3

(a) Input code:

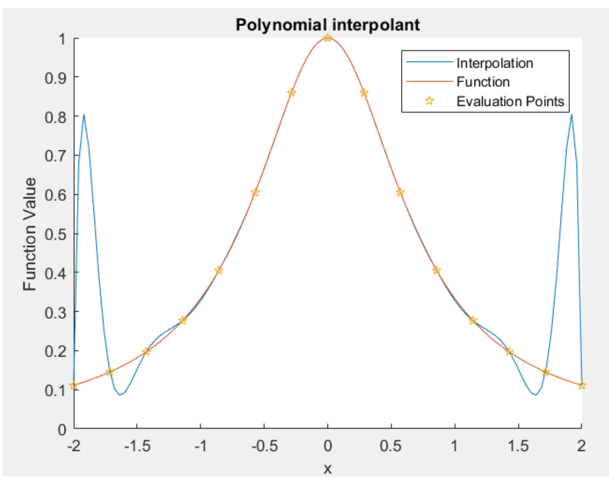
```
>> a = -2;
b = 2;
m = 101;
n = 15;
xd = zeros(1,n);

for i = 1:n
    xd(i) = a+(i-1)*(b-a)/(n-1);
    yd(i) = 1/(1+2*xd(i)^2);
end
clear i

for i = 1:m
    x(i) = a + (b-a)*(i-1)/(m-1);
    Fx(i) = 1/(1+2*x(i)^2);
    Poly(i) = polyInterp(xd,yd,x(i));
    Error(i) = Fx(i)-Poly(i);
end

figure
hold on
plot(x, Poly, '-');
plot(x,Fx, '-');
plot(xd,yd, 'p');
legend('Interpolation', 'Function','Evaluation Points','Location','NorthEast');
title('Polynomial interpolant');
xlabel('x')
ylabel('Function Value')

figure
plot(x, Error_a)
hold on
legend("error")
title('Error in Interpolation')
xlabel('X Value')
ylabel('Error Value')
```

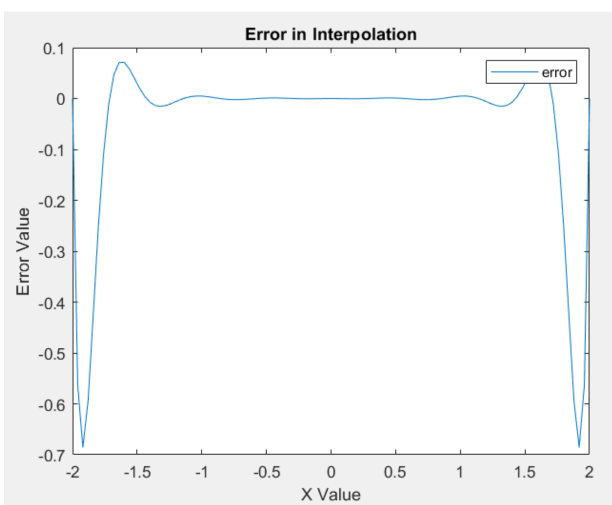
Function code:

```
function y = polyInterp( xd,yd,x )
    if length(xd) == length(yd)
        d = length(xd);
    end
    C = newtdd(xd,yd);
    y = nest(d-1,C,x,xd);
end
```

```
function c = newtdd( xd, yd)
    n1 = length(xd);
    n2 = length(yd);
    V = zeros(n1,n1);
    for i =1:n1
        V(i,1) = yd(i);
    end

    for i = 2:n1
        for j = 1:n1+1-i
            V(j,i) = (V(j+1,i-1)-V(j,i-1))/(xd(j+i-1)-xd(j));
        end
    end

    for i = 1:n1
        c(i) = V(1,i);
    end
end
```

Graph:

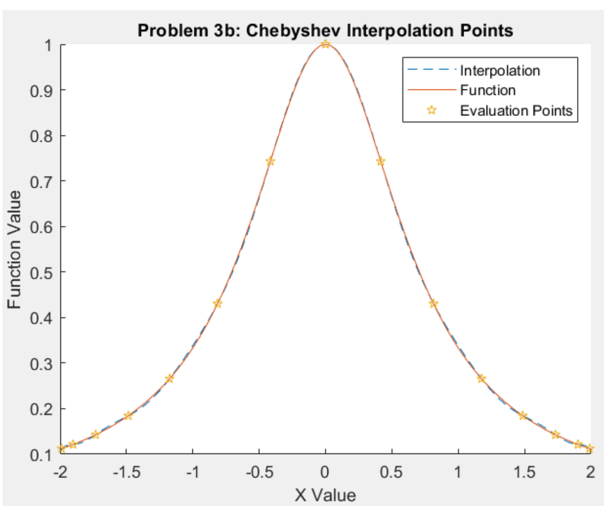


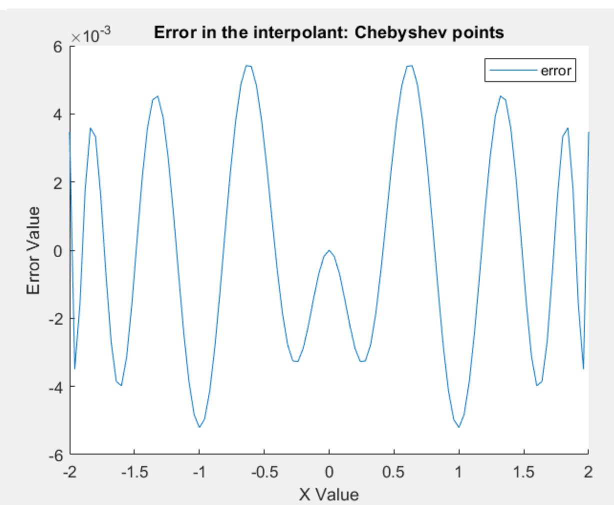


(b) Use previous code to test non-equally spaced Chebyshew points
Input code:

```
a = -2;
b = 2;
n = 15;
m = 101;
xd = zeros(1,n);

for i = 1:n
    xd(i) = (b-a)/2*cos((2*i-1)*pi/2/n)+(b+a)/2;
    yd(i) = 1/(1+2*xd(i)^2);
end
clear i

for i = 1:m
    x(i) = a+(i-1)*(b-a)/(m-1);
    Func(i) = 1/(1+2*x(i)^2);
    Poly(i) = polyInterp(xd,yd,x(i));
    Error(i) = Func(i)-Poly(i);
end
```

Graph:



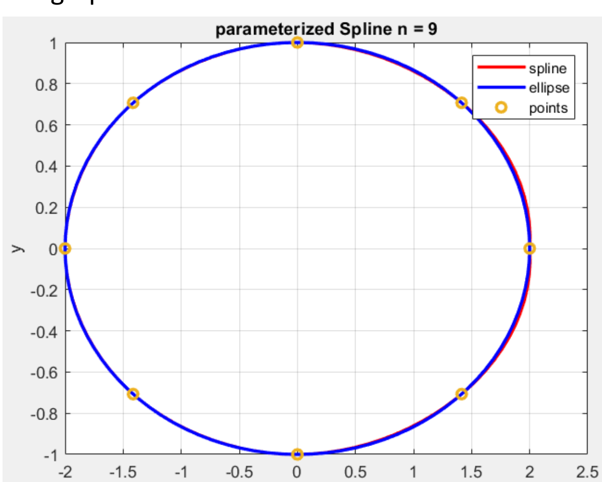


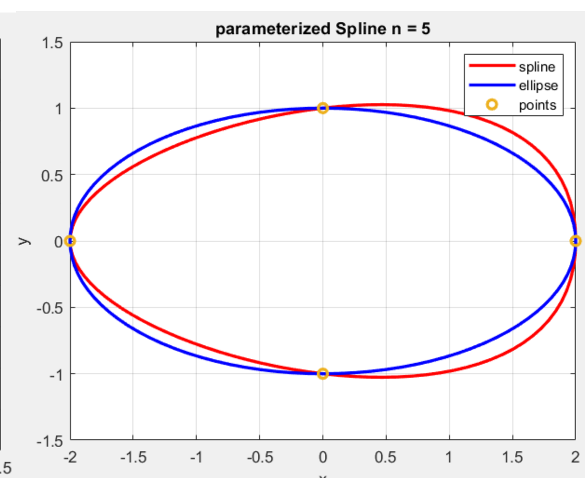
4.

(c) code for graphing:

```
clear x
a = 2;
b = 1;
m = 101;

for n = [5,9]
    theta = linspace(0, pi* 2, n);
    phi = linspace(0,2*pi,m);
    xx = 2*cos(theta);
    yy = sin(theta);
    ppx = spline(theta, xx);
    ppy = spline(theta, yy);
    x = ppval(ppx, phi);
    y = ppval(ppy, phi);
    xe=a*cos(phi);
    ye=b*sin(phi);

    plot(x,y,'r-', xe,ye,'b-', xx,yy,'o', 'LineWidth',2 );
    title(sprintf('parameterized Spline n = %d',n));
    xlabel('x'); ylabel('y');
    legend('spline','ellipse','points');
    grid on;
    print('-depsc2',sprintf('splineEllipsen%d.eps',n));
end
```

graph:





5

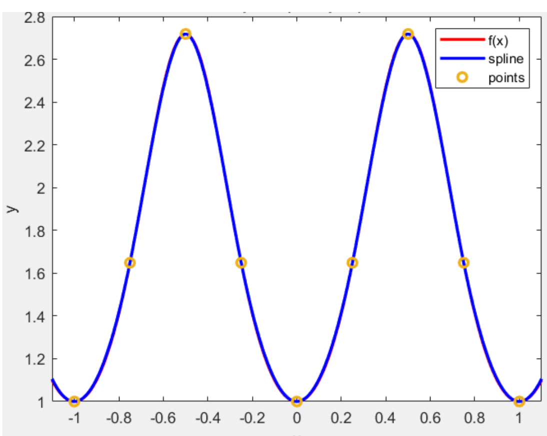
Code for coeff, code is written based on textbook:

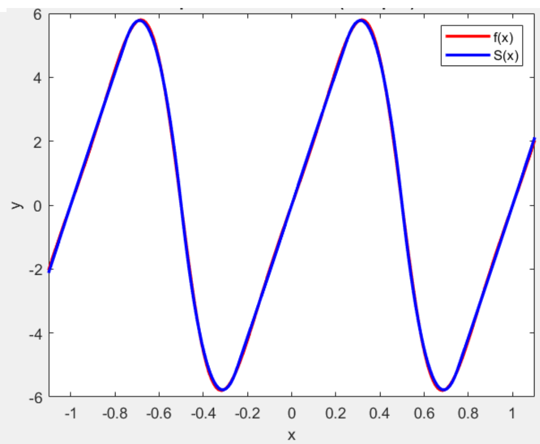
```
function coeff = splineCoeff(xs,ys, bcLeft,gLeft, bcRight,gRight)
    n = length(xs);
    A = sparse(n,n);
    r = zeros(n,1);
    for i=1:n-1
        dx(i)= xs(i+1)-xs(i);
        dy(i)= ys(i+1)-ys(i);
        if i >= 2
            A(i,i-1:i+1)= [dx(i-1), 2*(dx(i-1) + dx(i)), dx(i)];
            r(i)=3.*(dy(i)/dx(i)- dy(i-1)/dx(i-1));
        end
    end

    %gRight not used
    %gLeft not used
    if(bcLeft == 0)
        A(1,1)=1;
    end
    if(bcRight == 0)
        A(n,n)=1;
    end

    %S'(b)=gRight
    %S'(a)=gLeft
    if(bcLeft == 1)
        A(1,1:2) = [2*dx(1), dx(1)];
        r(1) = 3.*(dy(1)/dx(1) - gLeft);
    end
    if(bcRight == 1)
        A(n,n-1:n) = [dx(n-1), 2 * dx(n-1)];
        r(n) = 3.*(gRight - dy(n-1)/dx(n-1));
    end

    %S''(b)=gRight
    %S''(a)=gLeft
    if(bcLeft == 2)
        A(1,1) = 2;
        r(1) = gLeft;
    end
    if(bcRight == 2)
        A(n,n) = 2;
        r(n) = gRight;
    end

    %Calculate coeff base on dx dy and r
    coeff = zeros(n,3);
    coeff(:,2) = A\r;
    for i = 1:n-1
        coeff(i,3) = (coeff(i+1,2) - coeff(i,2))/(3.*dx(i));
        coeff(i,1) = dy(i)/dx(i)-dx(i)*(2.*coeff(i,2)+coeff(i+1,2))/3.;
    end
    coeff = coeff(1:n-1,1:3);
```

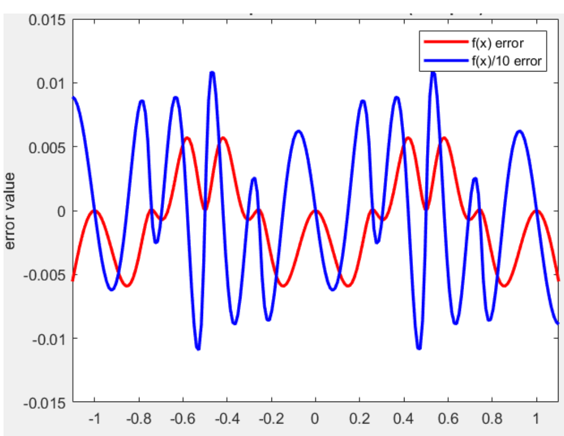
Function splineEval:

```
function [y,yx ] = splineEval(xs,ys,coeff,x)
    for i = 1:length(x)
        for j = 1:length(xs)-1
            a = j;
            %If x(i) > xs(n)then evaluate the last spline, Sn-1(x(i))
            if(xs(j+1) >= x(i))
                break;
            end
        end
        dx = x(i) - xs(a);
        y(i) = ys(a) + dx*(coeff(a,1) + dx*( coeff(a,2) + dx*coeff(a,3)));
        yx(i) = coeff(a,1) + dx*( 2.*coeff(a,2) + 3.*dx*coeff(a,3));
    end
end
```

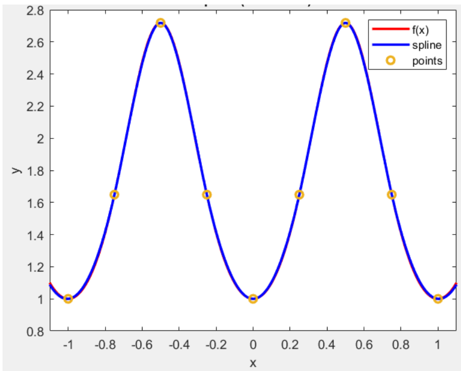
(a)

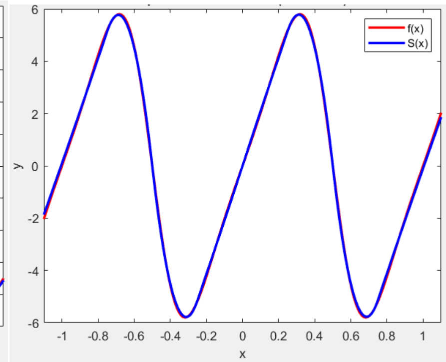


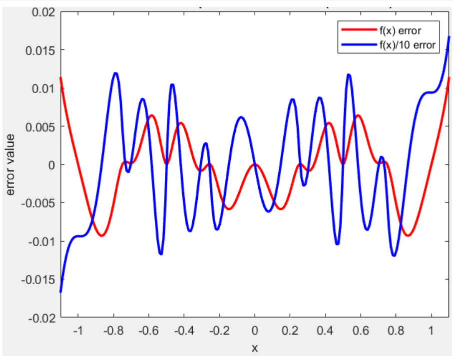




(b) Curvature boundary conditions with gLeft = f''(a) and gRight = f''(b).







(c) Natural boundary conditions.