TRAVAUX PRATIQUES DE GENIE INFORMATIQUE

1 • BUT DE LA SEANCE.

Cette première séance a pour but de vous faire découvrir les principes de base de la programmation sous Python. Cette découverte est dirigée mais nous vous conseillons de largement utiliser les modules d'aide de Python et les informations que vous pourrez trouver sur internet. Le but de ces séances de travaux pratiques est de vous rendre autonomes, donc tout le temps que vous passerez à découvrir les fonctionnalités de ce logiciel est du temps de gagné.

1.1 – Présentation de l'environnement de programmation.

Il y a plusieurs façons de "programmer" avec le langage Python :

- Dans la console
- Programmation avec fichier texte exécuté dans la console Python
- Programmation avec des environnements intégrés (IDE) : pour ces TP nous avons choisi d'utiliser l'IDE Spyder téléchargeable sur https://www.spyder-ide.org/. C'est un des environnements que vous avez sur les machines à la FDS.

Par défaut l'interface Spyder comporte 3 fenêtres :

- La fenêtre de script où vous tapez votre code (que vous pouvez exécuter en cliquant sur le triangle vert dans la barre de menu)
  Attention pour voir le résultat il faut sélection la fenêtre « IPython Console » à la place de « Terminal » en bas à droite.
- La fenêtre des variables
- Une fenêtre de commande pour le terminal (« Terminal ») ou la console Python (" IPython Console ")

Vous pouvez changer de fenêtre en cliquant sur les différends onglets

Pour plus d'information sur l'interface Spyder, vous pouvez aller sur le site officiel de Spyder.

Pour lancer Spyder sur les ordinateurs de l'Université, il suffit d'ouvrir une console (terminal) et de taper « spyder ». La fenêtre d'environnement spyder apparaitra rapidement.

1.2 - Algèbre linéaire

Durant ces TP nous allons utiliser des représentations matricielles et les opérations d'algèbre linéaire. Nous vous conseillons donc d'effectuer une petite remise à niveau dans ce domaine à l'aide du document de révision joint à ce TP.

2 • DECOUVERTE DE PYTHON (et l'IDE Spyder).

2.1 - Principe.

Le principe de l'ensemble de ce module est de vous rendre autonome sur l'utilisation d'un logiciel de calcul. Nous vous conseillons donc de solliciter votre curiosité et d'apprendre à aller chercher les informations dont vous avez besoin (sur internet ou autre).

2.2 - Quelques principes de base.

Après avoir exécuté le logiciel Spyder, vous devez être dans votre répertoire personnel. Nous vous demandons de créer votre arborescence pour ces TP en allant dans les préférences (Spyder, Préférences) et en spécifiant votre répertoire de travail dans " Working Directory ". Merci de créer un répertoire par séance.

Extension des capacités

Un peu comme C par exemple, Python repose sur l'importation de bibliothèques pour ajouter de nouvelles capacités. Vous trouverez ci-dessous la liste des bibliothèques dont nous aurons besoin. À vous de les importer quand elles seront nécessaires.

- Les opérations mathématiques : `import math as m`
- Les graphiques : `import matplotlib.pyplot as plt`
- Le calcul numérique (dont les tableaux/matrices) : `import numpy as np`
- Les nombres aléatoires : `import random as rand`

"`as m`", "`as plt`", "`as np`" et "`as rand`" permet d'utiliser les abréviations `m`, `plt`, `np` et `rand` devant les formules à la place du nom complet de la bibliothèque, comme nous le verrons par la suite)

Cela permet aussi d'avoir une aide en ligne : après avoir importé la bibliothèque mathématique (import math) essayez :

```
>> help(m)
```

Rm : il est possible de n'importer qu'une partie des éléments d'une bibliothèque ex :

```
>> from math import pi , log , exp
```

Contrairement au C, les variables utilisées sous Python ne nécessitent pas de déclaration (ce qui ne les empêche pas d'être typées). Pour créer une variable, il suffit de l'affecter. Par exemple :

`>> x = [1,2,3]` (ceci créera une variable de type list)

créez la variable `x` en tant que tableau à une ligne et 3 colonnes (ne pas oublier d'importer la bibliothèque `numpy` et utiliser le format tableau : `numpy.array()` ou `np.array()`).

Dans la console, vous pouvez visualiser `x` en tapant simplement son nom (ou en regardant dans la fenêtre des variables) :

```
>> x
```

Vous pouvez lister l'ensemble des variables utilisées dans l'espace courant en tapant la commande who :

```
>> who
```

Vous pouvez toute les remettre à zéro avec la commande `%reset -f` (et vous pouvez effacer la console avec la commande `clear`)

Attention `%reset -f` efface aussi l'environnement (les importations)

Pour créer un vecteur de 1 colonne et quatre lignes, il faut taper :

```
>> y = np.array([[1],[2],[3],[4]])
>> y # pour afficher le résultat
```

Vous pouvez obtenir le même résultat en utilisant la fonction `print()`.

```
>> print(y)
```

Essayez ces différentes commandes :

```
>> pi (importé numpy si necessaire)
>> 0/0
>> 5**3
```

Expliquez.

Les commentaires sont identifiés par : `#` s'ils sont sur une ligne ou `"""` en début et en fin s'ils sont sur plusieurs lignes.

Pour faire du calcul, il suffit d'écrire la formule (dans la console Python : onglet "IPython Console").

C'est très simple. Par exemple, essayez de calculer : $x = \frac{2*\left(512*cos\left(\frac{\pi}{3}\right)+13^2\right)}{127}$ (sans oublier bien sûr le nom de bibliothèque avant les fonctions ou les variables utilisées ex `math.cos()` ou `m.cos()` si vous avez utilisé "`as m`" lors de l'importation)

Comme vous pouvez le voir, un clou chasse l'autre, car le fait de donner une nouvelle valeur à `x` vous a fait perdre son ancienne valeur.

Essayez les opérateurs suivants : `< > <= >= == != and` et `or` (inférieur, supérieur, inférieur ou égal, supérieur ou égal, égal, différent, et, ou). Notez le symbole `!` qui signifie la négation.

Essayez d'utiliser des fonctions telles que racine carrée (`sqrt`), exponentielle (`exp`), logarithme (`log`), sinus, cosinus, tangente, arctangente.

2.3 - Calcul matriciel.

Pour effectuer des calculs matriciels avec Python il faut utiliser le format `array` de la bibliothèque `numpy`. Les principales fonctions de calcul matriciel de la bibliothèque `numpy` sont les suivantes : `numpy.dot()`, `numpy.linalg.det()`, `numpy.linalg.inv()`, `numpy.linalg.eig()` elles servent à calculer le produit, le déterminant, l'inverse et les valeurs propres et vecteurs propres

... il faut bien sûr faire attention aux dimensions des matrices. À titre d'exemple, calculez :

```
>> A = numpy.array([[1,2,3],[4,5,6]])
>> B = numpy.array([[1,2],[3,4],[5,6]]
>> C = numpy.dot(A,B)
>> D = numpy.dot(B,A)
>> E = B + A
>> F = A + A
```

A partir de la version 3,5 de python il est possible d'utiliser l'opérateur `@` pour effectuer la multiplication de 2 matrices.

Attention `*` est le produit terme à terme et array n'est pas matrix (surtout pour les vecteurs, pour convertir un tableau 1D en vecteur on peut utiliser la fonction `reshape()` de numpy)

```
>> A = numpy.array([[1,2],[3,4]])
>> B = numpy.array([[5,6],[7,8]])
>> C = A @ B
>> D = A * B
```

Constatez la différence entre ces deux opérations.

Toutes les fonctions de la bibliothèque `numpy` sont utilisables avec les matrices : `sin()`, `cos()`, `exp()`,...

Pour inverser une matrice A vous pouvez écrire `inv(C)` à condition d'avoir préalablement importé cette fonction depuis la bibliothèque numpy.linalg (`from numpy.linalg import inv`).

Pour transposer une matrice :

`>> D = A.T` (ou `np.transpose()`)

Il vous est aussi possible de créer des matrices spéciales telles que des matrices identités, des matrices ne contenant que des zéros et des matrices remplies de nombres aléatoires (tirages uniformes ou gaussiens). Découvrez ces fonctions : numpy.ones(), numpy.zeros(), numpy.random.rand() et numpy.random.randn().

2.4 – Affichage (matplotlib)

Le langage Python vous donne la possibilité d'afficher graphiquement les résultats d'expérimentations grâce à la bibliothèque `matplotlib`.

```
>> import matplotlib.pyplot as plt
```

Rm Il faut aussi importer `numpy` pour le calcul des vecteurs temps par exemple et les fonctions mathématiques de bases, cosinus, sinus, etc. (`>> import numpy as np`)

Pour afficher une fenêtre de figure (vide) :

```
>> plt.plot()
```

Comme vous pouvez le voir, ces commandes font apparaître une fenêtre d'affichage (dans la fenêtre "Plots", onglet à côté de "Variable Explorer")

Vous pouvez aussi donner des numéros à vos figures pour les différencier.

```
>> plt.figure(10)
>> plt.plot()
```

Chaque fois que vous voudrez afficher sur cette fenêtre, il vous suffira de faire précéder les commandes d'affichages par `>> plt.figure(10).`

-> - Exercice : trouvez et affichez une droite au sens des moindres carrés.

On va supposer une droite de pente a=2 et d'origine b=-3. Son équation est donc

y = ax + b. Vous allez créer un vecteur de 50 valeurs comprises entre o et 10 par :

`>> x = np.linspace(0,10,50)` ou `x = np.arange(0,10,1/5)`

Ensuite vous créez le vecteur des 50 valeurs de y correspondant à ces 50 valeurs d'abscisse :

```
>> a=2; b=-3; y=a*x+b
```

Pour visualiser la droite :

```
>> plt.figure(10)
>> plt.plot(x,y,color='red')
```

Il y a de nombreuses options pour la fonction `plot`. Découvrez-les par vous-même.

Par exemple, changez la couleur de la droite, affichez-la en pointillés, etc.

Nous allons bruiter cette correspondance linéaire entre `x` et `y` par un bruit gaussien centré de variance 2 :

```
>> y=y+(np.random.randn(np.size(y))*np.sqrt(2))
>> plt.figure(1l)
>> plt.plot(x,y,'b*')
```

La suite, c'est à vous de le faire avec cette petite aide mathématique :

Soient n points d'abscisses $x_1 \ldots x_n$ et d'ordonnées $y_1 \ldots y_n$, la droite de coordonnées (a,b) minimisant l'erreur ||ax + b - y|| est donnée par :

$\begin{bmatrix} a \\ b \end{bmatrix} = (A^T A)^{-1} A^T B$ avec $A = \begin{bmatrix} x_1 & 1 \\ \cdots & \cdots \\ x_n & 1 \end{bmatrix}$ et $B = \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix}$

l'opérateur ^{T} signifiant la transposition et l'opérateur $^{-1}$ l'inversion.

Calculez les coefficients de cette droite, tracez-la sur une autre figure (il vous faudra utiliser les fonction `vstack` ou `hstack` pour construire la matrices A).

Astuce : pour tracer plusieurs courbes sur la même figure à l'aide de la console il faut les mettre sur la même ligne : `plt.plot(x,ycourbe1);plt.plot(x,ycourbe2)`

Affichez la courbe et calculez la variance de l'erreur d'approximation, c'est-à-dire $\frac{1}{n}\sum_{i=1}^{n}(ax_i + b - y_i)^2$ en faisant des opérations matricielles (en remarquant que la norme d'un vecteur E est égale à E^TE). Que constatez-vous ? Vous pouvez aussi calculer la moyenne de cette erreur en utilisant la fonction `sum()` et `len()`.

Vous pouvez aussi visualiser des données tridimensionnelles. Un exercice classique est de tracer le sinus cardinal en 3D. Il faut aussi importer une bibliothèque supplémentaire (en plus de `numpy` et `matplotlib.pyplot`).

```
>> from mpl_toolkits.mplot3d import axes3d
>> x = np.arange(-4,4,0.2) ; y = x
>> X, Y = np.meshgrid(x, y)
>> z = np.sinc(X)*np.sinc(Y)
>> fig7=plt.figure(7)
>> ax = plt.axes(projection='3d') ; ax.plot_surface(X, Y, z, cmap='viridis')
>> fig8=plt.figure(8)
>> bx = plt.axes(projection='3d'); bx.plot_wireframe(X, Y, z, color='black')
```

Expliquez le résultat obtenu. Essayer de le refaire avec une gaussienne par exemple.

Regardez toutes les options des fonctions `surface` et `wireframe`.

Essayez de modifier les propriétés des images avec les fonctions `facecolors()` et `shade()`.

Regarder comment on peut faire tourner un graph avec la fonction `view_init()`

2.5 - Quelques fonctions.

Découvrez et tracez les fonctions `math` suivantes : `asin()`, `acos()`, `atan()`, `abs()`, `round()`, `floor()`, `ceil()`.

Essayez les fonctions `numpy` suivantes sur des échantillons aléatoires : `mod()`, `mean()`, `max()`, `min()`, `std()`.

2.6 - Les boucles.

Les boucles sous Python sont très simples, elles commencent par un ordre de début de boucle comme `for`, `while`, `if` mais ne terminent pas par un ordre de fin : tout est codé par l'indentation.

Exemples :

```
>> x=numpy.zeros(100)
>> for i in range (100):
>>    x[i]=(2.3*i+5*numpy.random.randn(1,1))
>> y=numpy.zeros(100)
>> for i in range (100)
>>    if x[i]>5:
>>          y[i]=1
>>    else:
>>          y[i]=-1
```

-> - Exercice

À titre d'exercice, essayez de faire une petite routine qui vous donnerait la plus petite valeur réelle représentable sur votre machine (ça dépend des machines).