Michael Telahun

CECS 622- Spring – HW2

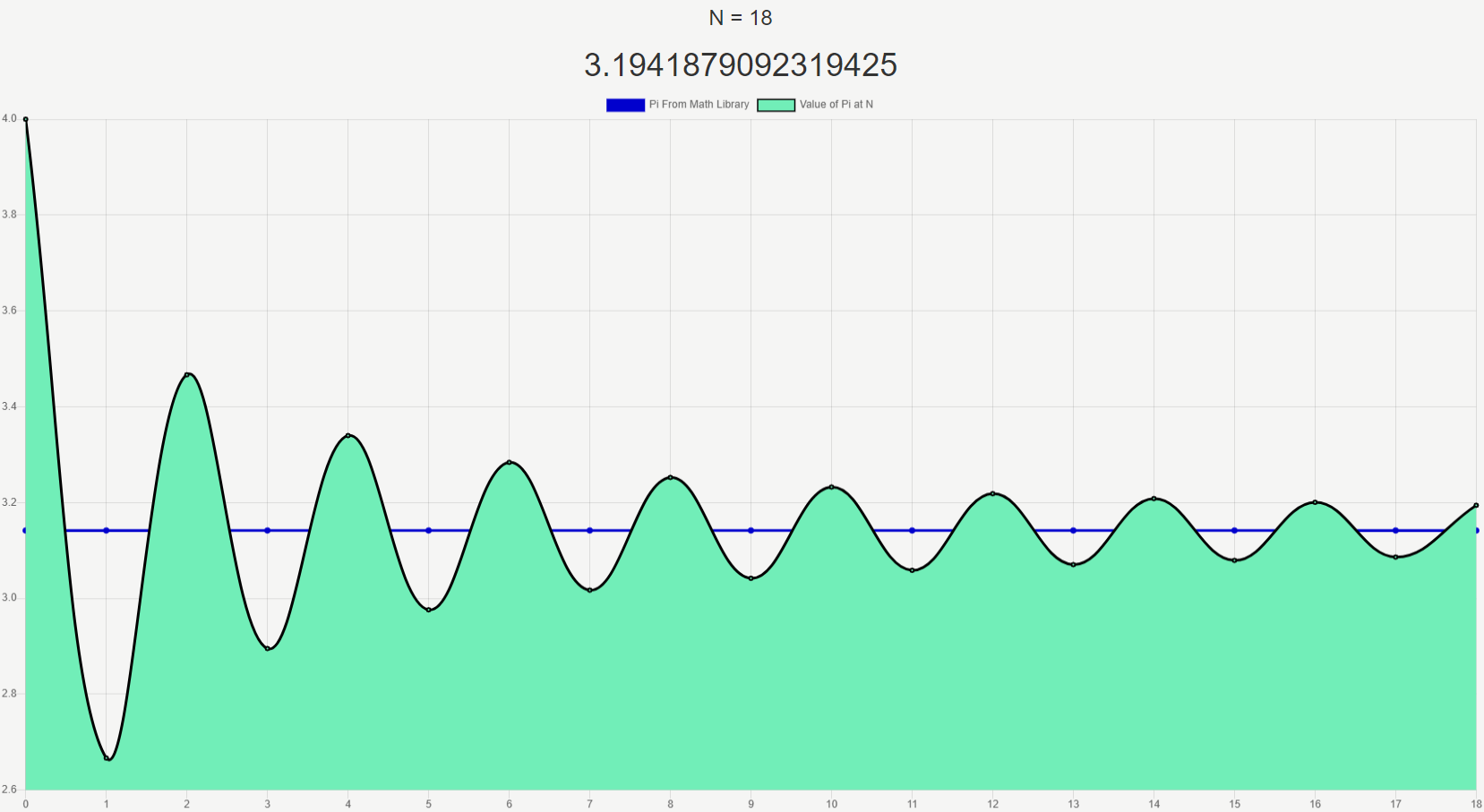
Dr. Elmaghraby

**Problem 1:**

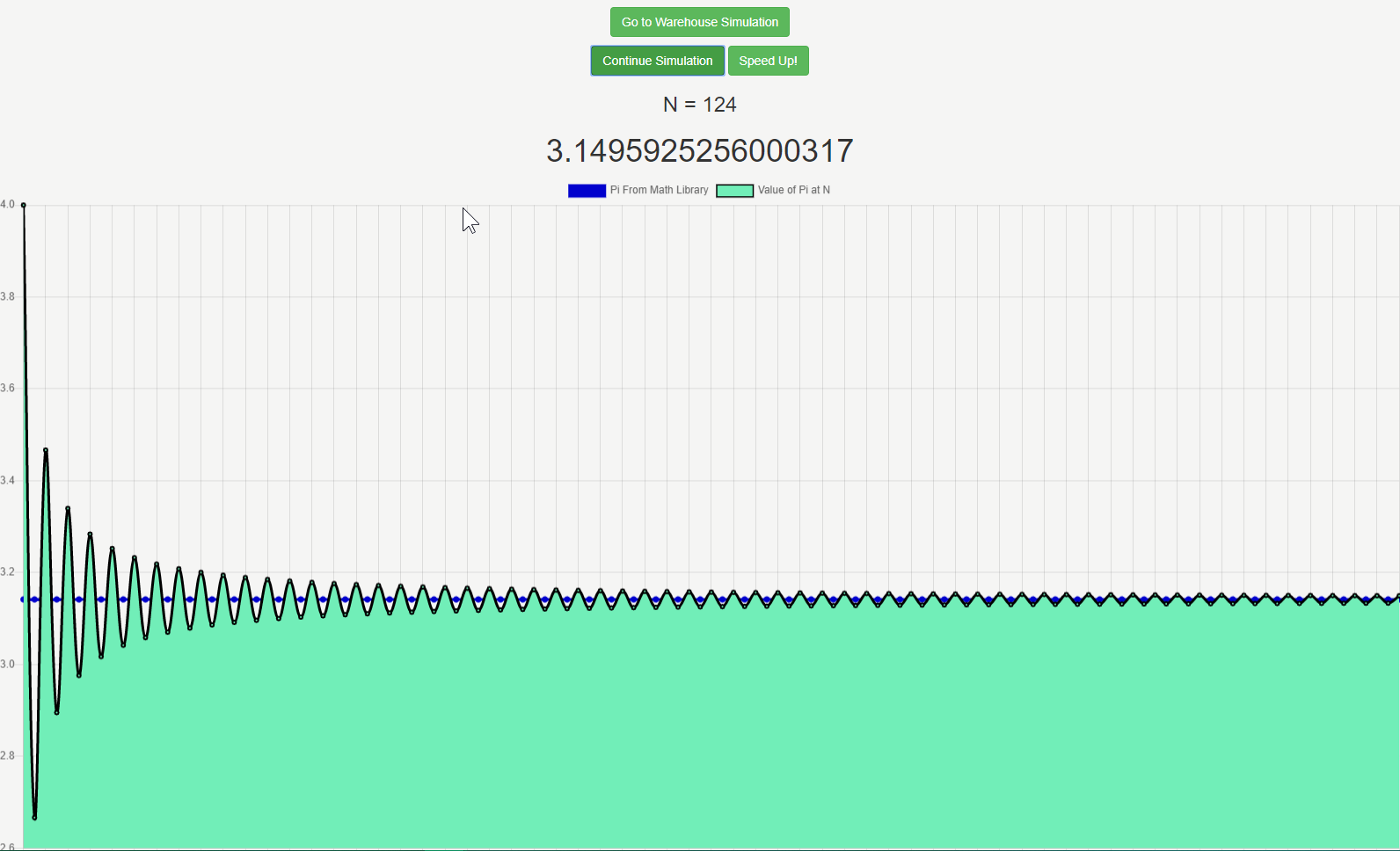
I thought this problem was fun and I enjoyed thinking about the solutions that I could use to display my simulation. For this problem I thought it interesting to watch the simulation of the values at N converge. In my program I plotted the points for calculated values of pi at N. The y-axis is the value of pi and the x-axis is the value of N. I used the formula

to calculate pi. I drew a line at where pi is defined in the math library for JavaScript, 3.141592653589793, which I will consider as pi for this problem. When the simulation starts, visually it looks like we are really far away from pi, and the truth is we are, if we consider how many values of N it would take to be close to the pi value in the JavaScript library. However, it also looks like it quickly converges after a small number of N. Because of this, I made an animation to change with larger values of N such that the points would reflect a more meaningful convergence. Essentially, I zoomed in on the points as N grew larger to reflect the decimal places that are still far from the true value of pi. This can be seen in the figures below. This was to show that relatively speaking a simulation is only as good as the way it is visualized. In this case calculating pi clearly converges but the rate can easily be misjudged from the initial visual inspection and that doesn’t tell the right story.

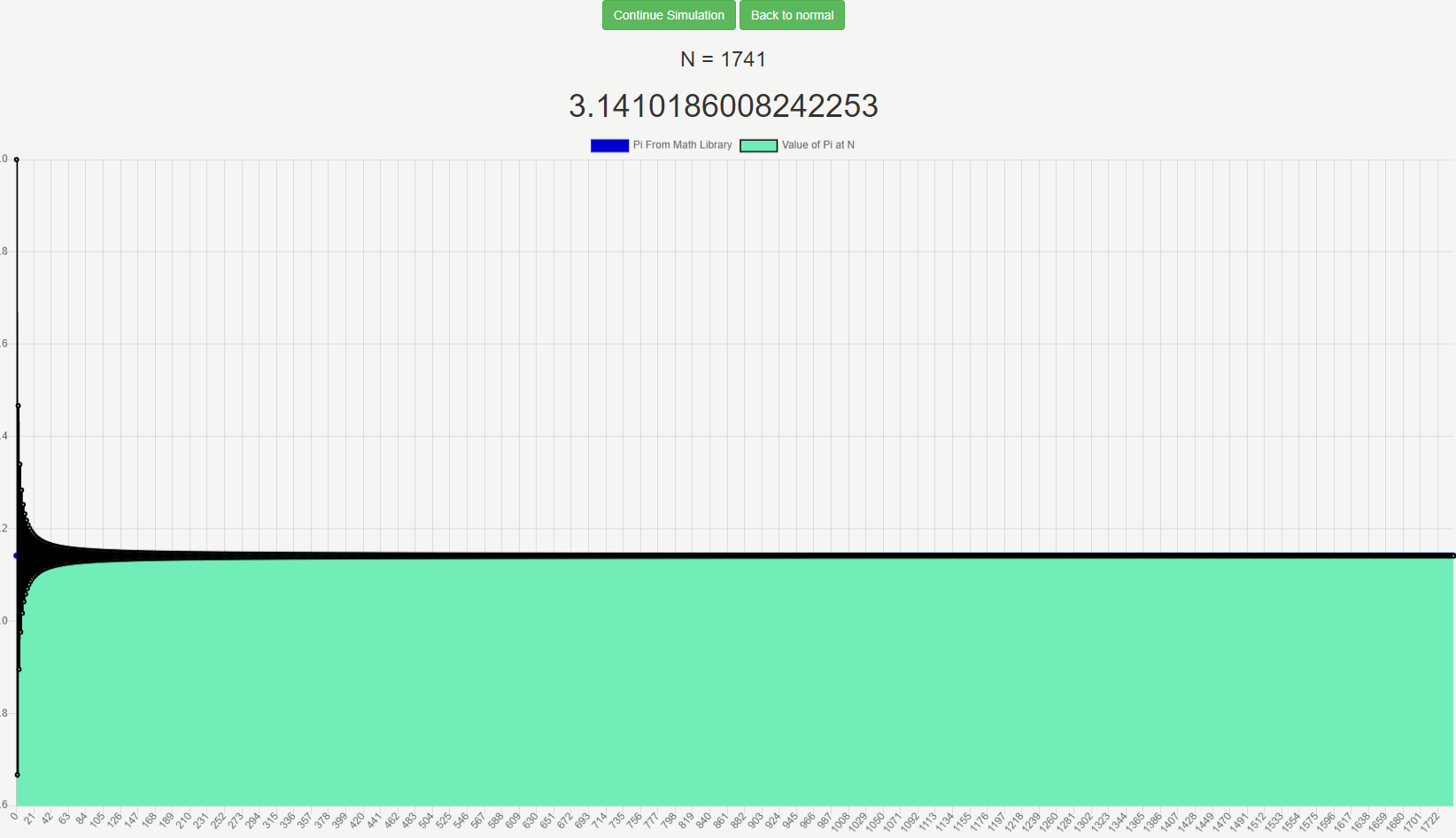
I wanted to include some of the other ways I came up with for plotting the values of pi, but I did not actually implement them. I thought about plotting the points using the area of a circle and showing the sharpness or irregularity of the circle at different values of N, but I struggled to come up with an appropriate way of zooming in on the circle especially for large values of N. I also had an idea to highlight the values in a histogram ranging from 0-9 of the values generated at N for the values of pi when they appear. For example, at 3.1 the 3 and 1 would be displayed in the histogram. To do this for repeated numbers like 3.141 (the two 1s) in the histogram would have been related to a color with a gradient, so a darker color would indicate the number of repeated digits. Similarly, the occurrence of values could have been represented by the height of the histogram. Another way of showing this was to draw a circle or a ten-sided polygon and plot 0-9 at equal distances and have an edge connect one value of pi to the next. So when the value at N is correct the edge would highlight. For repeated values this would be a little tricky because it is important to see each edge distinctly, because it is like a graph. I thought of an offset for each repeated value (say 2 pixels), from the last value drawn. I think this would have been the most interesting visualization to see. The last three solutions all require the value of pi to be known beforehand. I liked the graph I used in my example because we can see it converge towards pi at each value not just at the correct values. Also, zooming tells a much more interesting story.

**These images are without zooming. As you can see it looks like it quickly converged**.

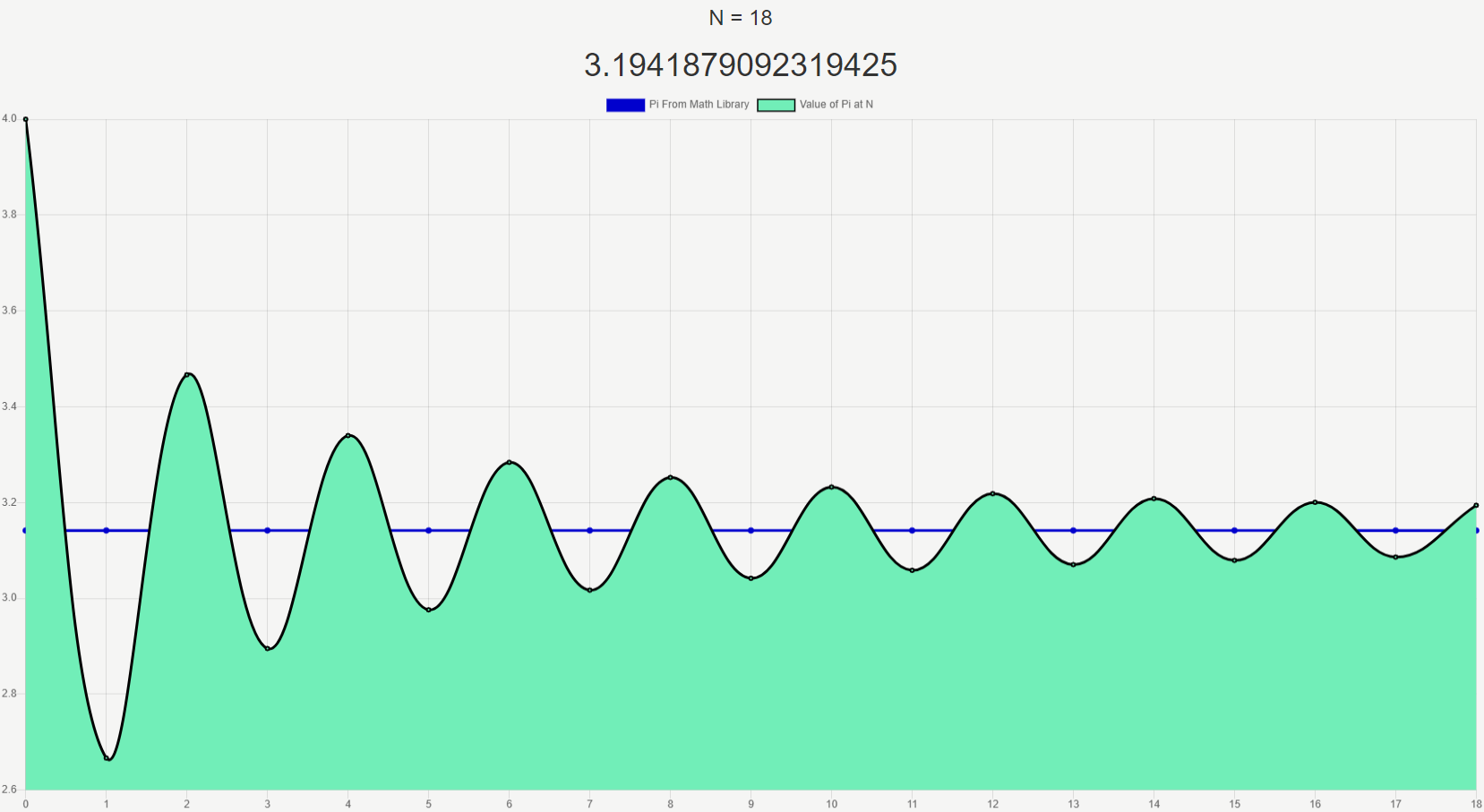
**At about N = 18 we get 3.1 from pi.**



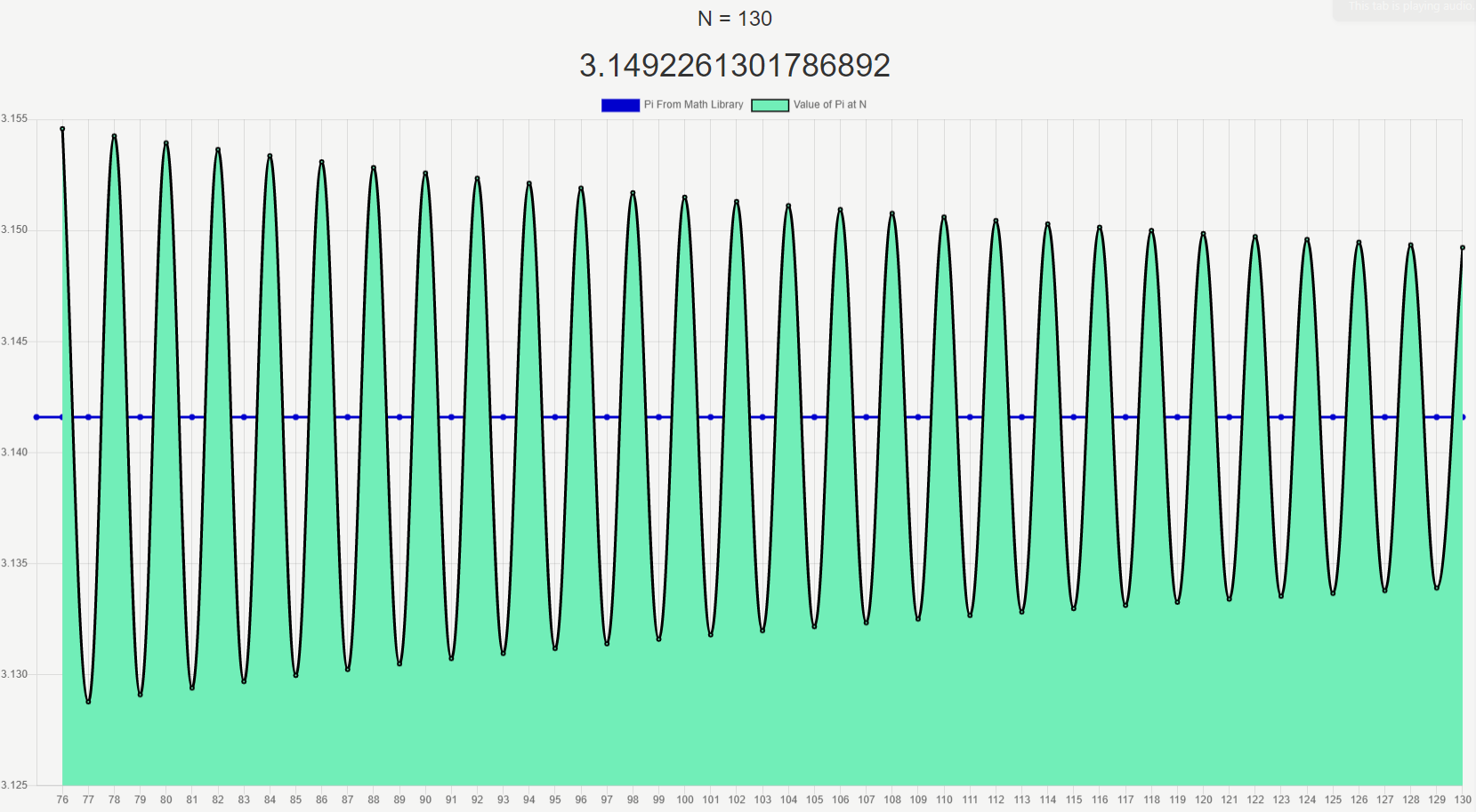
**At about N = 124 we get 3.14 from pi which is often used as a shorthand for pi.**



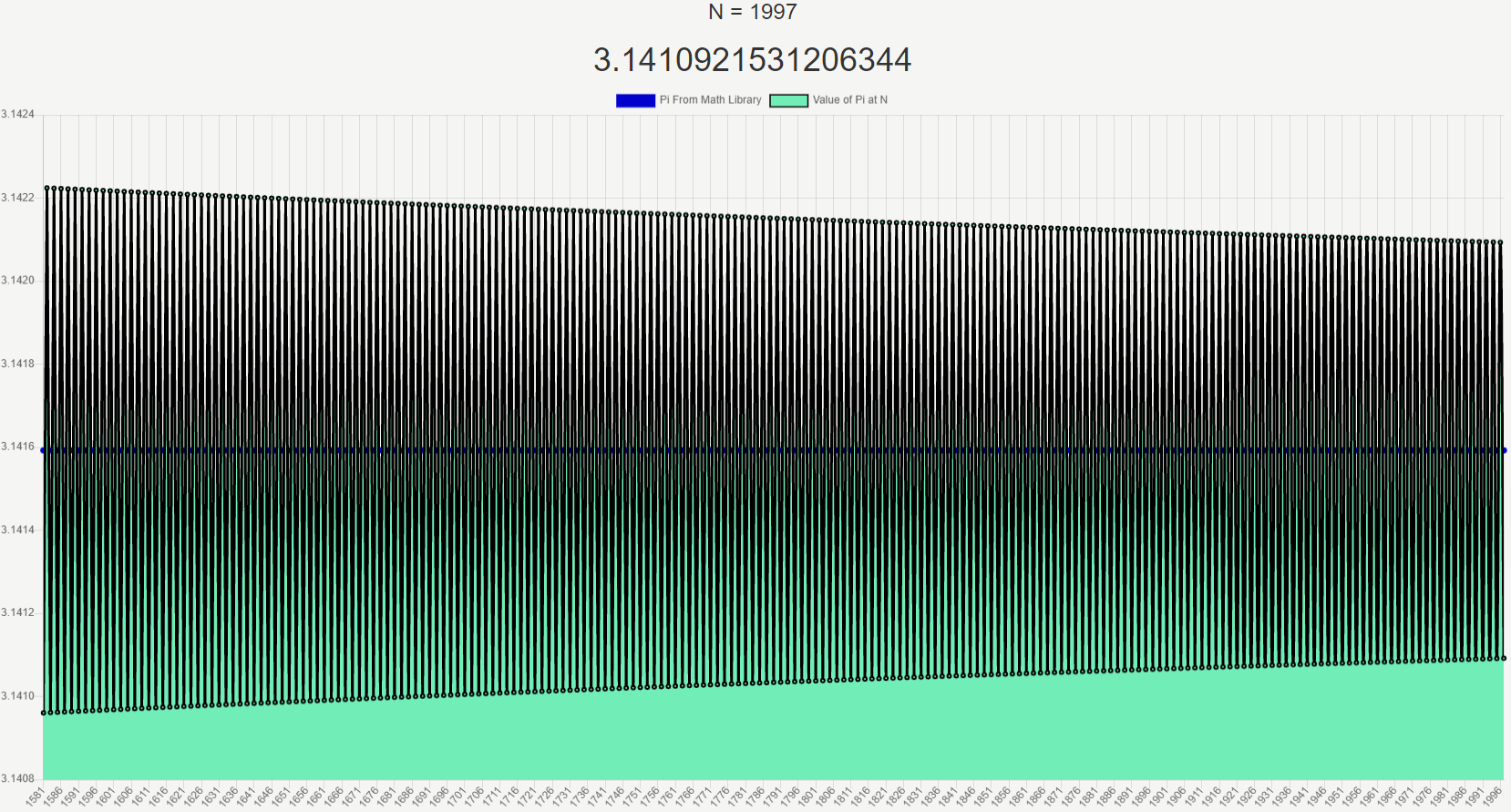
**At about N = 1740 we get 3.141.**

**With zooming we can see it tells a different story**.

**This is the same from above.**



**Here is the plot at N = 130 for pi. Clearly it has not converged as quickly as we assumed.**



**Here is the plot at about N = 2000. The story of pi is much clearer.**

And the zooming makes a lot of sense because we know pi is an irrational number which can be approximated/calculated infinitely. The way I plotted the graph was to relatively update the display so that larger and larger values of N would still visually converge towards pi.

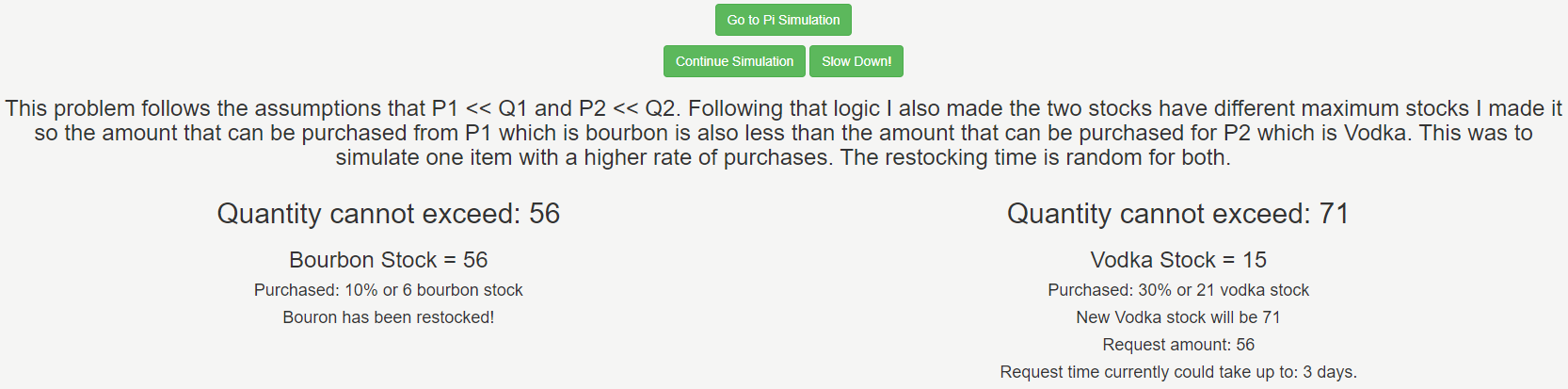
**Problem 2:**

For my simulation I simulated bourbon stock as ( and vodka stock as . In my solution I followed the assumptions for and . I expanded on this by having less than , this was to simulate the two different items with potentially two different purchasing amounts. The idea was to have one item be restocked more frequently. To do this I gave the two stocks random values with an offset of 30, where vodka was the higher. Each iteration a percent of each item can be purchased. For bourbon this amount was 0-20% and for vodka it was 0-40%, this is a percentage of the total stock not current stock. If it was a percentage of the current stock, I think it would have been a diminishing returns game which did not really make sense in my simulation. However, it is a valid approach to the problem and in the real world.

In my scenario there is a chance that the stock will go to zero however it did not happen frequently for the bourbon stock or fell below 60% then a purchase request was sent. This means there is a low likelihood of the bourbon ever running out. However, for vodka the restock request is signaled at 40% of stock remaining. This was to drive a kind of supply and demand scenario. We want to keep the vodka stock at the threshold and sometimes even run out of the stock because we know the need for vodka in this simulation is higher. When we consider the percentages for the stock hitting zero there are a few bad cases such as: (1) it can take 4 days to restock and each day someone wants to purchase 30-40% of the stock, (2) it could take 2 days to restock and someone wants to purchases 30% of stock on both days. In both cases we could either say we ran out of stock, the stock is on the way, or sell the remaining stock. In my case I chose to say we ran out. This is because I want to give the customer the opportunity to buy bourbon instead. The bourbon was the stock that gets purchased less frequently. I wanted to simulate a little bit of supply and demand so I could in a way drive people to the “second best option” when “the good stuff” is out.

In the screen shot you can see on the left that the bourbon stock was just recently restocked but a purchase also is also being requested for 6 bottles of bourbon stock. This is because a purchase can take place on the same day as the restocking day. On the right you can see the vodka stock has just created a request for restocking. It requests 56 bottles which is the difference in stock at the time of the request. The request time is a roll between 1-4 days for each restocking event for each stock. I expanded on this by giving the “manager” a heads up for the maximum amount of time the request could take. In this case it is no more than 3 days, so they are cutting it close with only 15 stock left. This program runs in sequence not in parallel so the purchased amount for the vodka reflects the previous iteration’s purchase. The screen will display “made customer angry and lost purchase” if a stock does go to zero before the request for restock is completed. To restock the amount that can be requested is the difference of the current stock at the time of the request from the maximum quantity. There is a possibility that you never have maximum stock after the start of the simulation. Also restocking occurs always at the beginning of the day or iteration.

For the store owner there are a lot of comments to be made. I think the first one is to be conservative about your stock but get to know which stock behaves differently. In my simulation we begin with bourbon having a small purchase rate but if there was a surge for bourbon, say for the Kentucky Derby, the simulation would constantly display that it was out of stock. Also, the time of day for restocking is very important. I think it is best to stock a storehouse before the day begins assuming you cannot have purchases 24hrs/day. If you can get ahead of the purchases in the “morning” then you could make a restocking request before the end of the day. Although it is not the case in my simulation because I chose to simulate bulk purchases instead of small batch purchases (since it’s a warehouse), the frequency of purchases per day can vary based on weekends or time of day. This is important to consider as you may want to have some sort of predictive just in time restocking to handle small fluxes in purchases. One of the most important pieces is to define a threshold that you can rely on that fits the min, mean, and maximum purchases to the number of days a restocking request can take. This could be hard, and I would suggest getting someone analytical to do this for you. This analysis can grow to be very complex if there are items that behave differently, like in my simulation of bourbon and vodka. So, an item that performs worse over time should receive less restocking and vice versa. To model this I would get some data that is consistent with the daily purchase history and try to fit the number of sales in a day to a normalize number so each n days you will make a request which will take m days. If at all possible, group the items that can come together in a timely fashion. These suggestions will keep the manager with a stock where in many cases. However, on days of overstock find an easy solution to either make extra sales at a discounted price or make your employees more appreciative of you. On days of understock I would find a way to market your under achieving items better.

****

**The warehouse simulation**

**Problem 3:**

I liked the 3D-Pursit problem because of how interesting it could be but it was very clear to me there was a major flaw after talking with you about the randomness of the coordinates. I went back and looked at the code and some of the original comments I made to myself about why I codded the simulation the way I did and I realized that early in development I made assumptions about what I could neglect in a simulation in order to simplify and reduce its complexity. It is obvious to me now, after your explanation, that the coordinates of the two craft cannot be random for each move.

When I think of this problem, I like to think neither pilot can escape so they must fight at parts of the pursuit. However, an interesting solution to the problem would be to create a threshold for an acceptable escape. Maybe the solution could have been at lightspeed so firing weapons would be counter intuitive unless you find a way to instantly fire. I think the pursuit problem will always exist and evolve as time goes on. There will always be form of it at least.

In general, I think my program could have improved if I had considered cases where the pursuit ended less randomly. Also having an escape sequence of events would have been more interesting and realistic. There also would have been the possibility of engine failure or if we considered how much fuel there was available. This could be equivalent or different from an operation time for the craft. I think it would have interesting also to add an arsenal inventory so when the chaser runs out of projectiles then they would have to stop pursuing. There are also several environmental problems such as meteors, radiation, gravity, etc. that could come into play. Also, the solution could have been a multi-pursuit solution where it was 1vs1, like in my program or MANYvs1, MANYvsMANY, or 1vsMANY. Also, there could have been more rules or logic related to distance and angle when pursuing so that distance could be used to intimidate or immediately end the simulation. For angle it could have been like diving directly towards the other craft and quickly decrease the distance. For this problem randomness could be used in many different ways and simulating the actual animation could be very interesting.