

## Outils Numériques pour l'Ingénieur·e en Physique

2024-2025

5N-028-SCI / ONIP-1

### Bloc 1 - Introduction à Python ( )

#### Concepts étudiés

[NUM] Vecteurs et matrices  
[NUM] Signaux numériques  
[NUM] Figures scientifiques  
[MATH] Transformée de Fourier  
[PHYS] Echantillonnage

#### Mots clefs

Vecteurs; Matrices; Signaux; Dis-  
crétisation

#### Sessions

0 Cours(s) - 1h30  
0 TD(s) - 1h30  
3 TD(s) Machine - 2h00  
0 TP(s) - 4h30

#### Travail

Seul ou par binôme

Institut d'Optique  
Graduate School, France  
<https://www.institutoptique.fr>

GitHub - Digital Methods  
<https://github.com/IOGS-Digital-Methods>

# Maitriser les concepts de base de Python pour la Science

Dans le monde scientifique, on part souvent d'une problématique (ou d'une observation) que l'on cherche à modéliser et à comprendre, à l'aide de diverses lois. Ce travail mène souvent à un ensemble d'équations qui nécessitent alors d'être résolues (équations linéaires, équations différentielles, intégrales...).

Pour arriver à résoudre ces systèmes d'équations de **manière exacte** (ou analytique), il est souvent nécessaire de faire des **hypothèses simplificatrices** qui mènent alors à une **résolution approchée** du phénomène physique observé.

Pour pouvoir **réaliser une étude plus poussée**, il est intéressant de conserver certaines parties des équations non simplifiées. La résolution analytique en devient alors souvent impossible. C'est là que l'informatique peut nous venir en aide.

**Python** est un langage de programmation de haut niveau, avec une syntaxe relativement naturelle pour l'être humain et un **large choix de bibliothèques** permettant de le rendre généraliste.

Nous allons voir à travers les trois premières séances des **outils de base** qu'un.e futur.e ingénieur.e dans le domaine de la physique doit maitriser : générer des signaux de tests valides, créer des fonctions basés sur des lois physiques ou/et mathématiques, produire des graphiques pertinents des résultats...

## Acquis d'Apprentissage Visés

En résolvant ces problèmes, les étudiant·e-s seront capables de :

### CÔTÉ NUMÉRIQUE

1. **Générer des signaux numériques** à partir de fonctions mathématiques
2. **Définir et documenter des fonctions** pour faciliter la réutilisabilité d'un code
3. **Produire des figures** claires et légendées à partir de signaux numériques - incluant un titre, des axes, des légendes
4. **Utiliser un environnement de développement** pour faciliter l'écriture et le débogage d'un code

### CÔTÉ PHYSIQUE

1. **Valider des données de test**

## Ressources

Cette séquence est basée sur le langage Python.

Nous utiliserons l'environnement **PyCharm** (édition Community 2023) et **Python 3.10** (inclus dans la distribution Anaconda 3).

Des tutoriels Python (et sur les bibliothèques classiques : Numpy, Matplotlib or Scipy) sont disponibles à l'adresse : <https://lense.institutoptique.fr/python/>.

## Travail à réaliser

---

### SÉANCE 1

---

Cette séance est consacrée à l'utilisation de l'environnement de développement (IDE) **PyCharm**, qui permet de faciliter l'écriture et le débogage d'un code en Python.

*D'autres IDE existent et permettent de faire des choses similaires. C'est le cas par exemple de **VSCode** (avec les extensions adéquates).*

---

En tant qu'étudiant.e à l'université Paris-Saclay, vous pouvez avoir accès gratuitement à la version professionnelle de **PyCharm**. Il existe également une version communautaire entièrement gratuite (mais avec quelques fonctionnalités en moins - la gestion des fichiers Jupyter par exemple).

---

Vous pouvez également vous aider de la documentation officielle de PyCharm <https://www.jetbrains.com/fr-fr/pycharm/learn/>

---

### Exercice 1 / Création d'un projet

Voir tutoriel (à rédiger !!)

---

## Outils Numériques

Fonctions et bibliothèques conseillées :

- **Python**:
  - `print`
  - `len`, `type`
  - `enumerate`
- **time** *temps*
  - `time`
- **Numpy** *matrices*
  - `shape`, `dtype`
  - `mean`
  - `dtype`
  - `arange`, `linspace`
  - `logspace`
  - `ones`, `zeros`
  - `meshgrid`
- **Matplotlib** *affichage*
  - `plotly`
  - `figure`, `plot`
  - `subplot`
  - `legend`, `title`
  - `xlabel`, `ylabel`
  - `show`
- **Scipy** *science*
  - `fftpack` sublibrary
  - `fft`, `ifft`
  - `fftshift`
  - `fftfreq`

---

Outils avancés :

- `rcParams` de Mat-  
`PlotLib.pyplot` pour  
l'amélioration de  
l'affichage de courbes

# Travail à réaliser (suite)

---

## SÉANCE 1 - SUITE

---

### Exercice 2 / Optimisation d'un code

On se propose d'étudier le code `first_try.py` fourni.

1. Créer un projet et ajouter le fichier mentionné précédemment dans votre projet.
2. Créer une configuration permettant d'exécuter ce script.
3. Que fait ce code ? Quelles améliorations pouvez-vous lui apporter ?
4. Utiliser l'option `Refactor / Refactor this...` pour modifier les noms de certaines fonctions ou certaines variables. Il est également possible de renommer les variables à l'aide du raccourci `MAJ + F6`.
5. Utiliser l'aide associée aux erreurs pour corriger le code. Il est possible de survoler les zones de codes soulignées avec la souris pour lire les recommandations de l'IDE.
6. L'affichage final est-il pertinent ? Générer un vecteur `c1` plus adapté et afficher à nouveau la courbe.
7. Ajouter les types des données d'entrée et de sortie de la fonction précédente. Ajouter une documentation de type `docstring` à votre fonction par un *clic droit* sur le nom de la fonction => `dessin d'ampoule` => `insert doc...`

---

### Exercice 3 / Utilisation de points d'arrêt et d'affichage formaté

On se propose d'étudier le code `sum_list_elements.py` fourni.

1. Ajouter le fichier mentionné précédemment dans votre projet.
2. Ajouter un point d'arrêt dans votre code, en double-cliquant dans la marge gauche de l'IDE.
3. Exécuter le code en mode **DEBUG** et contrôler l'exécution pas à pas (`resume`, `step over`, `step in/out...`).
4. Observer le contenu des différentes variables à l'aide de ...

---

### Exercice 4 / Codes à corriger

En vous aidant des outils précédemment évoqués, essayer de **corriger les codes** contenus dans les fichiers : `binary_search.py`, `linear_regression_and_noise.py` et `bubble_sort.py`.

Pour le code contenu dans le fichier `linear_regression_and_noise.py`, vous pourrez aussi améliorer l'affichage en réalisant par exemple une figure incluant 6 sous-figures...

---

## SÉANCE 2

---

### Exercice 1 / Listes

1. Générer une liste `L_temps` de 101 points régulièrement répartis entre 0 et 1 s.
2. Afficher la taille de cette liste.
3. Afficher le dernier élément de la liste.
4. Quelle est la période d'échantillonnage ?

---

### Exercice 2 / Temps d'exécution et fonction

1. Créer une fonction `get_list` qui génère une liste de `N` nombres réels régulièrement répartis entre une valeur `start` et une valeur `stop`. Les éléments `N`, `start` et `stop` seront des paramètres de cette fonction. Le paramètre `start` aura une valeur par défaut de 0.
2. Mesurer le temps d'exécution de la création d'une liste `L_test` de 10.000 points répartis entre 0 et 1.

---

### Exercice 3 / Génération d'un vecteur avec Numpy

1. Importer la bibliothèque `Numpy`.
  2. A l'aide de la bibliothèque `Numpy`, générer un vecteur `V_test` de 10.000 nombres réels répartis entre 0 et 1.
  3. Afficher la taille de ce vecteur.
  4. Afficher le dernier élément de ce vecteur.
  5. Mesurer le temps d'exécution de la génération d'un vecteur de 10.000 nombres réels répartis entre 0 et 1.
  6. Comparer à la mesure de l'exercice 2. Que concluez-vous ?
-

#### Exercice 4 / Génération de signaux et affichage

1. Créer un vecteur **temps** de 101 points régulièrement répartis entre 0 et 1 s. Quelle est la période d'échantillonnage ?
  2. Importer la bibliothèque **matplotlib.pyplot**.
  3. Tracer une sinusoïde de période 50 ms en rouge. Ajouter un titre, des axes et une légende au graphique.
  4. Que pensez-vous du résultat ? Améliorer le résultat.
  5. Tracer sur le même graphique une sinusoïde de période 20 ms en bleu.
  6. Faire un zoom pour n'afficher que quelques périodes des deux sinusoïdes.
  7. Faire une fonction qui prend comme argument la période de la sinusoïde, tester-la avec une période de 30 ms.
- 

#### Exercice 5 / Matrices

Soit la matrice **A** suivante :

$$A = \begin{pmatrix} 4 & 6 & -2 & 3 \\ 2 & -1 & 0 & 1 \\ -7 & 0 & 1 & 12 \end{pmatrix}$$

1. Créer une matrice **A** avec NUMPY (*np.array*) et l'afficher.
  2. Afficher le type de **A** et le type d'un des éléments de **A** . Afficher aussi la taille de la matrice.
  3. Afficher la seconde colonne puis la troisième ligne.
  4. Afficher la moyenne de tous les éléments, la moyenne par colonne, la moyenne par ligne.
  5. Utiliser l'affichage formaté pour afficher : "la moyenne de tous les éléments vaut ... "
  6. Modifier la matrice **A** pour que ses 2 premières lignes soient multipliées par 2 puis que sa dernière colonne soit divisée par 3. Quel est alors le type des éléments de **A** ?
  7. Créer une seconde matrice **B** de 3 lignes par 4 colonnes ne contenant que des 2. Remplacer la première colonne de **B** par des 0.
  8. Que représente **A\*B** ? Comment faire un produit matriciel entre **A** et **B** ?
  9. Créer une matrice **CC** qui ne conserve que les éléments supérieurs à 5 de la matrice **A\*B** et force les autres à 0.
- 

#### Exercice 6 / Fonction porte

1. Créer une fonction **porte** qui renvoie un vecteur de booléens à partir d'un vecteur **v\_in**. Ce vecteur sera **vrai** lorsque les valeurs de **v\_in** sont comprises entre une valeur **min** et une valeur **max** et **faux** sinon. Le vecteur **v\_in** ainsi que les valeurs **min** et **max** seront passés en paramètre.
  2. Tester cette fonction.
- 

#### Exercice BONUS / Résolution d'une équation du 2nd degré

Faire une fonction qui calcule les solutions d'une équation du second degré.

---

# Travail à réaliser (suite)

---

## SÉANCE 3

---

### Exercice 1 / Fonction mathématique et affichage

1. Faire une fonction  $f : x \rightarrow \sin(x) * \exp(-x/5)$ .
  2. Tracer la courbe représentative avec 500 points variant entre 0 et  $5\pi$ .
- 

### Exercice 2 / Fonction `fftshift`

1. Définir une fonction permettant de générer une gaussienne.  
`gauss : x -> exp(-x2)`
  2. Afficher cette courbe pour  $x$  allant de -5 à 5 sur 101 points.
  3. Utiliser la fonction `np.fft.fftshift` sur le signal gaussien et afficher le résultat. Que pouvez-vous conclure sur la fonction `np.fft.fftshift` ?
- 

### Exercice 3 / Transformée de Fourier

1. Générer un signal sinusoïdal de fréquence 200Hz avec une période d'échantillonnage de  $300\mu s$ , sur un intervalle de temps de 0.1s.
  2. Afficher ce signal.
  3. Calculer et afficher la transformée de Fourier (TF) discrète du signal précédent (parties réelle et imaginaire). Ajouter un titre, des axes et une légende.
  4. Afficher à présent le module de la TF discrète du signal précédent. Est-ce le résultat voulu ?
- 

### Exercice 4 / Utilisation d'un `meshgrid`

1. Générer une surface échantillonnée (2 dimensions) correspondant à une fonction sinusoïdale de taille 200 par 100, d'un pas de 10, dans le sens vertical, à l'aide d'une double boucle.
  2. Mesurer le temps d'exécution de cette fonction.
  3. Générer cette même surface échantillonnée à l'aide d'un objet de type `meshgrid` (bibliothèque `Numpy`).
  4. Mesurer le temps d'exécution de cette génération et comparer ce temps à celui obtenu précédemment. Qu'en concluez-vous ?
- 

### Exercice 5 / Surface échantillonnée 2D

1. Réaliser une fonction qui génère une trame en 2 dimensions sinusoïdale de taille  $M$  par  $N$ , d'un pas spatial de `step`, selon un angle `alpha`.
  2. Tester cette fonction et générer une trame de 200 par 100 selon un pas de 10.1 et un angle de 35 deg .
  3. Utiliser la fonction `np.fft.fftshift` sur la matrice générée.
  4. Faire la transformée de Fourier en 2D de la trame précédente et afficher le résultat.
-