Methode-Euler

1 Rappel du principe

On souhaite résoudre une équation différentielle de la forme

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y,t)$$

munit de la condition itiniale $y(t_0) = \alpha$.

Par exemple, l'équation $\frac{\mathrm{d}y}{\mathrm{d}t} - y(t) = \cos(t)$ correspond à la fonction $f(y,t) = y + \cos(t)$.

1.1 Interprétation physique

On peut réécrire l'équation différentielle comme $\frac{y(t+\mathrm{d}t)-y(t)}{\mathrm{d}t} = f(y,t)$ et on obtient donc $y(t+\mathrm{d}t) = y(t) + \mathrm{d}t f(y,t)$.

Informatiquement, on fixe $\mathrm{d}t$ "petit" et à partir de $y(t_0)$, on calcule $y(t_0+\mathrm{d}t), y(t_0+2\mathrm{d}t), \ldots$ jusqu'à arriver à un temps final fixé à l'avance.

1.2 Interprétation graphique

On connaît déjà $y(t_0)$ ainsi que $\beta = \frac{\mathrm{d}y}{\mathrm{d}t}(t_0) = f(y(t_0), t_0)$ (la valeur de la dérivée au temps initial, grâce à l'équation différentielle). Ainsi on peut déjà tracer la tangente au points initial. Celle ci est d'équation (on représente y en fonction du temps)

$$y = \alpha + \beta(t - t_0)$$

(penser à Taylor ou à vos cours de 1ère : en a, la tangente est d'équation $y = f(a) + f'(a)(x-a)$)

L'approximation d'Euler consiste alors à confondre la courbe avec sa tangente entre t_0 et $t_0 + \mathrm{d}t$.

On obtient alors $y(t_0 + \mathrm{d}t) = y(t_0) + f(y(t_0), t_0) \times \mathrm{d}t$. La même relation

2 Suites récurrentes

3 Obtenir une valeur

On considère la suite définie par $u_0 = 1$ et $\forall n \in \mathbb{N}\ u_{n+1} = 1 + \frac{1}{u_n}$. On souhaite calculer u_{20}.

Compléter le code python suivant :

```
u = 1  # initialisation. u contient la valeur u0
for i in range(   ): # combien d'itération ?
    u =
# maintenant la variable u contient la valeur de u20
```

3.1 Obtenir une suite de valeurs

Cette fois, on veut obtenir toutes les valeurs $u_0, \ldots, u_{30}$ pour la suite définie par $u_0 = 1$ et $\forall n \in \mathbb{N}$ $u_{n+1} = \cos(u_n)$.

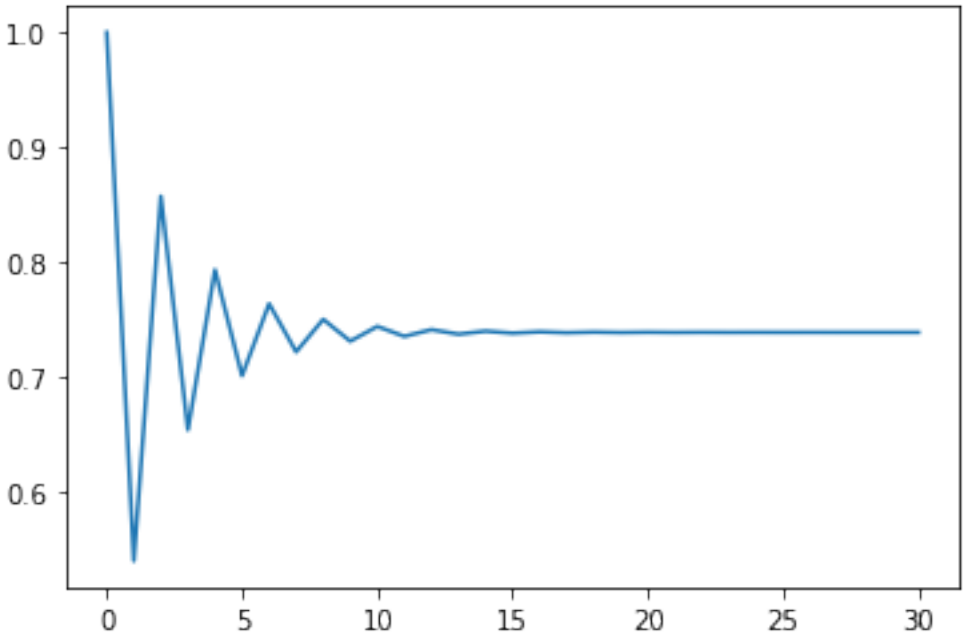
Construire, en python, une liste L contenant les éléments souhaités

```
import math
```

```
import matplotlib.pyplot as plt
```

```
plt.plot(range(31), L)  # range(31), les indices des u_i stockés dans L
```

```
[<matplotlib.lines.Line2D at 0x7f2cfa206400>]
```



4 Un premier exemple

On cherche à étudier la vitesse d'une masse en chute libre verticale, suibissant une force de frottement proportionelle à v^2.

L'équation différentielle correspondante est

$$\frac{\mathrm{d}v}{\mathrm{d}t} = g - \frac{k}{m}v^2$$

On choisit une vitesse initiale nulle au temps $t = 0$, une masse de 1kg et $g = 9.81$

On prendra $k = 2.10^{-3}$.

On peut remarquer que notre équation différentielle ne fait pas apparaître le temps t dans son expression.

4.1 Établir la relation de récurrence

On fixe $\mathrm{d}t$, le pas de discrétisation temporel. Donner l'expression de $v(t + \mathrm{d}t)$ en fonction de $v(t)$ et des paramètres de l'équation, en utilisant la méthode d'Euler.

4.2 Code !

Compléter la fonction suivante, qui retourne la liste des vitesses entre les temps 0 et *tf*, en utilisant un pas de temps égal à 10^{-4}, ainsi que la liste des temps correspondant.

La première liste retournée sera la liste des temps et la seconde la liste des vitesses. Pour retourner deux valeurs depuis une fonction, il suffit de placer une virgule entre les 2

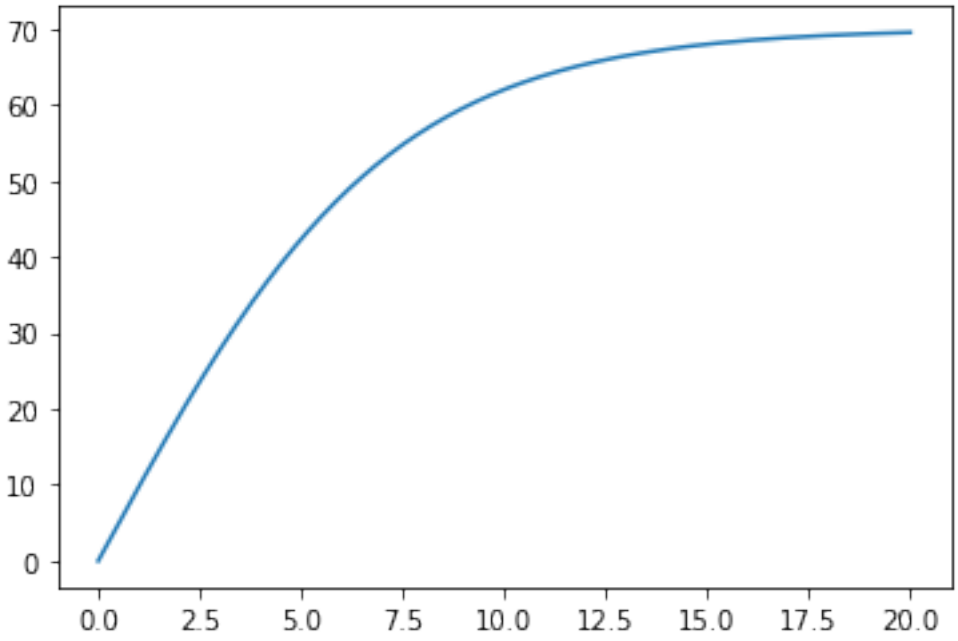
```
return valeur1, valeur2
```

```
g = 9.81
k = 2e-3
m = 1
# on peut utiliser ces valeurs dans la fonction suivante
```

```
def liste_vitesses(tf):
    v = 0 # la vitesse courante
    t = 0 # le temps courant
    # compléter la ligne suivante
    LV = [ v ] # liste des vitesses, à construire
    # compléter la ligne suivante
    LT = [ t ] # liste des temps, à construire
    dt = 1e-4
    # à vous de jouer !
```

```
T, V = liste_vitesses(20)
plt.plot(T, V)
```

```
[<matplotlib.lines.Line2D at 0x7f2cf9c4d1d0>]
```



4.3 Vérification

On obtient, graphiquement, une vitesse limite approximativement égale à 70.

Préciser l'unité, et vérifier que ce résultat est cohérent avec l'équation différentielle.

5 Equation vectorielle

5.1 Rappels sur numpy

La bibliothèque numpy permet d'effectuer facilement des opérations sur les "array" (que nous nommerons vecteurs) : les opérations mathématiques sont exécutées sur tous les éléments. Voici queslques exemples.

```
import numpy as np
```

```
v1 = np.array([1,2,3])
v2 = np.array([-1,0,4])
v1 + 2*v2
```

```
array([-1,  2, 11])
```

```
v1 * v2
```

```
array([-1,  0, 12])
```

```
v1**2
```

```
array([1, 4, 9])
```

```
np.log(v1)
```

```
array([0.        , 0.69314718, 1.09861229])
```

5.2 Application à la méthode d'Euler

La méthode d'Euler s'applique également quand l'inconnue est une fonction à valeurs vectorielles (par exemple pour chercher le vecteur vitesse si la chute n'est pas verticale), en utilisant la même approximation.

Cette fois, la quantité $\mathrm{d}t \times \frac{\mathrm{d}y}{\mathrm{d}t}$ est le produit d'un vecteur et d'un scalaire.

Numpy est donc parfaitement adapté à ces calculs, à condition d'utiliser les vecteurs numpy pour la condition initiale ainsi que dans les calculs.

5.2.1 Description de la situation

On considère 2 masses ponctuelles dans le plan et on souhaite étudier l'évolution de leurs positions au cours du temps en supposant qu'elles ne sont soumises qu'à l'attraction gravitationnelle. Notons m_1, m_2 leurs masses respectives et $P_1(t), P_2(t)$ leurs positions à un instant t. Alors l'action exercé par le corps 1 sur le deuxième est

$$\vec{F_{1,2}} = \frac{Gm_1m_2}{r^3}\overrightarrow{P_2(t)P_1(t)}$$

où r est la distance déparant les deux corps, c'est à dire $\|\overrightarrow{P_2(t)P_1(t)}\|$ (d'où le cube dans la formule...)

Évidemment, on a $\vec{F_{2,1}} = -\vec{F_{1,2}}$, c'est Newton qui le dit !

Informatiquement, P_1 et P_2 seront représentés par des vecteurs de taille 2. On prendra, pour les applications numériques, $G = 1$.

Compléter la fonction suivante qui retourne la force exercée par le corps 1 sur le corps 2

```
G = 1
def force_12(m1, P1, m2, P2):
    """
    Retourne le vecteur force représentant l'attraction gravitaionnelle
    exercée par le corps 1 de masse m1 et en position P1 sur le corps 2
    """
```

5.3 Équation à résoudre

Nous devons résoudre, pour $\{i, j\} = \{1, 2\}$, l'équation $\frac{\mathrm{d}^2\overrightarrow{OP_i}}{\mathrm{d}t^2} = \frac{\vec{F_{j,i}}}{m_i}$. L'équation à résoudre est d'ordre 2.

Pour se faire, on doit introduire une autre inconnue, la vitesse $\vec{v_i}$. On a alors

$$\begin{cases} \frac{\mathrm{d}\overrightarrow{OP_i}}{\mathrm{d}t} = \vec{v_i} \\ \frac{\mathrm{d}\vec{v_i}}{\mathrm{d}t} = \frac{\vec{F_{j,i}}}{m_i} \end{cases}$$

Nous allons donc appliquer deux fois la méthode d'Euler :

$$\begin{cases} \overrightarrow{OP_i}(t + \mathrm{d}t) = \overrightarrow{OP_i}(t) + dt\vec{v_i}(t) \\ \vec{v_i}(t + \mathrm{d}t) = \vec{v_i}(t) + \frac{\mathrm{d}t}{m_i}\vec{F_{j,i}}(t) \end{cases}$$

Compléter la fonction suivante :

```
    def etats_suivants(m1, P1, V1, m2, P2, V2, dt):
        """
        V1 et V2 sont les vitesses à l'instant t des corps 1 et 2

        Retourne 4 valeurs : position, vitesse à l'instant t + dt pour les␣
↪corps 1 et 2
        """




        nv_pos1 =
        nv_vit1 =
        nv_pos2 =
        nv_vit2 =
        return nv_pos1, nv_vit1, nv_pos2, nv_vit2
```

On considère les conditions initiales suivantes :

```
    V1 = np.array([0, 0])
    V2 = np.array([0, 5])
    m1 = 100
    m2 = 1
    P1 = np.array([0, 0])
    P2 = np.array([2, 0])
```

```
    etats_suivants(m1, P1, V1, m2, P2, V2, 1e-4)
```

```
(array([0., 0.]),
 array([0.0001, 0.    ]),
 array([1.e+00, 1.e-08]),
 array([-0.01  ,  0.0001]))
```

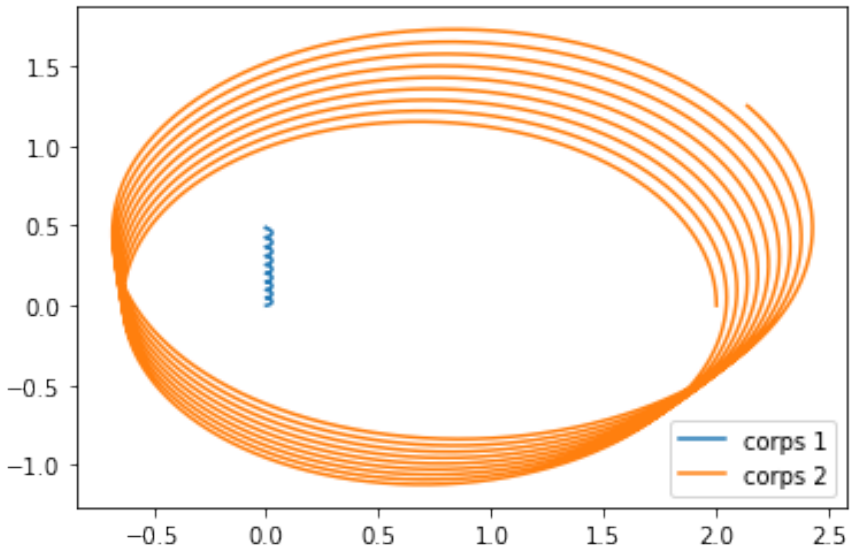
Calculer la listes des positions et des vitesses sucessives entre les temps 0 et 10, avec un pas de 10^{-4}.

```
V1 = np.array([0, 0])
V2 = np.array([0, 5])
m1 = 100
m2 = 1
P1 = np.array([0, 0])
P2 = np.array([2, 0])
positions1 = [P1]
positions2 = [P2]
vitesses1 = [V1]
vitesses2 = [V2]
t = 0
tf = 10
dt = 1e-4
```

Tracer maintenant les trajectoires des 2 corps

```
X1 = [p1[0] for p1 in positions1]  # on extrait les abcsises
Y1 = [p1[1] for p1 in positions1] # idem avec les odronnées
X2 = [p1[0] for p1 in positions2]
Y2 = [p1[1] for p1 in positions2]
plt.plot(X1, Y1, label="corps 1")
plt.plot(X2, Y2, label="corps 2")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f2cf6b04ba8>
```



6 Pour aller plus loin : une autre méthode d'intégration.

6.1 Formalisme

On cherche à résoudre deux équations de la forme $\frac{\mathrm{d}^2\overrightarrow{OM}}{\mathrm{d}t^2} = \frac{\vec{F}}{m}$.

Notons $M_0, \ldots, M_n$ les positions sucessives du points M (se seront des vecteurs numpy) et $v_0, \ldots, v_n$ les vecteurs vitesses correspondants et $F_0 \ldots, F_n$ les forces subies par le point M. La méthode d'Euler consiste à écrire :

$$\begin{cases} M_{k+1} = M_k + v_k \mathrm{d}t \\ v_{k+1} = v_k + \frac{F_k}{m}\mathrm{d}t \end{cases}$$

Ces deux formules sont identiques dans le sens où on applique le même principe d'approximation à M et v.

6.2 Méthode de Verlet

On va utiliser une méthode plus précise, traduite par les retations de récurrence suivantes :

$$\begin{cases} M_{k+1} = M_k + v_k \mathrm{d}t + \frac{F_k}{m}\frac{\mathrm{d}t^2}{2} \\ v_{k+1} = v_k + \left(\frac{F_k + F_{k+1}}{2m}\right)\mathrm{d}t \end{cases}$$

On utilise cette fois une approximation à l'ordre 2 pour M (remarquer les termes de Taylor, où la dérivée seconde est donnée par l'équation différentielle), et on prend la force moyenne $\frac{F_k+F_{k+1}}{2}$ à la place de la force à l'instant k.

Le problème technique vient du fait qu'il faut connaître F_{k+1} avant de pouvoir calculer v_{k+1} et on doit donc calculer la position en premier.

```
    def etat_suivants_verlet(m1, P1, V1, m2, P2, V2, dt):









        return nv_pos1, nv_vit1, nv_pos2, nv_vit2
```

6.3 Tracé !

Reprendre les étapes précédentes pour obtenir le tracé !

7 Terre et lune !

On voudrait obtenir les trajectoires de la Terre et de la Lune ! Chercher les valeurs numériques correspondantes

```
    G =          # la "vraie" valeur
    Mt =         # masse de la terre, en kg
    Ml =         # masse de la lune
    P1 = np.array([0, 0]) # plaçons la terre au centre
    distance_terre_lune =      # en m
    P2 = np.array([0, distance_terre_lune])
    V1 = np.array([0, 0])  # on se place dans le repère géocentrique
    vitesse_instantanee_lune =        # en m/s
    V2 = np.array([vitesse_instantanee_lune, 0])  # on approxime dans un␣
↪premier temps
    #la trajectoire par un cercle, la vitesse est tangente à la trajectoire,␣
↪donc perpendiculaire au rayon.
    # On fait en sorte de regarder le plan contenant la trajectoire de la␣
↪lune du côté où elle semble tourner
    # dans le sens trigonométrique !
    t = 0
    tf = 3 * 28 * 24 * 3600 # 3 mois
    dt = tf / (10**5)  # on effectue 10**5 calculs
    positions1 = [P1]
    positions2 = [P2]
    vitesses1 = [V1]
    vitesses2 = [V2]
```