MATLAB and Its Application in Engineering

Assoc. Prof. Kirin Shi

Shanghai Jiao Tong University

Notes

1. Welcome to our QQ group 615277998, all the information about homework, queries and exams will be informed in this chatting community.

2. How to submit your homework?

a) Please sent your homework with the compressed format .rar or .7z

b) Naming format should be "Name + studentID + Homework number" for example "梅迪_116020910137_1".

c) The compressed file should include a doc file and its pdf version, and also the .m file for the codes of every single question.

Review on Lecture 2

Elementary Matrix Operations

Systems of Linear Equations

Left division "\" → **Gaussian Elimination**

```
>> A = [2 3 1; 0 1 2; 4 2 0];
>> B = [4; 6; 12];
>> X = A\B
X =
    4.6667
   -3.3333
    4.6667
```

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & 1 & 2 \\ 4 & 2 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ 12 \end{bmatrix}$$

Element-by-element operations

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$$

$$A .* B = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} & A_{13}B_{13} \\ A_{21}B_{21} & A_{22}B_{22} & A_{23}B_{23} \\ A_{31}B_{31} & A_{32}B_{32} & A_{33}B_{33} \end{bmatrix} \qquad \mathrm{A} .\hat{} \, n = \begin{bmatrix} (A_{11})^n & (A_{12})^n & (A_{13})^n \\ (A_{21})^n & (A_{22})^n & (A_{23})^n \\ (A_{31})^n & (A_{32})^n & (A_{33})^n \end{bmatrix}$$

$$A ./ B = \begin{bmatrix} A_{11}/B_{11} & A_{12}/B_{12} & A_{13}/B_{13} \\ A_{21}/B_{21} & A_{22}/B_{22} & A_{23}/B_{23} \\ A_{31}/B_{31} & A_{32}/B_{32} & A_{33}/B_{33} \end{bmatrix}$$

Functions length, size, sum, mean, max, min

```
>> A = [1 2 3; 3 4 5; 6 7 9; 12 13
   14]
A =
     1     2     3
     3     4     5
     6     7     9
    12    13    14
```

min max mean

```
>> sum_A = sum(A,1)
sum_A =
    22    26    31
>> sum_A = sum(A,2)
sum_A =
     6
    12
    22
    39
```

Data type

String

- A string is an array of characters. It is created by typing the characters within single quotes.
- Strings can also be placed in a matrix by typing a semicolon ; (or pressing the Enter key) at the end of each row.
- All rows must have the same number of elements.

```
>> B='My name is John Smith'
B =
My name is John Smith
```

```
>> B(12:15)='Bill'
B =
My name is Bill Smith
>>
```

Using a colon to assign new characters to elements 12 through 15 in the vector B.

String in a matrix

```
variable_name = char('string 1','string 2','string 3')
```

```
>> Info=char('Student Name:','John Smith','Grade:','A+')

Info =
Student Name:
John Smith
Grade:
A+
>>
```

A variable named `Info` is assigned four rows of strings, each with different length.

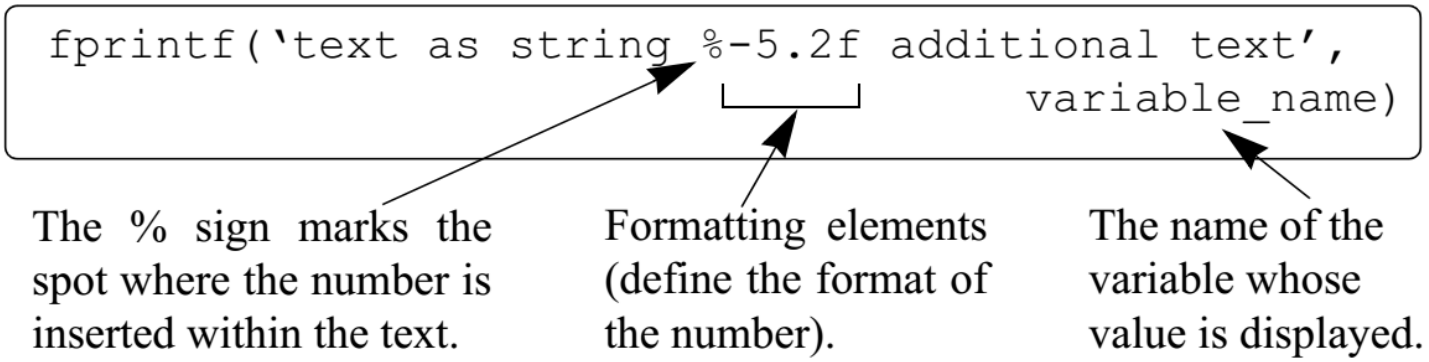
The function `char` creates an array with four rows with the same length as the longest row by adding empty spaces to the shorter lines.

```
>> x=536

x =
   536

>> y='536'

y =
536
>>
```

Data input and output

Output data to Screen

- Using the **fprintf** command to display a mix of text and numerical data:




```
>> fprintf('Kirin received %1.2f for his Matlab course \n',80);

Kirin received 80.00 for his Matlab course
```

Output data to a file

Example: fprintf_exp1.m

```
data1=[1;2;3;4];   data2=[5;6;7;8]; data3=[9;10;11;12];
result=[data1,data2,data3];
file_head = ['This is an example for fprint fopen command'];
fid = fopen('C:\Matlab_course\example2_3.dat', 'wt+');
fprintf(fid, '%s \n', file_head, 'VARIABLES = "X""Y""Z"');
[row,col] = size(result);
for i= 1:row
    for j= 1:col-1
        fprintf(fid, '%f\t', result(i, j));
    end
     fprintf(fid, '%f\n', result(i, col));
end
fclose(fid);
```

xlsread, xlswrite
imwrite, fwrite, fscanf, fread, fgetl, fgets

End of Review on Lecture 2

Relational & Logical operations

Relational operations

- A relational operator compares two numbers by determining whether a comparison statement (e.g., 5 < 8) is true or false.
- If the statement is true, it is assigned a value of 1, 0 otherwise.
- A logical operator examines true/false statements and produces a result that is true (1) or false (0) according to the specific operator.
- Relational and logical operators can be used in mathematical expressions and, are frequently used to control the flow of a computer program.

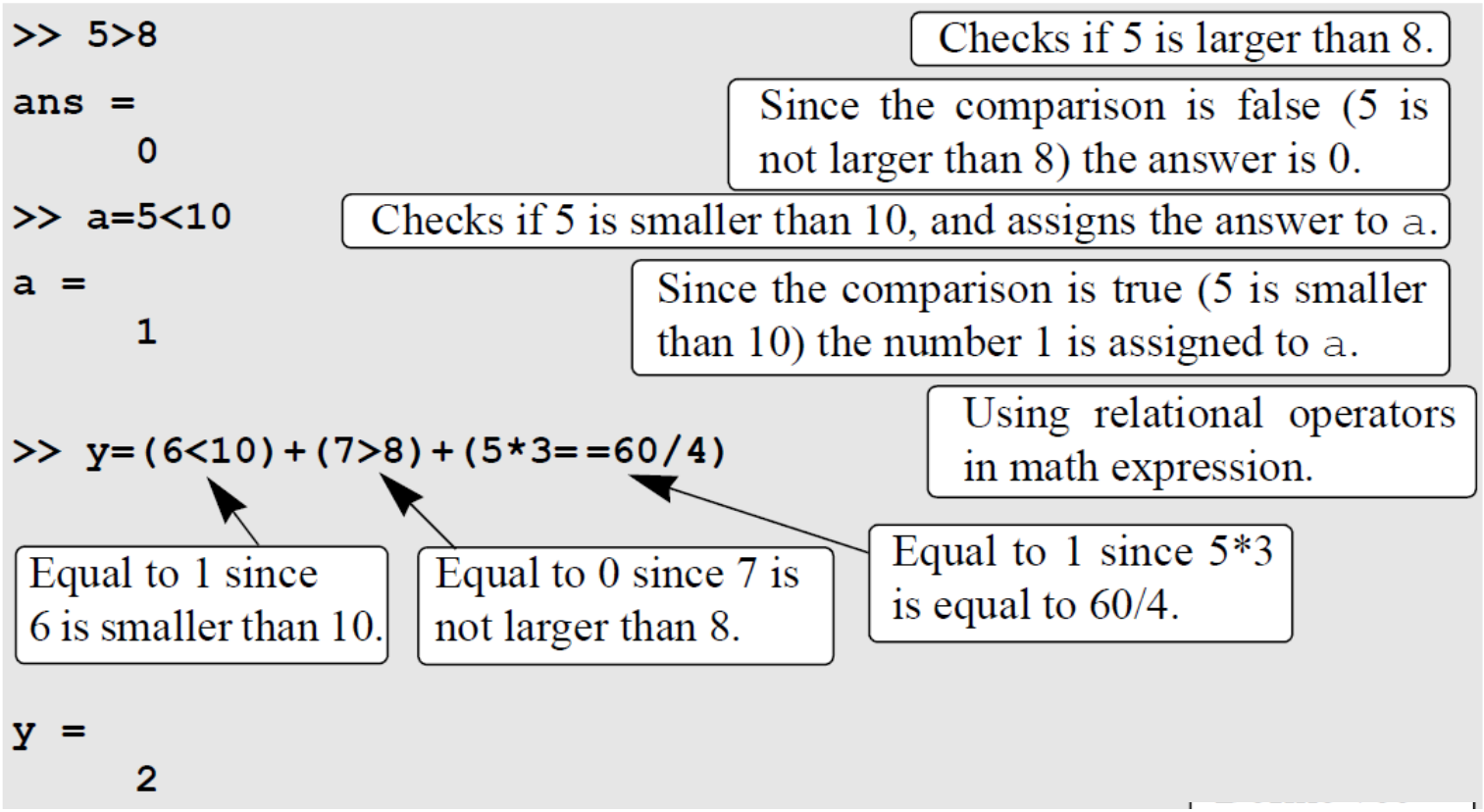
Relational operations

Relational operators in MATLAB are:

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

Relational operations

- When two numbers are compared, the result is 1 (logical true) if the comparison, according to the relational operator, is true, and 0 (logical false) if the comparison is false.



Relational operations

- If two arrays are compared (only arrays of the same size can be compared), the comparison is done element-by-element

```
      2
;          4 11 7 14]; c=[8 20 9 2 19 7 10];
```

Define vectors b and c.

```
>> d=c>=b
```

Checks which c elements are larger than or equal to b elements.

```
d =
     0     1     1     0     1     1     0
```

Assigns 1 where an element of c is larger than or equal to an element of b.

```
>> b == c
```

Checks which b elements are equal to c elements.

```
ans =
     0     0     1     0     0     1     0
>> b~=c
```

Checks which b elements are not equal to c elements.

```
ans =
     1     1     0     1     1     0     1
>> f=b-c>0
```

Subtracts c from b and then checks which elements are larger than zero.

```
f =
     1     0     0     1     0     0     1
```

Relational operations

- If a scalar is compared with an array, the scalar is compared with every element of the array, and the result is a logical array with 1s and 0s according to the outcome of the comparison of each element.

```
>> A=[2 9 4; -3 5 2; 6 7 -1]
A =
     2     9     4
    -3     5     2
     6     7    -1
>> B=A<=2
```

Define a 3×3 matrix A.

Checks which elements in A are smaller than or equal to 2. Assigns the results to matrix B.

```
B =
     1     0     0
     1     0     1
     0     0     1
```

Relational operations

- The results of a relational operation with vectors, which are vectors with 0s and 1s, are called logical vectors and can be used for addressing vectors.
- When a logical vector is used for addressing another vector, it extracts from that vector the elements in the positions where the logical vector has 1s.

```
>> r = [8 12 9 4 23 19 10]
```
Define a vector r.

```
r =
     8    12     9     4    23    19    10
>> s=r<=10
```
Checks which r elements are smaller than or equal to 10.

```
s =
     1     0     1     1     0     0     1
```
A logical vector s with 1s at positions where elements of r are smaller than or equal to 10.

```
>> t=r(s)
```
Use s for addresses in vector r to create vector t.

```
t =
     8     9     4    10
```
Vector t consists of elements of r in positions where s has 1s.

```
>> w=r(r<=10)
```
The same procedure can be done in one step.

```
w =
     8     9     4    10
```

Relational operations

- Numerical vectors and arrays with the numbers 0s and 1s are not the same as logical vectors and arrays with 0s and 1s. They can not be used for addressing.
- Logical vectors and arrays, however, can be used in arithmetic operations.

Relational operations

```
>> r = [8 12 9 4 23 19 10]
r =
    8    12     9     4    23    19    10
>> s=r<=10
s =
    1     0     1     1     0     0     1
>> s+r
ans =

    9    12    10     5    23    19    11
>> whos
  Name      Size            Bytes  Class     Attributes

  ans       1x7                56  double
  r         1x7                56  double
  s         1x7                 7  logical
```

Relational operations

- Order of precedence: In a mathematical expression that includes relational and arithmetic operations, the arithmetic operations have precedence over relational operations.
- The relational operators themselves have equal precedence and are evaluated from left to right.
- Parentheses can be used to alter the order of precedence.

```
>> 3+4<16/2
ans =
     1
>> 3+(4<16)/2
ans =
    3.5000
```

+ and / are executed first.

The answer is 1 since 7 < 8 is true.

4 < 16 is executed first, and is equal to 1, since it is true.

3.5 is obtained from 3 + 1/2.

Logical operations

Logical operators in MATLAB are:

Logical operator	Name	Description
& Example: A&B	AND	Operates on two operands (A and B). If both are true, the result is true (1); otherwise the result is false (0).
| Example: A|B	OR	Operates on two operands (A and B). If either one, or both, are true, the result is true (1); otherwise (both are false) the result is false (0).
~ Example: ~A	NOT	Operates on one operand (A). Gives the opposite of the operand; true (1) if the operand is false, and false (0) if the operand is true.

Logical operators

expression A	expression B	expression A & B
True	True	True
True	False	False
False	True	False
False	False	False

- For example,**5>3** is **True** and **2==1+3** is **False**. Therefore, the expression **5>3 & 2==1+3** is **False**.

```
>> b = 5>3 & 2==1+3

b =

     0
```

Logical operators

expression A	expression B	expression A | B
True	True	True
True	False	True
False	True	True
False	**False**	**False**

- For example,**5>3** is **True** and **2==1+5** is **False**. Therefore, the expression **5>3 | 2==1+5** is **True**.

```
>> b = 5>3 | 2==1+5

b =

     1
```

Logical operations

- Logical operators have numbers as operands. A nonzero number is true, and a zero number is false.
- Logical operators (like relational operators) are used as arithmetic operators within a mathematical expression. The result can be used in other mathematical operations, in addressing arrays, and together with other MATLAB commands (e.g., if) to control the flow of a program.

```
>> 3&7                                   3 AND 7.
ans =                                    3 and 7 are both true (nonzero), so the outcome is 1.
     1
>> a=5|0                                 5 OR 0 (assign to variable a).
a =                                      1 is assigned to a since at least one number is true (nonzero).
     1
>> ~25                                   NOT 25.
ans =                                    The outcome is 0 since 25 is true
     0                                   (nonzero) and the opposite is false.
```

Logical operations

- The logical operations AND and OR can have both operands as scalars, arrays, or one array and one scalar.
- If both are arrays, they must be of the same size and the logical operation is done element-by-element.
- If one operand is a scalar and the other is an array, the logical operation is done between the scalar and each of the elements in the array.

```
>> x=[9 3 0 11 0 15]; y=[2 0 13 -11 0 4];
>> x&y
ans =
     1     0     0     1     0     1
>> z=x|y
z =
     1     1     1     1     0     1
```

The outcome is a vector with 1 in every position where both x and y are true (nonzero elements), and 0s otherwise.

The outcome is a vector with 1 in every position where either or both x and y are true (nonzero elements), and 0s otherwise.

Logical operations

- The logical operations AND and OR can have both operands as scalars, arrays, or one array and one scalar.
- If both are arrays, they must be of the same size and the logical operation is done element-by-element.
- If one operand is a scalar and the other is an array, the logical operation is done between the scalar and each of the elements in the array.

```
>> x=[9 3 0 11 0 15];
>> 5&x
ans =
    1    1    0    1    0    1

>> 5|x
ans =
    1    1    1    1    1    1
```

Logical operations

- The logical operation NOT has one operand. When it is used with a scalar the outcome is a scalar 0 or 1.
- When it is used with an array, the outcome is an array of the same size with 1s in positions where the array has nonzero numbers and 0s in positions where the array has 0s.

```
>> x=[9 3 0 11 0 15]; y=[2 0 13 -11 0 4];
ans =
   11     3    13     0     0    19

>> ~(x+y)
ans =
    0     0     0     1     1     0
```

Logical operations

Order of precedence:

Precedence	Operation
1 (highest)	Parentheses (if nested parentheses exist, inner ones have precedence)
2	Exponentiation
3	Logical NOT (~)
4	Multiplication, division
5	Addition, subtraction
6	Relational operators (>, <, >=, <=, ==, ~=)
7	Logical AND (&)
8 (lowest)	Logical OR (|)

Logical operations

Order of precedence:

```
>> x=-2; y=5;
```
Define variables x and y.

```
>> -5<x<-1
ans =
     0
```
This inequality is correct mathematically. The answer, however, is false since MATLAB executes from left to right. $-5 < x$ is true (=1) and then $1 < -1$ is false (0).

```
>> -5<x & x<-1
ans =
     1
```
The mathematically correct statement is obtained by using the logical operator &. The inequalities are executed first. Since both are true (1), the answer is 1.

```
>> ~(y<7)
ans =
     0
```
$y < 7$ is executed first, it is true (1), and ~1 is 0.

```
>> ~y<7
ans =
     1
```
~y is executed first. y is true (1) (since y is nonzero), ~1 is 0, and $0 < 7$ is true (1).

```
>> ~((y>=8)|(x<-1))
ans =
     0
```
$y >= 8$ (false), and $x < -1$ (true) are executed first. OR is executed next (true). ~ is executed last, and gives false (0).

```
>> ~(y>=8)|(x<-1)
ans =
     1
```
$y >= 8$ (false), and $x < -1$ (true) are executed first. NOT of ($y >= 8$) is executed next (true). OR is executed last, and gives true (1).

Logical operations

Build in functions:

Function	Description	Example

`xor(a,b)`	Exclusive or. Returns true (1) if one operand is true and the other is false.	`>> xor(7,0)`
		`ans =`
		`1`
		`>> xor(7,-5)`
		`ans =`
		`0`

`all(A)`	Returns 1 (true) if all elements in a vector `A` are true (nonzero). Returns 0 (false) if one or more elements are false (zero).
	If `A` is a matrix, treats columns of `A` as vectors, and returns a vector with 1s and 0s.
		`>> A=[6 2 15 9 7 11];`
		`>> all(A)`
		`ans =`
		`1`
		`>> B=[6 2 15 9 0 11];`
		`>> all(B)`
		`ans =`
		`0`

Logical operations

Build in functions:

`any(A)`	Returns 1 (true) if any element in a vector `A` is true (nonzero). Returns 0 (false) if all elements are false (zero). If `A` is a matrix, treats columns of `A` as vectors, and returns a vector with 1s and 0s.	`>> A=[6 0 15 0 0 11];`
		`>> any(A)`
		`ans =`
		`1`
		`>> B = [0 0 0 0 0 0];`
		`>> any(B)`
		`ans =`
		`0`

`find(A)`
 	
`find(A>d)`
	If `A` is a vector, returns the indices of the nonzero elements.
	If `A` is a vector, returns the address of the elements that are larger than `d` (any relational operator can be used).
	A(find(A>4))
		`>> A=[0 9 4 3 7 0 0 1 8];`
		`>> find(A)`
		`ans =`
		`2 3 4 5 8 9`
		`>> find(A>4)`
		`ans =`
		`2 5 9`

Logical operations

`and(A,B)` equivalent to `A&B`
`or(A,B)` equivalent to `A|B`
`not(A)` equivalent to `~A`

INPUT		OUTPUT				
A	B	AND A&B	OR A|B	XOR (A,B)	NOT ~A	NOT ~B
false	false	false	false	false	true	true
false	true	false	true	true	true	false
true	false	false	true	true	false	true
true	true	true	true	false	false	false

Applications

The following were the daily maximum temperatures (in ^{0}C) in Shanghai,

during the month of April 2000:

14 23 23 12 10 9 13 23 23 19 21 17 23 28 29

33 34 32 33 27 15 21 13 18 17 19 18 23 17

21.

Use relational and logical operations to determine the following:

(a) The number of days the temperature was above 23 ^{0}C.

(b) The number of days the temperature was between 18 ^{0}C and 27 ^{0}C.

(c) The days of the month when the temperature was between 10 ^{0}C and 16 ^{0}C.

Applications

```
>> T=[14   23   23   12   10    9   13   23   23   19   21   17   23   28
29  33 34   32   33   27   15   21   13   18   17   19   18   23   17   21];

>> T_above_23=T>=23;
>> Ndays_T_above_23=sum(T_above_23);
>> T_bwn_18_27=(T>=18)&(T<=27);
>> Ndays_T_bwn_18_27=sum(T_bwn_18_27);
>> dates_T_bwn_10_16=find((T>=10)&(T<=16));

Ndays_T_above_23 =
13
Ndays_T_bwn_18_27 =
14
dates_T_bwn_10_16 =
 1    4    5    7   21   23
```

Conditional Control Structures

Conditional Statements

A conditional statement is a command that allows MATLAB to make a decision

of **whether to execute** a group of commands that follow the conditional statement, or to **skip** these commands.

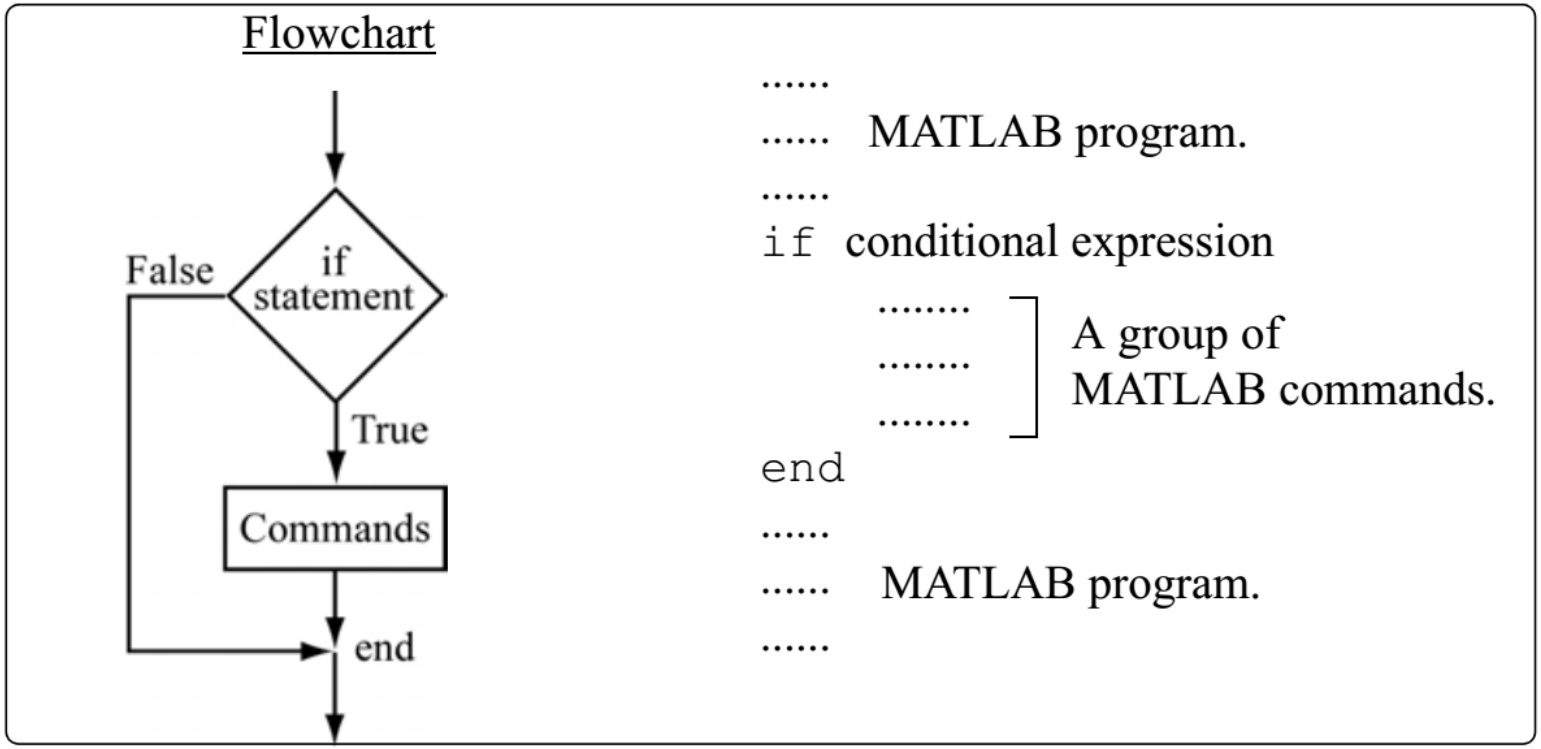
`if` conditional expression consisting of relational and/or logical operators.

Examples:

```
if a < b
if c >= 5
if a == b
if a ~= 0
if (d<h)&(x>7)
if (x~=13)|(y<0)
```

All the variables must have assigned values.

The if-end Structure



A worker is paid according to his hourly wage up to 40 hours, and 50% more for overtime.

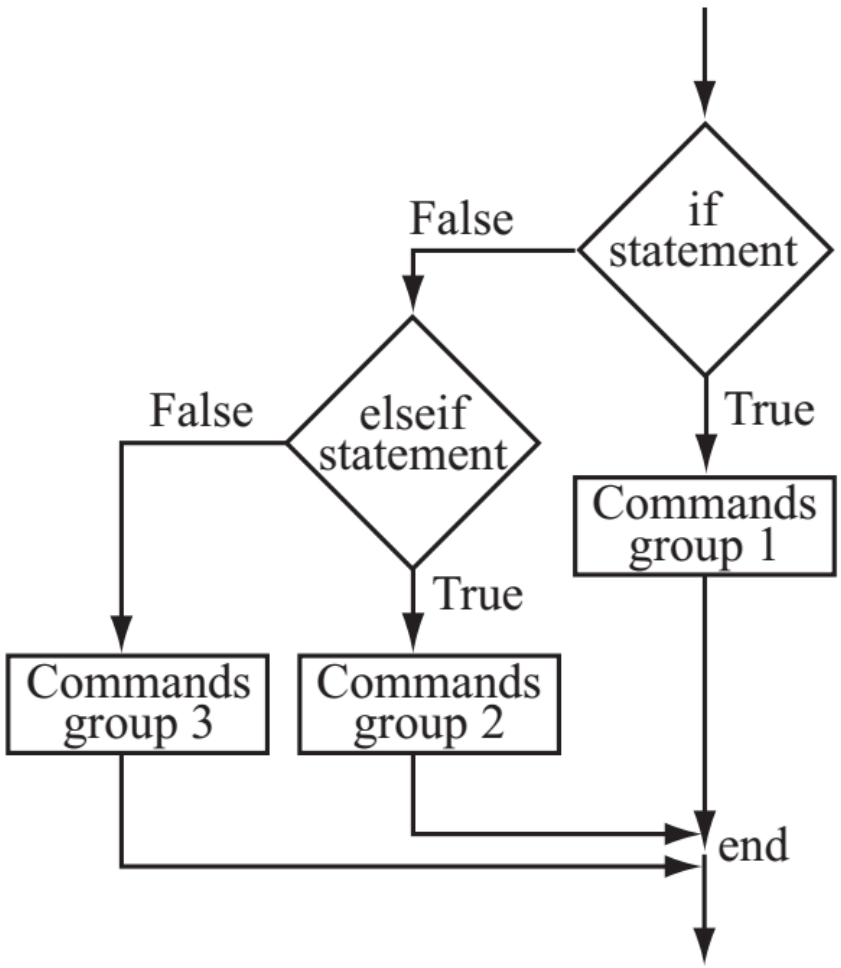
- Write a program in a script file that calculates the pay to a worker.
- The program asks the user to enter the number of hours and the hourly wage.
- The program then displays the pay.

The if-end Structure

```
t=input('Please enter the number of hours worked  ');
h=input('Please enter the hourly wage in $  ');
Pay=t*h;
if t>40
    Pay=Pay+(t-40)*0.5*h;
end
fprintf('The worker''s pay is  $ %5.2f',Pay)
```

```
>> Workerpay
Please enter the number of hours worked  35
Please enter the hourly wage in $  8
The worker's pay is  $ 280.00
>> Workerpay
Please enter the number of hours worked  50
Please enter the hourly wage in $  10
The worker's pay is  $ 550.00
```

The if-elseif-end Structure




```
.....
.....   MATLAB program.

if conditional expression
        .....
        .....   Group 1 of
        .....   MATLAB commands.
elseif conditional expression
        .....
        .....   Group 2 of
        .....   MATLAB commands.
else
        .....
        .....   Group 3 of
        .....   MATLAB commands.
end
.....
.....   MATLAB program.
.....
```

Example

- Buying box cars for a railroad company

No. of cars (N)	Price of each car (K)
N ≤ 3	22
4 ≤ N ≤ 10	20
N > 10	19

Example

```
function cost = box_car_cost(N);
Low_price = 19;
Medium_price = 20;
High_price = 22;
if N > 10
   cost = Low_price * N;
elseif N > 3
   cost = Medium_price * N;
else
   cost = High_price * N;
end
```

No. of cars (N)	Price of each car (K)
N ≤ 3	22
4 ≤ N ≤ 10	20
N > 10	19

Example

Write a MATLAB function to evaluate the value of **z = f(x,y)** for any scalar values of x and y, where the function f(x,y) is defined as :

$$f(x,y) = \begin{cases} x + y & x \geq 0 \text{ and } y \geq 0 \\ x + y^2 & x \geq 0 \text{ and } y < 0 \\ x^2 + y & x < 0 \text{ and } y \geq 0 \\ x^2 + y^2 & x < 0 \text{ and } y < 0 \end{cases}$$

X=1, y=3　　X=1, y=-3　　X=-1, y=-3

```
function z = function_of_x_and_y(x,y)
if (x>=0 & y>=0)
   z = x + y;
elseif (x>=0 & y<0)
   z = x + y^2;
elseif (x<0 & y>=0)
   z = x^2 + y;
else
   z = x^2 + y^2;
end
```

$$f(x,y)=\begin{cases} x+y & x \geq 0 \text{ and } y \geq 0 \\ x+y^2 & x \geq 0 \text{ and } y < 0 \\ x^2+y & x < 0 \text{ and } y \geq 0 \\ x^2+y^2 & x < 0 \text{ and } y < 0 \end{cases}$$

The switch-case-end structure

```
switch (control expression)
   Statement
   Statement                    block 1
case (condition 1)
   Statement
   Statement                    block 2
otherwise
   Statement
   Statement                    block 3
end
```

Example

```
function grade = score_to_grade(s);
s=input('Please enter your Matlab socre：score=');
switch fix(s/10)
        case 9
                grade='A+';
        case 8
                grade='A';
        case 7
                grade='B';
        case 6
                grade='B-';
        otherwise
                grade='C';
        end
```

fix: round towards zero

Example

Write a program in a script file that converts a quantity of energy (work) given in units of either **joule, ft-lb, cal, or eV** to the equivalent quantity in different units specified by the user.

The program asks the user to enter the quantity of energy, its current units, and the desired new units. The output is the quantity of energy in the new units.
The conversion factors are:

$$1\,\text{J} = 0.738\,\text{ft-lb} = 0.239\,\text{cal} = 6.24 \times 10^{18}\,\text{eV}.$$

Use the program to:
(a) Convert 150 J to ft-lb.
(b) Convert 2,800 cal to J.
(c) Convert 2.7 eV to cal.

Example

```
Ein=input('Enter the value of the energy (work) to be converted: ');
EinUnits=input('Enter the current units (J, ft-lb, cal, or eV):  ','s');
EoutUnits=input('Enter the new units (J, ft-lb, cal, or eV):  ','s');
error=0;
switch EinUnits
case 'J'
    EJ=Ein;
case 'ft-lb'
    EJ=Ein/0.738;
case 'cal'
    EJ=Ein/0.239;
case 'eV'
    EJ=Ein/6.24e18;
otherwise
    error=1;
end
```

Assign 0 to variable `error`.

First `switch` statement. Switch expression is a string with initial units.

Each of the four `case` statements has a value (string) that corresponds to one of the initial units, and a command that converts `Ein` to units of J. (Assign the value to `EJ`.)

Assign 1 to `error` if no match is found. Possible only if initial units were typed incorrectly.

Example

```
switch EoutUnits
case 'J'
    Eout=EJ;
case 'ft-lb'
    Eout=EJ*0.738;
case 'cal'
    Eout=EJ*0.239;
case 'eV'
    Eout=EJ*6.24e18;
otherwise
    error=1;
end
if error
    disp('ERROR current or new units are typed incorrectly.')
else
    fprintf('E = %g %s',Eout,EoutUnits)

end
```

Second `switch` statement. Switch expression is a string with new units.

Each of the four `case` statements has a value (string) that corresponds to one of the new units, and a command that converts `EJ` to the new units. (Assign the value to `Eout`.)

Assign 1 to `error` if no match is found. Possible only if new units were typed incorrectly.

`If-else-end` statement.

If `error` is true (nonzero), display an error message.

If `error` is false (zero), display converted energy.

```
>> EnergyConversion
Enter the value of the energy (work) to be converted: 2800
Enter the current units (J, ft-lb, cal, or eV):  cal
Enter the new units (J, ft-lb, cal, or eV):  J
E = 11715.5 J
```

Repetative Control Structures

for loop and while loop

1. The **for** loop, which repeatedly executes a block of statements a definite number of times.
2. The **while** loop, which repeatedly executes a block of statements as long as some condition is satisfied.

The **while** structure is most appropriate when the number of **iterations** in the loop is **not** known beforehand.

General form of the for loop

for index = first:increment:last

statement

statement

……

end

- where index is called the **loop variable** (or the **loop index**).
- The statements between the **for** statement and the **end** statement are referred to as the **body** of the loop.
- Make sure the index value is not changed inside the loop!!!!

Example: calculate sum(xx.^2) with for loop

```
XX = 11:2:46;
N_XX = length(XX);

sum_of_terms = 0;           %initialise
for i = 1:N_XX
   sum_of_terms = sum_of_terms + XX(i)^2;
end

disp('sum of the elements of array XX squared')
disp(sum_of_terms)
```

Example

```
for k=1:3:10
     x = k^2
end
>> x =
     1
x =
    16
x =
    49
x =
   100
```

```
k=1:3:10;
x=k.^2;
```

Example

(*a*) Use a `for-end` loop in a script file to calculate the sum of the first n terms of the series: $\sum_{k=1}^{n} \frac{(-1)^k k}{2^k}$. Execute the script file for $n = 4$ and $n = 20$.

(*b*) The function sin(x) can be written as a Taylor series by:

$$\sin x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

Write a user-defined function file that calculates sin(x) by using the Taylor series. For the function name and arguments use `y = Tsin(x,n)`. The input arguments are the angle `x` in degrees and `n` the number of terms in the series. Use the function to calculate sin(150°) using three and seven terms.

Example

```
n=input('Enter the number of terms ' );
S=0;
for k=1:n
    S=S+(-1)^k*k/2^k;
end
fprintf('The sum of the series is: %f',S)
```

Setting the sum to zero.

for-end loop.

In each pass one element of the series is calculated and is added to the sum of the elements from the previous passes.

```
>> Exp6_5a
Enter the number of terms 4
The sum of the series is: -0.125000
>> Exp7_5a
Enter the number of terms 20
The sum of the series is: -0.222216
```

Example

```
function y = Tsin(x,n)
% Tsin calculates the sin using Taylor formula.
% Input arguments:
% x The angle in degrees, n number of terms.

xr=x*pi/180;
y=0;
for k=0:n-1
     y=y+(-1)^k*xr^(2*k+1)/factorial(2*k+1);
end
```

Converting the angle from degrees to radians.

for-end loop.

```
>> Tsin(150,3)
ans =
      0.6523
```

Calculating sin(150^{o}) with three terms of Taylor series.

```
>> Tsin(150,7)
ans =
      0.5000
```

Calculating sin(150°) with seven terms of Taylor series.

The exact value is 0.5.

Example

A vector is given by *V* = [5, 17, –3, 8, 0, –7, 12, 15, 20, –6, 6, 4, –2, 16]. Write a program as a script file that doubles the elements that are positive and are divisible by 3 or 5, and, raises to the power of 3 the elements that are negative but greater than –5.

```
V=[5, 17, -3, 8, 0, -7, 12, 15, 20 -6, 6, 4, -2, 16];
n=length(V);
for k=1:n
    if V(k)>0 & (rem(V(k),3)==0 | rem(V(k),5)==0)
        V(k)=2*V(k);
    elseif V(k) < 0 & V(k) > -5
        V(k)=V(k)^3;
    end
end
V
```

Setting n to be equal to the number of elements in V.

for-end loop.

if-elseif-end statement.

Example

A vector is given by *V* = [5, 17, –3, 8, 0, –7, 12, 15, 20, –6, 6, 4, –2, 16]. Write a program as a script file that doubles the elements that are positive and are divisible by 3 or 5, and, raises to the power of 3 the elements that are negative but greater than –5.

```
>> V = [5, 17, -3, 8, 0, -7, 12, 15, 20, -6, 6, 4, -2, 16];

>> V(V>0 & (rem(V,3)==0 | rem(V,5)==0))=2*V(V>0 & (rem(V,3)==0 | rem(V,5)==0));

>> V(V< 0 & V>-5)=V(V< 0 & V>-5).^3;
```

General form of the while loop

while condition

statement

statement

……

end

- Statements will only be executed when the condition is satisfied.
- The statements between the while statement and the end statement are referred to as the body of the loop.
- Make sure the condition is changed through the execution of statements inside the loop

Example: calculate sum(xx.^2) with while loop

```
XX = 11:2:46;
sum_of_terms = 0;          %initialise
xx=11;
while xx<=46
   sum_of_terms = sum_of_terms + xx^2;
     xx=xx+2;
end

disp('sum of the elements of array xx squared')
disp(sum_of_terms)
```

Example: calculate sum(xx.^2) with while loop

When writing a while-end loop, the programmer has to be sure that the variable (or variables) that are in the conditional expression and are **assigned new values during the looping process** will eventually be assigned values that make the conditional expression in the while command false.

Since no one is free from making mistakes, a situation of indefinite looping can occur in spite of careful programming. If this happens, the user can stop the execution of an indefinite loop by pressing the **Ctrl + C** or **Ctrl + Break** keys.

Example

The function $f(x) = e^x$ can be represented in a Taylor series by $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$.

Write a program in a script file that determines e^x by using the Taylor series representation. The program calculates e^x by adding terms of the series and stopping when the absolute value of the term that was added last is smaller than 0.0001. Use a `while-end` loop, but limit the number of passes to 30. If in the 30th pass the value of the term that is added is not smaller than 0.0001, the program stops and displays a message that more than 30 terms are needed.

Use the program to calculate e^2, e^{-4}, and e^{21}.

The first few terms of the Taylor series are:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Example

```
x=input('Enter x ' );
n=1; an=1; S=an;
while abs(an) >= 0.0001 & n <= 30
    an=x^n/factorial(n);
    S=S+an;
    n=n+1;
end
if n >= 30
    disp('More than 30 terms are needed')
else
fprintf('exp(%f) = %f',x,S)
fprintf('\nThe number of terms used is: %i',n)
end
```

Start of the `while` loop.

Calculating the *n*th term.

Adding the *n*th term to the sum.

Counting the number of passes.

End of the `while` loop.

`if-else-end` loop.

Example

```
Enter x 2
exp(2.000000) = 7.389046
The number of terms used is: 12
>> expox
Enter x -4
exp(-4.000000) = 0.018307
The number of terms used is: 18
>> expox
Enter x 21
More than 30 terms are needed
```

Calculating exp(2).

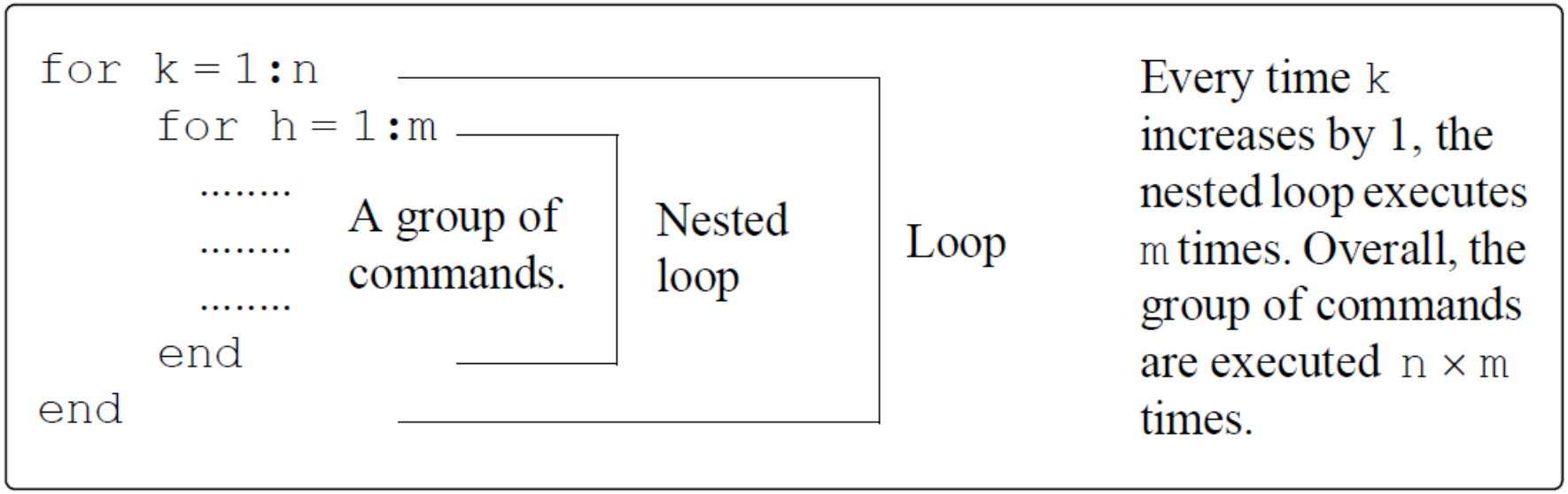
12 terms used.

Calculating exp(–4).

18 terms used.

Trying to calculate exp(21).

Nested Loops



There is no limit to the number of loops and conditional statements that can be nested.

It must be remembered, however, that each *if, case, for*, and *while* statement must have a corresponding *end* statement.

Nested Loops

```
disp('height   weight  bodymassindex')

for height = 1.50 : 0.10 : 1.90
  for weight = 50 : 10 : 90
        bodymassindex = weight/height^2;
        disp([height  weight  bodymassindex])
  end
end
```

height	weight	bodymassindex
1.5000	50.0000	22.2222
1.5000	60.0000	26.6667
1.5000	70.0000	31.1111
1.5000	80.0000	35.5556
1.5000	90.0000	40.0000
1.6000	50.0000	19.5312
1.6000	60.0000	23.4375
1.6000	70.0000	27.3437
1.6000	80.0000	31.2500
1.6000	90.0000	35.1562

Example

The program, shown below, has two loops (one nested) and a nested `if-elseif-else-end` structure. The elements in the matrix are assigned values row by row. The loop index variable of the first loop, `k`, is the address of the row, and the loop index variable of the second loop, `h`, is the address of the column.

```
n=input('Enter the number of rows ');
m=input('Enter the number of columns ');
A=[];                                          Define an empty matrix A
for k=1:n                                      Start of the first for-end loop.
   for h=1:m                                   Start of the second for-end loop.
      if k==1                                  Start of the conditional statement.
          A(k,h)=h;                            Assign values to the elements of the first row.
      elseif h==1
          A(k,h)=k;                            Assign values to the elements of the first column.
      else
          A(k,h)=A(k,h-1)+A(k-1,h);            Assign values to other elements.
      end                                      end of the if statement.
   end                                         end of the nested for-end loop.
end                                            end of the first for-end loop.
A
```

Example

```
>> Chap6_exp8
Enter the number of rows 4
Enter the number of columns 5
```

```
A =
     1     2     3     4     5
     2     4     7    11    16
     3     7    14    25    41
     4    11    25    50    91
```

break and continue commands

The *break* command:

- When inside a loop (for or while), the break command terminates the execution of the loop (the whole loop, not just the last pass).
- If the break command is inside a nested loop, only the nested loop is terminated.
- When a break command appears outside a loop, e.g. in a script or function file, it terminates the execution of the file.

break and continue commands

The *continue* command:

- The continue command can be used inside a loop (for or while) to stop the present pass and start the next pass in the looping process.
- The continue command is usually a part of a conditional statement. When MATLAB reaches the continue command, it does not execute the remaining commands in the loop, but skips to the end command of the loop and then starts a new pass.

Example

Six singers—John, Mary, Tracy, Mike, Katie, and David—are performing in a competition. Write a MATLAB program that generates a list of a random order in which the singers will perform.

Example

```
clear, clc
n=6;
L(1)=randi(n);
for p=2:n
    L(p)=randi(n);
    r=0;
    while r==0
      r=1;
      for k=1:p-1
          if L(k)==L(p)
            L(p)=randi(n);
            r=0;
            break
          end
      end
    end
end
```

Assign the first integer to `L(1)`.

Assign the next integer to `L(p)`.

Set `r` to zero.

See explanation below.

Set `r` to 1.

`for` loop compares the integer assigned to `L(p)` to the integers that have been assigned to previous elements.

If a match if found, a new integer is assigned to `L(p)` and `r` is set to zero.

The nested `for` loop is stopped. The program goes back to the `while` loop. Since `r = 0` the nested loop inside the `while` loop starts again and checks if the new integer that is assigned to `L(p)` is equal to an integer that is already in the vector `L`.

Example

```
for i=1:n
    switch L(i)
    case 1
        disp('John')
    case 2
        disp('Mary')
    case 3
        disp('Tracy')
    case 4
        disp('Mike')
    case 5
        disp('Katie')
    case 6
        disp('David')
    end
end
```

The `switch-case` statement lists the names according to the values of the integers in the elements of `L`.

```
Katie
Tracy
David
Mary
John
Mike
```

Example :Withdrawing from a retirement account

Problem:

A person in retirement is depositing $300,000 in a saving account that pays 5% interest per year. The person plans to withdraw money from the account once a year. He starts by withdrawing $25,000 after the first year, and in future years he increases the amount he withdraws according to the inflation rate.

For example, if the inflation rate is 3%, he withdraws $25,750 after the second year. Calculate the number of years the money in the account will last assuming a constant yearly inflation rate of 2%. Make a plot that shows the yearly withdrawals and the balanceof the account over the years.

Example :Withdrawing from a retirement account

Solution:

The problem is solved by using a loop (a **while** loop since the number of passes is not known before the loop starts). In each pass the amount to be withdrawn and the account balance are calculated.

The looping continues as long as the account balance is larger than or equal to the amount to be withdrawn.

The following is a program in a script file that solves the problem. In the program, year is a vector in which each element is a year number, **W** is a vector with the amount withdrawn each year, and **AB** is a vector with the account balance each year.

Example :Withdrawing from a retirement account

```
rate=0.05; inf=0.02;
clear W AB year
year(1)=0;
W(1)=0;
AB(1)=300000;
Wnext=25000;
ABnext=300000*(1 + rate);
n=2;
    while ABnext >= Wnext
        year(n)=n-1;
        W(n)=Wnext;
        AB(n)=ABnext-W(n);
        ABnext=AB(n)*(1+rate);
        Wnext=W(n)*(1+inf);
        n=n+1;
    end
fprintf('The money will last for %f years',year(n-1))
bar(year,[AB' W'],2.0)
```

First element is year 0.

Initial withdrawal amount.

Initial account balance.

The amount to be withdrawn after a year.

The account balance after a year.

`while` checks if the next balance is larger than the next withdrawal.

Amount withdrawn in year $n-1$.

Account balance in year $n-1$ after withdrawal.

The account balance after additional year.

The amount to be withdrawn after an additional year.

Example :Flight of a model rocket

Problem:

The flight of a model rocket can be modeled as follows. During the first **0.15s** the rocket is propelled upward by the rocket engine with a force of **16 N**. The rocket then flies up while slowing down under the force of gravity. After it reaches the apex, the rocket starts to fall back down.

When its downward velocity reaches **20 m/s** a parachute opens (assumed to open instantly), and the rocket continues to drop at a constant speed of **20 m/s** until it hits the ground.

Write a program that calculates and plots the speed and altitudeof the rocket as a function of time during the flight.

Example :Flight of a model rocket

Solution:

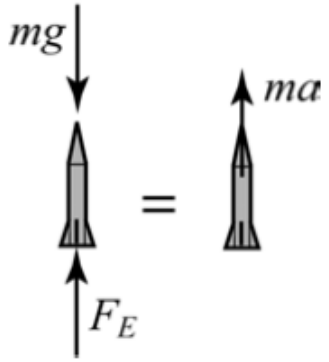
The rocket is assumed to be a particle that moves along a straight line in the vertical plane. For motion with constant acceleration along a straight line, the velocity and position as a function of time are given by:

$$v(t) = v_0 + at \quad \text{and} \quad s(t) = s_0 + v_0 t + \frac{1}{2}at^2$$

where v_0 and s_0 are the initial velocity and position, respectively. In the computer program the flight of the rocket is divided into three segments. Each segment is calculated in a while loop. In every pass the time increases by an increment.

Segment 1: The first 0.15s when the rocket engine is on. During this period, the rocket moves up with a constant acceleration.

$$+\uparrow \Sigma F = F_E - mg = ma$$



Solving the equation for the acceleration gives:

$$a = \frac{F_E - mg}{m} \qquad v(t) = 0 + at \quad \text{and} \quad h(t) = 0 + 0 + \frac{1}{2}at^2$$

Example :Flight of a model rocket

Solution:

Segment 2: The motion from when the engine stops until the parachute opens. In this segment the rocket moves with a constant deceleration g.

$$v(t) = v_1 - g(t - t_1) \quad \text{and} \quad h(t) = h_1 + v_1(t - t_1) - \frac{1}{2}g(t - t_1)^2$$

The time and height at the end of this segment are t_2 and h_2 .

Segment 3: The motion from when the parachute opens until the rocket hits the ground. In this segment the rocket moves with constant velocity (zero acceleration). The height as a function of time is given by

$$h(t) = h_2 - v_{chute}(t - t_2)$$

Where V_{chute} is the constant velocity after the parachute opens.

Example :Flight of a model rocket

```
m=0.05; g=9.81; tEngine=0.15; Force=16; vChute=-20; Dt=0.01;
clear t v h
n=1;
t(n)=0; v(n)=0; h(n)=0;
% Segment 1
a1=(Force-m*g)/m;
while t(n) < tEngine & n < 50000
    n=n+1;
    t(n)=t(n-1)+Dt;
    v(n)=a1*t(n);
    h(n)=0.5*a1*t(n)^2;
end
v1=v(n); h1=h(n); t1=t(n);
% Segment 2
while v(n) >= vChute & n < 50000
    n=n+1;
    t(n)=t(n-1)+Dt;
    v(n)=v1-g*(t(n)-t1);
```

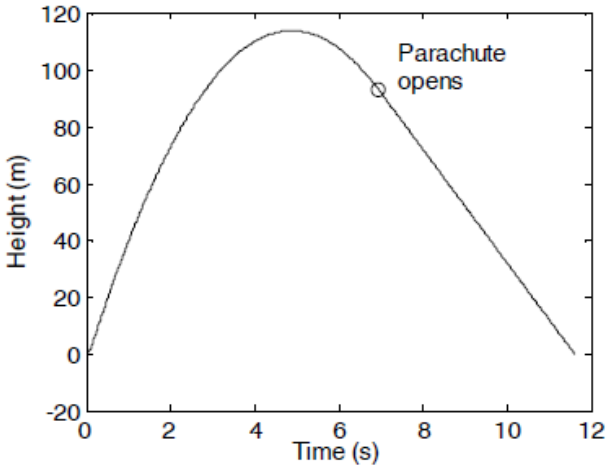
The first `while` loop.

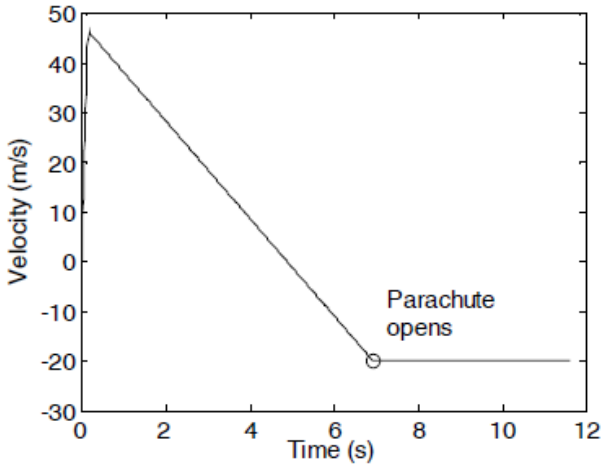
The second `while` loop.

Example :Flight of a model rocket

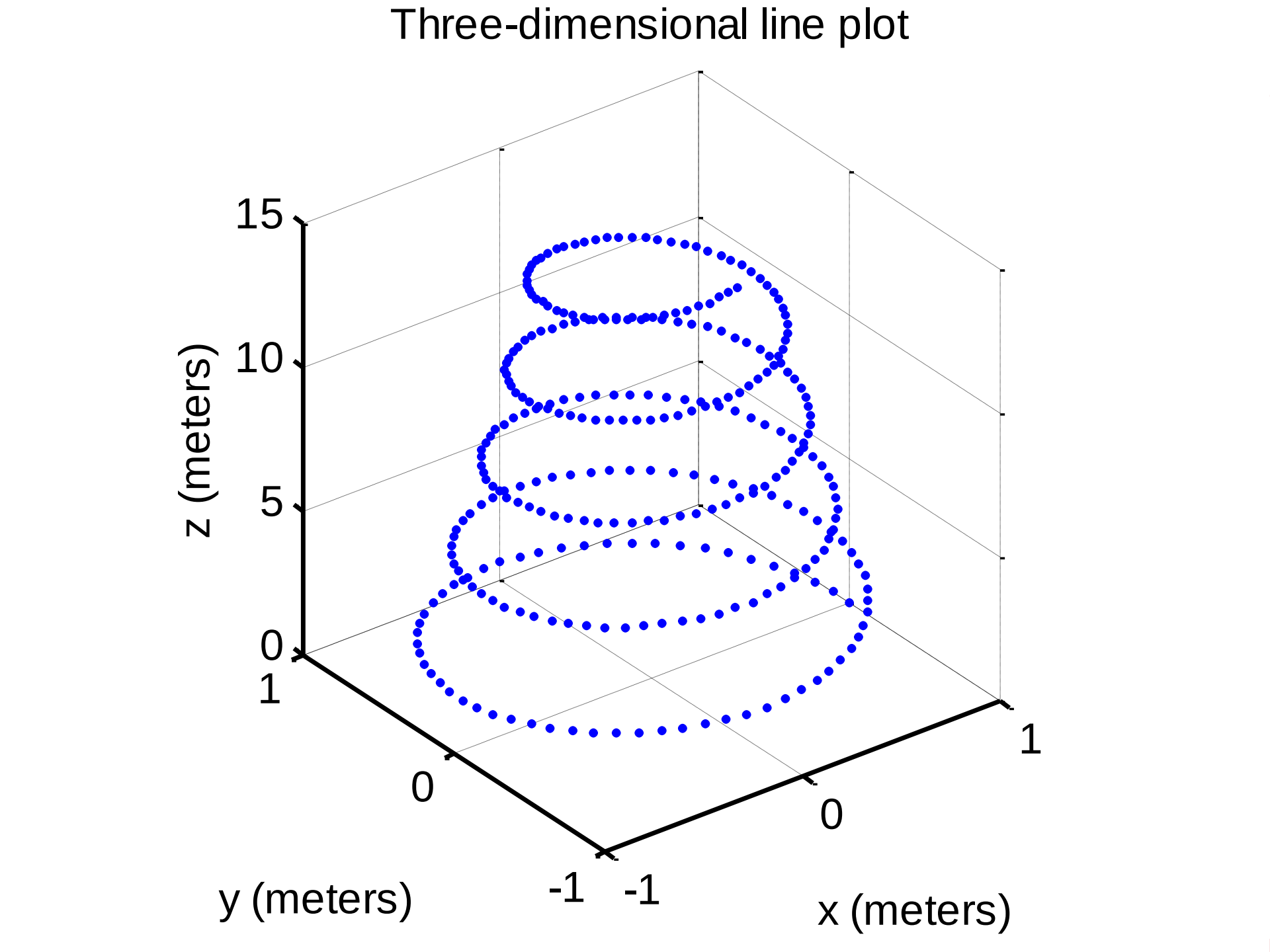
```
    h(n)=h1+v1*(t(n)-t1)-0.5*g*(t(n)-t1)^2;
end
v2=v(n); h2=h(n); t2=t(n);
% Segment 3
while h(n) > 0 & n < 50000
   n=n+1;
    t(n)=t(n-1)+Dt;
    v(n)=vChute;
    h(n)=h2+vChute*(t(n)-t2);
end
subplot(1,2,1)
plot(t,h,t2,h2,'o')
subplot(1,2,2)
plot(t,v,t2,v2,'o')
```

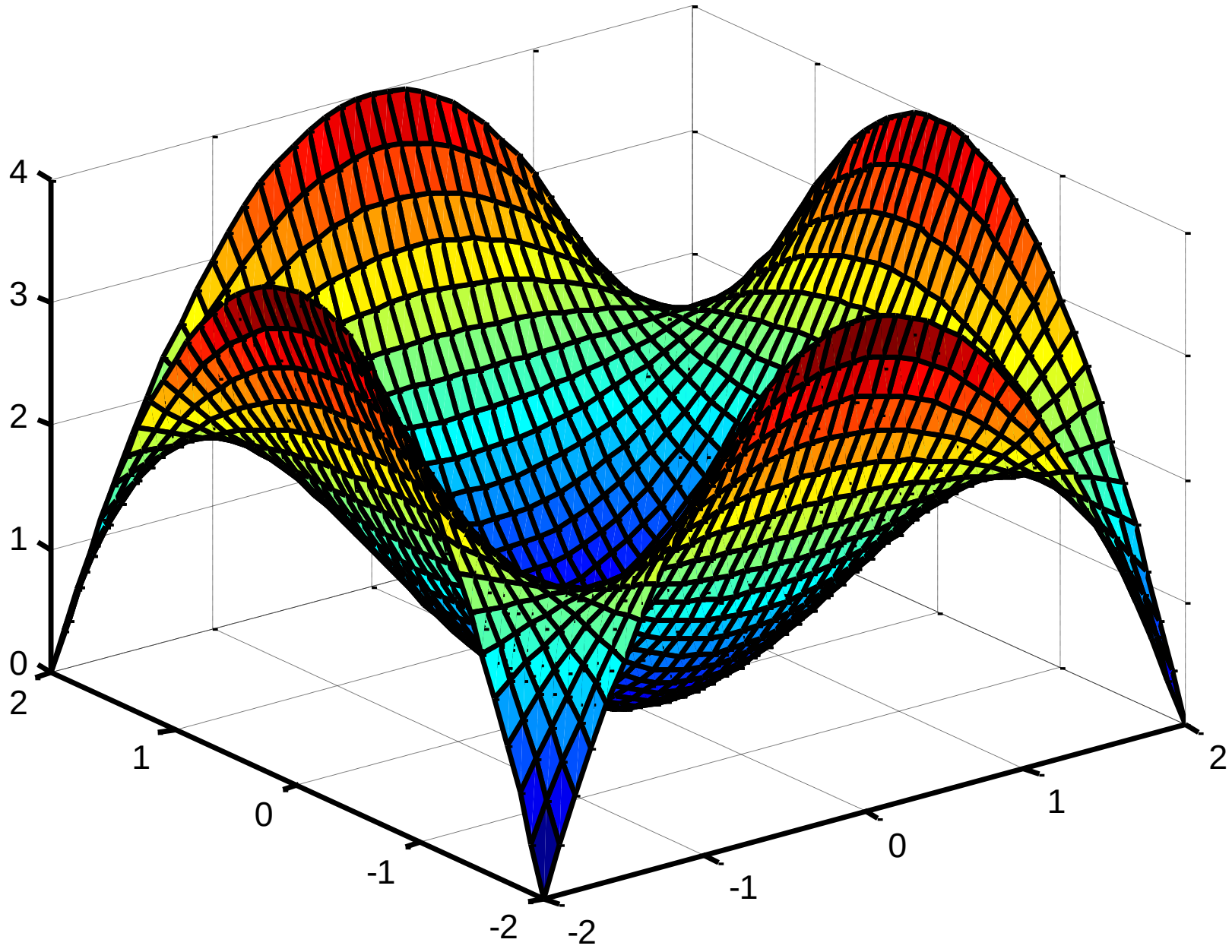
The third `while` loop.

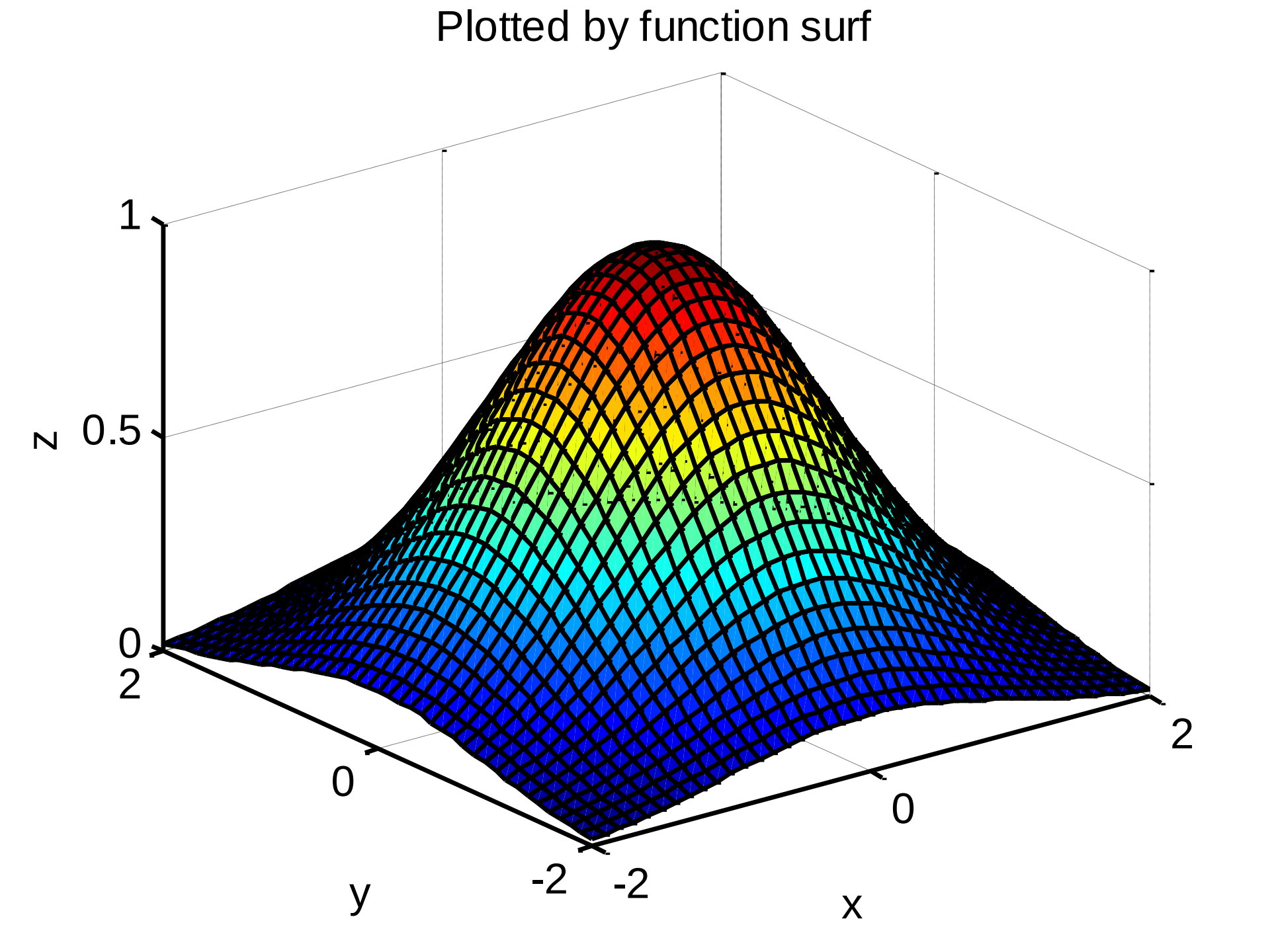


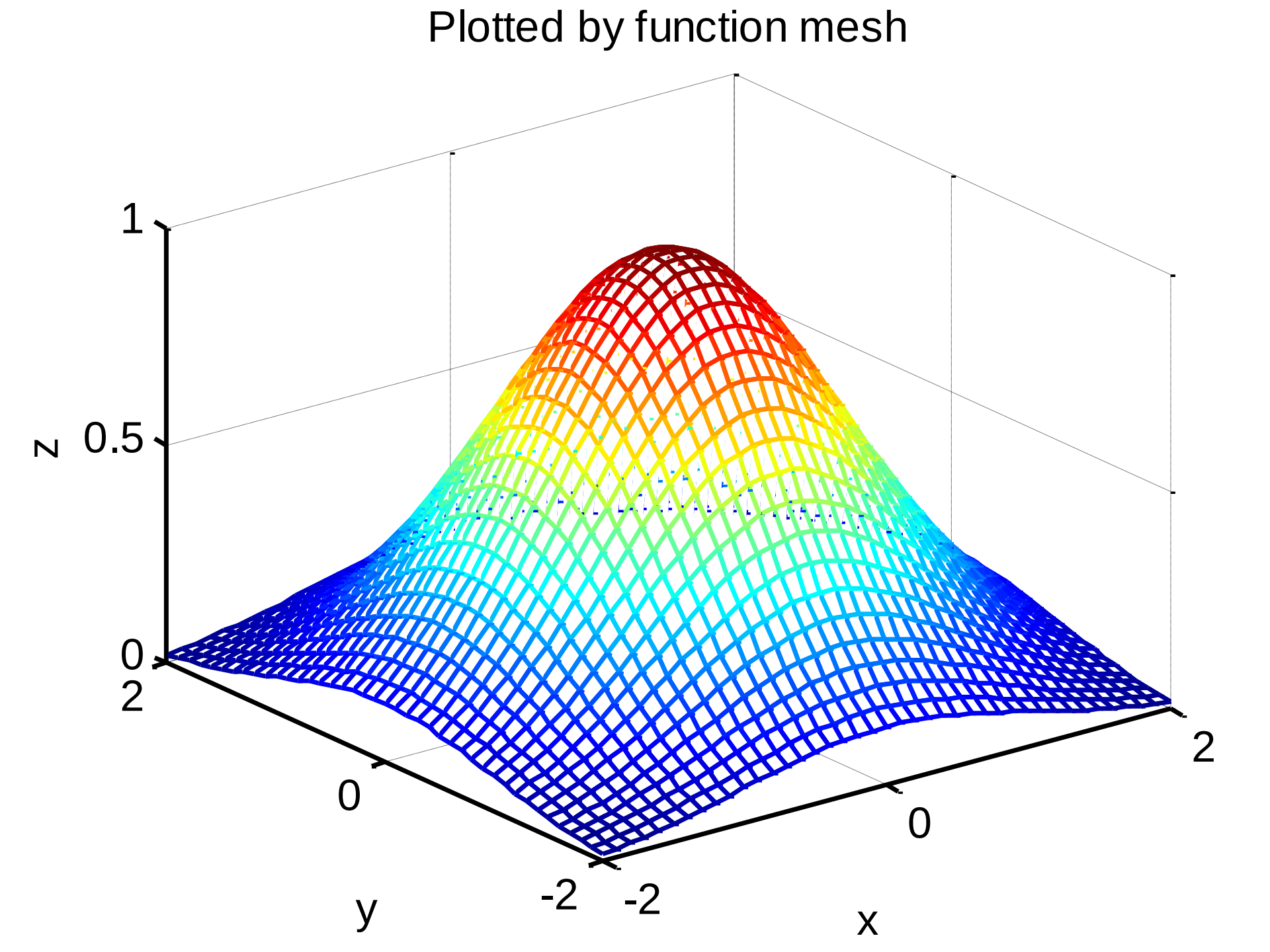


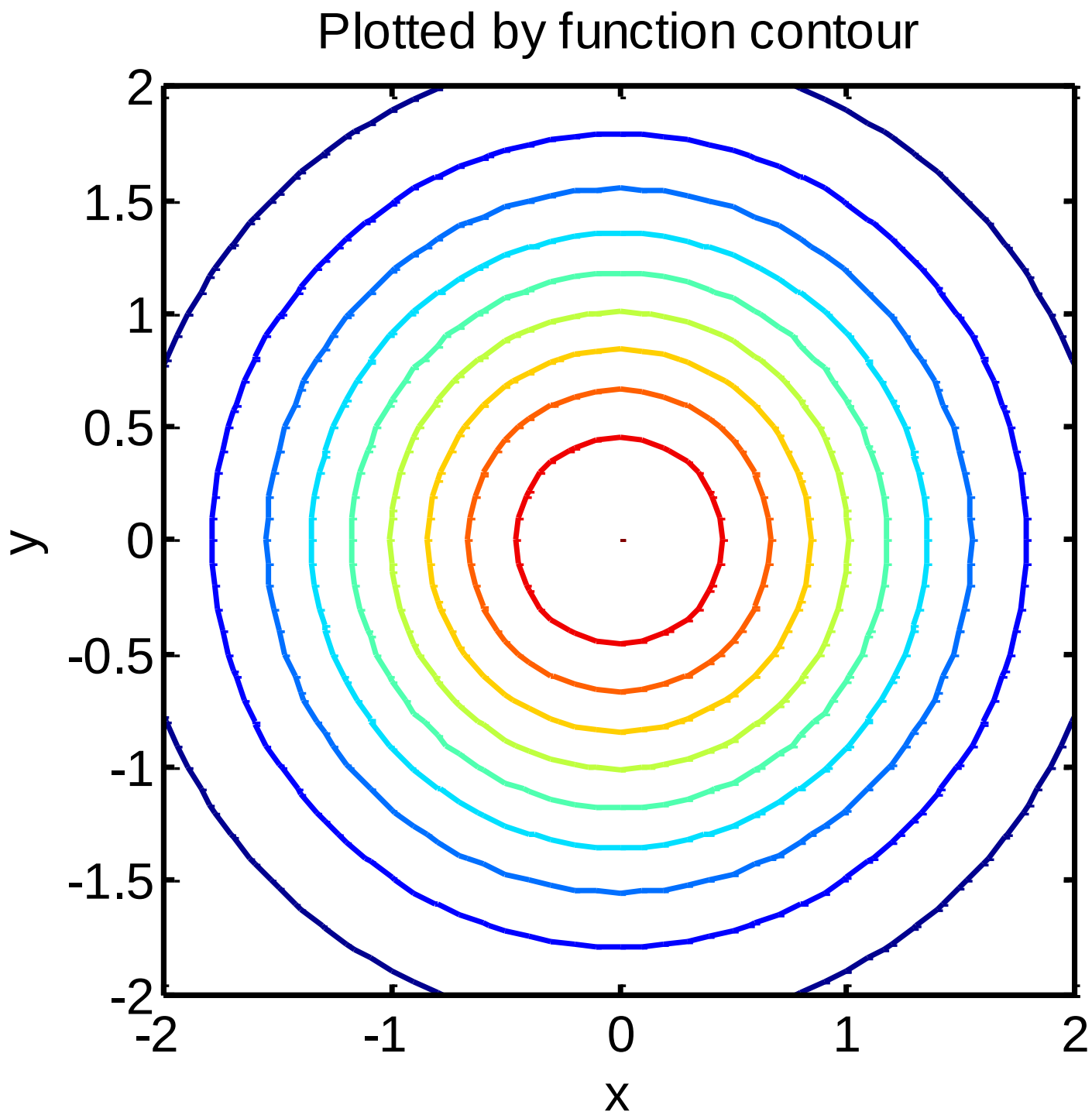
GRAPHICS

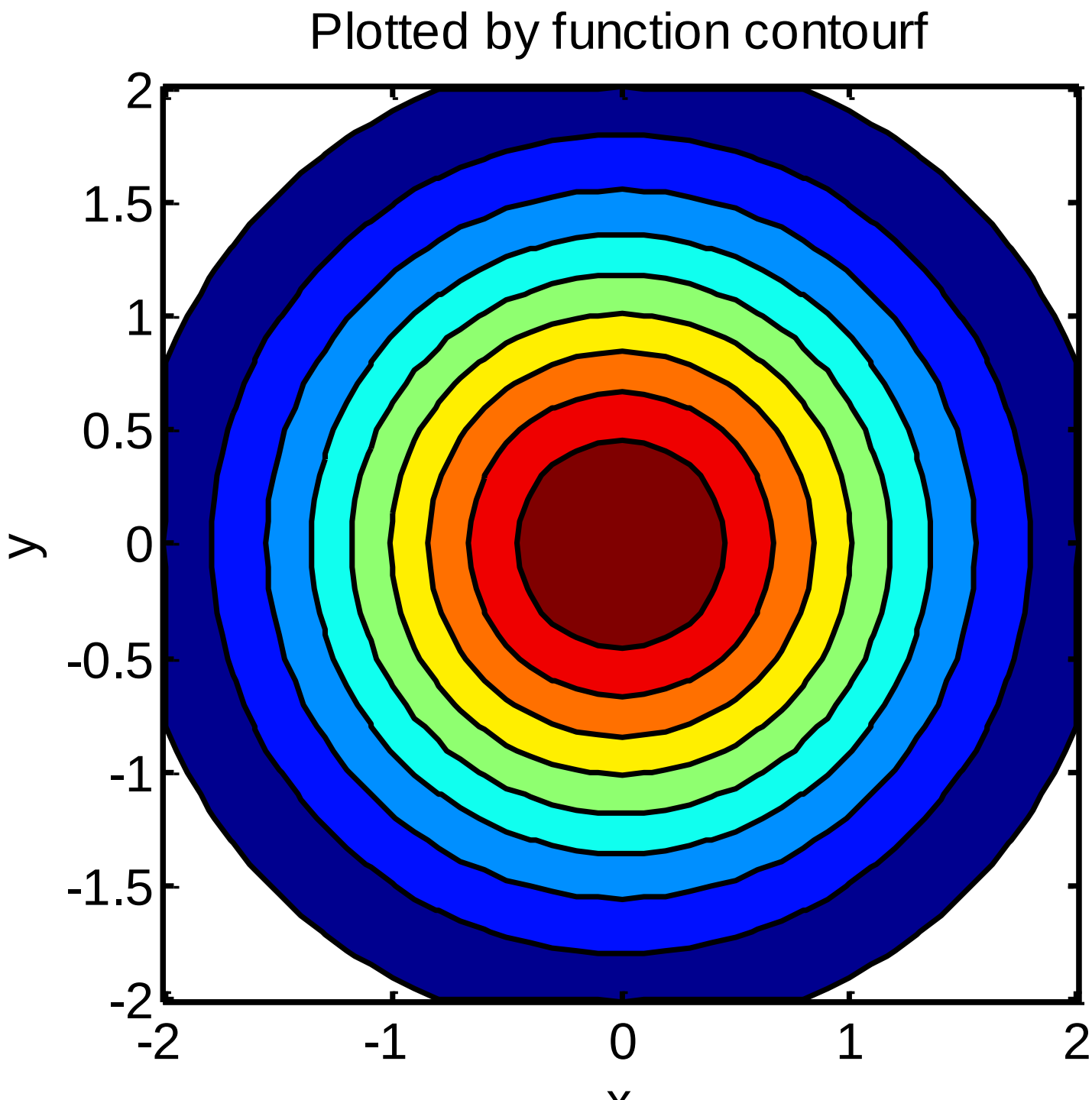

Three-dimensional line plot
15
10
5
0
z (meters)
1
0
-1
y (meters)
-1
0
1
x (meters)


4
3
2
1
0
2
1
0
-1
-2
-2
-1
0
1
2


Plotted by function surf
1
0.5
z
0
2
0
y
-2
-2
0
x
2


Plotted by function mesh
1
0.5
0
z
2
0
-2
y
-2
0
2
x

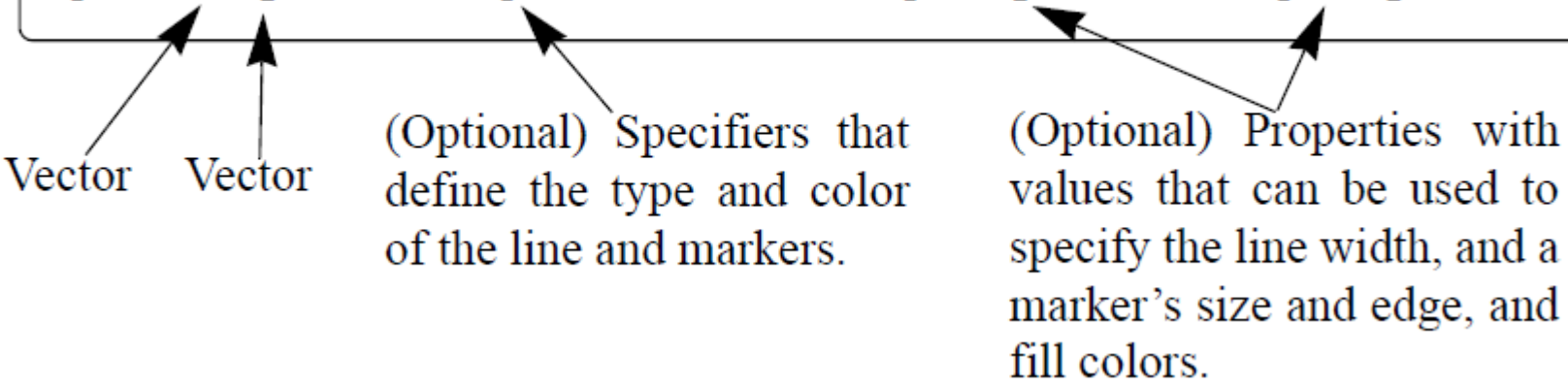

Plotted by function contour
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
y
-2
-1
0
1
2
x


Plotted by function contourf
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
y
-2
-1
0
1
2
x

2-D PLOTS

Command plot

```
plot(x,y,'line specifiers','PropertyName',PropertyValue)
```



Line Style	Specifier
solid (default)	-
dashed	--

Line Style	Specifier
dotted	:
dash-dot	-.

Line Color	Specifier
red	r
green	g
blue	b
cyan	c

Line Color	Specifier
magenta	m
yellow	y
black	k
white	w

Command plot

```
plot(x,y,'line specifiers','PropertyName',PropertyValue)
```

Vector Vector

(Optional) Specifiers that define the type and color of the line and markers.

(Optional) Properties with values that can be used to specify the line width, and a marker's size and edge, and fill colors.

Marker Type	Specifier		Marker Type	Specifier
plus sign	+		square	s
circle	o		diamond	d
asterisk	*		five-pointed star	p
point	.		six-pointed star	h
cross	x		triangle (pointed left)	<
triangle (pointed up)	^		triangle (pointed right)	>
triangle (pointed down)	v			

Command plot

Property name	Description	Possible property values
`LineWidth` (or `linewidth`)	Specifies the width of the line.	A number in units of points (default 0.5).
`MarkerSize` (or `markersize`)	Specifies the size of the marker.	A number in units of points.
`MarkerEdgeColor` (or `markeredgecolor`)	Specifies the color of the marker, or the color of the edge line for filled markers.	Color specifiers from the table above, typed as a string.
`MarkerFaceColor` (or `markerfacecolor`)	Specifies the color of the filling for filled markers.	Color specifiers from the table above, typed as a string.

Command plot

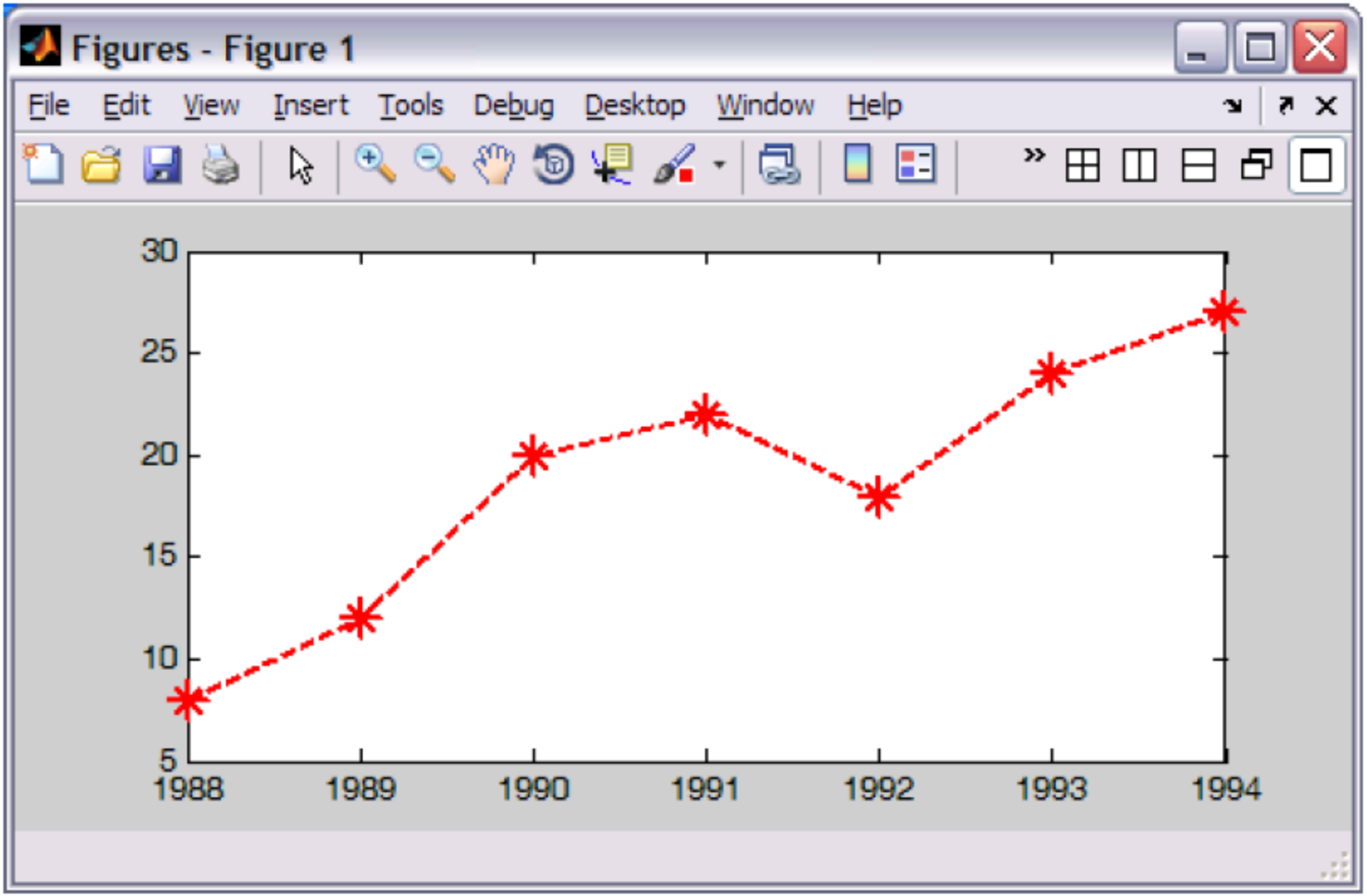
Year	1988	1989	1990	1991	1992	1993	1994
Sales (millions)	8	12	20	22	18	24	27

```
>> yr=[1988:1:1994];
>> sle=[8  12  20  22  18  24  27];
>> plot(yr,sle,'--r*','linewidth',2,'markersize',12)
>>
```

Line Specifiers: dashed red line and asterisk marker.

Property Name and Property Value: the line width is 2 points and the marker size is 12 points.

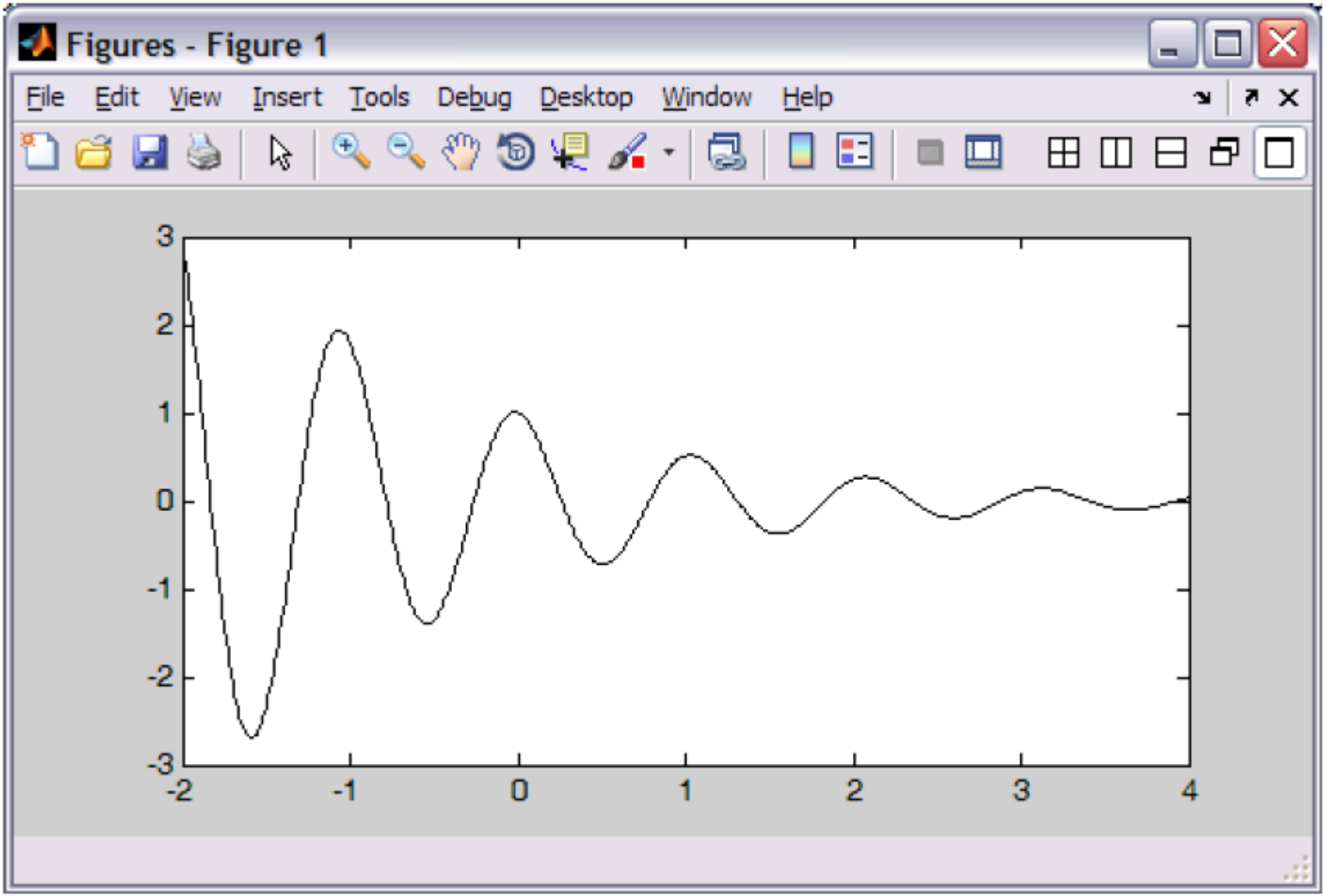
Command plot



Command plot

```
% A script file that creates a plot of
% the function: 3.5.^(-0.5*x).*cos(6x)
x=[-2:0.01:4];
y=3.5.^(-0.5*x).*cos(6*x);
plot(x,y)
```

Command plot



Command hold on, hold off

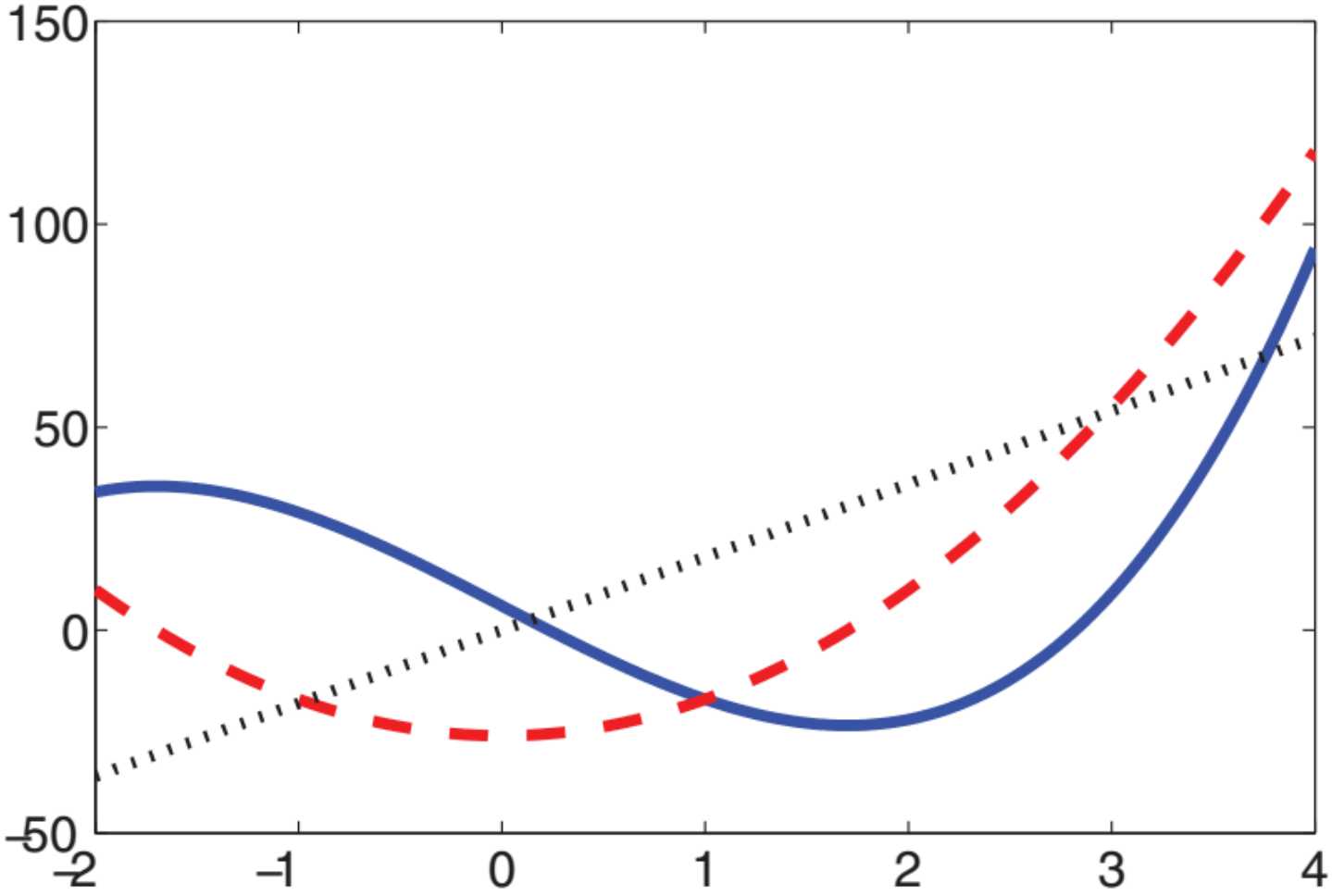
- The hold on command keeps the Figure Window with the first plot open, including the axis properties and formatting. Additional graphs can be added with plot commands that are typed next.
- The hold off command stops this process. It returns MATLAB to the default mode, in which the plot command erases the previous plot and resets the axis properties.

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'-b')
hold on
plot(x,yd,'--r')
plot(x,ydd,':k')
hold off
```

The first graph is created.

Two more graphs are added to the figure.

Command hold on, hold off



Command line

- The major difference between the plot and line commands is that the plot command starts a new plot every time it is executed,
- While the line command adds lines to a plot that already exists.

```
line(x,y,'PropertyName',PropertyValue)
```

(Optional) Properties with values that can be used to specify the line style, color, and width, marker type, size, and edge and fill colors.

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'LineStyle','-','color','b')
line(x,yd,'LineStyle','--','color','r')
line(x,ydd,'linestyle',':','color','k')
```

Formatting plot

The xlabel **and** ylabel **commands:**

```
xlabel('text as string')
ylabel('text as string')
```

The title command:

```
title('text as string')
```

The text command:

```
text(x,y,'text as string')
gtext('text as string')
```

- The text command places the text in the figure such that the first character is positioned at the point with the coordinates x, y (according to the axes of the figure).
- The gtext command places the text at a position specified by the user.

Formatting plot

The legend **command:**

```
legend('string1','string2', ..... ,pos)
```

`pos = -1`	Places the legend outside the axes boundaries on the right side.
`pos = 0`	Places the legend inside the axes boundaries in a location that interferes the least with the graphs.
`pos = 1`	Places the legend at the upper-right corner of the plot (default).
`pos = 2`	Places the legend at the upper-left corner of the plot.
`pos = 3`	Places the legend at the lower-left corner of the plot.
`pos = 4`	Places the legend at the lower-right corner of the plot.

Formatting plot

Formatting the text within the xlabel, ylabel, title, text **and** legend **commands:**

Modifier	Effect
\bf	bold font
\it	italic style
\rm	normal font

Modifier	Effect
\fontname{fontname}	specified font is used
\fontsize{fontsize}	specified font size is used

Subscript _{}
Superscript ^{}

Characters in the string	Greek letter
\alpha	α
\beta	β
\gamma	γ
\theta	θ
\pi	π
\sigma	σ

Characters in the string	Greek letter
\Phi	Φ
\Delta	Δ
\Gamma	Γ
\Lambda	Λ
\Omega	Ω
\Sigma	Σ

Formatting plot

Formatting the text within the xlabel, ylabel, title, text **and** legend **commands:**

```
text(x,y,'text as string',PropertyName,PropertyValue)
```

Property name	Description	Possible property values
`Rotation`	Specifies the orientation of the text.	Scalar (degrees) Default: 0
`FontAngle`	Specifies italic or normal style characters.	`normal`, `italic` Default: `normal`
`FontName`	Specifies the font for the text.	Font name that is available in the system.
`FontSize`	Specifies the size of the font.	Scalar (points) Default: 10
`FontWeight`	Specifies the weight of the characters.	`light`, `normal`, `bold` Default: `normal`
`Color`	Specifies the color of the text.	Color specifiers (see Section 5.1).
`Background-Color`	Specifies the background color (rectangular area).	Color specifiers (see Section 5.1).
`EdgeColor`	Specifies the color of the edge of a rectangular box around the text.	Color specifiers (see Section 5.1). Default: none.
`LineWidth`	Specifies the width of the edge of a rectangular box around the text.	Scalar (points) Default: 0.5

Formatting plot

The axis command:

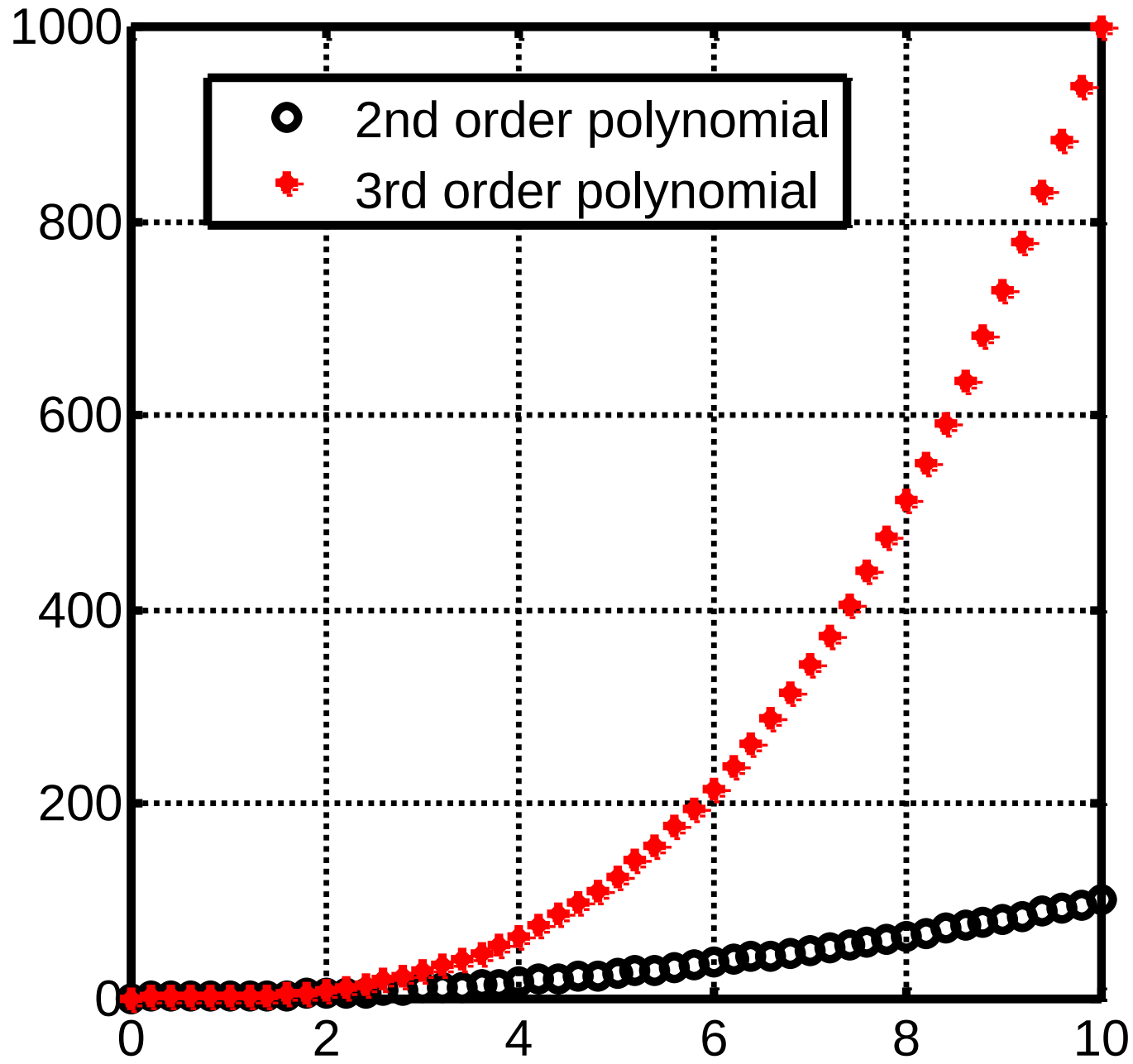
- When the plot(x,y) command is executed, MATLAB creates axes with limits that are based on the minimum and maximum values of the elements of x and y.
- The axis command can be used to change the range and the appearance of the axes.

`axis([xmin,xmax,ymin,ymax])`	Sets the limits of both the *x* and *y* axes (`xmin`, `xmax`, `ymin`, and `ymax` are numbers).
`axis equal`	Sets the same scale for both axes.
`axis square`	Sets the axes region to be square.
`axis tight`	Sets the axis limits to the range of the data.

Example

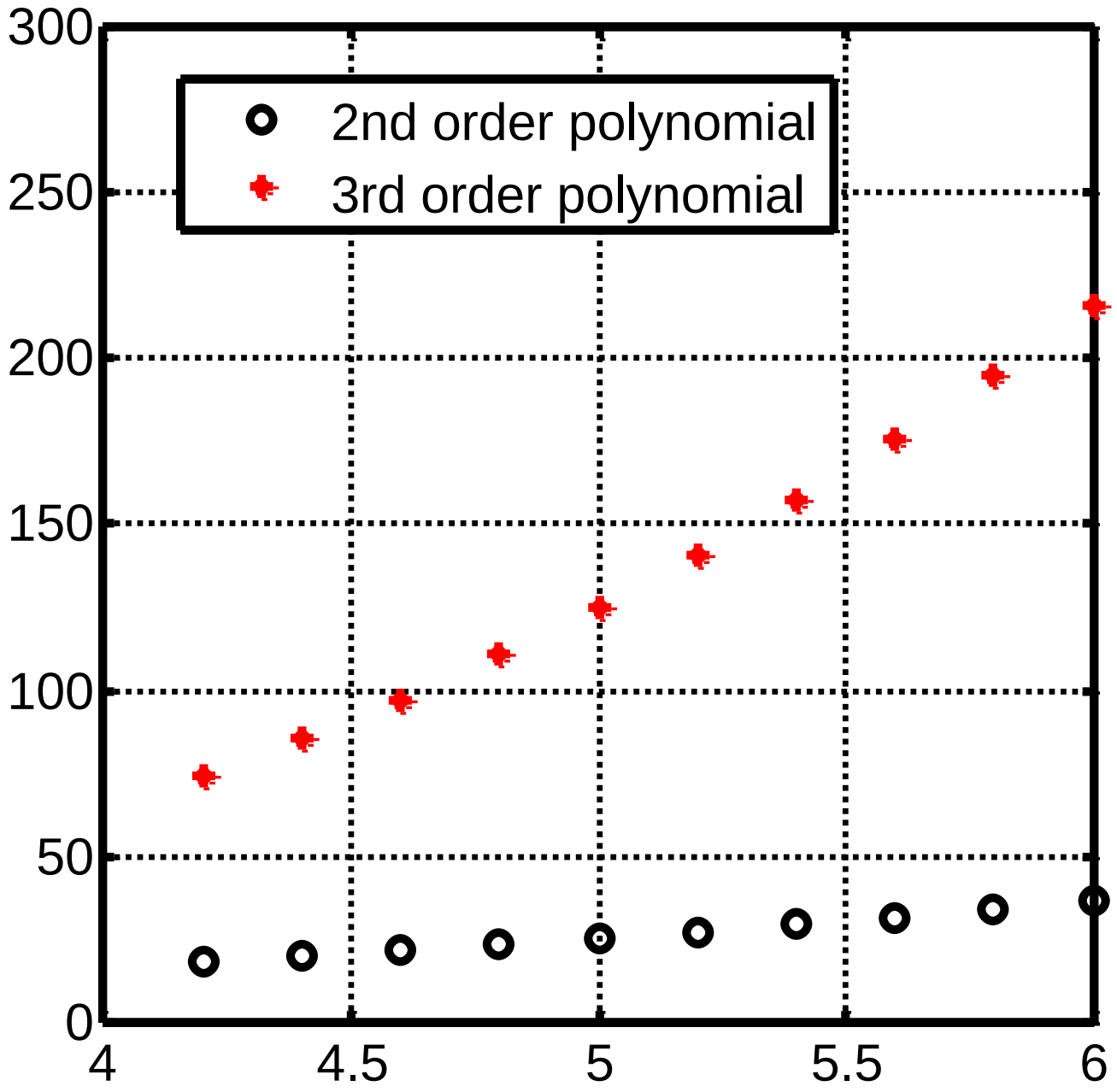
plot(x1,y1,’style_option_1’, x2,y2,’style_option2’,...)

```
x = 0:0.2:10;
y1 = x.^2;
y2 = x.^3;
figure, plot(x,y1,'ko',x,y2,'r');
legend('2nd order polynomial', '3rd order polynomial')


figure, axes('FontName','Times New Roman','FontSize',16);
plot(x,y1,'ko',x,y2,'r*','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize',10),
title('Polynomial lines','FontName','Times New Roman','FontSize',20);
xlabel('X','FontName','Times New Roman','FontSize',16);
ylabel('Y','FontName','Times New Roman','FontSize',16);
```


2nd order polynomial
3rd order polynomial
1000
800
600
400
200
0
0
2
4
6
8
10

Example

```
axis([x_min x_max y_min ymax])
x = 0:0.2:10;
y1 = x.^2;
y2 = x.^3;
figure, plot(x,y1,'ko',x,y2,'r*'),
legend('2nd order polynomial', '3rd order polynomial')
axis([4 6 0 300])
```


300
250
200
150
100
50
0
4
4.5
5
5.5
6
2nd order polynomial
3rd order polynomial

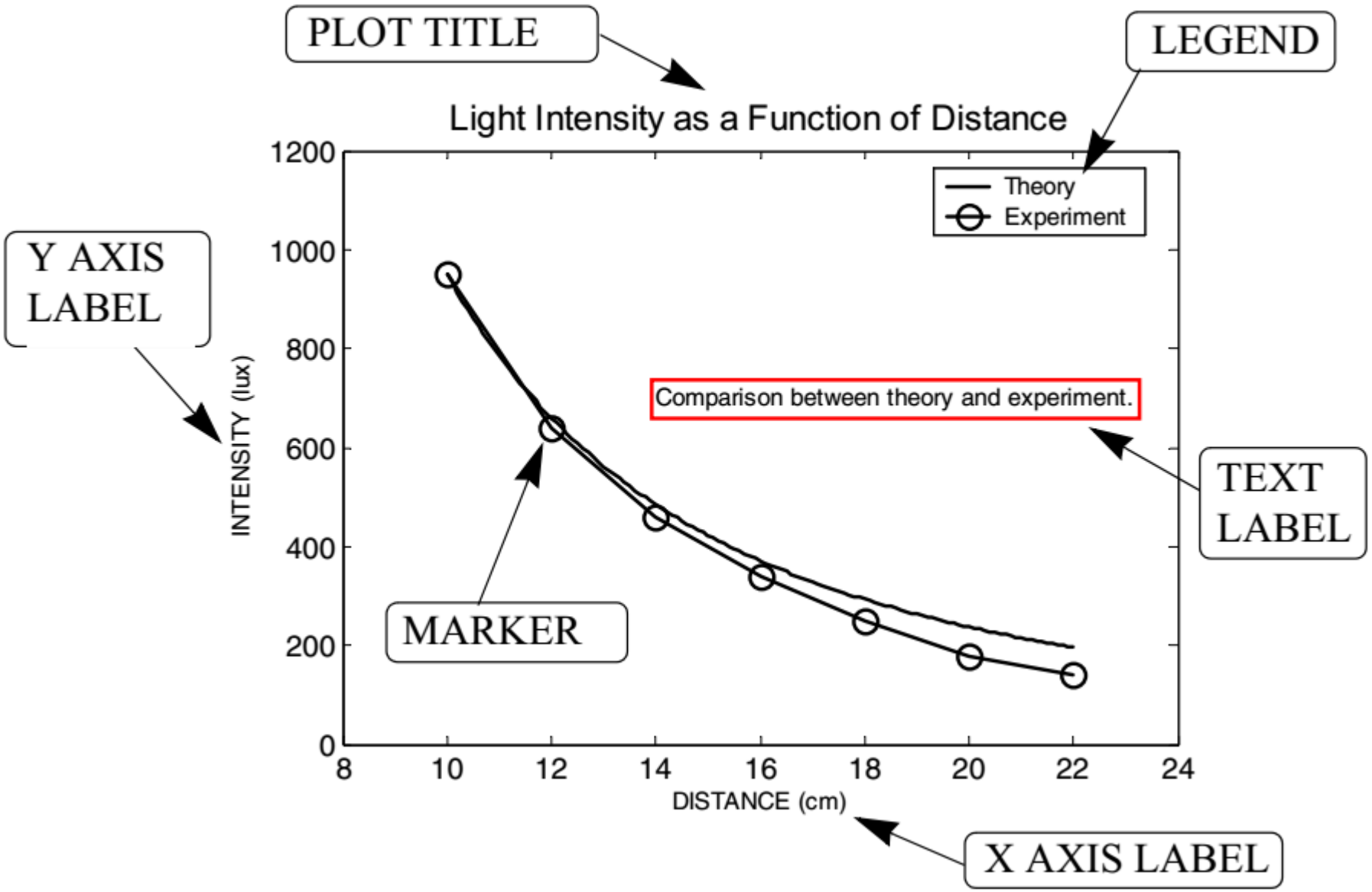
Example

```
x=[10:0.1:22];
y=95000./x.^2;
xd=[10:2:22];
yd=[950  640  460  340  250  180  140];
plot(x,y,'-','LineWidth',1.0)
xlabel('DISTANCE (cm)')
ylabel('INTENSITY (lux)')
title('\fontname{Arial}Light Intensity as a Function of Distance','FontSize',14)
axis([8 24 0 1200])
text(14,700,'Comparison between theory and experiment.','EdgeColor','r','LineWidth',2)
hold on
plot(xd,yd,'ro--','linewidth',1.0,'markersize',10)
legend('Theory','Experiment',0)
hold off
```

Formatting text inside the `title` command.

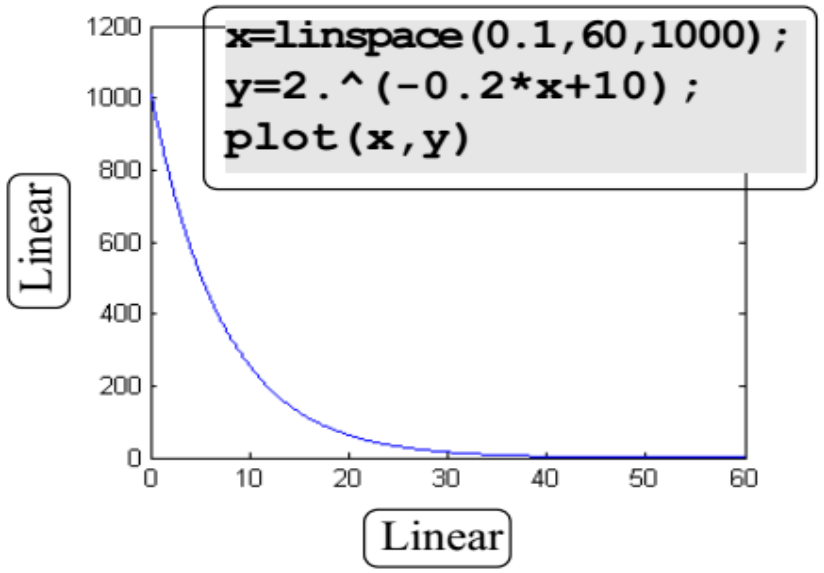
Formatting text inside the `text` command.

Example

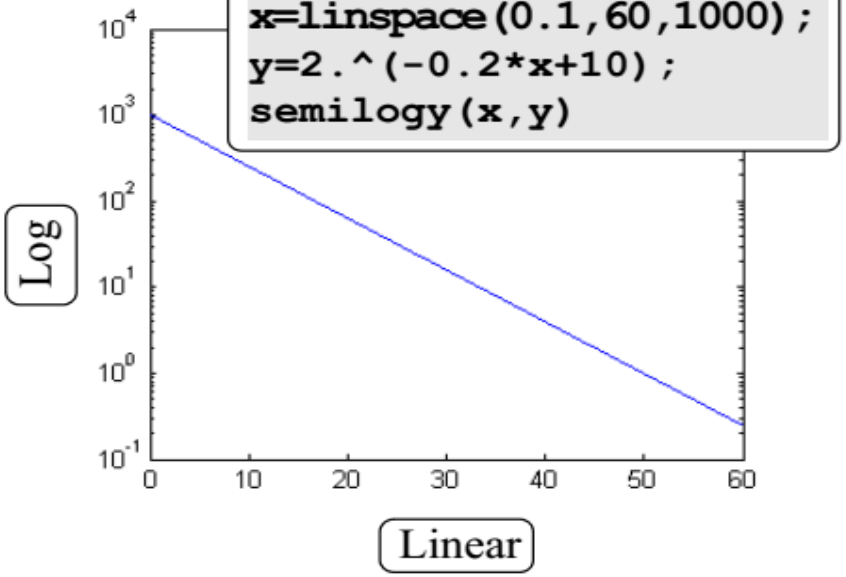

PLOT TITLE
LEGEND
Light Intensity as a Function of Distance
Theory
Experiment
Y AXIS LABEL
INTENSITY (lux)
1200
1000
800
600
400
200
0
Comparison between theory and experiment.
TEXT LABEL
MARKER
8
10
12
14
16
18
20
22
24
DISTANCE (cm)
X AXIS LABEL

Plots with Logarithmic Axes

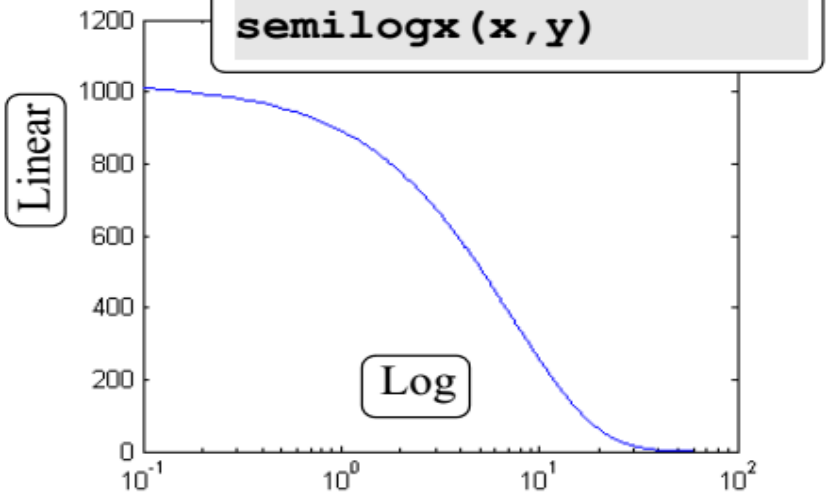
```
x=linspace(0.1,60,1000);
y=2.^(-0.2*x+10);
plot(x,y)
```



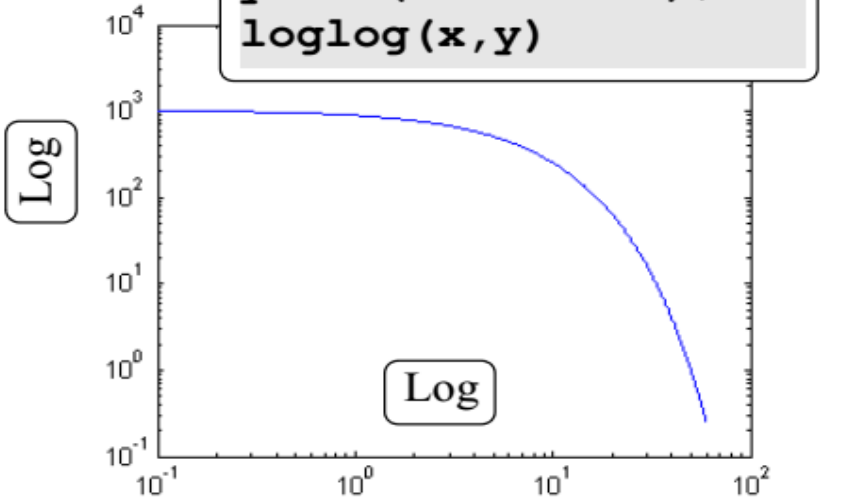

```
x=linspace(0.1,60,1000);
y=2.^(-0.2*x+10);
semilogy(x,y)
```




```
x=linspace(0.1,60,1000);
y=2.^(-0.2*x+10);
semilogx(x,y)
```

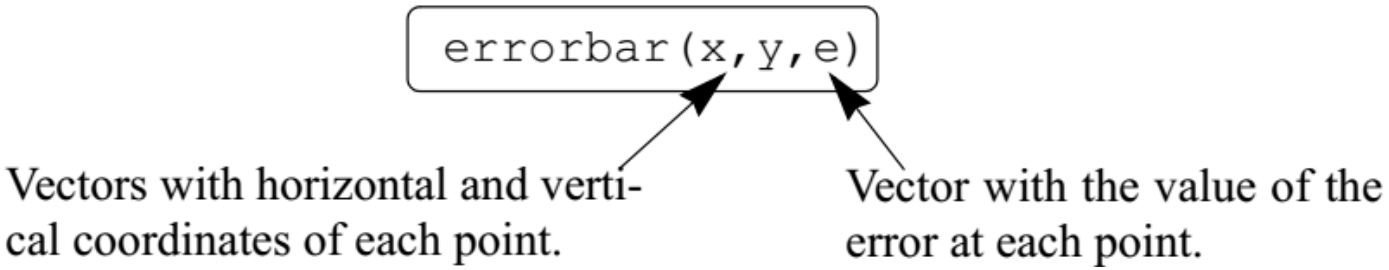



```
x=linspace(0.1,60,1000);
y=2.^(-0.2*x+10);
loglog(x,y)
```



Plots with Error Bars

- Plotting data that displays the error, or uncertainty

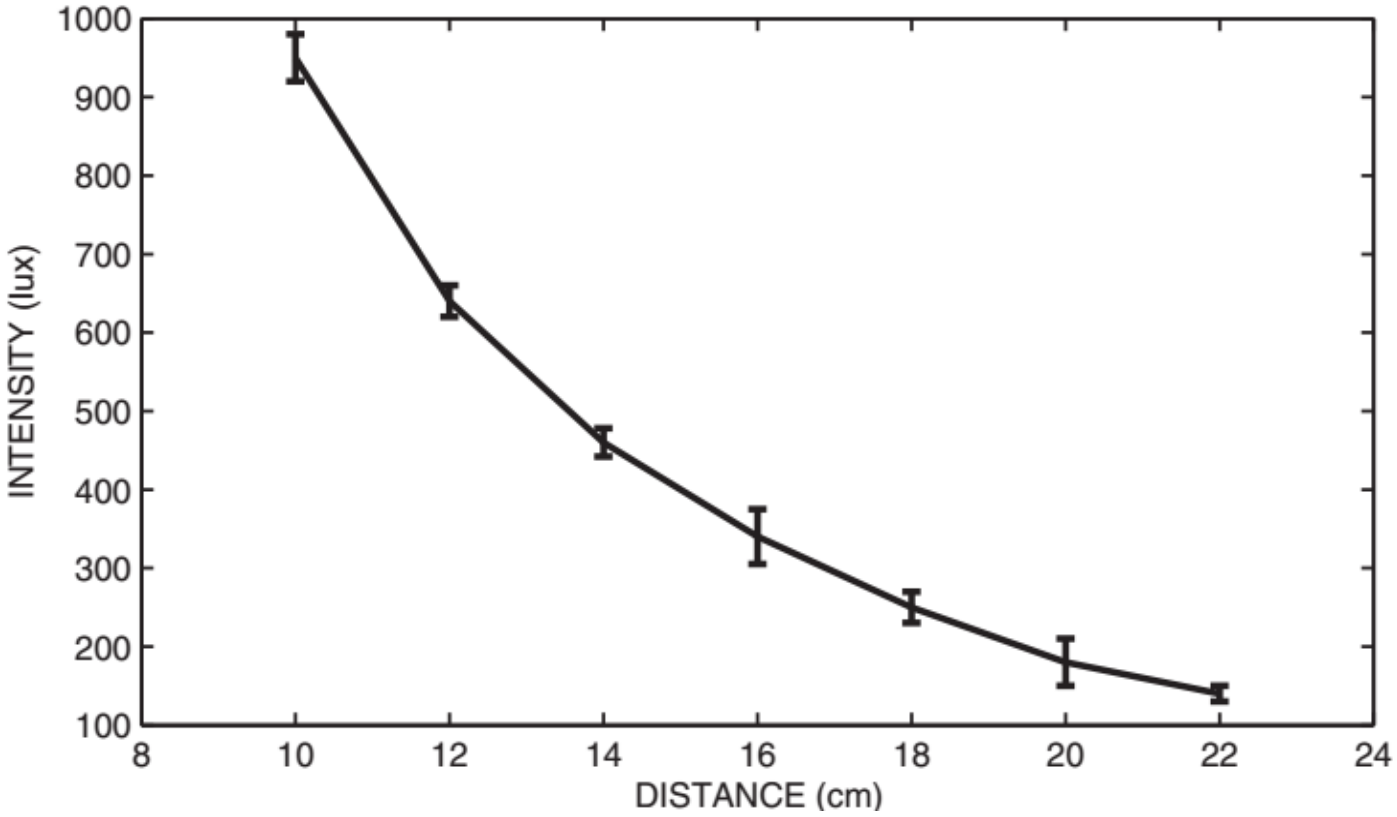


- The lengths of the three vectors x, y, and e must be the same.
- The length of the error bar is twice the value of e. At each point the error bar

  extends from y(i)-e(i) to y(i)+e(i).

Plots with Error Bars

```
xd=[10:2:22];
yd=[950 640 460 340 250 180 140];
ydErr=[30 20 18 35 20 30 10]
errorbar(xd,yd,ydErr)
xlabel('DISTANCE (cm)')
ylabel('INTENSITY (lux)')
```

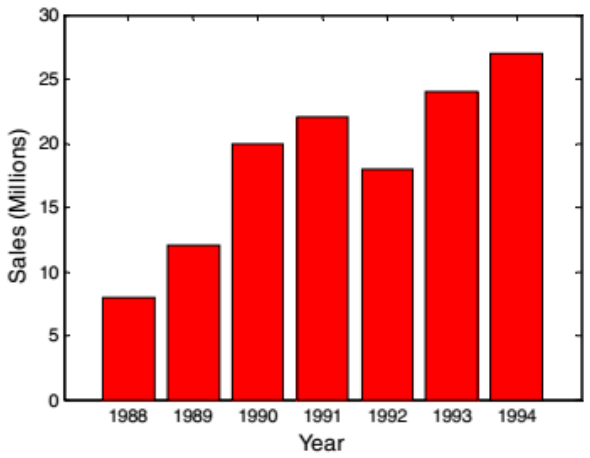


Plots with Special Graphics

Vertical Bar Plot

Function format:

`bar(x,y)`



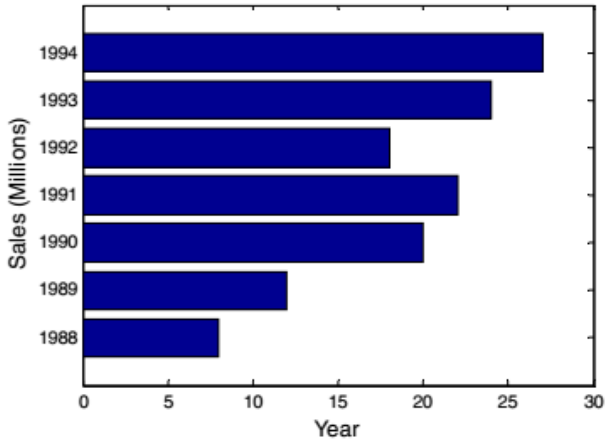

```
yr=[1988:1994];
sle=[8 12 20 22 18 24 27];
bar(yr,sle,'r')
xlabel('Year')
ylabel('Sales (Millions)')
```

The bars are in red.

Horizontal Bar Plot

Function format:

`barh(x,y)`



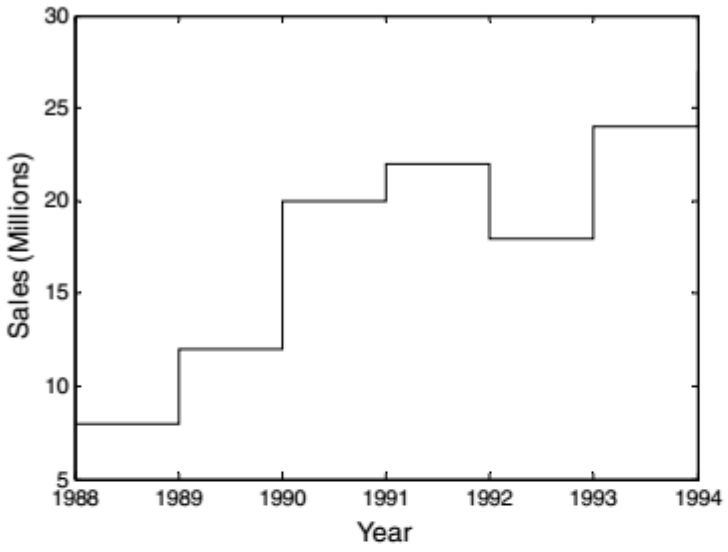

```
yr=[1988:1994];
sle=[8 12 20 22 18 24 27];
barh(yr,sle)
xlabel('Sales (Millions)')
ylabel('Year')
```

Plots with Special Graphics

Stairs Plot

Function format:

`stairs(x,y)`

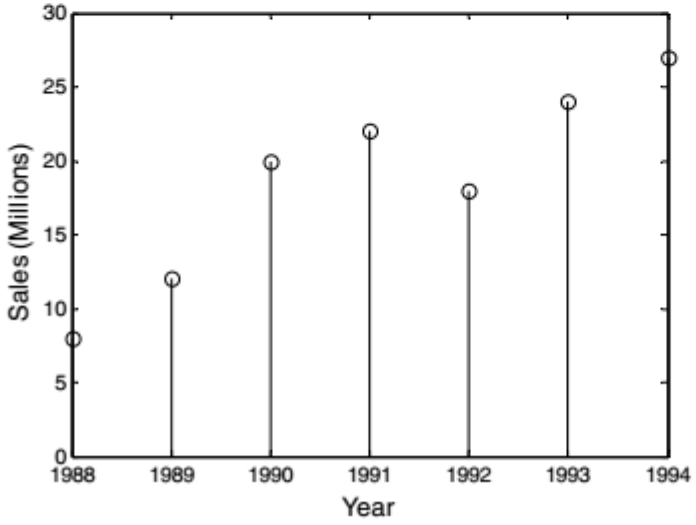



```
yr=[1988:1994];
sle=[8 12 20 22 18 24 27];
stairs(yr,sle)
```

Stem Plot

Function Format

`stem(x,y)`



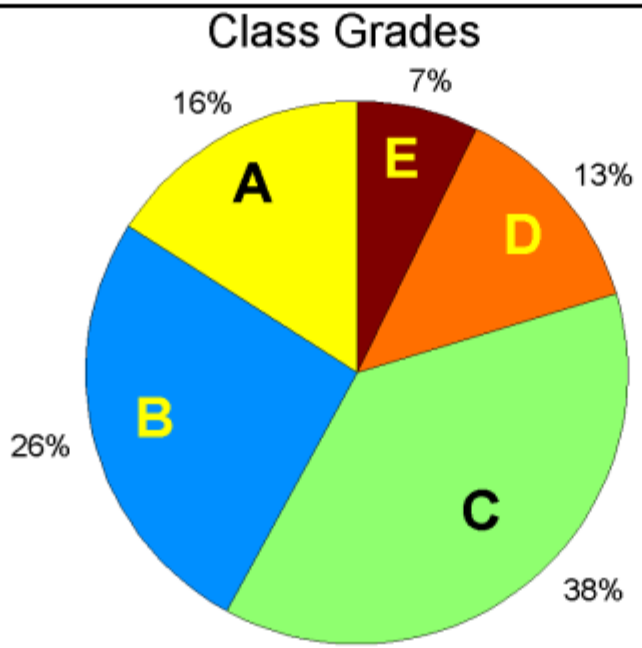

```
yr=[1988:1994];
sle=[8 12 20 22 18 24 27];
stem(yr,sle)
```

Plots with Special Graphics

Pie Plot

Function format:

`pie(x)`




```
grd=[11 18 26 9 5];
pie(grd)
title('Class Grades')
```

MATLAB draws the sections in different colors. The letters (grades) were added using the Plot Editor.

Command hist

```
hist(y)
```

`y` is a vector with the data points. MATLAB divides the range of the data points into 10 equally spaced subranges (bins) and then plots the number of data points in each bin.

```
hist(y,nbins)
```

or

```
hist(y,x)
```

`nbins` is a scalar that defines the number of bins. MATLAB divides the range in equally spaced subranges.

`x` is a vector that specifies the location of the center of each bin (the distance between the centers does not have to be the same for all the bins). The edges of the bins are at the middle point between the centers.

Command hist

`n=hist(y)`

`n=hist(y,nbins)`

`n=hist(y,x)`

`[n xout]=hist(y)`

`[n xout]=hist(y,nbins)`

n is the number of data points (frequency count) in the corresponding bin

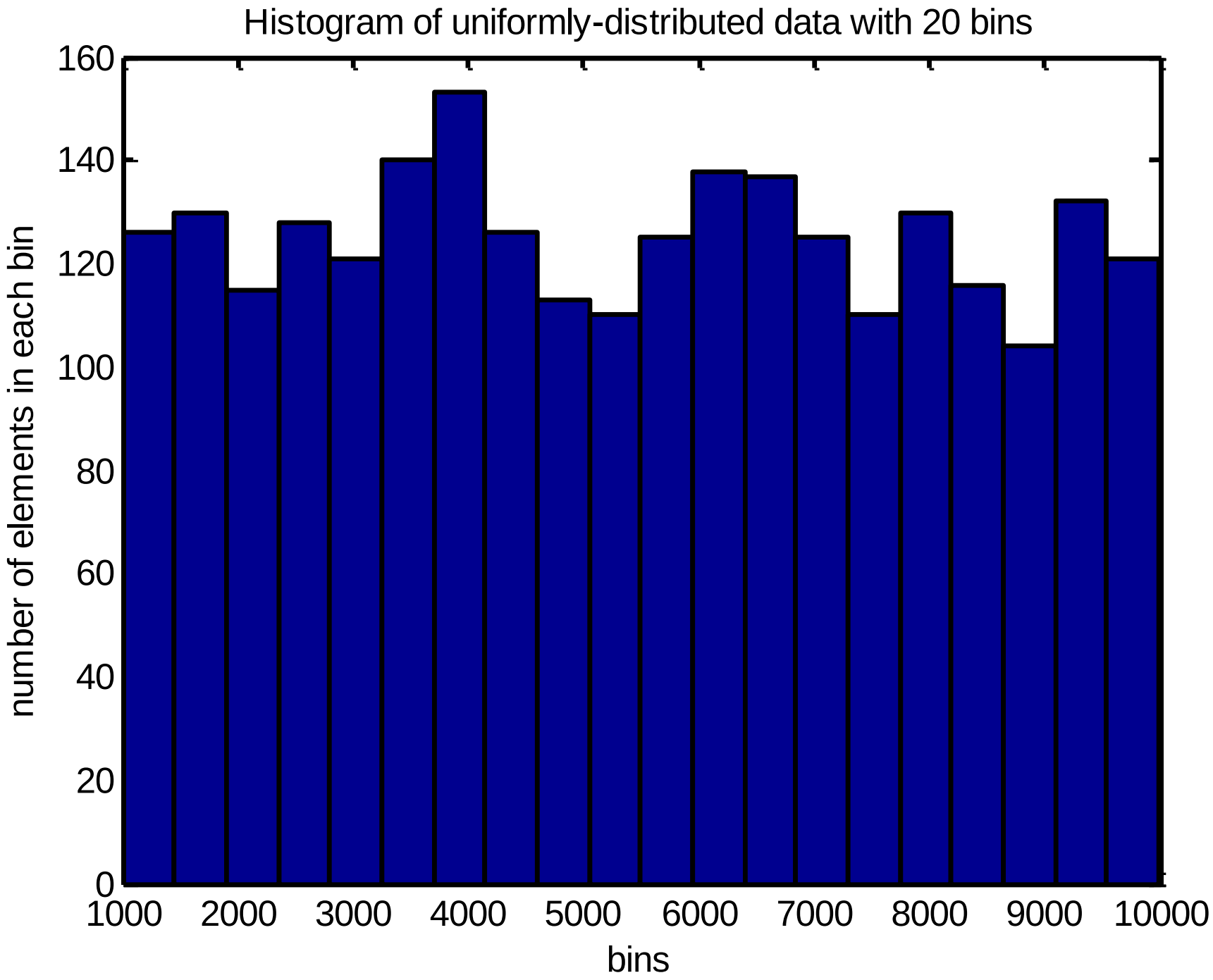
xout is a vector in which the value of each element is the location of the center of the corresponding bin.

```
>> y=[58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 91 93 89
91 80 59 69 56 64 63 66 64 74 63 69];
```

```
>> [n xout]=hist(y)

n =
   2   3   2   7   3   6   0   3   0   4

xout =
   50.2500  54.7500  59.2500  63.7500  68.2500  72.7500
77.2500  81.7500  86.2500  90.7500
```

Command hist

```
a=1000;b=10000; xx=rand(1,2500);

yy=(b-a)*xx + a;   % (1000~10,000之间正太分布的数)

hist(yy,20)

xlabel('bins')

ylabel('number of elements in each bin')

title('Histogram of uniformly-distributed data with 20 bins')
```


Histogram of uniformly-distributed data with 20 bins
number of elements in each bin
0
20
40
60
80
100
120
140
160
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
bins

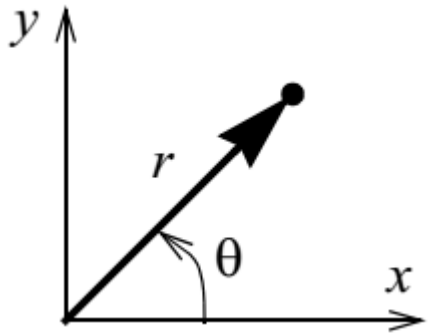
Ploar Plots

```
polar(theta,radius,'line specifiers')
```

Vector

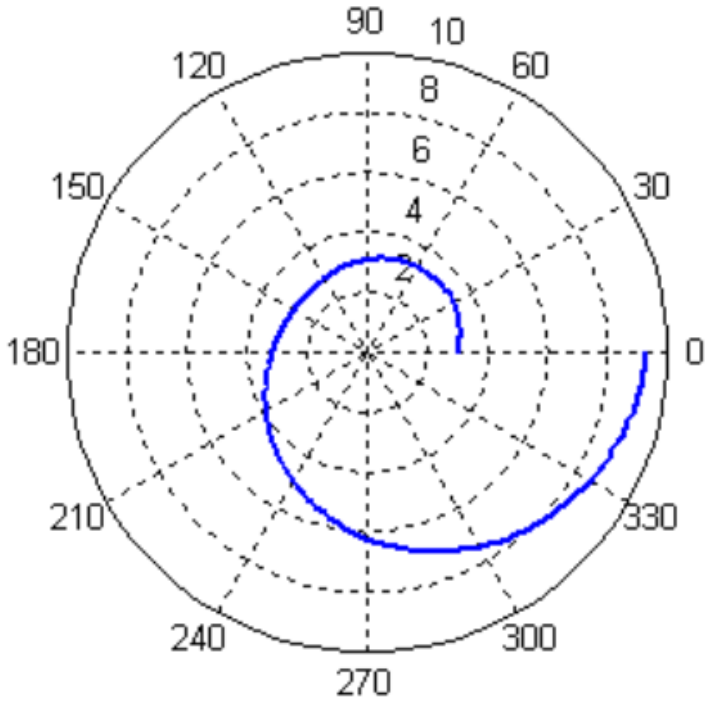
Vector

(Optional) Specifiers that define the type and color of the line and markers.



$$r = 3\cos^2(0.5\theta) + \theta \quad \text{for} \quad 0 \le \theta \le 2\pi$$

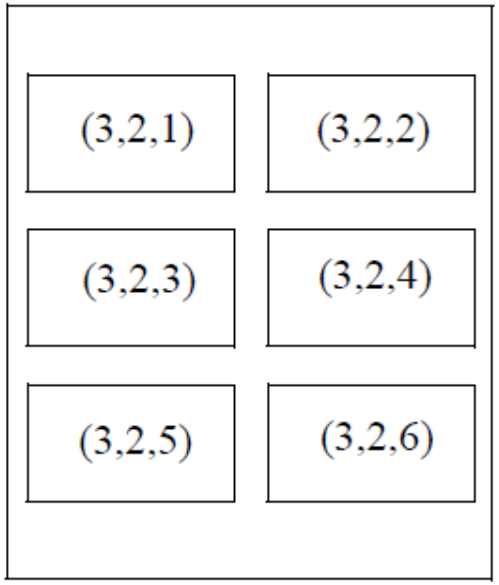
```
t=linspace(0,2*pi,200);
r=3*cos(0.5*t).^2+t;
polar(t,r)
```



Command subplot

`subplot(m,n,p)`

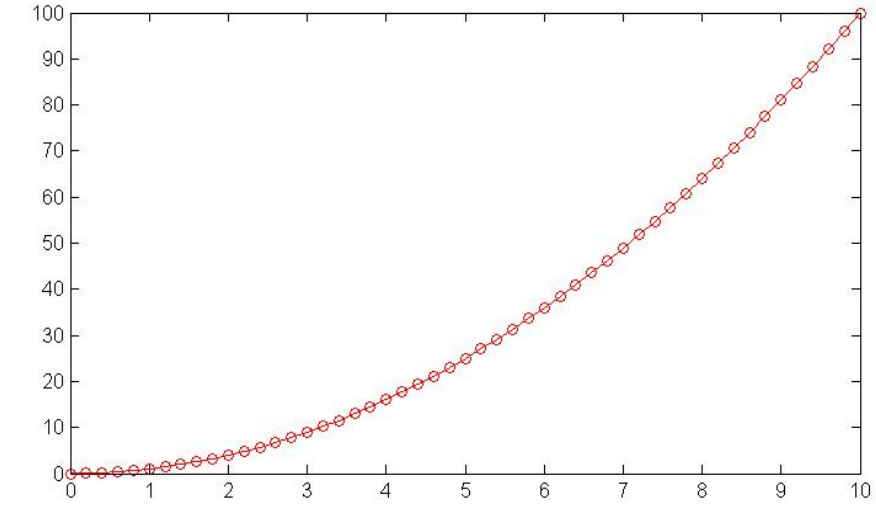
The command divides the Figure Window (and the page when printed) into $m \times n$ rectangular subplots. The subplots are arranged like elements in an $m \times n$ matrix where each element is a subplot. The subplots are numbered from 1 through $m \cdot n$. The upper left subplot is numbered 1 and the lower right subplot is numbered $m \cdot n$. The numbers increase from left to right within a row, from the first row to the last. The command `subplot(m,n,p)` makes the subplot `p` current. This means that the next plot command (and any formatting commands) will create a plot (with the corresponding format) in this subplot. For example, the command `subplot(3,2,1)` creates six areas arranged in three rows and two columns as shown, and makes the upper left subplot current. An example of using the `subplot` command is shown in the solution of Sample Problem 5-2.

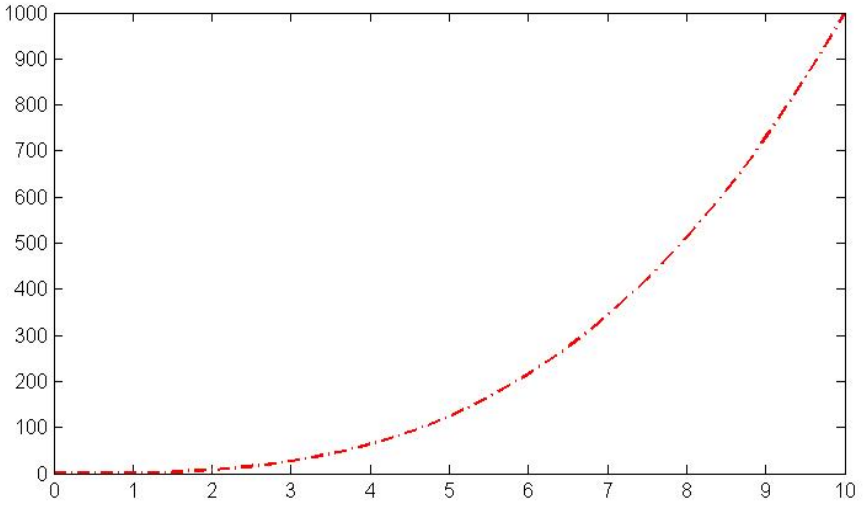


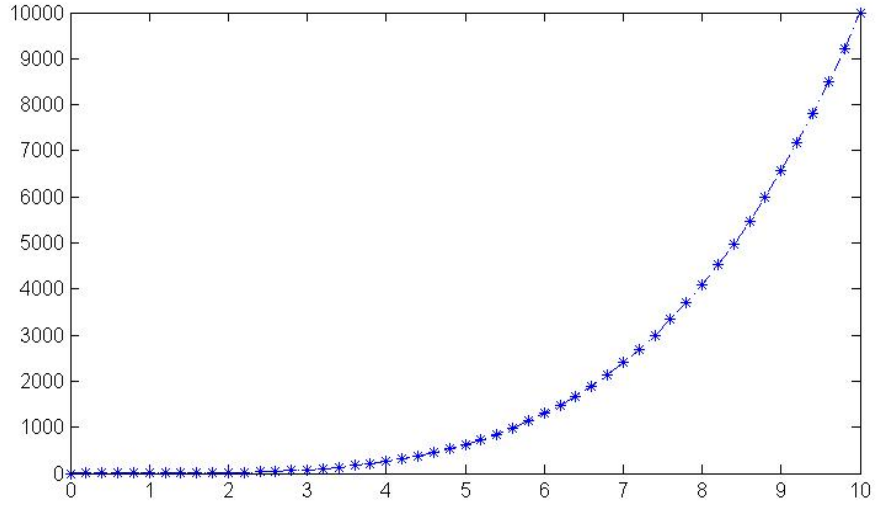
Command subplot

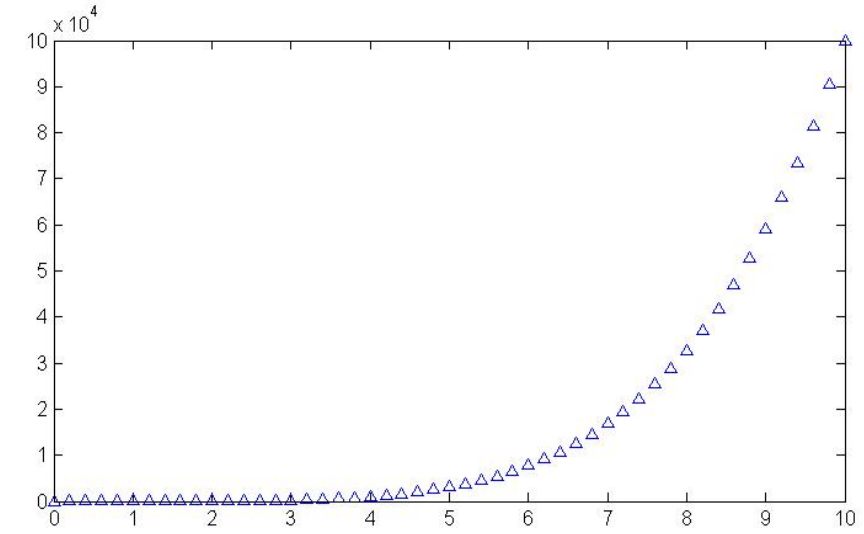
```
x = 0:0.2:10;
y1 = x.^2;
y2 = x.^3;
Subplot(2,2,1), plot(x,y1,'ko',x,y2,'r*'),
legend('2nd order polynomial', '3rd order polynomial')
Subplot(2,2,2) ….
Subplot(2,2,3)….
Subplot(2,2,4)…..
```

Command subplot





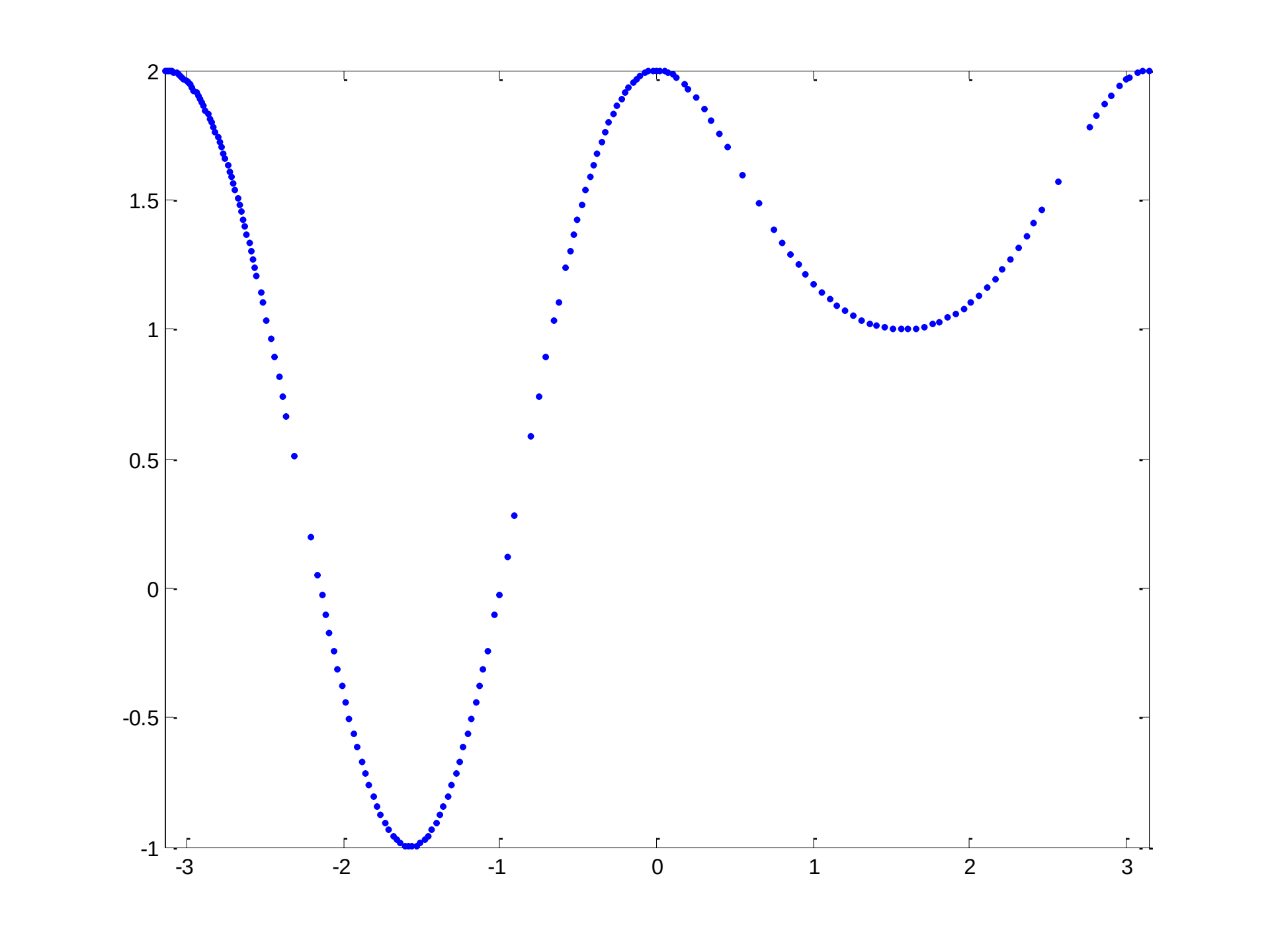




Command fplot

fplot(FUN,[x_min x_max], ’style_option’)

fplot('sin(x).^3 + 2*cos(x).^2', [-pi pi], '.')

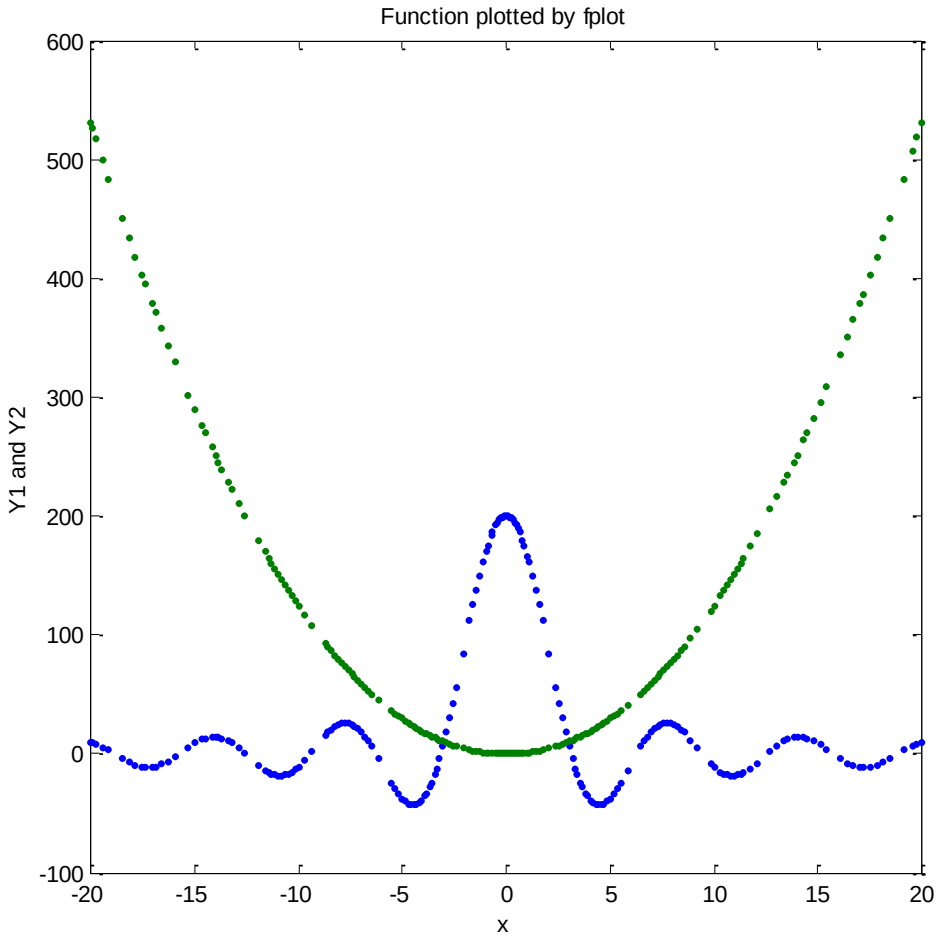

2
1.5
1
0.5
0
-0.5
-1
-3
-2
-1
0
1
2
3

Command fplot

```
function Y = fplot_example(X)
X = X(:);
Y = zeros(length(X),2);
Y(:,1) = 200*sin(X)./X;
Y(:,2) = abs(X).^(pi/1.5);
```

fplot

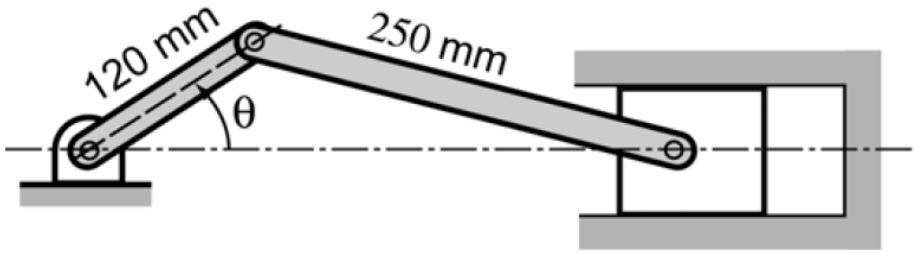
```
>> fplot('fplot_example', [-20 20], '.')
```



Example：Piston-crank mechanism

Problem:

The piston-rod-crank mechanism is used in many engineering applications. In the mechanism shown in the following figure, the crank is rotating at a constant speed of 500 rpm.



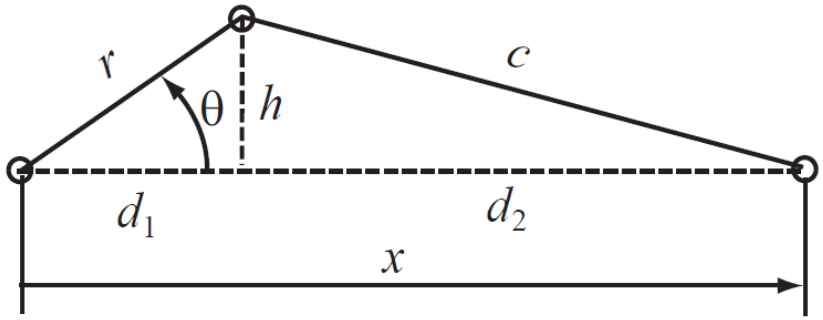
Calculate and plot the position, velocity, and acceleration of the piston for one revolution of the crank. Make the three plots on the same page. Set when t = 0.

Example :Withdrawing from a retirement account

Solution:

The crank is rotating with a constant angular velocity . This means that if we set θ = 0° when t = 0, then at time t the angle θ is given by $\theta = \dot{\theta}t$, and $\ddot{\theta} = 0$ that at all times. The distances d_1 and h are given by:

$$d_1 = r\cos\theta \quad \text{and} \quad h = r\sin\theta$$



With h known, the distance d_2 can be calculated using the Pythagorean Theorem:

$$d_2 = (c^2 - h^2)^{1/2} = (c^2 - r^2\sin^2\theta)^{1/2}$$

The derivative of x with respect to time gives the velocity of the piston:

$$\dot{x} = -r\dot{\theta}\sin\theta - \frac{r^2\dot{\theta}\sin 2\theta}{2(c^2 - r^2\sin^2\theta)^{1/2}}$$

The second derivative of x with respect to time gives the acceleration of the piston:

$$\ddot{x} = -r\dot{\theta}^2\cos\theta - \frac{4r^2\dot{\theta}^2\cos 2\theta(c^2 - r^2\sin^2\theta) + (r^2\dot{\theta}\sin 2\theta)^2}{4(c^2 - r^2\sin^2\theta)^{3/2}}$$

```
THDrpm=500; r=0.12; c=0.25;
THD=THDrpm*2*pi/60;
tf=2*pi/THD;
t=linspace(0,tf,200);
TH=THD*t;
d2s=c^2-r^2*sin(TH).^2;
x=r*cos(TH)+sqrt(d2s);
xd=-r*THD*sin(TH)-(r^2*THD*sin(2*TH))./(2*sqrt(d2s));
xdd=-r*THD^2*cos(TH)-(4*r^2*THD^2*cos(2*TH).*d2s+
(r^2*sin(2*TH)*THD).^2)./(4*d2s.^(3/2));
subplot(3,1,1)
plot(t,x)
grid
xlabel('Time (s)')
ylabel('Position (m)')
subplot(3,1,2)
plot(t,xd)
grid
xlabel('Time (s)')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
subplot(3,1,3)
plot(t,xdd)
grid
xlabel('Time (s)')
ylabel('Acceleration (m/s^2)')
```

Define $\dot{\theta}$, r, and c.

Change the units of $\dot{\theta}$ from rpm to rad/s.

Calculate the time for one revolution of the crank.

Create a vector for the time with 200 elements.

Calculate θ for each t.

Calculate d_2 squared for each θ.

Calculate x for each θ.

Calculate $\dot{x}$ and $\ddot{x}$ for each θ.

Plot x vs. t.

Format the first plot.

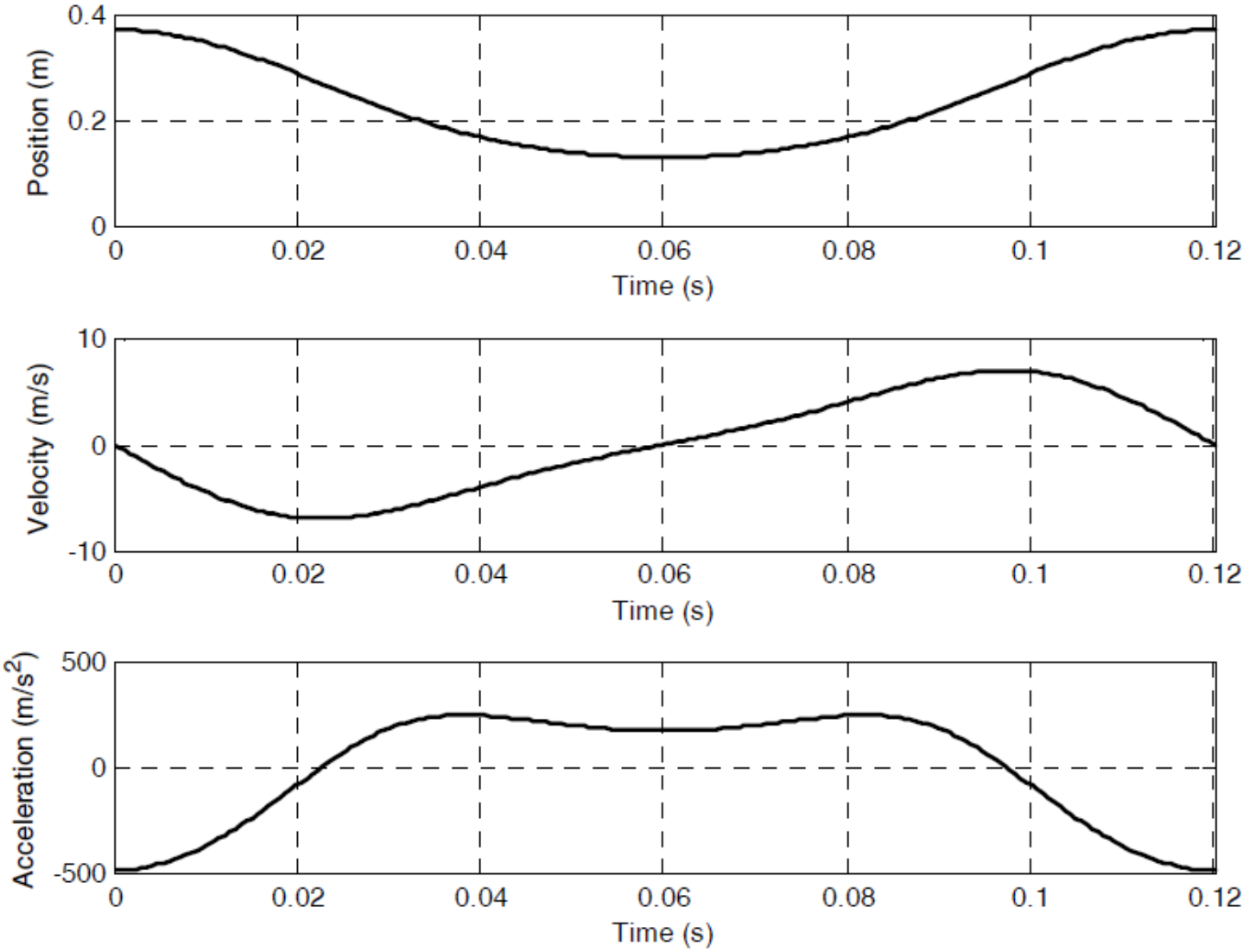
Plot $\dot{x}$ vs. t.

Format the second plot.

Plot $\ddot{x}$ vs. t.

Format the third plot.

Example :Withdrawing from a retirement account



Position, velocity, and acceleration of the piston vs. time

3-D Line PLOTS

3-D Line plots

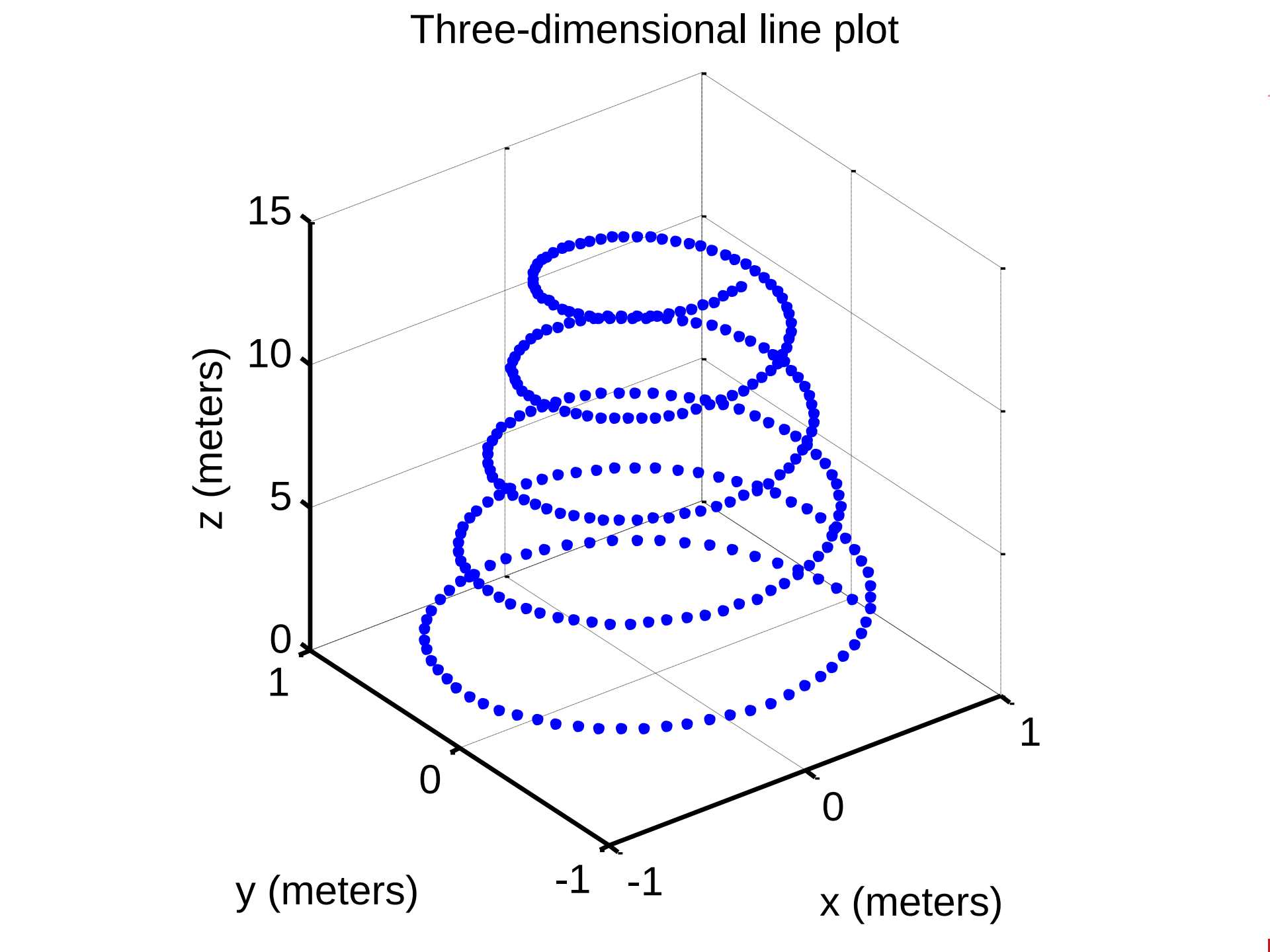
```
plot3(x,y,z,'line specifiers','PropertyName',property value)
```

x, y, and z are vectors of the coordinates of the points.

(Optional) Specifiers that define the type and color of the line and markers.

(Optional) Properties with values that can be used to specify the line width, and marker's size and edge and fill colors.

```
t = 0:0.05:15;
x = exp(-0.05*t).*cos(2*t);
y = exp(-0.05*t).*sin(2*t);
z = t;
figure, plot3 (x, y, z, '.')
xlabel('x (meters)'); ylabel('y (meters)'); zlabel('z (meters)');
title('Three-dimensional line plot');
```


Three-dimensional line plot
15
10
5
0
z (meters)
1
0
-1
y (meters)
-1
0
1
x (meters)

Surface Plots

Surface plots

- A surface is usually defined by a function of two independent variables;
- For example, function **z = f(x, y) = x.^2 + y**
- Function **f** is used to calculate **z** for each pair of **(x, y)** values.
- Therefore, to create a surface plot, we first need to generate a grid of **(x, y)** points
- Then find the value of **z** at each point of this grid.

Function meshgrid

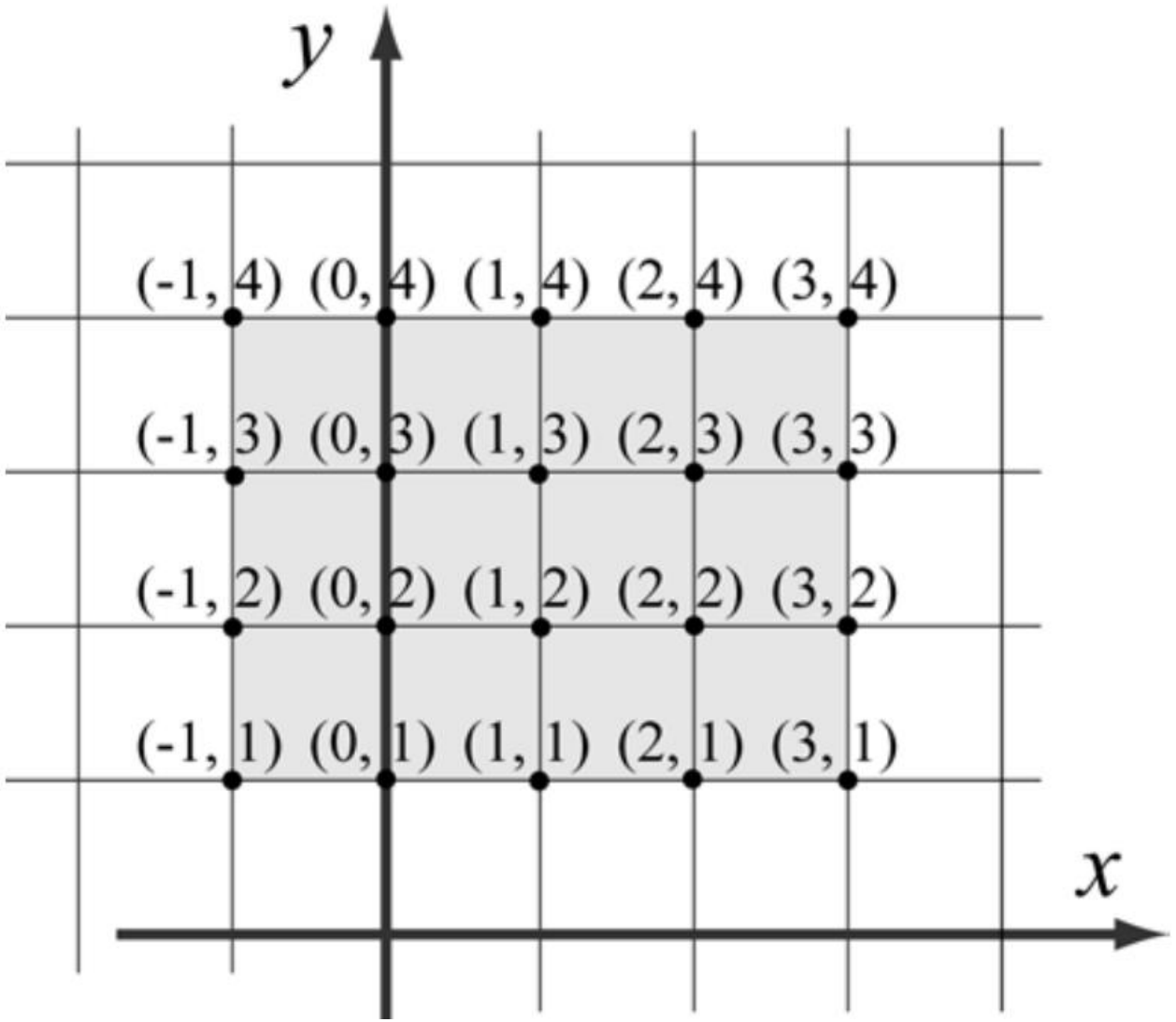
[x y] = meshgrid(xstart:xinc:xend,ystart:yinc:yend)

for example,

>> [xx yy] = meshgrid(-1:3, 1:4);

```
>> x=-1:3;
>> y=1:4;
>> [X,Y]=meshgrid(x,y)
X =
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
Y =
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
     4     4     4     4     4
```

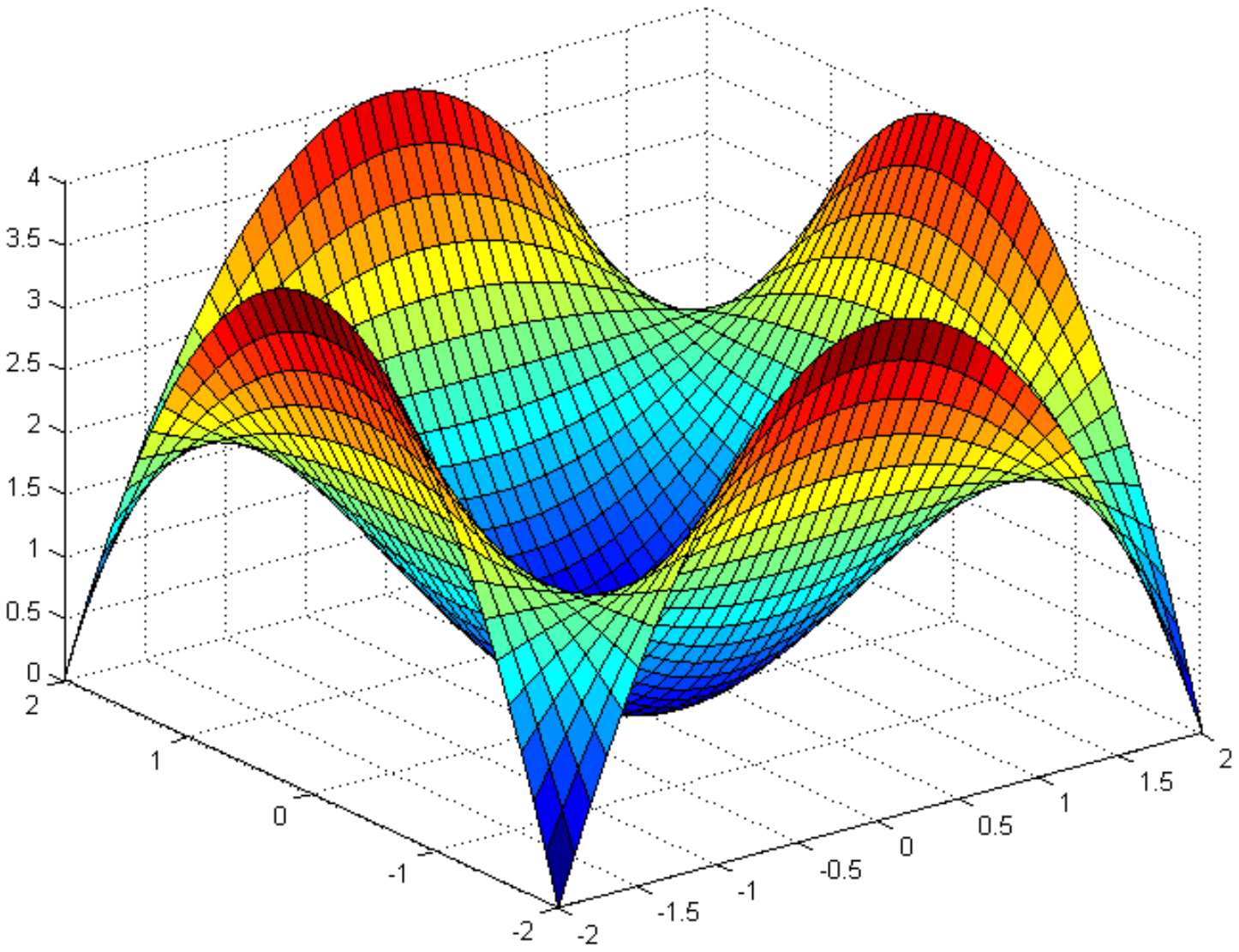
Coordinates of points by meshgrid



Function surf

```
t1 = -2:0.1:2;
t2 = -2:0.1:2;
[x y] = meshgrid(t1, t2);
z = (x.^2) + (y.^2) -0.5*(x.^2).*(y.^2);
figure, surf(x,y,z)
```

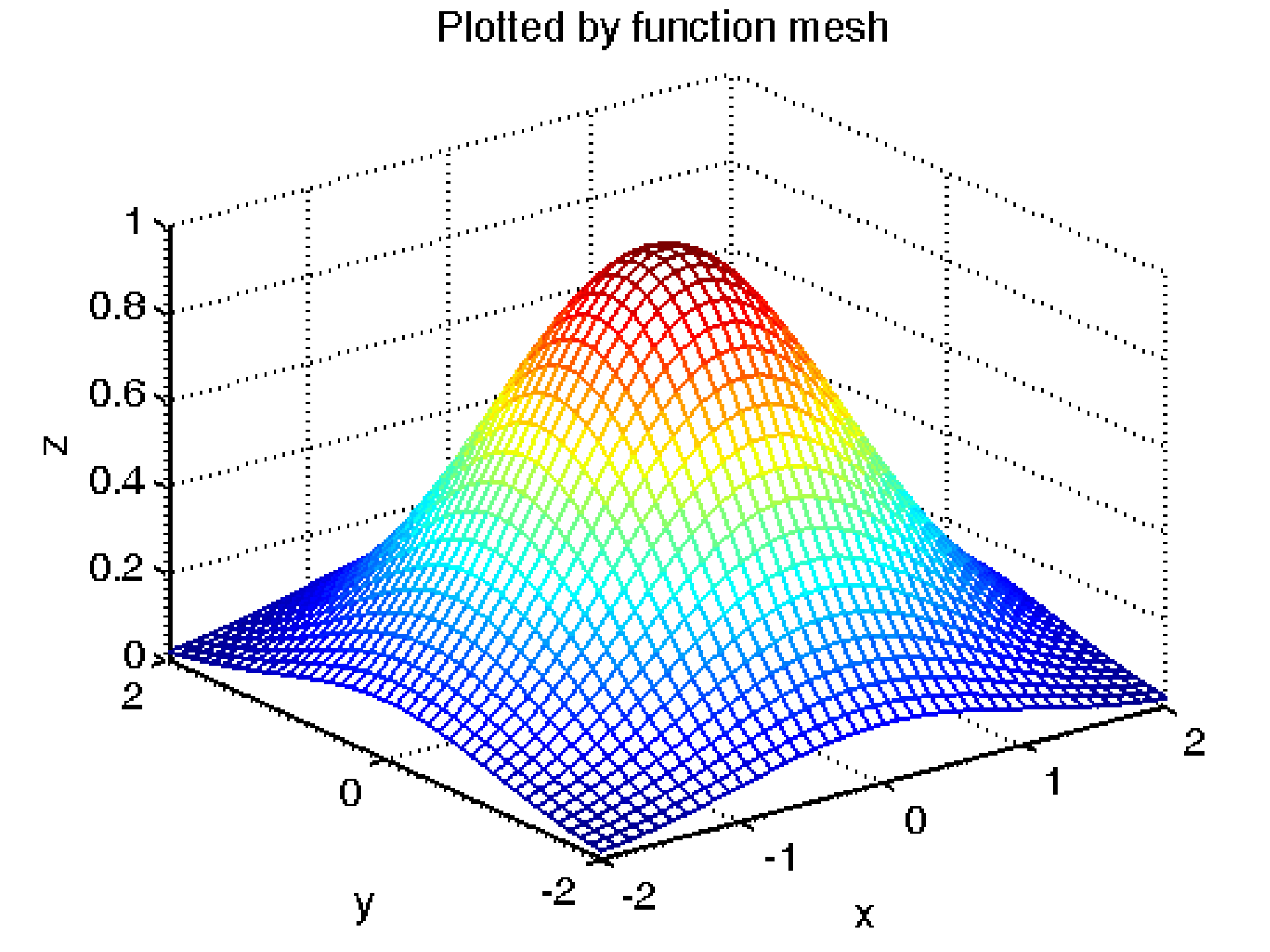
Function surf

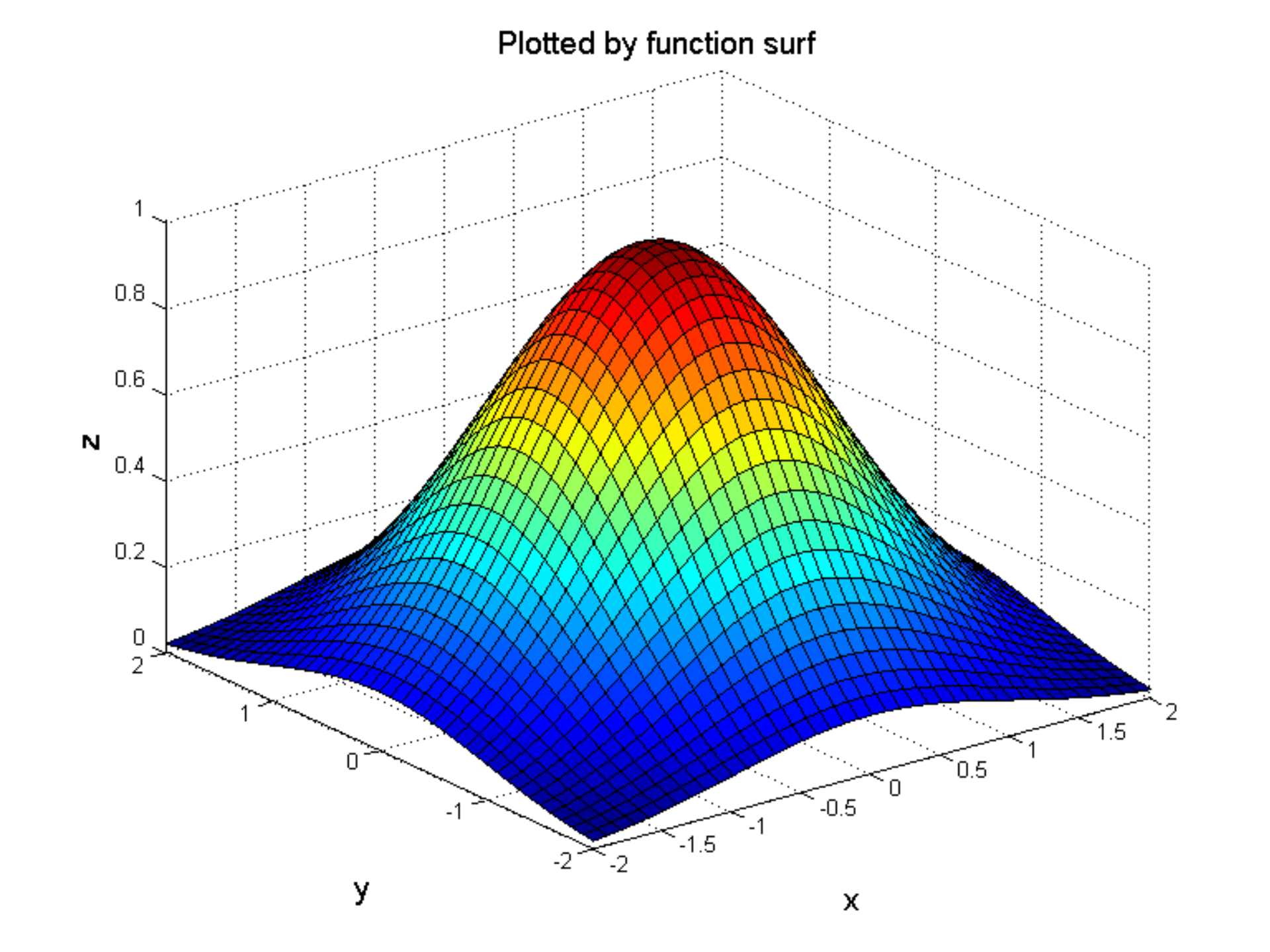


Function mesh & surf

- Function **mesh** creates a surface which has a **wire-frame** appearance.
- Function surf creates a surface which looks more like a real surface.

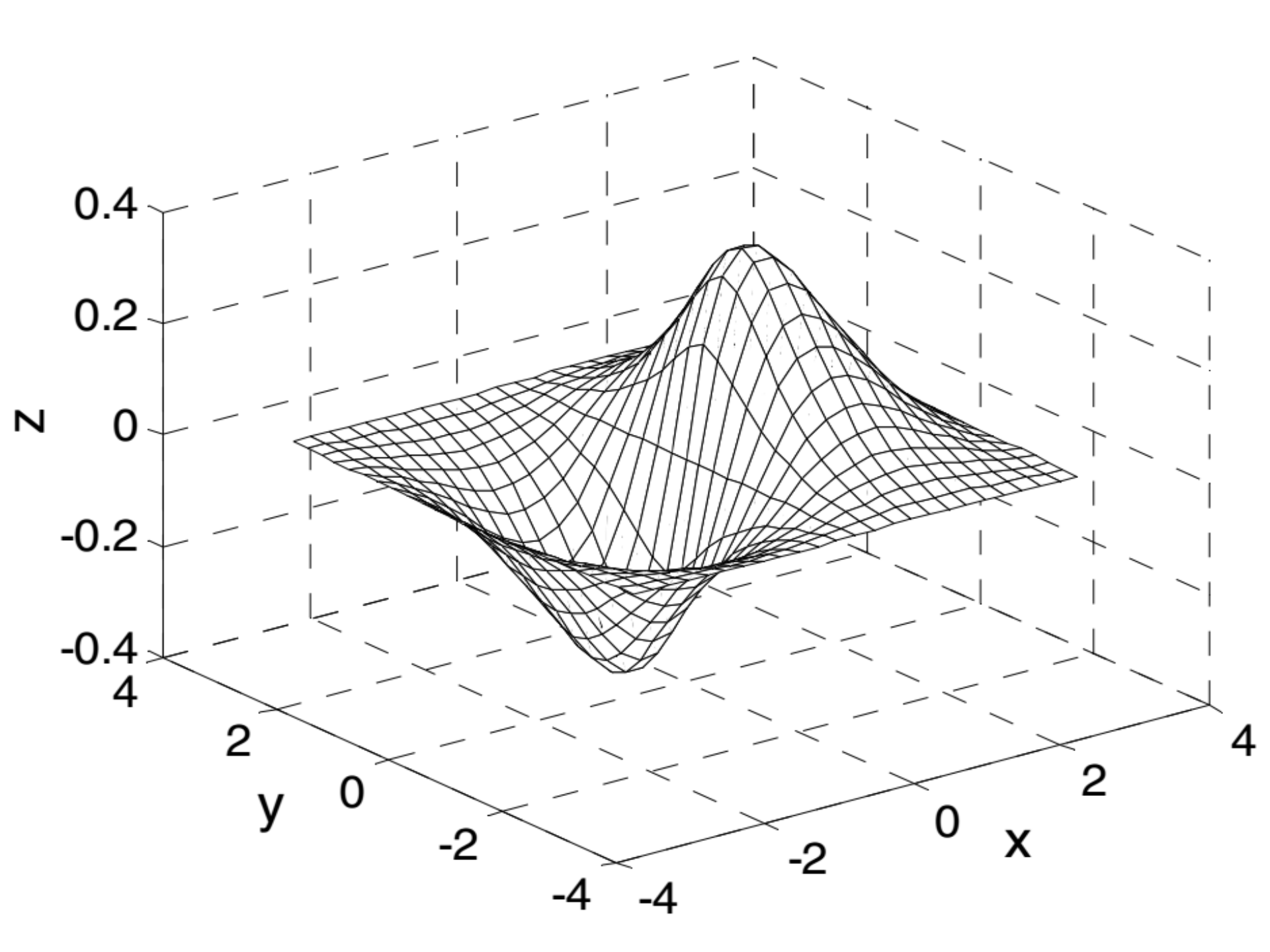
```
t1 = -2:0.1:2;
t2 = -2:0.1:2;
[x y] = meshgrid(t1,t2);
z = exp(-0.5*(x.^2 + y.^2));
figure, mesh(x,y,z)
figure, surf(x,y,z)
```

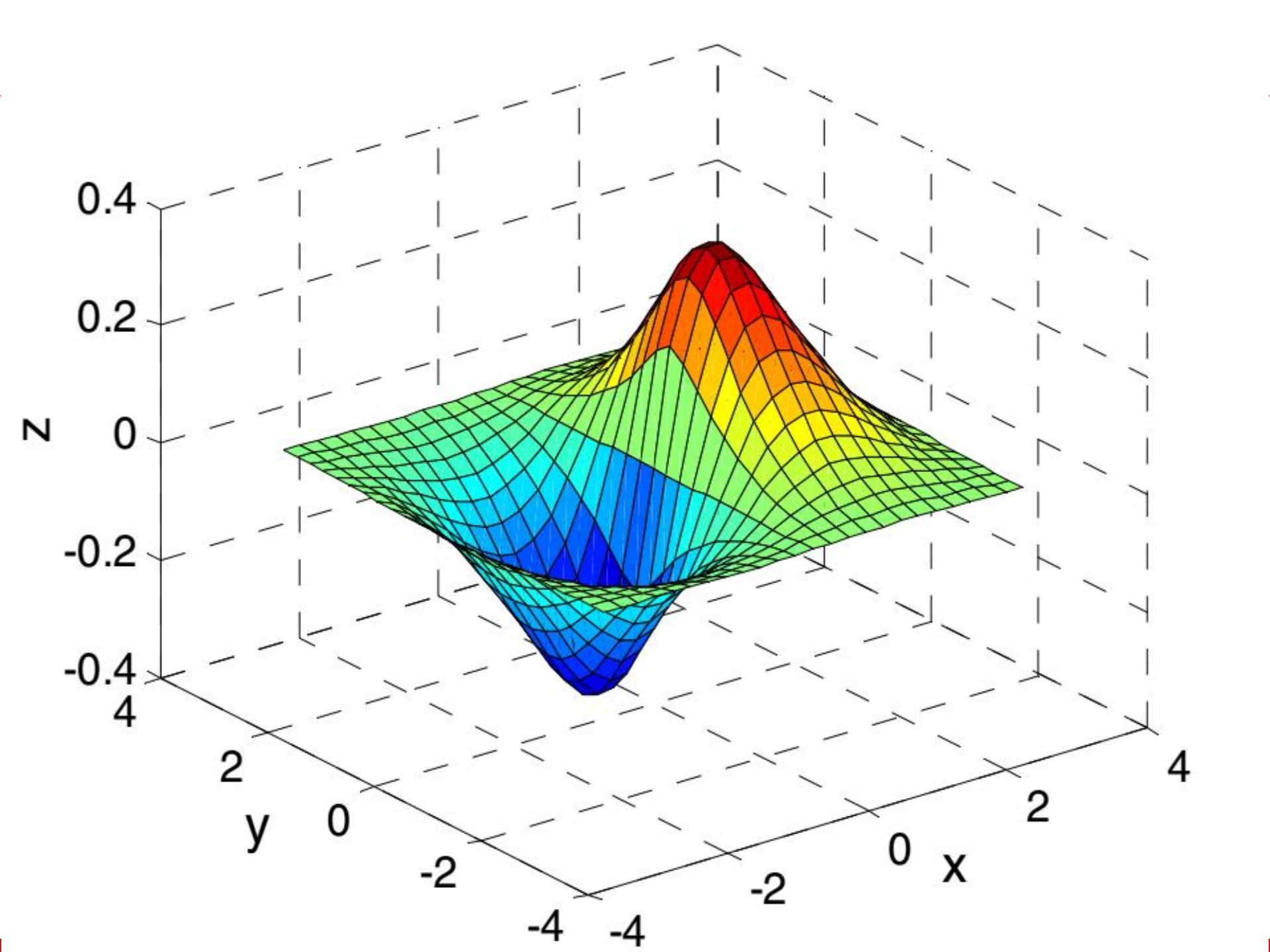

Plotted by function mesh
z
1
0.8
0.6
0.4
0.2
0
2
0
-2
y
-2
-1
0
1
2
x


Plotted by function surf
z
1
0.8
0.6
0.4
0.2
0
2
1
0
-1
-2
y
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
x

Function mesh & surf

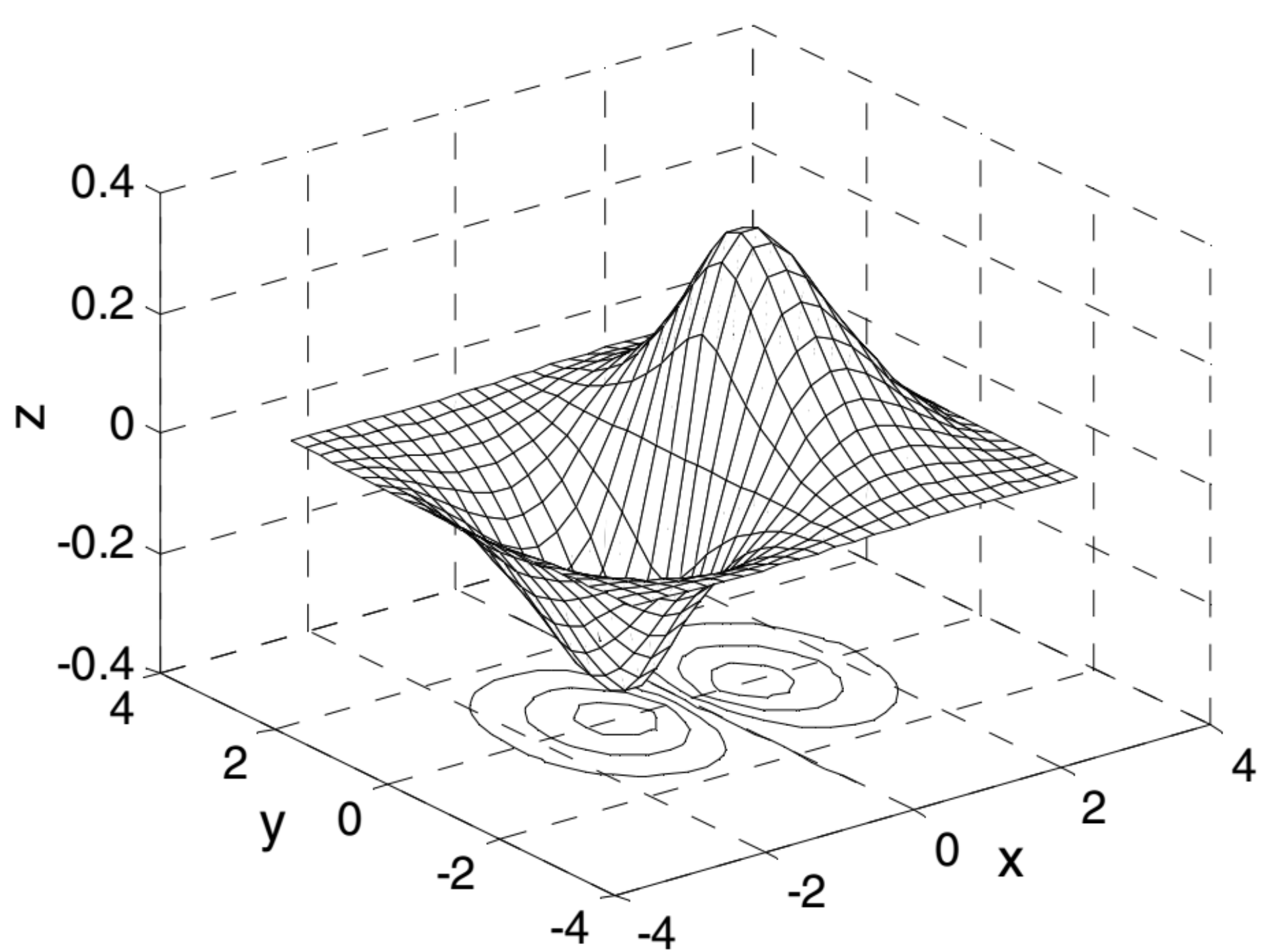
```
x=-3:0.25:3;
y=-3:0.25:3;
[X,Y] = meshgrid(x,y);
Z=1.8.^(-1.5*sqrt(X.^2+Y.^2)).*cos(0.5*Y).*sin(X);

figure(1)
mesh(X,Y,Z);xlabel('x');ylabel('y');zlabel('z')

figure(2)
surf(X,Y,Z);xlabel('x');ylabel('y');zlabel('z')
```

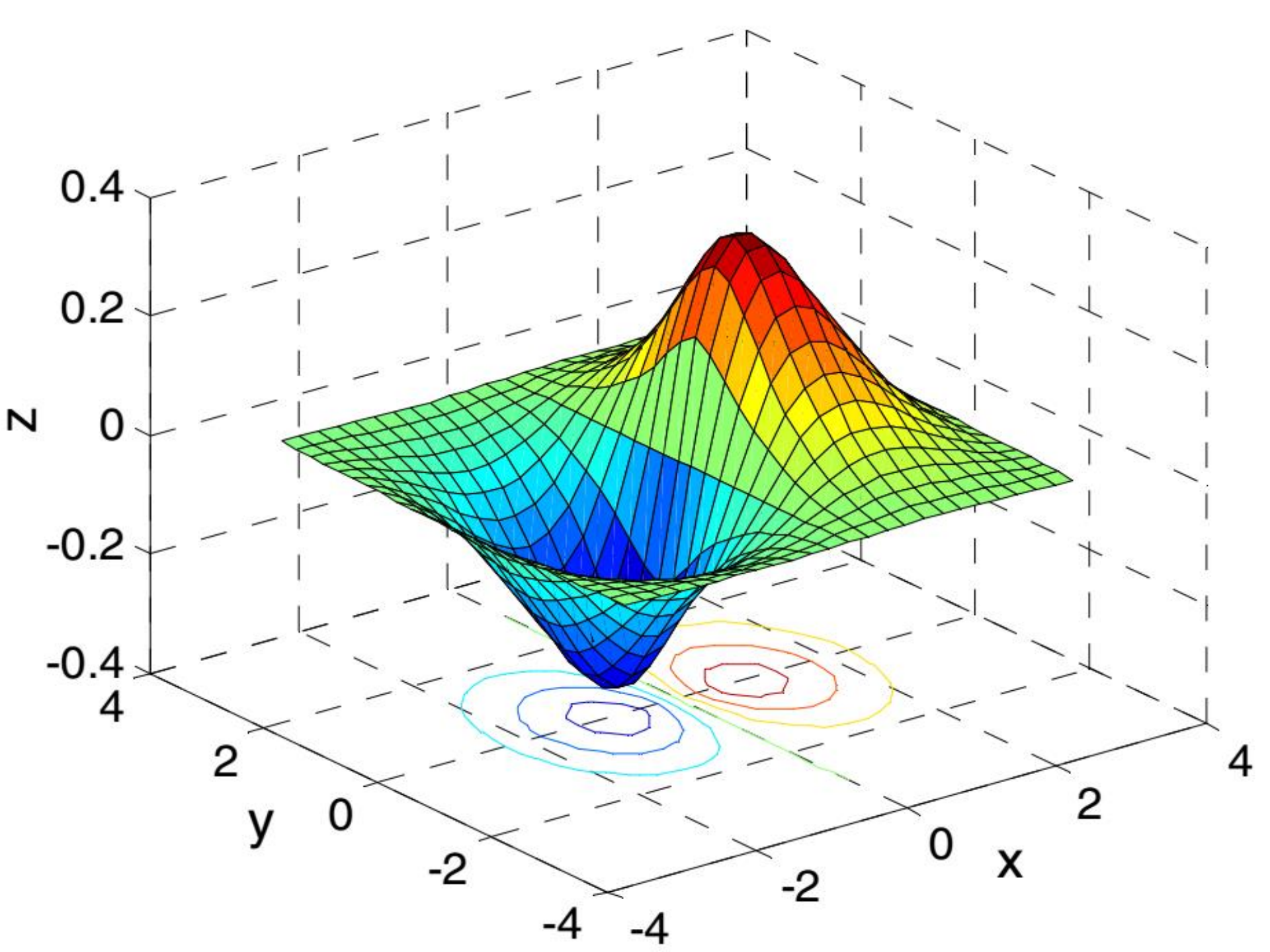

0.4
0.2
0
-0.2
-0.4
z
4
2
0
-2
-4
y
-4
-2
0
2
4
x


0.4
0.2
z
0
-0.2
-0.4
4
2
y
0
-2
-4
-4
-2
0
x
2
4

Function mesh & surf

```
x=-3:0.25:3;
y=-3:0.25:3;
[X,Y] = meshgrid(x,y);
Z=1.8.^(-1.5*sqrt(X.^2+Y.^2)).*cos(0.5*Y).*sin(X);

figure(1)
meshc(X,Y,Z);xlabel('x');ylabel('y');zlabel('z')

figure(2)
surfc(X,Y,Z);xlabel('x');ylabel('y');zlabel('z')
```


0.4
0.2
z
0
-0.2
-0.4
4
2
y
0
-2
-4
-4
-2
0
x
2
4


0.4
0.2
0
-0.2
-0.4
z
4
2
0
-2
-4
y
-4
-2
0
2
4
x

Contours

Function contour contourf

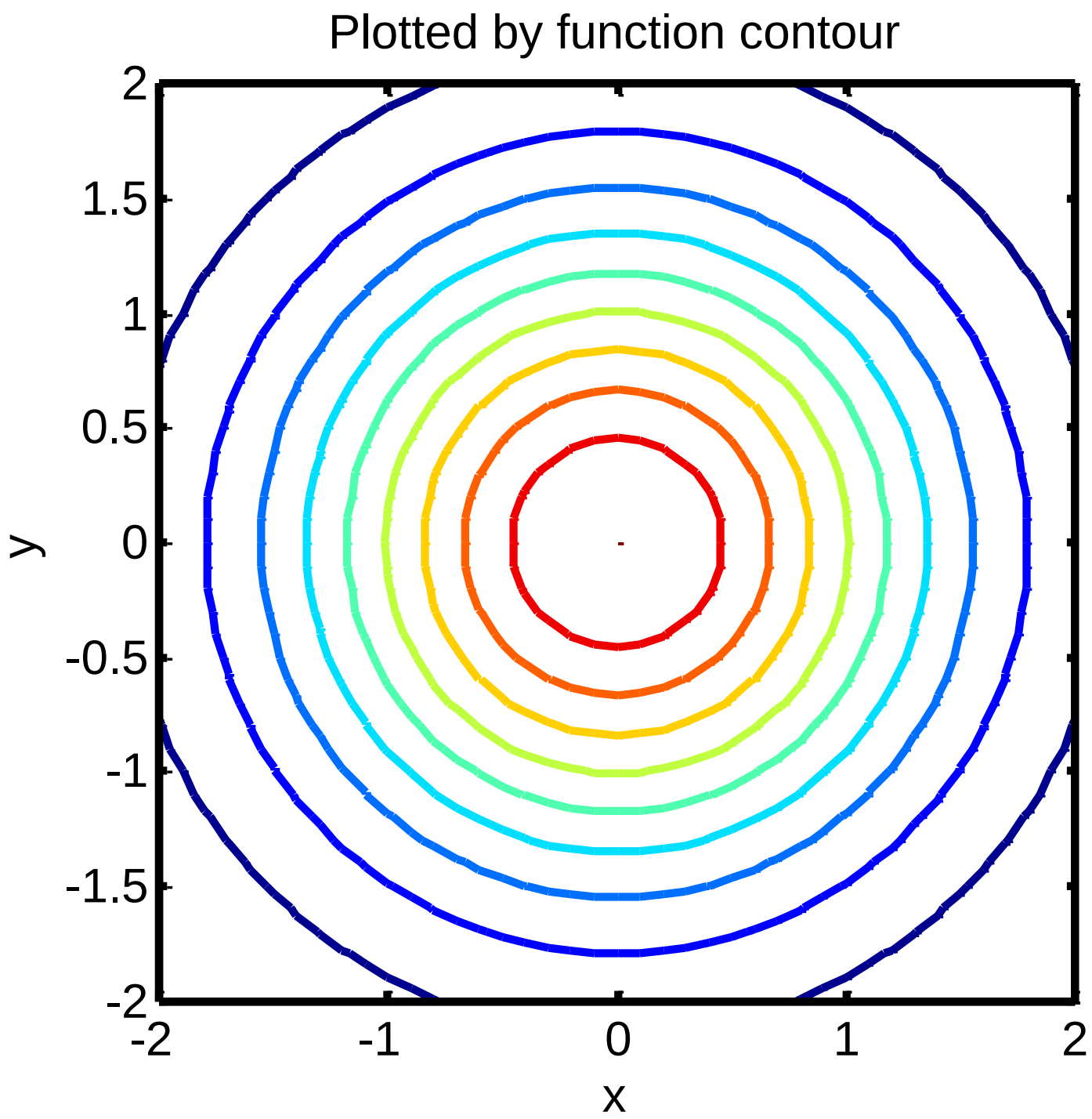
figure, **contour** (x, y, z**, [0.1:0.1:1.9]**)

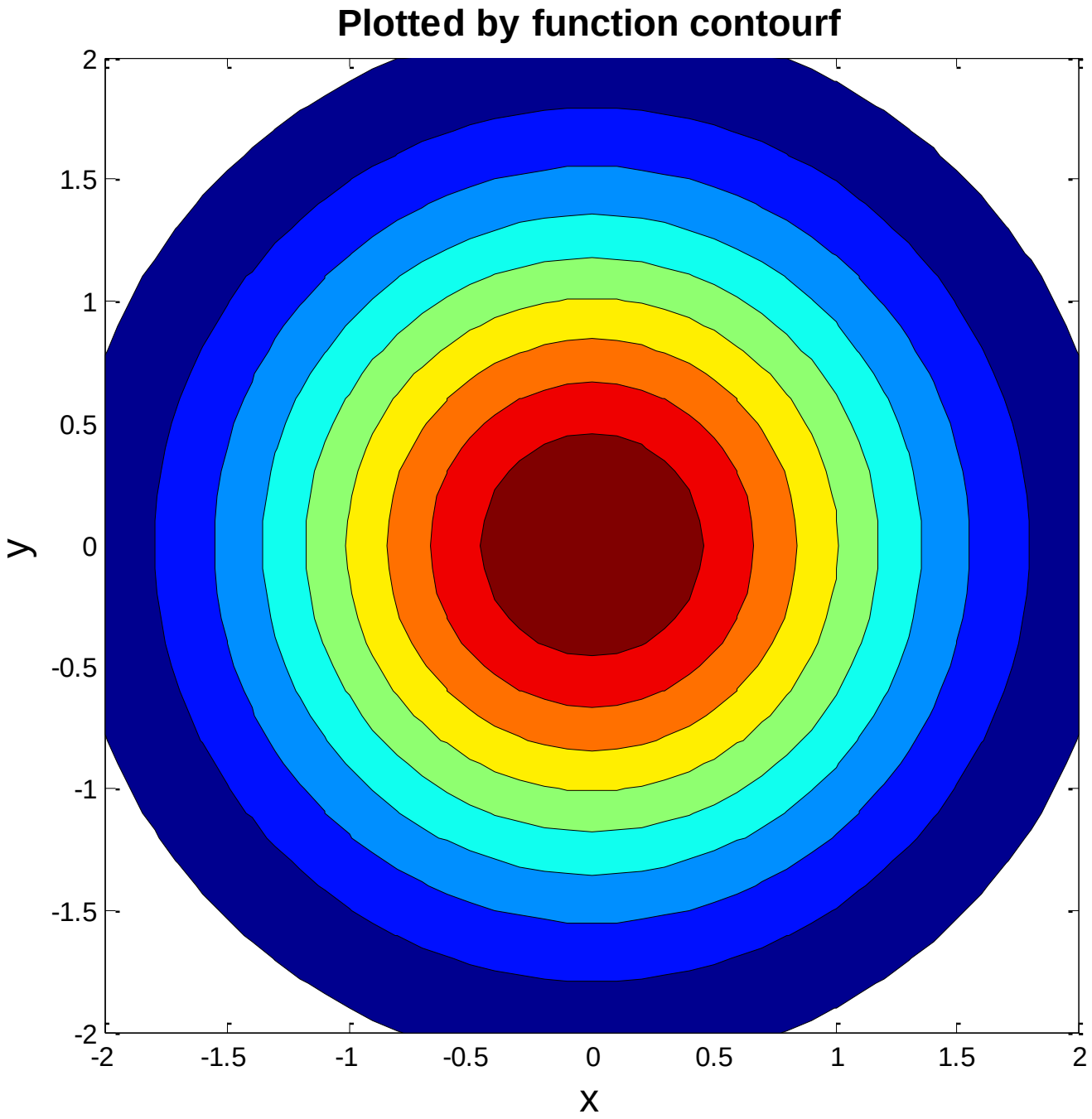
axis square

xlabel('x', 'Fontsize',14),

ylabel('y', 'Fontsize',14)

title('Plotted by function contour', 'Fontsize',14)

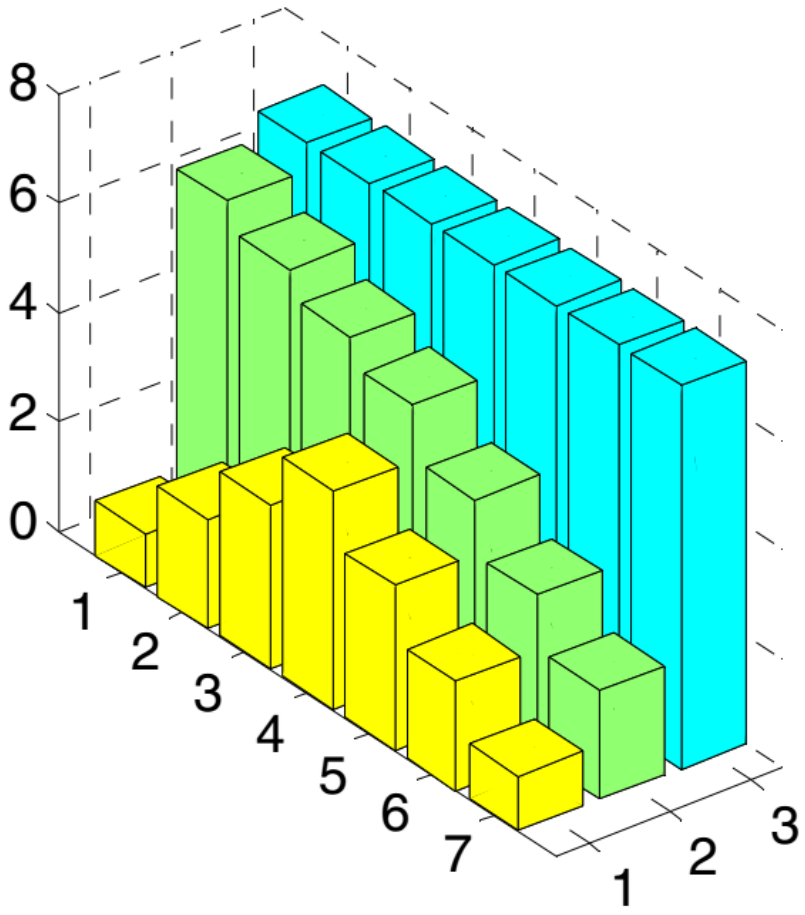

Plotted by function contour
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
y
-2
-1
0
1
2
x


Plotted by function contourf
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
y
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
x

Plots with special graphics

3-D Bar Plot

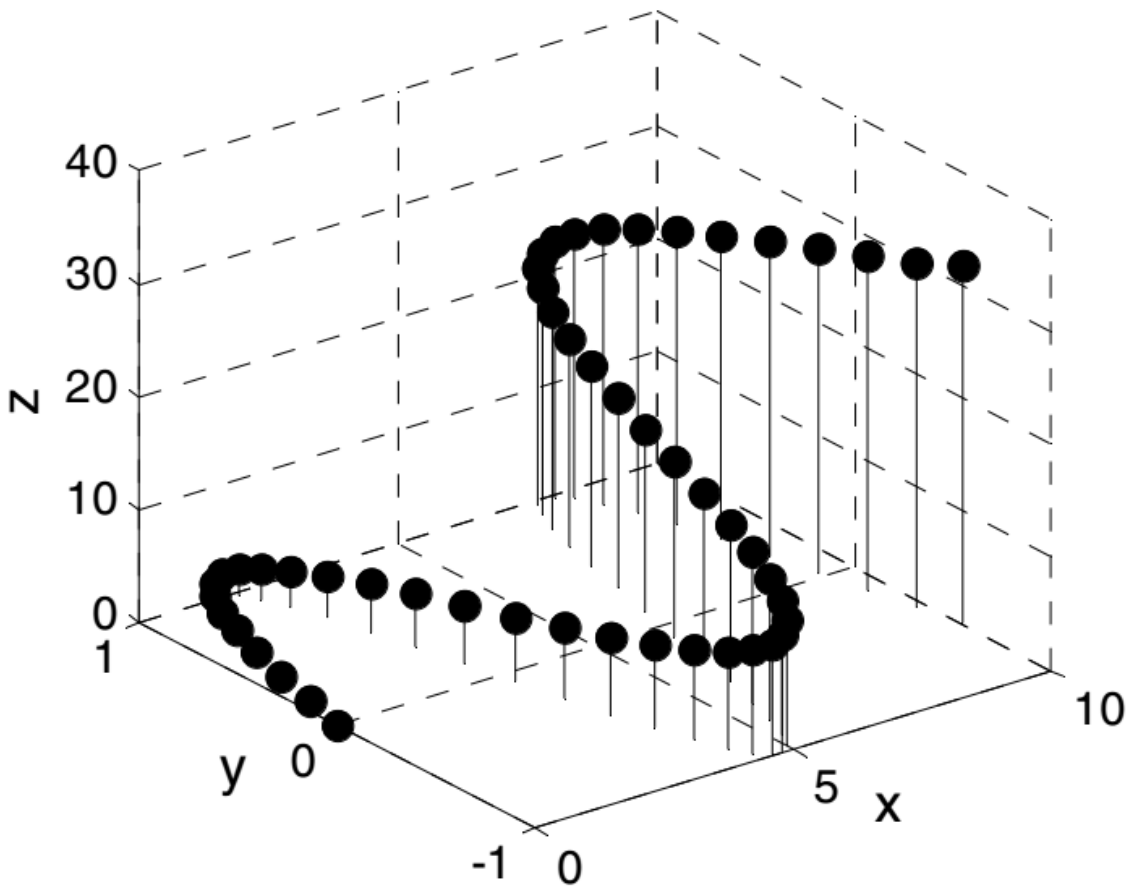
```
Y=[1 6.5 7; 2 6 7; 3 5.5 7; …
4 5 7; 3 4 7; 2 3 7; 1 2 7];
bar3(Y)
```



Plots with special graphics

3-D Stem Plot

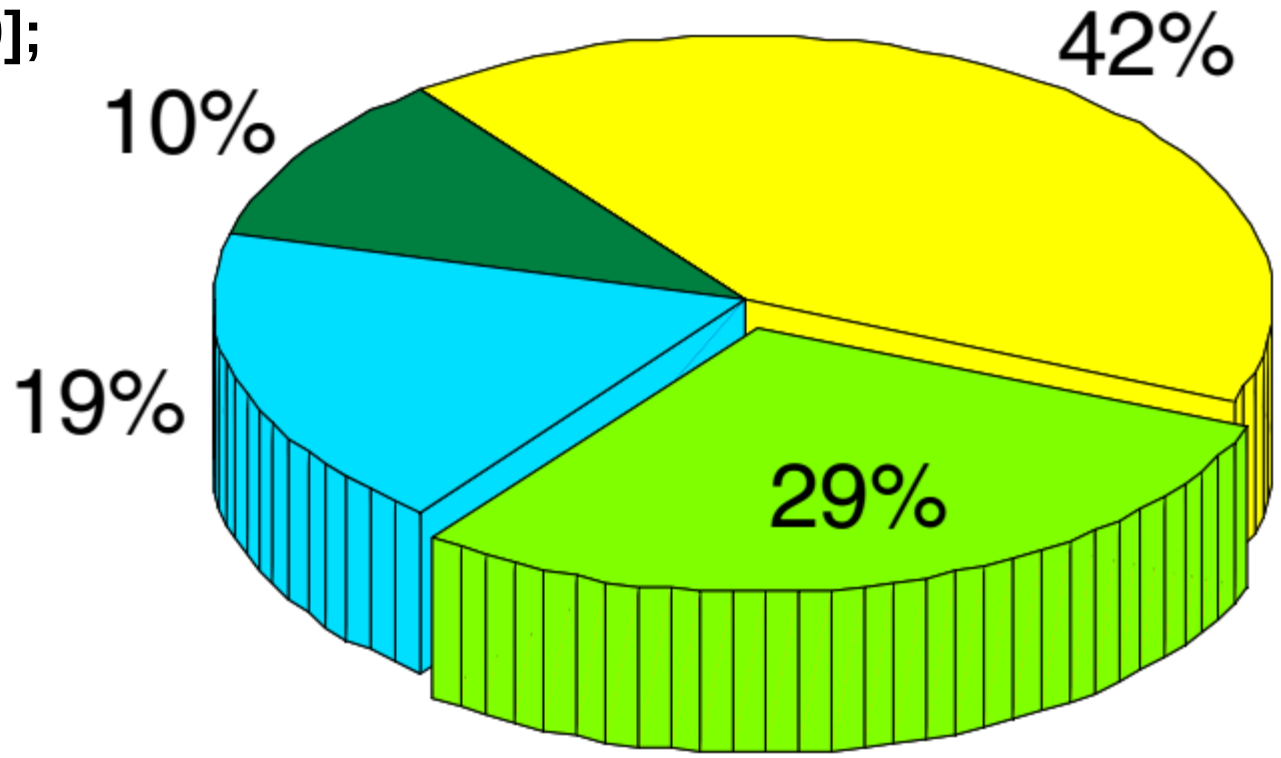
```
t=0:0.2:10;
x=t;
y=sin(t);
z=t.^1.5;
stem3(x,y,z,'fill')
grid on
xlabel('x');
ylabel('y')
zlabel('z')
```



Plots with special graphics

3-D Pie Plot

```
X=[5 9 14 20];
explode=[0 0 1 0];
pie3(X,explode)
```

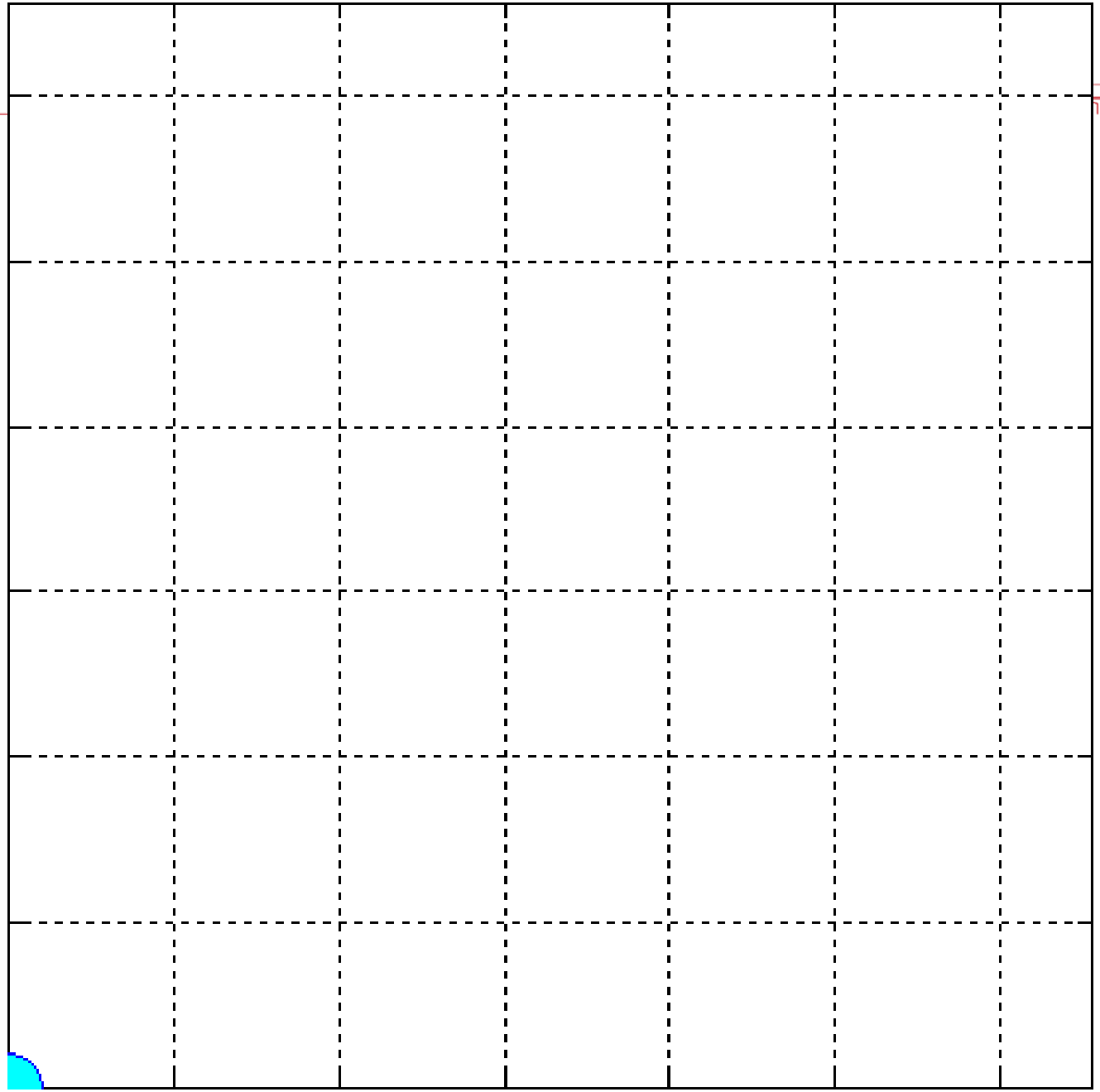


Fractals and Chaos

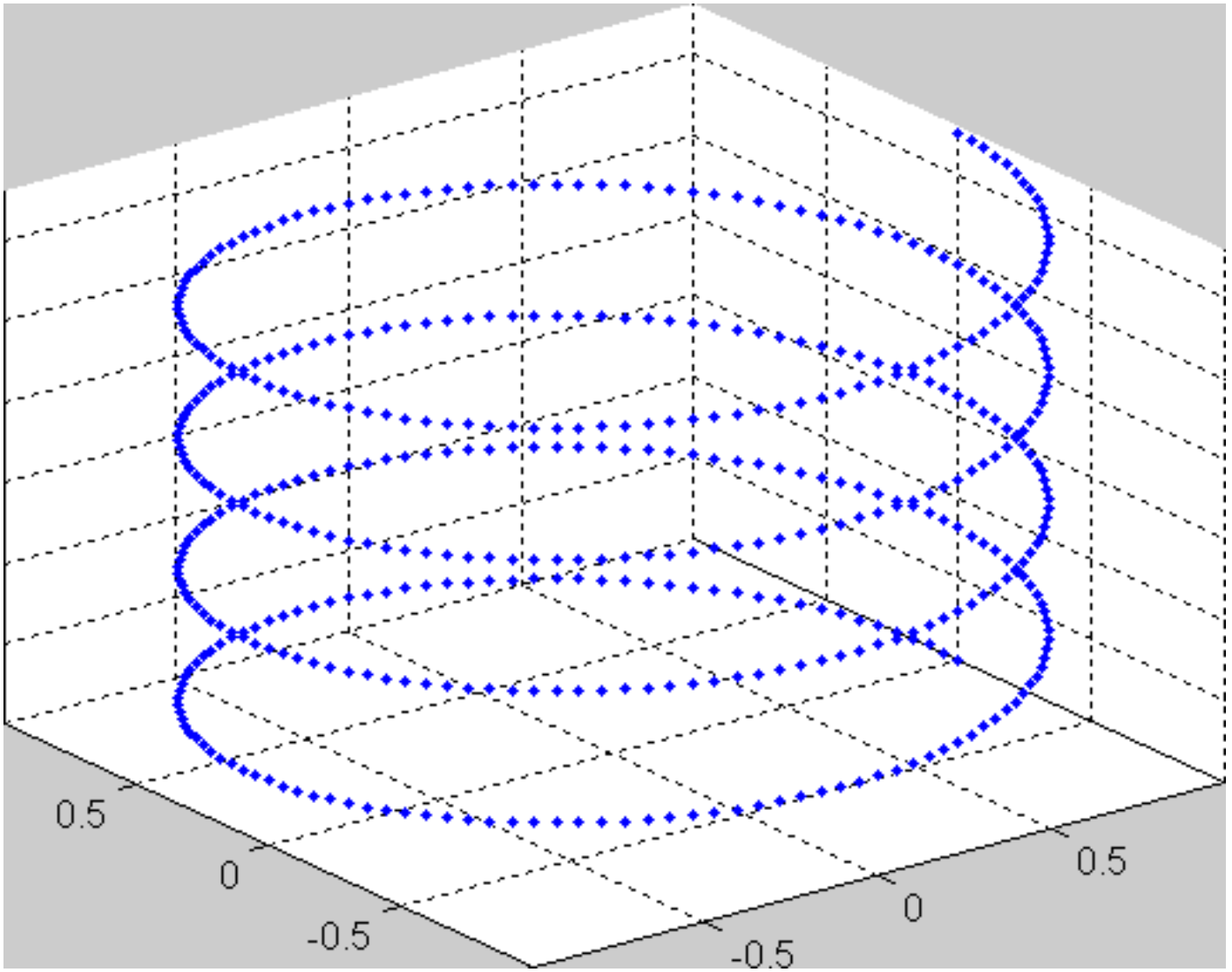
```
function fractal_leaf(number_of_points)
%example: fractal_leaf(10000)
 Mat1    = [0 0;0 0.16];
Mat2    = [0.85 0.04;-0.04 0.85];
Mat3    = [0.2 -0.26;0.23 0.22];
Mat4    = [-0.15 0.28;0.26 0.24];
Vector1 = [0;0];
Vector2 = [0;1.6];
Vector3 = [0;1.6];
Vector4 = [0;0.44];
Prob1   = 0.01;
Prob2   = 0.85;
Prob3   = 0.07;
Prob4   = 0.07;
P       = [0;0];
```

```
x       = zeros(1,number_of_points);
y       = zeros(1,number_of_points);
 prob    = rand(number_of_points,1);
for counter = 1:number_of_points
if prob(counter)<Prob1
   P=Mat1*P+Vector1;
elseif prob(counter)<Prob1+Prob2
   P=Mat2*P+Vector2;
elseif prob(counter)<Prob1+Prob2+Prob3
   P=Mat3*P+Vector3;
else
   P=Mat4*P+Vector4;
end
x(counter)=P(1);
y(counter)=P(2);
end
figure,plot(x,y,'r.')
axis equal, axis off
```

Making simple Movies



```
movie2avi(bullet_matrix, 'C:\Matlab_course\exp_mov_3_0', 'compression', 'None');
```

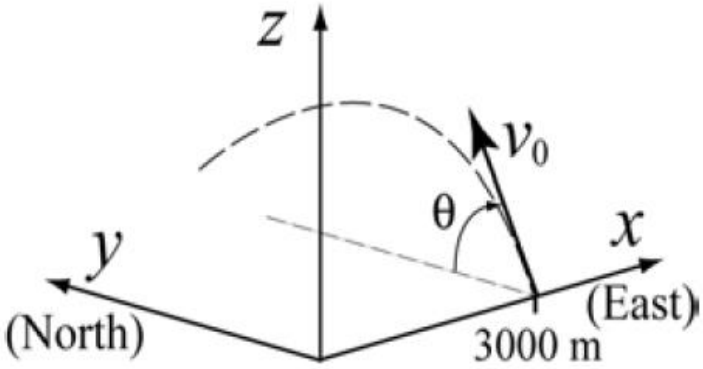


movie2avi(spring_matrix, 'C:\Matlab_course\exp_mov_3_1', 'compression', 'None','fps',15);

EXAMPLES OF MATLAB APPLICATIONS

Sample Problem : 3-D projectile trajectory

A projectile is fired with an initial velocity of 250 m/s at an angle of θ = 65° relative to the ground. The projectile is aimed directly north. Because of a strong wind blowing to the west, the projectile also moves in this direction at a constant speed of 30 m/s.



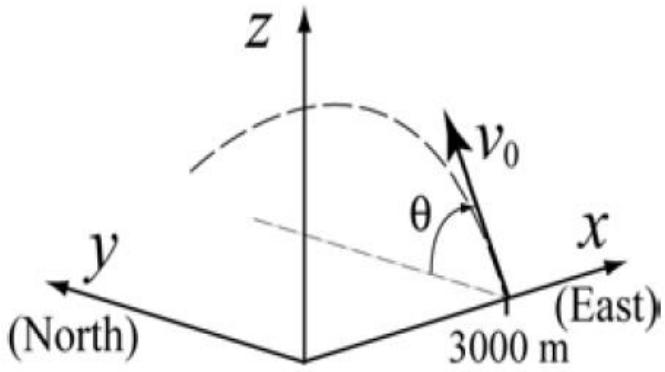
Determine and plot the trajectory of the projectile until it hits the ground. For comparison, plot also (in the same figure) the trajectory that the projectile would have had if there was no wind.

EXAMPLES OF MATLAB APPLICATIONS

Solution

Since the projectile is fired directly north, the initial v_0 velocity can be resolved into a horizontal y component and a vertical z component:

$$v_{0y} = v_0 \cos(\theta) \quad \text{and} \quad v_{0z} = v_0 \sin(\theta)$$



In addition, due to the wind the projectile has a constant velocity in the negative x direction, $v_x = 30$ m/s.

The initial position of the projectile (x0, y0, z0) is at point (3000, 0, 0). In the vertical direction the velocity and position of the projectile are given by

$$v_z = v_{0z} - gt \quad \text{and} \quad z = z_0 + v_{0z}t - \frac{1}{2}gt^2$$

The time it takes the projectile to reach the highest point (vz = 0) is $t_{hmax} = \frac{v_{0z}}{g}$.

The total flying time is twice this time, . In the horizontal direction the velocity is constant (both in the x and y directions), and the position of the projectile is given by:

$$x = x_0 + v_x t \quad \text{and} \quad y = y_0 + v_{0y}t$$

EXAMPLES OF MATLAB APPLICATIONS

```
v0=250; g=9.81; theta=65;
x0=3000; vx=-30;
v0z=v0*sin(theta*pi/180);
v0y=v0*cos(theta*pi/180);
t=2*v0z/g;
tplot=linspace(0,t,100);
z=v0z*tplot-0.5*g*tplot.^2;
y=v0y*tplot;
x=x0+vx*tplot;
xnowind(1:length(y))=x0;
plot3(x,y,z,'k-',xnowind,y,z,'k--')
grid on
axis([0 6000 0 6000 0 2500])
xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)')
```

Creating a time vector with 100 elements.

Calculating the x, y, and z coordinates of the projectile at each time.

Constant x coordinate when no wind.

Two 3-D line plots.

EXAMPLES OF MATLAB APPLICATIONS