TP1
Résolution numérique de $f(x) = 0$

3 octobre 2017

On reprend le TP précédent, dont on pourra s'inspirer pour la syntaxe python. On écrira le code dans un fichier `tp1.py`. Pour exécuter le code, ouvrir un terminal linux, saisir `ipython` et valider. Un terminal ipython se substitue au terminal linux ; à l'intérieur saisir `run tp1.py` et valider.

1. Création d'une fonction simple et représentation graphique ; notions python abordées : fonction python `def`, module `matplotlib`.
   (a) Ecrire la fonction $f(x) = x^2 - x - 1$ sous forme d'une fonction python.
   (b) Représentation graphique sur l'intervalle $[-1, 2]$; utiliser le module `matplotlib.pyplot`
   (c) Observer la présence de deux zéros et en donner une estimation grossière.
2. Recherche d'un zéro d'une fonction au moyen de la **méthode du point fixe** ; notion python abordée : boucle `for`.
   (a) Vérifier algébriquement que les points fixes de g sont les zéros de f. Ecrire la fonction $g(x) = 1 + 1/x$ sous forme d'une fonction python.
   (b) Calculer les 25 premiers termes de la suite définie par $x_{n+1} = g(x_n)$ et $x_0 = 1.0$ et enregistrer ces termes dans une liste.
   (c) Observer la convergence de cette suite vers l'un des deux zéros de f.
   (d) Dessiner, sur papier en repère orthonormé, la fonction g ainsi que la première bisectrice. Expliquer à partir du graphique la convergence observée.
3. Implémentation d'une fonction python `point_fixe`
   (a) Ecrire une fonction python `point_fixe` qui prend en arguments une fonction g, une valeur initiale x_0, un réel positif ϵ, et qui renvoie une approximation r d'un point fixe de la fonction g. Condition d'arrêt : $|x_{n+1} - x_n| < \epsilon$
   (b) Tests
      i. Tester `point_fixe` avec $g(x) = 1 + 1/x, x_0 = 1.6, \epsilon = 10^{-12}$. Vers quelle racine y-a-t'il convergence ?
      ii. Tester `point_fixe` avec $g(x) = 1 + 1/x, x_0 = -0.6, \epsilon = 10^{-12}$. Vers quelle racine y-a-t'il convergence ?
      iii. Sur le graphique, examiner la pente de g aux points fixes. Un point fixe r d'une fonction g tel que $|g'(r)| < 1$ est dit attractif ; si $|g'(r)| > 1$ il est dit répulsif. A partir de ça, expliquer le résultat des tests ci-dessus.
4. On veut, ici encore, calculer un zéro d'une fonction f. La **méthode de Newton** consiste à appliquer la méthode du point fixe à la fonction $g(x) = x - \frac{f(x)}{f'(x)}$.
   (a) Vérifier algébriquement que les points fixes de g sont les zéros de f.
   (b) Interpréter géométriquement la méthode de Newton.

(c) Ecrire une fonction python `newton` qui prend en arguments une fonction f, sa dérivée df, une valeur initiale x_0, un réel positif ϵ, et qui renvoie une approximation r d'une racine de la fonction f. Condition d'arrêt : $|x_{n+1} - x_n| < \epsilon$.

(d) Tests

i. Tester `newton` avec $f(x) = x^2 - x - 1, x_0 = 1.0, \epsilon = 10^{-12}$

ii. Tester `newton` avec $f(x) = x^2 - x - 1, x_0 = -1.0, \epsilon = 10^{-12}$

5. Toujours à la recherche des zéros de f La **méthode de la sécante** est une méthode itérative où chaque approximation est construite à partir des deux approximations précédentes. On doit donc partir de deux valeurs initiales distinctes, x_0, x_1 (en général les bornes d'un encadrement de la racine cherchée), puis on calcule par récurrence les termes de la suite $x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$. L'avantage sur la méthode de Newton est qu'on n'a pas besoin de la dérivée de f.

(a) Vérifier algébriquement que les points fixes de g sont les zéros de f.

(b) Interpréter géométriquement la méthode de la sécante.

(c) Ecrire une fonction python `secante` qui prend en arguments une fonction f, deux valeurs initiales x_0, x_1, un réel positif ϵ, et qui renvoie une approximation r d'une racine de la fonction f. Condition d'arrêt : $|x_{n+1} - x_n| < \epsilon$.

(d) Tests

i. Tester `secante` avec $f(x) = x^2 - x - 1, x_0 = 1.5, x_1 = 2.0, \epsilon = 10^{-12}$

ii. Tester `secante` avec $f(x) = x^2 - x - 1, x_0 = -1.0, x_1 = -0.5, \epsilon = 10^{-12}$

6. La **méthode de dichotomie** suppose que f est continue sur un intervalle (a, b) et change de signe sur cet intervalle ; on est donc assuré que f possède un zéro sur cet intervalle. Ensuite on coupe (a, b) en deux et on garde celui des deux intervalles où f change de signe. On obtient donc un nouvel encadrement a, b deux fois plus petit. On répète l'opération jusqu'à obtenir la précision souhaitée.

(a) Ecrire une fonction python `dichotomie` qui prend en arguments une fonction f, deux valeurs initiales a, b, un réel positif ϵ, et qui renvoie un encadrement a, b d'une racine de la fonction f. Condition d'arrêt : $|b - a| < \epsilon$.

(b) Tests

i. Tester `dichotomie` avec $f(x) = x^2 - x - 1, a = 1.5, b = 2.0, \epsilon = 10^{-12}$

ii. Tester `dichotomie` avec $f(x) = x^2 - x - 1, a = -1.0, b = 0.0, \epsilon = 10^{-12}$

7. Examinons maintenant la vitesse de convergence de ces méthodes.

(a) Dans chacune des méthodes, incorporer un compteur qui compte le nombre d'itérations `nbiter` effectuées et placer `nbiter` dans le return de la fonction. Par exemple, le return de la fonction `newton` s'écrira `return r, nbiter`. Lorsqu'on appelera `newton`, on écrira donc `r, nbiter = newton(f, df, x0, epsi)`

(b) Pour chacune des méthodes, faire varier la valeur de ϵ et compter le nombre d'itérations effectuées. Reporter les résultats dans un tableau, par exemple :

	10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}
`point_fixe`					
`newton`					
`secante`					
`dichotomie`					

8. Rédiger le compte-rendu du TP1, un compte-rendu par binôme, dans l'un des formats (que l'on pourra combiner, si besoin) :

— papier
— ipython notebook (extension .ipynb)
— latex (extension .tex)
— libreoffice (extension .odt)