RAPPORT NICOLAS VANDERSTIGEL

1)Notre espace de recherche est :]0,1[x[1,20]x[1,20]. On recherche le triplet a,b,c qui représente le mieux notre fonction.

2)Ma fonction fitness :

```
def Cout(self): #On charge le fichier excel donné
    cout = 0
    file = np.loadtxt('temperature_sample_calibrate2.csv', delimiter = ';', skiprows=1)
    for i in range(len([row[0] for row in file])): #on navigue dans la matrice en fonction du nombre de lignes dans celle-ci
        a = self.a
        b = self.b
        c = self.c
        somme = 0
        for n in range(c+1):
            somme += (a**n) * math.cos((b**n)* (math.pi) * file[i][0])#calcul de la fonction pour notre triplet
        cout += abs(somme - file[i][1]) #on compare cette fonction avec les valeurs données
    return cout
```

Ici je suis dans ma class « Triplet », comme ça chaque triplet à un cout qui lui est attribué, j'ai trouvé cela plus pratique pour faire mes calculs et même lors de la construction du code. Cela me permet d'afficher directement le cout d'un triplet en faisiant : « monTriplet.cout ».

3) Ma fonction croisement :

```
def NCroisement(n): #On croise les triplets entre 2 triplets pour en former 2 nouveaux
    compteur = 0
    while(compteur<n):
        triplet1 = random.randint(0, len(listTriplet) - 1)#On genere une position aleatoire pour recuperer le triplet
        triplet2 = random.randint(0, len(listTriplet) - 1)
        nouveauTriplet1 = classTriplet(listTriplet[triplet1].a,listTriplet[triplet2].b, listTriplet[triplet2].c)
        nouveauTriplet2 = classTriplet(listTriplet[triplet2].a,listTriplet[triplet1].b, listTriplet[triplet2].c)
        listTriplet.append(nouveauTriplet1)
        listTriplet.append(nouveauTriplet2)
        compteur+=1
```

Ici je sélectionne aléatoirement deux triplets dans ma liste de triplets. Je crée ensuite 2 nouveaux triplets en croisant les valeurs de mes 2 triplets initiaux. Cela me permet d'avoir 2 nouveaux individus formés à partir de 2 anciens triplets, ce qui est très intéressant pour affiner au maximum le résultat. J'ai décidé de répéter le processus n fois, que je peux changer dans mon Main sans soucis. J'ai par défaut choisis 40, afin d'avoir 80 croisements issus des 20 meilleurs triplets, pour me ramener à une population de 100 triplets. Selon moi c'était suffisant pour trouver les bons résultats.

Ma fonction mutation :

```
def Mutation(positionTriplet): #on va faire muter une valeur dans un triplet aleatoire
    varriableTriplet = random.randint(0,2)
    a = listTriplet[positionTriplet].a
    b = listTriplet[positionTriplet].b #On modifie aleatoirement une des variables de notre triplet
    c = listTriplet[positionTriplet].c
    if varriableTriplet == 0:
        a = round(random.uniform(0,1),4)
    if varriableTriplet == 1:
        b = random.randint(1,20)
    else:
        c = random.randint(1,20)
    triplet = classTriplet(a, b, c)
    del listTriplet[positionTriplet]
    return triplet

def NMutation(): #On effectue la mutation N fois
    compteur = 0
    while(compteur<10):
        positionAleatoire = random.randint(20, len(listTriplet) - 1)
        listTriplet.append(Mutation(positionAleatoire))
        compteur+=1
```

La mutation se décompose en 2 parties distinctes. Premièrement dans « NMutation » je viens selectionné un triplet au hasard parmis les moins bons (après le 20^{ième}) et je vais rajouter dans ma liste de triplet la mutation définie dans « Mutation ». J'ai choisis de piocher au hasard une des 3 varibales de mon triplet, et de modifier totalement sa valeur (j'utilise à nouveau un random). Ensuite je viens créer un triplet « muté » avec les anciennes variables et la nouvelle qui est modifiée. Je supprime le triplet que nous avons utilisé pour la mutation et je return notre nouveau triplet muté. A nouveau dans NMutation je viens ajouter ce triplet dans ma liste et je répéte ce processus 10 fois. J'ai décidé de travailler avec les populations nouvellement formé par le croisement (donc les 80 nouvelles, cf.Fonction Croisement) afin de maximiser les chances de converger vers le bon triplet. En effet, si dès le début toutes mes valeurs sont très loins du bon résultat, les mélanger ne changera rien. Alors que le fait de rajouter 10 mutations parmis les 80 nouveaux triplets, cela permet de rajouter un maximum de diversité et donc de possiblité de trouver le bon résultat.

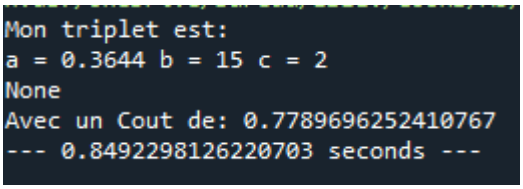
4) Mon processu de sélection :

```
listTriplet.sort(key=lambda x: x.cout)
listParent = listTriplet[:20].copy()
listTriplet = list(listParent)
listParent = []
```

Je viens donc trier ma liste, en fonction du cout (Cf.Cout au dessus), donc chaque triplet est trié par ordre croissant en fonction de son cout. Ensuite on stocke les 20 premiers triplets avec le plus petit cout dans une liste temporaire. Et notre liste initiale de 100 individus devient donc une liste composée des 20 individus avec le plus faible cout. Ensuite nous allons croiser ces 20 individus entre eux et faire muter ces croisements afin de diversifier au maximum.

5) Je génére une population de 100 triplets au début de mon programme. Au bout de 7 génértions je tombe vers une réponse toujours stable. Sauf pour mon a pusique j'ai indiqué 4 chiffres après la virgule pour être le plus précis possible, mais si je choisis d'en afficher 2 alors tout est stable)

6) En moyenne je tourne autour des 1 secondes. (voir moins) Je suis assez content de la vitesse d'execution, j'ai longtemps essayer d'optimiser mon code. J'ai par ailleurs réussis à rendre un code de moins de 100 lignes.



7) J'ai eu de mal avec l'importation de fichier. Je ne savais pas comment lire un fichier Excel, j'ai donc recherché sur internet afin de comprendre comment était rempli les fichiers excel.

Ensuite je n'avais pas créé de classe Triplet au début de mon programme et cela m'a posé problème lors du tri de mes triplets en fonction de leur cout. J'ai logntemps été bloqué sur ce point jusqu'à découvrir la fonction de tri (ou sélection) que j'ai détaillé plus haut qui est directement incluse dans les classes. Cela m'a fait gagné beaucoup de lignes de codes et a résolu bon nombre de mes problèmes.

Après, j'ai eu un soucis au niveau de mes mutations. Je faisais muter mes 20 meilleurs individus ce qui avait pour conséquence de brouiller tout mon algorithme, car cela pouvait faire muter un triplet très proche de la solution. J'ai donc décidé de ne faire muter que mes 80 nouveaux triplets.

Enfin, j'ai d'abord voulu utiliser des dictionnaires pour pouvoir utiliser la fonction « sorted » mais cela ne m'arrangeait pas d'avoir à la sortie une liste. J'ai été bloqué sur ce problème un certain temps puisque je manipulais à la fois des dictionnaires et des listes. Je savais bien que cela n'était pas du tout optimal et je devais m'en débarasser pour avoir un code le plus efficace et ergonomique possible.