

TP 04

Utilisation de modules de bibliothèques

Savoirs et compétences :

- Th. 3 : Utilisation de modules, de bibliothèques.

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")
```

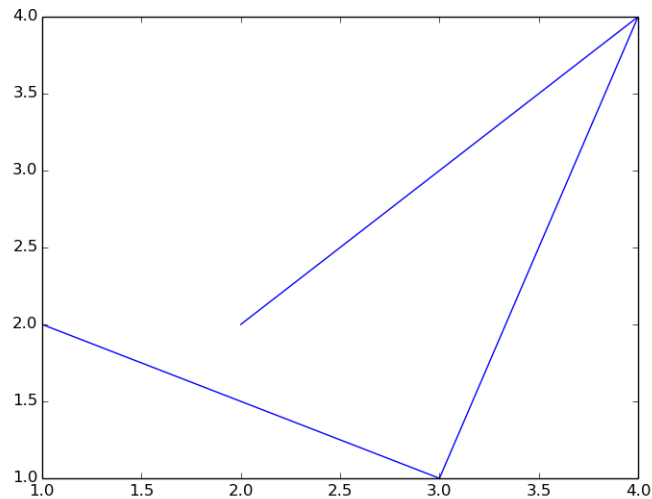


FIGURE 1: Style standard

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: Superposition 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 entre 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 continuellement.

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`, `markeredgewidth`, `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 `linspace()` 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')
```

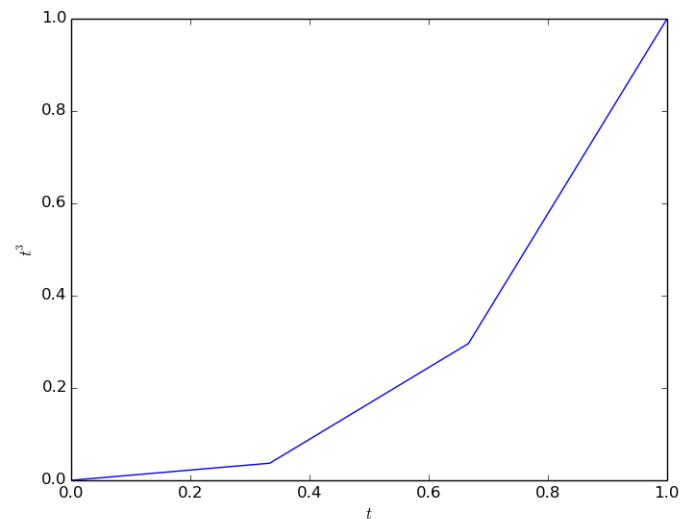


FIGURE 4: Subdivision en 3 segments

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')
```

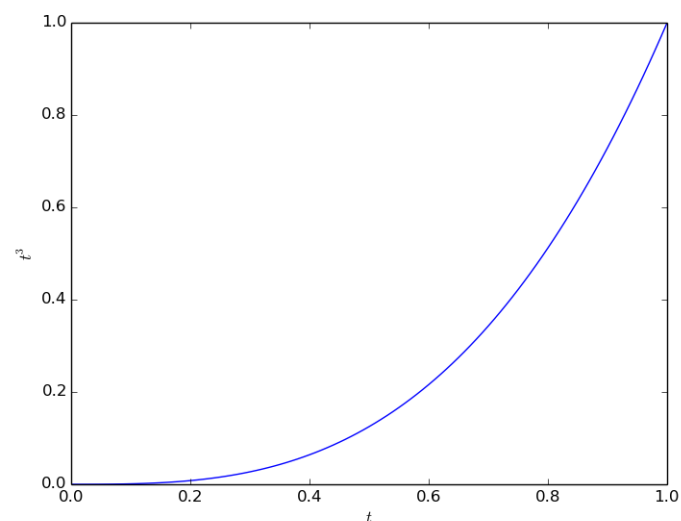


FIGURE 5: Subdivision en 100 segments

Activité 2 – Tracé de fonctions simples

En utilisant Python on peut tracer de nombreux types de graphiques. Nous allons utiliser une bibliothèque

regroupant de (très) nombreuses fonctions de tracé : `matplotlib`. En fait, cette bibliothèque est bien trop vaste, nous n'utiliserons que sa sous-bibliothèque `matplotlib.pyplot`.

Commencez par ouvrir votre IDE, puis créez un script nommé `tp05_ex_sin.py`. Recopiez dedans le script suivant.

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

n = 20
x = [k*10/n for k in range(n)]
y = [sin(t) for t in x]

plt.clf()
plt.plot(x,y,label='sin(x)')
plt.xlabel('x')
plt.legend(loc=0)
plt.title('Tracé du sinus sur [0,10]')
plt.savefig('tp05_ex_sin.png')
```

Exécutez ce script et vérifiez que la figure créée est en tout point semblable à celle présente sur le site de classe.

Question 1

Quel est le type de `x`?

Question 2

Comment Python représente-t-il graphiquement une fonction?

Question 3

Où le tracé s'arrête-t-il? Pourquoi?

On rappelle que l'on peut simplement créer une liste d'abscisses en utilisant la fonction `linspace` de la bibliothèque de calcul `numpy`.

Chargez cette fonction dans l'interpréteur interactif par la commande

```
from numpy import linspace
```

puis consultez son manuel par la commande `help(linspace)`.

Question 4

Écrire une fonction `ex_sin(nom_de_fichier)` permettant tracer de manière plus appropriée la courbe de l'exemple précédent et qui enregistre l'image produite dans le fichier `nom_de_fichier`. Vous produirez alors une image, que vous enverrez à votre enseignant.

Indication : Attention au type de `nom_de_fichier`! Notamment, on supposera que l'extension du fichier est déjà présente dans `nom_de_fichier`.

• Question 5 <

Écrire une fonction `transitoire(A,nom_de_fichier)` qui enregistre dans le fichier `nom_de_fichier` le graphe des fonctions

$$t \rightarrow A \left(1 - e^{-\frac{t}{\tau}} \right)$$

sur $[0, 10]$, pour chaque $\tau \in \left\{ \frac{1}{2}; 1; 2; 4; 8 \right\}$. Vous produirez une image (vous choisirez A), que vous enverrez à votre enseignant.

Indication : Attention au type de `nom_de_fichier`!

Activité 3 –

2.4 Introduction

Dans cette partie, on va s'appuyer sur un support de TP qui est utilisé aux oraux de concours : le système CoMax présenté sur la figure 1. Ce système est une adaptation pédagogique de la solution industrielle ZE de SAPELEM. Le principe de fonctionnement de ces deux systèmes repose sur l'utilisation d'un système de levage motorisé, associé à une poignée, équipée d'un capteur d'effort (voir Figure 6).

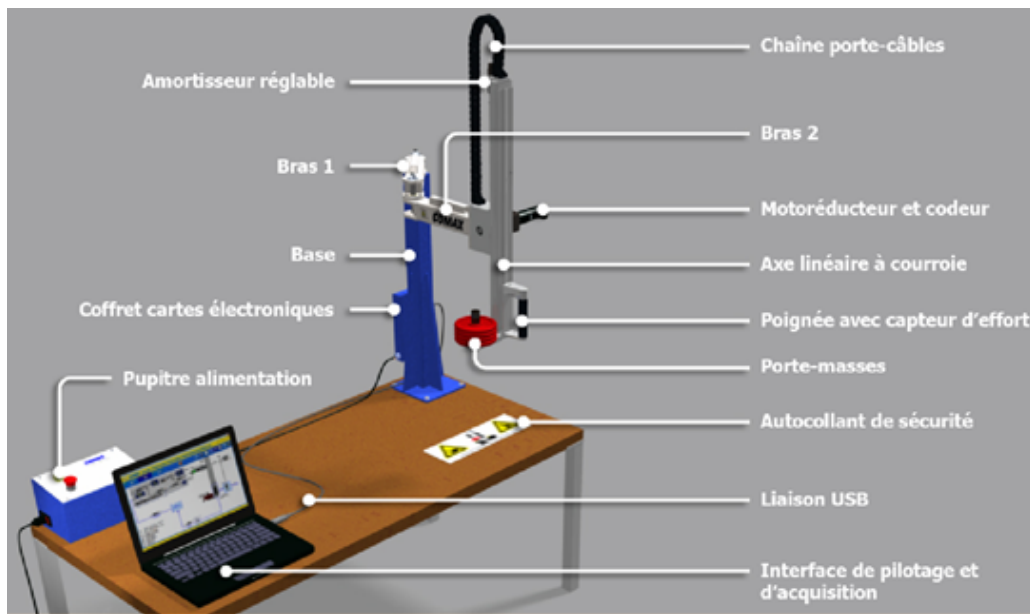


FIGURE 6: Présentation du système Comax

2.5 Extraction des données expérimentales

Nous allons dans cette partie utiliser les résultats d'expérimentations sur le robot CoMax. Ces résultats sont fournis dans le fichier *CoMax.txt*. Ils correspondent à un fichier brut avec les points de mesures expérimentales.

Question 6 Après avoir observé la structure du fichier *CoMax.txt*, proposer des instructions permettant d'extraire sous deux listes distinctes `temps` et `q_exp` les instants de prise d'échantillonnage et les positions codeur correspondantes en tops (nombre de points). Ces deux listes seront converties en tableaux Numpy (`import numpy as np`), grâce à la commande `liste=np.array(liste)`.

La position de l'axe linéaire $X(t)$ [en mm] est liée à la position du moteur q_m [en tops] renvoyée par le codeur selon la formule suivante : $X = \frac{3,41 \cdot q_m}{1000}$

Question 7 Générer alors un tableau des positions verticales de l'axe nommé `X_exp`, qui a une condition initiale nulle sur la position.

Question 8 Tracer l'évolution des positions mesurées expérimentales en fonction du temps, avec des croix bleues (+). On légendera correctement les axes, et on indiquera une légende du type : "points mesurés"

2.6 Étude des performances attendues du système

Dans un second temps, nous allons modéliser le comportement attendu système. La modélisation du système est faite en amont du système réel, lors de la phase de conception, mais elle est importante pour comprendre un système réel afin de proposer des modifications affectant les performances.

Les réponses attendues en vitesse $V(t)$ et en position $X(t)$ de l'axe linéaire sont représentées en Figure 7. Les caractéristiques de la loi en vitesse sont les suivantes :

- l'instant de début de mouvement : $t_0 = 0s$;
- la position initiale et la vitesse initiale : $X(0) = X_0 = 0m$ et $V(0) = 0m/s$;
- l'accélération : $A_{cmax} = 2,83m/s^2$;
- la vitesse maximale : $V_{max} = 0,68m/s$.

On peut montrer par intégration successives, en tenant compte des conditions initiales que :

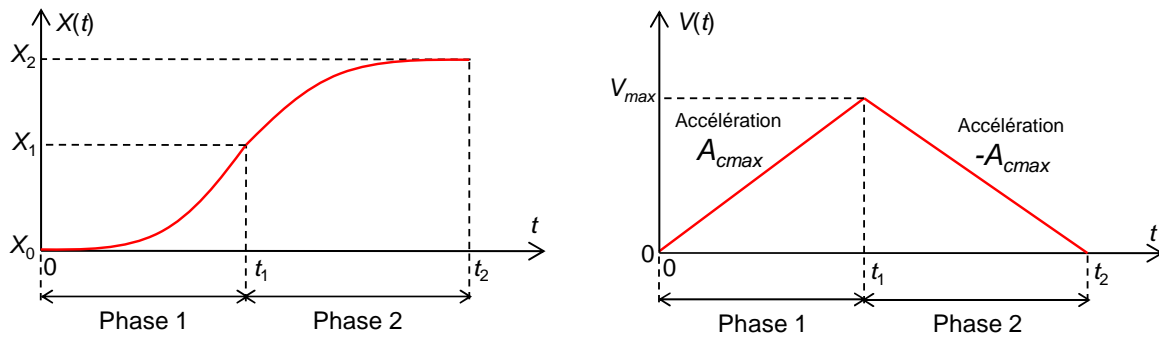


FIGURE 7: Lois d'évolution en position (à gauche) et en vitesse (à droite) du CoMax

Phase	1	2
instant t	$0 \leq t < t_1$	$t_1 \leq t \leq t_2$
accélération $A(t)$	A_{cmax}	$-A_{cmax}$
vitesse $V(t)$	$A_{cmax} \cdot t$	$V_{max} - A_{cmax} \cdot (t - t_1)$
position $X(t)$	$\frac{1}{2} \cdot A_{cmax} \cdot t^2$	$X_1 + V_{max} \cdot (t - t_1) - \frac{1}{2} \cdot A_{cmax} \cdot (t - t_1)^2$

On notera que : $X_1 = X(t_1)$, à l'issue de la phase 1 et $X_2 = X(t_2)$, à l'issue de la phase 2.

Question 9 Trouver les expressions littérales de t_1 , t_2 , X_1 et de X_2 en fonction de A_{cmax} et de V_{max} .

Question 10 Concevoir deux fonctions `Loi_Vitesse` et `Loi_position` prenant en argument un instant t et permettant de retourner la vitesse, respectivement la position, à cet instant. (A noter que l'on pourra introduire des variables globales, comme t_1 , t_2 , etc.)

On rappelle que la commande `y = np.zeros(n)` permet de générer un tableau de taille $n \times 1$ contenant des 0.

Question 11 Construire deux tableaux `X_th` et `V_th` où sont stockées les positions théoriques commandées, respectivement vitesses théoriques aux instants définis dans le tableau `temps`. Superposer la courbe d'évolution de la position théorique sur les points expérimentaux, obtenus précédemment (tracé en vert, ligne continue, avec légende explicite). Sur une nouvelle figure tracer en vert, trait continu, l'évolution de la vitesse théorique

2.7 Quantification et analyse des écarts

Nous allons quantifier l'écart de performance entre l'exigence du cahier des charges et le système réel. Pour cela, nous allons calculer les écarts et utiliser des outils de statistique pour quantifier ces écarts.

On définit l'écart relatif en % $\delta_{\%}$ entre une valeur théorique x_{th} et une valeur expérimentale x_{exp} : $\delta_{\%} =$

$$\left| \frac{x_{exp} - x_{th}}{x_{th}} \right|$$

Question 12 Concevoir une fonction `Calcul_ecarts` prenant en arguments deux tableaux à une dimension et retournant un tableau `Delta_X` de même dimension où sont stockés les écarts relatifs entre chacune des valeurs des deux tableaux spécifiés en arguments d'entrée.

Question 13 Tracer un histogramme montrant l'évolution des écarts relatifs en position en fonction du numéro de la mesure (on utilisera `plt.bar` et `plt.subplot`). On n'évaluera pas l'écart relatif sur la première valeur (nulle).

On rappelle ici quelques notions de statistiques :

- La médiane d'un ensemble de valeurs (échantillon, population, distribution de probabilités) est une valeur qui partage la série en deux parties de même effectif.

- L'écart type est défini comme la moyenne quadratique des écarts par rapport à la moyenne. Il a la même dimension que la variable statistique étudiée.

Question 14 Écrire une fonction `Calculs_stats` permettant, à partir d'un tableau `T` passé en argument, de retourner un tuple de 3 valeurs : moyenne, médiane et écart type. Indication : On pourra utiliser les fonctions de la bibliothèque `numpy` (`np.sum(T)`, pour faire la somme de tous les éléments du tableau `T`, `np.sort(T)`, pour trier le tableau `T` dans l'ordre croissant. Appliquer le résultat au tableau `Delta_X`.

Question 15 Comparer les résultats en utilisant les fonctions suivantes : `np.mean`, pour la moyenne ; `np.median`, pour la médiane et `np.std`, pour l'écart type