Project
for
Casimir Programming Course

By

Neda .Hesam Mahmoudi Nezhad

8/11/2019

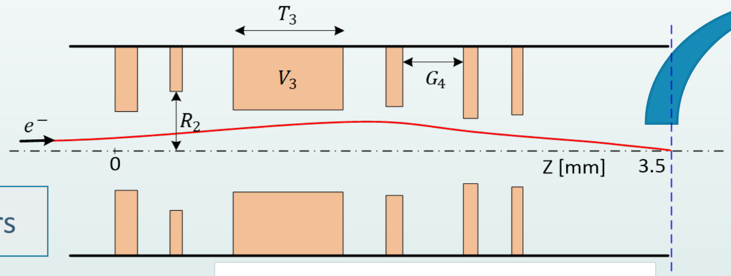
TUDelft- Applied Physics

Imaging Physics

Charged Particle Optics

TUDelft

Electrostatic Lens Optimization Using Genetic Algorithms



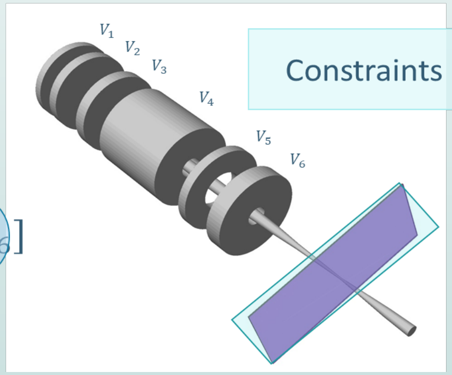
Free parameters

$$T = [T_1\ T_2\ T_3\ T_4\ T_5\ T_6]$$

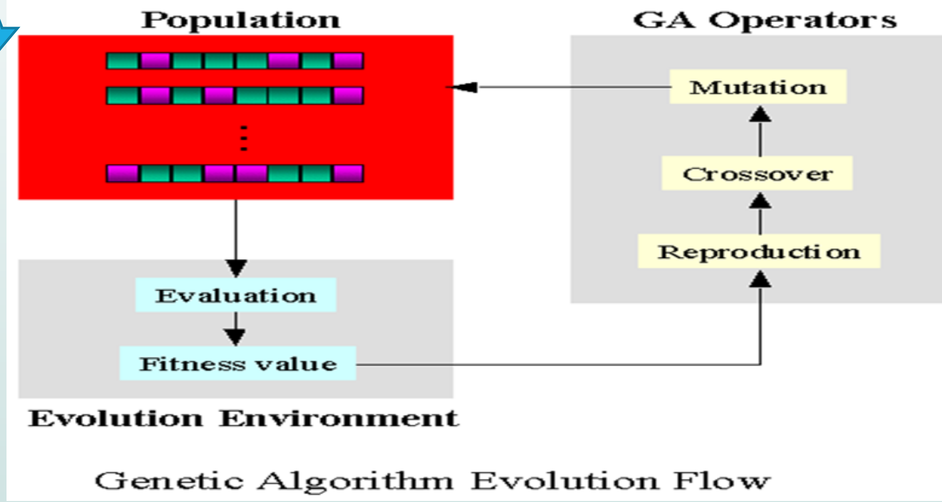
$$G = [G_1\ G_2\ G_3\ G_4\ G_5]$$

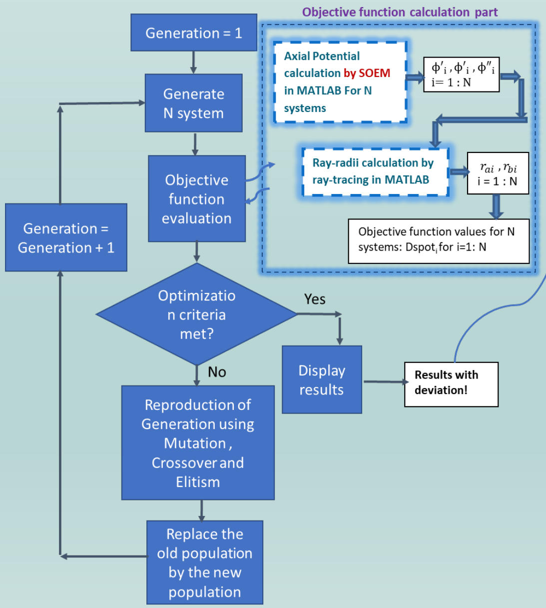
$$R = [R_1\ R_2\ R_3\ R_4\ R_5\ R_6]$$

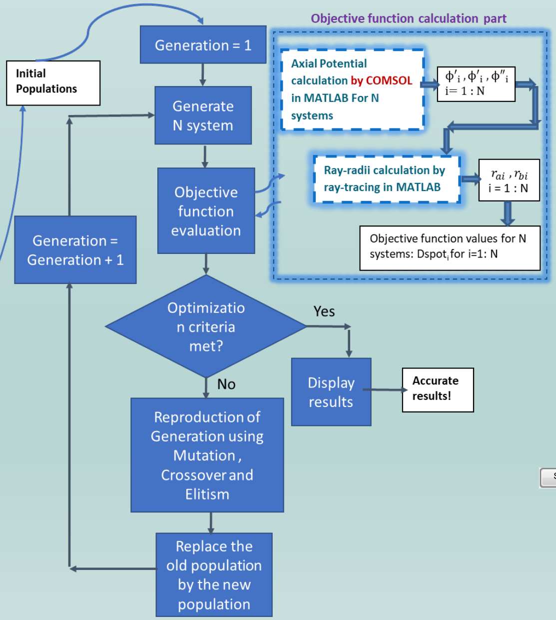
$$V = [V_1\ V_2\ V_3\ V_4\ V_5\ V_6]$$

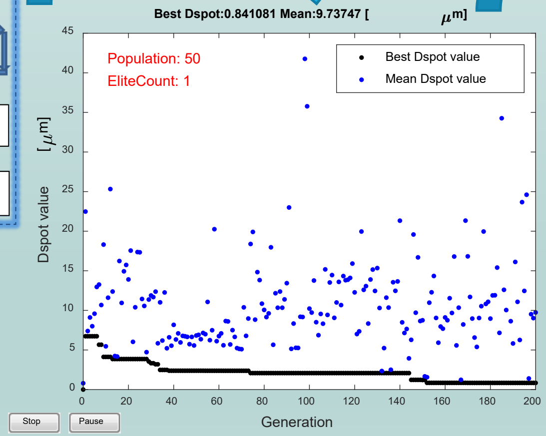


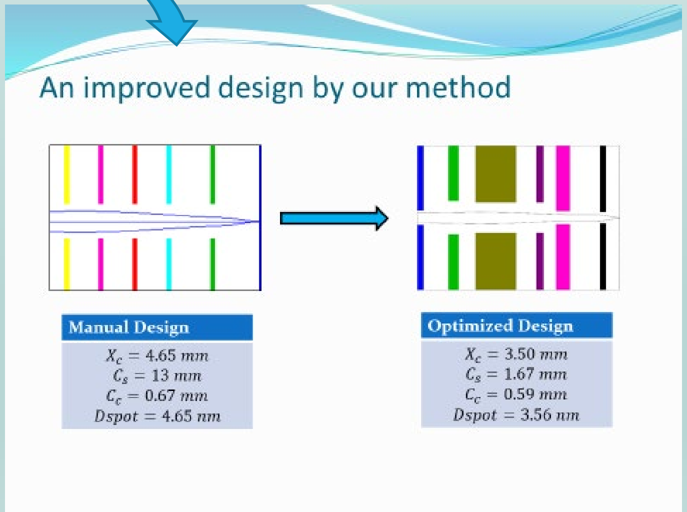
Constraints











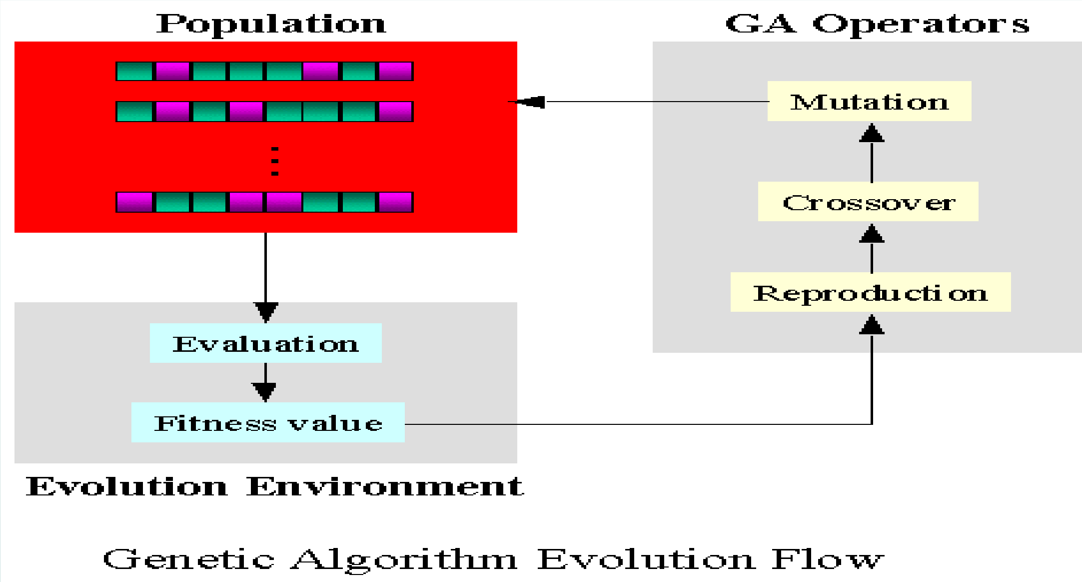
TUDelft

Optimization using Genetic Algorithms By Python!?

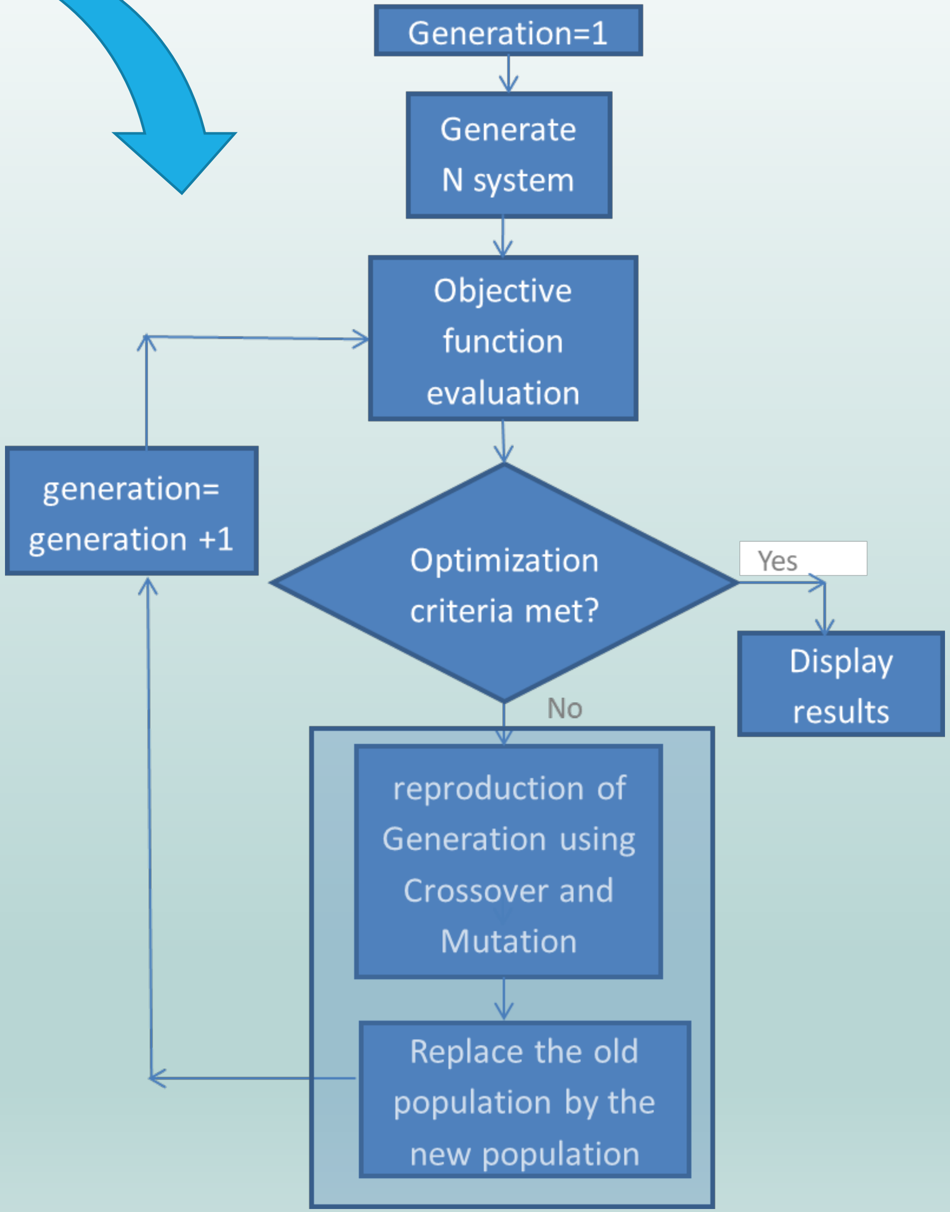
Search in Google →

Continuous Genetic Algorithm From Scratch With Python! ☺

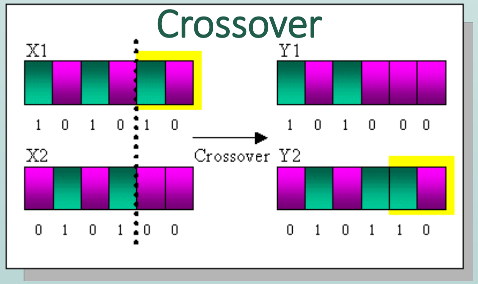
https://towardsdatascience.com/continuous-genetic-algorithm-from-scratch-with-python-ff29deedd099



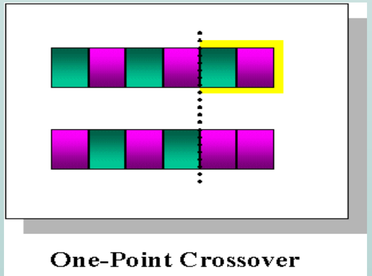
Population
GA Operators
Mutation
Crossover
Reproduction
Evaluation
Fitness value
Evolution Environment
Genetic Algorithm Evolution Flow



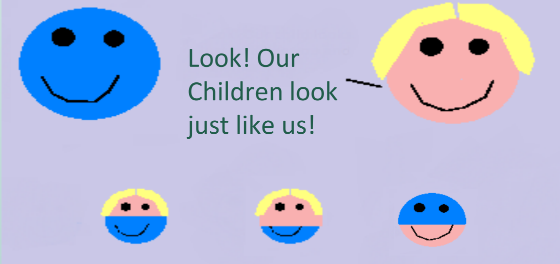
Generation=1
Generate N system
Objective function evaluation
generation= generation +1
Optimization criteria met?
Yes
Display results
No
reproduction of Generation using Crossover and Mutation
Replace the old population by the new population



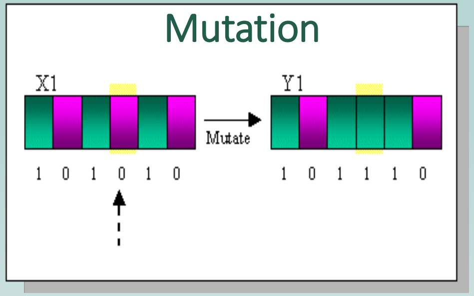
Crossover
X1
1 0 1 0 1 0
Y1
1 0 1 0 0 0
X2
0 1 0 1 0 0
Crossover
Y2
0 1 0 1 1 0



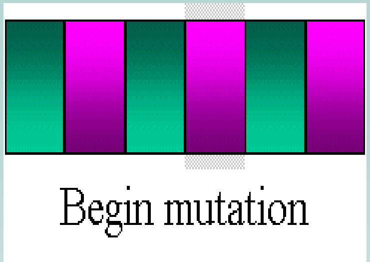
One-Point Crossover



Look! Our Children look just like us!



Mutation
X1
1 0 1 0 1 0
Mutate
Y1
1 0 1 1 1 0



Begin mutation

What Happened?! He doesn`t look like any of us!

```python
import numpy as np
from numpy.random import randint
from random import random as rnd
from random import gauss, randrange
def individual(number_of_genes, upper_limit, lower_limit):
    individual=[round(rnd()*(upper_limit-lower_limit)
                +lower_limit,1) for x in range(number_of_genes)]
    #round(a,n) round the values of a to n digits (here to one digit),
    #rnd()=random number from [0,1) (i.e. =generates Random float x, 0.0 <= x
    return individual
print(individual(4,1,10)) #the upper and lower limit should be integers not f
```

```
[7.2, 6.1, 4.3, 8.4]
```

-Calling functions from python library, such as numpy, randint, gauss

-Define function "individual" to create Individuals

- Input: "number of Genes", "upper limit", "lower limit"
- Output: an "Individual"

```python
def population(number_of_individuals,
               number_of_genes, upper_limit, lower_limit):
    return [
        individual(number_of_genes, upper_limit, lower_limit)
        for x in range(number_of_individuals)
    ]

print(population(8,4,10,1))
def fitness_calculation(individual):
    fitness_value = sum(individual)
    return fitness_value
  # print(fitness_value)
print(fitness_calculation([5.5, 3.8, 3.4, 9.9]))
```

```
[[4.0, 8.1, 7.5, 1.2], [8.4, 9.7, 8.1, 3.0], [8.1, 2.4, 4.6, 2.2], [9.2, 8.2, 8.9, 1.7],
[4.0, 6.7, 3.2, 5.7], [8.6, 3.1, 8.6, 2.5], [7.7, 4.3, 1.8, 6.8], [2.6, 5.7, 8.2, 1.5]]
22.6
```

-Define function "Population" to create many Individuals

- Input: "number of individuals", "number of Genes", "upper limit", "lower limit"
- Output: N "Individual"s

-Define function "Fitness Calculation" to calculate fitness

- Input: "individual"
- Output: fitness value of the individual

	Generation				
	Gene 1	Gene 2	Gene 3	Gene 4	Fitness
Individual 1	10	1	9	9	29
Individual 2	8	8	5	7	28
Individual 3	10	5	6	6	27
Individual 4	8	5	7	4	24
Individual 5	4	4	6	3	17
Individual 6	2	5	5	4	16
Individual 7	8	0	6	0	14
Individual 8	4	3	2	1	10

```python
def roulette(cum_sum, chance):
    veriable = list(cum_sum.copy())
    veriable.append(chance)
    veriable = sorted(veriable)
    return veriable.index(chance)
```

TUDelft

```python
def selection(generation, method='Fittest Half'):
    generation['Normalized Fitness'] = \
        sorted([generation['Fitness'][x]/sum(generation['Fitness'])
        for x in range(len(generation['Fitness']))], reverse = True)
    # this three-lines is one large line . \ is used when we want to continue the sentence

    generation['Cumulative Sum'] = np.array(
        generation['Normalized Fitness']).cumsum()
    if method == 'Roulette Wheel':
        selected = []
        for x in range(len(generation['Individuals'])//2):
            selected.append(roulette(generation
                ['Cumulative Sum'], rnd()))
            while len(set(selected)) != len(selected):
                selected[x] = \
                    (roulette(generation['Cumulative Sum'], rnd()))
        selected = {'Individuals':
            [generation['Individuals'][int(selected[x])]
                for x in range(len(generation['Individuals'])//2)]
                ,'Fitness': [generation['Fitness'][int(selected[x])]
                for x in range(
                    len(generation['Individuals'])//2)]}

    elif method == 'Fittest Half':
        selected_individuals = [generation['Individuals'][-x-1] #start to count from end of vector -1
            for x in range(int(len(generation['Individuals'])//2))] #// means round and make it intiger
        selected_fitnesses = [generation['Fitness'][-x-1]
            for x in range(int(len(generation['Individuals'])//2))]
        selected = {'Individuals': selected_individuals,
                    'Fitness': selected_fitnesses}

    elif method == 'Random':
        selected_individuals = \
            [generation['Individuals']
                [randint(1,len(generation['Fitness']))]
            for x in range(int(len(generation['Individuals'])//2))]
        selected_fitnesses = [generation['Fitness'][-x-1]
            for x in range(int(len(generation['Individuals'])//2))]
        selected = {'Individuals': selected_individuals,
                    'Fitness': selected_fitnesses}

    return selected
id1 = individual(4,1,10)
generation={"Fitness": fitness_calculation(id1)}
print (selection(generation, 'Fittest Half'))
```

-Define function "Selection" to select best Individuals to bring into next generation

- Input: "individuals", "generation", "method of selection"
- Output: N best selected "Individual"s

TU Delft

```python
def pairing(elit, selected, method = 'Fittest'):
    individuals = [elit['Individuals']]+selected['Individuals']
    fitness = [elit['Fitness']]+selected['Fitness']
    if method == 'Fittest':
        parents = [[individuals[x],individuals[x+1]]
                   for x in range(len(individuals)//2)]
    if method == 'Random':
        parents = []
        for x in range(len(individuals)//2):
            parents.append(
                [individuals[randint(0,(len(individuals)-1))],
                 individuals[randint(0,(len(individuals)-1))]])
            while parents[x][0] == parents[x][1]:
                parents[x][1] = individuals[
                    randint(0,(len(individuals)-1))]
    if method == 'Weighted Random':
        normalized_fitness = sorted(
            [fitness[x] /sum(fitness)
             for x in range(len(individuals)//2)], reverse = True)
        cummulitive_sum = np.array(normalized_fitness).cumsum()
        parents = []
        for x in range(len(individuals)//2):
            parents.append(
                [individuals[roulette(cummulitive_sum,rnd())],
                 individuals[roulette(cummulitive_sum,rnd())]])
            while parents[x][0] == parents[x][1]:
                parents[x][1] = individuals[
                    roulette(cummulitive_sum,rnd())]
    return parents
```

-Define function “pairing” to choose how many of the best Individuals we want to keep and bring into next generation

- Input: selected N “individuals”, “elit”, “method of pairing”
- Output: N new “Individual”s ,called “parents”

```python
def mating(parents, method='Single Point'):
    if method == 'Single Point':
        pivot_point = randint(1, len(parents[0]))
        offsprings = [parents[0] \
            [0:pivot_point]+parents[1][pivot_point:]]
        offsprings.append(parents[1]
            [0:pivot_point]+parents[0][pivot_point:])
    if method == 'Two Pionts':
        pivot_point_1 = randint(1, len(parents[0]-1))
        pivot_point_2 = randint(1, len(parents[0]))
        while pivot_point_2<pivot_point_1:
            pivot_point_2 = randint(1, len(parents[0]))
        offsprings = [parents[0][0:pivot_point_1]+
            parents[1][pivot_point_1:pivot_point_2]+
            [parents[0][pivot_point_2:]]]
        offsprings.append([parents[1][0:pivot_point_1]+
            parents[0][pivot_point_1:pivot_point_2]+
            [parents[1][pivot_point_2:]]])
    return offsprings
```

-Define function “mating ” to combine (crossover), the individuals to create new individuals

- Input: N selected “individuals” (=parents), “method of mating”
- Output: N new “individulas” (children) , called “offsprings”

```python
def mutation(individual, upper_limit, lower_limit, muatation_rate=2,
    method='Reset', standard_deviation = 0.001):
    gene = [randint(0, 7)]
    for x in range(muatation_rate-1):
        gene.append(randint(0, 7))
        while len(set(gene)) < len(gene):
            gene[x] = randint(0, 7)
    mutated_individual = individual.copy()
    if method == 'Gauss':
        for x in range(muatation_rate):
            mutated_individual[x] = \
            round(individual[x]+gauss(0, standard_deviation), 1)
    if method == 'Reset':
        for x in range(muatation_rate):
            mutated_individual[x] = round(rnd()* \
                (upper_limit-lower_limit)+lower_limit,1)
    return mutated_individual
```

-Define function “mutation” to insert some mutated individuals among others

- Input: N selected “individuals”, “method of mutation”
- Output: N “mutated individuals”

TUDelft

```python
def next_generation(gen, upper_limit, lower_limit):
    elit = {}
    next_gen = {}
    elit['Individuals'] = gen['Individuals'].pop(-1)
    elit['Fitness'] = gen['Fitness'].pop(-1)
    selected = selection(gen)
    parents = pairing(elit, selected)
    offsprings = [[[mating(parents[x])
                    for x in range(len(parents))]
                    [y][z] for z in range(2)]
                    for y in range(len(parents))]
    offsprings1 = [offsprings[x][0]
                   for x in range(len(parents))]
    offsprings2 = [offsprings[x][1]
                   for x in range(len(parents))]
    unmutated = selected['Individuals']+offsprings1+offsprings2
    mutated = [mutation(unmutated[x], upper_limit, lower_limit)
        for x in range(len(gen['Individuals']))]
    unsorted_individuals = mutated + [elit['Individuals']]
    unsorted_next_gen = \
        [fitness_calculation(mutated[x])
         for x in range(len(mutated))]
    unsorted_fitness = [unsorted_next_gen[x]
        for x in range(len(gen['Fitness']))] + [elit['Fitness']]
    sorted_next_gen = \

         for x in range(len(mutated))]
    unsorted_fitness = [unsorted_next_gen[x]
        for x in range(len(gen['Fitness']))] + [elit['Fitness']]
    sorted_next_gen = \
        sorted([[unsorted_individuals[x], unsorted_fitness[x]]
            for x in range(len(unsorted_individuals))],
                key=lambda x: x[1])
    next_gen['Individuals'] = [sorted_next_gen[x][0]
        for x in range(len(sorted_next_gen))]
    next_gen['Fitness'] = [sorted_next_gen[x][1]
        for x in range(len(sorted_next_gen))]
    gen['Individuals'].append(elit['Individuals'])
    gen['Fitness'].append(elit['Fitness'])
    return next_gen
```

-Define function “next generation”

- Input: generation, upper-limit, lower-limit
- Output: next generation of individuals

```python
# Generations and fitness values will be written to this file
Result_file = 'GA_Results.txt'
# Creating the First Generation
def first_generation(pop):
    fitness = [fitness_calculation(pop[x])
        for x in range(len(pop))]
    sorted_fitness = sorted([[pop[x], fitness[x]]
        for x in range(len(pop))], key=lambda x: x[1])
    population = [sorted_fitness[x][0]
        for x in range(len(sorted_fitness))]
    fitness = [sorted_fitness[x][1]
        for x in range(len(sorted_fitness))]
    return {'Individuals': population, 'Fitness': sorted(fitness)}
pop = population(20,8,1,0)
gen = []
gen.append(first_generation(pop))
fitness_avg = np.array([sum(gen[0]['Fitness'])/
                        len(gen[0]['Fitness'])])
fitness_max = np.array([max(gen[0]['Fitness'])])
res = open(Result_file, 'a')
res.write('\n'+str(gen)+'\n')
res.close()
finish = False
while finish == False:
    if max(fitness_max) > 6:
        break
    if max(fitness_avg) > 5:
        break
    if fitness_similarity_chech(fitness_max, 50) == True:
        break
    gen.append(next_generation(gen[-1],1,0))
    fitness_avg = np.append(fitness_avg, sum(
        gen[-1]['Fitness'])/len(gen[-1]['Fitness']))
    fitness_max = np.append(fitness_max, max(gen[-1]['Fitness']))
    res = open(Result_file, 'a')
    res.write('\n'+str(gen[-1])+'\n')
    res.close()
```

To be Continued….