

VALEUR APPROCHÉE DU ZÉRO D'UNE FONCTION

MÉTHODE DE NEWTON

Introduction

Dans ce chapitre nous allons voir comment évaluer *numériquement* une solution d'équation de la forme $f(x) = 0$.

Nous avons déjà vu la méthode de dichotomie qui s'appliquait à une fonction continue.

La méthode de Newton, plus délicate d'application, fournit cependant une vitesse de convergence bien plus élevée.

1. Explication de la méthode

1.1 Cadre

On se donne une fonction f définie sur un intervalle $[a, b]$ qui est supposée s'annuler en une abscisse α .

On veut construire une suite $(u_n)_{n \in \mathbb{N}}$ convergente vers α .

1.2 Protocole

Sous réserve de la validité de chaque étape on souhaite procéder comme suit.

Partons de $u_0 = b$.

Supposant u_n défini on fait la construction géométrique suivante :

1. On place le point M_n de coordonnées $(u_n, f(u_n))$.
2. On trace la tangente T_n à $\mathcal{C}_f$ au point M_n .
3. On place le point d'intersection de T_n avec l'axe des abscisses.

On définit alors u_{n+1} comme l'abscisse de ce dernier point.

Exercice 1 Déterminer l'expression de u_{n+1} en fonction de u_n .

$$\forall n \in \mathbb{N}, \quad u_{n+1} =$$

1.3 Questions problématiques

Exercice 2 Quels problèmes peuvent apparaître lors de la mise en place de la méthode ?

Condition suffisante de convergence de la méthode. Mathématiquement, on peut prouver que si f et f' sont croissantes alors la suite est bien définie, est décroissante et minorée par α : elle converge donc et par passage à la limite dans la relation de récurrence, il s'avère que cette limite est α .

Représentation graphique.

Remarque. Plus généralement, dès que f et f' sont monotones, on peut adapter la méthode en changeant éventuellement l'initialisation (en prenant $u_0 = a$ au lieu de $u_0 = b$).

2. Programmation

2.1 Un cas particulier

La suite u ainsi construite permet d'obtenir des valeurs approchées de α .

Pour programmer la recherche d'une valeur approchée à ε près, on peut effectuer une boucle "Tant que".

Il faut cependant préciser une condition d'arrêt de la boucle **while**.

Exercice 3

1. Dans le cas f et f' croissantes, justifier qu'on peut utiliser le test

$$\text{"tant que } f(u_n) * f(u_n - \varepsilon) > 0\text{"}$$

2. Comment transformer cette condition pour qu'elle fonctionne dans les quatre cas de figure ?

2.2 Cas général

Dans la pratique on souhaite s'affranchir des hypothèses sur f , hypothèses qui ne sont pas facilement vérifiables sur une fonction arbitraire...

Il est d'usage, pour cette méthode, d'obtenir la condition

$$u_{n+1} - u_n \leq \varepsilon$$

AVERTISSEMENT. Il s'agit d'un test empirique. En toute généralité, il n'est absolument pas fiable.

Pour plus de précautions on pourra une fois obtenue notre valeur approchée, utiliser le test vu auparavant

$$f(u_n - \varepsilon) \times f(u_n + \varepsilon) \leq 0$$

2.3 Programme Python

```
def newton(f, fprime, u0, epsilon = 1e-6, it_max=15):
    """ zéro approché de f à epsilon près

    u0 valeur initiale
    """

    u = u0 # u = u_0
    v = u - f(u)/fprime(u) # v = u_1
    k=1 # compte-tour
    while abs(v - u) > epsilon and k < it_max:
        # k-ème tour de boucle
        u = v # u = u_k
        v = v - f(v)/fprime(v) # v = u_{k+1}
        k=k+1
    return v
```

3. Rapidité de convergence

3.1 Étude d'un exemple

On considère ici $f : x \mapsto x^2 - 2$ avec comme valeur initiale $u_0 = 2$.

La méthode de Newton s'applique ici et donne, après simplification, la suite u définie par

$$\begin{cases} u_0 = 2 \\ u_{n+1} = \frac{1}{2} \left(u_n + \frac{2}{u_n} \right) \end{cases}$$

On peut prouver que la suite u converge vers $\ell = \sqrt{2}$ et on s'intéresse maintenant à la rapidité de convergence.

Exercice 4 Soit $\delta_n = |u_n - \sqrt{2}|$ l'erreur d'approximation.

1. Montrer que $\forall n \in \mathbb{N}, \delta_{n+1} = \frac{\delta_n^2}{2u_n} \leq \delta_n^2$.
2. En déduire une majoration de δ_n en fonction de δ_0 et de n .
3. Soit $\varepsilon > 0$. Donner une valeur suffisante de n pour avoir $\delta_n \leq \varepsilon$.
4. Comparer avec la méthode de dichotomie.

3.2 Cas général

On peut montrer le théorème suivant

Théorème. Soit f de classe $\mathcal{C}^2$ sur un intervalle I .

On suppose que f s'annule en α et que $f'(\alpha) \neq 0$.

Si u_0 est assez proche de α alors la méthode de Newton converge et on a la vitesse de convergence suivante

$$|u_n - \alpha| = O\left(\frac{1}{2^{2^n}}\right)$$

Morale. Ainsi, sous de bonnes hypothèses, la méthode de Newton converge très rapidement.

Plus précisément, le nombre de bits significatifs (les “décimales en base 2”) va doubler à chaque étape.

4. Pour aller plus loin

4.1 Évaluation de la dérivée

La méthode de Newton telle que nous l'avons programmée nécessite de connaître la dérivée de la fonction.

Cependant, c'est parfois loin d'être évident en pratique. Par exemple on peut ne connaître une fonction que par un échantillonnage de points sans formule explicite.

Des méthodes assez élémentaires permettent de déterminer des approximations de la dérivée à partir des valeurs de f . Nous en verrons certaines en TP.

4.2 Dichotomie ou méthode de Newton : un choix cornélien

Comme dans beaucoup de problèmes numériques, tout dépend de l'objectif fixé.

Fiabilité à coup sûr. La méthode de dichotomie est celle qui offre le plus de robustesse.

Les conditions d'application sont simples et, même si elle est moins rapide que la méthode de Newton, c'est tout de même souvent suffisant.

Rapidité à tout prix. Si on connaît une bonne approximation préalable d'un zéro et si on cherche à obtenir très rapidement une très bonne précision, la méthode de Newton s'appliquera.

Suivant la connaissance ou non de la dérivée, une variante pourra être utilisée (voir paragraphe précédent et en TP).

Compromis entre les deux. Dans des situations plus délicates, une alternance entre les deux méthodes pourra être envisagée.

Dichotomie pour avoir une première approximation, tentative de méthode de Newton, si cela n'a pas l'air de converger retour à la dichotomie, etc.

4.3 Utiliser les fonctions pré-implantées

Nous avons ici une ébauche de programmation de la méthode de Newton.

Il faut cependant bien réaliser que cette ébauche n'est pas du tout optimisée.

On peut trouver dans certains modules de Python (comme `numpy` ou `scipy`) des fonctions de recherche de racines très efficaces :

- Recherche de zéro par la méthode de dichotomie : fonction `bisect` de `scipy.optimize`
- Recherche de zéro par la méthode de Newton : fonction `newton` de `scipy.optimize` (avec ou sans calcul de la dérivée).
- Recherche de zéro d'une équation : fonction `fsolve` de `scipy.optimize`
- Racines d'un polynôme : fonction `roots` de `numpy`