I Rappels de cours

I.1 Utilisation de bibliothèques de fonctions

```
import nom_bibliotheque #on importe toute la bibliothèque

#ou bien juste une fonction
from nom_bibliotheque import fonction #on peut remplace "fonction" par "*" pour tout importer

#utilisation des fonctions importées :
nom_bibliotheque.fonction(...)
```

Utilisation d'un alias :

```
import nom_bibliotheque as nb #alias

#utilisation des fonctions importées :
nb.fonction(...)
```

Quelques bibliothèques très usuelles (et alias usuel) :

- `numpy` (`as np`) : manipulations de matrices et fonctions mathématiques
- `matplotlib.pyplot` (`as plt`) : graphismes mathématiques
- `random` (`as rd`) : simulations aléatoires
- `math` : (prononcer à l'anglaise) fonctions mathématiques

I.2 Lecture d'un fichier

```
alias=open(nom_fichier,'r') #ouverture en lecture ('r' pour read, 'w' pour write, 'a' pour append, i.e. ajout en fin de fichier)
        # (le fichier doit être situé dans le répertoire d'exécution de pyzo)
        # et attribution d'un nom (alias) pour le manipuler comme une variable
x=alias.readline() #lecture de la première ligne (de type string)
y=alias.readline() #lecture de la ligne suivante (de type string) ...

#ou bien
L=alias.readlines() #lecture de toute les lignes (de type list of string)

alias.close() # fermeture pour éviter les pertes de données
```

I.3 Tracés de fonctions (avec `pyplot`)

Le tracé informatique du graphique d'une fonction consiste à tracer la ligne brisée passant par un certain nombre, fini, de points de la courbe théorique. On obtient une « bonne » représentation de la courbe théorique lorsque le nombre de points utilisés est suffisamment grand.

```
import matplotlib.pyplot as plt

#[...]  #calcul de la liste des abscisses et des ordonnées des points à afficher

#écriture dans la mémoire graphique :
plt.plot(abscisses1,ordonnees1) # abscisses1 et ordonnee1 de type list of float
plt.plot(abscisses2,ordonnees2) # abscisses1 et ordonnee1 de type list of float

plt.show() #affichage de la mémoire graphique
#N.B. : fermer la fenêtre graphique avant de relancer un nouvel affichage
```

I.4 Rappels sur la complexité temporelle d'un algorithme

Mesurer la complexité d'un algorithme consiste à compter ou à majorer le nombre de fois où l'opération (ou le groupe d'opérations) la plus utilisée est réalisée, en fonction de la taille n de la donnée traitée. Ce calcul permet notamment de prévoir l'évolution du temps d'exécution de l'algorithme sur machine, en fonction de n.

Une complexité s'exprime sous la forme O[fonction des paramètres]). Par exemple un complexité en $O(n)$ signifie que le nombre d'opérations effectuées est (*au plus*) proportionnel à n. *Exemples classiques à connaître :*

```
#complexité en O(n)
for i in range(n):
    ...
#idem pour une boucle interrompue --- dans une fonction

#complexité en O(np)
for i in range(n):
    for j in range(p):
        ...

#complexité en O(n(n-1)/2)=O(n^2)
for i in range(n):
    for j in range(i):
        ...

#complexité en O(ln n) --- cf TD
[dichotomie sur une donnée de longueur n]
```

I.5 Algorithmes numériques classiques

- **Calcul approché d'intégrale par la méthode des rectangles**
  Soit f une fonction continue définie sur un segment $[a, b]$. La formule d'approximation de $\int_a^b f$ par la méthode des trapèzes à l'ordre n (ou somme de Riemann d'ordre n de f) s'obtient à l'aide de la somme des aires des n rectangles de largeurs égales appuyés en bas sur l'axe des abscisses et dont le sommet en haut à gauche s'appuie sur la courbe de f.
- **Calcul approché d'intégrale par la méthode des trapèzes**
  Soit f une fonction continue définie sur un segment $[a, b]$. La formule d'approximation de $\int_a^b f$ par la méthode des trapèzes à l'ordre n s'obtient en remplaçant les rectangles de la méthode précédente par des trapèzes rectangles, appuyés sur l'axe des abscisses, et dont les deux autres sommets s'appuient sur la courbe de f.
- **Recherche par dichotomie d'un élément dans une liste, ou d'une valeur dans un intervalle :**
  Elle consiste en une succession d'itérations qui consistent chacune à :
  - diviser la liste/l'intervalle en deux parties, si possible égales, sinon à peu près égales ;
  - tester dans quelle partie l'élément/la valeur recherché(e) se trouve ;
  - appliquer l'étape suivante sur la partie où se trouve l'élément/la valeur recherché(e).

```
debut,fin = [initialisation]
while condition d arrêt:  #ou "for", si nb d'étapes connu au départ
    milieu = (debut + fin) / 2  #ou //2 pour un entier - attention à la
   terminaison de l'algorithme
    if test pour savoir quelle partie garder:
       debut = milieu #fin inchangée
    else:
       fin = milieu #début inchangé
```

II Exercices

Exercice 1. Affichages graphiques

Q1 *Écrire une fonction* `subdivision(a,b,n)`*, qui renvoie la liste dans l'ordre croissant des abscisses des* $n+1$ *points du segment* $[a,b]$ *formant une subdivision à pas constant, c'est-à-dire les* x_i *tels que* $a = x_0 < x_1 < \cdots < x_n = b$ *et* $x_{i+1} - x_i = \frac{b-a}{n}$.

Q2 *Écrire une fonction* `calcule(f,abscisses)`*, où* `f` *est une fonction et où* `abscisses` *est une liste de* n *abscisses* $a = x_0 < x_1 < \cdots < x_{n-1} = b$*, qui renvoie la liste des* $f(x_0), f(x_1), \ldots, f(x_{n-1})$.

Q3 *Écrire une procédure (fonction sans résultat)* `affiche_graphique(abscisses,ordonnees)`*, où* `abscisse` *est une liste de* n *abscisses* $a = x_0 < x_1 < \cdots < x_{n-1} = b$ *et* `ordonnees` *une liste de* n *ordonnées* $y_0, \ldots, y_{n-1}$ *qui provoque l'affichage de la ligne brisée passant par tous les point* $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$.

Q4 *Écrire les lignes de code permettant l'affichage du graphique de la fonction cosinus sur* $[-3.14, 3.14]$.

Q5 *Écrire une procédure* `affiche_graphiques(abscisses,L)`*, où* `abscisse` *est une liste de* n *abscisses* $a = x_0 < x_1 < \cdots < x_{n-1} = b$ *et* `L` *est une liste de* p *listes* `[ordonnee1,...,ordonneep]` *de chacune* n *ordonnées, qui effectue l'affichage simultané de* p *courbes.*

Q6 *Écrire une procédure* `affiche_graphique_simultane(f,abscisses,r,s)`*, où* `f(x,i)` *est une fonction d'une variable* `x` *de type* `float` *et dépendant d'un paramètre entier* `i`*, et où* `abscisses` *est une liste de* n *abscisses* $a = x_0 < x_1 < \cdots < x_{n-1} = b$*, qui effectue l'affichage simultané des graphiques des fonctions* $x \mapsto f(x,i)$ *sur* $[a,b]$*, pour tout* $i \in [\![r, s[\![$*, en utilisant les points d'abscisses* $x_0, \ldots, x_{n-1}$.

Q7 *Écrire une procédure* `affiche_famille_courbe(r,s)`*, où* `r` *et* `s` *sont des entiers naturels, qui affiche la famille des courbes d'équations* $y = ix^i(1-x)$ *sur* $[0,1]$ *pour tout* $i \in [\![r, s[\![$.

Q8 *Un fichier texte* `fichier` *contient les données suivantes :*

- *sur la première ligne le nombre* n *;*
- *puis une liste de* n *abscisses* $a = x_0 < x_1 < \cdots < x_{n-1} = b$ *(une valeur par ligne) ;*
- *puis une liste de* n *ordonnées* $y_0, .., y_{n-1}$ *(une valeur par ligne).*

Écrire une fonction `lecture(fichier)` *qui renvoie ces deux listes.*

Q9 *Écrire une procédure* `affiche(fichier)` *qui lit* n *abscisses* x_i *et les valeurs* $y_i = f(x_i)$ *d'une fonction* f *en ces abscisses, codées dans le fichier texte* `fichier` *comme dans la question précédente, et qui affiche le graphique de* f *sur l'intervalle* $[a,b]$.

Exercice 2. Calculs approchés d'intégrale

Q10 *Écrire la formule donnant la somme de* Riemann *d'ordre* n *d'une fonction* f *continue sur* $[a,b]$*. Écrire une fonction* `rectangle(f,a,b,n)` *qui renvoie la valeur approchée de* $\int_a^b f$ *donnée par cette formule*

Q11 *En s'aidant si nécessaire d'un dessin, donner la formule de l'aire d'un trapèze rectangle. Écrire la formule donnant l'approximation de* $\int_a^b f$ *par la méthodes des trapèzes à l'ordre* n*. Écrire une fonction* `trapeze(f,a,b,n)` *qui renvoie la valeur approchée de* $\int_a^b f$ *donnée par cette formule.*

Q12 *Adapter la fonction précédente en une fonction* `trapeze2(fichier)` *lorsque la fonction à intégrer est donnée comme aux questions Q8 et Q9 — où les abscisses ne forment pas forcément une subdivision régulière.*

Exercice 3. Recherches dichotomiques

Q13 *Écrire une fonction* `dichotomie(f,a,b,precision)`*, où* `a,b,precision` *sont des nombres, et où* f *est une fonction continue sur* $[a,b]$ *telle que* $f(a)f(b)<0$*, qui renvoie une valeur approchée à* `precision` *près d'une solution* x *de l'équation* $f(x)=0$*, obtenue par dichotomie.*

Q14 *Écrire une fonction* `recherche_dichotomique(L,x)`*, où* `L` *est une liste de nombres réels (*`float`*) rangés dans l'ordre croissant et où* `x` *est un nombre réel, qui renvoie* `True` *si* `x` *est élément de la liste* `L` *et* `False` *sinon, en procédant par dichotomie.* On sera vigilant sur le problème de la terminaison de boucle et on testera sur `[x-1,x]` *et* `[x,x+1]` *Quelle est la complexité dans le pire des cas de cet algorithme en fonction de la longueur* n *de* `L`*.*

Q15 *Écrire une fonction* `dichotomie2(L)`*, où* `L` *contient les valeurs* $f(0), f(1), \ldots, f(p)$ *d'une fonction* f *telle que* $f(0)$ *et* $f(p)$ *sont de signes opposés, et qui renvoie deux entiers consécutifs* i *et* $i+1$ *dont les images par* f *sont de signe opposés.*

En faisant une interpolation linéaire de f *entre* i *et* $i+1$ *(remplacer la courbe de* f *par une droite), écrire une fonction* `resolution(L)` *qui renvoie une solution approchée de l'équation* $f(x)=0$*.*

III Pour ceux qui ont déjà fini

Exercice 4. *Révisions sur les listes*

Q16 *Écrire une fonction* `melange(L)`*, où* `L` *est une liste de longueur paire, qui renvoie la liste obtenue en mélangeant* `L` *à la manière d'un jeu de cartes : on coupe le jeu en deux tas égaux puis on mélange les tas en intercalant une carte du second tas après chaque carte du premier tas.* Par exemple : `melange([1,2,3,4,5,6])` → `[1,4,2,5,3,6]`

Q17 Pour les 5/2 : *Écrire une deuxième version de la fonction* `melange` *en utilisant uniquement les manipulations de piles (méthodes* `append` *et* `pop`*) — i.e. sans utiliser d'extraction de terme d'une liste (du type* `L[i]` *ou* `L[i:j]`*).*

Q18 *Écrire une fonction* `est_liste(L,l)`*, où* `L` *et* `l` *sont des listes, qui renvoie* `True` *si les éléments de* `l` *sont des éléments consécutifs de* `L` *et qui renvoie* `False` *sinon.*
Par exemple : `est_liste([1,3,5,7,9,11,13],[5,7,9])` → `True`
`est_liste([1,3,5,7,9,11,13],[5,7,11])` → `False`

Q19 *Écrire une fonction* `est_liste2(L,l)`*, où* `L` *et* `l` *sont des listes, qui renvoie* `True` *si les éléments de* `l` *sont des éléments de* `L`*, non nécessairement consécutifs, mais rangés dans le même ordre que celui de* `L`*, et qui renvoie* `False` *sinon.*
Par exemple : `est_liste2([1,3,5,7,9,11,13],[5,7,11])` → `True`
`est_liste2([1,3,5,7,9,11,13],[5,1,7])` → `False`

Corrigé

Q1

```
def subdivision(a,b,n):
    pas = (b-a)/n #calcul fait une fois pour toutes
    return [a+k*pas for k in range(n+1)]

# NB : fonction déjà disponible dans numpy :
# import numpy as np
# np.linspace(a,b,n) #mais resultat de type np.array
# voire, (mais il manque alors la dernière valeur) :
# np.arange(a,b,n)

#ou
def subdivision2(a,b,n):
    pas = (b-a)/n #calcul fait une fois pour toutes
    x = a
    sub = [x]
    for k in range(n):
        x = x + pas
        sub.append(x)
    return sub
```

Q2

```
def calcule(f,abscisses):
    return [f(x) for x in abscisses]

#ou
def calcule2(f,abscisses):
    ordonnees = []
    for i in range(len(abscisses)):
        ordonnees.append(f(abscisses[i]))
    return ordonnees
```

Q3

```
import matplotlib.pyplot as plt

def affiche_graphique(abscisses,ordonnes):
    plt.plot(abscisses,ordonnes)
    plt.show()
```

Q4

```
from numpy import cos #ou from math import cos

n = 100
abscisses = subdivision(-3.14,3.14,n) #choix arbitraire d'une valeur de
    n
ordonnees = calcule(cos,abscisses)
affiche_graphique(abscisses,ordonnees)
```

Q5

```
def affiche_graphiques(abscisses,L):
    for j in range(len(L)) :
        plt.plot(abscisses,L[j])
    plt.show()


#ou
```

```
def affiche_graphiques(abscisses,L):
    for ordonnees in L :
        plt.plot(abscisses,ordonnees)
    plt.show()
```

Q6

```
def affiche_graphique_simultane(f,abscisses,r,s):
    L = []
    for i in range(r,s):
        ordonnee=[]
        for k in range(len(abscisses)):
            ordonnee.append(f(abscisses[k],i))
        L.append(ordonnee)
    affiche_graphiques(abscisses,L)

#ou
def affiche_graphique_simultane(f,abscisses,r,s):
    L = []
    for i in range(r,s):
        ordonnee = []
        for x in abscisses:
            ordonnee.append(f(x,i))
        L.append(ordonnee)
    affiche_graphiques(abscisses,L)
```

Q7

```
def f(x,i):
    return i*x**i*(1-x)

def affiche_famille_courbes(r,s):
    n = 100 #choix d'une valeur de n arbitraire
    abscisses = subdivision(0,1,n)
    affiche_graphique_simultane(f,abscisses,r,s)
```

Q8

```
def lecture(fichier):
    donnees = open(fichier,'r')
    n=int(donnees.readline())
    abscisses,ordonnees = [],[]
    for i in range(n):
        abscisses.append(float(donnees.readline())) #conversion des chaî
   nes de caractères en flottants
    for i in range(n):
        ordonnees.append(float(donnees.readline()))
    donnees.close()
    return [abscisses,ordonnees]
```

Q9

```
def affiche(fichier):
    [abscisses,ordonnees] = lecture(fichier)
    affiche_graphique(abscisses,ordonnees)
```

Q10 $S_n = \dfrac{b-a}{n} \sum_{k=0}^{n-1} f\left(a + \frac{k(b-a)}{n}\right)$

```python
def rectangle(f,a,b,n):
    abscisses = subdivision(a,b,n) #NB : le dernier point ne sera pas utilisé
    somme = 0
    for i in range(n):
        somme += f(abscisses[i])
    return somme * (b-a)/n

#ou
def rectangle2(f,a,b,n):
    abscisses = subdivision(a,b,n)
    somme = 0
    for x in abscisses:
        somme += f(x)
    return somme * (b-a)/n
```

Q11 Aire d'un trapèze rectangle de base h dont les côté orthogonaux à la base sont de longueurs respectives ℓ et ℓ' :

$$h \times \frac{\ell + \ell'}{2}.$$

D'où la formule d'approximation : $T_n = \sum_{k=0}^{n-1} \frac{b-a}{n} \times \frac{f\left(a + \frac{k(b-a)}{n}\right) + f\left(a + \frac{(k+1)(b-a)}{n}\right)}{2}$.

Soit encore : $T_n = \frac{b-a}{n}\left(\frac{f(a)}{2} + \frac{f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + \frac{k(b-a)}{n}\right)\right)$.

```python
def trapeze(f,a,b,n):
    abscisses = subdivision(a,b,n)
    somme = 0
    for i in range(n):
        somme += (f(abscisses[i])+f(abscisses[i+1]))
    return somme * (b-a)/(2*n)

#ou avec la formule réduite
def trapezeb(f,a,b,n):
    abscisses=subdivision(a,b,n)
    somme = (f(abscisses[0])+f(abscisses[n]))/2
    for i in range(1,n):
        somme += f(abscisses[i])
    return somme * (b-a)/n
```

Q12

```python

def trapeze2(fichier):
    [x,y]=lecture(fichier)
    somme = 0
    for i in range(len(x)-1):
        somme += (y[i]+y[i+1])/2*(x[i+1]-x[i]) #/ prioritaire sur *
    return somme
```

Q13

```python
def dichotomie(f,a,b,precision):
    debut,fin = a,b
    while fin-debut > precision:
        milieu = (debut+fin)/2
        if f(milieu)*f(debut) > 0:
            debut = milieu
        else:
            fin = milieu
    return debut #ou fin, ou toute valeur entre les deux
```

Q14

```
def recherche_dichotomique(L,x):
     # debut (inclus) et fin (inclus) sont les indices entre lesquels on recherche x
    if L == []:
        return False
    else :
        debut,fin = 0,len(L)-1
        while fin-debut >= 0: #tant qu'il y a des coefficients à comparer avec x; on peut aussi écrire fin-debut > 1
            milieu = (debut+fin)//2
            if L[milieu] == x:
                return True
            elif L[milieu] < x:
                debut = milieu + 1
            else:
                fin = milieu - 1
        return False
```

Calcul de la complexité : comme la liste à examiner est divisée, en longueur, par au moins 2 à chaque itération, le nombre d'itérations de la boucle est au maximum p tel que $2^p = n$, soit $p = \frac{lnn}{\ln 2}$ d'où une complexité en $\boxed{\mathrm{O}(\log_2 n)}$ ou aussi $\boxed{\mathrm{O}(\ln n)}$.

Q15

```
def dichotomie2(L):
    debut,fin = 0,len(L)-1
    while debut+1 < fin: #i.e. fin-debut >= 0
        #print(debut,fin)
        milieu = (debut+fin)//2
        if L[milieu]*L[fin] <= 0:
            debut = milieu
        else:
            fin = milieu
    return debut, fin

def resolution(L):
    debut,fin = dichotomie2(L)
    #formule d'interpolation linéaire
    #(intersection de la droite passant par les deux points avec l'axe des abscisses)
    if L[fin] != L[debut]:
        return debut-(fin-debut)/(L[fin]-L[debut])*L[debut]
        #ou bien : resultat=debut+L[debut]/(L[fin]-L[debut]) puisque fin=debut+1
    else:
        return L[debut]
```

Q16

```
def melange(L):
    m=len(L)//2 #n est pair donc les deux tas de cartes ont le même nombre de cartes
    tas1 = L[:m]
    tas2 = L[m:]
    M = []
    for i in range(m):
        M.append(tas1[i])
        M.append(tas2[i])
    return M


```

```
#ou sans slicing
def melangeb(L):
    m=len(L)//2 #n est pair donc les deux tas de cartes ont le même nombre de cartes
    M = []
    for i in range(m):
        M.append(L[i])
        M.append(L[m+i])
    return M
```

Q17

```
def melange2(L):
    n = len(L)
    m = len(L)//2 #n est pair donc les deux tas de cartes ont le même nombre de cartes
    tas1,tas2 = [],[]
    #constitution des tas
    for i in range(n):
        if i >= m:
            tas1.append(L.pop())
        else:
            tas2.append(L.pop())
        i += 1
    #mélange des tas
    M = []
    for i in range(m):
        M.append(tas1.pop())
        M.append(tas2.pop())
    return M
```

Q18

```
def est_liste(L,l):
    '''L et l sont supposées non vides'''
    present = recherche_dichotomique(L,l[0])
    if not present:
        return False
    else:
        N = len(L)
        n = len(l)
        for i in range(N-n+1):
            if l[0] == L[i] and L[i:i+n] == l:
                return True
        return False
```

Q19

```
253 def est_liste2(L,l):
254     '''L et l sont supposées non vides'''
255     present = recherche_dichotomique(L,l[0])
256     if not present:
257         return False
258     else:
259         N = len(L)
260         n = len(l)
261         i = 0
262         for x in l:
263             j = i
264             xestdansL = False
265             while xestdansL == False and j < N:
266                 if L[j] == x:
267                     xestdansL = True
268                 j += 1
269             if xestdansL == False:
270                 return False
271             else:
272                 i = j
273         return True
```