Séance 09 :

Récursivité

1. Présentation

C.F. Sujet

2. PGCD

Donner une fonction récursive qui calcule le pgcd de deux nombres entiers.

```python
def pgcd(a, b):
    if (a < b):
        return pgcd(a, b)
    r = a % b
    if (r == 0):
        return b
    return pgcd(b, r)
```

3. Suite de Fibonacci

On définit la suite de Fibonacci par :

$$\begin{cases} f(0) = f(1) = 1 \\ f(n+2) = f(n+1) + f(n) \end{cases}$$

(a) ***Donner une fonction*** `firec(n)` ***qui calcule par récursivité*** $f(n)$.

```python
def firec(n):
    if (n == 0 or n == -1):
        return 1
    return firec(n-1) + firec(n-2)
```

(b) ***Vérifier que le temps de calcul pour calculer*** $f(n)$ ***avec*** $n \leq 30$ ***est :***

$$T_{recursif} = \alpha\varphi^n \text{ où } \varphi = \frac{1+\sqrt{5}}{2} \simeq 1.618$$

Pour estimer les temps de calcul on utilisera la fonction `time()` de la bibliothèque `time`.

- Je commence par importer les librairies habituelles, et je déclare la variable `phi` :

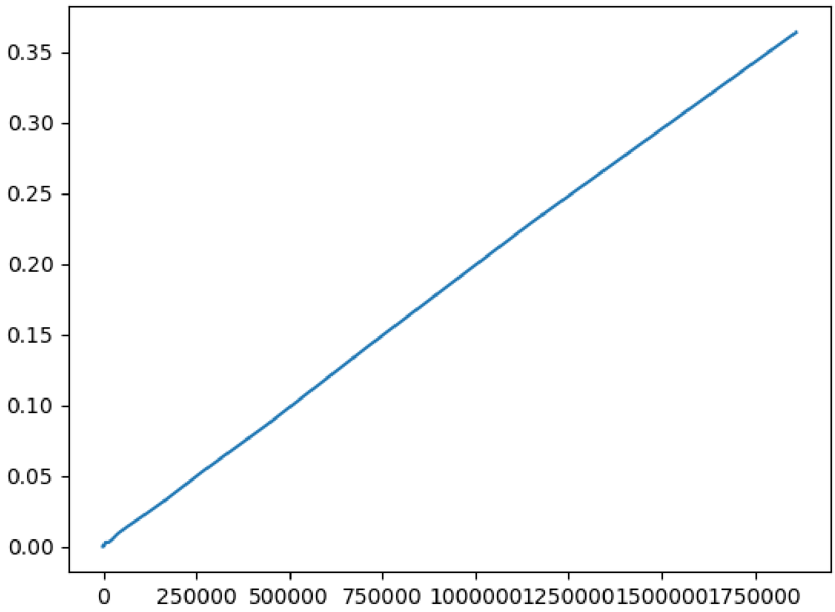
```python
import numpy as np
import matplotlib.pylab as plt
import time as t

phi = (1 + np.sqrt(5)) / 2
```

- Puis je crée une petite fonction pour mesurer le temps de calcul en fonction de n :

```python
def firecTime(n):
    T = []
    X = []
    for i in range(1, n+1):
        t1 = t.time()
        firec(i)
        T.append(t.time() - t1)
        X.append(i)
    return (X, T)
```

❏ Et je trace le temps en fonction de φ^n :

```
def traceTimeFirect(n):
    X, T = firecTime(n)
    Xp = [phi ** k for k in X]
    plt.plot(Xp, T)
    plt.show()
```



On remarque que c'est une droite de pente α, ce qui prouve que $T_{recursif} = \alpha\varphi^n$. Mais j'ai bien envie de calculer α.

❏ Je vais me servir de la méthode des moindres carrés :

```
def moindreCarre(n):
    X, T = firecTime(n)
    # On utilise numpy pour plus de simplicite
    X = np.array(X)
    T = np.array(T)

    # On linearise
    X = phi ** X

    # On calcule les variables des moindres carres
    n = X.shape[0]

    Sx = np.sum(X)
    St = np.sum(T)
    Sxt = np.sum(X * T)
    Sx2 = np.sum(X**2)

    # Formule des moindres carres
    a = (n*Sxt-Sx*St)/(n * Sx2 - Sx**2)
    b = (St - a * Sx) / n

    return a, b
```

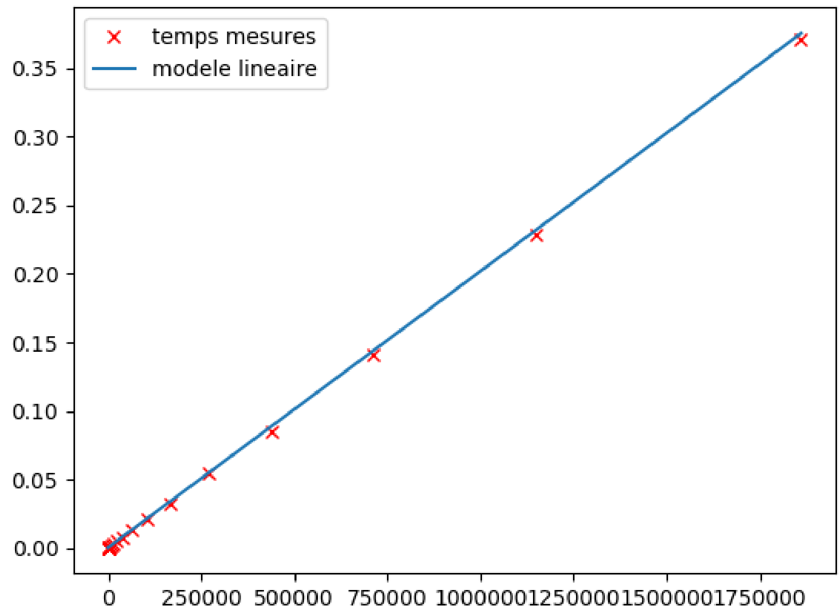
❏ Puis, pour avoir plus de précision, je répète `m` fois l'expérience, et je calcule la moyenne :

```
def mesureAB(n, m):
    a, b = np.zeros(m), np.zeros(m)
    for i in range(m):
        # On calcule les parametres a et b
        a[i], b[i] = moindreCarre(n)

        # On affiche le pourcentage fait
        print(f'{(i+1)/m*100:.2f}%')

    # On calcule la moyenne des parametres
    a = np.average(a)
    b = np.average(b)

    # On les affiche
    print(a, b)

    # On calcule les temps
    X, T = firecTime(n)

    # On linearise
    X = phi ** X

    # On calcule les points de la droite
    Y = a*X + b

    # On affiche tout ca
    plt.plot(X, T, 'rx', label='temps mesures')
    plt.plot(X, Y, label='modele lineaire')
    plt.legend()
    plt.show()
```

❏ Ainsi, en exécutant `mesureAB(30, 40)`, j'obtiens :

`a` = `2.0165040404262714e-07`
`b` = `0.00035222694980754904`

avec ce graphique :



(c) ***Expliquer ce temps de calcul.***
Etudions la complexité de `firec` :
Posons $C(n)$ le nombre d'additions à l'étape n.
Alors, la ligne 4 de `firec` donne la relation suivante sur $C(n)$:

$$C(n) = C(n-1) + C(n-2) + 1$$

Or, si cherche le terme général de cette suite d'ordre 2, on trouve :

$$C(n) = \lambda\left(\frac{1+\sqrt{5}}{2}\right)^n + \mu\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Ainsi, on retrouve une complexité de l'ordre de $T_{recursif} = \alpha\varphi^n$

(d) ***Donner la fonction*** `fibo(n)` ***pour calculer par itération*** $f(n)$ ***pour*** $n \leq 10000$***.***

```
def fibo(n):
    a, b = 1, 1
    for i in range(n):
        c = a + b
        b = a
        a = c
    return c
```

(e) ***Donner la fonction*** `firecrap(n)` ***qui calcule par récursivité*** $f(n)$ ***avec une complexité de n.***
J'avoue, c'est le prof qu'a donné la solution : On va boucler sur une liste contenant les deux derniers termes de la suite : `L = [f(n-2), f(n-1)]`

```
def firecrapB(n):
    if (n == 1):
        return ([1, 1])
    l = firecrapB(n-1)
    return [l[1], sum(l)]
```

Sauf qu'on veut renvoyer le dernier terme est pas une liste des 2 derniers termes, d'où :

```
def firecrap(n):
    A = firecrapB(n)
    return (A[0] + A[1])
```

4. Somme de Fibonacci

(a) ***Calculer par récursivité la somme*** $\sum_{k=0}^{n} \frac{1}{f(k)}$ ***et vérifier sa limite*** `L = 3.359885666243178` ***en estimant la vitesse de convergence.***
On applique le même principe que précédemment, en stockant à la fois les deux derniers termes de la suite de Fibonacci, et en stockant la somme de l'inverse de ces deux derniers termes :

```
def firecrapB(n):
    if (n == 1):
        return ([
            [1, 1],
            [0, 1]
        ])
    l = firecrapB(n-1)
    return [
        [l[0][1], sum(l[0])],
        [1/l[0][1], sum(l[1])]
    ]
```

```python
def firecrap(n):
    A = firecrapB(n)
    return (A[1][0] + A[1][1])
```

On trouve la même limite, et la vitesse de convergence est linéaire (complexité en n).

5. Triangle de Pascal

(a) ***Construire une fonction*** `pascal(n, p)` ***qui renvoie le triangle de pascal de*** $(n+1)$ ***lignes et*** $(p+1)$ ***colonnes par récursivité en utilisant des listes, et la méthode*** `extend` ***sur les listes.***

```python
def pascal(n, p):
    if (n == 0):
        return [[1] + [0]*p]
    t = pascal(n-1, p)
    newline = [[1] + [t[-1][k] + t[-1][k-1] for k in range(1, p+1)]]
    t.extend(newline)
    return t
```

(b) ***Afficher le triangle sous la forme d'une pyramide.***

```python
def Triangle(n):
    # On recupere les lignes du triangle de Pascal
    P = pascal(n, n)
    LignesTemp = []
    # Mettre en forme 1 par 1
    for ligne in P:
        # On commence par suppimer tous les zeros
        while (0 in ligne):
            ligne.remove(0)

        # On transforme les listes en chaine de caracteres
        a = ''
        for num in ligne:
            a += f'{num} '
        LignesTemp.append(a)

    # On recupere la longueur de la derniere ligne (la plus longue)
    h = len(LignesTemp[n]) + 1

    # Maintenant on va rajouter les espaces en debut de lignes pour centrer
    for ligne in LignesTemp:
        L = len(ligne) + 1

        # Si la ligne est paire, on rajoute d'abord un underscore au milieu
        if (L % 2 == 0):
            ligne = ligne[:L//2-1] + '_' + ligne[L//2-1:]

        # Nombre d'espace a rajouter
        nbSpaces = (h - L) // 2

        # On rajoute les espaces
        ligne = ' ' * nbSpaces + ligne

        # On affiche la ligne
        print(ligne)
```

En exécutant `Tiangle(12)`, on obtient :

```
                     1
                    1  1
                   1  2  1
                  1  3  3  1
                 1  4  6  4  1
               1  5  10  10  5  1
             1  6  15  2_0  15  6  1
            1  7  21  35  35  21  7  1
          1  8  28  56  7_0  56  28  8  1
        1  9  36  84  126  126  84  36  9  1
    1  10  45  120  210  252  210  120  45  10  1
  1  11  55  165  330  462  462  330  165  55  11  1
1  12  66  220  495  792  924  792  495  220  66  12  1
```