Defferential Evolution (eralize rosenbrock)

```
import numpy as np
import random
from array import *
import matplotlib.pyplot as plt
import statistics as st
import math


D = 10 # Dimensi dari permasalahan
#maxit = 99 #Max iterasi
maxit = 495 #Max iterasi
ukuranPopulasi = 10
LB = -30 #Batas Bawah
UB = 30 #Batas Atas
CR = 0.9
F = 0.5
```

+ Code + Text

```
import random
def initPopulasi(ukuranPopulasi, D, UB, LB): #Inisialisasi setiap baris kolom
    P = np.empty((ukuranPopulasi,D))
    for i in range(ukuranPopulasi):
        for j in range(D):
            P[i][j]= (random.random()*(UB-LB))+LB
    return P

P = initPopulasi(ukuranPopulasi, D, UB, LB)
print(P)
```

```
[[ -7.61081227  16.75428072  13.21043827   9.38809217  25.82375783
   -5.97266     16.72879063  13.07785531   9.79193829   6.31141543]
 [ 14.11978806  20.20459763  26.02100856  18.28858776  22.17751602
    8.0744026   -1.65119132  22.81829131 -10.81018187   7.66739252]
 [-15.10608193   0.97648733   2.78299542  20.21540426 -22.63091213
   -0.22592397 -17.87334542  -8.32939924   7.02288728 -26.12762238]
 [ -7.15930251  17.81181178 -26.86173356  20.98898672  14.43410529
   17.60415444  14.01785515   1.76782069  20.82162597  -2.22402291]
 [-25.67739085  29.91880602  18.05535191   5.13380508 -18.54210928
   -1.38544472   0.03642762 -19.30382664  23.95611823 -25.43411657]
 [ 10.7763051   -8.35561497   6.11354222 -26.7781645  -14.38960392
  -23.84762208 -11.61579588  -8.19720304 -25.1864147   -6.95201705]
 [ -4.62853907  27.95150174  14.55237837  23.55990911   3.19809128
   29.5817369  -23.35268425 -19.60324441  15.03486828  28.29263551]
 [ -8.1061652  -20.93191898  -2.15838804 -18.47794075  19.48967109
   27.08795231   5.98748489  22.23064494  22.13023058  25.32642625]
 [ -3.8058883   -9.50922113 -28.02666694  13.71607392   3.55230585
   12.0658694  -23.06839754   6.16264274  26.01362888  27.60170642]
 [-25.99298495   0.94112291  -3.11709472 -23.00655264 -21.68642299
   -6.54808417 -20.21544157 -29.37906683  -7.21067292  23.19687774]]
```

```
def Fitness_Func(populasi):
    ukuran = populasi.shape
    ukuranPopulasi = ukuran[0]
    dimensi = ukuran[1]

    Fobj = np.empty(ukuranPopulasi)

    for i in range (ukuranPopulasi):
        d = 0
        for j in range (dimensi):
            #d = d + populasi[i][j]**2
            #d = d + np.sum(np.abs(populasi[i][j])) + np.product(np.abs(populasi[i][j])) #
            d = d + np.sum(100*(populasi[i][j+1)-(populasi[i][j]))**2 + (populasi[i][j]-1
            # d =  d + np.sum( (populasi[i][j]**2) - (10*math.cos(2*math.pi*populasi[i][j]
        Fobj[i] = d
    return Fobj

Fobj = Fitness_Func(P)
print(Fobj)
```

```
 [20560275.41139579 30795754.29945384 21260650.63404217 27847314.01302745
  37536067.18847232 23447942.13629209 45209389.36433864 37066993.34578882
  32400777.01344717 33057531.1409331 ]
```

```
def crossOver(populasi,F,j):
    ukuran = populasi.shape
    ukuranPopulasi = ukuran[0]
    Dimensi = ukuran[1]

    r1 = random.randint(0, ukuranPopulasi-1)
    r2 = random.randint(0, ukuranPopulasi-1)
    r3 = random.randint(0, ukuranPopulasi-1)

    while r1 == r2:
        r2 = (r2 + 1)%ukuranPopulasi

    while r3 == r1 or r3 == r2:
        r3 = (r3 + 1)%ukuranPopulasi

    v = P[r3][j] + F*(P[r1][j] - P[r2][j])
    return v

v = crossOver(P,F,1)
print(v)
```

```
 -19.735526055207696
```

```
P = initPopulasi(ukuranPopulasi, D, UB, LB)
print("Inisialisasi Populasi :\n",P)
print(P)
```

```
Fobj = Fitness_Func(P)
print(Fobj)
print("\nFungsi Objective :\n",Fobj)

U = np.empty((1,D))
bestFobj = np.empty((maxit+1))
bestFobj[0] = Fobj.min()

for it in range(maxit):
    for i in range(ukuranPopulasi):
        for j in range(D):
            U[0][j] = P[i][j]

        jrand=random.randint(0,D)

        for j in range(D):
            if random.random() < CR or j == jrand:
                v = crossOver(P,F,j)
                U[0][j] = v

        FobjU = Fitness_Func(U)

        if FobjU < Fobj[i]:
            Fobj[i] = FobjU
            for j in range(D):
                P[i][j] = U[0][j]

    bestFobj[it+1] = Fobj.min()

    print("\nNilai Optimal : \n",bestFobj)

    for i in range(30):
        bestFobj
```

```
 Streaming output truncated to the last 5000 lines.
  4.04074231e+001 4.04050816e+001 4.04050816e+001 4.04050816e+001
  4.04032530e+001 4.04032530e+001 4.04028231e+001 4.04028231e+001
  4.04028231e+001 4.04025441e+001 4.04024657e+001 4.04024657e+001
  4.04023898e+001 4.04021328e+001 4.04018571e+001 4.04017783e+001
  4.04017783e+001 4.04017783e+001 4.04017783e+001 4.04017783e+001
  4.04012881e+001 4.04012881e+001 4.04012881e+001 4.04012881e+001
  4.04012881e+001 4.04012881e+001 4.04012881e+001 4.04012266e+001
  4.04012266e+001 4.04012074e+001 4.04011685e+001 4.04011685e+001
  4.04010691e+001 4.04010691e+001 4.04010466e+001 4.04010466e+001
  4.04010466e+001 4.04010466e+001 4.04010466e+001 4.04009850e+001
  4.04009850e+001 4.04009850e+001 4.04009850e+001 4.04009569e+001
  4.04009569e+001 4.04009569e+001 4.04009569e+001 4.04009569e+001
  4.04009569e+001 4.04009569e+001 4.04009566e+001 4.04009566e+001
  4.04009566e+001 4.04009506e+001 4.04009494e+001 4.04009424e+001
  4.04009424e+001 4.04009424e+001 4.04009424e+001 4.04009424e+001
  4.04009424e+001 4.04009424e+001 4.04009424e+001 4.04009424e+001
  4.04009424e+001 4.04009424e+001 4.04009424e+001 4.04009420e+001
  4.04009420e+001 4.04009420e+001 4.04009420e+001 4.04009416e+001
  4.04009416e+001 4.04009410e+001 4.04009399e+001 4.04009392e+001
  4.04009392e+001 4.04009392e+001 4.04009392e+001 4.04009392e+001
  4.04009392e+001 4.04009392e+001 4.04009390e+001 4.04009390e+001
  4.04009388e+001 4.04009388e+001 4.04009388e+001 4.04009387e+001
```

```
 4.04009387e+001 4.04009387e+001 4.04009387e+001 4.04009387e+001
 4.04009387e+001 4.04009387e+001 4.04009387e+001 4.04009387e+001
 4.04009387e+001 4.04009387e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 4.04009386e+001 4.04009386e+001
 4.04009386e+001 4.04009386e+001 6.90122489e-310 6.90120863e-310
 6.90121970e-310 6.90122494e-310 6.90122471e-310 6.90120862e-310
 6.90120837e-310 6.90122493e-310 6.90120834e-310 6.90120834e-310
 6.90120837e-310 6.90120834e-310 6.90120834e-310 6.90120834e-310
 6.90122291e-310 6.90122262e-310 6.90122262e-310 6.90122262e-310
 6.90122262e-310 6.90122263e-310 6.90120834e-310 6.90122262e-310
 6.90122291e-310 6.90122291e-310 6.90122262e-310 6.90122262e-310
 6.90122494e-310 6.90122490e-310 0.00000000e+000 0.00000000e+000
 0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000
 0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000
 6.90120844e-310 6.90120841e-310 6.90120841e-310 6.90120844e-310
 6.90120844e-310 6.90120844e-310 6.90122434e-310 6.90120844e-310
 6.90120841e-310 6.90120841e-310 6.90120839e-310 6.90122494e-310
```

```
import statistics as st
print("Nilai Mean : ",st.mean(bestFobj))
print("Nilai Standard Deviation : ",st.stdev(bestFobj))
print("Nilai Minimal : ",np.min(bestFobj))
```

```
Nilai Mean :  382855.90483743796
Nilai Standard Deviation :  1976442.722855424
Nilai Minimal :  40.40093858396244
```

```
x = np.linspace(0, 1, maxit+1)
plt.plot(x, bestFobj, label= 'f=Fobj')
plt.show()
```

Genetic Algorithm

```
import numpy as np
import random
from array import *
import matplotlib.pyplot as plt
import statistics as st
import math
```

```
N = 3
rows, cols = (N, 4)
induk = [[0 for i in range(cols)] for j in range(rows)]
print (induk)

N_Anak = 6
rows, cols = (N_Anak, 4)
anak = [[0 for i in range(cols)] for j in range(rows)]
print(anak)
```

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
#Inisialiasi Parameter
Pc = 0.9
Pm = 0.1

for i in range(N):
    a = random.randint(0, 15)
    induk[i] = [int (x) for x in '{:04b}'.format(a)]
    print(a, induk[i])
```

```
13 [1, 1, 0, 1]
12 [1, 1, 0, 0]
7 [0, 1, 1, 1]
```

```
#Function cross over
def crossover(ind1=[], ind2=[]):
```

```
    anak1 = [0, 0, 0, 0]
    anak2 = [0, 0, 0, 0]


    anak1[0] = ind1[0]
    anak1[1] = ind1[1]
    anak1[2] = ind2[2]
    anak1[3] = ind2[3]

    anak2[0] = ind2[0]
    anak2[1] = ind2[1]
    anak2[2] = ind1[2]
    anak2[3] = ind1[3]

    return anak1, anak2


#Function  mutasi
def mutasi(ind=[]):
    ind[1] = 1 - ind[1]
    return ind



#Function hitung Int
def hitungInt(ind = []):
    a = ind[3] * 1
    a = a + ind[2] * 2
    a = a + ind[1] * 4
    a = a + ind[0] * 8
    return a



#Buat function untuk melakukan hitung fitness
def hitungFitness(ind = []):
    a = hitungInt(ind)
    #y = a**2 #sphere
    #y = np.sum(np.abs(a)) + np.product(np.abs(a)) #schwefel 2.22
    y = np.sum(100*(a+1)-(a))**2 + (a-1)**2 #Generalize rosenbrock
    # y = np.sum( (a**2) - (10*math.cos(2*math.pi*a)) + 10 ) #rastrigins

    return y


i = 1
epochs = 495
#epochs = 99
MaxFitness = [0 for i in range(epochs)]
angkaFitness = [0 for i in range(epochs)]

#nilai fitness induk
for j in range(N):
    if (hitungFitness(induk[j])>MaxFitness[0]):
```

```
        MaxFitness[0] = hitungFitness(induk[j])

#'epochs' kali
for i in range(30):
  while i < epochs:
    print("Iterasi: ", i+1)
    #Reproduksi
    a = random.random()
    if (a<Pc):
        #Cross over
        anak[0], anak[1] = crossover(induk[0], induk[1])
        anak[2], anak[3] = crossover(induk[0], induk[2])
        anak[4], anak[5] = crossover(induk[1], induk[2])
    else:
        #Mutasi
        for j in range(N):
            induk [j] = mutasi(induk[j])
    #Elistism
    minFitness = 30
    idx = 0
    idxanak = 0
    #minFitness (induk terburuk)
    for j in range(len(induk)):
        if (hitungFitness(induk[j])<minFitness):
            minFitness = hitungFitness(induk[j])
            idx = j

    #cari maxFit (Anak terbaik)
    maxFit = -30
    for j in range(len(anak)):
        if (hitungFitness(anak[j])>maxFit):
            maxFit = hitungFitness(anak[j])
            idxanak = j
    #Individual replacement
    if (minFitness < maxFit):
        induk[idx] = anak[idxanak]

    #Populasi induk yang paling baik
    for j in range(N):
        if (hitungFitness(induk[j])>MaxFitness[i]):
            MaxFitness[i] = hitungFitness(induk[j])
    i+=1

    print(MaxFitness)
```

```
Iterasi:  1
[1923913, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
Iterasi:  2
[1923913, 1413821, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
Iterasi:  3
[1923913, 1413821, 1413821, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
Iterasi:  4
[1923913, 1413821, 1413821, 1413821, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
Iterasi:  5
[1923913, 1413821, 1413821, 1413821, 2512421, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
Iterasi:  6
```

```
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
Iterasi:  7
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 0, 0, 0, 0, 0, 0, (
Iterasi:  8
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 0, 0, 0, (
Iterasi:  9
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821, (
Iterasi:  10
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  11
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  12
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  13
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  14
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  15
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  16
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  17
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  18
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  19
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  20
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  21
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  22
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  23
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  24
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  25
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  26
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  27
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  28
[1923913, 1413821, 1413821, 1413821, 2512421, 1413821, 1413821, 1413821, 1413821,
Iterasi:  29
```

```
print("Nilai Mean : ", st.mean(MaxFitness))
print("Nilai Standard Deviation : ", st.stdev(MaxFitness))
print("Nilai Minimal : ",np.min(MaxFitness))
```

```
Nilai Mean :  2509012
Nilai Standard Deviation :  55969.79841843277
Nilai Minimal :  1413821
```

```
x = [i for i in range(epochs)]
plt.plot(x, MaxFitness)
```

```
[<matplotlib.lines.Line2D at 0x7f0a422fa310>]
```