MATLAB and Its Application in Engineering

Assoc. Prof. Kirin Shi

Shanghai Jiao Tong University

Review of Lecture 3

Relational & Logical operations

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.

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.

```
>> 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
>> t=r(s)
t =
     8     9     4    10
>> w=r(r<=10)
w =
     8     9     4    10
```

Define a vector r.

Checks which r elements are smaller than or equal to 10.

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

Use s for addresses in vector r to create vector t.

Vector t consists of elements of r in positions where s has 1s.

The same procedure can be done in one step.

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
```

Repetative Control Structures

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

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

>> mk1=V>0 & (rem(V,3)==0 | rem(V,5)==0)
 1    0    0    0    0    0    1    1    1    0    1    0    0    0

>> V(mk1)=2*V(mk1)
10  17   -3    8    0   -7   24   30   40   -6   12    4   -2   16

>> mk2=V< 0 & V>-5
 0    0    1    0    0    0    0    0    0    0    0    0    1    0

>> V(mk2)=V(mk2).^3
10  17  -27    8    0   -7   24   30   40   -6   12    4   -8   16
```

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.

3-D Line PLOTS

Function meshgrid

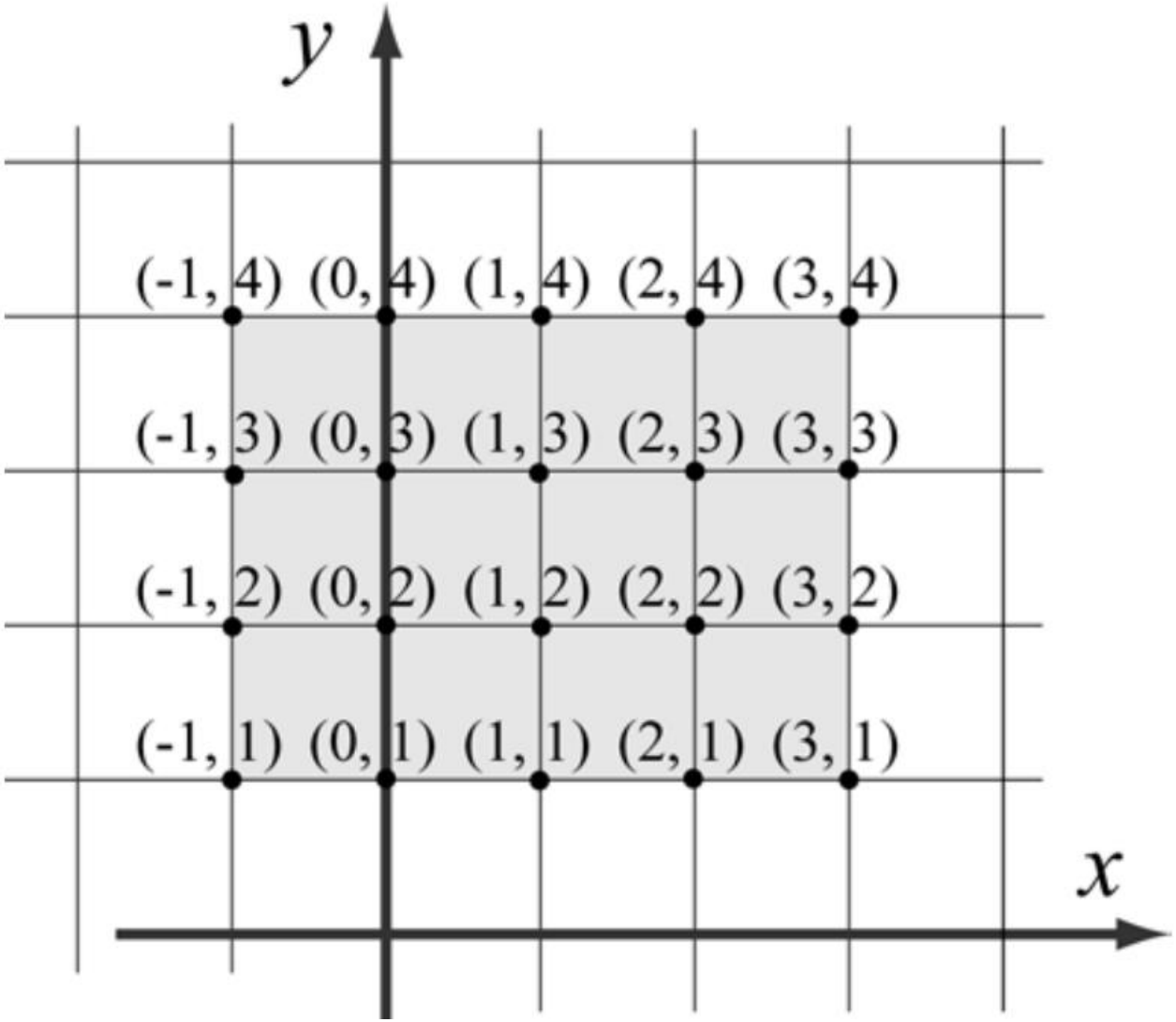
[x y] = meshgrid(xstart:xinc:xend,ystart:yinc:yend)

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
```

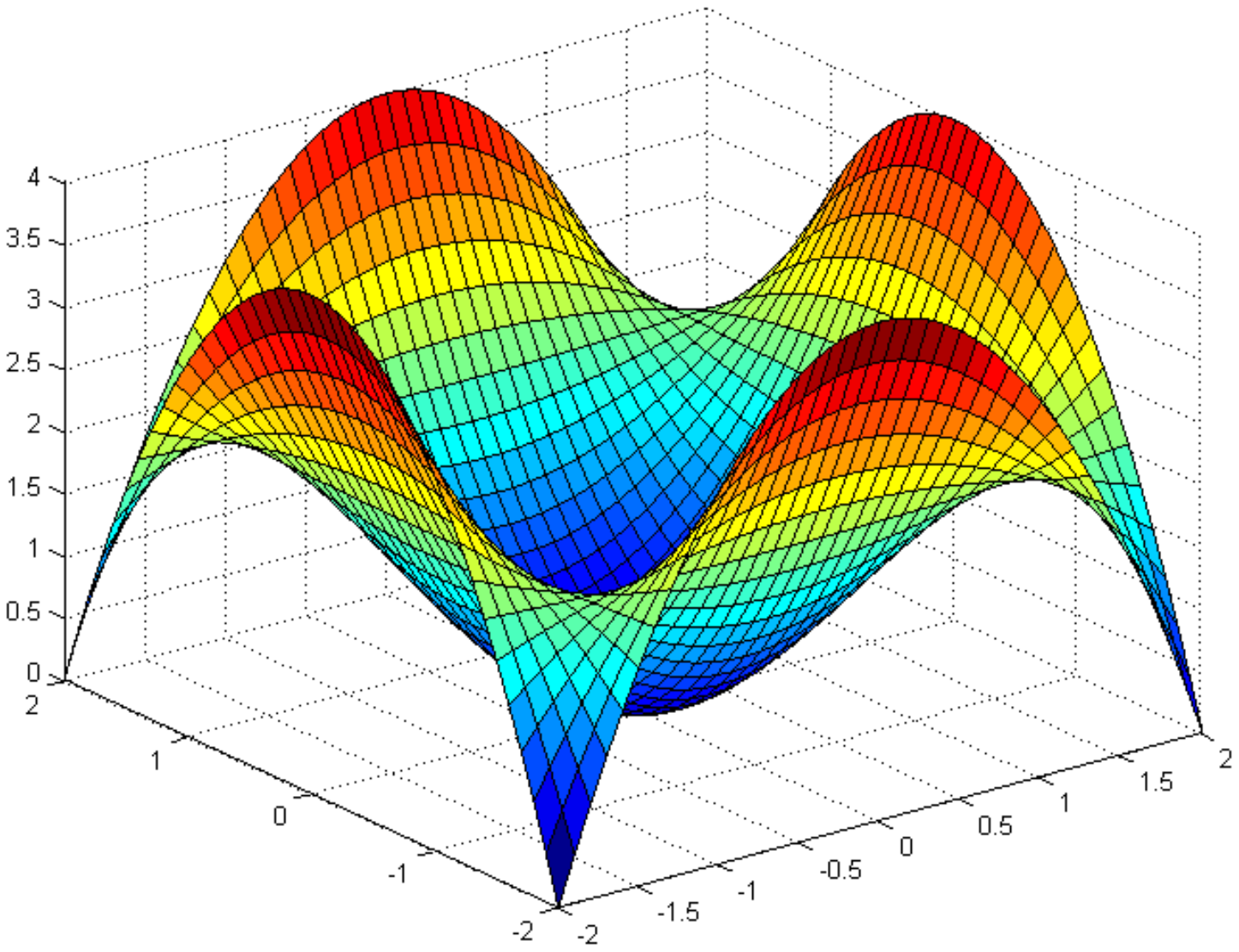
Coordinates of points by meshgrid



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



End of Review on Lecture 3

Symbolic math

Symbolic math

```
>> x=2
x =
2
```

```
>> y=x^2+1
y =
5
```

```
>> clear x
>> y=x^2+1
??? Undefined function or variable 'x'.
```

Can Matlab perform the following operation??

$$z = y^3 + 1 = (x^2 + 1)^3 + 1 = x^6 + 3x^4 + 3x^2 + 2$$

Symbolic math

Declaring symbolic variables

object_name = sym(‘string’)

```
>> a=sym('a');
>> b=sym('gamma');
>> c=sym(5);
>> d=5;

>> whos
  Name      Size            Bytes     Class

  a         1x1               112     sym
  b         1x1               112     sym
  c         1x1               112     sym
  d         1x1                 8     double
```

Symbolic math

Declaring symbolic variables

syms variable_name1 variable_name2 variable_name3

syms x a

```
>> f = (x-1)*(x-a)*(2*x^2+5)+ 20
```

syms without a list of variables, you would get the list of all the symbolic variables in your workspace

```
>> syms
    'a'    'f'    'x'
```

Symbolic math

When numerical variables are inserted in symbolic expressions their exact value is used, even if the variable was displayed before with an approximated value.

```
>> h=10/3
```

h is defined to be 10/3 (a numerical variable).

```
h =
    3.3333
```

An approximated value of h (numerical variable) is displayed.

```
>> k=sym(5); m=sym(7);
```

Define k and m as symbolic 5 and 7, respectively.

```
>> p=k/m+h
```

h, k, and m are used in an expression.

```
p =
85/21
```

The exact value of h is used in the determination of p.
An exact value of p (symbolic object) is displayed.

Symbolic math

The double(S) command can be used to convert a symbolic expression (object) S that is written in an exact form to numerical form.

```
>> pN=double(p)
pN =
    4.0476
>> y=sym(10)*cos(5*pi/6)
y =
-5*3^(1/2)
>> yN=double(y)
yN =
   -8.6603
```

p is converted to numerical form (assigned to pN).

Create a symbolic expression y.

Exact value of y is displayed.

y is converted to numerical form (assigned to yN).

Symbolic math

Existing symbolic expressions can be used to create new symbolic expressions.
This is done by simply using the name of the existing expression in the new expression.

```
>> syms x y
```
Define x and y as symbolic variables.

```
>> SA=x+y, SB=x-y
```
Create two symbolic expressions SA and SB.

```
SA =
x+y
SB =
x-y
```

$$SA = x + y$$

$$SB = x - y$$

```
>> F=SA^2/SB^3+x^2
```
Create a new symbolic expression F using SA and SB.

```
F =
(x+y)^2/(x-y)^3+x^2
```

$$F = (SA^2)/(SB^3) + x^2 = \frac{(x+y)^2}{(x-y)^3} + x^2$$

The findsym Command and the Default Symbolic Variable

The ***findsym*** command can be used to find which symbolic variables are present in an existing symbolic expression. The format of the command is:

`findsym(S)` or `findsym(S,n)`

The ***findsym(S)***command displays the names of all the symbolic variables (separated by commas) that are in the expression ***S*** in alphabetical order. The ***findsym(S,n)***command displays ***n*** symbolic variables that are in expression S in the default order.

For one-letter symbolic variables, the default order starts with x, and followed by letters, according to their closeness to x. If there are two letters equally close to x, the letter that is after x in alphabetical order is first (y before w, and z before v).

The default symbolic variable in a symbolic expression is the first variable in the default order. The default symbolic variable in an expression S can be identified by typing findsym(S,1).

Example of findsym Command

```
>> syms x h w y d t
```
Define x, h, w, y, d, and t as symbolic variables.

```
>> S=h*x^2+d*y^2+t*w^2
```
Create a symbolic expression S.

```
S =
t*w^2 + h*x^2 + d*y^2
```

```
>> findsym(S)
```
Use the findsym(S) command.

```
ans =
d, h, t, w, x, y
```
The symbolic variables are displayed in alphabetical order.

```
>> findsym(S,5)
```
Use the findsym(S,n) command (n = 5).

```
ans =
x,y,w,t,h
```
Five symbolic variables are displayed in the default order.

Symbolic math

The *collect* command:

Collects the terms in the expression that have the variable with the same power. In the new expression, the terms will be ordered in decreasing order of power.

```
>> syms x y
>> S=(x^2+x-exp(x))*(x+3)
S =
(x + 3)*(x - exp(x) + x^2)
>> F = collect(S)
F =
x^3+4*x^2+(3-exp(x))*x-3*exp(x)
```

Define x and y as symbolic variables.

Create the symbolic expression $(x+3)(x-e^x+x^2)$ and assign it to S.

Use the `collect` command.

MATLAB returns the expression: $x^3+4x^2+(3--e^x)x-3e^x$.

Symbolic math

The *collect* command:

Collects the terms in the expression that have the variable with the same power. In the new expression, the terms will be ordered in decreasing order of power.

```
>> T=(2*x^2+y^2)*(x+y^2+3)
T =
(2*x^2+y^2)*(y^2+x+3)
```

Create the symbolic expression T $(2x^2+y^2)(y^2+x+3)$.

```
>> G=collect(T)
```

Use the `collect(T)` command.

MATLAB returns the expression $2x^3+(2y^2+6)x^2+y^2x+y^2(y^2+3)$.

```
G =
2*x^3+(2*y^2+6)*x^2+y^2*x+y^2*(y^2+3)
>> H=collect(T,y)
```

Use the `collect(T,y)` command.

```
H =
y^4+(2*x^2+x+3)*y^2+2*x^2*(x+3)
```

MATLAB returns the expression $y^4+(2x^2+x+3)y^2+2x^2(x+3)$.

Symbolic math

The *expand* command:

```
>> syms x a
>> f = (x-1)*(x-a)*(2*x^2+5)+ 20
>> f_expand = expand(f)

f_expand =
2*x^4+5*x^2-2*x^3*a-5*x*a-2*x^3-5*x+2*x^2*a+5*a+20

>> a=2;
>> f_new = eval(f)
>> f_expand_new = eval(f_new)

f_expand_new =2*x^4+9*x^2-6*x^3-15*x+30
```

Symbolic math

The *factor* command:

The factor command changes an expression that is a polynomial to a product of polynomials of a lower degree.

```
>> syms x
```

Define x as a symbolic variable.

```
>> S=x^3+4*x^2-11*x-30
S =
x^3+4*x^2-11*x-30
>> factor(S)
ans =
(x+5)*(x-3)*(x+2)
```

Create the symbolic expression $x^3 + 4x^2 - 11x - 30$ and assign it to S.

Use the `factor` command.

MATLAB returns the expression $(x+5)(x-3)(x+2)$.

Symbolic math

The *simplify* command:

The simplify command uses mathematical operations to generate a simpler form of the expression

```
>> syms x y
>> S=(x^2+5*x+6)/(x+2)
S =
(x^2+5*x+6)/(x+2)
>> SA = simplify(S)
SA =
x+3
>> simplify((x+y)/(1/x+1/y))

ans =
x*y
```

Define x and y as symbolic variables.

Create the symbolic expression $(x^2 + 5x + 6)/(x + 2)$, and assign it to S.

Use the simplify command to simplify S.

MATLAB simplifies the expression to $x + 3$.

Simplify $(x + y)/\left(\frac{1}{x} + \frac{1}{y}\right)$.

MATLAB simplifies the expression to xy).

Symbolic math

The *simple* command:

The simple command finds the form of the expression with the fewest number of characters. In many cases this form is also the simplest.

When the command is executed, MATLAB creates several forms of the expression by applying the *collect, expand, factor, and simplify* commands, and other simplification functions. Then MATLAB returns the expression with the shortest form.

```
>> syms x
>> S=(x^3-4*x^2+16*x)/(x^3+64)
     S = (x^3-4*x^2+16*x)/(x^3+64)

>> F = simple(S)
     F =x/(x+4)

>> E=simplify(S)
     E =1 - 4/(x + 4)
```

Symbolic math

The *pretty* command:

The pretty command displays a symbolic expression in a format resembling the mathematical format in which expressions are generally typed.

```
>> syms a b c x
>> S=sqrt(a*x^2 + b*x + c)
S =
(a*x^2+b*x+c)^(1/2)
>> pretty(S)
                    2           1/2
                (a x  + b x + c)
```

Define a, b, c, and x as symbolic variables.

Create the symbolic expression $\sqrt{ax^2 + bx + c}$, and assign it to S.

The pretty command displays the expression in a math format.

Symbolic math

Solve algebraic equations

A single algebraic equation can be solved for one variable, and a system of equations can be solved for several variables with the solve function.

Solving a single equation:

```
>> S=x^2-x-6
S =
x^2-x-6
```

Create the symbolic expression $x^2 - x - 6$, and assign it to `S`.

```
>> k=solve(S)
```

Use the `solve(S)` command to solve $x^2 - x - 6 = 0$.

```
k =
 -2
  3
```

The equation has two solutions. They are assigned to `k`, which is a column vector with symbolic objects.

```
>> solve('cos(2*y)+3*sin(y)=2')
ans =
     pi/2
     pi/6
 (5*pi)/6
```

Use the `solve` command to solve $\cos(2y) + 3\sin(y) = 2$. (The equation is typed as a string in the command.)

The solution is assigned to `ans`.

Symbolic math

Solve algebraic equations

A single algebraic equation can be solved for one variable, and a system of equations can be solved for several variables with the solve function.

Solving a single equation:

```
>> T= a*x^2+5*b*x+20
T =
a*x^2+5*b*x+20
```

Create the symbolic expression $ax^2 + 5bx + 20$, and assign it to `T`.

```
>> solve(T)
ans =
 -(5*b+5^(1/2)*(5*b^2-16*a)^(1/2))/(2*a)
 -(5*b-5^(1/2)*(5*b^2-16*a)^(1/2))/(2*a)
```

Use the `solve(S)` command to solve $T = 0$.

The equation $T = 0$ is solved for the variable x, which is the default variable.

```
>> M = solve(T,a)
M =
-(5*b*x+20)/x^2
```

Use the `solve(eq,var)` command to solve $T = 0$.

The equation $T = 0$ is solved for the variable a.

Symbolic math

Solve algebraic equations

A single algebraic equation can be solved for one variable, and a system of equations can be solved for several variables with the solve function.

Solving a system of equations:

```
>> syms x y t
>> S=10*x+12*y+16*t;
>> [xt yt]=solve(S, '5*x-y=13*t')
xt =
2*t
yt =
-3*t
```

Define `x`, `y`, and `t` as symbolic variables.

Assign to `S` the expression $10x + 12y + 16t$.

Use the `solve` command to solve the system: $10x + 12y + 16t = 0$, $5x - y = 13t$

Output in a cell array with two cells named `xt` and `yt`.

The solutions for x and y are assigned to `xt` and `yt`, respectively.

Symbolic math

Solve algebraic equations

A single algebraic equation can be solved for one variable, and a system of equations can be solved for several variables with the solve function.

Solving a system of equations:

```
>> [tx yx]=solve(S,'5*x-y=13*t',y,t)
```

The variables for which the system is solved (y and t) are entered.

```
tx =
x/2
yx =
-(3*x)/2
```

The solutions for the variables for which the system is solved are assigned in alphabetical order. The first cell has the solution for t, and the second cell has the solution for y.

Sample Problem : Intersection of a circle and a line

Problem:

The equation of a circle in the *x y* plane with radius *R* and its center at point (2, 4) is given by $(x-2)^2 + (y-4)^2 = R^2$. The equation of a line in the plane is given by $y = \frac{x}{2} + 1$. Determine the coordinates of the points (as a function of *R*) where the line intersects the circle.

Solution:

The solution is obtained by solving the system of the two equations for x and y in terms of R. To show the difference in the output between using cell array and structure output forms of the solve command, the system is solved twice. The first solution has the output in a cell array:

Sample Problem : Intersection of a circle and a line

```
>> syms x y R
>> [xc,yc]=solve('(x-2)^2+(y-4)^2=R^2','y=x/2+1')

xc =
((4*R^2)/5 - 64/25)^(1/2) + 14/5
 14/5 - ((4*R^2)/5 - 64/25)^(1/2)
yc =
((4*R^2)/5 - 64/25)^(1/2)/2 + 12/5
 12/5 - ((4*R^2)/5 - 64/25)^(1/2)/2
```

The two equations are typed in the `solve` command.

Output in a cell array.

Output in a cell array with two cells named `xc` and `yc`. Each cell contains two solutions in a symbolic column vector.

The second solution has the output in a structure:

```
>> COORD=solve('(x-2)^2+(y-4)^2=R^2','y = x/2+1')

COORD =
    x: [2x1 sym]
    y: [2x1 sym]
>> COORD.x
ans =
((4*R^2)/5 - 64/25)^(1/2) + 14/5
14/5 - ((4*R^2)/5 - 64/25)^(1/2)
>> COORD.y
ans =
((4*R^2)/5 - 64/25)^(1/2)/2 + 12/5
12/5 - ((4*R^2)/5 - 64/25)^(1/2)/2
```

Output in a structure.

Output in a structure named `COORD` that has two fields, `x` and `y`. Each field is a 2 by 1 symbolic vector.

Type the address of the field `x`.

The content of the field (the solution for x) is displayed.

Type the address of the field `y`.

The content of the field (the solution for y) is displayed.

Differentiation

Symbolic differentiation can be carried out by

function **diff**(y,x,n)

calculate the 1st and 2nd derivative of a function
$y(x) = \sin(a*x) + 4x^5$ with respect to the variable x

```
>> syms a x

>> y_first_derivative = diff(y,x)

>> y_second_derivative = diff(y,x,2)

>> y_second_derivative_a = diff(y,a,2)
```

Differentiation

```
>> syms x y t
```
Define x, y, and t as symbolic variables.

```
>> S=exp(x^4);
```
Assign to S the expression e^{x^4}.

```
>> diff(S)
```
Use the diff(S) command to differentiate S.

```
ans =
4*x^3*exp(x^4)
```
The answer $4x^3e^{x^4}$ is displayed.

```
>> diff((1-4*x)^3)
```
Use the diff(S) command to differentiate $(1-4x)^3$.

```
ans =
-12*(1-4*x)^2
```
The answer $-12(1-4x)^2$ is displayed.

```
>> R=5*y^2*cos(3*t);
```
Assign to R the expression $5y^2\cos(3t)$.

```
>> diff(R)
```
Use the diff(R) command to differentiate R.

```
ans =
10*y*cos(3*t)
```
MATLAB differentiates R with respect to y (default symbolic variable); the answer $10y\cos(3t)$ is displayed.

```
>> diff(R,t)
```
Use the diff(R,t) command to differentiate R w.r.t. t.

```
ans =
-15*y^2*sin(3*t)
```
The answer $-15y^2\sin(3t)$ is displayed.

```
>> diff(S,2)
```
Use diff(S,2) command to obtain the second derivative of S.

```
ans =
12*x^2*exp(x^4)+16*x^6*exp(x^4)
```
The answer $12x^2e^{x^4} + 16x^6e^{x^4}$ is displayed.

Symbolic integration

Symbolic integration can be carried out by using the int command. The command can be used for determining indefinite integrals and definite integrals.

function **int**(f,x)

int(f,x,a,b)

$$A = \int f(x)dx \qquad A = \int_a^b f(x)dx$$

Integration can sometimes be a difficult task. A closed-form answer may not exist, or if it exists, MATLAB might not be able to find it. When that happens MATLAB returns *int(S)* and the message *Explicit integral could not be found*.

Numerical integration

- **trapz**, **quad** and **dblquad**
- **z = trapz (x,y)** computes the integral of y with respect to x using the **trapezoidal** rule

$$A = \int_1^3 \frac{\sin^2(x)}{x} dx$$

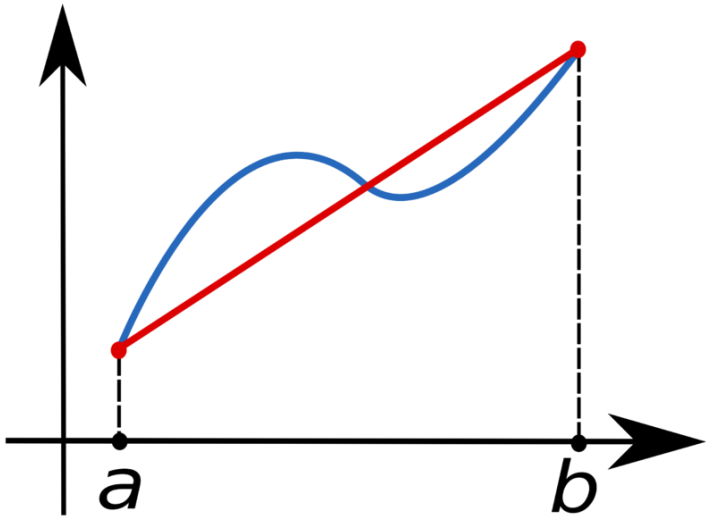
X = linspace (1,3,11);

Y = (sin(x).^2)./x;

A = trapz (x, y)

A =

0.7938



Effect of more intervals

```
x=linspace(1,3,21);
y=(sin(x).^2)./x;

A = trapz(x,y)
A =
0.7946
```

```
x=linspace(1,3,41);
y=(sin(x).^2)./x;

A = trapz(x,y)
A =
0.7948
```

```
>> syms x
>> Y = (sin(x).^2)./x;
>> int(Y,x,1,3)
>> cosint(2)/2 - cosint(6)/2 + log(3)/2
```

```
ans =

    0.7948
```

Effect of more intervals

```
tic
syms x
Y = (sin(x).^2)./x;
int(Y,x,1,3)
cosint(2)/2 - cosint(6)/2 + log(3)/2
toc
```

Elapsed time is **0.016324** seconds.

```
tic
x=linspace(1,3,41);
y=(sin(x).^2)./x;
A = trapz(x,y)
toc
```

Elapsed time is **0.004657** seconds.

Function quad

Numerically evaluating the integral of a function using adaptive **Simpson** quadrature, to within an error of 1.e-6.

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Q = quad (FUN, **lower_limit, upper_limit**)

A = quad ('sin(x).^2./x', 1, 3)

A =

0.7948

B = quad ('exp(-x.^2)', 0, 1)

B =

0.7468

$$A = \int_1^3 \frac{\sin^2(x)}{x}\,dx;$$

$$B = \int_0^1 e^{-x^2}\,dx$$

Effect of more intervals

tic

A = quad ('sin(x).^2./x', 1, 3)

toc

A= 0.7948

Elapsed time is **0.006759** seconds.

tic

x=linspace(1,3,**41**);

y=(sin(x).^2)./x;

A = trapz(x,y)

toc

Elapsed time is **0.004657** seconds.

tic

x=linspace(1,3,**21**);

y=(sin(x).^2)./x;

A = trapz(x,y)

toc

A= 0.7946

Elapsed time is **0.003137** seconds.

Function dblquad

numerically evaluate a double integral using adaptive Simpson quadrature.

$$A = \int_{-1}^{1} \int_{-1}^{5} x^2 y^3 \, dx \, dy$$

Q = dblquad(FUN, x1, x2, y1, y2)

A = dblquad ('**x.^2.*y.^3**', **-1, 1, -1, 5**)

A =

104.0000

quadgk: adaptive Gauss-Kronrod quadrature
quad2d: Numerically evaluate integral, adaptive Lobatto quadrature

Ordinary differential equation-symbolic

An ordinary differential equation (ODE) can be solved symbolically with the dsolve command.

In specifying the equation the letter `D` denotes differentiation. If y is the dependent variable and t is the independent variable, `Dy` stands for $\frac{dy}{dt}$. For example, the equation $\frac{dy}{dt} + 3y = 100$ is typed in as `'Dy + 3*y = 100'`.

A second derivative is typed as `D2`, third derivative as `D3`, and so on. For example, the equation $\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + 5y = \sin(t)$ is typed in as: `'D2y + 3*Dy + 5*y = sin(t)'`.

Ordinary differential equation-symbolic

An ordinary differential equation (ODE) can be solved symbolically with the dsolve command.

For example, a general solution of the first-order ODE $\frac{dy}{dt} = 4t + 2y$ is obtained by:

```
>> dsolve('Dy=4*t+2*y')

ans =
C1*exp(2*t) - 2*t - 1
```

The answer $y = C_1 e^{2t} - 2t - 1$ is displayed.

A general solution of the second-order ODE $\frac{d^2x}{dt^2} + 2\frac{dx}{dt} + x = 0$ is obtained by:

```
>> dsolve('D2x+2*Dx+x=0')
```

```
ans =
C1/exp(t)+(C2*t)/exp(t)
```

The answer $x = C_1 e^{-t} + C_2 t e^{-t}$ is displayed.

ordinary differential equation-symbolic

An ordinary differential equation (ODE) can be solved symbolically with the dsolve command.

```
>> dsolve('Ds=a*x^2')

ans =
a*t*x^2 + C1
```

The independent variable is t (default). MATLAB solves the equation $\frac{ds}{dt} = ax^2$.

The solution $s = ax^2t + C_1$ is displayed.

```
>> dsolve('Ds=a*x^2','x')

ans =
(a*x^3)/3 + C1
```

The independent variable is defined to be x. MATLAB solves the equation $\frac{ds}{dx} = ax^2$.

The solution $s = \frac{1}{3}ax^3 + C_1$ is displayed.

```
>> dsolve('Ds=a*x^2','a')

ans =
(a^2*x^2)/2 + C2
```

The independent variable is defined to be a. MATLAB solves the equation $\frac{ds}{da} = ax^2$.

The solution $s = \frac{1}{2}a^2x^2 + C_1$ is displayed.

Ordinary differential equation-symbolic

A particular solution of an ODE can be obtained if boundary conditions are specified.

For example, the first-order ODE $\frac{dy}{dt} + 4y = 60$, with the initial condition $y(0) = 5$ is solved with MATLAB by:

```
>> dsolve('Dy+4*y=60','y(0)=5')
ans =
15 - 10/exp(4*t)
```

The answer $y = 15 - (10/e^{4t})$ is displayed.

The second-order ODE $\frac{d^2y}{dt^2} - 2\frac{dy}{dt} + 2y = 0$, $y(0) = 1$, $\left.\frac{dy}{dt}\right|_{t=0} = 0$, can be solved

If MATLAB cannot find a solution, it returns an empty

symbolic object and the message Warning:

explicit solution could not be found.

Ordinary differential equation-numerical

Only a limited number of differential equations can be solved analytically. Numerical methods, on the other hand, can result in an approximate solution to almost any equation.

Steps for solving a single first-order ODE:

Step 1: Write the problem in a standard form.

$$\frac{dy}{dt} = f(t, y) \quad \text{for} \quad t_0 \le t \le t_f, \ \text{with} \quad y = y_0 \ \text{at} \ t = t_0.$$

Step 2: Create a user-defined function.

```
function dydt=ODEexp1(t,y)
dydt=(t^3-2*y)/t;
```

Ordinary differential equation-numerical

Only a limited number of differential equations can be solved analytically. Numerical methods, on the other hand, can result in an approximate solution to almost any equation.

Steps for solving a single first-order ODE:

Step 3: Select a method of solution.

ODE Solver Name	Description
`ode45`	For nonstiff problems, one-step solver, best to apply as a first try for most problems. Based on explicit Runge-Kutta method.
`ode23`	For nonstiff problems, one-step solver. Based on explicit Runge-Kutta method. Often quicker but less accurate than `ode45`.
`ode113`	For nonstiff problems, multistep solver.

Ordinary differential equation-numerical

Only a limited number of differential equations can be solved analytically. Numerical methods, on the other hand, can result in an approximate solution to almost any equation.

Steps for solving a single first-order ODE:

Step 3: Select a method of solution.

Stiff problems are ones that include fast and slowly changing components and require small time steps in their solution. One-step solvers use information from one point to obtain a solution at the next point.

Multistep solvers use information from several previous points to find the solution at the next point.

Ordinary differential equation-numerical

Only a limited number of differential equations can be solved analytically. Numerical methods, on the other hand, can result in an approximate solution to almost any equation.

Steps for solving a single first-order ODE:

Step 3: Select a method of solution.

It is impossible to know ahead of time which solver is the most appropriate for a specific problem. A suggestion is to first try ode45, which gives good results for many problems.

If a solution is not obtained because the problem is stiff, trying the solver ode15s is suggested.

Ordinary differential equation-numerical

Only a limited number of differential equations can be solved analytically. Numerical methods, on the other hand, can result in an approximate solution to almost any equation.

Steps for solving a single first-order ODE:

Step 3: Select a method of solution.

Step 4: Solve the ODE.

```
[t,y] = solver_name(ODEfun,tspan,y0)
```

Ordinary differential equation-numerical

For example, consider the solution to the problem stated in Step 1:

$$\frac{dy}{dt} = \frac{t^3 - 2y}{t} \quad \text{for} \quad 1 \le t \le 3 \quad \text{with} \quad y = 4.2 \quad \text{at} \quad t = 1,$$

```
>> [t y]=ode45(@ODEexp1,[1:0.5:3],4.2)
t =
    1.0000
    1.5000
    2.0000
    2.5000
    3.0000
y =
    4.2000
    2.4528
    2.6000
    3.7650
    5.8444
```

The initial value.

The vector `tspan`.

The handle of the user-defined function `ODEexp1`.

PLOTTING SYMBOLIC EXPRESSIONS

In many cases, there is a need to plot a symbolic expression. This can easily be done with the ***ezplot*** command. For a symbolic expression ***S*** that contains one variable ***var***, MATLAB considers the expression to be a function ***S(var)*** , and the command creates a plot of ***S(var)*** versus ***var***. For a symbolic expression that contains two symbolic variables ***var1*** and ***var2***, MATLAB considers the expression to be a function in the form ***S(var1,var2) = 0***, and the command creates aplot of one variable versus the other.

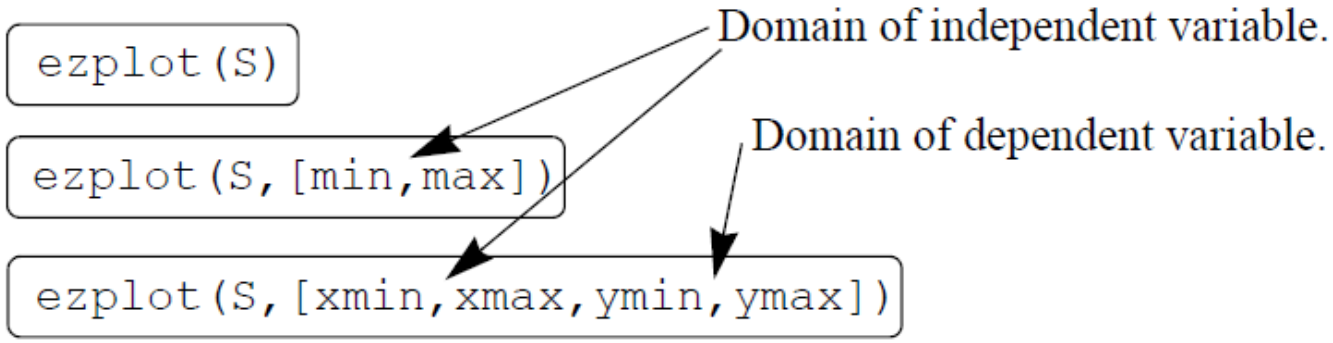
To plot a symbolic expression S that contains one or two variables, the ezplot command is:

```
ezplot(S)
or  ezplot(S,[min,max])
or  ezplot(S,[xmin,xmax,ymin,ymax])
```

Domain of independent variable.

Domain of dependent variable.

PLOTTING SYMBOLIC EXPRESSIONS



- ***S*** is the symbolic expression to be plotted. It can be the name of a previously created symbolic expression, or an expression can be typed in for ***S***.
- It is also possible to type the expression to be plotted as a string without having the variables in the expression first created as symbolic objects.
- If S has one symbolic variable, a plot of ***S(var)*** versus ***var*** is created, with the values of ***var*** (the independent variable) on the abscissa (horizontal axis), and the values of ***S(var)*** on the ordinate (vertical axis).
- If the symbolic expression ***S*** has two symbolic variables, ***var1*** and ***var2***, the expression is assumed to be a function with the form ***S(var1,var2) = 0***.

PLOTTING SYMBOLIC EXPRESSIONS

- In the ezplot(S) command, if S has one variable (S(var)), the plot is over the domain ***–2π < var < 2π*** (default domain) and the range is selected by MATLAB.If S has two variables (S(var1,var2)), the plot is over ***–2π < var1 < 2π*** and ***π***.
- In the ezplot(S,[min,max]) command the domain for the independent variable is defined by min and max:— min < var < max—and the range is selected by MATLAB.
- In the ***ezplot(S,[xmin,xmax,ymin,ymax])*** command the domain for the independent variable is defined by ***xmin*** and ***xmax***, and the domain of the dependent variable is defined by ***ymin*** and ***ymax***.

PLOTTING SYMBOLIC EXPRESSIONS

The ezplot command can also be used to plot a function that is given in a parametric form. In this case two symbolic expressions, S1 and S2, are involved, where each expression is written in terms of the same symbolic variable (independent parameter).
For example, for a plot of y versus x where ***x = x(t)*** and ***y = y(t)*** , the form of the ezplot command is:

```
ezplot(S1,S2)
```

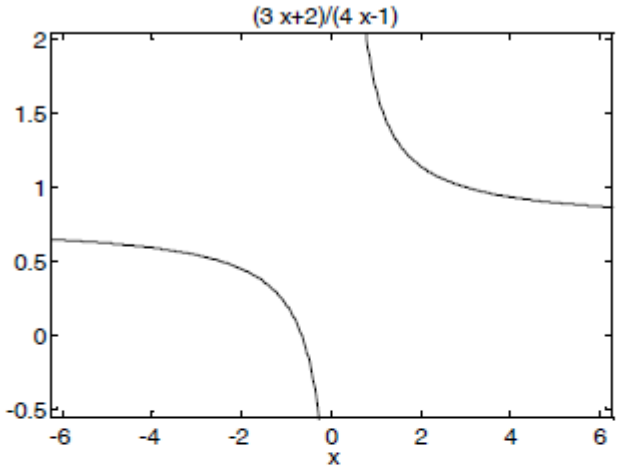
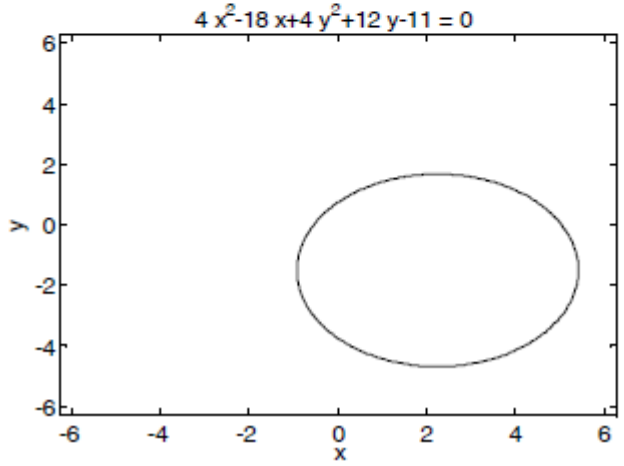
or

```
ezplot(S1,S2,[min,max])
```

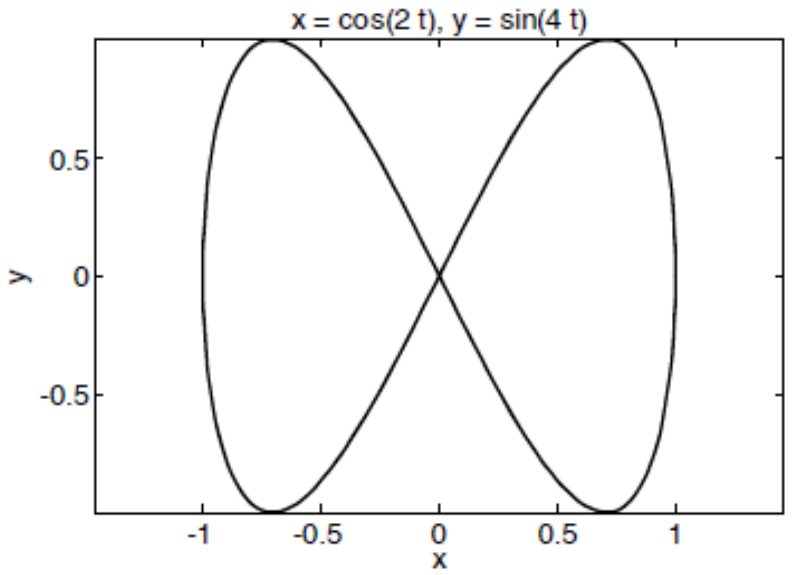
Domain of independent parameter.

- S1 and S2 are symbolic expressions containing the same single symbolic variable, which is the independent parameter.
- The command creates a plot of ***S2(var)*** versus ***S1(var).*** The symbolic expression that is typed first in the command (S1 in the definition above) is used for the horizontal axis, and the expression that is typed second (S2 in the definition above) is used for the vertical axis.
- In the ezplot(S1,S2)command the domain of the independent variable is ***0 < var < 2π*** (default domain).
- In the ezplot(S1,S2,[min,max])command the domain for the independent variable is defined by min and max:***min < var < max*** .

Plots with the ezplot command

Command	Plot
>> syms x >> S=(3*x+2)/(4*x-1) S = (3*x+2)/(4*x-1) >> ezplot(S)	
>> syms x y >> S=4*x^2-18*x+4*y^2+12*y-11 S = 4*x^2-18*x+4*y^2+12*y-11 >> ezplot(S)	

Plots with the ezplot command

Command	Plot
>> syms t >> x=cos(2*t) x = cos(2*t) >> y=sin(4*t) y = sin(4*t) >> ezplot(x,y)	

NUMERICAL CALCULATIONS WITH SYMBOLIC EXPRESSIONS

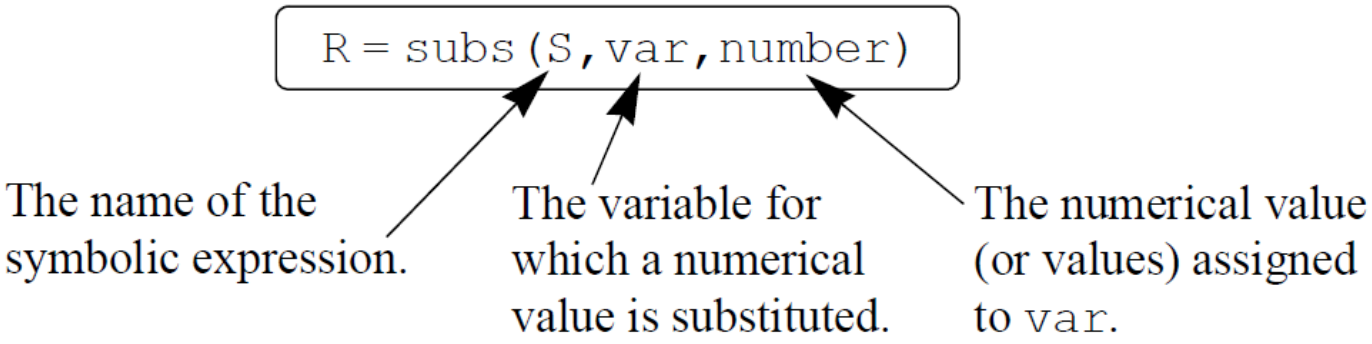
There may be a need to substitute numbers for the symbolic variables and calculate the numerical value of the expression. This can be done by using the ***subs*** command. The subs command has several forms and can be used in different ways. The following describes several forms that are easy to use and are suitable for most applications.

In one form, the variable (or variables) for which a numerical value is substituted and the numerical value itself are typed inside the subs command. ***In another form***, each variable is assigned a numerical value in a separate command and then the variable is substituted in the expression.

The subs command in which the variable and its value are typed inside the command is shown first. Two cases are presented—one for substituting a numerical value (or values) for one symbolic variable, and the other for substituting numerical values for two or more symbolic variables.

Substituting a numerical value for one symbolic variable

A numerical value (or values) can be substituted for one symbolic variable when a symbolic expression has one or more symbolic variables. In this case the subs command has the form:



number can be one number (a scalar), or an array with many elements (a vectoror a matrix).

The value of ***S*** is calculated for each value of number and the result is assigned to R, which will have the same size as number (scalar, vector, or matrix).

If S has one variable, the output ***R*** is numerical. If ***S*** has several variables and a numerical value is substituted for only one of them, the output ***R*** is a symbolic expression.

Substituting a numerical value for one symbolic variable

Example:

```
>> syms x
>> S=0.8*x^3+4*exp(0.5*x)
S =
4*exp(x/2) + (4*x^3)/5
>> SD=diff(S)
SD =
2*exp(x/2)+(12*x^2)/5)
>> subs(SD, x, 2)
ans =
   15.0366
>> SDU=subs(SD, x, [2:0.5:4])
SDU =
   15.0366   21.9807   30.5634   40.9092   53.1781
```

Define x as a symbolic variable.

Assign to S the expression $0.8x^3 + 4e^{(0.5x)}$.

Use the diff(S) command to differentiate S.

The answer $2e^{x/2} + 12x^2/5$ is assigned to SD.

Use the subs command to substitute $x = 2$ in SD.

The value of SD is displayed.

Use the subs command to substitute $x = [2, 2.5, 3, 3.5, 4]$ (vector) in SD.

The values of SD (assigned to SDU) for each value of x are displayed in a vector.

Substituting a numerical value for one symbolic variable

Example:

```
>> syms a g t v
>> Y=v^2*exp(a*t)/g
Y =
v^2*exp(a*t)/g
>> subs(Y,t,2)
ans =
v^2*exp(2*a)/g
>> Yt=subs(Y,t,[2:4])
Yt =
[ v^2*exp(2*a)/g, v^2*exp(3*a)/g, v^2*exp(4*a)/g]
```

Define a, g, t, and v as symbolic variables.

Create the symbolic expression $v^2e^{(at)}/g$ and assign it to Y.

Use the subs command to substitute $t = 2$ in SD.

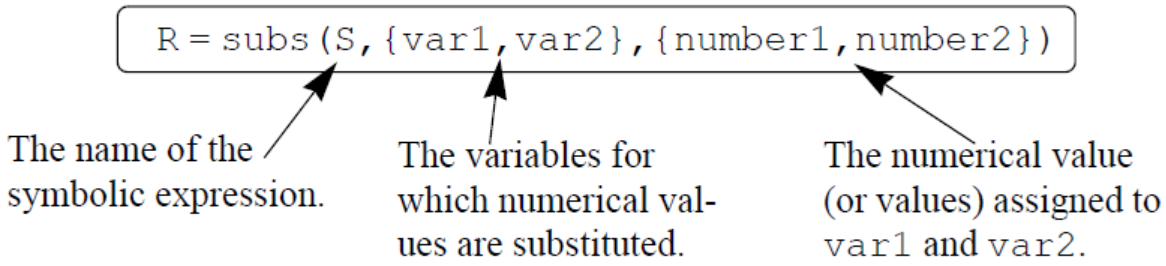
The answer $v^2e^{(2a)}/g$ is displayed.

Use the subs command to substitute $t = [2, 3, 4]$ (vector) in Y.

The answer is a vector with elements of symbolic expressions for each value of t.

Substituting a numerical value for two or more symbolic variables

A numerical value (or values) can be substituted for two or more symbolic variables when a symbolic expression has several symbolic variables. In this case the subs command has the following form (it is shown for two variables, but it can be used in the same form for more):



- The variables ***var1*** and ***var2*** are the variables in the expression ***S*** for which the numerical values are substituted. The variables are typed as a cell array (inside curly braces { }). A cell array is an array of cells where each cell can be an array of numbers or text.
- The numbers ***number1***,***number2*** substituted for the variables are also typed as a cell array (inside curly braces { }). The numbers can be scalars, vectors, or matrices. The first cell in the numbers cell array (***number1***) is substituted for the variable that is in the first cell of the variable cell array (***var1***), and so on.
- If all the numbers that are substituted for variables are scalars, the outcome will be one number or one expression (if some of the variables are still symbolic).

Substituting a numerical value for two or more symbolic variables

```
>> syms a b c e x
```
Define a, b, c, e, and x as symbolic variables.

```
>> S=a*x^e+b*x+c

S =
a*x^e+b*x+c
```
Create the symbolic expression $ax^e + bx + c$ and assigned it to S.

```
>> subs(S,{a,b,c,e,x},{5,4,-20,2,3})
```
Substitute in S scalars for all the symbolic variables.

Cell array. Cell array.

```
ans =
    37
```
The value of S is displayed.

```
>> T=subs(S,{a,b,c},{6,5,7})

T =
5*x+ 6*x^e+7
```
Substitute in S scalars for the symbolic variables a, b, and c.

The result is an expression with the variables x and e.

```
>> R=subs(S,{b,c,e},{[2 4 6],9,[1 3 5]})

R =
[  2*x+a*x+9, a*x^3+4*x+9, a*x^5+6*x+9]
```
Substitute in S a scalar for c, and vectors for b and e.

The result is a vector of symbolic expressions.

```
>> W=subs(S,{a,b,c,e,x},{[4 2 0],[2 4 6],[2 2 2],[1 3 5],[3 2 1]})

W =
    20    26     8
```
Substitute in S vectors for all the variables.

The result is a vector of numerical values.

Substituting a numerical value for two or more symbolic variables

A second method for substituting numerical values for symbolic variables in a symbolic expression is to first assign numerical values to the variables and then use the subs command. In this method, once the symbolic expression exists (at which point the variables in the expression are symbolic) the variables are assigned numerical values. Then the subs command is used in the form:

Once the symbolic variables are redefined as numerical variables they can no longer be used as symbolic. The method is demonstrated in the following examples.

Substituting a numerical value for two or more symbolic variables

```
>> syms A c m x y
```
Define A, c, m, x, and y as symbolic variables.

```
>> S=A*cos(m*x)+c*y
S =
c*y+A*cos(m*x)
```
Create the symbolic expression $A\cos(mx)+cy$ and assign it to S.

```
>> A=10; m=0.5; c=3;
```
Assign numerical values to variables A, m, and c.

```
>> subs(S)
ans =
3*y + 10*cos(x/2)
```
Use the subs command with the expression S.

The numerical values of variables A, m, and c are substituted in S.

```
>> x=linspace(0,2*pi,4);
```
Assign numerical values (vector) to variable x.

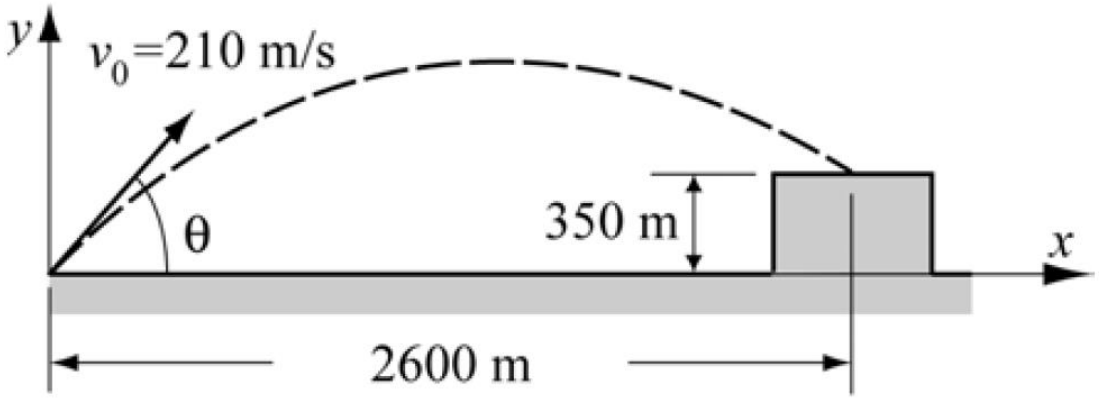
```
>> T = subs(S)
T =
[ 3*y+10, 3*y+5, 3*y-5, 3*y-10]
```
Use the subs command with the expression S.

The numerical values of variables A, m, c, and x are substituted. The result is a vector of symbolic expressions.

Sample Problem : Firing angle of a projectile

Problem:

A projectile is fired at a speed of 210 m/s and an angle θ. The projectile's intended target is 2,600 m away and 350m above the firing point.



(a) Derive the equation that has to be solved in order to determine the angle θ such that the projectile will hit the target.

(*b*) Use MATLAB to solve the equation derived in part (*a*).

(*c*) For the angle determined in part (*b*), use the ezplot command to make a plot of the projectile's trajectory.

Sample Problem : Firing angle of a projectile

Solution:

(a)The motion of the projectile can be analyzed by considering the horizontal and vertical components. The initial velocity can be resolved into horizontal and vertical components:

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

In the horizontal direction the velocity is constant, and the position of the projectile as a function of time is given by:

$$x = v_{0x} t$$

Substituting x = 2600 m for the horizontal distance that the projectile travels to reach the target and 210cos(θ) for v0x , and solving for t gives:

$$t = \frac{2600}{210 \cos(\theta)}$$

In the vertical direction the position of the projectile is given by:

$$y = v_{0y} t - \frac{1}{2} g t^2$$

Substituting *y* = 350 m for the vertical coordinate of the target, 210sin(θ) for v_{0x} , *g* = 9.81and *t* gives:

$$350 = 210 \sin(\theta) \frac{2600}{210 \cos(\theta)} - \frac{1}{2} 9.81 \left(\frac{2600}{210 \cos(\theta)} \right)^2 \quad \text{or} \quad 350 = \frac{2600 \sqrt{1 - \cos^2(\theta)}}{\cos(\theta)} - \frac{1}{2} 9.81 \left(\frac{2600}{210 \cos(\theta)} \right)^2$$

Sample Problem : Firing angle of a projectile

Solution:

(b) A solution of the equation derived in part (a) obtained by using the solve command (in the Command Window) is:

```
>> syms th
Angle = solve('2600*sqrt(1 - cos(th)^2)/cos(th) - 0.5*9.81*(2600/
(210*cos(th)))^2 = 350')
Angle =
    1.2453544972374161683138135806 56
  0.4592528070320712127778645203 7279
 -0.4592528070320712127778645203 7279
   -1.2453544972374161683138135806 56
```

MATLAB displays four solutions. The two positive ones are relevant to the problem.

```
>> Angle1 = Angle(1)*180/pi
```

Converting the solution in the first element of `Angle` from radians to degrees.

```
Angle1 =
224.16380950273491029648644451808/pi
```

MATLAB displays the answer as a symbolic object in terms of π.

```
>> Angle1=double(Angle1)
Angle1 =
   71.3536
```

Use the `double` command to obtain numerical values for `Angle1`.

```
>> Angle2=Angle(2)*180/pi
```

Converting the solution in the second element of `Angle` from radians to degrees.

```
Angle2 =
82.665505265772818300015613667102/pi
```

MATLAB displays the answer as a symbolic object in terms of π.

```
>> Angle2=double(Angle2)
Angle2 =
   26.3132
```

Use the `double` command to obtain numerical values for `Angle2`.

Sample Problem : Firing angle of a projectile

Solution:

(c) The solution from part (b) shows that there are two possible angles and thus two trajectories. In order to make a plot of a trajectory, the x and y coordinates of the projectile are written in terms of t (parametric form):

$$x = v_0 \cos(\theta) t \quad \text{and} \quad y = v_0 \sin(\theta) t - \frac{1}{2} g t^2$$

The domain for t is:

$$t = 0 \text{ to } t = \frac{2600}{210 \cos(\theta)}$$

These equations can be used in the ezplot command to make the plots shown in the following program written in a script file.

Sample Problem : Firing angle of a projectile

Solution:

```
xmax=2600; v0=210; g=9.81;
theta1=1.24535; theta2=.45925;
t1=xmax/(v0*cos(theta1));
t2=xmax/(v0*cos(theta2));
syms t
X1=v0*cos(theta1)*t;
X2=v0*cos(theta2)*t;
Y1=v0*sin(theta1)*t-0.5*g*t^2;
Y2=v0*sin(theta2)*t-0.5*g*t^2;
ezplot(X1,Y1,[0,t1])
hold on
ezplot(X2,Y2,[0,t2])
hold off
```

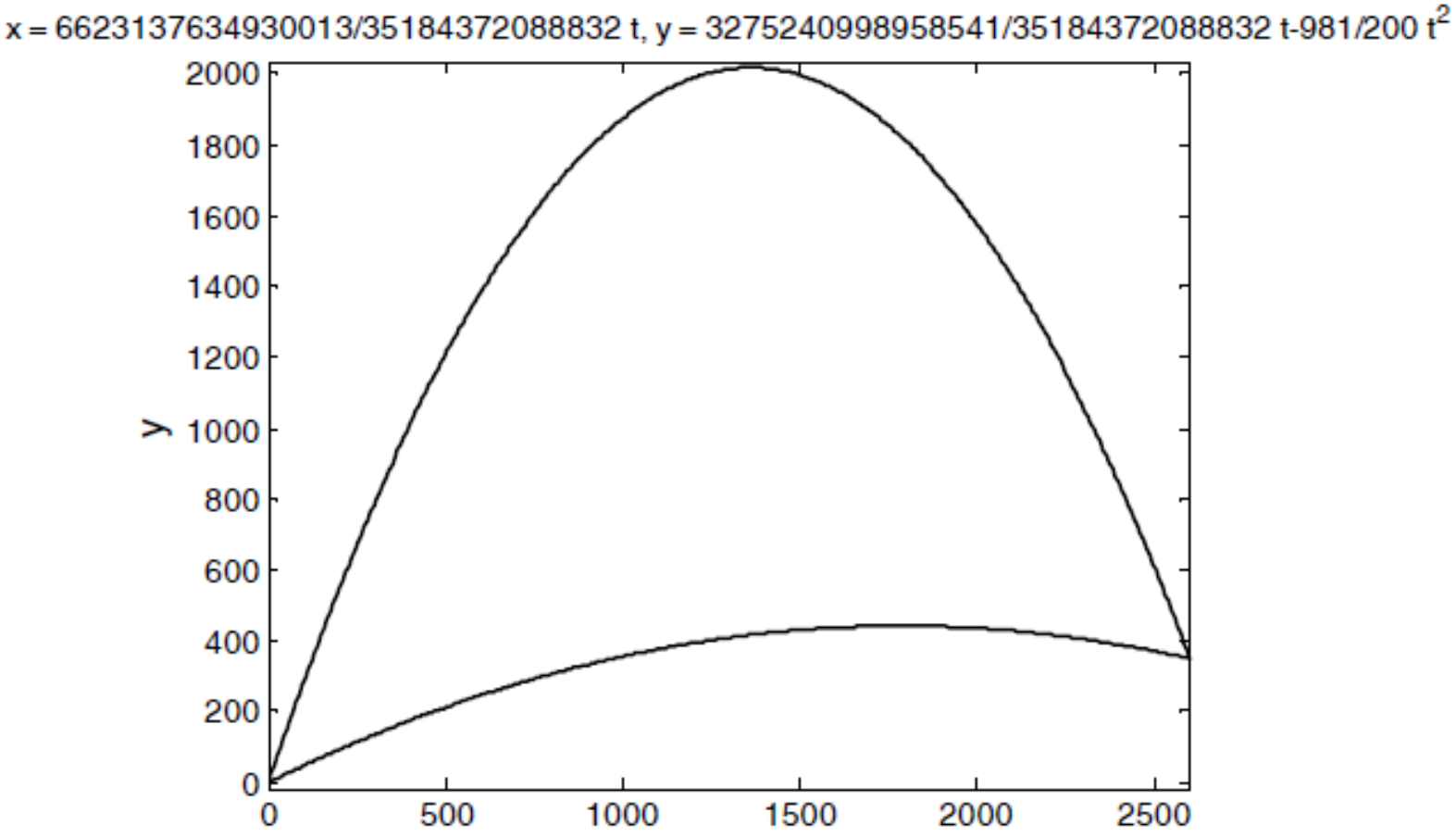
Assign the two solutions from part (*b*) to `theta1` and `theta2`.

Plot one trajectory.

Plot a second trajectory.

Sample Problem : Firing angle of a projectile

Solution:



End of Symbolic math

Polynomials

POLYNOMIALS

Many physical and engineering problems lead to polynomial functions.

A polynomial function of degree ***n*** has the general form

$$y = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

A solution or root of the equation is a number x_i, real or complex, such that

$$a_n x_i^n + a_{n-1} x_i^{n-1} + ... + a_1 x_i + a_0 = 0$$

Fundamental theorem of algebra

- A polynomial of degree ***n*** has exactly ***n*** roots.
- If these ***n*** roots are denoted by $x_1, x_2, \ldots, x_n$, then the polynomial function **y** can be described as

$$y = a_n x^n + a_{n-1} x^{n-1} + \ldots\ldots + a_1 x + a_0$$

$$y = a_n (x - x_1)(x - x_2) \ldots\ldots (x - x_{n-1})(x - x_n)$$

Representation of polynomials

- Represented by the vector of its coefficients, in descending powers of **x**.
- The length of the coefficient vector is always equal to the order of the polynomial **plus one**.
- For example,

$$x^2 + 2x + 1$$

$$2x^3 - 15x - 12$$

$$2x^3 - 15x$$

- **For example**

```
>> coef_poly_1 = [1  2  1]
coef_poly_1 =
1     2     1
>> coef_poly_2 = [2  0  -15  -12]
coef_poly_2 =
2     0   -15   -12
>> coef_poly_3 = [2  0  -15  0]
coef_poly_3 =
2     0   -15     0
```

$$x^2 + 2x + 1$$

$$2x^3 - 15x - 12$$

$$2x^3 - 15x$$

- **For example**

Polynomial	**MATLAB representation**
$8x + 5$	p = [8 5]
$2x^2 - 4x + 10$	d = [2 –4 10]
$6x^2 - 150$, MATLAB form: $6x^2 + 0x - 150$	h = [6 0 –150]
$5x^5 + 6x^2 - 7x$, MATLAB form: $5x^5 + 0x^4 + 0x^3 + 6x^2 - 7x + 0$	c = [5 0 0 6 –7 0]

Finding the values of a polynomial

- Ordinary procedure

  For example, the values of the function

$$y = x^5 - 2x^4 + 2x^3 + 3x^2 + 4$$

  for x = -2:0.1:2

>> x = -2:0.1:2;

>> y = x.^5 -2*x.^4 + 2*x.^3 +3*x.^2 + 4;

function polyval

$$y = x^5 - 2x^4 + 2x^3 + 3x^2 + 4$$

```
>> coef_poly_3 = [1  -2  2  3  0  4];

>> x = -2:0.1:2;

>> y = polyval(coef_poly_3,x);
```

Sample Problem : Calculating polynomials with MATLAB

Sample Problem 8-1: Calculating polynomials with MATLAB

For the polynomial $f(x) = x^5 - 12.1x^4 + 40.59x^3 - 17.015x^2 - 71.95x + 35.88$:

(*a*) Calculate $f(9)$.

(*b*) Plot the polynomial for $-1.5 \le x \le 6.7$.

Solution

The problem is solved in the Command Window.

(*a*) The coefficients of the polynomials are assigned to vector `p`. The function `polyval` is then used to calculate the value at $x = 9$.

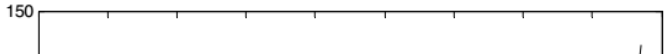
```
>> p = [1 -12.1 40.59 -17.015 -71.95 35.88];
>> polyval(p,9)
ans =
  7.2611e+003
```

(*b*) To plot the polynomial, a vector `x` is first defined with elements ranging from –1.5 to 6.7. Then a vector `y` is created with the values of the polynomial for every element of `x`. Finally, a plot of `y` vs. `x` is made.

```
>> x=-1.5:0.1:6.7;
>> y=polyval(p,x);
>> plot(x,y)
```

Calculating the value of the polynomial for each element of the vector `x`.

The plot created by MATLAB is presented below (axis labels were added with the Plot Editor).



Sample Problem : Calculating polynomials with MATLAB

Problem

For the polynomial $f(x) = x^5 - 12.1x^4 + 40.59x^3 - 17.015x^2 - 71.95x + 35.88$

(a) Calculate f(9).
(b) Plot the polynomial for $-1.5 \le x \le 6.7$.

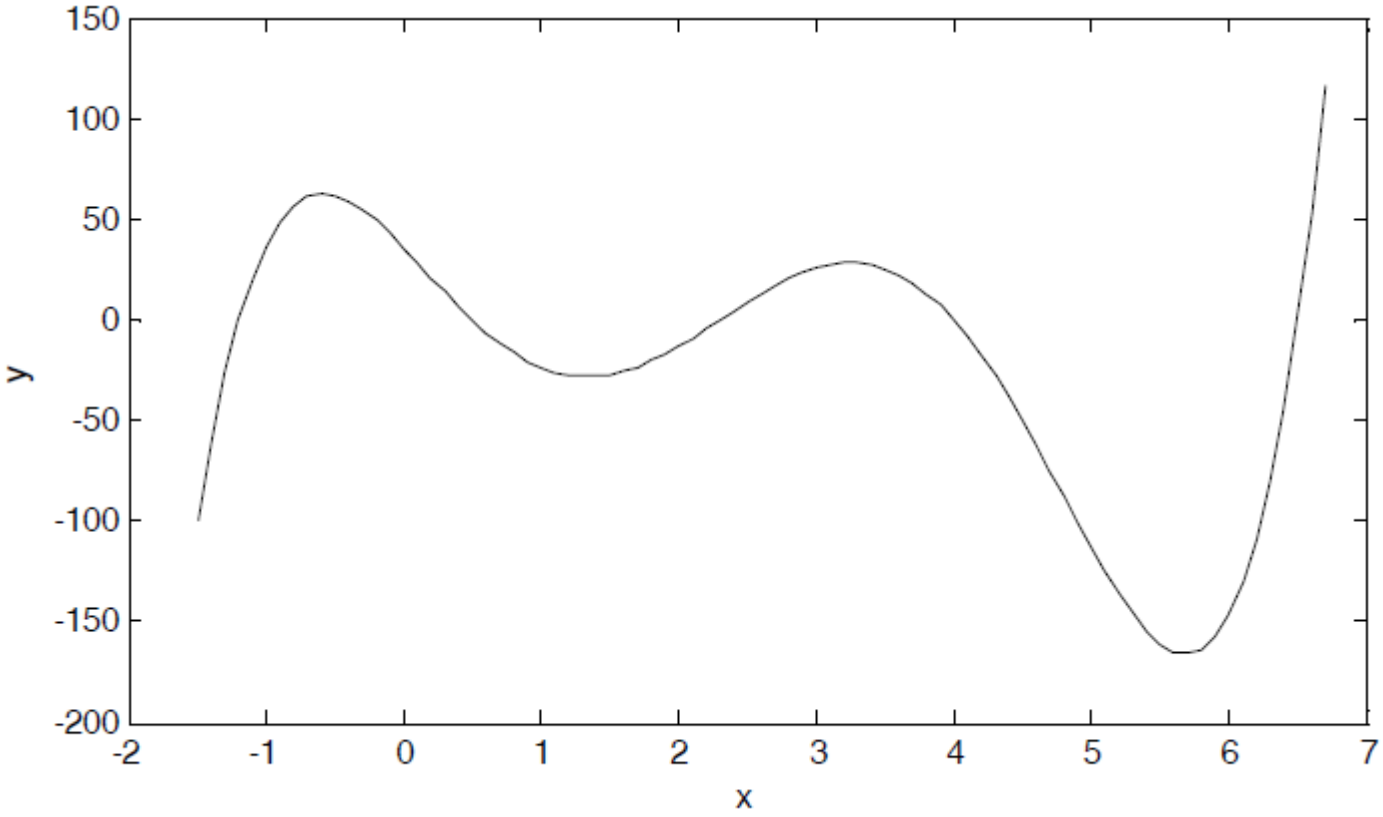
Solution

(*a*) The coefficients of the polynomials are assigned to vector p. The function ***polyval*** is then used to calculate the value at $x = 9$.

```
>> p = [1 -12.1 40.59 -17.015 -71.95 35.88];
>> polyval(p,9)
ans =
  7.2611e+003
```

(*b*) To plot the polynomial, a vector ***x*** is first defined with elements ranging from –1.5 to 6.7. Then a vector ***y*** is created with the values of the polynomial for every element of ***x.*** Finally, a plot of ***y*** vs***. x*** is made.

Sample Problem : Calculating polynomials with MATLAB



Solving polynomial equations

- Use the function **roots** for finding the roots of a polynomial equation.
- The argument of this function is the vector of the polynomial coefficients.

$$x^2 + 2x + 1$$

```
>> coef_poly_1 = [1  2  1]
>> roots_eq_1 = roots(coef_poly_1)
roots_eq_1 =
-1
-1
```

```
>> coef_poly_2 = [2  0  -15  -12]
>> roots_eq_2 = roots(coef_poly_2)
roots_eq_2 =
 3.0744
-2.1785
-0.8959
```

$$2x^3 - 15x - 12 = 0$$

```
>> coef_poly_3 = [2  0  -15  0]
>> roots_eq_3 = roots(coef_poly_3)
roots_eq_3 =
   0
 2.7386    -2.7386
```

$$2x^3 - 15x = 0$$

Roots of a polynomial can be complex numbers

$$x^5 - 2x^4 + 2x^3 + 3x^2 + x = -4$$

$$x^5 - 2x^4 + 2x^3 + 3x^2 + x + 4 = 0$$

>> coef_poly_4 = [1 -2 2 3 1 4]

coef_poly_4 =

1 -2 2 3 1 4

$$x^5 - 2x^4 + 2x^3 + 3x^2 + x + 4 = 0$$

```
>> coef_poly_4 = [1  -2   2   3   1   4]
>> roots_eq_4 = roots(coef_poly_4)
roots_eq_4 =
1.5336 + 1.4377i
1.5336 - 1.4377i
-1.0638
-0.0017 + 0.9225i
-0.0017 - 0.9225i
```

Function fzero

- In general, we would use the function **fzero for solving equations**.
- The command **fzero ('fun', X0)** tries to find a zero (a root) of the function **fun** near **X0**.

```
equation = '2*x.^4-55*x.^3 + 506.02*x.^2 -1755.89*x + 1670.79';
R1 = fzero (equation, -1)
R1 =
1.5000
```

>> R1 = fzero (equation, 3)

R1 =

1.5000

>> R1 = fzero (equation, 4)

R1 =

5.5000

>> R1 = fzero (equation, 7)

R1 =

5.5000

>> R1 = fzero (equation, 8)

R1 =

8.3000

>> R1 = fzero (equation, 12)

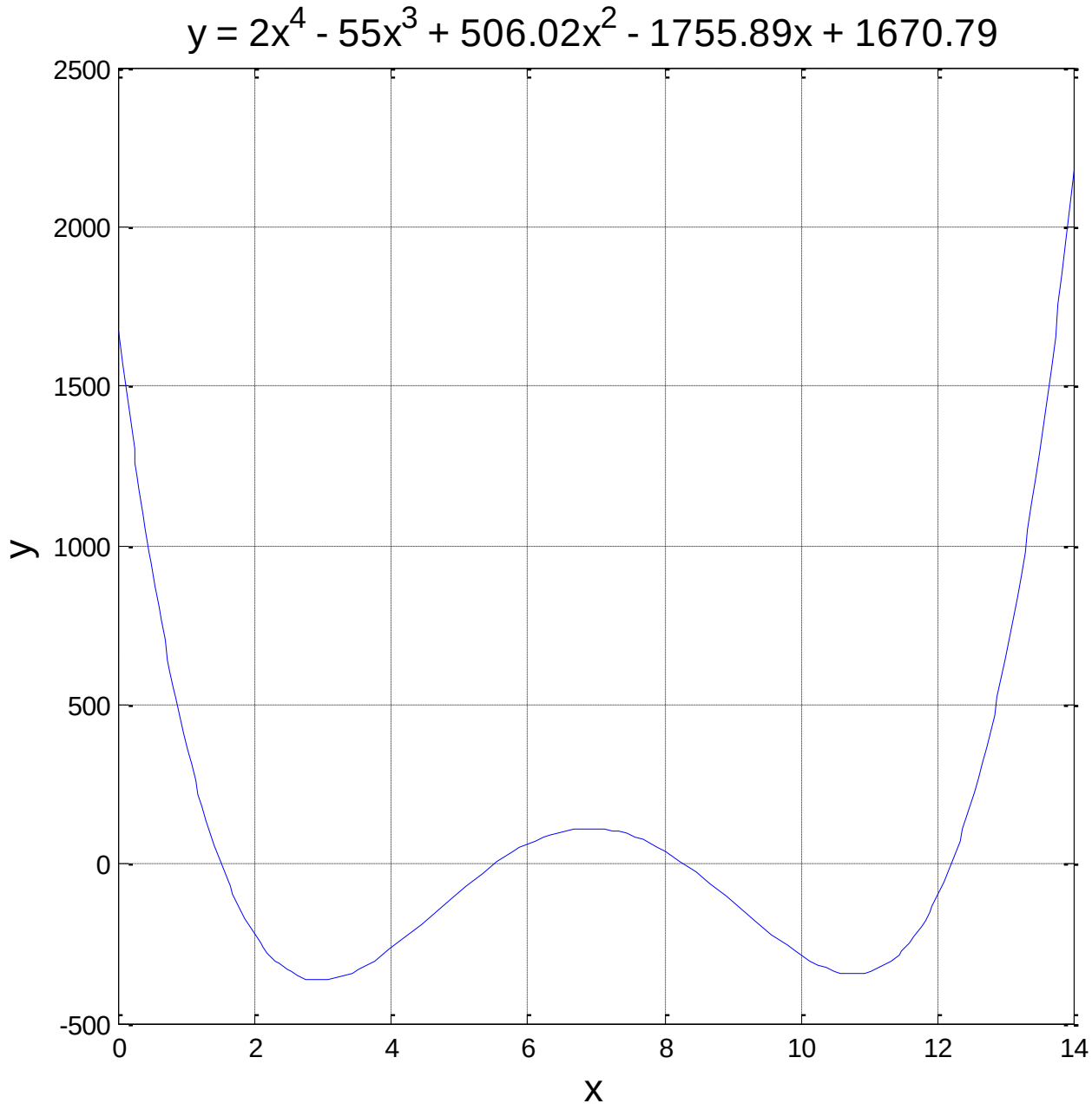
R1 =

12.2000

>> R1 = fzero (equation, 16)

R1 =

12.2000


y = 2x^4 - 55x^3 + 506.02x^2 - 1755.89x + 1670.79
2500
2000
1500
1000
500
0
-500
y
0
2
4
6
8
10
12
14
x

Defining a polynomial with known roots

- The function **poly** is the inverse of the function **roots**.
- For example, determine the coefficients of a polynomial whose roots are equal to

–1+i

1+i

2+3i

–4

```
>> equation_roots  =  [-1+i,  1+i,  2+3i,  -4];
>> equation_coefs  =  poly(equation_roots);
>> equation_coefs  =  equation_coefs(:)
equation_coefs =
 1.0000
 2.0000 - 5.0000i
-16.0000 - 16.0000i
-28.0000 + 22.0000i
 16.0000 + 24.0000i
```

$$x^4 + (2-5i)x^3 - (16+16i)x^2 + (-28+22i)x + (16+24i)$$

Addition of Polynomials

Two polynomials can be added (or subtracted) by adding (subtracting) the vectors of the coefficients.

If the polynomials are not of the same order (which means that the vectors of the coefficients are not of the same length),

the shorter vector has to be modified to be of the same length as the longer vector by adding zeros (padding) in front.

Addition of Polynomials

$$f_1(x) = 3x^6 + 15x^5 - 10x^3 - 3x^2 + 15x - 40 \text{ and } f_2(x) = 3x^3 - 2x - 6$$

```
>> p1=[3 15 0 -10 -3 15 -40];
>> p2=[3 0 -2 -6];
>> p=p1+[0 0 0 p2]
p =
     3    15     0    -7    -3    13   -46
```

Three 0s are added in front of `p2`, since the order of `p1` is 6 and the order of `p2` is 3.

Multiplication of Polynomials

Two polynomials can be multiplied using the MATLAB built-in function *conv*

```
c = conv(a,b)
```

`c` is a vector of the coefficients of the polynomial that is the product of the multiplication.

`a` and `b` are the vectors of the coefficients of the polynomials that are being multiplied.

Multiplication of Polynomials

Two polynomials can be multiplied using the MATLAB built-in function *conv*

$$f_1(x) = 3x^6 + 15x^5 - 10x^3 - 3x^2 + 15x - 40 \text{ and } f_2(x) = 3x^3 - 2x - 6$$

```
>> pm=conv(p1,p2)
pm =
     9    45    -6   -78   -99    65   -54   -12   -10   240
```

which means that the answer is:

$$9x^9 + 45x^8 - 6x^7 - 78x^6 - 99x^5 + 65x^4 - 54x^3 - 12x^2 - 10x + 240$$

Division of Polynomials

A polynomial can be divided by another polynomial with the MATLAB built-in function *deconv*

```
[q,r] = deconv(u,v)
```

`q` is a vector with the coefficients of the quotient polynomial.
`r` is a vector with the coefficients of the remainder polynomial.

`u` is a vector with the coefficients of the numerator polynomial.
`v` is a vector with the coefficients of the denominator polynomial.

Division of Polynomials

A polynomial can be divided by another polynomial with the MATLAB built-in function *deconv*

$2x^6 - 13x^5 + 75x^3 + 2x^2 - 60$ divided by $x^2 - 5$:

```
>> w=[2 -13 0 75 2 0 -60];
>> z=[1 0 -5];
>> [g h]=deconv(w,z)
g =
     2   -13    10    10    52
h =
     0     0     0     0     0    50   200
```

The quotient is: $2x^4 - 13x^3 + 10x^2 + 10x + 52$.

The remainder is: $50x + 200$.

Derivatives of Polynomials

The built-in function *polyder* can be used to calculate the derivative of a single polynomial, a product of two polynomials, or a quotient of two polynomials

`k = polyder(p)`	Derivative of a single polynomial. `p` is a vector with the coefficients of the polynomial. `k` is a vector with the coefficients of the polynomial that is the derivative.
`k = polyder(a,b)`	Derivative of a product of two polynomials. `a` and `b` are vectors with the coefficients of the polynomials that are multiplied. `k` is a vector with the coefficients of the polynomial that is the derivative of the product.
`[n d] = polyder(u,v)`	Derivative of a quotient of two polynomials. `u` and `v` are vectors with the coefficients of the numerator and denominator polynomials. `n` and `d` are vectors with the coefficients of the numerator and denominator polynomials in the quotient that is the derivative.

Derivatives of Polynomials

the derivatives of

$3x^2 - 2x + 4$, $(3x^2 - 2x + 4)(x^2 + 5)$, and $\dfrac{3x^2 - 2x + 4}{x^2 + 5}$

```
>> f1= 3 -2 4];
>> f2=[1 0 5];
```

Creating the vectors of coefficients of f_1 and f_2.

```
>> k=polyder(f1)
k =
     6    -2
```

The derivative of f_1 is: $6x - 2$.

```
>> d=polyder(f1,f2)
d =
    12    -6    38   -10
```

The derivative of $f_1 * f_2$ is: $12x^3 - 6x^2 + 38x - 10$.

```
>> [n d]=polyder(f1,f2)
n =
     2    22   -10
d =
     1     0    10     0    25
```

The derivative of $\dfrac{3x^2 - 2x + 4}{x^2 + 5}$ is: $\dfrac{2x^2 + 22x - 10}{x^4 + 10x^2 + 25}$.

Non-polynomial function

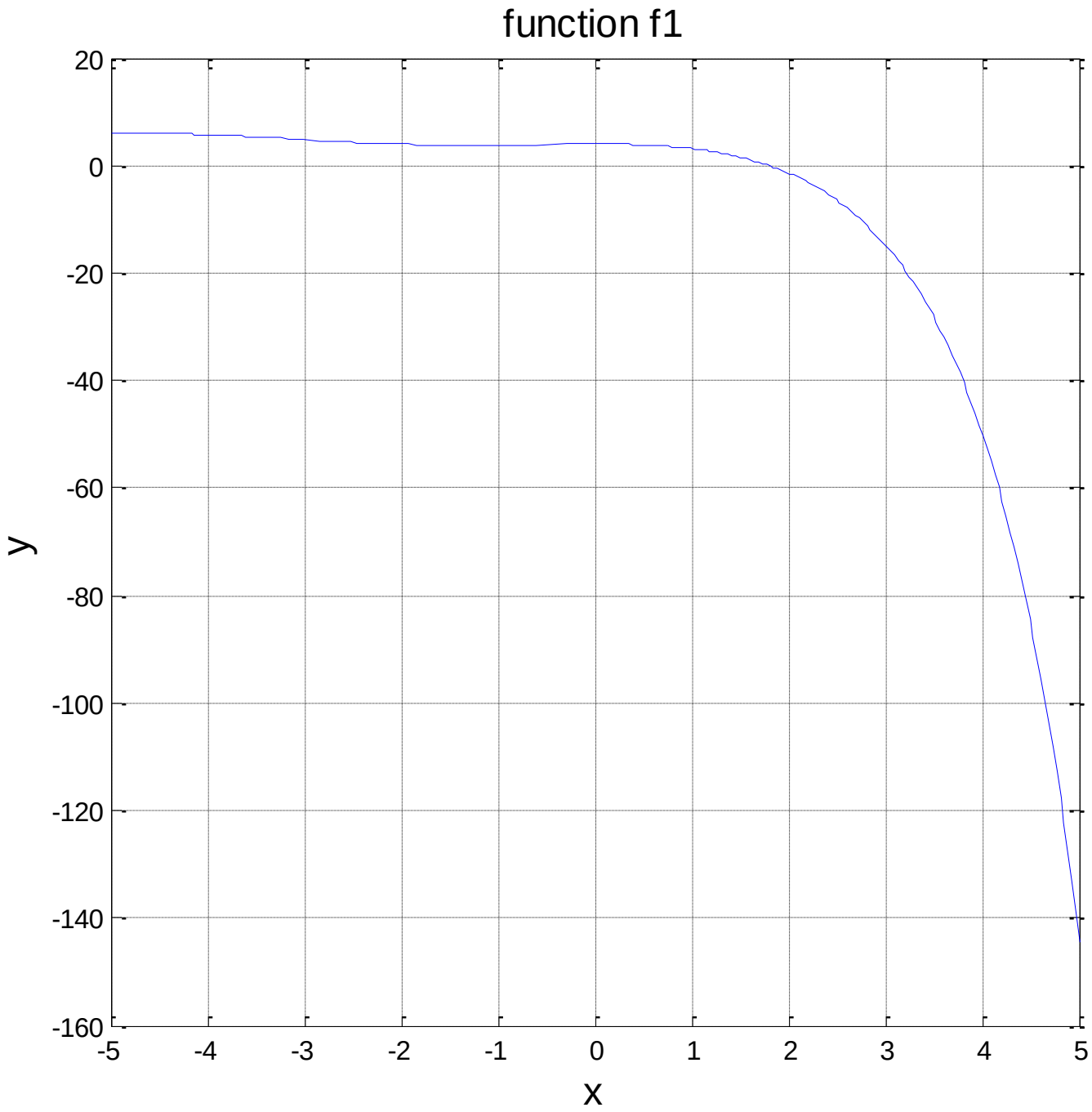
$$\sin(x) = e^{x} - 5$$

$$\sin(x) - e^{x} + 5 = 0$$

root1 = fzero ('sin(x)-exp(x)+5', 16)

root1 =

1.7878


function f1
20
0
-20
-40
-60
-80
-100
-120
-140
-160
y
-5
-4
-3
-2
-1
0
1
2
3
4
5
x

Non-polynomial function: Define equation by m-file

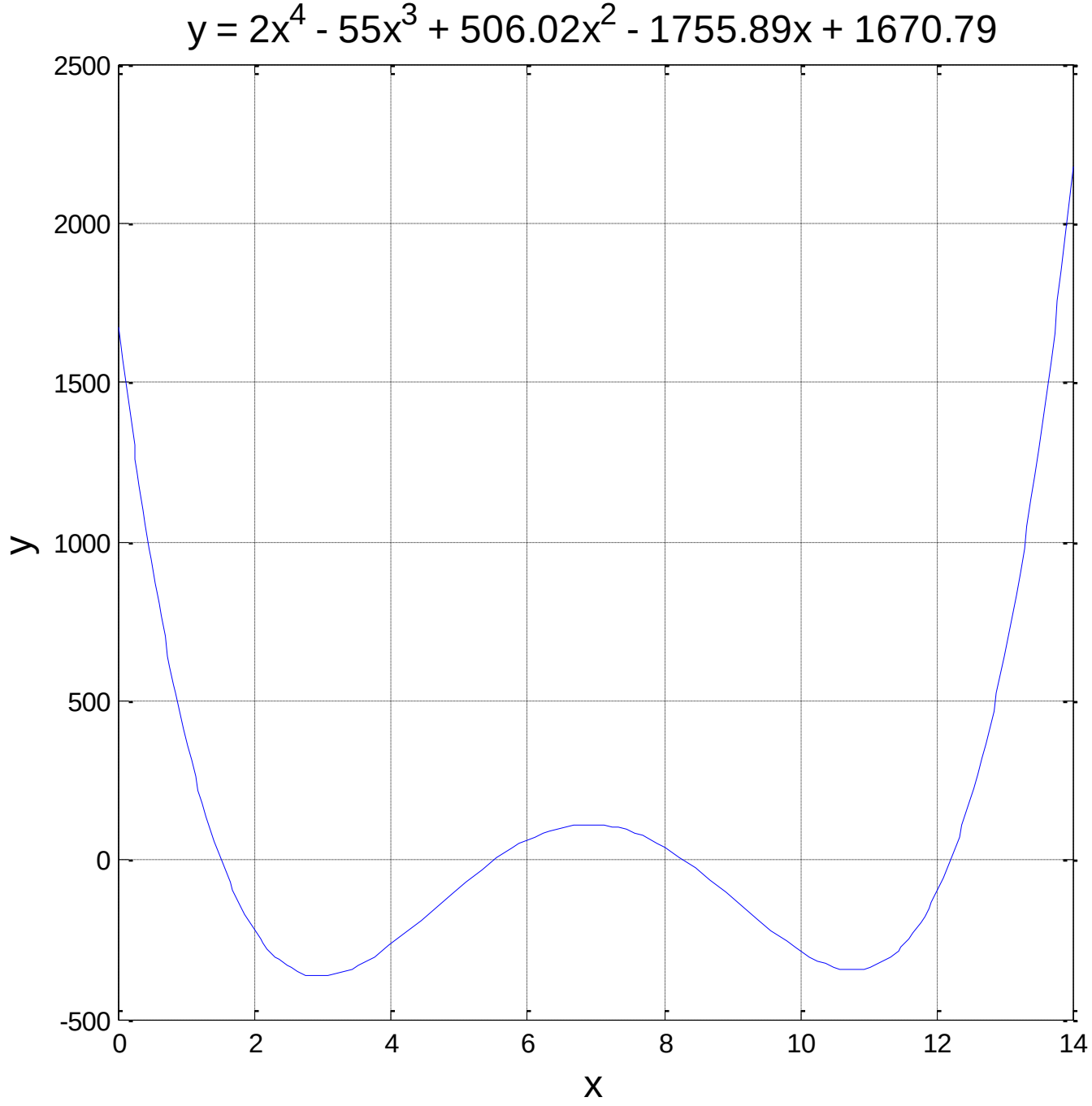
$$\sin(x) = e^{x} - 5$$

$$\sin(x) - e^{x} + 5 = 0$$

```
%function M-file to define f1
function y = f1(x)
y = sin(x) - exp(x) + 5;
```

```
>> root1 = fzero('f1',16)
root1 =
1.7878
```

FINDING THE MINIMUM OF A FUNCTION


y = 2x^4 - 55x^3 + 506.02x^2 - 1755.89x + 1670.79
2500
2000
1500
1000
500
0
-500
y
0
2
4
6
8
10
12
14
x

The command fminsearch

- **Fminsearch ('fun', X0)** tries to find a local minimum of the function fun near X0.

```
>> equation = '2*x.^4-55*x.^3+506.02*x.^2-
   1755.89*x+1670.79';
>> x_min_1 = fminsearch (equation,12)
x_min_1 =
10.7745
>> x_min_2 = fminsearch (equation, 5)
x_min_2 =
2.9537
```

Minimum of a function defined by M-file

```
function y = f_poly_1(x)
y = 2*x.^4 - 55*x.^3 + 506.02*x.^2 -1755.89*x + 1670.79;
>> x_min_1 = fminsearch('f_poly_1',12)
x_min_1 =
   10.7745
>> x_min_2 = fminsearch('f_poly_1',5)
x_min_2 =
    2.9537
```

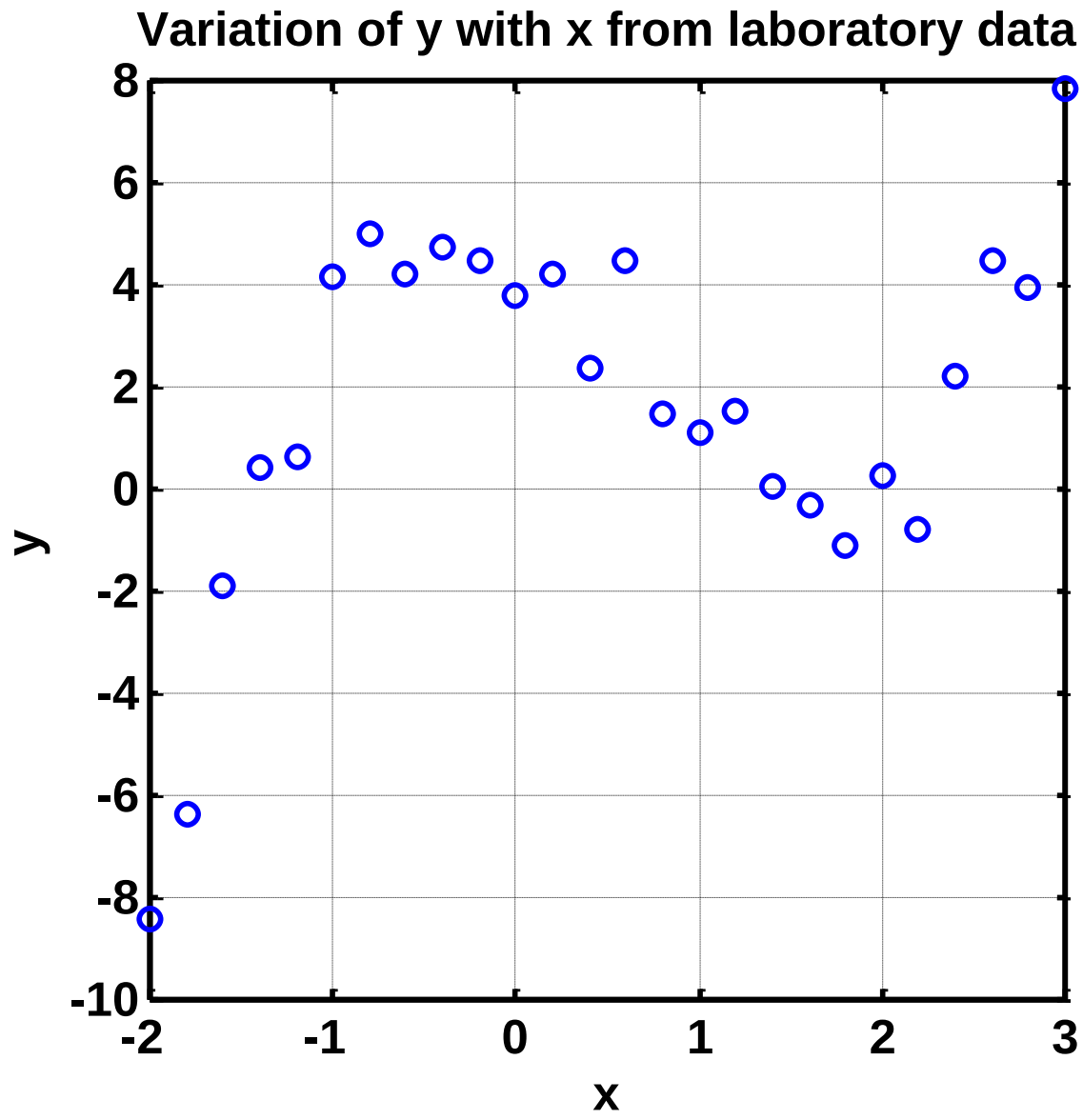
The value of the function at its minimum

```
>> y_min_1 = f_poly_1 (x_min_1)
y_min_1 =
 -345.1092

>> y_min_2 = f_poly_1(x_min_2)
y_min_2 =
 -365.9617
```

Curve Fitting

Fitting a polynomial to data

x	y	x	y
-2.00	-8.43	0.60	4.48
-1.80	-6.38	0.80	1.50
-1.60	-1.89	1.00	1.11
-1.40	0.42	1.20	1.51
-1.20	0.65	1.40	0.08
-1.00	4.19	1.60	-0.32
-0.80	5.00	1.80	-1.08



Variation of y with x from experimental data

Fitting a polynomial of order n to the data points (xi, yi)

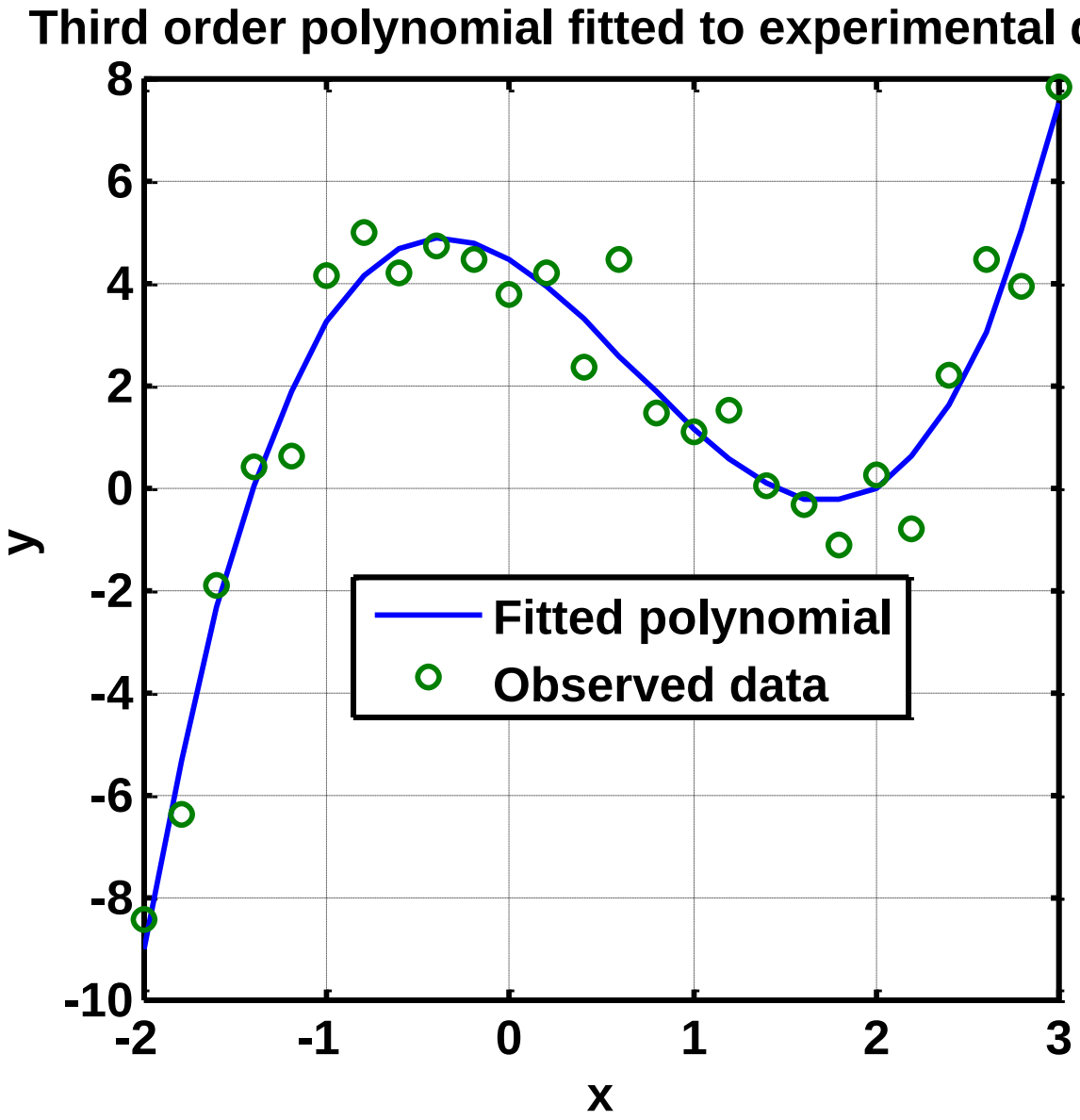
>> coef_poly = polyfit(x,y,n)

n is the order of the polynomial.

- **coef_poly** is the vector of the coefficients of a polynomial of order **n** that best fits the data (x_i, y_i) in a least square sense.

- We can then use the vector **coef_poly** to evaluate the polynomial for the given **x** values

```
>> x = [-2, -1.8,          ];
>> y = [-8.43, -6.38       ];

>> coef_poly = polyfit(x,y,3);
>> y_hat = polyval(coef_poly, x);

>> figure, plot(x, y_hat, '-', x, y, 'o')
```

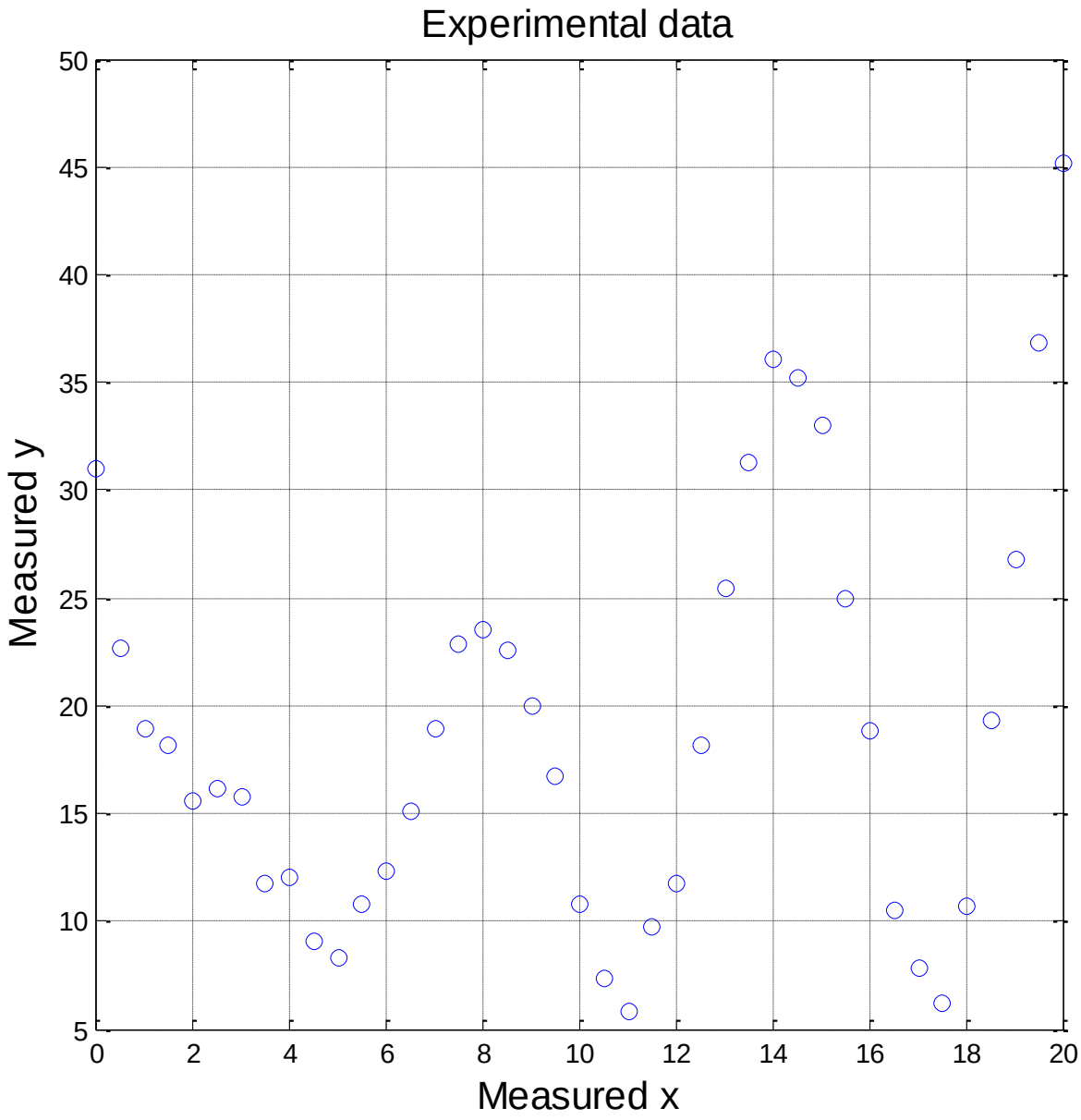


Third order polynomial fitted to experimental data

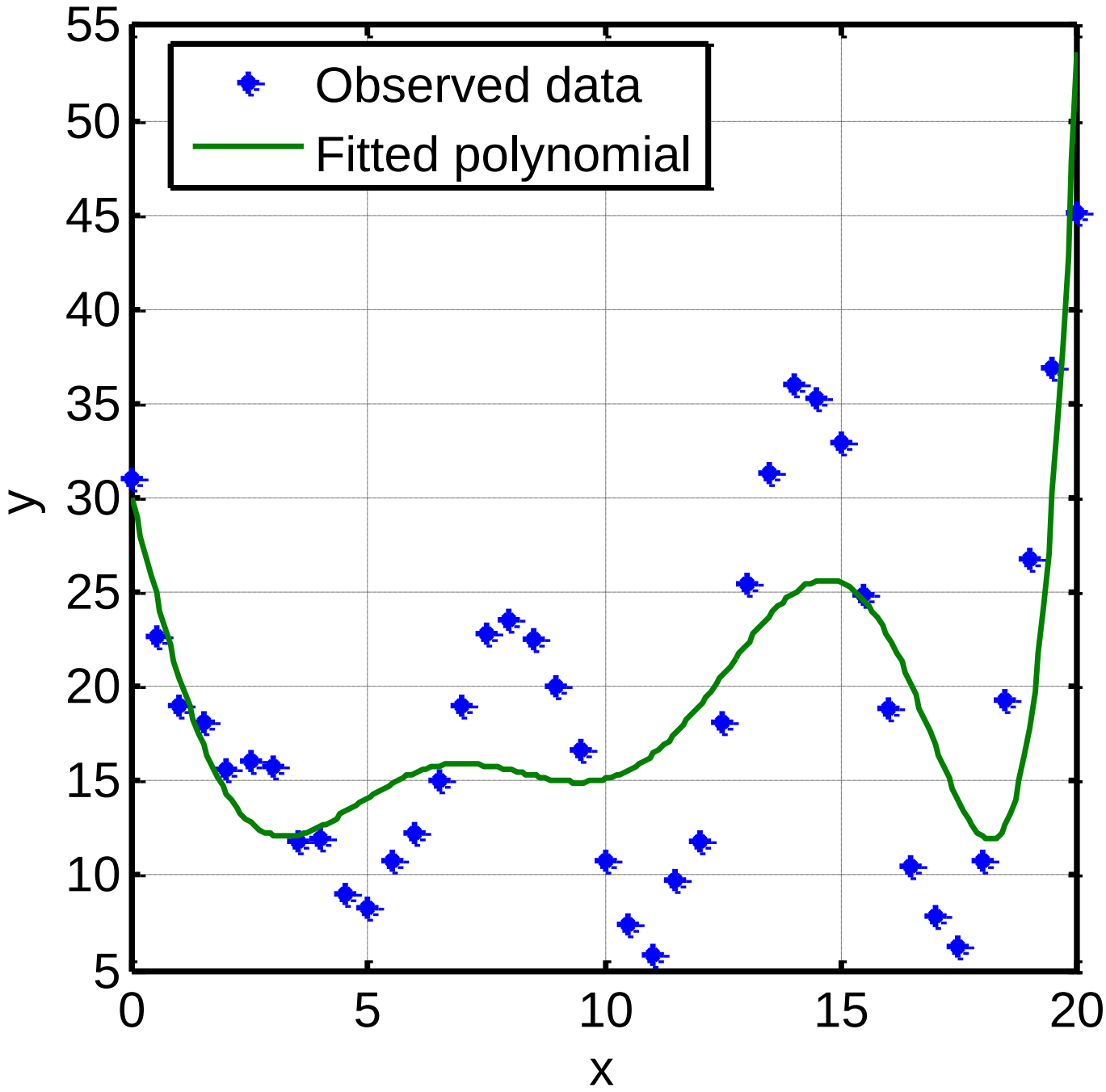
CURVE FITTING by LSE

- Technique used for fitting an algebraic function to data.
- Usually done by the Least Square Error (LSE) method.

x	y	x	y	x	y
0.00	30.99	7.00	18.95	14.00	36.03
0.50	22.67	7.50	22.83	14.50	35.25
1.00	18.96	8.00	23.50	15.00	32.99
1.50	18.17	8.50	22.56	15.50	24.94
2.00	15.62	9.00	20.02	16.00	18.85
2.50	16.14	9.50	16.73	16.50	10.46
3.00	15.78	10.00	10.84	17.00	7.79
3.50	11.79	10.50	7.39	17.50	6.22
4.00	12.01	11.00	5.78	18.00	10.74
4.50	9.08	11.50	9.73	18.50	19.31
5.00	8.28	12.00	11.78	19.00	26.79
5.50	10.76	12.50	18.12	19.50	36.83
6.00	12.28	13.00	25.41	20.00	45.17
6.50	15.08	13.50	31.28		



Graphical representation of the experimental data


7th order polynomial fitted to experimental data
Observed data
Fitted polynomial
y
x
55
50
45
40
35
30
25
20
15
10
5
0
5
10
15
20

Theoretical or empirical relationship

$$y = \frac{a}{x+1} + b\,x \; + c\,x\sin(x) + d$$

$$y_i = \frac{a}{x_i+1} + b\,x_i \; + c\,x_i\sin(x_i) + d$$

$$i = 1, 2, \quad \ldots\ldots \quad , 41$$

Derivation of parameters

- Only **4** unknown parameters (**a**, **b**, **c** and **d**), but **41** equations.
- Equations are said to be overdetermined.
- In other words, no set of **a**, **b**, **c** and **d** values would satisfy all the **41** equations.
- Consequently, for any choice of the parameters, the **41** equations can be written in the following form

$$y_i - \left(\frac{a}{x_i + 1} + b\,x_i \ + c\,x_i \sin(x_i) + d \right) = e_i$$

$$i = 1, 2, \quad \ldots\ldots \quad , 41$$

LSE Method

- **e_i** is the experimental error and/or inaccuracies in the theoretical model.
- Determine the parameters **a**, **b**, **c** and **d** to minimise the error terms **e_i** in some sense.
- not good to minimise the sum of the errors

  as the positive and negative errors can cancel each other out.
- Minimise the sum of the squares of the errors,

  because the squares are always positive and they do not cancel each other out.

$$\frac{a}{x_i + 1} + b\,x_i + c\,x_i \sin(x_i) + d = y_i$$

$$i = 1, 2, \quad \ldots\ldots \quad , 41$$

$$\begin{bmatrix} \frac{1}{x_1+1} & x_1 & x_1\sin(x_1) & 1 \\ \frac{1}{x_2+1} & x_2 & x_2\sin(x_2) & 1 \\ . & . & . & . \\ . & . & . & . \\ \frac{1}{x_i+1} & x_i & x_i\sin(x_i) & 1 \\ . & . & . & . \\ . & . & . & . \\ \frac{1}{x_{41}+1} & x_{41} & x_{41}\sin(x_{41}) & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_i \\ . \\ . \\ y_{41} \end{bmatrix}$$

A * parameters = y

- Define matrix **A** and the vertical vector **y.**
- Determine parameters **a**, **b**, **c** and **d** from **parameters = A\y**.
- Always use the backlash operator **** to solve a set of linear equations.
- If the number of **unknowns** and **equations** are equal to each other,
  - there would be a unique set of parameter values which would satisfy all the equations.

$A * parameters = y$

- If the number of equations are **more than** the number of parameters,
  - then the equations will be solved using the Least Square Error technique.

MATLAB Commands

```
x = 0:0.5:20;
y = [30.99  22.67  18.96  18.17  15.62  ...
12.01  9.08  8.28  10.76  12.28  15.08 ...
23.50  22.56  20.02  16.73  10.84  7.39 ...
18.12  25.41  31.28  36.03  35.25  24.94 ...];
x = x(:);
y = y(:);
N = length(x);
```

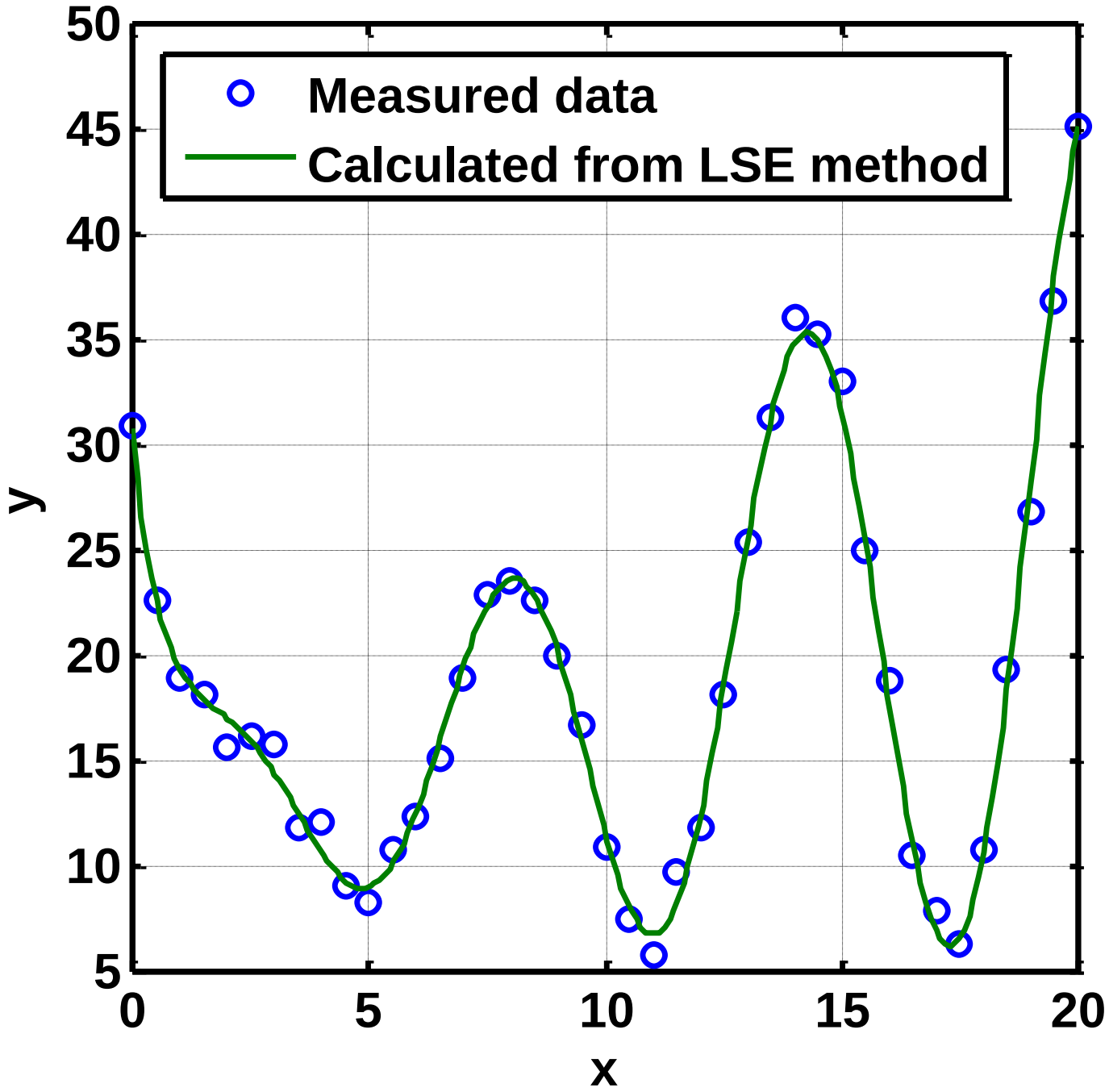
$A * parameters = y$

```
>> A = [1./(x+1)   x   x.*sin(x)   ones(N,1)];

>> C = A\y
```

Plot a smooth curve through the data points

```
x = linspace (0, 20, 200);
y_p = C(1)./(x+1)  +  C(2)*x + ... C(3)*x.*sin(x)  +  C(4);

figure, plot (x, y, 'o', xn, y_p, '-'),
grid on, axis square
xlabel('x','Fontsize',14),ylabel('y','Fontsize',14)
title('Fitting curve ', 'Fontsize',14)
legend('Measured data',' LSE')
```

Fitting a curve to data using Least Square Error method



Curve Fitting with Functions Other than Polynomials

Many situations in science and engineering require fitting functions that are not polynomials to given data. Theoretically, any function can be used to model data within some range. For a particular data set, however, some functions provide a better fit than others. This section covers curve fitting with ***power***, ***exponentia***l, ***logarithmic***, and ***reciprocal*** functions, which are commonly used. The forms of these functions are:

$$y = bx^m \quad \text{(power function)}$$

$$y = be^{mx} \quad \text{or} \quad y = b10^{mx} \quad \text{(exponential function)}$$

$$y = m\ln(x) + b \quad \text{or} \quad y = m\log(x) + b \quad \text{(logarithmic function)}$$

$$y = \frac{1}{mx + b} \quad \text{(reciprocal function)}$$

All of these functions can easily be fitted to given data with the ***polyfit*** function. This is done by rewriting the functions in a form that can be fitted with a linear polynomial (n = 1), which is

$$y = mx + b$$

Curve Fitting with Functions Other than Polynomials

The power, exponential and reciprocal equations can be rewritten as:

$$\ln(y) = m\ln(x) + \ln b \qquad \text{(power function)}$$

$$\ln(y) = mx + \ln(b) \quad \text{or} \quad \log(y) = mx + \log(b) \qquad \text{(exponential function)}$$

$$\frac{1}{y} = mx + b \qquad \text{(reciprocal function)}$$

The polyfit(x,y,1) function can be used to determine the best-fit constants *m* and *b* for best fit if, instead of x and y, the following arguments are used.

Function		`polyfit` **function form**
power	$y = bx^m$	`p=polyfit(log(x),log(y),1)`
exponential	$y = be^{mx}$ or	`p=polyfit(x,log(y),1)` or
	$y = b10^{mx}$	`p=polyfit(x,log10(y),1)`
logarithmic	$y = m\ln(x) + b$ or	`p=polyfit(log(x),y,1)` or
	$y = m\log(x) + b$	`p=polyfit(log10(x),y,1)`
reciprocal	$y = \frac{1}{mx + b}$	`p=polyfit(x,1./y,1)`

Curve Fitting with Functions Other than Polynomials

For given data it is possible to estimate, to some extent, which of the functions has the potential for providing a good fit. This is done by plotting the data using different combinations of linear and logarithmic axes. If the data points in one of the plots appear to fit a straight line, the corresponding function can provide a good fit according to the list below.

***x* axis**	***y* axis**	**Function**
linear	linear	linear $y = mx + b$
logarithmic	logarithmic	power $y = bx^m$
linear	logarithmic	exponential $y = be^{mx}$ or $y = b10^{mx}$
logarithmic	linear	logarithmic $y = m\ln(x) + b$ or $y = m\log(x) + b$

Other considerations in choosing a function:

- Exponential functions cannot pass through the origin.
- Exponential functions can fit only data with all positive y's or all negative y's.
- Logarithmic functions cannot model $x = 0$ or negative values of x.
- For the power function $y = 0$ when $x = 0$.
- The reciprocal equation cannot model $y = 0$.

Interpolation

INTERPOLATION

- Have a set of **x** and **y** values.
- Pass a **smooth curve through these points**
- or find the **value of y** for some **intermediate x** values.
- This operation is called **interpolation**.
- Not same as curve fitting, as in curve fitting, the fitted curve does not, in general, pass through the data points.

function interp1

The function **interp1** can be used for one-dimensional interpolation.

The general form of this function is

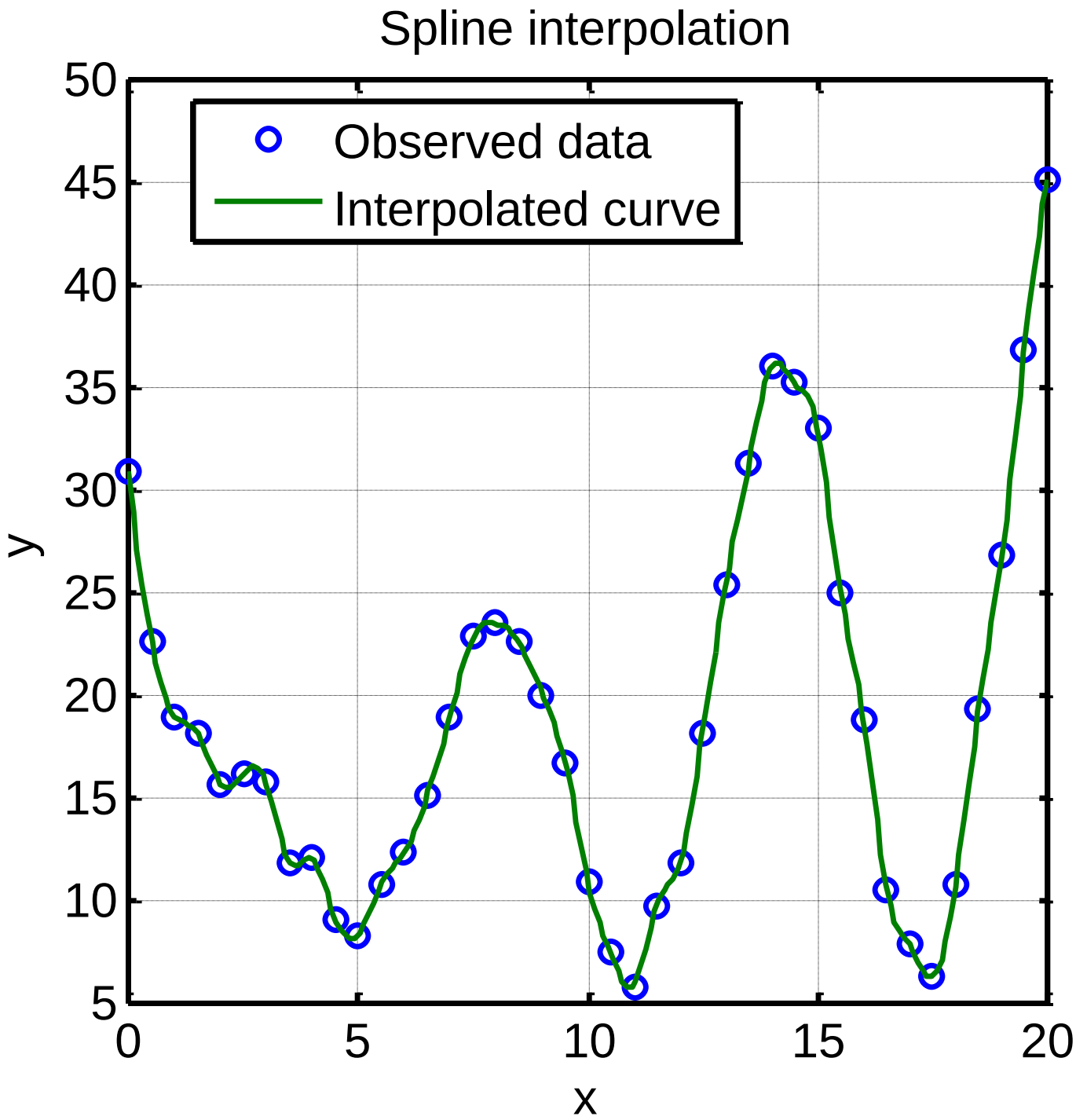
y_new = interp1 (x, y, x_new, 'method')

‘nearest’, ‘linear’, 'spline‘, ‘cubic’

'spline’: piecewise cubic spline interpolation

MATLAB commands

```
>> x_new = linspace(0,20,200);
>> y_new = interp1(x,y, x_new,'spline');

>> figure, plot (x, y, 'o', x_new, y_new, '-')
>> grid on, axis square
```


Spline interpolation
Observed data
Interpolated curve
y
x
50
45
40
35
30
25
20
15
10
5
0
5
10
15
20

THE BASIC FITTING INTERFACE

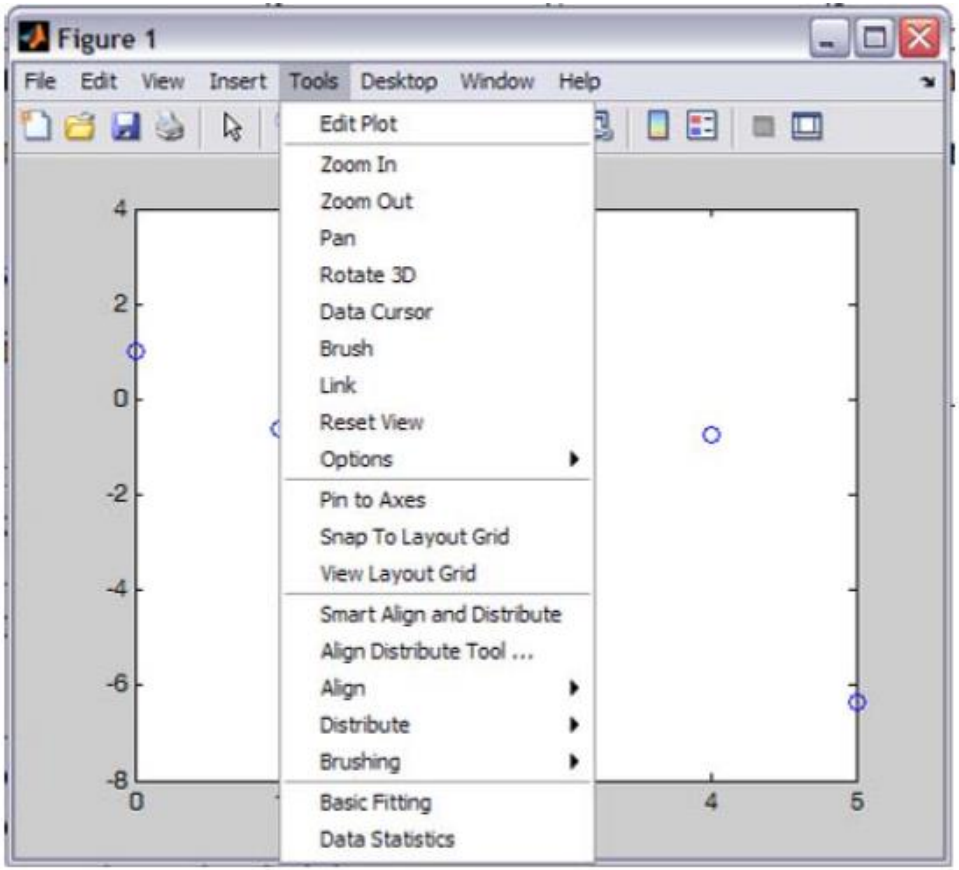
The basic fitting interface is a tool that can be used to perform curve fitting and interpolation interactively. By using the interface the user can:

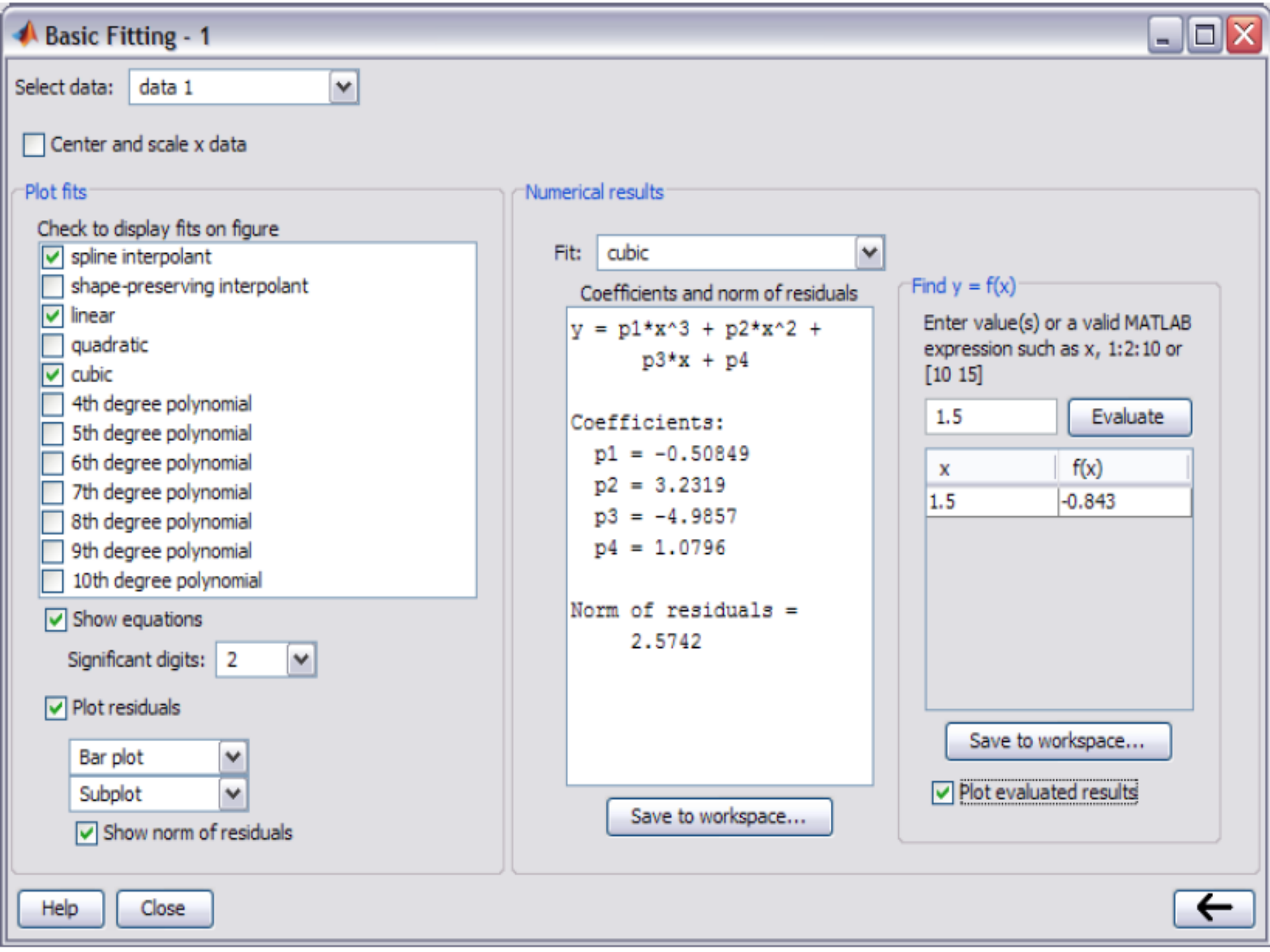
- Curve-fit the data points with polynomials of various degrees up to 10, and with spline and Hermite interpolation methods.
- Plot the various fits on the same graph so that they can be compared.
- Plot the residuals of the various polynomial fits and compare the norms of the residuals.
- Calculate the values of specific points with the various fits.
- Add the equations of the polynomials to the plot.

THE BASIC FITTING INTERFACE

To activate the basic fitting interface, the user first has to make a plot of the data points.

- select **Basic Fitting** in the **Tools** menu, as the following image shows




Basic Fitting - 1
Select data:
data 1
Center and scale x data
Plot fits
Check to display fits on figure
spline interpolant
shape-preserving interpolant
linear
quadratic
cubic
4th degree polynomial
5th degree polynomial
6th degree polynomial
7th degree polynomial
8th degree polynomial
9th degree polynomial
10th degree polynomial
Show equations
Significant digits:
2
Plot residuals
Bar plot
Subplot
Show norm of residuals
Numerical results
Fit:
cubic
Coefficients and norm of residuals
y = p1*x^3 + p2*x^2 +
p3*x + p4
Coefficients:
p1 = -0.50849
p2 = 3.2319
p3 = -4.9857
p4 = 1.0796
Norm of residuals =
2.5742
Save to workspace...
Find y = f(x)
Enter value(s) or a valid MATLAB expression such as x, 1:2:10 or [10 15]
1.5
Evaluate
x
f(x)
1.5
-0.843
Save to workspace...
Plot evaluated results
Help
Close

THE BASIC FITTING INTERFACE

Basic Fitting Window Panels:

- **Select data:** Used to select a specific set of data points for curve fitting in a figure. Only one set of data points can be curve-fitted at a time, but multiple fits can be performed simultaneously on the same set.
- **Center and scale x data:** When this box is checked, the data is centered at zero mean and scaled to unit standard deviation. This might be needed in order to improve the accuracy of numerical computation.

Plot fit Panels:

- **Check to display fits on figure:** The user selects the fits to be displayed in the figure. The selections include interpolation with spline interpolant (interpolation method) that uses the spline function, interpolation with Hermite interpolant.
- **Show equations:** When this box is checked, the equations of the polynomials that were selected for the fit are displayed in the figure. The equations are displayed with the number of significant digits selected in the adjacent sign menu.
- **Plot residuals:** When this box is checked, a plot that shows the residual at each data point is created .
- **Show norm of residuals:** When this box is checked, the norm of the residuals is displayed in the plot of the residuals. The norm of the residual is a measure of the quality of the fit. A smaller norm corresponds to a better fit.

THE BASIC FITTING INTERFACE

Numerical results Panels:

- **Fit:** The user selects the fit to be examined numerically. The fit is shown on the plot only if it is selected in the **Plot fit** panel.

- **Coefficients and norm of residuals:** Displays the numerical results for the polynomial fit that is selected in the **Fit** menu. It includes the coefficients of the polynomial and the norm of the residuals. The results can be saved by clicking on the **Save to workspace** button.

- **Find y = f(x):** Provides a means for obtaining interpolated (or extrapolated) numerical values for specified values of the independent variable. Enter the value of the independent variable in the box, and click on the **Evaluate** button. When the **Plot evaluated results** box is checked, the point is displayed on the plot.

function interp2

Two-dimensional interpolation.

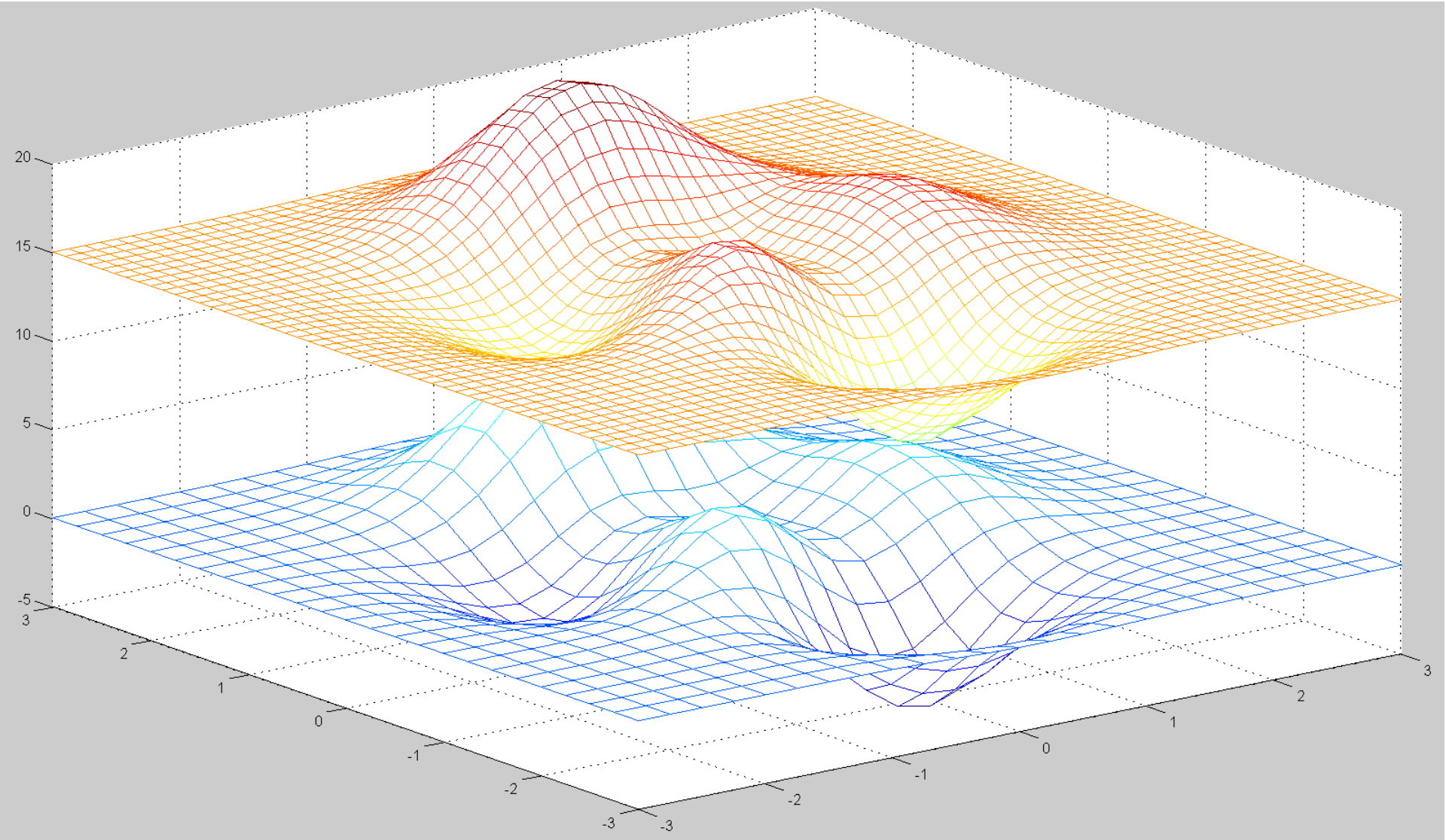
The general form of this function is

y_new = interp2 (x, y, f(x,y), x_new, y_new 'method')

‘nearest’, ‘linear’, 'spline‘, ‘cubic’

MATLAB commands

```
[X,Y] = meshgrid(-3:0.25:3);

Z = peaks(X,Y);

[XI,YI] = meshgrid(-3:0.125:3);

ZI = interp2(X,Y,Z,XI,YI);

figure, mesh(X,Y,Z), hold on, mesh(XI,YI,ZI+15)

hold off, axis([-3 3 -3 3 -5 20])
```


20
15
10
5
0
-5
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

Character Calculation

Coding and decoding message

Numerical Arrays

>> **RST** = rand(1,5)

RST =

0.8002 0.6478 0.4071 0.8756 0.3950

MATLAB will store these 5 numbers in five consecutive cells of computer memory and labels them as

RST(1), RST(2), RST(3), RST(4), RST(5).

RST, without any index, refers to all of these five numbers at the same time.

Character strings

- Variables can also be used to store a sequence of characters (i.e. strings such as 'Hello').
- The single quote mark is used to define strings in MATLAB.
- **>>** string1 = 'RST'
- **string1** =
- RST

- MATLAB locates three consecutive cells in the computer memory and labels them **string1**.

Character strings

- Now MATLAB stores character **R** in the first memory cell, i.e. the memory cell labelled **string1(1)**.
- Character **S** would be stored in the memory cell, labelled **string1(2)**.
- character **T** would be stored in the memory cell labelled **string1(3)**.
- **string1**, without any index, refers to all these three characters simultaneously.

Referring to a section of a character string

```
>> self_praise ='Dr Kirin was the winner of the best
   lecturer competition in year 2013'

self_praise =

Dr Kirin was the winner of the best lecturer competition
   in year 2013

>> self_praise(1)

ans =

   D
```

Referring to a section of a character string

```
>> self_praise ='Dr Kirin was the winner of the best
   lecturer competition in year 2013'
>> xyz = self_praise(10:15)
   xyz =
   an was

>> N_string = length(self_praise)
N_string =
   72
```

Combining character strings

```
>> x = 'MATLAB is '
   x =
   MATLAB is
>> y = 'a very powerful program'
   y =
   a very powerful program
>> x_and_y = [x, y]
   x_and_y =
   MATLAB is a very powerful program
```

Two-dimensional character arrays

```
>> Line_1 = 'I am a professional spy'
   Line_1 =
   I am a professional spy
>> Line_2 = 'MATLAB is essential to my work'
   Line_2 =
   MATLAB is essential to my work
>> Message = char(Line_1, Line_2)
   Message =
   I am a professional spy
   MATLAB is essential to my work
```

```
>> [N_rows, N_columns] = size(Message)
   N_rows =    2
   N_columns =        30
>> x = Message(1,:)
   x =
   I am a professional spy
>> y = Message(:,10:19)
   y =
   ofessional
      essential
```

Reversing the order of the columns of a matrix

```
Message =
I am a professional spy
MATLAB is essential to my work

>> zz = Message(:,end:-1:1)
zz =
      yps lanoisseforp a ma I
krow ym ot laitnesse si BALTAM
```

Functions int2str and num2str

The function int2str converts an integer number into a string

```
>> I=8324
I = 8324

>> A=int2str(I)
A =8324

>> size(I)
ans =     1

>> size(A)
ans =   1     4

>> x=A([1 3])
x =
82

>> x=A(2)
x =  3
```

Functions int2str and num2str

A = int2str(1234) is equivalent to A = '1234'

Numbers will be rounded if they are not integer

```
>> x=int2str(1234.78)
x = 1235
```

num2str converts a number into a string

```
>> X = num2str(pi,7)
X = 3.141593
```

```
>> X = num2str(0.1234567,3)
X = 0.123
```

```
>> X = num2str(1.1234567,3)
X = 1.12
```

Associated integer numbers

```
>> associated_number = double('X')
associated_number =
88
>> associated_number = double('x')
associated_number =
120
>> associated_number = double('(')
associated_number =
40
```

Associated integer numbers

```
>> associated_number = double(')')
associated_number =
41
>> associated_number = double(']')
associated_number =
93
>> associated_number = double(' ')
associated_number =
    32
```

Converting text to a matrix of integer numbers

```
>> Line_1 = 'I am a spy'
Line_1 =
I am a spy
>> Line_2 = 'So is 007'
Line_2 =
So is 007
>> original_message = char(Line_1,Line_2)
original_message =
I am a spy
So is 007
```

Converting text to a matrix of integer numbers and vice versa

```
>> associated_array =
double(original_message)
associated_array =
73   32   97  109   32   97   32  115  112  121
83  111   32  105  115   32   48   48   55   32

 >> decoded_message =
char(associated_array)
decoded_message =
I am a spy
So is 007
```

How to make life more difficult for CIA

```
>> Converted_array = associated_array.^(pi) + 2300;

>> Converted_array = Converted_array'
Converted_array =
1.0e+006 *
    0.7165    1.0713
    0.0558    2.6665
    1.7466    0.0558
    2.5186    2.2398
```

Decode the message

```
>> Converted_array = Converted_array';

>> New_array = (Converted_array-2300).^(1/pi);

>> associated_array = round(New_array)
associated_array =
73   32   97  109   32   97   32   115   112   121
83  111   32  105  115   32   48    48    55    32
>> decoded_message = char(associated_array)
decoded_message =
I am a spy
So is 007
```

Using random numbers

```
>> mean_x  = 2300;
>> sigma_x = 236;

>> random_numbers = sigma_x *
randn(size(associated_array)) + mean_x;

>> Converted_array = associated_array.^(pi) +
random_numbers;
```

Using random numbers

```
>> Converted_array = Converted_array'
Converted_array =
  1.0e+006 *
    0.7166    1.0713
    0.0559    2.6664
    1.7464    0.0559
    2.5188    2.2397
    0.0558    2.9802
    1.7466    0.0555
    0.0558    0.1934
    2.9799    0.1933
    2.7429    0.2955
```

Decode the message

```
>> Converted_array = Converted_array';
>> New_array = (Converted_array-
random_numbers).^(1/pi);

>> associated_array = round(New_array)
associated_array =
73   32   97  109   32   97   32   115   112   121
83  111   32  105  115   32   48    48    55    32
>> decoded_message = char(associated_array)
decoded_message =
I am a spy
So is 007
```

Comment regarding the use of random numbers

>> rand('state',J)

J is an integer number
Use the function rand as before, to generate random numbers.

The generated random numbers will be the same for any two people who use the same J number.
.

31th March

Computer based exercise (Room 210, Building 4 EE Department)