Résolution d'équations stationnaires 1D

Objectif :
L'objectif de ce cours est de traiter des problèmes stationnaires (constants au cours du temps), linéaires ou non, conduisant à la résolution approchée d'une équation algébrique ou transcendante à une seule variable réelle.

I. Introduction

I.1. Support de l'étude

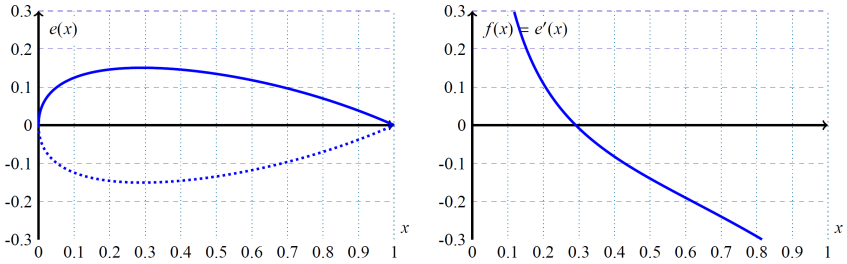
Comme support de cours, nous prenons un profil d'aile d'avion dont la demi-épaisseur $e(x)$ est approchée par l'équation :

$$e(x) = 0.15 \times \left(3.7 \times \sqrt{x} - 3.4 \times x - 0.3 \times x^4\right), \forall x \in [0,1]$$

Trouver la position x pour laquelle le profil est le plus épais, revient à résoudre :

$$f(x) = 0.15 \times \left(\frac{3.7}{2.\sqrt{x}} - 3.4 - 1.2 \times x^3\right) = 0$$

Le programme en annexe trouve cette racine en comparant différentes méthodes.



I.2. Linéarité

Soit f une fonction définie de E, un espace vectoriel, dans K, un corps commutatif. f est *linéaire*, si elle vérifie la propriété suivante :

$$\forall (x,y) \in E^2, \forall (\lambda,\mu) \in K^2, \qquad f(\lambda.x + \mu.y) = \lambda.f(x) + \mu.f(y)$$

Exemple : Résoudre une équation linéaire 1D (droite $f(x) = a.x + b$) dans $\mathbb{R}$ ou $\mathbb{C}$ est immédiat : $\forall a \neq 0,\ a.x + b = 0 \Longrightarrow x = -\frac{b}{a}$.

Résoudre un système linéaire de n équations à n inconnues suppose la résolution d'un système matriciel du type :

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} . \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \Leftrightarrow [A].[X] = [B] \overset{\det(A)\neq 0}{\Rightarrow} [X] = [A^{-1}].[B]$$

Nous verrons plus tard comment le résoudre par un pivot de Gauss.

I.3. Non linéarité et méthodes itératives

La vraie difficulté est de trouver les racines pour les équations non linéaires, ce qui concerne la majorité des cas. A part certains polynômes, on ne sait pas trouver les racines analytiquement. De plus, beaucoup d'équations ne sont même pas *algébriques* (polynômiales à coefficients rationnels). Elles sont dites *transcendantes*. La suite du cours présente des méthodes générales pour trouver des approximations numériques des racines.

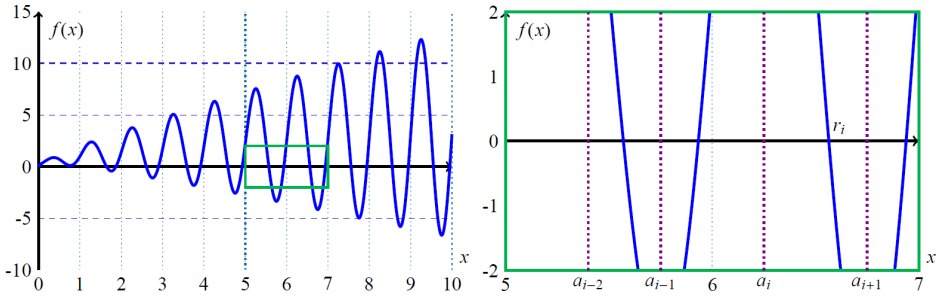
Ces méthodes sont **itératives** : on part d'une valeur approchée grossière de la solution, et on en améliore la précision par un algorithme bien choisi.

II. Convergence

II.1. Séparation des racines

La plupart des méthodes de recherche de racines d'une fonction, se comportent bien si **une seule racine** r (valeur pour laquelle la fonction s'annule) est présente dans l'intervalle d'étude. **Séparer les racines** r_i (i$^{\text{ème}}$ annulation de la fonction sur l'intervalle de départ) revient à trouver un intervalle $]a_i, a_{i+1}[$ où cette racine est unique.

Exemple : $f(x) = \sqrt{x} + x.\sin(2.\pi.x)$



Il convient alors de faire un test aux bornes de l'intervalle d'étude pour éliminer les cas d'inadéquation de la comparaison à zéro, i.e, les cas sans racines sur l'intervalle. Il est en effet, possible d'élaborer une procédure permettant de **déterminer la parité du nombre de racines** sur l'intervalle :

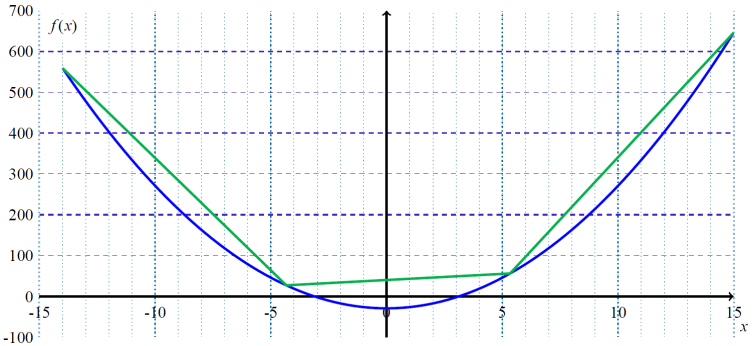
- Posons $p_i = f(a_i).f(a_{i+1})$
- En tenant compte de l'ordre de multiplicité des racines, nous avons :
  - si $p_i < 0$, il existe $2.n+1$ racines dans $]a_i, a_{i+1}[,\ n \in \mathbb{N}$
  - si $p_i > 0$, il existe $2.n$ racines dans $]a_i, a_{i+1}[,\ n \in \mathbb{N}$
  - si $p_i = 0$, alors a_i ou a_{i+1} est racine (voire les deux).

Evidemment, cet algorithme ne donne pas la valeur de n. Nous verrons plus loin l'utilité de ce test, notamment dans la recherche de la racine par dichotomie.

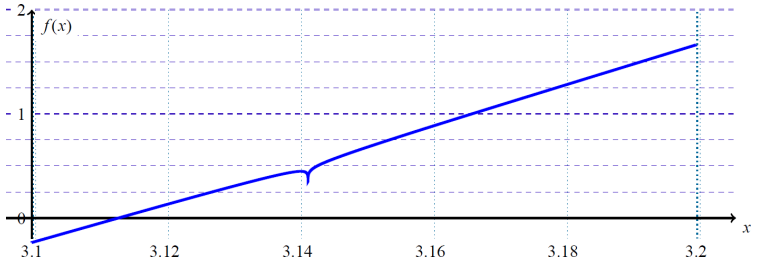
II.2. Représentations graphiques

Les représentations graphiques sont utiles parfois pour avoir une idée de la localisation des racines, et des propriétés de convergence de la méthode choisie. Cela permet d'obtenir à vu d'œil un premier intervalle ou un premier candidat pour démarrer une méthode numérique.
Cependant, il convient aussi de se méfier des représentations graphiques. Pour cela, il est préférable d'observer attentivement l'expression de la fonction et son domaine de définition.

Exemple : $f(x) = 3.x^3 + \dfrac{1}{\pi^4}.\ln\left[(\pi - x)^2\right] - 29$



Ci-dessus, les courbes sont tracées entre -14 et 15. La courbe verte n'ayant que 4 points, on ne constate pas d'intersection avec l'axe des abscisses. La courbe bleue comporte 100 points.



Avec le zoom sur l'intervalle $[3.1; 3.2]$, il semble encore qu'il n'y ait qu'une racine sur l'intervalle. Or $\lim\limits_{x \mapsto \pi} f(x) = -\infty$. Par continuité sur les intervalle $[3.14; \pi[$ et $[\pi; 3.15]$, la fonction f admet donc 3 racines dans l'intervalle $[3.1; 3.2]$.

II.3. Critères de convergence et terminaison

Les méthodes numériques ne permettent pas d'obtenir une réponse formelle à un problème. En revanche, elles permettent d'approcher la solution d'un problème avec une précision fonction du type de stockage des nombres (cf 1er semestre) et du calcul numérique nécessaire pour obtenir cette solution (cf la sensibilité de certains calculs aux erreurs d'arrondis).

Tests de convergence pour terminer l'itération

- sur la valeur de x :

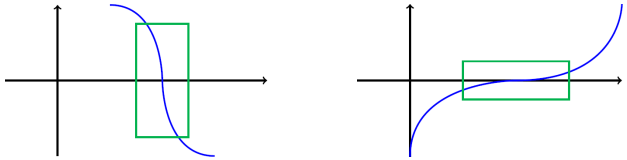
$$\left|\frac{x_{n+1} - x_n}{x_n}\right| < \varepsilon_x \quad \text{ou} \quad |x_{n+1} - x_n| < \varepsilon_x$$

  La seconde solution est préférable lorsque l'on travaille au voisinage de 0... ce qui est toujours préférable pour éviter des surpassements de capacité. On peut en effet souvent normaliser les grandeurs pour travailler avec des solutions de l'ordre de 1, ce qui maximise la résolution et évite des surpassements de capacité.
  Une tolérance (relative) de 10^{-8} est raisonnable. En revanche, chercher à obtenir la limite théorique de 10^{-16} (en 64 bits) est a priori non nécessaire et impossible.

- sur la valeur de la fonction f :

$$|f(x_n)| < \varepsilon_f$$

Suivant le gradient (pente) de la fonction, un des deux tests sera privilégié. Pour un fort (faible) gradient, il convient de suivre la valeur de f (de x).



Fort gradient Faible gradient

En l'absence d'information, on peut suivre les deux méthodes.

II.4. Erreur et ordre de convergence

DÉFINITION : Soit x_n la n$^{\text{ième}}$ valeur calculée pour résoudre l'équation $f(x) = 0$ sur l'intervalle donné. Soit r la solution exacte de cette équation. L'**erreur en abscisse** à l'itération n est

$$\varepsilon_n = x_n - r$$

DÉFINITION : La méthode numérique admet un ordre (de convergence) p si

$$\exists\, M > 0, \forall n \in \mathbb{N}, \quad |x_{n+1} - r| < M|x_n - r|^p$$

- Si $p = 1$, la méthode est dite **linéaire**.
- Si $p > 1$, la méthode est qualifiée de **super-linéaire**.
  En particulier, si $p = 2$ la méthode est dite **quadratique**.

CONSÉQUENCES :

- **Si la méthode admet un ordre, alors elle peut converger vers la racine.**
  En effet, on montre que :
  — pour $p = 1$, $|x_n - r| < M^{n+1}.|x_0 - r| \underset{n\to\infty}{\longrightarrow} 0$ si $M < 1$;
  — pour $p > 1$,

  $$|x_n - r| < \left(M^{\frac{1}{p-1}}.|x_0 - r|\right)^{p^{n+1}} .M^{-\frac{1}{p-1}}$$

  donc **si $x_0 - r$ est choisi suffisamment petit** pour que $M^{\frac{1}{p-1}}|x_0 - r| < 1$, alors $x_n \underset{n\to\infty}{\longrightarrow} r$.
- **Plus l'ordre p est grand, plus la méthode est susceptible de converger rapidement.**

III. Méthode par dichotomie

III.1. Principe

Soit f une fonction continue sur un intervalle $[a, b]$ tel que $f(a)f(b) < 0$.
D'après le théorème des valeurs intermédiaires, f admet au moins une racine sur cet intervalle.

Le principe est de :

- **diviser l'intervalle $[a, b]$ en deux parts égales** ;
- de **conserver la partie contenant la racine** ;
- et d'y reproduire l'opération jusqu'à ce que le critère de convergence soit satisfait.

Algorithme

À partir d'un intervalle initial donné $[a, b]$, encadrant une racine de la fonction f étudiée :

— Initialisation : $a_0 = a$ et $b_0 = b$

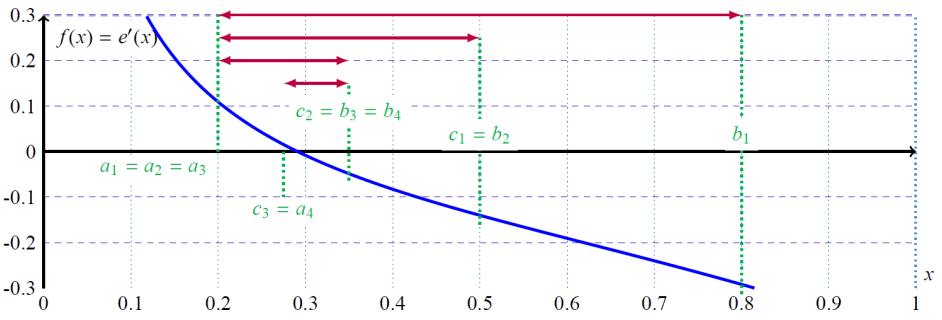
— Hérédité : calculer le point c_n milieu de l'intervalle : $c_n = \dfrac{a_n + b_n}{2}$

  — si $f(a_n).f(c_n) > 0$, il n'y a pas de racine dans l'intervalle $[a_n, c_n]$. La racine est donc dans l'intervalle $[c_n, b_n]$. On donne alors à a_{n+1} la valeur de c_n

  — sinon, la racine est dans $[a_n, c_n]$. On donne alors à b_{n+1} la valeur de c_n

— Critère d'arrêt : Evaluer le critère de convergence

— Recommencer tant que le critère de convergence n'est pas satisfait



III.2. Convergence de la méthode

Preuve de convergence

$$|c_n - r| \leq |b_{n+1} - a_{n+1}| = \frac{1}{2}|b_n - a_n| = \frac{|b-a|}{2^{n+1}} \underset{n\to\infty}{\longrightarrow} 0$$

Ordre de convergence

D'après ce qui vient d'être montré, la convergence **s'apparente à une convergence linéaire**.
Toutefois, on ne peut pas dire qu'elle admette un ordre : on peut avoir $|c_{n+1} - r| > K|c_n - r|$ avec K aussi grand qu'on veut, à condition d'avoir r très proche du milieu de $[a_n; b_n]$.

Critère d'arrêt

Pour test d'arrêt, il est possible de prendre :

— un encadrement de la solution exacte : $b_n - a_n \leq \varepsilon$

— la valeur de la fonction au point calculé : $|f(c_n)| < \varepsilon$

Le nombre d'itération n pour obtenir une erreur inférieure à ε est tel que :

$$\varepsilon \geq \frac{b-a}{2^n} \Rightarrow 2^n \geq \frac{b-a}{\varepsilon} \Rightarrow n.\ln(2) \geq \ln\left(\frac{b-a}{\varepsilon}\right) \Rightarrow n \geq \frac{1}{\ln 2}.\ln\left(\frac{b-a}{\varepsilon}\right)$$

Avec $\varepsilon = 10^{-p}$ (resp. $\varepsilon = 2^{-p}$) pour une approche décimale (resp. binaire) :

$$n \geq \frac{\ln(b-a) + p.\ln(10)}{\ln 2} \qquad \left(\text{resp. } n \geq p + \frac{\ln(b-a)}{\ln 2}\right)$$

En particulier, pour une racine de l'ordre de 1, on l'obtiendra avec n bits significatifs ($\varepsilon = 2^{-p}$) en un nombre d'opérations de l'ordre de n.

CONCLUSION :
La vitesse de convergence est **lente**
mais la méthode est **robuste** ... pour qu'elle converge, il suffit que

— la fonction f soit définie et continue sur l'intervalle $[a, b]$;

— la fonction n'admette qu'une seule racine sur l'intervalle $[a, b]$.

III.3. Application : déterminer $\sqrt{2}$

On applique la méthode par dichotomie avec $f(x) = x^2 - 2$ sur l'intervalle $[1, 2]$.

itérations	valeur
1	1,5
2	1,25
3	1,375
4	1,4375
5	1,40625
6	1,421875

7	1,4140625
13	1,414245605 47
18	1,414211273 19
21	1,414213657 38
23	1,414213538 17
28	1,414213560 52
32	1,414213562 15

IV. Méthode de Newton

IV.1. Principe

Soit f une fonction de classe $\mathcal{C}^1$ sur $[a, b]$, telle que $f(a).f(b) < 0$ (toujours avec une seule racine r dans l'intervalle...).

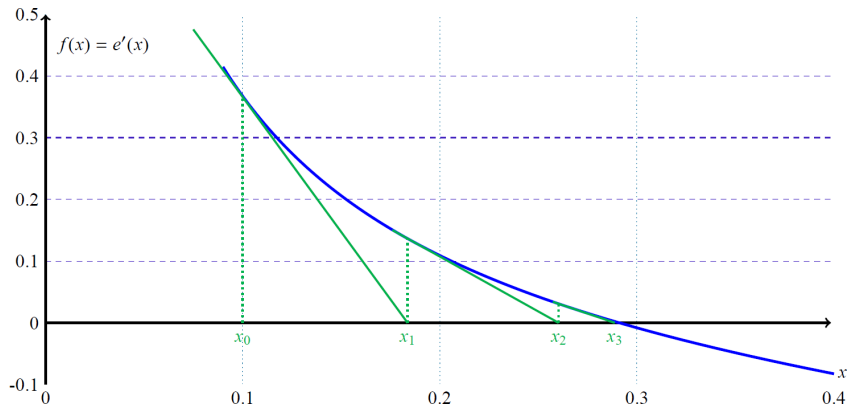
Le principe consiste à

- **approximer la courbe par sa tangente** au point courant ;
- et chercher l'**intersection avec l'axe des abscisses**.

De nouveau on itère le processus jusqu'à satisfaction du critère de convergence.

Ainsi, à partir d'un point $x_0 \in [a, b]$, on obtient une meilleure estimation x_1 de r vérifiant

$$f'(x_0) = \frac{0 - f(x_0)}{x_1 - x_0} \Leftrightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$



Algorithme

— Initialisation : Choisir un premier candidat x_0 dans l'intervalle I

— Hérédité : calculer $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$

— Terminaison : tester le critère de convergence $|f(x_{n+1})| < \varepsilon_f$ ou $|x_{n+1} - x_n| < \varepsilon_x$.

IV.2. Convergence de la méthode

On suppose ici la fonction f de classe $\mathcal{C}^2$ sur $[a, b]$. La formule de Taylor-Lagrange appliquée en r au voisinage de x_n s'écrit à l'ordre 2 :

$$f(r) = f(x_n) + f'(x_n)(r - x_n) + \frac{f''(\xi_n)}{2}(r - x_n)^2$$

où ξ_n est compris entre x_n et r. Or comme $f(r) = 0$, on a

$$-\frac{f(x_n)}{f'(x_n)} + x_n - r = \frac{f''(\xi_n)}{2f'(x_n)}(x_n - r)^2 \Leftrightarrow x_{n+1} - r = \frac{f''(\xi_n)}{2f'(x_n)}(x_n - r)^2$$

Si $f'(r) \neq 0$, alors f' ne s'annule pas et est de signe fixé au voisinage de r car elle est continue. On a alors un voisinage de r sur lequel $\left|\dfrac{f''(\xi_n)}{2f'(x_n)}\right| < M$.

La méthode est donc d'ordre 2 (quadratique) [1].

Conclusion : La convergence de la méthode de Newton est donc **quadratique** sous les conditions suivantes :

— $f'(x) \neq 0$ sur $[a; b]$,

— $f''(x)$ est bornée sur $[a; b]$,

— x_0 est assez proche de r.

IV.3. Application : déterminer $\sqrt{2}$

On applique la méthode de Newton avec $f(x) = x^2 - 2$ sur l'intervalle $[1, 2]$.

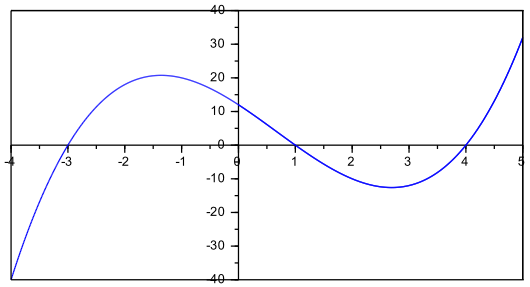
1. Plus généralement, les méthodes de Schröder sont basés sur les n premières dérivées d'une fonction f pour une méthode d'ordre $n+1$. Dans le cas de la méthode de Newton, on n'utilise que la dérivée première.

itérations	valeur
1	1,5
2	1,4166666666666667
3	1,4142156862745099
4	1,4142135623746899
5	1,4142135623730951

IV.4. Importance de l'initialisation

Dans le cas où les conditions de convergence ne sont pas satisfaites, il peut y avoir des comportements étonnants.

Exemple : $f(x) = x^3 - 2x^2 - 11x + 12$



— $x_0 = 2.352\,875\,27$, $x_\infty = 4$,

— $x_0 = 2.352\,841\,72$, $x_\infty = -3$,

— $x_0 = 2.352\,837\,35$, $x_\infty = 4$,

— $x_0 = 2.352\,836\,323$, $x_\infty = 1$,

Conclusion : La méthode de Newton est donc **plus rapide** que la dichotomie (quadratique), mais **moins robuste**...

Elle nécessite :

— une dérivée ne s'annulant pas ; $f'(x) \neq 0$ sur $[a; b]$ [a],

— un démarrage x_0 assez proche de r

a. Plus exactement, il faut que la forme de la courbe soit telle que le point courant ne sorte pas de l'intervalle où la dérivée ne s'annule pas... pas toujours facile à réaliser.

En pratique, on associe les deux méthodes en appliquant à la fois la dichotomie pour se rapprocher de la solution, puis Newton pour converger très rapidement. La méthode de Newton étant au moins quadratique, en moins de 10 itérations on obtient une solution très précise. Le principal souci est d'être dans le cadre des hypothèses (ce qui est souvent le cas pour les problèmes d'ingénierie classiques).

V. Conclusion

— La méthode la plus naïve est la dichotomie. A partir d'une fonction continue sur un intervalle $[a, b]$, avec $f(a).f(b) \leq 0$, la recherche de x tel que $f(x) = 0$ est simple et robuste mais lente (convergence linéaire).

— Si on cherche une valeur avec précision, une convergence quadratique sera préférée (méthode de Newton). Il convient alors d'avoir une idée approximative de la solution pour proposer un premier candidat assez proche, de sorte que la fonction y présente de bonnes propriétés. Si on ne souhaite pas calculer la dérivée de la fonction, on peut alors utiliser la méthode de la sécante.

— Dans le cas où l'on cherche rapidité et stabilité, on peut utiliser la méthode par dichotomie dans une premier temps pour localiser le zéro de la fonction, puis appliquer un algorithme de Newton. En cas d'instabilité de l'algorithme de Newton, il est toujours possible de réutiliser une méthode par dichotomie.

— Enfin, mentionnons qu'il existe aussi d'autres types d'algorithmes plus subtils qualifiés de méthodes métaheuristiques (recuit, tabou, algorithmes génétiques, colonies de fourmis...).

ANNEXE - Programme de la dichotomie

```
import numpy as np
import matplotlib.pyplot as plt
import time

def f(x): # fonction dont on cherche le zéro
    return 0.15*( 3.7/2/np.sqrt(x) -3.4 - 1.2*x**3 )

def fp(x): # dérivée de f(x) pour la méthode de Newton
    return 0.15*( -3.7/4*x**(-3/2) - 1.2*3*x**2 )

# Graphe de la fonction
xx = np.linspace(0.1,0.8,100)
ff = f(xx)
plt.plot(xx,ff,'b-')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.show()

# Fonctions de résolution pré-compilées SCIPY
import scipy.optimize as opt
# Dichotomie
tic = time.time()
resultat = opt.bisect(f,0.1,0.5,full_output=True)
print('\n *** Dichotomie Scipy :\n',resultat[1], \
    time.time()-tic)
# Newton
tic = time.time()
resultat = opt.newton(f,0.1,fprime=fp,full_output=True)
print('\n *** Newton Scipy :\n',resultat[1], \
    time.time()-tic)
# Fonction Solve générale
tic = time.time()
resultat = opt.fsolve(f,0.1)
print('\n *** fsolve Scipy : x=',resultat,' ; duree=', \
    time.time()-tic)

# Dichotomie "MAISON"
def dichotomie(f,a,b,eps = 1e-10):
    '''Calcule la racine de f(x) = 0 sur [a,b] par dichotomie
    avec une precision de l'ordre de eps.'''
```

```
    n = 0 # indice de nombre d'itérations
    c = (a+b)/2
    while (abs(a-b) > eps and n < 500): # critère de sortie double
        c = (a+b)/2 # milieu de l'intervalle courant
        if f(c)*f(a) < 0 : # signes différent
            b = c
        else:
            a = c
        n += 1
    return (a+b)/2,abs(a-b),n # racine, erreur, nb itérations

tic = time.time()
racine,erreur,niter = dichotomie(f,0.1,0.5)
duree = time.time()-tic
print('\n *** Dichotomie MAISON: \n \t x=',racine, \
    '\n \t erreur=',erreur,'\n \t niter=',niter, \
    '\n \t duree =',duree)

# Newton "MAISON"
def newton(f,fp,xini,eps = 1e-10):
    '''Calcule la racine de f(x) = 0 sur par Newton en partant
    de xini avec une precision de l'ordre de eps.'''
    n = 0 # indice de nombre d'itérations
    x = xini # initialisation
    erreur = 1
    while(erreur > eps and n < 500): # critère de sortie double
        xo = x # stocker la valeur précédente pour évaluer l'erreur
        x = x - f(x) / fp(x)
        erreur = abs(x-xo)
        n += 1
    return x,erreur,n # racine, erreur, nb itérations

tic = time.time()
racine,erreur,niter = newton(f,fp,0.1)
duree = time.time()-tic
print('\n *** Newton MAISON: \n \t x=',racine, \
    '\n \t erreur=',erreur,'\n \t niter=',niter, \
    '\n \t duree =',duree)
```

L'exécution produit les sorties suivantes en console :

```
*** Dichotomie Scipy :
      converged: True
           flag: 'converged'
function_calls: 40
    iterations: 38
           root: 0.29098150660720423 9.751319885253906e-05

*** Newton Scipy :
      converged: True
           flag: 'converged'
function_calls: 12
    iterations: 6
           root: 0.29098150660731853 0.0005974769592285156

*** fsolve Scipy : x= [0.29098151]  ; durée= 0.0003910064697265625

*** Dichotomie MAISON:
        x= 0.2909815066028386
        erreur= 9.313222415485711e-11
        niter= 32
        durée = 0.00018310546875

*** Newton MAISON:
        x= 0.2909815066073187
        erreur= 1.6653345369377348e-16
        niter= 7
        durée = 6.937980651855469e-05
```

ANNEXE - Pour aller plus loin

D'autres méthodes existent pour résoudre le problème d'équations stationnaires 1D, parfois généralisables à des situations multidimensionnelles.

Méthode du point fixe

DÉFINITION : Une application g est *contractante* si

$$\exists\, 0 < k < 1, \forall (x,y) \in I, |g(x) - g(y)| \leq k|x-y|$$

THÉORÈME : Soit une fonction g **contractante**, elle admet un **unique point fixe attractif**, qui est la limite de la suite $x_{n+1} = g(x_n)$.

THÉORÈME : Soit une fonction f dérivable sur $[a;b]$, possédant une unique racine dans cet intervalle : $f(r) = 0$. Alors on peut toujours trouver une constante C tel que la fonction φ vérifiant $\varphi(x) = x - C\,f(x)$ soit contractante.

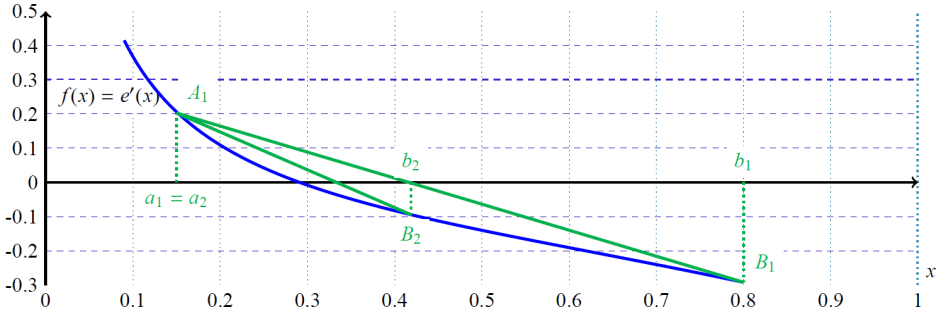
La convergence de cette méthode est **linéaire** donc plutôt **lente**.

Exemple : Déterminer la racine carrée de 2.
On choisi $\varphi(x) = (2-x^2)/2 + x$. Elle est contractante car $\varphi(1) = 3/2$ et $\varphi(2) = 1$ et φ est monotone sur cet intervalle (ou simplement en majorant la dérivée $\varphi'(x) = 1 - x$ autour de r par une valeur strictement inférieure à 1).

Méthode de la corde

Il s'agit d'une variante de la méthode par dichotomie. Au lieu de découper l'intervalle en deux parties égales, on coupe l'intervalle au point d'intersection de la corde, allant du point de coordonnée $(a, f(a))$ au point de coordonnée $(b, f(b))$, avec l'axe des abscisses. L'équation de la corde est donnée par :

$$y = \frac{f(b)-f(a)}{b-a}.(x-a)+f(a) \;\Rightarrow\; 0 = \frac{f(b)-f(a)}{b-a}.(c-a)+f(a) \;\Rightarrow\; c = \frac{a.f(b)-b.f(a)}{f(b)-f(a)}$$



Méthode de la sécante

Il s'agit d'une variante de la méthode de Newton. Lorsqu'on souhaite appliquer cette dernière, on ne connait pas forcément l'expression analytique de f et donc de sa dérivée f'. Il faut donc l'évaluer approximativement. Pour cela, on peut calculer la dérivée numériquement en posant $f(x_n) = \dfrac{f(x_n+h)-f(x_n)}{h}$ avec h petit... Un autre solution consiste à estimer la pente de la tangente en x_n en prenant la pente de la droite passant par $(x_{n-1}, f(x_{n-1}))$ et $(x_n, f(x_n))$. Ainsi $f'(x_n) = \dfrac{f(x_n)-f(x_{n-1})}{x_n - x_{n-1}}$. On en déduit que

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n)-f(x_{n-1})}(x_n - x_{n-1})\,.$$

Il faut évidemment initialiser l'algorithme avec deux valeurs bien choisies proche de la solution.