Scilab 超入門

齊藤 宣一*

2015 年 4 月 22 日

ウィキペディアでは次のように説明されている：

> Scilab（サイラボ）とは，1990 年からフランスの INRIA [*1] と ENPC [*2] で開発されているオープンソースの数値計算システムである．2003 年 5 月に Scilab コンソーシアムが組織されて以降は，INRIA によって開発されている．
> 数値計算機能以外に，信号処理，行列や多項式の数式処理，関数のグラフィック表示なども充実している．機能やコマンドは，MATLAB クローンと呼ばれるソフト群の中でも特に MATLAB によく似ているが，互換性はない．(http://ja.wikipedia.org/wiki/scilab)

参考 URL と参考書を挙げる：

- Scilab のホーム http://www.scilab.org/
- Scilab 日本語ページ http://www.geocities.jp/rui_hirokawa/scilab/
- 桜井鉄也：MATLAB/Scilab で理解する数値計算，東京大学出版会，2003 年（3,045 円）
- A. Quarteroni, F. Saleri, P. Gervasio：Scientific Computing with MATLAB and Octave, 4th edit., Springer, 2014（加古孝，千葉文浩訳，MATLAB と Octave による科学技術計算，丸善出版，2014 年）

この文書では，Scilab を用いて数値計算を行うための必要最低限の内容を説明する．より詳しいことは，上記の参考 Web ページや参考書で各自で勉強して欲しい．ただし，Scilab の良さを活かした数値計算プログラミングをしている本は少ないので，Quarteroni 先生の本にある MATLAB プログラムの例を参考に，自分で数値計算プログラムを書いてみることをお勧めする．この文書でも，この本のプログラミングを大いに参考にしている．

* norikazu[at]ms.u-tokyo.ac.jp

*1 http://www.inria.fr/

*2 http://www.enpc.fr/

目次

1　基本操作

デスクトップ上の Scilab のアイコン（図 1）をダブルクリックし，Scilab を起動する[*3]．

図 1　Scilab のアイコン

次のようなウインドウが出てくる．

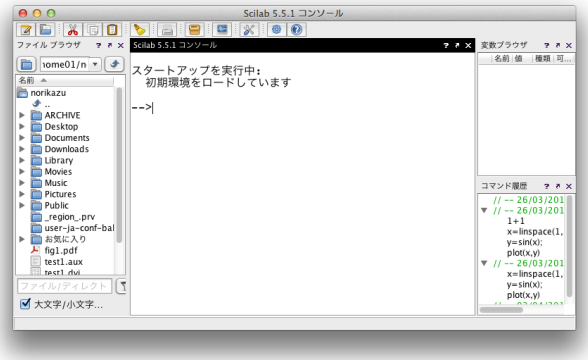

図 2　Scilab の画面

Scilab の画面に表示されている-->の後に，2 + 3 と入力して，[enter/return] をおすと，答えとして 5 が出力される．

```
--> 2 + 3  [enter/return]
 ans  =
    5.
```

この資料において，以後は，[enter/return] を省略する．

なお，フォルダの変更と，現在の作業フォルダの確認は次のようにする．

```
--> cd '~/WORK/1lectures/scilab/'
 ans  =
/Users/norikazu/WORK/1lectures/scilab

--> pwd
 ans  =
/Users/norikazu/WORK/1lectures/scilab
```

2　基本演算

2.1　四則演算など

例を挙げる．詳しい説明はしないが，意味は理解できるであろう．

```
--> 3 + 4
 ans  =
     7.
--> 3 - 4
 ans  =
  - 1.
--> 3 + (-4)
 ans  =
  - 1.
--> 3 * 4
 ans  =
    12.
--> 3/4
 ans  =
    0.75
--> 3^4
 ans  =
    81.
--> 4^(-2)
 ans  =
    0.0625
```

2.2　特別な記号名など

```
--> %pi
 %pi  =
    3.14159265359
-->3*%pi
 ans  =
```

[*3] 言うまでもなく，東京大学 ECCS の Mac OS 上での操作方法を説明する．

```
    9.424777960769
--> 2 + 3*%i
 ans  =
    2. + 3.i
-->(2 + 3*%i)*(2 - 3*%i)
 ans  =
    13.
-->%e
 %e  =
    2.718281828459
```

2.3 組み込み関数

絶対値	$\|x\|$	abs(x)
平方根	$\sqrt{x}$	sqrt(x)
指数関数	e^x	exp(x)
自然対数	$\log x$	log(x)
常用対数	$\log_{10} x$	log10(x)
三角関数	$\sin x$ など	sin(x) など
逆三角関数	acos $x = \cos^{-1} x$ など	acos(x) など
双曲線関数	$\tanh x$ など	tanh(x) など
整数部分		int(x)
剰余	a/b の余り	modulo(a, b)
偏角	$z = re^{i\theta}$ の $\theta \in [-\pi, \pi)$	atan2(z)

```
// 三角関数の値
--> sin(10*%pi)+2*cos(3*%pi)
 ans  =
   - 2.
// 余りの計算
--> modulo(7, 4)
 ans  =
    3.
// 整数部分
--> int(7/4)
 ans  =
    1.
```

なお，上の

```
//
```

ではじまる行はコメント行であり，計算に影響はない．以下では，何をしているのかを説明するために，適宜コメントを挿入するが，自分で実行する際には，省略して良い．

2.4 変数の型と表示桁数

format で表示桁数の制御ができる．以下において，15 や 20 は桁数の指定だが，“必ず 15” でなく，“約 15” と解釈される．また，D+00 は 10^0 の意味である．

```
--> x = 3.0*%pi
 x  =
    9.42477796D+00
--> format("v",15);
--> x
 x  =
    9.424777960769
--> format("v",20);
--> x
 x  =
    9.42477796076937935
--> format("e",15);
--> x
 x  =
    9.42477796D+00
--> format("e",20);
--> x
 x  =
    9.4247779607694D+00
```

ただし，表示桁数が変わっても，Scilab 内で記憶されている桁数は変わらない．

なお，Scilab では，数値はすべて倍精度実数型の変数として扱われる．特に整数として扱いたいときには，int16 などで変換する必要がある．

```
--> format("v",10);
--> n = 1
 n  =
    1.
--> m = int16(1)
 m  =
    1
--> format("e",20);
--> n
 n  =
    1.0000000000000D+00
--> m
```

```
 m  =
    1
```

2.5 その他

現在定義されている変数の一覧を見る.

```
--> who
```

3 ベクトルと行列

3.1 定義と演算

行列を扱う際には，次のようにする.

括弧 [ ]，カンマ , とセミコロン ; の役割に注意すること.

```
// 行列の定義
--> A = [3, -1, 0; 1, 4, 2; 1, 1, 3]
 A  =
    3.  - 1.    0.
    1.    4.    2.
    1.    1.    3.
--> B = [-2, 2, 5; 1, 3, 4; 1, -1, 2]
 B  =
  - 2.    2.    5.
    1.    3.    4.
    1.  - 1.    2.
// 行列の演算
--> 4*A - 2*B
 ans  =
    16.  - 8.   - 10.
    2.     10.    0.
    2.     6.     8.
--> A*B
 ans  =
  - 7.    3.     11.
    4.    12.    25.
    2.    2.     15.
// ベクトルの定義
--> u = [1, 2, 3]
 u  =
    1.    2.    3.
--> v = [4, 5, 6]'
 v  =
    4.
    5.
    6.
--> w = [7; 8; 9]
 w  =
    7.
    8.
    9.
//行列の転置
--> A'
 ans  =
    3.    1.    1.
  - 1.    4.    1.
    0.    2.    3.
--> u'
 ans  =
    1.
    2.
    3.
--> v'
 ans  =
     4.    5.    6.
//ベクトルと行列の演算
--> A*v
 ans  =
    7.
    36.
    27.
--> u*v
 ans  =
    32.
-->u*u'
 ans  =
    14.
```

また,

```
[a: h: b]
```

で，[a, a+h, a+2h, ..., b-h, b] の形のベクトルの自動生成もできる.

```
--> x=[1:0.5:3]'
 x  =
    1.
```

```
    1.5
    2.
    2.5
    3.
--> x=[1:0.1:2]'
 x  =
    1.
    1.1
    1.2
    1.3
    1.4
    1.5
    1.6
    1.7
    1.8
    1.9
    2.
```

あるいは,

```
linspace(a, b, n)
```

を使って次のようにしても良い.

```
--> linspace(0, 1, 5)
 ans  =
    0.    0.25    0.5    0.75    1.

--> linspace(0, 1, 8)'
 ans  =
    0.
    0.1428571
    0.2857143
    0.4285714
    0.5714286
    0.7142857
    0.8571429
    1.
```

3.2 ノルム，行列式と逆行列など

```
--> x = [1,2,3]';
--> y = [4,5,6];
--> A = [3, -1, 0; 1, 4, 2; 1, 1, 3];
--> B = [-2, 2, 5; 1, 3, 4; 1, -1, 2];
// ベクトルのノルム
// 2 ノルム
--> norm(x, 2)
 ans  =
     3.7416574
--> norm(y, 2)
 ans  =
     8.7749644
// 1 ノルム
--> norm(x, 1)
 ans  =
    6.
// infty ノルム
--> norm(y, %inf)
 ans  =
    6.
// 行列ノルム
--> norm(A, 2)
 ans  =
    5.2996529
--> norm(A, 1)
 ans  =
    6.
--> norm(A, %inf)
 ans  =
     7.
--> norm(A)
 ans  =
    5.2996529
//行列式
--> det(A), det(B)
 ans  =
    31.
 ans  =
  - 36.
//逆行列
--> A^(-1)
 ans  =
    0.3225806    0.0967742  - 0.0645161
  - 0.0322581    0.2903226  - 0.1935484
  - 0.0967742  - 0.1290323    0.4193548
--> inv(A)
 ans  =
```

```
    0.3225806    0.0967742  - 0.0645161
  - 0.0322581    0.2903226  - 0.1935484
  - 0.0967742  - 0.1290323    0.4193548
```

なお，処理の最後に ; を入れておくと，処理はされるが結果が表示されなくなる．

```
--> x = [1,2,3]'
 x  =
    1.
    2.
    3.
--> x = [1,2,3]';
```

3.3 特別な行列

```
// 単位行列
--> eye(3, 3)
 ans  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.
--> eye(4, 3)
 ans  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.
    0.    0.    0.
//成分がすべて 1 の行列
--> ones(2,3)
 ans  =
    1.    1.    1.
    1.    1.    1.
--> ones(3,3)
 ans  =
    1.    1.    1.
    1.    1.    1.
    1.    1.    1.
//零行列
--> zeros(3,4)
 ans  =
    0.    0.    0.    0.
    0.    0.    0.    0.
    0.    0.    0.    0.
```

```
//成分が乱数の行列
--> rand(2,3)
 ans  =
    0.2113249    0.0002211    0.6653811
    0.7560439    0.3303271    0.6283918
--> rand(2,3)
 ans  =
    0.8497452    0.8782165    0.5608486
    0.6857310    0.0683740    0.6623569
```

行列 A がすでに定義されているとき，引数に行列 A を入れると A と同じサイズの単位行列などを生成する．例えば，

```
--> A=[1,2;3,4];
--> eye(A)
 ans  =
    1.    0.
    0.    1.
--> zeros(A)
 ans  =
    0.    0.
    0.    0.
//
-->A=[1,2,3,4;5,6,7,8];
-->eye(A)
 ans  =
    1.    0.    0.    0.
    0.    1.    0.    0.
```

対角行列や上三角行列の作り方は次の通り．

```
--> u = [1, 2, 3]
 u  =
    1.    2.    3.
--> A = [1,2,3;4,5,6;7,8,9]
 A  =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
//対角行列
--> diag(u)
 ans  =
    1.    0.    0.
```

```
    0.    2.    0.
    0.    0.    3.
--> diag(diag(A))
 ans  =
    1.    0.    0.
    0.    5.    0.
    0.    0.    9.
//下三角行列
--> tril(A)
 ans  =
    1.    0.    0.
    4.    5.    0.
    7.    8.    9.
--> tril(A, 1)
 ans  =
    1.    2.    0.
    4.    5.    6.
    7.    8.    9.
--> tril(A, -1)
 ans  =
    0.    0.    0.
    4.    0.    0.
    7.    8.    0.
//上三角行列
--> triu(A)
 ans  =
    1.    2.    3.
    0.    5.    6.
    0.    0.    9.
--> triu(A, 1)
 ans  =
    0.    2.    3.
    0.    0.    6.
    0.    0.    0.
--> triu(A, -1)
 ans  =
    1.    2.    3.
    4.    5.    6.
    0.    8.    9.
```

3.4 成分毎の演算

行列 $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ と $B = (b_{ij}) \in \mathbb{R}^{n\times n}$ に対して，成分毎の積

$$A \otimes B = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}b_{n1} & a_{n2}b_{n2} & \cdots & a_{nn}b_{nn} \end{pmatrix},$$

成分毎の累乗

$$\underbrace{A \otimes A \otimes \cdots \otimes A}_{m\ 回} = \begin{pmatrix} a_{11}^m & a_{12}^m & \cdots & a_{1n}^m \\ a_{21}^m & a_{22}^m & \cdots & a_{2n}^m \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^m & a_{n2}^m & \cdots & a_{nn}^m \end{pmatrix},$$

成分毎の商

$$A \otimes (1/b_{ij}) = \begin{pmatrix} a_{11}/b_{11} & a_{12}/b_{12} & \cdots & a_{1n}/b_{1n} \\ a_{21}/b_{21} & a_{22}/b_{22} & \cdots & a_{2n}/b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}/b_{n1} & a_{n2}/b_{n2} & \cdots & a_{nn}/b_{nn} \end{pmatrix}$$

は次のように計算できる．これらは，あとで，3D グラフィクスを扱う際に役に立つ．

```
-->A = [3, -1, 0; 1, 4, 2; 1, 1, 3];
-->B = [-2, 2, 5; 1, 3, 4; 1, -1, 2];
--> A.*B
ans  =
   - 6.  - 2.     0.
    1.    12.    8.
    1.  - 1.     6.
-->A.^2
 ans  =
    9.    1.     0.
    1.    16.    4.
    1.    1.     9.
-->A./B
 ans  =
  - 1.5  - 0.5          0.
    1.     1.3333333    0.5
    1.   - 1.           1.5
```

3.5 ベクトルから行列を生成

```
--> u = [1, 2, 3]'
 u  =
    1.
    2.
```

```
    3.
--> v = [4, 5, 6]'
 v  =
    4.
    5.
    6.
--> X = [u, v]
 X  =
    1.    4.
    2.    5.
    3.    6.
--> w = [u ; v]
 w  =
    1.
    2.
    3.
    4.
    5.
    6.
--> z = X(:)
 z  =
    1.
    2.
    3.
    4.
    5.
    6.
-->u = [u, v, v, u]
 u  =
    1.    4.    4.    1.
    2.    5.    5.    2.
    3.    6.    6.    3.
```

4 2D グラフィクス

4.1 関数のグラフの描画

$0 \leq x \leq \pi$ で関数 $y = \sin x$ のグラフを描く．

そのために，x の区間の $[0, 2\pi]$ を $20 - 1$ 等分して，縦ベクトル x を用意して，つぎのように入力する．すると，x と同じ大きさ縦ベクトル y で，対応する $\sin x$ の値を成分にもつものが定義される．それを，plot2d で描画する．

```
--> x = linspace(0, 2*%pi, 20)';
--> y = sin(x);
--> plot2d(x, y)
```

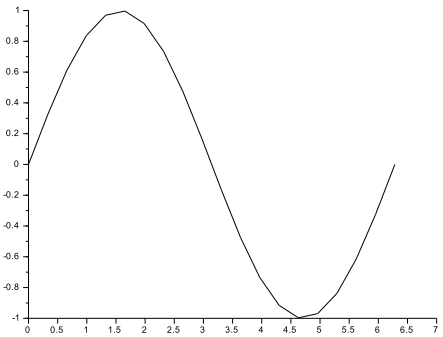

図 3

次のように，x や y を横ベクトルとして扱っても同じ出力が得られるが，後の利用との整合性のため，グラフ描画の際には縦ベクトルを使うことに決めておくと，混乱が少ないであろう．

```
--> x = linspace(0, 2*%pi, 20);
--> y = sin(x);
--> plot2d(x, y)
```

4.2 複数のグラフの描画

関数を重ねるには次のようにすれば良い．

```
--> x = linspace(0, 2*%pi, 100)';
--> y = sin(x);
--> plot2d(x, y);
--> y = sin(3.0*x);
--> plot2d(x, y)
```

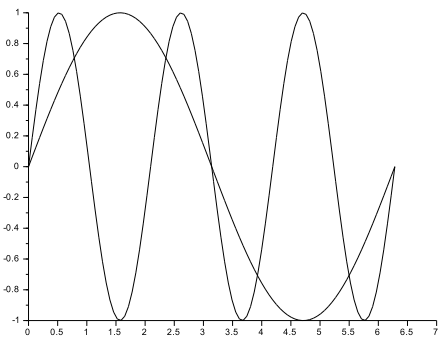

図 4

ただし，上のように入力するとグラフの色は常に黒となり，わかりにくい．グラフ毎に色を指定するには，次のようにする（図 5）．

```
--> x = linspace(0, 2*%pi, 100)';
--> y = sin(x);
--> plot2d(x, y, style = 5);
--> y = sin(3.0*x);
--> plot2d(x, y, style = 2);
```

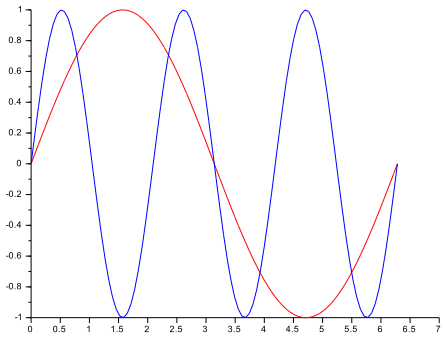

図 5

style で指定する数値の意味は次の通りである．

数値	色	数値	色
1	黒	5	赤
2	青	6	マゼンタ
3	緑	7	黄
4	シアン		

あるいは，次のように，RGB 値で指定することもできる．color(r, g, b) の r, g, b は，それぞれ，赤・緑・青の強さを表しており，0 ～ 255 の整数値として指定する．

```
--> plot2d(x, y, style = color(255, 224, 224));
```

グラフの数が多いときに，いちいち色を指定するのは煩わしい．そこで，次のように入力すると，図 6 のように，グラフ毎に色を替えて描画される．この入力では，はじめに 100 次元の縦ベクトル x, y1, y2 を定義した後に，100 × 2 行列 [y1, y2] を定義して，それを plot2d に渡す．そうすると，plot2d は，(x, y1) と (x, y2) のグラフを描画するのである．

```
--> x = linspace(0, 2*%pi, 100)';
--> y1 = sin(x);
--> y2 = sin(3.0*x);
--> plot2d(x, [y1, y2]);
```

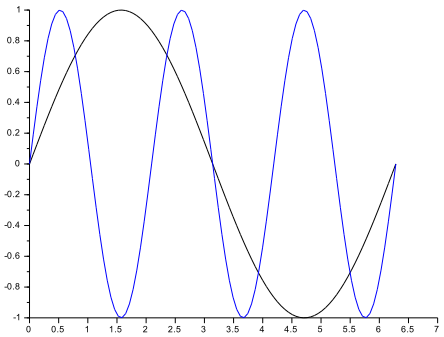

図 6

色を自分で指定したいときには次のようにする（出力は図 5 と同じ）．

```
--> x = linspace(0, 2*%pi, 100)';
--> y1 = sin(x);
--> y2 = sin(3.0*x);
--> plot2d(x, [y1, y2], style=[5,2]);
```

4.3 格子や凡例

格子の表示，凡例やタイトルの付け方は次の通り（図 7）．

```
--> x = linspace(0, 2*%pi, 100)';
--> y1 = sin(x);
--> y2 = sin(3.0*x);
--> plot2d(x, [y1, y2], style=[5,2]);
// 格子の表示
-->xgrid();
// 凡例を付ける
// (1: 右上) (2: 左上) (3: 右下) (4: 左下)
--> legend("sin(x)", "sin(3x)", 1);
// タイトルと軸のラベル
--> title("三角関数のグラフ");
--> xlabel("x");
--> ylabel("y");
// 上の三行は次で代用できる
```

```
--> xtitle("三角関数のグラフ","x","y");
```

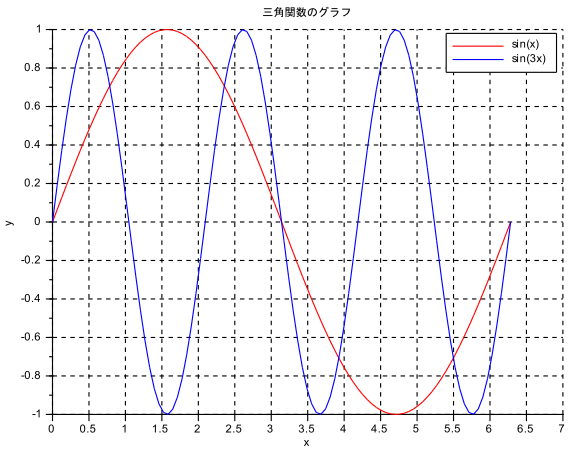

図 7

4.4 ファイルへの保存

描画したグラフをファイルに保存する．グラフを表示しているウインドウを選択した状態で，Scilab メニューで，

ファイル > エクスポート

とする．出てくるウインドウで次の項目を指定すると，「指定したフォルダ」に，`fig1.pdf`，`fig1.eps` などの名前でにファイルとして保存される（図 8）．

項目	例
Save As:	`fig1`
Save As:の下	指定したいフォルダ
File Format:	PDF 画像

あるいは，

項目	例
Save As:	`fig1`
Save As:の下	指定したいフォルダ
File Format:	カプセル化されたポスト スクリプトイメージ (EPS)

なお，この操作は，次のようにコマンドからも行える．保存されるのは，作業中のフォルダになる．数字の 0 は，グラフのウインドウに表示されている「グラフィクス・ウインドウ番号」である．

```
--> xs2pdf(0, "fig1.pdf")
```

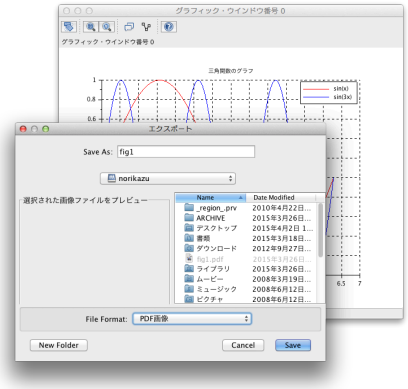

図 8　図の保存

```
--> xs2eps(0, "sinx.eps")
```

なお，グラフィクス・ウインドウには「グラフィクス・ウインドウ番号」が自動的に $0, 1, 2, \ldots$ とふられるが，これを自分で指定するときには，

```
--> scf(4); //グラフィクス・番号を 4 に指定
--> plot2d(x, y)
```

とする．

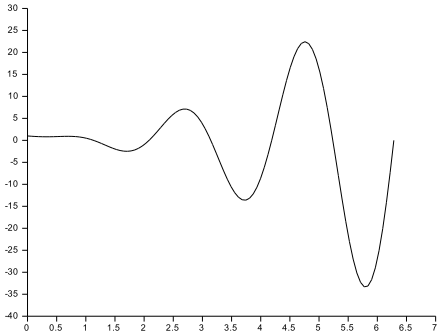

図 9

5 自前の関数の利用

関数 $f(x) = e^{-x} + x^2 \sin(3x)$ のグラフを書きたいときには，前に説明したように，

```
--> x = linspace(0, 2*%pi, 100)';
--> y = exp(-x) + (x.^2).*sin(3*x);
```

```
--> plot2d(x, y)
```

とすれば良い（図 9）．

　この $f(x)$ を，後で，再度利用したいときには，これを一つの関数として定義しておくと便利である．それには次のようにする．まず，図 10 で示したボタンを押し，SciNotes（Scilab に付属するテキストエディタ）を起動する（図 11）．あるいは，Scilab のウインドウで，

```
--> scinotes
```

とする．そして，次の Prog 5.1 ように入力する（図 11）．

Prog 5.1　funcs.sce の中身

```
// 関数 1
function y = func1(x)
y=exp(-x)+(x.^2).*sin(3.0*x);
endfunction
```

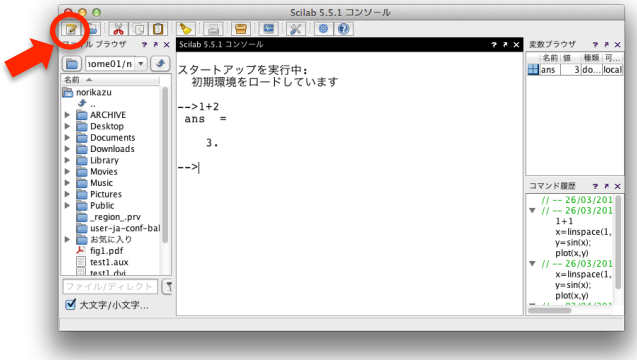

図 10　SciNotes 起動

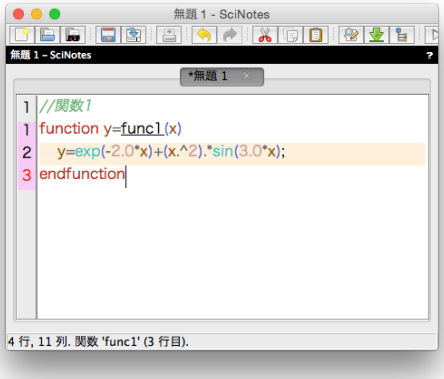

図 11　SciNotes への入力

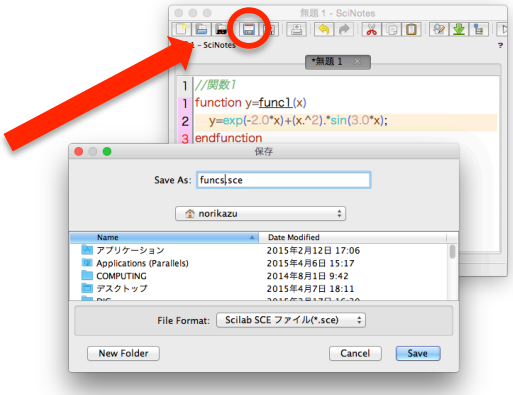

図 12　ファイルの保存

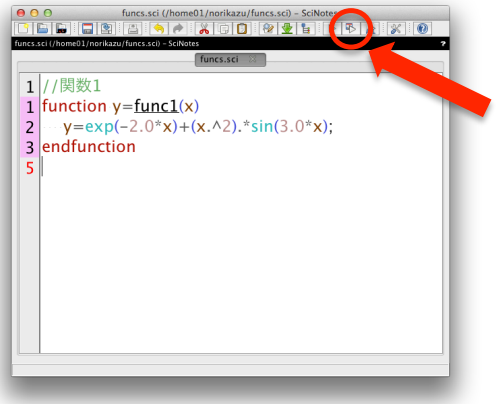

図 13　プログラムの実行

入力した後に，これを，（例えば，funcs.sce）という名前で保存する（図 12）．そして，図 13 に示す**実行ボタン**を押した後，

```
--> x = linspace(0, 2*%pi, 100)';
--> y = func1(x);
--> plot2d(x, y)
```

とする．実行ボタンを押す代わりに，F5 キーを押しても良いし，次のようにしても良い．

```
// funcs.sce を読み込む
--> exec("funcs.sce");
// グラフの描画
--> x = linspace(0, 2*%pi, 100)';
--> y = func1(x);
--> plot2d(x, y)
```

func1 を，例えば，Prog 5.2 のように修正したときには，再度，実行ボタンを押さなければならない（そうしないと，修正が反映されない）．

Prog 5.2　funcs.sce における関数 1 の修正

```
// 関数 1
function y = func1(x)
y=exp(-2.0*x)+(x.^2).*sin(3.0*x);
endfunction
```

しかし，これを実行すると，

```
-->exec("funcs.sce");
警告：関数の再定義です：func1，funcprot(0) を使ってこのメッセージを避けて下さい
```

という警告が表示されるが，これは，関数が再定義されたことを知らせてくれる警告なので，重要である（特に対処は必要ない）．

新しく関数を定義したいときには，funcs.sce に追加で書き込んでいけば良い（Prog 5.3）．

Prog 5.3　funcs.sce への追加書き込み

```
// 関数 1
function y = func1(x)
y=exp(-2.0*x)+(x.^2).*sin(3.0*x);
endfunction
// 関数 2
function y = func2(x)
y=exp(x)-2*x-1
endfunction
```

6　プログラミングと for 文

Scilab では，関数（数学の組み込み関数でなく，サブプログラム (sub-program) と呼ぶべきもの）を自前で定義することにより，プログラミングを行う．

例示のために，まずは，funcs.sce に Prog 6.1 を追加で書き込む（中身は後で説明する）．

Prog 6.1　funcs.sce への追加書き込み

```
// 1 + 2 + 3 + .... + n
function wa = summation1(n)
wa = 0.0;
for i=1:n
    wa = wa + i;
end
endfunction

// 1 + 2^2 + 3^2 + .... + n^2
function wa = summation2(n)
wa = 0.0;
for i=1:n
    wa = wa + i*i;
end
endfunction

// 1 + 1/2 + 1/3 + .... + 1/n
function wa = sumfrac1(n)
wa = 0.0;
for i=1:n
    wa = wa + 1.0/(i);
end
endfunction

// 1 + 1/(2^2) + 1/(3^2) + .... + 1/(n^2)
function wa = sumfrac2(n)
wa = 0.0;
for i=1:n
    wa = wa + 1.0/(i*i);
end
endfunction
```

この関数を利用するには次のようにする．

```
// funcs.sce を読み込む
--> exec("funcs.sce");
// 自前の関数を利用した計算
--> summation1(5)
 ans  =
    15.
--> summation2(10)
 ans  =
    385.
--> sumfrac1(10)
 ans  =
    2.9289683
--> sumfrac2(35)
 ans  =
    1.6167669
```

さて，Prog 6.1 の中身を簡単に説明する．まず，一つ一つの関数は，

```
function 答え = 関数の名前（データ）
  関数の中身
endfunction
```

の形をしている．次に，例えば summation1(n) の中で，

```
wa = 0.0;
for i=1:n
    wa =  wa + i;
end
```

の部分は **for 文**を用いた繰り返し計算をしている．これは，

for 文

```
for i=1:n
    （処理）
end
```

の形をしており，まず，i=1 に対して（**処理**）を行い，次に，i=2 に対して（**処理**）を行う，これを続けていき，最後に，i=n に対して（**処理**）を行ったら，この **for 文**の処理は終了である．したがって，例えば，summation1(10) を実行すると，i=1 のとき wa=0.0+1.0，i=2 のとき wa=0.0+1.0+2.0 となり，これが続けられ，最後に i=10 で，wa=0.0+1.0+2.0+...+10.0 となり，このときの wa が計算結果として表示される．

7 二分法と if 文

関数

$$f(x) = e^x - 2x - 1$$

について，方程式

$$f(a) = e^a - 2a - 1 = 0$$

の（唯一の）正の解 a を計算する方法を考察する．初等的な考察により，$1 < a < 2$，かつ，$f(1) < 0$，$f(2) > 0$ であることがわかる．

二分法 (bisection method) では，まず，

$$[\alpha_0, \beta_0] = [1, 2]$$

とする．そして，$k \geq 0$ に対して，

$$x_k = \frac{1}{2}(\alpha_k + \beta_k),$$

$$[\alpha_{k+1}, \beta_{k+1}] = \begin{cases} [\alpha_k, x_k] & (f(x_k)f(\beta_k) \geq 0 \text{ のとき}) \\ [x_k, \beta_k] & (f(x_k)f(\beta_k) < 0 \text{ のとき}) \end{cases}$$

と定める．このとき，

$$|x_k - a| \leq \beta_k - \alpha_k = \left(\frac{1}{2}\right)^{k+1} (\beta_0 - \alpha_0)$$

が成り立つ．これより，$|x_k - a| \leq \varepsilon$ を満たす近似解が欲しい場合には，反復回数が $k^* = \dfrac{1}{\log 2} \log \dfrac{|\beta - \alpha|}{\varepsilon}$ 程度必要となることもわかる．詳しくは，テキストの §4.2 を見よ．これを Scilab で実行するには Prog 7.1 のような関数を作成する．

Prog 7.1　funcs.sce への追加書き込み

```
// 二分法
// 入力 初期区間 [a, b], 許容誤差限界 ep
// 関数 fun（function で定義しておく）
// 出力 xvect = [x1, x2, ...], fx = [f(x1), f(x2),
    ...]
// xdiff = [e1, e2, ...] ek=(bk-ak)/2
function [xvect, fx, xdif] = bisec1(a, b, ep, fun)
// 最大反復回数
kmax = (1.0/log(2.0))*log((b-a)/ep) + 1;
// とりあえず成分が 0個のベクトルを定義
xvect=[]; fx=[]; xdif=[];
// 二分法の反復計算
for k = 1:kmax
// 誤差の見積もり ek = (bk-ak)/2
    xdif = [xdif ; 0.5*abs(b - a)];
// 中点xk=c=(b+a)/2を定義しf(xk)=f(c)を計算
    c = (a + b)/2; x = c; fc = fun(x);
    xvect = [xvect ; x]; fx = [fx ; fc];
    x = a;
// 新しい区間の定義
    if fc*fun(x)>=0.0
       a=c;
    else
       b=c;
    end // if の終了
end // for の終了
endfunction
```

bisec1(a, b, ep, fun) の中にある，a，b，ep，fun は，ユーザが与えるデータであり，これを**引数**と言う．ここでは，a $= \alpha_0$，b $= \beta_0$，ep $= \varepsilon$，func $= f(x) = e^x - 2x - 1$ の意味で用いている．

「新しい区間の定義」のために，ここでは，**if 文**を用いている．基本形は次の通りである．

if 文

```
if （条件）
    （処理）
end
```

この表現では，（**条件**）が成立するときには（**処理**）を実行する．また，

if 文

```
if （条件）
    （処理 1）
else
    （処理 2）
end
```

という書式もある．このときには，（**条件**）が成立するときには（**処理 1**）を実行し，それ以外の場合には，（**処理 2**）を実行する．

（**条件**）の書き方は次の通り．

$a \geq b$	a>=b
$a \leq b$	a<=b
$a > b$	a>b
$a < b$	a<b
$a = b$	a==b
$a \neq b$	a~=b

bisec1 を実行すると次のようになる．fun には，既に定義してある，func2 を代入する．なお，1.0D-16 は 1.0×10^{-16} の意味である．

```
--> [xv, fx, xd] = bisec1(1, 2, 1.0D-6, func2)
 xd  =
    0.5
    0.25
(中略)
    0.0000038
    0.0000019
    0.0000010
 fx  =
    0.4816891
  - 0.0096570
(中略)
  - 0.0000052
  - 0.0000023
  - 0.0000009
 xv  =
    1.5
    1.25
(中略)
    1.2564278
    1.2564297
    1.2564306
```

8 応用：Fourier 級数の収束

いままでのまとめとして，関数

$$f(x) = \pi - |x| \qquad (-\pi \leqq x \leqq \pi)$$

の Fourier 級数展開

$$f(x) = \frac{\pi}{2} + \frac{4}{\pi}\left(\frac{\cos x}{1^2} + \frac{\cos 3x}{3^2} + \frac{\cos 5x}{5^2} + \cdots\right)$$

の収束の様子を図示する関数を作成する．

$$S_n(x) = \frac{\pi}{2} + \frac{4}{\pi}\sum_{i=1}^{n}\frac{\cos((2i-1)x)}{(2i-1)^2}$$

とおく．

Prog 8.1 funcs.sce への追加

```
// Fourier 級数の例題 1
function [value] = four1(x, n)
value = cos(x);
for i=2:n
    value = value + cos((2*i-1)*x)/((2*i-1)^2);
end
value = %pi/2.0 + (4.0/%pi)*value;
endfunction

// Fourier 級数の描画（指定された n のみ）
function four_draw1(n, fun, m)
x = linspace(-1.5*%pi, 1.5*%pi, m)';
yy = []; y = fun(x, n);
scf(1); plot(x, y); xgrid();
xtitle("フーリエ級数の収束", "x", "y");
xs2eps(1,"four1.eps");
//xs2pdf(1,"four1.eps");
endfunction

// Fourier 級数の描画 (n = 1, 10, 20, 50のみ)
function four_draw2(fun, m)
x = linspace(-1.5*%pi, 1.5*%pi, m)'; yy = [];
y = fun(x, 1); yy = [yy, y];
y = fun(x, 10); yy = [yy, y];
y = fun(x, 20); yy = [yy, y];
y = fun(x, 50); yy = [yy, y];
scf(1); plot(x, yy); xgrid();
xtitle("フーリエ級数の収束 (n=1,10,20,50)", "x", "y");
xs2eps(1, "four2.eps");
//xs2pdf(1, "four2.pdf");
endfunction
```

簡単に説明すると，four1(x, n) では，与えられた x $= x$ と n $= n$ に対して，5 行目の end が終了した時点で，

$$\texttt{value} = \sum_{i=1}^{n}\frac{\cos((2i-1)x)}{(2i-1)^2}$$

を計算している．そして，最終的な答えとして，

$$\frac{\pi}{2} + \frac{4}{\pi}\texttt{value}$$

を出力するのである．

一方，four_draw1(n, fun, m) では，与えた n $= n$ に対して，$y = S_n(x)$ のグラフを描画する．そのために，区間 $[-\frac{3}{2}\pi, \frac{3}{2}\pi]$ を $m-1$ 等分してベクトル x を作り，描画用のデータ (x, y) を作る．また，描画の後に，結果を four1.eps と言う名前のファイルに保存する．four_draw2(fun, m) では，n $= 1, 10, 20, 50$ に対して，$y = S_n(x)$ のグラフを重ねて描画し，，結果を four2.eps と言う名前のファイルに保存する．fun には，上で作った four1(x, n) を与えれば良い．

```
--> exec("funcs.sce");
--> four_draw1(10, four1, 100)
 ans  =
     0.
--> four_draw2(four1, 100)
 ans  =
     0.
```

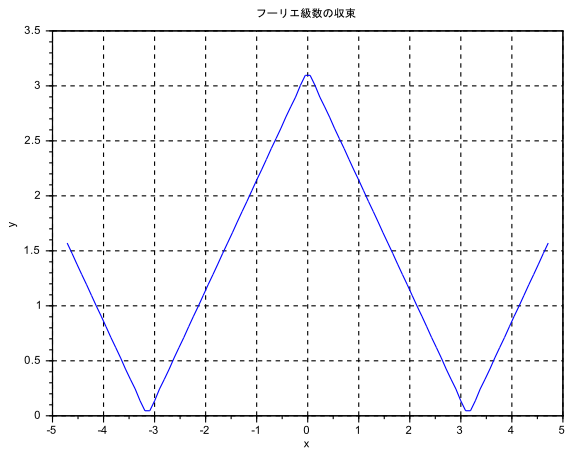

図 14　four_draw1(10, four1, 100) の結果.

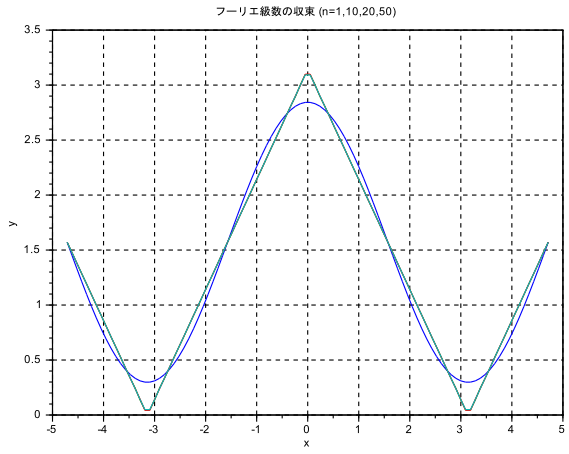

図 15　four_draw2(four1, 100) の結果.

9　ファイルへの入出力

9.1　出力

2 つのベクトルのデータを，test.txt に書き込む.

```
--> x = [1,2,3]';
--> y = [4,5,6]';
--> F1 = mopen("test.txt", "w");
--> mfprintf(F1,"%f %f\n",x,y)
--> mclose(F1);
```

test.txt

```
1.000000 4.000000
2.000000 5.000000
3.000000 6.000000
```

9.2　入力

先ほど作成した test.txt からデータを読み込む.

```
--> F1 = mopen("test.txt","r");
--> [n,X,Y] = mfscanf(%inf,F1,"%f %f");
--> mclose(F1);
--> n
 n  =
    2.
--> X
 X  =
    1.
    2.
    3.
--> Y
 Y  =
    4.
    5.
    6.
```

mfscanf の最初の引数では，読み込むデータの行数を指定する．%inf はデータがある限り読み込むよう指定する命令である.

10　3D グラフィクス

この節 （§10）は，講義では直接扱わないので，省略して良い.

10.1　準備

Scilab では，

$$\boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{pmatrix} \in \mathbb{R}^{N+1}, \quad \boldsymbol{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_M \end{pmatrix} \in \mathbb{R}^{M+1},$$

$$Z = \begin{pmatrix} z_{00} & z_{01} & \cdots & z_{0N} \\ z_{10} & z_{11} & \cdots & z_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ z_{M0} & z_{M1} & \cdots & z_{MN} \end{pmatrix} \in \mathbb{R}^{(M+1)\times(N+1)}$$

というデータが与えられたとき，

(x_0, z_{00}), $(x_0, z_{10}), \ldots, (x_0, z_{M0})$ を結んでできる曲線,
(x_1, z_{01}), $(x_0, z_{11}), \ldots, (x_1, z_{M1})$ を結んでできる曲線,

$$\vdots$$

(x_N, z_{0N}), $(x_N, z_{1N}), \ldots, (x_N, z_{MN})$ を結んでできる曲線

という N 本の曲線を空間座標内に描画するコマンド param3d1 がある.

10.2 曲線の描画

コマンド param3d1 を利用して，関数 $z = \sin x \cos y$ $(0 \leq x \leq \pi,\ -\pi/2 \leq y \leq \pi/2)$ のグラフを 3 次元空間内に描画する手順を述べる[*4].

そのために，まず，x 軸と y 軸に

$$x_j = j\frac{\pi}{N} \quad (0 \leq j \leq N),$$
$$y_k = -\frac{\pi}{2} + k\frac{\pi}{M} \quad (0 \leq k \leq M)$$

と格子点を生成して，ベクトルを

$$\boldsymbol{x} = \begin{pmatrix} x_0 \\ \vdots \\ x_N \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_0 \\ \vdots \\ y_M \end{pmatrix}$$

と定める．そして，補助的に 2 つの行列

$$X \in \mathbb{R}^{(M+1)\times(N+1)}, \quad Y \in \mathbb{R}^{(M+1)\times(N+1)}$$

を

$$X = \begin{pmatrix} x_0 & x_1 & \cdots & x_N \\ x_0 & x_1 & \cdots & x_N \\ \vdots & \vdots & \ddots & \vdots \\ x_0 & x_1 & \cdots & x_N \end{pmatrix} = (x_{jk}),$$

$$Y = \begin{pmatrix} y_0 & y_0 & \cdots & y_0 \\ y_1 & y_1 & \cdots & y_1 \\ \vdots & \vdots & \ddots & \vdots \\ y_M & y_M & \cdots & y_M \end{pmatrix} = (y_{jk})$$

で定義しておく．これらを用いて，さらに，

$$V = (\sin(x_{jk})) \in \mathbb{R}^{(M+1)\times(N+1)},$$
$$W = (\cos(y_{jk})) \in \mathbb{R}^{(M+1)\times(N+1)}$$

という行列を定義する．なお，これらを以後,

$$V = \sin X, \quad W = \cos Y$$

のように書く．そうして,

$$Z = \sin X \otimes \cos Y$$
$$= \begin{pmatrix} \sin(x_0)\cos(y_0) & \cdots & \sin(x_N)\cos(y_0) \\ \sin(x_0)\cos(y_1) & \cdots & \sin(x_N)\cos(y_1) \\ \vdots & \ddots & \vdots \\ \sin(x_0)\cos(y_M) & \cdots & \sin(x_N)\cos(y_M) \end{pmatrix}$$

を計算して（$\otimes$ は 7 ページで扱った行列の成分毎の乗算を表す記号である），これを param3d1 に渡せば良い.

この手順を Scilab のコマンドで具体的に書くと次のようになる（$N = M = 20$ とする．結果は図 16）.

```
// ベクトル x, y の生成
--> xx = linspace(0, %pi, 20)';
--> yy = linspace(-0.5*%pi, 0.5*%pi, 20)';
// 行列 X, Y の生成（専用のコマンドを使う）
--> [X, Y] = meshgrid(xx, yy);
// 2 つの行列の成分毎の積
--> Z = sin(X).*cos(Y);
// 関数の描画
--> param3d1(X, Y, Z, alpha = 80, theta = 45);
```

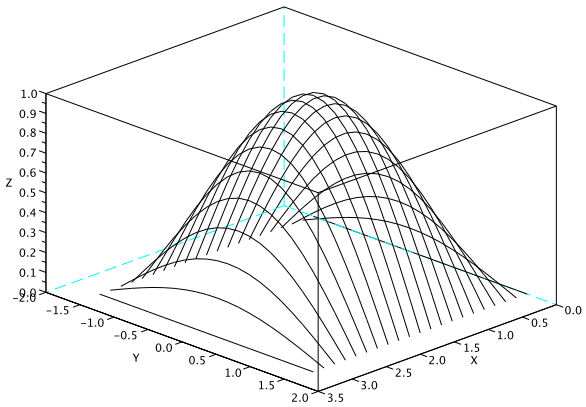

図 16

なお，alpha と theta は，視点を定義するためのパラメータで，それぞれ，z 軸と x 軸とのなす角を表す．これらの適切な数値を事前に知ることは難しいので，一度書いてみた後に，グラフィクス・ウインドウを選択した状態で，Scilab メニューで,

ツール > 2D/3D 回転 (R)

と進む．そうするとマウスの操作で図を回転することができるので，ちょうど良い視点を探せば良い．回転させている間は，alpha と theta の値がグラフィクス・ウインドウの左下に数値で表示される.

上の param3d1 の代わりに，mesh を使うと，x, y 軸の両方の方向に線を描く．また，陰線処理も行われる（図 17）.

[*4] 他にも良いやり方がありますが，ここでは，一番基本的な方法を紹介します.

```
// 関数の描画 x，y 軸方向両方に線を描く
--> mesh(X, Y, Z);
```

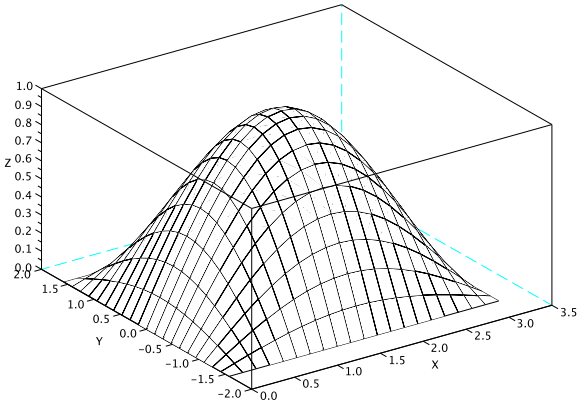

図 17

10.3 曲面とカラーマップ

曲線でなく曲面で表示する場合には，次のようにする (図 18)．

```
--> xx = linspace(0, %pi, 20)';
--> yy = linspace(-0.5*%pi, 0.5*%pi, 20)';
--> [X, Y] = meshgrid(xx, yy);
--> Z = sin(X).*cos(Y);
// 曲面の描画
--> surf(X, Y, Z);
```

しかしデフォルトの配色はとても “美しい” と言えるものではないので，自分で配色を指定した方が良い．

```
--> xx = linspace(0, %pi, 20)';
--> yy = linspace(-0.5*%pi, 0.5*%pi, 20)';
--> [X, Y] = meshgrid(xx, yy);
--> Z = sin(X).*cos(Y);
// 曲面の描画
--> surf(X, Y, Z);
// カラーマップの指定
--> xset("colormap", coolcolormap(32));
// カラーバーの表示
--> colorbar(min(Z), max(Z));
```

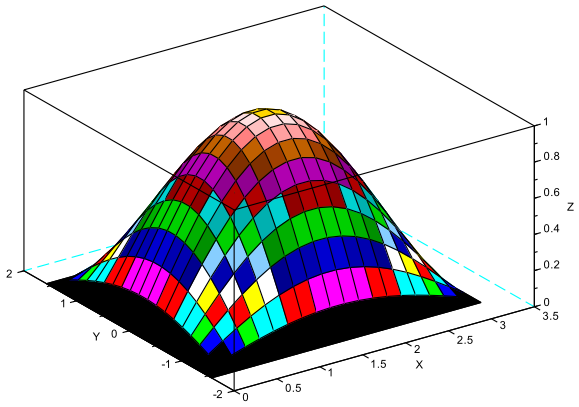

図 18

結果は図 19 となる．`coolcolormap(32)` の 32 は使用する色の数である．他にも次のような配色の設定が選べる．

autumncolormap	
bonecolormap	
coolcolormap	(図 19)
coppercolormap	
graycolormap	(図 20)
hotcolormap	
hsvcolormap	
jetcolormap	
oceancolormap	
pinkcolormap	
rainbowcolormap	(図 21)
springcolormap	(図 22)
summercolormap	
whitecolormap	
wintercolormap	

— 以上 —

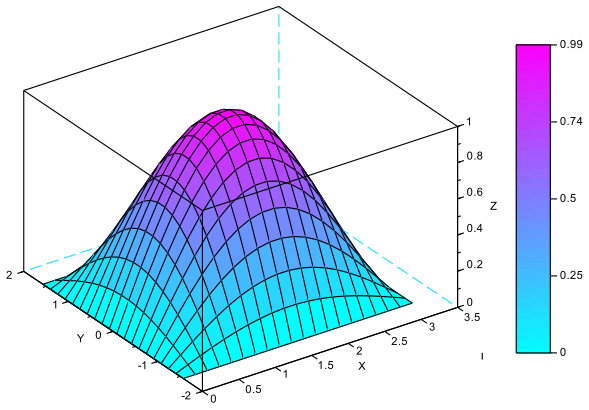

図 19

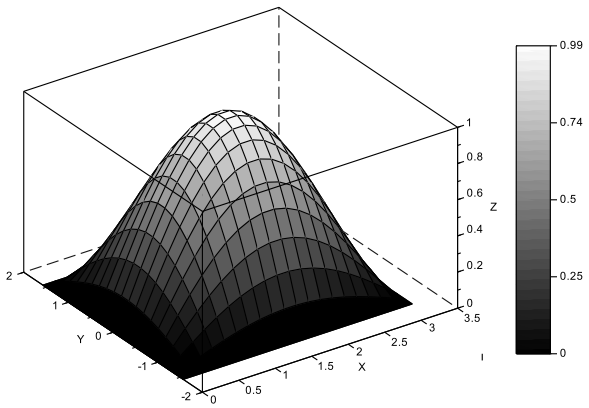

図 20

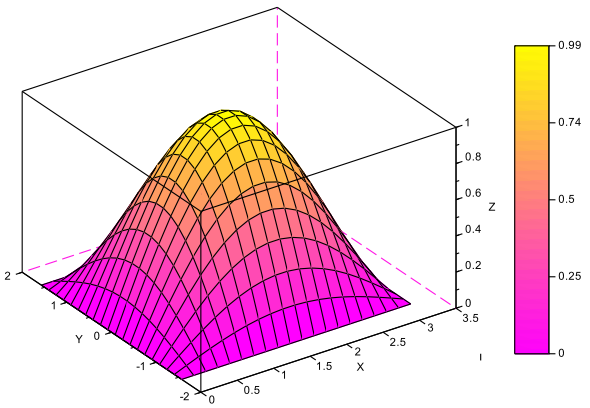

図 22

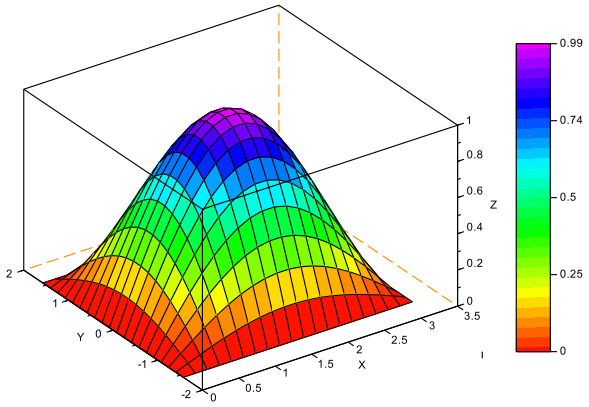

図 21