Foundations of Computational Math 2
Program 2
Chenchen Zhou

1. Program description

In this program, we just implement two interpolating spline algorithms. i.e Deriving the cubic interpolating splines from the view that solving the parameters by the given data and by conditions what the splines must be satisfied or from the other different view that cubic splines form a linear space. i.e. find the coefficients for the basis under the uniform grid condition.

2. Code description

I have three main functions. Two test functions.

Main function:
Firstd.m

In this file, I write a function which deriving the interpolating cubic spline via solving Ts'=d. And the function can evaluate the cubic spline value at any x you put in. (of course, x must be within in the interval which we interpolating in)

For the input arguments, I is the vector which saves the nodes you want to interpolating. i.e. x0,x1…xn

Flag is for the choice of the boundary conditions.

Alph and beta are flexible, when testing the cubic polynomials I don't need them, this two extra argument are introduced only when we test the functions

$$f(x) = \frac{\beta}{1 + \alpha x^2}$$

Detail explanation about the alph, beta will be followed.

Secondd.m

In this file, I write a function which deriving the interpolating cubic spline via solving Ts''=d. And the function can evaluate the cubic spline value at any x you put in. (of course, x must be within in the interval which we interpolating in)

B.m

In this file, I write a function which deriving the interpolating cubic spline via solving the coefficients by given that basis. i.e. using the B-splines interpolating cubic splines. As we just consider the uniformly spaced points. So we've know our basis.

Test code

Test1.m

In this file, I test the Firstd function and secondd function with both two kinds of boundary conditions. Choose 1000 linear spaced points and evaluate the functional value at those points. Compare the values between the given function f and the interpolating cubic splines which we derived. Use the infinite norm to see the error and draw the figure to see the quality of interpolating.

Test2.m

In this file, I just test the B function with the only one boundary condition. i.e. match the first derivative at the endpoints. Likewise, choose 1000 linear spaced points and evaluate them. Compute the infinite norm and graph functions.

Other files

F1.m

This file saves the different test functions. I separated them by %, so when you want to test one of them ,just remove the % in front of them.

F11.m

This is just the first derivative function of f1

F12.m

This is just the second derivative function of f1.

Ps:

Modification in codes when apply to test functions $f(x) = \frac{\beta}{1+\alpha x^2}$.

If the test function is cubic polynomials, remove all the alph and beta as you see any of them in any files.

If you want to test the $f(x) = \frac{\beta}{1+\alpha x^2}$

Keep alph and beta as the codes shows.

3. Experiments and conclusions.

Firstd and Secondd

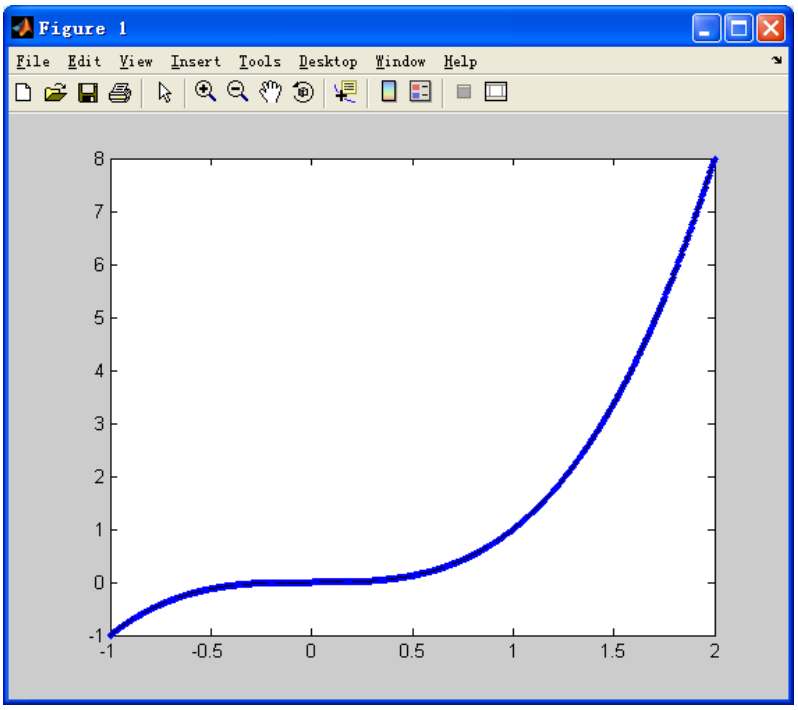
test1([-1,-0.7,-0.5,0,0.8,1.5,2])

nodes choose as [-1,-0.7,-0.5,0,0.8,1.5,2]

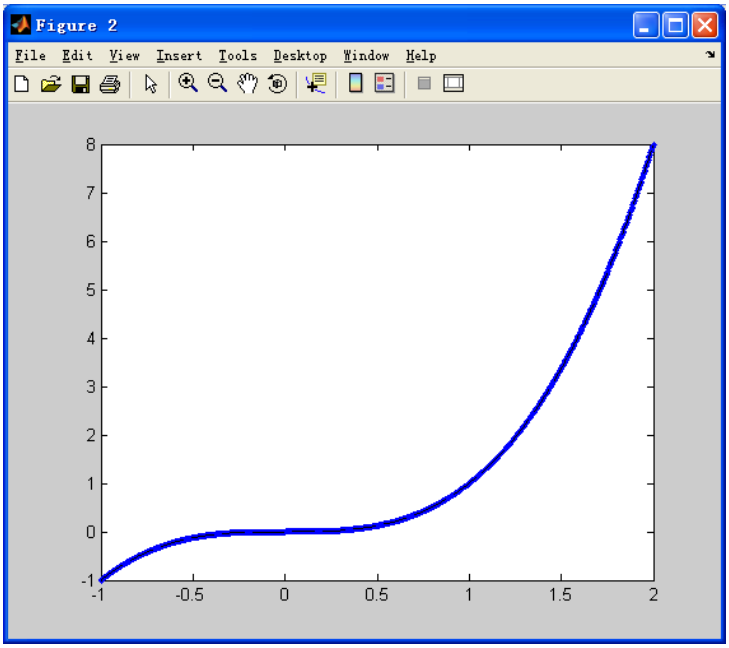
Test function

```
1. y=x^3;
```

y=x^3	flag=1	flag=2
Firstd	1.78E-15	1.78E-15
secondd	8.88E-16	8.88E-16
B	n=6	n=600
	0.3109	0.1548

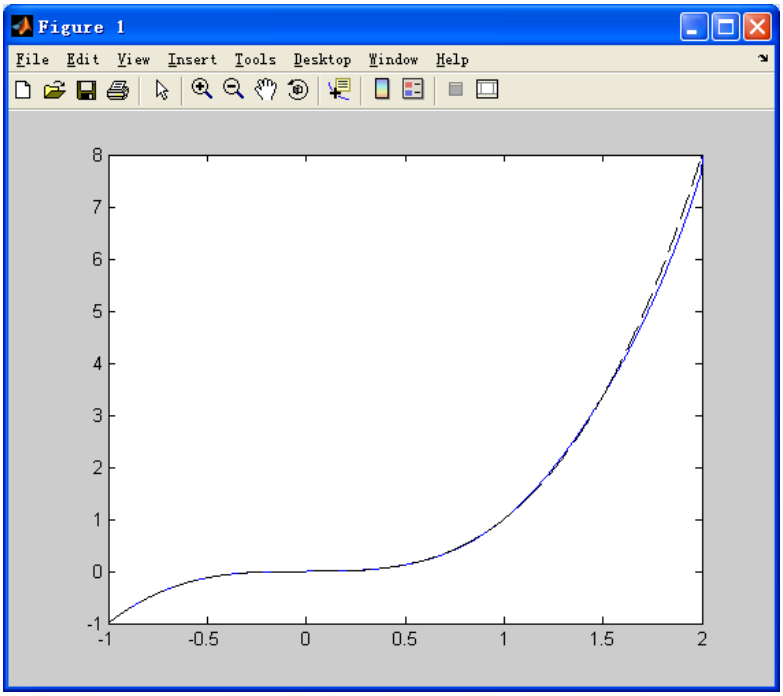



```
Flag=1
```

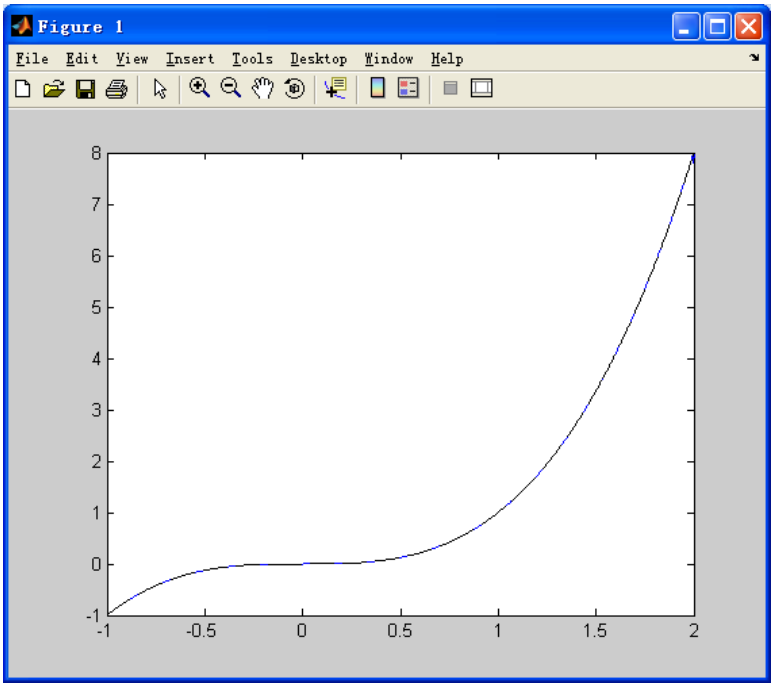


Flag=2(same in the following graphs)

For B function(B splines)



N=6 (n is the number of the subintervals)



N=600

2.y=x^3-x^2;

y=x^3-x^2;	flag=1	flag=2
Firstd	8.88E-16	8.88E-16
second	8.88E-16	8.88E-16
B	n=6	n=600
	0.1485	0.0774

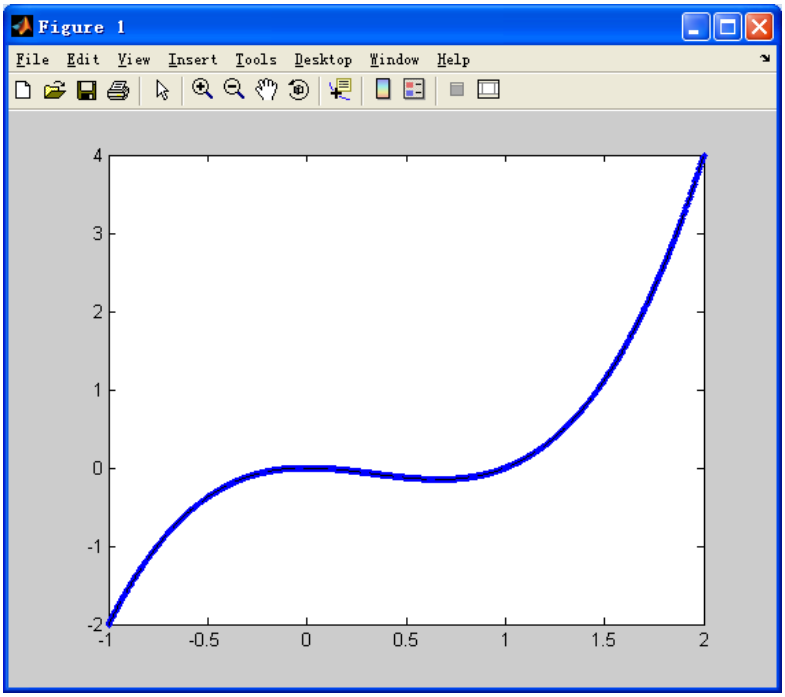


Figure 1
File Edit View Insert Tools Desktop Window Help
4
3
2
1
0
-1
-2
-1
-0.5
0
0.5
1
1.5
2

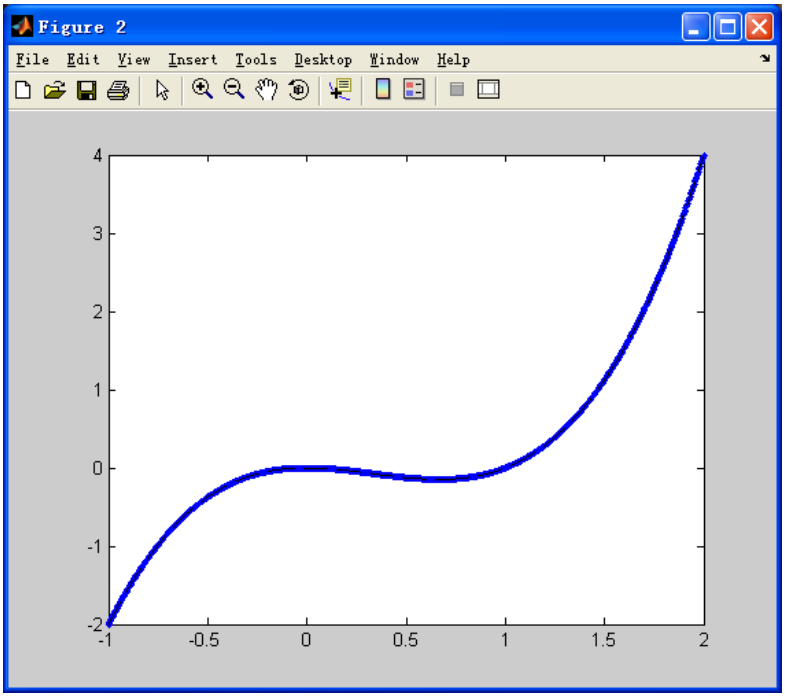
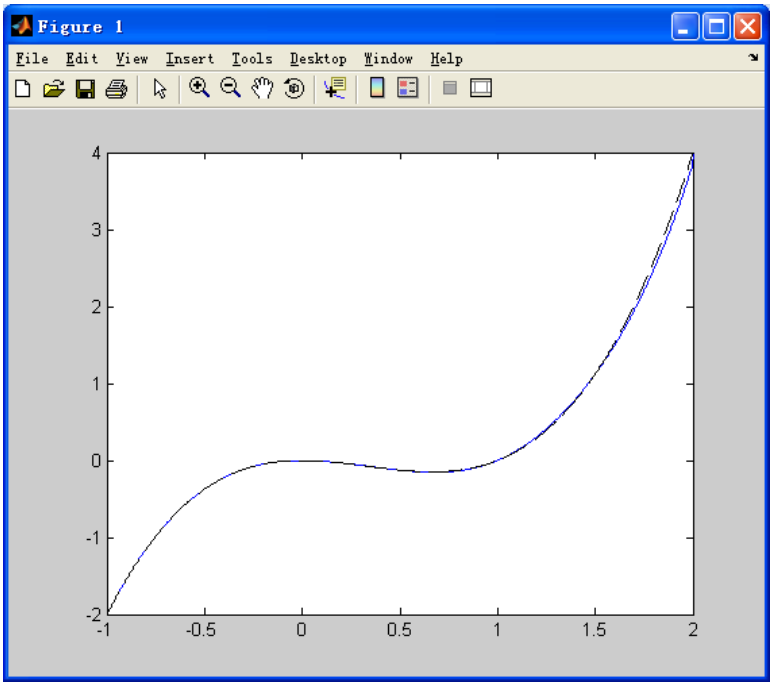
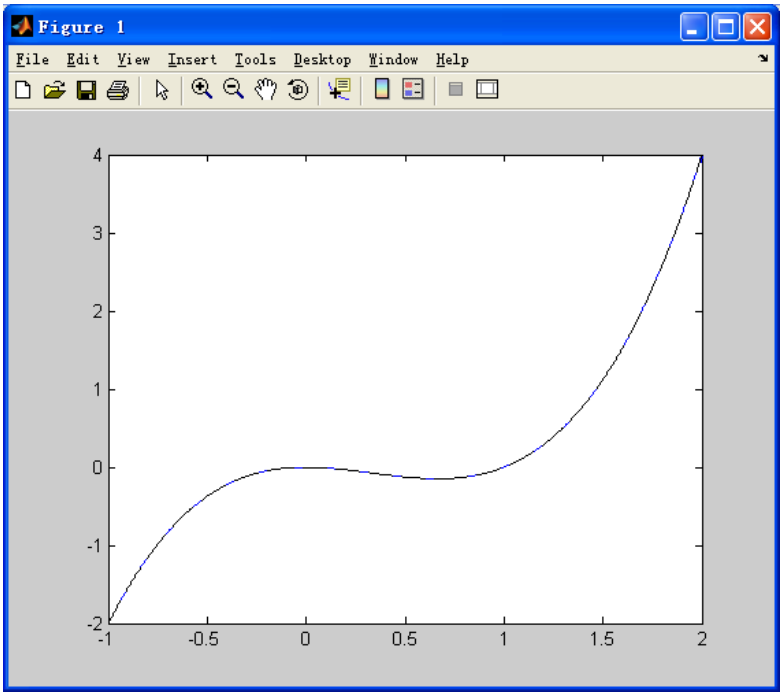


Figure 2
File Edit View Insert Tools Desktop Window Help
4
3
2
1
0
-1
-2
-1
-0.5
0
0.5
1
1.5
2

For B



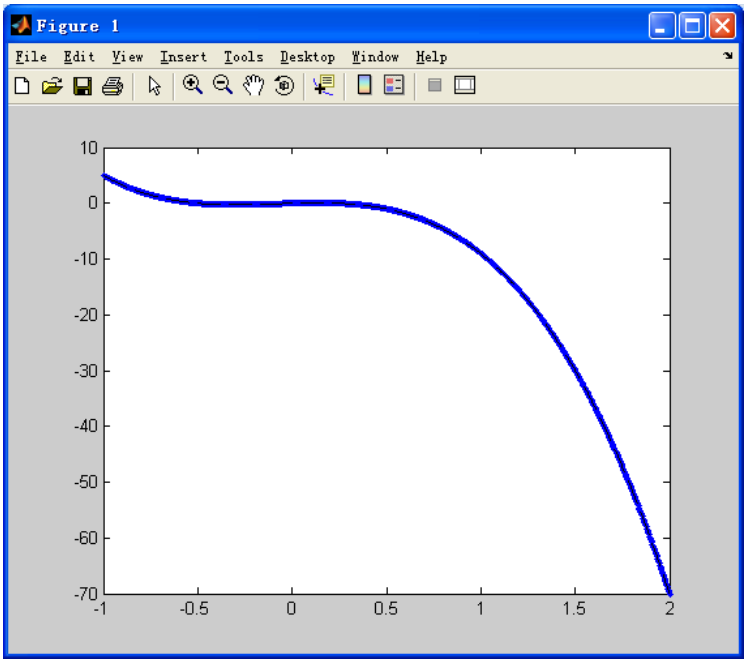
N=6

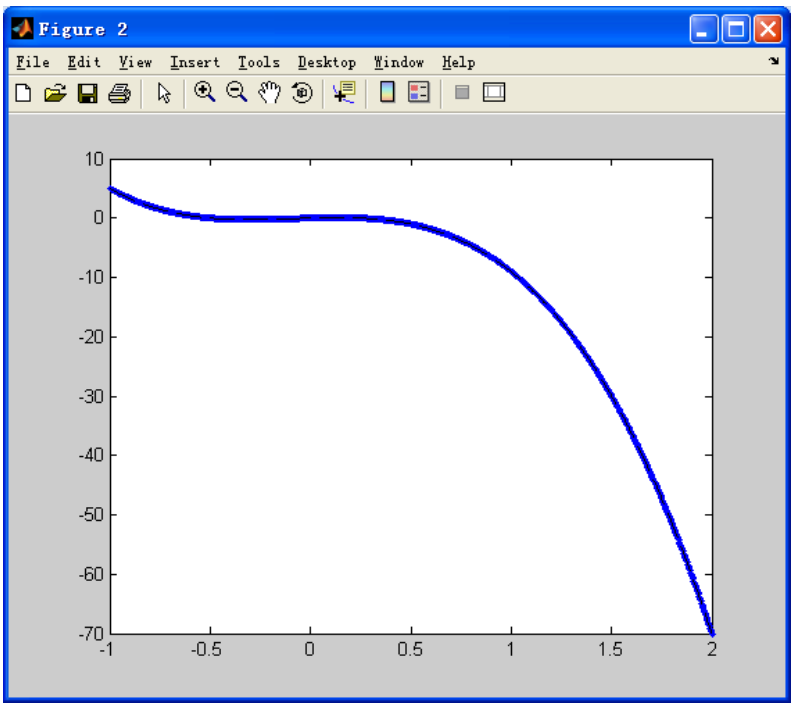


N=600

3.y=-8*x^3-2*x^2+x;

y=-8*x^3-2*x^2+x;	flag=1	flag=2	
Firstd	1.42E-14	1.42E-14	
second	1.42E-14	1.42E-14	
B	n=6	n=600	n=1000
	8.3503	0.0161	5.04E-05





For B

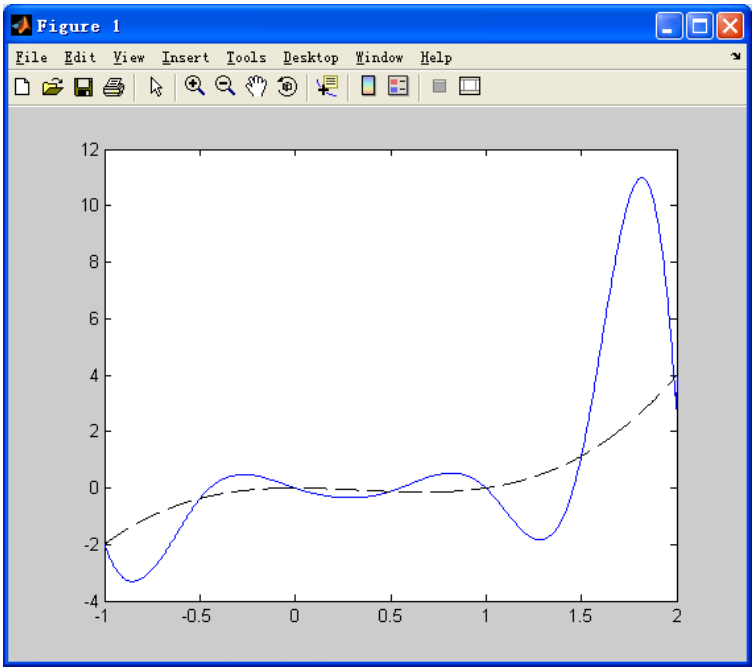


Figure 1
File Edit View Insert Tools Desktop Window Help
12
10
8
6
4
2
0
-2
-4
-1
-0.5
0
0.5
1
1.5
2

N=6

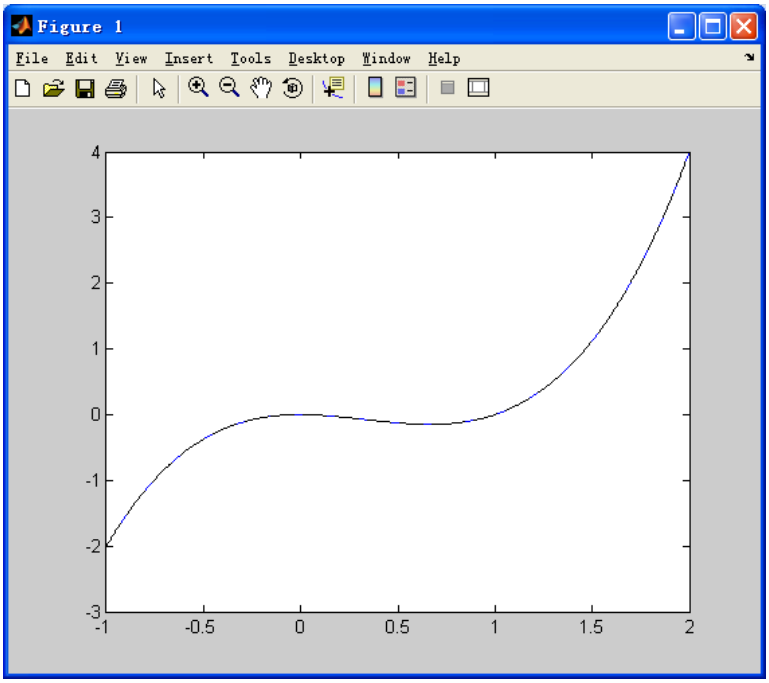
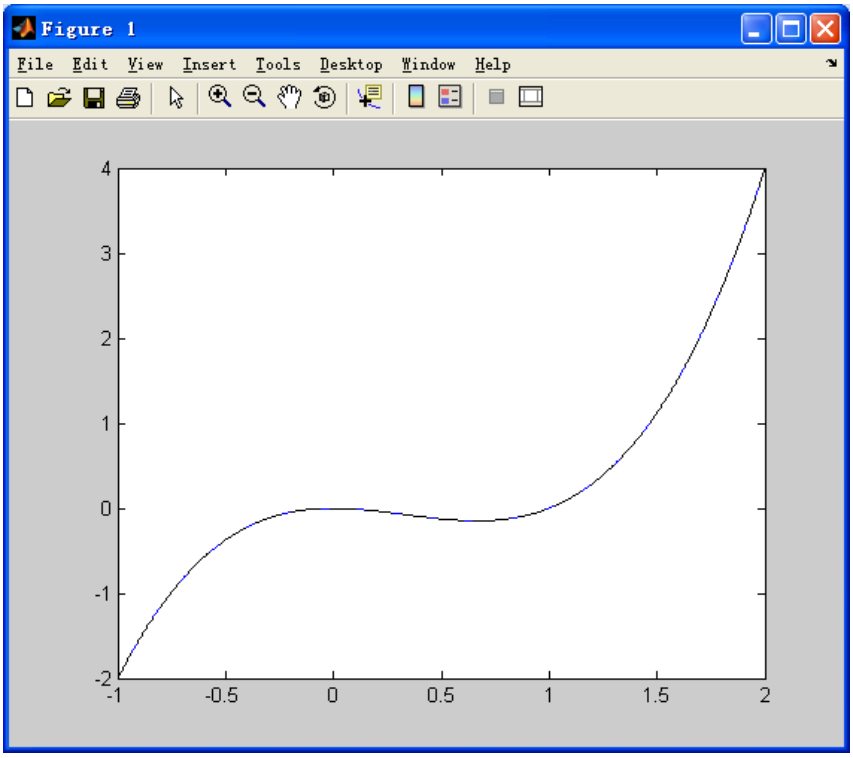


Figure 1
File Edit View Insert Tools Desktop Window Help
4
3
2
1
0
-1
-2
-3
-1
-0.5
0
0.5
1
1.5
2

N=600




```
N=1000
```

```
4.y=beta/(1+alph*x^2);
```

For $f(x) = \frac{\beta}{1+\alpha x^2}$

Change alpha and beta.

Firstd and second

```
alph=1:9:19
beta=1:9:19
```

i.e. alph=1,10,19

beta=1,10,19

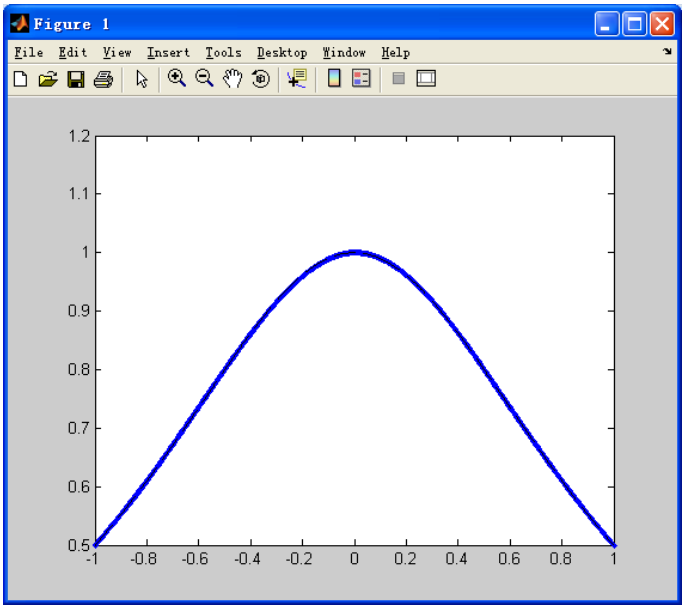
test1([-1,-0.8,-0.5,0,0.2,0.6,1])

choose nodes as [-1,-0.8,-0.5,0,0.2,0.6,1]

	beta=1		beta=10		beta=19		
alph=1	0.0022	0.0022	0.0220	0.0222	0.0418	0.0422	flag=1
	0.0022	0.0022	0.0222	0.0220	0.0422	0.0418	flag=2
alph=10	0.1383	0.1389	1.3830	1.3886	2.6277	2.6384	flag=1
	0.1389	0.1383	1.3886	1.3830	2.6384	2.6277	flag=2
alph=19	0.2846	0.2853	2.8458	2.8532	5.4071	5.4210	flag=1
	0.2853	0.2846	2.8532	2.8458	5.4210	5.4071	flag=2
	Firstd	secondd	Firstd	secondd	Firstd	secondd	

From the table, we see
If we fix beta, alpha increases, error increases.
If we fix alpha, beta increases, error increases.

Boundary conditions have not much effect on error.



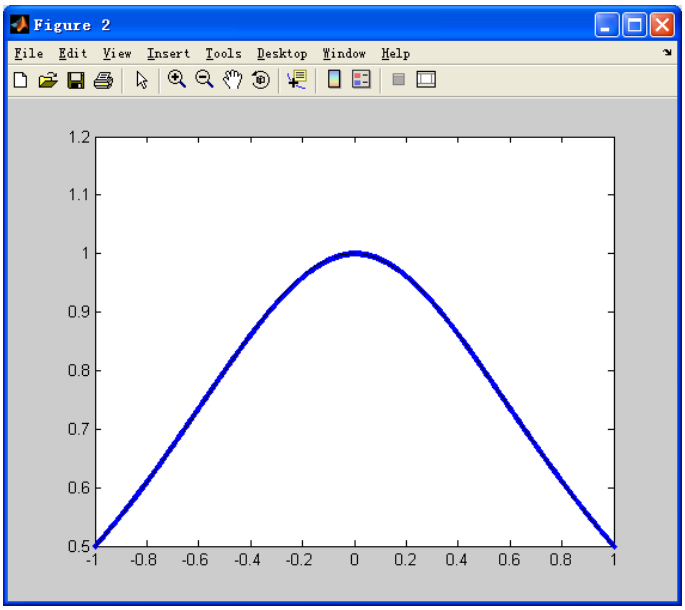


Figure 2
File Edit View Insert Tools Desktop Window Help
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

Alph=1 beta=1

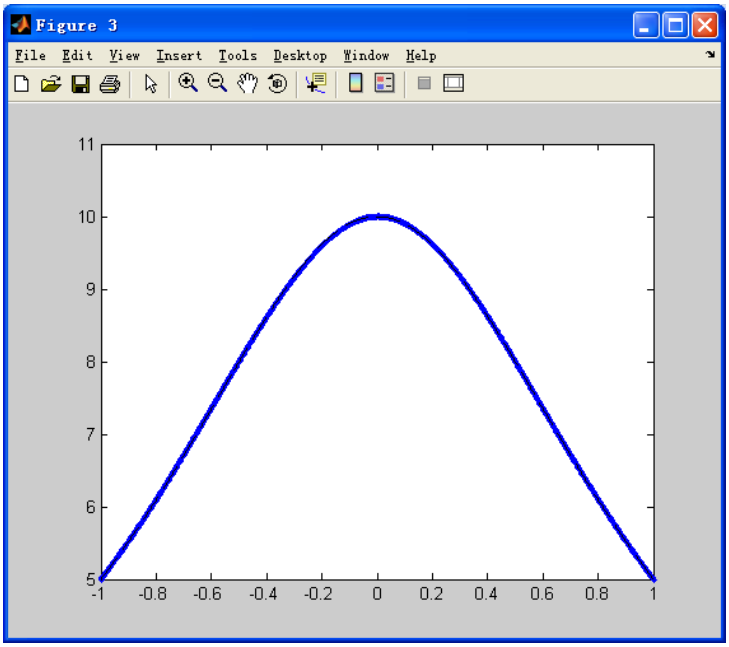
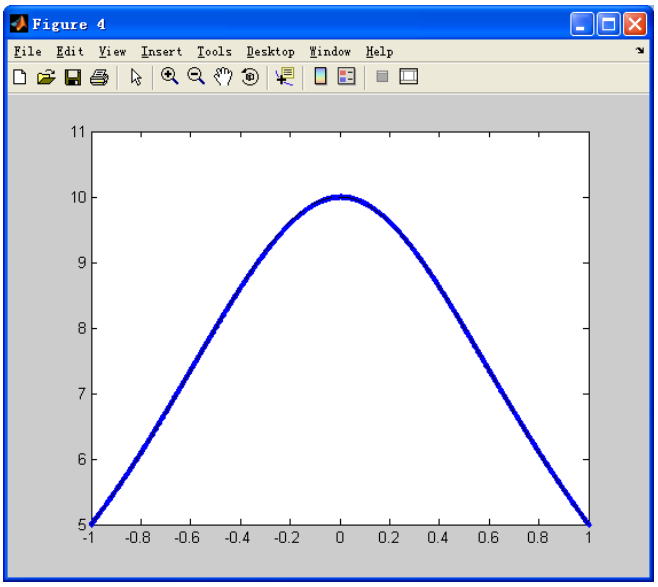
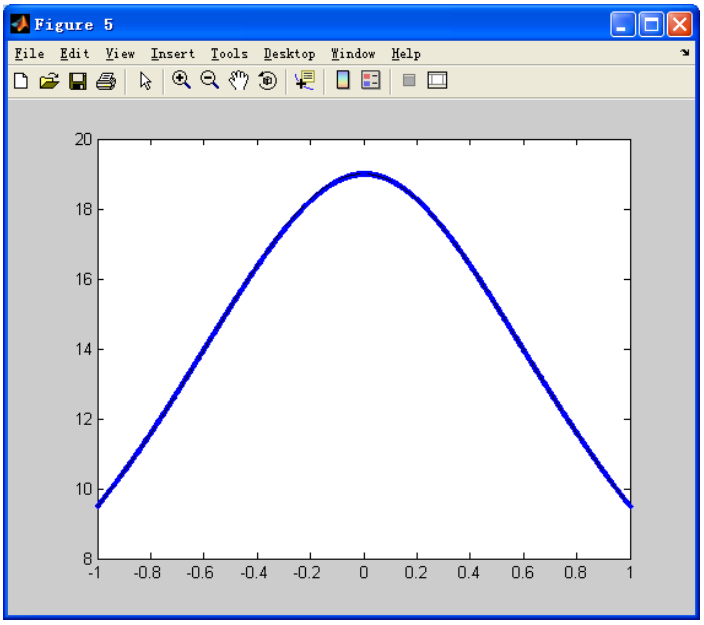
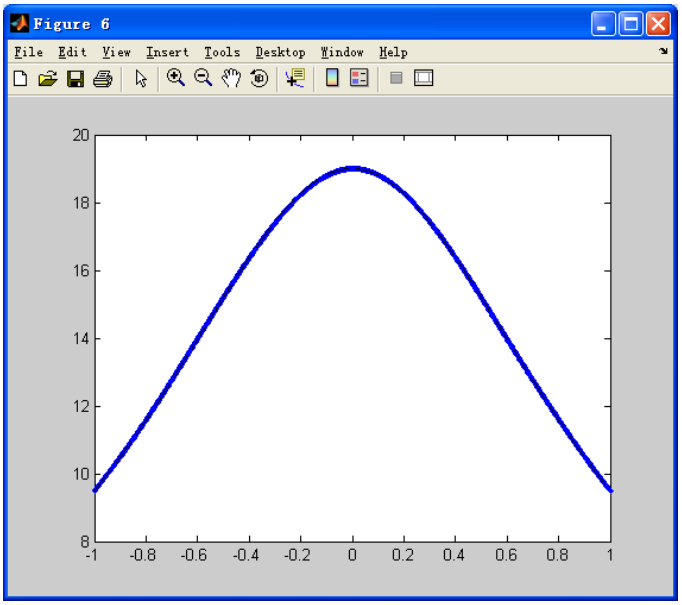


Figure 3
File Edit View Insert Tools Desktop Window Help
11
10
9
8
7
6
5
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

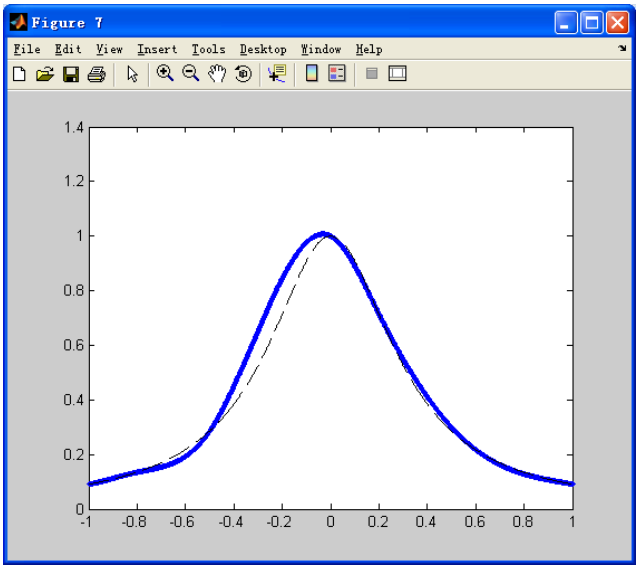


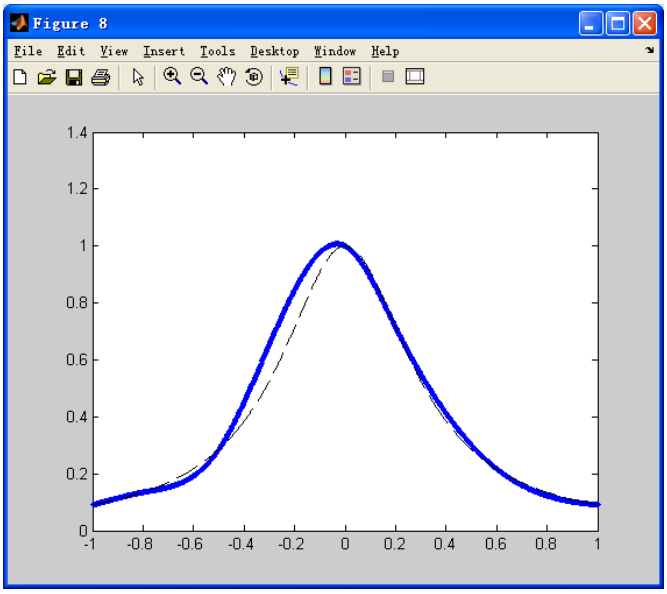
Alph=1 beta=10



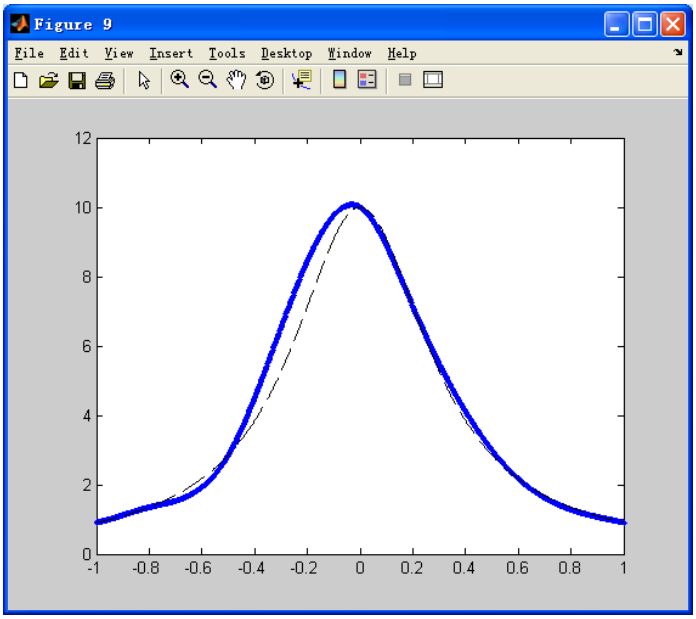


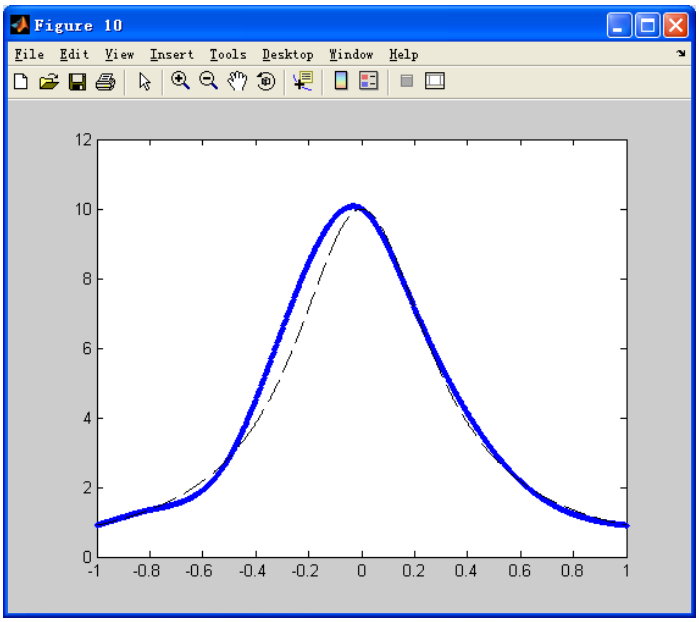
Alph=1 beta=19





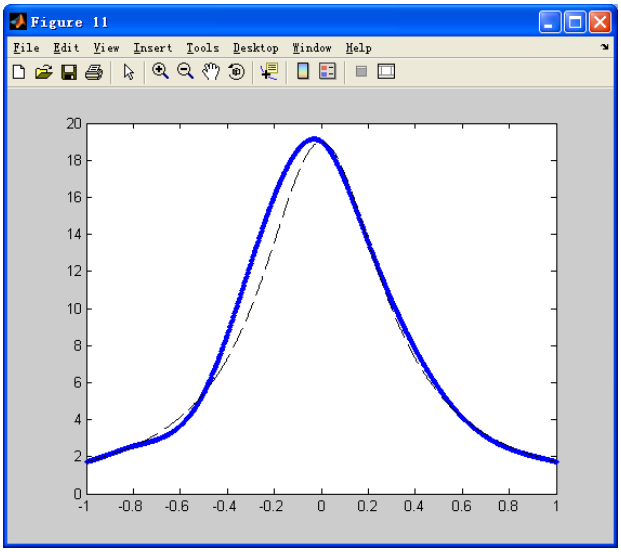
Alph=10 beta=1

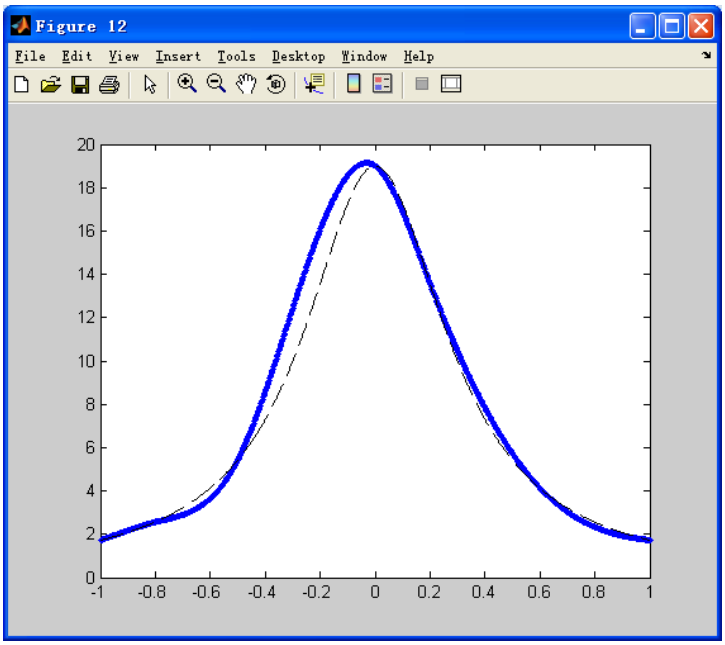




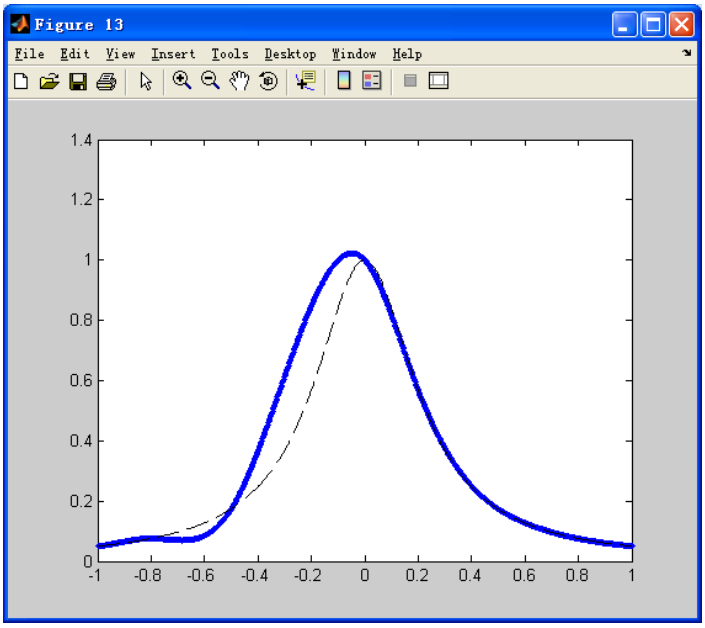
Alph=10 beta=10

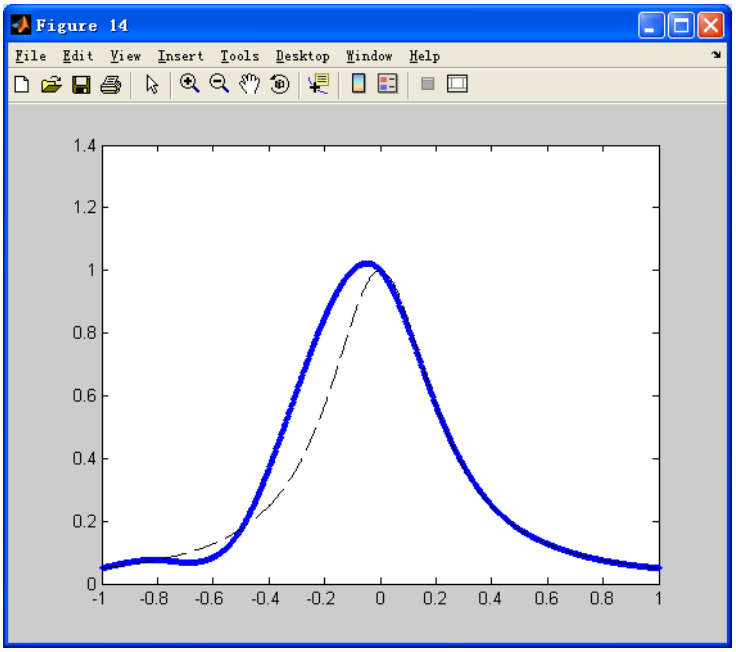
v



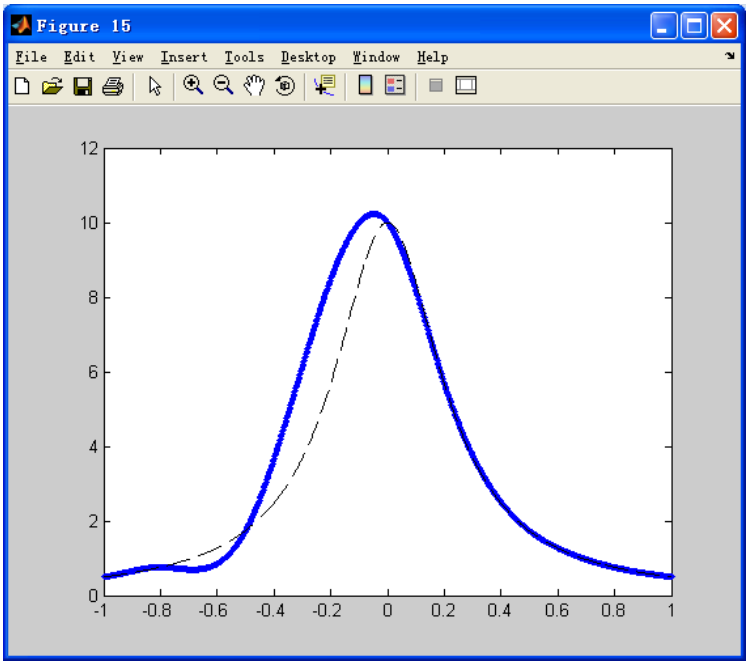


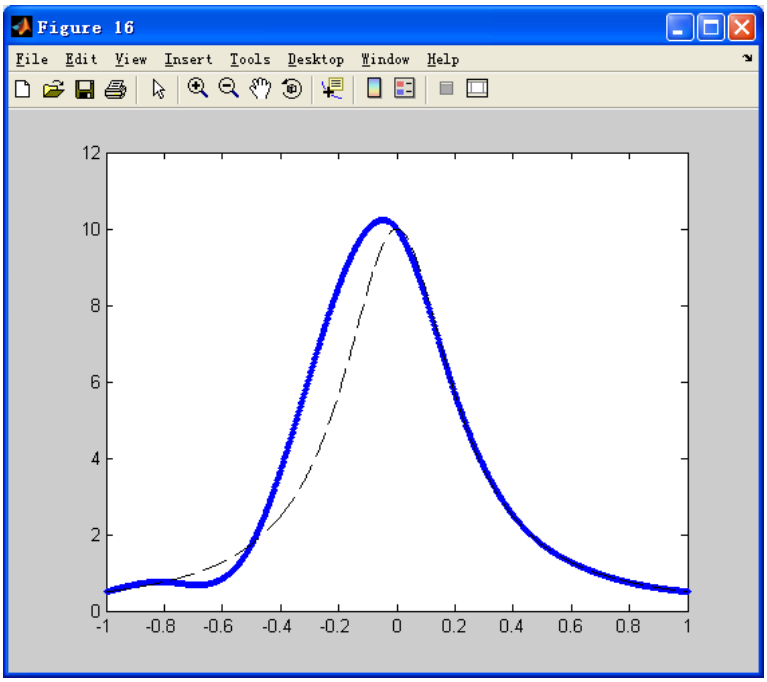
Alph=10 beta=19



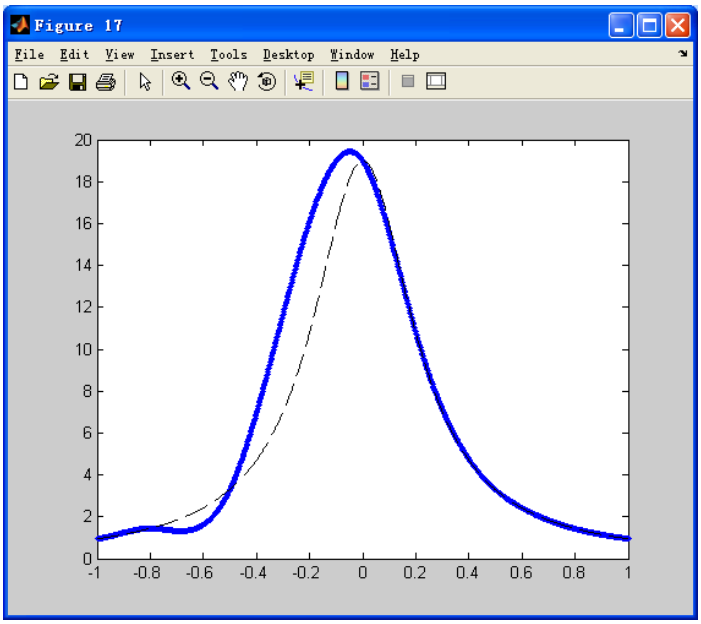


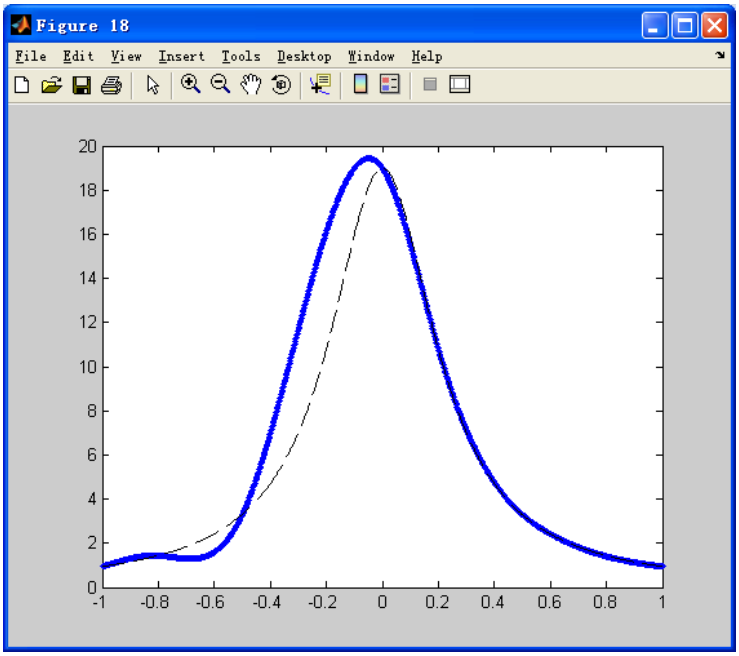
Alph=19 beta=1





Alph=19 beta=10





Alph=19 beta=19

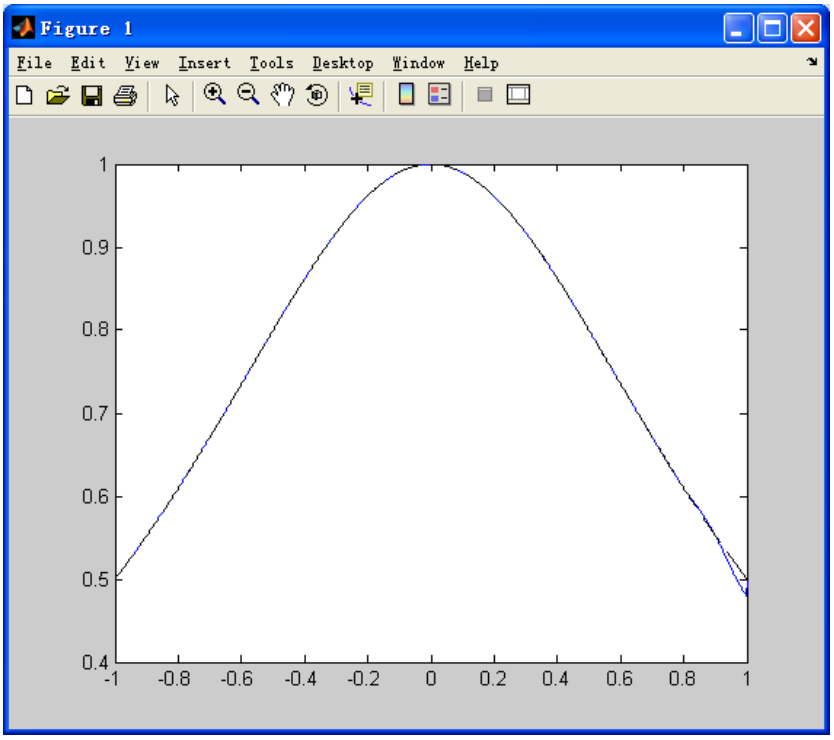
For B-spline

B(n=6)	beta=1	beta=10	beta=19
alph=1	0.0203157	0.2031569	0.3859982
alph=10	0.0283012	0.2830124	0.5377236
alph=19	0.0900793	0.900793	1.7115067
n=20	beta=1	beta=10	beta=19
alph=1	0.0206732	0.2067319	0.3927906
alph=10	0.0037354	0.0373538	0.0709722
alph=19	0.002234	0.0223402	0.0424464

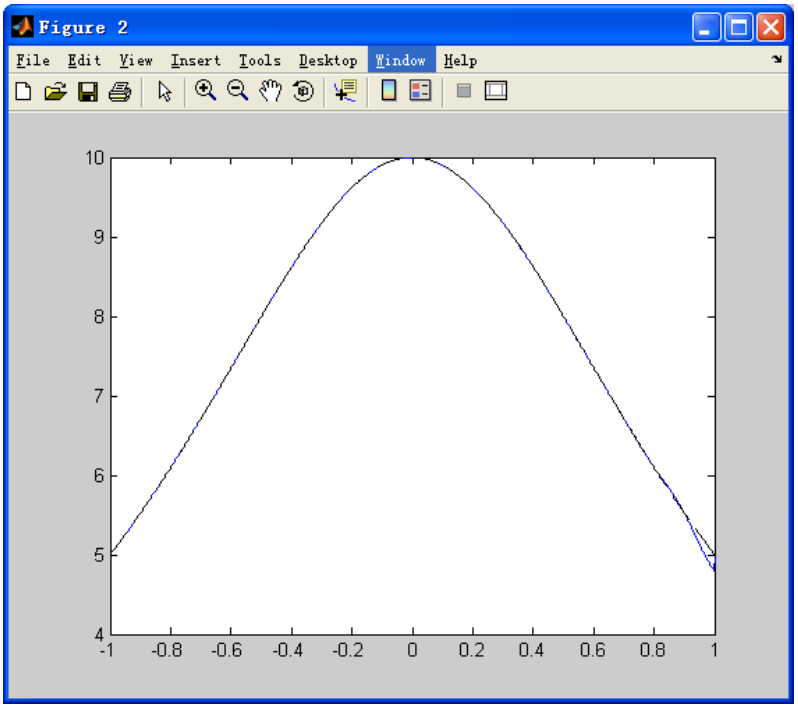
B performs much better than the other two methods when given the same number of subintervals.

As the B spline will better performs when we increase the number of intervals, I tried 20 intervals and outcome is good.

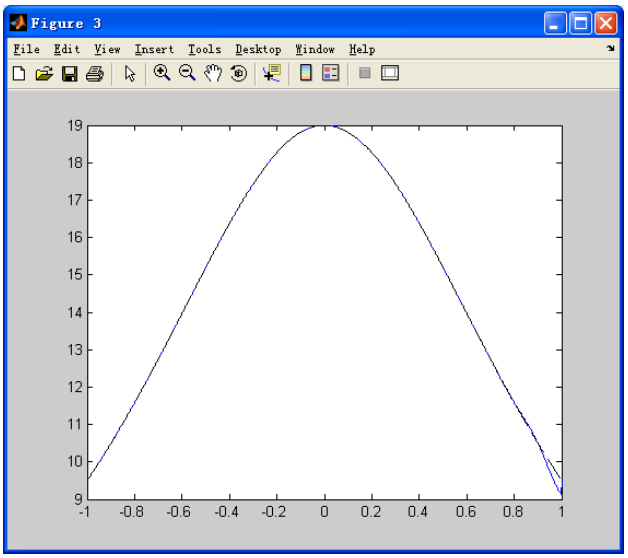
Graphs are for n=20



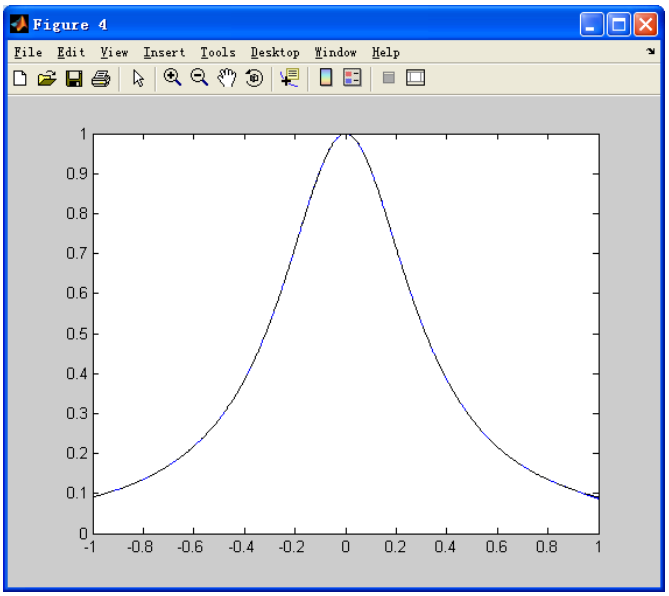
Alph=1 beta=1



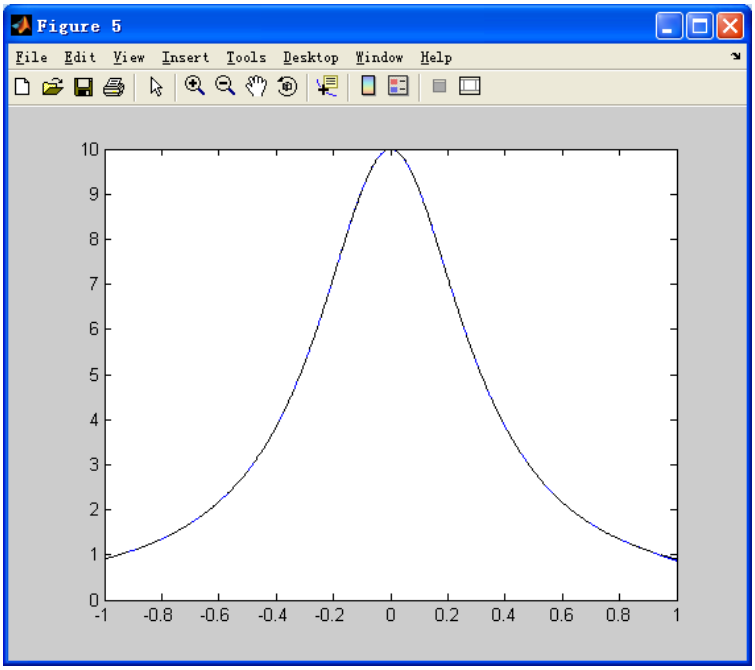
Alph=1 beta=10



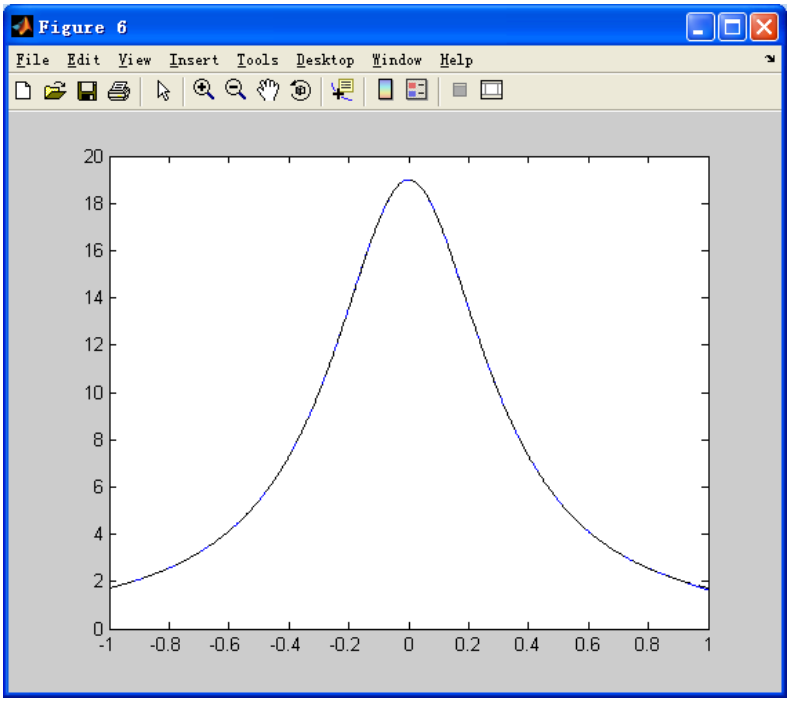
Alph=1 beta=19



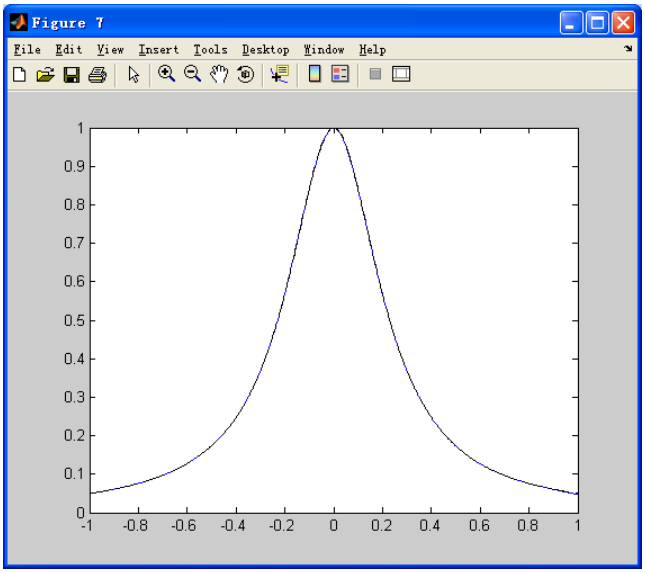
Alph=10 beta=1



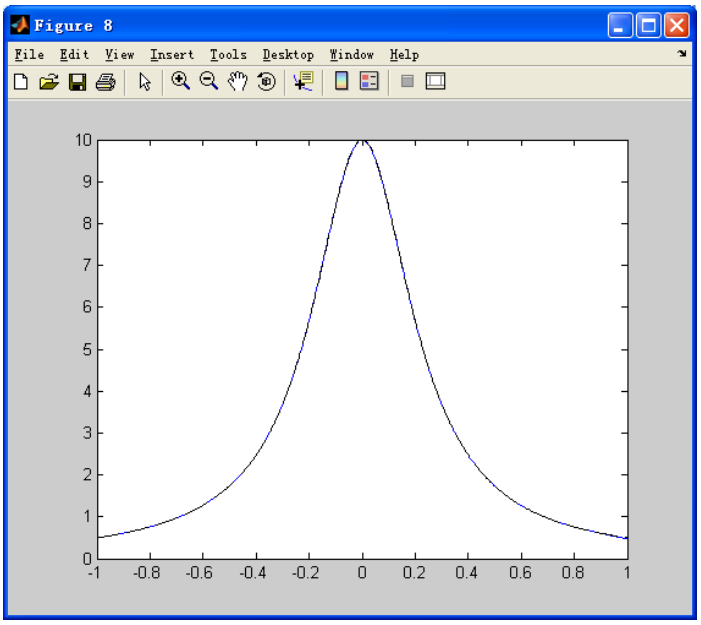
Alph=10 beta=10



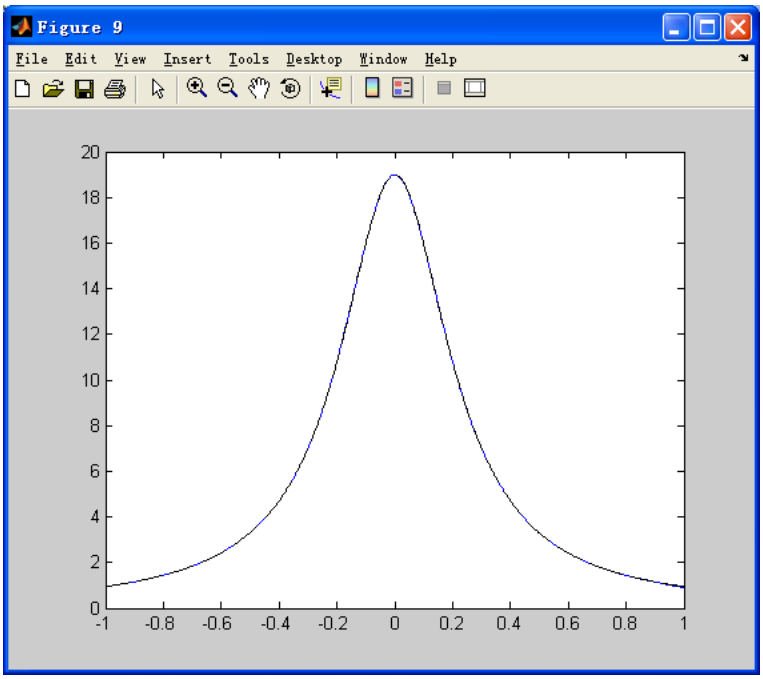
Alph=10 beta=19



Alph=19 beta=1



Alph=19 beta=10



Alph=19 beta=19

Conclusion:

(1) For the cubic polynomials, the interpolating cubic splines which derived from the Ts'=d(Firstd) and Ts''=d(secondd) performs perfectly well no matter what boundary conditions you apply.

For B-splines, it interpolates not as much well as the other two algorithms under the same nodes. To speak more clearly, if given the same number nodes to interpolate a cubic polynomials, Firstd and secondd work well than B.

However, if we increase the n, i.e. the number of the subintervals, the quality of interpolating will improve for B this algorithm.

But the improvement is slow, I need to increase n to 1000 to get a accuracy to 0.01.

(2)

For the other test function $f(x) = \frac{\beta}{1+\alpha x^2}$, the interpolating cubic splines which derived from the Ts'=d(Firstd) and Ts''=d(secondd) perform not as well as those for cubic polynomials. With the same nodes, Firstd and secondd both performs nicely when interpolating a cubic polynomial, but it results in a big error when interpolating the second family of functions $f(x) = \frac{\beta}{1+\alpha x^2}$.

In addition to that, B-splines interpolate much better than the other two algorithms under the same nodes. To speak more clearly, if given the same number of nodes to interpolate $f(x) = \frac{\beta}{1+\alpha x^2}$, B works well than Firstd and secondd.

(3)

For $f(x) = \frac{\beta}{1+\alpha x^2}$, the smaller the alph and beta we choose, the better quality of interpolating.

And for firstd and secondd, the right part i.e.in the interval (0,1),the cubic splines works better than the left part (-1,0).

(4) So there is no conclusion that which algorithm absolutely excels than the other. It depends on the kind of interpolating function, and the nodes you choose.

4. copy of code

Firstd.m

```
function [y] = Firstd(I,x,flag,alph,beta)
             %splines derived from Ts'=d.
             %evaluate the spline at x.output the value y.
             %I is a vector which saves the given points.ie. I gives out
a partition of
             %the interval; flag is the for the boundary conditions you
choose
[m,n]=size(I);

 %%compute functional value at given points.
 for i=1:n
  fun(i)=f1(I(i),alph,beta);
 end

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                  start        compute
s'   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 for i=1:n-1
    h(i)=I(i+1)-I(i);
 end
 for i=1:n-2
    miu(i)=h(i)/(h(i)+h(i+1));
    lamta(i)=h(i+1)/(h(i)+h(i+1));
 end                    %%%  compute the parameters miu and lamta. %%

 %% now start to compute the divided difference which we need  %%%%%%
   A=zeros(3,n);
   for j=1:n
      A(1,j)=fun(j);
   end

   for i=2:3
      for j=i:n
```

```
          A(i,j)=(A(i-1,j)-A(i-1,j-1))/(I(j)-I(j-i+1));
       end
    end
 for i=1:n-2
     Divided(i)=A(3,i+2);
 end

 %%%%%%%%start solve tridiagonal matrix.d is the righthand side vector
for
 %%%%%%%%this matrix.

 if flag==1      %%%%give the first dirivative
derix0=f11(I(1),alph,beta);
derixn=f11(I(n),alph,beta);%%%%%%%%save the boundary condition


for i=1:n-2
     d(i)=lamta(i)*A(2,i+1)+miu(i)*A(2,i+2);
     d(i)=3*d(i);
end

 d(1)=d(1)-lamta(1)*derix0;
 d(n-2)=d(n-2)-miu(n-2)*derixn;


 %%%%%%%%%%%% Gauss elimation  %%%%%%%%%%%%%%%%%%%%%%%%%
 for i=1:n-2
     Diag(i)=2;
 end
 for i=1:n-3
     d(i+1)=d(i+1)-d(i)*lamta(i+1)/Diag(i);
     Diag(i+1)=Diag(i+1)-miu(i)*lamta(i+1)/Diag(i);
 end


 s1(n-1)=d(n-2)/Diag(n-2);
 for j=1:n-3
     s1(n-1-j)=(d(n-2-j)-miu(n-2-j)*s1(n-j))/Diag(n-2-j);
 end
 s1(1)=derix0;
 s1(n)=derixn;
 end

 if flag==2
```

```
   derix0=f12(I(1),alph,beta);
   derixn=f12(I(n),alph,beta);%%%%%%%save          the          boundary
condition  ,secondary derivatives

     for i=2:n-1
        d(i)=3*(lamta(i-1)*A(2,i)+miu(i-1)*A(2,i+1));
     end

     d(1)=-h(1)*derix0/2+3*A(2,2);
     d(n)=h(n-1)*derixn/2+3*A(2,n);


     %%%%%%  Gauss elimination  %%%%%%%%%%%%%%%
      for i=1:n
      Diag(i)=2;
      end
     d(2)=d(2)-d(1)*lamta(1)/2;
     Diag(2)=Diag(2)-lamta(1)/2;

     for i=2:n-2
        d(i+1)=d(i+1)-d(i)*lamta(i)/Diag(i);
        Diag(i+1)=Diag(i+1)-miu(i-1)*lamta(i)/Diag(i);
     end

     d(n)=d(n)-d(n-1)/Diag(n-1);
     Diag(n)=Diag(n)-miu(n-2)/Diag(n-1);

    s1(n)=d(n)/Diag(n);
     for j=1:n-2
        s1(n-j)=(d(n-j)-miu(n-1-j)*s1(n-j+1))/Diag(n-j);
     end
     s1(1)=(d(1)-s1(2))/Diag(1);
 end

     %%%%%%%%%%%%%%%%%  Now start to evaluate interpolating functional
value at x %%%%%%%%%%%%%%%%%%%%%

     a=1; b=n;
     t=floor((a+b)/2);
     while (b-a)~=1
     if x==I(t)
        y=fun(t);
       break;
     elseif x>I(t)
```

```
            a=t;
    elseif x<I(t)
            b=t;
            end
    t=floor((a+b)/2);

        end

    %%%%%%%%% x lies in [I(a),I(b)] i=a %%%%%%%%%%%%%%%%%%
    if b-a==1
    l1=I(b)-x;
    l2=x-I(a);

y=([fun(a),fun(b)]*[l1^2;l2^2])/(h(a)^2)+([fun(a),fun(b)]*[l1^2*l2;l2
^2*l1])*2/(h(a)^3)+([s1(a),-s1(b)]*[l1^2*l2;l2^2*l1])/(h(a)^2);
    end
```

secondd.m

```
function [y] = secondd(I,x,flag,alph,beta)
             %splines derived from Ts"=d.
             %evaluate the spline at x.output the value y.
             %I is a vector which saves the given points.ie. I gives out
a partition of
             %the interval; flag is the for the boundary conditions you
choose
[m,n]=size(I);

 %%compute functional value at given points.
 for i=1:n
  fun(i)=f1(I(i),alph,beta);
 end

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%             start         compute
s"   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 for i=1:n-1
    h(i)=I(i+1)-I(i);
 end
 for i=1:n-2
```

```
    miu(i)=h(i)/(h(i)+h(i+1));
    lamta(i)=h(i+1)/(h(i)+h(i+1));
end    %%%  compute the parameters miu and lamta. %%

%% now start to compute the divided difference which we need  %%%%%%
   A=zeros(3,n);
   for j=1:n
      A(1,j)=fun(j);
   end

   for i=2:3
      for j=i:n
         A(i,j)=(A(i-1,j)-A(i-1,j-1))/(I(j)-I(j-i+1));
      end
   end
for i=1:n-2
    Divided(i)=A(3,i+2);
end

%%%%%%%%start solve tridiagonal matrix.d is the righthand side vector
for
%%%%%%%%this matrix.

if flag==1
derix0=f12(I(1),alph,beta);
derixn=f12(I(n),alph,beta);%%%%%%%%save the boundary condition give the
secondary derivatives

for i=1:n-2
    d(i)=6*Divided(i);
end
d(1)=d(1)-miu(1)*derix0;
d(n-2)=d(n-2)-lamta(n-2)*derixn;


%%%%%%%%%%%% Gauss elimation  %%%%%%%%%%%%%%%%%%%%%%%%
for i=1:n-2
    Diag(i)=2;
end
for i=1:n-3
    d(i+1)=d(i+1)-d(i)*miu(i+1)/Diag(i);
    Diag(i+1)=Diag(i+1)-lamta(i)*miu(i+1)/Diag(i);
end
```

```
s2(n-1)=d(n-2)/Diag(n-2);
for j=1:n-3
    s2(n-1-j)=(d(n-2-j)-lamta(n-2-j)*s2(n-j))/Diag(n-2-j);
end
s2(1)=derix0;
s2(n)=derixn;
end

if flag==2
  derix0=f11(I(1),alph,beta);
  derixn=f11(I(n),alph,beta);%%%%%%%save the boundary condition. give
the first derivatives

    for i=2:n-1
        d(i)=6*A(3,i+1);
    end
    d(1)=-6*(derix0-A(2,2))/h(1);
    d(n)=6*(derixn-A(2,n))/h(n-1);


    %%%%%%  Gauss elimination  %%%%%%%%%%%%%%%
   for i=1:n
    Diag(i)=2;
end
    d(2)=d(2)-d(1)*miu(1)/2;
    Diag(2)=Diag(2)-miu(1)/2;

    for i=2:n-2
        d(i+1)=d(i+1)-d(i)*miu(i)/Diag(i);
        Diag(i+1)=Diag(i+1)-lamta(i-1)*miu(i)/Diag(i);
    end

    d(n)=d(n)-d(n-1)/Diag(n-1);
    Diag(n)=Diag(n)-lamta(n-2)/Diag(n-1);

   s2(n)=d(n)/Diag(n);
    for j=1:n-2
        s2(n-j)=(d(n-j)-lamta(n-1-j)*s2(n-j+1))/Diag(n-j);
    end
    s2(1)=(d(1)-s2(2))/Diag(1);
end

    %%%%%%%%%%%%%%%%%  Now start to evaluate interpolating functional
```

```
value at x %%%%%%%%%%%%%%%%%%%%%

    a=1; b=n;
    t=floor((a+b)/2);
    while (b-a)~=1
    if x==I(t)
       y=fun(t);
      break;
    elseif x>I(t)
           a=t;
    elseif x<I(t)
           b=t;
           end
    t=floor((a+b)/2);

    end


    %%%%%%%%% x lies in [I(a),I(b)] i=a %%%%%%%%%%%%%%%%%%
    if b-a==1
    l1=I(b)-x;
    l2=x-I(a);

y=[1/(6*h(a)),h(a)/6,-(h(a)^2)/6]*[l1^3,l2^3;l2,-l2;1,0]*[s2(a);s2(b)
]+A(2,b)*l2+fun(a);

    %%%y=s2(a)*l1^3/(6*h(a))+s2(b)*l2^3/(6*h(a))+(A(2,b)-h(a)*(s2(b)-
s2(a)
    %%%)/6)*l2+fun(a)-s2(a)*h(a)^2/6;%%%%

    end
```

B.m

```
function [y] = B(n,a,b,x,alph,beta)
             %B-splines.evaluate the spline at x.output the value y.
             %a,b are the endpoints of the interval. n is the number of
             %subintervals.(n+1)points
h=(b-a)/n;
```

```
derivx0=f11(a,alph,beta);
derivxn=f11(b,alph,beta);

for i=0:n
    I(i+1)=a+i*h;
end

for i=1:n+1
    fun(i)=f1(I(i),alph,beta);
end

%%%%%%%%%%%%%%  start solve the linear system.  %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%  save d  %%%%%%%%%%%%%%%%%

for i=1:n+1
    d(i)=fun(i);
end
    d(1)=d(1)+h*derivx0/3;
    d(n+1)=d(n+1)-h*derivxn/3;

%%%%%%%%%%%%%%  Gauss elimination  %%%%%%%%%%%%%%%%%%%%

 for i=1:n+1
     Diag(i)=4;
 end
 d(2)=d(2)-d(1)/4;
Diag(2)=Diag(2)-1/2;

for i=2:n-1
    d(i+1)=d(i+1)-d(i)/Diag(i);
    Diag(i+1)=Diag(i+1)-1/Diag(i);
end
d(n+1)=d(n+1)-d(n)*2/Diag(n);
Diag(n+1)=Diag(n+1)-1/Diag(n);


alpha(n+1)=d(n+1)/Diag(n+1);
for j=1:n-1
    alpha(n+1-j)=(d(n+1-j)-alpha(n+2-j))/Diag(n+1-j);
end
alpha(1)=(d(1)-2*alpha(2))/Diag(1);

alpha(n+2)=h*derivxn/3+alpha(n);
alphaspecial=alpha(2)-h*derivx0/3;
```

```
%%%%%%%%%%%%%% start to evaluate the value at x %%%%%%%%%%%%%%%%%%%%%%%%
if (x-a)/h-floor((x-a)/h)==0
    y=f1(x,alph,beta);
else
     i0=floor((x-a)/h)+1;

     l2=(I(i0+1)-x)/h;
     l1=(x-I(i0))/h;

     if i0==1


y=[alpha(i0),alpha(i0+1)]*[1,l2,l2^2,l2^3;1,l1,l1^2,l1^3]*[1;3;3;-3]+
[alphaspecial,alpha(i0+2)]*[l2^3;l1^3];
     else


y=[alpha(i0),alpha(i0+1)]*[1,l2,l2^2,l2^3;1,l1,l1^2,l1^3]*[1;3;3;-3]+
[alpha(i0-1),alpha(i0+2)]*[l2^3;l1^3];
     end
     end
```

test1.m

```
function test1(I)
[m,n]=size(I);
xx=linspace(I(1),I(n),1000);
for alph=1:9:19
    for beta=1:9:19
for flag=1:2
    for i=1:1000
       p(i)=Firstd(I,xx(i),flag,alph,beta);
       q(i)=secondd(I,xx(i),flag,alph,beta);
       ft(i)=f1(xx(i),alph,beta);
       temp1(i)=abs(p(i)-ft(i));
       temp2(i)=abs(q(i)-ft(i));
    end
    error(1)=max(temp1);
    error(2)=max(temp2);
    figure;
```

```
    plot(xx,p,'.')
    hold on;
    plot(xx,q,'-')
     hold on;
    plot(xx,ft,'--k')
    hold on;
    error

end
    end
end
```

Test2.m

```
function test2(a,b,n)
xx=linspace(a,b,1000);
for alph=1:9:19
    for beta=1:9:19
    for i=1:1000
        p(i)=B(n,a,b,xx(i),alph,beta);
        ft(i)=f1(xx(i),alph,beta);
        temp(i)=abs(p(i)-ft(i));

    end
    error=max(temp);
    figure;
    plot(xx,p,'-')
    hold on;
    plot(xx,ft,'--k')
    hold on;
    error
end
    end
end
```

f1.m

```
function y=f1(x,alph,beta)
 %%%y=x^3;
```

```
%%%y=x^3-x^2;
%%%y=-8*x^3-2*x^2+x;
y=beta/(1+alph*x^2);
```

f11.m

```
function y=f11(x,alph,beta)
%%%y=3*x^2;
 %%%y=3*x^2-2*x;
%%%y=-24*x^2-4*x+1;
y=-2*alph*beta*x/(1+alph*x^2)^2;
```

f12.m

```
function y=f12(x,alph,beta)
%%%%y=6*x;
%%% y=6*x-2;
%%%% y=-48*x-4;
y=-2*alph*beta*(1+alph*x^2)*(1-3*alph*x^2)/(1+alph*x^2)^4;
```