Programmer en Python
Licence 2, Mathématiques
V. Monbet
TP2 - Recherche de racines

Exercice 1 - Retour sur le TD

Tester les codes écrits dans le TD2, sections 1 et 2.

Exercice 2 - Recherche de racine par dichotomie

Préambule - La librarie `matplotlib.pyplot` permet de tracer des courbes. Par exemple

```
import matplotlib.pyplot as plt
import numpy as np
import math
# Tracé d'une courbe
tt=np.linspace(0,math.pi*2,100)
plt.plot(tt,np.sin(tt))
plt.grid()
# Tracé de points
t1=np.linspace(0,math.pi*2,4)
plt.plot(t1,np.sin(t1),'r+') # 'r' pour rouge
```

1. Nous considérons la fonction f qui à $x \in [0,5]$ associe x^2-1. Tracer cette fonction.
2. Implémenter la méthode de la recherche de racine par Dichotomie et l'appliquer à la recherche de la racine de f sur [0,5].
   Vérifier que la précision de la solution est compatible avec la précision que vous avez choisie.
   Matérialiser la solution par un point rouge sur le graphe de la courbe.
3. Trouver la racine du polynôme

   $$p(x) = x^3 - 2x - 5$$

   sur l'intervalle [-5,5].
   Vous tracerez le polynôme et matérialiserez la solution sur le graphe.

Exercice 3 - Recherche de racine par la méthode de Newton Cas d'une dérivée connue

La méthode de recherche de racine par dichotomie n'est pas très efficace. Elle a de plus l'inconvénient de nécessiter la définition d'un intervalle de recherche.
La méthode de Newton est plus performante. Elle repose sur un développement de Taylor à l'ordre 1. En effet, on rappelle que pour toute fonction f dérivable,

$$f(x) \simeq f(x_0) - f'(x_0)(x - x_0)$$

Pour trouver un zéro de cette fonction d'approximation, il suffit de calculer l'intersection de la droite tangente avec l'axe des abscisses, c'est-à-dire résoudre l'équation affine :

$$f(x_0) + f'(x_0)(x - x_0) = 0$$

La solution

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

est plus proche de la racine que x_0.
Ainsi, la méthode de Newton consiste à construire la suite

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}$$

et à l'itérer jusqu'à convergence.

1. Implémenter l'algorithme de Newton pour trouver la racine d'une fonction réelle dérivable. La fonction

   `newton(x0,f,df,epsilon=1e-12,maxiter=30):`

   devra retourner la solution ainsi que le nombre d'itérations réalisées. Vous ferez attention au critère d'arrêt (convergence).
2. Tester cette fonction pour trouver la racine du polynôme

   $$p(x) = x^3 - 2x - 5$$

   Comparer le résultat à celui de la recherche par dichotomie.
3. Pour chacun des deux algorithmes, compter le nombre d'itérations nécessaire pour converger (en fixant une précision commune).
4. Chercher la (ou les) racines de l'équation

   $$\cos(x) - x^3 = 0.$$

Exercice 4 - Recherche de racine par la méthode de Newton Cas d'une dérivée inconnue

Il arrive parfois que la dérivée (ou la matrice jacobienne pour un système d'équations à plusieurs variables) de la fonction f soit coûteuse à calculer. La dérivée peut alors être approchée au moyen de différences finies. Par exemple, en approchant la dérivée $f'(x_k)$ par

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

on obtient la méthode de la sécante.

1. Tracer sur une même figure la dérivée explicite du polynôme $p(x) = x^3 - 2x - 5$ et son approximation.
   Idem pour $\cos(x) - x^3$.
2. Écrire une fonction `secante`, inspirée de la fonction `newton` qui permette de chercher la racine d'une fonction sans calculer explicitement la dérivée.
3. Utiliser la fonction `secante` pour approcher les racines de

   $$x^3 - 2x - 5 = 0$$

   et

   $$\cos(x) - x^3 = 0$$

4. Il existe bien sûr des algorithmes de recherche de zéros qui sont déjà implémentés dans Python. Exemple :

   ```
   from scipy.optimize import fsolve
   def func(x):
       return x + 2*np.cos(x)
   x0 = fsolve(func, 0.3)
   print(x0)
   x = np.linspace(-5,5,100)
   plt.plot(x,func(x))
   plt.grid()
   plt.plot(x0,func(x0),'r*')
   ```

   Utiliser la fonction `fsolve` pour les courbes définies plus haut et comparer les solutions obtenues avec les votres.

Exercice 5 - Temps d'exécution

1. Comparer les temps d'exécution de différentes façons de créer une liste. Quelle est la méthode la plus rapide ?

   ```
   import time
   # création d'une liste de 5 000 000 d'éléments
   # (à adapter suivant la vitesse de vos machines)
   taille = 5000000
   ```

```
print "Création d'une liste avec {0:8d} élements".format(taille)
toto = range( taille )
# la variable a accède à un élément de la liste
# méthode 1
start = time.time()
for i in range( len(toto) ):
    a = toto[i]
print "méthode 1 (for in range) : {0:2f} sec".format(time.time() - start)
# méthode 2
start = time.time()
for ele in toto:
    a = ele
print "méthode 1 (for in ) : {0:2f} sec".format(time.time() - start)
# méthode 3
start = time.time()
for i in xrange( len(toto) ):
    a = toto[i]
print "méthode 1 (for in xrange) : {0:2f} sec".format(time.time() - start)
# méthode 4
start = time.time()
for idx, ele in enumerate( toto ):
    a = ele
print "méthode 1 (for in enumerate) : {0:2f} sec".format(time.time() - start)
```

2. Utiliser cette technique pour comparer les temps d'exécution des trois algorithmes de recherche de racine que vous avez implémentés.

Exercice 6 - Newton pour une fonction à plusieurs variables

En pratique, on travaille rarement avec des fonctions à une seule variable.

1. Adapter la fonction `newton` pour la recherche de zéros d'une fonction à plusieurs variables.
2. Tester votre programme pour trouver une racine de la fonction

$$x^2 + y^2 - 1 = 0$$

3. Example avec la fonction `fsolve`

```
def func2(x):
    out = [x[0]*np.cos(x[1]) - 4]
    out.append(x[1]*x[0] - x[1] - 5)
    return out
x02 = fsolve(func2, [1, 1])
print x02
```

Exercice 7 - ROT13

PYTHON offre une méthode simple pour accéder aux caractères contenus dans une chaîne : chaque caractère est accessible directement par son rang dans la chaîne (numéroté à partir de 0), encadré par des crochets. Par exemple, si `s = 'abcdefghi'` alors `s[0] = 'a', s[1] = 'b', . . ., s[8] = 'i'`. À l'inverse, la méthode index appliquée à une chaîne de caractères `s` retourne le rang de la première apparition de ce caractère dans s. Par exemple, `s.index('c')` 2=, `s.index('g') = 6`, mais `s.index('k')` déclenche l'exception `ValueError` puisque le caractère `'k'` n'est pas présent dans s.

Le ROT13 (rotate by 13 places) est un cas particulier du chiffre de César ; comme son nom l'indique, il s'agit d'un décalage de 13 caractères de chaque lettre du texte à chiffrer : A devient N, B devient O, C devient P, etc. Son principal aspect pratique est que le codage et le décodage se réalisent exactement de la même manière, puisque notre alphabet contient 26 lettres.
ROT13 est parfois utilisé dans les forums en ligne comme un moyen de masquer la réponse à une énigme, un spoiler ou encore une expression grossière, et on trouve sur le net de nombreux sites se proposant de coder/décoder une phrase en suivant ce principe.
Définir une fonction nommée `rot13` qui prend en argument une chaîne de caractères et retourne cette même chaîne codée en ROT13. On conviendra que :
– les caractères `a, b, c, . . ., y, z` seront transformés en `n, o, p, . . ., l, m` ;
– tous les autres caractères (espaces, ponctuation, chiffres, caractères accentués, . . .) ne seront pas modifiés.
Indication. Définir une chaîne de caractères `alph` 'abcdefghijklmnopqrstuvwxyz'= et utiliser le fait que si `c` est un caractère et `s` une chaîne de caractères, l'instruction `c in s` renvoie un booléen caractérisant la présence ou non de `c` dans `s`.
Utilisez votre fonction pour connaître la réponse à l'énigme suivante :
Quelle est la différence entre un informaticien et une personne normale ?
har crefbaar abeznyr crafr dh'ha xvyb-bpgrg rfg étny à 1000 bpgrgf, ha vasbezngvpvra rfg pbainvaph dh'ha xvybzèger rfg étny à 1024 zègerf.

Et pour les plus rapides

Exercice 8 - suite de CONWAY

Les premiers termes de la suite de CONWAY sont : 1, 11, 21, 1211, 111221,$\cdots$, chaque terme étant obtenu en lisant à haute voix le terme précédent (c'est pourquoi Conway avait baptisé cette suite *Look and say*). Par exemple, le terme 1211 se lit « un 1, un 2, deux 1 » donc le terme suivant est 111221.
Rédiger en PYTHON une fonction `lookandsay(n)` qui retourne le terme qui suit l'entier n dans cette énumération.

Remarque. Pour rédiger cette fonction, il est plus facile de manipuler les entiers sous la forme de chaînes de caractères. Par exemple, `lookandsay('1332')` retournera la chaîne '112312'.

Afficher à l'aide de cette fonction les 20 premiers termes de la suite de CONWAY .

Il a été démontré que si on note u_n le nombre de chiffres du $n$eme terme de cette suite alors le rapport $\frac{u_{n+1}}{u_n}$ tend vers une constante, appelée constante de Conway. Calculez-en une valeur approchée.

Une autre propriété remarquable de cette constante est qu'elle ne dépend pas de la valeur initiale (à l'exception de 22 qui produit une suite constante). Le vérifier expérimentalement en testant différentes valeurs initiales.