Mémento numérique Python 3

`import matplotlib.pyplot as plt` → charge le module pyplot sous le nom `plt`

`plt.figure('titre')` → crée une fenêtre de tracé vide

`plt.plot(LX,LY,'o-b')` → trace le graphique défini par les listes LX et LY (abscisses et ordonnées)
- couleur : `'b'` (**blue**), `'g'` (**green**), `'r'` (**red**), `'c'` (cyan), `'m'` (**magenta**), `'y'` (yellow), `'k'` (**black**)
- type de ligne : `'-'` (trait plein), `'--'` (pointillé), `'-.'` (alterné)...
- marque : `'o'` (rond), `'h'` (hexagone), `'+'` (plus), `'x'` (croix), `'*'` (étoile)...

`plt.xlim(xmin,xmax)` → fixe les bornes de l'axe x

`plt.ylim(ymin,ymax)` → fixe les bornes de l'axe y

`plt.axis('equal')` → change les limites des axes x et y pour un affichage avec des axes orthonormés (le tracé d'un cercle donne un cercle)

`plt.show()` → affichage de la fenêtre

`plt.savefig(fichier)` → sauve le tracé dans un fichier (le suffixe du nom fichier peut donner le format ; par exemple, `'image.png'`)

`import numpy as np` → charge le module numpy sous le nom `np`

Construction de tableaux (de type ndarray)

`np.zeros(n)` → crée un vecteur dont les n composantes sont nulles

`np.zeros((n,m))` → crée une matrice $n \times m$, dont les éléments sont nuls

`np.eye(n)` → crée la matrice identité d'ordre n

`np.linspace(a,b,n)` → crée un vecteur de n valeurs régulièrement espacées de a à b

`np.arange(a,b,dx)` → crée un vecteur de valeurs de a incluse à b exclue avec un pas dx

`M.shape` → tuple donnant les dimensions de M

`M.size` → le nombre d'éléments de M

`M.ndim` → le nombre de dimensions de M

`M.sum()` → somme de tous les éléments de M

`M.min()` → plus petit élément de M

`M.max()` → plus grand élément de M

argument **axis** optionnel : 0 → lignes, 1 → colonnes :

`M.sum(0)` → somme des lignes

`M.min(0)` → plus petits éléments, sur chaque colonne

`M.max(1)` → plus grands éléments, sur chaque ligne

`import numpy.linalg as la`

`la.det(M)` → déterminant de la matrice carrée M

`la.inv(M)` → inverse de M

`la.eigvals(M)` → valeurs propres de M

`la.matrix_rank(M)` → rang de M

`la.matrix_power(M,n)` → M^n (n entier)

`la.solve(A,B)` → renvoie X tel que $A\,X = B$

`import scipy.integrate as spi`

`spi.odeint(F,Y0,LT)`
→ renvoie une solution numérique du problème de Cauchy $\mathbf{Y}'(t) = \mathbf{F}(\mathbf{Y}(t),t)$, où $\mathbf{Y}$ est un vecteur d'ordre n, avec la condition initiale $\mathbf{Y}(t_0) =$ `Y0`, pour les valeurs de t dans la liste `LT` de longueur k commençant par t_0, sous forme d'une matrice $n \times k$

`spi.quad(f,a,b)` → renvoie une évaluation numérique de l'intégrale : $\int_a^b f(t)\,dt$

Conversion ndarray <-> liste

`V = np.array([1,2,3])` → V : vecteur (1 2 3)

`L = V.tolist()` → L : liste [1, 2, 3]

`M = np.array([[1,2],[3,4]])` → M : matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

`L = M.tolist()` → L : liste [[1, 2], [3,4]]

Extraction d'une partie de matrice

`M[i], M[i,:]` → ligne de M d'index i

`M[:,j]` → colonne de M d'index j

`M[i:i+h,j:j+l]` → sous-matrice $h \times l$

Copier un tableau avec la méthode copy :

`M2 = M1.copy()`

`M1+M2, M1*M2, M**2` → opérations « terme-à-terme »

`c*M` → multiplication de la matrice M par le scalaire c

`M+c` → matrice obtenue en ajoutant le scalaire c à chaque terme de M

`V1.dot(V2)` / `np.dot(V1,V2)` → renvoie le produit scalaire de deux vecteurs

`M.dot(V)` / `np.dot(M,V)` → renvoie le produit d'une matrice par un vecteur

`M1.dot(M2)` / `np.dot(M1,M2)` → renvoie le produit de deux matrices

`M.transpose()` / `np.transpose(M)` → renvoie une copie de M transposée (ne modifie pas M)

`M.trace()` / `np.trace(M)` → renvoie la trace de M

Fonctions mathématiques usuelles

`np.exp, np.sin, np.cos, np.sqrt` *etc.*

→ fonctions qui s'appliquent sur des réels ou des complexes, mais aussi sur des vecteurs et des matrices (s'appliquent à chaque terme), qui sont optimisées en durée de calcul.

Rappel : ce mémento est fourni à titre indicatif. Il ne faut le considérer ni comme exhaustif, ni comme exclusif, ni comme un minimum à connaître absolument (l'examinateur n'attend pas du candidat qu'il connaisse parfaitement toutes ces fonctions et ces commandes).

Mémento numérique Python 3

`import matplotlib.pyplot as plt` → charge le module pyplot sous le nom `plt`

`plt.figure('titre')` → crée une fenêtre de tracé vide

`plt.plot(LX,LY,'o-b')` → trace le graphique défini par les listes LX et LY (abscisses et ordonnées)
- couleur : `'b'` (**blue**), `'g'` (**green**), `'r'` (**red**), `'c'` (cyan), `'m'` (**magenta**), `'y'` (yellow), `'k'` (**black**)
- type de ligne : `'-'` (trait plein), `'--'` (pointillé), `'-.'` (alterné)...
- marque : `'o'` (rond), `'h'` (hexagone), `'+'` (plus), `'x'` (croix), `'*'` (étoile)...

`plt.xlim(xmin,xmax)` → fixe les bornes de l'axe x

`plt.ylim(ymin,ymax)` → fixe les bornes de l'axe y

`plt.axis('equal')` → change les limites des axes x et y pour un affichage avec des axes orthonormés (le tracé d'un cercle donne un cercle)

`plt.show()` → affichage de la fenêtre

`plt.savefig(fichier)` → sauve le tracé dans un fichier (le suffixe du nom fichier peut donner le format ; par exemple, `'image.png'`)

`import numpy as np` → charge le module numpy sous le nom `np`

Construction de tableaux (de type ndarray)

`np.zeros(n)` → crée un vecteur dont les n composantes sont nulles

`np.zeros((n,m))` → crée une matrice $n \times m$, dont les éléments sont nuls

`np.eye(n)` → crée la matrice identité d'ordre n

`np.linspace(a,b,n)` → crée un vecteur de n valeurs régulièrement espacées de a à b

`np.arange(a,b,dx)` → crée un vecteur de valeurs de a incluse à b exclue avec un pas dx

`M.shape` → tuple donnant les dimensions de M

`M.size` → le nombre d'éléments de M

`M.ndim` → le nombre de dimensions de M

`M.sum()` → somme de tous les éléments de M

`M.min()` → plus petit élément de M

`M.max()` → plus grand élément de M

argument **axis** optionnel : 0 → lignes, 1 → colonnes :

`M.sum(0)` → somme des lignes

`M.min(0)` → plus petits éléments, sur chaque colonne

`M.max(1)` → plus grands éléments, sur chaque ligne

`import numpy.linalg as la`

`la.det(M)` → déterminant de la matrice carrée M

`la.inv(M)` → inverse de M

`la.eigvals(M)` → valeurs propres de M

`la.matrix_rank(M)` → rang de M

`la.matrix_power(M,n)` → M^n (n entier)

`la.solve(A,B)` → renvoie X tel que $A\,X = B$

`import scipy.integrate as spi`

`spi.odeint(F,Y0,LT)`
→ renvoie une solution numérique du problème de Cauchy $\mathbf{Y}'(t) = \mathbf{F}(\mathbf{Y}(t),t)$, où $\mathbf{Y}$ est un vecteur d'ordre n, avec la condition initiale $\mathbf{Y}(t_0) =$ `Y0`, pour les valeurs de t dans la liste `LT` de longueur k commençant par t_0, sous forme d'une matrice $n \times k$

`spi.quad(f,a,b)` → renvoie une évaluation numérique de l'intégrale : $\int_a^b f(t)\,dt$

Conversion ndarray <-> liste

`V = np.array([1,2,3])` → V : vecteur (1 2 3)

`L = V.tolist()` → L : liste [1, 2, 3]

`M = np.array([[1,2],[3,4]])` → M : matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

`L = M.tolist()` → L : liste [[1, 2], [3,4]]

Extraction d'une partie de matrice

`M[i], M[i,:]` → ligne de M d'index i

`M[:,j]` → colonne de M d'index j

`M[i:i+h,j:j+l]` → sous-matrice $h \times l$

Copier un tableau avec la méthode copy :

`M2 = M1.copy()`

`M1+M2, M1*M2, M**2` → opérations « terme-à-terme »

`c*M` → multiplication de la matrice M par le scalaire c

`M+c` → matrice obtenue en ajoutant le scalaire c à chaque terme de M

`V1.dot(V2)` / `np.dot(V1,V2)` → renvoie le produit scalaire de deux vecteurs

`M.dot(V)` / `np.dot(M,V)` → renvoie le produit d'une matrice par un vecteur

`M1.dot(M2)` / `np.dot(M1,M2)` → renvoie le produit de deux matrices

`M.transpose()` / `np.transpose(M)` → renvoie une copie de M transposée (ne modifie pas M)

`M.trace()` / `np.trace(M)` → renvoie la trace de M

Fonctions mathématiques usuelles

`np.exp, np.sin, np.cos, np.sqrt` *etc.*

→ fonctions qui s'appliquent sur des réels ou des complexes, mais aussi sur des vecteurs et des matrices (s'appliquent à chaque terme), qui sont optimisées en durée de calcul.

Rappel : ce mémento est fourni à titre indicatif. Il ne faut le considérer ni comme exhaustif, ni comme exclusif, ni comme un minimum à connaître absolument (l'examinateur n'attend pas du candidat qu'il connaisse parfaitement toutes ces fonctions et ces commandes).