MATH5004 LAB 4
MATLAB COMPUTATION AND GRAPHICS

1. MATLAB GETTING START

To start the MATLAB program in PC, double-click the MATLAB icon. A command window will open with the MATLAB prompt >>. To solve a problem by MATLAB, you can work directly in the command window by entering the MATLAB commands. Alternatively, you may create a M-file, enter all MATLAB commands in the file via the editor window and save the file, and then run the M-file on the command window to solve the problem. The following 2 sections demonstrate how to solve a simple problem in these two modes.

1.1 Work on Command Window

The MATLAB command window is the main window where you type commands directly to perform certain tasks.

Example 1.1 Enter $A=\begin{pmatrix}1 & 2\\ 2 & 5\end{pmatrix}$ and $B=\begin{pmatrix}2 & 5\\ 3 & 1\end{pmatrix}$. Then calculate $C=A+B$.

In the command window at the MATLAB prompt >>, we enter

```
>> A=[1,2;2,5];
>> B=[2,5;3,1];
>> C=A+B
```

which results in

```
C =
    3   7
    5   6
```

1.2 Work on Editor Window

The MATLAB editor window is a text editor where you can load, edit, save and also run your MATLAB program (sequences of MATLAB commands).

To open an editor window, choose in the command window

File → New → M-file for creating a new M-file

File → Open → M-file for opening an existing M-file

eg. for the example 1.1, we create an M-file ex1_1.m and enter the following commands

```
A=[1,2;2,5];
B=[2,5;3,1];
C=A+B
```

and then save the file ex1_1.m. Once a M-file is created, it can be run to perform the specific task through the editor window or the command window.

(a) To run a M-file in the editor window, simply choose the Debug/Run in the command menu.
(b) To run a M-file in the command window, simply type in the file name in the command line and press enter

eg.

>> ex1_1.m

will run the ex1_1.m and yield the following result

C =

3 7

5 6

Notes: If the path of the file you want to run is not listed in the MATLAB search path, you need to add the path to the MATLAB-path list by clicking the menu 'File→Set→Path', clicking the 'Add-Folder' button, browsing/choosing the folder name and finally clicking the save button.

1.3 Use of MATLAB Help Window

The MATLAB Help Window gives you access to a great deal of useful information about the MATLAB language and MATLAB computing environment. It also has a number of example programs and tutorials. In addition, the "lookfor" command can help to find relevant functions for your job. The "help" command helps you to know how to use a particular command.

eg. >> lookfor repeat

or >> help for

2. MATLAB ARITHMETIC COMPUTATIONS

Arithmetic operations ($+, -, \times, \div$) are the most fundamental operations performed by computers. To be able to write programs for these operations, we need to know how to store data values in computers, how to implement computations, how to input data values to computers and how to print the computed results. Thus in this chapter, we describe

- Methods for storing data with MATLAB by using constants and variables
- Assignment statements for arithmetic computations
- Simple input/output statements for introducing data values into computers or printing results

.

2.1 Constants and Variables

Data are stored in MATLAB by the use of constants and variables.

(a) Constants

- In MATLAB, integer and real numbers are represented by one of the following 2 forms

  Decimal form: -12.3, 0.0, 5 etc.
  Scientific form: -5e8, 0.82e-11 etc (-5×10^{8} and 0.82×10^{11})

- Imaginary numbers use either i or j as a suffix : -3.14i, 3e5i etc. Thus, a complex number 3.5+2i can be written as 3.5+2i or 3.5+2j.
- All numbers are stored internally using long format by the IEEE floating point standard with a finite precision of about 16 significant digits and a finite value range $10^{-308} - 10^{+308}$.

(b) Variables

In MATLAB, a variable represents an $n \times m$ rectangular matrix (array). Each element of the array can store one data value and thus a MATLAB variable can store a group of data.

- MATLAB does not require declaration on the data type and dimension of the variable. When a new variable name is encountered, MATLAB automatically creates the variable with proper data type and an appropriate amount of storage. If the variable already exits, its contents are changed and new storage locations are allocated.

  eg. number_of_values = 60 creates a 1-by-1 matrix named number_of_values and stores the value 60 in the single element.

  x_values = [10,20,26] creates a 1-by-3 matrix with 3 elements respectively storing the value 10, 20 and 26.

- A MATLAB variable name begins with a letter and can optionally followed any number of letters, digits and underscores. MATLAB use only the first N characters of the name. Thus, it is necessary to make each variable name unique in the 1st N characters. To find out the value of N for a particular machine, use the following function

  N = namelengthmax

  N = 63

- MATLAB is case sensitive.

2.2 Input and Output of Data

(a) I/O of data from MATLAB command window

In MATLAB, we can deal with vectors and matrices in the same way as scalars.

To input the matrix A and vectors B and C defined below

$$A = \begin{bmatrix} 1 & 2 \\ 6 & 2 \\ 5 & 8 \end{bmatrix}, \qquad B = \begin{bmatrix} -1 & 3 & 5 \end{bmatrix}, \qquad C = \begin{bmatrix} 1.5 \\ 5.2 \\ 2.6 \end{bmatrix}$$

we type in the MATLAB command window:

```
>> A = [1 2; 6 2; 5 8]
```

By pressing enter, the above gives

```
A =

   1   2

   6   2

   5   8
```

Similarly, B and C can be defined by

```
>> B = [-1  3  5];

>> C = [1.5;  5.2;  2.6];
```

Remarks:

- Data values for elements of matrices are separated by some spaces.
- At the end of each row, need to put a semicolon to indicate the end of the row
- At the end of a statement, press <enter> to check the result of executing the statement, or otherwise add a semicolon ";" before pressing enter to indicate that the results are not to be shown.
- The format of output can be controlled by using a "format" command.

```
>> x=1/3
   x =
       0.3333
```

```
>> format long
>> x
x =
   0.33333333333333

>> format long e
>> x
x =
   3.333333333333333e-001
```

(b) Input and Output of data from/to files

MATLAB can handle two types of files.

- Binary format mat_files "***.mat" can preserve the values of more than one variable, but cannot be shared with other programming environment except for the MATLAB environment.
- ASCII data files "***.dat" can be shared with other programming environments but preserve the values of only one variable.

(i) I/O of data from/to mat files

```
>> save xyz x y z        % store the values of x, y, z into the file "xyz.mat"

>> load xyz x y          % read the values of x, y, z from the file "xyz.mat"
```

(ii) I/O of data from/to ASCII files

- To store data into an ASCII dat-file (in the current directory), make the file name the same as the name of the variable storing the data and add '/ascii' at the end of the same statement, namely

```
>> save x.dat x /ascii
```

  Remarks: Only the value of one variable can be saved. Non-numeric data cannot be handled by using a dat-file.

- To read the data from the dat-file in MATLAB, just type the (lower case) filename ***.dat after load, namely

```
>> load x.dat
```

(c) I/O data from keyboard

The 'fprintf()' function (fprintf= "file print formatted") can be used to control the format of printout. The fprintf function uses one argument to give formatting instructions followed by a list of values to be printed, as shown below.

```
fprintf('format string', variables_to_be_printed)
```

The format string controls where and how the values in variables are to the printed. The following is some format strings

%wg : set a total width of w for printing the general real number;

%w.nf : set floating point format-total width w with n digits for fractional part;

%ws : set total width of w for character string output.

eg.

fprintf('%5g',10) □□□10

fprintf('%10.4f',523.456) ⇒ □□523.4560

fprintf('%10s', 'unix') □□□□□□unix

We can input a value or a string from the command line with the input() function.

```
yval=input('Enter a number: ');
name=input('Enter your name: ', 's');
```

eg.

```
>> yval=input('Enter a number: ');
Enter a number: 2
>> yval
yval =  2
>> name=input('Enter your name: ', 's');
Enter your name: SIAM
>> name
name = SIAM
```

2.3 Assignment Statement

Assignment statements are used to perform arithmetic computations and then assign the computed results to variables. The general form of an assignment statement is

variable_name = expression

Once an assignment statement is executed, the following two processes occur.

(1) Firstly, calculate the value of the expression
(2) then assign the value to the variable on the left hand side.

eg. x = 2.0 assign 2.0 to x

x=x+2.6 evaluate x+2.6 to yield 4.6, then the value 4.6 is assigned to x.

Notes: To understand how to correctly perform arithmetic computing by using assignment statements, we need to know how to translate mathematical formulae to arithmetic expressions and how to evaluate an expression.

Writing Arithmetic Expression

An arithmetic expression is a combination of constants, variables, intrinsic functions, operators and parentheses which can be evaluated to give a single value.

(a) Intrinsic Functions

- MATLAB provides a large number of standard elementary mathematical functions such as abs(x), sqrt(x), exp(x) and sin(x). Most of these functions accept complex arguments. To get a list of the elementary functions, type

  >> help elfun

- MATLAB also provides many advanced mathematical and matrix functions such as Bessel and Gamma functions. To get a list of advanced functions, type

  >> help specfun

  >> help elemat

- Elementary functions like sqrt and sin are part of the MATLAB core, while other functions like Bessel are implemented in M-files.

(b) Operators

The basic arithmetic calculation can be performed by using the following operators in MATLAB.

Operations	Operators
Addition	+
Subtraction	-
Multiplication	*
Division	/
Power	^
Complex conjugate transpose	’

To evaluate arithmetic expressions, MATLAB assigns the same priorities to operators as does mathematics.

3. MATLAB CONTROL STRUCTURES

3.1 Logical Expressions

Logical expressions are used to describe mathematical conditions. In general, a logical expression consists of relational expressions, logical constants, logical variables and logical operators.

(a) Relational expression

- A relational expressions compare the values of two arithmetic expressions using a relational operator, namely
  Arithmetic_expression1 *relational_operator* arithmetic_expression2

  eg. t > 2+x

List of relational operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal (check equality for two scalars.
~=	Not equal
isequal(A,B)	Yields 1(true) if matrix A equals matrix B
isempty(A)	Yields 1(true) if matrix A is empty.

- A relational expression can be used to describe simple conditions such as a>b+1. If the condition is true, the expression yields a value 1 (true) or otherwise 0 (false).

(b) Logical operators and logical expressions

By combining the relational expressions together using logical operators (&,|,~), we can form a logic condition to describe a complicated condition.

- **Definition of logical operators**

Name	Operators	MATLAB codes	Values
AND	&	a&b	1 (true) if both a and b are true.
OR	|	a|b	1 (true) if at least one of the a,b is true.
Not	~	~a	1 (true) if a is 0 (false) or vice versa.

- **Priority of logical operations**

Type	Operator	Execution order
Bracket	()	1
Arithmetic calculation	+,-,*,/,^,'	2
Relational calculation	<,<=,>,>= ==,~=, isequal(A,B) isempty(A)	3
Logical calculation	~	4
	&	5
	|	6

3.2 Selection Control

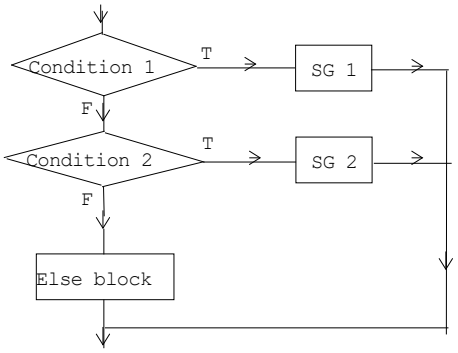
In practice, most problem require us to choose between alternative courses of action, depending upon circumstances that are not determined until the program is executed. MATLAB provides two selection structures for choosing alternative courses of action including the if-elseif-end structure and the switch-case structure.

(a) if-elseif-end structure

MATLAB code:

```
if  condition_1
    block_1
elseif  condition_2
   block_2
:
else
  else_block
end
```

Flow chart



eg.

```
A=input('Enter the first number: ');
B=input('Enter the second number: ');
if A > B
  'The first one is greater.'
elseif A < B
  'The first one is smaller.'
elseif A == B
  'Both are equal.'
else
  error('Unexpected situation')
end
```

(b) switch-case structure

MATLAB code:

```
switch (case_expression)
    case  case_selector_1
       block_1 of statements
         case  case_selecter_2
       block_2 of statements
       :
    otherwise  block_D of statement
end
```

Program for arithmetic calculation

```
A=input('Enter number a: ');
ch=input('Enter (+,-,*,/): ', 's');
B=input('Enter number b: ');
switch ch
    case '+'
        M = A+B;
    case '-'
        M = A-B;
    case '*'
        M = A*B;
    case '/'
        M = A/B;
    Otherwise
        error('This is impossible')
end
```

3.2 Loop Control

MATLAB provides two repetition structures, including for loop and while loop for repeating certain part of the program if certain condition is true.

(a) For loop

If the number of iteration is known or can be predetermined, we can use a counter-controlled for loop for the repetitive counting. The general form of MATLAB code for the for loop is :

```
for index = i_0 : increment : i_last
    statements
end
```

If the increment is 1, it can be omitted.

eg.
```
for i=1:m
  for j=1:n
     A(i,j)=1/(i+j+5);
  end
end
```

(b) While loop

A while loop repeats a group of statements while certain condition is true. The general form is

```
while condition
   statements
end
```

eg. The following codes solve the equation $x-\frac{\pi}{2}-\frac{1}{3}\sin x=0$ by using the fixed point iteration method $x_{i+1}=\frac{\pi}{2}+\frac{1}{3}\sin x_i=0$ from the initial guess $x_0=0,$ and stop when $|x_{i+1}-x_i|<\text{Tol.}$.

```
Tol=0.00000001;
x0=0.0;
x=pi/2+(1/3)*sin(x0);
while (abs(x-x0) > Tol)
   x0=x;
   x=pi/2+(1/3)*sin(x0);
end
```

(c) Continue and break statements

The continue statement passes control to the next iteration of the while loop.

4. Matlab MATRIX AND ARRAY CALCULATION

MATLAB provides facilities for matrix calculation at two different levels including linear algebra matrix operations and array element-by-element operations.

4.1 Linear Algebra Matrix Operations

(a) The following table lists the basic linear algebra matrix operations

Operations	*Operators*	*MATLAB code*	*Linear algebra form*
Transpose	'	A'	A^T
Addition/Subtraction	+(-)	A+B(A-B)	A+B(A-B)
Matrix multiplication	*	A*B	AB
Matrix left division	\	X=A\b	X=INV(A*b (solution of AX=b)
Matrix right division	/	X=B/A	X=B*INV(A) (solution of XA=B)

(b) MATLAB also provide various functions specially for linear algebra matrix operations eg.

[L,U]=LU(A) ⇒ factors A to L and U

INV(A) ⇒ calculates an inverse of A

X=A\b ⇒ solves of Ax=b

Eig(A) ⇒ returns the eigenvalues of A

[X,D]=eig(A) ⇒ Diagonal elements of D are eigenvalues of A and columns of X are the corresponding eigenvectors such that AX=XD.

[y,i]=max(X) ⇒ y=maximum, i=the index of the maximum value in X

4.1 Array Operation on Element-by-Element Basis

Array operation refers to element-by-element arithmetic operation instead of the usual linear algebra matrix operation. Preceding an operator with a period '.' indicates an array or element-by-element operation. The following lists some basic array element-by-element operations.

Array multiplication and division

For X=[1,2]; Y=[4,5]

Z=X.*Y

results in

Z=[4, 10]

Element by element power

For X=[1,2]; Y=[4,5]

Z=X.^Y

yields

Z=[1,32]

Matrix function

exp(A) and sqrt(A) are computed on element-by-element based.

Subscript triplet notation and matrix generation

subscript_b : stride: subscript_e

defines an ordered set of subscripts that starts at subscript_b and end on or before subscript_e and have a separation of stride between consecutive subscripts.

eg. 0:2:8 defines an order set 0 2 4 6 8

-0.5:0.25:0.5 defines an order set -0.5 -0.25 0 0.25 0.5

For x = 0.0:0.2:0.6;

y = exp(-x).*sin(x);

[x y] defines a matrix with vector x as column 1 and vector y as column 2.

0 0

0.2 0.1627

0.4 0.2610

0.6 0.3099

5. M-FILES : SCRIPTS AND FUNCTIONS

MATLAB is usually used in a command-driven mode. When a single-line command is entered, MATLAB processes it immediately. MATLAB is also capable of executing sequences of commands stored in files. Disk files that contain MATLAB statements are called M-files because they have a file type of ".m".

There are two kinds of M files

- Script files which contain a long sequence of MATLAB commands (program)
- Function files which define new functions that solve user-specific problems.

Both types of M-files are ASCII text files and can be created using an editor or word processor.

5.1 Script Files

When a script is invoked, MATLAB simply executes the commands in the file. The statements in the script file operate globally. Scripts are useful for solving problems that require long sequences of commands.

5.2 Function Files

If the first line of an M-file contains the keyword "function", the file is a function file. A function differs from a script in that arguments may be passed, and that variables inside the file are local only and do not operate globally. Function files create new MATLAB functions for solving specific problems using MATLAB language.

There are three forms of function definitions based on the number of outputs to be returned by the function

- **No output**

```
function func_name(arg1,…,argN)
```

or

```
function []=func_name(arg1,…, argN)
```

eg.
```
function printmessage()
% Print the messeage  "Thank you for using our services"
fprintf('%20s','Thanks for using our services');
```

- **Single-value output**

```
function  output=func_name(arg1,…,argN)
```

eg.
```
function y = mean(X)
[m,n] = size(X);
if (m = = 1)
      m = n;
end
y=sum(X)/m;
```

- **Multiple-value output**

```
function  [output1,…,outputM]=func_name(arg1,…,argN)
```

eg.
```
function [x,y]=polar(theta, r)
x = r*cos(theta);
y = r*sin(theta);
```

Remarks

(1) When a M-function file is invoked for the first time during a MATLAB session, it is complied and placed into memory. It is then available for subsequent use with recompilation. It remains in memory for the duration of the session.

(2) The what command shows the list of M-files in the current directory on your disk.

(3) User can put all his/her m-files in a folder within the MATLAB folder as his/her own library of M-files. MATLAB responses to them in the normal way.

6. GRAPHICS

6.1 2-D Plots

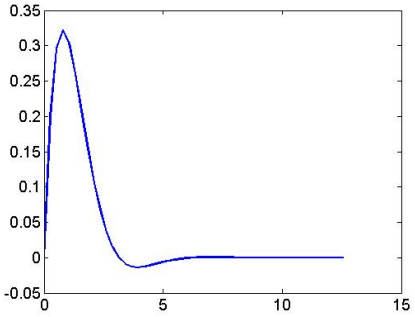
(a) Line x-y plot

If x and y are vectors of the same length, the command plot(x,y) draws an x-y plot of the elements of x versus the elements of y.

For example, the following codes

```
x = 0:pi/12:4*pi;
y=sin(x).*exp(-x);
plot(x,y)
```

produce a 2-D plot of a scalar-valued function $y = \sin(x)e^{-x}$ versus x, as shown on the right.



Remarks

(1) The statement in line 1 "x = 0:pi/12:4*pi;" creates a sequence of data values starting form o and ending at 4*pi with increment pi/12, and then stores the values in x.
(2) The statement in lines 2 generates a vector y from x via array element-element calculation.
(3) In plot(x,y), if x is a vector, y is a matrix and the number of element in each column of y is the same as the number of element in x, then plot(x,y) plots columns of y versus x, using a different line type for each.
(4) If x and y are both matrices of the same size, plot(x,y) plots the columns of x versus the corresponding columns of y.
(5) The color of the lines can be specified by using the color codes : r (red), g (green), b (blue), w (white).

For example,

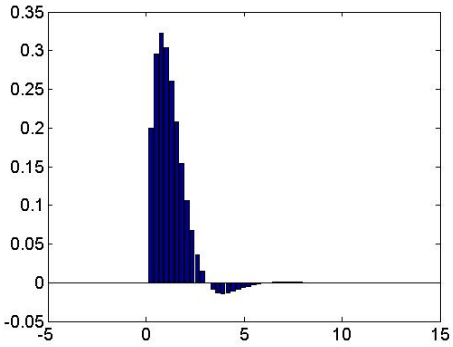
```
plot(x,y,'r')          % use a red line
plot(x,y,'+g')                          % use green line and + marks
```

(b) Bar plot

Example, the following codes

```
x = 0:pi/12:4*pi;
y = sin(x).*exp(-x);
bar(x, y);
```

produce a vertical bar chart of a scalar-valued function $y = \sin(x)e^{-x}$ versus x, as shown on the right.



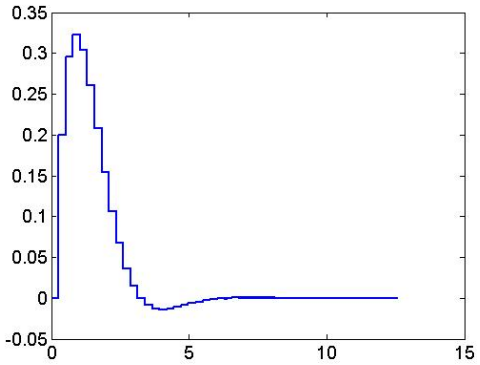
Remarks: bar(x, y) creates a vertical bar chart y versus x .

(c) Stairstep plot

Example. The following codes

```
x = 0:pi/12:4*pi;
y = sin(x).*exp(-x);
stairs(x,y);
```

produce a stair-step plot of y versus x, as shown on the right.



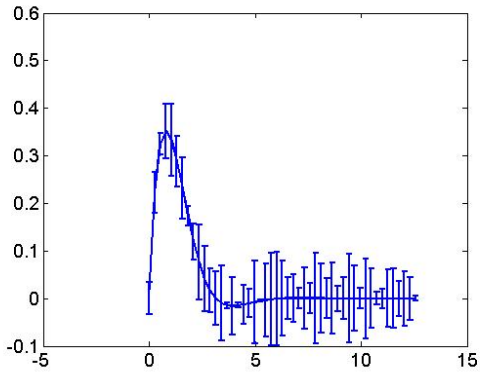
Remarks: stairs(x,y) creates a stair-step plot of y versus x.

(d) Errorbar plot

Example, the following codes

```
x=0:pi/12:4*pi;
y= sin(x).*exp(-x)
ey=erf(y);
e= rand(size(y))/10;
errorbar(x,ey,e);
```

plot the error bars to show the confidence level of data or the deviation along a curve, as shown on the right.



Remarks :

erf(), rand(), size() and errorbar() are built-in functions:
erf() represents an error function defined by

$$erf(y) = \int_0^y e^{-t^2}\, dt\ .$$

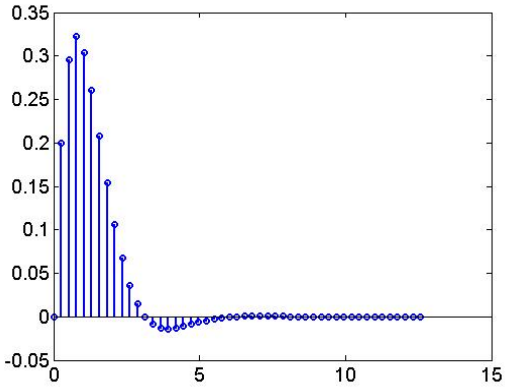
size() returns the numbers of rows/columns of a 1-D/2-D/3-D array.
rand() generates a vector consisting of uniformly distributed random numbers.
errorbar() shows the confidence level of data or the deviation along a curve. The command errorbar(x,ey,e) plots ey versus x with symmetric error bars 2*e(i) long.

(e) **Stem plot**

Example, the following codes

```
x = 0:pi/12:4*pi;
y = sin(x).*exp(-x);
stem(x,y)
```

produce discrete-sequence data plot of a scalar-valued function $y = \sin(x)e^{-x}$ versus x, as shown on the right.



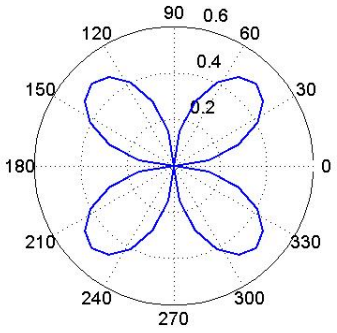
Remarks: stem() plots discrete sequence data y versus x.

(f) **Polar plot**

Example, the following codes

```
theta = 0:pi/20:2*pi;
r = sin(theta).*exp(-theta);
polar(theta,abs(r));
```

plot a graph r = r(θ) in polar coordinates in a Cartesian plane with polar grid, as shown on the right.



Remarks

(1) The statement in lines 2 generates a vector r from theta via array element-element calculation.

(2) polar()plots r=r(θ)in polar coordinates in a Cartesian plane with polar grid.

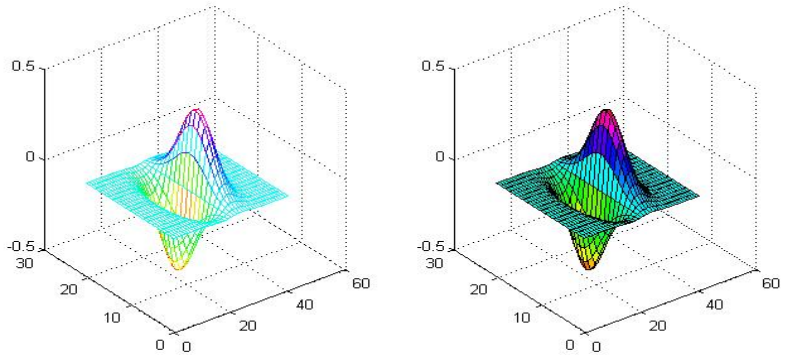
6.2 3-D Plots

(a) Mesh plot and surface plot

The following codes

```
xi = -4:.2:4;
yi = -2:.2:2;
[x,y] = meshgrid(xi,yi);
f = x.*exp(-x.^2 -y.^2);
subplot(1,2,1);
mesh(f);
colormap(hsv)
subplot(1,2,2);
surf(f);
```

generate two figures below including a mesh type graph and a 3-D shaded surface of f(x,y), respectively.



Remarks:

- The statement in line 1 "x = -4:0.2:4" creates a sequence of data values starting form -4 and ending at 4 with increment 0.2, and then stores the values in x.

- The statement in line 2 "y = -2:0.2:2" creates a sequence of data values starting form -2 and ending at 2 with increment 0.2, and then stores the values in y.
- meshgrid() in line 3 generates grid points for plotting a mesh-type graph.
- mesh() plots a mesh type graph of f(x,y).

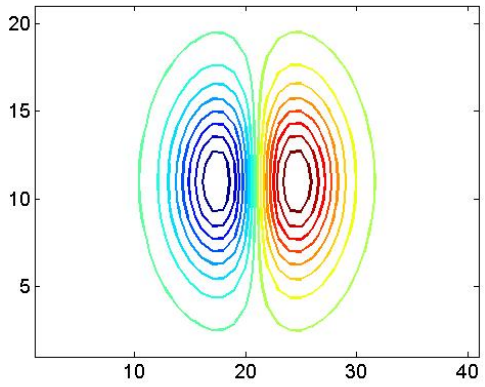
- subplot() divides the current figure into rectangular panes.
- In the statement "subplot(nrow, ncol, fops)", the first two arguments set the figure to consist of nrow*ncol subfigures. The last argument specifies the location of each subfigure. In this example, there are two subfigures. One is plotted on the left column following the command "subplot(1, 2, 1)" as shown in line 5 and another is plotted on the right column following the command "subplot(1, 2, 2)" as shown in line 8.
- colormap() is a built-in function for setting a color map which is a m-by-3 matrix of real numbers between 0.0 and 1.0. Each row is a RGB vector that defines one color.
- surf(f) creates a 3-D shaded surface of f(x,y).

(b) Contour plot

Example, the following codes

```
xi = -4:0.2:4;
yi = -2:0.2:2;
[x,y] = meshgrid(xi,yi);
z = x.*exp(-x.^2 -y.^2);
contour(z,16);
```

generate a 2-D contour plot of a scalar-valued function $z = xe^{(-x^2)(-y^2)}$ on grid point (x,y) generated from points (xi,yi), as shown on the right.



Remarks:

(1) meshgrid() in line 3 generates grid points for plotting a mesh-type graph.

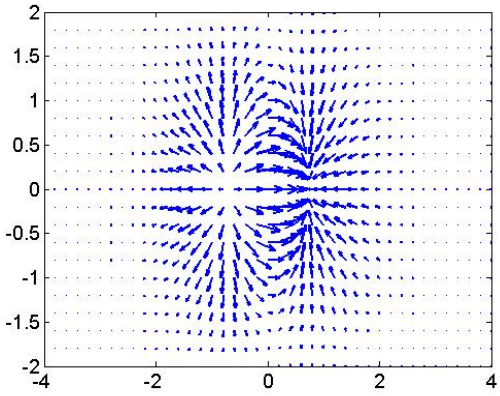
(2) contour(arg1,arg2) gives a 2-D contour plot of a scalar-valued function of two variables. In line 5, the statement "contour(z,16);" creates 16 contour lines of a scalar-valued function z.

(c) Vector plot

Example, the following coeds

```
xi = -4:0.2:4;
yi = -2:0.2:2;
[x,y] = meshgrid(xi,yi);
z = x.*exp(-x.^2 -y.^2);
[px,py] = gradient(z,.2,.2);
quiver(xi,yi,px,py,2);
```

plot a mesh-type graph of gradient vectors on grid points (xi, yi), as shown on the right.



Remarks:

(1) gradient(arg1,arg2,arg3) produces numerical gradient on each grid point.

(2) quiver(arg1,arg2,arg3,arg4,arg5) plots gradient vectors. In this example, "quiver(xi,yi,px,py,2)" plots gradient vector [px, py] corresponding to vector [xi, yi] with the scale of size 2. The last argument is optional, if it is omitted, quiver plots gradient vector with the scale of size 1.