Série 11 (Corrigé)

Mardi prochain : Parcourez les chapitres 5 et 6.1 des notebooks Jupyter et résolvez les exercices qui y sont proposés.

Partiellement en classe vendredi

Exercice 1

On considère l'équation différentielle suivante

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

On veut résoudre cette équation avec les méthodes d'Euler progressive et Euler rétrograde, dans l'intervalle $[0, 4]$ avec $N_h = 20$ sous-intervalles. Ceci équivaut à un pas de temps $h = 0.2$ et donc à approcher la solution exacte $y(t_n)$ aux instants $t_n = nh$, $n = 0, 1, \ldots 20$ (donc $t_n = 0.2, 0.4, 0.6, \ldots$) par une solution numérique u_n.

On veut utiliser les programmes Python créé précédemment en utilisant les appels de fonction suivants :

— *Euler progressive*

```
f = lambda t,x = —t*x**2
Nh = 20; tspan = [0,4]; y0 = 2
t_EP, y_EP = forwardEuler(f, tspan, y0, Nh)
```

Les variables de sortie `t_EP` et `y_EP` contiennent respectivement la suite des instants t_n et les valeurs u_n correspondantes calculées par la méthode.

— *Euler rétrograde*

La fonction `beuler` utilise la même syntaxe :

```
t_ER, y_ER = backwardEuler(f, tspan, y0, Nh);
```

Comparer la solution exacte et celles obtenues par les méthodes d'Euler progressive et rétrograde. Pour cela afficher la solution exacte et celles obtenues par les deux méthodes sur un même graphe.

Sol. : *Cette équation différentielle peut être résolut analystiquement par séparation de variables.*

$$\int \frac{dy}{-y^2} = \int t dt \Rightarrow y^{-1} = \frac{1}{2}t^2 + c \Rightarrow y(t) = \frac{1}{c + \frac{1}{2}t^2}$$

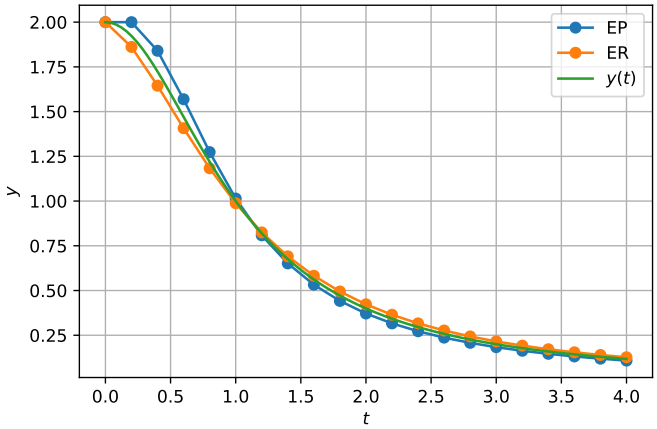
$$y(0) = 2 = \frac{1}{C} \Rightarrow y(t) = \frac{2}{1 + t^2}$$

```
  f = lambda t,x :   -t*x**2
y0 = 2; tspan=[0, 4]
Nh = 20
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 : 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(['EP', 'ER', '$y(t)$'])
plt.grid(True)
plt.show()

plt.savefig("EX007_fig.pdf", dpi=150)
```



Exercice 2

On considère le problème de Cauchy suivant :

$$\begin{cases} y'(t) = -e^t y^2(t) & t \in [1,3] \\ y(1) = 2 \end{cases}$$

a) Ecrivez la méthode d'Euler rétrograde pour approcher la solution $y(t)$.

b) Réécrivez la méthode d'Euler rétrograde sous la forme

$$u_{n+1} = \varphi(u_{n+1}; n, h, u_n),$$

où n, h, u_n sont considérés de paramètres, et écrivez le schéma de Newton pour résoudre cette équation non linéaire.

Sol. :

a) Il vaut mieux d'abord identifier les différentes données du problème :

$$f(t,x) = -e^t x^2, \qquad t_0 = 1, T = 3, I = [t_0, T], \qquad y_0 = 2$$

Soit $h > 0$ le pas de temps (fixé) et $N_h = \frac{T-t_0}{h} = \frac{2}{h}$ (On suppose que N_h est entier).

Soient $t_n = t_0 + nh = 1 + nh$ pour $n = 0, 1, ..., N_h$, on cherche une approximation u_n de $y(t_n)$. La méthode d'Euler progressif s'écrit

$$\begin{cases} u_{n+1} - u_n = -he^{t_{n+1}} u_{n+1}^2 & n = 0, 1, 2, ... \\ u_0 = 2 \end{cases}$$

Donc on a une équation quadratique à résoudre à chaque pas de temps :

$$\begin{cases} u_{n+1} = u_n - he^{1+nh+h} u_{n+1}^2 & n = 0, 1, 2, ... \\ u_0 = 2 \end{cases}$$

— *On peut écrire la méthode d'Euler rétrograde de la façon suivante :*

$$u_{n+1} = \varphi(u_{n+1}; n, h, u_n) = -he^{1+nh+h} u_{n+1}^2 + u_n \tag{1}$$

On a donc $\varphi(x) = -he^{1+nh+h}x^2$. Alors on peut réécrire (1) comme suit

$$x = \varphi(x; n, h, u_n) = -(he^{t_{n+1}})x^2 + u_n.$$

Trouver x (c-à-d l'approximation u_{n+1}) est équivalent à calculer le zéro de la fonction

$$F(x) = x - \varphi(x) = x + (he^{t_{n+1}})x^2 - u_n,$$

où on a omis les paramètres n, h, u_n. On écrit la méthode de Newton pour la fonction $F(x)$:

$$x^{(k+1)} = x^{(k)} - \frac{F(x^{(k)})}{F'(x^{(k)})} \text{ et } F'(x) = 1 - \frac{d\varphi}{dx} = 1 + 2he^{1+nh+h}x$$

Donc on trouve

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} + he^{1+nh+h}(x^{(k)})^2 - u_n}{1 + 2he^{1+nh+h}x^{(k)}} =$$
$$= \frac{x^{(k)} + 2he^{1+nh+h}(x^{(k)})^2 - x^{(k)} - he^{1+nh+h}(x^{(k)})^2 - u_n}{1 + 2he^{1+nh+h}x^{(k)}} = \frac{he^{1+nh+h}(x^{(k)})^2 - u_n}{1 + 2he^{1+nh+h}x^{(k)}}$$

Exercice 3

On considère la méthode de Crank–Nicolson

$$\begin{cases} \dfrac{u_{n+1} - u_n}{h} = \dfrac{1}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right], \quad n = 0, 1, \dots \\ u_0 = y_0, \end{cases} \tag{2}$$

h étant le pas de temps, pour approcher la solution du problème de Cauchy

$$\begin{cases} y'(t) = f(t, y(t)), \quad t \in (0, T), \\ y(0) = y_0. \end{cases}$$

Soit $\lambda < 0$ un nombre réel *négatif* donné. On considère le problème modèle

$$\begin{cases} y'(t) = \lambda y(t), \quad t > 0, \\ y(0) = 1, \end{cases} \tag{3}$$

dont la solution exacte est $y(t) = e^{\lambda t}$.

1. Écrivez le schéma de Crank–Nicolson pour l'approximation numérique du problème de Cauchy (3).
2. Résolvez cette équation et donnez une expression explicite de u_n en fonction de h, λ et n.
3. Trouvez en fonction de λ les valeurs de h pour lesquelles la méthode de Crank–Nicolson appliqué à ce problème est telle que $u_n \to 0$. On dira que la méthode de Crank–Nicolson est absolument stable pour ces valeurs de h et λ.

Sol. :

1. *Le schéma de Crank–Nicolson pour le problème modèle considéré s'écrit :*

$$\begin{cases} u_{n+1} = u_n + \dfrac{h}{2}(\lambda u_n + \lambda u_{n+1}), \\ u_0 = 1. \end{cases} \tag{4}$$

En particulier, on peut mettre (4) *sous la forme suivante :*

$$\begin{cases} u_{n+1} = \dfrac{1 + \dfrac{h}{2}\lambda}{1 - \dfrac{h}{2}\lambda} u_n, \\ u_0 = 1. \end{cases} \tag{5}$$

Chaque schéma donne une équation de récurrence qui définit la suite u_n, $n \in \mathbb{N}$, de la façon suivante : u_0 est égal à la valeur initiale $y(t_0) = 1$ et les valeurs u_n pour tout $n > 0$ se calculent grâce à l'expression (5) appliquée de manière répétée (on trouve u_1 à partir de u_0, u_2 à partir de u_1, etc...).

2. *Pour le schéma de Crank–Nicolson, on a*

$$u_n = \left(\frac{1 + \dfrac{h}{2}\lambda}{1 - \dfrac{h}{2}\lambda} \right)^n . \tag{6}$$

3. *Une suite définie par*

$$u_n = \gamma^n$$

converge vers zéro pour $n \to \infty$ *si et seulement si*

$$|\gamma| < 1.$$

D'après (6)*, le schéma de Crank–Nicolson correspond à* $\gamma = \left(1 + \frac{h}{2}\lambda\right) \Big/ \left(1 - \frac{h}{2}\lambda\right)$.

Vu que $h\lambda < 0$*, on a* $\left|1 + \frac{h}{2}\lambda\right| < \left|1 - \frac{h}{2}\lambda\right|$ *et donc*

$$\left|\frac{1 + \frac{h}{2}\lambda}{1 - \frac{h}{2}\lambda}\right| < 1.$$

En particulier, $|\gamma| < 1$*, donc* $u_n \to 0$ *lorsque* $n \to \infty$ *sans aucune condition sur le pas de temps* h*. On dit que le schéma de Crank–Nicolson est* inconditionnellement stable.

Python

Exercice 4

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

$$y'(t) = f(t, y(t)),\ t \in (T_0, T_f), \quad y(0) = y_0, \tag{7}$$

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`, utiliser 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')
```

Approcher 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.

Sol. :

```
def forwardEuler( fun, interval, y0, N ) :
  #  FORWARDEULER Solve differential equations using the forward Euler
     method.
  #  [T, U] = FORWARDEULER( FUN, INTERVAL, YO, N ), with INTERVAL = [TO, TF],
  #  integrates the system of differential equations y'=f(t, y) from time TO
  #  to time TF, with initial condition YO, 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
```

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

```
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.*

Exercice 5

Écrivez une fonction Python `backwardEuler` qui approche la solution du problème

$$y'(t) = f(t, y(t)),\ t \in (T_0, T_f), \quad y(0) = y_0, \tag{8}$$

en utilisant la méthode d'Euler rétrograde. 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
```

Remarque : Une fois écrite la fonction `backwardEuler`. 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).

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}')
```

Approcher 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.

Sol. :

```
def backwardEuler( fun, interval, y0, N ) :
  #  BACKWARDEULER Solve differential equations using the backward Euler
     method.
  #  [T, U] = BACKWARDEULER( FUN, INTERVAL, YO, N ), with INTERVAL = [TO,
     TF],
  #  integrates the system of differential equations y'=f(t, y) from time TO
  #  to time TF, with initial condition YO, 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 :

```
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 :

```
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()
```

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]` *dans* `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.*

Exercice 6

On considère une population de y individus dans un environnement où au plus $B = 1000$ individus peuvent coexister. On suppose qu'initialement le nombre d'individus est $y_0 = 100$ et que le facteur de croissance est égal à une constante C. Le modèle de l'évolution de la population considérée sur 100 années est le suivant :

$$y'(t) = Cy(t)\left(1 - \frac{y(t)}{B}\right),\ t \in (0, 100), \quad y(0) = y_0, \tag{9}$$

où t est mesuré *en années*, et $C = 2/15$ an^{-1}.

Soit u_n l'approximation de $y(t_n)$, où $t_n = nh$, $n = 0, 1, 2, \ldots, N_h$, $h = 100/N_h$ étant le pas de temps, N_h étant le nombre de pas temporels. Plus N_h est grand, plus le pas de temps est petit et plus l'approximation sera précise.

1. Ecrire les schémas d'Euler progressif (explicite) et rétrograde (implicite) pour calculer une approximation u_n de $y(t_n)$ (donner les équations de récurrence définissant la suite u_n dans les deux cas).
2. On prend $h = 1/12$ an ; donner la valeur u_1 qui approche le nombre d'individus $y(t_1)$ au temps t_1 (donc après *1 mois*), obtenue *i)* par la méthode d'Euler progressive, puis *ii)* par la méthode d'Euler rétrograde. Suggestion : resoudre à la main l'équation pour trouver la nouvelle valeur u_{i+1}.
3. En utilisant la méthode d'Euler progressif (copier et modifier la fonction forwardEuler.m créé précédemment), calculer les valeurs approchées u_n, $n = 0, 1, \ldots, N_h$ en Python, pour $N_h = 20$.
4. Tracer un graphe de la solution numérique trouvée en fonction du temps.
5. Faire à nouveau le calcul précédent, mais en prenant $N_h = 1000$; tracer la nouvelle solution numérique.
6. D'après les résultats trouvés, combien d'années sont nécessaires pour que la population atteigne le nombre de 900 individus ? *Suggestion : utiliser la commande* `find`.

Sol. :

1. *En général, un schéma numérique pour le calcul des approximations u_n des valeurs $y(t_n)$, s'écrit sous la forme d'une équation* de récurrence *définissant u_{n+1} en fonction de la valeur u_n trouvée au pas précedent. Dans notre cas, le schéma d'Euler progressif est donné par*

$$\begin{cases} u_{n+1} = u_n + hCu_n \left(1 - \dfrac{u_n}{B}\right) , \\ u_0 = 100, \end{cases}$$

tandis que celui d'Euler rétrograde s'écrit

$$\begin{cases} u_{n+1} = u_n + hCu_{n+1} \left(1 - \dfrac{u_{n+1}}{B}\right) , \\ u_0 = 100. \end{cases}$$

2. *Avec les valeurs données, aprés un mois la valeur approchée du nombre d'individus que l'on calcule par la méthode d'Euler progressive est*

$$u_1 = 100 + \frac{1}{12}\frac{2}{15}100\left(1 - \frac{100}{1000}\right) = 101.$$

Pour calculer l'approximation selon le schéma d'Euler rétrograde, il faut résoudre une équation avec u_1 comme inconnue :

$$hC\frac{1}{B}u_1^2 + (1 - hC)u_1 - u_0 = 0.$$

Dans ce cas, la solution n'est pas unique. On a deux racines :

$$u_1 = \frac{B}{2hC}\left[-(1 - hC) \pm \sqrt{(1 - hC)^2 + 4hC\frac{u_0}{B}}\right].$$

Néanmoins, seule la solution positive est acceptable (on rappelle que l'on cherche à calculer un nombre d'individus). Celle-ci est donc

$$u_1 = 1000 \cdot 45 \left[-\left(1 - \frac{1}{90}\right) + \sqrt{\left(1 - \frac{1}{90}\right)^2 + \frac{2}{450}} \right] = 101.008958.$$

3. *On sait que le schéma d'Euler progressif pour le problème considéré est*

$$\begin{cases} u_{n+1} = u_n + hCu_n \left(1 - \dfrac{u_n}{B}\right) , \\ u_0 = 100. \end{cases}$$

En Python, ceci s'écrit

```
C = 2/15; B=1000;          # valeurs des paramtres C et B
y0 = 100
Nh = 20;                   # valeur de Nh
t0 = 0; T = 100.; tsp = [t0, T]
h = (T-t0)/Nh;               # pas de temps

f = lambda t,x : ( C * x * (1-x/B) )
t, u = forwardEuler(f, tsp, y0, Nh);
```

où on a suivi la suggestion de l'énoncé.

4. *Pour tracer le graphe de la solution numérique en fonction du temps, il faut se rappeler que u_n est la valeur approchée au temps $t_n = nh$. Donc on peut taper*

```
plt.plot(t,u, 'o-')

# labels, title, legend
plt.xlabel('$t$'); plt.ylabel('$u_n$')
plt.legend('EP')
plt.grid(True)
plt.show()
```

pour obtenir le graphe cherché.

5. *On utilise les mêmes commandes qu'au point a), mais on change la valeur de* `Nh` *:*

```
C = 2/15; B=1000;          # valeurs des paramtres C et B
y0 = 100
Nh = 1000;                   # valeur de Nh
t0 = 0; T = 100.; tsp = [t0, T]
h = (T-t0)/Nh;               # pas de temps

f = lambda t,x : ( C * x * (1-x/B) )
t1, u1 = forwardEuler(f, tsp, y0, Nh);
```

Ensuite on trace le nouveau graphe avec

```
plt.plot(t,u, 'o-')
plt.plot(t1,u1, '-')

# labels, title, legend
plt.xlabel('$t$'); plt.ylabel('$u_n$')
plt.legend(['Nh=20', 'Nh=1000'])
plt.grid(True)
plt.title("Solutions de l'equation differentielle pour diverses valeurs
    de $N_h$.")

plt.plot([t0,T], [900,900], 'g-',linewidth=0.2)

plt.savefig("EX020_fig.pdf", dpi=150)
plt.show()
```

Le résultat est affiché en fig. 1. On s'attend à ce que la solution correspondant à

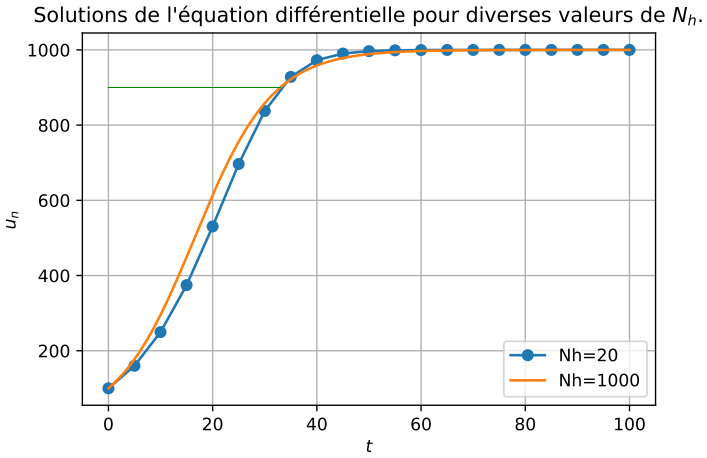


FIGURE 1 – Solutions de l'équation différentielle pour diverses valeurs de N_h.

$N_h = 1000$ (en rouge) soit la plus précise, donc la plus proche de l'évolution exacte de la population.

6. *On peut répondre soit en analysant le graphe de la solution numérique la plus précise ($N_h = 1000$), en tracent la droite correspondent à 'u=900' (avec la commande* `plt.plot([t0,T], [900,900], 'g-',linewidth=0.2)`*, soit en utilisant la commande* `np.where` *de numpy. On donne ici un exemple de comment utiliser cette fonction (*Remarque* : si* `u` *est un vecteur, alors* `(u>=900)` *est aussi un vecteur, dont la composante *i*-ème est égale à 1 si $u(i) \geq 900$, zéro autrement. La commande 'where' retourne alors le vecteur des indices non-nuls de* `(u>=900)` *; finalement, le premier élément du vecteur résultant nous donne le plus petit de ces indices) :*

```
index = np.where(u1 >= 900)

i = index[0][0];          # le plus petit des indices i
```

```
                                # tels que u(i)> =900
T = h*i
print(T)
```

Donc la population atteint le nombre de 900 individus après $T = 33$ *ans.*