Apports des outils numériques sur l'enseignement de l'automatique : ateliers à partir de notebook Jupyter – Atelier 1

Culture Sciences de l'Ingénieur

Javier OJEDA

Édité le 06/07/2020

école normale supérieure paris—saclay

Atelier 1 : Identification d'un système d'ordre supérieur à 2

Cette ressource est l'un des trois ateliers illustrant la ressource « *Apports des outils numériques sur l'enseignement de l'automatique : ateliers à partir de notebook Jupyter* ».

Résumé : L'évolution des outils numériques aussi bien logiciels que du matériel permet d'envisager l'enseignement de l'automatique de manière plus illustrative et participative. Illustrative, car des outils numériques, le plus souvent libres, sont disponibles pour les étudiants afin de mettre en image des notions parfois complexes de l'automatique et participative, car au travers d'ateliers réalisés par les étudiants, ceux-ci peuvent appréhender des notions principales ou annexes par leurs simulations et expérimentations. Par ailleurs, participative, par le fait que les outils numériques permettent des échanges enseignant-apprenant en ligne via un site web, des plateformes moodle, slack, etc. Dans cette ressource, nous allons nous intéresser aux outils numériques logiciels par le biais du langage Python et du notebook Jupyter. Pour cela nous allons décrire trois ateliers enseignant-apprenant réalisables en ligne et qui abordent trois notions de l'automatique : l'identification, le dimensionnement automatique d'un correcteur de type PID et le calcul des pôles en boucle fermée.

Ressource publiée sur Culture Sciences de l'Ingénieur : https://eduscol.education.fr/sti/si-ens-paris-saclay

Atelier_1

June 12, 2020

1 Atelier n°1 : Identification d'un système d'ordre supérieur à 2

Auteur : Javier OJEDA
Date : 25/05/2020
Version : 0.1

1.1 Importation des packages nécessaires

[1]:
```
import control as ctl # Package control pour l'automatique
```

Lien pour le package control

[2]:
```
#Affichage avec la bibliothèque graphique intégrée à Notebook
%matplotlib inline
import matplotlib.pyplot as plt # Package pour le représentation graphique
plt.rcParams.update({'font.size': 18}) # Changer la taille de police par défaut
plt.rcParams.update({'lines.linewidth': 3}) # Changer la taille des lignes

import numpy as np # Package pour le calcul scientifique

import warnings
warnings.filterwarnings('ignore') # Pour retirer les warnings intempestifs
```

1.2 Méthodologie

Objectif : Cet atelier doit permettre de mettre en place l'indentification d'un système apériodique par la méthode de Strejc.

Moyen : Nous allons mesurer la réponse à un echelon unitaire et reléver les temps caractéristiques nécessaires au calcul des coefficients intervenants dans la méthode de Strejc.

Méthode de Strejc :

Cette méthode s'applique aux systèmes sans dépassement que l'on cherche à approximer par une fonction de transfert de la forme :

$H_{app}(p) = \frac{K_0}{(1+T_d p)^n}$

n	$\frac{T_1}{T_d}$	$\frac{T_2}{T_d}$	$\frac{T_1}{T_2}$
2	0.2817	2.718	0.1036

n	$\frac{T_1}{T_d}$	$\frac{T_2}{T_d}$	$\frac{T_1}{T_2}$
3	0.8055	3.695	0.2180
4	1.425	4.464	0.3194

Avec T_1, le temps correspondant à l'intersection de la tangente au point d'inflexion et l'axe du temps et T_2, le temps correspondant à l'intersection de la tangente au point d'inflexion et l'axe du vecteur de sortie égal à l'unité (pour un échelon unité).

1.3 Définition du système à étudier

1.3.1 Sous la forme d'une fonction de transfert

[3]:
```
# Système à étudier
K_0 = 1.02              # Gain statique
TAU = 0.68e-3           # Constante de temps du 1er ordre
M = 0.4                 # Amortissement du 2nd ordre
W0 = 1/150e-6           # Pulsation du 2nd ordre

##############################################################################
# Fonction de transfert en BO
#

NUM = [K_0] # Numérateur de la fonction de transfert

# Dénominateur de la fonction de transfert par ordre décroissant en p ou s
DENUM = [TAU/W0**2, 2*M/W0*TAU + 1/W0**2, TAU + 2*M/W0, 1]

SYS_TF = ctl.tf(NUM, DENUM) # Fonction de transfert en p
```

1.4 Identification par la méthode de Strejc

1.4.1 Représentation de la réponse indicielle

Définition de la base de temps

[4]:
```
NB_T = 2**12 # Nombre de points de la réponse indicielle

T_MIN = 0 # Temps initial
T_MAX = 4.5e-3 # Temps final

TEMPS = np.linspace(T_MIN, T_MAX, NB_T) # Vecteur d'axe du temps

DELTA_T = T_MAX/NB_T # Pas d'echantillonnage
```

Calcul de la réponse indicielle

[5]:
```
t, ind = ctl.step_response(SYS_TF, TEMPS) # Calcul de la réponse indicielle en␣
↪boucle ouverte
```

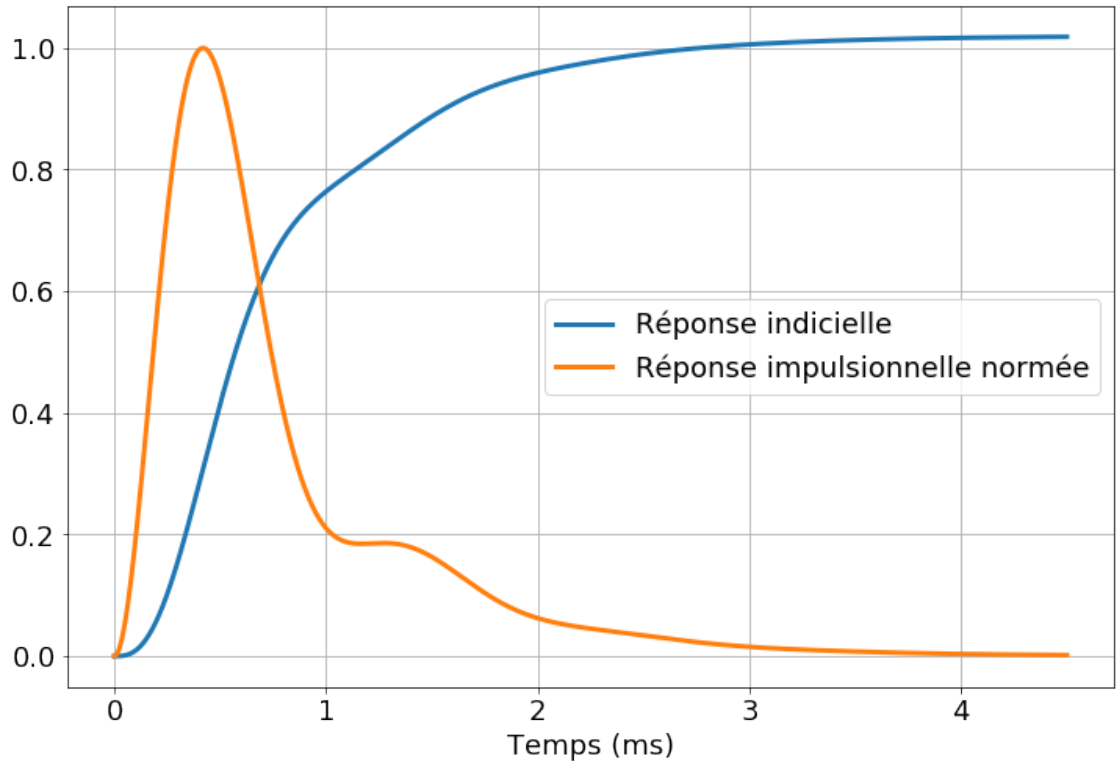
Calcul de la réponse impulsionnelle et de son maximum Le maximum de la réponse impulsionnelle correspond au point d'inflexion de la réponse indicielle

[6]:
```
t, imp = ctl.impulse_response(SYS_TF, TEMPS) # Calcul de la réponse␣
↪impulsionnelle en boucle ouverte

val_max = np.max(imp) # Valeur du maximum de la réponse impulsionnelle = point␣
↪d'inflexion de la réponse indicielle

arg_max = np.argmax(imp) # Indice du maximum de la réponse impulsionnelle =␣
↪point d'inflexion de la réponse indicielle
```

[7]:
```
imp = imp/val_max # Réponse impusionnelle normée à 1 (pour la visualisation)
```

Représentation de la réponse indicielle

[8]:
```
plt.figure(1, [12, 8])
plt.plot(t*1000, ind, label='Réponse indicielle')
plt.plot(t*1000, imp, label='Réponse impulsionnelle normée')
plt.grid(True)
plt.legend()
plt.xlabel('Temps (ms)')
plt.savefig("rep_ind_imp.png", dpi=300)
```



Calcul de la droite portée par la dérivée au point d'inflexion Au point d'infléxion t_i de la réponse indicielle, nous cherchons à trouver la droite d'équation :

$y(t) = at + b$

Une première équation est donnée en $t = t_i$: $y(t_i) = at_i + b$

Afin de trouver les paramètres a et b nous proposons une méthode des moindres carrées permettant de dimuninuer l'influence du bruit de mesure.

Nous prenons N points de mesure à droite et à gauche de t_i espacés de δT soit :

$y(t_i + n\delta T) = a(t_i + n\delta T) + b$

Ces $2N + 1$ équations sont mises sous la forme matricielle $\mathbf{y} = R\mathbf{p}$

Avec $\mathbf{p} = (a\, b)^t$ le vecteur des paramètres et la matrice des régresseurs $R = (t_i - N\delta T\, 1; ...; t_i\, 1; ...; t_i + N\delta T\, 1)$

L'algorithme des moindres carrés cherche à calculer la valeur des paramètres $\mathbf{p}$ qui permet de minimiser le critère :

$C(p) = \frac{1}{2}\left(y_{mesurebruite} - Rp\right)^2$

La solution de cette minimisation est obtenue pour :

$p_{mc} = \left(R^t R\right)^{-1} R^t y_{mesurebruite}$

[9]:
```
N_IND = 5 # Nombre de points pris dans la réponse indicielles 2N_IND+1
# Dans le cas d'une réponse bruitée, il est possible d'augmenter ce paramètre␣
↪pour effectuer un lissage du buit et ainsi
# avoir une meilleure précision sur les paramètres

R_MC = np.zeros((2*N_IND + 1, 2)) # Initialisation matrice des regresseurs

INDICE = np.linspace(-N_IND, N_IND, 2*N_IND + 1)

P_MC = np.zeros((2, 1)) # Initialisation du vecteur des paramètres
```

[10]:
```
for ite in range(2*N_IND + 1):
    R_MC[ite, :] = np.array((t[arg_max] + INDICE[ite]*DELTA_T, 1)) # Boucle de␣
↪remplissage de la matrice des regresseurs
```

[11]:
```
P_MC = np.linalg.solve(R_MC.T @ R_MC, R_MC.T @ ind[arg_max - N_IND: arg_max +␣
↪N_IND + 1])
```

[12]:
```
print("Les deux paramètres valent : " + str(P_MC[0]) + " et " + str(P_MC[1]))
```

```
Les deux paramètres valent : 1338.6330344676571 et -0.2521262879903778
```

Calcul des temps T_1 et T_2 Pour trouver ces deux valeurs, il faut résoudre les deux équations :

$1 = a(T_1 + T_2) + b$
$0 = aT_1 + b$

[13]:
```
T_1 = -P_MC[1]/P_MC[0]

T_2 = 1/P_MC[0]
```

[14]:
```
print("Les deux paramètres valent : T_1 =" + str(T_1*1000) + "ms et T_2=" +␣
↪str(T_2*1000) + "ms")
```

```
Les deux paramètres valent : T_1 =0.18834608253235174ms et
T_2=0.7470307203330572ms
```

[15]:
```
print("Le rapport T_1/T_2 =" + str(T_1/T_2))
```

```
Le rapport T_1/T_2 =0.2521262879903778
```

La valeur du rapport T_1/T_2 donne un ordre du système de 3

[16]:
```
T_D = T_1/0.8055
```

[17]:
```
print("La constante de temps estimée vaut =" + str(T_D*1000) + "ms")
```

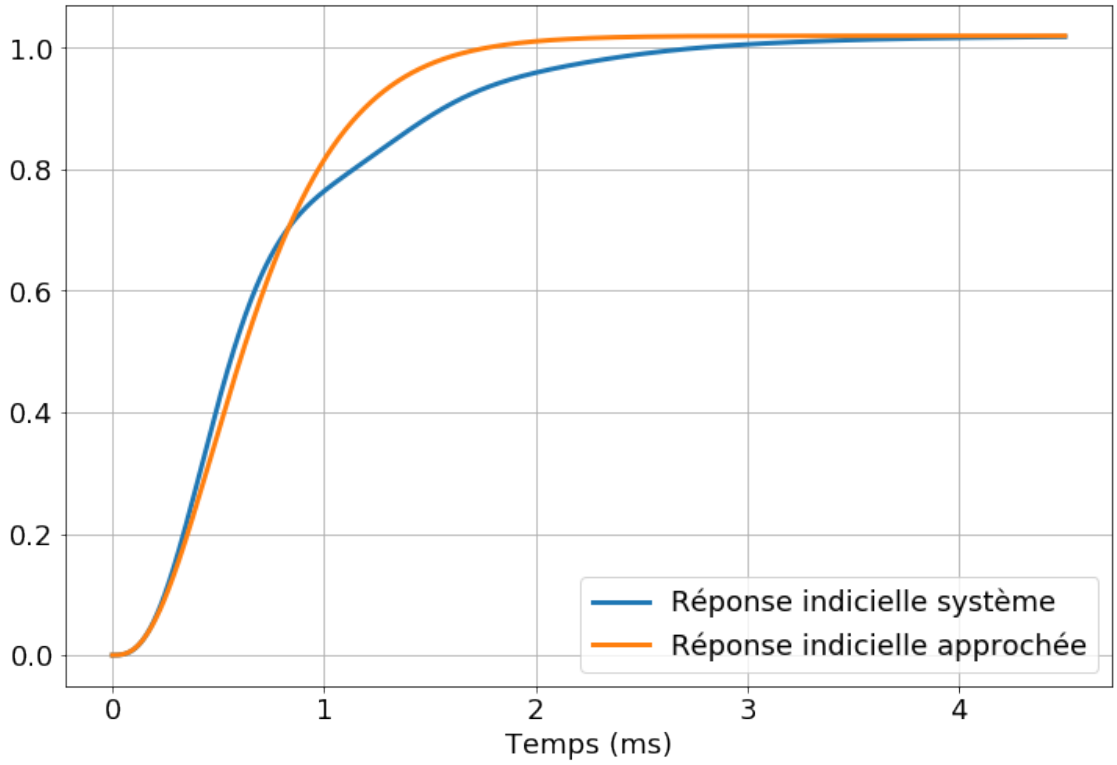
```
La constante de temps estimée vaut =0.23382505590608535ms
```

Comparaison des fonctions de transfert

[18]:
```
##############################################################################
# Fonction de transfert approximée
#

NUM_APP = [K_0] # Numérateur de la fonction de transfert

# Dénominateur de la fonction de transfert par ordre décroissant en p ou s
DENUM_APP = [T_D**3, 3*T_D**2, 3*T_D, 1]

SYS_APP = ctl.tf(NUM_APP, DENUM_APP) # Fonction de transfert en p
```

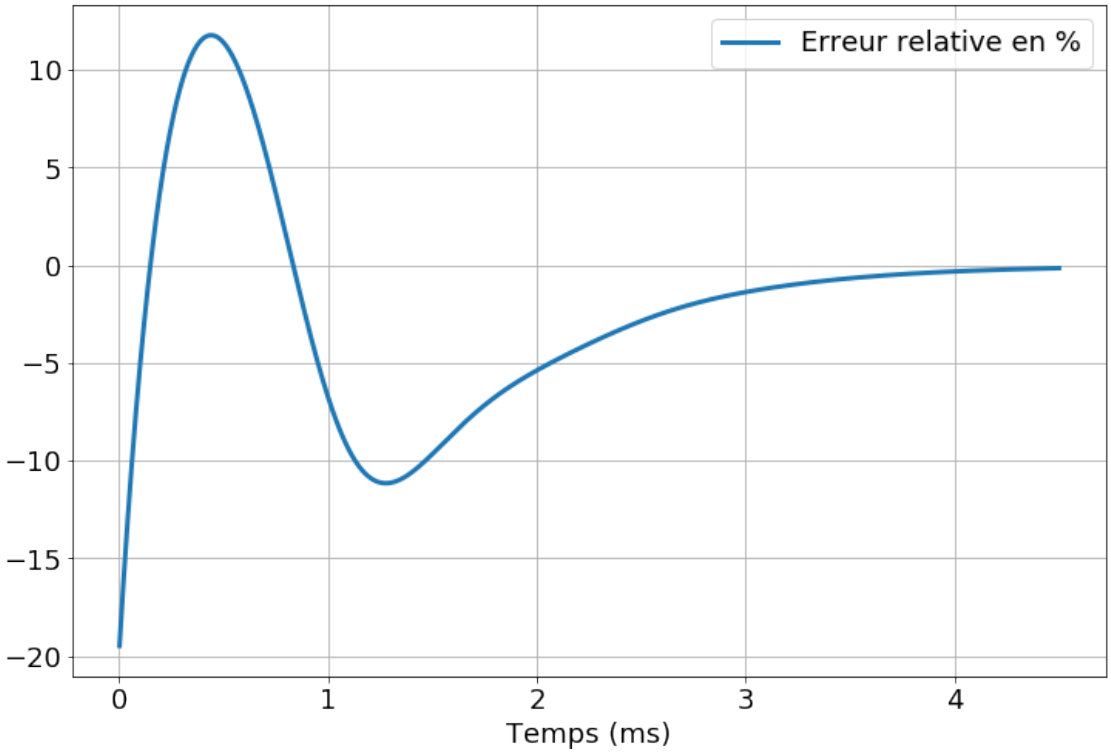
[19]:
```
t, ind_APP = ctl.step_response(SYS_APP, t) # Calcul de la réponse indicielle de␣
↪la fonction approximée
```

[20]:
```
plt.figure(2, [12, 8])
plt.plot(t*1000, ind, label='Réponse indicielle système')
plt.plot(t*1000, ind_APP, label='Réponse indicielle approchée')
plt.grid(True)
plt.legend()
plt.xlabel('Temps (ms)')
plt.savefig("rep_comp_sys_app.png", dpi=300)
```



Calcul de l'erreur

[21]:
```
plt.figure(3, [12, 8])
plt.plot(t*1000, (ind - ind_APP)/ind*100, label='Erreur relative en %')
plt.grid(True)
plt.legend()
plt.xlabel('Temps (ms)')
plt.savefig("rep_comp_err_sys_app.png", dpi=300)
```


Erreur relative en %
10
5
0
−5
−10
−15
−20
0
1
2
3
4
Temps (ms)

[ ]: