Entrée[2]:
```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

TP Python 11 : Méthodes de balayage et de dichotomie

Correction

Partie 1 : Objectifs

On considère une fonction $f : \mathcal{D}_f \to \mathbb{R}$. On s'intéresse à l'équation :

$$(\mathcal{E}) : \quad f(x) = 0.$$

- On appelle **zéro d'une fonction** f une solution de l'équation $f(x) = 0$
- L'objectif de cette séance est de construire des algorithmes permettant d'**approcher autant qu'on le souhaite** le(s) zéro(s) d'une fonction.
- Nous allons étudier deux méthodes, qui sont des méthodes dites **itératives** : on répète l'algorithme plusieurs fois de suite, jusqu'à ce qu'un **critère d'arrêt** soit vérifié.
- Dans le cas de la recherce de zéros d'une fonction, on souhaite s'arrêter **lorsqu'on s'approche de la solution à une précision ε donnée.**

D'un point de vue pratique, une machine ne pouvant chercher simultanément deux solutions à la fois, on se placera dans des cas simples :

On suppose par la suite que la fonction f considérée admet **un** zéro sur un intervalle I **donné**

Partie 2 : Algorithme de balayage

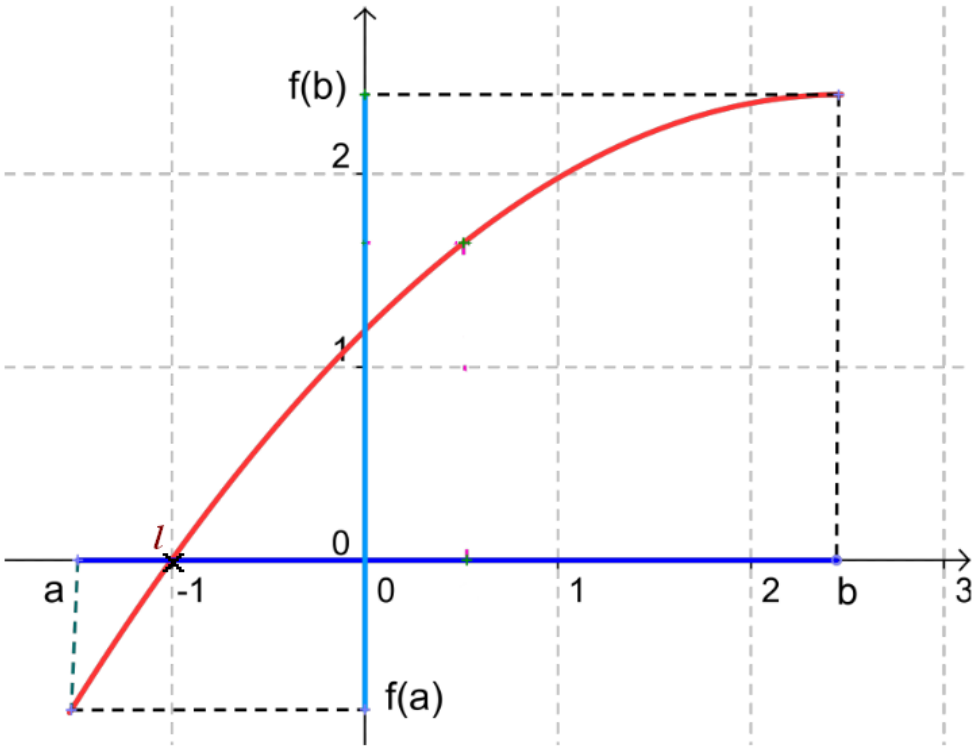
II.1) Préliminaires

Soient a et b deux nombres réels tels que $a < b$.

Soit f une fonction continue et strictement monotone sur l'intervalle $[a; b]$ telle que $f(a) \times f(b) < 0$.

Le théorème de la bijection permet d'affirmer qu'il existe un unique nombre réel $l \in [a; b]$ tel que $f(l) = 0$.

La méthode par balayage permet de déterminer un encadrement du nombre l à 10^{-n} près, où n est un nombre entier naturel fixé.



II.2) Principe de la méthode

On va chercher un encadrement du nombre $\sqrt{10}$.

1. Justifier que la fonction f, définie sur l'intervalle $[3;4]$ par $f(x) = x^2 - 10$, vérifie les conditions citées précédemment.
2. Indiquer la solution exacte l, sur l'intervalle $[3;4]$, de l'équation $f(x) = 0$.

À vous de faire 1 :

1. Implémenter une fonction Python `f` qui, étant donné un réel x en argument, renvoie la valeur $x^2 - 10$.
2. Tracer la courbe représentative de la fonction f sur l'intervalle $[2;5]$ pour plus de visibilité. Sur le même graphique, représenter la droite d'équation $y = 0$.

Entrée[10]:

```
def f(x):
    return x**2-10

X = np.linspace(2,5,100)
Y= []
Z= []
for k in range(len(X)):
    Y.append(f(X[k]))
    Z.append(0)

plt.clf()
plt.plot(X,Y)
plt.plot(X,Z)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

Comme $3^2 < 10 < 4^2$, on peut affirmer que $l = \sqrt{10} \in]3;4[$.

On peut le vérifier car $f(3) = -1$ et $f(3) < 0$ alors que $f(4) = 6$ et $f(4) > 0$.

La méthode, dite par balayage, est basée sur le principe suivant :

On commence par balayer l'intervalle $[3;4]$ avec un pas égal à 0.1, c'est-à-dire que l'on calcule $f(3)$, $f(3.1)$, $f(3.2)$, ... et on s'arrête dès que l'on a trouvé le chiffre p pour lequel $f(3+p\times 0.1)$ et $f(3+(p+1)\times 0.1)$ sont de signes opposés.

L'entier p représente le nombre d'**itérations** effectuée par la méthode de balayage avec ici un pas de 0.1.

À vous de faire 2 :
Écrire un programme qui détermine l'entier naturel p puis affiche p et un encadrement de $\sqrt{10}$ à 10^{-1} près.

Remarque On peut utiliser la fonction `print` de manière un peu sophistiquée, avec par exemple la commande suivante :

```
print("Après ",p," itérations, on obtient une approximation à 0.1 près de racine de 10, qui est comprise entre",3+p*pas,"et",3+(p+1)*pas)
```

Entrée[4]:

```
pas=0.1
a = 3
p = 0
while f(a)*f(a+pas)>0:
        a=a+pas
        p = p+1

print("Après ",p," itérations, on obtient une approximation à 0.1
```

```
Après  1  itérations, on obtient une approximation à 0.1 près de ra
cine de 10, qui est comprise entre 3.1 et 3.2
```

On poursuit ensuite le balayage de l'intervalle $[3,1\ ;\ 3,2]$ avec un pas égal à $0,01$ **afin de raffiner l'encadrement de** $\sqrt{10}$**.**

À vous de faire 3 :
Écrire un programme qui détermine la nouvelle valeur de l'entier naturel p puis affiche un encadrement de $\sqrt{10}$ à 10^{-2} près.

Entrée[5]:
```
pas=0.01
a = 3
p = 0
while f(a)*f(a+pas)>0:
        a=a+pas
        p = p+1

print("Après ",p," itérations, on obtient une approximation à 0.0
```

```
Après  16  itérations, on obtient une approximation à 0.01 près de
racine de 10, qui est comprise entre 3.16 et 3.17
```

Finalement, l'idée est la même quel que soit la **précision** que l'on souhaite obtenir pour l'encadrement de $\sqrt{10}$!

À vous de faire 4 :

1. Écrire **une fonction** `balayage` qui prend en argument le point de départ du balayage $a \in \mathbb{R}$ ainsi que l'ordre de précision $n \in \mathbb{N}$. Cette fonction doit déterminer un encadrement de $\sqrt{10}$ à 10^{-n} près, et afficher le nombre d'itérations nécessaires à l'obtention de cet encadrement ainsi que l'encadrement en lui-même.
2. Testez votre fonction pour $a = 3$ et $n = 1$, ainsi que $a = 3$ et $n = 1$ afin de vous assurez que votre programme fonctionne.
3. Testez votre fonction pour $a = 3$ et $n = 3$, puis $n = 4$, $n = 5$, $n = 6$, $n = 7$ et $n = 8$. À partir de quelle valeur de n l'ordinateur commence-t-il a prendre plus qu'une fraction de seconde pour effectuer son calcul? *On veillera à ne pas dépasser* $n = 8$*, pour le bien-être de votre machine...*

Entrée[38]:

```
def balayage(a,n):

    pas = 10**-n
    p = 0

    while f(a)*f(a+pas)>0:
        a=a+pas
        p = p+1

    print("Après ",p," itérations, on obtient une approximation à

    return

#balayage(3,2)
#balayage(3,1)
#balayage(3,3)
#balayage(3,8)
```

Partie 3 : Algorithme de dichotomie

III.1) Préliminaires

La méthode de la dichotomie est une autre méthode itérative de recherche des zéros d'une fonction. *Dichotomie* vient du grec *dikhotomia*, ou "division en deux parties".

Supposons qu'on dispose à nouveau d'une fonction f est continue sur l'intervalle $[a, b]$, avec $f(a) \leq 0$ et $f(b) \geq 0$. On sait donc qu'il existe au moins un réel c dans l'intervalle $[a, b]$ tel que $f(c) = 0$ par le théorème des valeurs intermédiaires.

L'idée est alors d'évaluer ce que vaut f au milieu de $[a, b]$, et de distinguer les deux cas suivants :

- si $f\left(\frac{a+b}{2}\right) \leq 0$, alors on sait qu'on a une racine dans l'intervalle $\left[\frac{a+b}{2}, b\right]$.
- sinon, $f\left(\frac{a+b}{2}\right) > 0$ et on sait qu'on a une racine dans l'intervalle $\left[a, \frac{a+b}{2}\right]$.

Ainsi, dans les deux cas, on a trouvé un intervalle de longueur moitié dans lequel est située une racine de l'équation $f(x) = 0$. On recommence alors avec cet intervalle, et ainsi de suite jusqu'à ce qu'on trouve une approximation qui nous convienne.

Formellement, on définit les suites (a_n) et (b_n) en posant

- $a_0 = a$ et $b_0 = b$.
- si $f\left(\frac{a_n+b_n}{2}\right) \leq 0$, alors $a_{n+1} = \frac{a_n+b_n}{2}$ et $b_{n+1} = b_n$.
- sinon, $a_{n+1} = a_n$ et $b_{n+1} = \frac{a_n+b_n}{2}$.

On a toujours une solution à l'équation $f(x) = 0$ dans l'intervalle $[a_n, b_n]$, qui est de longueur $(b - a)/2^n$.

À vous de faire 6 :
On cherche à approcher la solution de $f(x) = 0$ dans l'intervalle $[a, b]$ à une précision $\varepsilon > 0$ donnée.

Minorer le nombre d'étapes n à effectuer pour obtenir un encadrement de la solution à ε près.

→Cliquez-ici pour la réponse ←

III.2) Mise en pratique

À vous de faire 6 :

1. Toujours avec la même fonction $f : x \longmapsto x^2 - 10$, écrire une **fonction** `dichotomie` qui prend en argument les variables a et b représentant les bornes de l'intervalle, et la variable eps représentant la précision recherchée. Cette fonction doit **afficher** un encadrement de $\sqrt{10}$ à la précision donnée, ainsi que le **nombre d'itérations de la méthode**.
2. Quel est le type des variables a et b?
3. Quel est le type de la variable eps?
4. Testez votre fonction pour $a = 3$ et $b = 4$ lorsque $eps = 0.001$.

Entrée[35]:
```
def dichotomie(a,b,eps):
    it = 0
    while (b - a) > eps:
        c = (a + b) / 2  # Point médian
        if f(c) == 0:    # Si c est un zéro exact
              return

        elif f(a) * f(c) < 0:  # Le zéro est dans [a, c]
            b = c
        else:                  # Le zéro est dans [c, b]
            a = c
        it = it+1

    print("Après",it,"itérations, on obtiens un encadrement de ra
    return

dichotomie(3,4,0.001)
```

```
Après 10 itérations, on obtiens un encadrement de racine de 10 est
compris entre 3.162109375  et  3.1630859375  et on a bien un encadr
ement 0.0009765625 inférieur à 0.001
```

Partie 4 : Comparaison des deux algorithmes

À vous de faire 6 :
Comparez les deux méthodes en appelant votre fonction `balayage` et votre fonction `dichotomie` pour différentes valeurs de précision.

Attention : on veillera à bien faire la différence.

- Pour la méthode de balayage, on donne le **pas de discrétisation qui s'avère être aussi la précision**.
- Pour la méthode de dichotomie, **la précision correspond à la longueur de l'intervalle** obtenu!

Entrée[39]:
```
balayage(3,2)
dichotomie(3,4,0.01)

# Ce print n'est là que pour vous rendre la lecture des résultats
# Il s'agit d'un saut de ligne
print("\n")

balayage(3,5)
dichotomie(3,4,0.00001)
```

```
Après  16  itérations, on obtient une approximation à 0.01 près de
racine de 10, qui est comprise entre 3.16 et 3.17
Après 7 itérations, on obtiens un encadrement de racine de 10 est c
ompris entre 3.15625  et  3.1640625  et on a bien un encadrement 0.
0078125 inférieur à 0.01


Après  16227  itérations, on obtient une approximation à 1e-05 près
de racine de 10, qui est comprise entre 3.16227 et 3.16228
Après 17 itérations, on obtiens un encadrement de racine de 10 est
compris entre 3.1622772216796875  et  3.1622848510742188  et on a b
ien un encadrement 7.62939453125e-06 inférieur à 1e-05
```

La méthode de dichotomie est **bien plus puissante** que la méthode de balayage.

On dit que la **complexité** de la méthode de dichotomie est meilleure que celle de balayage : le nombre de calculs à effectuer pour la dichotomie est nettement inférieur au nombre de calculs à effectuer pour le balayage.