SujetTP1

March 25, 2021

1 TP1 : Prise en main de Python

Ce premier TP consiste à prendre en main python pour réaliser quelques travaux numériques pour le traitement du signal. Pour les quelques éléments de bases concernant Python, ce notebook est très fortement adapté de celui de David Gontier : https://www.ceremade.dauphine.fr/~gontier/enseignement.html.

1.1 Introduction aux notebook Jupiter

Pour lancer Jupyter, il faut, dans un terminal, lancer la commande

```
jupyter notebook
```

Ceci ouvrira une page dans un navigateur web dans laquelle on pourra travailler avec Python.

Les notebooks sont composés de cellules contenant du code (en python) ou du texte (simple ou formaté avec les balisages Markdown). Ces notebooks permettent de faire des calculs interactifs en Python, et constituent un outil de choix pour l'enseignement.

On peut éditer une cellule en double-cliquant dessus, et l'évaluer en tapant **Ctrl+Entrée** (on utilisera aussi souvent **Maj.+Entrée** pour évaluer et passer à la cellule suivante). Les boutons dans la barre d'outil vous seront très utiles, survolez-les pour faire apparaître une infobulle si leur pictogramme n'est pas assez clair. N'oubliez pas de sauvegarder de temps en temps votre travail, même si Jupyter fait des sauvegardes automatiques régulières.

L'apprentissage de l'interface se fait petit à petit par l'exploration, pour apprendre, il y a l'aide, ou des tonnes de resources sur internet.

Remarque : *L'utilisation de Jupyter est proposée pour simplifier l'utilisation de Python par l'utilisation de feuilles de calculs :* les notebooks. *Bien évidemment, celles et ceux qui voudront utiliser un autre outil, coder simplement avec Python dans un fichier texte avec son éditeur favoris, etc., peuvent tout à fait le faire.*

1.1.1 Cellule Markdown

Comme nous l'avons dit, les cellules sont soit du code Python, soit du texte qui peut être formaté avec les balisages Markdown.

Le Markdown est un format de texte qui accepte un minimum de formatage, il permet de rapidement : * faire des listes ; * faire des *italiques*, du **gras**, etc. ; * faire des liens liens ; * écrire des formules mathématiques en LaTeX.

1.2 Introduction expéditive à Python

En python, le commentaire se fait avec le caractère `#`, et l'évaluation d'une cellule python se fait avec **Maj.+Entrée**.

[1]:
```
# Ceci est une cellule Python où l'on a ajouté des commentaires qui permettent,
# en plus des cellules textes du notebook Jupyter, d'expliquer certaines␣
 ↪parties
# du code. Il est encouragé de bien commenter ses codes, surtout quand ceux-ci
# ne sont pas dans un notebook
a = 4
print("a=",a, "et 2*a=",2*a)
```

```
a= 4 et 2*a= 8
```

Pour obtenir de l'aide sur une fonction, il suffit d'exécuter une cellule python avec le nom de la fonction python suivi du caractère `?`.

[2]:
```
str?
```

1.2.1 Bibliothèques de calcul scientifique

En première cellule de nos notebooks, on chargera les librairies dont nous avons besoin, en particulier `numpy` que l'on va présenter très rapidement dans la suite.

[3]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

Pour le calcul scientifique, dans lequel s'insère le traitement numérique du signal, nous ferons appel à plusieurs bilbiothèques python. Parmis celles-ci, il y a la très connu numpy qui nous sera très utile.

1.2.2 Structure en python

Le code python est structuré par l'*indentation*. C'est elle qui définit les groupes de ligne de code. **Attention :** en python, et comme beaucoup de langage de programmation, les indices commencent à zéro.

[4]:
```
N = 10
print("J'ai", N, "morceaux de chocolat")

for i in range(N): # range(N) produit la liste [0,1,...,N-1]
    # Boucle for, l'indentation commence après les deux points
    if i < 2:
        #Boucle if. De nouveau les deux points
        print("Si je mange", i, "morceau de chocolat,")
    else:
        #else, au même niveau d'indentation que if
        print("Si je mange", i, "morceaux de chocolat,") # Pluriel !!!
```

```
    # On revient à l'indentation précédente, on est encore dans la boucle for
    print("  il en reste", N-i)

# On revient à l'indentation précédente, on quitte la boucle for
print("On est sorti de la boucle for")
```

```
J'ai 10 morceaux de chocolat
Si je mange 0 morceau de chocolat,
  il en reste 10
Si je mange 1 morceau de chocolat,
  il en reste 9
Si je mange 2 morceaux de chocolat,
  il en reste 8
Si je mange 3 morceaux de chocolat,
  il en reste 7
Si je mange 4 morceaux de chocolat,
  il en reste 6
Si je mange 5 morceaux de chocolat,
  il en reste 5
Si je mange 6 morceaux de chocolat,
  il en reste 4
Si je mange 7 morceaux de chocolat,
  il en reste 3
Si je mange 8 morceaux de chocolat,
  il en reste 2
Si je mange 9 morceaux de chocolat,
  il en reste 1
On est sorti de la boucle for
```

Exercice : Calculer la somme

$$\sum_{n=1}^{N} \frac{1}{n^2}$$

pour $N = 10, 100, 1000, 10000$. Comparer avec la valeur exacte. On fera attention que `range(N)` produit la liste `[0,1,...,N-1]` et que la puissance se cole `**`.

1.2.3 Les types de variables

[5]:
```
#Entier
a = 4
print("a = ", a, "\t\t\t\t est de type", type(a))

# Floatant
a = 3.5
print("a = ", a, "\t\t\t est de type", type(a))
a = 1e7 # Raccourci pour 10^7, --> float
print("a = ", a, "\t\t est de type", type(a))
a = np.pi # appel à numpy pour la constante pi
```

```
print("a = ", a, "\t\t est de type", type(a))

# Booleen
a = True
print("a = ", a, "\t\t\t est de type", type(a))

# String
a = "Hello World!"
print("a = ", a, "\t\t est de type", type(a))

# Listes (taille variable)
a = [1,2,3]
print("a = ", a, "\t\t\t est de type", type(a))

# Tuples (taille fixe)
a = (1.5, [1,2], "coucou")
print("a = ", a, "\t est de type", type(a))
```

```
a =  4                          est de type <class 'int'>
a =  3.5                        est de type <class 'float'>
a =  10000000.0                 est de type <class 'float'>
a =  3.141592653589793          est de type <class 'float'>
a =  True                       est de type <class 'bool'>
a =  Hello World!               est de type <class 'str'>
a =  [1, 2, 3]                  est de type <class 'list'>
a =  (1.5, [1, 2], 'coucou')    est de type <class 'tuple'>
```

En plus de ces types de base, nous travaillons avec la librairie numpy, qui introduit le type array (= matrice)

[6]:
```
# Array (matrices)
a = np.array([[1,2], [3,4]])
print("a = \n", a, "\t\t\t est de type", type(a))
```

```
a =
 [[1 2]
 [3 4]]                         est de type <class 'numpy.ndarray'>
```

1.2.4 Opérations de bases

Python connait les opérations +,-,/,*, et plein d'autres encore. Voici quelques astuces et pièges. Attention pour les utilisateurs et utilisatrices de python 2, quelques opérations peuvent être un peu différentes.

[7]:
```
print("sqrt{2} = ", 2**.5,"ou", np.sqrt(2)) #
print("17 modulo 3 = ", 17%3) # Avec %
print("3/2 = ", 3/2, "est un float")

a = 4
```

```
print("a = ", a)
a += 2 #Raccourci pour a = a+2 (marche aussi avec *=, /=, etc.)
print("a = ", a)
```

```
sqrt{2} =  1.4142135623730951 ou 1.4142135623730951
17 modulo 3 =  2
3/2 =  1.5 est un float
a =  4
a =  6
```

1.2.5 Listes

Le code en Python se veut très lisible, voici les choses à connaître :

[8]:
```
#La liste vide
L = []
print("La liste vide : L =", L, "\n")

# range(n) ou range(0, n) renvoie la liste (0, 1, ..., n-1) sous forme␣
 ↪"compacte"
print("range(5) = ", range(5))
print("mais aussi")
print("range(5) =", [i for i in range(5)], "\n")

# Création de liste par compréhension (plus lisible ET plus rapide !)
puissanceDe2 = [2**n for n in range(0,11)]
print("Les puissances de 2 sont", puissanceDe2)

# Ajouter un élément à une liste avec .append()
puissanceDe2.append(2**11)
print("Les puissances de 2 sont", puissanceDe2, "\n")

# Connaitre la taille de la liste avec len
print("Il y a", len(puissanceDe2),"éléments dans cette liste.")
```

```
La liste vide : L = []

range(5) =  range(0, 5)
mais aussi
range(5) = [0, 1, 2, 3, 4]

Les puissances de 2 sont [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
Les puissances de 2 sont [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048]

Il y a 12 éléments dans cette liste.
```

On accède à un élément d'une liste avec des crochets.

Remarque : En Python, * les parenthèses servent à évaluer une fonction ; * les crochets servent à accéder à une mémoire.

On rappele que les indices commencent à 0 en Python.

[9]:
```
L = [1,2,3,4,5]
print("L =", L)
print("Le premier élément est", L[0])
print("Le dernier élément est", L[-1]) #raccourci pour le dernier élément !

# On peut aussi accéder à une sous liste
print("L contient la sous liste", L[1:4]) # Ici, [1:4] est un raccourci pour␣
 ↪range(1,4), et vaut [1, 2, 3]
```

```
L = [1, 2, 3, 4, 5]
Le premier élément est 1
Le dernier élément est 5
L contient la sous liste [2, 3, 4]
```

1.2.6 Les matrices

Nous attaquons maintenant le calcul matriciel, les liste de Python ne conviennent pas (cf exemple suivant). Le bon type à utiliser est l'array. Celui-ci se trouve dans la librarie numpy, qu'on a déjà chargée.

[10]:
```
L = [1,2,3]
Larray = np.array(L)
print("L =", L, "\t\t\t\t Larray =", Larray)
print("2*L =", 2*L, "\t\t 2*Larray =", 2*Larray)
```

```
L = [1, 2, 3]                          Larray = [1 2 3]
2*L = [1, 2, 3, 1, 2, 3]               2*Larray = [2 4 6]
```

Comme pour les listes, on accède aux éléments avec des crochets. Cependant, pour une matrice, il faut maintenant 2 paires de crochets !

[11]:
```
A = np.array([[1, 2, 3], [4, 5, 6], [7,8,9]])
print("A = \n", A)
print("L'élément au milieu du bas est", A[2,1]) # de la forme A[i,j]
```

```
A =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
L'élément au milieu du bas est 8
```

Exercice : Comment récupérer la sous-matrice [4,5], [7,8] à partir de A ?

Les matrices de bases Numpy définit un certain nombre de matrices de référence. Nous utiliserons souvent les matrices suivantes.

[12]:
```
# La matrice nulle prend des dimensions en entrée :
A = np.zeros([2,3]) #de taille 2 x 3.
```

```
print ("A = \n", A)

# La matrice avec que des 1 :
B = np.ones([2,3])
print ("\nB = \n", B)

# La matrice identité est carrée et ne prend qu'un seul paramètre :
A = np.eye(3)
print("\n A = \n", A)
```

```
A =
 [[0. 0. 0.]
 [0. 0. 0.]]

B =
 [[1. 1. 1.]
 [1. 1. 1.]]

 A =
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Le calcul matriciel **ATTENTION**, le produit de base de numpy est le produit terme à terme, alors que le produit matriciel se fait avec dot.

[13]:
```
x = np.array([1,2,3])
y = np.array([4,5,6])
print ("x =", x, "y =", y, ", x*y =", x*y, "et dot(x,y) =", np.dot(x,y))
```

```
x = [1 2 3] y = [4 5 6] , x*y = [ 4 10 18] et dot(x,y) = 32
```

Python connait les fonctions “usuelles”, à savoir déterminant, trace, norm, inv, … On peut aussi résoudre des systèmes avec solve.

[14]:
```
A = np.array([[(i-j)**2 for j in range(3)] for i in range(3)])
b = np.array([0,0,1])
x = np.linalg.solve(A,b) # La solution de Ax = b

print("On a bien | Ax - b | = ", np.linalg.norm(np.dot(A,x) - b))
```

```
On a bien | Ax - b | =  0.0
```

Exercice : Que vaut exp(eye(3)) ? Comment créer l’exponentielle de eye(3) ?

1.3 Les fonctions

Avec Python, on peut créer des fonctions. La structure est la suivante :

```
def nomfonction(<parametre(s)>):
    # ce que fait ma fonction
```

```
    instruction 1
    instruction 2
    # la portée de la fonction étant définie par le groupe indenté
```

La syntaxe se comprend assez bien des exemples.

[15]:
```
def somme(a,b):
    return a+b
```

[16]:
```
somme(4,5)
```

[16]: 9

[17]:
```
# fonction qui retourne une phrase, donc une chaine de caractères
def EcrireSomme(a,b):
    return "La somme "+str(a)+"+"+str(b)+" est égale à "+str(a+b)

print(EcrireSomme(4,5))
```

```
La somme 4+5 est égale à 9
```

Exercice : Créer une fonction qui retourne une liste composée des éléments de la suite de Fibonacci jusqu'à un rang N donné en argument. La fonction s'invoquera de la manière suivante `Fibonacci(5)`. On utilisera une boucle `for`.

1.4 Internet

Évidemment, ce qui est présenté dans ce notebook n'est qu'une bien trop succinte introduction aux concepts de bases qui nous seront utiles. Les documentations sur internet sont pléthores et il est plus que nécessaire d'aller chercher par soi même.

2 Un peu de traitement du signal

2.1 Transformée de Fourier et théorème de Shannon

Nous allons commencer sur un exemple simple. Pour cela, générons un signal sinusoïdal, représentons le, puis étudions sont spectre dans le domaine de Fourier.

Remarque : Ici, puisqu'il s'agit de travaux pratiques informatiques, nous ne pouvons travailler qu'avec des signaux déjà échantillonnés et quantifiés.

Exercice : On veut générer la deux listes t_n et $s_n = \sin(2\pi f_0 t_n)$ pour $n \in [0, \dots, N-1]$ (t_n est la discrétisation de l'intervalle $[0, 1]$). 1. Quelle est la fréquence d'échantillonnage ? 2. En utilisant la fonction `np.arange` générez la suite (de type `np.ndarray`) (t_n) et gérérez la suite (s_n) en utilisant la fonction `np.sin` (qui peut prendre en argument une liste). On définira une variable `fo` égale à 3 et une variable `N` égale à 1000.

Pour comprendre un peu mieux ce que l'on fait, nous allons représenter notre signal. Pour cela on utilisera la librairie `matplotlib`.

[18]:
```
import matplotlib.pyplot as plt
```

On peut alors faire appel aux outils de tracés. Pour tracer une liste `y` en fonction d'une autre de même taille `x`, avec légende, il suffit d'exécuter :

```
plt.figure() # pour générer la figure
plt.plot(x,y)
plt.title('$y$ en fonction de $x$')
plt.xlabel("Abscisse")
plt.ylabel("Ordonnées")
```

Exercice : Tracer le signal défini précedemment.

Exercice : En utilisant la fonction de de transformée de fourier rapide (FFT) de `numpy` (`np.fft.fft`), calculer puis représentez le module de la transformée de Fourier du signal discrétisé (s_n). On utilisera la fonction `np.fft.fftfreq(<nbr points>,<time step>)` pour générer la liste des fréquences pour pouvoir tracer la FFT. Voir https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html#numpy.fft.fftfreq et https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html#numpy.fft.fft. Retrouve-t-on la fréquence du signal ?

Exercice : On dispose maintenant d'un signal sauvé dans le fichier `signal.npz` à télécharger ici, que l'on chargera sous forme d'un `np.ndarray` avec les lignes suivantes :

```
signal=np.load('signal.npz') #
```

Ce fichier contient les valeurs du signal temporel nommé en variable `x` que l'on obtiendra grâce à

```
s=signal['x'].flatten() #
```

Ce signal est échantillonné à la fréquence $F_e = 32$. 1. Quel est le pas de temps T_e entre deux échantillons ? 2. Construire la suite (t_n) de la même taille que le signal correspondant au temps. Tracer le signal. 3. Faite l'analyse spectrale grâce à la FFT, on utilisera un nombre 8 fois supérieur à la taille du signal temporel pour le calcul de la FFT.

Exercice : Créez un nouveau signal en répertant le signal lu 8 fois (fonction `np.tile`). Faites-en ensuite l'analyse spectrale. On n'hésitera pas à zomer sur le spectre pour bien voir se qu'il se passe (`plt.xlim`), on le comparera au spectre du signal non périodisé en le superposant lors du tracé : il suffit de faire plusieurs `plt.plot` sans ouvrir de nouvelle `plt.figure`. On pourra utiliser la fonction `plt.stem` au lieu de `plt.plot` pour le spectre du signal périodisé.

Remarque : *il y a plusieurs définition de la TFD, et l'amplitude peut être normalisé ou non. Il est possible qu'il faille pour bien comparer les deux spectres, renormaliser un des deux spectre.*

Nous allons maintenant sous échantillonner notre signal et voir ce qu'il se produit sur son spectre.

Exercice : 1. Soit (s_k), $k = 0, \dots, N-1$, un signal discrétisé avec N pair. À partir de la définition de la transformée de Fourier discrète :

$$\hat{s}_k = \sum_{n=0}^{N-1} s_n \mathrm{e}^{-2\mathrm{i}\pi \frac{n}{N}},$$

montrer que la transformée de Fourier du signal (s'_n), défini par $s'_k = s_{2k}$ pour tout $k = 0, \dots, (N-2)/2$, on a

$$\forall k = 0, \dots, (N-2)/2, \quad \hat{s}'_k = \sum_{n=0}^{(N-2)/2} s_{2n} \mathrm{e}^{-2\mathrm{i}\pi \frac{2n}{N}}.$$

2. Construire des signaux échantillonnés à 16, 8 et 4Hz à partir du signal de départ (on pourra utiliser une boucle). On fera pour cela une fonction python qui échantillonne : d'après la question 1, tout en gardant la même taille de signal, on peut extraire seulement certaines valeurs en complétant les autres par zéro. 3. Observer ce qu'il se produit sur le spectre. Que constate-t-on ?

2.2 Le phénomène de Gibbs

On se propose de regarder numériquement l'approche par série de Fourier d'un signal rectangulaire périodique.

Exercice : Définissez un fonction python qui code un signal $s(\cdot)$ carré périodique de période $T = 2$ et d'amplitude $A = 3$, c'est-à-dire que pour $t \in [0, T/2[$, $s(t) = A$ et pour $t \in [T/2, T[$, $s(t) = -A$.

On pourra mettre ces valeurs en paramètres optionels de la fonction. Celle-ci pourra avoir la forme suivante :

```
def carre(t,A=3,T=2):
    # instruction
    return valeur
```

On pourra utiliser le modulo pour gérer la périodicité de la fonction : pour deux réels a et b, le modulo en python s'écrit `a%b`.

Exercice : Représentez cette fonction. On propose de compléter le code suivant :

```
N=100
t=np.arange(N)*4/N
liste = []
for i in t:
    # à compléter
```

en utilisant la fonction `.append` et la transformation de liste en `np,ndarray`.

Exercice : Montrez que la série de Fourier de s est

$$s(t) = \sum_{k=0}^{+\infty} \frac{-4A}{(2k+1)\pi} \sin\left((2k+1)t\frac{2\pi}{T}\right),$$

puis pour différentes valeurs de plus en plus élevées de N tracer les fonctions

$$s_N(t) = \sum_{k=0}^{N} \frac{-4A}{(2k+1)\pi} \sin\left((2k+1)t\frac{2\pi}{T}\right).$$

Qu'observe-t-on ?

Exercice : 1. Implémentez une fonction qui calcule numériquement la décomposition en série de Fourier (jusqu'à au rang N fixé) d'une fonction réelle périodique quelconque. On utilisera les fonction d'intégration numérique de `numpy`. 2. Vérifiez sur le cas de la fonction créneau que votre fonction est correcte.

[ ]:

[ ]: