User documentation
YALTApy

Catherine Bonnet Hugo Cavalera Guilherme Mazanti

11 August 2021

Contents

1 The class of delay systems YALTAPy can handle

YALTAPy is a Python Toolbox dedicated to the stability analysis of (fractional) delay systems given by their transfer function.

YALTAPy considers the class of systems with transfer function of the type

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

where

- $\tau > 0$ is the nominal delay,
- t, p, q_k for all $k \in \mathbb{N}_N$, and t_κ for all $\kappa \in \mathbb{N}_{N'}$ are real polynomials in s^α for $0 < \alpha \leq 1$ which satisfy
  - $\deg p \geq \deg t$,
  - $\deg p \geq \deg t_\kappa$ for all $\kappa \in \mathbb{N}_{N'}$,
  - $\deg p \geq \deg q_k$ for all $k \in \mathbb{N}_N$.

  Here, the degree is interpreted as the degree in s^α.

If $\deg p = \deg q_k$ for at least one $k \in \mathbb{N}_N$, G defines a *neutral* time-delay system, otherwise defines a *retarded* system [2].

We write $z = e^{-s\tau}$ and suppose that for each k

$$\frac{q_k(s)}{p(s)} = \alpha_k + \mathcal{O}(s^{-\alpha}) \qquad \text{as} \quad |s| \to \infty. \tag{2}$$

The coefficient of the highest degree term of $p(s) + \sum_{k=1}^{N} q_k(s) e^{-ks\tau}$ can then be written as a multiple of the following polynomial in z

$$\tilde{c}_d(z) = 1 + \sum_{i=1}^{N} \alpha_i z^i. \tag{3}$$

Hypothesis (H1). The roots of $\tilde{c}_d$ are of multiplicity one.

Hypothesis (H2). The polynomials $p(s)$ and $q_k(s)$ satisfy

$$p(0) + \sum_{k=1}^{N} q_k(0) \neq 0.$$

Hypothesis (H3). In order to avoid the possibility of an infinite number of zero cancellations between the numerator and denominator of G, we suppose that the numerator of G satisfies either

a) $\deg t(s) > \deg t_k(s)$; or
b) $\deg t_k(s) = \deg t(s)$ for at least one k and the polynomial $\tilde{c}_n$ defined as in (3) relatively to the quasi-polynomial $n(s)$ has no root of modulus less than or equal to one, and no common root of modulus strictly greater than one with $\tilde{c}_d$.

2 The functionalities of YALTAPy

The questions YALTAPy can answer are:

- For neutral systems:
    - Find the position of asymptotic axes.
    - If the imaginary axis is an asymptotic axis, find if the asymptotic poles of the chain are to the left or to the right of the imaginary axis.
- In the case of retarded systems or of neutral systems with asymptotic axes in $\{\operatorname{Re} s < 0\}$, find:
    - For a given τ, the number and the position of unstable poles.
    - For which values of τ the system is stable;
    - For a set of values of the delay, the position of unstable poles (root locus).
    - The coprime factors (N, D) of G as well as an approximation (N_n, D_n) in H_∞-norm

YALTAPy (as YALTA) is based on [1, 3–8].

YALTAPy contains four main functions for the analysis of (1), which are described in Sections 3–6.

3 thread_analysis

Launches the analysis of a SISO delay system given by its transfer function G as in (1). The delay system must be of retarded type or neutral type, but in the latter case some outputs are available only for neutral systems with a finite number of poles in $\{\operatorname{Re}(s) > -a\}$ for some $a > 0$.

3.1 Syntax

```
res = thread_analysis(poly_mat, delay_vect, alpha, tau)
```

3.2 Inputs

- `poly_mat`: the quasi-polynomial in the denominator of (1), represented as the $(N+1) \times (n+1)$ matrix of its coefficients

$$\begin{pmatrix} p_n & p_{n-1} & \cdots & p_0 \\ q_{1,n} & q_{1,n-1} & \cdots & q_{1,0} \\ \vdots & \vdots & \ddots & \vdots \\ q_{N,n} & q_{N,n-1} & \cdots & q_{N,0} \end{pmatrix}$$

  where $p(s) = p_n(s^\alpha)^n + p_{n-1}(s^\alpha)^{n-1} + \cdots + p_0$ and $q_k(s) = q_{k,n}(s^\alpha)^n + q_{k,n-1}(s^\alpha)^{n-1} + \cdots + q_{k,0}$ for $k \in \{1, \ldots, N\}$. Thanks to the argument `delay_vect` below, rows which are identically zero can be omitted, and so `poly_mat` can have less than $N+1$ rows.

- `delay_vect`: the delay vector, which is a vector of values of k for which q_k is not null. Its length should be equal to the number of rows of `poly_mat` minus one, and the delay corresponding to the row `poly_mat[i+1, :]` is `delay_vect[i] * tau`.
- `alpha`: a number in the interval $(0, 1]$ describing the fractional power α in (1).
- `tau`: the value of the nominal delay τ.

3.3 Output

An `OrderedDict` containing the following fields:

- Type
- AsympStability
- RootsNoDelay
- RootsChain
- CrossingTable
- ImaginaryRoots

3.3.1 Type

This output is a string giving the type of the system [2]: retarded, neutral, or advanced (in case the user was wrong in defining the system).

3.3.2 AsympStability

The `AsympStability` is a string that describes one of the following situations:

- Infinite number of unstable poles.
- There is no unstable pole.
- There are d unstable poles, $1 \leq d < \infty$.
- Impossible to conclude, if YALTAPy cannot determine if a chain of poles clustering the imaginary axis is to the left or to the right of the axis (see [3]).

3.3.3 RootsNoDelay

An array of roots of the system with no delay, i.e., the roots of $p(s) + \sum_{k=0}^{N} q(s)$.

3.3.4 RootsChain

For neutral systems: An array of abscissa of the asymptotes of the root chains.

For retarded systems: a string stating that roots chains only computed for neutral systems.

3.3.5 CrossingTable

This array describes the points of crossing of the imaginary axis. The first column gives the first value of the delay for which a specific zero crosses the axis. The third column gives the frequency ω of crossing of a zero. The second column is $\frac{2\pi}{\omega}$. The fourth column gives the crossing direction: -1 is from right to left and 1 from left to right.

3.3.6 ImaginaryRoots

The set of all the imaginary roots for a delay between zero and the delay τ given in input. The first column is the delay value, the second column the position on the imaginary axis and the third column gives the direction as in the CrossingTable output.

4 thread_stability_windows

Launches the analysis of a SISO delay system given by its transfer function G as in (1). The delay system must be of retarded type or neutral type with a finite number of poles in $\{\mathrm{Re}(s) > -a\}$ for some $a > 0$. This function can be used if one only wants to have a graph of the stability windows for a bounded set of values of the delay.

4.1 Syntax

```
res = thread_stability_windows(poly_mat,delay_vect,alpha,tau,tmaxsw,tminsw,\
                               plot)
res = thread_stability_windows(poly_mat,delay_vect,alpha,tau,tmaxsw,tminsw)
res = thread_stability_windows(poly_mat,delay_vect,alpha,tau,tmaxsw)
res = thread_stability_windows(poly_mat,delay_vect,alpha,tau)
```

4.2 Inputs

The inputs `poly_mat`, `delay_vect`, `alpha`, and `tau` have the same meaning as those for the function `thread_analysis`, please refer to Section 3.2 for more details. The other inputs are:

- `tmaxsw`: the maximum delay for the stability window (default: τ).
- `tminsw`: the minimum delay for the stability window (default: 0).
- `plot`: a boolean variable determining if a graph is traced (`True`) or not (`False`) (default: `True`).

4.3 Output

An `OrderedDict` containing the following fields:

- Type
- RootsNoDelay

- RootsChain
- CrossingTable
- StabilityWindows
- NbUnstablePoles

The fields Type, RootsNoDelay, RootsChain, and CrossingTable have the same meaning as those from the function `thread_analysis`, please refer to Sections 3.3.1, 3.3.3, 3.3.4, 3.3.5 for more details on those outputs. The other fields are described below.

4.3.1 StabilityWindows

An array whose first row gives the values of delay where the stability changes, i.e., from stability to instability or vice-versa, including the minimum and maximum delays. Its second row indicates the stability of delay intervals, with 1 representing stability and 0 representing instability. The i-th entry of the second row corresponds to the delay interval from i-th to $(i + 1)$-th entries of the first row.

4.3.2 NbUnstablePoles

This array is similar to StabilityWindows, except that the first row contains the delays where the number of unstable poles change and the second row gives the number of unstable poles in delay intervals, following the same convention as StabilityWindows (i.e., the i-th entry of the second row corresponds to the delay interval from i-th to $(i + 1)$-th entries of the first row).

4.4 Display

If `plot` is true, displays two graphs, one for the stability windows and another for the numbers of unstable poles. See Figure 4.1 for an example of such graphs.

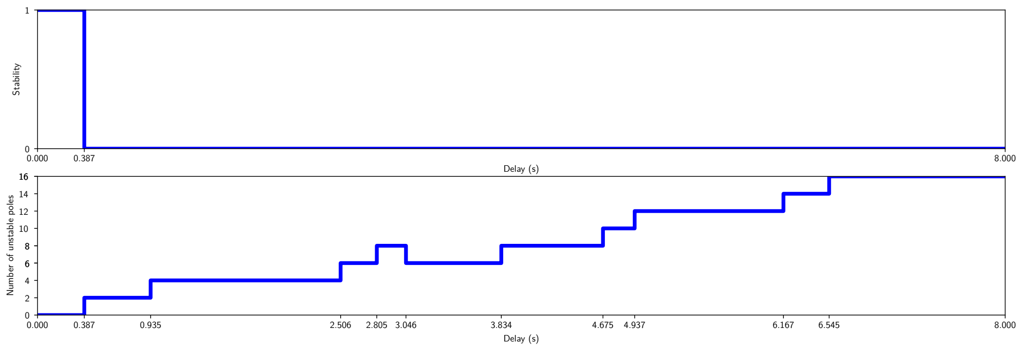

Figure 4.1: Example of graph displayed by the function `thread_stability_windows`

5 thread_root_locus

The function computes the root locus of a SISO delayed system given by its transfer function G as in (1). The delay system must be of retarded type or neutral type with a finite number of poles in $\{\mathrm{Re}(s) > -a\}$ for some $a > 0$.

5.1 Syntax

```
res = thread_root_locus(poly_mat, delay_vect, alpha, tau, plot, delta_tau)
res = thread_root_locus(poly_mat, delay_vect, alpha, tau, plot)
res = thread_root_locus(poly_mat, delay_vect, alpha, tau)
```

5.2 Inputs

The inputs `poly_mat`, `delay_vect`, `alpha`, and `tau` have the same meaning as those for the function `thread_analysis`, please refer to Section 3.2 for more details. The other inputs are:

- `plot`: a boolean variable determining if a graph is traced (`True`) or not (`False`) (default: `True`).
- `delta_tau`: precision of integration procedure for computing root locus (default: 10^{-4}).

5.3 Output

An `OrderedDict` containing the following fields:

- Type
- RootsNoDelay
- CrossingTable
- ImaginaryRoots
- UnstablePoles
- PolesError
- RootLocus

The fields Type, RootsNoDelay, CrossingTable, and Imaginary Roots have the same meaning as those from the function `thread_analysis`, please refer to Sections 3.3.1, 3.3.3, 3.3.5, 3.3.6 for more details on those outputs. The other fields are described below.

5.3.1 UnstablePoles

An array containing all the unstable poles at the delay value given in input (nominal delay).

5.3.2 PolesError

An array containing the estimated error of evaluation of each unstable pole.

5.3.3 RootLocus

This list of arrays contains branches of the displayed root locus. Those branches correspond to roots starting from unstable poles of delay free system as well as roots crossing the imaginary axis from left to right. Each array of the list represents a branch, as has three rows containing, respectively, the real and imaginary parts of the roots and the corresponding values of the delay.

5.4 Display

If `plot` is true, displays the graph of the root locus. See Figure 5.1 for an example of such a graph.

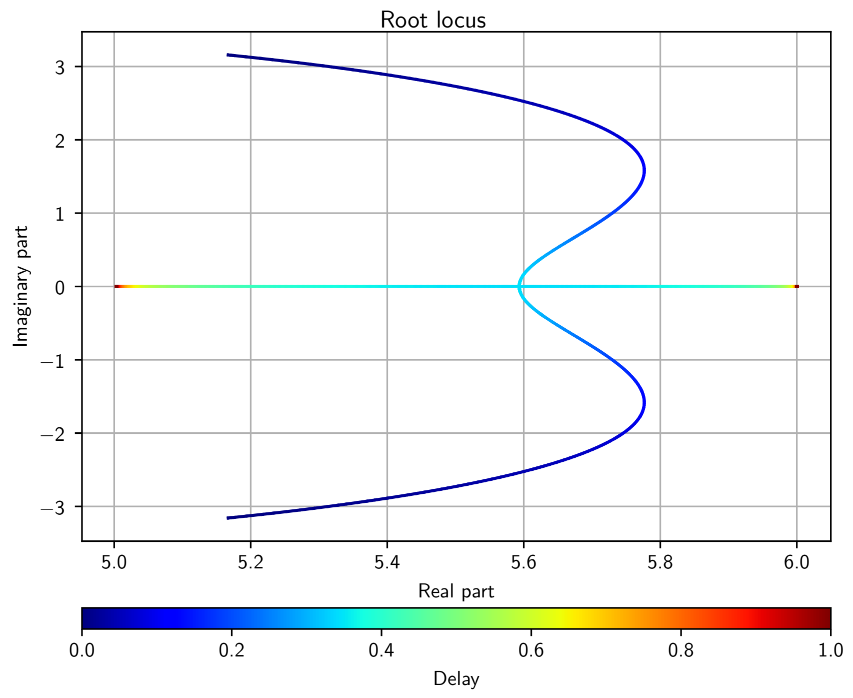

Figure 5.1: Example of graph displayed by the function `thread_root_locus`

6 compute_pade

Computes the Padé-2 approximation (see [7, 8]) of a *non-fractional* SISO delay system with transfer function G of the form

$$C(s) = \frac{p(s) + \sum_{k=1}^{N} q_k(s)e^{-k\tau s}}{(s+1)^{\delta}}, \tag{4}$$

where $p, q_1, \dots, q_N$ are *polynomials* in s, the degree of p is n, and $\delta \geq n+1$.

There are two modes for computing the Padé approximation: "ORDER" and "NORM", according to whether, respectively, the order of the approximation is fixed, or a maximum value of the H_∞ norm between $C(s)$ and its Padé approximation is provided.

Note: To get an approximation of $G(s)$ defined as in (1) and having a finite number of unstable poles, write $G(s) = \frac{\frac{n(s)}{(s+1)^{n+1}}}{\frac{d(s)}{(s+1)^{n+1}}} = \frac{N(s)}{D(s)}$. If N and D do not have common unstable zeros (this can be verified with `thread_root_locus`) this factorization (N, D) is indeed a coprime factorization of G. Let $N_k(s)$ and $D_k(s)$ be respectively Padé-2 approximations of $N(s)$ and $D(s)$. Then an approximation of $G(s)$ is given by $G_k(s) = \frac{N_k(s)}{D_k(s)}$.

6.1 Syntax

```
res = compute_pade(poly_mat, delta, tau, delay_vect, mod_arg, mode)
```

6.2 Inputs

The inputs `poly_mat`, `delay_vect`, and `tau` have the same meaning as those for the function `thread_analysis`, please refer to Section 3.2 for more details. The other inputs are:

- `delta`: the value of δ in (4), should be at least $n + 1$.
- `mode`: one of the strings "ORDER" or "NORM", defining the mode of computation (default: "ORDER").
- `mod_arg`: if `mode` is "ORDER", `mode_arg` should be the order of the approximation. If `mode` is "NORM", `mode_arg` should be the maximum desired value of the H_∞ norm between $C(s)$ and its Padé approximation.

6.3 Output

An object containing the following attributes:

- `num_approx`: vector of coefficients of the numerator of the transfer function of the approximation.
- `den_approx`: vector of coefficients of the denominator of the transfer function of the approximation.
- `error_norm`: the H_∞ norm of the difference between C and its approximation.
- `pade_order`: the order of the approximation.
- `roots`: the roots of the approximation.
- `roots_error`: an array of differences between the computed roots of C (by `thread_root_locus`) and the roots of the approximation.

7 Example

We consider in example that the following imports have been done in Python:

```
import numpy as np
import yaltapy as yp
```

Consider (1) with denominator $d(s)$ given by

$$\begin{aligned} d(s) = {} & s^3 + 3s^2 + 2s - 1 \\ & + e^{-\tau s}\left(3s^2 - 4s + 2\right) \\ & + e^{-2\tau s}\left(0.5s^3 + 0.7s^2 + 1.2s - 0.8\right) \\ & + e^{-3\tau s}\left(0.5s^3 + 1.34s^2 - 0.7s + 1.9\right) \\ & + e^{-6\tau s}\left(3.4s - 1.6\right), \end{aligned} \tag{5}$$

with a nominal value $\tau = 2$. This quasi-polynomial can be represented in YALTAPy through the inputs

```
poly_mat = np.array([[1, 3, 2, -1],
                     [0, 3, -4, 2],
                     [0.5, 0.7, 1.2, -0.8],
                     [0.5, 1.34, -0.7, 1.9],
                     [0, 0, 3.4, -1.6]])
delay_vect = np.array([1, 2, 3, 6])
alpha = 1
tau = 2
```

7.1 thread_analysis

Let us run `thread_analysis` through the command

```
res = yp.thread_analysis(poly_mat, delay_vect, alpha, tau)
```

Printing the outputs of `thread_analysis`, one gets the following:

```
>>> print(res["Type"])
Neutral
>>> print(res["AsympStability"])
There is (are) 4 unstable pole(s) in right half-plane
>>> print(res["RootsNoDelay"])
[-3.78654811+0.j         -0.11672594+0.22890664j -0.11672594-0.22890664j]
>>> print(res["RootsChain"])
[-0.04127457 -0.26402445]
>>> print(res["CrossingTable"])
[[ 0.38675487 10.21442933  0.61512837  1.        ]
 [ 0.9351039   1.86983146  3.36029499  1.        ]
 [ 2.50570977  3.66105881  1.71622081  1.        ]
 [ 3.04607105  5.08619117  1.23534195 -1.        ]
 [ 3.83421262 16.11501189  0.38989641  1.        ]
 [ 4.93674455  5.74886883  1.09294289  1.        ]
 [29.11158931 29.50880359  0.21292579 -1.        ]]
>>> print(res["ImaginaryRoots"])
[[ 0.38675487  0.61512837  1.        ]
 [ 0.38675487 -0.61512837  1.        ]
 [ 0.9351039   3.36029499  1.        ]
 [ 0.9351039  -3.36029499  1.        ]]
```

Hence (5) represents the denominator of a system of neutral type, with 4 unstable poles in the right half-plane. If $\tau = 0$, (5) reduces to a polynomial of degree 3, whose 3 roots were computed by YALTAPy and, in particular, have all negative real part. For the nominal value $\tau = 2$, YALTAPy has identified two root chains, whose real parts are asymptotic to -0.04127457 and -0.26402445.

As for the crossing table, its first row says that, for a delay $\tau = 0.38675487$, a root crosses from the left to the right (the fourth column is 1) at $\pm j\omega = \pm 0.61512837j$, and $\frac{2\pi}{\omega} = 10.21442933$. The other rows provide the other crossings.

The first two rows of the array ImaginaryRoots say that, for a delay $\tau = 0.38675487$, two imaginary roots are present, at the values $\pm j0.61512837$ and, as the delay increases, they cross from the left to the right (value 1 in the third column). Note that ImaginaryRoots goes from 0 until the nominal delay $\tau = 2$.

7.2 thread_stability_windows

Let us now run `thread_stability_windows` through the command

```
res = yp.thread_stability_windows(poly_mat, delay_vect, alpha, tau)
```

Note that this uses the default values for `tmaxsw`, `tminsw`, and `plot`. After running this command, we get the plot represented in Figure 7.1.

Figure 7.1 shows that the system is stable for $\tau < 0.387$, and becomes unstable for $\tau > 0.387$. There are no unstable poles for $\tau < 0.387$, a first pair of unstable poles appear at $\tau = 0.387$, and another pair appears at $\tau = 0.935$. Note that these conclusions could have been also obtained using the non-graphic outputs of the function `thread_analysis` shown in Section 7.1.

Printing the outputs of `thread_stability_windows`, one gets the following:

```
>>> print(res["Type"])
Neutral
>>> print(res["RootsNoDelay"])
[-3.78654811+0.j         -0.11672594+0.22890664j -0.11672594-0.22890664j]
>>> print(res["RootsChain"])
[-0.04127457 -0.26402445]
>>> print(res["CrossingTable"])
[[ 0.38675487 10.21442933  0.61512837  1.        ]
 [ 0.9351039   1.86983146  3.36029499  1.        ]
 [ 2.50570977  3.66105881  1.71622081  1.        ]
 [ 3.04607105  5.08619117  1.23534195 -1.        ]
```

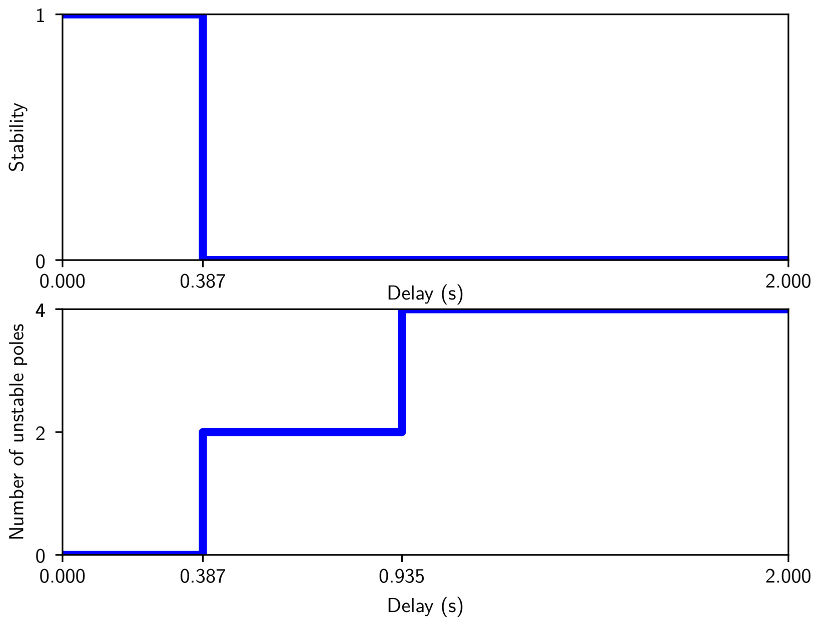

Figure 7.1: Graph displayed by the function `thread_stability_windows` when running the example of Section 7

```
 [ 3.83421262 16.11501189  0.38989641  1.        ]
 [ 4.93674455  5.74886883  1.09294289  1.        ]
 [29.11158931 29.50880359  0.21292579 -1.        ]]
>>> print(res["StabilityWindows"])
[[0.         0.38675487 2.        ]
 [1.         0.         0.        ]]
>>> print(res["NbUnstablePoles"])
[[0.         0.38675487 0.9351039  2.        ]
 [0.         2.         4.         4.        ]]
```

The first four outputs were already present in the outputs of `thread_analysis`, and the last two provide the information used for the plots in Figure 7.1.

7.3 thread_root_locus

Let us now run `thread_root_locus` through the command

```
res = yp.thread_root_locus(poly_mat, delay_vect, alpha, tau)
```

Note that this uses the default values for `plot` and `delta_tau`. After running this command, we get the plot represented in Figure 7.2, which shows the movement of the four roots of (5) that cross into the right half-plane as τ increases from 0 to 2.

Printing the outputs of `thread_root_locus`, one gets the following:

```
>>> print(res["Type"])
Neutral
```

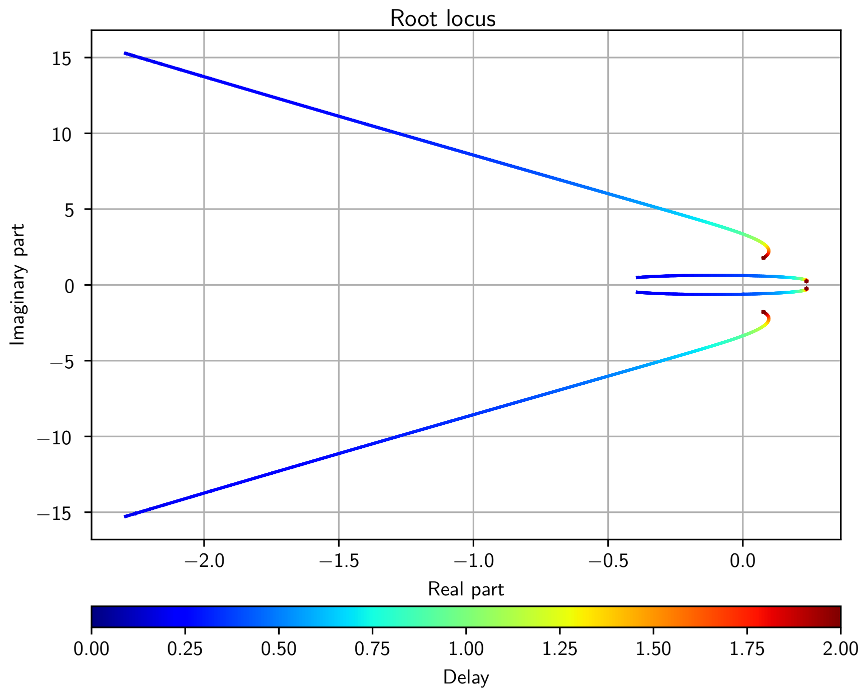

Figure 7.2: Graph displayed by the function `thread_root_locus` when running the example of Section 7

```
>>> print(res["RootsNoDelay"])
[-3.78654811+0.j         -0.11672594+0.22890664j -0.11672594-0.22890664j]
>>> print(res["CrossingTable"])
[[ 0.38675487 10.21442933  0.61512837  1.        ]
 [ 0.9351039   1.86983146  3.36029499  1.        ]
 [ 2.50570977  3.66105881  1.71622081  1.        ]
 [ 3.04607105  5.08619117  1.23534195 -1.        ]
 [ 3.83421262 16.11501189  0.38989641  1.        ]
 [ 4.93674455  5.74886883  1.09294289  1.        ]
 [29.11158931 29.50880359  0.21292579 -1.        ]]
>>> print(res["ImaginaryRoots"])
[[ 0.38675487  0.61512837  1.        ]
 [ 0.38675487 -0.61512837  1.        ]
 [ 0.9351039   3.36029499  1.        ]
 [ 0.9351039  -3.36029499  1.        ]]
>>> print(res["UnstablePoles"])
[0.07619789-1.77970107j 0.07619789+1.77970107j 0.23604754+0.2325349j
 0.23604754-0.2325349j ]
>>> print(res["PolesError"])
[1.e-11+0.j 1.e-12+0.j 1.e-11+0.j 1.e-11+0.j]
>>> print(res["RootLocus"])
```

The first four outputs were already present in the outputs of `thread_analysis`. UnstablePoles give the values of the four unstable poles at the nominal value $\tau = 2$, and PolesError provides the estimated error of this computation. The output RootLocus is not represented since it is too long.

7.4 compute_pade

Let us finally compute a Padé approximation of $d(s)$ from (5). Since $n = 3$, any Padé approximation should be of order 4 or more. We can compute the Padé approximation by:

```
res = yp.compute_pade(poly_mat, 4, tau, delay_vect, 4, "ORDER")
```

The corresponding results can be shown as follows:

```
>>> print(res.num_approx)
[2.00000000e+00 1.92040000e+02 9.15123333e+03 2.96964980e+05
 7.32903870e+06 1.44984579e+08 2.37207030e+09 3.27808542e+10
 3.88478826e+11 3.99343801e+12 3.59375116e+13 2.85303832e+14
 2.01133000e+15 1.26630760e+16 7.15476853e+16 3.64307073e+17
 1.67764881e+18 7.00820054e+18 2.66250502e+19 9.21895467e+19
 2.91442571e+20 8.42436140e+20 2.22913849e+21 5.40422384e+21
 1.20111492e+22 2.44808428e+22 4.57579845e+22 7.84116785e+22
 1.23112476e+23 1.76936357e+23 2.32464527e+23 2.78733178e+23
 3.04375031e+23 3.01947675e+23 2.71316157e+23 2.20064462e+23
 1.60487122e+23 1.04763783e+23 6.09200819e+22 3.14057500e+22
 1.43024001e+22 5.75557863e+21 2.06528567e+21 6.76492585e+20
 2.09371143e+20 6.23481927e+19 1.72712265e+19 4.09590121e+18
 7.49240883e+17 9.32803382e+16 6.44577697e+15 1.40737488e+14]
>>> print(res.den_approx)
[1.00000000e+00 1.16000000e+02 6.58466667e+03 2.43529333e+05
 6.59350278e+06 1.39239870e+08 2.38655292e+09 3.41158404e+10
 4.14843449e+11 4.35539933e+12 3.99422730e+13 3.22943875e+14
 2.31946517e+15 1.48908781e+16 8.58979764e+16 4.47175100e+17
 2.10870426e+18 9.03581478e+18 3.52776119e+19 1.25777567e+20
 4.10316487e+20 1.22673632e+21 3.36576102e+21 8.48374261e+21
 1.96622449e+22 4.19268258e+22 8.22891178e+22 1.48686371e+23
 2.47328439e+23 3.78664683e+23 5.33362280e+23 6.90694756e+23
 8.21580848e+23 8.96611602e+23 8.96421661e+23 8.19593068e+23
 6.83797741e+23 5.19258874e+23 3.57798233e+23 2.22900045e+23
 1.25001984e+23 6.27776182e+22 2.80580459e+22 1.10756022e+22
 3.82520017e+21 1.14234003e+21 2.90544670e+20 6.16882548e+19
 1.06357971e+19 1.43026818e+18 1.40714032e+17 9.00719925e+15
 2.81474977e+14]
>>> print(res.error_norm)
0.27800678706844917
>>> print(res.pade_order)
4
>>> print(res.roots)
[0.23607262+0.23247355j 0.23607262-0.23247355j]
>>> print(res.roots_error)
[6.62831957e-05 6.62846832e-05]
```

References

[1] D. Avanessoff, A. Fioravanti, and C. Bonnet. Yalta: a matlab toolbox for the h_∞-stability analysis of classical and fractional systems with commensurate delays. In *IFAC Joint Conference, 11th Workshop on Time-Delay Systems*, February 2013.

[2] R. Bellman and K. L. Cooke. *Differential-Difference Equations*. Academic Press, New York, London, 1963.

[3] C. Bonnet, A. R. Fioravanti, and J. Partington. Stability of neutral systems with commensurate delays and poles asymptotic to the imaginary axis. *SIAM Journal on Control and Optimization*, 49:498–516, 2011.

[4] A. Fioravanti, C. Bonnet, and H. Ozbay. Stability of fractional neutral systems with multiple delays and poles asymptotic to the imaginary axis. In *IEEE Conference on Decision and Control*, Atlanta, USA, December 2010.

[5] A. Fioravanti, C. Bonnet, H. Ozbay, and S.-I. Niculescu. Stability windows and unstable poles for linear time-delay systems. In *9th IFAC Workshop on Time-Delay Systems*, Prague, June 2010.

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

[7] P. M. Mäkilä and J. R. Partington. Laguerre and Kautz shift approximations of delay systems. *Internat. J. Control*, 72(10):932–946, 1999.

[8] J. Partington. Approximation of unstable infinite-dimensional systems using coprime factors. *Systems and Control Letters*, 16:89–96, 1991.