report.md

Genetic Algorithm Project

Team 38 - GENTOOmen

Roll No.	Name
2019114001	**Akshett Rai Jindal**
2019111031	**Zishan Kazi**

Task

The task was to get the weights of the features for a model, using Genetic Algorithms provided that the weights of the features corresponding to an overfit model are given to us.

Running the Program

```
$ python3 main.py
```

The output of the program is `output.json` which contains the weight vectors of the last generation.

About G.A.

A **genetic algorithm** is a search heuristic that is inspired by Charles Darwin's theory of natural evolution.

We have implemented it as follows:

- Initialize a population
- Determine fitness of population
- Until convergence *repeat* the following:
  - **Select parents** from the population
  - **Crossover** to generate children vectors
  - **Mutate** the new population
  - Calculate **fitness** for new population

Each population contains multiple individuals where each individual represents a point in search space and possible solution.

Every individual is a vector of size `11` with `11` floating point values in the range of `[-10, 10]`. These 11 values represent the weights of the different features of the 11 features of the data.

Selection

Selection is basically taking the individuals with best fitness and pass their genes/bits onto next generations to maintain good fitness scores.

Crossover

Here two vectors generate an offspring. These two vectors are taken from `Selection` function. The two vectors are then manipulated to generate an offspring.

Mutations

We change each bit of the vector with a probability to diversify the population and to prevent similar vectors which can led to convergence.

Code

The first population is created using an initial given overfit vector. Copies of this vector are made on which we mutate to generate a population of size `POPULATION_SIZE`.

For each vector, at every index mutation is performed with a probability of `6/10`. Then the value at that index is replaced with the value at the overfit vector at that index multiplied by some factor chosen uniformly between (0.7, 1.3).

The **fitness** of the population is an linear combination of the train error and the validation error.

After the population is initialized, a mating pool is made containing the top `MATING_POOL_SIZE` parents sorted according to their fitness.

```
population_fitness = population_fitness[np.argsort(population_fitness[:,-1])]
mating_pool = population_fitness[:MATING_POOL_SIZE]
```

Then parents are uniformly chosen from the mating pool as follows:

```
index1 = random.randint(0, MATING_POOL_SIZE-1)
index2 = random.randint(0, MATING_POOL_SIZE-1)

parent1 = mating_pool[index1]
parent2 = mating_pool[index2]
```

Then offsprings are produced from the parents by applying **Simulated Binary Crossover** on them. This is followed by mutation of chromosomes, details of which are given here.

The new population is created by choosing `POPULATION_SIZE - PASSED_FROM_PARENTS` top children generated and `PASSED_FROM_PARENTS` top parents.

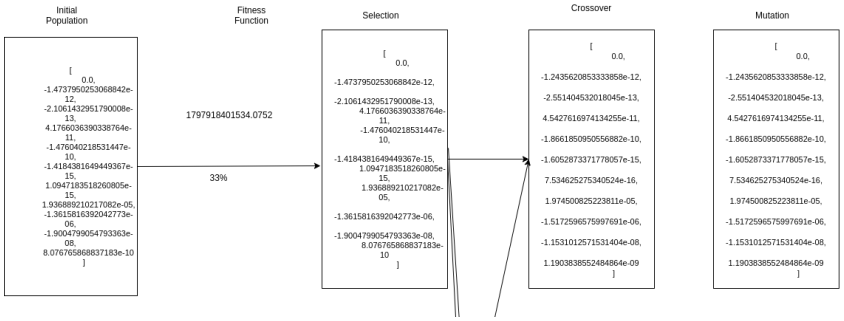
```
def create_next_gen(parents_fitness, children):
    child_fitness = get_population_fitness(children)[:(POPULATION_SIZE-PASSED_FROM_PARENTS)]
    parents_fitness = parents_fitness[:PASSED_FROM_PARENTS]
    generation = np.concatenate((parents_fitness, child_fitness))
    generation = generation[np.argsort(generation[:,-1])]
    return generation
```

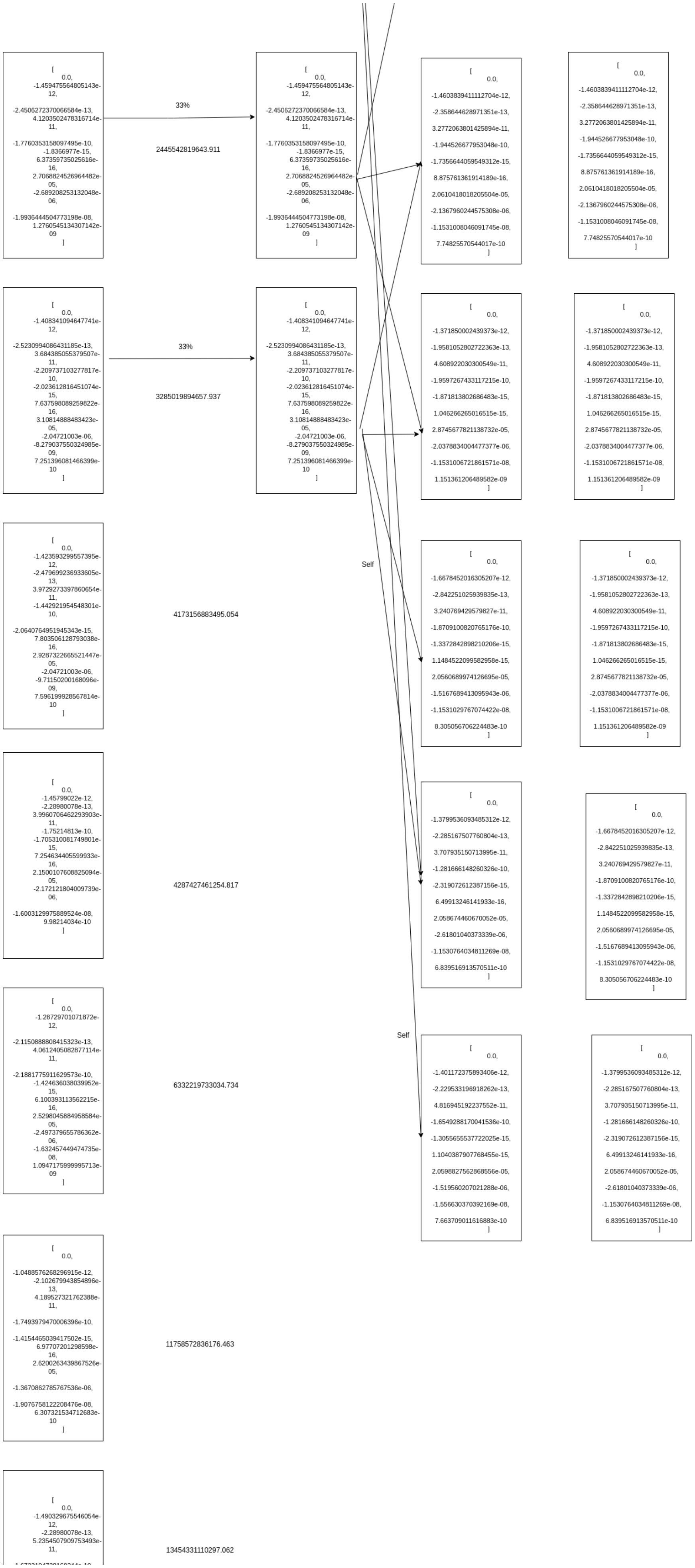
This process is repeated and the values are stored in a JSON object file from which it is read for next run of the program.

As you can see the code is **vectorized** and **completely modular** as separate function have been written for each significant step.

Iteration Diagrams

First Iteration




[0.0, -1.459475564805143e-12, -2.4506272370066584e-13, 4.1203502478316714e-11, -1.7760353158097495e-10, -1.8366977e-15, 6.37359735025616e-16, 2.7068824526964482e-05, -2.689208253132048e-06, -1.9936444504773198e-08, 1.2760545134307142e-09]
33%
2445542819643.911
[0.0, -1.459475564805143e-12, -2.4506272370066584e-13, 4.1203502478316714e-11, -1.7760353158097495e-10, -1.8366977e-15, 6.37359735025616e-16, 2.7068824526964482e-05, -2.689208253132048e-06, -1.9936444504773198e-08, 1.2760545134307142e-09]
[0.0, -1.4603839411112704e-12, -2.358644628971351e-13, 3.2772063801425894e-11, -1.944526677953048e-10, -1.7356644059549312e-15, 8.875761361914189e-16, 2.0610418018205504e-05, -2.1367960244575308e-06, -1.1531008046091745e-08, 7.74825570544017e-10]
[0.0, -1.4603839411112704e-12, -2.358644628971351e-13, 3.2772063801425894e-11, -1.944526677953048e-10, -1.7356644059549312e-15, 8.875761361914189e-16, 2.0610418018205504e-05, -2.1367960244575308e-06, -1.1531008046091745e-08, 7.74825570544017e-10]
[0.0, -1.408341094647741e-12, -2.5230994086431185e-13, 3.684385055379507e-11, -2.209737103277817e-10, -2.023612816451074e-15, 7.637598089259822e-16, 3.10814888483423e-05, -2.04721003e-06, -8.279037550324985e-09, 7.251396081466399e-10]
33%
3285019894657.937
[0.0, -1.408341094647741e-12, -2.5230994086431185e-13, 3.684385055379507e-11, -2.209737103277817e-10, -2.023612816451074e-15, 7.637598089259822e-16, 3.10814888483423e-05, -2.04721003e-06, -8.279037550324985e-09, 7.251396081466399e-10]
[0.0, -1.371850002439373e-12, -1.9581052802722363e-13, 4.608922030300549e-11, -1.9597267433117215e-10, -1.871813802686483e-15, 1.046266265016515e-15, 2.8745677821138732e-05, -2.0378834004477377e-06, -1.1531006721861571e-08, 1.151361206489582e-09]
[0.0, -1.371850002439373e-12, -1.9581052802722363e-13, 4.608922030300549e-11, -1.9597267433117215e-10, -1.871813802686483e-15, 1.046266265016515e-15, 2.8745677821138732e-05, -2.0378834004477377e-06, -1.1531006721861571e-08, 1.151361206489582e-09]
[0.0, -1.423593299557395e-12, -2.479699236933605e-13, 3.972927339786054e-11, -1.442921954548301e-10, -2.0640764951945343e-15, 7.803506128793038e-16, 2.9287322665521447e-05, -2.04721003e-06, -9.71150200168096e-09, 7.596199928567814e-10]
4173156883495.054
Self
[0.0, -1.6678452016305207e-12, -2.842251025939835e-13, 3.240769429579827e-11, -1.8709100820765176e-10, -1.3372842898210206e-15, 1.1484522099582958e-15, 2.0560689974126695e-05, -1.5167689413095943e-06, -1.1531029767074422e-08, 8.305056706224483e-10]
[0.0, -1.371850002439373e-12, -1.9581052802722363e-13, 4.608922030300549e-11, -1.9597267433117215e-10, -1.871813802686483e-15, 1.046266265016515e-15, 2.8745677821138732e-05, -2.0378834004477377e-06, -1.1531006721861571e-08, 1.151361206489582e-09]
[0.0, -1.45799022e-12, -2.28980078e-13, 3.9960706462293903e-11, -1.75214813e-10, -1.705310081749801e-15, 7.254634405599933e-16, 2.1500107608825094e-05, -2.172121804009739e-06, -1.6003129975889524e-08, 9.98214034e-10]
4287427461254.817
[0.0, -1.3799536093485312e-12, -2.285167507760804e-13, 3.707935150713995e-11, -1.281666148260326e-10, -2.319072612387156e-15, 6.49913246141933e-16, 2.058674460670052e-05, -2.61801040373339e-06, -1.1530764034811269e-08, 6.839516913570511e-10]
[0.0, -1.6678452016305207e-12, -2.842251025939835e-13, 3.240769429579827e-11, -1.8709100820765176e-10, -1.3372842898210206e-15, 1.1484522099582958e-15, 2.0560689974126695e-05, -1.5167689413095943e-06, -1.1531029767074422e-08, 8.305056706224483e-10]
[0.0, -1.28729701071872e-12, -2.1150888808415323e-13, 4.0612405082877114e-11, -2.1881775911629573e-10, -1.424636038039952e-15, 6.100393113562215e-16, 2.5298045884958584e-05, -2.497379655786362e-06, -1.632457449474735e-08, 1.0947175999995713e-09]
6332219733034.734
Self
[0.0, -1.401172375893406e-12, -2.229533196918262e-13, 4.816945192237552e-11, -1.6549288170041536e-10, -1.3055655537722025e-15, 1.1040387907768455e-15, 2.0598827562868556e-05, -1.519560207021288e-06, -1.556630370392169e-08, 7.663709011616883e-10]
[0.0, -1.3799536093485312e-12, -2.285167507760804e-13, 3.707935150713995e-11, -1.281666148260326e-10, -2.319072612387156e-15, 6.49913246141933e-16, 2.058674460670052e-05, -2.61801040373339e-06, -1.1530764034811269e-08, 6.839516913570511e-10]
[0.0, -1.0488576268296915e-12, -2.102679943854896e-13, 4.189527321762388e-11, -1.7493979470006396e-10, -1.4154465039417502e-15, 6.97707201298598e-16, 2.6200263439867526e-05, -1.3670862785767536e-06, -1.9076758122208476e-08, 6.307321534712683e-10]
11758572836176.463
[0.0, -1.490329675546054e-12, -2.28980078e-13, 5.2354507909753493e-11,
13454331110297.062

[
0.0,
-1.6722194728168244e-10,
-2.2411262591141495e-15,
8.52944406e-16,
2.1436906960243266e-05,
-2.047210036-06,
-1.59792834e-08,
9.104897150956366e-10
]

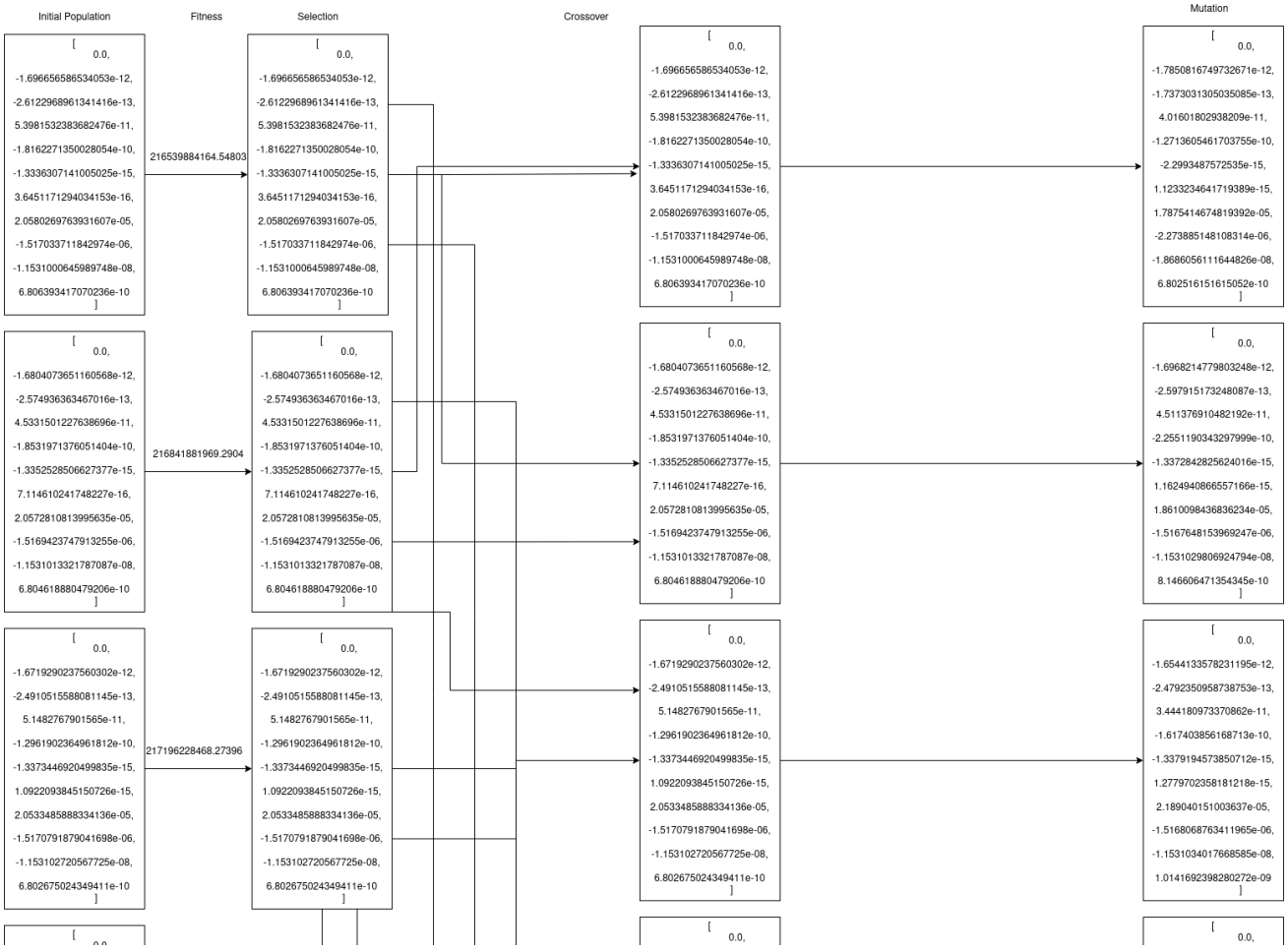
[
0.0,
-1.4201706752360025e-12,
-2.1710402404103709e-13,
5.1150515278778 52e-11,
-1.75214813e-10,
-1.7948359142041 54e-15,
8.9939484479268 66e-16,
3.0193608074907925e-05,
-2.5274155443288565e-06,
-1.613702423903899e-08,
1.2923362109872806e-09
]

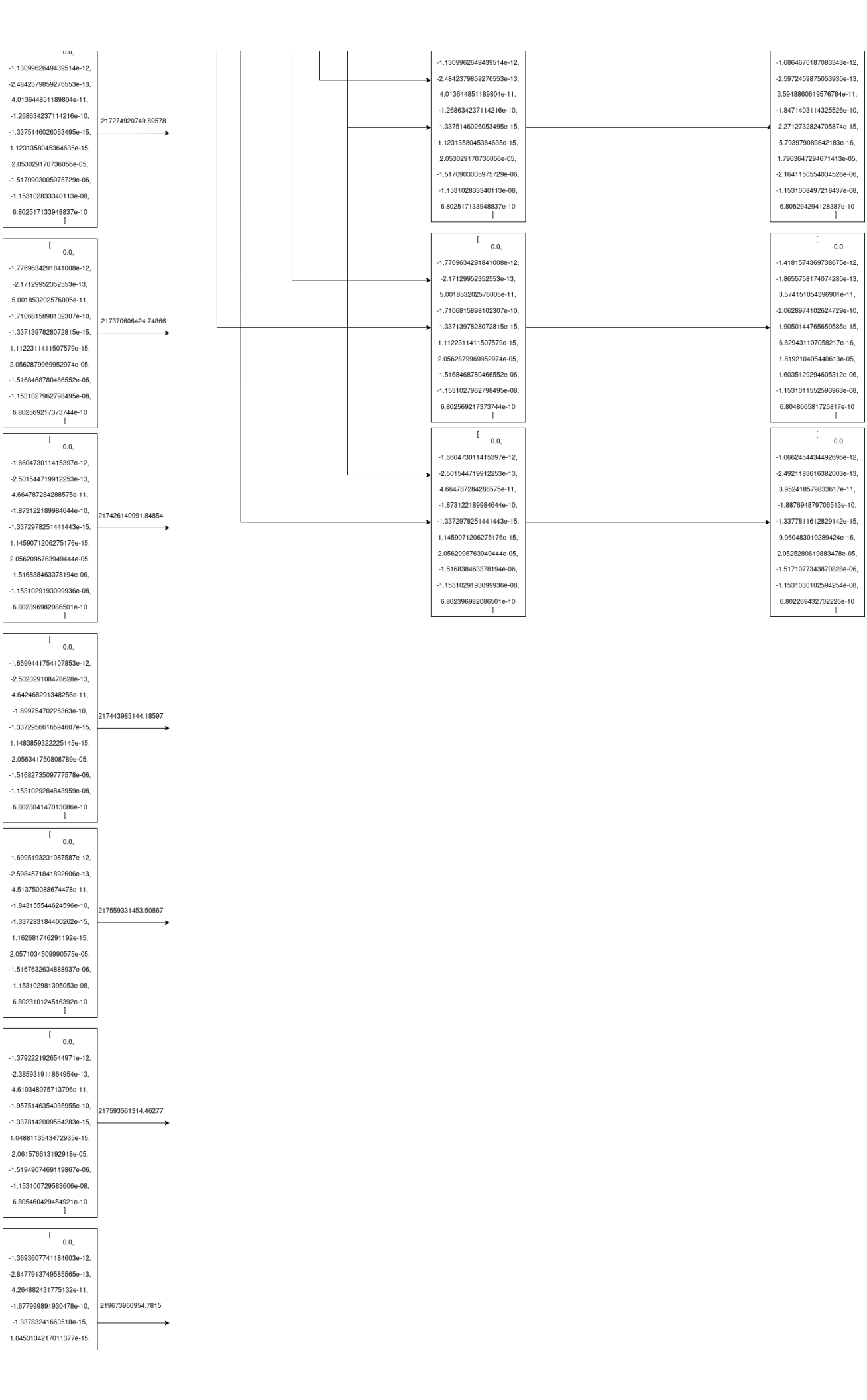
34111041175538.746

[
0.0,
-1.4743657583252537e-12,
-2.1018964150770835e-13,
4.1789191926787864e-11,
-1.75214813e-10,
-1.4110980365430348e-15,
6.9539585136442 53e-16,
1.5458390338717505e-05,
-1.3537868720898707e-06,
-1.9110452623441447e-08,
1.0689463754872 54e-09
]

87245049445526.34

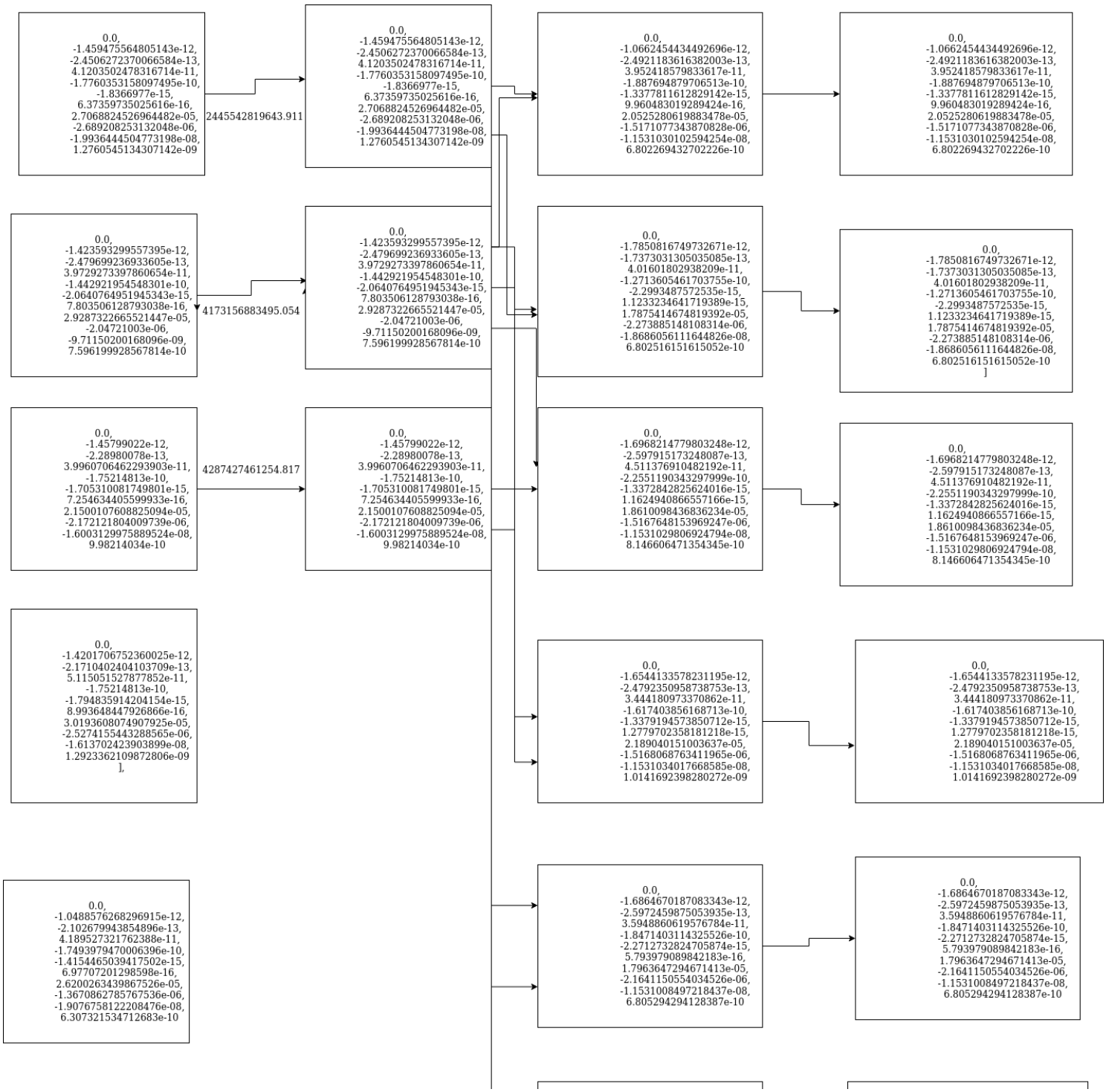
Second Iteration

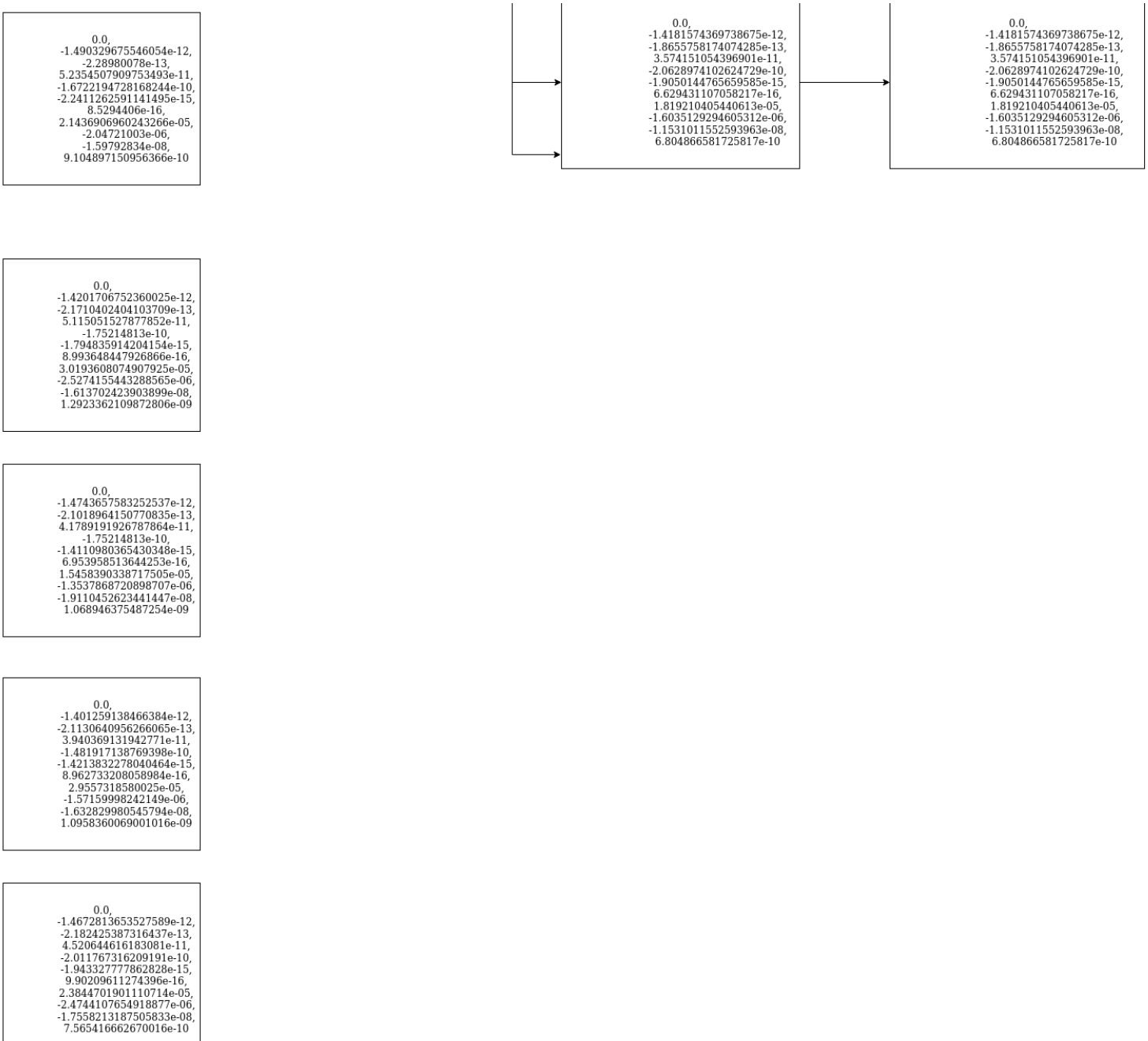



0.0,
-1.1309962649439514e-12,
-2.4842379859276553e-13,
4.013644851189804e-11,
-1.268634237114216e-10,
-1.3375146026053495e-15,
1.1231358045364635e-15,
2.053029170736056e-05,
-1.5170903005975729e-06,
-1.153102833340113e-08,
6.802517133948837e-10
]
217274920749.89578
-1.6864670187083343e-12,
-2.5972459875053935e-13,
3.5948860619576784e-11,
-1.8471403114325526e-10,
-2.2712732824705874e-15,
5.793979089842183e-16,
1.7963647294671413e-05,
-2.1641150554034526e-06,
-1.1531008497218437e-08,
6.805294294128387e-10
[
0.0,
-1.7769634291841008e-12,
-2.17129952352553e-13,
5.001853202576005e-11,
-1.7106815898102307e-10,
-1.3371397828072815e-15,
1.1122311411507579e-15,
2.0562879969952974e-05,
-1.5168468780466552e-06,
-1.1531027962798495e-08,
6.802569217373744e-10
]
217370606424.74866
[
0.0,
-1.4181574369738675e-12,
-1.8655758174074285e-13,
3.5741510543969016e-11,
-2.0628974102624729e-10,
-1.9050144765659585e-15,
6.629431107058217e-16,
1.8192104054406l3e-05,
-1.6035129294605312e-06,
-1.1531011552593963e-08,
6.804866581725817e-10
]
[
0.0,
-1.6604730114153397e-12,
-2.5015447199122253e-13,
4.664787284288575e-11,
-1.8731221899846444e-10,
-1.3372978251441443e-15,
1.1459071206275176e-15,
2.0562096763949444e-05,
-1.516838463378194e-06,
-1.1531029193099936e-08,
6.802396982086501e-10
]
217426140991.84854
[
0.0,
-1.0662454434492696e-12,
-2.4921183616382003e-13,
3.952418579833617e-11,
-1.887694879706513e-10,
-1.3377811612829142e-15,
9.9604830192894424e-16,
2.0525280619883478e-05,
-1.5171077343870828e-06,
-1.1531030102594254e-08,
6.802269432702226e-10
]
[
0.0,
-1.6599441754107853e-12,
-2.502029108478628e-13,
4.642468291348256e-11,
-1.89975470225363e-10,
-1.3372956616594607e-15,
1.1483859322225145e-15,
2.056341750808789e-05,
-1.5168273509777578e-06,
-1.1531029284843959e-08,
6.802384147013086e-10
]
217443983144.18597
[
0.0,
-1.6995193231987587e-12,
-2.5984571841892606e-13,
4.513750088674478e-11,
-1.8431555446245960e-10,
-1.3372831844002620e-15,
1.162681746291192e-15,
2.0571034509990575e-05,
-1.5167632634888937e-06,
-1.153102981395053e-08,
6.802310124516392e-10
]
217559331453.50867
[
0.0,
-1.3792221926544971e-12,
-2.3859319111864954e-13,
4.6103489757137796e-11,
-1.9575146354035955e-10,
-1.3378142009564283e-15,
1.0488113543472935e-15,
2.061576613192918e-05,
-1.5194907469119867e-06,
-1.153100729583606e-08,
6.805460429454921e-10
]
217593561314.46277
[
0.0,
-1.3693607741184603e-12,
-2.8477913749585565e-13,
4.264882431775132e-11,
-1.677999891930478e-10,
-1.33783241660518e-15,
1.0453134217011377e-15,
219673960954.7815

2.0617605074802182e-05,
-1.5161777650526904e-06,
-1.1531006523389213e-08,
6.805568495479519e-10
]

Third Iteration





Initial-Population

```
def initial_population():
    first_population = []
    for i in range(POPULATION_SIZE):
        first_population.append(np.copy(first_parent))

    for i in range(POPULATION_SIZE):
        for j in range(VECTOR_SIZE):
            vary = 0
            mutation_prob = random.randint(0, 100)
            if mutation_prob < 60:
                vary = 1 + random.uniform(-0.05*j, 0.05*j)
                rem = overfit_vector[j]*vary

                if abs(rem) < 10:
                    first_population[i][j] = rem
                elif abs(first_population[i][j]) >= 10:
```

```
                first_population[i][j] = random.uniform(-1,1)
    return first_population
```

We used overfit vectors provided to us as the initial population. We applied variation on the vectors (mutation). We changed every bit of every vector with a probability.

Fitness

```
def get_population_fitness(population):
    fit = np.empty((POPULATION_SIZE, 3))

    for i in range(POPULATION_SIZE):
        error = get_errors(SECRET_KEY, list(population[i]))
        fit[i][0] = error[0]
        fit[i][1] = error[1]
        fit[i][2] = abs(error[0]*TRAIN_RATIO + error[1])

    ret = np.column_stack((population, fit))
    ret = ret[np.argsort(ret[:,-1])]
    return ret
```

`get_errors` requests are sent to the server which returns the train error and validation error for every vector in the population.

`Fitness = Train Ratio * Train Error + Validation Error`

Train ratio was tested and selected so as to obtain less Train error, Validation error and better rank on leaderboard. It was tested on following values:

Train factor = 0.7

Idea behind it was to give more weightage to validation error than train error. But on testing, it gave more error. We also tested on lower values but the result wasn't positive. So we tested on greater values than 0.7

Train factor = 1

On testing, train factor=1 gave the best output. Giving equal weightage to both train and validation error.

Simulated-Binary-Crossover

Simulated binary crossover generates tow children from the two parents which follow the following equation while giving us control over the variation by manipulating the distribution index value.

$$\frac{x_1^{\text{new}} + x_2^{\text{new}}}{2} = \frac{x_1 + x_2}{2}$$

A random number is choosed `u` between `[0,1)`. The distribution index is assigned desired value and then The crossover is done by choosing a random number in the range `[0, 1)`. The distribution index is assigned its value and then β is calculated:

$$\beta = \begin{cases} (2u)^{\frac{1}{\eta_c + 1}} & u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c + 1}} & \text{Otherwise} \end{cases}$$

> Distribution index that determines how far children go from parents. The greater its value the closer the children are to parents.

The distribution index is a value between `[2, 5]` and the offsprings are calculated as follows:

$$x_1^{\text{new}} = 0.5\left[(1+\beta)x_1 + (1-\beta)x_2\right]$$
$$x_2^{\text{new}} = 0.5\left[(1-\beta)x_1 + (1+\beta)x_2\right]$$

The code is as shown:

```
def crossover(parent1, parent2):
    child1 = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    child2 = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

    r1 = random.random()
    n = 3

    beta = (2 * r1)**((n + 1)**-1)
    if (r1 >= 0.5):
        beta = ((2*(1-r1))**-1)**((n + 1)**-1)

    p1 = np.array(parent1)
    p2 = np.array(parent2)
    child1 = 0.5*((1 + beta) * p1 + (1 - beta) * p2)
    child2 = 0.5*((1 - beta) * p1 + (1 + beta) * p2)

    return [child1, child2]
```

We varied the **Distributed Index** value depending on the population.

Mutation

```
def mutation(vector):
    for i in range(VECTOR_SIZE):
        mutation_prob = random.randint(0, 100)
        if mutation_prob < 50:
            var = 1 + random.uniform(-0.3, 0.3)
            rem = overfit_vector[i]*var # multiply to create variations
            if abs(rem) <= 10:
                vector[i] = rem
    return vector
```

- Our mutations are probabilistic in nature. For the vector, at every index a mutation is decided to be performed with a probability of `1/2`.
- This was even tested for smaller values and larger, but due to limit of calls we stick to `1/2` probability which gave us decent results.

Hyperparameters

Population size

The `POPULATION_SIZE` parameter is set to **10**. Due to limited api calls it is set so low. Increasing this size will definitely improve the working the code.

Mating pool size

The `MATING_POOL_SIZE` variable is **8**. We sort the parents by the fitness value and choose the top 8 that are selected for the mating pool.

Number of parents passed down to new generation

We set this variable to **8**.

- We kept the value small when we were just starting out and were reliant on more variations in the children to get out of the overfit. We did not want to forcefully bring down more parents as that would waste a considerable size of the new population.
- When we were unsure of how our mutations and crossover were performing or when we would change their parameters, we would increase this variable. We did this so that even if things go wrong, our good vectors are still retained in the new generations.

Distribution index (Crossover point)

This parameter was applied in the `Simulated Binary Crossover`. It determines how far children go from parents. The greater its value the closer the children are to parents. The Distribution Index is changed according to need. If the vectors felt too alike, the value was lowered, otherwise if they seemed too different, it was changed to 5.

Mutation Range

We varied our mutation range drastically throughout the assignment.

- We made the variation as little as between factors of (0.09, 1.01) for when we had to fine tune our vectors. We did this when we were confident we had a good vector and excessive mutations were not helping. So we tried small variations, to get its best features.
- When our vectors would reach a local minima - we would mutate extensively to get out of the minima. The factor for multiplication could vary anywhere from (0.3, 1.7).
- We would even assign random numbers at times to make drastic changes when no improvement was shown by the vectors. We did this initially when we first ran the 0 vector and it helped us achieve good results.

Heuristics

- **Initial vector**: Before we used overfit vector as initial population. After changing this to probabilistic mutation of every bit of overfit vector, we saw some better resuts.
- **Probabilistic mutation**: Earlier we were mutating on one index only. But we changed our code to mutate each index with a probability of `1/2`, this brought more variation in the genes and worked well for our populations.
- **Varying mutations for different indices** : After some days into assignment, we decreased the probabilty of mutation of each bit. This was because the older generation were coming out to be better than newer. So tried to make newer generation not a lot varying from older ones.

Generation_i.json

The trace for the final submitted vectors has all the generations in which they occurred.

The format is as follows,

```
[
    {
        "Generation": $genNumber,
        "Population": [
            $populationVector1,
            ...
        ],
        "Details": [
            {
                "Child Number": $childNumber,
                "Parent 2": $secondParentVector,
                "Parent 1": $firstParentVector,
                "After Crossover": $crossoverOfParents,
                "After Mutation": $mutationOfCrossoverVector
            }
        ]
```

```
    }
]
```

As 8 parents are brought down to new generation, the last 8 children (when sorted by fitness) are not included in the new population for the next generation. s