Révisions de Sup

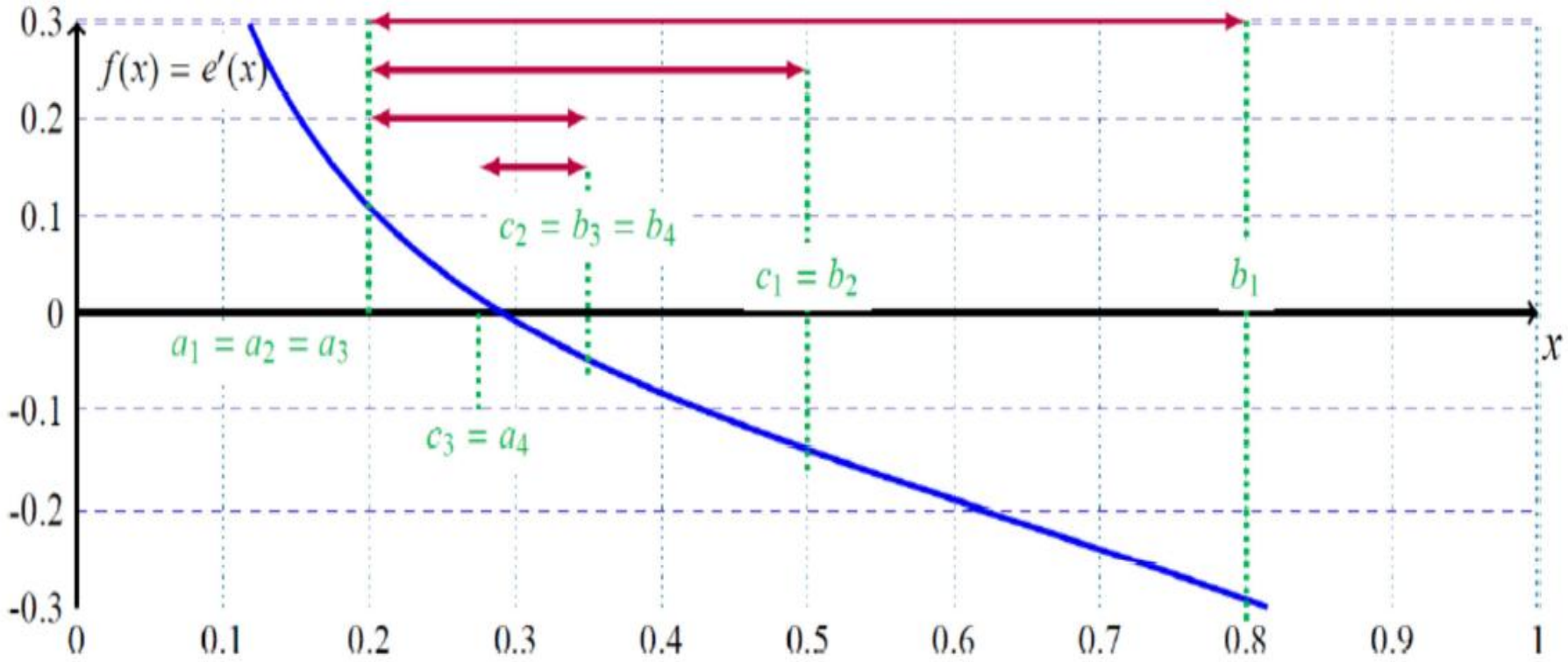
Résolution approchée de l'équation $f(x) = 0$

Méthode de Dichotomie

Méthode de Newton

I. Méthode de dichotomie

1) Principe de la méthode



2) Programmation de l'algorithme de dichotomie

```
def dichotomie(f,a,b,eps,itemax):
    """ Entrées :
    f : la fonction continue   ;    [a,b] : l'intervalle d'étude   ;
    eps>0 : mesure de précision   ; itemax : nombre max d'itérations
    """

    ite=0
    while d-g > 2*eps and ite < itemax :
        m = (a+b)/2
        if f(m)==0:
                return m    # quel bol !
        elif f(g)*f(m)< 0:
                d=m
        else :
                g=m
        ite=ite+1
    return m
```

Rem : La *méthode par Dichotomie* est déjà implémentée dans le *module Scipy* de Python :

```
>>> import scipy.optimize
>>> x=scipy.optimize.bisect(f,a,b,ε,itemax)
```

Voir l'aide pour plus de détails

3) Convergence de la méthode de dichotomie

Proposition :

Si la fonction *f* est :
- définie et continue sur l'intervalle [*a*, *b*],
- n'admet qu'une seule racine sur l'intervalle [*a*, *b*],

alors la ***méthode de dichotomie converge***.

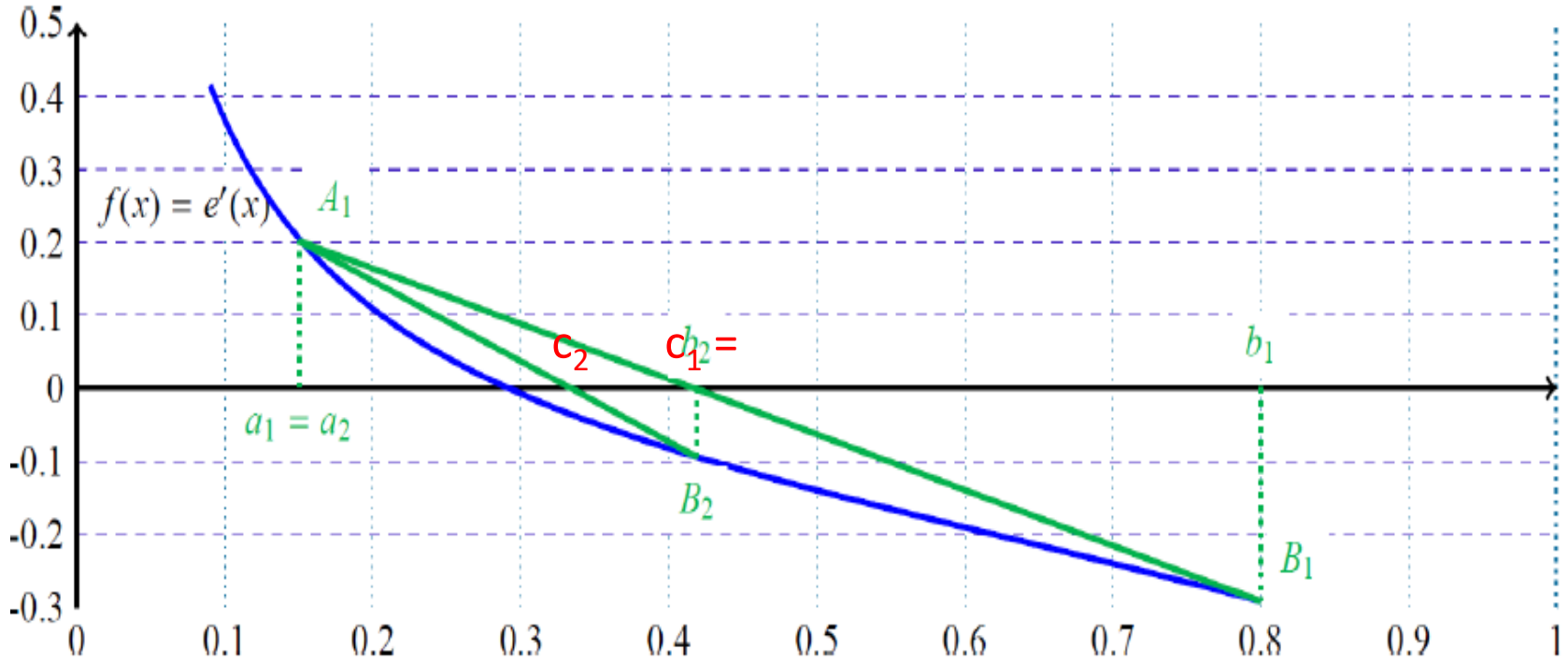
4) Complexité de l'algorithme pour le test d'arrêt « tant que d-g > 2 ε » :

Dénombrons le nombre d'itérations.
La boucle « tant que » s'exécute jusqu'à ce que :

$$\frac{b-a}{2^n} \leq 2\varepsilon \leq \frac{b-a}{2^{n-1}} \Leftrightarrow \frac{b-a}{2\varepsilon} \leq 2^n \leq \frac{b-a}{\varepsilon} \Leftrightarrow \log_2\left(\frac{b-a}{2\varepsilon}\right) \leq n \leq \log_2\left(\frac{b-a}{\varepsilon}\right)$$

La complexité de cet algorithme est donc de type ***logarithmique*** en **O(ln(b-a)).**

Addendum : amélioration par la méthode de Lagrange (ou méthode de la corde)



Exercice : programmer en Python cette méthode.

III. Résolution par la méthode de Newton (- Raphson)

(1669)

1) Principe de la méthode de Newton

Soit f une fonction dérivable sur un intervalle I et telle que l'équation $f(x)=0$ admette un zéro unique α sur l'intervalle I.

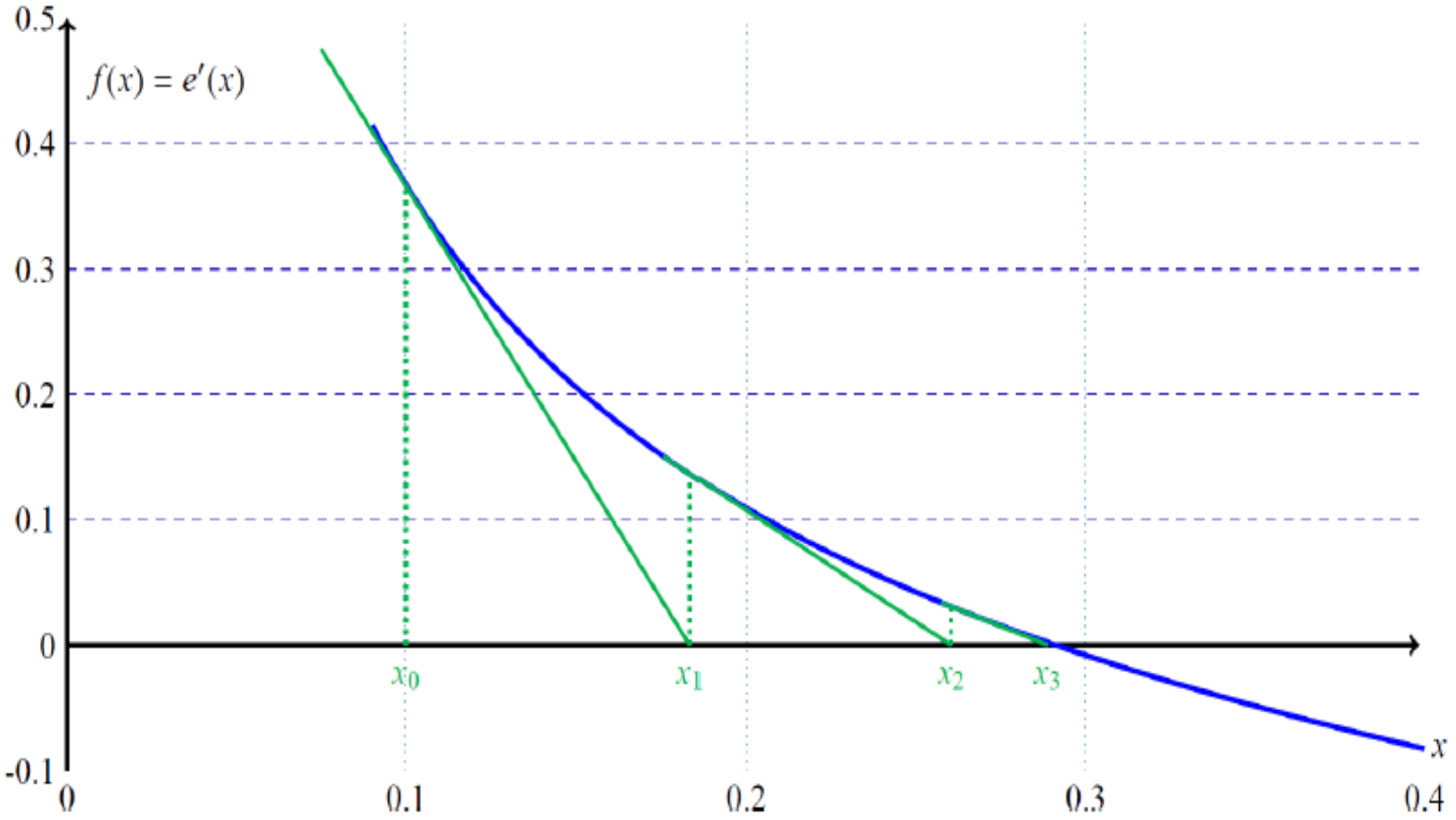
A partir d'une abscisse choisie x_0, la méthode de Newton consiste à

approcher, itérativement, la fonction f par sa fonction « affine tangente » f_T .

Ceci conduit à la construction de la suite (x_n) définie par :

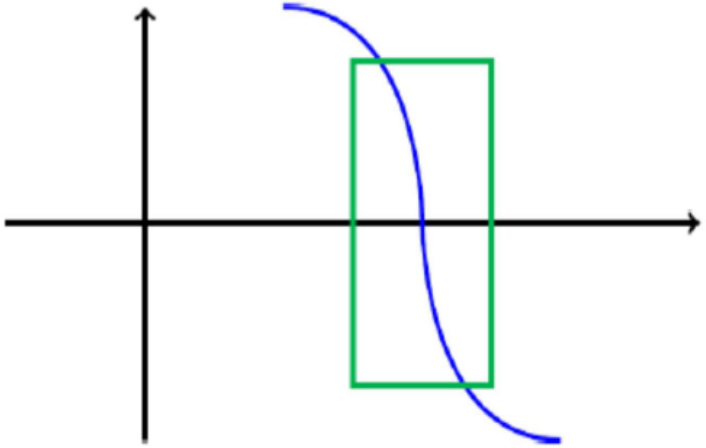
$$\forall n \in \mathbb{N}, \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Principe de la méthode de Newton

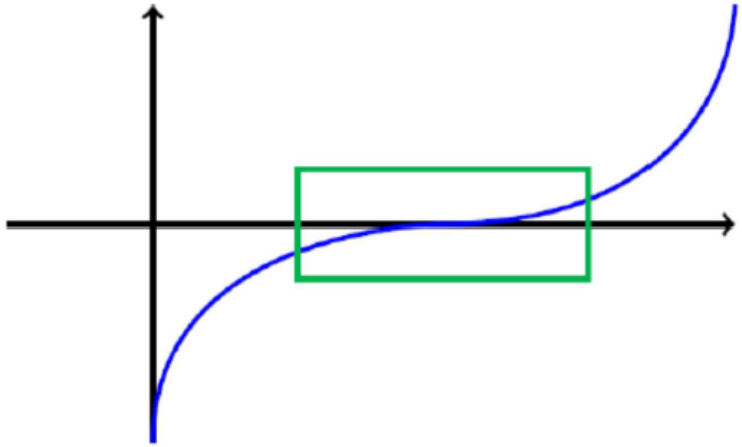


Choix d'un critère de convergence

On fixe **ε >0.**



Courbe à **fort** gradient aux alentours de la racine, le test « tant que |fc| > ε » sera plus précis.



Courbe à **faible** gradient aux alentours de la racine, le test « tant que $b_n - a_n > 2\,\varepsilon$ » sera plus précis.

On itère l'algorithme **tant que :**

$$|f(x_n)| > \varepsilon$$

$$|x_{n+1} - x_n| > \varepsilon$$

2) Programmation de l'algorithme de la méthode de Newton (revoir efficacement la SUP)

Rem : La *méthode de Newton* est déjà implémentée dans le *module Scipy* de Python :

```
>>> import scipy.optimize
>>>x=scipy.optimize.newton(f,x0,f',ε,itemax)
```

Voir l'aide pour plus de détails

3) Convergence de la méthode de Newton

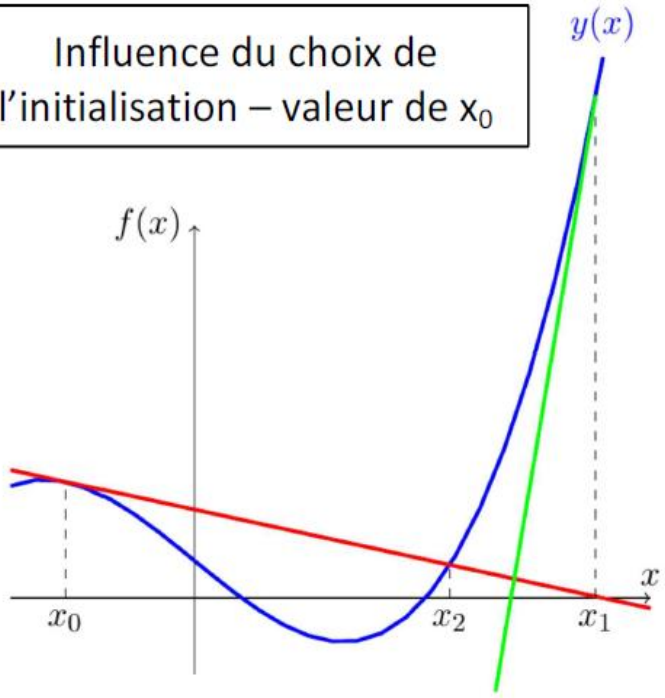
Théorème (admis) : soit *f* de classe C^1 sur *I* et admettant une unique racine sur *I.*

Si :

- *f est strictement monotone sur I*
- *f' est croissante sur I (on dit que f est* convexe)

Alors la méthode de Newton converge.

Influence du choix de l'initialisation – valeur de x_0



4) Problème du calcul de la dérivée numérique $f'(x_n)$

Si l'on ne peut pas calculer algébriquement la dérivée, alors, on peut utiliser :

- l'approximation « à un pas » $h > 0$ fixé :

$$\forall n \in \mathbb{N}, \quad f'(x_n) \approx \frac{f(x_n + h) - f(x_n)}{h}$$

- plus précise, l'approximation « à 2 pas », avec $h > 0$ fixé :

$$\forall n \in \mathbb{N}, \quad f'(x_n) \approx \frac{f(x_n + h) - f(x_n - h)}{2h}$$

IV. Comparaison de la dichotomie et de Newton

Définition :
La **robustesse** d'une **méthode** d'analyse est une mesure de sa capacité à supporter sans conséquence de « petites » variations des paramètres.

– La méthode par ***dichotomie*** est **robuste mais lente.**

(vitesse de convergence logarithmique)

– La méthode de ***Newton*** est **peu robuste mais rapide.**

(vitesse de convergence quadratique)

♥ Dans le cas où l'on cherche rapidité et stabilité, on peut utiliser :
-la méthode par dichotomie dans un premier temps pour localiser le zéro de la fonction,
- puis la méthode de Newton une fois proche de la solution.

il existe d'autres algorithmes de recherche avec

leurs avantages et inconvénients

V. Résolution par la méthode du point fixe (voir Maths)

1) Principe de la méthode du point fixe

Le principe de cette méthode consiste à transformer l'équation f(x) = 0 en une équation équivalente g(x) = x où g est une fonction auxiliaire « bien » choisie.

En général : **g(x) = f(x) + x**

Le point c tel que f(c) = 0 est alors le *point fixe de g, tel que* ***g(c) = c***.
Approcher les zéros de f revient à approcher les points fixes de g.

On construit alors une suite (x_n) telle que :

$$\begin{cases} x_0 \text{ dans le voisinage de l de c} \\ \forall n \geq 0,\ x_{n+1} = g(x_n) \end{cases}$$

2) Programmation de la méthode du point fixe

Rem : La *méthode du point fixe* est déjà implémentée dans le *module Scipy* de Python :

```
>>> import scipy.optimize
>>>x=scipy.optimize.fixed_point(g,x0, ε,itemax)
```

Voir l'aide pour plus de détails

3) Convergence de la méthode du point fixe

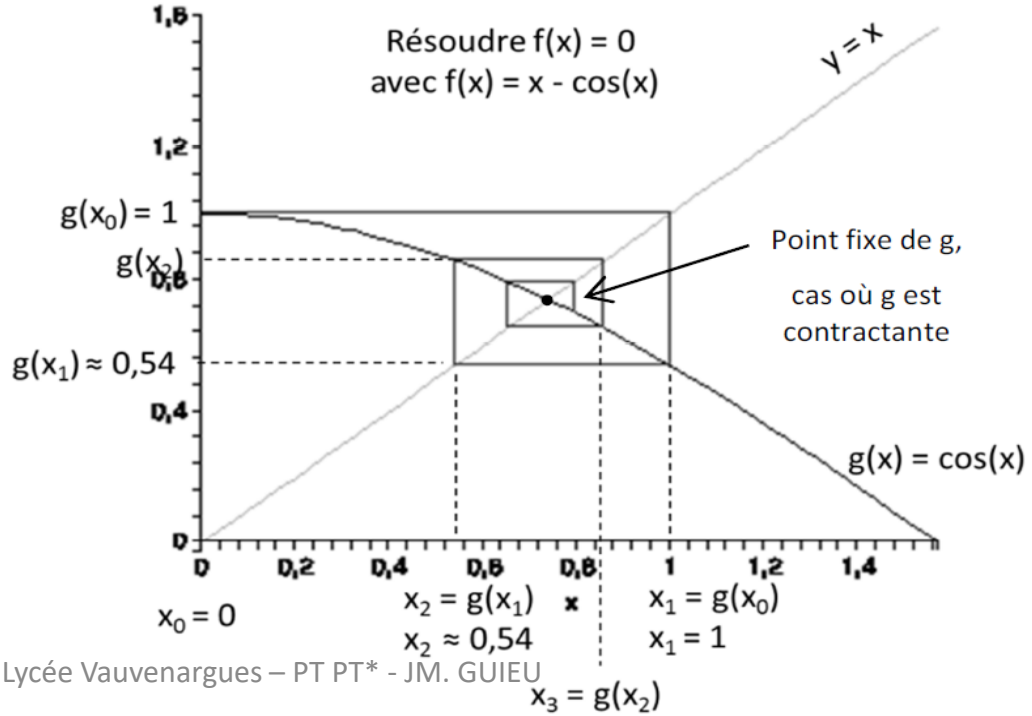
Théorème (accroissements finis) :
Si **g contractante** dans un voisinage I de c, c'est-à-dire :

$$|g'(x)| < 1, \quad \forall x \in I,\ I \text{ voisinage de } c,$$

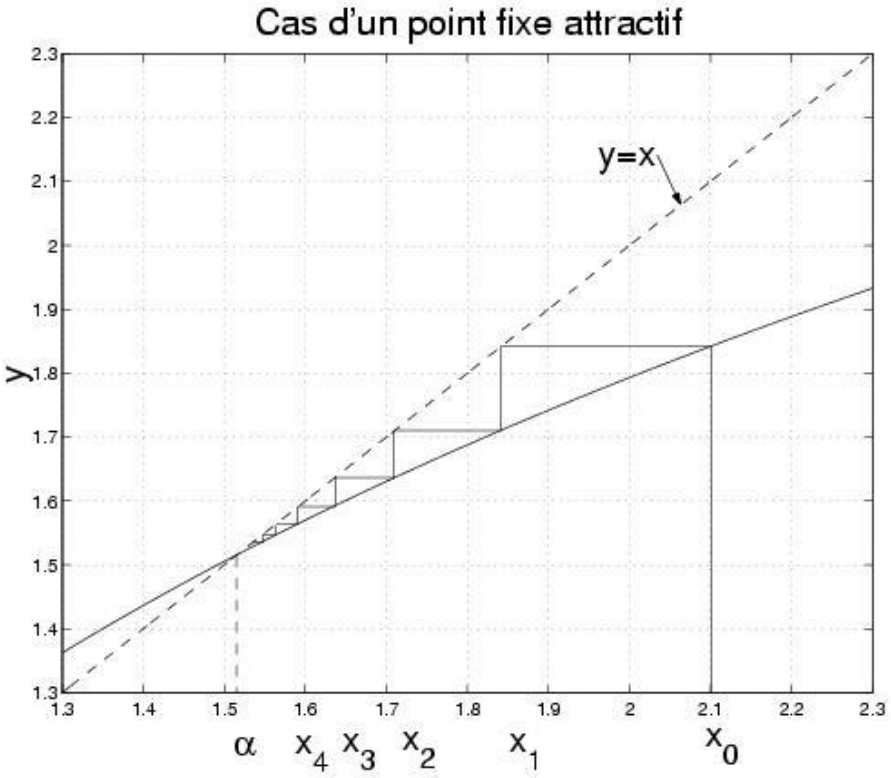
Alors la méthode du point fixe converge.

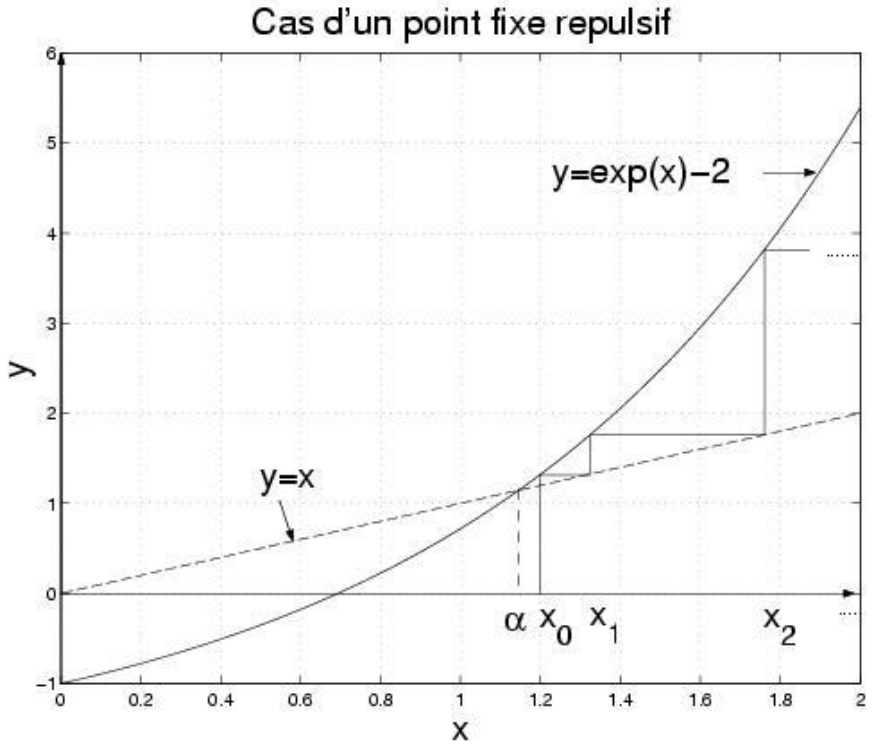
convergence « en colimaçon » ou « spirale »

Illustration graphique de la méthode du point fixe



Autre illustrations graphiques de la méthode du point fixe :





convergence « en escalier »

La fonction n'est pas contractante au voisinage de c.
La méthode diverge !!