TE0 : Introduction à python (interface ipython)

Objectifs

Les objectifs de ce TE numérique sont :

i) de prendre en main l'interface ipython ;

ii) d'introduire les modules numpy et matplotlib.pyplot ;

iii) d'écrire un premier script python ;

iv) d'inclure des fonction dans un script python.

La programmation sera réalisée en `python 3`. Il est recommandé d'écrire des scripts faisant appel à des fonctions élémentaires (fonctions graphiques ou de calcul).
Dans votre répertoire de personnel, créer les répertoires `TE0`, `data` et `plots`.
Les scripts python et les modules de fonctions seront des fichiers placés dans le répertoire `TE0`. Les fichiers graphiques seront placés dans le répertoire `plots`, les données dans le répertoire `data`.

Outils python

Vous aurez besoin d'outils mathématiques, tel le module `numpy` pour le calcul, et graphiques, tel `matplotlib.pyplot` pour la visualisation. Au commencement de chaque session interactive, ou au début de chaque script, il conviendra de charger ces modules, et plus généralement tous ceux dont on aura besoin.

```
import numpy as np                      # "as" introduit un raccourci (alias)
import matplotlib.pyplot as plt         # pour le module chargé
```

Remarque : les commentaires sont précédés du caractère `#`. Il est vivement recommandé de commenter vos scripts par souci de lisibilité.
Les appels aux fonctions de ces modules seront précédés par leur alias (par exemple `plt.plot(x,y)`, ou `y = np.log(x)`).
Il existe bien d'autres modules utiles que vous découvrirez petit à petit. Vous serez amenés aussi à créer vos propres modules.

Travail à faire

Les questions qui suivent consistent en l'écriture de commandes python. Vous pourriez le faire de façon interactive dans la console `ipython`. Il est grandement préférable d'enregistrer ces commandes dans un fichier, i.e. un script python, que vous exécuterez ensuite. Lors de l'exécution, les commandes seront interprétées séquentiellement.
Éditer votre fichier avec un éditeur de texte (`gedit`, `emacs`, `vi`, ...). Il est recommandé de commencer tout programme python par les lignes suivantes, indiquant le chemin de l'interpréteur python et le codage des caractères du fichier.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

Importer ensuite les modules `numpy` et `matplotlib.pyplot`

1. Création d'un vecteur 1D et tracé

— Créer un tableau de nombres réels `x` de 101 termes, variant de 0 à 2π avec un pas fixe (fonctions `np.linspace` et constante `np.pi`).
— Afficher les 3 premières et les 3 dernières valeurs du tableau `x`. Le tableau `x` comprend-il les valeurs 0 et 2π ?
— Créer le tableau `xa` variant de 0 à 2π avec le même pas que le tableau `x` à l'aide de la fonction `np.arange`. Que remarquez vous concernant les premières et dernières valeurs de `xa` ?
— Définir les tableaux $y = \cos(x)$ et $z = \sin(x)$ (fonctions `np.cos` et `np.sin`).

2. Création d'une figure et de fichiers graphiques

— Créer la figure numéro 1 (`plt.figure(1)`). Tracer `y` et `z` en fonction de `x` dans la figure 1 (fonction `plt.plot`). Visualiser la figure à l'écran (fonction `plt.show`).
— Superposer en rouge, avec le marqueur `'.'`, les points `(x,y)` en ne prenant qu'un point sur 3.
— Ajouter une grille, un titre et des libellés d'axes.
— Sauvegarder la figure sous forme d'un fichier `pdf`, dans le répertoire `plots` (fonction `plt.savefig`, le suffixe du nom du fichier déterminant le format).
— À partir d'un autre terminal, afficher la figure sauvegardée :
  — `evince nom_du_fichier.pdf &`
  On peut enrichir les figures sous ipython, en ajoutant par exemple des légendes aux courbes (argument `label` de la fonction plt.plot et fonction `plt.legend`. Sauvegarder de nouveau puis rafraîchir l'affichage du fichier pdf.

3. Écriture d'une fonction python élémentaire

— Écrire les fonctions numériques suivantes :
  — `double(t): double de t`
  — `norme(x, y)` : norme euclidienne du vecteur de composantes `x` et `y`
— Appliquer `double` à des scalaires.
— En utilisant une boucle, créer le tableau xd, double de `x`. Afficher le résultat.
— À l'aide d'une boucle, appliquer `norme` aux tableaux 1D `y` et `z`.
— Reprendre les 2 questions précédentes en appliquant les fonctions sur les vecteurs `x`, `y` et `z` (sans utiliser de boucle).

4. Création d'une fonction dont l'argument est une fonction

— Écrire une fonction `tabule(f, xdeb, xfin, n=51)`.
  — l'argument f de tabule est une fonction numérique scalaire
  — elle crée deux tableaux 1D `x` et `y` de chacun `n` valeurs,
    — `x` est constitué de n points régulièrement espacés entre `xdeb` et `xfin` (bords inclus)
    — `y` contient les valeurs prises par `f` aux points de `x` : `yi = f(xi)` tabule (échantillonne) la fonction `f` sur l'intervalle `[xdeb, xfin]`
  — la fonction `tabule` renvoie les deux tableaux `x` et `y`
  — l'argument `n` de tabule est optionnel : s'il n'est pas spécifié dans l'appel à la fonction `tabule` il prend la valeur 51 (valeur par défaut).
— Appliquer `tabule` à `double` puis à `np.cos` sans fournir `n`, puis avec `n=11`. Afficher (commande `print`).

Remarque : Vous avez deux possibilités pour placer la définition d'une nouvelle fonction :

1. soit vous placez cette définition au sein de votre script, avant toute commande (en préambule),
2. soit vous la placez dans un autre fichier, fichier que vous importerez dans le script (directive `import`) avec éventuellement un alias. La fonction sera appelée avec la syntaxe `alias.nom_de_la_fonction`. Le fichier pourra contenir les définitions de plusieurs fonctions. Il sera réutilisable pour d'autres scripts.

5. Écriture des tableaux 1D de même longueur dans un fichier

— Appeler `tabule` pour échantillonner par exemple `double` sur l'intervalle $[0, 2]$.
— Sauvegarder les deux tableaux ainsi obtenus, `x` et `y`, dans un fichier texte nommé `double.txt` dans le répertoire `data`, à raison d'un couple `x[i]`, `y[i]` par ligne (fonction `np.savetxt`).

— Vérifier en affichant dans un terminal le contenu (texte) du fichier créé. À l'aide d'une commande unix (en utilisant la syntaxe `!commande_unix` depuis la console ipython). Afficher le nombre de lignes et de mots du fichier.

6. Lecture d'un fichier de deux colonnes et tracé

— Écrire une fonction `tracefic` qui lit le fichier texte nom_fic et trace la deuxième colonne en fonction de la première. (La fonction `np.loadtxt` permet de lire des fichiers textes contenant des données numériques en colonnes.)
— Appeler `tracefic` avec pour argument le fichier `double.txt`. Sauvegarder la figure et visualiser la.

7. Manipulation d'un tableau 2D

— Créer le tableau 2D `xyz` de 101 lignes, 3 colonnes (fonction `np.ndarray((l,c))`). Ranger les tableaux `x, y` et `z` dans `xyz`.
— Examiner les «méthodes» `shape` et `size` appliquées au tableau `xyz`.
— Calculer la somme des termes de la seconde colonne du tableau (i.e. `y`) (fonction `np.sum`).
— Définir le vecteur `ipi` des indices d'abscisse `x` (première colonne) inférieures à π (fonction `np.where`). Calculer la somme des termes de la seconde colonne du tableau (i.e. `y`) correspondant aux abscisses inférieures à π. En déduire une expression approchée de l'intégrale $\int_0^\pi \cos(x)\mathrm{d}x$.
— Afficher le résultat (fonction `print`).

8. Écriture d'une fonction : intégration d'une fonction tabulée

L'estimation de l'intégrale de cosinus comme la somme de rectangles est grossière. Une meilleure estimation est obtenue par la méthode dite des trapèzes. L'aire sous la courbe $f(x)$ entre les abscisses x_i et x_{i+1} est approchée par :

$$\int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x = \frac{f(x_i)+f(x_{i+1})}{2}(x_{i+1}-x_i)$$

La méthode consiste à substituer à la fonction f dans l'intervalle $[x_i, x_{i+1}]$ son interpolation linéaire entre les abscisses x_i et x_{i+1}.

— Écrire une fonction python `trapeze_t` calculant l'intégrale d'une fonction tabulée de $y(x)$. La syntaxe de l'appel sera : `I = trapeze_t(x,y)`.
— Calculer et ranger dans un tableau `I[n]` l'intégrale de $\int_0^\pi \cos(t)$, le vecteur x variant de 0 à π avec $n-1$ pas.
— Tracer la valeur absolue de I (erreur sur l'intégrale de cos entre 0 et π) pour n variant de 10 à 1000. Tracer avecune échelle semi-logarithmique (`plt.semilogy`).