Applied Math: Assignment 01

Lily Dao, Noah Rand, Braden Vaccari

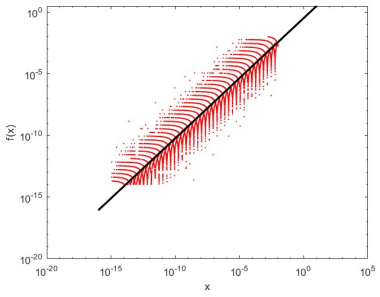
September 19, 2024

Contents

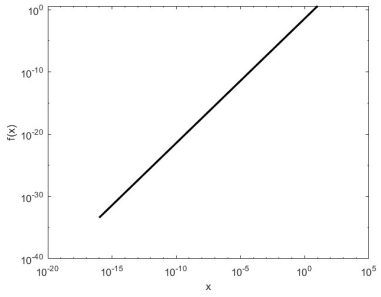
1 Time Spent

- Lily Dao: 26 hours
- Noah Rand: 28 hours
- Braden Vaccari: 15 hours

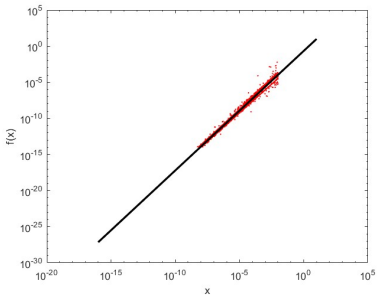
2 Convergence Experiment Plots



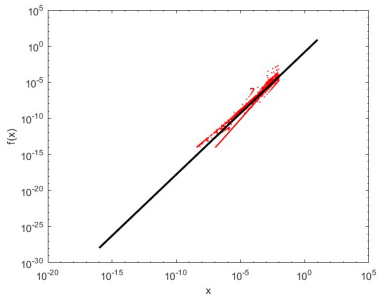
(a) Bisection Root Solver



(b) Newton's Root Solver



(c) Secant Root Solver



(d) Fzero Function

3 Predicted vs. Measured Values

Predicted vs. Measured Values				
Method	k Predicted	p Predicted	k Measured	p Measured
Bisection	0.5000	1	0.3250	0.9741
Newton's	-0.0373	2	0.0373	1.9989
Secant	NA	1.6180	0.2182	1.6558
Fzero	0	0	0.1544	1.6963

For Newton's method, our predictions for p and k are very close to the actual data. This makes sense because the data fits the line of best fit nearly perfectly, and very closely mimics the ideal case. Newton's method also converges superlinearly because it has a p value greater than 1.

There is not a method for calculating the predicted k value for the secant method, however our predicted p value was very close to the actual k value. The prediction was not as accurate for the secant method because the convergence data has a larger spread than Newton's method. However, the predicted value was still very accurate. Like Newton's method, the secant method also converges superlinearly because its p value is greater than 1.

4 Discussion

Newton's Method vs. Bisection Method

Comparing the speed of Newton's method to the bisection method, it's easy to say that Newton's method is much faster. This is because Newton's method has quadratic convergence while the bisection method has linear convergence. With guesses close to the root, if Newton's method took n steps to converge, the bisection method would take approximately

$$1.8 * n^2$$

steps to converge.

Secant Method vs. Newton's Method

With guesses sufficiently close to the root, we found that Newton's method converged in the same number of steps as secant method. This makes sense for close guesses in simple functions. However, given guesses further from the root, Newton's method will converge quicker than secant method. If Newton's method converged in n steps, Secant method may converge in

$$1.2 * n$$

steps.

Fzero Algorithm

After examining the documentation and analyzing the convergence plots, we found that fsolve uses a combination of secant and bisection methods. The values of k and p for secant and Newton's solvers are very close to each other, especially when considering the other functions. Additionally, the convergence

plots are visually similar, with a similar amount of variation in the same locations. We know that fsolve also uses bisection because it is able to find the zero from any point within the sigmoid function (see below).

Quadratic Function on Newton's and Secant Method

The quadratic function with a global minimum at the root breaks secant solver, Newton's solver, and fsolve. This is because as the function approaches its root the slope approaches zero, moving the x-intercept of the tangent (or secant) line, (x) very far from the root. This continues in a cycle until the new point moves so far from the root, that it results in a near-infinite value for f(x).

Sigmoid Convergence

Newton's solver is not effective at all points in a sigmoid function. This is because, in areas of the function with a near horizontal line, the tangent line intersects the x axis very far from the root. Far from the root, the line is closer to being horizontal, escalating the problem into an infinite loop.

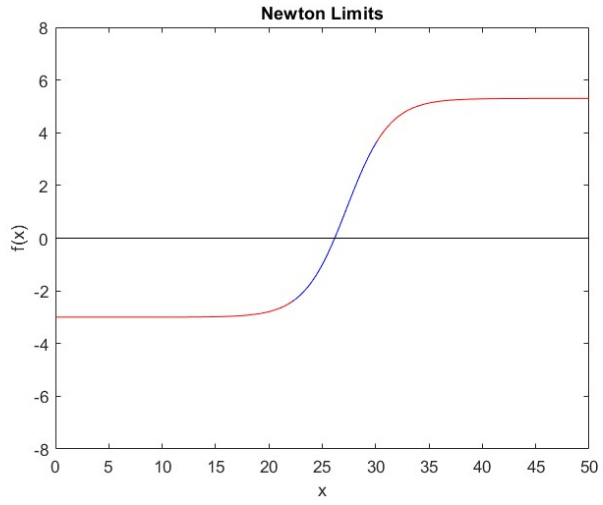
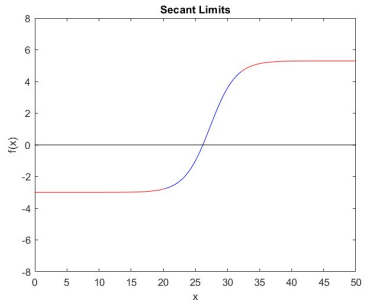
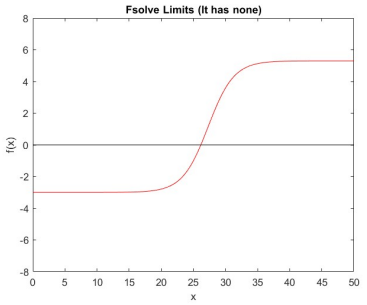


Figure 2: Newton's Solver limits: The solver is effective at finding the root within the blue section.



(a) Secant Limits: The solver is effective at finding the root within the blue section.



(b) fsolve Limits: The solver is effective at finding the root within the blue section.

Secant method has a similar problem to Newton's method. Due to it using one of the points from before to draw the next secant line, it is functional farther from the zero than Newton's solver (see the figures). However, the gain is marginal so the solver runs into the same issues. Fsolve, on the other hand, is seemingly able to handle every point on the graph. With fsolve, we tried points very far from the zero, even out to 26 units away, with no problems. We think this is because fsolve uses a combination of bisection and secant solvers. Wer believe fsolve by default tries to use secant, but if it runs into an error, it switches to bisection. This would explain why it is able to determine zeros when the other functions cannot, and the vertical string at the beginning of running the function.

Bisection continues to function with the sigmoid function because it simply reduces the space between guesses until the difference between its two guess points is minimized. With this in mind, even if two guesses were on perfectly straight lines, extremely far from the root, it would still converge as long as the signs of the guesses are opposite.

5 Strengths & Weaknesses

For the Bisection solver, if the function has both positive and negative values (and you use one of each as the input), it is guaranteed to find at least one root. Although certainty of finding a root is higher, the convergence is slow and functions whose roots are at a global maximum or minimum do not converge. .

For Newton's solver, unlike the Bisection solver, this method is fast to find the convergence. It may be fast to find the convergence but it is not guaranteed to find it. This would be unfavorable if one wanted the convergence 100% of the time. minimum do not converge. .

Secant solver, combines the speed of Newton's method with the versatility of the bisection method. Secant converges superlinearly, but at a lower rate than Newton's method. Becuase of this, secant method can find zeros in fewer steps than the bisection method. Secant method is also able to find the root with less precise guesses than Newton's method (see the sigmoid section above). Unfortunately, there is no free lunch, as secant method still struggles to converge both when the root is a at a local minimum, and with guesses far from the root in a sigmoid function.

6 Tumbling Egg Problem

The objective of the tumbling egg problem was to plot the trajectory of an egg, having the egg stop at collision with a set wall or ground. To start the problem, we were given a function that gave the parametric curve for the shape of the egg.

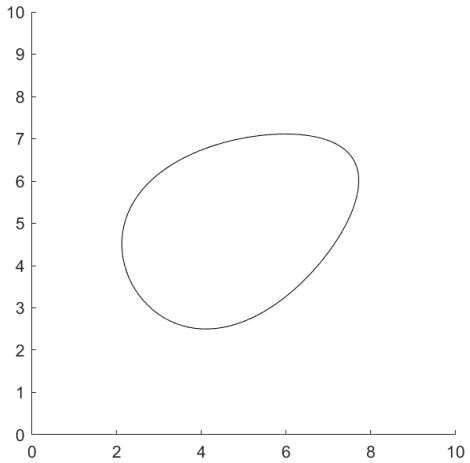


Figure 4: Egg plotted using function given for parametric curve.

The next thing we did was find a bounding box for the egg. The bounding box for the egg would be a rectangle with sides parallel to the x and y-axis. The purpose of the bounding box is to help us find the point in which the egg would collide with the wall or the ground. We did this by using the root finding algorithms developed above, specifically with the secant solver. The bounding function takes in the x, y, and theta values for the position of the egg, as well as the parameters to make the egg curve. The function outputs the upper and lower bounds in the x-direction and y-direction. By using the secant solver with the given inputs, we are able to find the roots and therefore, find the upper and lower bounds. Plotting these lines give us a box that has the egg completely enclosed but leaves no additional space.

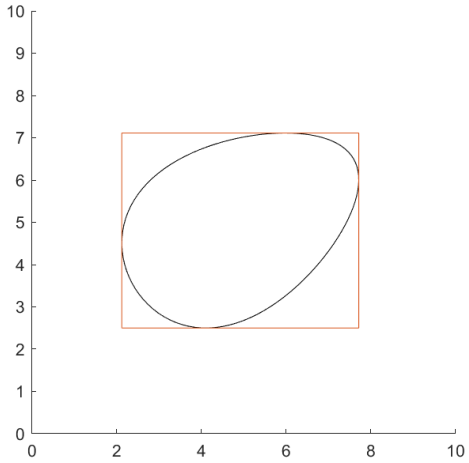


Figure 5: Egg contained in bounding box calculated using secant solver.

After this, we needed our egg to follow a trajectory. We chose a few equations to influence the x, y, and theta position of the egg as it followed the trajectory. The x and y values gave us the parabolic trajectory of the egg and the theta value dictated the rotation of the egg along its path.

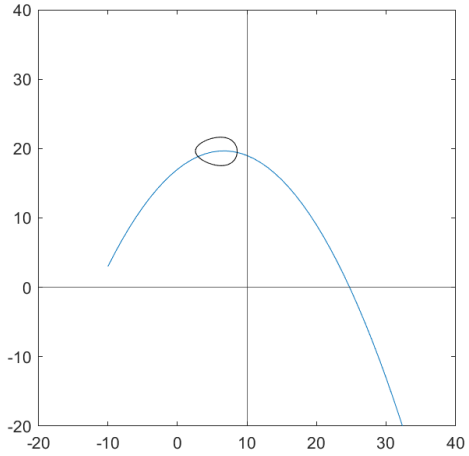


Figure 6: Set trajectory of egg before collision.

The last step was determining when the egg collides with the wall or ground. The position of the wall and ground are determined by the user. This involved using the bounding box function we had used previously to find when the distance between the bounding box and the wall or ground was equal to zero. In a similar manner to finding the bounding box for the egg, we used a root finder to find the roots of the trajectory. This would help us find when the egg would collide

with the wall. For this application, we decided to use the bisection solver. We found that this would make the most sense for a parabolic function that we would expect to cross y = 0 after x = 0. The output of the collision function gives the time that the egg will collide with the ground and with the wall.

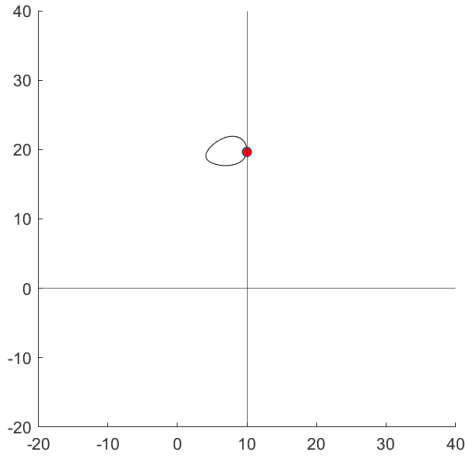


Figure 7: Egg colliding with wall.

Once we determined the trajectory of the egg and when it would collide, we were able to animate the motion of the egg. We called all of the above functions to animate the motion of the egg. We also plotted the wall and ground values on the same plot. To have the animation pause at the collision, we have a for loop that only runs until the time of the collision. To determine the time of the collision, we took the lower time value output of the collision function. This would give us when it would collide first. We also had it plot a point at this point of collision by taking the point of the trajectory at the stop time and plotting it when the collision happens.

7 Video and Code

Click here for a link to our git repo

Click here for a video of the egg animation