Université Toulouse 3,
L3 ESR et L3 Spé

Septembre 2018

Analyse numérique - TP 1

Le but de ce TP est de découvrir (re-découvrir) les bases de Python (pour ceux qui ne les connaissent pas ou les ont oubliées) et surtout les principales bibliothèques Python que nous serons amenés à utiliser pendant l'année. Il s'agit de :

1. numpy : pour la manipulation de complexes et des tableaux (pour les matrices)
2. numpy.linalg : pour effectuer des opérations d'algèbre linéaire.
3. matplotlib.pylab : pour les représentations graphiques

Dans une première section nous donnons des rappels sur l'utilisation générale de Python, vous êtes autorisés à passer rapidement dessus si vous vous sentez à l'aise. Dans une deuxième partie nous nous attachons plus particulièrement aux rappels sur ces bibliothèques (vous les avez peut-être déjà utilisées en L1 et/ou L2). Dans la dernière partie vous trouverez les premiers exercices de TP concernant le cours de cette année.

Je vous conseille vivement de vous envoyer par mail vos fichiers à la fin de chaque séance pour le cas où vous ne pourriez pas les retrouver à la séance suivante.

1. Quelques rappels généraux sur l'utilisation de Python

1.1 Opérateurs usuels

On rappelle dans ce tableau les opérateurs mathématiques classiques sous Python. Attention, je vous rappelle que Python est un langage typé...

Opérateur	**Opération**
`=`	Affectation de variable. On peut affecter plusieurs variables en une fois, en regroupant variables et valeurs à affecter dans des conteneurs. Par exemple: `a,b=1,2` ou encore `[x,y]=(0,0)`
`+`	Addition, concaténation entre itérables de même type
`*`	Multiplication, répétition d'un itérable
`-`	Négation, opposé ou soustraction pour les valeurs numériques. Différence pour les ensembles
`/`	Division
`//`	Division entière
`**`	Exposant
`%`	Modulo
`not or and`	Opérations booléennes
`< <= == != <> >= >`	Comparaison $< \leq = \neq \neq \geq >$

Opérateur	Opération
`in`	Présence d'une valeur dans un conteneur
`not in`	Absence d'une valeur dans un conteneur
`is`	Tester si une valeur est d'un type donné
`| & ^`	Opérations bits à bits sur les valeurs numériques. Union, intersection ou différence symétrique pour les ensembles

Python propose aussi des opérateurs combinés avec une affectation, dits in situ. En voici quelques-uns.

Opérateur	Opération
`x += y`	`x = x + y`
`x *= y`	`x = x * y`
`x -= y`	`x = x - y`
`x /= y`	`x = x / y`
`x //= y`	`x = x // y`
`x **= y`	`x = x ** y`
`x %= y`	`x = x % y`

1.2 Fonctions

1.2.a Fonctions de base

Les fonctions peuvent prendre plusieurs paramètres, séparés par des virgules. Certains paramètres peuvent avoir des valeurs par défaut, il est alors possible de les omettre lors de l'évaluation.

Pour consulter la rubrique d'aide relative à une fonction, on peut taper son nom suivi ou précédé d'un point d'interrogation.

Voici une liste de quelques fonctions prédéfinies.

Fonction	Usage
`all(c)` ou `any(c)`	Indique si respectivement tous ou l'un des éléments du conteneur `c` convertis en booléens valent `True`
`input(prompt = '')`	Demande à l'utilisateur d'entrer une valeur avec une invite optionnelle
`len(c)`	Renvoie la longueur (`int`) du conteneur `c`
`map(f, c)`	Renvoie le conteneur de même type que `c` contenant les valeurs de la fonction `f` appliquée aux éléments de `c`
`min(c)` ou `max(c)`	Minimum et maximum d'un conteneur dont les éléments sont comparables entre eux
`range(start = 0, stop, step = 1)`	Renvoie une liste contenant les entiers de `min` à `max` exclus par pas de longueur `step`
`round(x, n = 0)`	Renvoie l'arrondi de la valeur numérique `x` sous flottante à (au plus) `n` chiffres décimaux après la virgule

Fonction	Usage
`sum(c)`	Renvoie la somme des éléments du conteneur `c`
`type(x)`	Renvoie le type de `x`

1.2.b Définition de nouvelles fonctions

Voici un exemple de syntaxe.

```
def f(x, y=4, z=5):
    # instructions
    return valeur
```

Cette instruction stocke dans la variable `f` une fonction à trois variables `x`, `y` et `z`. Les variables `y` et `z` ont des valeurs par défaut. Le tableau suivant regroupe différentes manières d'appeler `f` ainsi que les valeurs prises par les paramètres `x`, `y` et `z`.

Appel	x	y	z
`f(1, 2, 3)`	1	2	3
`f(x=1, y=2, z=3)`	1	2	3
`f(1, 2)`	1	2	5
`f(1)`	1	4	5
`f(1, z=3)`	1	4	3

En principe, on doit s'assurer que les instructions composant le corps de la fonction mènent toujours à une instruction `return`, quels que soient les éventuels branchements conditionnels. Dans le cas contraire, on prend le risque que la fonction renvoie `None`.

La première instruction `return` rencontrée interrompt immédiatement l'exécution de la fonction. On prendra garde à ne pas confondre `print` et `return`, la première se contentant de modifier l'affichage sans renvoyer de valeur.

In [ ]:

```
def f(x):
    if x > 0:
        return x

print f(1)
print f(-1)
```

1.2.c Les fonctions Lambda

Les fonctions Lambda permettent de définir plus simplement les fonctions simples se définissant en une seule instruction. Elles peuvent avoir un ou plusieurs arguments et peuvent renvoyer n'importe quel type de données. Les exemples suivants vous montrerons simplement comment les utiliser.

In [ ]:

```
f = lambda x: x**2+1  # définit la fonction x-> x^2+1
print(f(2))

g=lambda x,y : x*y  # définit la fonction (x,y)-> x*y
print(g(2,3))

from math import sin
h=lambda x: (sin(x)**2)  # définit la fonction x-> (sin x)^2
print(h(2))

k=lambda x:(x,2*x)  # définit la fonction x-> (x,2x)
print(k(2))
a,b=k(2)  # permet de stocker le résultat de k(2) dans le couple (a,b)
print(a)
print(b)

applique=lambda u,x : u(x)  # applique la fonction u à x
print(applique(f,2))
```

1.2 Structures de contrôle

La structure syntaxique en Python est déterminée par les niveaux d'indentation du code.

Les blocs de code relatifs à une structure de contrôle (boucle ou branchement conditionnel) sont introduits par le symbole **:** à la fin de la ligne qui précède et doivent être indentés d'un niveau supplémentaire par rapport au niveau parent. Le retour au niveau d'indentation précédent signale au compilateur la fin du bloc.

En principe, on indente de quatre espaces supplémentaires à chaque niveau. Ceux-ci peuvent être obtenus facilement grâce à la touche de tabulation.

1.2.a Branchements conditionnels

La syntaxe générale est

```
if condition1:
    instructions
elif condition2:
    instructions
elif condition3:
    instructions
else:
    instructions
```

In [ ]:

```
a=-2

if a < 0:
    print 'a est negatif'
elif a>0:
    print 'a est positif'
else:
    print 'a est nul'

print(a)
```

1.2.b Boucles for (itération)

La syntaxe générale d'une boucle for est

```
for variable in iterable:
    instructions
```

In [ ]:

```
n=25
s=0
for k in range(1,n+1):
    if k%2==0:
        s=s+k
print s

# verification
pairs=[k for k in range(1,n+1) if k%2==0]
print(pairs)
so=sum(pairs)
print(so)
print(s==so)
```

1.2.c Boucles while (itération)

La syntaxe générale d'une boucle While est

```
while condition:
    instructions
```

2. Bibliothèques Python

2.1 La bibliothèque `numpy` de calcul scientifique

La bibliothèque `numpy` introduit le type `ndarray` ainsi que de nombreuses fonctions de calcul flottant notamment. Traditionnellement, on l'importe:

- soit directement dans l'environnement courant (`from numpy import *`);
- soit sous un nom abrégé (`import numpy as np`).

La bibliothèque numpy est spécialisée dans la manipulation des tableaux (array). On l'utilisera essentiellement pour les vecteurs et les matrices

• Les tableaux numpy ne gèrent que les objets de même type

• La bibliothèque numpy propose un grand nombre de routines pour un accès rapide aux données (ex. recherche, extraction), pour les manipulations diverses (ex. tri), pour les calculs (ex. calcul statistique et scientifique, calcul élément par élément, calcul matriciel)

• Les tableaux numpy sont plus performants (rapidité, gestion de la volumétrie) que les itérables usuel de Python (listes, tuples...)

• Les tableaux numpy sont sous-jacents à de nombreux packages dédiés au calcul scientifique sous Python.

• Une matrice est un tableau (array) à 2 dimensions (lignes et colonnes), un vecteur est un tableau (array) avec une seule ligne

• À la différence d'autres langages classiques, la plupart des opérateurs $(+, *, **, etc.)$ agissent élément par élément sur les tableaux.

Pour plus d'infos sur cette bibliothèque : http://docs.scipy.org/doc/numpy/reference/index.html (http://docs.scipy.org/doc/numpy/reference/index.html)

2.1.a Les vecteurs

On utilise pour créer des vecteurs numpy les matrices avec une seule ligne... Attention à une chose : la numérotation pour l'indexation se fait à partir de 0, c'est-à -dire que le premier élément est numéroté 0 et non 1.

In [ ]:

```
# Création de vecteurs :
from numpy import *

T1 = array([5, 4, 3, 2, 1, 0])
print(T1)
print(type(T1)) # type ndarray
print(len(T1))
```

In [ ]:

```
# Indexation, accès
T1 = array([5, 4, 3, 2, 3, 0])
print(T1[2])  # attention l'indice 2 correspond au 3eme élément (num à partir de 0)
print(T1[:2]) #accès aux éléments 0 et 1 (2 est exclu)
print(T1[T1])
print(T1>2) # crée la liste des booléens T1[k]>2
print(T1[T1 > 2]) # extrait de T1 ses éléments strict. plus grands que 2
T1[T1 > 2] *= -1 # multiplie par -1 les éléments strict. plus grands que 2 du vecteu
print(T1)
```

In [ ]:

```
# Création de vecteurs de valeurs particuliers :
#suite arithmétique
a=arange(1,10)
b=arange(start=10,stop=0,step=-2)
c=arange(10)
print(a)
print(b)
print(c)


#subdivision uniforme de [a,b] de pas h ATTENTION b n'est pas inclus !!
c=arange(0,1,0.1)
d=arange(start=0.2,stop=1.2,step=0.3)
print(c)
print(d)
c=arange(0,1,0.1)
print(c)
```

In [ ]:

```
# subdivision uniforme de [a,b] à n éléments
# ATTENTION b EST INCLUS CETTE FOIS !!!!
a=linspace(0,10,5)
print(a)
```

In [ ]:

```
# vecteur de 1...
a=ones(5)
print(a)
b=full(5,3.2) # crée un vecteur de 5 éléments égaux à 3.2
print(b)
```

In [ ]:

```
# ATTENTION Duplication !!!
t1 = array([1, 2, 3, 4])
t2 = t1[1:3]    # Pas de duplication, toute modification de t2 aura un effet sur t1
print(t1)
print(t2)
t2[0] = 5
print(t2)
print(t1)
```

In [ ]:

```
t1 = array([1, 2, 3, 4])
t2=copy(t1)
t2 = t1[1:3]    # Pas de duplication, toute modification de t2 aura un effet sur t1
print(t1)
print(t2)
t2[0] = 5
print(t2)
print(t1)
```

2.1.b Les matrices

In [ ]:

```
T2 = array([[0, 1],    # Il est possible de passer à la ligne avant de fermer des c
            [2, 3],    # ça permet de rendre le code plus lisible
            [4, 5]])
print(T2)
T3 = array([[0, 1],[2, 3],[4, 5]])
print(T3)
print(type(T2))
print(len(T2))
print(T2[0])
print(T2[2, 1])
print(T2[1, :])
# demander la taille de la matrice
print(T2.shape)
```

La bibliothèque `numpy` fournit des fonctions pour créer et gérer facilement des tableaux à plusieurs dimensions.

Fonction	**Usage**
`arange(start=0, stop, step=1)`	Similaire à `range`, mais renvoie un tableau et les arguments peuvent être de type flottant
`linspace(start, stop, n=50)`	Renvoie un tableau à `n` valeurs uniformément réparties entre `start` et `stop` (bornes incluses)
`array(c)`	Crée un tableau à partir du conteneur `c`. Le type des éléments peut aussi être choisi.
`zeros(shape)` `ones(shape)`	Crée un tableau de la forme désirée, rempli de zéros ou de uns. Le type des éléments peut aussi être choisi.
`identity(n)`	Crée une matrice identité de taille $n \times n$
`diag(A)`	Diagonale d'une matrice (si `A` est bidimensionnel) ou matrice diagonale (si `A` est monodimensionnel)
`dot(A, B)`	Produit matriciel
`reshape(T, shape)`	Renvoie le tableau obtenu en réarrangeant le tableau selon une nouvelle forme
`concatenate(l)`	"Concatène" une liste de tableaux entre eux
`roll(a, shift)`	Effectue une permutation circulaire des éléments d'un tableau en décalant les éléments vers la droite (les derniers reviennent à gauche)
`meshgrid(range(m), range(n))`	Renvoie deux matrices $m \times n$, la première contenant les indices des lignes, la seconde ceux des colonnes. Utile pour créer une matrice à partir de son terme général
`fromfunction(f, shape)`	Crée un tableau à partir d'une fonction
`fromfile(filename, dtype)`	Création depuis un fichier. L'inconvénient par rapport à `tofile` est qu'il faut préciser le type des éléments (`dtype`) et la forme est perdue (le tableau obtenu n'aura qu'une seule dimension, comme s'il avait subi `flatten()`)

Attention, la produit matriciel ou les puissances des matrices ne s'obtiennent pas avec les multiplication et l'exponenciation classique (ces opérations se fonct élément par élément).

In [ ]:

```
A = identity(2, int)
B = array([[1, 2], [3, 4]])
print(A)
print(B)
print(A * B)
print(dot(A, B))
```

2.1.c Fonctions introduites par `numpy`

La bibliothèque `numpy` introduit ou redéfinit un grand nombre de fonctions numériques. Celles-ci peuvent agir sur des valeurs numériques individuelles, ou sur des tableaux (ou listes) élément par élément. En voici quelques-unes.

Fonction	**Usage**
`numpy.abs` `conj` `real` `imag` `angle`	Module, conjugué, partie réelle, partie imaginaire, argument
`cos` `sin` `tan` `sinh` `cosh` `tanh`	
`arccos` `arcsin` `arctan` `arcsinh` `arccosh` `arctanh`	
`exp` `log`	
`numpy.round` `floor` `ceil`	Arrondi au plus proche, arrondi inférieur, arrondi supérieur

Elle introduit également de nombreuses fonctions spécifiques aux tableaux. En voici une petite liste.

Fonction	**Usage**
`sum` `prod`	Somme et produit des éléments
`numpy.min` `numpy.max` `mean` `median` `std`	Minimum, maximum, moyenne, médiane, écart-type
`cumprod` `cumsum`	Produit ou somme cumulés
`argmin` `argmax` `argsort`	Indice du minimum, du maximum, tableau d'indices permettant de trier le tableau
`convolve`	Convolution discrète

Mentionnons encore quelques constantes utiles: `pi`, `e`, `inf`, `nan`, etc.

Attention: importer `numpy` dans l'espace de base sans préfixe (`from numpy import *`) ne remplace pas les fonctions prédéfinies de Python (`min`, `max`, `abs`...). On devra donc les préfixer si on souhaite utiliser la version `numpy`. Cela n'a en général pas d'importance, sauf pour `min`, `max` et `round`.

In [ ]:

```
t = array([[1, 2, 3, 4],[2,3,1,5]])

print(numpy.max(t))
print(t.max())
```

2.1.d Fonctions introduites par `numpy.linalg`

Fonction	Usage
`dot(A, B)`	Produit matriciel
`matrix_power(M, n)`	Exposant matriciel
`norm(x)`	Norme (euclidienne par défaut) d'un vecteur
`det(A)` `tr(A)`	Déterminant et trace
`eig(A)` `eigvals(A)`	Vecteurs et valeurs propres
`matrix_rank(A)`	Rang d'une matrice
`solve(A, b)`	Résolution du système linéaire $Ax = b$
`inv(A)`	Inversion matricielle
`cholesky(A)`	Décomposition de Cholesky
`qr(A)`	Décomposition QR

D'autres fonctions sont décrites sur http://docs.scipy.org/doc/numpy/reference/routines.linalg.html (http://docs.scipy.org/doc/numpy/reference/routines.linalg.html).

In [ ]:

```
from numpy.linalg import *

A = array([[2, 3, 5, 7],
           [11, 13, 17, 19],
           [23, 29, 31, 37],
           [41, 43, 47, 53]])

print(det(A),trace(A))
eigenvalues, eigenvectors = eig(A)
print(eigenvalues)
print(eigenvectors)
# On calcule inv(P).A.P
print(numpy.round(dot(dot(inv(eigenvectors), A), eigenvectors), 10))
v = solve(A, [1, 2, 3, 4])
print(dot(A, v))
print(v)
```

2.2 La bibliothèque `Matplotlib.pyplot`

La bibliothèque `matplotlib` (et sa sous-bibliothèque pyplot) sert essentiellement à afficher des graphismes. Son utilisation ressemble beaucoup à celle de Matlab. Traditionnellement, on l'importe:

- soit directement dans l'environnement courant (`from matplotlib.pyplot import *`);
- soit sous un nom abrégé (`import matplotlib.pyplot as plt`).

Dans ce qui suit, on suppose qu'on utilise le backend `inline`, propre aux notebooks. Si on utilise un autre backend, il pourra être nécessaire d'ajouter `show()` à la fin des commandes pour faire apparaitre la fenêtre qui contient le dessin.

Voici une courte liste de commandes possibles.

Commande	**Effet**
plot(y)	Affiche les points dont les ordonnées sont dans `y`
plot(x, y)	Affiche les points de coordonnées (x_k, y_k)
matshow(a)	Affiche une matrice
imshow(a)	Idem, mais avec plus d'options (et la possibilité de faire des sous-figures)
hist(y, bins=10)	Affiche un histogramme de la répartition des valeurs des y_k. On peut choisir le nombre de barres, voire leurs bornes
figure()	Commence une nouvelle figure (si on ne l'a pas fait, cette fonction est appelée implicitement dès qu'on commence à afficher quelque chose)
figure(figsize=(a,b))	Idem, mais permet d'imposer une taille (en pouces) à la figure
subplot(lines, columns, index)	Indique qu'on va commencer une sous-figure
suptitle("titre")	Afficher un titre au-dessus de la figure. Attention aux caractères accentués (précéder la chaîne du préfixe `u`). Il est possible de mettre du LaTeX entre dollars `$`
legend()	Afficher les légendes (s'il y en a)
axis([xmin, xmax, ymin, ymax])	Définit explicitement la taille des axes. Par défaut, ceux-ci s'adaptent automatiquement à la taille du contenu de la figure
axis("scaled")	Indique que le repère doit être orthonormé (pas de distorsion à l'affichage). À utiliser **après** `plot(...)` car son utilisation désactive l'ajustement automatique des axes
axis("off")	Ne pas afficher les axes
loglog()	Passer en échelle log-log

La fonction `plot` (la plus utilisée) peut prendre une grande variété de paramètres supplémentaires pour contrôler l'aspect des courbes tracées. Par défaut, les points sont reliés par des segments. Voici quelques paramètres optionnels possibles:

- `linestyle=s`, avec `s` qui vaut par exemple
  - `"solid"` ou `"-"` (lignes pleines),
  - `"dashed"` ou `"--"` (tirets),
  - `"dotted"` ou `":"` (pointillés),
  - `""` (pas de lignes);
- `color=c`, avec `c` qui vaut par exemple `"red"`, `"green"`, `"blue"` etc.
- `marker=m`, avec `m` qui vaut par exemple
  - `""` (pas de marqueur);
  - `","` (petits points),
  - `"."` (gros points),
  - `"+"` ou `"x"` (croix),
  - `"*"` (étoiles),
  - `"o"` (cercles),
  - `"s"` (carrés);
- `markersize=s` pour contrôler la taille des marqueurs;
- `legend=l` où `l` est une chaîne de caractères pour donner un titre à la courbe. Celui-ci n'apparaitra que si l'on utilise `legend()` par la suite. Mêmes remarques que pour `suptitle`.

On peut aussi utiliser l'écriture `plot(x, y, "style")` où `style` est donné sous forme abrégée, par exemple:

- `"y-x"` pour des croix reliées par des lignes jaunes;
- `"g:*"` pour des étoiles reliées par des pointillés verts;
- `"bo"` pour des cercles bleus sans lignes;
- `"c-"` pour des lignes cyans sans marqueur.

Une vaste liste de fonctions supplémentaires et d'exemples illustrés peut se trouver à la page http://matplotlib.org/index.html (http://matplotlib.org/index.html)

In [ ]:

```
from matplotlib.pyplot import *
%pylab inline
# permet d'importer numpy et matplotlib.pyplot

plot([0,1,-1,2,-2,3,-3])
```

In [ ]:

```
%pylab inline
# permet d'importer numpy et matplotlib.pyplot

plot([0,1,-1,2,-2,3,-3])
```

In [ ]:

```
# Représentation d'une suite

n = arange(50)
u = (-1) ** n * exp(-n / 20.)
plot(u, "r+")
```

In [ ]:

```
# Représentation d'un graphe de fonction

x = linspace(-5 * pi, 5 * pi, 200)
y = sin(x) / x
plot(x, y, "g", label='$\\frac{\\sin x}{x}$')    # Ici il faut échapper le caractère
legend()
```

In [ ]:

```
# Représentation d'une courbe paramétrée

t = linspace(0, 2 * pi, 500)
x = cos(t)
y = sin(t) + sqrt(abs(cos(t)))
plot(x, y)
```

In [ ]:

```
# Echelle logarithmique

figure(figsize=(13,5))

t = linspace(0, 2, 100)
x = 2 ** t
y = 1 / x ** 3

subplot(1, 2, 1)
title('Echelle lineaire')    # Pour les chaîne avec caractères accentués, mettre un
plot(x, y)

subplot(1, 2, 2)
title('Echelle logaritmique')
plot(log(x), log(y))
```

3. Et maintenant à vous de 'jouer'

Exercice 1 (Petits systèmes, conditionnement et stabilité)

On considère les sytèmes linéaires perturbés suivants, étudiés en TD:

$$A = \begin{pmatrix} 7 & 10 \\ 5 & 7 \end{pmatrix}, \quad Y_1 = \begin{pmatrix} 10 \\ 7 \end{pmatrix} \quad \text{et} \quad Y_2 = \begin{pmatrix} 10,1 \\ 6,9 \end{pmatrix}.$$

Résoudre les équations $AX = Y1$ et $AX = Y2$ à l'aide de la commande solve de la librairie `numpy.linalg`. Calculer le conditionnement de A et conclure.

Exercice 2 (Normes subordonnées)

On considère la matrice $A = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}$. En utilisant la commande `norm`, calculer ses normes $\|\|_1$, $\|\|_2$ et $\|\|_\infty$ et retrouver en calculant les membres de droite (de manière la plus concise possible à partir des fonctions pré-programmée de Python) les formules suivantes vues en cours et TD:

- $\|A\|_\infty = \max_{i\in\{1,\dots,N\}} \sum_{j=1}^{N} |a_{i,j}|$
- $\|A\|_1 = \max_{j\in\{1,\dots,N\}} \sum_{i=1}^{N} |a_{i,j}|$
- $\|A\|_2 = (\rho(A^T A))^{\frac{1}{2}}$

In [ ]:

```
? norm
```