Assignment 2 | Linear Regression using Genetic Algorithms

Author : Alapan Sau and Pavani Babburi

For a linear regression model, an initial vector which overfits the training data is given. The aim of the assignment is to find a model which generalises well to unseen data.

Summary

Genetic algorithm is a search technique used to find the optimum model. The main steps involved in creating the genetic algorithm are as follows:

- Initial Population
- Fitness Function
- Selection
- Crossover
- Mutation

Each of the following steps are described below in more detail.

Initial Population

Genetic Algorithm is very stongly guided by the selected Initial Population. The higher the diversity, the more the computational expense for the algorithm to converge, however a less diverse population gets stuck into a local minima and does not cover a lot of search space. So we tried the following two ideas:

Initially we tried with randomly generating the vectors with each data point uniformly chosen between the given range $[-10, 10]$. However the search space being so large, we didn't get any significantly fit vector even after a running a lot of generations.

On failure with our initial attempts, we dropped the above idea. We decided on selecting a sufficiently good vector and creating our first population by adding noise to it randomly. We chose our initital vector to be the given overfitting vector which gave us much better results than before. However, the individuals created from a single overfit vector were mostly overfitting the dataset and were very less diverse. This realisation led us to select more than one vectors to create the initial population. We started constructing the initial population by combining the initial overfit vector with a vector of zeroes and adding normal noise to each individual. The initial vector of zeroes was chosen as it is a vector which performs similarly on both the train and validation set, and such adds enough diversity to the initial population.

After obtaining a good vector, the intial population was generated by adding a sufficiently high normal noise to the vector randomly. After the zero initialisation worked out, we tried to fine tune each position individually. We noticed that positions 3 and 4 were most sensitive to change after trying all possibilities. So, we explored these positions further and noticed that the values of 8 and -1 at positions 3 and 4 respectively gave extremely good results. Using this heuristic, we tried to create a population. The best vector of the population generated was used as the initial vector for the population submitted.

Other methods of initialisation we tried were adding noise to the intial population through various ways including normal, uniform, multiplication with random numbers, etc. However, normal noise gave the best results.

Why Normal Noise?

Initially we tried to incorporate noise by multiplying the given overfit vector by a uniform random value from a certain range $[1 - x, 1 + x]$. The values tried for x include 0.1, 0.5, 0.7, etc. However, we realised that the uniform multiplication was not yeilding proper results as it had certain limitation, like giving equal priority to all the values meant that we had to choose a high value of x, which may not scale similarly to all the weights as they were of different magnitudes. Also, unless we choose significantly high values for x, the numbers remained in the same sign as in the inital overfit vector. The normal noise allowed us to scale better to each individual as we generated a random normal variable from a normal distribution with mean equal to the initial value and standard deviation $|\text{initialValue}|/i$, where the values of i tried include 1,2,5,10, 20, 0.5, etc. The value of i was 10 was for the best model.

Fitness Function

Fitness Function is the most significant attribute of the Genetic Algorithm. This acts as the objective optimisation expression. The essential target of a Regression Model is to optimise the MSE. However this might lead to a model which is overfitting the training data. So its very important to validate the data and reduce the variation.

Based on the above approach, we decided to bring a balance between the two factors into our fitness expression:
a. The training and validation errors
b. The difference between our validation and training error

To do so, we tried and tested with lots of different expressions depending on the amount of diversity in the intial population. We tried the following expressions

$$\implies fitness = w_1 * (train\_error + validation\_error) + w_2 * |train\_error - validation\_error|$$
$$\implies fitness = w_1 * (train\_error^2 + validation\_error^2) + w_2 * (train\_error - validation\_error)^2$$

Here w_1 and w_2 are the weights multiplied to the factors respectively. A trial and error approach was taken to tune them.
The expression that gave us significantly better performance was:
$$\implies fitness = train\_error + validation\_error + 2 * |train\_error - validation\_error|$$

```
def fitness(error):
    return ( (error[0]+ error[1]) + 2 * abs(error[0]- error[1]))
```

Selection

We used two ways of selection for most of our Algorithm in different instances:

Method 1: The selection was done by generating probabilities for each vector based on the fitness values. As the values of the fitness values are relatively large, we used logarithmic functions to bring them to comparable values. We then negated them to give lower fitness values a better probability, and then took the exponential of the negated log values. These exponential values were used to calculate the probabilty. In some cases, we also tried to take the power of the negated probabilities to a different number other than e. Depending on the fitness function used, the logarithm base was decided to ensure proper mapping from fitness to the probabilty.

Pros:

- Preserves diversity as even "bad" vectors have a chance of selection, albeit low.

Cons:

- Depends too much on the proper choosing of the probabilty function. Algorithms starting with good initial values have failed because of improper probabilty function.

Method 2: In this method we select a pool of candidates from each generation. Any individual from that pool is selected with uniform probability for mating. We tried with pools of different sizes at different stages of the algorithm and noticed, smaller pool sizes work better at the initial runs, however, larger pool sizes add higher diversity in the higher generations which helps to avoid local minimas.

Pros:

- Allows us to eliminate vectors which are among the worst performers in that generation
- Not too much importance given to probabilty calculation.

Cons:

- Pool Size becomes a hyperparameter to be tuned. Low pool size eliminates diversity while a high pool size negatively impacts the future generation as the selection is done uniformly. Both the effects have to be weighed and a proper pool size has to be choosen.

For Best Vector

We used a pool size of `10` for a population size of `20`. The probabilty function (not used for selection, but used for crossover) was calculated by doing logarithm to base `10`, and power to base `e`.

```
def calculate_generation_probability(population_fitness):
    population_log = np.log10(population_fitness)
    population_inverse_log = -population_log
    population_exp = np.exp(population_inverse_log)

    population_probability = population_exp/ np.sum(population_exp)
    return population_probability

def selection(population,population_probability):
    df = pd.DataFrame()
    df['population'] = population
    df['population_probability'] = population_probability
    df = df.sort_values(by = ['population_probability'])
    pool = (df.tail(10).index)
    parent1, parent2 = np.random.choice(pool, 2)
    return parent1, parent2
```

Crossover

The crossover used was simulated binary crossover, which is a weighted average of the two parents based on their fitness values or probability values. We also tried single point crossover as well as selecting each position randomly from either of the two parents with probability proportional to the probability used for selection. When pool size was used, we also tried to take a direct average of the parents at the later stages of the algorithm. For the best vector, we used a simulated binary crossover with weights equivalent to the probabilty values. The probability values give better results as they give better weightage to each parent, compared to the fitness values which, due to the large values possible, penalise some parents unfairly.

```
def crossover(parent1, parent2, error1, error2):
    weight1 = error2/ (error1 + error2)
    weight2 = error1/ (error1 + error2)

    return (parent1*weight1 + weight2*parent2)/(weight1+weight2)
```

Mutation

Genetic Algorithms often tend to get stuck into local minimas. After running for a considerable number of generations, the search space decreases significantly leading to this trouble.To avoid this problem we added a mutation to every offspring after the crossover. This mutation was done by adding noise randomly to the data points. We tested various probability distributations like uniform, gaussian etc. Normal Distribution performed considerably better as mentioned above for initital population. The standard deviation of the normal distribution was tuned and decided to be $val/20$ for a point with value of val.
The mutation was decreased in the later models compared to the initial ones as the initial population gave better results and thus, the need for diversification decreased.

```
def mutate(child):
    for i in range(0,11):
        if random.random() <= 6/11:
            child[i] = np.random.normal(child[i], abs(child[i]/20))
            child[i] = max(min(child[i],10),-10)
    return child
```

Hyper Paramters:

Initially we selected our parameters from a diverse set of values. However, with experirnce earned through multiple trials, we selected the best parameters which were also used in most of our later runs of the algorithm. These parameters are as follows:

- `Population Size : 20`
- `Pool Size : 10`
- `w1` (Weight of (`training_error + validation_error`) in fitness): 1
- `w2` (Weight of (`abs(training_error - validation_error)`) in fitness): 2

Error Acheived

The vectors submitted have both training and validation errors around `1.1e11` . We believe this vector is not overfitting the training data as the results obtained on the training and validation set are similar. The performance on the unseen leaderboard data is also good, which eliminates the possibiltiy of overfitting both training and validation sets.

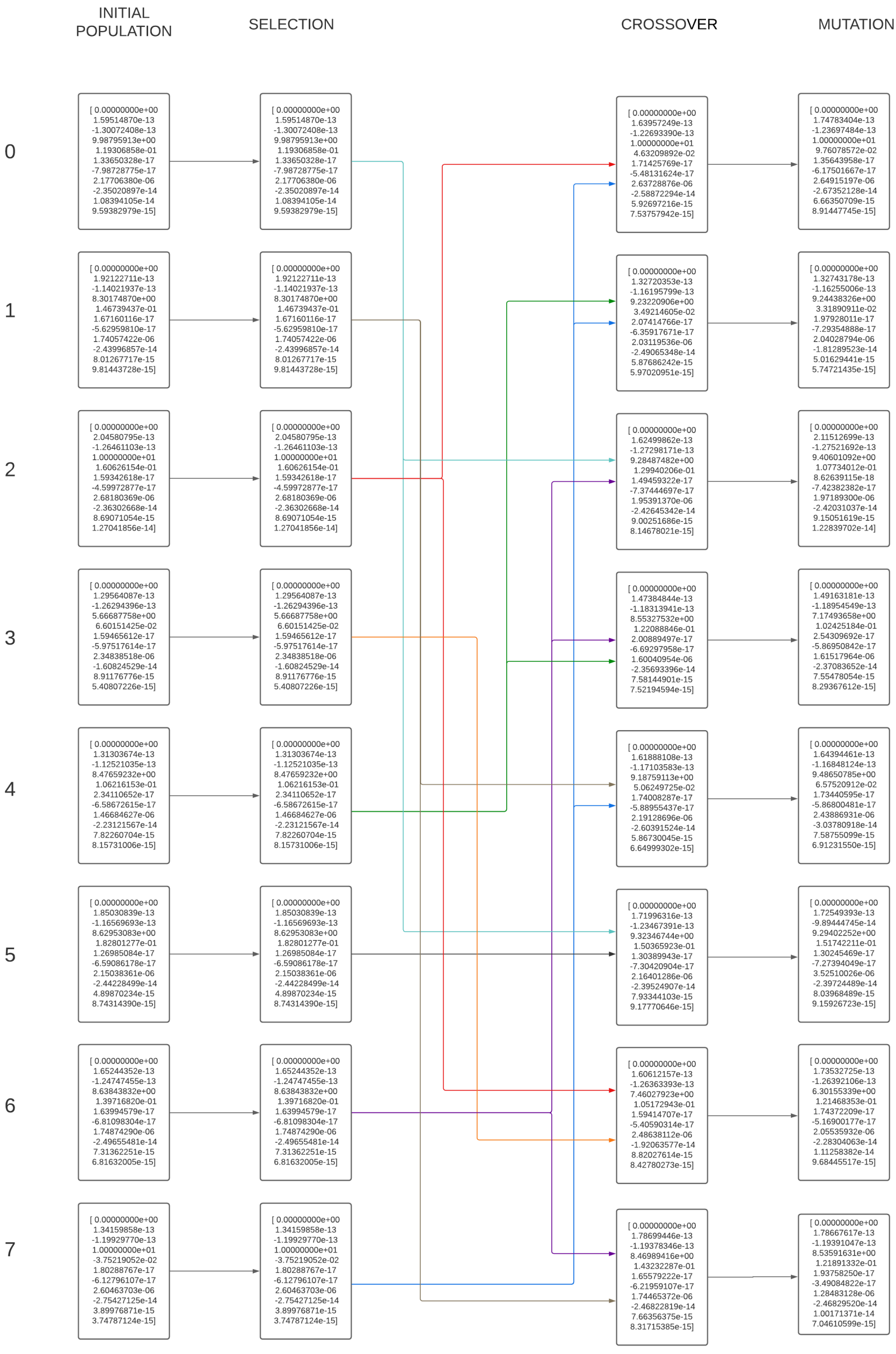
Iterations of the Algorithm

We are exhibiting three iterations of one instance of our genetic algorithm run. The `population_size` was decided to be `8`. The a `pool_size` of 8 was choses to have uniform probability of selection. Therefore each individual had a `selection probability` of `12.5%`. Each data point had a `mutation probability` of `6/11`.

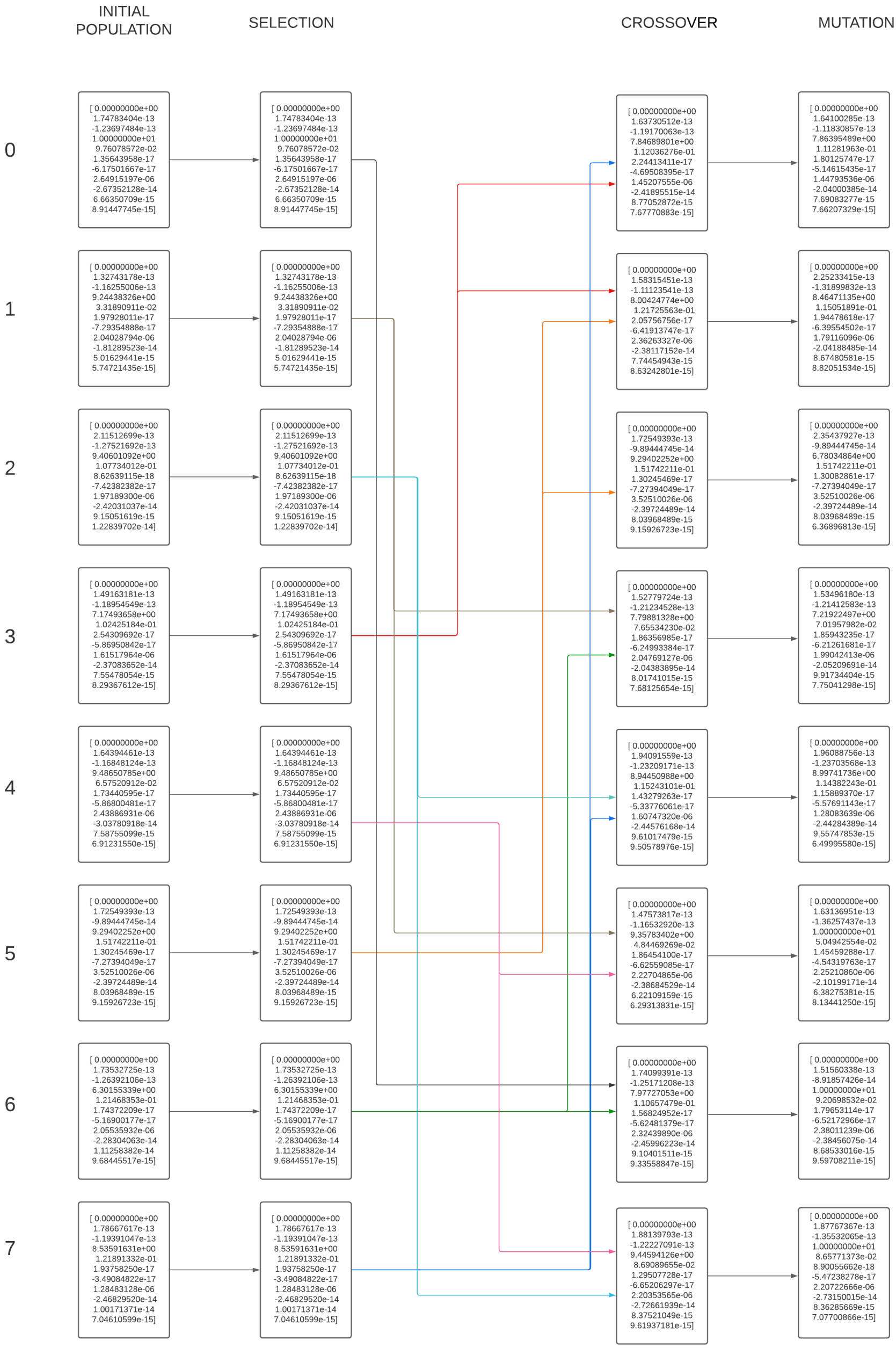
Iteration 1:


-6.71273534e-17
2.48455594e-06
-2.35020897e-14
1.08394105e-14
9.59382979e-15]
-6.71273534e-17
2.48455594e-06
-2.35020897e-14
1.08394105e-14
9.59382979e-15]
-5.19812041e-17
2.21654010e-06
-2.43818404e-14
7.49676789e-15
3.70029202e-15]
-6.12796107e-17
2.60463703e-06
-2.75427125e-14
3.89976871e-15
3.74787124e-15]

Iteration 2:


INITIAL POPULATION
SELECTION
CROSSOVER
MUTATION
0
1
2
3
4
5
6
7

Iteration 3


INITIAL POPULATION
SELECTION
CROSSOVER
MUTATION
0
[0.00000000e+00 1.74783404e-13 -1.23697484e-13 1.00000000e+01 9.76078572e-02 1.35643958e-17 -6.17501667e-17 2.64915197e-06 -2.67352128e-14 6.66350709e-15 8.91447745e-15]
[0.00000000e+00 1.74783404e-13 -1.23697484e-13 1.00000000e+01 9.76078572e-02 1.35643958e-17 -6.17501667e-17 2.64915197e-06 -2.67352128e-14 6.66350709e-15 8.91447745e-15]
[0.00000000e+00 1.63730512e-13 -1.19170063e-13 7.84689801e+00 1.12036276e-01 2.24413411e-17 -4.69508395e-17 1.45207555e-06 -2.41895515e-14 8.77052872e-15 7.67770883e-15]
[0.00000000e+00 1.64100285e-13 -1.11830857e-13 7.86395489e+00 1.11281963e-01 1.80125747e-17 -5.14615435e-17 1.44793536e-06 -2.04000385e-14 7.69083277e-15 7.66207329e-15]
1
[0.00000000e+00 1.32743178e-13 -1.16255006e-13 9.24438326e+00 3.31890911e-02 1.97928011e-17 -7.29354888e-17 2.04028794e-06 -1.81289523e-14 5.01629441e-15 5.74721435e-15]
[0.00000000e+00 1.32743178e-13 -1.16255006e-13 9.24438326e+00 3.31890911e-02 1.97928011e-17 -7.29354888e-17 2.04028794e-06 -1.81289523e-14 5.01629441e-15 5.74721435e-15]
[0.00000000e+00 1.58315451e-13 -1.11123541e-13 8.00424774e+00 1.21725563e-01 2.05756756e-17 -6.41913747e-17 2.36263327e-06 -2.38117152e-14 7.74454943e-15 8.63242801e-15]
[0.00000000e+00 2.25233415e-13 -1.31899832e-13 8.46471135e+00 1.15051891e-01 1.94478618e-17 -6.39554502e-17 1.79116096e-06 -2.04188485e-14 8.67480581e-15 8.82051534e-15]
2
[0.00000000e+00 2.11512699e-13 -1.27521692e-13 9.40601092e+00 1.07734012e-01 8.62639115e-18 -7.42382382e-17 1.97189300e-06 -2.42031037e-14 9.15051619e-15 1.22839702e-14]
[0.00000000e+00 2.11512699e-13 -1.27521692e-13 9.40601092e+00 1.07734012e-01 8.62639115e-18 -7.42382382e-17 1.97189300e-06 -2.42031037e-14 9.15051619e-15 1.22839702e-14]
[0.00000000e+00 1.72549393e-13 -9.89444745e-14 9.29402252e+00 1.51742211e-01 1.30245469e-17 -7.27394049e-17 3.52510026e-06 -2.39724489e-14 8.03968489e-15 9.15926723e-15]
[0.00000000e+00 2.35437927e-13 -9.89444745e-14 6.78034864e+00 1.51742211e-01 1.30082861e-17 -7.27394049e-17 3.52510026e-06 -2.39724489e-14 8.03968489e-15 6.36896813e-15]
3
[0.00000000e+00 1.49163181e-13 -1.18954549e-13 7.17493658e+00 1.02425184e-01 2.54309692e-17 -5.86950842e-17 1.61517964e-06 -2.37083652e-14 7.55478054e-15 8.29367612e-15]
[0.00000000e+00 1.49163181e-13 -1.18954549e-13 7.17493658e+00 1.02425184e-01 2.54309692e-17 -5.86950842e-17 1.61517964e-06 -2.37083652e-14 7.55478054e-15 8.29367612e-15]
[0.00000000e+00 1.52779724e-13 -1.21234528e-13 7.79881328e+00 7.65534230e-02 1.86356985e-17 -6.24993384e-17 2.04769127e-06 -2.04383895e-14 8.01741015e-15 7.68125654e-15]
[0.00000000e+00 1.53496180e-13 -1.21412583e-13 7.21922497e+00 7.01957982e-02 1.85943235e-17 -6.21261681e-17 1.99042413e-06 -2.05209691e-14 9.91734404e-15 7.75041298e-15]
4
[0.00000000e+00 1.64394461e-13 -1.16848124e-13 9.48650785e+00 6.57520912e-02 1.73440595e-17 -5.86800481e-17 2.43886931e-06 -3.03780918e-14 7.58755099e-15 6.91231550e-15]
[0.00000000e+00 1.64394461e-13 -1.16848124e-13 9.48650785e+00 6.57520912e-02 1.73440595e-17 -5.86800481e-17 2.43886931e-06 -3.03780918e-14 7.58755099e-15 6.91231550e-15]
[0.00000000e+00 1.94091559e-13 -1.23209171e-13 8.94450988e+00 1.15243101e-01 1.43279263e-17 -5.33776061e-17 1.60747320e-06 -2.44576168e-14 9.61017479e-15 9.50578976e-15]
[0.00000000e+00 1.96088756e-13 -1.23703568e-13 8.99741736e+00 1.14382243e-01 1.15889370e-17 -5.57691143e-17 1.28083639e-06 -2.44284389e-14 9.55747853e-15 6.49995580e-15]
5
[0.00000000e+00 1.72549393e-13 -9.89444745e-14 9.29402252e+00 1.51742211e-01 1.30245469e-17 -7.27394049e-17 3.52510026e-06 -2.39724489e-14 8.03968489e-15 9.15926723e-15]
[0.00000000e+00 1.72549393e-13 -9.89444745e-14 9.29402252e+00 1.51742211e-01 1.30245469e-17 -7.27394049e-17 3.52510026e-06 -2.39724489e-14 8.03968489e-15 9.15926723e-15]
[0.00000000e+00 1.47573817e-13 -1.16532920e-13 9.35783402e+00 4.84469269e-02 1.86454100e-17 -6.62559085e-17 2.22704865e-06 -2.38684529e-14 6.22109159e-15 6.29313831e-15]
[0.00000000e+00 1.63136951e-13 -1.36257437e-13 1.00000000e+01 5.04942554e-02 1.45459288e-17 -4.54319763e-17 2.25210860e-06 -2.10199171e-14 6.38275381e-15 8.13441250e-15]
6
[0.00000000e+00 1.73532725e-13 -1.26392106e-13 6.30155339e+00 1.21468353e-01 1.74372209e-17 -5.16900177e-17 2.05535932e-06 -2.28304063e-14 1.11258382e-14 9.68445517e-15]
[0.00000000e+00 1.73532725e-13 -1.26392106e-13 6.30155339e+00 1.21468353e-01 1.74372209e-17 -5.16900177e-17 2.05535932e-06 -2.28304063e-14 1.11258382e-14 9.68445517e-15]
[0.00000000e+00 1.74099391e-13 -1.25171208e-13 7.97727053e+00 1.10657479e-01 1.56824952e-17 -5.62481379e-17 2.32439890e-06 -2.45996223e-14 9.10401511e-15 9.33558847e-15]
[0.00000000e+00 1.51560338e-13 -8.91857426e-14 1.00000000e+01 9.20698532e-02 1.79653114e-17 -6.52172966e-17 2.38011239e-06 -2.38456075e-14 8.68533016e-15 9.59708211e-15]
7
[0.00000000e+00 1.78667617e-13 -1.19391047e-13 8.53591631e+00 1.21891332e-01 1.93758250e-17 -3.49084822e-17 1.28483128e-06 -2.46829520e-14 1.00171371e-14 7.04610599e-15]
[0.00000000e+00 1.78667617e-13 -1.19391047e-13 8.53591631e+00 1.21891332e-01 1.93758250e-17 -3.49084822e-17 1.28483128e-06 -2.46829520e-14 1.00171371e-14 7.04610599e-15]
[0.00000000e+00 1.88139793e-13 -1.22227091e-13 9.44594126e+00 8.69089655e-02 1.29507728e-17 -6.65206297e-17 2.20353565e-06 -2.72661939e-14 8.37521049e-15 9.61937181e-15]
[0.00000000e+00 1.87767367e-13 -1.35532065e-13 1.00000000e+01 8.65771373e-02 8.90055662e-18 -5.47238278e-17 2.20722666e-06 -2.73150015e-14 8.36285669e-15 7.07700866e-15]