Homework 10

Math 3607, Autumn 2021

Marco LoPiccolo

Table of Contents

Problem 1.)

1. (Polynomial vs. piecewise polynomial interpolation; **FNC** 5.1.2) The following table gives the life expectancy in the U.S. by year of birth:

year	1980	1985	1990	1995	2000	2005	2010
expectancy	73.7	74.7	75.4	75.8	77.0	77.8	78.7

(a) Defining "year since 1980" as the independent variable, use `polyfit` to construct and plot the polynomial interpolant of the data.

(b) Use `interp1` to construct and plot a piecewise cubic interpolant (use `'spline'` option) of the data.

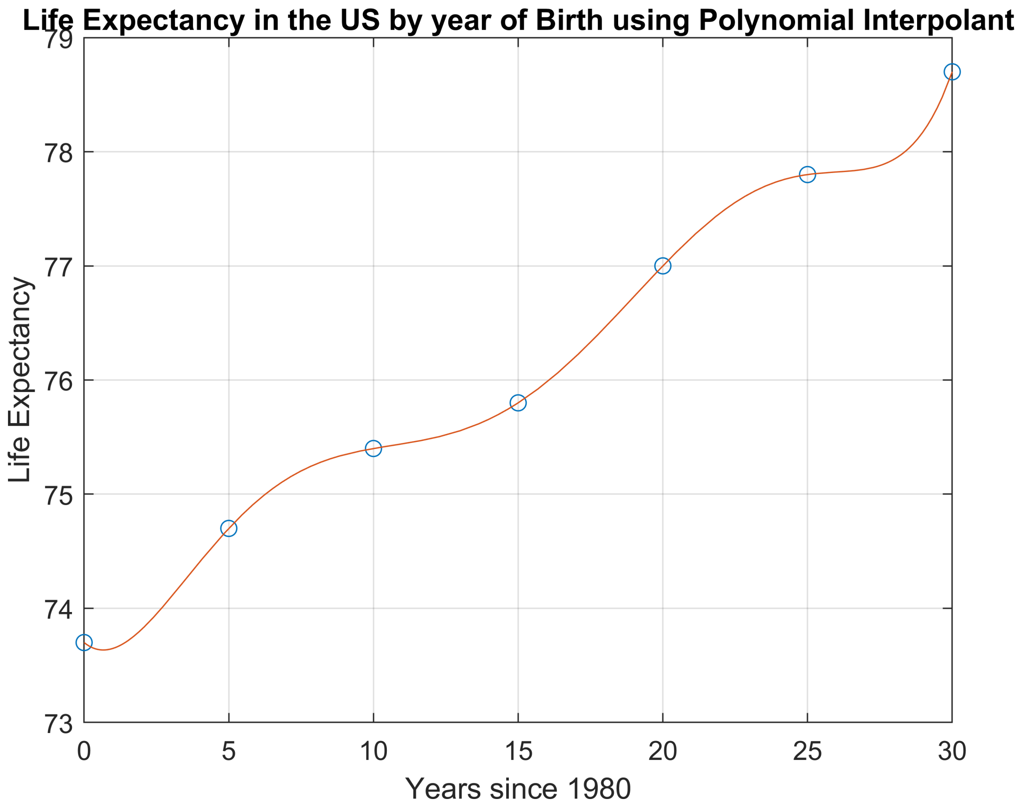
(c) Use both methods to estimate the life expectancy for a person born in 2007. Which value is more believable?

Part a.)

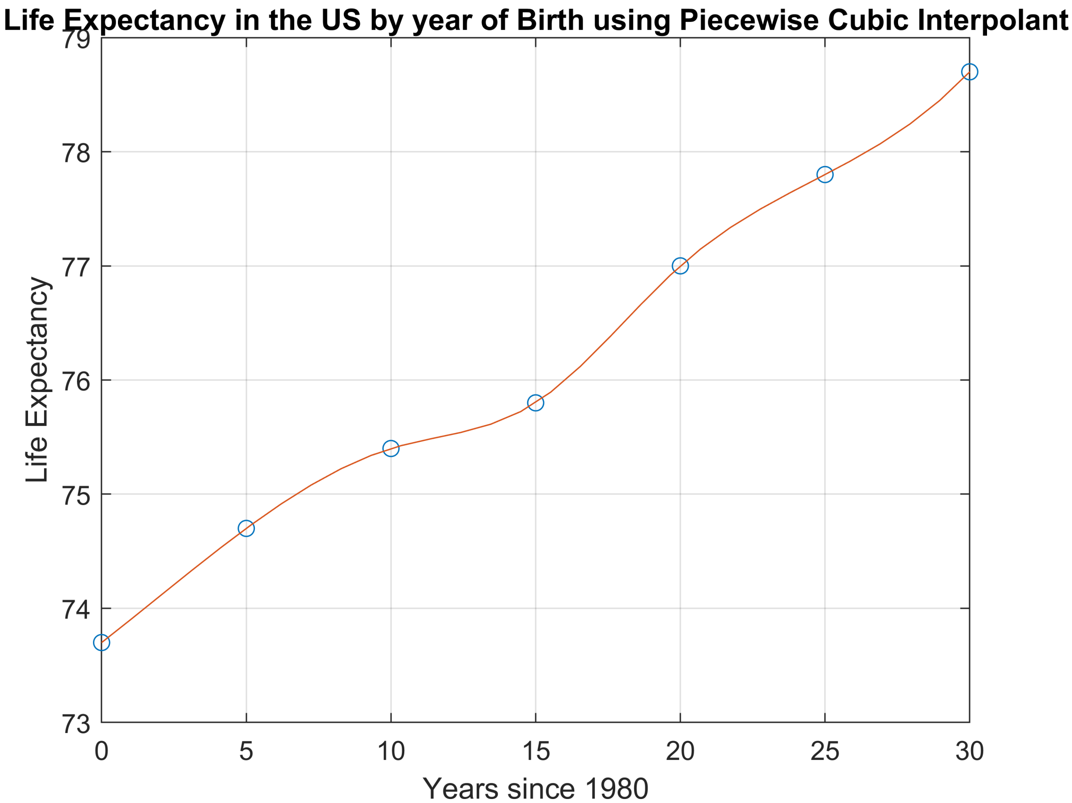
```
xdp = [1980 1985 1990 1995 2000 2005 2010]-1980;
ydp = [73.7 74.7 75.4 75.8 77.0 77.8 78.7];
c = polyfit(xdp, ydp, length(xdp)-1);
p = @(x) polyval(c, x);

clf
plot(xdp, ydp, 'o')
grid on, hold on
fplot(p, [1980 2010]-1980)
title('Life Expectancy in the US by year of Birth using Polynomial Interpolant')
xlabel('Years since 1980')
ylabel('Life Expectancy')
```

Part b.)

```
hold off
```




```
clf
x = linspace(1980,2010,30)-1980;
plot(xdp, ydp, 'o')
grid on, hold on
plot(x, interp1(xdp, ydp, x, 'spline'))
title('Life Expectancy in the US by year of Birth using Piecewise Cubic Interpolant')
xlabel('Years since 1980')
ylabel('Life Expectancy')
```



Part c.)

```
Polyat2007 = p(27)
```

```
Polyat2007 = 77.8475
```

```
Cubicat2007 = interp1(xdp, ydp, 27, 'spline')
```

```
Cubicat2007 = 78.0842
```

The data from the cubic polynomial is more believable at it follows a much smoother graph that has less dips while the polynomial interpolant is much less stable as you get closer to the endpoints of the graph from Runge's phenomenon which is why there are weird oscilations around the endpoints. Because of this we know that data is a little less reliable versus the cubic interpolant that has a much smoother graph with less oscilations.

Problem 2.)

2. (Piecewise cubic interpolation; **FNC** 5.1.3) The following two point sets define the top and bottom of a flying saucer shape:

Top:

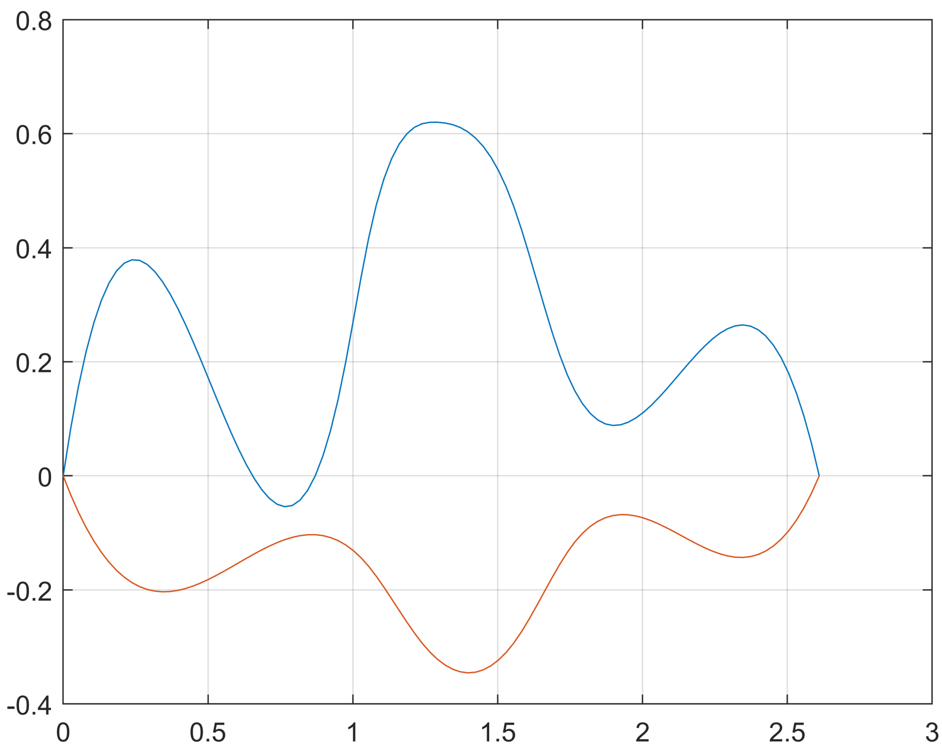
x	0	0.51	0.96	1.06	1.29	1.55	1.73	2.13	2.61
y	0	0.16	0.16	0.43	0.62	0.48	0.19	0.18	0

Bottom:

x	0	0.58	1.04	1.25	1.56	1.76	2.19	2.61
y	0	-0.16	-0.15	-0.30	-0.29	-0.12	-0.12	0

Use piecewise cubic interpolation to make a picture of the flying saucer.

```
xdptop = [0 0.51 0.96 1.06 1.29 1.55 1.73 2.13 2.61];
ydptop = [0 0.16 0.16 0.43 0.62 0.48 0.19 0.18 0];
x = linspace(0, 2.61, 100);

xdpbottom = [0 0.58 1.04 1.25 1.56 1.76 2.19 2.61];
ydpbottom = [0 -0.16 -0.15 -0.30 -0.29 -0.12 -0.12 0];

clf
plot(x, interp1(xdptop, ydptop, x, 'spline'))
grid on, hold on
plot(x, interp1(xdpbottom, ydpbottom, x, 'spline'))
```



Problem 3.)

3. (Quadratic interpolant by hand; **FNC** 5.1.4) Define

$$q(x) = \frac{a}{2}x(x-1) - b(x-1)(x+1) + \frac{c}{2}x(x+1).$$

(a) Show that q is a polynomial interpolant of the points $(-1, a)$, $(0, b)$, $(1, c)$.

(b) Use a change of variable to find a quadratic polynomial interpolant p for the points $(x_0 - h, a)$, (x_0, b), $(x_0 + h, c)$.

3.) $q(x) = \frac{a}{2}x(x-1) - b(x-1)(x+1) + \frac{c}{2}x(x+1)$

a.) Show that q is a polynomial interpolant of the points $(-1, a)$, $(0, b)$, $(1, c)$

$$q(-1) = \frac{a}{2}\,{-1}(-1-1) - b(-1-1)(-1+1) + \frac{c}{2}{-1}(-1+1)$$

$$= \frac{a}{2} \cdot 2 - b(-2 \cdot 0) + \frac{c}{2}(0)$$

$$= \boxed{a}$$

$$q(0) = \frac{a}{2}\,0(0-1) - b(0-1)(0+1) + \frac{c}{2}0(0+1)$$

$$= \boxed{b}$$

$$q(1) = \frac{a}{2}\,1(1-1) - b(1-1)(1+1) + \frac{c}{2}1(1+1)$$

$$= \frac{a}{2} \cdot 0 - b(0) \cdot 2 + \frac{c}{2} \cdot 2$$

$$= \boxed{c}$$

Therefore is a polynomial interpolant of the points.

b) $q(x) = \frac{a}{2}x(x-1) - b(x-1)(x+1) + \frac{c}{2}x(x+1)$

Given

What it does $\begin{cases} q(-1) = a \\ q(0) = b \\ q(1) = c \end{cases}$

$p(x)$ = another quadratic

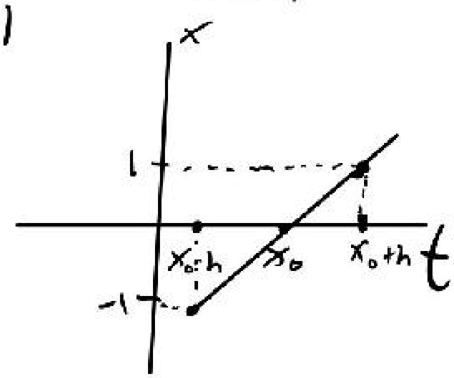
want: $\begin{cases} p(x_0-h) = a \\ p(x_0) = b \\ p(x_0+h) = c \end{cases}$

Hint: change of variable: $x = \varphi(t)$ linear

$\varphi(x_0-h) = -1$

$\varphi(x_0) = 0$

$\varphi(x_0+h) = 1$



$x = \underbrace{(\text{slope})t + \text{y-intercept}}_{\varphi(t)}$

Punchline

$x_0-h \overset{\varphi}{\longmapsto} -1 \overset{q}{\longmapsto} a$

$x_0 \longmapsto 0 \overset{q}{\longmapsto} b$

Sub that into $q(x)$

to get

$q(\varphi(t))$ don't need to simplify

$$\text{Slope} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{1-0}{x_0 + h - x_0} = \frac{1}{h}$$

$$0 = \frac{1}{h}x_0 + \text{y-int.}$$

$$\text{y-int.} = -\frac{1}{h}x_0$$

$$y(t) = \frac{1}{h}t - \frac{x_0}{h}$$

$$\boxed{\begin{aligned} p = q(y(t)) = {} & \frac{a}{2}\left(\frac{1}{h}t - \frac{x_0}{h}\right)\left(\left(\frac{1}{h}t - \frac{x_0}{h}\right)-1\right) \\ & - b\left(\left(\frac{1}{h}t - \frac{x_0}{h}\right)-1\right)\left(\left(\frac{1}{h}t - \frac{x_0}{h}\right)+1\right) \\ & + \frac{c}{2}\left(\frac{1}{h}t - \frac{x_0}{h}\right)\left(\left(\frac{1}{h}t - \frac{x_0}{h}\right)+1\right) \end{aligned}}$$

Problem 4.)

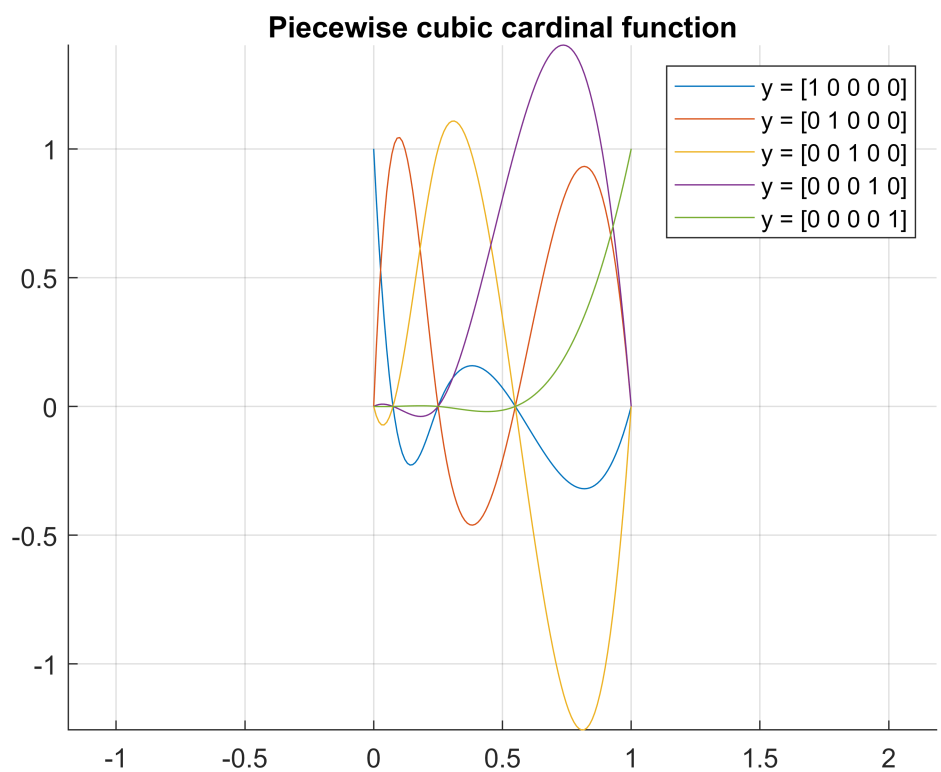
4. (Cardinal cubic splines; **FNC** 5.3.5) Although the cardinal cubic splines are intractable in closed form, they can be found numerically. Each cardinal spline interpolates the data from one column of an identity matrix. Define the nodes $\mathbf{t} = [0, 0.075, 0.25, 0.55, 1]^T$. Plot over $[0,1]$ the five cardinal functions for this node set over the interval $[0,1]$.

```
t = [0 0.075 0.25 0.55 1]';
x = linspace(0,1,100);
y = eye(5);

ydp = y(:,1)';
ydp
```

```
ydp = 1×5
     1     0     0     0     0
```

```
clf
```

```
hold on
plot(x, interp1(t, ydp, x, 'spline'))
grid on, axis equal
hold on
title('Piecewise cubic cardinal function')

ydp = y(:,2)';
plot(x, interp1(t, ydp, x, 'spline'))
ydp = y(:,3)';
plot(x, interp1(t, ydp, x, 'spline'))
ydp = y(:,4)';
plot(x, interp1(t, ydp, x, 'spline'))
ydp = y(:,5)';
plot(x, interp1(t, ydp, x, 'spline'))
legend('y = [1 0 0 0 0]', 'y = [0 1 0 0 0]', 'y = [0 0 1 0 0]', 'y = [0 0 0 1 0]', 'y = [0 0 0
```

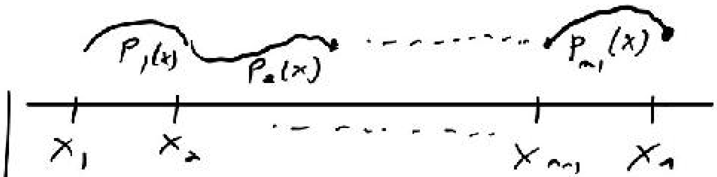


Problem 5.)

5. (Piecewise quadratic interpolation; adapted from **FNC** 5.3.6.) Suppose you were to define a piecewise quadratic spline that interpolates n given values and has a continuous first derivative. Follow the derivation presented in lecture to express all of the interpolation and continuity conditions. How many additional conditions are required to make a square system for the coefficients? Justify your answer.

Set-up

n nodes $\rightarrow (n-1)$ subintervals



$\mathbb{P}(x) = p_j(x)$ if $x \in [x_j, x_{j+1})$

$$p_j(x) = \overset{\text{const.}}{c_{i,1}} + \overset{\text{linear}}{c_{i,2}(x - x_i)} + \overset{\text{quad.}}{c_{i,3}(x-x_i)^2} \rightarrow \text{3 unknowns}$$

$\Rightarrow 3(n-1)$ unknowns total $= 3n-3$

"Spline conditions"

① interpolate

$$\underbrace{\mathbb{P}(x_j) = y_j \quad j = 1, \dots, n}_{n \text{ eqns}}$$

② matching values

$$\underbrace{p_j(x_{j+1}) = p_{j+1}(x_{j+1}) \quad j = 1, \dots, n-2}_{n-2 \text{ eqns}}$$

③ matching first derivatives

$$\underbrace{p_j'(x_{j+1}) = p_{j+1}'(x_{j+1}) \quad j = 1, \dots, n-2}_{n-2 \text{ eqns}}$$

$3n-4$ equations

$\Rightarrow 3n - 3 + (-3n+4) = 1$

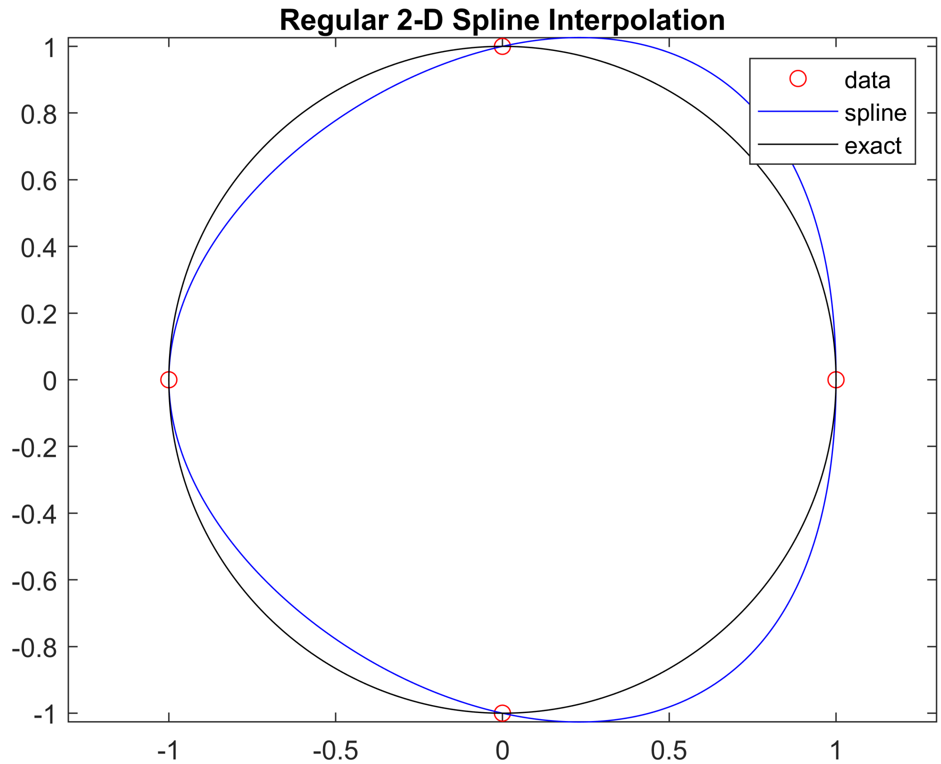
We will 1 additional condition required to make a square system for the coefficients.

Problem 6.)

6. (Cubic splines in 2-D) At the top of p. 1569 of **LM**, the term *pseudo-arc length* is introduced with an example script `an_ant.m`. Read it. Then do **LM** 12.2–15.

15) Now let's play with splines in two dimensions.
   a) We begin with a circle and equally spaced data points. Use **spline** in MATLAB and determine accuracy for $n = 5, 9, 17$, and 33 data points (where the last point is the same as the first). Do this letting $x = x(s)$ and $y = y(s)$, where s is the pseudo-arc length between adjacent data points. The er should be the maximum distance between corresponding points on the curve and circle, using about 1(points so that the curves are "filled in". Also plot the circle and the curve since the difference should visible, at least when there are only a few points.
   (i) Again use **spline**, but now use the correct first-derivative boundary conditions.

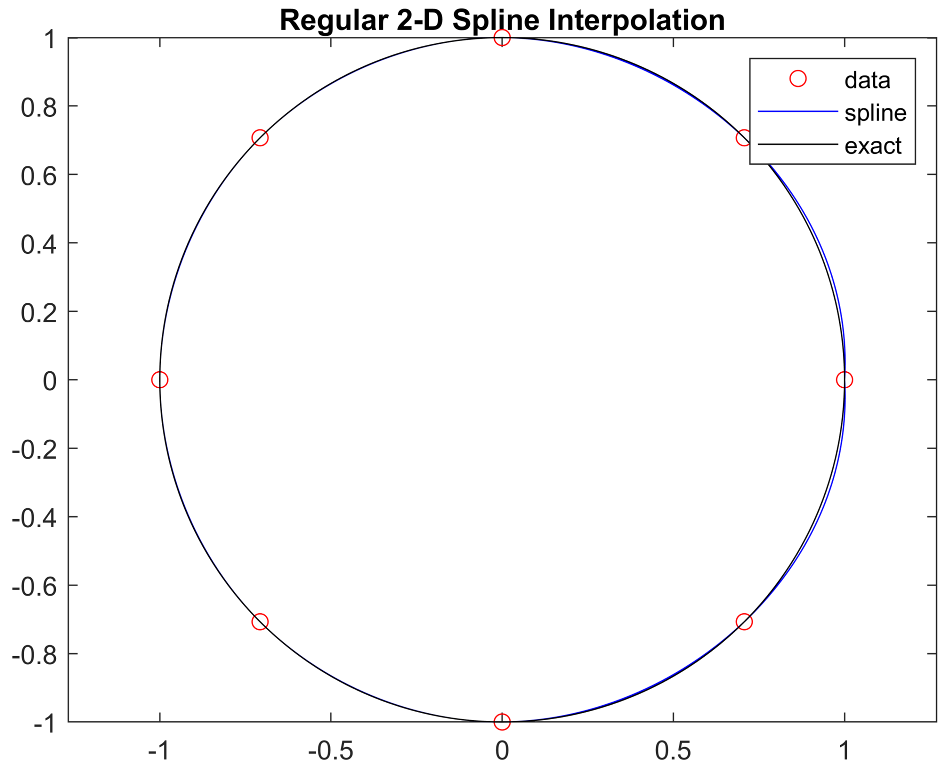
```
%To find error find the max error over the entire approx. circle and actual
%circle using sqrt((xactual - xapprox)^2 + (yactual - yapprox.)^2)

n = 5;
fx = @(theta) cos(theta);
fy = @(theta) sin(theta);

tdp = linspace(0, 2*pi, n)';
xdp = fx(tdp);
ydp = fy(tdp);

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
title('Regular 2-D Spline Interpolation')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 0.1815
```

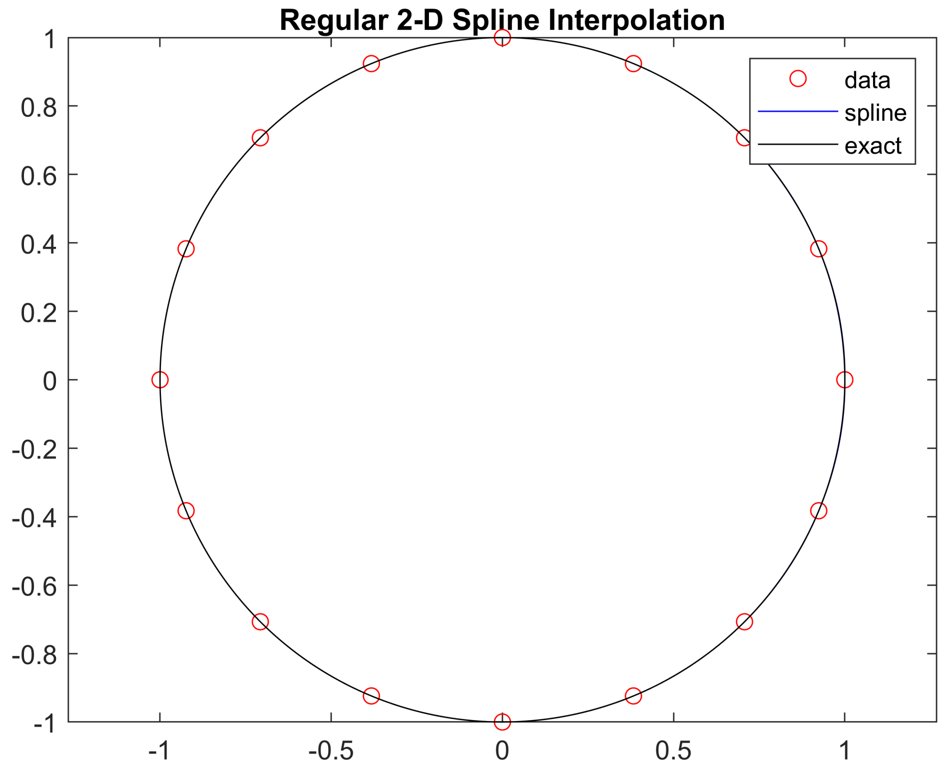
```
n = 9;
fx = @(theta) cos(theta);
fy = @(theta) sin(theta);

tdp = linspace(0, 2*pi, n)';
xdp = fx(tdp);
ydp = fy(tdp);

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
title('Regular 2-D Spline Interpolation')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 0.0106
```

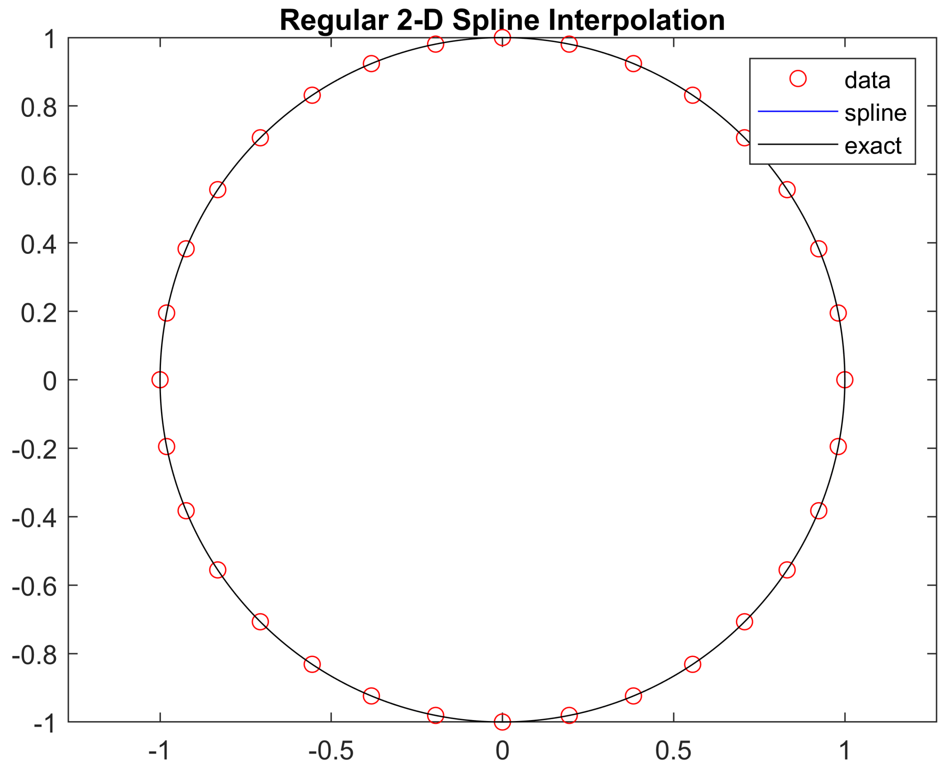
```
n = 17;
fx = @(theta) cos(theta);
fy = @(theta) sin(theta);

tdp = linspace(0, 2*pi, n)';
xdp = fx(tdp);
ydp = fy(tdp);

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
title('Regular 2-D Spline Interpolation')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 6.7032e-04
```

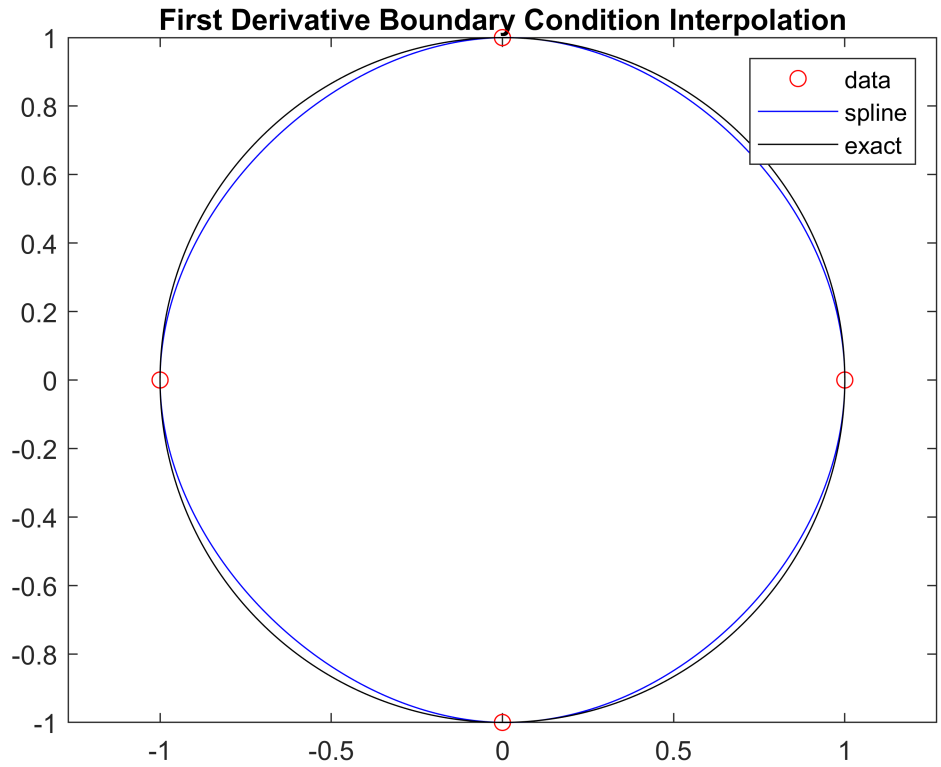
```
n = 33;
fx = @(theta) cos(theta);
fy = @(theta) sin(theta);

tdp = linspace(0, 2*pi, n)';
xdp = fx(tdp);
ydp = fy(tdp);

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
title('Regular 2-D Spline Interpolation')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 4.1942e-05
```

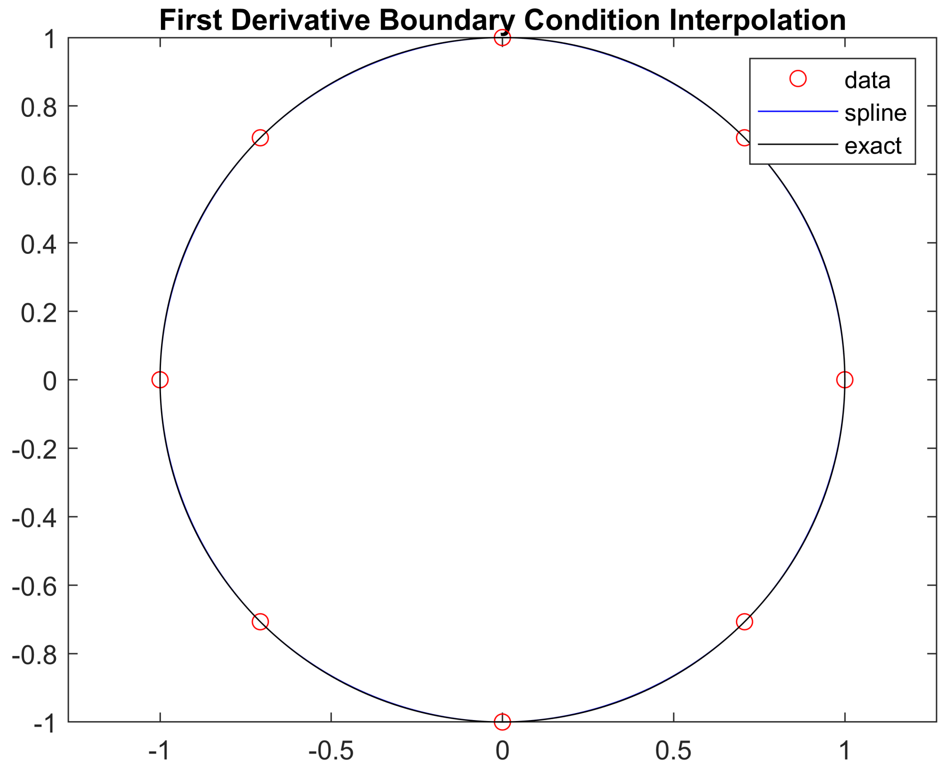
```
n = 5;
fx = @(theta) cos(theta);
fy = @(theta) sin(theta);
fxprime = @(theta) -sin(theta);
fyprime = @(theta) cos(theta);

tdp = linspace(0, 2*pi, n)';
xdp = [fxprime(tdp(1)); fx(tdp); fxprime(tdp(end))];
ydp = [fyprime(tdp(1)); fy(tdp); fyprime(tdp(end))];

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
title('First Derivative Boundary Condition Interpolation')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 0.0305
```

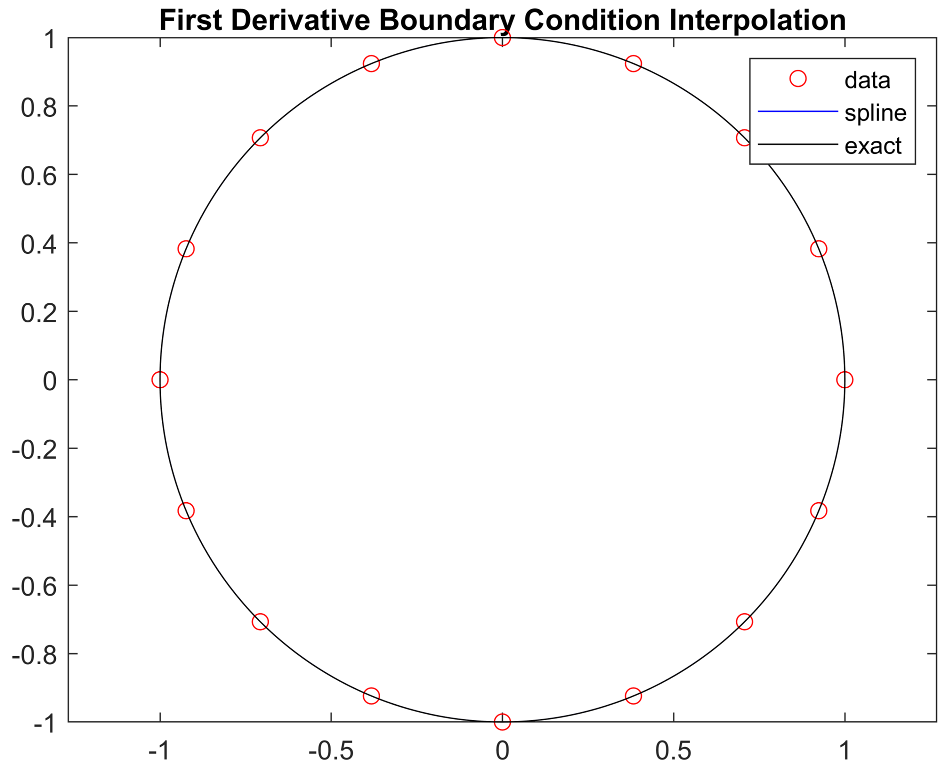
```
n = 9;
fx = @(theta) cos(theta);
fy = @(theta) sin(theta);
fxprime = @(theta) -sin(theta);
fyprime = @(theta) cos(theta);

tdp = linspace(0, 2*pi, n)';
xdp = [fxprime(tdp(1)); fx(tdp); fxprime(tdp(end))];
ydp = [fyprime(tdp(1)); fy(tdp); fyprime(tdp(end))];

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
title('First Derivative Boundary Condition Interpolation')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 0.0012
```

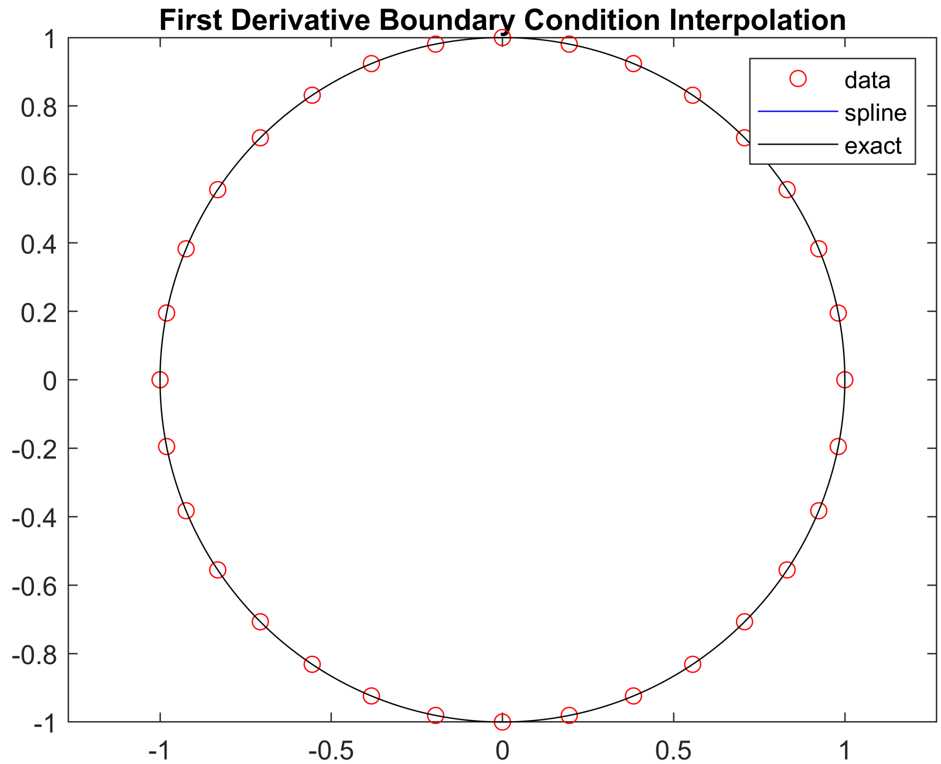
```
n = 17;
fx = @(theta) cos(theta);
fy = @(theta) sin(theta);
fxprime = @(theta) -sin(theta);
fyprime = @(theta) cos(theta);

tdp = linspace(0, 2*pi, n)';
xdp = [fxprime(tdp(1)); fx(tdp); fxprime(tdp(end))];
ydp = [fyprime(tdp(1)); fy(tdp); fyprime(tdp(end))];

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
title('First Derivative Boundary Condition Interpolation')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 6.5596e-05
```

```

n = 33;
fx = @(theta) cos(theta);
fy = @(theta) sin(theta);
fxprime = @(theta) -sin(theta);
fyprime = @(theta) cos(theta);

tdp = linspace(0, 2*pi, n)';
xdp = [fxprime(tdp(1)); fx(tdp); fxprime(tdp(end))];
ydp = [fyprime(tdp(1)); fy(tdp); fyprime(tdp(end))];

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
title('First Derivative Boundary Condition Interpolation')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 3.9276e-06
```

b) Repeat the previous part using the ellipse

$$x^2 + \frac{y^2}{9} = 1$$

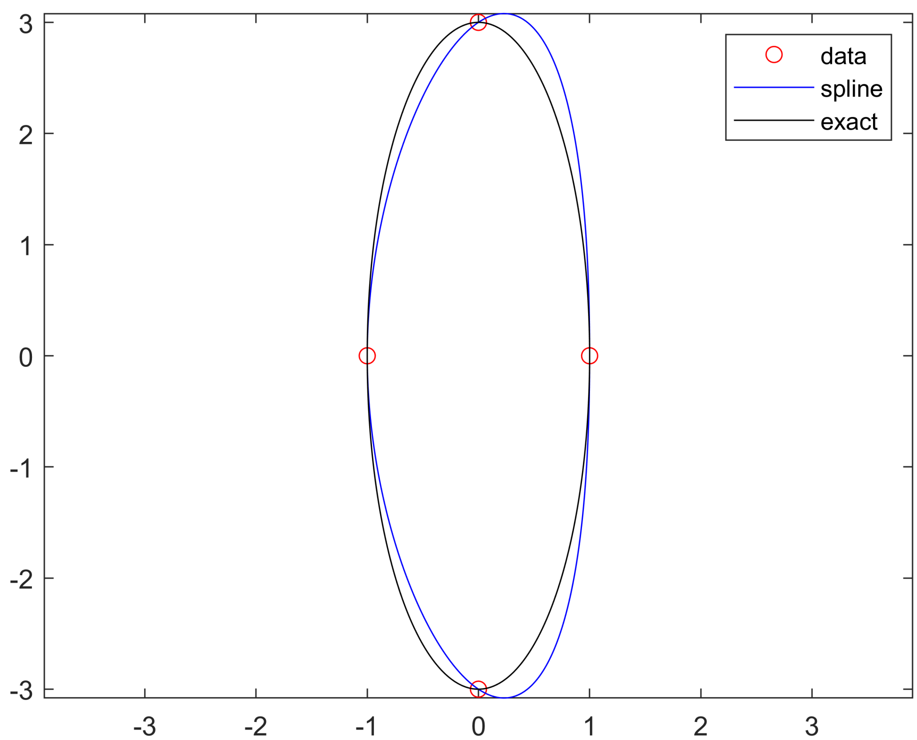
which can be easily plotted using the trigonometric representation

$$x = \cos t \quad \text{and} \quad y = 3\sin t \quad \text{for } t \in [0, 2\pi].$$

Use the parameter t, which is not the angle from the origin to the point $(\cos t, 3\sin t)$ except when t is multiple of $\pi/2$, to locate the points for calculate the error.

```
n = 5;
fx = @(theta) cos(theta);
fy = @(theta) 3*sin(theta);

tdp = linspace(0, 2*pi, n)';
xdp = fx(tdp);
ydp = fy(tdp);

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
```

```
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
```

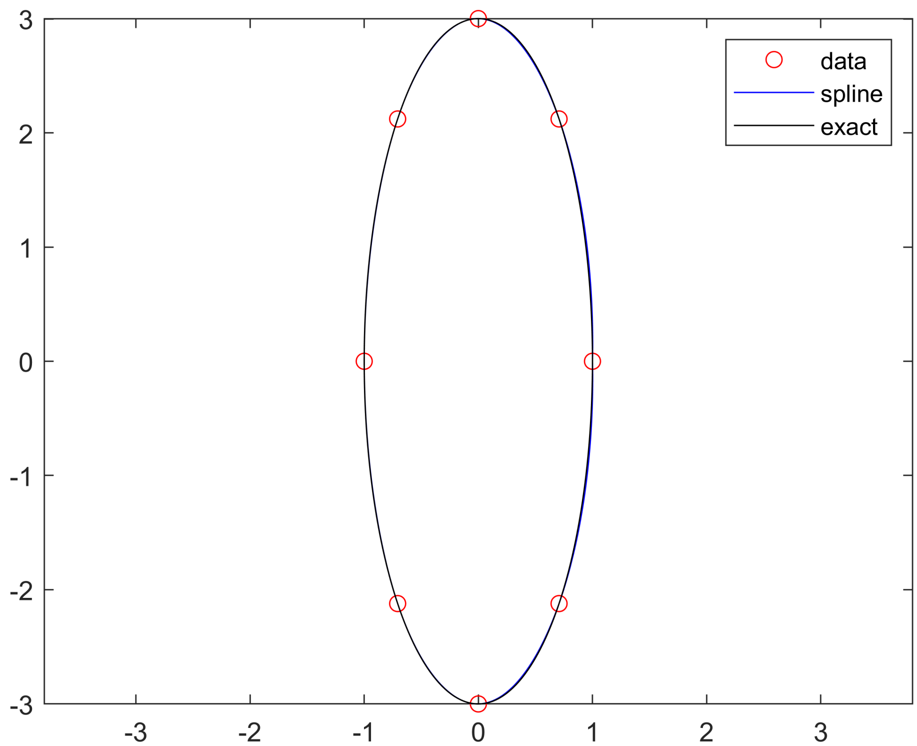



```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 0.5425
```

```
n = 9;
fx = @(theta) cos(theta);
fy = @(theta) 3*sin(theta);

tdp = linspace(0, 2*pi, n)';
xdp = fx(tdp);
ydp = fy(tdp);

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
```

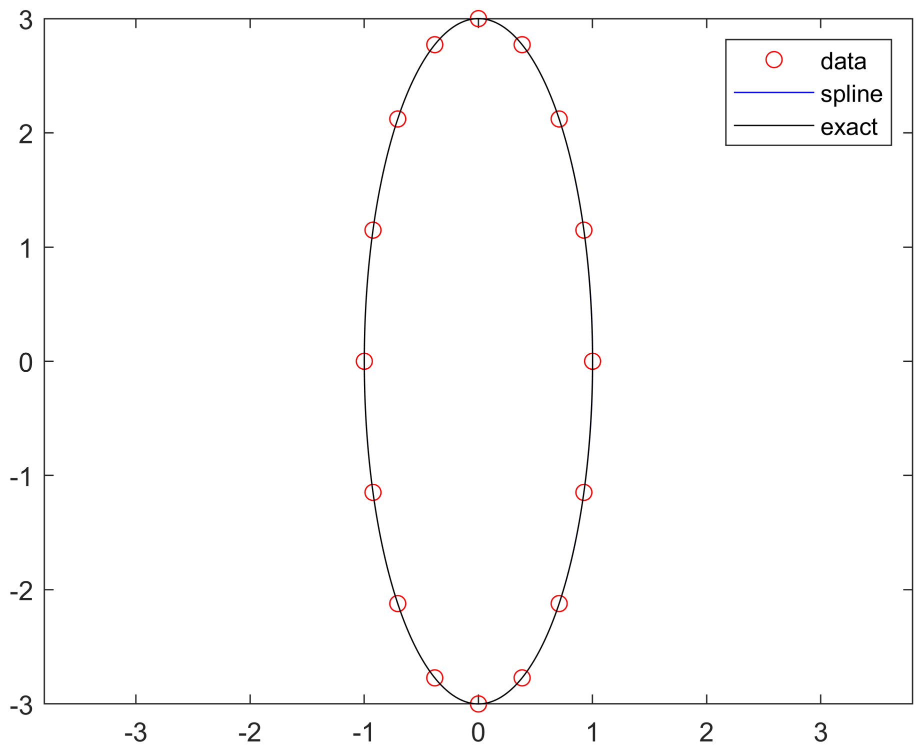
```
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 0.0244
```

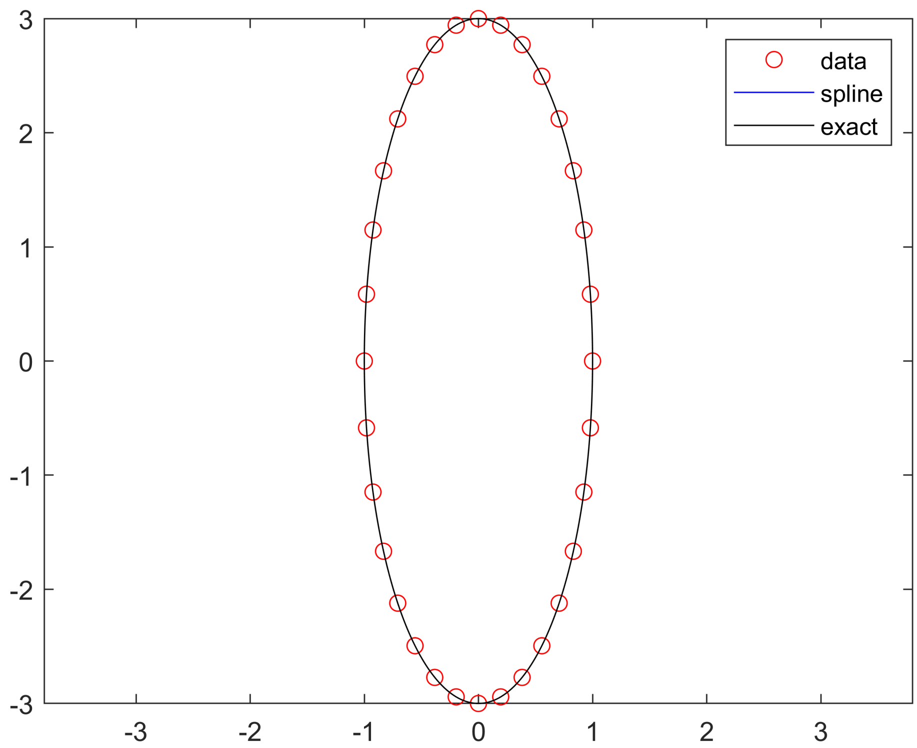
```
n = 17;
fx = @(theta) cos(theta);
fy = @(theta) 3*sin(theta);

tdp = linspace(0, 2*pi, n)';
xdp = fx(tdp);
ydp = fy(tdp);

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 0.0010
```

```

n = 33;
fx = @(theta) cos(theta);
fy = @(theta) 3*sin(theta);

tdp = linspace(0, 2*pi, n)';
xdp = fx(tdp);
ydp = fy(tdp);

%interpolation
t = linspace(0, 2*pi, 1000)';
x = spline(tdp, xdp, t);
y = spline(tdp, ydp, t);

clf
plot(xdp, ydp, 'ro'), hold on
axis equal
plot(x, y, 'b')
plot(fx(t), fy(t), 'k')
legend('data', 'spline', 'exact')
```




```
error = max(sqrt((fx(t) - x).^2 + (fy(t) - y).^2))
```

```
error = 4.8214e-05
```