1行目から3行目まで，1列目から3列目までを取り出します（部分行列）．

```
> M[ 1..3, 1..3 ];
```

$$\begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix} \tag{44}$$

4行目から5行目まで，3列目から6列目までを取り出します（部分行列）．

```
> M[ 4..5, 3..6 ];
```

$$\begin{bmatrix} m_{4,3} & m_{4,4} & m_{4,5} & m_{4,6} \\ m_{5,3} & m_{5,4} & m_{5,5} & m_{5,6} \end{bmatrix} \tag{45}$$

1行目だけを取り出します.

```
> M[ 1, 1..6 ];
```

$$\begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \end{bmatrix} \tag{46}$$

1列目だけを取り出します.

```
> M[ 1..6, 1];
```

$$\begin{bmatrix} m_{1,1} \\ m_{2,1} \\ m_{3,1} \\ m_{4,1} \\ m_{5,1} \\ m_{6,1} \end{bmatrix} \tag{47}$$

最後の要素を -1 とすることで，要素数が不明な行列も m 行目だけ，あるいは n 列目だけを取り出すことができます．以下は，それぞれ行ベクトル，列ベクトルになります.

（注意）要素指定の違いをわかりやすくするためにスペースを余分に入れています.

```
> M[     1, 1..-1 ];
```

$$\begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} & m_{1,5} & m_{1,6} \end{bmatrix} \tag{48}$$

```
> M[ 1..-1,     1 ];
```

$$\begin{bmatrix} m_{1,1} \\ m_{2,1} \\ m_{3,1} \\ m_{4,1} \\ m_{5,1} \\ m_{6,1} \end{bmatrix} \tag{49}$$

ベクトル

初期化します.

```
> restart;
```

はじめに，線形代数パッケージをロードします.

```
> with( LinearAlgebra ):
```

（参考）コロン（:）で，結果を非表示にしています.

ベクトルの定義

ベクトルはVectorコマンドを用いて定義します.

ベクトルのサイズのみで，要素がベクトルに与えられない場合，すべての値は0（デフォルト値）で埋め尽くされます．特に指定がなければ，列ベクトルが定義されます.

```
> Vector( 2 );
```

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{50}$$

（参考）零ベクトルを明示的に定義する場合は，ZeroVectorコマンドを使用します.

```
> ZeroVector( 3 );
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{51}$$

ベクトルのサイズを指定し，要素をすべて5で定義します.

```
> Vector( 1..3, 5 );
```

$$\begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix} \tag{52}$$

ベクトルの要素を（リストで）指定します.

```
> Vector( [ 1, 2, 3 ] );
```

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{53}$$

行ベクトルを定義します.

```
> Vector[row]( [ 1, 2, 3 ] );
```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \tag{54}$$

記号で要素を指定します.

```
> Vector( 3, symbol=v );
```

(55)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{55}$$

（参考）山括弧< >を用いて，ベクトルを定義することもできます．また，カンマ（, ）は要素を縦方向に並べる働きがあり，バーティカルバー（| ）は要素を横方向に並べる働きがあります．

```
> vc := < 1, 2, 3 >;
```

$$vc := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{56}$$

```
> vr := < 1 | 2 | 3 >;
```

$$vr := \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \tag{57}$$

ランダムにベクトルの要素を定義する RandomVector コマンドがあります．

要素は実行するたびに異なります．

```
> RandomVector( 3 );
```

$$\begin{bmatrix} 92 \\ -31 \\ 67 \end{bmatrix} \tag{58}$$

UnitVector コマンドを用いて，単位ベクトルをシーケンス形式で定義します．シーケンスはカンマ（, ）で区切られた Maple オブジェクトのひとつになります．

```
> UnitVector( 1, 3 ), UnitVector( 2, 3 ), UnitVector( 3, 3 );
```

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{59}$$

ベクトルの操作

列ベクトルを定義します．

```
> v1 := Vector( [ a, b, c ] );
```

$$v1 := \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{60}$$

要素を抽出します．

```
> v1[1]; v1[2]; v1[3];
```

$$\begin{matrix} a \\ b \\ c \end{matrix} \tag{61}$$

行ベクトルを定義します．

```
> v2 := Vector[row]( [ x, y, z ] );
```

$$v2 := \begin{bmatrix} x & y & z \end{bmatrix} \tag{62}$$

要素を抽出します.

```
> v2[1]; v2[2]; v2[3];
```

$$\begin{matrix} x \\ y \\ z \end{matrix} \tag{63}$$

足し算

列ベクトル p を定義します.

```
> p := Vector( 3, symbol=a );
```

$$p := \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{64}$$

列ベクトル q を定義します.

```
> q := Vector( 3, symbol=b );
```

$$q := \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{65}$$

VectorAdd コマンドを用いて２つのベクトルを足し合わせます.

```
> VectorAdd( p, q );
```

$$\begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ a_3 + b_3 \end{bmatrix} \tag{66}$$

ベクトルの角度

直交する２つのベクトルを定義します.

（参考）２つのベクトルがなす角度は $\frac{\pi}{2}$ になります.

```
> v3 := Vector( [ 1, 0 ] );
  v4 := Vector([0,1]);
```

$$v3 := \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$v4 := \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{67}$$

VectorAngle コマンドを用いて，２つのベクトルの角度を計算します.

```
> a := VectorAngle( v3, v4 );
```

$$a := \frac{\pi}{2} \tag{68}$$

内積（スカラー積）と外積（ベクトル積）

初期化します.

```
> restart;
```

ここでは，ベクトルのプロットをするために，学生向けのベクトル解析（VectorCalculus）パッケージをロードします.

```
> with( Student[VectorCalculus] );
```

[&x, `*`, `+`, `-`, `.`, <,>, <|>, *About*, *ArcLength*, *BasisFormat*, *Binormal*, *ConvertVector*, *CrossProduct*, *Curl*, *Curvature*, D, *Del*, *DirectionalDiff*, *Divergence*, *DotProduct*, *FlowLine*, *Flux*, *GetCoordinates*, *GetPVDescription*, *GetRootPoint*, *GetSpace*, *Gradient*, *Hessian*, *IsPositionVector*, *IsRootedVector*, *IsVectorField*, *Jacobian*, *Laplacian*, *LineInt*, *MapToBasis*, ∇, *Norm*, *Normalize*, *PathInt*, *PlotPositionVector*, *PlotVector*, *PositionVector*, *PrincipalNormal*, *RadiusOfCurvature*, *RootedVector*, *ScalarPotential*, *SetCoordinates*, *SpaceCurve*, *SpaceCurveTutor*, *SurfaceInt*, *TNBFrame*, *Tangent*, *TangentLine*, *TangentPlane*, *TangentVector*, *Torsion*, *Vector*, *VectorField*, *VectorFieldTutor*, *VectorPotential*, *VectorSpace*, *diff*, *evalVF*, *int*, *limit*, *series*] (69)

２つの（３次元）ベクトルv1，v2を定義します.

（参考）e_x, e_y, e_z は，それぞれ３次元の直交座標系（デカルト座標系）の単位ベクトルを表します.

$$e_x, e_y, e_z = \left[\begin{matrix} 1 \\ 0 \\ 0 \end{matrix}\right], \left[\begin{matrix} 0 \\ 1 \\ 0 \end{matrix}\right], \left[\begin{matrix} 0 \\ 0 \\ 1 \end{matrix}\right]$$

```
> v1 := Vector( [1, 2, 1] );
```

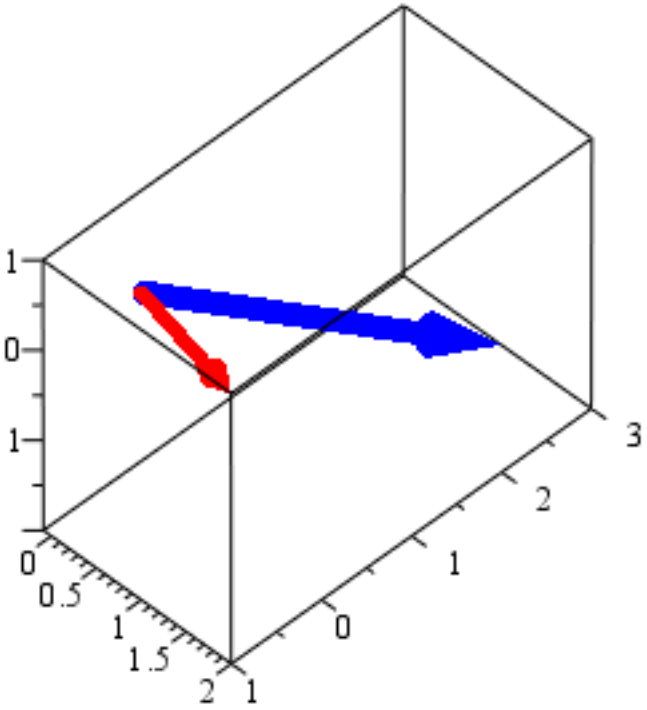
$$v1 := e_x + 2e_y + e_z \quad (70)$$

```
> v2 := Vector( [ -3, 1, -2 ] );
```

$$v2 := -3e_x + e_y - 2e_z \quad (71)$$

ベクトルをプロットします.

```
> PlotVector( [ v1, v2 ], color=[ red,  blue ], axes=boxed, scaling=
  constrained );
```



ベクトルv1とv2の内積（スカラー積）をDotProductコマンドを用いて計算します.

```
> dpv := DotProduct( v1, v2 );
```

$$dpv := -3 \tag{72}$$

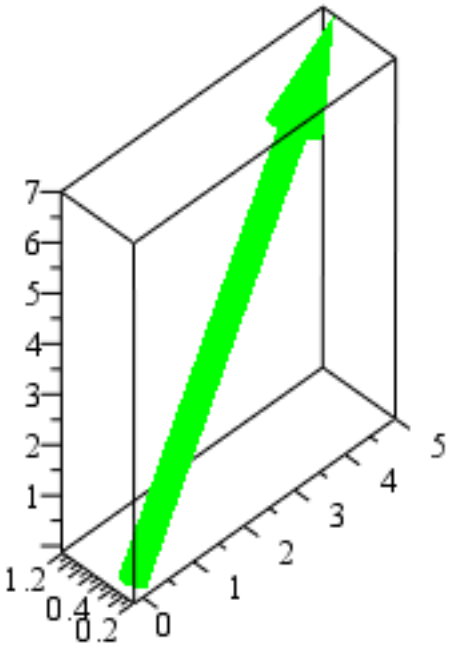
ベクトルv1とv2の外積（ベクトル積）をCrossProductコマンドを用いて計算します.

```
> cpv := CrossProduct( v1, v2 );
```

$$cpv := -5e_x - e_y + 7e_z \tag{73}$$

cpvをプロットしてみます.

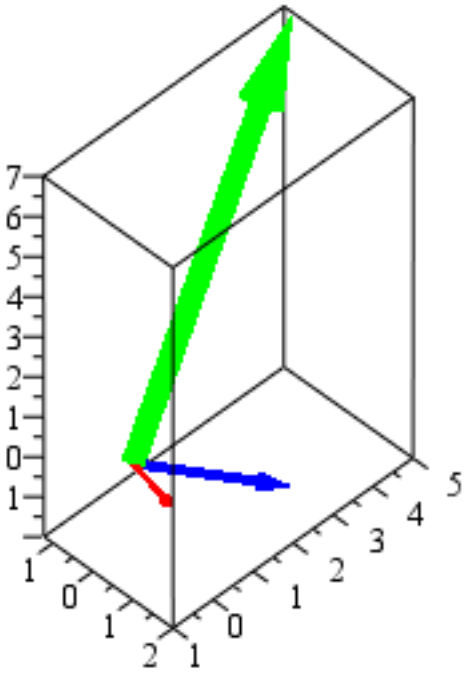
```
> PlotVector( cpv, color=green, axes=boxed, scaling=constrained );
```



ベクトルv1, v2, cpvを同時にプロットします.

（参考）外積は２つのベクトルから新たなベクトルを定義します.

```
> PlotVector([v1, v2, cpv], color=[red, blue, green], axes=boxed, scaling=
  constrained );
```



ノルム（ベクトルの大きさ）

初期化します.

```
> restart:
```

線形代数（LinearAlgebra）パッケージをロードします.

```
> with( LinearAlgebra ):
```

（参考）コロン（：）で，結果を非表示にしています.

ベクトルを定義します.

```
> v := Vector( [ a, b ] );
```

$$v := \begin{bmatrix} a \\ b \end{bmatrix} \tag{74}$$

ベクトル v の 1-ノルム

```
> Norm( v, 1 );
```

$$|a| + |b| \tag{75}$$

ベクトル v の 2-ノルム

```
> Norm( v, 2 );
```

$$\sqrt{|a|^2 + |b|^2} \tag{76}$$

ベクトル v の 3-ノルム

```
> Norm( v, 3 );
```

$$\left(|a|^3+|b|^3\right)^{1/3} \tag{77}$$

ベクトル v の 3/2-ノルム

```
> Norm( v, 3/2 );
```

$$\left(|a|^{3/2}+|b|^{3/2}\right)^{2/3} \tag{78}$$

ベクトル v の 無限大-ノルム

```
> Norm( v, infinity );
```

$$\max(|a|,|b|) \tag{79}$$

固有値と固有ベクトル

初期化します.

```
> restart;
```

線形代数（LinearAlgebra）パッケージをロードします.

```
> with( LinearAlgebra ):
```

（参考）コロン（: ）で，結果を非表示にしています.

固有値の計算

2# 2の行列Aを定義し，その固有値を計算します.

```
> A := Matrix( 2,2, [ 2, 0, 7, -5 ] );
```

$$A := \begin{bmatrix} 2 & 0 \\ 7 & -5 \end{bmatrix} \tag{80}$$

行列Aの固有値をEigenvaluesコマンドを用いて求めます.

```
> r := Eigenvalues( A );
```

$$r := \begin{bmatrix} 2 \\ -5 \end{bmatrix} \tag{81}$$

（参考）以下は，教科書にあるような手順で固有値を求めています.

はじめ，2# 2の単位行列を定義します.

```
> E := IdentityMatrix( 2 );
```

$$E := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{82}$$

DiagonalMatrixコマンドを用いた定義

```
> DiagonalMatrix( [ 1, 1 ] );
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{83}$$

Aを正方行列，l を固有値，x を固有ベクトルとした場合，$Ax = \lambda x$ を変形して得られる以下の式（***行列式*** $= 0$）より，固有方程式（特性方程式）を導きます.

$|A - \lambda E| = 0$

```
> M := A - lambda*E;
```

$$M := \begin{bmatrix} 2-\lambda & 0 \\ 7 & -5-\lambda \end{bmatrix} \tag{84}$$

行列式を求めるDeterminantコマンドを使用します.

```
> ceq := Determinant( M ) = 0;
```

$$ceq := (-2+\lambda)(5+\lambda) = 0 \tag{85}$$

固有方程式（特性方程式）$ceq := (-2+\lambda)(5+\lambda) = 0$ の解を求めます.

```
> R := solve( ceq, lambda );
```

$$R := 2, -5 \tag{86}$$

$r := \begin{bmatrix} 2 \\ -5 \end{bmatrix}$ と同じように固有値が導かれます．

（参考）固有値方程式と特性方程式は同じものです．特性方程式として解を求めた場合，導かれた根を特性根と呼びます．

固有ベクトルの計算

はじめに定義した 2# 2の行列Aについて，その固有値と固有ベクトルを計算します．

Eigenvaluesコマンドは，固有値と固有ベクトルを同時に計算します．そこで，２つの変数e，v を用意して，２つの固有値をそれぞれ別々の変数e1，e2に割り当てます．

```
> ( e, v ) := Eigenvectors( A );
```

$$e, v := \begin{bmatrix} 2 \\ -5 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{87}$$

```
> e1 := e[1];
  e2 := e[2];
```

$$e1 := 2$$
$$e2 := -5 \tag{88}$$

２つの固有ベクトルをそれぞれ別々の変数v1，v2に割り当てます．

```
> v1 := v[ 1..-1, 1 ]; # １列目を抽出
  v2 := v[ 1..-1, 2 ]; # ２列目を抽出
```

$$v1 := \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
$$v2 := \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{89}$$

A を正方行列，l を固有値，x を固有ベクトルとした場合，$Ax = \lambda x$ の関係式が成り立ちますので，求められた固有値と固有ベクトルをそれぞれ関係式に代入してみます．

$A\,v1 = e1\,v1$ より，以下のようになります．

（参考）ドット（．）は内積を表します．

```
> A.v1 = e1.v1;
```

$$\begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \tag{90}$$

同様に，$A\,v2 = e2\,v2$ を調べます．

```
> A.v2 = e2.v2;
```

$$\begin{bmatrix} 0 \\ -5 \end{bmatrix} = \begin{bmatrix} 0 \\ -5 \end{bmatrix} \tag{91}$$

1次変換（線形写像）のプロット

初期化します.

```
> restart;
```

1次変換（線形写像）のプロットコマンドを使用するために，学生向け線形代数（LinearAlgebra）パッケージをロードします.

```
> with( Student[LinearAlgebra] ):
```

（参考）コロン（:）で，結果を非表示にしています.

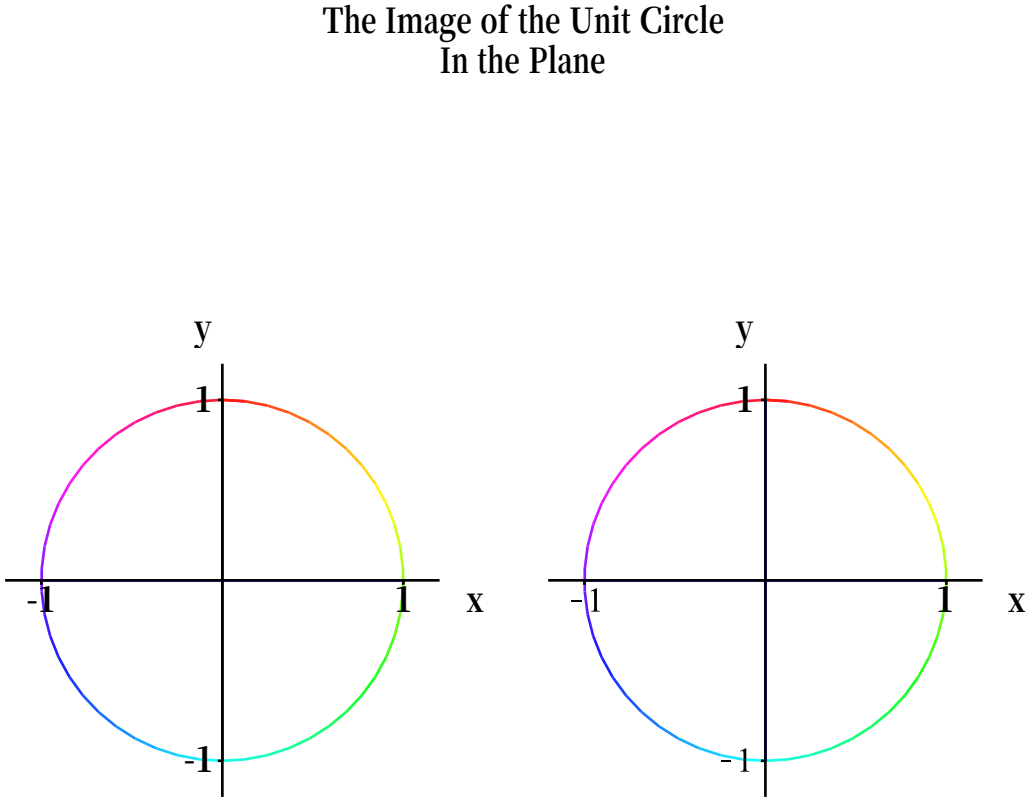
表現行列を単位行列として定義します.

```
> E := IdentityMatrix( 2 );
```

$$E := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{92}$$

1次変換をLinearTransformPlotコマンドを用いてプロットします．左側の単位円（半径1の円）が変換されます．単位行列の場合，自分自身に変換されます（恒等変換）.

```
> LinearTransformPlot( E );
```



適当な表現行列を定義します.

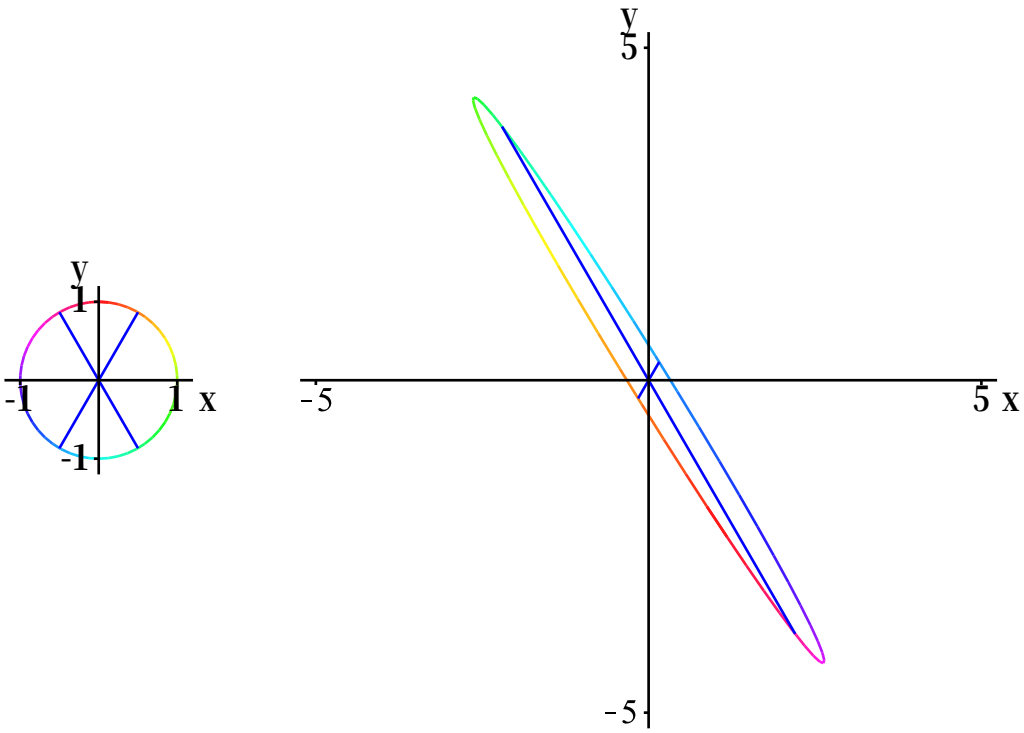
```
> M := Matrix( 2,2, [ -2, 1, 3, -2 ] );
```

(93)

$$M := \begin{bmatrix} -2 & 1 \\ 3 & -2 \end{bmatrix} \tag{93}$$

1次変換を実行します.

```
> LinearTransformPlot( M );
```



表現行列を3次元に拡大します.

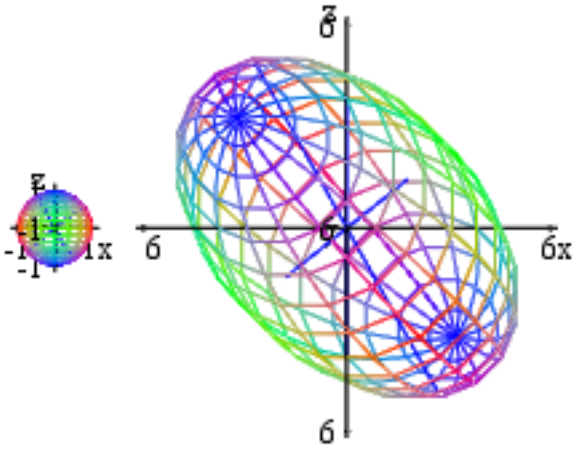
```
> A := Matrix( 3,3, [ -2,  3,  3,
                      2, -3,  2,
                      3,  2, -3 ] );
```

$$A := \begin{bmatrix} -2 & 3 & 3 \\ 2 & -3 & 2 \\ 3 & 2 & -3 \end{bmatrix} \tag{94}$$

1次変換を実行します.

```
> LinearTransformPlot( A );
```

The Image of the Unit Sphere In 3-Space



連立１次方程式の解法

初期化します.

```
> restart;
```

線形代数（LinearAlgebra）パッケージをロードします.

```
> with( LinearAlgebra ):
```

（参考）コロン（: ）で，結果を非表示にしています.

はじめに連立１次方程式を（係数を文字として）定義します.

```
> eq := [ a11*x + a12*y + a13*z = p,
          a21*x + a22*y + a23*z = q,
          a31*x + a32*y + a33*z = r ];
```

$$eq := [a11\,x + a12\,y + a13\,z = p,\ a21\,x + a22\,y + a23\,z = q,\ a31\,x + a32\,y + a33\,z = r] \tag{95}$$

各１次方程式を表示します.

```
> eq[1];
  eq[2];
  eq[3];
```

$$a11\,x + a12\,y + a13\,z = p$$
$$a21\,x + a22\,y + a23\,z = q$$
$$a31\,x + a32\,y + a33\,z = r \tag{96}$$

係数を定義します.

```
> cffs := [ a11 = 1, a12 = 1, a13 = -1, p =  5,
            a21 = 2, a22 = 2, a23 =  0, q = -1,
            a31 = 0, a32 = 1, a33 =  3, r =  2 ];
```

$$cffs := [a11 = 1,\ a12 = 1,\ a13 = -1,\ p = 5,\ a21 = 2,\ a22 = 2,\ a23 = 0,\ q = -1,\ a31 = 0,\ a32 = 1,\ a33 = 3,\ r = 2] \tag{97}$$

係数を，式(95)に代入します.

```
> eqs := eval( eq, cffs );
```

$$eqs := [x + y - z = 5,\ 2\,x + 2\,y = -1,\ y + 3\,z = 2] \tag{98}$$

従属変数を定義します.

```
> v := [ x, y, z ];
```

$$v := [x, y, z] \tag{99}$$

solveコマンドを使用して連立1次方程式を解きます.

```
> sol := solve( eqs, v );
```

$$sol := \left[\left[x = -19,\ y = \frac{37}{2},\ z = -\frac{11}{2}\right]\right] \tag{100}$$

以下は行列操作によって連立方程式を解きます.

係数行列を抽出します.

```
> ( A, b ) := GenerateMatrix( eqs, v );
```

(101)

$$A, b := \begin{bmatrix} 1 & 1 & -1 \\ 2 & 2 & 0 \\ 0 & 1 & 3 \end{bmatrix}, \begin{bmatrix} 5 \\ -1 \\ 2 \end{bmatrix} \tag{101}$$

抽出された係数行列を表示します.

```
> A; b;
```

$$\begin{bmatrix} 1 & 1 & -1 \\ 2 & 2 & 0 \\ 0 & 1 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 5 \\ -1 \\ 2 \end{bmatrix} \tag{102}$$

連立1次方程式の係数行列を引数として解を求めるLinearSolveコマンドを使用します.

```
> sol1 := LinearSolve( A, b );
```

$$sol1 := \begin{bmatrix} -19 \\ \frac{37}{2} \\ -\frac{11}{2} \end{bmatrix} \tag{103}$$

$A.v=b$ の式から$v=A^{-1}.b$を計算して解を求めます. MatrixInverseコマンドは逆行列を計算します.

```
> Sol2 := MatrixInverse( A ).b;
```

$$Sol2 := \begin{bmatrix} -19 \\ \frac{37}{2} \\ -\frac{11}{2} \end{bmatrix} \tag{104}$$

以下は, ガウス・ジョルダン消去法を用いた解法になります.

係数拡大行列Mを作成します. 同じコマンドにオプションを追加します.

```
> M1 := GenerateMatrix( eqs, v, augmented=true );
```

$$M1 := \begin{bmatrix} 1 & 1 & -1 & 5 \\ 2 & 2 & 0 & -1 \\ 0 & 1 & 3 & 2 \end{bmatrix} \tag{105}$$

(参考) 行列(105)は, 以下の方法でも得ることができます.

$$A, b := \begin{bmatrix} 1 & 1 & -1 \\ 2 & 2 & 0 \\ 0 & 1 & 3 \end{bmatrix}, \begin{bmatrix} 5 \\ -1 \\ 2 \end{bmatrix}$$

```
> < A | b >;
```

$$\begin{bmatrix} 1 & 1 & -1 & 5 \\ 2 & 2 & 0 & -1 \\ 0 & 1 & 3 & 2 \end{bmatrix} \tag{106}$$

ガウス・ジョルダンの消去法を適用します.

```
> M2 := ReducedRowEchelonForm( M1 );
```

$$M2 := \begin{bmatrix} 1 & 0 & 0 & -19 \\ 0 & 1 & 0 & \frac{37}{2} \\ 0 & 0 & 1 & -\frac{11}{2} \end{bmatrix} \tag{107}$$

連立１次方程式を視覚化するコマンドを使用するために，学生向けの線形代数（LinearAlgebra）パッケージのコマンドを使用してみます.

```
> with( Student[LinearAlgebra] ):
```

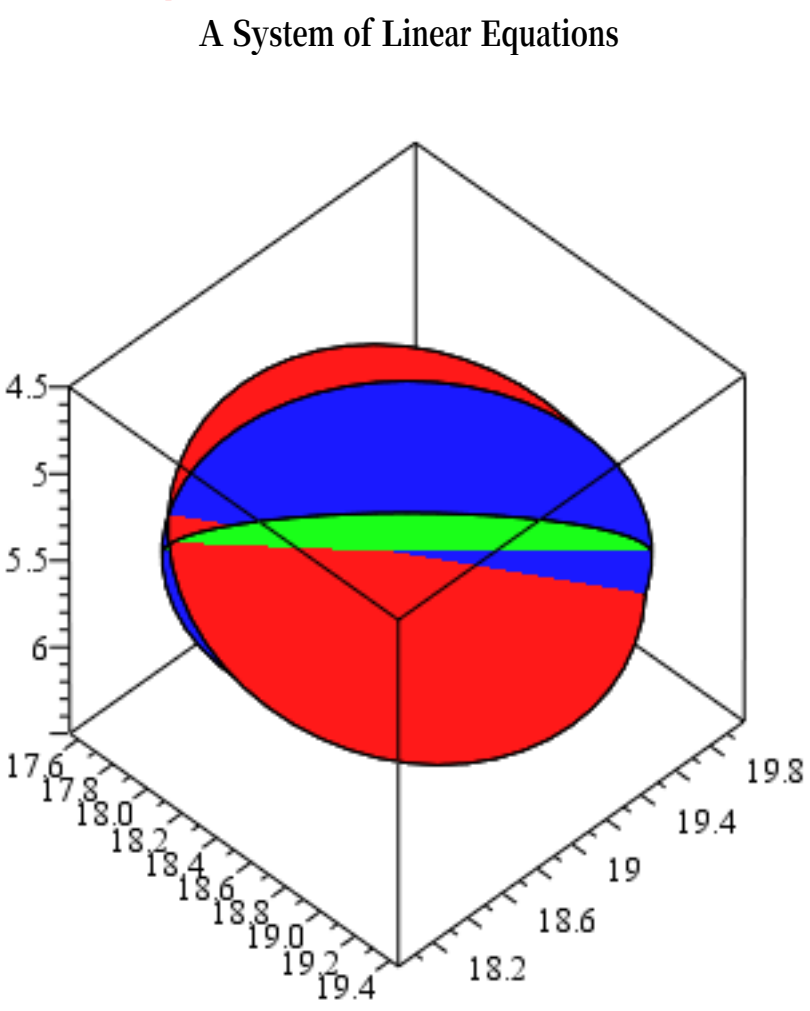
連立１次方程式eqsの内容を確認します.

```
> eqs;
```

$$[x + y - z = 5, 2x + 2y = -1, y + 3z = 2] \tag{108}$$

LinearSystemPlotコマンドを使用します．各方程式は，それぞれ平面の式を表しています.

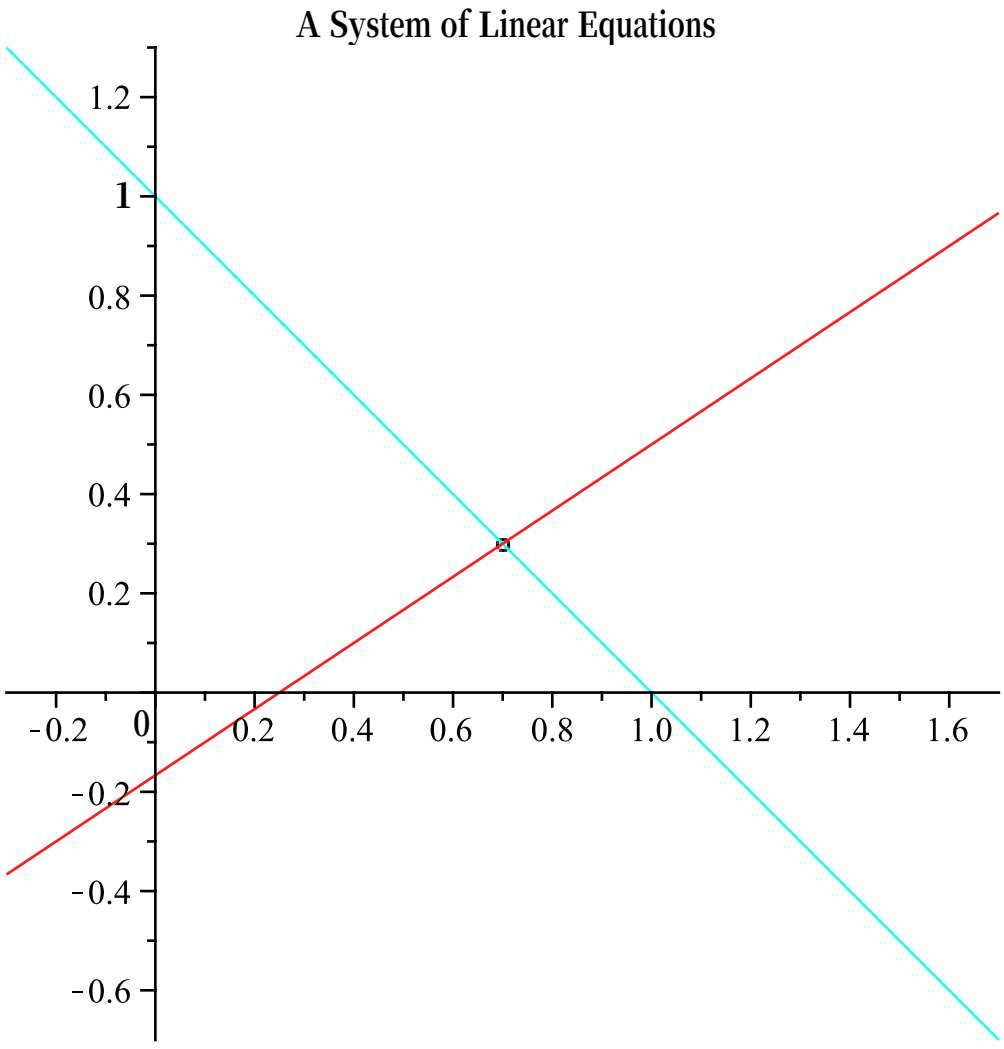
```
> LinearSystemPlot( eqs );
```



（参考）以下の連立方程式を LinearSystemPlot コマンドで解きます.

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

```
> LinearSystemPlot( [ x + y = 1,
                      2*x - 3*y = 1/2 ] );
```



ステップ･バイ･ステップ式 **はじめての Maple 応用編**

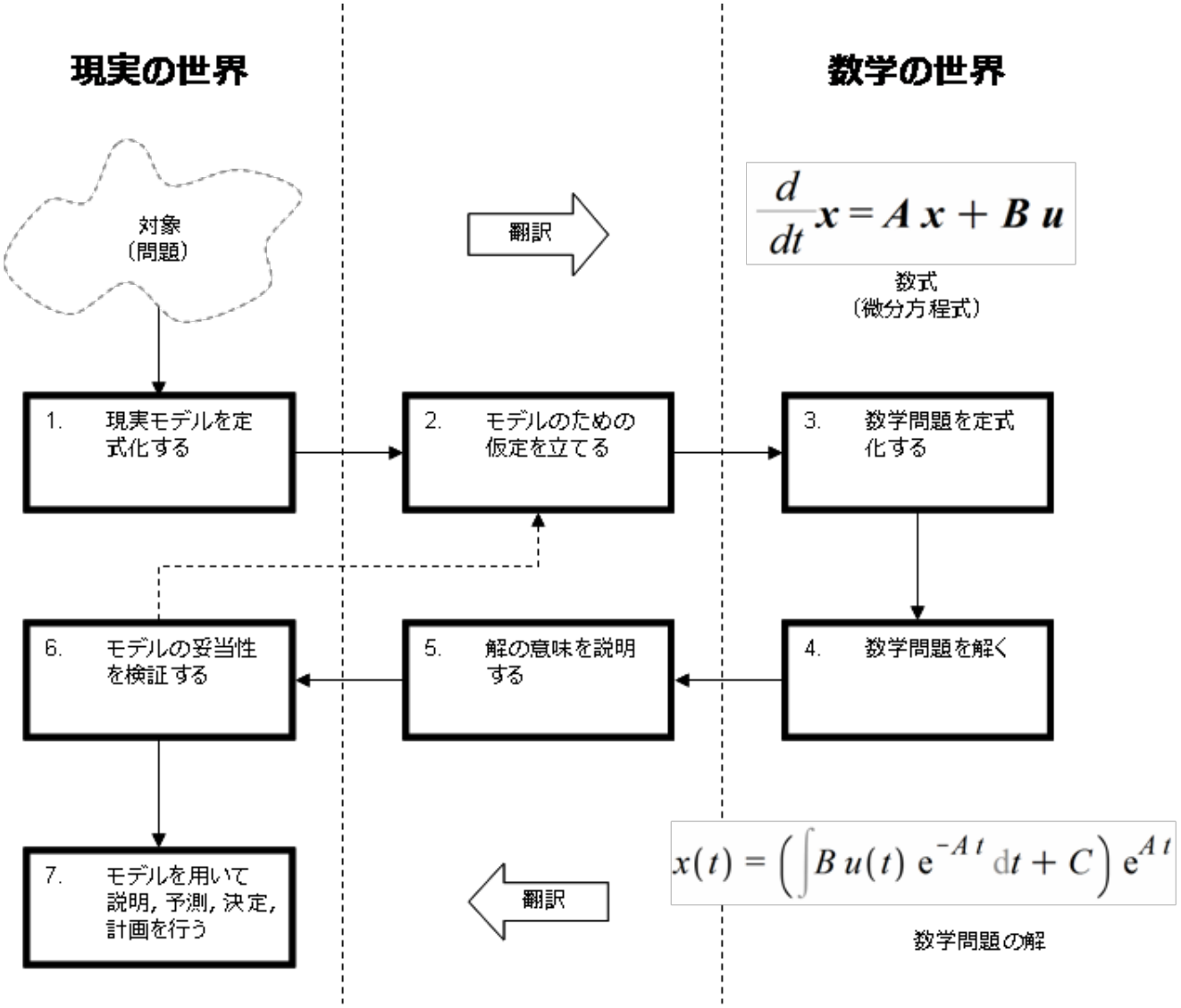
微分方程式に挑む！

微分方程式の定義方法およびその解法を習得します.

目次

数学モデルの作り方

1. 現実モデルを定式化する
2. モデルのための仮定を立てる
3. 数学問題を定式化する
4. 数学問題を解く
5. 解の意味を説明する
6. モデルの妥当性を検証する（必要に応じて２に戻る）
7. モデルを用いて説明，予測，決定，計画を行う



図．現実社会の問題をモデル化するおもな過程

参考図書：『微分方程式で数学モデルを作ろう』，日本評論社，D・バージェス／M・ポリー著，垣田高夫／大町比佐栄訳

（注意）一部情報を加筆・追加しました．

常微分方程式（ODE）の定義と解析解

少しだけ常微分方程式のこと

関数の微分である導関数を含む方程式を微分方程式と呼び，特に独立変数がひとつの微分方程式を常微分方程式と呼びます．例えば，$y(x)$ や $x(t)$ などの関数が微分され，それらで式が組み立てられます．

$$\frac{dy}{dx}, \frac{d^2y}{dx^2}, \frac{dx}{dt}, \frac{d^2x}{dt^2}, \ldots\ldots$$

（参考） 常微分方程式の英語は Ordinary Differential Equation になります．通常，頭文字から ODE と呼ばれます．

１階常微分方程式

方程式に含まれる導関数の中でもっとも高い次数が１の場合（階数が１階である場合），１階常微分方程式と呼びます．通常，テキストなどでは，以下のように１階常微分方程式が記述されています．

$$\frac{dy}{dx} = (1 - y^2)\tan x$$

あるいは，

$$y' = (1 - y^2)\tan x$$

ただし，もう少し記述を正確にすると，以下のようになります．

$$y'(x) = (1 - y(x)^2)\tan(x)$$

つまり，y は x の関数になります．**本テキストでは，最後の記述スタイル** $y'(x) = (1 - y(x)^2)\tan(x)$ **を用いています．**

ここで，初期化します．

```
> restart;
```

（例１）　$\frac{d}{dx}y(x) = (1 - y(x)^2)\tan(x)$

１階の常微分方程式 $\frac{d}{dx}y(x) = (1 - y(x)^2)\tan(x)$ を定義します．

```
> ode1 := diff( y(x), x ) = ( 1 - y(x)^2 )*tan(x);
```

$$ode1 := y'(x) = (1 - y(x)^2)\tan(x) \quad (1)$$

微分方程式の求解コマンド dsolve を用いて，定義した微分方程式の一般解を求めます．_C1 は，Maple が内部処理で用いる任意定数になります．

```
> sol1 := dsolve( ode1 );
```

$$sol1 := y(x) = \tanh(-\ln(\cos(x)) + \_C1) \quad (2)$$

ここで，任意定数 _C1 をパラメータ p に置き換えます．

```
> sol1a := subs( _C1 = p, sol1 );
```

$$sol1a := y(x) = \tanh(-\ln(\cos(x)) + p) \quad (3)$$

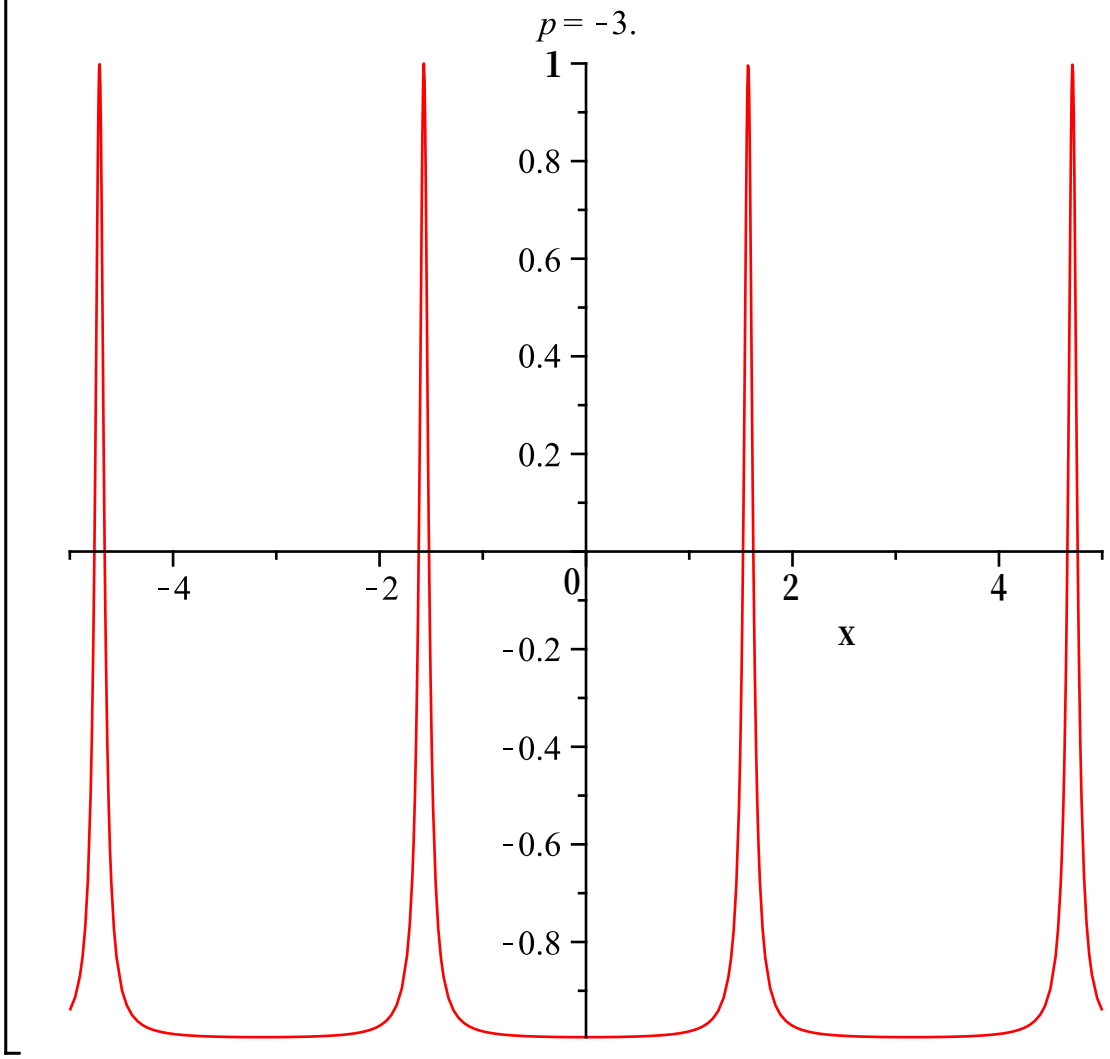
式 $sol1a := y(x) = \tanh(-\ln(\cos(x)) + p)$ の右辺を eq1 に割り当てます．

```
> eq1 := rhs( sol1a );
```

$$eq1 := \tanh(-\ln(\cos(x)) + p) \quad (4)$$

plotsパッケージのanimateコマンドを用いて，パラメータpの変化を見てみます．

```
> plots[animate]( plot, [ eq1, x=-5..5 ], p=-3..3 );
```



（参考）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．

（例2） $x\left(\frac{d}{dx}y(x)\right)=y(x)+\sqrt{x^2+y(x)^2}$

1階の常微分方程式 $x\left(\frac{d}{dx}y(x)\right)=y(x)+\sqrt{x^2+y(x)^2}$ を定義します．

```
> ode2 := x*diff( y(x), x ) = y(x) + sqrt( x^2 + y(x)^2 );
```

$$ode2 := x\,y'(x)=y(x)+\sqrt{x^2+y(x)^2} \tag{5}$$

ode2の一般解を求めます．

```
> sol2 := dsolve( ode2 );
```

$$sol2 := \frac{y(x)}{x^2}+\frac{\sqrt{x^2+y(x)^2}}{x^2}-\_C1=0 \tag{6}$$

ここで，isolateコマンドを用いて，y(x)で式を整理します．

```
> sol2a := isolate( sol2, y(x) );
```

$$sol2a := y(x) = \frac{-1 + \_C1^2 x^2}{2\,\_C1} \tag{7}$$

任意定数 _C1 を q で置き換えます.

```
> sol2b := eval( sol2a, [ _C1 = q ] );
```

$$sol2b := y(x) = \frac{-1 + q^2 x^2}{2\,q} \tag{8}$$

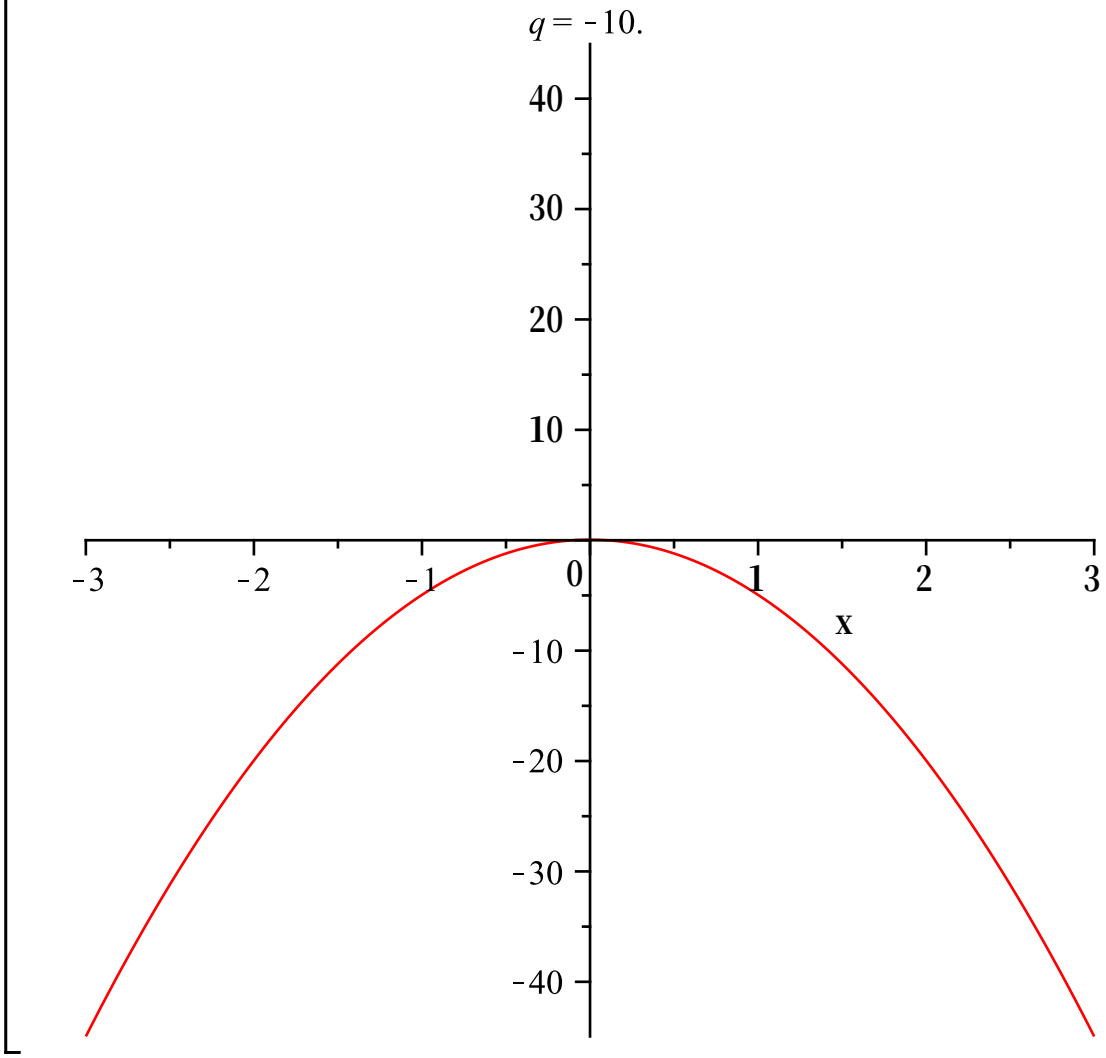
式 $sol2b := y(x) = \frac{-1 + q^2 x^2}{2\,q}$ の右辺を eq2 に割り当てます.

```
> eq2 := rhs( sol2b );
```

$$eq2 := \frac{-1 + q^2 x^2}{2\,q} \tag{9}$$

plots パッケージの animate コマンドを用いて，パラメータ q の変化を見てみます.

```
> plots[animate]( plot, [ eq2, x=-3..3], q=-10..10 );
```



（参考）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.

（例３）

$$\frac{d}{dx}y(x)+2y(x)\tan(x)=\sin(x)$$

1階の常微分方程式 $\frac{d}{dx}y(x)+2y(x)\tan(x)=\sin(x)$ を定義します.

```
> ode3 := diff( y(x), x ) + 2*y(x)*tan(x) = sin(x);
```

$$ode3 := y'(x)+2y(x)\tan(x)=\sin(x) \tag{10}$$

ode3の一般解を求めます.

```
> sol3 := dsolve( ode3 );
```

$$sol3 := y(x)=\frac{2\cos(2x)+2+3\_C1\cos(x)+\_C1\cos(3x)}{4\cos(x)} \tag{11}$$

一般解sol3を展開します.

```
> sol3a := expand( sol3 );
```

$$sol3a := y(x)=\cos(x)+\cos(x)^2\_C1 \tag{12}$$

任意定数_C1をr で置き換えます.

```
> sol3b := subs( _C1=r, sol3a );
```

$$sol3b := y(x)=\cos(x)+\cos(x)^2 r \tag{13}$$

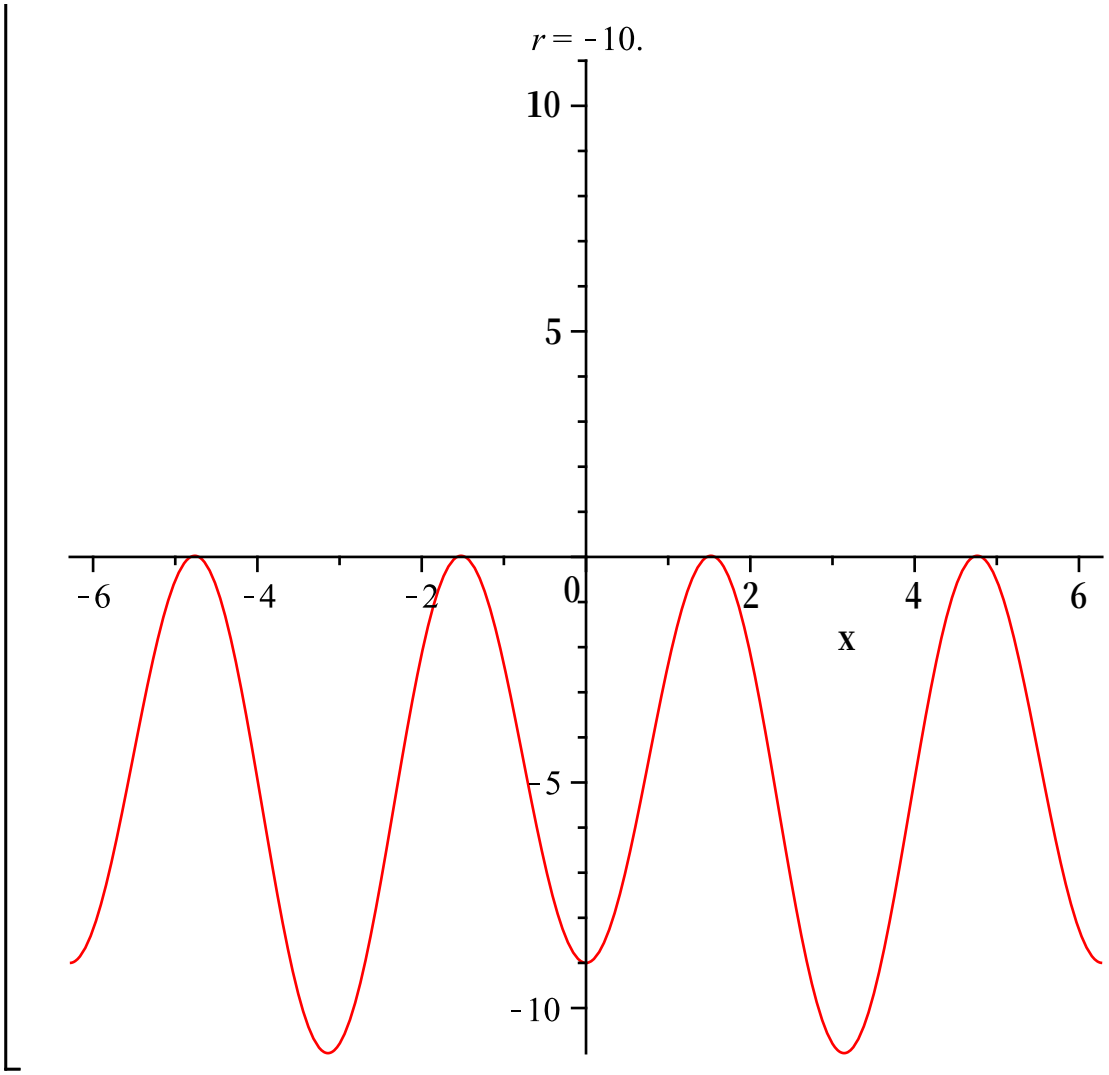
式 $sol3b := y(x)=\cos(x)+\cos(x)^2 r$ の右辺をeqに割り当てます.

```
> eq3 := rhs( sol3b );
```

$$eq3 := \cos(x)+\cos(x)^2 r \tag{14}$$

plotsパッケージのanimateコマンドを用いて，パラメータr の変化を見てみます.

```
> plots[animate]( plot, [ eq3, x=-2*Pi..2*Pi ], r=-10..10 );
```



（参考）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．

２階線形常微分方程式の解

方程式に含まれる導関数の中でもっとも高い次数が２の場合（階数が２階である場合），２階常微分方程式と呼びます．ここでは，線形な２階常微分方程式を対象とします．

・解を理解するためのポイント①

解の形が指数関数を用いて

$$x(t) = C \cdot e^{\lambda t}$$

と仮定されます．ただし，1，Cは定数になります．

・解を理解するためのポイント②

重ね合わせの原理によって，解は以下のように記述できます（ただし，C_1 と C_2 は任意定数）．

$$x(t) = C_1 \cdot e^{\lambda_1 t} + C_2 \cdot e^{\lambda_2 t}$$

・解を理解するためのポイント③

λ_1 と λ_2 の組は以下の３つの型に分けられます．

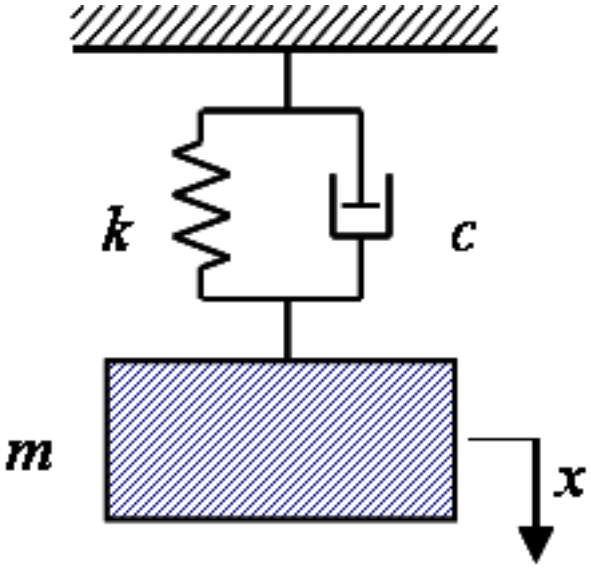
１）異なる２つの実数になる場合

２）異なる２つの複素数になる場合

３）同じ二つの実数になる場合

１自由度粘性減衰振動系の解

以上のポイントを踏まえて，以下の減衰する振動（１自由度粘性減衰振動系）を考えます．



初期化します．

```
> restart:
```

ニュートンの第２法則 $m\cdot\frac{d^2}{dt^2}x(t)=F$ より，力は加速度に比例し，加速度は時間に関する位置の２階微分で表されることがわかります．

以上の知見をもとに，１自由度の振動系を運動方程式で記述します．すなわち，ここでは，2階の常微分方程式を定義することになります．

（復習）方程式に含まれる導関数の中でもっとも高い次数が２の場合（階数が２階である場合），２階（線形）常微分方程式と呼びます．

```
> deq1 := m*diff( x(t), t, t ) + c*diff( x(t), t ) + k*x(t) = 0;
```

$$deq1 := m\ddot{x}(t)+c\dot{x}(t)+kx(t)=0 \tag{15}$$

解を理解するためのポイント①より

方程式の一般解を以下のような指数関数型で仮定します（Cは任意定数）．

```
> eq1 := x(t) = C*exp( lambda*t );
```

$$eq1 := x(t)=C\mathrm{e}^{\lambda t} \tag{16}$$

続けて，式

$$deq1 := m\ddot{x}(t)+c\dot{x}(t)+kx(t)=0$$

に

$$eq1 := x(t)=C\mathrm{e}^{\lambda t}$$

を代入します．

```
> eq2 := eval( deq1, [ eq1 ] );
```

$$eq2 := mC\lambda^2\mathrm{e}^{\lambda t}+cC\lambda\mathrm{e}^{\lambda t}+kC\mathrm{e}^{\lambda t}=0 \tag{17}$$

次に，式

$$eq2 := mC\lambda^2 e^{\lambda t} + cC\lambda e^{\lambda t} + kC e^{\lambda t} = 0$$

の両辺を

$$C e^{\lambda t}$$

で割ります．

```
> eq2a := eq2/( C*exp( lambda*t ) );
```

$$eq2a := \frac{mC\lambda^2 e^{\lambda t} + cC\lambda e^{\lambda t} + kC e^{\lambda t}}{C e^{\lambda t}} = 0 \quad (18)$$

最後に，式

$$eq2a := \frac{mC\lambda^2 e^{\lambda t} + cC\lambda e^{\lambda t} + kC e^{\lambda t}}{C e^{\lambda t}} = 0$$

を展開します．

```
> eq2b := expand( eq2a );
```

$$eq2b := m\lambda^2 + c\lambda + k = 0 \quad (19)$$

この式

$$eq2b := m\lambda^2 + c\lambda + k = 0$$

は特性方程式と呼ばれ，その根は特性根と呼ばれます．

ここで，特性方程式

$$eq2b := m\lambda^2 + c\lambda + k = 0$$

をλについて解きます．また，２つの特性根をλ_1, λ_2に，そのまま割り当てます．

```
> ( lambda[1], lambda[2] ) := solve( eq2b, lambda );
```

$$\lambda_1, \lambda_2 := \frac{-c + \sqrt{c^2 - 4mk}}{2m}, -\frac{c + \sqrt{c^2 - 4mk}}{2m} \quad (20)$$

特性根λ_1, λ_2の性質は，$c^2 - 4mk$の符号によって変わります．すなわち；

1) $c^2 - 4mk > 0$のとき，異なる２つの実根を持ちます．

2) $c^2 - 4mk < 0$のとき，異なる２つの虚根を持ちます．

3) $c^2 - 4mk = 0$のとき，重根（実数）を持ちます．

（参考）これは，高校数学で習う判別式を表しています．判別式とは，２次方程式が異なる２つの実根を持つかどうかを判別するための式になります．

以上から，**解を理解するためのポイント③**

λ_1とλ_2の組は以下の３つの型に分けられます．

１）異なる２つの実数になる場合
２）異なる２つの複素数になる場合
３）同じ二つの実数になる場合

は，以下のような表現に改めることができます．

特性方程式 $m\lambda^2+c\lambda+k=0$ は，特性根 λ_1 と λ_2 の性質によって３つの型に分けられます．

１）異なる２つの実根を持つ場合（$c^2-4mk>0$）
２）異なる２つの虚根を持つ場合（$c^2-4mk<0$），そして
３）重根（実数）を持つ場合（$c^2-4mk=0$）

ここで，解を理解するためのポイント②

重ね合わせの原理から，式

$$eq2b := m\lambda^2+c\lambda+k=0$$

の一般解は，はじめに仮定した

$$eq1 := x(t)=C\mathrm{e}^{\lambda t}$$

より，以下のように求められます．

$$x(t)=C1\,\mathrm{e}^{\lambda_1 t}+C2\,\mathrm{e}^{\lambda_2 t}$$

ただし，$C1$，$C2$ は任意定数です．

以上を sol1 として定義します．

```
> sol1 := C1*exp( lambda[1] * t ) + C2*exp( lambda[2] * t );
```

$$sol1 := C1\,\mathrm{e}^{\frac{\left(-c+\sqrt{c^2-4mk}\right)t}{2m}}+C2\,\mathrm{e}^{-\frac{\left(c+\sqrt{c^2-4mk}\right)t}{2m}} \tag{21}$$

同様に，dsolve コマンドを用いて，はじめの２階線形常微分方程式

$$deq1 := m\ddot{x}(t)+c\dot{x}(t)+kx(t)=0$$

の解を求めてみます．

```
> dsolve( deq1 );
```

$$x(t)=\_C1\,\mathrm{e}^{\frac{\left(-c+\sqrt{c^2-4mk}\right)t}{2m}}+\_C2\,\mathrm{e}^{-\frac{\left(c+\sqrt{c^2-4mk}\right)t}{2m}} \tag{22}$$

ひとつずつ式を変形しながら求めた解

$$sol1 := C1\,\mathrm{e}^{\frac{\left(-c+\sqrt{c^2-4mk}\right)t}{2m}}+C2\,\mathrm{e}^{-\frac{\left(c+\sqrt{c^2-4mk}\right)t}{2m}}$$

と同じになります．

この一般解は，特性方程式の３つの型（特性根が実根，虚根，あるいは重根）によって，それぞれ異なる挙動を示します．

１）異なる２つの実根を持つ場合（$c^2-4mk>0$）
２）異なる２つの虚根を持つ場合（$c^2-4mk<0$）
３）重根（実数）を持つ場合（$c^2-4mk=0$）

指数関数の特性

指数関数に具体的な値を代入して，その挙動を調べます．

はじめに，初期化します．

```
> restart:
```

基本となる指数関数を f として，以下のように定義します．

```
> f := exp( lambda1*t ) + exp( lambda2*t );
```

$$f := e^{\lambda 1\, t} + e^{\lambda 2\, t} \tag{23}$$

1） 異なる２つの実根を持つ場合

a) ともに正の場合

特性根を定義します．

```
> pa := [ lambda1 = 1, lambda2 = 2 ];
```

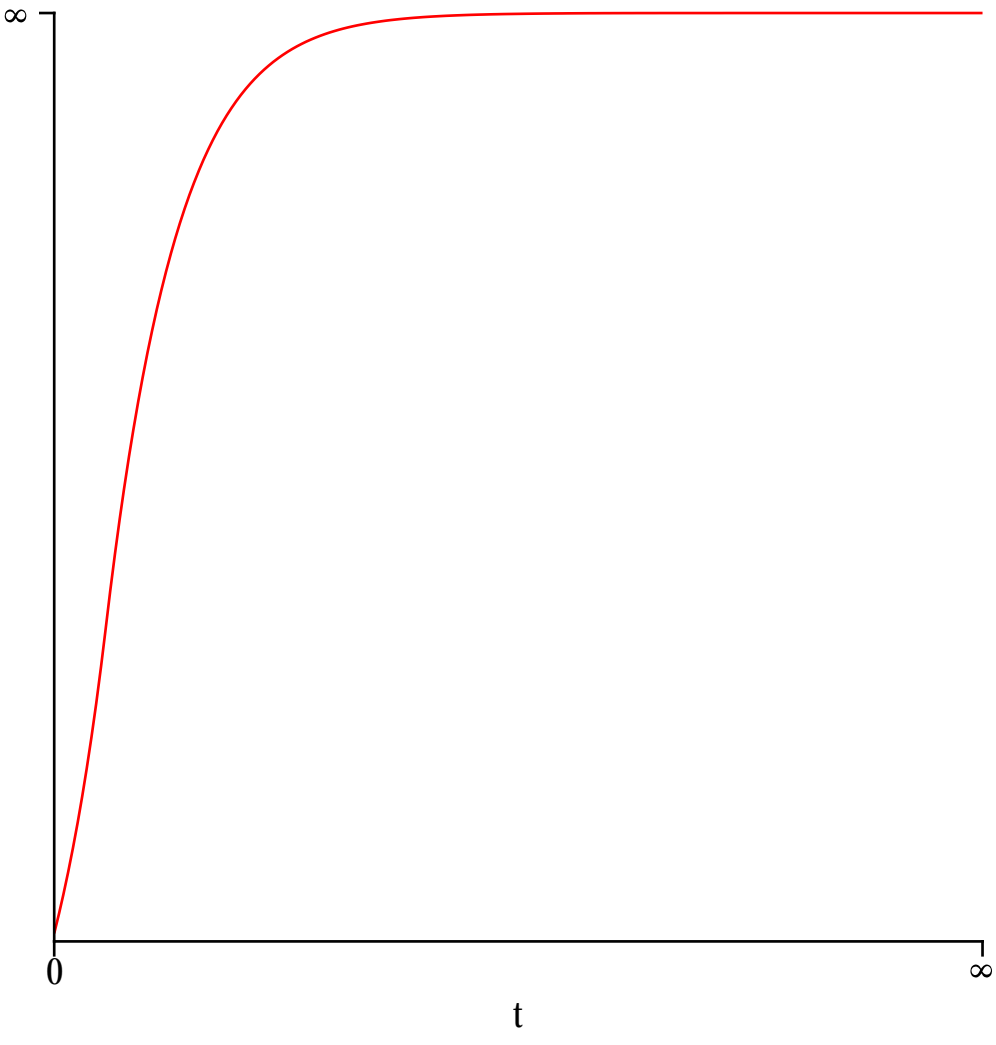
$$pa := [\lambda 1 = 1, \lambda 2 = 2] \tag{24}$$

```
> fa := eval( f, pa );
```

$$fa := e^{t} + e^{2\, t} \tag{25}$$

時間が十分経った（t→+∞）としてプロットします．

```
> plta := plot( fa, t=0..infinity ):
  plots[display]( plta );
```



+∞ に発散します．

（注意）
数学定数 infinity（無限大，N）を用いたプロットは，対象，例えば数列の第 n 項や関数などの定性的な挙動を示したものであり，定量的あるいは数値的に正しい挙動をプロットしたものではありません．

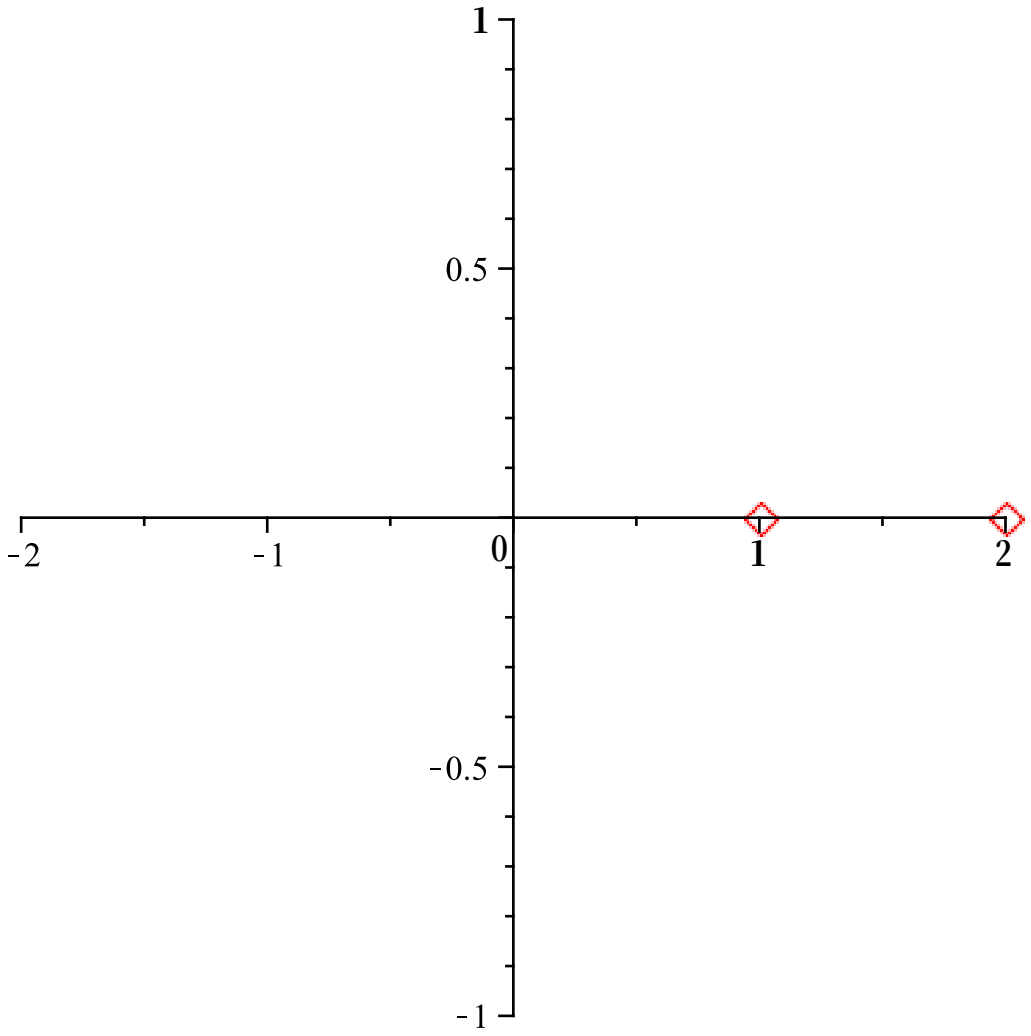
特性根を複素数として定義し，複素平面上にプロットしてみます．

```
> va := [ 1 + 0*I, 2 + 0*I ];
```

(26)

$$va := [1, 2] \tag{26}$$

```
> cplta := plots[complexplot]( va, -2..2, style=point, symbolsize=24 )
  :
  plots[display]( cplta );
```



実軸上にプロットされます.

b) 正と負の場合

特性根を定義します.

```
> pb := [ lambda1 = -1, lambda2 = 1 ];
```

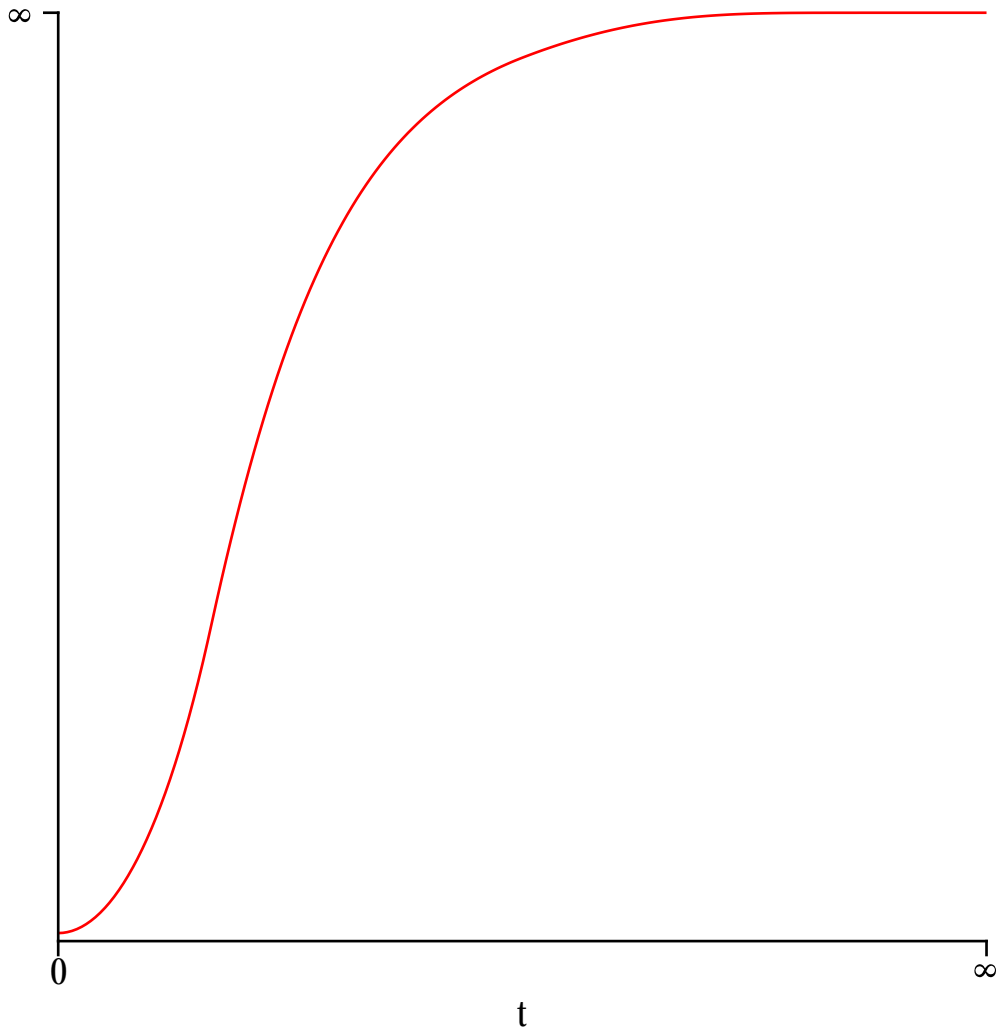
$$pb := [\lambda 1 = -1, \lambda 2 = 1] \tag{27}$$

```
> fb := eval( f, pb );
```

$$fb := e^{-t} + e^{t} \tag{28}$$

時間が十分経った（t→+∞）としてプロットします.

```
> pltb := plot( fb, t=0..infinity ):
  plots[display]( pltb );
```



やはり発散します.

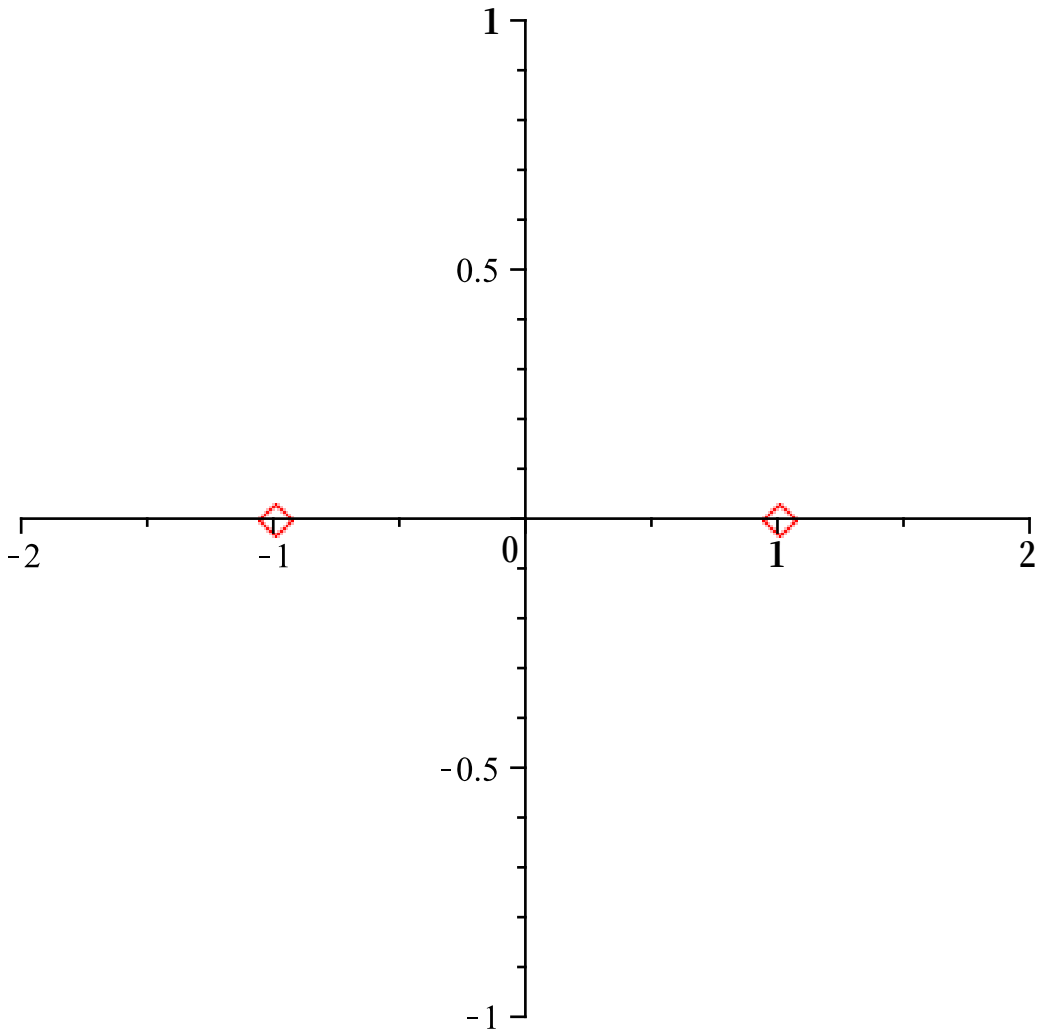
（注意）
数学定数infinity（無限大，N）を用いたプロットは，対象，例えば数列の第 n 項や関数などの定性的な挙動を示したものであり，定量的あるいは数値的に正しい挙動をプロットしたものではありません.

特性根を複素数として定義し，複素平面上にプロットしてみます.

```
> vb := [ -1 + 0*I, 1 + 0*I ];
```

$$vb := [-1, 1] \tag{29}$$

```
> cpltb := plots[complexplot]( vb, -2..2, style=point, symbolsize=24 )
  :
  plots[display]( cpltb );
```



実軸上にプロットされます.

c) ともに負の場合

特性根を定義します.

```
> pc := [ lambda1 = -1, lambda2 = -2 ];
```

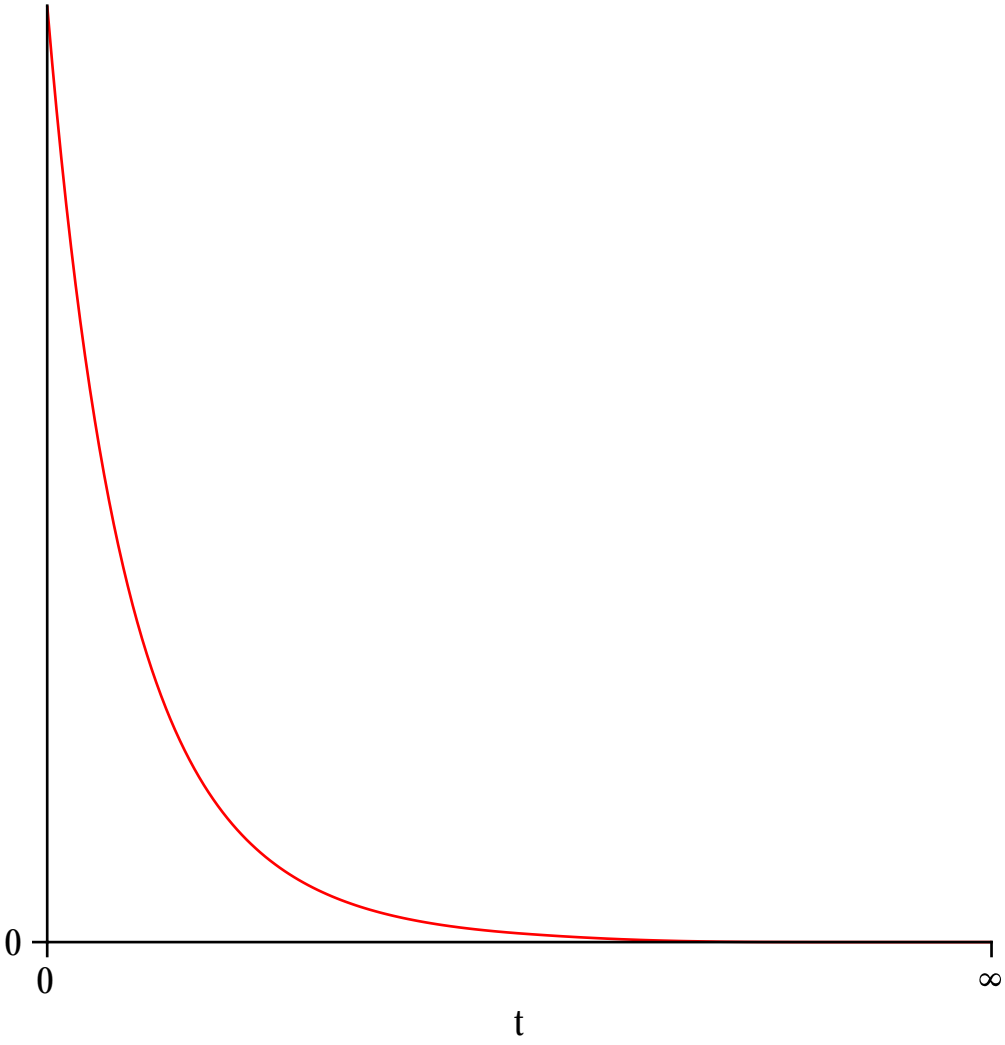
$$pc := [\lambda 1 = -1, \lambda 2 = -2] \tag{30}$$

```
> fc := eval( f, pc );
```

$$fc := e^{-t} + e^{-2t} \tag{31}$$

時間が十分経った（t→+∞）としてプロットします.

```
> pltc := plot( fc, t=0..infinity ):
  plots[display]( pltc );
```



0 に収束します.

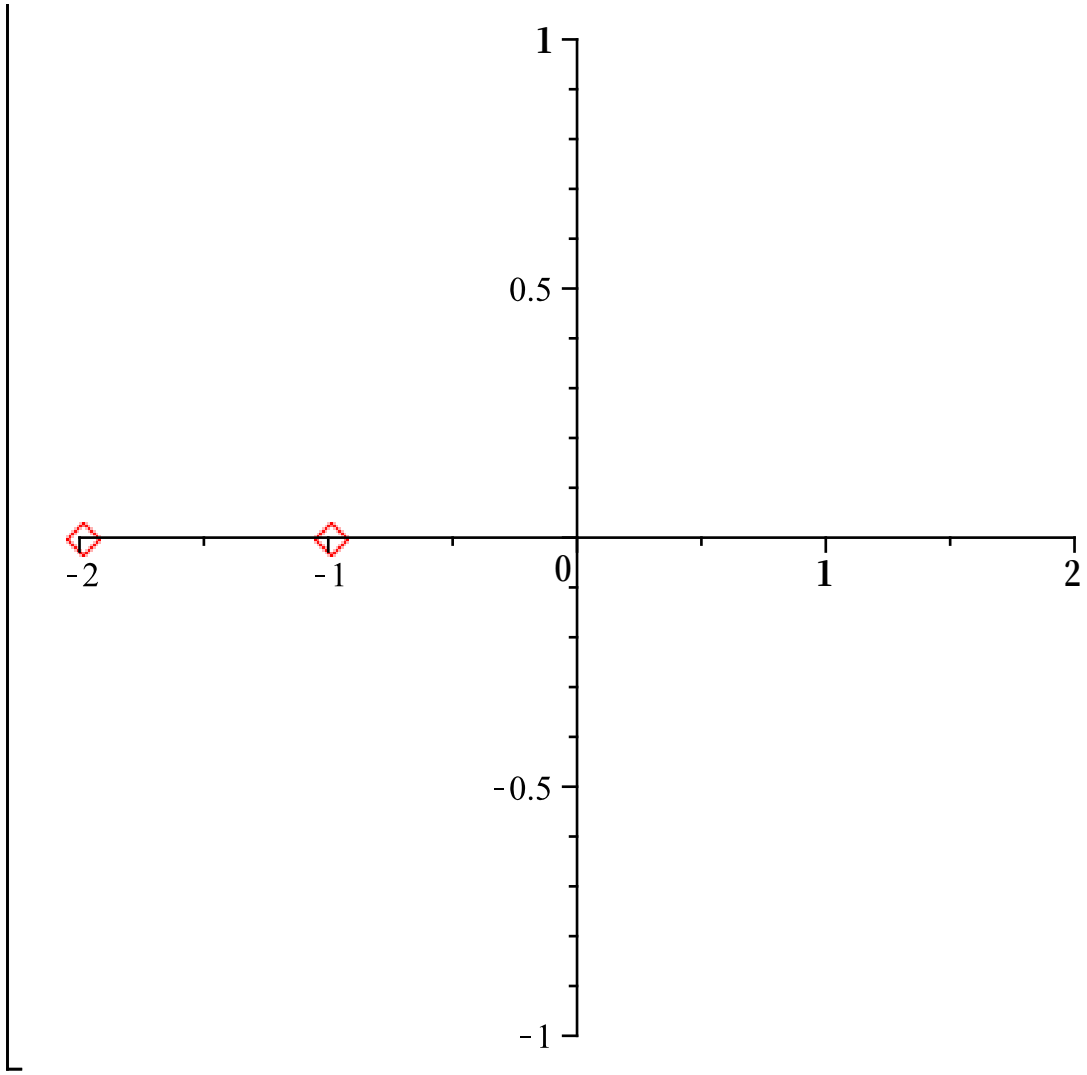
(注意)
数学定数 infinity（無限大，N）を用いたプロットは，対象，例えば数列の第 n 項や関数などの定性的な挙動を示したものであり，定量的あるいは数値的に正しい挙動をプロットしたものではありません.

特性根を複素数として定義し，複素平面上にプロットしてみます.

```
> vc := [ -1 + 0*I, -2 + 0*I ];
```

$$vc := [-1, -2] \tag{32}$$

```
> cpltc := plots[complexplot]( vc, -2..2, style=point, symbolsize=24 )
  :
  plots[display]( cpltc );
```



実軸上にプロットされます.

2) 異なる２つの虚根を持つ場合

d) 実部が正の場合

特性根を定義します

```
> pd := [ lambda1 = 1+I, lambda2 = 1-I ];
```

$$pd := [\lambda 1 = 1 + \mathrm{I}, \lambda 2 = 1 - \mathrm{I}] \tag{33}$$

```
> fd1 := eval( f, pd );
```

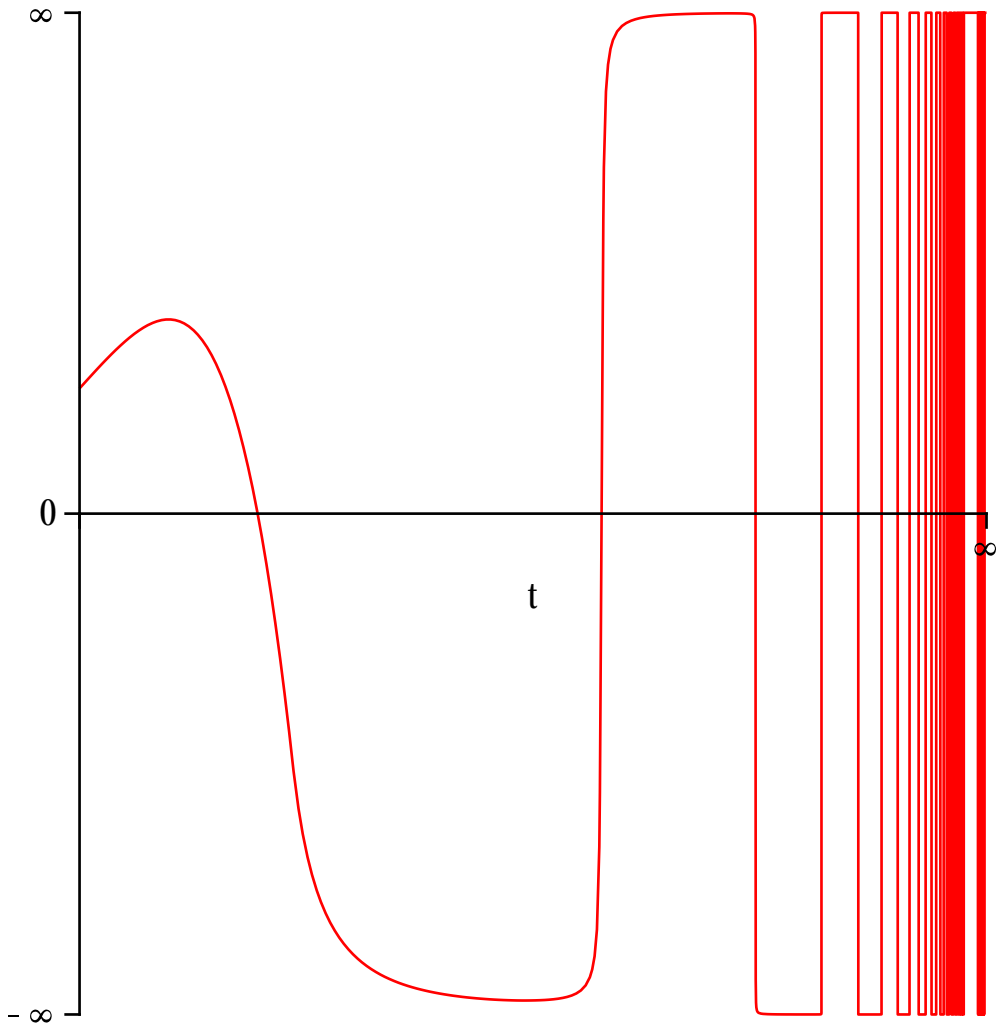
$$fd1 := e^{(1+\mathrm{I})t} + e^{(1-\mathrm{I})t} \tag{34}$$

```
> fd2 := evalc( fd1 );
```

$$fd2 := 2\,e^{t}\cos(t) \tag{35}$$

時間が十分経った（t→+∞）としてプロットします.

```
> pltd := plot( fd2, t=0..infinity ):
  plots[display]( pltd );
```



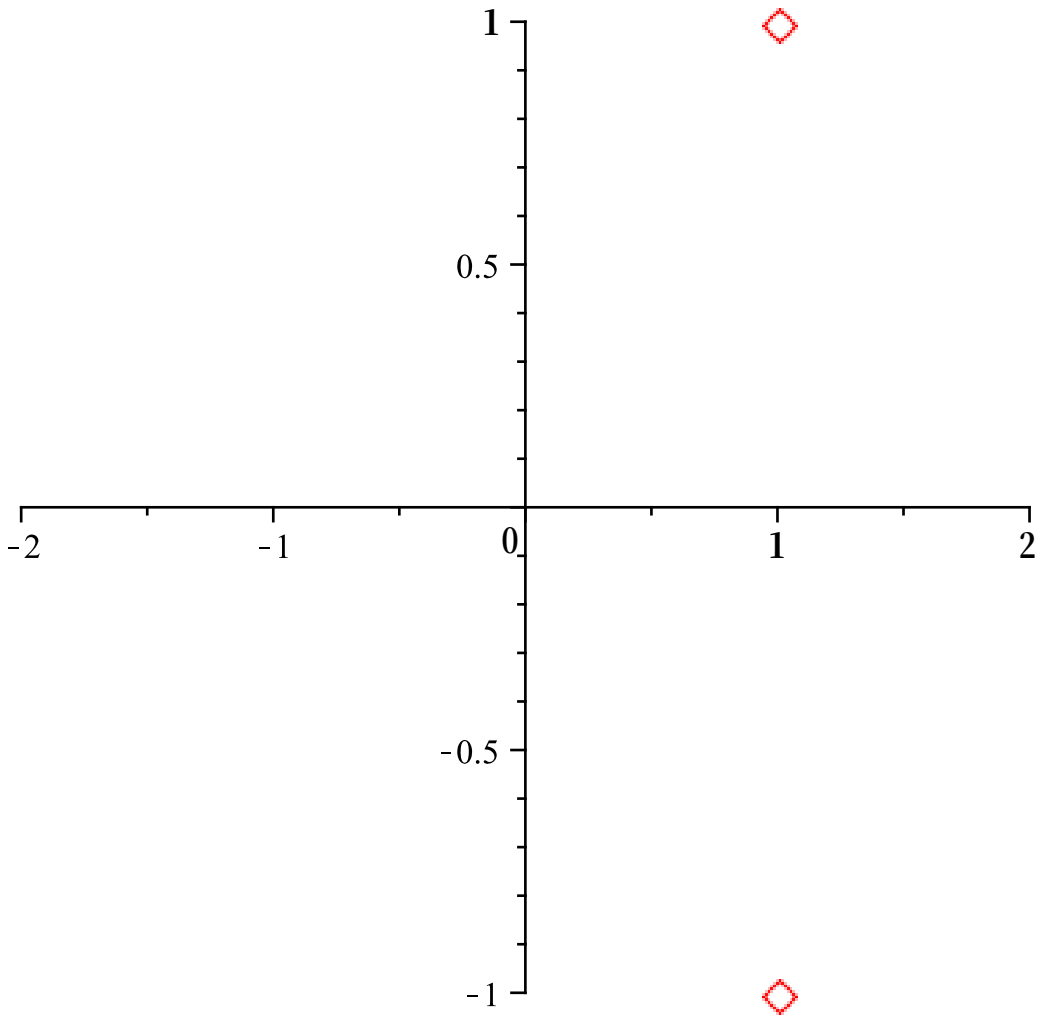
(注意)
数学定数 infinity（無限大，N）を用いたプロットは，対象，例えば数列の第 n 項や関数などの定性的な挙動を示したものであり，定量的あるいは数値的に正しい挙動をプロットしたものではありません.

特性根を複素平面上にプロットしてみます.

```
> vd := [ 1 + 1*I, 1 - 1*I ];
```

$$vd := [1 + I, 1 - I] \tag{36}$$

```
> cpltd := plots[complexplot]( vd, -2..2, style=point, symbolsize=24 )
  :
  plots[display]( cpltd );
```



右半平面にプロットされます.

e) 実部が負の場合

特性根を定義します.

```
> pe := [ lambda1 = -1+I, lambda2 = -1-I ];
```

$$pe := [\lambda 1 = -1 + I, \lambda 2 = -1 - I] \tag{37}$$

```
> fe1 := eval( f, pe );
```

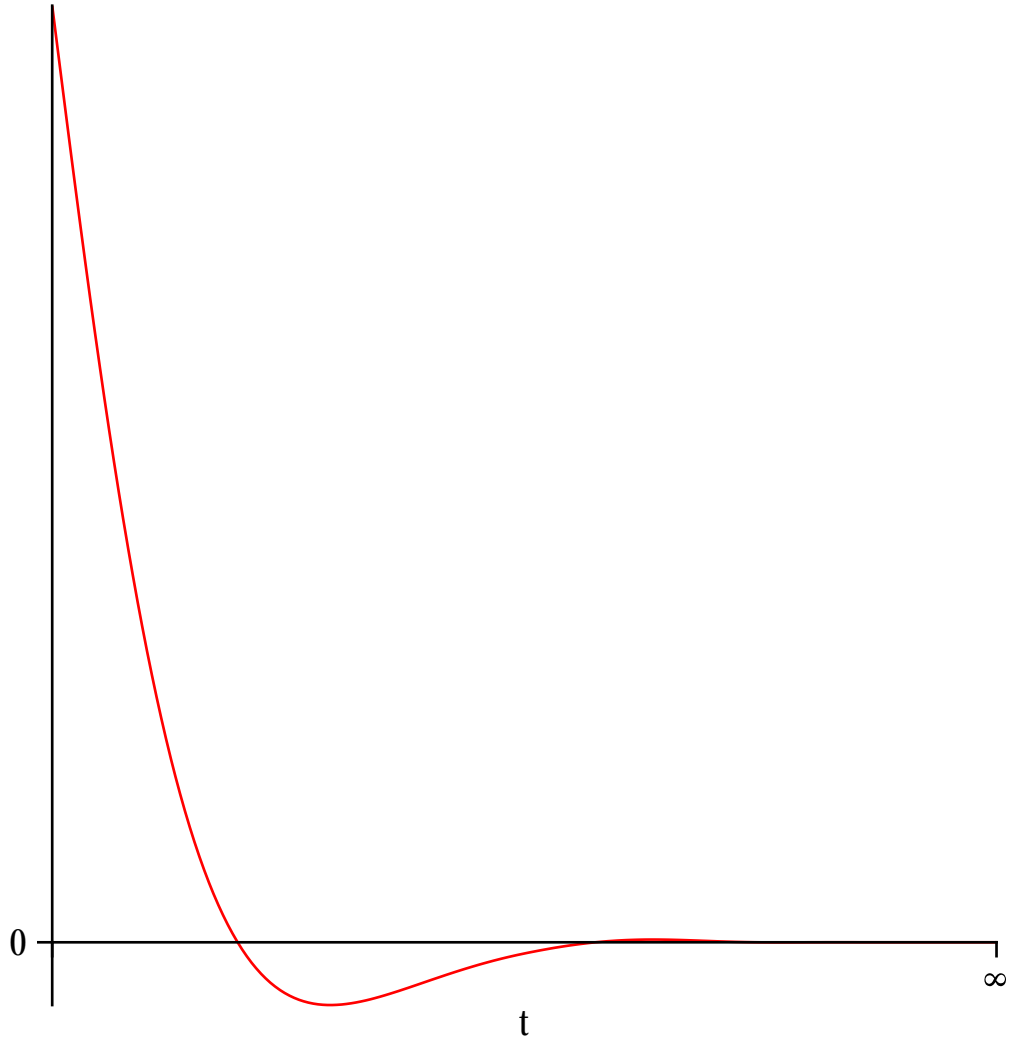
$$fe1 := e^{(-1+I)t} + e^{(-1-I)t} \tag{38}$$

```
> fe2 := evalc( fe1 );
```

$$fe2 := 2\,e^{-t}\cos(t) \tag{39}$$

時間が十分経った（t→+∞）としてプロットします.

```
> plte := plot( fe2, t=0..infinity ):
  plots[display]( plte );
```



0 に収束します.

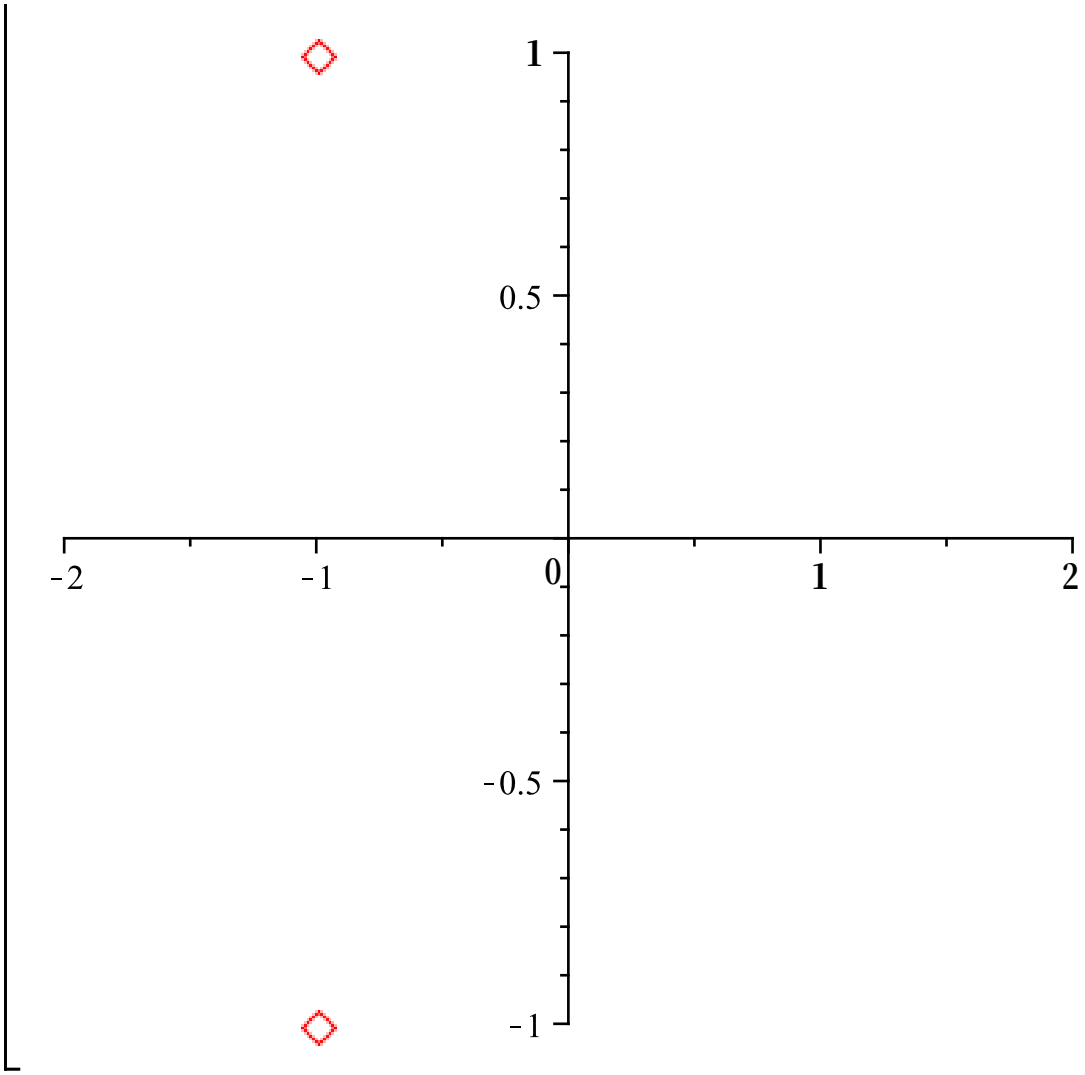
(注意)
数学定数 infinity（無限大，N）を用いたプロットは，対象，例えば数列の第 n 項や関数などの定性的な挙動を示したものであり，定量的あるいは数値的に正しい挙動をプロットしたものではありません.

特性根を複素平面上にプロットしてみます.

```
> ve := [ -1 + 1*I, -1 - 1*I ];
```

$$ve := [-1 + I, -1 - I] \tag{40}$$

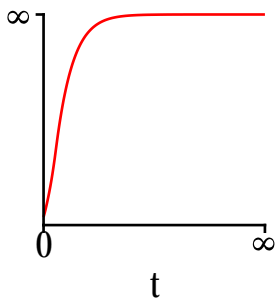
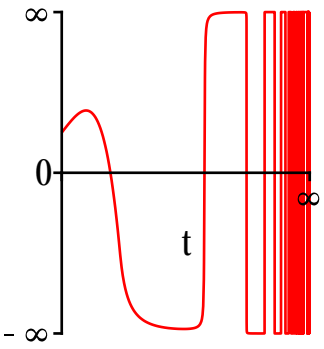
```
> cplte := plots[complexplot]( ve, -2..2, style=point, symbolsize=24 )
  :
  plots[display]( cplte );
```

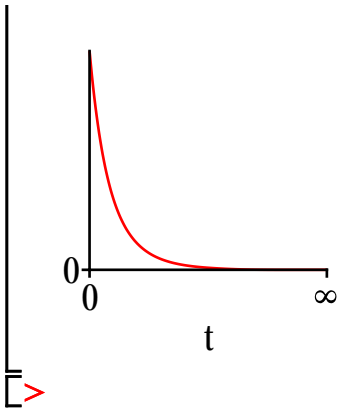


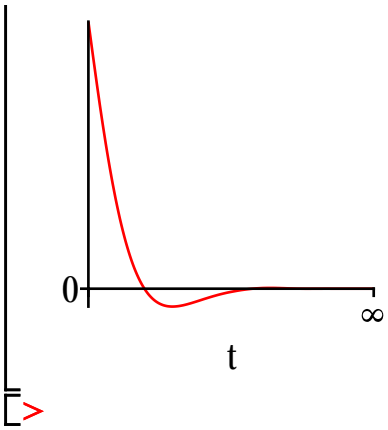
左半平面にプロットされます.

少しだけ考察…

特性根の実部の符号が正のとき発散し，負のとき収束します．また，虚部がある場合，挙動は振動的になります.

	虚部なし（虚部=0）	虚部あり（虚部s 0）
特性根の実部が正	`> plots[display]( plta ) ;`  `>`	`> plots[display]( pltd );`  `>`
特性根の実部が負	`> plots[display]( pltc ) ;`	`> plots[display]( plte );`





(注意)
数学定数infinity（無限大，N）を用いたプロットは，対象，例えば数列の第 n 項や関数などの定性的な挙動を示したものであり，定量的あるいは数値的に正しい挙動をプロットしたものではありません．

2階線形常微分方程式の1階化

初期化します．

```
> restart:
```

2階線形常微分方程式deq1を再定義し，1階線形常微分方程式の形式に変換します．

```
> deq1 := m*diff( x(t), t, t ) + c*diff( x(t), t ) + k*x(t) = 0;
```

$$deq1 := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0 \tag{41}$$

時間で変化する新たな（状態）変数を導入します．x(t) を時間 t について 1 階微分して得られる変数を v(t) とします．

```
> deq2 := diff( x(t), t ) = v(t);
```

$$deq2 := \dot{x}(t) = v(t) \tag{42}$$

deq1にdeq2を代入して，deq3とします．

```
> deq3 := eval( deq1, [ deq2 ] );
```

$$deq3 := m\dot{v}(t) + cv(t) + kx(t) = 0 \tag{43}$$

deq3を $\frac{\mathrm{d}}{\mathrm{d}t}v(t)$ で整理します．

```
> deq4 := isolate( deq3, diff( v(t), t ) );
```

$$deq4 := \dot{v}(t) = \frac{-cv(t) - kx(t)}{m} \tag{44}$$

deq4を展開します．

```
> deq5 := expand( deq4 );
```

$$deq5 := \dot{v}(t) = -\frac{cv(t)}{m} - \frac{kx(t)}{m} \tag{45}$$

行列でdeq2およびdeq5を整理するために，変数 $x(t)$ と $v(t)$ をリスト xL として定義します．

```
> xL := [ x(t), v(t) ];
```

$$xL := [x(t), v(t)] \tag{46}$$

deq2とdeq5を確認します．

```
> deq2;
```

(47)

$$\dot{x}(t) = v(t) \tag{47}$$

```
> deq5;
```

$$\dot{v}(t) = -\frac{c\,v(t)}{m} - \frac{k\,x(t)}{m} \tag{48}$$

deq2，deq5 ともに１次の導関数で整理され，１階常微分方程式が導かれます．

ここで，以下のように１階常微分方程式 deq2，deq5 を行列形式で整理するために，deq2，deq5 の右辺をそれぞれ抽出し，リスト rhsdeqs（方程式系）としてまとめます．

$$\begin{bmatrix} \dot{x}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}$$

```
> rhsdeqs := [ rhs( deq2 ), rhs( deq5 ) ];
```

$$rhsdeqs := \left[v(t), -\frac{c\,v(t)}{m} - \frac{k\,x(t)}{m} \right] \tag{49}$$

GenerateMatrix コマンドを用いて，（１階常微分方程式の系）rhsdeqs から係数行列 A を求めます．

（参考）dummy は，本処理において意味のない出力となります．

```
> ( A, dummy ) := LinearAlgebra[GenerateMatrix]( rhsdeqs, xL );
```

$$A, dummy := \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{50}$$

```
> A;
```

$$\begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \tag{51}$$

ここまでの結果を用いて，以下の式を定義します．

$$\begin{bmatrix} \dot{x}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}$$

行列（ベクトル）表現とするために変数リスト xL をベクトル型に変換します．

```
> xV := convert( xL, Vector );
```

$$xV := \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} \tag{52}$$

はじめに，表記を行列形式に留めるために，subs コマンドを利用して式を整理します．

```
> deq6 := subs( [ m=A, n=xV ], map( diff, xV, t ) = m.n );
```

$$deq6 := \begin{bmatrix} \dot{x}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} \tag{53}$$

以上の式を，eval コマンドを用いて評価すると，行列の演算が実行されます．

```
> deq7 := eval( deq6 );
```

$$deq7 := \begin{bmatrix} \dot{x}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ -\frac{c\,v(t)}{m} - \frac{k\,x(t)}{m} \end{bmatrix} \tag{54}$$

ここで，係数行列 A の固有値を求めてみます．

```
> LinearAlgebra[Eigenvalues]( A );
```

$$\begin{bmatrix} \frac{-c+\sqrt{c^2-4mk}}{2m} \\ -\frac{c+\sqrt{c^2-4mk}}{2m} \end{bmatrix} \tag{55}$$

これは，特性方程式の特性根と同じ値になります．

特性方程式： $m\lambda^2 + c\lambda + k = 0$

特性根： $\lambda_1, \lambda_2 := \frac{-c+\sqrt{c^2-4mk}}{2m}, -\frac{c+\sqrt{c^2-4mk}}{2m}$

すなわち，

$$deq1 := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

の一般解：

$$x(t) = \_C1\, e^{\frac{\left(-c+\sqrt{c^2-4mk}\right)t}{2m}} + \_C2\, e^{-\frac{\left(c+\sqrt{c^2-4mk}\right)t}{2m}}$$

の挙動は係数行列 A

$$\begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix}$$

の固有値を調べることよっても判別することができます．

$$deq6 := \begin{bmatrix} \dot{x}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}$$

を（制御工学の分野で）状態方程式と呼びます．また，A を係数行列と呼び，制御システムの理論的な設計に用いられます．

（参考）連立微分方程式の解法

２つの１階常微分方程式

$$deq2 := \dot{x}(t) = v(t)$$

および

$$deq5 := \dot{v}(t) = -\frac{c\,v(t)}{m} - \frac{k\,x(t)}{m}$$

を連立させて（解析）解を求めてみます.

dsolveコマンドを用いて連立微分方程式を解きます.

```
> sol := dsolve( [ deq2, deq5 ] );
```

$$sol := \left\{ v(t) = \frac{\left(-\frac{c}{2} - \frac{\sqrt{c^2-4mk}}{2}\right) \_C2\, e^{-\frac{\left(c+\sqrt{c^2-4mk}\right)t}{2m}}}{m} + \frac{\left(-\frac{c}{2} + \frac{\sqrt{c^2-4mk}}{2}\right) \_C1\, e^{\frac{\left(-c+\sqrt{c^2-4mk}\right)t}{2m}}}{m}, x(t) = \_C1\, e^{\frac{\left(-c+\sqrt{c^2-4mk}\right)t}{2m}} + \_C2\, e^{-\frac{\left(c+\sqrt{c^2-4mk}\right)t}{2m}} \right\} \tag{56}$$

解を抽出します.

```
> sol[1];
```

$$v(t) = \frac{\left(-\frac{c}{2} - \frac{\sqrt{c^2-4mk}}{2}\right) \_C2\, e^{-\frac{\left(c+\sqrt{c^2-4mk}\right)t}{2m}}}{m} + \frac{\left(-\frac{c}{2} + \frac{\sqrt{c^2-4mk}}{2}\right) \_C1\, e^{\frac{\left(-c+\sqrt{c^2-4mk}\right)t}{2m}}}{m} \tag{57}$$

```
> sol[2];
```

$$x(t) = \_C1\, e^{\frac{\left(-c+\sqrt{c^2-4mk}\right)t}{2m}} + \_C2\, e^{-\frac{\left(c+\sqrt{c^2-4mk}\right)t}{2m}} \tag{58}$$

初期値問題

初期化します.

```
> restart;
```

２階線形常微分方程式を再定義します.

```
> ode := m*diff( x(t), t, t ) + c*diff( x(t), t ) + k*x(t) = 0;
```

$$ode := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0 \tag{59}$$

式

$$ode := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

の解を求めます.

```
> dsolve( ode );
```

$$x(t) = \_C1\, e^{\frac{\left(-c+\sqrt{c^2-4km}\right)t}{2m}} + \_C2\, e^{-\frac{\left(c+\sqrt{c^2-4km}\right)t}{2m}} \tag{60}$$

初期条件を以下のように定義します.

$$x(0) = -1,\ \frac{dx(0)}{dt} = 0.$$

初期条件の定義には，微分演算子 D を用い，シーケンスとして定義します．

```
> ics := x(0)=-1, D(x)(0)=0;
```

$$ics := x(0) = -1, \mathrm{D}(x)(0) = 0 \tag{61}$$

{ }（集合の記号）を用いて，方程式と初期条件を整理します（括ります）．

適当な係数 m, c, k を定義し，方程式に代入します．

```
> params := [ m=1, c=1/3, k=11 ];
```

$$params := \left[m = 1, c = \frac{1}{3}, k = 11\right] \tag{62}$$

式

$$ode := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

に

$$params := \left[m = 1, c = \frac{1}{3}, k = 11\right]$$

を代入します．

```
> odea := eval( ode, params );
```

$$odea := \ddot{x}(t) + \frac{\dot{x}(t)}{3} + 11x(t) = 0 \tag{63}$$

初期値問題として解きます．

```
> sol := dsolve( { odea, ics } );
```

$$sol := x(t) = -\frac{\sqrt{395}\, e^{-\frac{t}{6}} \sin\left(\frac{\sqrt{395}\, t}{6}\right)}{395} - e^{-\frac{t}{6}} \cos\left(\frac{\sqrt{395}\, t}{6}\right) \tag{64}$$

求められた解は，特殊解と呼ばれます．

なお，引数は以下のような集合になっています．

（復習） 集合の定義において，順番は保持されず，重複する要素は省かれます．

```
> { ode, ics };
```

$$\{m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0, x(0) = -1, \mathrm{D}(x)(0) = 0\} \tag{65}$$

式

$$sol := x(t) = -\frac{\sqrt{395}\, e^{-\frac{t}{6}} \sin\left(\frac{\sqrt{395}\, t}{6}\right)}{395} - e^{-\frac{t}{6}} \cos\left(\frac{\sqrt{395}\, t}{6}\right)$$

の右辺を eq1 に割り当てます．

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

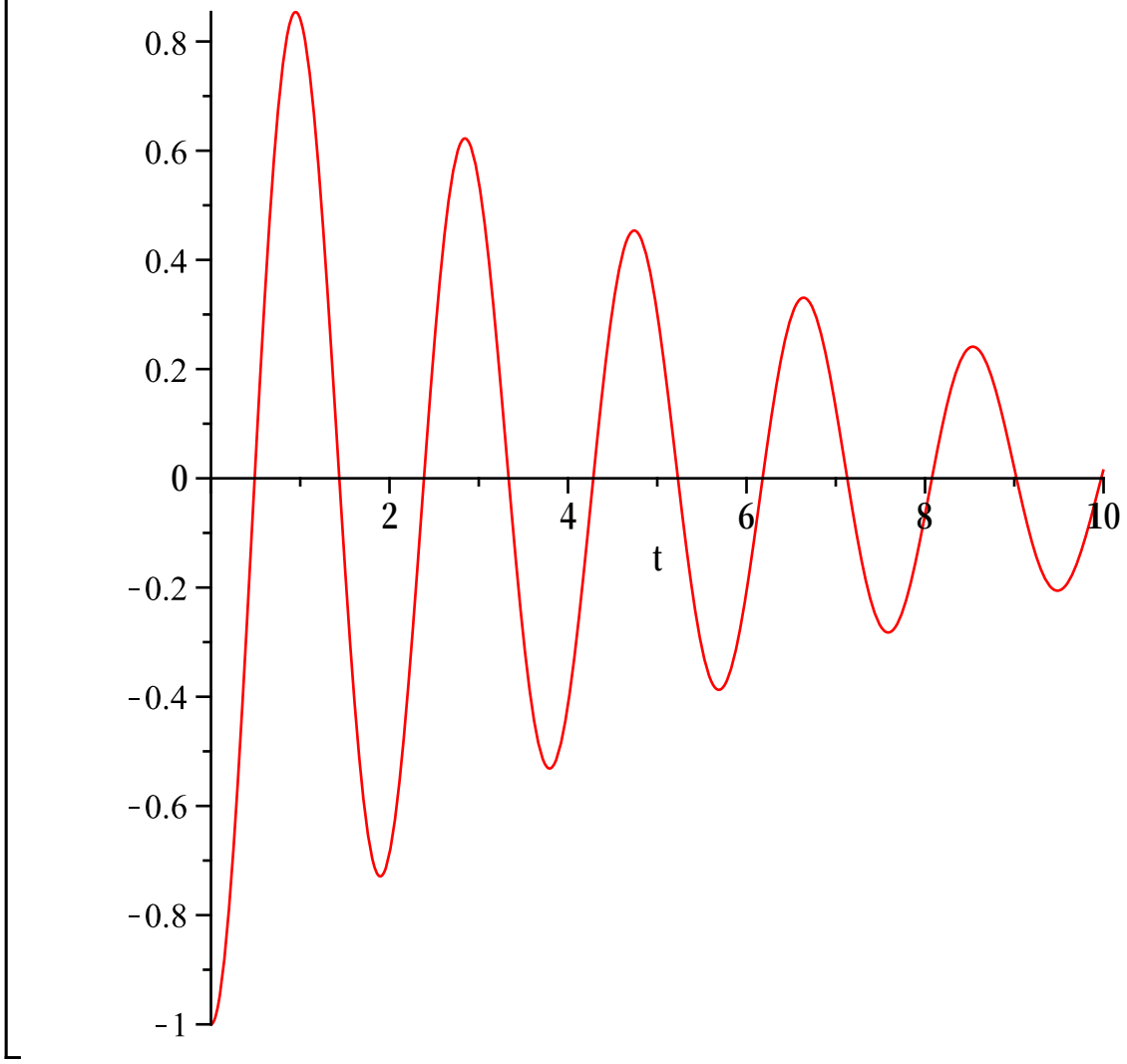
$$eq1 := -\frac{\sqrt{395}\, e^{-\frac{t}{6}} \sin\left(\frac{\sqrt{395}\, t}{6}\right)}{395} - e^{-\frac{t}{6}} \cos\left(\frac{\sqrt{395}\, t}{6}\right) \tag{66}$$

式

$$eq1 := -\frac{\sqrt{395}\, e^{-\frac{t}{6}} \sin\left(\frac{\sqrt{395}\, t}{6}\right)}{395} - e^{-\frac{t}{6}} \cos\left(\frac{\sqrt{395}\, t}{6}\right)$$

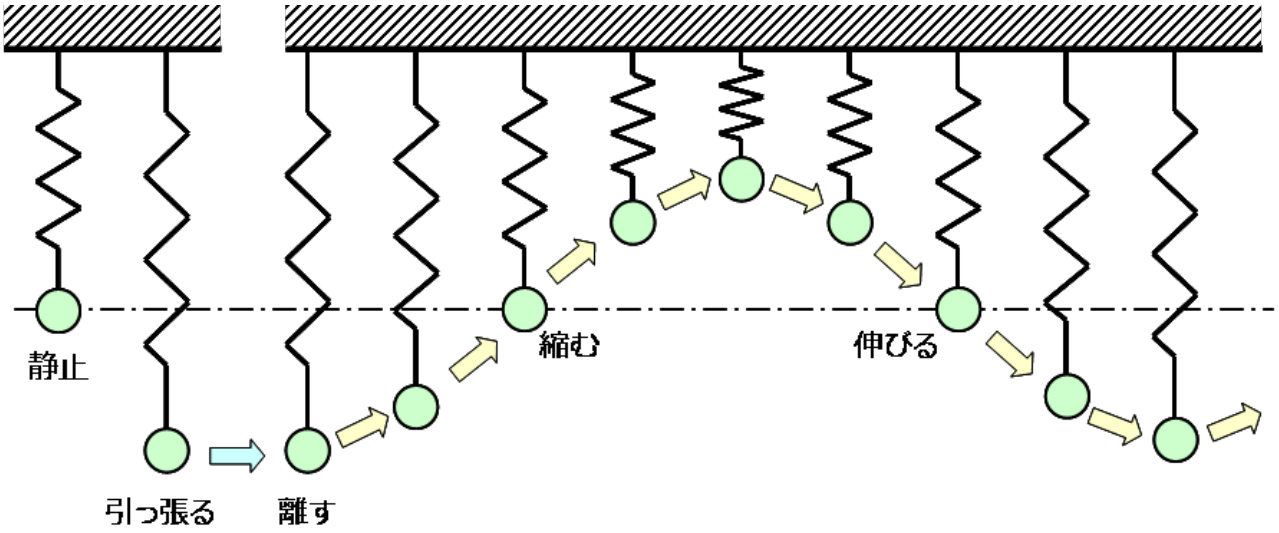
をプロットします.

```
> plot( eq1, t=0..10 );
```



t = 0のときの値は，定義した初期値$x(0) = -1$になっています.

（参考）自由振動の原理



初期条件をパラメータ $x0$ として定義しなおします.

```
> pics := x(0) = x0, D(x)(0) = 0;
```

$$pics := x(0) = x0,\ \mathrm{D}(x)(0) = 0 \tag{67}$$

定義した初期条件

$$pics := x(0) = x0,\ \mathrm{D}(x)(0) = 0$$

で方程式

$$ode := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

の解を求めます.

```
> psol := dsolve( { odea, pics } );
```

$$psol := x(t) = \frac{x0\sqrt{395}\, e^{-\frac{t}{6}} \sin\left(\frac{\sqrt{395}\, t}{6}\right)}{395} + x0\, e^{-\frac{t}{6}} \cos\left(\frac{\sqrt{395}\, t}{6}\right) \tag{68}$$

$$psol := x(t) = \frac{x0\sqrt{395}\, e^{-\frac{t}{6}} \sin\left(\frac{\sqrt{395}\, t}{6}\right)}{395} + x0\, e^{-\frac{t}{6}} \cos\left(\frac{\sqrt{395}\, t}{6}\right)$$

の右辺を eq2 に割り当てます.

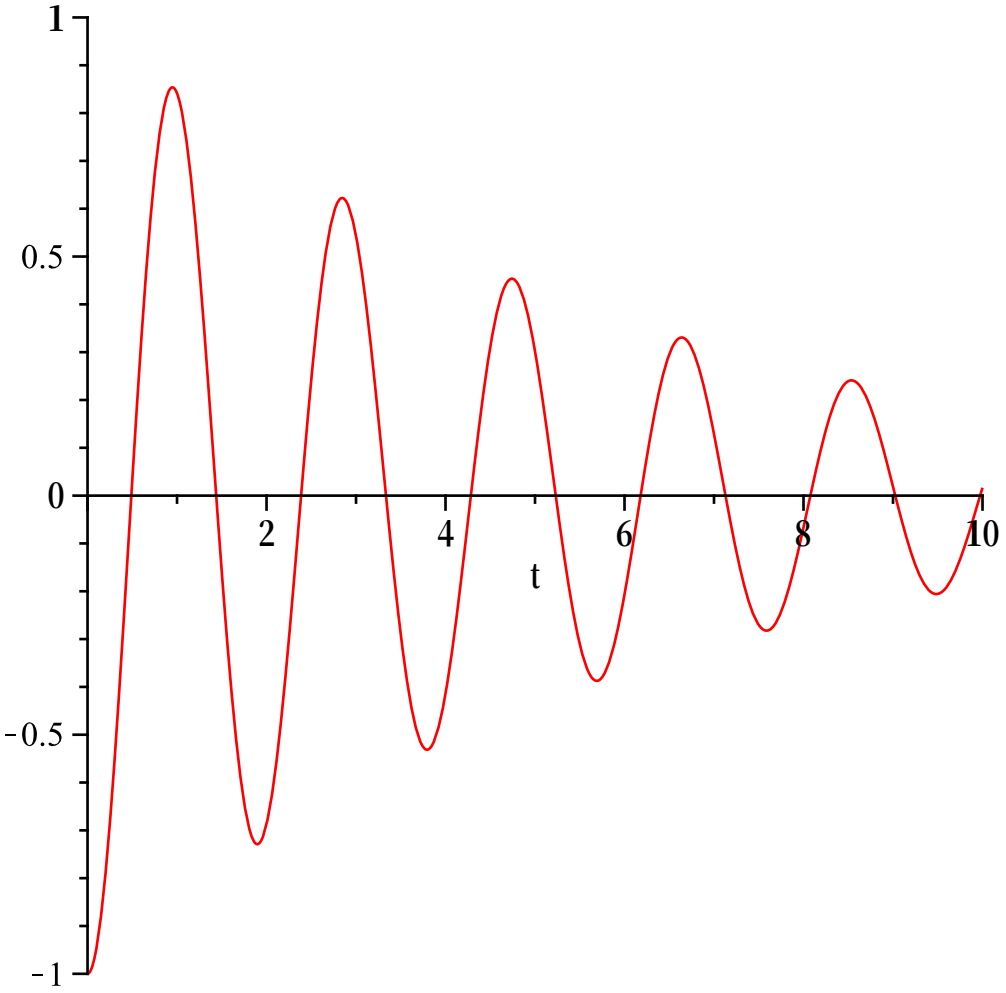
```
> eq2 := rhs(psol);
```

$$eq2 := \frac{x0\sqrt{395}\, e^{-\frac{t}{6}} \sin\left(\frac{\sqrt{395}\, t}{6}\right)}{395} + x0\, e^{-\frac{t}{6}} \cos\left(\frac{\sqrt{395}\, t}{6}\right) \tag{69}$$

初期値を変化させて，アニメーションを作成します.

```
> plots[animate]( plot, [ eq2, t=0..10], x0=-1..1 );
```

$x0 = -1.$



（参考）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．

ラプラス変換による初期値問題の解法

初期化します．

```
> restart:
```

積分変換は，数学的に（あるいは物理的に）関数$f(t)$をある領域tから別の領域αへ変換する積分です．

$$g(\alpha) = \int_a^b f(t)\, K(\alpha, t)\, \mathrm{d}t$$

積分変換として，ラプラス変換やフーリエ変換がよく知られています．

例えば，ラプラス変換は，微分方程式を代数方程式に変換します（計算が簡単になります）．また，フーリエ変換は，時系列データを時間領域から周波数領域へ変換します（見えないものが見えてきます）．

積分変換を利用するには，inttransパッケージのコマンドを使用します．

```
> with( inttrans );
```

[*addtable*, *fourier*, *fouriercos*, *fouriersin*, *hankel*, *hilbert*, *invfourier*, *invhilbert*, *invlaplace*, *invmellin*, *laplace*, *mellin*, *savetable*] (70)

ラプラス変換を用いて微分方程式の初期値問題を解いてみます.

ここで，２階線形常微分方程式を再定義します.

```
> deq := diff( x(t), t,t ) + 2*diff( x(t), t ) - 3*x(t) = 0;
```

$$deq := \ddot{x}(t) + 2\dot{x}(t) - 3x(t) = 0 \tag{71}$$

初期値$x(0)=0,\ \frac{\mathrm{d}}{\mathrm{d}t}x(0)=1$を集合{ }として定義します.

（参考）Dは微分演算子になります.

```
> ics := { x(0)=0, D(x)(0)=1 };
```

$$ics := \{x(0)=0, \mathrm{D}(x)(0)=1\} \tag{72}$$

微分方程式をラプラス変換します（領域tから領域sに変換します）.

```
> Leq1 := laplace( deq, t, s );
```

$$Leq1 := s^2 laplace(x(t),t,s) - \mathrm{D}(x)(0) - sx(0) + 2s\,laplace(x(t),t,s) - 2x(0) - 3\,laplace(x(t),t,s) = 0 \tag{73}$$

式を見やすくするために，いったん$laplace(x(t),t,s)$をXに置き換えます.

```
> Leq2 := subs( laplace( x(t), t, s )=X, Leq1 );
```

$$Leq2 := s^2X - \mathrm{D}(x)(0) - sx(0) + 2sX - 2x(0) - 3X = 0 \tag{74}$$

初期条件を式に代入・評価します.

```
> Leq3 := eval( Leq2, ics );
```

$$Leq3 := s^2X - 1 + 2sX - 3X = 0 \tag{75}$$

Xで式を整理します.

```
> Leq4 := isolate( Leq3, X );
```

$$Leq4 := X = \frac{1}{s^2+2s-3} \tag{76}$$

Xを元の$laplace(x(t),t,s)$に置き換えます.

```
> Leq5 := subs( X=laplace( x(t), t, s ), Leq4 );
```

$$Leq5 := laplace(x(t),t,s) = \frac{1}{s^2+2s-3} \tag{77}$$

右辺を部分分数に変換します.

```
> Leq6 := convert( Leq5, parfrac );
```

$$Leq6 := laplace(x(t),t,s) = \frac{1}{4(s-1)} - \frac{1}{4(s+3)} \tag{78}$$

式

$$Leq6 := laplace(x(t),t,s) = \frac{1}{4(s-1)} - \frac{1}{4(s+3)}$$

を**逆**ラプラス変換します(領域sから領域tに変換します).

```
> Leq7 := invlaplace( Leq6, s, t );
```

$$Leq7 := x(t) = \frac{\mathrm{e}^{t}}{4} - \frac{\mathrm{e}^{-3t}}{4} \tag{79}$$

初期値問題の解になります.

（参考）以下は，dsolve コマンドを用いた解になります.

```
> dsol := dsolve( { deq } union ics );
```

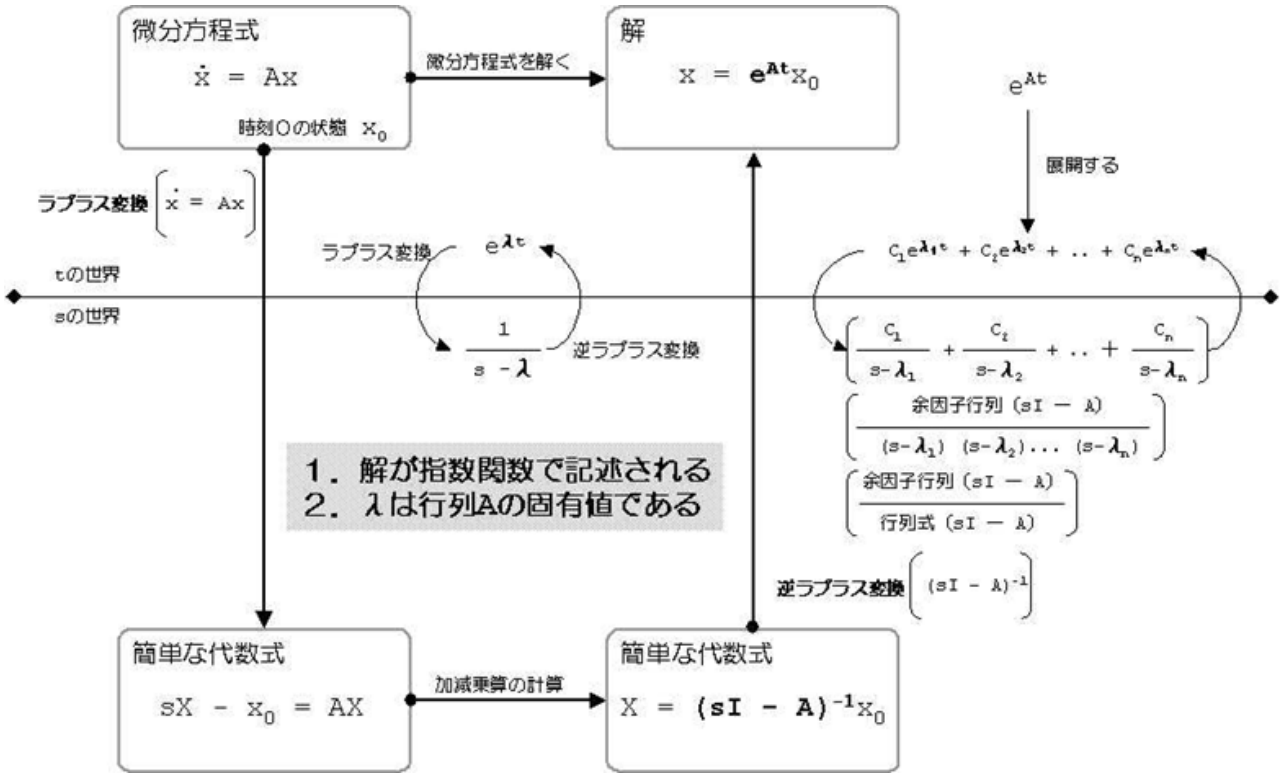
$$dsol := x(t) = \frac{e^{t}}{4} - \frac{e^{-3t}}{4} \tag{80}$$

（参考）union は集合の和（演算子）になります.

```
> {deq} union ics;
```

$$\{\ddot{x}(t) + 2\dot{x}(t) - 3x(t) = 0, x(0) = 0, D(x)(0) = 1\} \tag{81}$$

（参考）t の世界と s の世界



常微分方程式（ODE）の数値解法

初期化します.

```
> restart;
```

２階の線形常微分方程式を定義します.

```
> ode := m*diff( x(t), t,t ) + c*diff( x(t), t ) + k*x(t) = 0;
```

$$ode := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0 \tag{82}$$

パラメータを定義します.

```
> params := [ m=1, c=1/3, k=11 ];
```

$$params := \left[m = 1, c = \frac{1}{3}, k = 11\right] \tag{83}$$

定義したパラメータ

$$params := \left[m = 1, c = \frac{1}{3}, k = 11\right]$$

を式

$$ode := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

に代入します.

```
> odea := eval( ode, params );
```

$$odea := \ddot{x}(t) + \frac{\dot{x}(t)}{3} + 11\,x(t) = 0 \tag{84}$$

初期条件ｉｃｓを定義します.

```
> ics := x(0)=-1, D(x)(0)=0;
```

$$ics := x(0) = -1, \mathrm{D}(x)(0) = 0 \tag{85}$$

数値解を求めるためのオプション（numeric）を指定します.

```
> dsol := dsolve( {odea,ics}, numeric );
```

(86)

数値解を求める処理が一種のプログラム形式で出力されます．すなわち，ｄｓｏｌにある値を引数として与えることで，その値に対する出力値が計算されます.

このプログラム形式の数値解を，**解関数**と呼びます．さらに，Maple の中では一般的な名称として，**プロシージャ**（処理の単位）という言葉が使用されます.

$t = 0$ として，解関数ｄｓｏｌを計算してみます.

```
> dsol(0);
```

$$[t = 0., x(t) = -1., \dot{x}(t) = 0.] \tag{87}$$

$t = 0$ のときの $x(0)$ と $\frac{d}{dt}x(0)$ の値が計算されます.

同様に $t = 1$ のときの値を計算してみます.

```
> dsol(1);
```

$$[t = 1., x(t) = 0.841399896866335, \dot{x}(t) = -0.477906337087004] \tag{88}$$

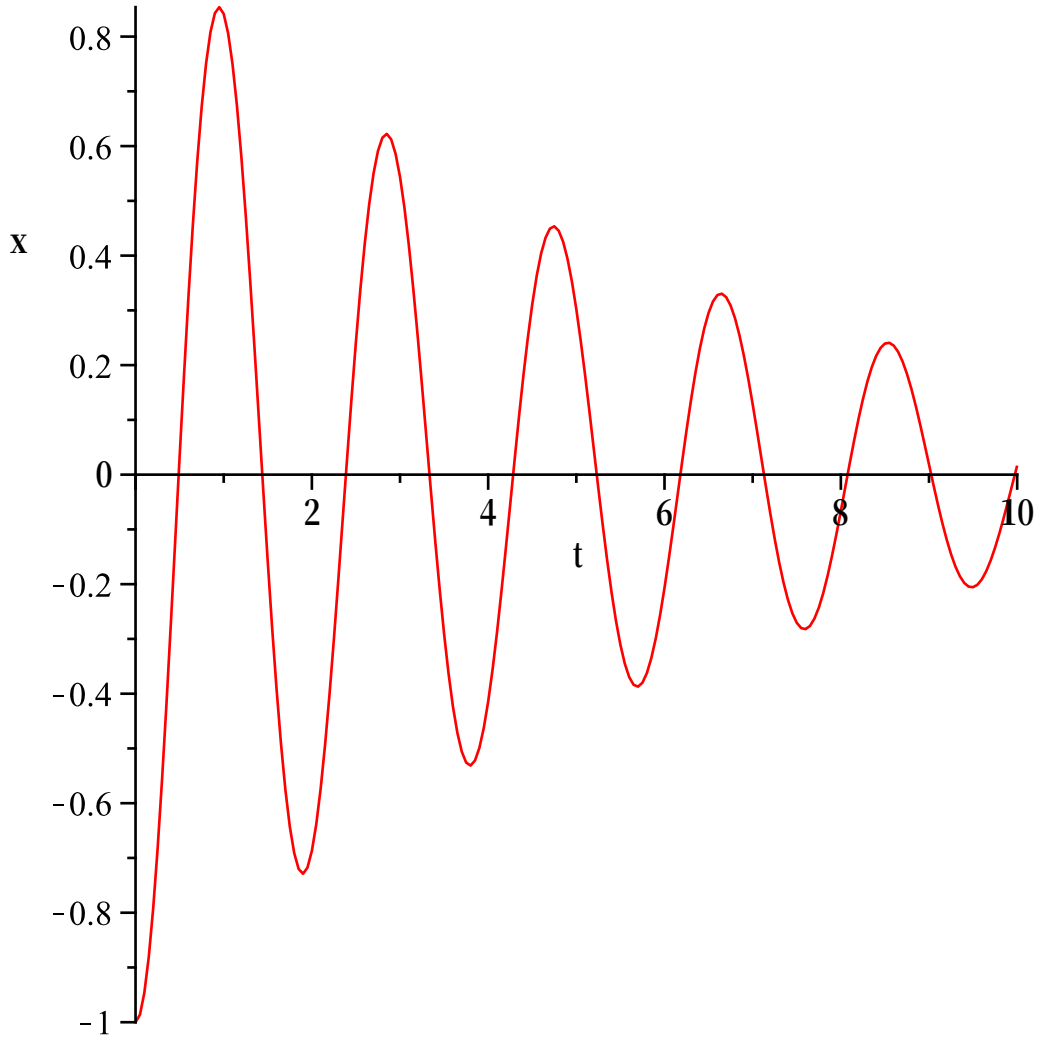
以下は，$t = \pi$ のときの値です.

```
> dsol(Pi);
```

$$[t = 3.14159265358979,\ x(t) = 0.353992482077245,\ \dot{x}(t) = -1.63544709648800] \quad (89)$$

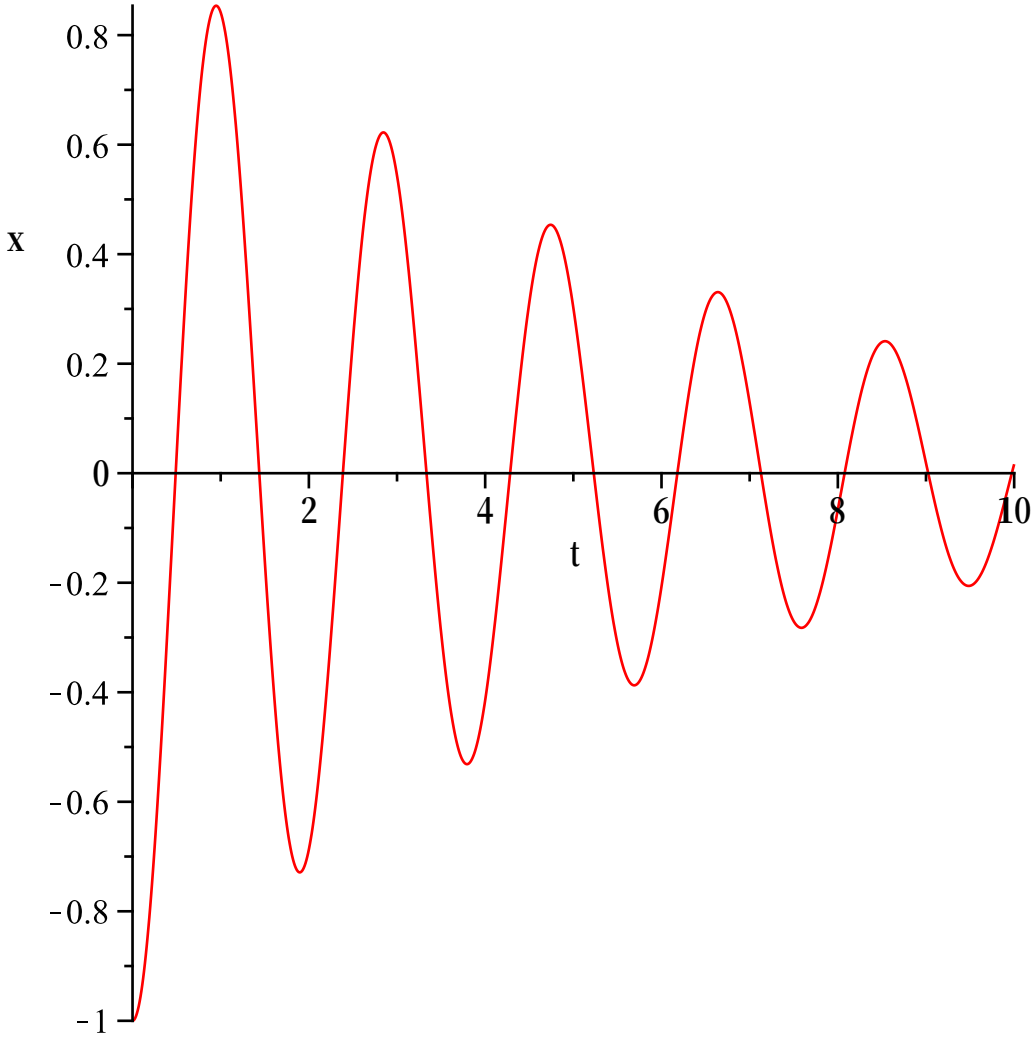
微分方程式の解関数は，plotsパッケージのodeplotコマンドを用いて，横軸と縦軸を指定しプロットすることができます．

```
> plots[odeplot]( dsol, [t, x(t)], 0..10 );
```



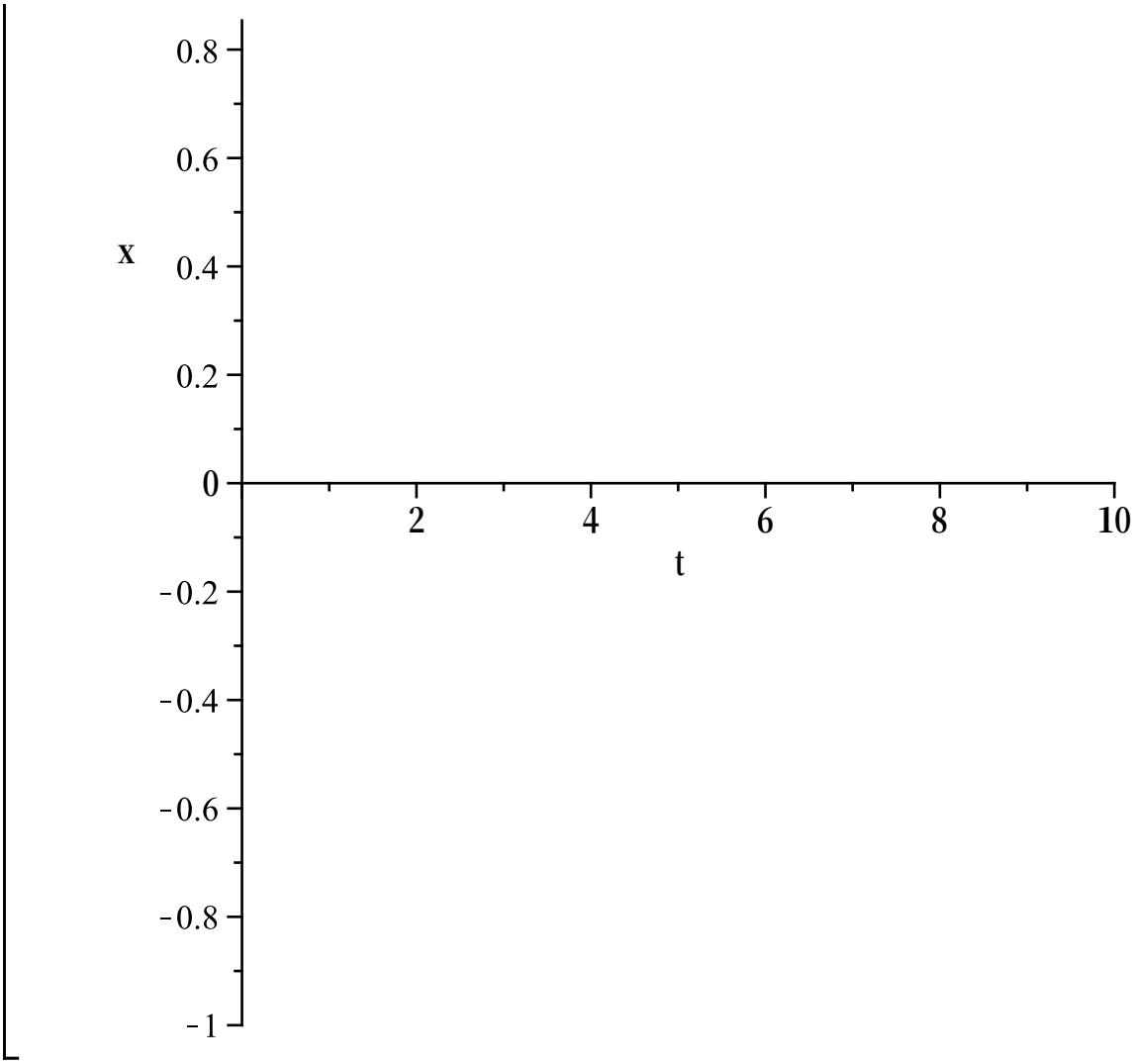
オプションnumpointsを用いて，プロット点数を多めに指定することによって，多少なめらかな曲線にすることができます．ここで，numpoints=1000として，再プロットしてみます．

```
> plots[odeplot]( dsol, [t, x(t)], 0..10, numpoints=1000 );
```



オプションframesでフレーム数を指定することによって，アニメーションを作成することができます．

```
> plots[odeplot]( dsol, [ t, x(t) ], 0..10, numpoints=1000, frames=52 );
```



（参考）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．

【参考】物体の運動の可視化

以下に，数値的に求められた物体の運動（解関数の計算結果）を可視化します．

•計算時間

```
> endT := 10;
```

$$endT := 10 \tag{90}$$

•データ数

```
> N := 100;
```

$$N := 100 \tag{91}$$

•ステップ数

```
> dT := endT/N;
```

$$dT := \frac{1}{10} \tag{92}$$

•時間をリストとして定義

```
> T := [ seq( evalf( k*dT ), k=0..N ) ]:
```

•長方形（物体）を描画するための座標
□スケーリング係数 s は長方形の大きさを調整します．

```
> s := 2; # スケーリング係数
  ( x1, y1 ) := -0.8/s, -0.5/s; # 座標1
  ( x2, y2 ) :=  0.8/s,  0.5/s; # 座標2
```

$$s := 2$$

$$x1, y1 := -0.4000000000, -0.2500000000$$

$$x2, y2 := 0.4000000000, 0.2500000000 \quad (93)$$

以下に解関数の結果から所望の値だけを抽出する操作を段階的に示します.

- 解関数の操作（1）
  - 時間の抽出

```
> T[1];
  T[N+1];
```

$$0.$$

$$10. \quad (94)$$

- 解関数の操作（2）

```
> dsol( T[1] );
  dsol( T[N+1] );
```

$$[t = 0., x(t) = -1., \dot{x}(t) = 0.]$$

$$[t = 10., x(t) = 0.0164981510659941, \dot{x}(t) = 0.621293952341519] \quad (95)$$

- 解関数の操作（3）

```
> dsol( T[1] )[];
  dsol( T[N+1] )[];
```

$$t = 0., x(t) = -1., \dot{x}(t) = 0.$$

$$t = 10., x(t) = 0.0164981510659941, \dot{x}(t) = 0.621293952341519 \quad (96)$$

- 解関数の操作（4）

```
> ( dsol( T[1]   )[] )[2];
  ( dsol( T[N+1] )[] )[2];
```

$$x(t) = -1.$$

$$x(t) = 0.0164981510659941 \quad (97)$$

- 解関数の操作（5）

```
> rhs( ( dsol( T[1]   )[] )[2] );
  rhs( ( dsol( T[N+1] )[] )[2] );
```

$$-1.$$

$$0.0164981510659941 \quad (98)$$

以上の操作を応用して，アニメーションを作成します.

- 物体の位置の描画オブジェクトを生成
  - 解析解と d[i] の定義が異なります.
  - 解析解の場合：d[i] := evalf(Re(eval(eq2, t=T[i]))):

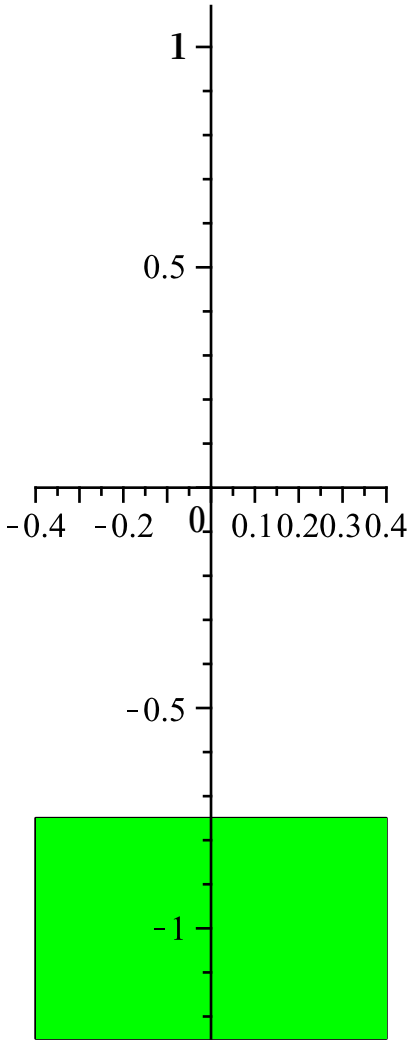
```
> for i from 1 to N+1 do

    # オフセットの計算
    d[i] := rhs( ( dsol( T[i] )[] )[2] ):

    # 物体の位置
    r[i] := plots[display](
      plottools[rectangle]( [ x1, y1+d[i] ],
                            [ x2, y2+d[i] ],
                            color=green ) ):

  end do:
```

- 描画オブジェクトをシーケンスとして生成

```
> req := [ seq( r[i], i=1..N+1 ) ]:
```

・アニメーションの生成

```
> plots[display]( req, insequence=true, scaling=constrained );
```

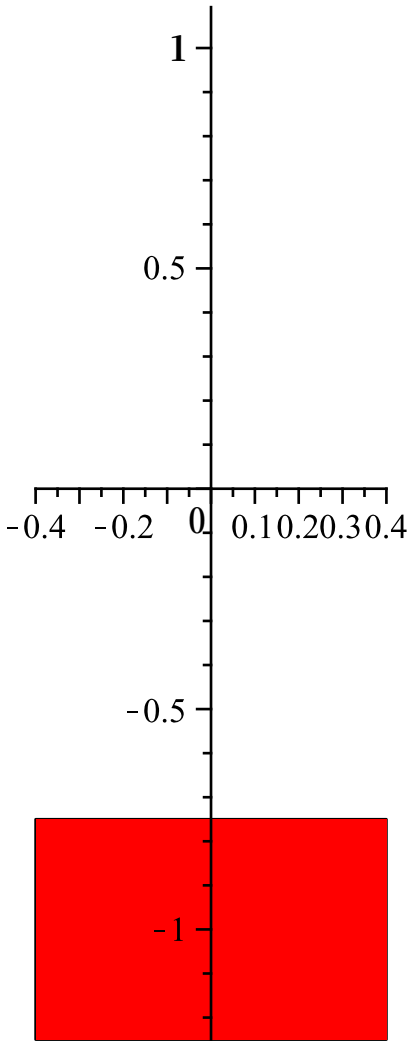


（参考１）アニメーションの操作は，以下の操作パネルから行うことができます．操作パネルは，プロットされたグラフをクリックするとツールバーの下に表示されます．

（参考２）アニメーションの生成

```
> # ------------------------------------------------------
  # アニメーション時間の定義
  # ------------------------------------------------------
  endT := 10:
  N    := 100:
  dT   := endT/N:
  T    := [ seq( evalf( k*dT ), k=0..N ) ]:

  # ------------------------------------------------------
  # 物体形状の定義
  # ------------------------------------------------------
  s := 2:
  ( x1, y1 ) := -0.8/s, -0.5/s:
  ( x2, y2 ) :=  0.8/s,  0.5/s:

  # ------------------------------------------------------
  # アニメーションの生成
  # ------------------------------------------------------
```

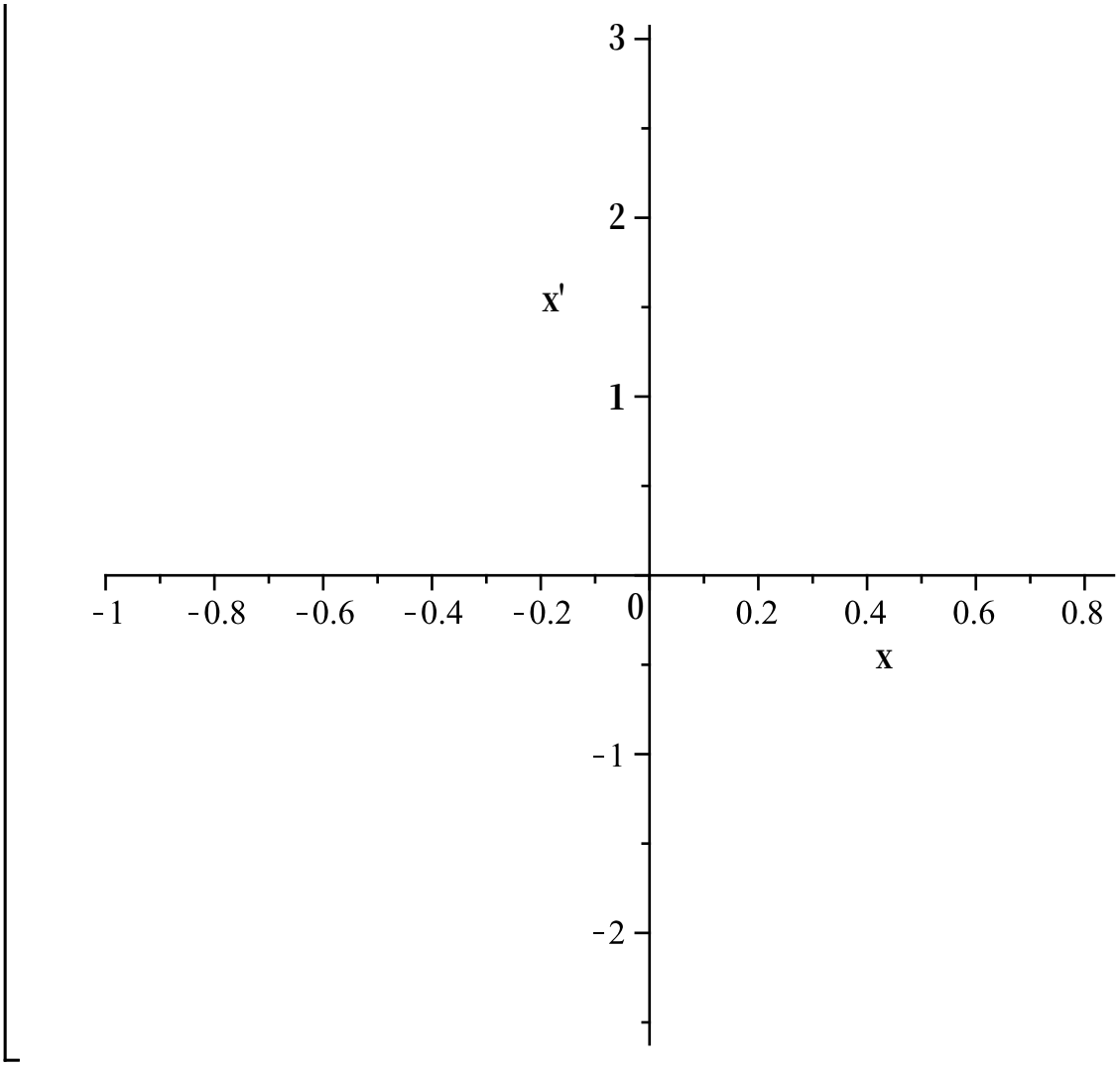
```
for i from 1 to N+1 do
  d[i] := rhs( ( dsol( T[i] )[] )[2] ):
  r[i] := plots[display](
    plottools[rectangle]( [ x1, y1+d[i] ],
                          [ x2, y2+d[i] ],
                          color=red ) ): # 赤（red）に変更
end do:
req := [ seq( r[i], i=1..N+1 ) ]:
plots[display]( req, insequence=true, scaling=constrained );
```



【参考】位相平面のプロット

縦軸，横軸にそれぞれ位置$x(t)$，速度 $\frac{d}{dt}x(t)$ を取ったときのアニメーションを作成します． このプロットは，位相面軌道あるいは位相平面と呼ばれ，システム（微分方程式系）の動的な性質を解析するために用いられます．

```
> plots[odeplot]( dsol, [ x(t), diff( x(t), t ) ], 0..10, numpoints=1000,
  frames=24 );
```



（参考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．

（参考２） システム（微分方程式系）の位相平面を描画するコマンドとして，DEtools パッケージに DEplot コマンドが用意されています．詳細はオンラインヘルプを参照してください．

```
> ?DEplot
```

ステップ・バイ・ステップ式 **はじめての Maple** 応用編

プログラミング前夜

Maple プログラミング言語の基礎を習得します.

目次

処理の再利用

プロシージャ化（処理を１行で記述可能な場合）

（基本構文）

プロシージャ名 := (変数) -> コマンド（変数を含む処理を記述）;

•変数はプロシージャの引数になります.
•-> （アロー演算子）を使用します.

初期化します.

```
> restart:
```

引数がひとつ（x）のプロシージャを定義します.

```
> myproc1 := x -> x^2;
```

$$myproc1 := x \mapsto x^2 \tag{1}$$

プロシージャに値を与えて実行します．入力方法，実行方法は通常のコマンドと同じです.

```
> myproc1(2);
```

$$4 \tag{2}$$

変数（引数）がふたつ（x, y）のプロシージャを定義します.

```
> myproc2 := ( x, y ) -> x^2 + y^2;
```

$$myproc2 := (x, y) \mapsto x^2 + y^2 \tag{3}$$

```
> myproc2( 3, 4 );
```

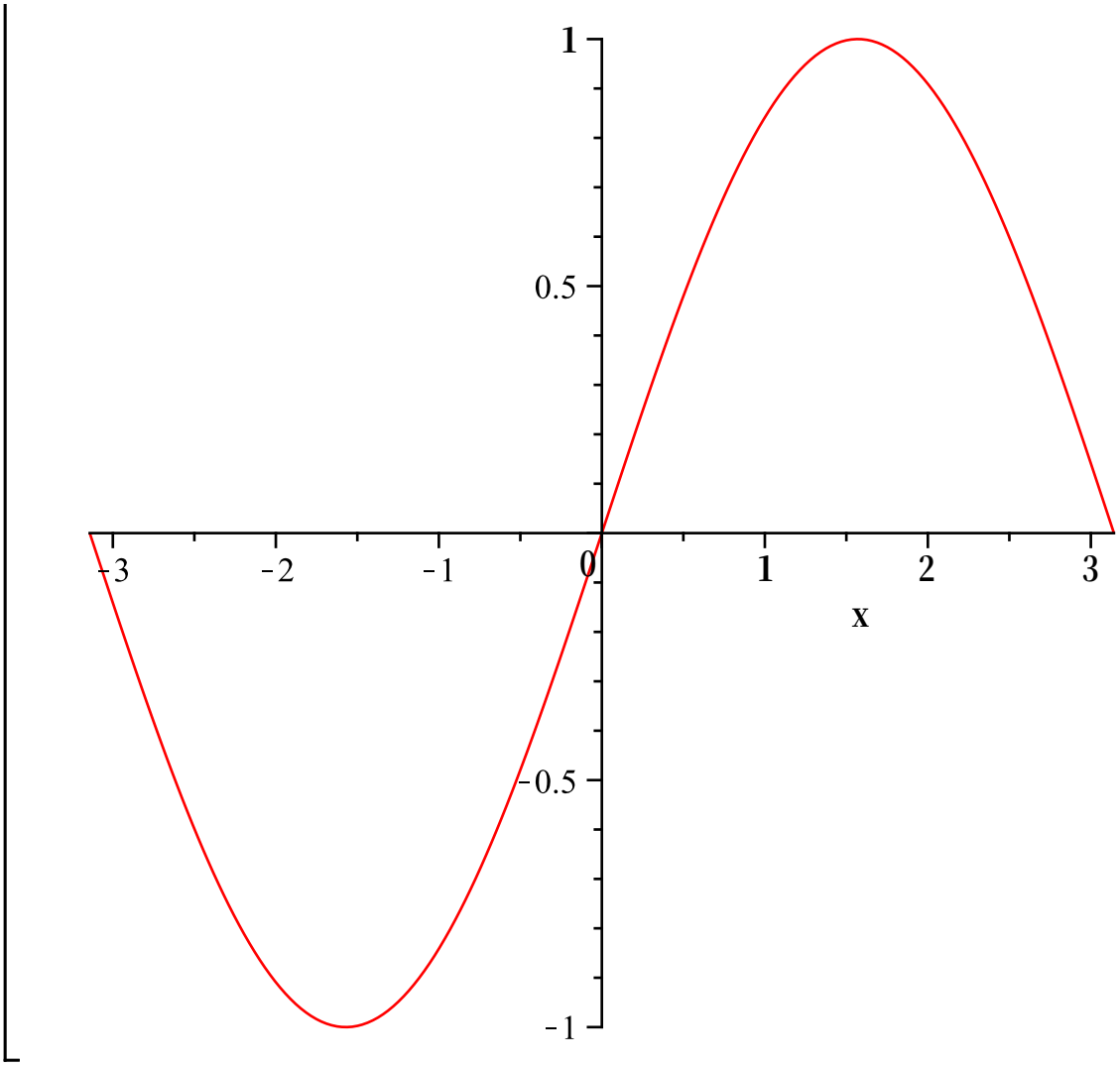
$$25 \tag{4}$$

変数（引数）なしのプロシージャになります.

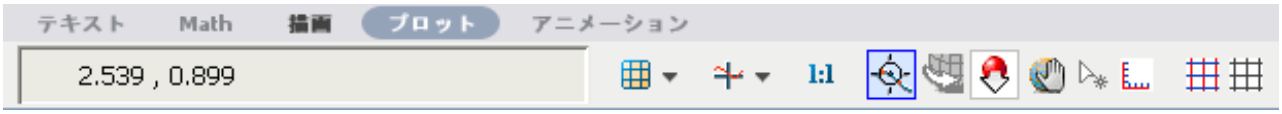
```
> myproc3 := ( ) -> plot( sin(x), x=-Pi..Pi );
```

$$myproc3 := ( ) \mapsto plot(\sin(x), x = -\pi..\pi) \tag{5}$$

```
> myproc3();
```



（参考）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．



以降，１行で記述可能なプロシージャを便宜上「**ファンクション**」を呼ぶことにします．これは，次のプロシージャと区別するためになります．

プロシージャ化（処理を複数行で記述する場合）

（基本構文）

```
プロシージャ名 := proc( 引数 )
    処理;
end proc;
```

・一般的に，処理は複数行で記述します．

初期化します．

```
> restart;
```

引数なしのプロシージャを定義します．

```
> myproc1 := proc()
```

```
    print( "Hello World!" );
end proc;
```

$$myproc1 := \mathbf{proc}()\ print(\text{"Hello World!"})\ \mathbf{end\ proc} \tag{6}$$

（参考）ファンクションによる定義

```
> myfunc1 := ( ) -> print( "Hello World!" );
```

$$myfunc1 := () \mapsto print(\text{"Hello World!"}) \tag{7}$$

プロシージャを実行してみます.

```
> myproc1;
```

$$\text{myproc1} \tag{8}$$

ただし，プロシージャ名が表示されるだけです．プロシージャを実行するためには（ ）が必要になります.

```
> myproc1();
```

$$\text{"Hello World!"} \tag{9}$$

（参考）ファクション myfunc1 の実行

```
> myfunc1();
```

$$\text{"Hello World!"} \tag{10}$$

プロシージャを 1 行で定義することも可能です.

```
> myproc2 := proc() printf( "Hello World!" ); end proc;
```

$$myproc2 := \mathbf{proc}()\ printf(\text{"Hello World!"})\ \mathbf{end\ proc} \tag{11}$$

プロシージャを実行します.

```
> myproc2();
Hello World!
```

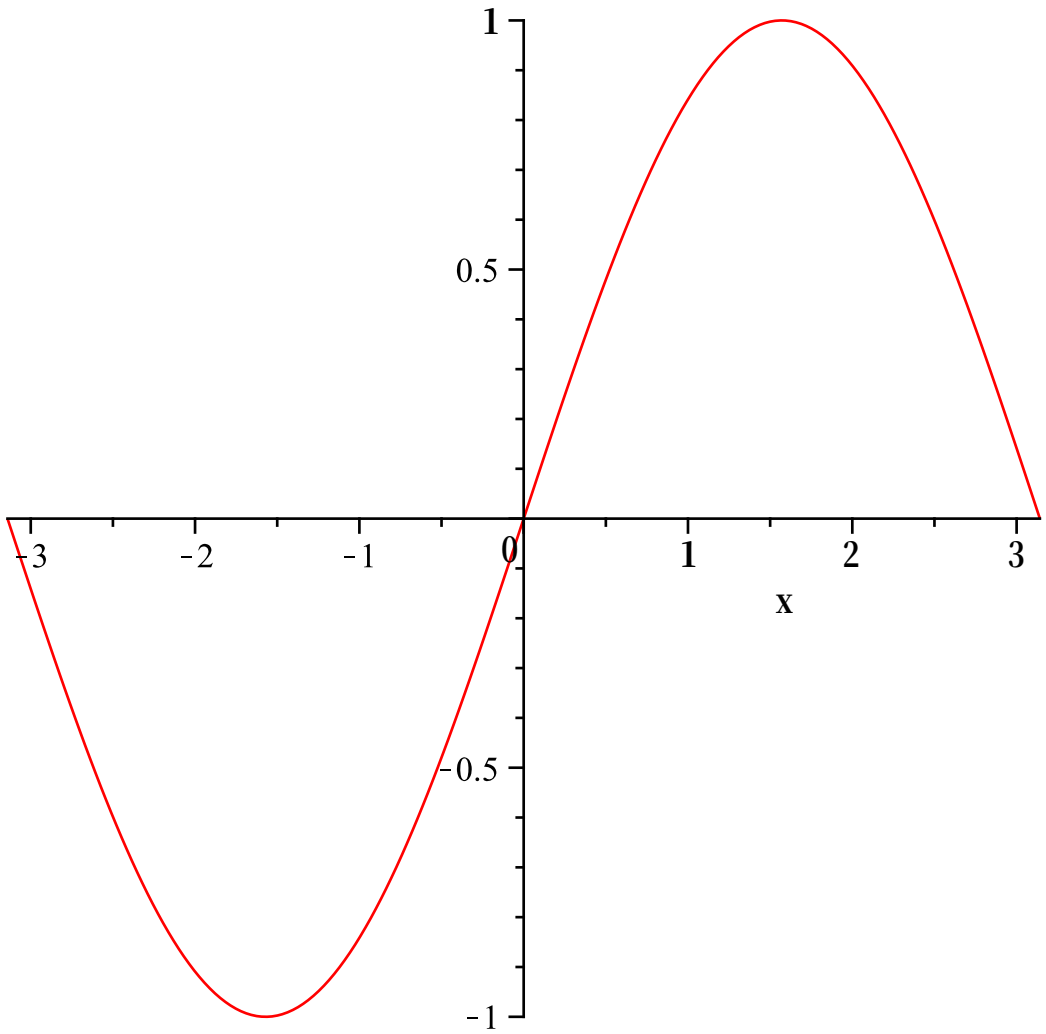
ある範囲（区間）で $\sin(x)$ をプロットするプロシージャ

```
> myproc3 := proc( a, b )
    plot( sin(x), x=a..b );
end proc;
```

$$myproc3 := \mathbf{proc}(a, b)\ plot(\sin(x), x = a..b)\ \mathbf{end\ proc} \tag{12}$$

-Pi..Piの範囲でプロットします.

```
> myproc3( -Pi, Pi );
```

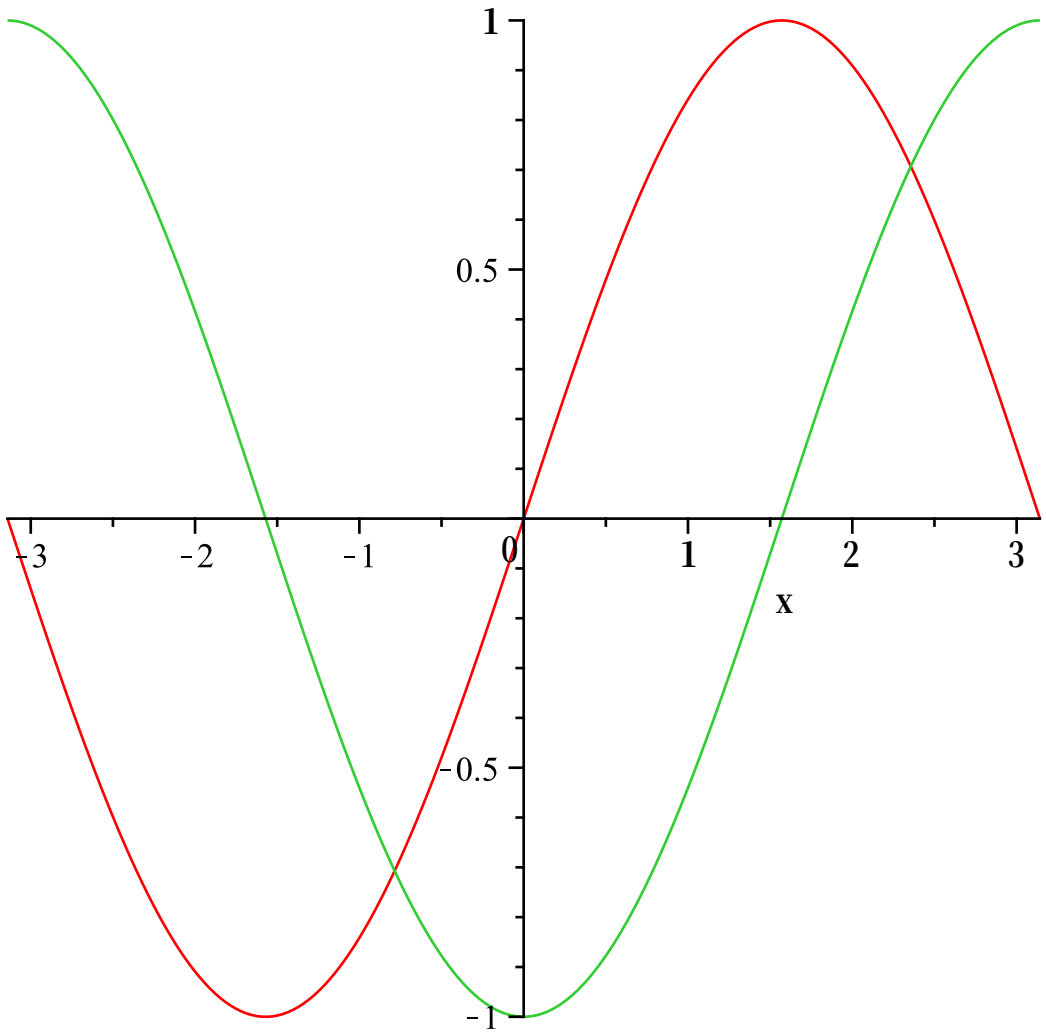


ある関数（$\sin(x)$）を積分し，その結果（$-\cos(x)$）とともに，もとの（被積分）関数（$\sin(x)$）を同時にプロットするプロシージャを定義します.

```
> myproc4 := proc( f, a, b )

    # f と int( f, x ) をプロット
    plot( [ f, int( f, x ) ], x=a..b );

  end proc:
```

end proc:のように，最後をコロン（:）で閉じると，他の処理と同じように結果（プロシージャの内容）が表示されません.

```
> myproc4( sin(x), -Pi, Pi );
```

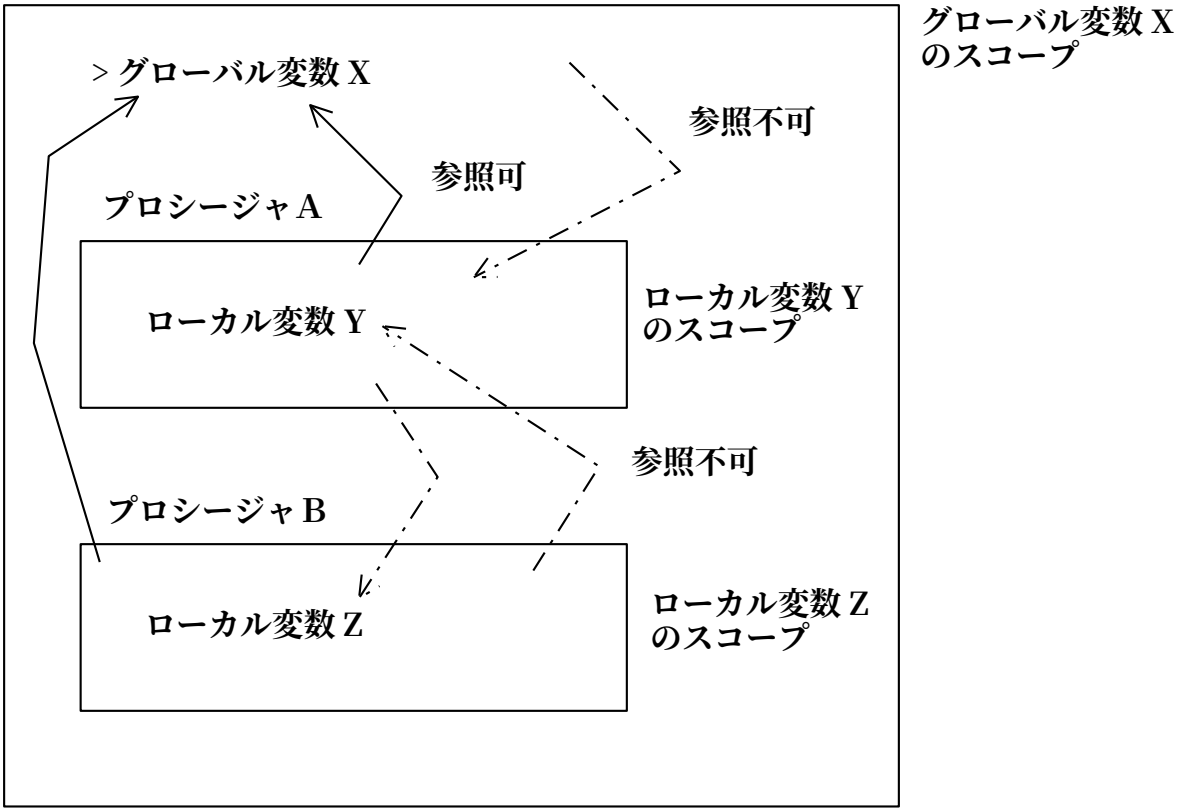


グローバル変数とローカル変数

一般的に，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（大域的に参照可能な変数）になり，ローカル変数は，そのプロシージャ内でのみ参照可能な変数（局所的に参照可能な変数）になります．

処理の内容によって，グローバル変数とローカル変数を使い分けることになりますが，通常［プロシージャの作成において］グローバル変数の使用はできるだけ避けるようにします．これは，グローバル変数に格納されている値や式が，全体の処理の中で意図せず変更される可能性を防ぐためです．

Maple のワークシート内



初期化します.

```
> restart;
```

（例）グローバル変数とローカル変数の違い

プロシージャ名を myproc5 として，以下のプロシージャを定義します.

はじめに，変数 X に 5 を割り当てます（格納します）．この X は，グローバル変数になります．つまり，スコープ（参照の有効範囲）はワークシート内すべてになります.

```
> X := 5;
```

$$X := 5 \tag{13}$$

Maple のワークシート内

```
> X := 5;
```

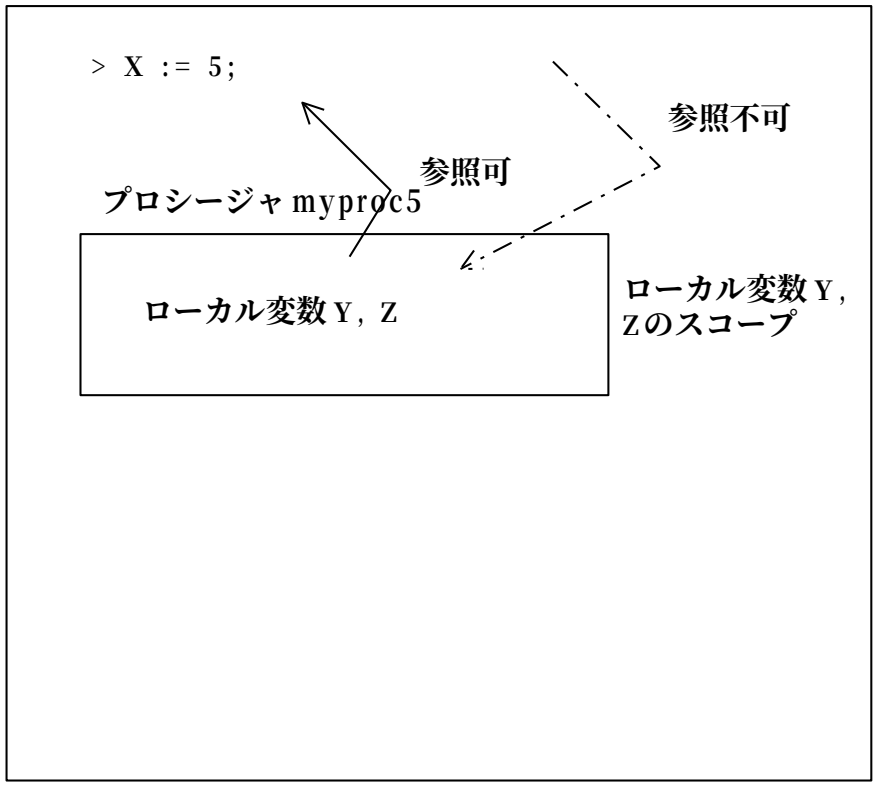
グローバル変数 X のスコープ

（参考）コメントの追加
プロシージャの先頭に，コメントとして処理の説明を記述します．シャープ（＃）の後ろは，Maple の処理には含まれません（コメントとして無視されます）．

（参考）ローカル変数の定義
ローカル変数は，プロシージャ内の先頭において，local を用いて定義します（あるいは，宣言します）．複数定義する場合は，コンマ（，）で変数を区切ります．

（例） `local a, b, c, d, e, f, abc, xyz;`

```
> # ---------------------------------
  # X, Y, Z の内容を表示するプロシージャ
  # ---------------------------------
  myproc5 := proc()

     # ローカル変数の宣言
     local Y, Z;

     # 値の代入
     Y := 6;
     Z := 15;

     # 変数の内容を表示
     print( X, Y, Z );

  end proc: # myproc5
```



YおよびZは，ローカル変数になります．スコープ（参照の有効範囲）は，myproc5 プロシージャ内だけで参照可能となります．例えば，Yを表示してます．

```
> Y;
```

$$Y \tag{14}$$

変数Yには，何も格納されていません．

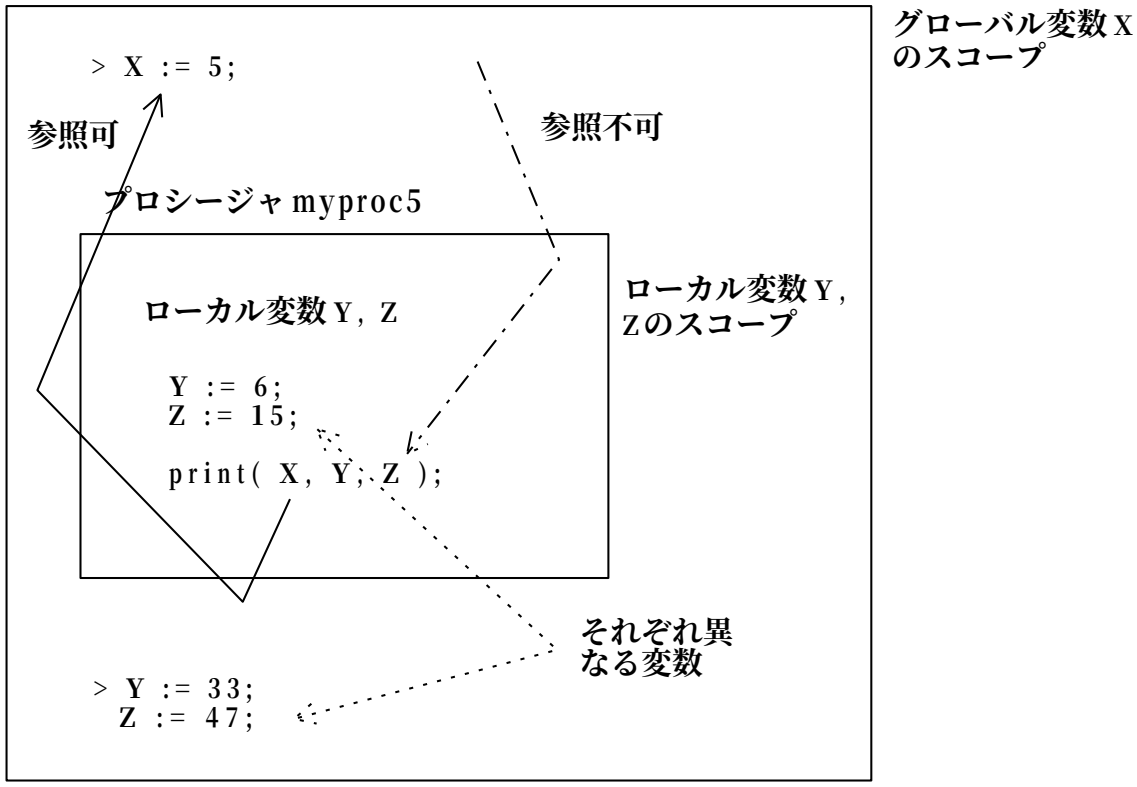
ここで，YとZにそれぞれ33と47を割り当てて，myproc5を実行してみます．

```
> Y := 33;
  Z := 47;
```

$$Y := 33$$
$$Z := 47 \tag{15}$$

```
> myproc5();
```

$$5, 6, 15 \tag{16}$$



プロシージャ内の Y と Z には，何も影響しません．

引数の型・変数の型

プロシージャ，引数，およびローカル変数の型を明示的に与えることができます．

（基本構文）

```
プロシージャ名 := proc( 引数: :型 ) : :型
    local 変数: :型
    処理;
end proc;
```

２つの整数値（integer）を足し合わせ，計算結果も整数値（integer）とします．

```
> # ------------------------------
  # 整数値の足し算
  # ------------------------------
  myAdd := proc( a::integer, b::integer )::integer;

     # ローカル変数の宣言
    local sum;

     # 引数 a と b の足し算
    sum := a + b;

  end proc: # myAdd
```

1 と 2 の足し算を実行し，変数 p に割り当てます（格納します）．

```
> p := myAdd(1, 2);
```

$$p := 3 \tag{17}$$

変数 p の型を確認します.

```
> whattype( p );
```

integer (18)

数値 1.0 （浮動小数点型＝float） と整数値 2 （整数型＝integer） の足し算を実行します.

```
> p := myAdd( 1.0, 2 );
```

Error, invalid input: myAdd expects its 1st argument, a, to be of type integer, but received 1.0

エラーメッセージが出力されます. 第 1 番目の引数（argument） = 1.0 が数値（浮動小数点型）のため，整数型と一致しません.

制御構文

処理の流れを制御する構文です.

条件分岐（if-then-else-end if）

条件（式）によって処理が分岐します.

（基本構文）

```
if 条件式1 then
    処理1;
elif 条件式2 then
    処理2;
else
    処理3;
end if;
```

初期化します.

```
> restart;
```

数字の大小を判別する条件分岐

数字の大小を判別する条件分岐を作成します.

はじめに，a および b に適当な（整数）値を代入します.

```
> a := 1;
```

$$a := 1 \tag{19}$$

```
> b := 2;
```

$$b := 2 \tag{20}$$

条件式 a ＞ b の評価結果に応じて出力されるメッセージが変わります.

```
> if a > b then
    print( "a のほうが大きい" );
  else
    print( "b のほうが大きい" );
  end if;
```

"b のほうが大きい" (21)

条件式 a ＞ b を evalb コマンドで評価してみます.

```
> evalb( a > b );
```

false (22)

条件式を a ＜ b として，evalb コマンドで評価してみます.

```
> evalb( a < b );
```

true (23)

次に，a と b が等しい場合の処理を追加します．この時点で，定義されている値は a=1 および b=2 にな

ります．つまり，a ＜ b（1 ＜ 2）の関係にあります．

```
> if a > b then
    print( "a のほうが大きい" );
  elif a = b then
    print( "a と b は等しい" );
  else
    print( "b のほうが大きい" );
  end if;
```

"b のほうが大きい"　(24)

ここで，a と b を等しい値で定義しなおします．

```
> a := 3;
```

$a := 3$　(25)

```
> b := 3;
```

$b := 3$　(26)

```
> if a > b then
    print( "a のほうが大きい" );
  elif a = b then
    print( "a と b は等しい" );
  else
    print( "b のほうが大きい" );
  end if;
```

"a と b は等しい"　(27)

条件分岐を含むプロシージャの作成手順

上記の処理を引数 a，b としてプロシージャにまとめます．すなわち，再利用可能な処理にまとめます．ここでは，以下の手順でまとめることにします．

□ プロシージャ名を myproc6 として，proc 構造を用意します．また，引数を a，b とします，

```
> myproc6 := proc( a, b )


  end proc:
```

□ 上の大小を評価してメッセージを出力する if 文を挿入します．

```
> myproc6 := proc( a, b )
  if a > b then
    print( "a のほうが大きい" );
  elif a = b then
    print( "a と b は等しい" );
  else
    print( "b のほうが大きい" );
```

```
   end if:

   end proc:
```

□ i f から e n d i f ; までの行を右方向に下げます（インデントします）.

（参考）通常，その処理が，上位の処理にネストされているとき（入れ子になっているとき），インデントを設けます．２文字文から４文字程度が一般的です．ここでは，インデントを３文字文にしています．

```
> myproc6 := proc(a, b)
  ###
     if a > b then
      ###
        print( "a のほうが大きい" );
  ###
     elif a = b then
      ###
        print( "a と b は等しい" );
  ###
     else
      ###
        print( "b のほうが大きい" );
  ###
     end if;

   end proc:
```

□ 説明やコメント行（#）を追加します.

（注意）###は取り除きます．###はインデントを示すために挿入しました．通常は必要ありません.

```
> # ---------------------------------
  # ２変数に格納されている値の大小を比較
  # ---------------------------------
  myproc6 := proc(a, b)

     # a > b の場合
     if a > b then

        # 以下のメッセージを表示
        print( "a のほうが大きい" );

     # a = b の場合
     elif a = b then

        # 以下のメッセージを表示
        print( "a と b は等しい" );

     # ２つの条件式から外れた場合の処理
     else

        # 以下のメッセージを表示
        print( "b のほうが大きい" );

     end if;

  end proc: # myproc6
```

a と b を a > b として定義しなおします.

```
> a := 5;
```

$$a := 5 \tag{28}$$

```
> b := 3;
```

$$b := 3 \tag{29}$$

myproc6 に値を与えます.

```
> myproc6( a, b );
```

"a のほうが大きい"　(30)

引数の変数名は a，b 以外でも使用できます．

```
> X := 30;
  Y := -20;
```

$$X := 30$$
$$Y := -20$$　(31)

同様に myproc6 に値を与えます．

```
> myproc6( X, Y );
```

"a のほうが大きい"　(32)

（参考）引数を比較するプロシージャに変更

１番目の引数と２番目の引数を比較するプロシージャであると解釈しなおします．

表示内容を書き換えます．

```
> # ---------------------------------
  # ２変数に格納されている値の大小を比較
  # ---------------------------------
  myproc6 := proc( a, b )

     # a > b の場合
     if a > b then

        # 以下のメッセージを表示
        print( "引数１のほうが大きい" );

     # a = b の場合
     elif a = b then

        # 以下のメッセージを表示
        print( "引数１と引数２は等しい" );

     # ２つの条件式から外れた場合の処理
     else

        # 以下のメッセージを表示
        print( "引数２のほうが大きい" );

     end if;

  end proc: # myproc6
```

（少しだけ）一般化されたプロシージャになります．

```
> myproc6( X, Y );
```

"引数１のほうが大きい"　(33)

繰り返し（for-do-end do）

（基本構文）

```
for インデックス名 from 開始数 by インクリメント数 to 終了数 do
   処理（インデックス名を含む）;
end do;
```

初期化します．

```
> restart;
```

足し算の実行

0, 1, 2, 3, 4 の足し算を繰り返し文を用いて実行します.

繰り返し文の前に

はじめに，ここではひとつの変数 total を使用して足し算を行い，はじめの値を 0 とします．この 0 を初期値と呼びます.

```
> total := 0; # はじめの値を０とします
```

$$total := 0 \tag{34}$$

```
> total := total + 1;
  # total := ( 0 ) + 1; <-- 実際の計算
```

$$total := 1 \tag{35}$$

```
> total := total + 2;
  # total := ( ( 0 ) + 1 ) + 2; <-- 実際の計算
```

$$total := 3 \tag{36}$$

```
> total := total + 3;
  # total := ( ( ( 0 ) + 1 ) + 2 ) + 3; <-- 実際の計算
```

$$total := 6 \tag{37}$$

```
> total := total + 4;
  # total := ( ( ( ( 0 ) + 1 ) + 2 ) + 3 ) + 4; <-- 実際の計算
```

$$total := 10 \tag{38}$$

実際の足し算は以下のような記述で簡単に実行されます.

```
> 0 + 1 + 2 + 3 + 4;
```

$$10 \tag{39}$$

ここで，インデックス i を導入します．とくに，i である必要はありませんが，一般的に 1 次元のときは i を使用します.

（参考）また，２次元のときは i と j を，さらに，３次元のときは，i，j，および k を使用します.

```
> total := 0; # はじめの値を０とします

  i := 1;
  total :=      total   +   i;
# total :=      ( 0 )   +   1;

  i := 2;
  total :=      total       +   i;
# total :=    ( ( 0 ) + 1 )   +   2;

  i := 3;
  total :=      total           +   i;
# total :=   ( ( ( 0 ) + 1 ) + 2 )   +   3;

  i := 4;
  total :=      total               +   i;
# total := ( ( ( ( 0 ) + 1 ) + 2 ) + 3 )   +   4;
```

$$\begin{aligned} total &:= 0 \\ i &:= 1 \\ total &:= 1 \\ i &:= 2 \\ total &:= 3 \\ i &:= 3 \\ total &:= 6 \end{aligned}$$

$$i := 4$$
$$total := 10 \tag{40}$$

ここで，いったい何がなされたのでしょうか？？？

実は，

```
total := total + 1
total := total + 2
total := total + 3
total := total + 4
```

をインデックスi を用いて

```
total := total + i
```

と式を**一般化**しました．

（参考）コメントを除いて整理した処理になります．

```
> total := 0;

  i := 1;
  total := total + i;

  i := 2;
  total := total + i;

  i := 3;
  total := total + i;

  i := 4;
  total := total + i;
```

$$total := 0$$
$$i := 1$$
$$total := 1$$
$$i := 2$$
$$total := 3$$
$$i := 3$$
$$total := 6$$
$$i := 4$$
$$total := 10 \tag{41}$$

繰り返し文の利用

i が 1 から 4 まで 1 ずつおおきくなります（これを，**インクリメント**，と表現します）．Maple の処理で記述すると以下のようになります．

```
i from 1 to 4 by 1
```

もしくは

```
i from 1 by 1 to 4
```

さらに，Maple の繰り返し文（for-do-end do文）を追加します．

```
for i from 1 to 4 by 1 do

end do;
```

ここで，i の変化を見ます（i に格納されている値を表示するだけの処理です）．

print コマンドは，表示処理を実行します．

```
> for i from 1 to 4 by 1 do
    print( 'i' = i );
  end do;
```

$$i = 1$$
$$i = 2$$
$$i = 3$$
$$i = 4$$ (42)

（参考）'i' は文字そのものを表します．

以下の処理を繰り返し文（ for-do-end do文）を用いて整理します．

```
total := 0;
```

（ここからfor文が始まり・・・）

```
i := 1;
total := total + i;

i := 2;
total := total + i;

i := 3;
total := total + i;

i := 4;
total := total + i;
```

（・・・ここでfor文が終わります）

すなわち，

for - doとend do;の間に

```
total := total + i
```

を挿入し，処理を整理します．なお，totalのはじめの値（初期値）は0になります．

```
> # 初期値の設定
  total := 0;

  for i from 1 to 4 by 1 do

    # i を参照するために残してあります
    print( 'i' = i );

    total := total + i;

  end do;
```

$$total := 0$$
$$i = 1$$
$$total := 1$$
$$i = 2$$
$$total := 3$$
$$i = 3$$
$$total := 6$$
$$i = 4$$

$$total := 10 \tag{43}$$

i が 1 ずつインクリメントされる場合，by 1 を省略できます．

```
> # 初期値の設定
  total := 0;

  for i from 1 to 4 do

     # i を参照するために残してあります
     print( 'i' = i );

     total := total + i;

  end do;
```

$$\begin{aligned} total &:= 0 \\ i &= 1 \\ total &:= 1 \\ i &= 2 \\ total &:= 3 \\ i &= 3 \\ total &:= 6 \\ i &= 4 \\ total &:= 10 \end{aligned} \tag{44}$$

プロシージャへの拡張

ここで，0, 1, 2, 3, ..., n までを足し合わせるプロシージャを作成します．

```
> # ---------------------------------
  # 0, 1, 2, 3, ..., n までの足し算
  #
  # 引数： 正の整数（n）
  #
  # ---------------------------------
  mySum := proc( n::posint )

     # ローカル変数の定義
     local i, total;

     # 初期値の設定
     total := 0;

     # 0 から n までの加算
     for i from 1 to n do

        # 加算
        total := total + i;

     end do;

  end proc; # mySum
```

$$\begin{aligned} &mySum := \mathbf{proc}(n::posint) \\ &\quad \mathbf{local}\ i, total; \\ &\quad total := 0;\ \mathbf{for}\ i\ \mathbf{to}\ n\ \mathbf{do}\ total := total + i\ \mathbf{end\ do} \\ &\mathbf{end\ proc} \end{aligned} \tag{45}$$

n = 4 として，mySum を実行します．

```
> mySum( 4 );
```

$$10 \tag{46}$$

n = 10として，mySum を実行します．

```
> mySum( 10 );
```

$$55 \tag{47}$$

（参考）sum コマンドを用いて実行します

数式入力 $\sum_{k=0}^{4} k$ をテキスト入力で実行します．

```
> sum( k, k=0..4 );
```

$$10 \tag{48}$$

数式入力 $\sum_{k=0}^{10} k$ をテキスト入力で実行します．

```
> sum( k, k=0..10 );
```

$$55 \tag{49}$$

sum コマンドを用いて，第 n 項を計算します．

```
> sum( k, k=0..n );
```

$$\frac{(n+1)^2}{2} - \frac{n}{2} - \frac{1}{2} \tag{50}$$

確認のため，n=10 を代入してみます．

```
> eval( (50), [ n=10 ] );
```

$$55 \tag{51}$$

（参考）引数に式を用いた場合

```
> sum( k^2, k = 0..n );
```

$$\frac{(n+1)^3}{3} - \frac{(n+1)^2}{2} + \frac{n}{6} + \frac{1}{6} \tag{52}$$

```
> sum( a[k]*x^k, k = 0..4 );
```

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \tag{53}$$

（参考）2つのインデックス操作

3×5 の行列 A を記号で定義します．

```
> A := Matrix( 3, 5, symbol=a );
```

$$A := \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \end{bmatrix} \tag{54}$$

各要素をインデックスといっしょにひとつずつ表示します．このケースでは i が行のインデックス，j が列のインデックスになります．

```
> for i from 1 to 3 do
    for j from 1 to 5 do
```

```
        # インデックス i, j,
        # 行列要素 a[i, j] の順番で表示
        print( i, j, A[i,j] );

    end do: # j

end do: # i
```

$$1, 1, a_{1, 1}$$
$$1, 2, a_{1, 2}$$
$$1, 3, a_{1, 3}$$
$$1, 4, a_{1, 4}$$
$$1, 5, a_{1, 5}$$
$$2, 1, a_{2, 1}$$
$$2, 2, a_{2, 2}$$
$$2, 3, a_{2, 3}$$
$$2, 4, a_{2, 4}$$
$$2, 5, a_{2, 5}$$
$$3, 1, a_{3, 1}$$
$$3, 2, a_{3, 2}$$
$$3, 3, a_{3, 3}$$
$$3, 4, a_{3, 4}$$
$$3, 5, a_{3, 5} \tag{55}$$

それは振動しますか？

２階常微分方程式の係数および初期条件を変化させることによって，解の挙動を調べます．

初期化します．

```
> restart;
```

Maple による２階常微分方程式の解法と解のプロット

以下は，２階常微分方程式を定義するところから，必要な値や条件を定義して，解をプロットするところまでの処理になります．

２階常微分方程式を状態方程式に変換して定義します．

状態方程式とは，行列表現に変換した１階の常微分方程式（Ordinary Differential Equation，ODE）系になります．

（参考）「微分方程式に挑む！」の［２階常微分方程式の行列表現］を参照してください．

```
> ode := m*diff( x(t), t,t ) + c*diff( x(t), t ) + k*x(t) = 0;
```

$$ode := m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0 \tag{56}$$

ODE を行列表現にする過程で以下の１階の常微分方程式が導かれます．

```
> deq1 := diff( x(t), t ) = v(t);
```

$$deq1 := \dot{x}(t) = v(t) \tag{57}$$

```
> deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);
```

$$deq2 := \dot{v}(t) = -\frac{kx(t)}{m} - \frac{cv(t)}{m} \tag{58}$$

適当な係数 m，c，および k を定義します．

```
> params := [ m=1, c=1, k=6 ];
```

$$params := [m = 1, c = 1, k = 6] \tag{59}$$

定義した係数を deq1，deq2 に代入し，deq1p，deq2p をそれぞれ定義します．

```
> deq1p := eval( deq1, params );
  deq2p := eval( deq2, params );
```

$$deq1p := \dot{x}(t) = v(t)$$

$$deq2p := \dot{v}(t) = -6x(t) - v(t) \tag{60}$$

初期条件を定義します．

```
> ics := { x(0)=-1, D(x)(0)=0 };
```

$$ics := \{x(0) = -1, \mathrm{D}(x)(0) = 0\} \tag{61}$$

deq1p と deq2p を初期条件 ics として，以下のように連立して解きます．

```
> sol := dsolve( { deq1p, deq2p } union ics );
```

$$sol := \left\{ v(t) = \frac{12\,e^{-\frac{t}{2}} \sin\left(\frac{\sqrt{23}\,t}{2}\right)\sqrt{23}}{23},\ x(t) = e^{-\frac{t}{2}}\left(-\frac{\sin\left(\frac{\sqrt{23}\,t}{2}\right)\sqrt{23}}{23} \right. \right. \tag{62}$$

$$-\cos\left(\frac{\sqrt{23}\,t}{2}\right)\Bigg)\Bigg\}$$

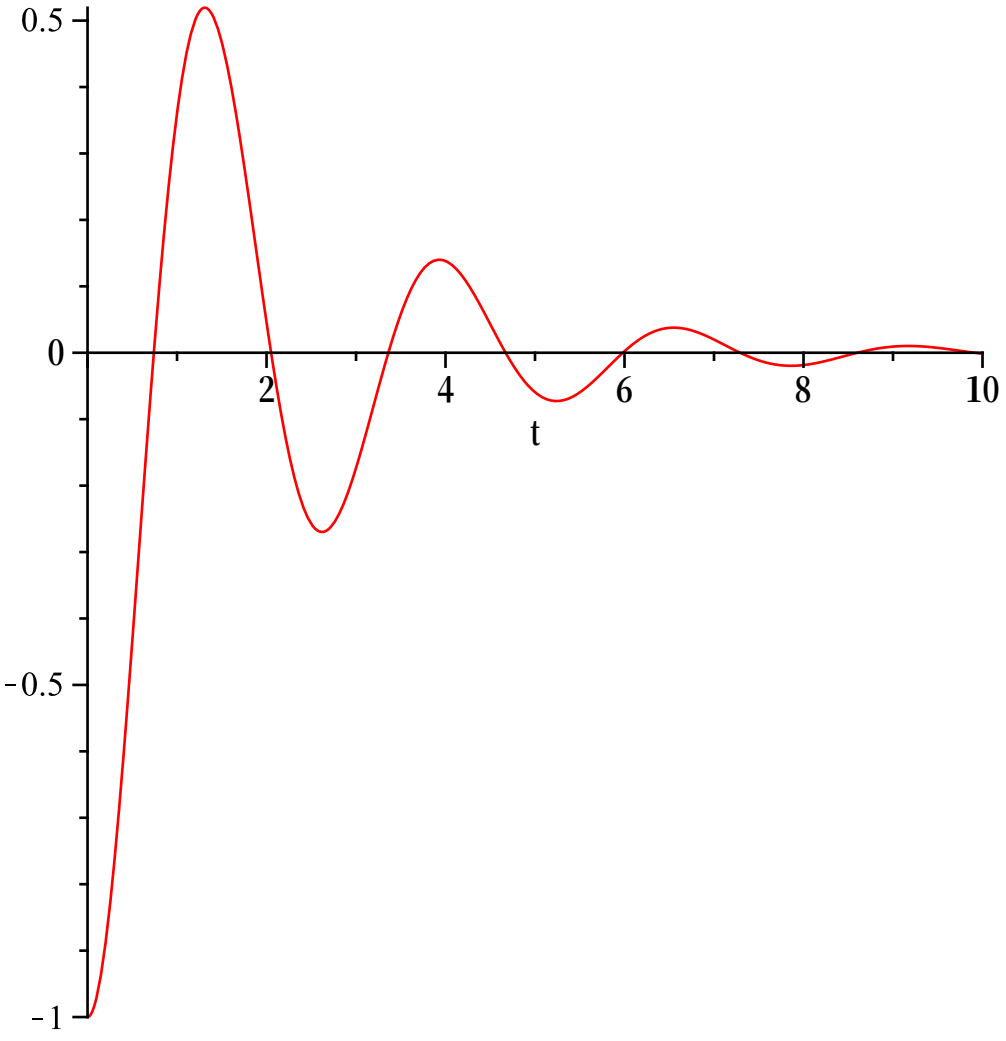
厳密解solから第２式の右辺を抽出して，eqに割り当てます．

```
> eq := rhs( sol[2] );
```

$$eq := e^{-\frac{t}{2}}\left(-\frac{\sin\left(\frac{\sqrt{23}\,t}{2}\right)\sqrt{23}}{23}-\cos\left(\frac{\sqrt{23}\,t}{2}\right)\right) \tag{63}$$

求められた解をプロットします．

```
> plot( eq, t=0..10 );
```



「それは振動しますか？」プロシージャの作成手順

２階常微分方程式の解法をプロシージャ化

以上をプロシージャとしてまとめます．プロシージャ内で使用する変数をローカル変数として宣言します．

（注意）コメントは省略しています．

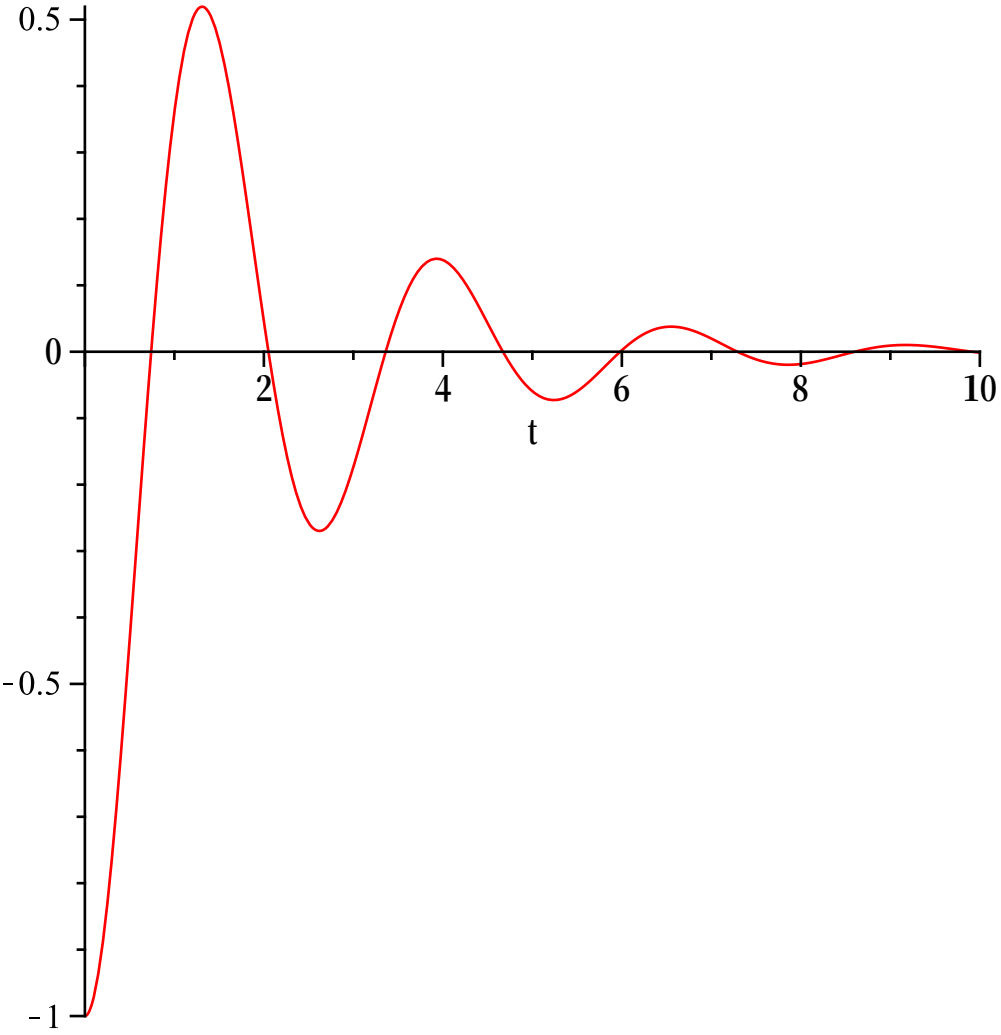
```
> mySim := proc()
      local deq1, deq2, params, deq1p, deq2p, ics, sol, eq;
      deq1 := diff( x(t), t ) = v(t);
```

```
    deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);

    params := [ m=1, c=1, k=6 ];
    deq1p := eval( deq1, params );
    deq2p := eval( deq2, params );

    ics := { x(0)=-1, D(x)(0)=0 };
    sol := dsolve( { deq1p, deq2p } union ics );

    eq := rhs( sol[2] );
    plot( eq, t=0..10 );

end proc: # mySim
```

プロシージャの出力を非表示にするために，end procの最後は，コロン（:）で閉じています．

mySimを実行します．

```
> mySim();
```



プロシージャのパラメータ化

係数m，c，k，初期条件x(0)，D(x)(0)，および時間区間0..tの値を変化させて解をプロットするようなプロシージャに拡張します．すなわち，係数と初期条件をパラメータとして，解をプロットするプロシージャに変更します．

初期化します．

```
> restart;
```

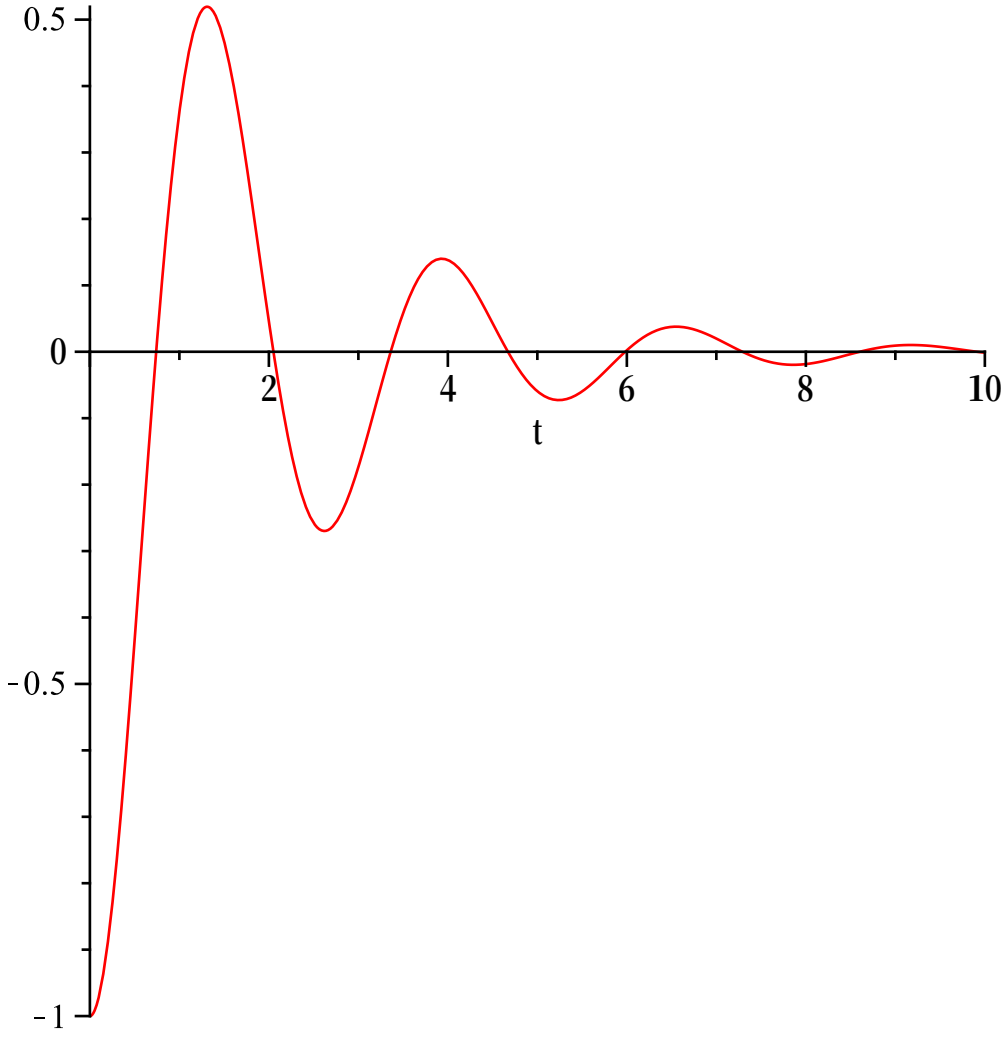
```
> mySim2 := proc( m, c, k, x0, dx0, et )

    local deq1, deq2, ics, sol, eq;
```

```
    deq1 := diff( x(t), t ) = v(t);
    deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);

    ics := { x(0)=x0, D(x)(0)=dx0 };
    sol := dsolve( { deq1, deq2 } union ics );

    eq := rhs( sol[2] );
    plot( eq, t=0..et );

end proc:# mySim2
```

mySim2に係数m, c, k, 初期条件x0, dx0, 及び終了時間e tを順番に与えて実行します.

```
m   = 1
c   = 1
k   = 6
x0  = -1   ( x(0)=-1 )
dx0 = 0    ( D(x)(0)=0 )
et  = 10
```

```
> mySim2( 1, 1, 6, -1, 0, 10 );
```



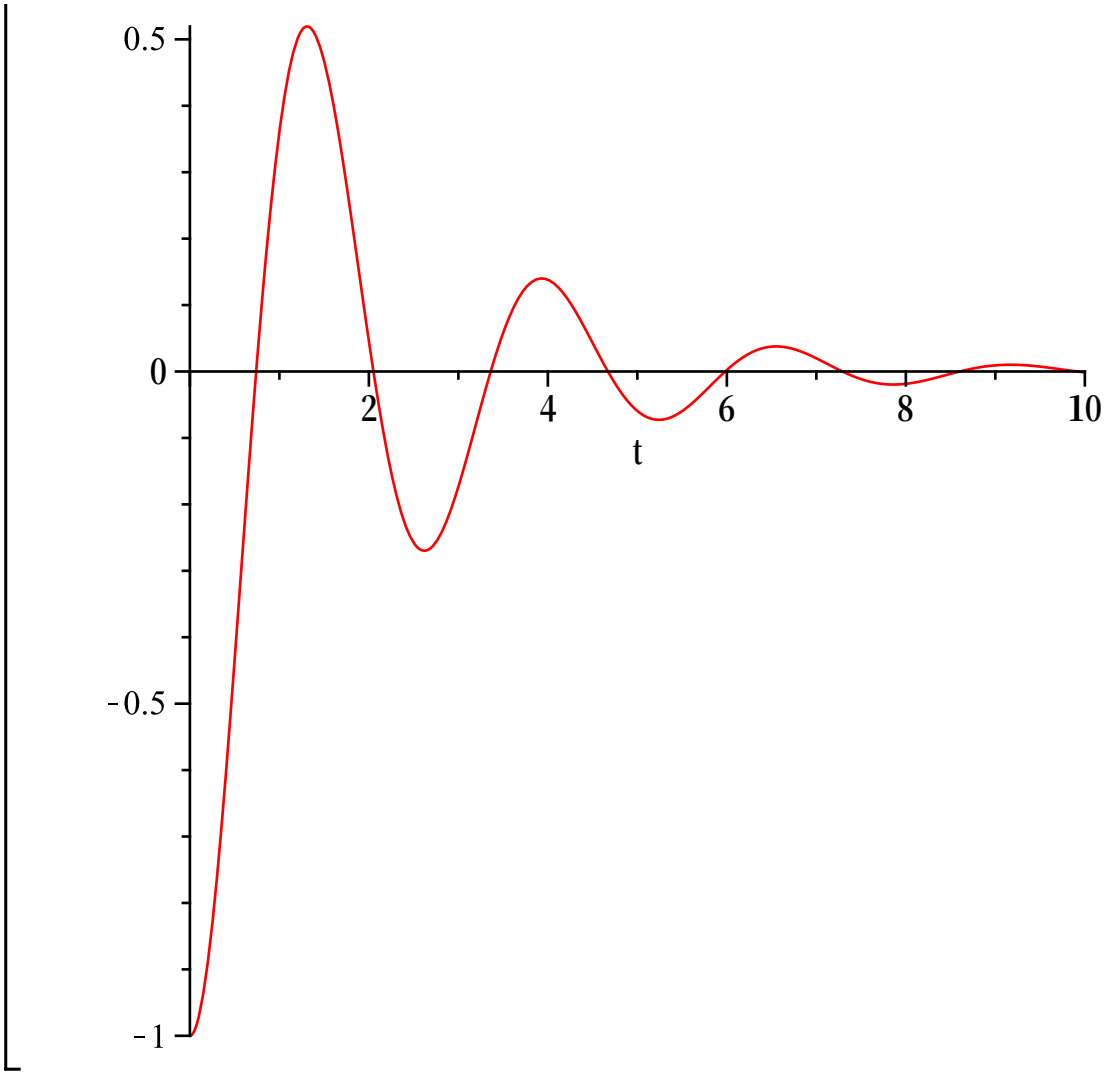
パラメータをシーケンスとして定義します.

```
> sp := 1, 1, 6, -1, 0, 10;
```

$$sp := 1, 1, 6, -1, 0, 10 \tag{64}$$

mySim2にsｐを引数として与えて実行します.

```
> mySim2( sp );
```



固有値の計算

ここで，プロシージャ内で係数行列Aの定義とその固有値を求める計算を追加します．

初期化します．

```
> restart;
```

```
> mySim3 := proc( m, c, k, x0, dx0, et )
      local A, evals, plt1, deq1, deq2, ics, sol, eq;

      A := Matrix( 2,2, [0, 1, -k/m, -c/m] );
      evals := LinearAlgebra[Eigenvalues]( A );

      deq1 := diff( x(t), t ) = v(t);
      deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);

      ics := { x(0)=x0, D(x)(0)=dx0 };
      sol := dsolve( { deq1, deq2 } union ics );

      eq := rhs( sol[2] );
      plt1 := plot( eq, t=0..et );

      return(plt1, evals);

  end proc: # mySim3
```

パラメータを定義します．

```
> sp := 1, 1, 6, -1, 0, 10;
```

$$sp := 1, 1, 6, -1, 0, 10 \tag{65}$$

mySim3をspで実行します．return(plt1, evals);で指定した通りに，プロットオブジェクトplt1と固有値evalsが出力されます．

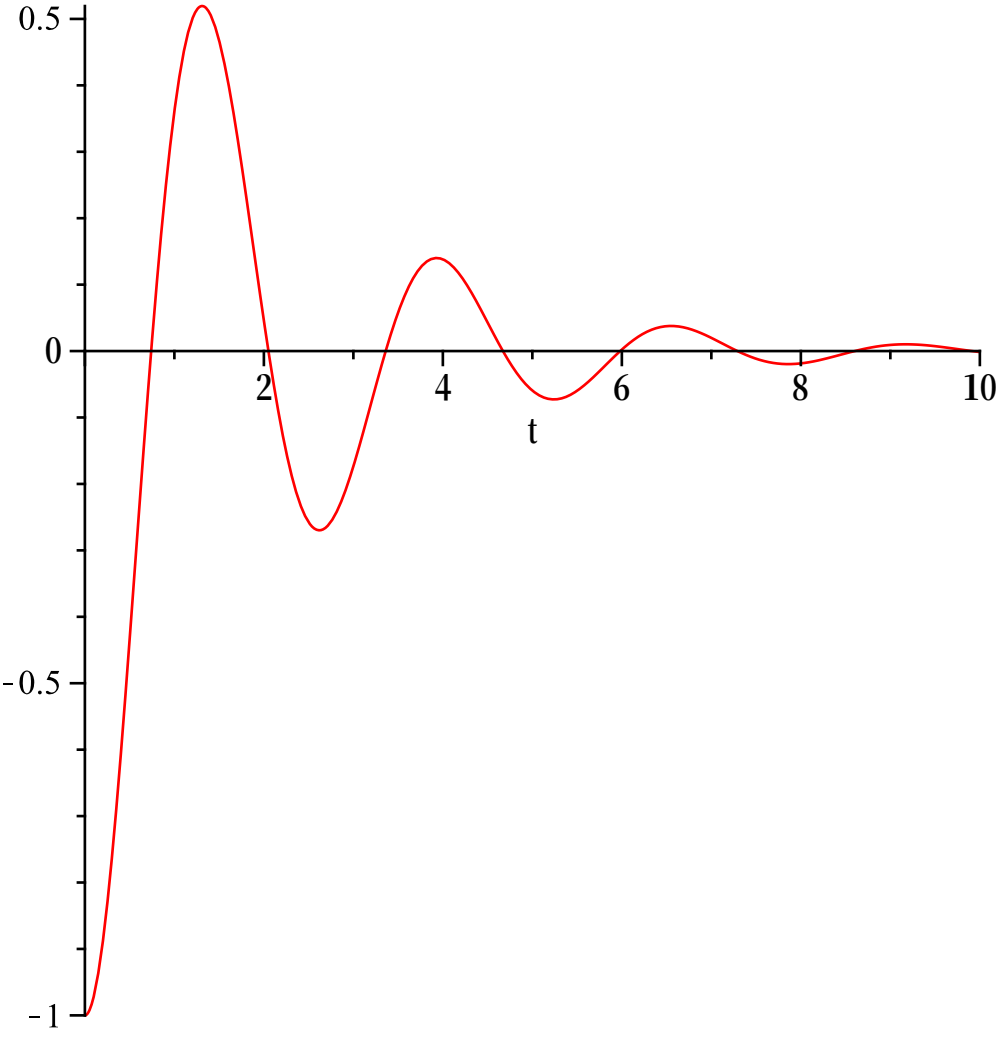
（参考）ただし，plt1およびevalsはプロシージャ内でのみ参照可能なローカル変数なので，プロシージャの外から参照することはできません．

```
> rslt := mySim3( sp );
```

$$rslt := PLOT(\ldots), \begin{bmatrix} -\frac{1}{2} + \frac{I\sqrt{23}}{2} \\ -\frac{1}{2} - \frac{I\sqrt{23}}{2} \end{bmatrix} \tag{66}$$

プロットイメージを描画します．

```
> plots[display]( rslt[1] );
```



固有値の実部・虚部抽出

改めて，固有値のみを表示します．

```
> rslt[2];
```

$$\begin{bmatrix} -\frac{1}{2} + \frac{I\sqrt{23}}{2} \\ -\frac{1}{2} - \frac{I\sqrt{23}}{2} \end{bmatrix} \tag{67}$$

固有値（特性方程式の根）から解の振る舞いが決定されます．そこで，m, c, k から固有値の性質（実

部の符号，虚部のありなし）を判別し，解の振る舞いが振動的になるのか，そうならないのかを調べることにします．

ここで，処理の流れを見やすくするために，

```
# --------------------------------
```

を小さな処理のまとまりに対して追加します．

初期化します．

```
> restart;
```

```
> mySim4 := proc( m, c, k, x0, dx0, et )

    # --------------------------------
    local r1, i1, r2, i2, ri, A, evals, plt1,
          deq1, deq2, ics, sol, eq;

    # --------------------------------
    A := Matrix( 2,2, [ 0, 1, -k/m, -c/m ] );
    evals := LinearAlgebra[Eigenvalues]( A );

    # --------------------------------
    r1 := Re( evals[1] );
    i1 := Im( evals[1] );
    r2 := Re( evals[2] );
    i2 := Im( evals[2] );
    ri := [ r1, i1, r2, i2 ];

    # --------------------------------
    deq1 := diff( x(t), t ) = v(t);
    deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);

    # --------------------------------
    ics := { x(0)=x0, D(x)(0)=dx0 };
    sol := dsolve( { deq1, deq2 } union ics );

    # --------------------------------
    eq := rhs( sol[2] );
    plt1 := plot( eq, t=0..et );

    # --------------------------------
    return( plt1, evals, ri );

end proc: # mySim4
```

パラメータを定義し，mySim4 を実行します．

```
> sp := 1, 1, 6, -1, 0, 10;
```

$$sp := 1, 1, 6, -1, 0, 10 \tag{68}$$

```
> mySim4( sp );
```

$$PLOT(\ldots), \begin{bmatrix} -\frac{1}{2} + \frac{I\sqrt{23}}{2} \\ -\frac{1}{2} - \frac{I\sqrt{23}}{2} \end{bmatrix}, \left[-\frac{1}{2}, \frac{\sqrt{23}}{2}, -\frac{1}{2}, -\frac{\sqrt{23}}{2} \right] \tag{69}$$

実部の符号と条件分岐

ここで，実部の符号から，その解の振る舞いを調べる処理（条件分岐）を追加します．

初期化します．

```
> restart;
```

```
> mySim5 := proc( m, c, k, x0, dx0, et )
```

```
    # ---------------------------------
    local r1, i1, r2, i2, ri, A, evals, plt1,
          deq1, deq2, ics, sol, eq;

    # ---------------------------------
    A := Matrix( 2,2, [ 0, 1, -k/m, -c/m ] );
    evals := LinearAlgebra[Eigenvalues]( A );

    # ---------------------------------
    r1 := Re( evals[1] );
    i1 := Im( evals[1] );
    r2 := Re( evals[2] );
    i2 := Im( evals[2] );
    ri := [ r1, i1, r2, i2 ];

    # ---------------------------------
    if r1 < 0 and r2 < 0 then

        # 収束します

    elif r1 > 0 or r2 > 0 then

        # 発散します

    end if;

    # ---------------------------------
    deq1 := diff( x(t), t ) = v(t);
    deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);

    # ---------------------------------
    ics := { x(0)=x0, D(x)(0)=dx0 };
    sol := dsolve( { deq1, deq2 } union ics );

    # ---------------------------------
    eq := rhs( sol[2] );
    plt1 := plot( eq, t=0..et );

    # ---------------------------------
    return( plt1, evals, ri );

end proc: # mySim5
```

虚部の符号と条件分岐

続けて，虚部のありなしから，その解の振る舞いを調べる処理（条件分岐）を追加します．

```
> mySim6 := proc( m, c, k, x0, dx0, et )

    # ---------------------------------
    local r1, i1, r2, i2, ri, A, evals, plt1,
          deq1, deq2, ics, sol, eq;

    # ---------------------------------
    A := Matrix( 2,2, [ 0, 1, -k/m, -c/m ] );
    evals := LinearAlgebra[Eigenvalues]( A );

    # ---------------------------------
    r1 := Re( evals[1] );
    i1 := Im( evals[1] );
    r2 := Re( evals[2] );
    i2 := Im( evals[2] );
    ri := [ r1, i1, r2, i2 ];

    # ---------------------------------
    if r1 < 0 and r2 < 0 then

      if i1 = 0 or i2 = 0 then

          # 収束しますが，振動はしません.
```

```
        else
            # 振動しながら収束します.
        end if;

    elif r1 > 0 or r2 > 0 then
        if i1 = 0 or i2 = 0 then
            # 発散し，振動はしません.
        else
            # 振動しながら発散します.
        end if;
    end if;

    # ---------------------------------
    deq1 := diff( x(t), t ) = v(t);
    deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);

    # ---------------------------------
    ics := { x(0)=x0, D(x)(0)=dx0 };
    sol := dsolve( { deq1, deq2 } union ics );

    # ---------------------------------
    eq := rhs( sol[2] );
    plt1 := plot( eq, t=0..et );

    # ---------------------------------
    return( plt1, evals, ri );
end proc: # mySim6
```

コメント部分を，文字列として出力します．evalfコマンドおよびevalbコマンドを追加します．returnコマンドは，いったん無効にします（コメントにします）．

```
> mySim6 := proc( m, c, k, x0, dx0, et )

    # ---------------------------------
    local r1, i1, r2, i2, ri, A, evals, plt1,
          deq1, deq2, ics, sol, eq;

    # ---------------------------------
    A := Matrix( 2,2, [ 0, 1, -k/m, -c/m ] );
    evals := LinearAlgebra[Eigenvalues]( A );

    # ---------------------------------
    r1 := evalf( Re( evals[1] ) );
    i1 := evalf( Im( evals[1] ) );
    r2 := evalf( Re( evals[2] ) );
    i2 := evalf( Im( evals[2] ) );
    ri := [ r1, i1, r2, i2 ];

    # ---------------------------------
    if evalb( r1 < 0 ) and evalb( r2 < 0 ) then
        if i1 = 0 or i2 = 0 then
            print( "収束しますが，振動はしません. " );
        else
            print( "振動しながら収束します. " );
        end if;
```

```
    elif evalb( r1 > 0 ) or evalb( r2 > 0 ) then
      if i1 = 0 or i2 = 0 then
          print( "発散し，振動はしません．" );
      else
          print( "振動しながら発散します．" );
      end if;
    end if;

    # ---------------------------------
    deq1 := diff( x(t), t ) = v(t);
    deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);

    # ---------------------------------
    ics := { x(0)=x0, D(x)(0)=dx0 };
    sol := dsolve( { deq1, deq2 } union ics );

    # ---------------------------------
    eq := rhs( sol[2] );
    plt1 := plot( eq, t=0..et );

    # ---------------------------------
    # return( plt1, evals, ri );
  end proc: # mySim6
```

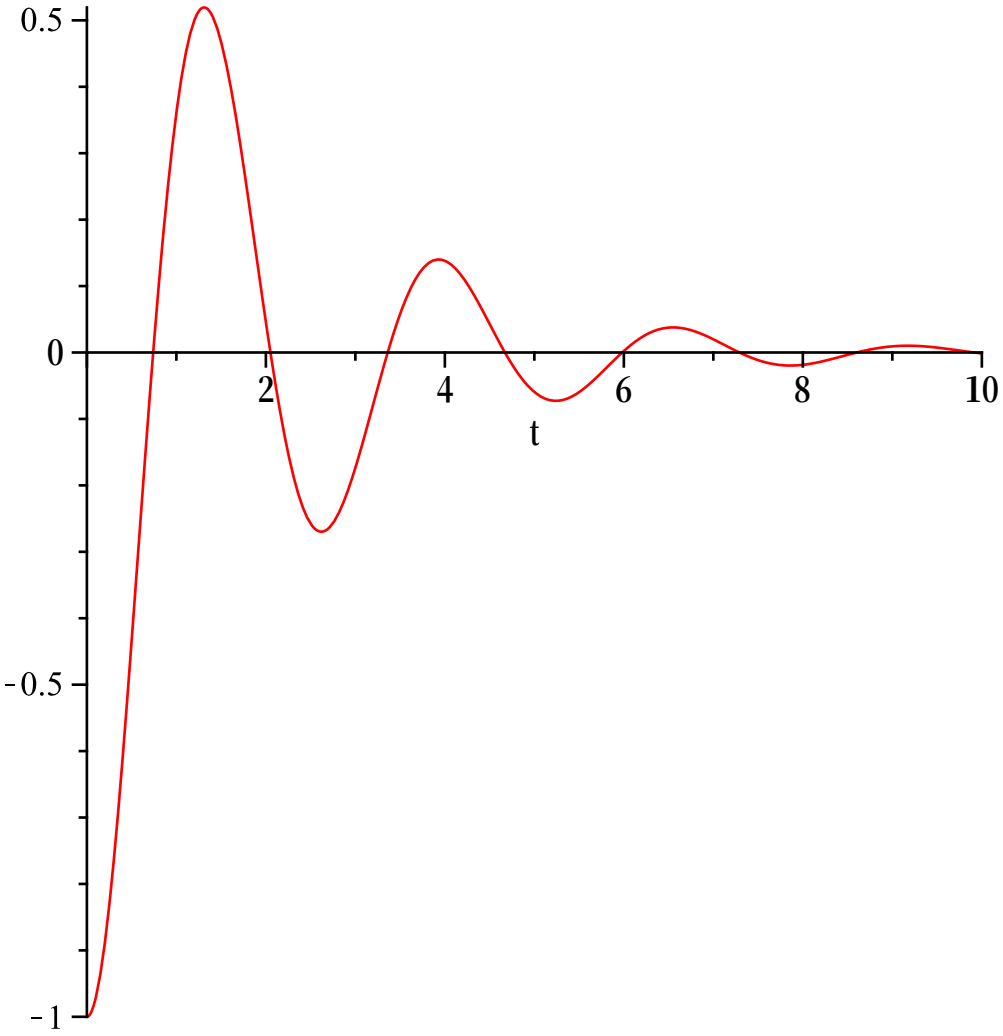
パラメータを定義し，mySim6 を実行します．

```
> sp := 1, 1, 6, -1, 0, 10;
```

$$sp := 1, 1, 6, -1, 0, 10 \tag{70}$$

```
> mySim6( sp );
```

"振動しながら収束します．"



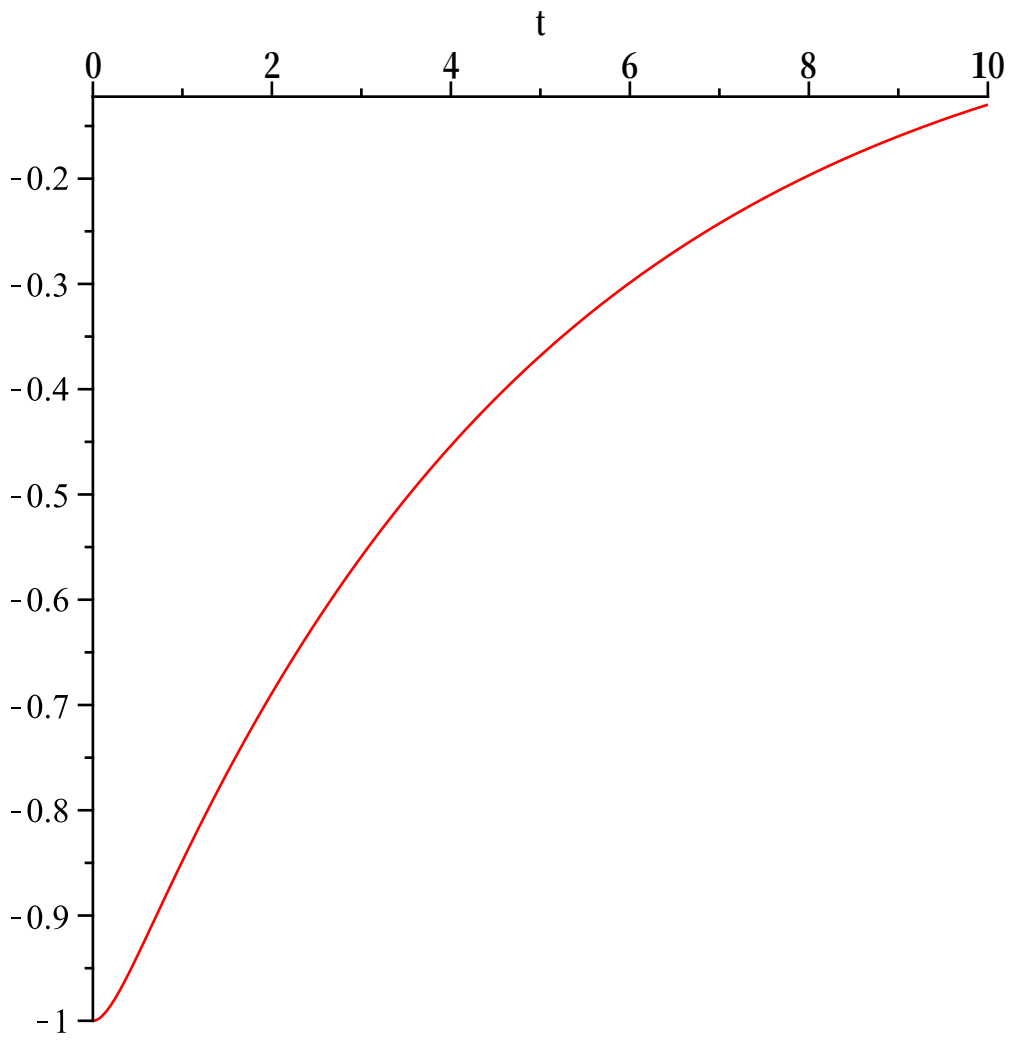
別のパラメータを定義し，mySim6 を実行します．

```
> sp := 1, 5, 1, -1, 0, 10;
```

$$sp := 1, 5, 1, -1, 0, 10 \tag{71}$$

```
> mySim6( sp );
```

"収束しますが，振動はしません．"



固有値のプロット

ここで，固有値を複素平面上にプロットします.

```
> mySim7 := proc( m, c, k, x0, dx0, et )

    # ---------------------------------
    local r1, i1, r2, i2, ri, A, evals, plt1, plt2, arrplt,
          deq1, deq2, ics, sol, eq;

    # ---------------------------------
    A := Matrix( 2,2, [ 0, 1, -k/m, -c/m ] );
    evals := LinearAlgebra[Eigenvalues]( A );

    # ---------------------------------
    r1 := evalf( Re( evals[1] ) );
    i1 := evalf( Im( evals[1] ) );
    r2 := evalf( Re( evals[2] ) );
    i2 := evalf( Im( evals[2] ) );
    ri := [ r1, i1, r2, i2 ];

    # ---------------------------------
    if evalb( r1 < 0 ) and evalb( r2 < 0 ) then

      if i1 = 0 or i2 = 0 then

          print( "収束しますが，振動はしません. " );

       else

          print( "振動しながら収束します. " );

       end if;
```

```
    elif evalb( r1 > 0 ) or evalb( r2 > 0 ) then
      if i1 = 0 or i2 = 0 then
          print( "発散し，振動はしません．" );
       else
          print( "振動しながら発散します．" );
      end if;
    end if;
    # ---------------------------------
    deq1 := diff( x(t), t ) = v(t);
    deq2 := diff( v(t), t ) = -(k/m)*x(t) - (c/m)*v(t);
    # ---------------------------------
    ics := { x(0)=x0, D(x)(0)=dx0 };
    sol := dsolve( { deq1, deq2 } union ics );
    # ---------------------------------
    eq := rhs( sol[2] );
    plt1 := plot( eq, t=0..et );
    plt2 := plots[complexplot]( convert( evals, list ), style=point,
  symbolsize=24 );
    arrplt := Array( [ plt2, plt1 ] );
    plots[display]( arrplt );
    # ---------------------------------
    # return( plt1, evals, ri );
  end proc: # mySim7
```

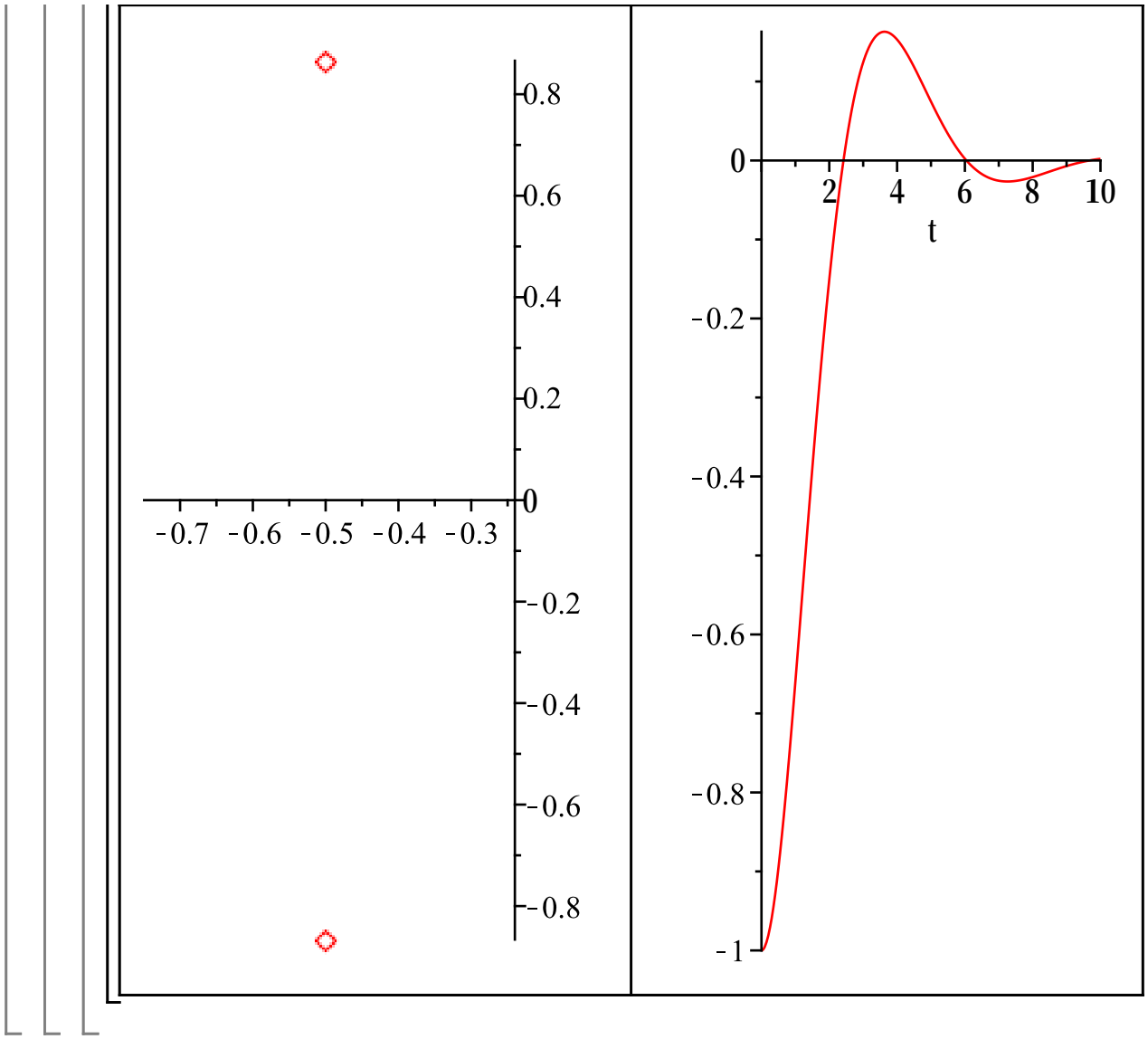
パラメータを定義し，mySim7 を実行します．

```
> sp :=  1,  1,  1, -1, 0, 10;
```

$$sp := 1, 1, 1, -1, 0, 10 \tag{72}$$

```
> mySim7( sp );
```

"振動しながら収束します．"



ステップ・バイ・ステップ式 **はじめての Maple 応用編**

みんなの Maple

Maple コンポーネントの基礎を習得します.

目次

Maple コンポーネントの基礎

Maple コンポーネントは，ワークシート上にユーザ定義のグラフィカル・ユーザー・インターフェース（GUI）環境を構築するための仮想的なインターフェース（デバイス）になります．

初期化します．

```
> restart;
```

Maple コンポーネントの基本的な利用方法

Maple コンポーネントおよびその機能（操作，入力，出力，表示）を利用するためには，DocumentTools パッケージをロードする（読込む）必要があります．

```
> with(DocumentTools);
```
[*Do*, *GetDocumentProperty*, *GetProperty*, *Retrieve*, *SetDocumentProperty*, *SetProperty*]

各コンポーネントには，固有の名前が割り当てられます．以下は，コンポーネントの典型的な使用方法（設計方法）になります．

図．コンポーネントの典型的な利用方法

上図の使用例をもとに具体的な GUI 環境の構築例をいかに示します（ユーザマニュアルより抜粋）．

プロットのためのプロシージャ

```
> Plotter := proc( expr, xmin, xmax, ymin, ymax, col, constr )

    DocumentTools[Do](
      %plotdisplay=plot(
         expr,
        x=xmin..xmax,
        y=ymin..ymax,
         color=col,
         title=typeset(expr, "のプロット"),
         scaling=constr
        )
     ):

  end proc:
>
```

□ 任意の１変数（x）の関数を入力してください.

$x^2 - 3 \quad x - 1$

□ 式の入力後，プロットボタンをクリックしてください.

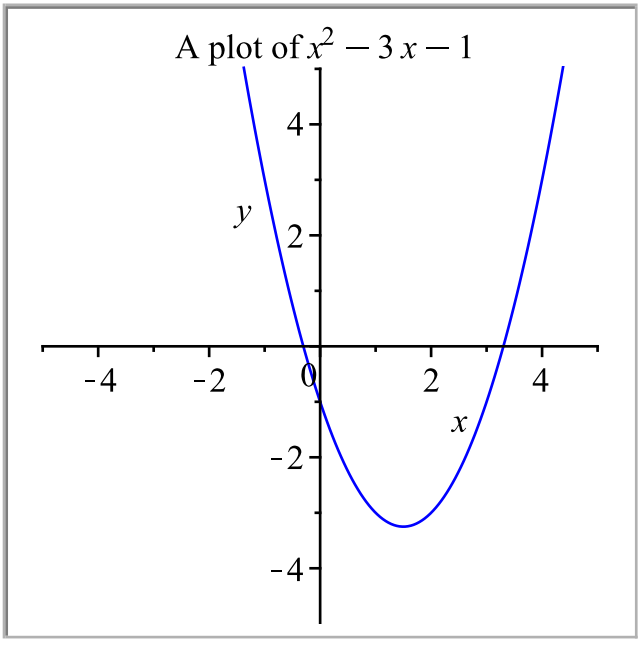
プロット

□ 必要に応じて，プロットの範囲や線の色などを変更します.

プロット範囲　$x =$ -5 to 5　$y =$ -5 to 5

線の色　Blue

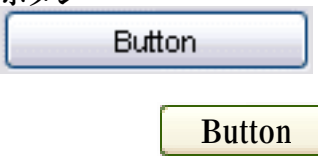
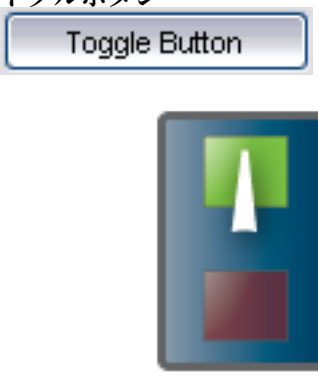
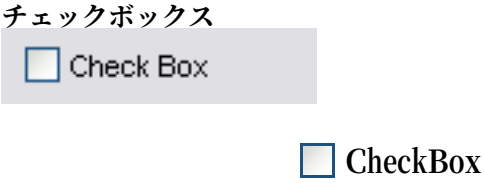
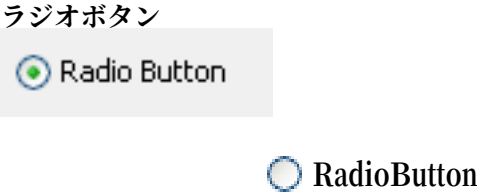
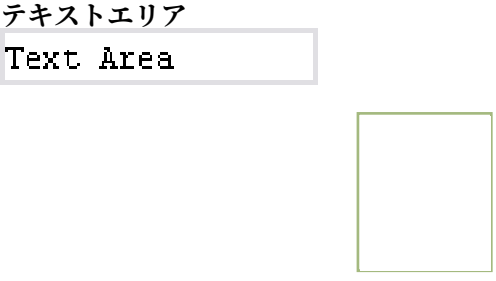
縦横比　固定する / 固定しない

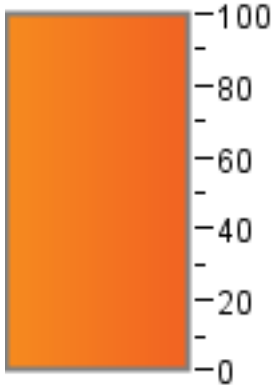
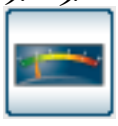


Maple コンポーネントの種類

Maple コンポーネントには，プロシージャ，数式，プロット，あるいはテキストなどを操作する様々なインターフェースデバイスが用意されています（以下は，マニュアルからの抜粋です）.

コンポーネントとアイコン	機能説明

ボタン 	クリックして処理を実行します。つまり、実行コードを実行します。
トグルボタン 	2 つの選択肢のうち 1 つを選択または表示します。表示された画像を変更し、値が変化した際に実行するコードを入力します。
コンボボックス 	ドロップダウンメニューにリストアップされた項目から 1 つを選択します。リスト項目を変更し、値が変化した際に実行するコードを入力します。
チェックボックス 	選択または選択の解除を行います。キャプションを変更し、値が変化した際に実行するコードを入力します。
ラジオボタン 	複数の中から 1 つを選択するために、他のラジオボタンと併せて使用します。値が変化した際に実行するコードを入力します。
テキストエリア 	テキストを入力または表示します。ドキュメントまたは別の埋め込みコンポーネント内のコードに基づいて、値を更新することができます。また、値が変化した際に実行するコードを入力することもできます。
ラベル 	ラベルを表示します。ドキュメントまたは別の埋め込みコンポーネント内のコードに基づいて、値を更新することができます
リストボックス	項目のリストを表示します。リストアップされた項目を変

 	更し、項目の 1つが選択された際に実行するコードを入力します。
スライダー  	整数または浮動小数点の値を選択または表示します。表示を変更し、値が変化した際に実行するコードを入力します。
回転ゲージ  	整数または浮動小数点の値を選択または表示します。表示を変更し、値が変化した際に実行するコードを入力します。
ボリュームゲージ  	整数または浮動小数点の値を選択または表示します。表示を変更し、値が変化した際に実行するコードを入力します。
メーター 	整数または浮動小数点の値を選択または表示します。表示を変更し、値が変化した際に実行するコードを入力します。

ダイヤル

整数または浮動小数点の値を選択または表示します。表示を変更し、値が変化した際に実行するコードを入力します。

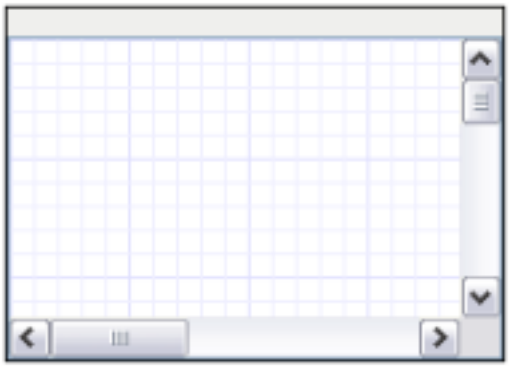
プロット

埋込みプロットウィンドウ

2-D もしくは 3-D プロット、または、アニメーションを表示します。このプロットまたはアニメーションは、他のプロットと同様の操作方法で操作することが可能です(プロットおよびアニメーション参照)。ドキュメントまたは別の埋め込みコンポーネント内のコードに基づいて、値を更新することができます。また、プロット領域内でクリックまたはドラッグするために実行コードポインタが使用されている場合、実行するコードを入力することも可能です。

数式

式を入力または表示します。ドキュメントまたは別の埋め込みコンポーネント内のコードに基づいて、値を更新することができます。

MapleSim  	MapleSim のモデルを表示します．MapleSim は，Maple の拡張機能であり，物理システムのモデリング・シミュレーションを行うためのツールです．

グラフィカル・ユーザー・インターフェース（GUI）の構築

『プログラム前夜』の例題（mySim7 プロシージャ）をコンポーネントを用いて GUI 環境に拡張します.

「それは振動しますか？」プロシージャの確認

系（システム）の固有値を複素平面上にプロットし，そのときの時間応答も同時にプロットする『プログラム前夜』の例題（mySim7 プロシージャ）を再度定義し，適当な値を与えて実行してみます.

```
> mySim7 := proc(m, c, k, x0, dx0, et)

    # ---------------------------------
    local r1, i1, r2, i2, ri, A, evals, plt1, plt2, arrplt,
         deq1, deq2, ics, sol, eq;

    # ---------------------------------
    A := Matrix( 2,2, [0, 1, -k/m, -c/m]);
    evals := LinearAlgebra[Eigenvalues](A);

    # ---------------------------------
    r1 := evalf( Re(evals[1]) );
    i1 := evalf( Im(evals[1]) );
    r2 := evalf( Re(evals[2]) );
    i2 := evalf( Im(evals[2]) );
    ri := [r1, i1, r2, i2];

    # ---------------------------------
    if evalb( r1 < 0 ) and evalb( r2 < 0 ) then

       if i1 = 0 or i2 = 0 then

           print("収束しますが，振動はしません. ");

        else

           print("振動しながら収束します. ");

       end if;


    elif evalb( r1 > 0 ) or evalb( r2 > 0 ) then

       if i1 = 0 or i2 = 0 then

           print("発散し，振動はしません. ");

        else

           print("振動しながら発散します. ");

       end if;

    end if;

    # ---------------------------------
    deq1 := diff(x(t), t) = v(t);
    deq2 := diff(v(t), t) = -(k/m)*x(t) - (c/m)*v(t);

    # ---------------------------------
    ics := {x(0)=x0, D(x)(0)=dx0};
    sol := dsolve( {deq1, deq2} union ics );

    # ---------------------------------
    eq := rhs( sol[2] );
    plt1 := plot( eq, t=0..et );

    plt2 := plots[complexplot]( convert(evals,list), style=point,
symbolsize=24);
```

```
    arrplt := Array([plt2, plt1]);
    plots[display](arrplt);

    # --------------------------------
    # return(plt1, evals, ri);

end proc: # mySim7
```

適当なパラメータの組合わせ（パラメータセット）を定義し，mySim7 プロシージャを実行します．

なお，m，c，k はシステム（微分方程式）の係数，x0，dx0 は微分方程式の初期値，e t は計算時間に該当します．

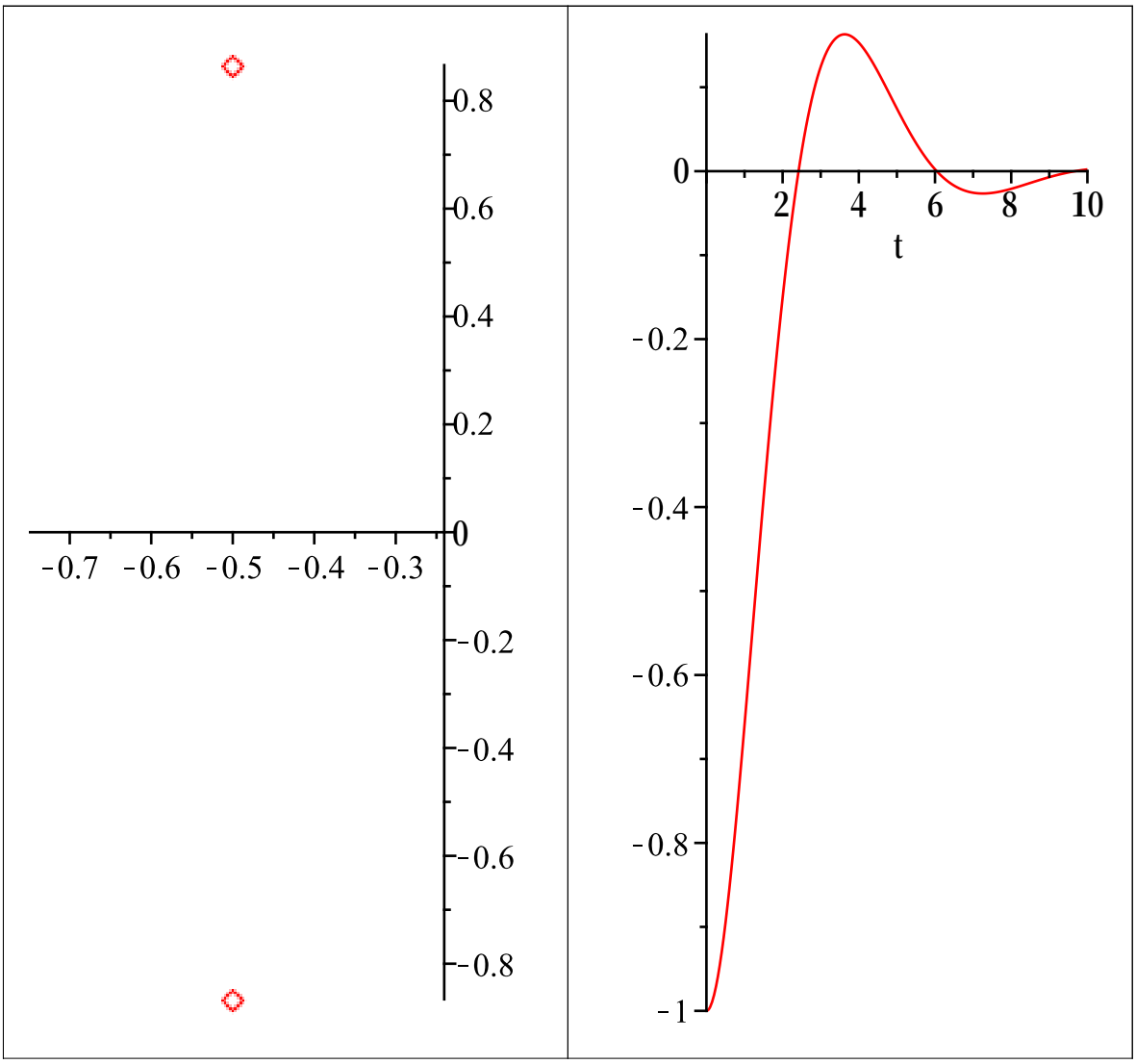
（参考）物理的には，例えば，m，c，k は物体の質量，システムの減衰係数，システムのバネ定数に，また，x0，dx0 は物体の初期位置および初期速度にそれぞれ相当します．

```
> sp := 1, 1, 1, -1, 0, 10;
```

$$sp := 1, 1, 1, -1, 0, 10 \tag{1}$$

```
> mySim7(sp);
```

"振動しながら収束します．"



「それは振動しますか？」の GUI 化

パラメータの値を Maple のスライダーコンポーネント  から与えるように GUI 化します．

mySim7 プロシージャの変更

スライダーコンポーネントの追加に合わせて，mySim7 プロシージャをもとに plotSim プロシージャを定義します．

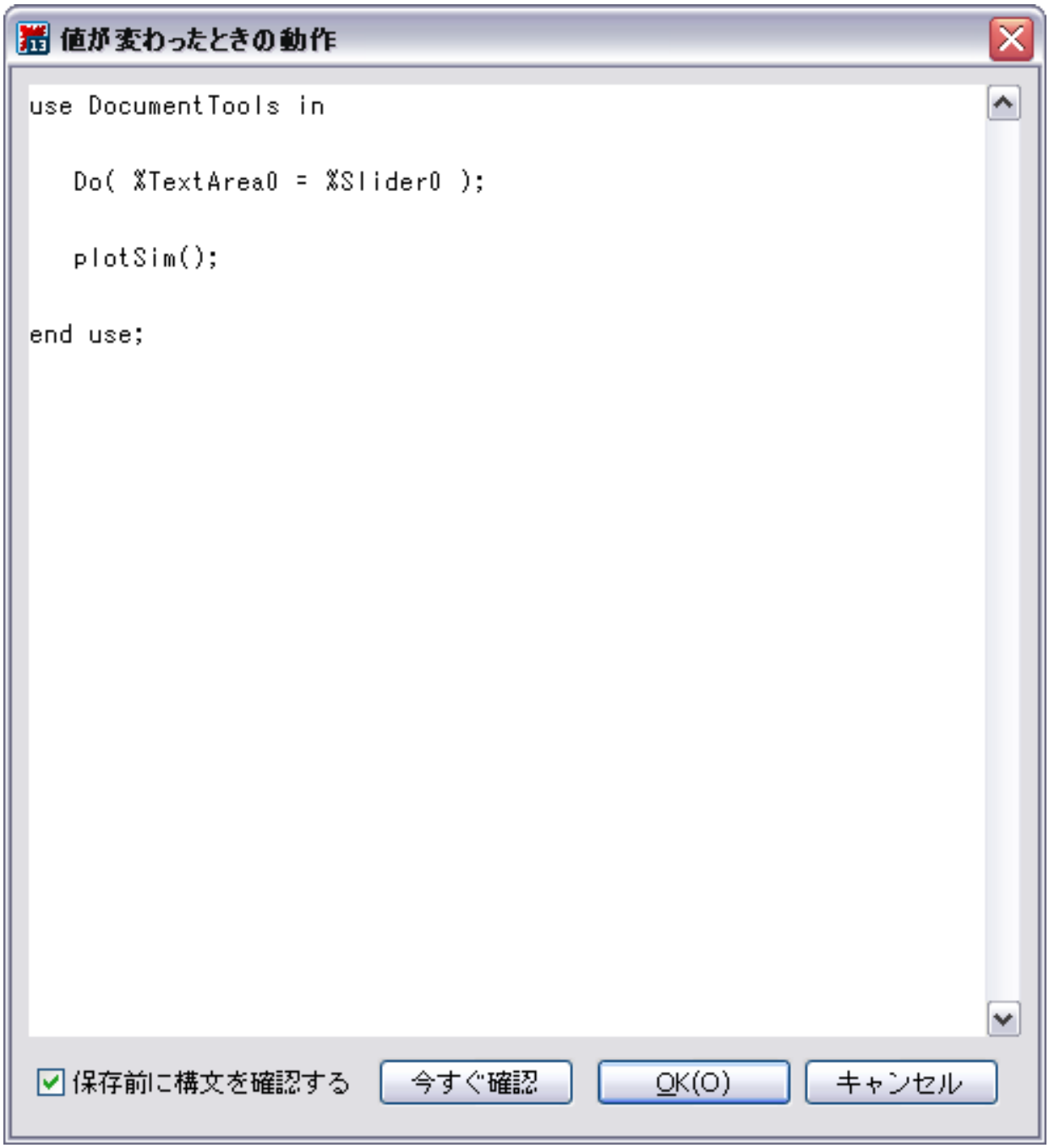
```
> plotSim := proc( )

    # ---------------------------------
    local r1, i1, r2, i2, ri, A, evals, eval1, eval2, plt1, plt2a,
  plt2b, arrplt,
          deq1, deq2, ics, sol, eq,
           m, c, k, x0, dx0, et;

    # ---------------------------------
    DocumentTools[Do]( m   = %TextArea0);
    DocumentTools[Do]( c   = %TextArea1);
    DocumentTools[Do]( k   = %TextArea2);
    DocumentTools[Do]( x0  = %TextArea3);
    DocumentTools[Do]( dx0 = %TextArea4);
    DocumentTools[Do]( et  = %TextArea5);

    # ---------------------------------
    A := Matrix( 2,2, [0, 1, -k/m, -c/m]);
    evals := LinearAlgebra[Eigenvalues](A);
    eval1 := [ evals[1] ];
    eval2 := [ evals[2] ];

    # ---------------------------------
    r1 := evalf( Re(evals[1]) );
    i1 := evalf( Im(evals[1]) );
    r2 := evalf( Re(evals[2]) );
    i2 := evalf( Im(evals[2]) );
    ri := [r1, i1, r2, i2];

    # ---------------------------------
    if evalb( r1 < 0 ) and evalb( r2 < 0 ) then

      if i1 = 0 or i2 = 0 then

       DocumentTools[Do]( %TextArea6 = "収束しますが，振動はしません．" )
  ;

       else

       DocumentTools[Do]( %TextArea6 = "振動しながら収束します．");

      end if;


    elif evalb( r1 > 0 ) or evalb( r2 > 0 ) then

      if i1 = 0 or i2 = 0 then

       DocumentTools[Do]( %TextArea6 = "発散し，振動はしません．");

       else

       DocumentTools[Do]( %TextArea6 = "振動しながら発散します．");

      end if;

    end if;

    # ---------------------------------
    deq1 := diff(x(t), t) = v(t);
    deq2 := diff(v(t), t) = -(k/m)*x(t) - (c/m)*v(t);

    # ---------------------------------
    ics := {x(0)=x0, D(x)(0)=dx0};
    sol := dsolve( {deq1, deq2} union ics );

    # ---------------------------------
    eq := rhs( sol[2] );
```

```
    plt1 := plot( eq, t=0..et, y=-10..10 );

    plt2a := plots[complexplot]( eval1, x=-5..5, y=-5..5, style=point,
symbolsize=24, color=red );

    plt2b := plots[complexplot]( eval2, x=-5..5, y=-5..5, style=point,
symbolsize=24, color=blue );

   DocumentTools[Do]( %Plot0 = plots[display]( [ plt2a, plt2b ] ) );
   DocumentTools[Do]( %Plot1 = plt1 );

end proc: # plotSim
> # terminator
```

（参考）スライダーコンポーネントの設定例

コンポーネントプロパティ

コード（値が変わったときの動作：）


値が変わったときの動作
use DocumentTools in
Do(%TextArea0 = %Slider0);
plotSim();
end use;
保存前に構文を確認する
今すぐ確認
OK(O)
キャンセル

□テーブルを利用しています.

パラメータ名	値の調整	値	単位
質量（m）		33.8716	[kg]
減衰係数（c）		5	[Ns/m]
ばね定数（k）		65	[N/m]
質量の初期位置（x 0）		7	[m]
質量の初期速度（d x 0）	初期速度は常に 0 とします.	0	[m/s]
応答時間（e t）		20	[s]

振動しながら収束します.