En physique ou en sciences de l'ingénieur, nous recherchons souvent l'évolution temporelle d'une grandeur caractéristique du système étudié.

Parfois l'évolution de cette grandeur est connue par des valeurs numériques recueillies par une expérience.

L'expérience peut nous avoir donné un tableau des temps `T=[t0,t1, ...]` et un tableau des valeurs de la grandeur étudiée à ces instants `Y=[y0, y1, ...]`.

On peut vouloir en déduire un tableau `dY` des valeurs approchées de la dérivée :

```
T=[t0,t1, ...]
Y=[y0, y1, ...]
n=len(Y)
dY=[]


.
```

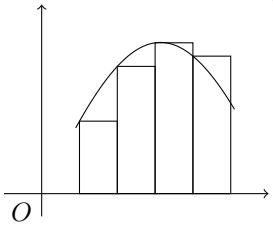
Quel est la taille du tableau `dY` ?

Parfois les intervalles de temps sont réguliers et on n'a pas besoin du tableau des temps mais juste de l'intervalle de temps `dt`

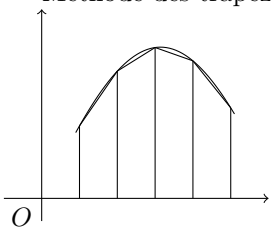
```
Y=[y0, y1, ...]
n=len(Y)
dt = ... #intervalle de temps
dY=[]


.
```

A l'inverse, on peut vouloir "intégrer numériquement" la fonction, c'est à dire soit calculer l'aire sous la courbe :

Méthode des rectangles :



Méthode des trapèzes :



soit approcher numériquement une primitive : c'est la méthode d'Euler.

L'étude théorique peut nous amener à une équation différentielle non linéaire. Les outils mathématiques ne nous permettent pas toujours la résolution exacte de ce type d'équation, on utilise alors une méthode numérique pour approcher la solution.

I Equations différentielles de degré 1

Le théorème de Cauchy-Lipschitz assure « sous des conditions raisonnables » sur F qu'il existe une unique application y de classe $\mathscr{C}^1$ sur $[a;b]$ telle que : $\begin{cases} y(a) &= y_0 \\ \dfrac{dy}{dt} &= F(t,y) \end{cases}$

Nous utiliserons F de classe au moins $\mathscr{C}^1$ sur $[a;b]$ donc y sera au moins $\mathscr{C}^2$ sur $[a;b]$.

La théorie nous dit qu'une solution existe, mais pas comment la trouver ! L'objectif des schémas numériques est d'obtenir une approximation de cette solution, par la construction d'un certain nombre de points sur un intervalle $[a;b]$.

Au programme, le **schéma d'Euler explicite** s'appuie sur un découpage de l'intervalle $[a,b]$ en sous intervalles suffisamment petits sur lesquels on considérera la dérivée comme constante (on assimile la courbe à sa tangente).

— On peut choisir directement le pas dt ou le nombre de points à construire n.
Sur l'intervalle $[a;b]$, si on construit donc n nouveaux points à partir du point $x=a$ (déjà connu), ils seront donc espacés d'un pas fixe : $dt = \dfrac{b-a}{n}$.
On aura donc $t_k = a + k \times dt$, où k varie entre 0 et n.

— On va calculer pas à pas une approximation y_k de $y(t_k)$.
$y_1 = y_0 + F(t_0,y_0) \times dt$
$y_2 = y_1 + F(t_1,y_1) \times dt$
$y_3 = y_2 + F(t_2,y_2) \times dt$
...
$y_{k+1} = y_k + F(t_k,y_k) \times dt$

Justification mathématique :

$$\int_{t_k}^{t_{k+1}} \frac{dy}{dt} dt = y(t_{k+1}) - y(t_k) \approx y_{k+1} - y_k$$
$$= \int_{t_k}^{t_{k+1}} F(t,y(t))dt \approx F(t_k, y(t_k)) \times dt \approx F(t_k, y_k) \times dt$$

Si l'intervalle $[t_k;t_{k+1}]$ est suffisamment petit, on peut considérer $F(t,y(t))$ comme constante, égale à $F(t_k,y(t_k))$. Mais comme on ne connait pas exactement $y(t_k)$ mais seulement son approximation y_k, on approxime cette dérivée $F(t_k,y(t_k))$ par $F(t_k,y_k)$. Il y a deux niveaux d'approximation.

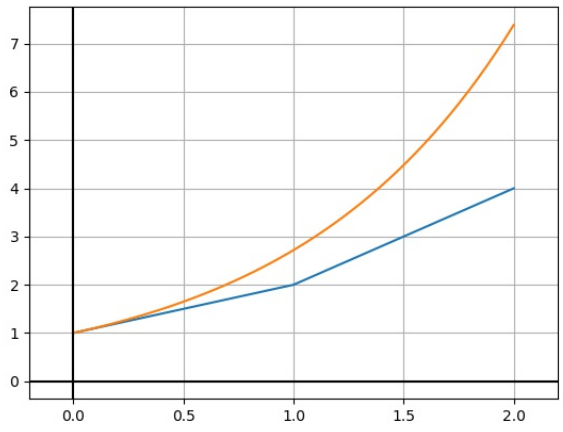
Écrivons ensemble la fonction `Eulerexplicite(F,a,b,y0,n)` :

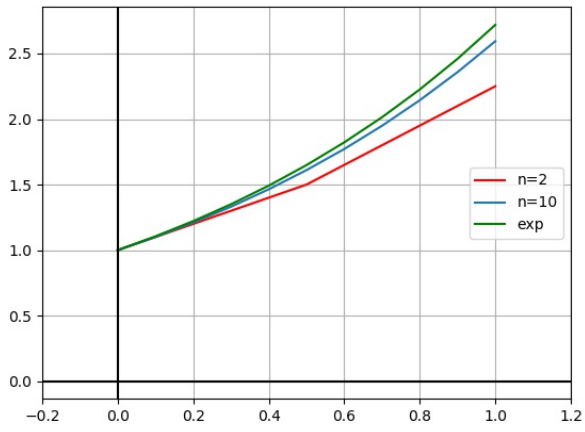
Quelques exemples très simples afin de comprendre l'importance du choix du pas dans l'erreur commise

exemple 0 : la fonction exponentielle a été définie en terminale comme la fonction qui vaut 1 en 0 et qui est sa propre dérivée donc : $\begin{cases} y(0) &= 1 \\ \dfrac{dy}{dt} &= y \end{cases}$ On a donc ici $F(t,y) = y$.

Construisons sa courbe sur $[0;1]$

```
import matplotlib.pyplot as pl
import numpy as np

def Eulerexponentielle(b,n):
    dt = ...
    t = ...
    y = ...
    T = [t]
    Y=[y]
    for k in range(n):
        y = y + dt * ...
        Y.append(y)
        t = t + dt
        T.append(t)
    return T,Y

T,Y =  Eulerexponentielle(1,2)
pl.plot(T,Y,color='red')
T,Y =  Eulerexponentielle(1,10)
pl.plot(T,Y)
#pl.plot(T,np.exp(T))
pl.grid()  #cree un quadrillage
pl.axhline(color='black') #affichage de l'axe des x
pl.axvline(color='black') #affichage de l'axe des y
pl.xlim(-0.2,1.2)
pl.show()
```



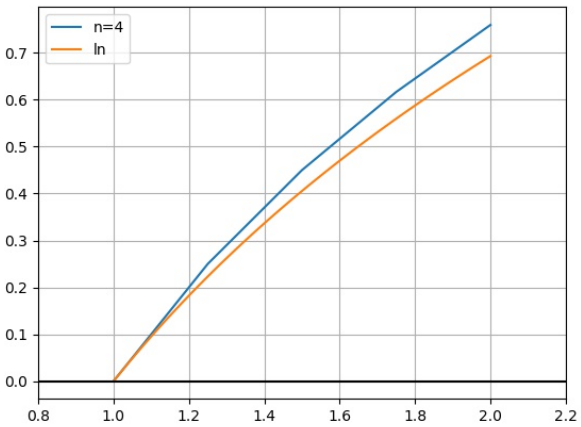


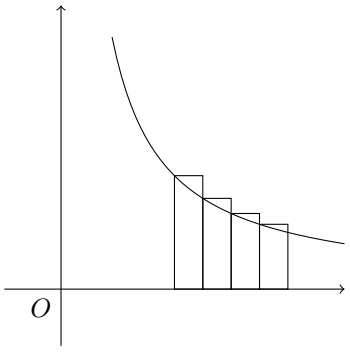
D'un point de vue théorique, plus le pas est petit, meilleure est l'approximation. Néanmoins cela augmente le nombre donc le temps de calcul. De plus, cela augmente aussi les erreurs de calcul de la machine sur les flottants. Il faut donc trouver en pratique le bon compromis.

Remarque : Dans le cas où on construit la courbe sur $[0;1]$, On a donc $dt = \dfrac{1}{n}$.
L'approximation pour $e = exp(1)$ est donc $(1+\frac{1}{n})^n$.
La véritable erreur est donc $E(\frac{1}{n}) = e - (1+\frac{1}{n})^n = \dfrac{e}{2n} + o(\frac{1}{n})$.
Vous pouvez le démontrer en utilisant les développements limités...

exemple 0 bis : la fonction logarithme népérien a été définie en terminale comme la fonction qui vaut 0 en 1 et dont la dérivée est $x \to \frac{1}{x}$ donc : $\begin{cases} y(...) &= ... \\ \dfrac{dy}{dt} &= ... \end{cases}$ On a donc ici $F(t,y) = ...$

```python
import matplotlib.pyplot as pl
import numpy as np

def Eulerlogarithme(b,n):
    dt = ...
    t = ...
    y = ...
    T = [t]
    Y=[y]
    for k in range(n):
        y = y + dt * ...
        Y.append(y)
        t = t + dt
        T.append(t)
    return T,Y

T,Y =  Eulerlogarithme(2,5) #calcul d'une solution sur [1;2]
pl.plot(T,Y )
pl.plot(T,np.log(T))
pl.grid()  #cree un quadrillage

pl.axhline(color='black') #affichage de l'axe des x
pl.axvline(color='black') #affichage de l'axe des y
pl.xlim(-1, 2.2)
pl.show()
```





La méthode d'Euler correspond à la méthode dite « des rectangles »d'approximation d'une intégrale.

Quantification de l'erreur : On appelle note : $e_k = y(t_k) - y_k$ l'erreur entre la vraie valeur en t_k et l'estimation. L'erreur totale pour un pas de dt est donc $E = \sum_{k=1}^{n} |e_k|$ (la somme de toutes les erreurs cumulées).

Or si y est de classe $\mathscr{C}^2$, un utilisant la formule de Taylor, on peut justifier que la somme des erreurs cumulées est en $O(dt)$: La méthode d'Euler est donc une méthode dite d'ordre 1.

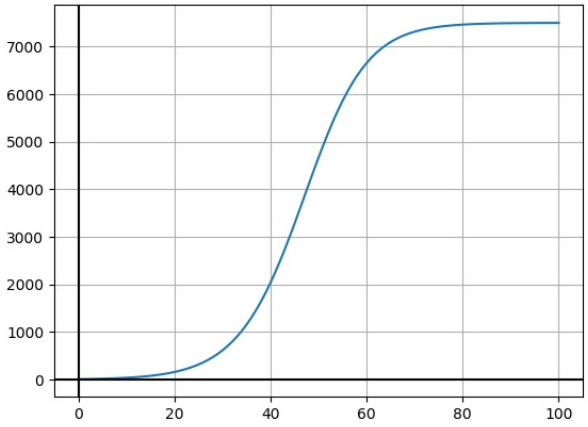
exemple 1 : Le modèle de Verhulst de modélisation de la dynamique des populations. y désigne la taille de la population à l'instant t. Cette modélisation s'applique par exemple aux évolutions auto-catalytiques, comme les cellules tumorales dont l'accroissement est proportionnel au nombres de cellules touchées et au nombre de cellules saines.

Elle est solution de l'équation différentielle : $\frac{dy}{dt} = y \times (n(y) - m(y))$ où $n(y)$ désigne le taux de natalité décroissant en y et $m(y)$ désigne le taux de mortalité, croissant en y (effet de surpopulation). En supposant que ce sont des fonctions affines, on obtient :
$\frac{dy}{dt} = y \times ((a - by))$ a et b étant des réels positifs.
$\frac{dy}{dt} = ay \times (1 - \frac{y}{K})$ avec $K = \frac{a}{b}$ capacité d'accueil (car si $y > K$ alors y est décroissante et si $y < K$ alors y est croissante).

Prenons l'exemple du Parc Kruger : L'éléphant africain de la savane se comptait par millions dans la savane africaine avant qu'il ne soit décimé, durant des siècles, par des chasseurs. En 1905, il ne restait que 10 individus en Afrique du Sud, et il fut décidé la création d'un parc naturel : le parc Kruger. Au $20^{\text{ième}}$ siècle, grâce à des mesures de protections strictes des animaux et de leur milieu, la population d'éléphants a cru de manière naturelle : lentement jusque dans les années 30 (on en comptait alors 403), puis rapidement jusqu'aux années 60. C'est alors qu'on observa un ralentissement du taux de croissance et, en même temps, un début de dégradation par les éléphants d'autres espèces de l'écosystème (comme les baobabs par exemple). On dénombrait alors 7500 éléphants.

Le modèle de Verhulst modélisant cette évolution est : $y_0 = 10$ et $\frac{dy}{dt} = 0{,}15y \times \left(1 - \frac{y}{7500}\right)$

```python
import matplotlib.pyplot as pl
import numpy as np

def Eulerelephants(b,n):
    dt = b/n
    t = 0
    y = 10
    T = [t]
    Y=[y]
    for k in range(n):
        y = y + dt * 0.15*y*(1-y/7500)
        Y.append(y)
        t = t + dt
        T.append(t)
    return T,Y

T,Y =  Eulerelephants(100,100)
pl.plot(T,Y)
pl.grid()  #cree un quadrillage
pl.axhline(color='black') #affichage de l'axe des x
pl.axvline(color='black') #affichage de l'axe des y
pl.show()
```

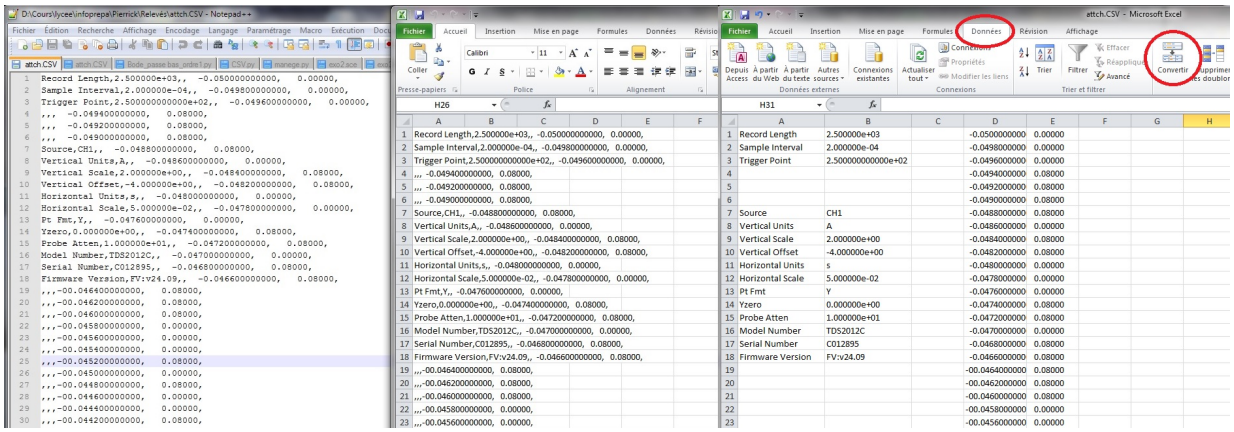


TD méthode d'Euler

Exo 1 : (intégration numérique)

L'attacheur de végétation de la société Pellenc est un appareil très utilisé dans la viticulture. Il est alimenté par une batterie de 3 000 mAh qui doit permettre une autonomie de 10 000 attachages par jour. Une mesure de la consommation du courant lors d'un attachage a été réalisée à l'oscilloscope.
Afin de vérifier l'autonomie réelle de l'attacheur, on veut estimer la quantité d'électricité pour une attache. Cela est représenté par la surface sous la courbe. Il est impossible de calculer cette surface directement. On a donc enregistré les points de la courbe au format csv dans le fichier attch.csv.
Nous allons donc exploiter le fichier qui contient tous les points de la courbe en appliquant la méthode des rectangles ou trapèzes pour déterminer cette surface.

Étape 1 : Récupérer le fichier attch.csv contenant les données
Dans un premier temps, visualisons le contenu du fichier CSV.
Vous pouvez ouvrir le fichier avec un éditeur de texte simple comme notepad, wordpad ...
Ou vous pouvez l'ouvrir avec un tableur comme Excel, OpenOfficeCalc ... Il faut alors parfois spécifier que le séparateur de données est la virgule. On constate que seules les valeurs des colonnes 3 et 4 nous intéressent...



Vue sous Notepad++, vue sous Excel avant et après spécification du séparateur

Étape 2 : Récupérer le fichier code_eleve_exercice1.py et compléter le afin de calculer par la méthode des rectangles puis la méthode des trapèzes l'aire sous la courbe. Attention de modifier le chemin d'accès a votre fichier attch.csv !

```
import matplotlib.pyplot as pl

#recuperer les donnees
f=open('D:/fichierspython/attch.CSV','r')
releve=f.readlines()  #releve est un tableau dont chaque ligne correspond a une ligne du releve
f.close()
X=[]
Y=[]
for i in range(0,2500):
    ligne=releve[i].split(',')  #permet de repartir dans 5 colonnes les 5 donnees de la ligne
    X.append(float(ligne[3]))
    Y.append(float(ligne[4]))

pl.plot(X,Y)
pl.show()

#calcul de l aire sous la courbe par la methode des rectangles

#calcul de l aire sous la courbe par la methode des trapezes
```

Étape 3 : Conclure ! Cette batterie permet elle bien une autonomie de 10 000 attachages ?

Exo 2 : chute libre avec frottements proportionnels au carré de la vitesse

Nous supposerons, qu'exaspérés, vous décidez de jeter un de vos professeur par la fenêtre afin d'en étudier la chute libre.
On émet l'hypothèse raisonnable que ce corps est soumis à son poids et à une force de frottement (résistance de l'air) proportionnelle au carré de la vitesse $\overrightarrow{F} = -k||\overrightarrow{v}||\overrightarrow{v}$.

On considérera l'axe des z dirigé vers le haut.
On distinguera une phase ascendante (vous étiez particulièrement énervé, cela a décuplé votre force, vous avez réussi lancer votre prof vers le haut...) puis une phase descendante au mouvement.
On notera $v(t)$ la vitesse algébrique.

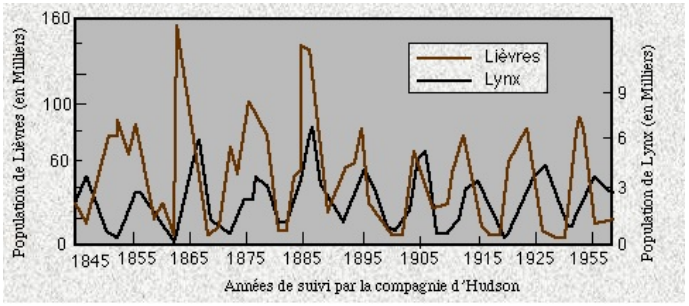
1. A l'aide du principe fondamental de la dynamique (appliqué deux fois selon si vous êtes en phase ascendante ou descendante), montrer que la vitesse algébrique vérifie

$$\frac{dv}{dt} = f(v) \text{ avec } f(v) = \begin{cases} -g - \dfrac{k}{m} * v^2 \text{ si } v \geqslant 0 \\ -g + \dfrac{k}{m} * v^2 \text{ si } v < 0 \end{cases}$$

   La fonction f n'étant **pas linéaire**, la méthode du cours de mathématiques de résolution d'équations différentielles linéaires ne peut s'appliquer. Nous allons donc résoudre numériquement cette équation avec la méthode d'Euler.
2. Afin de préciser la valeur des paramètres, vous observez que le corps d'une masse de 80 kg, atteint une vitesse limite de 69.5 mètres par secondes. Vous savez que $g = 9.8m.s^{-2}$. justifier que $k = 0{,}16kg.m^{-1}$.
3. En Python, écrire une fonction $f(v)$ qui renvoie l'expression de la fonction f du problème considéré (v pouvant être positive ou négative)
4. En Python, écrire une fonction $Euler(f,v0,T,N)$ qui prend en argument la fonction f, la vitesse initiale $v0$, la durée de l'expérience T (nombre entier de seconde) et le nombre N de points souhaités. Elle renvoie les N valeurs de t et les N valeurs de v obtenues grâce à la méthode d'Euler (Vous pouvez inclure les valeurs initiales).
5. Représenter à l'aide de la méthode d'Euler la courbe de la vitesse pendant les 10 premières secondes. On prendra pour vitesse initiale zéro, puis $4m.s^{-1}$.
6. Améliorer votre fonction $Euler$ en une fonction $Euler(f,v0,z0,T,N)$ qui prend en plus en argument la position initiale $z0$ et renvoie en plus les N valeurs de z.
7. Tracer sur le même graphe les fonctions $v(t)$ et $z(t)$ pour vitesse initiale $4m.s^{-1}$ et une position initiale $z0 = 0$ (on pourra se limiter a $T = 2s$).

II Généralisation à des systèmes différentiels de degré 1 (s'il ne reste que peu de temps, passez directement au III)

Exo 3 : Citons Wikipedia :
En mathématiques, les équations de Lotka-Volterra, que l'on désigne aussi sous le terme de « modèle proie-prédateur », sont un couple d'équations différentielles non linéaires du premier ordre, et sont couramment utilisées pour décrire la dynamique de systèmes biologiques dans lesquels un prédateur et sa proie interagissent. Elles ont été proposées indépendamment par Alfred James Lotka en 1925 et Vito Volterra en 1926. Ce système d'équations est classiquement utilisé comme modèle pour la dynamique du lynx et du lièvre des neiges, pour laquelle de nombreuses données de terrain ont été collectées sur les populations des deux espèces par la Compagnie de la baie d'Hudson au XIXe siècle.



En notant $x(t)$ et $y(t)$ le nombre respectif de lynx et de lièvres, on peut modéliser la situation de la manière suivante :

$$\begin{cases} \dfrac{x'(t)}{x(t)} &= -M + Ny(t) \\ \dfrac{y'(t)}{y(t)} &= N' - M'x(t) \end{cases} \text{ donc } \begin{cases} x'(t) &= -x(t)\,(M - Ny(t)) \\ y'(t) &= y(t)\,(N' - M'x(t)) \end{cases}$$

M est le taux de mortalité des lynx (uniquement du à la vieillesse) alors que leur taux de natalité est proportionnel au nombre de lièvres (source de nourriture).
N' est le taux de natalité des lièvres alors que leur taux de mortalité est proportionnel au nombre de lynx.

L'observation a permis sur le terrain a permis d'obtenir : M = 0,3 et N = 0,002 pour le lynx
M' = 0,01 et N' = 0,5 pour le lièvre

$$\text{donc } \begin{cases} x'(t) &= -x(t)\,(0{,}3 - 0{,}002y(t)) \\ y'(t) &= y(t)\,(0{,}5 - 0{,}01x(t)) \end{cases}$$

1. Représenter graphiquement l'évolution de ces populations sur 100 ans avec comme populations initiales 50 lynx et 200 lièvres.
2. Si la chasse aux lynx était ré-autorisée et que leur mortalité passait à M = 0,6 , comment cela impacterait-il la dynamique des populations ? et si le réchauffement climatique faisait augmenter le taux de natalité des lièvres à N' = 0,8 ? ou au contraire si leur mortalité (à cause de la réintroduction de loups) passait à M' = 0,025 ?

III Généralisation à des équations différentielles de degré supérieur

Exo 4 : L'étude du pendule simple non amorti conduit à une équation différentielle d'ordre 2 de la forme $\dfrac{d^2\theta}{dt^2} + \omega^2 \sin\theta = 0$
On posera ici $\omega = 1$ et on supposera que $\theta(0) = 0$ et $\theta'(0) = 1$

1. En posant $x(t) = \theta(t)$ et $y(t) = \theta'(t)$, montrer que l'équation peut s'écrire sous la forme d'un système $\begin{cases} x'(t) &= y(t) \\ y'(t) &= -\sin(x(t)) \end{cases}$
2. Résoudre numériquement ce problème sur $[0, 6\pi]$.
3. Tracer l'évolution de $\theta(t)$ au cours du temps.
4. Tracer le portrait de phase, c'est-à-dire l'ensemble des points de coordonnées $(\theta(t), \theta'(t))$ pour $t \in [0, 4\pi]$.

remarque : En fait, le module scipy de Python propose la fonction odeint() qui vous permet de résoudre numériquement des équations différentielles. La syntaxe est

```
from scipy.integrate import odeint
v=odeint(f,v0,T)
```

où f est une fonction (éventuellement vectorielle) dépendant de v et t
où $v0$ est la condition initiale (éventuellement vectorielle)
où T est un tableau des valeurs de t considérées.

En fait, nous aurions pu écrire en début de TD une fonction *Eulerexplicitevectorielle*(F,a,b,$X0$,n) prenant en paramètres une fonction vectorielle F et un tableau de conditions initiales $X0$ qu'on se serait contenté d'appeler, y compris dans les parties II et III, avec les fonctions F adaptées.
Vous pouvez donc refaire le TP, mais ça sera cette fois plus rapide... (pensez à importer numpy !)
Maintenant, vous avez vraiment envie de jeter un prof par la fenêtre ...