ANALYSE NUMERIQUE

Résolution d'équations non linéaires (logiciel Python)

TABLE DES MATIÈRES

Vous pouvez **télécharger** ce document au format PDF à l'adresse suivante :

https://www.sismondi.ch/disciplines/applications-des-mathematiques/cours-eleves

1 Résolution d'équations non linéaires

1.1 Introduction

Certains problèmes scientifiques conduisent à la résolution d'équations polynomiales de degré plus grand ou égal à 5 ou à des équations contenant des fonctions trigonométriques, logarithmiques ou exponentielles pour lesquelles il n'existe pas de résolution simple voir pas de résolution analytique.

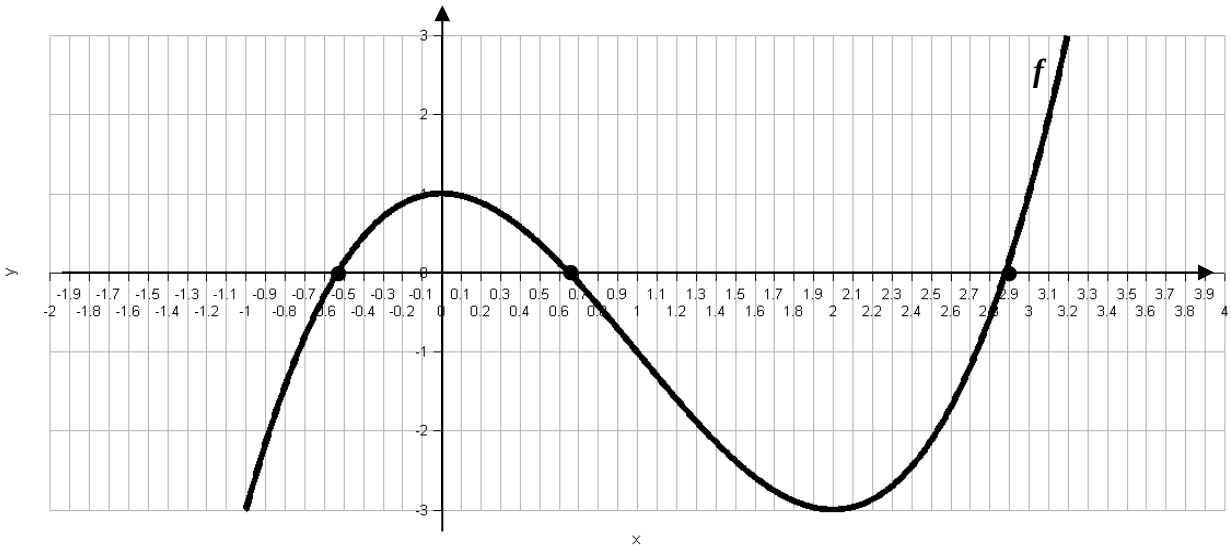
Les équations suivantes illustre ce propos :

a) $3x^{13} - 2x^4 + 12x^3 - 10 = 0$ **b)** $\sqrt{x} = ln(x) + sin(x)$ **c)** $e^x = 3x - 2$

L'idée que nous allons exploiter dans ce chapitre pour résoudre des équations est le fait que :
« La résolution d'une équation peut se ramener à rechercher les zéros d'une fonction. »

Exemple

Chercher toutes les solutions réelles de l'équation polynomiale $x^3 - 3x^2 + 1 = 0$ est équivalent à chercher les zéros de la fonction $f(x) = x^3 - 3x^2 + 1$ c'est-à-dire les valeurs de *x* tel que $f(x) = 0$. Graphiquement :



On observe graphiquement que *f* admet trois zéros tous situés dans l'intervalle $[-1\,;3]$.

Nous allons étudier dans ce cours ***trois méthodes*** numériques permettant par approximations successives de déterminer les zéros avec la *précision demandée*. Ces méthodes nécessitent de connaître pour chaque zéro un intervalle $[a;b]$ qui ne contienne pas d'autre zéro que celui recherché. En général quelques calculs d'images et une esquisse du graphique de la fonction permettent de localiser les zéros.

Les trois méthodes sont :
1) La méthode de la **bissection**.
2) La méthode de **Newton**.
3) La méthode du **point fixe**.

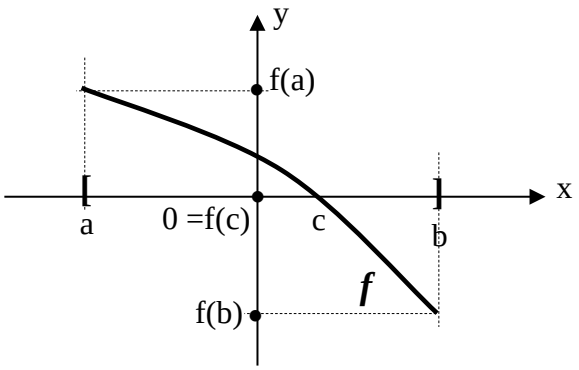
1.2 Méthode de la bissection

Théorème de Bolzano

Si	f est *continue* sur $[a;b]$ et si *f(a)* et *f(b)* sont de signes contraires c-à-d $f(a)\cdot f(b) \leq 0$,
alors	il existe au moins un nombre réel $c \in [a;b]$ tel que $f(c)=0$.

La démonstration de ce théorème, ne sera pas exposée dans ce cours.

Illustration : f est continue sur $[a;b]$ avec $f(a)\cdot f(b) \leq 0$

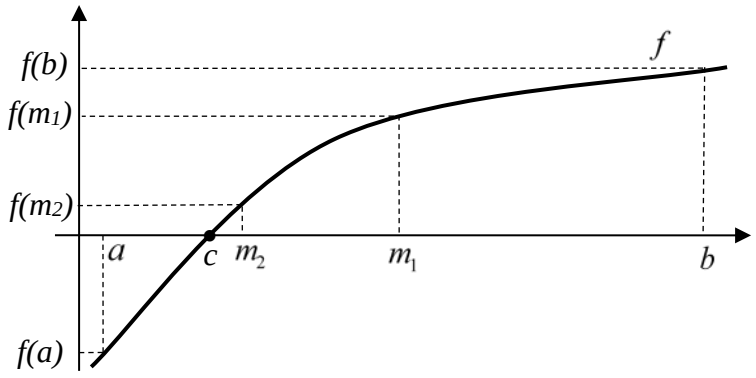


Remarque : Le *théorème de Bolzano* ne donne pas la méthode pour trouver c tel que $f(c)=0$.

Description de la méthode de la bissection

Considérons une fonction f satisfaisant les hypothèses du théorème de *Bolzano* sur $[a;b]$.
On veut déterminer le nombre *c*, zéro de f sur cet intervalle.

Illustration



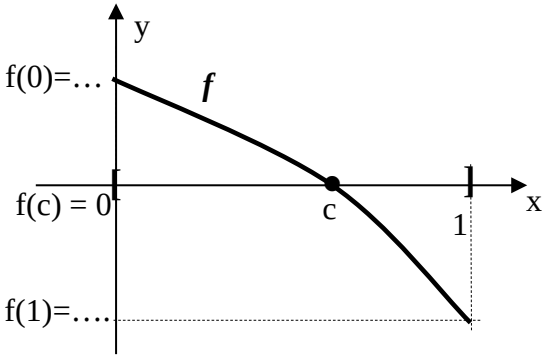
1) On divise l'intervalle $[a;b]$ en deux parties égales et on note son milieu par $m_1 = \dfrac{a+b}{2}$.

2) **Si** $f(a)\cdot f(m_1) \leq 0$ **alors** le zéro se trouve dans cet intervalle et on continue la méthode sur l'intervalle $[a;m_1]$ en prenant $m_2 = \dfrac{a+m_1}{2}$ le milieu de $[a;m_1]$.

3) **Sinon**, on a nécessairement que $f(m_1)\cdot f(b) \leq 0$ et on poursuit avec l'intervalle $[m_1;b]$.

4) On poursuit les calculs tant que la longueur de l'intervalle est supérieure à une *tolérance positive donnée (précision).*

Activité 01



- Montrons que l'équation $x^3 - 3x^2 + 1 = 0$ possède une solution dans l'intervalle $[0\ ;1]$.

  Posons $f(x) =$.....

  On a $f(0) \cdot f(1)$ …..…..…..et f est une fonction …….. sur $\mathbb{R}$ (car c'est une fonction polynomiale).

  Donc, d'après le *théorème de Bolzano,* il existe un nombre $c \in$ [….. ; .....] tel que $f(c) =$.....

- Déterminons une ***approximation*** du nombre ***c*** à l'aide de la ***méthode de la bissection*** avec une précision de *Tol = 0,3*.

$a_0 = 0 < c < 1 = b_0$

$b_0 - a_0 = \text{.....} = Tol$

$m_1 = \dfrac{a_0 + b_0}{2} = \dfrac{\quad}{2} = \text{.....}$

$f(a_0) \cdot f(m_1) = f(\text{.....}) \cdot f(\text{.....}) = \text{.....}$

Itération 1 :

$a_1 = \text{.....} < c < \text{.....} = b_1$

$b_1 - a_1 = \text{.....} = Tol$

$m_2 = \dfrac{a_1 + b_1}{2} = \dfrac{\quad}{2} = \text{.....}$

$f(a_1) \cdot f(m_2) = f(\text{.....}) \cdot f(\text{.....}) = \text{.....}$

Itération 2 :

$a_2 = \text{.....} < c < \text{.....} = b_2$

$b_2 - a_2 = \text{.....} = Tol$

- ***Un zéro*** de $f(x) = x^3 - 3x^2 + 1$ se trouve entre $a_2 =$ ..... *et* $b_2 =$ .....

- ***L'erreur maximale*** commise est de $b_2 - a_2 =$ ..... si l'on choisit $a_2 =$ ..... *ou* $b_2 =$ ..... comme approximation d'un zéro de *f*.

- Dans ce cas, il a suffit de …… ***itérations*** pour obtenir la précision (tolérance) désirée.

Remarques

a) La méthode de la bissection consiste à construire ***une suite d'intervalles emboîtés*** $[a_n ; b_n]$ avec $n \in \mathbb{N}$ qui contiennent un zéro de *f*. Les suites de nombre $\{a_n\}_{n\in\mathbb{N}}$ *et* $\{b_n\}_{n\in\mathbb{N}}$ obtenues avec la méthode de la bissection ***convergent*** vers la solution si on choisit $a_0 = 0$ et $b_0 = 1$.

b) $b_n - a_n = \dfrac{1}{2^n}(b_0 - a_0) \qquad \forall\, n \in \mathbb{N}$

Si ***c*** désigne le zéro de la fonction commun à tous les intervalles $[a_n ; b_n]$, on a :

$|a_n - c| \le \dfrac{1}{2^n}(b_0 - a_0)$ *et* $|b_n - c| \le \dfrac{1}{2^n}(b_0 - a_0) \qquad \forall\, n \in \mathbb{N}$

c) En Analyse numérique, une ***méthode itérative*** est une méthode qui résout un problème (comme une équation ou un système d'équations) en trouvant une succession d'approximations en commençant par une valeur initiale. La méthode de la bissection en est une.

Programmation de la méthode de la bissection avec Python 3

La ***méthode de la bissection*** comme vous avez pu le constater lors de l'activité précédente est répétitive. Elle nécessite, pour obtenir l'approximation souhaitée, un nombre important d'opérations qui dépend de la fonction f et de la précision demandée.

L'idée est alors de déterminer *un algorithme* et ensuite d'écrire *un programme* qui permet à un ordinateur (par le biais d'un logiciel ; dans notre cas ce sera *Python 3*) d'exécuter répétitivement et rapidement cette méthode.

On peut proposer le **programme Python 3** ci-dessous utilisant ***la méthode de la bissection*** pour déterminer un encadrement d'un zéro de $f(x)=x^3-3x^2+1$ sur l'intervalle $[a;b]$ en fixant une précision de Tol .

```
# Fonction f

def f(x):
    return x**3-3*x**2+1

# Méthode de la bissection

def BISSECTION(y,a,b,Tol):
    while b-a>Tol:
        m=(a+b)/2
        if y(a)*y(m)<=0:
            b=m
        else:
            a=m
    return a,b

# Programme principal

a=eval(input("Entrez la borne inférieure de l'intervalle : a="))
b=eval(input("Entrez la borne supérieure de l'intervalle : b="))
Tol=eval(input("Entrez la précision : Tol="))

while f(a)*f(b)>0 or Tol<0:
    a=eval(input("Entrez la borne inférieure de l'intervalle : a="))
    b=eval(input("Entrez la borne supérieure de l'intervalle : b="))
    Tol=eval(input("Entrez la précision : Tol="))

a,b = BISSECTION(f,a,b,Tol)

print("Un zéro de f se trouve dans l'intervalle [",a,";",b,"]")
```

Activité 02

Compléter le tableau ci-dessous afin de tester le programme se trouvant à la page précédente avec :

- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad a = 0 \quad ; \quad b = 1 \quad ; \quad Tol = 0{,}3$

	a	b	Tol	m	b-a	f(a)·f(m)	Booléen
Entrées	**0**	**1**	**0.3**				
Boucle 1							
Boucle 2							
Boucle 3							
Sortie							

Exercice 03

a) Ecrire dans un éditeur et comprendre ***le programme en Python 3*** qui donne un encadrement d'un zéro de f à l'aide de **la méthode de la bissection .**

Tester le avec :

- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad a = 0 \quad ; \quad b = 1 \quad ; \quad Tol = 0{,}3$
- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad a = 0 \quad ; \quad b = 1 \quad ; \quad Tol = 10^{-5}$
- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad a = 0 \quad ; \quad b = 4 \quad ; \quad Tol = 10^{-5}$

b) Que constate-t-on ?

Nom du fichier : ex_03_bissection_cours.py

Exercice 04

a) Modifier le programme « **ex_03_bissection_cours.py »** pour qu'il ***indique / affiche*** en plus :

1) les résultats de la méthode de la bissection à chaque itération.

Autrement dit : $i \rightarrow [a_i ; b_i]$

$i+1 \rightarrow [a_{i+1} ; b_{i+1}]$

…….

2) La *précision* désirée et le *nombre d'itérations* nécessaires pour obtenir cette précision.

3) *L'erreur maximale* commise.

b) Tester votre programme avec :

- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad a = 0 \quad ; \quad b = 1 \quad ; \quad Tol = 0{,}3$
- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad a = 0 \quad ; \quad b = 1 \quad ; \quad Tol = 10^{-5}$

Exemple : Sortie Python 3 (console)

```
Entrez la borne inférieure de l'intervalle : a=0
Entrez la borne supérieure de l'intervalle : b=1
Entrez la précision : Tol=0.3
>>>
i    [a ; b]
0    [ 0 ; 1 ]
1    [ 0.5 ; 1 ]
2    [ 0.5 ; 0.75 ]
Un zéro de f se trouve dans l'intervalle [ 0.5 ; 0.75 ] .
Pour atteindre la précision de 0.3 le nombre d'itérations nécessaire doit être au minimum de  2 .
L'erreur maximale est de  0.25 .
```

Nom du fichier : ex_04_bissection_affichage.py

Exercice 05

Pour chacune des fonctions suivantes : **a)** $f(x) = x^3 - 10$ **b)** $f(x) = x - 1 + \frac{1}{2}sin(x)$

1) Tracer à ***l'aide de Python*** le graphique de la fonction, puis choisir un intervalle satisfaisant les hypothèses du théorème de Bolzano.

2) Déterminer à ***l'aide de Python*** et en utilisant la ***méthode de la bissection***, un encadrement de(s) zéro(s) de chaque fonction avec une précision de $Tol = 10^{-3}$.

Remarque : L'unique zéro réel de la fonction $f(x) = x^3 - 10$ est la racine cubique de 10

c'est-à-dire : $x = \sqrt[3]{10}$

Nom du fichier : ex_05_bissection.py

Exercice 06

Un stère de bois (cube d'un mètre de côté) est entassé contre une grande maison.

A quelle distance x (au centimètre près) de la maison faut-il poser le pied d'une échelle de *10 m* pour qu'elle s'appuie tant contre la façade que contre le coin de la stère ?

Indication :

Montrer que le problème revient à résoudre l'équation $\sqrt{100-x^2}\cdot(x-1)-x=0$ puis la résoudre à ***l'aide de Python*** et de la ***méthode de la bissection***.

Remarque : Il y deux solutions.

Nom du fichier : ex_06_bissection.py

Exercice 07

La ***méthode de la bissection*** a l'avantage de fournir un encadrement d'une solution *c* de l'équation $f(x)=0$. Il est donc facile d'avoir ***une majoration de l'erreur***. En effet, à chaque étape, la taille de l'intervalle contenant *c* est divisée par *2*.

Au départ, on sait que $c\in[a_0;b_0]$ qui est de longueur : b_0-a_0

Puis, $c\in[a_1;b_1]$ qui est de longueur : $b_1-a_1=\dfrac{b_0-a_0}{2}$

….

A l'itération $n\in\mathbb{N}$, $c\in[a_n;b_n]$ qui est de longueur : $b_n-a_n=\dfrac{b_0-a_0}{2^n}$

Si on souhaite obtenir une approximation de *c* à 10^{-N} (= Tol) près, comme on sait que $c\in[a_n;b_n]$, on obtient : $|c-a_n|\le|b_n-a_n|=\dfrac{b_0-a_0}{2^n}$ ou $|c-b_n|\le|b_n-a_n|=\dfrac{b_0-a_0}{2^n}$

Il suffit donc de choisir un nombre entier *n* tel que : $\dfrac{b_0-a_0}{2^n}\le 10^{-N}$

Sans utiliser de logiciels :

a) Résoudre l'inéquation : $\dfrac{b_0-a_0}{2^n}\le 10^{-N}$ (inconnue : n)

b) On veut calculer une approximation d'un zéro de $f(x)=x^3-3x^2+1$ sur l'intervalle $[0;1]$ à l'aide de la méthode de la bissection.

b.1) Déterminer, sans logiciel, *le nombre minimum d'itérations* nécessaires pour avoir une précision de 10^{-5} si $a_0=0$ *et* $b_0=1$.
Ce calcul dépend-t-il de la fonction *f* dont on cherche les zéros ?

b.2) On veut *10* chiffres exacts après la virgule. Combien faut-il d'itération au minimum ?

Exercice 08

Voici un ***programme Python 3*** contenant la fonction MYSTERE. 

a) Etudiez-le, programmez-le et testez-le.

b) Quel est le rôle de la fonction MYSTERE ?

c) Quel est la particularité de ce programme ?

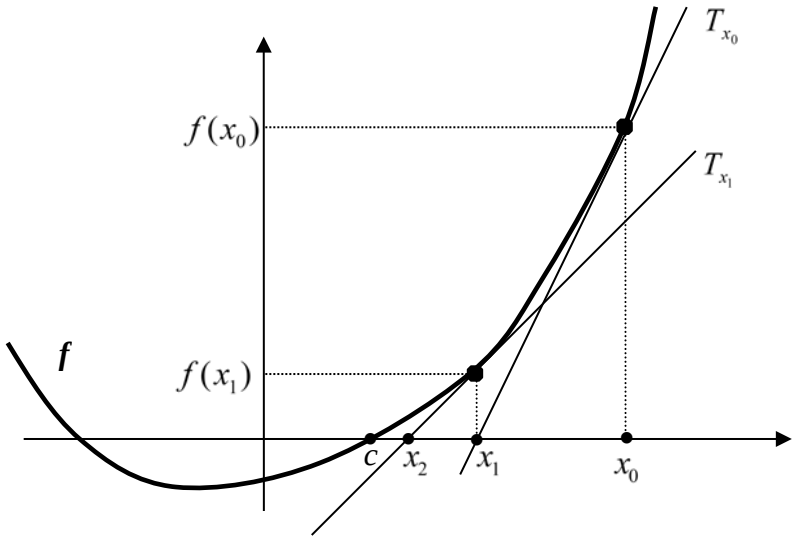
```
def f(x):
    return x**3-3*x**2+1

def MYSTERE(y,a,b,Tol):
    if b-a<Tol:
        return a,b
    else:
        m=(a+b)/2
        if y(a)*y(m)<=0:
            return MYSTERE(y,a,m,Tol)
        else:
            return MYSTERE(y,m,b,Tol)
    return a,b
```

Nom du fichier : ex_08_fonction_mystere.py

1.3 Méthode de Newton

Idée Approcher le graphique de la fonction f, dont on cherche à déterminer un zéro, par une droite tangente pour laquelle il sera simple de déterminer un zéro.

Illustration



Description de la méthode de Newton

- Par définition, si $\boldsymbol{c}$ est un zéro de f, alors $\boldsymbol{f(c) = 0}$.
  Géométriquement, c'est en $\boldsymbol{c}$ que le graphique de f coupe l'axe des x.

- Premièrement, on choisit un nombre $\boldsymbol{x_0}$ proche de $\boldsymbol{c}$. La tangente $\boldsymbol{T_{x0}}$ à f en $\boldsymbol{(x_0 ; f(x_0))}$ coupe l'axe des abscisses en un point $\boldsymbol{x_1}$ normalement plus proche de $\boldsymbol{c}$ que $\boldsymbol{x_0}$ et en ce sens, meilleure approximation de $\boldsymbol{c}$.

- La pente de $\boldsymbol{T_{x0}}$ est donnée par $f'(x_0)$ et son équation par : $\boldsymbol{y = f'(x_0)(x - x_0) + f(x_0)}$.
  Le point d'intersection de $\boldsymbol{T_{x0}}$ avec l'axe des x se produit là où y = 0, c'est-à-dire en $\boldsymbol{x_1}$ tel que $0 = f'(x_0)(\mathbf{x_1} - x_0) + f(x_0)$.

  Si $f'(x_0) \neq 0$, l'équation précédente est équivalente à $\boxed{x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}}$.

- En se servant de $\boldsymbol{x_1}$ comme on l'a fait de $\boldsymbol{x_0}$, à savoir chercher le point d'intersection avec l'axe des x de la tangente en ($\boldsymbol{x_1}$, f($\boldsymbol{x_1}$)), on obtient, si $f'(x_1) \neq 0$, $\boxed{x_2 = x_1 - \dfrac{f(x_1)}{f'(x_1)}}$.

- On construit ainsi, ***une suite*** (de Newton) $x_0 ; x_1 ; x_2 ; x_3 ; x_4 ....$ qui devrait ***converger*** vers le zéro de f cherché jusqu'à atteindre la précision souhaitée.

Conclusion

Soit f une fonction dérivable et soit $\boldsymbol{c}$ un zéro réel de f.

Si $\boldsymbol{x_n}$ est une approximation de $\boldsymbol{c}$,

alors l'approximation suivante $\boldsymbol{x_{n+1}}$ est donnée par $\boxed{\boldsymbol{x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}}}$

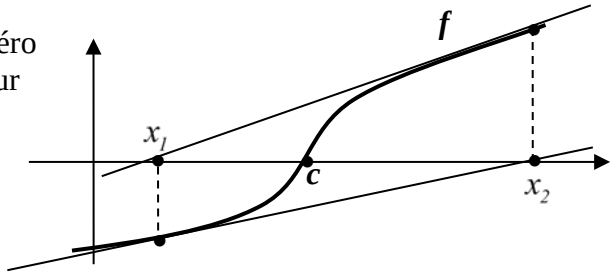
à condition que $f'(x_n) \neq 0$.

Remarques

a) le calcul n'est possible que si la dérivée de f ne s'annule pas.

b) La convergence d'une suite de Newton vers un zéro de f n'est pas automatique. Voici un exemple pour laquelle la méthode ne fonctionne pas.

c) Il existe un théorème sur la convergence de la méthode de Newton. Nous l'étudierons ultérieurement.



d) On peut montrer que l'erreur $|x_n - c|$ peut être correctement estimée par ***la différence entre deux termes successifs de la suite de Newton en valeur absolue***. Il est donc raisonnable de poursuivre les calculs aussi longtemps que $|x_{n+1} - x_n|$ est supérieur à *une tolérance positive donnée*.

e) La méthode de Newton est en général plus « rapide » que celle de la bissection ; elle nécessite souvent moins d'itérations.

Programmation de la méthode de Newton avec Python 3

On peut proposer **le programme Python 3** ci-dessous utilisant ***la méthode de Newton*** pour déterminer plusieurs termes d'une suite de Newton qui converge peut-être vers un zéro de $f(x) = x^3 - 3x^2 + 1$ en fixant comme premier terme de la suite x_0 et une précision de Tol.

On calcule « à la main » la dérivée de f qui est $f'(x) = 3x^2 - 6x$.

```
# Fonctions f et f '

def f(x):
    return x**3-3*x**2+1

def df(x):
    return 3*x**2-6*x

# Methode de Newton

def NEWTON(y,dy,x0,Tol):
    x1=x0-y(x0)/dy(x0)
    while abs(x1-x0)>Tol:
        x0=x1
        x1=x0-y(x0)/dy(x0)
    return x1

# Programme « principal »

x0=eval(input("Entrez l'approximation initiale d'un zéro x0= "))
Tol=eval(input("Entrez la précision : Tol="))

x1 = NEWTON(f,df,x0,Tol)

print("Une approximation d'un zéro de f est ",x1)
```

Activité 09

Compléter le tableau ci-dessous afin de tester le programme se trouvant à la page précédente avec :

- $f(x) = x^3 - 3x^2 + 1$; $x_0 = 0.5$; $Tol = 0,1$

	x0	x1	Tol	$\lvert x1 - x0 \rvert$	Booléen
Entrées	**0.5**		**0.1**		
Boucle 1					
Boucle 2					
Sortie					

Remarque : $f(.....) \cong .....$

Exercice 10

a) Ecrire dans un éditeur et comprendre ***le programme en Python 3*** qui calcule les termes d'une suite obtenue à l'aide de **la méthode de Newton.** Cette suite converge peut-être vers un zéro de f.

Tester le avec :
- $f(x) = x^3 - 3x^2 + 1$; $x_0 = 0.5$; $Tol = 0,1$
- $f(x) = x^3 - 3x^2 + 1$; $x_0 = 0.5$; $Tol = 10^{-5}$

b) Que constate-t-on ?

Nom du fichier : ex 10_newton_cours.py

Exercice 11

a) Modifier le programme **« ex 10_newton_cours.py »** pour qu'il ***indique / affiche*** en plus :

1) les résultats de la méthode de Newton à chaque itération.

Autrement dit : $i \rightarrow x_i$

$i+1 \rightarrow x_{i+1}$

…….

2) La *précision* désirée et le *nombre d'itérations* nécessaires pour obtenir cette précision.

b) Tester votre programme avec :

- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad x_0 = 0.5 \quad ; \quad Tol = 0,1$
- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad x_0 = 0.5 \quad ; \quad Tol = 10^{-5}$

Exemple : Sortie Python 3 (console)

```
Entrez l'approximation initiale d'un zéro x0= 0.5
Entrez la précision : Tol=0.1
>>>
i    xi
0    0.5
1    0.6666666666666666
2    0.6527777777777778
Une approximation d'un zéro de f est  0.6527777777777778
Pour atteindre la précision de  0.1 le nombre d'itérations nécessaire doit être au minimum de  2 .
```

Nom du fichier : ex 11_newton_affichage.py

Exercice 12

La pression de la vapeur P d'un matériau s'exprime en fonction de la température T et des constantes A,B,C propres à chaque matériau par la relation : $ln(P) = A + \frac{B}{T} + C \cdot ln(T)$.

À l'aide à ***l'aide de Python*** et de ***la méthode de Newton*** déterminer à quelle température (au centième de degré) correspond, pour le plomb, la pression de vapeur de *0,01* atmosphère sachant, que pour le plomb les constantes sont : $A = 18,19$, $B = -23180$ *et* $C = -0,8858$.

Indication :

En Python 3, la fonction logarithme naturel *ln* se note : **log** ; il faut importer le module **math.**

Nom du fichier : ex 12_newton.py

Théorème de convergence pour la méthode de Newton

Soit $f:[a;b]\to\mathbb{R}$ une fonction.

Si

1) f est deux fois continûment dérivable sur $[a;b]$ (f' est dérivable et f'' est continue)

2) $f(a)$ et $f(b)$ sont de signes opposés càd $f(a)\cdot f(b)\le 0$

3) f' et f'' ont un signe constant sur $[a;b]$ $\left(f'(x)\neq 0 \ et \ f''(x)\neq 0 \quad \forall x\in[a;b]\right)$

4) on choisit $x_0\in[a;b]$ tel que $f(x_0)$ et $f''(x_0)$ sont de même signe càd $f(x_0)\cdot f''(x_0)>0$

Alors

la suite de Newton de premier terme x_0 converge vers l'unique solution $\boldsymbol{c}$ de l'équation $f(x) = 0$ dans $[a;b]$.

Démonstration

La démonstration de ce théorème, ne sera pas exposée dans ce cours.

Exemple

Soit la fonction $f:[1;2]\to\mathbb{R}$ définie par $f(x)= x^2-2$.

Cette fonction satisfait les hypothèses du *théorème* sur l'intervalle $[1;2]$ car :

1) $f'(x)=2x$ dérivable sur $[1;2]$ et $f''(x)=2$ continue sur $[1;2]$.
2) $f(1)\cdot f(2)<0$
3) $f'(x)>0 \ \ et \ \ f''(x)>0 \quad \forall x\in[1;2]$
4) On choisit $x_0=2\in[1;2]$ tel que $f(2)\cdot f''(2)>0$

Conclusion : la suite de Newton de premier terme $x_0=2$ converge vers $\sqrt{2}$ l'unique solution de l'équation $x^2-2=0$ dans $[1;2]$.

Lorsque qu'on fait le calcul explicite de x_{n+1} , on s'aperçoit que _

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}=x_n-\frac{x_n^{\,2}-2}{2x_n}=\frac{x_n^{\,2}+2}{2x_n}=\frac{1}{2}\left(x_n+\frac{2}{x_n}\right)$$

Remarque : C'est *l'algorithme des babyloniens* de recherche des racines carrées.

Remarque

Les conditions à remplir sont nombreuses pour obtenir la convergence de la suite de Newton $(x_n)_{n\in\mathbb{N}}$. Cependant, si toutes les conditions ne sont pas remplies, il peut y avoir quand même convergence car : $HYP \Rightarrow CONCL \not\Leftarrow$ *Négation HYP* $\Rightarrow$ *Négation CONCL*

Exercice 13

Soit la fonction f définie par $f(x) = -5x^3 + 7x^2 + 3x - 3$.

a) Tracer à ***l'aide de Python*** le graphique de la fonction f, de sa dérivée première et seconde dans le même repère sur l'intervalle $[-1;2]$.

b) Calculer à ***l'aide de Python*** les *10* premiers termes de ***la suite de Newton*** si l'on choisit :

i) $x_0 = 0$ **ii)** $x_0 = 1$ **iii)** $x_0 = 0.5$

Que remarque-t-on ?

c) Vérifier si la fonction f satisfait aux hypothèses du théorème de convergence sur l'intervalle :

i) $[0;1]$ **ii)** $[0.5;1]$

Nom du fichier : ex 13_newton.py

Exercice 14

Pour un nombre $\boldsymbol{w} \in \mathbb{R}_+$, on considère la fonction $\boldsymbol{f(x) = x^2 - w}$.

Les zéros de cette fonction sont : $f^{-1}(0) = \{-\sqrt{w};\sqrt{w}\}$.

a.1) Montrer, en utilisant le théorème de convergence, que la suite produite par ***la méthode de Newton*** converge vers $\sqrt{w}$ si l'estimation initiale est $x_0 = 1$ et $0 \le w \le 1$.

a.2) Montrer, en utilisant le théorème de convergence, que la suite produite par ***la méthode de Newton*** converge vers $\sqrt{w}$ si l'estimation initiale est $x_0 = w$ et $w > 1$.

b) Ecrire ***un programme en Python*** qui calcule à l'aide de la ***méthode de Newton***, une approximation de la racine carrée d'un nombre réel positif ***w*** sans utiliser d'autres opérations élémentaires que **+ , - , * , et /.**

Structure du programme : • Entrées : *w et Tol* (variable : nombre)
• Sortie : *x1* (variable : nombre)

c) Tester votre programme avec le calcul de $\sqrt{2}$ et $\sqrt{\frac{1}{2}}$ à une précision de $Tol = 10^{-10}$.

Comparer le résultat obtenu avec celui de votre machine à calculer scientifique.

Nom du fichier : ex 14_newton_racine.py

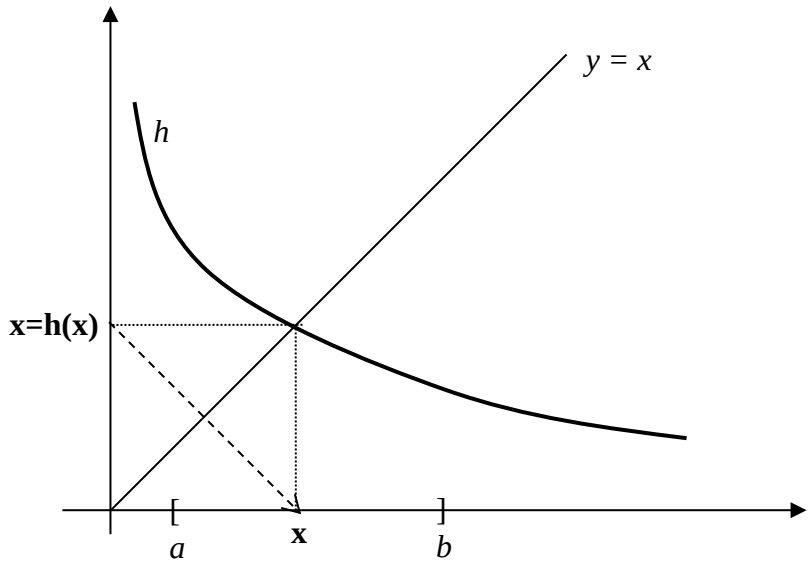
1.4 Méthode du point fixe

Définition

Soit *h* une fonction donnée, alors toute solution de l'équation $\boldsymbol{h(x) = x}$ est appelée ***un point fixe de h***. Graphiquement, cela revient à déterminer les points d'intersections entre le graphique de la fonction *h* et la droite identité : *y = x*. C'est aussi une valeur *x* tel que *la préimage* est égale à *l'image* par *h*.

Exemple $\underbrace{x^2-2}_{=h(x)} = x \Leftrightarrow x^2-x-2=0 \Leftrightarrow (x-2)(x+1)=0 \Leftrightarrow x=2 \quad ou \quad x=-1$ sont deux points fixes de *h*.

Illustration



Remarques

a) Si une fonction h est continue sur $[a;b]$ et que $sgn(a-h(a)) \neq sgn(b-h(b))$ alors la fonction admet au moins un point fixe sur $[a;b]$.

b) Toute équation du type $f(x)=0$ peut se ramener (de plusieurs manières) à une équation du type $h(x)=x$.

Exemple : $\underbrace{e^x-2x-3}_{=f(x)}=0 \Leftrightarrow \underbrace{e^x-x-3}_{=\hat{h}(x)}=x \Leftrightarrow x=\underbrace{ln(2x+3)}_{=h(x)} \Leftrightarrow \ldots$

Conclusion

La résolution de l'équation $f(x)=0$ peut se ramener à la recherche d'un point fixe pour h.

Explication : $f(x)=0 \overset{+x}{\Leftrightarrow} \underbrace{f(x)+x}_{h(x)} = x \Leftrightarrow h(x)=x$

Autrement dit, les *x* qui satisfont $f(x)=0$ satisfont aussi $h(x)=x$ et réciproquement.

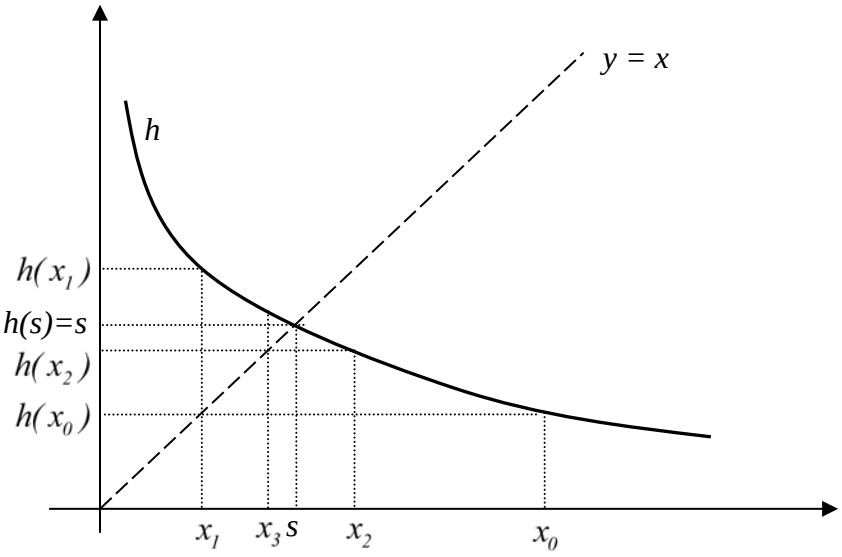
Description de la méthode du point fixe

• On construit une suite avec :

$$x_0 \in [a\,;b] \;\; ; \;\; x_1 = h(x_0) \;\; ; \;\; x_2 = h(x_1) \;; \ldots . \; ; \; x_{n+1} = h(x_n) \;; \ldots .$$

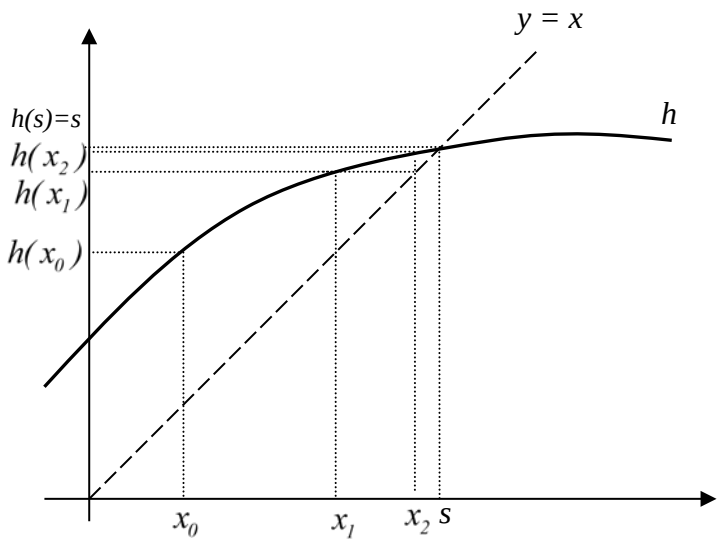
On peut ainsi observer *4 situations types* :

a) convergence alternée, point fixe attirant (*s* est un point fixe de *h*)



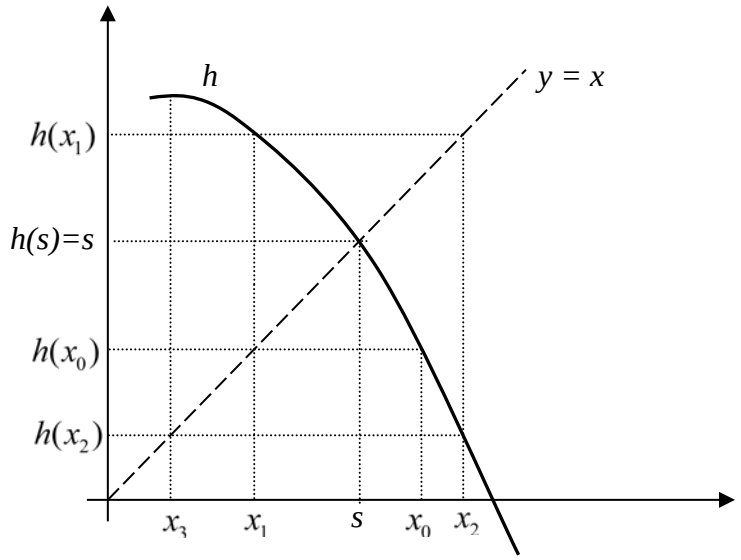
La suite $\{x_n\}_{n\in\mathbb{N}}$ converge vers *s*.

b) convergence monotone, point fixe attirant (*s* est un point fixe de *h*)



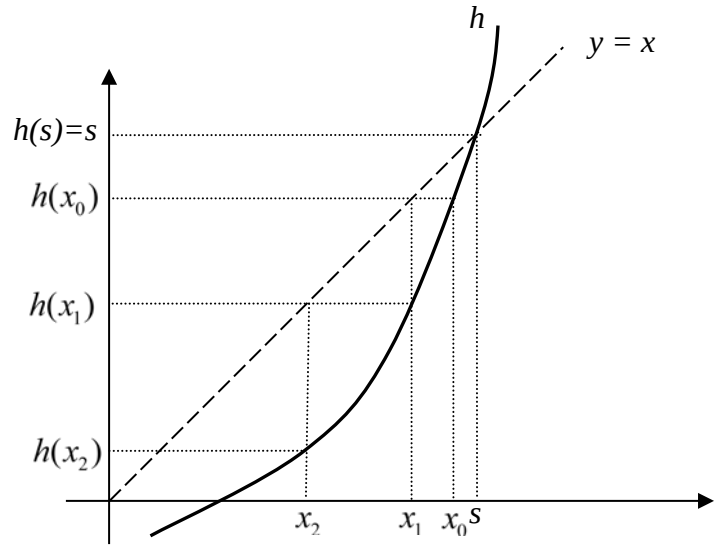
La suite $\{x_n\}_{n\in\mathbb{N}}$ converge vers *s*.

c) divergence, point fixe repoussant (*s* est un point fixe de *h*)



La suite $\{x_n\}_{n\in\mathbb{N}}$ ne converge pas vers *s*.

d) divergence, point fixe repoussant (*s* est un point fixe de *h*)



La suite $\{x_n\}_{n\in\mathbb{N}}$ ne converge pas vers *s*.

Remarques

a) On peut observer qu'il y a convergence lorsque les pentes en valeur absolue des tangentes au graphe de h sont inférieures à *un* dans l'intervalle considéré et divergence dans le cas contraire.

b)

x_0

$x_1 = h(x_0)$

$x_2 = h(x_1) = h(h(x_0)) = (h \circ h)(x_0)$

.....

$x_{n+1} = h(x_n) = ... = \underbrace{(h \circ h \circ ..... \circ h)}_{n+1 \text{ compositions de } h}(x_0)$

Programmation de la méthode du point fixe avec Python 3

On veut résoudre par exemple l'équation suivante que l'on transforme en problème de point fixe :

$$\underbrace{e^x - 2x - 3}_{=f(x)} = 0 \Leftrightarrow e^x = 2x + 3 \Leftrightarrow x = \underbrace{ln(2x+3)}_{=h(x)}$$

La fonction *h* dans cet exemple *est :* $h(x) = ln(2x+3)$.

On peut proposer **le programme Python 3** ci-dessous utilisant ***la méthode du point fixe*** pour déterminer plusieurs termes d'une suite qui converge peut-être vers un point fixe de *h* et donc un zéro de f en fixant comme premier terme de la suite x_0 et une précision de Tol .

```
# Fonction h

import math as mt

def h(x):
    return mt.log(2*x+3)

# Méthode du Point fixe

def POINTFIXE(y,x0,Tol):
    x1=y(x0)
    while abs(x1-x0)>Tol:
        x0=x1
        x1=y(x0)
    return x1

# Programme « principal »

x0=eval(input("Entrez l'approximation initiale d'un point fixe x0= "))
Tol=eval(input("Entrez la précision : Tol="))

x1 = POINTFIXE(h,x0,Tol)

print("Une approximation d'un point fixe de h est ",x1)
```

Activité 15

Compléter le tableau ci-dessous afin de tester le programme se trouvant à la page précédente avec :

• $h(x) = ln(2x+3)$; $x_0 = 1.5$; $Tol = 0.1$

	x0	x1	Tol	$\lvert x1 - x0 \rvert$	Booléen
Entrées	**1.5**		**0.1**		
Boucle 1					
Boucle 2					
Sortie					

Remarque : $h(.....) \cong .....$ *et* $f(.....) \cong .....$

Exercice 16

a) Ecrire dans un éditeur et comprendre ***le programme en Python 3*** qui calcule les termes d'une suite obtenue à l'aide de **la méthode du point fixe.** Cette suite converge peut-être vers un zéro de f.

b) Modifier le programme du point a) pour qu'il ***indique / affiche*** en plus :

1) les résultats de la méthode du point fixe à chaque itération.

Autrement dit : $i \rightarrow x_i$

$i+1 \rightarrow x_{i+1}$

…….

2) La *précision* désirée et le *nombre d'itérations* nécessaires pour obtenir cette précision.

c) Tester votre programme avec :
- $h(x) = ln(2x+3)$; $x_0 = 1.5$; $Tol = 0,1$
- $h(x) = ln(2x+3)$; $x_0 = 1.5$; $Tol = 10^{-5}$

Que constate-t-on ?

Exemple : Sortie Python 3 (console)

```
Entrez l'approximation initiale d'un point fixe x0= 1.5
Entrez la précision : Tol=10**-1
>>>
i    xi
0    1.5
1    1.791759469228055
2    1.8845693954648075
Une approximation d'un point fixe de h est  1.8845693954648075
Pour atteindre la précision de  0.1 le nombre d'itérations nécessaire doit être au minimum de 2 .
```

Nom du fichier : ex 16_point fixe_affichage.py

Théorème de convergence du point fixe

Si **1)** h est une fonction continûment dérivable sur $[a;b]$ (h est dérivable et h' est continue)

2) $sgn(a-h(a)) \neq sgn(b-h(b))$ c-à-d $(a-h(a))\cdot(b-h(b))<0$

3) $|h'(x)|<1 \ \forall x \in [a;b]$

alors h possède un unique point fixe $s \in [a;b]$ et la suite

$x_0 \in [a;b]$; $x_1 = h(x_0)$; $x_2 = h(x_1)$; .... ; $x_{n+1} = h(x_n)$; .... converge vers s.

Exemple

Équation : $\underbrace{e^x - 2x - 3}_{=f(x)} = 0 \Leftrightarrow \underbrace{e^x - x - 3}_{=\hat{h}(x)} = x \Leftrightarrow x = \underbrace{ln(2x+3)}_{=h(x)}$.

Cela revient donc à chercher *un point fixe* pour la fonction $h(x) = ln(2x+3)$.

La fonction *h* satisfait les hypothèses du *théorème* sur l'intervalle $[1;3]$ car :

1) h est une fonction continûment dérivable sur $[1;3]$ car $h'(x) = \dfrac{2}{2x+3}$.

2) $1 - h(1) < 0$ et $3 - h(3) > 0$

3) $|h'(x)| = \left|\dfrac{2}{2x+3}\right| < 1 \quad \forall x \in [1;3]$ *(vérification graphique)*

Conclusion : la suite de *la méthode du point fixe* converge vers l'unique point fixe $s \in [1;3]$.

Démonstration *

i) Soit f une fonction définie par $f(x) = x - h(x)$. f possède au moins un zéro sur $[a;b]$ car $sgn(a-h(a)) \neq sgn(b-h(b))$ et f est continue $[a;b]$.

Supposons par l'absurde que f admette deux zéros s_1 et s_2, autrement dit que h admette deux points fixes s_1 et s_2.

La fonction *h* satisfait les hypothèses du théorème des accroissements finis et donc :

$\dfrac{h(s_2)-h(s_1)}{s_2-s_1} = h'(c)$ avec $c \in]s_1;s_2[$ et comme $h(s_1) = s_1$ et $h(s_2) = s_2$ on a que $h'(c) = 1$

$\Rightarrow\Leftarrow$ avec l'hypothèse $|h'(x)| < 1 \ \forall x \in [a;b]$ donc h possède un unique point fixe sur $[a;b]$.

ii) La fonction *h* satisfait les hypothèses du théorème des accroissements finis et donc :

$\dfrac{h(x_{n-1})-h(s)}{x_{n-1}-s} = h'(c_n)$ avec $c_n \in]x_{n-1};s[$ et $\forall n \geq 1$.

On a : $\dfrac{x_n - s}{x_{n-1} - s} = h'(c_n) \Leftrightarrow |x_n - s| = |h'(c_n)| \cdot |x_{n-1} - s|$

Soit M le maximum de $|h'(x)| \ \forall x \in [a;b]$ avec $0 \leq M < 1$.

On a $|x_n - s| \leq M \cdot |x_{n-1} - s| \leq M \cdot M \cdot |x_{n-2} - s| \leq ... \leq M^n |x_0 - s|$

Et $\lim\limits_{n\to\infty} |x_n - s| \leq \lim\limits_{n\to\infty} M^n |x_0 - s| = 0$ se qui prouve bien que la suite des x_n converge vers s. □

Remarques

a) On peut prouver que plus $|h'(s)|$ est petit, plus la convergence est rapide.

b) On peut également prouver qu'il est raisonnable de continuer les calculs aussi longtemps que que $|x_{n+1} - x_n|$ est supérieur à *une tolérance positive donnée*.

Exercice 17

Considérons l'équation suivante : $\underbrace{\frac{1}{2}\left(x+\frac{5}{x}\right)}_{h(x)} = x$ avec $x>0$ (problème de point fixe)

a) Tracer à ***l'aide de Python*** le graphique de la fonction *h*, de sa dérivée première *h'* en valeur absolue et de la fonction $i(x)=x$ dans le même repère sur un intervalle $[a;b]$.

b) Choisir un intervalle $[a;b]$ sur lequel les hypothèses du théorème du point fixe sont vérifiées.

c) Calculer à ***l'aide de Python*** et en utilisant la ***méthode du point fixe*** une approximation du ***point fixe de h*** avec une *précision* de 10^{-10}.

d) Résoudre algébriquement l'équation et comparer la valeur exacte obtenue avec l'approximation calculée en c).

Nom du fichier : ex 17_point fixe.py

Exercice 18

a) L'aire d'un triangle rectangle vaut *12 cm²*. La hauteur issue de l'angle droit partage l'hypoténuse en deux parties de longueurs respectives 2 cm et *x* cm .

Montrer que *x* est solution de l'équation $(2+x)\sqrt{2x}-24=0$

b) Soient deux fonctions *h* et $\hat{h}$ définies par $h(x)=\frac{24}{\sqrt{2x}}-2$ et $\hat{h}(x)=\frac{1}{2}\left(\frac{24}{2+x}\right)^2$.

Montrer que les équations $h(x)=x$ et $\hat{h}(x)=x$ sont équivalentes à l'équation : $(2+x)\sqrt{2x}-24=0$.

c) Calculer à ***l'aide de Python*** et en utilisant la ***méthode du point fixe*** une approximation du ***point fixe de h*** avec une *précision* de 10^{-2}.

Vérifier auparavant si les hypothèses du théorème de convergence sont satisfaites pour *h* sur un intervalle $[a;b]$.

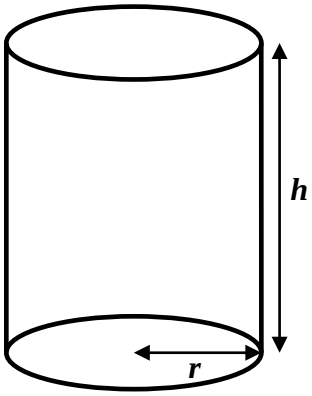
d) Calculer à ***l'aide de Python*** et en utilisant la ***méthode du point fixe*** une approximation du ***point fixe de*** $\hat{h}$ avec une *précision* de 10^{-2}.

Vérifier auparavant si les hypothèses du théorème de convergence sont satisfaites pour $\hat{h}$ sur un intervalle $[a;b]$.

Nom du fichier : ex 18_point fixe.py

Exercice 19 **(problème d'optimisation)**

On désire fabriquer **une canette en aluminium** qui ait la forme d'un cylindre circulaire droit de hauteur *h* et de rayon *r* fermé aux deux extrémités, d'un volume de *33 cl* (voir les figures).



En utilisant le calcul différentiel, répondre aux questions suivantes :

a) Quelles sont les cotes *h* et *r* en *cm* qui **minimisent le coût** de la canette si l'aluminium employé pour les disques coûte 0,002 Fr/cm^2 et celui utilisé pour le cylindre circulaire droit, 0,001 Fr/cm^2 ?

Quelle est son *coût minimum* ?

b) Par souci d'écologie, le fabriquant aimerait construire cette canette avec un *minimum d'aluminium*. Est-ce compatible avec un *coût minimum* ? Justifier par des calculs.

Etapes de résolutions du problème

1) Proposer un schéma avec des cotes.
2) Identifier les variables et les constantes du problème avec les unités.
3) Etablir les liens entre les variables et les constantes (obtention d'une fonction f).
4) Représenter à l'aide de ***Python*** le graphique de la fonction f et identifier les éventuels extremums locaux.
5) Rechercher les extremums locaux de f à l'aide du calcul différentiel :
5.1) Calculer la dérivée première de f en indiquant toutes les étapes de manière ***analytique et sans logiciel***.
5.2) Rechercher les solutions de $f'(x)=0$ à l'aide des méthodes suivantes :
i) ***Analytique* et *sans logiciel***.
Numérique avec ***Python*** :
ii) méthode de la ***bissection*** et affichage des itérations (précision : 10^{-3})
iii) méthode de ***Newton*** et affichage des itérations (précision : 10^{-3})
iv) méthode du ***point fixe*** et affichage des itérations (précision : 10^{-3})
5.3) Etablir le tableau des variations de f.
6) Réponse à la question posée (phrase en français et avec les unités).

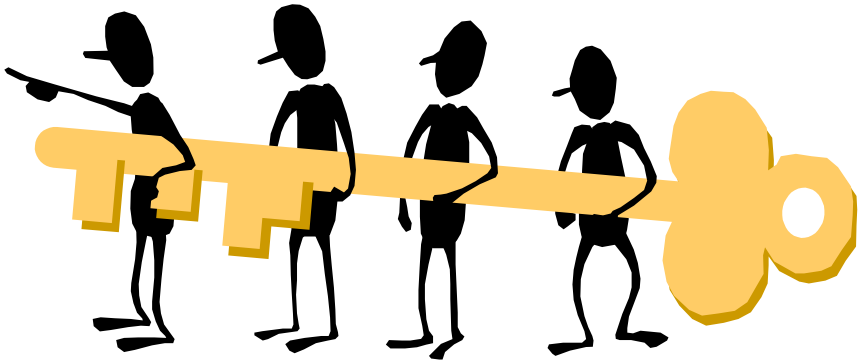
Nom du fichier : ex 19_optimisation.py

ANALYSE NUMERIQUE

Résolution d'équations non linéaires (logiciel Python)

CORRECTIONS

ACTIVITES ET EXERCICES

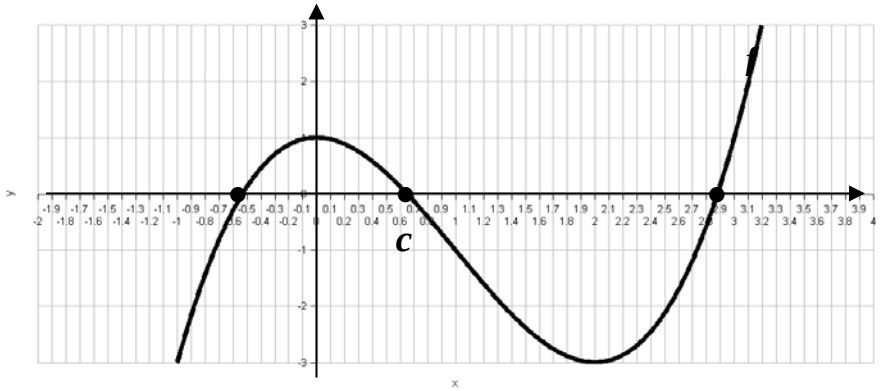


- Montrons que l'équation $x^3 - 3x^2 + 1 = 0$ possède une solution dans l'intervalle $[0\ ;1]$.

  Posons $f(x) = x^3 - 3x^2 + 1$.

  On a $f(0) \cdot f(1) \leq 0$ et *f* est une fonction **continue** sur $\mathbb{R}$ (car c'est un polynôme).

  Donc, d'après le théorème de Bolzano, il existe un nombre c∈**[0;1]** tel que f(c) = **0**.



- Déterminons une ***approximation*** du nombre ***c*** à l'aide de la ***méthode de la bissection*** avec une précision de *Tol = 0,3*.

$$a_0 = 0 < c < 1 = b_0$$
$$b_0 - a_0 = 1 > 0.3 = Tol$$
$$m_1 = \frac{a_0 + b_0}{2} = \frac{0+1}{2} = 0,5$$
$$f(a_0) \cdot f(m_1) = f(0) \cdot f(0.5) = 1 \cdot 0.375 > 0$$

Itération 1 :

$$a_1 = 0.5 < c < 1 = b_1$$
$$b_1 - a_1 = 0.5 > 0.3 = Tol$$
$$m_2 = \frac{a_1 + b_1}{2} = \frac{0.5+1}{2} = 0,75$$
$$f(a_1) \cdot f(m_2) = f(0.5) \cdot f(0.75) = 0.375 \cdot (-0,265625) \leq 0$$

Itération 2 :

$$a_2 = 0.5 < c < 0.75 = b_2$$
$$b_2 - a_2 = 0.25 < 0.3 = Tol$$

- ***Un zéro*** de $f(x) = x^3 - 3x^2 + 1$ se trouve entre $a_2 = 0.5$ *et* $b_2 = 0.75$.
- ***L'erreur maximale*** commise est de $b_2 - a_2 = 0.25$ si l'on choisit $a_2 = 0.5$ *ou* $b_2 = 0.75$ comme approximation d'un zéro de *f*.
- Dans ce cas, il a suffit de ***2 itérations*** pour obtenir la précision (tolérance) désirée.

Correction Activité 02

- $f(x) = x^3 - 3x^2 + 1 \quad ; \quad a = 0 \quad ; \quad b = 1 \quad ; \quad Tol = 0{,}3$

	a	b	Tol	m	b-a	f(a)·f(m)	Booléen
Entrées	**0**	**1**	**0.3**				
Boucle 1							
Test	0	1	0.3		1		V
$m = \frac{a+b}{2}$	0	1	0.3	0.5			
Test	0	1	0.3	0.5		1·0,375	F
$a = m$	0.5	1	0.3	0.5			
Boucle 2							
Test	0.5	1	0.3	0.5	0.5		V
$m = \frac{a+b}{2}$	0.5	1	0.3	0.75			
Test	0.5	1	0.3	0.75		0,375·(–0,265625)	V
$b = m$	0.5	0.75	0.3	0.75			
Boucle 3							
Test	0.5	0.75	0.3	0.75	0.25		F
Sortie	**0.5**	**0.75**					

Exercice 03

a) Programme Python (zone de script)

```
# Fonction f

def f(x):
    return x**3-3*x**2+1

# Méthode de la bissection

def BISSECTION(y,a,b,Tol):
    while b-a>Tol:
        m=(a+b)/2
        if y(a)*y(m)<=0:
            b=m
        else:
            a=m
    return a,b

# Programme principal

a=eval(input("Entrez la borne inférieure de l'intervalle : a="))
b=eval(input("Entrez la borne supérieure de l'intervalle : b="))
Tol=eval(input("Entrez la précision : Tol="))

while f(a)*f(b)>0 or Tol<0:
    a=eval(input("Entrez la borne inférieure de l'intervalle : a="))
    b=eval(input("Entrez la borne supérieure de l'intervalle : b="))
    Tol=eval(input("Entrez la précision : Tol="))

a,b = BISSECTION(f,a,b,Tol)

print("Un zéro de f se trouve dans l'intervalle [",a,";",b,"]")
```

Nom du fichier : ex_03_bissection_cours.py

Sortie Python (console)

```
Entrez la borne inférieure de l'intervalle : a=0
Entrez la borne supérieure de l'intervalle : b=1
Entrez la précision : Tol=10**-5

>>>
Un zéro de f se trouve dans l'intervalle [ 0.6527023315429688 ; 0.6527099609375 ]
```

b) Que constate-t-on ?

Si on augmente la précision : Tol on aura une meilleure approximation d'un zéro de f.

a) Programme Python (zone de script) 

```
# Fonction f

def f(x):
    return x**3-3*x**2+1

# Entrees

a=eval(input("Entrez la borne inférieure de l'intervalle : a="))
b=eval(input("Entrez la borne supérieure de l'intervalle : b="))
Tol=eval(input("Entrez la précision : Tol="))

while f(a)*f(b)>0 or Tol<0:
    a=eval(input("Entrez la borne inférieure de l'intervalle : a="))
    b=eval(input("Entrez la borne supérieure de l'intervalle : b="))
    Tol=eval(input("Entrez la précision : Tol="))

# Methode de la bissection avec affichage intermediaire

i=0
print("i    [a;b]")
print(i,"   [",a,";",b,"]")

while b-a>Tol:
    m=(a+b)/2
    if f(a)*f(m)<=0:
        b=m
    else:
        a=m
    i=i+1
    print(i,"   [",a,";",b,"]")

print("Un zéro de f se trouve dans l'intervalle [",a,";",b,"] .")

print("Pour atteindre la precision de ",Tol,"le nombre \
d'itérations nécessaire doit être au minimum de ",i,".")

print("L'erreur maximale est de ",b-a,".")
```

Nom du fichier : ex_04_bissection_affichage.py

b) Sortie Python (console)

```
Entrez la borne inférieure de l'intervalle : a=0
Entrez la borne supérieure de l'intervalle : b=1
Entrez la précision : Tol=0.3

>>>
i    [a;b]
0    [ 0 ; 1 ]
1    [ 0.5 ; 1 ]
2    [ 0.5 ; 0.75 ]
Un zéro de f se trouve dans l'intervalle [ 0.5 ; 0.75 ] .
Pour atteindre la précision de  0.3 le nombre d'itérations nécessaire doit être au minimum de  2 .
L'erreur maximale est de  0.25 .
```

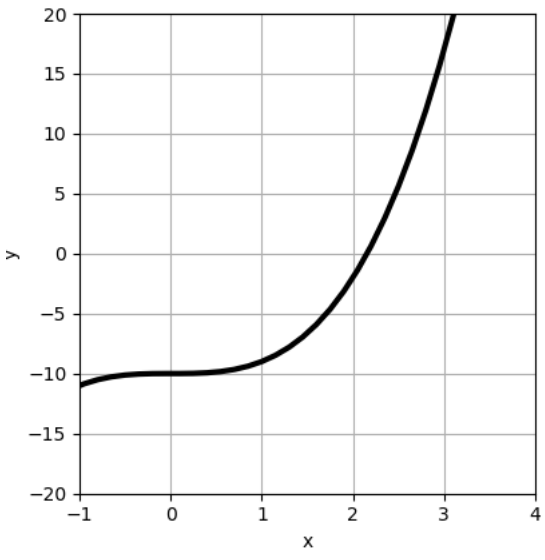
```
Entrez la borne inférieure de l'intervalle : a=0
Entrez la borne supérieure de l'intervalle : b=1
Entrez la précision : Tol=10**-5

>>>
i    [a;b]
0    [ 0 ; 1 ]
1    [ 0.5 ; 1 ]
2    [ 0.5 ; 0.75 ]
3    [ 0.625 ; 0.75 ]
4    [ 0.625 ; 0.6875 ]
5    [ 0.625 ; 0.65625 ]
6    [ 0.640625 ; 0.65625 ]
7    [ 0.6484375 ; 0.65625 ]
8    [ 0.65234375 ; 0.65625 ]
9    [ 0.65234375 ; 0.654296875 ]
10    [ 0.65234375 ; 0.6533203125 ]
11    [ 0.65234375 ; 0.65283203125 ]
12    [ 0.652587890625 ; 0.65283203125 ]
13    [ 0.652587890625 ; 0.6527099609375 ]
14    [ 0.65264892578125 ; 0.6527099609375 ]
15    [ 0.652679443359375 ; 0.6527099609375 ]
16    [ 0.6526947021484375 ; 0.6527099609375 ]
17    [ 0.6527023315429688 ; 0.6527099609375 ]
Un zéro de f se trouve dans l'intervalle [ 0.6527023315429688 ; 0.6527099609375 ] .
Pour atteindre la précision de  1e-05 le nombre d'itérations nécessaire doit être au
minimum de  17 .
L'erreur maximale est de  7.62939453125e-06 .
```

a.1) Programme Python (zone de script)

```
# Fonction f

def f(x):
    return x**3-10

# Affichage du graphique de f

def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y

import matplotlib.pyplot as plt

a,b,n=-5,10,100
plt.plot(tab_val(f,a,b,n)[0],tab_val(f,a,b,n)[1],'k-',linewidth=3,label="f")
plt.axis([-1,4,-20,20])
plt.grid(True)
plt.xlabel("x", fontsize=10)
plt.ylabel("y", fontsize=10)
```

Nom du fichier : ex_05_bissection.py

a.1) Sortie Python (console)



a.2) Programme Python (zone de script)

```
# Fonction f

def f(x):
    return x**3 -10

# Methode de la bissection

def BISSECTION(y,a,b,Tol):
    while b-a>Tol:
        m=(a+b)/2
        if y(a)*y(m)<=0:
            b=m
        else:
            a=m
    return a,b
```

Nom du fichier : ex_05_bissection.py

a.2) Sortie Python (console)

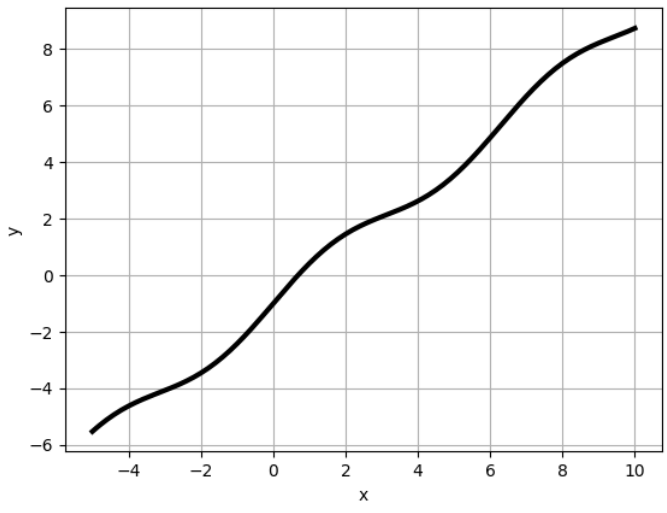
```
BISSECTION(f,2,3,10**-3)
>>> (2.154296875, 2.1552734375)
```

b.1) Programme Python (zone de script)

```
# Fonction f

import math as mt

def f(x):
    return x-1+(1/2)*mt.sin(x)

# Affichage du graphique de f

def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y

import matplotlib.pyplot as plt

a,b,n=-5,10,100
plt.plot(tab_val(f,a,b,n)[0],tab_val(f,a,b,n)[1],'k-',linewidth=3,label="f")
plt.axis([-6,10,-6,8])
plt.grid(True)
plt.xlabel("x", fontsize=10)
plt.ylabel("y", fontsize=10)
```

Nom du fichier : ex_05_bissection.py

b.1) Sortie Python (console)



b.2) Programme Python (zone de script)

```
# Fonction f

import math as mt

def f(x):
    return x-1+(1/2)*mt.sin(x)

# Methode de la bissection

def BISSECTION(y,a,b,Tol):
    while b-a>Tol:
        m=(a+b)/2
        if y(a)*y(m)<=0:
            b=m
        else:
            a=m
    return a,b
```

Nom du fichier : ex_05_bissection.py

b.2) Sortie Python (console)

```
BISSECTION(f,0,1,10**-3)
>>> (0.68359375, 0.6845703125)
```

Correction exercice 06

Théorème de Thalès :

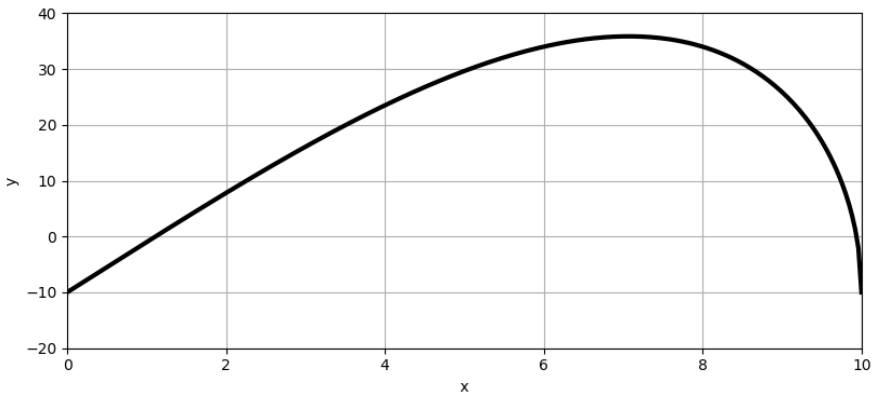
Triangles semblables $\Rightarrow \dfrac{\sqrt{100-x^2}}{x}=\dfrac{1}{x-1} \Leftrightarrow \sqrt{100-x^2}\cdot(x-1)-x=0$

Programme Python (zone de script)

```
# Fonction f

import math as mt

def f(x):
    return mt.sqrt(100-x**2)*(x-1)-x

# Affichage du graphique de f

def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y

import matplotlib.pyplot as plt

a,b,n=0,10,2**8
plt.plot(tab_val(f,a,b,n)[0],tab_val(f,a,b,n)[1],'k-',linewidth=3,label="f")
plt.axis([0,10,-20,40])
plt.grid(True)
plt.xlabel("x", fontsize=10)
plt.ylabel("y", fontsize=10)
```

Nom du fichier : ex_06_bissection.py

Sortie Python (console)



Programme Python (zone de script)

```
# Fonction f

import math as mt

def f(x):
    return mt.sqrt(100-x**2)*(x-1)-x

# Methode de la bissection

def BISSECTION(y,a,b,Tol):
    while b-a>Tol:
        m=(a+b)/2
        if y(a)*y(m)<=0:
            b=m
        else:
            a=m
    return a,b
```

Nom du fichier : ex_06_bissection.py

Sortie Python (console)

```
BISSECTION(f,0,2,10**-3)
>>> (1.111328125, 1.1123046875)
BISSECTION(f,8,10,10**-3)
>>> (9.9375, 9.9384765625)
```

Correction exercice 07

a) $\frac{b_0-a_0}{2^n} \le 10^{-N} \Leftrightarrow \frac{b_0-a_0}{10^{-N}} \le 2^n \Leftrightarrow (b_0-a_0)\cdot 10^N \le 2^n \Leftrightarrow log\left((b_0-a_0)\cdot 10^N\right) \le log\left(2^n\right)$

$\Leftrightarrow log(b_0-a_0)+N \le n\cdot log(2) \Leftrightarrow \boxed{n \ge \frac{log(b_0-a_0)+N}{log(2)}}$

b.1) $n \ge \frac{log(1-0)+5}{log(2)} \cong \boxed{17}$

Non, car f n'intervient pas dans le calcul de $n \ge \frac{log(b_0-a_0)+N}{log(2)}$.

Seuls a_0, b_0 *et* N influencent le nombre d'itérations *n*.

b.2) 10 chiffres exacts après la virgule : $Tol = 10^{-10}$ donc $N = 10$.

$n \ge \frac{log(1-0)+10}{log(2)} = \frac{10}{log(2)} \cong \boxed{34}$

a) Programme Python (zone de script) 

```
# Fonction f

def f(x):
    return x**3-3*x**2+1

# Methode de la bissection (recursif)

def MYSTERE(y,a,b,Tol):
    if b-a<Tol:
        return a,b
    else:
        m=(a+b)/2
        if y(a)*y(m)<=0:
            return MYSTERE(y,a,m,Tol)
        else:
            return MYSTERE(y,m,b,Tol)
    return a,b
```

Nom du fichier : ex_08_fonction_mystere.py

Sortie Python (console)

```
MYSTERE(f,0,1,10**-9)
>>> (0.6527036437764764, 0.652703644707799)

MYSTERE(f,2,3,10**-9)
>>> (2.879385241307318, 2.879385242238641)

MYSTERE(f,-1,0,10**-9)
>>> (-0.5320888869464397, -0.5320888860151172)
```

b) Quel est le rôle de la fonction MYSTERE ? Effectuer la méthode de la bissection.

c) Quel est la particularité de ce programme ?

Programme récursif : Appel de la fonction MYSTERE dans la fonction MYSTERE.

- $f(x) = x^3 - 3x^2 + 1$; $x_0 = 0.5$; $Tol = 0,1$

	x0	x1	Tol	$\lvert x1 - x0 \rvert$	Booléen
Entrées	**0.5**		**0.1**		
$x1 \leftarrow x0 - \frac{f(x0)}{f'(x0)}$	*0.5*	*0,666667*	*0.1*		
Boucle 1			*0.1*		
Test	*0.5*	*0,666667*	*0.1*	*0.166667*	V
$x0 \leftarrow x1$	*0.666667*	*0,666667*	*0.1*		
$x1 \leftarrow x0 - \frac{f(x0)}{f'(x0)}$	*0.666667*	*0.652778*	*0.1*		
Boucle 2			*0.1*		
Test	*0.666667*	*0.652778*	*0.1*	*0.013889*	F
Sortie		***0.652778***			

Remarque : $f(0.652778) \cong 0$

Correction exercice 10

a) Programme Python (zone de script)

```
# On definit la fonctions f et f '

def f(x):
    return x**3-3*x**2+1

def df(x):
    return 3*x**2-6*x

# Methode de Newton avec comme parametres : x0 et Tol

def NEWTON(x0,Tol):
    x1=x0-f(x0)/df(x0)
    while abs(x1-x0)>Tol:
        x0=x1
        x1=x0-f(x0)/df(x0)
    return x1

# Programme principal

x0=eval(input("Entrez l'approximation initiale d'un zéro x0= "))
Tol=eval(input("Entrez la précision : Tol="))

x1 = NEWTON(x0,Tol)

print("Une approximation d'un zéro de f est ",x1)
```

Nom du fichier : ex 10_newton_cours.py

b) Sortie Python (console)

```
Entrez l'approximation initiale d'un zéro x0= 0.5
Entrez la précision : Tol=0.1
>>>
Une approximation d'un zéro de f est  0.6527777777777778
```

```
Entrez l'approximation initiale d'un zéro x0= 0.5
Entrez la précision : Tol=10**-5
>>>
Une approximation d'un zéro de f est  0.6527036446661393
```

c) Que constate-t-on ? Si on augmente la précision : Tol on « devrait » avoir une meilleure approximation d'un zéro de f.

a) Programme Python (zone de script)

```
# On definit la fonction f

def f(x):
    return x**3-3*x**2+1

def df(x):
    return 3*x**2-6*x

# Entrees

x0=eval(input("Entrez l'approximation initiale d'un zéro x0= "))
Tol=eval(input("Entrez la précision : Tol="))

# Methode de Newton avec affichage intermediaire

i=0
print("i     xi")
print(i,"   ",x0)

x1=x0-f(x0)/df(x0)

i=1
print(i,"   ",x1)

while abs(x1-x0)>Tol:
    x0=x1
    x1=x0-f(x0)/df(x0)
    i=i+1
    print(i,"   ",x1)

print("Une approximation d'un zéro de f est ",x1)

print("Pour atteindre la précision de ",Tol,"le nombre d'itérations nécessaire doit être
au minimum de ",i,".")
```

Nom du fichier : ex 11_newton_affichage.py

b) Sortie Python (console)

```
Entrez l'approximation initiale d'un zéro x0= 0.5
Entrez la précision : Tol=0.1

>>>
i     xi
0     0.5
1     0.6666666666666666
2     0.6527777777777778
Une approximation d'un zéro de f est  0.6527777777777778
Pour atteindre la précision de  0.1 le nombre d'itérations nécessaire doit être au
minimum de  2 .
```

```
Entrez l'approximation initiale d'un zéro x0= 0.5
Entrez la précision : Tol=10**-5

>>>
i     xi
0     0.5
1     0.6666666666666666
2     0.6527777777777778
3     0.652703646836132
4     0.6527036446661393
Une approximation d'un zéro de f est  0.6527036446661393
Pour atteindre la précision de  1e-05 le nombre d'itérations nécessaire doit être au
minimum de  4 .
```

Correction exercice 12

$$ln(0,01)=18,19-\frac{23180}{T}-0,8858\cdot ln(T) \Leftrightarrow 18,19-\frac{23180}{T}-0,8858\cdot ln(T)-ln(0,01)=0$$

Considérons la fonction : $f(T)=18,19-\frac{23180}{T}-0,8858\cdot ln(T)-ln(0,01)$

On cherche T tel que $f(T)=0$. La dérivée première de f : $f'(T)=\frac{23180}{T^2}-\frac{0,8858}{T}$

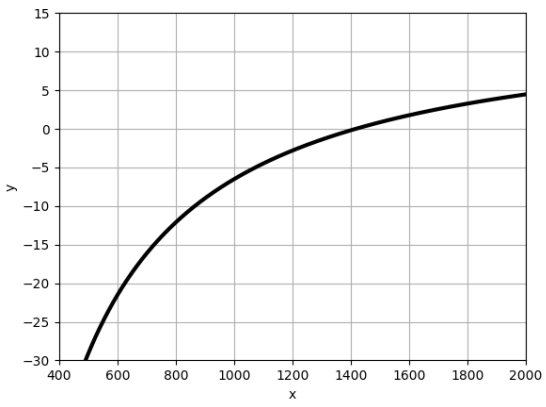
Programme Python (zone de script)

```
# Fonctions f et df

import math as mt

def f(x):
    return 18.19-23180/x-0.8858*mt.log(x)-mt.log(0.01)

def df(x):
    return 23180/x**2-0.8858/x


# Affichage du graphique de f

def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y
```

```
import matplotlib.pyplot as plt

a,b,n=400,2000,100
plt.plot(tab_val(f,a,b,n)[0],\
tab_val(f,a,b,n)[1],'k-',linewidth=3,label="f")
plt.axis([400,2000,-30,15])
plt.grid(True)
plt.xlabel("T", fontsize=10)
plt.ylabel("y", fontsize=10)


# Methode de Newton

def NEWTON(y,dy,x0,Tol):
    x1=x0-y(x0)/dy(x0)
    while abs(x1-x0)>Tol:
        x0=x1
        x1=x0-y(x0)/dy(x0)
    return x1
```

Nom du fichier : ex 12_newton.py

Sortie Python (console)




```
NEWTON(f,df,1400,10**-3)
>>> 1416.1727720923482
```

a) $f(x) = -5x^3 + 7x^2 + 3x - 3$ $\quad f'(x) = -15x^2 + 14x + 3$ $\quad f''(x) = -30x + 14$

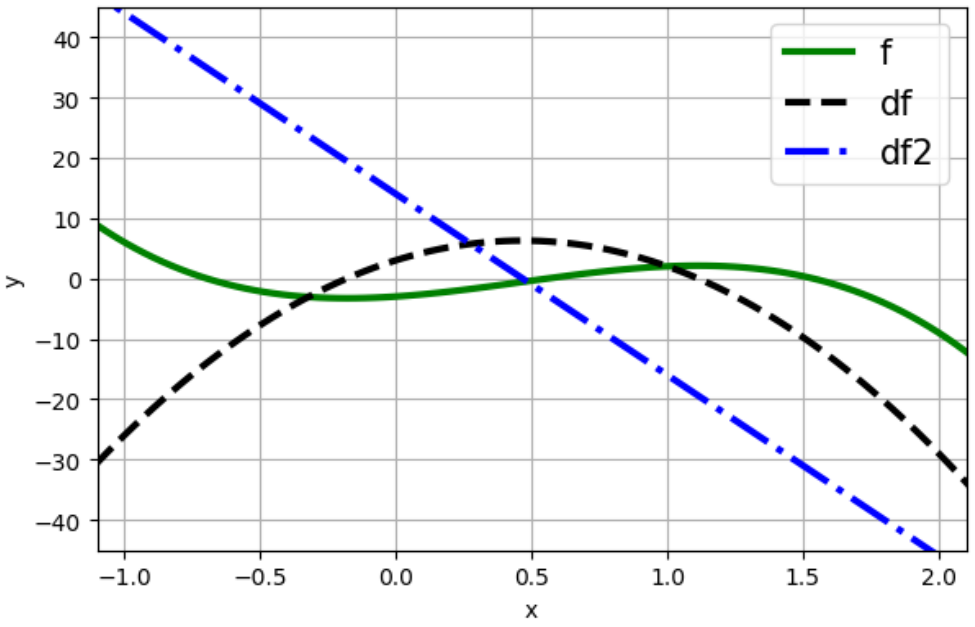
Programme Python (zone de script)

```
# Fonctions f , f' et f''

def f(x):
    return -5*x**3+7*x**2+3*x-3

def df(x):
    return -15*x**2+14*x+3

def d2f(x):
    return -30*x+14

# Affichage du graphique de
# f, df, d2f dans le même repère

def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y
```

```
import matplotlib.pyplot as plt

a,b,n=-1.2,2.2,100

plt.plot(tab_val(f,a,b,n)[0],tab_val(f,a,b,n)[1],\
        'g-',linewidth=3,label="f")
plt.plot(tab_val(df,a,b,n)[0],tab_val(df,a,b,n)[1],\
        'k--',linewidth=3,label="df")
plt.plot(tab_val(d2f,a,b,n)[0],tab_val(d2f,a,b,n)[1],\
        'b-.',linewidth=3,label="df2")

plt.axis([-1.1,2.1,-45,45])
plt.xlabel("x", fontsize=10)
plt.ylabel("y", fontsize=10)
plt.legend(fontsize=15)
plt.grid(True)

plt.show()
```

Nom du fichier : ex 13_newton.py

Sortie Python (console)



b) Programme Python (zone de script)

```
# Méthode de Newton :

x0=eval(input("Entrez l'approximation initiale d'un zéro x0= "))

i=0
print("i     xi")
print(i,"   ",x0)

x1=x0-f(x0)/df(x0)

i=1
print(i,"   ",x1)

for i in range(2,10):
    x0=x1
    x1=x0-f(x0)/df(x0)
    i=i+1
    print(i,"   ",x1)
```

Nom du fichier : ex 13_newton.py

Sortie Python (console)

i xi	**i xi**	**i xi**
0 0 1 1.0 3 0.0 4 1.0 5 0.0 6 1.0 7 0.0 8 1.0 9 0.0 10 1.0	0 1 1 0.0 3 1.0 4 0.0 5 1.0 6 0.0 7 1.0 8 0.0 9 1.0 10 0.0	0 0.5 1 0.56 3 0.5604693611473273 4 0.5604694115063871 5 0.5604694115063876 6 0.5604694115063876 7 0.5604694115063876 8 0.5604694115063876 9 0.5604694115063876 10 0.5604694115063876
Si on choisit : $x_0 = 0$ la suite de Newton ***diverge par oscillation.***	Si on choisit : $x_0 = 1$ la suite de Newton ***diverge par oscillation.***	Si on choisit : $x_0 = 0.5$ la suite de Newton semble ***converger.***

c.i) Vérifications des hypothèses du théorème de convergence sur l'intervalle $[0;1]$:

$f(x) = -5x^3 + 7x^2 + 3x - 3$ sur $[0;1]$

1) $f'(x) = -15x^2 + 14x + 3$ dérivable sur $[0;1]$ et $f''(x) = -30x + 14$ continue sur $[0;1]$

2) $f(0) \cdot f(1) < 0$

3) $f'(x) > 0$ *sur* $[0;1]$ et $f''(x)$ *change de signe sur* $[0;1]$ car $f''(0.4\overline{6}) = 0$

4) En choisissant $x_0 = 0$ ou $x_0 = 1$ on a $f(x_0) \cdot f''(x_0) < 0$

Conclusion : Les hypothèses 3) et 4) du théorème de convergence ne sont pas vérifiées.
On ne peut donc rien dire sur la convergence de la suite de Newton.

c.ii) Vérifications des hypothèses du théorème de convergence sur l'intervalle $[0.5;1]$:

$f(x) = -5x^3 + 7x^2 + 3x - 3$ sur $[0.5;1]$

1) $f'(x) = -15x^2 + 14x + 3$ dérivable sur $[0.5;1]$ et $f''(x) = -30x + 14$ continue sur $[0.5;1]$

2) $f(0.5) \cdot f(1) < 0$

3) $f'(x) > 0$ *sur* $[0.5;1]$ et $f''(x) < 0$ *sur* $[0.5;1]$ avec $f''(0.4\overline{6}) = 0$

4) En choisissant $x_0 = 0.5$ on a $f(x_0) \cdot f''(x_0) > 0$

Conclusion : Toutes les hypothèses du théorème de convergence sont vérifiées, ce qui assure la convergence de la suite de Newton .

a.1) Vérification des hypothèses du théorème pour le cas : 0 ≤ w ≤ 1

Constat : $0 \le w \le \sqrt{w} \le 1$

Regardons sur l'intervalle $[w;1]$:

1) $f(x) = x^2 - w$; $f'(x) = 2x$ dérivable sur $\mathbb{R}$; $f''(x) = 2$ continue sur $\mathbb{R}$

2)
$f(w) = w^2 - w = w(w-1) < 0$ *et* $f(1) = 1^2 - w = 1 - w > 0$
donc $f(w) \cdot f(1) < 0$

3) $f'(x) > 0$ *sur* $[w;1]$ et $f''(x) > 0$ *sur* $[w;1]$

4) En choisissant $x_0 = 1$ nous avons $f(1) \cdot f''(1) > 0$

Conclusion : Convergence de la suite de Newton si l'on choisit $x_0 = 1$.

La limite de cette suite est $\sqrt{w}$.

a.2) Vérification des hypothèses du théorème pour le cas : w > 1

Constat : $1 < \sqrt{w} < w$

Regardons sur l'intervalle $[1;w]$:

1) $f(x) = x^2 - w$; $f'(x) = 2x$ dérivable sur $\mathbb{R}$; $f''(x) = 2$ continue sur $\mathbb{R}$

2)
$f(1) = 1^2 - w = 1 - w < 0$ *et* $f(w) = w^2 - w = w(w-1) > 0$
donc $f(1) \cdot f(w) < 0$

3) $f'(x) > 0$ *sur* $[1;w]$ et $f''(x) > 0$ *sur* $[1;w]$

4) En choisissant $x_0 = w$ nous avons $f(w) \cdot f''(w) > 0$

Conclusion : Convergence de la suite de Newton si l'on choisit $x_0 = w$.

La limite de cette suite est $\sqrt{w}$.

b) Programme Python (zone de script)

```
# Définition de la fonction f et df

from math import *

def f(x):
    return x**2-w
def df(x):
    return 2*x

# Entrees

w=eval(input("Calcul de la racine carrée de "))
while w<0:
    w=eval(input("Calcul de la racine carrée de "))
if w>1:
    x0=w
else:
    x0=1

Tol=eval(input("Entrez la précision : Tol="))
while Tol<=0:
    Tol=eval(input("Entrez la précision : Tol="))

# Méthode de Newton

i=0
print("i     xi")
print(i,"   ",x0)

x1=x0-f(x0)/df(x0)

i=1
print(i,"   ",x1)

while abs(x1-x0)>Tol:
    x0=x1
    x1=x0-f(x0)/df(x0)
    i=i+1
    print(i,"   ",x1)

print("  ")
print("Une approximation de la racine carrée de ",w,"est",x1)
print("  ")
print("Pour atteindre la précision de ",Tol,"le nombre d'itérations
nécessaire doit être au minimum de ",i,".")
```

Nom du fichier : ex 14_newton_racine.py

c) Sortie Python (console)

Calcul de $\sqrt{2}$ avec une précision de 10^{-10}.

```
Calcul de la racine carrée de 2
Entrez la précision : Tol=10**-10

>>>
i     xi
0     2
1     1.5
2     1.4166666666666667
3     1.4142156862745099
4     1.4142135623746899
5     1.4142135623730951

Une approximation de la racine carrée de  2 est 1.4142135623730951

Pour atteindre la précision de  1e-10 le nombre d'itérations nécessaire doit être au minimum de  5 .
```

c) Sortie Python (console)

Calcul de $\sqrt{\frac{1}{2}}$ avec une précision de 10^{-10}.

```
Calcul de la racine carrée de 1/2
Entrez la précision : Tol=10**-10

>>>
i     xi
0     1
1     0.75
2     0.7083333333333334
3     0.7071078431372549
4     0.7071067811873449
5     0.7071067811865476

Une approximation de la racine carrée de  0.5 est 0.7071067811865476

Pour atteindre la précision de  1e-10 le nombre d'itérations nécessaire doit être au minimum de  5 .
```

• $h(x) = ln(2x+3)$; $x_0 = 1.5$; $Tol = 0.1$

	x0	x1	Tol	$\lvert x1 - x0 \rvert$	Booléen
Entrées	**1.5**		**0.1**		
$x1 \leftarrow h(x0)$	1.5	1.791759	0.1		
Boucle 1					
Test	1.5	1.791759	0.1	0.291759	V
$x0 \leftarrow x1$	1.791759		0.1		
$x1 \leftarrow h(x0)$	1.791759	1.884569	0.1		
Boucle 2					
Test	1.791759	1.884569	0.1	0.09281	F
Sortie		**1.884569**			

Remarque : $h(1.884569) \cong 1.884569$ *et* $f(1.884569) \cong 0$

b) Programme Python (zone de script)

```
# On definit la fonction h

import math as mt

def h(x):
    return mt.log(2*x+3)

# Méthode du Point fixe avec comme parametres : x0 et Tol

x0=eval(input("Entrez l'approximation initiale d'un point fixe x0= "))
Tol=eval(input("Entrez la précision : Tol="))

i=0
print("i     xi")
print(i,"   ",x0)

x1=h(x0)

i=1
print(i,"   ",x1)

while abs(x1-x0)>Tol:
    x0=x1
    x1=h(x0)
    i=i+1
    print(i,"   ",x1)

print("Une approximation d'un point fixe de h est ",x1)

print("Pour atteindre la précision de ",Tol,"le nombre d'itérations nécessaire
doit être au minimum de ",i,".")
```

Nom du fichier : ex 16_point fixe_affichage.py

c) Sortie Python (console)

```
Entrez l'approximation initiale d'un point fixe x0= 1.5
Entrez la précision : Tol=0.1

>>>
i     xi
0     1.5
1     1.791759469228055
2     1.8845693954648075
Une approximation d'un point fixe de h est  1.8845693954648075
Pour atteindre la précision de  0.1 le nombre d'itérations nécessaire doit
être au minimum de  2 .
```

```
Entrez l'approximation initiale d'un point fixe x0= 1.5
Entrez la précision : Tol=10**-5

>>>
i     xi
0     1.5
1     1.791759469228055
2     1.8845693954648075
3     1.9123738692207763
4     1.920555378987429
5     1.9229501099671762
6     1.9236499650579206
7     1.923854403782401
8     1.9239141156755313
9     1.923931555486168
10     1.9239366490035965
Une approximation d'un point fixe de h est  1.9239366490035965
Pour atteindre la précision de  1e-05 le nombre d'itérations nécessaire doit
être au minimum de  10 .
```

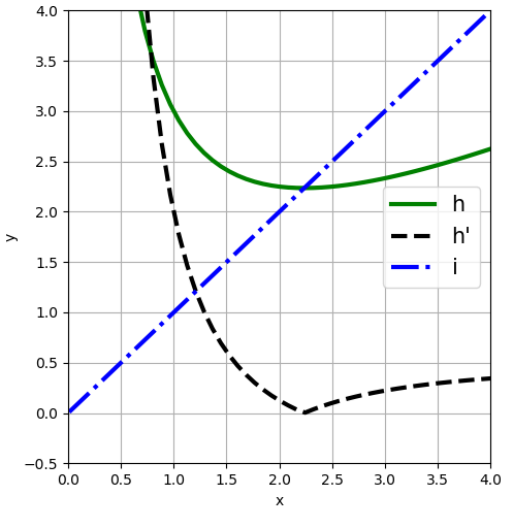
Que constate-t-on ? Si on augmente la précision : Tol on « devrait » avoir une meilleure approximation d'un point fixe de h et donc d'un zéro de f.

```
>>> h( 1.8845693954648075)
1.9123738692207763

>>> h(1.9239366490035965)
1.923938136624934
```

a) Graphiques de : $h(x)=\frac{1}{2}\left(x+\frac{5}{x}\right)$, $|h'(x)|=\left|\frac{1}{2}\left(1-\frac{5}{x^2}\right)\right|$ et $i(x)=x$.

Programme Python (zone de script) **Sortie Python (console)**

```
# Fonctions i, h et h'
def i(x):
    return x
def h(x):
    return (1/2)*(x+5/x)
def abs_dh(x):
    return abs((1/2)*(1-5/(x**2)))

# Affichage du graphique de i, h, h'
def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y

import matplotlib.pyplot as plt

a,b,n=0.01,4,50
plt.plot(tab_val(h,a,b,n)[0],\
tab_val(h,a,b,n)[1],'g-',linewidth=3,label="h")
plt.plot(tab_val(abs_dh,a,b,n)[0],\
tab_val(abs_dh,a,b,n)[1],'k--',linewidth=3,label="h'")
plt.plot(tab_val(i,a,b,n)[0],\
tab_val(i,a,b,n)[1],'b-.',linewidth=3,label="i")
plt.axis([0,4,-0.5,4])
plt.grid(True)
plt.xlabel("x", fontsize=10)
plt.ylabel("y", fontsize=10)
plt.legend(fontsize=15)
plt.show()
```



Nom du fichier : ex 17_point fixe.py

b) Sur $[2;3]$ on a : **1)** h est une fonction continûment dérivable sur $[2;3]$ car $h'(x)=\frac{1}{2}\left(1-\frac{5}{x^2}\right)$

2) $2-h(2)<0$; $3-h(3)>0$

3) $|h'(x)|<1\ \forall x\in[2;3]$

Remarque : Les hypothèses du théorème de convergence sont satisfaites sur $[2;3]$.

La suite obtenue par la méthode du point fixe converge vers l'unique point fixe $s\in[2;3]$. On prendra $x_0=2\in[2;3]$.

c) Programme Python (zone de script)

```
# Méthode du point fixe

def POINTFIXE(y,x0,Tol):
    x1=y(x0)
    while abs(x1-x0)>Tol:
        x0=x1
        x1=y(x0)
    return x1
```

Nom du fichier : ex 17_point fixe.py

Sortie Python (console)

```
POINTFIXE(h,2,10**-10)
>>>  2.23606797749979
```

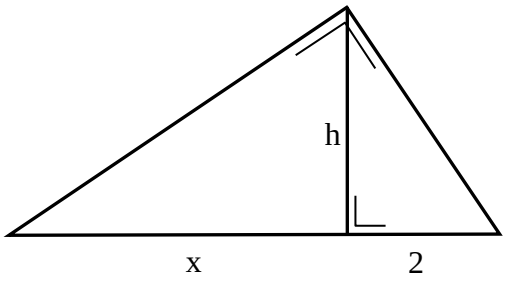
Remarque

```
>>> h(2.23606797749979)
2.23606797749979
```

d) $\frac{1}{2}\left(x+\frac{5}{x}\right)=x \Leftrightarrow x+\frac{5}{x}=2x \Leftrightarrow \frac{5}{x}=x \Leftrightarrow x^2=5 \Leftrightarrow x_1=-\sqrt{5}$ *et* $x_2=\sqrt{5}\in[2;3]$

Correction exercice 18

a)



Aire du triangle $=\frac{(x+2)\cdot h}{2}=12$

et le théorème de la hauteur (Thalès) : $h^2=2x$

Donc $\frac{(x+2)\cdot h}{2}=12$ avec $h=\sqrt{2x}$ donne :

$$\frac{(x+2)\cdot\sqrt{2x}}{2}=12 \Leftrightarrow (x+2)\cdot\sqrt{2x}=24 \Leftrightarrow (x+2)\cdot\sqrt{2x}-24=0$$

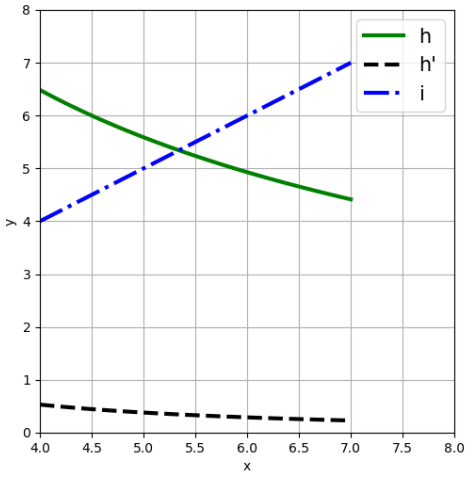
b) $h(x)=\frac{24}{\sqrt{2x}}-2=x \Leftrightarrow 24=(x+2)\sqrt{2x} \Leftrightarrow (x+2)\sqrt{2x}-24=0$

$$\hat{h}(x)=\frac{1}{2}\left(\frac{24}{2+x}\right)^2=x \Leftrightarrow \frac{24}{2+x}=\sqrt{2x} \Leftrightarrow 24=\sqrt{2x}(2+x) \Leftrightarrow (x+2)\sqrt{2x}-24=0$$

c) Graphiques de : $h(x)=\dfrac{24}{\sqrt{2x}}-2$ $\quad |h'(x)|=\left|\dfrac{-12}{\sqrt{8x^3}}\right|$ et $i(x)=x$.

Programme Python (zone de script) **Sortie Python (console)**

```
# Fonctions i, h et h'

import math as mt
def i(x):
    return x
def h(x):
    return 24/mt.sqrt(2*x)-2
def abs_dh(x):
    return abs(-12/mt.sqrt(8*x**3))

# Affichage du graphique de i, h, h'

def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y

import matplotlib.pyplot as plt

a,b,n=4,7,50
plt.plot(tab_val(h,a,b,n)[0],\
tab_val(h,a,b,n)[1],'g-',linewidth=3,label="h")
plt.plot(tab_val(abs_dh,a,b,n)[0],\
tab_val(abs_dh,a,b,n)[1],'k--',linewidth=3,label="h'")
plt.plot(tab_val(i,a,b,n)[0],\
tab_val(i,a,b,n)[1],'b-.',linewidth=3,label="i")
plt.axis([4,8,0,8])
plt.grid(True)
plt.xlabel("x", fontsize=10)
plt.ylabel("y", fontsize=10)
plt.legend(fontsize=15)
plt.show()
```



Nom du fichier : ex 18_point fixe.py

Sur $[5;6]$ on a : **1)** h est une fonction continûment dérivable sur $[5;6]$ car $h'(x)=\dfrac{-12}{\sqrt{8x^3}}$

2) $5-h(5)<0$ *et* $6-h(6)>0$

3) $|h'(x)|<1 \quad \forall x\in[5;6]$

Remarque : Les hypothèses du théorème de convergence sont satisfaites sur $[5;6]$.

La suite obtenue par la méthode du point fixe converge vers l'unique point fixe $s\in[5;6]$. On prendra $x_0=5\in[5;6]$.

Programme Python (zone de script)

```
# Méthode du point fixe

def POINTFIXE(y,x0,Tol):
    x1=y(x0)
    while abs(x1-x0)>Tol:
        x0=x1
        x1=y(x0)
    return x1
```

Nom du fichier : ex 18_point fixe.py

Sortie Python (console)

```
POINTFIXE(h,5,10**-2)
>>> 5.338392085737446
```

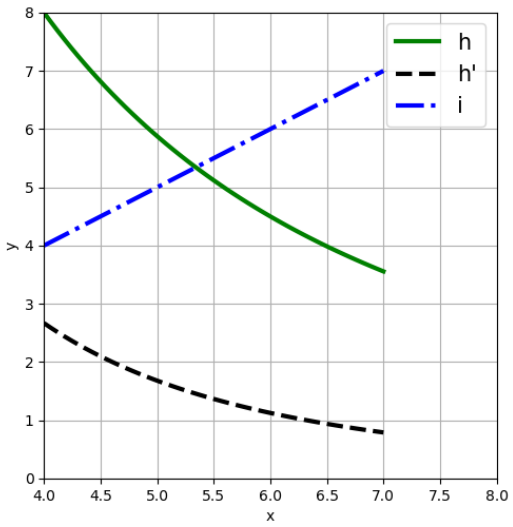
Remarque

```
>>> h(5.338392085737446)
5.344986635027914
```

d) Graphiques de : $\hat{h}(x)=\frac{1}{2}\left(\frac{24}{2+x}\right)^2$ $\quad\left|\hat{h}'(x)\right|=\left|-\frac{24^2}{(2+x)^3}\right|$ et $i(x)=x$.

Programme Python (zone de script) **Sortie Python (console)**

```
# Fonctions i, h et h'

def i(x):
    return x
def h(x):
    return (1/2)*(24/(2+x))**2
def abs_dh(x):
    return abs((-1)*(24**2/(2+x)**3))

# Affichage du graphique de i, h, h'

def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y

import matplotlib.pyplot as plt

a,b,n=4,7,50
plt.plot(tab_val(h,a,b,n)[0],\
tab_val(h,a,b,n)[1],'g-',linewidth=3,label="h")
plt.plot(tab_val(abs_dh,a,b,n)[0],\
tab_val(abs_dh,a,b,n)[1],'k--
',linewidth=3,label="h'")
plt.plot(tab_val(i,a,b,n)[0],\
tab_val(i,a,b,n)[1],'b-.',linewidth=3,label="i")
plt.axis([4,8,0,8])
plt.grid(True)
plt.xlabel("x", fontsize=10)
plt.ylabel("y", fontsize=10)
plt.legend(fontsize=15)
plt.show()
```



Nom du fichier : ex 18_point fixe.py

Sur $[5;6]$ on a : **1)** h est une fonction continûment dérivable sur $[5;6]$ car $h'(x)=\frac{-12}{\sqrt{8x^3}}$

2) $5-h(5)<0$ *et* $6-h(6)>0$

3) $\left|h'(x)\right|>1 \quad \forall x\in[5;6]$

Remarque : Les hypothèses du théorème de convergence **ne sont pas** satisfaites sur $[5;6]$.

On ne sait donc pas si la suite obtenue par la méthode du point fixe va converger vers l'unique point fixe $s\in[5;6]$. On prendra $x_0=5\in[5;6]$.

Programme Python (zone de script)

```
# Méthode du point fixe

x0=5
Tol=10**-2

i=0
print("i     xi")
print(i,"   ",x0)
x1=h(x0)
i=1
print(i,"   ",x1)
while abs(x1-x0)>Tol:
    x0=x1
    x1=h(x0)
    i=i+1
    print(i,"   ",x1)

print("Une approximation d'un point fixe de h est ",x1)
print("Pour atteindre la précision de ",Tol,"le nombre d'itérations nécessaire doit être au
minimum de ",i,".")
```

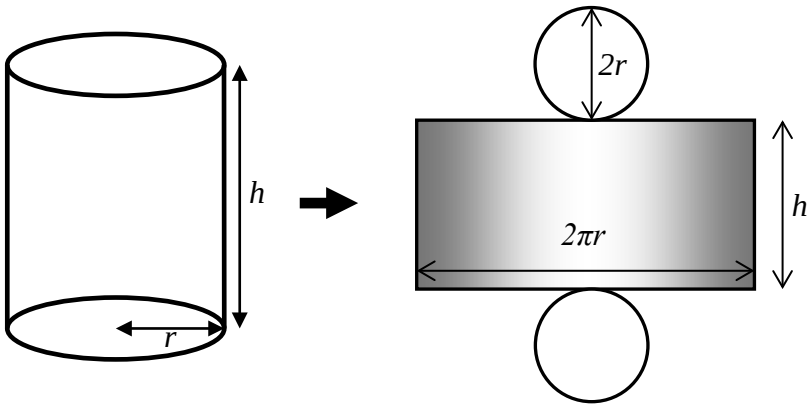
Nom du fichier : ex 18_point fixe.py

Sortie Python (console)

```
>>>
i     xi
0     5
1     5.877551020408163
2     4.640983650567801
3     6.530217789075341
4     3.9579677279404
5     8.113274965228038
6     2.8158457273330595
7     12.417877049102968
8     1.3854467952133571
9     25.128149191325335
10     0.3913381227802303
11     50.36287485290012
12     0.10503777727412626
13     64.99391520560614
…..     …..
6540     0.05887450304571879
6541     67.9411254969543
6542     0.05887450304571879
6543     67.9411254969543
6544     0.05887450304571879
6545     67.9411254969543
6546     0.05887450304571879
6547     67.9411254969543
6548     0.05887450304571879
…..     …..
```

Remarque : La suite obtenue par la méthode du point fixe semble diverger.

1) Schéma avec cotes



2) Déclaration des variables et des constantes du problème

Variables :

r = rayon du cylindre en cm

h = hauteur du cylindre en cm

C = coût du cylindre avec couvercle en Fr. (variable à optimiser).

Constantes :

V = volume du cylindre = *33 cl = 330 cm^3*

Coût de l'aluminium employé pour les disques : *0,002 Fr/cm^2*

Coût de l'aluminium employé pour le cylindre circulaire droit: *0,001 Fr/cm^2*

3) Liens entre les variables et les constantes (obtention d'une fonction *f*)

(I) $330 = \pi r^2 h \Rightarrow h = \dfrac{330}{\pi r^2}$

(II) $C(r;h) = \dfrac{2}{1000} \cdot 2\pi r^2 + \dfrac{1}{1000} \cdot 2\pi r h$

(I) dans **(II) :** $\boxed{C(r)} = \dfrac{2}{1000} \cdot 2\pi r^2 + \dfrac{1}{1000} \cdot 2\pi r \cdot \dfrac{330}{\pi r^2} = \dfrac{\pi r^2}{250} + \dfrac{330}{500 \cdot r} = \boxed{\dfrac{2\pi r^3 + 330}{500 \cdot r}}$

(Coût de la canette C en fonction du rayon r)

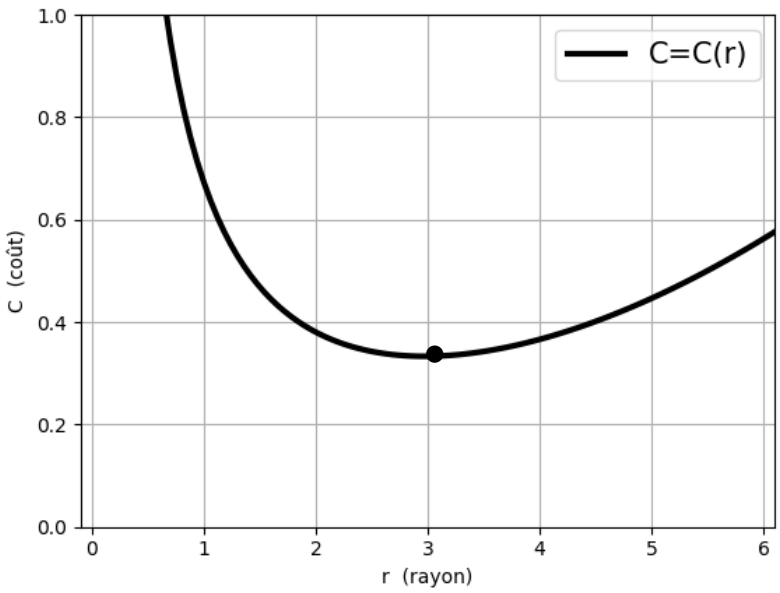
4) Représentation graphique de la fonction $C(r)=\dfrac{2\pi r^3+330}{500\cdot r}$

Programme Python (zone de script)

```
import math as mt

def f(x):
    return (2*mt.pi*x**3+330)/(500*x)

def tab_val(y,a,b,n):
    X=[ ]
    Y=[ ]
    x=a
    pas=(b-a)/n
    for k in range(n+1):
        X.append(x)
        Y.append(y(x))
        x=x+pas
    return X, Y

import matplotlib.pyplot as plt

a,b,n=0.1,6.1,100
plt.plot(tab_val(f,a,b,n)[0],tab_val(f,a,b,n)[1],'k-',linewidth=3,label="C=C(r)")
plt.axis([-0.1,6.1,0,1])
plt.grid(True)
plt.xlabel("r  (rayon)", fontsize=10)
plt.ylabel("C  (coût)", fontsize=10)
plt.legend(fontsize=15)
plt.show()
```

Nom du fichier : ex 19_optimisation.py

Sortie Python (console)



5) Recherche des extremums locaux de *f* à l'aide du calcul différentiel

5.1) Calcul de la dérivée première avec toutes les étapes de manière analytique et sans logiciel.

$$\boxed{C'(r)} = \left(\frac{2\pi r^3 + 330}{500 \cdot r}\right)' = \frac{\left(2\pi r^3 + 330\right)' \cdot \left(500 \cdot r\right) - \left(2\pi r^3 + 330\right) \cdot \left(500 \cdot r\right)'}{\left(500 \cdot r\right)^2} = \boxed{\frac{2\pi r^3 - 165}{250 \cdot r^2}}$$

5.2) Recherche des solutions de $C'(r) = 0$ à l'aide des méthodes suivantes :

i) *Analytique* et *sans logiciel*

$$\begin{aligned} & C'(r) = 0 \\ \Leftrightarrow\ & \frac{2\pi r^3 - 165}{250 \cdot r^2} = 0 \\ \Leftrightarrow\ & 2\pi r^3 - 165 = 0 \\ \Leftrightarrow\ & r = \sqrt[3]{\frac{165}{2\pi}} \cong 2{,}97 \end{aligned}$$

ii) ***Numérique*** avec ***Python*** : méthode de la ***bissection*** et affichage des itérations (préc: 10^{-3})

$$C'(r)=0 \Leftrightarrow \frac{2\pi r^3-165}{250\cdot r^2}=0 \Leftrightarrow \underbrace{2\pi r^3-165}_{\hat{C}'(r)}=0$$

Programme Python (zone de script)

```
import math as mt

def df(x):
    return 2*mt.pi*x**3-165

a,b,Tol=2.5,3,10**-3
i=0
print("i    [a;b]")
print(i,"   [",a,";",b,"]")
while b-a>Tol:
    m=(a+b)/2
    if df(a)*df(m)<=0:
        b=m
    else:
        a=m
    i=i+1
    print(i,"   [",a,";",b,"]")

print("Un zéro de f se trouve dans l'intervalle [",a,";",b,"] .")
print("Pour atteindre la précision de ",Tol,"le nombre d'itérations nécessaire doit
être au minimum de ",i,".")
print("L'erreur maximale est de ",b-a,".")
```

Nom du fichier : ex 19_optimisation.py

Sortie Python (console)

```
i    [a;b]
0    [ 2.5 ; 3 ]
1    [ 2.75 ; 3 ]
2    [ 2.875 ; 3 ]
3    [ 2.9375 ; 3 ]
4    [ 2.96875 ; 3 ]
5    [ 2.96875 ; 2.984375 ]
6    [ 2.96875 ; 2.9765625 ]
7    [ 2.96875 ; 2.97265625 ]
8    [ 2.970703125 ; 2.97265625 ]
9    [ 2.9716796875 ; 2.97265625 ]

Un zéro de f se trouve dans l'intervalle [ 2.9716796875 ; 2.97265625 ] .
Pour atteindre la précision de  0.001 le nombre d'itérations nécessaire
doit être au minimum de  9 .
L'erreur maximale est de  0.0009765625 .
```

iii) ***Numérique*** avec ***Python*** : méthode de ***Newton*** et affichage des itérations (préc: 10^{-3})

$\widehat{C}'(r) = 2\pi r^3 - 165$ $\widehat{C}''(r) = 6\pi r^2$

Programme Python (zone de script)

```
import math as mt

def df(x):
    return 2*mt.pi*x**3-165

def d2f(x):
    return 6*mt.pi*x**2

x0,Tol=2.5,10**-3

i=0
print("i     xi")
print(i,"   ",x0)
x1=x0-df(x0)/d2f(x0)
i=1
print(i,"   ",x1)
while abs(x1-x0)>Tol:
    x0=x1
    x1=x0-df(x0)/d2f(x0)
    i=i+1
    print(i,"   ",x1)

print("Une approximation d'un zéro de f est ",x1)
print("Pour atteindre la précision de ",Tol,"le nombre d'itérations nécessaire doit
être au minimum de ",i,".")
```

Nom du fichier : ex 19_optimisation.py

Sortie Python (console)

```
i     xi

0     2.5
1     3.067230165875346
2     2.975263784680373
3     2.9723624941974065
4     2.972359660436733

Une approximation d'un zéro de f est  2.972359660436733

Pour atteindre la précision de  0.001 le nombre d'itérations nécessaire doit être au minimum
de  4 .
```

iv) ***Numérique*** avec ***Python*** : méthode du ***point fixe*** et affichage des itérations (préc: 10^{-3})

$$C'(r)=0 \Leftrightarrow \frac{2\pi r^3-165}{250\cdot r^2}=0 \Leftrightarrow \frac{\pi r}{125}-\frac{33}{50\cdot r^2}=0 \Leftrightarrow 50\cdot r^2=\frac{33\cdot 125}{\pi r} \Leftrightarrow r^2=\frac{165}{2\pi r} \Leftrightarrow r=\underbrace{\sqrt{\frac{165}{2\pi r}}}_{h(r)}$$

Programme Python (zone de script)

```
import math as mt

def h(x):
    return mt.sqrt(165/(2*mt.pi*x))

x0,Tol=2.5,10**-3

i=0
print("i     xi")
print(i,"   ",x0)
x1=h(x0)
i=1
print(i,"   ",x1)
while abs(x1-x0)>Tol:
    x0=x1
    x1=h(x0)
    i=i+1
    print(i,"   ",x1)

print("Une approximation d'un point fixe de h est ",x1)
print("Pour atteindre la précision de ",Tol,"le nombre d'itérations nécessaire
doit être au minimum de ",i,".")
```

Nom du fichier : ex 19_optimisation.py

Sortie Python (console)

```
i     xi

0     2.5
1     3.2410224072142872
2     2.846498905328389
3     3.037361804968293
4     2.9403821270250754
5     2.9884786036773616
6     2.9643328209311735
7     2.97638122718367
8     2.970350915185604
9     2.9733645424176065
10     2.9718573468036054
11     2.972610849086894

Une approximation d'un point fixe de h est  2.972610849086894

Pour atteindre la précision de  0.001 le nombre d'itérations nécessaire doit être au
minimum de  11 .
```

5.3) Tableau des variations de $C = C(r)$.

r		$2{,}97$	
$C'(r)$	–	0	+
$C(r)$	↘	min	↗

$$h = \frac{330}{\pi \cdot (2{,}97)^2} \cong 11{,}9 \qquad C_{min} = C(2{,}97) \cong 0{,}33$$

6) Réponse à la question posée a)

- Pour que le coût de la canette soit minimale, le rayon doit être d'environ **2,97 cm** et la hauteur d'environ **11,9 cm**.

  Remarque : Le diamètre de la boîte n'est pas égal à sa hauteur.

- Le coût minimal de la canette est d'environ **0,33 Fr.**

Réponse à la question posée b)

- $A(r) = 2\pi r^2 + 2\pi r \cdot \frac{330}{\pi r^2} = 2\pi r^2 + \frac{660}{r} = \frac{2\pi r^3 + 660}{r}$
- $A'(r) = \frac{4\pi r^3 - 660}{r^2}$ et $A'(r) = 0 \Leftrightarrow \pi r^3 - 660 = 0 \Leftrightarrow r = \sqrt[3]{\frac{660}{\pi}} \cong 5{,}94$
- $C(5{,}94) \cong 0{,}45$
- En minimisant la quantité d'aluminium on obtient un coût d'environ **0,45 Fr**. qui est plus élevé que le coût minimal qui est d'environ **0,33 Fr**.

Notes personnelles

Notes personnelles