TP

Tri

Sources :

Savoirs et compétences :

- ❒ *AN.S3 : Méthodes des rectangles et des trapèzes pour le calcul approché d'une intégrale sur un segment.*
- ❒ *SN.C1 : Réaliser un programme complet structuré*
- ❒ *SN.C2 : Étudier l'effet d'une variation des paramètres sur le temps de calcul, sur la précision des résultats, sur la forme des solutions pour des programmes d'ingénierie numérique choisis, tout en contextualisant l'observation du temps de calcul par rapport à la complexité algorithmique de ces programmes*
- ❒ *SN.C3 : Utiliser les bibliothèques de calcul standard*
- ❒ *SN.C5 : Tenir compte des aspects pratiques comme l'impact des erreurs d'arrondi sur les résultats, le temps de calcul ou le stockage en mémoire.*
- ❒ *SN.S1 : Bibliothèques logicielles*

Activité 1 : Mise en situation et généralité

1 Utilisation de bibliothèques

En python, il existe de nombreuses manières de produire des graphiques. Pour ce faire, nous allons utiliser la bibliothèque `matplotlib.pyplot`.

Une bibliothèque est un regroupement de fonctions, livrées directement avec Python ou écrites par un tiers. Nous allons en utiliser plusieurs cette année :

- `math`, pour utiliser des fonctions mathématiques de base ;
- `random`, pour simuler des expériences aléatoires ;
- `time`, pour disposer d'un chronomètre ;
- `scipy.optimize`, pour résoudre certains problèmes numériques ;
- `scipy.integrate` pour résoudre d'autres problèmes numériques ;
- `numpy`, pour utiliser des outils de calcul vectoriel ;
- et donc `matplotlib.pyplot`, pour avoir des outils de tracé graphique.

Certaines de ces bibliothèques sont toujours disponibles avec Python, comme les bibliothèques `math` et `random`, elles font partie de la bibliothèque standard. D'autres doivent être installées séparément, comme la bibliothèque `matplotlib.pyplot`, notamment.

On charge une bibliothèque avec le mot clef `import`. Il existe plusieurs manières de charger une bibliothèque. Prenons l'exemple de la bibliothèque `math`.

On peut d'abord charger l'intégralité de la bibliothèque.

▪ **Exemple**

```
import math
math.sqrt(2)
```

▪

Remarquez qu'il est alors obligatoire de rappeler de quelle bibliothèque vient chaque commande (plusieurs bibliothèques peuvent proposer des implémentations différentes d'une même fonction).

▪ **Exemple** La fonction `exp` du module `math` ne s'applique qu'à des nombres flottants, alors que celle de `numpy` peut s'appliquer à des vecteurs (cette structure sera détaillée ultérieurement).

```
import numpy
math.exp(1)
numpy.exp(1)
numpy.exp([1,2,3])
```

On peut donner un alias à une bibliothèque avec le mot clef `as`, afin de pouvoir l'utiliser rapidement.

Exemple

```
import math as m
m.pi
```

On peut aussi décider de n'importer que certaines fonctions d'une bibliothèque, avec le mot clef `from`. Dans ce cas, on peut utiliser la fonction importée sans préfixe.

Exemple

```
from math import log
log(2)
```

On peut aussi importer toutes les fonctions d'un coup de cette manière, en utilisant le joker `*`.

Exemple

```
from math import *
tan(pi/4)
```

2 Tracé de fonctions : utilisations de Matplotlib.pyplot

La bibliothèque `matplotlib.pyplot` fournit des outils de tracés graphiques très proches de ceux du logiciel Matlab. On essaiera toujours de l'utiliser de la manière suivante.

1. Appeler la fonction `clf()` (pour *clean figure*) pour effacer la figure précédente, et donc en commencer une nouvelle.
2. Tracer des courbes, nuages de points, en appelant (éventuellement plusieurs fois) la fonction `plot()`. On remarquera que rien n'est ici affiché : seul un objet est créé par Python.
3. Éventuellement, paramétrer le graphique (axes, titre, légendes *etc.*) en utilisant les fonctions adéquates.
4. Afficher le graphique avec la fonction `show()` ou, mieux, le sauvegarder avec la fonction `savefig()`. On dispose alors d'une image (différents formats possibles).

2.1 Utilisation de la fonction `plot`

Cette fonction peut-être utilisée de nombreuses manières différentes, pour obtenir des résultats assez fins. Nous ne les détaillerons pas toutes ici. L'idée générale est de lui donner en argument deux tableaux de nombres, décrivant respectivement les abscisses et les ordonnées des points à tracer. Sans autre option, ces points sont reliés par des segments bleus, en traits pleins.

Exemple Le code suivant produit la figure 1.

```
import matplotlib.pyplot as plt

x = [1, 3, 4, 2]
y = [2, 1, 4, 2]

plt.clf()
plt.plot(x,y)
plt.savefig("ex_base_01.png")
```

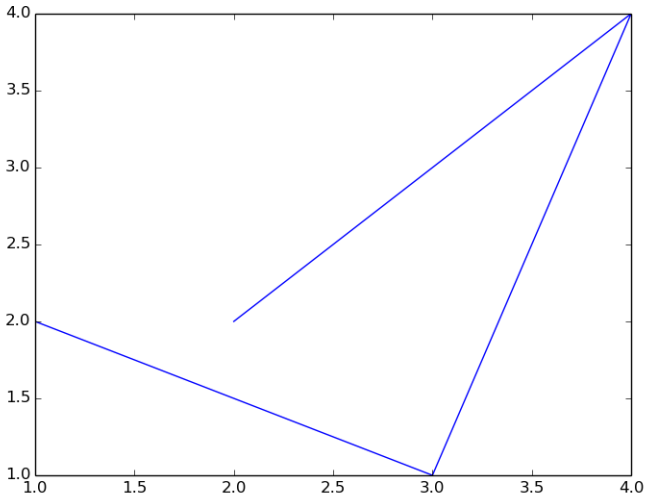

FIGURE 1 – Style standard

▪

Les options graphiques permettent notamment de modifier le type de segment (pointillés, sans segment), d'afficher ou non un marqueur au niveau de chaque point et de changer la couleur de ces objets.

▪ **Exemple** Le code suivant produit la figure 2.

```
import matplotlib.pyplot as plt

x = [1, 3, 4, 2]
y = [2, 1, 4, 2]

plt.clf()
plt.plot(x,y,'or--')
plt.savefig("ex_base_02.png")
```

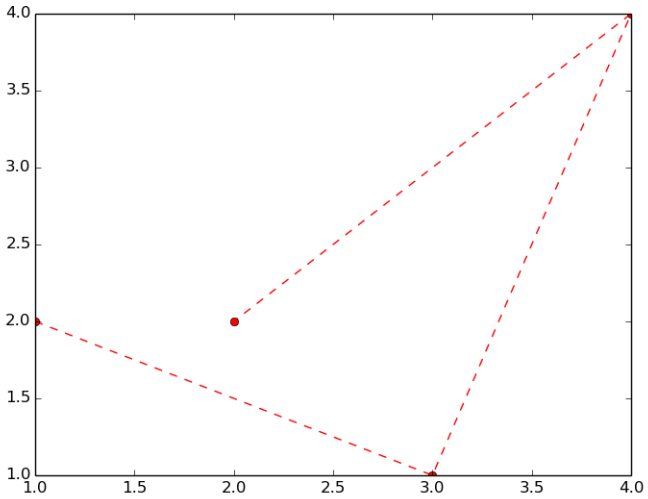

FIGURE 2 – Points marqués, ligne coupée, couleur rouge.

On remarquera que cette figure aurait aussi pu être obtenue par la commande suivante, plus explicite mais moins concise.

```
plt.plot(x,y,marker='o',color='r',linestyle='--')
```

▪

On peut enfin appeler plusieurs fois la fonction `plot` successivement. Les tracés sont alors superposés.

▪ **Exemple** Le code suivant produit la figure 3.

```
import matplotlib.pyplot as plt

x = [1.5, 3, 3.5, 2]
y = [3, 2, 4, 2]

plt.clf()
plt.plot(x,y,'or')
plt.plot([1,4],[1,4])
plt.savefig("ex_base_03.png")
```

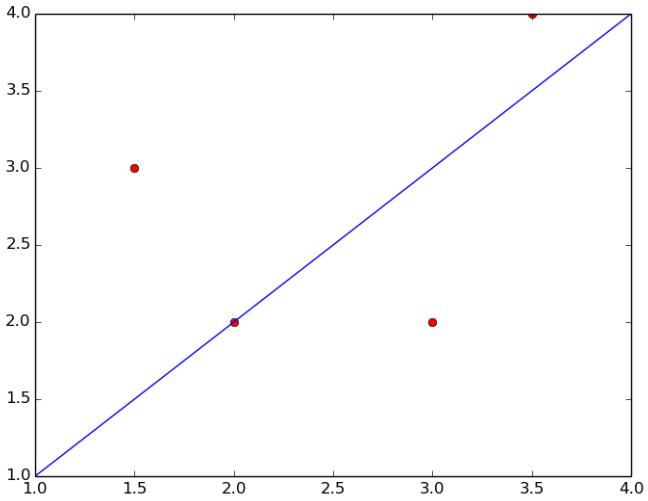

FIGURE 3 – Supperposition de deux tracés

▪

R Il n'est pas réellement possible de tracer des courbes lisses, mais seulement des lignes brisées et des nuages de points.

2.2 Quelques options graphiques disponibles

La fonction `plot()` possède énormément d'options, nous n'en détaillerons que quelques-unes ici. On suppose dans cette partie que la bibliothèque `matplotlib.pyplot` a été importée avec l'alias `plt`.

2.2.1 Couleurs (option `color`)

On peut décrire de nombreuses couleurs très précisément, mais certaines sont disponibles avec les raccourcis suivants. L'argument donné est une chaîne de caractères et la couleur par défaut est le bleu.

▪ **Exemple** Pour tracer une courbe en magenta, on écrira la commande suivante.

```
plt.plot(x,y,color = 'm')
```

▪

2.2.2 Étiquette (option `label`)

On peut adjoindre à un tracé une chaîne de caractères (son étiquette), qui pourra notamment être utilisée dans les légendes. Les parties entres deux symboles $ sont interprétées comme du code LaTeX (attention, en Python, le caractère \ se code par le caractère `'\\'`). Par défaut, il n'y a pas d'étiquette.

Mot clef	Couleur
'b'	Bleu
'g'	Vert
'r'	Rouge
'c'	Cyan
'm'	Magenta
'y'	Jaune
'k'	Noir
'w'	Blanc

TABLE 1 – Couleurs prédéfinies dans `matplotlib`.

▪ **Exemple** Pour donner le nom « $\arccos(t)$ » à la courbe, on écrira la commande suivante.

```
plt.plot(x,y,label = '$\\arccos(t)$')
```

▪

2.2.3 Type de ligne (option `linestyle`)

On peut spécifier plusieurs types de lignes, qui seront utilisés pour tracer les segments reliant les points donnés en argument. L'argument donné est une chaîne de caractères et, par défaut, la ligne est tracée continuement.

Mot clef	Type de ligne
'-'	Ligne continue
'- -'	Tirets
'-.'	Alterne tirets et points
':'	Ligne pointillée
'None' ou '' ou ''	Pas de ligne

TABLE 2 – Types de ligne disponibles.

▪ **Exemple** Pour tracer une courbe en pointillés, on écrira la commande suivante.

```
plt.plot(x,y,linestyle = ':')
```

▪

2.2.4 Épaisseur des lignes (option `linewidth`)

On peut régler l'épaisseur des lignes reliant les points. L'argument donné est un flottant, qui vaut 1 par défaut.

▪ **Exemple** Pour tracer une courbe dont le trait est deux fois plus épais que la normale, on écrira la commande suivante.

```
plt.plot(x,y,linewidth = 2)
```

▪

2.2.5 Type de point (option `marker`)

On peut changer la manière dont les points sont représentés. L'argument donné est une chaîne de caractères et, par défaut, les points ne sont pas représentés.

R La liste donnée dans la table 3 est loin d'être exhaustive.

▪ **Exemple** Pour placer des carrés à l'emplacement des points indiqués, on écrira la commande suivante.

```
plt.plot(x,y,marker = 's')
```

▪

Mot clef	Type de point
'.'	Point
'o'	Cercle
's'	Carré
'*'	Étoile
'+'	Croix (forme +)
'x'	Croix (forme ×)
'None' ou ' ' ou ''	Pas de point
'$...$'	Texte Latex (à la place de ...)

TABLE 3 – Quelques types de points disponibles.

R Les options `markeredgecolor`, `markeredgwidth`, `markerfacecolor` et `markersize` permettent de paramétrer plus finement l'apparence des points.

2.3 Autres fonctions utiles

Nous avons déjà vu comment utiliser de manière élémentaire les fonctions `clf()`, `plot()` et `savefig()`.

2.3.1 Paramètres avancés d'un graphique

On peut affecter une étiquette à chaque axe avec les fonctions `xlabel()` et `ylabel()`.

Les fonctions `xlim()` et `ylim()` permettent de régler manuellement les étendues des deux axes, et donc la forme du graphique. On pourra utiliser à la place la fonction `axis()`. On pourra aussi utiliser la fonction `autoscale()`. Les fonctions `xticks()` et `yticks()` permettent de régler les graduations des axes.

La fonction `figtext()` permet d'écrire du texte dans une figure.

La fonction `legend()` permet d'ajouter une légende à une figure, la fonction `title()` permet d'ajouter un titre à une figure.

Les commandes `xscale('log')` et `yscale('log')` permettent de passer les axes des abscisses ou des ordonnées en échelle logarithmique.

2.3.2 Quelques outils utiles en statistiques

Cette partie ne sera pas utilisée en informatique, mais pourra vous intéresser dans le cadre de votre TIPE, notamment.

Les fonctions `barh()` et `hist()` permettent de réaliser des histogrammes, la fonction `boxplot()` permet de réaliser des diagrammes en boîtes à moustaches.

La fonction `errorbar()` permet de tracer un graphe avec des barres d'erreurs (intervalles de confiances).

2.3.3 Subdivision régulière d'un intervalle

Pour tracer une fonction, on trace en fait des cordes de cette fonction, à partir d'une subdivision de l'intervalle considéré. La fonction `linespace()` de la bibliothèque `numpy` permet de créer automatiquement de telles subdivisions, avec un nombre de points à régler manuellement.

▪ **Exemple** Pour tracer le graphe de la fonction $x \mapsto x^3$ sur $[0,1]$ avec une subdivision en 3 segments, donc en utilisant 4 points, on peut utiliser le code suivant, qui donne la figure 4.

```
import matplotlib.pyplot as plt
from numpy import linspace

x = linspace(0,1,4)
y = [t**3 for t in x]

plt.clf()
plt.plot(x,y)
plt.xlabel('$t$')
plt.ylabel('$t^3$')
plt.savefig('ex_numpy_01.png')
```

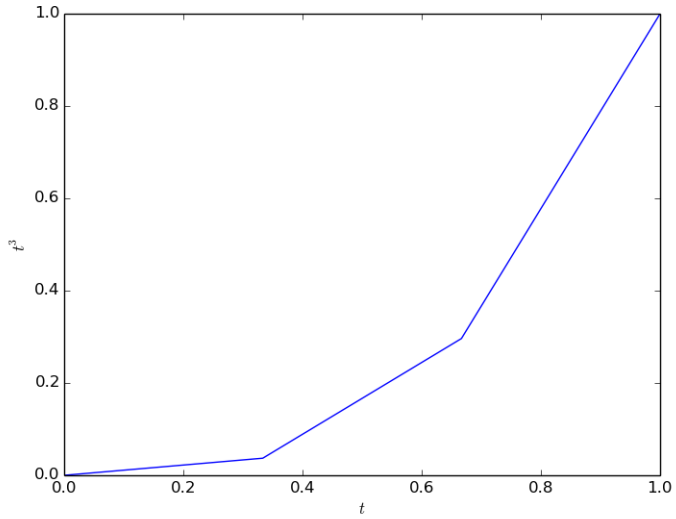

FIGURE 4 – Subdivision en 3 segments

Pour obtenir une courbe plus lisse, on peut utiliser une subdivision plus fine. Par exemple, avec 100 segments, et donc 101 points, on peut utiliser le code suivant, qui donne la figure 5.

```
import matplotlib.pyplot as plt
from numpy import linspace

x = linspace(0,1,101)
y = [t**3 for t in x]

plt.clf()
plt.plot(x,y)
plt.xlabel('$t$')
plt.ylabel('$t^3$')
plt.savefig('ex_numpy_01.png')
```

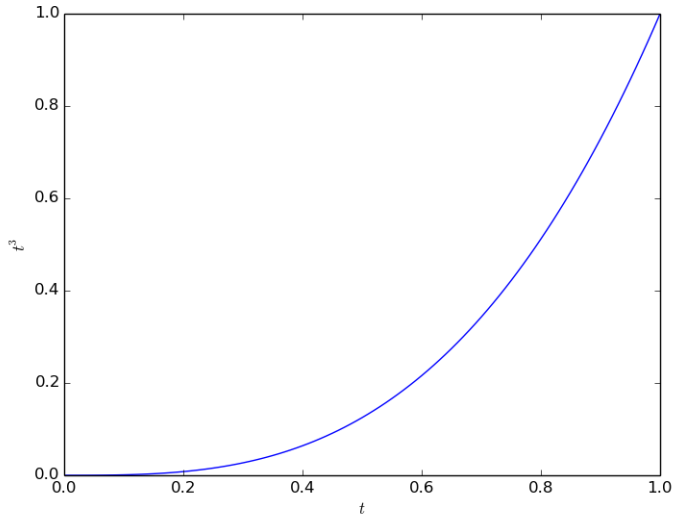

FIGURE 5 – Subdivision en 100 segments

■