Project 2: Numerical Analysis I – Fall 2009

Robert Clewley

October 25, 2009

Introduction

Your project involves using Bezier curves for graphic design, numerical differentiation, and iterative optimization of a model using least squares minimization by gradient descent.

Project Tasks

Task 1 (35 points)

We briefly discussed Bezier curves in class and your first task is to convert Algorithm 3.6 on p. 162 into a working program. Write a general-purpose function, `get_coeffs`, that accepts two endpoints (x_0, y_0) and (x_1, y_1), and two associated control points (α_0, β_0) and (α_1, β_1). Your function will generate a list of the coefficients needed to define the cubics given by Eqs. (3.24) and (3.25) in the form shown in the equation under Algorithm 3.6.

Task 2 (35 points)

Write a second function, `make_bezier`, that represents the Bezier curve as a function. This function must accept a list of coefficients as produced by the above function and a t value between 0 and 1, and returns an (x, y) pair corresponding to the position of the curve at that t value. Remember, t runs from 0 to 1 along the whole curve, although t=0.5 will not necessarily mean you are half way along the curve.

You will test your code using Exercises 3 and 4 from Section 3.5. Plot each Bezier curve making up the entire curve that your code generates using the above functions. The cubics for Ex. 3 are given in the solutions in the back of the book, and the result of Ex. 4 is self-evident.

Task 3 (30 points)

This task is not directly related to Bezier curves, but is used in the extra credit portion of the project if you choose to attempt it. Section 4.1 presents a variety of approximate numerical derivatives to a scalar function $f(x)$, for a given step size h. Fixing h to be 1e-3, write a function `df` that returns the approximate gradient at x0 given f and scalar x0 as arguments, using the three point central difference formula. Test your gradient function on $f(x) = 2x^2$ and $g(x) = \exp(3x)$ at $x = 3$. Demonstrate correctness using calculus.

Extra credit portion

In these tasks we will explore a more common form of least squares minimization than you encountered in class. The goal will be to automatically find the "best" Bezier curve to fit a set of sample data points instead of by trial and error. For many nonlinear problems, there are no closed-form analytic expressions for the parameters of the best fitting function, i.e. no Normal Equations for coefficients. Instead, an iterative process must be used, along the lines of Newton's Method-like iteration. Since the slope of the error function is also generally not available in closed form, we will approximate that using the numerical gradient function you defined in Task 2 above.

Task X1 (10 points)

We will use a simple and very intuitive local optimization method known as "Gradient Descent". You may read about it at the Wikipedia page en.wikipedia.org/wiki/Gradient_descent, although that is a more advanced treatment than we need. Briefly, this method attempts to minimize an error function $f(x)$ from some starting position x0 in the following way. The local gradient of f is computed at x0 and moving a small distance ε backwards along the gradient generates a new point x1. In a multidimensional problem, we descend along the steepest direction in the error function "landscape", but for our simple case we just take steps to either increase or decrease x appropriately since f is a scalar function only. The steeper the gradient, the more quickly we descend. We stop when the gradient becomes sufficiently flat that we accept the x value as a local minimum. We will assume that our initial conditions will lie in the convergence region for the minima that we wish to find.

Fixing ε to be 1e-1, write a function that performs gradient descent using the stopping condition that successive x values differ by less than xtol=1e-4 or if there are more than 500 iterations. Test your algorithm on the error function $E(x) = (x-2)^2$ from a starting point $x = 4$.

Questions:

1) What is the gradient at this initial point?
2) What should be the solution?
3) Does your code find it? How quickly?

Task X2 (10 points)

We will now use this iterative optimization method to solve the following problem. Consider the data points (x_i, y_i) for $i = 0,\ldots,6$ that are sampled from the shape of a cursive '*v*' character traced from a document: [0.284, 1.0], [0.305, 0.65], [0.317, 0.3], [0.32, 0.0], [0.335, 0.25], [0.338, 0.6], [0.32, 1.0].

Plot lines between these points on a graph with axes limits [0.26, 0.38] in x and [-0.1, 1.1] in y to see the approximate shape.

By trial and error, a graphic designer of PostScript computer fonts has already painstakingly found one Bezier curve B1(t) that describes the left part of the character, but has not completed a second curve B2(t) for the right side of the character. Curve B1 is specified by the endpoint coordinates [0.284, 1], [0.32, 0] and control point coordinates [0.305, 0.7], [0.32, 0.4]. Similarly, curve B2 is currently specified by [0.32, 0], [0.32, 1] and [0.38, 0.3], [0.34, 0.7].

Plot each of the Bezier curves using your functions from Tasks 1 and 2 above and also plot the data points to see how well each matches. Graphically, what is the problem remaining with B2? We will discover that the lower control point's alpha value (x coordinate of the control point) is not a good choice.

Write a function `make_B2` that takes just alpha as an argument and uses your code from Task 1 to return a new version of B2, keeping the remaining endpoints and control point coordinates the same. From the above data, your initial value of α_0 is therefore 0.38.

Write a new error function that takes an α_0 value as input and returns the error between the data point targets and B2. Measure this error as the sum of the squares of the distance between the data points (x_4, y_4), (x_5, y_5) and the Bezier curve B2 at two sample points specified by t=0.3 and t=0.6. For instance, in pseudo-code it might look like this:

```
function E(alpha0)
    B2 = make_B2(alpha0)
    p = B2(0.3), q = B2(0.6)
    return $(p_x-x_4)$**2 + $(q_y-y_4)$**2 + $(p_x-x_5)$**2 + $(q_y-y_5)$**2
```

Questions:

1) What is the error at the initial point?
2) What is the gradient of your error function at the initial point?
3) Do you expect this will lead to a rapid convergence to a solution using gradient descent? Why? (Compare to the rate of convergence for your test problem in X2.)

Task X3 (10 points)

Use this function as the error function for an application of your gradient descent algorithm from Task X1 using the same tolerances. Demonstrate that this fixes the problem with B2 by plotting successive iterations of the B2 curve from within $E(\alpha_0)$ and finding that the final curve matches the data to produce the '*v*' character correctly. You might like to plot the *p*, *q* points as dots from inside your error function to show their progress at each iterate.

Questions:

1) How fast does your algorithm achieve its optimal solution?
2) Plot a graph of the one-dimensional error landscape as a function of α_0 ranging from 0 to 10. What shape does it resemble? (This is the shape that optimization algorithms are always best suited for!)
3) Is it reasonable to expect that we have found a *global* minimum? Why?

Submission, grading, and advice

Your numerical grade will be based on your *documented success* in writing the code to solve the project tasks, and any other analysis you wish to include. That involves writing a **short report** of no more than 6 pages typeset, including no more than 8 modestly sized embedded figures (wherever appropriate to your explanation), and not including source code (which you may list in an appendix if you wish).

The professionalism of your technical writing is one of the assessment criteria, and includes being able to state ideas concisely, to use clear logic, and to take advantage of mathematical concepts you have learned in this course when appropriate. You will submit it electronically as a single document (**PDF preferred**). You may submit a hand-written paper copy of your report if you prefer, but see the Deadlines section below.

Your grade will also reflect an evaluation of your code. Your goal is to use clear logical principles to break down the tasks and to comment and describe your code in your report so that I can easily comprehend your solution. If I judge your code to be particularly difficult to comprehend or untrustworthy in its assumptions then you will lose points. Use of modularity, spacing, and comments helps greatly in this respect. Therefore, you will also submit your **original source code file(s)** (archived as appropriate depending on how many you have). You may *not* use *any* pre-existing library implementations that come with Maple, Matlab, or the Python numerical libraries, without modification. All algorithms must be your own implementations or versions that you adapt from codes that you find in the libraries, the internet or books (including the ones I provided for python) written or rewritten in whatever language you are using for this project.

Deadlines: Midnight of Wednesday November 18 if you wish me to give you feedback on your report and code before you resubmit a final version. Midnight of **Monday November 25 is the final deadline**. Submissions between those dates will be considered final, those thereafter will be graded with zero as per the policies explained in the syllabus. *Start the project early and speak to me after class and in office hours before you get behind in your work.* If you prefer to submit a paper copy of your project report I will need it submitted no later than the end of class on Monday, November 25.

Help and plagiarism: You are welcome to ask me *any* questions to clarify your understanding of the project tasks, the math, or the programming during office hours or by email, but use your common sense and always attempt to solve a problem before coming to me with it. Get started early, and don't wait until a few days before the deadline to realize that you need clarification on a range of issues.

You may discuss the problem solving and any language-learning issues with others but both your code and your report must be **entirely your own work** (the GSU policy on plagiarism applies, as described in the syllabus).

A good student's
submission

Objectives

The primary objectives of this project were to experiment with Bezier curves, numerical differentiation, and least squares minimization using gradient descent. Several computer programs were written and modified to achieve the objectives of this project. In addition to the programs, several plots were made to analyze and compare the output.

Bezier Curves

A Bezier curve is a type of parametric curve that allows complex shapes to be represented in mathematical graphing, computer graphics, animation, and simulation programs. The generation of the curve requires two endpoints to be input along with a pair of control points.

To begin experimenting with Bezier curves it was first necessary to write a simple computer program in Python 2.6 to accept the required data and to generate a plot of the curve. The program that was written had two main functions, get_coeffs and make_bezier.

The get_coeffs function took data describing the endpoints and the control points and used the data to generate a set of cubics that represent a set of coefficients that define the Bezier curve. The cubics were returned by the function and used by the make_bezier function.

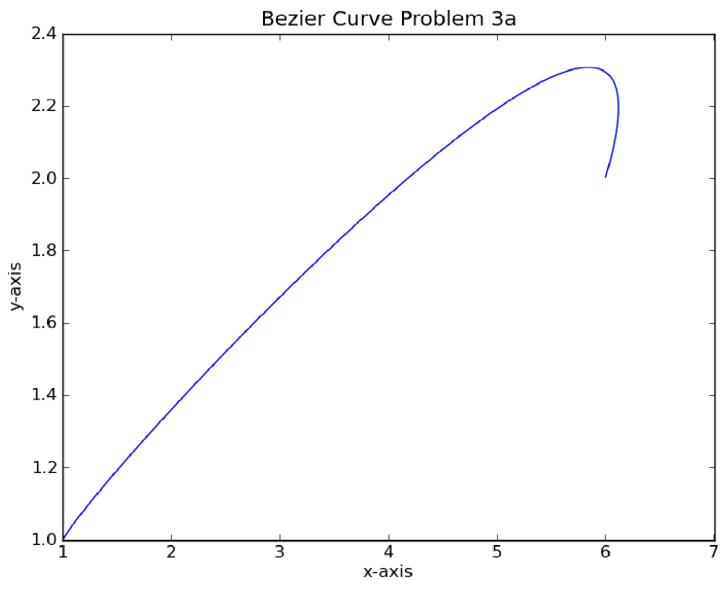
The make_bezier function took in a t value between 0 and 1, and the cubics mentioned above, and generated a pair of (x, y) coordinates that corresponded to the position of the curve at the designated t value.

The output from the make_bezier function was stored in two lists, plotx and ploty, and the data from these lists was used to plot the curves generated.

Once the computer program was written it was tested using problems from page 163 of the textbook. The results follow:

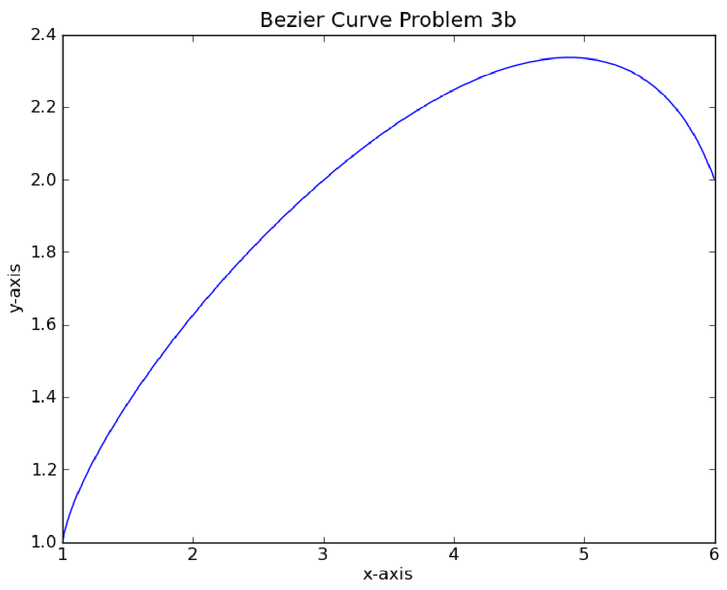
Problem 3a:

Input: Start (1,1)
Control (1.5, 1.25)
End (6, 2)
Control (7, 3)



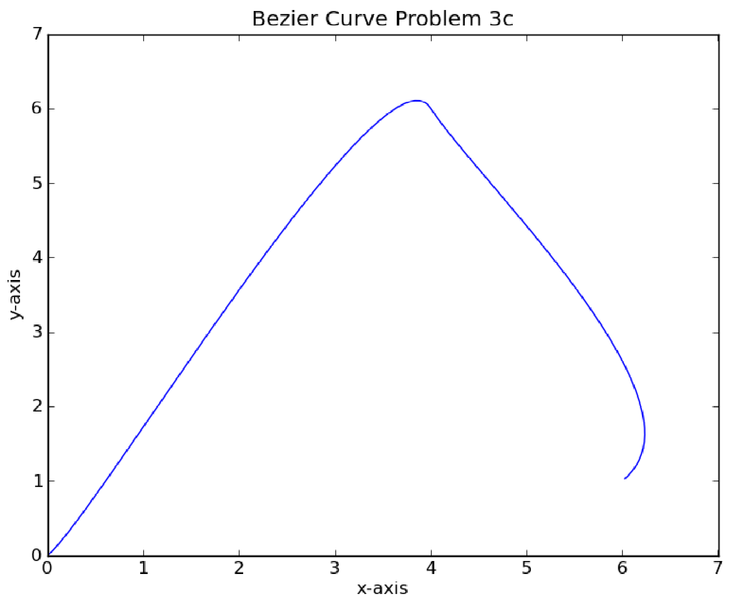
Problem 3b:

Input: Start (1, 1)
Control (1.25, 1.5)
End (6, 2)
Control (5, 3)



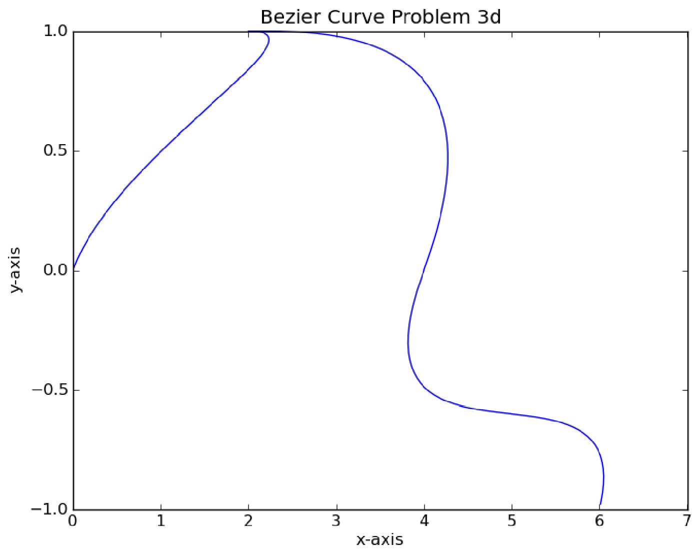
Problem 3c:

Input:	Start (0, 0)
	Control (0.5, 0.5)
	Midpoint (4, 6)
	Entering Control (3.5, 7)
	Exiting Control (4.5, 5)
	End (6, 1)
	Control (7, 2)



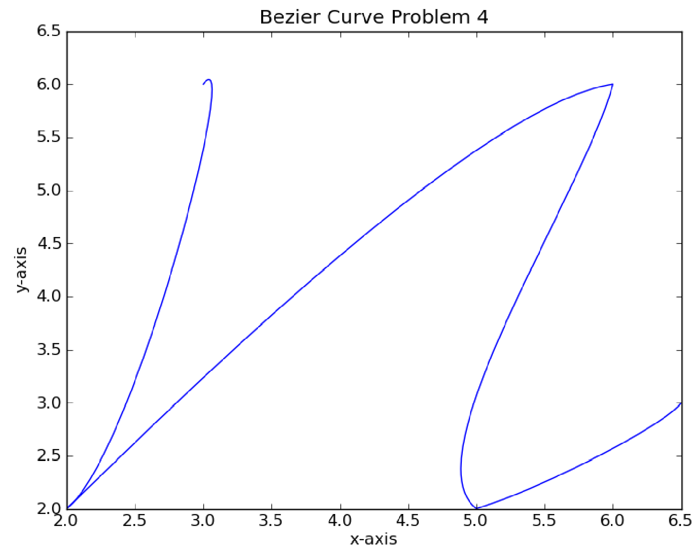
Problem 3d:

Input: Start (0, 0)
Control (0.5, 0.5)
Mid 1 (2, 1)
Entering Control (3, 1)
Exiting Control (3, 1)
Mid 2 (4, 0)
Entering Control (5, 1)
Exiting Control (3, -1)
End (6.5, -0.25)



Problem 4 used the following table for input and was intended to approximate the shape of a cursive N.

i	X	Y	A	B	A’	B’
0	3	6	3.3	6.5		
1	2	2	2.8	3.0	2.5	2.5
2	6	6	5.8	5.0	5.0	5.8
3	5	2	5.5	2.2	4.5	2.5
4	6.5	3			6.4	2.8



The resulting program was able to generate some good results, although the initial runs required some adjustments. The textbook uses absolute coordinates for the control values and my program was written to use relative coordinates. My initial problems were mostly a matter of becoming familiar with the differences between the textbook and the requirements of my code. In the end the Besier curves generated seem pretty good, although the N from Problem 4 is a bit stretched and the third segment makes a rather sharp transition from segment 2. I tried to smooth the curve out by adjusting the control coordinates but the result was always worse than what the table values provided.

Central Difference Formula

The next task was to write a gradient function called df by using a three point central difference formula. The gradient was approximated by using formula 4.5 on page 171 of the textbook. The value of h was specified to be 1e-3 and the function was tested using $f(x) = 2x^2$, and $g(x) = \exp(3x)$, where $x = 3$.

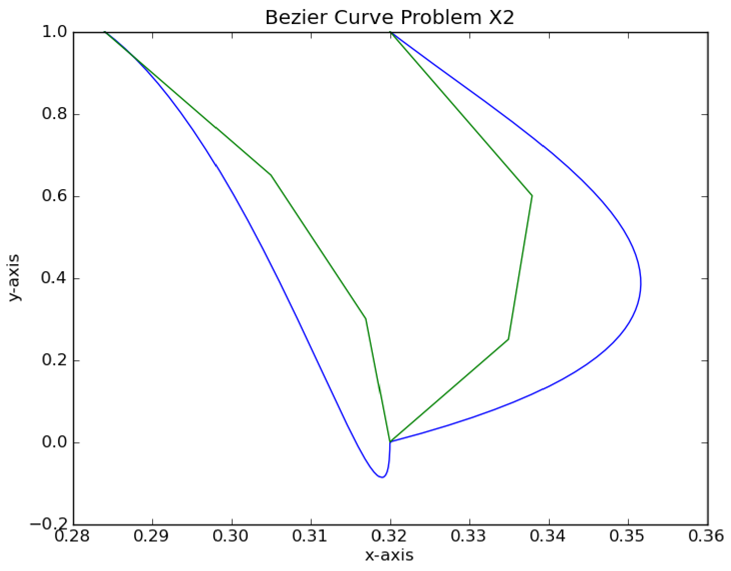
Using calculus the gradient of the first function is 12 when x = 3 and this is exactly the same as the computer program's result. The second function is 3exp(9) which is equal to 24309.25178 and the computer program's result was 24309.2882466. The error associated with the second function was a bit larger than I expected. I was expecting the error to be no larger than h, although the error associated with this technique of numerical differentiation is $O(h^2)$.

Gradient Descent

A small function was written to perform gradient descent using a tolerance value and a loop counter as stopping conditions. The tolerance value was 1e-4 and the loop counter was 500. The function provided to test the function was $E(x) = (x-2)^2$, starting at x = 4. From algebra it can be seen that the global minimum of the equation exists at 2. The function found the value of 2.000332307 in 39 iterations.

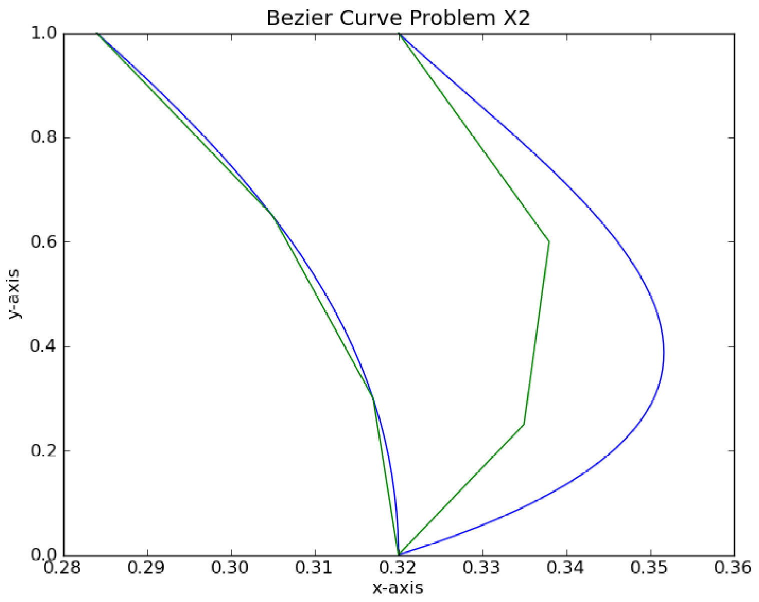
Iterative Optimization

A series of points were provided which represented sample points from a graph in the shape of a cursive v. These points were plotted to show the approximate shape of the figure. Coordinate points for a Bezier curve labeled B1 were then provided to represent the first half of the figure. A partial solution for the second curve labeled B2 was also provided. The original set of points and the two curves are shown in the following plot.



The original sampled points can be seen in green and the two curves B1 and B2 can be

seen in blue. Neither B1 nor B2 fit the given data and after some examination, I discovered that I had made a sign error in alpha1 for B1. The corrected plot shows the result.



It was given that the control coordinate alpha0 was the problem causing B2

to bow outward.

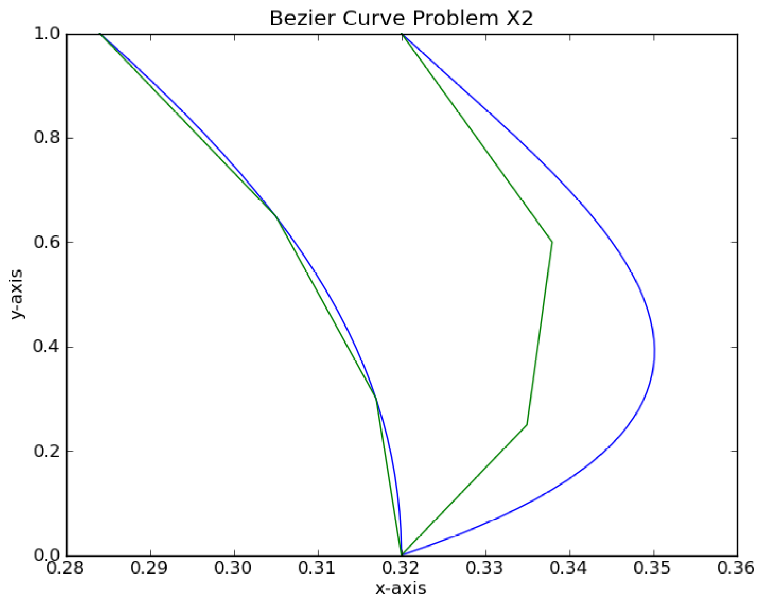
In order to find an optimal solution, a new function was written to calculate values for alpha0 while leaving the other coordinates unchanged. This function was called make_B2 and it accepted the value of alpha0 as input and returned the appropriate cubic value.

An error function war written called Err and it was used to calculate the error associated with various values for alpha0. This was later applied to the gradient descent function to find an optimal solution for alpha0.

An initial value of 0.38 was used for alpha0 and the result was an error value of 0.226725. The gradient at the initial point was found to be 2.6424. This should result in a rapid convergence to an optimal solution because the steeper the gradient, the faster the rate of convergence. This is one of the advantages of the gradient descent method.

I was unable to find an optimal solution using the computer program that I wrote. I think that this is due to errors in the Err function and in the my implementation of make_B2.

The gradient descent function should have been able to quickly find and optimal value for alpha0. I found a reasonable solution by trial and error at (0.351, 0.3). The closest solution that I found with my program is shown in the plot.



This is only marginally better than the original data points. The make_B2 function needs to be rewritten to calculate both x and y values for the control coordinate. In its current implementation it only calculates the x value.

For the most part though, this project yielded some very good results and I think that it was very worthwhile in helping to deepen my knowledge of approximation methods.