École Centrale de Nantes

SIMUS

September 19, 2025

But. L'objectif de cette séance est de se familiariser avec la boîte à outils de Matlab, Simulink, qui est dédiée à la simulation de systèmes dynamiques. A travers quelques développements simples, l'idée est de permettre d'acquérir un minimum de compétences et de réflexes, afin de simuler une large gamme de systèmes et de pouvoir exploiter efficacement les résultats obtenus par la simulation.

On considère le système linéaire suivant

$$\begin{array}{rcl} \dot{x}_1 & = & x_2 \\ \dot{x}_2 & = & K\omega_n^2 u - \omega_n^2 x_1 - 2\zeta\omega_n x_2 \end{array} \tag{1}$$

avec K, ω_n et ζ les paramètres du système.

1. Ecrire ce système sous forme de fonction de transfert

$$\frac{Y(s)}{U(s)} = G(s) \tag{2}$$

avec la sortie $y(t) = x_1$.

2. Cliquer sur *Simulink*, puis sur *Blank Model*. Une fenêtre d'édition s'ouvre. Cliquer sur *Library Browser*. Dans le but de simuler le comportement dynamique de ce système, on se propose d'utiliser 3 voies offertes par Simulink

- les blocs "fonction de transfert";
- les blocs "Fcn - f(u)";
- les blocs "Matlab function".

Les paramètres du système seront initialisés grâce à un script matlab `init.m`, et sont fixés à

$$K = 1, \ \ \zeta = 0.7, \ \ \omega_n = 10 \text{ rad}.s^{-1} \tag{3}$$

L'ensemble des fonctions Simulink sont disponibles dans `Simulink Library Browser`. Débuter le développement du simulateur en utilisant un bloc "fonction de transfert" (dossier `Continuous`): lors de la définition du système, penser à

utiliser les noms des variables définies dans `init.m` (et non les valeurs numériques).

Le simulateur doit permettre le choix entre plusieurs types d'entrées (dossier `Sources`):

- échelon (`Step`): échelon unitaire, valeur intiale nulle;
- rampe (`Ramp`): pente unitaire, instant initial nul, valeur initiale nulle;
- sinusoïde (`Signal Generator`): amplitude égale à 1, fréquence égale à 10 Hz.

Pour assurer le choix entre les différents types d'entrée, on peut utiliser un commutateur manuel à 3 entrées (`Multiport Switch` dans le dossier `Signal Routing`). Avant de lancer la simulation, il faut choisir l'algorithme d'intégration (onglet `Simulation`, sous-menu `Model Configuration Parameters`), le type (pas variable ou fixe), éventuellement la valeur du pas d'intégration.

Dans un premier temps, on utilisera l'algorithme d'intégration `Euler`, avec un pas d'échantillonnage T_e qui sera défini dans le fichier `init.m`.

Fixer la durée de la simulation à 10 s. Dans le but d'utiliser les informations dans l'environnement Matlab, on souhaite sauvegarder l'entrée et la sortie du système dans des vecteurs respectivement notés `U` et `Y`. Pour cela, utiliser les blocs `To Workspace` (dossier `Sinks`). Penser à adapter les options dans `Configuration Parameters`, sous-menu `Data Import/Export`.

Faire des essais de simulation **en boucle ouverte** avec une période d'échantillonnage $T_e = [0.001\ s\ ;\ 0.01\ s\ ;\ 0.1\ s]$. Après chaque simulation, afficher la sortie grâce à la commande `plot` de Matlab. Afficher les 3 résultats de la simulation sur la même figure, en utilisant une couleur différente pour chaque courbe (taper `help plot` pour voir comment paramétriser la couleur). Commenter les résultats obtenus. La valeur finale de la sortie était-elle prévisible ?

Utiliser maintenant l'algorithme d'intégration `Runge-Kutta` avec $T_e = 0.1\ s$, puis $T_e = 0.001\ s$. Commentaires.

3. On utilise désormais l'algorithme d'intégration `Runge-Kutta`. Appliquer les entrées en rampe et sinusoïdale, avec $T_e = [0.001\ s\ ;\ 0.01\ s\ ;\ 0.1\ s]$. Commenter. On souhaite maintenant appliquer un correcteur proportionnel (y_c étant la consigne qui peut être un échelon, une rampe, ou une sinusoïde)

$$u(t) \ = \ K_p \cdot (y_c - y)$$

à ce système et étudier son comportement en boucle fermée. Réaliser le simulateur de ce système en boucle fermée, et fixer K_p à 10 (via le fichier `init` - le gain K_p sera intégré au simulateur via la bloc `Gain` disponible dans le dossier `Math Operations`). La période d'échantillonnage sera fixée à 0.01 s. La consigne sera sauvegardée dans une variable `Yc`. Tracer les réponses à un échelon, à une rampe, et à un signal sinusoïde (en mettant systématiquement la consigne). Pour les réponses à un échelon et à une rampe, tracer également les erreurs en régime permanent, et justifier leurs valeurs par le calcul.

4. On se propose maintenant de simuler le comportement du système en utilisant soit un bloc `f(u)`, soit un bloc `MATLAB function`. Dans ces deux cas, il faut utiliser le système sous sa forme d'état (1). Le simulateur devra enregistrer l'état sous la forme de deux variables distinctes `X1` et `X2` d'une part, et `X1a` et `X2a` d'autre part. Les conditions initiales de ces deux variables seront fixées par `X10` et `X20` définis dans le fichier `init`. Ces schémas peuvent nécessiter l'utilisation de multiplexeurs (bloc `Mux` dans dossier `Signal Routing`), de démultiplexeurs (bloc `Demux` dans dossier `Signal Routing`) et d'intégrateurs (bloc `Integrator` dans dossier `Continuous`). Simuler dans le même fichier les 3 schémas (fonction de transfert, bloc `MATLAB function`), bloc `f(u)`. Vérifier que les 3 méthodes fournissent les mêmes résultats (sous réserve d'avoir les mêmes conditions initiales!) pour une entrée en échelon. Pour celà, tracer sur la même figure, en haut `Y`, `X1` et `X1a`, et la commande `U` en bas: utiliser la commande `subplot`.

5. On choisit maintenant de travailler uniquement avec la `MATLAB function`. Faire des simulations pour les 3 types d'entrées (échelon, rampe, signal sinusoïdal). Pour cela, on modifiera le fichier `init.m` qui, en plus d'initialiser les paramètres, devra

- lancer l'exécution du fichier Simulink (`.mdl`) grâce à la commande `sim`;
- sauvegarder les variables présentes dans le `Workspace` dans un fichier noté `Essai_`*type_d_entree*`.mat` une fois la simulation terminée grâce à la commande `save`.

Une fois l'ensemble des simulations faites, faire un script `Matlab` pour tracer l'ensemble des résultats: pour cela, il faudra charger les fichiers `.mat` préalablement créés (commande `load`) de manière à tracer les variables dignes d'intérêts (ne pas oublier la commande !). On pourra par exemple tracer sur chaque figure en haut, `Yc` et `X1`, et en bas `U` (penser à mettre un titre (`title`) et des légendes pour l'axe des abscisses (`xlabel`) et des ordonnées (`ylabel`).

6. Aujourd'hui, les commandes pilotant les systèmes physiques (qui sont la plupart du temps, des systèmes continus) sont générées par des calculateurs, donc par essence même, de nature échantillonnée. Il convient donc *a minima* de vérifier, par simulation, le comportement du système bouclé face à cet échantillonnage. Pour cela, le système va être simulé à une période déchantillonnage $T_e = 10^{-5}$ s, et on ajoute un bloqueur (`Zero-order hold` dans `Discrete`) juste entre la commande et l'entrée du système. Expliquer le rôle du bloqueur, et la raison de son emploi (en quoi cela est intéressant pour évaluer rapidement l'effet de l'échantillonnage de la commande). Procéder différents blocages ($10T_e$, $100T_e$, ...). Tracer sur la même figure la variable `X1` pour ces différentes valeurs, et commenter.

7. Supprimer le bloqueur. En remplacement de la correction proportionnelle, on se propose maintenant de concevoir une loi de commande par retour d'état en supposant tout d'abord que x_1 et x_2 sont mesurés. On a donc une commande du type

$$u(x) \;=\; -k_2 x_2 - k_1 x_1 .$$

Calculer les gains k_1 et k_2 de façon à placer les pôles du système en boucle fermée à -10 et -10. Pour cela, utiliser la fonction `acker`. Tracer sur une même figure `X1`, `X2` et la commande `U`.

8. Dans la plupart des cas, seul x_1 est mesuré. Il faut donc calculer numériquement sa dérivée de façon à estimer x_2. Dans ce cas, on utilise le bloc $\Delta u/\Delta t$ (dans `Continuous`) dans lequel se trouve la fonction de transfert

$$\frac{s}{1+\tau s}$$

Expliquer pourquoi ce filtre peut jouer le rôle de dérivateur et quelles en sont ses limites. Tester ce dérivateur en fonction du réglage de τ.