YALTAPy and YALTAPy_Online: Python toolboxes for the H∞-stability analysis of classical and fractional systems with commensurate delays

Hugo Cavalera, Jayvir Raj, Guilherme Mazanti, Catherine Bonnet

YALTAPy and YALTAPy_Online: Python toolboxes for the H_∞-stability analysis of classical and fractional systems with commensurate delays

Hugo Cavalera * **Jayvir Raj** ** **Guilherme Mazanti** *** **Catherine Bonnet** ***

* *Université Paris-Saclay, CNRS, CentraleSupélec, Inria, Laboratoire de signaux et systèmes, 91190 Gif-sur-Yvette, France*
** *Université Paris-Saclay, CNRS, CentraleSupélec, Inria, Laboratoire de signaux et systèmes, 91190 Gif-sur-Yvette, France and IPSA, 63 boulevard de Brandebourg, 94200 Ivry-sur-Seine, France*
*** *Université Paris-Saclay, CNRS, CentraleSupélec, Inria, Laboratoire de signaux et systèmes, 91190 Gif-sur-Yvette, France (e-mail: firstname.lastname@inria.fr)*

Abstract: The aim of this paper is to give a presentation of the Python toolbox YALTAPy dedicated to the stability study of standard and fractional delay systems as well as its online version YALTAPy_Online. Both toolboxes are derived from YALTA whose functionalities will be recalled here. Examples will be given to show how these toolboxes may be used.

Keywords: Stability of delay systems, fractional-order systems, Model reduction, Python Toolbox, Online Software

1. INTRODUCTION

The stability analysis of linear delay systems is a deep research subject which has been the source of countless studies in time and frequency domains since the 1950's, such as Pontryagin (1955); Bellman and Cooke (1963); Niculescu and Gu (2004); Richard (2003); Walton and Marshall (1987); Niculescu and Michiels (2014). More recently, the literature involved the stability study of fractional linear systems with delays, as fractional systems successfully model complex phenomena in a compact manner.

Many results have been obtained in both settings concerning stability characterization (asymptotic/exponential stability, H_∞-stability) of retarded delay systems. However, much remains to be done for neutral delay systems. Even though stability characterization (by sufficient or necessary and sufficient conditions) is a central issue, several related questions such as easy-to-check stability conditions, precise location of stable/unstable poles, behavior of stable/unstable poles when the delay varies, are of interest for a full and efficient study of practical situations.

In parallel of theoretical and numerical studies, e.g. Olgac and Sipahi (2004); Gumussoy and Michiels (2012); Hwang and Cheng (2006); Fioravanti et al. (2012), toolboxes have been developed to facilitate and popularize the use of the obtained results both in the academic and industrial environments. We cite here *Quasi-Polynomial Mapping Based Rootfinder* (QPmR, Vyhlídal and Zítek (2003)), *Tool for Robust Analysis and Characteristic Equations of Delay Differential Equations* (TRACE-DDE, Maset and Vermiglio (2005)), *Bifurcation analysis of delay differential equations* (DDE-Biftool, Engelborghs et al. (2001)), *Yet another LTI TDS Algorithm* (YALTA, Avanessoff et al. (2013)), *Partial pole placement via delay action* (P3δ, Boussaada et al. (2020, 2021)), and *Parallel Processing Delay Margin Finder* (parDMF, Ramírez et al. (2021)). The references Avanessoff et al. (2013, 2014) present the functionalities of the Matlab toolbox YALTA and briefly describe its positioning relative to earlier toolboxes. We also refer to Sipahi (2019) and Pekar and Gao (2018) for a recent overview on effective methods for the stability analysis of delay systems where toolboxes are also presented.

The aim of this paper is to present YALTAPy, a Python version of the Matlab toolbox YALTA, as well as YALTAPy_Online, which is an online version of YALTAPy. Note that YALTAPy is an open source software.

In Section 2 we will describe the main functionalities of YALTAPy (i.e., briefly recall those of YALTA). Section 3 will detail a bit the features of YALTAPy_Online. Then examples of use of YALTAPy and YALTAPy_Online will respectively be given in Sections 4 and 5.

2. MAIN FEATURES OF YALTAPY

YALTAPy and YALTAPy_Online consider delay systems described by transfer functions of the type:

$$G(s) = \frac{t(s)+\sum_{\kappa=1}^{N'} t_\kappa(s)e^{-\kappa\tau s}}{p_0(s)+\sum_{k=1}^{N} p_k(s)e^{-k\tau s}} = \frac{n(s)}{d(s)}, \qquad (1)$$

where $\tau > 0$ is the nominal delay, $\alpha \in (0,1]$ is the fractional exponent, and, for every $k \in \{0,\dots,N\}$, $p_k(s)$ is a polynomial in the variable s^α. We let n denote the degree

of p_0 and we assume that $n \geq \deg t$, $n \geq \deg t_\kappa$ for every $\kappa \in \{1, \ldots, N'\}$, and $n \geq \deg p_k$ for every $k \in \{1, \ldots, N\}$, i.e., the system is either of retarded or neutral type.

We refer to Bellman and Cooke (1963); Fioravanti et al. (2010a); Partington (2004) for an analysis of the position of chains of poles of such systems:

(1) If $\deg p_0 > \deg p_k$ for all $k \in \{1, \ldots, N\}$, then there are only chains of retarded type;
(2) If $\deg p_0 = \deg p_N$, then there are only chains of neutral type;
(3) If $\deg p_0 = \deg p_k > \deg p_N$, for some $k \in \{1, \ldots, N-1\}$, then there are chains of both neutral and retarded types.

We recall that a neutral chain is asymptotic to a vertical axis in the complex plane, while a retarded chain contains poles with arbitrarily large negative real part. The coefficient of the highest degree term of $p_0(s)+\sum_{k=1}^{N} p_k(s)e^{-k\tau s}$ is given by

$$\tilde{c}_d(z) = 1 + \sum_{i=1}^{N} \alpha_i z^i. \tag{2}$$

When dealing with neutral systems, in order to avoid the possibility of an infinite number of zero cancellations between the numerator and denominator of G, we make the same Hypothesis (H) than in Avanessoff et al. (2013, 2014) in terms of $\deg t$, $\deg t_k$ and roots of $\tilde{c}_d$ and $\tilde{c}_n$.

We refer to Avanessoff et al. (2013, 2014) for a complete description of the functionalities of YALTA and just precise below that YALTAPy and YALTAPy_Online will give:

- the type of delay system: retarded or neutral
- the position of asymptotic axes for a neutral system
- the stability property when the delay is zero
- stability windows
- a root locus showing the displacement of the poles when the delay varies between two chosen values
- a Padé-2 approximation of the system (only for standard systems).

Let us now describe how inputs should be provided to YALTAPy functions. The quasi-polynomial $d(s)$ may be rewritten

$$d(s) = P_0(s) + \sum_{k=1}^{M} P_k(s)e^{-n_k\tau s}, \tag{3}$$

where, for every $k \in \{0, \ldots, M\}$, n_k is a positive integer and $P_k(s) = p_{n_k}(s)$. In other words, we eliminate from the denominator of (1) all delays $k\tau$ corresponding to values of k for which p_k is identically zero, and $M \in \{1, \ldots, N\}$ denotes thus the number of polynomials among $p_1, \ldots, p_N$ that are not identically zero.

The data of the system corresponding to (3) is hence described by the coefficients of the polynomials $P_0, \ldots, P_M$, the exponent α, the delay τ, and the integer multiples $n_1, \ldots, n_M$ of the delay τ that appear in the system.

In order to input the coefficients of $P_0, \ldots, P_M$ to YALTAPy, one should provide the $(M+1) \times (n+1)$ matrix

$$\begin{pmatrix} p_{0,n} & p_{0,n-1} & \cdots & p_{0,0} \\ p_{1,n} & p_{1,n-1} & \cdots & p_{1,0} \\ \vdots & \vdots & \ddots & \vdots \\ p_{M,n} & p_{M,n-1} & \cdots & p_{M,0} \end{pmatrix},$$

i.e., the matrix whose kth row (starting from $k = 0$) describes the coefficients of P_k in descending order of power. This matrix should be represented in Python as a 2-dimensional `numpy` array (see Harris et al. (2020) for details on `numpy`). Concerning the integer multiples $n_1, \ldots, n_M$ of the delay τ, they should be input as a vector of size M,

$$(n_1 \ \cdots \ n_M),$$

which should be represented as a 1-dimensional `numpy` array of integers. Finally, τ and α should be input to YALTAPy as floating-point numbers or integers. Examples of use of YALTAPy are provided in Section 4 below.

3. YALTAPY_ONLINE

The team working on YALTA has recently developed an interface for facilitating the use of YALTAPy, based on a Python's *Jupyter Notebook*, an open document format which can contain live code, equations, visualizations, and text. Our Jupyter Notebook implements a friendly graphical user interface for YALTAPy_Online thanks to interactive widgets from Python's `ipywidgets` module. The next aim is to facilitate the use of YALTAPy_Online by making it available with no need of installation thanks to the *Binder* service (Project Jupyter et al., 2018), which allows the creation of personalized computing environments directly from a *git* repository. The *Binder* service is free to use and is powered by *BinderHub*, an open-source tool that deploys the service in the cloud.

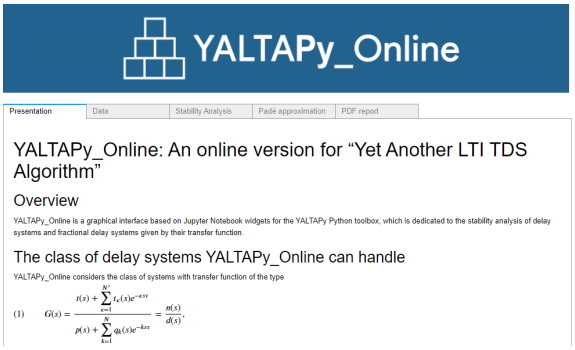

Fig. 1. Main screen of YALTAPy_Online.

YALTAPy_Online starts by a presentation screen (see Fig. 1) recalling the class of systems that it handles and its main features. It is divided into five tabs: the first one is the presentation screen, the second one is dedicated to the input of the data, the third one provides features of stability analysis, stability windows, and root locus, the third one provides a tool for computing Padé approximations, and the last one allows the user to obtain a PDF report with the obtained results. A detailed example of the use of YALTAPy_Online is provided in Section 5.

4. EXAMPLES OF USE OF YALTAPY

In this section, we present two examples illustrating the use of YALTAPy for the stability analysis of time-delay systems. The first one, described in Section 4.1, considers a time-delay system of order 3 with a single delay, whereas the second one, described in Section 4.2, illustrates how to use YALTAPy to study the stability of a fractional system.

4.1 A third-order system

Consider the time-delay system given by

$$\dot{x}(t) = A_0 x(t) + A_1 x(t-\tau), \tag{4}$$

where $x(t) \in \mathbb{R}^n$, A_0 and A_1 are $n \times n$ matrices with real coefficients, and $\tau > 0$ is the delay. The stability of such a system can be studied through the roots of its characteristic equation

$$d(s) = \det(s\,\mathrm{Id} - A_0 - A_1 e^{-s\tau}),$$

where Id denotes the $n \times n$ identity matrix.

In this example, we consider $n = 3$, $\tau = 3$, and

$$A_0 = \begin{pmatrix} 0 & 1 & -1 \\ 1 & -\frac{1}{2} & -\frac{1}{4} \\ -1 & 0 & -\frac{4}{5} \end{pmatrix}, \qquad A_1 = \begin{pmatrix} -3 & -2 & \frac{1}{2} \\ -1 & 0 & \frac{1}{5} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{pmatrix}.$$

In this case, a straightforward computation shows that

$$d(s) = s^3 + \tfrac{13}{10}s^2 - \tfrac{8}{5}s - \tfrac{31}{20} + + \left(3s^2 + \tfrac{163}{20}s + \tfrac{323}{60}\right) e^{-\tau s} - \left(\tfrac{12}{5}s + \tfrac{173}{60}\right) e^{-2\tau s} + \tfrac{7}{30} e^{-3\tau s}. \tag{5}$$

In order to input the characteristic equation (5) into YALTAPy, we define the `numpy` array

```
P = np.array([[1, 13/10,  -8/5 ,  -31/20],
              [0,     3, 163/20,  323/60],
              [0,     0,  -12/5, -173/60],
              [0,     0,      0,    7/30]])
```

(assuming that `numpy` was previously imported using `import numpy as np`). Similarly, we input the vector `n` of the multiples of the delays and the values of τ and α by

```
n = np.array([1, 2, 3]), tau = 3, alpha = 1
```

Stability analysis With the previous definitions, one studies the stability of (4) by using YALTAPy's `thread_analysis` function. After importing YALTAPy using `import yaltapy as yp` and defining `P`, `n`, `tau`, and `alpha` as above, the `thread_analysis` function is called by

```
res = yp.thread_analysis(P, n, alpha, tau)
```

After its execution, `thread_analysis` returns a dictionary with information on the stability of the system. In this example, the variable `res` contains the following data:

```
Type: Retarded
AsympStability: There is (are) 4 unstable
pole(s) in the right half-plane
RootsNoDelay: [-3.07585975+0.j
         -0.61207012+0.10043131j
         -0.61207012-0.10043131j]
RootsChain: Roots chains only computed for
neutral systems
CrossingTable:
 [[0.38196552 1.86170973 3.37495433 1.        ]]
ImaginaryRoots:
[[ 0.38196552  3.37495433  1.        ]
 [ 0.38196552 -3.37495433  1.        ]
 [ 2.24367525  3.37495433  1.        ]
 [ 2.24367525 -3.37495433  1.        ]]
```

Hence, YALTAPy has identified that the given system is of retarded type and that it has 4 poles in the right half-plane, yielding the conclusion that, with the given parameters, the system is unstable. YALTAPy has also computed the characteristic roots of the system without delay, i.e., when $\tau = 0$. In this case, the characteristic equation of the system reduces to the polynomial $d(s) = s^3 + \frac{43}{10}s^2 + \frac{83}{20}s + \frac{71}{60}$, whose three complex roots are those provided in YALTAPy output `RootsNoDelay`.

Since the system is of retarded type, it presents no chains of roots asymptotic to a vertical line, a fact that is recalled in the output `RootsChain`. The outputs `CrossingTable` and `ImaginaryRoots` describe the behavior of the system as the delay increases from 0 to its nominal value. For instance, `ImaginaryRoots` states that, at $\tau \approx 0.3820$, two roots cross the imaginary axis, at the values $\pm 3.3750i$, and the direction of crossing is from the left to the right (value 1 in the third column), and, at $\tau \approx 2.2437$, two other roots cross the imaginary axis, once again at the values $\pm 3.3750i$ and from the left to the right. As for the `CrossingTable`, its first column gives the first value of the delay for which a specific zero crosses the axis, its third column gives the frequency ω of crossing of a zero, its second column is $\frac{2\pi}{\omega}$, and its fourth column provides the crossing direction.

Stability windows YALTAPy provides a function to obtain stability windows for the system under consideration, called `thread_stability_windows`. After defining `P`, `n`, `tau`, and `alpha` as above, this function is called by

```
res = yp.thread_stability_windows(P, n,
      alpha, tau)
```

As for `thread_analysis`, the function `thread_stability_windows` also returns a dictionary with information on stability windows for the system. In the present example, in addition to outputs already given by `thread_analysis`, the variable `res` contains the following data:

```
StabilityWindows:
 [[0.         0.38196552 3.         ]
  [1.         0.         0.        ]]
NbUnstablePoles:
 [[0.         0.38196552 2.24367525 3.        ]
  [0.         2.         4.         4.        ]]
```

The outputs `StabilityWindows` and `NbUnstablePoles`, in this example, are interpreted as follows: at $\tau = 0$, the system is stable (value 1 at `StabilityWindows`) and has no poles in the right half-plane (value 0 at `NbUnstablePoles`). At $\tau \approx 0.3820$, the system becomes unstable (value 0 at `StabilityWindows`) and has 2 poles in the right half-plane (value 2 at `NbUnstablePoles`). At $\tau \approx 2.2437$, two additional poles appear in the complex right half-plane, and the system remains unstable and with 4 poles in the right half-plane until the nominal delay value $\tau = 3$.

In addition to those outputs, `thread_stability_windows` also plots the data in `StabilityWindows` and `NbUnstablePoles` by setting its optional input argument `plot` to `True` (default). The plot for this example is provided in Fig. 2.

Root locus The root locus of the roots of a system that eventually become unstable is computed using YALTAPy's function `thread_root_locus`. For systems of neutral type, this function only runs if the system with nominal delay possesses chains asymptotic to vertical axes located in the

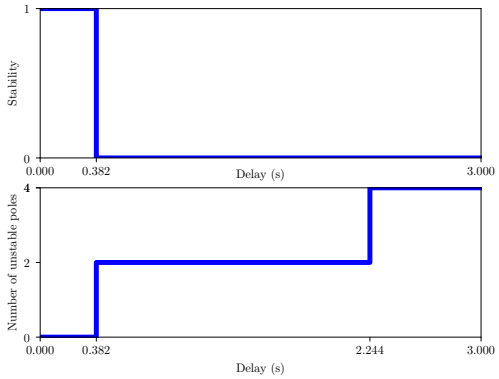

Fig. 2. Stability windows for the example from Section 4.1.

open left half-plane.Similarly to the previous functions, this function is called by

```
res = yp.thread_root_locus(P, n, alpha, tau)
```

and it returns a dictionary with information on the root locus. In the present example, the dictionary `res` returned by the function contains, in addition to outputs also provided by the previous functions, the outputs

```
UnstablePoles:
[0.08427953+2.50094915e+00j
 0.08427953-2.50094915e+00j
 1.01660266+2.18573355e-13j]
PolesError: [1.e-11+0.j 1.e-12+0.j 1.e-10+0.j]
```

The dictionary also contains an entry `RootLocus` containing information on the movement of the roots, but since it is too long, it is not shown here.

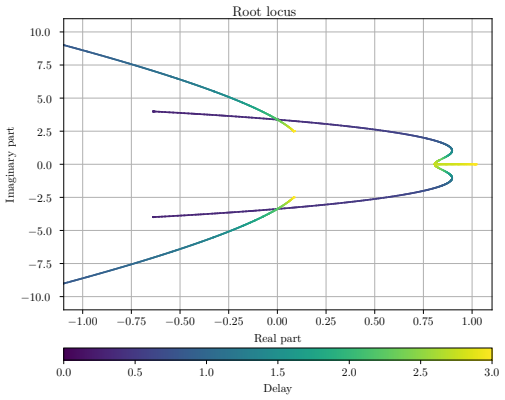

Fig. 3. Root locus for the example from Section 4.1.

The output `UnstablePoles` provides the location of the 4 unstable poles at the nominal value of the delay $\tau = 3$ (we have a real pole of multiplicity 2 in this example) and the respective estimated numerical errors in `PolesError`. If the optional input argument `plot` is set to `True`, the root locus is graphically displayed, as shown in Fig. 3, where we observe the behavior of the four roots of the system that become unstable as the delay increases from 0 to 3.

4.2 A fractional system

As a second example, we consider the fractional system described by its characteristic equation

$$d(s) = s^{\frac{1}{2}} + \frac{15}{2} - \left(\frac{1}{6}s^{\frac{1}{2}} + \frac{20}{7}\right)e^{-\tau s} + \frac{80}{11}e^{-3\tau s},$$

with $\tau = 2.5$. Such a system is input into YALTAPy by defining the variables `P`, `n`, `tau`, and `alpha` as

```
P = np.array([[   1,  15/2],
              [-1/6, -20/7],
              [   0,  80/11]])
n = np.array([1, 3]), alpha = 0.5, tau = 2.5
```

Stability analysis Similarly to the example in Section 4.1, we run `thread_analysis` in the present example, obtaining as a result

```
Type: Neutral
AsympStability: There is (are) 4 unstable
pole(s) in the right half-plane
RootsNoDelay: []
RootsChain: [-0.71670379]
CrossingTable:
[[0.42100844 2.96762299 2.11724512 1.        ]
 [1.96353668 2.32594339 2.70134919 1.        ]]
ImaginaryRoots:
[[ 0.42100844  2.11724512  1.        ]
 [ 0.42100844 -2.11724512  1.        ]
 [ 1.96353668  2.70134919  1.        ]
 [ 1.96353668 -2.70134919  1.        ]]
```

YALTAPy identifies that the system has a single chain of poles asymptotic to the vertical line with real part -0.7167, indicates that the system is unstable for the nominal value of the delay, and identifies the crossings of the imaginary axis as the delay increases from 0 to 2.5, providing the value of the delay at which the crossing occurs and the position and direction of crossing.

Stability windows Using YALTAPy to compute stability windows, we obtain the following output (only fields not already returned by `thread_analysis` are shown)

```
StabilityWindows:
[[0.         0.42100844 2.5        ]
 [1.         0.         0.        ]]
NbUnstablePoles:
[[0.         0.42100844 1.96353668 2.5        ]
 [0.         2.         4.         4.        ]]
```

Root locus Finally, we use YALTAPy to compute the root locus of the present example, obtaining the following output (outputs already present in the previous functions are omitted, as well as the output `RootLocus`, too long to be shown here)

```
UnstablePoles:
[0.00224791+2.12366734j
 0.00224791-2.12366734j
 0.01138514-0.36488849j
 0.01138514+0.36488849j]
PolesError: [1.e-10+0.j 1.e-13+0.j
             1.e-12+0.j 1.e-11+0.j]
```

The corresponding root locus is presented in Fig. 4, where we observe in particular the crossings already known from the previous computation of stability windows.

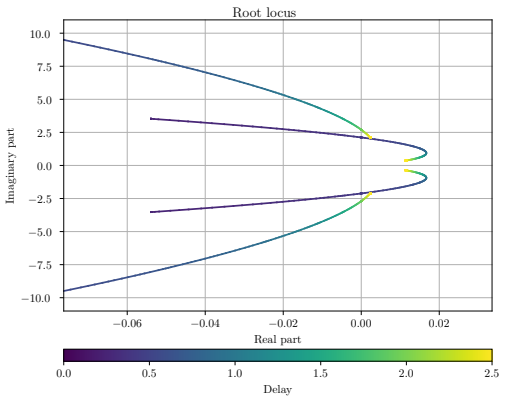

Fig. 4. Root locus for the example from Section 4.2.

4.3 Padé approximation

YALTAPy can compute Padé approximation of functions of the form $(p_0(s) + \sum_{k=1}^{N} p_k(s)e^{-ks\tau})/(s+1)^\delta$ where $p_0(s), \ldots, p_k(s)$ are polynomials in the variable s following the same assumptions as before and δ is an integer greater than the degree of p_0. We illustrate this functionality of YALTAPy in the case $\delta = 2$ and

$$d(s) = s + 2 - e^{-s}. \tag{6}$$

To input such a system into YALTAPy, we define

```
P = np.array([[1, 2], [0, -1]]),
n = np.array([1]), tau = 1, delta = 2
```

Padé approximations can be computed through YALTAPy's `compute_pade` function, which, in addition to the above variables, takes as arguments also the mode of approximation ("ORDER" or "NORM") and a parameter `mod_arg` whose interpretation depends on the mode: `mod_arg` is the order of approximation in mode "ORDER" and the maximum error in H_∞ norm for the approximation in mode "NORM". For instance, a first-order Padé approximation of (6) is obtained by

```
res = yp.compute_pade(P, delta, tau, n,
        mod_arg = 1, mode = "ORDER")
```

The variable `res` will contain the data of the approximation: `res.num_approx` and `res.den_approx` contain the coefficients of the numerator and denominator, respectively, of the Padé approximation, while `res.error_norm` and `res.pade_order` contain, respectively, the H_∞ norm of the approximation error and the order of approximation. After running the present example, these four variables contain, respectively, the values

```
array([ 1.,  9., 45., 79., 54., 12.])
array([ 1., 10., 42., 88., 97., 54., 12.])
0.032917024033214635
1
```

5. EXAMPLE OF USE OF YALTAPY_ONLINE

Let us now illustrate the use of YALTAPy_Online through the example from Section 4.1 above. The first step is to

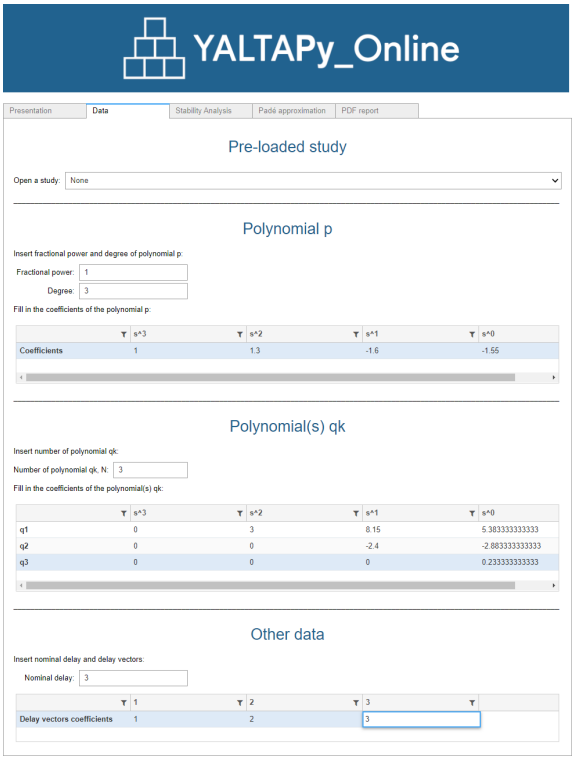

Fig. 5. Data input in YALTAPy_Online.

input the data of the problem in the tab "Data" (see Fig. 5). The polynomials without delays and with delays are entered separately. After selecting the fractional power of s^α (which is $\alpha = 1$ in this example) and degree of the polynomial P_0 without delay terms (3 in this example), the user enters its coefficients in the corresponding field. The user is then prompted for the number of delayed terms and for the coefficients of the corresponding polynomials. Finally, the last part is dedicated to the nominal delay and the vector containing the integers multiplying the delay.

After entering the data, the user goes to the tab "Stability Analysis" to obtain YALTAPy_Online's study of the system. As YALTAPy, YALTAPy_Online will perform a stability analysis (corresponding to YALTAPy's function `thread_analysis`), compute stability windows (corresponding to `thread_stability_windows`), and compute root locus of unstable roots (corresponding to `thread_root_locus`). Those analyses are started by the respective "Run" buttons on the page and all results are displayed on the screen. Long outputs are automatically hidden, but can be shown through the corresponding "Show" buttons. Graphs are interactive, and the user can zoom in parts of the graph or export it as an image. The screen obtained after running all available analyses for the example from Section 4.1 is provided in Fig. 6 (the root locus was omitted due to space constraints).

REFERENCES

D. Avanessoff, A. Fioravanti, and C. Bonnet (2013). YALTA: a Matlab toolbox for the H_∞-stability analysis of classical and fractional systems with commensurate delays. *11th IFAC Workshop on Time-Delay Systems*.

D. Avanessoff, A. Fioravanti, C. Bonnet, and L.H. V. Nguyen (2014). H_∞-stability analysis of (fractional)

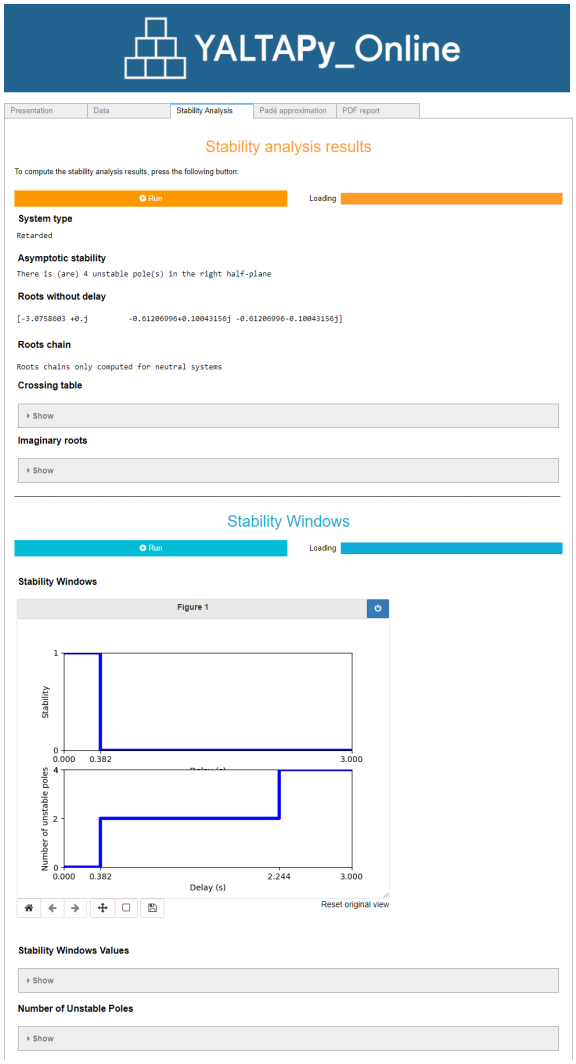

Fig. 6. Stability analysis in YALTAPy_Online (root locus omitted for simplicity).

delay systems of retarded and neutral type with the Matlab Toolbox YALTA. *Delay systems, from theory to numerics and applications*, ADDS Series, Springer.

R. Bellman and K. Cooke (1963). *Differential-Difference Equations*. Academic Press.

I. Boussaada, G. Mazanti, S.-I. Niculescu, J. Huynh, F. Sim, and M. Thomas (2020). Partial pole placement via delay action: A Python software for delayed feedback stabilizing design. *ICSTCC 2020*, pp. 196–201.

I. Boussaada, G. Mazanti, S.-I. Niculescu, A. Leclerc, J. Raj, and M. Perraudin (2021). New Features of P3δ software: Partial Pole Placement via Delay Action. *IFAC-PapersOnLine*, 54(18), pp. 215–221.

K. Engelborghs, T. Luzyanina, and G. Samaey (2001). DDE-BIFTOOL v. 2.00: a Matlab package for bifurcation analysis of delay differential equations. *Technical Report TW-330, K.U.Leuven*, Belgium.

A.R. Fioravanti, C. Bonnet, H. Özbay, and S.-I. Niculescu (2010). A numerical method to find stability windows and unstable poles for linear neutral time-delay systems. *9th IFAC Workshop on Time Delay Systems*.

A.R. Fioravanti, C. Bonnet, H. Ozbay, and S.-I. Niculescu (2012). A numerical method for stability windows and unstable root-locus calculation for linear fractional time-delay systems. *Automatica*, 48(11), pp. 2824–2830.

S. Gumussoy and W. Michiels (2012). Root-locus for SISO dead-time systems: A continuation based approach. *Automatica*, 43(3), pp. 480–489.

C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J.F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T.E. Oliphant (2020). Array programming with NumPy. *Nature*, 585(7825), pp. 357–362.

C. Hwang and Y.C. Cheng (2006). A numerical algorithm for stability testing of fractional delay systems. *Automatica*, 42(5), pp. 825–831.

S. Maset and R. Vermiglio (2005). Pseudospectral differencing methods for characteristic roots of delay differential equations. *SIAM J. Sci. Comput.*, 27(2), pp. 482–495.

S.-I. Niculescu and K. Gu (2004). *Advances in Time-Delay Systems*. Springer.

S.-I. Niculescu and W. Michiels (2014). *Stability, control, and computation of time-delay systems. An eigenvalue based approach*. Advances in Design and Control 27, SIAM, Philadelphia, PA, Second Edition.

N. Olgac and R. Sipahi (2004). A practical method for analyzing the stability of neutral type LTI-time delayed systems. *Automatica*, 40, pp. 847–853.

L. Pekar and Q. Gao (2018). Spectrum Analysis of LTI continuous-time systems with constant delays: a literature overview of some recent results. *IEEE Access*, July 2018.

J.R. Partington (2004). *Linear Operators and Linear Systems: An Analytical Approach to Control Theory*. Cambridge University Press.

L.S. Pontryagin (1955). On the zeros of some elementary transcendental functions. *Amer. Math. Soc. Transl.*, 2(1), pp. 95–110.

Project Jupyter, M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, M. Pacer, Y. Panda, F. Perez, B. Ragan Kelley, and C. Willing (2018). Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. In F. Akici, D. Lippa, D. Niederhut, and M. Pacer (eds.), *Proceedings of the 17th Python in Science Conference*, pp. 113–120.

A. Ramírez, D. Breda, and R. Sipahi (2021). A scalable approach to compute delay margin of a class of neutral-type time delay systems. *SIAM J. Control Optim.* 59(2), pp. 805–824.

J.P. Richard (2003). Time-Delay Systems: An Overview of Some Recent Advances and Open Problems, *Automatica*, 39, pp. 1667–1694.

R. Sipahi (2019). *Mastering Frequency Domain Techniques for the Stability Analysis of LTI Time Delay Systems*. SIAM.

T. Vyhlídal and P. Zítek (2003). Quasipolynomial mapping based rootfinder for analysis of Time delay systems, *Proc. of IFAC Workshop on Time-Delay Systems*.

K. Walton and J.E. Marshall (1987). Direct method for TDS stability analysis. newblock *IEE Proceedings*, 134, pp. 101–107.