TABLEAUX NUMPY ET TRACÉS DE COURBES AVEC MATPLOTLIB

⚠ Le contenu de ce TP fait office de cours : vous devez savoir manipuler des tableaux `Numpy` et faire des tracés de courbes.

Vous aurez besoin d'utiliser les modules `numpy` et `matplotlib.pyplot`. Commencer avant tout par les importer.

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

Tableaux numpy

Numpy permet de manipuler des tableaux (`array`) au vrai sens informatique : données de type homogène, accès et modification des éléments en temps constant. Le constructeur `np.array` prend en premier argument un objet à convertir en tableau (par exemple un itérable, comme une liste) et (entre autres) en argument optionnel un type (appelé `dtype`, `float64` par défaut) : comme `int` ou `float` ou `complex`, ou bien `np.uint8`, `np.uint16`, `np.uint32` (`uint` = unsigned integer), `np.int8`, `np.int16`, `np.int32`, `np.float16`, `np.float32`, `np.float64`, etc.

1. Tester directement en console (et comprendre les résultats) :

```
help(np.array)
t = np.array([1, 2, 3, 4, 5, 6])
t2 = np.array([1, 2, 3.5])
t3 = np.array([1, 2, 3], float)
t4 = np.array([1, 2, 3], complex)
```

```
t5 = np.array([-12, 10 ** 6], np.int8)
t6 = np.array([-12, 10 ** 6], np.uint8)
t7 = np.array([-12, 10 ** 6], np.float16)
t8 = np.array([-12, 10 ** 6], np.float32)
np.who()
```

Accès aux éléments, slicing

On peut aussi, comme pour les listes, lire et écrire un élément d'un tableau avec `t[i]`, faire du slicing (format `[a:b:pas]` pour $[a, b[$ comme pour les listes), ou obtenir un sous-tableaux ne contenant que les éléments d'indice dans une liste `L` avec `t[L]`.

2. Tester :

```
t[1]
t[1] = 0
```

```
t
t[1:5]
```

```
t[::-1]
t[[0, 3, 4]]
```

Constructeurs de tableaux

Pour créer un tableau d'une dimension `n` donnée, on peut utiliser (à chaque fois, un type peut être ajouté en deuxième argument) :

- `np.zeros(n)`
- `np.ones(n)`
- `np.empty(n)` (le tableau n'est pas vraiment vide !)
- `np.random.rand(n)`
- `np.linspace(a, b, n)` : subdivision régulière de $[a, b]$ avec n points (les bornes sont incluses)
- `np.arange(a, b, pas)` : comme `range` sauf que ça renvoie un tableau de flottant, `a`, `b` et `pas` peuvent être flottants. La borne `b` n'est pas incluse.

3. Les tester toutes.

Fonctions vectorialisées

Un des atouts majeurs dans tableaux `numpy` est de pouvoir appliquer directement des fonctions dites *vectorialisées* dessus, qui vont s'opérer sur chaque élément du tableau, et ce de façon optimisée. Pas besoin de boucle !

Les fonctions mathématiques du module `numpy` sont vectorialisées et donc à privilégier :

`np.sqrt`	`np.sin`	`np.cos`	`np.tan`	`np.arcsin`
`np.arccos`	`np.arctan`	`np.sinh`	`np.cosh`	`np.exp`
`np.log`	`np.log10`	`np.floor`	`np.ceil`	etc.

Les opérations basiques `+, *, /, //, **, %` le sont aussi.

A noter qu'il existe aussi des constantes mathématiques : `np.pi`, `np.e`, etc.

4. Tester

```
t = np.arange(10)
t + t
t ** 2
t % 3
for i in range(10):
    t[i] /= 2
t # Explication ?
t / 2
```

```
t = np.linspace(0, 2 * np.pi, 10)
np.sin(t)
np.sqrt(t)
np.sum(t)
np.product(t[1:])
np.mean(t)
np.min(t)
```

5. Écrire une fonction **sans boucle** `d(n)` renvoyant la distance euclidienne entre deux vecteurs $\vec{x}$ et $\vec{y}$ de dimension n de coordonnées aléatoires dans $[0, 10[$: $d_n = \mathrm{d}(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$.

Remarque : Avec une fonction existante, on peut aussi faire directement `np.linalg.norm(x - y)`.

On peut faire opérer ses propres fonctions sur les éléments d'un tableau, en les vectorialisant grâce à la fonction `np.vectorize`.

6. Tester :

```
def fact(n):
    "renvoie n!"
    f = 1
    for i in range(1, n + 1):
        f *= i
    return f
```

```
t = np.arange(10)
fact(t)
afact = np.vectorize(fact)
afact(t)
afact([1, 2, 3])
```

Tracés de graphes

Les facilités sur la manipulation des tableaux `numpy` permettent de tracer tout aussi facilement des graphes.

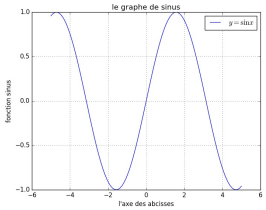
`Matplotlib` offre de nombreuses possibilités pour faire des tracés graphiques. On présente ici un infirme partie des possibilité, il faut prendre l'habitude de consulter l'aide (en ligne, c'est plus confortable) pour personnaliser ses tracés.

La plupart des fonctions possèdent des arguments avec valeur par défaut. On n'a pas besoin de préciser ces arguments si on ne veut pas les changer, et l'ordre dans lequel on les appelle n'importe pas si on les désigne par leur nom : `fonction(arg1=val1, arg2=val2, ...)`.

Quelques commandes de base :

- `plt.figure(titre)` crée un figure (fenêtre) avec le titre précisé. Si on n'utilise pas cette fonction, la figure est créée automatiquement.
- `plt.plot(x, y, label=chaine)`, où `x` et `y` sont des tableaux, calcule la courbe correspondante, `label` permet de préciser une légende (qui peut contenir du LaTeX). Il faut appeler `plt.legend()` pour afficher la légende, l'argument optionnel `loc='best'` permet de laisser python choisir l'emplacement le moins gênant.
- `plt.xlabel(chaine)` et `plt.ylabel(chaine)` permettent de préciser un titre pour chaque axe.
- `plt.title(chaine)` permet de préciser un titre.
- `plt.show()` permet d'afficher le graphe.
- `plt.savefig(nom_fichier)` permet d'enregistrer le graphe dans un fichier image.
- `plt.grid()` permet de voir une grille.

```
x = np.linspace(-5, 5, 100)
plt.plot(x, np.sin(x),
         label='$y=\sin x$')
plt.ylabel('fonction sinus')
plt.xlabel("l'axe des abcisses")
plt.title('le graphe de sinus')
plt.legend(loc='best')
plt.grid()
plt.show()
```



7. Tracer le graphe de la fonction $f : x \mapsto \dfrac{1}{1+25x^2}$ sur $[-1,1]$. Faire apparaître une légende, et une grille. Enregistrer le graphe dans un fichier.

- `plt.axis([xmin, xmax, ymin, ymax])` ou `plt.xlim(xmin, xmax)` ou `plt.ylim(ymin, ymax)` permet de préciser l'échelle des axes.

8. Tracer la fonction $x \mapsto e^{e^x}$ sur $[0,5]$. S'arranger pour qu'on voie quelque chose !

L'aide sur `plt.plot` donne les diverses options possibles pour le tracer.

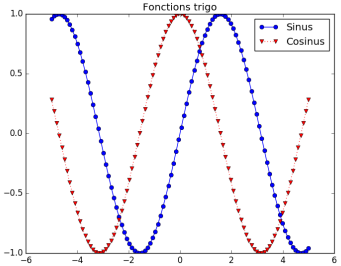
Par exemple, un troisième paramètre à `'r'` donnera une courbe rouge, `'b'` donnera une courbe bleue Avec `color='#24e5cf'` on peut préciser une couleur en RGB (hexadécimal).

On peut marquer les points avec l'option `marker=` : `'o'` pour des ronds, `'+'` pour des plus, `'*'` pour des étoiles, etc.

Le style de ligne peut aussi être précisé avec `linestyle=` ou `ls=` : `'-'` pour continu, `'--'` pour tirets, `':'` pour pointillés, `'-.'`, etc.

Enfin, on peut utiliser un argument condensé : par exemple `'bo'` ou `'g-v'` ou `'--'`, etc.

```
x = np.linspace(-5, 5, 100)
plt.plot(x, np.sin(x), marker='o',
         label='Sinus')
plt.plot(x, np.cos(x), 'rv:',
         label='Cosinus')
plt.title("Fonctions trigo")
plt.legend(loc='best')
plt.show()
```

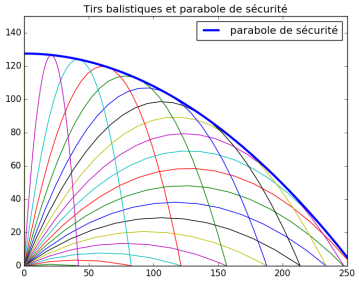


9. Tirs balistiques : on jette un projectile M à partir du point O (origine) avec une vitesse v_0 constante, mais faisant un angle variable θ avec l'horizontale. Si le projectile est soumis seulement à la gravitation, la trajectoire est connue : ce sera une parabole. On démontre même de façon classique que l'ensemble des trajectoires est « enveloppée » par une parabole, dite de sécurité.

Le principe fondamental de la dynamique donne, avec une masse $m = 1$, $x'' = 0$ et $y'' = -g$ soit $x(t) = (v_0 \cos\theta)t$ et $y(t) = (v_0 \sin\theta)t - \frac{g}{2}t^2$.

Superposer sur un même graphe les trajectoires pour une vingtaine d'angles entre 0 et $\frac{\pi}{2}$ (Ne garder que $y \geqslant 0$!), avec $g = 9{,}8$, $v_0 = 50$ (USI).

Tracer également sur le même graphe la parabole de sécurité d'équation $y = \dfrac{v_0^2}{2g} - \dfrac{g}{2v_0^2}x^2$.

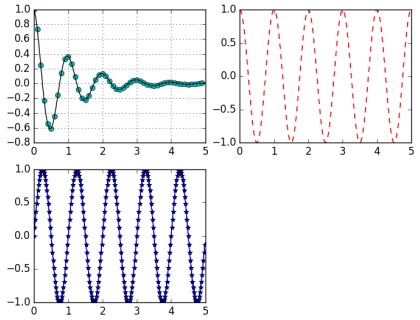


On peut aussi séparer le grapheur en plusieurs sous-figures avec `plt.subplot(xyz)` ou `plt.subplot(x,y,z)` où `x` est le nombre de lignes, `y` le nombre de colonnes, `z` le numéro de la figure.

```
def f(t) :
  return np.exp(-t) * np.cos(2*np.pi * t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.subplot(221)
plt.plot(t1, f(t1), 'co')
plt.plot(t2, f(t2), 'k')
plt.grid()
plt.subplot(222)
plt.plot(t2, np.cos(2*np.pi * t2), 'r--')
plt.subplot(223)
plt.plot(t2, np.sin(2*np.pi * t2), 'b*-')
plt.show()
```



10. La conservation de l'énergie mécanique pour un pendule simple permet d'obtenir l'équation : $E = \dfrac{1}{2}mL^2\dot{\theta}^2 + mgL(1-\cos\theta)$ d'où on peut tirer $\dot\theta$ en fonction de θ pour tracer un portrait de phase $dot\theta = \pm\sqrt{2(C-1+\cos\theta)}$.

10.a) Superposez quelques portraits de phase ($\dot\theta$ en fonction de θ) en faisant varier $C \geqslant 0$. Observer le cas limite $C = 2$ que vous devez avoir étudié en sciences physiques.

10.b) Avec l'approximation des petits angles, on sait résoudre l'équation différentielle, et on trouve, pour $\dot\theta(0) = 0$, $\theta(t) = \theta_0 \cos(\omega_0 t)$ avec $\omega_0 = \sqrt{\frac{g}{L}}$. On prend $L = 0{,}1$ m.

Tracer, dans des subplots d'une même figure, quelques θ en fonction de t et les portraits de phase de la question précédente.