Séance II : Analyse numérique des équations différentielles

A) Objectifs de la séance

A la fin de cette séance,

- Je sais définir le schéma d'Euler explicite.
- Je sais définir le schéma d'Euler implicite.
- Je sais définir la consistance/l'erreur de consistance d'un schéma.
- Je sais montrer que les schémas d'Euler sont consistants avec l'EDO.
- Je sais programmer le schéma d'Euler explicite en Python ou en Matlab.
- Je sais déterminer numériquement l'ordre d'une méthode.

B) Pour se familiariser avec les concepts (à traiter avant les séances de TD)

Les questions II.1 et II.2 sont à traiter avant la deuxième séance de TD. Les corrigés sont disponibles sur internet. *Ne regardez pas* les solutions avant d'avoir fini de travailler sur ces questions.

Question II.1
On considère l'EDO

$$\begin{cases} x' &= -y \\ y' &= x - y + z \\ z' &= x - 2y + 2z \end{cases}$$

où x, y et z sont des fonctions définies sur $\mathbb{R}$.

Q. II.1.1 Soient $x(0) = x^0$, $y(0) = y^0$ et $z(0) = z^0$ donnés. Trouver une solution explicite pour x, y et z.

Q. II.1.2 Donner une condition nécessaire et suffisante sur x^0, y^0 et z^0 pour que les solutions x, y et z soient périodiques.

Question II.2
On considère le problème de Cauchy

$$\begin{cases} y'(t) = -3y(t) \\ y(0) = 1 \end{cases}.$$

Q. II.2.1 Trouver une solution explicite y.

Q. II.2.2 Implémenter la méthode d'Euler explicite en Python pour approcher y sur $[0, 10]$. Utiliser les pas $h = 1$, $h = 0.1$, $h = 0.01$ et calculer l'erreur globale dans chaque cas.

Q. II.2.3 Implémenter la méthode d'Euler implicite en Python pour approcher y sur $[0, 10]$. Utiliser les pas $h = 1$, $h = 0.1$, $h = 0.01$ et calculer l'erreur globale dans chaque cas.

Q. II.2.4 Conclure.

C) Exercices

Exercice II.1
En 1963, Edward N. Lorenz developpa un modèle très simplifié de l'atmosphère sur la base de trois variables :

- x l'intensité du mouvement de convection,
- y la différence de température horizontale,
- z la différence de température verticale,

et les liens entre ces variables :

$$\begin{cases} x' &= \sigma(y - x), \\ y' &= x(\rho - z) - y, \\ z' &= xy - \beta z. \end{cases}$$

où

σ est un paramètre comparant la rapidité des phénomènes thermiques et des phénomènes hydrodynamiques,
ρ est un paramètre concernant le transfert de chaleur,
β est un paramètre relatif aux dimensions de la couche d'atmosphère.

Nous prendrons désormais $\sigma = 10$, $\rho = 28$ et $\beta = 8/3$, choix fait par Edward Lorenz.

E. II.1.1 Trouver un ou une élève ayant choisi l'électif science des transferts en SG1, l'inviter à déjeuner et lui demander de vous expliquer les différents paramètres et la modélisation de Lorenz.

E. II.1.2 Programmer la méthode d'Euler en Python pour approcher les solutions sur $[0, 50]$ avec les conditions initiales $x(0) = 1$, $y(0) = 1$ et $z(0) = 1$. Vous êtes invité(e) à choisir un pas $\Delta t = 0.01$.

E. II.1.3 Modifier les valeurs des conditions initiales. Que remarquez vous ?

Exercice II.2
Soient $(\alpha_n)_{n\in\mathbb{N}}$ et $(\beta_n)_{n\in\mathbb{N}}$ des suites positives. Soit $q \in \mathbb{R}^+$. On suppose

$$\forall n \in \mathbb{N}^*,\ \alpha_{n+1} \leq q\alpha_n + \beta_n.$$

Démontrer que

$$\forall n \in \mathbb{N}^*,\ \alpha_n \leq q^n\alpha_0 + \sum_{k=0}^{n-1} q^{n-1-k}\beta_k.$$

Exercice II.3
Pour un champ de vecteur f, le schéma de Crank-Nicolson s'écrit

$$\begin{cases} z^0 \text{ donné} \\ z^{n+1} = z^n + \dfrac{\Delta t}{2} f(t^n, z^n) + \dfrac{\Delta t}{2} f(t^{n+1}, z^{n+1}). \end{cases}$$

Démontrer que le schéma de Crank-Nicolson est convergent au second ordre.

D) Approfondissement

Dans la suite du TD, on s'intéresse au problème suivant : soient $d \geq 1$, I un intervalle de $\mathbb{R}$ contenant 0, $f : (t, y) \in I \times \mathbb{R}^d \mapsto f(t, y) \in \mathbb{R}^d$ un champ de vecteur supposé de classe C^1 et globalement lipschitzien de constante L. On considère le problème de Cauchy suivant :

$$\begin{cases} y' = f(t, y), \\ y(0) = y^0 \end{cases} \tag{II.1}$$

pour $y^0 \in \mathbb{R}^d$. Ce problème admet une et une seule solution dans $C^1(I)$. Soit $T > 0$ tel que $[0, T] \subset I$. Pour $N \geq 1$, on pose $\Delta t = T/N$ et $t^n := n\Delta T$, $n \in \{0, \ldots, N\}$. Soit $z^0 \in \mathbb{R}^d$.

Exercice II.4
On définit une méthode à un pas par

$$\forall n \in \{0, \dots, N-1\}, \quad z^{n+1} = z^n + \Delta t\, \Phi(t^n, \Delta t, z^n), \tag{II.2}$$

où Φ est une fonction donnée.

E. II.4.1 Écrire le schéma d'Euler explicite vu en cours sous cette forme. À quelle condition peut-on écrire le schéma d'Euler implicite sous cette forme ?

E. II.4.2 Rappeler la définition de la consistance de (II.2) avec (II.1).

E. II.4.3 (**) Donner et démontrer une condition nécessaire et suffisante de consistance de la méthode associée à Φ de (II.2) avec (II.1).

E. II.4.4 Rappeler la définition de stabilité.

E. II.4.5 Montrer que l'hypothèse "Φ globalement lipschitzien par rapport à z" de constante K assure que la méthode associée à Φ est stable.

E. II.4.6 Prouver le théorème de Lax :

Théorème. *Une méthode à un pas consistante (II.2) avec l'EDO $y' = f(t, y)$ et stable est convergente.*

Chapitre II : Corrections des exercices

Solution de Q. II.1.1 Soit

$$A = \begin{pmatrix} 0 & -1 & 0 \\ 1 & -1 & 1 \\ 1 & -2 & 2 \end{pmatrix} \quad U = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Le système s'écrit $U' = AU$. Les valeurs propres de A sont :

$$\lambda_1 = 1,\ \lambda_2 = i,\ \lambda_3 = -i$$

et les vecteurs propres associés sont respectivement

$$v_1 = \begin{pmatrix} -1 \\ 1 \\ 3 \end{pmatrix},\ v_2 = \begin{pmatrix} i \\ 1 \\ 1 \end{pmatrix},\ v_3 = \begin{pmatrix} -i \\ 1 \\ 1 \end{pmatrix}$$

Soit

$$P^{-1} = [v_1, v_2, v_3]\ ,\ D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & i & 0 \\ 0 & 0 & -i \end{pmatrix}\ ,\ P = \frac{1}{4}\begin{pmatrix} 0 & -2 & 2 \\ -2i & -3+i & -1-i \\ 2i & 3-i & -1+i \end{pmatrix}$$

On a $A = P^{-1}DP$, par suite

$$\exp(tA) = P^{-1}\begin{pmatrix} e^t & 0 & 0 \\ 0 & e^{it} & 0 \\ 0 & 0 & e^{-it} \end{pmatrix} P$$

En utilisant $e^{it} = \cos(t) + i\sin(t)$ on obtient

$$\exp(tA) = \frac{1}{2}\begin{pmatrix} 2\cos(t) & e^t - \cos(t) - 3\sin(t) & -e^t + \cos(t) + \sin(t) \\ 2\sin(t) & -e^t + 3\cos(t) - \sin(t) & e^t - \cos(t) + \sin(t) \\ 2\sin(t) & -3e^t + 3\cos(t) - \sin(t) & 3e^t - \cos(t) + \sin(t) \end{pmatrix}$$

Ainsi

$$\begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2\cos(t) & e^t - \cos(t) - 3\sin(t) & -e^t + \cos(t) + \sin(t) \\ 2\sin(t) & -e^t + 3\cos(t) - \sin(t) & e^t - \cos(t) + \sin(t) \\ 2\sin(t) & -3e^t + 3\cos(t) - \sin(t) & 3e^t - \cos(t) + \sin(t) \end{pmatrix}\begin{pmatrix} x^0 \\ y^0 \\ z^0 \end{pmatrix}$$

Finalement

$$\begin{cases} x(t) = \frac{1}{2}(y^0 - z^0)e^t + \frac{1}{2}(2x^0 - y^0 + z^0)\cos(t) + \frac{1}{2}(-3y^0 + z^0)\sin(t) \\ y(t) = -\frac{1}{2}(y^0 - z^0)e^t + \frac{1}{2}(3y^0 - z^0)\cos(t) + \frac{1}{2}(2x^0 - y^0 + z^0)\sin(t) \\ z(t) = -\frac{3}{2}(y^0 - z^0)e^t + \frac{1}{2}(3y^0 - z^0)\cos(t) + \frac{1}{2}(2x^0 - y^0 + z^0)\sin(t) \end{cases}$$

Solution de Q. II.1.2 $y^0 = z^0$.

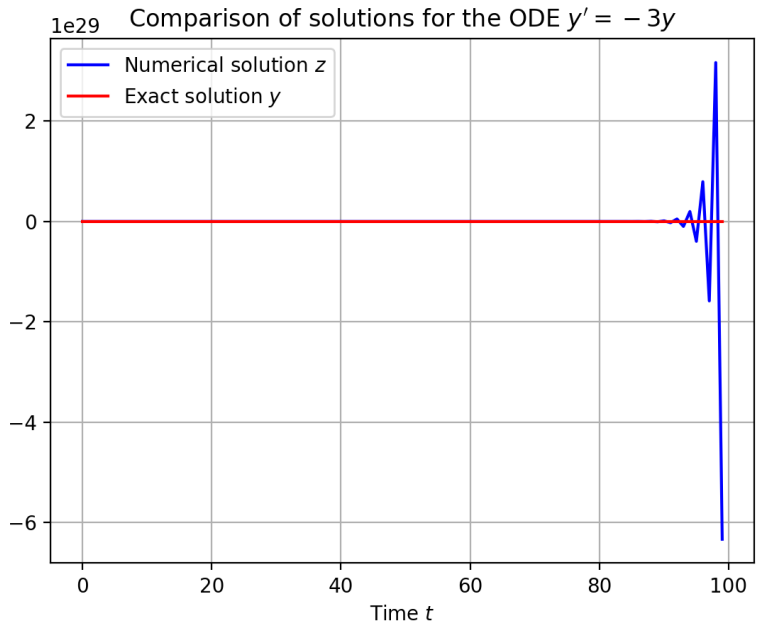
Solution de Q. II.2.1 La solution est $y(t) = \exp(-3t)$.

Solution de Q. II.2.2

```
# -*- coding: utf-8 -*-
"""
The Euler Forward Method
CentraleSupelec 2018-2019
John Cagnol and Corentin Jeudy
"""


import math
import numpy as np
import matplotlib.pyplot as plt


##############################################################################
# DEFINE THE ODE                                                             #
##############################################################################

# y' = f(t,y)
# y(t0) = t0

# Define the function f
def f(time,x):
    """
    - brief: Computes the value for the ODE defined by y' = -3y
    - input: (float) time, (float) x
    - output: (float) -3 * x
    """
    return -3 * x

# Define t0 and y0
time_origin = 0
initial_condition = 1


##############################################################################
# DEFINE THE MESH                                                            #
##############################################################################

simulation_time = 100     # End of the interval [t0,T]
number_iterations = 100     # Number of iterations
time_step = simulation_time/number_iterations

# Define the arrays holding the values of t and y at each point of the mesh
time_array = np.empty(number_iterations)
numerical_solution = np.empty(number_iterations)

print("Time origin t0 = ",time_origin)
print("simulation time T = ",simulation_time)
print("Time step dt = ",time_step)


##############################################################################
# APPROXIMATE THE SOLUTION USING THE EULER FORWARD METHOD                    #
##############################################################################

numerical_solution[time_origin] = initial_condition
```

```
for i in range(0, number_iterations - 1):
    time_array[i+1] = time_array[i] + time_step
    numerical_solution[i+1] = numerical_solution[i] + time_step*f(time_array[i],numerical_solution[i])


##############################################################################
# COMPARE THE APPROXIMATE SOLUTION TO THE EXACT SOLUTION                     #
##############################################################################

# Array to hold the values of the exact solution at each point of the mesh
exact_solution = np.zeros([number_iterations])

# Array to hold the difference between exact and approximate solution (error)
error_array = np.zeros([number_iterations])

# Compute the exact solution at each point of the mesh, and the error.
for i in range(0, number_iterations - 1):
    exact_solution[i] = initial_condition * math.exp( -3 * time_array[i])
    error_array[i] = abs(numerical_solution[i] - exact_solution[i])

print("Error max: {}".format(error_array.max()))


##############################################################################
# PLOT THE APPROXIMATE SOLUTION AND THE EXACT SOLUTION                       #
##############################################################################

figure = plt.figure()

plt.title("Comparison of solutions for the ODE $y' = -3y$")

plt.grid(True)
plt.xlabel("Time $t$")

plt.plot(time_array,numerical_solution,'b', label = "Numerical solution $z$")
plt.plot(time_array,exact_solution,'r', label = "Exact solution $y$")
plt.legend(loc = "upper left")

plt.show()
```

https://github.com/cagnol/CIPPDE/blob/master/EFM.py




```
Time origin t0 =  0
simulation time T =  100
Time step dt =  1.0
Error max: 3.1691265005705735e+29
```

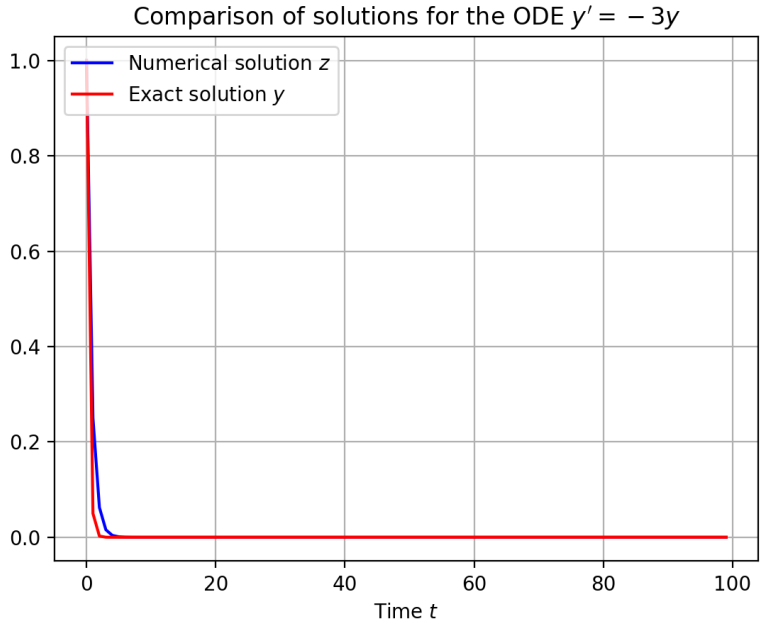
Solution de Q. II.2.3

```
# -*- coding: utf-8 -*-
"""
The Euler Backward Method
CentraleSupelec 2018-2019
John Cagnol and Corentin Jeudy
"""

import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve


##############################################################################
# DEFINE THE ODE                                                             #
##############################################################################

# y' = f(t,y)
# y(t0) = t0

# Define the function f
def f(time,x):
    """
    - brief: Computes the value for the ODE defined by y' = -3y
```

```
    - input: (float) time, (float) x
    - output: (float) -3 * x
    """
    return -3 * x

# Define t0 and y0
time_origin = 0
initial_condition = 1


##############################################################################
# DEFINE THE MESH                                                            #
##############################################################################

simulation_time = 100     # End of the interval [t0,T]
number_iterations = 100     # Number of iterations
time_step = simulation_time/number_iterations

# Define the arrays holding the values of t and y at each point of the mesh
time_array = np.empty(number_iterations)
numerical_solution = np.empty(number_iterations)

print("Time origin t0 = ",time_origin)
print("simulation time T = ",simulation_time)
print("Time step dt = ",time_step)


##############################################################################
# APPROXIMATE THE SOLUTION USING THE EULER BACKWARD METHOD                   #
##############################################################################

numerical_solution[time_origin] = initial_condition

for i in range(0, number_iterations - 1):
    time_array[i+1] = time_array[i] + time_step

    # We need to solve z[i+1] = z[i] + dt * f(t[i+1],z[i+1])
    # Let X = z[i+1] be the unknown
    # We need to find the root of g(X) = X - (z[i] + Dt*f(t[i+1],X)
    g = lambda X : X - (numerical_solution[i] + time_step*f(time_array[i+1],X))

    # Solve for g(X) = 0 for X and guess the solution is close to z[i]
    numerical_solution[i+1] = fsolve(g,numerical_solution[i])



##############################################################################
# COMPARE THE APPROXIMATE SOLUTION TO THE EXACT SOLUTION                     #
##############################################################################

# Array to hold the values of the exact solution at each point of the mesh
exact_solution = np.zeros([number_iterations])

# Array to hold the difference between exact and approximate solution (error)
error_array = np.zeros([number_iterations])
```

```
# Compute the exact solution at each point of the mesh, and the error.
for i in range(0, number_iterations - 1):
    exact_solution[i] = initial_condition * math.exp( -3 * time_array[i])
    error_array[i] = abs(numerical_solution[i] - exact_solution[i])

print("Error max: {}".format(error_array.max()))


##########################################################################
# PLOT THE APPROXIMATE SOLUTION AND THE EXACT SOLUTION                   #
##########################################################################

figure = plt.figure()

plt.title("Comparison of solutions for the ODE $y' = -3y$")

plt.grid(True)
plt.xlabel("Time $t$")

plt.plot(time_array,numerical_solution,'b', label = "Numerical solution $z$")
plt.plot(time_array,exact_solution,'r', label = "Exact solution $y$")
plt.legend(loc = "upper left")

plt.show()
```

https://github.com/cagnol/CIPPDE/blob/master/EBM.py




```
Time origin t0 =  0
simulation time T =  100
Time step dt =  1.0
Error max: 0.20021293163213605
```

Solution de Q. II.2.4 Les questions précédentes illustrent que pour Euler, la méthode implicite est A-stable ce qui n'est pas le cas de la méthode explicite.

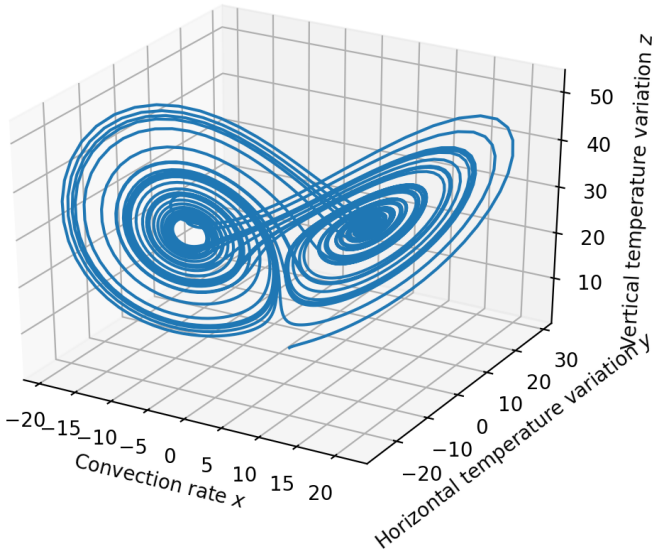
Solution de Q. II.1.1 Nous esperons que votre déjeuner s'est bien déroulé.

Solution de Q. II.1.2

```
# -*- coding: utf-8 -*-
"""
Lorenz
CentraleSupelec 2018-2019
John Cagnol and Corentin Jeudy
"""

"""
IN THE CODE:
- CR stands for Convection Rate
- HTV stands for Horizontal Temperature Variation
- VTV stands for Vertical Temperature Variation

These abbreviations are solely used to make the improve the code readability.
"""

import math
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


# Define the parameters
sigma = 10.0
rho   = 28.0
beta  = 8.0/3.0

# Define t0 and the initial conditions
time_origin = 0
initial_CR = 1.0
initial_HTV = 1.0
initial_VTV = 1.0

# Define de Mesh and print its characteristics
simulation_time = 50     # End of the interval [t0,T]
number_iterations = 5000     # Number of iterations
time_step = simulation_time/number_iterations
print("Time origin t0 = ",time_origin)
print("simulation time T = ",simulation_time)
print("Time step dt = ",time_step)

# Define the arrays holding the values of t and y at each point of the mesh
time_array = np.empty(number_iterations)
CR = np.empty(number_iterations)
HTV = np.empty(number_iterations)
VTV = np.empty(number_iterations)
```

```
# Approximate the solution using the Euler Forward Method
time_array[0] = time_origin
CR[0] = initial_CR
HTV[0] = initial_HTV
VTV[0] = initial_VTV

for i in range(0, number_iterations - 1):
    time_array[i+1] = time_array[i] + time_step
    CR[i+1] = CR[i] + time_step*sigma*(HTV[i] - CR[i])
    HTV[i+1] = HTV[i] + time_step*(CR[i]*(rho - VTV[i]) - HTV[i])
    VTV[i+1] = VTV[i] + time_step*(CR[i]*HTV[i] - beta*VTV[i])

# Plot the (approximate) solution
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.set_xlabel("Convection rate $x$")
ax.set_ylabel("Horizontal temperature variation $y$")
ax.set_zlabel("Vertical temperature variation $z$")
ax.plot(CR, HTV, VTV)
plt.show()
```

https://github.com/cagnol/CIPPDE/blob/master/lorenz.py




```
Time origin t0 =  0
simulation time T =  50
Time step dt =  0.01
```

Solution de Q. II.1.3 Une petite variation des conditions initiales change la solution de manière importante. Le système est chaotique.

Solution de Q. II.2 Raisonnons par récurrence.

- L'initialisation (H_1) est satisfaite car $\alpha_1 \leq q\alpha_0 + \beta$
- Suppons (H_n), alors

$$\alpha_n \leq q^n \alpha_0 + \sum_{k=0}^{n-1} q^{n-1-k}\beta_k$$

$$q\alpha_n \leq q^{n+1}\alpha_0 + \sum_{k=0}^{n-1} q^{n-k}\beta_k$$

$$q\alpha_n + \beta_n \leq q^{n+1}\alpha_0 + \sum_{k=0}^{n} q^{n-k}\beta_k$$

ce qui implique (H_{n+1}).

Solution de Q. II.3

Consistance : Soit

$$\varepsilon^n = y(t^{n+1}) - \left[y(t^n) + \frac{\Delta t}{2} f(t^n, y(t^n)) + \frac{\Delta t}{2} f(t^{n+1}, y(t^n + \Delta t))\right]$$

En utilisant

$$\begin{aligned} y(t^{n+1}) &= y(t^n + \Delta t) \\ &= y(t^n) + \Delta t\, y'(t^n) + \frac{\Delta t}{2} y''(t^n) + O(\Delta t^3) \\ f(t^n, y(t^n)) &= y'(t^n) \\ f(t^n + \Delta t, y(t^n + \Delta t)) &= y'(t^{n+1}) \\ &= y'(t^n + \Delta t) \\ &= y'(t^n) + \Delta t\, y''(t^n) + O(\Delta t^2) \end{aligned}$$

il vient

$$\varepsilon^n = y(t^n) + \Delta t\, y'(t^n) + \frac{\Delta t}{2} y''(t^n) + O(\Delta t^3) - \left[y(t^n) + \frac{\Delta t}{2} y'(t^n) + \frac{\Delta t}{2}\left(y'(t^n) + \Delta t\, y''(t^n) + O(\Delta t^2)\right)\right]$$

Ainsi

$$\varepsilon^n = O(\Delta t^3)$$

donc

$$\sum_{n=0}^{N-1} \|\varepsilon^n\| = \sum_{n=0}^{N-1} O(\Delta t^3) = N\, O(\Delta t^3) = \frac{T}{\Delta t}\, O(\Delta t^3) = O(\Delta t^2)$$

Il en résulte que la méthode est consistante d'ordre 2.

Stabilité : On considère

$$\begin{aligned} z^{n+1} &= z^n + \tfrac{\Delta t}{2}[f(t^n, z^n) + f(t^{n+1}, z^{n+1})] \\ w^{n+1} &= w^n + \tfrac{\Delta t}{2}[f(t^n, w^n) + f(t^{n+1}, w^{n+1})] + \eta^n. \end{aligned}$$

Soit $\kappa > 0$ la constante de Lipschitz de f, alors

$$\|z^{n+1} - w^{n+1}\| \leqslant \|z^n - w^n\| + \frac{\kappa \Delta t}{2}[\|z^n - w^n\| + \|z^{n+1} - w^{n+1}\|] + \|\eta^n\|$$

qui donne

$$\|z^{n+1} - w^{n+1}\| \leqslant \frac{1 + \kappa \Delta t/2}{1 - \kappa \Delta t/2}\|z^n - w^n\| + \frac{1}{1 - \kappa \Delta t/2}\|\eta^n\|$$

où $\frac{1}{1-\kappa \Delta t/2} \leqslant 2$ pour Δt assez petit.
En faisant des calculs, on peut montrer que la fonction

$$t \mapsto \frac{1 + \kappa t/2}{1 - \kappa t/2}$$

est inférieure à $t \mapsto e^{Lt}$ sur $[0, 1]$ pour une constante $L = L(\kappa)$ assez grande.
Par conséquent, on a l'inégalité

$$\|z^{n+1} - w^{n+1}\| \leqslant e^{L\Delta t}\|z^n - w^n\| + 2\|\eta^n\|.$$

En appliquant le lemme de Gronwall (II.5.13) , on obtient

$$\|z^n - w^n\| \leqslant 2e^{LT}\left(\|z^0 - w^0\| + \sum_{k=0}^{N-1} \|\eta^k\|\right)$$

D'où la stabilité.
Merci à Brice Hannebicque pour la rédaction de cette partie.

Convergence : Si le schéma numérique est stable et consistant, il est convergent en vertu du théorème de Lax.

Solution de Q. II.4.1

- Euler explicite : $\Phi : (t, \Delta t, z) \mapsto f(t, z)$.
- Euler implicite : le schéma s'écrit pour tout $n \in \{0, \ldots, N\}$,

  $$z^{n+1} = z^n + \Delta t f(t^{n+1}, z^{n+1})$$

  soit encore

  $$z^{n+1} - \Delta t f(t^{n+1}, z^{n+1}) = z^n.$$

  Il faut donc pouvoir définir, au moins localement, une fonction inverse de $z \mapsto z - \Delta t f(t^{n+1}, z)$ (voir le théorème d'inversion locale). En général, cette fonction **n'est pas explicite**, d'où le nom du schéma... On a donc recours à l'algorithme de Newton pour trouver z^{n+1}.

Solution de Q. II.4.2 Voir cours..

Solution de Q. II.4.3 Soit $y \in C^1(I)$ la solution globale de (II.1), dont l'existence et l'unicité est assurée par le théorème de Cauchy-Lipschitz.
On suppose qu'il existe $c > 0$ tel que Φ soit dérivable par rapport à Δt sur $[0, c]$.
Calculons l'erreur de consistance de la méthode (II.2) :
Alors, pour $n \in \{0, \dots, N\}$,

$$\varepsilon^n = y(t^{n+1}) - y(t^n) - \Delta t\, \Phi(t^n, \Delta t, y(t^n)).$$

Alors

$$\begin{aligned}\varepsilon^n &= y(t^{n+1}) - y(t^n) - \Delta t \Phi(t^n, \Delta t, y(t^n)) \\ &= \Delta t y'(t^n) + \Delta t h(\Delta t) - \Delta t \Phi(t^n, 0, y(t^n)) - \Delta t \psi(\Delta t) \\ &= \Delta t (f(t^n, y(t^n)) - \Phi(t^n, 0, y(t^n))) + \Delta t (h(\Delta t) - \psi(\Delta t))\end{aligned}$$

où h et ψ sont des fonctions continues valant 0 en 0 et où on utilise le fait que y est la solution de (II.1).
Calculons la somme des erreurs de consistance :

$$\left| \sum_{n=0}^{N-1} \|\varepsilon^n\| - \sum_{n=0}^{N-1} \Delta t \|f(t^n, y(t^n)) - \Phi(t^n, 0, y(t^n))\| \right| \leq \sum_{n=0}^{N-1} \Delta t \|h(\Delta t) - \psi(\Delta t)\|.$$

Or h et ψ tendent vers 0 en 0.
Soit $\epsilon > 0$. Alors il existe $\eta > 0$ tel que, $\forall \Delta t \in]0, \eta[$, $\|h(\Delta t) - \psi(\Delta t)\| \leq \epsilon$. Supposons que $\Delta t \leq \eta$.
Alors

$$\sum_{n=0}^{N-1} \Delta t \|h(\Delta t) - \psi(\Delta t)\| \leq \varepsilon \sum_{n=0}^{N-1} \Delta t = T\varepsilon.$$

On a donc

$$\lim_{\Delta t \to 0} \sum_{n=0}^{N-1} \|\varepsilon^n\| = \lim_{\Delta t \to 0} \sum_{n=0}^{N-1} \Delta t \|f(t^n, y(t^n)) - \Phi(t^n, 0, y(t^n))\| = \int_0^T \|f(t, y(t)) - \Phi(t, 0, y(t))\| \mathrm{d}t.$$

La méthode est consistante avec (II.1) si et seulement si

$$\sum_{n=0}^{N-1} \|\varepsilon^n\| \underset{N \to +\infty}{\longrightarrow} 0$$

pour toute solution y, c'est-à-dire si et seulement si $f(t, z) - \Phi(t, 0, z) = 0$ pour tout $(t, z) \in I \times \mathcal{U}$.

Solution de Q. II.4.4 voir cours.

Solution de Q. II.4.5 Soit $(\eta^n)_{n \in \{0, \dots, N\}} \subset \mathcal{U}$ donnée. On définit

$$\begin{aligned}\forall n \in \{0, \dots, N-1\}, \quad z^{n+1} &= z^n + \Delta t\, \Phi(t^n, \Delta t, z^n) \\ w^{n+1} &= w^n + \Delta t \Phi(t^n, \Delta t, w^n) + \eta^n.\end{aligned}$$

Alors, pour tout $n \in \{0, \dots, N\}$,

$$z^{n+1} - w^{n+1} = z^n - w^n + \Delta t(\Phi(t^n, \Delta t, z^n) - \Phi(t^n, \Delta t, w^n)) - \eta^n.$$

Donc

$$\|z^{n+1} - w^{n+1}\| \leq (1 + K\Delta t)\|z^n - w^n\| + \|\eta^n\| \leq \exp(K\Delta t)\|z^n - w^n\| + \|\eta^n\|.$$

En utilisant Montrons que, si une suite $(\alpha^n)_n \subset \mathbb{R}^+$ satisfait l'inégalité arithmético-géométrique suivante

$$\forall n \geq 0, \quad \alpha^{n+1} \leq \exp(K\Delta t)\alpha^n + \beta^n$$

où $(\beta^n)_n$ est une suite donnée à valeurs positives, alors

$$\forall n \geq 0, \quad \alpha^n \leq \exp(K(t^n - t^0))\alpha^0 + \sum_{k=0}^{n-1} \exp(K(t^n - t^{k+1}))\beta^k. \tag{II.3}$$

En effet, en écrivant, pour $n \geq 0$,

$$\begin{aligned}
\alpha^n &\leq \exp(K\Delta t)\alpha^{n-1} + \beta^{n-1} && (\times\ 1)\\
\alpha^{n-1} &\leq \exp(K\Delta t)\alpha^{n-2} + \beta^{n-2} && (\times\ \exp(K\Delta t))\\
&\vdots\\
\alpha^1 &\leq \exp(K\Delta t)\alpha^0 + \beta^0 && (\times\ \exp(K\Delta t)\dots\exp(K\Delta t))
\end{aligned}$$

puis en opérant les multiplications entre parenthèses et en sommant, on obtient (II.3). On a donc, pour tout $n \in \{0, \dots, N\}$,

$$\|z^n - w^n\| \leq \exp(KT)\left(\|z^0 - w^0\| + \sum_{k=0}^{n-1} \|\eta^k\|\right).$$

La méthode est donc stable.

Solution de Q. II.4.6 Soit $N \in \mathbb{N}$. Soit une méthode à un pas définie par Φ, supposée consistante et stable de constante K.
Considérons $(w^n)_{n\in\{0,\dots,N\}}$ définie par $w^n = y(t^n)\ \forall n \in \{0, \dots, N\}$. D'après la définition de l'erreur de consistance, on a

$$\forall n \in \{0, \dots, N\}, \quad w^{n+1} = w^n + \Delta t\Phi(t^n, \Delta t, w^n) + \varepsilon^n.$$

D'après la définition de la stabilité, pour $n \in \{0, \dots, N\}$, l'erreur de convergence $e^n = w^n - z^n$ peut être estimée par

$$\|e^n\| \leq e^{Kn\Delta t} \sum_{k=0}^{n-1} \|\varepsilon^k\|.$$

Par consistance, le schéma est donc convergent.