TP n° 07 – Résolution d'équations simples

I Résolution d'équations du second ordre

L'objectif est de résoudre les équations de type (E): $ax^2 + bx + c = 0$ où a, b et c sont des réels, (donc seront des flottants dans vos programmes).

Exercice 1. Dans cet exercice, on suppose que $a \neq 0$.
Écrire une fonction `solution(a,b,c)` qui renvoie les solutions de (E): $ax^2 + bx + c = 0$ et précise la nature de ses solutions. Par exemple :

```
>>> solution(2,-6,4)
Deux solutions reelles  x1=1.0 et x2=2.0
>>> solution(4,-4,1)
Une solution double  x=0.5
>>> solution(1,-2,2)
Deux solutions complexes  x1=1+1j et x2=1-1j
```

Exercice 2. Dans cet exercice, (E) n'est pas forcément une équation du second degré : a peut être nul. Ecrire une fonction `solution2` pour prendre en compte tous les cas. (On commencera par construire sur feuille un algorigramme.)
Par exemple :

```
>>> solution2(1,-3,2)
Deux solutions reelles : x1=1.0 et x2=2.0
>>> solution2(0,2,0)
Une solution reelle : x=0.0
>>> solution2(0,0,1)
Pas de solution
>>> solution2(0,0,0)
Une infinite de solutions : tous les reels
```

Exercice 3.

1. Résolvez **à la main** l'équation suivante :

$$(E_5)\colon x^2 + (1 + 2^{-50})x + 0,25 + 2^{-51} = 0$$

2. Résolvez cette équation à l'aide de la fonction `solution`. Que constatez-vous ? Pourquoi ?

Exercice 4.

1. Résolvez **à la main** les deux équations suivantes :

$$(E_3)\colon x^2 + 6x + 9 \qquad (E_4)\colon 0.1x^2 + 0.6x + 0.9 = 0$$

2. Résolvez ces équations à l'aide de la fonction `solution`. Que constatez-vous ? Pourquoi ?

II Résolution par dichotomie

ATTENTION : vous aurez besoin de cet algorithme au TP n°10. Donc, sauvegardez proprement et au bon endroit votre programme.

II.1 Principe

Soit f continue telle que $f(a)$ et $f(b)$ soient de signe contraire. Alors un zéro de f est dans $[a, b]$.
On construit une suite d'intervalles $[a_n, b_n]$ qui contiennent ce zéro. A chaque étape :
on note $c_n = \dfrac{a_n + b_n}{2}$.
- si $f(a_n)$ et $f(c_n)$ sont de signe contraire, alors on pose : $a_{n+1} = a_n$ et $b_{n+1} = c_n$.
- sinon, on pose : $a_{n+1} = c_n$ et $b_{n+1} = b_n$.

On s'arrête quand $c_n = \dfrac{a_n + b_n}{2}$ est une approximation à ϵ près d'une solution, autrement dit quand :

$$b_n - a_n \leqslant 2\epsilon$$

Avantages : dès lors que $f(a)$ et $f(b)$ sont de signe contraire et que f est continue, la méthode converge vers une solution. On peut aussi prévoir à l'avance le nombre d'itérations nécessaires pour une précision choisie.
Inconvénient : la convergence n'est pas très rapide comparée à d'autres méthodes.

II.2 Application

Exercice 5. Écrire une fonction `dicho` qui prend comme entrée la fonction f à étudier, les bornes initiales a et b, la précision ϵ et qui renvoie $\dfrac{a_n + b_n}{2}$, approximation d'une solution à ϵ près.

Exercice 6.
1. Testez la fonction `dicho` sur $f(x) = x^2 - 2$ pour obtenir une approximation de $\sqrt{2}$ à $\epsilon = 0,001$ près.
2. Si on prend $a = 2$ et $b = 3$, que renvoie le programme ? Est-ce bien l'approximation d'une solution ? Pourquoi le programme renvoie cette valeur ?
3. Faites en sorte que votre fonction `dicho` affiche un message d'erreur dans ces cas là.

III Résolution avec des suites récurrentes

III.1 Principe

L'objectif est de résoudre une équation du type $f(x) = x$.
On considère une suite définie par récurrence de la façon suivante :

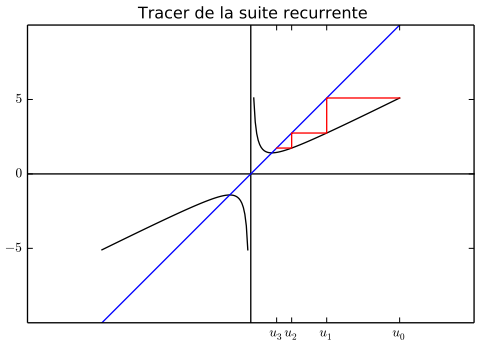
$$\begin{cases} u_0 = \text{constante} \\ u_{n+1} = f(u_n) \end{cases} \qquad (*)$$

Dans certains cas favorables, la suite $(u_n)_{n\in\mathbb{N}}$ converge. À ce moment là, sa limite est un point fixe de f, c'est-à-dire une solution de $f(x) = x$.

III.2 Application

Exercice 7. Écrire une fonction `rec` qui prend comme entrée f, u_0, n et qui renvoie u_n, le $n$ième terme de la suite définie par récurence en $(*)$.

Exercice 8. Tester votre fonction `rec` avec $f(x) = \dfrac{1}{2}\left(x + \dfrac{2}{x}\right)$ et $u_0 \in \mathbb{R}^*$.
On peut montrer que la suite $(u_n)_{n\in\mathbb{N}}$ converge et que les points fixes de f sont : $\sqrt{2}$ et $-\sqrt{2}$.



Exercice 9. Avec la fonction `dicho` appliquée à $g(x) = f(x) - x$, on retrouve les solutions de $f(x) = x$. Testez-le avec la fonction f de l'exercice précédent.
Entre `dicho` et `rec`, quel est l'algorithme le plus rapide ?

IV Annexe : les complexes

Un complexe se note : `z=1+2j`.
Attention : si vous voulez le complexe $z = j$, il faut écrire `z=1j` et non pas `z=j` car Python considère alors `j` comme une variable et non comme le complexe j.
Quelques fonctions :

`z.real`	partie réelle de z
`z.imag`	partie imaginiare de z
`z.conjugate()`	conjugué de z
`abs(z)`	module de z

Correction TP n° 07 – Résolution d'équations simples

Solution 1.

```
from math import sqrt
def solution(a,b,c):
        delta=b**2-4*a*c
        if delta >0:
                x1=(-b+sqrt(delta))/float(2*a)
                x2=(-b-sqrt(delta))/float(2*a)
                return "Deux solutions reelles", x1,x2
        elif delta==0:
                x=-b/float(2*a)
                return "Une solution double" ,x
        else:
                x1=(-b+sqrt(-delta)*1j)/float(2*a)
                x2=(-b-sqrt(-delta)*1j)/float(2*a)
                return "Deux solutions complexes", x1,x2

print solution(2,-6,4)
```

Solution 2.

```
def solution2(a,b,c):
        if a != 0:
                return(solution(a,b,c))
        else:
                if b!=0:
                        return "Une solution reelle" , -c/float(b)
                else :
                        if c!=0:
                                return "pas de solution"
                        else:
                                return "Une infinite de solutions : tous les reels."

print solution(0,0,0)
```

Solution 3.

1. $\Delta = 1 + 2^{-49} + 2^{-100} - 1 - 2^{-49} = 2^{-100} \neq 0$.
   On trouve deux solutions réelles distinctes.
2. `solution(1,(1+2**(-50)),0.25+2**(-51))` renvoie une unique solution. En effet, dans la représentation des nombres, on a vu que Python admet une limite de précision pour les flottants. 2^{-100} dépasse cette limite et Python évalue Δ à zéro.

Solution 4.

1. Dans les deux cas, on trouve une solution double : $x = -3$
2. `solution(1,6,9)` renvoie l'unique solution `-3`. Mais `solution(0.1,0.6,0.9)` renvoie deux solutions complexes.
   Dans le premier cas, Δ est un entier. Python le compare à zéro sans erreur. Dans le deuxième cas, Δ est un flottant qui vaut $\approx -5.10^{-17}$. Le comparer à zéro n'est plus exact.

Solution 5.

```
def dicho(f,a,b,eps):
    while (b-a)>2*eps:
        c=(float(a)+b)/2
        if f(a)*f(c)<0:
            b=c
        else:
            a=c
    return((a+b)/2)
```

Solution 6.

1.
```
def carre(x):
        return(x**2-2)
```

2. Sur $[2,3]$, f ne s'annule pas. Donc dans l'algorithme, on aura toujours $f(a)f(c)>0$, donc `a=c`. Le programme renvoie donc $3-\epsilon$.

3.
```
def dicho(f,a,b,eps):
    if f(a)*f(b)>0:
        return('nous ne savons pas si f s annule entre a et b')
    while (b-a)>2*eps:
        c=(float(a)+b)/2
        if f(a)*f(c)<0:
            b=c
        else:
            a=c
    return((a+b)/2)
```

Solution 7.

```
def rec(f,u,n):
   for i in range(n):
   u=f(u)
   return(u)
```

Solution 8. Selon le choix de $u_0 \neq 0$, la suite converge vers $\sqrt{2}$ ou $-\sqrt{2}$.

Solution 9. Avec `dicho(g,1,2,0.001)`, on trouve une approximation de $\sqrt{2}$ à $0,001$ près.
Pour comparer les deux programmes, on ajoute un compteur à la fonction `dicho` pour compter le nombre d'itérations effectuées par l'algorithme :

```
def dicho(f,a,b,eps):
        compteur=0
    if f(a)*f(b)>0:
        return('nous ne savons pas si f s annule entre a et b')
    while (b-a)>2*eps:
            compteur=compteur+1
        c=(float(a)+b)/2
        if f(a)*f(c)<0:
            b=c
        else:
            a=c
    return((a+b)/2,compteur)
```

Par exemple, pour `dicho(g,1,2,0.001)`, on obtient les trois premiers chiffres de $\sqrt{2}$ en 9 itérations.
Avec `rec(f,1,9)`, on obtient les 12 premiers chiffres de $\sqrt{2}$.
La fonction `rec` est donc plus rapide que `dicho`.