I21 - Algorithmique élémentaire
TP 2

Année universitaire 2018/19

Le module matplotlib

Dans les exercices suivants, on fera appel au module `matplotlib`. Ce dernier propose quelques fonctions permettant de tracer simplement des courbes (et bien d'autres choses). Nous nous contenterons ici d'une utilisation relativement simple.

Premières courbes avec `matplotlib`

1. Recopier le code suivant et exécuter le:

```
import matplotlib.pyplot as plt

# On cree la liste des abscisses des point que l'on veut tracer
abscisse = []
for i in range(10):
  abscisse += [i]

# On cree la liste des ordonnees des points que l'on veut tracer:
carre = []
for i in range(len(abscisse)):
  carre += [abscisse[i]**2]

cube = []
for i in range(len(abscisse)):
  cube += [abscisse[i]**3]

# plot() trace les courbes correspondantes et show() les affiche
# a l'ecran.
plt.plot(abscisse, carre, abscisse, cube)
plt.show()
```

2. Écrire une fonction `intervalle(a,b,inc)` qui retourne sous forme de liste les valeurs comprises entre `a` et +`b` avec un incrément de `inc`. Par exemple

```
>>> intervalle(0,1,0.2)
 [0, 0.2, 0.4, 0.6, 0.8, 1]
```

3. Écrire deux fonctions `valeur_sin(t)` et `valeur_cos(t)` qui retourne la liste des valeurs prises par les fonctions python du module `math` `sin` et `cos` aux points donnés par la liste `t`. Par exemple

```
>>> valeur_sin([0,0.2,0.4,0.8,1])
 [0.0, 0.19866933079506122, 0.3894183423086505, 0.5646424733950354,
  0.7173560908995228, 0.8414709848078965]
```

4. Tracer les courbes des fonctions `sin` et `cos` entre -2π et 2π.

Étude de la multiplication en Python

1. Écrire une fonction `multiplication(n1,n2)` qui retourne le produit des nombres `n1` et `n2` donnés sous la forme de tableau de nombres entre 0 et 9. Par exemple le nombre 6704 sera représenté par `[4,0,7,6]`.

2. Pour mesurer la vitesse d'exécution d'un algorithme en Python on peut utiliser la fonction `time()` du module `time`. Le code suivant permet par exemple de mesurer le temps d'exécution de 10000 multiplications de nombres de 20 chiffres en Python.

```
from time import time
from random import randrange

lim = 10**20
# on cree une liste contenant des paires de nombres aleatoires
test = []
for i in range(10000):
  n1=randrange(lim)
  n2=randrange(lim)
  test+=[(n1,n2)]

# on stocke l'heure de depart
start=time()
for n1,n2 in test:
  n3=n1*n2

# on recupere l'heure de fin et on affiche le resultat
end=time()
print(end-start)
```

   Écrire une fonction `time_mult_py(n,k)` qui retourne le temps d'exécution de k multiplications de nombres de n chiffres. Par exemple le programme précédent correspond à l'appel de fonction `time_mult_py(20,10000)`.

3. Écrire une fonction `nombre_alea(n)` qui retourne un nombre de n chiffres sous forme de tableau. Par exemple

```
>>> nombre_alea(5)
 [0,5,4,8,1]
>>> nombre_alea(5)
 [5,4,3,3,7]
```

4. Écrire une fonction `time_my_mult(n,k)` qui retourne le temps d'exécution de k multiplications de nombres de n chiffres en utilisant la fonction `multiplication(n1,n2)`. Mesurer le temps de calcul de 100 multiplications pour plusieurs valeurs de n en le doublant à chaque fois. Comment évolue ce temps d'exécution ? Que peut en déduire sur la complexité asymptotique de votre algorithme ?

5. Tracer les courbes correspondant au temps d'exécution de chacune des multiplications en fonction de la taille des entrées. On considérera successivement des entiers de taille n où n parcours les multiples de 64 jusqu'à 512. Pour chaque taille on mesurera le temps d'exécution pour 100000 exécutions de l'algorithme de multiplication de Python et de seulement 10 exécutions de votre fonction `multiplication(n1,n2)`. Les deux algorithmes semblent ils équivalents asymptotiquement ?

Étude de l'instruction '+' sur les listes

Il y a en Python trois manières d'ajouter un élément à la fin d'une liste: deux utilisant l'opérateur `'+'` et une utilisant la méthode `append` (on admettra le fonctionnement des méthodes sans trop chercher plus loin, il s'agit d'un de programmation objet qui seront vus en 3ème année).

```
L = list(range(1,10,2))
print(L)
L = L+[11]
print(L)
L += [1212]
print(L)
L.append(13)
print(L)
```

1. Écrire trois fonctions `time_plus(n,k)`, `time_inc(n,k)` et `time_append(n,k)` qui retourne le temps d'exécution de l'ajout de k éléments successivement dans une liste de taille n à l'aide de chacune des trois manières précédentes.
2. Tracer les courbes des temps d'exécution des trois opérateurs pour l'ensemble des tailles de tableau suivantes pour des ajouts de 10 éléménts: [256,512,1024,2048,4096,8192].
3. En déduire la complexité de chacun des trois opérateurs.

Fonctions `min` et `max`

Python fournit deux fonctions `min` et `max` qui retournent respectivement le plus petit et le plus grand élément d'une liste.

1. Écrire une fonction `list_alea(n,k)` qui renvoie une liste de taille n contenant des entiers aleatoires compris entre 0 et k (exclu).
2. Écrire deux fonctions `time_min(n,k)` et `time_max(n,k)` qui retourne le temps d'exécution des fonctions `min` et `max` sur des tableaux de taille n contenant des entiers aleatoires compris entre 0 et k.
3. Tracer les courbes des temps d'exécution de deux fonctions pour l'ensemble des tailles de tableau suivantes: [256,512,1024,2048,4096,8192]. La valeur de k change t'elle les résultats ?
4. En déduire la complexité de chacune des deux fonctions.

Opérateur `in` sur les listes et les dictionnaires

L'opérateur `in` permet de savoir si un élément appartient à une structure de données.

```
L = list(range(1,10,2))
print(L)
print(3 in L)

D = {'toto':0,'titi':-5,'tutu':12}
print(D)
print('tata' in D)
```

1. Écrire une fonction `list_alea(n)` qui renvoie une liste de taille n contenant des entiers aleatoires compris entre 0 et n (exclu).
2. Écrire une fonction `dict_alea(n)` qui renvoie un dictionnaire de n éléments dont les clés sont tirées aléatoirement.
3. Tracer les courbes des temps d'exécution de l'opérateur `in` pour l'ensemble des tailles de tableau et de dictionnaire suivantes: [256,512,1024,2048,4096,8192].
4. En déduire la complexité de l'opérateur sur les tableaux et les dicotionnaires.