Find Stable Solutions to the Richardson Equation

CSE 589: Bioinspired Computing

Bryce Pedroza†  
 CIDSE  
Arizona State University  
 Tempe, Arizona USA  
bapedroz@asu.edu

Stephanie Forrest  
Biodesign Center for Biocomputation, Security and Society  
 Arizona State University  
 Tempe, Arizona USA  
steph@asu.edu

ABSTRACT

The purpose of this assignment is to find stable arms control scenarios using the Richardson Model. To this end, a genetic algorithm was written in Python 3 from scratch to find parameters that would lead to arms control among the three countries in the differential equations in the Richardson Model. This GA is represented using an object-oriented programming paradigm to simplify its design for end users. To determine its effectiveness, a combination of different metrics is observed. A novel normalization factor is proposed for to the fitness function in an attempt to promote more interesting individuals in the GA. This normalization feature is toggleable and its impact on results was also recorded. By including the normalization to the fitness function, more consistent results were reached that could produce the ideal arms control systems between the three countries. The result is an effective GA that can produce quality parameters that lead to fast arms control scenarios.

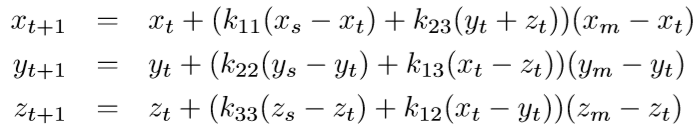
KEYWORDS

Genetic Algorithms, Richardson Equation, normalization, mutation, crossover, selection, Root-mean-square deviation, arms control

1. Introduction

Lewis Fry Richardson expressed an idea that the push and pull of arms spending was indicative of the outbreak of war. Therefore, he developed a system of differential equations that could estimate the probability of two countries engaging in war according to their arms expenditures and other fiscal constraints, thus known as the Richardson Arms Race Model. This model has been widely adopted to two country systems, but is there a reason it could not be adjusted to multiple countries? For the purpose of this assignment, the Richardson Equation was updated to reflect the introduction of a third country that can contribute to the outbreak of war. This iteration more accurately reflects the advanced international economy of the world today (USA, Russia, and China, for example). In this version, the two weaker countries become allies, and pool their resources against the dominant country.

This model can exhibit some strange behaviors. First, there is the possibility of a shift in alliance. This would be the result of one of the weaker countries overtaking the dominant country, thus causing a new alliance between the new set of weaker countries. Secondly, instability within the nations can present itself in the form of a runaway arms race in which the countries continue to increase their spending on arms over each time step. Ideally, the countries would find a scenario in which the spending remains constant and shifts in alliance no longer occur. This would be considered as arms control and is the steady state proposed in the Richardson Equation.

****

**Eq 1.** Richardson model expressed in three differential equations

However, there are 12 parameters that can be adjusted in this equation, so it can be difficult to manually determine how these values can be set such that a state of arms control between the three countries can be achieved. To do this, a genetic algorithm can be created to produce the possible parameterizations.

2. Methodology

The genetic algorithm was implemented Python 3 without the use of genetic algorithm libraries. The parameters for this GA are all real numbers between 0 and 1. The GA is designed to create a variable size population with random initial parameters, perform crossover between individuals, mutate individuals, and perform a selection tournament to pick the fittest individuals. By performing crossover and mutations during each iteration of the genetic algorithm, the search space is continuously explored, and genetic diversity is increased, thus leading to more fit generations over time.  These actions are performed over multiple generations with the goal of increasing the fitness of the overall population over time.

To perform mutation, one value for the individual is randomly chosen. This value is then increased or decreased by 5% of its current value. Now, these are real values and must be in between 0 and 1, so if the mutation was to increase or decrease the value above or below these thresholds, mutation would be limited, and the value would be kept within this predefined range. There is a variable mutation rate that can limit the number of mutations in a generation.

Crossover switches one value from two randomly selected individuals in the population. There is a variable crossover rate to limit the number of crossovers performed in a generation. The crossover is done with replacement. Thus, an individual can be crossed over more than once.

The selection tournament is a one versus one selection tournament with replacement, thus allowing for an individual to be added to the next population more than once. The fitter of the two individuals is selected. The fitness of these individuals is calculated using a fitness function. An individual is more fit if the current arms spending between the dominant country and the two allied countries is close to 0.

**Eq 2.** Standard fitness function for Richardson Model

3. Representation

The values for the GA are being represented using Python objects. The lowest level object is a Country(). Each Country holds each of their four unique parameters that are randomly assigned, mutated, and crossed over. Along with this, each country holds its current arms spending and whether or not it is the current dominant country. A country has the ability to take in all of the above parameters for the purpose of debugging possible parameterization.

The Richardson() object simulates a single individual within the GA population. Each Richardson contains three countries, x, y, and z. Along with these three countries, it contains its fitness and mutation rate. The Richardson class can take all of these as parameters, again for debugging purpose, but by default it uses a static method to randomly create the three countries and assigns its initial fitness to -1. The mutation rate must be passed into each Richardson object. Its publicly available functions are to mutate and to perform fitness calculations. Mutation is performed as described in the previous section. To perform fitness calculations, Eq. 1 is performed dynamically depending on which country is currently dominant. After the spending is calculated, the countries are reassessed, and a new dominant country is assigned if one of the allied countries has started to outspend the dominant country. Once any possible reassignments are complete, Eq. 2 is performed and assigns the fitness for the individual.

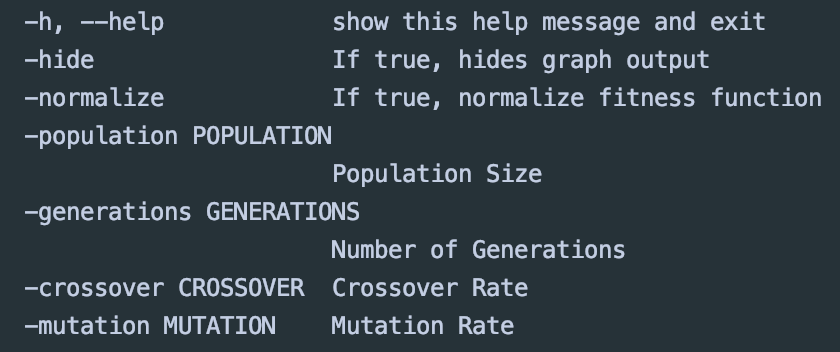
The GeneticAlgorithm() object contains all the information needed to perform the entire GA as the name suggests. There are four required parameters for the GA: population size, crossover rate, mutation rate, and a normalization flag (this flag will be addressed later). Upon initialization, the GA creates a population of Richardson objects according to its parameterization. It has one publicly available function, which is to iterate the population. This function performs crossover, mutation, selection tournament, and historical tracking, in this order. Mutation and crossover have been described in detail in previous sections.

However, there is a small caveat when performing the selection tournament. When adding the fitter individual to the next generation, a deep copy for that individual is sent and not the individual itself. A deep copy will create a new object with the same attributes as the original. If a deep copy is not used, then the new generation will have multiple references to the same individual in the population. This means, without the deep copy, if one individual is crossed over then that change is reflected everywhere that object is referenced in the population. This could cause individuals to be mutated or crossed over significantly more than intended.

Once the selection tournament is performed, the GA object performs some additional calculations to track the best global solutions and the average fitness of the population for each generation. With these objects in place, the GA can be run quite easily for a variable number of generations to observe how performance changes.

4. Parameterization

Because the final GeneticAlgorithm() object takes in 4 parameters and there is one additional parameter for the number of generations that will be performed, there are the 5 values that can be adjusted from run to run. These values can be adjusted with command line arguments to allow for quick parameter adjustments without having to touch the source code. Below are the parameters that can be adjusted at the command line. A full detailed list of ways to run the code can be found in the README.md

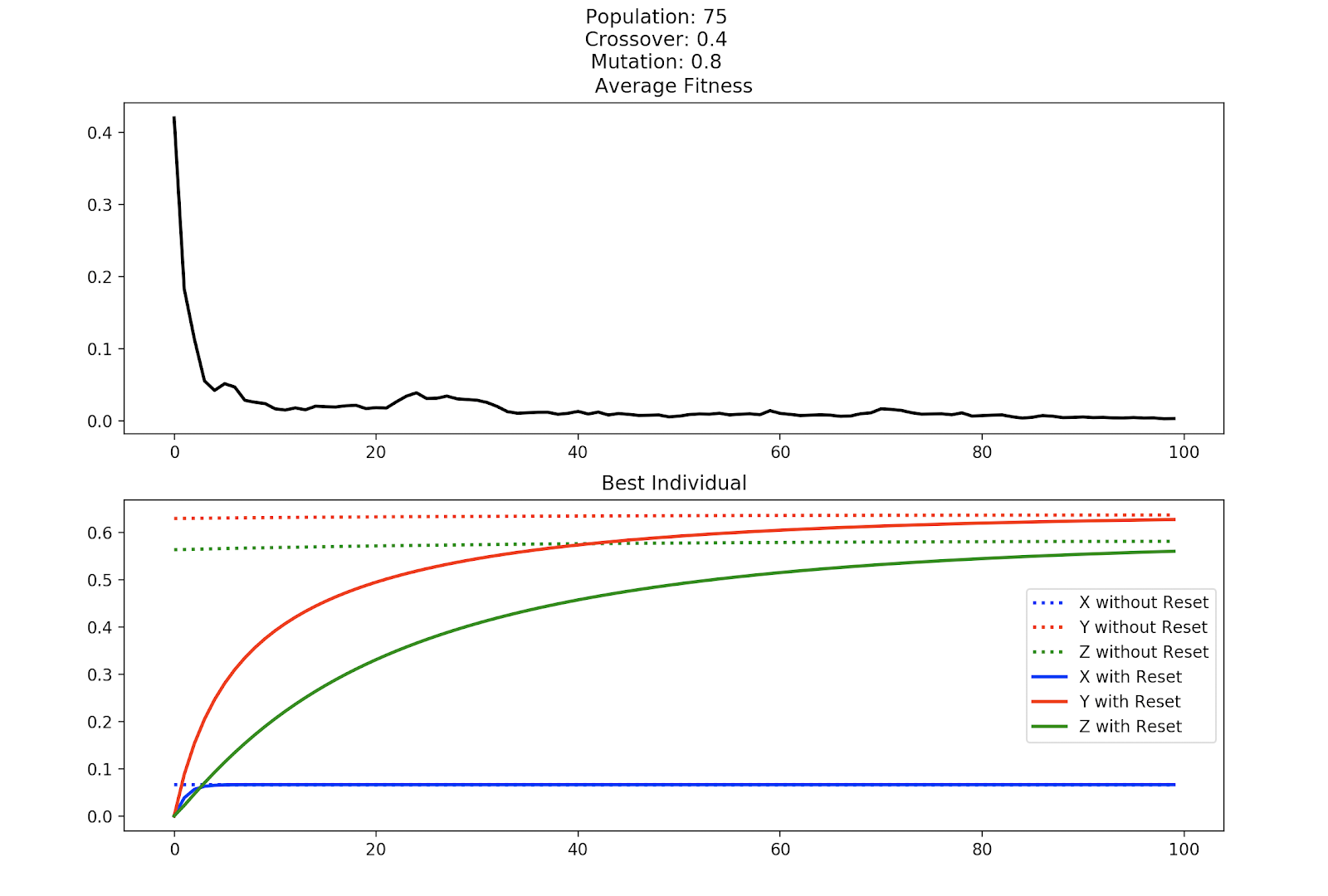


**Fig 1.** Here are the possible command line arguments that can be used when running the GA. These params will adjust the main parameters for the GA. The appendix will include additional details on how to run the code.

5. Experimental Design

Once a GA runs for the specified amount of generations, a way to assess the results had to be produced. A visualization was created to observe any patterns that might have manifested in the final results. The first was a fitness curve to show average fitness across each generation in the GA.

The next visualization was built to study how the best individual’s parameters performed over time.  The individual that produced the highest fitness was taken from the population and simulated again with its countries’ parameters and where *t* is the number of generations simulated. These same parameters were used again but initial spending was set to 0, . This was done to compare how the parameters would perform according to the initial state assumed in the assignment. These calculations were recorded for the number of generations ran by the GA, giving the Richardson Equation enough time to accurately observe its expected results vs its actual results.



**Fig 2.** Here is a standard output from the GA. The first visualization shows how the average fitness for the GA converges towards 0, which is ideal. There are some perturbations in the average fitness that can be attributed to the GA mutating and performing crossover to promote genetic diversity and to increase the search space. The second visualization shows the best individual and its counties’ spending over time. The dotted lines assume initial conditions . The solid lines assume initial conditions . The faster the actual results can simulate arms control, the better the final solution is.

Root-mean-square deviation (RMSD or RMSE) is used to measure how the expected model behaves () vs. how the actual conditions behave (). RMSD can thus be used to determine how the fitness and country arms spending of the actual function compare to that of the expected function. A low RMSD indicates the differences between the actual and expected values are small. RMSD can be calculated for fitness as well as for the countries’ average spending. RMSD for fitness is a measure of how consistent the fitness is over time. RMDS for the countries is a measure of how fast the countries converge to their expected arms control point.

However, when running ad hoc experiments to observe how parameters could impact performance, the best individuals were typically spending an extremely limited amount on arms. This would result in them having incredibly low fitness, but their RMSD measures would be disproportionately higher, meaning their parameters did not actually perform well. These individuals rise to the top because of how the fitness function promotes extremely low spending on arms. Consider . Using Eq. 2, this fitness, F, is 0.01. Now consider . This tuple would produce , a fitness 5 times worse than the previous. However, if the two fitness were normalized, the latter is better than former by nearly a factor of two. So, while the second could have relatively a more stable solution, the current fitness function does not promote that behavior.

To address this problem faced in the fitness function, the ability to normalize the fitness function was proposed. (Adjusting the fitness function might have been out of scope for this project but normalizing the fitness function did provide much better results) By standardizing the fitness function, low scale results are evaluated fairly against the larger spending results. When enabled with the normalization flag mentioned in section 3, the fitness function is updated to Eq. 3, eliminating the problem addressed in the paragraph above. With Eq. 3, would produce and would produce , accurately reflecting that the latter is closer to arms control. To further support normalization of the results, the normalized root-mean-square deviation (NRMSD) was utilized to evaluate the results.

**Eq 3.** Normalized fitness function

**Eq 4.** NRMSD, where y is either fitness or country spending

Experiments were designed to study how population size, number of generations, mutation rate, crossover rate, would impact the average fitness of the GA, the best individual fitness produced by the GA, the country NRMSD, and the fitness NRMSD. Additionally, with the introduction of the normalization technique, running trials with it enabled and disabled should help in understanding how it impacts the overall results from the program. To assess all of these different possible combinations, a standard baseline of values was set. The values in the baseline appeared to give decent results before beginning official experiments. Thus, adjusting from these baseline values should yield results that are acceptable but still show how parameters can affect different parts of the output. Each set of parameters is run 10 times, and the average across all the runs is recorded. These 10 runs are classified as a single experiment.

To evaluate how consistent the results are in a single experiment, the standard deviation was also recorded for the country NRMSD and fitness NRMSD across all runs. This should assist in understanding how the results vary between each run in the experiment.

6. Experimental Results

The results can be found as figure 1 in the appendix, and the entire recorded results for all the runs can be found in the excel sheet in the submitted zip file. Here is what was observed. The parameters that produced the results will be displayed in order of [**Population, Crossover, Mutation, Generations, Normalization]**. The baseline was determined to be ***[100, 0.8., 0.4, 100, True]***

For average fitness, the best results were seen with *[75 0.4 0.8 100 False]*. However, it should be noted that with normalization set to false, Eq. 2 was utilized for the fitness function. Thus, individuals with lower spending would be classified as fitter than those spending more even though the lower spending results might not lead to arms control. The worst fitness was with *[50 0.4 0.8 100 True].* This makes sense because smaller population can lead to less coverage in the search space. This can mean it takes longer to find the best fit. This same run, *[50 0.4 0.8 100 True]*, also had the worst best fitness overall, which again makes sense due to the small population size. *[100 0.3 0.8 100 False]* was found to have the best overall fitness, which again makes sense as again normalization was disabled.

*[75 0.4 0.8 100 True]* found to have the best country NRMSD and fitness NRMSD across all different experiments. In fact, all experiments in which normalization was enabled had a lower country NRMSD and fitness NRMSD than all runs where normalization was disabled. Therefore, the best individuals for the normalized runs consistently converged to the expected values while the non-normalized experiments were significantly higher, thus suggesting a low correlation between the expected arms control and the actual results with non-normalized results. Figure 2 in the appendix will show what a run with small arms spending actually performs against its expected.

But how much did these runs vary? In terms of the most consistent result for country NRMSD, *[100 0.4 0.8 100 True]* produced a standard deviation of only 0.1093963639.  *[100 0.3 0.8 100 True]* gave the best results for fitness NRMSD, with a standard deviation of 0.106332323. This appears to suggest that mutation rate is more important than crossover rate in impacting how the results from each experiment change because even though the crossover rate was changed by a factor of 0.1, the best standard deviations did not vary too much. This is further supported by the runs that had the worst standard deviations. Country NRMSD *[100 0.45 0.75 100 True]* had a standard deviation of 0.3439660882 across its experiments, nearly three times worse. Fitness NRMSD had a similar result of 0.307012628 for *[100 0.4 0.7 100 False]*. These can show that decreasing the mutation rate will cause a decrease in the search space, causing runs to converge to different local optimums.

The results do not indicate that there is an end all set of parameters that perform better than all other options, but the results do indicate that parameters can be tuned to find the results that fit different use cases. For overall fitness, removing normalization can prove effective. Including normalization will increase the relationship between the expected results of the parameters and the actual results. Keeping the mutation rate high will ensure a higher search space, causing a decrease in the variation of results from run to run. If speed is being considered, decreasing the number of individuals in the population could prove to be successful while not impacting results significantly. However, it would be best to keep the population above 50 individuals to avoid poor results.

7. Conclusion

The GA implemented was able to produce consistently fitter individuals over time. The quality of the individual did vary depending on whether normalization was considered in the fitness function. Results that did have normalization more consistently converged to the arms control observed in the expected results. Experiments without normalization often had the desired low fitness score but their actual results were not consistent with their expected results. Adjusting mutation rate and crossover rate impacted the variation of results for each experiment, and as population sizes decreased, the average fitness across all individuals should be expected to fall. The difference in results by varying the number of generations in an acceptable range (50-100 generations) did not have a large impact on any of the results, but lower number of generations will not give the GA enough time to produce meaningful results.

Overall, the goals of the assignment were met, and a genetic algorithm was produced that can find parameters that will lead to arms control in the three-country version of the Richardson model. This GA is robust and can quickly take different command line arguments such that it can produce results that adhere to the user’s needs.

REFERENCES

[1] Patricia S. Abril and Robert Plant, 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan, 2007), 36-44. DOI: <https://doi.org/>10.1145/1188913.1188915.

[2] Sten Andler. 1979. Predicate path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '79)*. ACM Press, New York, NY, 226-236. DOI:https://doi.org/10.1145/567752.567774

[3] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. DOI:https://doi.org/10.1007/3-540-09237-4.

[4] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY..

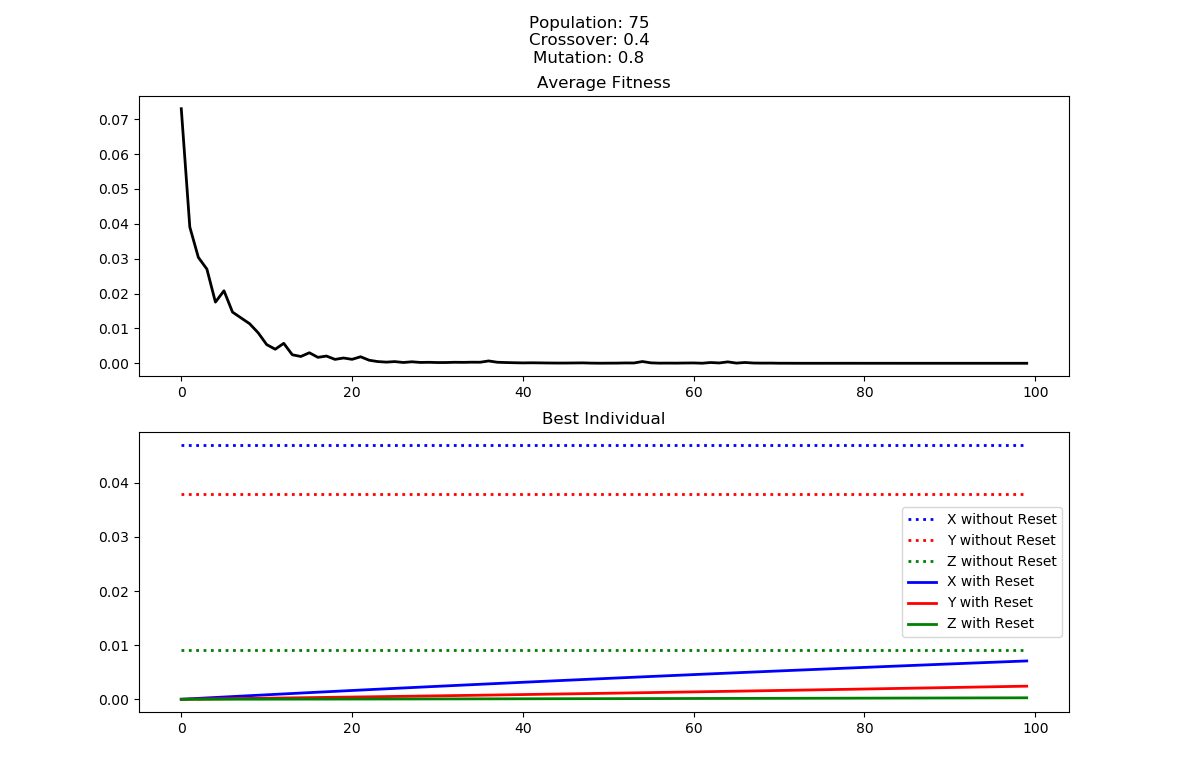
APPENDIX

Figure 1. Results from all experiments

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Population** | **Crossover** | **Mutation** | **Generations** | **Avgerage Fitness** | **Best Fitness** | **Country nrmsd** | **Fitness nrmsd** | **Normalized** | **Country Std** | **Fitness Std** |
| 100 | 0.4 | 0.7 | 100 | 0.01631626207 | 0.000000630084281 | 0.3346878173 | 0.3437710291 | TRUE | 0.2511887288 | 0.3086389107 |
| 100 | 0.4 | 0.8 | 100 | 0.01685531494 | 0.000002272712464 | 0.1945235586 | 0.3432609852 | TRUE | **0.1093963639** | 0.3298928015 |
| 100 | 0.4 | 0.9 | 100 | 0.01847826125 | 0.000001573612222 | 0.2634253185 | 0.3285033556 | TRUE | 0.2743957689 | 0.3360761761 |
| 100 | 0.3 | 0.8 | 100 | 0.01341115812 | 0.000000348833956 | 0.2304426464 | 0.2376130068 | TRUE | 0.1796079115 | **0.106332323** |
| 100 | 0.5 | 0.8 | 100 | 0.01995389054 | 0.000001058808867 | 0.2598852606 | 0.350828571 | TRUE | 0.1358812332 | 0.1925513282 |
| 100 | 0.4 | 0.8 | 50 | 0.01633072643 | 0.0000008716843948 | 0.2414153473 | 0.2089512361 | TRUE | 0.1455322115 | 0.1944027671 |
| 100 | 0.4 | 0.8 | 75 | 0.01654754323 | 0.0000006328915733 | 0.2691529668 | 0.2437952015 | TRUE | 0.2550234376 | 0.2400725665 |
| 50 | 0.4 | 0.8 | 100 | **0.02046016512** | **0.00000835921213** | 0.2094759122 | 0.3774923729 | TRUE | 0.2036091128 | 0.2862198777 |
| 75 | 0.4 | 0.8 | 100 | 0.0169443973 | 0.0000026972738 | **0.1665288972** | **0.1464077718** | TRUE | 0.1457311863 | 0.1441694512 |
| 100 | 0.45 | 0.75 | 100 | 0.01744664092 | 0.0000006138544667 | 0.4695331389 | 0.4996227007 | TRUE | 0.2911755065 | **0.3439660882** |
| 100 | 0.4 | 0.7 | 100 | 0.00342042553 | 0.0000002727898985 | 0.701007073 | 0.4964170771 | FALSE | **0.307012628** | 0.2133165304 |
| 100 | 0.4 | 0.8 | 100 | 0.003328745817 | 0.0000001152894789 | 0.6493698878 | 0.5698808348 | FALSE | 0.2244728468 | 0.2382174576 |
| 100 | 0.4 | 0.9 | 100 | 0.003958966301 | 0.0000001242246193 | **0.7142177412** | 0.5156610268 | FALSE | 0.3042933296 | 0.2354372441 |
| 100 | 0.3 | 0.8 | 100 | 0.00329564188 | **0.00000006482445911** | 0.4956860304 | 0.5977189744 | FALSE | 0.2882069606 | 0.2641118459 |
| 100 | 0.5 | 0.8 | 100 | 0.004014235746 | 0.0000002044208845 | 0.5527731655 | 0.4624114124 | FALSE | 0.2783065647 | 0.2167981229 |
| 100 | 0.4 | 0.8 | 50 | 0.003695560598 | 0.0000001981718058 | 0.4718826639 | 0.5215500832 | FALSE | 0.2736253718 | 0.2452940505 |
| 100 | 0.4 | 0.8 | 75 | 0.003645172888 | 0.0000001927136484 | 0.5773292457 | 0.6003885167 | FALSE | 0.2126699178 | 0.2690960677 |
| 50 | 0.4 | 0.8 | 100 | 0.004307531693 | 0.0000007945771678 | 0.4343247617 | 0.5056075878 | FALSE | 0.3015196459 | 0.297391287 |
| 75 | 0.4 | 0.8 | 100 | **0.003259582753** | 0.0000001680992508 | 0.5996055907 | **0.6117177023** | FALSE | 0.1966789072 | 0.2123096287 |
| 100 | 0.45 | 0.75 | 100 | 0.00375658629 | 0.0000004436559557 | 0.6459887814 | 0.4351678966 | FALSE | 0.3026548114 | 0.276065642 |

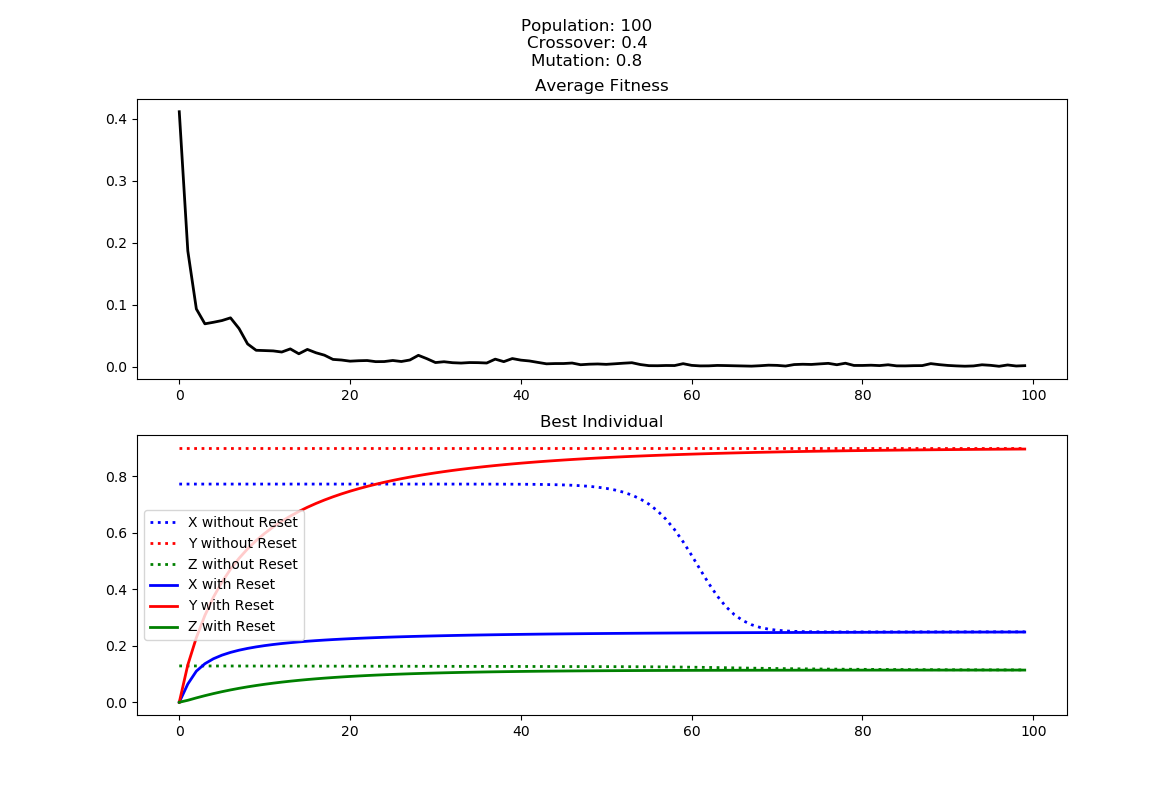
**Figure 2.**

This run shows the non-normalization problem in which the best individual has extremely low arms spending, and actual results do not converge



**Figure 3.**

An interesting output. Screenshots of outputs can be found in the interesting\_outputs directory.



**Figure 4. Steps to run code locally**

*Requirement: Python3 is installed*

Install pipenv. Pipenv is a package manager for python, which allows you to create and share virtual environments easily

pip3 install pipenv

Install the packages from the Pipfile

pipenv install

Try to run the code with basic parameters

pipenv run python assignment1.py

Add the -h flag to see all possible options for more help

pipenv run python assignment1.py -h

Here is an example run with full command line goodness

pipenv run python assignment1.py -normalize -population 75 -generations 100 -crossover 0.4 -mutation 0.8

**Figure 5. Code to run experiments**

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -normalize -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.7 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.4 -mutation 0.9 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.3 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.5 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 50 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 75 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 50 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

pipenv run python assignment1.py -hide -population 75 -generations 100 -crossover 0.4 -mutation 0.8 >> output.txt

echo Next Test >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

pipenv run python assignment1.py -hide -population 100 -generations 100 -crossover 0.45 -mutation 0.75 >> output.txt

**Figure 6. Richardson.py**

import random

import logging

logger = logging.getLogger()

class Country:

def \_\_init\_\_(self, dominant=False, \*\*kwargs):

self.dominant = dominant

self.curr = 0

self.\_\_dict\_\_.update(kwargs)

class Richardson:

def \_\_init\_\_(self, new=True, x=None, y=None, z=None, mutation\_rate=0.9, normalize=False):

if new:

self.x = Richardson.new\_country(dominant=True)

self.y = Richardson.new\_country()

self.z = Richardson.new\_country()

else:

self.x = x

self.y = y

self.z = z

self.fitness = -1

self.mutation\_rate = mutation\_rate

self.normalize = normalize

def mutate(self, individual=-1):

"""

We mutate one of the fields if a random number from 0-1 that we pick

is less than the mutation rate. ie. if our mutation rate is .9,

and the random is less or equal then we mutate, thus simulating a

90% mutation rate.

:return:

"""

if random.uniform(0, 1) <= self.mutation\_rate:

# Then we mutate one of the fields!

temp\_name = random.choice(['x', 'y', 'z'])

random\_country = self.\_\_getattribute\_\_(temp\_name)

random\_value = random.choice(

['expend', 'econ\_rest', 'k\_self', 'k\_others'])

temp\_value1 = random\_country.\_\_getattribute\_\_(random\_value)

# Mutate value by 5% Either add or subtract.

if random.randint(0, 1) == 1:

temp\_value2 = temp\_value1 + temp\_value1 \* .05

else:

temp\_value2 = temp\_value1 - temp\_value1 \* .05

# Make sure the value is in between 0 and 1

if temp\_value2 > 1:

temp\_value2 = 1

elif temp\_value2 < 0:

temp\_value2 = 0

random\_country.\_\_setattr\_\_(random\_value, temp\_value2)

logger.debug(f"Mutating individual {individual}'s {random\_value} "

f"from country {temp\_name} to {temp\_value2}, from {temp\_value1}")

def perform\_calculations(self):

if self.x.dominant:

Richardson.calculate\_spending(self.x, self.y, self.z)

elif self.y.dominant:

Richardson.calculate\_spending(self.y, self.x, self.z)

elif self.z.dominant:

Richardson.calculate\_spending(self.z, self.y, self.x)

# Is x the biggest?

if self.x.curr > self.y.curr and self.x.curr > self.z.curr:

self.x.dominant = True

self.y.dominant = False

self.z.dominant = False

# Is y the biggest?

if self.y.curr > self.x.curr and self.y.curr > self.z.curr:

self.y.dominant = True

self.x.dominant = False

self.z.dominant = False

# Is z the biggest?

if self.z.curr > self.y.curr and self.z.curr > self.x.curr:

self.z.dominant = True

self.y.dominant = False

self.x.dominant = False

if self.x.dominant:

self.fitness = Richardson.calculate\_fitness(self.x, self.y, self.z, self.normalize)

elif self.y.dominant:

self.fitness = Richardson.calculate\_fitness(self.y, self.x, self.z, self.normalize)

elif self.z.dominant:

self.fitness = Richardson.calculate\_fitness(self.z, self.y, self.x, self.normalize)

def reset\_current\_spending(self):

self.x.curr = 0

self.y.curr = 0

self.z.curr = 0

# reset who is dominating too, x is default

self.x.dominant = True

self.y.dominant = False

self.z.dominant = False

@staticmethod

def calculate\_fitness(dominant, a, b, normalize):

fitness = abs(dominant.curr - (a.curr + b.curr))

if normalize:

fitness = fitness/abs(dominant.curr-min(a.curr, b.curr))

return fitness

@staticmethod

def calculate\_spending(dominant: Country, a: Country, b: Country):

dominant.curr = dominant.curr + (

dominant.k\_self \* (dominant.expend-dominant.curr) +

dominant.k\_others \* (a.curr+b.curr)) \* \

(dominant.econ\_rest-dominant.curr)

a.curr = a.curr + (

a.k\_self \* (a.expend-a.curr) +

a.k\_others \* (dominant.curr-b.curr)) \* \

(a.econ\_rest-a.curr)

b.curr = b.curr + (

b.k\_self \* (b.expend-b.curr) +

b.k\_others \* (dominant.curr-a.curr)) \* \

(b.econ\_rest-b.curr)

@staticmethod

def new\_country(dominant=False):

return Country(

dominant=dominant,

expend=random.uniform(0, 1),

econ\_rest=random.uniform(0, 1),

k\_self=random.uniform(0, 1),

k\_others=random.uniform(0, 1)

)

def get\_county\_props(self):

return {

'x': self.x.\_\_dict\_\_,

'y': self.y.\_\_dict\_\_,

'z': self.z.\_\_dict\_\_

}

def \_\_lt\_\_(self, other):

return self.fitness < other.fitness

**Figure 7. genetic\_algo.py**

from genetic\_alogithm.richardson import \*

import copy

logger = logging.getLogger()

class GeneticAlgorithm:

def \_\_init\_\_(self, population, crossover\_rate, mutation\_rate, normalize):

logger.info(f"Population: {population} Crossover: {crossover\_rate} Mutation Rate: {mutation\_rate}")

self.population = []

self.crossover\_rate = crossover\_rate

for i in range(population):

self.population.append(Richardson(mutation\_rate=mutation\_rate, normalize=normalize))

self.best\_individuals = []

self.average\_fitness = []

def crossover(self):

"""

Apply crossover as per the crossover ratio.

Some possible ways to change the crossover:

1. apply average of both values to each attribute

2. only allow individuals to crossover one.

:return:

"""

pass

for i in range(len(self.population)):

# Check if we perform crossover

if random.uniform(0, 1) <= self.crossover\_rate:

# Make sure that we don't select current individual

index = random.randint(0, len(self.population)-1)

while index == i:

index = random.randint(0, len(self.population)-1)

# Then we swap attributes of two individuals

temp\_name1 = random.choice(['x', 'y', 'z'])

random\_country1 = self.population[i].\_\_getattribute\_\_(temp\_name1)

random\_value1 = random.choice(

['expend', 'econ\_rest', 'k\_self', 'k\_others'])

value1 = random\_country1.\_\_getattribute\_\_(random\_value1)

temp\_name2 = random.choice(['x', 'y', 'z'])

random\_country2 = self.population[index].\_\_getattribute\_\_(temp\_name2)

random\_value2 = random.choice(

['expend', 'econ\_rest', 'k\_self', 'k\_others'])

value2 = random\_country2.\_\_getattribute\_\_(random\_value2)

logger.debug(f"Crossing individual {i}'s {random\_value1} from country {temp\_name1}: {value1} "

f"to individual {index}'s {random\_value2} from country {temp\_name2}: {value2}")

# swap the two values

temp = value1

value1 = value2

value2 = temp

# set them back to the right individual

random\_country1.\_\_setattr\_\_(random\_value1, value1)

random\_country2.\_\_setattr\_\_(random\_value2, value2)

def mutate(self):

"""

We mutate each individual in the population

:return:

"""

for i in range(len(self.population)):

self.population[i].mutate(individual=i)

def selection\_tournament(self):

"""

Selection Tournament with Replacement

:return:

"""

for i in range(len(self.population)):

self.population[i].perform\_calculations()

fitness\_vals = ",".join([str(i.fitness) for i in self.population])

logger.info(f"New fitness values: {fitness\_vals}")

new\_generation = []

# Currently only 1v1 selection

for i in range(len(self.population)):

# Make sure that we don't select current individual

individual1 = random.randint(0, len(self.population)-1)

while individual1 == i:

individual1 = random.randint(0, len(self.population)-1)

# WE WANT TO ADD LOWER FITNESS. LOWER IS MORE STABLE

if self.population[i].fitness <= self.population[individual1].fitness:

new\_generation.append(copy.deepcopy(self.population[i]))

else:

new\_generation.append(copy.deepcopy(self.population[individual1]))

self.population = new\_generation

def calculate\_best(self):

avg\_fitness = 0

for i in self.population:

avg\_fitness += i.fitness

self.average\_fitness.append(avg\_fitness/len(self.population))

logger.info(f"Average Fitness: {self.average\_fitness[-1]}")

self.best\_individuals += self.population

self.best\_individuals.sort(key=lambda x: x.fitness)

self.best\_individuals = self.best\_individuals[:10]

fitness\_vals = ",".join([str(i.fitness) for i in self.best\_individuals])

logger.info(f"Best Fitness: {fitness\_vals}")

def iterate\_population(self):

self.crossover()

self.mutate()

self.selection\_tournament()

self.calculate\_best()

Figure 7. assignment1.py

from genetic\_alogithm.genetic\_algo import \*

from genetic\_alogithm import logger

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import argparse

# Change these as you wish :)

POPULATION = 75

CROSSOVER\_RATE = 0.4

MUTATION\_RATE = 0.8

NUM\_GENERATIONS = 100

def parse\_my\_args():

parser = argparse.ArgumentParser()

parser.add\_argument('-hide',

default=False,

action='store\_true',

help='If true, hides graph output')

parser.add\_argument('-normalize',

default=False,

action='store\_true',

help='If true, normalize fitness function')

parser.add\_argument('-population',

type=int,

default=POPULATION,

help='Population Size')

parser.add\_argument('-generations',

type=int,

default=NUM\_GENERATIONS,

help='Number of Generations')

parser.add\_argument('-crossover',

type=float,

default=CROSSOVER\_RATE,

help='Crossover Rate')

parser.add\_argument('-mutation',

type=float,

default=MUTATION\_RATE,

help='Mutation Rate')

return parser.parse\_args()

if \_\_name\_\_ == "\_\_main\_\_":

# Init Logging and args

args = parse\_my\_args()

log\_file = "./logs/gen\_alg.log"

logger.create\_rotating\_log(log\_file)

log = logger.logger

log.info("----- Starting Genetic Algorithm -----")

# Perform GA

g = GeneticAlgorithm(args.population, args.crossover, args.mutation, args.normalize)

for x in range(args.generations):

g.iterate\_population()

# Log average fitness across all generations

log.info("Average fitness across all generations")

log.info(g.average\_fitness)

# Ok now so we have the best people and their params. Lets log them

log.info("Best individuals across all generations")

for x in g.best\_individuals:

log.info(x.get\_county\_props())

best\_fitness = g.best\_individuals[0].fitness

average\_fitness = np.average(g.average\_fitness)

# Lets study how these params work over time.

x\_no\_reset = []

y\_no\_reset = []

z\_no\_reset = []

fitness\_no\_reset = []

# We will only consider the fittest individual in the calculation

for x in range(args.generations):

x\_no\_reset.append(g.best\_individuals[0].x.curr)

y\_no\_reset.append(g.best\_individuals[0].y.curr)

z\_no\_reset.append(g.best\_individuals[0].z.curr)

fitness\_no\_reset.append(g.best\_individuals[0].fitness)

g.best\_individuals[0].perform\_calculations()

# reset current spending so only 12 params remain.

g.best\_individuals[0].reset\_current\_spending()

x\_reset = []

y\_reset = []

z\_reset = []

fitness\_reset = []

for x in range(args.generations):

x\_reset.append(g.best\_individuals[0].x.curr)

y\_reset.append(g.best\_individuals[0].y.curr)

z\_reset.append(g.best\_individuals[0].z.curr)

fitness\_reset.append(g.best\_individuals[0].fitness)

g.best\_individuals[0].perform\_calculations()

# Let's calculate how far off the lines are from eventually coming to a steady state.

x\_nrmsd = np.sqrt(

np.mean((np.array(x\_reset) - np.array(x\_no\_reset))) \*\* 2) \

/ (np.max(x\_no\_reset))

y\_nrmsd = np.sqrt(

np.mean((np.array(y\_reset) - np.array(y\_no\_reset))) \*\* 2) \

/ (np.max(y\_no\_reset))

z\_nrmsd = np.sqrt(

np.mean((np.array(z\_reset) - np.array(z\_no\_reset))) \*\* 2) \

/ (np.max(z\_no\_reset))

fitness\_nrmsd = np.sqrt(

np.mean((np.array(fitness\_reset) - np.array(fitness\_no\_reset))) \*\* 2) \

/ (np.max(fitness\_reset))

# The larger the ssd, the worse the result

country\_nrmsd = (x\_nrmsd + y\_nrmsd + z\_nrmsd)/3

# Final Results

log.info("Population, Crossover, Mutation, Generations, "

"Average Fitness, Best Fitness, Country rmsd, Fitness rmsd, Normalized")

log.info(f"{args.population}\t{args.crossover}\t{args.mutation}\t"

f"{args.generations}\t{average\_fitness}\t{best\_fitness}\t"

f"{country\_nrmsd}\t{fitness\_nrmsd}\t{args.normalize}")

print(f"{args.population},{args.crossover},{args.mutation},"

f"{args.generations},{average\_fitness},{best\_fitness},"

f"{country\_nrmsd},{fitness\_nrmsd},{args.normalize}")

# Display results

if not args.hide:

df = pd.DataFrame({'domain': range(args.generations),

'X without Reset': x\_no\_reset,

'Y without Reset': y\_no\_reset,

'Z without Reset': z\_no\_reset,

'X with Reset': x\_reset,

'Y with Reset': y\_reset,

'Z with Reset': z\_reset})

# multiple line plot

fig, axs = plt.subplots(2, figsize=(12, 8))

fig.suptitle(f"Population: {args.population}\nCrossover: {args.crossover}\nMutation: {args.mutation}")

axs[1].set\_title('Best Individual')

axs[1].plot('domain', 'X without Reset', data=df, marker='', color='blue', linewidth=2, linestyle='dotted')

axs[1].plot('domain', 'Y without Reset', data=df, marker='', color='red', linewidth=2, linestyle='dotted')

axs[1].plot('domain', 'Z without Reset', data=df, marker='', color='green', linewidth=2, linestyle='dotted')

axs[1].plot('domain', 'X with Reset', data=df, marker='', color='blue', linewidth=2)

axs[1].plot('domain', 'Y with Reset', data=df, marker='', color='red', linewidth=2)

axs[1].plot('domain', 'Z with Reset', data=df, marker='', color='green', linewidth=2)

axs[1].legend()

axs[0].set\_title('Average Fitness')

axs[0].plot(range(args.generations), g.average\_fitness, marker='', color='black', linewidth=2)

plt.show(block=True)