Apports des outils numériques sur l'enseignement de l'automatique : ateliers à partir de notebook Jupyter – Atelier 3

Culture Sciences de l'Ingénieur

Javier OJEDA

Édité le 06/07/2020

école normale supérieure paris—saclay

Atelier 3 : Représentation des pôles pour l'étude de la stabilité

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_3

June 12, 2020

0.1 Atelier n°3 : Représentation des pôles pour l'étude de la stabilité

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

0.2 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': 16}) # 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 sympy as sm # Package pour le calcul symbolique

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

0.3 Méthodologie

Objectif : Cet atelier doit permettre de mettre en place l'exploitation du lieu des pôles pour apprehender la stabilité des systèmes en boucle fermée.

Moyen : Les pôles des fonctions de transfert en boucle fermée peuvent être calculés numériquement sans l'utilisation de critères algébriques (Routh,...) puis représentés dans le plan des pôles (partie réelle et partie imaginaire).

0.4 Définition du système à étudier

0.4.1 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

# SYS_TF(p) = NUM(p) / DENUM(p)
```

Calcul des pôles par le calcul symbolique

[4]:
```
p = sm.symbols('p') # Variable de Laplace
```

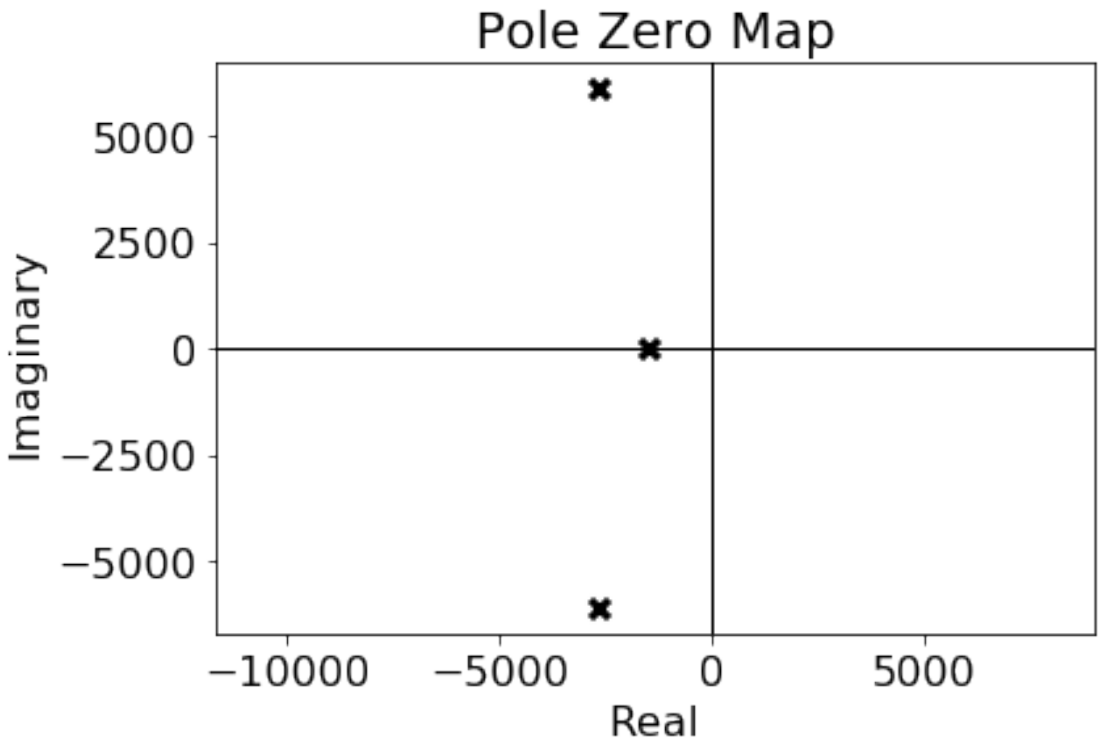
[5]:
```
# Résolution de l'équation caractéristique
POLES_TF_SM = sm.solve(DENUM[0]*p**3 + DENUM[1]*p**2 + DENUM[2]*p + DENUM[3], p)
```

[6]:
```
print('Les pôles sont ' + str(POLES_TF_SM[0]) + ' ; ' + str(POLES_TF_SM[1]) + '␣
↪; ' + str(POLES_TF_SM[2]))
```

```
Les pôles sont -1470.58823529412 ; -2666.66666666667 - 6110.10092660779*I ;
-2666.66666666667 + 6110.10092660779*I
```

Lieu des pôles en boucle ouverte

[7]:
```
POLES_TF, ZEROS_TF = ctl.pzmap(SYS_TF) # Calcul des pôles et des zéros puis␣
↪representation dans un plan

plt.savefig("lieuevans.png", dpi=300)
```



[8]:
```
print('Les pôles sont ' + str(POLES_TF[0]) + str(POLES_TF[1]) +␣
↪str(POLES_TF[2]))
```

```
Les pôles sont (-2666.666666666658+6110.100926607786j)(-2666.666666666658-6110.1
00926607786j)(-1470.5882352941185+0j)
```

0.5 Calcul des pôles pour une rétroaction et un correcteur proportionnel

Calcul des pôles par le calcul symbolique

[9]:
```
p = sm.symbols('p') # Variable de Laplace
K = sm.symbols('K') # Correcteur proportionnel
```

[10]:
```
POLES_BF_SM = sm.
↪solve(DENUM[0]*p**3+DENUM[1]*p**2+DENUM[2]*p+DENUM[3]+K*NUM[0], p)
```

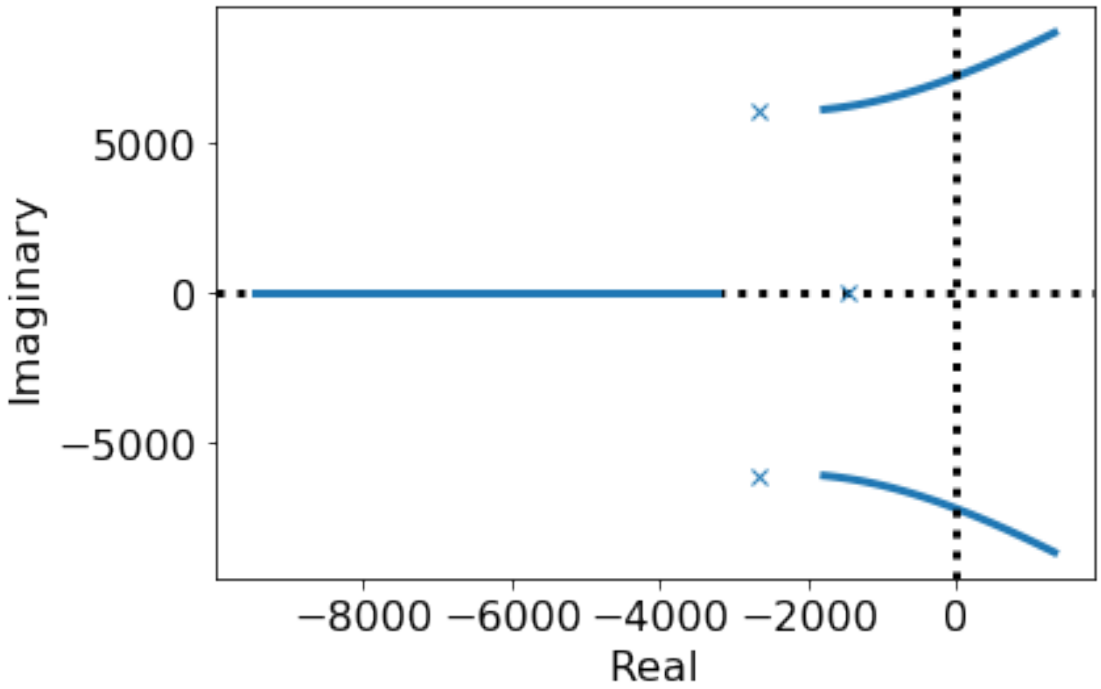
[11]:
```
print('Les pôles sont ' + str(POLES_BF_SM[0]) + ' ; ' + str(POLES_BF_SM[1]) + '␣
↪; ' + str(POLES_BF_SM[2]))
```

```
Les pôles sont -3505.06611739835*(-0.5 -
0.866025403784439*I)*(0.774087583500286*K + (0.599211586929312*(K -
0.448437185630799)**2 + 1)**0.5 - 0.347129657376614)**(1/3) - 2267.97385620915 +
3505.06611739835*(-0.5 + 0.866025403784439*I)/(0.774087583500286*K +
(0.599211586929312*(K - 0.448437185630799)**2 + 1)**0.5 -
```

```
0.347129657376614)**(1/3) ; -3505.06611739835*(-0.5 +
0.866025403784439*I)*(0.774087583500286*K + (0.599211586929312*(K -
0.448437185630799)**2 + 1)**0.5 - 0.347129657376614)**(1/3) - 2267.97385620915 +
3505.06611739835*(-0.5 - 0.866025403784439*I)/(0.774087583500286*K +
(0.599211586929312*(K - 0.448437185630799)**2 + 1)**0.5 -
0.347129657376614)**(1/3) ; -3505.06611739835*(0.774087583500286*K +
(0.599211586929312*(K - 0.448437185630799)**2 + 1)**0.5 -
0.347129657376614)**(1/3) - 2267.97385620915 +
3505.06611739835/(0.774087583500286*K + (0.599211586929312*(K -
0.448437185630799)**2 + 1)**0.5 - 0.347129657376614)**(1/3)
```

Lieu des pôles en boucle fermée

[12]:
```
VAL_K = np.linspace(1, 10, 100) # Valeurs souhaitées pour le correcteur␣
↪proportionnel

POLES_BF, K_POLES = ctl.root_locus(SYS_TF, VAL_K)

plt.savefig("lieuevansretour.png", dpi=300)
```



0.6 Calcul des pôles pour une rétroaction et un correcteur proportionnel intégral

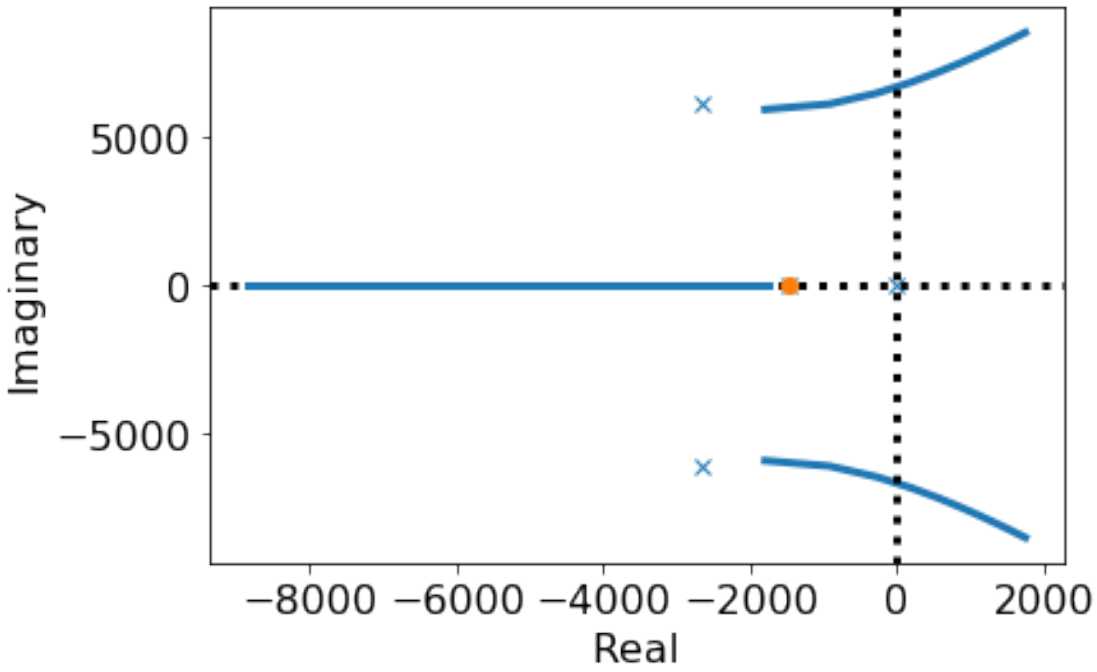
Lieu des pôles en boucle fermée

[13]:
```
# Correcteur C(p) = NUM_C(p) / DENUM_C(p)

NB_K = 10 # Nombre d'éléments dans le vecteur des paramètres du correcteur␣
 ↪proportionnel
MAX_K = 10 # Valeur maximale de K
MIN_K = 1 # Valeur minimale de K
VAL_K = np.linspace(MIN_K, MAX_K, NB_K) # Valeurs souhaitées pour le correcteur␣
 ↪proportionnel

NB_TI = 10 # Nombre d'éléments dans le vecteur des paramètres du correcteur␣
 ↪intégral
MAX_TI = 10e-4 # Valeur maximale de Ti
MIN_TI = 1e-4 # Valeur minimale de Ti
VAL_TI = np.linspace(MIN_TI, MAX_TI, NB_TI) # Valeurs souhaitées pour le␣
 ↪correcteur intégral
```

Nous cherchons le lieu des pôles pour le système augmenté en fixant la valeur de TI

[14]:
```
##############################################################################
# Correcteur PI par compensation de pôle dominant TAU
#

T_I = TAU

NUM_PI = [T_I, 1] # On fixe la valeur de TI

# Dénominateur de la fonction de transfert par ordre décroissant en p ou s
DENUM_PI = [T_I, 0]

SYS_PI = ctl.tf(NUM_PI, DENUM_PI) # Fonction de transfert en p
```

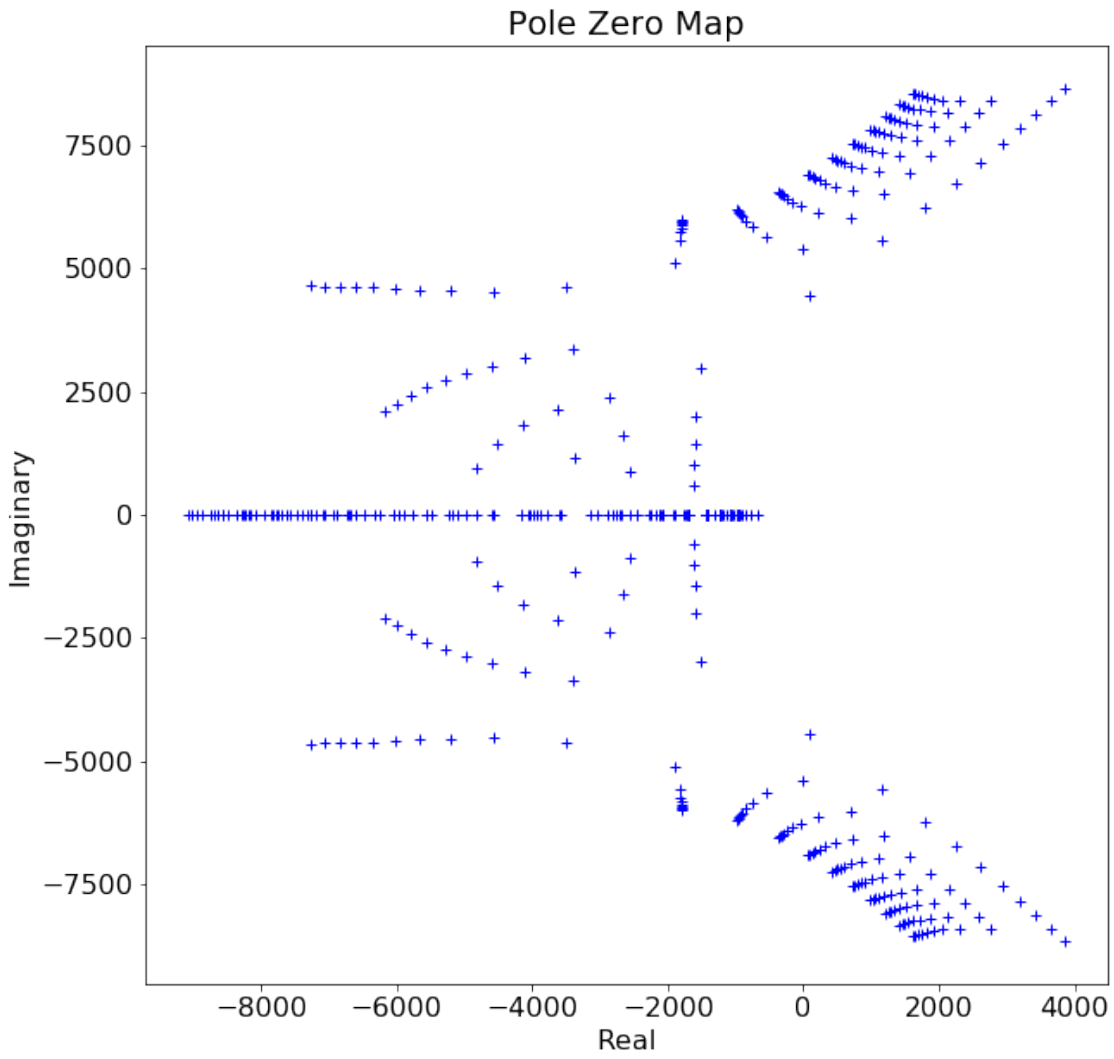
[15]:
```
POLES_BF, K_POLES = ctl.root_locus(SYS_TF*SYS_PI, VAL_K)

plt.savefig("lieuevansretourI.png", dpi=300)
```



Nous cherchons les pôles de $NUM(p) * NUM_C(p) + DENUM(p) * DENUM_C(p) = 0$

[16]:
```
# Boucle sur les valeurs de K et de TI

p = sm.symbols('p') # Variable de Laplace

RESULTATS = []
VALUES = []

for ite_K in range(NB_K):
    for ite_TI in range(NB_TI):
        POLES_BF_SM = sm.solve(VAL_TI[ite_TI]*p*(DENUM[0]*p**3 + DENUM[1]*p**2 +
                                                 DENUM[2]*p + DENUM[3]) +
                               VAL_K[ite_K]*(1 + VAL_TI[ite_TI]*p)*NUM[0], p)
        RES = [float(sm.re(POLES_BF_SM[0])) + 1j*float(sm.im(POLES_BF_SM[0])),
                     float(sm.re(POLES_BF_SM[1])) + 1j*float(sm.
 ↪im(POLES_BF_SM[1])),
                           float(sm.re(POLES_BF_SM[2])) + 1j*float(sm.
 ↪im(POLES_BF_SM[2])),
                                 float(sm.re(POLES_BF_SM[3])) + 1j*float(sm.
 ↪im(POLES_BF_SM[3]))]
        RESULTATS.append(RES)
        VALUES.append((VAL_K[ite_K], VAL_TI[ite_TI]))
```

[17]:
```
plt.figure(1, [10, 10])
plt.plot(np.real(RESULTATS), np.imag(RESULTATS), 'b+')
plt.xlabel("Real")
plt.ylabel("Imaginary")
plt.title("Pole Zero Map")
plt.savefig("lieuevansretourPI.png", dpi=300)
```



[ ]: