

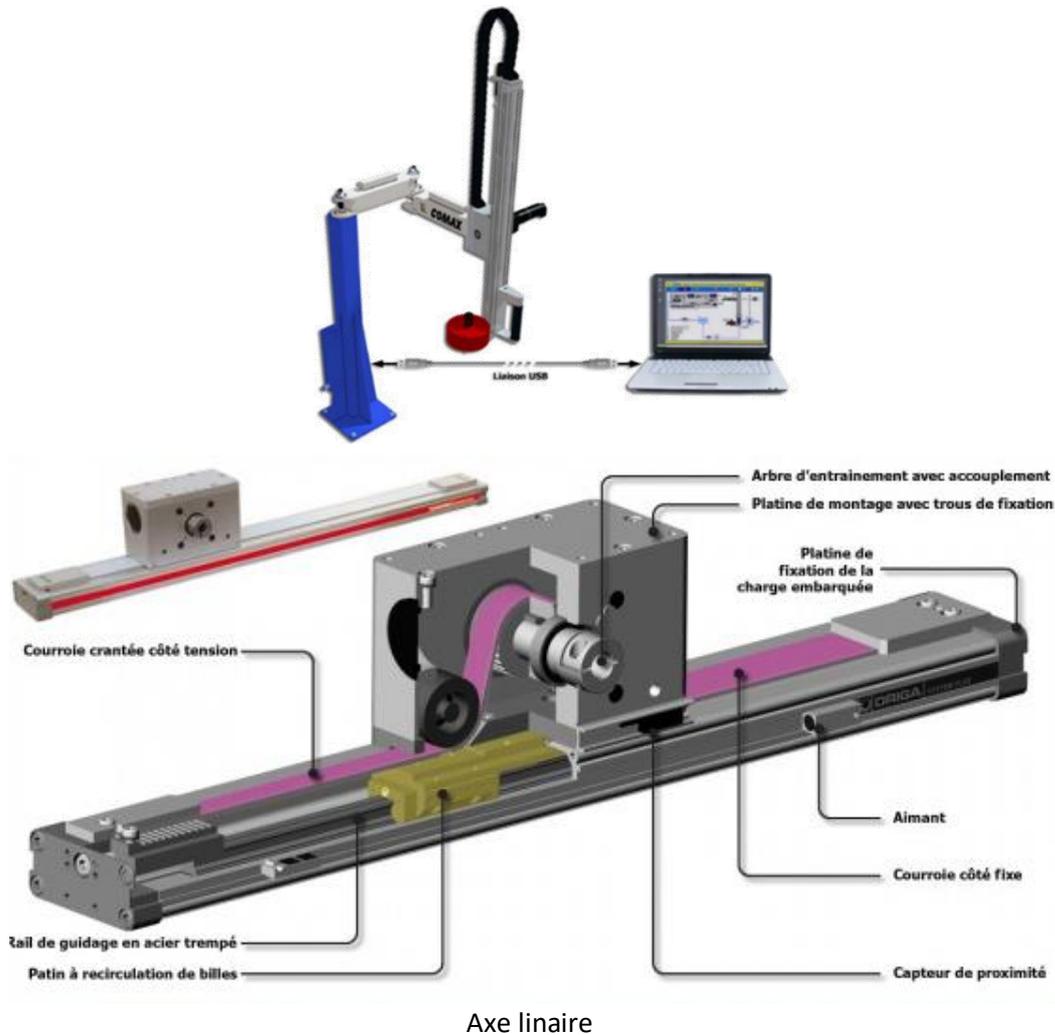
Informatique

Intelligence artificielle

Régression linéaire multiple – Comax

Comax

Le robot Comax est un robot collaboratif didactisé permettant à un opérateur de porter des charges lourdes sans efforts. Un moteur à courant continu (MCC) est asservi et pilote la translation de l'ensemble en translation noté E , de masse m , par l'intermédiaire d'un réducteur de rapport de réduction K_{red} et suivi d'un axe linéaire (poulie/courroie) de rapport de réduction K_{al} .



Données pour la suite :

- On note $K = \frac{V}{\omega} = K_{red}K_{al}$ le rapport cinématique entre la rotation moteur ω et la V vitesse de translation de E
- L'axe linéaire provoque une translation de E de $108 \text{ mm} \cdot \text{tour}^{-1}$
- La constante de couple du MCC vaut $k_c = 0,0302 \text{ Nm} \cdot \text{A}^{-1}$
- L'accélération de la pesanteur vaut $g = 9,81 \text{ m} \cdot \text{s}^{-2}$
- Le rapport de réduction du réducteur vaut 15,88

Application de la RLM

Dans un grand nombre de systèmes asservis comme le Comax, le couple C_{ext} d'une action résistante sur la sortie (gravité sur E) ramené à l'axe moteur est constant.

L'application du principe fondamental de la dynamique en régime stationnaire (à vitesse constante pour supprimer les effets d'inertie) permet de déterminer la relation en couples sur l'axe moteur :

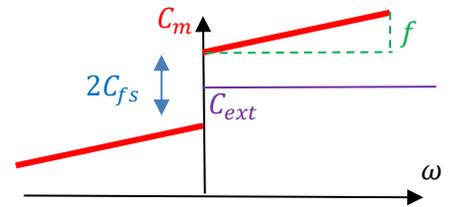
$$C_m = k_c I = C_{ext} \pm C_{f_s} + C_{f_v} = C_{ext} \pm C_{f_s} + f \omega_\infty$$

Avec :

- k_c la constante de couple du MCC
- I l'intensité du moteur
- C_{f_s} le couple de frottements sec (lois de Coulomb) constant opposé au mouvement
- C_{f_v} le couple des frottements visqueux proportionnels à la vitesse de rotation (coefficient f)
- C_{ext} le couple de gravité sur E ramené à l'axe moteur

Pour une masse m donnée, on a :

$$C_m = \begin{cases} F(\omega) = a\omega + b^1 & \text{si } \omega > 0 \\ f(\omega) = a\omega + b^2 & \text{si } \omega < 0 \end{cases} ; \quad \begin{cases} a = f \\ b^1 = C_{ext} + C_{f_s} \\ b^2 = C_{ext} - C_{f_s} \end{cases}$$

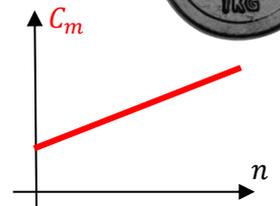


On obtient donc une évolution linéaire à décomposer en deux demi-droites côté $\omega < 0$ et $\omega > 0$.

La masse de E est écrite $m = m_0 + nm_i$ avec m_0 la masse de la partie en translation de l'axe linéaire seul et n le nombre de masse ajoutées de masse $m_i = 1kg$, soit finalement : $m = m_0 + n$



Si on se place à une vitesse constante, on s'attend à obtenir une évolution du couple moteur linéaire par rapport à la masse embarquée :



On voit donc apparaître la notion de régression linéaire multiple avec un couple moteur C_m , donc une intensité moteur I , qui sera de la forme :

$$I = \begin{cases} f(n, \omega) & \text{si } \omega < 0 \\ F(n, \omega) & \text{si } \omega > 0 \end{cases}$$

Nous allons mettre en place deux modèles RLM de prédiction de l'intensité moteur en fonction du couple $[n, \omega]$, l'un pour $\omega < 0$, l'autre pour $\omega > 0$. Il sera alors possible d'obtenir des valeurs assez fiables de :

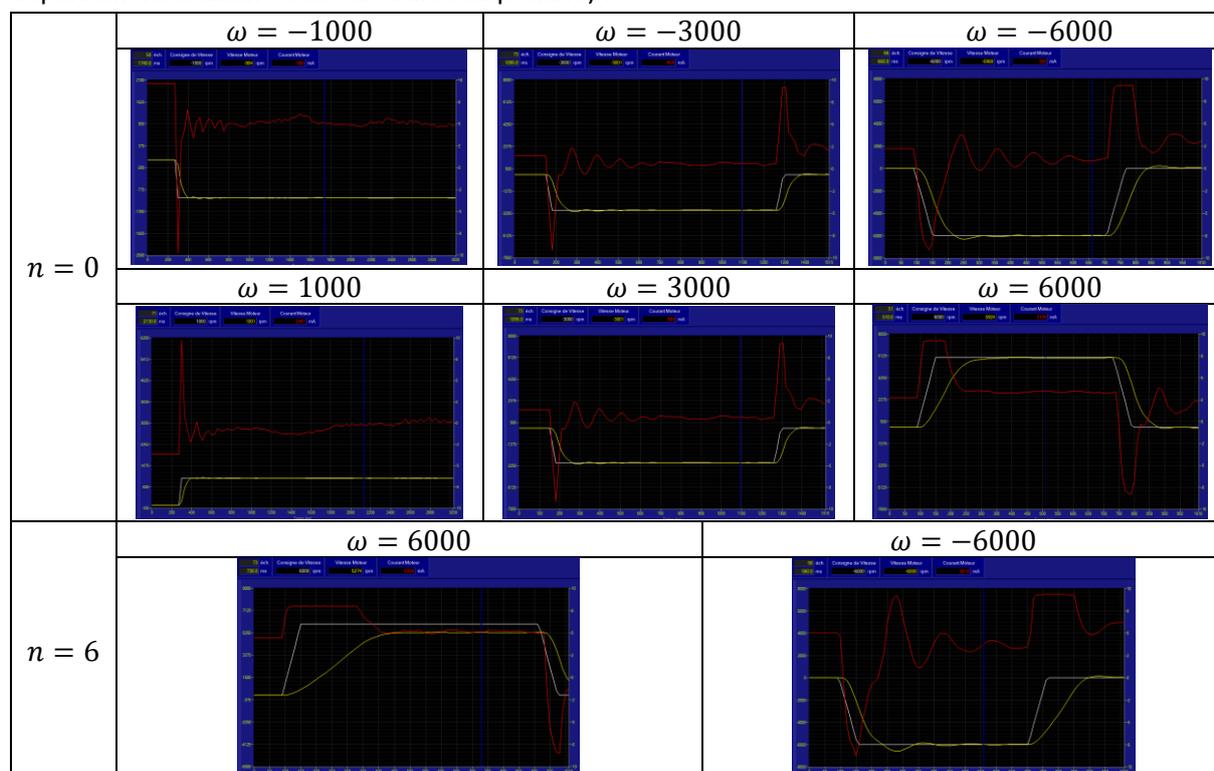
- La masse inconnue m_0
- Le couple de frottements secs inconnu C_{f_s}
- Le coefficient des frottements visqueux inconnu f
- La masse des masses ajoutées attendue à 1kg

Méthode de régression multilinéaire

Cette méthode diffère des méthodes knn et rn par le fait qu'il faut que les résultats présentent une linéarité vis-à-vis des données. Exactement comme pour une régression linéaire à un seul paramètre x qui associe une courbe $y = ax + b$ avec un coefficient de corrélation, on donne en entrée d'une régression linéaire multiple un ensemble de données à n dimensions $\{x_1, x_2, \dots, x_n\}$ afin de prévoir une valeur résultat y pour chacune de ces données.

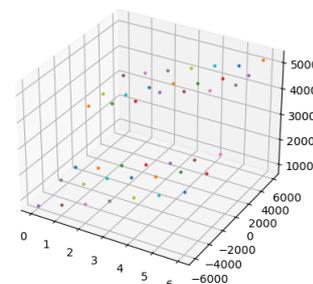
Données

Une campagne de mesures a été réalisée sur le système afin d'obtenir, pour des consignes de vitesse de rotation ω en $tr.min^{-1}$ dans la liste $l\omega = [-6000, -5000, -4000, -3000, -2000, -1000]$ et dans la liste $L\omega = [1000, 2000, 3000, 4000, 5000, 6000]$ et pour un nombre de masses embarquées n dans la liste $Ln = [0, 1, 2, 3, 4, 5, 6]$. On a alors relevé l'intensité moteur « moyenne » en régime permanent et, quand la vitesse de consigne n'était pas atteinte, la vitesse atteinte. Voici un extrait de cette campagne (on a veillé à adapter le paramètre de période d'échantillonnage afin d'obtenir un déplacement utilisant toute la course disponible) :



Vous avez à disposition dans le code élève créant les listes suivantes :

- ld et li ($\omega < 0$): ld liste de listes des couples $[n, \omega]$ des différentes mesures et li des intensités associées
- LD et LI ($\omega > 0$): LD liste de listes des couples $[n, \omega]$ des différentes mesures et LI des intensités associées
- Ln : Liste des 7 valeurs de n possibles $[0, 1, 2, 3, 4, 5, 6]$



Le code élèves affiche ensuite les données en 3D (intensité i et I en mA en fonction de n et ω).

Méthode RLM

On effectue l'import suivant ([documentation](#)) :

```
from sklearn.model_selection import train_test_split
```

On écrit alors :

```
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0)
```

Cette instruction sépare les données x (données des images de la database) et y (caractères espérés pour chaque image issue des données) en deux familles, « train » et « test ». Sans aucune précision, 75% des données servent pour « train » et 25% pour « test ». L'option « random_state » à définir avec un entier positif permet d'obtenir un découpage aléatoire constant des données quelle que soit l'exécution du code (afin d'obtenir les mêmes prévisions). S'il n'est pas précisé, chaque exécution de l'apprentissage utilisera une répartition des données différente.

On importe alors le modèle rlm, on l'entraîne sur les données « train » (fit) et on affiche son score sur le set « test » ainsi :

```
from sklearn.linear_model import LinearRegression
rlm = LinearRegression()
rlm.fit(x_train,y_train)
Score = rlm.score(x_test,y_test)
print("Score:",Score)
```

Le score affiché est le coefficient de corrélation de la régression réalisée.

On peut alors estimer la valeur de prédiction du modèle à l'aide des lignes suivantes :

```
L = [n,w]
rlm.predict(L) [0]
```

Il est possible d'enregistrer ou de charger le modèle ainsi :

```
from joblib import dump, load
dump(rlm, 'Modèle_RLM.joblib')
rlm = load('Modèle_RLM.joblib')
```

Programmation

Dans toute la suite, on utilisera des lettres minuscules pour les données « d » à $\omega < 0$ et des lettres majuscules pour les données « D » à $\omega > 0$.

Question 1: Mettre en place les lignes de code créant les données « train » et « test » sur les données d et « TRAIN » et « TEST » sur les données D

Question 2: Mettre en place les lignes de code réalisant l'entraînement en deux modèles rlm (données d) et RLM (données D) sur chacun des sets de données et affichant les scores obtenus

Vérifier :

```
score: 0.9893080589802739
SCORE: 0.9879136192625299
```

On souhaite réaliser la prédiction de la valeur B d'un pixel connaissant sa liste [R,G]. Cette prédiction sera :

- Arrondie à l'entier le plus proche
- Transformée en entier
- Relimitée à l'intervalle [0,255] si nécessaire

Question 3: Proposer les fonctions Prediction_rlm(L) et Prediction_RLM(L) réalisant la prédiction d'intensité moteur en A associée au couple L=[n,w]

Vérifier :

```
>>> Prediction_rlm([0, -1000])
0.9353162165220064
```

```
>>> Prediction_RLM([0, 1000])
3.02387797292206
```

Identification des coefficients inconnus

A partir des deux modèles entraînés RLM et rlm, on souhaite mettre en place des démarches permettant de trouver m_0 , m_i , C_{f_s} et f .

Dans un premier temps, nous souhaitons créer un graphique 3D des couples moteurs issus des modèles pour toutes les valeurs de n dans Ln et toutes les valeurs de w dans lw (modèle rlm) et $L\omega$ (modèle RLM).

Question 4: Mettre en place le code permettant d'afficher le graphique de C_m en fonction de n et ω

On souhaite maintenant calculer le coefficient f à tous ces points.

Question 5: Mettre en place le code permettant d'afficher le graphique de f en fonction de n et ω

Question 6: En déduire la valeur moyenne de f à tous ces points

Question 7: Mettre en place le code permettant de tracer la masse m en fonction de n

Question 8: En utilisant une régression linéaire (`np.polyfit`), en déduire les valeurs de m_0 et m_i

Question 9: Mettre en place le code permettant de tracer la masse C_{f_s} en fonction de n

Question 10: En déduire la valeur moyenne de C_{f_s}