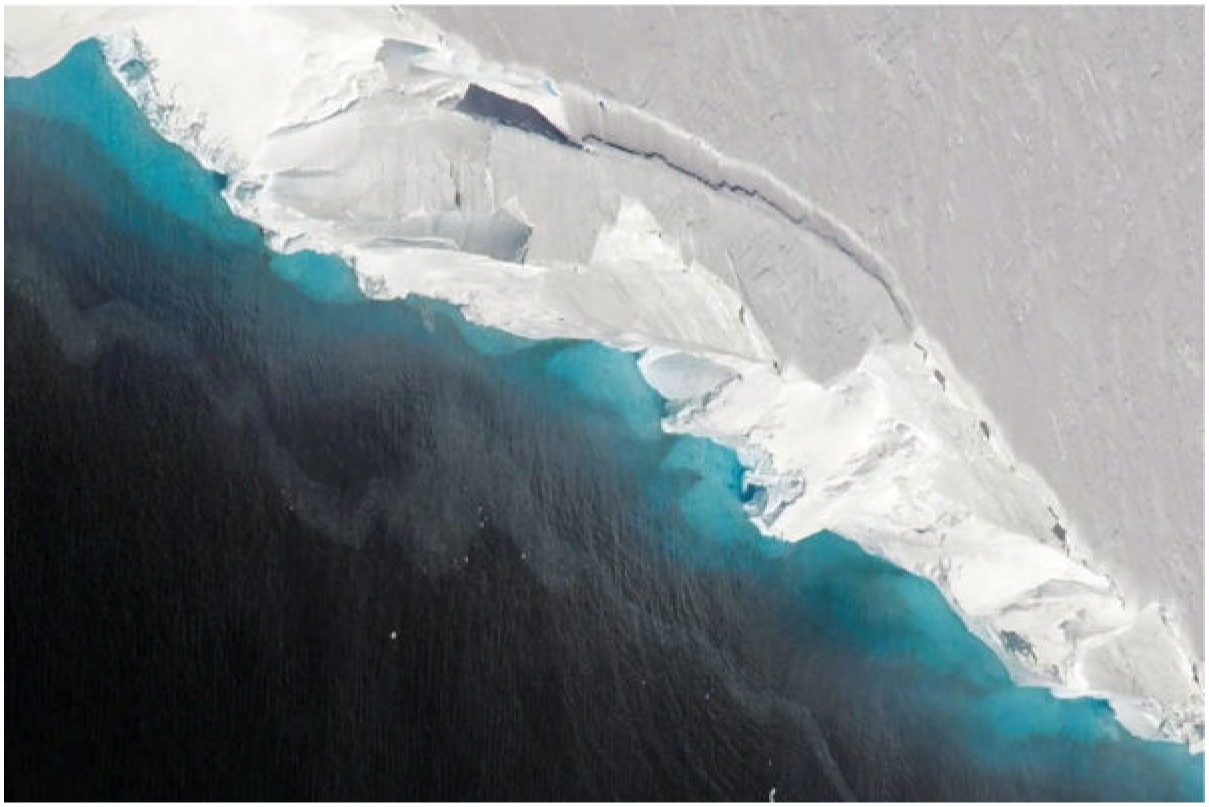
Iceberg calving

This activiti is based on a project created by prof. Donna Calhoun and uses data collected and processed by dr. Ellyn Enderlin and her students.

Iceberg calving is a process of breaking off chunks of ice of the Antarctic or Greenland ice sheets. As temperatures rise, this calving is more frequent. At the same time, it is an important mechanism for transporting fresh water from the ice-sheet to the ocean, and directly affects sea-level rise. The ice which is landed - the ice sheet - is not submerged in the ocean, and therefore not contributing to the ocean level. Once it breaks off and is floating, the sea level rises by a height corresponding to the iceberg's volume.



The Thwaites glacier. Image source: NASA/ Jeremy Harbeck (read more here)

In this project, you will work with data provided by dr. Ellyn Enderlin from Boise State Geophysics Department. The data contains an outline of a few icebergs which recently broke off of Thwaites glacier in Antarctica. The Thwaites glacier is often called a Doomsday Glacier, as recent studies (see here and here) indicate that its collapse could trigger a sea level rise by nearly 4 feet. The data was collected from satellite images by students of dr. Enderlin manually (i.e. by looking at the image and clicking the locations of the edge of the iceberg).

We will explore whether we can improve the process of computation of the iceberg area (which directly links to its volume and contribution to sea level rise) by using polynomial interpolation. We will examine whether by using high-order interpolation we can reduce the number of points required to delineate the iceberg while maintaining the accurate computation of its area. In other words, we will see whether we can save poor geophysics students some clicking. We will also explore other integration techniques than we have studied so far.

Note: this is an actual research project, and I do not know whether the answer is yes or no - it may very well not work better than just using a ton of data points and simple integration technique.

Task 1

You need to read the data in the .shp (shapefile) files. You will have to use the `shaperead()` function for it. It is included in the Mapping Toolbox. If you are using MatlabOnline, this won't be a problem, as it is accessible by default. If you have installed Matlab on your computer, click on Add-ons -> Get add-ons in the top Matlab menu, and then search for Mapping Toolbox and install it (btw, check out other toolboxes if you are interested, there is a ton of stuff there). Once you have it installed, read data on each iceberg by using the command below:

```
iceberg1 = shaperead("WV_20200123154232_icebergshape01.shp")
```

The resulting `iceberg1` is a structure, which contains information about the shape of the iceberg. What interests us is the (x, y) coordinates of the points delineating the iceberg. You can access them by taking

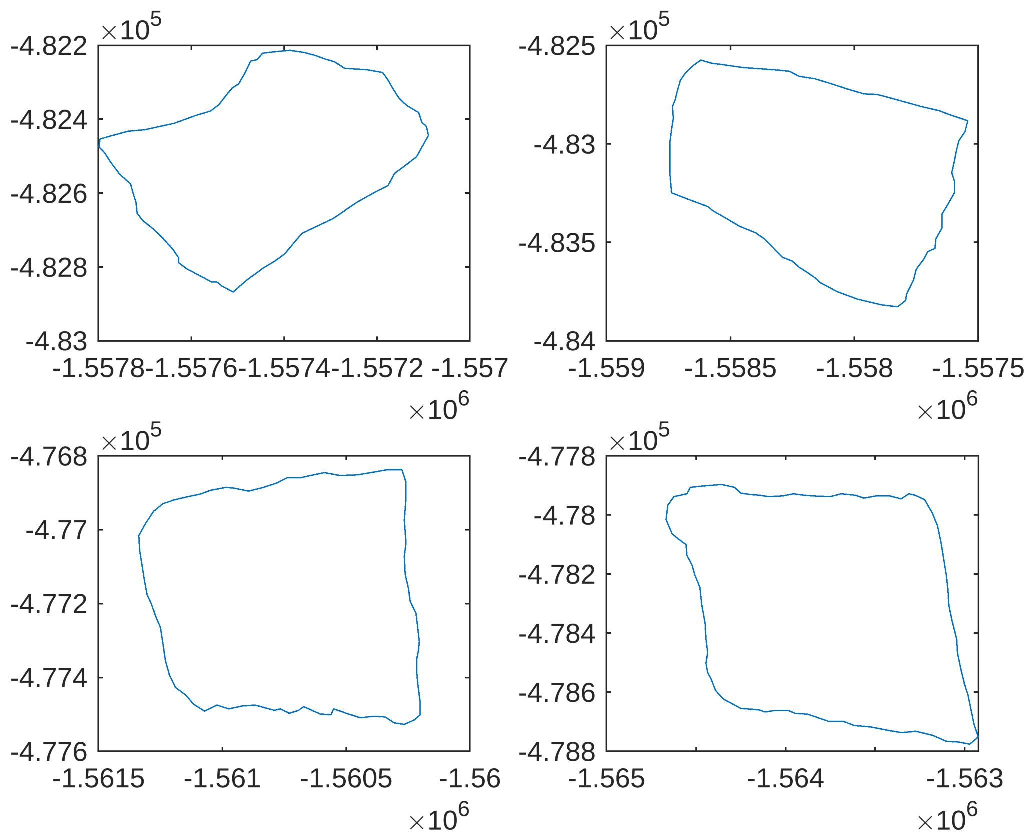
```
x1 = iceberg1.X;
y1 = iceberg1.Y;
```

Extract the (x, y) coordinates for all four icebergs and plot them. Make sure each iceberg plot has appropriate title, axis labels, etc. Use `subplot()` command to fit all four icebergs in one plot on a 2x2 grid.

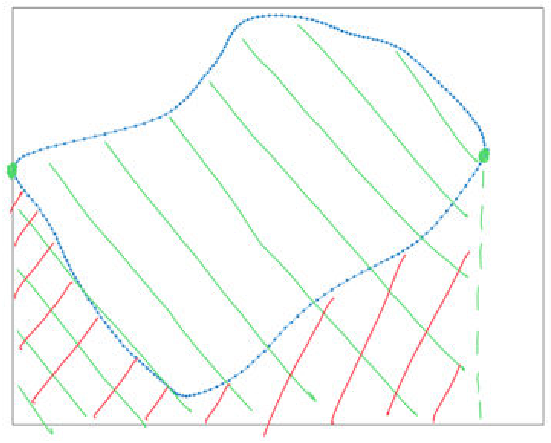
Note that the coordinates are in meters relative to 70 degrees south latitude and 0 degrees longitude and were calculated usign Antarctic Polar Stereographic Projection WGS84.

```
iceberg1 = shaperead("WV_20200123154232_icebergshape01.shp");
iceberg2 = shaperead("WV_20200123154232_icebergshape02.shp");
iceberg3 = shaperead("WV_20200123154232_icebergshape03.shp");
iceberg4 = shaperead("WV_20200123154232_icebergshape04.shp");

% original data

x1 = iceberg1.X;
y1 = iceberg1.Y;
x2 = iceberg2.X;
y2 = iceberg2.Y;
x3 = iceberg3.X;
y3 = iceberg3.Y;
x4 = iceberg4.X;
y4 = iceberg4.Y;

ice_x1 = x1(1:end-1);
ice_y1 = y1(1:end-1);
ice_x2 = x2(1:end-1);
ice_y2 = y2(1:end-1);
ice_x3 = x3(1:end-1);
ice_y3 = y3(1:end-1);
ice_x4 = x4(1:end-1);
ice_y4 = y4(1:end-1);
```

```
figure(1); clf
subplot(2,2,1)
plot(ice_x1,ice_y1)
subplot(2,2,2)
plot(ice_x2,ice_y2)
subplot(2,2,3)
plot(ice_x3,ice_y3)
subplot(2,2,4)
plot(ice_x4,ice_y4)
```



Once you have plotted the icebegs shapes, compute area of each of them using the trapezoidal rule. The trapezoidal rule assumes you are integrating a function, i.e. each x has a qunique f(x) associated with it. Here we do not have that, as the shape is clearly not a function. It's area, however, is the difference between the integral of the top curve (area marked with green lines below) and the bottom curve (area in red).



Luckily, the trapezoidal rule will handle this automatically, as it's formula:

$$A = \sum_{k=1}^{N-1} \frac{1}{2}(y_{k+1} + y_k)(x_{k+1} - x_k)$$

computes the length of the interval as $(x_{k+1} - x_k)$, so if $x_{k+1} > x_k$ the area will be positive (unless the sum $(y_{k+1} + y_k)$ is negative, which can happen if y coordinates are negative - remember they were computed with respect to some fairly arbitrary point on the globe), and when $x_{k+1} < x_k$ then the area will switch sign. In other words, you should be able to compute the area by simply applying the formula above. If you get a negative area as a result, just flip the sign.

```
% use the trapazoidal function to compute the area.
A1 = Trapazoid_ice(ice_x1, ice_y1)
```

```
A1 = 2.5656e+05
```

```
A2 = Trapazoid_ice(ice_x2, ice_y2)
```

```
A2 = 9.9520e+05
```

```
A3 = Trapazoid_ice(ice_x3, ice_y3)
```

```
A3 = 6.3650e+05
```

```
A4 = Trapazoid_ice(ice_x4, ice_y4)
```

```
A4 = 1.0971e+06
```

The area of the first iceberg should be around 256557 square meters. Compute the area for all four icebergs. We will use those values as "truth", even though it is still an approximation using many data points and trapezoidal rule (so effectively linear interpolation). You may want to create a function which integrated the area of the shape given the (x, y) coordinates, rather than repeating the same code multiple times.

Task 2

We want to interpolate the iceberg shape using polynomial interpolation (either Lagrange, pchip or spline), but as we mentioned above, y is not a function of x. The solution to this problem is to express both x and y as functions of some independent variable t such that we have $x = x(t)$ and $y = y(t)$. The independent variable can be anything - you can imagine it to be the angle from the center of the shape, which covers the range $t \in (0, 2\pi)$, or simply the index $t = 1, 2, \ldots, N$ where N is the number of data points in each iceberg's delineation. We call the representation $(x(t), y(t))$ a parametric curve. Once we do that, we can interpolate each function $x(t)$ and $y(t)$ independently and with respect to t.

First, create the independent variable t:

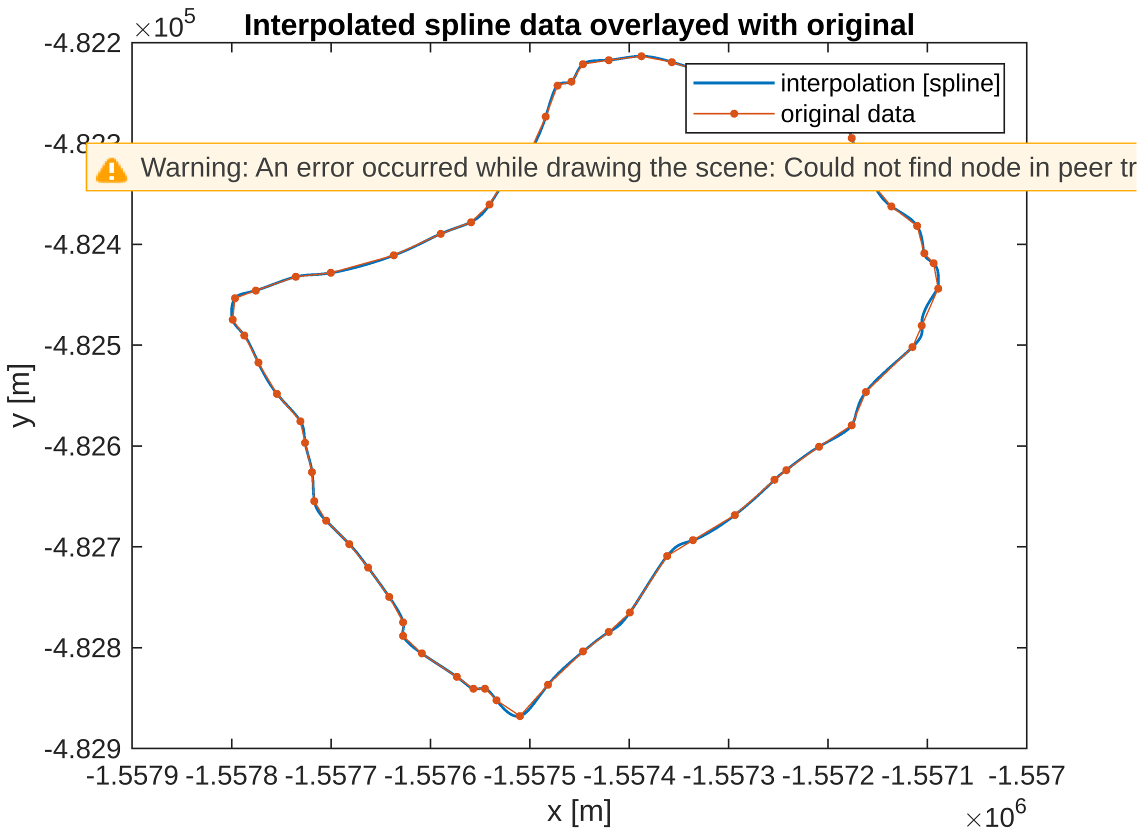
```
figure(2); clf
t = 1:length(ice_x1);
x = ice_x1(t)
```

```
x = 1×68
10^6 ×
   -1.5571   -1.5571   -1.5571   -1.5571   -1.5571   -1.5572   -1.5572   -1.5572 ···
```

```
y = ice_y1(t);
```

Then, choose an interpolation method (pchip or spline, as Lagrange is probably very hard for so many data points) and use it to interpolate both $x(t)$ and $y(t)$ on ten times as many points as there originally are in each set (i.e. if $N = 63$ then use 630 interpolation points). Plot the interpolated (x, y) shape and compare it with the original data points (plot both on the same plot) to check whether the interpolated shape is still ok.

```
tt = 1:0.1:length(x);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
xx = spline(t,x,tt); % 't' is the independant data, and 'x' is the dependant
data.
yy = spline(t,y,tt); % 't' is the independant data, and 'y' is the dependant
data.
plot(xx,yy,'LineWidth',1)
hold on
plot(x,y,'.-','MarkerSize',10)
legend('interpolation [spline]','original data')
title('Interpolated spline data overlayed with original')
xlabel('x [m]'); ylabel('y [m]')
```



Warning: An error occurred while drawing the scene: Could not find node in peer tr

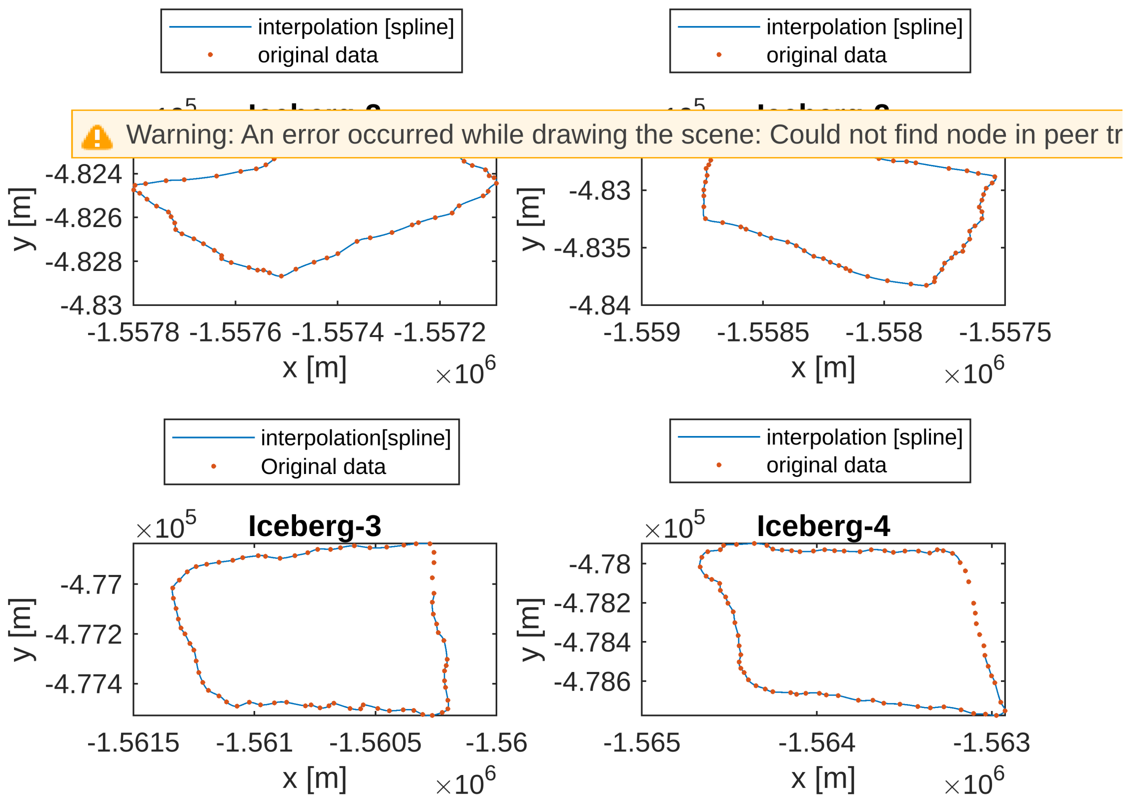
It may happen that there are some wiggles at the end of the interpolated shape (i.e. the last point does not match well with the first point). In that case add the first point (i.e. x(1)) to the end of the x array (you can do something like $x(N+1) = x(1)$, and similarly for y) and adjust the independent variable so it has the same array size as new x and y. This should fix this issue.

Interpolate all the icebergs using your selected method and plot them all together with the original data, again using subplot() command to lay out the results on a 2x2 grid.

```
figure(3); clf
subplot(2,2,1)
t = 1:length(ice_x1);
x = ice_x1(t);
y = ice_y1(t);
tt = 1:0.1:length(x);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
xx = spline(t,x,tt); % 't' is the independant data, and 'x' is the dependant
data.
yy = spline(t,y,tt); % 't' is the independant data, and 'y' is the dependant
data.
plot(xx,yy)
hold on
plot(x,y,'.')
```

```
legend('interpolation [spline]','original data', 'Location', 'NorthOutside')
title('Iceberg-2')
xlabel('x [m]'); ylabel('y [m]')

subplot(2,2,2)
t2 = 1:length(ice_x2);
t_new2 = 1:length(ice_x2)+1;
xi = ice_x2(t2);
x_2 = [ice_x2(t2),xi(1)];
yi = ice_y2(t2);
y_2 = [ice_y2(t2),yi(1)];
tt = 1:0.1:length(x2);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
xx = spline(t_new2,x_2,tt); % 't' is the independant data, and 'x' is the
dependant data.
yy = spline(t_new2,y_2,tt); % 't' is the independant data, and 'y' is the
dependant data.
plot(xx,yy)
hold on
plot(x_2,y_2,'.')
legend('interpolation [spline]','original data', 'Location', 'NorthOutside')
% title('Interpolated spline data overlayed with original')
title('Iceberg-2')
xlabel('x [m]'); ylabel('y [m]')

subplot(2,2,3)
t3 = 1:length(ice_x3);
t_new3 = 1:length(ice_x3)+1;
xj = ice_x3(t3);
x_3 = [ice_x3(t3),xj(1)];
yj = ice_y3(t3);
y_3 = [ice_y3(t3),yj(1)];
tt = 1:0.1:length(x2);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
xx = spline(t_new3,x_3,tt); % 't' is the independant data, and 'x' is the
dependant data.
yy = spline(t_new3,y_3,tt); % 't' is the independant data, and 'y' is the
dependant data.
plot(xx,yy)
hold on
plot(x_3,y_3,'.')
legend('interpolation[spline]', 'Original data', 'Location', 'NorthOutside')
% title('Interpolated spline data overlayed with original')
title('Iceberg-3')
xlabel('x [m]'); ylabel('y [m]')
```

```
subplot(2,2,4)
t4 = 1:length(ice_x4);
t_new4 = 1:length(ice_x4)+1;
xk = ice_x4(t4);
x_4 = [ice_x4(t4),xk(1)];
yk = ice_y4(t4);
y_4 = [ice_y4(t4),yk(1)];
tt = 1:0.1:length(x2);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
xx = spline(t_new4,x_4,tt); % 't' is the independant data, and 'x' is the
dependant data.
yy = spline(t_new4,y_4,tt); % 't' is the independant data, and 'y' is the
dependant data.
plot(xx,yy)
hold on
plot(x_4,y_4,'.')
legend('interpolation [spline]','original data', 'Location', 'NorthOutside')
title('Iceberg-4')
xlabel('x [m]'); ylabel('y [m]')
```


Warning: An error occurred while drawing the scene: Could not find node in peer tr

Task 3

Our goal is to see what happens if we decrease the number of data points, and whether the polynomial interpolation can still represent the original shape of the iceberg. First, lets select every m-th data point from the original (x, y), where $m = 2, 3, 4, \ldots$ can be freely chosen by you (make sure you create a variable m which can be easily changed).

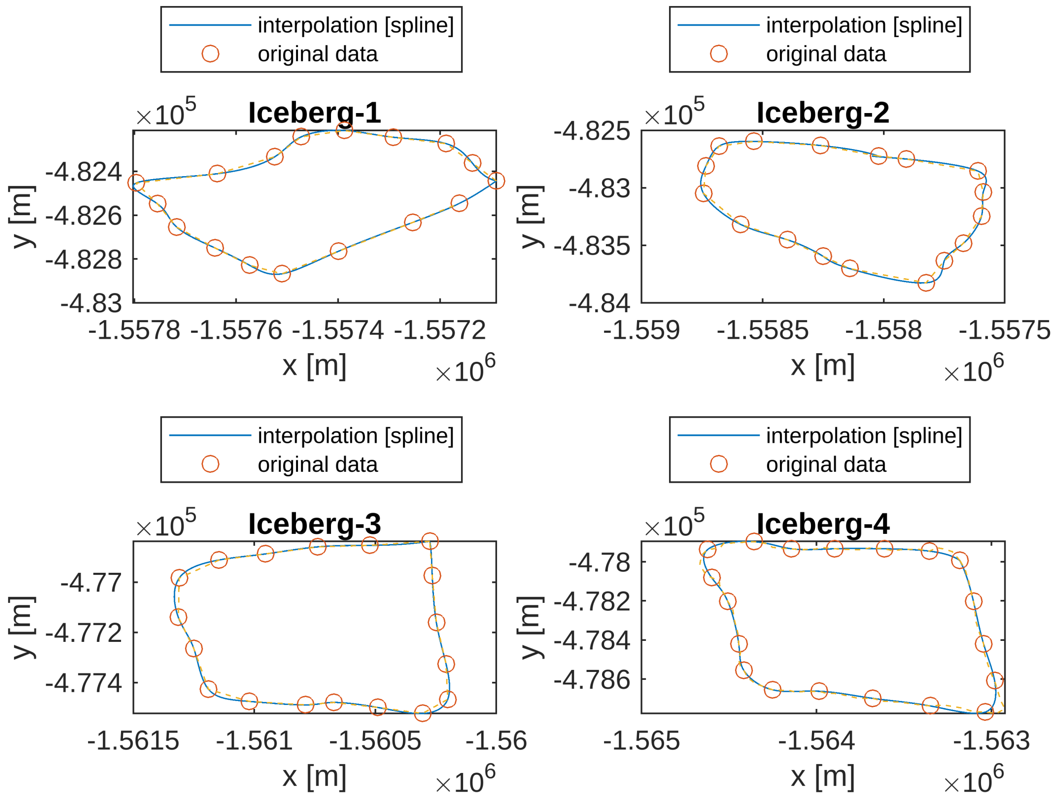
```
close all
figure(4); clf
m = 4;
```

Then, interpolate the new, smaller data set using the same method as in Task 2 and plot the results for all four icebergs, and compare them against both the original data points and the smaller set (all in one plot, choose the markers for the data points such that the plots are readable, make sure the legends are there)

```
subplot(2,2,1)
%t = 1:length(ice_x1);
x = ice_x1;%(t);
x2 = x(1:m:end);
x_2 = [x2,x2(1)];
y = ice_y1;%(t);
y2 = y(1:m:end);
y_2 = [y2,y2(1)];
t1 = 1:length(x_2);

tt1 = 1:0.1:length(x_2);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
xx1 = spline(t1,x_2,tt1);
yy1 = spline(t1,y_2,tt1); % 't' is the independant data, and 'y' is the
dependant data.
plot(xx1,yy1);
hold on
plot(x_2,y_2,'o');
plot(x2,y2,'--')
title('Iceberg-1')
legend('interpolation [spline]','original data', 'Location', 'NorthOutside')
xlabel('x [m]'); ylabel('y [m]')

subplot(2,2,2)
tb = 1:length(ice_x2);
xb = ice_x2(tb);
x2b = xb(1:m:end);
x_2b = [x2b,x2b(1)];
yb = ice_y2(tb);
y2b = yb(1:m:end);
y_2b = [y2b,y2b(1)];
t1b = 1:length(x_2b);
tt2 = 1:0.1:length(x_2b);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
```

```
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
xx2 = spline(t1b,x_2b,tt2);
yy2 = spline(t1b,y_2b,tt2); % 't' is the independant data, and 'y' is the
dependant data.
plot(xx2,yy2)
hold on
plot(x_2b,y_2b,'o');
plot(x_2b,y_2b,'--');
legend('interpolation [spline]','original data', 'Location', 'NorthOutside')
title('Iceberg-2')
xlabel('x [m]'); ylabel('y [m]')

subplot(2,2,3)
tc = 1:length(ice_x3);
xc = ice_x3(tc);
x2c = xc(1:m:end);
x_2c = [x2c,x2c(1)];
yc = ice_y3(tc);
y2c = yc(1:m:end);
y_2c = [y2c,y2c(1)];
t1c = 1:length(x_2c);
tt3 = 1:0.1:length(x_2c);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
xx3 = spline(t1c,x_2c,tt3);
yy3 = spline(t1c,y_2c,tt3); % 't' is the independant data, and 'y' is the
dependant data.
plot(xx3,yy3)
hold on
plot(x_2c,y_2c,'o');
plot(x_2c,y_2c,'--');
legend('interpolation [spline]','original data', 'Location', 'NorthOutside')
title('Iceberg-3')
xlabel('x [m]'); ylabel('y [m]')

subplot(2,2,4)
td = 1:length(ice_x4);
xd = ice_x4(td);
x2d = xd(1:m:end);
x_2d = [x2d,x2d(1)];
yd = ice_y4(td);
y2d = yd(1:m:end);
y_2d = [y2d,y2d(1)];
t1d = 1:length(x_2d);
tt4 = 1:0.1:length(x_2d);
%spline returns a vector of interpolated values 'xx' or 'yy' corresponding to
%the query points in tt. The values of 'xx' or 'yy' are determined by cubic
spline interpolation of 'x' and 'y'.
```

```
xx4 = spline(t1d,x_2d,tt4);
yy4 = spline(t1d,y_2d,tt4); % 't' is the independant data, and 'y' is the
dependant data.
plot(xx4,yy4)
hold on
plot(x_2d,y_2d,'o');
plot(ice_x4,ice_y4,'--');
legend('interpolation [spline]','original data', 'Location', 'NorthOutside')
title('Iceberg-4')
xlabel('x [m]'); ylabel('y [m]')
```



In your opinion, what is the maximum value of the skip m which still preserves the original shape of the icebergs?

Answer:

I would not want to reduce my points to less than 10 for these images. Anything less will begin to cause large amounts of distortion. Perferably an (m) size of 4 or less seems to be ideal based on my visual analysis. I would say an (m) size of 4 seems ideal for understanding macro processes in Cryospher Geophysics while saving field time and manual interpolation.

Task 4

Create a function, which accepts the original data set (x, y), the value of the skip m, interpolates the reduced data set (just like in Task 3) on 1000 data points and computes the area of the interpolated shape. For the integration you can use the trapezoidal rule from Task 1, or the more accurate polynomial integration discussed in the appendix (extra credit).

```
Area = Trapazoid_ice2(ice_x1, ice_y1)
```

```
Area = 2.5673e+05
```

```
Area = Trapazoid_ice2(ice_x2, ice_y2)
```

```
Area = 9.9687e+05
```

```
Area = Trapazoid_ice2(ice_x3, ice_y3)
```

```
Area = 6.3733e+05
```

```
Area = Trapazoid_ice2(ice_x4, ice_y4)
```

```
Area = 1.0984e+06
```

```

m = 4
```

```
m = 4
```

```
Area1 = Trapazoid_ice3(ice_x1,ice_y1,m)
```

```
Area1 = 2.5979e+05
```

```
Area2 = Trapazoid_ice3(ice_x2,ice_y2,m)
```

```
Area2 = 9.9303e+05
```

```
Area3 = Trapazoid_ice3(ice_x3,ice_y3,m)
```

```
Area3 = 6.3344e+05
```

```
Area4 = Trapazoid_ice3(ice_x4,ice_y4,m)
```

```
Area4 = 1.0943e+06
```

Once you are confident your function works correctly, use it to compute the area of each iceberg for different values of m (use your judgement how high you can go with m and still not completely loose the shape of the iceberg - see Task 3). Use the areas computes in Task 1 as reference, and compute the relative error of the interpolated area computation.

$$error = \frac{|A_{interpolated} - A_{reference}|}{|A_{reference}|}$$

```
error1(m) = abs(Area1 - A1)/abs(A1)
```

```
error1 = 1×4
         0         0         0    0.0126
```

```
error2(m) = abs(Area2 - A2)/abs(A2)
```

```
error2 = 1×4
         0         0         0    0.0022
```

```
error3(m) = abs(Area3 - A3)/abs(A3)
```

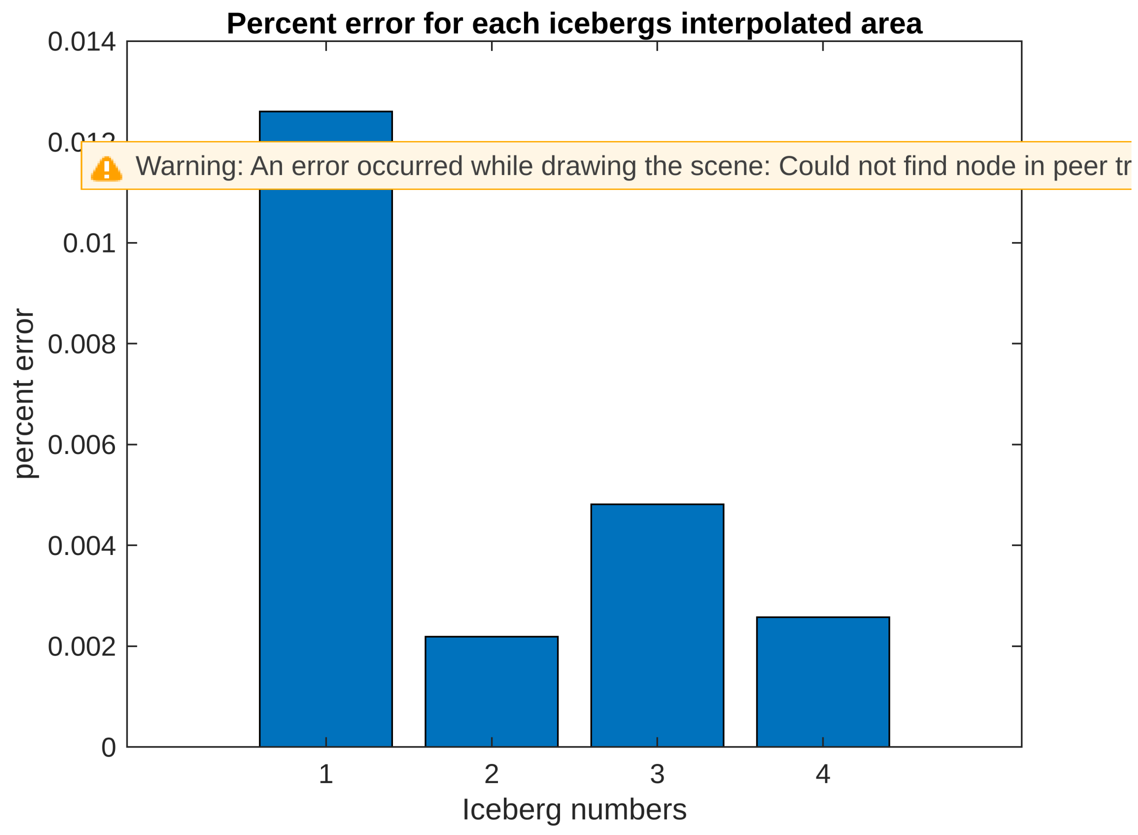
```
error3 = 1×4
         0         0         0    0.0048
```

```
error4(m) = abs(Area4 - A4)/abs(A4)
```

```
error4 = 1×4
         0         0         0    0.0026
```

Plot the error as a function of m for each iceberg on one plot. Choose the best way to present this data (i.e. line plot, logarithmic plot, bar plot, etc).

```
close all
error = [error1(m) error2(m) error3(m) error4(m)];
m_array = [1 2 3 4];
bar(m_array,error);
xlabel('Iceberg numbers')
ylabel('percent error')
title('Percent error for each icebergs interpolated area')
```



Warning: An error occurred while drawing the scene: Could not find node in peer tr

Does this approach make sense? I.e., are we getting similar estimates of the iceberg area when using the interpolation on the reduced number of points as we have in Task 1? We can assume that 10% error is acceptable. What is the maximum value of m which produces estimates below that error? (**note:** I don't know the answer to this question, and it is possible this will not work at all - use your critical thinking here)

Answer:

The greator the step function the higher the percent error will become, I started by using an m value of 4 giving me the highest percent error for iceberg-1 equal to 1.26 percent error. This is significantly less than 10% so it is acceptable. The maximum value of (m) that will produce acceptable bellow the 10% error threshold is 15.

Appendix - polynomial integration and derivation

In this project we first interpolate the shape of the iceberg with a cubic (third order) polynomial, and then integrate it using a trapezoidal rule, whch essentially assumes a linear interpolation between the integration points. Putting aside the question whether out polynomial interpolation is correct in this case or not (as you have explored in Task 2 and 3), we are not using the polynomial interpolant to it's full potential while integrating it.

Note that we know how to integrate and derivate polynomials exactly. Take the following function:

$$p(t) = c_1 t^3 + c_2 t^2 + c_3 t + c_4$$

The integral and derivative is given by:

$$\int_a^b p(t)dt = \left[\frac{1}{4}c_1 t^4 + \frac{1}{3}c_2 t^3 + \frac{1}{2}c_3 t^2 + c_4 t\right]_a^b$$

$$\frac{d}{dt}p(t) = 3c_1 t^2 + 2c_2 t + c_3$$

In other words, just by manipulating the polynomial coefficients, we can create new polynomials which will compute exact integrals and derivatives. This means that if we can represent our functions and processes as polynomials accurately (regardless whether they are Lagrange polynomials or piecewise polynomials), then we can compute very accurate integrals and derivatives, which has huge consequences in numerical solutions of differential equations, approximation theory, and many other fields.

How does this apply to this problem?

We want to compute the area of an iceberg delimited by a parametric curve ($(x(t), y(t))$). From calculus we know that such area can be computed as:

$$A = \int_0^T x(t)y'(t)dt$$ where we assumed $t \in (0, T)$.

We have both $x(t)$ and $y(t)$ given as polynomials. Even though they are piecewise polynomials, we can compute the integral as a sum of integrals on each segment separately, and we can do it exactly, as explained above. The integrand, however, is a bit weird, as it involves a product of $x(t)$ and derivative $y'(t)$. If $x(t)$ is a cubic polynomial, then $y'(x)$ will be a quadratic, and the product of the two will be a fifth order polynomial, which we can integrate exactly, and we can calculate it's coefficients directly from the coefficients of $x(t)$ and $y(t)$. This is a bit much for this short project (but is a nice idea for the final one, if you are interested - let me know), but let's explore this integration a bit further.

We can use a built in `integrate()` routine to integrate any function with fairly good accuracy, but we first need to create the integrand function $f(t) = x(t)y'(t)$. To do that, we can compute exact derivative of $y(t)$. We note from the formulas above that all it takes to compute an exact derivative is to manipulate the coefficients of the original polynomial. Let's explore this on an example.

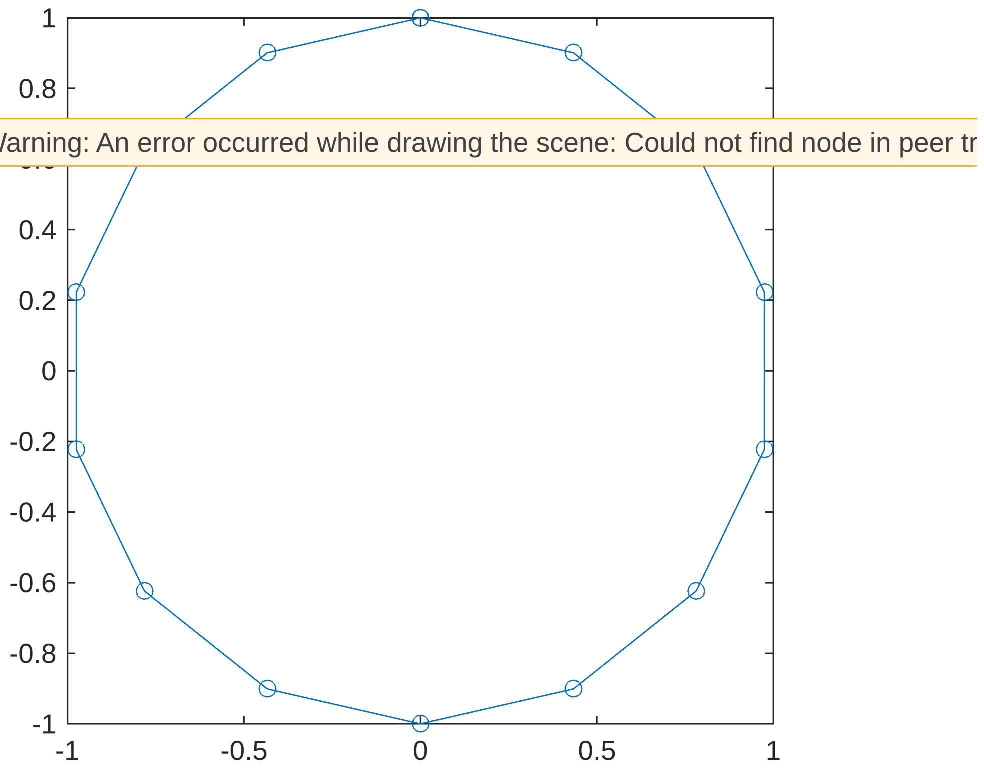
Let's parametrize the circle as:

$$x(t) = \sin(t)$$

$$y(t) = \cos(t)$$

with $t \in [0, 2\pi]$.

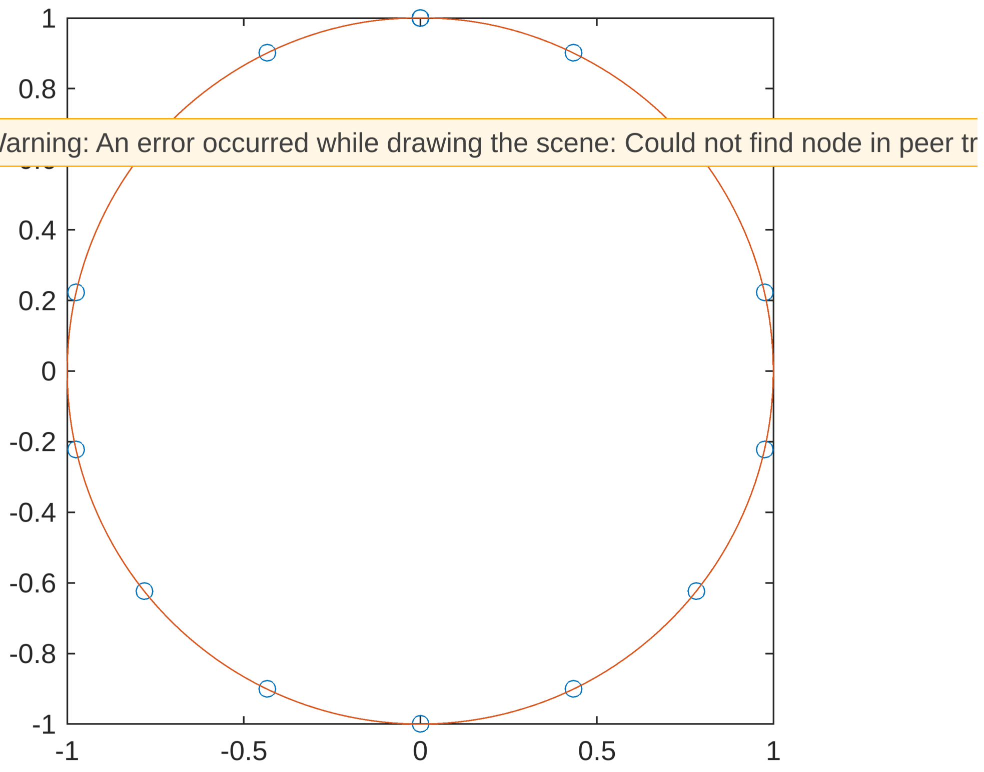
```
t = linspace(0,2*pi,15);
x = sin(t);
y = cos(t);
plot(x,y,'-o')
daspect([1 1 1])
```



Warning: An error occurred while drawing the scene: Could not find node in peer tr

I am going to interpolate this paramerić curve with a cubic spline. The representation of my circle gets much better.

```
ti = linspace(0,2*pi,200);
xj = spline(t,x,ti);
yj = spline(t,y,ti);
plot(x,y,'o',xj,yj)
daspect([1 1 1])
```



Warning: An error occurred while drawing the scene: Could not find node in peer tr

Iinstead of extracting the values of the interpolant at selected points `ti`, I can extract the coefficients of the piecewise polynomials themselves.

```
px = spline(t,x)
```

```
px = struct with fields:
      form: 'pp'
    breaks: [0 0.4488 0.8976 1.3464 1.7952 2.2440 2.6928 3.1416 3.5904 4.0392 4.4880 4.9368 5.3856 5.8344
     coefs: [14×4 double]
    pieces: 14
     order: 4
       dim: 1
```

```
py = spline(t,y)
```

```
py = struct with fields:
      form: 'pp'
    breaks: [0 0.4488 0.8976 1.3464 1.7952 2.2440 2.6928 3.1416 3.5904 4.0392 4.4880 4.9368 5.3856 5.8344
     coefs: [14×4 double]
    pieces: 14
     order: 4
       dim: 1
```

The results are two structures px and py, with coefficients given as 14 x 4 matrix, since we have 14 segments (15 original data points) and cubic polynomials, so 4 coefficients each. Each row of this matrix corresponds to one set of cubic polynomial coefficients.

```
px.coefs
```

```
ans = 14×4
   -0.1383   -0.0272    1.0068         0
   -0.1383   -0.2133    0.8989    0.4339
   -0.0711   -0.3995    0.6239    0.7818
   -0.0005   -0.4952    0.2223    0.9749
    0.0731   -0.4958   -0.2224    0.9749
    0.1314   -0.3975   -0.6234    0.7818
    0.1639   -0.2206   -0.9008    0.4339
    0.1639    0.0000   -0.9998    0.0000
    0.1314    0.2206   -0.9008   -0.4339
    0.0731    0.3975   -0.6234   -0.7818
     ⋮
```

To integrate the circle area I need to use the formula

$$A = \int_0^{2\pi} x(t)y'(t)dt$$

so I need to compute $y'(t)$ first. I can use the relation

$$y(t) = c_1t^3 + c_2t^2 + c_3t + c_4$$

$$y'(t) = 3c_1t^2 + 2c_2t + c_3 = b_1x^2 + b_2x + b_3$$

where $[b_1, b_2, b_3]$ are the coefficients of the quadratic polynomial for the derivative, and

$$b_1 = 3c_1\,,\ b_2 = 2c_2\,,\ b_3 = c_3$$

In other words, I need to transform my matrix of coefficients C from a 14x4 matrix to a 14x3 matrix B using the relation above. I can do that by taking a matrix product

$$B = CD$$

where D is a 4x3 matrix which looks as follows:

$$D = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

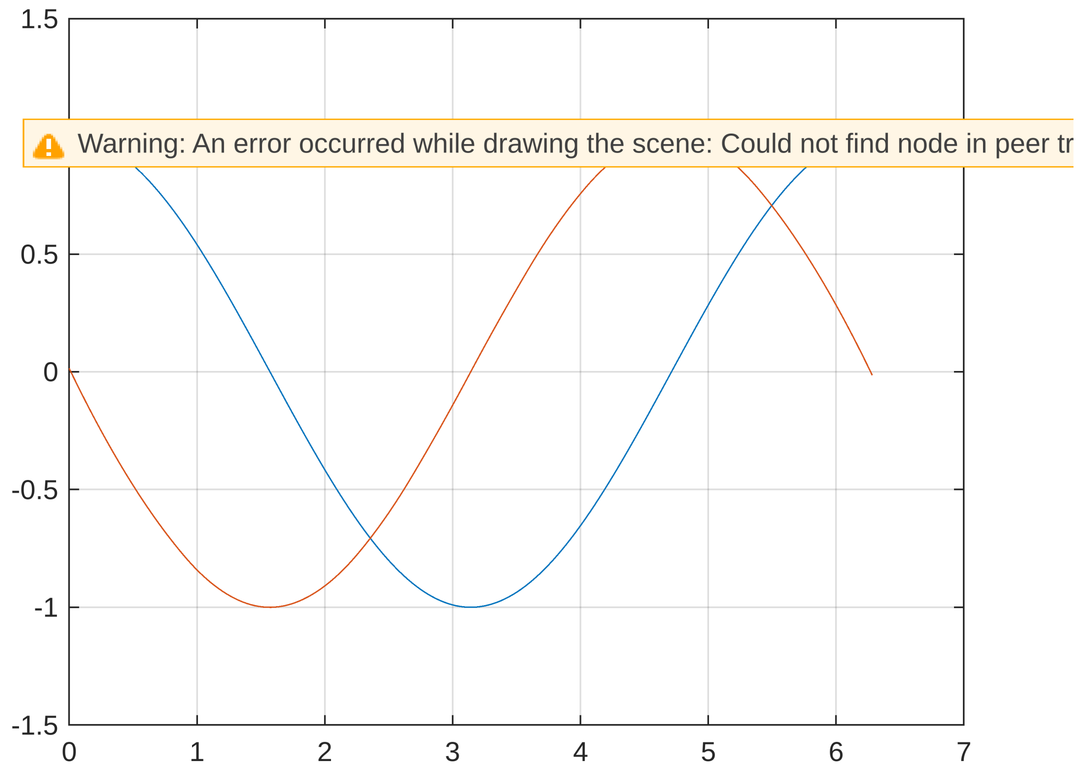
This will compute the polynomial coefficients of the derivative. I will then create a new spline by copying the original spline, modify the number of coefficients and the coefficient values like do:

```
D = [3 0 0; 0 2 0; 0 0 1; 0 0 0];

pdydt = py;
pdydt.order = 3;
pdydt.coefs = py.coefs*D
```

```
pdydt = struct with fields:
      form: 'pp'
    breaks: [0 0.4488 0.8976 1.3464 1.7952 2.2440 2.6928 3.1416 3.5904 4.0392 4.4880 4.9368 5.3856 5.8344
     coefs: [14×3 double]
    pieces: 14
     order: 3
       dim: 1
```

As a result we have a piecewise spline polynomial representing the derivative of $y(t)$. We can check that by plotting both $y(t)$ and $y'(t)$ by using `ppval()` function, which evaluates a piecewise polynomial:

```
plot(ti,ppval(py,ti),ti,ppval(pdydt,ti))
grid on
```



```
Warning: An error occurred while drawing the scene: Could not find node in peer tr
```

Since $\cos(t) = \frac{d}{dt}\sin(t)$, it looks like we got it right. Now on to creating the integrand and integrating it.

```
fun = @(t) ppval(px,t).*ppval(pdydt,t)
```

```
fun = function_handle with value:
    @(t)ppval(px,t).*ppval(pdydt,t)
```

```
A = integral(fun,0, 2*pi)
```

```
A = -3.1417
```

Since we are integrating the circle, its area must be $A = \pi r^2$, so we got pretty close by interpolating 15 points on that circle and integrating the parametric curve area. Our result is actually negative (due to the choice of the direction of the parameterization), but taking an absolute value will produce the correct result.