MATLAB

تعليـــم الماتلاب خطوة بخطـــــــوة

اعداد وتقديم

المهندس:- احمد محمد الفلاح الرابطى.

التخصص:- الهندسة الكهربية.

القسم:- التحكم الألى.

الجامعة:- جامعة الجبل الغربى.

المهنة:- معيد فى كلية الهندسة.

البريد الألكترونى:- ahmad_engineer21@yahoo.com

المستوى التعليمى:- بكالوريوس فى الهندسة الكهربية شعبة التحكم الألـى من جامعة الجبل الغربى ودوبلوما فى الدراسات العليا فى الهندسـة الكهربيـة شـعبة الـتحكم الألـى من جامعة الفاتح.

السنة الدراسية:- 2009 – 2010 ف.

الهواية:- المطالعة والشطرنج وكرة القدم.

العنوان:- ليبيا – غريان – الرابطة.

Numerical Integration

1- Trapezoidal Rule

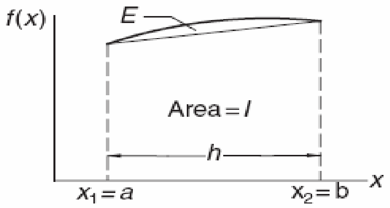


Figure . Trapezoidal rule.

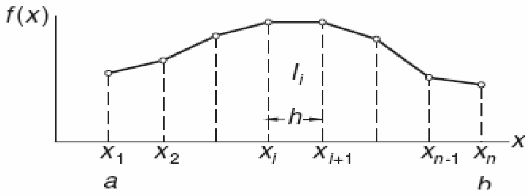


Figure . Composite trapezoidal rule.

The composite trapezoidal rule.

$$I = \sum_{i=1}^{n-1} I_i = [f(x_1) + 2f(x_2) + 2f(x_3) + \cdots + 2f(x_{n-1}) + f(x_n)]\frac{h}{2}$$

Example

Suppose we wished to integrate the function trabulated the table below for $f(x) = e^x$ over the interval from x=1.8 to x=3.4 using n=8

$$Am = \int_a^b f(x)dx = \int_{1.8}^{3.4} (e^x)dx$$

x	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6	3.8
f(x)	4.953	6.050	7.389	9.025	11.023	13.464	16.445	20.086	24.533	29.964	36.598	44.701

Solution

```
%---Trapezoidal Rule---------------
clc
a=1.8;
b=3.4;
h=0.2;
n=(b-a)/h
f=0;
x=2;
for i=1:n;
    %c=a+(i-1/2)*h;
    %f=f+(c^2+1);
    f=(f+exp(x))
    x=x+h;
end
Am_approx=h/2*(exp(a)+2*f+exp(b))
syms t
Am_exact=int(exp(t),1.8,3.4)
error=Am_exact-Am_approx
E_t=(error/(Am_approx+error))*100
```

```
E_a=((Am_approx-Am_exact)/Am_approx)*100
%--------------------------------
```

2- Simpson's 1/3 rule

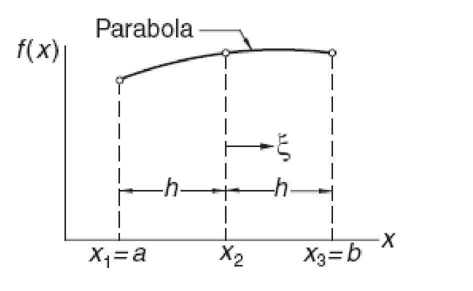


Figure . Simpson's 1/3 rule.

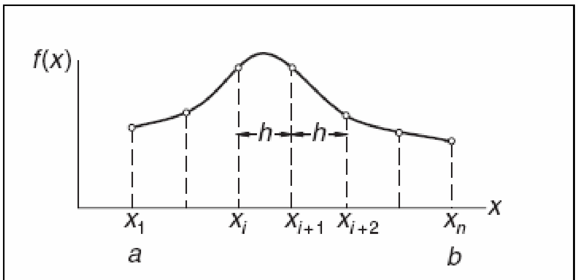


Figure. Composite Simpson's 1/3 rule.

The composite Simpson's 1/3 rule

$$\int_a^b f(x)\,dx \approx I = [f(x_1) + 4f(x_2) + 2f(x_3) + 4f(x_4) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]\frac{h}{3}$$

Example

Suppose we wished to integrate the function using Simpson's 1/3 rule and Simpson's 3/8 rule the table below for $f(x)=e^x$ over the interval from x=1.8 to x=3.4 using n=8

$$Am=\int_a^b f(x)dx=\int_{1.8}^{3.4}(e^x)dx$$

x	1.6	1.8	2	2.2	2.4	2.6	2.8	3	3.2	3.4	3.6	3.8
f(x)	4.953	6.050	7.389	9.025	11.023	13.464	16.445	20.086	24.533	29.964	36.598	44.701

Solution

```
%---Simpson's 1/3 rule --------------------------------
clc
   a=1.8;
   b=3.4;
   h=0.2;
   n=(b-a)/h
   f=0;
   m=0;
for x=2:(h+h):3.2;
      f=(f+exp(x));
end

for x=2.2:(h+h):3;
      m=(m+exp(x));
end
  Am_approx=h/3*(exp(a)+4*f+2*m+exp(b))
syms t
  Am_exact=int(exp(t),1.8,3.4)
  error=Am_exact-Am_approx
```

```
 E_t=(error/(Am_approx+error))*100
 E_a=((Am_approx-Am_exact)/Am_approx)*100
%-----------------------------------------------------
```

3-Simpson's 3/8 rule

The composite Simpson's 3/8 rule

$$\int_a^b f(x)\,dx \approx I = [f(x_1) + 3f(x_2) + 3f(x_3) + 2f(x_4) + \cdots + 3f(x_{n-2}) + 3f(x_{n-1}) + f(x_n)]\,\frac{3h}{8}$$

```
%--------------------Simpson's 3/8 rule --------------
clc
   a=1.8;
   b=3.4;
   h=0.2;
   n=(b-a)/h;
   f=0;
   m=0;
%---------------------------------------------------
    for x=2:h:2+h;
        f=f+exp(x)
    end
%---------------------------------------------------
    x=x+h;
    m=exp(x);
%---------------------------------------------------
     for x=2.6:h:2.6+h;
        f=f+exp(x);
     end
%---------------------------------------------------
    x=x+h;
    m=m+exp(x);
    x=x+h;
    f=f+exp(x);
%---------------------------------------------------
    Am_approx=((3*h)/8)*(exp(a)+3*f+2*m+exp(b))
%---------------------------------------------------
syms t
   Am_exact=int(exp(t),1.8,3.4)
   error=Am_exact-Am_approx
   E_t=(error/(Am_approx+error))*100
   E_a=((Am_approx-Am_exact)/Am_approx)*100
%---------------------------------------------------
```

```
%--------------------Simpson's 3/8 rule --------------
clc
   a=1.8;b=3.4;h=0.2;n=(b-a)/h;f=0;m=0;
%-----------------------------------------------------
   for x=2:h:3.2;
      switch x
        case {2,2.2}
             f=f+exp(x)
        case {2.4}
             m=exp(x);
        case {2.6,2.8}
            f=f+exp(x);
        case {3}
            m=m+exp(x);
        otherwise
           f=f+exp(x);
      end
   end
%-----------------------------------------------------
    Am_approx=((3*h)/8)*(exp(a)+3*(f)+2*(m)+exp(b))
%-----------------------------------------------------
syms t
   Am_exact=int(exp(t),1.8,3.4)
   pretty(Am_exact)
   error=Am_exact-Am_approx
   pretty(error)
   E_t=(error/(Am_approx+error))*100
   pretty(E_t)
   E_a=((Am_approx-Am_exact)/Am_approx)*100
   pretty(E_a)
%-----------------------------------------------------
```

4-Lagrange Interpolating Polynomial Method

Lagrange's interpolation method uses the formula

$$f(x) = \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)}f(x_0) + \frac{(x-x_0)(x-x_2)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)\dots(x_1-x_n)}f(x_1)$$

$$+ \frac{(x-x_0)(x-x_1)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_2)\dots(x_n-x_{n-1})}f(x_n)$$

EXAMPLE

Given the data points

x	0	2	3
y	7	11	28

use Lagrange's method to determine y at $x = 1$.

Solution

$$\ell_1 = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{(1-2)(1-3)}{(0-2)(0-3)} = \frac{1}{3}$$

$$\ell_2 = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} = \frac{(1-0)(1-3)}{(2-0)(2-3)} = 1$$

$$\ell_3 = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} = \frac{(1-0)(1-2)}{(3-0)(3-2)} = -\frac{1}{3}$$

$$y = y_1\ell_1 + y_2\ell_2 + y_3\ell_3 = \frac{7}{3} + 11 - \frac{28}{3} = 4$$

```
%-----------Lagrange's interpolation method ---------
clc
x=1;
%syms x
%-----------------------------------------------------
x1=0;
x2=2;
x3=3;
%-----------------------------------------------------
y0=7;
y1=11;
y2=28;
%-----------------------------------------------------
l0=((x-x2)*(x-x3))/((x1-x2)*(x1-x3))
l1=((x-x1)*(x-x3))/((x2-x1)*(x2-x3))
l2=((x-x1)*(x-x2))/((x3-x1)*(x3-x2))
%-----------------------------------------------------
y=y0*l0+y1*l1+y2*l2
%-----------------------------------------------------
```

Example 2

Construct the polynomial interpolating the data by using Lagrange polynomials

X	1	1/2	3
F(x)	3	-10	2

Solution

```
%-----------Lagrange's interpolation method ---------
clc
syms x
%----------------------------------------------------
x1=1;
x2=0.5;
x3=3;
%----------------------------------------------------
y0=3;
y1=-10;
y2=2;
%----------------------------------------------------
l0=((x-x2)*(x-x3))/((x1-x2)*(x1-x3))
l1=((x-x1)*(x-x3))/((x2-x1)*(x2-x3))
l2=((x-x1)*(x-x2))/((x3-x1)*(x3-x2))
%----------------------------------------------------
y=y0*l0+y1*l1+y2*l2;
collect(y)
%----------------------------------------------------

%-------------Lagrange's interpolation method--------

clc
syms x
p=0;
s=[1 1/2 3];
f=[3 -10 2];
n=length(s);
for i=1:n;
    l=1;
    for j=1:n;
        if (i~=j);
            l=((x-s(j))/(s(i)-s(j)))*l;
        end
      end
  p=l.*f(i)+p;
end
p=collect(p)

%----------------------------------------------------
```

Example 2

Construct the polynomial interpolating the data by using Lagrange polynomials

X	1	1/2	3
F(x)	3	-10	2

Solution

```
%------------Lagrange's interpolation method--------
clc
x=input(' enter value of x:')
p=0;
s=[1 1/2 3];
f=[3 -10 2];
n=length(s);
for i=1:n;
    l=1;
    for j=1:n;
        if (i~=j);
            l=((x-s(j))/(s(i)-s(j)))*l;
        end
      end
  p=l.*f(i)+p;
end
p;
fprintf('\n p(%3.3f)=%5.4f',x,p)
%-----------------------------------------------------
syms x
p=0;
for i=1:n;
    l=1;
    for j=1:n;
        if (i~=j);
            l=((x-s(j))/(s(i)-s(j)))*l;
        end
      end
  p=l.*f(i)+p;
end
p=collect(p)
%-----------------------------------------------------
```

p = -283/10 -53/5 *x^2 + 419/10 *x

enter value of x:5

x =5

p(5.000)=-83.8000

%---

Example 3

Find the area by lagrange polynomial using 3 nodes

X	1.8	2.6	3.4
F(x)	6.04964	13.464	29.964

Solution

```
%-----------Lagrange's interpolation method ---------
clc
syms x
%-----------------------------------------------------
x1=1.8;
x2=2.6;
x3=3.4;
%-----------------------------------------------------
F0=6.04964;
F1=13.464;
F2=29.964;
%-----------------------------------------------------
l0=((x-x2)*(x-x3))/((x1-x2)*(x1-x3))
A0=int(l0,1.8,3.4)
l1=((x-x1)*(x-x3))/((x2-x1)*(x2-x3))
A1=int(l1,1.8,3.4)
l2=((x-x1)*(x-x2))/((x3-x1)*(x3-x2))
A2=int(l2,1.8,3.4)
%-----------------------------------------------------
F=F0*A0+F1*A1+F2*A2
collect(F)
%-----------------------------------------------------

%-------------Lagrange's interpolation method---
clc
syms x
format long
p=0;
s=[1.8 2.6 3.4];
f=[6.04964 13.464 29.964];
n=length(s);
for i=1:n;
    l=1;
    for j=1:n;
        if (i~=j);
            l=((x-s(j))/(s(i)-s(j)))*l;
        end
    end
  A=int(l,s(1),s(n))
  p=A*f(i)+p;
end
 p
%-----------------------------------------------
```

5-Mid Point Rule

Example

Find the mid point approximation for

$$Am=\int_{a}^{b} f(x)dx=\int_{-1}^{2}(x^{\wedge}2+1)dx$$

using n=6

Solution

```
%---Mid Point Rule---------------
clc
a=-1;
b=2;
n=6;
h=(b-a)/n;
f=0;
for i=1:n;
    c=a+(i-1/2)*h;
    f=f+(c^2+1);
end
Am=h*f
%--------------------------------
```

6- Taylor series

A function $f(x)$ which possesses all derivatives up to order n at a point $x = x_0$ can be expanded in a *Taylor series* as

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

If $x_0 = 0$, reduces to

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \ldots + \frac{f^{(n)}(0)}{n!}x^n$$

Example

Compute the first three terms of the Taylor series expansion for the function

$$y = f(x) = \tan x$$

at $a = \pi/4$.

Solution:

The Taylor series expansion about point a is given by

$$f_n(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots$$

and since we are asked to compute the first three terms, we must find the first and second derivatives of $f(x) = \tan x$.

From math tables, $\frac{d}{dx}\tan x = \sec^2 x$, so $f'(x) = \sec^2 x$. To find $f''(x)$ we need to find the first derivative of $\sec^2 x$, so we let $z = \sec^2 x$. Then, using $\frac{d}{dx}\sec x = \sec x \cdot \tan x$, we get

$$\frac{dz}{dx} = 2\sec x \frac{d}{dx}\sec x = 2\sec x(\sec x \cdot \tan x) = 2\sec^2 x \cdot \tan x$$

Next, using the trigonometric identity

$$\sec^2 x = \tan^2 x + 1$$

and by substitution , we get,

$$\frac{dz}{dx} = f''(x) = 2(\tan^2 x + 1)\tan x$$

Now, at point $a = \pi/4$ we have:

$$f(a) = f\left(\frac{\pi}{4}\right) = \tan\left(\frac{\pi}{4}\right) = 1 \qquad f'(a) = f'\left(\frac{\pi}{4}\right) = 1 + 1 = 2 \qquad f''(a) = f''\left(\frac{\pi}{4}\right) = 2(1^2 + 1)1 = 4$$

and by substitution into (6.125),

$$f_n(x) = 1 + 2\left(x - \frac{\pi}{4}\right) + 2\left(x - \frac{\pi}{4}\right)^2 + \dots$$

We can also obtain a Taylor series expansion with the MATLAB **taylor(f,n,a)** function where **f** is a symbolic expression, **n** produces the first n terms in the series, and **a** defines the Taylor approximation about point a.

The following MATLAB script computes the first 8 terms of the Taylor series expansion of $y = f(x) = \tan x$ about $a = \pi/4$.

```
%-------------------- Taylor series --------------
clc
   a=pi/4;
   sym x
   y=tan(x);
   z=taylor(y,8,a);
   pretty(z)
%-----------------------------------------------------
```

Example

Express the function

$$y = f(t) = e^t$$

in a Maclaurin's series.

Solution:

A Maclaurin's series has the form, that is,

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \ldots + \frac{f^{(n)}(0)}{n!}x^n$$

For this function, we have $f(t) = e^t$ and thus $f(0) = 1$. Since all derivatives are e^t, then, $f'(0) = f''(0) = f'''(0) = \ldots = 1$ and therefore,

$$f_n(t) = 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \ldots$$

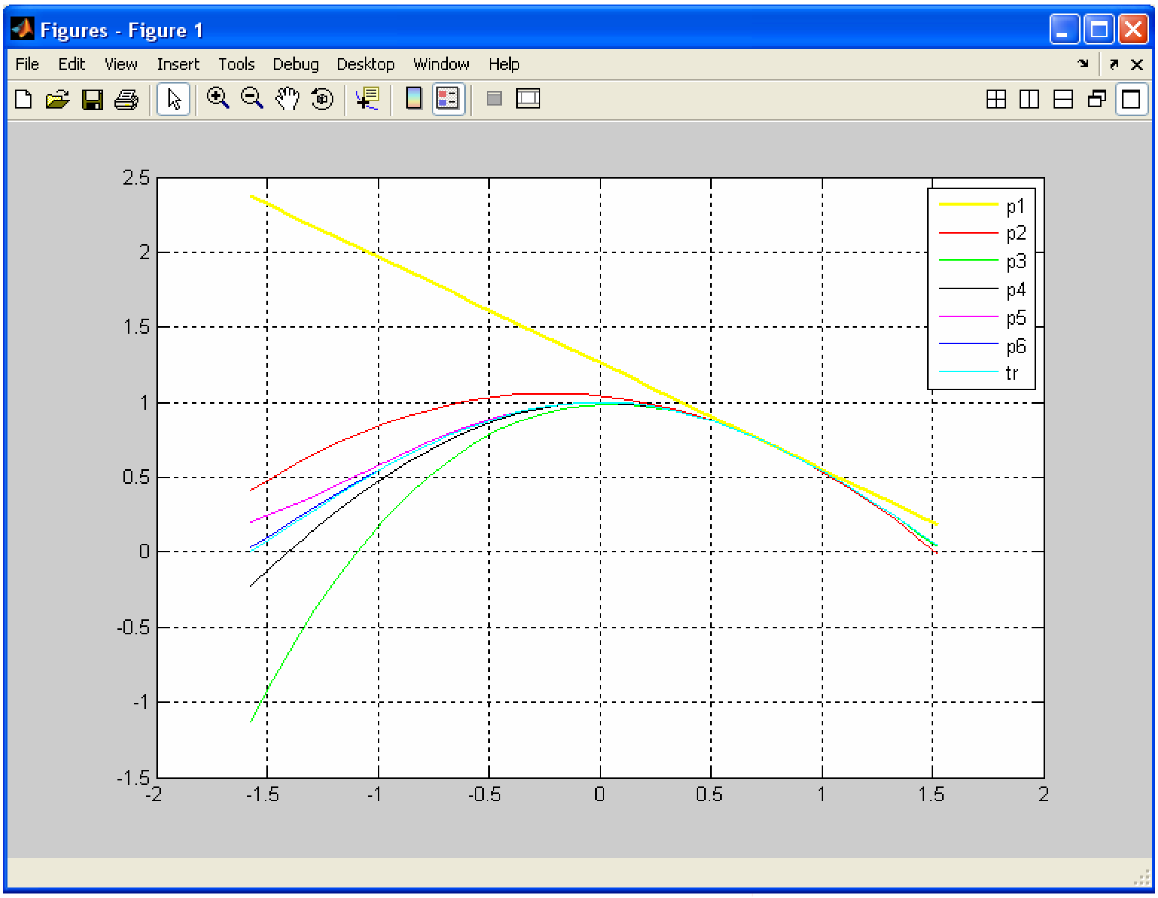
MATLAB displays the same result.

```
%-------------------- Taylor series --------------
clc
  syms t
  fn=taylor(exp(t));
  pretty(fn)
%-----------------------------------------------------
```

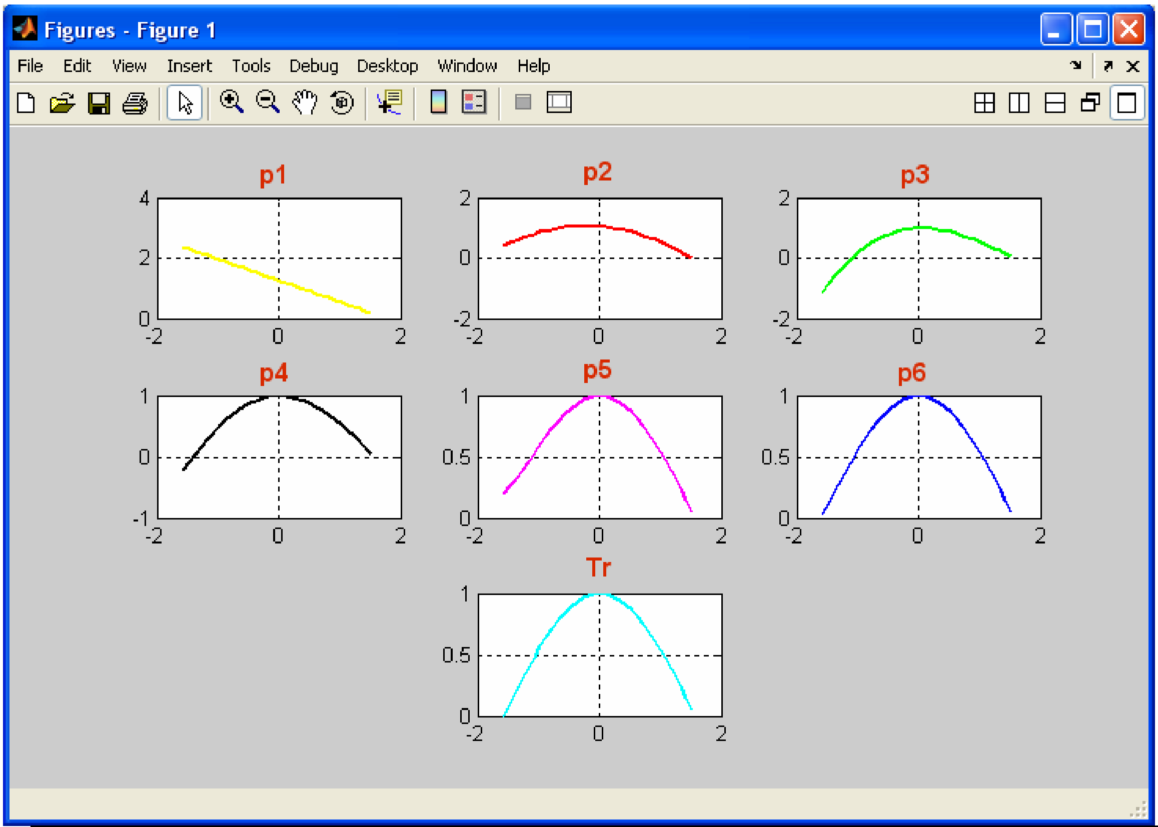
```
%--------------------------------------------------------------------
 clc
 clear
%syms x
%p2=taylor(cos(x),7,pi/4)
format long
x=pi/3
true=cos(pi/3)
p1=1/2*2^(1/2)-1/2*2^(1/2)*(x-1/4*pi)
et_1=((true-p1)/true)*100
p2=1/2*2^(1/2)-1/2*2^(1/2)*(x-1/4*pi)-1/4*2^(1/2)*(x-1/4*pi)^2
et_2=((true-p2)/true)*100
p3=1/2*2^(1/2)-1/2*2^(1/2)*(x-1/4*pi)-1/4*2^(1/2)*(x-
1/4*pi)^2+1/12*2^(1/2)*(x-1/4*pi)^3
et_3=((true-p3)/true)*100
p4=1/2*2^(1/2)-1/2*2^(1/2)*(x-1/4*pi)-1/4*2^(1/2)*(x-
1/4*pi)^2+1/12*2^(1/2)*(x-1/4*pi)^3+1/48*2^(1/2)*(x-1/4*pi)^4
et_4=((true-p4)/true)*100
p5=1/2*2^(1/2)-1/2*2^(1/2)*(x-1/4*pi)-1/4*2^(1/2)*(x-
1/4*pi)^2+1/12*2^(1/2)*(x-1/4*pi)^3+1/48*2^(1/2)*(x-1/4*pi)^4-
1/240*2^(1/2)*(x-1/4*pi)^5
et_5=((true-p5)/true)*100
p6=1/2*2^(1/2)-1/2*2^(1/2)*(x-1/4*pi)-1/4*2^(1/2)*(x-
1/4*pi)^2+1/12*2^(1/2)*(x-1/4*pi)^3+1/48*2^(1/2)*(x-1/4*pi)^4-
1/240*2^(1/2)*(x-1/4*pi)^5-1/1440*2^(1/2)*(x-1/4*pi)^6
et_6=((true-p6)/true)*100
%--------------------------------------------------------------------
```

```
x =

   1.047197551196598

true =

   0.500000000000000

p1 =

   0.521986658763282

et_1 =

  -4.397331752656441

p2 =

   0.497754491403425

et_2 =

   0.449101719315004

p3 =

   0.499869146930044

et_3 =

   0.026170613991194

p4 =

   0.500007550810613

et_4 =

  -0.001510162122553

p5 =

   0.500000304000373

et_5 =

   -6.080007448616696e-005

p6 =

   0.499999987798625

et_6 =

    2.440274993187329e-006
```

```
%--------------------------------------------------------------------
clc
clear
x=-pi/2:0.1:pi/2;
p1=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi);
p2=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-1/4*pi).^2;
p3=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-
1/4*pi).^2+1/12*2.^(1/2)*(x-1/4*pi).^3;
p4=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-
1/4*pi).^2+1/12*2.^(1/2)*(x-1/4*pi).^3+1/48*2.^(1/2)*(x-1/4*pi).^4;
p5=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-
1/4*pi).^2+1/12*2.^(1/2)*(x-1/4*pi).^3+1/48*2.^(1/2)*(x-1/4*pi).^4-
1/240*2.^(1/2)*(x-1/4*pi).^5;
p6=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-
1/4*pi).^2+1/12*2.^(1/2)*(x-1/4*pi).^3+1/48*2.^(1/2)*(x-1/4*pi).^4-
1/240*2.^(1/2)*(x-1/4*pi).^5-1/1440*2.^(1/2)*(x-1/4*pi).^6;
tr=cos(x);
plot(x,p1,'y'),pause(1)
hold on
plot(x,p2,'r'),pause(1)
hold on
plot(x,p3,'g'),pause(1)
hold on
plot(x,p4,'k'),pause(1)
hold on
plot(x,p5,'m'),pause(1)
hold on
plot(x,p6,'b'),pause(1)
hold on
plot(x,tr,'c'),pause(1)
hold on
legend('p1','p2','p3','p4','p5','p6','tr')
grid
%--------------------------------------------------------------------
```


Figures - Figure 1
File Edit View Insert Tools Debug Desktop Window Help
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
p1
p2
p3
p4
p5
p6
tr

```
%----------------------------------------------------------------------
clc
clear
x=-pi/2:0.1:pi/2;
p1=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi);
p2=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-1/4*pi).^2;
p3=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-
1/4*pi).^2+1/12*2.^(1/2)*(x-1/4*pi).^3;
p4=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-
1/4*pi).^2+1/12*2.^(1/2)*(x-1/4*pi).^3+1/48*2.^(1/2)*(x-1/4*pi).^4;
p5=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-
1/4*pi).^2+1/12*2.^(1/2)*(x-1/4*pi).^3+1/48*2.^(1/2)*(x-1/4*pi).^4-
1/240*2.^(1/2)*(x-1/4*pi).^5;
p6=1/2*2.^(1/2)-1/2*2.^(1/2)*(x-1/4*pi)-1/4*2.^(1/2)*(x-
1/4*pi).^2+1/12*2.^(1/2)*(x-1/4*pi).^3+1/48*2.^(1/2)*(x-1/4*pi).^4-
1/240*2.^(1/2)*(x-1/4*pi).^5-1/1440*2.^(1/2)*(x-1/4*pi).^6;
tr=cos(x);
subplot(331)
plot(x,p1,'y'),pause(4)
title('p1')
grid on
subplot(332)
plot(x,p2,'r'),pause(4)
title('p2')
grid on
subplot(333)
plot(x,p3,'g'),pause(4)
title('p3')
grid on
subplot(334)
plot(x,p4,'k'),pause(4)
title('p4')
grid on
subplot(335)
plot(x,p5,'m'),pause(4)
title('p5')
grid on
subplot(336)
plot(x,p6,'b'),pause(4)
title('p6')
grid on
subplot(338)
plot(x,tr,'c')
title('Tr')
grid on
%----------------------------------------------------------------------
```



Example

Find first and second derivatives for F(x)=x^2+2x+2

Solution

```
%------To find first and second derivatives of Pn(x)-----
--
clc
a=[1 2 3];
syms x
p=a(1);
for i=1;
    p=a(i+1)+x*p;
end
disp('First derivative')
  p2=p+x*diff(p)
disp('Second derivative')
  p22=diff(p2)
```

```
%-----------------------------------------------
First derivative

  p2 =

      2+2*x


Second derivative

  p22 =

       2
```

Example

P4(x)=3x^4-10x^3-48x^2-2x+12 at r=6 deflate the polynomial with Horners algorithm Find P3(x).

Solution

```
%------------Horner alogorithm-----------------
clc
a=[3 -10 -48 -2 12];
r=6;
b(1)=a(1);
p=0;
n=length(a);
for i=2:n;
    b(i)=a(i)+r.*b(i-1);
end
syms x
for i=1:n;
    p=p+b(i)*x^(4-i);
end
disp('P3(x)=')
p
%-----------------------------------------------

P3(x)=
        3*x^3+8*x^2-2
```

Numerical Differentiation

1-Finite Difference Approximations

The derivation of the finite difference approximations for the derivatives of *f* (*x*) are based on forward and backward Taylor series expansions of *f* (*x*) about *x*, such as

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) + \cdots \quad \text{(a)}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) - \cdots \quad \text{(b)}$$

$$f(x+2h) = f(x) + 2hf'(x) + \frac{(2h)^2}{2!}f''(x) + \frac{(2h)^3}{3!}f'''(x) + \frac{(2h)^4}{4!}f^{(4)}(x) + \cdots \quad \text{(c)}$$

$$f(x-2h) = f(x) - 2hf'(x) + \frac{(2h)^2}{2!}f''(x) - \frac{(2h)^3}{3!}f'''(x) + \frac{(2h)^4}{4!}f^{(4)}(x) - \cdots \quad \text{(d)}$$

We also record the sums and differences of the series:

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{12}f^{(4)}(x) + \cdots \quad \text{(e)}$$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3}f'''(x) + \cdots \quad \text{(f)}$$

$$f(x+2h) + f(x-2h) = 2f(x) + 4h^2 f''(x) + \frac{4h^4}{3}f^{(4)}(x) + \cdots \quad \text{(g)}$$

$$f(x+2h) - f(x-2h) = 4hf'(x) + \frac{8h^3}{3}f'''(x) + \cdots \quad \text{(h)}$$

First Central Difference Approximations

The solution of Eq. (f) for $f'(x)$ is

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}f'''(x) - \cdots$$

Keeping only the first term on the right-hand side, we have

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2)$$

which is called the *first central difference approximation* for $f'(x)$. The term $\mathcal{O}(h^2)$ reminds us that the truncation error behaves as h^2.

From Eq. (e) we obtain

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \frac{h^2}{12}f^{(4)}(x) + \cdots$$

or

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \mathcal{O}(h^2)$$

Central difference approximations for other derivatives can be obtained from Eqs. (a)–(h) in a similar manner. For example, eliminating $f'(x)$ from Eqs. (f) and (h) and solving for $f'''(x)$ yield

$$f'''(x) = \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3} + \mathcal{O}(h^2)$$

The approximation

$$f^{(4)}(x) = \frac{f(x+2h) - 4f(x+h) + 6f(x) - 4f(x-h) + f(x-2h)}{h^4} + \mathcal{O}(h^2)$$

First Noncentral Finite Difference Approximations

These expressions are called *forward* and *backward* finite difference approximations.

Noncentral finite differences can also be obtained from Eqs. (a)–(h). Solving Eq. (a) for $f'(x)$ we get

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2}f''(x) - \frac{h^2}{6}f'''(x) - \frac{h^3}{4!}f^{(4)}(x) - \cdots$$

Keeping only the first term on the right-hand side leads to the *first forward difference approximation*

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)$$

Similarly, Eq. (b) yields the *first backward difference approximation*

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \mathcal{O}(h)$$

Note that the truncation error is now $\mathcal{O}(h)$, which is not as good as the $\mathcal{O}(h^2)$ error in central difference approximations.

We can derive the approximations for higher derivatives in the same manner. For example, Eqs. (a) and (c) yield

$$f''(x) = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} + \mathcal{O}(h)$$

Second Noncentral Finite Difference Approximations

Finite difference approximations of $\mathcal{O}(h)$ are not popular due to reasons that will be explained shortly. The common practice is to use expressions of $\mathcal{O}(h^2)$. To obtain noncentral difference formulas of this order, we have to retain more terms in the Taylor series. As an illustration, we will derive the expression for $f'(x)$. We start with Eqs. (a) and (c), which are

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + \cdots$$

$$f(x+2h) = f(x) + 2hf'(x) + 2h^2 f''(x) + \frac{4h^3}{3}f'''(x) + \frac{2h^4}{3}f^{(4)}(x) + \cdots$$

We eliminate $f''(x)$ by multiplying the first equation by 4 and subtracting it from the second equation. The result is

$$f(x+2h) - 4f(x+h) = -3f(x) - 2hf'(x) + \frac{2h^2}{3}f'''(x) + \cdots$$

Therefore,

$$f'(x) = \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} + \frac{h^2}{3}f'''(x) + \cdots$$

or

$$f'(x) = \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} + \mathcal{O}(h^2)$$

This Equation is called the *second forward finite difference approximation*.

EXAMPLE

Use forward difference approximations of oh to estimate the first

% derivative of

fx = -0.1.*x.^4-0.15.*x.^3-0.5.*x.^2-0.25.*x+1.2

solution

```
%----------------------------------------------------------------
% Use forward difference approximations to estimate the first
% derivative of fx=-0.1.*x.^4-0.15.*x.^3-0.5.*x.^2-0.25.*x+1.2
clc
h=0.5;
x=0.5;
x1=x+h
fxx=[-0.1 -0.15 -0.5 -0.25 1.2]
fx=polyval(fxx,x)
fx1=polyval(fxx,x1)
tr_va=polyval(polyder(fxx),0.5)
fda=(fx1-fx)/h
et=(tr_va1-fda)/(tr_va1)*100
%-----------------------------------------------------------------
```

EXAMPLE

Comparison of numerical derivative for backward difference and central difference method with true derivative and with standard deviation of 0.025

x = [0:pi/50:pi];

yn = sin(x)+0.025

True derivative=td=cos(x)

solution

```
%--------------------------------------------------------
clc
% Comparison of numerical derivative algorithms
x = [0:pi/50:pi];
n = length(x);
% Sine signal with Gaussian random error
yn = sin(x)+0.025*randn(1,n);
% Derivative of noiseless sine signal
td = cos(x);
% Backward difference estimate noisy sine signal
dynb = diff(yn)./diff(x);
subplot(2,1,1)
plot(x(2:n),td(2:n),x(2:n),dynb,'o')
xlabel('x')
ylabel('Derivative')
axis([0 pi -2 2])
legend('True derivative','Backward difference')
% Central difference
dync = (yn(3:n)-yn(1:n-2))./(x(3:n)-x(1:n-2));
subplot(2,1,2)
plot(x(2:n-1),td(2:n-1),x(2:n-1),dync,'o')
xlabel('x')
ylabel('Derivative')
axis([0 pi -2 2])
legend('True derivative','Central difference')
%--------------------------------------------------------
```

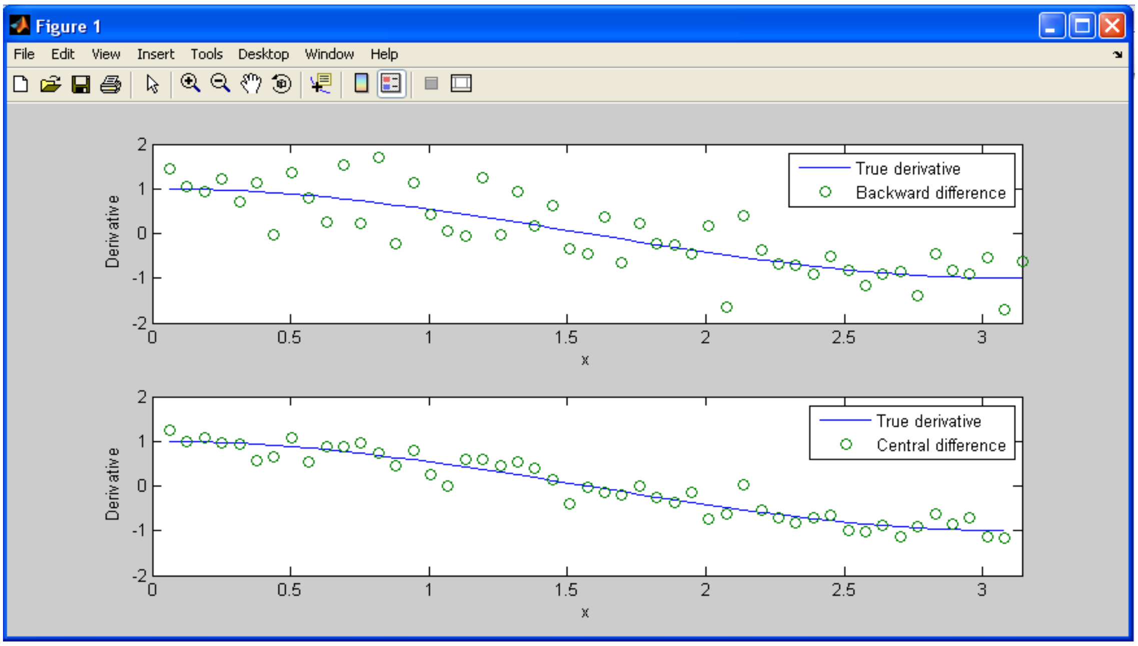


Figure. Comparison of backward difference and central difference methods

Example

Consider a `Divided Difference table` **for points following**

x	0	0.5	1	1.5
$f(x)$	0.0000	1.1487	2.7183	4.9811

Solution

x_k	$f[x_k]$	$f[x_k, x_{k+1}]$	$f[x_k, .., x_{k+2}]$	$f[x_k, .., x_{k+3}]$
0.0	<u>0.0000</u>			
		<u>2.2974</u>		
0.5	1.1487		<u>0.8418</u>	
		3.1392		<u>0.36306</u>
1.0	2.7183		1.3864	
		4.5256		
1.5	4.9811			

$$
\begin{aligned}
p(x) &= f(x_0) + x - x_0 f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2] + (x - x_0)(x - x_1)(x - x_2) f[x_0, x_1, x_2, x_3] \\
&= 0.00 + (x - 0.0)2.2974 + (x - 0.0)(x - 0.5)0.8418 + (x - 0.0)(x - 0.5)(x - 1.0)0.36306 \\
&= 2.05803x + 0.29721x^2 + 0.36306x^3
\end{aligned}
$$

```matlab
%----------------Divided Difference table algorithm-------------
clc
disp('******** divided difference table *********')
x=[2 4 6 8 10]
y=[4.077 11.084 30.128 81.897 222.62]
     f00=y(1);
    for i=1:4
        f1(i)=(y(i+1)-y(i))/(x(i+1)-x(i));
        f01=f1(1);
    end
 f1=[f1(1) f1(2) f1(3) f1(4)]
    for i=1:3
        f2(i)=(f1(i+1)-f1(i))/(x(i+2)-x(i));
        f02=f2(1);
    end
 f2=[f2(1) f2(2) f2(3)]
    for i=1:2
        f3(i)=(f2(i+1)-f2(i))/(x(i+3)-x(i));
        f03=f3(1);
    end
 f3=[f3(1) f3(2)]
    disp('*************************************')
y=input('enter value of y:')
p4x=f00+((y-x(1))*f01)+((y-x(1))*(y-x(2))*f02+((y-x(1))*(y-
x(2))*f02))
fprintf('\np4(%3.3f)=%5.4f',y,p4x)
syms y
p4x=f00+((y-x(1))*f01)+((y-x(1))*(y-x(2))*f02+((y-x(1))*(y-
x(2))*f02))
%------------------------------------------------------------------
f1 =  3.5035    9.5220   25.8845   70.3615

f2 =       1.5046    4.0906   11.1193

f3 =              0.4310    1.1714

p4x = -293/100+7007/2000*y+12037/4000*(y-2)*(y-4)
enter value of y:8
y = 8
p4(8.000)=97.3200

% ------------------------------------------------------------------
```

Example { H.W }

Find the divided differences (newten's Interpolating) for the data and compare with lagrange interpolating.

X	1	1/2	3
F(x)	3	-10	2

Solution

```
************** divided difference table *****************

f1 =

      26.000000000000000   4.800000000000000


f2 =

               -10.600000000000000
----------------Divided Difference table algorithm-------
----------------{ newtens Interpolating }----------------
enter value of y:5

p4(5.000)=-83.8000
px = -283/10-53/5*y^2+419/10*y



----------------------compare with ----------------------
-------------Lagranges interpolation method--------------
 enter value of x:5

 p(5.000)=-83.8000
 p = -283/10-53/5*m^2+419/10*m
```

```
%------------------------Solve H.W-----------------------------
%----------------Divided Difference table algorithm-------------
%----------------{ newten's Interpolating }---------------------
clc
disp('******** divided difference table *********')
x=[1 0.5 3];
y=[3 -10 2];
     f00=y(1);
    for i=1:2;
        f1(i)=(y(i+1)-y(i))/(x(i+1)-x(i));
        f01=f1(1);
    end
 f1=[f1(1) f1(2)]
    for i=1;
        f2(i)=(f1(i+1)-f1(i))/(x(i+2)-x(i));
        f02=f2(1);
    end
 f2=f2(1)
disp('----------------Divided Difference table algorithm-----------')
disp('----------------{ newtens Interpolating }--------------------')
y=input('enter value of y:');
px=f00+((y-x(1))*f01)+((y-x(1))*(y-x(2))*f02);
fprintf('\npx(%3.3f)=%5.4f',y,px)
syms y
px=f00+((y-x(1))*f01)+((y-x(1))*(y-x(2))*f02);
px=collect(px)
%----------------------compare with --------------------------
%-------------Lagrange's interpolation method------------------
disp('---------------------compare with -------------------------')
disp('-------------Lagranges interpolation method------------------')
m=input(' enter value of x:');
p=0;
s=[1 1/2 3];
f=[3 -10 2];
n=length(s);
for i=1:n;
    l=1;
    for j=1:n;
        if (i~=j);
            l=((m-s(j))/(s(i)-s(j)))*l;
        end
      end
  p=l.*f(i)+p;
end
p;
fprintf('\n p(%3.3f)=%5.4f',m,p)
syms m
p=0;
for i=1:n;
    l=1;
    for j=1:n;
        if (i~=j);
            l=((m-s(j))/(s(i)-s(j)))*l;
        end
      end
  p=l.*f(i)+p;
end
p=collect(p)
%--------------------------------------------------------------
```

Example { H.W }

Estimate the In(3) for

Xi	2	4	6
F(x)	In(2)	In(4)	In(6)

a) Linear Interpolation.
B) Quardratic Interpolation
compare between a&b

Solution

a) Linear Interpolation.

```
F1(x)=f(x0)+((f(x1)-f(x0))/(x1-x0))*(x-x0)
```

b) Quardratic Interpolation

```
f2(x)=b0+b1*(x-x0)+b2*(x-x0)*(x-x1)
```

b0= f(x0) = 0.693147180559945;
b1= (f(x1)-f(x0))/(x1-x0) = 0.346573590279973
b2= ((f(x2)-f(x1))/(x2-x1))-b1/(x2-x0) = -0.035960259056473;

```
---------a) Linear Interpolation---------------------

fx1 = 0.693147180559945-0.346573590279973 (X-2)
            inter value x:3
fx1 = 1.039720770839918

---------b) Quardratic Interpolation-----------------
```

fx2 = 0.346573590279973X+(-0.035960259056473X+0.071920518112945)*(X-4)

```
           inter value x:3
fx2 = 1.075681029896391
----------  compare between a&b----------------------
---------a) Linear Interpolation---------------------

Et1 =5.360536964281382 %

---------b) Quardratic Interpolation-----------------

Et2 = 2.087293124994937 %
```

Quardratic Interpolation is better than Linear Interpolation

```
%----------  Solve H.W------------------------------------------
%---------a) Linear Interpolation-----------------------------
%---------b) Quardratic Interpolation-----------------------
%----------  compare between a&b---------------------------
clc
x=input('inter value x:');
format long
xi=[2  4  6];
fx=[log(2)  log(4)  log(6)];
disp('---------a) Linear Interpolation----------------------')
fx1=fx(1)+((fx(2)-fx(1))/(xi(2)-xi(1)))*(x-xi(1))
disp('---------b) Quardratic Interpolation----------------')
b0=fx(1);
b1=(fx(2)-fx(1))/(xi(2)-xi(1));
b2=(((fx(3)-fx(2))/(xi(3)-xi(2)))-b1)/(xi(3)-xi(1));
fx2=b0+b1*(x-xi(1))+b2*(x-xi(1))*(x-xi(2));
% pretty(fx2)%expand(fx2)%collect(fx2)
disp('----------  compare between a&b---------------------')
Tv=log(3);
disp('---------a) Linear Interpolation----------------------')
Et1=abs((Tv-fx1)/Tv)*100
disp('---------b) Quardratic Interpolation----------------')
Et2=abs((Tv-fx2)/Tv)*100
if Et1>Et2;
   disp('Quardratic Interpolation is better than Linear Interpolation')
else
   disp('Linear Interpolation is better than Quardratic Interpolation')
end
syms x
disp('---------a) Linear Interpolation----------------------')
fx1=fx(1)+((fx(2)-fx(1))/(xi(2)-xi(1)))*(x-xi(1))
disp('---------b) Quardratic Interpolation----------------')
b0=fx(1);
b1=(fx(2)-fx(1))/(xi(2)-xi(1));
b2=(((fx(3)-fx(2))/(xi(3)-xi(2)))-b1)/(xi(3)-xi(1));
fx2=b0+b1*(x-xi(1))+b2*(x-xi(1))*(x-xi(2))
```

The Bisection Method for Root Approximation

we can compute the midpoint x_m of the interval $x_1 \le x \le x_2$ with

$$x_m = \frac{x_1 + x_2}{2}$$

Knowing x_m, we can find $f(x_m)$. Then, the following decisions are made:

1. If $f(x_m)$ and $f(x_1)$ have the same sign, their product will be positive, that is, $f(x_m) \cdot f(x_1) > 0$. This indicates that x_m and x_1 are on the left side of the *x–axis* crossing as shown in Figure. In this case, we replace x_1 with x_m.

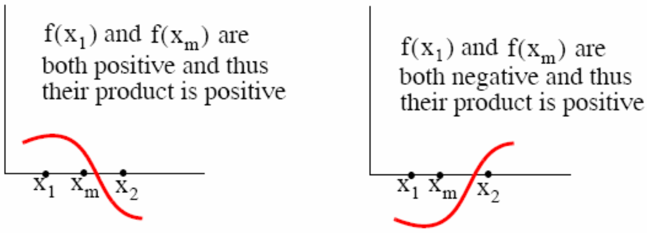


Figure . Sketches to illustrate the bisection method when $f(x_1)$ *and* $f(x_m)$ *have same sign*

2. If $f(x_m)$ and $f(x_1)$ have opposite signs, their product will be negative, that is, $f(x_m) \cdot f(x_1) < 0$. This indicates that x_m and x_2 are on the right side of the *x–axis* crossing as in Figure. In this case, we replace x_2 with x_m.

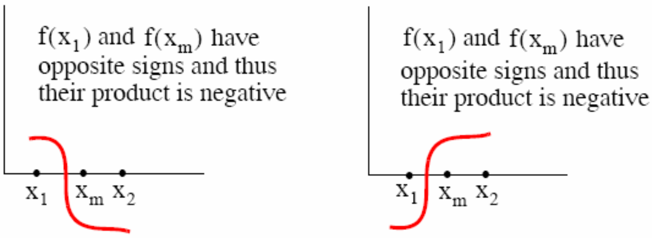


Figure . Sketches to illustrate the bisection method when $f(x_1)$ *and* $f(x_m)$ *have opposite signs*

After making the appropriate substitution, the above process is repeated until the root we are seeking has a specified tolerance. To terminate the iterations, we either:

a. specify a number of iterations

b. specify a tolerance on the error of $f(x)$

Example

Use the Bisection Method with MATLAB to approximate one of the roots of

$$y = f(x) = 3x^5 - 2x^3 + 6x - 8$$

by

a. by specifying ***16*** iterations, and using a for end loop MATLAB program

b. by specifying ***0.00001*** tolerance for f(x), and using a while end loop MATLAB program

Solution:

```
%-------------------------------------------------------------------
function y= funcbisect01(x);
y = 3 .* x .^ 5 - 2 .* x .^ 3 + 6 .* x - 8;
% We must not forget to type the semicolon at the end of the line
above;
% otherwise our script will fill the screen with values of y
%-------------------------------------------------------------------
```

call for function under name funcbisect01.m

```
%-------------------------------------------------------------------
clc
x1=1;
x2=2;
disp('-------------------------')
disp('   xm              fm')  % xm is the average of x1 and x2, fm is
f(xm)
disp('-------------------------')        % insert line under xm and
fm
for k=1:16;
f1=funcbisect01(x1); f2=funcbisect01(x2);
xm=(x1+x2) / 2; fm=funcbisect01(xm);
fprintf('%9.6f %13.6f \n', xm,fm)        % Prints xm and fm on same
line;
  if (f1*fm<0)
    x2=xm;
   else
    x1=xm;
  end
end
disp('-------------------------')
x=1:0.05:2;
y = 3 .* x .^ 5 - 2 .* x .^ 3 + 6 .* x - 8;
plot(x,y)
grid
%-------------------------------------------------------------------
```

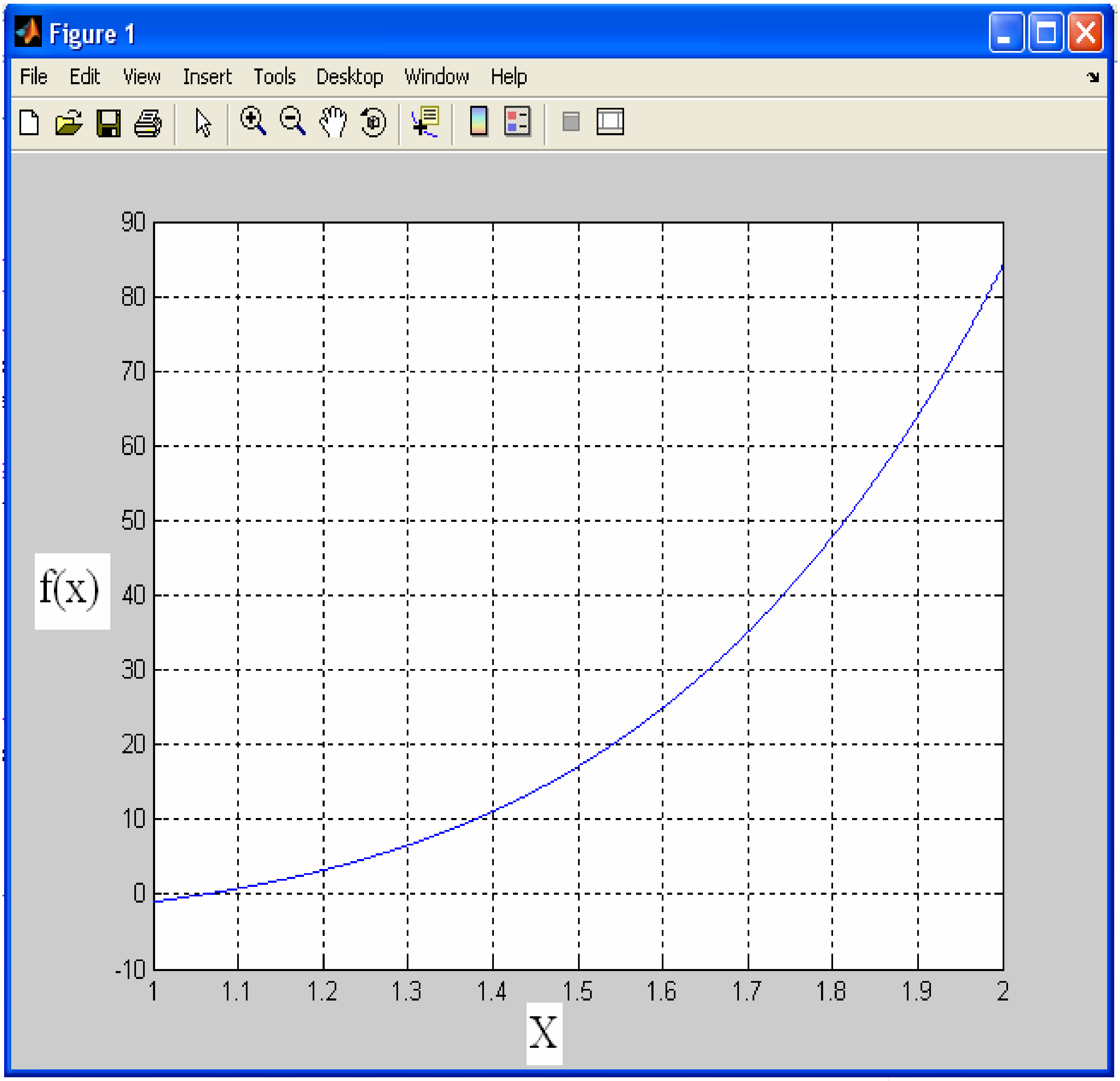


Figure 1
File Edit View Insert Tools Desktop Window Help
f(x)
X
90
80
70
60
50
40
30
20
10
0
-10
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2

```
%-------------------------------------------------------------------
function y= funcbisect01(x);
y = 3 .* x .^ 5 - 2 .* x .^ 3 + 6 .* x - 8;
% We must not forget to type the semicolon at the end of the line
above;
% otherwise our script will fill the screen with values of y
%-------------------------------------------------------------------
```

call for function under name funcbisect01.m

```
%-------------------------------------------------------------------
%-------------------------------------------------------------------
clc
x1=1;
x2=2;
tol=0.00001;
disp('-------------------------')
disp(' xm                 fm');
disp('-------------------------')
while (abs(x1-x2)>2*tol);
f1=funcbisect01(x1);
f2=funcbisect01(x2);
xm=(x1+x2)/2;
fm=funcbisect01(xm);
fprintf('%9.6f %13.6f \n', xm,fm);
if (f1*fm<0);
x2=xm;
else
x1=xm;
end
end
disp('-------------------------')
%-------------------------------------------------------------------
```

```
-------------------------
 xm                 fm
-------------------------
 1.500000     17.031250
 1.250000      4.749023
 1.125000      1.308441
 1.062500      0.038318
 1.031250     -0.506944
 1.046875     -0.241184
 1.054688     -0.103195
 1.058594     -0.032885
 1.060547      0.002604
 1.059570     -0.015168
 1.060059     -0.006289
 1.060303     -0.001844
 1.060425      0.000380
 1.060364     -0.000732
 1.060394     -0.000176
 1.060410      0.000102
-------------------------
```

Example

Use the Bisection Method with MATLAB to approximate one of the roots of (to find the roots of)

Y=f(x)= x.^3-10.*x.^2+5;

That lies in the interval (0.6,0.8) by specifying ***0.00001*** tolerance for f(x), and using a while end loop MATLAB program

Solution:

```
%-----------------------------------------------------------------------
function y= funcbisect01(x);
y = x.^3-10.*x.^2+5;
% We must not forget to type the semicolon at the end of the line
above;(% otherwise our script will fill the screen with values of y)
%-----------------------------------------------------------------------
```

call for function under name funcbisect01.m

```
%-----------------------------------------------------------------------
clc
x1=0.6; x2=0.8;tol=0.00001;
disp('-------------------------')
disp(' xm                 fm');
disp('-------------------------')
while (abs(x1-x2)>2*tol);
f1=funcbisect01(x1);
f2=funcbisect01(x2);
xm=(x1+x2)/2;
fm=funcbisect01(xm);
fprintf('%9.6f %13.6f \n', xm,fm);
if (f1*fm<0);
x2=xm;
else
x1=xm;
end
end
disp('-------------------------')
%-----------------------------------------------------------------------
```

xm	fm
0.700000	0.443000
0.750000	-0.203125
0.725000	0.124828
0.737500	-0.037932
0.731250	0.043753
0.734375	0.002987
0.735938	-0.017453
0.735156	-0.007228
0.734766	-0.002120
0.734570	0.000434
0.734668	-0.000843
0.734619	-0.000204
0.734595	0.000115
0.734607	-0.000045

Newton–Raphson Method

The Newton–Raphson formula can be derived from the Taylor series expansion of $f(x)$ about x:

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2 \tag{a}$$

If x_{i+1} is a root of $f(x) = 0$, Eq. (a) becomes

$$0 = f(x_i) + f'(x_i)\,(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2 \tag{b}$$

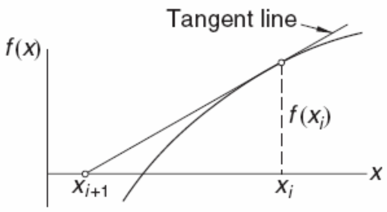
Assuming that x_i is a close to x_{i+1}, we can drop the last term in Eq. (b) and solve for x_{i+1}. The result is the Newton–Raphson formula

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{c}$$

If x denotes the true value of the root, the error in x_i is $E_i = x - x_i$. It can be shown that if x_{i+1} is computed from Eq. (c) , the corresponding error is

$$E_{i+1} = -\frac{f''(x_i)}{2f'(x_i)}E_i^2$$

indicating that the Newton–Raphson method converges *quadratically* (the error is the square of the error in the previous step). As a consequence, the number of significant figures is roughly doubled in every iteration, provided that x_i is close to the root.



Figure(a)Graphical interpretation of the Newton–Raphson formula.

A graphical depiction of the Newton–Raphson formula is shown in Fig.(a) The formula approximates $f(x)$ by the straight line that is tangent to the curve at x_i. Thus x_{i+1} is at the intersection of the x-axis and the tangent line.

The algorithm for the Newton–Raphson method is simple: it repeatedly applies Eq. (c), starting with an initial value x_0, until the convergence criterion

$$|x_{i+1} - x_1| < \varepsilon$$

is reached, ε being the error tolerance. Only the latest value of x has to be stored. Here is the algorithm:

1. Let x be a guess for the root of $f(x) = 0$.
2. Compute $\Delta x = -f(x)/f'(x)$.
3. Let $x \leftarrow x + \Delta x$ and repeat steps 2-3 until $|\Delta x| < \varepsilon$.

EXAMPLE

A root of $f(x) = x^3 - 10x^2 + 5 = 0$ lies close to $x = 0.7$. Compute this root with the Newton–Raphson method.

Solution

The derivative of the function is $f'(x) = 3x^2 - 20x$, so that the Newton–Raphson formula in Eq. (c) is

$$x \leftarrow x - \frac{f(x)}{f'(x)} = x - \frac{x^3 - 10x^2 + 5}{3x^2 - 20x} = \frac{2x^3 - 10x^2 - 5}{x\,(3x - 20)}$$

It takes only two iterations to reach five decimal place accuracy:

$$x \leftarrow \frac{2(0.7)^3 - 10(0.7)^2 - 5}{0.7\,[3(0.7) - 20]} = 0.735\,36$$

$$x \leftarrow \frac{2(0.735\,36)^3 - 10(0.735\,36)^2 - 5}{0.735\,36\,[3(0.735\,36) - 20]} = 0.734\,60$$

Example

Use the Newton–Raphson Method to estimate the root of f(x)=e^(-x)-x, employing an initial guess of x0=0

$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$	$E_{i+1} = -\frac{f''(x_i)}{2f'(x_i)} E_i^2$

Solution

```
%------Newton-Raphson Method-------------------------
clc
x=[0];
tol=0.0000000007;
format long
for i=1:5;
     fx=exp(-x(i))-x(i);
     fxx=-exp(-x(i))-1 ;
     fxxx=exp(-x(i));
     x(i+1)=x(i)-(fx/fxx);
     T.V(i)=(abs((x(i+1)-x(i))/x(i+1)))*100;
end
for i=1:5;
     e(i)=x(6)-x(i);
     fxx=-exp(-x(6))-1 ;
     fxxx=exp(-x(6));
     e(i+1)=(-fxxx/2*fxx)*(e(i))^2;
end
if abs(x(i+1)-x(i))<tol
    disp(' enough to here')
    disp('------------')
    disp('  X(i+1)  ')
    disp('-------------')
    x'
    disp('------------')
    disp('   T.V    ')
    disp('-------------')
    T.V'
    disp('------------')
    disp('   E(i+1)   ')
    disp('-------------')
    e'
    disp('-------------')
end
%-----------------------------------------------------
```

enough to here

X(i+1)

0
0.500000000000000
0.566311003197218
0.567143165034862
0.567143290409781
0.567143290409784

T.V

1.0e+002 *

1.000000000000000
0.117092909766624
0.001467287078375
0.000000221063919
0.000000000000005

E(i+1)

0.567143290409784
0.067143290409784
0.000832287212566
0.000000125374922
0.000000000000003
0.000000000000000

The secant Formula Method

A popular method of hand computation is the *secant formula* where the improved estimate of the root (x_{i+1}) is obtained by linear interpolation based two previous estimates (x_i and x_{i-1}):

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i)$$

Example

Use the The secant Formula Method to estimate the root of f(x)=e^(-x)-x, employing an initial guess of x(i-1)=0 & x(0)=0

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i)$$

Solution

```
%------The secant Formula Method ------------------------
clc
x=[0 1];
TV=0.567143290409784;
format long
for i=2:6;
    fx=exp(-x(i-1))-x(i-1);
    fxx=exp(-x(i))-x(i);
    x(i+1)=x(i)-((x(i)-x(i-1))*fxx)/(fxx-fx);
    E_T(i)=(abs((TV-x(i+1))/TV))*100;
end
    disp('------------')
    disp('  X(i+1)  ')
    disp('-------------')
    x'
    disp('------------')
    disp('   E_T    ')
    disp('-------------')
    E_T'
    disp('------------')

%-----------------------------------------------------
```

```
----------------------
       X(i+1)
----------------------

                   0
   1.000000000000000
   0.612699836780282
   0.563838389161074
   0.567170358419745
   0.567143306604963
   0.567143290409705

---------------------
        E_T
---------------------

                   0
   8.032634281467328
   0.582727734700312
   0.004772693324310
   0.000002855570996
   0.000000000013997

--------------------
```

Example

Use N.R. Quadratically Method to estimate the multiple root of f(x)=x^3-5x^2+7x-3, initial guess of x(0)=0

$$x_{i+1} = x_i - \frac{f(x_i)\, f'(x_i)}{f'(x_i)^2 - f(x_i)\, f''(x_i)}$$

Solution

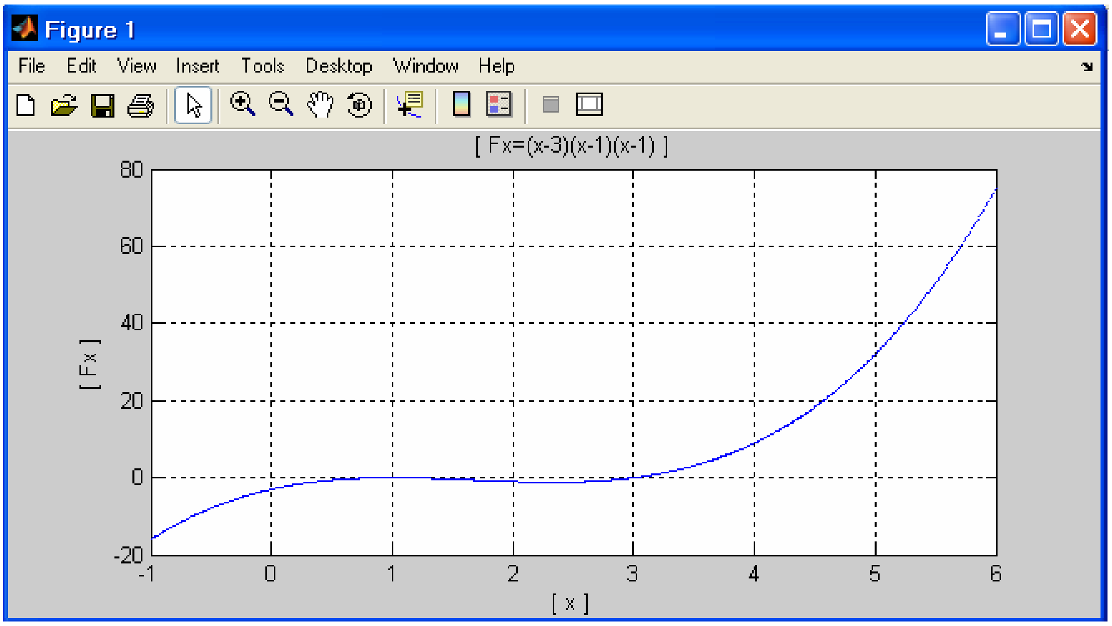
```
%------The N.R. Quadratically  Method   -----------------
clc
TV=1;
x=[0];
format long
for i=1:6;
    fx=x(i)^3-5*x(i)^2+7*x(i)-3
    fxx=3*x(i)^2-10*x(i)+7
    fxxx=6*x(i)-10
    x(i+1)=x(i)-(fx*fxx)/((fxx)^2-fx*fxxx);
    E_T(i)=(abs((TV-x(i+1))/TV))*100;
end
    disp('------------')
    disp('  X(i+1)  ')
    disp('-------------')
    x'
    disp('------------')
    disp('   E_T    ')
    disp('-------------')
    E_T'
    disp('------------')

%--------------------------------------------------
%------Multiple Roots---------
%--fx=(x-3)(x-1)(x-1)---------
clc
for x=-1:0.01:6;
   fx=x.^3-5.*x.^2+7.*x-3
   plot(x,fx)
   hold on
end
grid
title('(x-3)(x-1)(x-1)')
xlabel('x')
ylabel('fx')
%----------------------------
```

X(i+1)

0
1.105263157894737
1.003081664098603
1.000002381493816
1.000000000037312
1.000000000074625
1.000000000074625

E_T

10.526315789473696
0.308166409860333
0.000238149381548
0.000000003731215
0.000000007462475
0.000000007462475



Example

Use the Newton–Raphson Method to estimate the root of f(x)=x^3-5x^2+7x-3, initial guess of x(0)=4

$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$	$E_{i+1} = -\frac{f''(x_i)}{2 f'(x_i)} E_i^2$

Solution

```
%------Newton-Raphson Method--------------------------
clc
x=[4];
tol=0.0007;
TV=3;
format long
for i=1:5;
     fx=x(i)^3-5*x(i)^2+7*x(i)-3;
     fxx=3*x(i)^2-10*x(i)+7;
     x(i+1)=x(i)-(fx/fxx);
     E_T(i)=(abs((TV-x(i+1))/TV))*100;
end
for i=1:5;
     e(i)=x(6)-x(i);
     fx=x(i)^3-5*x(i)^2+7*x(i)-3;
     fxx=3*x(i)^2-10*x(i)+7;
     fxxx=6*x(i)-10;
     e(i+1)=(-fxxx/2*fxx)*(e(i))^2;
end
if abs(TV-x(i+1))<tol
    disp(' enough to here')
    disp('-------------')
    disp('  X(i+1)  ')
    disp('--------------')
    x'
    disp('-------------')
    disp('   T.V    ')
    disp('--------------')
    E_T'
    disp('-------------')
    disp('   E(i+1)   ')
    disp('--------------')
    e'
    disp('--------------')
end
%-----------------------------------------------------
```

```
    enough to here
---------------------
        X(i+1)
---------------------

   4.000000000000000
   3.400000000000000
   3.100000000000000
   3.008695652173913
   3.000074640791192
   3.000000005570623

---------------------
          T.V
---------------------

  13.333333333333330
   3.333333333333322
   0.289855072463781
   0.002488026373060
   0.000000185687436
   0.000000007462475

--------------------
        E(i+1)
--------------------

  -0.999999994429377
  -0.399999994429377
  -0.099999994429377
  -0.008695646603290
  -0.000074635220569
  -0.000000089144954

---------------------
```

ترقبوا المزيد من الشروحات للأمثلة فى التحليل العددى والرياضيات والتحكم الألى والأتصالات والإلكترونات القدرة ونظم التشغيل والدارات التماثلية والنظم الرقمية واسس الألكترونات وغيرها من المواد فى اغلب التخصصات راجين من الله سبحانه وتعالى التوفيق فلله الحمد والمنة وبه التوفيق والعصمة.