ANEXO 1 – Código implementado

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 10 12:17:33 2020
@author: Amanda
"""
## importação das bibliotecas
import random, numpy as np, math, matplotlib.pyplot as plt
import datetime

## Função de perturbação discreta da rota
def perturbação(rota):
    n_cidades = np.size(rota)
    [i,j] = sorted(random.sample(range(n_cidades),2));
    novaRota =  rota[:i] + rota[j:j+1] +  rota[i+1:j] + rota[i:i+1] + rota[j+1:];
    return novaRota

## Função Custo (calcula a distância entre todos os pontos da rota)
def J(rota,cidades):
    distancia = 0
    n_cidades = np.size(rota)
    for i in range (n_cidades):
        if i == (n_cidades-1):
            distancia +=  math.sqrt((cidades[rota[i]][0] - cidades[rota[0]][0])**2 +
 (cidades[rota[i]][1] - cidades[rota[0]][1])**2)
        else:
            distancia +=  math.sqrt((cidades[rota[i]][0] - cidades[rota[i+1]][0])**2
 + (cidades[rota[i]][1] - cidades[rota[i+1]][1])**2)
    return distancia

n_cidades = 29  ## Problema com 29 cidades
a = np.array([[1.1500,1.7600], ## Coordenadas (x,y) de cada cidade
  [   0.6300   ,1.6600],  [   0.0400   ,2.0900],  [   0.7500   ,1.1000],
  [   0.7500   ,2.0300],  [   1.0300   ,2.0700],  [    1.6500  ,0.6500],
  [    1.4900  ,1.6300],  [    0.7900  ,2.2600],  [    0.7100  ,1.3100],
  [    0.8400  ,0.5500],  [    1.1700  ,2.3000],  [    0.9700  ,1.3400],
  [    0.5100  ,0.7000],  [    0.7500  ,0.9000],  [    1.2800  ,1.2000],
  [    0.2300  ,0.5900],  [    0.4600  ,0.8600],  [    1.0400  ,0.9500],
  [    0.5900  ,1.3900],  [    0.8300  ,1.7700],  [    0.4900  ,0.5000],
  [    1.8400  ,1.2400],  [    1.2600  ,1.5000],  [    1.2800  ,0.7900],
  [    0.4900  ,2.1300],  [    1.4600  ,1.4200],  [    1.2600  ,1.9100],
  [    0.3600  ,1.9800]])

cidades = a.tolist() #Transforma o conjunto de coordenadas em lista
rota = random.sample(range(n_cidades),n_cidades) #rota inicial aleatória
N=int(1e5); K=150;  T0=0.5 #Parâmetros
fim=0; n=0; k=0; Jmin=J(rota,cidades); X = Xmin = rota; T=T0;
history_J=np.zeros([int(N*K),1]); history_T=np.zeros([int(N*K),1])
now0 = datetime.datetime.now() #Salva o horário de início da execução
```

```
print (now0.strftime("%Y-%m-%d %H:%M:%S"))
## Simulate Annealing
while not(fim):
    T=T0/np.log2(2+k)  #Resfriamento do SA clássico
    #T=T0/(k+1)        #Resfriamento do FSA
    for n in range(0,N):
        Xhat = perturbação(X)
        JX=J(X,cidades); JXhat=J(Xhat,cidades)
        if np.random.uniform()<np.exp((JX-JXhat)/T):
            X=Xhat; JX=JXhat
            if JX<Jmin:
                Jmin=JX; Xmin=X;
        history_J[k*N+n]=JX
        history_T[k*N+n]=T
        if np.remainder(n+1,1000)==0:
            print([k,n+1,Jmin])
    k+=1
    if k==K: fim=1

print(Jmin)
plt.rc('font',size=12,weight='bold')  ## Plota a rota ótima encontrada
plt.plot([cidades[Xmin[i % n_cidades]][0] for i in range(n_cidades+1)], [cidades[Xmi
n[i % n_cidades]][1] for i in range(n_cidades+1)], 'xb-');
titulo = 'SA: Rota para N=' +  str(N)  + ', K = ' + str(K) +', T0 = '+ str(T0)
plt.title(titulo)
plt.show()

print ("Inicio" )#Imprimi na tela a data e os hórarios de início e final da execução
print(now0.strftime("%Y-%m-%d %H:%M:%S"))
now = datetime.datetime.now()
print (now.strftime("%Y-%m-%d %H:%M:%S"))
#Plota o gráfico do histórico do da função custo (J) e da temperatura (T) ao longo
das iteração
plt.rc('font',size=12,weight='bold')
plt.figure()
plt.subplot(211)
plt.xlabel('Nº iterações')
plt.ylabel('Custo J')
plt.plot(history_J)
plt.title(titulo)
plt.grid()
plt.subplot(212)
plt.xlabel('Nº iterações')
plt.ylabel('T')
plt.plot(history_T)
plt.grid()
```