+1/1/60+

EPFL

1

Enseignant : Roger Sauser
ICS - CMS
17 avril 2024
Durée : 105 minutes

RS

SCIPER : **22**

Attendez le début de l'épreuve avant de tourner la page. Ce document est imprimé recto-verso et il contient 12 pages qu'il ne faut pas désagrafer. Un total de 32 points (dont deux points de bonus) est réparti sur 7 questions.

- Posez **votre carte d'étudiant** sur la table.
- **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** ("multiple choice"), 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 !**
- 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 exactement comme si vous le tapiez 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).

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, deux questions de “type ouvert”

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

Question 1: *Cette question est notée sur 5 points.*

Le code reproduit dans le cadre suivant est extrait d’une cellule Jupyter Notebook et s’exécute sans erreur :

```
import numpy as np
m = np.array([[[0,1],[2,3]],[[4,5],[6,7]],[[8,9],[10,11]]])
v = np.linspace(-3,7,3)

print('===1===')
print(m.ndim)
print(m.shape)
print(v.ndim)
print(v.shape)
print('===2===')
print(m[2,1])
print(v)
print('===3===')
print(v+2)
print(m[:,0,0]*v)
print('===4===')
v[1:2] = 0.5
print(v)
w = v>0
print(w)
```

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.

Solution

```
===1===
3
(3, 2, 2)
1
(3,)
===2===
[10 11]
[-3.  2.  7.]
===3===
[-1.  4.  9.]
[-0.  8. 56.]
===4===
[-3.   0.5  7. ]
[False  True  True]
```

Question 2: *Cette question est notée sur 17 points.*

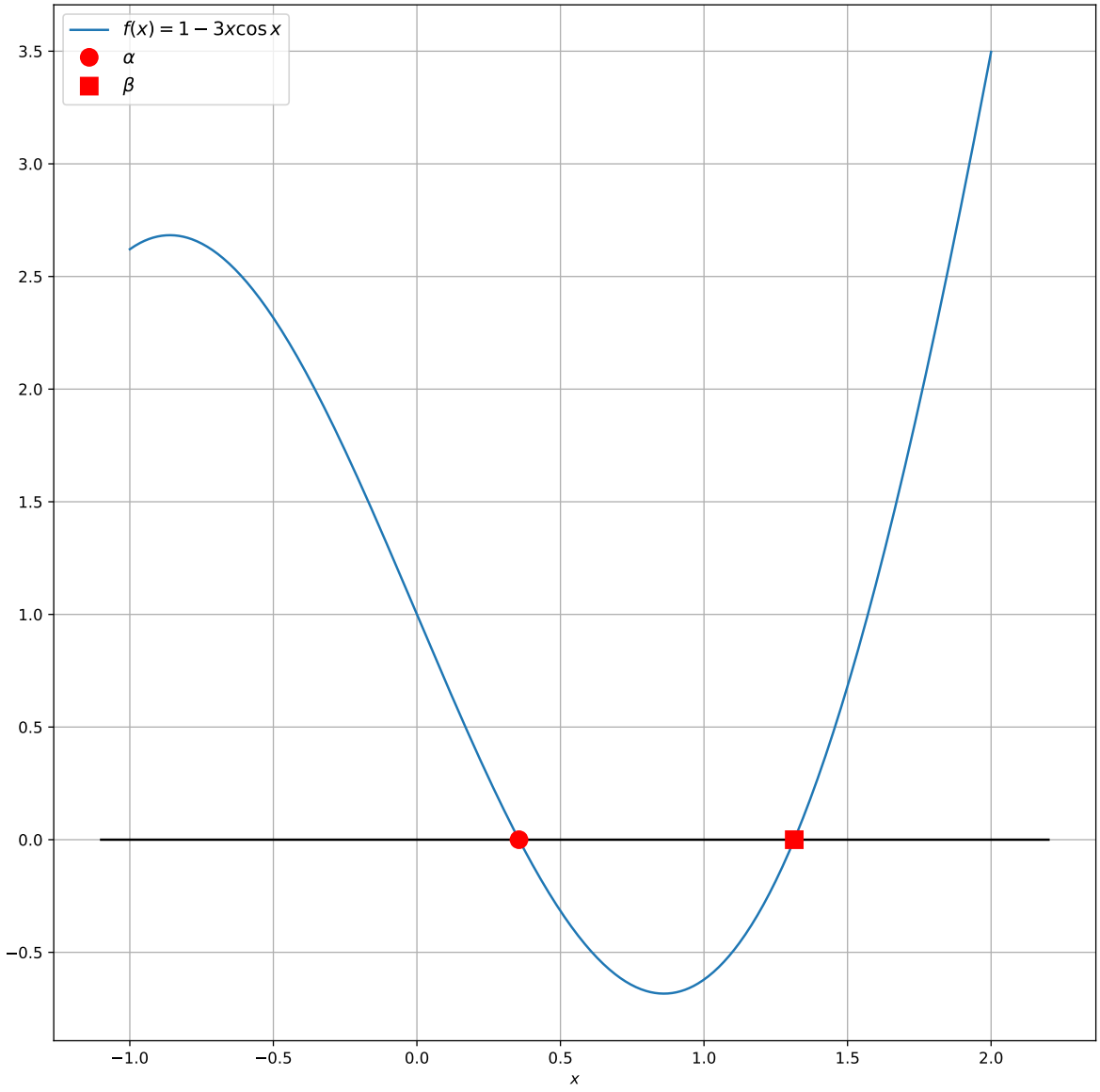
Soit la fonction réelle d'une variable réelle $f : \mathbb{R} \to \mathbb{R}$ définie par

$$f(x) = 1 - 3x \cos x$$

dont la représentation graphique est donnée ci-dessous sur l'intervalle $I = [-1, 2]$.
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 Halley** proposée par la fonction `newton` de la librairie SciPy ;
(b) ...de α grâce à la **méthode de la corde** ;
(c) ...de β grâce à la **méthode de Newton** .

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

Completez le code Python ci-dessous de manière à ce qu'il permette de déterminer (et d'afficher) une approximation du zéro α grâce à la **méthode de Halley proposée par la fonction** `newton` **de la librairie SciPy**. Le code doit faire en sorte que la fonction utilise la **méthode de Halley**. Il doit s'exécuter de manière indépendante et sans erreur. Choisissez la tolérance proposée par défaut par la fonction.

```
# DEBUT DU CODE A COMPLETER
#
import numpy as np


def f(x):
    return 1 - 3*x*np.cos(x)









# FIN DU CODE A COMPLETER
```

(b) **Approximation de α** (notée sur 10 points) **à l'aide de la méthode de la corde**

Completez le code Python en page suivante de manière à ce qu'il permette d'approcher numériquement la solution α de l'équation $f(x) = 0$ à l'aide de la **méthode de la corde**.

Plus précisément, le code à compléter doit :

1. définir la fonction `f` ;
2. définir une fonction Python nommée `mcorde` permettant de trouver une solution approchée de l'équation $f(x) = 0$ par la méthode de la corde. Cette fonction doit avoir comme paramètres (arguments) :
   - la fonction `f` pour laquelle on veut trouver un zéro ;
   - les extrémités `a` et `b` d'un "bon" intervalle de départ renfermant le zéro cherché ;
   - le point de départ `x_0` de la méthode ;
   - la tolérance `eps` souhaitée ;
   - le nombre maximum d'itérations autorisées `k_max`.

   Cette fonction `mcorde` doit implémenter la méthode de la corde en s'assurant que la valeur utilisée comme approximation de la pente à partir des extrémités `a` et `b` est supérieure à 10^{-6}. Une valeur inférieure doit conduire à une exception "ValueError" et à l'affichage "Attention : choix inadapté de l'intervalle !".

   La fonction doit 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` ; la fonction retourne alors la valeur approchée courante de la solution ainsi que le nombre d'itérations effectuées ;

- soit le nombre maximum d'itérations est atteint ; la fonction retourne alors la valeur approchée courante de la solution 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 `mcorde` pour trouver le zéro α de la fonction f définie ci-dessus en imposant une tolérance de 10^{-8} et un nombre maximum d'itérations de 50 ; l'intervalle $[a, b]$ et le point de départ x_0 doivent être choisis de manière appropriée ;
4. 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 : "Approximation de la racine : ? (obtenue après ? itérations)".

```
# DEBUT DU CODE A COMPLETER
# (veuillez respecter les notations spécifiées dans l'énoncé ci-dessus)
#
import numpy as np
# 1. définition de la fonction f
def









# 2. définition de la fonction Python mcorde
def mcorde

















    return

# 3. appel de la fonction mcorde pour trouver l'approximation cherchée
a =
b =
x_0 =
tolerance = 1e-8
k_max = 50



# 4. affichage de la valeur trouvée



# FIN DU CODE A COMPLETER
```

(c) **Approximation de β** (notée sur 3 points) **à l'aide de la méthode de Newton**

Déterminez la valeur approchée du zéro β obtenue après trois itérations en appliquant la **méthode de Newton** "à la main" en choisissant comme valeur de départ $x_0 = 1$. "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 4 les différentes étapes de la méthode.

Précisez au mieux votre raisonnement et indiquez sur la figure en page 4 la position des valeurs approchées x_1, x_2 et x_3 correspondant aux trois premières itérations de la méthode.

Solution

(a)

```
# DEBUT DU CODE A COMPLETER
#
import numpy as np
from scipy import optimize
def f(x):
    return 1 - 3*x*np.cos(x)

def fprime(x):
    return - 3*np.cos(x) +3*x*np.sin(x)

def fseconde(x):
    return +6*np.sin(x) + 3*x*np.cos(x)

alpha = optimize.newton(f,-0.5,fprime=fprime,fprime2=fseconde)
print(alpha)

# FIN DU CODE A COMPLETER
```

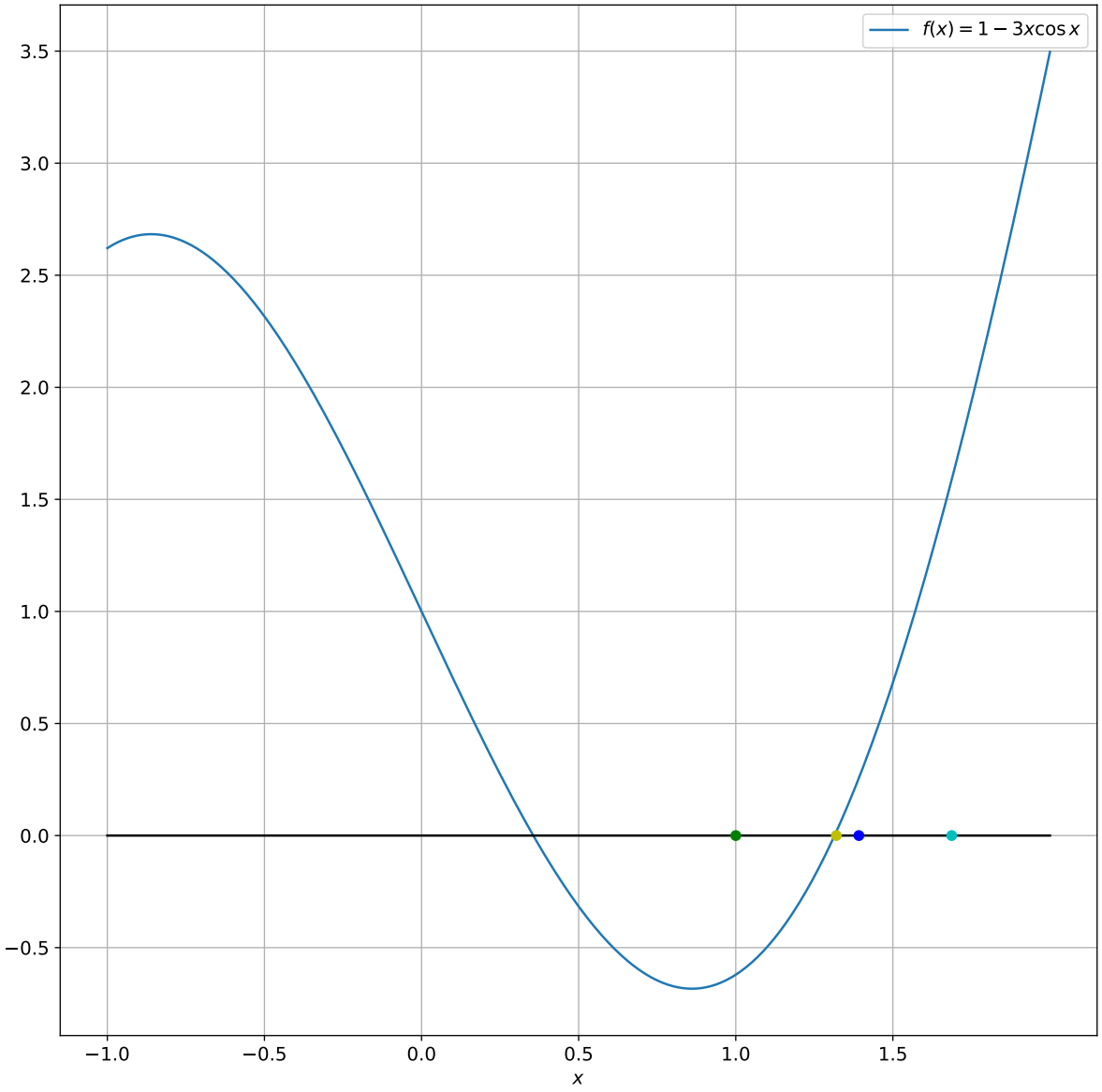
```
0.3555759893429734
```

(b)

```
# DEBUT DU CODE A COMPLETER
# (veuillez respecter les notations spécifiées dans l'énoncé ci-dessus)
#
import numpy as np
# 1. définition de la fonction f
def f(x):
    return 1 - 3*x*np.cos(x)

# 2. définition de la fonction Python mcorde
def mcorde(f,a,b,x_0,eps,k_max):
    nb_iterations = 0
    erreur = abs(a-b)
    x_k = x_0
    q = (f(b)-f(a))/(b-a)
    if abs(q) < 1e-6:
            raise ValueError("Attention : choix inadapté de l'intervalle !")
    while erreur > eps and nb_iterations < k_max:
        x = x_k - f(x_k)/q
        erreur = abs(x-x_k)
        x_k = x
        nb_iterations += 1
    return x_k, nb_iterations

# 3. appel de la fonction mcorde pour trouver l'approximation cherchée
a = 0.5
b = 0.6
x_0 = 0.2
tolerance = 1e-8
k_max = 50

x, nb_it = mcorde(f,a,b,x_0,tolerance,k_max)

# 4. affichage de la valeur trouvée
print('Approximation de la racine : ',x, '(obtenue après ',nb_it,'itérations)')

# FIN DU CODE A COMPLETER
```

```
Approximation de la racine :  0.3555759866610588 (obtenue après  22 itérations)
```

(c)




```
Approximation trouvée par la méthode (Newton) : x = 1.314393803189098
x_0 (vert) = 1
x_1 (bleu) = 1.6872194454729699
x_2 (cyan) = 1.3918323810223283
x_3 (jaune) = 1.3200183961424858
```

Seconde 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 seconde partie comprend un **total de 8 points + 2 points de bonus**.

Question 3 (à 2 points, question bonus)

On souhaite déterminer une approximation de la racine carrée de 2 en cherchant le zéro positif α de la fonction $f(x) = x^2 - 2$. Pour ce faire, on construit, à l'aide de la méthode de la bissection appliquée sur l'intervalle de départ $[1, 1.5]$, une suite $\{x_k\}$, $k \geq 0$, de valeurs approchées de α. A partir de quel entier k_{min}, les approximations x_k, $k \geq k_{\text{min}}$, seront-elles distantes de moins de 10^{-5} de la valeur exacte $\alpha = \sqrt{2}$?

☐ $k_{\text{min}} = 6$	☐ $k_{\text{min}} = 22$	☐ $k_{\text{min}} = 21$	■ $k_{\text{min}} = 15$
☐ $k_{\text{min}} = 8$	☐ $k_{\text{min}} = 14$	☐ $k_{\text{min}} = 23$	☐ $k_{\text{min}} = 7$
☐ $k_{\text{min}} = 16$	☐ $k_{\text{min}} = 11$	☐ $k_{\text{min}} = 12$	☐ $k_{\text{min}} = 18$

Question 4 (à 2 points)

On cherche à déterminer une approximation des zéros d'une fonction réelle f d'une variable réelle à l'aide de la méthode de la bissection.
Parmi les cinq affirmations suivantes laquelle est vraie ?

- ☐ La méthode de la bissection est une méthode de point fixe.
- ☐ La méthode de la bissection utilise la pente de f.
- ■ La méthode de la bissection est une méthode itérative qui converge toujours.
- ☐ Avec la méthode de la bissection, l'erreur numérique absolue commise diminue à chaque itération.
- ☐ La méthode de la bissection permet toujours de déterminer une approximation des zéros de f.

Question 5 (à 2 points)

On cherche à déterminer numériquement les deux zéros $\alpha = 1$ et $\beta = 2$ de la fonction

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

Pour ce faire, on applique la méthode de Picard en utilisant la fonction d'itération

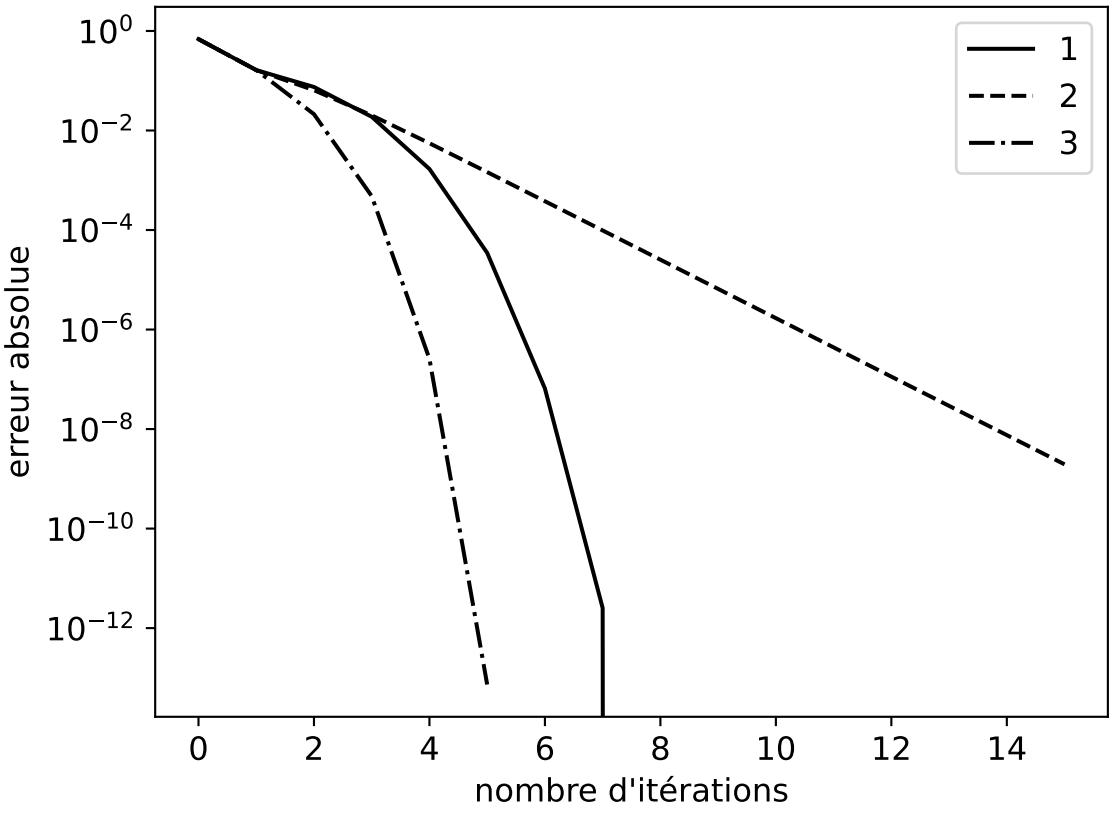
$$\phi(x) = x + f(x) = x^2 - 2x + 2\,.$$

En supposant avoir choisi un point de départ x_0 suffisamment proche du zéro cherché, laquelle des quatre affirmations suivantes est vraie ?

- ☐ La méthode converge vers α et vers β.
- ☐ La méthode converge vers β, mais pas vers α.
- ☐ La méthode ne converge ni vers α, ni vers β.
- ■ La méthode converge vers α, mais pas vers β.

Question 6 (à 2 points)

La représentation graphique suivante indique l'erreur absolue commise par trois méthodes itératives en fonction du nombre d'itérations effectuées dans le cas de la recherche du zéro β de la fonction f définie dans la Question 2 en page 4. Les trois méthodes itératives considérées sont : la méthode de la corde, la méthode de la sécante et la méthode de Newton.



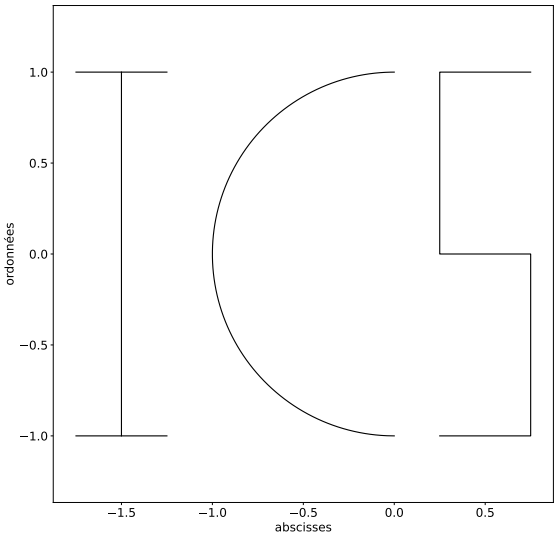
Parmi les six propositions ci-dessous, laquelle correspond à la légende la plus vraisemblable pour la représentation graphique ?

- ☐ 1 : méthode de la sécante, 2 : méthode de Newton, 3 : méthode de la corde
- ■ 1 : méthode de la sécante, 2 : méthode de la corde, 3 : méthode de Newton
- ☐ 1 : méthode de la corde, 2 : méthode de Newton, 3 : méthode de la sécante
- ☐ 1 : méthode de la corde, 2 : méthode de la sécante, 3 : méthode de Newton
- ☐ 1 : méthode de Newton, 2 : méthode de la corde, 3 : méthode de la sécante
- ☐ 1 : méthode de Newton, 2 : méthode de la sécante, 3 : méthode de la corde

Question 7 (à 2 points)

On aimerait que le code Python ci-dessous produise la figure ci-contre. Par quelles lignes de code doit-on remplacer le “BLOC DE CODE MANQUANT” pour que ce soit effectivement le cas ?

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-1,1,10000)
dc = np.sqrt(1-x**2)
plt.figure(figsize=(12,12))
plt.xlabel('abscisses', size=16)
plt.ylabel('ordonnées', size=16)

# BLOC DE CODE MANQUANT

plt.xticks(size=16)
plt.yticks(size=16)
plt.axis('equal')
plt.show()
```



Choisissez parmi les cinq propositions suivantes, le bloc de code permettant d’obtenir la figure souhaitée :

☐
```
plt.plot([-1.75,-1.25],[1,1],c='black')
plt.plot([-1.5,-1.5],[-1,1],c='black')
plt.plot([-1.75,-1.25],[-1,-1],c='black')
plt.plot(x,-dc,c='black')
plt.plot([0.75, 0.25, 0.25, 0.75, 0.75, 0.25],[1,1,0,0,-1,-1],c='black')
```

■
```
plt.plot([-1.75,-1.25],[1,1],c='black')
plt.plot([-1.5,-1.5],[-1,1],c='black')
plt.plot([-1.75,-1.25],[-1,-1],c='black')
plt.plot(-dc,x,c='black')
plt.plot([0.25,0.75,0.75,0.25,0.25,0.75],[-1,-1,0,0,1,1],c='black')
```

☐
```
plt.plot([-1.75,-1.25],[1,1],c='black')
plt.plot([-1.5,-1.5],[-1,1],c='black')
plt.plot([-1.75,-1.25],[-1,-1],c='black')
plt.plot(dc,x,c='black')
plt.plot([0.25,0.75,0.75,0.25,0.25,0.75],[-1,-1,0,0,1,1],c='black')
```

☐
```
plt.plot([-1.75,-1.25],[-1,-1],c='black')
plt.plot([-1.75,-1.25],[1,1],c='black')
plt.plot([-1.5,-1.5],[-1,1],c='black')
plt.plot(-dc,x,c='black')
plt.plot([0.75, 0.25, 0.25, 0.75, 0.75, 0.75],[1,1,0,0,-1,-1],c='black')
```

☐
```
plt.plot([-1.75,-1.25],[1,1],c='black')
plt.plot([-1.5,-1.5],[-1,1],c='black')
plt.plot([-1.75,-1.25],[-1,-1],c='black')
plt.plot(x,dc,c='black')
plt.plot([0.25,0.75,0.75,0.25,0.25,0.75],[-1,-1,0,0,1,1],c='black')
```

Aide-mémoire (librairies 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)
```