1 Méthode par dichotomie

Exercice 1

a) i. Justifier que l'équation $x + x^2 + x^3 = 1$ a une unique solution dans $[0, 1]$.

ii. Définir dans python la fonction $f : x \mapsto x + x^2 + x^3 - 1$ (qui aura le nom f).

iii. En utilisant une méthode par dichotomie, compléter la fonction python suivante pour qu'elle calcule l'unique solution de l'équation précédente, à une précision donnée par l'utilisateur.

```
def resolve(p):
    a=___
    b=___
    _____ (b-a___p):
        if (___):
            a=___
            b=___
        else:
            a=___
            b=___
    return ___
```

b) Mêmes questions pour l'équation $\sum_{k=1}^{n} x^k = 1$, avec $n \in \mathbb{N}^*$ quelconque (qui devra ainsi faire partie des arguments des fonctions).

c) En reprenant les programmes précédents, proposer un fonction python utilisant une méthode par dichotomie qui calcule, lorsque l'utilisateur donne :
— une fonction continue f,
— un réel α,
— un intervalle $[a, b]$ tel que $f(a) < \alpha < f(b)$ ou $f(a) > \alpha > f(b)$ (le programme pourra vérifier cette condition),
— une précision ε,
une valeur approchée d'une solution de l'équation $f(x) = \alpha$, à ε près.

Exercice 2

On s'intéresse à l'équation $(E) : \dfrac{1}{\mathrm{e}^x} = \ln\left(\dfrac{1}{x}\right)$.

a) **Étude mathématique**

On pose $g : x \mapsto \mathrm{e}^{-x} + \ln(x)$.

i. Justifier que la fonction g est dérivable sur $\mathbb{R}_+^*$, et calculer sa dérivée.

ii. En utilisant le fait que pour tout $x \in \mathbb{R}$, $\mathrm{e}^x > x$, montrer que g est strictement croissante sur $\mathbb{R}_+^*$.

iii. Calculer les limites de g en 0 et $+\infty$.

iv. En déduire que g s'annule une, et une seule fois sur $\mathbb{R}_+^*$, en un point que l'on appellera α.

v. Montrer que α est l'unique solution de (E).

vi. Montrer que $\frac{1}{\mathrm{e}} < \alpha < 1$.

b) **Programmation**

Compléter la fonction python suivante pour qu'elle calcule, au moyen d'une méthode par dichotomie, une valeur approchée de α à n chiffres après la virgule.

```
import numpy as np

def alpha(n):
    ___
    ___
    ___:
        c=___
        if np.___<np.___:
            a=___
        else:
            b=___
    return (a+b)/2
```

2 Par des suites récurrentes

2.1 Version TLM

Exercice 3

On veut ici résoudre l'équation $(E) : x = 1 + \sqrt{x}$.

a) **Existence de la solution :**

i. Étudier la fonction $f : x \mapsto 1 + \sqrt{x} - x$, et justifier que l'équation (E) a une unique solution strictement positive, que l'on notera r.

ii. Justifier que $1 \leqslant r \leqslant 4$.

b) **Suites récurrentes :**

On considère les suites (u_n) et (v_n) définies comme suit :

$$\left\{\begin{array}{rcl} u_0 &=& 1 \\ \forall n \in \mathbb{N} \quad u_{n+1} &=& 1 + \sqrt{(u_n)} \end{array}\right. \text{ et } \left\{\begin{array}{rcl} v_0 &=& 4 \\ \forall n \in \mathbb{N} \quad v_{n+1} &=& 1 + \sqrt{v_n} \end{array}\right.$$

i. a. Montrer par récurrence que pour tout $n \in \mathbb{N}$, $u_n < r < v_n$.

b. Montrer que (u_n) est croissante et (v_n) est décroissante.

c. En déduire que (u_n) et (v_n) convergent toutes les deux vers r.

c) **Programmation :**

i. Définir dans `python` la fonction $g : x \mapsto 1 + \sqrt{x}$.

ii. Compléter le programme suivant pour qu'il calcule, pour une précision p donnée, un encadrement de r de cette précision.

```
def solution(p):
    u=___
    v=___
    while (___):
        u=___
        v=___
    return [u,v]
```

Exercice 4

On s'intéresse aux solutions de l'équation $(E) : x = 2\ln(1 + x)$.

a) **Existence de la solution :**

i. Étudier la fonction $f : x \mapsto 2\ln(1 + x) - x$, et justifier que l'équation (E) a une unique solution strictement positive, que l'on notera r.

ii. Justifier que $2 \leqslant r \leqslant 4$.

b) **Suites récurrentes :**

On considère les suites (u_n) et (v_n) définies comme suit :

$$\left\{\begin{array}{rcl} u_0 &=& 2 \\ \forall n \in \mathbb{N} \quad u_{n+1} &=& 2\ln(1 + u_n) \end{array}\right. \text{ et } \left\{\begin{array}{rcl} v_0 &=& 4 \\ \forall n \in \mathbb{N} \quad v_{n+1} &=& 2\ln(1 + v_n) \end{array}\right.$$

i. Montrer par récurrence que ces deux suites sont bien définies, que (u_n) est croissante, (v_n) décroissante, et que pour tout $n \in \mathbb{N}$, $u_n \leqslant r \leqslant v_n$.

ii. Montrer que les deux suites (u_n) et (v_n) convergent, et que leur limite vaut r.

c) **Programmation :**

i. Définir dans `python` la fonction $g : x \mapsto \ln(1 + x)$.

ii. Écrire un programme en `python` qui calcule u_n et v_n pour n donné.

iii. Écrire un programme en `python` qui calcule une valeur approchée de r à une précision ε donnée.

2.2 Version IAF

Exercice 5

Cette fois-ci, c'est l'équation $(E) : x = \frac{1}{\sqrt{1+x}}$ que l'on veut résoudre sur $\mathbb{R}_+$.

On pose $f : x \mapsto \frac{1}{\sqrt{1+x}}$.

a) **Existence de la solution :**

i. Montrer que f est dérivable sur $\mathbb{R}_+$, et en donner la dérivée.

ii. Montrer que pour tout $x \in \mathbb{R}_+$, $|f'(x)| < \frac{1}{2}$.

iii. En déduire que la fonction $g : x \mapsto f(x) - x$ est strictement décroissante sur $\mathbb{R}_+$, et en donner les limites en 0 et $+\infty$.

iv. En déduire que l'équation (E) a une unique solution positive, que l'on notera φ.

v. Montrer que $0 < \varphi < 1$.

b) **Suite récurrente :**

On considère à nouveau une suite (u_n) définie par

$$\begin{cases} u_0 &= 0 \\ \forall n \in \mathbb{N} \quad u_{n+1} &= f(u_n) \end{cases}$$

i. Montrer que pour tout $n \in \mathbb{N}$

$$|u_{n+1} - \varphi| < \frac{|u_n - \varphi|}{2}$$

ii. En déduire que pour tout $n \in \mathbb{N}$

$$|u_n - \varphi| < \frac{1}{2^n}$$

iii. En déduire que $u_n \underset{n\to+\infty}{\longrightarrow} \varphi$.

c) **Programmation :**

i. Compléter le programme suivant pour qu'il calcule, pour une précision p donnée, un encadrement de φ de cette précision.

```
def solution(p):
    n=0
    u=___
    ___:
        u=___
        n=n+1
    return u
```

d) Bonus : Était-il possible d'utiliser le TLM dans cette situation (voir 3) ?

Exercice 6

Cette fois-ci, c'est l'équation $(E) : x = \mathrm{e}^{-\frac{x^2}{2}}$ que l'on veut résoudre sur $\mathbb{R}_+$.

On pose $f : x \mapsto \mathrm{e}^{-\frac{x^2}{2}}$.

a) **Existence de la solution :**

i. Montrer que f est dérivable deux fois sur $\mathbb{R}_+$, et en donner la dérivée, ainsi que sa dérivée seconde.

ii. Étudier la convexité de f.

iii. En déduire que pour tout $x \in \mathbb{R}_+$, $|f'(x)| < \frac{1}{\sqrt{\mathrm{e}}}$.

iv. En déduire que la fonction $g : x \mapsto f(x) - x$ est strictement décroissante sur $\mathbb{R}_+$, et en donner les limites en 0 et $+\infty$.

v. En déduire que l'équation (E) a une unique solution positive, que l'on notera φ.

vi. Montrer que $0 < \varphi < 1$.

b) **Suite récurrente :**

On considère à nouveau une suite (u_n) définie par

$$\begin{cases} u_0 &= 0 \\ \forall n \in \mathbb{N} \quad u_{n+1} &= f(u_n) \end{cases}$$

i. Montrer que pour tout $n \in \mathbb{N}$
$$|u_{n+1} - \varphi| < \frac{|u_n - \varphi|}{\sqrt{\mathrm{e}}}$$

ii. En déduire que pour tout $n \in \mathbb{N}$
$$|u_n - \varphi| < \mathrm{e}^{-\frac{n}{2}}$$

iii. En déduire que $u_n \underset{n\to+\infty}{\longrightarrow} \varphi$.

c) **Programmation :**

i. Proposer un programme `python` qui calcule, pour une précision p donnée, un encadrement de φ de cette précision.

3 Pour aller plus loin...

3.1 Généralité de la méthode "IAF"

Exercice 7

On suppose ici qu'on a affaire à une fonction f *contractante* sur un intervalle I, c'est-à-dire que :
— I est stable par f (c'est-à-dire que si $x \in I$, $f(x) \in I$),
— f est dérivable sur I,
— il existe une constante k telle que $\forall x \in I$, $|f'(x)| \leqslant k < 1$.

a) Montrer tout d'abord que, quelque soit $x, y \in I$, $|f(x)-f(y)| \leqslant k|x-y| < |x-y|$ (d'où l'appellation de "contractante").

b) Montrer que la fonction $g : x \mapsto f(x) - x$ est strictement décroissante sur I.

c) En déduire que f a un unique point fixe sur I, que l'on notera φ. On pourra passer par les étapes suivantes :
— Si l'intervalle $I = [a, b]$ est un segment, considérer le signe de $f(x) - x$ en a et en b.
— Si l'intervalle I est fini (mais pas forcément fermé), considérer le signe des limites à la place,
— Si l'intervalle I est infini, montrer que
$$f(x) - x \leqslant (k-1)x - ky + f(y)$$
(où y est un point quelconque de I) et en déduire que f tend vers $-\infty$ en $+\infty$. (on pourra utiliser l'autre version de l'IAF, et adapter le résultat pour une limite en $-\infty$)

d) Prenons une suite u_n vérifiant l'équation $u_{n+1} = f(u_n)$, de premier terme $u_0 \in I$.

i. Montrer que $\forall n \in \mathbb{N},\ u_n \in I$.

ii. Montrer que $\forall n \in \mathbb{N},\ |u_{n+1} - \varphi| \leqslant k|u_n - \varphi|$.

iii. En déduire que $\forall n \in \mathbb{N},\ |u_n - \varphi| \leqslant k^n|u_0 - \varphi|$.

iv. En déduire le théorème suivant :

Théorème 3.1
Soit f une fonction contractante sur I. Toute suite (u_n) vérifiant pour tout $n \in \mathbb{N}$ $u_{n+1} = f(u_n)$, de premier terme dans I, converge vers l'unique point fixe de f sur I.

e) Vérifier que les fonctions suivantes sont contractantes, et tester au moyen du programme créé au 3.2 le théorème précédent :
— $x \mapsto \sqrt{x}$, $I = \left]\frac{1}{2}, +\infty\right[$,
— $x \mapsto \mathrm{e}^{-x}$, $I =]\varepsilon, +\infty[$ (dès que $\varepsilon > 0$),
— $x \mapsto \frac{1}{1+\mathrm{e}^x}$, $I = \mathbb{R}$,
— $x \mapsto \mathrm{e}^{-x^2}$, $I = \mathbb{R}$.

3.2 Étude graphique des suites $u_{n+1} = f(u_n)$

Il est possible de représenter graphiquement l'évolution d'une telle suite :

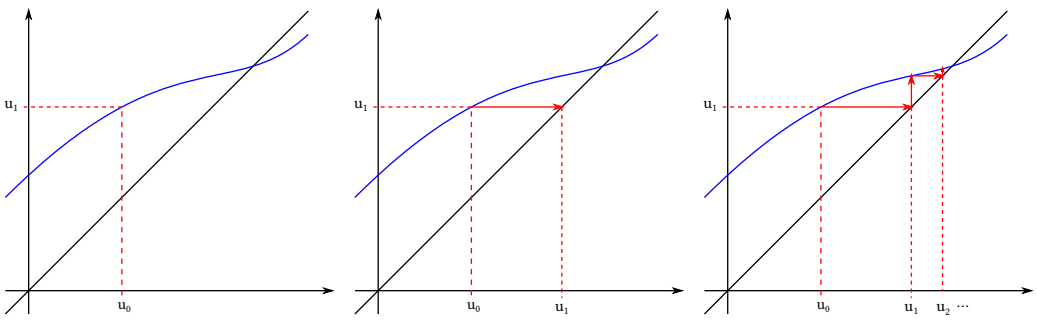

Figure 1 – Représentation graphique d'une suite récurrente

Pour tracer un tel graphe, il suffit (outre de tracer le graphe de la fonction f et de $x \mapsto x$) de relier les points d'abscisse et d'ordonnée

x	u_0	u_1	u_1	u_2	u_2	u_3	$\dots$
y	u_1	u_1	u_2	u_2	u_3	u_3	$\dots$

On remarque que les ordonnées sont les même que les abscisses, décalées de 1 cran.

a) Compléter le programme python suivant pour qu'il trace le graphique correspondant à l'évolution de la suite récurrente avec $f : x \mapsto \sqrt{x}$:

```
import numpy as np
import matplotlib.pyplot as plt

n=int(input("nombre de termes?"))
u0=float(input("premier terme?"))
def f(x):
    return ___
sam=500
plt.clf()
plt.xlim([0, 2])
plt.ylim([0, 2])
S=np.zeros(2*n+2)
S[1]=u0
for i in range(1,n+1):
    S[2*i]=___
    S[2*i+1]=S[2*i]
X=np.linspace(0,2,sam)
Y=___
plt.plot(X,X,color='black')
plt.plot(X,___,color='blue')
plt.plot(S[range(1,(2*n)+1)],S[range(2,(2*n+1)+1)],color='red')
```

b) Quel semble être le comportement de cette suite, quelque soit le premier terme (à condition qu'il soit positif, bien sûr !) ? Démontrer votre hypothèse.

c) En faire de même avec la fonction $f : x \mapsto \mathrm{e}^{-x}$.

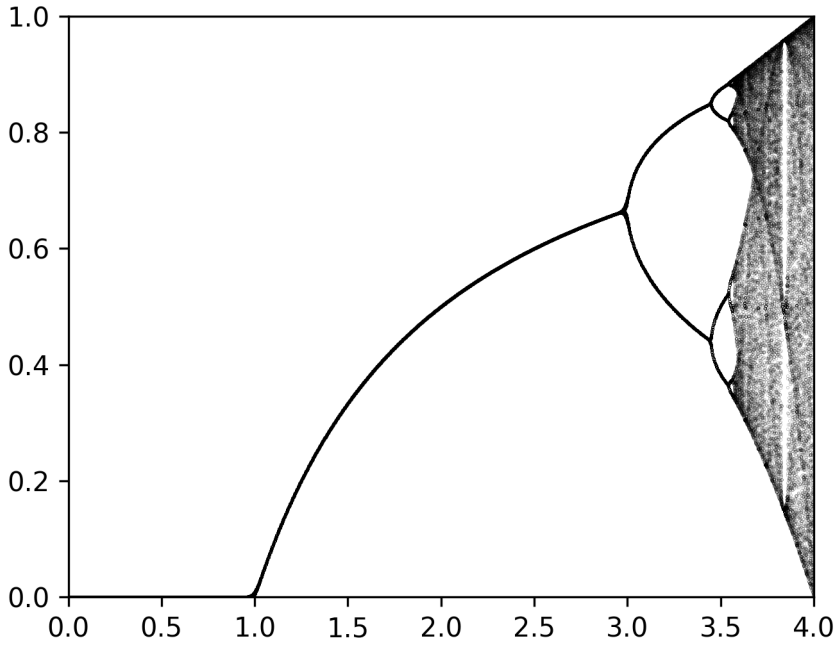

Figure 2 – Diagramme de bifurcation de la suite logistique

3.3 Un cas... chaotique

Certains détails de cette section utilisent le théorème 3.1 de la section précédente, mais il est possible de simplement lire ce théorème et la discussion ci-dessous sans chercher à tout justifier. Les simulations informatiques, intéressantes par elles-mêmes et illustrant les résultats décrits, ne nécessitent aucune justification.

Une suite logistique est une suite vérifiant la relation $u_{n+1} = \mu u_n(1-u_n) = f(u_n)$. Une interprétation classique est l'évolution d'une population : la population de la génération suivante (u_{n+1}) est calculé comme le produit d'un coefficient de fertilité μ, de la population courante u_n (chaque individu ayant ainsi en moyenne μ enfants) et d'un facteur de surpopulation $(1 - u_n)$ (seule une fraction $1 - u_n$ de la population survit à l'age adulte, s'il y a trop d'individus les ressources viennent à manquer).

a) Justifier que si $\mu \in [0, 4]$, la suite u_n reste entre 0 et 1 (on suppose que $u_0 \in [0, 1]$).

b) i. Justifier que si $\mu < 1$, f est contractante, et la population finit par s'éteindre. Ce résultat reste vrai pour $\mu = 1$, même si f n'est plus contractante. (`simulation` avec par exemple $\mu = .75$)

ii. Montrer que sinon, la fonction f a un unique point fixe $\varphi = \frac{\mu-1}{\mu}$ dans $]0, 1[$.

iii. Montrer que si $\mu \in]1, 2]$, la suite est monotone si $u_0 \leqslant \frac{1}{2}$ (et donc à partir du rang 1 dans tous les cas), et la population tend vers ce point fixe. (`simulation` avec par exemple $\mu = 1.75$)

iv. Montrer que si $\mu \in]2, 3[$, f est contractante au voisinage de φ. La population finit aussi par tendre vers le point fixe, mais cette fois-ci en oscillant autour. (`simulation` avec par exemple $\mu = 2.75$)

v. C'est si $\mu \in]3, 4]$ que tout fiche le camp : si $\mu \leqslant 3.57$ (environ), la suite oscille entre plusieurs valeurs (`simulation` avec $\mu = 3.54$). Et si $\mu \geqslant 3.57$ (`simulation` avec $\mu = 3.82$), le comportement est complètement chaotique : il dépend fortement de la population initiale, et s'approche d'un grand nombre de points. Ces points "attracteurs" ont une forme particulière (appelée "attracteur étrange"), la figure 2 en présente le graphe en fonction du paramètre μ.

c) Reprendre le programme du 3.2 pour construire une fonction `logplot(mu,u0,n0,n)` qui trace, pour μ et u_0 donnés, le graphe des termes de la suite logistique correspondante entre n_0 et n. Tester ce programme pour $\mu = .75, 1.75, 2.75, 3.3, 3.82, 3.84, 3.85, 3.86, \ldots$

Le programme suivant permet de construire une approximation (assez grossière) de l'attracteur dans python, en traçant les valeurs des suites logistiques u_n en fonction du paramètre μ, pour n entre `N0` et `N`. `step` correspond à l'échantillonnage en μ. Plus ces valeurs sont importantes, plus le programme sera long à tourner...

```
mport numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(5, 3), dpi=300)
plt.clf()
plt.xlim(0,4)
plt.ylim(0,1)
N0=100
N=100
step=0.002
MU=np.arange(0,4,step)
for mu in MU:
    M=np.zeros(N0);P=np.zeros(N0)
    u=0.1
    for n in range (1,N0):
        u=mu*u*(1-u)
    for n in range(0,N):
        u=mu*u*(1-u)
        M[n]=mu
        P[n]=u
    plt.plot(M,P,'.',color='black',markersize=0.1)

plt.savefig('LogAtt.png')
```

A Commandes

A.1 Tableaux et arrays

Syntaxe (librairie numpy)

```
import numpy as np
```

Syntaxe (déclaration d'arrays)
Déclaration type "liste" :

```
np.array([a,b,...]) #déclaration explicite
np.array([f(x) for x in A]) #déclaration paramétrique
np.array([x for x in A if condition]) #déclaration implicite
```

N.B. : un tableau à plusieurs entrées est simplement vu comme une "liste de liste".
Déclarations spéciales :

```
np.arange(début,fin,pas) #plage de données: écart entre les valeurs
np.linspace(début,fin,nb) #plage de données: nombre de terme
np.ones(n)      #vecteur-ligne rempli de 1
np.ones([n,p])    #matrice (n,p) remplie de 1
np.zeros(n)     #vecteur-ligne rempli de 0
np.zeros([n,p]) #matrice (n,p) remplie de 0
```

Syntaxe (accès aux données dans un `array`*)*

```
A[i,j]  # case (i,j)
A[i,:]  #i-ieme ligne
A[:,j]  #j-ieme colonne
A[i1:i2,j1:j2]#sous-matrice
```

A.2 Représentations graphiques

Syntaxe (graphes : librairie `matplotlib.pyplot`*)*

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

Ne pas oublier de "nettoyer" la fenêtre graphique au moyen de `plt.clf()`.
Selon les versions de python, il peut être nécessaire de demander explicitement d'afficher la fenêtre graphique par

```
plt.show()
```

Syntaxe (graphe)

```
plt.plot(Abscisses,Ordonnées)
```

(python trace simplement les points d'abscisse et d'ordonnées spécifiées, et les relie...)

B Dichotomie : correction

Le programme suivant permet de résoudre, à une précision p donnée par l'utilisateur, l'équation $f(x) = 0$, où f est une fonction prédéfinie dans python, négative en a et positive en b :

```
def resolve(a,b,p):
    while (b-a>p):
        if (f((a+b)/2)>0):
            a=a
            b=(a+b)/2
        else:
            a=(a+b)/2
            b=b
    return (a+b)/2
```