TP : résolution numérique d'équations

Nous avons vu en TD l'algorithme de la dichotomie ainsi que la méthode de Newton pour résoudre des équations du type $f(x) = 0$ avec f une fonction supposée régulière (au moins dérivable).

L'objet de ce TP est de mettre en oeuvre ces algorithmes pour résoudre $x^3+3x+1 = 0$. On pose $g(x) = x^3+3x+1$, on a vu en TD qu'il existe une unique solution à l'équation $g(x) = 0$, on la note α. On a également vu $-1 < \alpha < 0$.

1 Mise en oeuvre

1. Programmer une fonction `dichotomie` qui renvoie une valeur approchée avec une précision `epsilon` de la solution de l'équation `f(x)=0`, lorsqu'on sait que cette solution existe entre les nombres `a` et `b`.

   Remarque : dans un premier temps, on peut se limiter au cas où la fonction est croissante *(dans un premier temps seulement)*.

2. Programmer une fonction `mon_newton` qui renvoie une valeur approchée avec une précision `epsilon` de la solution de l'équation `f(x)=0`, lorsqu'on une première approximation « pas trop mauvaise » (c'est-à-dire assez correcte pour que la méthode de Newton converge, on ne prévoira pas de test de divergence).
   Remarque : dans un premier temps, on pourra prendre en paramètre le nombre d'itérations à effectuer plutôt que la condition de précision *(ensuite, il faudra réflechir à ce* `epsilon`*)*.

3. (Facultatif) Programmez une version récursive de la méthode de la dichotomie.

2 Comparaison des différentes fonctions

1. Programmer des fonctions `g` et `dg` qui prennent en paramètre un nombre x et renvoient respéctivement $g(x)$ et $g'(x)$.

2. On cherche à comparer la qualité des différentes fonctions. Sachant qu'elles fournissent toutes des approximations ayant la même précision, que va-t-on comparer ?

3. En guise de résultat, on veut un graphique. Que va-t-on prendre pour les axes ?

4. On peut mesurer le temps d'exécution d'une tâche en faisant la différence entre le temps mesuré avant et après son exécution. La fonction `time()` du module `time` (qu'il faudra donc importer) permet cela : elle renvoie le temps en secondes écoulé depuis le 1er janvier à 0h00m00s.

   Programmez une procédure (c'est-à-dire une fonction qui ne renvoie rien) `compareDichoNewton()` qui provoque l'affichage d'un graphique qui permet de comparer les algorithmes de la dichotomie et de Newton (on prendra comme premier encadrement $-1 < \alpha < 0$ pour la dichotomie, comme premier terme 0 pour la méthode de Newton).
   Remarque : si votre graphique a la précision en abscisse, demandez-vous où sont représentées les meilleurs approximations. Est-ce ce que l'on souhaite ? Comment régler ce problème ?

5. On va, à présent comparer notre implémentation de la méthode de Newton à la fonction `newton` fournie dans le module `scipy.optimize`. Vous consulterez la documentation de cette fonction pour voir quels sont ces paramètres. En particulier, remarquez que la donnée de la dérivée est optionnelle. Selon vous, quel algorithme est alors mis en oeuvre ?

   Proposer un graphique qui illustre les efficacités comparéees de notre implémentation, de celle fournie par Scipy (qu'on testera avec et sans la dérivée).

Rendre des figures `pyplot` présentables
`matplotlib.pyplot` est importé sous le préfixe `plt`
Les commandes `plt.xlabel()` et `plt.ylabel()` qui prennent en paramètre une chaîne de caractères permettent de légender les axes du graphique.
Lorsqu'on crée le graphique, l'ajout d'un `label` permet d'indiquer à quoi correspond la courbe tracée.
`plt.legend()` provoque l'affichage des `label` qu'on a prévu lors de la création du graphique.