MATH 571: Coding Assignment #1

Due on Wednesday, October 16, 2013

TTH 12:40pm

Wenqiang Feng

Contents

Problem 1

1. See Listing 3.

2. See Listing 2.

3. We should get the Identity square matrices. But we did not get the actual Identity square matrices through the Gram-Schmidt Algorithm. For case 1-2, we only get the matrices which $diag(Q^*Q) = \{\overbrace{1, \cdots, 1}^{n}\}$ and the other elements approximate to 0 in the sense of $C \times 10^{-16} \sim 10^{-17}$. For case 3, Classical Gram-Schmidt Algorithm is not stable for case 3, since some elements of matrix Q^*Q do not approximate to 0, then the matrix Q^*Q is not diagonal any more.

4. For case 1-2, we also did not get the actual Identity square matrices by using the Modified Gram-Schmidt Algorithm. We only get the matrices which $diag(Q^*Q) = \{\overbrace{1, \cdots, 1}^{n}\}$ and the other elements approximate to 0 in the sense of $C \times 10^{-17} \sim 10^{-18}$. For case 3, the Modified Gram-Schmidt Algorithm works well for case 3, we get the matrix which $diag(Q^*Q) = \{\overbrace{1, \cdots, 1}^{n}\}$ and the other elements approximate to 0 in the sense of $C \times 10^{-8} \sim 10^{-13}$. So, Modified Gram-Schmidt Algorithm is more stable than the Classical one.

Listing 1 shows the main function for problem1.

Listing 1: Main Function of Problem1

```
%Main function
clc
clear all
m=20;n=10;
fun1=@(i,j) ((2*i-21)/19)^(j-1);
fun2=@(i,j) 1/(i+j);
A1=rand(m,n);
A2=matrix_gen(m,n,fun1);
A3=matrix_gen(m,n,fun2);
% Test  for the random case 1
[CQ1,CR1]=gschmidt(A1)
[MQ1,MR1]=mgschmidt(A1)
q11=CQ1'*CQ1
q12=MQ1'*MQ1
% Test for case 2
[CQ2,CR2]=gschmidt(A2)
[MQ2,MR2]=mgschmidt(A2)
q21=CQ2'*CQ2
q22=MQ2'*MQ2
% Test for case 3
[CQ3,CR3]=gschmidt(A3)
[MQ3,MR3]=mgschmidt(A3)
q31=CQ3'*CQ3
q32=MQ3'*MQ3
```

Listing 2 shows the matrices generating function.

Listing 2: Matrices Generating Function

```
function A=matrix_gen(m,n,fun)
A=zeros(m,n);
for i=1:m
    for j=1:n
        A(i,j)=fun(i,j);
    end
end
```

Listing 3 shows Classical Gram-Schmidt Algorithm.

Listing 3: Classical Gram-Schmidt Algorithm

```
function [Q,R]=gschmidt(V)
% gschmidt:  classical Gram-Schmidt algorithm
%
% USAGE
%        gschmidt(V)
%
% INPUT
%      V: V is an m by n matrix of full rank m<=n
%
% OUTPUT
%     Q: an m-by-n matrix with orthonormal columns
%     R: an n-by-n upper triangular matrix
%
% AUTHOR
%    Wenqiang Feng
%    Department of Mathematics
%    University of Tennessee at Knoxville
%    E-mail: wfeng@math.utk.edu
%    Date:   9/14/2013

[m,n]=size(V);
Q=zeros(m,n);
R=zeros(n);
R(1,1)=norm(V(:,1));
Q(:,1)=V(:,1)/R(1,1);
for k=2:n
    R(1:k-1,k)=Q(:,1:k-1)'*V(:,k);
    Q(:,k)=V(:,k)-Q(:,1:k-1)*R(1:k-1,k);
    R(k,k)=norm(Q(:,k));
        if R(k,k) == 0
             break;
        end
    Q(:,k)=Q(:,k)/R(k,k);
end
```

Listing 4 shows Modified Gram-Schmidt Algorithm.

Listing 4: Modified Gram-Schmidt Algorithm

```
function [Q,R]=mgschmidt(V)
```

```
% mgschmidt:   Modified Gram-Schmidt algorithm
%
% USAGE
%        mgschmidt(V)
%
% INPUT
%      V: V is an m by n matrix of full rank m<=n
%
% OUTPUT
%     Q: an m-by-n matrix with orthonormal columns
%     R: an n-by-n upper triangular matrix
%
% AUTHOR
%    Wenqiang Feng
%    Department of Mathematics
%    University of Tennessee at Knoxville
%    E-mail: wfeng@math.utk.edu
%    Date:   9/14/2013

[m,n]=size(V);
Q=zeros(m,n);
R=zeros(n);

for k=1:n
    R(k,k)=norm(V(:,k));
        if R(k,k) == 0
             break;
        end
    Q(:,k)=V(:,k)/R(k,k);
        for j=k+1:n
            R(k,j)=Q(:, k)'* V(:,j);
            V(:, j) = V(:, j)-R(k, j) * Q(:,k);
        end
end
```

Problem 2

1. I Chose $N = 10$ and I got the polynomial p_{10} is as follow:

$$
\begin{aligned}
P_10 &= -220.941742081448x^{10} + 7.38961566181029e^{-13}x^9 + 494.909502262444x^8 \\
&\quad -1.27934383856085e^{-12}x^7 - 381.433823529411x^6 + 5.56308212237901e^{-13}x^5 \\
&\quad +123.359728506787x^4 - 1.16016030941682e^{-14}x^3 - 16.8552036199095x^2 \\
&\quad -5.86232968237562e^{-15}x + 1.00000000000000
\end{aligned}
$$

2. See Figure 1.

3. See Listing 6.

4. Since A is Vandermonde Matrix and all the points x_i are different, then $det(A) \neq 0$. Therefore A has full rank.

5. I varied N from 3 to 15. For every fixed N, I varied n form 1 to N. Then I got the following table (Table.1). From table (Table.1), we can get that $n \approx 2\sqrt{N+1}$, where the N is the number of the partition.

$N \setminus h$	1	2	3	4	5	6	7	8	$\cdots$
3	0.23	$3.96 \cdot 10^{-17}$	$5.55 \cdot 10^{-17}$						
4	0.82	0.56	0.56	$5.10 \cdot 10^{-17}$					
5	0.50	0.28	0.28	$9.04 \cdot 10^{-16}$	$9.32 \cdot 10^{-16}$				
6	0.84	0.62	0.62	0.43	0.43	$8.02 \cdot 10^{-15}$			
7	0.71	0.46	0.46	0.25	0.25	$3.32 \cdot 10^{-15}$	$3.96 \cdot 10^{-15}$		
8	0.89	0.64	0.64	0.45	0.45	0.30	0.30	$1.39 \cdot 10^{-14}$	
$\vdots$									

Table 1: The L^2 norm of the Least squares polynomial fit

Fix $N = 10$, vary n(Figure 2-Figure 11).

Listing 5 shows main function of problem2.1.

Listing 5: Main Function of Problem2.1

```
% Main function of A2
clc
clear all
N=10;
n=N;
fun= @(x) 1./(1+25*x.^2);
x=-1:2/N:1;
y=fun(x);

x1=-1:2/(2*N):1;
a = polyfit(x,y,n);
p = polyval(a,x1)
plot(x,y,'o',x1,p,'-')

for m=1:10
least_squares(x, y, m)
end
```

Listing 6 shows Polynomial Least Squares Fitting Algorithm.

Listing 6: Polynomial Least Squares Fitting Algorithm

```
%Main function for pro#2.5
clc
clear all
for N=3:15
j=1;
for n=1:N%3:N;
fun= @(x) 1./(1+25*x.^2);
```

```
x=-1:2/N:1;
b=fun(x);

A=MatrixGen(x,n);
cof=GSsolver(A,b);
q=0;
     for i=1:n+1
        q=q+cof(i)*(x.^(i-1));
     end
error(j)=norm(q-b);
j=j+1;
error
end
end


function A=MatrixGen(x,n)
m=size(x,2);
A=zeros(m,n+1);
for i=1:m
    for j=1:n+1
        A(i,j)=x(i).^(j-1);
    end
 end


function x=GSsolver(A,b)
   [Q,R]=mgschmidt(A);
    x= R\(Q'*b');
```

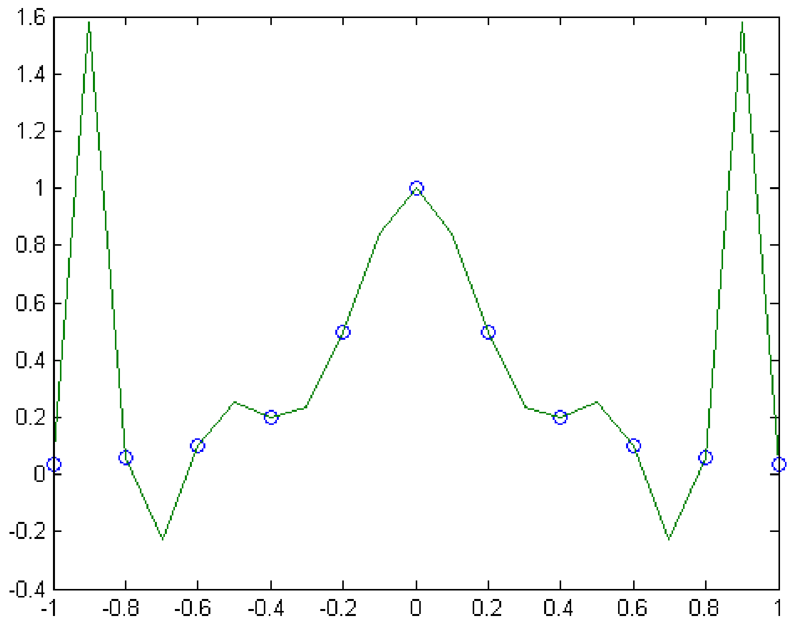

Figure 1: Runge's phenomenon of Polynomial interpolation with 2N points.

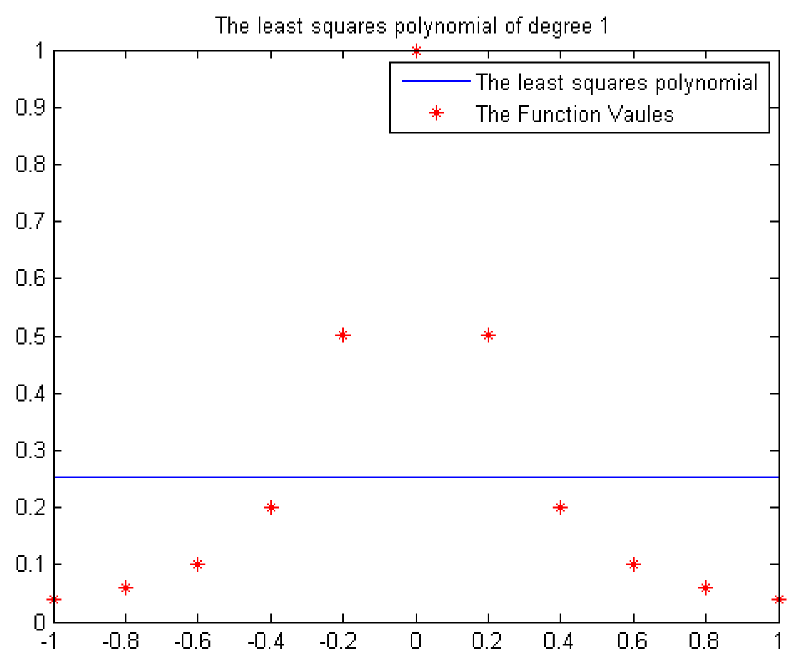

Figure 2: Least Square polynomial of degree=1, N=10.

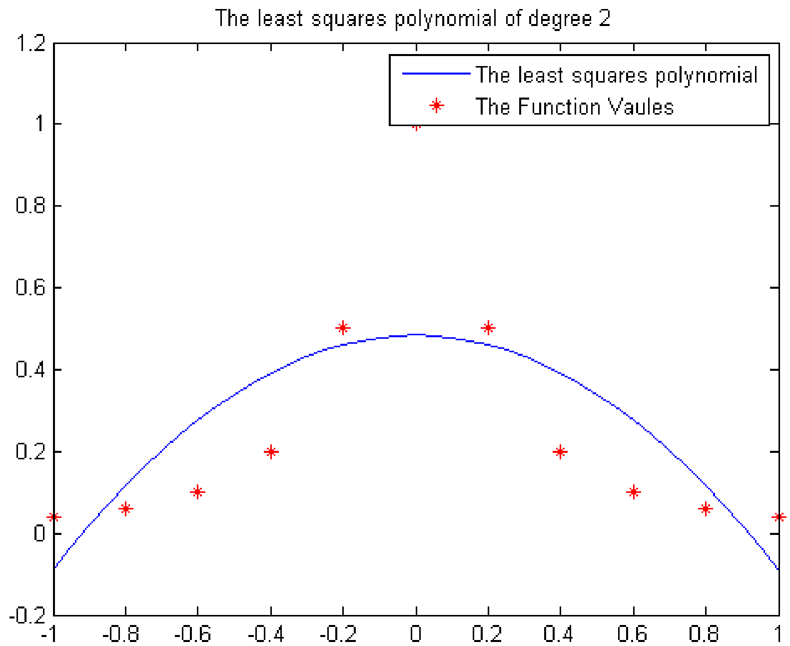

Figure 3: Least Square polynomial of degree=2, N=10.

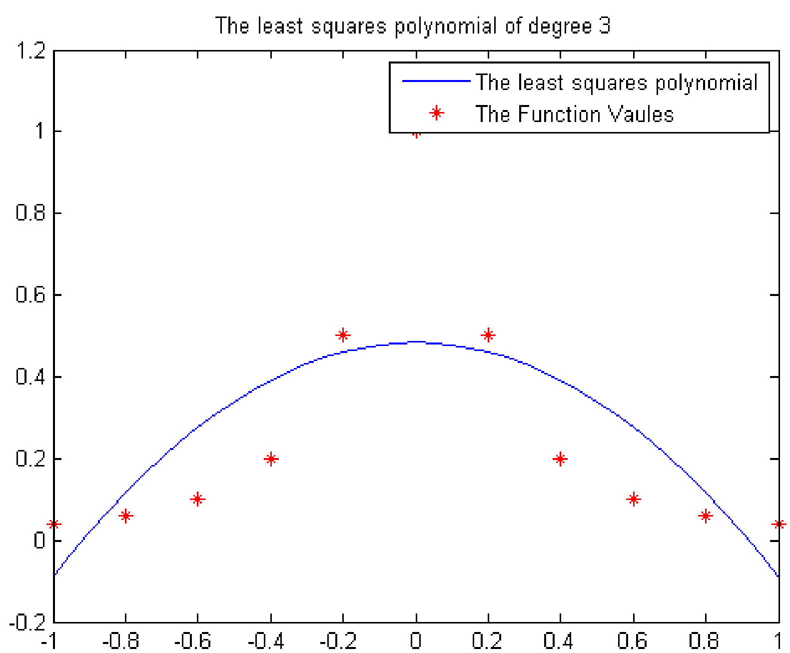

Figure 4: Least Square polynomial of degree=3, N=10.

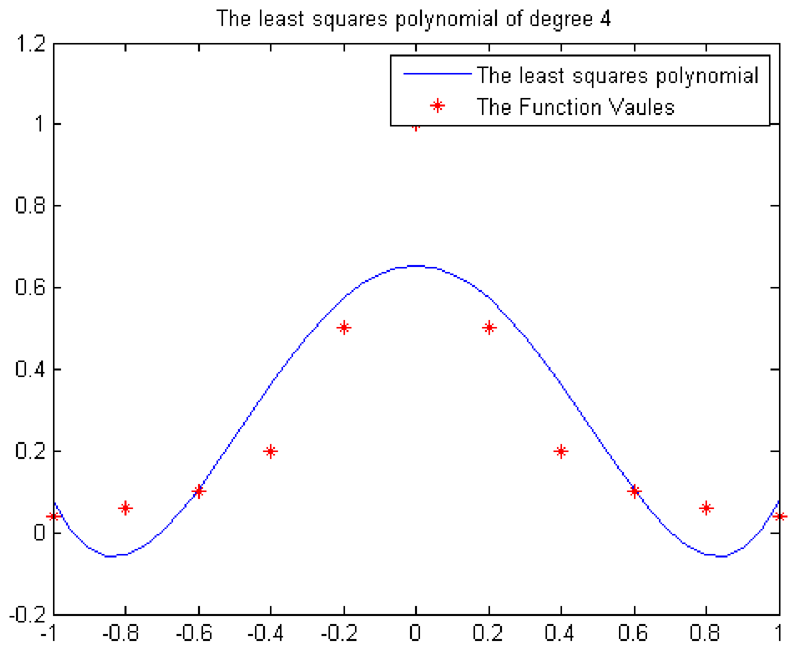

Figure 5: Least Square polynomial of degree=4, N=10.

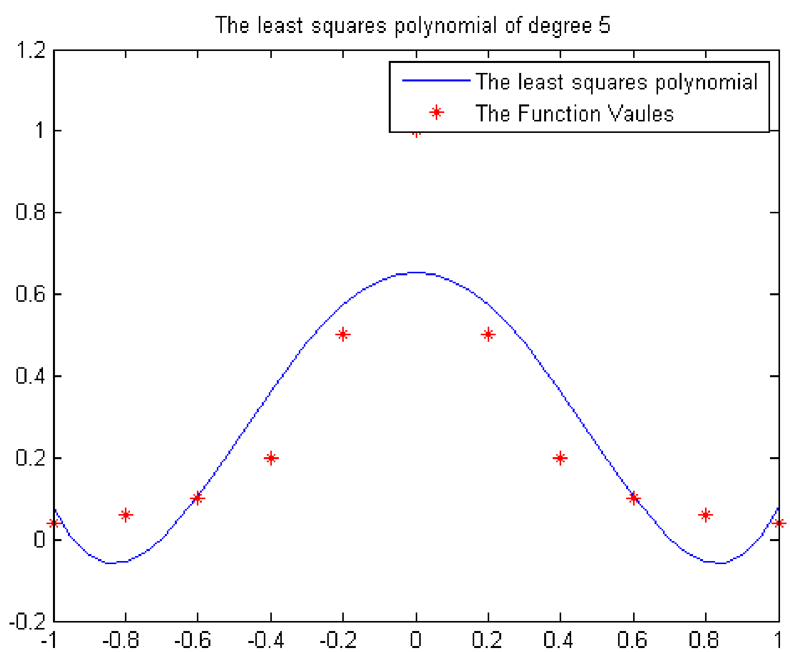

Figure 6: Least Square polynomial of degree=5, N=10.

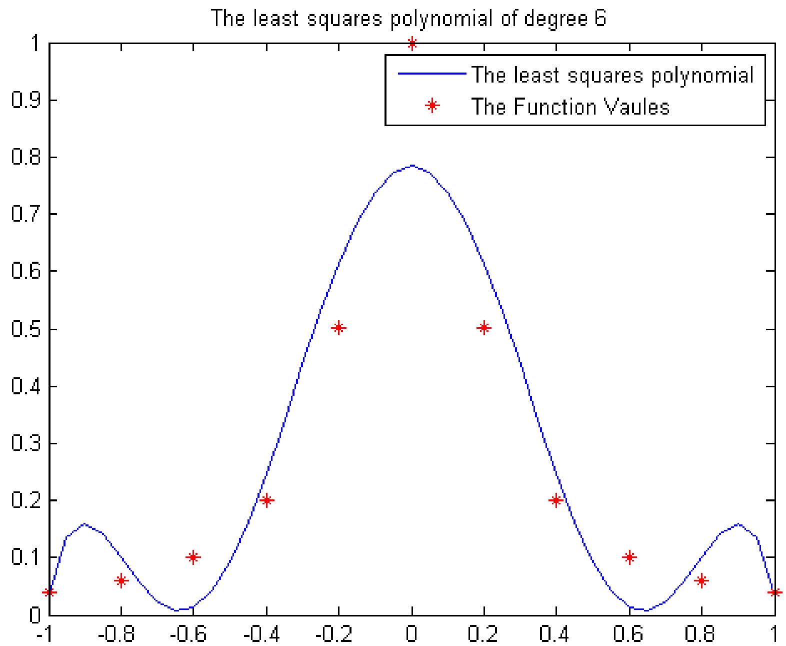

Figure 7: Least Square polynomial of degree=6, N=10.

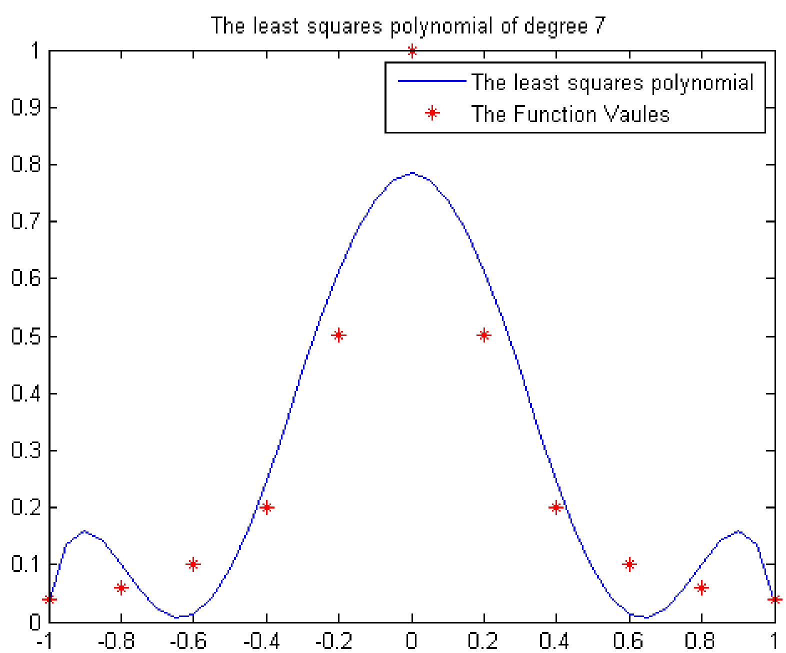

Figure 8: Least Square polynomial of degree=7, N=10.

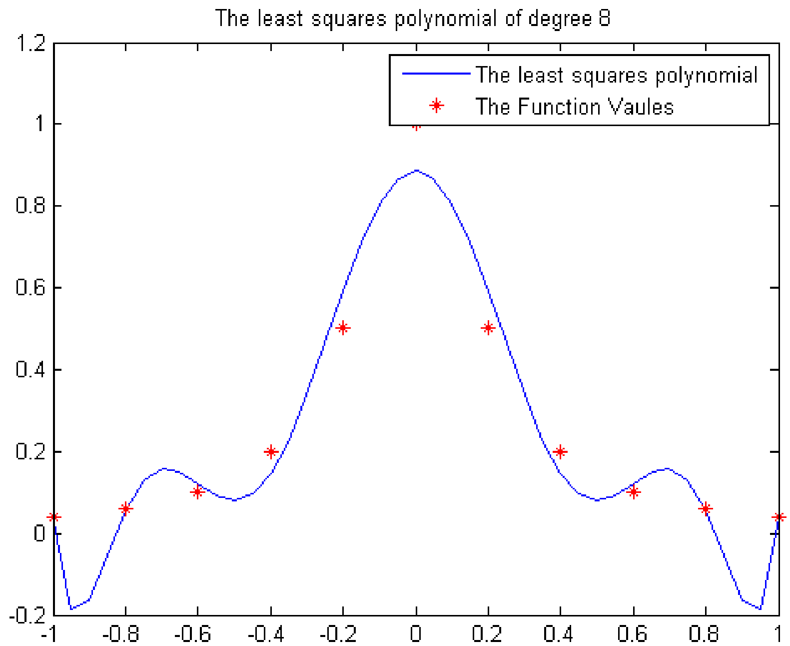

Figure 9: Least Square polynomial of degree=8, N=10.

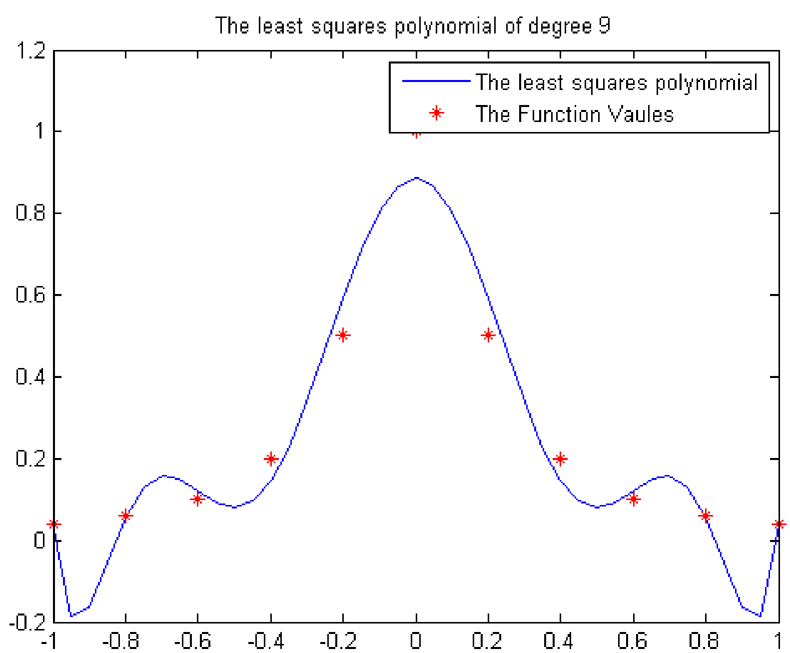

Figure 10: Least Square polynomial of degree=9, N=10.

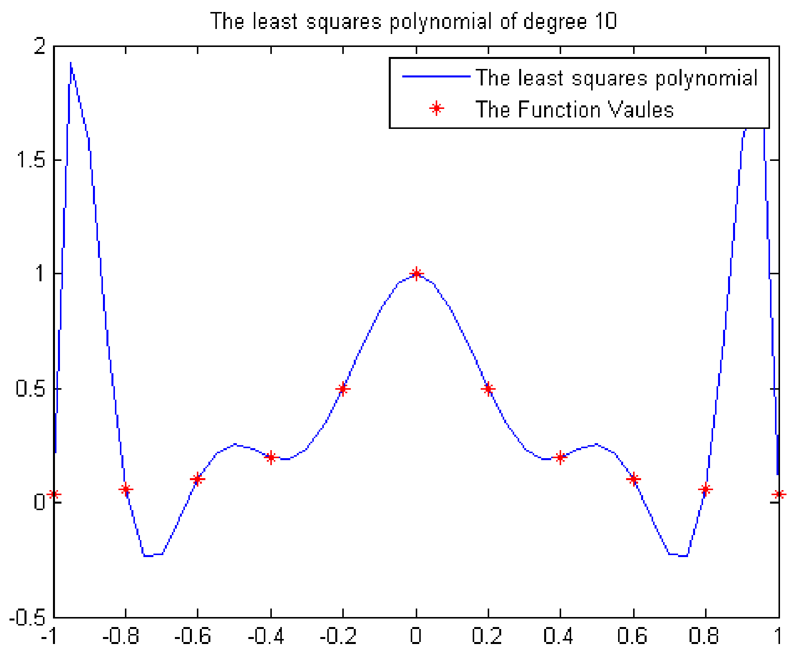

Figure 11: Least Square polynomial of degree=10, N=10.