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 $\mathbb{R}_+$:

$$(G) \qquad y' = ay \ln\left(\frac{\kappa}{y}\right) \quad \text{et} \quad y(t_0) = y_0$$

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 la méthode d'Euler.
Dans toute cette partie, on prendra : `a` $= 0.036$ et `kappa` $= 760$, $(t_0, y_0) = (0, 16)$, $T = 200$.

1.1 Commande `odeint`

La commande `odeint` du module `scipy.integrate` permet de résoudre de manière approchée des équations différentielles. Pour résoudre une équation différentielle

$$y'(t) = F(y(t), t)$$

avec condition initiale $y(t_0) = y_0$ le script est le suivant

1. Solution approchée par `odeint`.

```
import scipy.integrate as integr

a = 0.036
kappa = 760
y0 = 16
T= 200

def F(y,t):
        return a*y*np.log(kappa/y)

Abs =  np.linspace(0,T,1000)
Y = integr.odeint(F,y_0,Abs)
plt.plot(Abs,Y,label='solution␣scipy')
```

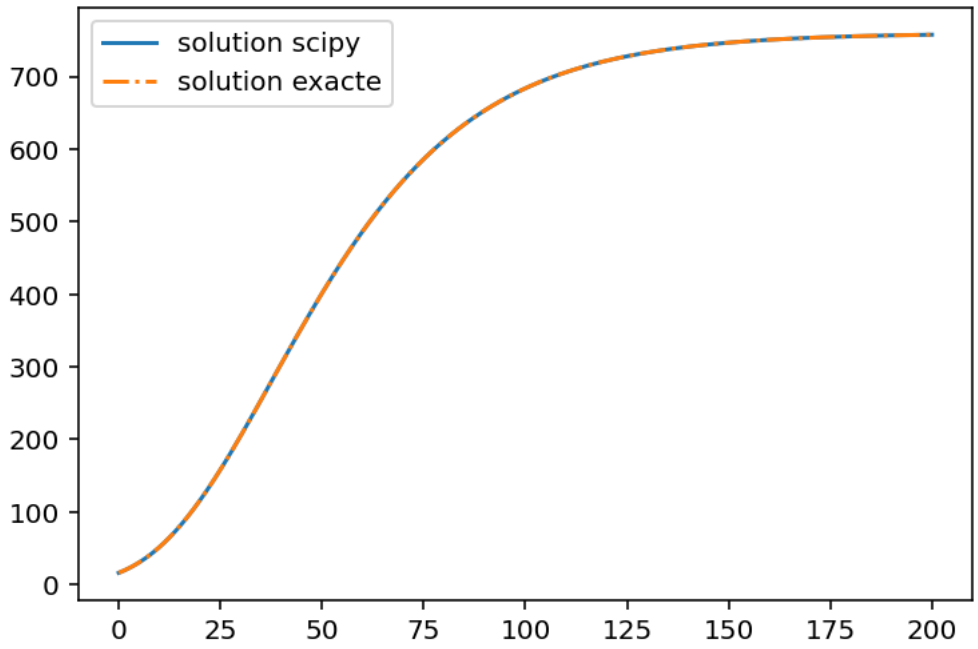
1. Voir l'exercice 7 du TD4 et son corrigé.

2. 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))

plt.plot(Abs,solution(Abs),'-.', label='solution exacte')
plt.legend()
plt.plot()
```

On obtient le graphique suivant qui montre que la solution approchée donnée par Python semble très proche de la solution exacte.



1.2 Méthode d'Euler

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

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

1. Programmation de la méthode d'Euler.

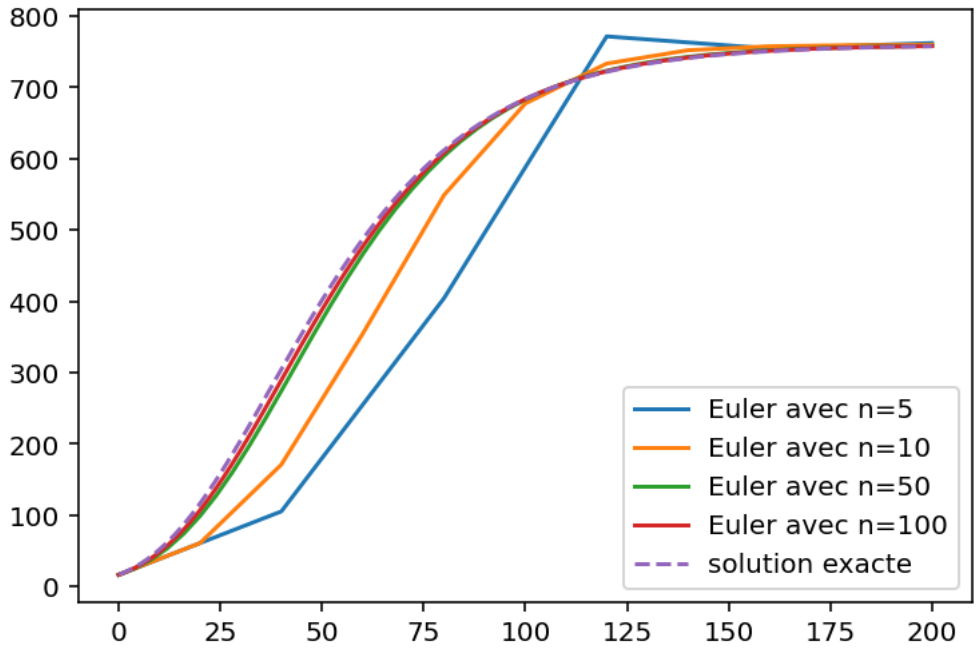
```
def Euler(y0,T,n):
    '''
    Entrées : [0,T] intervalle de déf. d'une solution
              y0 une condition initiale
              n le nombre de subdivision
    Sortie : une approximation de la solution de y'=F(y)
    sur [a,b] telle que y(a)=y0 avec Euler
    '''
    h=T/n  # pas
    Y=[y0]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

2. Solution approchée par Euler.

```
for n in [5,10,50,100]:
    Temps = [k*T/n for k in range(n+1)]
    Ord = Euler(y0,T,n)
    plt.plot(Temps,Ord,label='Euler avec n='+str(n))
#plt.plot(Abs,Y,label='solution scipy')

plt.plot(Abs,solution(Abs),'--', label='solution exacte')
plt.legend()
plt.plot()
plt.show()
```

On obtient le graphe ci-dessous qui montre que lorsque n tend vers $+\infty$, la solution approchée obtenue par la méthode d'Euler se rapproche de la solution exacte.



3. **Erreur de la méthode d'Euler :** l'erreur commise par la méthode avec n subdivisions sur $[0,T]$ est définie par :

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

Il s'agit de l'écart entre la solution approchée et la solution exacte.

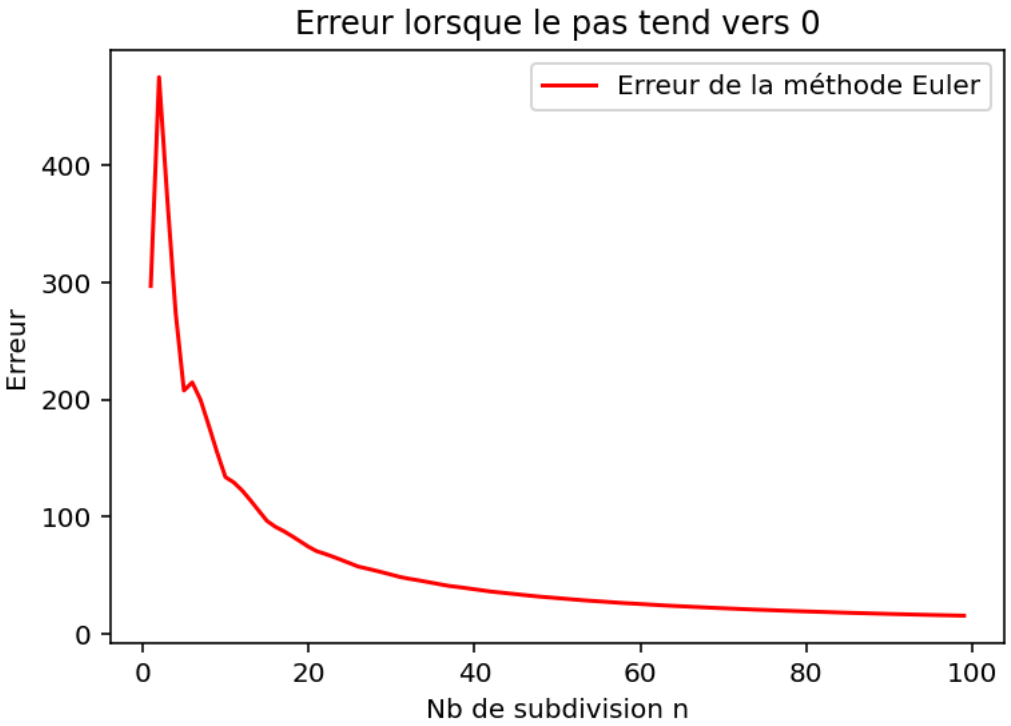
(a) La fonction `Erreur` prenant en entrées `T`, `y0` et `n` et renvoyant la valeur de $e_n(T)$.

```
1 def Erreur(y0,T,n):
2     Y1 = Euler(y0,T,n)
3     E1 = [np.abs(solution(k*T/n)-Y1[k]) for k in range(n+1)]
4     return max(E1)
```

(b) Tracé du graphique de l'erreur en fonction du nombre de subdivisions $n \in [\![1,100]\!]$.

```
1 Nb = range(1,100)   # nombre de subdivision
2 Erreur1= [Erreur(y0,T,k) for k in Nb]
3 plt.plot(Nb,Erreur1, color='red', label='Erreur')
4 plt.title('Erreur lorsque le pas tend vers 0')
5 plt.xlabel('Nb de subdivision n')
6 plt.ylabel('Erreur')
7 plt.legend()
8 plt.show()
```

On obtient le graphe suivante qui semble indiquer que l'erreur tend vers 0 quand n tend vers $+\infty$. Cela signifie que lorsque le pas tend vers 0, la solution approchée obtenue par la méthode d'Euler s'approche de la solution exacte.



2 Pendule pesant

On considère le système mécanique décrit en figure 1 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) :

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

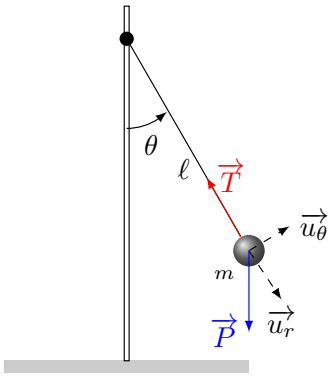


FIGURE 1 – Le pendule pesant

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 :

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

En posant

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

on transforme l'équation $(\star)$ du second ordre en une équation vectorielle (un système

d'équations scalaires) du premier ordre :

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

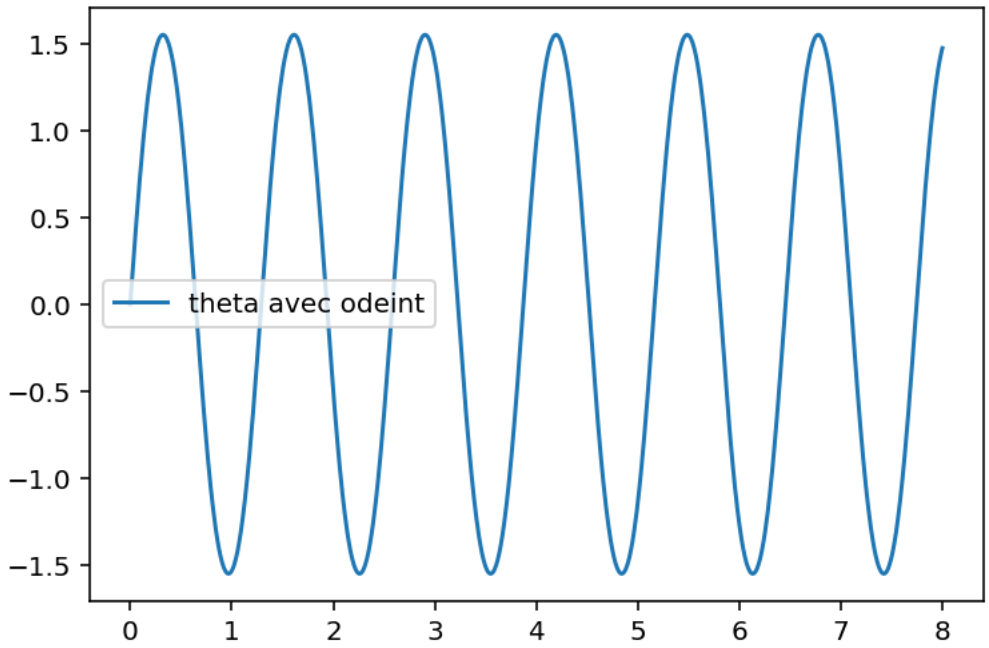
Pour les simulations numériques, on prendra : $T = 8$, $t_0 = 0$, $(\theta_0, \omega_0) = (0, 8)$, $g = 9.80665$ et $\ell = 0.3$.

2.1 Résolution avec la commande `odeint`

1. Solution approchée avec `odeint`.

```
1 def F(X,t):
2     return np.array([X[1],-g/l*np.sin(X[0])])
3
4 Abs = np.linspace(t0,T,1000)
5 X = integr.odeint(F,np.array([theta0,omega0]),Abs)
6 plt.plot(Abs,X[:,0],label="theta avec odeint")
7 plt.legend()
8 plt.show()
```

2. On obtient le graphe de θ suivant.



2.2 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 :

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

1. Méthode d'Euler pour le pendule pesant.

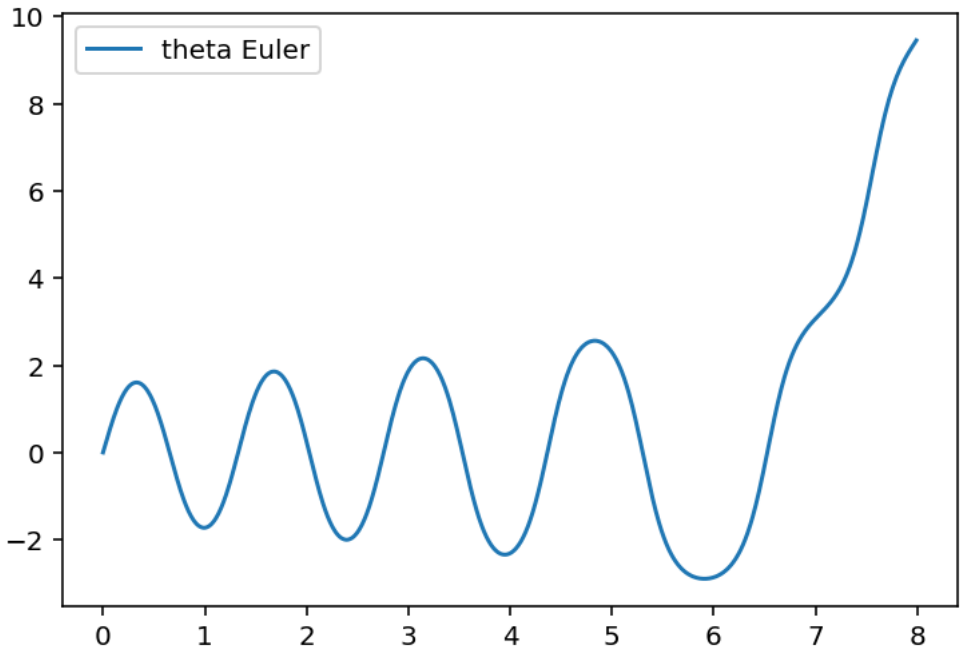
```
def f(Y):
        fY = np.zeros_like(Y)
        fY[0] = Y[1]
        fY[1] = -g/l* np.sin(Y[0])
        return fY

def pendule(theta0,omega0,t0,tmax,n) :
    """
    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):
        # On applique la relation d'Euler vectorielle
        thetas[i + 1] = thetas[i] + h * f(thetas[i])
    return thetas
```

2. Tracé de la courbe $(t, \theta(t))$ pour $t \in [0, T]$ avec $n = 1000$.

```
temps = [k*(T-t0)/n for k in range(n)]
Sol_Euler = pendule(theta0,omega0,t0,T,n)
plt.plot(temps,Sol_Euler[:,0],label='theta␣Euler')
plt.show()
```



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

2.3 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 $E^* = 2\frac{g}{\ell}$.

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 $E(\theta, \omega)$.

```
def energie(theta,omega):
    return 0.5*omega**2+g/l*(1-np.cos(theta))

Energie_Euler = energie(Sol_Euler[:,0],Sol_Euler[:,1])
plt.plot(temps,Energie_Euler)
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 2 suivante :

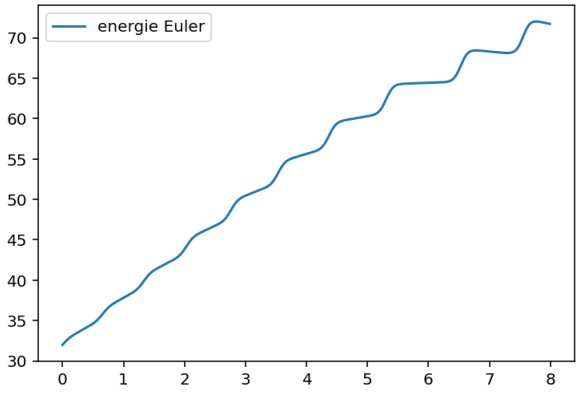


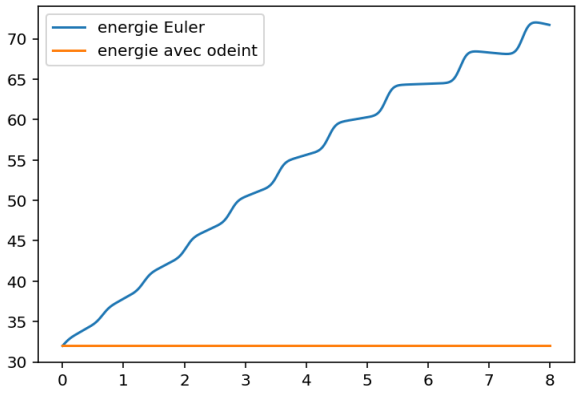
FIGURE 2 – Énergie obtenue par Euler

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

4. Pour l'énergie avec la solution obtenue par `odeint`.

```
Energie_scipy = energie(X[:,0],X[:,1])
plt.plot(Abs,Energie_scipy, label='energie avec odeint')
plt.legend()
plt.show()
```

   Là encore, on constate que l'énergie n'est pas constante mais en comparant avec la méthode d'Euler, on voit que les variations de l'énergie sont plus modérées.



3 Modèle proies-prédateurs de Lotka-Volterra

Soit a, b, c, d des réels strictement positifs.
On considère le système différentiel 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 que si x_0 et y_0 sont strictement positifs alors l'unique couple solution (x, y) vérifie la propriété (P_1) suivante :

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

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** $d = 2$**.**

3.1 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 :

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

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.

```
1 def F(x,y):
2     return [(1-y)*x,(x-2)*y]
3
4 def Euler_syst(x0,y0,T,n):
5     '''
6     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 $\left(\left(k\frac{T}{n}, x_k\right)\right)_{k\in[\![0,n]\!]}$, $\left(\left(k\frac{T}{n}, y_k\right)\right)_{k\in[\![0,n]\!]}$ pour $x_0 = 1$, $y_0 = 0.5$, $T = 20$ et $n = 1000$.

```
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 3 :

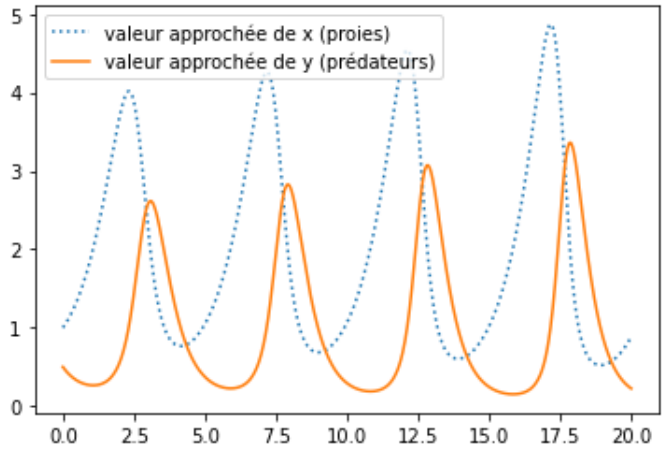


FIGURE 3 – Approximation des solutions par Euler

3. (a) Pour le tracer (voir figure 4) :

```
def phi(x,y):
```

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

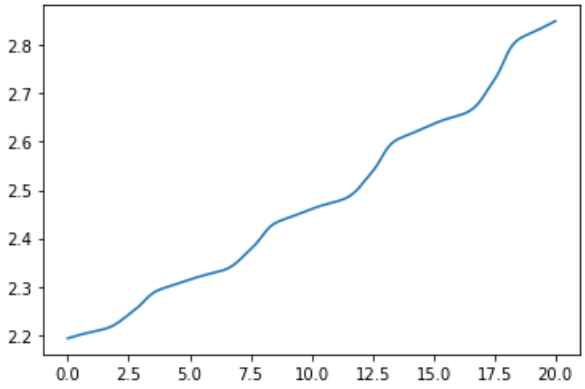


FIGURE 4 – Approximation des solutions par Euler

(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 dû aux erreurs d'approximation qui s'accumulent.

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

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

On obtient le graphique suivant :

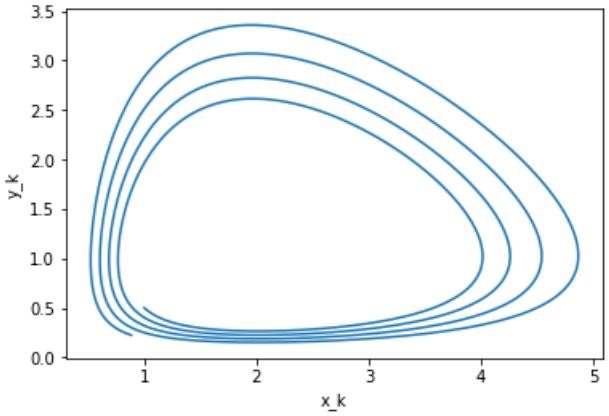


FIGURE 5 – Représentation graphique de (y_k) en fonction de (x_k)

(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.

3.2 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 :

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

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

```
[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 6 :

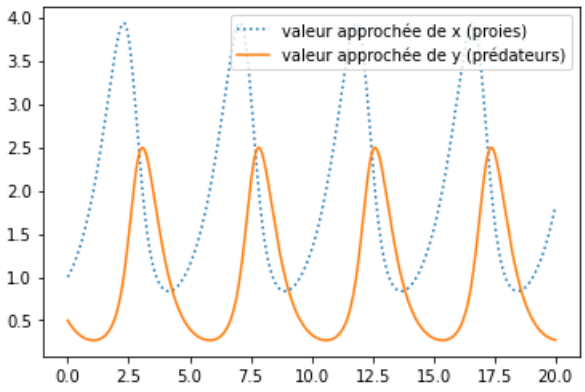


FIGURE 6 – Approximation des solutions par Heun

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

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

On obtient la figure 7 :

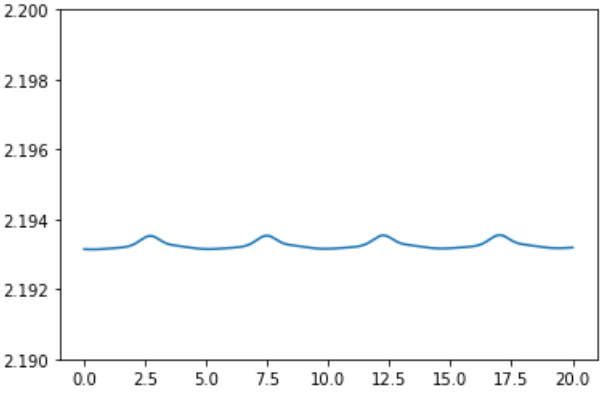


FIGURE 7 – Approximation des solutions par Heun

(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 4).

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 8 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.

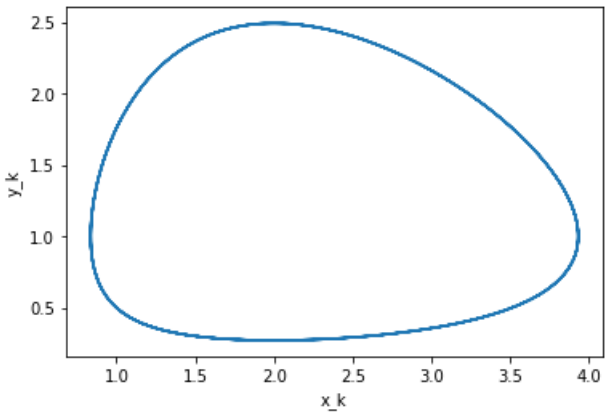


FIGURE 8 – Représentation graphique de (y_k) en fonction de (x_k)

Avec la méthode de Heun, cela semble être le cas contrairement à la méthode d'Euler (voir figure 5).