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

27 novembre 2017

Tous les fichiers créés dans ce TP devront être rangés dans un dossier nommé TP1. On écrira le code dans un fichier nommé `tp1.c`.

1. Programme `values.c` : création d'une fonction simple et écriture de ses valeurs dans un fichier texte.
   (a) dessiner sur papier la fonction $f(x) = x^3 - 3x + 1$ et localiser approximativement ses zéros.
   (b) Dans `values.c`, écrire une fonction `f` prenant en argument x, de type `double`, et renvoyant $x^3 - 3x + 1$.
   (c) Ecrire une fonction `write_values` prenant en arguments a, b, h , de type `double`, et qui écrit dans un fichier nommé `values.txt` tous les couples $x, f(x)$ (un couple par ligne), pour x variant sur $[a, b]$ avec un pas de h.
   (d) Ecrire la fonction `main` sous la forme

   ```
   int main(int argc, char *argv[]){
           double a, b, h;
           sscanf(argv[1], "%lf", &a);
           sscanf(argv[2], "%lf", &b);
           sscanf(argv[3], "%lf", &h);
           write_values(a, b, h);
           return EXIT_SUCCESS;
   }
   ```

   On dira que `main` prend en arguments a, b, h.
   (e) En ligne de commande, compiler `values.c` sous la forme `gcc values.c -o values` et vérifier que l'exécutable `values` est créé, puis exécuter `./values a b h`, avec a, b, h des valeurs au choix ; vérifier la création du fichier `values.txt`.
   (f) A l'aide du fichier `values.txt`, donner des encadrements des zéros de f à la précision $h = 0.01$.
2. Programme `newton.c` : implémentation de la **méthode de Newton**.
   (a) Soit f une fonction quelconque et x_0 un réel, appelé valeur initiale ; soit T la tangente à la courbe représentative de f, au point $(x_0, f(x_0))$; T coupe l'axe des x au point d'abscisse x_1 ; exprimer x_1 à l'aide des quantités $x_0, f(x_0), f'(x_0)$.
   (b) Illustrer la construction ci-dessus sur un dessin avec $f(x) = x^3 - 3x + 1$ et $x_0 = 2.0$; que vaut x_1 ?
   (c) La méthode de Newton consiste à prendre x_1 comme nouvelle valeur initiale, et à répéter le processus ci-dessus ; on obtient ainsi des valeurs $x_2, x_3, \cdots$, etc ; sur le dessin précédent, vers quoi semble converger la suite x_n ?
   (d) Dans `newton.c`, écrire une fonction `f` prenant en argument x, de type `double` et renvoyant x^3-3x+1, puis écrire une fonction `df` représentant la dérivée de f.
   (e) Ecrire une fonction `newton` qui prend en arguments x_0, ϵ, de type `double` et qui écrit dans un fichier nommé `newton.txt` les couples $x_n, |x_n - x_{n-1}|$ (un couple par ligne ; la quantité $|x_n - x_{n-1}|$ s'appelle

erreur de la méthode de Newton à l'étape n) ; condition d'arrêt : $|x_n - x_{n-1}| < \epsilon$; la première ligne, correspondant à $n = 0$, ne contiendra que la valeur x_0.

3. 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 $f(x) = x^2 - x - 1, x_0 = -1.0, x_1 = -0.5, \epsilon = 10^{-12}$

4. 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 $f(x) = x^2 - x - 1, a = -1.0, b = 0.0, \epsilon = 10^{-12}$

5. 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`					

6. 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)