

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

15 janvier 2018

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` les couples  $x, f(x)$  - un couple par ligne -, pour  $x$  variant sur  $[a, b]$  avec un pas de  $h$ .
  - (d) Ecrire une fonction `main` prenant en arguments `a`, `b`, `h` (ces arguments seront passés en ligne de commande au moment de l'exécution du programme); la fonction `main` fera ensuite appel à `write_values` avec les arguments `a`, `b`, `h`.
  - (e) En ligne de commande, compiler `values.c` sous la forme `gcc values.c -o values`, vérifier que le fichier 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) Examiner le contenu du fichier `values.txt` et donner des encadrements des zéros de  $f$  à la précision  $h = 0.1$ .
2. Programme `dichotomie.c` : Soit  $f$  une fonction continue sur un intervalle  $(a, b)$ ; la **méthode de dichotomie** suppose que  $f$  change de signe sur l'intervalle; on est donc assuré que  $f$  possède un zéro sur cet intervalle. Ensuite on coupe l'intervalle  $(a, b)$  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) Illustrer quelques étapes de la construction précédente sur un dessin avec  $f(x) = x^3 - 3x + 1$ .
  - (b)
    - i. Dans `dichotomie.c`, écrire la fonction `f` ci-dessus.
    - ii. Ecrire une fonction `dichotomie` qui prend en arguments `a`, `b`, `epsilon` de type `double` et qui écrit dans un fichier `dichotomie.txt` les encadrements successifs  $(a, b)$  - un couple par ligne; condition d'arrêt :  $|b - a| < \epsilon$ .
  - (c) Compiler `gcc dichotomie.c -o dichotomie`, exécuter `./dichotomie a b epsilon` pour différentes valeurs de `a` `b` `epsilon` et vérifier vos résultats dans le fichier `dichotomie.txt`.
3. Programme `newton.c` : **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  $x_2, x_3, \dots$ , etc; sur le dessin précédent, vers quoi semble converger la suite  $x_n$ ?
  - (d) Dans `newton.c`, écrire la fonction `f` habituelle, puis écrire la fonction `df`, dérivée de  $f$ .

- (e) Ecrire une fonction `newton` qui prend en arguments `x0`, `epsilon`, 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  $x_0$ .
- (f) Compiler `gcc newton.c -o newton`, exécuter `./newton x0 epsilon` pour différentes valeurs de `x0 epsilon` et vérifier vos résultats dans le fichier `newton.txt`.
4. Programme `newton_numdiff.c` : méthode de Newton avec **dérivée numérique**.  
 Pour éviter le calcul à la main de la dérivée de  $f$ , qui peut s'avérer difficile, voire impossible, on peut utiliser l'approximation mathématique

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \quad (1)$$

où  $h$  est un réel positif fixé.

- (a) Sauvegarder le programme `newton.c` dans un fichier `newton_numdiff.c`
- (b) Dans `newton_numdiff.c`, supprimer la fonction `df` et ajouter la constante `h = 1.0E-8`.
- (c) Modifier la fonction `newton` en accord avec l'approximation (1); les résultats du calcul seront écrits dans un fichier `newton_numdiff.txt`.
- (d) Compiler et exécuter `newton_numdiff` et comparer les résultats avec ceux de `newton`.
5. Programme `secante.c` : **méthode de la sécante**.
- (a) Soit  $f$  une fonction et  $x_0, x_1$  deux réels distincts - les valeurs initiales; soit  $T$  la droite - appelée sécante - s'appuyant sur les points  $(x_0, f(x_0))$  et  $(x_1, f(x_1))$ ;  $T$  coupe l'axe des  $x$  au point d'abscisse  $x_2$ ; exprimer  $x_2$  à l'aide des quantités  $x_0, f(x_0), x_1, f(x_1)$ .
- (b) Illustrer cette construction avec  $f(x) = x^3 - 3x + 1$  et  $x_0 = 1.0, x_1 = 2.0$ ; que vaut  $x_2$ ?
- (c) La méthode de la sécante consiste à prendre  $x_1, x_2$  comme nouvelles valeurs initiales, et à répéter le processus; on obtient ainsi des valeurs  $x_2, x_3, \dots$ , etc; sur le dessin précédent, vers quoi semble converger la suite  $x_n$ ?
- (d) Dans `secante.c`, outre la fonction `f`, écrire une fonction `secante` qui prend en arguments `x0`, `x1`, `epsilon` et qui écrit dans un fichier `secante.txt` les couples  $x_n, |x_n - x_{n-1}|$ . Condition d'arrêt :  $|x_n - x_{n-1}| < \epsilon$ .
- (e) Compiler `gcc secante.c -o secante`, exécuter `./secante x0 x1 epsilon` pour différentes valeurs `x0 x1 epsilon` et vérifier vos résultats dans `secante.txt`.
6. (a) Choisir un zéro de  $f$ ; pour chacune des méthodes présentées, fixer des valeurs initiales, faire varier la valeur de  $\epsilon$  et compter le nombre d'itérations effectuées; reporter les résultats dans le tableau :

$\epsilon$	$10^{-3}$	$10^{-6}$	$10^{-9}$	$10^{-12}$	$10^{-15}$
dichotomie					
newton					
sécante					

- (b) Rédiger un court compte-rendu, contenant explications, commentaires, dessins, table.
7. Programme `newton2.c` : **méthode de Newton, multi-dimensionnelle**.  
 Dans cette partie nous montrons comment généraliser la méthode de Newton pour calculer un zéro d'un système de plusieurs équations à plusieurs inconnues.
- (a) Dans cet exemple, on considère un système de deux équations à deux inconnues  $x_0, x_1$ ; on posera  $\mathbf{X} = (x_0, x_1)$ . Dans `newton2.c`, écrire la fonction `f` définie par  $f(x_0, x_1) = (x_0^2 + 2x_1^2 - 3.0, x_0x_1 - 1.0)$ ; la fonction `f` prendra en argument `X`, et retournera `fX`; `X` et `fX` sont des `arrays` de longueur 2 et de type `double`.
- (b) Ecrire la fonction `df`, dérivée de  $f$ ; mathématiquement, la dérivée de  $f$  au point  $(x_0, x_1)$  est la matrice, dite Jacobienne :

$$df(x) = \begin{pmatrix} \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial x_1} \\ \frac{\partial f_1}{\partial x_0} & \frac{\partial f_1}{\partial x_1} \end{pmatrix} \quad (2)$$

avec  $f_0 = x_0^2 + 2x_1^2 - 3.0$ ,  $f_1 = x_0x_1 - 1.0$ ; la fonction **df** prendra en argument **X** et retournera **dfX**, avec  $\mathbf{dfX} = [\frac{\partial f_0}{\partial x_0}, \frac{\partial f_0}{\partial x_1}, \frac{\partial f_1}{\partial x_0}, \frac{\partial f_1}{\partial x_1}]$ ; **X** et **dfX** sont des **arrays** de type **double**, l'un de longueur 2, l'autre de longueur 4.

- (c) La méthode de Newton consiste à choisir une valeur initiale  $X_0 \in \mathbb{R}^2$  et à calculer la suite définie par la récurrence  $X_n = X_{n-1} - df(X_{n-1})^{-1}X_{n-1}$ . Pour de nombreux choix de  $X_0$ , la méthode de Newton converge, très rapidement, vers un zéro de  $f$ ; il peut arriver cependant, pour certains choix de  $X_0$ , que la méthode ne converge pas.
  - i. Ecrire une fonction **inverse**, qui prend une matrice **mat** carrée de taille 2, **array** de type **double** et de longueur 4 et qui renvoie son inverse **imat**.
  - ii. Ecrire une fonction **multiply**, qui prend une matrice **mat**, ainsi qu'un vecteur **v** et qui renvoie le produit de **mat** par **v**; on notera **w** ce produit; **v** et **w** sont des **arrays** de type **double** et de longueur 2.
- (d) Ecrire une fonction **newton2** qui prend en arguments une valeur initiale **X0** et un scalaire **epsilon**; **newton2** écrira dans un fichier nommé **newton2.txt** les couples  $X_n$  ainsi que l'erreur  $\|X_n - X_{n-1}\|$  - il y aura donc valeurs numériques par ligne, et seulement deux valeurs sur la première ligne; en pratique, pour l'erreur  $\|X_n - X_{n-1}\|$ , on pourra prendre au choix  $\sqrt{(X_n[0] - X_{n-1}[0])^2 + (X_n[1] - X_{n-1}[1])^2}$  ou  $\|X_n - X_{n-1}\| = |X_n[0] - X_{n-1}[0]| + |X_n[1] - X_{n-1}[1]|$ , si la fonction racine carrée n'est pas disponible.
- (e) Compiler `gcc newton2.c -o newton2`, exécuter `./newton2 X0[0] X0[1] epsilon` pour différentes valeurs de **X0** **epsilon** et vérifier vos résultats dans le fichier **newton2.txt**.