# Informatique
# TP5 : Initiation aux outils de calcul scientifique
# CPP 2A

Romain Casati, Wafa Johal, Frederic Devernay, Matthieu Moy

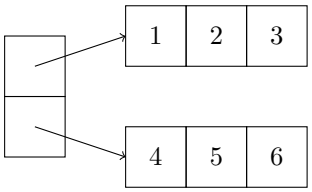
Septembre - octobre 2015

## 1 Utilisation de listes de listes pour représenter des matrices

Dans un premier temps, nous allons manipuler des matrices représentées par des listes de listes. Par exemple, la matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ Peut-être représentée en Python comme ceci :

```
A = [[1, 2, 3],
     [4, 5, 6]]
```

Techniquement, `A` est une liste à deux éléments (liste de lignes) qui sont eux-mêmes des listes à 3 éléments. Dans la mémoire de l'ordinateur, `A` a la structure suivante :




Téléchargez le fichier `addition.py` depuis la page du cours. Chargez-le dans Spyder et exécutez-le.

**Exercice 1 (Calculs simples dans l'interprète interactif)** *Pour chacune des expressions suivantes, essayez de deviner leur valeur, puis affichez-les avec l'interprète interactif Python :* `A`, `len(A)`, `A[0]`, `len(A[0])`, `A[0][1]` *et* `A[1][0]`.

*Que se passe-t-il si on évalue les expressions* `A[2]` *ou* `A[1][3]` *?*

*Exécutez l'instruction* `A[1][2] = 42` *puis affichez* `A`.

Vous avez pu remarquer que l'affichage fait par Python n'est pas très joli (il affiche par exemple `[[1, 2, 3], [4, 5, 6]]`, sur une seule ligne).

**Exercice 2 (Affichage d'une matrice en Python)** *Écrivez la fonction* `afficher_matrice(m)` *qui affiche la matrice* `m` *avec le format suivant :*

```
[ 1  2  3 ]
[ 4  5  6 ]
```

*Vous aurez besoin d'imbriquer deux boucles* `for`*. Pour afficher une valeur* `x` *sans revenir à la ligne, on peut utiliser* `print(x, end='')`*. Testez votre fonction sur* `A`, `B` *et* `C` *(cf. la fin de* `addition.py`*).*

Nous allons maintenant écrire une fonction `addition(a, b)` qui calcule la somme de deux matrices. La première chose à faire est de créer une matrice `resultat` vide. Pour créer une liste à `N` éléments vides, on peut utiliser `[None] * N` (par exemple, `[None] * 3` vaut `[None, None, None]`). On doit procéder en deux temps :

- Créer la liste de lignes (avec pour l'instant les valeurs `None` à la place des lignes) : `resultat = [None] * n`.
- Pour chaque élément de cette liste, créer une ligne et l'affecter : `resultat[i] = [None] * m`.

**Exercice 3 (Création d'une matrice vide)** *Écrivez la fonction* `creer_matrice(n, m)` *qui crée une matrice de* `n` *lignes et* `m` *colonnes. Testez votre code avec plusieurs valeurs de* `n` *et* `m`.

**Exercice 4 (Visualiser l'exécution de `creer_matrice` dans PythonTutor)** *Ouvrez votre navigateur à la page* ***http://pythontutor.com/****. Cliquez sur « Start writing and visualizing code now ! », sélectionnez « Python 3 » et entrez votre code dans le champ texte (la définition et au moins un appel de* `creer_matrice`*). Visualisez l'exécution.*

Une implémentation naïve de `creer_matrice(n, m)` est « return [[None] * n] * m », mais elle est incorrecte à cause de problèmes d'alias. Pour les curieux : exécutez ce code dans PythonTutor et/ou lisez l'encadré « Attention, Partage de tableau » page 161 du livre « Informatique pour tous en CPGE » pour comprendre.

**Exercice 5 (Somme de deux matrices)** *Écrivez maintenant la fonction* `addition(a, b)` *qui calcule la somme des matrices* `m1` *et* `m2`*. Il faut procéder en trois temps :*

- *Récupérer le nombre de lignes* `n` *et de colonnes* `m` *de* `m1` *ou* `m2` [1]
- *Appeler* `creer_matrice(n, m)`
- *Faire la somme composante par composante avec deux boucles* `for` *imbriquées.*

En pratique, utiliser des listes de listes en Python pour représenter des matrices est peu pratique, lent, et sujet à erreurs. La bibliothèque NumPy évite ces problèmes.

## 2 Rappels et compléments sur NumPy

NumPy est une bibliothèque Python qui permet à la fois de manipuler des tableaux multidimensionels mais aussi de faire des calculs sur ces objets. En cours, vous avez vu comment

- créer un tableau avec NumPy :

```
>>> import numpy
>>> v = numpy.array([1, 2, 3])
>>> v
array([1, 2, 3])
```

- faire des opérations algébriques sur les tableaux :

```
>>> w = 3 * v - numpy.array([3, 6, 9])
>>> w
array([0, 0, 0])
```

- multiplier des matrices et des vecteurs mais nous ne nous en servirons pas dans ce TP.

NumPy peut faire beaucoup plus que ça. Comme le module `math` de Python, NumPy dispose de la plupart des fonctions mathématiques que vous connaissez. Cependant, les fonctions NumPy ont l'avantage de s'appliquer sur les tableaux (composantes par composantes).

**Exercice 6 (NumPy et les fonctions)** *Que donne le code suivant ?*

```
import numpy
v = numpy.array([1, 2, 3, 4]) * numpy.pi / 4  # numpy.pi = π
w = numpy.sin(v)
print(w)
```

NumPy dispose aussi de fonctions permettant de générer des tableaux.

**Exercice 7 (Génération de tableaux)** *Que font les fonctions de NumPy suivantes :* `zeros([num])`*,* `ones([num])`*,* `linspace(debut, fin, num)`*,* `random.random([num])` *?*

## 3 Python + Matplotlib : mieux qu'une calculatrice graphique

Matplotlib est une bibliothèque Python qui permet de faire toutes sortes de tracés. Nous allons essentiellement l'utiliser pour tracer des fonctions. Les instructions de base pour se servir de Matplotlib sont illustrées ici :

1. Pour être rigoureux, il faudrait les tailles de `m1` *et* `m2`, et vérifier que ces tailles sont égales.

```
import matplotlib.pyplot as plt  # plt devient l'abrege de matplotlib.pyplot

x = [0, 1, 2]  # liste des abscisses
y = [1, -1, 0]  # liste des ordonnees

plt.plot(x, y)  #  trace y en fonction de x
plt.show()  # affiche la fenetre du trace
```

**Exercice 8 (Premiers tracés)** *En utilisant les fonctions* `linspace` *et* `cos` *de NumPy, tracer la fonction* $y = \cos(x)$ *sur* $[0, 10\pi]$. *Quelle est l'influence du nombre de points passé à la fonction* `linspace` *? Grâce à un second appel à la fonction* `plt.plot` *avant l'appel à* `plt.show`*, superposer le graphe de la fonction* $y = \exp(-x/10)\cos(x)$. *Toujours avant l'appel à* `plt.show`*, ajouter un titre avec* `plt.title('Le titre de votre choix')` *et des noms aux axes avec* `plt.xlabel('x')` *et* `plt.ylabel('y=f(x)')`.

Matplotlib n'est pas seulement capable de tracer des fonctions de variable réelle mais aussi des *courbes paramétrées*. Vous étudierez ces objets en mathématiques si ce n'est pas déjà fait mais pour faire simple, vous pouvez penser à la trajectoire d'un point au cours du temps. Décrire cette trajectoire, c'est se donner à chaque instant $t$ l'abscisse et l'ordonnée du point étudié. Plus formellement, on se donne une fonction d'un intervalle $I \subset \mathbb{R}$ de la forme $t \mapsto (x(t), y(t))$.

**Exercice 9 (Courbes paramétriques)** *En modifiant le code ci-dessous, tracer l'une des courbes suivantes :*

- *La Lemniscate de Bernoulli* $\begin{cases} x(t) = \frac{\sin t}{1+\cos^2 t} \\ y(t) = \frac{\sin t \cos t}{1+\cos^2 t} \end{cases}$ *sur* $[0, 2\pi]$.
- *La spirale d'Archimède* $\begin{cases} x(t) = t\cos(t) \\ y(t) = t\sin(t) \end{cases}$ *sur* $[0, 10\pi]$.
- *La courbe du cœur* $\begin{cases} x(t) = 16\sin^3 t \\ y(t) = 13\cos t - 5\cos(2t) - 2\cos(3t) - \cos(4t) \end{cases}$ *sur* $[0, 2\pi]$.
- *Les cyclo-harmoniques* $\begin{cases} x(t) = \left(1 + \cos\left(\frac{p}{q}t\right)\right)\cos(t) \\ y(t) = \left(1 + \cos\left(\frac{p}{q}t\right)\right)\sin(t) \end{cases}$ *pour* $p$ *et* $q$ *entiers sur* $[0, 2q\pi]$.

```
import matplotlib.pyplot as plt
import numpy as np  # np devient l'abrege de numpy

t = np.linspace(..., ..., ...)

x = ...  # x(t)
y = ...  # y(t)

plt.plot(x, y)
plt.show()
```

Le corrigé de la section 1 est sur la page du cours.

**Correction exercice 6** *On s'attend à trouver* $\left[\frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 0\right]$

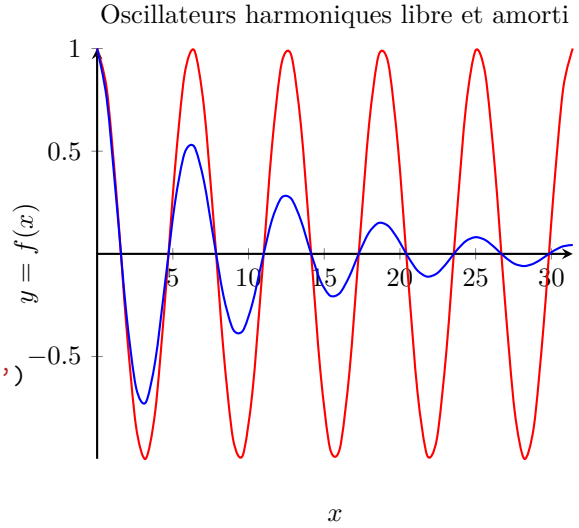
```
[  7.07106781e-01   1.00000000e+00   7.07106781e-01   1.22464680e-16]
```

**Correction exercice 7**

- `zeros([num])` *retourne un tableau de taille* `num` *ne contenant que des 0.*
- `ones([num])` *retourne un tableau de taille* `num` *ne contenant que des 1.*
- `linspace(a, b, n)` *retourne un échantillonage uniforme de l'intervalle* $[a, b]$ *de taille* $n$*. Les éléments sont les* $a + k\frac{b-a}{n-1}$ *pour* $k = 0, ..., n-1$*.*
- `random.random([num])` *retourne un vecteur de taille* `n` *dont les éléments sont tirés aléatoirement entre 0 et 1.*

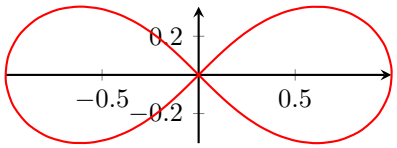
**Correction exercice 8** *Si le nombre de points passé à la fonction* `linspace` *n'est pas suffisant, le graphe ressemble à une ligne brisée et n'est pas lisse.*

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

x = np.linspace(0, 10 * np.pi, 100)
y1 = np.cos(x)
y2 = np.exp(-x/10) * y1

plt.plot(x, y1)
plt.plot(x, y2)

plt.title('Oscillateurs harmoniques libre et amorti')
plt.xlabel('x')
plt.ylabel('y = f(x)')
plt.show()
```

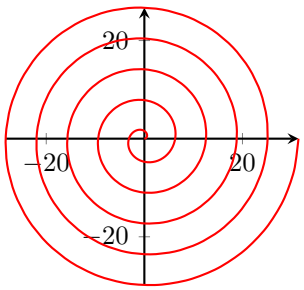



**Correction exercice 9**

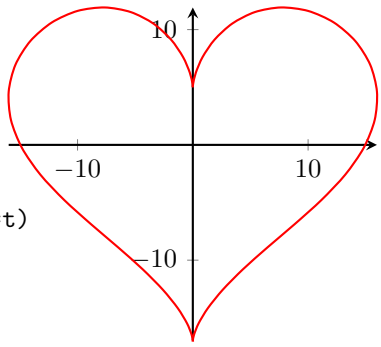
```
import matplotlib.pyplot as plt
import numpy as np

t = np.linspace(0, 2 * np.pi, 100)
# Lemniscate de Bernoulli
x = np.sin(t) / (1 + np.cos(t) ** 2)
y = x * np.cos(t)

plt.plot(x, y)
plt.show()
```




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

t = np.linspace(0, 10 * np.pi, 300)
# Spirale d'Archimede
x = t * np.cos(t)
y = t * np.sin(t)

plt.plot(x, y)
plt.show()
```



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

t = np.linspace(0, 2 * np.pi, 100)
# Courbe du coeur
x = 16 * np.sin(t) ** 3
y = 13*np.cos(t) - 5*np.cos(2*t) - 2*np.cos(3*t) - np.cos(4*t)

plt.plot(x, y)
plt.show()
```




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

p = 8
q = 3

t = np.linspace(0, 2 * q * np.pi, 400)
# Cyclo-harmonique
tmp = 1 + np.cos(p / q * t)
x = tmp * np.cos(t)
y = tmp * np.sin(t)

plt.plot(x, y)
plt.show()
```