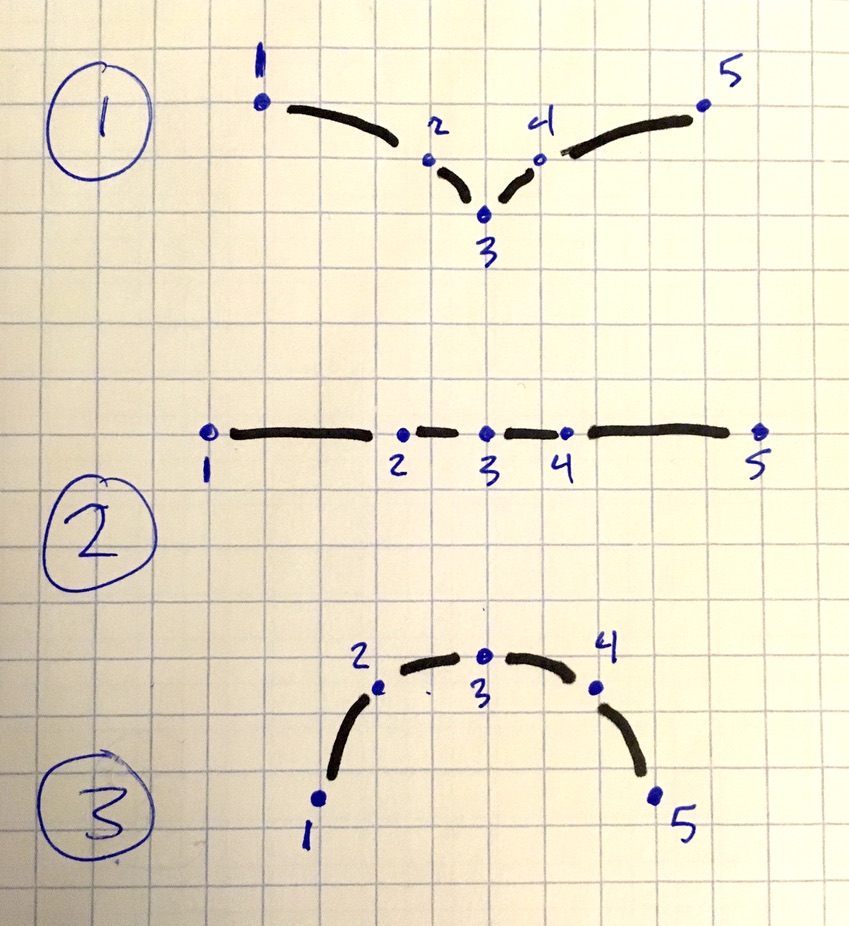
Project 3 John Krone

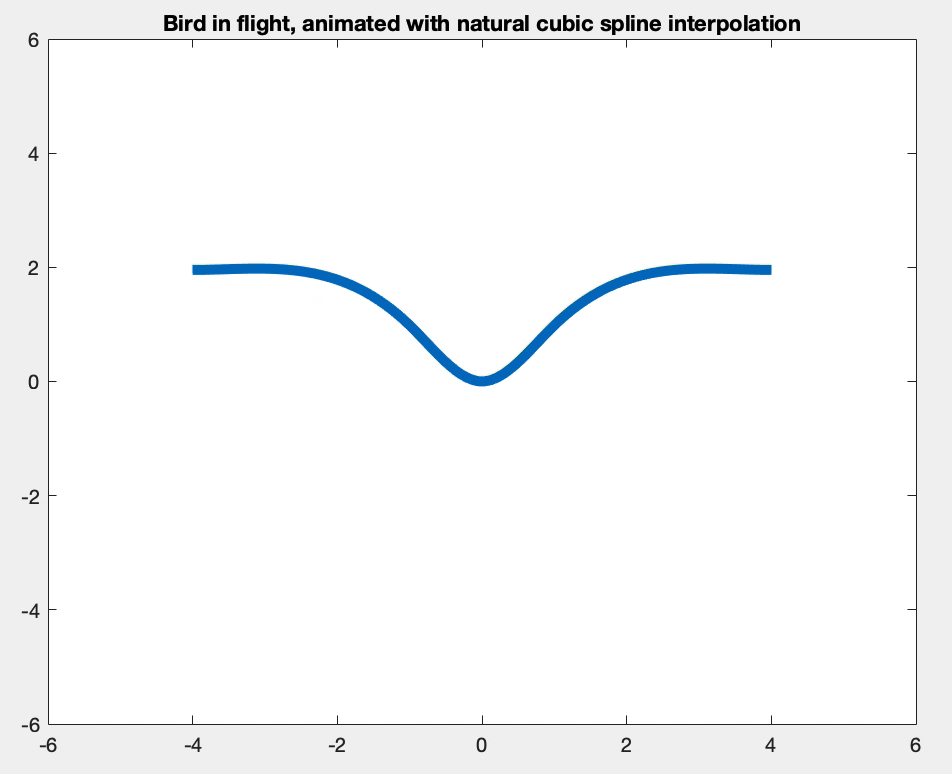
Math 458

In this project I decided to animate a flying bird. Below is a sketch of the frames I attempted to animate. Each frame is numbered in the order in which it appears, and the control points are the blue dots, labeled with their respective numbers. The black curves in between dots are an approximation of what the start frames look like.

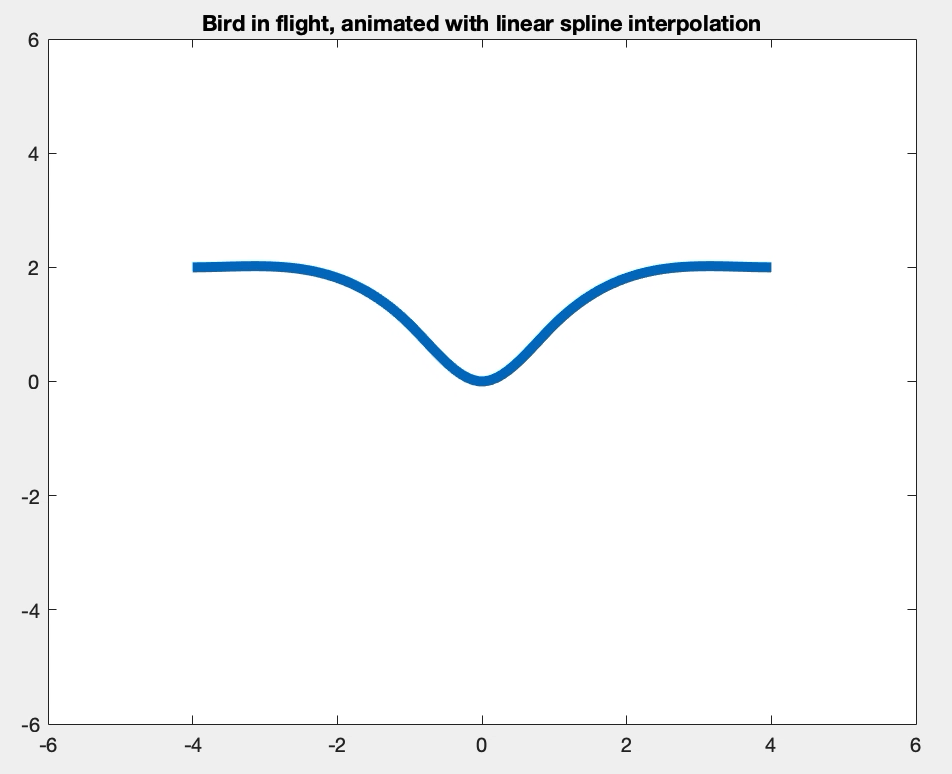
 I implemented three different animations: one with polynomial interpolation, one with natural cubic spline interpolation, and one with linear spline interpolation. I need to make one important note before explaining further. In each of the three animations, there is technically two interpolations used.

* Interpolation #1: I used the first interpolation to decide (or estimate, rather) how the control points should move. Interpolation #1 corresponds with the three that I listed above.
* Interpolation #2: The second interpolation connects the control points in each frame, creating the image of the bird.

Interpolation #1 is the focus of this project. Interpolation #2 is not the focus of this project. It is only used as a convenient means of illustrating a bird. For consistency, interpolation #2 is the same in each of the three different animations. It is a natural spline interpolation. You can tell by looking at the figures on the next page, all three of which have derivative of 0 at the endpoints throughout the animation.

A close up of a map

Description automatically generated



On this page you can examine how each of the three implementations turned out. Each of these is a “.mov” file, and you can double click on them to watch the animation. If you have any issues playing the animations, please email me and I will try to send this project in a different from. At first glance the animations will all appear quite similar, especially the polynomial interpolation and cubic spline interpolation implementations. These two animations look quite similar because each control point only had three predetermined points, corresponding with the 3 frames in the image on page 1. As we have learned in class, cubic spline is preferable to polynomial interpolation because it is much less oscillatory, but since there are only three interpolation points, there is not much opportunity for the polynomial interpolation animation to oscillate wildly. If I did this project over again, I might include 10 or 15 predetermined points for each control point. In this case it would be very likely that the polynomial interpolation and cubic spline interpolation would appear quite different.

Next, let’s take a closer look at the specific differences between each animation. To do so, focus on control point #1, the leftmost point of the bird. In the first animation, the polynomial interpolation has a nice quadratic shape. Follow it closely as the animation plays. You will notice that it moves like a parabola. You can do the same thing with the second animation and notice that control point #1 moves almost like a sideways normal distribution that you would see in an intro probability or statistics course. It does this because the second animation uses a natural spline, so the derivatives at the endpoints are zero. Of course, the derivatives here appear to be undefined. That is because I switched the x and y values of the control points to interpolate them, and then switched them back to plot them. Otherwise, the polynomial interpolation would be exceedingly oscillatory. Finally examine the third animation and you notice that control point 1 moves in a piecewise linear fashion as we would expect from a linear spline interpolation. These paths are sketched out in the image to the right. And the control point is denoted ‘1,i’ where ‘i’ is the appropriate predetermined point number.

You can further examine the other control points and you will find they all move in a somewhat similar fashion to control point #1, according to the type of interpolation that was used. This is a very satisfactory and expected result.

A close up of text on a white background

Description automatically generated

Finally, if you are interested, feel free to examine my code. I utilized cell arrays throughout in order to be as efficient as possible.

%{

John Krone

MATH 458 Fall 2019

Project 3

jkrone@usc.edu

%}

clear; clc;

%% Control point coordinates

% create cp coordinates

cp{1} = {[-4,2], [-5,0], [-3,-2.5]};

cp{2} = {[-1,1], [-1.5,0], [-2,-.5]};

cp{3} = {[1,1], [1.5,0], [2,-.5]};

cp{4} = {[4,2], [5,0], [3,-2.5]};

% organize x and y vals

for i = 1:4

for j = 1:3

cpx{i}(j) = cp{i}{j}(1);

cpy{i}(j) = cp{i}{j}(2);

end

end

%% Polynomial Interpolation of control points

% find coefficients

% note: I must switch x and y vals to get a perfect poly fit (error = 0)

for i = 1:4

coef\_PI{i} = polyfit(cpy{i},cpx{i}, 2);

end

% find interpolated values

% note backwards x and y vals still

y\_vals = {[],[],[],[]};

x\_vals = {[],[],[],[]};

for i = 1:4

y\_vals{i} = linspace(min(cpy{i}),max(cpy{i}));

x\_vals{i} = polyval(coef\_PI{i},y\_vals{i});

end

% reorganize x\_vals and y\_vals to be used in animation

% first add the origin coordinates

y\_vals = {y\_vals{1:2}, zeros(1,100), y\_vals{3:4}};

x\_vals = {x\_vals{1:2}, zeros(1,100), x\_vals{3:4}};

% now "transpose" the cell array

for i = 1:100

for j = 1:5

x\_reorg{i}(j) = x\_vals{j}(i);

y\_reorg{i}(j) = y\_vals{j}(i);

end

end

% animate a figure

figure(1)

frames = [1:100,100:-1:2];

for j = 1:3

for k = 1:198

i = frames(k);

XX = linspace(min(x\_reorg{i}), max(x\_reorg{i}));

YY = spline(x\_reorg{i}, [0 y\_reorg{i} 0], XX);

plot(XX,YY,'LineWidth',5)

title("Bird in flight, animated with polynomial interpolation")

xlim([-6,6])

ylim([-6,6])

pause(0.01)

end

end

%% Cubic Spline Interpolation of control points

% find coefficients

% again switching x and y vals for the spline portion of the function

for i = 1:4

splines{i} = spline(cpy{i},[0 cpx{i} 0]); % This is natural spline

coefs\_CS{i} = splines{i}.coefs;

end

% find interpolated values

% note backwards x and y vals still

y\_vals = {[],[],[],[]};

x\_vals = {[],[],[],[]};

for i = 1:4

y\_vals{i} = linspace(min(cpy{i}),max(cpy{i}));

x\_vals{i} = ppval(splines{i},y\_vals{i});

end

% reorganize x\_vals and y\_vals to be used in animation

% first add the origin coordinates

y\_vals = {y\_vals{1:2}, zeros(1,100), y\_vals{3:4}};

x\_vals = {x\_vals{1:2}, zeros(1,100), x\_vals{3:4}};

% now "transpose" the cell array

for i = 1:100

for j = 1:5

x\_reorg{i}(j) = x\_vals{j}(i);

y\_reorg{i}(j) = y\_vals{j}(i);

end

end

% animate a figure

figure(2)

frames = [1:100,99:-1:2];

for j = 1:3

for k = 1:198

i = frames(k);

XX = linspace(min(x\_reorg{i}), max(x\_reorg{i}));

YY = spline(x\_reorg{i}, [0 y\_reorg{i} 0], XX);

plot(XX,YY,'LineWidth',5)

title("Bird in flight, animated with natural cubic spline interpolation")

xlim([-6,6])

ylim([-6,6])

pause(0.01)

end

end

%% Linear Spline Interpolation of control points

% No need to find coefficients this time

% again switching x and y vals for the spline portion of the function

% First reset x and y vals.

y\_vals = {[],[],[],[]};

x\_vals = {[],[],[],[]};

for i = 1:4

y\_vals{i} = linspace(min(cpy{i}),max(cpy{i}));

x\_vals{i} = interp1(cpy{i}, cpx{i}, y\_vals{i}, 'linear');

end

% reorganize x\_vals and y\_vals to be used in animation

% first add the origin coordinates

y\_vals = {y\_vals{1:2}, zeros(1,100), y\_vals{3:4}};

x\_vals = {x\_vals{1:2}, zeros(1,100), x\_vals{3:4}};

% now "transpose" the cell array

for i = 1:100

for j = 1:5

x\_reorg{i}(j) = x\_vals{j}(i);

y\_reorg{i}(j) = y\_vals{j}(i);

end

end

% animate a figure

figure(2)

frames = [1:100,99:-1:2];

for j = 1:3

for k = 1:198

i = frames(k);

XX = linspace(min(x\_reorg{i}), max(x\_reorg{i}));

YY = spline(x\_reorg{i}, [0 y\_reorg{i} 0], XX);

plot(XX,YY,'LineWidth',5)

title("Bird in flight, animated with linear spline interpolation")

xlim([-6,6])

ylim([-6,6])

pause(0.01)

end

end