Algorithm

Version 1: Basic GA

Population size (Pop size) = 1000
No. of Generations (Gen size) = 200
No. of Random Seeds (Seed size) =10

for i in range (seed size):

Initializing “x1 input”, “x2 input”, “fy max”, “fy avg”, “child1 max”, “child2 max” as empty list

for i in range (pop size):

x1 input assigned uniform random number between (-12, 12)
x2 input assigned uniform random number between (-12, 12)

for i in range (gen size):

child1, child2 = crossover function (x1_in, x2_in) #function call

Initializing 4 temporary variables as empty lists
temp_ali1, temp_ali2, temp_fy, temp_fy_real

temp_ali1 = concatenating x1 input along with child1
temp_ali2 = concatenating x2 input along with child2

temp_fy =storing all fitness values by calling cost function(temp_ali1[j], temp_ali2[j]))

Stacking all lists “temp_fy, temp_ali1, temp_ali2” vertically to form a 2d array
Sorting the array by using fitness values in decreasing order

Taking best half of the array by using fitness values

Now again splitting the 2d array into 3 individual lists “max fitness, x1 input, x2 input”

Storing the maximum fitness values along with x1 and x2 values

Calculating and storing the average of all max fitness values

Seed ymax = stacking all the max fitness values vertically
Seed yavg= stacking all the average fitness values vertically

Finding and storing maximum fitness values of each and every seed along with x1 and x2

Calculating the maximum & average fitness values mean and standard deviation

```
Cost function (x1, x2):

    fitness= 21.5 + (x1 * sin (4 * pi * x1)) + (x2 * sin (20 * pi * x2))

    return fitness


Selection function (x1, x2):

    x1_out = random choosing the input x1 value
    x2_out = random choosing the input x1 value

    return x1_out, x2_out


Crossing function (x1, x2):
    x1_out, x2_out = calling selection function (x1_in, x2_in)
    Initializing two empty lists “child1, child2”

    for j in range (pop size//2):
        u = uniform random value between (0,1)

        x1 = randomly choosing one x1 value
        x2 = randomly choosing one x2 value
        x1_1 = randomly choosing one x1 value
        x2_1 = randomly choosing one x1 value
        c11 = (x1 * u) + (x1_1 * (1-u))
        c12 = (x2 * u) + (x2_1 * (1-u))
        c21 = (x1 * (1-u)) + (x1_1 * u)
        c22 = (x2 * (1-u)) + (x2_1 * u)

        creating two crossover children child1, child2

    return child1, child2
```

Version 2: Improved GA

Population size (Pop size) = 1000
No. of Generations (Gen size) = 200
No. of Random Seeds (Seed size) =10

for i in range (seed size):

Initializing “x1 input”, “x2 input”, “fy max”, “fy avg”, “child1 max”, “child2 max” as empty list

for i in range (pop size):

x1 input assigned uniform random number between (-12, 12)
x2 input assigned uniform random number between (-12, 12)

for i in range (gen size):

child1, child2 = crossover function (x1_in, x2_in) #function call

Initializing 4 temporary variables as empty lists
temp_ali1, temp_ali2, temp_fy, temp_fy_real

temp_ali1 = concatenating x1 input along with child1
temp_ali2 = concatenating x2 input along with child2

temp_fy =storing all fitness values by calling cost function(temp_ali1[j], temp_ali2[j]))

Stacking all lists “temp_fy, temp_ali1, temp_ali2” vertically to form a 2d array
Sorting the array by using fitness values in decreasing order

Taking best half of the array by using fitness values

Now again splitting the 2d array into 3 individual lists “max fitness, x1 input, x2 input”

Storing the maximum fitness values along with x1 and x2 values

Calculating and storing the average of all max fitness values

Seed ymax = stacking all the max fitness values vertically
Seed yavg= stacking all the average fitness values vertically

Finding and storing maximum fitness values of each and every seed along with x1 and x2

Calculating the maximum & average fitness values mean and standard deviation

```
Cost function (x1, x2):

   fitness= 21.5 + (x1 * sin (4 * pi * x1)) + (x2 * sin (20 * pi * x2))

   return fitness


Selection function (x1, x2):

   Calculating the fitness value by using cost function(x1,x2)

   And finding the probability of each fitness value by diving with total fitness values

   By using these probabilities, we are going to select the child1 and child 2

   return child1, child2


Crossing function (x1, x2):

   x1_out, x2_out = calling selection function (x1_in, x2_in)
   Initializing two empty lists “child1, child2”

   for j in range (pop size//2):
      u = uniform random value between (0,1)

      x1 = randomly choosing one x1 value
      x2 = randomly choosing one x2 value
      x1_1 = randomly choosing one x1 value
      x2_1 = randomly choosing one x1 value
      c11 = (x1 * u) + (x1_1 * (1-u))
      c12 = (x2 * u) + (x2_1 * (1-u))
      c21 = (x1 * (1-u)) + (x1_1 * u)
      c22 = (x2 * (1-u)) + (x2_1 * u)

      creating two crossover children child1, child2

   Calling mutation function by passing child1 and child2
   return child1, child2


Mutation function (x1, x2):

   Mutation rate =0.1
```

```
for i in range (pop size):
    u1 = uniform random value between (0,1)
    u2 = uniform random value between (0,1)
    if (u1<mutation rate) and (u1 random value + x1[i]) must lie between (-12,12) :
        u1 random value + x1[i] value is stored
    else:
        only x1[i] is stored in child1 variable

    if (u2<mutation rate) and (u2 random value + x1[i]) must lie between (-6,6) :
        u2 random value + x2[i] value is stored
    else:
        only x2[i] is stored in child2 variable

return child1, child2
```

Version 3: Best GA

Population size (Pop size) = 1000
No. of Generations (Gen size) = 200
No. of Random Seeds (Seed size) =10

for i in range (seed size):

Initializing “x1 input”, “x2 input”, “fy max”, “fy avg”, “child1 max”, “child2 max” as empty list

for i in range (pop size):

x1 input assigned uniform random number between (-12, 12)
x2 input assigned uniform random number between (-12, 12)

for i in range (gen size):

child1, child2 = crossover function (x1_in, x2_in) #function call

Initializing 4 temporary variables as empty lists
temp_ali1, temp_ali2, temp_fy, temp_fy_real

temp_ali1 = concatenating x1 input along with child1
temp_ali2 = concatenating x2 input along with child2

temp_fy =storing all fitness values by calling cost function(temp_ali1[j], temp_ali2[j]))

Stacking all lists “temp_fy, temp_ali1, temp_ali2” vertically to form a 2d array
Sorting the array by using fitness values in decreasing order

Taking best half of the array by using fitness values

Now again splitting the 2d array into 3 individual lists “max fitness, x1 input, x2 input”

Storing the maximum fitness values along with x1 and x2 values

Calculating and storing the average of all max fitness values

Seed ymax = stacking all the max fitness values vertically
Seed yavg= stacking all the average fitness values vertically

Finding and storing maximum fitness values of each and every seed along with x1 and x2

Calculating the maximum & average fitness values mean and standard deviation

Cost function (x1, x2):

fitness= 21.5 + (x1 * sin (4 * pi * x1)) + (x2 * sin (20 * pi * x2))

return fitness

Selecting by using uniform randomness (x1, x2):

Uniform selecting 2 variables from x1 and x2

return child1, child2

Selection by using probability function (x1, x2):

Calculating the fitness value by using cost function(x1,x2)

And finding the probability of each fitness value by diving with total fitness values

By using these probabilities, we are going to select the child1 and child 2

return child1, child2

Crossing function (x1, x2):
x1_out, x2_out = calling Selection by using probability function (x1_in, x2_in)
x1_out, x2_out = Selecting by using uniform randomness (x1_out, x2_out)

temp_child1, temp_child2 = Storing half of the selected elements into temporary variables respectively

for j in range (pop size//4):
u = uniform random value between (0,1)

x1 = randomly choosing one x1 value
x2 = randomly choosing one x2 value
x1_1 = randomly choosing one x1 value
x2_1 = randomly choosing one x1 value
c11 = (x1 * u) + (x1_1 * (1-u))
c12 = (x2 * u) + (x2_1 * (1-u))
c21 = (x1 * (1-u)) + (x1_1 * u)
c22 = (x2 * (1-u)) + (x2_1 * u)

And remaining half of the children is calculated by using above equations and storing in child1 and child2 respectively

```
Calling mutation function by passing child1 and child2

return child1, child2
```

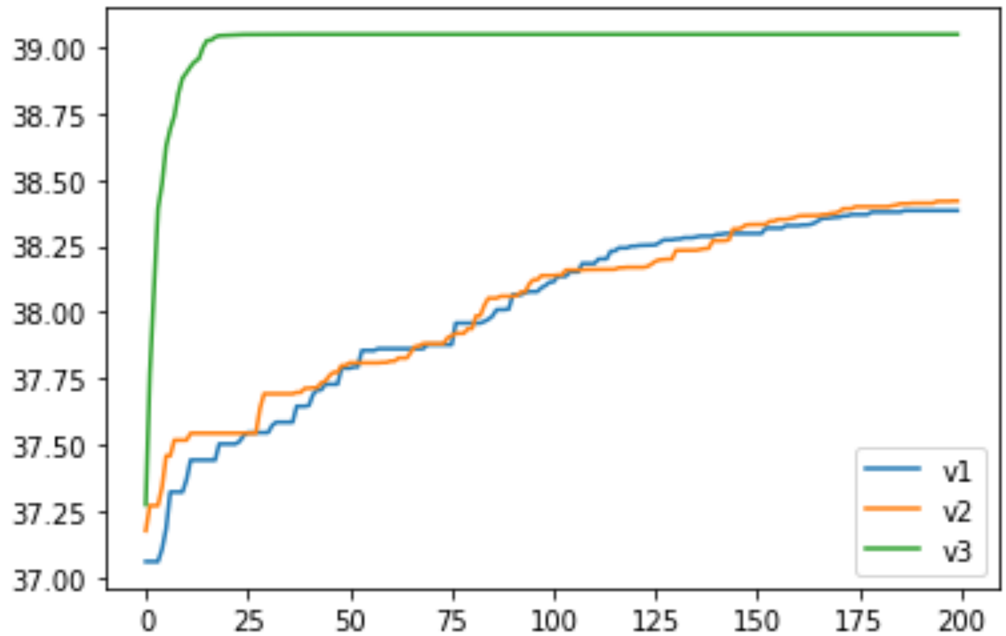
Mutation function (x1, x2):

```
Mutation rate =0.1
for i in range (pop size):
    u1 = uniform random value between (0,1)
    u2 = uniform random value between (0,1)
    if (u1<mutation rate) and (u1 random value + x1[i]) must lie between (-12,12):
        u1 random value + x1[i] value is stored
    else:
        only x1[i] is stored in child1 variable

    if (u2<mutation rate) and (u2 random value + x1[i]) must lie between (-6,6):
        u2 random value + x2[i] value is stored
    else:
        only x2[i] is stored in child2 variable

return child1, child2
```

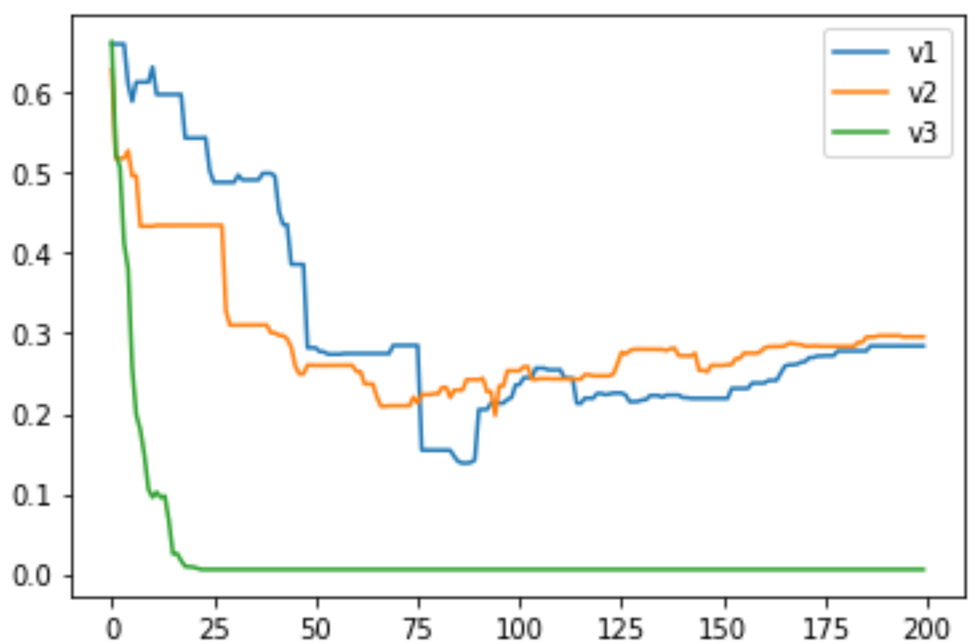
Results

The mean and standard deviation of both *average* and *maximum* fitness of a 1000-individual population over 200 generations in 3 different GA versions.

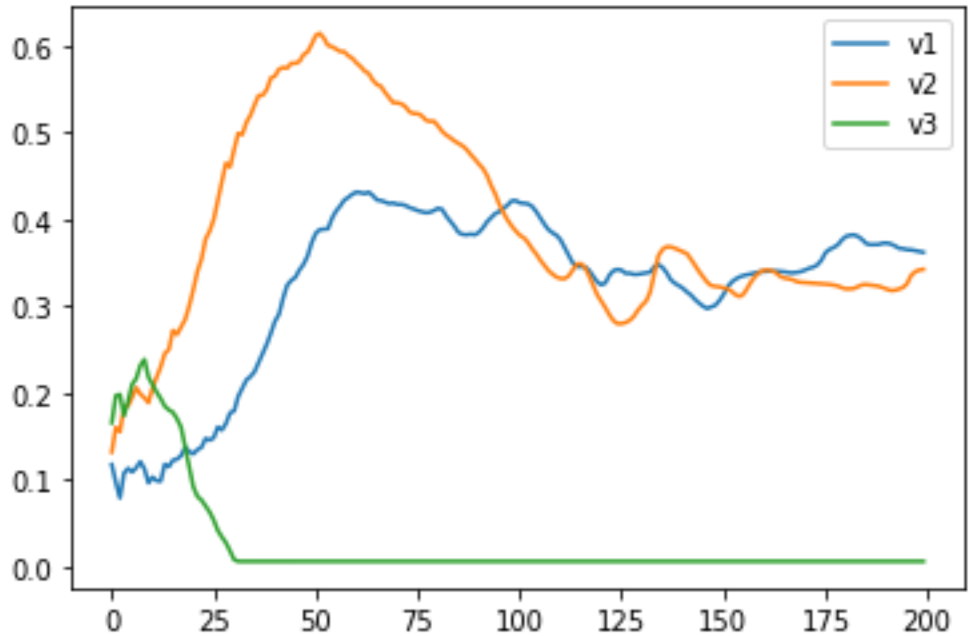
Mean of Maximum fitness values



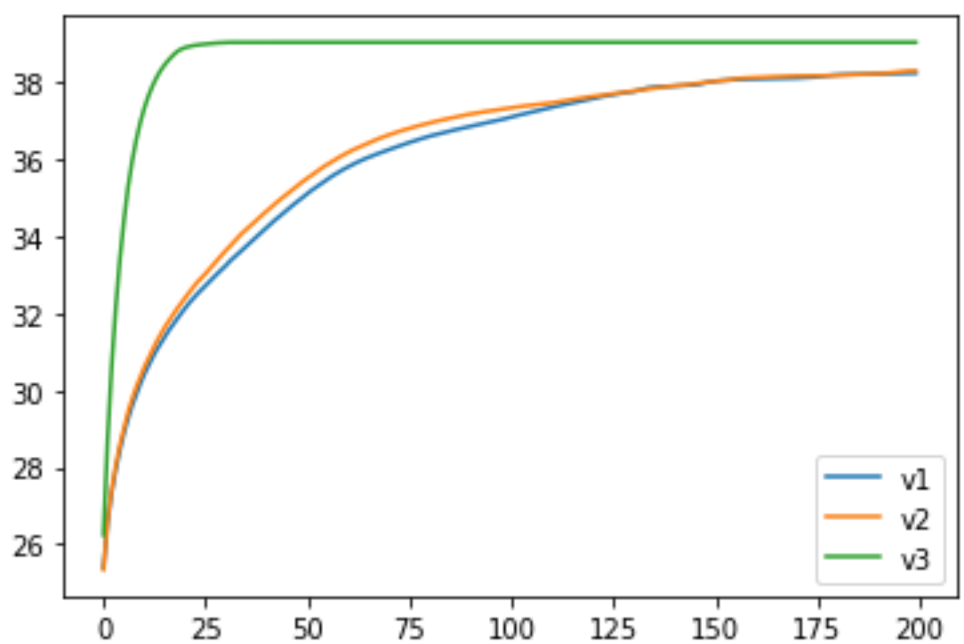
Standard deviation of Maximum fitness values



Mean of Average fitness values



Standard deviation of Average fitness values



Maximum fitness Values for each seed (FF)

Version 1			
Seed No	*F(x)*	*X1*	*X2*
1	38.748284844188944	11.624064884262982	-5.625045564084084
2	38.47390879898389	11.625544703320656	-5.422372315297497
3	38.14921195610446	11.126685810551908	-5.525049328992822
4	38.550295702785576	11.625544700602052	-5.42504669297389
5	38.04326406050072	-11.128371728543636	5.425162411211298
6	38.450296141195615	11.625544703969677	5.3250475679591105
7	38.593265284620834	-11.625680816692693	5.5227571633574195
8	38.72626921834058	-11.613906918817333	5.725044305489543
9	38.24662905685443	-11.125569180466854	-5.624469435984338
10	37.85030883603047	11.125569182690327	5.225048478444867

Version 2			
Seed No	*F(x)*	*X1*	*X2*
1	38.650295275271716	11.62554698542071	-5.525045712953133
2	37.849430396705415	11.12556918237556	-5.225340315756094
3	38.15030751988936	11.125569181937465	-5.525045846166891
4	38.85024373164648	11.625556703959528	5.724977319752917
5	38.4502387578529	-11.625794721072225	-5.325047828573033
6	38.423329753137594	-11.120025816414373	-5.82505238396253
7	38.28151805342908	-11.111684682650633	5.825043567772918
8	38.44955527290818	11.12556907383564	5.824787901269606
9	38.23715288100807	-11.125162209552	-5.623962354922831
10	38.85029447944742	11.625544703825641	-5.725044244817224

Version 3			
Seed No	*F(x)*	*X1*	*X2*
1	39.05029373271072	-11.625544704054107	5.925042751072551
2	39.05029373271072	-11.625544702813656	5.925042750841101
3	39.05029373271072	11.625544703639829	-5.925042750572667
4	39.05029373271072	-11.625544703759761	5.925042751200088
5	39.05029373271072	11.625544704159768	-5.925042751405904
6	39.05029373271072	-11.625544703112663	5.925042751443591
7	39.05029373271072	11.625544703386174	-5.925042751488166
8	39.05029373271072	11.625544703779081	-5.9250427509571715
9	39.03156079444814	11.625544703461308	5.926308575765038,
10	39.05029373271072	11.625544703459902	5.925042751000145

Conclusion

In version 1 basic GA design, I used only the crossover method with random selection. In version 2, I used the crossover method along with fitness-based selection to increase the chance of best fitness values and I implemented a mutation method to avoid local minima. Finally, in version 3, I increased the selection methods by taking two different functions and using a crossover method for only least half of the population. So in conclusion, I can say that because of the two selection methods along with the crossover and mutation methods my version 3 values are higher than other two versions.