TD 14 -
Recherche de racines d'une fonction.
Recherche par dichotomie.
Méthode de Newton
Méthode de Babylone

Informatique
MPSI-PCSI - Lycée Thiers

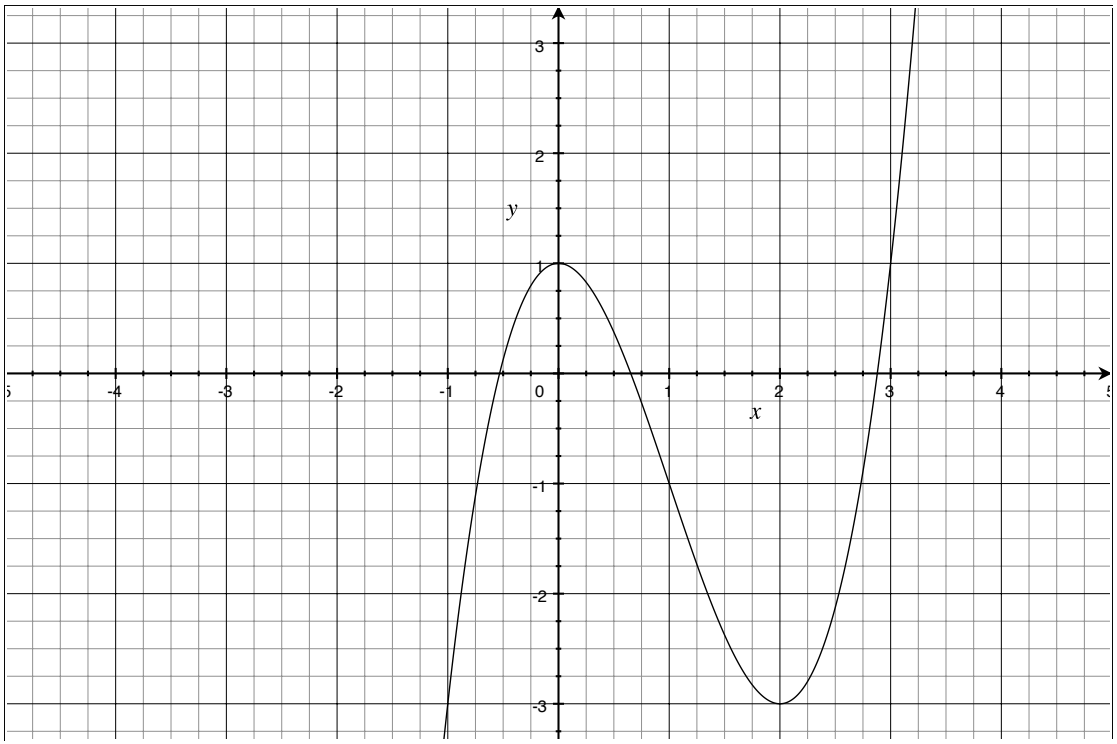
Exercice 2

Soit la fonction

$$f : x \mapsto x^3 - 3x^2 + 1$$

1. Vérifier que $f(0) = 1 > 0$ et $f(2) = -3 < 0$, et en déduire l'existence d'une solution unique dans $[0, 2]$ à l'équation $f(x) = 0$.
2. Tracer la courbe représentative de la fonction dans le plan muni d'un repère orthonormé.
3. Ecrire une fontion `dich_solve()` prenant en paramètre une fonction f, un entier n, des réels (float) `a<b` tels que `f(a)f(b)<0` et qui retourne une valeur approchée à 10^{-n} près de la solution. La recherche s'effectuera *par dichotomie sur l'intervalle* `[a,b]`.
4. Combien de passages dans la boucle `while` sont nécessaires pour obtenir une valeur approchée à 10^{-5} près ?
5. Essayer avec une précision de 10^{-15}, puis de 10^{-16}. Expliquer le comportement obtenu.

Exercice 1



1) L'application est polynomiale et donc dérivable sur $\mathbb{R}$, de dérivée $f' : x \mapsto 3x(x-2)$ qui prend des valeurs strictement négatives dans $]0,2[$, l'application est donc strictement monotone sur $[0,2]$, et réalise donc une bijection de $[0,2]$ sur $f([0,2]) = [-3,1]$. Elle y admet donc une unique racine.

Exercice 1

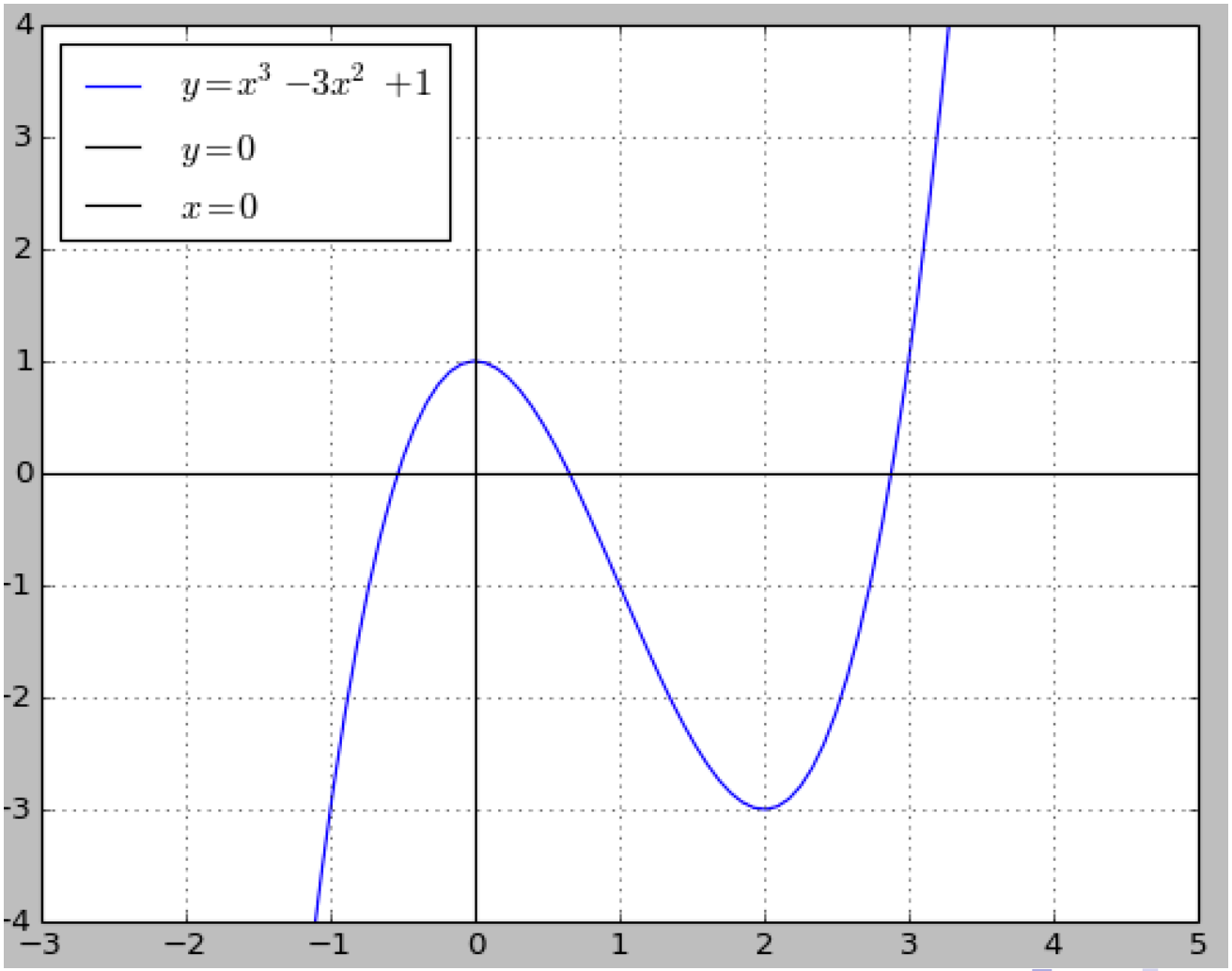
1) Tracé "enjolivé" de la courbe.

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

f = lambda x : x**3-3*x**2+1 # Définition fonction f
A, B = -3,5 # Délimitation abscisses du tracé
X = np.linspace(A,B,100)
Y = f(X)
plt.figure(1) # Ouverture figure 1
plt.clf() # Effacement figure 1
plt.plot(X,Y) # Tracé courbe représentative
plt.axhline() # Tracé axe des abscisses
plt.axis([A,B,-(B-A)/2,(B-A)/2]) # Fenêtre du tracé
plt.axvline() # Tracé axe des ordonnées
plt.legend(('$y=x^3-3x^2+1$','$y=0$','$x=0$'),'upper left') #
Légende
plt.grid() # Afficher une grille
plt.show()
```

Exercice 1

Ce qui produit le graphique :



Exercice 1

3)

```
def dich_solve(f,a,b,n):
    e = 10**-n
    while (b-a)/2 >= e:
    # Tant que précision non atteinte
        m=(a+b)/2                    # Prendre le milieu de [a,b]
        if f(a) * f(m) <= 0
                a, b = a, m          # dichotomie à gauche
            else:
                a, b = m, b          # dichotomie à droite
    return a,b                     # Retourner un encadrement
```

```
>>> dich_solve(f,0,2,5)    # Dichotomie dans [0,2] à 10**-5 près
(0.6527023315429688, 0.6527099609375)
```

Exercice 1

3) La question devient : combien de fois faut-il découper en deux parties égales un intervalle d'amplitude 2 pour obtenir des intervalles d'amplitude 10^{-5}.

Après n dichotomie l'amplitude de l'intervalle est :

$$2 \times \left(\frac{1}{2}\right)^n = 2^{1-n}$$

On cherche donc le plus petit entier n tel que $2^{1-n} \leqslant 10^{-5}$:

$$\begin{aligned}
&\iff \exp((1-n)\ln(2) \leqslant \exp(-5\ln(10)) \\
(\exp \text{ est croissante}) &\iff (1-n)\ln(2) \leqslant -5\ln 10 \\
&\iff (n-1)\ln(2) \geqslant 5\ln(10) \\
(\ln(2) > 0) &\iff n-1 \geqslant 5\frac{\ln(10)}{\ln(2)} \; n \geqslant 1 + 5\log_2(10)
\end{aligned}$$

Le plus petit entier n vérifiant cela est $\lceil 1 + 5\log_2(10) \rceil$. Soit 18 itérations :

```
>>> from math import log, ceil
>>> ceil(1+5*log(10,2))
18
```

Exercice 1

Plus généralement pour une précision à 10^{-n}, mathématiquement il faut ici $1 + n\log_2(10)$ itérations. (Informatiquement la précision a pour limite la représentation des flottants).

4.

```
>>> dich_solve(f,0,2,15)
(0.652703644666139, 0.652703644661398)
```

Encadrement de la racine sur $[0,2]$ à 10^{-15} près.

Pour une valeur approchée à 10^{-16} près, le programme boucle infiniment. En effet entre 0,5 et 1 (où se situe la racine), le plus petit écart entre deux nombres est 2^{-53} (en flottant 64 bits), soit, en notation scientifique :

```
>>> 2**-53
1.110223024625156e-16
```

C'est pourquoi une précision de 10^{-16} (inférieure à 2^{-53}) ne peut être atteinte : lors de la dichotomie au bout d'un certain rang, le milieu `m` de `a` et `b` donne `a` (à cause de l'erreur d'approximation) et la boucle tourne infiniment avec des valeurs de `a` et `b` qui demeurent constantes et d'écart `b-a` supérieur à 10^{-16}.

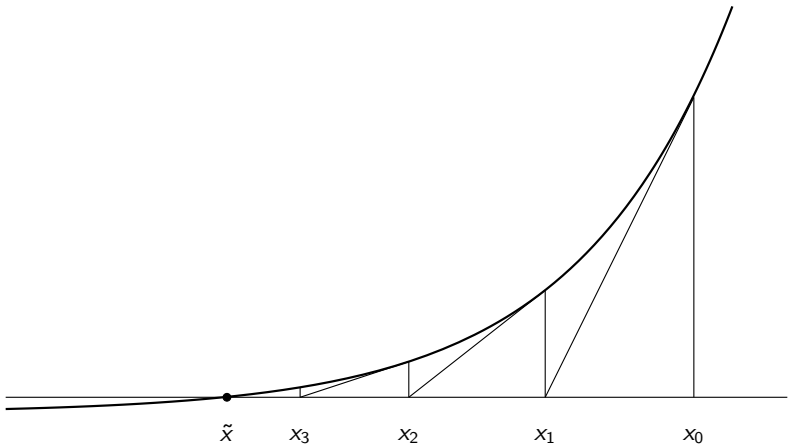
Exercice 2

La méthode de Newton permet de déterminer une valeur approchée d'une racine :

Théorème

Si $f : I \longrightarrow \mathbb{R}$ est de classe C^1 (dérivable à dérivée continue) et a une racine r sur I en laquelle $f'(r) \neq 0$, si x_0 est suffisamment proche de r, alors la suite :

(x_n) définie par $\forall\, n \in \mathbb{N}$, $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$ converge vers la racine r.



Méthode de Newton pour la recherche du zéro d'une application $f : \mathbb{R} \longrightarrow \mathbb{R}$.

Exercice 2

1. Ecrire une fonction `newton(f,df,x,n)` qui retourne la valeur approchée x_n de la racine de f.
2. On reprend la fonction
$$f : x \mapsto x^3 - 3x^2 + 1$$
de l'Exercice 1 (définie par : `f = lambda x: x**3-2*x**2+1` grâce à l'instruction `lambda`.)
Retrouver une valeur approchée de la racine de f sur $[0, 2]$ par la méthode de Newton en partant du point $x = 1$. Que se passe-t-il si l'on prend $x = 0$ ou 2 ?
3. Appliquer la méthode de Newton pour trouver une valeur approchée de la racine à 10^{-5} près. Sous l'hypothèse de monotonie de la fonction, on peut prendre comme critère d'arrêt :
$$f(u_n - 10^{-5}) \times f(u_n + 10^{-5}) \leqslant 0$$
4. Comparer les temps d'exécution pour la recherche d'une racine à 10^{-5} près de f par les deux méthodes de dichotomie et de Newton.

Exercice 2

1)

```
def newton(f,g,x,n):
    for i in range(n):
        x=x-f(x)/g(x)
    return x
```

2)

```
>>> f = lambda x : x**3-3*x**2+1
>>> df = lambda x : 3*x**2-6*x
>>> newton(f,df,1,5)
0.6527036446661393
```

En partant de 0 ou de 2, le calcul de la méthode de Newton provoque une erreur de division par 0. En effet en 0 et 2 la tangente à la courbe est horizontale : $f'(0) = f'(2) = 0$.

Exercice 2

3) Sur l'intervalle $[0,2]$ la fonction est monotone. En initiant la suite de la méthode de Newton, les termes de la suite restent bien dans l'intérieur de cet intervalle. Aussi pour déterminer une valeur approchée à 10^{-5} près on prend comme critère d'arrêt :

$$f(x-10^{-5}) \times f(x+10^{-5}) \leqslant 0$$

```
e = 1e-5
x = 1
compteur = 0
while f(x-e)*f(x+e)>0:
    x = x -f(x)/df(x)
    compteur += 1
print("Valeur approchée par Newton à 10-5 près", x)
print(compteur,"itérations")
```

On compte aussi le nombre d'itérations :

```
Valeur approchée par Newton à 10-5 près 0.652703646836132
3 itérations
```

4) On a effectué 3 itérations contre 18 pour une recherche par dichotomie !

Exercice 3

Exercice 1. *Extraction de racine par la méthode de Babylone*

Pour la résolution de $x^2 = a$ où $a > 0$, c'est à dire pour l'extraction de racines carrée cette méthode s'appelle la *méthode de Babylone* ou *Méthode de Héron d'Alexandrie*.

1. Appliquer la fonction `newton` pour le calcul approché de $\sqrt{2}$ avec plusieurs valeurs du paramètre `n` et comparer le résultat obtenu à la valeur réelle.
2. Ecrire une fonction `babylone(a,e)` qui retourne une valeur approchée à e près de $\sqrt{a}$. On appliquera la méthode de Newton en prenant comme point initial $u_0 = a$, et on essaiera de trouver un critère d'arrêt adéquat (la fonction $x^2 - a$ n'est monotone que sur $\mathbb{R}^+$.

Exercice 3

Ici on applique la méthode de Newton pour la recherche d'une racine de $f(x) = x^2 - a$:

$$u_{n+1} = u_n - \frac{f(u_n)}{f((u_n)} = u_n - \frac{u_n^2 - a}{2u_n} = \frac{u_n}{2} + \frac{a}{2u_n}$$

On démontre que si $u_0 > 0$ alors $(u_n)_{n\geqslant 1}$ est décroissante et à termes positifs.
Donc (Théorème de la limite monotone puis passage à la limite) :

si $u_0 > 0$ alors (u_n) converge vers $\sqrt{a}$.

```
def heron(a,n):
    x=a
    for i in range(n):
        x=0.5*(x+a/x)
    return x
```

```
>>> heron(2,5)
1.414213562373095
>>> 2**0.5
1.4142135623730951
```

Après 5 itérations tous les 16 chiffres significatifs sont exacts ! !

Exercice 3

2) La fonction $f(x) = x^2 - a$ n'est pas monotone, mais elle est strictement croissante sur $\mathbb{R}^+$. Aussi on peut prendre comme critère d'arrêt :

$$f(\max(x-e,0)) \times f(x+e) \leqslant 0$$

```
def babylone(a,e):
    f = lambda x : x**2-a
    u = a
    while f(max(0,u-e)) * f(u+e) > 0:
        u = u/2 + a/2/u
    return u
```