+1/1/60+

EPFL

1

Enseignant : Roger Sauser
ICS - Contrôle 4 - CMS
12 juin 2024
Durée : 105 minutes

Nom Prénom

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.e** 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 20 points + 2 points de bonus**.

Question 1: *Cette question est notée sur 12 points (dont 2 points de bonus).*

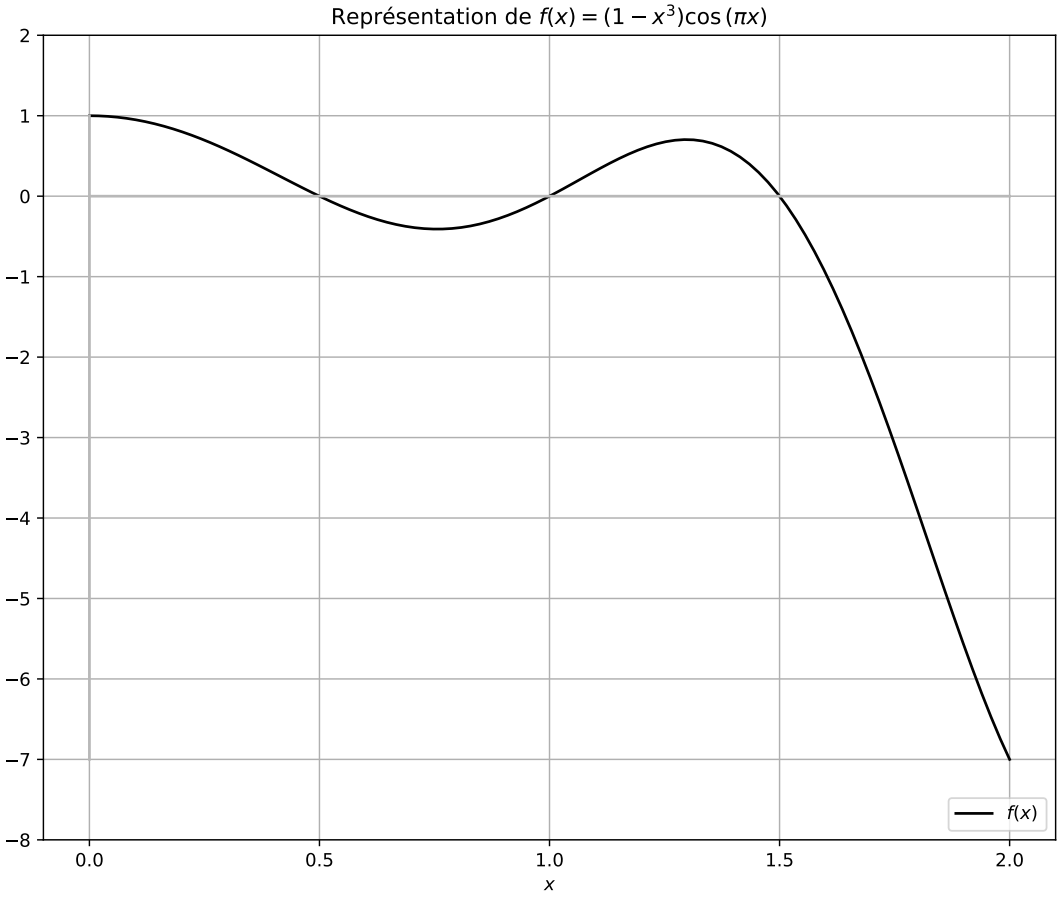
☐0 ☐.5 ☐1 ☐.5 ☐2 ☐.5 ☐3 ☐.5 ☐4 ☐.5 ☐5 ☐.5 ☐6 ☐.5 ☐7 ☐.5 ☐8 ☐.5 ☐9 ☐.5 ☐10 ☐.5 ☐11 ☐.5 ☐12

On considère la fonction réelle d’une variable réelle $f : \mathbb{R} \to \mathbb{R}$ définie par

$$f(x) = (1 - x^3)\cos(\pi x)\,.$$

On donne ci-dessous la représentation graphique de cette fonction sur l’intervalle (en abscisse) $[0, 2]$ et on considère l’intégrale définie

$$I = \int_0^2 f(x)\,dx\,.$$



Dans les deux parties (a) et (b) ci-dessous, on cherche à approximer l’intégrale I à l’aide de deux méthodes (formules) de quadrature différentes. La partie (c) est une question bonus à 2 points concernant l’efficacité des deux méthodes utilisées dans les parties (a) et (b).

(a) En vous aidant du dessin en page précédente, déterminez “à la main” (sans écrire de code destiné à être exécuté par un ordinateur) la valeur de l’approximation numérique J^{TR} de I obtenue en appliquant la méthode de quadrature composite basée sur la **formule du trapèze**. Considérez une partition régulière du domaine d’intégration $[0, 2]$ en quatre sous-intervalles. Précisez au mieux votre raisonnement en indiquant les points utiles aux calculs sur le dessin.

(b) Complétez le code Python ci-contre en page 5 de manière à ce que ce dernier permette de déterminer une approximation numérique J^{PM} de I grâce à la **formule (méthode composite) du point milieu**.

Plus précisément, le code doit :

- importer la (ou les) librairie(s) utile(s). La librairie `SciPy` ne doit pas être utilisée ;
- définir la fonction Python `f` qui correspond à la fonction mathématique $f(x)$ à intégrer définie plus haut ;
- définir une fonction `int_pm` admettant comme arguments :
  - la fonction `f` à intégrer ;
  - les bornes `a` et `b` de l'intervalle d'intégration ;
  - le nombre `N` de sous-intervalles à considérer ;

  Cette fonction `int_pm` doit :
  - vérifier que les trois derniers arguments donnés à la fonction sont bien choisis, c'est-à-dire qu'ils sont tels que $b \geq a$, $N \geq 1$, N étant un entier, et, si ce n'est pas le cas, afficher le message "`Attention au choix des arguments !`" ; suite à l'affichage du message, la fonction doit s'arrêter et retourner `None` ;
  - calculer le pas `h` qui sera utilisé par la méthode composite ;
  - déterminer la valeur de l'approximation `Jpm` en implémentant la méthode composite du point milieu avec `N` sous-intervalles ;
  - retourner la valeur de `Jpm` ;
- afficher l'approximation J^{PM} de l'intégrale définie I obtenue grâce à la fonction `int_pm` en considérant quatre sous-intervalles.

(le code à compléter se trouve en page suivante)

(c) L'exécution du code qu'il vous est demandé de compléter dans la partie (b) permettrait en principe de comparer la méthode du point milieu à la méthode du trapèze utilisée dans la partie (a). Sans faire de calcul, mais en justifiant votre réponse, comparez en particulier l'erreur absolue commise dans l'approximation de I : une des deux méthodes est-elle plus précise que l'autre ? Si oui, pouvez-vous indiquer le gain en précision obtenu en utilisant une méthode plutôt qu'une autre ?

Suite de la réponse (partie (c)) :

Code à compléter (partie (b)) :

```
# DEBUT du code à compléter
# (veuillez respecter les notations spécifiées dans l'énoncé ci-dessus)
#
# importation de la (ou des) librairie(s) utile(s)
import numpy as np



# définition de la fonction f à intégrer
def f(x):



# définition de la fonction Python int_pm
def int_pm(f,a,b,N):
                                                    # test de cohérence des trois
                                                    # derniers arguments et affichage
                                                    # d'un message si nécessaire




        Jpm = 0.0
        h =                                 # calcul du pas h
```

```
        return Jpm

# affichage de l'approximation de I demandée



# FIN du code à compléter
```

Question 2: *Cette question est notée sur 10 points.*

Dans cette question, on considère le problème de Cauchy suivant :

$$\begin{cases} y'(t) &= f\big(t, y(t)\big)\,, \quad \forall t \in I = [t_0, T]\,, \\ y(t_0) &= y_0\,, \end{cases}$$

avec

$$f\big(t, y(t)\big) = \frac{3t^2}{2y(t)}\,,\ t_0 = 0\,,\ T = 4 \ \text{ et } \ y_0 = 3\,.$$

Les parties (a), (b) et (c) ci-dessous peuvent être résolues de manière indépendante.

(a) Vérifiez, par un calcul analytique, que la fonction $y(t) = \sqrt{t^3 + 9}$ est solution du problème de Cauchy ci-dessus.

(b) Calculez “à la main”, en utilisant la **méthode d’Euler progressive**, une approximation de la solution $y(t)$ du problème de Cauchy ci-dessus en $T = 4$. Déterminez cette approximation en considérant deux sous-intervalles avec un pas de 2.

(c) Complétez le code Python donné en page suivante de manière à ce qu'il permette de déterminer, grâce à la **méthode de Heun**, une approximation de la solution $y(t)$ du problème de Cauchy ci-dessus de $t_0 = 0$ à $T = 4$.

Plus précisément, le code doit :

- importer la librairie NumPy ;
- définir le problème de Cauchy donné ci-dessus en définissant la fonction Python `f` correspondant à $f(t, y)$ ainsi que la condition initiale (condition de Cauchy) ;
- définir une fonction Python nommée `Heun` admettant pour paramètres (arguments) la fonction `f`, le moment (temps) initial `t_0`, la valeur initiale `y_0`, le moment final `T`, ainsi que le nombre de sous-intervalles `N` considérés.

  La fonction `Heun` doit créer un "vecteur" (`ndarray`) temps `t` renfermant $N + 1$ éléments régulièrement espacés entre t_0 et T.

  De plus, la fonction `Heun` doit créer un "vecteur" (`ndarray`) `u` avec également $N + 1$ éléments. Le premier élément de `u` correspond à la condition initiale et les autres éléments sont déterminés en utilisant la **méthode de Heun**.

  La fonction `Heun` doit retourner le vecteur `t` ainsi que le vecteur `u` contenant la solution approchée ;
- appeler la fonction `Heun` avec un nombre $N = 10$ de sous-intervalles ;
- afficher la valeur du pas utilisé dans la méthode ainsi que la valeur approchée de $y(T)$ à la place des points d'interrogation dans le message suivant : "Avec un pas de ?, la solution approchée prend la valeur ? en T=4.".

(le code à compléter se trouve en page suivante)

Code à compléter (partie (c)) :

```
# DEBUT DU CODE A COMPLETER
# (veuillez respecter les notations spécifiées dans l'énoncé ci-dessus)
#
# importation de la librairie NumPy
import numpy as np

# définition du problème de Cauchy
def f(t,y):









# définition de la fonction Python Heun
def Heun(f,t_0,y_0,T,N):

    t =

















    return

# appel de la fonction Heun


# affichage demandé




# FIN DU CODE A COMPLETER
```

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 10 points**.

Les questions 3 et 4 se rapportent à l'énoncé suivant.
Soit le tableau de Butcher suivant associé à une méthode de Runge-Kutta à un pas :

$$\begin{array}{c|ccc} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1/3} & \mathbf{1/3} & \mathbf{0} & \mathbf{0} \\ \mathbf{2/3} & \mathbf{0} & \mathbf{2/3} & \mathbf{0} \\ \hline & \mathbf{1/4} & \mathbf{0} & \mathbf{3/4} \end{array}$$

Question 3 (à 2 points)

La méthode définie par le tableau de Butcher ci-dessus est une...

- ☐ ...méthode implicite à 3 étapes.
- ☐ ...méthode implicite à 2 étapes.
- ☐ ...méthode implicite à 4 étapes.
- ☐ ...méthode explicite à 4 étapes.
- ☐ ...méthode explicite à 3 étapes.
- ☐ ...méthode explicite à 2 étapes.

Question 4 (à 2 points)

Parmi les pentes et les schémas numériques suivants, laquelle (ou lequel) n'est pas associé(e) au tableau de Butcher ci-dessus ?

- ☐ $u_{n+1} = u_n + h\frac{1}{4}K_1 + h\frac{3}{4}K_3$
- ☐ $u_{n+1} = u_n + h\frac{1}{3}K_2 + h\frac{2}{3}K_3$
- ☐ $K_3 = f(t_n + \frac{2}{3}h, u_n + h\frac{2}{3}K_2)$
- ☐ $K_2 = f(t_n + \frac{1}{3}h, u_n + h\frac{1}{3}K_1)$
- ☐ $K_1 = f(t_n, u_n)$
- ☐ $K_2 = f(t_0 + \frac{3n+1}{3}h, u_n + h\frac{1}{3}K_1)$

Les questions 5 et 6 se rapportent à l'énoncé suivant.
Soit f une fonction réelle de la variable réelle t :

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

Question 5 (à 2 points)

Parmi les six polynômes ci-dessous, lequel est le polynôme d'interpolation (de Lagrange) qui passe par les trois points d'abscisse $t_1 = -1$, $t_2 = 0$ et $t_3 = 2$ du graphe de la fonction $f(t)$?

- ☐ $-t^2 + 3t + 4$
- ☐ $t^3 - t^2 - 4t + 4$
- ☐ $2t^2 - 6t + 4$
- ☐ $t^4 - t^2 + 4$
- ☐ $-2t^2 + 2t + 4$
- ☐ $(1+t)(t-2)$

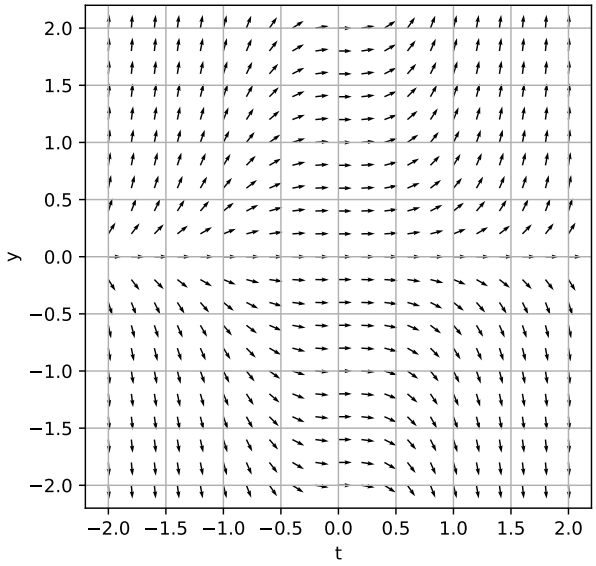
Question 6 (à 2 points)

Parmi les six polynômes ci-dessous, lequel est un polynôme de la base de Lagrange associée aux trois points d'abscisse $t_1 = -1$, $t_2 = 0$ et $t_3 = 2$ du graphe de la fonction $f(t)$?

- ☐ $\frac{1}{6}t(t+1)$
- ☐ $-\frac{3}{2}t^2 + \frac{3}{2}t + 4$
- ☐ $-2t^2 + 2t + 4$
- ☐ $\frac{1}{3}(t+1)$
- ☐ $t^3 - 3t^2 + 4$
- ☐ $t + 1$

Question 7 (à 2 points)

On aimerait que le code Python ci-dessous produise la figure ci-contre qui représente le champ de directions de l'équation différentielle ordinaire $y' = 2t^2y$. 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
def f(t,y):
    return 2*y*t**2
axe_t = np.linspace(-2, 2, 21)
axe_y = np.linspace(-2, 2, 21)
plt.figure(figsize=(5,5))

# BLOC DE CODE MANQUANT

plt.grid()
plt.xlabel('t')
plt.ylabel('y')
plt.show()
```

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

☐
```
t, y = np.meshgrid(axe_t, axe_y)
direction_t = 1
direction_y = y
norme = np.sqrt(direction_t**2 + direction_y**2)
plt.quiver(t, y, direction_t/norme, direction_y/norme)
```

☐
```
t, y = np.meshgrid(axe_t, axe_y)
direction_t = 1
direction_y = f(t,y)
norme = direction_t**2 + direction_y**2
plt.quiver(t, y, direction_t/norme, direction_y/norme)
```

☐
```
t, y = np.meshgrid(axe_t, axe_y)
direction_t = 1
direction_y = f(t,y)
plt.quiver(t, y, direction_t, direction_y)
```

☐
```
direction_t = 1
direction_y = f(axe_t,axe_y)
norme = np.sqrt(direction_t**2 + direction_y**2)
plt.quiver(axe_t, axe_y, direction_t/norme, direction_y/norme)
```

☐
```
t, y = np.meshgrid(axe_t, axe_y)
direction_t = 1
direction_y = f(t,y)
norme = np.sqrt(direction_t**2 + direction_y**2)
plt.quiver(t, y, direction_t/norme, direction_y/norme)
```

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
from scipy import integrate

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)
integrate.quad(f, a, b)
integrate.odeint(f, y0, t, tfirst=True)
```

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)
plt.quiver(x,y,u,v)
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.fill_between(x, y1, y2)
plt.clabel(cs)
plt.axis('equal')
plt.axis('scaled')
plt.grid()
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)
```