Michał Wilkosz

Evolution Strategies – Report

1. Introduction

Presented solution was developed in Python language with external libraries such as Matplotlib(for drawing plots) and Numpy (for calculations). In connection with the necessity to contain used libraries, .exe file size exceed the allowable capacity which is possible to send via mail. Therefore, we provide you a link from which you can download the folder with the entire task. Exe file needs to unarchive .zip folder to work. Execution of provided file takes some time(~2min). Thank you in advance for your understanding.

Link:

https://drive.google.com/file/d/1lSJvcaFdglE00yYOiqSeTIkzhw8lLNs9/view?usp=sharing

The uploaded file is the solution to the first task, using dataset 3 up to the following date of birth – 30.05.1996.

2. Source code

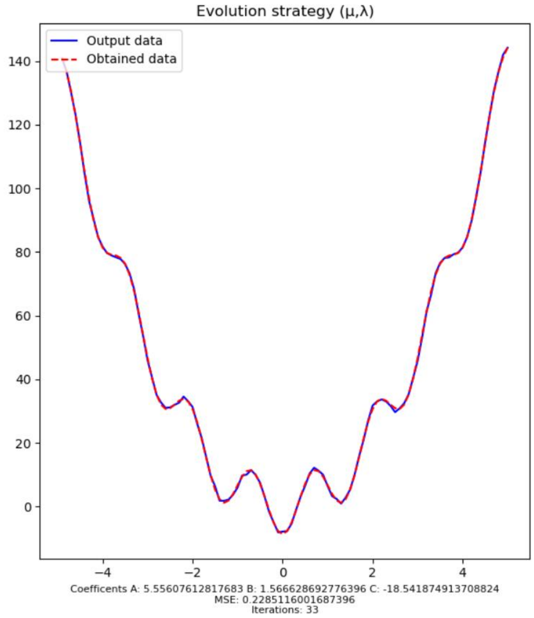
```
import numpy as np
import matplotlib.pyplot as plt
from copy import copy

# 30.05.1996 % 30 = model3.txt
x = np.array([
       -4.9, -4.8, -4.7, -4.6, -4.5, -4.4, -4.3, -4.2, -4.1, -4. , -3.9,
       -3.8, -3.7, -3.6, -3.5, -3.4, -3.3, -3.2, -3.1, -3. , -2.9, -2.8,
       -2.7, -2.6, -2.5, -2.4, -2.3, -2.2, -2.1, -2. , -1.9, -1.8, -1.7,
       -1.6, -1.5, -1.4, -1.3, -1.2, -1.1, -1. , -0.9, -0.8, -0.7, -0.6,
       -0.5, -0.4, -0.3, -0.2, -0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5,
        0.6,  0.7,  0.8,  0.9,  1. ,  1.1,  1.2,  1.3,  1.4,  1.5,  1.6,
        1.7,  1.8,  1.9,  2. ,  2.1,  2.2,  2.3,  2.4,  2.5,  2.6,  2.7,
        2.8,  2.9,  3. ,  3.1,  3.2,  3.3,  3.4,  3.5,  3.6,  3.7,  3.8,
        3.9,  4. ,  4.1,  4.2,  4.3,  4.4,  4.5,  4.6,  4.7,  4.8,  4.9,
        5.
    ])

o = np.array([
       140.82144   , 136.80255   , 130.16584   , 123.09854   ,
       114.39532   , 104.59727   ,  96.025621  ,  90.400328  ,
        84.738771  ,  81.702265  ,  79.660375  ,  78.901891  ,
        78.319333  ,  77.792199  ,  76.203148  ,  73.302882  ,
        68.357995  ,  60.673796  ,  53.659575  ,  45.933771  ,
        40.527897  ,  35.071689  ,  32.667226  ,  30.980565  ,
```

```
        31.142812  ,  31.947988  ,  32.60861   ,  34.54083   ,
        33.087014  ,  31.259531  ,  26.126994  ,  21.595553  ,
        15.708787  ,   9.6828575 ,   6.3519347 ,   1.7911671 ,
         1.7895385 ,   2.1555959 ,   3.7013691 ,   5.9731705 ,
         9.7150275 ,  10.00916   ,  11.432421  ,   9.9223725 ,
         7.4511493 ,   3.0999315 ,  -1.7130687 ,  -4.954152  ,
        -8.1451161 ,  -7.9157024 ,  -7.7265311 ,  -5.4184115 ,
        -1.0247075 ,   3.1055918 ,   6.4619733 ,  10.332118  ,
        12.17703   ,  11.234617  ,   9.9610569 ,   6.4187102 ,
         3.229897  ,   2.3680728 ,   0.88608265,   2.7783012 ,
         5.3082157 ,   9.9966809 ,  15.789251  ,  20.939401  ,
        26.937442  ,  31.721399  ,  33.030669  ,  33.657274  ,
        33.010668  ,  31.599644  ,  29.644646  ,  30.833292  ,
        32.41725   ,  35.2346    ,  40.404607  ,  45.742281  ,
        53.113719  ,  61.332217  ,  66.598888  ,  73.333123  ,
        76.564083  ,  78.123728  ,  78.311912  ,  79.310875  ,
        79.77859   ,  81.440103  ,  84.696972  ,  89.857463  ,
        97.291561  , 105.03889   , 114.43444   , 123.03707   ,
       130.96645   , 136.87229   , 142.06587   , 144.23329
    ])

#Constant values
abc = np.random.uniform(-10, 10, size = (100,3))
sigma = np.random.uniform(0,10, size = (100,3))
tau1 = 1/np.sqrt(2*6)
tau2 = 1/np.sqrt(2*np.sqrt(6))

#Evaluate function
def evaluate(abc, x, i):
    return(abc[0]*((x[i]**2)-(abc[1]*np.cos(abc[2]*np.pi*x[i]))))

#Fitness for a, b, c coefficients
def abc_fitness(abc, N):
    o_dash=np.zeros(N)
    for i in range (100):
        o_dash[i]=evaluate(abc,x,i)
    mes=np.mean((o-o_dash)**2)
    return mes

#Evluate population
def evaluate_pop(pop_v):
    fit=np.zeros(len(pop_v[0]))
    for i in range(len(pop_v[0])):
        fit[i]=abc_fitness(pop_v[0,i,:],100)
    return fit

#Main algorithm loop
#strategy == 1 ((μ+λ) evolution strategy)
#strategy == 2 ((μ,λ) evolution strategy)
```

```python
#v = chromsome
def evolution_strategy(strategy):

    #Chromosome format [abc,sigma] 6 values
    v = np.stack([abc,sigma])

    iteration=0
    while True:

        parents_fit = evaluate_pop(v)
        index = np.random.randint(100, size=500)
        r_lambda = v[:,index,:]
        off_lambda = np.zeros((2,len(r_lambda[1]),3))

        for i in range(len(r_lambda[1])):

            #Mutation
            a_off = r_lambda[0,i,0] + r_lambda[1,i,0]*np.random.normal()
            off_lambda[0,i,0] = a_off
            b_off = r_lambda[0,i,1] + r_lambda[1,i,1]*np.random.normal()
            off_lambda[0,i,1] = b_off
            c_off = r_lambda[0,i,2] + r_lambda[1,i,2]*np.random.normal()
            off_lambda[0,i,2] = c_off

            r_sigm1 = tau1 * np.random.normal()
            r_sigm2 = tau2 * np.random.normal(size=3)

            sigma_a_off = r_lambda[1,i,0] * np.exp(r_sigm1) * np.exp(r_sigm2[0
])
            off_lambda[1,i,0] = sigma_a_off
            sigma_b_off = r_lambda[1,i,1] * np.exp(r_sigm1) * np.exp(r_sigm2[1
])
            off_lambda[1,i,1] = sigma_b_off
            sigma_c_off = r_lambda[1,i,2] * np.exp(r_sigm1) * np.exp(r_sigm2[2
])
            off_lambda[1,i,2] = sigma_c_off

        #Selection (μ+λ) fittest offspring and parents are merged
        if strategy == 1:
            fit_lambda = evaluate_pop(off_lambda)
            mi_p_lambd = np.concatenate((v, off_lambda), axis=1)
            mi_p_lambd_mse = np.concatenate((parents_fit, fit_lambda), axis=0)

            choosen_indx = mi_p_lambd_mse.argsort()[:100]
            v = mi_p_lambd[:,choosen_indx,:]

        #Selection (μ,λ) all parents are discarded
        if strategy == 2:
            fit_lambda = evaluate_pop(off_lambda)
```

```
            choosen_indx = fit_lambda.argsort()[:100]
            v = off_lambda[:,choosen_indx,:]

        iteration += 1
        #Stop criterion
        if abs((min(parents_fit) - min(fit_lambda))) < 1e-6:
            coeff_output = []
            min_mse = min(fit_lambda)
            min_mse_index = int(np.argwhere(fit_lambda==min_mse))
            coeff_abc = off_lambda[0,min_mse_index]

            for i in x:
                coeff_output.append(coeff_abc[0]*((i**2)-
(coeff_abc[1]*np.cos(coeff_abc[2]*np.pi*i))))

            if strategy == 1:
                plot=plt.plot(x,o,'b-',label="Output data")
                plt.plot(x,coeff_output,'r--',label="Obtained data")
                plt.title("Evolution strategy (μ+λ)")
                plt.xlabel("Coefficents A: {} B: {} C: {}\nMSE: {}\nIterations
: {}".format(coeff_abc[0],coeff_abc[1],coeff_abc[2],min_mse,iteration),fontsiz
e=8)
                plt.legend(loc='upper left')

            if strategy == 2:
                plot=plt.plot(x,o,'b-',label="Output data")
                plt.plot(x,coeff_output,'r--',label="Obtained data")
                plt.title("Evolution strategy (μ,λ)")
                plt.xlabel("Coefficents A: {} B: {} C: {}\nMSE: {}\nIterations
: {}".format(coeff_abc[0],coeff_abc[1],coeff_abc[2],min_mse,iteration),fontsiz
e=8)
                plt.legend(loc='upper left')

            break

    return plot



gs = plt.GridSpec(2,2)
fig = plt.figure()
ax1 = fig.add_subplot(gs[:, 0])
ax1=evolution_strategy(1)
ax2 = fig.add_subplot(gs[:, 1])
ax2=evolution_strategy(2)
plt.show()
```

3. Solution



Conclusions:

- The number of iterations is usually smaller for (μ+λ) strategy.
- Sometimes for (μ,λ) strategy it happens that the algorithm stops reaching MSE = 38.xx, this is due to the stop criterion which compares MSE differences between iterations.
- The best possible solution is usually achieved after ~30+ iterations for (μ+λ) strategy and ~40+ iterations for (μ,λ) strategy.
- With several algorithm runs the final results are 0.288xx for both strategies.