Lycée Pierre-Gilles de Gennes 2024-2025

TP4 Python

Équations différentielles

L'objet de ce TP est de mettre en œuvre la méthode d'Euler sur différents exemples d'équations différentielles.
Ce TP prendra 2 séances.

1 Modèle de Gompertz

On considère le problème de Cauchy suivant sur $\mathbb{R}_+$:

$$(G) \qquad y' = ay \ln\left(\frac{\kappa}{y}\right) \quad \text{et} \quad y(0) = y_0$$

où a, κ sont des réels strictement positifs et $y_0 > 0$.
On rappelle [1] que l'unique solution y de (G) est donnée par :

$$\forall t \geq 0, \quad y(t) = \kappa e^{\ln\left(\frac{y_0}{\kappa}\right)e^{-at}}.$$

On se propose de déterminer numériquement une valeur approchée de y sur un segment $[0, T]$ $(T > 0)$ par deux méthodes.

1.1 Méthode d'Euler

Soit $n \in \mathbb{N}^*$. On pose pour tout $k \in [\![0, n-1]\!]$:

$$y_{k+1} = y_k + \frac{T}{n} a y_k \ln\left(\frac{\kappa}{y_k}\right).$$

Travail demandé

1. Définir des variables `a` et `kappa` initialisées avec les valeurs 0.036 et 760 respectivement.

```
a = 0.036
kappa = 760
```

2. Écrire une fonction `Euler` prenant en entrées y_0, T et n et renvoyant la suite $(y_k)_{k \in [\![0,n]\!]}$ obtenue par la méthode d'Euler.

```
def Euler(y0,T,n):
    '''
    Entrées : [0,T] les extrémités de l'intervalle
              y0 une condition initiale
              n le nombre de subdivision
```

1. Voir l'exercice 7 du TD4 et son corrigé.

```
    Sortie : une approximation de la solution de y'=F(y)
    sur [a,b] telle que y(a)=y0 avec Euler
    '''
    h=T/n  # pas
    Y=[y0]
    for _ in range(1,n+1):
        yold=Y[-1]
        ynew = yold + h*a*yold*np.log(kappa/yold)
        Y.append(ynew)# liste des approximation de y(a+k(b-a)/n)
    return Y
```

3. Tracés des suites de points $\left(\left(k\frac{T}{n}, y_k\right)\right)_{k\in[\![0,n]\!]}$ pour $y_0 = 16$, $T = 200$, $n \in \{1, 5, 10, 100\}$ et de la solution exacte.

```
def solution(x):
    '''solution exacte de l'équation de Gompertz'''
    c = np.log(y0/kappa)
    return kappa*np.exp(c*np.exp(-a*x))

y0 = 16
T= 200
N = [1,5,10,100]
for n in N:
    temps = [k*T/n for k in range(n+1)]
    Y = Euler(y0,T,n)
    plt.plot(temps,Y, label = 'Euler avec n='+str(n))

abs = np.linspace(0,T,1000)
plt.plot(abs, solution(abs),':',label = 'solution exacte')
plt.legend()
```

On obtient la figure 1 :

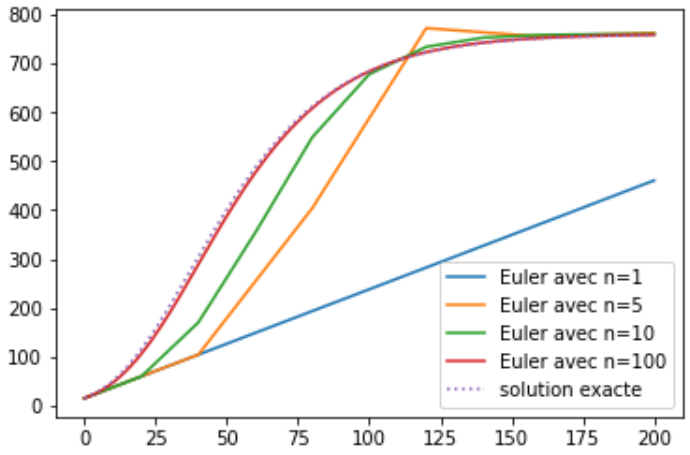

FIGURE 1 – Approximation de la solution par la méthode d'Euler

4.

5. On obtient la figure 2 :

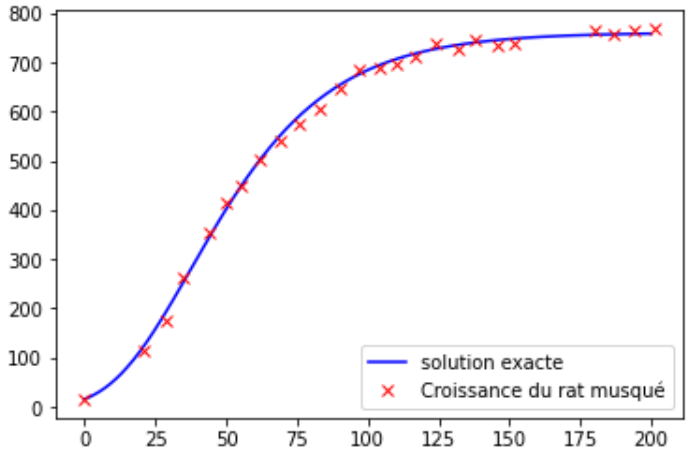

FIGURE 2 – Croissance des rats musqués

On constate que la l'évolution de la masse au cours du temps est bien modélisée par la solution de l'équation de Gompertz.

1.2 Méthode de Heun

La méthode d'Euler utilise la valeur de la dérivée à l'extrémité inférieure de l'intervalle $\left[k\frac{T}{n},(k+1)\frac{T}{n}\right]$ pour estimer la pente de la courbe sur l'intervalle.
La méthode de Heun tente d'améliorer cette estimation de pente en calculant la pente de la tangente à l'autre extrémité de l'intervalle, que l'on estime à l'aide de la méthode d'Euler, et en faisant la moyenne des pentes obtenues.
Soit $n\in\mathbb{N}^*$. On pose pour tout $k\in[\![0,n-1]\!]$:

$$z_{k+1}=y_k+\frac{T}{n}ay_k\ln\left(\frac{\kappa}{y_k}\right)\quad\text{puis}\quad y_{k+1}=y_k+\frac{T}{2n}\left((ay_k\ln\left(\frac{\kappa}{y_k}\right)+az_{k+1}\ln\left(\frac{\kappa}{z_{k+1}}\right)\right).$$

Travail demandé

1. Écrire une fonction `Heun` prenant en entrées y_0, T et n et renvoyant la suite $(y_k)_{k\in[\![0,n]\!]}$ obtenue par la méthode de Heun.

```
def Heun(y0,T,n):
    '''
    Entrées : [0,T] les extremités de l'intervalle
              y0 une condition initiale
              n le nombre de subdivision
    Sortie : une approximation de la solution de y'=F(y)
    sur [a,b] telle que y(a)=y0 avec Heun
    '''
    h=T/n   # pas
    Y=[y0]
```

```
11      for _ in range(1,n+1):
12          y = Y[-1]
13          z = y+h*a*y*np.log(kappa/y)
14          y = y+h/2*(a*y*np.log(kappa/y)+a*z*np.log(kappa/z))
15          Y.append(y)# liste des approximation de y(a+k(b-a)/n)
16      return Y
```

2. Tracer la suite de points $\left(\left(k\frac{T}{n}, y_k\right)\right)_{k\in[\![0,n]\!]}$ pour $y_0 = 16$, $T = 200$ et $n = 1$, 5, 10 et 100 :

```
1  N = [1,5,10,100]
2  for n in N:
3      temps = [k*T/n for k in range(n+1)]
4      Y = Heun(y0,T,n)
5      plt.plot(temps,Y, label = 'Heun avec n='+str(n))
6
7  abs = np.linspace(0,T,1000)
8  plt.plot(abs, solution(abs),':',label = 'solution exacte')
9  plt.legend()
```

On obtient la figure 3 :

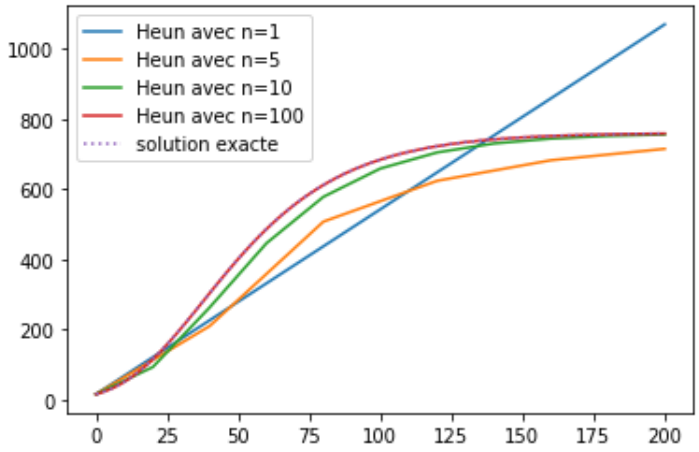

FIGURE 3 – Approximation de la solution par la méthode de Heun

3. On constate que la méthode de Heun semble donné une bonne approximation de la solution exacte.

1.3 Comparaison des méthodes

Que ce soit pour la méthode d'Euler ou la méthode de Heun, l'erreur commise par la méthode avec n subdivisions sur $[0, T]$ est définie par :

$$e_n(T) = \max\left\{|y_k - y\left(k\frac{T}{n}\right)| \ ; \ k \in [\![0, n]\!]\right\}.$$

On dit que la méthode converge lorsque, à T fixé, l'erreur tend vers 0 quand n tend vers $+\infty$.

Travail demandé

1. Écrire une fonction `Erreur` prenant en entrées `T`, `y0` et `n` et renvoyant la valeur de $e_n(T)$ pour les deux méthodes.

```
def Erreur(y0,T,n):
    Y1 = Euler(y0,T,n) # approximation par Euler
    Y2 = Heun(y0,T,n)  # approximation par Heun
    E1 = [np.abs(solution(k*T/n)-Y1[k]) for k in range(n+1)]
    E2 = [np.abs(solution(k*T/n)-Y2[k]) for k in range(n+1)]
    return max(E1),max(E2)
```

2. **Convergence des méthodes.**
   (a) Pour $T = 200$ et $y_0 = 16$ tracer le graphique de l'erreur en fonction du nombre de subdivisions $n \in [\![1, 100]\!]$ pour les deux méthodes.

```
Nb = range(1,100)   # nombre de subdivision
Erreur1= [Erreur(y0,T,k)[0] for k in Nb]
Erreur2= [Erreur(y0,T,k)[1] for k in Nb]
plt.plot(Nb,Erreur1)   # erreur de la méthode d'Euler
plt.plot(Nb,Erreur2) # erreur de la méthode de Heun
```

   On obtient la figure 4 :

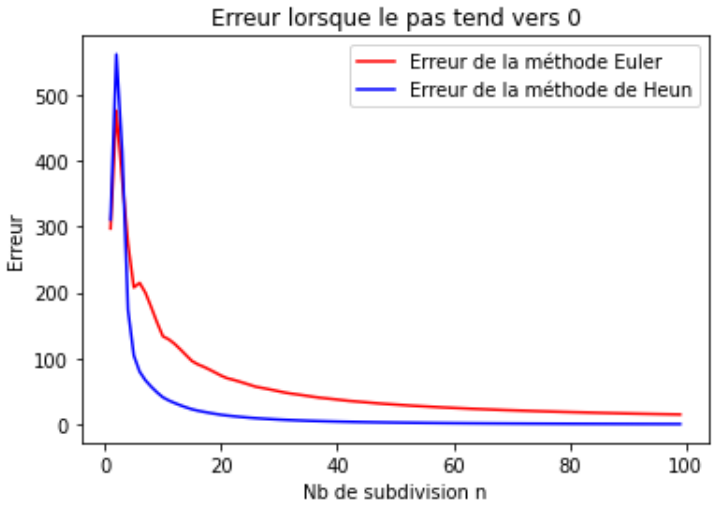

FIGURE 4 – Évolution de $e_n(T)$ en fonction de n

   (b) On constate que l'erreur semble tendre vers 0 quand n tend vers $+\infty$: cela signifie que lorsque le pas tend vers 0, les solutions approchées s'approchent de la solution exacte.

   De plus, l'erreur de la méthode de Heun tend plus rapidement vers 0 : elle approche donc plus rapide de la solution exacte.

   La méthode de Heun semble donc plus précise.

3. **Erreur à pas constant.**
   (a) Pour $n = 100$ et $y_0 = 16$ tracer le graphique de l'erreur en fonction de la taille $T \in [1, 50]$ de l'intervalle pour les deux méthodes.

```
intervalle = range(1,50)
Erreur1= [Erreur(y0,k,100)[0] for k in intervalle]
Erreur2= [Erreur(y0,k,100)[1] for k in intervalle]
plt.plot(intervalle,Erreur1)  # erreur de la méthode d'Euler
plt.plot(intervalle,Erreur2) # erreur de la méthode de Heun
```

On obtient la figure 5 :

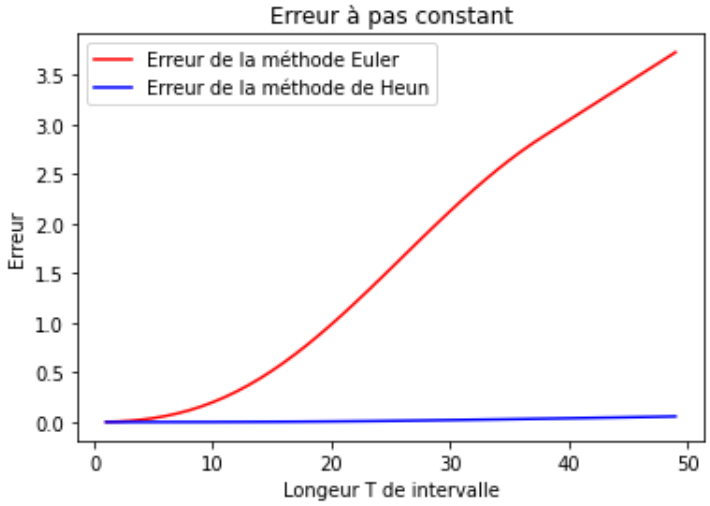

FIGURE 5 – Évolution de $e_n(T)$ en fonction de n

(b) Là encore la méthode de Heun est plus efficace : à pas constant (n fixé) l'erreur explose beaucoup plus rapidement pour la méthode d'Euler que pour la méthode de Heun lorsque l'intervalle de définition grandit.

2 Modèle proies-prédateurs de Lotka-Volterra

Soit a, b, c, d des réels strictement positifs.
On considère le système différentiel [2] ci-dessous où x et y désignent respectivement l'effectif d'une population de proies et l'effectif d'une population de prédateurs en fonction de $t \in \mathbb{R}_+$.
Enfin, $(x_0, y_0) \in \mathbb{R}^2$ désigne les effectifs relevés à un instant $t_0 \in \mathbb{R}_+$.
On admet, pour tout $t_0 \in \mathbb{R}_+$ et tout $(x_0, y_0) \in \mathbb{R}^2$ l'**existence d'un unique couple** (x, y) de fonctions dérivables sur $\mathbb{R}_+$ vérifiant :

$$(S) \qquad \forall t \in \mathbb{R}_+, \quad \begin{cases} x'(t) &= (a - by(t))x(t) \\ y'(t) &= (cx(t) - d)y(t) \end{cases} \quad \text{et} \quad \begin{cases} x(t_0) = x_0 \\ y(t_0) = y_0 \end{cases}$$

On note $\varphi : \mathbb{R}_+^* \times \mathbb{R}_+^* \to \mathbb{R}$ définie par :

$$\forall (x, y) \in \left(\mathbb{R}_+^*\right)^2, \quad \varphi(x, y) = cx - d\ln(x) + by - a\ln(y).$$

On rappelle [3] que si x_0 et y_0 sont strictement positifs alors l'unique couple solution (x, y) vérifie la propriété (P_1) suivante :

$$(P_1) \qquad \forall t \geq 0, \quad \varphi(x(t), y(t)) = \varphi(x_0, y_0).$$

2. Voir le DS2.
3. Voir le DS2.

On admet également que les solutions sont périodiques et on note ρ la période. Ainsi, (x, y) vérifie la propriété (P_2) suivante :

$$(P_2) \qquad \forall t \geq 0, \quad (x(t+\rho), y(t+\rho)) = (x(t), y(t)).$$

On se propose de déterminer numériquement une valeur approchée d'un couple solution (x, y) sur un segment $[0, T]$ $(T > 0)$ par les méthodes d'Euler et de Heun.

Dans la suite, on prendra $a = b = c = 1$ et $d = 2$.

2.1 Méthode d'Euler

Pour tout $n \in \mathbb{N}^*$, à partir de la condition initiale $(x_0, y_0) \in (\mathbb{R}_+^*)^2$, on construit deux suites $(x_k)_{k\in[\![0,n]\!]}$ et $(y_k)_{k\in[\![0,n]\!]}$ définies par :

$$\forall k \in [\![0, n-1]\!], \quad \begin{cases} x_{k+1} &= x_k + \dfrac{T}{n} x_k(a - by_k) \\ y_{k+1} &= y_k + \dfrac{T}{n} y_k(cx_k - d) \end{cases}$$

Travail demandé

1. Écrire une fonction `Euler_syst` prenant en entrées x_0, y_0, T et n et renvoyant les suites $(x_k)_{k\in[\![0,n]\!]}$ et $(y_k)_{k\in[\![0,n]\!]}$ obtenues par la méthode d'Euler.

```
def F(x,y):
    return [(1-y)*x,(x-2)*y]

def Euler_syst(x0,y0,T,n):
    '''
    Entrées : [0,T] les extremités de l'intervalle de déf
              (x0,y0) une condition initiale
              n le nombre de subdivision
    Sortie : une approximation de la solution du système
    sur [0,T] avec la condition initiale respectée
    '''
    h=T/n  # pas
    X = [x0]
    Y = [y0]
    for _ in range(1,n+1):
        x = X[-1]
        y = Y[-1]
        X.append(x+h*F(x,y)[0])# liste des approximation de x(a+k(b-a)/n)
        Y.append(y+h*F(x,y)[1])# liste des approximation de y(a+k(b-a)/n)
    return [X,Y]
```

2. Tracer les suites de points $\left(\left(k\dfrac{T}{n}, x_k\right)\right)_{k\in[\![0,n]\!]}$, $\left(\left(k\dfrac{T}{n}, y_k\right)\right)_{k\in[\![0,n]\!]}$ pour $x_0 = 1$, $y_0 = 0.5$, $T = 20$ et $n = 1000$.

```
x0=1
y0 = 0.5
T= 20
n = 1000
[X,Y] = Euler_syst(x0,y0,T,n)
temps = [k*T/n for k in range(n+1)]
plt.plot(temps,X, ':', label='valeur approchée de x (proies)')
plt.plot(temps,Y,  label='valeur approchée de y (prédateurs)')
plt.legend()
plt.show()
```

On obtient la figure 6 :

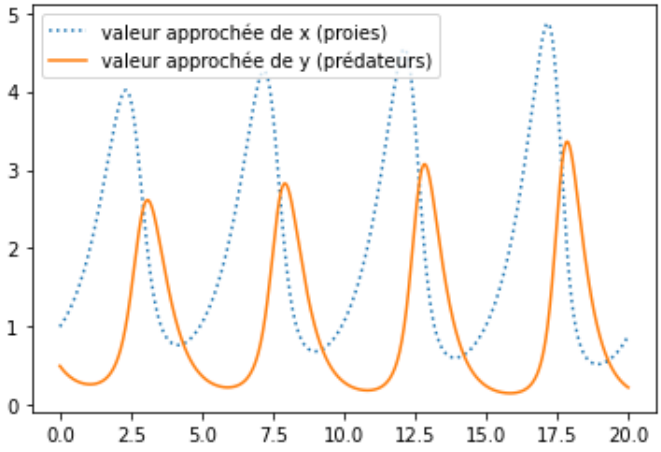

FIGURE 6 – Approximation des solutions par Euler

3. (a) Pour le tracer (voir figure 7) :

```
def phi(x,y):
    '''Energie '''
    return x-2*np.log(x)+y-np.log(y)
plt.plot(temps, phi(X,Y))
plt.show()
```

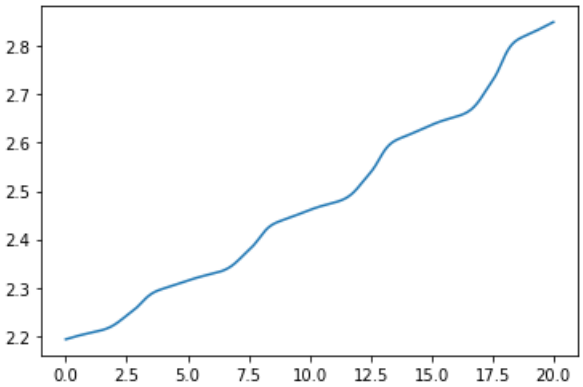

FIGURE 7 – Approximation des solutions par Euler

(b) D'après la propriété (P_1) attendue en théorie, la fonction $t \mapsto \varphi(x(t), y(t))$ est constante. On s'attend donc à ce que la courbe ci-dessus soit horizontale.

Or ce n'est pas le cas : la méthode d'Euler ne rend pas compte de la propriété (P_1). Cela est du aux erreurs d'approximation qui s'accumulent.

4. (a) Tracer la représentation graphique de (y_k) en fonction de (x_k) :

```
1 [X,Y] = Euler_syst(x0,y0,T,n)
2 plt.plot(X,Y)
3 plt.show()
```

On obtient le graphique suivant :

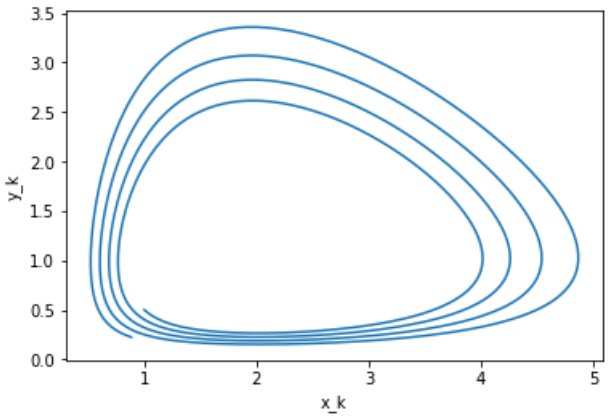

FIGURE 8 – Représentation graphique de (y_k) en fonction de (x_k)

(b) D'après la propriété (P_2) attendue en théorie, $t \mapsto (x(t), y(t))$ est périodique. On s'attend donc à ce que la courbe « se referme » à l'issue d'une période. Or se n'est pas le cas ici : une fois encore la méthode d'Euler ne rend pas compte des propriétés des solutions.

2.2 Méthode de Heun

Pour tout $n \in \mathbb{N}^*$, à partir de la condition initiale $(x_0, y_0) \in (\mathbb{R}_+^*)^2$, on construit deux suites $(x_k)_{k\in[\![0,n]\!]}$ et $(y_k)_{k\in[\![0,n]\!]}$ définies pour tout $k \in [\![0, n-1]\!]$ par :

$$\begin{cases} x_{k+1}^* &= x_k + \dfrac{T}{n} x_k(a - by_k) \\ y_{k+1}^* &= y_k + \dfrac{T}{n} y_k(cx_k - d) \end{cases} \quad \text{et} \quad \begin{cases} x_{k+1} &= x_k + \dfrac{T}{2n}(x_k(a - by_k) + x_{k+1}^*(a - by_{k+1}^*)) \\ y_{k+1} &= y_k + \dfrac{T}{2n}(y_k(cx_k - d) + y_{k+1}^*(cx_{k+1}^* - d)) \end{cases}$$

Travail demandé

1. Écrire une fonction `Heun_syst` prenant en entrées x_0, y_0, T et n et renvoyant les suites $(x_k)_{k\in[\![0,n]\!]}$ et $(y_k)_{k\in[\![0,n]\!]}$ obtenues par la méthode de Heun :

```
def Heun_syst(x0,y0,T,n):
    '''
    Entrées : [0,T] les extremités de l'intervalle de déf
              (x0,y0) une condition initiale
              n le nombre de subdivision
    Sortie : une approximation de la solution du système
    sur [0,T] avec la condition initiale respectée
    '''
    h=T/n  # pas
    X = [x0]
    Y = [y0]
    for i in range(1,n+1):
        x = X[-1]
        y = Y[-1]
        x_temp = x+h*F(x,y)[0]
        y_temp = y+h*F(x,y)[1]
        X.append(x+h/2*(F(x,y)[0]+F(x_temp,y_temp)[0]))
        Y.append(y+h/2*(F(x,y)[1]+F(x_temp,y_temp)[1]))
    return [X,Y]
```

2. Tracer les suites de points $\left(\left(k\frac{T}{n}, x_k\right)\right)_{k\in[\![0,n]\!]}$, $\left(\left(k\frac{T}{n}, y_k\right)\right)_{k\in[\![0,n]\!]}$ pour $x_0 = 1$, $y_0 = 0.5$, $T = 20$ et $n = 1000$.

```
[X,Y] = Heun_syst(x0,y0,T,n)
temps = [k*T/n for k in range(n+1)]
plt.plot(temps,X, ':', label='valeur approchée de x (proies)')
plt.plot(temps,Y,  label='valeur approchée de y (prédateurs)')
plt.legend()
plt.show()
```

On obtient le graphique 9 :

3. (a) Tracer la suite de points $\left(\left(k\frac{T}{n}, \varphi(x_k, y_k)\right)\right)_{k\in[\![0,n]\!]}$.

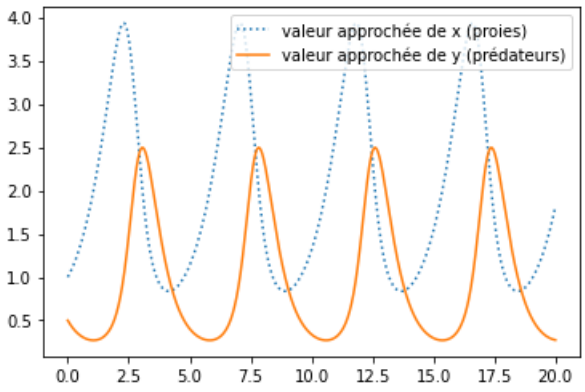

FIGURE 9 – Approximation des solutions par Heun

```
1 plt.plot(temps, phi(X,Y))
2 plt.ylim([2.19, 2.2])
3 plt.show()
```

On obtient la figure 10 :

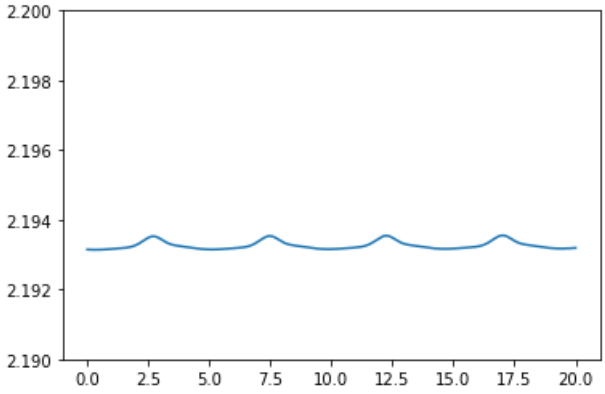

FIGURE 10 – Approximation des solutions par Heun

(b) D'après la propriété (P_1) attendue en théorie, la fonction $t \mapsto \varphi(x(t), y(t))$ est constante. On s'attend donc à ce que la courbe ci-dessus soit horizontale.

Même si ce n'est pas tout à fait le cas ici, le résultat est tout de même beaucoup plus proche d'une droite horizontale que dans le cas de la méthode d'Euler (voir la figure 7).

4. (a) Tracer la représentation graphique de (y_k) en fonction de (x_k) :

```
1 plt.plot(X,Y)
2 plt.show()
```

On obtient la figure 11 suivante :

(b) D'après la propriété (P_2) attendue en théorie, la fonction $t \mapsto (x(t), y(t))$ est périodique. On s'attend donc à ce que la courbe ci-dessous soit se referme sur elle-même.

Avec la méthode de Heun, cela semble être le cas contrairement à la méthode d'Euler (voir figure 8).

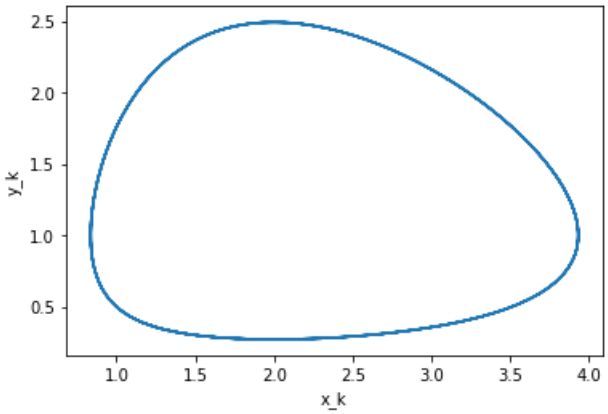

FIGURE 11 – Représentation graphique de (y_k) en fonction de (x_k)

3 Pendule pesant

On considère le système mécanique décrit en figure 12 composé d'une masse m suspendue à un fil rigide de longueur ℓ. La variable d'intérêt est l'angle θ que fait le fil avec la verticale et l'on cherche à décrire son évolution en fonction du temps t sachant qu'à un instant $t = t_0$ (on peut prendre $t_0 = 0$ pour fixer les idées) on a pour conditions initiales (position et vitesse angulaire initiales) :

$$\theta(t_0) = \theta_0 \quad , \quad \frac{d\theta}{dt}(t_0) = \omega_0.$$

La masse étant soumise à son poids et à la tension du fil rigide, le principe fondamental de la dynamique habilement projeté sur la direction tangentielle au mouvement se traduit en l'équation différentielle du second ordre :

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell}\sin\theta = 0 \qquad (\star)$$

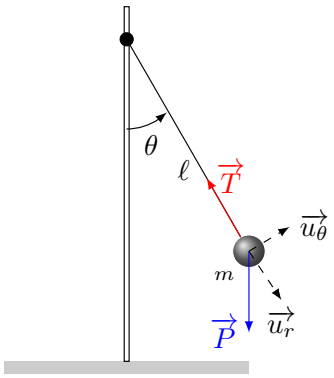

FIGURE 12 – Le pendule pesant

En posant

$$\omega = \frac{d\theta}{dt} \text{ (la vitesse angulaire)} \quad \text{et} \quad X = \begin{pmatrix}\theta\\ \omega\end{pmatrix} \quad ; \quad F(X) = \begin{pmatrix}\omega\\ -\frac{g}{\ell}\sin\theta\end{pmatrix},$$

on transforme l'équation $(\star)$ du second ordre en une équation vectorielle (un système d'équations scalaires) du premier ordre :

$$\frac{dX}{dt} = F(X).$$

3.1 Méthode d'Euler

L'équation $(\star)$ mise sous cette forme, avec des conditions initiales adéquates, peut être résolue approximativement sur un segment $[t_0, T]$ par un schéma d'Euler en considérant, pour tout $n \in \mathbb{N}^*$ la suite $(Y_k)_{[\![0,n]\!]}$ définie par :

$$Y_0 = \begin{pmatrix}\theta_0\\ \omega_0\end{pmatrix} \quad \text{et} \quad \forall k \in [\![0, n-1]\!], \quad Y_{k+1} = Y_k + \frac{T - t_0}{n} F(Y_k).$$

Travail demandé

1. Programmer la méthode d'Euler pour le pendule pesant avec $g = 9.80665$ et $\ell = 0.3$.

```
1 g =9.80665
2 l =0.3
3
4 def f(Y):
```

```
5           fY = np.zeros_like(Y)
6           fY[0] = Y[1]
7           fY[1] = -g/l* np.sin(Y[0])
8           return fY
9
10  def pendule(theta0,omega0,t0,tmax,n) :
11      """
12      Parameters
13      ----------
14      theta0 : float
15          l'angle initial.
16      omega0 : float
17          vitesse angulaire initiale.
18      n : nb de subdivisions
19      g_sur_ell : float, optional
20          le coeffficient g/ell. The default is 1.
21
22      Returns
23      -------
24      ndarray sol de shape (T,2) avec
25      sol[:,0] contenant les angles
26      sol[:,1] contenant les vitesses angulaires
27      """
28      h = (tmax-t0)/n
29      i_max = int(np.ceil(tmax / h))
30      # On créé un tableau qu'on va remplir petit à petit
31      thetas = np.zeros((i_max, 2))
32      # Conditions initiales
33      thetas[0, 0] = theta0
34      thetas[0, 1] = omega0
35      # Iteration
36      for i in range(0, i_max - 1):
37          # On applique la relation d'Euler vectorielle
38          thetas[i + 1] = thetas[i] + h * f(thetas[i])
39      return thetas
```

2. Tracer la courbe $(t, \theta(t))$ pour $t \in [0, T]$ avec $(\theta_0, \omega_0) = (0, 8)$ et $n = 1000$.

```
t0 = 0.0
tmax = 8
n = 1000
temps = [k*(tmax-t0)/n for k in range(n)]
theta0 = 0.0
omega0 = 8.0
```

On obtient la figure 13 suivante :

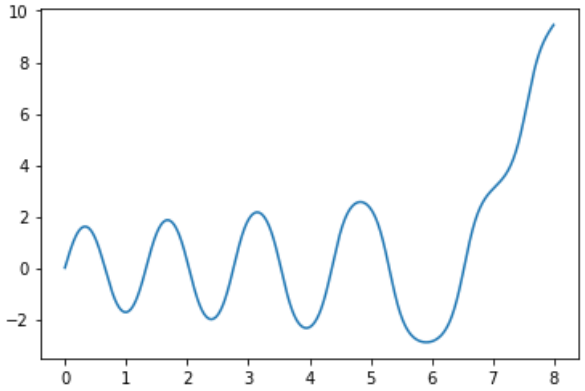

FIGURE 13 – Solution approchée par Euler

3. On constate que l'amplitude des oscillations augmente au cours du temps. C'est complètement incohérent avec la réalité physique.

3.2 Énergie conservée

Si on pose $X = (\theta, \omega)$ alors l'énergie mécanique totale du système est (à un facteur constant $m\ell^2$ près) :

$$E(X) = E(\theta, \omega) = \frac{1}{2}\omega^2 + \frac{g}{\ell}(1 - \cos(\theta)). \tag{E}$$

D'après la loi de conservation de l'énergie, pour $t \mapsto X(t)$ une solution de $(\star)$ sur un intervalle temporel I la fonction $t \mapsto E(X(t))$ est constante sur I.
La trajectoire de la courbe $t \mapsto X(t) = (\theta(t), \omega(t))$ est donc contenue dans une courbe de niveau de la fonction E, la courbe de niveau correspondant au niveau initial $E_0 = E(\theta_0, \omega_0)$.
La formule de l'énergie est réglée de sorte que cette énergie soit toujours positive et que la position d'équilibre stable $(\theta_*, \omega_*) = (0, 0)$ soit d'énergie $E(0, 0) = 0$.
La position d'équilibre instable est $(\theta^*, \omega^*) = (\pi, 0)$ et l'énergie de cette position est $E^* = 2\frac{g}{\ell}$.

Travail demandé

1. La fonction θ étant deux fois dérivables sur I (car solution d'une équation différentielle d'ordre 2), on sait que ω est dérivable. Par opération, la fonction $f : t \mapsto E(X(t))$ est dérivable sur I et on a pour tout $t \in I$:

$$\begin{aligned} f'(t) = \omega'(t)\omega(t) - \frac{g}{\ell}\theta'(t)\sin(\theta(t)) & \\ &= \theta''(t)\theta'(t) - \frac{g}{\ell}\theta'(t)\sin(\theta(t)) \\ &= \theta'(t)\left(\theta''(t) - \frac{g}{\ell}\sin(\theta(t))\right) \\ &= 0 \end{aligned}$$

car θ est solution de $(*)$. Ainsi f' est nulle sur I donc f est bien constante sur I.

2. Écrire une fonction `energie` prenant en entrées des flottant θ et ω et renvoyant $E(\theta, \omega)$.

```
1 def energie(theta,omega):
2     return 0.5*omega**2+g/l*(1-np.cos(theta))
3
4 Energie_Euler = energie(Sol_Euler[:,0],Sol_Euler[:,1])
5 plt.plot(temps,Energie_Euler)
6 plt.show()
```

3. En reprenant les mêmes valeurs qu'au paragraphe précédent, tracer la courbe $(t, E(\theta(t), \omega(t))$ où $t \mapsto (\theta(t), \omega(t))$ est la valeur approchée obtenue par la méthode d'Euler.
   On obtient la figure 14 suivante :

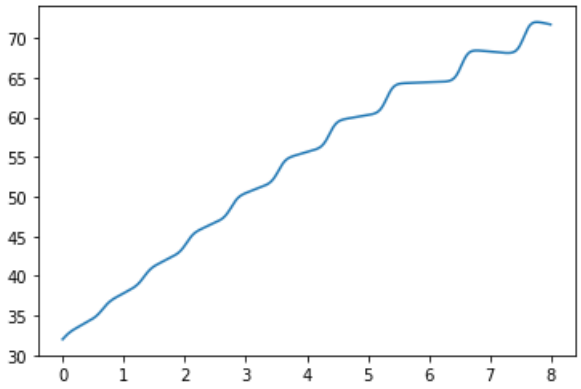

FIGURE 14 – Énergie obtenue par Euler

4. On constate que l'énergie augmente ! C'est contraire au principe de conservation de l'énergie !

5. (a) On peut implémenter la méthode de Heun qui s'est montrée plus performante sur les exemples précédents.
   (b) Programmer cette méthode et comparer les courbes obtenues.

```
1 def pendule_Heun(theta0,omega0,t0,tmax,n) :
2     """
3     Parameters
```

```
    ----------
    theta0 : float
        l'angle initial.
    omega0 : float
        vitesse angulaire initiale.
    n : nb de subdivisions
    g_sur_ell : float, optional
        le coeffficient g/ell. The default is 1.

    Returns
    -------
    ndarray sol de shape (T,2) avec
    sol[:,0] contenant les angles
    sol[:,1] contenant les vitesses angulaires
    """
    h = (tmax-t0)/n
    i_max = int(np.ceil(tmax / h))
    # On créé un tableau qu'on va remplir petit à petit
    thetas = np.zeros((i_max, 2))
    # Conditions initiales
    thetas[0, 0] = theta0
    thetas[0, 1] = omega0
    # Iteration
    for i in range(0, i_max - 1):
        thetas[i + 1] = thetas[i] + h * f(thetas[i])
        thetas[i+1]=thetas[i] + h/2*(f(thetas[i])+f(thetas[i+1]))
    return thetas
```

On obtient pour la courbe de l'approximation numérique de θ (voir figure 15) : On constate qu'avec cette méthode, l'amplitude des oscillations augmente

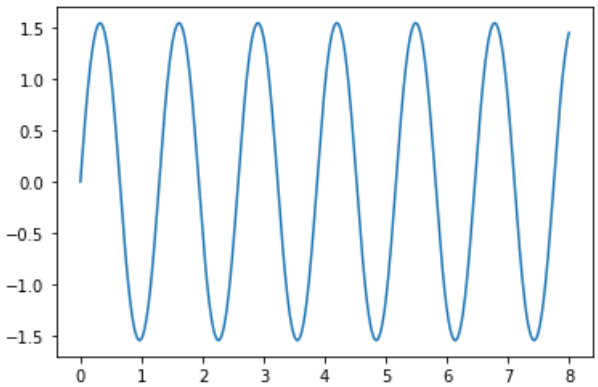

FIGURE 15 – Solution approchée par Heun

beaucoup plus lentement.

De même, pour l'énergie on obtient la figure 16

L'énergie augmente toujours mais de manière beaucoup moins significative par rapport à la méthode d'Euler.

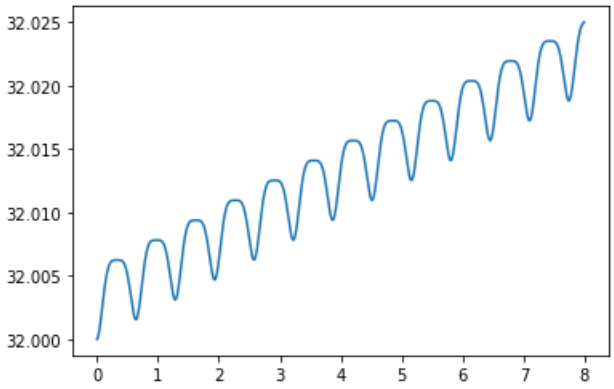

FIGURE 16 – Énergie obtenue par Heun