GENETIC ALGORITHM

TEAM NO: 62
Mallika Subramanian - 2018101041
Tanvi Karandikar - 2018101059

SUMMARY OF GENETIC ALGORITHM:

The genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

The stepwise explanation of our implementation is as follows :

> *STEP 1: GENERATE INITIAL POPULATION*

To start the genetic algorithm, we require a population of some **pop_size** number of individuals. These individuals are generated by mutating the vector (overfit on the training dataset) that is provided to us. This produces the starting generation for the GA.

We have ensured that the first generation of individuals is diverse by keeping the probability of mutation factor high, that is the probability with which a gene is mutated in a chromosome is high. This is so that the algorithm does not converge to a local minima in the very beginning itself

```
# generate the population
for i in range(pop_size):
    temp = np.copy(vector_og)
    population[i] = np.copy(mutateall(temp,0.9,mutate_range))
```

> *STEP 2: OBTAIN ERRORS FOR THIS GENERATION*

Once the first generation is ready, we obtain the errors of all the individuals (parents for the next generation). This is done to determine the fitness of each individual and to decide their progress into the next generation.

```
# generate errors for each individual in the population
for j in range(pop_size):
    # passing a list to the get_errors function
    temp = population[j].tolist()
    err = server.get_errors(key, temp)
    # adding the two errors and storing in parenterror - fitness function
    parenterrors[j] = np.copy((err[0]+err[1]))
    parenterrors1[j] = np.copy((err[0]))
    parenterrors2[j] = np.copy((err[1]))
```

> *STEP 3: CROSSOVER OF PARENT POULATION*

- Once the parent errors have been obtained, the parent population is sorted in the **increasing** order of their *errors*. Then the crossover step takes place.
- Crossovers happpen until `pop_size` children have been produced. The parents for crossover are chosen from a set of top `select_sure` parents in the parent population.
- The parents chosen from the previous step are sent to the crossover function which returns two children (two vectors). The `crossover` function returns **mutated** the child vectors. These 2 child vectors are then appended to the child population.
- If the child vector is identical to the parent vector, it is not included in the child population and that iteration is discarded.

```
child_population = np.zeros((pop_size, MAX_DEG))
new_iter = 0

while(new_iter < pop_size):


    #Select randomly among top k parents  (For now k =10)
    arr = crossover_select2(parenterrors, cross_select_from)

    # Sending parents for crossover
    temp = crossover(population[arr[0]], population[arr[1]],mutate_range,prob_mut_cross)

    #case1 : child vector is identical to parent vector
    if temp[0].tolist() == population[arr[0]].tolist() or temp[1].tolist() == population[arr[0]].tolist() or
temp[0].tolist() == population[arr[1]].tolist() or temp[1].tolist() == population[arr[1]].tolist():
        continue
```

```
    #case2 : append to child population
    child_population[new_iter] = np.copy(temp[0])
    new_iter += 1

    child_population[new_iter] = np.copy(temp[1])
    new_iter += 1
```

STEP 4: OBTAIN ERRORS FOR CHILD POPULATION

The errors for the child population are obtained. The child population is then sorted in the increasing order of their errors (sum of the train and validation errors obtained).

```
    childerrors = np.zeros(pop_size)
    childerrors1 = np.zeros(pop_size)
    childerrors2 = np.zeros(pop_size)

    # generate errors for each child
    for j in range(pop_size):
        temp = child_population[j].tolist()
        err = server.get_errors(key, temp)

        # adding the two errors and storing in parenterror
        childerrors[j] = np.copy((err[0]+err[1]))
        childerrors1[j] = np.copy((err[0]))
        childerrors2[j] = np.copy((err[1]))

    # Sort children
    childinds = np.copy(childerrors.argsort())
    childerrors = np.copy(childerrors[childinds[::1]])
    childerrors1 = np.copy(childerrors1[childinds[::1]])
    childerrors2 = np.copy(childerrors2[childinds[::1]])
    child_population = np.copy(child_population[childinds[::1]])
```

STEP 5: CREAT THE NEXT GENERATION

- Now that we have the **pop_size** parents and **pop_size** children we have to prepare a population of **pop_size** indivduals by selection them from the parents and children.
- The new generation will have top `select_sure` number of parents and children with cerainity. This is so that the best (best fitness and least error) chromosomes of both population are advanced to the next generation. This is stored in a `tempbank`.

```
# now the children are sorted and stored in child and parents are sorted in population
# we will now create a tempbank array to store top k parents, top k childs and rest being sorted taking from the top
tempbankerr = np.zeros(pop_size)
tempbankerr1 = np.zeros(pop_size)
tempbankerr2 = np.zeros(pop_size)
tempbank= np.zeros((pop_size, MAX_DEG))

for j in range(select_sure):

    #choosing the top jth parent and putting in the array
    tempbank[j]=np.copy(population[j])
    tempbankerr[j]=np.copy(parenterrors[j])
    tempbankerr1[j]=np.copy(parenterrors1[j])
    tempbankerr2[j]=np.copy(parenterrors2[j])

    #choosing the top jth child and putting it into the array
    tempbank[j+select_sure]=np.copy(child_population[j])
    tempbankerr[j+select_sure]=np.copy(childerrors[j])
    tempbankerr1[j+select_sure]=np.copy(childerrors1[j])
    tempbankerr2[j+select_sure]=np.copy(childerrors2[j])
```

- The parent and children population is combined into a single `candidates` array and the remaining `pop_size - select_sure` number of individuals are selected.

```
# combining parents and children into one array
candidates = np.copy(np.concatenate([population[select_sure:], child_population[select_sure:]]))
candidate_errors = np.copy(np.concatenate([parenterrors[select_sure:], childerrors[select_sure:]]))
candidate_errors1 = np.copy(np.concatenate([parenterrors1[select_sure:], childerrors1[select_sure:]]))
candidate_errors2 = np.copy(np.concatenate([parenterrors2[select_sure:], childerrors2[select_sure:]]))

# sorting all the candidates by error
candidate_errors_inds = candidate_errors.argsort()
candidate_errors = np.copy(candidate_errors[candidate_errors_inds[::1]])
```

```
candidate_errors1 = np.copy(candidate_errors1[candidate_errors_inds[::1]])
candidate_errors2 = np.copy(candidate_errors2[candidate_errors_inds[::1]])
candidates = np.copy(candidates[candidate_errors_inds[::1]])

# TODO: Select the best popsize - 2*(select_sure)
cand_iter = 0

while(cand_iter + 2*select_sure < pop_size):
    tempbank[cand_iter+2*select_sure] = np.copy(candidates[cand_iter])
    tempbankerr[cand_iter+2*select_sure] = np.copy(candidate_errors[cand_iter])
    tempbankerr1[cand_iter+2*select_sure] = np.copy(candidate_errors1[cand_iter])
    tempbankerr2[cand_iter+2*select_sure] = np.copy(candidate_errors2[cand_iter])
    cand_iter += 1
```

STEP 6: SETTING THE NEW POPULATION

This is now set as the next generation of individuals for the GA. Their errors are computed and the population is sorted.

```
#now setting the next population
population=np.copy(tempbank)
parenterrors=np.copy(tempbankerr)
parenterrors1=np.copy(tempbankerr1)
parenterrors2=np.copy(tempbankerr2)

#we will now sort before updating min_error
parenerrorsinds = parenterrors.argsort()
parenterrors = np.copy(parenterrors[parenerrorsinds[::1]])
parenterrors1 = np.copy(parenterrors1[parenerrorsinds[::1]])
parenterrors2 = np.copy(parenterrors2[parenerrorsinds[::1]])
population = np.copy(population[parenerrorsinds[::1]])
```

STEP 7: CHECK MINIMUM

If the error of the fittest individual is lesser than the current minimum error, the `min_error` and the `to_send` vector are updated.

```
if(min_error == -1 or min_error > parenterrors[0]):
            to_send = np.copy(population[0])
            min_error = np.copy(parenterrors[0])
            min_error1 = np.copy(parenterrors1[0])
            min_error2 = np.copy(parenterrors2[0])
            nochange=0
else:
    print("no improvement!!!")
```

STEP 8: REPEAT STEPS 3 TO 7

The Genetic Algorithm is repeated (Step 3 - Step 7) for `iter` number of iterations (this was done considering the requests that can be made to the server in order to obtain the errors for the different vectors was limited).

DIAGRAMS FOR 3 CONSECUTIVE ITERATIONS

The diagrams representing the 3 iterations of the genetic algorithm are as follows.

Each diagram is accompanied by a table which contains its own set of Population, Children and Mutation vectors which are indexed and shown along with it. The diagram makes use of these indices.

NOTE : The DIAGRAMS were created with the code that was run with the specified sample values only for demonstration purposes and are not indicative of actual values used to submit the final best vector.

Table Coloumns :

- *Col 1:* Represents the indices of the individuals in the population.
- *Col 2:* Represents the errors of the individuals in the population.
- *Col 3:* Represents the probability with which the individuals are selected for crossover based on the errprs (fitness funciton).
- *Col 4:* Represents the parents selected for crossover from the top 8 parents (vectors before cross over).
- *Col 5:* Represents the set of indices at which the genes/elements are swapped between the two chromosomes/parent vectors.
- *Col 6:* Represents the indices of the children produced.
- *Col 1:* Represents the indices of the mutated children.

DIAGRAM 1 :

Index	Population
P0	[[-1.09755638e-19 1.32939389e-01 -6.20801458e+00 4.82661643e-02 3.96228158e-02 8.15629029e-05 -6.01876916e-05 -1.24702571e-07 3.53143519e-08 4.03142598e-11 -6.88874246e-12]]
P1	[[-1.00000000e-19 1.15134807e-01 -6.51215903e+00 4.93390314e-02 3.81084816e-02 8.01307733e-05 -6.40425896e-05 -1.19767609e-07 3.71610253e-08 4.56887165e-11 -6.73242018e-12]]
P2	[[-9.09763536e-20 1.30434319e-01 -6.11840514e+00 4.85948369e-02 4.12593911e-02 8.44665286e-05 -7.03590574e-05 -1.32420351e-07 3.98912788e-08 3.91827360e-11 -6.94435309e-12]]
P3	[[-1.11197351e-19 1.39367615e-01 -6.51215903e+00 4.65916205e-02 3.96728558e-02 8.48181486e-05 -6.28290949e-05 -1.15944053e-07 3.53143519e-08 4.57629945e-11 -6.73242018e-12]]
P4	[[-1.01056370e-19 1.24031745e-01 -5.75612274e+00 5.20613226e-02 3.79131959e-02 8.13236610e-05 -6.39756202e-05 -1.25591436e-07 3.75600932e-08 3.95388711e-11 -6.80035809e-12]]
P5	[[-1.21101805e-19 1.24714464e-01 -5.34825838e+00 5.38364813e-02 3.90654112e-02 8.28230832e-05 -6.27259213e-05 -1.15944053e-07 3.63157478e-08 4.26977467e-11 -6.59115286e-12]]
P6	[[-1.15512390e-19 1.46013266e-01 -6.45580885e+00 4.68807811e-02 3.65696493e-02 8.23872873e-05 -6.55149925e-05 -1.28091673e-07 3.59771743e-08 3.86749851e-11 -6.33812772e-12]]
P7	[[-1.18483103e-19 1.18832809e-01 -5.52816801e+00 5.54092443e-02 3.58695097e-02 7.89244528e-05 -5.84264973e-05 -1.15944053e-07 3.35363809e-08 4.69782245e-11 -6.73242018e-12]]
P8	[[-1.02653775e-19 1.13780657e-01 -5.97033624e+00 4.93390314e-02 4.02372223e-02 7.91155558e-05 -6.59138787e-05 -1.36661306e-07 3.48409638e-08 4.10380479e-11 -6.17514423e-12]]
P9	[[-9.26792843e-20 1.22458875e-01 -6.57984124e+00 4.45370475e-02 3.88810110e-02 8.21298415e-05 -6.31342497e-05 -1.19413996e-07 3.53143519e-08 4.03142598e-11 -7.12333387e-12]]

Index	Children
C0	[[-1.11197351e-19 1.24714464e-01 -6.51215903e+00 5.38364813e-02 3.96728558e-02 8.48181486e-05 -6.27259213e-05 -1.15944053e-07 3.53143519e-08 4.26977467e-11 -6.59115286e-12]]
C1	[[-1.21101805e-19 1.39367615e-01 -5.34825838e+00 4.65916205e-02 3.90654112e-02 8.28230832e-05 -6.28290949e-05 -1.15944053e-07 3.63157478e-08 4.57629945e-11 -6.73242018e-12]]
C2	[[-1.21101805e-19 1.32939389e-01 -5.34825838e+00 5.38364813e-02 3.96228158e-02 8.28230832e-05 -6.01876916e-05 -1.15944053e-07 3.53143519e-08 4.03142598e-11 -6.59115286e-12]]
C3	[[-1.09755638e-19 1.24714464e-01 -6.20801458e+00 4.82661643e-02 3.90654112e-02 8.15629029e-05 -6.27259213e-05 -1.24702571e-07 3.63157478e-08 4.26977467e-11 -6.88874246e-12]]
C4	[[-1.01056370e-19 1.24031745e-01 -6.45580885e+00 5.20613226e-02 3.79131959e-02 8.13236610e-05 -6.39756202e-05 -1.28091673e-07 3.59771743e-08 3.86749851e-11 -6.33812772e-12]]
C5	[[-1.15512390e-19 1.46013266e-01 -5.75612274e+00 4.68807811e-02 3.65696493e-02 8.23872873e-05 -6.55149925e-05 -1.25591436e-07 3.75600932e-08 3.95308711e-11 -6.80035809e-12]]
C6	[[-1.00000000e-19 1.15134807e-01 -6.51215903e+00 5.54092443e-02 3.81084816e-02 8.01307733e-05 -5.84264973e-05 -1.19767609e-07 3.35363809e-08 4.69782245e-11 -6.73242018e-12]]
C7	[[-1.18483103e-19 1.18832809e-01 -5.52816801e+00 4.93390314e-02 3.58695097e-02 7.89244528e-05 -6.40425896e-05 -1.15944053e-07 3.71610253e-08 4.56887165e-11 -6.73242018e-12]]
C8	[[-1.09755638e-19 1.32939389e-01 -6.20801458e+00 4.82661643e-02 3.81884816e-02 8.01307733e-05 -6.40425896e-05 -1.19767609e-07 3.71610253e-08 4.56887165e-11 -6.88874246e-12]]
C9	[[-1.00000000e-19 1.15134807e-01 -6.51215903e+00 4.93390314e-02 3.96228158e-02 8.15629029e-05 -6.01876916e-05 -1.24702571e-07 3.53143519e-08 4.03142598e-11 -6.73242018e-12]]

Index	Mutated Children
M0	[[-1.18184884e-19 1.31551583e-01 -6.51215903e+00 5.08600761e-02 4.25518674e-02 9.23212945e-05 -5.65115663e-05 -1.24418710e-07 3.63721594e-08 4.29548023e-11 -6.69703124e-12]]
M1	[[-1.18001261e-19 1.27015079e-01 -5.78513239e+00 4.75706155e-02 3.90654112e-02 8.26922983e-05 -6.29198690e-05 -1.21039765e-07 3.33023627e-08 4.75387745e-11 -6.73242018e-12]]
M2	[[-1.36146644e-19 1.27989266e-01 -5.79377615e+00 5.89729722e-02 4.16891293e-02 7.93034888e-05 -5.76232010e-05 -1.06842033e-07 3.75675652e-08 4.15504200e-11 -7.05652796e-12]]
M3	[[-1.03574954e-19 1.36935819e-01 -6.20801458e+00 5.05197530e-02 4.12749773e-02 7.46199965e-05 -6.05173787e-05 -1.20932618e-07 3.94583133e-08 3.99738912e-11 -6.95306201e-12]]
M4	[[-1.02746931e-19 1.24031745e-01 -6.59668774e+00 5.15691483e-02 3.92910163e-02 8.44099330e-05 -6.13673681e-05 -1.39155625e-07 3.90378530e-08 3.72990255e-11 -6.96342082e-12]]
M5	[[-1.18916474e-19 1.58107624e-01 -5.75612274e+00 4.27208183e-02 3.65696493e-02 8.23872873e-05 -7.13610745e-05 -1.26932641e-07 3.67375839e-08 3.62262932e-11 -7.42952104e-12]]
M6	[[-9.82588492e-20 1.18736095e-01 -6.29551722e+00 5.65571975e-02 3.96130179e-02 8.01307733e-05 -6.28887961e-05 -1.28224740e-07 3.30702647e-08 5.02505574e-11 -7.30405405e-12]]
M7	[[-1.22126800e-19 1.20981147e-01 -6.02769787e+00 4.93390314e-02 3.26478637e-02 8.29986241e-05 -6.78383949e-05 -1.15509594e-07 3.97312848e-08 4.45176663e-11 -6.20918391e-12]]
M8	[[-1.01406129e-19 1.42666168e-01 -5.94671581e+00 4.82661643e-02 3.58161293e-02 7.99771127e-05 -6.35757789e-05 -1.10253956e-07 3.73624007e-08 4.94084706e-11 -6.87677550e-12]]
M9	[[-9.78507898e-20 1.09069814e-01 -7.06328488e+00 5.16778115e-02 4.11957767e-02 7.68807915e-05 -5.75818661e-05 -1.28549982e-07 3.24504309e-08 3.93914903e-11 -6.45976524e-12]]

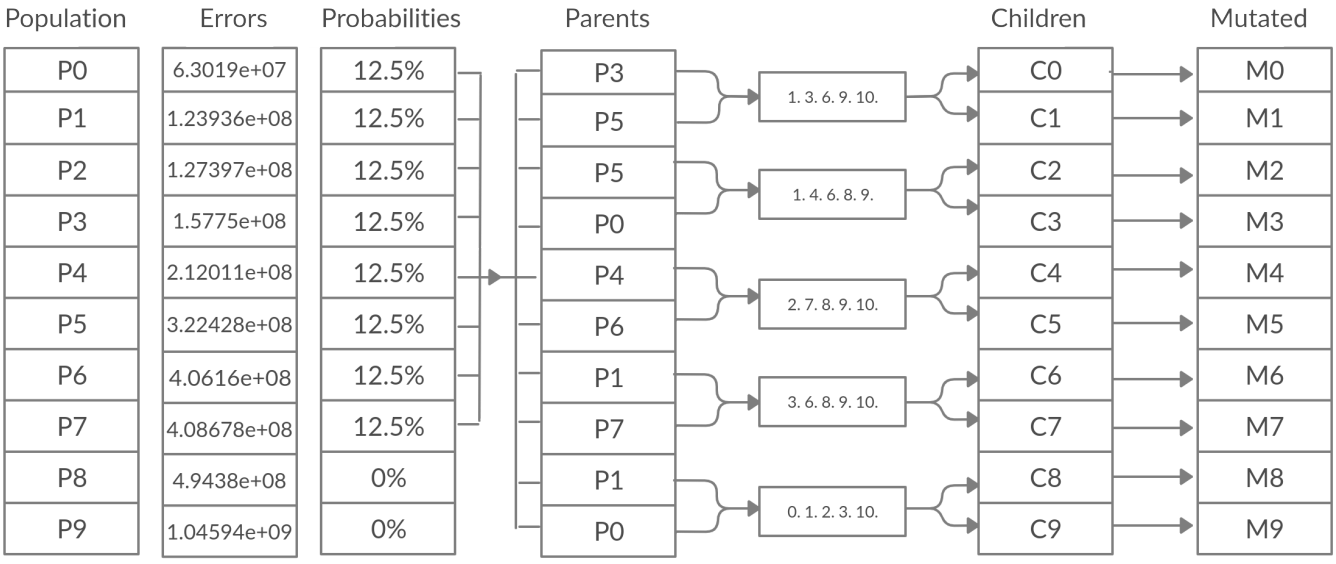


DIAGRAM 2 :

Index	Population
P0	[[-9.78507898e-20 1.09069814e-01 -7.06328488e+00 5.16778115e-02 4.11957767e-02 7.68807915e-05 -5.75818661e-05 -1.28549982e-07 3.24504309e-08 3.93914903e-11 -6.45976524e-12]]
P1	[[-1.09755638e-19 1.32939389e-01 -6.20801458e+00 4.82661643e-02 3.96228158e-02 8.15629029e-05 -6.01876916e-05 -1.24702571e-07 3.53143519e-08 4.03142598e-11 -6.88874246e-12]]
P2	[[-1.00000000e-19 1.15134807e-01 -6.51215903e+00 4.93390314e-02 3.81084816e-02 8.01307733e-05 -6.40425896e-05 -1.19767609e-07 3.71610253e-08 4.56887165e-11 -6.73242018e-12]]
P3	[[-9.09763536e-20 1.30434319e-01 -6.11040514e+00 4.85948369e-02 4.12593911e-02 8.44665286e-05 -7.03590574e-05 -1.32420351e-07 3.98912788e-08 3.91827360e-11 -6.94435309e-12]]
P4	[[-1.11197351e-19 1.39367615e-01 -6.51215903e+00 4.65916205e-02 3.96728558e-02 8.48181486e-05 -6.28290949e-05 -1.15944053e-07 3.53143519e-08 4.57629945e-11 -6.73242018e-12]]
P5	[[-1.01406129e-19 1.42666168e-01 -5.94671581e+00 4.82661643e-02 3.58161293e-02 7.99771127e-05 -6.35757709e-05 -1.10253956e-07 3.73624007e-08 4.94084706e-11 -6.87677550e-12]]
P6	[[-1.01056370e-19 1.24031745e-01 -5.75612274e+00 5.20613226e-02 3.79131959e-02 8.13236610e-05 -6.39756202e-05 -1.25591436e-07 3.75600932e-08 3.95308711e-11 -6.80035809e-12]]
P7	[[-1.21101805e-19 1.24714464e-01 -5.34825838e+00 5.38364813e-02 3.90654112e-02 8.28230832e-05 -6.27259213e-05 -1.15944053e-07 3.63157478e-08 4.26977467e-11 -6.59115286e-12]]
P8	[[-1.15512390e-19 1.46013266e-01 -6.45580885e+00 4.68807811e-02 3.65696493e-02 8.23872873e-05 -6.55149925e-05 -1.28091673e-07 3.59771743e-08 3.86749851e-11 -6.33812772e-12]]
P9	[[-1.22126800e-19 1.20981147e-01 -6.02769787e+00 4.93390314e-02 3.26478637e-02 8.29986241e-05 -6.78383949e-05 -1.15509594e-07 3.97312848e-08 4.45176663e-11 -6.20918391e-12]]

Index	Children
C0	[[-1.00000000e-19 1.15134807e-01 -6.51215903e+00 5.16778115e-02 4.11957767e-02 8.01307733e-05 -5.75818661e-05 -1.28549982e-07 3.71610253e-08 4.56887165e-11 -6.45976524e-12]]
C1	[[-9.78507898e-20 1.09069814e-01 -7.06328488e+00 4.93390314e-02 3.81084816e-02 7.68807915e-05 -6.40425896e-05 -1.19767609e-07 3.24504309e-08 3.93914903e-11 -6.73242018e-12]]
C2	[[-1.00000000e-19 1.15134807e-01 -6.20801458e+00 4.82661643e-02 3.81084816e-02 8.01307733e-05 -6.01876916e-05 -1.19767609e-07 3.53143519e-08 4.03142598e-11 -6.73242018e-12]]
C3	[[-1.09755638e-19 1.32939389e-01 -6.51215903e+00 4.93390314e-02 3.96228158e-02 8.15629029e-05 -6.40425896e-05 -1.24702571e-07 3.71610253e-08 4.56887165e-11 -6.88874246e-12]]
C4	[[-1.01056370e-19 1.24031745e-01 -5.94671581e+00 5.20613226e-02 3.79131959e-02 8.13236610e-05 -6.35757709e-05 -1.25591436e-07 3.73624007e-08 4.94084706e-11 -6.87677550e-12]]
C5	[[-1.01406129e-19 1.42666168e-01 -5.75612274e+00 4.82661643e-02 3.58161293e-02 7.99771127e-05 -6.39756202e-05 -1.10253956e-07 3.75600932e-08 3.95308711e-11 -6.80035809e-12]]
C6	[[-9.09763536e-20 1.15134807e-01 -6.11040514e+00 4.85948369e-02 4.12593911e-02 8.44665286e-05 -7.03590574e-05 -1.19767609e-07 3.71610253e-08 4.56887165e-11 -6.73242018e-12]]
C7	[[-1.00000000e-19 1.30434319e-01 -6.51215903e+00 4.93390314e-02 3.81084816e-02 8.01307733e-05 -6.40425896e-05 -1.32420351e-07 3.98912788e-08 3.91827360e-11 -6.94435309e-12]]
C8	[[-1.01056370e-19 1.24031745e-01 -5.75612274e+00 4.82661643e-02 3.96228158e-02 8.15629029e-05 -6.39756202e-05 -1.25591436e-07 3.53143519e-08 3.95308711e-11 -6.88874246e-12]]
C9	[[-1.09755638e-19 1.32939389e-01 -6.20801458e+00 5.20613226e-02 3.79131959e-02 8.13236610e-05 -6.01876916e-05 -1.24702571e-07 3.75600932e-08 4.03142598e-11 -6.80035809e-12]]

Index	Mutated Children
M0	[[-1.00000000e-19 1.16839539e-01 -6.87187217e+00 4.87470801e-02 4.06458032e-02 7.25460868e-05 -5.62900331e-05 -1.35906973e-07 3.90172858e-08 4.97953020e-11 -6.32179141e-12]]
M1	[[-9.00600792e-20 1.18662726e-01 -7.28366196e+00 4.65243360e-02 3.52010465e-02 7.90159838e-05 -6.58705577e-05 -1.19767609e-07 3.24504309e-08 3.60170859e-11 -6.51170179e-12]]
M2	[[-9.96142407e-20 1.24836224e-01 -6.20801458e+00 4.72689105e-02 3.62997469e-02 8.09591595e-05 -6.29094820e-05 -1.25063040e-07 3.75600687e-08 4.22529705e-11 -6.18839390e-12]]
M3	[[-1.09755638e-19 1.33139569e-01 -6.86775771e+00 4.93390314e-02 4.10902168e-02 8.56755139e-05 -6.63740513e-05 -1.36369938e-07 3.71610253e-08 4.37197698e-11 -6.97362532e-12]]
M4	[[-1.05733669e-19 1.21844012e-01 -5.94577887e+00 5.39183051e-02 3.49278297e-02 7.38793989e-05 -6.27355118e-05 -1.18171226e-07 4.08378896e-08 4.94788675e-11 -7.45968805e-12]]
M5	[[-1.00952071e-19 1.50391418e-01 -5.54049265e+00 5.01334842e-02 3.24206554e-02 8.25379014e-05 -5.79245864e-05 -1.17317886e-07 3.96128373e-08 4.25484384e-11 -6.66558904e-12]]
M6	[[-9.10281687e-20 1.08302016e-01 -5.86203961e+00 4.85948369e-02 4.13047662e-02 8.44665286e-05 -6.42779931e-05 -1.12589425e-07 3.40970018e-08 4.73658337e-11 -7.39839547e-12]]
M7	[[-9.33611038e-20 1.42976212e-01 -6.75552597e+00 4.79805764e-02 4.16848240e-02 8.35946938e-05 -6.24857920e-05 -1.26582732e-07 3.66065388e-08 3.98921772e-11 -7.40371252e-12]]
M8	[[-1.01126567e-19 1.18066815e-01 -5.85349651e+00 4.82661643e-02 3.91943462e-02 8.15629029e-05 -6.99740143e-05 -1.35009120e-07 3.31041462e-08 3.92820748e-11 -7.38357836e-12]]
M9	[[-1.11358315e-19 1.32939389e-01 -5.91298103e+00 5.63605247e-02 3.96167895e-02 8.19466141e-05 -6.01876916e-05 -1.22758996e-07 3.61597953e-08 4.31212228e-11 -6.74089609e-12]]

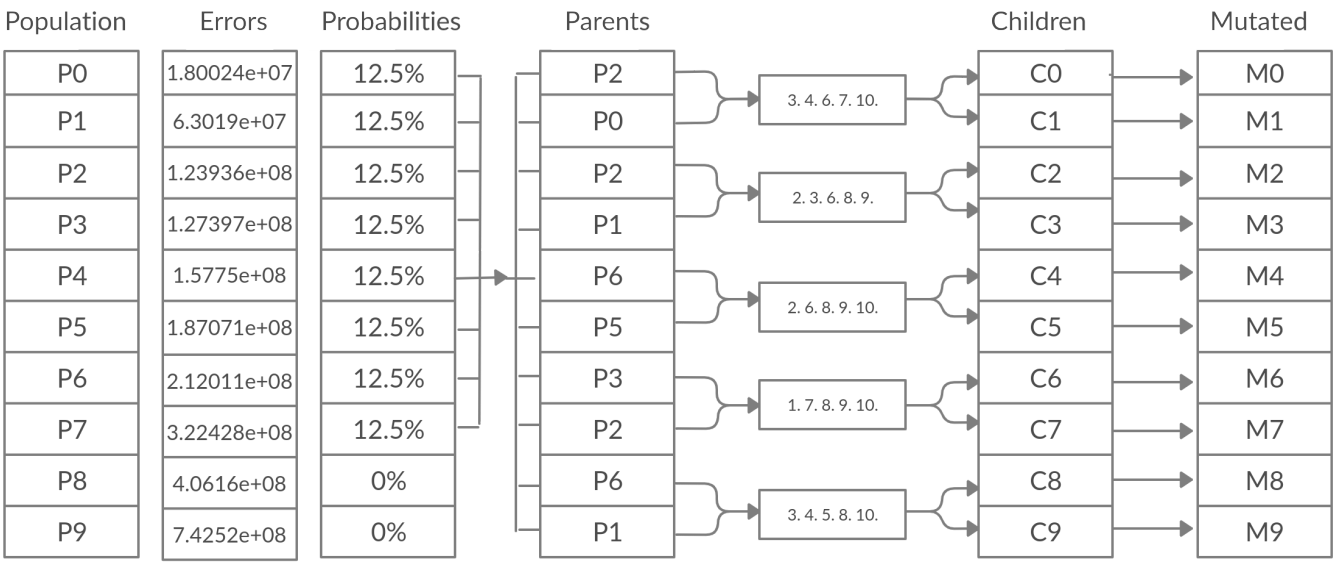
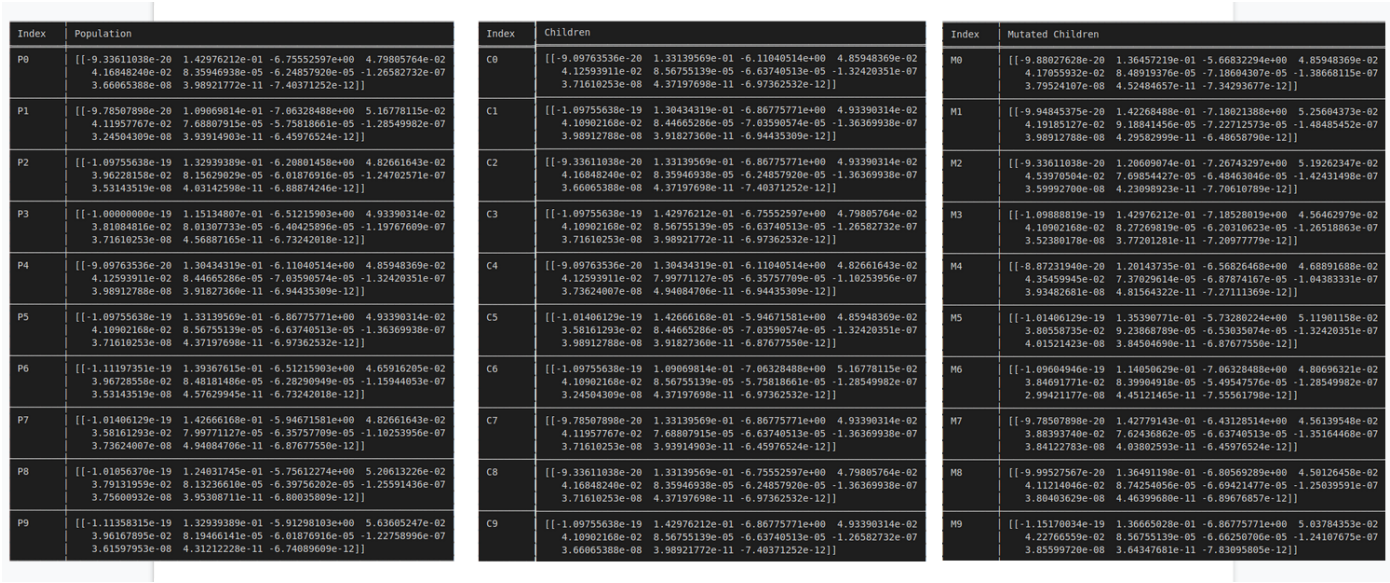


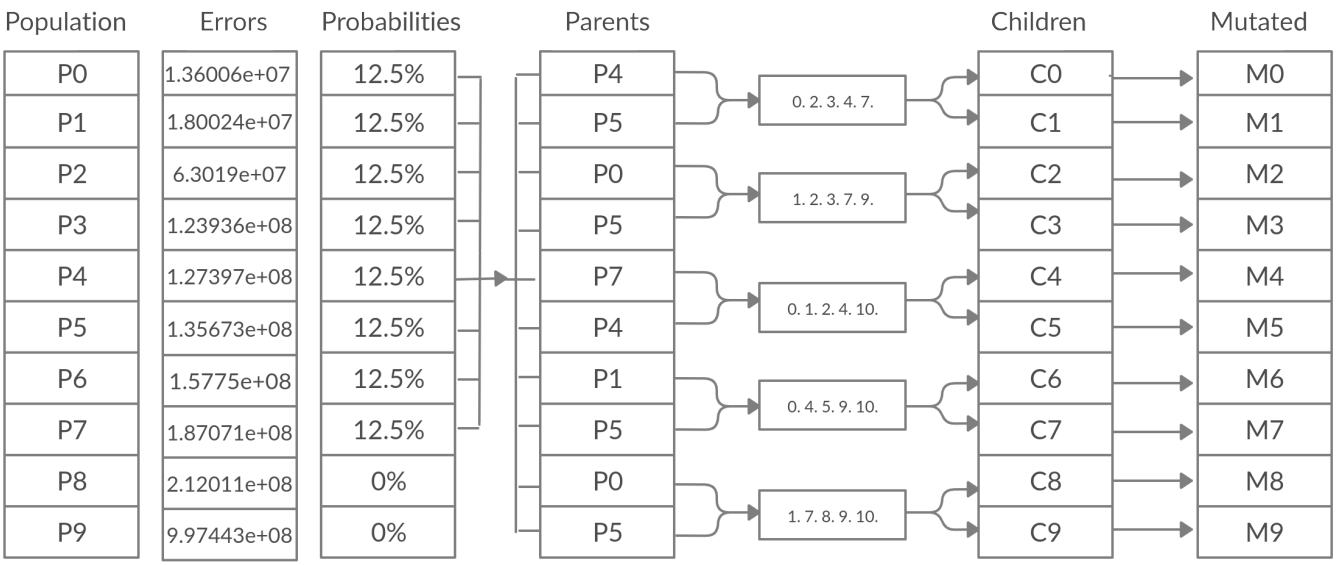
DIAGRAM 3 :



Index	Population
P0	[[-9.33611038e-20 1.42976212e-01 -6.75552597e+00 4.79805764e-02 4.16848240e-02 8.35946938e-05 -6.24857920e-05 -1.26582732e-07 3.66065388e-08 3.98921772e-11 -7.40371252e-12]]
P1	[[-9.78507898e-20 1.09069814e-01 -7.06328488e+00 5.16778115e-02 4.11957767e-02 7.68807915e-05 -5.75818661e-05 -1.28549982e-07 3.24504309e-08 3.93914903e-11 -6.45976524e-12]]
P2	[[-1.09755638e-19 1.32939389e-01 -6.20801458e+00 4.82661643e-02 3.96228158e-02 8.15629029e-05 -6.01876916e-05 -1.24702571e-07 3.53143519e-08 4.03142598e-11 -6.88874246e-12]]
P3	[[-1.00000000e-19 1.15134807e-01 -6.51215903e+00 4.93390314e-02 3.81004816e-02 8.01307733e-05 -6.40425896e-05 -1.19767609e-07 3.71610253e-08 4.56887165e-11 -6.73242818e-12]]
P4	[[-9.09763536e-20 1.30434319e-01 -6.11040514e+00 4.85948369e-02 4.12593911e-02 8.44665286e-05 -7.03590574e-05 -1.32420351e-07 3.98912788e-08 3.91827360e-11 -6.94435309e-12]]
P5	[[-1.09755638e-19 1.33139569e-01 -6.86775771e+00 4.93390314e-02 4.10902168e-02 8.56755139e-05 -6.63740513e-05 -1.36369938e-07 3.71610253e-08 4.37197698e-11 -6.97362532e-12]]
P6	[[-1.11197351e-19 1.39367615e-01 -6.51215903e+00 4.65916205e-02 3.96728558e-02 8.48181486e-05 -6.28290949e-05 -1.15944053e-07 3.53143519e-08 4.57629945e-11 -6.73242818e-12]]
P7	[[-1.01406129e-19 1.42666168e-01 -5.94671581e+00 4.82661643e-02 3.58161293e-02 7.99771127e-05 -6.35757709e-05 -1.10253956e-07 3.73624007e-08 4.94084706e-11 -6.87677550e-12]]
P8	[[-1.01056370e-19 1.24031745e-01 -5.75612274e+00 5.20613226e-02 3.79131959e-02 8.13236610e-05 -6.39756202e-05 -1.25591436e-07 3.75600932e-08 3.95308711e-11 -6.88035809e-12]]
P9	[[-1.11358315e-19 1.32939389e-01 -5.91298103e+00 5.63605247e-02 3.96167895e-02 8.19466141e-05 -6.01876916e-05 -1.22758996e-07 3.61597953e-08 4.31212228e-11 -6.74089609e-12]]

Index	Children
C0	[[-9.09763536e-20 1.33139569e-01 -6.11040514e+00 4.85948369e-02 4.12593911e-02 8.56755139e-05 -6.63740513e-05 -1.32420351e-07 3.71610253e-08 4.37197698e-11 -6.97362532e-12]]
C1	[[-1.09755638e-19 1.30434319e-01 -6.86775771e+00 4.93390314e-02 4.10902168e-02 8.44665286e-05 -7.03590574e-05 -1.36369938e-07 3.98912788e-08 3.91827360e-11 -6.94435309e-12]]
C2	[[-9.33611038e-20 1.33139569e-01 -6.86775771e+00 4.93390314e-02 4.16848240e-02 8.35946938e-05 -6.24857920e-05 -1.36369938e-07 3.66065388e-08 4.37197698e-11 -7.40371252e-12]]
C3	[[-1.09755638e-19 1.42976212e-01 -6.75552597e+00 4.79805764e-02 4.10902168e-02 8.56755139e-05 -6.63740513e-05 -1.26582732e-07 3.71610253e-08 3.98921772e-11 -6.97362532e-12]]
C4	[[-9.09763536e-20 1.30434319e-01 -6.11040514e+00 4.82661643e-02 4.12593911e-02 7.99771127e-05 -6.35757709e-05 -1.10253956e-07 3.73624007e-08 4.94084706e-11 -6.94435309e-12]]
C5	[[-1.01406129e-19 1.42666168e-01 -5.94671581e+00 4.85948369e-02 3.58161293e-02 8.44665286e-05 -7.03590574e-05 -1.32420351e-07 3.98912788e-08 3.91827360e-11 -6.87677550e-12]]
C6	[[-1.09755638e-19 1.09069814e-01 -7.06328488e+00 5.16778115e-02 4.10902168e-02 8.56755139e-05 -5.75818661e-05 -1.28549982e-07 3.24504309e-08 4.37197698e-11 -6.97362532e-12]]
C7	[[-9.78507898e-20 1.33139569e-01 -6.86775771e+00 4.93390314e-02 4.11957767e-02 7.68807915e-05 -6.63740513e-05 -1.36369938e-07 3.71610253e-08 3.93914903e-11 -6.45976524e-12]]
C8	[[-9.33611038e-20 1.33139569e-01 -6.75552597e+00 4.79805764e-02 4.16848240e-02 8.35946938e-05 -6.24857920e-05 -1.36369938e-07 3.71610253e-08 4.37197698e-11 -6.97362532e-12]]
C9	[[-1.09755638e-19 1.42976212e-01 -6.86775771e+00 4.93390314e-02 4.10902168e-02 8.56755139e-05 -6.63740513e-05 -1.26582732e-07 3.66065388e-08 3.98921772e-11 -7.40371252e-12]]

Index	Mutated Children
M0	[[-9.88027628e-20 1.36457219e-01 -5.66832294e+00 4.85948369e-02 4.17055932e-02 8.48919376e-05 -7.18604307e-05 -1.38668115e-07 3.79524107e-08 4.52484657e-11 -7.34293677e-12]]
M1	[[-9.94845375e-20 1.42268488e-01 -7.18021388e+00 5.25604373e-02 4.19185127e-02 9.18841456e-05 -7.22712573e-05 -1.48485452e-07 3.98912788e-08 4.29582999e-11 -6.48658790e-12]]
M2	[[-9.33611038e-20 1.20609074e-01 -7.26743297e+00 5.19262347e-02 4.53970504e-02 7.69854427e-05 -6.48463046e-05 -1.42431498e-07 3.59992700e-08 4.23098923e-11 -7.70610789e-12]]
M3	[[-1.09888819e-19 1.42976212e-01 -7.18528019e+00 4.56462979e-02 4.10902168e-02 8.27269819e-05 -6.20310623e-05 -1.26518863e-07 3.52380178e-08 3.77201281e-11 -7.20977779e-12]]
M4	[[-8.87231940e-20 1.20143735e-01 -6.56826468e+00 4.68891688e-02 4.35459945e-02 7.37029614e-05 -6.87874167e-05 -1.04383331e-07 3.93482681e-08 4.81564322e-11 -7.27111369e-12]]
M5	[[-1.01406129e-19 1.35390771e-01 -5.73280224e+00 5.11901158e-02 3.80558735e-02 9.23868789e-05 -6.53035074e-05 -1.32420351e-07 4.01521423e-08 3.84504690e-11 -6.87677550e-12]]
M6	[[-1.09604946e-19 1.14050629e-01 -7.06328488e+00 4.80696321e-02 3.84691771e-02 8.39904918e-05 -5.49547576e-05 -1.28549982e-07 2.99421177e-08 4.45121465e-11 -7.55561798e-12]]
M7	[[-9.78507898e-20 1.42779143e-01 -6.43128514e+00 4.56139548e-02 3.88393740e-02 7.62436862e-05 -6.63740513e-05 -1.35164468e-07 3.84122783e-08 4.03802593e-11 -6.45976524e-12]]
M8	[[-9.99527567e-20 1.36491198e-01 -6.80569289e+00 4.50126458e-02 4.11214046e-02 8.74254056e-05 -6.69421477e-05 -1.25039591e-07 3.80403629e-08 4.46399680e-11 -6.89676857e-12]]
M9	[[-1.15170034e-19 1.36665028e-01 -6.86775771e+00 5.03784353e-02 4.22766559e-02 8.56755139e-05 -6.66250706e-05 -1.24107675e-07 3.85599720e-08 3.64347681e-11 -7.83095805e-12]]



FITNESS FUNCTION

The fitness funciton is one that is used to decide whether an indivdual from a population will advance to the next generation or not. Greater the fitness, better is the individual.
However, here we have been provided with the *train and validation errors* of the individual vectors.

The relation between fitness and error is : **Lower is the error => Better is the fitness**

The fitness function we have used while coding the Genetic Algorihtm is :

```
err = err1 + err2
```

We have not included weights in our fitness function since otherwise the function would be biased to a particular error. Here, both training and validation error are equally important and hence it is hard to say which is more fit.

CROSSOVER FUNCTION

In the GA, crossover is a function that is performed on two parents to produce offsprings. There are two parts in the crossover function :

1. Select the parents for crossover
2. Perform the crossover

In our implementation, we make our selection for the crossover from the top `cross_select` number of parents, where the parents are sorted in increasing order of their *errors*

As for the actual crossover function -

It first generates a random list of `crossover_no` numbers. The chromosoes (elements) at these indices are swapped between the two parent vectors resulting in the formation of two child vectors.

The child vectors are then mutated and returned from the function.

```
def crossover(vector1, vector2, mutate_range,prob_mut_cross, index=-1):
    send1 = vector1.tolist()
    send2 = vector2.tolist()

    #crossover_no is the 2nd argument, here it is 5
    a = np.random.choice(np.arange(0, 11), 5, replace=False)

    for i in a.tolist():
        send1[i] = np.copy(vector2[i])
        send2[i] = np.copy(vector1[i])

    return mutateall(send1,prob_mut_cross,mutate_range), mutateall(send2,prob_mut_cross,mutate_range)
```

MUTATIONS

Our mutation function takes 3 parameters

- `temp` : This is the vector that is to be mutated.
- `prob` : The probability with which a particular gene in a chromosome will be mutated. As the iterations of the GA proceed, we increase the probability of mutation by 0.01 (every 6 iterations). This is also helpful in ensuring that the algorithm does not converge to a local minima. In case it is approaching a local minima, then the increased probability of mutation helps in bringing **diversity** in the population.

- `mutate_range` : The range within which the element will be mutated. This is a crucial parameter. Since the given overfit vector is very sensitive and if the genes are changed by a relatively large amount (changing the order of the element itself) then this could reflect in the train and validation errors in a negative way. Further, we have implemented **simulated annealing** here wherein we start with a `mutate_range` of 0.1 and then gradually decrease it over the GA iterations (by 0.01 every 6 iterations)

We have used mutations in 2 places :

1. To produce the first generation (initial population) for the genetic algorithm : Here we pass a high initial `prob` of mutation so that a the degree of mutaiton is high and the first generation of population is diverse enough.
2. On child vectors after crossover : Here the child vectors are mutated based on the parameters that have been passed to the function.

HYPERPARAMETERS

```
team_name = "team_62"  # for our reference
MAX_DEG = 11  # number of features
key = '847FWwSwTAxKTPvfixjzNnxbeudiTzJ9psoVIdUxqehtQ5efNo'
ranger = 10
pc = 0.2
pop_size = 30
select_sure = 5
cross_select_from = 10
crossover_no = 3
iter = 40
mutate_range=0.1
prob_mut_cross = 0.7
```

- `pop_size` : Initially we started with a population size of 100. As we progressed in the assignment, we were able to better our GA and realised that a pool size of **30** suited the best. This also gave us room to run more iterations and not exceeding the number of requests to the server per day.
- `select_sure` : In order to ensure that the individuals with the best genes are not lost in the future generations we made sure that we select the top few parents and children for sure (sorted in the asc order of errors). While experimenting with this value we found that on choosing a very high `select_sure` value - there was no diversity in the future generations. All the points were clustered together and the algorithm converged to a local minima. On choosing a very low `select_sure` the algorithm performed poorly. Hence after multiple tries, we fixed a `select_sure` of **5**
- `cross_select_from` : This parameter selects the top few parents to send to the `crossover` function. If this is very low, then the options to choose the parents for crossover are restricted. Likewise a very high `cross_select_from` leads to too much randomness. We stuck to a value of **10**
- `crossover_no` : This number indicates the number of indices at which the elements will be swapped between the two parents in the `crossover` function. We have chosen a value of **5** to ensure that there is a sufficient degree of crossover that can help in the GA. We also tried randomly generating a number between 0-5 but that did not help.
- `mutate_range` : We have set this parameter to **0.1**. The overfit vector is sensitive and if the mutation is drastic it will lead to a high error. Hence we made sure that the elements of the vector undergoing mutation change by this formula :

```
fact=random.uniform(-mutate_range, mutate_range)
vector[i] = np.random.choice([vector[i]*(fact+1), vector[i]], p=[prob,1-prob])
```

With simulated annealing, this range *decreases by 0.01* every 6 iterations and `prob_mut_cross` increases by 0.01.

- `prob_mut_cross` : This parameter is set to **0.7** to start with a large degree of mutations. This will ensure diversity and prevent converging to a local minima. Further the `prob_mut_cross` increases every 6 iterations.

STATISTICAL INFORMATION :

Below is a table that contains statistical inferences and details that we have derived whilst performing the Genetic Algorithm.

The table consists of outputs of many different *runs* of the algorithm. The parameters of each run have beem specified as well.

pop size	iter	cross select from	select sure	prob mut cross	mutate range	crossover no	file	sim. ann.	train error	validation error	comments
10	18	2	3	0.5	0.1	5	02_04_5	yes, 0.01	646685.2314547407	1451230.3945630102	error func is train + 1.5*val till 10th it

pop size	iter	cross select from	select sure	prob mut cross	mutate range	crossover no	file	sim. ann.	train error	validation error	comments
10	40	2	3	0.5	0.1	5	02_04_4	yes, 0.01	818778.2585458648	1411297.2916152105	error func is train + 1.5*val till 10th it
10	40	2	3	0.5	0.1	5	02_04_3	yes, 0.01	2288031.2945214207	655413.9543223411	error func is train + 1.5*val
10	40	2	3	0.5	0.1	5	02_04_2	yes, 0.01	1611429.8575185223	1634666.6757553013	error func is train + 1.5*val
10	40	2	3	0.5	0.1	5	02_04_1	yes, 0.01	443878.6561975613	3371648.054347555	error func is train + 1.5*val
10	18	2	3	0.5	0.1	5	0104_4	yes, 0.01	3617324.8067470654	3630659.506563143	error func is train + 1.5*val
30	39	13	6	0.5	0.1	5	0104_3	yes, 0.01	803726.2793879907	1513600.5745310718	error func is train + 1.5*val
30	39	10	5	0.5	0.1	5	0104_2	yes, 0.01	2031799.9012306575	634542.5036451207	error func is train + 1.5*val
30	39	7	5	0.7	0.1	5	0104_1	yes, 0.01	1566576.511383684	1077161.2608285465	error func is train + 1.5*val
30	39	10	5	0.7	0.1	5	0104	yes, 0.01	679575.7006452213	1923291.54914161	error func is train + 1.5*val
30	40	10	5	0.7	0.1	random	3103_2	yes, 0.01	1215576.5333894524	2444060.231379668	
30	40	10	5	0.7	0.1	random	3103_1	yes, 0.01	1012256.7349054145	1797902.3090917815	
30	40	10	5	0.7	0.1	3	3103	yes, 0.01	572523.343610025	3327798.9412914915	
30	40	10	5	0.7	0.1	5	3003_3	yes, 0.01	1527808.4147913724	497429.4173338076	
30	40	10	5	0.7	0.1	5	3003_2	yes, 0.01	1485648.3782189102	1164865.0420789355	
30	40	7	5	0.7	0.1	5	3003_1	yes, 0.01	665415.2265824392	1687931.0229168932	parents mutating with 0.9
30	40	7	5	0.7	0.1	5	3003	yes, 0.01	1944743.8014151566	782550.4807442231	parents mutating with 0.9
30	40	7	5	0.7	0.1	5	2903_3	yes, 0.015	912177.449795386	1806047.647304091	
30	40	7	5	0.7	0.1	5	2903_2	yes, 0.02	1998806.5615125368	952296.2811102594	repeat run
30	40	7	5	0.7	0.1	5	2903_1	yes, 0.02	951678.2408062371	1734695.3278837525	
30	40	7	5	0.7	0.1	5	2903	yes, 0.01	660381.9279306813	2192867.2831713646	
30	39	13	3	0.7	0.1	5	2803_3	yes, 0.01	742833.1563856753	843956.1519189278	

pop size	iter	cross select from	select sure	prob mut cross	mutate range	crossover no	file	sim. ann.	train error	validation error	comments
30	39	13	3	0.7	0.1	5	2803_2	yes, 0.01	1920056.1907173607	1183183.5002919834	
30	39	13	3	0.7	0.1	5	2803_1	yes, 0.01	1765034.6475311567	620777.6981618816	
30	39	13	3	0.7	0.1	5	2803	yes, 0.01	2514326.850566617	4521127.444235682	
30	20	13	3	0.7	0.1	5	2703_3	no	528858.6935751587	2301976.8057521987	
50	30	10	5	0.7	0.1	5	2703_2	no	918456.6020391921	1417566.1347763892	repeat run
50	30	10	5	0.7	0.1	5	2703_1	no	1982854.5607651887	1108493.5319184358	
30	30	10	5	0.7	0.1	5	2703	no	859024.1161797692	849670.8717230024	
50	30	10	5	0.5	0.7	5	2603_7	no	6317965.3754429165	24329450.087357055	stop if no improvement 10 times
30	30	10	5	0.5	0.1	5	2603_8	no	1708927.894908347	891858.3512177946	stop if no improvement 10 times
30	30	10	5	0.5	random(0,1)	5	2603_9	no	2963905.4910056787	4259197.040362741	stop if no improvement 10 times
30	20	10	5	0.5	random(0,1)	5	2603_10	no	5225454.083665686	9832884.087375896	stop if no improvement 10 times

HEURISTICS APPLIED:

While constructing the Genetic Algorithm, the heuristics that we applied include :

1. *Using a fitness function with weights* : We ran the algorithm with a fitness function as follows : `err = err[0] + 1.5 * err[1]`. This function however did not seem to help our algorithm to converge to the global minima. The train and validation errors we received were still quite high.
2. *Modifying the fitness function mid-way* : To try and imporve the algorithm, we also tried changing the fitness function midway. For first `k` iterations we had a fitness function of `err = err[0] + 1.5*err[1]`. After `k` iterations, we changed the fitness function to simply `err = err[0] + err[1]`. We did so considering that with a 'not so random' population in the later iterations of the GA, we can see that both training nad validation errors are equally important and one cannot overpower the other in deciding the fitness of an individual. However, this approach did not help in reducing the errors either.
3. *Simulated Annealing* : In this method, we reduce the range within which a particular gene/element of the chromosome/vector can be mutated. We start off with a `mutate_range` of **0.1**. After every 6 iterations, this is **decreased** by **0.01**. As a consequence of the decreased `mutate_range` the vectors may now come very close to each other. To prevent the algorithm from converging to a local minima, we started off with a `prob_mut` of **0.7** and **increased** by **0.01** every 6 iterations. This method helped us in the process of achieving the global minima.

TRACE FOR FIRST 10 ITERATIONS

The following tables represent the *Trace* of the first 10 iterations

- *Table 1* : This table shows the following :
  - COL 1: Indices of the intiial population (P0,P1,P2,P3...)
  - COL 1: Individuals of the populations (Vectors)
  - COL 1: Errors corresponding to each individual in the population
- *Table 2* : This table shows the following :
  - COL 1: Indices of the children population produced (C0,C1,C2,C3...)
  - COL 2: The Vectors selected for crossover (That is the parents from which the children have been produced)
  - COL 3: The indices at which the parent vector genes are swapped.
  - COL 4: The child vectors produced after applying the crossover
- *Table 3* : This table shows the following :
  - COL 1: Indices of the mutated children (M0,M1,M2, M3...)
  - COL 2: Mutated child vectors after applying mutation
  - COL 3: Errors corresponding to each mutated child

The parameters for this trace are :

- `pop_size`: 10
- `iter`: 15
- `cross_select_from`: 8
- `select_sure`: 3
- `prob_mut_cross`: 0.9
- `mutate_range`: 0.1

TRACE HAS BEEN ATTACHED IN THE ZIP FOLDER AND CAN BE FOUND -- `trace.txt` *NOTE : The TRACE was run with the specified sample values only for demonstration purposes and are not indicative of actual values used to submit the final best vector.*