TP2-Balayage-Dichotomie-Version2

November 3, 2016

0.1 Import des bibliothèques Python

```
In [3]: import numpy as np                  #pour disposer des tableaux de type array
        import matplotlib.pyplot as plt     #pour les graphiques

In [57]: % matplotlib inline
         #pour l'affichage des graphiques dans la page et non pas dans une fenetre

In [58]: import operator                     #pour utiliser les opérateurs de base sc

In [59]: from sympy import *                 #pour le calcul formel
         init_printing()
         t = symbols('t')

In [7]: def dérivée(exp, t):
            return diff(exp,t)

        def simplifier(exp):
            return simplify(exp)
```

0.2 Etude des variations de la fonction $f : x \mapsto (6-x)\sqrt{x}$ sur $[0;6]$

Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = (6-x)\sqrt{x}$ definie sur $[0;6]$ et dérivable sur $]0;6]$.

f est dérivable sur $]0;6]$ comme produit de fonctions dérivables sur $]0;6]$ mais elle n'est pas dérivable en 0.

0.2.1 Question 1 : Calcul de dérivée

```
In [8]: #expression de f(x)
        fexp = (6 - t)*sqrt(t)
        fexp
```

Out[8]:

$$\sqrt{t}\,(-t+6)$$

```
In [9]: #expression de f'(x)
        fprimexp = dérivée(fexp, t)
        fprimexp
```

Out[9]:

$$-\sqrt{t}+\frac{-t+6}{2\sqrt{t}}$$

In [10]: simplifier(fprimexp)

Out[10]:

$$\frac{1}{2\sqrt{t}}\left(-3t+6\right)$$

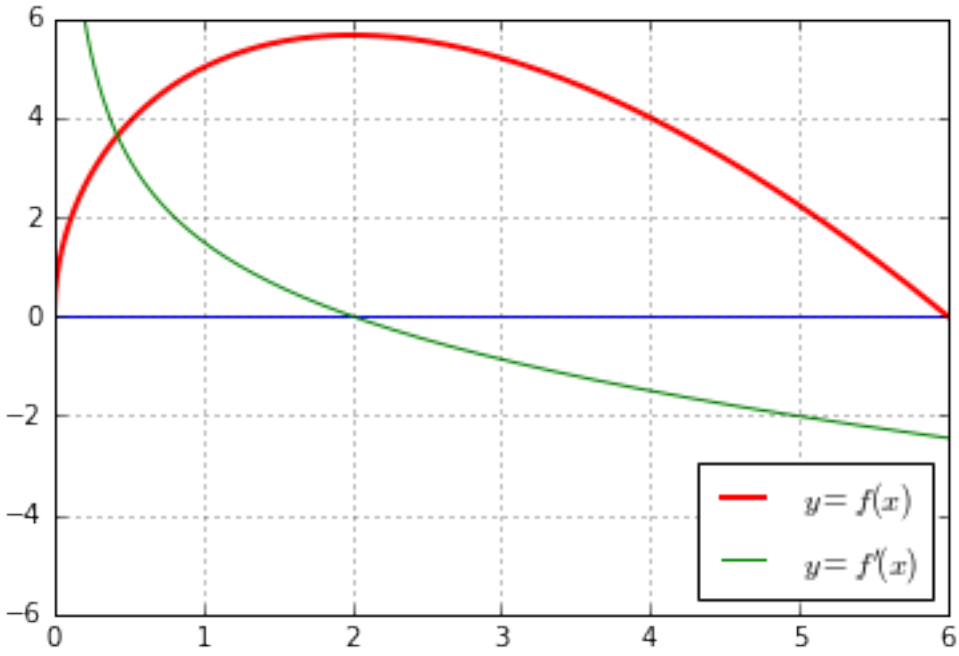
0.2.2 Questions 2 et 3 Etude des variations de f

```
In [11]: f = lambdify(t, fexp,"numpy")
         fprim = lambdify(t, fprimexp,"numpy")
```

```
In [61]: #tracé des  courbes  de f et f'
         #Message d'erreur pour f' pour le point d'abscisse 0 (division par 0)
         xmin, xmax, ymin, ymax = 0, 6, -6 , 6
         plt.axis([xmin, xmax, ymin, ymax])
         tx = np.linspace(xmin, xmax, 1001)
         ty = f(tx)
         tz = fprim(tx)
         plt.axhline(color='blue')
         plt.axvline(color='blue')
         plt.grid(True)
         plt.plot(tx, ty, linestyle='-', linewidth=2, color='red', label=r'$y=f(x)$
         plt.plot(tx, tz, linestyle='-', linewidth=1, color='green', label=r"$y=f'
         plt.legend(loc='lower right')
```

```
/usr/lib/python3/dist-packages/numpy/__init__.py:1: RuntimeWarning: divide by zero
  """
```

Out[61]: <matplotlib.legend.Legend at 0xaf50dd0c>



0.2.3 Question 4 Existence de solutions de l'équation $f(x) = 4$

- $f : x \mapsto (6 - x)\sqrt{x}$ est dérivable donc continue sur $[0, 2]$
- $f(0) < 4$ et $f(2) > 4$
- f est strictement croissante sur $[0; 2]$

D'après un corollaire du théorème des valeurs intermédiaires, l'équation $f(x) = 4$ possède donc une unique solution α dans l'intervalle $[0; 2]$

- $f : x \mapsto (6 - x)\sqrt{x}$ est dérivable donc continue sur $[2, 6]$
- $f(2) > 4$ et $f(6) < 4$
- f est strictement décroissante sur $[0; 2]$

D'après un corollaire du théorème des valeurs intermédiaires, l'équation $f(x) = 4$ possède donc une unique solution β dans l'intervalle $[2, 6]$. De plus cette solution est 4 car $f(4) = 4$.

0.3 Résolution approchée par balayage

```
In [13]: def balayage(g, a, b, pas, k):
             """Retourne un intervalle d'amplitude pas encadrant l'unique solution
             dans l'intervalle [a,b]"""
             if g(a) < k:
                 comparaison = lambda u, v : operator.lt(u,v)
             else:
                 comparaison = lambda u, v : operator.gt(u,v)
             x = a
```

```
            #en-tete du tableau
            print('|{etape:^16}|{t:^12}|{ft:^12}|'.format(etape='Etape', t='t', ft
            count = 1
            while comparaison(g(x), k):
                print('|{etape:^16}|{t:^12.6f}|{ft:^12.6f}|'.format(etape=count,t=
                x += pas
                count += 1
            print('|{etape:^16}|{t:^12.6f}|{ft:^12.6f}|'.format(etape=count,t=x, f
            return x - pas, x
```

In [14]:
```
balayage(f, 0, 2, 0.2, 4)
```

```
|      Etape      |        t        |      g(t)       |
|        1        |    0.000000     |    0.000000     |
|        2        |    0.200000     |    2.593839     |
|        3        |    0.400000     |    3.541751     |
|        4        |    0.600000     |    4.182822     |
```

Out[14]:

$$(0.4000000000000001,\quad 0.6000000000000001)$$

In [16]:
```
balayage(f, 0.4, 0.6, 0.02, 4)
```

```
|      Etape      |        t        |      g(t)       |
|        1        |    0.400000     |    3.541751     |
|        2        |    0.420000     |    3.616253     |
|        3        |    0.440000     |    3.688087     |
|        4        |    0.460000     |    3.757411     |
|        5        |    0.480000     |    3.824368     |
|        6        |    0.500000     |    3.889087     |
|        7        |    0.520000     |    3.951684     |
|        8        |    0.540000     |    4.012264     |
```

Out[16]:

$$(0.5200000000000001,\quad 0.5400000000000001)$$

0.4 Résolution approchée par dichotomie

0.4.1 Fonctions Python

In [49]:
```
def dicho(f,a,b,e, k):
    """valeur approchée à e près de la solution de f(x)=k
    sur [a,b]. On admet que l'utilisateur saisit des paramètres
    où la dichotomie peut s'appliquer. Construit une figure
    à chaque étape.
```

```
    Les valeurs de a et b affichées sont celles en sortie de boucle."""
    etape = 0 #nombre d'étapes
    if f(a) <= f(b):
        #croissant ou du moins passage de - à +
        croissant = True
    else:
        croissant = False
    while b - a > e:
        etape += 1
        #on calcule le milieu du segment [a,b]
        m = (a+b)/2
        #figure
        x = np.linspace(a,b,500)
        y = f(x)
        plt.xlim(a,b)
        if croissant:
            plt.ylim(max(f(a),-50),min(f(b),50))
        else:
            plt.ylim(max(f(b),-50),min(f(a),50))
        plt.plot(x,y,color='red')
        plt.grid(True)
        plt.axhline(k)
        plt.title('a=%.4f et m=%.4f et b=%.4f'%(a,m,b))
        plt.show()
        #fin de la figure
        s = (f(m) - k)*(f(a) - k)
        #si f(m) - k  et f(a) - k sont de meme signe, f(x)=k dans ]m,b[
        if s > 0:
            a = m
        #si f(m) - k  et f(a) - k sont de signes opposés, f(x)=k dans ]a,m
        else:
            b = m
    return a, b, etape

def dicho_tab(f,a,b,e, k):
    """valeur approchée à e près de la solution de f(x)=k
    sur [a,b]. On admet que l'utilisateur saisit des paramètres
    où la dichotomie peut s'appliquer.
    Ne retourne rien mais remplit un tableau avec les valeurs
    de a, b et m aux differentes etapes.
    Les valeurs de a et b affichées sont celles en sortie de boucle"""
    count = 0 #nombre d'étapes
    if f(a) <= f(b):
        #croissant ou du moins passage de - à +
        croissant = True
    else:
        croissant = False
    #en-tete du tableau
```

```
        print('|{etape:^16}|{median:^12}|{test:^10}|{binf:^12}|{bsup:^12}|'.fo
                                                                    binf=
                                                                    test=

        #première ligne du tableau
        #remplissage de la ligne du tableau
        print('|{etape:^16}|{median:^12}|{test:^10}|{binf:^12}|{bsup:^12}|'.fo
                                                                    binf=
                                                                    test=

        while b - a > e:
            count += 1
            #on calcule le milieu du segment [a,b]
            m = (a+b)/2
            s = (f(m) - k)*(f(a) - k)
            #si f(m) - k  et f(a) - k sont de meme signe, f(x)=k dans ]m,b[
            if s > 0:
                a = m
            #si f(m) - k  et f(a) - k sont de signes opposés, f(x)=k dans ]a,m
            else:
                b = m
            #remplissage de la ligne du tableau
            print('|{etape:^16}|{median:^12}|{test:^10}|{binf:^12}|{bsup:^12}|
                                                                    binf=
                                                                    test=
```

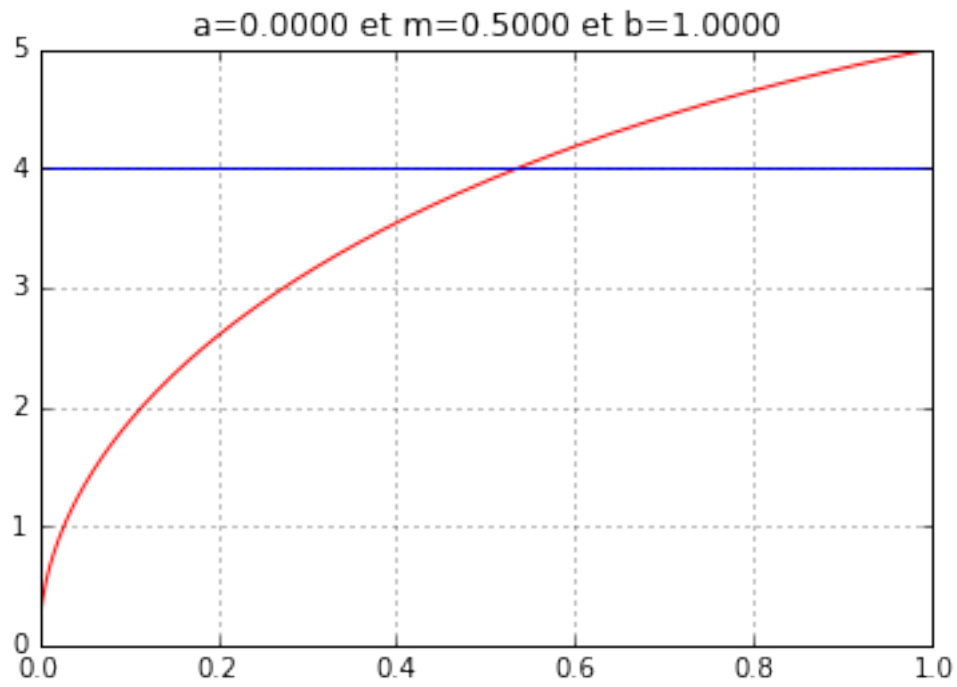
0.5 Résolution par dichotomie de l'équation $f(x) = 4$ dans l'intervalle $[0, 1]$

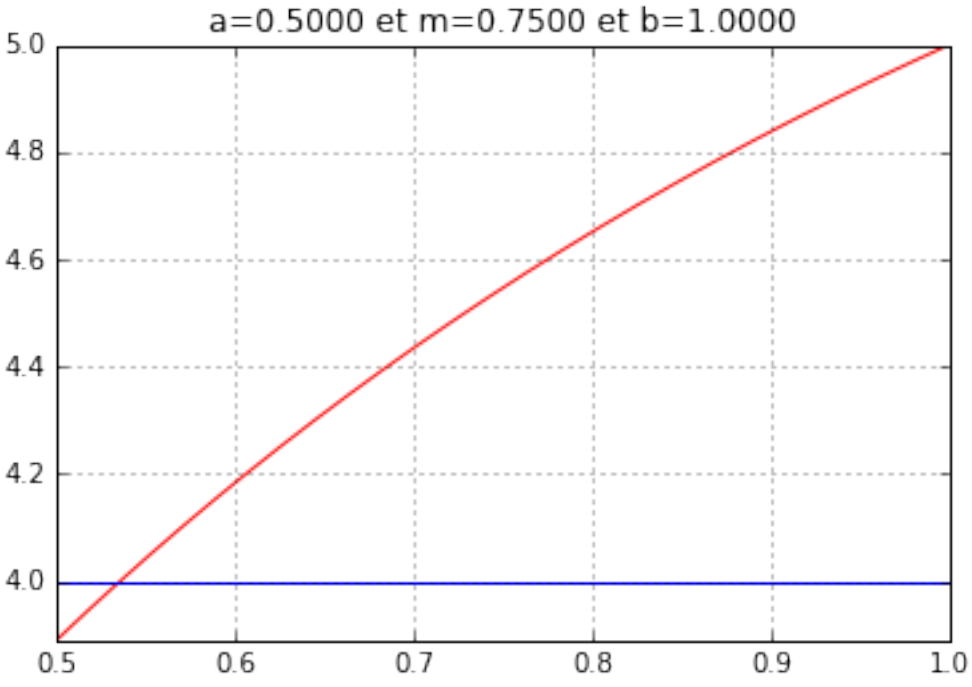
0.5.1 D'abord on s'arrete lorsque l'amplitude de l'intervalle $[a, b]$ est inférieure ou égale à $0,02$

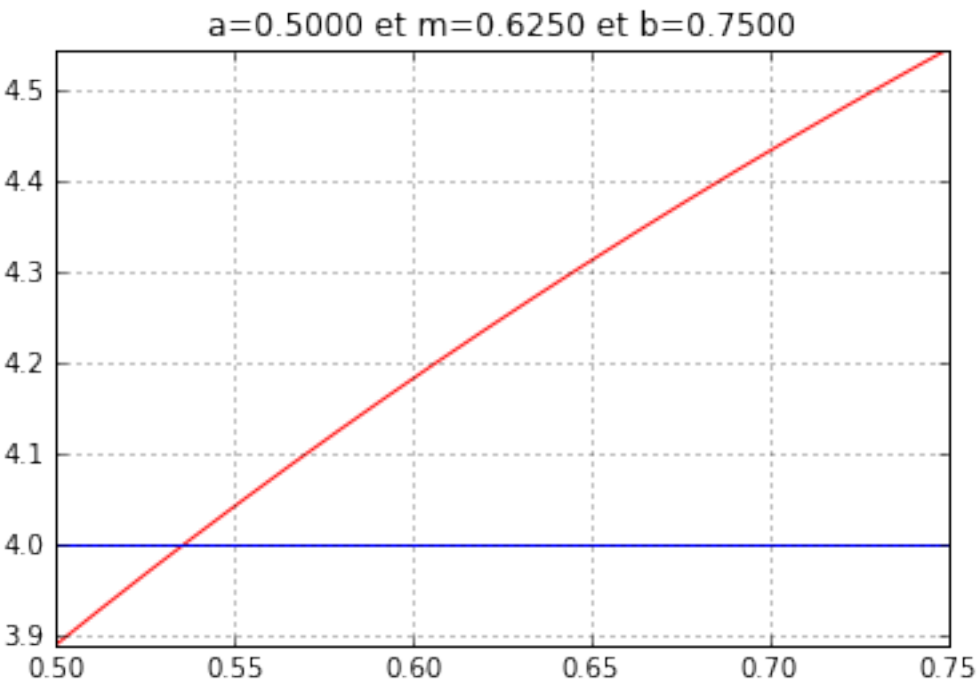
```
In [46]: dicho_tab(f, 0, 1, 0.02, 4)
```

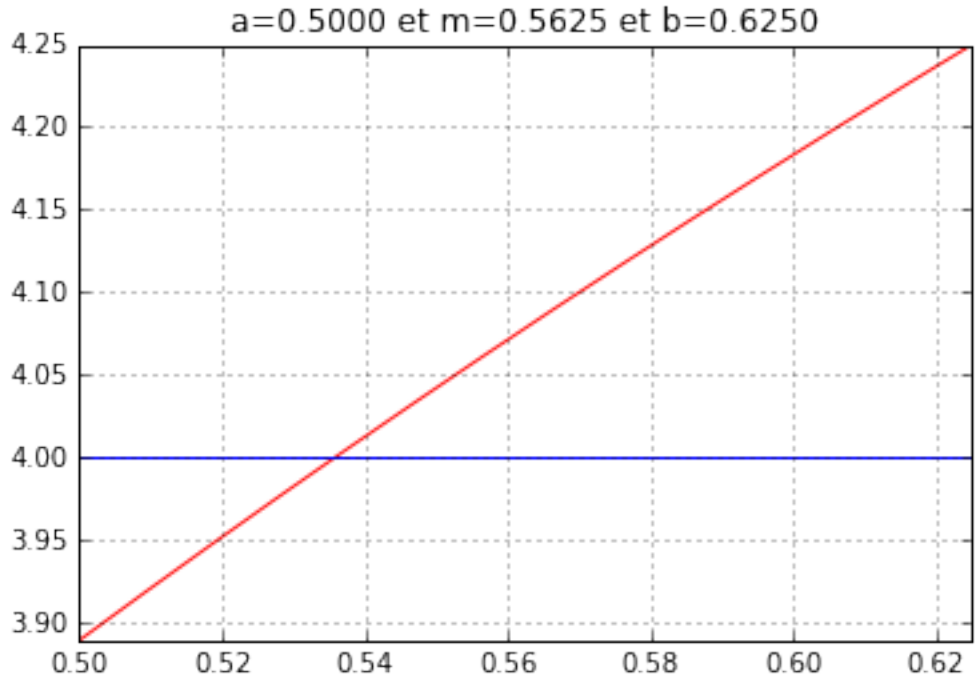
Etape	m	Choix ?	a	b
initialisation	None	None	0	1
1	0.5	droite	0.5	1
2	0.75	gauche	0.5	0.75
3	0.625	gauche	0.5	0.625
4	0.5625	gauche	0.5	0.5625
5	0.53125	droite	0.53125	0.5625
6	0.546875	gauche	0.53125	0.546875

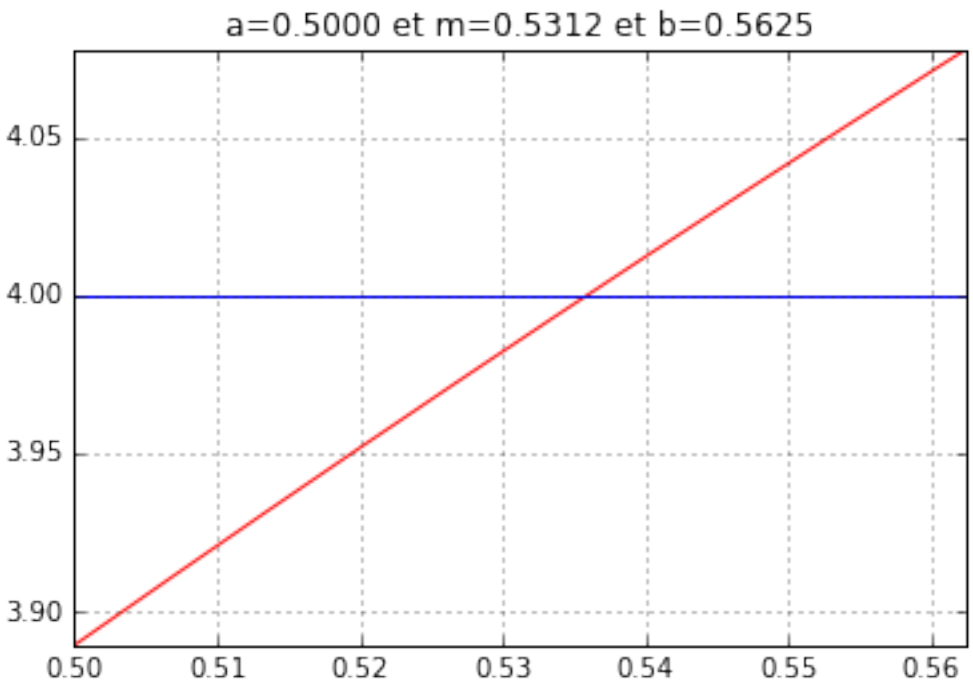
```
In [50]: dicho(f, 0, 1, 0.02, 4)
```

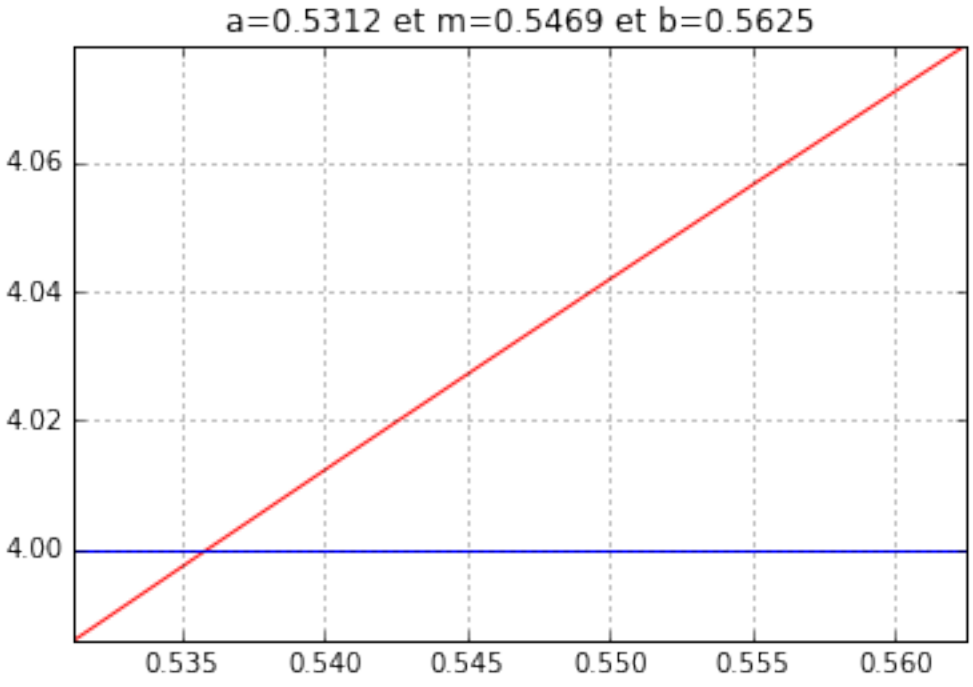

a=0.0000 et m=0.5000 et b=1.0000
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


a=0.5000 et m=0.7500 et b=1.0000
5.0
4.8
4.6
4.4
4.2
4.0
0.5
0.6
0.7
0.8
0.9
1.0


a=0.5000 et m=0.6250 et b=0.7500
4.5
4.4
4.3
4.2
4.1
4.0
3.9
0.50
0.55
0.60
0.65
0.70
0.75


a=0.5000 et m=0.5625 et b=0.6250
4.25
4.20
4.15
4.10
4.05
4.00
3.95
3.90
0.50
0.52
0.54
0.56
0.58
0.60
0.62





Out[50]:

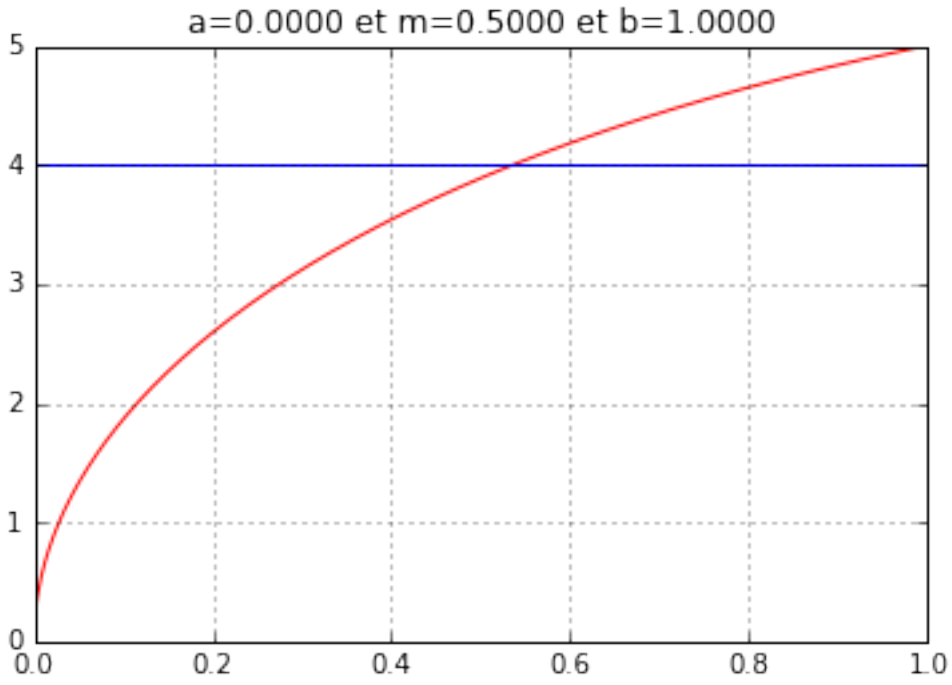
$$(0.53125, \quad 0.546875, \quad 6)$$

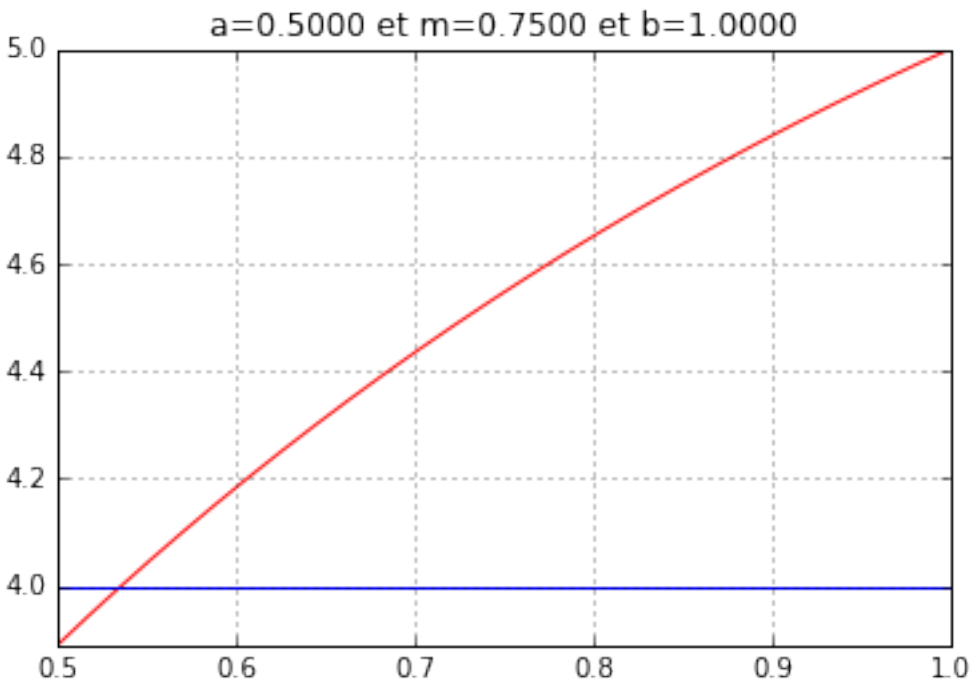
0.5.2 Ensuite on s'arrete lorsque l'amplitude de l'intervalle $[a, b]$ est inférieure ou égale à $0,002$

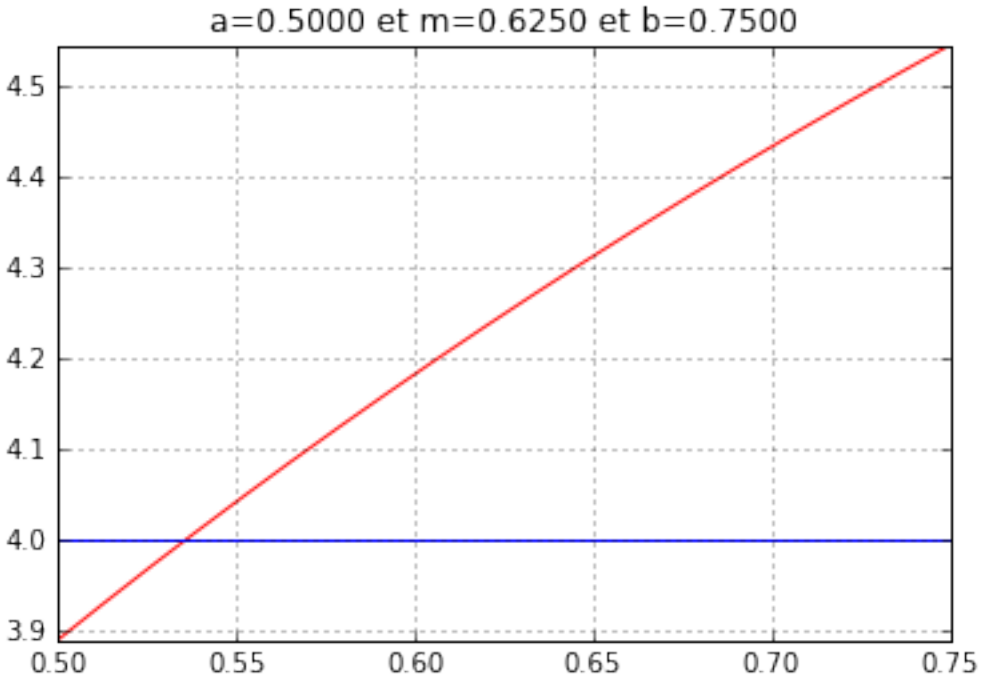
```
In [51]: dicho_tab(f, 0, 1, 0.002, 4)
```

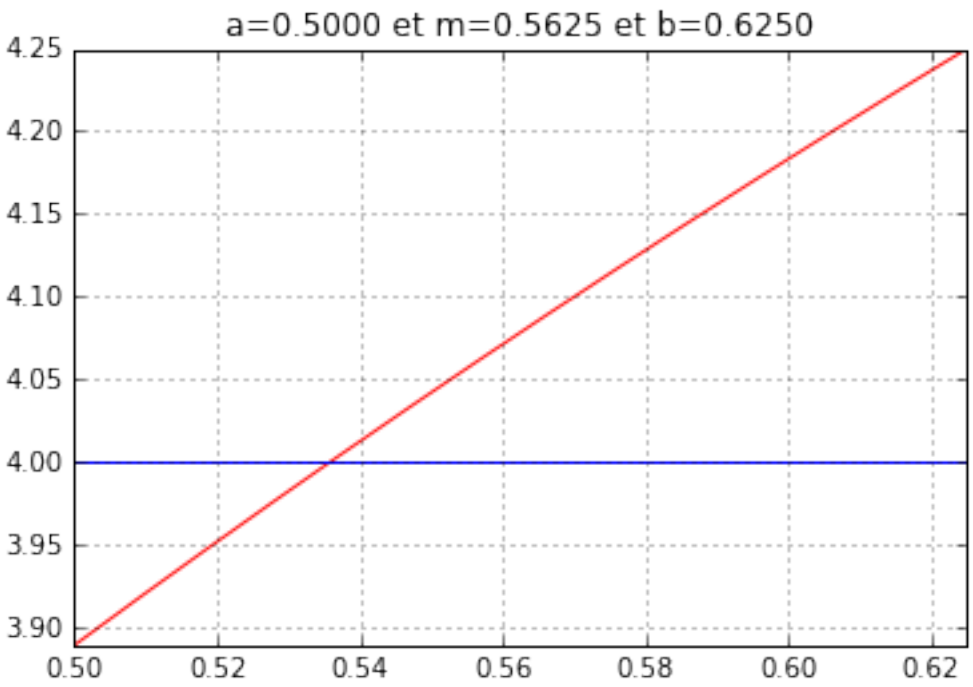
Etape initialisation	m None	Choix ? None	a 0	b 1
1	0.5	droite	0.5	1
2	0.75	gauche	0.5	0.75
3	0.625	gauche	0.5	0.625
4	0.5625	gauche	0.5	0.5625
5	0.53125	droite	0.53125	0.5625
6	0.546875	gauche	0.53125	0.546875
7	0.5390625	gauche	0.53125	0.5390625
8	0.53515625	droite	0.53515625	0.5390625
9	0.537109375	gauche	0.53515625	0.537109375

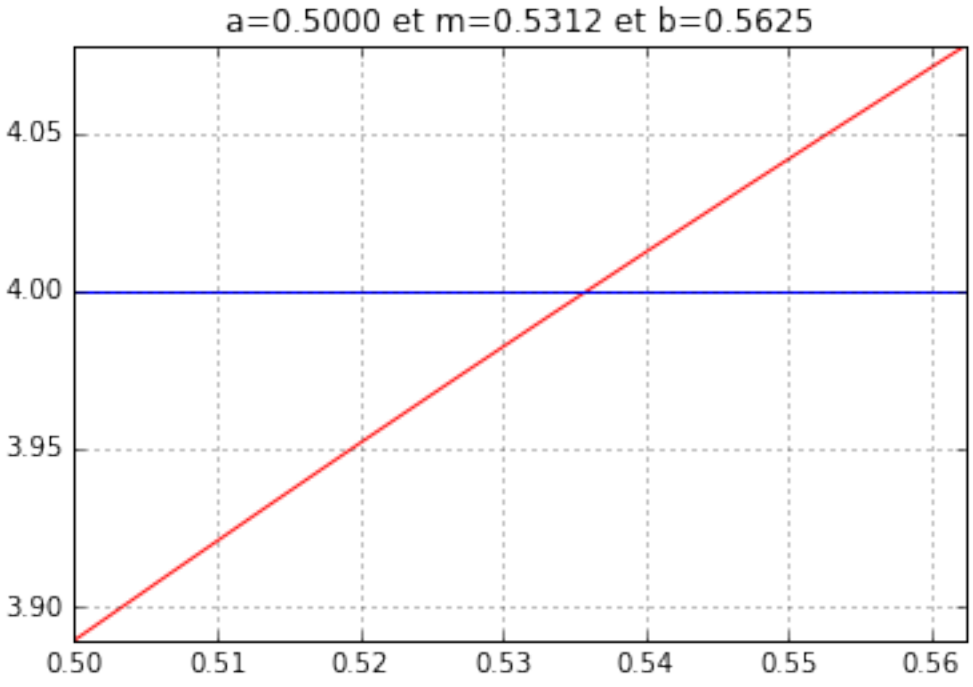
```
In [52]: dicho(f, 0, 1, 0.02, 4)
```

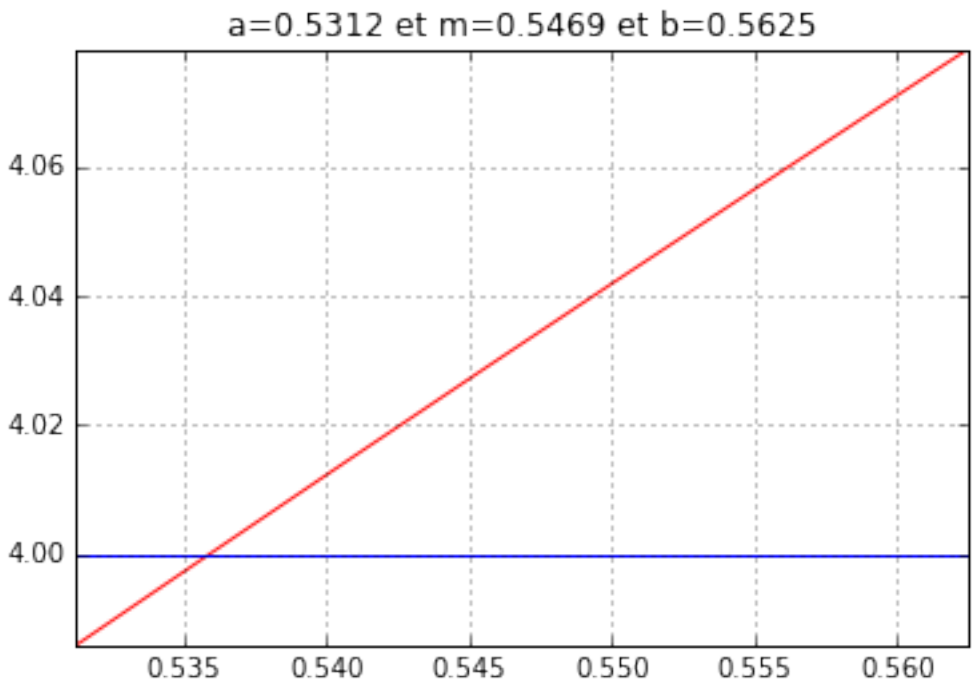



a=0.5000 et m=0.7500 et b=1.0000
5.0
4.8
4.6
4.4
4.2
4.0
0.5
0.6
0.7
0.8
0.9
1.0


a=0.5000 et m=0.6250 et b=0.7500
4.5
4.4
4.3
4.2
4.1
4.0
3.9
0.50
0.55
0.60
0.65
0.70
0.75


a=0.5000 et m=0.5625 et b=0.6250
4.25
4.20
4.15
4.10
4.05
4.00
3.95
3.90
0.50
0.52
0.54
0.56
0.58
0.60
0.62


a=0.5000 et m=0.5312 et b=0.5625
4.05
4.00
3.95
3.90
0.50
0.51
0.52
0.53
0.54
0.55
0.56



Out[52]:

$$(0.53125, \quad 0.546875, \quad 6)$$

In [53]:
```
n = 0
while 1/2**n > 0.002:
    n += 1
print(n)
```

```
9
```

L'intervalle de départ a pour amplitude 1 ($a_0 = 0$ et $b_0 = 1$) A chaque étape l'amplitude de l'intervalle de recherche est divisée par 2 Au bout de n étapes, l'amplitude de la zone de recherche est de $\frac{b_0 - a_0}{2^n}$ soit $\frac{1}{2^n}$ ici. Avec la calculatrice (voir ci-dessus ou le logarithme népérien) on peut vérifier que le plus petit entier n tel que $\frac{1}{2^n} \leqslant 0,002$ est $n = 9$