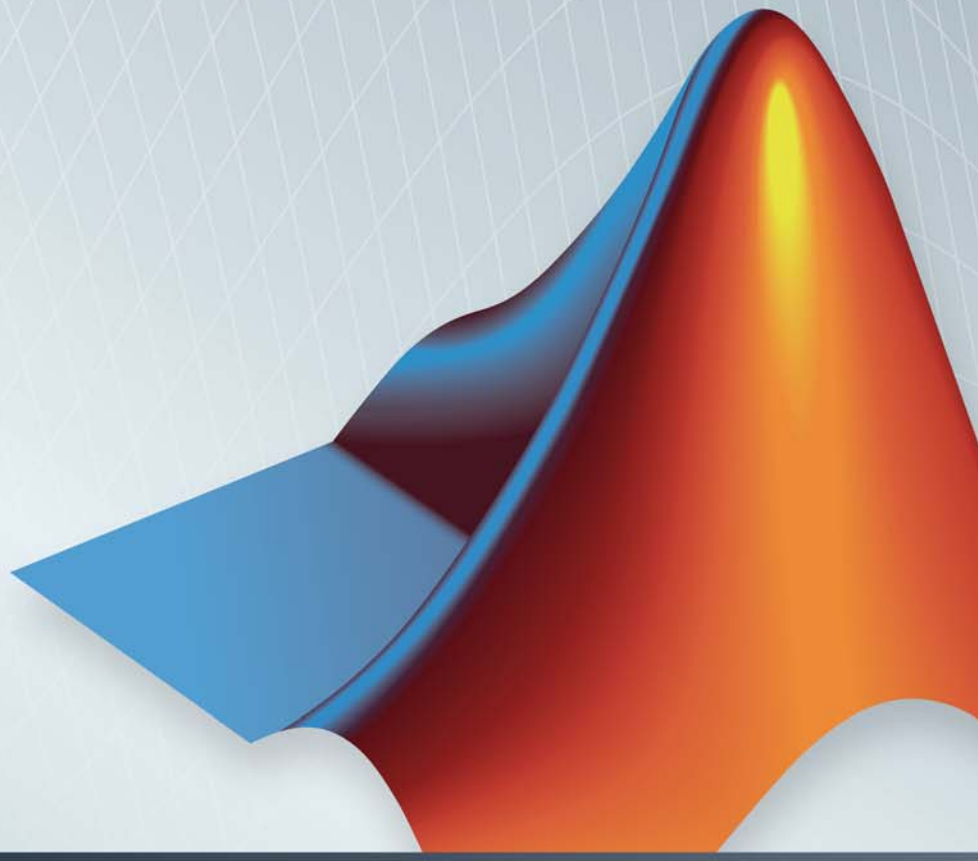
MATLAB®

入門

R2013a



MATLAB®

MathWorks®

MathWorks へのお問い合わせ

www.mathworks.co.jp	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.co.jp/contact_TS.html	テクニカル サポート

support@mathworks.com	製品の機能拡張の提案 (英語)
support@mathworks.com	バグの報告 (英語)
doc@mathworks.co.jp	ドキュメンテーションに関する問題点の報告
service@mathworks.co.jp	注文状況、ライセンス更新、パスコード
service@mathworks.co.jp	営業部、製品価格、一般情報

TEL: 03-6367-6700 (大代表)

FAX: 03-6367-6710

The MathWorks GK
〒107-0052 東京都港区赤坂4丁目15-1
赤坂ガーデンシティ

世界各地のオフィスへのお問い合わせ情報は、MathWorks Web サイトをご覧ください。

MATLAB® 入門

改訂履歴

1996 年 12 月	初版	MATLAB 5 版
1997 年 5 月	第 2 版	MATLAB 5.1 版
1998 年 9 月	第 3 版	MATLAB 5.3 版
2000 年 9 月	第 4 版	MATLAB 6 改訂版 (Release 12)
2001 年 6 月	オンライン専用	MATLAB 6.1 改訂版 (Release 12.1)
2002 年 7 月	オンライン専用	MATLAB 6.5 改訂版 (Release 13)
2002 年 8 月	第 5 版	MATLAB 6.5 改訂版
2004 年 6 月	第 6 版	MATLAB 7.0 改訂版 (Release 14)
2004 年 10 月	オンライン専用	MATLAB 7.0.1 改訂版 (Release 14SP1)
2005 年 3 月	オンライン専用	MATLAB 7.0.4 改訂版 (Release 14SP2)
2005 年 6 月	第 7 版	MATLAB 7.0.4 マイナー改訂版 (Release 14SP2)
2005 年 9 月	オンライン専用	MATLAB 7.1 マイナー改訂版 (Release 14SP3)
2006 年 3 月	オンライン専用	MATLAB 7.2 マイナー改訂版 (Release 2006a)
2006 年 9 月	第 8 版	MATLAB 7.3 マイナー改訂版 (Release 2006b)
2007 年 3 月	第 9 版	MATLAB 7.4 マイナー改訂版 (Release 2007a)
2007 年 9 月	第 10 版	MATLAB 7.5 マイナー改訂版 (Release 2007b)
2008 年 3 月	第 11 版	MATLAB 7.6 マイナー改訂版 (Release 2008a)
2008 年 10 月	第 12 版	MATLAB 7.7 マイナー改訂版 (Release 2008b)
2009 年 3 月	第 13 版	MATLAB 7.8 マイナー改訂版 (Release 2009a)
2009 年 9 月	第 14 版	MATLAB 7.9 マイナー改訂版 (Release 2009b)
2010 年 3 月	第 15 版	MATLAB 7.10 マイナー改訂版 (Release 2010a)
2010 年 9 月	第 16 版	MATLAB 7.11 改訂版 (R2010b)
2011 年 4 月	オンライン専用	MATLAB 7.12 改訂版 (R2011a)
2011 年 9 月	第 17 版	MATLAB 7.13 改訂版 (R2011b)
2012 年 3 月	第 18 刷	Version 7.14 改訂版 (R2012a)(『MATLAB ご利用の前に』から名称変更)
2012 年 9 月	第 19 刷	Version 8.0 改訂版 (R2012b)
2013 年 3 月	第 20 版	Version 8.1 改訂版 (R2013a)

目次

1 クイック スタート

2 言語の基礎

3 数学

4 グラフィックス

5 プログラミング

1

クイック スタート

製品の説明

技術計算言語

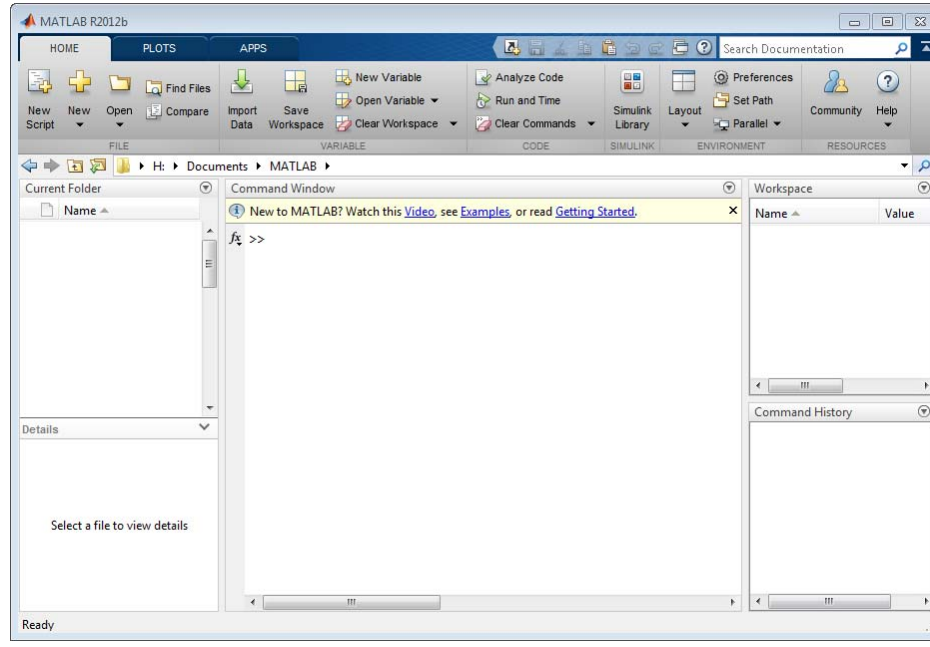
MATLAB® は、数値計算、可視化およびプログラミングのための高水準言語および対話型環境です。MATLAB を使用して、データ解析、アルゴリズム開発、各種モデルおよびアプリケーションの作成を行うことができます。言語、ツール、組み込み関数により、多様なアプローチの調査が可能となり、スプレッドシートや C/C++、Java® などの従来のプログラミング言語よりもすばやく解を求めることができます。MATLAB は、信号処理、通信、画像処理、ビデオ処理、制御システム、テストおよび測定、金融工学、情報生命科学など、幅広い分野での利用が可能です。技術計算言語 MATLAB は、世界中で 100 万人を超える業界や学界の技術者や科学者に利用されています。

主な機能

- 数値計算、可視化、アプリケーション開発のための高水準言語
- 反復探索や設計、問題解決を行う対話型環境
- 線形代数、統計、フーリエ解析、フィルター処理、最適化、数値積分、常微分方程式求解の数学関数
- データ可視化用組み込みグラフィックスおよびカスタム プロット作成ツール
- コードの質や保守容易性を向上し、最大の性能を引き出すための開発ツール
- カスタム グラフィカル インターフェイスによるアプリケーション構築ツール
- MATLAB ベースのアルゴリズムを C、Java、.NET、Microsoft® Excel® などの外部アプリケーションや言語と統合する関数

デスクトップの基礎

MATLAB を起動すると、既定のレイアウトでデスクトップが表示されます。



デスクトップには以下のパネルが表示されます。

- [現在のフォルダー] – 使用するファイルにアクセスします。
- [コマンド ウィンドウ] – プロンプト (>>) が指示するコマンド ラインにコマンドを入力します。
- [ワークスペース] – 作成したデータやファイルからインポートしたデータを探索します。
- [コマンド履歴] – コマンド ラインに入力したコマンドを表示または再実行します。

MATLAB の作業では、変数の作成や関数の呼び出しを行うコマンドを実行します。たとえば、次のステートメントをコマンド ラインに入力して a という名前の変数を作成します。

```
a = 1
```

MATLAB によって変数 a がワークスペースに追加され、コマンド ウィンドウに結果が表示されます。

```
a =

     1
```

さらに変数をいくつか作成します。

```
b = 2

b =

     2

c = a + b

c =

     3

d = cos(a)

d =

    0.5403
```

出力変数が設定されていなければ、MATLAB は変数 ans (answer の省略形) を使って計算結果を保存します。

```
sin(a)

ans =

    0.8415
```

ステートメントの最後にセミコロンを付けると、MATLAB は計算を実行しますが、コマンド ウィンドウへの出力表示は行いません。

```
e = a*b;
```

上方向キー（↑）や下方向キー（↓）を押すと、以前のコマンドが再度呼び出せます。空のコマンド ライン上またはコマンドの最初の数文字の入力後に方向キーを押します。たとえば、コマンド b = 2 を呼び出すには、b と入力してから上方向キーを押します。

行列と配列

この節の内容...
配列の作成 (p. 1-6)
行列と配列の演算 (p. 1-7)
連結 (p. 1-9)
複素数 (p. 1-10)

MATLAB は "matrix laboratory" を短縮した名称です。他のほとんどのプログラミング言語では、数値は一度に 1 つずつ扱いますが、MATLAB は基本的に行列や配列全体を演算するように設計されています。

すべての MATLAB 変数は多次元 "配列" であり、これはデータの型とは無関係です。"行列" とは線形代数で多用される 2 次元配列です。

配列の作成

4 つの要素を含む単一行の配列を作成するには、要素をコンマ (,) とスペースのいずれかで区切ります。

```
a = [1 2 3 4]
```

このコードは以下を返します。

```
a =

     1     2     3     4
```

このタイプの配列が "行ベクトル" です。

複数の行をもつ行列を作成するには、行をセミコロンで区切ります。

```
a = [1 2 3; 4 5 6; 7 8 10]
```

```
a =

     1     2     3
     4     5     6
```

```
     7     8    10
```

行列を作成するもうひとつの方法は、ones、zeros、rand などの関数を使用することです。たとえば、ゼロから構成される 5 行 1 列の列ベクトルを作成します。

```
z = zeros(5,1)

z =

     0
     0
     0
     0
     0
```

行列と配列の演算

MATLAB では、単一の算術演算子または関数を使用して行列中のすべての値を処理することができます。

```
a + 10

ans =

    11    12    13
    14    15    16
    17    18    20

sin(a)

ans =

    0.8415    0.9093    0.1411
   -0.7568   -0.9589   -0.2794
    0.6570    0.9894   -0.5440
```

行列を転置するには、一重引用符 (') を使用します。

```
a'

ans =
```

```
     1     4     7
     2     5     8
     3     6    10
```

* 演算子を使用して、各行と各列の内積計算による標準の行列乗算を実行することができます。たとえば、行列にその逆行列を乗じると単位行列が返されることを確認するには、次のようにします。

```
p = a*inv(a)

p =

    1.0000         0   -0.0000
         0    1.0000         0
         0         0    1.0000
```

p が整数値行列ではないという点に注意してください。MATLAB では数値が浮動小数点値として保存され、算術演算では実際の値とその浮動小数点表記のわずかな誤差が影響します。format コマンドを使用して、表示される桁数を増やすことができます。

```
format long
p = a*inv(a)

p =

   1.000000000000000                   0  -0.000000000000000
                   0   1.000000000000000                   0
                   0                   0   0.999999999999998
```

短形式の表示に戻すには、次のようにします。

```
format short
```

format は数値の表示のみに影響し、MATLAB での計算方法や保存方法には影響しません。

行列の乗算ではなく要素単位の乗算を実行するには、.* 演算子を使用します。

```
p = a.*a

p =
```

```
     1     4     9
    16    25    36
    49    64   100
```

乗算、除算およびべき乗の行列演算子にはそれぞれ対応する配列演算子があり、これは要素単位の演算を行います。たとえば、a の各要素を次のように 3 乗します。

```
a.^3
```

```
ans =

           1           8          27
          64         125         216
         343         512        1000
```

連結

″連結″ とは、配列を合わせてより大きな配列にする処理です。実際、最初の配列は個々の要素を連結して作っています。大かっこ ([]) は連結演算子です。

```
A = [a,a]
```

```
A =

     1     2     3     1     2     3
     4     5     6     4     5     6
     7     8    10     7     8    10
```

コンマを使用して配列を次の配列と相互に連結することを ″水平″ 連結と呼びます。各配列の行数は同じでなければなりません。同様に、配列の列数が同じであれば、セミコロンを使用して ″垂直″ に連結することができます。

```
A = [a; a]
```

```
A =

     1     2     3
     4     5     6
     7     8    10
     1     2     3
     4     5     6
     7     8    10
```

複素数

複素数には実数部と虚数部があり、虚数単位は -1 の平方根です。

```
sqrt(-1)

ans =

        0 + 1.0000i
```

複素数の虚数部を表すには、i または j のいずれかを使用します。

```
c = [3+4i, 4+3j, -i, 10j]

c =

   3.0000 + 4.0000i   4.0000 + 3.0000i   0 - 1.0000i   0 +10.0000i
```

配列のインデックス付け

MATLAB のあらゆる変数は配列であり、多数の数値を保持できます。配列内の特定要素にアクセスするには、インデックス付けを行います。

たとえば、次のような 4 行 4 列の魔方陣 A について考えます。

```
A = magic(4)
```

```
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

配列内の特定要素を参照する方法は 2 つあります。最も一般的な方法は、次のように行と列の添字を指定することです。

```
A(4,2)
```

```
ans =
    14
```

あまり一般的ではありませんが役立つものとして、単一の添字を使用して各列を上から下へと順に辿っていく方法があります。

```
A(8)
```

```
ans =
    14
```

単一の添字を使用した配列内の特定要素の参照を "線形インデックス" と呼びます。

代入ステートメントの右側で配列外要素の参照を行うと、MATLAB はエラーを返します。

```
test = A(4,5)
```

```
Attempted to access A(4,5); index out of bounds because size(A)=[4,4].
```

ただし、代入ステートメントの左側では、現在の次元外の要素を指定することができます。配列のサイズは新しい要素に合わせて大きくなります。

```
A(4,5) = 17

A =
    16     2     3    13     0
     5    11    10     8     0
     9     7     6    12     0
     4    14    15     1    17
```

配列の複数の要素を参照するにはコロン演算子を使用します。これを使用すると、start:end の形式で範囲を指定できます。たとえば、A の最初の 3 行、第 2 列の要素を列挙するには、次のようにします。

```
A(1:3,2)

ans =
     2
    11
     7
```

開始値と終了値のない、コロンのみの場合、その次元内のすべての要素が指定されます。たとえば、A の 3 行目のすべての列を選択するには、次のようにします。

```
A(3,:)

ans =
     9     7     6    12     0
```

コロン演算子はまた、より一般的な start:step:end という形式を使用することにより、値が等間隔に並んだベクトルを作成します。

```
B = 0:10:100

B =

     0    10    20    30    40    50    60    70    80    90   100
```

start:end のように中央の step を省略すると、MATLAB では既定のステップ値 1 が使用されます。

ワークスペース変数

"ワークスペース" には、MATLAB で作成した変数や、データ ファイルや他のプログラムからインポートした変数が表示されます。たとえば、次のステートメントによってワークスペースに変数 A と変数 B が作成されます。

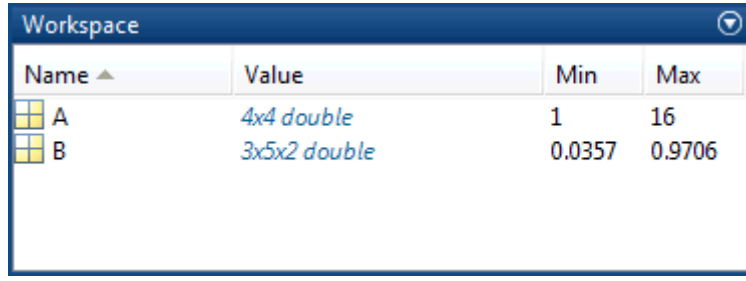
```
A = magic(4);
B = rand(3,5,2);
```

ワークスペースの内容は whos を使用して表示することができます。

```
whos
```

```
  Name        Size              Bytes  Class     Attributes

  A           4x4                 128  double
  B           3x5x2               240  double
```

変数は、デスクトップの [ワークスペース] ペインにも表示されます。



ワークスペース変数は MATLAB の終了後は保持されません。後で使用できるようにデータを save コマンドで保存してください。

```
save myfile.mat
```

これによって、ワークスペースは、.mat という拡張子をもつ圧縮ファイル (MAT ファイル) として現在の作業フォルダーに保存されます。

すべての変数をワークスペースからクリアするには、clear コマンドを使用します。

load を使用すると MAT ファイルからワークスペースにデータが復元されます。

```
load myfile.mat
```

文字列

"文字列" は、一重引用符で囲まれた任意の数の文字から成るシーケンスです。文字列は変数に代入することができます。

```
myText = 'Hello, world';
```

テキストに一重引用符が含まれている場合は、2 つの一重引用符を定義内で使用します。

```
otherText = 'You''re right'
```

```
otherText =

You're right
```

すべての MATLAB 変数と同様、myText と otherText は配列です。その "クラス" またはデータ型は char で、これは character の短縮形です。

```
whos myText
```

```
  Name        Size            Bytes  Class    Attributes

  myText      1x12               24  char
```

数値配列を連結する場合と同様に、大かっこを使用して文字列を連結することができます。

```
longText = [myText,' - ',otherText]
```

```
longText =

Hello, world - You're right
```

数値を文字列に変換するには、num2str や int2str などの関数を使用します。

```
f = 71;
c = (f-32)/1.8;
tempText = ['Temperature is ',num2str(c),'C']
```

```
tempText =
```

```
Temperature is 21.6667C
```

関数の呼び出し

MATLAB には、計算タスクを実行するための関数が数多く用意されています。関数は、他のプログラミング言語における "サブルーチン" や "メソッド" に相当します。

ワークスペースに次のような変数 A と変数 B が含まれているとします。

```
A = [1 3 5];
B = [10 6 4];
```

関数を呼び出すには、関数の入力引数をかっこで囲みます。

```
max(A);
```

入力引数が複数ある場合には、それらをコンマで区切ります。

```
max(A,B);
```

関数の出力は変数に代入して返します。

```
maxA = max(A);
```

出力引数が複数ある場合には、それらを大かっこで囲みます。

```
[maxA,location] = max(A);
```

文字列入力は、すべて一重引用符で囲みます。

```
disp('hello world');
```

入力の必要がなく出力を一切返さない関数を呼び出すには、関数名だけを入力します。

```
clc
```

関数 clc はコマンド ウィンドウをクリアします。

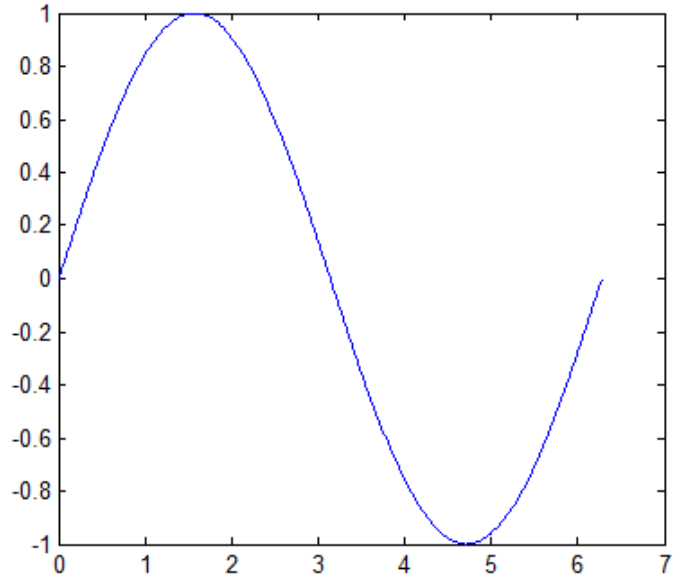
2 次元および 3 次元プロット

この節の内容…
ライン プロット (p. 1-18)
3 次元プロット (p. 1-21)
サブプロット (p. 1-22)

ライン プロット

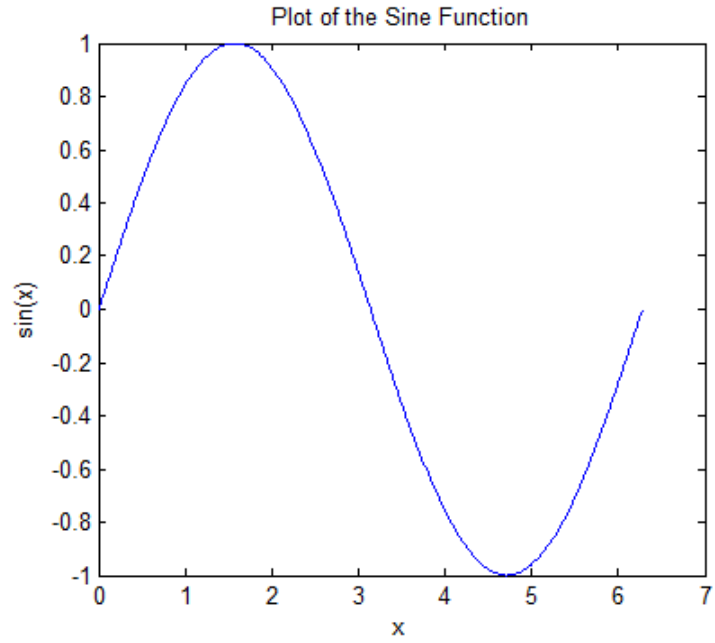
2 次元ライン プロットを作成するには、関数 plot を使用します。たとえば、正弦関数の値を 0 ～ 2π の間でプロットするには、次のようにします。

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```



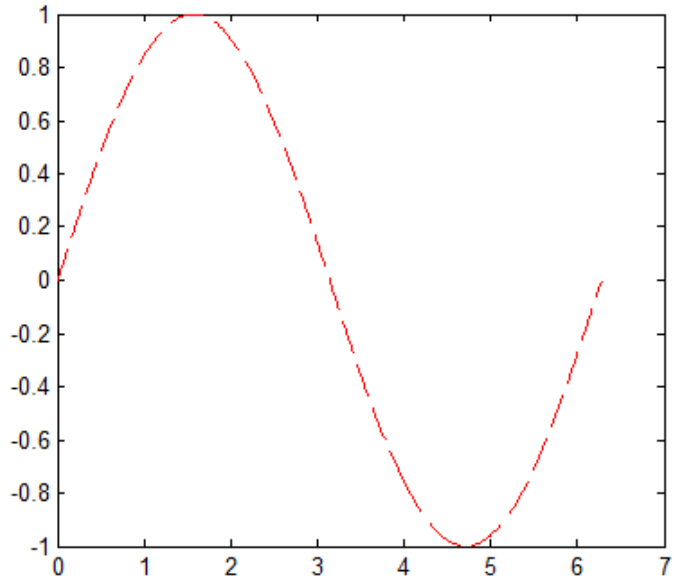
座標軸にラベルを付け、タイトルを加えることができます。

```
xlabel('x')
ylabel('sin(x)')
title('Plot of the Sine Function')
```



関数 plot に 3 番目の入力引数を追加することにより、同じ変数を赤い破線でプロットできます。

```
plot(x,y,'r--')
```

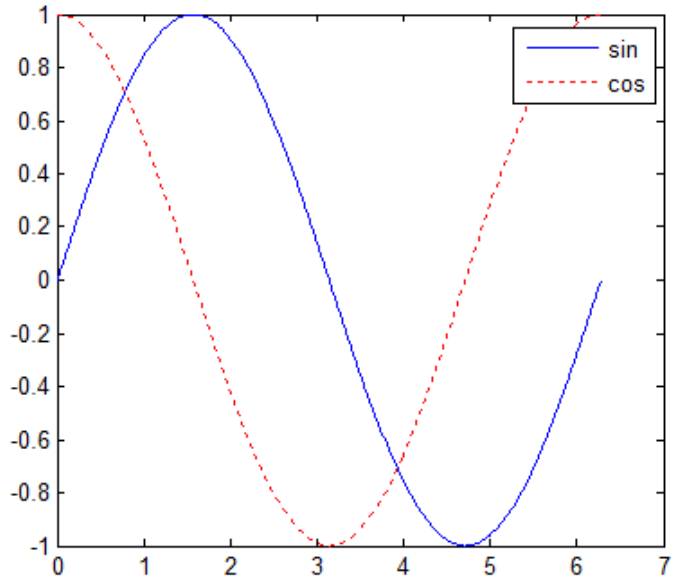


文字列 'r--' は "ライン仕様" です。それぞれの仕様には、ラインの色、スタイルおよびマーカーを表す文字を含めることができます。マーカーとは、プロットされた各データ点に表示される、+、o、* などの記号です。たとえば 'g:*' では、緑色の点線に * がマーカーとして表示されます。

現在の Figure ウィンドウ内に最初のプロットで定義したタイトルとラベルが表示されなくなったことに注目してください。既定の設定では、プロット関数を呼び出すたびに MATLAB によって Figure がクリアされ、新たなプロットに備えて座標軸や他の要素がリセットされます。

既存の Figure にプロットを追加するには、hold を使用します。

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)

hold on

y2 = cos(x);
plot(x,y2,'r:')
legend('sin','cos')
```



hold off を使用するかウィンドウを閉じるまで、すべてのプロットが現在の Figure ウィンドウに表示されます。

3 次元プロット

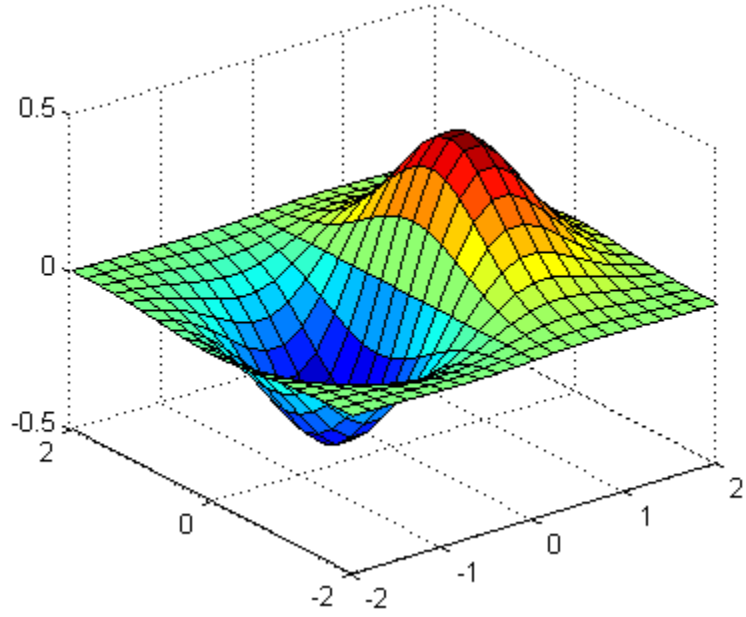
通常、3 次元プロットでは、*z* = f(*x*,*y*) のような 2 変数関数によって定義される表面が表示されます。

z の値を求めるには、まず meshgrid を使用して、関数の定義域で一連の (*x*,*y*) の点を作成します。

```
[X,Y] = meshgrid(-2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
```

次に、表面のプロットを作成します。

```
surf(X,Y,Z)
```



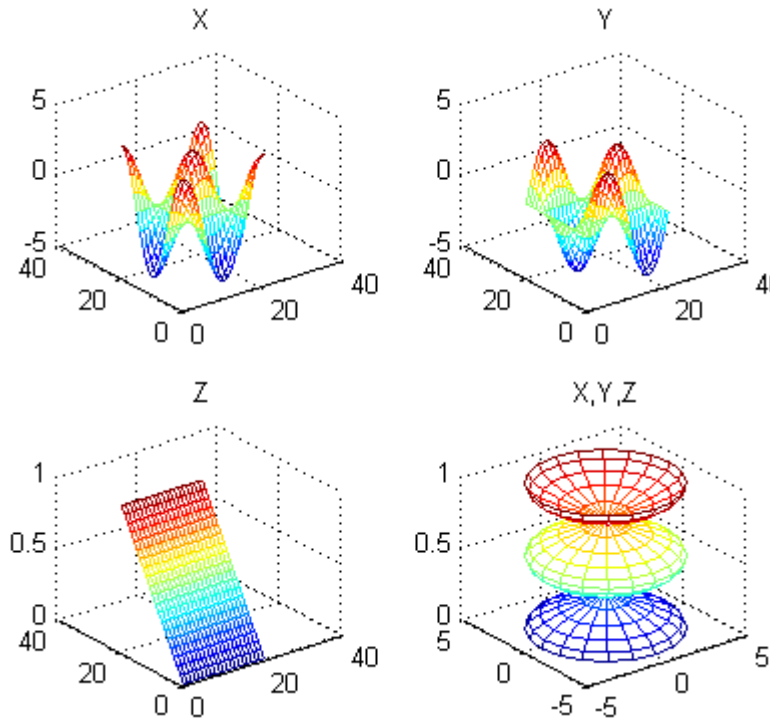
surf 関数とその対になる関数 mesh により、表面が 3 次元で表示されます。surf は、接続線と表面の構成面をカラー表示します。mesh は、定義点を結ぶラインのみをカラー表示した、ワイヤーフレームの表面を描きます。

サブプロット

関数 subplot を使用して、同じウィンドウの異なるサブ領域に複数のプロットを表示することができます。

たとえば、Figure ウィンドウ内の 2 行 2 列のグリッドに 4 つのプロットを作成します。

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X); title('X');
subplot(2,2,2); mesh(Y); title('Y');
subplot(2,2,3); mesh(Z); title('Z');
subplot(2,2,4); mesh(X,Y,Z); title('X,Y,Z');
```



関数 subplot の最初の 2 つの入力は、各行と各列におけるプロットの数を示します。3 番目の入力は、どのプロットがアクティブであるかを指定します。

プログラミングとスクリプト

この節の内容…

最も単純なタイプの MATLAB プログラムは "スクリプト" と呼ばれます。スクリプトは .m の拡張子をもつファイルで、複数行から成る一連の MATLAB コマンドと関数呼び出しが記載されています。スクリプトは、コマンド ラインにその名前を入力して実行することができます。

サンプル スクリプト

スクリプトを作成するには edit コマンドを使用します。

```
edit plotrand
```

これによって plotrand.m という名前の空白のファイルが開きます。乱数データのベクトルをプロットするコードを入力します。

```
n = 50;
r = rand(n,1);
plot(r)
```

次に、平均値を示す水平線をプロットに描画するコードを追加します。

```
m = mean(r);
hold on
plot([0,n],[m,m])
hold off
title('Mean of Random Uniform Data')
```

コードを記述する際は、常にそのコードについて説明するコメントを追加することをお勧めします。コメントがあれば他者にもそのコードが理解しやすくなり、後ほどコードに立ち戻る際に思い出しやすくなります。コメントは、パーセント記号 (%) を使用して追加します。

```
% Generate random data from a uniform distribution
```

```
% and calculate the mean. Plot the data and the mean.

n = 50;             % 50 data points
r = rand(n,1);
plot(r)

% Draw a line from (0,m) to (n,m)
m = mean(r);
hold on
plot([0,n],[m,m])
hold off
title('Mean of Random Uniform Data')
```

ファイルを現在のフォルダーに保存します。スクリプトを実行するには、その名前をコマンド ラインに入力します。

```
plotrand
```

スクリプトは、エディターで [実行] ボタン、  を押しても実行できます。

ループと条件付きステートメント

スクリプトでは、キーワード for、while、if および switch を使用してコードのセクションをループさせたり、セクションを条件付きで実行したりすることができます。

たとえば、for ループを使用して 5 つの無作為標本の平均と全体の平均を計算する calcmean.m というスクリプトを作成します。

```
nsamples = 5;
npoints = 50;

for k = 1:nsamples
    currentData = rand(npoints,1);
    sampleMean(k) = mean(currentData);
end
overallMean = mean(sampleMean)
```

ここで、反復ごとに結果を表示できるように for ループに変更を加えます。現在の反復回数を含むテキストをコマンド ウィンドウに表示し、sampleMean の代入ステートメントからセミコロンを削除します。

```
for k = 1:nsamples
   iterationString = ['Iteration #',int2str(k)];
   disp(iterationString)
   currentData = rand(npoints,1);
   sampleMean(k) = mean(currentData)
end
overallMean = mean(sampleMean)
```

スクリプトを実行すると中間結果が表示され、続いて全体平均が計算されます。

```
calcmean

Iteration #1

sampleMean =

    0.3988

Iteration #2

sampleMean =

    0.3988    0.4950

Iteration #3

sampleMean =

    0.3988    0.4950    0.5365

Iteration #4

sampleMean =

    0.3988    0.4950    0.5365    0.4870

Iteration #5

sampleMean =
```

```
    0.3988    0.4950    0.5365    0.4870    0.5501

overallMean =

    0.4935
```

エディターで、calcmean.m の最後に、overallMean の値に合わせて異なるメッセージを表示する条件付きステートメントを追加します。

```
if overallMean < .49
   disp('Mean is less than expected')
elseif overallMean > .51
   disp('Mean is greater than expected')
else
   disp('Mean is within the expected range')
end
```

calcmean を実行し、計算された overallMean に応じた正しいメッセージが表示されることを確認します。以下に例を示します。

```
overallMean =

    0.5178

Mean is greater than expected
```

スクリプトの保存場所

MATLAB では、特定の場所にあるスクリプトや他のファイルを検索します。スクリプトを実行するには、ファイルが現在のフォルダーか "検索パス" 上のフォルダーになければなりません。

既定の設定では、MATLAB インストーラーは MATLAB フォルダーを検索パス上に作成します。プログラムを他のフォルダーに保存して実行する場合は、そのフォルダーを検索パスに追加します。現在のフォルダー ブラウザーでフォルダーを選択して右クリックし、[パスに追加] を選択します。

ヘルプとドキュメンテーション

すべての MATLAB 関数には対応するドキュメンテーションが例と共に提供されており、関数の入出力や呼び出し構文が説明されています。コマンド ラインからこの情報へのアクセスにはいくつかの方法があります。

- doc コマンドを使用して関数のドキュメンテーションを別のウィンドウで開く。

  ```
  doc mean
  ```

- 関数の引数入力時に左かっこを入力後一時停止し、関数のヒント (関数のドキュメンテーションの構文部分) をコマンド ウィンドウに表示する。

  ```
  mean(
  ```

- help コマンドを使用して関数のドキュメンテーションの要約版をコマンド ウィンドウに表示する。

  ```
  help mean
  ```

完全版の製品ドキュメンテーションには、ヘルプ アイコン  をクリックしてアクセスします。

2

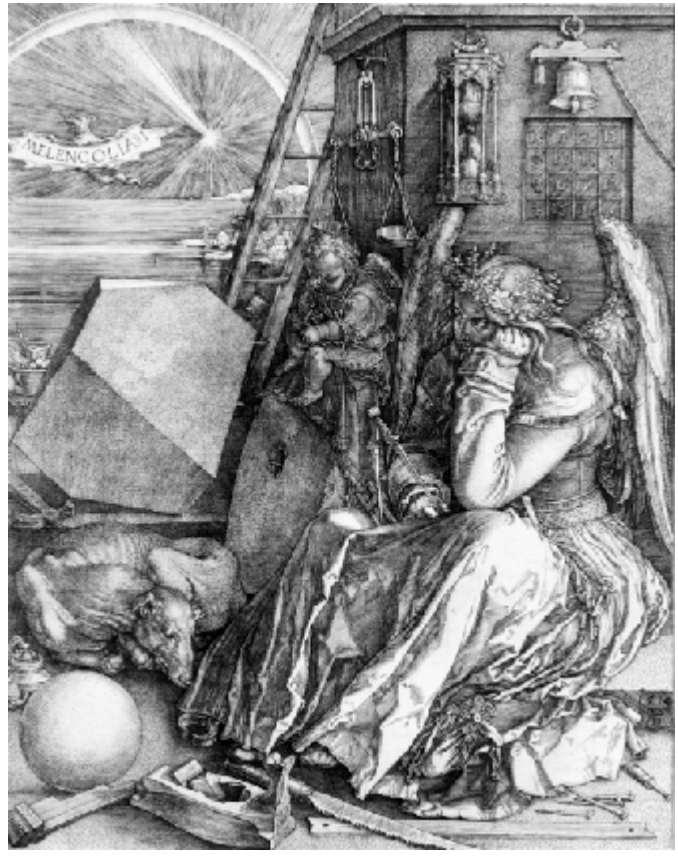
言語の基礎

行列と魔方陣

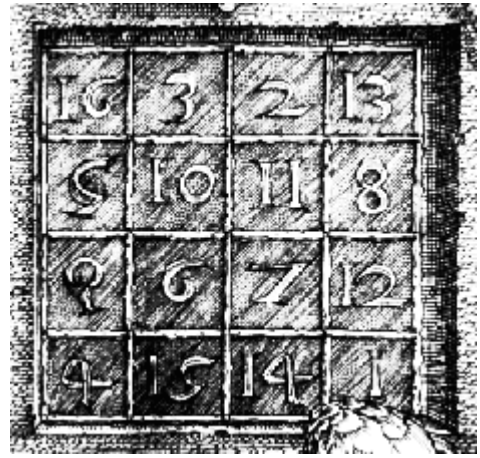
この節の内容…
行列について (p. 2-2)
行列の入力 (p. 2-4)
和、転置、対角 (p. 2-5)
関数 magic (p. 2-7)
行列の作成 (p. 2-8)

行列について

MATLAB 環境において、行列は数字を使った四角形配列です。1 行 1 列の行列 (スカラー) と、列または行が 1 つしかない行列 (ベクトル) は、特別な意味をもつ場合があります。MATLAB では数値データおよび非数値データを格納する他の方法も提供していますが、最初はすべてを行列として考えたほうがよいでしょう。MATLAB の中の演算は、できるだけ自然であるように作られています。他のプログラミング言語が、一度に数値一つに機能する部分で、MATLAB は迅速、かつ簡単に行列全体に働きます。この本を通して使われている良い例としての行列は、ドイツの芸術家でアマチュアの数学者であるアルブレヒト・デューラーによるルネッサンス銅版画である Melencolia I に現われているものです。



このイメージは数学的なシンボルで満たされています。そして、注意深く見ると、上の右隅に行列が見えます。この行列は、魔方陣として知られているもので、デューラーの時代に本当に魔法的な性質をもっていると多くの人に信じられていたものです。これは、表現に値するいくつかの素晴らしい機能をもっています。



行列の入力

MATLAB をはじめるにあたり、行列の取り扱いについて学ぶことからはじめるのが最もよいでしょう。MATLAB を起動し、各例を順に試みてください。

MATLAB に行列を定義する方法は、たくさんあります。

- 要素ごとに明示的に入力
- 外部データ ファイルから行列を読み込む
- 組み込み関数を使って行列を作成
- ユーザー独自の関数を使って行列を作成し、ファイルに保存

デューラーの行列を要素ごとに入力することから始めましょう。ユーザーがこれを行うには、2、3 の基本的な規則に従うだけです。

- 空白、またはコンマを使って、行の要素を分離
- セミコロン ; を使って、行の終わりを設定
- 要素全体を角かっこ [ ] で囲む

デューラー行列を入力するには、コマンド ウィンドウに単に次のように入力してください。

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLABは、入力された行列を次のように表現します。

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

これは、銅版画の中の数字と一致しています。行列を一度入力すると、MATLAB のワークスペースに自動的に記憶されます。ユーザーは、その行列を単に A として参照できます。さて、ワークスペース内に A を定義したので、見てみましょう。なぜ、魔法なのか

和、転置、対角

魔方陣行列の特別な性質、すなわち、各要素の種々の和に関していることは、多分ご存知でしょう。行または列に沿って、または二つの主対角に沿って和を計算すると、同じ数字（結果）を得るでしょう。MATLAB を使って確かめて見ましょう。まず、次のステートメントを試します。

```
sum(A)
```

MATLAB は、次の結果を表示します。

```
ans =
    34    34    34    34
```

出力引数を設定しないと、MATLAB は変数 ans（answer を省略）を用いて計算結果を格納します。これにより、A の各列の和を要素とする行ベクトルを計算すると、列のおのおのの要素はすべて同じ、すなわち魔方陣の和は 34 です。

行の和はどうでしょう。MATLAB では、行列の列に関しての処理が優先されます。したがって、行の和を得る方法の 1 つは、行列を転置し、転置した列の和を計算して、その結果を再度転置することです。

MATLAB には 2 つの転置演算があります。アポストロフィ演算子（例: A'）は複素共役転置を実行します。これは主対角に対して行列を入れ替え、行列の複素数成分の虚数要素の符号を変更します。ドットアポストロフィ演算子（A.'）は複素数要素

の符号を変えずに転置を実行します。すべての要素が実数の行列では、これら 2 つの演算子の処理は同じになります。

したがって

```
A'
```

は、以下の結果を出力します。

```
ans =
    16     5     9     4
     3    10     6    15
     2    11     7    14
    13     8    12     1
```

および

```
sum(A')'
```

は、行方向の和を含んだ列ベクトルを出力します。

```
ans =
    34
    34
    34
    34
```

2 回の転置を行わずに行を合計する他の方法として、関数 sum に対して次元の引数を使用します。

```
sum(A,2)
```

は、以下の結果を出力します。

```
ans =
    34
    34
    34
    34
```

主対角要素の和は、関数 sum と diag を使って得られます。

```
diag(A)
```

は、以下の結果を出力します。

```
ans =
    16
    10
     7
     1
```

および

```
sum(diag(A))
```

は、以下の結果を出力します。

```
ans =
    34
```

他の対角部、すなわち逆対角は、数学的にはあまり重要ではありません。そのため MATLAB にはそれに対応するような関数はありません。しかしグラフィックスの中で使われる関数 fliplr は、行列の左から右への順番を逆にします。

```
sum(diag(fliplr(A)))
ans =
    34
```

これでデューラーの銅版画の中の行列は、本当に魔方陣であることが確認されました。そして、この過程には、2、3 の MATLAB の行列演算をサンプルとして示しています。次の節では、MATLAB の他の機能を示すためにこの行列を使います。

関数 magic

MATLAB は、任意のサイズの魔方陣を作る組み込み関数を用意しています。当然ながら、この関数名は magic です。

```
B = magic(4)
B =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

この行列は、デューラーの銅版画の中のものとほぼ同じで、同様な「魔方陣」の性質をもっています。ただし、中央の列が入れ替わっていることが異なります。

この B をデューラーの A にするため、中央の 2 つの列を入れ替えましょう。

```
A = B(:,[1 3 2 4])
```

この添字を使うと、行列 B の行のそれぞれに対して、要素の順番を 1、3、2、4 の順に並べ替えます。次の結果を得ます。

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

行列の作成

MATLAB は、基本的な行列を作成する 4 つの関数を用意しています。

zeros	要素がすべて 0 の行列
ones	要素がすべて 1 の行列
rand	一様分布する乱数要素からなる行列
randn	正規分布する乱数要素からなる行列

以下にいくつかの例を示します。

```
Z = zeros(2,4)
Z =
     0     0     0     0
     0     0     0     0

F = 5*ones(3,3)
F =
     5     5     5
     5     5     5
     5     5     5
```

```
N = fix(10*rand(1,10))
N =
     9     2     6     4     8     7     4     0     8     4

R = randn(4,4)
R =
    0.6353    0.0860   -0.3210   -1.2316
   -0.6014   -2.0046    1.2366    1.0556
    0.5512   -0.4931   -0.6313   -0.1132
   -1.0998    0.4620   -2.3252    0.3792
```

式

この節の内容…

変数

他の大部分のプログラミング言語と同様に、MATLAB は数学的な式を使いますが、他のプログラム言語と異なり、これらの式は行列全体を含んでいます。

MATLAB は、タイプの宣言や次元を宣言するステートメントを必要としていません。MATLAB は新しい変数名を使おうとするとき、自動的に変数を作成し、適切なサイズのストレージを割り当てます。変数が既に存在していると、MATLAB は必要ならその内容を変更し、新しいストレージを割り当てます。たとえば、次の例を考えてみましょう。

```
num_students = 25
```

は、1 行 1 列の num_students と名付けた行列を作成し、その単一要素に値 25 を格納します。任意の変数に割り当てられた 行列を見るには、変数名を入力するだけです。

変数名は、1 つの文字を先頭に、その後に任意の数の文字、数字、またはアンダースコアを続けて表します。MATLAB では大文字と小文字が区別されます。つまり、A と a は、同じ変数では "ありません"。

変数名は任意の長さにすることができますが、MATLAB は先頭から N 個の文字のみを使用して (ここでは N は、関数 namelengthmax から返される数字です)、それ以降の残りの文字を無視します。このため MATLAB が変数を識別できるように、各変数名の先頭から N 個の文字をユニークにしなければなりません。

```
N = namelengthmax
N =
    63
```

関数 genvarname は、適切かつ重複しない変数名の作成に有効です。

数字

MATLAB は、通常の小数点表示を行います。これは、オプションの小数点をもち、数字の先頭にプラスまたはマイナスの符号をつけます。科学的な記法として、10 のべき乗のスケール係数を設定する文字 e を使います。虚数は、i または j をサフィックスとして使います。正しく表現された数字の例を示します。

```
3              -99            0.0001
9.6397238      1.60210e-20    6.02252e23
1i             -3.14159j      3e5i
```

すべての数字は、IEEE® 浮動小数点標準で指定される long 形式を使用して内部的に保存されます。浮動小数点数はおよそ 16 桁の数字の有限精度をもち、10^{-308} から 10^{+308} の有限な範囲に入ります。

double 形式で表される数字は、52 ビットの最高精度をもちます。52 を超えるビット数が必要な double 型では、精度が多少低下します。たとえば、2 つの異なる値が切り捨てられることによって両方とも等しい値になることを、次のコードで示します。

```
x = 36028797018963968;
y = 36028797018963972;
x == y
ans =
      1
```

整数では 8 ビット、16 ビット、32 ビット、および 64 ビットの精度を使用できます。64 ビットの整数として同じ数字を保存すると、精度を保持できます。

```
x = uint64(36028797018963968);
y = uint64(36028797018963972);
x == y
ans =
      0
```

MATLAB は複素数の実数部と虚数部を格納します。そしてコンテキストに応じて様々な方法で、各部分大きさを取り扱います。たとえば関数 sort は大きさに基づいて並べ替えてから位相角で順序を決めます。

```
sort([3+4i, 4+3i])
```

```
ans =
   4.0000 + 3.0000i   3.0000 + 4.0000i
```

これは位相角が次のようになっているためです。

```
angle(3+4i)
ans =
    0.9273
angle(4+3i)
ans =
    0.6435
```

「等価」関係演算子 == は、実数部と虚数部の両方が等しいことを要求します。他の二項関係演算子 > <、>=、<= は数の虚数部を無視し、実数部のみを考慮します。

行列演算子

式には、馴染みの深い算術演算子と優先順位則を使います。

+	加算
−	減算
*	乗算
/	除算
¥	左除算
^	べき乗
'	複素共役転置
( )	演算順序の変更

配列演算子

線形代数の世界から離れたとき、行列は 2 次元の数値配列になります。配列上の代数演算は、要素単位で行われます。このことは、加算、減算では、行列も配列も同じで、乗算演算では異なることになります。MATLAB では、乗法的配列演算をドット、小数点を使って表現します。

演算機能の一覧を示します。

+	加算
-	減算
.*	要素単位の乗算
./	要素単位の除算
.¥	要素単位の左除算
.^	要素単位のべき算
.'	共役を計算しないで転置のみを行う

デューラーの魔方陣に、要素単位でそれ自身を乗算すると、

```
A.*A
```

結果は 1 から 16 までの整数の二乗を含んだ配列になります。

```
ans =
   256     9     4   169
    25   100   121    64
    81    36    49   144
    16   225   196     1
```

テーブルの作成

配列演算は、テーブルを作成するのに有効です。n を列ベクトルとしましょう。

```
n = (0:9)';
```

したがって、

```
pows = [n  n.^2  2.^n]
```

これは二乗と 2 のべき乗のテーブルになります。

```
pows =
     0     0     1
     1     1     2
     2     4     4
     3     9     8
     4    16    16
     5    25    32
     6    36    64
     7    49   128
     8    64   256
     9    81   512
```

初等数学関数は、配列上に要素単位で働きます。

```
format short g
x = (1:0.1:2)';
logs = [x log10(x)]
```

により、対数テーブルを作成できます。

```
 logs =
     1.0         0
     1.1   0.04139
     1.2   0.07918
     1.3   0.11394
     1.4   0.14613
     1.5   0.17609
     1.6   0.20412
     1.7   0.23045
     1.8   0.25527
     1.9   0.27875
     2.0   0.30103
```

関数

MATLAB は、abs、sqrt、exp、sin を含む多くの標準的な初等数学関数を用意しています。負の数の平方根や対数は、エラーにはなりません。適切な複素数で表された結果が自動的に出力されます。MATLAB は、ベッセル関数やガンマ関数など、より高度な数学関数も提供します。これらの関数のほとんどは、複素数と共に使えます。初等数学関数の一覧を得るには、

```
help elfun
```

そして、より高度な数学関数や行列関数の一覧を得るのは、

```
help specfun
help elmat
```

と入力します。sqrt や sin のような関数は、組み込みです。これら組み込み関数は、MATLAB のコアの一部で、非常に効率よく作られていますが、計算の詳細を見ることはできません。その他の関数は MATLAB プログラミング言語で実装されているため、計算の詳細を参照できます。

組み込み関数とその他の関数ではいくつかの違いがあります。たとえば、ユーザーは組み込み関数の内部コードを読むことはできません。その他の関数はユーザーが内容を見ることができ必要なら内容を変更することもできます。

いくつかの特別な関数は、有用な定数値を用意しています。

pi	3.14159265…
i	虚数単位、$\sqrt{-1}$
j	i に同じ
eps	浮動小数点の相対精度、$\varepsilon = 2^{-52}$
realmin	最小浮動小数点数、2^{-1022}
realmax	最大浮動小数点数、$(2-\varepsilon)2^{1023}$
Inf	無限大
NaN	Not-a-number

無限大は、非ゼロ値をゼロで割った場合や、明確に定義された数式において realmax を "オーバーフロー" する (超える) 値が求められた場合に生成されます。Not-a-number は、0/0 または Inf-Inf のようなうまく定義できない数学的な値を計算するときに作成されます。

関数の名前は、予約されていません。これらは、次のような新しい変数で上書きできます。

```
eps = 1.e-6
```

そしてその値は、その後の計算で使われます。もともと定義されている関数は、以下を使うことにより、既定の値に戻ります。

```
clear eps
```

方程式の例

MATLAB の式のいくつかの例を既に見てきました。ここでも、いくつかの例とその結果を示します。

```
rho = (1+sqrt(5))/2
rho =
    1.6180

a = abs(3+4i)
a =
     5

z = sqrt(besselk(4/3,rho-i))
z =
   0.3730+ 0.3214i

huge = exp(log(realmax))
huge =
  1.7977e+308

toobig = pi*huge
toobig =
   Inf
```

コマンドの入力

この節の内容…

関数 format

関数 format は、表示される数値形式を制御するものです。この関数は、数値の表示にのみ影響するもので、MATLAB で計算をする方法や保存する方法に影響するものではありません。ここでは、異なる大きさの成分からなるベクトル x を使って、種々の形式設定に従って出力される結果を表示します。

メモ: 適切な間隔を保証するために、Courier などの固定幅のフォントを用いてください。

```
x = [4/3 1.2345e-6]

format short

   1.3333    0.0000

format short e

   1.3333e+000  1.2345e-006

format short g

   1.3333  1.2345e-006

format long

   1.33333333333333   0.00000123450000
```

```
format long e

   1.333333333333333e+000      1.234500000000000e-006

format long g

   1.33333333333333                    1.2345e-006

format bank

   1.33            0.00

format rat

   4/3             1/810045

format hex

   3ff5555555555555    3eb4b6231abfd271
```

行列の中の最大の要素が 10^3 より大きいかまたは、10^{-3} より小さいならば、MATLAB は short と long 形式に共通のスケール係数を適用します。

上で示した関数 format に加え、

```
format compact
```

は、出力の中に現れる多くの空白行を取り去ります。この形式は、スクリーンまたはウィンドウ上に、より多くの情報を表示するためのものです。さらに、出力形式を制御するためには、関数 sprintf や fprintf を使ってください。

出力表示の抑制

あるステートメントを入力し、Return キーまたは Enter キーを押すと、MATLAB は自動的にスクリーン上に結果を表示します。しかし、ラインの終わりにセミコロンを付けると、MATLAB で計算は実行されますが、出力は一切表示されなくなります。これは特に大きな行列を作成するとき有効です。たとえば、次の例を考えてみましょう。

```
A = magic(100);
```

長いステートメントの入力

ステートメントが長すぎて 1 行に収まらない場合、... を付け、Return キーまたは Enter キーを押して、新しい行にコマンドを続けることができます。たとえば、次の例を考えてみましょう。

```
s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
      - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

=、+、- の周りの空白スペースはオプションで、読み易くするものです。

コマンド ラインの編集

ユーザーのキーボード上の種々の矢印やコントロール キーを使って、以前に入力したコマンドを再表示したり、編集したり、再使用したりできます。たとえば、誤って次のように入力したと仮定します。

```
rho = (1 + sqt(5))/2
```

ここで、sqrt をタイプミスしています。MATLAB は、次のように表示します。

```
Undefined function 'sqt' for input arguments of type 'double'.
```

ライン全体を入力し直す代わりに ↑ キーを押してください。タイプミスをした入力ラインが再表示されます。←キーを使用してカーソルを移動し、入力し忘れた r を入力してください。↑ キーを続けて使用すると、以前に入力した行が再表示されます。何文字か入力した後に ↑ キーを押すと、以前に入力した、それらの文字で始まるラインが表示されます。コマンド履歴から前に実行したコマンドをコピーすることもできます。

インデックス

この節の内容…

添字付け

A の i 番目の行、j 番目の列の要素を A(i,j) と定義します。たとえば A(4,2) は 4 番目の行の 2 番目の列の数字です。この例で使っている魔方陣では A(4,2) は 15 です。A の 4 番目の列の要素の和は、次のように入力することで求まります。

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

これにより、

```
ans =
     34
```

しかし単一の列の和を計算する方法では、最も洗練されていない方法です。

これは単一の添字をもつ行列要素 (たとえば A(k)) と考えることもできます。単一の添字は、行ベクトルと列ベクトルを参照する通常の方法です。しかし、2 次元の行列にも適用することができます。この場合、もとの行列を列方向に分解して作られる一つの長い列ベクトルとしての配列と考えられます。それを使って、例で使っている魔方陣において、A(8) は A(4,2) に格納されている値 15 と考えることもできます。

行列で定義されている要素以外のものを使おうとすると、エラーが生じます。

```
t = A(4,5)
Index exceeds matrix dimensions.
```

一方、行列で定義されている要素以外の要素に値を格納しようとすると、新しい要素を行列に付加えてサイズが大きくなります。

```
X = A;
X(4,5) = 17

X =
    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4    15    14     1    17
```

コロン演算子

コロン記号 : はMATLAB の重要な演算子の一つです。これは種々の異なる型で使われます。式

```
1:10
```

は、1 から 10 までの整数を含む行ベクトルを作成します。

```
1     2     3     4     5     6     7     8     9    10
```

単位間隔でないものを得るには、間隔を示す量を設定します。たとえば、次の例を考えてみましょう。

```
100:-7:50
```

これは次のようになります。

```
100    93    86    79    72    65    58    51
```

および

```
0:pi/4:pi
```

これは次のようになります。

```
0    0.7854    1.5708    2.3562    3.1416
```

コロン記号を含んだ添字表現は、行列のある一部を表現します。

```
A(1:k,j)
```

は、A の j 番目の列の最初から k 番目までの要素に対応します。したがって、次のようになります。

```
sum(A(1:4,4))
```

は、4 番目の列の和を計算することになります。しかし、この計算を実行するには、さらに良い方法があります。コロン自身は、行列の行または列の中の全部の要素を意味しています。そして、キーワード end は、行または列の最後を意味します。したがって、次のようになります。

```
sum(A(:,end))
```

は、A の最後の列の要素の和を計算します。答えは、

```
ans =
     34
```

なぜ、4 行 4 列の正方形に対して、魔方陣の和は 34 なのですか。1 から 16 までの整数は、和が等しくなるように 4 つのグループに並べ替えられたとすると、その和は、

```
sum(1:16)/4
```

にならなければなりません。それでもちろん次のようになります。

```
ans =
     34
```

連結

連結は、小さな行列を合わせてより大きな行列にする作業です。実際に、最初の行列は、個々の要素を連結して作ったものです。[] 記号は、連結を行う演算子です。たとえば、4 行 4 列の魔方陣 A を使って、次の形式を作成します。

```
B = [A  A+32; A+48  A+16]
```

結果は、4 つの小行列を連結した 8 行 8 列の行列になります。

```
B =

    16     3     2    13    48    35    34    45
     5    10    11     8    37    42    43    40
     9     6     7    12    41    38    39    44
     4    15    14     1    36    47    46    33
    64    51    50    61    32    19    18    29
    53    58    59    56    21    26    27    24
    57    54    55    60    25    22    23    28
    52    63    62    49    20    31    30    17
```

この行列は、別の魔方陣を作成する方法の途中結果になっています。この要素は、整数 1:64 を並べ替えたもので、その列の和は、8 行 8 列の魔方陣行列の各列の和と等しい値になります。

```
sum(B)

ans =
   260   260   260   260   260   260   260   260
```

しかし、その行の和 sum(B')' は、すべて同じものにはなりません。正しい 8 行 8 列の魔方陣を作成するには、さらに操作が必要です。

行と列の削除

大かっこ [ ] を使って、行列から行や列を削除することができます。次のように指定します。

```
X = A;
```

そして、X の 2 番目の列を削除するため、次のように行います。

```
X(:,2) = []
```

X は次のようになります。

```
X =
    16     2    13
     5    11     8
     9     7    12
     4    14     1
```

行列から単一要素を削除する場合、結果はもはや行列になりません。すなわち、次のような表現

```
X(1,2) = []
```

は、エラーになります。しかし、単一の添字を使って、単一要素または連続している要素群を削除して、残りの要素を行ベクトルとして表現することはできます。したがって

```
X(2:2:10) = []
```

結果は次のようになります。

```
X =
    16     9     2     7    13    12     1
```

スカラー拡張

行列とスカラーは、種々の方法で組み合わされます。たとえば、スカラーは行列との減算では、行列の各要素から自分自身を差し引きます。例の魔方陣の要素の平均値は 8.5 です。

```
B = A - 8.5
```

は、列の和がゼロになる行列を作成します。

```
B =
      7.5     -5.5     -6.5      4.5
     -3.5      1.5      2.5     -0.5
      0.5     -2.5     -1.5      3.5
     -4.5      6.5      5.5     -7.5

sum(B)

ans =
     0     0     0     0
```

スカラーの拡張と共に、MATLAB は範囲内のすべてのインデックスに指定したスカラー値を与えることができます。たとえば、次の例を考えてみましょう。

```
B(1:2,2:3) = 0
```

これによって B の一部分がゼロに設定されます。

```
B =
     7.5       0       0     4.5
    -3.5       0       0    -0.5
     0.5    -2.5    -1.5     3.5
    -4.5     6.5     5.5    -7.5
```

論理添字

論理演算や比較演算から作成される論理ベクトルが、サブ配列を参照するのに使われます。X を通常の行列で、L をある論理演算の結果求まった同じサイズの行列とします。そして、X(L) は、L の要素がゼロでない部分の X の要素を指定します。

この種の添字機能は、添字表現と同じように論理演算を指定することにより 1 手順の中で処理されます。次のデータを考えましょう。

```
x = [2.1 1.7 1.6 1.5 NaN 1.9 1.8 1.5 5.1 1.8 1.4 2.2 1.6 1.8];
```

NaN は、アンケート質問表での無回答に対応する項目のような、データの欠測部分です。論理インデックスによって欠測データを取り除くために isfinite(x) を使うと、有限の数値は真、NaN や Inf は偽となります。

```
x = x(isfinite(x))
x =
  2.1 1.7 1.6 1.5 1.9 1.8 1.5 5.1 1.8 1.4 2.2 1.6 1.8
```

ある観測値 5.1 は、他のものと比べ非常に異なって見えます。これを外れ値と言います。ここでの例では、平均値から標準偏差の 3 倍以上の部分を外れ値として取り除きます。

```
x = x(abs(x-mean(x)) <= 3*std(x))
x =
  2.1 1.7 1.6 1.5 1.9 1.8 1.5 1.8 1.4 2.2 1.6 1.8
```

他の例として、デューラーの魔方陣の中で、論理インデックス法とスカラー拡張法を使って素数でないものに 0 を設定して素数の位置を強調します (関数 magic (p. 2-7)を参照)。

```
A(~isprime(A)) = 0

A =
     0     3     2    13
     5     0    11     0
```

```
     0     0     7     0
     0     0     0     0
```

関数 find

関数 find は、任意の論理条件を満たす配列要素のインデックスを判別します。この最も簡単な型では、関数 find がインデックスの列ベクトルを出力します。インデックスの行ベクトルを得るためには、そのベクトルを転置します。たとえば、デューラーの魔方陣で再び始めます。関数 magic (p. 2-7)を参照してください。

```
k = find(isprime(A))'
```

は、1 次元のインデックス法を使って、魔方陣の中の素数の位置を出力します。

```
k =
     2     5     9    10    11    13
```

これらの素数を表示するのに、k により決められた順番に行ベクトルとして配列します。

```
A(k)

ans =
     5     3     2    11     7    13
```

代入ステートメントで左辺のインデックスとして k を使用すると、行列構造は保持されます。

```
A(k) = NaN

A =
    16   NaN   NaN   NaN
   NaN    10   NaN     8
     9     6   NaN    12
     4    15    14     1
```

配列のタイプ

この節の内容…
多次元配列 (p. 2-27)
セル配列 (p. 2-29)
文字とテキスト (p. 2-31)
構造体 (p. 2-34)

多次元配列

MATLAB の多次元配列は、3 つ以上の添字をもった配列です。これは、2 つ以上の引数と共に zeros、ones、rand、randn を呼び出すことによって作成されます。たとえば、次の例を考えてみましょう。

```
R = randn(3,4,5);
```

は、3*4*5 = 60 個の正規分布する乱数要素をもつ 3 x 4 x 5 の配列を作成します。

3 次元配列は、3 次元の物理的データを表わします。たとえば、四角形グリッド上でサンプリングした室温を表します。また、行列の数列 $A^{(k)}$ や時間依存行列のサンプル A(t) を表します。後者の場合、t_k 番目の行列、または k 番目の行列の (i,j) 番目の要素は、A(i,j,k) によって表されます。

4 次の魔方陣の MATLAB バージョンとデューラー バージョンは、2 つの列が入れ替わっています。列を入れ替えることにより、異なる魔方陣を作ることができます。ステートメント

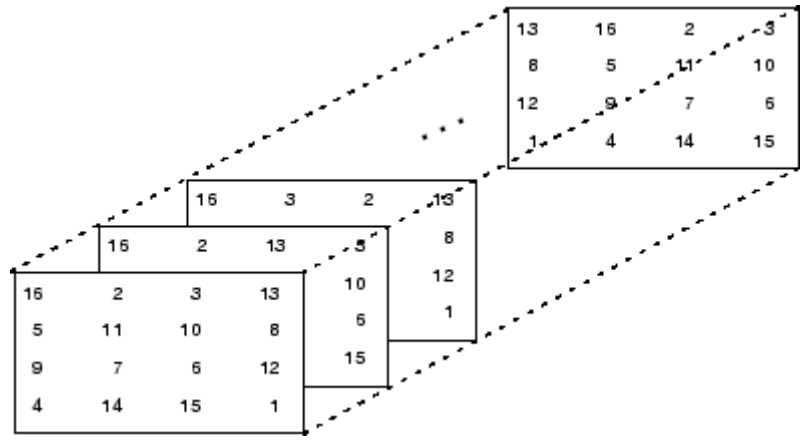
```
p = perms(1:4);
```

は、1:4 の順列 4! = 24 の組み合わせを生成します。k 番目の順列は、行ベクトル p(k,:) です。したがって、

```
A  =  magic(4);
M = zeros(4,4,24);

for k = 1:24
   M(:,:,k) = A(:,p(k,:));
end
```

は、3 次元配列 M に 24 種の魔方陣を格納します。M のサイズは、

```
size(M)

ans =
     4     4    24
```



メモ: この例に示されている行列の順はユーザーの結果と異なる可能性があります。関数 perms は常に入力ベクトルの置き換えを返しますが、その順序は MATLAB バージョンによって違うことがあります。

ステートメント

```
sum(M,d)
```

は、d 番目の添字を変更して和を計算します。

```
sum(M,1)
```

は、24 個の行ベクトル

```
34     34     34     34
```

および

```
sum(M,2)
```

は、24 個の列ベクトル

```
34
34
34
34
```

最終的に、

```
S = sum(M,3)
```

は、24 個の行列を足したものなります。結果は 4 x 4 x 1 になり、4 行 4 列の配列に見えます。

```
S =
   204   204   204   204
   204   204   204   204
   204   204   204   204
   204   204   204   204
```

セル配列

MATLAB のセル配列は、多次元配列で、その要素は他の配列のコピーです。空行列のセル配列は、関数 cell で作成できます。しかし、非常に頻繁にセル配列は、中かっこ {} を使って様々なものを囲むことによって作成されます。中かっこは、種々のセルの内容にアクセスするための添字と共に使われることもあります。たとえば、次の例を考えてみましょう。

```
C = {A sum(A) prod(prod(A))}
```

は、1 行 3 列のセル配列を作成します。3 つのセルは、魔方陣、列方向の和を表わす行ベクトル、すべての要素の積から構成されています。C を表示すると、次のようになります。

```
C =
    [4x4 double]    [1x4 double]    [20922789888000]
```

これは、最初の 2 つのセルは、スペースの関係で出力するには大きすぎるためです。しかし、3 番目のセルは 1 つの数値 16! なので、出力されています。

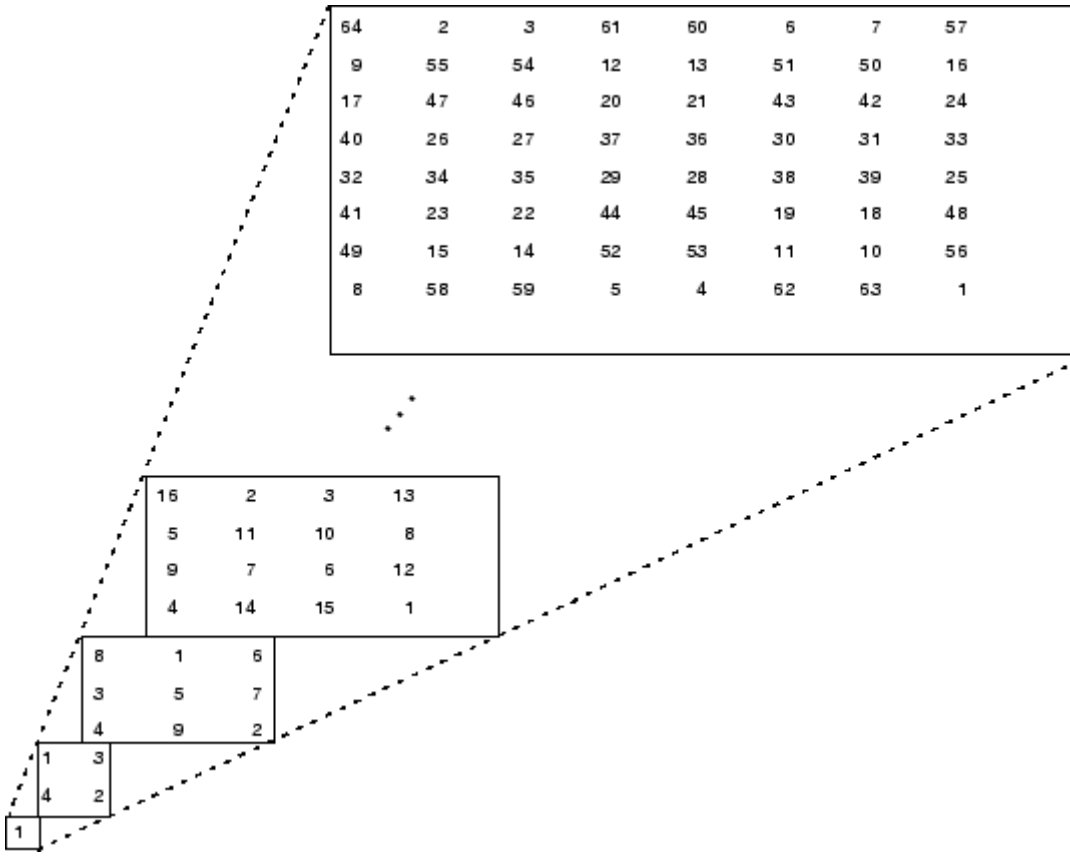
覚えておく必要のある 2 つの重要なことがあります。まず、セルの中の個々の内容を取り出すには、添字を中かっこで囲んで使います。たとえば、C{1} は魔方陣で、C{3} は 16! です。2 番目に、セル配列は、他の配列のコピーを含み、他の配列へのポインターではありません。その後 A を変更しても C は何も変更されません。

3 次元配列は、同じサイズの一連の行列を格納するときに使うことができます。セル配列は、異なるサイズの一連の行列を格納するためにも使うことができます。たとえば、次の例を考えてみましょう。

```
M = cell(8,1);
for n = 1:8
   M{n} = magic(n);
end
M
```

は、異なる次数の一連の魔方陣を作成します。

```
M =
    [           1]
    [ 2x2  double]
    [ 3x3  double]
    [ 4x4  double]
    [ 5x5  double]
    [ 6x6  double]
    [ 7x7  double]
    [ 8x8  double]
```



今まで使ってきた馴染みのある魔方陣も取り出すことができます。

```
M{4}
```

文字とテキスト

MATLAB へテキストを入力するには、一重引用符を利用します。たとえば、次の例を考えてみましょう。

```
s = 'Hello'
```

結果は、今まで取り扱ってきた数値行列や配列と異なる種類のものです。これは、1 行 5 列の文字配列です。

内部的に、文字は数字として格納されますが、浮動小数点形式ではありません。
ステートメント

```
a = double(s)
```

は、文字配列を ASCII コードの浮動小数点表現を含む数値行列に変換します。
結果は、次のようになります。

```
a =
    72    101    108    108    111
```

ステートメント

```
s = char(a)
```

は、逆の変換を行います。

数字を文字に変換することにより、ユーザーのコンピューターで使用可能なフォントの種類を調べることができます。基本的な ASCII 文字セットの中のプリント可能な文字は、整数 32:127 で表されます(33 よりも小さい整数は、プリントできない制御文字を表します)。これらの整数は、次のように適切な 6 行 16 列の配列に並べられます。

```
F = reshape(32:127,16,6)';
```

拡張 ASCII 文字セットの中のプリント可能な文字は、F+128 によって表されます。これらの整数が文字として解釈されるとき、結果は現在使われているフォントに依存します。次のステートメントをタイプします。

```
char(F)
char(F+128)
```

するとコマンド ウィンドウで使われているフォントが変化します。フォントを変更するには、[ホーム] タブの [環境] セクションで、[設定] 、[フォント] をクリックします。コードの行にタブを使っている場合は、他の行とタブ位置を合わせるために Monospaced などの固定幅フォントをお使いください。

大かっこの中の連結は、テキスト変数をより大きい文字列にします。ステートメント

```
h = [s, ' world']
```

は、横方向に文字列を連結し、次の結果を出力します。

```
h =
```

```
   Hello world
```

ステートメント

```
v = [s; 'world']
```

は、縦方向に文字列を連結し、次の結果を出力します。

```
v =
   Hello
   world
```

h で 'w' の前に空白が挿入されていなければならないことと、v で 2 つの単語は同じ長さでなければならないことに注意してください。結果の配列は、共に文字配列で、h は 1 行 11 列で、v は 2 行 5 列です。

異なる 長さのラインを含むテキストの内容を取り扱うには、文字配列を付け加えて同じ長さにするか、文字列のセル配列にするかの 2 つの方法があります。文字配列を作る際には配列のすべての行を同じ長さにしなければなりません。(短い行の末尾を空白文字で埋めてください。)関数 char はこの埋め合わせを自動的に行います。たとえば、次の例を考えてみましょう。

```
S = char('A','rolling','stone','gathers','momentum.')
```

は、5 行 9 列の文字配列です。

```
S =
A
rolling
stone
gathers
momentum.
```

または、1 つのセル配列にテキストを格納することができます。たとえば、次の例を考えてみましょう。

```
C = {'A';'rolling';'stone';'gathers';'momentum.'}
```

は、配列の各行が異なる長さであることができるので埋め合わせの必要のない、5 行 1 列のセル配列を作ります。

```
C =
```

```
    'A'
    'rolling'
    'stone'
    'gathers'
    'momentum.'
```

空白が追加された文字配列を、次のコマンドで文字列からなるセル配列に変換できます。

```
C = cellstr(S)
```

そして、逆も可能です。

```
S = char(C)
```

構造体

構造体は、テキストのフィールド識別子によりアクセスできる要素をもつ多次元 MATLAB 配列です。たとえば、次の例を考えてみましょう。

```
S.name = 'Ed Plum';
S.score = 83;
S.grade = 'B+'
```

は、3 つのフィールドをもったスカラー構造体を作成します。

```
S =
     name: 'Ed Plum'
    score: 83
    grade: 'B+'
```

MATLAB の中のすべてのものと同じように、構造体は配列で、そのため付加的な要素も挿入することができます。この場合、配列の各々の要素は、いくつかのフィールドをもつ構造体です。フィールドは、一度に一つのものを加えることができます。

```
S(2).name = 'Toni Miller';
S(2).score = 91;
S(2).grade = 'A-';
```

または、すべての要素が単一ステートメントで加えられます。

```
S(3) = struct('name','Jerry Garcia',...
              'score',70,'grade','C')
```

構造体は、内容の概略だけをプリントしても十分に大きいものです。

```
S =
1x3 struct array with fields:
    name
    score
    grade
```

種々のフィールドを他の MATLAB 配列に集め直すのにいくつかの方法があります。これらは、コンマでリストを分離する記法をもとにしています。以下のように入力すると、

```
S.score
```

次の事柄と等価です。

```
S(1).score, S(2).score, S(3).score
```

これは、各要素をコンマによる分離で表現するものです。

大かっこ内のリストなどを作成する式を囲むと、MATLAB はリストからの各項目を配列に保存します。この例では、MATLAB は、構造体配列 S の各要素の score フィールドを含む、数値の行ベクトルを作成します。

```
scores = [S.score]
scores =
    83    91    70

avg_score = sum(scores)/length(scores)
avg_score =
   81.3333
```

テキスト フィールドの 1 つ (たとえば、name) から文字配列を作成するには、S.name で生成されるコンマ区切りのリストに関数 char を呼び出します。

```
names = char(S.name)
names =
   Ed Plum
   Toni Miller
   Jerry Garcia
```

同様に、中かっこ内のリストを作成する式を囲むと name フィールドからセル配列を作成できます。

```
names = {S.name}
names =
    'Ed Plum'    'Toni Miller'    'Jerry Garcia'
```

構造体配列の各要素のフィールドを、構造体の外側の別々の変数に割り当てるには、出力全体を大かっこで囲み等号の左に指定します。

```
[N1 N2 N3] = S.name
N1 =
   Ed Plum
N2 =
   Toni Miller
N3 =
   Jerry Garcia
```

ダイナミックなフィールド名

構造体のデータにアクセスする最も一般的な方法は、参照するフィールド名を指定することです。構造体データにアクセスする他の方法は、ダイナミックなフィールド名を使うことです。こうした名前は、MATLAB により実行時に評価される可変表現としてフィールドを表します。以下のドットとかっこの構文は、ダイナミックなフィールド名を表現します。

```
structName.(expression)
```

MATLAB の標準のインデックス構文を使ってこのフィールドにインデックスを付けることができます。たとえば、フィールド名の式を評価し、7 行目の 1 列から 25 列のフィールドの値を得るには、以下を実行します。

```
structName.(expression)(7,1:25)
```

ダイナミックなフィールド名の例 – 下記の関数 avgscore は、テストの平均点を計算し、ダイナミックなフィールド名を使って testscores 構造体から情報を取得します。

```
function avg = avgscore(testscores, student, first, last)
for k = first:last
   scores(k) = testscores.(student).week(k);
end
avg = sum(scores)/(last - first + 1);
```

ダイナミックなフィールドである student に異なる値を与えて次の関数を実行することができます。最初に、25 週間の得点を含む構造体を初期化します。

```
testscores.Ann_Lane.week(1:25) = ...
  [95 89 76 82 79 92 94 92 89 81 75 93 ...
   85 84 83 86 85 90 82 82 84 79 96 88 98];

testscores.William_King.week(1:25) = ...
  [87 80 91 84 99 87 93 87 97 87 82 89 ...
   86 82 90 98 75 79 92 84 90 93 84 78 81];
```

ここで、ダイナミックなフィールド名を使用して、実行時に testscores 構造体に学生の名前のフィールドを指定し、avgscore を実行します。

```
avgscore(testscores, 'Ann_Lane', 7, 22)
ans =
   85.2500

avgscore(testscores, 'William_King', 7, 22)
ans =
   87.7500
```

3

数学

線形代数

この節の内容...
MATLAB 環境の行列 (p. 3-2)
線形方程式 (p. 3-11)
逆行列と行列式 (p. 3-20)
因数分解 (p. 3-24)
べき乗と指数 (p. 3-32)
固有値 (p. 3-35)
特異値プロット (p. 3-38)

MATLAB 環境の行列

- 行列の作成 (p. 3-2)
- 行列の加算と減算 (p. 3-4)
- ベクトル積と転置 (p. 3-5)
- 行列の乗算 (p. 3-7)
- 単位行列 (p. 3-8)
- クロネッカー テンソル積 (p. 3-9)
- ベクトルと行列のノルム (p. 3-9)
- 線形代数関数のマルチスレッド計算 (p. 3-10)

行列の作成

MATLAB 環境は行列を使って、2 次元グリッドに配置した実数または複素数の変数を示します。"配列" とは一般的に数値のベクトル、行列または高次元グリッドです。MATLAB のすべての配列は四角形になり、すべての次元の成分は同じ長さになります。

Symbolic Math Toolbox™ というソフトウェアは、MATLAB の機能を数式の行列に拡張します。

MATLAB には、さまざまな行列を作成する関数が用意されています。そのうちの 2 つの関数を、この章全体を通して使用する 3 行 3 列の行列の例の作成に使用します。はじめに対称行列を作成します。

```
A = pascal(3)

A =
     1     1     1
     1     2     3
     1     3     6
```

非対称行列を作成します。

```
B = magic(3)

B =
     8     1     6
     3     5     7
     4     9     2
```

ランダムな整数で作成する 3 行 2 列の長方行列を作成します。

```
  C = fix(10*rand(3,2))

  C =
     9     4
     2     8
     6     7
```

列ベクトルは *m* 行 1 列の行列、行ベクトル は 1 行 *n* 列の行列、スカラー は 1 行 1 列の行列です。ステートメント

```
u = [3; 1; 4]

v = [2 0 -1]

s = 7
```

は、列ベクトル、行ベクトル、スカラーを作成します。

```
u =
       3
       1
       4

v =
       2     0    -1

s =
       7
```

行列の加算と減算

行列の加算と減算は、配列に対して要素単位で定義されます。A と B を加算し、その結果から A を減算すると B になることを示します。

```
A = pascal(3);
B = magic(3);
X = A + B

X =
       9     2     7
       4     7    10
       5    12     8

Y = X - A

Y =
       8     1     6
       3     5     7
       4     9     2
```

加算と減算では、2 つの行列が同じ次元であるか、またはいずれかがスカラーである必要があります。次元に整合性がない場合はエラーになります。

```
C = fix(10*rand(3,2)) X = A + C プラスの使用中にエラーが発生しました。行列の次元は一致しなけれ

w = 9     7     6
```

ベクトル積と転置

同じ長さの行ベクトルと列ベクトルの乗算は、順番を変えても計算結果は同じになります。結果は、スカラーすなわち「内積」、または行列すなわち「外積」のいずれかになります。

```
u = [3; 1; 4];
v = [2 0 -1];
x = v*u

x =
       2

X = u*v

X =
       6     0    -3
       2     0    -1
       8     0    -4
```

実数行列の転置は、a_{ij} と a_{ji} を交換します。MATLAB はアポストロフィ演算子 (') を使うと複素共役転置を行い、ドットアポストロフィ演算子 (.') を使うと転置を行います。すべての要素が実数の行列では、これら 2 つの演算子の処理は同じになります。

上記の行列 A は「対称」であるため、A' は A と等しくなります。B は対称ではないため B' とは等しくなりません。

```
B = magic(3);
X = B'

X =
       8     3     4
       1     5     9
       6     7     2
```

転置は行ベクトルを列ベクトルに置き換えます。

```
x = v'

x =
       2
       0
```

```
      -1
```

x と y が共に実数の列ベクトルの場合、積 x*y は定義できません。ただし、次の積

```
x'*y
```

と

```
y'*x
```

は定義でき、同じスカラー値を出力します。これらは使用頻度が高く、「内積」、「スカラー」積、「ドット」積と呼ばれています。

複素数ベクトルや行列 z に対して、z' は複素共役転置を表します。すなわち、各要素の虚数部の符号が逆になります。たとえば、

```
z = [1+2i 7-3i 3+4i; 6-2i 9i 4+7i]
z =
   1.0000 + 2.0000i   7.0000 - 3.0000i   3.0000 + 4.0000i
   6.0000 - 2.0000i        0 + 9.0000i   4.0000 + 7.0000i
```

このとき、以下のようになります。

```
z'
ans =
   1.0000 - 2.0000i   6.0000 + 2.0000i
   7.0000 + 3.0000i        0 - 9.0000i
   3.0000 - 4.0000i   4.0000 - 7.0000i
```

各要素の虚数部がその符号を保持し、共役を取らない複素転置は z.' で定義します。

```
z.'
ans =
   1.0000 + 2.0000i   6.0000 - 2.0000i
   7.0000 - 3.0000i        0 + 9.0000i
   3.0000 + 4.0000i   4.0000 + 7.0000i
```

複素ベクトルに対して 2 つのスカラー積 x'*y と y'*x は互いに複素共役で、スカラー積 x'*x は実数です。

行列の乗算

行列の乗算は、線形変換の構成を反映するように定義されており、線形方程式を簡潔に表現できます。行列積 C = AB は、*A* の列の次元が *B* の行の次元と等しいとき、またはどちらかがスカラーの場合に定義されます。*A* が *m* 行 *p* 列で、*B* が *p* 行 *n* 列の行列の場合、積 *C* は *m* 行 *n* 列の行列になります。積は、実際には MATLAB の for ループ、colon 表記、ベクトルのドット積を使って定義されます。

```
A = pascal(3);
B = magic(3);
m = 3; n = 3;
for i = 1:m
     for j = 1:n
        C(i,j) = A(i,:)*B(:,j);
     end
end
```

MATLAB では、アスタリスクで行列乗算を定義します。次の 2 つの例では、行列積が可換でないことを示します。すなわち *AB* は通常、*BA* と等しくありません。

```
X = A*B

X =
      15    15    15
      26    38    26
      41    70    39

Y = B*A

Y =
      15    28    47
      15    34    60
      15    28    43
```

行列は、列ベクトルを右から、行ベクトルを左から乗算します。

```
u = [3; 1; 4];
x = A*u

x =
       8
      17
```

```
      30

v = [2 0 -1];
y = v*B

y =
      12    -7    10
```

長方行列の乗算では、次元の整合性が合う必要があります。

```
C = fix(10*rand(3,2)); X = A*C

X = 17    19 31    41 51    70

Y = C*A

mtimes の使用中にエラーが発生しました。内部行列の次元は一致しなければなりません。
```

スカラーは、すべてに対して乗算できます。

```
s = 7;
w = s*v

w =
      14     0    -7
```

単位行列

一般的な数学表記では、大文字の *I* で単位行列を表します。単位行列は対角要素が 1 で、他の要素が 0 である任意のサイズの行列です。この行列は次元の整合性がある場合、*AI* = *A* かつ *IA* = *A* になります。MATLAB の以前のバージョンでは大文字と小文字を区別して認識していなかったため、単位行列に *I* を使用できませんでした。これは *i* を既に添字や複素数単位として使用していたためです。そこで、英語の語呂合わせで関数名を作成しました。関数

```
eye(m,n)
```

は *m* 行 *n* 列の単位長方行列を返し、eye(n) は *n* 行 *n* 列の単位正方行列を返します。

クロネッカー テンソル積

2 つの行列のクロネッカー積 kron(X,Y) は、X と Y の要素のすべての組み合わせの積で作成される大規模な行列です。X が *m* 行 *n* 列で、Y が *p* 行 *q* 列の場合、kron(X,Y) は *mp* 行 *nq* 列の行列になります。要素は次の順番で並べられます。

```
[X(1,1)*Y   X(1,2)*Y   . . .   X(1,n)*Y
                       . . .
 X(m,1)*Y   X(m,2)*Y   . . .   X(m,n)*Y]
```

クロネッカー積は 0 と 1 からなる行列を使って、繰り返し小さな行列のコピーを作成します。たとえば X が 2 行 2 列の行列であるとき、

```
X =
       1     2
       3     4
```

I = eye(2,2) が 2 行 2 列の単位行列とすると、2 つの行列

```
kron(X,I)
```

と

```
kron(I,X)
```

は、次のようになります。

```
       1     0     2     0
       0     1     0     2
       3     0     4     0
       0     3     0     4
```

と

```
       1     2     0     0
       3     4     0     0
       0     0     1     2
       0     0     3     4
```

ベクトルと行列のノルム

ベクトル *x* の *p* ノルムは次のように定義され、

$$\|x\|_p = \left(\sum |x_i|^p\right)^{1/p},$$

norm(x,p) で計算されます。これは、$p > 1$ を満たす任意の値で定義されていますが、実際に使う p の値は 1、2、および ∞ です。既定値は $p = 2$ で、「ユークリッド長」に対応しています。

```
v = [2 0 -1];
[norm(v,1) norm(v) norm(v,inf)]

ans =
       3.0000      2.2361      2.0000
```

行列 A の p ノルム

$$\|A\|_p = \max_x \frac{\|Ax\|_p}{\|x\|_p},$$

は $p = 1, 2, \infty$ に対して norm(A,p) で計算されます。ここでも既定値は $p = 2$ です。

```
C = fix(10*rand(3,2));
[norm(C,1) norm(C) norm(C,inf)]

ans =
      19.0000    14.8015    13.0000
```

線形代数関数のマルチスレッド計算

MATLAB は多数の線形関数と要素ごとに計算する数値関数のマルチスレッド計算をサポートしています。これらの関数は自動的にマルチスレッドで実行されます。関数や式に対し、複数の CPU を使用して計算を高速化するにはさまざまな条件があります。

1 関数は同時実行可能な部分へ簡単に分割できる処理を実行します。この分割部分は各処理間で通信をほとんど行わずに実行されるものでなければなりません。すなわちシーケンシャルな処理がほとんどないものを扱います。

2 データを分割し、複数の実行スレッドを管理する時間を含めても同時実行する価値があるように、データ サイズは十分に大きいものを扱います。たとえば配列が数千個以上の要素を含む場合は、ほとんどの関数に対して高速化の効果があります。

3 処理がメモリに拘束されないものを扱います。すなわち処理時間の大部分がメモリのアクセス時間にならないものを扱います。一般的には、シンプルな処理より複雑な処理をする関数の方が、高速化の効果があります。

行列の乗算 (X*Y) と行列のべき乗 (X^p) の処理は、大規模な倍精度配列 (10,000 個以上の要素) に対して計算負荷を増大させます。行列解析関数 det、rcond、hess、expm も同様です。

線形方程式

計算上の考慮事項

技術計算の中で最も重要な問題の 1 つは、線形方程式を解くことです。

行列表記では一般的な問題は次のような形式になります。[2 つの行列 *A* と *B* があるとき、*AX* = *B* または *XA* = *B* の条件のいずれかを満たす一意の行列 *X* は存在しますか。]

1 行 1 列の簡単な例を考えます。たとえば次の方程式について考えます。

$$7x = 21$$

上記に一意の解は存在しますか。

もちろん存在します。この方程式には一意の解 x = 3 が存在します。解は除算から簡単に得られます。

$$x = 21/7 = 3.$$

解を求める場合、7 の逆数すなわち $7^{-1} = 0.142857\ldots$ を計算し、この 7^{-1} に 21 を乗算する方法は通常"使いません"。この方法では余計な計算ステップが必要となり、7^{-1} を有限の小数値で打ち切ると精度も低下します。複数の未知数がある線形方程式にも同様の考え方で、MATLAB は逆行列を使わずにこのような方程式を解きます。

標準の数学的な表記ではありませんが、MATLAB ではスカラーの場合と同じ除算記号を使用して一般的な連立方程式の解を記述します。2 つの除算記号、スラッシュ / と、バックスラッシュ ¥ はそれぞれ 2 つの関数 mldivide と mrdivide に対応します。関数 mldivide と関数 mrdivide は、それぞれ未知の行列が係数行列の左または右から乗算されている場合に使用します。

X = B/A	行列方程式 $XA = B$ の解です。
X = A¥B	行列方程式 $AX = B$ の解です。

$AX = B$ または $XA = B$ の方程式の両辺を A で割ると考えます。係数行列 A は常に「分母」になります。

X = A¥B に対する次元の整合性の条件は、2 つの行列 A と B の行数が同じであることです。その結果、解 X は B と同じ列数になり、行数は A の列数と同じになります。X = B/A に対しては、行と列の処理が反転します。

実際には AX = B の形式の線形方程式の方が XA = B の形式より頻繁に使用します。そのためバックスラッシュはスラッシュよりも頻繁に使用されます。この節の以降ではバックスラッシュ演算子を中心に説明します。スラッシュ演算子の特性は次の等式から推定できます。

```
(B/A)' = (A'¥B')
```

係数行列 A は正方である必要はありません。A が m 行 n 列の行列の場合、次の 3 つのケースが考えられます。

m = n	正方システム。厳密解が得られます。
m > n	過決定システム。最小二乗解を求めます。
m < n	劣決定システム。最大で m 個のゼロでない構成要素をもつ基本解を求めます。

mldivide アルゴリズム

mldivide 演算子は、さまざまアルゴリズムを使ってさまざまなタイプの係数行列を取り扱います。実行するアルゴリズムは係数行列を調べて自動的に決められます。

三角行列の置換 – 関数 mldivide は 0 の要素を調べて三角行列であるかどうかを調べます。行列 A が三角行列である場合、MATLAB は三角行列用の処理を行い、解のベクトル x を計算します。A が三角行列を置換したものである場合、MATLAB は置換用のアルゴリズムを使います。

正方行列 – A が実数で正の対角要素をもつ対称行列である場合、MATLAB はコレスキー分解を実行します。コレスキー分解に失敗した場合は、対称な非正定値行列用の分解をします。A が上 Hessenberg 型である場合はガウスの消去法を使い、方程式を三角行列にします。A が正方行列であり、置換された三角行列ではなく、対称な正定値行列やヘッセンベルグ行列でもない場合、MATLAB は部分ピボットを使った LU 分解 (lu のリファレンス ページを参照) により、三角行列に分解します。

長方行列 – A が長方行列の場合、関数 mldivide は最小二乗解を返します。MATLAB は QR 分解 (qr のリファレンス ページを参照) を使って過決定システムを解きます。劣決定システムの場合、MATLAB は 0 要素数が最大の場合の解を返します。

関数 mldivide のリファレンス ページには、詳細なアルゴリズムの説明が記載されています。

一般解

線形方程式 $AX = b$ の一般解は、すべての解を示します。次のようにして一般解を求めることができます。

1. 対応する同次方程式 $AX = 0$ を解きます。null コマンドを使用して「null(A)」と入力します。$AX = 0$ の解空間の基底が返されます。解はすべて基底ベクトルの線形結合になります。
2. 非同次方程式 $AX = b$ の特殊解を導出します。

AX = b の解はすべて、ステップ 2 の AX = b の特殊解に、手順 1 の基底ベクトルの線形結合を加えたものとして表現できます。

この節の以降では、手順 2 の AX = b の特殊解の導出方法を説明します。

正方システム

一般的な例は、正方係数行列 A と右辺に列ベクトル b がある場合です。

特異でない係数行列 – 行列 A が特異でない場合、x = A¥b は b と同じサイズになります。たとえば、

```
A = pascal(3);
u = [3; 1; 4];
x = A¥u

x =
      10
     -12
       5
```

A*x が u と等価であることを確認してください。

A と B が正方で同じサイズの場合、X = A¥B も同じサイズになります。

```
B = magic(3);
X = A¥B

X =
      19    -3    -1
     -17     4    13
       6     0    -6
```

A*X が B と等価であることを確認してください。

上記 2 つの例は、厳密に整数の解になります。これは係数行列が pascal(3) であり、この行列の行列式が 1 であるためです。

特異な係数行列 – 正方行列 *A* が線形に独立した列をもっていない場合を「特異」といいます。*A* が特異な場合、AX = B の解は存在しないかまたは一意ではありません。バックスラッシュ演算 A¥B は、A が特異に近い場合は警告を出力し、特異である場合はエラーを出力します。

A が特異で *AX* = *b* が解をもつ場合は、次のように入力することで一意ではない特殊解を導出できます。

```
P = pinv(A)*b
```

P は *A* の疑似逆行列です。*AX* = *b* が厳密解をもたない場合、pinv(A) は最小二乗解を返します。

たとえば、

```
A = [ 1     3     7
     -1     4     4
      1    10    18 ]
```

が特異であることは以下のように入力することでわかります。

```
det(A)

ans =
     0
```

メモ: 関数 pinv を使用して係数行列が四角形になる方程式を解く方法の詳細は、「疑似逆行列 (p. 3-22)」を参照してください。

厳密解

b =[5;2;12] に対して、方程式 *AX* = *b* は次のような厳密解をもちます。

```
pinv(A)*b

ans =
    0.3850
   -0.1103
    0.7066
```

以下のように入力すると、pinv(A)*b が厳密解であることを確かめることができます。

```
A*pinv(A)*b

ans =
    5.0000
    2.0000
   12.0000
```

最小二乗解

ただし、b = [3;6;0] の場合、*AX* = *b* は厳密解をもちません。この例の場合、pinv(A)*b は最小二乗解を返します。以下のように入力すると、

```
A*pinv(A)*b

ans =
   -1.0000
    4.0000
    2.0000
```

元のベクトル b には戻りません。

AX = b が厳密解であるかどうかは、拡大係数行列 [A b] を階段型行列に変換することによって判断できます。例では以下のようになります。

```
rref([A b])
ans =
    1.0000         0    2.2857         0
         0    1.0000    1.5714         0
         0         0         0    1.0000
```

最終行は最後の要素以外はすべてゼロであるため、方程式は解をもちません。この例の場合、pinv(A) は最小二乗解を返します。

過決定システム

連立線形方程式の過決定システムは、実験データの曲線近似などで使われています。ここでは次の例を考えます。値 *y* は時刻 *t* で測定され、観測値は次のとおりです。

t	y
0.0	0.82
0.3	0.72
0.8	0.63
1.1	0.60
1.6	0.55
2.3	0.50

このデータを次のステートメントで MATLAB に入力します。

```
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
```

このデータを指数的に減衰させる次の関数でモデル化します。

$$y(t) = c_1 + c_2 e^{-t}.$$

この方程式は定数 1 の定数ベクトルと成分 e^{-t} のベクトルを線形結合して、ベクトル y を近似することを表します。未知係数 c_1 と c_2 は、モデルとデータの偏差の二乗和を最小にする最小二乗を行うことで計算します。2 個の未知数をもつ 6 個の方程式が、6 行 2 列の行列で表されます。

```
E = [ones(size(t)) exp(-t)]

E =
       1.0000    1.0000
       1.0000    0.7408
       1.0000    0.4493
       1.0000    0.3329
       1.0000    0.2019
       1.0000    0.1003
```

最小二乗解はバックスラッシュ演算子で求めます。

```
c = E¥y

c =
       0.4760
       0.3413
```

別の表記では、データの最小二乗近似は次のように表せます。

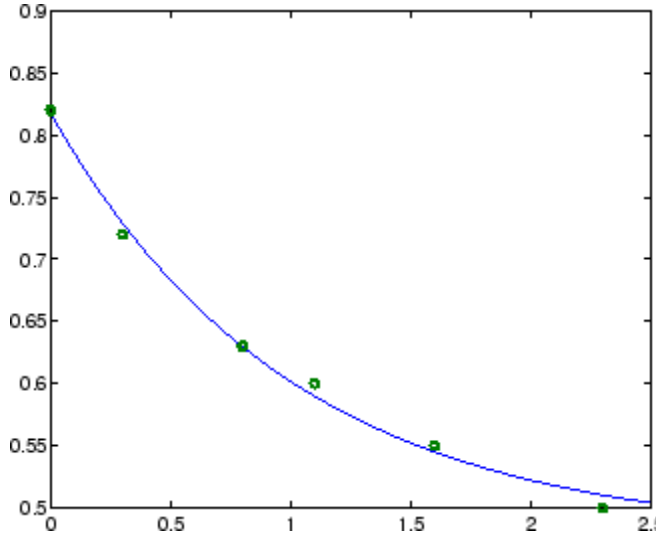
$$y(t) = 0.4760 + 0.3413\ e^{-t}.$$

次のステートメントは等間隔な時刻 t についてモデルを実行し、元のデータと共に結果をプロットします。

```
T = (0:0.1:2.5)';
Y = [ones(size(T)) exp(-T)]*c;
```

```
plot(T,Y,'-',t,y,'o')
```

E*c は y と厳密に一致しませんが、その差は元データに含まれる測定誤差よりもかなり小さくなります。



長方行列 *A* は線形に独立していない列があると、「ランク落ち」になります。*A* がランク落ちの場合、*AX = B* の最小二乗解は一意になりません。バックスラッシュ演算子 A¥B は、A がランク落ちの場合に警告を出力し、システムが解をもたない場合は最小二乗解を作成し、システムが解を無限にもつ場合は基本解を作成します。

format コマンドを使用して、有理形式で解を表示します。特殊解は次のステートメントで得られます。

```
format rat
p = R¥b
p =
        0
       -3/7
        0
       29/7
```

R(:,2) は R の最大ノルムをもつ列であるため、ゼロ以外の要素の 1 つは p(2) になります。R(:,2) を取り除くと R(:,4) が最大ノルムをもつため、他のゼロ以外の要素は p(4) になります。

劣決定システムの完全解はヌル空間の任意のベクトルを付加することによって特性化され、有理数で表示するオプションを関数 null に入力して求めることができます。

```
Z = null(R,'r')
Z =
      -1/2          -7/6
      -1/2           1/2
       1             0
       0             1
```

R*Z がゼロになります。任意のベクトル x は任意のベクトル q に対して

```
x = p + Z*q
```

となり、R*x = b を満たします。

連立線形方程式のマルチスレッド計算

MATLAB は多数の線形関数と要素ごとに計算する数値関数のマルチスレッド計算をサポートしています。これらの関数は自動的にマルチスレッドで実行されます。関数や式に対し、複数の CPU を使用して計算を高速化するにはさまざまな条件があります。

1. 関数は同時実行可能な部分へ簡単に分割できる処理を実行します。この分割部分は各処理間で通信をほとんど行わずに実行されるものでなければなりません。すなわちシーケンシャルな処理がほとんどないものを扱います。

2. データを分割し、複数の実行スレッドを管理する時間を含めても同時実行する価値があるように、データ サイズは十分に大きいものを扱います。たとえば、配列が数千個以上の要素を含む場合は、ほとんどの関数に対して高速化の効果があります。

3. 処理がメモリに拘束されないものを扱います。すなわち処理時間の大部分がメモリのアクセス時間にならないものを扱います。一般的には、シンプルな処理より複雑な処理をする関数の方が、高速化の効果があります。

関数 inv、lscov、linsolve、mldivide はマルチスレッド計算を使うと、大規模な倍精度配列 (10,000 個以上の要素) に対する計算を大幅に高速化します。

線形方程式を解く反復法

係数行列 A が大規模でスパース性がある場合、分解する方法は一般に効果がありません。反復法は近似解を生成できます。MATLAB は大規模でスパース性のある行列を扱う、いくつかの反復法を用意しています。

pcg
: 前処理を使用した共役勾配法。この方法はエルミート正定係数行列 A に適しています。

bicg
: 双共役傾斜法

bicgstab
: 双共役傾斜安定化法

bicgstabl
: BiCGStab(l) 法

cgs
: 共役勾配二乗法

gmres
: 一般化最小残差法

lsqr
: LSQR 法

minres
: 最小残差法。この方法はエルミート係数行列 A に適しています。

qmr
: 準最小残差法

symmlq
: 対称 LQ 法

tfqmr
: 転置なし準最小残差法

逆行列と行列式

- はじめに (p. 3-21)
- 疑似逆行列 (p. 3-22)

はじめに

A が正方で特異でない場合、方程式 AX = I と XA = I は同じ解 X となります。この解は *A* の逆行列と呼ばれ、A^{-1} で表し、関数 inv で計算できます。

行列の "行列式" は理論的な考察や数式計算に役立ちますが、スケーリングや丸め誤差の特性によって数値計算の信頼性は低下します。この条件の下で、関数 det は正方行列の行列式を計算します。

```
A = pascal(3)

A =
       1     1     1
       1     2     3
       1     3     6
d = det(A)
X = inv(A)

d =
       1

X =
       3    -3     1
      -3     5    -2
       1    -2     1
```

A は対称で整数要素をもち、行列式が 1 であるため逆行列も整数要素になります。ただし、

```
B = magic(3)

B =
       8     1     6
       3     5     7
       4     9     2
d = det(B)
X = inv(B)

d =
    -360
```

```
X =
    0.1472   -0.1444    0.0639
   -0.0611    0.0222    0.1056
   -0.0194    0.1889   -0.1028
```

上記のように、X の要素を詳細に調べたり、format rat を使用すると、これらは 360 で割った整数であることがわかります。

A が正方で特異でない場合に丸め誤差を無視すると、理論的には X = inv(A)*B は X = A¥B と等価で、Y = B*inv(A) は Y = B/A と等価です。ただし、バックスラッシュとスラッシュを使う計算を推奨します。この計算方法を使うと、計算時間が短く、使用メモリが少なく、かつエラーの検出が容易であるという利点があるためです。

疑似逆行列

長方行列は、逆行列または行列式をもたないことがあります。少なくとも方程式 AX = I と XA = I のいずれかは解をもちません。逆行列に対する部分的な置換は「Moore-Penrose 疑似逆行列」によって与えられ、関数 pinv で計算されます。

```
format short
C = fix(10*gallery('uniformdata',[3 2],0));
X = pinv(C)

X =
    0.1159   -0.0729    0.0171
   -0.0534    0.1152    0.0418
```

行列

```
Q = X*C

Q =
    1.0000    0.0000
    0.0000    1.0000
```

は 2 行 2 列の単位行列になりますが、行列

```
P = C*X

P =
    0.8293   -0.1958    0.3213
```

```
   -0.1958    0.7754    0.3685
    0.3213    0.3685    0.3952
```

は 3 行 3 列の単位行列ではありません。ただし P は対称で P*C は C に等しく、X*P は X に等しいという点では、P は空間の一部で単位行列のように機能します。

ランク落ちのシステムに対する解法 – A が *m* > *n* の m 行 *n* 列で、フル ランクが *n* の場合、次の 3 つの各ステートメントは理論的には同じ最小二乗解 x を計算します。

```
x = A¥b
x = pinv(A)*b
x = inv(A'*A)*A'*b
```

ただし、バックスラッシュ演算子の方が高速に処理できます。

ただし A がランク落ちの場合、最小二乗問題の解は一意ではなくなります。次の値を最小にするベクトル x は多数あります。

```
norm(A*x -b)
```

x = A¥b によって計算される解が基本解です。この解は *r* を A のランクとすると、最大 *r* 個のゼロ以外の要素をもちます。x = pinv(A)*b によって計算される解は norm(x) を最小にするため、最小ノルム解といいます。A'*A が特異であるため、x = inv(A'*A)*A'*b を使って解を計算しても失敗します。

さまざまな解法例を示します。

```
A = [ 1  2  3
      4  5  6
      7  8  9
     10 11 12 ];
```

上記は、ランク落ちです。2 番目の列は、1 番目の列と 3 番目の列の平均になっています。以下の場合、

```
b = A(:,2)
```

が 2 番目の列の場合、A*x = b になる解は x = [0 1 0]' になります。ただし、この x を計算する方法はありません。バックスラッシュ演算子は次の結果を出力します。

```
x = A¥b
```

```
警告: ランク落ち、rank = 2、tol = 1.4594e-014.  x = 0.5000  0  0.5000
```

この解は 2 つのゼロ以外の要素をもちます。疑似逆行列のアプローチは次のようになります。

```
y = pinv(A)*b

y =
       0.3333
       0.3333
       0.3333
```

ランク落ちに対する警告はありません。ただし norm(y) = 0.5774 は norm(x) = 0.7071 よりも小さくなります。最後に、

```
z = inv(A'*A)*A'*b
```

は失敗します。

```
警告: 行列が特異値に近いか、スケーリングが正しくありません。結果が不正確になる可能性があります
```

因数分解

はじめに

この節で説明する 3 つの因子分解では、対角要素の上部または下部の要素がすべてゼロである「三角行列」を使います。三角行列を含む線形方程式は、「前進代入」または「後退代入」のいずれかを使うと、簡単かつ高速に解くことができます。

コレスキー分解

コレスキー分解は、対称行列を三角行列と転置行列との積として表現します。

$$A = R'R,$$

ここで、*R* は上三角行列です。

対称行列すべてがこの方法で分解できるわけではなく、適用できる行列は正定行列と呼ばれています。この行列は、*A* の対角要素がすべて正で、非対角要素が「大きすぎない」行列です。パスカル行列を例に行列分解を行います。この章全体を通して行列の例 A は 3 行 3 列のパスカル行列を扱いますが、ここでは一時的に 6 行 6 列の行列を作成します。

```
A = pascal(6)

A =
     1     1     1     1     1     1
     1     2     3     4     5     6
     1     3     6    10    15    21
     1     4    10    20    35    56
     1     5    15    35    70   126
     1     6    21    56   126   252
```

A の要素は二項係数になります。各要素は上と左の要素の和になります。コレスキー分解は次のようになります。

```
R = chol(A)

R =
     1     1     1     1     1     1
     0     1     2     3     4     5
     0     0     1     3     6    10
     0     0     0     1     4    10
     0     0     0     0     1     5
     0     0     0     0     0     1
```

結果の要素は再び二項係数になります。R'*R が A に等しいということは、A が二項係数の積の和を含んでいることを示しています。

メモ: コレスキー分解は複素数行列にも適用できます。コレスキー分解を行った複素数行列は A′ = A を満たし、"エルミート正定値行列" といわれています。

コレスキー分解を使用すると、次の線形方程式

$$Ax = b$$

を次の式で置き換えることができます。

$$R'Rx = b.$$

バックスラッシュ演算子は三角行列を認識するため、次のように高速に解くことができます。

```
x = R¥(R'¥b)
```

A が *n* 行 n 列の行列の場合、chol(A) の計算の複雑度は $O(n^3)$ になりますが、バックスラッシュの計算の複雑度は $O(n^2)$ です。

LU 分解

LU 分解またはガウスの消去法は任意の正方行列 *A* を、下三角行列と上三角行列の置換行列の積として表します。

$$A = LU,$$

ここで L は対角要素に 1 をもつ下三角行列を並べ替えたもので、*U* は上三角行列を並べ替えたものです。

並べ替えは理論上および計算上の面で必要になります。行列

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

は行を並べ替えないと、三角行列の積として表現することはできません。ただし、行列は次のようになります。

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix}$$

は三角行列の積として表すことができます。ε が小さい場合、因子の要素は大きくなり誤差も大きくなります。よってこの並べ替えは厳密には必要ありませんが、推奨されます。部分的なピボットは *L* の要素の大きさを 1 以下に制限し、*U* の要素が *A* の要素より大きくならないようにします。

たとえば、

```
[L,U] = lu(B)

L =
    1.0000         0         0
    0.3750    0.5441    1.0000
    0.5000    1.0000         0

U =
    8.0000    1.0000    6.0000
         0    8.5000   -1.0000
         0         0    5.2941
```

A を LU 分解すると、線形方程式

```
A*x = b
```

は次の式で高速に解くことができます。

```
x = U¥(L¥b)
```

行列式と逆行列は、次の関係を使って LU 分解から計算されます。

```
det(A) = det(L)*det(U)
```

と

```
inv(A) = inv(U)*inv(L)
```

`det(A) = prod(diag(U))` を使用して行列式を計算することもできますが、行列式の符号が逆になる場合もあります。

QR 分解

「直交」行列または直交性の列がある行列とは、各列が単位長さをもち相互に直交関係になる実数行列です。*Q* が直交であれば、

$$Q' Q = 1.$$

になります。最も簡単な直交行列は 2 次元の座標回転変換です。

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}.$$

複素数行列に対応する概念は「ユニタリ」です。直交ユニタリ行列は長さと角度が保存され誤差が大きくならないため、数値計算に効果的です。

直交分解または QR 分解は任意の長方行列を、直交行列またはユニタリ行列と上三角行列の積で表現します。列の置換も含まれます。

$A = QR$

または

$AP = QR,$

ここで、*Q* は直交行列またはユニタリ行列、*R* は上三角行列、*P* は置換行列です。

QR 分解には、フル サイズとエコノミー サイズ、列置換をもつものともたないものの 4 種類あります。

過決定線形システムは、列よりも多くの行をもつ長方行列を含んでいます。すなわち、*m* 行 *n* 列で *m* > *n* です。フル サイズの QR 分解は、*m* 行 *m* 列の正方直交行列 *Q* と *m* 行 *n* 列の上三角長方行列 *R* を生成します。

```
C=gallery('uniformdata',[5 4], 0);
[Q,R] = qr(C)

Q =

    0.6191    0.1406   -0.1899   -0.5058    0.5522
    0.1506    0.4084    0.5034    0.5974    0.4475
    0.3954   -0.5564    0.6869   -0.1478   -0.2008
    0.3167    0.6676    0.1351   -0.1729   -0.6370
    0.5808   -0.2410   -0.4695    0.5792   -0.2207


R =

    1.5346    1.0663    1.2010    1.4036
```

```
         0    0.7245    0.3474   -0.0126
         0         0    0.9320    0.6596
         0         0         0    0.6648
         0         0         0         0
```

多くの場合、*Q* の最後の m - n 列は *R* の下部にゼロを乗算することになるため必要ありません。したがってエコノミー サイズの QR 分解は、直交要素をもつ *m* 行 *n* 列の長方行列 *Q* と、正方の n 行 *n* 列の上三角行列 *R* を生成します。この 5 行 4 列の例ではあまり大きな影響はありませんが、非常に大きい長方行列の場合は、時間とメモリが節約されるため数値計算ではこの手法が非常に効果的になります。

```
[Q,R] = qr(C,0)
Q =

    0.6191    0.1406   -0.1899   -0.5058
    0.1506    0.4084    0.5034    0.5974
    0.3954   -0.5564    0.6869   -0.1478
    0.3167    0.6676    0.1351   -0.1729
    0.5808   -0.2410   -0.4695    0.5792


R =

    1.5346    1.0663    1.2010    1.4036
         0    0.7245    0.3474   -0.0126
         0         0    0.9320    0.6596
         0         0         0    0.6648
```

LU 分解とは異なり、QR 分解はピボットや置換は必要ありません。ただし 3 番目の出力引数を設定すると、オプションの列置換が出力され、特異性やランク落ちを調べる場合に便利です。因子分解の各ステップでは、分解していない残りの行列の列の中で最も大きなノルムをもつ列がそのステップでの基底として使われます。この計算方法により *R* の対角要素を小さい順に並べることができ、要素間の関係性を調べることによって各列の間に存在する線形依存性を明らかにすることができます。この簡単な例では、C の 2 列目は 1 列目のノルムよりも大きいノルムをもちます。そこで 2つの列を交換します。

```
[Q,R,P] = qr(C)

Q =
   -0.3522    0.8398   -0.4131
   -0.7044   -0.5285   -0.4739
   -0.6163    0.1241    0.7777

R =
  -11.3578   -8.2762
         0    7.2460
         0         0

P =
     0     1
     1     0
```

エコノミー サイズと列の置換を組み合わせると、3 番目の引数は置換行列ではなく置換ベクトルになります。

```
[Q,R,p] = qr(C,0)

Q =
   -0.3522    0.8398
   -0.7044   -0.5285
   -0.6163    0.1241

R =
  -11.3578   -8.2762
         0    7.2460


p =
     2     1
```

QR 分解では過決定の線形方程式を等価な三角行列に変換します。次の式を見てみましょう。

```
norm(A*x - b)
```

は以下と等価です。

```
norm(Q*R*x - b)
```

直交行列の乗算はユークリッド ノルムを保存するため、この式は以下と同じです。

```
norm(R*x - y)
```

ここで y = Q'*b です。*R* の最後の *m*-*n* 行はゼロであるため、この式は 2 つに分けられます。

```
norm(R(1:n,1:n)*x - y(1:n))
```

と

```
norm(y(n+1:m))
```

A がフル ランクの場合は x に対して解くことができ、最初の式はゼロになります。次に 2 番目の式が残差のノルムを指定します。A がフル ランクでない場合は、R の三角構造から最小二乗問題に対する基本解を計算します。

行列分解のマルチスレッド計算

MATLAB は多数の線形関数と要素ごとに計算する数値関数のマルチスレッド計算をサポートしています。これらの関数は自動的にマルチスレッドで実行されます。関数や式に対し、複数の CPU を使用して計算を高速化するにはさまざまな条件があります。

1. 関数は同時実行可能な部分へ簡単に分割できる処理を実行します。この分割部分は各処理間で通信をほとんど行わずに実行されるものでなければなりません。すなわちシーケンシャルな処理がほとんどないものを扱います。

2. データを分割し、複数の実行スレッドを管理する時間を含めても同時実行する価値があるように、データ サイズは十分に大きいものを扱います。たとえば配列が数千個以上の要素を含む場合は、ほとんどの関数に対して高速化の効果があります。

3. 処理がメモリに拘束されないものを扱います。すなわち処理時間の大部分がメモリのアクセス時間にならないものを扱います。一般的には、シンプルな処理より複雑な処理をする関数の方が、高速化の効果があります。

関数 lu と関数 qr を使うと、大規模な倍精度配列 (10,000 個以上の要素) に対する計算を大幅に高速化します。

べき乗と指数

正の整数のべき乗

A が正方行列で p が正の整数の場合、A^p は A を p-1 回乗算します。以下に例を示します。

```
A = [1 1 1;1 2 3;1 3 6]

A =

     1     1     1
     1     2     3
     1     3     6

X = A^2

X =
     3     6    10
     6    14    25
    10    25    46
```

逆数と分数のべき乗

A が正方行列で特異でない場合、A^(-p) は inv(A) を p-1 回乗算します。

```
Y = A^(-3)

Y =

  145.0000 -207.0000   81.0000
 -207.0000  298.0000 -117.0000
   81.0000 -117.0000   46.0000
```

A^(2/3) のような分数のべき乗も計算可能です。結果は行列の固有値の分布によって異なります。

要素単位のべき乗

要素単位のべき乗は .^ 演算子で行われます。以下に例を示します。

```
X = A.^2

A =
     1     1     1
     1     4     9
     1     9    36
```

指数

関数

```
sqrtm(A)
```

は、正確なアルゴリズムを使って A^(1/2) を計算します。関数 sqrtm の m は、A.^(1/2) と同じように要素単位の積を求める関数 sqrt(A) と、この関数を区別するために付けられています。

定数係数線形常微分方程式は、次のように表現されます。

$$dx/dt = Ax,$$

ここで $x = x(t)$ は t の関数ベクトルで、A は t に対して独立した行列です。解は指数行列として表現されます。

$$x(t) = e^{tA}x(0).$$

関数

```
expm(A)
```

は指数行列を計算します。例として次の 3 行 3 列の係数行列を考えます。

```
A =
     0    -6    -1
     6     2   -16
```

```
    -5     20    -10
```

初期条件 x(0) は次のように考えます。

```
x0 =
     1
     1
     1
```

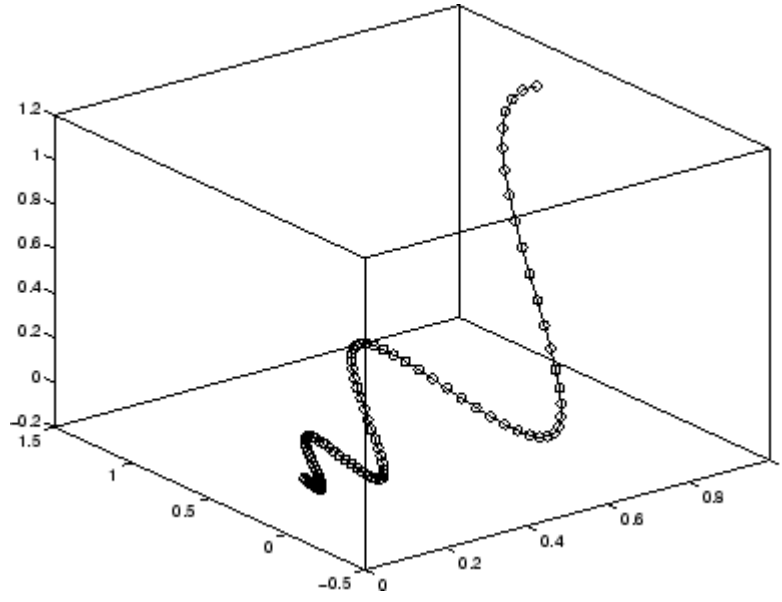
指数行列を使用して、$0 \le t \le 1$ の区間の 101 点で微分方程式の解 $x(t)$ を計算します。

```
X = [];
for t = 0:.01:1
   X = [X expm(t*A)*x0];
end
```

3 次元位相空間プロットを作成します。

```
plot3(X(1,:),X(2,:),X(3,:),'-o')
```

このプロットは解が原点に対して螺旋状になることを示しています。この動作は次の節で説明する係数行列の固有値に関連しています。



固有値

固有値分解

正方行列 A の "固有値" と "固有ベクトル" は、次の関係を満たすスカラー λ とゼロ以外のベクトル v で表します。

$$Av = \lambda v.$$

対角行列 Λ の対角要素に固有値を配置し、行列 V の列に対応する固有ベクトルを配置すると、次のような関係になります。

$$AV = V\Lambda.$$

V が特異でない場合、これは固有値分解になります。

$$A = V\Lambda V^{-1}.$$

前の節の「常微分方程式」の係数行列を考えます。

```
A =
     0    -6    -1
     6     2   -16
    -5    20   -10
```

ステートメント

```
lambda  =  eig(A)
```

は固有値を含む列ベクトルを作成します。この行列の例の固有値は複素数になります。

```
lambda =
    -3.0710
    -2.4645+17.6008i
    -2.4645-17.6008i
```

各固有値の実数部は負で、$e^{\lambda t}$ は t が増加するとゼロに近づきます。2 つの固有値のゼロ以外の虚数部 $\pm\omega$ は、微分方程式の解の振動構成成分 $\sin(\omega t)$ に関連します。

2 つの出力引数を設定すると eig は固有値ベクトルを計算し、対角に固有値を出力します。

```
[V,D] = eig(A)

V =
  -0.8326          0.2003 - 0.1394i   0.2003 + 0.1394i
  -0.3553         -0.2110 - 0.6447i  -0.2110 + 0.6447i
  -0.4248         -0.6930            -0.6930

D =
  -3.0710                  0                  0
        0          -2.4645+17.6008i         0
        0                  0           -2.4645-17.6008i
```

最初の固有ベクトルは実数で、他の 2 つのベクトルは互いに複素共役になっています。3 つのベクトルはすべてユークリッド長で、norm(v,2) が 1 になるように標準化されています。

V*D/V を簡潔に表現できる行列 V*D*inv(V) は、A の丸め誤差の範囲内に入ります。そして inv(V)*A*V または V¥A*V は D の丸め誤差の範囲内に入ります。

複数の固有値

固有ベクトル分解できない行列もあります。これらの行列は対角化できません。たとえば、

```
A = [ 6    12    19
     -9   -20   -33
      4     9    15 ]
```

この行列に対して、

```
[V,D] = eig(A)
```

は、以下の結果を出力します。

```
V =
```

```
   -0.4741   -0.4082   -0.4082
    0.8127    0.8165    0.8165
   -0.3386   -0.4082   -0.4082

D =

   -1.0000         0         0
         0    1.0000         0
         0         0    1.0000
```

$\lambda = 1$ の場合、2 つの固有値が存在します。V の 2 番目と 3 番目の列は同じものです。この行列に対して、完全に線形に独立した固有ベクトルの組は存在しません。

Schur 分解

MATLAB の高度な行列計算では固有値分解を必要とせず、代わりに Schur 分解を使います。

$$A = USU^T.$$

ここで U は直交行列で、S は 1 行 1 列および 2 行 2 列のブロックを対角要素にもつ上三角行列のブロックです。固有値は対角要素と S のブロックで表されます。一方、U の列は固有ベクトルよりも数値特性の良い基底を与えます。フル ランクではない行列の Schur 分解の例を示します。

```
[U,S]  =  schur(A)

U =
   -0.4741    0.6648    0.5774
    0.8127    0.0782    0.5774
   -0.3386   -0.7430    0.5774

S =
   -1.0000   20.7846  -44.6948
         0    1.0000   -0.6096
         0         0    1.0000
```

重根になる固有値は、S の下の 2 行 2 列のブロックに含まれます。

メモ: A が複素数の場合、schur は複素 Schur 型を返します。これは対角に A の固有値をもつ上三角行列です。

特異値プロット

長方行列 *A* の特異値および対応する特異ベクトルは、スカラー σ および *u* と *v* の 1 組のベクトルで表され、次の式を満たします。

$$Av = \sigma u$$
$$A^T u = \sigma v.$$

対角行列 Σ の対角に特異値を配置します。2 つの直行行列 *U* と *V* の列からなる特異ベクトルは次のような関係になります。

$$AV = U\Sigma$$
$$A^T U = V\Sigma.$$

U と *V* は直交であるため、これは特異値分解になります。

$$A = U\Sigma V^T.$$

m 行 *n* 列の非スパースな特異値分解は、*m* 行 *m* 列の *U*、*m* 行 *n* 列の Σ、*n* 行 *n* 列の *V* になります。言い換えれば *U* と *V* は共に正方で、Σ は *A* と同じサイズです。*A* の行数が列数より多いと、*U* の結果は非常に大きな行列になりますが、その列の大部分は Σ 内のゼロと乗算されます。このような場合エコノミー サイズで分解すると、m 行 n 列の *U*、*n* 行 *n* 列の Σ、*V* が作成され、計算時間もメモリ サイズも節約されます。

常微分方程式を表現する場合ように、あるベクトル空間内でマッピングを行う場合、固有値分解はその行列の解析に適切な方法になります。ただし、あるベクトル空間から他のベクトル空間へマッピングする行列を解析する場合は特異値分解が適切な方法になります。特異値分解は異なる次元についても扱えます。ほとんどの連立線形方程式は、この 2 番目の方法のカテゴリに入ります。

A が正方、対称、かつ正定の場合は、この固有値分解と特異値分解は等しくなります。ただし *A* が対称および正定でない場合、2 つの分解の結果のズレは大きくなります。特に実数行列の特異値分解は常に実数ですが、実数の非対称行列の固有値分解は複素数になることもあります。

例として行列を作成します。

```
A =
     9     4
     6     8
     2     7
```

A に対して非スパースな特異値分解は次のようになります。

```
[U,S,V] = svd(A)

U =

    0.6105   -0.7174    0.3355
    0.6646    0.2336   -0.7098
    0.4308    0.6563    0.6194


S =

   14.9359         0
         0    5.1883
         0         0


V =

    0.6925   -0.7214
    0.7214    0.6925
```

U*S*V' が丸め誤差の範囲内で A に等しいことが確かめられます。このような小さな行列の問題をエコノミー サイズで分解すると、わずかですが使用メモリ サイズを小さくできます。

```
[U,S,V] = svd(A,0)

U =

    0.6105   -0.7174
    0.6646    0.2336
    0.4308    0.6563
```

```
S =

   14.9359         0
         0    5.1883


V =

    0.6925   -0.7214
    0.7214    0.6925
```

U*S*V' が丸め誤差の範囲内で A に等しいことを再確認できます。

非線形関数の演算

この節の内容…

関数ハンドル

任意の MATLAB 関数にハンドルを作成し、関数を参照するための手段としてそのハンドルを使用することができます。関数ハンドルは、一般に、他の関数に引数リストの中で渡され、そこで、ハンドルを使った関数を実行したり計算します。

MATLAB の中で、関数ハンドルは、関数名の前に符号 @ を使って作成します。次の例は、関数 sin に対する関数ハンドルを作成し、それに、変数 fhandle を割り当てたものです。

```
fhandle = @sin;
```

関数名で関数を呼び出すのと同じ方法で、関数ハンドルによって関数を呼び出すことができます。その構文は次のとおりです。

```
fhandle(arg1, arg2, ...);
```

以下に示す関数 plot_fhandle は関数ハンドルとデータを受け取り、その関数ハンドルを使って *y* 軸のデータを生成しプロットします。

```
function plot_fhandle(fhandle, data)
plot(data, fhandle(data))
```

下に示す関数 sin へのハンドルと引数と共に plot_fhandle を呼び出すと、計算結果は正弦波プロットとして表示されます。

```
plot_fhandle(@sin, -pi:0.01:pi)
```

関数を引数とする関数

「関数を引数とする関数」と呼ばれる関数クラスは、スカラー変数の非線形関数と共に機能します。すなわち、1 つの関数が他の関数上で機能するものです。次のような事柄を含んでいます。

- 零点の検出
- 最適化
- 求積
- 常微分方程式

MATLAB では、非線形関数を、それを定義するファイルによって表します。たとえば、matlab/demos フォルダーに含まれている関数 humps の簡単なバージョンを以下に示します。

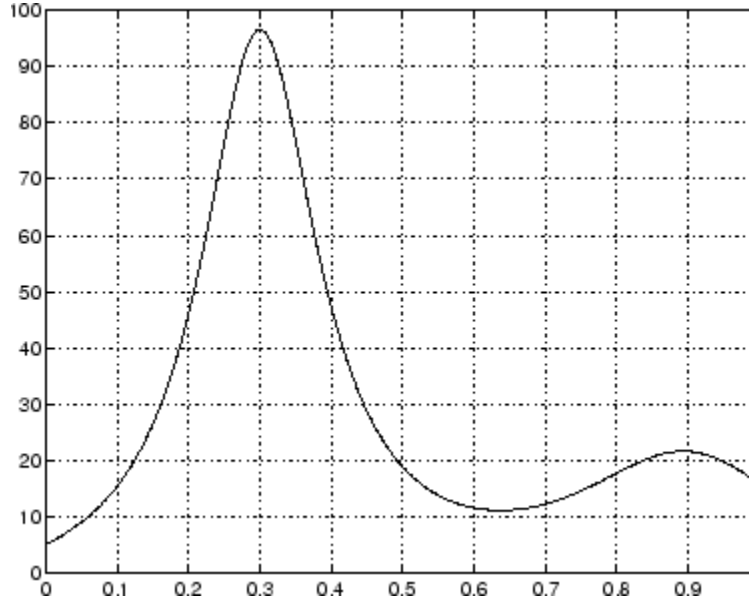
```
function y = humps(x)
y = 1./((x-.3).^2 + .01) + 1./((x-.9).^2 + .04) - 6;
```

区間 0 ≤ x ≤ 1 で、この関数を計算します。

```
x = 0:.002:1;
y = humps(x);
```

そして、関数をプロットします。

```
plot(x,y)
```



グラフは、関数が x = 0.6 近傍で局所的最小値をもつことを示しています。関数 fminsearch は、関数の最小値をもつ x の値、最小点を探索します。fminsearch への最初の引数は、最小化される関数の関数ハンドル、2 番目の引数は最小値の位置のラフな推定値です。

```
p = fminsearch(@humps,.5)
p =
    0.6370
```

最小点で、関数を計算するには、

```
humps(p)

ans =
   11.2528
```

数値解析は、定積分の数値的近似と常微分方程式の数値積分とを区別するため求積と積分の項を使います。MATLAB の求積法は、quad と quadl です。ステートメント

```
Q = quadl(@humps,0,1)
```

は、グラフ曲線の下の領域を計算し、その結果を出力します。

```
Q =
   29.8583
```

最終的に、グラフは、関数がこの区間で 0 にならないことを示しています。それで、次のステートメントで 0 の位置を探索します。

```
z = fzero(@humps,.5)
```

結果、区間の外に見つかりました。

```
z =
   -0.1316
```

多変量データ

MATLAB は、多変数統計データに対して列方向の解析を行います。データ セットの中の各々の列は 1 つの変数を表し、行は観測値です。(i,j) 番目の要素は、j 番目の変数の i 番目の観測値です。

たとえば、3 変数をもつデータを考えましょう。

- 心拍数
- 重み
- 週に行う運動時間

5 つの観測値に対して、結果の配列は、次のようになります。

```
D = [ 72          134          3.2
      81          201          3.5
      69          156          7.1
      82          148          2.4
      75          170          1.2 ]
```

最初の行は、患者 1 の心拍数、体重、運動時間を、2 番目の行は、患者 2 の、等を含みます。さて、このデータ セットに MATLAB のデータ解析関数を適用してみましょう。たとえば、各列の平均、標準偏差を計算します。

```
mu = mean(D), sigma = std(D)

mu =
      75.8        161.8          3.48

sigma =
    5.6303       25.499        2.2107
```

MATLAB の中で利用可能なデータ解析関数の一覧は、次のステートメントで得られます。

```
help datafun
```

Statistics Toolbox™ にアクセスしたい場合は、次のように入力してください。

```
help stats
```

データ解析

はじめに

データ解析には、いくつかの標準要素があります。

- 前処理 – 可能なモデルを同定するために、外れ値、欠損値、平滑化したデータを考える。
- まとめ – 全体的な位置、スケール、データの形状を記述する基本的な統計量を計算する。
- 可視化 – パターンと傾向を確認するために、データをプロットする。
- モデリング – 新しい値の予測に適したデータ傾向を詳しく記述する。

データ解析は、以下の 2 つの基本的な目的をもち、上記のことを行います。

1 正確な予測を行える単純なモデルを用いてデータのパターンを記述する。

2 モデルを導く変数間の関係を理解する。

この節では、MATLAB 環境で基本的なデータ解析を行う方法を説明します。

データの前処理

概要

データを適切な MATLAB コンテナー変数に読み込み、「不適切な」データと「適切な」データを並べ替えることによりデータ解析を始めます。これは、解析のその後の過程で意味のある結果が得られることを保証するための準備の手順です。

メモ: この節ではじめるデータ解析は、データのまとめ (p. 3-51)、データの可視化 (p. 3-55)、データのモデリング (p. 3-67) の節まで続きます。

データの読み込み

はじめに、count.dat のデータを読み込みます。

```
load count.dat
```

24 行 3 列の配列 count は、3 つの交差点それぞれが各列に対応し、ある一日の 1 時間ごとの交通量を行に含みます。

欠損データ

MATLAB では、NaN (Not a Number: 数値ではない) の値は通常、欠損データを表すために使用されます。NaN の値は欠損データのある変数が構造体 (以下では、3 つすべての交点で一致するインデックスをもつ 24 行 1 列のベクトル) を維持できるようにします。

関数 isnan を用いて、3 つ目の交差点におけるデータの NaN 値をチェックします。

```
c3 = count(:,3); % Data at intersection 3
c3NaNCount = sum(isnan(c3))
c3NaNCount =
     0
```

isnan は、c3 と同じサイズの論理ベクトルを出力します。このベクトルには、データ内の 24 要素のそれぞれについて、NaN の値が存在するか (1)、または存在しない (0) かが記入されています。この場合、論理値の総和が 0 になるので、データに値 NaN はありません。

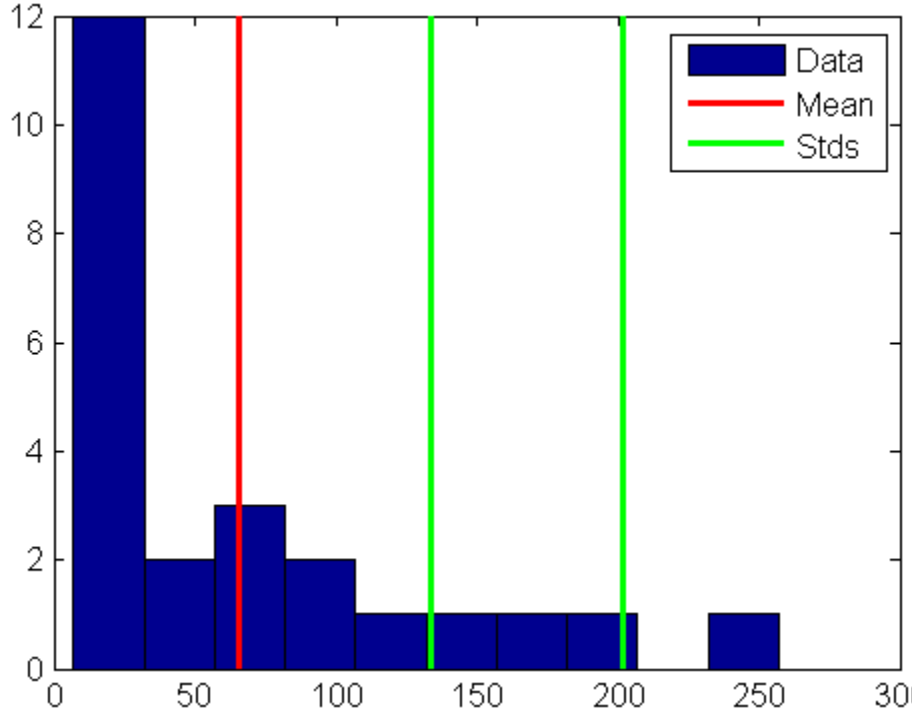
NaN の値は、外れ値 (p. 3-46) の節でデータに導入されます。

外れ値

外れ値は、それ以外のデータとは傾向が著しく異なるデータ値です。外れ値は、測定誤差により発生したものである場合もあれば、データの重要な特徴を表している場合もあります。外れ値を特定し、それらをどのように取り扱うかは、データとその原因を理解してから決めます。

外れ値と認識するための一般的な方法の 1 つは、平均 μ から一定数の標準偏差 σ 分離れた値を探すことです。以下のコードは、μ と $n = 1, 2$ の場合の $\mu + n\sigma$ のラインと共に、3 つ目の交差点でのデータのヒストグラムをプロットします。

```
bin_counts = hist(c3); % Histogram bin counts
N = max(bin_counts); % Maximum bin count
mu3 = mean(c3); % Data mean
sigma3 = std(c3); % Data standard deviation

hist(c3) % Plot histogram
hold on
plot([mu3 mu3],[0 N],'r','LineWidth',2) % Mean
X = repmat(mu3+(1:2)*sigma3,2,1);
Y = repmat([0;N],1,2);
plot(X,Y,'g','LineWidth',2) % Standard deviations
legend('Data','Mean','Stds')
hold off
```



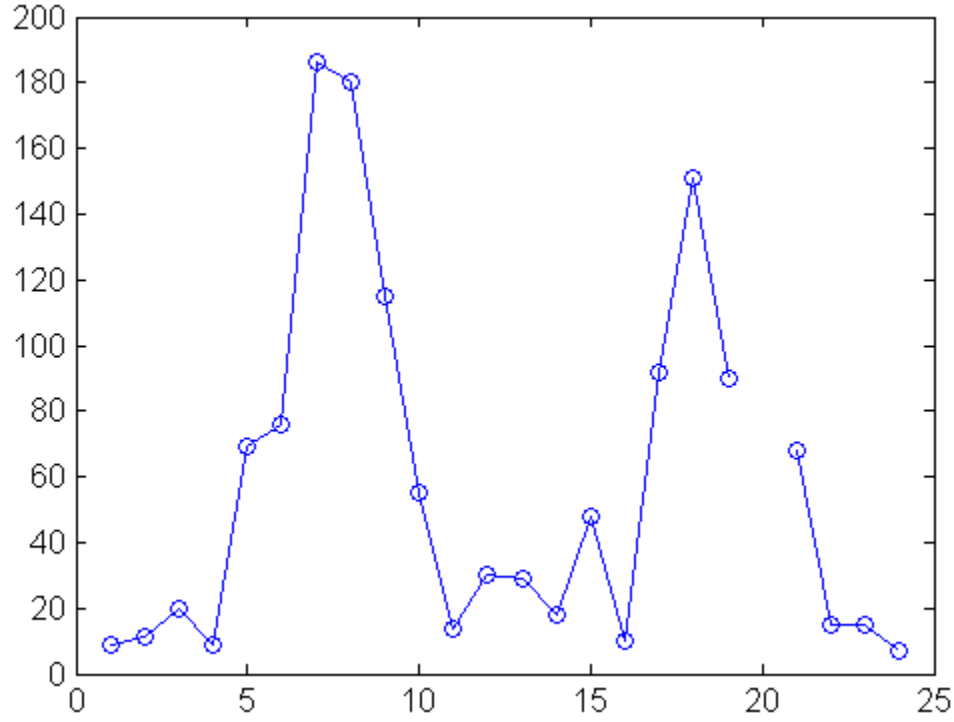
このプロットから、いくつかのデータは平均よりも 2 標準偏差以上超えていることがわかります。これらのデータを（特徴としてではなく）誤差として扱う場合は、これらを以下のように値 NaN で置き換えます。

```
outliers = (c3 - mu3) > 2*sigma3;
c3m = c3; % Copy c3 to c3m
c3m(outliers) = NaN; % Add NaN values
```

平滑化とフィルター処理

3 つ目の交差点でのデータ（外れ値 (p. 3-46)で除かれた外れ値をもつ）の時系列プロットは、以下のようになります。

```
plot(c3m,'o-')
hold on
```



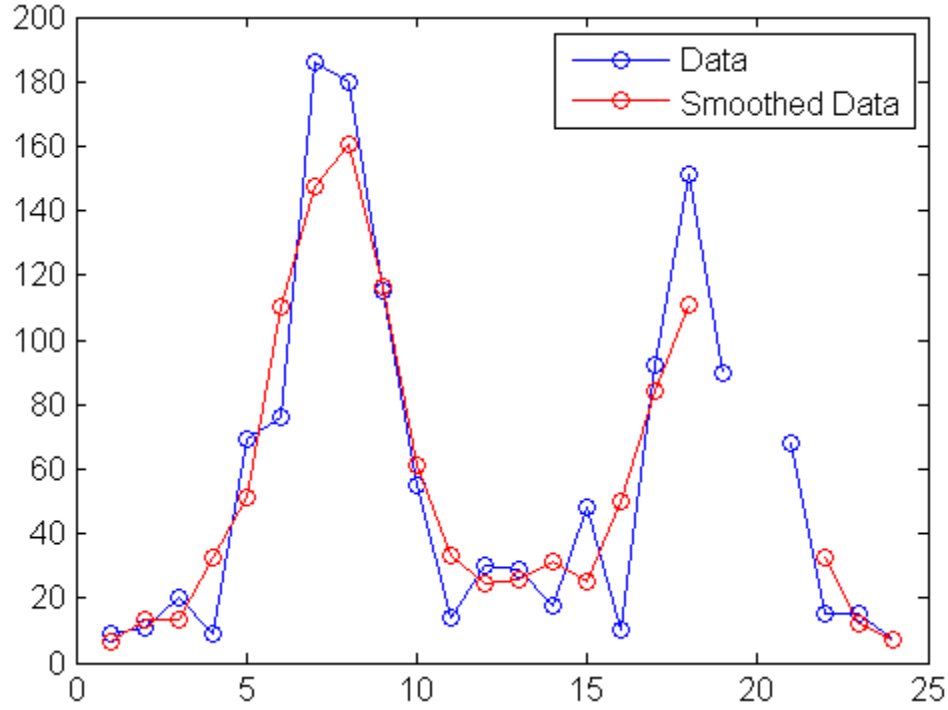
20 時間での NaN の値は、プロットにおいてギャップとして現れます。NaN の値の取り扱いは、MATLAB のプロット関数の特徴です。

ノイズのあるデータは、期待値がランダムに変動します。モデルを構築する前に、データの主な特徴を明らかにするためにデータを平滑化することをお勧めします。平滑化する際には、2 つの基本的な仮定をします。

- 予測変数（時間）と応答（交通量）の関係が滑らかである。
- 平滑化アルゴリズムの結果は、ノイズが減少したために、期待値のより適切な推定値となる。

MATLAB の関数 convn を用いて、データに対して単純移動平均平滑化を適用します。

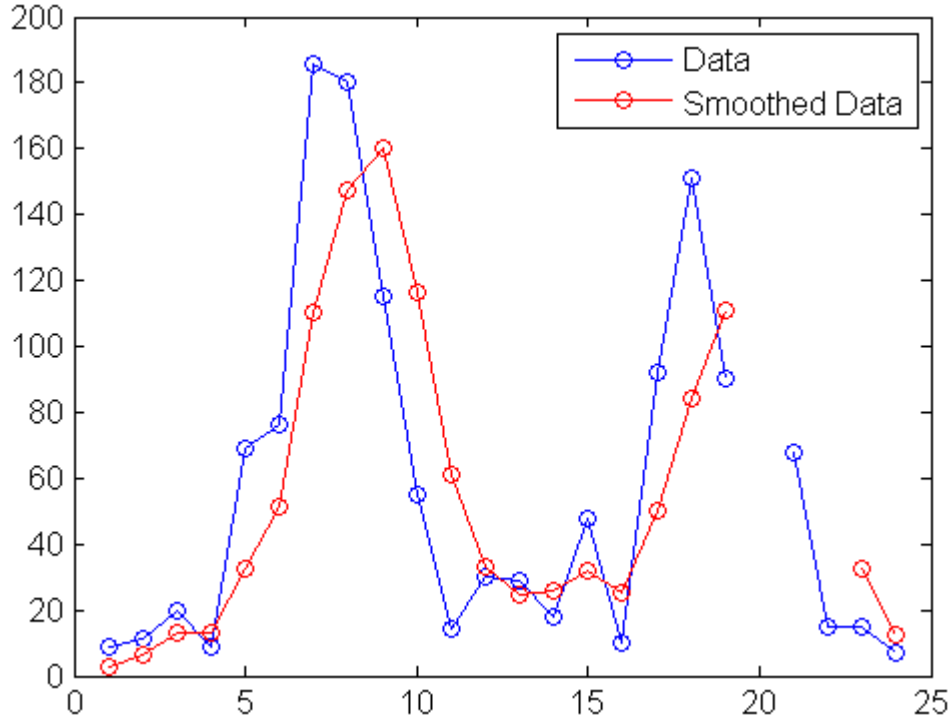
```
span = 3; % Size of the averaging window
window = ones(span,1)/span;
smoothed_c3m = convn(c3m,window,'same');

h = plot(smoothed_c3m,'ro-');
legend('Data','Smoothed Data')
```



平滑化の範囲は、変数 span を用いて制御します。平均化の計算では、平滑化ウィンドウがデータに値 NaN を含むときは、値 NaN を返すので、平滑化データのギャップのサイズが増加します。

関数 filter もデータを平滑化するために使用されます。

```
smoothed2_c3m = filter(window,1,c3m);

delete(h)
plot(smoothed2_c3m,'ro-');
```



平滑化データは、以前のプロットからシフトします。'same' パラメーターを用いた convn は、データと同じ長さで合成積の中央部分を出力します。filter は、データと同じ長さで合成積の初期部分を出力します。それ以外には、アルゴリズムは同じです。

平滑化は、予測の各値において応答値の分布の中心を推定します。このため、多くの近似アルゴリズムの基本的な仮定、予測の各値での誤差は独立であることが無効になります。したがって、モデルを同定するためには平滑化されたデータを使用しますが、モデルを近似するために平滑化されたデータを使用することは避けます。

データのまとめ

- 概要 (p. 3-52)
- 位置の測度 (p. 3-52)
- スケールの尺度 (p. 3-52)

概要

多くの MATLAB 関数によって、全体的な位置、スケール、データ サンプルの形状を把握することができます。

MATLAB の優れた機能の 1 つは、関数が 1 つのスカラー値の他に、データ配列全体を操作できることです。このことは、関数がベクトル化されているといわれます。ベクトル化によって、配列データを用いて効率的に問題を表すことや、ベクトル化された統計関数を用いた効率的な計算が可能になります。

位置の測度

「標準的な」値を見つけることによりデータ サンプルの位置を把握します。位置または「中心傾向」の通常の指標は、関数 mean、median、mode により計算されます。

```
load count.dat
x1 = mean(count)
x1 =
   32.0000   46.5417   65.5833

x2 = median(count)
x2 =
   23.5000   36.0000   39.0000

x3 = mode(count)
x3 =
    11     9     9
```

他の統計関数と同様に、上記の MATLAB 関数は、変数を列に保持しながら、観測値ごとにデータを行にまとめます。関数は、3 つの交差点それぞれでのデータの位置を 1 回の呼び出しで計算します。

スケールの尺度

データ標本のスケールまたは「ばらつき」を測定する多くの方法があります。MATLAB 関数 max、min、std、var は、通常の尺度を計算します。

```
dx1 = max(count)-min(count)
dx1 =
```

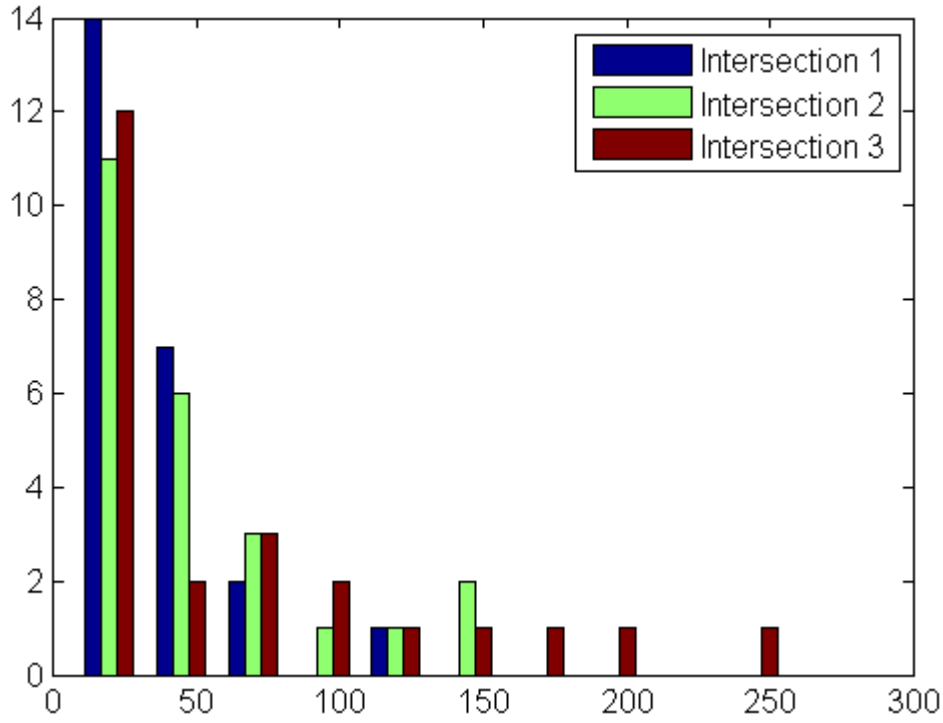
```
   107    136    250

dx2 = std(count)
dx2 =
   25.3703    41.4057    68.0281

dx3 = var(count)
dx3 =
  1.0e+003 *
    0.6437     1.7144     4.6278
```

他の統計関数と同様に、上記の MATLAB 関数は、変数を列に保持しながら、観測値ごとにデータを行にまとめます。関数は、3 つの交差点それぞれでのデータのスケールを 1 回の呼び出しで計算します。

分布の形状

分布の形状は、分布の位置またはスケールに比べて説明し難いです。MATLAB 関数 hist は、視覚的に把握できるヒストグラムをプロットします。

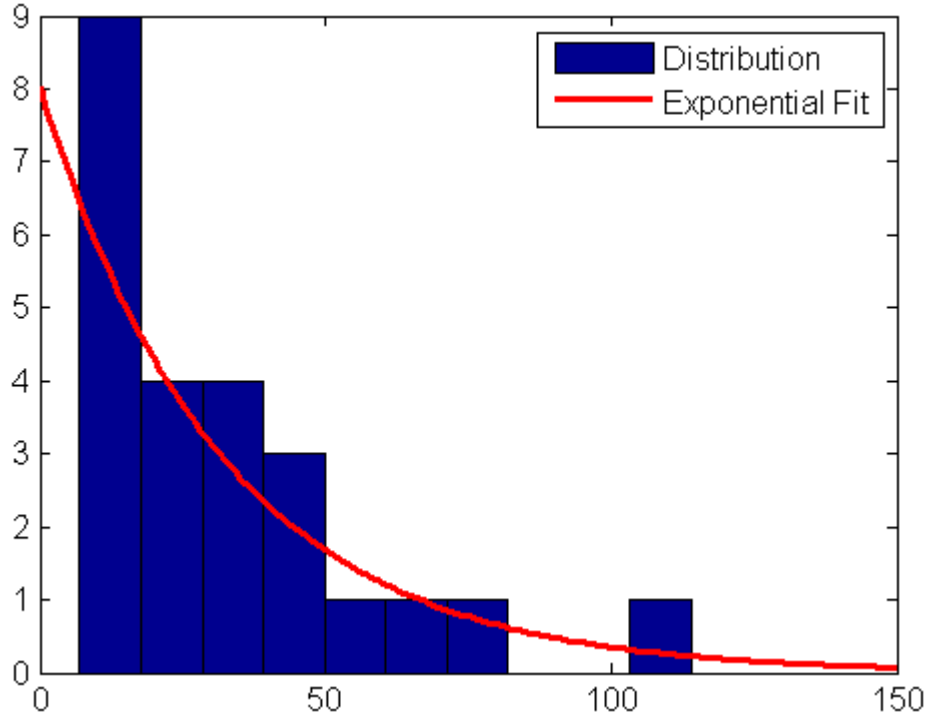
```
figure
hist(count)
legend('Intersection 1',...
       'Intersection 2',...
       'Intersection 3')
```



パラメトリック モデルでは、分布の形状の解析的に把握できます。データ平均のパラメーター mu をもつ指数分布は、交通量のデータに対する選択として適切です。

```
c1 = count(:,1); % Data at intersection 1
[bin_counts,bin_locations] = hist(c1);
bin_width = bin_locations(2) - bin_locations(1);
hist_area = (bin_width)*(sum(bin_counts));

figure
hist(c1)
hold on

mu1 = mean(c1);
exp_pdf = @(t)(1/mu1)*exp(-t/mu1); % Integrates
                                   % to 1
t = 0:150;
y = exp_pdf(t);
```

```
plot(t,(hist_area)*y,'r','LineWidth',2)
legend('Distribution','Exponential Fit')
```



データ分布を一般のパラメトリック モデルで近似する方法は、この節では取り扱いません。Statistics Toolbox ソフトウェアでは、分布パラメーターの最尤推定値を計算する関数が提供されています。

データの可視化

- 概要 (p. 3-56)
- 2 次元散布図 (p. 3-56)
- 3 次元散布図 (p. 3-58)
- 散布図配列 (p. 3-60)
- グラフ内のデータの探査 (p. 3-60)

概要

データのパターンと傾向を可視化するために、多くの種類の MATLAB グラフを利用できます。この節で述べる散布図によって、異なる交差点での交通量データの関係を可視化できます。データ探査ツールによって、グラフ上の個々のデータ点を参照したり、対話形式での操作が行えます。

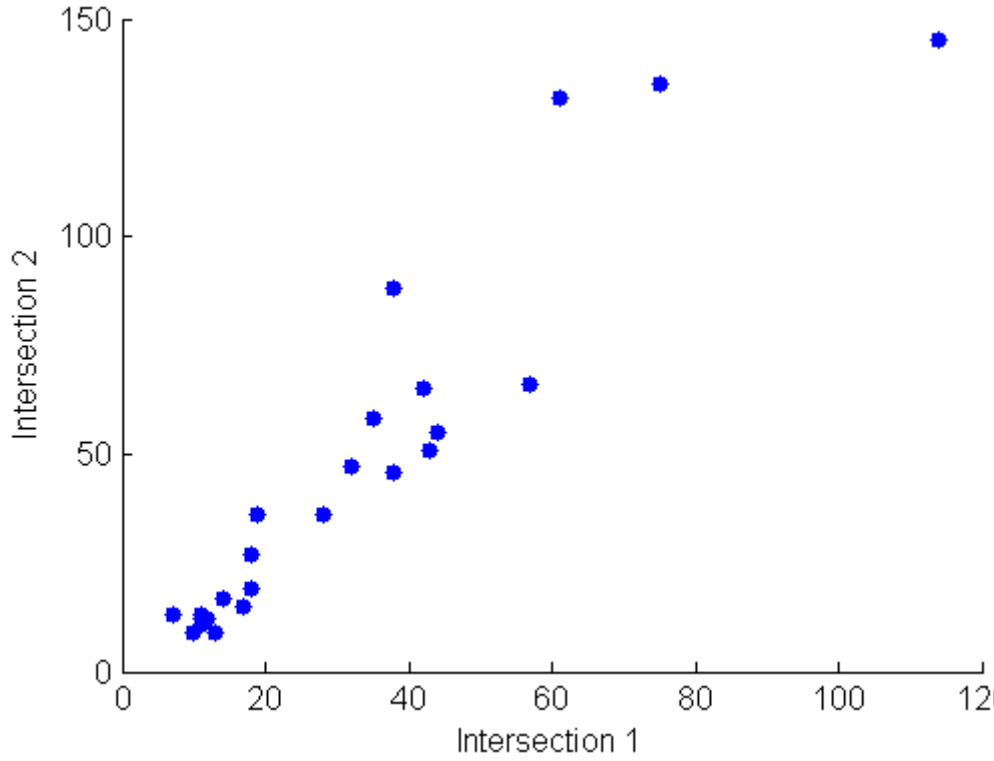
メモ: この節では、データのまとめ (p. 3-51)のデータ解析を続けます。

---

2 次元散布図

関数 scatter で作成される 2 次元散布図は、最初の 2 つの交差点における交通量の関係を示します。

```
load count.dat
c1 = count(:,1); % Data at intersection 1
c2 = count(:,2); % Data at intersection 2

figure
scatter(c1,c2,'filled')
xlabel('Intersection 1')
ylabel('Intersection 2')
```



関数 cov で計算される "共分散" は、2 変数間の線形関係の強さを測定します。散布図から、最小二乗ラインに沿ってデータがどれほど密にあるかを評価します。

```
C12 = cov([c1 c2])
C12 =
  1.0e+003 *
    0.6437    0.9802
    0.9802    1.7144
```

結果は、対称正方行列として表示されます。(*i*,*j*) 番目の位置の要素は、*i*番目の変数と*j*番目の変数の共分散です。*i*番目の対角要素は、*i*番目の変数の分散です。

共分散は、個々の変数の測定に用いる単位に依存するという不都合があります。共分散の値は、変数の標準偏差で除算することで +1 と -1 の間に基準化できます。関数 corrcoef では "相関係数" を計算します。

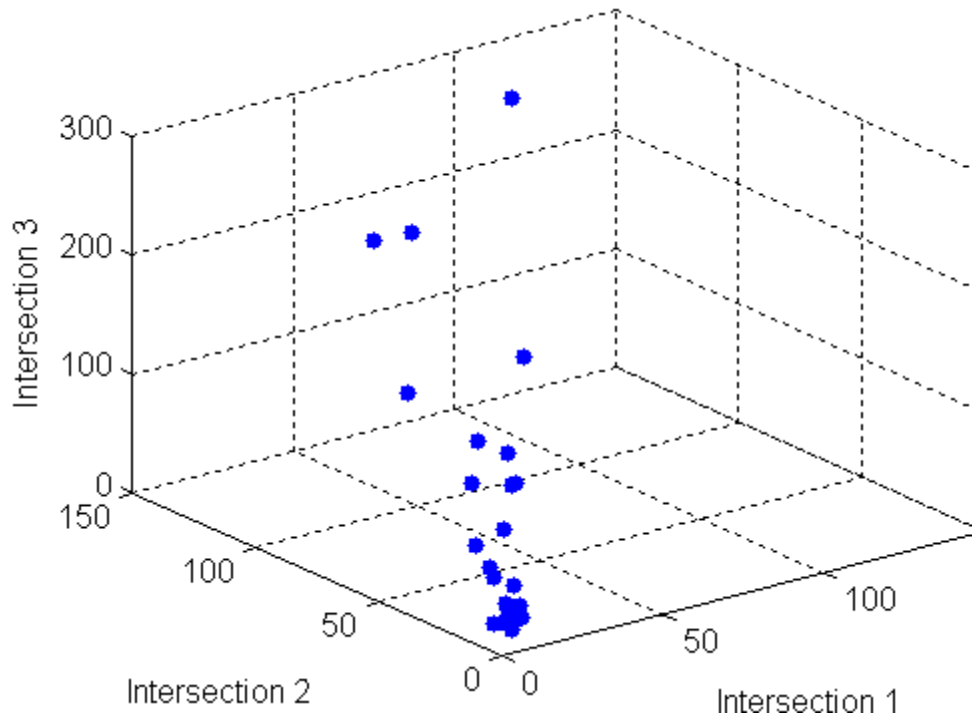
```
R12 = corrcoef([c1 c2])
R12 =
    1.0000    0.9331
    0.9331    1.0000

r12 = R12(1,2) % Correlation coefficient
r12 =
    0.9331

r12sq = r12^2 % Coefficient of determination
r12sq =
    0.8707
```

相関係数の値は基準化されているので、交差点の他の組の値と容易に比較できます。相関係数の 2 乗である "決定係数" は、最小二乗ラインからの変動を平均からの変動で除算したものです。したがって、決定係数は応答 (この場合、交差点 2 の交通量) における変動の部分であり、散布図から削除されるか、または最小二乗ラインによって統計的に説明されます。

3 次元散布図

関数 scatter3 で作成される 3 次元散布図は、3 つの交差点すべてにおける交通量間の関係を示します。前の手順で作成した変数 c1、c2、c3 を使用します。

```
figure
scatter3(c1,c2,c3,'filled')
xlabel('Intersection 1')
ylabel('Intersection 2')
zlabel('Intersection 3')
```



関数 eig で共分散行列の固有値を計算することにより、3 次元散布の変数間の線形関係の強さを測定します。

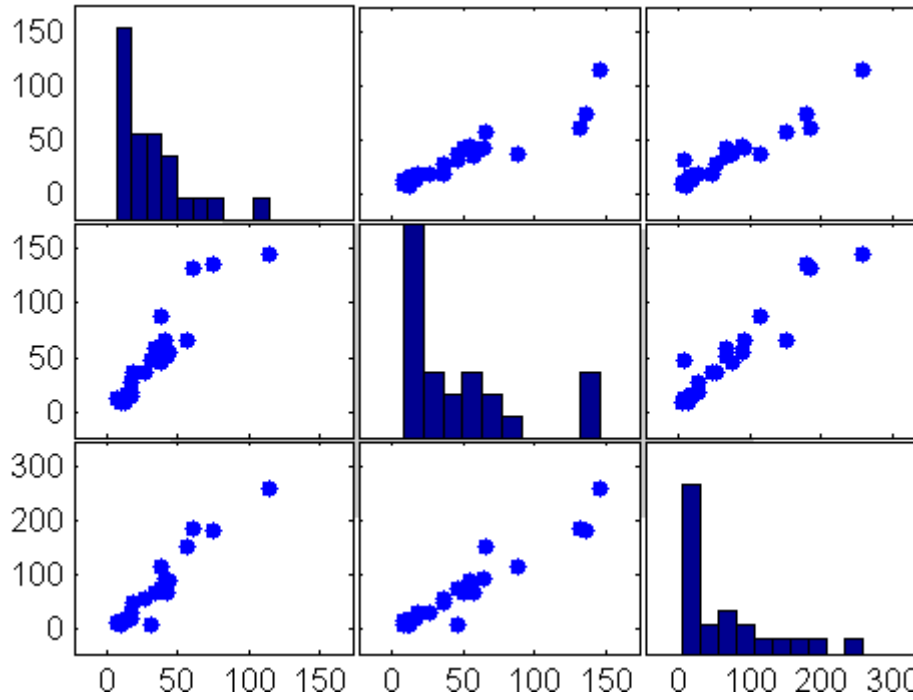
```
vars = eig(cov([c1 c2 c3]))
vars =
  1.0e+003 *
    0.0442
    0.1118
    6.8300

explained = max(vars)/sum(vars)
explained =
    0.9777
```

固有値は、データの "主成分" の分散です。変数 explained は、データの軸に沿った、第 1 主成分によって説明される変動の割合を測ります。2 次元散布に対する決定係数とは異なり、この尺度では予測変数と応答変数が区別されます。

散布図配列

関数 plotmatrix を使用して、交差点の複数の組間の関係を比較します。

```
figure
plotmatrix(count)
```



配列の (*i*,*j*) 番目の位置のプロットは、垂直軸上の *i* 番目と水平軸上の *j* 番目の変数の散布図です。*i* 番目の対角位置のプロットは、*i*番目の変数のヒストグラムです。

グラフ内のデータの探査

ほとんどの MATLAB グラフの観測値は、Figure ツール バーの 2 つのツールを利用してマウスで選択できます。

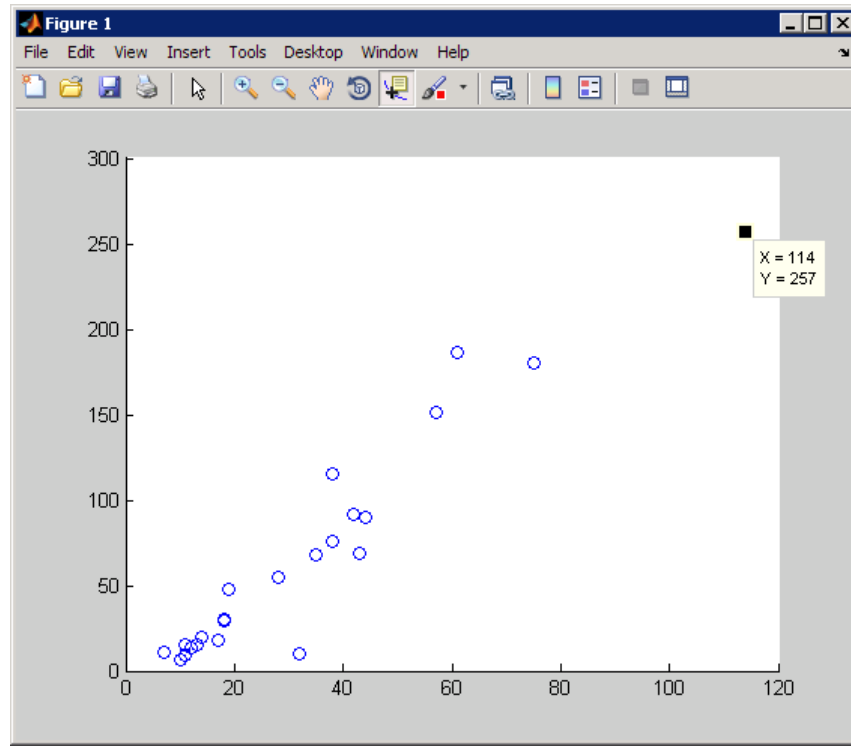
- データ カーソル 
- データのブラシ選択 

これらの各ツールはどれも探査モードでの利用となり、そこでグラフ上のデータ点を選択して値を確認したり、特定の観測値を含むワークスペース変数を作成したりできます。データのブラシ選択を使った場合、選択した観測値のコピー、削除または置き換えも可能です。

たとえば、count の第 1 列と第 3 列の散布図を作成します。

```
load count.dat
scatter(count(:,1),count(:,3))
```

[データ カーソル ツール]  を選択し、右端のデータ点をクリックします。点の *x* と *y* の値を表示するデータは、以下にあります。

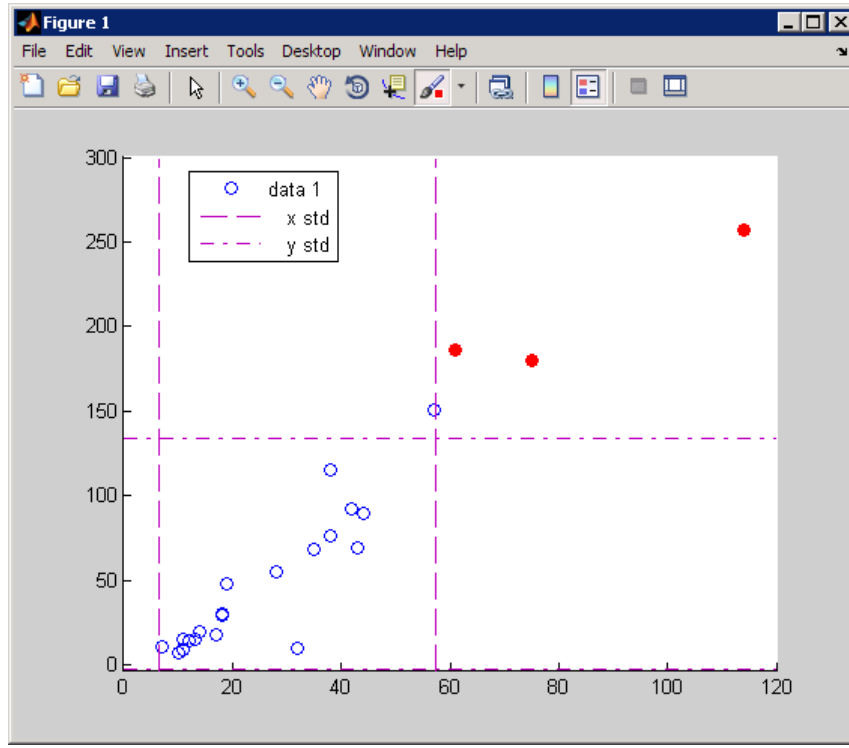


既定の設定では、データヒントには *x* 座標、*y* 座標および *z* 座標 (3 次元プロットの場合) が表示されます。あるデータ点から他のデータ点にデータをドラッグして新規の値を見るか、データを右クリックしてコンテキスト メニューを利用して、デー

タ ヒントを追加します。MATLAB コードを使用して、データヒントで表示するテキストをカスタマイズすることもできます。

"データのブラシ選択" は、クリックまたはドラッグによって、グラフ上の 1 つまたは複数の観測値を強調表示できるようにした関連機能です。データのブラシ選択モードに入るには、Figure ツール バーで、[データのブラシ選択] ツール  の左側をクリックします。ツール アイコンの右側の矢印をクリックすると、観測値のカラーを選択するためのドロップダウンのカラー パレットが開きます。次の Figure は前の Figure と同じ散布図ですが、標準偏差を超えるすべての観測値 ([ツール] 、[データの統計] GUI により特定) が赤でブラシ選択されています。

```
scatter(count(:,1),count(:,3))
```

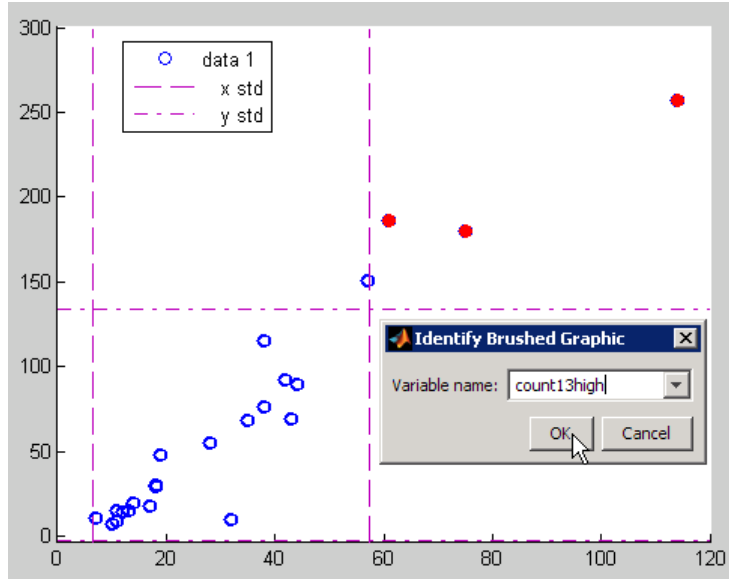


データの観測値をブラシ選択した後、次の操作を実行できます。

- データの観測の削除

- データの観測を定数値で置き換え
- データの観測を NaN 値で置き換え
- データの観測を、コマンド ウィンドウにドラッグ、コピー、貼り付け
- データの観測をワークスペース変数として保存

たとえば、データのブラシ選択のコンテキスト メニュー、または [ツール] 、[ブラシ選択中] 、[新規変数の作成] オプションを使用して、count13high と呼ばれる新しい変数を作成します。



ワークスペースの新しい変数は、次のようになります。

```
count13high

count13high =
    61   186
    75   180
   114   257
```

"リンク付きプロット" または data linking は、データのブラシ選択に関連する機能です。プロットは、プロットが描くワークスペース データに応答する接続をもつ場合

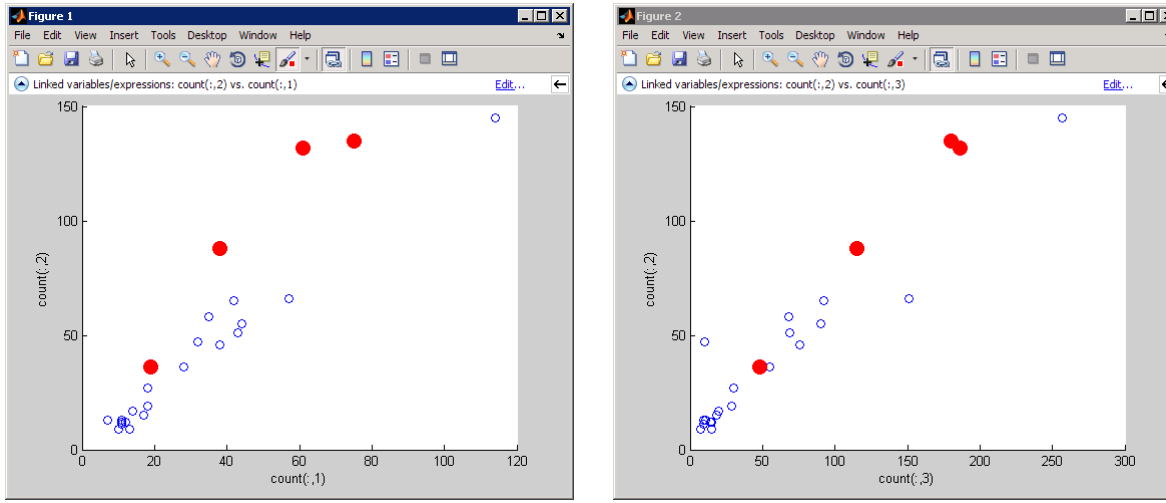
に、リンクしているといわれます。プロット オブジェクトの XData、YData (必要に応じて、ZData) に保存された変数のコピーは、これらがリンクしているワークスペース変数が変更または削除されるときに自動的に更新されます。そのため、これらの変数を表示しているグラフが自動的に更新されます。

変数にプロットをリンクすると、さまざまな表示で特定の観測値をトラックできます。リンクされたプロットのデータ点をブラシ選択する場合、1 つのグラフでブラシ選択すると、同じ変数にリンクしている各グラフ上の同じ観測値が強調表示されます。

データ リンクは、変数エディターがワークスペース変数とやりとりするのと同じように、Figure とワークスペース変数の 2 方向の直接のやりとりを確立します。Figure ツール バー上のデータのリンク プロット ツール  をアクティブにしてリンクを作成します。このツールをアクティブにすると、次の図に示すリンク プロットのメッセージ バー (おそらくタイトルは非表示) がプロットの上部に表示されます。プロットとリンク解除せずに (以下の図に示す) メッセージ バーを非表示にできます。この場合、メッセージ バーは表示されず、Figure と共に保存されません。

以下の 2 つのグラフは、左のグラフのいくつかの観測値をブラシ選択した後、リンクしたデータの散布図プロットを表します。共通の変数 count は、右の Figure にブラシ選択のマークを置きます。右のグラフは、データのブラシ選択モードではなくても、その変数にリンクしているのでブラシ選択のマークを表示します。

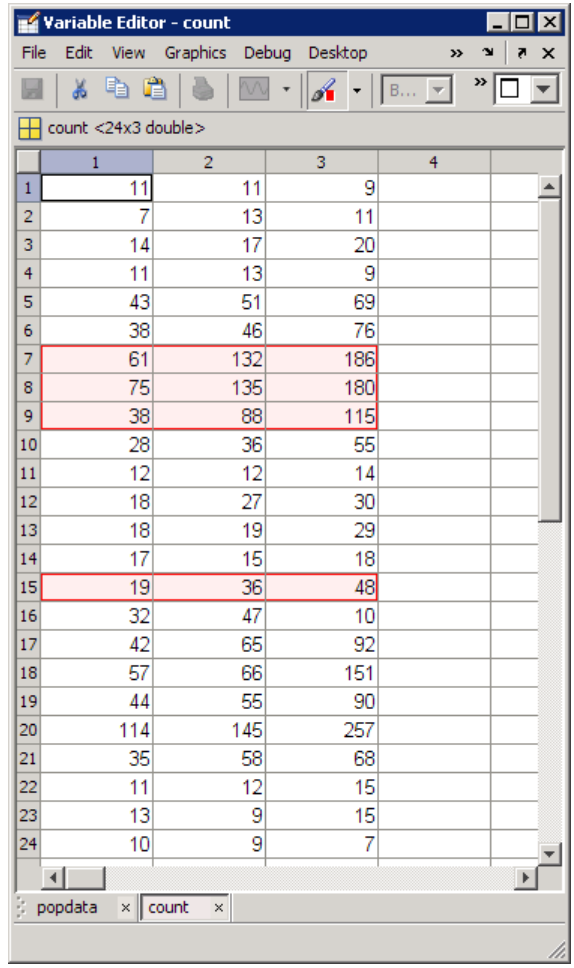
```
figure
scatter(count(:,1),count(:,2))
xlabel ('count(:,1)')
ylabel ('count(:,2)')
figure
scatter(count(:,3),count(:,2))
xlabel ('count(:,3)')
ylabel ('count(:,2)')
```



右のプロットは、ブラシ選択された観測値が左のプロットに比べてより直線的な並びであることを示しています。

ブラシ選択されたデータ観測値は、次に示すように、変数エディターにこれらの変数を表示すると、ブラシ カラーで強調表示されます。

```
openvar count
```



	1	2	3	4
1	11	11	9	
2	7	13	11	
3	14	17	20	
4	11	13	9	
5	43	51	69	
6	38	46	76	
7	61	132	186	
8	75	135	180	
9	38	88	115	
10	28	36	55	
11	12	12	14	
12	18	27	30	
13	18	19	29	
14	17	15	18	
15	19	36	48	
16	32	47	10	
17	42	65	92	
18	57	66	151	
19	44	55	90	
20	114	145	257	
21	35	58	68	
22	11	12	15	
23	13	9	15	
24	10	9	7	

変数エディターにおいて、リンクしたプロット データの任意の値を変更でき、グラフが編集結果を反映します。変数エディターからデータ観測値をブラシ選択するには、[ブラシ選択ツール] ボタン  をクリックします。ブラシ選択した変数がリンク プロットに現在描かれている場合には、ブラシ選択した観測値は変数エディター同様、プロットでも強調表示されます。行列の列である変数をブラシ選択すると、その行内のその他の列も ブラシ選択されます。つまり、行ベクトルまたは列ベクトル内の個々の観測値のブラシ選択はできますが、クリックした観測値だけでなく、行列内のすべての列がブラシ選択行で強調表示になります。

データのモデリング

概要

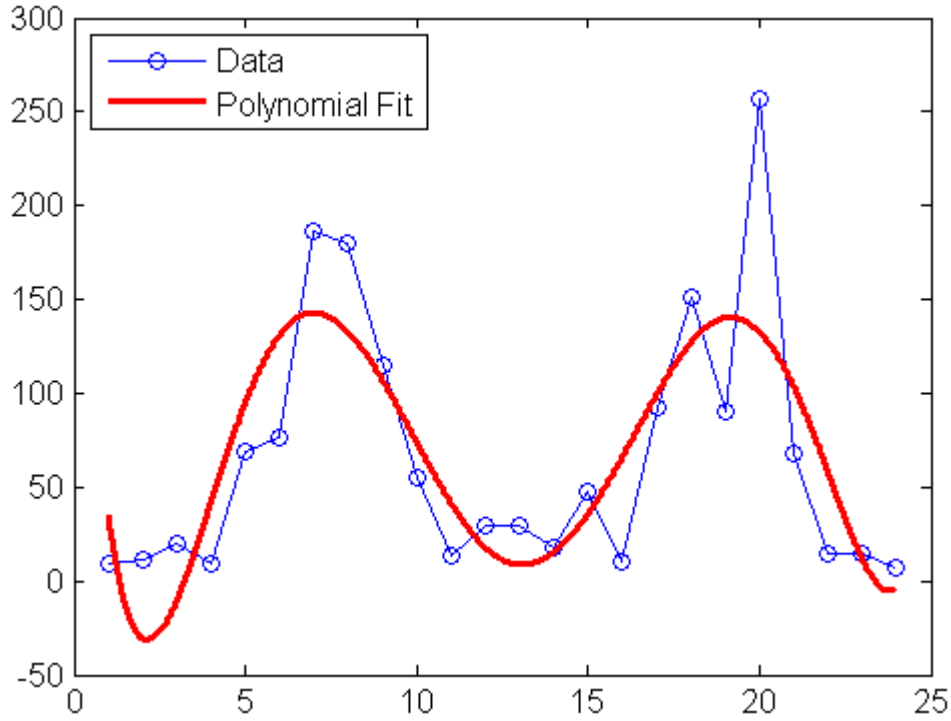
パラメトリック モデルは、データの関連性に関する理解を予測機能をもつ解析ツールに変えます。多項式モデルと正弦関数モデルは、上下する傾向をもつ交通量データに対しての単純な選択です。

多項式回帰

関数 polyfit を使用して多項式モデルの係数を推定し、次に関数 polyval を用いて、予測変数の任意の値でモデルを評価します。

以下のコードは、6 次多項式モデルを用いて 3 つ目の交差点での交通量データを近似します。

```
load count.dat
c3 = count(:,3); % Data at intersection 3
tdata = (1:24)';
p_coeffs = polyfit(tdata,c3,6);

figure
plot(c3,'o-')
hold on
tfit = (1:0.01:24)';
yfit = polyval(p_coeffs,tfit);
plot(tfit,yfit,'r-','LineWidth',2)
legend('Data','Polynomial Fit','Location','NW')
```



このモデルには、上下する傾向がありますが単純であるという利点があります。ただし、その予測機能の精度は、特にデータの両端で不正確になります。

一般線形回帰

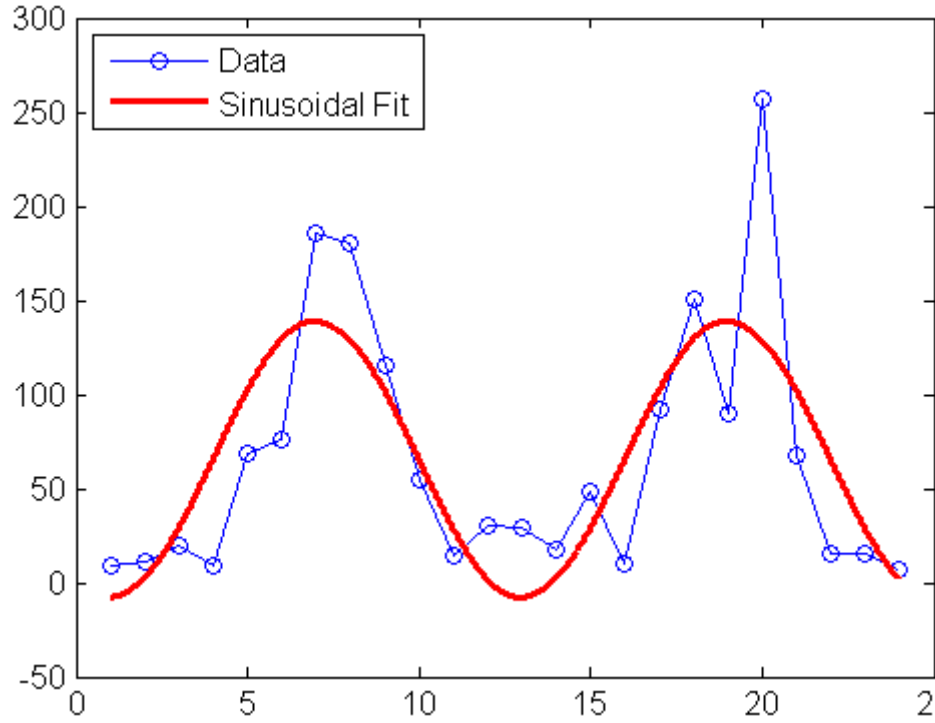
データには、12 時間の周期があり、7 時付近にピークがあることを仮定すると、次の形の正弦波モデルで近似するのが適切です。

$$y = a + b\cos((2\pi/12)(t-7))$$

係数 *a* と *b* は、線形にみえます。MATLAB `mldivide`（バックスラッシュ）演算子を用いて、一般線形モデルに近似します。

```
load count.dat
c3 = count(:,3); % Data at intersection 3
tdata = (1:24)';
X = [ones(size(tdata)) cos((2*pi/12)*(tdata-7))];
```

```
s_coeffs = X¥c3;

figure
plot(c3,'o-')
hold on
tfit = (1:0.01:24)';
yfit = [ones(size(tfit)) cos((2*pi/12)*(tfit-7))]*s_coeffs;
plot(tfit,yfit,'r-','LineWidth',2)
legend('Data','Sinusoidal Fit','Location','NW')
```



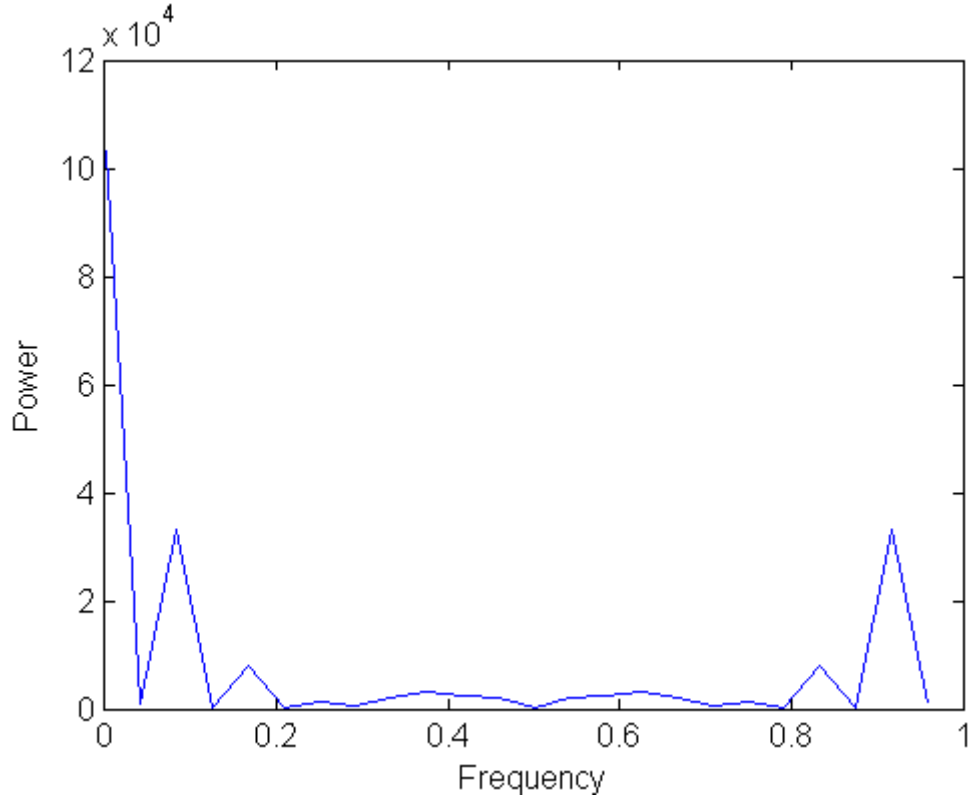
関数 lscov を使用して、推定された 係数の標準誤差と平均二乗誤差など、近似に関する統計量を計算します。

```
[s_coeffs,stdx,mse] = lscov(X,c3)
s_coeffs =
   65.5833
   73.2819
```

```
stdx =
    8.9185
   12.6127
mse =
  1.9090e+003
```

関数 fft を使用して計算された "ピリオドグラム" を使用して、データの 12 時間周期の仮定をチェックします。

```
Fs = 1; % Sample frequency (per hour)
n = length(c3); % Window length
Y = fft(c3); % DFT of data
f = (0:n-1)*(Fs/n); % Frequency range
P = Y.*conj(Y)/n; % Power of the DFT

figure
plot(f,P)
xlabel('Frequency')
ylabel('Power')

predicted_f = 1/12
predicted_f =
    0.0833
```



0.0833 付近のピークは、わずかに高い周波数で起こりますが、仮定を支持するものです。モデルは、適宜調整できます。

4

グラフィックス

基本プロット関数

この節の内容…

プロットの作成

関数 plot は、入力引数により形式が異なります。

- y がベクトルならば、plot(y) は、y の要素のインデックスに対して、y の要素群の区分的な線形グラフを作成します。
- 引数として 2 つのベクトルが指定されている plot(x,y) は、x に対する y のグラフを作成します。

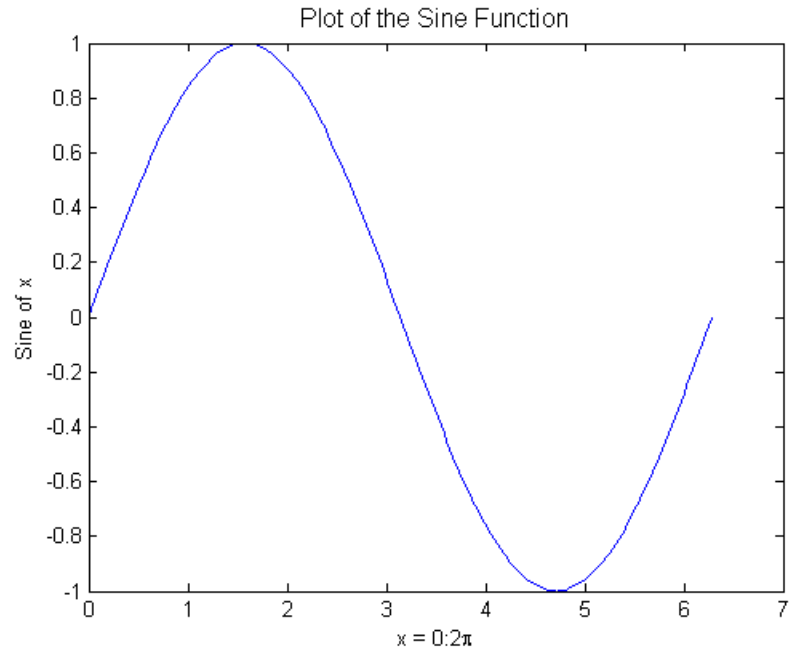
たとえば、次のステートメントは、colon 演算子を使ってゼロから 2π までの x の値のベクトルを作成し、これらの値の正弦を計算して結果をプロットします。

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```

座標軸ラベルとタイトルを追加します。

```
xlabel('x = 0:2¥pi')
ylabel('Sine of x')
title('Plot of the Sine Function','FontSize',12)
```

文字 ¥pi で記号 π を作成し、FontSize プロパティでタイトルに使用するテキストのサイズを大きくします。



1 つのグラフ内に複数のデータ セットをプロット

複数の x-y ペアの引数から、plot の 1 回の呼び出しにより複数のグラフが作成されます。MATLAB は、それぞれのラインを異なる色で示します。

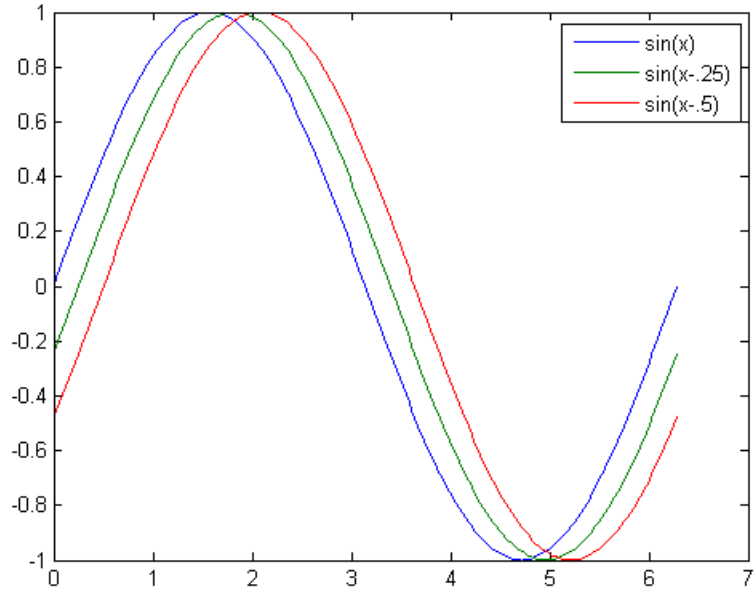
たとえば、次のステートメントは 3 つの関連する x の関数をプロットします。

```
x = 0:pi/100:2*pi;
y = sin(x);
y2 = sin(x-.25);
y3 = sin(x-.5);
```

```
plot(x,y,x,y2,x,y3)
```

関数 legend により、個々のラインを簡単に識別する方法が提供されます。

```
legend('sin(x)','sin(x-.25)','sin(x-.5)')
```



ライン スタイルと色の指定

plot コマンドを使用してデータをプロットする場合、色、ライン スタイル、(正符号や円のような) シンボル マーカーを、次の構文により指定できます。

```
plot(x,y,'color_style_marker')
```

color_style_marker は、色、ライン スタイル、マーカー タイプを表す (一重引用符で囲まれた) 1 ～ 4 個の文字を含む文字列です。たとえば、次の例を考えてみましょう。

```
plot(x,y,'r:+')
```

データは赤い点線でプロットされ、各データ点には + のマーカーが配置されます。

この文字列は、以下の要素の組み合わせで構成されます。

種類	値	平均
色	'c' 'm' 'y' 'r' 'g' 'b' 'w' 'k'	シアン マゼンタ 黄 赤 緑 青 白 黒
ライン スタイル	'-' '--' ':' '.-' 文字なし	直線 破線 点線 一点鎖線 線なし
マーカー タイプ	'+' 'o' '*' 'x' 's' 'd' '^' 'v' '>' '<' 'p' 'h' 文字なし	プラス記号 空の丸 アスタリスク x 文字 塗りつぶした四角 塗りつぶした菱形 塗りつぶした上向き三角 塗りつぶした下向き三角 塗りつぶした右向き三角 塗りつぶした左向き三角 塗りつぶした五角形 塗りつぶした六角形 マーカーなし

線とマーカーのプロット

マーカー タイプを指定してライン スタイルは設定しない場合、MATLAB ではマーカーのみが描かれ、ラインは描かれません。たとえば、

```
plot(x,y,'ks')
```

は、各データ点に黒の四角形をプロットしますが、マーカーを線で接続しません。

ステートメント

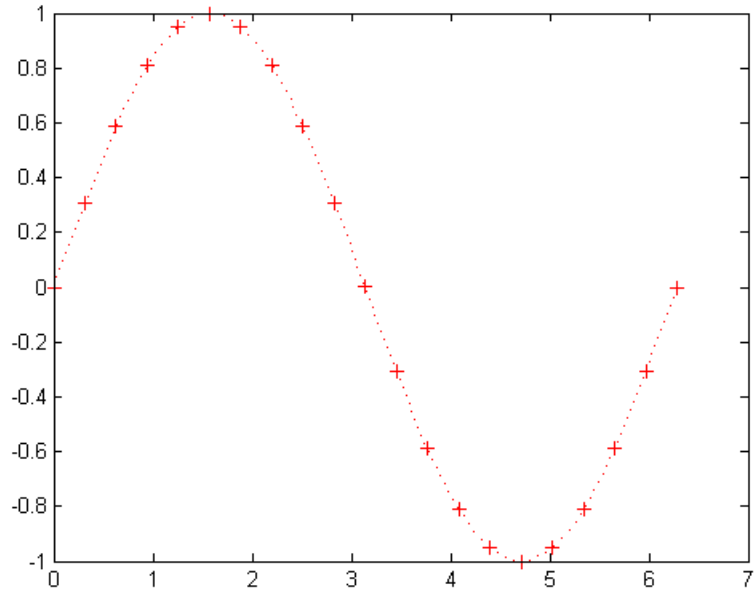
```
plot(x,y,'r:+')
```

は、赤色の点線を描画し、各データ点にプラス記号のマーカーを置きます。

10 データ点ごとにマーカーを置く

マーカーは、線をプロットするときよりも少ないデータ点でプロットできます。次の例は、点線とマーカー プロットに対して異なる点数を使ってデータを 2 回プロットします。

```
x1 = 0:pi/100:2*pi;
x2 = 0:pi/10:2*pi;
plot(x1,sin(x1),'r:',x2,sin(x2),'r+')
```



虚数データと複素数データのグラフ

MATLAB では、複素数値を引数として plot に渡す場合、単一の複素数引数を渡すとき "以外" は虚数部が無視されます。この特別な場合として、実数部に対して虚数部をプロットするための省略形として、コマンドを使うことができます。従って、

```
plot(Z)
```

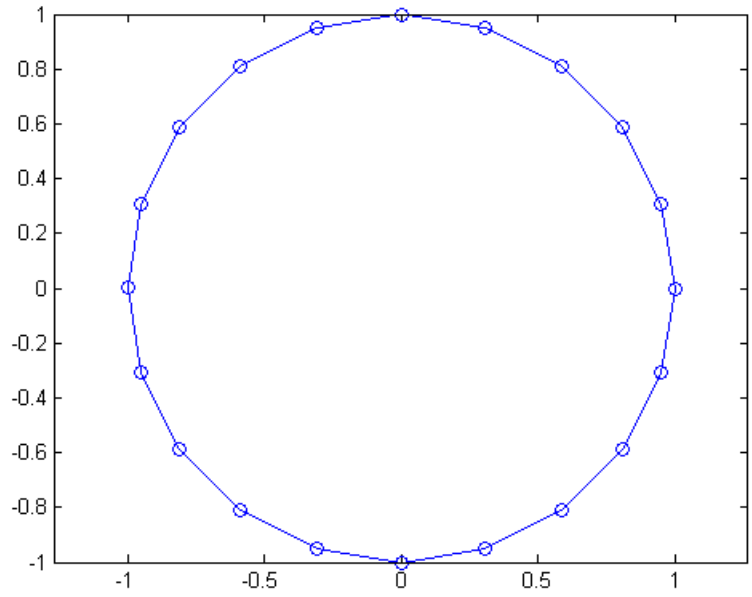
は、Z が複素数ベクトルまたは行列ならば、次のステートメントと等価です。

```
plot(real(Z),imag(Z))
```

たとえば、

```
t = 0:pi/10:2*pi;
plot(exp(i*t),'-o')
axis equal
```

は、頂点を小さな円で表わした 20 角形です。axis equal コマンドは、x 軸と y 軸の目盛り間隔を同じ長さにして、プロットの外観をより円形に近いものにします。



既存のグラフにプロットを追加

hold コマンドを使って、既存のグラフにプロットを追加することができます。次のように入力した場合、

```
hold on
```

他のプロット コマンドを発行しても、MATLAB は既存のグラフを置き換えません。代わりに、MATLAB は新しいグラフを現在のグラフに組み合わせます。

たとえば、次のステートメントは、関数 peaks の等高線図を作成した後に、同じ関数の pcolor (擬似カラー) プロットを重ね書きします。

```
% Obtain data from evaluating peaks function
[x,y,z] = peaks;
% Create pseudocolor plot
pcolor(x,y,z)
% Remove edge lines a smooth colors
shading interp
% Hold the current graph
hold on
% Add the contour graph to the pcolor graph
contour(x,y,z,20,'k')
% Return to default
hold off
```

hold on コマンドは、1 つの Figure 上に pcolor プロットと contour プロットを重ねて表示します。

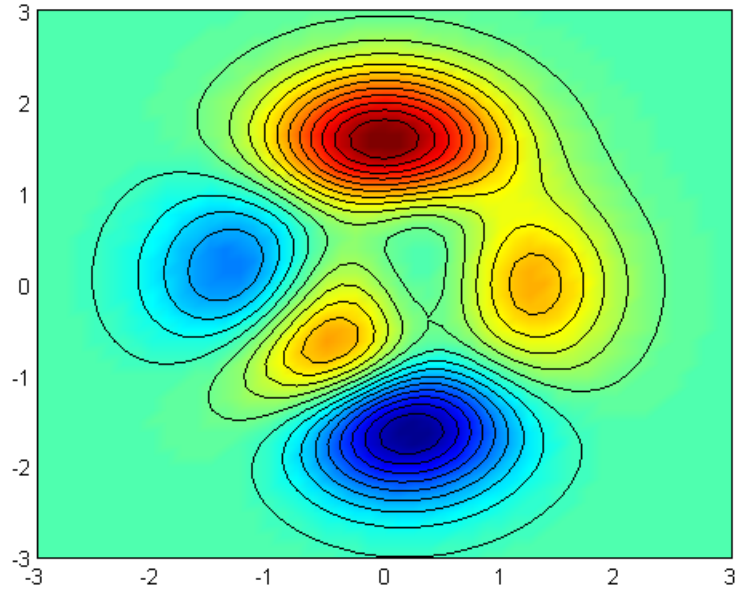


Figure ウィンドウ

Figure ウィンドウが未作成の場合、グラフ化関数は自動的に新しい Figure ウィンドウを開きます。複数の Figure ウィンドウが開いている場合、MATLAB は "現在の Figure" として示されている Figure ウィンドウ (最後に利用またはクリックされた Figure) が使用されます。

既存の Figure ウィンドウを現在の Figure にするには、ポインターをそのウィンドウ内に置いた状態でマウスをクリックするか、次のように入力します。

```
figure(n)
```

ここで、n は Figure ウィンドウのタイトル バーにおける数字です。

新規の Figure ウィンドウを開き、現在の Figure にします。

```
figure
```

新規プロット用に Figure をクリア

Figure が既に存在するとき、ほとんどのプロット関数は、座標を消去して、この Figure を使って新規プロットを作成します。しかし、これらのコマンドは、背景色やカラー マップのような Figure のプロパティをリセットしません。前のプロットで Figure のプロパティを設定した場合は、Figure のプロパティを既定の設定に戻すために、新規プロットを作成する前に clf コマンドと reset オプションを使う場合があります。

```
clf reset
```

次に、Figure のプロパティをその既定の設定に復元する新しいプロットを作成します。

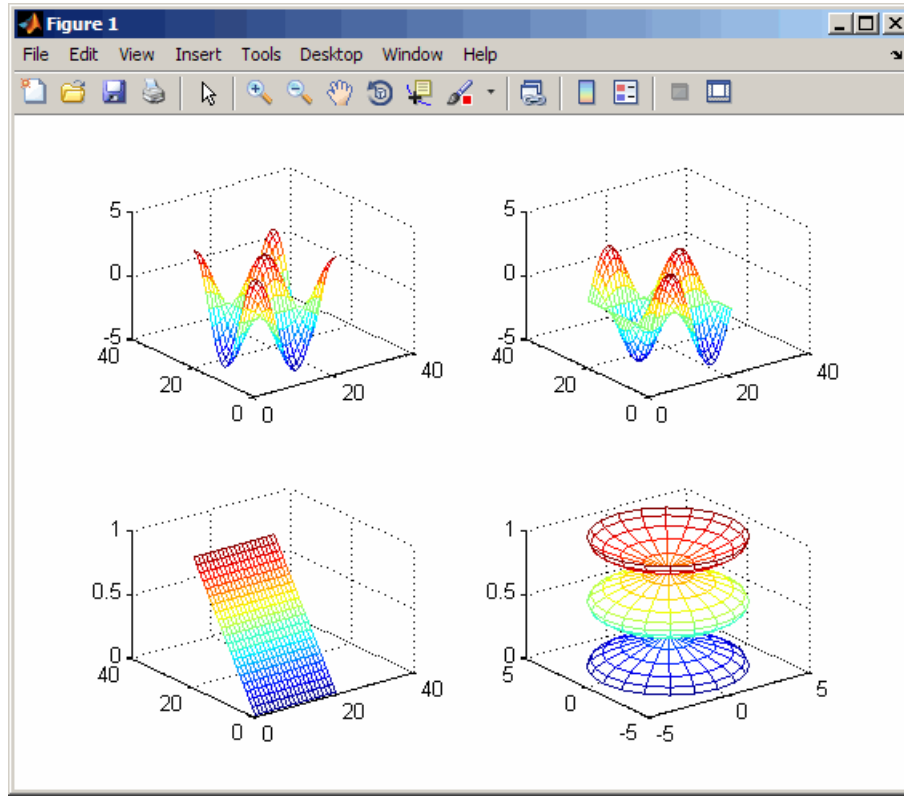
1 つの Figure ウィンドウに複数のプロットを表示

subplot コマンドを使って、同じウィンドウに複数のプロットを表示したり、同じ用紙に印刷したりできます。以下を入力すると

```
subplot(m,n,p)
```

Figure ウィンドウが m 行 n 列の小さなサブプロットに分割され、現在のプロットとして p 番目のサブプロットが選択されます。プロットは、まず Figure ウィンドウの一番上の行、その次に 2 番目の行、と順に番号が付けられます。たとえば、次のステートメントは、Figure ウィンドウの 4 つのサブ領域にデータをプロットします。

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X)
subplot(2,2,2); mesh(Y)
subplot(2,2,3); mesh(Z)
subplot(2,2,4); mesh(X,Y,Z)
```



軸の制御

axis コマンドは、スケーリング、方向、縦横比を設定するための多数のオプションをサポートしています。

座標軸の範囲と目盛りの自動設定

既定の設定では、MATLAB はデータの最大値と最小値を検出し、その範囲とする座標軸の限界値を選択します。MATLAB は、データを明確に表示するグラフを作成するよう、範囲と座標軸の目盛りの値を選択します。しかし、関数 axis または xlim、ylim および zlim の関数の使用により範囲を独自に設定できます。

メモ: ある座標軸の範囲を変更すると、データをより見やすく表示するため、他の範囲を変更することになる場合があります。範囲の自動設定を無効にするには、axis manual コマンドを入力します。

軸の範囲の設定

axis コマンドを使ってユーザー独自に範囲を指定することができます。

```
axis([xmin xmax ymin ymax])
```

または、3 次元グラフに対しては、次のようにします。

```
axis([xmin xmax ymin ymax zmin zmax])
```

コマンド

```
axis auto
```

を使うと、自動範囲選択が可能になります。

軸の縦横比の設定

axis コマンドでも、あらかじめ定義された多数のモードを指定できます。たとえば、

```
axis square
```

は、x 軸と y 軸を同じ長さにします。

```
axis equal
```

は、x 軸と y 軸の個々の目盛りの増分を等しくします。次のステートメントは、

```
plot(exp(i*[0:pi/10:2*pi]))
```

に、axis square または axis equal を続けると、楕円を円にします。

```
axis auto normal
```

は、軸のスケーリングを既定の自動モードに戻します。

軸の視覚状態の設定

axis コマンドを使用して、軸を表示または非表示にできます。

```
axis on
```

は、軸を表示します。これは既定です。

```
axis off
```

は、軸を非表示にします。

グリッド ラインの設定

grid コマンドは、グリッド ラインのオンとオフを切り替えます。ステートメント

```
grid on
```

は、グリッド ラインを表示し、

```
grid off
```

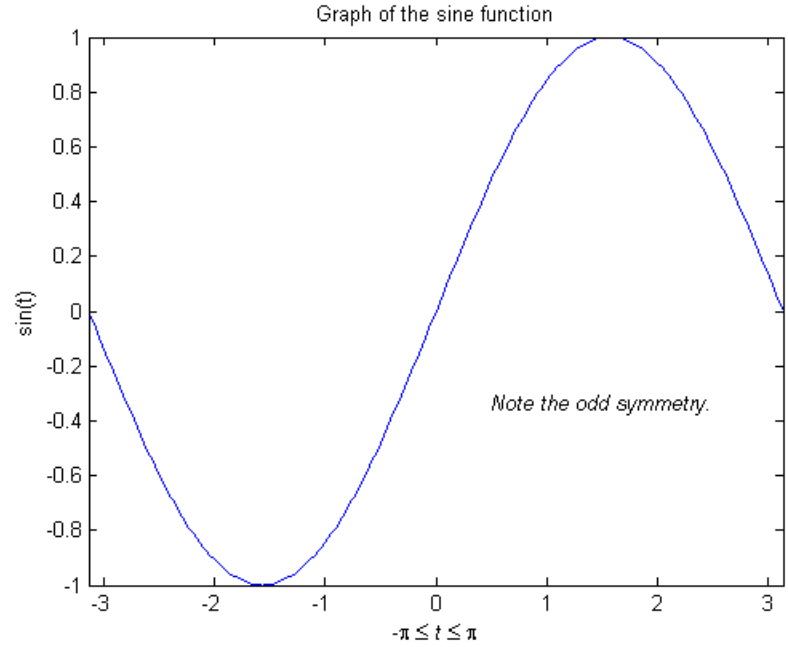
は、グリッド ラインを非表示にします。

軸のラベルとタイトルの追加

xlabel、ylabel、zlabel コマンドは、x 軸、y 軸、z 軸にラベルを付けます。title コマンドは、Figure の上部にタイトルを付け、関数 text は Figure の任意の位置にテキストを挿入します。

以下の例に示すように、テキスト文字列に LaTex 記法を用いて数学シンボルを生成することが可能です。

```
t = -pi:pi/100:pi;
y = sin(t);
plot(t,y)
axis([-pi pi -1 1])
xlabel('-¥pi ¥leq {¥itt} ¥leq ¥pi')
ylabel('sin(t)')
title('Graph of the sine function')
text(0.5,-1/3,'{¥itNote the odd symmetry.}')
```



テキスト文字列の位置は、座標軸の単位 (すなわち、データと同じ単位) で定義されます。関数 annotation を使用すると、Figure の標準単位でテキストを配置できます。

Figure の保存

Figure を保存するには、[ファイル] メニューから [保存] を選択します。これによって、Figure がプロパティ データ、メニュー、uicontrol およびすべての注釈と共に (つまり、ウィンドウ全体が) ファイルに書き込まれます。以前に Figure を保存していない場合は、[名前を付けて保存] ダイアログが表示されます。このダイアログ ボックスには、Figure を FIG ファイルとして保存したり、グラフ形式でエクスポートしたりするためのオプションが用意されています。

Figure を以前に保存している場合は、[保存] を利用して Figure を保存すると、[名前を付けて保存] ダイアログを表示せずに保存できます。

他のアプリケーションで利用するために TIFF または JPG などの標準グラフィックス形式を用いて Figure を保存するには、[ファイル] メニューから [名前を付けて保存] (または追加設定が必要なときには [エクスポートの設定]) を選択します。

メモ: Figure を保存するための形式を指定すると、次回にその Figure または新規の Figure を保存するときに再びそのファイル形式が使われます。以前に使用した形式では保存しない場合は、[名前を付けて保存] を使用して、書き込むファイルの種類を [ファイルの種類] ドロップダウン メニューで設定します。

または、saveas コマンドを使いコマンド ラインからでも保存できます。このコマンドには、Figure を他の形式で保存するためのオプションがあります。採用されているレンダリング法によってビットマップ ファイルまたはメタファイルに Figure を保存する、より制限された hgexport コマンドも利用可能です。

ワークスペース データの保存

ワークスペース変数は、Figure ウィンドウの [ファイル] メニューの [別名でワークスペースを保存] を選択して保存できます。Figure ウィンドウの [ファイル] メニューの [データのインポート] アイテムを使って保存されたデータを再び読み込むことができます。MATLAB は、さまざまなデータ ファイル形式をサポートします。拡張子 .mat をもつ MATLAB データ ファイルも含まれます。

Figure の再作成のための MATLAB コードの生成

Figure とその Figure が含むグラフを再作成する MATLAB コードを、Figure ウィンドウの [ファイル] メニューの [コード生成] を選択して生成できます。このオプションは、プロット ツールを使ってグラフを作成し、同じまたは異なるデータを使って同じグラフを作成する場合に特に役立ちます。

メッシュ プロットと表面プロットの作成

この節の内容…

メッシュ プロットと表面プロットについて

MATLAB では、x-y 平面に引かれたグリッド上にある点の z 座標によって表面を定義し、隣接する点を結ぶ直線を使用してこれを表します。関数 mesh と関数 surf によって表面を 3 次元で表示します。

- mesh では、定義点を結ぶラインのみに色の付たワイヤーフレームの表面が描かれます。
- surf では、点を結ぶラインと表面の構成面が色彩表示されます。

MATLAB では、Figure のカラー マップのインデックスに z データ値をマッピングすることにより表面を彩色します。

変数関数の可視化

2 変数関数 z = f(x,y) を表示するには、

1 関数の領域をカバーするように、それぞれ行と列を繰り返し使って X と Y 行列を作成します。

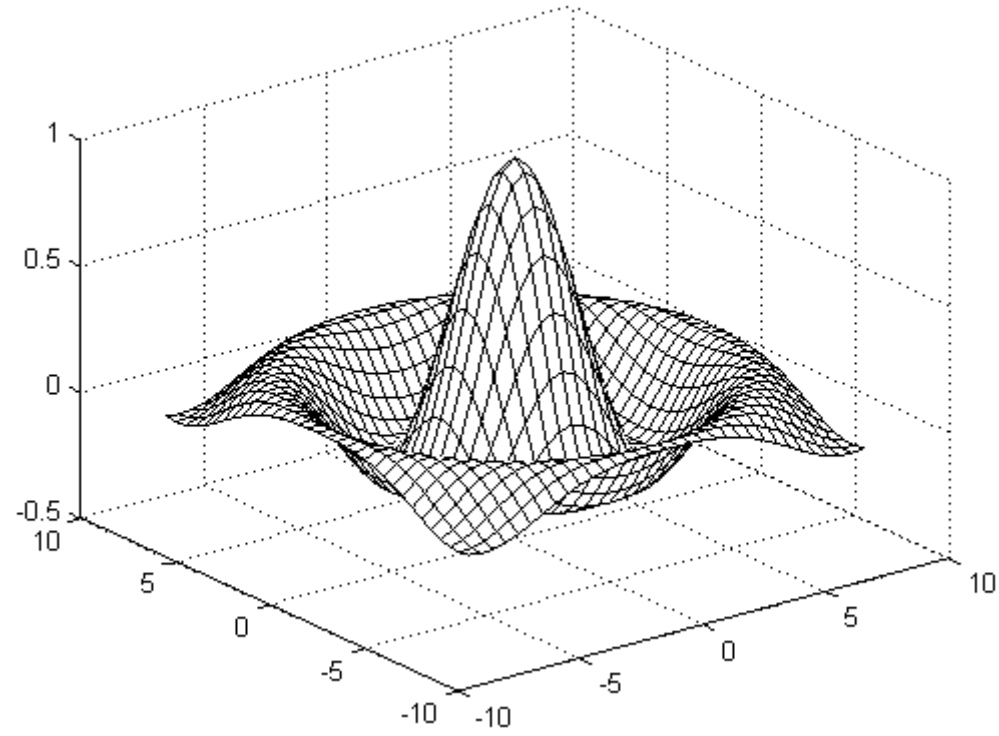
2 X と Y を使って関数を評価し、グラフ化します。

関数 meshgrid は、2 変数関数の実行に用いるために、単一のベクトルまたは 2 つのベクトル x と y で指定される領域を行列 X と Y に変換します。X の行はベクトル x のコピーで、Y の列はベクトル y のコピーです。

例 ― 関数 sinc のグラフ化

この例は、2 次元の関数 sinc、sin(r)/r を x および y 方向で実行しグラフ化します。R は、行列の中心にある原点からの距離です。eps (小さな浮動小数点数を出力する MATLAB コマンド) を加えると、原点で 0/0 が中間で生じることを避けることができます。

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(X,Y,Z,'EdgeColor','black')
```



既定の設定では、MATLAB は現在のカラー マップを使用してメッシュを彩色します。ただし、この例では、EdgeColor 表面のプロパティを指定することにより、単色のメッシュを使用します。

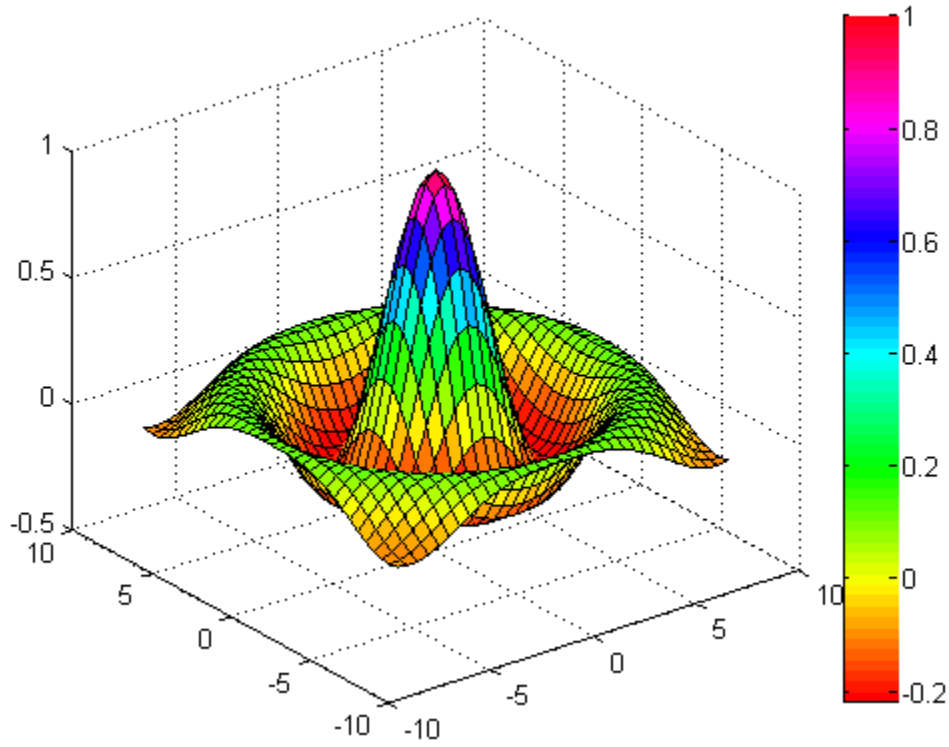
陰線の除去を無効にすることにより、透明なメッシュを作成できます。

```
hidden off
```

例 – 色の付いた表面プロット

表面プロットは、四角形の面が色付けされることを除いて、メッシュ プロットに似ています。面の色は、Z の値とカラー マップによって決定されます (colormap は、順番付けられた色のリストです)。次のステートメントは、関数 sinc を表面プロットとしてグラフ化し、カラー マップを選択し、カラー バーを付加して、データの色へのマッピングを示します。

```
surf(X,Y,Z)
colormap hsv
colorbar
```

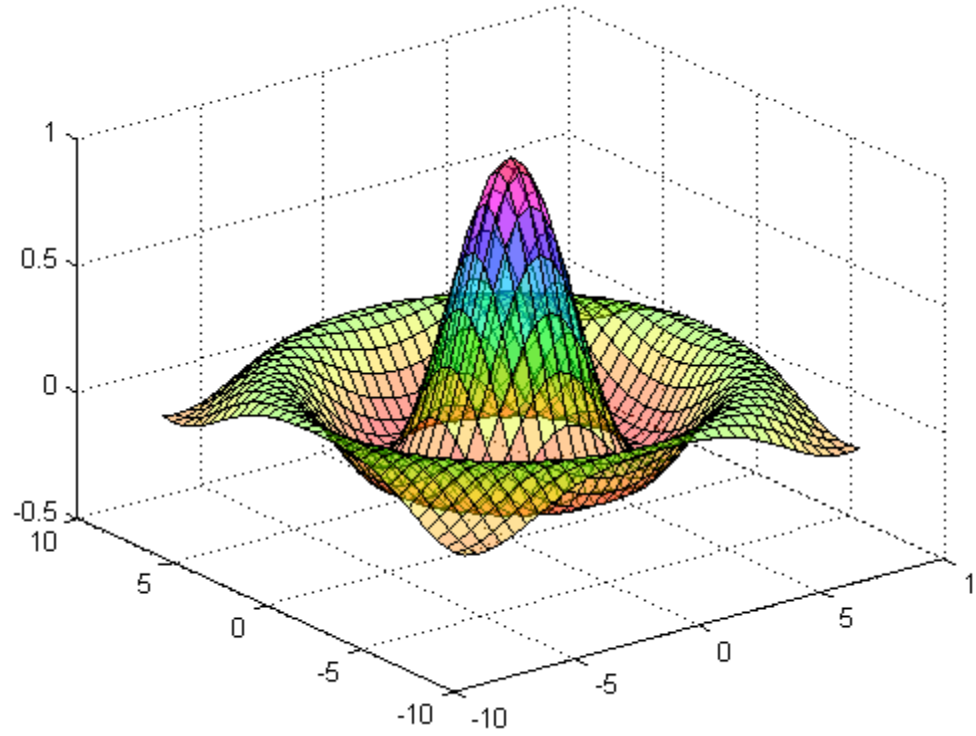


表面の透明化

曲面の面は、可変の程度で透明にすることができます。透明性 (alpha 値として参照されます) は、オブジェクト全体に対して指定されるか、またはカラー マップと同様に機能する alphamap に基づきます。たとえば、

```
surf(X,Y,Z)
colormap hsv
alpha(.4)
```

は、面の alpha 値が 0.4 である表面を生成します。alpha 値の範囲は 0 (完全に透明) から 1 (不透明) です。

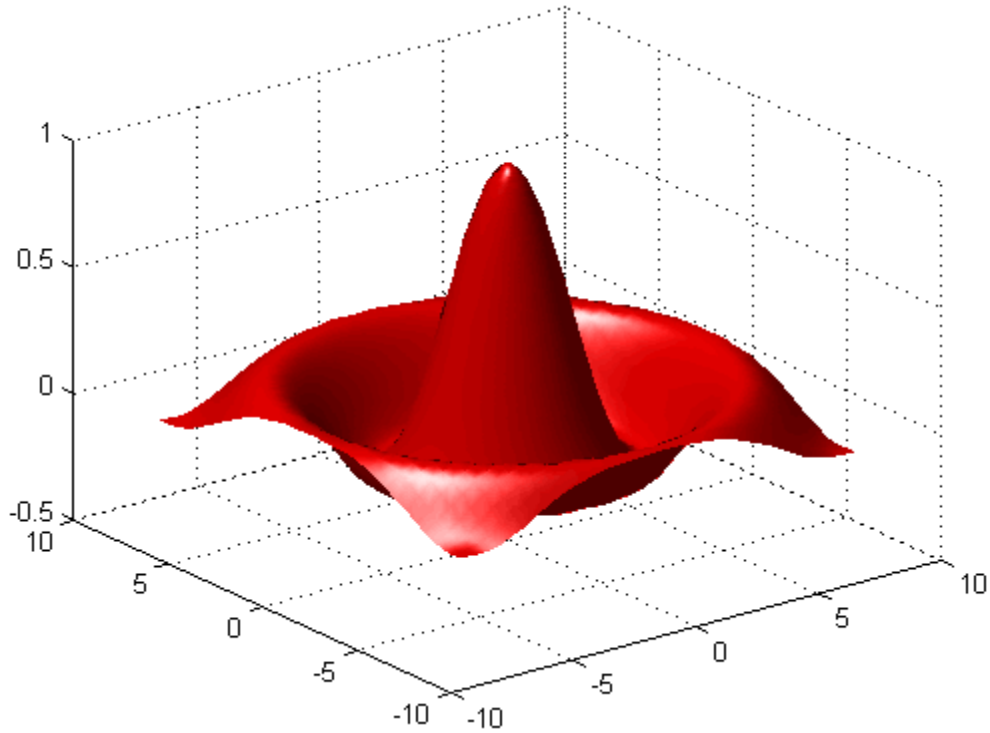


表面プロットをライトで照らす

ライティングは、方向性のある光源によってオブジェクトを照らす手法です。この手法は、表面の形状に見やすい微妙な違いを与えます。ライティングは、3 次元グラフに現実味を加えるためにも用いることができます。

次の例は、前の例と同じ表面を使い、赤色にカラーリングし、メッシュ ラインを除去します。light オブジェクトは、「カメラ」の左側に付加されます (カメラは、表面を見ている空間内の位置です)。

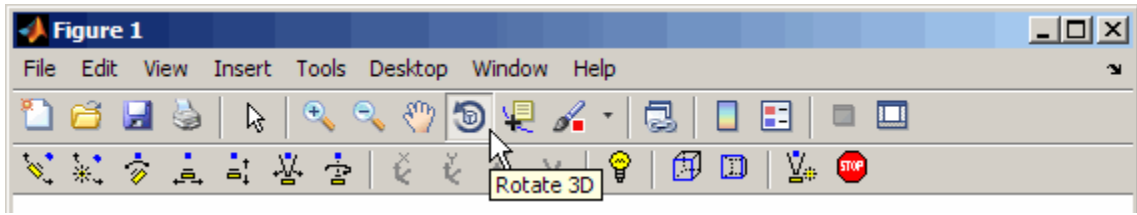
```
surf(X,Y,Z,'FaceColor','red','EdgeColor','none')
camlight left; lighting phong
```



表面の操作

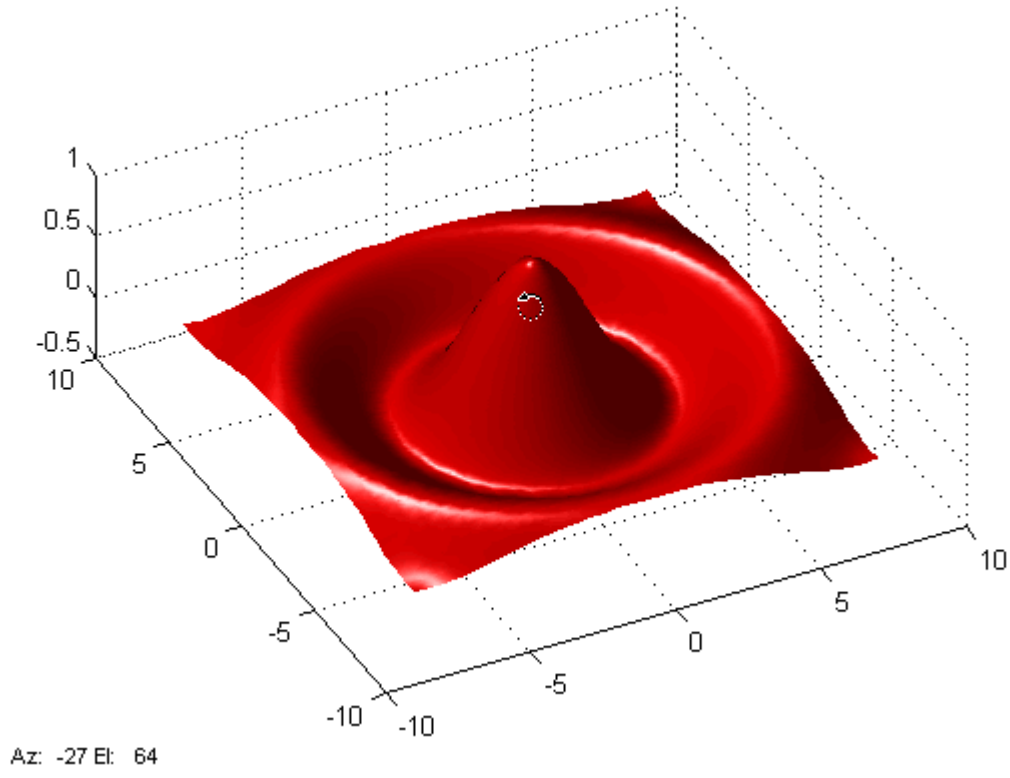
Figure ツール バーとカメラ ツール バーは、3 次元グラフィックスを対話的に調査する方法を提供します。カメラ ツール バーは、Figure ツール バーの [表示] メニューから [カメラのツール バー] を選択することにより表示します。

次の図は、両方のツール バーと [3 次元回転] ツールが選択されていることを示します。



これらのツールを使って、カメラを表面オブジェクトの周囲で回転、拡大、ライティングの追加、その他のビュー操作をコマンドを実行せずに行うことができます。

次の図は、[3 次元回転] を利用して下方向にカメラを回転して表面を表示します。ツールのカーソル アイコンが表面に表示されているのが分かります。ドラッグすると、視点の方位角と仰角が読み取られて座標軸の左下隅に表示されます。


1
0.5
0
-0.5
10
5
0
-5
-10
-10
-5
0
5
10
Az: -27 El: 64

イメージ データのプロット

この節の内容…

イメージ データのプロットについて

2 次元配列は、イメージ として表示されます。ここで、配列要素はイメージの明るさまたは色を決めます。たとえば、ステートメント

```
load durer
whos
Name          Size         Bytes  Class

  X           648x509      2638656  double array
  caption     2x28             112  char array
  map         128x3           3072  double array
```

は、ファイル durer.mat を読み込み、ワークスペースに 3 個の変数を追加します。行列 X は 648 行 509 列で、map はこのイメージに対するカラー マップである 128 行 3 列の行列です。

durer.mat のような MAT ファイルは、あるプラットフォームで作成され、後で他のプラットフォーム上の MATLAB が読み込むことができるバイナリ ファイルです。

X の要素は、1 から 128 の間の整数で、カラー マップ map のインデックスです。したがって、

```
image(X)
colormap(map)
axis image
```

は、アルブレヒト・デューラーのエッチングを再現します。右上隅の魔方陣の高解像度スキャンは、他のファイルで利用可能です。次のように入力します。

```
load detail
```

と入力し、キーボードの上矢印キーを使用して、image、colormap、axis コマンドを再実行します。

イメージの読み込みと書き込み

標準のイメージ ファイル (TIFF、JPEG、BMP など) は、関数 imread を使用して読み取ることができます。imread により出力されるデータのタイプは、読み込んでいるイメージのタイプにより異なります。

MATLAB データは、関数 imwrite を使ってさまざまな標準イメージ形式に書き込むことが可能です。

グラフィックスの印刷

この節の内容...

印刷の概要

MATLAB Figure を、コンピューターに接続されているプリンターに直接印刷したり、MATLAB がサポートする標準グラフィックス ファイル形式の 1 つにエクスポートすることができます。Figure の印刷およびエクスポートには、2 つの方法があります。

- [ファイル] メニューの [印刷]、[印刷プレビュー] または [エクスポートの設定] GUI オプションを利用します。
- コマンド ラインから Figure を印刷またはエクスポートするには、print コマンドを使用します。

print コマンドは、ドライバーやファイル形式をより詳細にコントロールします。[印刷プレビュー] ダイアログ ボックスを使って、Figure のサイズ、縦横比、配置、ページ見出しをコントロールできます。

ファイル メニューからの印刷

[ファイル] メニューには、印刷に関連するメニュー オプションが 2 つあります。

- [印刷プレビュー] オプションはダイアログ ボックスを表示します。このダイアログ ボックスでは、出力するページをプレビューしながら、印刷する Figure のレイアウトとスタイルを決めることができます。さらに、このダイアログ ボックスからその Figure を印刷できます。このダイアログ ボックスには、以前は [ページ設定] ダイアログ ボックスの一部であったオプションが含まれます。
- [印刷] オプションは、プリンターの選択、標準の印刷オプションの選択、Figure の印刷のためのダイアログ ボックスを表示します。

印刷する出力が希望通りかどうかをみるためには、[印刷プレビュー] を使います。[ページ設定] ダイアログ ボックスの [ヘルプ] ボタンをクリックすると、ページの設定方法に関する情報が表示されます。

Figure をグラフィックス ファイルにエクスポート

[ファイル] メニューの [エクスポートの設定] オプションは、グラフィックス ファイルとして保存する Figure のテキストのサイズ、フォント、スタイルなどのグラフィックスの特性を設定できる GUI を開きます。[エクスポートの設定] ダイアログ ボックスでは、出力をカスタマイズして標準化するためのテンプレートを定義し、適用することができます。設定の後で、Figure を EPS、PNG、TIFF などさまざまな標準グラフィックス ファイル形式にエクスポートできます。

印刷コマンドの利用

print コマンドは、プリンターに送られる出力タイプにおいてより柔軟性があり、関数ファイルやスクリプト ファイルから印刷を制御できます。結果は、直接既定のプリンターに送られるか、または指定した出力ファイルに保存されます。TIFF、JPEG、PNG などのさまざまな出力形式が利用可能です。

たとえば、次のステートメントは、現在の Figure ウィンドウの内容を PNG グラフィックスとして magicsquare.png というファイルに保存します。

```
print -dpng magicsquare.png
```

Figure を画面上の Figure と同じサイズで保存するには、次のステートメントを使用します。

```
set(gcf,'PaperPositionMode','auto')
print -dpng -r0 magicsquare.png
```

同じ Figure を 200 dpi の解像度で TIFF ファイルとして保存するためには、次のコマンドを使います。

```
print -dtiff -r200 magicsquare.tiff
```

コマンド ラインで print と入力した場合、

```
print
```

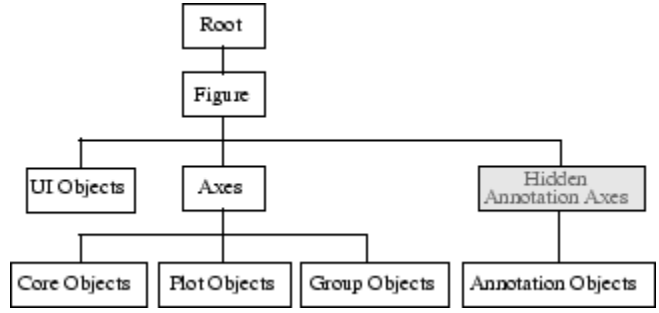
既定のプリンターに現在の Figure を印刷します。

Handle Graphics オブジェクトの取り扱い

この節の内容…
グラフィックス オブジェクト (p. 4-28)
オブジェクト プロパティの設定 (p. 4-30)
オブジェクトの利用に関する関数 (p. 4-33)
座標軸または Figure の指定 (p. 4-34)
既存のオブジェクトのハンドル番号の検索 (p. 4-36)

グラフィックス オブジェクト

グラフィックス オブジェクトは、グラフィックスやユーザー インターフェイスの要素を表示するために用いる基本要素です。次の表に示すように、これらのオブジェクトは階層構造にまとめられます。



プロット関数を呼び出すと、MATLAB は Figure ウィンドウ、座標軸、ライン、テキストなどのさまざまなグラフィックス オブジェクトを使用してグラフを作成します。それぞれのオブジェクトには既定の一連のプロパティがあり、それを使用してグラフの動作と外観を管理できます。

たとえば、次のステートメントは、白色の背景をもつ Figure を作成します。Figure ツール バーは表示されません。

```
figure('Color','white','Toolbar','none')
```

よく使用されるグラフィックス オブジェクト

MATLAB では、関数を呼び出してグラフを作成するときにグラフィックス オブジェクトの階層が作成されます。たとえば、関数 plot を呼び出すと、以下のグラフィックス オブジェクトが作成されます。

- Figure ― 座標軸、ツール バー、メニューなどを含むウィンドウ
- Axes ― データを表すラインを含む座標系
- lineseries plot オブジェクト ― 関数 plot に渡されるデータを表すライン
- Text ― 軸目盛りのラベル、およびオプションのタイトルと注釈

グラフのタイプによりデータを表すオブジェクトは異なります。すべてのデータ オブジェクトは座標軸に含まれ、すべてのオブジェクト (ルート以外) は Figure に含まれます。

ルートは、コンピューターまたは MATLAB の状態に関する情報を主に保存する抽象的なオブジェクトです。ユーザーは、ルート オブジェクトは作成できません。ルート オブジェクトのハンドルは、常に 0 です。

オブジェクト ハンドル

MATLAB は、グラフィックス オブジェクトの作成時、オブジェクトに識別子を割り当てます。この識別子を "ハンドル" と呼びます。このハンドルを使って関数 set および get により、オブジェクトのプロパティにアクセスできます。たとえば、次のステートメントではグラフが作成され、lineseries オブジェクトのハンドルが h に出力されます。

```
x = 1:10;
y = x.^3;
h = plot(x,y);
```

ハンドル h を使って lineseries オブジェクトのプロパティを設定することができます。たとえば、Color プロパティを設定することができます。

```
set(h,'Color','red')
```

プロット関数を呼び出すときにも lineseries プロパティを指定することができます。

```
h = plot(x,y,'Color','red');
```

lineseries プロパティをクエリして、現在の値を確認することができます。

```
get(h,'LineWidth')
```

関数 get によって結果が (LineWidth の点の画図を単位として) 返されます。

```
ans =
 0.5000
```

オブジェクトのプロパティの検出

ハンドルのみを指定して get を呼び出すと、MATLAB によってオブジェクトのプロパティのリストが返されます。

```
get(h)
```

ハンドルのみを指定して set を呼び出すと、MATLAB によって、可能な値に関する情報と共にオブジェクトのプロパティのリストが返されます。

```
set(h)
```

オブジェクト プロパティの設定

すべてのオブジェクト プロパティは、既定値をもっています。しかし、いくつかのプロパティの設定を変更してグラフをカスタマイズすることができます。オブジェクト プロパティの設定方法には 2 種類あります。

- オブジェクトの作成時にプロパティに対して値を指定。
- 既存のオブジェクトについてプロパティ値を設定。

プロット コマンドからプロパティを設定

オブジェクト プロパティと値の組は、plot、mesh、surf のような多くのプロット関数の引数として指定することができます。

たとえば、lineseries または surfaceplot オブジェクトを作成するプロット コマンドを使って、プロパティ名/プロパティ値の組を引数として指定できます。コマンド

```
[x,y,z] = peaks;
surf(x,y,z,...
   'FaceColor','interp',...
   'EdgeColor',[.7,.7,.7])
```

は、surfaceplot オブジェクトで内挿された面の色とライト グレーのエッジを使用して、変数 x、y および z のデータをプロットします。

既存のオブジェクトのプロパティを設定

既存のオブジェクトのプロパティ値を変更するには、関数 set を使用します。プロット関数は、作成したデータ オブジェクト (ライン、表面、イメージなど) のハンドルを返します。たとえば、次のステートメントは、5 行 5 列の行列をプロットし (列ごとに 1 つずつ合計 5 つの lineseries オブジェクトを作成)、Marker プロパティを四角に、MarkerFaceColor プロパティを緑に設定します。

```
y = magic(5);
h = plot(y);
set(h,'Marker','s','MarkerFaceColor','g')
```

この場合、h は 5 つのハンドルを含むベクトルで、各ハンドルは、プロット内の 5 つの lineseries の各々です。set ステートメントは、すべての lineseries の Marker および MarkerFaceColor プロパティを同じ値に設定します。

1 つのオブジェクトにプロパティ値を設定するには、ハンドル配列にインデックスを付けます。

```
set(h(1),'LineWidth',2)
```

複数のプロパティ値の設定

各 lineseries のプロパティを違う値に設定する場合は、セル配列を使ってすべてのデータを保存し、set コマンドに渡すことができます。たとえば、プロットを作成し、lineseries のハンドル番号を保存します。

```
h = plot(magic(5));
```

各線に別々のマーカーを追加し、マーカーの面の色を線と同じ色にするものとします。2 つのセル配列を定義しなければなりません。1 つはプロパティ名を含み、もう 1 つはプロパティの希望する値を含みます。

セル配列 prop_name には、2 要素が含まれます。

```
prop_name(1) = {'Marker'};
prop_name(2) = {'MarkerFaceColor'};
```

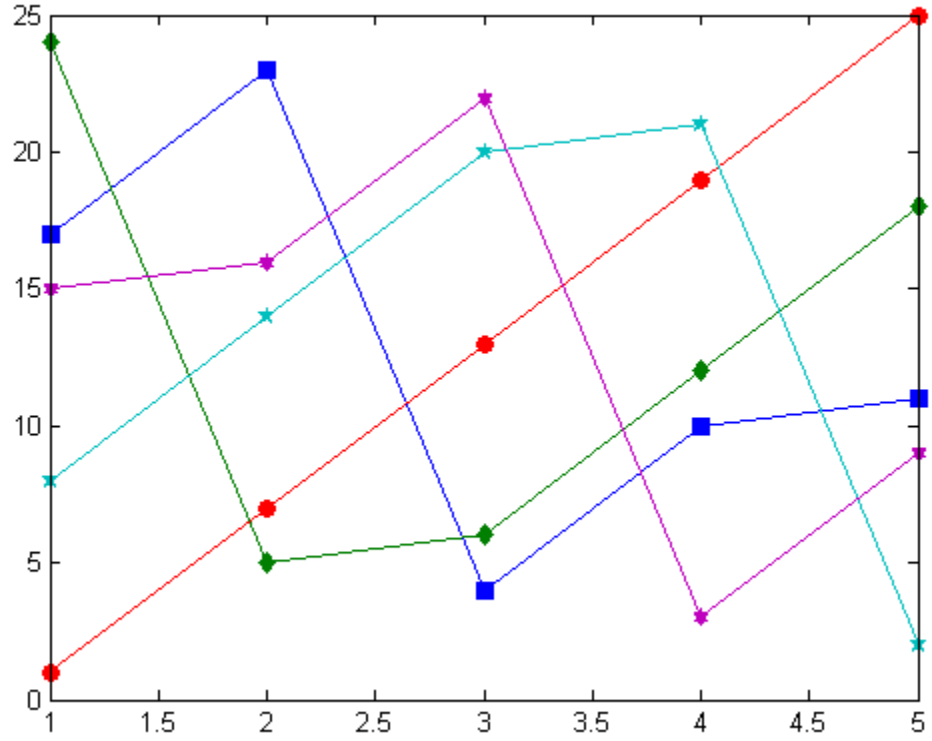
セル配列 prop_values は、10 個の値を含みます。Marker プロパティに対する値が 5 個で、MarkerFaceColor プロパティに対する値が 5 個です。prop_values は、2 次元セル配列であることに注意してください。1 次元目は、値が適用される h のハンドル番号を示し、2 次元目は、値が割り当てられるプロパティを示します。

```
prop_values(1,1) = {'s'};
prop_values(1,2) = {get(h(1),'Color')};
prop_values(2,1) = {'d'};
prop_values(2,2) = {get(h(2),'Color')};
prop_values(3,1) = {'o'};
prop_values(3,2) = {get(h(3),'Color')};
prop_values(4,1) = {'p'};
prop_values(4,2) = {get(h(4),'Color')};
prop_values(5,1) = {'h'};
prop_values(5,2) = {get(h(5),'Color')};
```

MarkerFaceColor には常に、対応するラインの色（関数 get で lineseries の Color プロパティを取得することにより取得）の値が代入されます。

セル配列を定義した後で、set を呼び出して新規のプロパティ値を指定します。

```
set(h,prop_name,prop_values)
```



オブジェクトの利用に関する関数

次の表は、オブジェクトの利用に際して一般的に用いられる関数の一覧です。

関数	目的
allchild	指定したオブジェクトのすべての子オブジェクトを求める。
ancestor	グラフィックス オブジェクトの上位オブジェクトを求める。
copyobj	グラフィックス オブジェクトをコピーする。
delete	オブジェクトの削除。

関数	目的
findall	すべてのグラフィックス オブジェクトの検索 (非表示のハンドルを含む)。
findobj	指定したプロパティ値をもつオブジェクトのハンドルの検索。
gca	現在の座標のハンドル番号を出力。
gcf	現在の Figure のハンドル番号を出力。
gco	現在のオブジェクトのハンドル番号を出力。
get	オブジェクトのプロパティ値の取得。
ishandle	有効なオブジェクト ハンドル番号の検出。
set	オブジェクトのプロパティ値の設定。

座標軸または Figure の指定

MATLAB は、プロット コマンドの実行時に座標軸または Figure が存在しなければ、常にこれを作成します。それでも、プログラム ファイルからグラフィックスを作成する際は、特に他にプログラムを使用する人がいる場合、親座標軸と Figure を明示的に作成および指定することをお勧めします。親を指定することにより、以下の問題を回避することができます。

- プログラムによって現在の Figure にグラフが上書きされます。Figure は、クリックすると現在の Figure になります。
- 現在の Figure は、予期しない状態またはプログラムが予想する通りに動作しない場合があります。

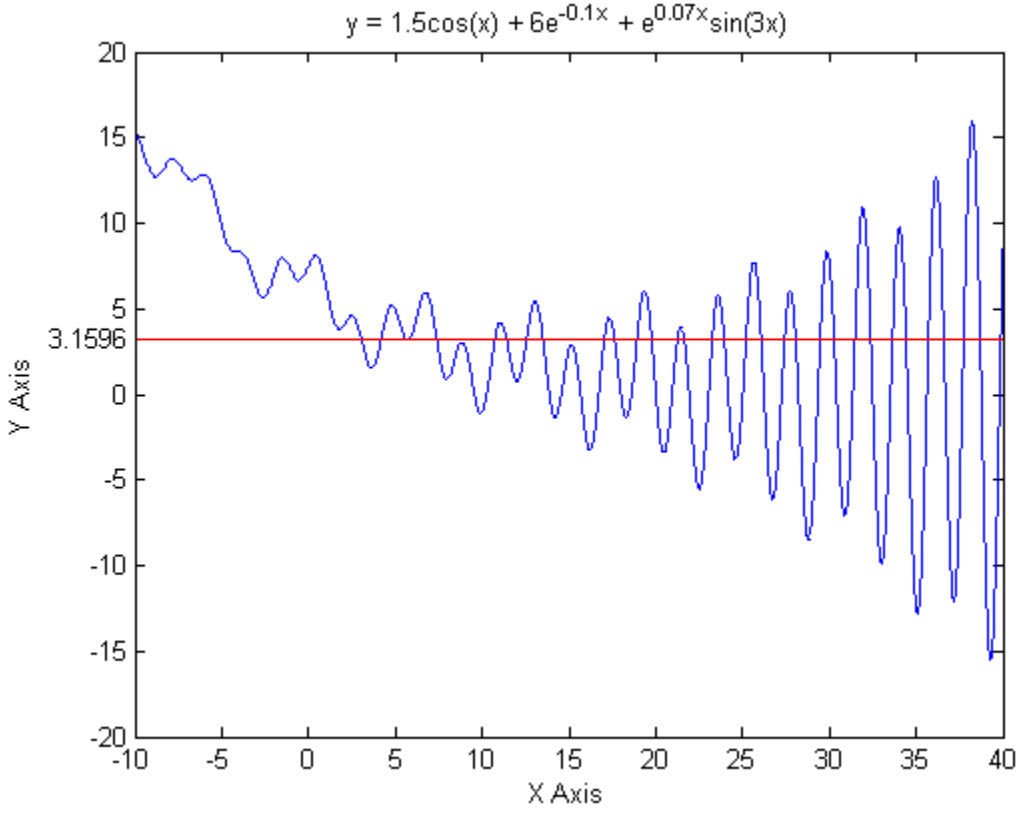
次の例は、入力引数 x で指定された範囲で数式を評価して、結果をプロットする MATLAB 関数です。plot 関数の 2 回目の呼び出しで、結果の mean の値が赤の線でプロットされます。

```
function myfunc(x)
   % Evaluate the expression using the input argument
   y = 1.5*cos(x) + 6*exp(-.1*x) + exp(.07*x).*sin(3*x);

   % Calculate the mean
   ym = mean(y);
```

```
    % Create a figure, axes parented to that axes
    % and the using the axes
    hfig = figure('Name','Function and Mean');
    hax = axes('Parent',hfig);
    plot(hax,x,y)

    % Hold the current plot and add a red line along the mean value
    hold on
    plot(hax,[min(x) max(x)],[ym ym],'Color','red')
    hold off

    % Add a tick label that shows the mean value
    % and add a title and axis labels
    ylab = get(hax,'YTick');
    set(hax,'YTick',sort([ylab ym]))
    title ('y = 1.5cos(x) + 6e^{-0.1x} + e^{0.07x}sin(3x)')
    xlabel('X Axis'); ylabel('Y Axis')
end
```

まず、入力引数の値を定義して、関数を呼び出します。

```
x = -10:.005:40;
myfunc(x)
```



既存のオブジェクトのハンドル番号の検索

関数 findobj を使って、特定のプロパティ値をもつオブジェクトを検索することにより、グラフィックス オブジェクトのハンドル番号を取得することができます。findobj によって、プロパティの値を任意に組み合わせて指定できるため、多くのオブジェクトから簡単に 1 つをに選ぶことができます。findobjは正規表現も認識します。

たとえば、面が青色である四角形のマーカーをもつ青色の線を検索する場合があります。複数の Figure または座標軸が存在する場合には、どれから検索を開始するのかを指定することができます。以下の 4 節では、findobj の使用法を説明する例を述べます。

特定のタイプのすべてのオブジェクトの検索

すべてのオブジェクトはオブジェクト型を識別する Type プロパティをもつため、特定の型のすべてのオブジェクトのハンドルを検索できます。たとえば、

```
h = findobj('Type','patch');
```

は、すべての patch オブジェクトのハンドルを検索します。

特定のプロパティをもつオブジェクトの検索

検索範囲を狭めるために複数のプロパティを指定することができます。たとえば、

```
h = findobj('Type','line','Color','r','LineStyle',':');
```

は、赤い点線のすべてハンドル番号を検索します。

検索範囲の制限

検索を開始する Figure または座標のハンドル番号を引数として渡すことにより、オブジェクト階層内での検索開始点を指定することができます。たとえば、

```
h = findobj(gca,'Type','text','String','¥pi/2');
```

は、現在の座標軸内のみで文字列 π/2 を検索します。

findobj を引数として利用

findobj は検索したオブジェクトのハンドル番号を出力するので、ハンドル引数の代わりに利用することができます。たとえば、

```
set(findobj('Type','line','Color','red'),'LineStyle',':')
```

は、すべての赤色の線を検索し、それらのライン スタイルを点線に設定します。

5

プログラミング

フロー制御

この節の内容…

条件付き制御 - if、else、switch

条件付きステートメントでは、ランタイムに実行するコードのブロックを選択できます。最もシンプルな条件付きステートメントは、if ステートメントです。以下に例を示します。

```
% Generate a random number
a = randi(100, 1);

% If it is even, divide by 2
if rem(a, 2) == 0
    disp('a is even')
    b = a/2;
end
```

if ステートメントでは、オプションのキーワード elseif または else を使用して、代替の選択肢を含めることができます。以下に例を示します。

```
a = randi(100, 1);

if a < 30
    disp('small')
elseif a < 80
    disp('medium')
else
    disp('large')
end
```

または、一連の既知の値を使用して等価性をテストする場合は、switch ステートメントを使用します。以下に例を示します。

```
[dayNum, dayString] = weekday(date, 'long', 'en_US');

switch dayString
   case 'Monday'
      disp('Start of the work week')
   case 'Tuesday'
      disp('Day 2')
   case 'Wednesday'
      disp('Day 3')
   case 'Thursday'
      disp('Day 4')
   case 'Friday'
      disp('Last day of the work week')
   otherwise
      disp('Weekend!')
end
```

if と switch のいずれの場合も、最初の真の条件まで対応するコードが実行されて、コード ブロックが終了されます。各条件付きステートメントでは、end キーワードが必要です。

一般的に、多数の既知の値で条件分岐する場合は、if ステートメントよりも switch ステートメントの方が読みやすくなります。ただし、switch の値と case の値の間で非等価性をテストすることはできません。たとえば、switch で次のような条件を使用することはできません。

```
yourNumber = input('Enter a number: ');

if yourNumber < 0
    disp('Negative')
elseif yourNumber > 0
    disp('Positive')
else
    disp('Zero')
end
```

条件付きステートメントにおける配列の比較

関係演算子と if ステートメントが行列にどのように機能するかを理解することは重要です。2 つの変数間の等価性をチェックする場合、次のようにします。

```
if A == B, ...
```

これは正しい MATLAB のコードで、A と B がスカラーであるときに予想されることを実行します。しかし、A と B が行列の場合、A == B はそれらが "等しいかどうか" をテストせずに、それらの "どこが等しいか" をテストします。結果は、要素ごとの等価性を表わす 0 と 1 からなる行列になります。実際に、A と B が同じサイズでなければ、A == B はエラーになります。

```
A = magic(4);      B = A;      B(1,1) = 0;

A == B
ans =
     0     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

2 つの変数間の等価性をチェックする適切な方法は、関数 isequal を使うことです。

```
if isequal(A, B), ...
```

isequal は、関数 if による結果の表現として、行列ではなくスカラー論理値 1(真を表す)または 0(偽)を出力します。上記の行列 A と B を使って以下を得ます。

```
isequal(A, B)
ans =
    0
```

次に、このことを強調する他の例を示します。A と B がスカラーである場合、次のプログラムは予想通りの結果になります。列を交換した魔方陣を含む行列のほとんどの組に対して、行列の条件 A > B、A < B、または A == B は "どの" 要素に対しても真ではないため、else 句が実行されます。

```
if A > B
   'greater'
elseif A < B
   'less'
elseif A == B
```

```
   'equal'
else
   error('Unexpected situation')
end
```

次の関数は、if と共に使って、行列の比較結果をスカラー条件にするのに使うことができます。

```
isequal
isempty
all
any
```

ループ制御 - for、while、continue、break

この節では、プログラムのループ制御を与える以下の MATLAB 関数を述べます。

for

for ループは、前もって定義した固定回数だけステートメント群を繰り返し実行します。対応する end でステートメント群を区別します。

```
for n = 3:32
   r(n) = rank(magic(n));
end
r
```

ループ内のステートメントの最後のセミコロンは、繰り返し表示されないようにして、最終結果を表示するためループの後に r を使います。

読みやすくするため、特に入れ子状態のときには、ループにインデントを適用することは良い考えです。

```
for i = 1:m
   for j = 1:n
      H(i,j) = 1/(i+j);
   end
end
```

while

while ループは、回数を定義しないで、繰り返し回数を論理条件によりコントロールするときに使います。対応する end でステートメント群を区別します。

while、if、else、end を使ったプログラムを示します。これは、多項式のゼロを見つけるために区間 2 分割を使います。

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
   x = (a+b)/2;
   fx = x^3-2*x-5;
   if sign(fx) == sign(fa)
      a = x; fa = fx;
   else
      b = x; fb = fx;
   end
end
x
```

結果は、多項式 $x^3 - 2x - 5$ の根になります。

```
x =
   2.09455148154233
```

if ステートメントの節で議論した行列比較に関する注意は、while ステートメントにも適用されます。

continue

ステートメント continue は、ループの本体の中で、for、または、while のループの中で、あるステートメントをスキップして、次の反復をコントロールします。入れ子のループ内の continue ステートメントに対して同じことが当てはまります。つまり、continue ステートメントが遭遇したループの初めで実行が続きます。

次の例は、continue ループを使って、ファイル magic.m の中の空白行とコメント行をスキップしたコードのラインをカウントするものです。continue ステートメントを使って、空白行やコメント行に出会うとカウントしないで、magic.m の中の次のラインに進みます。

```
fid = fopen('magic.m','r');
count = 0;
```

```
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) || strncmp(line,'%',1) || ~ischar(line)
        continue
    end
    count = count + 1;
end
fprintf('%d lines¥n',count);
fclose(fid);
```

break

break ステートメントは、for または while ループから抜け出すためのものです。入れ子になったループでは、break は最下部のループのみから退出します。

前の節の例を改良したものを以下に示します。break の使用はなぜ良い考えなのでしょうか?

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
   x = (a+b)/2;
   fx = x^3-2*x-5;
   if fx == 0
      break
   elseif sign(fx) == sign(fa)
      a = x; fa = fx;
   else
      b = x; fb = fx;
   end
end
x
```

プログラムの終了 - return

この節では、プログラムを最後まで実行せず途中で終了させる MATLAB 関数 return について述べます。

出力

return はコマンドの現在のシーケンスを終了し、起動関数またはキーボードに制御を戻します。return は、キーボード モードの終了にも使用されます。呼ばれた関数は通常その機能を終了すると、それを起動した関数に制御を移します。呼ばれた関数の中に return を挿入することで強制的に早く終了し起動した関数に制御を移すことができます。

ベクトル化

MATLAB をスピードアップする 1 つの方法は、ユーザーの M ファイルのアルゴリズムをベクトル化することです。他のプログラミング言語が for または DO ループを使う部分で、MATLAB ではベクトルまたは行列演算を使うことができます。次のコードは、簡単な例として、対数表を作成するものです。

```
x = .01;
for k = 1:1001
   y(k) = log10(x);
   x = x + .01;
end
```

同じコードをベクトル化すると、次のようになります。

```
x = .01:.01:10;
y = log10(x);
```

さらに複雑なコードに対しては、ベクトル化の効果が常に顕著とは限りません。

事前割り当て

コードの一部をベクトル化することができない場合、出力結果をストアする ベクトルまたは配列を前もって設定しておくことによって for ループをスピードアップできます。たとえば、次のコードは、for ループで作成されるベクトルを前もって作成するために関数 zeros を使うものです。これにより、for ループの実行はかなり速くなります。

```
r = zeros(32,1);
for n = 1:32
    r(n) = rank(magic(n));
end
```

前の例の中で前もってスペースを設定しないと、MATLAB インタープリターは、ループに関する 1 回の実行で、1 要素ずつ r ベクトルを拡大します。ベクトルの領域を前もって確保することは、この手順を省略させ、より高速の実行になります。

スクリプトとファンクション

この節の内容…

概要

MATLAB 製品は、対話形式の計算環境と同様に強力なプログラミング言語を提供します。MATLAB コマンド ラインから 1 つずつ言語のコマンドを入力するか、MATLAB 関数を実行するように一連のコマンドをファイルにして実行できます。関数ファイルを作成するには、MATLAB エディターまたはその他のテキスト エディターを使用します。その他の MATLAB 関数またはコマンドを呼び出すのと同様に、これらの関数を呼び出します。

プログラム ファイルには 2 種類のものがあります。

- スクリプト、このファイルは、入力引数を受け入れたり、出力引数を出力したりしません。このファイルは、ワークスペースの中のデータで機能します。
- ファンクション、このファイルは、入力引数を受け入れ、出力引数を出力します。内部変数は、関数のローカル変数です。

ユーザーが MATLAB のプログラミングの初心者であれば、実行するプログラム ファイルを現在のフォルダーに作成してください。ユーザー自身のファイルを多数作成したときには、他のフォルダーや個人のツールボックスにそれらをまとめて、MATLAB の検索パスに加えてください。

関数名が重複すると、MATLAB は検索パスの中で最初に現れるファイルを実行します。

たとえば、プログラム ファイル myfunction.m の内容を表示するには、次のようにします。

```
type myfunction
```

スクリプト

ユーザーがスクリプトを読み込むと、MATLAB は、単にファイルの中のコマンドを実行します。スクリプトは、ワークスペースの中に存在するデータを取り扱う、または演算するための新しいデータを作成します。スクリプトは、出力引数を出力しませんが、作成する変数はワークスペースに残り、その後の計算に使われます。加えて、スクリプトは、plot コマンド等を使って、グラフィカル出力を作成することができます。

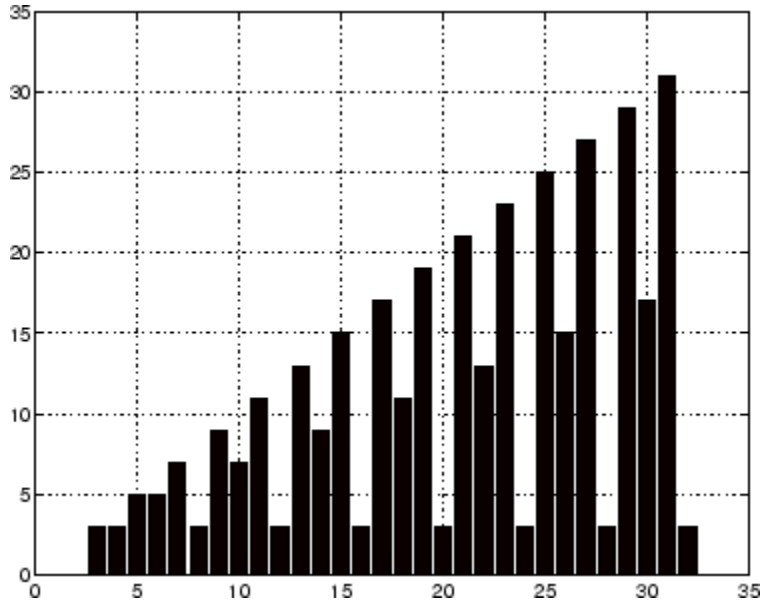
たとえば、次の MATLAB コマンドを含んだ magicrank.m と呼ばれるファイルを作成します。

```
% Investigate the rank of magic squares
r = zeros(1,32);
for n = 3:32
   r(n) = rank(magic(n));
end
r
bar(r)
```

ステートメント

```
magicrank
```

により、MATLAB はコマンドを実行し、最初の 30 個の魔方陣のランクを計算し、結果を棒グラフにプロット表示します。ファイルの実行が終了すると、変数 n と r がワークスペースに残ります。



関数

関数は、入力引数をもち、出力引数を出力するファイルです。ファイルの名前と関数の名前は同じものにしてください。関数 M ファイルは、それ自身がもつワークスペースの中で変数を使い、MATLAB コマンド プロンプトでアクセスするワークスペースと区別します。

良い例は、rank を使って示されます。ファイル rank.m は、次のフォルダーにあります。

```
toolbox/matlab/matfun
```

ファイルの内容は、次のコマンドで表示されます。

```
type rank
```

内容は、次のとおりです。

```
function r = rank(A,tol)
%   RANK Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
```

```
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

s = svd(A);
if nargin==1
   tol = max(size(A')) * max(s) * eps;
end
r = sum(s > tol);
```

関数の最初の行は、function というキーワードで始まります。これは、関数名と引数の順序を与えるものです。この場合、入力引数は 2 つで、出力引数は 1 です。

最初の空白行または実行可能行までの数行は、ヘルプ テキストを与えるコメント行です。これらの行は、ユーザーが

```
help rank
```

と入力すると表示されます。ヘルプ テキストの最初の行は H1 ラインと呼ばれ、lookfor コマンドを使用するか、またはフォルダー上の help を呼び出すことにより MATLAB で表示されるものです。

ファイルの残りは、関数を定義している実行可能な MATLAB コードです。最初の行の中の変数 r、A、tol と同様に関数の中に導入されている変数 s は、すべて関数のローカル変数です。すなわち、MATLAB ワークスペースの中の変数とは異なるものです

この例は、MATLAB 関数の一つの見方を示すもので、通常の他のプログラミング言語の中には見られません。すなわち、引数の数を可変にできます。関数 rank は、種々の方法で使うことができます。

```
rank(A)
r = rank(A)
r = rank(A,1.e-6)
```

多くの関数はこのように機能します。出力引数を設定しなければ、結果は ans に保存されます。2 番目の入力引数が設定されなければ、関数は既定の値を使って計算します。関数の中で、nargin と nargout と名付けられた 2 つの量は、関数の中で特殊な使い方で、入力引数の数や出力引数の数を出力するものです。関数 rank は、nargin を使っていますが、nargout は使う必要がありません。

関数のタイプ

MATLAB は、プログラミングで利用する異なるタイプの関数を提供しています。

無名関数

″無名関数″ は、シンプルな形式の MATLAB 関数であり、単一の MATLAB ステートメント内で定義されます。これは、単一の MATLAB 表現と任意数の入出力引数で構成されます。無名関数は、MATLAB コマンド ラインから、または関数やスクリプト内で定義できます。これは、ファイルをその都度作らなくても単純な関数を簡単に作成できる方法です。

式から無名関数を作成するための構文は、以下のとおりです。

```
f = @(arglist)expression
```

下記のステートメントは、数値のニ乗を計算する無記名関数を作成します。この関数を呼び出すと、MATLAB は渡された値を変数 x に割り当て、x を使って x.^2 式を計算します。

```
sqr = @(x) x.^2;
```

上で定義した関数 sqr を実行するには、次のように入力します。

```
a = sqr(5)
a =
   25
```

基本関数とサブ関数

無名でない関数はファイル内で定義しなければなりません。各関数ファイルには、必須の ″基本関数″ を先頭とし、それに続いて任意の数の ″サブ関数″ が含まれます。基本関数は、サブ関数よりも広い範囲をもちます。つまり、基本関数はそれを定義するファイルの外 (たとえば、MATLAB コマンド ラインや他のファイル内の関数など) から呼び出せますが、サブ関数は呼び出せません。サブ関数は、そのファイル内の基本関数と他のサブ関数からのみ参照できます。

関数 (p. 5-12)の節に示した関数 rank は、基本関数の例です。

プライベート関数

"プライベート関数" は、基本関数の 1 つのタイプです。プライベート関数の固有な特徴は、他の関数の限られたグループからのみ、見ることができる点です。このタイプの関数は、関数へのアクセスを制限する場合、または関数のインプリメンテーションを外部に見えないように選択する場合に有効です。

プライベート関数は特別な名前 private. でサブフォルダーに含まれています。これらは、親フォルダー内でのみ参照できる関数です。たとえば、フォルダー newmath が MATLAB 検索パス上にあると仮定します。private という newmath のサブフォルダーには、newmath 内にある関数のみから呼び出せる関数を含めることができます。

プライベート関数は親フォルダーの外からは参照できないため、他のフォルダー内で同じ関数名を使用できます。この機能は、他のフォルダーに元の関数を残したまま、特定の関数の固有バージョンを作成する場合に便利です。MATLAB では、標準のファイル関数よりも先にプライベート関数が検索されるため、プライベートではないファイル test.m より先に、test.m というプライベート関数が検索されます。

入れ子関数

他の関数の本体内で関数を定義できます。これらは、外部関数の入れ子と呼ばれます。入れ子関数には、他の関数のいずれか、またはすべてが含まれています。次の例では、関数 B は関数 A の入れ子です。

```
function x = A(p1, p2)
...
B(p2)
   function y = B(p3)
   ...
   end
...
end
```

他の関数と同様に、入れ子関数は、関数が用いる変数を格納する独自のワークスペースをもちます。しかし、入れ子になっているすべての関数のワークスペースにもアクセスします。そのため、たとえば、基本関数によって割り当てられた値をもつ変数は、基本関数内の任意のレベルの入れ子関数によって読み込み、または上書きすることができます。同様に、入れ子関数内で割り当てられた変数は、どの関数を含む任意の関数によって読み込み、または上書きすることができます。

グローバル変数

1 つの変数の単一コピーを共有し、複数の関数で使いたい場合、変数をすべての関数の中で単に global として宣言してください。変数をアクセスするために基本ワークスペースを使う場合、コマンド ラインで同じこと(グローバル宣言)をします。グローバル宣言は、変数が実際に関数の中で使われる前に行わなければなりません。絶対に必要ではありませんが、大文字を使うことにより、他の変数と区別できて便利です。たとえば、falling.m というファイルに新しい関数を作成します。

```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

そして、対話的にステートメントを入力します。

```
global GRAVITY
GRAVITY = 32;
y = falling((0:.1:5)');
```

2 つのグローバル ステートメントが、関数の内部で利用可能なコマンド プロンプトで GRAVITY に値を割り当てます。いくつかのファイルを編集しないで、対話的に GRAVITY を変更して、新しい解を得ることもできます。

コマンドと関数の構文

かっこや引用符を使わずに文字列引数を受け取る MATLAB 関数を作成することができます。つまり、MATLAB は

```
foo a b c
```

として

```
foo('a','b','c')
```

ただし、引用符なしのコマンド形式を使う場合、MATLAB は出力引数を返せません。たとえば、次の例を考えてみましょう。

```
legend apples oranges
```

は、文字列 apples と oranges をラベルとして使ってプロットに凡例を作成します。legend コマンドにより出力引数を出力させたい場合は、引用符付きの形式を使う必要があります。

```
[legh,objh] = legend('apples','oranges');
```

さらに、引数のいずれかが文字列でない場合も引用符付きの形式を使う必要があります。

注意:　引用符なしのコマンド構文は便利である一方、MATLAB がエラーを出す原因とならないながらも、不適切に使用される場合があります。

コード内の文字列引数の作成

引用符付きの形式の関数を使って、コード内で文字列引数を作成することができます。次の例は、複数のデータ ファイル、August1.dat、August2.dat 等を処理します。ファイル名を作成するために整数を文字に変換する関数 int2str を使っています。

```
for d = 1:31
   s = ['August' int2str(d) '.dat'];
   load(s)
   % Code to process the contents of the d-th file
end
```