
TRAVAUX PRATIQUES N° 1 : Prise en main de Python pour l'optimisation

Préambule : Pour ce TP, on se basera sur le fichier `tp1.py`¹. On privilégiera le logiciel Vscodium sur les machines de l'université (ou VSCode sur vos machines personnelles), avec le package python installé (taper `"ctrl + shift + p"`², puis "Extensions: Install Extensions", et choisir Python). On pourra alors lancer des cellules de code en tapant sur `shift+enter` dans une cellule délimitée par les symboles `"# %%"` :

```
# %%  
# Début de cellule  
print(1+3)  # commentaire en ligne  
# %%  
# Une autre cellule  
print(2**3)  # commentaire en ligne
```

On peut aussi utiliser lancer une cellule en cliquant sur le bouton "run cell" dans Vscodium.

Dans ce TP on utilisera beaucoup l'aide de base de python (en particulier les commandes `help(nom-de-la-fonction)` ou `?nom-de-la-fonction` seront bien utiles). On pourra aussi utiliser l'aide en ligne, plus agréable pour matplotlib et des éléments graphiques plus détaillés.


Enfin les questions notées Pour aller plus loin, peuvent être omise en première lecture.


1 Introduction à numpy et matplotlib

On charge les packages utiles de la manière classique suivante :

```
import numpy as np  
import matplotlib.pyplot as plt
```

1.1 Test d'égalité et précision en numérique


Dans cette partie, on va faire un retour sur les nombres flottants ( : *floats*) dans un ordinateur, et quand est-ce que deux nombres flottants sont dits égaux.


QUESTION 1. (Dépassement) À l'aide de l'import `sys`, stocker le plus grand nombre représentable en machine : n_{\max} (obtenu avec `sys.float_info.max`). Tester l'égalité entre $n_{\max} + 1$ et n_{\max} . Multipliez ce nombre par une quantité très proche, mais supérieure, à 1. Qu'observez-vous ? Quel est le résultat de l'opération $1.1 \cdot n_{\max} - 1.1 \cdot n_{\max}$, comment l'expliquez-vous ? On parle de dépassement ( : *overflow*) quand un nombre dépasse le nombre le plus grand que l'ordinateur peut représenter en mémoire. Notez que les problèmes de dépassement mal gérés peuvent coûter très cher, même à d'excellents ingénieurs³.

1. `tp1.py` est disponible ici : <http://josephsalmon.eu/enseignement/Montpellier/HAX606X/TP/tp1.py>

2. La touche `shift` : https://fr.wikipedia.org/wiki/Touche_majuscule

3. https://fr.wikipedia.org/wiki/Vol_501_d%27Ariane_5

QUESTION 2. (Soupassement) Le soupassement ( : *underflow*) est le phénomène inverse du dépassement. Cela se produit quand un nombre est tellement proche de 0 que l'ordinateur ne le différencie plus de 0. Observez un cas d'*underflow* (on pourra notamment essayer de trouver à partir de quelle puissance négative de 2 ce phénomène apparaît).

 : La précision machine ne pose pas problème qu'à partir de ces limites. Manipuler des grandeurs extrêmes est un jeu à la fois dangereux et délicat. Quand les puissances résultent en des nombres "extrêmes", des arrondis sont effectués⁴.

QUESTION 3. (Arrondis) Utilisez la fonction `spacing` de `numpy` pour savoir à partir de quand cet arrondi est effectué sur une grandeur quelconque (cette fonction renvoie la distance entre un nombre et son plus proche voisin). Testez l'égalité entre :

- `10**9` et `10**9 + 10**(-8)`,
- `10**9` et `10**9 + 10**(-7)`,
- `10**9` et `10**9 + np.spacing(10**9)/2`,
- `10**9` et `10**9 + np.spacing(10**9)/1.9`,
- `0.6` et `0.3 + 0.2 + 0.1`,
- `0.6` et `0.1 + 0.2 + 0.3`.

Pour aller plus loin : <https://docs.python.org/3/tutorial/floatpoint.html> approfondit les questions autour des nombres flottants.

QUESTION 4. (Précision relative / absolue) Pour éviter que des erreurs d'arrondis, on contrôle l'égalité d'un point de vue numérique en acceptant une certaine marge d'erreur

$$a \approx b \iff |a - b| \leq \text{atol} + \text{rtol} |b| \text{ .}$$


Utilisez la fonction `isclose` de `numpy` en choisissant des paramètres `atol`, `rtol` pour que l'égalité dans le dernier cas de la question ci-dessus soit vraie puis fausse. Remarque : on pourra aussi utiliser la fonction `allclose`, qui fonctionne de manière similaire pour des tableaux.

1.2 Opérations matricielles et visualisation 1D

Pour cette partie, un rappel visuel des créations/transformation usuelles de matrices est disponible ici : <https://github.com/josephsalmon/HAX712X/tree/main/Courses/ScipyNumpy>.


QUESTION 5. (Algèbre linéaire) Manipulez les opérations classiques sur des matrices (arrays) de `numpy` (si vous êtes déjà habitué à `numpy`, vous pouvez passer à la question 2).

- Additions (+) / soustractions (-)
- Multiplications : terme à terme (*) / matricielle (@ ou `np.dot(A, B)`)
- Puissance : terme à terme (**) / matricielle (`np.linalg.matrix_power(A,n)`)
- Inversion matricielle : (`np.linalg.inv(A)`)
- Résolution de système (`np.linalg.solve(A, b)`)

 : en pratique vous n'utiliserez presque jamais l'inversion de matrice ! En effet, on n'inverse **JAMAIS JAMAIS JAMAIS** une matrice, sauf si l'on a une bonne raison de le faire. La plupart du temps on doit résoudre un système linéaire $Ax = b$, et il n'est pas utile de calculer $A^{-1}b$ pour cela (c'est en effet souvent trop lent de procéder par inversion).

- Matrices usuelles : `np.zeros(m, n)`, `np.eye(n)`, `np.empty(n)`, `a.fill(np.pi)`, `np.ones(n)`.

4. Plus d'information : <https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html> chapitre 9

- Extraction de parties d'une matrice : mettre à zero une ligne sur deux de la matrice identité de taille 5×5 . Voir le "tranchage" ( : *slicing*).
- Grille de nombres : `np.linspace`, `np.logspace`, `np.arange`, etc. Créer un vecteur qui contient toutes les puissances de 10 de l'exposant 1 à l'exposante 9.
- Tailles / changement de taille : `np.shape(A)` ou `A.shape`; `np.reshape(A, (n1, n2))`.
- Partant de `d=np.arange(6)`, quelles sont les dimensions de `d`, `d.reshape(2, 3)`, `d.reshape(3,2)`, `d.reshape(6,)`, `d.reshape(1, 6)`. Commentez.
- ⚠ : `numpy` utilise pour les vecteurs des tableaux n'ayant pas de deuxième dimension (car dans la mémoire de l'ordinateur la seconde dimension n'existe pas, il y a donc une simple instruction qui dit si l'on parcourt le tableau en ligne ou en colonne).
- Transposition `A.T` ou `np.transpose(A)`.

Pour aller plus loin : On pourra découvrir les différences entre copies et vues d'une matrice :
<https://www.geeksforgeeks.org/copy-and-view-in-numpy-array/>

Pour aller plus loin : Les `numpy` arrays peuvent être parcourus / stockés soit en convention "lecture en colonne" (*F-order*, F pour FORTRAN) soit "lecture en ligne" (*C-order*, C pour le langage C). Par défaut `numpy` utilise cette dernière convention :
<https://numpy.org/doc/stable/dev/internals.html#numpy-internals>

QUESTION 6. (Affichage de fonctions 1D) Le but ici est de tracer les graphes de fonctions avec `matplotlib` et de mettre en valeurs plusieurs points intéressants.

- Tracez la fonction $f : x \mapsto \cos(x)$ sur l'intervalle $[-10, 10]$.
- Utilisez la fonction `hlines` de `matplotlib` pour tracer les droites $y = \pm 1$ (on pourra modifier `linestyles` pour faire un trait pointillé).
- Utilisez la fonction `scatter` pour mettre en valeur ses points critiques (modifier l'option "s" pour modifier la taille des marqueurs).
- **Tout graphique doit contenir un titre, être légendé et être lisible!** Modifiez la taille de l'image avec l'argument `figsize` et rendez ce graphique présentable. En modifiant le paramètre `zorder` faire en sorte que les points apparaissent au-dessus de la courbe.

QUESTION 7. (Graphes multiples)

- En utilisant `subplot`, affichez une double figure représentant les fonctions $f : x \mapsto \exp(-\lambda x)$ sur l'intervalle $[0, 10]$, pour $\lambda = 1, 2, 3, 4, 5$. Le premier graphe (en haut) sera en échelle habituelle et le second (en bas) sera en échelle semi-logarithmique sur l'axe des y, pour mieux visualiser les convergences vers 0. On choisira une palette continue pour afficher chaque valeur de λ de manière graduelle, voir :
<https://matplotlib.org/stable/tutorials/colors/colormaps.html>
- Ajouter des titres et sous-titres.

Pour aller plus loin : Afficher un troisième graphique reproduisant la version avec échelle semi-logarithmique en transformant directement les données. Pour plus d'informations sur les différents types d'échelles et une gestion plus fine des transformations (par exemple quand il y a des valeurs négatives, la base logarithmique utilisée, etc.) voir :
<https://matplotlib.org/stable/tutorials/introductory/pyplot.html#logarithmic-and-other-nonlinear-axes>

2 Fonctions à plusieurs variables

2.1 Fonctions contre-exemples

Rappel : Théorème 1.2.2 : « Pour tout ouvert U de \mathbb{R}^2 , si $x_0 \in U$ est un minimum local de la fonction $f \in \mathcal{C}^2(U)$, alors $f'(x_0) = 0$ et $f''(x_0)$ est semi-définie positive. » La réciproque de ce théorème est fautive : on considère ainsi $f_{\text{selle}}(x, y) \mapsto x^2 - y^4$ en $(0, 0)$ définie de \mathbb{R}^2 dans \mathbb{R} .

QUESTION 8. (Conditions du premier et deuxième ordre) Ajouter le point à l'origine sur les graphiques des courbes de niveaux et la surface 3D de la fonction f_{selle} . On pourra utiliser la fonction `plt.scatter`. Interpréter visuellement le comportement de la surface en $(0, 0)$

QUESTION 9. (Dérivées directionnelles) On considère maintenant la fonction :

$$f_{\text{directionnelle}} : (x, y) \mapsto \begin{cases} \frac{xy}{x^2+y^2}, & \text{si } (x, y) \neq (0, 0) \\ 0, & \text{sinon} \end{cases}.$$

Expliquer pourquoi cette fonction admet des dérivées partielles en tout point. Est-ce que la fonction $f_{\text{directionnelle}}$ est continue en tout point (en particulier à l'origine) ? Visualiser les lignes de niveau de cette fonction (en afficher 12) et la surface engendrée.

3 La gestion de l'aléa dans Python

Le module `random` de `numpy` permet d'utiliser l'aléatoire et des lois usuelles en Python. On crée d'abord un générateur qui nous permettra ensuite d'appeler les lois voulues comme suit :

```
generateur = np.random.default_rng()
generateur.normal()
```

QUESTION 10. (Random)

- Créer une matrice de taille 4×5 dont les entrées sont *i.i.d.* de loi de Laplace d'espérance 0 et de variance 2. Lancez plusieurs fois la cellule et observez les changements. Vous pourrez consulter l'aide de `numpy` pour cela : <https://numpy.org/doc/stable/reference/random/generator.html>
- Pour produire des résultats reproductibles ou déboguer un code, il est utile de "figer" l'aléa. On utilise pour cela une graine (🇬🇧 : *seed*) dans la création du générateur. Fixez la graine à 0 dans la création du générateur et relancez plusieurs fois la cellule. Qu'observez-vous ?
- Afficher avec `plt.subplot` un histogramme comparant : 100 tirages de loi gaussienne (centrée-réduite) ; 100 tirages de loi de Cauchy ; 100 tirages de loi de Laplace. On choisira les mêmes paramètres de centrage et d'échelle pour les trois lois.

Pour aller plus loin : Toutes les lois usuelles disponibles sont dans la documentation <https://numpy.org/doc/stable/reference/random/generator.html>, et vous pourrez en manipuler avec des widgets ici : <https://github.com/josephsalmon/Random-Widgets>.⁵

5. En ligne, vous trouverez parfois des codes qui utilisent l'ancienne manière de générer de l'aléa avec `numpy` : `np.random.laplace(...)`. C'est une manière dépréciée de générer des tirages aléatoires qui n'est plus recommandée (même si encore fréquemment rencontrée, notamment sur internet).

Ressources utiles

- https://numpy.org/doc/stable/user/absolute_beginners.html
- <https://scipy-lectures.org/intro/numpy/operations.html>