CSC 467 2.0 Evolutionary Computing  
Assignment 2  
**Index # AS2016525  
M.D.C. Rukshan Suriyaaratchie**

**Table of Contents**

[**Code 2**](#_Toc46080381)

[Run 1:   
Results 3  
Plot 3](#_Toc46080382)

[Run 2:   
Results 4  
Plot 4](#_Toc46080382)

[Run 3:   
Results 5  
Plot 5](#_Toc46080382)

[Run 4:   
Results 6  
Plot 6](#_Toc46080382)  
 [**Discussion 7**](#_Toc46080382) [**Conclusion 7**](#_Toc46080382)

**Code**

import numpy as np

import matplotlib.pyplot as plt

def de(fobj, bounds, mut=0.8, crossp=0.7, popsize=20, its=1000):

    dimensions = len( bounds )

    pop = np.random.rand( popsize, dimensions )

    min\_b, max\_b = np.asarray( bounds ).T

    diff = np.fabs( min\_b - max\_b )

    pop\_denorm = min\_b + pop \* diff

    fitness = np.asarray( [fobj( ind ) for ind in pop\_denorm] )

    best\_idx = np.argmin( fitness )

    best = pop\_denorm[best\_idx]

    for i in range( its ):

        for j in range( popsize ):

            idxs = [idx for idx in range( popsize ) if idx != j]

            a, b, c = pop[np.random.choice( idxs, 3, replace=False )]

            mutant = np.clip( a + mut \* (b - c), 0, 1 )

            cross\_points = np.random.rand( dimensions ) < crossp

            if not np.any( cross\_points ):

                cross\_points[np.random.randint( 0, dimensions )] =True

            trial = np.where( cross\_points, mutant, pop[j] )

            trial\_denorm = min\_b + trial \* diff

            f = fobj( trial\_denorm )

            if f < fitness[j]:

                fitness[j] = f

                pop[j] = trial

                if f < fitness[best\_idx]:

                    best\_idx = j

                    best = trial\_denorm

        yield best, fitness[best\_idx]

def f\_obj(x1, x2):

    return pow( (x1 + 2 \* x2 - 7), 2 ) + pow( (2 \* x1 + x2 - 5), 2 )

for d in range( 30, 300, 20 ):

    result = list( de( lambda x: f\_obj( x[0], x[1] ), bounds=[(-10, 10)] \* 2, its=d ) )

    x, f = zip( \*result )

    plt.plot( f, label='iteration={}'.format( d ) )

    print(result[-1])

plt.grid()

plt.rcParams["figure.figsize"] = (750, 750)

plt.legend()

**Run 1:**

**Results**

(array([0.95388423, 3.02623318]), 0.004396113851650085)

(array([0.99908238, 3.00012905]), 3.3460495769480616e-06)

(array([1.00002093, 2.9999564 ]), 4.3938700822717066e-09)

(array([1.00000246, 2.99999336]), 1.2023340924545983e-10)

(array([1.00000035, 2.99999961]), 2.844034227115767e-13)

(array([1., 3.]), 7.599191752763452e-18)

(array([1., 3.]), 2.369530508132133e-17)

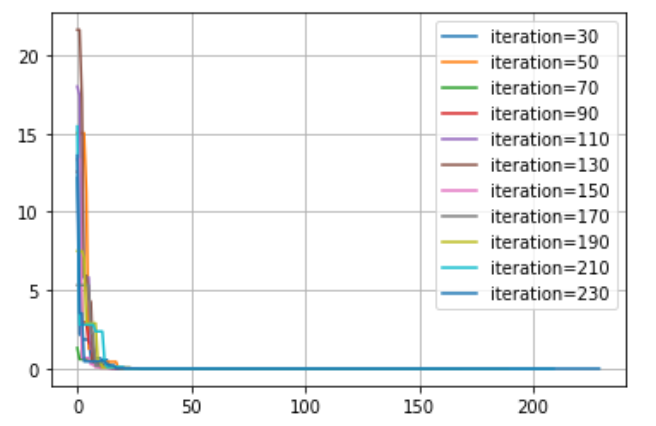
(array([1., 3.]), 4.917839410068993e-22)

(array([1., 3.]), 1.2315522902911288e-23)

(array([1., 3.]), 1.2549199280255856e-25)

(array([1., 3.]), 0.0)

**Plot**

****

**Run 2:**

**Results**

(array([1.03449198, 2.95212714]), 0.004197699537059782)

(array([0.99796553, 3.00248738]), 1.1146600595691996e-05)

(array([0.99999777, 2.9999622 ]), 7.845466247206218e-09)

(array([1.00000539, 2.99999444]), 6.011816613792951e-11)

(array([1.00000004, 2.99999996]), 3.3021580991045217e-15)

(array([1.00000001, 2.99999999]), 2.02880681192957e-16)

(array([1., 3.]), 4.1929226997116175e-19)

(array([1., 3.]), 2.2358987291685123e-19)

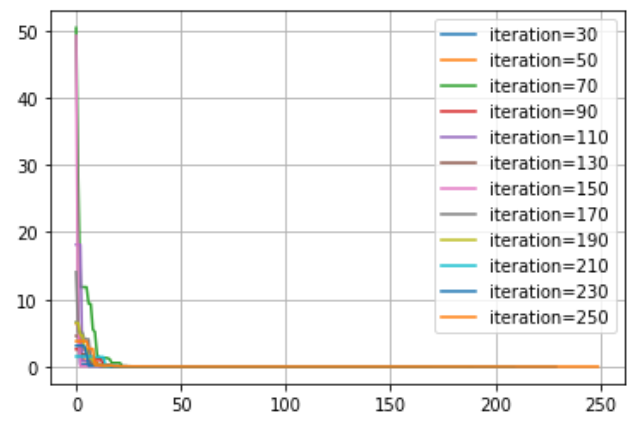
(array([1., 3.]), 1.2910813119209168e-25)

(array([1., 3.]), 2.145701662201152e-28)

(array([1., 3.]), 2.524354896707238e-29)

(array([1., 3.]), 0.0)

**Plot**



**Run 3:**

**Results**

(array([1.00068176, 3.00492581]), 0.00015050797765621486)

(array([0.99886079, 3.00115553]), 2.6341147923691524e-06)

(array([1.00001934, 2.99992965]), 1.5727944417040896e-08)

(array([0.99999131, 3.00000053]), 3.4232542474686633e-10)

(array([0.99999996, 3.00000001]), 5.943460138058679e-15)

(array([0.99999999, 3.00000003]), 2.615297231700319e-15)

(array([1., 3.]), 2.0281013559234127e-19)

(array([1., 3.]), 1.004208604303749e-22)

(array([1., 3.]), 1.834221511496446e-23)

(array([1., 3.]), 3.6716741972606773e-26)

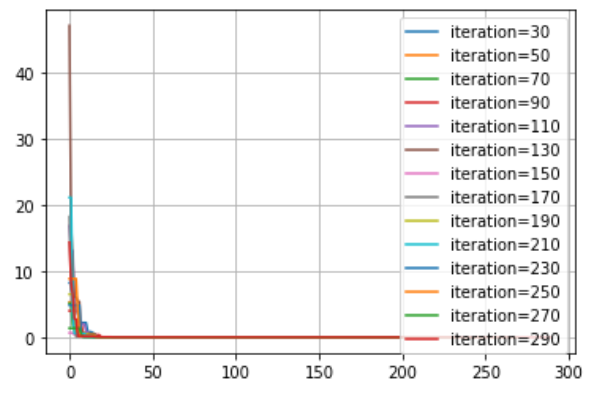
(array([1., 3.]), 0.0)

(array([1., 3.]), 0.0)

(array([1., 3.]), 0.0)

(array([1., 3.]), 0.0)

**Plot**

****

**Run 4:**

**Results**

(array([1.00700877, 2.99827387]), 0.00016372727278238003)

(array([0.99935719, 2.99812599]), 2.9262666278892852e-05)

(array([0.99998194, 3.00001443]), 5.869641947625117e-10)

(array([0.99998587, 3.00001101]), 3.599169978435052e-10)

(array([1.00000006, 2.99999999]), 1.3161892430613935e-14)

(array([1.00000004, 2.99999996]), 3.1798009471751253e-15)

(array([1., 3.]), 1.2099921285280378e-21)

(array([1., 3.]), 1.1383689272897364e-19)

(array([1., 3.]), 8.283713507578045e-23)

(array([1., 3.]), 6.310887241768094e-29)

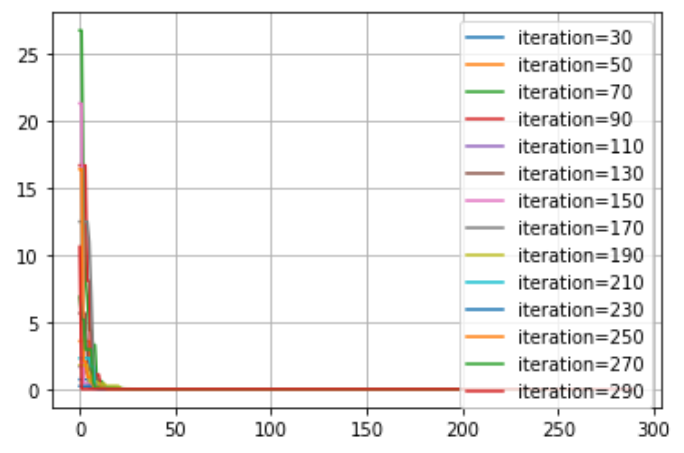
(array([1., 3.]), 1.5777218104420236e-29)

(array([1., 3.]), 0.0)

(array([1., 3.]), 0.0)

(array([1., 3.]), 0.0)

**Plot**

****

**Discussion**

There are 4 sets of results mentioned above and those are taken according to the number of runs with 300 iterations. Each result output line time the number of iterations is setup to increment by 20 and the initial output iteration is 30 at each run.

From the start of the run the numbers are off and with each increment the numbers get better. After the 10th & 11th Results which are the 210th and the 230th iteration the numbers tend to be good. So above 10th and 11th iterations we can see a constant stable result output at each run.

**Conclusion**

According to the results and the graphs it is clear that when the number of iterations is getting higher the accuracy tends to be high.

From the above mentioned 4 different runs there are no any relation with the changes of number of runs according to my perspective.