Modélisation 2 : Méthode d'Euler
Partie I

Contents

Préambule python :

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt, cos, sin, exp, pi
```

1 Théorie et mise en pratique de la méthode d'Euler

1.1 La méthode d'Euler

La méthode d'Euler est une méthode mathématique permettant de construire une suite de valeurs, approximant la fonction solution d'une équation différentielle. Le fait de construire une suite permet d'utiliser aisément cette méthode en informatique : on dit alors qu'on **discrétise** le problème.

Commençons avec le point de vue mathématique, pour une équation différentielle du premier degré, accompagnée de sa condition initiale, écrite ainsi :

$$\begin{cases} y'(t) = f(y(t)) \quad (E) \\ y(t_i) = y_i \end{cases}$$

En mathématiques, ce problème est nommé "problème de Cauchy".

Q 1 : Donner, pour les équations différentielles suivantes, la fonction f correspondante :

$$y'(t) = 3y(t) + 2 \qquad y'(t) + y^2(t) = 3y(t) \qquad 2y'(t) = ln(3y(t) - 5)$$

Lesquelles êtes-vous capables de résoudre théoriquement "*à la main*" ?

$y'(t) = 3y(t) + 2 = f(y(t))$ avec $f(u) = 3u + 2$, qu'on sait normalement résoudre théoriquement !

$y'(t) + y^2(t) = 3y(t) \iff y'(t) = 3y(t) - y^2(t) = f(y(t))$ avec $f(u) = 3u - 3u^2$

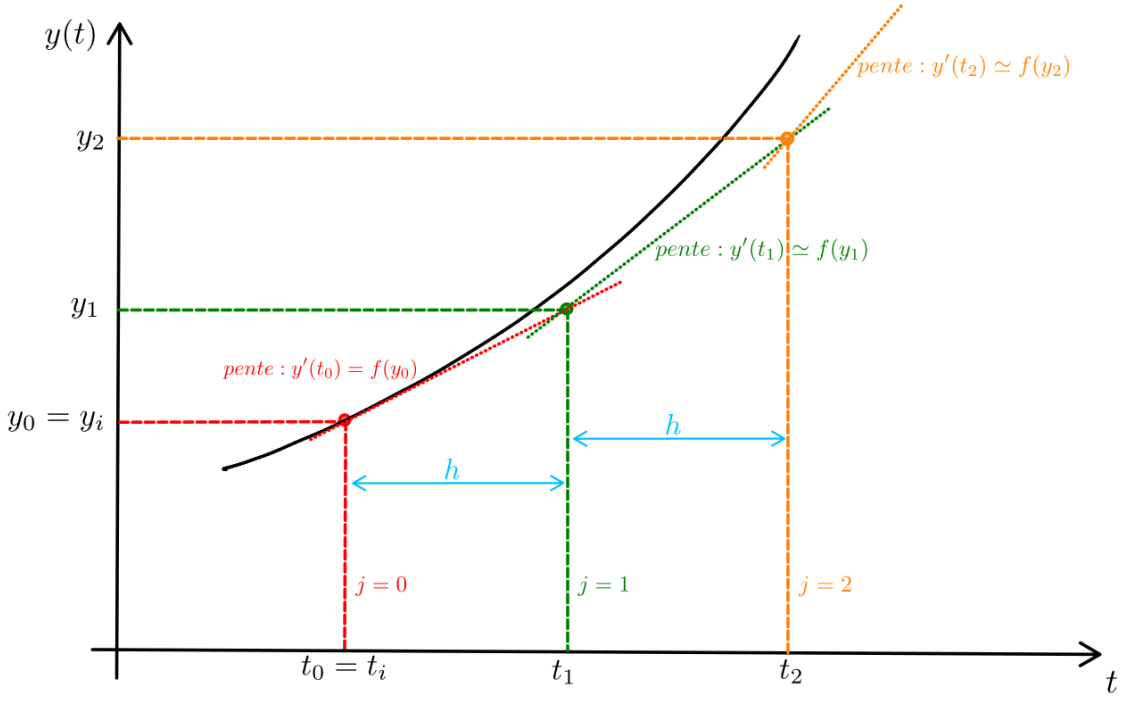
$2y'(t) = ln(3y(t) - 5) \iff y'(t) = \frac{1}{2}ln(3y(t) - 5) = f(y(t))$ avec $f(u) = \frac{1}{2}ln(3u - 5)$

Notre but est de résoudre ce problème sur un intervalle de temps $I = [t_i, t_f]$. **Résoudre l'équation de ce problème, c'est trouver la fonction $y(t)$ qui vérifie l'équation (E) et la condition initiale.**

L'équation (E) est une équation différentielle du premier ordre (car ne faisant intervenir, comme fonction, que $y(t)$ et $y'(t)$, la dérivée première de $y(t)$), mais pas forcément linéaire (elle ne l'est que si la fonction f l'est aussi). Nous allons ainsi, grâce à cette méthode, pouvoir résoudre approximativement des équations différentielles non linéaires ! Le travail préliminaire à effectuer est de discrétiser l'intervalle sur lequel nous allons résoudre cette équation.

Notons t_j les $n+1$ valeurs ($j \in [\![0, n]\!]$) discrètes prises par la variable t sur cet intervalle : $t_0 = t_i, t_1, t_2, ..., t_{n-1}, t_n = t_f$. Le mot *discret* s'oppose ici à *continu*. En effet, la variable t ne varie plus continûment entre t_i et t_f, mais par *bonds*, nommé *pas*, de valeur $h = \frac{t_f - t_0}{n}$ ($n+1$ valeurs pour les temps t_j, donc n pas). A chacune de ces valeurs t_j, nous allons faire correspondre une valeur y_j, valeur approchée de la fonction y en t_j : $y_j \simeq y(t_j)$. Il s'agit d'une valeur approchée car cette méthode n'est pas exacte.

La méthode d'Euler repose sur une construction itérative (proche de ce qu'on pourrait faire dans une démonstration par récurrence) : on part du premier point (indexé par $j = 0$), de coordonnées $(t_0 = t_i, y_0 = y_i)$, donné exactement par la condition initiale.



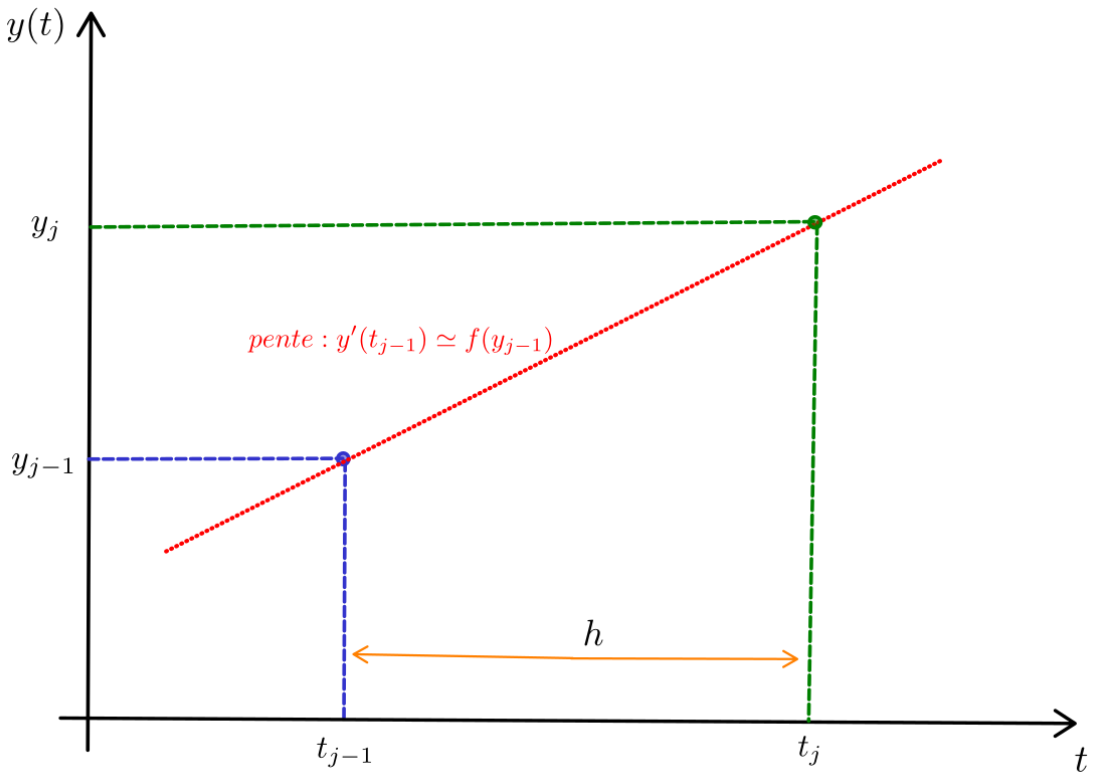
On construit ensuite la suite des points de coordonnées (t_j, y_j), en s'aidant de l'équation différentielle, selon le schéma indiqué par la figure ci-dessus (attention la courbe en noir est la courbe recherchée, elle n'est a priori pas connue !) :

- Pour obtenir y_1 à partir de y_0, on approxime la courbe recherchée par sa tangente en (t_0, y_0)

: c'est effectivement ce qu'on peut faire de mieux, avec les informations dont nous disposons. La pente de cette tangente est $y'(t_0) = f(y(t_0)) = f(y_0)$ d'après l'équation différentielle et on en déduit ensuite y_1 en "avançant" le temps du pas h jusqu'à la prochaine valeur discrète du temps t_1.

- De même, pour obtenir y_2 à partir de y_1, on approxime la courbe recherchée par sa tangente en (t_1, y_1): la pente de cette tangente est $y'(t_1) = f(y(t_1)) \simeq f(y_1)$ d'après l'équation différentielle et on en déduit ensuite y_2 en "avançant" le temps de h jusqu'au prochain pas t_2.
- Et ainsi de suite, on répète ces étapes encore $n-2$ fois jusqu'à arriver à la fin de l'intervalle, et obtenir les $n+1$ points de coordonnées (t_0, y_0), (t_1, y_1), (t_2, y_2), ..., (t_{n-1}, y_{n-1}),(t_n, y_n).

Cherchons le liens entre deux valeurs consécutives approchées de la fonction, c'est-à-dire exprimons y_j en fonction de y_{j-1}. Cela pourrait correspondre à une étape dite d'*hérédité* dans une démonstration par récurrence.



Q 2 : Comment s'écrit la pente de la tangente à la courbe représentant la fonction $y(t)$ en t_{j-1} ? L'exprimer à l'aide de la fonction f.

La pente p à la courbe représentant la fonction $y(t)$ en t_{j-1} s'écrit $p = y'(t_{j-1})$. Grâce à l'équation différentielle, on peut exprimer la dérivée en fonction de la fonction : $y'(t_{j-1}) = f(y(t_{j-1}))$, puis, en approximant la valeur de la fonction $y(y)$ en t_{j-1} :

$$p = y'(t_{j-1}) \approx f(y_{j-1})$$

Q 3 : En s'aidant du graphe ci-dessus, exprimer alors cette même pente, mais cette fois-ci en fonction de y_j, y_{j-1}, t_j et t_{j-1}, puis en fonction de y_j, y_{j-1} et h.

Aide : Écrire un taux d'accroissement.

Taux d'accroissement :

$$p = \frac{y_j - y_{j-1}}{t_j - t_{j-1}} = \frac{y_j - y_{j-1}}{h}$$

Q 4 : Déduire des deux questions précédentes l'expression de y_j en fonction de y_{j-1}, h et $f(y_{j-1})$.

$$p = \frac{y_j - y_{j-1}}{h} \approx f(y_{j-1})$$

On en déduit :

$$y_j \approx y_{j-1} + h \times f(y_{j-1})$$

En modélisation, pour retrouver cette formule (et en trouver d'autres plus complexes et/ou générales…), on préférera utiliser des D.L.

Q 5 : Exprimer y_j en fonction de y_{j-1}, via un D.L.(1) de $y(t_j)$ en t_{j-1}. Retrouver alors la formule approchée précédente.

D.L.(1) :

$$y(t_j) = y(t_{j-1}) + (t_j - t_{j-1}) \times y'(t_{j-1}) + o(t_j - t_{j-1}) = y(t_{j-1}) + h \times y'(t_{j-1}) + o(h)$$

En remplaçant $y'(t_{j-1})$ par $f(y(t_{j-1})) \approx f(y_{j-1})$, on remarque alors qu'effectivement, l'expression approchée est bien :

$$y_j \simeq y_{j-1} + h \times f(y_{j-1})$$

Cette expression devient exacte lorsque le pas h tend vers 0, ce qui revient à prendre n infiniment grand. Malheureusement, cela implique aussi un nombre de calculs pour l'ordinateur infiniment grand…

Pour obtenir toutes les valeurs de la fonction y sur l'intervalle discrétisé, il faut alors construire la suite des y_j en utilisant la condition initiale : $y_0 = y_i$. On peut alors construire la suite des y_j de proche en proche, selon le schéma suivant :

$$\begin{cases} y_0 = y_i \\ y_j = y_{j-1} + h \times f(y_{j-1}),\ j \in [\![1, n]\!] \end{cases}$$

Les valeurs discrétisées du temps sont aussi nécessaires

Q 6 : Exprimer la suite des valeurs t_j pour $j \in [\![0, n]\!]$.

Pour le temps :

$$\begin{cases} t_0 = t_i \\ t_j = t_{j-1} + h,\ j \in [\![1, n]\!] \end{cases}$$

ou même, de façon explicite :

$$t_j = t_i + j \times h,\ j \in [\![0, n]\!]$$

Un exemple "simple"

Testons sur l'exemple le plus simple cette méthode, pour cela on considère le problème de Cauchy suivant :

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

La solution de ce problème est naturellement la fonction exponentielle, mais effectuons la résolution numérique approchée en utilisant la méthode d'Euler.

Q 7 : Écrire la fonction `Euler_simple(ti,tf,yi,n)` retournant les listes (ou tableaux numpy) `T` et `Y`, listes des temps et des valeurs approchées de la fonction y(t) aux temps correspondants, par la méthode d'Euler avec $n+1$ points. Il est conseillé, dans un premier temps, de tenter de partir sans aide, pour faire sa "*propre*" version. En cas de difficultés, vous pouvez vous aider du code à trous fournis dans le fichier *M2_Euler.py*.

Correction : différentes versions ! Celle suivant le code à trous :

[8]:
```
def Euler_simple(ti,tf,yi,n):
    """ Retourne les listes T et Y, selon la méthode d'Euler.
    La liste T contient les n+1 valeurs du temps discrétisées, allant de ti à␣
 ↪tf avec un pas constant
    La liste Y contient les n+1 valeurs de la fonction y(t) correspondants aux␣
 ↪valeurs du temps
    Condition Initiale : yi = y(ti)
    Equa diff : y'(t) = y(t)
    """
    # Pas temporel :
    h = (tf-ti)/n # intervalle total allant de longueur tf-ti, contenant n␣
 ↪sous-intervalles de longueur h
    # Création des tableaux contenant pour l'instant des 0 pour le temps t et␣
 ↪l'ordonnée y, pour les n+1 valeurs discrétisées
    T, Y = np.zeros(n+1), np.zeros(n+1)
    # C.I.
    T[0] = ti   #t0 = ti
    Y[0] = yi   #y0 = yi
    # Boucle d'"hérédité", soit la méthode d'Euler
    for j in range(1, n+1): # Pour les n points restant à faire
        T[j] = T[j-1]+h # valeur de tj en fonction de tj-1
        # Autre possibilité : T[j] = ti + j*h
        Y[j] = Y[j-1]+h*Y[j-1] # valeur de yj en fonction de yj-1 et tj-1
    return T,Y
```

Sans tableau numpy, juste avec des listes, et la fonction append :

[7]:
```
def Euler_simple2(ti,tf,yi,n):
    h = (tf-ti)/n
    # C.I.
    T=[ti]
    Y=[yi]
    for j in range(1,n+1):
```

```
        T.append(T[j-1]+h) # nouvelle valeur de t
        Y.append(Y[j-1]+h*Y[j-1]) # nouvelle valeur de y
    return T,Y
```

Autre version, en écrivant dès le début le tableau des T :

[6]:
```
def Euler_simple3(ti,tf,yi,n): # on suit toujours nos notations
    h = (tf-ti)/n # pas
    Y = np.zeros(n+1) # on créé un tableau  contenant pour l'instant des 0 pour␣
 ↪l'ordonnée y, pour les n+1 valeurs discrétisées
    # C.I.
    Y[0] = yi
    # T
    T = [ti+j*h for j in range(n+1)]
    # Boucle d'"hérédité", soit la méthode d'Euler
    for j in range(1, n+1):
        Y[j] = Y[j-1]+h*Y[j-1] # valeur de yj en fonction de yj-1 et tj-1
    return T,Y
```

De manière générale, je conseille ici l'emploi de la syntaxe `Y[j]=...`, et non `Y.append(...)`; de même `Y[j-1]` plutôt que `Y[-1]` quand cela est possible, cela évite bien des erreurs ! Mais voici malgré tout le code avec cette autre solution :

[5]:
```
def Euler_simple4(ti,tf,yi,n):
    h = (tf-ti)/n
    # C.I.
    T=[ti]
    Y=[yi]
    for k in range(n):
        Y.append(Y[-1]+h*Y[-1]) # nouvelle valeur de y
        T.append(T[-1]+h) # nouvelle valeur de t
    return T,Y
```

Encore une autre version, en employant ici des variables intermédiaires t_j et y_j:

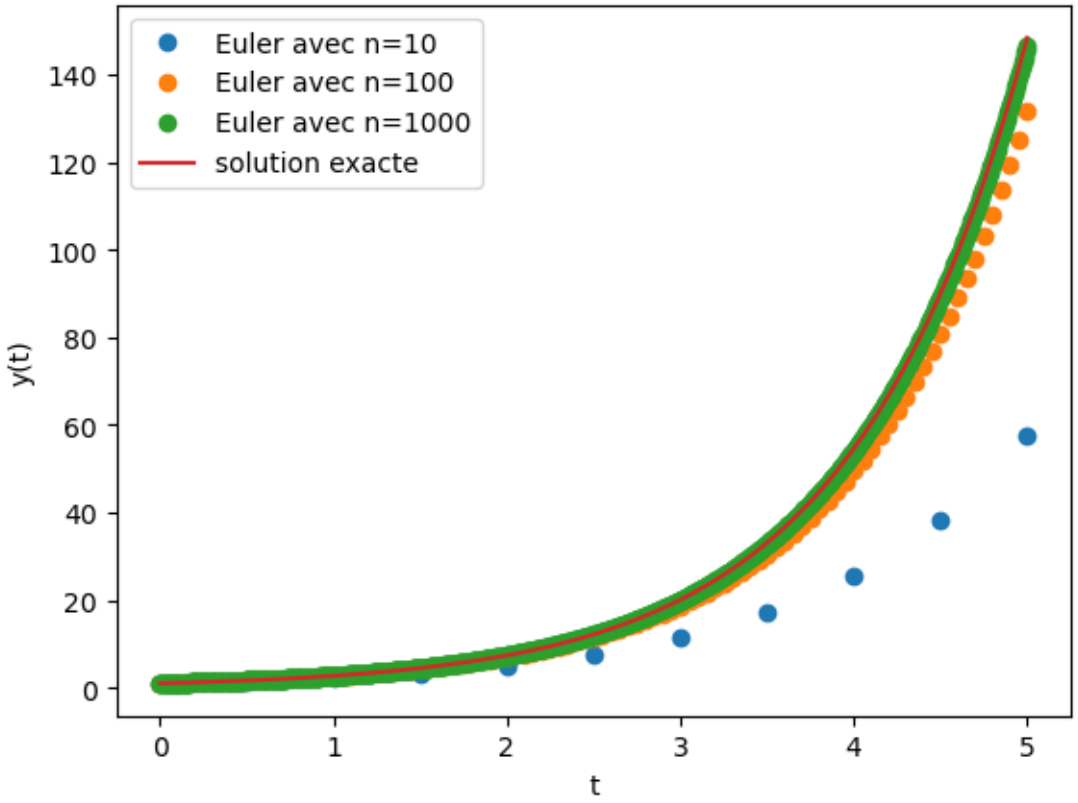
[4]:
```
def Euler_simple5(ti,tf,yi,n):
    h = (tf-ti)/n
    # C.I.
    T=[ti]
    Y=[yi]
    # Valeurs initiales de tj et yj (on est pour l'instant à j = 0)
    tj = ti
    yj = yi
    # Méthode d'Euler (hérédité)
    for k in range(n):
        # mise à jour des valeurs de tj et yj
        tj, yj = tj + h, yj + h*yj
        T.append(tj) # nouvelle valeur de t
```

```
        Y.append(yj) # nouvelle valeur de y
    return T,Y
```

Personnellement, je déconseille ces dernières versions (voir suite) ... mais l'essentiel est de trouver votre version, et de l'écrire juste !

Q 8 : Utiliser la fonction `Euler_simple` définie précédemment pour réaliser un graphe comparant la solution numérique par méthode d'Euler et la solution exacte du problème de Cauchy. On fera la résolution sur l'intervalle $[0, 5]$, et on étudiera l'influence du choix de n.

[14]:
```
for k in np.logspace(1,3,3):
    T,Y = Euler_simple(0,5,1,int(k))
    plt.plot(T,Y,'o',label='Euler avec n='+str(int(k)))

plt.plot(T,np.exp(T), label ='solution exacte')

plt.xlabel("t")
plt.ylabel("y(t)")
plt.legend()
plt.show()
```



On remarque que naturellement, plus le nombre de discrétisation n est grand, plus la solution

approchée se rapproche de la solution exacte.

Remarque : La méthode utilisant un D.L. est plus générale, et permettra de trouver des méthodes similaires à celle d'Euler, mais pour des équations plus complexes (d'ordre 2, avec plusieurs variables). Pour cela, il est nécessaire de bien connaître la formule de Taylor-Young des D.L. :

$$f(x+h) = f(x) + \sum_{k=1}^{n} \frac{h^k}{k!} \frac{\mathrm{d}^k f}{\mathrm{d}x^k}(x) + o(h^n)$$

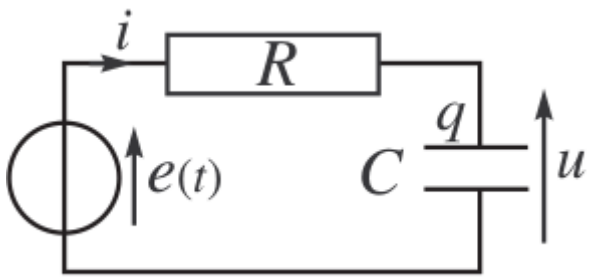
Ou bien :

$$f(x) = f(x_0) + \sum_{k=1}^{n} \frac{(x-x_0)^k}{k!} \frac{\mathrm{d}^k f}{\mathrm{d}x^k}(x_0) + o((x-x_0)^{n+1})$$

2 Résolution d'un problème de physique

2.1 Circuit RC - version "simple"

On considère le circuit RC suivant :



Q 9 : Montrer que l'équation différentielle sur u_C peut s'écrire :

$$\frac{du(t)}{dt} + \frac{u(t)}{\tau} = \frac{e(t)}{\tau}$$

Donner l'expression de τ.

$\tau = RC$

On cherche à trouver la solution approchée de cette équation différentielle à l'aide de la méthode d'Euler.

Q 10 : Donner l'expression de la fonction f dans ce cas du circuit RC.

On peut ré-écrire l'équation différentielle sous la forme :

$$\frac{du(t)}{dt} = \frac{e(t) - u(t)}{\tau} = f(u(t))$$

Avec :

$$f(x) = \frac{e - x}{\tau}$$

Remarque : pour être complétement rigoureux, il faudrait écrire $\frac{du(t)}{dt} = \frac{e(t)-u(t)}{\tau} = f(u(t), t)$ *avec* $f(x,t) = \frac{e(t)-x}{\tau}$.

On considère dans un premier temps que la source de tension génère un échelon de tension :

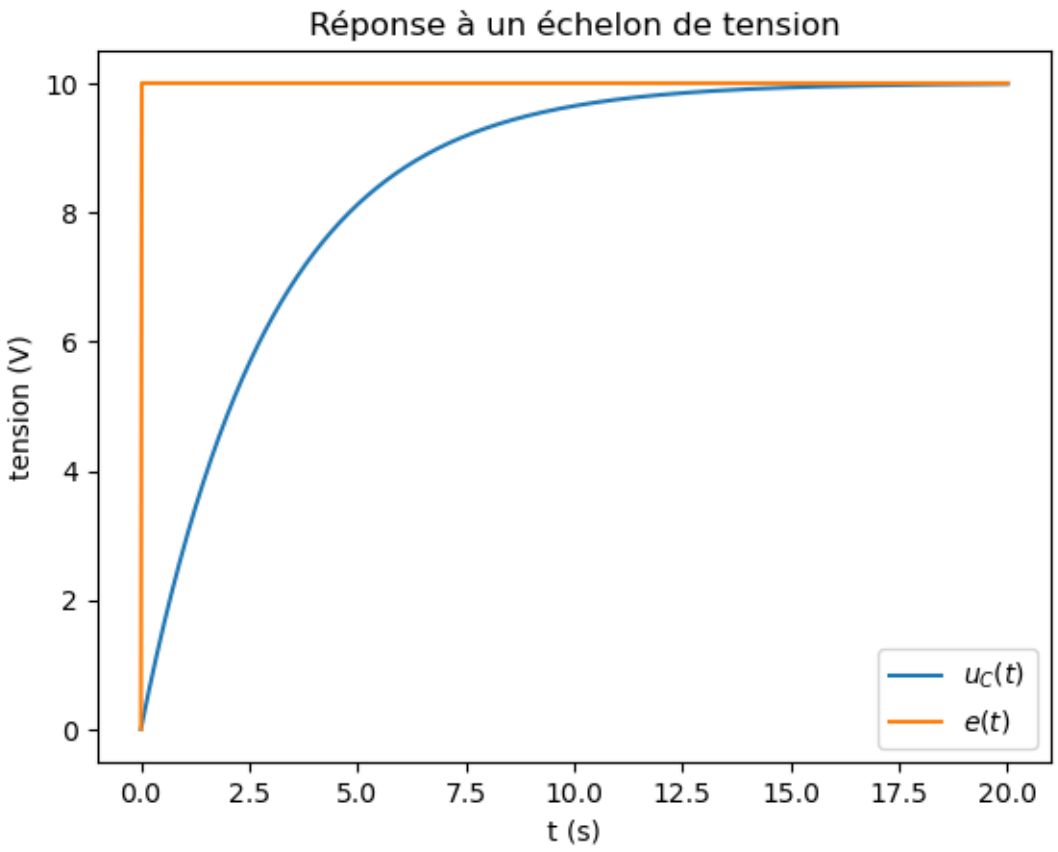
$$e(t) = \begin{cases} 0 & \text{pour } t \leq 0 \\ 10 & \text{pour } t > 0 \end{cases}$$

On rappelle qu'alors, on a la C.I. $u_C(0) = 0$.

Q 11 : Pour cet échelon de tension, construire alors les listes `T` et `U` contenant 1001 valeurs de t et $u_C(t)$ sur l'intervalle $I = [0, 20]$ (en secondes), pour $\tau = 3s$ et avec la condition initiale $u_C(0) = 0$. Tracer ensuite le graphique de $u_C(t)$ et $e(t)$ en fonction de t sur l'intervalle I.

[15]:
```
n = 1000
tau = 3
ti = 0
tf = 20
h = (tf - ti) / n

T, U = np.zeros(n+1), np.zeros(n+1)

for j in range(n):
    T[j+1] = T[j] + h
    U[j+1] = U[j] + h * (10 - U[j]) / tau
```

[16]:
```
E = [10 for temps in T]
E[0] = 0

plt.plot(T,U,label=r'$u_C (t)$')
plt.plot(T,E,label=r'$e (t)$')
plt.xlabel("t (s)")
plt.ylabel("tension (V)")
plt.title("Réponse à un échelon de tension")
plt.legend()

plt.show()
```



2.2 Circuit RC - version plus générale

Tout le travail précédent n'a que peu d'intérêt théorique, car nous sommes capable de déterminer analytiquement la solution :

$$u_C(t) = 10\left(1 - exp\left(-\frac{t}{\tau}\right)\right)$$

On peut adopter un second point de vue, en définissant une fonction Euler adaptée à ce problème spécifique (circuit RC), mais prenant en argument une fonction "*générale*" $e(t)$. Ainsi on pourra varier l'allure de la fonction générée par le générateur, et ne pas se limiter à un simple échelon de tension ! On en profite aussi pour passer en argument τ et la C.I..

Q 12 : Écrire la fonction `Euler_RC(e,ti,tf,ui,tau,n)` donnant, par la méthode d'Euler, une solution approchée de cette équation différentielle. Cette fonction devra retourner deux listes ou tableaux `T` et `U` contenant les $n+1$ valeurs de t et $u_C(t)$ sur l'intervalle $I = [t_i, t_f]$. `e` est la fonction de la variable `t` donnant la tension aux bornes du générateur $e(t)$ (ne pas l'expliciter ici), `tau` le temps caractéristique du circuit et `ui` la condition initiale `ui`$= u_C(t_i)$.

[2]:
```
def Euler_RC(e,ti,tf,ui,tau,n):
    h = (tf-ti)/n # pas
    T, U = np.zeros(n+1), np.zeros(n+1)
```

```
    # C.I.
    T[0] = ti    #t0 = ti C.I. sur le temps
    U[0] = ui    #u0 = ui  C.I. sur la tension aux bornes de C
    # Boucle d'"hérédité", soit la méthode d'Euler
    for j in range(1, n+1):
        T[j] = T[j-1]+h # nouvelle valeur du temps en fonction de l'ancienne␣
↪tj-1
        U[j] = U[j-1]+h*(e(T[j-1])-U[j-1])/tau # nouvelle valeur de uc en␣
↪fonction de l'ancienne ucj-1 et de e et tj-1
    return T, U
```

Intéressons-nous désormais à des problèmes plus intéressants, pour lesquels nous ne sommes pas capables de déterminer la solution analytique. Essayons donc un double échelon :

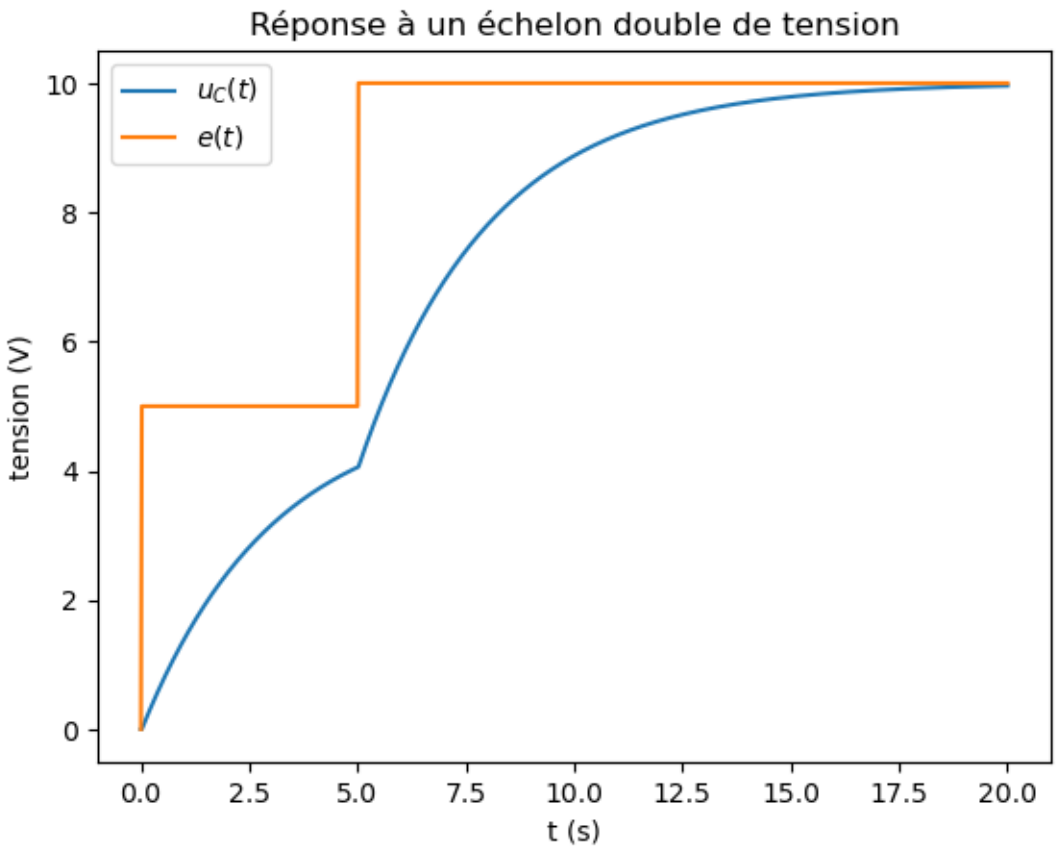
Q 13 : On a définit une nouvelle fonction `echelon_double(t)` selon la fonction suivante :

$$e(t) = \begin{cases} 0 & \text{pour } t \leq 0 \\ 5 & \text{pour } 0 < t < 5 \\ 10 & \text{pour } 5 \leq t \end{cases}$$

Écrire la définition de la fonction python correspondante puis faire la résolution de l'équation différentielle via la méthode d'Euler en utilisant la fonction `Euler_RC`. Tracer la solution obtenue.

[19]:
```
def echelon_double(t):
    if t <= 0 :
        return 0
    elif t < 5 :
        return 5
    else :
        return 10
```

[20]:
```
T, U = Euler_RC(echelon_double,0,20,0,3,1000)
E = [echelon_double(temps) for temps in T]

plt.figure()
plt.plot(T,U,label=r'$u_C (t)$')
plt.plot(T,E,label=r'$e (t)$')
plt.xlabel("t (s)")
plt.ylabel("tension (V)")
plt.title("Réponse à un échelon double de tension")
plt.legend()
plt.show()
```

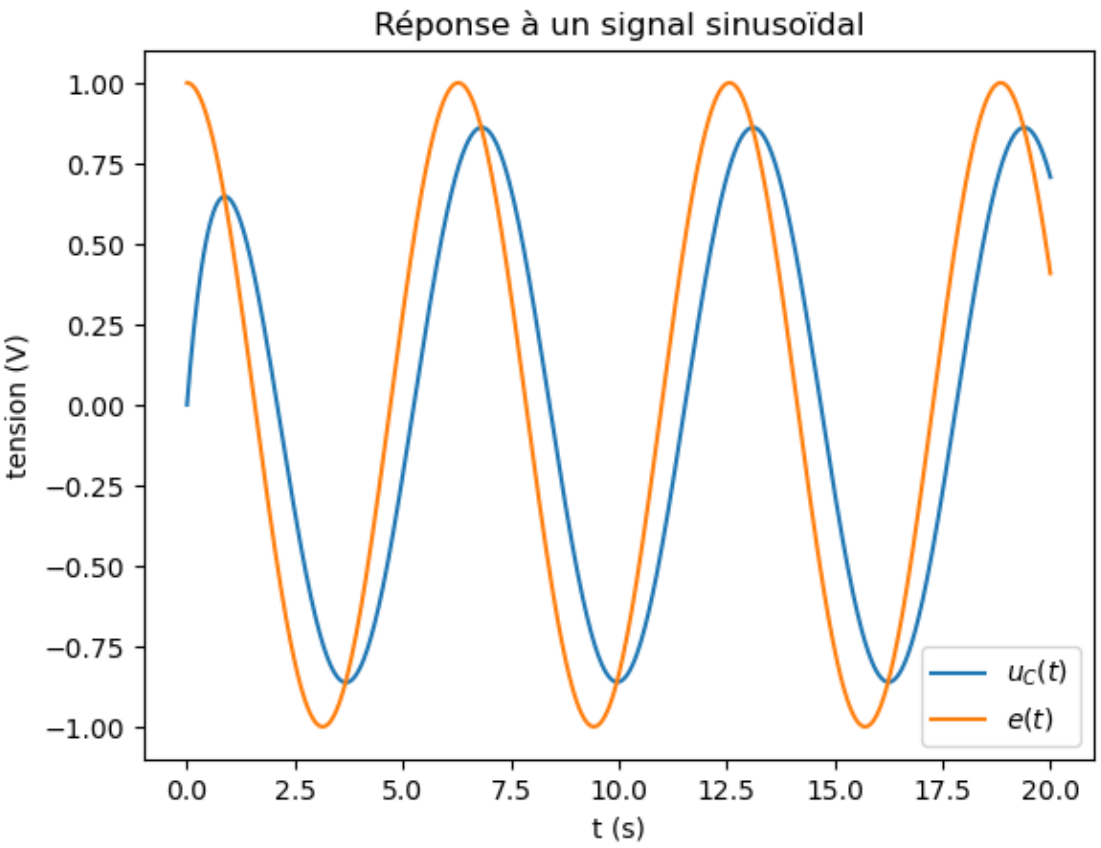


Q 14 : Essayer avec une autre fonction pour la tension aux bornes du générateur.

Essayons avec une tension sinusoïdale :

[9]:
```
def cosinus(t):
    return cos(t)
```

[17]:
```
T, U_cosin = Euler_RC(cosinus,0,20,0,0.6,1000)
E_cosin = [cosinus(temps) for temps in T]

plt.figure()
plt.plot(T,U_cosin,label=r'$u_C (t)$')
plt.plot(T,E_cosin,label=r'$e (t)$')
plt.xlabel("t (s)")
plt.ylabel("tension (V)")
plt.title("Réponse à un signal sinusoïdal")
plt.legend()
plt.show()
```



Après un cours régime transitoire (de l'ordre de $3\tau \approx 2\ s$), on se retrouve en R.S.F.