M1 Computational Methods
PW2

Mohammed Akib Iftakher
Etudiant Number: 20215282

Lecturer: Professor L. NOUVELIERE & Professor Hichem ARIOUI

PW2 - Interpolation Methods

I- Polynomial interpolation

The following function is given,

$$f(x) = \log(x) - \frac{2(x-1)}{x}$$

(a) Plot this function f under MATLAB, by creating a matlab function f.m.

```
xr = 1:0.01:10;
plot(xr,f(xr),'b--')
title('The function f')
```

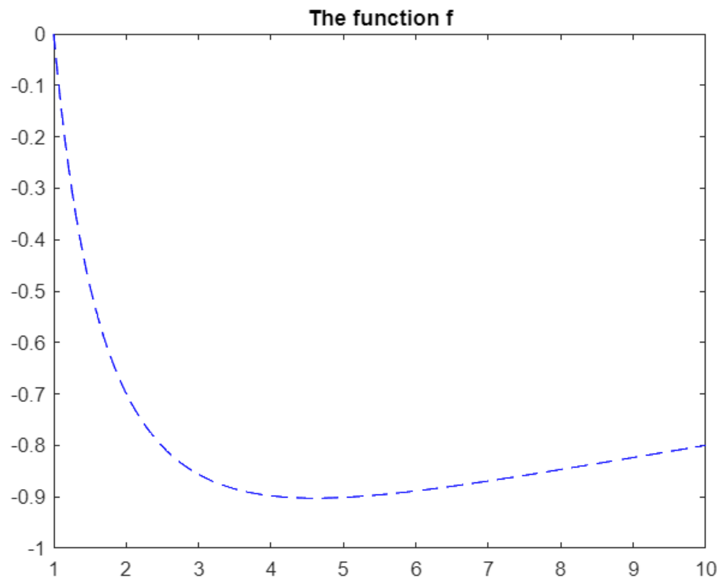


Figure 1: The function f.

b) Calculate the values of the function at points x0 = 1, x1 = 2, x2 = 4, x3 = 8, x4 = 10.

```
x = [1,2,4,8,10];
y = f(x)
```

```
y = 1×5
         0   -0.6990   -0.8979   -0.8469   -0.8000
```

c) Calculate the Lagrange polynomial of degree 4 interpolating this function.

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

```
for i=1:n
    y(i)=f(x(i));
end

syms s;
equa(s)=0*s;
l=ones(1,n);

for i=1:n
    temp=1+0*s;
    for j=1:n
        if i~=j
            l(i)=l(i)/(x(i)-x(j));
            temp=temp*(s-x(j));
        end
    end

    equa(s)=equa(s)+y(i)*l(i)*temp;
end

equa(s)=collect(equa(s))
```

```
 x=[1,2,4,8,10];
 n = length(x);
equa_1 = Lagrange(x,n)
```

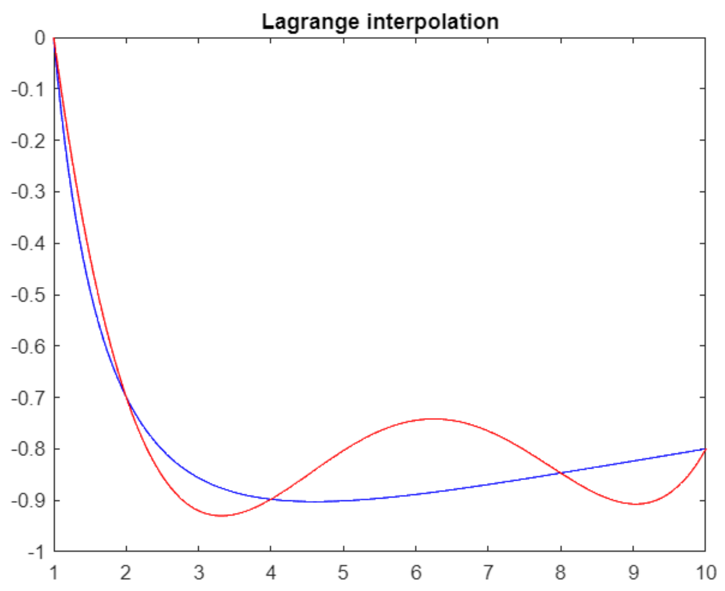


Figure 2: Lagrange Interpolation.

d) Give the number of divisions made to calculate the coefficients.

If the algorithm has been analyzed, it can be noticed that there are two for loops. But in the first loop iterates until n and second loop iterates until n-1 (because i~=j).

So, the total multiplication is n*(n-1). So, in another word, number of divisions made to calculate the coefficients $\frac{1}{n(n-1)} = \frac{1}{5(5-1)} = \frac{1}{20}$.

e) What is the approximate value of the function at points x = 2.9 and x = 5.25? Give the value of the error.

```
x1 = double(equa_1(2.9)) % for 2.9
```

```
x1 = -0.9123
```

```
x2 = double(equa_1(5.25)) % for 5.25
```

```
x2 = -0.7826
```

```
y1 = f(2.9)
```

```
y1 = -0.8479
```

```
y2 = f(5.25)
```

```
y2 = -0.8989
```

```
error1 = y1 - x1
```

```
error1 = 0.0644
```

```
error2 = y2 - x2
```

```
error2 = -0.1163
```

I- B Newton interpolation

$$g(x) = \frac{1}{1+x^2}:$$

a) Plot this function under MATLAB in the interval [-5, 5], creating a function g.m.

```
x = linspace(-5,5);
plot(x,g(x),'r-')
title('The function g')
```

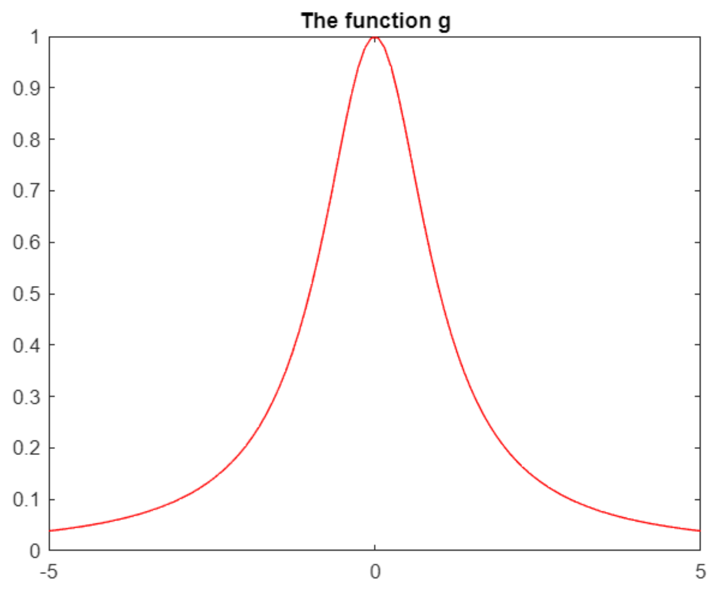


Figure 3: The Runge function

b) Calculate the values of the function at points x0 = -5, x1 = -3, x2 = -1, x3 = 1, x4 = 3, x5 = 5.

```
x_n = [-5:2:5];
n = length(x_n)
```

```
n = 6
```

```
y_n = g(x_n)
```

```
y_n = 1×6
    0.0385    0.1000    0.5000    0.5000    0.1000    0.0385
```

```
% x_n_1 = [-7:2:7];
```

c) Calculate the Newtonian polynomial P of degree n = 5 interpolating this function g. We will program the algorithm of divided differences.

```
P = newton(x_n,n)
```

```
equ(s) =
```

$$\frac{s^4}{520} - \frac{9\,s^2}{130} + \frac{59}{104}$$

```
P(s) =
```

$$\frac{s^4}{520} - \frac{9\,s^2}{130} + \frac{59}{104}$$

d) Write a program making it possible to calculate the value of the polynomial at any point.

```
function equ_new = newton(x,n)

y_new=zeros(1,n);

for i=1:n
    y_new(i)=g(x(i));
end

arr=zeros(n,n);

for i=1:n
    arr(i,1)=y_new(i);
end


for j=2:n
    for i=1:(n-j+1)
        arr(i,j)=(arr(i+1,j-1)-arr(i,j-1) )/( x(i+j-1)-x(i));
    end
end

syms s;
equ_new(s)=0*s;
for i=1:n
    n_i=1;
    for j=1:i-1
        n_i=n_i*(s-x(j));
    end
   equ_new(s)=equ_new(s)+arr(1,i)*n_i;
end
equ_new(s)=collect(equ_new(s))
```

```
P(-5)
```

```
ans =
```

$$\frac{1}{26}$$

e) Plot on the same graph the function g as well as the polynomial P. Examine the edge effects.

```
x_n = [-5:2:5];
n = length(x_n);
P = newton(x_n,n);
```

```
equ(s) =
```

$$\frac{s^4}{520} - \frac{9\,s^2}{130} + \frac{59}{104}$$

```
x_m = [-5:1:5];
m = length(x_m);
P1 = newton(x_m,m);
```

```
equ(s) =
```

$$-\frac{s^{10}}{44200} + \frac{7\,s^8}{5525} - \frac{83\,s^6}{3400} + \frac{2181\,s^4}{11050} - \frac{149\,s^2}{221} + 1$$

```
xt=-5:0.01:5;
plot(xt,g(xt),'b',xt,P(xt),'r-',xt,P1(xt),'g-')
title('Newton interpolation')
```

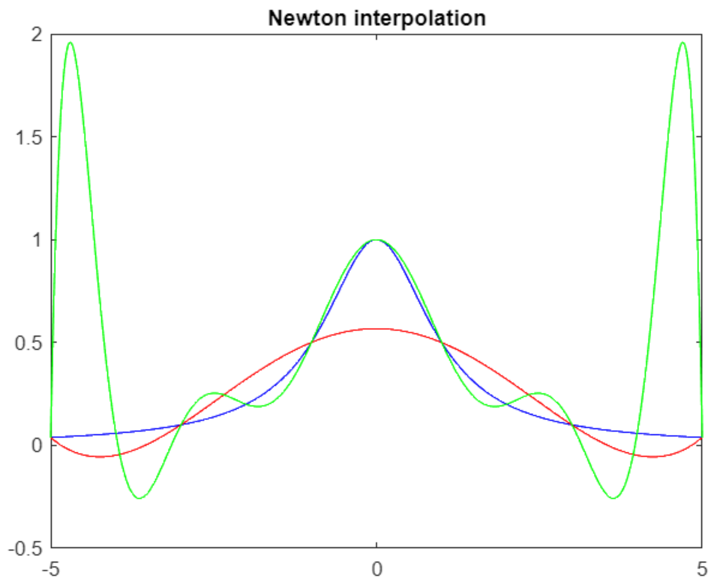


Figure 4: Newton Interpolation.

There are some oscillations around the interval's edges, as can be seen. This is solely attributable to high-degree polynomials over a set of evenly spaced interpolation points. Runge's phenomenon is the name given to this phenomenon.

f) How do these effects evolve with the order of the polynomial?

As mentioned earlier, there are some oscillations. In another word edge effect can be noticed. The number of oscillations will increase according to the polynomial order.

So, it can be said that the more the polynomial order is the more the oscillations can be noticed which is due to more critical points. As a result, the effects evolve more.

g) Go back to the 5 previous questions by taking, as a subdivision of the interval, Chebychev's points on the interval [a, b] = [-5, 5],

```
function equ = chebychev(a,b,n)

x=(b-a)/2*cos( (n-(0:n-1))/n*pi)+(b+a)/2;

for i=1:n
    y(i)=g(x(i));
end

arr=zeros(n,n);

for i=1:n
    arr(i,1)=y(i);
end


for j=2:n
    for i=1:(n-j+1)
        arr(i,j)=(arr(i+1,j-1)-arr(i,j-1) )/( x(i+j-1)-x(i));
    end
end

syms s;
equ(s)=0*s;
for i=1:n
    n_i=1;
    for j=1:i-1
        n_i=n_i*(s-x(j));
    end
   equ(s)=equ(s)+arr(1,i)*n_i;
end
equ(s)=collect(equ(s))

xt=-5:0.01:5;
plot(xt,g(xt),'b',xt,equ(xt),'r-');
```

```
n=6;
b=5;
a=-5;
eq = chebychev(a,b,n)
```

```
equ(s) =
```

$$\frac{40\,s^5}{29783}+\left(\frac{100\,\sqrt{3}}{29783}+\frac{8821560052684819142638812376289 9}{755088756531947125295526231314268 16}\right)s^4+\left(\frac{441078002634240957131940618814495\,\sqrt{3}}{1510177513063894250591052462628536 32}-\frac{10348928349279280991531331267582828375614192508095 99}{26786402009000788684837135614024035184253931647664128}\right)s^3+\left(-\frac{5174464174639}{535728040180}\right.$$

```
eq(s) =
```

$$\frac{40\,s^5}{29783} + \left(\frac{100\ \sqrt{3}}{29783} + \frac{882156005268481914263881237628 99}{7550887565319471252955262313142 6816}\right) s^4 + \left(\frac{441078002634240957131940618814495\ \sqrt{3}}{151017751306389425059105246262853632} - \frac{10348928349279280991531331}{26786402009000788684837135}\right.$$

```
title('Newton interpolation with Chebychev points')
```

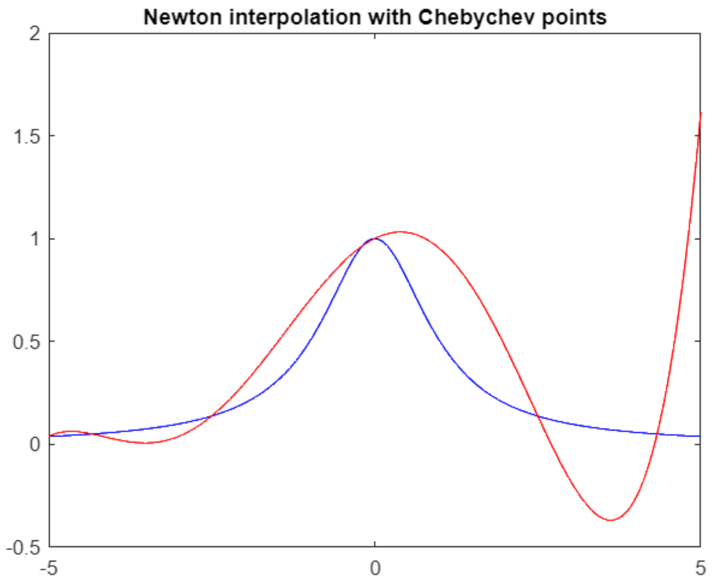


Figure 5: Newton Interpolation with Chebyshev points

h) Conclude.

It may be established that interpolation of higher order newton polynomials leads in unanticipated oscillations (edge effect). Surprisingly, using interpolation with a large number of points can result in more error due to oscillations. As a result, the Chebyshev points are used to reduce the error caused by Runge's phenomenon.

II- Spline interpolation

I- A Quadratic spline

a) Write a general program of interpolation by quadratic splines. The program receives, as input, the pairs (xi, yi), i = 0, 1,…, n as well as the value of the derivative at point x0. The program will return a 3 x n matrix containing the coefficients of the polynomials Si(x).

```
function [a,b,c]= qs(x,y,ydd0)

n=size(x,2);

z=zeros(1,n);
a=zeros(1,n-1);
b=zeros(1,n-1);
c=zeros(1,n-1);

z(1)=ydd0;
```

```
for i=1:n-1
    z(i+1)=2*( y(i+1)-y(i) )/(x(i+1)-x(i)) -z(i);
end

for i=1:n-1
    c(i)=y(i);
    b(i)=z(i);
    a(i)=( z(i+1)-z(i) )/( 2*(x(i+1)-x(i) ) );
end

end
```

```
n=6;

x=[-5,-3,-1,1,3,5];
y=zeros(1,n);

for i=1:n
    y(i)=g(x(i));
end

[a,b,c]=qs(x,y,0)
```

```
a = 1×5
     0.0154     0.0692    -0.1692     0.0692     0.0154
b = 1×5
          0     0.0615     0.3385    -0.3385    -0.0615
c = 1×5
     0.0385     0.1000     0.5000     0.5000     0.1000
```

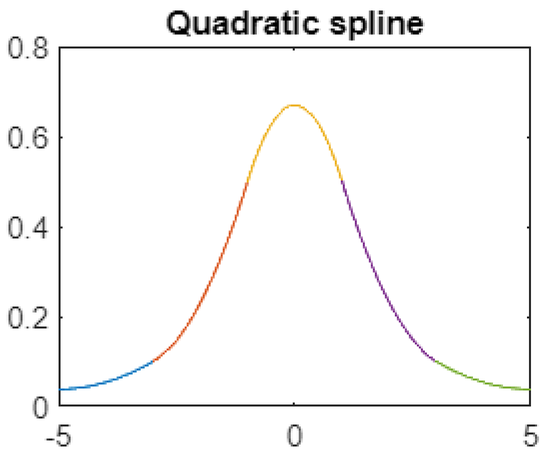


Figure 6: Quadratic Spline Interpolation.

b) Apply to points X = [0, 1, 2, 3]; Y = [0, 0.5, 2.0, 1.5] as well as to the function of Runge.

```
n=4;

x=[0, 1, 2, 3];
y = [0, 0.5, 2.0, 1.5];
[a,b,c]=qs(x,y,0)
```

```
a = 1×3
    0.5000    0.5000   -2.5000
b = 1×3
     0     1     2
c = 1×3
         0    0.5000    2.0000
```

```
for i=1:n-1
    xt=x(i):0.01:x(i+1);
    plot(xt,a(i)*(xt-x(i)).^2+b(i)*(xt-x(i))+c(i) )
    hold on
end

title('Quadratic spline')
```

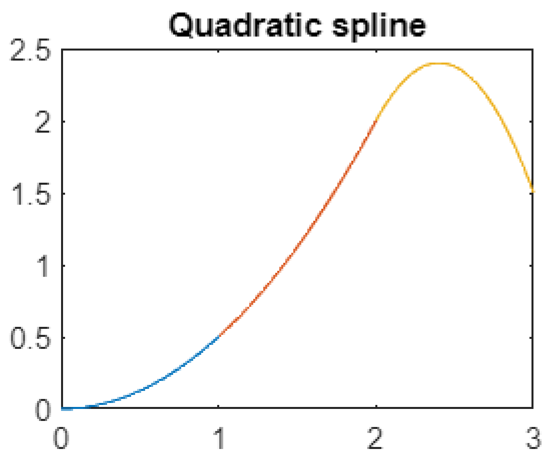


Figure 7: Quadratic spline interpolation.

c) Examine the effect of different values of S'(x0).

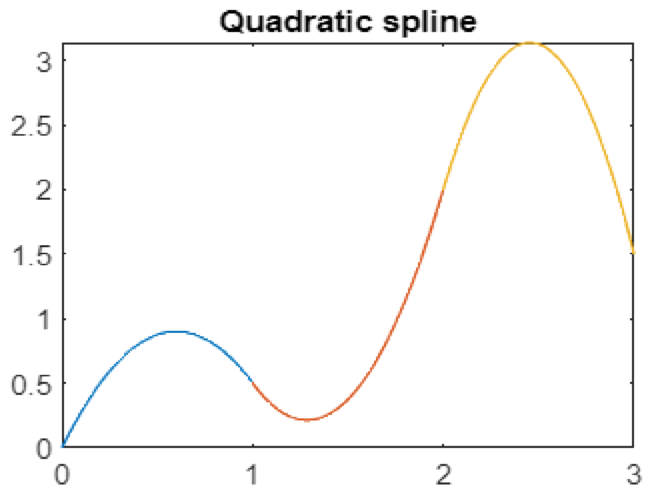


Figure 8: S'(0) is 3.

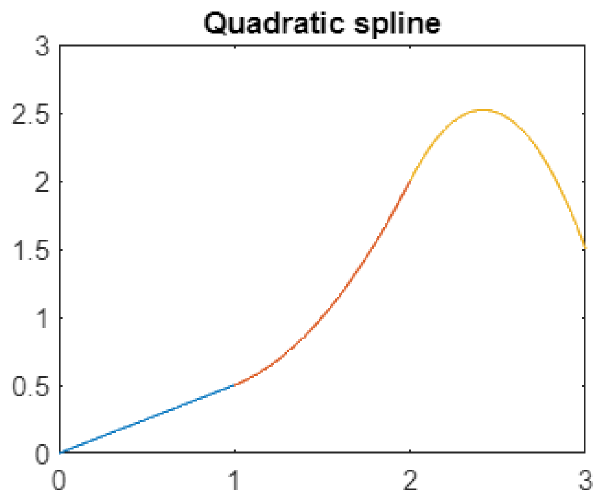


Figure 9: S'(0) is 0.5.

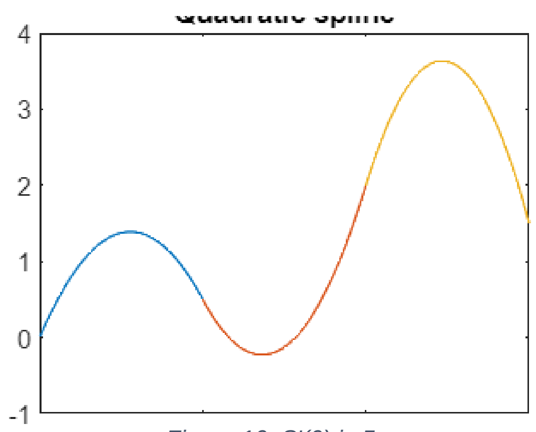


Figure 10: S'(0) is 5.

A spline of degree 2 is a piecewise quadratic function on each subinterval. The first derivative exists and is continuous, but it is not smooth.

We can see by comparing the multiple values of S'(x0) that when we increase and decrease the value of S'(x0) away from zero, the final quadratic spline will have more oscillations. In this respect, selecting smaller numbers closer to zero assists us in obtaining a more piecewise linear interpolation of the curve.

It can also be noticed that the number of the equation is more than unknown value and the tangent is also changing. Moreover, each part of the interpolation has almost one zero.

I- B Cubic spline

a) Write a main program of interpolation by cubic splines. The program receives, as input, the pairs (xi, yi), i = 0, 1,…, n as well as the value of the derivative at point x0. The program will return a 4 x n matrix containing the coefficients of the polynomials Si(x).

```
function [a,b,c,d]= qs(x,y,ydd1,yddn)

n=size(x,2);

A=zeros((n-2),(n-2));
B=zeros((n-2),1);
h=zeros(1,(n-1));
a=zeros(1,n-1);
b=zeros(1,n-1);
c=zeros(1,n-1);
d=zeros(1,n-1);
ydd=zeros(n,1);
ydd(1)=ydd1;
ydd(n)=yddn;
j=0;

for i=1:n-1
    h(i)=x(i+1)-x(i);
end

for i=1:n-2
    if(j>0)
        A(i,j)= h(i);
    end

    A(i,j+1)=2*( h(i)+h(i+1) );

    if(j+2<=n-2)
        A(i,j+2)= h(i+1);
    end
    j=j+1;

    if(i==0)    B(i)= -h(1) * ydd1;   end
    if(i==n-2)  B(i)= -h(n-1)*yddn;   end
    B(i)=-6/h(i)*( y(i+1)-y(i) )+6/h(i+1)*( y(i+2)-y(i+1) );
```

```
end

ydd(2:n-1)=inv(A)*B;

for i=1:n-1
    a(i)=1/(6*h(i))*(ydd(i+1)-ydd(i));
    b(i)=1/2*ydd(i);
    c(i)=1/h(i)*(y(i+1)-y(i))-1/6*h(i)*(ydd(i+1)+2*ydd(i));
    d(i)=y(i);
end

end
```

```
n=7;

x=[-5,-3,-1,0,1,3,5];
y=zeros(1,n);

for i=1:n
    y(i)=g(x(i));
end


[a,b,c,d]=cs(x,y,0,0)
```

```
a = 1×6
   -0.0021    0.0529   -0.4023    0.4023   -0.0529    0.0021
b = 1×6
         0   -0.0127    0.3046   -0.9023    0.3046   -0.0127
c = 1×6
    0.0392    0.0138    0.5977         0   -0.5977   -0.0138
d = 1×6
    0.0385    0.1000    0.5000    1.0000    0.5000    0.1000
```

b) Apply to points X = [0, 1, 2, 3]; Y = [0, 0.5, 2.0, 1.5] as well as to the function of Runge.

```
n=4;

x=[0, 1, 2, 3];
y = [0, 0.5, 2.0, 1.5];

[a,b,c,d]=cs(x,y,0,0);

for i=1:n-1
    xt=x(i):0.01:x(i+1);
    plot(xt,a(i)*(xt-x(i)).^3 + b(i)*(xt-x(i)).^2 + c(i)*(xt-x(i))+ d(i) )
    hold on;
end
hold off
title('Cubic spline')
```

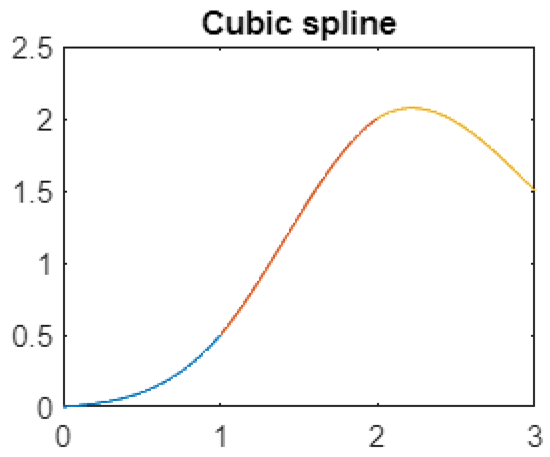


Figure 11: Cubic spline interpolation.

```

n=7;

x=[-5,-3,-1,0,1,3,5];
y=zeros(1,n);

for i=1:n
    y(i)=g(x(i));
end


[a,b,c,d]=cs(x,y,0,0)

for i=1:n-1
    xt=x(i):0.01:x(i+1);
    plot(xt,a(i)*(xt-x(i)).^3 + b(i)*(xt-x(i)).^2 + c(i)*(xt-x(i))+ d(i) )
    hold on;
end
hold off
title('Cubic spline')
```

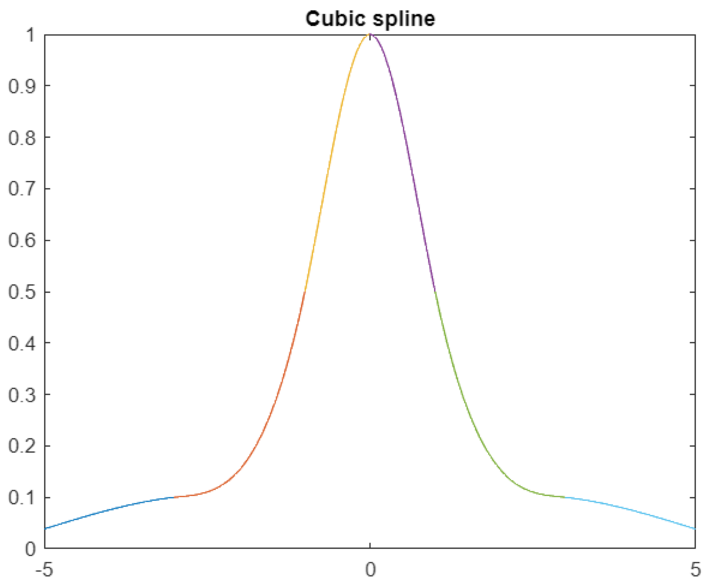
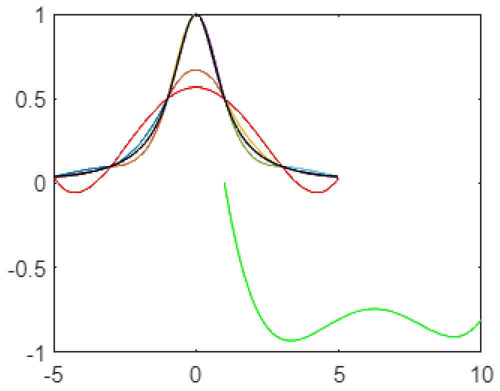


Figure 12: Cubic spline interpolation.

The results obtained by polynomial interpolation, quadratic and cubic splines will be plotted on the same graph. Compare and conclude.



It is clear from the observations that Lagrange interpolation is out of range and Newton interpolation is less smooth and stiff. Although quadratic interpolation does not match (medium smooth and rigid), it is superior to Newton interpolation. In addition, the final cubic interpolation is more rigid and smooth than the prior interpolation. As a result, quadratic and cubic interpolations are more efficient and have lower error rates.

At the data points, the quadratic spline produces derivatives that are not smooth. The cubic spline produces smooth derivatives that are extremely near to the true derivative of the Runge function used to create the data.

Furthermore, splines match derivative conditions at spline intersections rather than derivatives in the data itself. Splines, in other terms, are piecewise functions that approximate data by filling in the gaps between data: their matching criteria provide smooth connections between themselves.