1 Question 1

** All codes are wriittern and compiled by Matlab R2017b.*

1.1

Function for solving an $n \times n$ tridiagonal matrix.

```
%Solving linear system Ax=(abc)x=d;
function x = tridisolve(a,b,c,d)
x = d;
n = length(x);

for j = 1:n-1
mu = a(j)/b(j);
b(j+1) = b(j+1) - mu*c(j);
x(j+1) = x(j+1) - mu*x(j);
end

x(n) = x(n)/b(n);
for j = n-1:-1:1
x(j) = (x(j)-c(j)*x(j+1))/b(j);
end

end
```

Where

$$A = \begin{pmatrix} b_1 & c_1 & & & & \\ a_1 & b_2 & c_2 & & & \\ & a_2 & b_3 & c_4 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{pmatrix} \tag{1}$$

and $a = (a_1, \ldots, a_{n-1})$, $b = (b_1, \ldots, b_n)$, $c = (c_1, \ldots, c_{n-1})$

1.2

Function for Computing the coefficients $\{c_i\}_{i=-1}^{n+1}$ of the natural C^2-Cubic B-spline $q_3^n(x)$ that interpolates f

```
%Function f
function F=func(x)

F=exp(-x).*cos(6*pi*x);

end
```

where F would be a vector such that $F(x) = (F(x_1), \ldots, F(x_d)), x_i \in R$.

```
    %Computing coefficients
    function C= Bspline(f)
    n=length(f);
    trid=@tridisolve;
    a=zeros(1,n-3);
    b=zeros(1,n-2);
    c=zeros(1,n-3);
    d=zeros(1,n-2);
    a(1:n-3)=1;
    b(1:n-2)=4;
    c(1:n-3)=1;

    d(1)=f(2)-f(1)/6;%f1-f0/6
    d(2:n-3)=f(3:n-2);
    d(n-2)=f(n-1)-f(n)/6;

    C=zeros(1,n+2);
    C(3:n)=trid(a,b,c,d);
    C(2)=f(1)/6; %C0
    C(1)=2*C(2)-C(3); %first-derivative conditions,C-1
    C(n+1)=f(n)/6;
    C(n+2)=2*C(n+1)-C(n);
    end
```

1.3

Function for computing values of the spline $q_3^n(x)$

```
function [xhat,q]= evaluate(Coef,xnode,a,b)
B=@splinefunc;
n=length(xnode);%index of xi is from 0 to n-1
h=xnode(2)-xnode(1);
distance=(b-a)/(20*n-20);%20*(n-1)+1 point with same distances

xhat=zeros(1,20*n-19);
q=zeros(1,20*n-19);
xhat(1)=xnode(1);
xhat(end)=xnode(end);
q(1)=Coef(1)+4*Coef(2)+Coef(3); %condition for q(x) in node points
q(end)=Coef(end-2)+4*Coef(end-1)+Coef(end);

for i=2:20*n-20
xhat(i)=xhat(i-1)+distance;
k=floor((xhat(i)-xnode(1))/h)+3; %location of xhat(i)=k-2
if (k<4)
%B-1(x)=B((x-x0+h)/h)
q(i)=Coef(1)*B((xhat(i)-xnode(1)+h)/h)+Coef(2)*B((xhat(i)-xnode(1))/h)
+Coef(3)*B((xhat(i)-xnode(2))/h)+Coef(4)*B((xhat(i)-xnode(3))/h);
elseif(k>n)
%B(n+1)(x)=B((x-xn-h)/h)
q(i)=Coef(k-2)*B((xhat(i)-xnode(k-3))/h)+Coef(k-1)*B((xhat(i)-xnode(k-2))/h)
+Coef(k)*B((xhat(i)-xnode(k-1))/h)+Coef(k+1)*B((xhat(i)-xnode(k-1)-h)/h);
else
q(i)=Coef(k-2)*B((xhat(i)-xnode(k-3))/h)+Coef(k-1)*B((xhat(i)-xnode(k-2))/h)
+Coef(k)*B((xhat(i)-xnode(k-1))/h)+Coef(k+1)*B((xhat(i)-xnode(k))/h);
end
end
```

where B is one of the spline function

$$B(x)=\begin{cases}0, & x\le -2\\ (x+2)^3, & -2\le x\le -1\\ 1+3(x+1)+3(x+1)^2-3(x+1)^3, & -1\le x\le 0\\ 1+3(1-x)+3(1-x)^2-3(1-x)^3, & 0\le x\le 1\\ (2-x)^3, & 1\le x\le 2\\ 0, & 2\le x\end{cases} \tag{2}$$

```
%Spline function B
function B=splinefunc(x)
if (x<=-2)
B=0;
elseif (-2<=x && x<=-1)
B=(x+2)^3;
elseif (-1<=x && x<=0)
B=1+3*(x+1)+3*(x+1)^2-3*(x+1)^3;
elseif (0<=x && x<=1)
B=1+3*(1-x)+3*(1-x)^2-3*(1-x)^3;
elseif (1<=x && x<=2)
B=(2-x)^3;
else
B=0;
end
end
```

1.4

Table 1 illustrates $\|f-q_3^n\|_{\infty,[-1,1]}$ with different n and Figure 1 shows $\log_{10}\left(\|f-q_3^n\|_{\infty,[-1,1]}\right)$ against $\log_{10}(n)$.

	n=16	n=32	n=64	n=128
$\|f-q_3^n\|_{\infty,[-1,1]}$	1.087447370563231	0.216700752320041	0.048126171304783	0.011648941831482

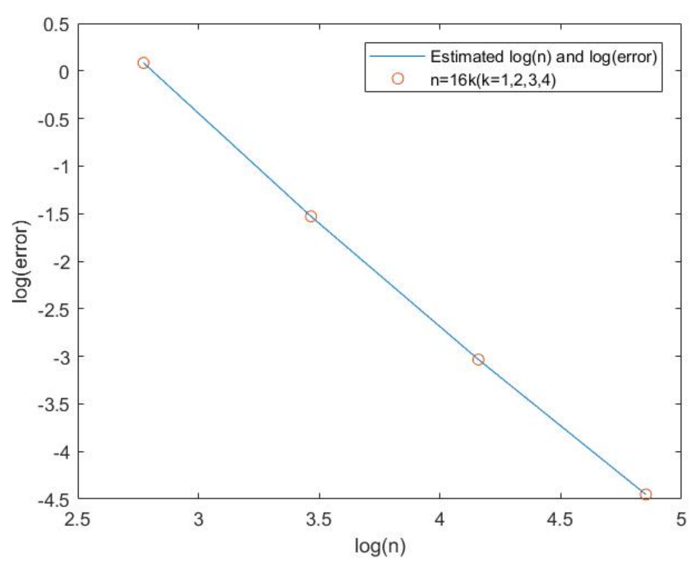

Figure 1: log(n) and log(error)

As Figure 1 shows, the relationship between $\log_{10}\left(\|f-q_3^n\|_{\infty,[-1,1]}\right)$ and $\log_{10}(n)$ is almost linear. Moreover, we will get a more accurate solution when test multiple n by using OLS regression. By testing $n = 16, 18, 20, \cdots, 128$, we get the relationship between actual rate of convergence(log(error)) and log(n).

$$\log_{10}\left(\|f-q_3^n\|_{\infty,[-1,1]}\right) = -2.158\log_{10}(n) + 5.972 \tag{3}$$

Theoretically, $\log_{10}\left(\|f-q_3^n\|_{\infty,[-1,1]}\right)$satisfies

$$\log_{10}(\|f-q\|_{\infty}) \le \log_{10}(\frac{5}{384}h^4\left\|f^{(4)}\right\|_{\infty}) = \log_{10}(\frac{5}{384}\frac{(b-a)^4}{n^4}\left\|f^{(4)}\right\|_{\infty}) = -4\log_{10}(n) + logC \tag{4}$$

where C is a constant number.
Figure 2,3,4 and 5 show the trajectories of $q_3^n(x)$ with n=16,32,64,128 respectively.

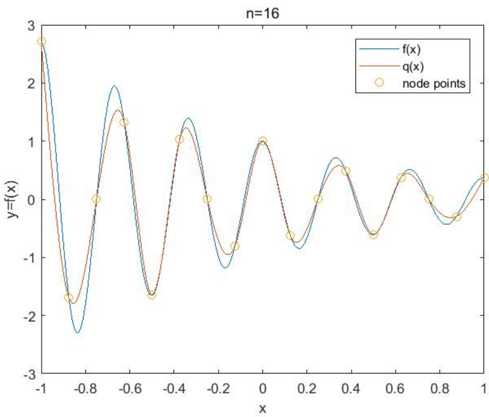

Figure 2: n=16

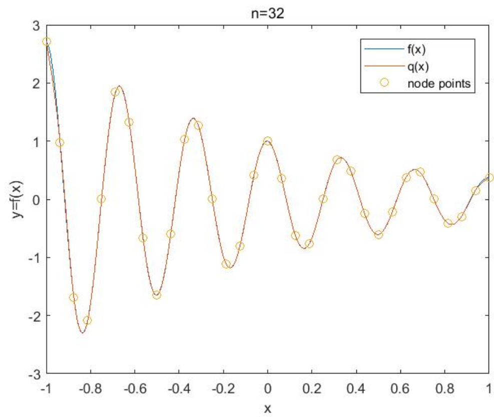

Figure 3: n=32

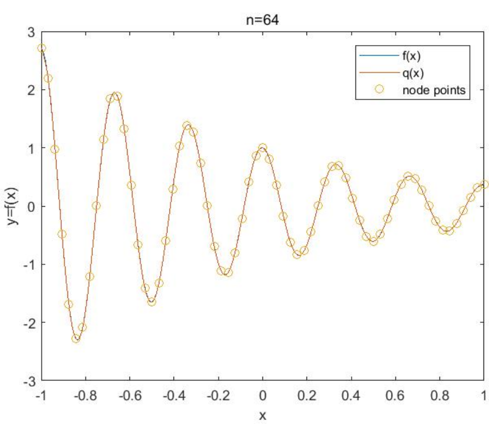

Figure 4: n=64

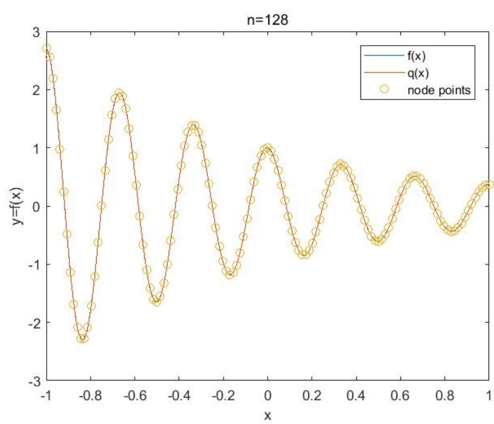

Figure 5: n=128

2 Question 2

2.1

Function for integrating a function using a Gauss–Legendre quadrature with n = 10

```
%function to integrate a function using a -GaussLegendre quadrature
function G=Gauss(f,n) %f is a vector valued function with dimension n

G=zeros(n,1);
%first column are weights and second one are quadrature points
Gaussnode=[0.2955242247147529,  -0.1488743389816312;
0.2955242247147529, 0.1488743389816312;
0.2692667193099963, -0.4333953941292472;
0.2692667193099963, 0.4333953941292472;
0.2190863625159820, -0.6794095682990244;
0.2190863625159820, 0.6794095682990244;
0.1494513491505806, -0.8650633666889845;
0.1494513491505806, 0.8650633666889845;
0.0666713443086881, -0.9739065285171717;
0.0666713443086881, 0.9739065285171717];

G=f(Gaussnode(:,2)',n)*Gaussnode(:,1);
end
```

We take testgaussfunc as example(note that n=1)

```
%Vector-valued function to do Gauss integrate
function F=testguassfunc(x,n)
dimension=length(x); %Dimension of vector x

F=zeros(n,dimension);
if n==1
F(1,:)=exp(-x.^2);
end
```

Then we get the approximation

$$\int_{-1}^{1} e^{-x^2}\,\mathrm{d}x \approx 1.493648265624351 \tag{5}$$

And the error is

$$err = |\sqrt{\pi}\,\mathrm{erf}(1) - 1.493648265624351| = 5.033751193650460e-13 \tag{6}$$

2.2

Function that will compute the LSQ coefficients for a given f and n

```
%Compute  the  LSQ  coefficients
function C=Lsqcoef(Func,n)
G=@Gauss;
C=G(Func,n);%Compute inner product(f(x),Pi(x))
for i=1:n+1
C(i)=(2*i-1)*C(i)/2; %inner product(Pk(x),Pk(x)=2/(2k+1)
end
```

Where Func is Vector-valued function such that $Func(x) = f(x)(P_1(x), \cdots, P_n(x)^T$ where P(x) is Legendre polynomials and inner product is computing by Gauss-integrate which is introduced in 2.1.

```
%F is vector-valued function that F(i)=f(x)*Pi(x) where Pi(x) is Legendre
function F=Func(x,n)
P=@Legendre;
f=@funct;
dimension=length(x);%length of vector x
F=zeros(n+1,dimension);
z=P(x,n);
for i=1:n+1
F(i,:)=f(x).*z(i,:);%F(i,:)=(Fi(x0),Fi(x1),...,Fi(xd))
end
end
```

And Legendre polynomial

```
%Vector-valued function that P(i)=Pi(x) where Pi(x) is Legendre
function P=Legendre(x,n)
dimension=length(x);
P=zeros(n+1,dimension);
P(1,:)=1;
P(2,:)=x;
for i=3:n+1
%P(i,:)=(Pi(x0),Pi(x1),...,Pi(xd))
P(i,:)=(2*i-3)*x.*P(i-1,:)/(i-1)-(i-2)*P(i-2,:)/(i-1);
end
end
```

Actually, we shall get a matrix with given vector x in both codes above, which means

$$P(x) = \begin{pmatrix} P_0(x_0) & P_0(x_1) & \dots & P_0(x_d) \\ P_1(x_0) & P_1(x_1) & \dots & P_1(x_d) \\ \dots & & \dots & \\ P_n(x_0) & P_n(x_1) & \dots & P_n(x_d) \end{pmatrix} \tag{7}$$

and

$$F(x) = \begin{pmatrix} f(x_0)P_0(x_0) & f(x_1)P_0(x_1) & \dots & f(x_d)P_0(x_d) \\ f(x_0)P_1(x_0) & f(x_1)P_1(x_1) & \dots & f(x_d)P_1(x_d) \\ \dots & & \dots & \\ f(x_0)P_n(x_0) & f(x_1)P_n(x_1) & \dots & f(x_d)P_n(x_d) \end{pmatrix} \tag{8}$$

Then we can easily get LSQ approximation with given vector a

$$LSQ(a) = C * P(a) = (C_0, C_1, \cdots, C_n) * P(a) \tag{9}$$

2.3

Test the code for $f(x) = e^{-x}\cos(\pi x)$

```
%Function to test the LSQ
function f=funct(x)
f=exp(-x).*cos(pi*x);
end
```

Figure 6,7 and 8 shows the results with different n=3,5,7 respectively.

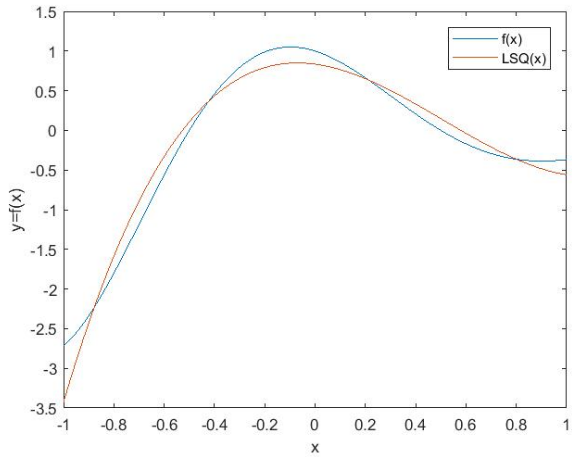

Figure 6: n=3

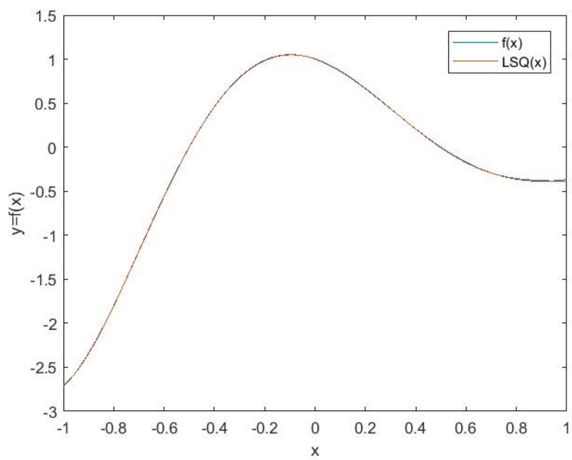

Figure 7: n=5

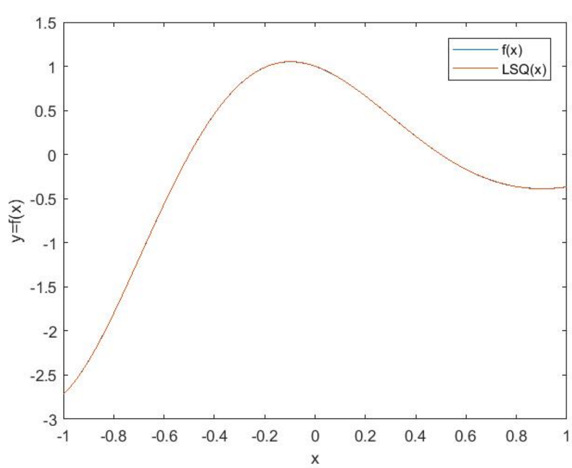

Figure 8: n=7

3 Question 3

3.1

Function to compute the Jacobian of a smooth vector-valued function F.

```
function J=Jaccobi(Func,x,epsilon)
n=length(x);
J=zeros(n,n);
deltax=zeros(1,n);
for i=1:n
deltax(i)=epsilon; %delatax=x+epsilon*ej
J(:,i)=((Func(x+deltax)-Func(x))/epsilon)';
deltax(i)=0;
end
end
```

3.2

Function to solve the k-dimension nonlinear system of equations using Newton's method to get $(n-1)^{th}$ iteration.

```
function X=Newton(n,Func,x0,epsilon)
J=@Jaccobi;
k=length(x0);
Xs=zeros(n,k);
Xs(1,:)=x0;
for i=2:n
Xs(i,:)=Xs(i-1,:)+(J(Func,Xs(i-1,:),epsilon)\(-Func(Xs(i-1,:))))';
end
X=Xs(end,:); %(n-1)th iteration
end
```

Test the code with a 2-dimension function.

```
function F=Func(x) %x is a two-dimension vector
F=zeros(2,1);
F(1)=x(1)^2-x(2);
F(2)=x(1)^2+x(2)^2-2;
end
```

Table 2 shows the fifth iteration $\left(x_1^{(5)}, x_2^{(5)}\right)$ with different $\mathcal{E}$.

	$\mathcal{E}$=0.005	$\mathcal{E}$=0.05	$\mathcal{E}$=0.5
$x_1^{(5)}$	1.00000	1.000001554936712	1.001443216203944
$x_2^{(5)}$	1.00000	1.000000088939143	1.000164972604241