情報通信基礎1 Octave資料

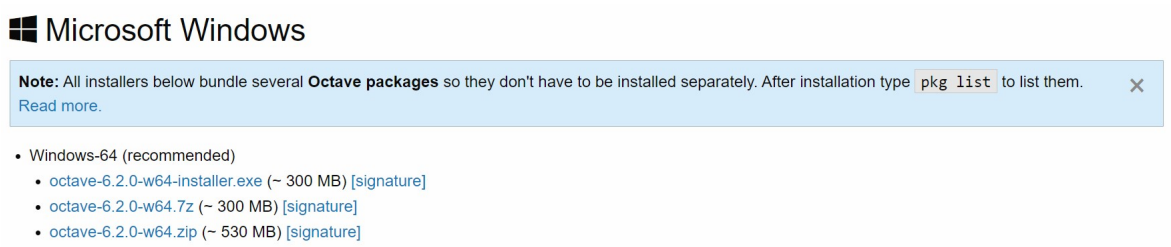
Author: Takuya Fujihashi
Date: 9 June, 2021
16 June, 2021 (改訂 1 版)

1　はじめに

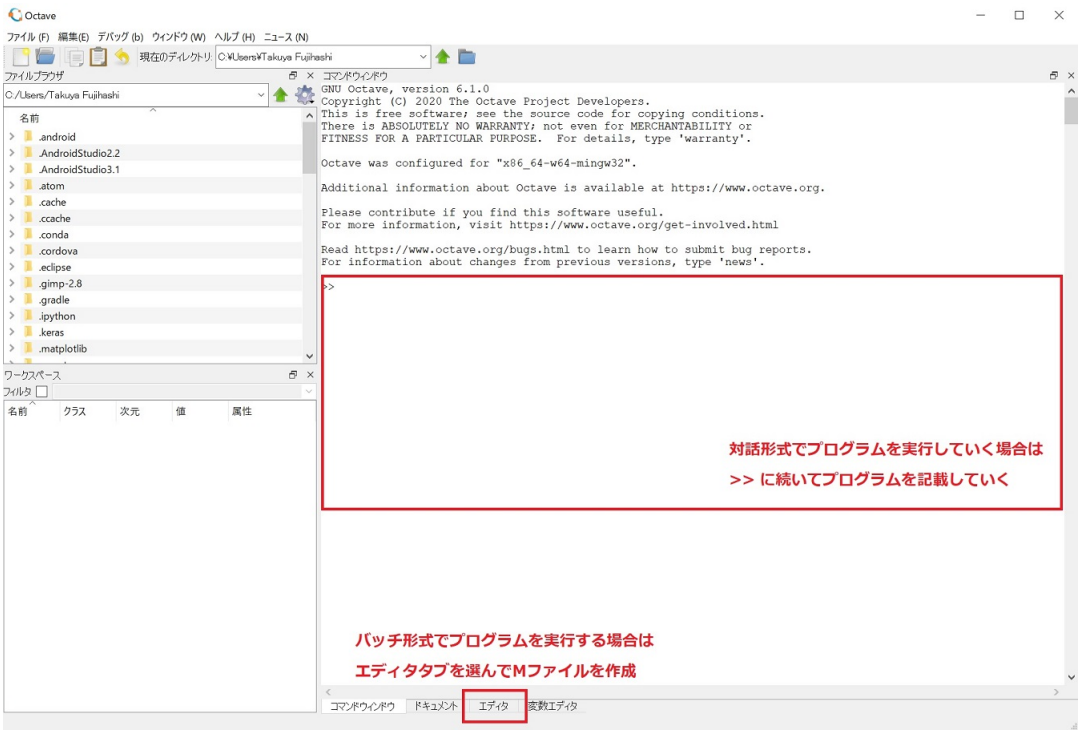
本資料は Octave を使用した情報通信基礎 1 の演習を実施するにあたって，その基礎知識となる Octave のインストール，プログラムの実行方法，Octave の基本文法について紹介するものである．GNU Octave は主に数値解析を目的とした高レベルプログラミング言語である．John W. Eaton によって 1992 年頃から開発が進められ，現在はバージョン 6.2.0 が最新となっている．Octave は線形ならびに非線形問題を数値的に解くためのコマンドライン·インタフェースを提供する．プログラミングで必須とされる基本文法（ほぼ Matlab 互換）を持っており，対話形式とバッチ形式の両方で使える．なお，Matlab は MathWorks 社が開発している技術計算言語で，多様な分野で主に研究用途で用いられているプログラミング言語である．Octave は Matlab と同様に，ベクトル，行列，複素数の取り扱いが容易で，行列式や固有値に関する計算，線形および非線形方程式の求解，多項式演算，微分方程式の求解，グラフ表示機能を利用できる．

2　Octave のインストール (Windows 10 の場合)

1. ウェブページを開く `https://www.gnu.org/software/octave/index`
2. Download ページ下部から最新の 64 ビット版インストーラを選ぶ



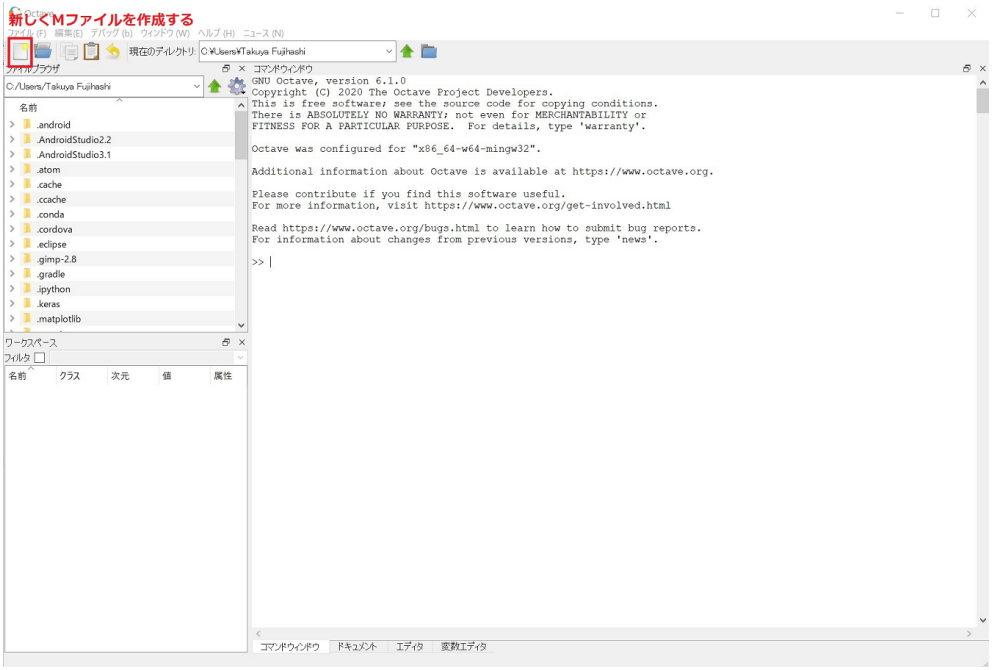
3. ダウンロードした .exe ファイルを実行 (ファイル名は octave-6.2.0-w64-installer.exe) して Octave をインストール
4. Octave を起動する．
5. 下図のように Octave のウインドウが開く．赤枠内の >> に続いて各種命令を入力することで対話形式でプログラムを実行できる．



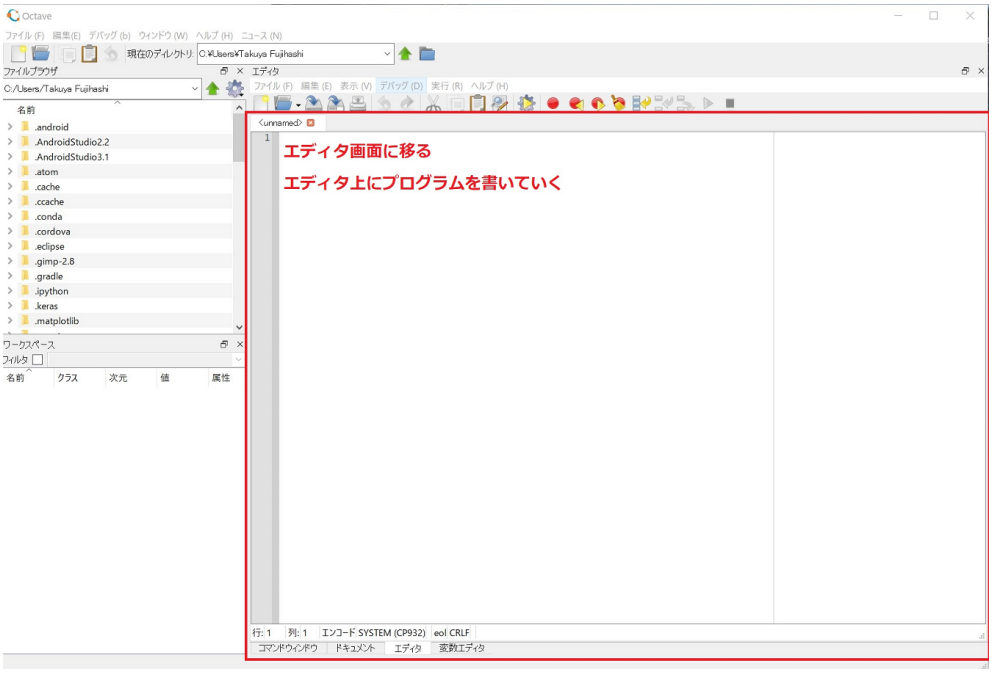
6. バッチ形式でプログラムを実行する場合はプログラムをエディタで作成する（M ファイルと呼ばれる）．M ファイルは拡張子 .m をつけて「ファイル名.m」として作業ディレクトリに保存する．Octave から M ファイルを実行するためには「 >> ファイル名 」 と入力して実行する．また，実行中にプログラムを停止させたいときは Ctrl-c（Ctrl キーを押しながら c を押す）と入力することで中断できる．
7. Octave を終了するには `quit` または `exit` を入力する．

3　M ファイルの作成方法とバッチ形式での実行方法

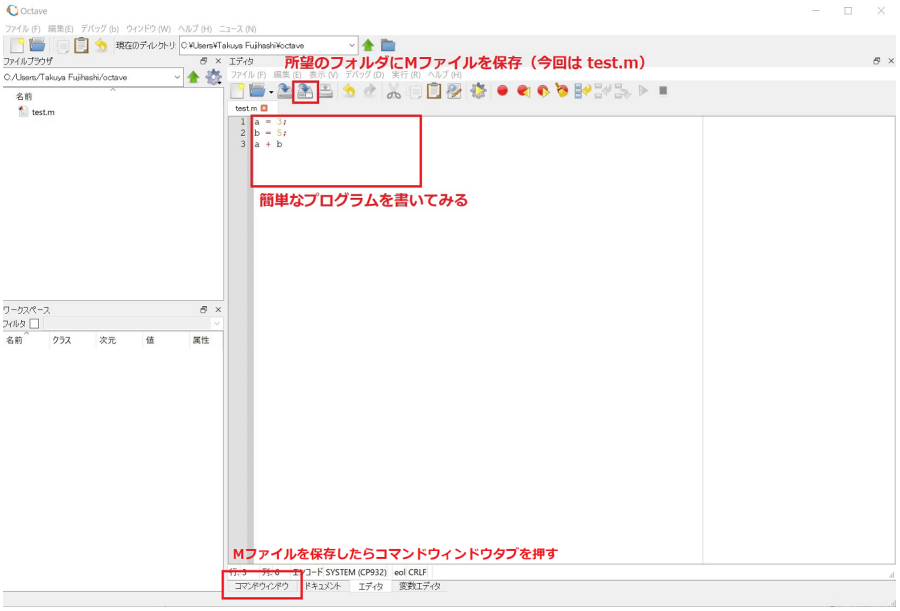
本節ではバッチ形式でプログラムを実行するための手順を示す．まず，Octave を起動して画面上部にある「新規のスクリプト」ボタンを押す.「新規のスクリプト」ボタンを押すと，空白のエディタ画面に移行する．



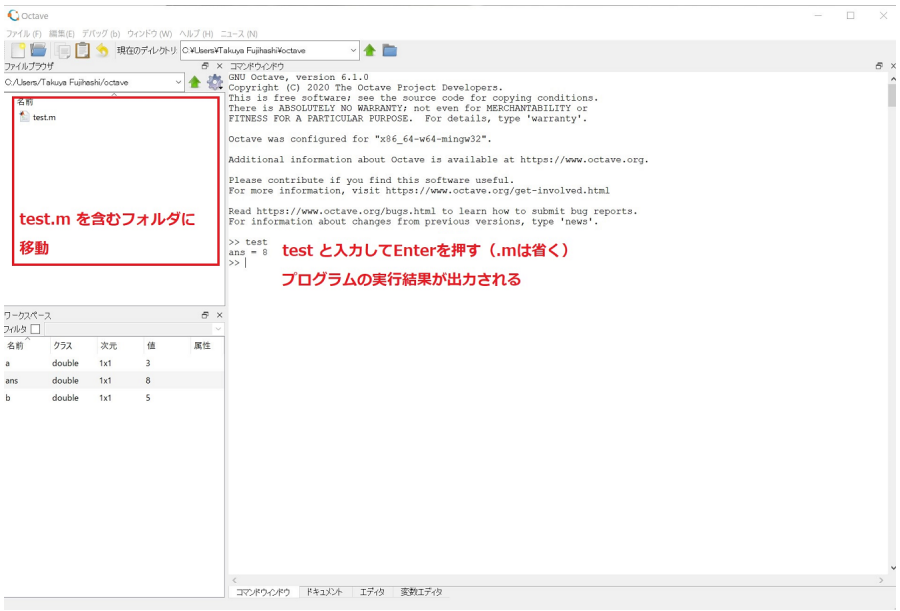
エディタ画面に移行したら，エディタ上に所望のプログラムを書いていく．Octave の文法については本節以降でその一部を紹介しているので参考にすること．また，様々な Web ページで本資料では紹介していない文法についても紹介されているので，適宜参考にすること．



本節では，例えば，a=3，b=5，a+b を出力するプログラムを書いたことを想定する．プログラムを作成したら「ファイルを別名で保存」ボタンを押して，M ファイルを適当なフォルダに保存する．M ファイルを実行するためには，Octave を操作して M ファイルを保存したフォルダに移動する必要があるため，M ファイルを保存したフォルダは必ず覚えておくこと．



M ファイルを保存したら，画面左側にあるエクスプローラー画面とマウス操作を利用して，M ファイルを保存したフォルダへと移動する．フォルダへ移動したら，画面下部のコマンドウィンドウタブを押して，>> が表示された画面へと移動する．その後，>> に続いて “test” と入力することで M ファイル内のプログラムをバッチ形式で実行する．このとき，拡張子に相当する “.m” は省略する．バッチ形式で実行すると計算結果である 8 が画面上に出力される．



4　Octave の基本的な文法

皆さんはこれまで C 言語を通してプログラミングを学んできたかと思います．C 言語と比べると Octave（あるいは互換性を持つ Matlab）はプログラムを簡単に書けるという特徴があります．具体的には，例えば C 言語で必要な変数宣言やメモリ管理といった記述が Octave では必要ありません．C 言語ではいろいろなことを考えてプログラムを書かなければならないですし，プログラムがとても長くなるという特徴があります．その点，Octave はプログラムを書くためにエネルギーをあまり必要とせず，実現したい技術そのものに注力しやすくなります．また，変数宣言等が必要ないため，完成するプログラムもとても短くなります．

本節では，Octave における文法についてその一部を紹介しています．C 言語と類似するものもありますが，記法が変わっているので注意してください．また，C 言語では利用できない機能がいくつも含まれています．

4.1　コメントアウト

```
# #以降はコメントとしてみなされます
% %も#と同様にコメントとしてみなされます
```

4.2　結果出力

octave は基本的に計算結果を出力する．結果を見るために printf 等の関数は必要ない．「;」を行末につけると結果が出力されなくなる．デバックのときは「;」をとることで所望の変数の中身を確認することができる．

```
>> A = [1 2 3;4 5 6]
# => A =
#    1   2   3
#    4   5   6
>> A = [1 2 3;4 5 6];
# 出力されない
```

書式付で計算結果を出力するためには C 言語と同様に printf 関数を用いることができる．

```
>> printf("hello, Octave\n");
# hello, Octave
>> x = pi;
>> printf("variable x = %f\n",x);
# x = 3.141593
>> y = 1 + 2i;
>> printf("complex y = %f + %fi\n",real(y),imag(y));
# complex y = 1.000000 + 2.000000i
```

4.3　多量のデータ出力時の制御（必要な人だけ）

Octave では変数に含まれる値や計算結果を出力するとき，デフォルトでは全ての結果が流れるように表示されます．変数に多くの値が含まれている場合，どのような値が含まれているかを確認することは困難です．値が less 出力されて数行ずつ結果を確認したい場合は `more on` と入力します．

```
>> more on # 数行ずつ結果を出力
# 元に戻したいとき
>> more off # 一度に全ての結果を出力
```

4.4 for 文による繰り返し

C 言語と同様に，for 文による繰り返しが利用できる．C 言語とは異なり，endfor までの範囲が繰り返しの範囲となる．

```
>> for j=1:5
>>  j
>> endfor
# =>
# j =  1
# j =  2
# j =  3
# j =  4
# j =  5
```

4.5 if 文による条件分岐

C 言語と同様に，Octave でも if 文による条件分岐が利用できる．

```
>>  if(i == 0 && j == 0)
>>    k
>>  elseif i = 1
>>    j
>>  else
>>    i
>>  endif
```

4.6 数値の取り扱い

変数に数値を代入する場合は次のとおりとなる．C 言語のように int, float などの宣言は必要ない．

```
>> a = 1
# a = 1
>> b = 1.23
# b = 1.2300
>> c = 3;
# ; がついているため何も出力されない
```

変数名として，英数字と _（アンダースコア）が使用できる．英字の大文字と小文字は区別され，先頭文字に数字は使えない．また，変数名の長さは無制限である．

複素数を扱うこともできる．虚数単位 $\sqrt{-1}$ として記号 i，j，I，J　が使用できる．なお，記号 i，j，I，J がすでに変数として使用されている場合は変数の定義が優先されるため，注意すること．虚数単位と使わず，complex 関数を使用することで複素数を扱うこともできる．

```
>> y = 1+2I
# y = 1 + 2i
>> z = I
# z = 0 + 1i
>> z*z
# ans = -1
>> y = complex(1, 3) # この表現も可能
# y =  1 + 3i
```

4.7 行列の定義

行列を取り扱う場合は次のように入力する．行ごとに入力し，; (セミコロン) で次の行となる．Octave では配列を定義する必要がなく，必要な数だけ自動的に確保される．

```
# 行ベクトル
>> A = [1 2 3]
# =>
# A =
#    1   2   3

# 列ベクトル
>> A = [1;2;3]
# =>
# A =
#    1
#    2
#    3

# 行列
>> A = [1 2 3;4 5 6;7 8 9]
# =>
# A =
#    1   2   3
#    4   5   6
#    7   8   9

# 転置行列
>> A.'
# =>
# A.' =
#    1   4   7
#    2   5   8
#    3   6   9

# 行列のベクトル化
>> A(:)
# =>
# A(:) =
#    1
#    4
#    7
#    2
#    5
#    8
#    3
#    6
#    9
```

4.8 行列内の要素へのアクセス

行列から特定の要素あるいは特定の行・列を取り出したいときは次のように入力する．

```
>> A = [1 2 3;4 5 6;7 8 9]
>> A(2, 3)
# 2行3列目
# =>
# ans = 6

>> A([1, 3], [2, 3])
# 1, 3行目で2, 3列目
# =>
# ans =
#     2    3
#     8    9

>> A(2, :)
# 「:」は全部を表す 2行目の全部
# =>
# ans =
#      4    5    6
```

4.9 行列への代入・削除

行列の要素に対する数値の代入あるいは行列の要素の削除については次のように入力する．

```
A = [1 2 3;4 5 6;7 8 9]
A(1, 2) = 1
# 1行2列を1にする
# =>
# A =
#     1    1    3
#     4    5    6
#     7    8    9

A(:, 2) = 5
# 2列のすべてを5にする
# =>
# A =
#     1    5    3
#     4    5    6
#     7    5    9

A(3, :) = []
# 3行目をすべて削除(行の削除)
# =>
# A =
#     1    5    3
#     4    5    6
```

4.10 行列の結合

行列 A に行列 B を追加して新しい行列 C を作る場合は，次のように入力する．

- $C = [AB]$ の場合

```
>> C = [A B]
```

- $C = \begin{bmatrix} A \\ B \end{bmatrix}$ の場合

```
>> C = [A ; B]
```

4.11 行列の和・差・積・剰余・転置

行列 A と行列 B の和，差は次のように入力する．

```
>> X=A+B
>> Y=A-B
```

また，行列の積は次のとおりとなる．このとき，行列 A および行列 B の次元は各演算が定義できるように整合性を持つ必要がある．

```
>> X = A*B
```

しかしながら，一方がスカラーの場合は例外となり，行列 A，スカラー t に対して次のように入力することで A のすべての要素に t が掛けられる．

```
>> X = A*t
```

行列 A のスカラー t に対する剰余は次のように求められる．

```
>> X = mod(A, t)
```

行列 A の転置行列および共役転置行列は次のように入力する．

```
>> X = A.'  # 転置行列
>> X = A' # 共役転置行列
```

4.12 変数への入力

プログラムの実行途中に，ある変数に対して所望の数値や行列を代入したいときには input を利用する．

```
>> a = input('Enter value a  ') # 数値を代入する場合
Enter value a 5 #（入力受付状態になるので数値5を入力してEnterを押す）
# a = 5
>> A = input('Enter matrix A  ') # 行列を代入する場合
Enter matrix A [1 2 3; 4 5 6] #（入力受付状態になるので行列 [1 2 3; 4 5 6] を入力して
    Enterを押す）
#A =
#    1    2    3
#    4    5    6
```

4.13 変数の確認および削除

すでに作成されている変数の確認をしたいときは who あるいは whos を，作成されている変数を削除したいときは clear を入力する．なお，clear と入力すると全変数が削除され，clear 変数名 と入力すると指定した変数名が削除される．

```
>> a = 1;
>> b = [ 1 2 3; 4 5 6];
>> whos # 作成済の変数を確認
#.Variables visible from the current scope:
#
#variables in scope: top scope
#
#   Attr Name        Size                     Bytes  Class
#   ==== ====        ====                     =====  =====
#        a           1x1                          8  double
#        b           2x3                         48  double
# Total is 7 elements using 56 bytes
>> clear b # 変数bの削除
>> whos # 変数bが表示されなくなる
# Variables visible from the current scope:
#
#variables in scope: top scope
#
#   Attr Name        Size                     Bytes  Class
#   ==== ====        ====                     =====  =====
#        a           1x1                          8  double
# Total is 1 element using 8 bytes
>> clear # 全変数の削除
>> whos
# 全変数を削除したため，何も表示されない
```

4.14 ベクトルに対する関数

Octave ではベクトルに対する関数が用意されている．例えば，ベクトル内の要素の総和・平均・標準偏差を求める場合は Octave が提供している sum 関数，mean，std 関数を利用するとよい．

```
>> v1 = [1, 2, 3];
# 総和
>> sum(v1);
# 平均
>> mean(v1);
# 標準偏差
>> std(v1);
```

4.15 行列関数

Octave には行列を対象とした関数も多数用意されている．例えば，正則行列 A の逆行列は inv 関数を利用することで求めることができる．

```
>> X = inv(A)
```

その他にも下記のような行列関数を使用することができる．行列関数の使用方法については https://octave.org/doc/v6.2.0/ を参照するとよい．

Tabela 1: 代表的な行列関数

関数名	役割
norm	行列またはベクトルノルム
rank	行列のランク（階数）
det	行列式
inv	逆行列
eig	固有値と固有ベクトル

4.16 単位行列・零行列・対角行列

n 次単位行列 $\boldsymbol{X} = \boldsymbol{I}(n \times n)$ を定義する場合は次のように入力する．

```
>> X = eye(n,n)
```

また，零行列 $\mathbf{0}$ $(n \times m)$，すべての要素が 1 の行列 $\mathbf{1}$ $(n \times m)$ を定義する場合は次のように入力する．

```
>> zeros(n,m)
>> ones(n,m)
```

対角行列を定義する場合は次のように入力する．

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \tag{1}$$

```
>> v=[1 9 4 4 2];
>> D=diag(v)
```

4.17 行列のサイズ

プログラム中で行列 A が与えられていて，このサイズ（行と列の数）を知りたいときには次のように入力する．ここで，A が 3×4 行列であると仮定する．

```
>> d = size(A)
# d =
# 3 4
```

このとき d(1)=3，d(2)=4 を表している．

A と同じサイズの零行列 Z を作る場合は次のように定義する

```
>> Z=zeros(size(A))
```

4.18　数学関数

Octave では sin, cos などの数学関数が一通り用意されている．また，このような関数はベクトル計算もできる．例えば，$\sin(\boldsymbol{t})$ をベクトル $\boldsymbol{t}$ に対して計算する場合は次のように記述できる．

```
>> t = 0:0.1:10;
>> y = sin(t);
#  Columns 1 through 27:
#         0    0.0998    0.1987    0.2955    0.3894    0.4794    0.5646    0.6442    0.7174
    0.7833    0.8415    0.8912    0.9320    0.9636    0.9854    0.9975    0.9996    0.9917
    0.9738    0.9463    0.9093    0.8632    0.8085    0.7457    0.6755    0.5985    0.5155
#
# Columns 28 through 54:
#
#    0.4274    0.3350    0.2392    0.1411    0.0416   -0.0584   -0.1577   -0.2555   -0.3508   -
    0.4425   -0.5298   -0.6119   -0.6878   -0.7568   -0.8183   -0.8716   -0.9162   -0.9516   -
    0.9775   -0.9937   -0.9999   -0.9962   -0.9825   -0.9589   -0.9258   -0.8835   -0.8323
#
# Columns 55 through 81:
#
#   -0.7728   -0.7055   -0.6313   -0.5507   -0.4646   -0.3739   -0.2794   -0.1822   -0.0831
    0.0168    0.1165    0.2151    0.3115    0.4048    0.4941    0.5784    0.6570    0.7290
    0.7937    0.8504    0.8987    0.9380    0.9679    0.9882    0.9985    0.9989    0.9894
#
# Columns 82 through 101:
#
#    0.9699    0.9407    0.9022    0.8546    0.7985    0.7344    0.6630    0.5849    0.5010
    0.4121    0.3191    0.2229    0.1245    0.0248   -0.0752   -0.1743   -0.2718   -0.3665   -
    0.4575   -0.5440
```

その他，Octave では対数関数，指数関数，絶対値関数など数学関数を用いることができる．

Tabela 2: 代表的な数学関数

関数名	役割
sin(x)	$\sin x$
cos(x)	$\cos x$
tan(x)	$\tan x$
exp(x)	e^x
log(x)	$\log x$
log10(x)	$\log_1 0x$
sqrt(x)	$\sqrt{x}$
abs(x)	$\lvert x \rvert$
real(x)	複素数の実部
imag(x)	複素数の虚部

4.19　乱数

Octave では様々な分布にしたがう乱数を生成することもできます．例えば，0〜1 までの範囲において一様分布にしたがう乱数は rand，一様分布にしたがう整数の乱数は randi，標準正規分布にしたがう乱数は randn，指数分布にしたがう乱数は rande からそれぞれ生成することができる．

```
>> x = rand(1, 10) # [0, 1] までの乱数を生成．1つ目の引数は行方向，2つ目の引数は列方
    向の要素数を示す
#x =
#    0.071149    0.226326    0.843703    0.479242    0.318660    0.463096    0.548496
    0.340929    0.592355    0.482719
>> x = randi([1, 10], 1, 10) # 1つ目の要素は取りうる整数の範囲，2つ目の引数は行方向
    ，3つ目の引数は列方向の要素数を示す
#x =
```

```
#    10     5     1     5     2     8     9     6     9     9
>> x = randn(1,10) # 標準正規分布(平均0, 分散1)にしたがう乱数を生成. 1つ目の引数は行
    方向, 2つ目の引数は列方向の要素数を示す
#x =
#   -0.6030   -0.1871    0.5012   -0.8193   -1.2558    1.1476   -0.3326   -1.6135    0.2864
    0.9332
```

4.20 自作関数

関数を自作するときは関数ごとに別々のファイルを用意する．function 構文は次のとおりとなる．

function [戻り値] = 関数名（引数）

ファイル名は 関数名.m とする必要がある．メインプログラムと同じディレクトリにファイルが存在していれば関数名を入力することで関数を実行できる．関数名は変数名と同様，英字で始め，他に数字やアンダーバーの記号を使用できる．

関数に対する返り値と引数はそれぞれ複数を設定することができる．引数（複数の場合コンマで区切る）を関数に渡して，返り値（複数の場合コンマで区切る）を関数から得ることができる．引数と返り値以外の関数中の変数はメインプログラムや他の関数プログラムに対して独立している（ローカル変数と考えてもらって OK）．もちろん，引数や戻り値は行列やベクトルでもよい．

返り値は C 言語の場合，return を使って指定するが，Octave では返り値として宣言した変数に代入することで返り値を設定する．Octave でも同様に return を利用できるが，関数の呼び出し元に戻るだけで，return 以前に返り値に対して適切な値が設定されているか注意すること．例えば，n 個のランダムなデータが与えられたとき，その平均と分散を出力する関数は次のように作成できる．

ファイル名: mean_var.m

```
# Mファイルの中身
function [a, b] = mean_var (x)
# n個のランダムなデータを含む x から平均 a, 分散 b を取得する関数
n = length (x) # xの要素数
a = sum(x)/n; # 平均の取得
b = sum((x-a).^2)/n; # 分散の取得
endfunction # 省略可能
```

作成した mean_var 関数を呼び出すプログラムの一例を下記に示す．

```
>> x = rand(1, 10) # [0, 1] までの10個のランダムな値を取得する
# x =
#    0.160747    0.635412    0.766828    0.037829    0.969216    0.354765    0.712706
    0.811218    0.558583    0.834992
>> [heikin, bunsan] = mean_val(x) # heikin に平均値を, bunsan に分散を出力する
# heikin = 0.5842
# bunsan = 0.084629
```

4.21　グラフへのデータプロット

ここでは，以下のように作成した sin, cos のデータを使用して下図のグラフを作成することを考える．

```
>> t = [1:0.01:2];     # x軸の数値作成
>> y1 = cos(2*pi*4*t);  # x軸に対応するcosデータ作成
>> y2 = sin(2*pi*4*t);  # x軸に対応するsinデータ作成
```

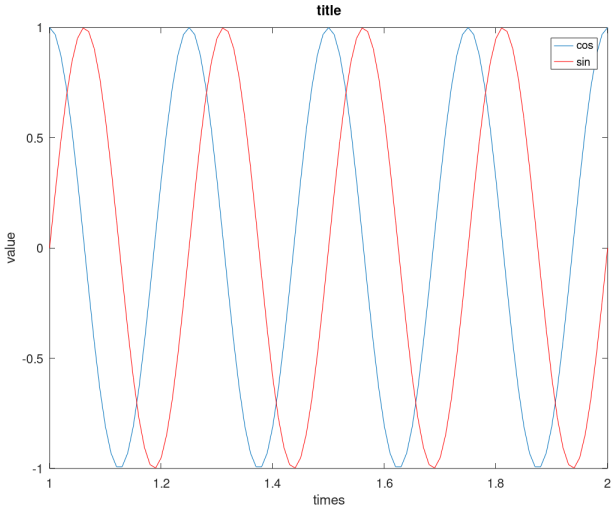


Figura 1: 対象のデータから作成したいグラフ

このとき，plot 関数に x 軸および y 軸を渡すことで小さなウィンドウが開くとともにグラフをプロットできる．plot 関数の引数は (x 軸のデータ，y 軸のデータ，グラフの色等のオプション) で構成される．通常，複数のデータを plot で連続で処理しても，グラフは上書きされ一つのデータしか表示されない．hold on と入力することで，複数の plot 関数を用いてデータを重ねてグラフに表示することが可能になる．また，plot に対しては'r' のような引数を渡すことでグラフの色を指定することも可能となる．plot 関数を一度使って複数のデータをグラフに表示したいときは plot(1 番目の x 軸のデータ，1 番目の y 軸のデータ，1 番目のデータに対するオプション，2 番目の x 軸のデータ，2 番目の y 軸のデータ，2 番目のデータに対するオプション，...，n 番目の x 軸のデータ，n 番目の y 軸のデータ，n 番目のデータに対するオプション) と入力することもできる．なお，作成したグラフを一旦クリアして異なるグラフを描画したいときには clf を利用する．また，新たなウィンドウを開いて別のグラフを描画したいときには figure を利用する．

```
>> plot(t,y1) # cosデータプロット
>> hold on  # グラフをそのままに
>> plot(t,y2,'r') # sinデータを重ねてプロット．線の色は赤色に
# または
>> clf # グラフをクリア
>> plot(t,y1,t,y2,'r')
```

各軸ラベルは xlabel（x 軸），ylabel（y 軸），凡例は legend（複数の線があるときはコンマでそれぞれの凡例を設定），タイトルは title を用いることでグラフに追加できる．

```
>> xlabel('times')  # x軸ラベルを追加
>> ylabel('value')  # y軸ラベルを追加
>> legend('cos','sin')  # 凡例を追加
>> title('title') # タイトルを追加
```

また，新たなウィンドウを開いて別のグラフを描画したいときには figure を利用する．

```
>> figure # 新たなウィンドウが開く
>> plot(t,y1,t,y2,'g') # 2番目のデータの線が緑色に変わる
```

また，x 軸あるいは y 軸が対数スケールとなる片対数グラフ，x 軸および y 軸双方が対数スケールとなる両対数グラフはそれぞれ semilogx 関数，semilogy 関数，loglog 関数を利用することでプロットできる．このとき，各関数に対する引数は plot 関数と同様に，(1 番目の x 軸のデータ，1 番目の y 軸のデータ，1 番目のデータに対するオプション，2 番目の x 軸のデータ，2 番目の y 軸のデータ，2 番目のデータに対するオプション，...，n 番目の x 軸のデータ，n 番目の y 軸のデータ，n 番目のデータに対するオプション) を利用できる．

```
# x軸が対数スケールとなる片対数グラフ
>> semilogx(t,y1,t,y2,'r')
# y軸が対数スケールとなる片対数グラフ
>> semilogy(t,y1,t,y2,'r')
# x軸およびy軸が対数スケールとなる両対数グラフ
>> loglog(t,y1,t,y2,'r')
```