TP 3

# Temps d'exécution et complexité : expérimentations avec la suite de Fibonacci

**Instructions**

Les questions marquées d'astérisques (* ou **) pourront être traitées après toutes les autres.

## 1 La suite de Fibonacci

On rappelle que la suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est donnée par

$$F_0 = 0 \qquad F_1 = 1 \qquad F_{n+2} = F_n + F_{n+1}$$

Voici quelques termes de la suite de Fibonacci :

$$0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad \dots$$

On considère dans un premier temps une implémentation Python récursive de la fonction qui à $n$ associe le terme $F_n$ de la suite de Fibonacci.

```
1 def fibrec(n) :
2     if n < 2 :
3         return n
4     return fibrec(n-1)+fibrec(n-2)
```

## 2 Temps d'exécution

Le fichier tp03utils.py contient des fonctions Python pour mesurer et représenter graphiquement des temps d'exécution de fonctions. Une fois correctement téléchargé, on pourra utiliser les fonctions de tp03utils.py dans des scripts avec la ligne suivante. [1]

```
from tp03utils import *
```

### 2.1 Mesurer des temps d'exécution

Dans ce §2.1, on s'intéresse uniquememt à la mesure du temps d'exécution d'un appel de fonction. On utilisera la bibliothèque time (importée par la ligne `import time`).

Les deux fonctions suivantes prennent en arguments une fonction Python `f` et une valeur `v`. Elles renvoient le temps d'exécution de l'appel à `f` sur l'argument `v`.

```
def time_fun(f,v) :
    start = time.process_time()
    f(v)
    end = time.process_time()
    return end-start
```

```
def time_fun_ns(f,v) :
    start = time.process_time_ns()
    f(v)
    end = time.process_time_ns()
    return end-start
```

La fonction `time_fun` renvoie un nombre à virgule flottante représentant des fractions de secondes. La fonction `time_fun_ns` renvoie un entier représentant des nanosecondes. Par exemple, le temps d'exécution de `fibrec(35)` peut être obtenu par la commande suivante.

1. Le fichier `tp03utils.py` est disponible à http://perso.ens-lyon.fr/colin.riba/teaching/cpes/tp/tp03utils.py.

```
>>> time_fun(fibrec,35)
```

Le nombre flottant renvoyé représente des fractions de secondes.

**Question 1.** *Utiliser la fonction* `time_fun_ns` *pour obtenir le temps d'exécution en nanosecondes de* `fibrec(35)`.

## 2.2 Listes de temps d'exécution

À partir de maintenant, on s'intéresse uniquement à des fonctions Python `f` qui prennent un argument entier. Nous allons chercher à obtenir la liste des temps d'exécution de `f(k)` pour $\texttt{k} = 0, \ldots, \texttt{N} - 1$, où `N` est un entier donné.

La première chose à faire est de déterminer une valeur de `N` raisonnable.

**Question 2.** *Écrire une fonction* Python `seuil` *qui prend en arguments une fonction* `f` *et un nombre* `s`. *Un appel à* `seuil(f,s)` *doit renvoyer le plus grand entier* `N` *tel que* `f(N)` *s'exécute en moins de* `s` *secondes.*

**Question 3.** *Utiliser la fonction* `seuil` *de la Question 2 pour déterminer le plus grand entier* `N` *tel que* `fibrec(N)` *s'exécute en moins de 10 secondes.*

**Dans toute la suite, l'entier `N` obtenu à la Question 3 sera noté `Nrec`.**

La fonction suivante renvoie la liste des temps d'exécution de `f(k)` pour $\texttt{k} = 0, \ldots, \texttt{n} - 1$. Les temps sont en fractions de secondes.

```
1 def list_times(f,n) :
2     y = list(range(n))
3     for i in range(n) :
4         y[i] = time_fun(f,i)
5     return y
```

**Question 4.** *Utiliser la fonction* `list_times` *pour obtenir la liste des temps d'exécution de* `fibrec(k)` *pour* $\texttt{k} = 0, \ldots, \texttt{Nrec}$. *Les temps doivent être en fractions de secondes.*

**Question 5.** *Écrire une fonction* Python `list_times_ns` *qui prend en arguments une fonction* `f` *est un entier* `n`. *Une appel à* `list_times_ns(f,n)` *doit renvoyer la liste des temps d'exécution de* `f(k)` *pour* $\texttt{k} = 0, \ldots, \texttt{n} - 1$. *Les temps doivent être en nanosecondes.*

**Question 6.** *Utiliser la fonction* `list_times_ns` *de la Question 5 pour obtenir la liste des temps d'exéction de* `fibrec(k)` *pour* $\texttt{k} = 0, \ldots, \texttt{Nrec}$. *Les temps doivent être en nanosecondes.*

## 2.3 Tracer des temps d'exécution

Nous allons maintenant tracer les temps d'exécution de `f(k)` en fonction de $\texttt{k} = 0, \ldots, \texttt{n} - 1$. Pour cela nous utilisons la bibliothèque matplotlib, importée par la ligne suivante. [2]

```
import matplotlib.pyplot as plt
```

La fonction suivante trace les temps d'exécution de `f(k)` en fonction de $\texttt{k} = 0, \ldots, \texttt{n} - 1$. Les temps sont en fractions de secondes et sont affichés en ordonnées.

2. La documentation de matplotlib est disponible sur https://matplotlib.org/stable/.

```
def plot_fun(f,n) :
    y = list_times(f,n)
    plt.xlabel('input')
    plt.ylabel('computation␣time␣(seconds)')
    plt.plot(range(n),y,'x')
```

Le dernier argument de `plt.plot` est le **marqueur** `'x'`, qui indique que les points du tracé doivent être représentés par des croix. Si on avait utilisé `'o'` au lieu de `'x'`, alors l'appel à `plot_fun` produirait des tracés dont les points sont représentés par des disques. [3]

**Attention.** Sous Spyder, les graphiques sont visualisables dans l'onglet « Graphes » (ou « Plots »). Avec d'autres environnements de développement, il peut être nécessaire d'exécuter la commande `plt.show()` dans la console.

**Question 7.** *Utiliser la fonction* `plot_fun` *pour tracer les temps d'exécution de* `fibrec(k)` *en fonction de* $\mathtt{k} = 0, \dots, \mathtt{Nrec}$.

**Question 8.** *Écrire une fonction* Python `plot_fun_ns(f,n)` *qui trace les temps d'exécution de* `f(k)` *en fonction de* $\mathtt{k} = 0, \dots, \mathtt{n} - 1$. *Les temps doivent être donnés en nanosecondes et être affichés en ordonnées.*

*Utiliser la fonction* `plot_fun_ns` *pour tracer les temps d'exécution de* `fibrec(k)` *en fonction de* $\mathtt{k} = 0, \dots, \mathtt{Nrec}$.

## 3 Retour sur Fibonacci : programmation dynamique

Au vu des Questions 7 et 8, on peut soupçonner que le temps d'exécution de `fibrec(n)` est exponentiel en `n`. En effet, il existe un nombre réel $\alpha > 1$ tel que pour tout entier `n` suffisemment grand, l'exécution de `fibrec(n)` effectue au moins $\alpha^{\mathtt{n}}$ opérations.

Il est en fait relativement facile d'écrire une fonction Python beaucoup plus efficace que `fibrec`. Rappelons le code de la fonction `fibrec` du §1 :

```
def fibrec(n) :
    if n < 2 :
        return n
    return fibrec(n-1)+fibrec(n-2)
```

Le problème avec la fonction `fibrec` est que lorsqu'on l'appelle sur une entrée `n` donnée, les valeurs de `fibrec(k)` pour $\mathtt{k} = 0, \dots, \mathtt{n} - 1$ sont en fait calculées plusieurs fois.

**Question* 9.** *Considérons les exécutions de* `fibrec(n)` *pour* $\mathtt{n} = 4$ *et* $\mathtt{n} = 5$. *Pour chaque* $\mathtt{k} \in \{0, \dots, \mathtt{n}\}$, *donner le nombre d'appels à* `fibrec(k)` *lors de l'exécution de* `fibrec(n)`.

**Question** 10.** *Donner un réel* $\alpha > 1$ *tel que pour tout* `n` *suffisemment grand,* `fibrec(n)` *effectue au moins* $\alpha^{\mathtt{n}}$ *opérations.* [4]

Une manière simple de calculer plus efficacement le terme $F_{\mathtt{n}}$ de la suite de Fibonacci est d'éviter de calculer plusieurs fois les valeurs des termes $F_0, \dots, F_{\mathtt{n}-1}$. Pour ce faire, on peut stocker les résultats de ces calculs intermédiaires (par exemple dans une liste). C'est ce que l'on appelle de la **programmation dynamique**.

**Question 11.** *Écrire une fonction* Python `fib(n)` *qui renvoie le terme* $F_{\mathtt{n}}$ *de la suite de Fibonacci.* **On attend un nombre d'additions linéaire en `n` (c'est-à-dire en** $O(\mathtt{n})$**),** *cf note 4.*

*Tester* `fib(n)` *sur de grandes valeurs de* `n`.

3. La liste de marqueurs possibles est disponible à https://matplotlib.org/stable/api/markers_api.html.
4. On pourra se baser sur https://wiki.python.org/moin/TimeComplexity.

On souhaiterait maintenant obtenir le plus grand entier `Ndyn` tel que `fib(Ndyn)` s'exécute en moins de 10 secondes.

**Question 12.** *Quel est le temps d'exécution de* `fib(2**10)` *?*

La Question 12 suggère d'exprimer `Ndyn` sous la forme d'une puissance de 2. On va chercher à obtenir le plus grand entier `n` tel que `fib(2**n)` s'exécute en moins de 10 secondes.

**Question 13.** *Écrire une fonction* Python `seuil_exp` *qui prend en arguments une fonction* `f` *et un nombre* `s`*. Un appel à* `seuil_exp(f,s)` *doit renvoyer le plus grand entier* `n` *tel que* `f(2**n)` *s'exécute en moins de* `s` *secondes.*

**Question 14.** *Utiliser la fonction* `seuil_exp` *de la Question 13 pour déterminer le plus grand entier* `n` *tel que* `fib(2**n)` *s'exécute en moins de 10 secondes.*

**On notera `Ndyn` l'entier $2^{\texttt{n}}$, où `n` est l'entier obtenu à la Question 14.**

**Question 15.** *Tracer les temps d'exécution de* `fib(k)` *en fonction de* $\texttt{k} = 0, \ldots, \texttt{N}$ *où* `N` *est un grand entier bien choisi.*

### 3.1 Tracés simultanés*

Il peut être utile de tracer simultanément les temps d'exécution de plusieurs fonctions. On va écrire et utiliser des fonctions Python qui prennent en arguments une liste de fonctions et une liste de marqueurs (voir §2.3). Notons qu'on peut construire la liste de fonctions `[fibrec,fib]`.

Considérons dans un premier temps la fonction suivante.

```
1  def plot_list(n,list_fun,list_mark) :
2      assert len(list_fun) == len(list_mark)
3      nbfun = len(list_fun)
4      a = [None]*(1+2*nbfun)
5      a[0] = range(n)
6      for i in range(nbfun) :
7          a[1+2*i] = list_times(list_fun[i],n)
8          a[1+2*i+1] = list_mark[i]
9      plt.xlabel('input')
10     plt.ylabel('computation␣time␣(seconds)')
11     plt.plot(*a)
```

La fonction `plot_list` prend en arguments un entier `n` ainsi que deux listes `list_fun` et `list_mark`. `list_fun` est une liste de fonctions et `list_mark` est une liste de marqueurs. Ces deux listes doivent être de même longueur, notée `nbfun`. Un appel `plot_list(n,lf,lm)` trace simultanément les temps d'exécution de chacune des fonctions `lf[i]` pour $\texttt{i} = 0, \ldots, \texttt{nbfun} - 1$. Les temps d'exécution de la fonction `lf[i]` sont représentés par le marqueur `lm[i]`. Pour chaque $\texttt{i} = 0, \ldots, \texttt{nbfun} - 1$, le temps d'exécution de `lf[i](k)` est tracé en fonction de $\texttt{k} = 0, \ldots, \texttt{n} - 1$. Les temps sont donnés en fractions de secondes et sont représentés en ordonnées.

**Question 16.** *Utiliser la fonction* `plot_list` *pour tracer simultanément les temps d'exécution de* `fibrec(k)` *et* `fib(k)` *en fonction de* $\texttt{k} = 0, \ldots, \texttt{Nrec}$*. Le tracé pour* `fibrec` *devra être représenté par des croix (marqueur* `'x'`*) et celui pour* `fib` *par des disques (marqueur* `'o'`*).*

**Question 17.** *Écrire une fonction* `plot_list_ns` *ayant la même spécification que* `plot_list`*, mais avec des temps en nanosecondes.*

**Question 18.** *Utiliser la fonction* `plot_list_ns` *pour tracer simultanément les temps d'exécution de* `fibrec(k)` *et* `fib(k)` *en fonction de* $\texttt{k} = 0, \ldots, \texttt{Nrec}$*. Les temps doivent être en nanosecondes.*