TP 4

Introduction

Voici quelques commandes à taper dans l'interpréteur pour prendre en main le module numpy. Cherchez à comprendre le résultat renvoyé.

Création d'un tableau

```
>>> import numpy as np
>>> a=np.eye(5,5)
>>> print(a)
>>> x=np.arange(0,10,1)
>>> x
>>> y=np.arange(1,8,2)
>>> print(y)
>>> z=np.array([0, 1, 2, 5])
>>> print(z)
>>> v=np.array([[1,2],[2,3],[3,4]])
>>> print(v)
>>> type(v)
```

Opérations sur les vecteurs

```
>>> import numpy as np
>>> x=np.arange(0,11,1)
>>> y=2*x
>>> np.size(x)
>>> np.shape(x)
>>> x[0]
>>> pow(x,2)
>>> x[1:]
>>> x[2:5]
>>> x[2:2:6]
>>> x=x[2:-1]
>>> y=y[1:-2]
>>> x+y
>>> z=x<5
>>> z.astype(float)
```

Elements non nuls d'un vecteur

```
nz = np.nonzero([1,2,0,0,4,0])
print(nz)
```

Opérations logiques

```
>>> x=np.arange(0,11,1)
>>> z=x<5
>>> z.astype(float)
```

```
>>> z2=x>3
>>> z3=x<6
>>> z2 and z3
```

Le dernier cas marche-t-il ?

Exercice 1 :

Que font les commandes :

$$\texttt{np.ones}((3,1)), \quad \texttt{np.ones}((1,3)), \quad \texttt{np.ones}(3), \quad \texttt{np.eye}(3), \quad \texttt{np.eye}(3,2)?$$

A partir de maintenant vous allez écrire vos programmes dans des fichiers exécutables par Python. On aura préalablement importé `numpy` sous le nom `np`.

Exercice 2 :

Définir de la façon la plus simple possible les vecteurs

$$v1 = (1, 2, 3, 4, \ldots, 15, 16)$$
$$v2 = (28, 26, 24, 22, 20, 18, 16, 14, 12)$$
$$v3 = (1, 2, 4, 8, 16, \ldots, 128, 256)$$

et les tableaux

$$A = \begin{bmatrix} 1 & 2 & 4 & \ldots & 2^8 \\ 1 & 3 & 9 & \ldots & 3^8 \\ 1 & 5 & 25 & \ldots & 5^8 \end{bmatrix}$$

Exercice 3 :

Écrire un programme qui prend en entrée deux entiers m (lignes) et n (colonnes) et qui crée une matrice bidimensionnelle A (prenez $m = 3$ et $n = 4$). L'élément a_{ij} de la ligne i et de la colonne j doit être égal à $i \cdot j$. Afficher cette matrice. Extraire de la matrice A les deux vecteurs v et w correspondant à la première et la dernière colonne. Calculer et afficher le produit scalaire entre ces deux vecteurs.

Exercice 4 : *Modélisation de la chute libre avec la méthode d'Euler*

On va s'intéresser à la chute libre d'une balle de tennis, sans vitesse initiale et sans prendre en compte les frottements de l'air, jusqu'au moment où elle touche le sol. Dans un premier temps, nous allons voir comment modéliser la trajectoire et l'évolution de la vitesse de la balle à l'aide de la méthode d'Euler. Nous verrons, ensuite, l'importance du choix du pas de temps et nous nous servirons des listes pour créer les vecteurs vitesses et positions.

Nous lâchons une balle d'une hauteur donnée. Une fois que la balle a quitté la main, la seule force externe qui s'appliquera sur elle est la force de gravité, $\vec{P}$ (nous négligeons les frottements). On peut alors écrire le principe fondamental de la dynamique :

$$m \cdot \vec{a} = \sum \vec{F_{ext}} = \vec{P}$$

En projettant sur l'axe vertical y, on obtient :

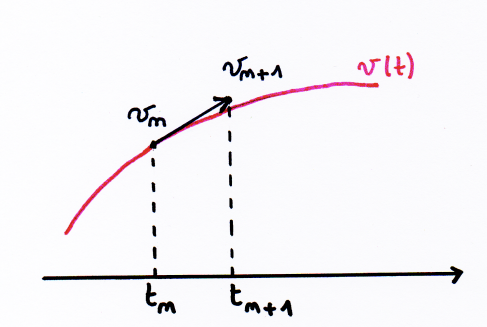
$$m \cdot \frac{dv}{dt} = -m \cdot g$$

$$\frac{dv}{dt} = -g$$

On remarque que la masse de l'objet n'intervient pas dans la vitesse de chute. Nous allons maintenant calculer à chaque instant la position et la vitesse de la balle à l'aide d'une méthode numérique de résolution de l'équation différentielle basée sur le schéma d'Euler, décrit ci-dessous. À partir de la connaissance d'une valeur de la vitesse à un instant (v_n à l'instant t_n), on calcule la valeur suivante (v_{n+1} à l'instant t_{n+1}) à l'aide de l'approximation de la dérivée :

$$\frac{dv}{dt} = \lim_{(t_{n+1}-t_n)\to 0} \frac{v(t_{n+1}) - v(t_n)}{t_{n+1} - t_n}$$

Cela se traduit géométriquement par le calcul du coefficient directeur de la tangente à la courbe, comme montré dans la figure ci-dessous :



Un choix simple est de prendre un intervalle de temps régulier, c'est à dire de fixer la valeur de t_{n+1} - t_n. Mais on pourrait tout-à-fait choisir un pas de temps variable en fonction de l'optimisation et de la précision du calcul pour des moments choisis. On remplace donc la dérivée par la valeur de la pente. Une telle approximation n'est raisonnable que si la variation entre les deux instants est toute petite. Ainsi on peut écrire :

$$\frac{\Delta v}{\Delta t} = -g$$

D'où :

$$\Delta v = -g \cdot \Delta t$$

On a alors :

$$v_{n+1} - v_n = -g \cdot \Delta t$$

Ce qui peut s'écrire :

$$v_{n+1} = v_n - g \cdot \Delta t$$

Finalement, il suffit de répéter l'itération afin de connaitre à chaque instant la valeur de la vitesse le long de la chute :

$$v_1 = v_0 - g \cdot \Delta t$$
$$v_2 = v_1 - g \cdot \Delta t$$
$$v_3 = v_2 - g \cdot \Delta t$$
$$...$$
$$...$$

En utilisant cette approche, écrire un programme qui calcule et affiche grafiquement à chaque instant la position et la vitesse de la balle. Suivre les instructions suivantes :

1. Dans un premier temps, on importe les bibliothèques python nécessaires au traitement des données et à l'affichage des graphiques.
2. On renseigne les valeurs de constante (gravitation, 9.81 N/m^2), paramètre (pas de temps, 0.0001 s) et données (rayon de la balle de tennis, r=0.03 m, hauteur du lâché, y_0=1.5 m, vitesse initiale, $v_{y0} = 0.0$ m/s).
3. Maintenant, nous appliquons la méthode d'Euler pour coder le mouvement. Dans un premier temps, on initialise la position initiale de la balle à la valeur donnée précédemment ($y = y_0$) et sa vitesse initiale ($v_y = v_{y0}$). Nous introduisons la variable i pour compter le nombre d'itérations du calcul le long de la trajectoire. La variable temps, t, va nous permettre de mesurer le temps de chute. Dans cet exemple on s'intéresse à la trajectoire de la balle jusqu'à ce qu'elle touche le sol, on choisit alors d'utiliser une boucle while de la manière suivante : **Tant que le centre d'inertie de la balle est plus haut que son rayon (y > r) (la balle n'a pas touché le sol), on calcule, pour chaque itération, sa vitesse et sa position, on incrémente le compteur i de 1 et le compteur temps d'un pas de temps. On trace un nouveau point y de la trajectoire.**
4. Lorsque l'on sort de la boucle, on ajoute à la représentation graphique : un titre, les grandeurs et unités des axes, et un quadrillage. À l'aide de la fonction **print**, on affiche le temps de chute de la balle. La fonction **round(x,3)** renvoie la valeur de x arrondie à 3 décimales. On affiche le nombre d'itérations du calcul et le pas de temps.
5. L'hypothèse de départ de cette modélisation est de confondre une dérivée avec un Δ. Cela signifie que, pour être au plus juste, il est nécessaire de choisir un pas de temps très petit, de telle façon qu'entre deux pas, il n'y ait qu'une toute petite variation des grandeurs. Pour tester notre méthode de résolution numérique, on propose de tracer sur le même graphe, le résultat analytique du problème :

   $$y(t) = \frac{1}{2}gt^2 + v_{y0}t + y_0$$

   Nous avons besoin de créer deux vecteurs : $temps_{analyt}$ correspondant aux abcsisses et y_{analyt} correspondant aux ordonnées. La création du vecteur est réalisée à l'aide de la fonction **linspace** du paquet **numpy**. Le vecteur correspondant à la trajectoire y_{analyt} est donné par l'équation de la trajectoire.
6. On trace le résultat analytique avec une ligne rouge (-r) assez épaisse (lw=2.5). On ajoute une légende (label) pour chaque courbe, l'affichage est géré par la fonction **plt.legend()**.
7. Observons les effets du choix du pas de temps sur la méthode de résolution. Nous augmentons successivement la valeur du pas de temps à 10 ms ; 50 ms ; 100 ms et 200 ms. Quelles sont les differences entre la courbe numérique et la résolution analytique ?

8. Récrire le code en consignant les valeurs des vitesses et posistion dans un tableau pour chaque pas de temps. Cette opération est réalisée par la création de listes dont on augmente la taille au cours du temps. On trace, ensuite les représentations graphiques de la vitesse et de la position en fonction du temps. Comme on utilise les listes, nous ne sommes plus obligés d'incrémenter le compteur "temps" à chaque pas de temps. Pour optimiser le calcul, il suffit d'utiliser la variable i qui nous renseigne, en sortie de boucle, sur le nombre d'itérations exécutées.