TP n° 09 – Dichotomie-Newton

L'objectif de ce TD est de résoudre numériquement des équations du type $f(x) = 0$ avec $f : \mathbb{R} \to \mathbb{R}$ supposée continue. On utilisera pour ça deux algorithmes : la méthode par dichotomie et la méthode de Newton.

I Dichotomie

Lors du TP n°7, vous avez écrit une fonction `dicho` qui prend comme entrée la fonction à étudier, les bornes initiales a et b, la précision ϵ et qui renvoie $\dfrac{a_n + b_n}{2}$, approximation d'une solution à ϵ près. Retrouvez cette fonction et sauvegardez-la dans votre nouveau programme du TP n°09.

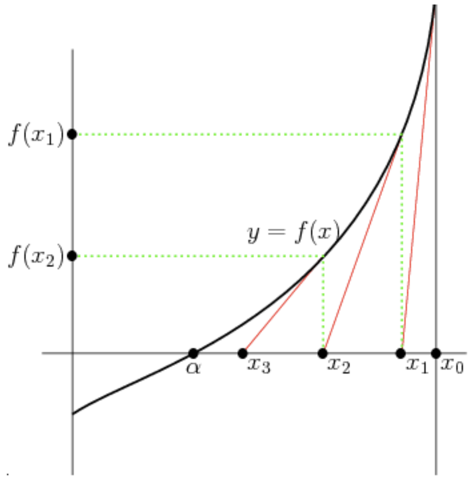
II Méthode de Newton

II.1 Principe de la méthode

La méthode de Newton est un algorithme qui permet d'obtenir une approximation d'une solution α de l'équation $f(x) = 0$.
Partant d'un point x_0, on considère la tangente à la courbe en x_0 :

$$T_{x_0} : y = f(x_0) + f'(x_0)(x - x_0)$$

Elle intersecte l'axe des abscisses en x_1 : $0 = f(x_0) + f'(x_0)(x_1 - x_0) \Leftrightarrow x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$.
Sous certaines conditions, x_1 peut être une meilleure approximation de α que x_0.



On construit ainsi de manière récurrente la suite $(x_n)_{n\in\mathbb{N}}$ d'approximations de α :

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

Avantage : quitte à ne pas être trop éloigné de la solution et avec des hypothèses raisonnables sur f, la convergence est très rapide. On dit qu'elle est quadratique.
Inconvénients :

1. la méthode ne converge pas nécessairement. Notamment, si f' s'annule ou devient très petit, on peut avoir des problèmes avec la division par $f'(x_n)$.
   Elle peut aussi "boucler" de manière infinie.
2. On a plusieurs choix pour le test d'arrêt de l'algorithme :
   (a) on se fixe un nombre d'itérations de la boucle ;
   (b) à $\epsilon > 0$ fixé, le critère d'arrêt est : $|x_{n+1} - x_n| \leqslant \epsilon$.
   Le test peut être validé sans pour autant que x_n soit une approximation de α à ϵ près, mais en général, cela fonctionne bien.

(c) $|f(x_n)| \leqslant \epsilon$. Là encore, f peut être petite en x_n mais ne pas s'annuler près de x_n.

Comme il y a un risque que la méthode de Newton ne converge pas, on mettra toujours dans le test d'arrêt un nombre d'itérations maximal.

3. Il faut connaitre f'.

II.2 Applications

Exercice 1. 1. Ecrire une fonction `Newton1` qui prend comme entrée la fonction à étudier, sa dérivée, une valeur initiale de la suite x_0 et un nombre d'itérations n et qui renvoie x_n approximation d'une solution.

2. Testez cette fonction sur $f\colon x \mapsto x^2 - 2$ avec différentes valeurs de x_0 et de n.
3. Que se passe-t-il si vous prenez $x_0 = 0$? Pourquoi ?
4. La suite (x_n) peut converger vers $\sqrt{2}$ ou $-\sqrt{2}$. Au bout de combien d'itérations a-t-on une valeur approchée de $\sqrt{2}$ à 10^{-10} près en partant de $x_0 = 2$?

Exercice 2. Appliquer `Newton1` à la fonction $x \mapsto x^3 - 2x + 2$ en démarrant à $x_0 = 0$. Que se passe-t-il ? Quand vous avez une réponse, reportez-vous au graphique en fin d'énoncé pour visualiser la suite (x_n). Que se passe-t-il si on démarre avec un x_0 assez proche de 0 ou 1 ?

Exercice 3. Modifier la fonction `Newton1` en une fonction `Newton2` pour que le critière d'arrêt soit $|x_{n+1} - x_n| \leqslant \epsilon$.
La fonction `Newton2` prendra comme entrée : la fonction à étudier f, sa dérivée f', une valeur initiale de la suite x_0 et la marge d'erreur ϵ. Pour éviter les boucles infinies, on fera en sorte que dans cette fonction, le nombre d'itérations ne dépasse pas 1000.

Exercice 4. Testez la fonction `Newton2` sur $x \mapsto \sin(x)$ pour déterminer une approximation de π. Comparer la valeur de $x_n - \pi$ avec la valeur ϵ choisie. Qu'en pensez-vous ?

Exercice 5. Comparaison de la vitesse des deux méthodes.
On veut comparer la vitesse des deux algorithmes, dichotomie et méthode de Newton sur la fonction $f(x) = x^2 - 2$, avec une précision $\epsilon = 10^{-10}$.
Modifier les fonctions `dicho` et `Newton2` pour savoir laquelle est la plus rapide. Pour cela, on pourra comparer le nombre d'itérations effectuées dans chaque programme ou le temps mis par chaque algorithme en utilisant la méthode `time()` de la bibliothèque `time`.

Exercice 6. Méthode dans le cas où on ne connait pas f'.
Il se peut qu'on ne connaisse pas la dérivée de f. Dans ce cas, on peut approximer $f'(x_n)$ par $\dfrac{f(x_n + h) - f(x_n)}{h}$ pour un h fixé petit.
Modifier la fonction `Newton2` en une fonction `Newton3` qui a comme entrée la fonction f, x_0, ϵ, h et qui renvoie x_n.

Exercice 7. Méthode de la sécante
Avec cette méthode, on ne connait pas f'. On approxime $f'(x_n)$ par le coefficient directeur de la corde passant par $(x_n, f(x_n))$ et $(x_{n-1}, f(x_{n-1}))$: $f'(x_n) \approx \dfrac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$
La relation de récurrence de la suite (x_n) est donnée par :

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

On a besoin de deux points x_0 et x_1 pour initialiser la suite (x_n).
Ecrire une fonction `secante` qui prend comme entrée f, x_0, x_1 et ϵ renvoie x_n.

Exercice 8. Méthode de Newton en complexe.

1. Quelle sont les racines de la fonction $f(x) = x^2 + 1$? Testez la fonction `Newton1` avec f pour x_0 réel puis un x_0 complexe.
2. Les solutions complexes de l'équation $z^3 = 1$ sont : 1 ; $-\dfrac{1}{2} + i\dfrac{\sqrt{3}}{2}$; $-\dfrac{1}{2} - i\dfrac{\sqrt{3}}{2}$. A l'aide de la fonction `Newton1`, retrouvez ces valeurs en choisissant un bon x_0.

III Annexe

III.1 Vitesse de convergence

Dichotomie : chaque étape de l'algorithme, on divise la longueur de l'intervalle par 2. Si on note α le zéro de f, on a :

$$|c_{n+1} - \alpha| \leqslant \frac{1}{2}|c_n - \alpha|$$

On dit que la convergence est linéaire.

Méthode de Newton : avec des hypothèses raisonnables sur f (par exemple, $f'(\alpha) > 0$ et $f'' \geqslant 0$), alors il existe une constante C telle que à partir d'un certain rang,

$$|x_{n+1} - \alpha| \leqslant C|x_n - \alpha|^2$$

On dit que la vitesse de convergence est quadratique.

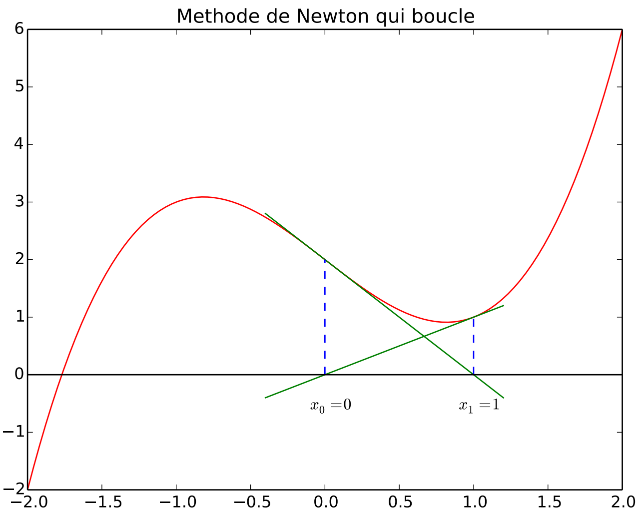
Exemple de comparaison des deux algorithmes : Si on intialise les deux algorithmes en x_0 à une distance $\frac{1}{2}$ de α :

$$|c_n - \alpha| \leqslant \left(\frac{1}{2}\right)^{n+1}$$

$$|x_n - \alpha| \leqslant \left(\frac{1}{2}\right)^{2^n}$$

La méthode de Newton est beaucoup plus rapide que la dichotomie.

III.2 Illustration de l'exercice 2



La suite (x_n) de l'exercice 2

Correction TP n° 09 – Dichotomie-Newton

Solution 1.

1.
```
def Newton1(f,fprime,x0,nb_iteration):
    x=float(x0)
    for i in range(nb_iteration):
        x=x-f(x)/fprime(x)
    return(x)
```

2.
```
def f(x):
    return(x**2-2)
def fprime(x):
    return(2*x)
```

3. $f'(0) = 0$: Python renvoie un message d'erreur car il ne peut pas diviser par $0 = f'(0)$.
4. Newton1$(f, f', 2, 3) - \sqrt{2} = O(10^{-6})$ et Newton1$(f, f', 2, 4) - \sqrt{2} = O(10^{-12})$.
   Donc pour $n = 4$, on a une approximation à 10^{-10} près.

Solution 2. La suite "boucle", elle vaut : $0, 1, 0, 1, 0, 1, \cdots$
Les points $x = 0$ et $x = 1$ sont "supers attractifs" : si on initialise la suite assez près de ces valeurs, les termes de la suite vont converger vers 0 et 1. Pour Python, la suite va finir par boucler $0, 1, 0, \cdots$

Solution 3.

```
def Newton2(f,fprime,x0,epsilon):
    x0=float(x0)
    x1=x0-f(x0)/fprime(x0)
    compteur=0     # on compte le nb d iterations de la boucle
    # test d arret : on itere la boucle si l ecart entre xn et x(n+1) est grand
    # et si on a fait moins de 1000 iterations
    while abs(x1-x0)>epsilon and compteur<1000 :
        x0=x1
        x1=x1-f(x1)/fprime(x1)
        compteur=compteur+1
    return(x1)
```

Méthode 2 :

```
def Newton2Bis(f,fprime,x0,epsilon):
    x=float(x0)
    compteur=0
    # l ecart entre xn et x(n+1) est f(xn)/fprime(xn)
    while abs(f(x)/fprime(x))>epsilon and compteur<1000 :
        x=x-f(x)/fprime(x)
        compteur=compteur+1
    return(u)
```

Solution 4. On teste : `Newton2(sin,cos,2,1e-3)-pi`. On trouve 9.10^{-11}.
La différence est bien inférieure à ϵ, elle est même très inférieure, car l'algorithme est très rapide.

Solution 5. On compare le nombre d'itérations et le temps mis par les deux algorithmes.

```
import time

def dicho_comparaison(f,a,b,eps):
    t=time.time()            # donne le temps de calcul
    compteur=0           # va compter le nombre d'iterations
    if f(a)*f(b)>0:
```

```
        return('nous ne savons pas si f s annule entre a et b')
    while (b-a)>2*eps:
        compteur=compteur+1
        c=(float(a)+b)/2
        if f(a)*f(c)<0:
            b=c
        else:
            a=c
    return((a+b)/2,time.time()-t,compteur)

def Newton_comparaison(f,fprime,u0,epsilon):
    t=time.time()              # donne le temps de calcul
    compteur=0            # va compter le nombre d'iterations
    u0=float(u0)
    u1=u0-f(u0)/fprime(u0)
    while abs(u1-u0)>epsilon and compteur<1000 :
        u0=u1
        u1=u1-f(u1)/fprime(u1)
        compteur=compteur+1
    return(u1,time.time()-t,compteur)
```

On teste ensuite sur la même fonction, avec la même marge d'erreur :

```
Newton_comparaison(carre,carre_prime,2,1e-15)
dicho_comparaison(carre,0,2,1e-15)
```

Solution 6.

```
def Newton3(f,h,x0,epsilon):
    x0=float(x0)
    derivee=(f(x0+h)-f(x0))/h
    x1=x0-f(x0)/derivee
    compteur=0
    while abs(x1-x0)>epsilon and compteur<1000 :
        x0=x1
        derivee=(f(x1+h)-f(x1))/h
        x1=x1-f(x1)/derivee
        compteur=compteur+1
    return(x1)
```

Solution 7. Attention, il se peut que lorsque x_{n+1} et x_n sont trop proches, Python renvoie un message d'erreur car $f(x_n) - f(x_{n-1}) \approx 0$.

```
def secante(f,x0,x1,eps):
    x0=float(x0)
    x1=float(x1)
    compteur=0
    while abs(x0-x1)>eps and compteur<1000:
        temp=x1
        x1=x1-f(x1)*(x1-x0)/(f(x1)-f(x0))
        x0=temp
        compteur=compteur+1
    return(x1)
```

Solution 8. Avec un x_0 réel, l'algorithme ne converge pas. Avec un x_0 complexe, l'algorithme renvoie `1j`.