Analyse Numérique
Equations Différentielles Ordinaires

Simone Deparis

May 17, 2023

Contents

1 Equations Différentielles Ordinaires

1.1 Problème de Cauchy

$f : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$ continue, $y_0 \in$ donné. On cherche $y : t \in I \subset_+ \to y(t) \in$ qui satisfait le problème suivant

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

où $y'(t) = \dfrac{dy(t)}{dt}$.

Exemple Écrivez la discretisation par la méthode d'Euler progressive et rétrograde du problème de Cauchy

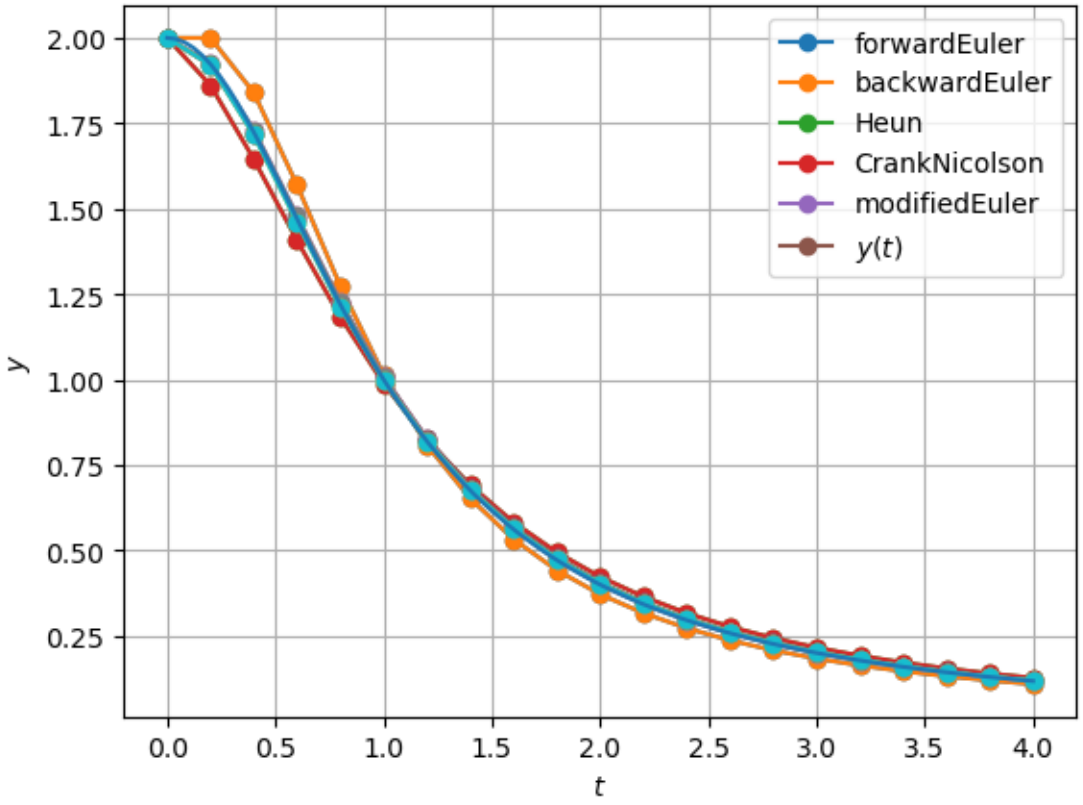
$$\begin{cases} y'(t) = -t\, y(t)^2 & \forall t \in [0, 4] \\ y(t_0) = 2 \end{cases}$$

La solution de ce problème est $y(t) = \frac{2}{1+t^2}$ Avec les méthodes de Euler Proressive et Rétrograde, Heun, Crank-Nicolson, et Euler modifié.

[1]:
```
# importing libraries used in this book
import numpy as np
import matplotlib.pyplot as plt

from OrdinaryDifferentialEquationsLib import␣
↪forwardEuler,backwardEuler,Heun,CrankNicolson,modifiedEuler
```

[2]:
```
f = lambda t,x :   -t*x**2
y0 = 2; tspan=[0, 4]
```

```
Nh = 20

for method in [forwardEuler,backwardEuler,Heun,CrankNicolson,modifiedEuler] :

    t, y = method(f, tspan, y0, Nh)
    plt.plot(t, y,'o-')
    plt.plot(t, y,'o-')

y = lambda t : 2/(1+t**2)
t = np.linspace(tspan[0],tspan[1],100)
plt.plot(t, y(t),'-')
# labels, title, legend
plt.xlabel('$t$'); plt.ylabel('$y$')
plt.
 ↪legend(['forwardEuler','backwardEuler','Heun','CrankNicolson','modifiedEuler','$y(t)$'])
plt.grid(True)
plt.show()
```



1.2 Euler Progressif

Ecrivez une fonction `forwardEuler` qui approche la solution du problème

$$y'(t) = f(t, y(t)),\ t \in (T_0, T_f), \quad y(0) = y_0,$$

en utilisant la méthode d'Euler progressif. L'entête de la fonction doit être la suivante:

```
def forwardEuler( fun, interval, y0, N ) :
    #  FORWARDEULER Solve differential equations using the forward Euler method.
    #  [T, U] = FORWARDEULER( FUN, INTERVAL, Y0, N ), with INTERVAL = [T0, TF],
    #  integrates the system of differential equations y'=f(t, y) from time T0
    #  to time TF, with initial condition Y0, using the forward Euler method on
    #  an equispaced grid of N intervals. Function FUN(T, Y) must return
    #  a column vector corresponding to f(t, y). Each row in the solution
    #  array U corresponds to a time returned in the column vector T.
```

Une fois écrite la fonction `forwardEuler`, utilisez les commandes suivantes pour calculer la solution:

```
f = lambda t,x : ( 2/15*x*(1-x/1000) ) # C=2/15 et B = 1000
tsp = [0,100]
y0 = 100; Nh = 25;
t25,u25 = forwardEuler(f,tsp,y0,Nh)

plt.plot(t25,u25,'o')
```

Approchez la solution de l'équation différentielle

$$y'(t) = \frac{2\,y}{15}(1 - y/1000),\ t \in (0, 100), \quad y(0) = 100,$$

avec $N_h = 25$.

Que se passe-t-il avec $N_h = 7$? Discutez les résultats obtenus.

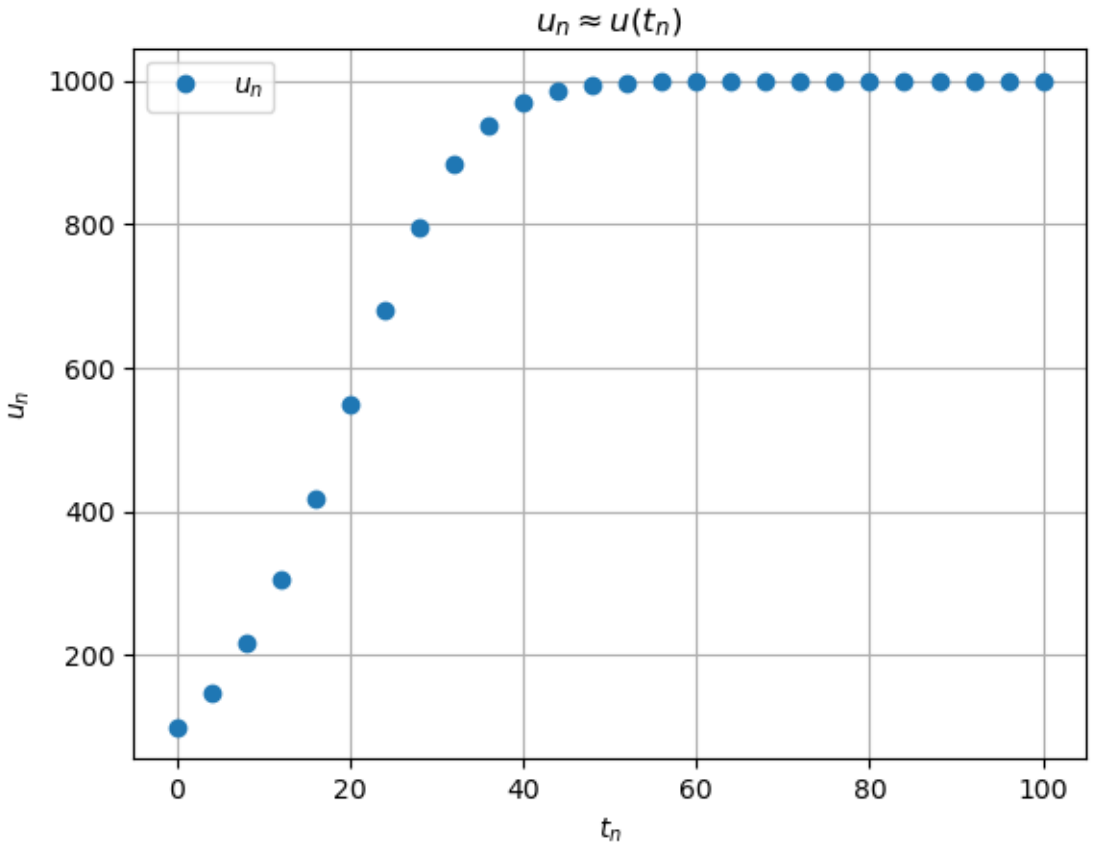
[3]:
```
# importing libraries used in this book
import numpy as np
import matplotlib.pyplot as plt
```

[4]:
```
def forwardEuler( fun, interval, y0, N ) :
    #  FORWARDEULER Solve differential equations using the forward Euler method.
    #  [T, U] = FORWARDEULER( FUN, INTERVAL, Y0, N ), with INTERVAL = [T0, TF],
    #  integrates the system of differential equations y'=f(t, y) from time T0
    #  to time TF, with initial condition Y0, using the forward Euler method on
    #  an equispaced grid of N intervals. Function FUN(T, Y) must return
    #  a column vector corresponding to f(t, y). Each row in the solution
    #  array U corresponds to a time returned in the column vector T.

    # time step
    h = ( interval[1] - interval[0] ) / N

    # time snapshots
    t = np.linspace( interval[0], interval[1], N+1 )
```

```
    # initialize the solution vector
    u = np.zeros(N+1)
    u[0] = y0

    # time loop (n=0,...,n, but array indeces in Matlab start at 1)
    for n in range(N) :
        u[n+1] = u[n] + h * fun( t[n], u[n] )

    return t, u
```

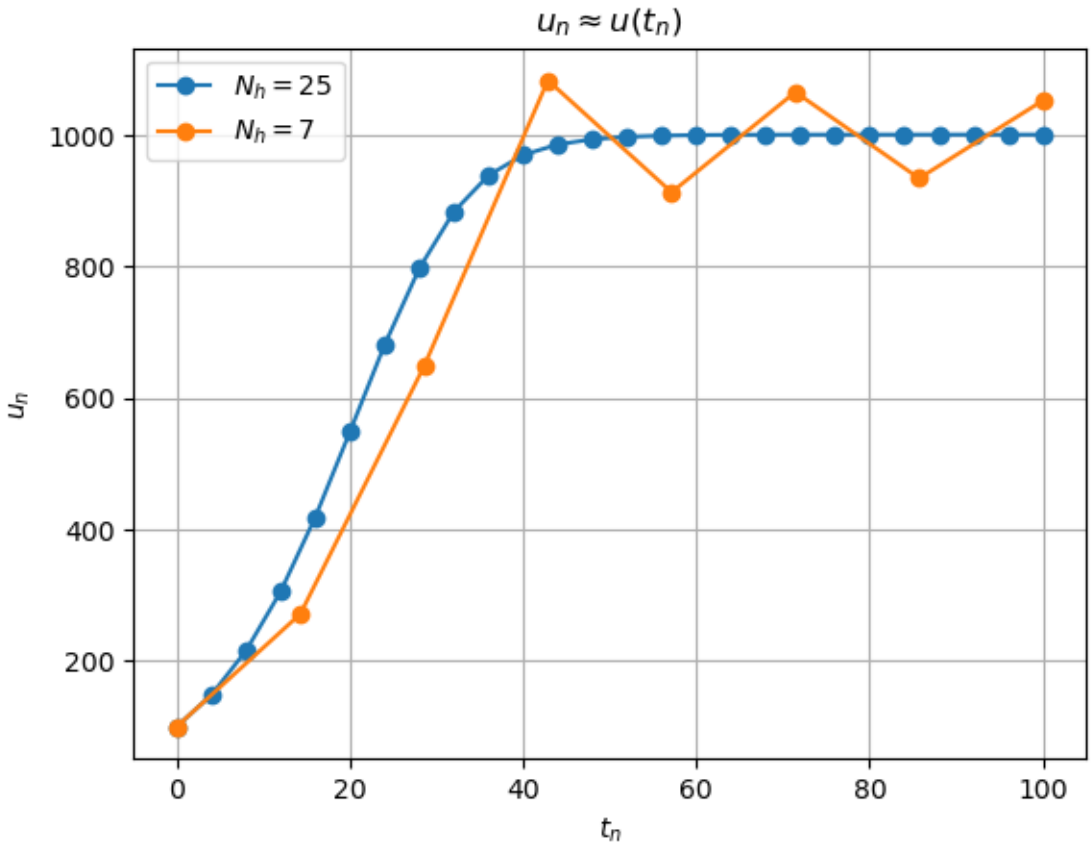
Nous pouvons alors résoudre l'exercice avec les commandes:

[5]:
```
f = lambda t,x : ( 2/15*x*(1-x/1000) ) # C=2/15 et B = 1000
tsp = [0,100]
y0 = 100; Nh = 25;
t25,u25 = forwardEuler(f,tsp,y0,Nh)

plt.plot(t25,u25,'o')

# labels, title, legend
plt.xlabel('$t_n$'); plt.ylabel('$u_n$'); #plt.title('data')
plt.legend(['$u_n$'])
plt.title('$u_n\\approx u(t_n)$')
plt.grid(True)
plt.show()
```



Pour résoudre le problème avec $N_h = 7$ et tracer le graphe de la solution, on utilise les commandes suivantes:

[6]:
```
Nh = 7
t7,u7 = forwardEuler(f,tsp,y0,Nh);

plt.plot(t25,u25,'o-')
plt.plot(t7,u7,'o-')

# labels, title, legend
plt.xlabel('$t_n$'); plt.ylabel('$u_n$'); #plt.title('data')
plt.legend(['$N_h=25$','$N_h=7$'])
plt.title('$u_n\\approx u(t_n)$')
plt.grid(True)
plt.show()
```



Solutions de l'équation différentielle $y'(t) = \frac{2y}{15}(1 - y/1000),\ t \in (0, 100), \quad y(0) = 100$ *pour diverses valeurs de* N_h.

On voit que la solution avec $N_h = 7$ est oscillante et ne tend pas vers 1000 lorsque $t \to \infty$: en effet, la condition de stabilité sur le pas de discretisation N_h n'est pas satisfaite.

[ ]:

1.3 Euler Retrograde

Ecrivez une fonction Matlab `backwardEuler` qui approche la solution du problème

$$y'(t) = f(t, y(t)),\ t \in (T_0, T_f), \quad y(0) = y_0,$$

en utilisant la méthode d'Euler progressif. L'entête de la fonction doit être la suivante:

```
def backwardEuler( fun, interval, y0, N ) :
    #  BACKWARDEULER Solve differential equations using the backward Euler method.
    #  [T, U] = BACKWARDEULER( FUN, INTERVAL, Y0, N ), with INTERVAL = [T0, TF],
    #  integrates the system of differential equations y'=f(t, y) from time T0
    #  to time TF, with initial condition Y0, using the backward Euler method on
    #  an equispaced grid of N intervals. Function FUN(T, Y) must return
```

```
    #  a column vector corresponding to f(t, y). Each row in the solution
    #  array U corresponds to a time returned in the column vector T.
    from scipy.optimize import fsolve
```

Vous disposez de la fonction `scipy.optimize.fsolve` pour résoudre une équation non-linéaire (`fsolve` cache plusieurs méthodes de resolution, comme Newton ou le point fixe).

Exemple d'utilisation de `fsolve`: on peut chercher le zéro de `F` près de `x0` en utilisant le code suivant:

```
from scipy.optimize import fsolve
a = 5; h = 0.1;
F = lambda x :  a + np.sin(x) - h*x;

x0 = a+h
zero = fsolve(F, x0)

print(f'F({zero[0]:5.2f}) = {F(zero)[0]:4.1e}')
```

Approchez la solution de l'équation différentielle

$$y'(t) = \frac{2\,y}{15}(1 - y/1000),\ t \in (0, 100), \quad y(0) = 100,$$

avec $N_h = 25$. Que se passe-t-il avec $N_h = 7$? Discuter les résultats obtenus.

[7]:
```
# importing libraries used in this book
import numpy as np
import matplotlib.pyplot as plt
```

[8]:
```
from scipy.optimize import fsolve
a = 5; h = 0.1;
F = lambda x :  a + np.sin(x) - h*x;

x0 = a+h
zero = fsolve(F, x0)

print(f'F({zero[0]:5.2f}) = {F(zero)[0]:4.1e}')
```

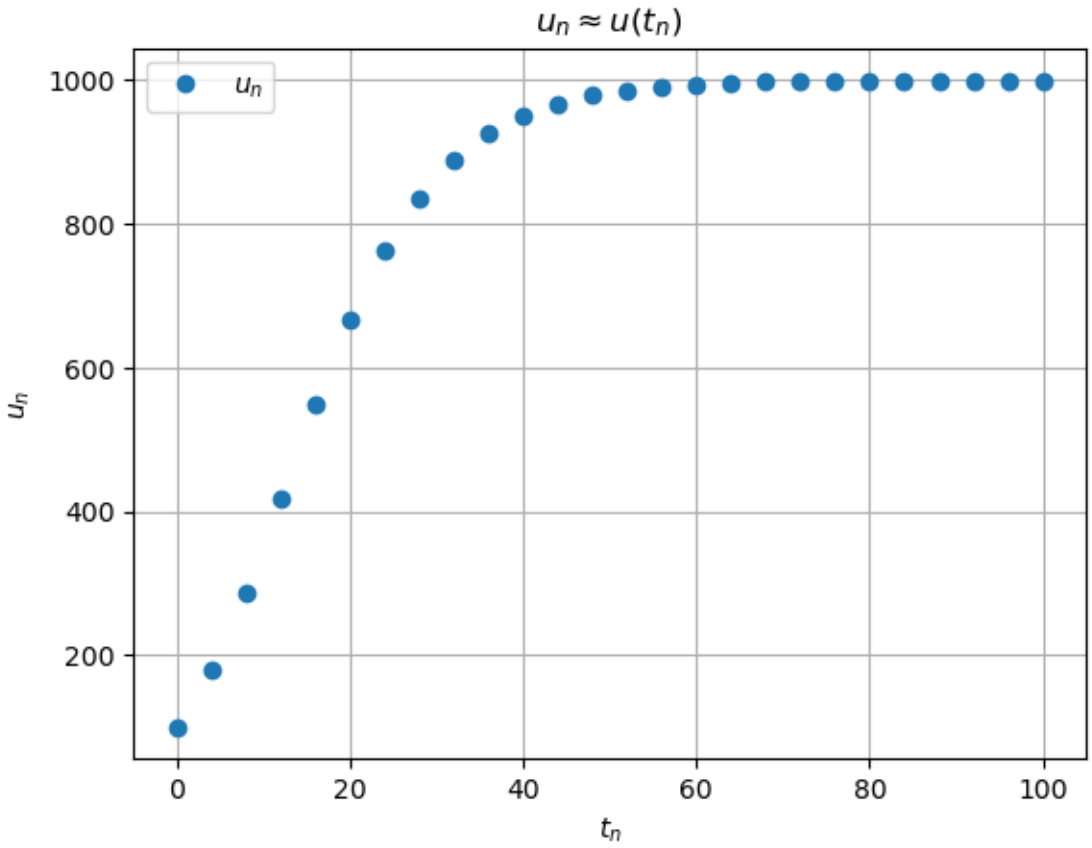
```
F(41.80) = -8.7e-13
```

[9]:
```
def backwardEuler( fun, interval, y0, N ) :
    #  BACKWARDEULER Solve differential equations using the backward Euler␣
 ↪method.
    #  [T, U] = BACKWARDEULER( FUN, INTERVAL, Y0, N ), with INTERVAL = [T0, TF],
    #  integrates the system of differential equations y'=f(t, y) from time T0
    #  to time TF, with initial condition Y0, using the backward Euler method on
    #  an equispaced grid of N intervals. Function FUN(T, Y) must return
    #  a column vector corresponding to f(t, y). Each row in the solution
    #  array U corresponds to a time returned in the column vector T.
    from scipy.optimize import fsolve
```

```
    # time step
    h = ( interval[1] - interval[0] ) / N

    # time snapshots
    t = np.linspace( interval[0], interval[1], N+1 )

    # initialize the solution vector
    u = np.zeros(N+1)
    u[0] = y0

    # time loop (n=0,...,n, but array indeces in Matlab start at 1)
    for n in range(N) :
        # non-linear function
        F = lambda x : u[n] + h * fun(t[n+1],x) - x
        # solve the non-linear equation using the built-in matlab function␣
↪"fsolve"
        # to compute u[n+1]
        u[n+1] = fsolve( F, u[n]);
        # u[n+1] = fsolve( F, u[n]+ h * fun( t[n], u[n] ));

        # NOTE:
        # in the call of fsolve, a more accurate initial guess is obtained
        # by replacing u[n] with the forward euler method:
        # u[n+1] = fsolve( F, u(n) + h * fun( t(n), u(n) ), options );

    return t, u
```

Nous pouvons alors résoudre l'exercice avec les commandes:

[10]:
```
f = lambda t,x : ( 2/15*x*(1-x/1000) ) # C=2/15 et B = 1000
tsp = [0,100]
y0 = 100; Nh = 25;
t25,u25 = backwardEuler(f,tsp,y0,Nh)

plt.plot(t25,u25,'o')

# labels, title, legend
plt.xlabel('$t_n$'); plt.ylabel('$u_n$'); #plt.title('data')
plt.legend(['$u_n$'])
plt.title('$u_n\\approx u(t_n)$')
plt.grid(True)
plt.show()
```

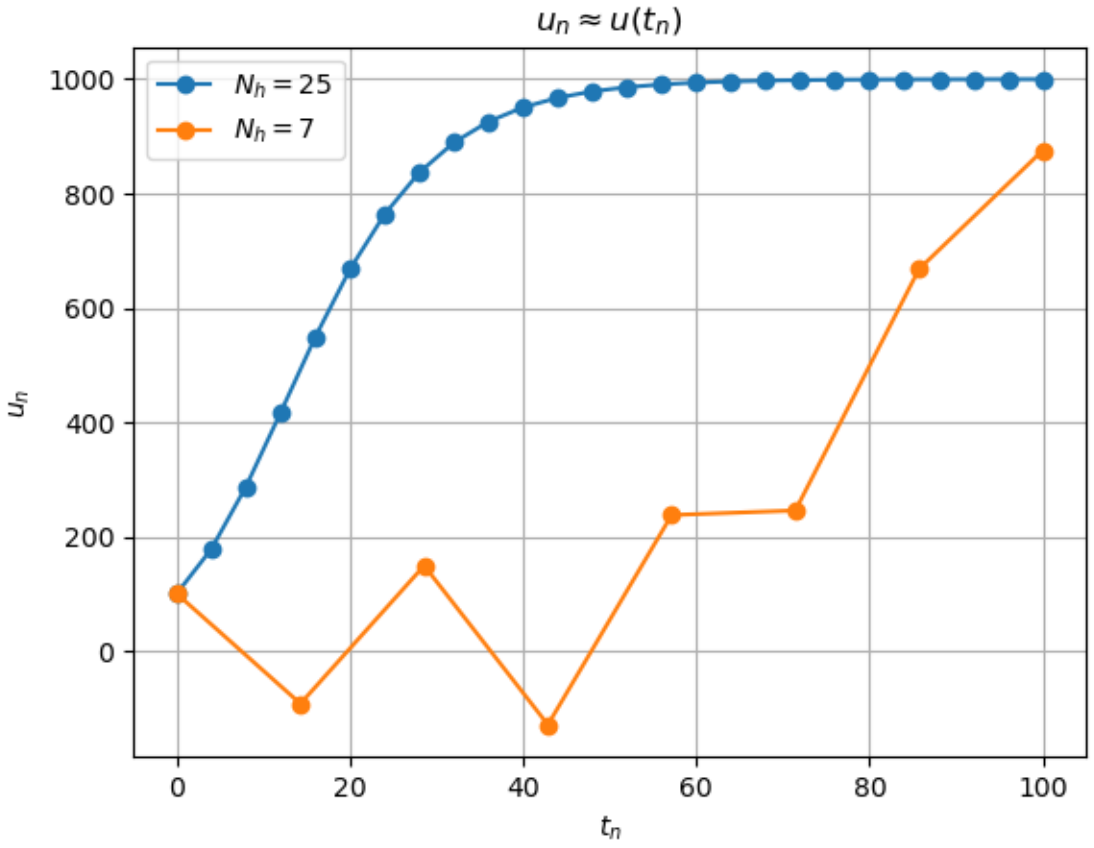


Pour résoudre le problème avec $N_h = 7$ et tracer le graphe de la solution, on utilise les commandes suivantes:

[11]:
```
Nh = 7
t7,u7 = backwardEuler(f,tsp,y0,Nh);

plt.plot(t25,u25,'o-')
plt.plot(t7,u7,'o-')

# labels, title, legend
plt.xlabel('$t_n$'); plt.ylabel('$u_n$'); #plt.title('data')
plt.legend(['$N_h=25$','$N_h=7$'])
plt.title('$u_n\\approx u(t_n)$')
plt.grid(True)
plt.show()
```

```
/Users/simone/opt/anaconda3/lib/python3.7/site-
packages/scipy/optimize/minpack.py:175: RuntimeWarning: The iteration is not
making good progress, as measured by the
  improvement from the last five Jacobian evaluations.
  warnings.warn(msg, RuntimeWarning)
```

```
/Users/simone/opt/anaconda3/lib/python3.7/site-
packages/scipy/optimize/minpack.py:175: RuntimeWarning: The iteration is not
making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```



Solutions de l'équation différentielle $y'(t) = \frac{2y}{15}(1 - y/1000)$, $t \in (0, 100)$, $\quad y(0) = 100$ *pour diverses valeurs de* N_h.

On voit que `fsolve` n'arrive pas à resoudre l'équation non-linéaire pour $N_h = 7$. Il faut chercher un meilleur `x0`.

Dans `backwardEuler`, replacez le `u[n]`

```
u[n+1] = fsolve( F, u[n]);
```

par la solution correspondante à la méthode d'Euler progressive, i.e.

```
u[n+1] = fsolve( F, u[n]+ h * fun( t[n], u[n] ));
```

Maintenant, pas seulement `fsolve` trouve une solution, mais en plus l'approximation de l'EDO ne présente pas d'oscillations.

[ ]:

[ ]:

1.4 Stabilité

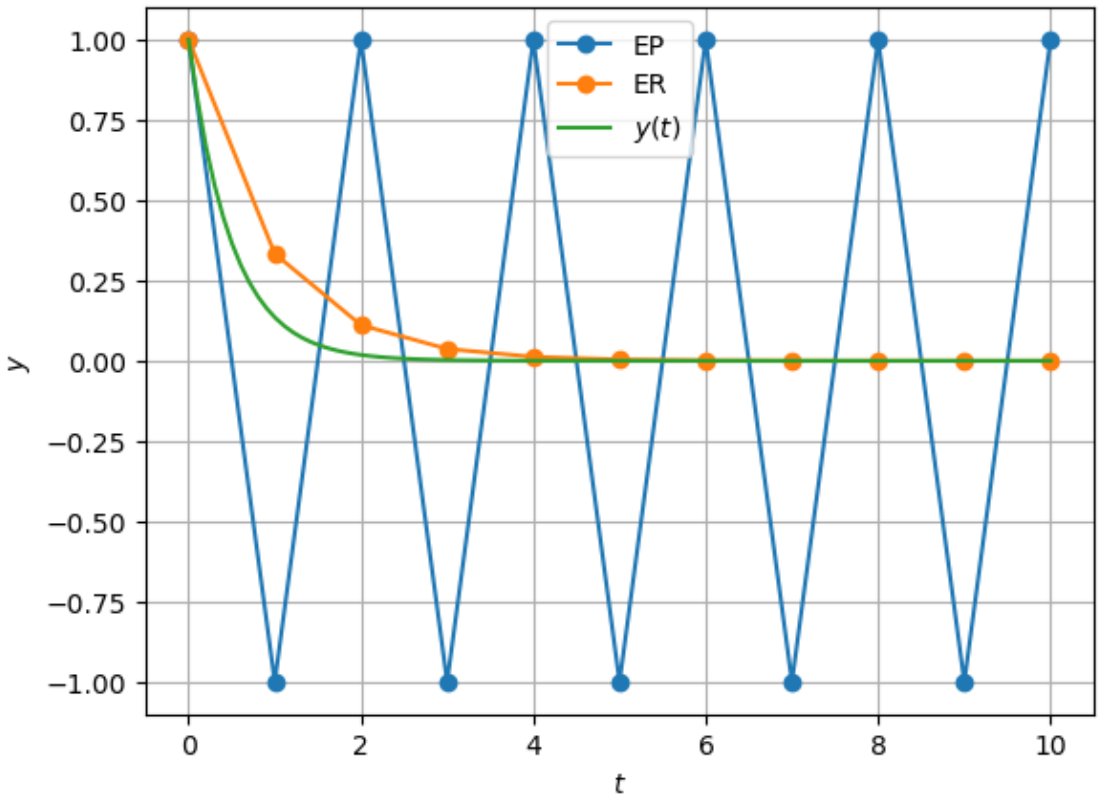
On considère le problème de Cauchy

$$\begin{cases} y'(t) = -2y(t), \quad t > 0 \\ y(0) = 1 \end{cases}$$

La solution exacte de ce problème est $y(t) = e^{-2t}$

Résolvez ce problème par les méthodes d'Euler Progressive et Rétrograde sur l'intervalle $[0, 10]$ avec un pas de temps $h = 0.9$ et 1.1.

[12]:
```
l = -2
f = lambda t,x :   l*x
y0 = 1; tspan=[0, 10]
h = 1.1; Nh = np.ceil((tspan[1] - tspan[0])/h).astype(int)
t_EP, y_EP = forwardEuler(f, tspan, y0, Nh)
t_ER, y_ER = backwardEuler(f, tspan, y0, Nh)

plt.plot(t_EP, y_EP,'o-')
plt.plot(t_ER, y_ER,'o-')

y = lambda t : np.exp(l*t)
t = np.linspace(tspan[0],tspan[1],100)
plt.plot(t, y(t),'-')
# labels, title, legend
plt.xlabel('$t$'); plt.ylabel('$y$')
plt.legend(['EP','ER','$y(t)$'])
plt.grid(True)
plt.show()
```



1.5 Convergence

On considère le problème de Cauchy

$$\begin{cases} y'(t) = -y(0.1 - \cos(t)), & t > 0 \\ y(0) = 1 \end{cases}$$

Resolvez ce problème par les méthodes d'Euler progressive et de Heun sur l'intervalle $[0, 12]$ avec un pas de temps $h = 0.4$.

La solution exacte est $y(t) = e^{-0.1t+\sin(t)}$. On remarque que la solution obtenue par la méthode de Heun est beaucoup plus précise que celle d'Euler progressive. Par ailleurs, on peut voir que si on réduit le pas de temps, la solution obtenue par la méthode d'Euler progressive s'approche de la solution exacte.

[13]:
```
f = lambda t,y : (np.cos(t) - 0.1)*y

tspan = [0,12]; y0 = 1;
h = 0.4; Nh = np.ceil((tspan[1] - tspan[0])/h).astype(int)

t_EP, y_EP = forwardEuler(f, tspan, y0, Nh)
t_H, y_H = Heun(f, tspan, y0, Nh)
```

```
plt.plot(t_EP, y_EP,'o-')
plt.plot(t_H, y_H,'o-')

y = lambda t : np.exp(-0.1*t+np.sin(t))
t = np.linspace(tspan[0],tspan[1],100)
plt.plot(t, y(t),'-')
# labels, title, legend
plt.xlabel('$t$'); plt.ylabel('$y$')
plt.legend(['EP','Heun','$y(t)$'])
plt.grid(True)
plt.show()
print("Figure: Approximation par le méthodes de Euler Retrograde et Heun.")
```

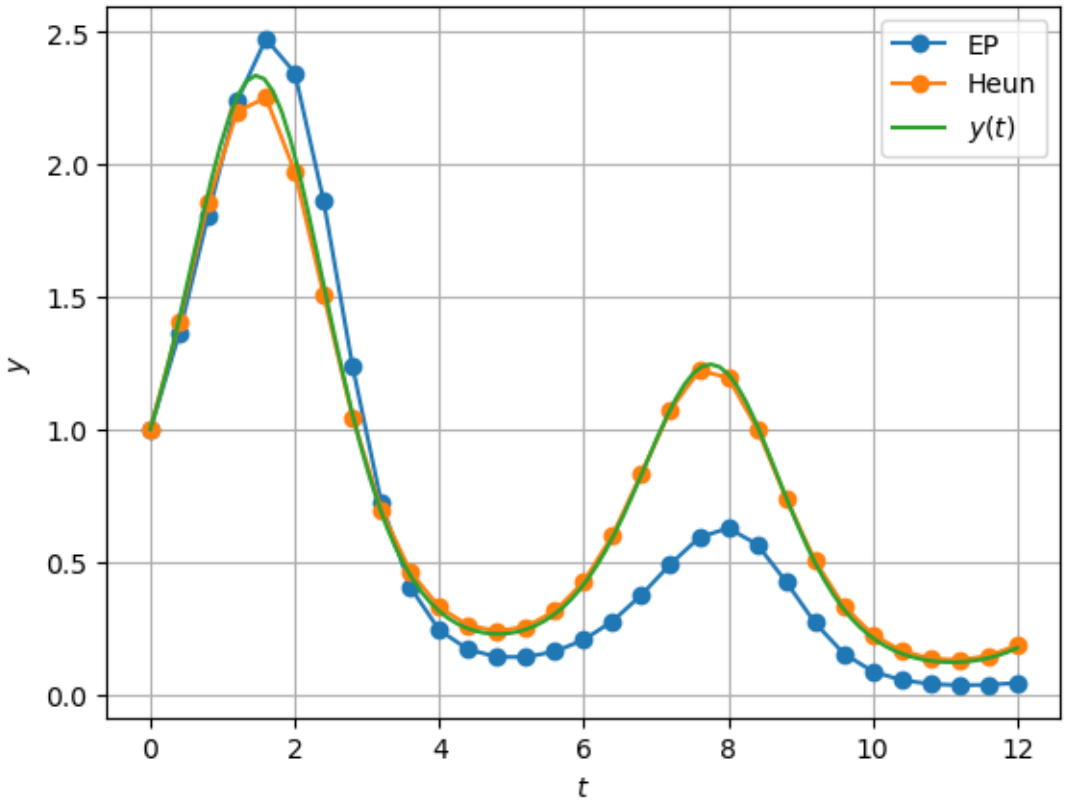


Figure: Approximation par le méthodes de Euler Retrograde et Heun.

[14]:
```
tspan = [0,12]; y0 = 1;
NhRange = [30, 50, 100, 500]
for Nh in NhRange :
    t, y = forwardEuler(f,tspan,y0,Nh)
    plt.plot(t, y,'-')
```

```
yt = lambda t : np.exp(-0.1*t+np.sin(t))
t = np.linspace(tspan[0],tspan[1],100)
plt.plot(t, yt(t),':')
# labels, title, legend
plt.xlabel('$t_n$'); plt.ylabel('$u_n$')
plt.legend(NhRange+['$y(t)$'])
plt.title('$u_n\\approx u(t_n)$')
plt.grid(True)
plt.show()
print("Figure: Solutions obtenues par la méthode d'Euler progressive pour␣
 ↪différents pas de temps.")
```

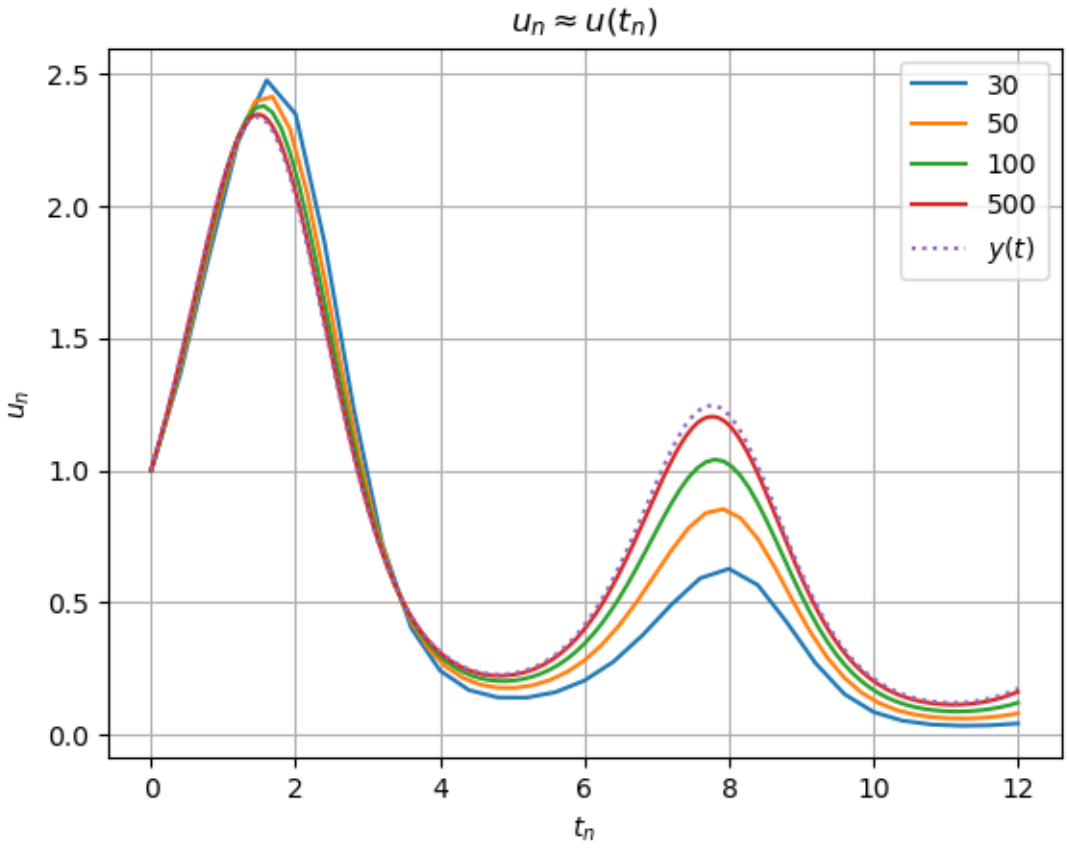


Figure: Solutions obtenues par la méthode d'Euler progressive pour différents pas de temps.

On veut, maintenant, estimer l'ordre de convergence de ces deux méthodes. Pour cela, on résout le problème avec différents pas de temps et on compare les résultats obtenus à l'instant $t = 6$ avec la solution exacte.

[15]:
```
tspan=[0,6]
NhRange = [30, 50, 100, 500]
errEP = []
errH = []
# Solution at end time
y6 = yt(tspan[1])
for Nh in NhRange :
    # Forward Euler
    t, y = backwardEuler(f,tspan,y0,Nh)
    # Error at the end of the simulation
    errEP.append( np.abs(y6 - y[-1] ) )

    # Heun
    [t, y] = Heun(f, tspan, y0, Nh);
    # Error at the end of the simulation
    errH.append( np.abs(y6 - y[-1] ) )

h = (tspan[1] - tspan[0])/np.array(NhRange)
plt.loglog(h,errEP,'o-b',h,errH,'o-r')
plt.loglog(h,h*(errEP[0]/h[0]),':',h,(h**2*(errH[0]/h[0]**2)),':')
plt.xlabel('$h$'); plt.ylabel('$|y(6)-u_{N_h}|$')
plt.legend(['EP','Heun','$h$','$h^2$'])
plt.title('Decay of the error')
plt.grid(True)
plt.show()

print("Figure: Erreurs en échelle logarithmique commises par les méthodes" +
      " d'Euler progressive et de Heun dans le calcul de y(6).")
```

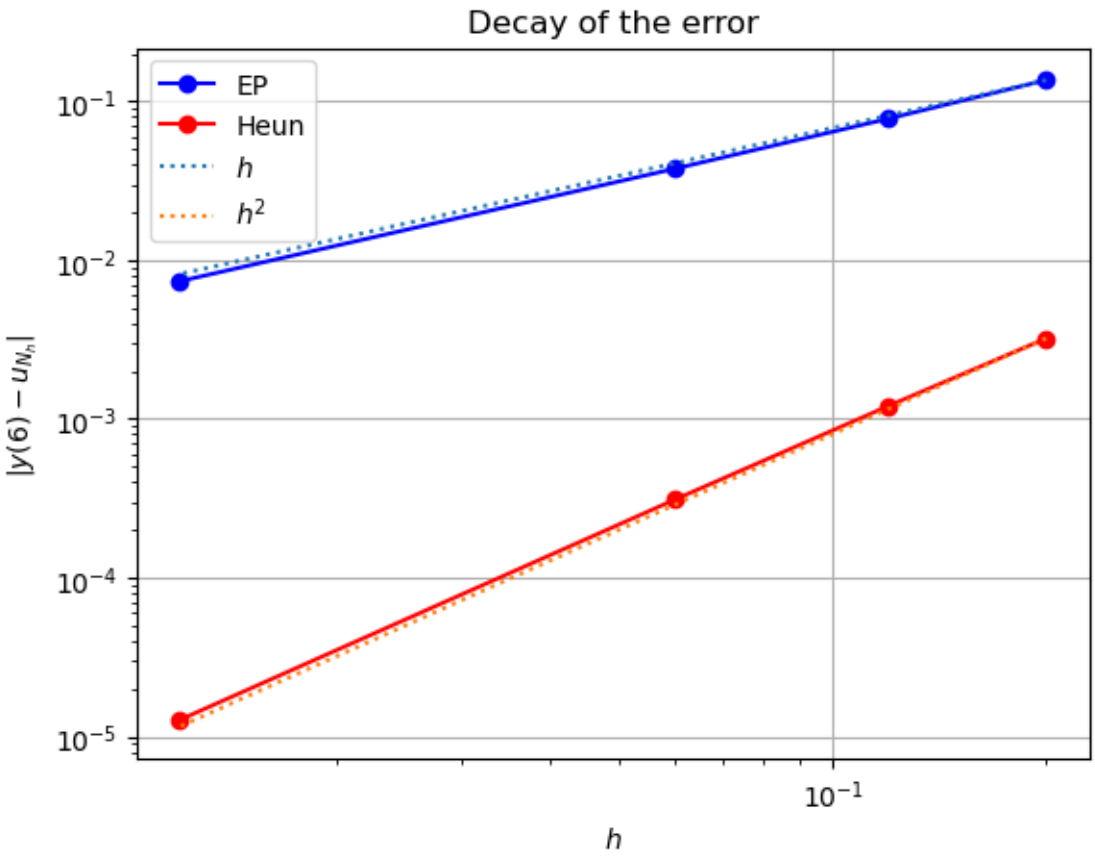


Figure: Erreurs en échelle logarithmique commises par les méthodes d'Euler progressive et de Heun dans le calcul de y(6).

La figure montre, en échelle logarithmique, les erreurs commises par les deux méthodes en fonction de h. On voit bien que la méthode d'Euler progressive converge à l'ordre 1 tandis que celle de Heun à l'ordre 2.

[ ]: