Programmer en Python
Licence 2, Mathématiques
V. Monbet
TD 2.

1 Applications directes du cours

Préambule, voici des exemples de formatage et d'utilisation avancée de la fonction `print`.

```
>>> for x in range(1,4):
...     print '{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x)
...
 1   1    1
 2   4    8
 3   9   27
```

```
>>> import math
>>> print 'The value of PI is approximately {0:.3f}.'.format(math.pi)
The value of PI is approximately 3.142.
```

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print '{0:10} ==> {1:10d}'.format(name, phone)
...
Jack       ==>       4098
Dcab       ==>       7678
Sjoerd     ==>       4127
```

1. Que dessinent les programmes suivant ?

   ```
   def mystere1(n):
      for k in range(n, 0,-1):
         print('*'*k)
   mystere1(5)
   ```

   La première ligne du programme ci-dessous permet d'utiliser la syntaxe de la fonction `print` de Python 3 en Python 2.

```
from __future__ import print_function
def mystere2(n):
    for k in range(1, n+1):
       print(' '*(n-k), '* '*k, sep='')
mystere2(5)
```

2. Le Talkhys est un traité d'arithmétique d'Ibn Albanna, mathématicien marocain de la première moitié du XIIIe siècle. On y trouve un certain nombre d'identités remarquables.

   (a) Quelles sont les identités reproduites par la fonction ci-dessous

```
def talkhys2():
    n = 0
    for k in range(1, 10):
        n = 10 * n + k
        print('8 x {:<9} + {} = {:<9}'.format(n, k, 8*n+k))
talkhys2()
```

   (b) Ecrire une fonction `talkhys` qui permette de reproduire la table

9 x 1	+ 2 = 11
9 x 12	+ 3 = 111
9 x 123	+ 4 = 1111
9 x 1234	+ 5 = 11111
9 x 12345	+ 6 = 111111
9 x 123456	+ 7 = 1111111
9 x 1234567	+ 8 = 11111111
9 x 12345678	+ 9 = 111111111
9 x 123456789	+ 10 = 1111111111

2 Représentation des nombres et égalité

La représentation des nombres réels dans un ordinateur conduit à des approximations. Par exemple

```
In [1]: 0.1 + 0.2
Out[1]: 0.30000000000000004

In [2]:(0.1+0.2)-0.3
Out[2]: 5.551115123125783e-17
```

Ce résultat est dû à la représentation dyadique des réels (voir ci-dessous).

Définition **Nombre décimal**
Un nombre décimal s'écrit $\frac{x}{10^n}$ où $x \in \mathbb{Z}$.

Définition **Nombre dyadique**
Un nombre dyadique s'écrit $\frac{x}{2^n}$ où $x \in \mathbb{Z}$.

Problème : un nombre décimal n'est pas forcément dyadique.
Conséquence :
- la conversion décimal (humain) vers dyadique (machine) peut causer une première approximation ;
- le calcul machine entre nombre dyadiques peut causer une seconde approximation.
Quand on utilise le type float, on utilise jamais l'égalité de valeur `==` ; on fera toujours intervenir une marge d'erreur (absolue ou relative).

```
In [3]: 0.1 + 0.2 == 0.3
Out[3]: False

In [4]: abs(0.1 + 0.2 - 0.3) <= 1e-10
Out[4]: True
```

Inconvénient : une valeur choisie dans l'absolu parce qu'elle nous parait petite peut se révéler trop grande lorsque les nombres à comparer sont eux-même très petits ou trop petite lorsque les nombres sont très grands.
- La tolérance absolue consiste à fixer $\epsilon > 0$ et à convenir que x et y sont proches dès lors que $|x - y| < \epsilon$.
• La tolérance relative consiste à fixer$\epsilon > 0$ et à convenir que x et y sont proches dès lors que $|x - y| < \epsilon|y|$. L'inconvénient de ce choix est que la relation n'est pas symétrique.
Dans le module `numpy` la fonction `isclose(x,y)` renvoie `true` si $|x - y| < \epsilon_a + \epsilon_r|y|$.
Dans le module `math` la fonction `isclose(x,y)` renvoie `true` si $|x-y| < \epsilon_a+\epsilon_r max(|x|, |y|)$.

Exercice. Equation linéaire du second degré

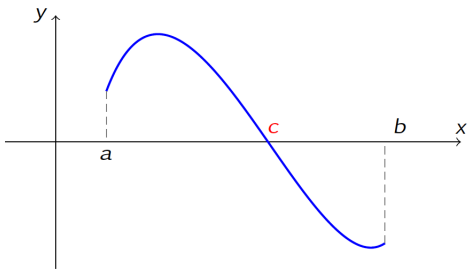
1. Écrire une fonction `solve(a,b,c)` qui prend en argument les valeurs a,b puis c pour désigner le polynôme $P(x) = ax^2 + bx + c$
   et qui retourne la ou les racines réelles du polynôme ou, le cas échéant, un message indiquant que ce polynôme n'a pas de racine réelle.
2. Expliquer les résultats suivants.

   ```
   >>> solve(0.01, 0.2, 1)
   deux racines simples -10.0000001317 et -9.99999986829
   >>> solve(0.011025, 0.21, 1)
   pas de solution
   ```

3. Lorsque $\Delta = b^2 - 4ac$ est petit devant b^2 les deux racines sont quasiment confondues. Comment peut-on modifier la fonction pour éviter ces erreurs d'approximation ?

3 Recherche d'une racine par dichotomie

Soit $f : [a, b] \to \mathbb{R}$ continue telle que $f(a)f(b) \leq 0$.



1. Prouver que f admet au moins une racine.
2. On considère le cas d'une racine unique. Montrer que suivant le signe de $f\left(\frac{a+b}{2}\right)$ l'une des deux relations est vérifiée

$$f(a)f\left(\frac{a+b}{2}\right) \leq 0 \text{ ou } f\left(\frac{a+b}{2}\right)f(b) \leq 0$$

3. On définie deux suites $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ par $u_0 = a$, $v_0 = b$ et

$$(u_{n+1}, v_{n+1}) = \begin{cases} (u_n, m) & \text{si } f(u_n)f(m) \leq 0 \\ (m, v_n) & sinon \end{cases} \qquad \text{avec} m = \frac{u_n + v_n}{2}$$

   Montrer que ces deux suites sont convergentes et qu'elles admettent la même limite qui est la solution de l'équation $f(x) = 0$.
4. Proposer une fonction `dicho` pernant comme arguments `f,a,b,epsilon` qui itère les deux suites $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ jusqu'à convergence des suites.
   Exemples de résultats

```
>>> from numpy import sin
>>> dicho(sin, 3, 4)
3.141592653589214
>>> dicho(lambda x: x * x - 2, 1, 2)
1.4142135623724243
```

5. Analyser la terminaison de l'algorithme. Et, pour ceux qui vont vite, déterminer le coût de l'algorithme en fonction de la précision p choisie $\epsilon = 10^{-p}$. On définira le coût comme le nombre d'itérations de la boucle conditionnelle.
6. Proposer une adaptation de la fonction `dicho` qui permette d'ajouter un nombre maximal d'itérations avant d'aboutir à un échec.

Remarque, la fonction `bisect` du module `scipy.optimize` réalise une recherche dichotomique.

4 Recherche d'une racine par dichotomie

L'idée de la méthode de la sécante est très simple : pour une fonction f continue sur un intervalle $[a, b]$, et vérifiant $f(a) < 0$, $f(b) > 0$,on trace le segment $[AB]$ où $A = (a, f(a))$ et $B = (b, f(b))$.Si le segment reste au-dessus du graphe de f alors la fonction s'annule

sur l'intervalle $[a', b]$ où $(a', 0)$ est le point d'intersection de la droite (AB) avec l'axe des abscisses. La droite (AB) s'appelle la sécante. On recommence en partant maintenant de l'intervalle $[a', b]$ pour obtenir une valeur a''.

Proposition
Soit $f : [a, b] \to$ une fonction continue, strictement croissante et convexe telle que $f(a) < 0$, $f(b) > 0$. Alors la suite définie par

$$a_0 = a \quad \text{et } a_{n+1} = a_n - \frac{b - a_n}{f(b) - f(a_n)} f(a_n)$$

est croissante et converge vers la solution ℓ de $(f(x) = 0)$.

1. Faire un dessin représentant la méthode de la sécante.
2. Justifier la construction de la suite récurrente.
3. Montrer que (a_n) est croissante. On pourra commencer par montrer par récurrence que $f(a_n) \leq 0$.
4. Prouver la convergence de (a_n) et déterminer sa limite.
5. Écrire une fonction `secante` pour implémenter la méthode de la sécante.