Tutorial 2

Programming in MATLAB

Edward Neuman
Department of Mathematics
Southern Illinois University at Carbondale
edneuman@siu.edu

This tutorial is intended for those who want to learn basics of MATLAB programming language. Even with a limited knowledge of this language a beginning programmer can write his/her own computer code for solving problems that are complex enough to be solved by other means. Numerous examples included in this text should help a reader to learn quickly basic programming tools of this language. Topics discussed include the m-files, inline functions, control flow, relational and logical operators, strings, cell arrays, rounding numbers to integers and MATLAB graphics.

2.1 The m-files

Files that contain a computer code are called the *m-files*. There are two kinds of m-files: the *script files* and the *function files*. Script files do not take the input arguments or return the output arguments. The function files may take input arguments or return output arguments.

To make the m-file click on **File** next select **New** and click on **M-File** from the pull-down menu. You will be presented with the **MATLAB Editor/Debugger** screen. Here you will type your code, can make changes, etc. Once you are done with typing, click on **File**, in the **MATLAB Editor/Debugger** screen and select **Save As…** . Chose a name for your file, e.g., **firstgraph.m** and click on **Save**. Make sure that your file is saved in the directory that is in MATLAB's search path.

If you have at least two files with duplicated names, then the one that occurs first in MATLAB's search path will be executed.

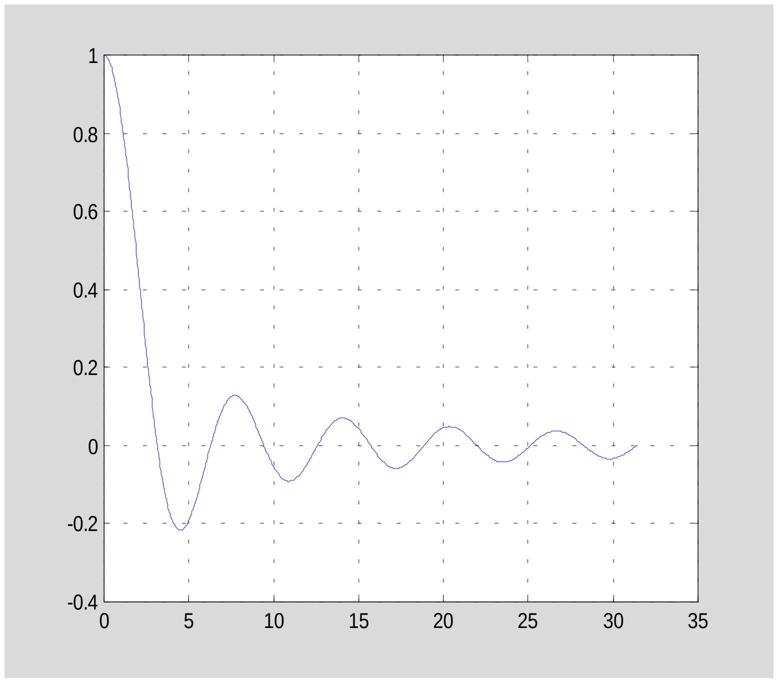
To open the m-file from within the **Command Window** type **edit firstgraph** and then press **Enter** or **Return** key.

Here is an example of a small script file

```
% Script file firstgraph.

x = pi/100:pi/100:10*pi;
y = sin(x)./x;
plot(x,y)
grid
```

Let us analyze contents of this file. First line begins with the percentage sign **%**. This is a *comment*. All comments are ignored by MATLAB. They are added to improve readability of the code. In the next two lines arrays **x** and **y** are created. Note that the semicolon follows both commands. This suppresses display of the content of both vectors to the screen (see Tutorial 1, page 5 for more details). Array **x** holds 1000 evenly spaced numbers in the interval **[π/100 10π]** while the array **y** holds the values of the *sinc function* **y = sin(x)/x** at these points. Note use of the *dot operator* **.** before the *right division operator* **/**. This tells MATLAB to perform the *componentwise division* of two arrays **sin(x)** and **x**. Special operators in MATLAB and operations on one- and two dimensional arrays are discussed in detail in Tutorial 3, Section 3.2. The command **plot** creates the graph of the sinc function using the points generated in two previous lines. For more details about command **plot** see Section 2.8.1 of this tutorial. Finally, the command **grid** is executed. This adds a *grid* to the graph. We invoke this file by typing its name in the **Command Window** and next pressing the **Enter** or **Return** key

```
firstgraph
```



Here is an example of the function file

```
function [b, j] = descsort(a)

% Function descsort sorts, in the descending order, a real array a.
% Second output parameter j holds a permutation used to obtain
% array b from the array a.
```

```
[b ,j] = sort(-a);
b = -b;
```

This function takes one input argument, the array of real numbers, and returns a sorted array together with a permutation used to obtain array **b** from the array **a**. MATLAB built-in function **sort** is used here. Recall that this function sort numbers in the ascending order. A simple trick used here allows us to sort an array of numbers in the descending order.

To demonstrate functionality of the function under discussion let

```
a = [pi –10 35 0.15];

[b, j] = descsort(a)

b =
   35.0000    3.1416    0.1500  -10.0000
j =
     3     1     4     2
```

You can execute function **descsort** without output arguments. In this case an information about a permutation used will be lost

```
descsort(a)

ans =
   35.0000    3.1416    0.1500  -10.0000
```

Since no output argument was used in the call to function **descorder** a sorted array **a** is assigned to the default variable **ans**.

2.2 Inline functions and the feval command

Sometimes it is handy to define a function that will be used during the current MATLAB session only. MATLAB has a command **inline** used to define the so-called *inline functions* in the **Command Window**.

Let

```
f = inline('sqrt(x.^2+y.^2)','x','y')

f =
     Inline function:
     f(x,y) = sqrt(x.^2+y.^2)
```

You can evaluate this function in a usual way

```
f(3,4)

ans =
     5
```

Note that this function also works with arrays. Let

```
A = [1 2;3 4]

A =
     1     2
     3     4
```

and

```
B = ones(2)

B =
     1     1
     1     1
```

Then

```
C = f(A, B)

C =
    1.4142    2.2361
    3.1623    4.1231
```

For the later use let us mention briefly a concept of the *string* in MATLAB. The *character string* is a text surrounded by single quotes. For instance,

```
str =  'programming in MATLAB is fun'

str =
programming in MATLAB is fun
```

is an example of the string. Strings are discussed in Section 2.5 of this tutorial.

In the previous section you have learned how to create the function files. Some functions take as the input argument a name of another function, which is specified as a strin*g*. In order to execute function specified by string you should use the command **feval** as shown below

feval('functname', input parameters of function functname)

Consider the problem of computing the *least common multiple* of two integers. MATLAB has a built-in function **lcm** that computes the number in question. Recall that the least common multiple and the *greatest common divisor* (**gcd**) satisfy the following equation

$$ab = \text{lcm}(a, b)\text{gcd}(a, b)$$

MATLAB has its own function, named **gcd**, for computing the greatest common divisor.

To illustrate the use of the command **feval** let us take a closer look at the code in the m-file **mylcm**

```
 function c = mylcm(a, b)

% The least common multiple c of two integers a and b.

if feval('isint',a) & feval('isint',b)
   c = a.*b./gcd(a,b);
else
   error('Input arguments must be integral numbers')
end
```

Command **feval** is used twice in line two (I do do not count the comment lines and the blank lines). It checks whether or not both input arguments are integers. The *logical and operator* **&** used here is discussed in Section 2.4. If this condition is satisfied, then the least common multiple is computed using the formula mentioned earlier, otherwise the error message is generated. Note use of the command **error,** which takes as the argument a string. The conditional **if - else - end** used here is discussed in Section 2.4 of this tutorial. Function that is executed twice in the body of the function **mylcm** is named **isint**

```
function k = isint(x);

% Check whether or not x is an integer number.
% If it is, function isint returns 1 otherwise it returns 0.

if abs(x - round(x)) < realmin
   k = 1;
else
   k = 0;
end
```

New functions used here are the absolute value function (**abs**) and the round function (**round**). The former is the classical math function while the latter takes a number and rounds it to the closest integer. Other functions used to round real numbers to integers are discussed in Section 2.7. Finally, **realmin** is the smallest positive real number on your computer

```
format long

realmin

ans =
    2.225073858507201e-308

format short
```

The Trapezoidal Rule with the correction term is often used to numerical integration of functions that are differentiable on the interval of integration

$$\int_a^b f(x)dx \approx \frac{h}{2}[f(a)+f(b)] - \frac{h^2}{12}[f'(a)-f'(b)]$$

where **h = b – a**. This formula is easy to implement in MATLAB

```
function y = corrtrap(fname, fpname, a, b)

% Corrected trapezoidal rule y.
% fname - the m-file used to evaluate the integrand,
% fpname - the m-file used to evaluate the first derivative
% of the integrand,
% a,b - endpoinds of the interval of integration.

h = b - a;
y = (h/2).*(feval(fname,a) + feval(fname,b))+ (h.^2)/12.*( ...
   feval(fpname,a) - feval(fpname,b));
```

The input parameters **a** and **b** can be arrays of the same dimension. This is possible because the dot operator proceeds certain arithmetic operations in the command that defines the variable **y**.

In this example we will integrate the sine function over two intervals whose end points are stored in the arrays **a** and **b**, where

```
a = [0 0.1];
b = [pi/2 pi/2 + 0.1];

y = corrtrap('sin', 'cos', a, b)

y =
    0.9910    1.0850
```

Since the integrand and its first order derivative are both the built-in functions, there is no need to define these functions in the m-files.

2.3 Control flow

To control the flow of commands, the makers of MATLAB supplied four devices a programmer can use while writing his/her computer code

- the **for** loops
- the **while** loops
- the **if-else-end** constructions
- the **switch-case** constructions

2.3.1 Repeating with **for** loops

Syntax of the **for** loop is shown below

```
for k = array
    commands
end
```

The commands between the **for** and **end** statements are executed for all values stored in the **array**.

Suppose that one-need values of the sine function at eleven evenly spaced points **πn/10**, for **n = 0, 1, …, 10**. To generate the numbers in question one can use the **for** loop

```
for n=0:10
   x(n+1) = sin(pi*n/10);
end


x

x =
  Columns 1 through 7
         0    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511
  Columns 8 through 11
    0.8090    0.5878    0.3090    0.0000
```

The **for** loops can be nested

```
H = zeros(5);
for k=1:5
   for l=1:5
       H(k,l) = 1/(k+l-1);
   end
end

H

H =
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
```

Matrix **H** created here is called the *Hilbert matrix*. First command assigns a space in computer's memory for the matrix to be generated. This is added here to reduce the overhead that is required by loops in MATLAB.

The **for** loop should be used only when other methods cannot be applied. Consider the following problem. Generate a 10-by-10 matrix **$A = [a_{kl}]$**, where **$a_{kl} = \sin(k)\cos(l)$**. Using nested loops one can compute entries of the matrix **A** using the following code

```
A = zeros(10);
for k=1:10
   for l=1:10
      A(k,l) = sin(k)*cos(l);
   end
end
```

A loop free version might look like this

```
k = 1:10;
A = sin(k)'*cos(k);
```

First command generates a row array **k** consisting of integers 1, 2, … , 10. The command **sin(k)'** creates a column vector while **cos(k)** is the row vector. Components of both vectors are the values of the two trig functions evaluated at **k**. Code presented above illustrates a powerful feature of MATLAB called *vectorization*. This technique should be used whenever it is possible.

2.3.2 Repeating with while loops

Syntax of the **while** loop is

```
while expression
  statements
end
```

This loop is used when the programmer does not know the number of repetitions a priori.

Here is an almost trivial problem that requires a use of this loop. Suppose that the number π is divided by 2. The resulting quotient is divided by 2 again. This process is continued till the current quotient is less than or equal to 0.01. What is the largest quotient that is greater than 0.01? To answer this question we write a few lines of code

```
q = pi;
while q > 0.01
   q = q/2;
end

q

q =
    0.0061
```

2.3.3 The if-else-end constructions

Syntax of the simplest form of the construction under discussion is

```
if expression
  commands
end
```

This construction is used if there is one alternative only. Two alternatives require the construction

```
if expression
  commands (evaluated if expression is true)
else
  commands (evaluated if expression is false)
end
```

Construction of this form is used in functions **mylcm** and **isint** (see Section 2.3).

If there are several alternatives one should use the following construction

```
if expression1
  commands (evaluated if expression 1 is true)
elseif expression 2
  commands (evaluated if expression 2 is true)
elseif …
.
.
.
else
  commands (executed if all previous expressions evaluate to false)
end
```

Chebyshev polynomials $T_n(x)$, $n = 0, 1, \dots$ of the first kind are of great importance in numerical analysis. They are defined recursively as follows

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad n = 2, 3, \dots, \; T_0(x) = 1, T_1(x) = x.$$

Implementation of this definition is easy

```
function T = ChebT(n)

% Coefficients T of the nth Chebyshev polynomial of the first kind.
% They are stored in the descending order of powers.

t0 = 1;
t1 = [1 0];
if n == 0
   T = t0;
elseif n == 1;
   T = t1;
else
   for k=2:n
      T = [2*t1 0] - [0 0 t0];
      t0 = t1;
      t1 = T;
   end
end
```

Coefficients of the cubic Chebyshev polynomial of the first kind are

```
coeff = ChebT(3)

coeff =
     4     0    -3     0
```

Thus **$T_3(x) = 4x^3 - 3x$**.

2.3.4 The switch-case construction

Syntax of the **switch-case** construction is

```
switch expression (scalar or string)
  case value1 (executes if expression evaluates to value1)
     commands
  case value2 (executes if expression evaluates to value2)
     commands
.
.
.
  otherwise
    statements
end
```

Switch compares the input expression to each case value. Once the match is found it executes the associated commands.

In the following example a random integer number x from the set {1, 2, … , 10} is generated. If x = 1 or x = 2, then the message Probability = 20% is displayed to the screen. If x = 3 or 4 or 5, then the message Probability = 30% is displayed, otherwise the message Probability = 50% is generated. The script file **fswitch** utilizes a switch as a tool for handling all cases mentioned above

```
% Script file fswitch.

x = ceil(10*rand);  % Generate a random integer in {1, 2, ... , 10}
switch x
   case {1,2}
      disp('Probability = 20%');
   case {3,4,5}
      disp('Probability = 30%');
   otherwise
      disp('Probability = 50%');
end
```

Note use of the curly braces after the word **case**. This creates the so-called *cell array* rather than the one-dimensional array, which requires use of the square brackets.

Here are new MATLAB functions that are used in file **fswitch**

rand – uniformly distributed random numbers in the interval (0, 1)
ceil – round towards plus infinity infinity (see Section 2.5 for more details)
disp – display string/array to the screen

Let us test this code ten times

```
for k = 1:10
   fswitch
end
```

```
Probability = 50%
Probability = 30%
Probability = 50%
Probability = 50%
Probability = 50%
Probability = 30%
Probability = 20%
Probability = 50%
Probability = 30%
Probability = 50%
```

2.4 Relations and logical operators

Comparisons in MATLAB are performed with the aid of the following operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater
>=	Greater or equal to
==	Equal to
~=	Not equal to

Operator == compares two variables and returns ones when they are equal and zeros otherwise.

Let

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

```
a =
     1     1     3     4     1
```

Then

```
ind = (a == 1)

ind =
     1     1     0     0     1
```

You can extract all entries in the array **a** that are equal to 1 using

```
b = a(ind)

b =
     1     1     1
```

This is an example of so-called *logical addressing* in MATLAB. You can obtain the same result using function **find**

```
ind = find(a == 1)

ind =
     1     2     5
```

Variable **ind** now holds indices of those entries that satisfy the imposed condition. To extract all ones from the array **a** use

```
b = a(ind)

b =
     1     1     1
```

There are three *logical operators* available in MATLAB

Logical operator	Description
|	And
&	Or
~	Not

Suppose that one wants to select all entries **x** that satisfy the inequalities **x** ρ **1** or **x** < **-0.2** where

```
x = randn(1,7)

x =
   -0.4326   -1.6656    0.1253    0.2877   -1.1465    1.1909    1.1892
```

is the array of normally distributed random numbers. We can solve easily this problem using operators discussed in this section

```
ind = (x  >= 1) | (x < -0.2)

ind =
     1     1     0     0     1     1     1

y = x(ind)
```

```
y =
   -0.4326   -1.6656   -1.1465    1.1909    1.1892
```

Solve the last problem without using the logical addressing.

In addition to relational and logical operators MATLAB has several logical functions designed for performing similar tasks. These functions return **1** (true) if a specific condition is satisfied and **0** (false) otherwise. A list of these functions is too long to be included here. The interested reader is referred to [1], pp. 85-86 and [4], Chapter 10, pp. 26-27. Names of the most of these functions begin with the prefix **is**. For instance, the following command

```
isempty(y)

ans =
     0
```

returns **0** because the array **y** of the last example is not empty. However, this command

```
isempty([ ])

ans =
     1
```

returns **1** because the argument of the function used is the empty array **[ ]**.

Here is another example that requires use of the **isempty** command

```
function dp = derp(p)

% Derivative dp of an algebraic polynomial that is
% represented by its coefficients p. They must be stored
% in the descending order of powers.

n = length(p) - 1;
p = p(:)';                    % Make sure p is a row array.
dp = p(1:n).*(n:-1:1);        % Apply the Power Rule.
k = find(dp ~= 0);
if ~isempty(k)
   dp = dp(k(1):end);         % Delete leading zeros if any.
else
   dp = 0;
end
```

In this example $p(x) = x^3 + 2x^2 + 4$. Using a convention for representing polynomials in MATLAB as the array of their coefficients that are stored in the descending order of powers, we obtain

```
dp = derp([1 2 0 4])

dp =
     3     4     0
```

2.5 Strings

String is an array of characters. Each character is represented internally by its ASCII value.

This is an example of a string

```
str = 'I am learning MATLAB this semester.'

str =
I am learning MATLAB this semester.
```

To see its ASCII representation use function **double**

```
str1 = double(str)

str1 =
  Columns 1 through 12
    73    32    97   109    32   108   101    97   114   110   105
110
  Columns 13 through 24
   103    32    77    65    84    76    65    66    32   116   104
105
  Columns 25 through 35
   115    32   115   101   109   101   115   116   101   114    46
```

You can convert array **str1** to its character form using function **char**

```
str2 = char(str1)

str2 =
I am learning MATLAB this semester.
```

Application of the string conversion is used in Tutorial 3, Section 3.11 to uncode and decode messages.

To compare two strings for equality use function **strcmp**

```
iseq = strcmp(str, str2)

iseq =
     1
```

Two strings can be concatenated using function **ctrcat**

```
strcat(str,str2)

ans =
I am learning MATLAB this semester.I am learning MATLAB this semester.
```

Note that the concatenated strings are not separated by the blank space.

You can create two-dimensional array of strings. To this aim the *cell array* rather than the two-dimensional array must be used. This is due to the fact that numeric array must have the same number of columns in each row.

This is an example of the cell array

```
carr = {'first name'; 'last name'; 'hometown'}

carr =
    'first name'
    'last name'
    'hometown'
```

Note use of the curly braces instead of the square brackets. Cell arrays are discussed in detail in the next section of this tutorial.

MATLAB has two functions to categorize characters: **isletter** and **isspace**. We will run both functions on the string **str**

```
isletter(str)

ans =
  Columns 1 through 12
     1     0     1     1     0     1     1     1     1     1     1
1
  Columns 13 through 24
     1     0     1     1     1     1     1     1     0     1     1
1
  Columns 25 through 35
     1     0     1     1     1     1     1     1     1     1     0
```

```
isspace(str)

ans =
  Columns 1 through 12
     0     1     0     0     1     0     0     0     0     0     0
0
  Columns 13 through 24
     0     1     0     0     0     0     0     0     1     0     0
0
  Columns 25 through 35
     0     1     0     0     0     0     0     0     0     0     0
```

The former function returns 1 if a character is a letter and 0 otherwise while the latter returns 1 if a character is whitespace (blank, tab, or new line) and 0 otherwise.

We close this section with two important functions that are intended for conversion of numbers to strings. Functions in question are named **int2str** and **num2str**. Function **int2str** rounds its argument (matrix) to integers and converts the result into a string matrix.

Let

```
A = randn(3)

A =
   -0.4326    0.2877    1.1892
   -1.6656   -1.1465   -0.0376
    0.1253    1.1909    0.3273
```

Then

```
B = int2str(A)

B =
 0  0  1
-2 -1  0
 0  1  0
```

Function **num2str** takes an array and converts it to the array string. Running this function on the matrix **A** defined earlier, we obtain

```
C = num2str(A)

C =
-0.43256     0.28768      1.1892
 -1.6656     -1.1465   -0.037633
 0.12533      1.1909     0.32729
```

Function under discussion takes a second optional argument - a number of decimal digits. This feature allows a user to display digits that are far to the right of the decimal point. Using matrix A again, we get

```
D = num2str(A, 18)

D =
-0.43256481152822068      0.28767642035854885       1.1891642016521031
  -1.665584378238097      -1.1464713506814637     -0.037633276593317645
 0.12533230647483068       1.1909154656429988      0.32729236140865414
```

For comparison, changing format to **long**, we obtain

```
format long

A

A =
  -0.43256481152822   0.28767642035855   1.18916420165210
  -1.66558437823810  -1.14647135068146  -0.03763327659332
   0.12533230647483   1.19091546564300   0.32729236140865


format short
```

Function **num2str** his is often used for labeling plots with the **title**, **xlabel**, **ylabel**, and **text** commands.

2.6 Cell arrays

Two data types the *cell arrays* and *structures* make MATLAB a powerful tool for applications. They hold other MATLAB arrays. In this section we discuss the cell arrays only. To learn about structures the interested reader is referred to [4], Chapter 13 and [1], Chapter 12.

To create the cell array one can use one of the two techniques called the *cell indexing* and the *content indexing*. The following example reveals differences between these two techniques. Suppose one want to save the string 'John Brown' and his SSN 123-45-6789 (without dashes) in the cell array.

1. Cell indexing

```
A(1,1) = {'John Brown'};

A(1,2) = {[1 2 3 4 5 6 7 8 9]};
```

2. Content indexing

```
B{1,1} = 'John Brown';

B{1,2} = [1 2 3 4 5 6 7 8 9];
```

A condensed form of the cell array **A** is

```
A

A =
    'John Brown'    [1x9 double]
```

To display its full form use function **celldisp**

```
celldisp(A)

A{1} =
John Brown
A{2} =
     1     2     3     4     5     6     7     8     9
```

To access data in a particular cell use content indexing on the right-hand side. For instance,

```
B{1,1}

ans =
John Brown
```

To delete a cell use the empty matrix operator **[ ]**. For instance, this operation

```
B(1) = []

B =
    [1x9 double]
```

deletes cell **B(1, 1)** of the cell array **B**.

This command

```
C = {A B}

C =
    {1x2 cell}    {1x1 cell}
```

creates a new cell array

```
celldisp(C)

C{1}{1} =
John Brown
C{1}{2} =
     1     2     3     4     5     6     7     8     9
C{2}{1} =
     1     2     3     4     5     6     7     8     9
```

How would you delete cell **C(2,1)**?

2.7 Rounding to integers. Functions ceil, floor, fix and round

We have already used two MATLAB functions **round** and **ceil** to round real numbers to integers. They are briefly described in the previous sections of this tutorial. A full list of functions designed for rounding numbers is provided below

Function	Description
floor	Round towards minus infinity
ceil	Round towards plus infinity
fix	Round towards zero
round	Round towards nearest integer

To illustrate differences between these functions let us create first a two-dimensional array of random numbers that are normally distributed (mean = 0, variance = 1) using another MATLAB function **randn**

```
randn('seed', 0)       % This sets the seed of the random numbers generator to zero

T = randn(5)
```

```
T =
    1.1650    1.6961   -1.4462   -0.3600   -0.0449
    0.6268    0.0591   -0.7012   -0.1356   -0.7989
    0.0751    1.7971    1.2460   -1.3493   -0.7652
    0.3516    0.2641   -0.6390   -1.2704    0.8617
   -0.6965    0.8717    0.5774    0.9846   -0.0562
```

```
A = floor(T)

A =
     1     1    -2    -1    -1
     0     0    -1    -1    -1
     0     1     1    -2    -1
     0     0    -1    -2     0
    -1     0     0     0    -1
```

```
B = ceil(T)

B =
     2     2    -1     0     0
     1     1     0     0     0
     1     2     2    -1     0
     1     1     0    -1     1
     0     1     1     1     0
```

```
C = fix(T)

C =
     1     1    -1     0     0
     0     0     0     0     0
     0     1     1    -1     0
     0     0     0    -1     0
     0     0     0     0     0
```

```
D = round(T)

D =
     1     2    -1     0     0
     1     0    -1     0    -1
     0     2     1    -1    -1
     0     0    -1    -1     1
    -1     1     1     1     0
```

It is worth mentioning that the following identities

floor(x) = fix(x)   for **x ρ 0**

and

ceil(x) = fix(x)   for **x β 0**

hold true.

In the following m-file functions **floor** and **ceil** are used to obtain a certain representation of a nonnegative real number

```
function [m, r] = rep4(x)

% Given a nonnegative number x, function rep4 computes an integer m
% and a real number r, where 0.25 <= r < 1, such that x = (4^m)*r.

if x == 0
   m = 0;
   r = 0;
   return
end
u = log10(x)/log10(4);
if u < 0
   m = floor(u)
else
   m = ceil(u);
end
r = x/4^m;
```

Command **return** causes a return to the invoking function or to the keyboard. Function **log10** is the decimal logarithm.

```
[m, r] = rep4(pi)

m =
     1
r =
    0.7854
```

We check this result

```
format long

(4^m)*r

ans =
   3.14159265358979


format short
```

2.8 MATLAB graphics

MATLAB has several high-level graphical routines. They allow a user to create various graphical objects including two- and three-dimensional graphs, graphical user interfaces (GUIs), movies, to mention the most important ones. For the comprehensive presentation of the MATLAB graphics the interested reader is referred to [2].

Before we begin discussion of graphical tools that are available in MATLAB I recommend that you will run a couple of demos that come with MATLAB. In the **Command Window** click on **Help** and next select **Examples and Demos**. Chose**Visualization**, and next select **2-D Plots**. You will be presented with several buttons. Select **Line** and examine the m-file below the graph. It should give you some idea about computer code needed for creating a simple graph. It is recommended that you examine carefully contents of all m-files that generate the graphs in this demo.

2.8.1 2-D graphics

Basic function used to create 2-D graphs is the **plot** function. This function takes a variable number of input arguments. For the full definition of this function type **help plot** in the **Command Window**.

In this example the graph of the rational function $f(x) = \frac{x}{1+x^2}$, -2 β x β 2, will be plotted using a variable number of points on the graph of f(x)

```
% Script file graph1.

% Graph of the rational function y = x/(1+x^2).

for n=1:2:5
   n10 = 10*n;
   x = linspace(-2,2,n10);
   y = x./(1+x.^2);
   plot(x,y,'r')
   title(sprintf('Graph %g.  Plot based upon n = %g points.' ...
   , (n+1)/2, n10))
   axis([-2,2,-.8,.8])
   xlabel('x')
   ylabel('y')
   grid
   pause(3)
end
```

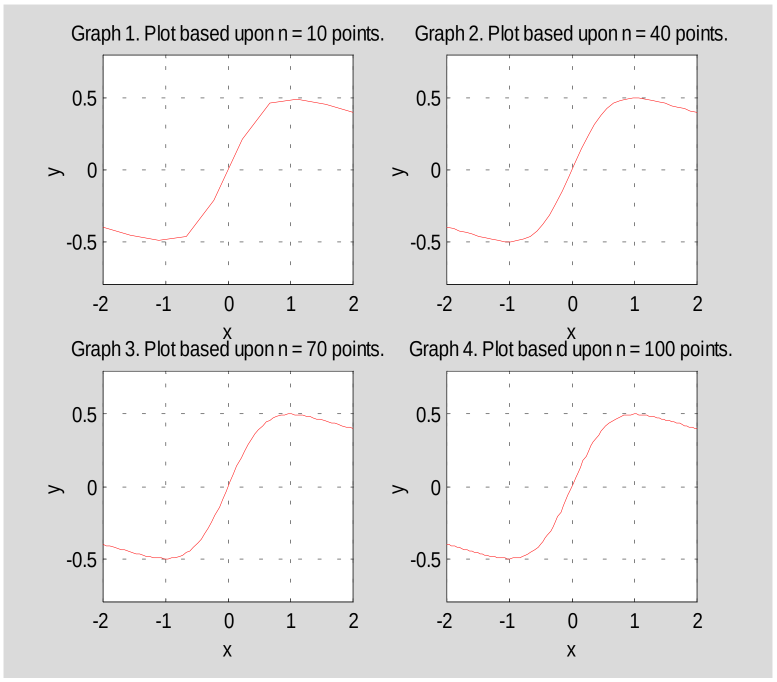
Let us analyze contents of this file. The loop **for** is executed three times. Therefore, three graphs of the same function will be displayed in the **Figure Window**. A MATLAB function **linspace(a, b, n)** generates a one-dimensional array of **n** evenly spaced numbers in the interval **[a b]**. The y-ordinates of the points to be plotted are stored in the array **y**. Command **plot** is called with three arguments: two arrays holding the x- and the y-coordinates and the string **'r'**, which describes the color (red) to be used to paint a plotted curve. You should notice a difference between three graphs created by this file. There is a significant difference between smoothness of graphs 1 and 3. Based on your visual observation you should be able to reach the following conclusion: "more points you supply the smoother graph is generated by the function **plot**". Function **title** adds a descriptive information to the graphs generated by this m-file and is followed by the command **sprintf**. Note that **sprintf** takes here three arguments: the string and names of two variables printed in the title of each graph. To specify format of printed numbers we use here the construction **%g**, which is recommended for printing integers. The command **axis** tells MATLAB what the dimensions of the box holding the plot are. To add more information to the graphs created here, we label the x- and the y-axes using commands **xlabel** and the **ylabel**, respectively. Each of these commands takes a string as the input argument. Function **grid** adds the grid lines to the graph. The last command used before the closing end is the **pause** command. The command **pause(n)** holds on the current graph for **n** seconds before continuing, where **n** can also be a fraction. If **pause** is called without the input argument, then the computer waits to user response. For instance, pressing the **Enter** key will resume execution of a program.

Function **subplot** is used to plot of several graphs in the same **Figure Window**. Here is a slight modification of the m-file **graph1**

```
% Script file graph2.

% Several plots of the rational function y = x/(1+x^2)
% in the same window.

k = 0;
for n=1:3:10
   n10 = 10*n;
   x = linspace(-2,2,n10);
   y = x./(1+x.^2);
   k = k+1;
   subplot(2,2,k)
   plot(x,y,'r')
   title(sprintf('Graph %g. Plot based upon n = %g points.' ...
      , k, n10))
   xlabel('x')
   ylabel('y')
   axis([-2,2,-.8,.8])
   grid
   pause(3);
end
```

The command **subplot** is called here with three arguments. The first two tell MATLAB that a 2-by-2 array consisting of four plots will be created. The third parameter is the running index telling MATLAB which subplot is currently generated.

```
graph2
```



Using command **plot** you can display several curves in the same **Figure Window**.

We will plot two ellipses

$$\frac{(x-3)^2}{36}+\frac{(y+2)^2}{81}=1 \quad \text{and} \quad \frac{(x-7)^2}{4}+\frac{(y-8)^2}{36}=1$$

using command **plot**

```
% Script file graph3.

% Graphs of two ellipses

%                 x(t) = 3 + 6cos(t), y(t) = -2 + 9sin(t)

% and

%                 x(t) = 7 + 2cos(t), y(t) = 8 + 6sin(t).


t = 0:pi/100:2*pi;
x1 = 3 + 6*cos(t);
y1 = -2 + 9*sin(t);
x2 = 7 + 2*cos(t);
y2 = 8 + 6*sin(t);
h1 = plot(x1,y1,'r',x2,y2,'b');
set(h1,'LineWidth',1.25)
axis('square')
xlabel('x')
```

```
h = get(gca,'xlabel');
set(h,'FontSize',12)
set(gca,'XTick',-4:10)
ylabel('y')
h = get(gca,'ylabel');
set(h,'FontSize',12)
set(gca,'YTick',-12:2:14)
title('Graphs of (x-3)^2/36+(y+2)^2/81 = 1 and (x-7)^2/4+(y-8)^2/36 =
1.')
h = get(gca,'Title');
set(h,'FontSize',12)
grid
```

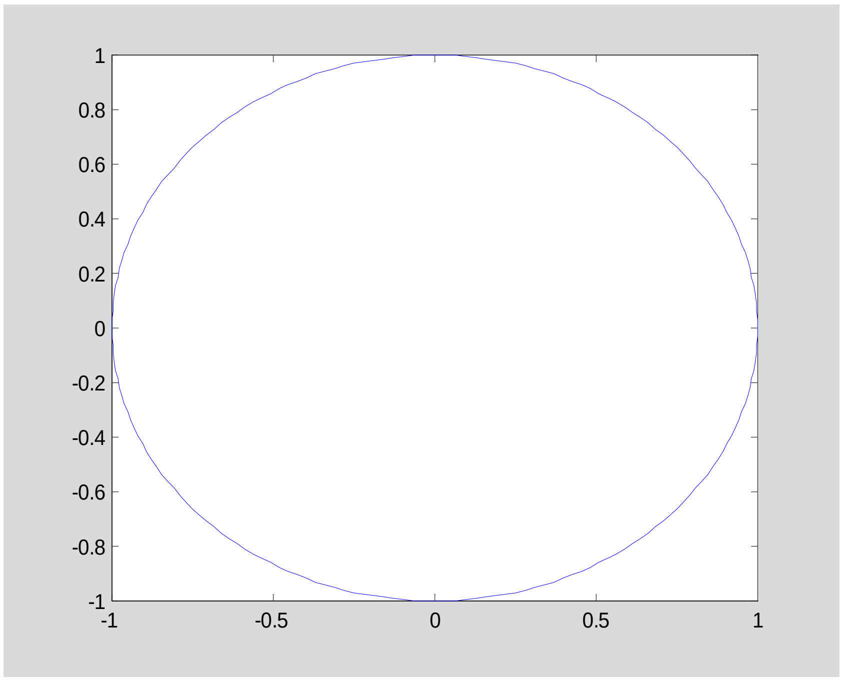
In this file we use several new MATLAB commands. They are used here to enhance the readability of the graph. Let us now analyze the computer code contained in the m-file **graph3**. First of all, the equations of ellipses in rectangular coordinates are transformed to parametric equations. This is a convenient way to plot graphs of equations in the implicit form. The points to be plotted, and smoothed by function **plot**, are defined in the first five lines of the file. I do not count here the comment lines and the blank lines. You can plot both curves using a single **plot** command. Moreover, you can select colors of the curves. They are specified as strings (see line 6). MATLAB has several colors you can use to plot graphs:

y	**yellow**
m	**magenta**
c	**cyan**
r	**red**
g	**green**
b	**blue**
w	**white**
k	**black**

Note that the command in line 6 begins with **h1 = plot**… Variable **h1** holds an information about the graph you generate and is called the *handle graphics*. Command **set** used in the next line allows a user to manipulate a plot. Note that this command takes as the input parameter the variable **h1**. We change thickness of the plotted curves from the default value to a width of our choice, namely 1.25. In the next line we use command **axis** to customize plot. We chose option **'square'** to force axes to have square dimensions. Other available options are: **'equal'**, **'normal'**, **'ij'**, **'xy'**, and **'tight'**. To learn more about these options use MATLAB's help.

If function **axis** is not used, then the circular curves are not necessarily circular. To justify this let us plot a graph of the unit circle of radius 1 with center at the origin

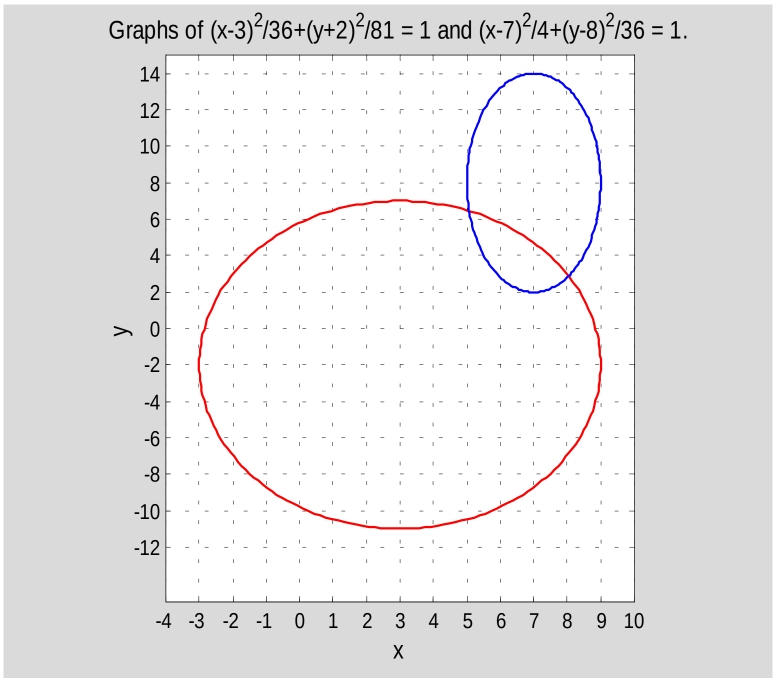
```
t = 0:pi/100:2*pi;
x = cos(t);
y = sin(t);
plot(x,y)
```



Another important MATLAB function used in the file under discussion is named **get** (see line 10). It takes as the first input parameter a variable named **gca** = *get current axis*. It should be obvious to you, that the axis targeted by this function is the x-axis. Variable **h = get(gca, …)** is the *graphics handle* of this axis. With the information stored in variable **h**, we change the font size associated with the x-axis using the **'FontSize'** string followed by a size of the font we wish to use. Invoking function **set** in line 12, we will change the tick marks along the x-axis using the **'XTick'** string followed by the array describing distribution of marks. You can comment out temporarily line 12 by adding the percent sign **%** before the word **set** to see the difference between the default tick marks and the marks generated by the command in line 12. When you are done delete the percent sign you typed in line 12 and click on **Save** from the **File** menu in the **MATLAB Editor/Debugger**. Finally, you can also make changes in the title of your plot. For instance, you can choose the font size used in the title. This is accomplished here by using function **set**. It should be obvious from the short discussion presented here that two MATLAB functions **get** and **set** are of great importance in manipulating graphs.

Graphs of the ellipses in question are shown on the next page

`graph3`



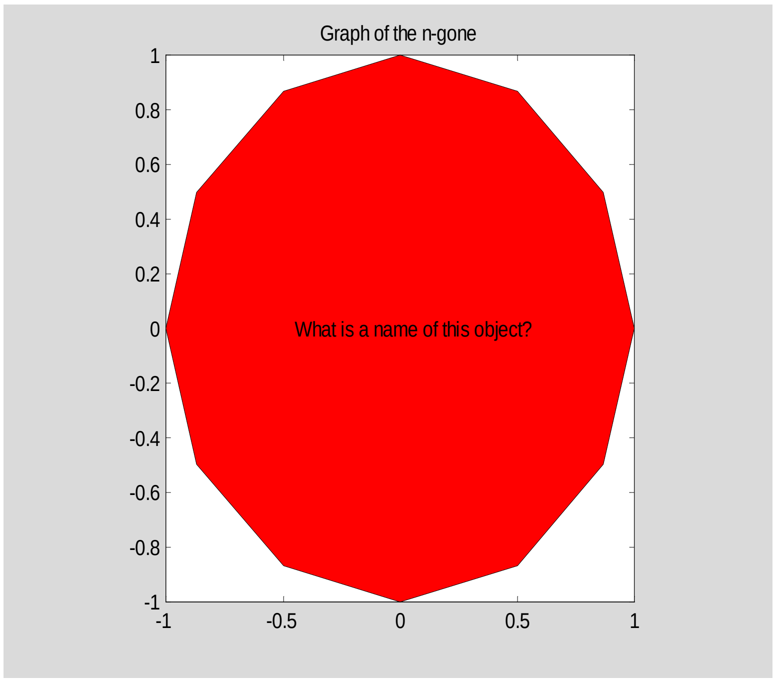
MATLAB has several functions designed for plotting specialized 2-D graphs. A partial list of these functions is included here **fill**, **polar**, **bar**, **barh**, **pie**, **hist**, **compass**, **errorbar**, **stem**, and **feather**.

In this example function **fill** is used to create a well-known object

```
n = -6:6;
x = sin(n*pi/6);
y = cos(n*pi/6);
fill(x, y, 'r')
axis('square')
title('Graph of the n-gone')
text(-0.45,0,'What is a name of this object?')
```

Function in question takes three input parameters - two arrays, named here **x** and **y**. They hold the x- and y-coordinates of vertices of the polygon to be filled. Third parameter is the user-selected color to be used to paint the object. A new command that appears in this short code is the **text** command. It is used to annotate a text. First two input parameters specify text location. Third input parameter is a text, which will be added to the plot.

Graph of the filled object that is generated by this code is displayed below



2.8.2 3-D graphics

MATLAB has several built-in functions for plotting three-dimensional objects. In this subsection we will deal mostly with functions used to plot curves in space (**plot3**), mesh surfaces (**mesh**), surfaces (**surf**) and contour plots (**contour**). Also, two functions for plotting special surfaces, **sphere** and **cylinder** will be discussed briefly. I recommend that any time you need help with the 3-D graphics you should type **help graph3d** in the **Command Window** to learn more about various functions that are available for plotting three-dimensional objects.

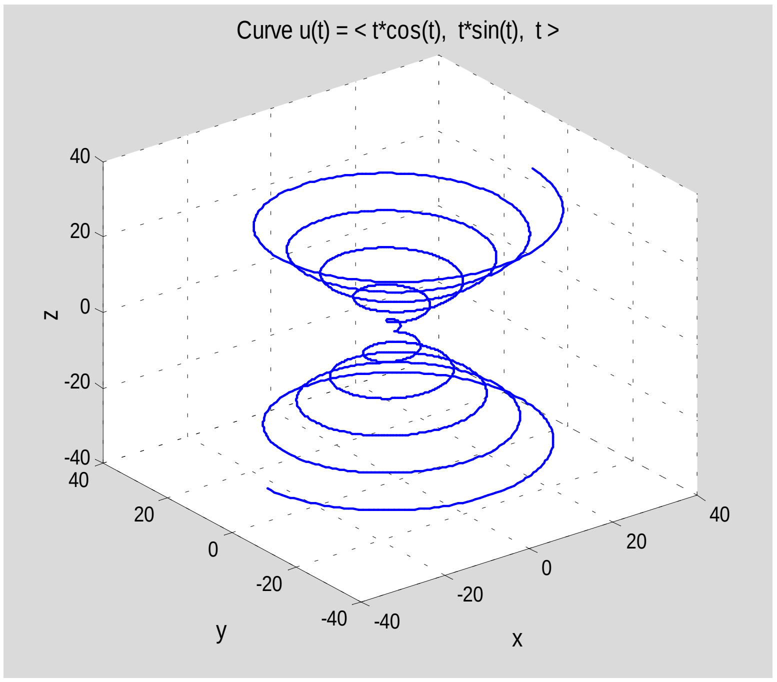
Let **r(t) = < t cos(t), t sin(t), t** >, **-10π β t β 10π**, be the space curve. We plot its graph over the indicated interval using function **plot3**

```
% Script file graph4.

% Curve r(t) = < t*cos(t), t*sin(t), t >.

t = -10*pi:pi/100:10*pi;
x = t.*cos(t);
y = t.*sin(t);
h = plot3(x,y,t);
set(h,'LineWidth',1.25)
title('Curve u(t) = < t*cos(t),  t*sin(t),  t >')
h = get(gca,'Title');
set(h,'FontSize',12)
xlabel('x')
h = get(gca,'xlabel');
set(h,'FontSize',12)
ylabel('y')
h = get(gca,'ylabel');
```

```
set(h,'FontSize',12)
zlabel('z')
h = get(gca,'zlabel');
set(h,'FontSize',12)
grid
```

Function **plot3** is used in line 4. It takes three input parameters – arrays holding coordinates of points on the curve to be plotted. Another new command in this code is the **zlabel** command (see line 4 from the bottom). Its meaning is self-explanatory.

```
graph4
```



Function **mesh** is intended for plotting graphs of the 3-D mesh surfaces. Before we begin to work with this function, another function **meshgrid** should be introduced. This function generates two two-dimensional arrays for 3-D plots. Suppose that one wants to plot a mesh surface over the grid that is defined as the Cartesian product of two sets

```
x = [0 1 2];
y = [10 12 14];
```

The **meshgrid** command applied to the arrays **x** and **y** creates two matrices

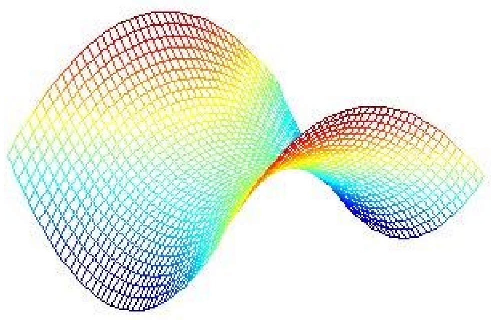
```
[xi, yi] = meshgrid(x,y)
```

```
xi =
     0     1     2
     0     1     2
     0     1     2
yi =
    10    10    10
    12    12    12
    14    14    14
```

Note that the matrix **xi** contains replicated rows of the array **x** while **yi** contains replicated columns of **y**. The z-values of a function to be plotted are computed from arrays **xi** and **yi**.

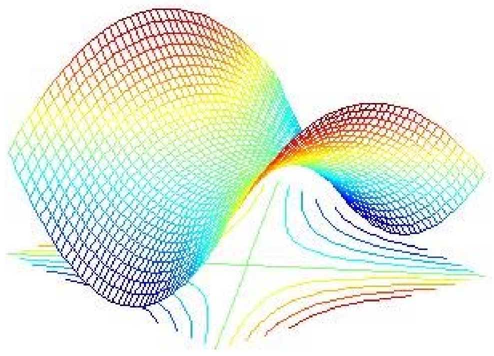
In this example we will plot the hyperbolic paraboloid $z = y^2 – x^2$ over the square **–1 β x β 1**, **-1 β y β 1**

```
x = -1:0.05:1;
y = x;
[xi, yi] = meshgrid(x,y);
zi = yi.^2 – xi.^2;
mesh(xi, yi, zi)
axis off
```



To plot the graph of the mesh surface together with the contour plot beneath the plotted surface use function **meshc**

```
meshc(xi, yi, zi)
axis off
```



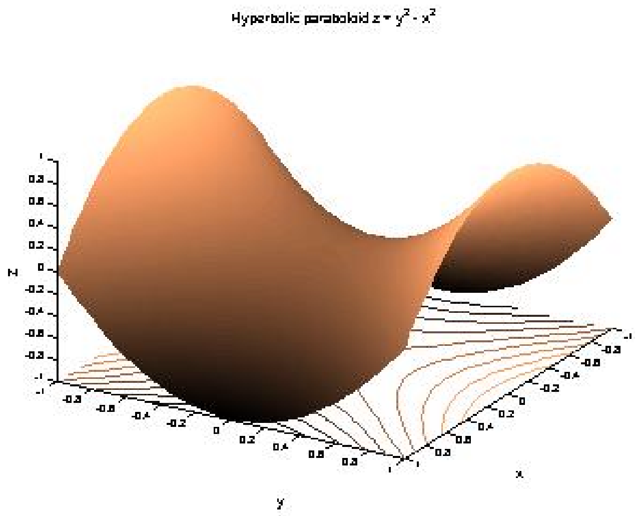
Function **surf** is used to visualize data as a shaded surface.

Computer code in the m-file **graph5** should help you to learn some finer points of the 3-D graphics in MATLAB

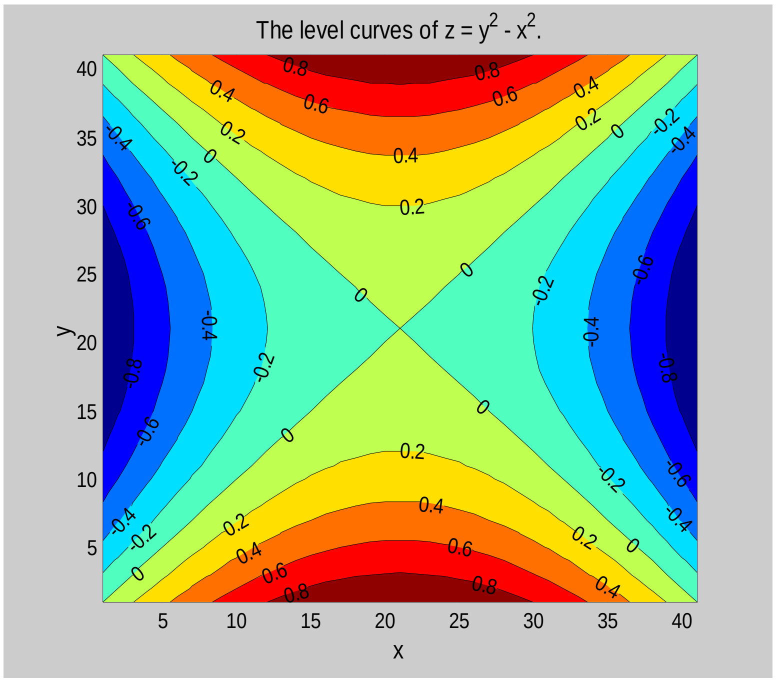
```
% Script file graph5.

% Surface plot of the hyperbolic paraboloid z = y^2 - x^2
% and its level curves.

x = -1:.05:1;
y = x;
[xi,yi] = meshgrid(x,y);
zi = yi.^2 - xi.^2;
surfc(xi,yi,zi)
colormap copper
shading interp
view([25,15,20])
grid off
title('Hyperbolic paraboloid z = y^2 – x^2')
h = get(gca,'Title');
set(h,'FontSize',12)
xlabel('x')
h = get(gca,'xlabel');
set(h,'FontSize',12)
ylabel('y')
h = get(gca,'ylabel');
set(h,'FontSize',12)
zlabel('z')
h = get(gca,'zlabel');
set(h,'FontSize',12)
pause(5)
figure
contourf(zi), hold on, shading flat
[c,h] = contour(zi,'k-'); clabel(c,h)
title('The level curves of z = y^2 - x^2.')
h = get(gca,'Title');
set(h,'FontSize',12)
```

```
xlabel('x')
h = get(gca,'xlabel');
set(h,'FontSize',12)
ylabel('y')
h = get(gca,'ylabel');
set(h,'FontSize',12)
```

```
  graph5
```



A second graph is shown on the next page.



There are several new commands used in this file. On line 5 (again, I do not count the blank lines and the comment lines) a command **surfc** is used. It plots a surface together with the level lines beneath. Unlike the command **surfc** the command **surf** plots a surface only without the level curves. Command **colormap** is used in line 6 to paint the surface using a user-supplied colors. If the command **colormap** is not added, MATLAB uses default colors. Here is a list of color maps that are available in MATLAB

hsv	- hue-saturation-value color map
hot	- black-red-yellow-white color map
gray	- linear gray-scale color map
bone	- gray-scale with tinge of blue color map
copper	- linear copper-tone color map
pink	- pastel shades of pink color map
white	- all white color map
flag	- alternating red, white, blue, and black color map
lines	- color map with the line colors
colorcube	- enhanced color-cube color map
vga	- windows colormap for 16 colors
jet	- variant of HSV
prism	- prism color map
cool	- shades of cyan and magenta color map
autumn	- shades of red and yellow color map
spring	- shades of magenta and yellow color map
winter	- shades of blue and green color map
summer	- shades of green and yellow color map

Command **shading** (see line 7) controls the color shading used to paint the surface. Command in question takes one argument. The following

shading flat sets the shading of the current graph to flat
shading interp sets the shading to interpolated
shading faceted sets the shading to faceted, which is the default.

are the shading options that are available in MATLAB.

Command **view** (see line 8) is the 3-D graph viewpoint specification. It takes a three-dimensional vector, which sets the view angle in Cartesian coordinates.

We will now focus attention on commands on lines 23 through 25. Command **figure** prompts MATLAB to create a new **Figure Window** in which the level lines will be plotted. In order to enhance the graph, we use command **contourf** instead of **contour**. The former plots filled contour lines while the latter doesn't. On the same line we use command **hold on** to hold the current plot and all axis properties so that subsequent graphing commands add to the existing graph. First command on line 25 returns matrix **c** and graphics handle **h** that are used as the input parameters for the function **clabel**, which adds height labels to the current contour plot.

Due to the space limitation we cannot address here other issues that are of interest for programmers dealing with the 3-D graphics in MATLAB. To learn more on this subject the interested reader is referred to [1-3] and [5].

2.8.3 Animation

In addition to static graphs discussed so far one can put a sequence of graphs in motion. In other words, you can make a movie using MATLAB graphics tools. To learn how to create a movie, let us analyze the m-file **firstmovie**

```
 % Script file firstmovie.

% Graphs of y = sin(kx) over the interval [0, pi],
% where k = 1, 2, 3, 4, 5.

m = moviein(5);
x = 0:pi/100:pi;
for i=1:5
   h1_line = plot(x,sin(i*x));
   set(h1_line,'LineWidth',1.5,'Color','m')
   grid
   title('Sine functions sin(kx), k = 1, 2, 3, 4, 5')
   h = get(gca,'Title');
   set(h,'FontSize',12)
   xlabel('x')
   k = num2str(i);
   if i > 1
      s = strcat('sin(',k,'x)');
   else
      s = 'sin(x)';
   end
   ylabel(s)
   h = get(gca,'ylabel');
   set(h,'FontSize',12)
   m(:,i) = getframe;
   pause(2)
```

```
end
movie(m)
```

I suggest that you will play this movie first. To this aim type **firstmovie** in the **Command Window** and press the **Enter** or **Return** key. You should notice that five frames are displayed and at the end of the "show" frames are played again at a different speed.

There are very few new commands one has to learn in order to animate graphics in MATLAB. We will use the m-file **firstmovie** as a starting point to our discussion. Command **moviein**, on line 1, with an integral parameter, tells MATLAB that a movie consisting of five frames is created in the body of this file. Consecutive frames are generated inside the loop **for**. Almost all of the commands used there should be familiar to you. The only new one inside the loop is **getframe** command. Each frame of the movie is stored in the column of the matrix **m**. With this remark a role of this command should be clear. The last command in this file is **movie(m)**. This tells MATLAB to play the movie just created and saved in columns of the matrix **m**.

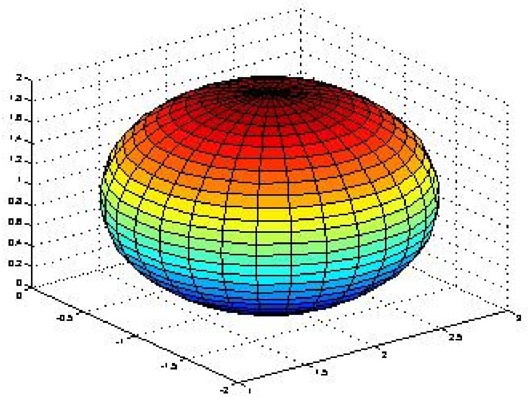
Warning. File **firstmovie** cannot be used with the **Student Edition of MATLAB, version 4.2**. This is due to the matrix size limitation in this edition of MATLAB. Future release of the **Student Edition of MATLAB, version 5.3** will allow large size matrices. According to MathWorks, Inc., the makers of MATLAB, this product will be released in September 1999.

2.8.4 Commands sphere and cylinder

MATLAB has some functions for generating special surfaces. We will be concerned mostly with two functions- **sphere** and **cylinder**.

The command **sphere(n)** generates a unit sphere with center at the origin using **$(n+1)^2$** points. If function **sphere** is called without the input parameter, MATLAB uses the default value n = 20. You can translate the center of the sphere easily. In the following example we will plot graph of the unit sphere with center at (2, -1, 1)

```
[x,y,z] = sphere(30);
surf(x+2, y-1, z+1)
```

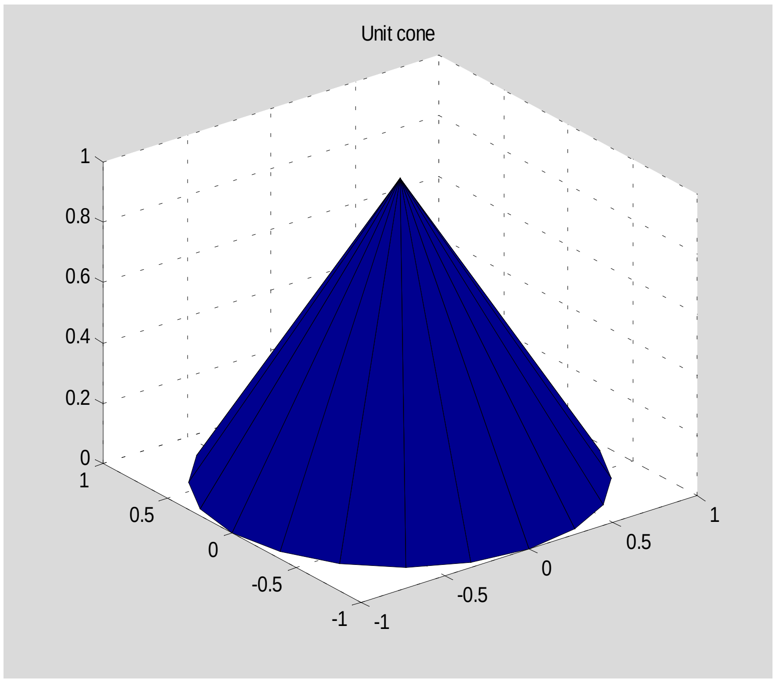


Function **sphere** together with function **surf** or **mesh** can be used to plot graphs of spheres of arbitrary radii. Also, they can be used to plot graphs of ellipsoids. See Problems 25 and 26.

Function **cylinder** is used for plotting a surface of revolution. It takes two (optional) input parameters. In the following command **cylinder(r, n)** parameter **r** stands for the vector that defines the radius of cylinder along the z-axis and **n** specifies a number of points used to define circumference of the cylinder. Default values of these parameters are **r = [1 1]** and **n = 20**. A generated cylinder has a unit height.

The following command

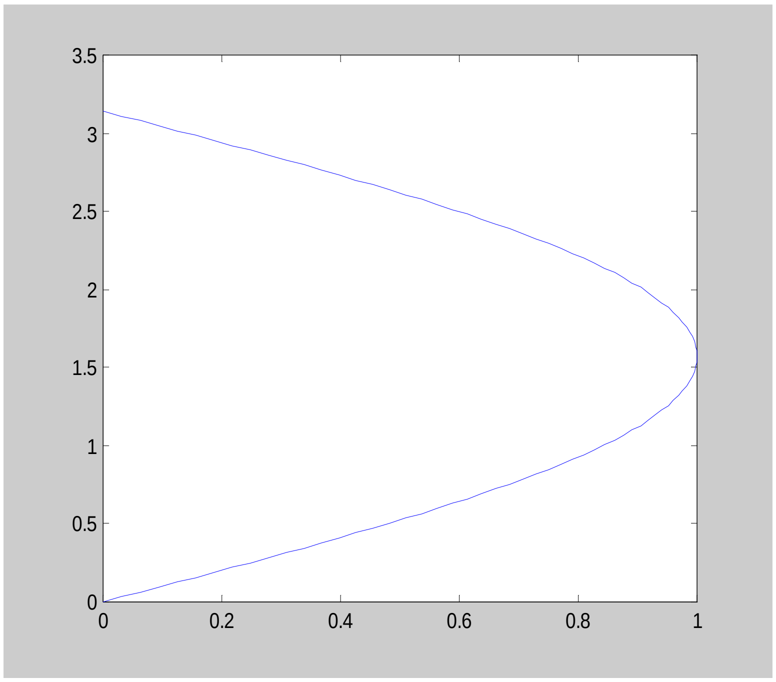
```
cylinder([1 0])
title('Unit cone')
```



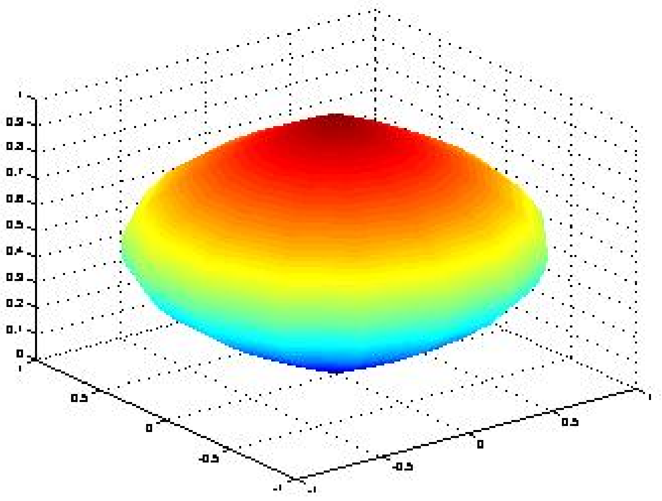
plots a cone with the base radius equal to one and the unit height.

In this example we will plot a graph of the surface of revolution obtained by rotating the curve **r(t) = < sin(t), t >**, **0 β t β π** about the y-axis. Graphs of the generating curve and the surface of revolution are created using a few lines of the computer code

```
t = 0:pi/100:pi;
r = sin(t);
plot(r,t)
```



```
cylinder(r,15)
shading interp
```



2.8.5 Printing MATLAB graphics

In this section we deal with printing MATLAB graphics. To send a current graph to the printer click on **File** and next select **Print** from the pull down menu. Once this menu is open you may wish to preview a graph to be printed be selecting the option **PrintPreview…** first. You can also send your graph to the printer using the **print** command as shown below

```
x = 0:0.01:1;
plot(x, x.^2)
print
```

You can print your graphics to an m- file using built-in device drivers. A fairly incomplete list of these drivers is included here:

-depsc	Level 1 color Encapsulated PostScript
-deps2	Level 2 black and white Encapsulated PostScript
-depsc2	Level 2 color Encapsulated PostScript

For a complete list of available device drivers see [5], Chapter 7, pp. 8-9.

Suppose that one wants to print a current graph to the m-file Figure1 using level 2 color Encapsulated PostScript. This can be accomplished by executing the following command

```
print –depsc2 Figure1
```

You can put this command either inside your m-file or execute it from within the **Command Window**.

References

[1] D. Hanselman and B. Littlefield, Mastering MATLAB 5. A Comprehensive Tutorial and Reference, Prentice Hall, Upper Saddle River, NJ, 1998.

[2] P. Marchand, Graphics and GUIs with MATLAB, Second edition, CRC Press, Boca Raton, 1999.

[3] K. Sigmon, MATLAB Primer, Fifth edition, CRC Press, Boca Raton, 1998.

[4] Using MATLAB, Version 5, The MathWorks, Inc., 1996.

[5] Using MATLAB Graphics, Version 5, The MathWorks, Inc., 1996.

Problems

In **Problems 1- 4** you cannot use loops **for** or **while**.

1. Write MATLAB function **sigma = ascsum(x)** that takes a one-dimensional array **x** of real numbers and computes their sum **sigma** in the ascending order of magnitudes.
   Hint: You may wish to use MATLAB functions **sort**, **sum**, and **abs**.

2. In this exercise you are to write MATLAB function **d = dsc(c)** that takes a one-dimensional array of numbers **c** and returns an array **d** consisting of all numbers in the array **c** with all neighboring duplicated numbers being removed. For instance, if **c = [1 2 2 2 3 1]**, then **d = [1 2 3 1]**.

3. Write MATLAB function **p = fact(n)** that takes a nonnegative integer **n** and returns value of the factorial function n! = 1*2* … *n. Add an error message to your code that will be executed when the input parameter is a negative number.

4. Write MATLAB **function [in, fr] = infr(x)** that takes an array **x** of real numbers and returns arrays **in** and **fr** holding the integral and fractional parts, respectively, of all numbers in the array **x**.

5. Given an array **b** and a positive integer **m** create an array **d** whose entries are those in the array **b** each replicated m-times. Write MATLAB **function d = repel(b, m)** that generates array **d** as described in this problem.

6. In this exercise you are to write MATLAB **function d = rep(b, m)** that has more functionality than the function **repel** of Problem 5. It takes an array of numbers **b** and the array **m** of positive integers and returns an array **d** whose each entry is taken from the array **b** and is duplicated according to the corresponding value in the array **m**. For instance, if **b = [1 2]** and **m = [2 3]**, then **d = [1 1 2 2 2]**.

7. A *checkerboard* matrix is a square block diagonal matrix, i.e., the only nonzero entries are in the square blocks along the main diagonal. In this exercise you are to write MATLAB **function A = mysparse(n)** that takes an odd number **n** and returns a checkerboard matrix as shown below

```
   A = mysparse(3)

A =
     1     0     0
     0     1     2
     0     3     4


   A = mysparse(5)
```

```
A =
     1     0     0     0     0
     0     1     2     0     0
     0     3     4     0     0
     0     0     0     2     3
     0     0     0     4     5
```

```
   A = mysparse(7)
```

```
A =
     1     0     0     0     0     0     0
     0     1     2     0     0     0     0
     0     3     4     0     0     0     0
     0     0     0     2     3     0     0
     0     0     0     4     5     0     0
     0     0     0     0     0     3     4
     0     0     0     0     0     5     6
```

First block in the upper-left corner is the 1-by-1 matrix while the remaining blocks are all 2-by-2.

8. The Legendre polynomials **$P_n(x)$**, **n = 0, 1, …** are defined recursively as follows

$$nP_n(x) = (2n-1)xP_{n-1} - (n-1)P_{n-2}(x),\quad n = 2, 3, \dots,\ P_0(x) = 1,\ P_1(x) = x.$$

Write MATLAB **function P = LegendP(n)** that takes an integer **n** – the degree of **$P_n(x)$** and returns its coefficient stored in the descending order of powers.

9. In this exercise you are to implement Euclid's Algorithm for computing the *greatest common divisor* (**gcd**) of two integer numbers **a** and **b**:

$$\text{gcd}(a, 0) = a,\quad \text{gcd}(a, b) = \text{gcd}(b, \text{rem}(a, b)).$$

Here **rem(a, b)** stands for the remainder in dividing **a** by **b**. MATLAB has function **rem**. Write MATLAB **function gcd = mygcd(a,b)** that implements Euclid's Algorithm.

10. The Pascale triangle holds coefficients in the series exapansion of **$(1 + x)^n$**, where **n = 0, 1, 2, …** . The top of this triangle, for **n = 0, 1, 2**, is shown here

```
      1
    1   1
  1   2   1
```

Write MATLAB **function t = pasctri(n)** that generates the Pascal triangle **t** up to the level **n**. **Remark**. Two-dimensional arrays in MATLAB must have the same number of columns in each row. In order to aviod error messages you have to add a certain number of zero entries to the right of last nonzero entry in each row of **t** but one. This

```
t = pasctri(2)
```

```
t =
     1     0     0
     1     1     0
     1     2     1
```

is an example of the array **t** for **n = 2**.

11. This is a continuation of Problem 10. Write MATLAB **function t = binexp(n)** that computes an array **t** with row **k+1** holding coefficients in the series expansion of **(1-x)^k**, **k = 0, 1, ... , n**, in the ascending order of powers. You may wish to make a call from within your function to the function **pasctri** of Problem 10. Your output sholud look like this (case **n = 3**)

```
t = binexp(3)

t =
   1     0     0     0
   1    -1     0     0
   1    -2     1     0
   1    -3     3    -1
```

12. MATLAB come with the built-in function **mean** for computing the *unweighted arithmetic mean* of real numbers. Let x = {x_1, x_2, … , x_n} be an array of n real numbers. Then

$$mean(x) = \frac{1}{n}\sum_{k=1}^{n} x_n$$

In some problems that arise in mathematical statistics one has to compute the *weighted arithmetic mean* of numbers in the array x. The latter, abbreviated here as **wam**, is defined as follows

$$wam(x, w) = \frac{\sum_{k=1}^{n} w_k x_k}{\sum_{k=1}^{n} w_k}$$

Here w = {w_1, w_2, … , w_n} is the array of weights associated with variables x. The weights are all nonnegative with $w_1 + w_2 + \ldots + w_n > 0$.
In this exercise you are to write MATLAB **function y = wam(x, w)** that takes the arrays of variables and weights and returns the weighted arithmetic mean as defined above. Add three error messages to terminate prematurely execution of this file in the case when:

- arrays **x** and **w** are of different lengths
- at least one number in the array w is negative
- sum of all weights is equal to zero.

13. Let w = {w_1, w_2, … , w_n} be an array of positive numbers. The *weighted geometric mean*, abbreviated as **wgm**, of the nonnegative variables x = {x_1, x_2, … , x_n} is defined as follows

$$wgm(x, w) = x_1^{\;w_1} x_2^{\;w_2} \ldots x_n^{\;w_n}$$

Here we assume that the weights **w** sum up to one.
Write MATLAB **function y = wgm(x, w)** that takes arrays **x** and **w** and returns the weighted geometric mean **y** of **x** with weights stored in the array **w**. Add three error messages to terminate prematurely execution of this file in the case when:

- arrays **x** and **w** are of different lengths
- at least one variable in the array **x** is negative
- at least one weight in the array **w** is less than or equal to zero

Also, normalize the weights **w**, if necessary, so that they will sum up to one.

14. Write MATLAB **function [nonz, mns] = matstat(A)** that takes as the input argument a real matrix A and returns all nonzero entries of A in the column vector **nonz**. Second output parameter **mns** holds values of the unweighted arithmetic means of all columns of **A**.

15. Solving triangles requires a bit of knowledge of trigonometry. In this exercise you are to write MATLAB **function [a, B, C] = sas(b, A, c)** that is intended for solving triangles given two sides **b** and **c** and the angle **A** between these sides. Your function should determine remaining two angels and the third side of the triangle to be solved. All angles should be expressed in the degree measure.

16. Write MATLAB **function [A, B, C] = sss(a, b, c)** that takes three positive numbers **a**, **b**, and **c**. If they are sides of a triangle, then your function should return its angles **A**, **B**, and **C**, in the degree measure, otherwise an error message should be displayed to the screen.

17. In this exercise you are to write MATLAB **function dms(x)** that takes a nonnegative number **x** that represents an angle in the degree measure and converts it to the form **x deg. y min. z sec.**. Display a result to the screen using commands **disp** and **sprintf**.
Example:

```
dms(10.2345)
```

```
Angle = 10 deg. 14 min. 4 sec.
```

18. Complete elliptic integral of the first kind in the Legendre form K(k^2), $0 < k^2 < 1$,

$$K(k^2) = \int_0^{\pi/2} \frac{dt}{\sqrt{1 - k^2 \sin^2(t)}}$$

cannot be evaluated in terms of the elementary functions. The following algorithm, due to C. F. Gauss, generates a sequence of the arithmetic means {a_n} and a sequence of the geometric means {b_n}, where

$$a_0 = 1, \quad b_0 = \sqrt{1-k^2}$$

$$a_n = (a_{n-1} + b_{n-1})/2, \quad b_n = \sqrt{a_{n-1}b_{n-1}} \quad n = 1, 2, \ldots\ .$$

It is known that both sequences have a common limit g and that $a_n \geq b_n$, for all n. Moreover,

$$K(k^2) = \frac{\pi}{2g}$$

Write MATLAB **function K = compK(k2)** which implements this algorithm. The input parameter **k2** stands for k^2. Use the loop **while** to generate consecutive members of both sequences, but do not save all numbers generated in the course of computations. Continue execution of the **while** loop as long as $a_n - b_n \geq$ **eps**, where **eps** is the *machine epsilon*

```
eps

ans =
     2.2204e-016
```

Add more functionality to your code by allowing the input parameter **k2** to be an array. Test your m-file and compare your results with those included here

```
format long

compK([.1 .2 .3 .7 .8 .9])

ans =
   1.61244134872022
   1.65962359861053
   1.71388944817879
   2.07536313529247
   2.25720532682085
   2.57809211334794


format short
```

19. In this exercise you are to model one of the games in the Illinois State Lottery. Three numbers, with duplicates allowed, are selected randomly from the set {0,1,2,3,4,5,6,7,8,9} in the game Pick3 and four numbers are selected in the Pick4 game. Write MATLAB **function winnumbs = lotto(n)** that takes an integer n as its input parameter and returns an array **winnumbs** consisting of **n** numbers from the set of integers described in this problem. Use MATLAB function **rand** together with other functions to generate a set of winning numbers. Add an error message that is displayed to the screen when the input parameter is out of range.

20. Write MATLAB **function t = isodd(A)** that takes an array **A** of nonzero integers and returns **1** if all entries in the array **A** are odd numbers and **0** otherwise. You may wish to use MATLAB function **rem** in your file.

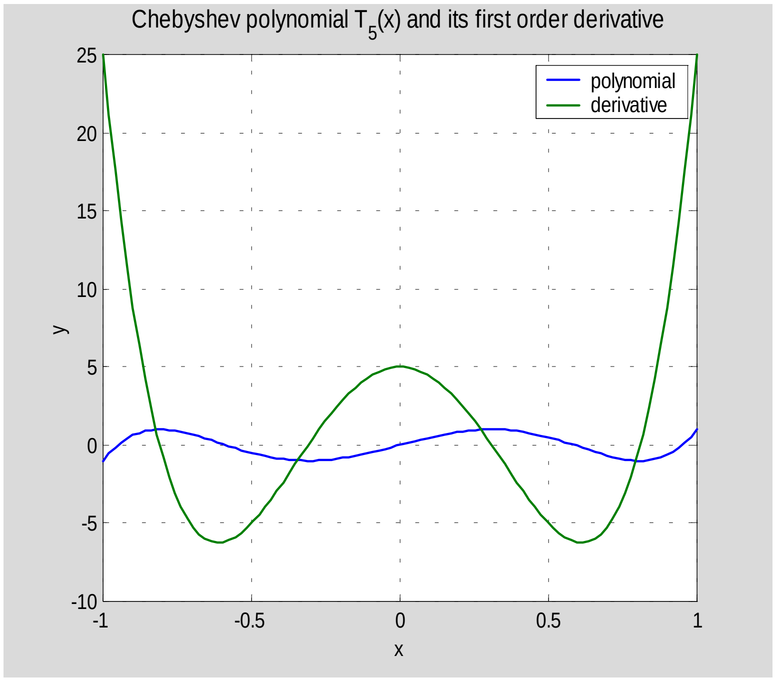
21. Given two one-dimensional arrays **a** and **b**, not necessarily of the same length. Write MATLAB **function c = interleave(a, b)** which takes arrays **a** and **b** and returns an array **c** obtained by interleaving entries in the input arrays. For instance, if **a = [1, 3, 5, 7]** and **b = [-2, –4]**, then **c = [1, –2, 3, –4, 5, 7]**. Your program should work for empty arrays too. You cannot use loops **for** or **while**.

22. Write a script file **Problem22** to plot, in the same window, graphs of two parabolas **$y = x^2$** and **$x = y^2$**, where **–1 β x β 1**. Label the axes, add a title to your graph and use command **grid**. To improve readability of the graphs plotted add a legend. MATLAB has a command **legend**. To learn more about this command type **help legend** in the **Command Window** and press **Enter** or **Return** key.

23. Write MATLAB **function eqtri(a, b)** that plots the graph of the equilateral triangle with two vertices at **(a,a)** and **(b,a)**. Third vertex lies above the line segment that connects points **(a, a)** and **(b, a)**. Use function **fill** to paint the triangle using a color of your choice.

24. In this exercise you are to plot graphs of the Chebyshev polynomial **T_n(x)** and its first order derivative over the interval **[-1, 1]**. Write MATLAB **function plotChT(n)** that takes as the input parameter the degree **n** of the Chebyshev polynomial. Use functions **ChebT** and **derp**, included in Tutorial 2, to compute coefficients of **T_n(x)** and **T'_n(x)**, respectively. Evaluate both, the polynomial and its first order derivative at **x = linspace(-1, 1)** using MATLAB function **polyval**. Add a meaningful title to your graph. In order to improve readability of your graph you may wish to add a descriptive legend. Here is a sample output

```
plotChT(5)
```

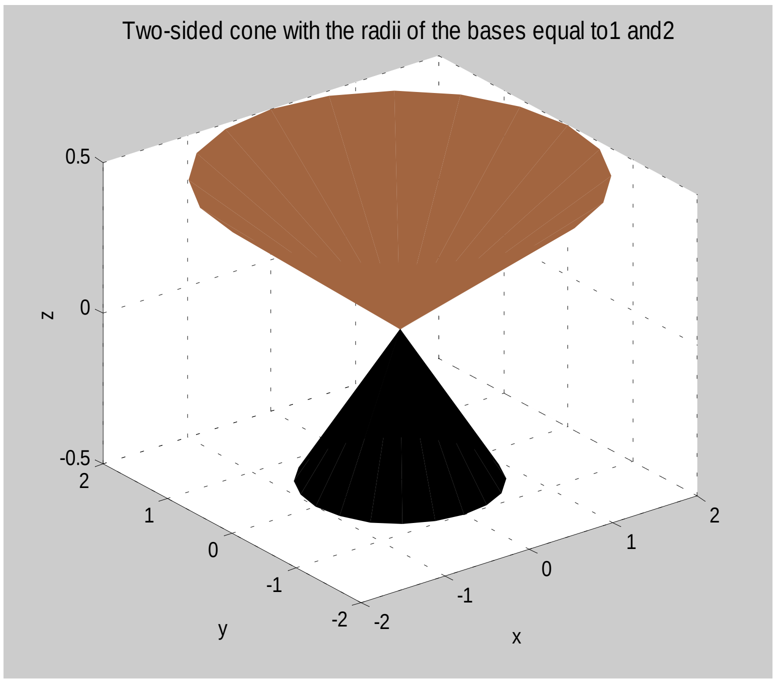


25. Use function **sphere** to plot the graph of a sphere of radius **r** with center at **(a, b, c)**. Use MATLAB function **axis** with an option **'equal'**. Add a title to your graph and save your computer code as the MATLAB function **sph(r, a, b, c)**.

26. Write MATLAB **function ellipsoid(x0, y0, z0, a, b, c)** that takes coordinates (**x0, y0, z0)** of the center of the ellipsoid with semiaxes **(a, b, c)** and plots its graph. Use MATLAB functions **sphere** and **surf**. Add a meaningful title to your graph and use function **axis('equal')**.

27. In this exercise you are to plot a graph of the two-sided cone, with vertex at the origin, and the-axis as the axis of symmetry. Write MATLAB **function cone(a, b)**, where the input parameters **a** and **b** stand for the radius of the lower and upper base, respectively. Use MATLAB functions **cylinder** and **surf** to plot a cone in question. Add a title to your graph and use function **shading** with an argument of your choice. A sample output is shown below

```
cone(1, 2)
```



28. The space curve r**(t) = < cos(t)sin(4t), sin(t)sin(4t), cos(4t)** >, **0 β t β 2π**, lies on the surface of the unit sphere $x^2 + y^2 + z^2 = 1$. Write MATLAB script file **curvsph** that plots both the curve and the sphere in the same window. Add a meaningful title to your graph. Use MATLAB functions **colormap** and **shading** with arguments of your choice. Add the **view([150 125 50])** command.

29. This problem requires that the professional version 5.x of MATLAB is installed. In this exercise you are to write the m-file **secondmovie** that crates five frames of the surface **z = sin(kx)cos(ky)**, where **0 β x, y β π** and **k = 1, 2, 3, 4, 5**. Make a movie consisting of the

frames you generated in your file. Use MATLAB functions **colormap** and **shading** with arguments of your choice. Add a title, which might look like this
Graphs of z = sin(kx)*cos(ky), 0 <= x, y <= π, k =1, 2, 3, 4, 5. Greek letters can be printed in the title of a graph using TeX convention, i.e., the following \pi is used to print the Greek letter π. Similarly, the string \alpha will be printed as α.