**Newton – Dichotomie**

Informatique

***Cours***

[Newton - Dichotomie 3](#_Toc157078821)

[1.I. Dichotomie 3](#_Toc157078822)

[1.I.1 Contexte 3](#_Toc157078823)

[1.I.2 Objectif 3](#_Toc157078824)

[1.I.3 Principe 4](#_Toc157078825)

[1.I.4 Exemple 4](#_Toc157078826)

[1.I.5 Remarque 5](#_Toc157078827)

[1.I.6 Analyse de l’algorithme 5](#_Toc157078828)

[1.I.6.a Correction 5](#_Toc157078829)

[1.I.6.b Terminaison et complexité 5](#_Toc157078830)

[1.I.7 Exemple de programmation 6](#_Toc157078831)

[1.II. Newton 7](#_Toc157078832)

[1.II.1 Contexte 7](#_Toc157078833)

[1.II.2 Objectif 7](#_Toc157078834)

[1.II.3 Principe 8](#_Toc157078835)

[1.II.4 Exemple 8](#_Toc157078836)

[1.II.5 Remarques 9](#_Toc157078837)

[1.II.6 Analyse de l’algorithme 9](#_Toc157078838)

[1.II.7 Limites 10](#_Toc157078839)

[1.II.7.a Division par zéro 10](#_Toc157078840)

[1.II.7.b Non convergence 10](#_Toc157078841)

[1.II.7.c Convergence vers une mauvaise solution 11](#_Toc157078842)

[1.II.8 Exemple de programmation 11](#_Toc157078843)

[1.III. Comparaison des méthodes 12](#_Toc157078844)

[1.IV. Remarque 13](#_Toc157078845)

[1.V. Fonction natives Python 13](#_Toc157078846)

[1.V.1 Dichotomie – Fonction « bisect » 13](#_Toc157078847)

[1.V.2 Newton – Fonction « newton » 14](#_Toc157078848)

###### Newton - Dichotomie

# Dichotomie

Le principe de la dichotomie est de diviser le domaine de recherche par 2 à chaque itération.

## Contexte

Soit une fonction définie et continue sur un intervalle de départ telle qu’il existe tel que :

On se placera ici dans le cas où la fonction est monotone sur l’intervalle d’étude .

Autrement dit, la fonction possède un signe sur et le signe opposé sur .

Exemple :

## Objectif

On souhaite déterminer une solution approchée de l’équation :

Appelons la solution exacte de cette équation et la solution approchée obtenue par dichotomie. Nous allons par cette méthode déterminer un intervalle tel que :

On saura alors que l’écart entre la solution exacte et la solution approché est inférieur à (et même à ) :

Où est un réel définissant la précision de la solution obtenue.

Remarque : écrire ou n’a que peu d’importance.

## Principe

Le principe de cette recherche par dichotomie est le suivant :

* Vérifier éventuellement l’existence de la solution sur . Attention, la solution existe sur l’intervalle si et seulement si – Le « ou égale » est important
* Initialiser
* Déterminer l’abscisse au centre de l’intervalle et calculer l’image
* Identifier dans quel intervalle ou se trouve la solution de par étude du signe de ou de
* Définir le nouvel intervalle de recherche comme celui dans lequel la solution existe
* Continuer tant que l’intervalle de recherche est de largeur supérieure à un dépendant du critère imposé

A la fin, il faut renvoyer une valeur de dans l’intervalle obtenu lorsque le critère est vérifié.

* Renvoyer ou permet d’obtenir une précision inférieure à
* Renvoyer la valeur au centre permet d’obtenir une précision inférieure à

Il faut veiller à mettre le bon test , , ou face au bon sous intervalle atteint :

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

On remarque qu’il faut que le sous intervalle atteint contienne les abscisses du test nul.

## Exemple

Soit un critère tel que la longueur de l’intervalle final soit inférieure à la longueur de la flèche ci-dessous :

On procède par itérations successives à la division par 2 de l’intervalle de recherche :

## Remarque

Il est envisageable de proposer un critère d’arrêt sur les ordonnées plutôt que sur les abscisses, voire les deux en même temps. Cela dépend de la forme de la fonction et de la précision attendue, sur ou sur . Voici l’illustration pour deux fonctions en prenant les mêmes critères :

## Analyse de l’algorithme

### Correction

Vous montrerez dans le cours de mathématiques que :

Si , l'algorithme permet bien d'obtenir une valeur approchée de la solution de l'équation sur l'intervalle

Ceci prouvera la correction de l'algorithme.

### Terminaison et complexité

La longueur de l'intervalle étant divisée par 2 à chaque itération de la boucle while, on peut remarquer aisément que, au bout de itérations de la boucle :

Pour démontrer la terminaison et déterminer la complexité de l'algorithme on résout donc l'inéquation d’inconnue suivante où est le critère d’arrêt :

On a trouvé un variant qui est le numéro de l’itération , entier qui croit strictement dans l’intervalle . Ainsi, l'algorithme se termine toujours et la complexité est en soit une complexité logarithmique.

Soit le nombre d’intervalles de longueur entre et . Alors la complexité de l’algorithme de dichotomie continue est en .

Plus on recherche la solution sur un intervalle grand, et plus le critère est petit, plus le temps d’exécution sera long.

## Exemple de programmation

|  |
| --- |
| **def** f**(**x**):**  **return** x**\*\***2**-**1 |
| **def** dicho**(**f**,**a**,**b**,**eps**):**  **assert** f**(**a**)\***f**(**b**)** **<=** 0**,**"Pas de solution dans [a,b]"  **while** **abs(**b**-**a**)** **>** 2**\***eps**:**  m **=** **(**a**+**b**)/**2  **if** f**(**a**)\***f**(**m**)** **<=** 0**:**  b **=** m  **else:**  a **=** m  **return** **(**a**+**b**)/**2  **def** dicho**(**f**,**a**,**b**,**eps**):**  **assert** f**(**a**)\***f**(**b**)** **<=** 0**,**"Pas de solution dans [a,b]"  **while** **abs(**b**-**a**)** **>** eps**:**  m **=** **(**a**+**b**)/**2  **if** f**(**a**)\***f**(**m**)** **<=** 0**:**  b **=** m  **else:**  a **=** m  **return** m  sol **=** dicho**(**f**,**0**,**1**,**0.00001**)** |
| **def** dicho\_rec**(**f**,**a**,**b**,**eps**):**  **assert** f**(**a**)\***f**(**b**)** **<=** 0**,**"Pas de solution dans [a,b]"  m **=** **(**a**+**b**)/**2  **if** **abs(**b**-**a**)** **<** 2**\***eps**:**  **return** m  **else:**  **if** f**(**a**)\***f**(**m**)<=** 0**:**  b **=** m  **else:**  a **=** m  **return** dicho\_rec**(**f**,**a**,**b**,**eps**)**  Sol **=** dicho\_rec**(**f**,**0**,**1**,**0.00001**)** |
| **print(**Sol**)** |
| Sans assertion, une solution est renvoyée même si elle n’existe pas ! |

# Newton

La méthode de Newton est une seconde méthode de résolution d’équation de la forme qui converge plus vite que la méthode de dichotomie vers la solution recherchée.

## Contexte

Soit une fonction définie, continue et dérivable sur un intervalle telle qu’il existe tel que :

On se placera ici dans le cas où la fonction est monotone sur l’intervalle d’étude .

Autrement dit, la fonction possède un signe sur et le signe opposé sur .

Exemple :

## Objectif

On souhaite déterminer une solution approchée de l’équation :

Contrairement à la résolution par Dichotomie vue au paragraphe précédent, nous ne sommes pas en mesure d’être sûrs que la solution approchée par la méthode de Newton est à une distance précise de la solution réelle.

Nous allons suivre l’écart entre deux solutions successives et proposer un critère d’arrêt lorsque deux solutions successives sont « assez proches ».

## Principe

Le principe de la méthode de Newton consiste à approcher la courbe de avec sa tangente en un point initial choisi arbitrairement. On déterminer alors l’abscisse du point d’intersection de cette tangente avec l’axe des abscisses, soit .

On procède alors ainsi par itérations :

Soit :

On procède alors ainsi jusqu’à ce que :

On renvoie alors avec une précision

## Exemple

Soit un critère tel que la longueur de l’intervalle final soit inférieure à la longueur de la flèche ci-dessous :

On procède par itérations successives :

## Remarques

Comme pour la dichotomie, il est envisageable de proposer un critère d’arrêt sur les ordonnées plutôt que sur les abscisses, voire les deux en même temps. Cela dépend de la forme de la fonction et de la précision attendue, sur x ou sur . Voici l’illustration pour deux fonctions en prenant les mêmes critères :

* Pour pouvoir appliquer la méthode de Newton, il faut a priori connaitre la fonction et sa dérivée . Dans le cas où on ne connait que la fonction *f* et que l'on ne peut pas déterminer sa dérivée, il est possible d'obtenir une approximation de à partir des valeurs de en utilisant l’une des formules suivantes :
* En choisissant d'approximer par dans la formule de calcul de de la méthode de Newton, on obtient une méthode dérivée de la méthode de Newton appelée **méthode de la sécante** (utile quand on ne connaît pas la dérivée de f):

## Analyse de l’algorithme

On peut montrer mathématiquement qu'une condition suffisante pour assurer la convergence de la méthode de Newton est que soit de classe sur avec sur et . **Ceci sera démontré en maths** lorsque vous aurez les outils nécessaires.

La complexité sera démontrée en maths. Quand la méthode converge, la convergence est quadratique. **La précision double à chaque itération**. Autrement dit, est dominé par une suite .

## Limites

### Division par zéro

À tout moment, on calcule :

Il peut donc y avoir un problème si . En effet, la tangente étant horizontale, il n’est plus possible de trouver une nouvelle valeur de x…

D’une manière générale, si , risque de devenir très grand. On pourra donc ajouter un test du type : si alors on stoppe le programme.

### Non convergence

Il peut arriver que la méthode de Newton ne converge pas. En voici un exemple :

En effet, la fonction tracée ne respecte pas le critère : sur .

Remarque : on pourra compter les itérations réalisées et stopper le programme après N itérations…

### Convergence vers une mauvaise solution

Imaginons une fonction dont la pente est très verticale sur un intervalle (fonction presque non dérivable…) :

Du fait de la présence d’un critère d’arrêt sur deux valeurs successives de la solution, on voit clairement que selon la pente de la fonction et la largeur du critère, il y aura arrêt sur une solution éloignée de la vraie solution.

## Exemple de programmation

|  |
| --- |
| **def** newton**(**f**,**fp**,**x0**,**eps**):**  xi **=** x0  xip1 **=** x0 **+** 2**\***eps  **while** **abs(**xip1**-**xi**)** **>=** eps**:**  xi **=** xip1  xip1 **=** xi **-** f**(**xi**)/**fp**(**f**,**xi**)**  **return** xip1  **def** f**(**x**):**  **return** x**\*\***2**-**1  **def** fp1**(**f**,**x**):** # arg f nécessaire pour newton  Val **=** 2 **\*** x  **return** Val  **def** fp2**(**f**,**x**):** # Approximation  dx **=** 1e-5  Val **=** **(**f**(**x**+**dx**)-**f**(**x**))/**dx  **return** Val  sol **=** newton**(**f**,**fp1**,**0**,**0.00001**)**  **print(**sol**)** |
| Si la solution n’existe pas, le code risque de ne jamais s’arrêter |

# Comparaison des méthodes

Lorsque la méthode de Newton converge, elle converge vite comparé à la méthode de Dichotomie. Toutefois, elle est dépendante de :

* L’allure de la fonction (monotonie en particulier)
* La dérivée (tangente horizontale et verticale problématiques)

La méthode de Newton peut effectuer de grandes variations d’une itération à l’autre et induire des temps de convergence longs.

La méthode de dichotomie, bien que plus lente, assure cependant une progression constante de l’intervalle de précision. Elle est très bien adaptée aux fonctions non monotones par exemple.

On pourra envisager des stratégies du type :

* Résolution de Newton sur 100 itérations
* Si la convergence n’est pas atteinte, résolution par Dichotomie

# Remarque

Quelle que soit la méthode, garder en tête que nous manipulons des nombres codés en virgule flottante, et ne pas chercher des critères trop petits.

# Fonction natives Python

## Dichotomie – Fonction « bisect »

A la condition que le produit , il existe une fonction qui trouve le 0 d’une fonction sur un intervalle :

|  |  |
| --- | --- |
| Code | Exécution |
| **from** scipy**.**optimize **import** bisect  **def** f**(**x**):**  **return** x**\*\***2**-**1  a **=** 0  b **=** 1  Res **=** bisect**(**f**,**a**,**b**)**  **print(**Res**)** |  |

## Newton – Fonction « newton »

A la condition que le produit , il existe une fonction qui trouve le 0 d’une fonction sur un intervalle :

|  |  |
| --- | --- |
| Code | Exécution |
| **from** scipy**.**optimize **import** newton  **def** f**(**x**):**  **return** x**\*\***2**-**1  **def** fp**(**x**):**  **return** 2**\***x  x0 **=** 2  Res **=** newton**(**f**,**x0**,**fp**)**  **print(**Res**)** |  |
| **from** scipy**.**optimize **import** newton  **def** f**(**x**):**  **return** x**\*\***2**-**1  x0 **=** 2  Res **=** newton**(**f**,**x0**,**fprime**=None)**  **print(**Res**)** |  |
| **from** scipy**.**optimize **import** newton  **def** f**(**x**):**  **return** x**\*\***2**-**1  **def** fp**(**x**):**  **return** 2**\***x  x0 **=** 2  Res **=** newton**(**f**,**x0**,**tol**=**0.1**)**  **print(**Res**)** |  |

On peut préciser ne pas préciser « fprime**=None**» en 3° argument , ce qui revient au même. Qu’on le précise ou non, on applique finalement la méthode de la sécante. Mais si on le souhaite, on peut préciser en 3° argument la fonction dérivée si elle est connue, pour plus de précision.

Lorsque l’on ne précise pas de critère dans la fonction de Newton, celle-ci estime d’elle-même évoluant à chaque itération, proche du carré de la précision à l’étape précédente. Voici l’extrait de la documentation associée :

Une image contenant texte

Description générée automatiquement