Genetic Algorithm Report

Background

A genetic algorithm is a type of neural network that is taken from the idea of Darwin’s principal of natural selection. When it is hard to decipher a clear solution, genetic algorithms are used to simulate the performance of complex systems. The idea of them is that with each iteration of optimization, the algorithm should increasingly develop a better solution, or fitter set of population members.

The key components of a genetic algorithm consist of a population class. This can either be initialised at random or be a select set that is being tested. A fitness calculation will need to be performed. Another component needed is a method in which we can determine the score of the population members that we have. This is known as the fitness calculation. The last three components that comprise of a genetic algorithm would be selection, crossover and mutation.

The purpose of this algorithm is to find the best solution to a problem through an evolutionary process. After we have initialised the population. We need to do the fitness calculation. This is how we find out the scores of the population members in preparation for our selection process. There are different types of selection processes for genetic algorithms such as linear rank selection, exponential rank selection or tournament selection. With the scores from the fitness calculation, you apply them to which ever selection method you have chosen. When the selection process is completed, two parents’ population members are assigned. Given that the probability of crossover is higher than the probability of a mutation occurring, we would most likely see crossover occurring. This is when a part of each parent is joined together to create a child member. If mutation occurs, this would be when we change a small part of the child’s gene. In the case of this experiment. It would be the changing of a single value.

After this is completed, we simply repeat the process of our selection process, crossover, and mutation until we create a new population.

**Explain how I built mine**

Using C# on virtual studios I started by using a list of lists to initialise the population size of 100 members. The second list was used to hold a set of 60 weights. So in total the population comprised of a hundred sets of 60 weights. The next step was to perform the fitness calculation. I did this by creating a method called “getScores” where I looped over all the members of the population and used the “GetResults” method to find the scores for every population member. I then stored the scores in a list called “scores.” After the fitness calculation was performed, I proceeded to use tournament selection as my selection process. How a tournament selection works is that a select number of population members are selected at random, in the instance of this experiment I used four. Then the population members with the highest two fitness scores become the parents. To do this, I created a method called “GetTournamentValues” in which the four population members are selected at random, stored in a list called “TournamentListValues.” I then calculate the scores of the four population members using the “GetResults” method and then store the scores in a list called “TournamentScores.” Using the “.Max()” function, I found out the top two highest scores of the four population members and stored each member in a separate lists called “TournamentValuesContainerA” and “TournamentValuesContainerB.”

Since the fittest population members become the parents, it increases the chances of creating better children when performing crossover and mutation. After I’ve stored the fittest values of the tournament, I clear rest of the values in “TournamentListValues” to reset the list. When this method was completed, I called it in the main method

When the Parents have been found, I made another method to perform crossover on the two parents. I started by setting the crossover rate to a random double between 0 and 1. I then made an if statement saying that if the crossover rate is less than 0.5, store the first 30 values in the first parent list and the last 30 values in the second parent list in a List called “Child.” This means that a single child has been created through crossover. After that, I added the value in my child list to another list called “Children.”

Underneath this if statement I made an else statement incase the crossover rate was higher than 0.5. If this happens, then I added the highest scoring parent to the child list then added it to the children list. I then return the children list. In the main, I created a list called “mainChildren” and stored the values in the children’s list there. This was to stop passing the values by reference issue I was having.

Next, I made a method to perform mutation. In this method, I looped through all the weights in the child list. Inside this for loop, I set the mutation rate variable to a random double between 0 and 1. Next, I created an if statement saying that if the mutation rate is less than 0.05, pick a random double and replace that weight’s value with the new random value. After that, I increment a variable called “Count” by one. This variable holds the number of iterations that the program has done when making a child. I then called mutation method in the main method.

Experimental Design

**Comparing two different mutation rates and the effect they might have on my results:**

For this experiment, I am going to be using a low mutation rate of 0.05 for the first run and a high mutation rate of 0.95 for the second set. For the population that I run with a low mutation rate, I predict that over time, when plotted on a graph, the average population score will improve at a faster rate. On the other hand, I predict that if the mutation rate is high, the average population score will increase at a slower rate.

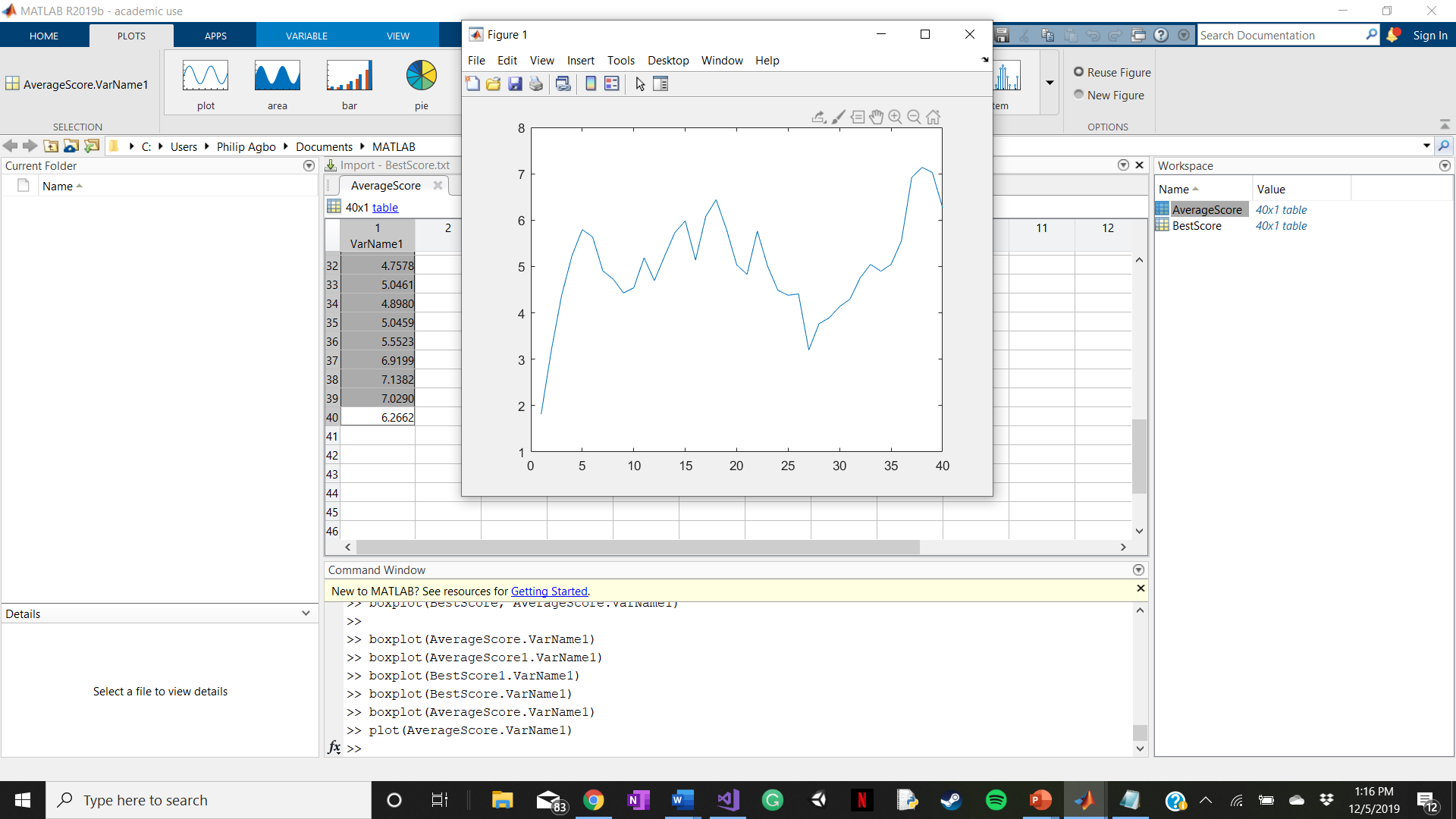
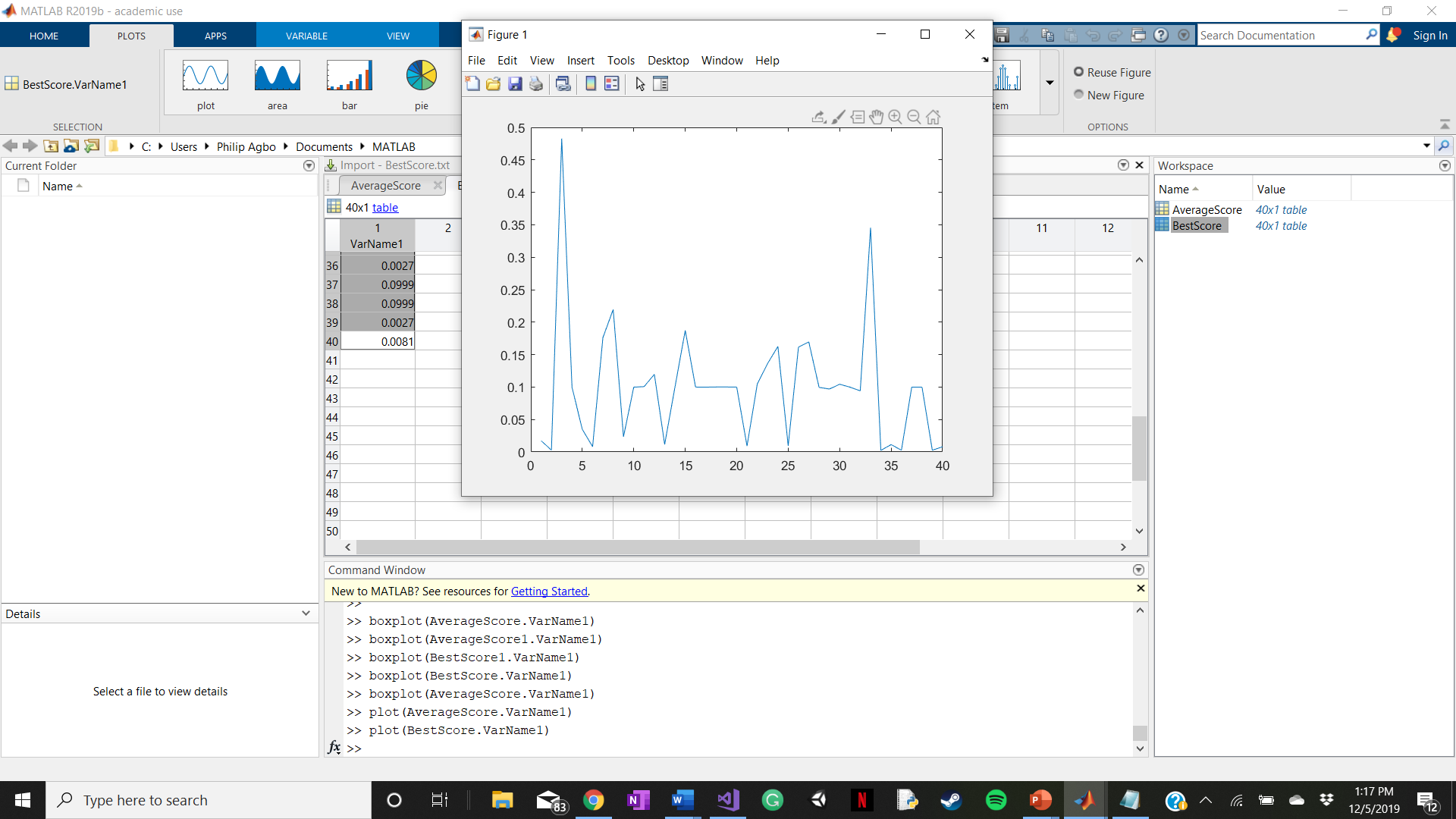
I chose 0.05 and 0.95 as my parameters because there are opposite ends of the spectrum for a mutation rate. I also feel like this will also give me two very distinctly different sets of results. However, on the other hand I feel like using a high mutation rate such as 0.95, It might significantly increase the run time of the genetic algorithm as it has more lines to execute before creating a child.

Experimental Results

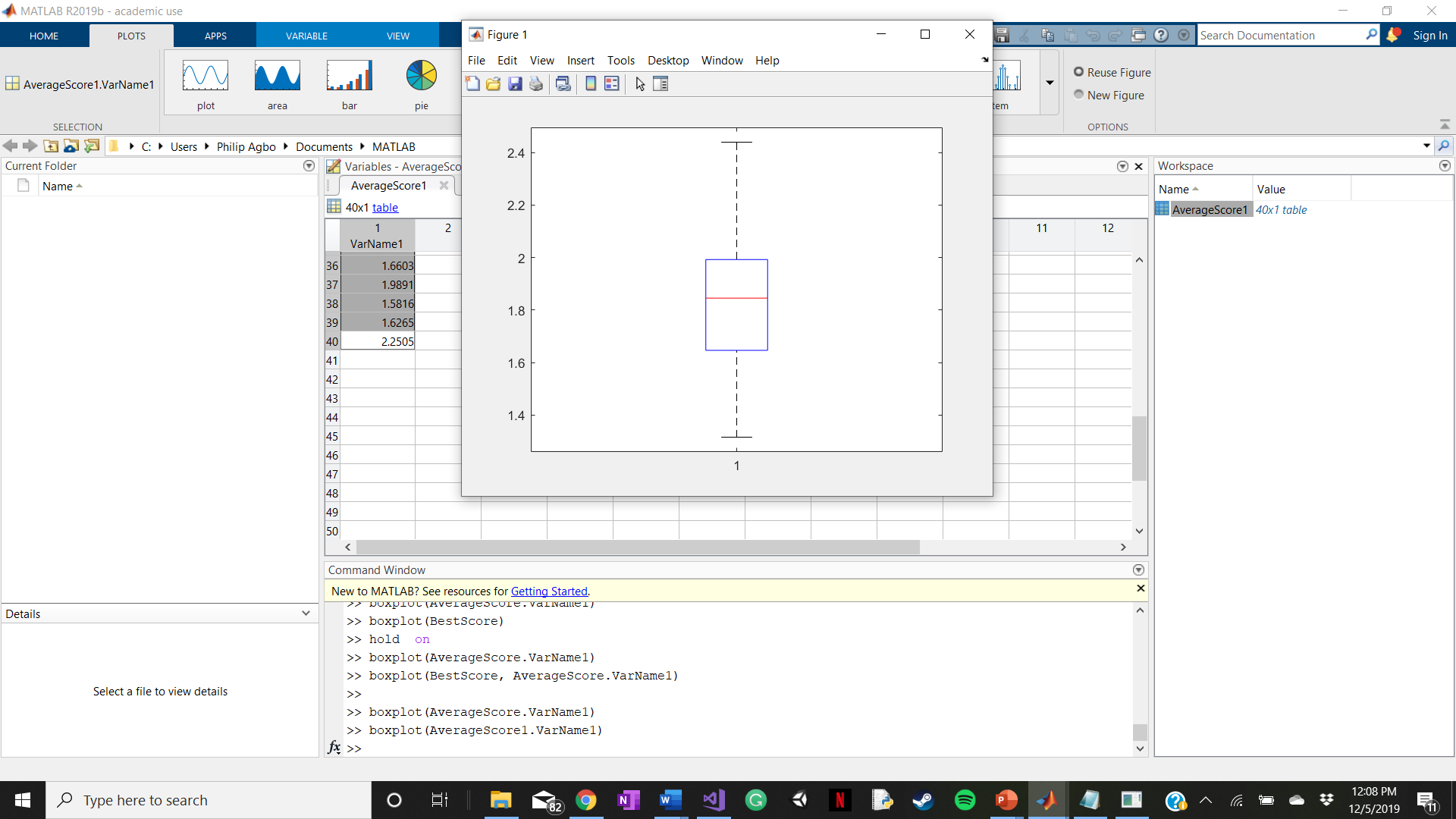
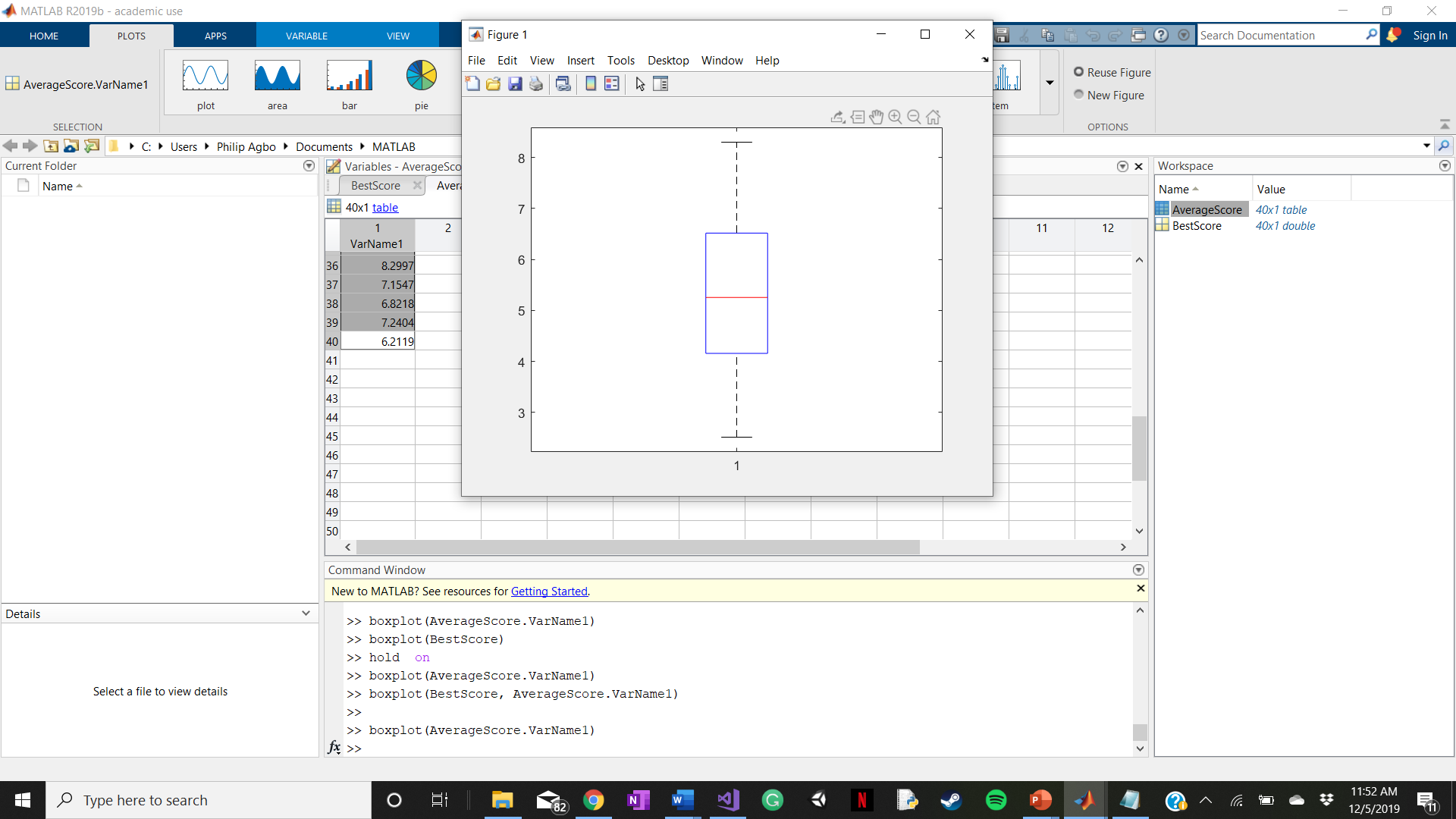
I was able to execute a large majority of the functionalities needed for my algorithm to work. I was able to successfully create a population of a hundred values each with 60 random weights in them. I was able to perform the fitness calculation on all the population members and display them. My algorithm was also successful in performing crossover and mutation on the population members.

Best Score

Average Score



These two graphs plot the best score and average score of 40 generations. From the Best score graph, we can see that there is a random increase and degrees in the best scores over the length of generations. The highest best score reached 0.48. From the average score graph we can see a gradual increase of the average score over time. This tells us that from the start of the first generation to the last, the population is getting fitter overtime.



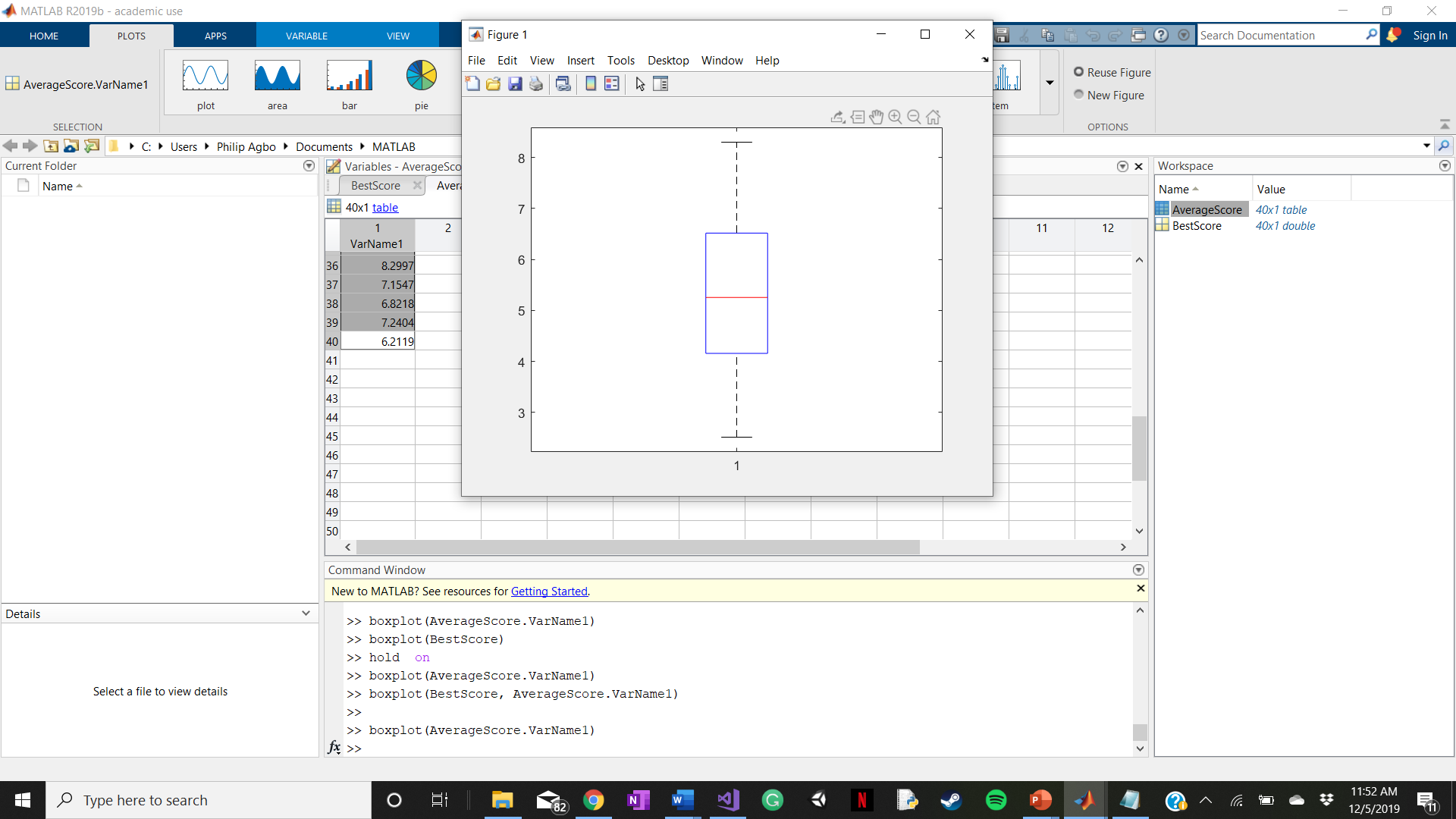
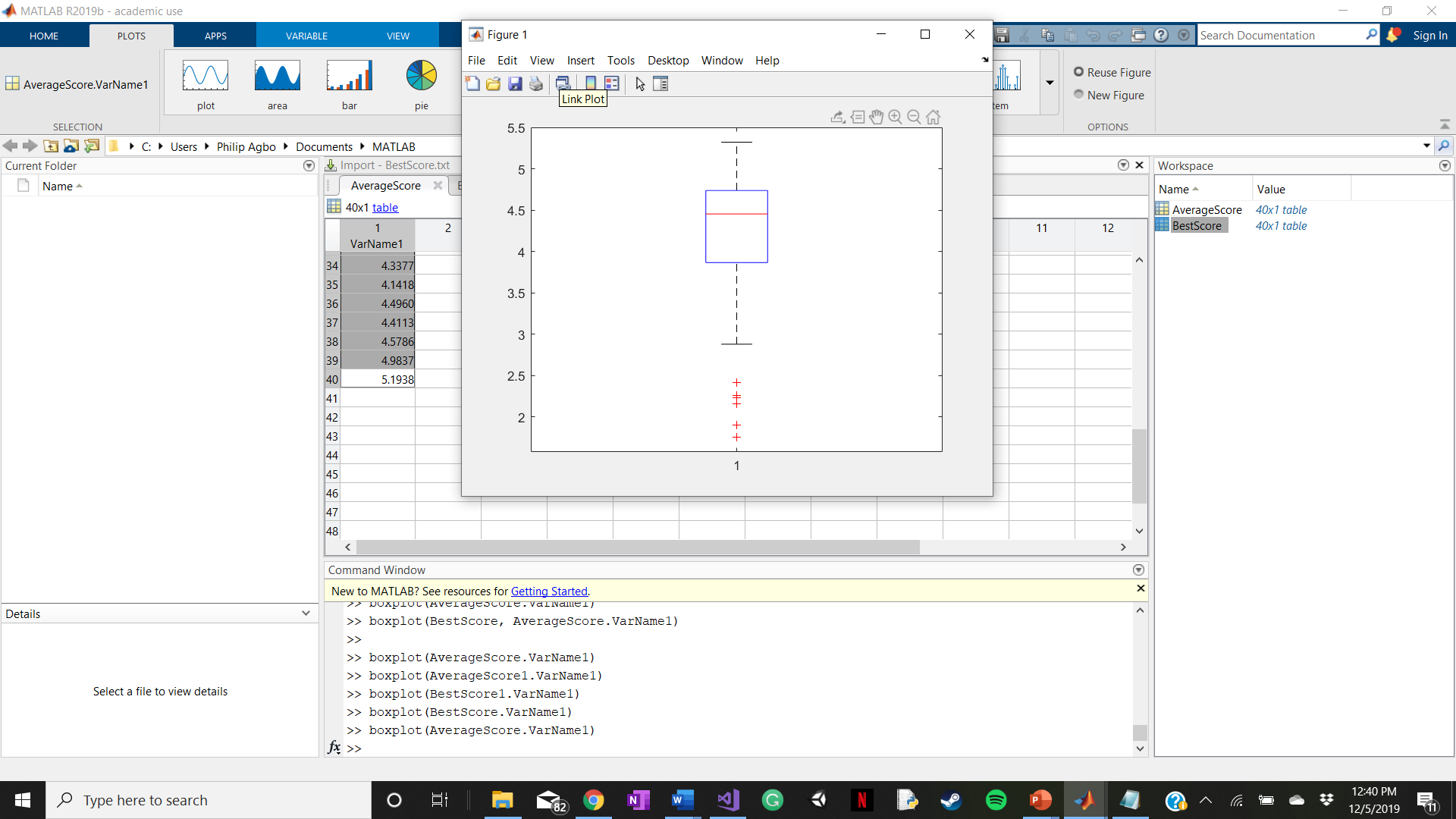
Mutation rate = 0.05

Mutation rate = 0.95

These two box plot graphs show us the results of the genetic algorithm when two different mutation rates are used. From the graph on the left, we can see that when the genetic algorithm is set to a lower value of 0.05, the average scores are higher than that of one with a high mutation rate of 0.95. We can also see that when the mutation rate is set to 0.95, the scores are between 2.4 and 1.2 which is a much closer range than the graph with a lower mutation rate of 0.05.

Crossover Rate = 0.5

Crossover Rate = 0.95



These two graphs show us what happens to the genetic algorithm when there is a change in the crossover rate. From the graph on the left with a lower mutation rate, we can see that the average scores are higher than the algorithm with a higher crossover rate. On the other hand, the algorithm with the lower crossover rate have more spread out values in comparison to that of the higher crossover rate which has values closely packed together.

Conclusion

**What my results show**

From my results, you can see that over 40 generations, the genetic algorithm improves its fitness score. We can also see that fitter scores are achieved when lower mutation and crossover rates are used. This surprised me because I thought that increasing the crossover rate would significantly improve the fitness of the population.

**What I would have done differently**

In conducting this experiment, something that I would have done differently would be to try and use another selection method like linear rank selection or exponential rank selection just to see if that would have had an effect on the fitness scores or the rate in which the fitness scores increase in comparison to tournament selection.

**What I need to get the best results**

To get the best results, I would have to increase the number of generations that I ran. This would mean that the population has more chances to evolve into a population with fitter scores. A second way to get the best results would be to increase the number of values that are initially used in the tournament selection. So instead of choosing selecting four population members at random to choose the best weights, I could choose to select ten population members at random. Lastly, to get better and more accurate results, I would need to run the program multiple times to ensure that I have the most accurate results.

**How can the algorithm be used in a different context?**

Genetic algorithms have numerous implementations in the real world. A very well-known use for them would be in gaming. When a player is playing against the “AI” they usually have some sort of a genetic algorithm built into them that allows the to interact with the player depending on their actions. These AI will can also learn. In some fighting games, when the player uses a specific move or strategy against the AI, it will learn from that, and counter that strategy, meaning that the player not use the same strategy or move to try and beat the player.

References:  
What is a genetic algorithm: blog.floydhub.com/introduction-to-genetic-algorithms/

researchgate.net/publication/259461147\_Selection\_Methods\_for\_Genetic\_Algorithms

<https://www.brainz.org/15-real-world-applications-genetic-algorithms/>