+1/1/60+

EPFL

1

Enseignant : Roger Sauser
ICS - CMS
14 juin 2023
Durée : 105 minutes

RS

SCIPER : **22**

Attendez le début de l'épreuve avant de tourner la page. Ce document est imprimé recto-verso. Il contient 20 pages et 7 questions. Ne pas dégrafer.

- 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 du papier et un stylo) :
  - respectez la syntaxe Python (parenthèses, crochets, accolades, 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 elle doit permettre d'identifier vos blocs de code de manière claire et immédiate) ;
  - votre réponse doit comporter uniquement du code exécutable, à l'exception de quelques commentaires (au format habituel) si ceux-ci sont véritablement nécessaires et aident à la compréhension.

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, trois 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 24 points** (**22 points + 2 points de bonus**).

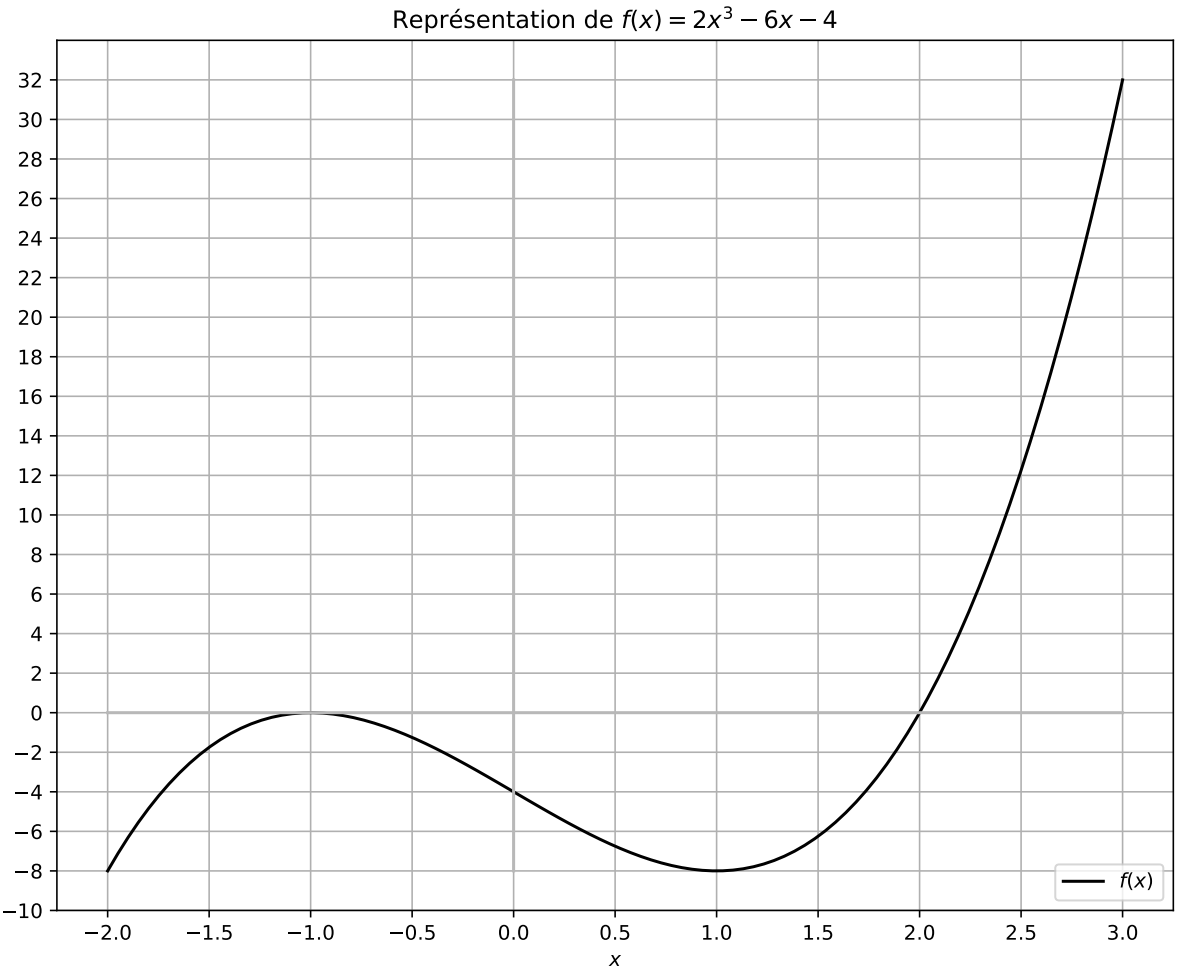
Question 1: *Cette question est notée sur 12 points.*

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

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

On donne ci-dessous la représentation graphique de cette fonction sur l'intervalle $[-2, 3]$ et on considère l'intégrale définie

$$I = \int_{-2}^{3} f(x)\,dx\,.$$



Les parties (a) et (b) en pages 3 et 4 peuvent être résolues de manière indépendante. Pour comprendre la partie (b), il faut toutefois avoir lu la partie (a).

(a) Complétez le code Python en page suivante de manière à ce qu'il permette de déterminer une approximation numérique J de l'intégrale définie I grâce à l'une des deux méthodes de quadrature composite suivantes :

- méthode basée sur la **formule dite du point de gauche** ;
- méthode basée sur la **formule de Simpson**.

Plus précisément, le code doit définir une fonction `integration` 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 ;
- la méthode choisie : `gauche` pour la méthode du point de gauche ou `simpson` pour la méthode de Simpson.

Cette fonction `integration` doit :

- vérifier que la borne `b` est bien plus grande que la borne `a` ; la fonction doit afficher le message "`L'intervalle [a,b] est mal défini.`" et retourner "`None`" si ce n'est pas le cas ;
- vérifier que la méthode donnée en argument à la fonction `integration` est soit `gauche` soit `simpson` ; si ce n'est pas le cas, la fonction doit afficher le message "`La méthode demandée n'est pas implémentée.`" et retourner "`None`" ;
- calculer le pas `dx` qui sera utilisé par la méthode composite ;
- déterminer la valeur `J` de l'approximation en implémentant la méthode composite choisie ;
- retourner la valeur `J`.

(le code à compléter se trouve en page suivante)

Code à compléter pour la partie (a) de la Question 1 :

```
# DEBUT du code à compléter
# (veuillez respecter les notations spécifiées dans l'énoncé)
#
# définition de la fonction Python integration
def

    if                                                          # test de la cohérence de
                                                                # l'intervalle et affichage
                                                                # d'un message d'erreur
                                                                #
                                                                # test de la méthode
                                                                # demandée et affichage
        print("La méthode demandée n'est pas implémentée.")     # d'un message d'erreur
                                                                #


    else:
        J = 0.0
        dx =                                                    # calcul du pas dx


        x_gauche = a                # initialisation des bornes des sous-intervalles
        x_droite = a + dx           # utilisées ensuite dans l'implémentation de la
                                    # méthode d'intégration numérique choisie

        for _ in range                          # implémentation de la méthode précisée
                                                # en argument
            if

        return J

# FIN du code à compléter
```

(b) Donnez l'affichage qui sera alors produit par la suite d'instructions suivante :

```
def f(x):
    return 2*(x+1)**2*(x-2)

print("J_gauche =",integration(f,-2,3,5,'gauche'))
print("J_simpson =",integration(f,-2,3,1,'simpson'))
```

Votre réponse doit être soigneusement justifiée, par exemple en vous appuyant sur la représentation graphique de la fonction f donnée en page 2. Comparez les valeurs approchées obtenues à la valeur exacte de I (sans forcément faire le calcul de la valeur exacte de I, mais en justifiant votre conclusion).

(réponse en page suivante)

Question 2: *Cette question est notée sur 3 points.*

.5 .5 .5

0 1 2 3

Soient $m+1$ noeuds (points) de quadrature distincts $-1 \leq t_0 < t_1 < \ldots < t_m \leq 1$ et $\{\varphi_0, \varphi_1, \ldots, \varphi_j, \ldots, \varphi_m\}$ la base de Lagrange de l'espace vectoriel $\mathcal{P}_m$ (espace vectoriel des polynômes de degré inférieur ou égal à m) associée à ces points de quadrature.

Soit la formule de quadrature

$$J(g) = \sum_{j=0}^{m} \mu_j g(t_j)\,.$$

Montrez que si les poids de quadrature μ_j, $\forall j \in \{0, 1, \ldots, m\}$, sont donnés par

$$\mu_j = \int_{-1}^{1} \varphi_j(t) dt\,,$$

alors le degré d'exactitude de la formule de quadrature ci-dessus est (au moins) m.

Question 3: *Cette question est notée sur 9 points.*

Soit $c(t)$ une fonction du temps t correspondant à la concentration d'une substance dans le sang d'un être vivant. On suppose que l'évolution de c est décrite par l'équation différentielle suivante :

$$c'(t) = 2\exp(-t) - (c(t))^2\,,$$

où le temps t est donné en heures et la concentration c en g/l. Cette équation différentielle est complétée par la condition initiale (à $t_0 = 0\,\mathrm{h}$) :

$$c(0) = 0\,\mathrm{g/l}\,.$$

Les parties (a) et (b) ci-dessous peuvent être résolues de manière indépendante. Pour comprendre la partie (b), il faut toutefois avoir lu la partie (a). La partie (b) est une question bonus valant 2 points sur les 9 points de la question.

(a) Complétez le code Python donné en page suivante de manière à ce qu'il permette de déterminer l'évolution de la concentration c de $t_0 = 0\,\mathrm{h}$ à $T = 10\,\mathrm{h}$.

Plus précisément, le code doit :

- importer la librairie NumPy ;
- définir le problème de Cauchy que l'on se propose de résoudre dans cette question en définissant la fonction $f(t, y)$ ainsi que la condition initiale ;
- créer un "vecteur" (`ndarray`) temps `t` avec 51 instants régulièrement espacés entre 0 et 10 heures ;
- créer un "vecteur" (`ndarray`) concentration `c` avec 51 éléments dont le premier élément correspond à la condition initiale et dont les autres éléments seront déterminés par la suite ;
- déterminer l'évolution de la concentration en utilisant la méthode de Crank-Nicolson (pour la partition régulière mentionnée) et mettre à jour au fur et à mesure les valeurs des éléments du "vecteur" `c` ; l'équation implicite intervenant dans la méthode de Crank-Nicolson doit être résolue à l'aide de la méthode de Picard (méthode du point fixe) en prenant comme point de départ la valeur obtenue en appliquant le schéma d'Euler progressif, et en choisissant une tolérance de 10^{-8} et un nombre maximum d'itérations de 20 ; la librairie SciPy ne doit pas être utilisée ;
- afficher la valeur approchée de la concentation après 10 heures.

(le code à compléter se trouve en page suivante)

Code à compléter pour la partie (a) de la Question 3 :

```
# DEBUT du code à compléter
# (veuillez respecter les notations spécifiées dans l'énoncé)
#
                                                    # importation de la librairie NumPy

# définition du problème de Cauchy :
# * équation différentielle à résoudre (membre de droite)
def



# * condition initiale (valeur de la concentration c en t=0)
y_0 = 0

t_0 = 0        # instant initial
T = 10         # instant final
N = 51         # nombre d'instants

tol =          # tolérance

i_max = 20     # nombre maximum d'itérations

t =

c =

c[0] = y_0

h = T/(N-1)

for n in range(N-1):
    fn = f(t[n], c[n])
    # calcul du point de départ avec Euler progressif
```

```
# FIN du code à compléter
```

(b) **Cette partie (b) est une question bonus valant 2 points.**

En représentant les résultats obtenus en fonction du temps (c'est-à-dire en représentant le "vecteur" `c` en fonction du "vecteur" `t`), on observe que la concentration commence par augmenter avant de diminuer.

En supposant avoir exécuté le code de la partie (a), donnez ci-dessous les quelques lignes de code supplémentaires permettant d'estimer après combien d'heures la concentration est redescendue à la moitié de la valeur maximale qu'elle a atteinte.

(réponse en page suivante)

Seconde partie, quatre 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**.

Question 4 (à 2 points)

On cherche à intégrer sur un intervalle $[a, b]$ une fonction d'une variable réelle suffisamment régulière (plus précisément, une fonction dont les dérivées sont continues). Pour ce faire, on utilise une formule de quadrature composite basée sur une méthode de quadrature non composite de degré d'exactitude 3.
En passant d'une partition régulière comprenant n sous-intervalles à une partition régulière plus fine comprenant $3n$ sous-intervalles, on observe que l'erreur absolue commise est divisée par un certain facteur k. Quel est, en bonne approximation, ce facteur ?

- ☐ $k = 8$
- ☐ $k = 9$
- ☐ $k = 64$
- ☐ $k = 16$
- ☐ $k = 81$
- ☐ $k = 3$

Question 5 (à 2 points)

On considère le polynôme de Lagrange $p(t)$ associé aux cinq points suivants :

$$P_0 = (-2, 1),\ P_1 = (-1, 0),\ P_2 = (0, -1),\ P_3 = (1, -2) \text{ et } P_4 = (2, 21)\,.$$

Parmi les affirmations suivantes, laquelle est fausse ?

- ☐ Le polynôme $p(t)$ a pour expression : $p(t) = t^4 + 2t^3 - t^2 - 3t - 1\,.$
- ☐ Le polynôme $p(t)$ est de degré $4\,.$
- ☐ Le polynôme $p(t)$ peut s'écrire sous la forme d'une combinaison linéaire : $p(t) = \sum_{j=0}^{3} p_j \varphi_j(t)$, où les polynômes $\varphi_j(t)$, $j \in \{0, 1, 2, 3\}$, forment une base de l'espace vectoriel des polynômes de degré inférieur ou égal à $4\,.$
- ☐ Le polynôme $p(t)$ est l'unique polynôme d'interpolation pour les cinq points P_0, P_1, P_2, P_3 et P_4.
- ☐ Le polynôme $p(t)$ passe par les cinq points P_0, P_1, P_2, P_3 et $P_4\,.$

Question 6 (à 2 points)

Soit le problème de Cauchy suivant :

$$\begin{cases} y'(t) &= t^2 \sin\left(y(t)\right), \ \forall t \in I = [t_0, T], \\ y(t_0) &= \dfrac{\pi}{2}. \end{cases}$$

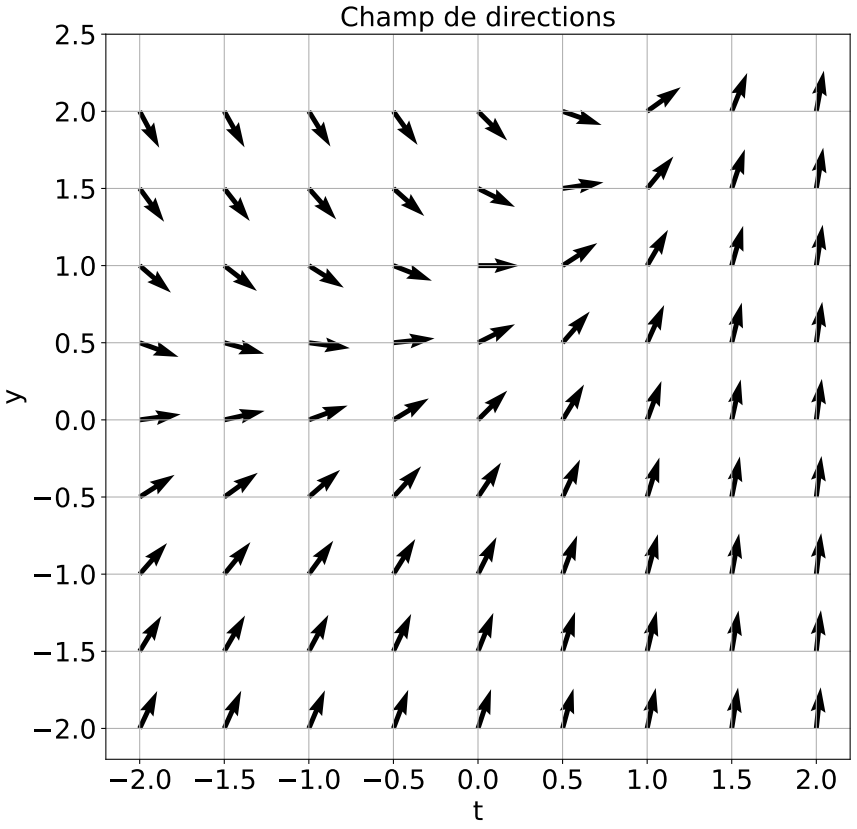
Vers quelle valeur réelle va tendre la solution $y(t)$ lorsque l'on considère des temps t grands (c'est-à-dire lorsque l'on considère un intervalle I avec T grand) ?

- ☐ -2π
- ☐ 2π
- ☐ 0
- ☐ π
- ☐ $-\pi/2$
- ☐ $-\pi$

Question 7 (à 2 points)

A laquelle des équations différentielles suivantes correspond le champ de directions représenté ci-dessous ?

- ☐ $y'(t) = y(t) + \exp(t)$
- ☐ $y'(t) = -y(t) - \exp(-t)$
- ☐ $y'(t) = y(t) - \exp(t)$
- ☐ $y'(t) = -y(t) + \exp(t)$



Aide-mémoire (langage Python)

Opérations communes aux séquences – cas du type list	Opérations avec des sets (objets set ou frozenset)
lis[i]	len(se)
lis[i:j]	el in se
lis[i:j:k]	el not in se
lis + seq	s_this.isdisjoint(s_that)
lis * n ou n * lis	s_this.issubset(s_that)
x in lis	s_this <= s_that
x not in lis	s_this < s_that
lis.index(x)	s_this.issuperset(s_that)
lis.count(x)	s_this >= s_that
len(lis)	s_this > s_that
min(lis)	s_this.union(*s_that)
max(lis)	s_this | s_that_1 | …
Opérations communes aux séquences mutables – cas du type list	s_this.intersection(*s_that)
lis[i] = x	s_this & s_that_1 & …
lis[i:j] = it	s_this.difference(*s_that)
lis[i:j:k] = it	s_this - s_that_1 - …
lis.append(x)	s_this.symmetric_difference(s_that)
lis.insert(i, x)	s_this ^ s_that
lis.extend(it) ou lis += it	s_this.copy()
lis *= n	
del lis[i:j:k]	
lis.pop(i)	
lis.remove(x)	
lis.clear()	
lis.copy()	
lis.reverse()	

Opérations avec des objets mutables de type set	iter(dic)
s_this.update(*s_that)	reversed(dic)
s_this |= s_that_1 | …	dic.values()
s_this.intersection_update(*s_that)	dic.clear()
s_this &= s_that_1 & …	dic.copy()
s_this.difference_update(*s_that)	dic.setdefault(key, val)
s_this -= s_that_1 | …	dic1.update(dic2)
s_this.symmetric_difference_update(s_that)	dic1 | dic2
s_this ^= s_that	dic1 |= dic2
s_this.add(el)	Opérations avec des strings
s_this.remove(el)	s[i]
s_this.discard(el)	s[i:j]
s_this.pop()	s[i:j:k]
s_this.clear()	s1 + s2
Opérations avec des dictionnaires	s * n ou n * s
len(dic)	sub in s
dic[key]	sub not in s
dic[key] = val	s.index(sub)
del dic[key]	s.count(sub)
key in dic	len(s)
key not in dic	min(s)
dic.get(key)	max(s)
dic.pop(key)	
dic.popitem()	
dic.items()	
dic.keys()	
list(dic)	

Méthodes d'instance de la classe str

- s.upper()
- s.lower()
- s.capitalize()
- s.split(*sep=None*, *maxsplit=-1*)
- s.rsplit(*sep=None*, *maxsplit=-1*)
- s.splitlines(*keepends*)
- s.join(*iterable*)
- s.find(sub, *start*, *end*)
- s.rfind(sub, *start*, *end*)
- s.rindex(sub)
- s.replace(old, new, *count*)
- s.strip(*chars*)
- s.lstrip()
- s.rstrip()
- s.startswith(prefix, *start*, *end*)
- s.endswith(suffix, *start*, *end*)
- s.removeprefix(prefix)
- s.removesuffix(suffix)
- s.swapcase()

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)

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.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)

Aide-mémoire (préparé par les étudiant(e)s)

Matplotlib, Numpy et Scipy

- `plt.axhline(y, xmin, xmax)` # dessiner une ligne horizontale / `xmin` et `xmax` arguments optionnels
- `plt.axvline(x, ymin, ymax)` # dessiner une ligne verticale / `ymin` et `ymax` arguments optionnels
- **Graphe d'un champ vectoriel** Pour représenter un champ vectoriel à 2 dimensions $\vec{C} = \begin{pmatrix} C_x(x,y) \\ C_y(x,y) \end{pmatrix}$ défini par une fonction scalaire $\Psi(x,y) : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$, qu'on définira ici par `psi(xv, yv)` (formule généralement donnée en exercice), avec `a` et `b` bornes inférieures et supérieures pour `x` et `y` (même longueur) :

```
a, b = -4, 4 # bornes du plan considéré
x = y = np.linspace(a, b, 40) # on peut choisir autre que 40
xv, yv = np.meshgrid(x, y, indexing = 'ij') # création de deux matrices avec les composantes
                                              x et y à chaque point
tableau_psi = psi(xv, yv)
direction_x, direction_y = np.gradient(tableau_psi) # dérivées partielles pour représenter
                                                    la pente à chaque point

fig = plt.figure(1); ax = plt.gca()
ax.quiver(xv, yv, direction_x, direction_y) # cette fonction représente le champ vectoriel
                                            sur la figure
```

Compléments d'analyse

- **Formule de Taylor**. Pour $f : I \to \mathbb{R}$ une fonction $n+1$ fois différentiable sur l'intervalle ouvert I,

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \ldots + f^{(n)}(x_0)\frac{(x-x_0)^n}{n!} + f^{(n+1)}(\xi)\frac{(x-x_0)^{n+1}}{(n+1)!}$$

- **Interpolation de Lagrange**. Pour approximer une fonction passant par $m+1$ points distincts connus $\{(t_0, p_0), \ldots (t_k, p_k), \ldots (t_m, p_m) | (t_k, p_k) \in \mathbb{R}^2\}$, on a:
  - Les polynômes $\varphi_k(t) = \prod_{j \neq k}^{m} \frac{t - t_j}{t_k - t_j}$ où l'on exclut l'indice k de l'itération pour $j : 0, \ldots, k-1, \not{k}, k+1 \ldots, m$
  - L'approximation finale $p(t) = \sum_{k=1}^{n} p_k \varphi_k(t)$ qui est un polynôme de degré m

Équations non linéaires

- **Méthode des parties proportionnelles** :

$$x_n = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}, \quad \text{pour un intervalle fermé } [a_n, b_n] \text{ respectant la condition de Bolzano}$$

- **Méthode de Newton** : $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
- **Méthode de la corde** : $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)}, \quad$ pour un x_0 choisi au départ
- **Méthode de la sécante** : $x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$

Calcul intégral

Formules de quadrature non composites Pour une fonction donnée f sur un intervalle $[i, j]$,

- ⋆ "Scaling" : $\int_{x_i}^{x_{i+1}} f(x)\,dx = \frac{x_{i+1}-x_i}{2}\int_{-1}^{1} g(t)\,dt$ où $g(t) = f\left(\frac{x_i+x_{i+1}}{2} + t\cdot\frac{x_{i+1}-x_i}{2}\right)$, et $x = \frac{x_i+x_{i+1}}{2} + t\cdot\frac{x_{i+1}-x_i}{2}$
- ⋆ Poids de quadrature $\mu_j = \int_{-1}^{1}\varphi_j(t)\,dt$, où φ_j est la base de Lagrange associée aux points t_j, $j = 0,\dots,m$.
- **Point milieu** $(m=0)$: $J_i^{PM}(f) = (x_{i+1}-x_i) f\left(\frac{x_i+x_{i+1}}{2}\right)$
- **Trapèze** $(m=1)$: $J_i^{Tr}(f) = (x_{i+1}-x_i)\left(\frac{f(x_i)+f(x_{i+1})}{2}\right)$
- **Simpson** $(m=2)$: $J_i^{S}(f) = (x_{i+1}-x_i)\left[\frac{1}{6}f(x_i) + \frac{4}{6}f\left(\frac{x_i+x_{i+1}}{2}\right) + \frac{1}{6}f(x_{i+1})\right]$
- $\frac{3}{8}$ **de Newton** $(m=3)$:
  $J_i^{N}(f) = (x_{i+1}-x_i)\left[\frac{1}{8}f(x_i) + \frac{3}{8}f\left(x_i+\frac{1}{3}(x_{i+1}-x_i)\right) + \frac{3}{8}f\left(x_i+\frac{2}{3}(x_{i+1}-x_i)\right) + \frac{1}{8}f(x_{i+1})\right]$

Formules de quadrature composites Pour un intervalle $[a, b]$ divisé en n sous-intervalles,

- Formule générale : $\int_a^b f(x)\,dx \simeq I(f) = \sum_{i=0}^{n-1} J_i(f)$, où J_i est une formule de quadrature non-composite.
- Avec pas régulier $h = \frac{b-a}{n}$: h peut être mis en évidence (en facteur devant l'opérateur de sommation Σ).
  - Simpson "dopé" $\left(I_{reg}^{S}\right)^{++} = \frac{16\left(I_{reg}^{S}\right)_{k_{\text{fin}}} - \left(I_{reg}^{S}\right)_{k_{\text{fin}-1}}}{15}$

Estimation d'erreur Pour une formule de quadrature composite $I(f)$ qui utilise une formule non-composite d'exactitude r et à pas (fixe) h, $\exists$ constante $C \in \mathbb{R}^*$ telle que $\left|\int_a^b f(x)\,dx - I(f)\right| \le Ch^{r+1}$, avec :

- Point Milieu $(r=1)$: $C_{reg}^{PM} = \frac{b-a}{24}\max_{\xi\in]a,b[}|f''(\xi)|$
- Trapèze $(r=1)$: $C_{reg}^{Tr} = \left|-\frac{b-a}{12}\right|\max_{\xi\in]a,b[}|f''(\xi)|$
- Simpson $(r=3)$: $C_{reg}^{S} = \frac{b-a}{90\cdot 32}\max_{\xi\in]a,b[}\left|f^{IV}(\xi)\right|$

Équations différentielles (ordinaires)

Méthodes de résolution d'EDO d'ordre 1 *(problème de Cauchy)*
En posant $f_n = f(t_n, u_n)$, $h = t_{n+1} - t_n$, et $f_{n+1} = f(t_{n+1}, u_{n+1})$:

- **Schéma général**
$$\begin{cases} u_{n+1} = u_n + h\cdot(\text{méthode}) \\ u_0 = y_0 \end{cases}$$
- Euler progressif: $u_{n+1} = u_n + hf_n$
- Euler rétrograde: $u_{n+1} = u_n + hf_{n+1}$, avec f_{n+1} qui dépend de u_{n+1}
- Crank-Nicolson: $u_{n+1} = u_n + \frac{h}{2}(f_n + f_{n+1})$
- Heun: $u_{n+1} = u_n + \frac{h}{2}[f_n + f(t_n+h, u_n+hf_n)]$
- Euler modifiée:
  $u_{n+1} = u_n + hf\left(t_n + \frac{1}{2}h, u_n + \frac{h}{2}f_n\right)$
- Runge-Kutta classique:
  $u_{n+1} = u_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$
  - $\hookrightarrow K_1 = f(t_n, u_n)$
  - $\hookrightarrow K_2 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2}K_1\right)$
  - $\hookrightarrow K_3 = f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2}K_2\right)$
  - $\hookrightarrow K_4 = f(t_{n+1}, u_n + hK_3)$

- **Tableau de Butcher**

Le tableau suivant :

$$\begin{array}{c|ccc} c_1 & a_{1,1} & \dots & a_{1,s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s,1} & \dots & a_{s,s} \\ \hline & b_1 & \dots & b_s \end{array}$$

se décode dans la forme du *schéma général d'une méthode de résolution d'EDO1* de la manière suivante:

$u_{n+1} = u_n + h\left(b_1 K_1 + \ldots + b_i K_i + \ldots + b_s K_s\right)$
avec $K_i = f\left(t_n + c_i h, u_n + h\sum_{j=1}^{s} a_{i,j} K_j\right)$

Estimation d'erreur

- **Erreur de troncature** : connaissant les solutions exactes y_{n-1} et y_n, la solution approchée (obtenue à partir de la solution exacte au point précédent) $\tilde{u}_n = y_{n-1} + h \cdot$ (méthode), ainsi que la solution approximée u_n à partir du schéma de base, on aura à une itération n:
  - **Erreur totale** $e_{\text{tot}} = |y_n - u_n| = |y_n - \tilde{u}_n + \tilde{u}_n - u_n|$
  - **Erreur locale absolue** $e_{\text{loc}} = |y_n - \tilde{u}_n|$
  - **Erreur transportée absolue** $e_{\text{trans}} = |\tilde{u}_n - u_n|$
- **Erreur de troncature locale**

  $|y_n - \tilde{u}_n| = \frac{1}{2}h^2|y''(\theta_n)|$

  $\tau_n(h) = \dfrac{|y_n - \tilde{u}_n|}{h}$ (erreur de troncature locale unitaire)

  $\tau(h) = \frac{1}{2}hM = \frac{1}{2}h\max_{t\in]t_0,T[}|y''(t)|$
- **Stabilité et erreur de troncature transportée**

  condition de stabilité globale : $h \leq \frac{2}{\max_{t\in]t_0,T[}\left|\frac{\partial f}{\partial y}(t,y(t))\right|}$
- **Précisions supplémentaires**

  Euler rétrograde: $\tau_n(h) = \frac{1}{2}h|y''(\theta_n)|$

  Crank-Nicolson: $\tau_n(h) = \frac{1}{12}h^2|y'''(\theta_n)|$