+1/1/60+

EPFL

1

Enseignant : Roger Sauser
ICS - CMS
9 avril 2025
Durée : 105 minutes

Dalton Joe

SCIPER : **987654**

Signature

Absent.e

Attendez le début de l'épreuve avant de tourner la page. Ce document est imprimé recto-verso et il contient 12 pages, les dernières pouvant être vides. Un total de 32 points (dont deux points de bonus) est réparti sur 7 questions.

- Posez votre **carte d'étudiant.e** sur la table, **vérifiez** votre nom et votre numéro SCIPER sur la première page et apposez votre **signature**.
- **Aucun** document n'est autorisé. L'utilisation d'une **calculatrice** et de tout outil électronique est interdite pendant l'épreuve.
- Pour les questions à **choix unique**, on comptera :
  les points indiqués si la réponse est correcte,
  0 point s'il n'y a aucune ou plus d'une réponse inscrite,
  0 point si la réponse est incorrecte.
- Utilisez un **stylo** à encre **noire ou bleu foncé** et effacez proprement avec du **correcteur blanc** si nécessaire. Toute réponse doit être rédigée en utilisant la place réservée à cet effet à la suite de la question. N'écrivez **pas dans les marges !**
- Si une question est erronée, l'enseignant se réserve le droit de l'annuler.
- Veuillez vous conformer aux indications suivantes pour les sujets qui demandent d'**écrire du code Python** (avec papier-stylo) :
  - respectez la syntaxe Python (parenthèses, crochets, deux points, mots-clés, etc.) ;
  - mettez en forme votre code pour qu'il soit formaté en vue d'une exécution sans erreur ;
  - respectez les indentations (en sachant que la taille de l'indentation n'importe pas en soi, mais qu'elle doit permettre d'identifier vos blocs de code de manière claire et immédiate).
- Les brouillons ne sont pas à rendre : ils ne seront pas corrigés. Ne pas dégrafer.

Respectez les consignes suivantes | Observe this guidelines | Beachten Sie bitte die unten stehenden Richtlinien		
choisir une réponse | select an answer Antwort auswählen	ne PAS choisir une réponse | NOT select an answer NICHT Antwort auswählen	Corriger une réponse | Correct an answer Antwort korrigieren
ce qu'il ne faut **PAS** faire | what should **NOT** be done | was man **NICHT** tun sollte		

Première partie, cinq questions à choix unique

Pour chaque question, marquez la case correspondante à la réponse correcte sans faire de ratures. Il n'y a qu'**une seule réponse correcte** par question. Cette première partie comprend un **total de 8 points + 2 points de bonus**.

Question 1 (2 points)
On cherche à déterminer une approximation des zéros d'une fonction réelle $f \in \mathcal{C}^2(\mathbb{R})$ d'une variable réelle à l'aide de la méthode de Newton.

Parmi les cinq affirmations suivantes laquelle est fausse ?

- ☐ A chaque itération, la méthode de Newton considère la tangente à la fonction f au point courant et le point courant suivant est pris égal au zéro de cette tangente.
- ☐ La méthode de Newton est une méthode de point fixe.
- ☐ La distance $|x_{k+1} - x_k|$ entre deux éléments successifs de la suite itérative définie par la méthode de Newton est inversément proportionnelle à la pente de f en x_k.
- ■ La méthode de Newton est une méthode itérative qui converge toujours avec un ordre 2.
- ☐ Avec la méthode de Newton, appliquée au voisinage d'un zéro simple ou double de f, l'erreur numérique absolue commise diminue à chaque itération.

Question 2 (2 points)
On applique la méthode de la bissection simple à la fonction $f(x) = x(x-1)(x+2)$ en prenant pour intervalle de départ l'intervalle $I = [-3, 2]$.

De quel zéro de f la méthode va-t-elle fournir une approximation ?

- ☐ La méthode de la bissection va fournir une approximation du zéro $\alpha = 1$.
- ☐ La méthode de la bissection ne va pas converger vers un zéro de f.
- ☐ La méthode de la bissection va fournir une approximation du zéro $\alpha = 0$.
- ■ La méthode de la bissection va fournir une approximation du zéro $\alpha = -2$.

Question 3 (2 points)
Parmi les quatre fonctions suivantes, laquelle admet au moins un point fixe dans l'intervalle $I = [1, 2]$?

- ☐ $f(x) = \sin(x)$
- ☐ $f(x) = 2.2$
- ☐ Aucune des quatre fonctions proposées
- ☐ $f(x) = -\frac{x}{3} + 3$
- ■ $f(x) = -x^2 + 2x + 1$

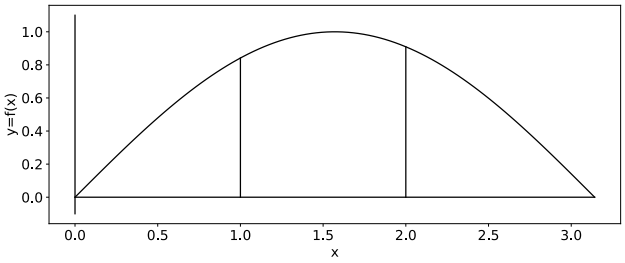
Question 4 (2 points, question bonus)
On cherche à déterminer, par itérations successives, une approximation de la racine cinquième du nombre 9, $\sqrt[5]{9} = 9^{1/5}$. En supposant que l'on se base sur la méthode de Newton, quelle est alors la relation, pour $k \geq 0$, qui permet de passer d'une approximation à la suivante ?

- ☐ $x_{k+1} = x_k^5 + x_k - 9$
- ☐ $x_{k+1} = \frac{4}{5}x_k - \frac{9}{5x_k^4}$
- ■ $x_{k+1} = \frac{4}{5}x_k + \frac{9}{5x_k^4}$
- ☐ $x_{k+1} = 5x_k - \frac{9}{x_k^4}$
- ☐ $x_{k+1} = 5x_k + \frac{9}{x_k^4}$
- ☐ $x_{k+1} = -x_k^5 + x_k + 9$

Question 5 (2 points)

On aimerait que le code Python ci-dessous à gauche produise la figure ci-dessous à droite.

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(12,12))
plt.xlabel('x', size=16)
plt.ylabel('y=f(x)', size=16)
#
# BLOC DE CODE MANQUANT
#
plt.xticks(size=16)
plt.yticks(size=16)
plt.axis('scaled')
plt.show()
```



Par quelles lignes de code doit-on remplacer le “BLOC DE CODE MANQUANT” pour que ce soit effectivement le cas ?

☐
```
x = np.linspace(0,np.pi,100)
plt.plot([x[0],x[-1]], [0,0], c='black')
plt.plot([0,0], [-0.1,1.1], c='black')
plt.plot(x,np.sin(x),c='black')
plt.plot([1,2], [np.sin(1),np.sin(2)], c='black')
```

■
```
x = np.linspace(0,np.pi,100)
plt.plot([x[0],x[-1]], [0,0], c='black')
plt.plot([0,0], [-0.1,1.1], c='black')
plt.plot(x,np.sin(x),c='black')
plt.plot([1,1], [0,np.sin(1)], c='black')
plt.plot([2,2], [0,np.sin(2)], c='black')
```

☐
```
def f(x):
    return np.sin(x)
x = np.linspace(0,np.pi,100)
plt.plot([x[0],x[-1]], [0,0], c='black')
plt.plot([0,0], [-0.1,1.1], c='black')
plt.plot(x,f(x),c='black')
plt.plot([1,1], [2,2], c='black')
plt.plot([0,f(1)], [0,f(2)], c='black')
```

☐
```
def f(x):
    return np.sin(x)
x = np.linspace(0,np.pi,5)
plt.plot([x[0],x[-1]], [0,0], c='black')
plt.plot([0,0], [-0.1,1.1], c='black')
plt.plot(x,f(x),c='black')
plt.plot([1,1], [0,f(1)], c='black')
plt.plot([2,2], [0,f(2)], c='black')
```

☐
```
def f(x):
    return np.sin(x)
x = np.linspace(0,np.pi,100)
plt.plot([x[0],x[-1]], [0,0], c='black')
plt.plot([0,0], [-0.1,1.1], c='black')
plt.plot(f(x),x,c='black')
plt.plot([1,1], [0,f(1)], c='black')
plt.plot([2,2], [0,f(2)], c='black')
```

Seconde partie, deux questions de “type ouvert”

Répondez dans l’espace dédié. Laissez libres les cases à cocher : elles sont réservées au correcteur. Cette seconde partie comprend un **total de 22 points**.

Question 6: *Cette question est notée sur 7 points.*

Le code reproduit dans le cadre suivant s’exécute sans erreur :

```
import numpy as np

a = np.array([[[0,1,2],[3,4,5],[6,7,8]],[[9,10,11],[12,13,14],[20,21,22]]])
x = np.linspace(-1,14,6)
r = np.arange(5)

print('===1===')
print(a.ndim)
print(a.shape)
print(x.ndim)
print(x.shape)
print(r.ndim==x.ndim and r.shape==x.shape)
print('===2===')
print(a[1,0,2])
print(a[1,:,0])
print('===3===')
print(x)
y = x[1:] - x[:-1]
print(y)
print('===4===')
print(r)
print(r**2-1)
```

(a) Ecrivez ci-dessous les résultats affichés suite à l’exécution de ce code, en respectant le contenu et la mise en forme de chaque affichage. Il n’est pas nécessaire de justifier vos réponses.

(b) Ecrivez également une ligne de code permettant de remplacer la séquence de nombres `[20,21,22]` du tableau `a` par la séquence `[15,16,17]`.

Solution

```
===1===
3
(2, 3, 3)
1
(6,)
False
===2===
11
[ 9 12 20]
===3===
[-1.  2.  5.  8. 11. 14.]
[3. 3. 3. 3. 3.]
===4===
[0 1 2 3 4]
[-1  0  3  8 15]
```

```
a[1,2,:] = [15,16,17]
```

Question 7: *Cette question est notée sur 15 points.*

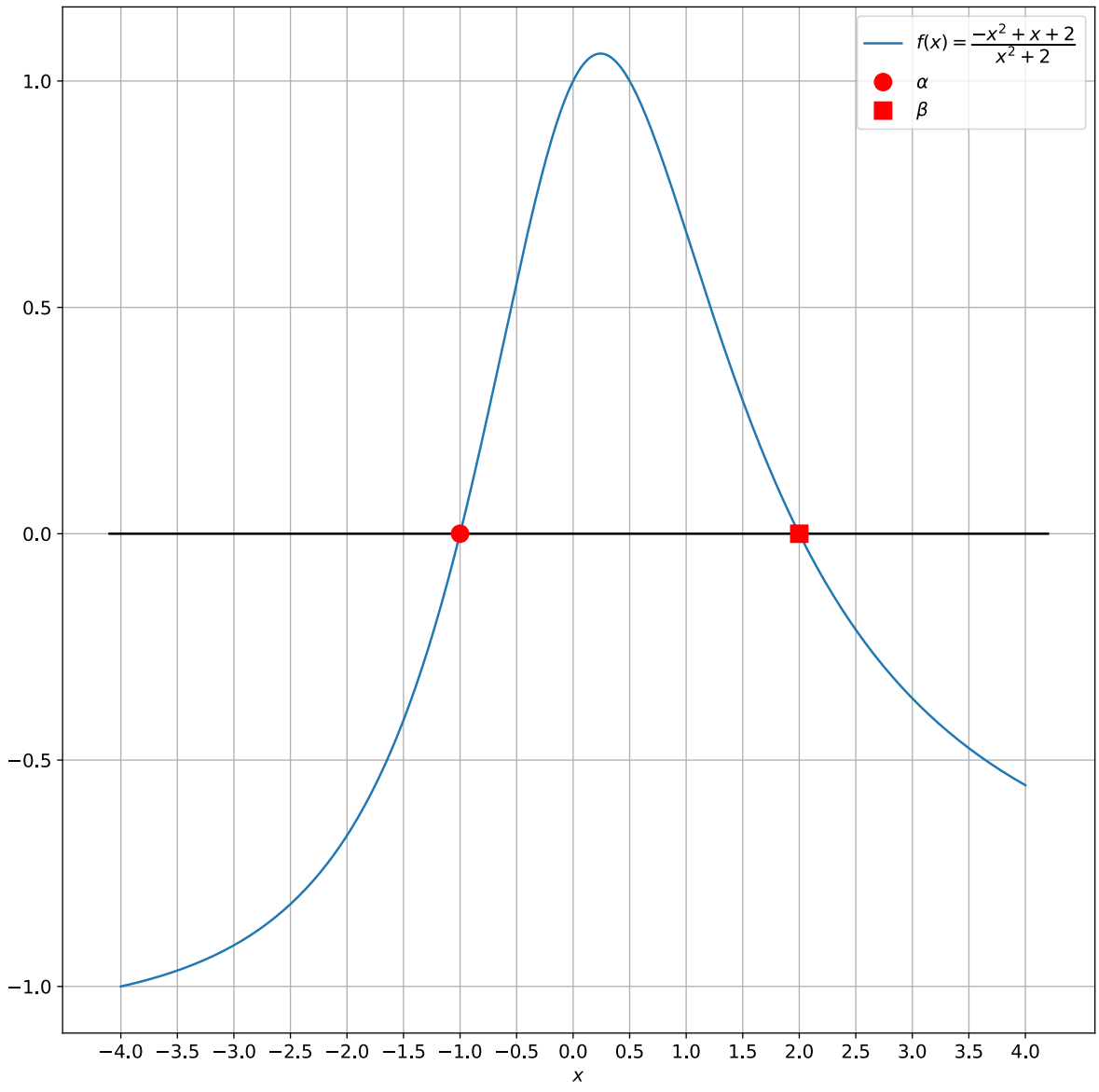
Soit la fonction réelle d'une variable réelle $f : \mathbb{R} \to \mathbb{R}$ définie par

$$f(x) = -\frac{(x+1)(x-2)}{x^2+2} = \frac{-x^2+x+2}{x^2+2}$$

dont la représentation graphique est donnée ci-dessous sur l'intervalle $I = [-4, 4]$.
On cherche à résoudre numériquement l'équation

$$f(x) = 0 \text{ dans l'intervalle } I,$$

c'est-à-dire que l'on cherche à trouver les deux valeurs α et β vérifiant $f(\alpha) = 0 = f(\beta)$.
Ces deux zéros de f sont indiqués sur la représentation graphique ci-dessous.



Il vous est demandé de déterminer une approximation . . .

(a) . . . de α grâce à la **méthode de la sécante** ;
(b) . . . de β grâce à la **méthode de point fixe** .

(a) **Approximation de α** (notée sur 4 points) **à l'aide de la méthode de la sécante**

Déterminez graphiquement la valeur approchée x_4 du zéro α obtenue après quatre itérations en appliquant la **méthode de la sécante** "à la main" en choisissant comme valeurs de départ $x_{-1} = -4$ et $x_0 = 0$. "A la main" signifie ici sans écrire de code destiné à être exécuté par un ordinateur, en "dessinant" au stylo (ou au crayon en prenant soin de bien appuyer sur la mine) sur la figure donnée en page 6 les différentes étapes de la méthode.

Précisez au mieux votre raisonnement à l'aide des traits de construction et indiquez sur la figure en page 6 la position des valeurs approchées x_1, x_2, x_3 et x_4 correspondant aux quatre premières itérations de la méthode.

(b) **Approximation de β** (notée sur 11 points) **à l'aide de la méthode de point fixe**

Pour déterminer la valeur (approchée) du zéro β, il vous est demandé d'appliquer la **méthode de point fixe** avec la fonction auxiliaire (d'itération)

$$\Phi(x) = \frac{3x + 2}{x + 2},$$

en procédant en deux parties.

Première partie

Commencez par vérifier que la fonction Φ donnée ci-dessus est une fonction d'itération acceptable pour la recherche du zéro β. Montrez en particulier que la méthode de point fixe va converger vers le zéro β au voisinage de β.

Seconde partie
Complétez le code Python en page suivante de manière à ce qu'il permette d'approcher numériquement la solution β de l'équation $f(x) = 0$ à l'aide de la **méthode de point fixe**.

Plus précisément, le code à compléter doit :

1. définir les fonctions `f` et `Phi` ;
2. définir une fonction Python nommée `mptfixe` permettant de trouver une solution approchée de l'équation $\Phi(x) = x$ par la méthode de point fixe. Cette fonction doit avoir comme paramètres (arguments) :
   - la fonction `Phi` pour laquelle on veut trouver un point fixe ;
   - le point de départ `x_0` de la méthode ;
   - la tolérance `eps` souhaitée ;
   - le nombre maximum d'itérations autorisées `k_max` .

   Cette fonction `mptfixe` doit implémenter la méthode de point fixe et itérer jusqu'à ce que l'une des deux conditions suivantes soit satisfaite :
   - soit la valeur absolue de l'incrément $|x_{k+1} - x_k|$ est inférieure à la tolérance `eps` ;
   - soit le nombre maximum d'itérations est atteint.

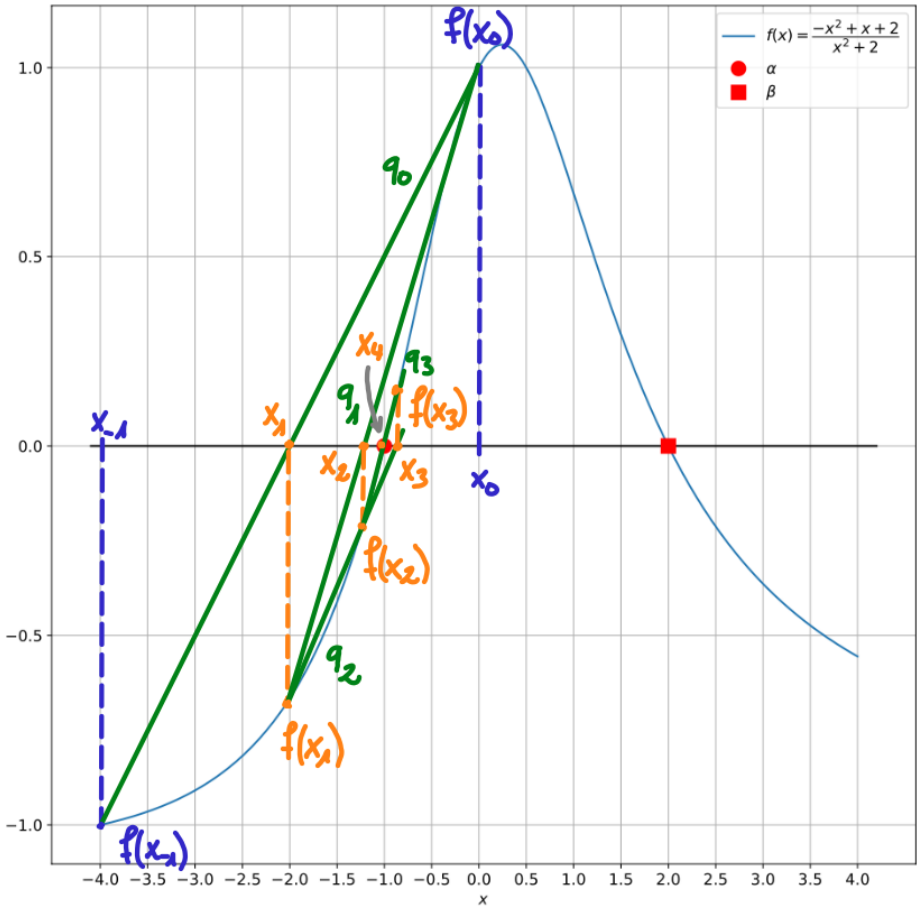
   Dans les deux cas, la fonction doit alors retourner la valeur approchée courante de la solution, une liste renfermant les valeurs de l'incrément $|x_{k+1} - x_k|$ à chaque itération, ainsi que le nombre d'itérations effectuées. Il n'est pas nécessaire de compléter la définition de la fonction avec une "docstring" ;
3. faire appel à la **fonction** `bisect` **de la librairie SciPy** pour trouver l'approximation du zéro β qui sera utilisée comme point de départ x_0 pour la fonction `mptfixe`. Les arguments de la fonction `bisect` doivent être choisis de manière appropriée et une tolérance de 10^{-2} doit être imposée ;
4. faire appel à la fonction `mptfixe` pour trouver une approximation du zéro β en demandant une tolérance de 10^{-10} et un nombre maximum d'itérations de 30 ;
5. afficher la valeur approchée trouvée, ainsi que le nombre d'itérations qui ont été nécessaires à la place des points d'interrogation dans le message suivant : "Valeur approchée trouvée après ? itérations : ? ." ;
6. représenter graphiquement simplement (sans "décorations" particulières) l'évolution de l'incrément $|x_{k+1} - x_k|$, $k \geq 0$, en fonction de l'itération $k + 1$, en veillant à ce que l'axe des ordonnées soit en échelle logarithmique.

(voir code à compléter en page suivante)

```
# DEBUT DU CODE A COMPLETER
# (veuillez respecter les notations spécifiées dans l'énoncé ci-dessus)
#
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize

# 1. Définitions des fonctions f et Phi
def f(x):
    return -(x+1)*(x-2)/(x**2+2)

def Phi(x):
    return (3*x+2)/(x+2)

# 2. Définition de la fonction Python mptfixe
def mptfixe(









# 3. Appel de la fonction bisect de SciPy pour trouver un bon point de départ



# 4. Appel de la fonction mptfixe
resultat = mptfixe(Phi,x_0,1e-10,30)

# 5. Affichage de l'approximation trouvée du zéro beta



# 6. Représentation de l'évolution de l'incrément



plt.yscale('log')
plt.show()

# FIN DU CODE A COMPLETER
```

Solution

(a)



$x_1 = -2$
$x_2 = -1.2$
$x_3 \cong -0.89$
$x_4 \cong -1.008$

(b) **Première partie**

Tout d'abord, on vérifie que les points fixes de la fonction d'itération

$$\Phi(x) = \frac{3x+2}{x+2}$$

correspondent bien aux zéros $\alpha = -1$ et $\beta = 2$ de la fonction

$$f(x) = -\frac{(x+1)(x-2)}{x^2+2} = \frac{-x^2+x+2}{x^2+2}$$

dont on cherche les racines :

- $\Phi(\alpha) = \Phi(-1) = \dfrac{3\cdot(-1)+2}{-1+2} = \dfrac{-1}{1} = -1 = \alpha$;
- $\Phi(\beta) = \Phi(2) = \dfrac{3\cdot 2+2}{2+2} = \dfrac{8}{4} = 2 = \beta$.

Pour s'assurer que la méthode de point fixe converge bien avec cette fonction d'itération, on évalue la dérivée de cette dernière en α et β. Comme

$$\Phi'(x) = \frac{3}{x+2} - \frac{3x+2}{(x+2)^2} = \frac{4}{(x+2)^2},$$

on a :

- $\Phi'(\alpha) = \Phi'(-1) = \dfrac{4}{1^2} = 4 > 1$;
- $\Phi'(\beta) = \Phi(2) = \dfrac{4}{4^2} = \dfrac{1}{4} < 1$.

Par le théorème d'Ostrowski, il existe donc un voisinage de la racine β dans lequel la méthode de point fixe converge vers β.

(b) **Seconde partie**

```
# DEBUT DU CODE A COMPLETER
# (veuillez respecter les notations spécifiées dans l'énoncé ci-dessus)
#
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize

# 1. Définitions des fonctions f et Phi
def f(x):
    return -(x+1)*(x-2)/(x**2+2)

def Phi(x):
    return (3*x+2)/(x+2)

# 2. Définition de la fonction Python mptfixe
def mptfixe(Phi,x_0,eps,k_max):
    iteration = 0
    condition = True
    liste = []
    while iteration < k_max and condition:
        x_1 = Phi(x_0)
        condition = abs(x_1-x_0) > eps
        liste.append(abs(x_1-x_0))
        x_0 = x_1
        iteration += 1
    return x_0,liste,iteration

# 3. Appel de la fonction bisect de SciPy pour trouver un bon point de départ
x_0 = optimize.bisect(f,0.5,3,xtol=1e-2)

# 4. Appel de la fonction mptfixe
resultat = mptfixe(Phi,x_0,1e-10,30)

# 5. Affichage de l'approximation trouvée du zéro beta
print(f'Valeur approchée trouvée après',resultat[2],'itérations:',resultat[0],'.')

# 6. Représentation de l'évolution de l'incrément
k = np.arange(1,len(resultat[1])+1)
plt.plot(k,resultat[1])
plt.yscale('log')
plt.show()

# FIN DU CODE A COMPLETER
```

Aide-mémoire (bibliothèques) Python)

NumPy

import numpy as np

np.linspace(start, stop, num=50, endpoint=True, retstep=False)
np.logspace(start, stop, num=50, endpoint=True)
np.arange(start, stop, step)
np.zeros(shape, dtype=float)
np.ones(shape, dtype=None)
np.empty(shape, dtype=float)
np.array(object, dtype=None)
np.empty_like(a)
np.zeros_like(a)
np.ones_like(a)
ndarray.ndim
ndarray.shape
ndarray.dtype
np.eye(N)
np.reshape(a, newshape)
np.dot(a, b)
a.T
a.transpose()
np.linalg.det(a)
np.linalg.inv(a)
np.linalg.eig(a)
np.random.rand(N)
np.meshgrid(x,y,indexing='ij')
np.gradient(f)
np.loadtxt(fname, comments='#', skiprows=0, usecols=None, unpack=False)
np.savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n', header=' ', footer=' ', comments='#')

SciPy

from scipy import constants
from scipy import optimize
from scipy import misc

constants.c
constants.m_e
constants.g
constants.physical constants["speed of light in vacuum"]
constants.physical constants["electron mass"]
constants.physical constants["standard acceleration of gravity"]
optimize.curve_fit(f, xdata, ydata)
optimize.bisect(f, a, b, xtol=2e-12, maxiter=100, full_output=False)
optimize.newton(func, x0, fprime=None, tol=1.48e-08, maxiter=50, fprime2=None, full_output=False)
optimize.fsolve(func, x0, xtol=1.49012e-08)
misc.derivative(func, x0, dx=1.0, n=1, order=3)

Matplotlib

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

plt.figure(num=None, figsize=None)
plt.plot(x, y)
plt.errorbar(x, y, yerr=None, xerr=None)
plt.scatter(x, y)
fig.suptitle(str)
ax = plt.subplot(m,n,a)
ax = plt.subplot()   (ou  ax = fig.gca())
ax = plt.subplot(projection='3d')   (ou  ax = fig.gca(projection='3d') )
ax.plot(x, y)
ax.barh(y, width)
ax.contour(x,y,z,levels)
ax.plot_surface(x,y,z)
ax.quiver(x,y,u,v)
ax.streamplot(x, y, u, v, linewidth=1, density = 2, arrowstyle = '->', arrowsize = 1.5)
ax.set_title(str)
ax.set_yticks(labels)
ax.set_xlabel(xlabel)
ax.set_ylabel(ylabel)
ax.set_zlabel(zlabel)
ax.imshow(x)
plt.clabel(cs)
plt.axis('equal')
plt.axis('scaled')
plt.xlabel(xlabel)
plt.ylabel(ylabel)
plt.legend()
plt.title(label)
plt.savefig(fname)
plt.imshow(x)
plt.show()
mpimg.imread(fname)
mpimg.imsave(fname)
```