Bryan Bridge
7755317
12/11/14

Bézier Curve Flight Path

Problem and Motivation

Our initial goal was to apply the methods we learned in class to track the path of a commercial airliner as it flew. We settled on a data fitting problem using Bézier curves to describe a hypothetical flight path. Noisy data points were generated along known curves, and using the method of steepest descent, we found we could approximate the original curves from these points. This concept would be applicable in a situation where a geolocation device gave inaccurate coordinates, and one wanted to know where the plane had been throughout the duration of its flight. It could prove useful when trying to recover some type of aid drop/debris from a plane, or following a crash or other disaster.

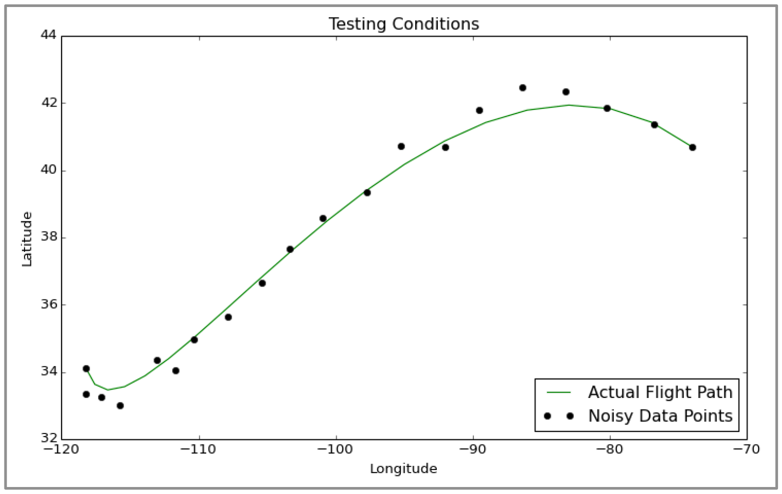
Creation of Flight Path and Data

The initial setup of our problem was fairly straightforward. We used Algorithm 3.6 found in the text to generate the functions for our Bézier curves, and were able to simplify this algorithm as our problem dealt with curves of only two endpoints and two guide points.

In order to have values for comparison and generate noisy data, we first created the true paths. The latitude and longitude of major airline hubs were used as endpoints, and we chose corresponding guide points that would create flight paths that seemed realistic yet gave some complexity for our steepest descent algorithm to sort out. The Bézier algorithm uses these points to return coefficients for the X(t) and Y(t) functions. Those functions depend on a variable 't' on the bound [0,1], so we created an array of values from zero to one with an equal interval. Twenty was chosen as a constant number of increments for all paths. This array served as the "times" that for example a GPS in the plane might ping a radio tower or satellite, and by evaluating it on a given X(t) and Y(t) we had true coordinates from the flight path.

We simulated noise by applying a method that returned random values over a Gaussian distribution; the true coordinate arrays served as the mean values for the distribution. This method allowed us to a set a standard deviation, and we settled on a constant for all of our test cases. We assumed that if a real world geolocation device had some level of error it would still have an expected degree of accuracy, and that it would be absolute; that is, it would not scale with larger/shorter flight distances. In addition, for the sake of realism the endpoints were kept the same, as we would know the takeoff and landing coordinates. When generating this noisy data as well as the true flight paths, we output a plot displaying each. This is what allowed us to decide on the standard deviation that seemed appropriate, as well as pick guide points that gave us our desired paths. An example from our first data set, Los Angeles to New York, is shown in Figure 1.

Figure 1



Approximation of Curve by Steepest Descent

We needed to determine the flight path from a noisy data sample. This meant using the method of steepest descent. In a nutshell, we found guide points for a Bézier curve that produced the minimum total error between the sample points and all of the curve's correlating points of the same time 't'.

Describing how we implemented this method is easiest if we start with the Mean Squared Error. The MSE, shown below in Equation 1, is the function we minimized and in doing so found the optimal guide points for the Bézier curve.

Equation 1

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\Big(\big(u(t_i), v(t_i)\big) - \big(X(t_i), Y(t_i)\big)\Big)^2$$

While (u(t), v(t)) is simply a coordinate from the sample data, this function also depends on two functions within it, X(t) and Y(t). These two functions, shown in Equations 2 and 3, generate the points along a single segment Bézier curve.

Equation 2

$$X(t) = x_0 + (3(x_0^+ - x_0))t + (3(x_0 + x_1^- - 2x_0^+))t^2 + (x_1 - x_0 + 3x_0^+ - 3x_1^-)t^3$$

Equation 3

$$Y(t) = y_0 + (3(y_0^+ - y_0))t + (3(y_0 + y_1^- - 2y_0^+))t^2 + (y_1 - y_0 + 3y_0^+ - 3y_1^-)t^3$$

$*(x_0, y_0), (x_1, y_1)$: *endpoints*; $(x_0^+, y_0^+), (x_1^-, y_1^-)$: *guide points*

Here we can see how the Bézier curve is dependent on the endpoints and guide points. Since the endpoints are fixed in our problem, the only values we could vary to minimize the error function in Equation 1 are those for the guide points.

Once we had unraveled the MSE function the next step was to use it to seek out the values of $x_0^+, y_0^+, x_1^-, y_1^-$ (the guide points) that would minimize it. The steepest (or gradient) descent method does this by using the gradient of the function to incrementally move toward a minimum for all variables. Equation 4 shows the gradient as we solved it, broken into the four partial derivatives:

Equation 4

$$\nabla E = \nabla MSE = \begin{pmatrix} \frac{\partial MSE}{\partial x_0^+} = -\frac{2}{n}\sum_{i=1}^{n}(u(t_i) - X(t_i))(3t - 6t^2 + 3t^3) \\ \frac{\partial MSE}{\partial y_0^+} = -\frac{2}{n}\sum_{i=1}^{n}(v(t_i) - Y(t_i))(3t - 6t^2 + 3t^3) \\ \frac{\partial MSE}{\partial x_1^-} = -\frac{2}{n}\sum_{i=1}^{n}(u(t_i) - X(t_i))(3t^2 - 3t^3) \\ \frac{\partial MSE}{\partial y_1^-} = -\frac{2}{n}\sum_{i=1}^{n}(v(t_i) - Y(t_i))(3t^2 - 3t^3) \end{pmatrix}$$

**From here on MSE is referred to as E for simplicity*

Since X(t) and Y(t) are not expanded in Equation 4 it is less apparent, but as with the MSE each derivative in the gradient depends on the guide points. For any set of guide points, the gradient returns the slope of the tangent line for each guide point variable. This means we can move in the negative direction on each tangent slope and we will approach the minimum of the error function. It should be noted that this is the shortest route to the minimum, hence the name "steepest descent." This is hard to visualize, a gradient with only two variables and their tangent lines could be plotted on a 3D graph and we'd see their slope. Here there are four variables, but the concept is no different. If all of the tangent slopes approach zero, we will know we have reached the minimum of the error function.

The algorithm we wrote to do this portion of our problem employed a single method that computed the gradient and incrementally updated the guide points. Equation 5 succinctly states the method of steepest descent and what this algorithm did:

Equation 5

$$q_{n+1} = q_n - s\nabla E$$

**q = $(x_0^+, y_0^+, x_1^-, y_1^-)$, s = step size*

As input the method required the current guide points, the noisy data sample, and a step size. We settled on a step size of 0.5 because values less than that did not seem to improve the results, and the algorithm was fast enough that it need not be any larger. The method returned the updated guide points, as well as a sum of the absolute values of the gradient evaluated at the current guide points. That sum was used to stop the method from iterating. Our method ran in a while loop with the clause that the returned sum must be greater than 1×10^{-3}.

Results and Conclusion

We found that our algorithm could approximate the flight paths with surprising accuracy. Throughout three test cases, with three noisy data samples for each, the guide point values never had an absolute error of more than 1°. In addition, for each test case we ran the true data through our steepest descent algorithm and the absolute error was always in the thousandths or less. Below, figures 2a, 3a, and 4a compare our approximated trajectories (for one noisy data set) to the actual one, and 2b, 3b, and 4b show the console output for the actual errors. In addition to the actual error we computed the relative error, but realized that it was meaningless, as the values for the guide points are not a magnitude,

Figure 2a

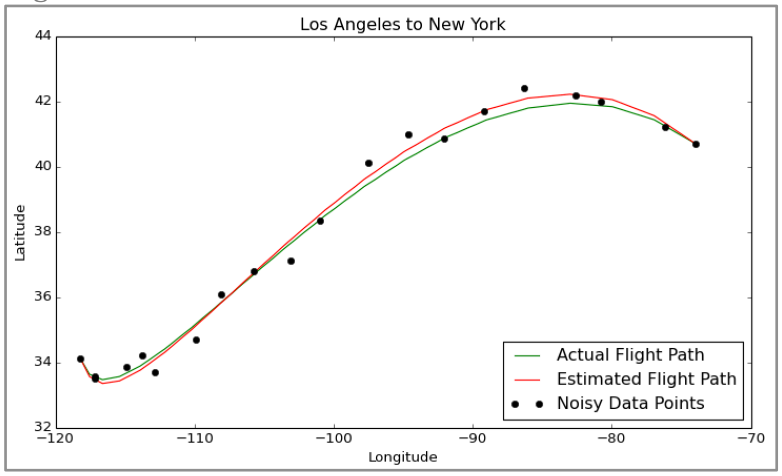


Figure 2b

```
- Absolute Error -
******************

Est. from noisy data set 1:
x_0_plus  0.019858756105
y_0_plus  0.572533801182
x_1_minus 0.0593832470548
y_1_minus 0.927702652504

Est. from noisy data set 2:
x_0_plus  0.613911447159
y_0_plus  0.0927638280969
x_1_minus 0.57184639794
y_1_minus 0.433150488485

Est. from noisy data set 3:
x_0_plus  0.031849631692
y_0_plus  0.10213714128
x_1_minus 0.356760600545
y_1_minus 0.70120260618

Est. from clean data set:
x_0_plus  0.00706070775401
y_0_plus  0.0050855758498
x_1_minus 0.00706070775441
y_1_minus 0.0050855758499
```

Figure 3a

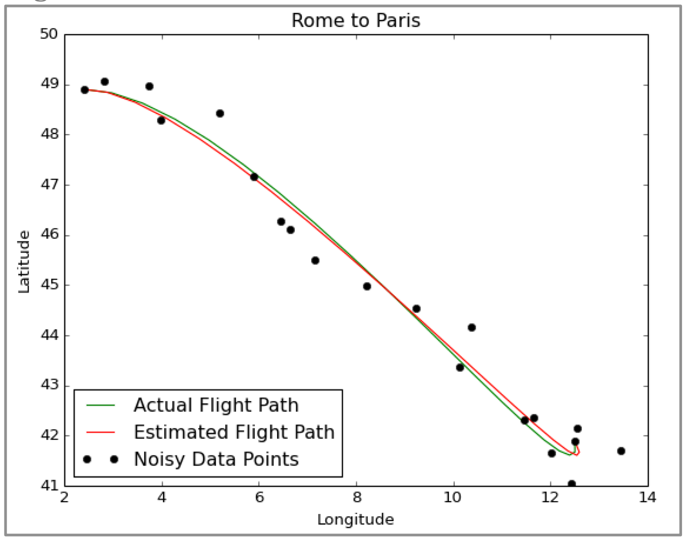


Figure 3b

```
- Absolute Error -
******************

Est. from noisy data set 1:
x_0_plus  0.669230291293
y_0_plus  0.0237809302593
x_1_minus 0.682438387137
y_1_minus 0.0749348289548

Est. from noisy data set 2:
x_0_plus  0.0202154598654
y_0_plus  0.390782480148
x_1_minus 0.0198839762212
y_1_minus 0.802029940062

Est. from noisy data set 3:
x_0_plus  0.459613890743
y_0_plus  0.808646041422
x_1_minus 0.215459981154
y_1_minus 0.456602631774

Est. from clean data set:
x_0_plus  0.0052716148444
y_0_plus  0.0067777905142
x_1_minus 0.00527161484442
y_1_minus 0.0067777905143
```

Figure 4a

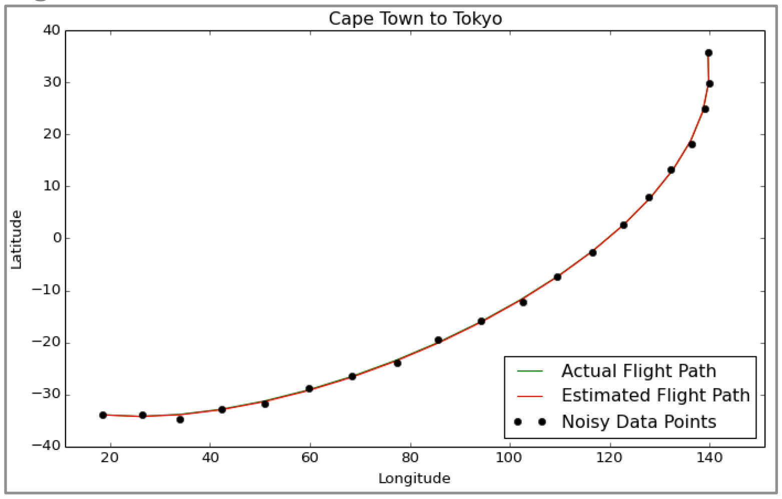


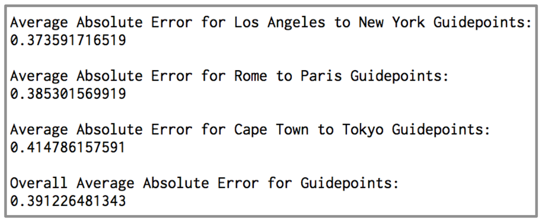
Figure 4b

```
- Absolute Error -
******************

Est. from noisy data set 1:
x_0_plus  0.711988962852
y_0_plus  0.193289423753
x_1_minus 0.300586238176
y_1_minus 0.0871982641749

Est. from noisy data set 2:
x_0_plus  0.203698553585
y_0_plus  0.65391515883
x_1_minus 0.06317625219
y_1_minus 0.472845635055

Est. from noisy data set 3:
x_0_plus  0.782311940763
y_0_plus  0.170888848645
x_1_minus 0.99068883727
y_1_minus 0.3468457758

Est. from clean data set:
x_0_plus  0.0083608566631
y_0_plus  0.003878166969
x_1_minus 0.00836085666299
y_1_minus 0.00387816696929
```

but a coordinate.

In addition to finding the absolute errors their averages were found, for each flight and overall - Figure 5 shows the results from the python console. We can expect an average error of < 0.4°, < 24’ , or < 28 miles along the estimated paths using our methods. We must keep in mind our results would not be accurate near the poles, however. Latitude stays roughly the same in distance per degree all over the globe, but longitude shrinks: from ~69 mi./deg. (similar to latitude) at the equator, to 0 mi./deg. at the poles. This happens slowly initially, but as a flight path approached the poles estimates of distance would be unreliable.

Figure 5



```
Average Absolute Error for Los Angeles to New York Guidepoints:
0.373591716519

Average Absolute Error for Rome to Paris Guidepoints:
0.385301569919

Average Absolute Error for Cape Town to Tokyo Guidepoints:
0.414786157591

Overall Average Absolute Error for Guidepoints:
0.391226481343
```

Our project only covered one standard deviation for the amount of noise in the sample data. Given more time additional trials could be repeated with different levels of noise - for comparison and to test the limits of our algorithm. It is clear at this point that this is the determining factor in our error, and it would be insightful to see how the error scales with increasing noise. Regardless, I feel our algorithm performs as we had intended and serves as an example for how we could use the methods we learned in class in an exciting, real-world example.