Université de Rennes - ESIR

Module MATH-S2 : Mathématiques

CUPGE1 2022/2023 - Groupe 2

Compte rendu de TP : Résolution d'équations non linéaires

Auteurs :
- Romain COURAUD
- Balthazar GIROT

21 mai 2023

Table des matières

Nous attestons sur l'honneur que le contenu de ce document est original et est issu de nos réflexions personnelles. Toutes les sources d'informations utilisées et les citations aux auteurs on été mentionnées conformément aux usages en vigueur et répertoriées dans la partie "Références".

1 Introduction

L'objectif de ce TP de Mathématiques était, dans un premier temps de trouver la racine du polynôme $P(x) = x^3 + 4x^2 - 10$ se trouvant dans l'intervalle $[0, 5]$ à l'aide de plusieurs méthodes algorithmiques de recherche de zéro d'une fonction. Puis dans un second temps, de comparer ces différentes méthodes selon plusieurs critères.

Ainsi, nous avons construit et implémenté des algorithmes sous MATLAB et Octave à partir des méthodes suivantes :

- Méthode de la dichotomie
- Méthode de la trichotomie
- Méthode du point fixe
- Méthode de Newton
- Méthode de la sécante
- Méthode de la fausse position

Ces méthodes permettent toutes d'approcher le zéro d'une fonction, avec toutefois des performances différentes. Afin de les comparer, nous nous baserons sur 3 critères principaux : le **nombre d'itérations** requises pour approcher le zéro, la **vitesse moyenne** d'exécution de l'algorithme et **l'erreur** de l'approximation.

Rechercher la racine d'une fonction polynomiale revient à résoudre une équation non linéaire. Ainsi, les algorithmes présentés dans ce compte rendu de TP fonctionneront pour la recherche de zéro de toute fonction non linéaire.

L'intérêt principal des méthodes algorithmiques de recherche de zéro est d'offrir une approximation rapide et précise des racines d'une fonction dont il n'existe pas de solution analytique connue.

Les algorithmes présentés sont cependant dépendants de plusieurs facteurs. Tout d'abord, leur efficacité dépend fortement des conditions initiales. Dans ce TP, nous sélectionnerons les conditions initiales de manière à optimiser la convergence des algorithmes, choix que nous détaillerons pour chaque méthode. De plus, la vitesse d'exécution est dépendante – certes faiblement – des capacités de la machine sur laquelle sont effectués les tests. De ce fait, afin de comparer de façon significative les méthodes, tous les tests de vitesse présentés dans ce compte rendu ont été effectués sur la même machine.

2 Méthodes et matériels

Dans cette section, nous présenterons les méthodes étudiées et leur mise en oeuvre sous forme d'algorithme en pseudo-code. Vous retrouverez les fonctions codées en langage MATLAB dans le fichier .zip fourni avec ce compte-rendu, ou bien en suivant ce lien si vous n'avez pas accès au dossier.

Chaque fonction présentée prend en paramètres les informations nécessaires à la méthode qu'elle implémente et un handle `fun` représentant la fonction dont on cherche la racine. Cela permet de tester les algorithmes pour différentes fonctions. Elles renverront la valeur approchée du zéro qu'elles ont obtenu, le nombre d'itérations nécessaires pour l'atteindre et l'erreur par rapport au zéro réel de fonction (calculé soit à la main, soit avec `roots`, une méthode native de MATLAB renvoyant les racines d'un polynôme).
Afin de les étudier, nous récupérerons les informations (valeur approchée, nombre d'itérations, erreur relative et temps d'exécution moyen sur 100 exécutions) de chaque méthode de la façon suivante (codée en MATLAB) :

```
1  [dichotomic_value, nbIterDichotomie, errDichotomie] = dichotomic_func(fun , 0, 5, 10^-3,
       100, trueValue);
2
3  total_time = 0;
4  for i = 1:100
5      t_start = clock;
6      dichotomic_func(fun, 0, 5, 10^-3, 100, trueValue);
7      t_end = clock;
8      exec_time = etime(t_end, t_start);
9      total_time = total_time + exec_time;
10 end
11 averageTimeDichotomie = total_time/100;
12
13 disp(['Zero approche par dichotomie : ' num2str(dichotomic_value)]);
14 disp(['Erreur  : ' num2str(errDichotomie)]);
15 disp(['Nombre d iterations : ' num2str(nbIterDichotomie)]);
16 disp(['Temps moyen sur 100 executions : ' num2str(averageTimeDichotomie) 's' char(10)]);
```

Code 1 – Récupération des données d'une méthode

Il est à noter que par souci de simplicité, nous considérerons que toutes les fonctions étudiées admettent un zéro.
Une fois ces données récupérées, nous pourrons les comparer et les analyser (cf. Résultats et Discussions).

2.1 Méthode de la dichotomie

La méthode de la dichotomie est un algorithme de recherche d'un zéro d'une fonction. Elle implique de diviser successivement un intervalle en deux sous-intervalles et de conserver le sous-intervalle contenant la racine de la fonction. Après un certain nombre de divisions, l'algorithme converge vers le zéro exact de la fonction.
Pour implémenter notre algorithme, nous considérerons une fonction f continue sur $[a, b]$ telle que $f(a)$ et $f(b)$ sont de signes opposés et non nuls. D'après le théorème des valeurs intermédiaires, il existe un point c tel que $f(x) = 0$, qui correspond à notre racine. À chaque

itération, nous allons considérer le milieu c de l'intervalle $[a, b]$ comme une approximation du zéro de la fonction, puis déterminer si le zéro est dans $[a, c]$ ou dans $[c, b]$ pour réitérer la recherche dans ce nouvel intervalle contenant le zéro.
Ainsi, on peut écrire l'algorithme de dichotomie selon ce pseudo-code :

```
Entrée : fun, a, b, tol, iterMax, trueValue
Sortie: approximation de la racine, nombre d'itérations, erreur relative
Step 1 : i ← 1
Step 2 : TANT QUE i < iterMax ET |b - a| > tol FAIRE
Step 3 :    i ← i + 1
Step 4 :    c ← (a + b)/2
Step 5 :    fc ← fun(c)
Step 6 :    SI fc = 0
Step 7 :        BREAK
Step 8 :    FIN SI
Step 9 :    SI fun(a) * fc < 0
Step 10 :       b ← c
Step 11 :   SINON
Step 12 :       a ← c
Step 13 :   FIN SI
Step 14 : FIN TANT QUE
Step 15 : xfinal ← c
Step 16 : err ← |xfinal - trueValue|
```

Après avoir utilisé cet algorithme pour trouver le zéro (connu comme étant environ $1,36523$) de la fonction $f(x) = x^3 + 4x^2 - 10$, sur l'intervalle $[0, 5]$ avec pour tolérance 10^{-3} et en limitant les itérations à 100, nous obtenons :

- Approximation de la racine : $1,3654$
- Nombre d'itérations : 13
- Puissance de l'erreur : 10^{-4}
- Temps moyen d'exécution (sur 100 exécutions) : $0,324$ ms

Nous pouvons donc en conclure que l'algorithme converge bien vers une approximation de la racine de f.

2.2 Méthode de la trichotomie

La méthode de la trichotomie utilise un fonctionnement similaire à celui de la dichotomie. En effet, au lieu de diviser l'intervalle par 2, elle implique un division en 3 sous-intervalles. De la même façon que pour la dichotomie, on choisit parmi ces trois sous-intervalles celui qui contient le zéro et on réitère la méthode, jusqu'à converger vers le zéro réel de la fonction.
Ainsi, on peut écrire l'algorithme de trichotomie selon ce pseudo-code :

```
Entrée : fun, a, b, tol, iterMax, trueValue
Sortie: approximation de la racine, nombre d'itérations, erreur relative
Step 1 : i ← 1
Step 2 : TANT QUE i < iterMax ET |b - a| > tol FAIRE
```

```
Step 3 :    i ← i + 1
Step 4 :    c1 ← (a + b)/3
Step 5 :    c2 ← 2*(a + b)/3
Step 6 :    fc1 ← fun(c1)
Step 7 :    fc2 ← fun(c2)
Step 8 :    SI fc1 = 0
Step 9 :        xfinal ← c1
Step 10 :       err = 0
Step 11 :       EXIT
Step 12 :   SINON SI fc2 = 0
Step 13 :       xfinal ← c2
Step 14 :       err = 0
Step 15 :       EXIT
Step 16 :   FIN SI
Step 17 :   SI fun(a) * fc1 < 0
Step 18 :       b ← c1
Step 19 :   SINON SI fun(a) * fc2 < 0
Step 20 :       a ← c1
Step 21 :       b ← c2
Step 22 :   SINON
Step 23 :       a ← c2
Step 24 :   FIN SI
Step 25 : FIN TANT QUE
Step 26 : xfinal ← (a + b)/2
Step 27 : err ← |xfinal - trueValue|
```

Après avoir utilisé cet algorithme pour trouver le zéro (connu comme étant environ $1,36523$) de la fonction $f(x) = x^3 + 4x^2 - 10$, sur l'intervalle $[0, 5]$ avec pour tolérance 10^{-3} et en limitant les itérations à 100, nous obtenons :

- Approximation de la racine : $1,3653$
- Nombre d'itérations : 8
- Puissance de l'erreur : 10^{-5}
- Temps moyen d'exécution (sur 100 exécutions) : $0,285$ ms

Nous pouvons donc en conclure que l'algorithme converge bien vers une approximation de la racine de f.

2.3 Méthode du point fixe

La méthode du point fixe repose sur la notion de point fixe. Soit f une application d'un ensemble E dans lui-même, on dit qu'un élément x de E est un point fixe de f si $f(x) = x$. Nous utiliserons la méthode du point fixe pour calculer une approximation de la racine d'une fonction f donnée. Toutefois, pour assurer la convergence de l'algorithme, il est nécessaire que la fonction étudiée respecte des conditions :

- D'existence : Si g est une fonction continue sur $[a, b]$ et si $g(x) \in [a, b]$, alors g possède au moins un point fixe dans $[a, b]$.

— D'unicité : Si en outre g est dérivable sur $[a, b]$ et s'il existe une constante $K \in [0, 1[$ telle que $|g'(x)| \leq K$ pour tout $x \in [a, b]$, alors ce point fixe de $[a, b]$ est unique.

Nous choisirons donc des fonctions pouvant être utilisées dans l'algorithme du point fixe telles que :

- $f_1(x) = x^3 + 4x^2 - 10$
- $f_2(x) = x^4 + 2x^2 - x - 3$
- $f_3(x) = 6x^3 - x - 1$
- $f_4(x) = 3x^5 + x^2 - 2x$

Pour résoudre l'équation $f(x) = 0$, on fait intervenir une fonction g telle que si p est la solution de l'équation $f(x) = 0$, alors $g(p) = p$, puis nous déterminerons le point fixe de cette fonction g.

Pour cela, si g vérifie bien les conditions énoncées précédemment, on considère la suite $u_n+1 = g(u_n)$. Si (u_n) converge vers un élément l , alors l est un point fixe de g. Nous utiliserons cette propriété pour déterminer le point fixe de g, et par extension, la racine de f

Par exemple, associons une fonction g_n à chaque fonction f_n listé précédemment telle que $f_n(p) = 0 \iff g_n(p) = p$:

- $g_1(x) = \frac{\sqrt{10-x^3}}{2}$
- $g_2(x) = (3 + x - 2x^2)^{1/4}$
- $g_3(x) = (\frac{1}{6} + \frac{1}{6}x)^{1/3}$
- $g_4(x) = (-\frac{1}{3}x + \frac{2}{3}x)^{1/5}$

Avec toutes ces informations, on peut écrire l'algorithme du point fixe selon ce pseudo-code en prenant `fun` la fonction dont on cherche le point fixe :

```
Entrée : fun, p0, tol, iterMax, trueValue
Sortie: approximation de la racine, nombre d'itérations, erreur relative
Step 1 : i ← 1
Step 2 : x ← p0
Step 3 : TANT QUE i < iterMax FAIRE
Step 4 :    newX ← fun(x)
Step 5 :    SI |newX - x| < tol
Step 6 :        BREAK
Step 7 :    FIN SI
Step 8 :    i ← i + 1
Step 9 :    x ← newX
Step 10 : FIN TANT QUE
Step 11 : xfinal ← x
Step 12 : err ← |xfinal - trueValue|
```

Après avoir utilisé cet algorithme pour trouver le zéro (connu comme étant environ $1,36523$) de la fonction $f(x) = x^3 + 4x^2 - 10$, avec pour fonction associée $g_1(x) = \frac{\sqrt{10-x^3}}{2}$ dont on cherche le point fixe, en prenant $p_0 = 1$ avec pour tolérance 10^{-3} et en limitant les itérations à 100, nous obtenons :

— Approximation de la racine : $1,3654$
— Nombre d'itérations : 10
— Puissance de l'erreur : 10^{-4}
— Temps moyen d'exécution (sur 100 exécutions) : $0,439$ ms

Nous pouvons donc en conclure que l'algorithme converge bien vers une approximation de la racine de f.

2.4 Méthode de Newton

La méthode de Newton est une méthode itérative de recherche de zéro. Elle se base sur l'approximation d'une fonction par une tangente, et utilise cette tangente pour estimer la racine. Pour cela, on choisit une valeur initiale p_0 si possible relativement proche de la racine, puis on estime la racine de la fonction f en calculant $p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}$. On réitère l'opération en calculant p_2 à partir de p_1. On peut l'écrire sous la forme d'une suite (U_n) de terme initial u_0 définir par $u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}$.
Ainsi, on peut écrire l'algorithme de la méthode de Newton selon ce pseudo-code :

```
Entrée : fun, deriv, p0, tol, iterMax, trueValue
Sortie: approximation de la racine, nombre d'itérations, erreur relative
Step 1 : i ← 1
Step 2 : x ← p0
Step 3 : TANT QUE i < iterMax ET |fun(x)| > tol FAIRE
Step 4 :    i ← i + 1
Step 5 :    x = x - fun(x)/deriv(x)
Step 6 : FIN TANT QUE
Step 7 : xfinal ← x
Step 8 : err ← |xfinal - trueValue|
```

Par souci de simplicité et d'accessibilité, nous dériverons "à la main" la fonction `fun` pour obtenir `deriv`. Il est possible en MATLAB de dériver un handle tel que `fun` mais cela nécessite un package non natif.

Après avoir utilisé cet algorithme pour trouver le zéro (connu comme étant environ $1,36523$) de la fonction $f(x) = x^3 + 4x^2 - 10$, de dérivée $f'(x) = 3x^2 + 8x$, en prenant $p_0 = 5$, avec pour tolérance 10^{-3} et en limitant les itérations à 100, nous obtenons :

- Approximation de la racine : $1,3653$
- Nombre d'itérations : 5
- Puissance de l'erreur : 10^{-5}
- Temps moyen d'exécution (sur 100 exécutions) : $0,183$ ms

Nous pouvons donc en conclure que l'algorithme converge bien vers une approximation de la racine de f.

2.5 Méthode de la sécante

La méthode de la sécante est une méthode de recherche de zéro d'une fonction qui est relativement similaire à celle de Newton dans son expression. On considère deux points x_0 et x_1 si possible relativement proches du zéro de la fonction f. On trace un segment reliant $f(x_0)$ et $f(x_1)$, et l'intersection entre ce point x_0 l'axe de abscisses constituera l'approximation du zéro de f. Ce point a pour abscisse : $x_2 = x_1 - \frac{x_1 - x_0}{f(x_1) - f(x_0)}$. De ce fait, la méthode de la sécante est assez semblabe à celle de Newton, mais on remplace $f'(x_n)$ par $\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$. Il n'est pas nécessaire que le zéro de f soit dans l'intervalle $[x_0, x_1]$.

Ainsi, on peut écrire l'algorithme de la méthode de la sécante selon ce pseudo-code :

```
Entrée : fun, a, b, tol, iterMax, trueValue
Sortie: approximation de la racine, nombre d'itérations, erreur relative
Step 1 : i ← 1
Step 2 : x0 ← a
Step 3 : x1 ← b
Step 4 : TANT QUE i < iterMax ET |x1 - x0| > tol FAIRE
Step 5 :    i ← i + 1
Step 6 :    x2 ← x1 - fun(x1)*(x1 - x0)/(fun(x1) - fun(x0))
Step 7 :    x0 ← x1
Step 8 :    x1 ← x2
Step 9 : FIN TANT QUE
Step 10 : xfinal ← x1
Step 11 : err ← |xfinal - trueValue|
```

Après avoir utilisé cet algorithme pour trouver le zéro (connu comme étant environ $1,36523$) de la fonction $f(x) = x^3 + 4x^2 - 10$, avec $a = 0$ et $b = 5$, avec pour tolérance 10^-3 et en limitant les itérations à 100, nous obtenons :

- Approximation de la racine : $1,3652$
- Nombre d'itérations : 10
- Puissance de l'erreur : 10^-7
- Temps moyen d'exécution (sur 100 exécutions) : $0,541$ ms

Nous pouvons donc en conclure que l'algorithme converge bien vers une approximation de la racine de f.

2.6 Méthode de la fausse position

La méthode de la fausse position est semblable à celle de de Newton et de la sécante, mais apporte une subtilité. Dans la méthode de la sécante, le zéro de la fonction n'est pas nécessairement encadré par les valeurs x_0 etx_1. La méthode de la fausse position fait intervenir une redéfinition des bornes à chaque itération pour toujours encadrer le zéro.

Ainsi, on peut écrire l'algorithme de la fausse position selon ce pseudo-code :

```
Entrée : fun, a, b, tol, iterMax, trueValue
Sortie: approximation de la racine, nombre d'itérations, erreur relative
Step 1 : i ← 1
Step 2 : x0 ← a
Step 3 : x1 ← b
Step 4 : TANT QUE i < iterMax
Step 5 :    i ← i + 1
Step 6 :    (x0*fun(x1) - x1*fun(x0))/(fun(x1) - fun(x0))
Step 7 :    SI |fun(x2)| < tol
Step 8 :        BREAK
Step 9 :    FIN SI
Step 10 :   SI fun(x0) * fun(x2) < 0
Step 11 :       x1 ← x2
```

```
Step 12 :   SINON
Step 13 :       x0 ← x2
Step 14 :   FIN SI
Step 15 : FIN TANT QUE
Step 16 : xfinal ← x2
Step 17 : err ← |xfinal - trueValue|
```

Nous prenons le parti de définir `fun(x2) < 0` comme condition d'arrêt relative à la tolérance. Après avoir utilisé cet algorithme pour trouver le zéro (connu comme étant environ $1,36523$) de la fonction $f(x) = x^3 + 4x^2 - 10$, avec $a = 0$ et $b = 5$, avec pour tolérance 10^{-3} et en limitant les itérations à 100, nous obtenons :

- Approximation de la racine : $1,3652$
- Nombre d'itérations : 33
- Puissance de l'erreur : 10^{-5}
- Temps moyen d'exécution (sur 100 exécutions) : $0,502$ ms

Nous pouvons donc en conclure que l'algorithme converge bien vers une approximation de la racine de f.

3 Résultats

On regroupe les résultats obtenus pour la recherche de zéro de la fonction $f(x) = x^3+4x^2-10$ dans le tableau 1 :

Méthode	**Résultat**	**Nb Itérations**	**Vitesse moyenne**	**Puissance erreur**
Dichotomie	$1,3654$	13	$0,280$ ms	10^{-4}
Trichotomie	$1,3653$	8	$0,385$ ms	10^{-5}
Point fixe	$1,3654$	10	$0,439$ ms	10^{-4}
Newton	$1,3653$	5	$0,183$ ms	10^{-5}
Sécante	$1,3652$	10	$0,502$ ms	10^{-7}
Fausse position	$1,3652$	33	$0,541$ ms	10^{-5}

Table 1 – Résultats pour la fonction $f(x) = x^3 + 4x^2 - 10$

Ce tableau nous permet d'observer que la méthode nécessitant le moins d'itérations pour converger est la méthode de Newton. Toutefois, elle n'offre pas la meilleure précision puisque la méthode de la sécante approche la fonction à 10^{-7} près.
La comparaison entre les méthodes de dichotomie et trichotomie (basées sur le même fonctionnement) montre que la trichotomie est plus rapide à converger et plus précise. En revanche, elle est plus lente au niveau du temps d'exécution.
La méthode qui semble la moins efficace est la méthode de la fausse position avec ses 33 itérations.
On représente l'ensemble des erreurs que nous avons obtenu pour chaque technique sur une échelle logarithmique sur cette figure :

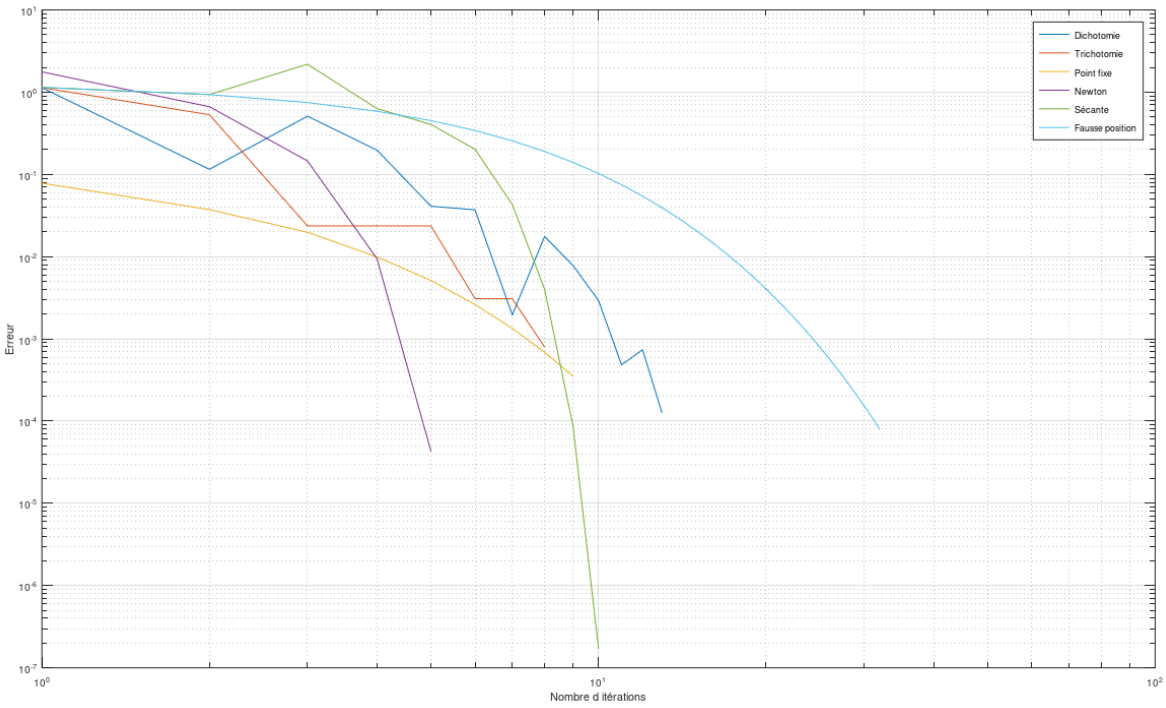

FIGURE 1 – Erreur pour chaque méthode en fonction de l'itération

Cette figure représente bien les résultats énoncés dans le tableau.

4 Discussions

Tout d'abord, toutes les méthodes de calcul de zéro que nous avons codées en MATLAB convergent bien vers le zéro de la fonction f considérée. De plus, les temps nécessaires pour s'approcher très près ce zéro sont infimes, de l'ordre du dixième de millisecondes pour toutes les fonctions. Ainsi, elles sont toutes rapides et fonctionnelles pour approcher la racine d'une fonction. Cependant, toutes n'ont pas la même efficacité, et c'est ce dont nous allons discuter.

4.1 Comparaison des méthodes

Pour rappel, les données rapportées dans la partie Résultats se font pour une fonction précise et dans des conditions particulières. Afin de comparer en profondeur les algorithmes, nous allons faire varier ces conditions pour observer leurs comportements.
En analysant les données du tableau 1, on ne peut être sur de savoir quelle est la "meilleure" méthode, c'est-à-dire celle approchant le zéro le plus précisément possible et le plus rapidement possible. Nous allons donc aborder le problème avec l'objectif d'approcher le zéro le plus précisément possible.
Pour déterminer celle qui approche le plus vite le zéro, nous fixerons une tolérance très basse et observerons le nombre d'itérations nécessaires à atteindre le zéro exact. On regroupe ces données dans ce tableau :

Méthode	Nombre d'itérations
Dichotomie	54
Trichotomie	35
Point fixe	53
Newton	7
Sécante	13
Fausse position	+100

Table 2 – Nombre d'itérations pour atteindre le zéro exact

On voit que la méthode de Newton est la plus efficace pour ce genre de problème, puisqu'elle atteint le zéro exact en seulement 7 itérations. La moins efficace est celle de la fausse position qui dépasse les 100 itérations.

Ainsi, la méthode de Newton semble la plus efficace pour calculer le zéro d'une fonction non linéaire. Toutefois, elle présente des inconvénients. En effet, f doit être régulière car la convergence vers la racine n'est pas assurée. De plus, elle nécessite que f soit dérivable (car la méthode fait intervenir $f'(x)$). De la même façon, les méthodes de la sécante et de la fausse position, qui dépendent elles aussi de leur dérivée, présentent cet inconvénient.

La méthode du point fixe elle est encore plus limitée. En effet, pour trouver la racine d'une fonction f, il est nécessaire de connaître une fonction g associée dont on doit chercher le point fixe. Il serait possible d'implémenter une fonctionnalité déterminant cette fonction g automatiquement, mais cela serait très complexe et nécessiterait des packages non natifs.

La dichotomie et la trichotomie, dont nous discuterons en détail dans la partie suivante offrent une méthode efficace et sûre puisqu'elle ne dépend que du fait que le zéro se trouve dans l'intervalle considéré au début. Même si elles ne sont pas les plus rapides ou celles qui convergent le plus vite, elles sont "solides" dans leur fonctionnement.

4.2 Autour de la trichotomie et de la dichotomie

La dichotomie et la trichotomie suivent un fonctionnement similaire. Elle dépendent toutes les deux d'un intervalle initial $[a, b]$ qui sera divisé successivement. La différence se situe dans la division. La dichotomie fait intervenir une division par 2 et la trichotomie par 3. Nous pouvons donc nous interroger sur laquelle des deux est la plus rapide à converger.

Les valeurs citées dans le Tableau 1 semblent montrer que la trichotomie est plus efficace que la dichotomie. Au niveau du temps d'exécution toutefois, en moyenne la dichotomie est plus rapide que la trichotomie. Cela est du au fait que la trichotomie fait intervenir plus de calculs que la dichotomie en redéfinissant selon 4 bornes au lieu de 3.
Ainsi, la vitesse de convergence de la trichotomie "itérations par secondes" est inférieure à celle de la dichotomie. Dans le cas considéré, elle est de $\frac{8}{385.10^{-4}} = 207$ itérations par seconde pour la trichotomie, et $\frac{13}{280.10^{-4}} = 464$ itérations par seconde pour la dichotomie. La dichotomie convergera beaucoup plus de deux fois plus vite que la dichotomie. Ainsi,

cela peut poser problème sur des très grand intervalles nécessitant de nombreuses itérations, puisque la trichotomie prendre un temps sensiblement plus long que la dichotomie.

5 Conclusion

En conclusion, nous avons exploré différentes méthodes algorithmiques de recherche de zéro, puis comparé leur rapidité et leur précision. Nous avons montré que, même si elles étaient toutes fonctionnelles, la méthode de Newton était la plus efficace dans le cas de la recherche d'une fonction dérivable et stable. La méthode de la sécante offre également une convergence rapide, mais dépend aussi de sa dérivabilité. La méthode du point fixe est aussi relativement efficace, mais dépend d'une fonction annexe fastidieuse à déterminer. La dichotomie et la trichotomie offrent des solutions stables et efficaces même si elles ne sont pas les plus rapides. Enfin, la méthode de la fausse position a montrée des résultats similaires mais de façon plus lente et moins stable.

Ainsi, les méthodes étudiées permettent d'obtenir une approximation du zéro d'une fonction non-linéaire dont il serait difficile de calculer les racines de façon analytique. Elles offrent des solutions différentes mais complémentaires à ce type de problème.

6 Références

— **Source 1 :** Méthode de Newton. (2023, avril 13). Dans *Wikipédia*. Récupéré le 18 mai 2023, de https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Newton
— **Source 2 :** Point fixe. (2023, avril 13). Dans *Wikipédia*. Récupéré le 18 mai 2023, de https://fr.wikipedia.org/wiki/Point_fixe
— **Source 3 :** Albera, L. (2023). Module de mathématiques CUPGE1 ESIR.
— **Source 4 :** Lemonnier, F. (2015, 7 juin). *Méthode de Newton* [PDF]. Récupéré de https://perso.eleves.ens-rennes.fr/~lgay/Agregation/Me%CC%81thode%20de%20Newton.pdf
— **Source 5 :** Bodin, A., Borne, N. Desideri, L. (2015). *Zéros des fonctions*. Récupéré de http://exo7.emath.fr/cours/ch_zeros.pdf
— **Source 6 :** Méthode de la fausse position. (2023, janvier 24). Dans *Wikipédia*. Récupéré le 18 mai 2023, de https://fr.wikipedia.org/wiki/M%C3%A9thode_de_la_fausse_position