Foundations of Computational Math 2
Program 1
Chenchen Zhou

1. Program description

We are given a function f and asked to derive the Newton form of the interpolating polynomial and write a routine which can evaluate the polynomial at any value x.

2. Code description

Main function

In my code, the main function is to evaluate the Newton form of the interpolating polynomial at value x.

1. Save the specific points for interpolating.

There are three ways for us to generate the interpolating points on the domain. They are uniformly spaced points, Chebyshev points of first kind and Chebyshev points of second kind.
I just save the points via a vector y.

2. Derive the divided difference table and save it

We just depend on the divided difference table to derive our Newton form.
As we know, the divided difference table has exactly n*(n+1)/2 elements. And it's naturally to save the divided difference via a matrix for the reason that it will be easier and more clearly for the implements of choosing a path later.
As with the derivation of this table, we should use the important equation that is

$$D^k f_i = f[x_i, \ldots, x_{i+k}] = \frac{D^{k-1} f_{i+1} - D^{k-1} f_i}{(x_{i+k} - x_i)}$$

So this matrix is a (n+1) by (n+1) matrix whose first row is the functional value of the interpolating points. For the rest entries, we just use the recursive equation to derive them.

3. Save the order which determines the path we use

I just specified the path as a sequence of integers listing the indices of the points in the order they should be included in the form of the Newton polynomial. And save this sequence in a vector I which the total number of integers can be less than (n+1). This is for the reason in case we need evaluate the interpolating polynomials of degree less than n.

4. Choose a path and save the divided difference we need

According to the vector I, which just records the order the points, how can we specify our path? Because choosing a specific path, we just equivalently choose the divided difference which will be used in our Newton form of the interpolating polynomial. There are n+1 divided differences (including the functional value) in total to be used when we interpolating the f via the polynomial of degree of n. So to specify the path, I use a vector which saves the (n+1) divided differences. As we just choose only one entry in one row, and we choose them from row 1 to row n+1. So we just choose the order of indices of column. Notice that the divided difference which we choose must be successive which means that the index of next point must be less than the index of the point before by one or greater than it by one, otherwise it will request the divided difference which doesn't appear on the table. What's more, the value of divided difference has nothing to do with the order of the involving points. Combine those two truth, we can draw a conclusion that for the ith divided difference, the column it locates is just the index of largest indices of the first i points.

So we have two things need to do, first is when we go along row by row, once we get a new point, we just sort all the points we've got now in ascending order and get the largest index. I just create a new vector J to help to store those indices in an ascending order. By doing this, once we get the next index, we just compare it to the smallest index and the largest index in J to see whether the difference is one. If it is true, then the path can exist in table, otherwise we will need divided difference which is not shown in the table. The second thing is once we get that largest index, we get the divided difference for that row. So I use another vector a to store the divided difference in order.

5. Evaluate the polynomial at value x

Inspired by the Horner's rule, we can also take the Newton form into a similar form which is:

$$f(x_{i0}) + (x - x_{i0})(g(x_{i0}, x_{i1}) + (x - x_{i1})(g(x_{i0}, x_{i1}, x_{i2}) + ...))$$

Here g denotes the divided difference.

So inside one pair of parenthesis, there are 3 computations, and there are n

parentheses. So the complexity of evaluating the interpolating polynomial at value x is just O(n).

Test code

1.2(a)

Verify that if f(x) is a polynomial of degree n or less then any interpolation polynomial of degree n defined by (x0, f(x0)), . . . , (xn, f(xn)) should be equal to f(x).

I take f1(x)=x^10-x^7. This is a polynomial of degree 10. So in order to verify the statement. I set m=n=12,18,25, and choose an ascending order. i.e. I=(0,1,2…18.),(0,1,2…18),(0,1,2…25)

For test, I choose 1000 linear spaced points on the interval. And evaluate the interpolating polynomial and f at those points respectively. Use the norm to represent the error and plot the both functions to see the result in another way.

I test for all the three points set.

Code is in test1.m

1.2(b)

As it wants to show that the interpolating polynomial is unique by given the specific data and no matter what path we choose, it doesn't affect the polynomial.

What's more, we also require the routine to detect path that requires the divided difference not in the table.

So I take the f same as 1.2(a). And in this case, take m=n=10, the same degree of the given polynomial. And set flag=1. Take

I1= [0;1;2;3;4;5;6;7;8;9;10], I2=[3;4;5;6;2;1;7;8;9;10;0]

For detecting the path,

Run the routine again by using

I1=[0,1,2,3,5,6,7,8,9,10,4], I2=[3;4;5;6;2;1;7;8;9;10;0]

Code is in test2.m

1.2(c)

Just verify that the code can evaluate the polynomial of degree less than n.

Consider this kind of cases, when I am writing the code, I set up another parameter m which is the degree of the polynomial less or equal than n.

So I take I=[2,3,4] which means m=2 n=10 f is same as before.

I just plot the 1000 linear spaced points to see the polynomial function.

Code is in test4.m

1.2(d)

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

Compare the performance of interpolating polynomial for different alpha and different degree using three points set.

The routine is similar to 1.2(a), just need to modify f.

So to be convenient, I just change the function f in 1.2(a). And do the test.

I take alpha={2,5,8}.

M=n={10,20,40}

Flag={1,2,3}

Code is in test3.m

3. Experiments and conclusions.

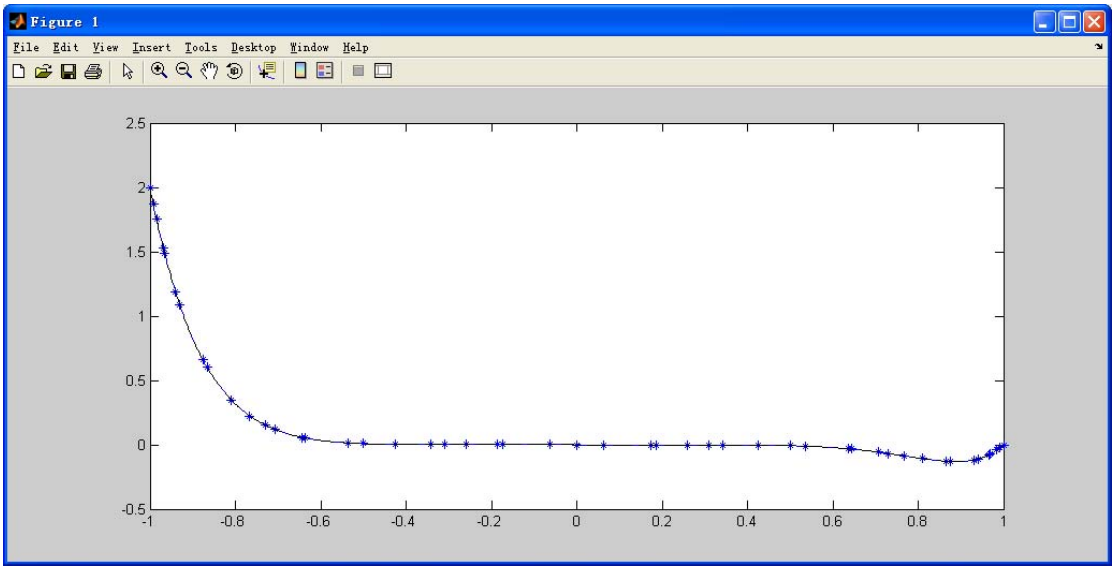
Figure 1 is for uniformly points set

Figure 2 is for 1st Chebyshev points set

Figure3 is for 2nd Chebyshev points set

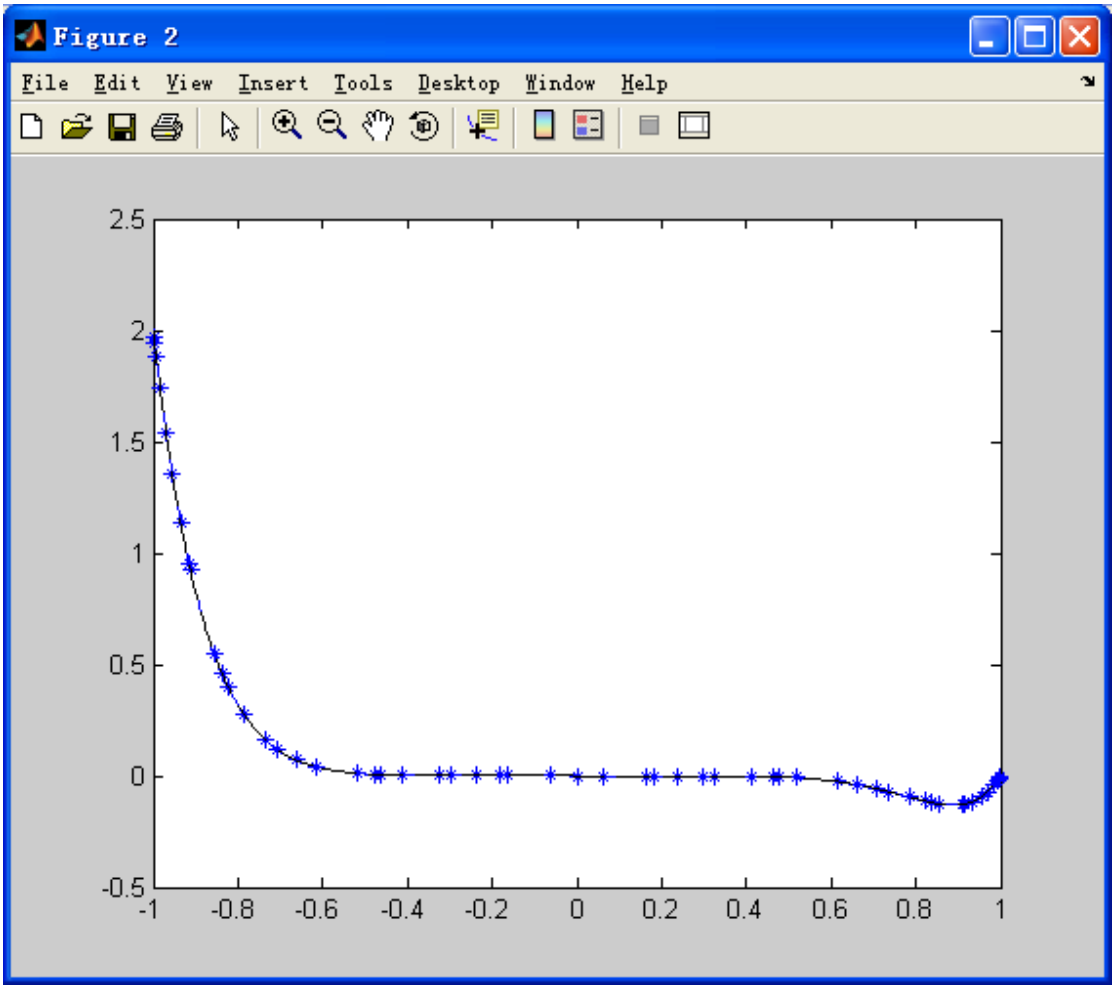
1.2(a)

	error(flag=1)	error(flag=2)	error(flag=3)
m=n=12	9.54E-14	1.01E-13	9.24E-14
m=n=18	2.97E-13	3.92E-13	5.35E-13
m=n=25	3.67E-13	9.14E-13	1.11E-12



As we can know from the error table, the error is small enough. As we choose 1000 points to compare the value of the interpolating polynomial to the value of f1. We can conclude that the polynomial are same to f1.

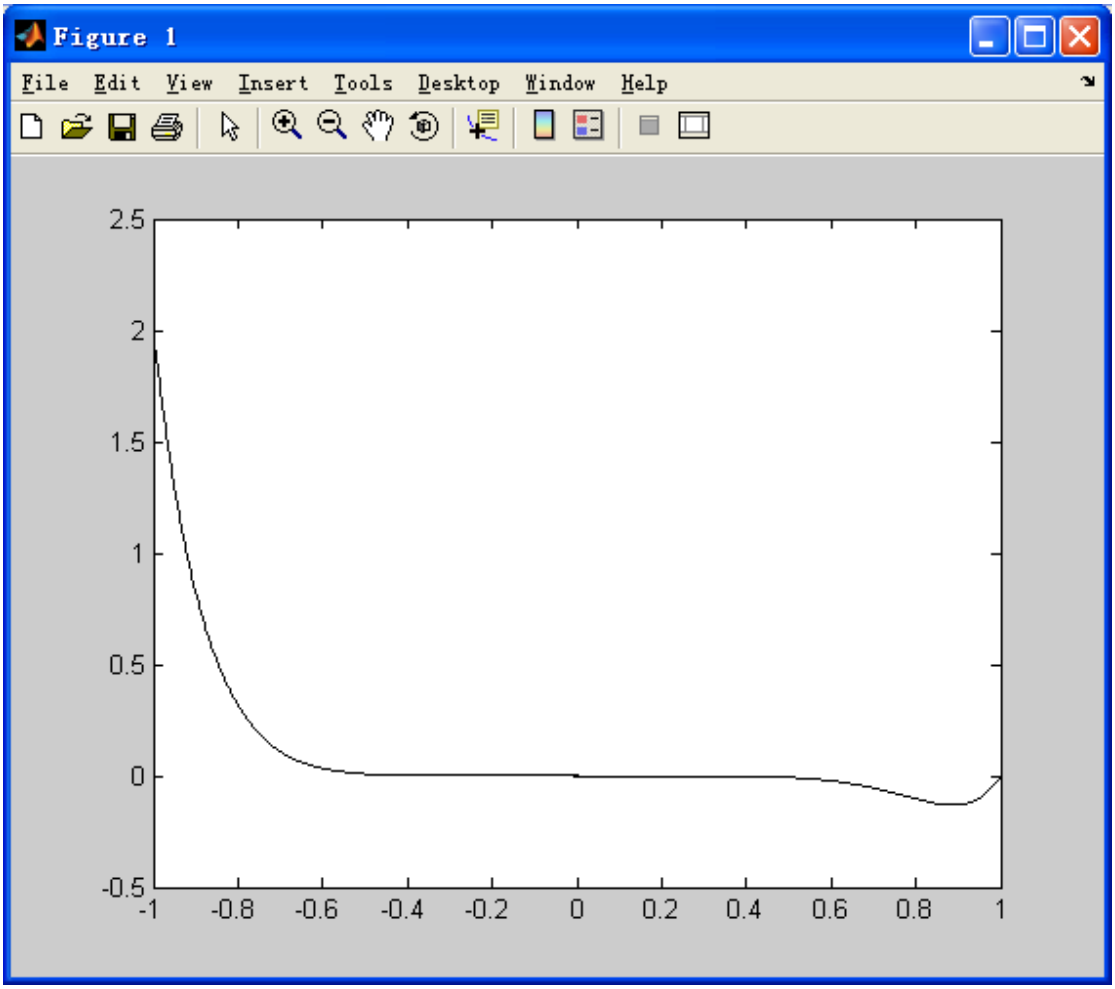
We can also say that the figure for polynomial function and f1 are exactly matched.



This figure I just hold all the figures for all of the n. As we can see, the figure just matches finely, which concludes that the interpolating polynomial are exactly same no matter what n is.

1.2(b)

test2([0;1;2;3;4;5;6;7;8;9;10],[3;4;5;6;2;1;7;8;9;10;0],10,10,1)
error = 3.5832e-014



I choose different path, and we get the same interpolating polynomial which we can see from both the error table and the figure shown above.

Actually, I just choose two paths. We can keep verify by contrasting more available paths.

For detect wrong path

```
test2([0,1,2,3,5,6,7,8,9,10,4],[3;4;5;6;2;1;7;8;9;10;0],10,10,1)

??? Error using ==> evaluate
the path is not available

Error in ==> test2 at 4
            [p1(i),y,fun]=evaluate(xx(i),I1,m,n,flag);
```

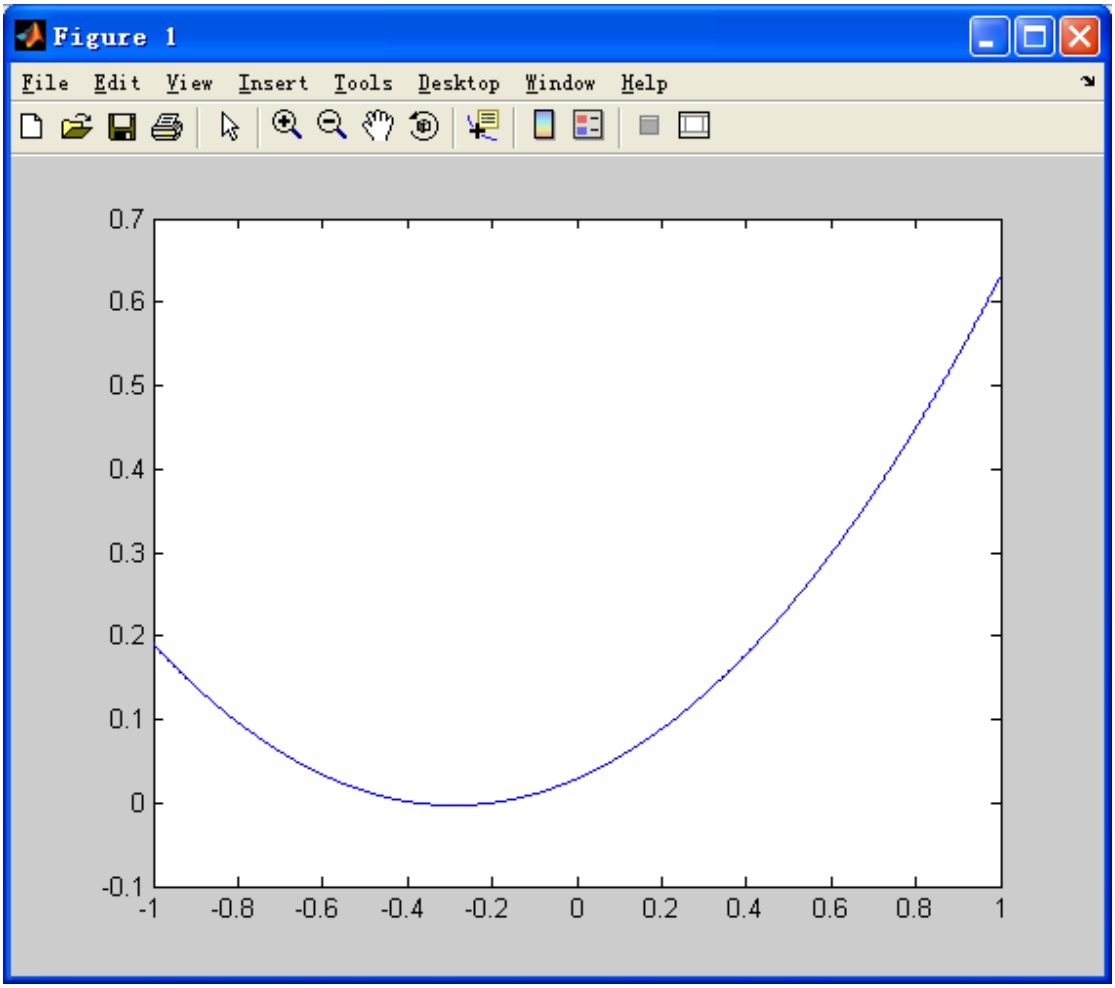
The first path I choose is not good. It will request the divided difference which is not in the

divided difference table. The code indeed detected that and show error.

1.2(c)

M=2;[2,3,4]n=10 flag=1

Value is on this curve



From the curve above, we can draw conclusion that

(1) My code indeed can evaluate the interpolating polynomial which has a less than n degree. Otherwise the matlab can't plot the figure.

(2) It is just quadratic function which we can see from the figure, it matches with m=2.

1.2(d,e)

Alpha=2

M=n=10

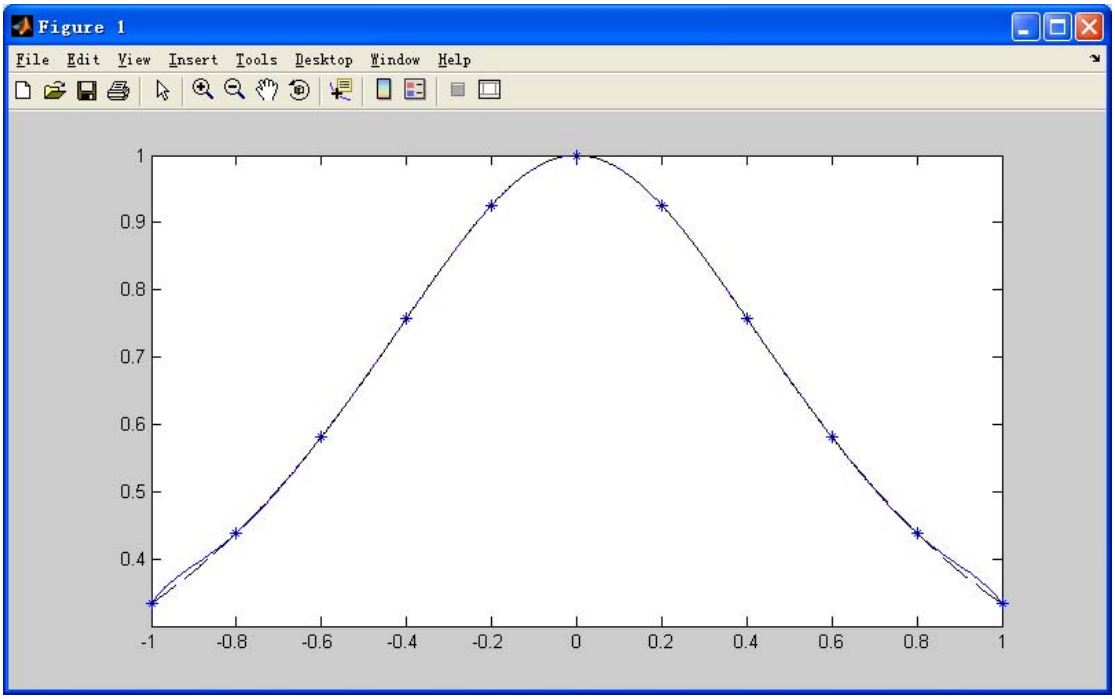


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

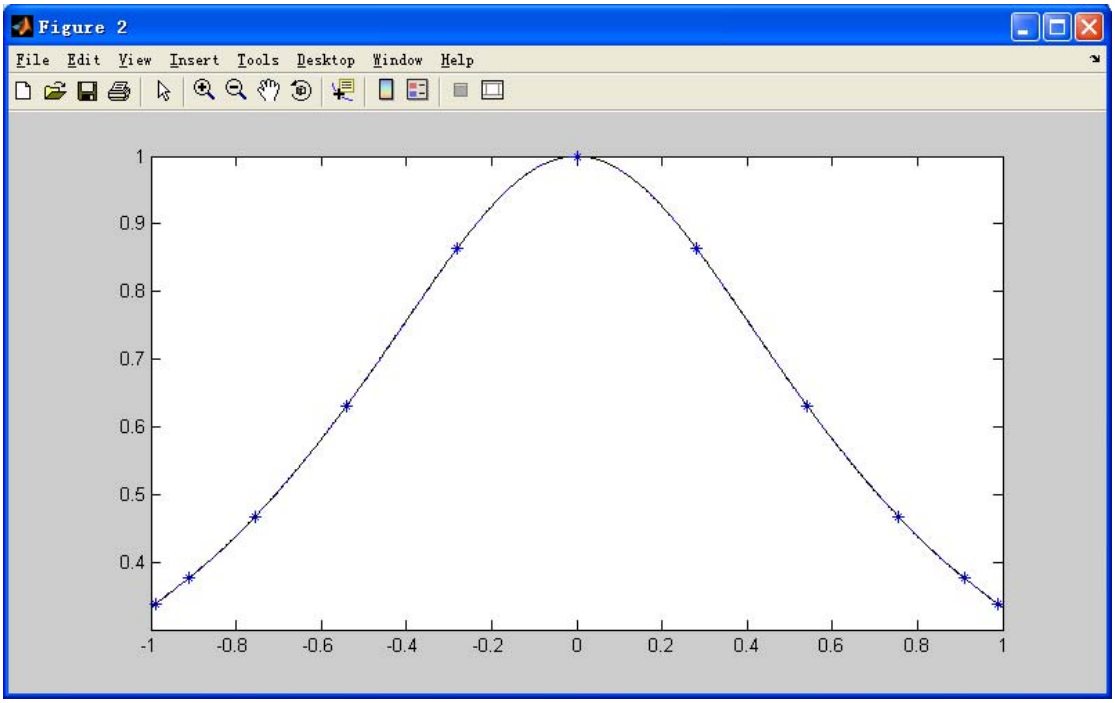


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

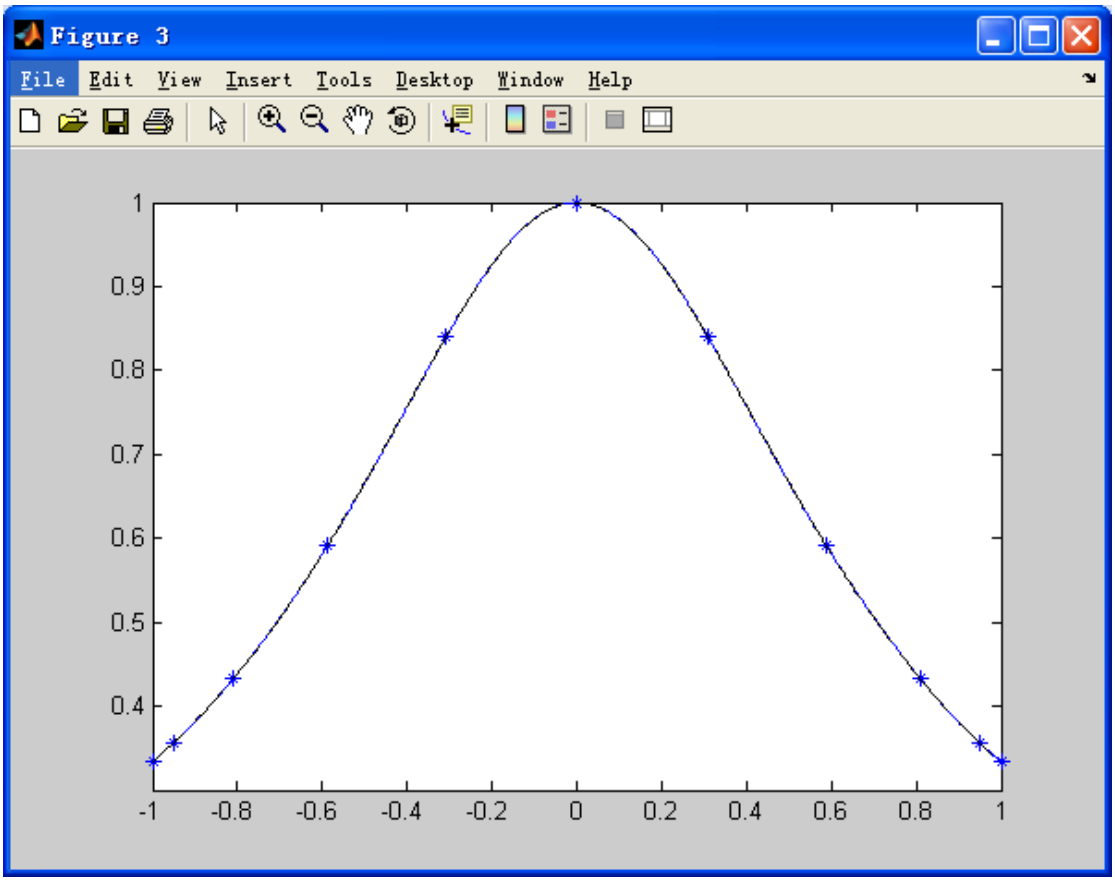


Figure 3
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

N=20

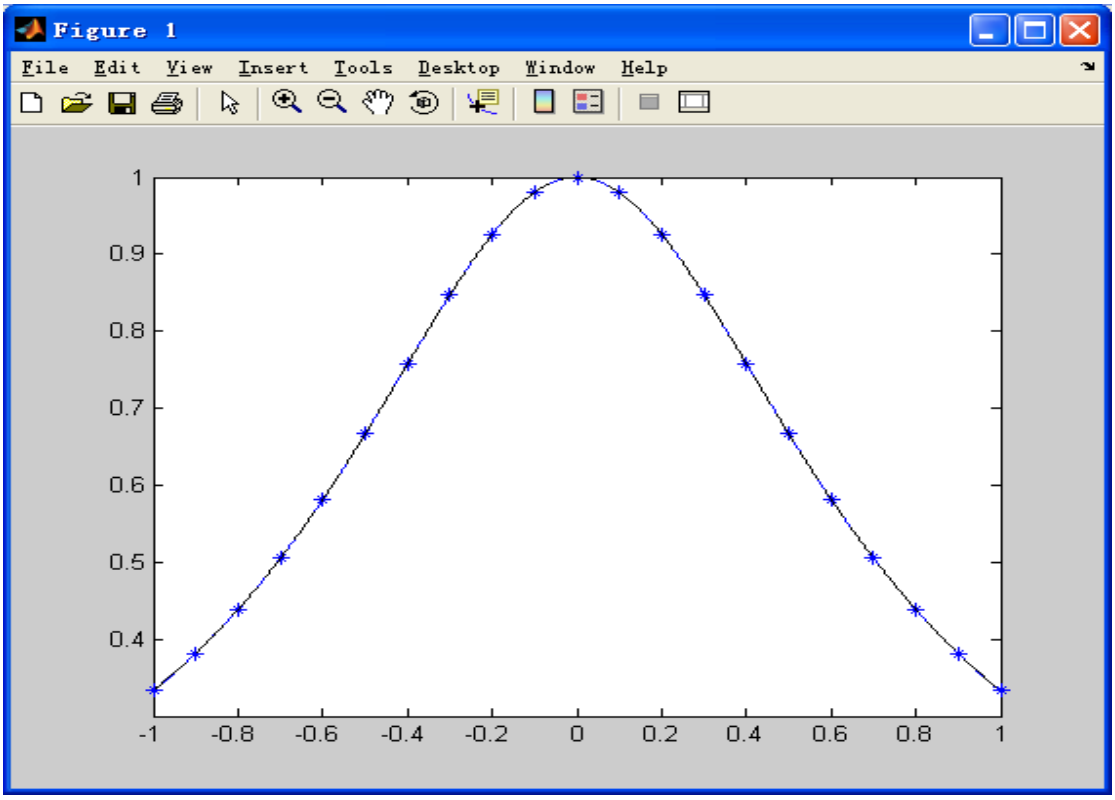


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

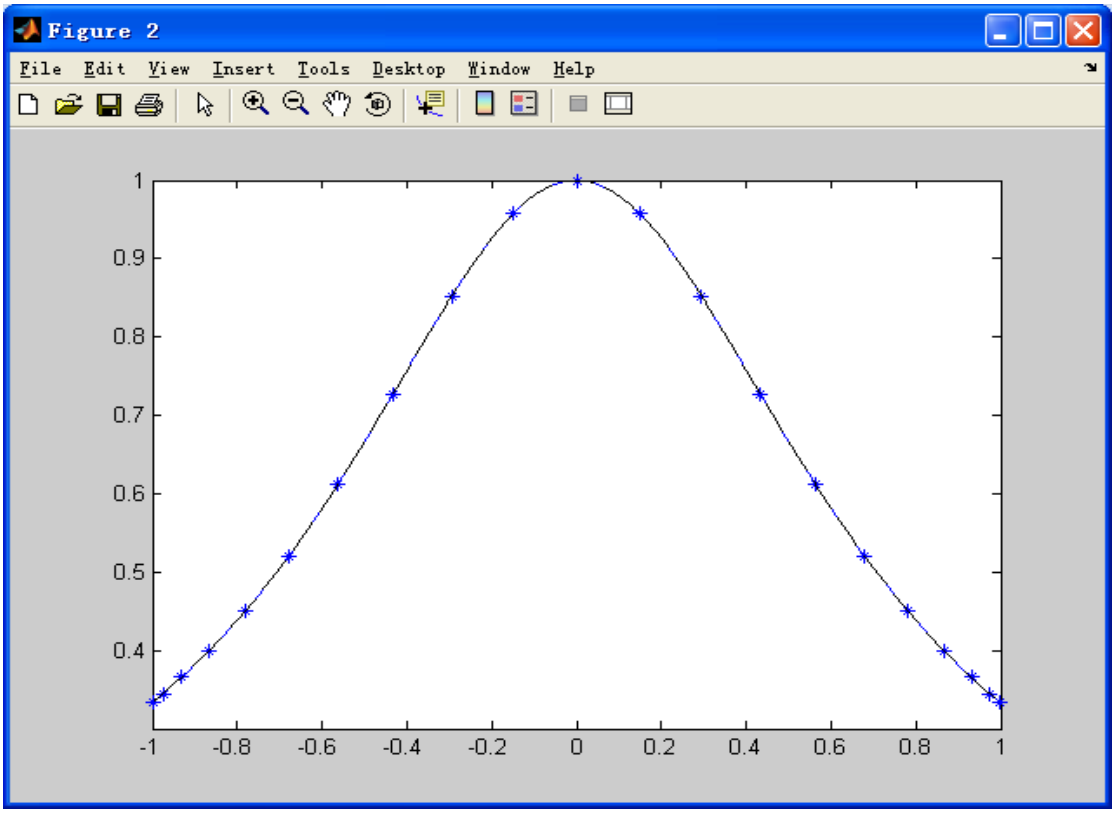


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

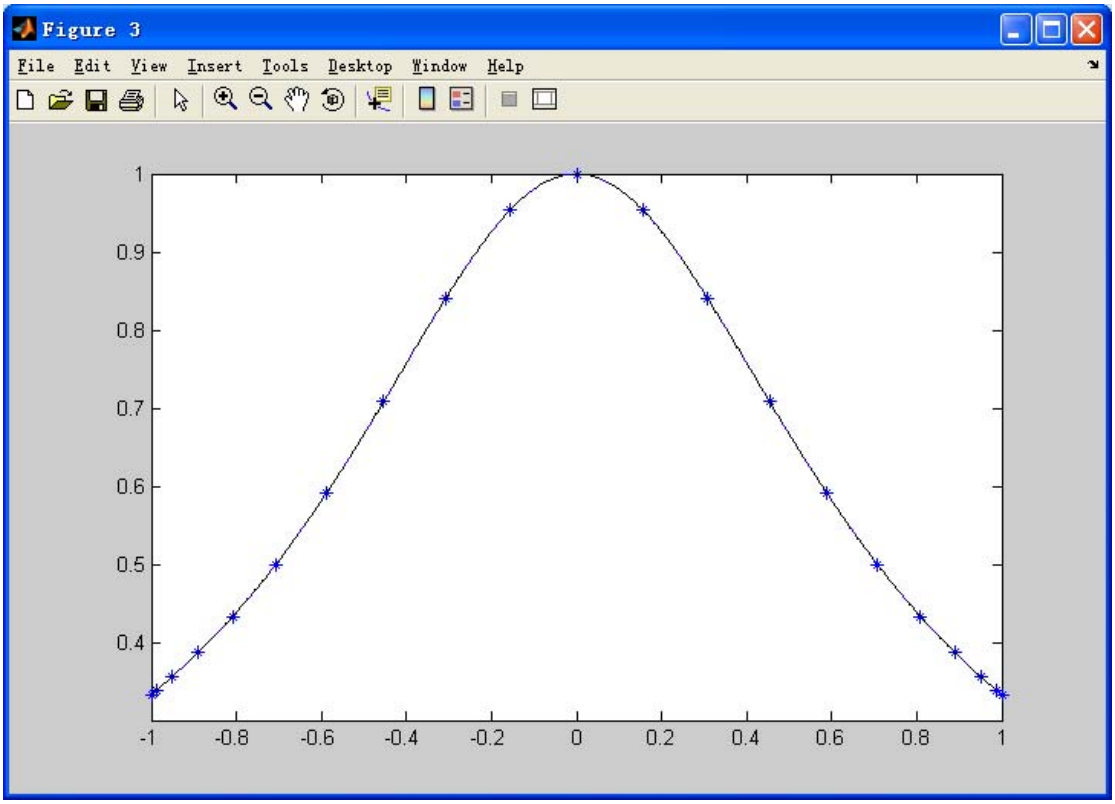


Figure 3
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

N=40

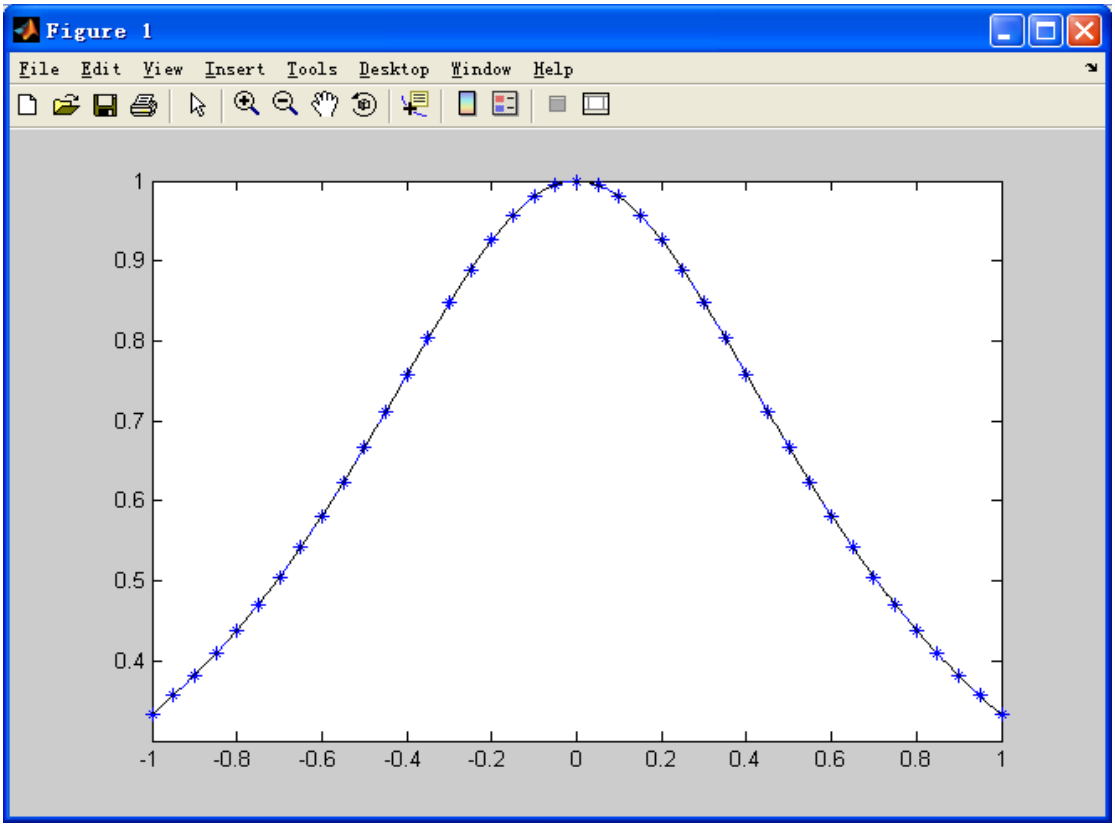


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

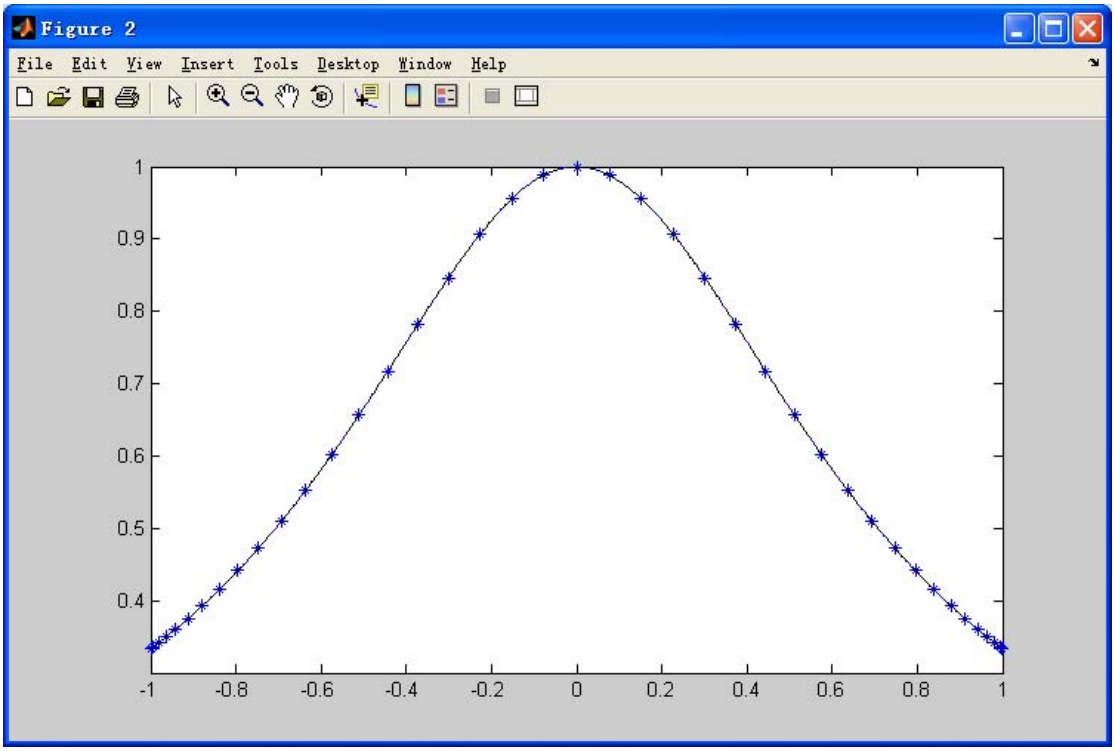
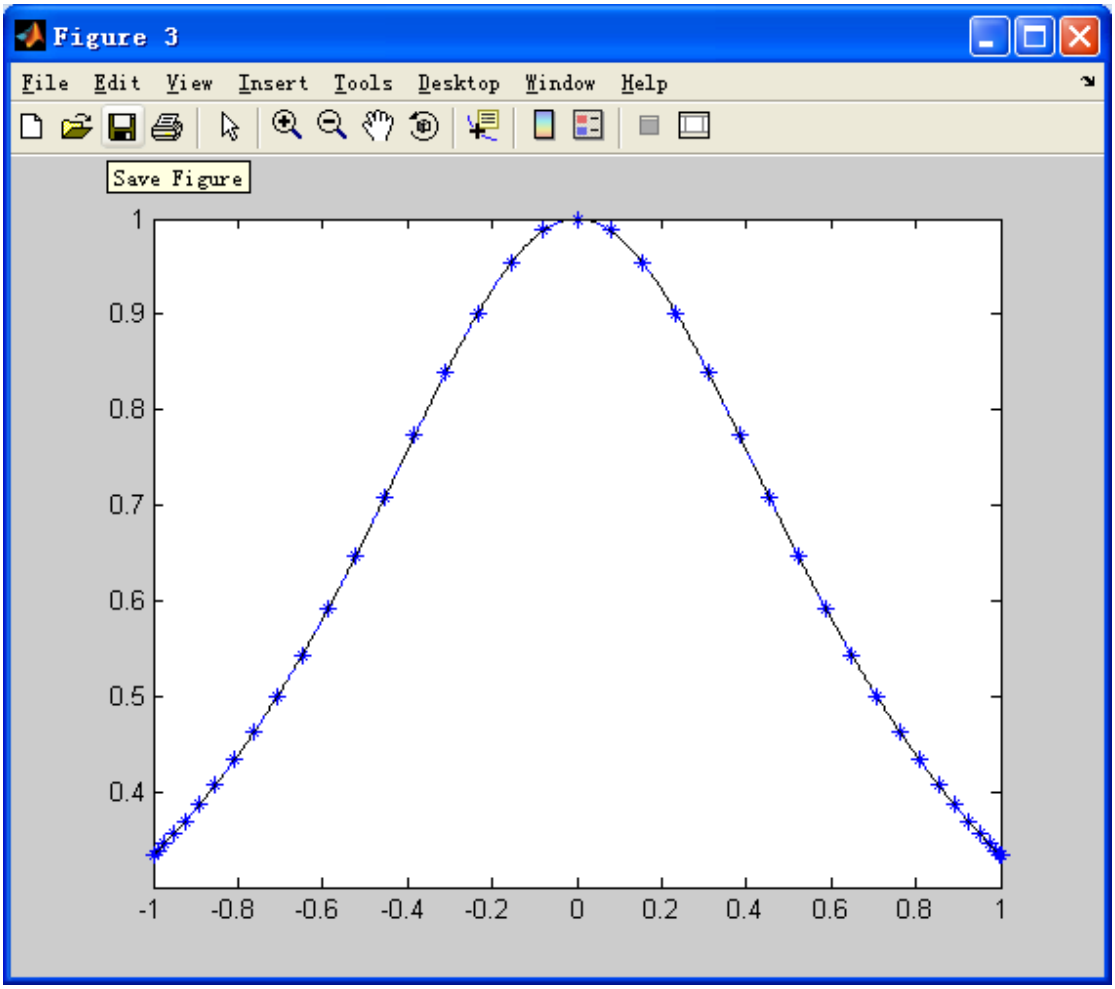


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1



All the interpolating polynomials match the f2 very well.

Alpha=5

N=10

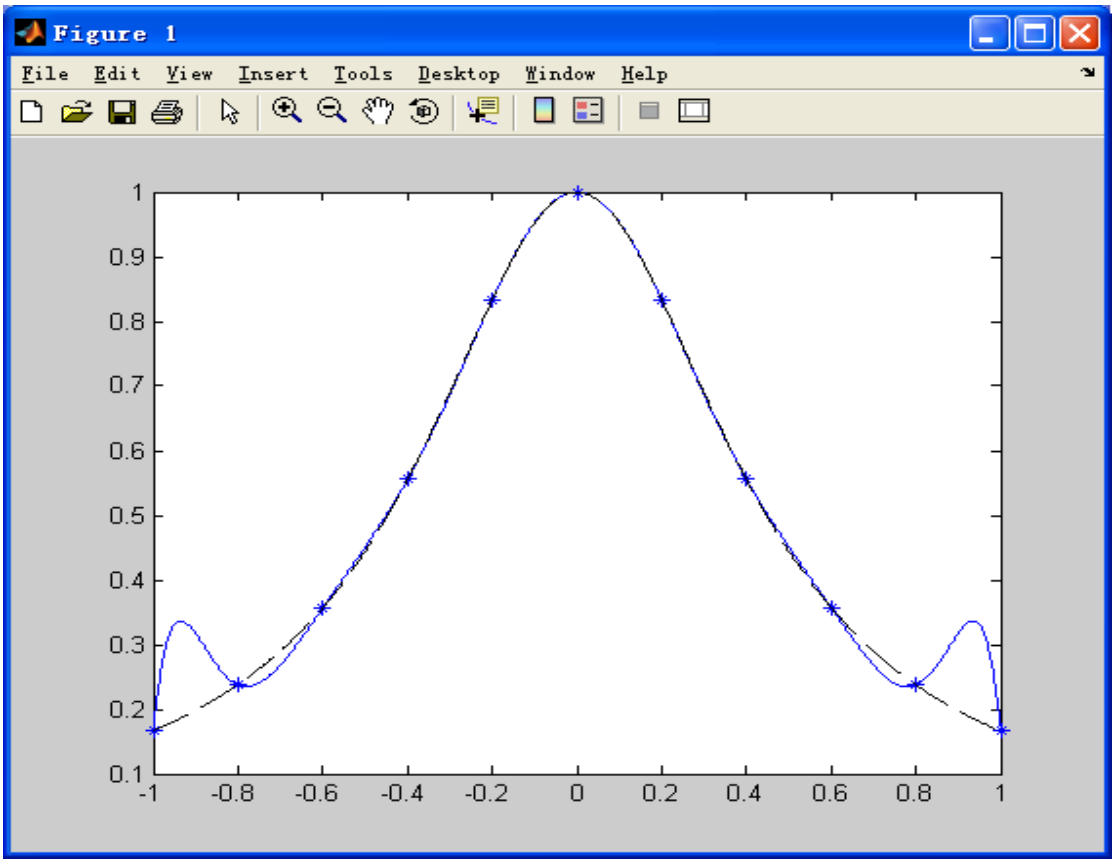


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

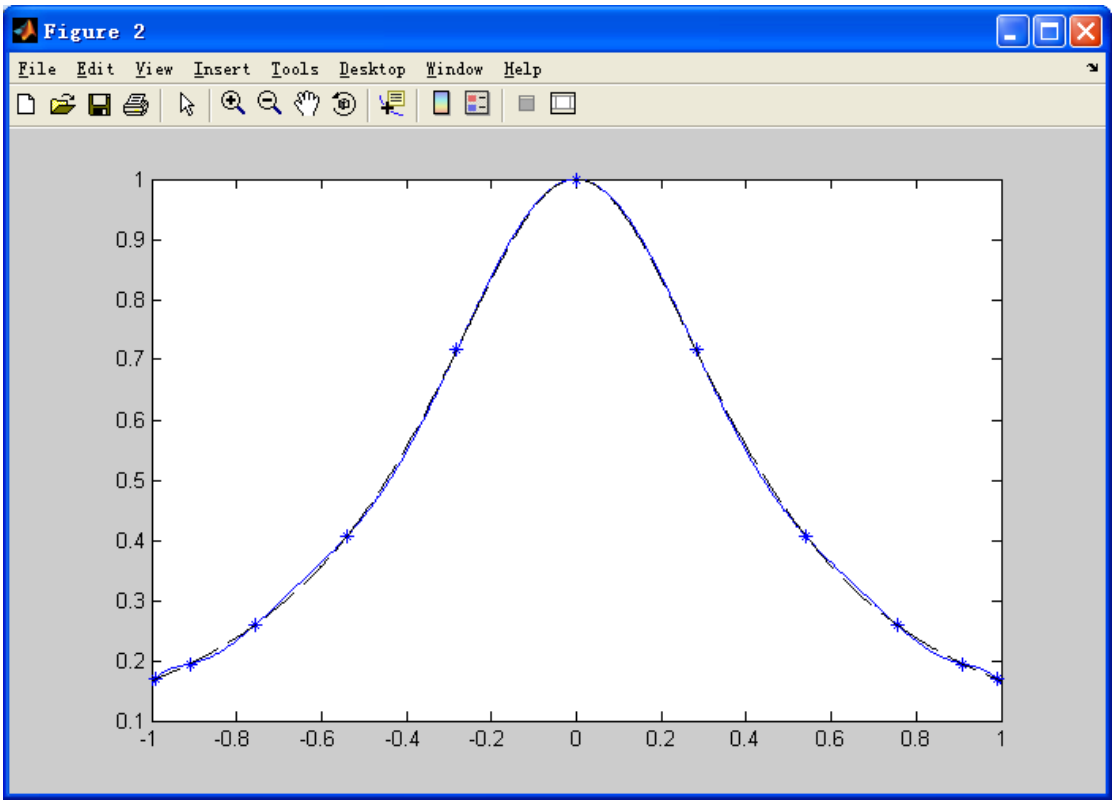


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

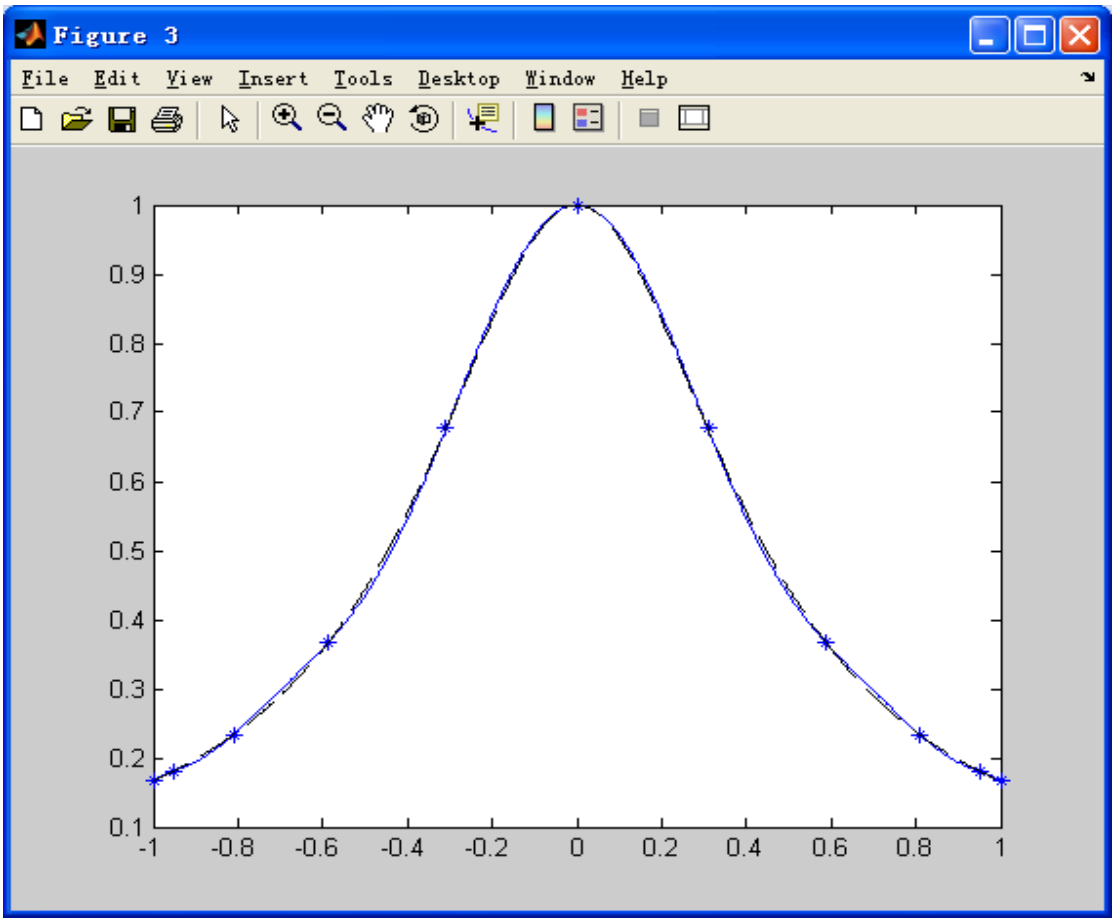


Figure 3
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

N=20

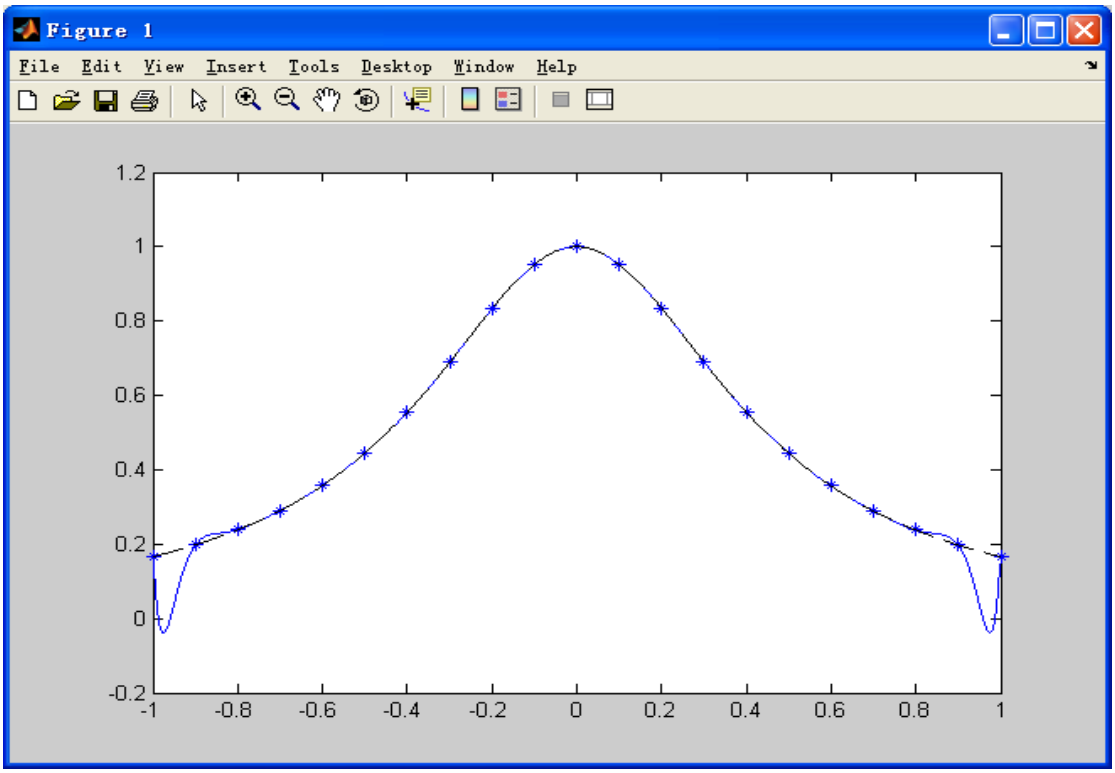


Figure 1
File Edit View Insert Tools Desktop Window Help
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

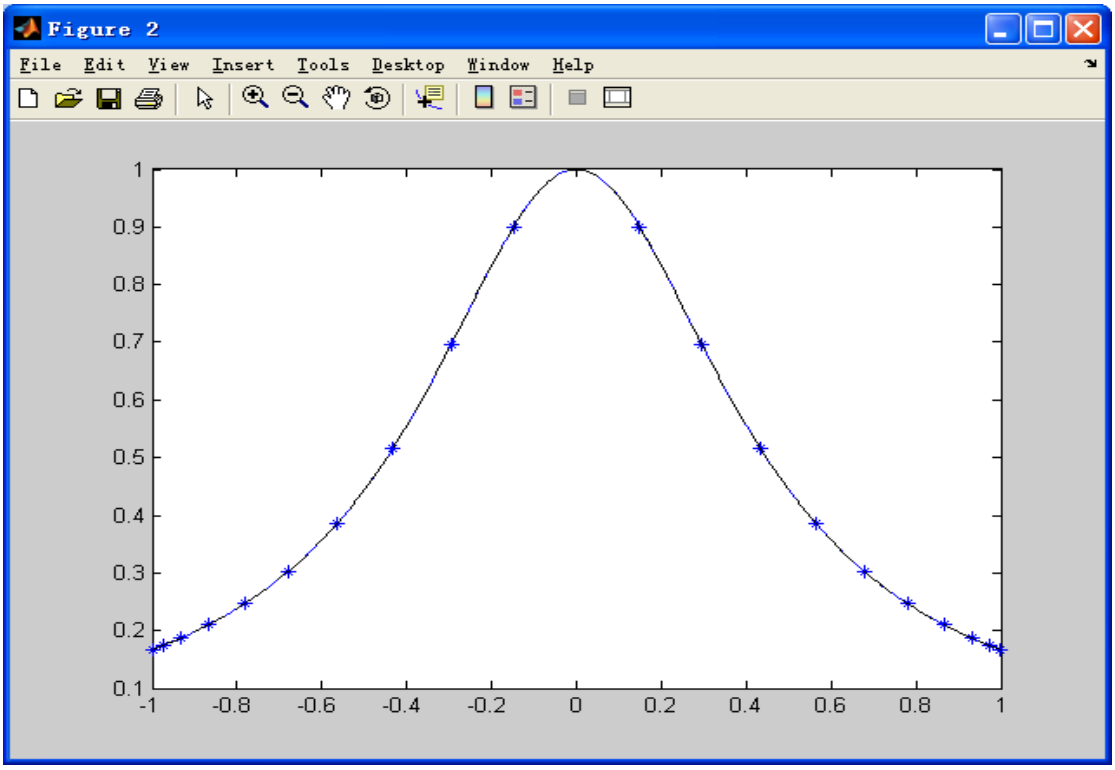


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

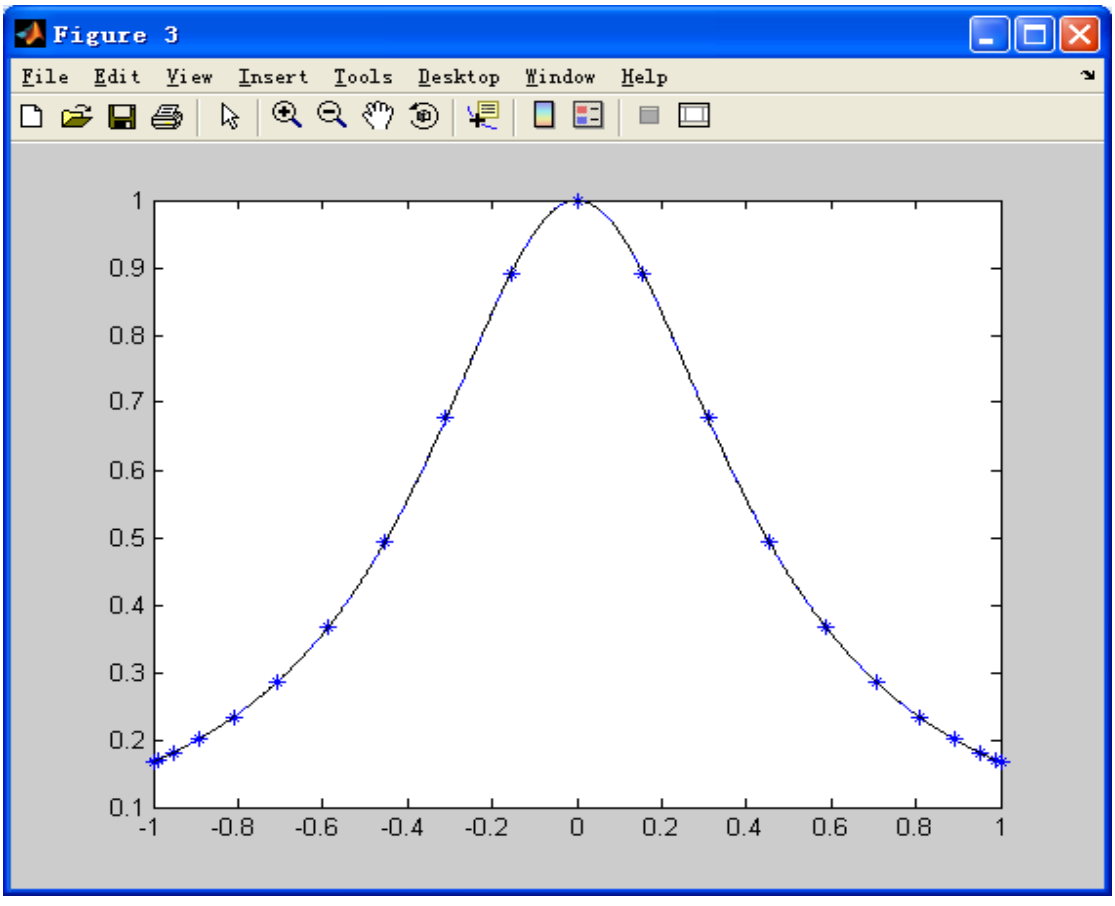


Figure 3
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

N=40

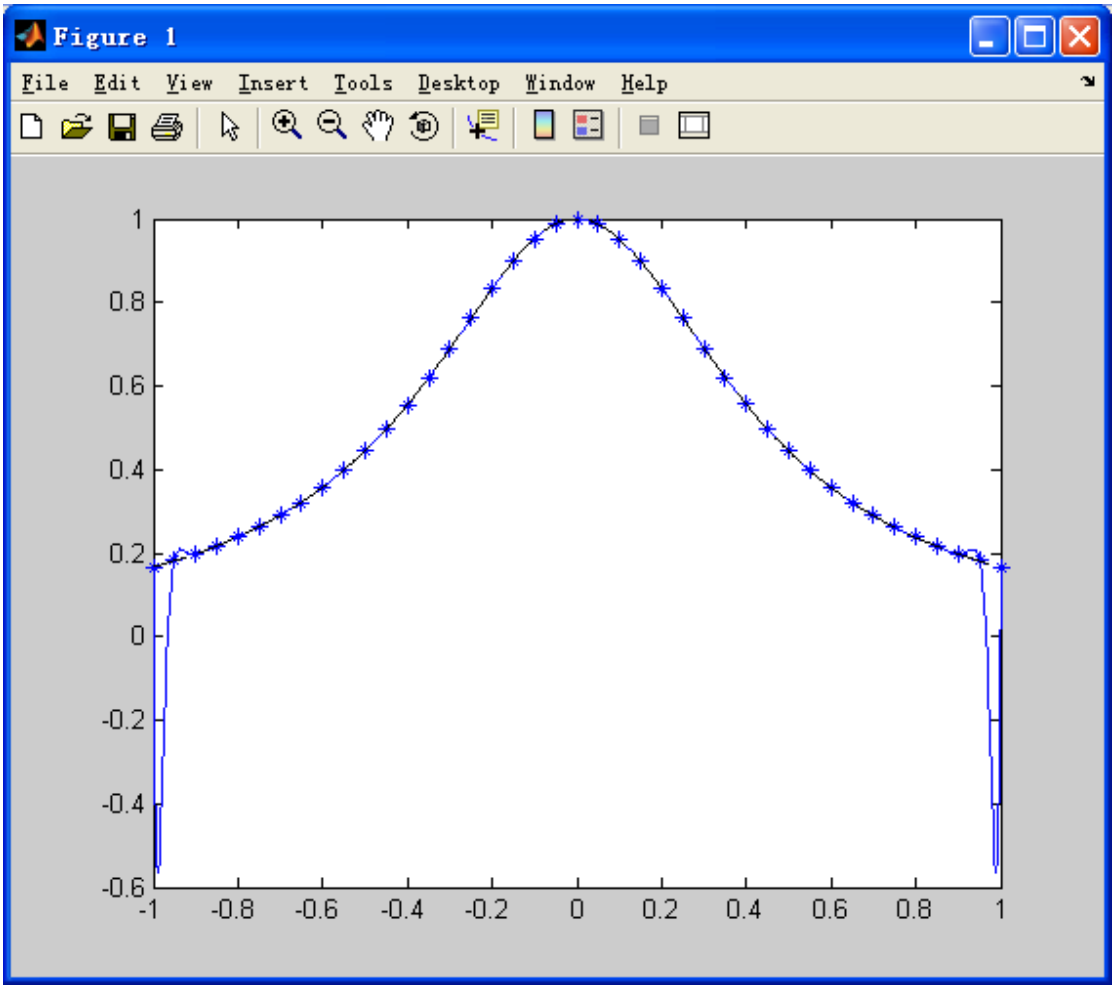


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

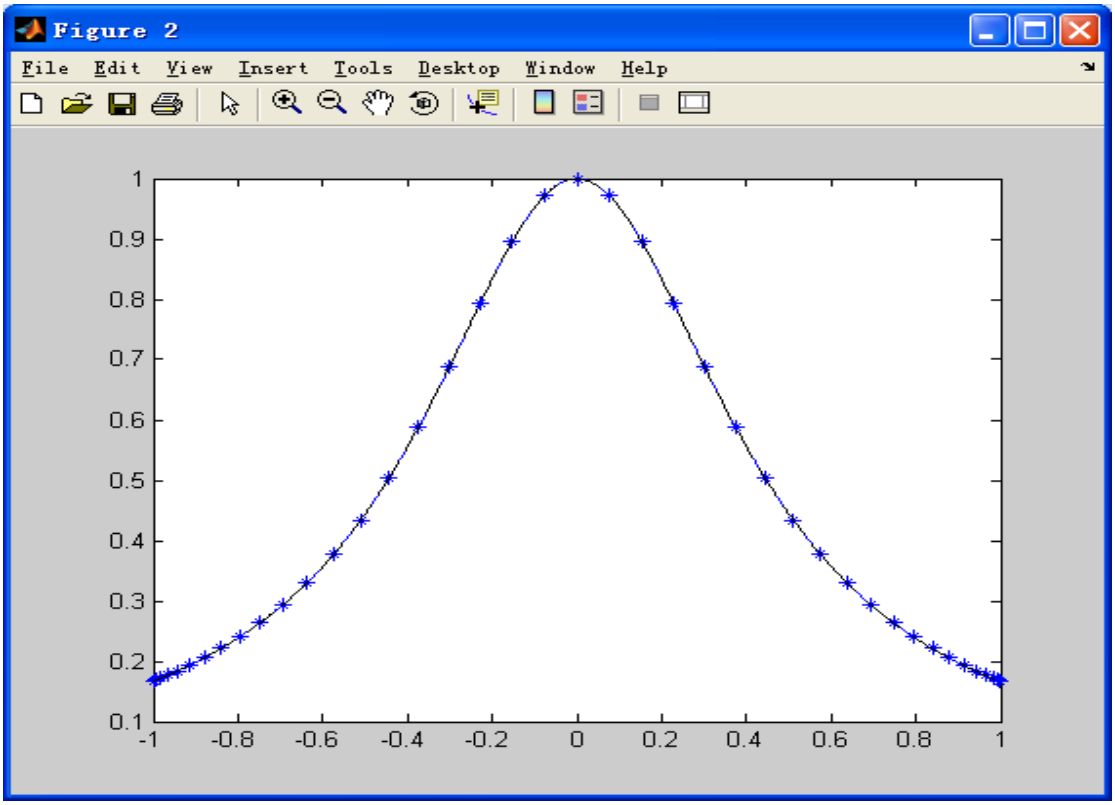


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

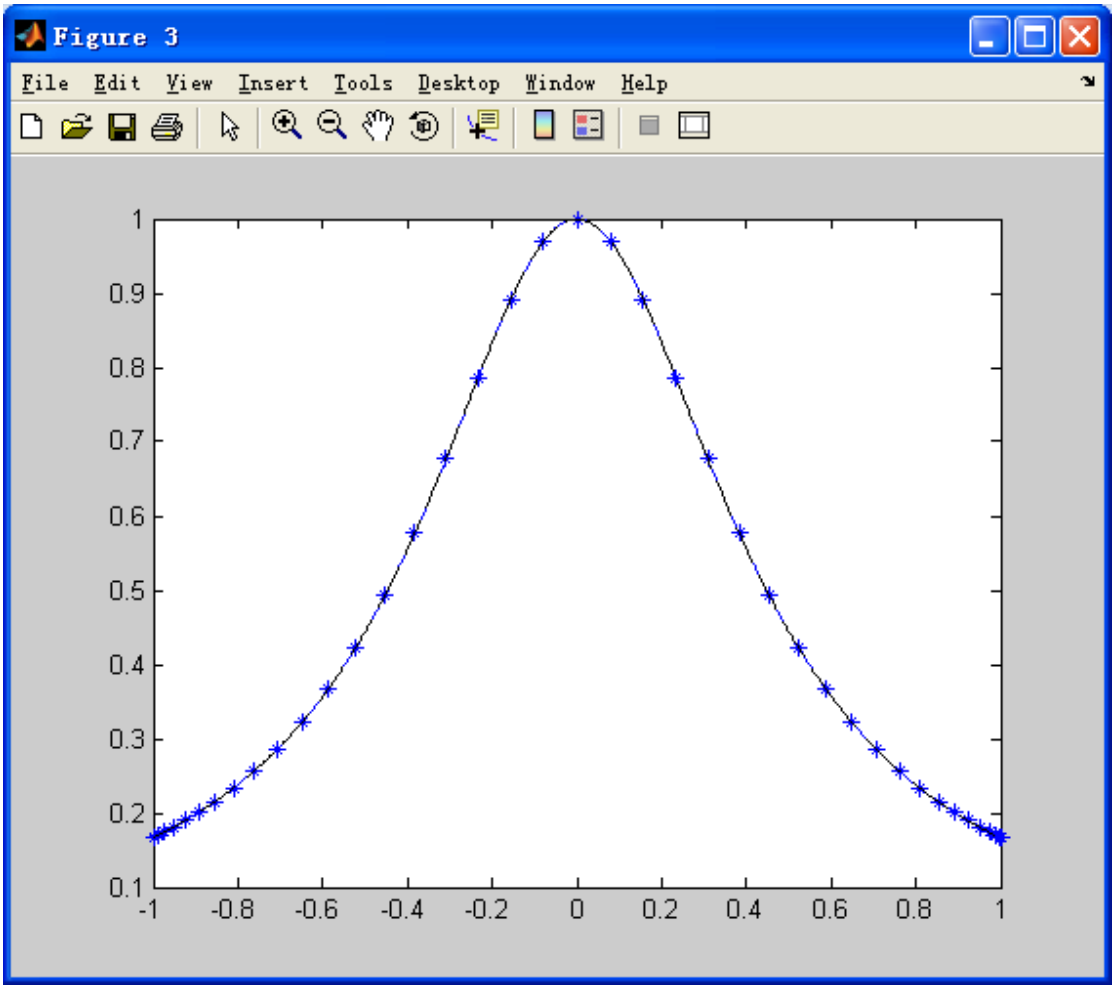


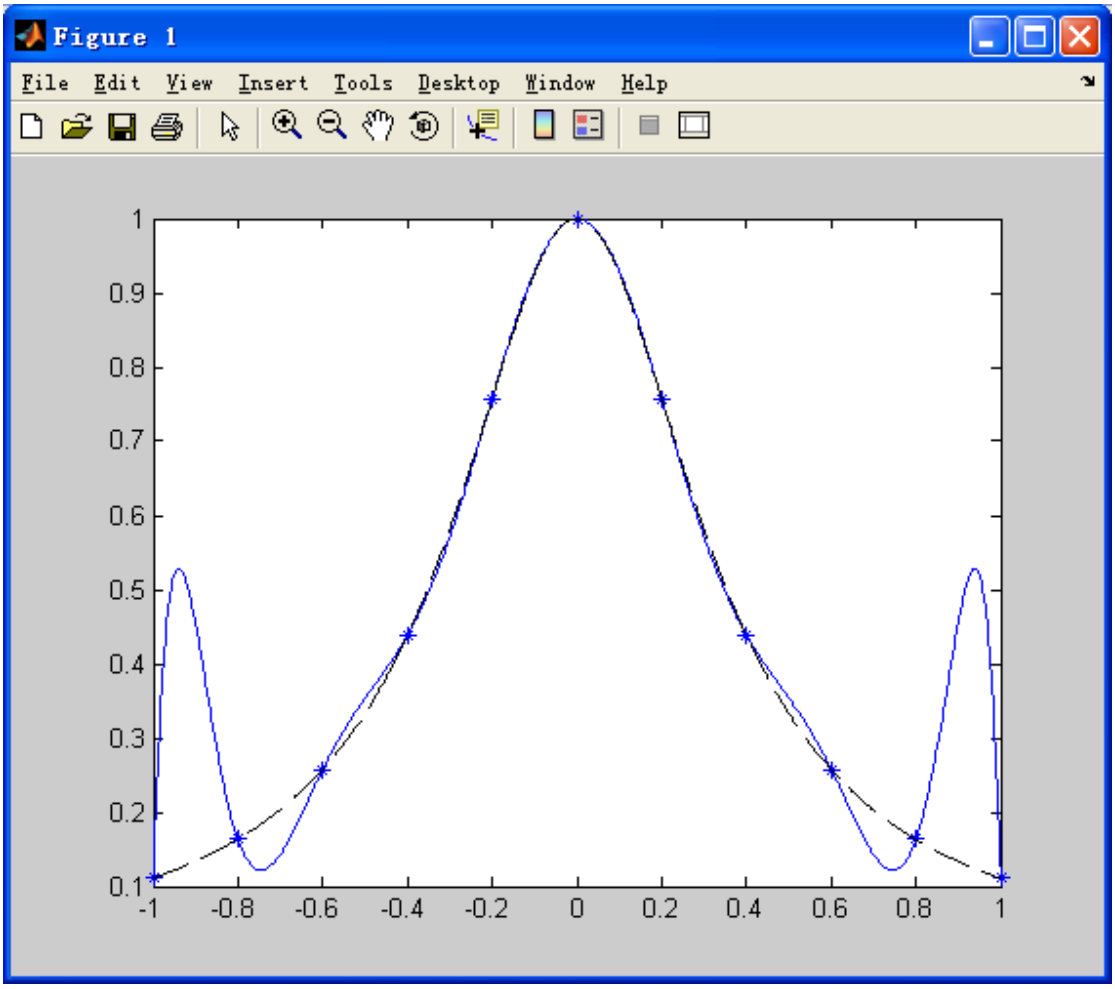
Figure 3
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

Both 1st Chebyshev points and 2nd Chebyshev points derive perfect interpolating polynomials

The uniformly points matched the f3 perfectly in the middle of interval but not well near the two endpoints. It performs totally different from f3 and oscillates near the two end points.

Alpha=8

N=10



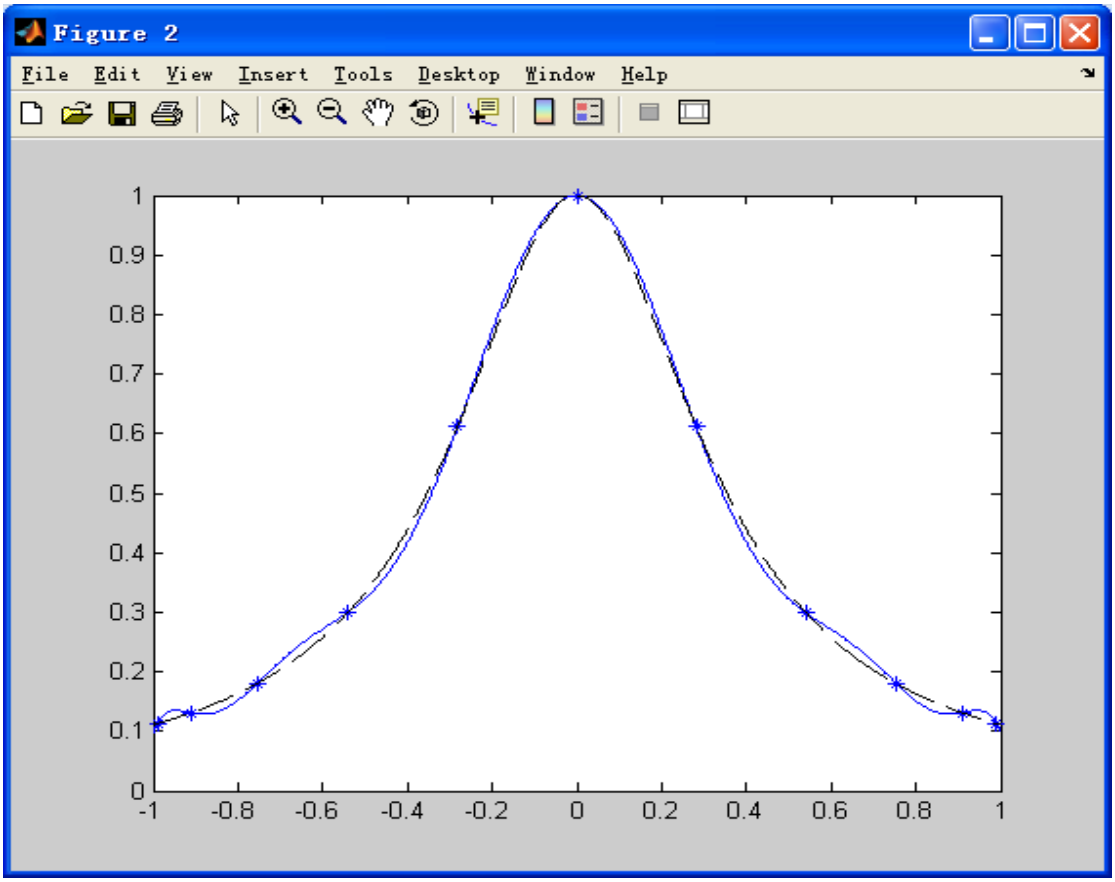


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

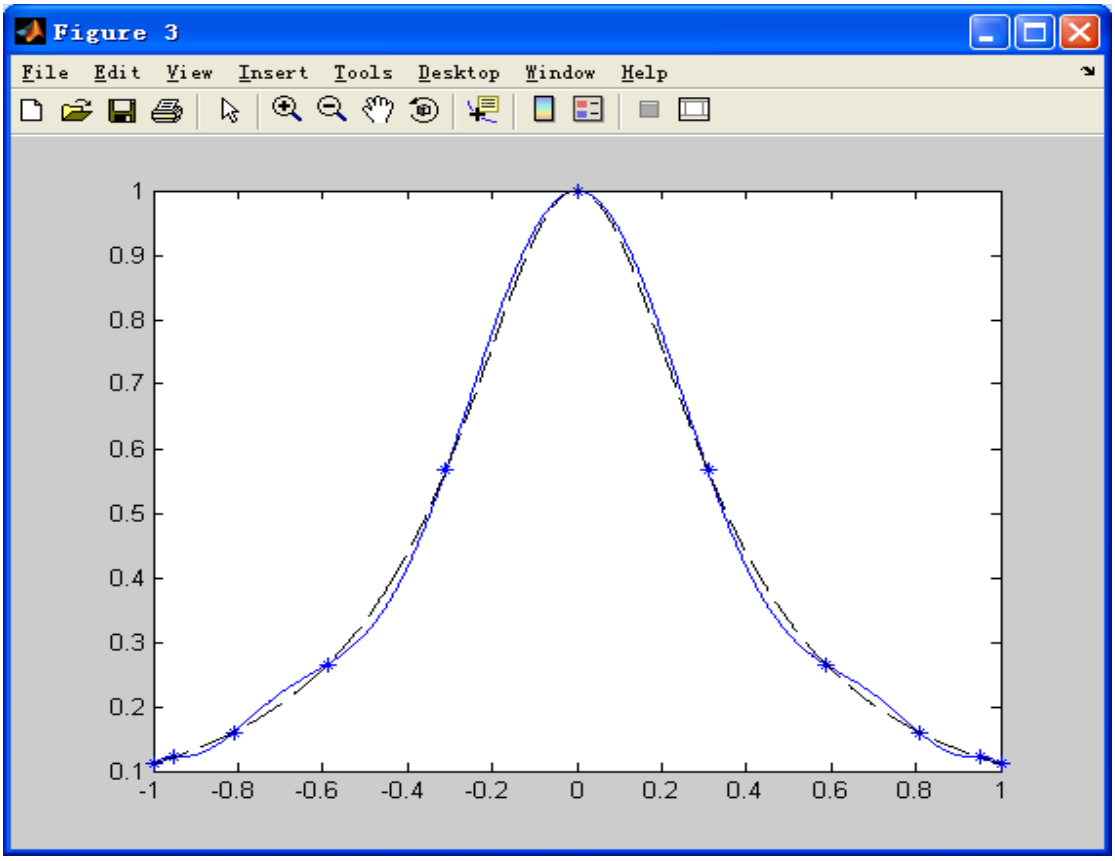


Figure 3
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

N=20

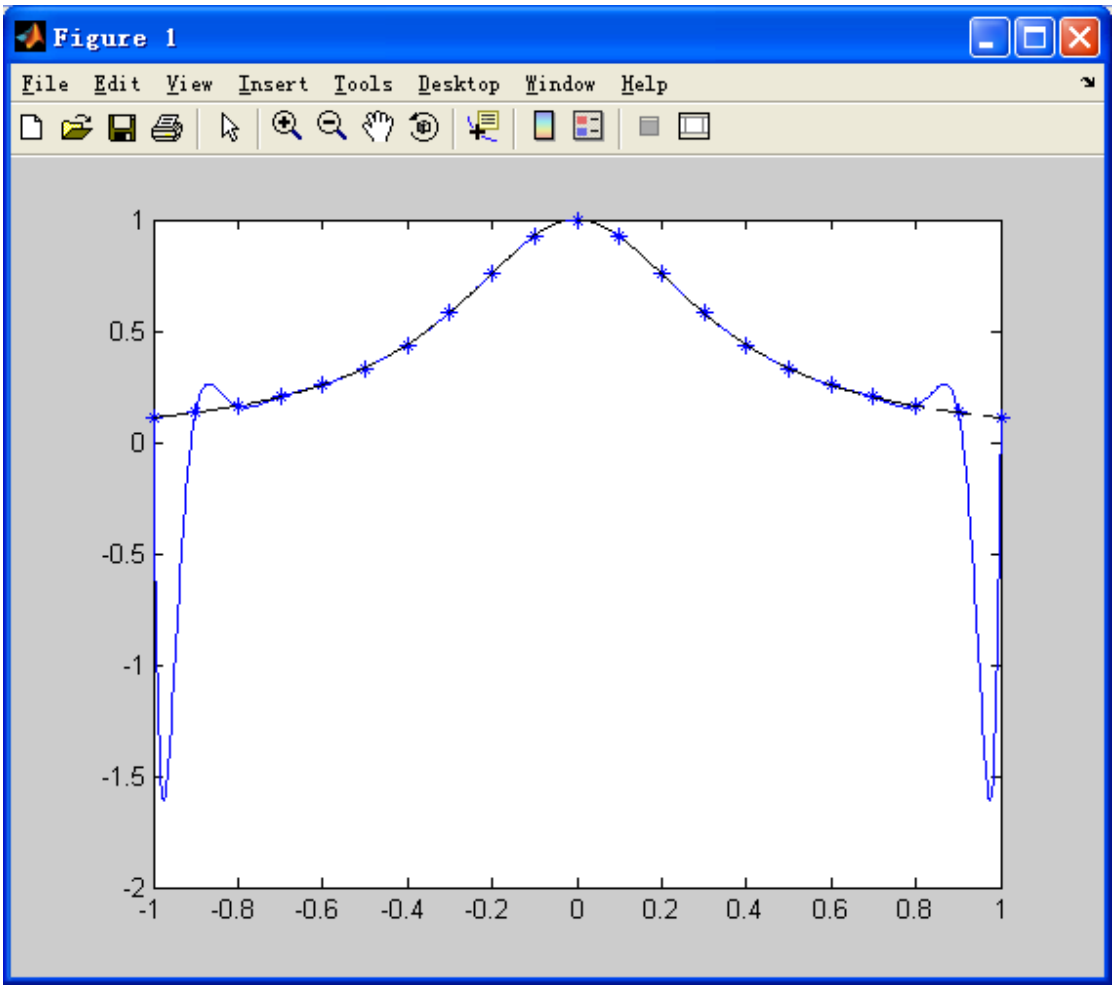


Figure 1
File Edit View Insert Tools Desktop Window Help
1
0.5
0
-0.5
-1
-1.5
-2
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

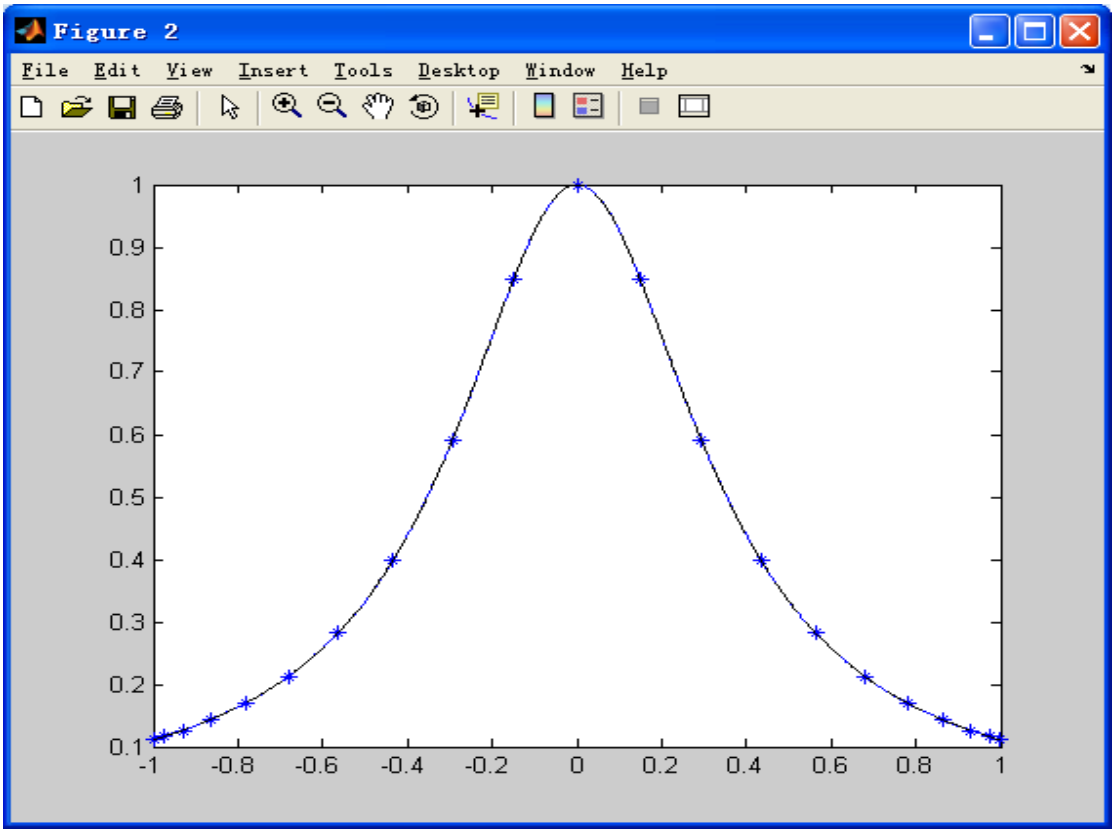


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

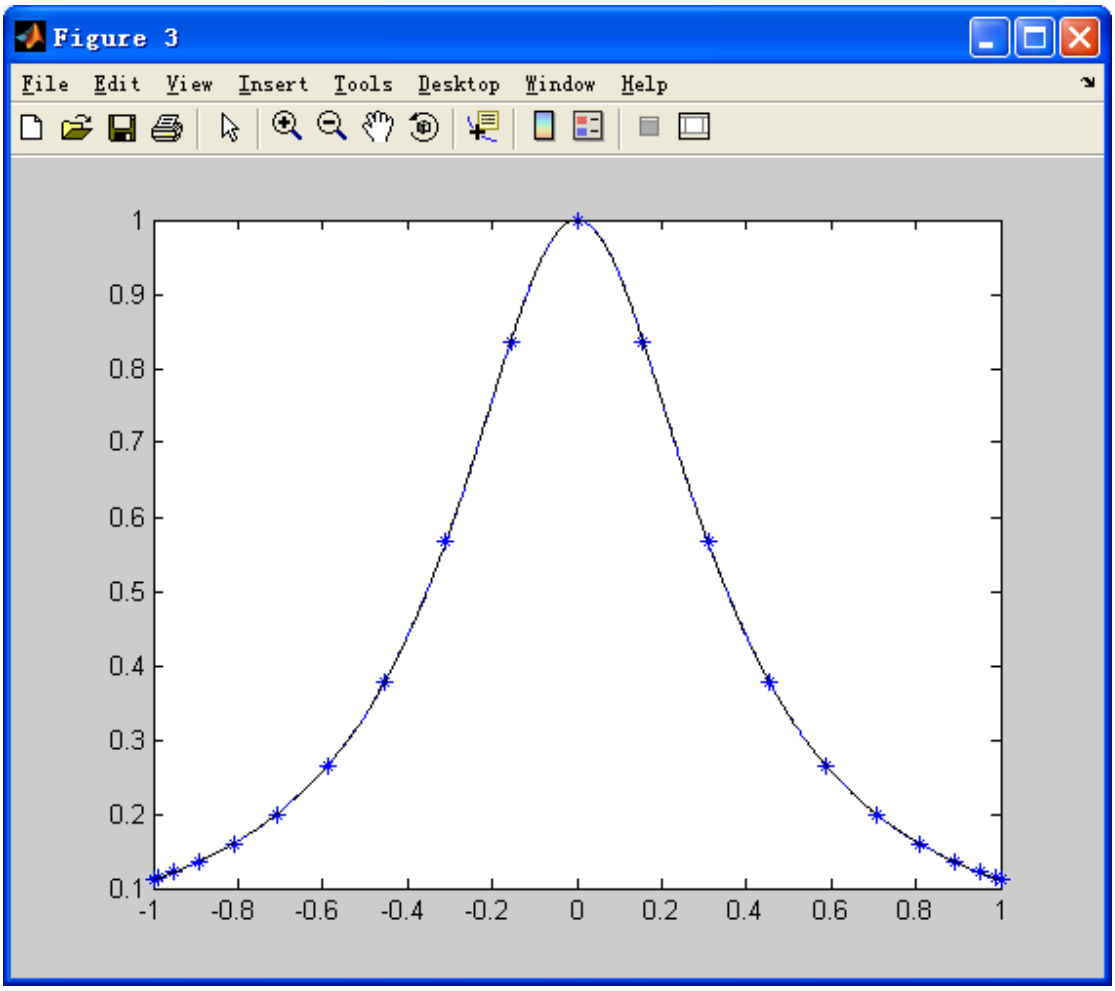


Figure 3
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

N=40

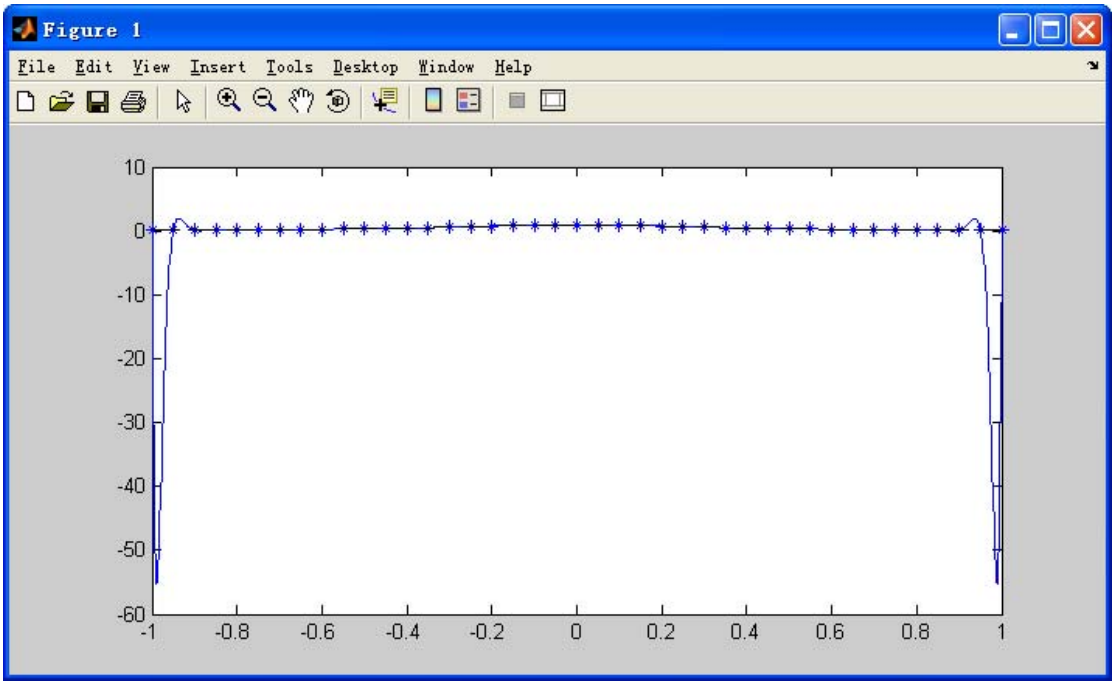


Figure 1
File Edit View Insert Tools Desktop Window Help
10
0
-10
-20
-30
-40
-50
-60
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

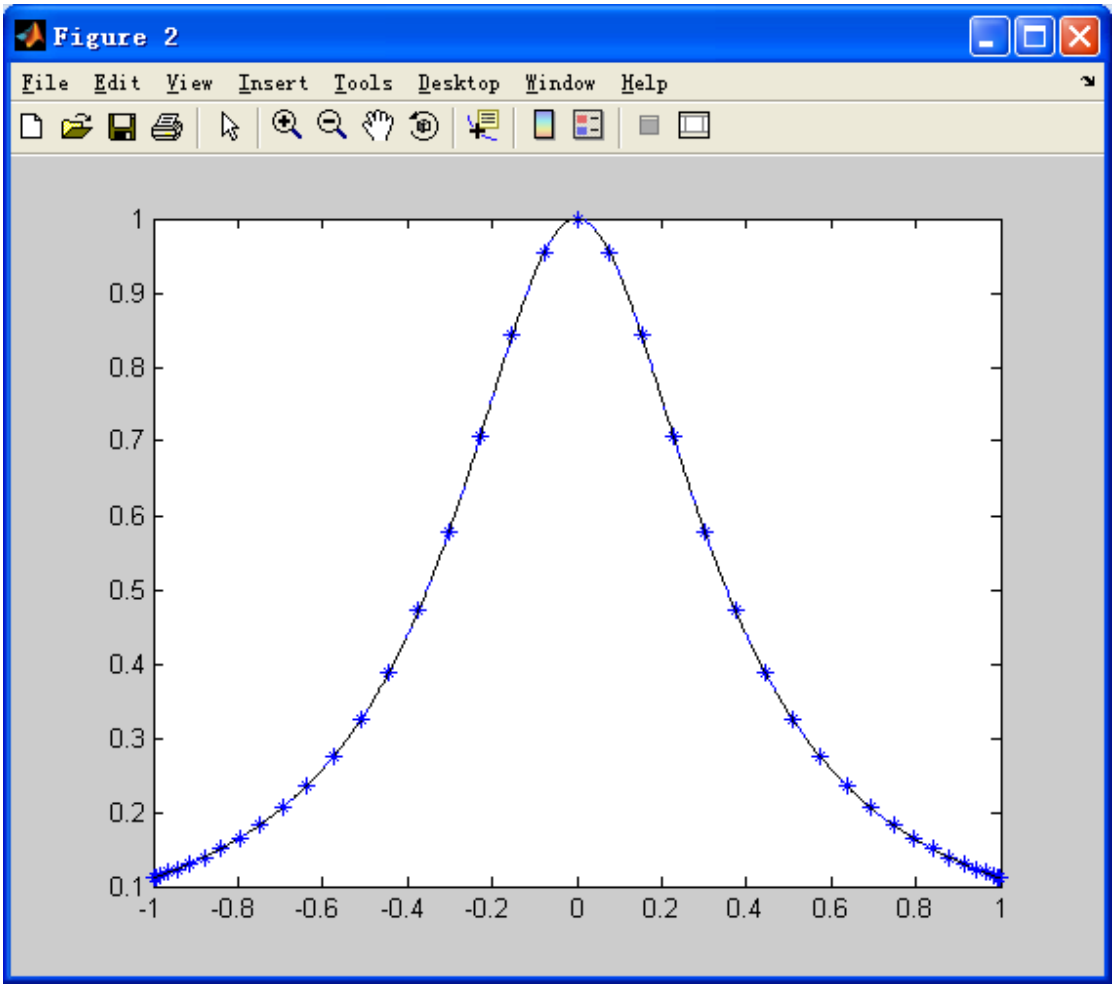
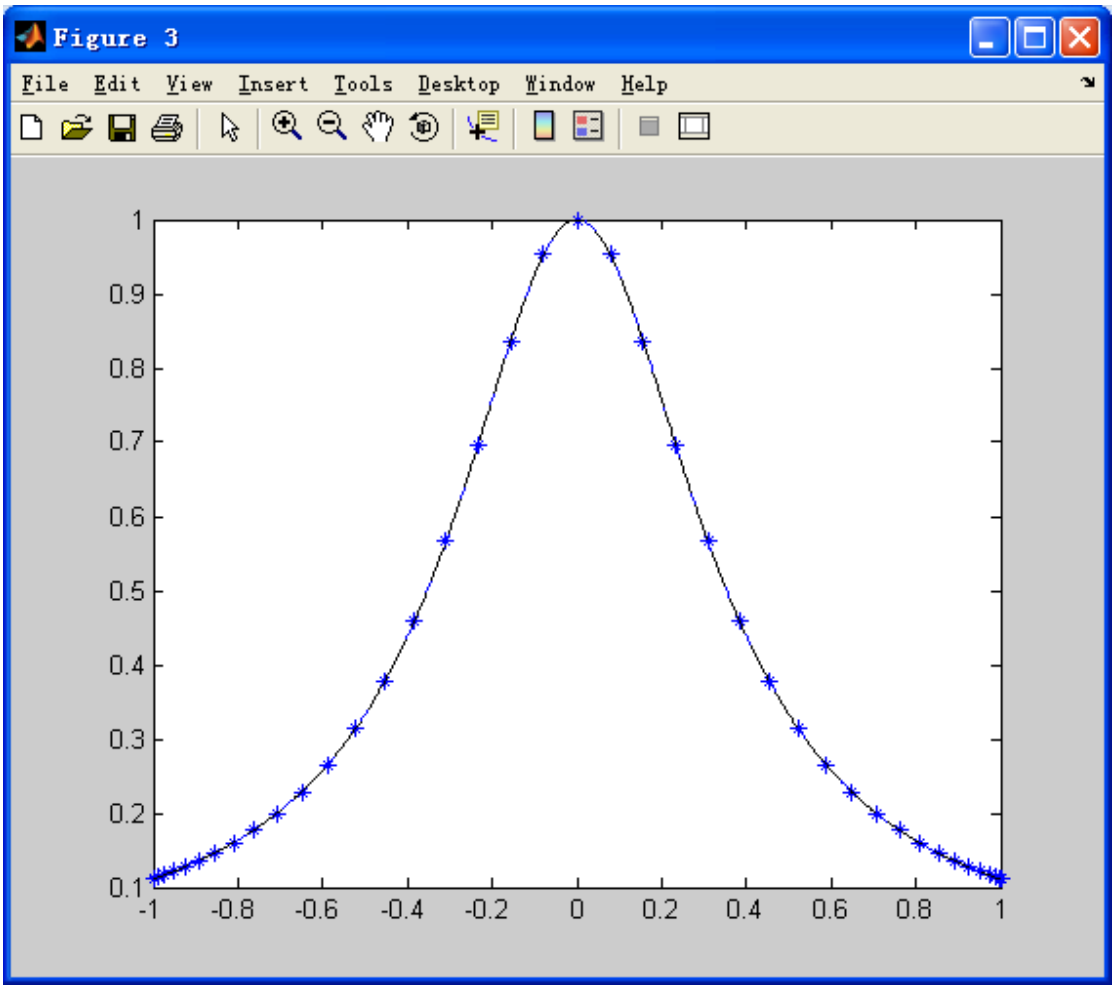


Figure 2
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1



Both 1st Chebyshev points and 2nd Chebyshev points derive perfect interpolating polynomials especially you increase the degree n.

The uniformly points matched the f4 only in the very middle of interval but simulate f4 very badly near the two endpoints. It performs totally different from f4 and oscillates huge near the two end points and produce a huge error near two end points.

alpha=2	error(flag=1)	error(flag=3)	error(flag=3)
n=10	0.0111	0.0007	0.0009
n=20	0.001	0	0
n=40	1.51E-05	0.00E+00	0.00E+00
alpha=5			
n=10	0.1515	0.0085	0.0107
n=20	0.2124	0.0001	0.0001
n=40	0.7357	0	0
alpha=8			
n=10	0.4045	0.0218	0.0256
n=20	1.7224	0.0007	0.0008
n=40	55.3879	0	0

From the table, also combine with the figures

We can conclude that

(1) Both 1st Chebyshev points and 2nd Chebyshev points derive perfect interpolating polynomials. And the error or the infinite norm decrease when n increase.

(2) The uniformly points set derive polynomials matches the function only in the very middle of interval but behaves very badly near the two endpoints. It will oscillate near the two end points and produce a huge error near two end points.

This algorithm is not trustworthy because it only behaves well for the specific function. i.e. only when alpha=2. It has a nice interpolating polynomial.

(3) All of the polynomials interpolating with smaller alpha behaves better than the those with larger alpha. For uniformly points, the larger the alpha is, the much worse it interpolates.

4.copy of code

Evaluate.m

```
function [value,y,fun]=evaluate(x,I,m,n,flag)%%evaluate poly at x,I is
the sequence of points oder
```

```
                              %%,m is degree of poly,n is the number of points
you choose,
                              %%flag is point set you choose.
if flag==1
   for i=1:(n+1)
      y(i)=-1+2*(i-1)/n;
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5

elseif flag==2
      for i=1:(n+1)
      y(i)=cos((2*i-1)*pi/(2*n+2));
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

elseif flag==3

         for i=1:(n+1)
      y(i)=cos(pi*(i-1)/n);
         end
end
         %% choose three kinds of points

   for i=1:(n+1)
   fun(i)=f1(y(i));
   end
    %%compute the functional value at the interpolating points
   for i=1:m+1
      yorder(i)=y(I(i)+1);
   end %%the points according to the sequence I

   %%now start to build the divided difference table
   A=zeros(n+1,n+1);
   for j=1:n+1
      A(1,j)=fun(j);
   end

   for i=2:n+1
      for j=i:n+1
         A(i,j)=(A(i-1,j)-A(i-1,j-1))/(y(j)-y(j-i+1));
```

```
        end
    end

    %% now we just choose specific path
    J(1)=I(1);
    a(1)=A(1,J(1)+1);
    for i=2:m+1

        if (I(i)-J(i-1))==1
            J(i)=I(i);
        elseif (J(1)-I(i))==1

                j=i;
               while j>=2
                J(j)=J(j-1);
                j=j-1;
               end
            J(1)=I(i);

        else

           error('the path is not available');
           break;

        end
                a(i)=A(i,J(i)+1);
    end

    %%now we start to evaluate the value at x
    value=a(m+1);
    i=m;
    while(i>=1)
        value=value*(x-yorder(i));
        value=value+a(i);
        i=i-1;
    end
```

test1.m

```
function test1(m,n)
for i=1:(m+1)
    I(i)=i-1;
end
xx=linspace(-1,1,1000);
```

```
for flag=1:3
    for i=1:1000
        [p(i),y,fun]=evaluate(xx(i),I,m,n,flag);
        ft(i)=f1(xx(i));
        temp(i)=abs(p(i)-ft(i));
    end
    error(flag)=max(temp);
    figure(flag);
    plot(y,fun,'*')
    hold on;
    plot(xx,p,'-')
     hold on;
    plot(xx,ft,'--k')
    hold on;

end
error
```

test2.m

```
function test2(I1,I2,m,n,flag)
xx=linspace(-1,1,1000);
    for i=1:1000
        [p1(i),y,fun]=evaluate(xx(i),I1,m,n,flag);
        [p2(i),y,fun]=evaluate(xx(i),I2,m,n,flag);
    end
    error=max(abs(p1-p2))
    figure(1);
plot(xx,p1,'-k')
hold on;
plot(xx,p2,'--k')
hold on;
```

test3.m

```
function test3(m,n)
for i=1:(m+1)
    I(i)=i-1;
end
xx=linspace(-1,1,1000);
for flag=1:3
    for i=1:1000
```

```
        [p(i),y,fun]=evaluate(xx(i),I,m,n,flag);
        ft(i)=f4(xx(i));
        temp(i)=abs(p(i)-ft(i));
    end
    error(flag)=max(temp);
    figure(flag);
    plot(y,fun,'*')
    hold on;
    plot(xx,p,'-')
     hold on;
    plot(xx,ft,'--k')
    hold on;

end
error
```

test4.m

```
function test4(I1,m,n,flag)
xx=linspace(-1,1,1000);
    for i=1:1000
        [p1(i),y,fun]=evaluate(xx(i),I1,m,n,flag);
    end
    plot(xx,p1,'-')



function y=f1(x)
y=x.^10-x.^7;



function y=f2(x)
y=1/(1+2*x^2);


function y=f3(x)
y=1/(1+5*x^2);


function y=f4(x)
y=1/(1+8*x^2);
```