ANEXO 1 – Código implementado

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 13 18:33:07 2020

@author: Amanda
"""


import datetime
import random
from random import randint
import math
import numpy as np
import operator

import matplotlib.pyplot as plt


inicio = datetime.datetime.now()


class Cidade:
    def __init__ (self, id, x, y):
        self.id = id
        self.x = x
        self.y = y

class Rota:
    def __init__ (self, sequencia, matriz_ady):
        self.sequencia = sequencia
        self.aptitud = self.CalcAptidão(self.sequencia, matriz_ady)

    def CalcAptidão(self, recor, matriz_ady):
        dist = 0.0
        for i in range (0, len(recor)-1):
            dist += matriz_ady[recor[i]][recor[i+1]]
        #print ("X ",matriz_ady[len(recor)-1][recor[0]])
        dist += matriz_ady[recor[len(recor)-1]][recor[0]]
        return dist


cidades = []
população = []   #população de Rotas
população_mem = []
pool = []
calculada = []
calculada_media = []
```

```
calculada_pior = []
calculada_mem = []
calculada_media_mem = []
calculada_pior_mem = []
lista_x =[]
lista_y =[]
piores =[] # Rotas que nunca mais devem ser usadas
AES_result = []
AES_ME_result = []

# Leitura das coordenadas das cidades de um arquivo .txt
arq = open("dados29.txt","r")
num_Cidades =  int(arq.readline())
NNO=np.zeros(num_Cidades)
x=np.zeros(num_Cidades)
y=np.zeros(num_Cidades)
i=0
k=0
for linha in arq:

    if i<num_Cidades:

        valores = linha.split()
        NNO[i]=valores[0]
        x[i]=valores[1]
        y[i]=valores[2]
        c = Cidade(NNO[i], x[i]/1000, y[i]/1000)
        cidades.append(c)
        i= i + 1
arq.close()



#Geração matriz adjacencia ponderada
m_ady = np.zeros((num_Cidades,num_Cidades), dtype=float)

for i in range(0, num_Cidades):
    for j in range(i+1, num_Cidades):
        dist = math.sqrt(pow(cidades[i].x - cidades[j].x,2) +
         pow(cidades[i].y - cidades[j].y,2))
        m_ady[i][j] = m_ady[j][i] = dist

#Geração da população inicial
pai = np.empty([num_Cidades], dtype = int)
for i in range(0,num_Cidades):
    pai[i] = i


def populaçãoInicial(TamanhoPopulação):
```

```
    for i in range(0,TamanhoPopulação):
        ar = np.array(pai, copy = True)
        np.random.shuffle(ar)
        R = Rota(ar, m_ady)
        população.append(R)
        população_mem.append(R)

#Ordena a população pela aptidão
def Ranking (população):
    população.sort(key = operator.attrgetter('aptitud'))


def Memetico (população):
    piores.append(população[-memetico_size])

#Crossover
def Crossover (pai, pai2):
    cad_pai = []
    cad_pai2 = []

    filho = np.empty([num_Cidades], dtype = int)
    mascara = np.zeros([num_Cidades], dtype = bool)

    gen1 = randint(0, num_Cidades-1)
    gen2 = randint(0, num_Cidades-1)

    inicio = min(gen1,gen2)
    fim = max(gen1,gen2)

    for i in range(inicio,fim):
        cad_pai.append(pai[i])
        mascara[i] = 1

    for item in pai2:
        if not item in cad_pai:
            cad_pai2.append(item)

    i_m = 0
    i_p = 0
    for i in range(0,num_Cidades):
        if mascara[i]:
            filho[i] = cad_pai[i_p]
            i_p+=1
        else:
            filho[i] = cad_pai2[i_m]
            i_m+=1
    return filho

#Geração da população para reprodução
```

```
def GenPool (população):
    for i in range(0, pool_size+1):
        pool.append(população[i])

#Mutação
def Mutação(população):
    for i in range(0,Mutação_size):
        especimen = random.choice(população)
        gen1 = randint(0, num_Cidades-1)
        gen2 = randint(0, num_Cidades-1)
        especimen.sequencia[gen1], especimen.sequencia[gen2] = \
            especimen.sequencia[gen2], especimen.sequencia[gen1]


#Criação da próxima geração
def Geração (população):
    global AES
    #Conservar a população com elitismo
    for i in range(elitism_size, TamanhoPopulação):
        while (True):
            candidato = Rota(Crossover(random.choice(pool).sequencia,
            random.choice(pool).sequencia), m_ady)
            AES = (AES + 1)
            try :
                piores.index(candidato)
            except:
                população[i] = candidato
                break
    return AES

def Geração_me (população):
    global AES_ME
    #Conservar a população com elitismo
    for i in range(elitism_size, TamanhoPopulação):
        while (True):
            candidato = Rota(Crossover(random.choice(pool).sequencia,
            random.choice(pool).sequencia), m_ady)
            AES_ME = (AES_ME + 1)
            try :
                piores.index(candidato)
            except:
                população[i] = candidato
                break
    return AES_ME

def AlgGenetico():
    Ranking(população)
    for i in range(0,num_Geraçãoes):
        GenPool(população)
```

```
        Geração(população)
        Mutação(população)
        Ranking(população)
        calculada.append(população[0].aptitud)



def AlgMemetico():
    Ranking(população_mem)
    for i in range(0,num_Geraçãoes):
        GenPool(população_mem)
        Geração_me(população_mem)
        Mutação(população_mem)
        Ranking(população_mem)
        calculada_mem.append(população_mem[0].aptitud)
        Memetico(população_mem)


def PontosPercorridos(camino, cor):
    for i in range(0,len(camino)-1):
        x1 = cidades[camino[i]].x
        y1 = cidades[camino[i]].y
        x2 = cidades[camino[i+1]].x
        y2 = cidades[camino[i+1]].y
        plt.plot([x1,x2],[y1,y2], cor)
    x1 = cidades[camino[len(camino)-1]].x
    y1 = cidades[camino[len(camino)-1]].y
    x2 = cidades[camino[0]].x
    y2 = cidades[camino[0]].y
    plt.plot([x1,x2],[y1,y2], cor)

def Plot(calculada, pob, titulo):
    plt.title(titulo)
    plt.plot(calculada,label='Melhor')
    # plt.plot(calculada_media,label='Média')
    # plt.plot(calculada_pior,label='Pior')
    plt.legend(loc='upper right', frameon=False)
    plt.grid()
    plt.show()
    for i in range(0, num_Cidades):
        lista_x.append(cidades[i].x)
        lista_y.append(cidades[i].y)
    plt.plot(lista_x,lista_y, 'ro')
    for i in range(len(lista_x)):
        x, y = lista_x[i], lista_y[i]
        plt.annotate(xy=(x, y), s="{0}".format(cidades[i].id))
    PontosPercorridos(pob[0].sequencia, 'k-')
    plt.title(titulo)
```

```
    plt.show()
    lista_x.clear()
    lista_y.clear()

hist_GA = []
hist_ME = []
TamanhoPopulação = [80]
for TamanhoPopulação in TamanhoPopulação:
    num_Geraçãoes = [1500]
    for num_Geraçãoes in num_Geraçãoes:
        taxa_crossover = [0.3,0.1,0.8]
        for taxa_crossover in taxa_crossover:
            Mutação_pc = [0.001,0.1,0.4,0.01]
            for Mutação_pc in Mutação_pc:
                elitism_pc = [0.2]
                for elitism_pc in elitism_pc:
                    SR_SGA = 0
                    SR_ME = 0
                    num_execuções = 1
                    MBF_SGA = np.zeros(num_execuções)
                    MBF_ME = np.zeros(num_execuções)
                    AES = 0
                    for i in range(num_execuções):
                        memetico_pc = 0.2
                        elitism_size = int(TamanhoPopulação * elitism_pc)
                        pool_size = int(TamanhoPopulação * taxa_crossover
)
                        Mutação_size = int(TamanhoPopulação* Mutação_pc)
                        memetico_size = int(TamanhoPopulação* memetico_pc
)
                        populaçãoInicial(TamanhoPopulação)
                        AES = num_Cidades #número de chamadas para a popu
laçao inicial
                        AES_ME = num_Cidades #número de chamadas para a p
opulaçao inicial
                        AlgGenetico()
                        AlgMemetico()
                        # print(AES)
                        # print(AES_ME)
                        hist_GA.append(população[0].aptitud)
                        hist_ME.append(população_mem[0].aptitud)
                        Ranking(população)
                        # print ("Rota: ", população[0].sequencia)
                        if np.abs(1-
população[0].aptitud/9.0741)<0.05: ##SUCESSO
                            SR_SGA = SR_SGA + 1
                            AES_result.append(AES)
                        if np.abs(1-
população_mem[0].aptitud/9.0741)<0.05:
```

```
                            SR_ME = SR_ME + 1
                            AES_ME_result.append(AES_ME)
                        print("Iteração = ", i+1)
                        print ("Distancia percorrida SGA: ", população[0]
.aptitud)
                        print ("Distancia percorrida memetico: ", populaç
ão_mem[0].aptitud)
                        print("                    ")
                        MBF_SGA[i] = população[0].aptitud
                        MBF_ME[i] = população_mem[0].aptitud


                        Plot(calculada, população, "Algoritmo Genético")
                        Plot(calculada_mem, população_mem, "Algoritmo Mem
etico")
                        população.clear()
                        população_mem.clear()
                        calculada_mem.clear()
                        calculada_pior_mem.clear()
                        calculada.clear()
                        calculada_media.clear()
                        calculada_pior.clear()
                        lista_x.clear()
                        lista_y.clear()
                        pool.clear()

                    SR_SGA = SR_SGA/100
                    SR_ME = SR_ME/100

                    MBF__SGA = np.mean(MBF_SGA)
                    MBF__ME = np.mean(MBF_ME)

                    print("SR_GA = " + str(SR_SGA))
                    print("SR_ME = " + str(SR_ME))

                    print("MBF__GA = " + str(MBF__SGA))
                    print("MBF__ME = " + str(MBF__ME))

                    print("AES_GA = " + str(np.mean(AES_result)))
                    print("AES_ME = " + str(np.mean(AES_ME_result)))


fim = datetime.datetime.now()
print ("Tempo de execução: ", fim-inicio)
plt.hist(hist_GA)
plt.title("Algoritmo Genético")
plt.show()
plt.hist(hist_ME)
plt.title("Algoritmo Memético")
```

```
plt.show()
```

Conjunto de dados no arquivo .txt (nº de cidades e coordenadas *x,y* de cada cidade)

29

1 1150.0 1760.0

2 630.0 1660.0

3 40.0 2090.0

4 750.0 1100.0

5 750.0 2030.0

6 1030.0 2070.0

7 1650.0 650.0

8 1490.0 1630.0

9 790.0 2260.0

10 710.0 1310.0

11 840.0 550.0

12 1170.0 2300.0

13 970.0 1340.0

14 510.0 700.0

15 750.0 900.0

16 1280.0 1200.0

17 230.0 590.0

18 460.0 860.0

19 1040.0 950.0

20 590.0 1390.0

21 830.0 1770.0

22 490.0 500.0

23 1840.0 1240.0

24 1260.0 1500.0

25 1280.0 790.0

26 490.0 2130.0

27 1460.0 1420.0

28 1260.0 1910.0

29 360.0 1980.0