```
import random
import numpy as np

e = 0.036 #Choisir une valeur faible pour les premiers runs puis adapter en fonction
          #de la convergence de la solution
          #ici la meilleure prÃ©cision est de 0,036

A = [i for i in range(1,99)]
for i in range(len(A)):
   A[i]=float(A[i])/100
B = [i for i in range(21)]
C = [i for i in range(21)]
ensembles = [A, B, C]
I = []
f = open(r'C:\Users\Elie\Downloads\temperature_sample.csv') #Chemin d'accÃ¨s au fichier Ã  Ã©tudier
z=0
for line in f:
   ligne = line.split(';')
   if(z!=0):
      ajout = [float(ligne[0]),float(ligne[1])]
      I.append(ajout)
   z=z+1
f.close()

class individu:
   def __init__(self, val=None):
      if val==None:
         self.val=[None,None,None]
         self.val[0],self.val[1],self.val[2] = random.choice(A), random.choice(B), random.choice(C)
      else:
         self.val = val
      self.distance = self.fitness()
   def __str__(self):
      return str(self.val) + str(self.distance)
   def fitness(self):
      self.distance = 0
      for i in I:
         theo = 0
         for n in range(self.val[2]+1):
            theo = theo + (self.val[0]**n)*np.cos((self.val[1]**n)*np.pi*i[0])
         self.distance = self.distance + np.abs(theo-i[1])
      self.distance = self.distance / (len(I)+1)
      return self.distance

def create_rand_pop(count):
   pop = []
   for i in range(count):
      pop.append(individu())
   return pop

def evaluate(pop):
   return sorted(pop, key = lambda individu:individu.distance)

def selection(pop, hcount, lcount):
   return pop[:hcount],pop[-lcount:]

def croisement(ind1,ind2):
```

```
    ind1c, ind2c = individu([ind1.val[0],ind2.val[1],ind1.val[2]]), individu([ind2.val[0],ind1.val[1],ind2.val[2]])
    return ind1c, ind2c

def mutation(ind):
    i,j = random.randint(0,2), random.randint(0,2)
    while(i==j):
        i,j = random.randint(0,2), random.randint(0,2)
    ind.val[i], ind.val[j] = random.choice(ensembles[i]), random.choice(ensembles[j])
    return ind

def AlgoopSimple():
    pop = create_rand_pop(250)
    solutiontrouvee = False
    nbiteration = 0
    while not solutiontrouvee:
        print("Iteration : ",nbiteration)
        evaluation = evaluate(pop)
        if evaluation[0].distance<=e:
            solutiontrouvee = True
        else:
            select = selection(evaluation,100,40)
            croises = []
            for i in range(0,len(select),2):
                croises+=croisement(select[1][i],select[1][i+1])
            mutes=[]
            for i in select:
                for j in i:
                    mutes.append(mutation(j))
            newalea = create_rand_pop(50)
            pop = select[0]+select[1]+croises+mutes+newalea
            nbiteration = nbiteration + 1
        print(evaluation[0])

if __name__ == '__main__' :
    AlgoopSimple()
```