Introduction à Python

Salim Lardjane
Université de Bretagne Sud

Les bases

Installation

- Le langage Python est très compact mais dispose d'une grande quantité d'extensions et de librairies qui permettent d'effectuer un grand nombre de tâches

- Les plus utilisées pour la Statistique, le Data Mining et le Machine Learning sont NumPy, SciPy et Matplotlib.

- La façon la plus simple d'installer Python est d'installer la suite Anaconda, qui contient aussi le Notebook Jupyter et l'IDE Spyder.

Débuter

- La façon la plus simple d'utiliser Python est via un IDE comme Spyder. On dispose alors d'une fenêtre d'édition et d'une console de commande permettant de saisir des instructions Python.

- Comme R, Python exécute immédiatement les commande saisies dans la console.

Variables

- On crée une variable en Python comme en R : on lui donne un nom et on lui affecte une valeur.

- Le symbole d'affectation est = .

- Python est un langage fortement typé (les variables entières par exemple restent entières à moins qu'on spécifie explicitement un nouveau type).

- Cependant, Python prend en charge lui-même la déclaration et la création des variables, contrairement à C, par exemple.

Opérations arithmétiques

- On peut effectuer toutes les opérations arithmétiques usuelles sous Python.

- L'élévation à une puissance est faite, par exemple, avec a**2 ou pow(a,2).

Comparaisons

- L'égalité est testée à l'aide de l'opérateur ==, qui renvoie une valeur booléenne : True (1) ou False (0).

- Les autres opérateurs de comparaison standards sont disponibles tels <, <=, >, >=, ainsi que l'opérateur != (non égal), également noté <>.

- Un autre opérateur utile est l'opérateur is qui permet de vérifier si deux variables pointent vers le même objet.

- C'est important car Python travaille par référence, c'est-à-dire que la commande a = b ne recopie pas la valeur de b dans a mais assigne à a une référence à la variable b. On y reviendra dans la suite.

Opérateurs logiques

- Les opérateurs logiques sont un peu différents sous Python.

- Les opérateurs and, or et not sont les opérateurs standards.

- Les symboles & et | servent à effecteur un « et » et un « ou » bit par bit, respectivement, ce qui peut être très utile.

Types

- Le types les plus utilisés sont les types entier (integer) et flottant (float).

- Python permet aussi de manipuler des chaînes de caractères (string).

- Celles-ci sont spécifiées à l'aide de guillemets doubles " ou simples '.

- L'opérateur + est surchargé pour les chaînes puisqu'il correspond à la concaténation.

Structures de données

- Les types de base précédents peuvent être combinés en différentes structures de données.

- Listes : une liste est une combinaison de types de base, encadrée par des crochets [ ].

- Exemple : maliste = [0, 3, 2,'bonjour'] est une liste valide, contenant des entiers et une chaîne.

- Cela est possible car Python est un langage orienté objet.

- Chaque variable est un objet et une liste est simplement une collection d'objets.

- On peut donc également construire des listes de listes sans problème.

- Exemple : nouvelleliste = [3, 2, [5, 4, 3], [2, 3, 2]]

Listes

- L'accès aux éléments d'une liste se fait simplement à l'aide d'un indice.

- Comme sous C, mais pas comme sous R, l'indexation sous Python commence à 0.

- On peut également indexer à partir de la fin en utilisant le signe - (qui a donc une signification différente sous Python et sous R).

- Exemple : nouvelleliste[-1] renvoie le dernier élément de la liste, nouvelleliste[-2] l'avant dernier, etc.

Listes

- La longueur d'une liste est donnée par la commande len.

- Ainsi, len(nouvelleliste) renvoie 4.

- Remarque : nouvelleliste[3] renvoie la liste en 3ème position dans nouvelleliste ; pour accéder à un élément de cette liste, il faut spécifier un nouvel index : nouvelleliste[3][1] renvoie 3.

Opérateur slice

- L'opérateur slice, noté :, permet d'accéder facilement à des parties d'une liste.

- Par exemple, nouvelleliste[2:4] renvoie les éléments de nouvelleliste en positions 2 et 3.

- l'opérateur slice est inclusif au début et exclusif à la fin (le deuxième paramètre est le premier exclu).

- En fait, l'opérateur slice peut prendre trois arguments [start:stop:step], le troisième élément spécifiant le pas à utiliser.

- Nouvelleliste[0:4:2] renvoie les éléments en positions 0 et 2.

Opérateur slice

- On peut utiliser l'opérateur slice pour inverser l'ordre d'une liste : nouvelleliste[::-1].

- Si on ne spécifie pas l'argument start (par exemple [:3]), celui-ci vaut 0; si on ne spécifie pas de valeur pour l'argument stop (par exemple [1:]), on considère qu'on va jusqu'à la fin de la liste.

- Ces opérateurs sont très utiles, particulièrement le deuxième, puisqu'ils permettent d'éviter de calculer la longueur d'une liste chaque fois qu'on veut la parcourir.

- nouvelleliste[:] renvoie toute la liste.

Opérateur slice

- Cette dernière utilisation de l'opérateur slice peut sembler inutile.
- Cependant, comme Python est orienté objet, tous les noms de variables sont simplement des références à des objets.
- Cela signifie que recopier une variable de type list n'est pas aussi évident qu'il y paraît.
- Exemple : la commande uneliste = maliste ne fait pas de copie de maliste.
- Pour le vérifier, saisissez uneliste[3] = 100 et vérifiez le contenu de maliste. Vous verrez que le troisième élément vaut 100 à présent.
- On doit donc faire attention lorsqu'on recopie des objets.

Opérateur slice

- L'opérateur slice permet de faire des copies.
- Par exemple, uneliste = maliste[:].
- Malheureusement, il y a un problème supplémentaire si on a une liste de listes.
- Les listes sont des références à des objets. On a utilisé l'opérateur slice pour accéder à la valeur des objets, mais ceci ne fonctionne que sur un niveau.
- En position 2 de nouvelleliste, il y a une liste, et l'opérateur slice ne fait que recopier la référence à cette liste.

Deep Copy

- Pour tout copier, on doit utiliser la commande deepcopy.

- Pour pouvoir l'utiliser, on doit importer le module copy à l'aide de la commande import copy

- On peut alors saisir cliste = copy.deepcopy(nouvelleliste) ce qui permet d'avoir une copie de la liste complète.

Fonctions sur listes

- Diverses fonctions peuvent être appliquées aux listes mais le fait qu'une liste soit un objet a d'autres conséquences intéressantes.

- Les fonctions (méthodes) qui peuvent être utilisées font partie de la classe objet et modifient la liste elle-même sans renvoyer de nouvelle liste.

- Exemple : liste = [3, 2, 4, 1]

- Supposons qu'on souhaite afficher une liste de ces nombres ordonnés.

Fonctions sur listes

- Il existe une fonction sort() pour ce faire, mais la commande immédiate liste.sort() renvoie la sortie None, ce qui signifie qu'aucune valeur n'a été renvoyée.

- Cependant, les deux commandes liste.sort() suivie de print liste font exactement ce que l'on souhaite.

- Ainsi, les fonctions sur listes modifient les listes et toute opération subséquente est appliquée à la liste modifiée.

Fonctions sur listes

- D'autres fonctions disponibles opérant sur les listes :
- append(x) : rajoute x en fin de liste
- count(x) : calcule le nombre d'occurrences de x dans la liste
- extend(L) : rajoute les éléments de la liste L en fin de liste
- index(x) : renvoie l'indice du premier élément de la liste correspondant à x

Fonctions sur listes

- insert(i,x) : insère l'élément x à l'emplacement i de la liste et décale tout le reste

- pop(i) : enlève l'élément d'indice i

- remove(x) : supprime le premier élément qui correspond à x

- reverse() : inverse l'ordre de la liste

- sort() : déjà vu

Comparaison de listes

- On peut comparer des listes à l'aide de l'opérateur == qui fonctionne élément par élément, comparant chaque élément à l'élément correspondant de la deuxième liste

- Le résultat est True si le test est vrai pour chaque paire d'éléments (et si les deux listes sont de même longueur), et False sinon.

Tuples

- Un tuple est une liste immuable, ce qui signifie qu'il est accessible uniquement en lecture et ne peut être modifié.

- Les tuples sont définis à l'aide de parenthèses.

- Exemple : montuple = (0 ,3, 2, 'h').

- Ils sont utiles pour spécifier des listes qui ne peuvent être modifiées, même par erreur.

Dictionnaires

- Dans les listes qu'on a vues précédemment, chaque élément était identifié par son indice dans la liste.

- Dans un dictionnaire, on assigne une clef à chaque élément pour y accéder.

- Supposons qu'on souhaite faire une liste du nombre de jours dans chaque mois. On peut utiliser pour cela un dictionnaire.

- Celui est défini par des accolades { }.

- Exemple : mois = { 'Jan' : 31, 'Fev' : 28, ' Mar' : 31}

- Pour accéder à un élément, on utilise sa clef : mois['Jan'] renvoie 31.

- Spécifier une clef non valide génère un message d'erreur.

Dictionnaires

- La fonctions mois.keys() renvoie une liste de toutes les clefs du dictionnaire, ce qui est utile pour boucler sur les éléments du dictionnaire.

- La fonction mois.values() renvoie la liste des valeurs correspondant à toutes les clefs.

- La fonction mois.items() renvoie une liste de tuples contenant tout.

Fichiers

- Il y a un autre type de données sous Python et c'est le fichier.
- Il permet de lire et d'écrire très simplement dans des fichiers.
- Les fichiers sont ouverts à l'aide de la commande input = open('filename') et fermés à l'aide de input.close().
- La lecture et l'écriture sont réalisées à l'aide des commandes readlines(), read(), writelines() et write().
- Les fonctions readline() et writeline() lisent et écrivent une ligne à la fois.

Python pour utilisateurs de R

- Le package Python Numpy présente beaucoup de similitudes avec R.

- La principale différence est que, sous Python, l'indexation commence à 0 au lieu de 1.

Bases du code

- Python dispose d'un petit ensemble de commandes conçu pour être le plus réduit et le plus facile d'utilisation possible.

- Dans la suite, on va s'intéresser à cet ensemble de commandes.

Ecrire et importer du code

- Python est un langage scripté, ce qui signifie que tout peut être exécuté de façon interactive à partir de la ligne de commande.

- Toutefois, dès qu'on souhaite soumettre une quantité réaliste de code, il est préférable d'utiliser un IDE comme Spyder.

- Point important : c'est l'indentation qui définit les blocs sous Python. Il est dont préférable de disposer d'un éditeur qui gère celle-ci correctement.

Ecrire et importer du code

- Le fichier peut contenir un script, c'est-à-dire une simple suite de commandes, ou un ensemble de fonctions et de classes.

- Dans tous les cas, il doit être sauvegardé avec l'extension .py ; il est compilé par Python en un fichier .pyc quand on le charge pour la première fois.

- Tout ensemble de commandes ou de fonctions est appelé module sous Python et pour le charger, on utilise la commande import.

Ecrire et importer du code

- La forme la plus simple de la commande est : import nom

- Si on importe un fichier script, alors Python l'exécute immédiatement, mais si c'est un ensemble de fonctions, il n'exécute rien.

- Pour lancer une fonction, on utilise nom.nomdefonction(), où nom est le nom du module et nomdefonction le nom de fonction adapté.

- Les arguments peuvent être passés entre parenthèses, mais même s'il n'y en a pas, celles-ci sont requises.

- Certains noms sont très longs ; il peut donc être utile d'utiliser import x as y, ce qui permet d'utiliser la commande y.nomdefonction().

Ecrire et importer du code

- Lorsqu'on développe du code en ligne de commande, une caractéristique gênante de Python est qu'import ne fonctionne qu'une seule fois pour un module.

- Une fois qu'un module a été chargé, si on change le code et qu'on souhaite que Python utilise la nouvelle version, on doit utiliser la commande reload(nom)

- Sinon, utiliser import ne génère pas de message d'erreur mais ne fonctionne tout simplement pas.

Ecrire et importer du code

- De nombreux modules contiennent plusieurs sous-ensembles ; ainsi, il peut être nécessaire de préciser les choses lors d'une importation.

- On peut importer une partie spécifique d'un module à l'aide d'une commande du type from x import y ou tout importer à l'aide de la commande from x import *.

- Cette dernière solution n'est pas optimale puisque certains modules peuvent être très longs.

- Enfin, on peut spécifier un nom à utiliser pour le module importé en utilisant from x import y as z.

Ecrire et importer du code

- Dans les programmes, on doit importer tout les modules utilisés et c'est fait généralement en début de programme (bien que cela ne soit pas imposé).

- De plus, Python utilise la variable pythonpath pour déterminer où chercher le code.

- Celle-ci peut être modifiée à l'aide d'une option de menu sous Spyder.

- Plus généralement, c'est fait sous Python à l'aide des commandes : import sys puis sys.path.append('mypath')

Instructions de contrôle

- La caractéristique la plus étrange de Python comparé aux autres langages de programmation est que l'indentation a un sens : les espaces servent à définir des blocs de code.

- Ainsi, pour une boucle ou une autre instruction de contrôle, l'équivalent des accolades de R {} est un double point après le mot-clef et des commandes intentées à la suite.

- Cela peut paraître étrange au début mais c'est très pratique lorsqu'on s'y habitue.

Instructions de contrôle

- L'autre caractéristique étrange de Python est qu'on dispose d'une instruction else (optionnelle) pour les boucles.

- Cette clause est exécutée lorsque la boucle se termine normalement. Si on sort de la boucle à l'aide de la commande break, la clause else n'est pas exécutée.

- Les structures de contrôle disponibles sont if, for et while.

If

- La syntaxe du if est la suivante :

```
if énoncé:
    commandes
elif:
    commandes
else:
    commandes
```

For

- La boucle la plus commune est la boucle for, qui diffère légèrement sous Python en ce qu'elle itère sur les éléments d'une liste de valeurs :

```
for var in ensemble:
    commandes
else:
    commandes
```

For

- Une commande très utile à utiliser avec les boucles for est la commande range, qui renvoie une liste.

- Sa forme la plus simple est par exemple range(4), qui renvoie la liste [0, 1, 2, 3].

- Toutefois, elle peut également prendre deux ou trois arguments, et fonctionne de la même façon que l'opérateur slice, mais avec des virgules à la place des double points : .

- Exemple : range(start, stop, step)

- Cela inclut le fait de parcourir la liste dans l'autre sens. Ainsi, range(5,-3,-2) renvoie [5, 3, 1, -1].

While

```
while condition:
    commandes
else:
    commandes
```

Fonctions

- Les fonctions sont définies par

```
def name(args):
    commandes

    return valeur
```

- La ligne `return valeur` est optionnelle et permet de renvoyer une valeur ; sinon, la fonction renvoie `None`

Fonctions

- On peut spécifier plusieurs objets à retourner en les séparant par des virgules

- Une fois la fonction définie, on peut l'appeler depuis la ligne de commande et depuis d'autres fonctions

- Python, comme R, est sensible à la capitalisation, donc pour les variables et les fonctions, Nom est différent de nom

Fonctions

- Exemple : fonction calculant la longueur de l'hypoténuse d'un triangle rectangle à partir des longueurs des autres côtés (x et y) :

```
def pythagore(x,y):
    '' '' ''Calcul de la longueur de l'hypothénuse '' '' ''
    h =  pow(x**2+y**2,0.5)
    # pow(x,0.5) est la racine carrée
    return h
```

Fonctions

- L'appel de pythagore(3,4) renvoie la valeur attendue 5.0

- On peut également spécifier les paramètres par mots-clefs, comme dans pythagore(y=4,x=3)

- On peut spécifier des valeurs par défaut des arguments, comme dans

```
def pythagore(x=3, y=4):
```

La chaîne doc

- On accède à l'aide sous Python en utilisant help()
- Pour l'aide sur un module spécifique, on fait help(nom du module)
- Exemple : help(pythagore)
- Une ressource utile est la chaîne doc, qui est la première chose définie dans une fonction. C'est une chaîne de caractère comprises entre trois guillemets doubles ''' '''.
- Elle sert à documenter la fonction ou la classe
- On y accède à l'aide de la commande

print nomdefonction.__doc__

- Le générateur de documentation de Python, pydoc, utilise ces chaînes de caractères pour documenter automatiquement les fonctions.

map et lambda

- Python traite les appels répétés de fonction de façon spéciale.

- Si on souhaite appliquer la même fonction à tous les éléments d'une liste, on n'a pas besoin de boucler sur les éléments de la liste ; on utilise à la place la commande map avec la syntaxe map(fonction, liste).

- Cela applique la fonction à tous les éléments de la liste.

map et lambda

- Détail supplémentaire : la fonction utilisée peut être anonyme (créée juste pour cette utilisation, sans avoir de nom) en utilisant la commande lambda, avec la syntaxe lambda args : commande.

- Une fonction lambda n'exécute qu'une commande, mais permet d'écrire du code très court pour faire des choses relativement compliquées.

map et lambda

- Exemple : l'instruction suivante calcule le cube de chaque élément d'une liste et lui rajoute 7

```
map(lambda x:pow(x,3)+7,liste)
```

- Une autre façon d'utiliser lambda est de l'utiliser en conjonction avec la commande filter. Cela permet de renvoyer les éléments d'une liste pour laquelle une condition est vraie.

- Exemple : filter(lambda x:x>=2,liste)

- Ceci peut être fait plus simplement avec NumPy pour les tableaux, comme on le verra dans la suite.

Classes

- Pour ceux qui souhaitent l'utiliser de cette façon, Python est complètement orienté objet.

- Les classes et leurs constructeurs sont définies par la syntaxe :

```
class maclasse(superclasse):

    def __init__(self,args):

    def nomdefonction(self,args):
```

Classes

- Si on ne spécifie pas de super-classe, alors la classe n'hérite de rien.

- Le constructeur est __init__(self, args).

- On peut également définir un destructeur __del__(self), bien que cela soit rarement utilisé.

- La syntaxe permettant d'accéder aux méthodes est

nomdeclasse.nomdefonction()

- L'argument self peut être ignoré dans les appels de fonctions, Python se chargeant de l'inclure, mais doit être spécifie dans la définition des fonctions

NumPy et Matplotlib

Utilisation

- Pour obtenir de l'aide sur une fonction NumPy, on utilise la commande help(np.nomdefonction), par exemple help(np.dot).

- NumPy propose une collection de base de fonctions mais des packages additionnels doivent être importés si on veut les utiliser.

- Pour importer la librairie NumPy de base, on saisit :

```
import numpy as np
```

Tableaux

- La structure de données de base pour l'analyse numérique et la programmation statistique est celle d'array.

- Elle correspond aux tableaux multidimensionnels (matrices) dans les autres langages.

- Elle consiste en une dimension ou plus de nombres ou de caractères.

- Contrairement aux listes Python, les éléments d'un array (tableau) doivent être de même type. Ce dernier peut être Booléen, entier, réel ou complexe.

Tableaux

- Les arrays sont définies à l'aide d'un appel de fonction et les valeurs sont passées en liste ou en un ensemble de listes, en dimension supérieures.

- Les tableaux sous Python peuvent avoir jusqu'à 40 dimensions.

Exemples de tableaux uni- et bi-dimensionnels :

```
myarray = np.array([4, 3, 2])

mybigarray = np.array([[3,2,4],[3,3,2],[4,5,2]])

print myarray

print mybigarray
```

Fonctions de création de tableaux

- np.arange() produit un tableau contenant les valeurs spécifiées et correspond à la version pour tableaux de range. Exemple : np.arange(5) = array([0,1,2,3,4]) et np.arange(3,7,2) = array([3,5])

- np.ones() produit un tableau ne contenant que des un. Pour np.ones() et np.zeros(), on doit utiliser deux parenthèses pour construire des tableaux de dimensions supérieures à un.

- np.ones(3) = array([1.,1.,1.])

- np.ones((3,4)) = array([[1.,1.,1.,1.],[1.,1.,1.,1.],[1.,1.,1.,1.]])

- On peut spécifier le type du tableau en faisant par exemple a = np.ones((3,4), dtype = float).

Fonctions de création de tableaux

- np.zeros() : analogue à np.ones() excepté qu'elle renvoie des 0 au lieu de 1

- np.eye() : Produit la matrice identité, c'est-à-dire la matrice contenant des zéros partout sauf sur la diagonale, où elle contient des un.

- Avec un argument, elle produit une matrice identité carrée : np.eye(3) = [[1. 0. 0.] [0. 1. 0.] [0. 0. 1.]]

- Avec deux arguments elle remplit les lignes et colonnes en plus avec des zéros : np.eye(3,4) = [[1. 0. 0. 0.] [0. 1. 0. 0.] [0. 0. 1. 0.]]

Fonctions de création de tableaux

- np.linspace(start,stop,npoints) : produit une matrice contenant les éléments d'un suite arithmétique. On spécifie le nombre d'éléments et non pas le pas. Exemple : np.linspace(3,7,3) = array([3., 5., 7.])

- np.r_[] et np.c_[] permettent de concaténer en lignes et en colonnes respectivement . Exemple : np.r[1:4,0,4] = array([1,2,3,0,4]).

Opérations sur tableaux

- Soit a le tableau défini par a = np.arange(6).reshape(3,2), ce qui produit

array([[0, 1],[2, 3],[4, 5]])

- L'indexation des tableaux est faite à l'aide des crochets [et]. Les indices commencent à 0.

- Ainsi, a[2,1] vaut 5 et a[:,1] renvoie array([1, 3, 5]).

Opérations sur tableaux

- np.ndim(a) : renvoie le nombre de dimensions (ici 2)
- np.size(a) : renvoie le nombre d'éléments (ici, 6)
- np.shape(a) : renvoie la taille du tableau sur chaque dimension (ici (3,2)). On accède au premier élément du résultat en faisant shape(a)[0].
- np.reshape(a,(2,3)) : modifie les dimensions du tableau de la façon spécifiée. On peut utiliser -1 pour une dimension, ce qui indique à Python qu'il doit calculer la dimension requise. Exemple : np.reshape(a,(2,-1)) ou np.reshape(a, (-1,2))

Opérations sur tableaux

- np.ravel(a) : transforme le tableau en vecteur (tableau uni-dimensionnel). Ici, cela donne array([0, 1, 2, 3, 4, 5,]).

- np.transpose(a) : calcule la transposée

- a[[::-1]] : inverse l'ordre des éléments sur chaque dimension

- np.min(a), np.max(a), np.sum(a) renvoient le plus petit élément, le plus grand élément et la somme des éléments de la matrice. Souvent utilisé pour calculer la somme des lignes ou des colonnes à l'aide de l'option axis : np.sum(axis=0) pour les colonnes et np.sum(axis=1) pour les lignes.

- np.copy() : réalise une copie (une deep copy) d'une matrice.

Opérations sur tableaux

- La plupart de ces fonctions ont une syntaxe alternative, comme a.min() qui renvoie le minimum du tableau a. De même, on peut utiliser la version abrégée a.T pour la transposée.

- Comme le reste de Python, NumPy travaille généralement sur des références aux objets plutôt qu'avec les objets eux-mêmes. Pour faire une copie d'un tableau, on doit donc utiliser c = a.copy().

Opérateurs arithmétiques

- Les opérations suivantes sont définies pour les tableaux :
- a+b : addition
- a*b : multiplication élément par élément
- np.dot(a,c) : multiplication matricielle
- pow(a,2) : calcule les puissances des éléments d'une matrice
- pow(2,a) : calcule un nombre élevé aux puissances des éléments de la matrice

Opérateurs arithmétiques

- La soustraction de matrices - et la division terme à terme sont aussi définies.

np.where()

- np.where() est une commande de localisation très utile.
- Elle prend deux formes : x = np.where(a>2) renvoie les indices où la condition est vraie, x = np.where(a>2,0,1) renvoie une matrice de même dimensions que a contenant 0 aux indices pour lesquels la condition est vérifiée et 1 partout ailleurs.
- On peut combiner des conditions à l'aide des opérateurs logiques. Par exemple, indices = np.where((a[:,0]>3) | (a[:,1]<3)) renvoie la liste des indices pour lesquels l'une ou l'autre condition est vérifiée.

Nombres pseudo-aléatoires

- NumPy propose quelques bons générateurs de nombres aléatoires, auxquels on peut accéder à l'aide de np.random après avoir importé NumPy.

- np.random.rand(matsize) : produit un tableau de dimensions matsize dont les éléments sont des nombres uniformément distribués entre 0 et 1

- np.random.randn(matsize) : produit des nombres distribués selon une loi gaussienne standard

Nombres pseudo-aléatoires

- np.random.normal(mean,stdev,matsize) : produit des nombres pseudo-aléatoires gaussiens de moyenne et d'écart-type spécifiés

- np.random.uniform(low, high, mat size) : produit des nombres uniformément distribués entre low et high

- np.random.randint(low, high, mat size) : produit des valeurs entières pseudo-aléatoires entre low et high

Algèbre linéaire

- NumPy propose un package d'algèbre linéaire conséquent implémentant des fonctions standard.

- np.linalg.inv(a) : calcule l'inverse d'une matrice carrée a

- np.linalg.pinv(a) : calcule une pseudo-inverse, définie même si a n'est pas carrée

- np.linalg.det(a) : calcule le déterminant de a

- np.linalg.eig(a) : calcule les valeurs propres et vecteurs propres de a

Graphiques

- Le package Matplotlib, également appelé pylab, qu'on importe à l'aide de la commande import pylab as pl, propose un ensemble de fonctions graphiques.

- Celles-ci sont conçues pour ressembler le plus possible aux fonctions graphiques de Matlab.

- Les plus utilisés sont pl.plot et pl.hist.

- Parfois, les graphiques souhaités n'apparaissent pas. On doit alors soumettre la commande pl.ion() qui active les graphiques interactifs.

- La commande show() peut également être utile (pour certains IDE) pour éviter que les fenêtres graphiques ne soit fermées dès la fin de l'exécution du programme.

Exemple

```
import pylab as pl

import numpy as np

gaussian = lambda x:np.exp(-(0.5-x)**2/1.5)

x = np.arange(-2,2.5,0.01)

y = gaussian(x)

pl.ion()

pl.figure()

pl.plot(x,y)

pl.xlabel('x values')

pl.ylabel('density')

pl.title('Gaussian Density')

pl.show()
```

Exemple

- Pour plus de détails sur les fonctions de Matplotlib, on pourra consulter la page web du package.

- Par exemple la fonction pl.contour peut être utilisée après avoir utilisé la fonction np.meshgrid (analogue de outer puis contour sous R).

Exercices

- Construire un tableau a de dimensions 6 x 4 dont tous les éléments valent 2
- Construire un tableau b de dimensions 6 x 4 avec des 3 sur la diagonale et 1 partout ailleurs (ne pas utiliser de boucle)
- Peut-on multiplier ces matrices ? Pourquoi a*b fonctionne-t-il mais pas dot(a,b).
- Calculer dot(a.transpose(),b) et dot(a,b.transpose()). Pourquoi les résultats sont-ils de tailles différentes ?
- Ecrire une fonction qui imprime quelque chose à l'écran et vérifier que vous pouvez l'exécuter sous Spyder.
- Ecrire une fonction qui génère des tableaux pseudo-aléatoires et affiche leurs sommes, leurs valeurs moyennes, etc.
- Ecrire une fonction consistant en un ensemble de boucles parcourant un tableau pour déterminer le nombre de 1 qu'il contient. Faites la même chose à l'aide de la fonction where().