

Projet maths-info  
Résolution d'équations non-linéaires

Benamara Abdelkader  
Benichou Yaniv

08 Juin 2020

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Méthode De Dichotomie</b>	<b>4</b>
2.1	Théoreme des valeurs intermédiaires : . . . . .	4
2.2	Théoreme de Bolzano . . . . .	4
2.3	L'algorithme de dichotomie . . . . .	4
2.4	Interpretation Géometrique . . . . .	6
2.5	Convergence . . . . .	6
<b>3</b>	<b>Méthode de Newton</b>	<b>7</b>
3.1	Présentation . . . . .	7
3.2	Interpretation Géometrique . . . . .	8
3.3	Implementation du code . . . . .	9
3.4	Convergence . . . . .	10
<b>4</b>	<b>Méthode des Cordes</b>	<b>11</b>
4.1	Présentation . . . . .	11
4.2	Interpretation Géometrique . . . . .	11
4.3	Implementation du code . . . . .	12
4.4	Convergence . . . . .	13
<b>5</b>	<b>Méthode de la Fausse-Position</b>	<b>14</b>
5.1	Présentation . . . . .	14
5.2	Interpretation Géometrique . . . . .	14
5.3	Implementation du code . . . . .	15
5.4	Convergence . . . . .	16
<b>6</b>	<b>Interface Graphique</b>	<b>17</b>
6.1	Outils utilisés . . . . .	17
6.2	Accueil . . . . .	17
6.3	Dichotomie . . . . .	18
6.3.1	Convergence . . . . .	19
6.4	Newton . . . . .	20
6.5	Cordes . . . . .	21
6.6	Fausse Position . . . . .	22
6.7	Gestion des erreurs . . . . .	23
6.7.1	Formule Erronée . . . . .	23
6.7.2	Monotonie . . . . .	23
6.7.3	Interval Erroné . . . . .	24
6.7.4	Max d'itération atteint sans trouver de solution . . . . .	24
<b>7</b>	<b>Conclusion</b>	<b>25</b>

# 1 Introduction

Le but de ce projet est de décrire des algorithmes les plus fréquemment utilisés pour résoudre des équations non linéaires du type :

$f(x) = 0$ . Ainsi, l'objet essentiel de ce chapitre est l'approximation des racines d'une fonction réelle d'une variable réelle, c'est-à-dire la résolution approchée du problème suivant : étant donné  $f : I = ]a, b[ \subset \mathbb{R} \rightarrow \mathbb{R}$ , trouver  $\alpha \in \mathbb{R}$  tel que  $f(\alpha) = 0$ .

Les méthodes que nous verrons ici sont itératives, c'est à dire qu'elles consistent à construire une suite  $x_k$  telle que  $\lim_{k \rightarrow +\infty} x_k = \alpha$ . Ainsi, au delà d'approximer une solution, nous nous intéresseront également à la vitesse de convergence, s'il y a convergence ou non, d'une méthode pour une fonction donnée. Pour cela, nous définissons la convergence des itérations par la notion suivante : dit qu'une suite  $x^k$  construite par une méthode numérique converge vers  $\alpha$  avec un ordre  $p \geq 1$  si  $\exists C > 0 : \frac{|x^{k+1} - \alpha|}{|x^k - \alpha|^p} \leq C, \quad \forall k \geq k_0$ .

où  $k_0 \geq 0$  est un entier. Dans ce cas, on dit que la méthode est d'ordre  $p$ . Remarquer que si  $p$  est égal à 1, il est nécessaire que  $C$  soit inférieur à 1 pour que  $x_k$  converge vers  $\alpha$ . On appelle alors la constante  $C$  facteur de convergence de la méthode.

## 2 Méthode De Dichotomie

### 2.1 Théoreme des valeurs intermédiaires :

Pour toute application continue  $f : [a, b] \rightarrow \mathbb{R}$  et tout reel  $u$  compris entre  $f(a)$  et  $f(b)$ , il existe au moins un reel  $c$  compris entre  $a$  et  $b$  tel que  $f(c) = u$ .  
En particulier on a la version de Bolzano qui nous interesse précisément.

### 2.2 Théoreme de Bolzano

Soit une fonction continue  $f : [a, b] \rightarrow \mathbb{R}$   
si  $f(a)f(b) < 0$  alors  $\exists \alpha \in ]a, b[$  tel que  $f(\alpha) = 0$ .

On considère deux nombres réels  $a$  et  $b$  et une fonction réelle  $f$  continue sur l'intervalle  $[a, b]$  telle que  $f(a)$  et  $f(b)$  soient de signes opposés. Supposons que nous voulions résoudre l'équation  $f(x) = 0$ . D'après le théorème des valeurs intermédiaires,  $f$  a au moins un zéro dans l'intervalle  $[a, b]$ . La méthode de dichotomie consiste à diviser l'intervalle en deux en calculant  $m = (a+b) / 2$ . Il y a maintenant deux possibilités : ou  $f(a)$  et  $f(m)$  sont de signes contraires, ou  $f(m)$  et  $f(b)$  sont de signes contraires.

L'algorithme de dichotomie est alors appliqué au sous-intervalle dans lequel le changement de signe se produit, ce qui signifie que l'algorithme de dichotomie est récursif.

### 2.3 L'algorithme de dichotomie

---

**Algorithm 1:** Méthode de dichotomie

---

**Result:**  $millieu$  tq  $f(millieu) = 0$

initialization;

**while**  $fin - debut \geq err$  **do**

$millieu = (debut + fin) / 2$ ;

**if**  $f(millieu) \neq 0$  **then**

$fin = millieu$ ;

**else**

$debut = millieu$ ;

**end**

**end**

---

```

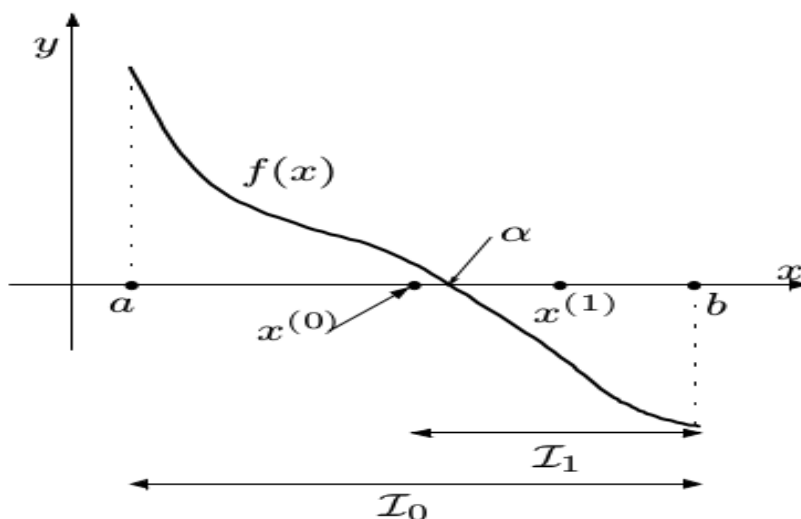
1 class Dichotomie(Equa_Solver):
2
3     def solve(self):
4         # Donnees en parametres
5         a , b = self.a , self.b
6         err=self.err
7         try:
8             f = lambda x: eval(str(self.f))
9         except (TypeError, SyntaxError):
10             return
11         x_list=[]
12
13         print(f" Fonction      : {self.f}")
14         print(f" Intervalle   : [{a},{b}]")
15         print(f" Erreur       : {err} \n")
16
17
18         if( f(a)*f(b) > 0):
19             raise SolverException(" f(a) et f(b) doivent etre de
20             signe different !")
21
22         else:
23             debut = a
24             fin = b
25             n=1
26
27             while (fin - debut > err):
28                 millieu = (debut + fin) / 2
29                 x_list.append(millieu)
30                 print(f"Found solution after {n} iterations : {
31                 millieu} ")
32                 n+=1
33                 if (f(debut) * f(millieu) < 0):
34                     fin = millieu
35                 else:
36                     debut = millieu
37
38             print(f" Solution approchee de f(x) = 0 est : {millieu
39             }\n")
40             return x_list

```

Listing 1: Méthode de dichotomie en Python

## 2.4 Interpretation Géométrique

L'algorithme de dichotomie est un algorithme classique en algorithmique. Sur un intervalle  $[a, b]$ , l'intervalle va à chaque itération se réduire de moitié, puisque soit  $a$  soit  $b$  prendra la valeur de leur moyenne, soit  $(a+b)/2$ . à chaque itération, jusqu'à obtention de l'approximation souhaitée.



Les deux premiers pas de la méthode de dichotomie.

## 2.5 Convergence

L'algorithme de dichotomie produit deux suites de valeurs : une suite croissante (qui donne une estimation par le bas de la solution) et une suite décroissante (qui donne une estimation par le haut de la solution). La quantité d'intérêt ici est la longueur de l'intervalle d'incertitude, qui décroît toujours de la même façon. Ceci rend l'algorithme de dichotomie un bon algorithme "informatif" puisqu'il permet, à n'importe quelle itération de l'algorithme, de nous assurer la présence de la solution entre un intervalle [ début, fin ]. On a donc une incertitude explicite et non asymptotique, contrairement à d'autres méthodes que nous verrons comme Newton ou Cordes. Ainsi, pour mettre en évidence cette convergence, nous avons tracer un trait rouge (estimation par le bas) et un bleu (estimation par le haut) qui permettront à tout pas de l'algorithme de délimiter cet intervalle, où la solution  $y$  est.

De plus, on peut également affirmer que l'algorithme de dichotomie converge sans surprise, mais à une vitesse lente.

## 3 Méthode de Newton

### 3.1 Présentation

Afin de mettre au point des algorithmes possédant de meilleures propriétés de convergence que la méthode de dichotomie, il est nécessaire de prendre en compte les informations données par les valeurs de  $f$  et, éventuellement, par sa dérivée  $f'$  (si  $f$  est différentiable) ou par une approximation convenable de celle-ci.

L'algorithme de la méthode de Newton peut être présenté brièvement comme suit: à chaque itération, la fonction dont on cherche un zéro est linéarisée en l'itéré (ou point) courant et l'itéré suivant est pris égal au zéro de la fonction linéarisée. Cette description sommaire indique qu'au moins deux conditions sont requises pour la bonne marche de l'algorithme : la fonction doit être différentiable aux points visités (pour pouvoir y linéariser la fonction) et les dérivées ne doivent pas s'y annuler (pour que la fonction linéarisée ait un zéro) ; s'ajoute à ces conditions la contrainte forte de devoir prendre le premier itéré assez proche d'un zéro régulier de la fonction (i.e., en lequel la dérivée de la fonction ne s'annule pas), pour que la convergence du processus soit assurée.

Ecrivons pour cela le développement de Taylor de  $f$  en  $\alpha$  au premier ordre. On obtient alors la version linéarisée du problème  $f(\alpha) = 0 = f(x) + (\alpha - x)f'(\eta)$ , où  $\eta$  est entre  $\alpha$  et  $x$ . L'équation conduit à la méthode itérative suivante :  $\forall k \geq 0$ , étant donné  $x^k$ , déterminer  $x^{k+1}$  en résolvant l'équation :  $f(x^k) + (x^{k+1} - x^k)q_k = 0$ , où  $q_k$  est une approximation de  $f'(x^k)$ .

A présent et pour les prochaines méthodes que nous présenterons, nous effectuerons des choix sur  $q_k$ .

Ici on pose :  $q_k = f'(x_k) \quad \forall k \geq 0$

et en se donnant la valeur initiale  $x^0$ , on obtient la méthode de Newton :

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)} \quad \forall k \geq 0$$

A la  $k$ -ème itération, la méthode de Newton nécessite l'évaluation des deux fonctions  $f$  et  $f'$  au point  $x^k$ . Cet effort de calcul supplémentaire est plus que compensé par une accélération de la convergence, la méthode de Newton étant d'ordre 2.

Or, il est possible que la dérivée de la fonction  $f$  soit relativement pénible à calculer et c'est pour ça que nous avons présenté une deuxième version d'implémentation de cette méthode, sans dérivée donnée en paramètre, puisqu'elle est automatiquement calculée. Cela rend la méthode de Newton très agréable à utiliser, puisqu'elle converge très rapidement et ne nécessite donc, uniquement un point approximatif  $x_0$  comme argument supplémentaire à la fonction.

### 3.2 Interpretation Géométrique

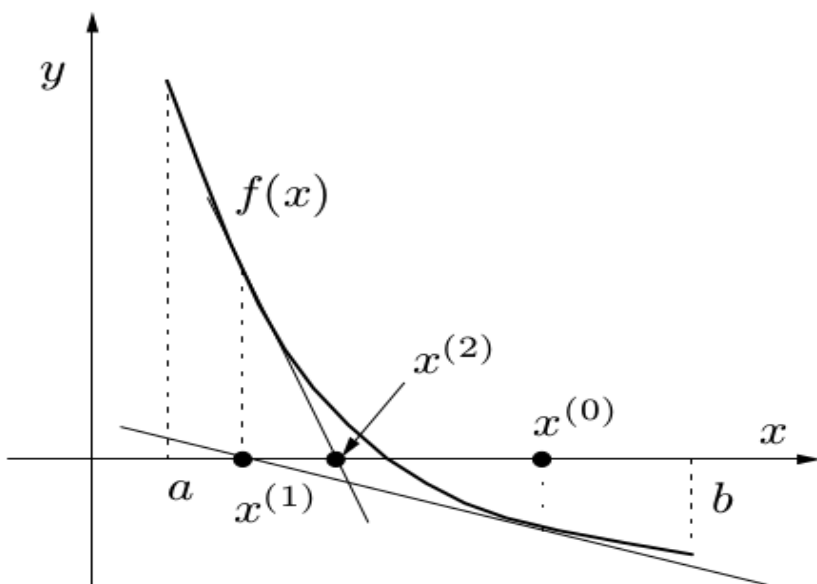
Autrement dit, on veut approcher la fonction au premier ordre, c'est à dire, on la considère asymptotiquement égale à sa tangente en ce point :

$$f(x) \simeq f(x_0) + f'(x_0)(x - x_0).$$

Partant de là, pour trouver un zéro de cette fonction d'approximation, on calcule l'intersection de la droite tangente avec l'axe des abscisses, c'est-à-dire résoudre l'équation affine :

$$0 = f(x_0) + f'(x_0)(x - x_0).$$

On obtient alors un point  $x^1$  qui en général a de bonnes chances d'être plus proche du vrai zéro de  $f$  que le point  $x^0$  précédent. Par cette opération, on peut donc espérer améliorer l'approximation par itérations successives (voir illustration) : on approche à nouveau la fonction par sa tangente en  $x_1$  pour obtenir un nouveau point  $x^2$ , etc.



Les deux premières étapes de la méthode de Newton.



### 3.3 Implementation du code

```
1
2 class Newton(Equa_Solver):
3
4     def solve_with_df(self):
5         f=self.f
6         Df=self.df
7         max_iter=self.max_iter
8         x0=self.x0
9         epsilon=self.err
10        x_list=[]
11
12        fx = lambda x: eval(str(f))
13        dfx = lambda x: eval(str(Df))
14        print("\n\nfunction f : ", f, "\n", "Derivative f' : ", Df,
15              "\n", "-----")
16        xn = x0
17        for n in range(0, max_iter):
18            fxn = fx(xn)
19            x_list.append(xn)
20            self.affiche_info(n,xn,fxn)
21            if abs(fxn) < epsilon:
22                print('Found solution after', n, 'iterations.')
23                print("the approximate solution is : ",x_list[-1])
24                return x_list
25
26            Dfxn = dfx(xn)
27
28            if Dfxn == 0:
29                print('Zero derivative. No solution found.')
30                return None
31            xn = xn - fxn / Dfxn
32
33        print('Exceeded maximum iterations. No solution found.')
34        return None
35
36    def solve_without_df(self):
37        f=self.f
38        max_iter=self.max_iter
39        x0=self.x0
40        epsilon=self.err
41        x_list=[]
42
43        x = Symbol('x')
44        fx = lambda x: eval(str(f))
45        dfx = lambda x: eval(str(diff(f)))
46        print("\n\nfunction f : ", f, "\n", "Derivative f' : ",
47              diff(f), "\n", "-----")
48
49        xn = x0
50
51        for n in range(0, max_iter):
52            fxn = fx(xn)
53            x_list.append(xn)
54            self.affiche_info(n, xn, fxn)
```

```

54         if abs(fxn) < epsilon:
55             print('Found solution after', n, 'iterations.')
56             print("the approximate solution is : ",x_list[-1])
57             return x_list
58
59         Dfxn = dfx(xn)
60         if Dfxn == 0:
61             print('Zero derivative. No solution found.')
62             return None
63         xn = xn - fxn / Dfxn
64     print('Exceeded maximum iterations. No solution found.')
65     return None

```

Listing 2: Méthode de Newton en Python

### 3.4 Convergence

L'algorithme produit une suite de valeurs qui convergent vers la solution de l'exercice. En outre on ne connaît pas la vraie solution, la théorie nous dit simplement qu'on converge vers elle. Pour l'implémentation du calcul de la convergence, il faut donc utiliser la définition de cette dernière définie dans l'introduction.

```

1     def rate(x_list, x_final):
2         e = [abs(x_ - x_final) for x_ in x_list]
3         q = [(log(e[n+1]/e[n]))/(log(e[n]/e[n-1])) for n in range
4             (1, len(e)-1, 1)]
5         return q

```

Listing 3: Calcul de convergence en Python

A ce jour, la méthode de Newton est la méthode qui converge le plus rapidement, s'il y a convergence, parmi celles présentées aujourd'hui. En effet, celle-ci est rapide (souvent quadratique), et de plus elle nécessite uniquement un seul point de départ (généralement grossièrement proche de la solution). Mais, malheureusement  $f$  doit être suffisamment régulière, la convergence n'est pas assurée dans tous les cas, s'il y a plusieurs racines elle ne converge pas forcément vers la plus proche du point de départ. Aussi, la fonction doit être  $C^1$  et il faudrait parfois connaître la dérivée.

## 4 Méthode des Cordes

### 4.1 Présentation

La méthode des cordes est une méthode comparable à celle de Newton, où l'on remplace  $f'(x_n)$ , par  $\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$

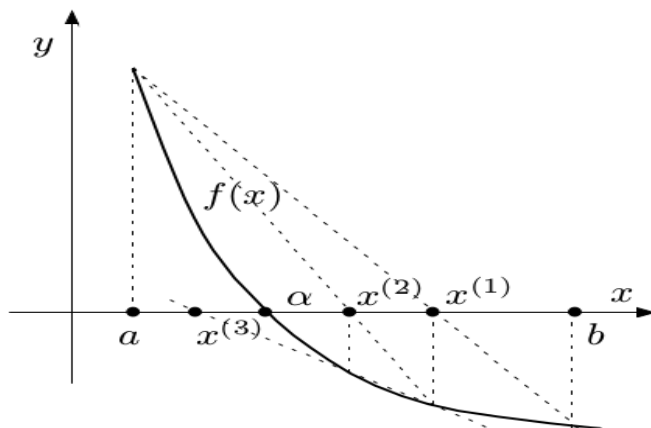
L'initialisation nécessite deux points  $x_0$  et  $x_1$ , proches, si possible, de la solution recherchée. Il n'est pas nécessaire que  $x_0$  et  $x_1$  encadrent une racine de  $f$ . La méthode des cordes peut aussi être vue comme une généralisation de la méthode de la fausse position, où les calculs sont itérés.

Ici on pose donc :  $q_k = \frac{f(x^k) - f(x^{k-1})}{x^k - x^{k-1}} \quad \forall k \geq 0$  d'où on déduit, en se donnant deux valeurs initiales  $x^1$  et  $x^0$ , la relation suivante :

$$x^{k+1} = x^k - \frac{x^k - x^{k-1}}{f(x^k) - f(x^{k-1})} f(x^k)$$

### 4.2 Interprétation Géométrique

De manière très intuitive, cette méthode consiste à tracer une droite entre les points  $f(a)$  et  $f(b)$ , qui passera forcément par l'axe des abscisses en un point,  $x^1$  qui sera la prochaine itération. On réitère le procédé jusqu'à l'approximation de la solution.



Sur ce graphe, on peut voir les deux premières étapes de la méthode pour la résolution d'une fonction  $f$ .

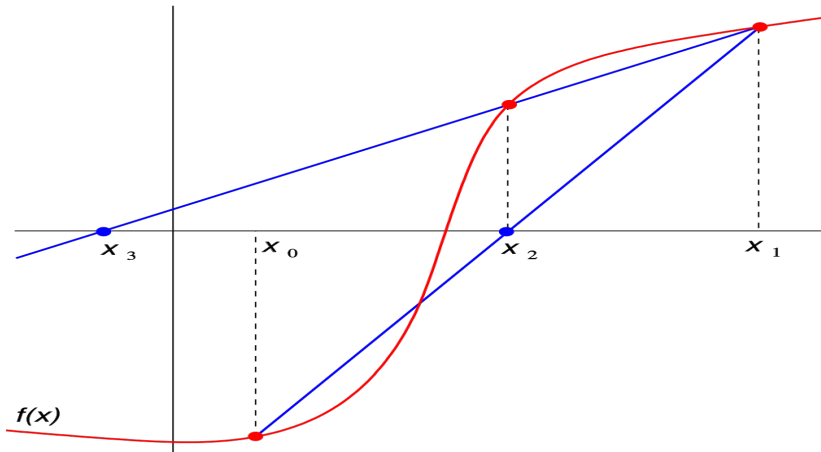


Illustration des deux premières itérations, pour une autre courbe (ici, la méthode va diverger car  $x^0$  et  $x^1$  sont choisis trop loin de la solution).

### 4.3 Implementation du code

```

1  class Cordes(Equa_Solver):
2
3  def solve(self):
4      f=self.f
5      a,b =self.a,self.b
6      max_iter=self.max_iter
7      epsilon=self.err
8      x_list=[]
9
10     fx = lambda x: eval(str(f))
11     print("\n\nfunction f : ", f, " dans l'intervalle [", a, ", ",
12           ", b, "] \n", "-----")
13
14     for n in range(0, max_iter):
15         self.affiche_info(n, b, fx(b))
16         x_list.append(b)
17         if (abs(a - b) < epsilon):
18             print('Found solution after', n, 'iterations.')
19             return x_list
20
21         z = (a * fx(b) - b * fx(a)) / (fx(b) - fx(a))
22         a, b = b, z
23
24     print('Exceeded maximum iterations =', max_iter, '.No
25     solution found.')
26     return None

```

Listing 4: Méthode des cordes en Python

## 4.4 Convergence

L'algorithme produit une suite de valeurs qui convergent vers la solution de l'exercice. En outre on ne connaît pas la vraie solution, la théorie nous dit simplement que l'on converge vers elle. Pour l'implémentation du calcul de la convergence, il faut donc utiliser la définition de cette dernière défini dans l'introduction.

```
1 def rate(x_list, x_final):
2     e = [abs(x_ - x_final) for x_ in x_list]
3     q = [(log(e[n+1]/e[n]))/(log(e[n]/e[n-1])) for n in range
4         (1, len(e)-1, 1)]
5     return q
```

Listing 5: Calcul de convergence en Python

Si les valeurs initiales  $x^0$  et  $x^1$  sont suffisamment proches de la solution, la méthode aura un ordre de convergence de  $\varphi = \frac{1+\sqrt{5}}{2} \simeq 1,618$  qui est le nombre d'or. Ce qui est plus rapide que la méthode de dichotomie par exemple. Sinon, nous avons vu que la méthode peut diverger.

## 5 Méthode de la Fausse-Position

### 5.1 Présentation

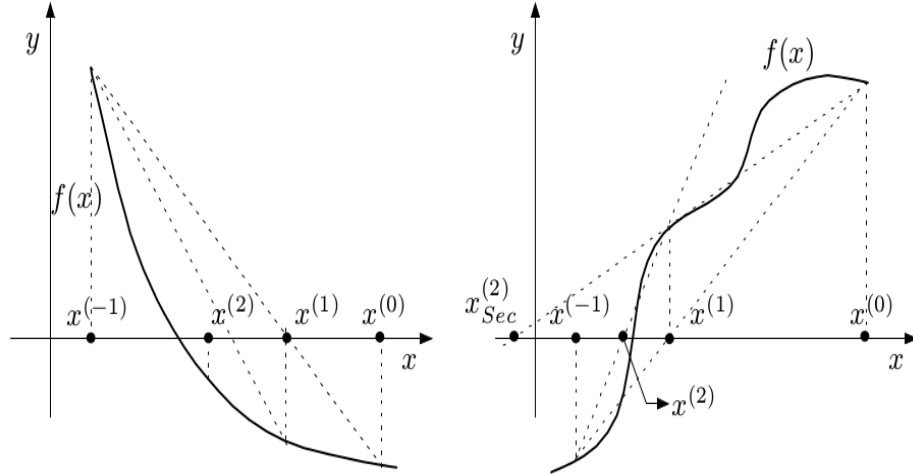
C'est une variante de la méthode des cordes dans laquelle, au lieu de prendre la droite passant par les points  $(x^k, f(x^k))$  et  $(x^{k-1}, f(x^{k-1}))$ , on prend celle passant par  $(x^k, f(x^k))$  et  $(x^{k'}, f(x^{k'}))$ .

Plus précisément, une fois trouvées deux valeurs  $x^1$  et  $x^0$  telles que  $f(x^1)\Delta f(x^0) < 0$ , on pose

$$x^{k+1} = x^k - \frac{x^k - x^{k'}}{f(x^k) - f(x^{k'})} \forall k \geq 0$$

Ainsi, les itérations construites sont toutes contenues dans l'intervalle de départ,  $[x^{-1}, x^0]$ , à la différence de la méthode des cordes.

### 5.2 Interpretation Géométrique



**Fig. 6.3.** Les deux premières étapes de la méthode de la fausse position pour deux fonctions différentes

### 5.3 Implementation du code

```
1 class FalsePosition(Equa_Solver):
2
3     def solve(self):
4         f=self.f
5         a,b=self.a,self.b
6         tol=self.err
7         x_list=[]
8         fx = lambda x: eval(str(f))
9         print("\n\nfunction f : ", f, " dans l'intervalle [", a, ",
10             ", b, "]" \n", "-----")
11
12         if fx(a) * fx(b) > 0:
13             raise SolverException(" f(a) et f(b) doivent etre de
14             signe different !")
15
16         n = 0
17         while abs(b - a) > 2 * tol:
18             c = (a * fx(b) - b * fx(a)) / (fx(b) - fx(a))
19             self.affiche_info(n, c, fx(c))
20             n += 1
21
22             x_list.append(c)
23
24             if fx(c - tol) * fx(c + tol) <= 0:
25                 print('Found solution after', n, 'iterations.')
26                 return x_list
27             if fx(a) * fx(c) > 0:
28                 a = c
29             else:
30                 b = c
31
32         print('Found solution after', n, 'iterations.')
33         x_list.append((a+b)/2)
34         return x_list
```

Listing 6: Méthode de la Fausse Position en Python

## 5.4 Convergence

Pour l'implémentation du calcul de la convergence, il faut donc utiliser la définition de cette dernière défini dans l'introduction.

```
1 def rate(x_list, x_final):
2     e = [abs(x_ - x_final) for x_ in x_list]
3     q = [(log(e[n+1]/e[n]))/(log(e[n]/e[n-1])) for n in range
4         (1, len(e)-1, 1)]
5     return q
```

Listing 7: Calcul de convergence en Python

La méthode de la fausse position, bien qu'ayant la même complexité que la méthode des cordes, a une convergence linéaire. La méthode de la fausse position peut être vue comme une méthode globalement convergente, tout comme celle de dichotomie. La méthode de la fausse position peut se voir un peu comme un entre-deux, entre les méthodes de dichotomie (globalement convergente) et des cordes (meilleur taux de convergence).



## 6 Interface Graphique

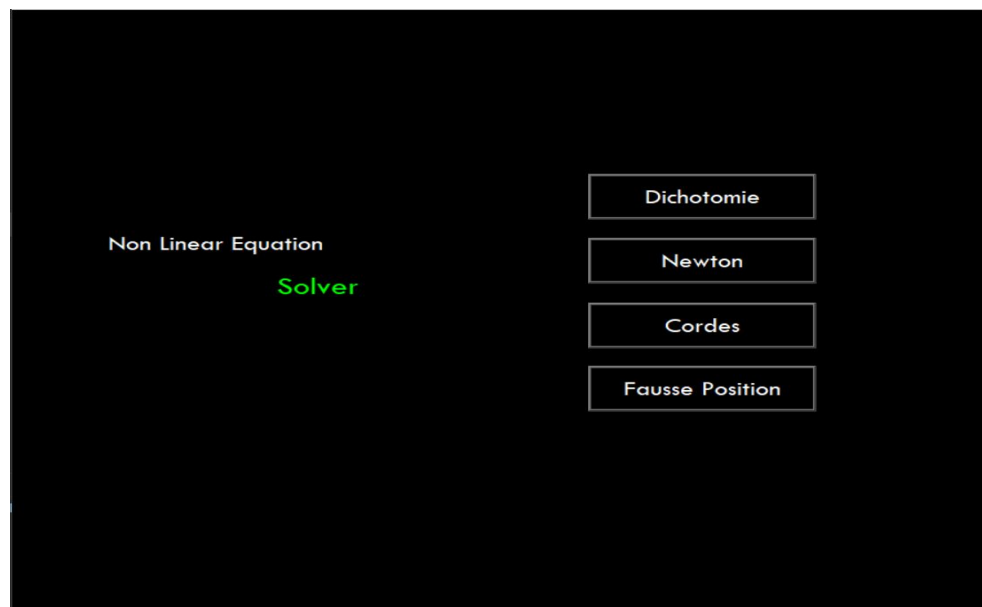
### 6.1 Outils utilisés

Afin de produire une version plus pratique à l'utilisateur plutôt que faire un affichage sur console délicat de suivre les résultats avec on a privilégié de réaliser une interface graphique plutôt simple mais efficace.

La réalisation de cette interface a été effectuée à l'aide de la bibliothèque Tkinter des modules déjà existants sur Python et aussi la réalisation des graphes pour pouvoir afficher les courbes et les différents résultats était fait à l'aide de la bibliothèque Matplotlib qui du fait doit être installé pour le bon fonctionnement du projet.

Notre application est constituée de 5 principales pages :

### 6.2 Accueil



L'interface d'accueil de notre application est constituée des quatre méthodes déjà citées afin de résoudre des équations  $f(x)=0$  et pour accéder à chaque méthode il suffit de suivre le bouton correspondant à la méthode.

Alors dans ce qui suit on va détailler chaque page dans notre interface graphique.

### 6.3 Dichotomie

Méthode De dichotomie

$f(x)$

**a** **b**

Solve

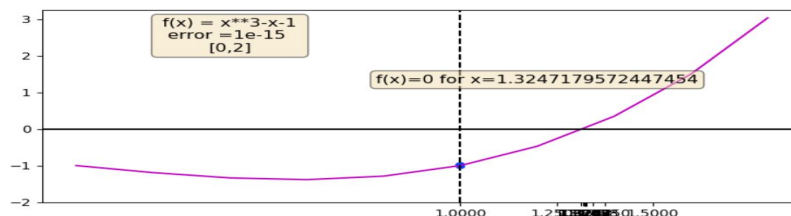
Go Back

Sur cette page on s'occupe de la méthode de dichotomie donc on prend en parametres :

- une fonction  $f$
- un interval  $[a,b]$

Alors si l'utilisateur de notre application rentre bien les infomations attendus et puis il clique sur le bouton solve il aura un affichage :

$x_n$	$f(x_n)$
1.3247179572454115	2.83817414015175e-12
1.3247179572449568	8.988365607365267e-13
1.3247179572447294	-7.105427557601002e-14
1.324717957244843	4.1389114358025836e-13
1.3247179572447862	1.7141843500212417e-13
1.3247179572447575	5.0182080713057076e-14
1.3247179572447436	-1.021405182655144e-14
1.3247179572447507	1.9984014443252818e-14
1.3247179572447472	4.884981308350689e-15
1.3247179572447454	-2.6645352591003757e-15

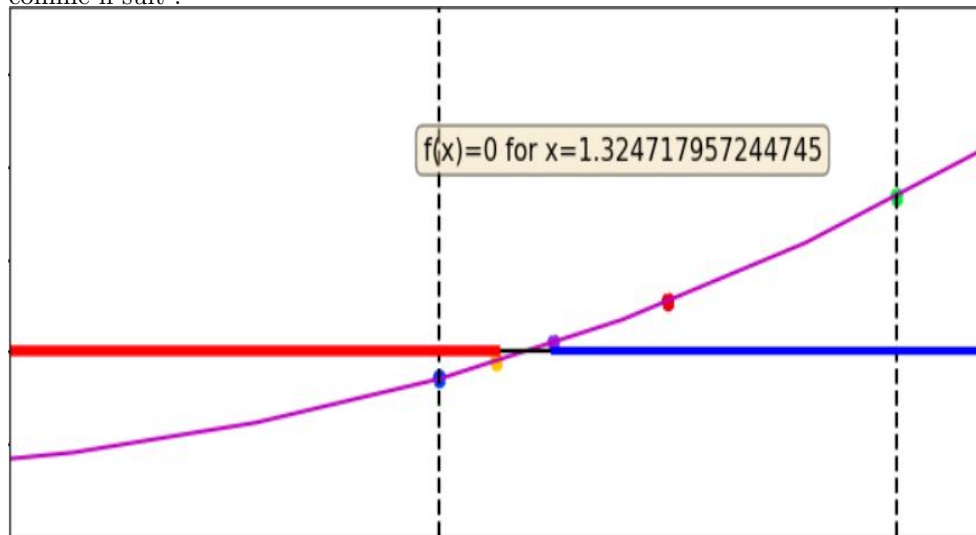


Dans la fenetre on remarque bien les informations relatives à la fonction rentrée mais aussi les résultats retournés apres execution de la méthode de Dichotomie

NB : le tableau affiche les résultats des 10 dernieres itérations de cette méthode

### 6.3.1 Convergence

Pour la méthode de dichotomie la convergence vers la solution est graphiquement représentés par des lignes horizontales qui se rapprochent du point cherché comme il suit :



## 6.4 Newton

**Méthode De Newton**

**f(x)**

**f'(x)**

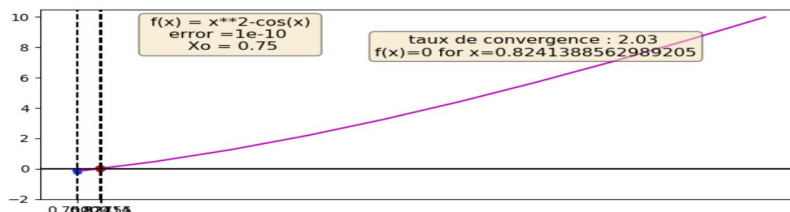
**Xo**

**Error**

Dans cette page on résout les equations  $f(x)=0$  à l'aide de la méthode de Newton donc on prends en arguments une fonction  $f$  et optionnellement sa dérivée et aussi un point de départ de notre récurrence  $x_0$  et un champ error qui indique avec quelle précision les résultats devront etre pris en compte cette valeur est initialisée à  $10^{-15}$  de base.

Après renseignement des champs et cliquer sur le bouton solve une fenetre va s'afficher :

$x_n$	$f(x_n)$
0.75	-0.1691888688738209
0.8275512756621592	0.008160388437742694
0.8241388562989205	1.558931842038369e-05



avec toutes les information comme déjà expliquer dans le paragraphe précédent mais cette fois ci le taux de convergence est explicitement calculé et donné avec les résultats

## 6.5 Cordes

**Méthode De Cordes**

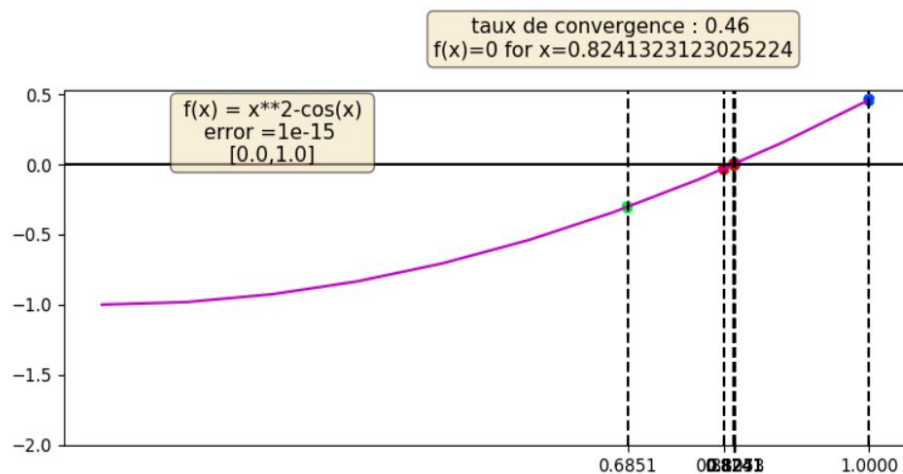
**f(x)**

a                      b

Solve

Go Back

Cette page s'occupe comme son nom l'indique de résolution des equation  $f(x)=0$  à l'aide de la méthode des cordes elle prend en arguments une fonction f et un interval  $[a,b]$  et un clique sur le bouton solve nous donne cette fenetre :



Comme bien illustré sur le graphique on nous permet de suivre la convergence de la solution voulu tout en affichant le taux de convergence correspondant avec les informations nécessaires de la fonction et l'interval.

## 6.6 Fausse Position

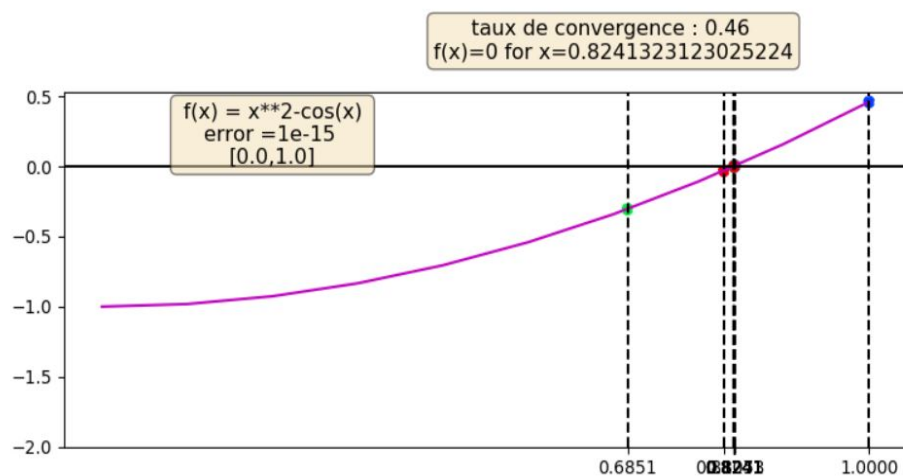
**Méthode De Fausse Position**

**f(x)**

**Solve**

**Go Back**

Cette page s'occupe comme son nom l'indique de résolution des equation  $f(x)=0$  à l'aide de la méthode de fausse position elle prend en arguments une fonction f et un interval  $[a,b]$  et un clique sur le bouton solve nous donne cette fenetre :

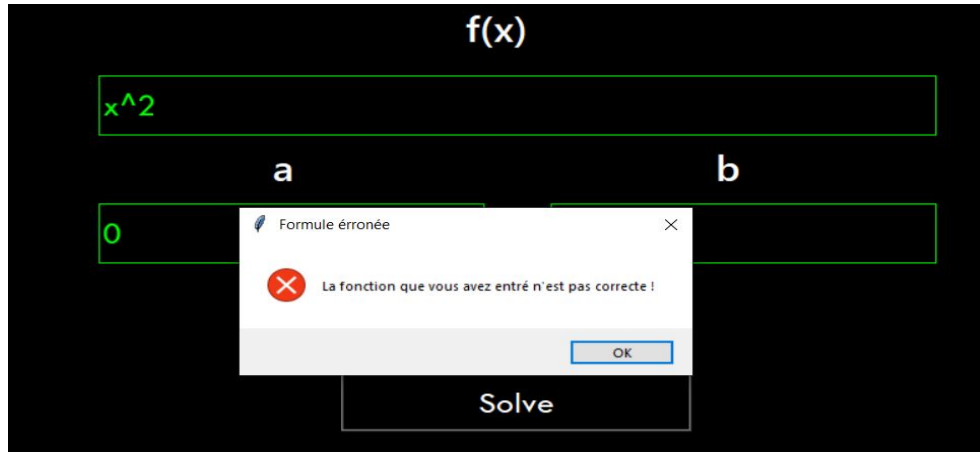


Comme bien illustré sur le graphique on nous permet de suivre la convergence de la solution voulu tout en affichant le taux de convergence correspondant avec les informations nécessaires de la fonction et l'intervall.

## 6.7 Gestion des erreurs

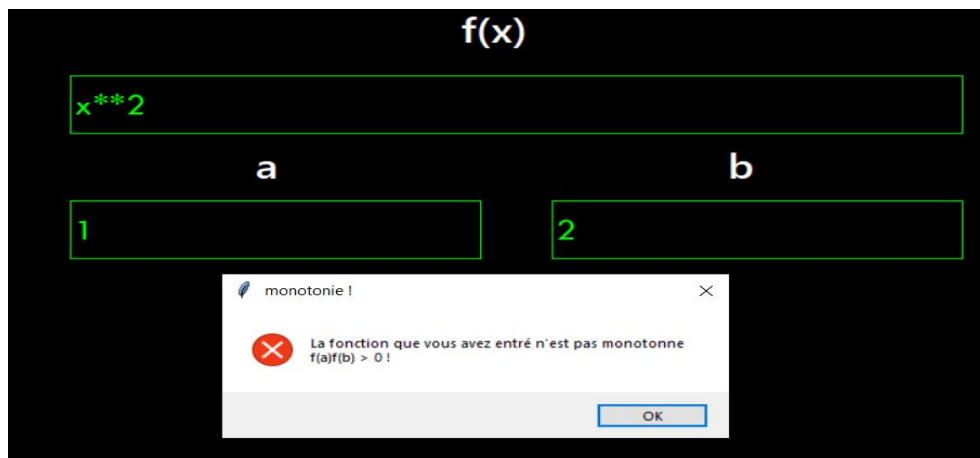
Comme déjà cité dans les différentes fenêtrées de notre application une vérification de données tapées est nécessaire pour le bon fonctionnement du projet alors les vérifications faites sont :

### 6.7.1 Formule Erronée



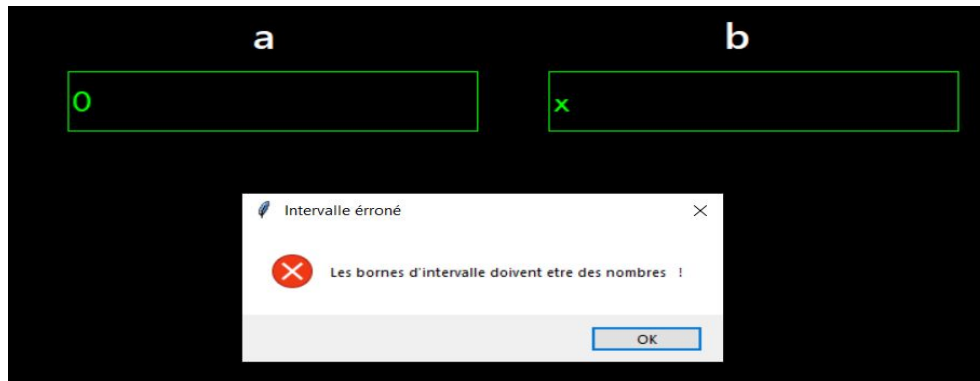
Une formule entrée en champ dédiée est dite erronée si elle est mal exprimée au sens de Python donc toutes les formules sont supposées écrites en syntaxe de Python

### 6.7.2 Monotonie



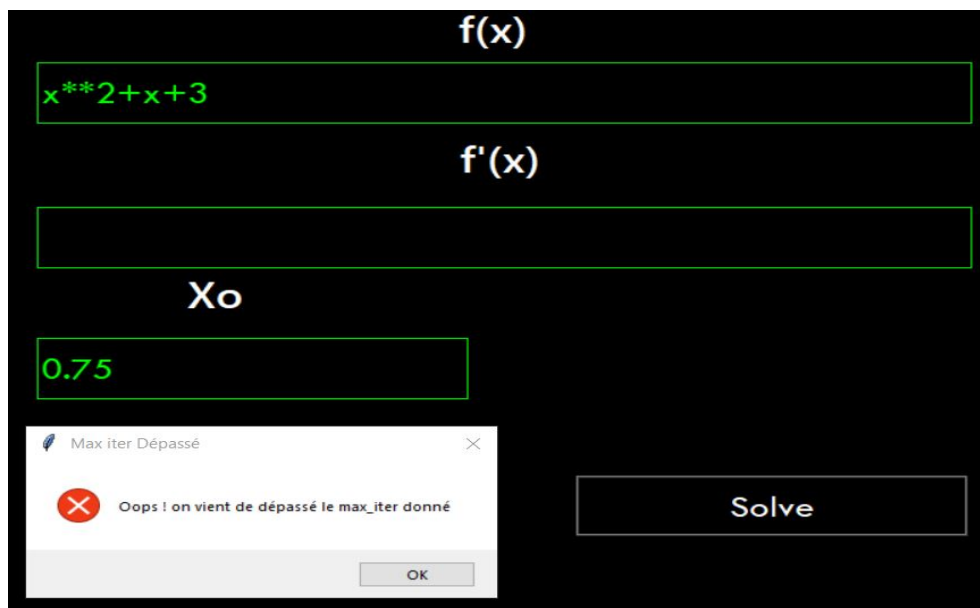
Si on rentre une fonction qui ne vérifie pas la condition des hypothèses de théorème de valeurs intermédiaires une erreur s'affichera indiquant ce fait .

### 6.7.3 Intervall Erroné



Si on rentre un interval qui ne correspond un de ces bornes à des nombres (entiers ou floats) une erreur de type interval erroné s'affichera.

### 6.7.4 Max d'itération atteint sans trouver de solution



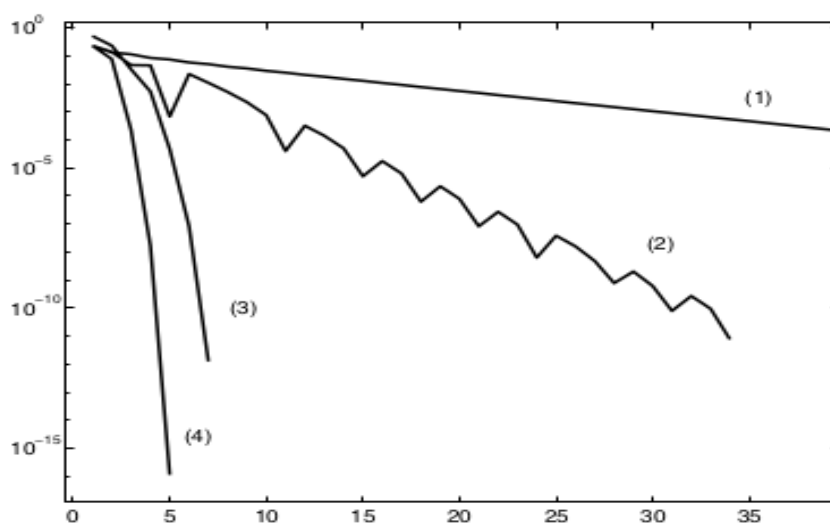
Si notre fonction n'admet pas de valeurs qui le rendent nulle dans l'intervall donné et donc on atteint le maximum d'itération sans trouver de solution pour  $f(x)=0$  alors on affiche la fenetre d'erreur ci dessous



## 7 Conclusion

Ainsi, nous avons pu étudier quelques méthodes utiles pour la résolution d'équations non linéaires en les présentant chacun avec leur caractéristiques propres, leurs avantages et leurs inconvénients.

L'interface graphique, qui a été conçu pour être la plus facilement utilisable et fiable, permet de tester chacune de ces méthodes et d'y avoir une visualisation graphique mais aussi en interface terminale pour plus de détails sur les itérations. De plus, nous avons préparé un fichier test permettant de comparer chacune des méthodes sur 5 résolutions différentes. Pour comparer les vitesses de convergence des méthodes vues précédemment, nous pouvons le voir sur un exemple précis :



historique des convergences de la résolution de  $f(x) = \cos^2(2x) - x^2$  sur l'intervalle  $]0, 1.5[$  pour les méthodes de dichotomie (2), des cordes (3) et de Newton (4). Le nombre d'itérations est reporté sur l'axe des x et l'erreur absolue sur l'axe des y

Cependant, un bon choix qui peut sembler optimal est de combiner certaines de ces méthodes comme par exemple, utiliser la dichotomie qui est lente mais sans surprise pour trouver une première approximation grossière, avant d'utiliser cette dernière comme point d'entrée pour la méthode de Newton qui est rapide et nécessite justement, ce  $x_0$  "grossier".

Comme ouverture théorique, nous pouvons voir également des moyens d'accélération de la convergence.