CPGE - Sciences Industrielles de l'Ingénieur

Ressource UPSTI
Réforme des programmes
2021

TP

TP5 – Bibliothèques, fichiers, graphiques

V1

Table des matières

I Mise en situation

On s'intéresse dans ce sujet aux Entrées/Sorties fichier, autrement dit à la lecture ou écriture d'un fichier, ainsi qu'à l'utilisation de modules ou bibliothèques, utiles par exemple dans la réalisation de courbes, graphiques, histogrammes, etc.

I.1 Répertoire de travail courant

En règle général, il peut s'avérer nécessaire de travailler dans un répertoire de travail commun où l'on regroupe es scripts python et le fichiers à exploiter (ou alors si les fichiers sont nombreux on peut les inclure dans un sous-dossier). Quoi qu'il en soit, il faut définir un répertoire de travail courant. En effet, lorsque l'on cherche à accéder à un fichier dans un script ou dans la console, PYTHON ira chercher ce fichier dans le répertoire de travail courant (« current work directory »).

Le module `os`, qui s'importe en faisant `import os` contient deux fonctions permettant de connaitre et de modifier le répertoire de travail courant :

— La fonction `os.getcwd` (« get current work directory ») ne prend aucun argument et retourne une chaine de caractère donnant le chemin du répertoire de travail courant.

— La fonction `os.chdir` (« change directory ») prend comme argument une chaine de caractère correspondant à un chemin de répertoire et définit ce répertoire comme étant le répertoire de travail courant.

On notera qu'il faut mettre deux « \ » pour indiquer l'arborescence complète. Par exemple, pour indiquer que répertoire de travail courant est « D :\Dossier\Sous-dossier », on saisira :

```
import os
os.chdir("D:\\Dossier\\Sous-dossier")
```

I.2 Lecture / écriture de fichiers

L'ouverture d'un fichier se fait grâce à la fonction : `open(nom, mode)`. `nom` : Chaîne de caractère, nom du fichier `mode` : Chaîne de caractère, mode accès au fichier (`'r'` : read, `'w'` : write)

La fonction open retourne un objet de type file *itérable*. Les principales méthodes de la classe file sont les suivantes :

`.read()`	retourne tout le fichier comme un `str`
`.read(n)`	retourne un `str` d'au moins n octets
`.readline()`	retourne une ligne
`.readlines()`	retourne la liste de toutes les lignes du fichier
`.writeslines(lines)`	écrit `lines` (liste de `str`) dans le fichier
`.close()`	ferme le fichier

Afin de tester quelques fonctionnalités, on disposera du fichier `data1.txt`. On pourra l'ouvrir de la façon suivante et nommer l'objet associé « MonFichier » :

```
MonFichier = open("data1.txt",'r')
```

Q1. Tester et commenter les lignes de code suivantes, après avoir observé la structure du fichier `data1.txt` sous python ou tout autre éditeur de texte.

```
myFile = open("data1.txt",'r')
data = []
for lu in myFile:
    if lu[0] != '#':
        data.append(lu)
myFile.close()
```

Afin de séparer les termes de chaque élément de la liste précédemment obtenue, nous allons utiliser la méthode `.split()` associée aux chaînes de caractères. Si on ne précise rien en paramètres de la méthode, tout espace blanc est séparateur (espaces, tabulations (\t), retour à la ligne (\n),...). Toutefois elle peut prendre deux paramètres optionnels : en chaîne de caractère, le symbole séparateur et un nombre, indiquant en combien de morceaux on souhaite scinder la chaîne au maximum.

Q2. Tester les lignes suivantes et les commenter :

```
t, temperature = [], []
for lu in data:
    (x1,x2)=lu.split()
    t.append(float(x1))
    temperature.append(float(x2))
```

Il est également possible d'écrire dans un fichier, mais nous ne développerons pas ici les techniques qui s'y rapportent.

I.3 Tracés de graphes

FIGURE 1 – Page d'accueil du site internet sur Matplotlib

Le tracé de courbes est effectué grâce au très riche module matplotlib, qui propose des outils de tracés. Le site internet dédié à ce module (https ://matplotlib.org/) vous montrera entre autre la galerie impressionnante de tracés possibles (cliquer sur l'ongle « Examples » sur la page d'accueil du site), ainsi que toute la documentation (onglet « Documentation »).

I.3.1 Premier exemple

Pour tracer une courbe, il faut dans un premier temps ouvrir une fenêtre graphique dans laquelle elle sera effectivement tracée. Voici un premier exemple :

```
1 import matplotlib.pyplot as plt
2 plt.figure() plt.title("Titre de la figure")
3 plt.xlabel('axe des abcisses')
4 plt.xlim(0,10)
5 plt.ylabel('axe des ordonnees en bleu', color='b')
6 plt.ylim(0,100)
7 plt.show()
```

Q3. Tester les lignes précédentes et les commenter.

Une courbe peut alors être tracée grâce à la ligne de commande `plt.plot(x,y)`. `x` et `y` représentent une liste de flottants (`y` ordonnées des abscisses de `x`). Bien entendu, le nombre d'éléments de chaque liste doit être identique. `x` et `y` peuvent également être des vecteurs. Enfin, on peut ajouter autant de courbes que l'on souhaite sur la même fenêtre, en saisissant à la suite la fonction `plt.plot`...

Il est possible d'ajouter sur la courbe tracée, des motifs, tels que des croix, points, etc... Pour connaître toutes les options, le mieux est de se référer à la documentation de matplotlib. Voyons ici quelques-unes d'entre elles :

- Création d'une nouvelle figure : on commence par créer une nouvelle figure pour y insérer des courbes ou autres graphes. Cela se fait avec `plt.figure()`. A noter que l'on peut éventuellement spécifier un numéro de figure ; par exemple `plt.figure(5)`. Il sera alors possible de tracer des courbes sur cette figure en appelant `plt.figure(5)`.
- bornes : spécifier un rectangle de représentation, ce qui permet un zoom, Permettant de restreindre la tracé sur une zone donnée, se fait via la commande `plt.axis([xmin,xmax,ymin,ymax])` ;
- couleur du trait : pour changer la couleur du tracé une lettre `g` vert (green), `r` rouge (red), `k` noir, `b` bleu, `c` cyan, `m` magenta, `y` jaune (yellow), `w` blanc (white). `plt.plot(...,'r')` tracera une courbe en rouge. Pour avoir encore plus de couleurs, on pourra utiliser le mot clé `color=[R,G,B]` avec les 3 paramètres flottants R (Red), G (Green) et B (Blue) flottants compris entre 0 et 1, pour indiquer leur niveau.
- symboles : mettre des symboles aux points tracés se fait via l'option `marker`. Les possibilités sont nombreuses parmi `'+'`, `'*'`, `','`, `'.'`, et bien d'autres...
- style du trait : pointillés, absences de trait, etc se décident avec `linestyle`. Au choix `'-'`, ligne continue, `'--'` tirets, `'-.'` points-tirets, `':'` pointillés, etc.
- épaisseur du trait : `linewidth` = flottant (comme par exemple : `linewidth=2`).
- taille des symboles (markers) : `markersize` = flottant comme pour l'épaisseur du trait. D'autres paramètres sont modifiables, comme `markeredgecolor` pour la couleur du trait du pourtour du marker, `markerfacecolor` pour la couleur de l'intérieur (si le marker possède un intérieur comme `'o'`), `markeredgsize` = flottant pour l'épaisseur du trait du pourtour du *marker*. On pourra remarquer que si la couleur n'est pas spécifiée pour chaque nouvel appel la couleur des *markers* change de façon cyclique.
- étiquettes sur les axes des abcisses et ordonnées : Matplotlib décide tout seul des graduations sur les axes. Tout ceci se modifie via : `plt.xticks(tf)`, `plt.yticks(tf)` où `tf` est un vecteur de flottants ordonnés de façon croissante.

— ajouter un titre : `plt.title("Mon titre")`.

— légendes : il faut ajouter dans les arguments de chaque tracé, le mot clef `label` : `label='nom de la courbe'` : `plt.plot(x,y,label='nom')`. Puis : `plt.legend()`.

Q4. Proposer une suite d'instructions permettant de tracer la courbe suivante, en respectant légendes et titres. La courbe est tracée en rouge.

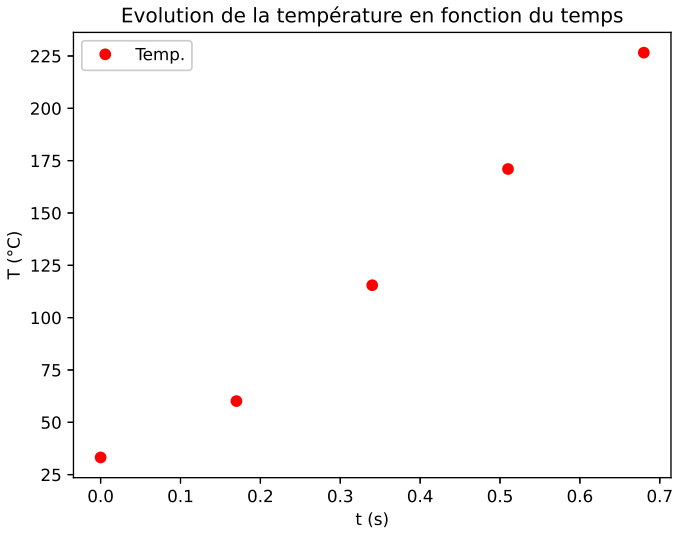

FIGURE 2 – Courbe à tracer

Il est également possible de tracer des histogrammes avec la commande `plt.hist` ou la commande `plt.bar`.

Q5. Tester et commenter les lignes de code suivantes :

```
import random as rd
import matplotlib.pyplot as plt
y=[]
for i in range(1,201):
    y.append((rd.randint(1,100)))
plt.figure()
plt.subplot(211)
plt.hist(y, 10, color='r')
plt.subplot(212)
plt.bar([i for i in range(200)], y, color=[0.2,0.5,0.8])
plt.show()
```

II Exploiter les données enregistrées sur le système Comax

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 3).

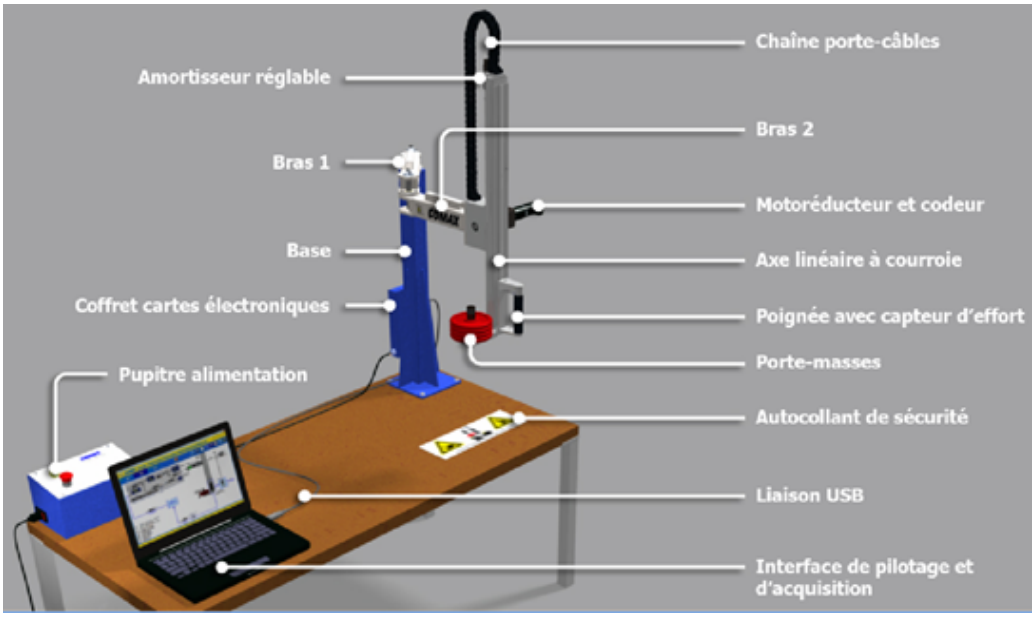

FIGURE 3 – Présentation du système Comax

II.1 Extraction des données expérimentales

Nous allons dans cette partie utiliser les résultats d'expérimentions 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.

Q6. 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 : $X = \dfrac{3,41.q_m}{1000}$

Q7. Générer alors un tableau des positions verticales de l'axe nommé `X_exp`, qui a une condition initiale nulle sur la position.

Q8. 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"

II.2 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 4 . 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 : $V_{max} = 0,68m/s$.

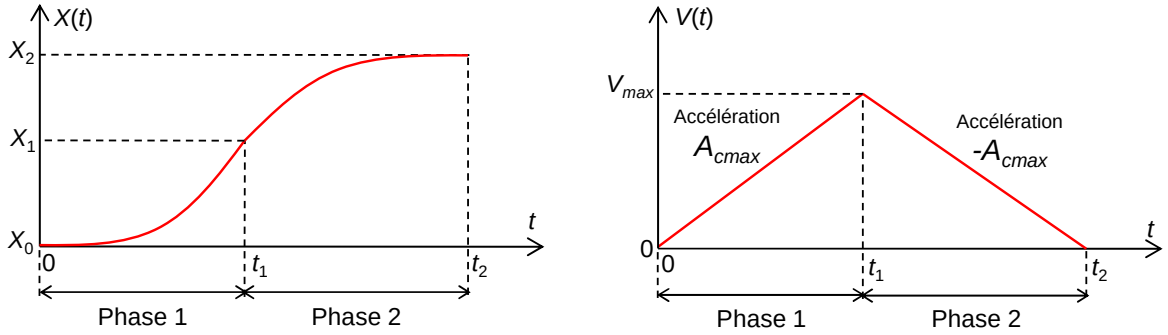

FIGURE 4 – Lois d'évolution en position (à gauche) et en vitesse (à droite) du CoMax

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

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 $V(t)$	$A_{cmax}.t$	$V_{max} - A_{cmax}.(t - t_1)$
position $X(t)$	$\frac{1}{2}.A_{cmax}.t^2$	$X_1 + V_{max}.(t - t_1) - \frac{1}{2}.A_{cmax}.(t - t_1)^2$

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.

Q9. Trouver les expressions littérales de t_1, t_2, X_1 et de X_2 en fonction de A_{cmax} et de V_{max}.

Q10. 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.

Q11. 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.

II.3 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} : $\delta_{\%} = \left| \dfrac{x_{exp} - x_{th}}{x_{th}} \right|$

Q12. 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.

Q13. 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.

Q14. 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`.

Q15. 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