Correction d'un système asservi par PID et Réseau de neurones

Mise en situation

Un bras manipulateur est le bras d'un robot généralement programmable, avec des fonctions similaires à un bras humain. Les différentes parties du robot reliées entre elles permettent, soit des mouvements de rotations et/ou de translations linéaires.
Dans le cas d'une imitation complète d'un bras humain, un bras manipulateur a donc 3 mouvements de rotation et 3 mouvements de translation sur son élément terminal. Il peut être autonome ou contrôlé manuellement et peut être utilisé pour effectuer une variété de tâches avec une grande précision.

Le cahier des charges fixe les paramètres suivants pour une entrée échelon :

- **Une rapidité avec un temps de réponse à 5% de 1 s.**
- **Pas de dépassement.**
- **Une erreur statique nulle.**

Objectif

L'objectif du TD est de comparer la correction du système par solution PID et par un correcteur de type réseau de neurones.

Modélisation par schéma-blocs

La modélisation de la machine à courant continu reprend les équations classiques avec prise en compte de l'inductance et d'un frottement visqueux :

L'équation électrique est issue de la loi des mailles : $$\boldsymbol{u(t) = e(t) + R.i(t) + L.\frac{di(t)}{dt}}$$

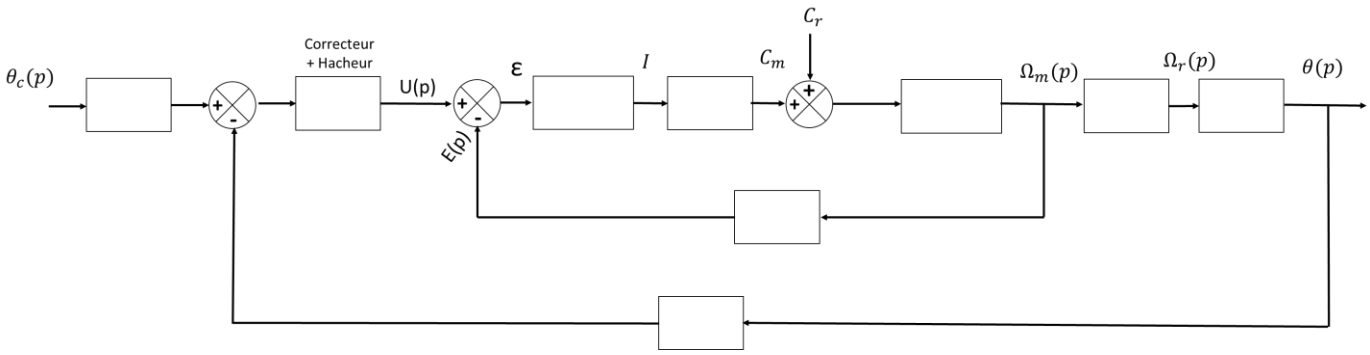
L'équation mécanique est issue d'un théorème de l'énergie cinétique à l'ensemble du bras du robot :

$$\boldsymbol{J_{eq}.\frac{d\omega_m(t)}{dt} = C_m(t) + C_r(t) - f.\omega_m(t)}$$

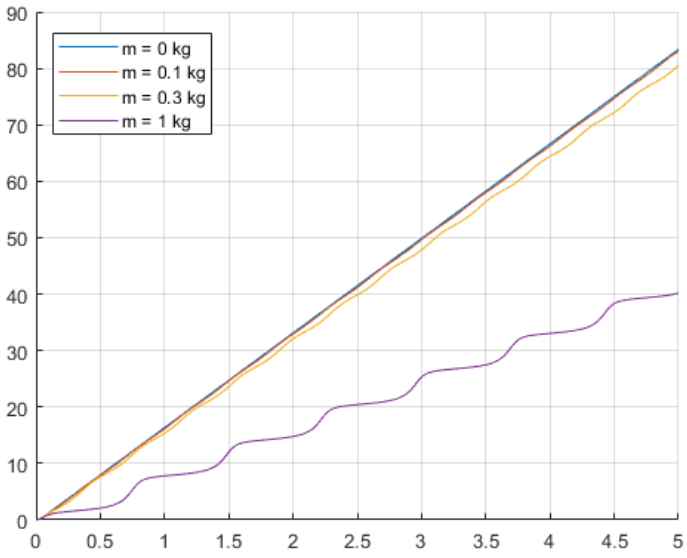
Equations caractéristiques du moteur à courant continu : $\boldsymbol{C_m(t) = K_t.i(t)}$ et $\boldsymbol{e(t) = K_e.\omega_m(t)}$

La transmission est classique par introduction du train d'engrenages et de son rapport de réduction *r* et l'intégration de la vitesse en position. Un correcteur et un hacheur sont placés devant le moteur pour le commander, leur fonction de transfert sera noté C(p). Un capteur de position de gain G permet de mesurer la position du bras à chaque instant et de la comparer à la position de consigne. Les conditions initiales sont supposées nulles.

Q1 : Compléter le schéma-bloc complet du système avec perturbation.



Le couple résistant ramené sur l'arbre moteur vaut $\boldsymbol{C_r(t) = -m.g.d.r.sin(\theta)}$.
La perturbation due au couple résistant engendre des oscillations mais en plus créé un phénomène non-linéaire assimilable à une dissymétrie du comportement lors de la descente de la charge par rapport à sa phase de montée.

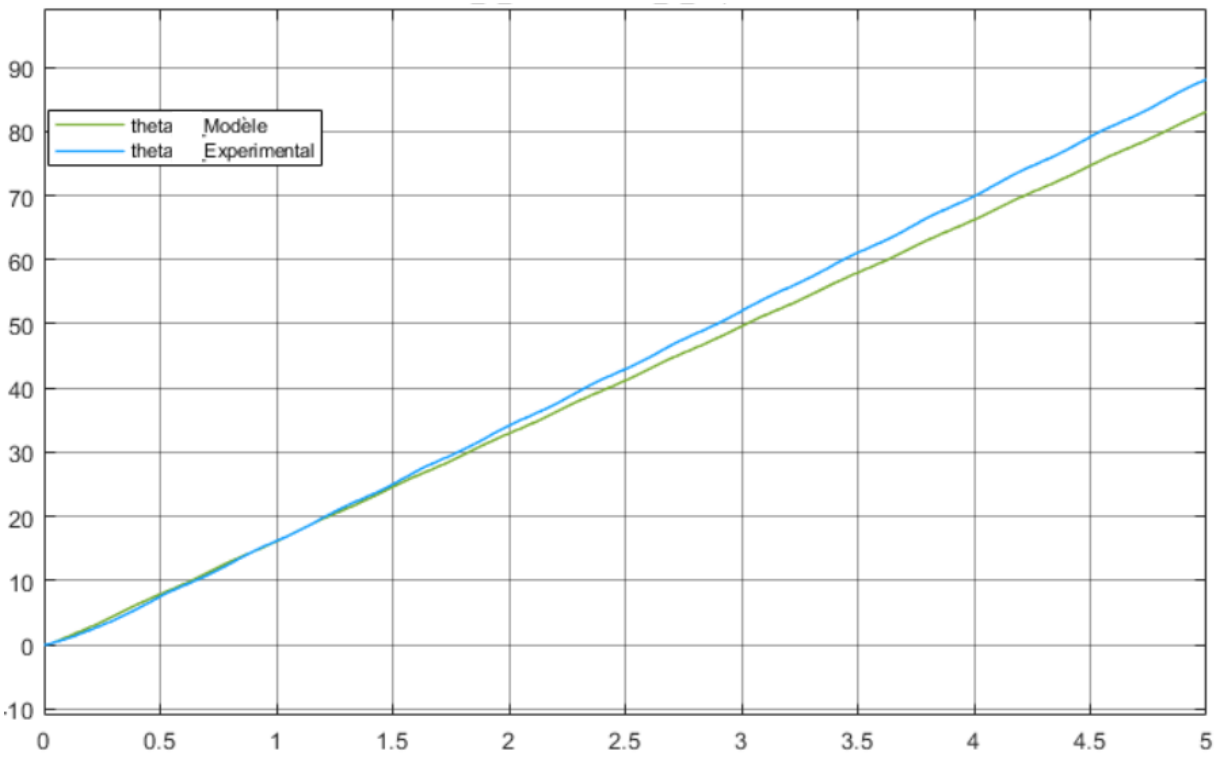


Influence de la masse (perturbation du système)

Q2 : Proposer l'ensemble des expériences à mettre en place afin de déterminer les constantes R, K_t et J.

Afin d'améliorer les valeurs numériques des paramètres trop éloignées de leur valeurs pratiques déterminées expérimentalement, nous cherchons à superposer une réponse expérimentale à une réponse simulée.

Q3 : On réalise un essai en BO pour une entrée échelon d'une tension de 9 V pendant 5 s et on superpose cette réponse à la réponse simulée. En déduire les deux principaux paramètres mal évalués.



Afin de réduire les écarts entre le modèle et le réel, un algorithme de type $f(x) = 0$ va être utilisé sur les deux paramètres identifiés précédemment. L'algorithme proposé est celui de *Newton* introduit sous la forme :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

qui se généralise pour un système d'équations à plusieurs variables :

$$\underline{x_{n+1}} = \underline{x_n} - \boldsymbol{J}^{-1}\left(\underline{x_n}\right).\underline{f}(\underline{x_n})$$

où $\boldsymbol{J}$ représente la matrice jacobienne définie par :

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_p} \end{bmatrix}$$

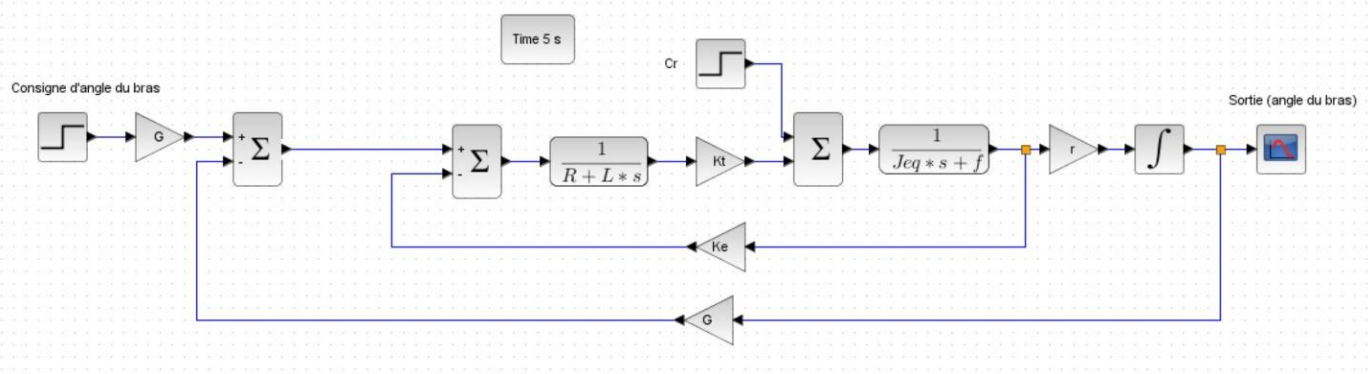
Q4 : A la lecture du code proposé en annexe, retrouver les étapes suivantes :

A. Construction de la matrice Jacobienne
B. Première itération
C. Boucle
D. Condition de fin d'itération

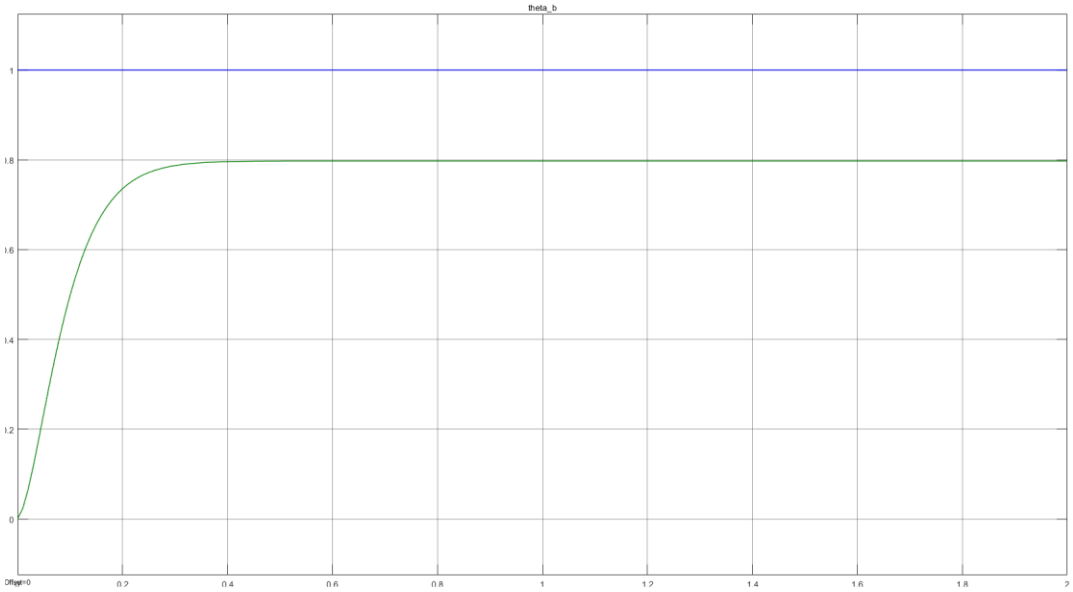
L'algorithme de *Newton* permet d'obtenir deux nouvelles valeurs de K_t et J_{eq} qui permettent à la simulation de coller à l'expérimentation. Le modèle proposé semble assez robuste afin de prévoir des réponses à une entrée en échelon.

Correction par PID

On réalise le schéma-bloc du bras de robot asservi en position sans correction sur le logiciel Scilab.



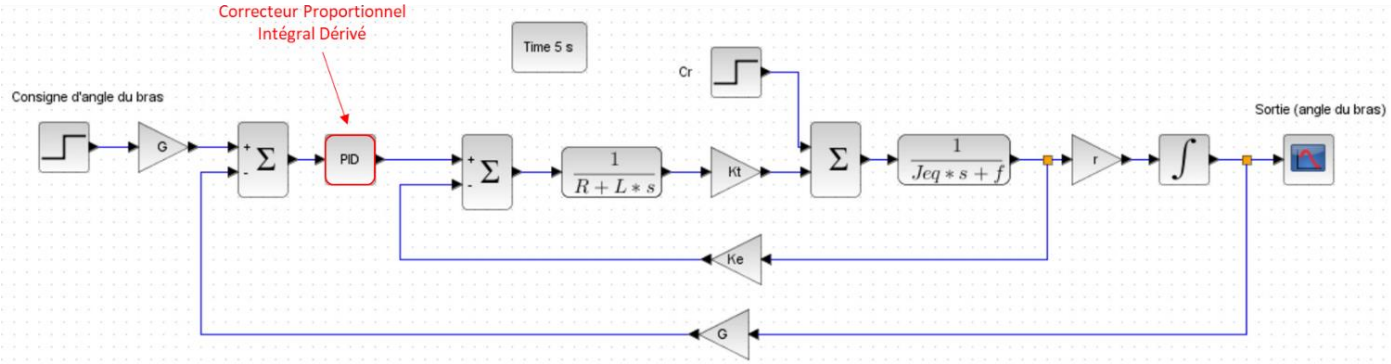
On lance la simulation pour visualiser la réponse du système non corrigé pour une entrée échelon unitaire.



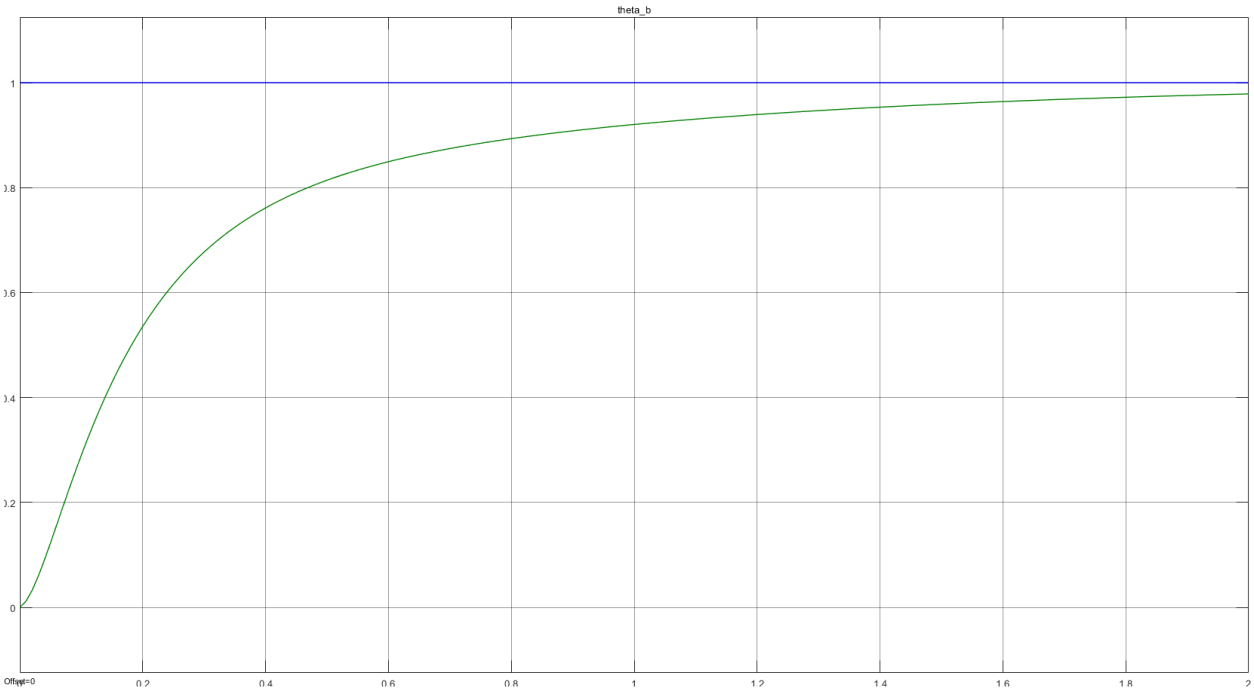
Réponse du système bouclé sans correction

Q5 : Quelles sont les performances qui ne sont pas validées ? En déduire un type de correction nécessaire.

Pour améliorer l'ensemble des performances, un correcteur PID est utilisé par la suite et implémenté dans la simulation sous Scilab. On utilise par la suite un module spécifique de Scilab permettant d'ajuster automatiquement au mieux les paramètres du PID pour un cahier des charges donné.



On relance la simulation pour visualiser la réponse du système nouvellement corrigé pour une entrée échelon unitaire. On obtient le résultat suivant :



Réponse du système bouclé avec correction optimale

Q6 : Les exigences du cahier des charges sont-elles satisfaites ? Expliquer quelle peut être la source de ces écarts entre la prévision du module de réglage automatique du PID et la simulation.

Correction par réseau de neurones

Dans cette partie, le système sera corrigé à l'aide d'un PID dont les coefficients seront calculés et optimisés par intelligence artificielle via un réseau de neurones.

Il s'agit dans un premier temps d'entrainer le réseau de neurones. On retrouve les étapes classiques d'un entrainement :

- Choix des données.
- Structure du réseau.
- Entrainement.
- Analyse des résultats.
- Itération.
- Fermeture du modèle.

On rappelle ici que la modélisation par un réseau de neurones est un processus itératif.

Pré-entrainement

Q7 : Proposer un modèle idéal à identifier (parmi ceux vus en cours d'automatique - 1er ou 2nd ordre) afin de satisfaire les exigences du cahier des charges. On pourra se demander quel est le comportement souhaité, en régime permanent et en régime transitoire, quel ordre permet de vérifier ceci, etc…

Le point crucial d'une bonne modélisation par réseau de neurones réside dans les données.

Proposition de données

Tout d'abord, listons les différentes formes de signaux que l'on peut exploiter : échelon, rampe, sinus, exponentiel, etc. On sait en avance qu'il faudra que la commande soit une forme exponentielle (pour une entrée en échelon ou en rampe). Par conséquent, une entrée de la forme $K\left(1-e^{-\frac{t}{\tau}}\right)$ serait idéale. Cela élimine donc un entrainement avec des signaux en sinus. Il est par contre assez facile d'approximer ces fonctions exponentielles par une « somme » d'échelons (ou de rampe).

Une façon donc assez pratique est d'entrainer notre réseau de neurones par des fonctions échelon ayant des durées variables entre une valeur faible par rapport au temps de réponse caractéristique et une valeur de l'ordre du temps caractéristique. Par conséquent, le réseau, pourra traduire le régime transitoire ainsi que le régime permanent. De, plus il faut balayer l'ensemble des tensions de commande du moteur, soit de 0 V à 9 V. Ces deux paramètres doivent disposer d'une répartition aléatoire mais suivant une distribution uniforme pour avoir autant de chance d'apparaître. Pour le nombre d'échantillons à générer, plus il y a de données, meilleur sera le réseau de neurones. Par conséquent, il ne faut pas hésiter à exploiter les capacités du système.

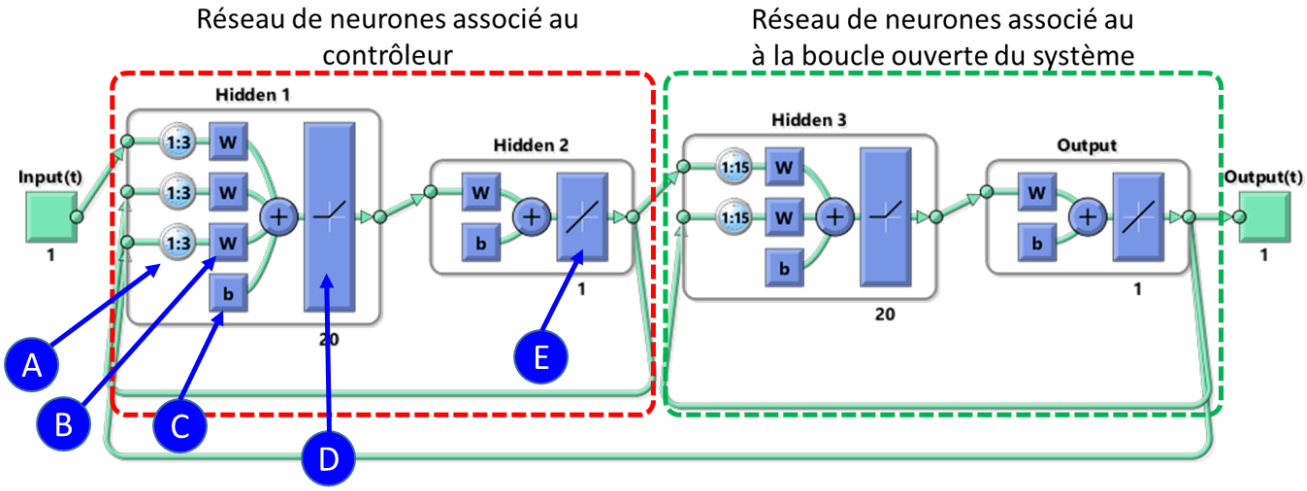
Le système sera donc sollicité essentiellement en réponse à des échelons de positions. On se fixe une période d'échantillonnage liée aux caractéristiques de la carte de traitement soit *Te* = 0,01 s. Les durées de ces échelons doivent être compris entre 0,5 s (choix arbitraire) et 5 s (choix

permettant de mettre en évidence le régime permanent). Les amplitudes de sortie correspondent aux possibilités du système. Ici, nous souhaitons entraîner le réseau de neurones sur une amplitude de rotation de $[-2\pi\ ;\ 2\pi\ ]$ rad. Le nombre de 20 000 points permet d'obtenir globalement une petite centaine d'échelons, ce qui semble être un nombre correct vis-à-vis de l'entraînement qui suit.

Structure du réseau de neurones

La théorie sur les réseaux de neurones précise qu'on peut approximer tous les systèmes par un réseau comportant 2 couches. Pour le nombre de neurones, il faut se fixer une valeur pour débuter. Il n'existe pas de règles pour prédire cette valeur. Nous allons dans un premier temps essayer avec 20 neurones. La fonction d'activation entre la couche intermédiaire et la couche de sortie sera une fonction ReLU, puis une fonction linéaire en sortie du réseau.

La structure simplifiée, proposée pour le réseau de neurones, est donnée ci-dessous :



Q8 : Identifier et nommer les éléments A, B, C, D et E.

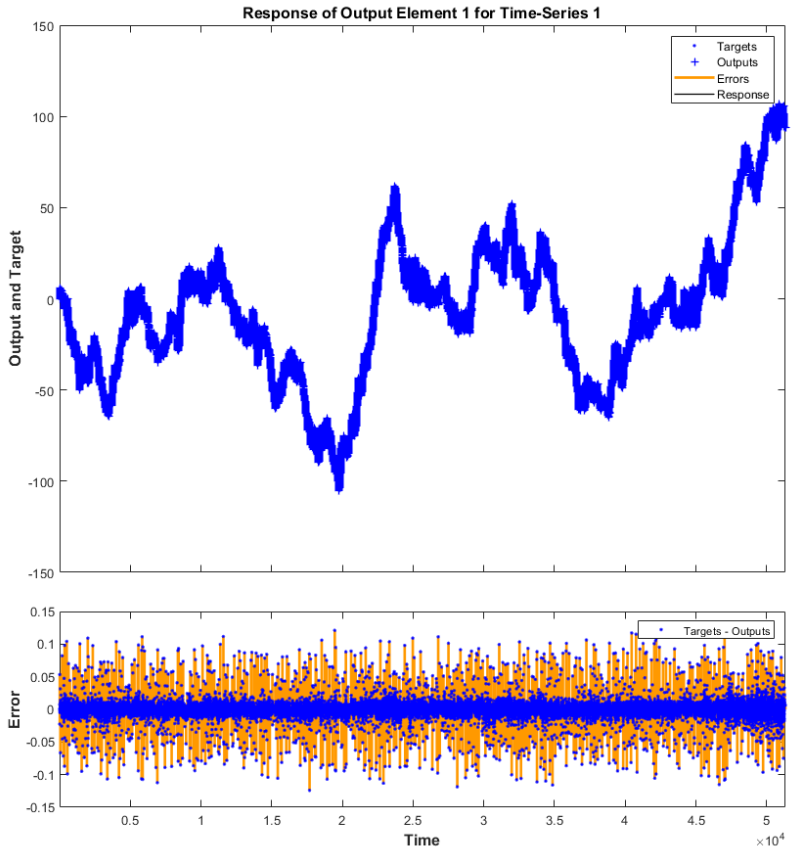
Entrainement

On entrainera le modèle avec deux types d'entrainement :

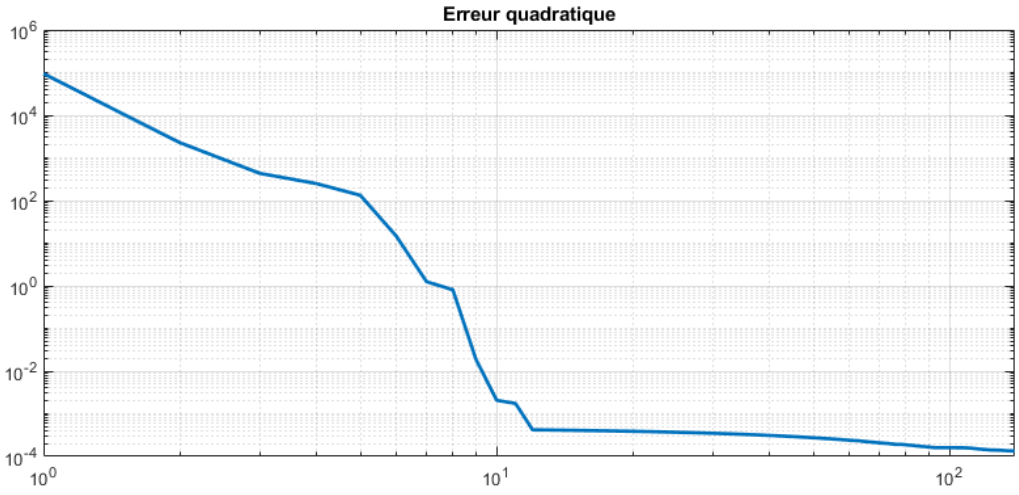
- Régularisation de Bayes
- Levenberg-Marquardt

Dans un premier temps, la régularisation de Bayes est utilisé afin d'exploiter son résultat intéressant de proposition du nombre de paramètres utilisés par rapport au nombre de paramètres disponibles.

La mise en place de cet entrainement montre, à travers les deux figures ci-après, que le modèle par réseau de neurones colle très bien aux données.



Comparaison modèle/données

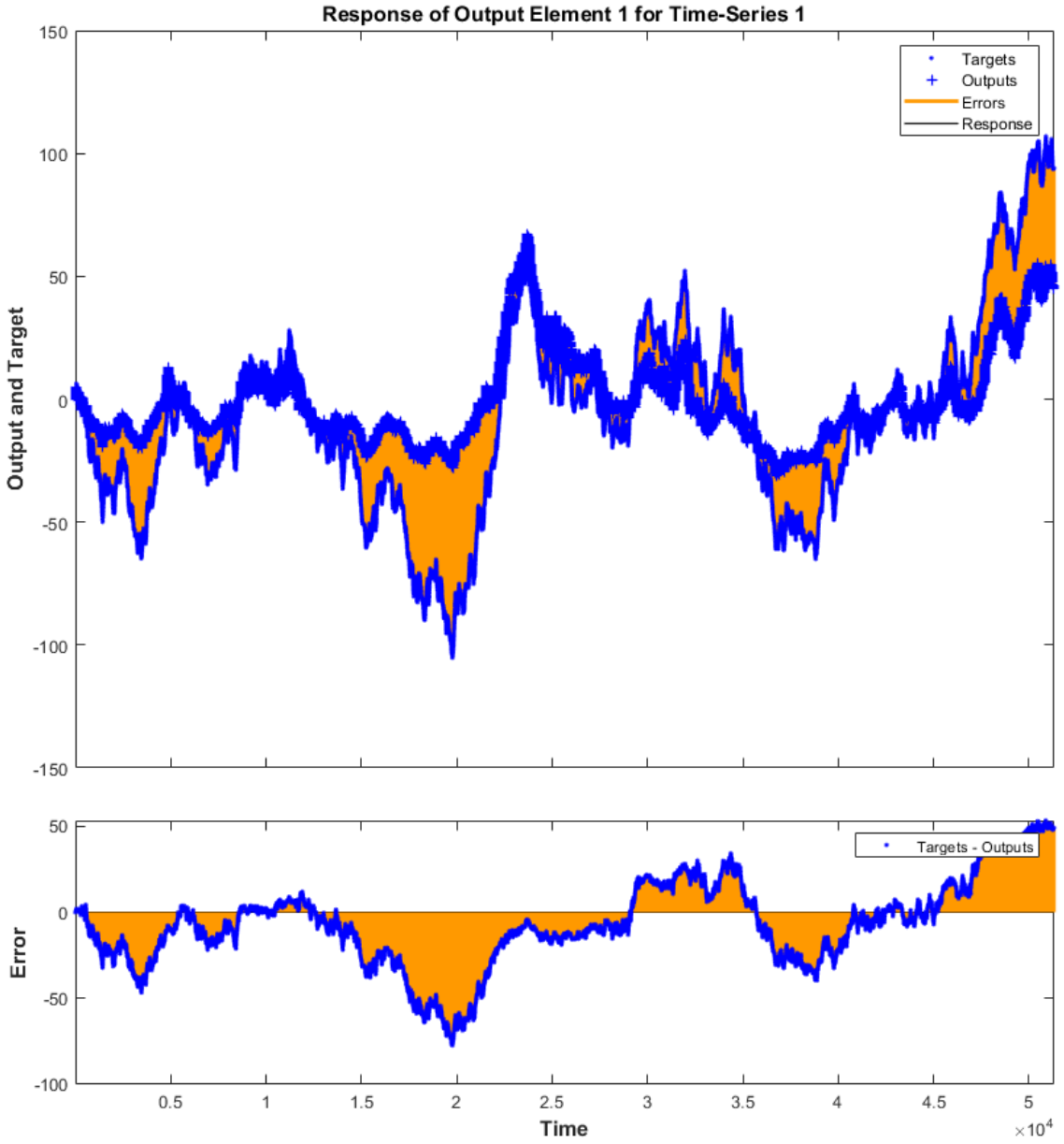


Evolution de la qualité du modèle

Q9 : Que faut-il faire pour ré-itérer le processus d'entrainement afin de valider la modélisation ?

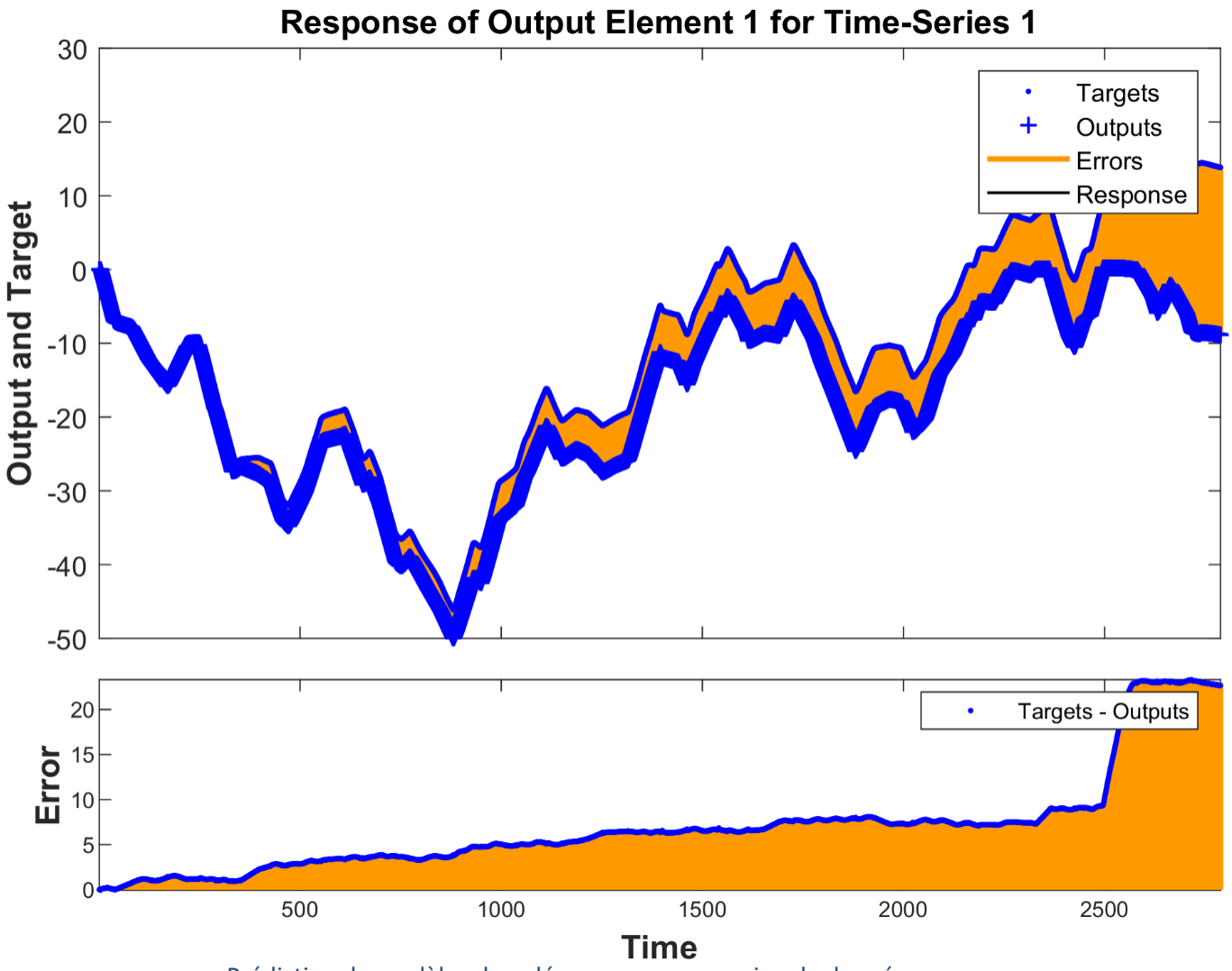
Post-entrainement

Le fait de construire le modèle sur les données renvoyés par le modèle lui-même montre que celui-ci perd en qualité. À cette étape il est alors possible de ré-entrainer le modèle sur le modèle réellement bouclé afin d'améliorer ses qualités de prédiction pour un coût de calcul raisonnable.



Prédiction du modèle « bouclé » par rapport aux données

On teste ensuite le réseau de neurones sur une nouveau jeu de données. On obtient les résultats suivants :

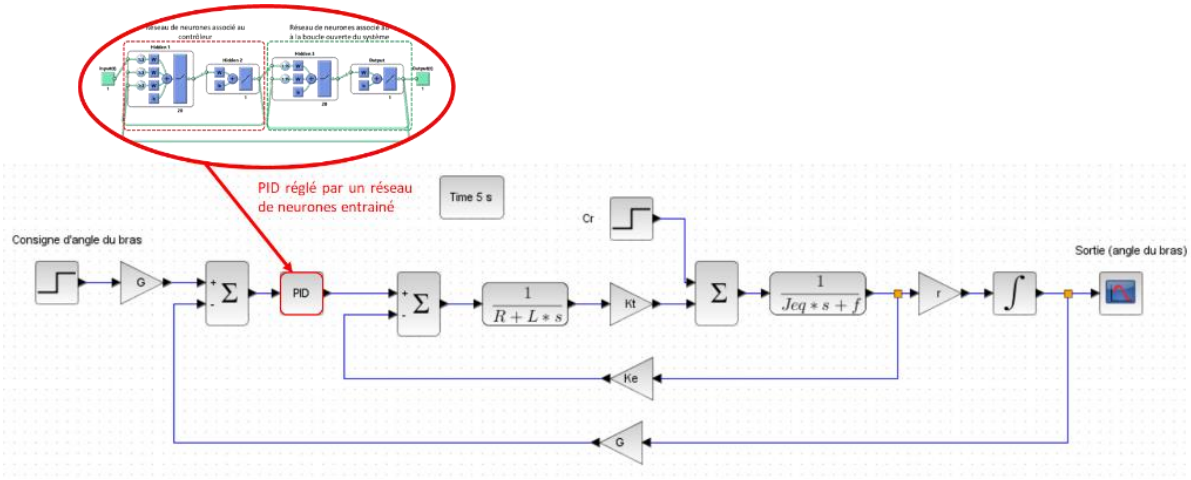


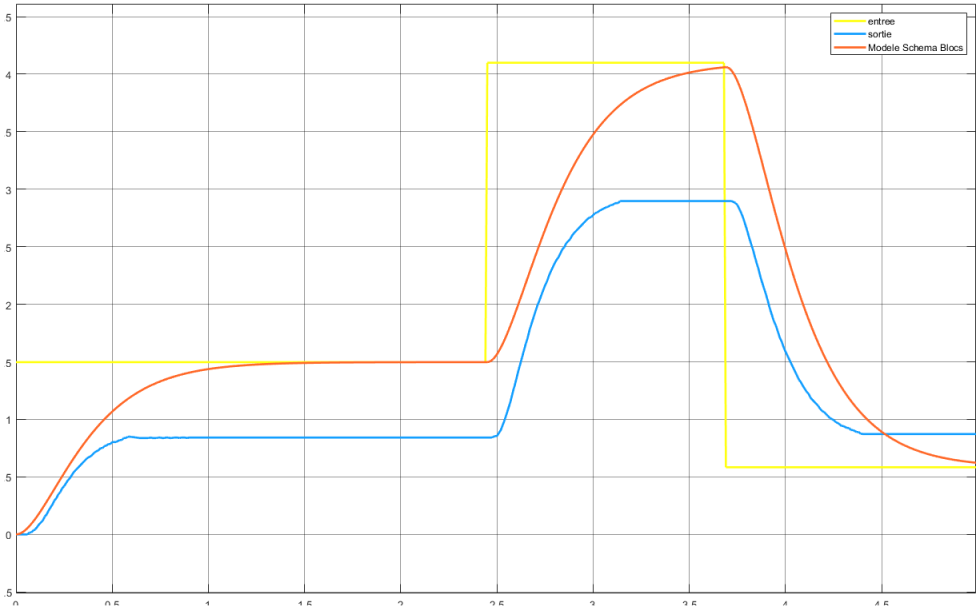
Prédiction du modèle « bouclé » sur un nouveau jeu de données

Q10 : Analyser les résultats obtenus.

Implémentation du réseau de neurones dans le schéma-blocs

Les coefficients des gains du correcteur PID sont maintenant calculés et optimisés par intelligence artificielle via un réseau de neurones entrainé.





Résultat donné par le système contrôlé par un réseau de neurones

Q11 : A partir de la courbe précédente, évaluer les écarts par rapport au cahier des charges. Expliquer la source des éventuelles différences.

Q12 : Proposer une synthèse sur les avantages et inconvénients sur l'utilisation d'un réseau de neurones pour déterminer les paramètres optimaux d'un correcteur PID.

ANNEXE – Algorithme de la méthode de Newton avec construction de la matrice Jacobienne

```
# Détermination de la sous-matrice jacobienne par rapport à Kt
coeff = 1e-6 # Pas de la différence finie
Kt = 0.0111  # Valeur initiale de Kt
Jeq = 3e-6   # Valeur initiale de Jeq

# Simulation 1
KtPlus = Kt*(1+coeff) # Valeur supérieure
Kt = KtPlus
a = sim(fichier_sim,'SimulationMode','normal')
b = a.get('simout')
sPlus = b.Data

# Simulation 2
KtMoins = Kt*(1-coeff) # Valeur inférieure
Kt = KtMoins
a = sim(fichier_sim,'SimulationMode','normal')
b = a.get('simout')
sMoins = b.Data

JKt = (-sPlus + sMoins) / (2*Kt*coeff)

# Détermination de la sous-matrice jacobienne par rapport à Jeq
Kt = 0.0111 # Valeur initiale de Kt
Jeq = 3e-6  # Valeur initiale de Jeq

JeqPlus = Jeq*(1+coeff)
JeqMoins = Jeq*(1-coeff)

# Simulation 1
Jeq = JeqPlus
a = sim(fichier_sim,'SimulationMode','normal')
b = a.get('simout')
sPlus = b.Data

# Simulation 2
Jeq = JeqMoins
a = sim(fichier_sim,'SimulationMode','normal')
b = a.get('simout')
sMoins = b.Data

JJeq = ( - sPlus + sMoins) / (2*Jeq*coeff)

#Concaténation de la matrice jacobienne
J = [JKt , JJeq]
Kt = Kt
Jeq = Jeq


X = [Kt,Jeq]
Xnouveau = X - J\F


compteur = 0
while (abs(norm(Xnouveau-X))>1e-5 && compteur < 10) :

    X = Xnouveau

    # pour Kt

    Kt = X(1)
    Jeq = X(2)
    #f = X(3)

    KtPlus = Kt*(1+coeff)
    KtMoins = Kt*(1-coeff)

    Kt = KtPlus
    a = sim(fichier_sim,'SimulationMode','normal')
    b = a.get('simout')
    sPlus = b.Data

    Kt = KtMoins
    a = sim(fichier_sim,'SimulationMode','normal')
    b = a.get('simout')
    sMoins = b.Data

    JKt = ( - sPlus + sMoins) / (2*Kt*coeff)

    # pour Jeq

    Kt = X(1)
    Jeq = X(2)
    # f = X(3)

    JeqPlus = Jeq*(1+coeff)
    JeqMoins = Jeq*(1-coeff)

    Jeq = JeqPlus
    a = sim(fichier_sim,'SimulationMode','normal')
    b = a.get('simout')
    sPlus = b.Data

    Jeq = JeqMoins
    a = sim(fichier_sim,'SimulationMode','normal')
    b = a.get('simout')
    sMoins = b.Data

    JJeq = ( - sPlus + sMoins) / (2*Jeq*coeff)

    # Concaténation de la matrice jacobienne
    J = [JKt , JJeq]

    Kt = X(1)
    Jeq = X(2)
    a = sim(fichier_sim,'SimulationMode','normal')
    b = a.get('simout')
    s = b.Data
    F = exp.signals.values - s

    compteur = compteur + 1

    X = [Kt,Jeq]

    Xnouveau = X - J\F
```