INTRODUCTION TO "RORPack" FOR MATLAB

A MATLAB SOFTWARE PACKAGE FOR ROBUST OUTPUT REGULATION

LASSI PAUNONEN

Wednesday 30th June, 2021

Abstract. This document contains the mathematical introduction to RORPack — a software package for robust output tracking and disturbance rejection for linear PDE systems. The RORPack library is open-source and freely available at https://github.com/lassipau/rorpack/ and https://github.com/lassipau/rorpack-matlab/ The package contains functionality for automated construction of robust internal model based controllers, simulation of the controlled systems, visualisation of the results, as well as a collection of example cases on robust output regulation of controlled heat and wave equations.

Contents

Download links

Both the Python and Matlab versions of **RORPack** software are distributed as open source under the GNU General Public License version 3 (see `LICENSE.txt` for detailed license and copyright information) and can freely downloaded at the addresses

https://github.com/lassipau/rorpack/ (Python version)
https://github.com/lassipau/rorpack-matlab/ (Matlab version)

The project version information can be found in the `CHANGELOG.md` files, and installation instructions are included in the `README.md` files.

1. General Description

RORPack is a software package for controller design and simulation of robust output tracking and disturbance rejection for linear partial differential equation models. The package is available as a Python library and a Matlab package. The software contains a number of complete examples on robust controller design and simulation of PDE models of the following types:

- one-dimensional diffusion equations with either boundary or distributed control and observation.
- two-dimensional heat equation on a rectangular domain.
- one-dimensional wave equations with either distributed or boundary control and observation.

New examples will also be added in the future versions of the package.

The purpose of this document is to give a general introduction to the background theory of robust output regulation for linear PDE systems, to describe the mathematical models that are included as the example cases, and to document the usage of the software package on a general level.

The purpose of **RORPack** is to serve as a tool to *illustrate* the theory of robust output regulation for distributed parameter systems and it should not (yet) be considered as a serious controller design software. The developers of the software do not take any responsibility and are not liable for any damage caused through use of this software. In addition, at its present stage, the library is not optimized in terms of numerical accuracy. Any helpful comments on potential improvements of the numerical aspects will be greatly appreciated!

This documentation is also published in arXiv.org. The website of the project is located at the address (hosted by GitHub Pages)

https://lassipau.github.io/rorpack/

All comments and suggestions for improvements are welcome! These can be sent directly to `lassi.paunonen@tuni.fi`.

1.1. Controller Design and Simulation Workflow.

Basic workflow for robust controller design and simulation for a given system is that the user creates a main file with the following parts:

(1) Calling of a user-defined Matlab function that returns a numerical approximation of the linear PDE model. In the provided examples this function is in a separate file with the prefix "Constr".
(2) Defining the reference and disturbance signals to be considered.
(3) Calling of a **RORPack** routine for construction of a robust controller that is suitable for the type of PDE model. This typically involves choosing appropriate parameters for the controller construction.
(4) Calling of **RORPack** routines for construction and simulation of the closed-loop system. The routines return numerical data describing the output, the error, the control signal, and the closed-loop state.
(5) Calling of **RORPack** routines for visualising the behaviour of the output and the output error, as well as user-defined routines for illustrating the behaviour of the state of the controlled PDE system (multidimensional plots or animations).

The example cases included in **RORPack** are built around main files that follow the same structure, and the users are encouraged to implement their own simulations by modifying the example codes.

It is also possible to combine other Python software packages and Matlab toolboxes in the study of robust controller design for PDE models in order to employ ready-made numerical approximations or additional numerical methods in the computation of required controller parameters. This approach is illustrated in one of the PDE example cases "`BeamKelvinVoigt_main`" described in Section 5.4 (Case Euler-Bernolli Beam with Kelvin-Voigt Damping). In this example the system is approximated using the Galerkin method based on a nonlocal basis based on Chebyshev polynomials with the **Chebfun package** [**?**] (https://chebfun.org/) which provides flexible tools for solutions of differential equations with extreme accuracy using spectral methods.

2. Introduction to Robust Output Regulation

In this section we give a general introduction to the mathematical theory of *robust output regulation*. The purpose of the **RORPack** package is to illustrate controller design for linear distributed parameter systems of the form

$$\dot{x}(t) = Ax(t) + Bu(t) + B_d w_{dist}(t), \qquad x(0) = x_0 \in X \tag{2.1a}$$

$$y(t) = C_\Lambda x(t) + Du(t) + D_d w_{dist}(t) \tag{2.1b}$$

on a Banach or a Hilbert space X. Controlled linear PDE models describing diffusion-convection phenomena, waves and vibrations and elastic deformations can be written in this form with a suitable differential operator A [**?**, **?**, **?**]. In our main control problem the goal is to design a dynamic error feedback controller in such a way that the output $y(t)$ of the system converges to a given reference signal $y_{ref}(t)$ despite the external disturbance signal $w_{dist}(t)$. In addition, the controller needs to be *robust* in the sense that it achieves the output tracking and disturbance rejection even if the parameters (A, B, B_d, C, D, D_d) are perturbed or contain small uncertainties.

Our main emphasis is on robust output regulation of *diffusion-convection equations*, *wave equations*, and *beam and plate equations*. However, the **RORPack** package can also be used for construction of controllers for finite-dimensional systems with given matrices (A, B, B_d, C, D, D_d).

The full robust output regulation problem is defined in the following way.

The Robust Output Regulation Problem. *Given a reference signal $y_{ref}(t)$, design a dynamic error feedback controller such that the output $y(t)$ of the system converges to the reference signal asymptotically, i.e.,*

$$\lim_{t\to\infty} \|y(t) - y_{ref}(t)\|_Y = 0 \tag{2.2}$$

despite the disturbance signal $w_{dist}(t)$. Moreover, the controller is required to be robust *in the sense that it achieves the convergence of the output (2.2) even under small uncertainties and changes in the parameters (A, B, B_d, C, D, D_d) of the system.*

The reference signal $y_{ref}(t)$ and the disturbance signals $w_{dist}(t)$ are assumed to be of the form

$$y_{ref}(t) = a_0^1 + \sum_{k=1}^{q} (a_k^1 \cos(\omega_k t) + b_k^1 \sin(\omega_k t)) \tag{2.3a}$$

$$w_{dist}(t) = a_0^2 + \sum_{k=1}^{q} (a_k^2 \cos(\omega_k t) + b_k^2 \sin(\omega_k t)) \tag{2.3b}$$

for some **known frequencies** $\{\omega_k\}_{k=0}^q \subset \mathbb{R}$ with $0 = \omega_0 < \omega_1 < \ldots < \omega_q$ and unknown amplitudes $\{a_k^j\}_{k,j}, \{b_k^j\}_{k,j} \subset \mathbb{R}$ (some of which may zero).

Dynamic feedback is essential for achieving robust output regulation, and the control problem can indeed be solved with a dynamic error feedback controller (see Figure 1). The classical *internal model principle* gives a characterization for the controllers that solve the robust output regulation problem. This fundamental result was first introduced for finite-dimensional linear systems in the 1970's by Francis and Wonham [**?**] and Davison [**?**] (see [**?**, Ch. 1] for an excellent overview). The internal model principle was later extended for infinite-dimensional linear systems by the Systems Theory Research Group at Tampere, Finland in the references [**?**, **?**, **?**, **?**, **?**][1].

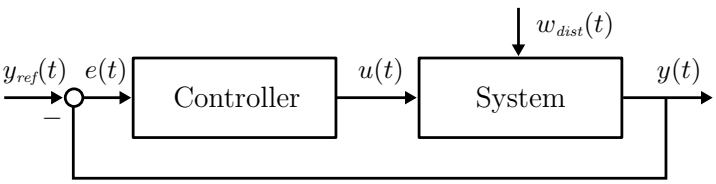

FIGURE 1. Dynamic error feedback control scheme.

The internal model principle [**?**, Thm. 6.9] is also the most important tool in designing controllers for robust output regulation. The result states that a controller solves the robust output regulation problem if and only if the following conditions are satisfied.

- The error feedback controller incorporates "an internal model" of the signals $y_{ref}(t)$ and $w_{dist}(t)$ in (2.3).
- The closed-loop system is exponentially or strongly stable.

In controller design, the internal model property can be guaranteed by choosing a suitable *structure* for the controller. The rest of parameters are subsequently chosen so that the closed-loop system becomes stable.

The detailed description of the theory of robust output regulation problem and the internal model principle can be found in the references listed below. The main emphasis in the list is (shamelessly) on the publications by the Systems Theory Research Group at Tampere University, Finland.

- [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**] (including PI-control for PDE models): Robust controller design in various forms for infinite-dimensional linear systems.

[1]Our focus is on linear systems, but there is also an extensive literature on the internal model principle for nonlinear systems, see, e.g., [**?**, **?**, **?**] and references therein.

- [?] and [?]: The Internal Model Principle for infinite-dimensional linear systems with bounded and unbounded, respectively, input and output operators B and C.
- [?, ?]: Robust controller design for *regular linear systems*.
- [?, ?]: Robust controller design for *port-Hamiltonian systems* and other boundary controlled partial differential equations.
- [?, ?, ?, ?]: Robust controller design *impedance passive systems*.
- [?, ?, ?, ?] for robust finite-dimensional low-order controller design for parabolic systems using Galerkin approximations and model reduction.
- References on the output regulation *without* the robustness requirement: [?, ?, ?, ?, ?, ?, ?, ?, ?]

The considered controllers are of the form

$$\dot{z}(t) = \mathcal{G}_1 z(t) + \mathcal{G}_2(y(t) - y_{ref}(t)), \qquad z(0) = z_0 \in Z \tag{2.4a}$$

$$u(t) = Kz(t) + D_c(y(t) - y_{ref}(t)). \tag{2.4b}$$

Here $y(t) - y_{ref}(t)$ is the *regulation error*. The construction of the robust controllers are based on the references [?, ?] with certain modifications and improvements. In particular, the internal models are defined in their "real forms", making the controller real whenever the parameters of the plant are real. The same controllers also achieve robust output tracking also for reference and disturbance signals with complex coefficients $\{a_k^j\}_{k,j}, \{b_k^j\}_{k,j} \subset \mathbb{C}$.

The constructions of the robust controllers use the knowledge of the frequencies $\{\omega_k\}_k$ of the reference and disturbance signals, the number of outputs $p := \dim Y$ of the system (2.1), and certain knowledge of the system. In particular, the "minimal controllers" require knowledge of the values $P(i\omega_k)$ of the transfer function $P(\lambda) = C_\Lambda R(\lambda, A)B + D$ of the system at the complex frequencies $\{i\omega_k\}_k \subset i\mathbb{R}$ of the reference and disturbance signals (2.3). The other controller structures also use knowledge of the parameters (A, B, C, D) of the system as they involve designing an observer for (2.1), and require the user to provide stabilizing state feedback and output injection operators. See Section 4 for more information on the controller-specific requirements.

3. Using the Software

TODO: UPDATE this section to Matlab Version

The `rorpack` Matlab library can be installed for Matlab R2020a and later (earlier versions may be fine too) as instructed in the `README.md` file included in the software package (this mainly involves copying the package on your hard drive and adding the "RORPack" folder and its subfolders to the Matlab PATH). The subdirectory `examples/` contains the included PDE examples and simulation files (documented in detail in Section 5).

The following commented example file explains the typical structure and workflow of the controller construction and simulation with **RORPack**. The considered example cases included in the files `Heat1DtestCase3.m` and `Constr1DHeatCase3.m` and documented in Section 5.1. Due to the properties of Matlab, the constructor routine used in the main simulation is in its own, separate, function file.

Contents of the files `Heat1DtestCase3.m` and `Constr1DHeatCase3.m`: The file `Heat1DtestCase3.m` begins with comment lines and adding the "RORPack" folder to the top of the search path for the current Matlab session.

```
%% Heat equation on the interval [0,1] with
% Neumann boundary control and Dirichlet boundary observation
% Approximation with a Finite differences scheme

% Neumann boundary disturbance at x=0, two distributed ...
    controls and
% two distributed measurement regulated outputs. The controls act
% on the intervals 'IB1' and 'IB2' (Default 'IB1' = [0.3,0.4] and
% 'IB2' = [0.6,0.7]) and the measurements are the average
% temperatures on the intervals 'IC1'  and 'IC2' (Default
% 'IC1' = [0.1,0.2] and 'IC2' = [0.8,0.9]).

addpath(genpath('../RORPack/'))
```

The next part defines the parameters N, `cfun` and `x0fun` used in the construction of the system (A, B, B_d, C, D) with the function `Constr1DHeatCase3`, which is introduced next. The parameter N is the size of the approximation and `cfun` is a function describing the spatially varying thermal diffusivity of the material. $x0fun$ describes the initial heat profile of the plant.

```
% Parameters for this example.
N = 100;

% Initial state of the plant
%x0fun = @(x) zeros(size(x));
x0fun = @(x) 1*(1+cos(pi*(1-x)));
%x0fun = @(x) 3*(1-x)+x;
%x0fun = @(x) 0.5*(1+cos(pi*(1-x)));
%x0fun = @(x) 1/2*x.^2.*(3-2*x)-1;
%x0fun = @(x) 1/2*x.^2.*(3-2*x)-1/2;
%x0fun = @(x) 1*(1-x).^2.*(3-2*(1-x))-1;
%x0fun = @(x) .5*(1-x).^2.*(3-2*(1-x))-.5;
%x0fun = @(x) 1/4*(x.^3-1.5*x.^2)-1/4;
%x0fun = @(x) .2*x.^2.*(3-2*x)-.5;

% The spatially varying thermal diffusivity of the material
% cfun = @(t) ones(size(t));
% cfun = @(t) 1+t;
% cfun = @(t) 1-2*t.*(1-2*t);
% cfun = @(t) 1+0.5*cos(5/2*pi*t);
cfun = @(t) 0.3-0.6*t.*(1-t);

IB1 = [.3, .4];
IB2 = [.6, .7];
IC1 = [.1, .2];
IC2 = [.8, .9];

[x0,Sys,spgrid,BCtype] = ...
    Constr1DHeatCase3(cfun,x0fun,N,IB1,IB2,IC1,IC2);
```

The function `Constr1DHeatCase3` used to approximate the system begins with comments describing this particular approximation along with argument checking since the inputs `IB1`, `IB2`, `IC1` and `IC2` are not explicitly required to construct the approximation.

```
function [x0,Sys,spgrid,BCtype] = Constr1DHeatCase3(...
    cfun,x0fun,N,IB1,IB2,IC1,IC2)
% [x0,Sys,spgrid] = Constr1DHeatCase3(x0fun,N)
%
% Finite Differences approximation of a 1D Heat equation
% with a Neumann boundary condition at x=0 and a Dirichlet ...
    condition at x=1
% Usage: Can also use solely for defining the initial state x0
% cfun = spatially variying thermal diffusivity of the material
% x0fun = initial heat profile, function handle
% Sys = system parameters, (sys.A,sys.B,sys.Bd,sys.C,sys.D)
% N = dimension of the approximated system (does not include ...
    point x=1)
%
% Case 3: Neumann boundary disturbance at x=0, 2 distributed ...
    controls and
% two distributed measurements regulated output y(t). The ...
    controls affect
% the intervals 'IB1' = [a_1,b_1] and 'IB2' = [a_2,b_2], and the
% measurements are the averages of the temperatures on the ...
    intervals
% 'IC1' = [c_1,d_1] and 'IC2' = [c_2,d_2]. If these parameters ...
    are not
% given, then default configuration   IB1 = [.3, .4], IB2 = ...
    [.6, .7],
% IC1 = [.1, .2], and IC2 = [.8, .9] is used.

if nargin ≤ 3
  IB1 = [.3, .4];
  IB2 = [.6, .7];
  IC1 = [.1, .2];
  IC2 = [.8, .9];
end

% Check that the intervals are defined correctly
if ¬isempty(find([IB1(:);IB2(:);IC1(:);IC2(:)]<0))...
        || ¬isempty(find([IB1(:);IB2(:);IC1(:);IC2(:)]>1))
  error(...
      'All limits of the intervals IB1, IB2, IC1, and IC2 need ...
         to be on [0,1].')
elseif IB1(2)≤IB1(1) || IB2(2)≤IB2(1) ||...
       IC1(2)≤IC1(1) || IC2(2)≤IC2(1)
  error(...
      'All intervals IB1, IB2, IC1, and IC2 need to be of ...
         positive lengths.')
end
```

The rest of the function approximates the system with the appropriate boundary conditions.

```
% The point x=1 has a Dirichlet boundary condition, and is not ...
    chosen as a
% variable
spgrid = linspace(0,1,N+1);
h = 1/N;
% Case 3 has a Neumann boundary condition at x=0
% and a Dirichlet condition at x=1
BCtype = 'ND';

[A,spgrid] = DiffOp1d(cfun,spgrid,BCtype);

% Two distributed inputs on the intervals IB1 and IB2
B1 = 1/(IB1(2)-IB1(1))*(spgrid(1:N)≥IB1(1) & spgrid(1:N)≤IB1(2));
B1 = B1(:);
B2 = 1/(IB2(2)-IB2(1))*(spgrid(1:N)≥IB2(1) & spgrid(1:N)≤IB2(2));
B2 = B2(:);

% Neumann boundary disturbance at x=0, sign is based on the
% _outwards_ normal derivative
Bd = sparse([-2/h;zeros(N-1,1)]);

% Two distributed outputs on the intervals IC1 and IC2
C1 = h/(IC1(2)-IC1(1))*(spgrid(1:N)≥IC1(1) & spgrid(1:N)≤IC1(2));
C2 = h/(IC2(2)-IC2(1))*(spgrid(1:N)≥IC2(1) & spgrid(1:N)≤IC2(2));

x0 = x0fun(spgrid(1:N)).';

Sys.A = A;
Sys.B = [B1,B2];
Sys.Bd = Bd;
Sys.C = [C1;C2];
Sys.D = zeros(2,2);
Sys.Dd = zeros(2,1);
```

The next part of the main file defines the reference signal $y_{ref}(t)$ (in `yref`) and the disturbance signal $w_{dist}(t)$ (in `wdist`) and lists the (real) frequencies $\{\omega_k\}_{k=1}^q$ in the variable `freqsReal`. Alternative reference and disturbance signals are commented out in the code for further simulation experiments. *If disturbance inputs are not present in the simulation, $w_{dist}(t)$ should be defined as a zero signal with* `wdist = @(t) zeros(size(t))`. *Also* `Sys.Bd` *should be defined as a* $\mathrm{dimX} \times 1$ *zero matrix, and this can be done automatically using the function* `SysConsistent`.

```
% Define the reference and disturbance signals, and list the
% required frequencies in 'freqsReal'

%yref = @(t) sin(2*t)+.1*cos(6*t);
%yref = @(t) sin(2*t)+.2*cos(3*t);
%yref = @(t) ones(size(t));

%wdist = @(t) ones(size(t));
%wdist = @(t) sin(t);
%wdist = @(t) zeros(size(t));

% Case 1:
yref = @(t) [sin(2*t);2*cos(3*t)];
```

```
%wdist = @(t) zeros(size(t));
wdist = @(t) sin(6*t);

% Case 2:
% yref = @(t) ones(size(t));
% wdist = @(t) ones(size(t));

% Case 3:
% yref = @(t) sin(2*t)+.1*cos(6*t);
% wdist = @(t) sin(t);

freqsReal = [1 2 3 6];
```

The next part constructs the chosen controller structure with the corresponding function from the **RORPack** "controllers" folder. Alternative controller structures are commented out in the code for easy comparison of controller performances.

```
%% Construct the controller

% A Low-Gain 'Minimal' Robust Controller

% dimX = size(Sys.A,1);
% Pappr = @(s) Sys.C*((s*eye(dimX)-Sys.A)\Sys.B)+Sys.D;
%
% Pvals = cell(1,length(freqsReal));
% for ind = 1:length(freqsReal)
%   Pvals{ind} = Pappr(freqsReal(ind));
% end
%
% epsgainrange = [0.01,6];
% % epsgain = .1;
% [ContrSys,epsgain] = ...
    LowGainRC(freqsReal,Pvals,epsgainrange,Sys);
% epsgain

% An observer-based robust controller
% Stabilizing state feedback and output injection operators K ...
    and L
% These are chosen based on collocated design. Only the single ...
    unstable
% eigenvalue at s=0 needs to be stabilized
K = -Sys.B';
% K = zeros(2,N);
% PlotEigs(full(Sys.A+Sys.B*K),[NaN .1 -.3 .3])

%
L = -10*Sys.C';
%  PlotEigs(full(Sys.A+L*Sys.C),[-20 1 -.3 .3])

% ContrSys = ObserverBasedRC(freqsReal,Sys,K,L,'LQR',1);
% ContrSys = ObserverBasedRC(freqsReal,Sys,K,L,'poleplacement',1);
% ContrSys = DualObserverBasedRC(freqsReal,Sys,K,L,'LQR',1);
% ContrSys = ...
    DualObserverBasedRC(freqsReal,Sys,K,L,'poleplacement',1);

% A reduced order observer-based robust controller
```

```
%
% The construction of the controller uses a Galerkin approximation
% of the heat system: The Galerkin arpproximation used in the ...
    controller
% design is a lower dimensional numerical approximation of the ...
    PDE model.
Nlow = 50;
[¬,Sys_Nlow,¬,¬] = ...
    Constr1DHeatCase3(cfun,x0fun,Nlow,IB1,IB2,IC1,IC2);

% Store the numerical approximation in "SysApprox".
SysApprox.AN = Sys_Nlow.A;
SysApprox.BN = Sys_Nlow.B;
SysApprox.CN = Sys_Nlow.C;
SysApprox.D = Sys_Nlow.D;
alpha1 = 1;
alpha2 = 0.5;
Q0 = eye(IMdim(freqsReal,size(SysApprox.CN,1))); % Size = ...
    dimension of the IM
Q1 = eye(size(SysApprox.AN,1)); % Size = dim(V_N)
Q2 = eye(size(SysApprox.AN,1)); % Size = dim(V_N)
R1 = eye(size(SysApprox.CN,1)); % Size = dim(Y)
R2 = eye(size(SysApprox.BN,2)); % Size = dim(U)
ROMorder = 3;

ContrSys = ObserverBasedROMRC(...
    freqsReal,SysApprox,alpha1,alpha2,R1,R2,Q0,Q1,Q2,ROMorder);

Sys = SysConsistent(Sys,yref,wdist,freqsReal);
```

The next part constructs and simulates the behaviour of the closed-loop system.

```
%% Closed-loop construction and simulation

% Construct the closed-loop system
CLSys = ConstrCLSys(Sys,ContrSys);

stabmarg = CLStabMargin(CLSys)

xe0 = [x0;zeros(size(ContrSys.G1,1),1)];

Tend = 8;
tgrid = linspace(0,Tend,300);

CLsim = SimCLSys(CLSys,xe0,yref,wdist,tgrid,[]);
```

Finally, the results of the simulation are plotted in separate figures and the behaviour of the controlled state is animated using the user-defined routines. The option to save the animation as an MPEG-4 or AVI file is commented out.

```
%% Visualization

figure(1)
PlotEigs(CLSys.Ae,[-30 .3 -6 6])
```

```
% Choose whether or not to print titles of the figures
PrintFigureTitles = true;

figure(2)
subplot(3,1,1)
plotOutput(tgrid,yref,CLsim,PrintFigureTitles)
subplot(3,1,2)
plotErrorNorm(tgrid,CLsim,PrintFigureTitles)
subplot(3,1,3)
plotControl(tgrid,CLsim,ContrSys,N,PrintFigureTitles)

% In plotting and animating the state,
% fill in the homogeneous Dirichlet boundary condition at x=1
spgrid = [spgrid 1];

figure(3)
colormap jet
Plot1DHeatSurf(CLsim.xesol(1:N,:),spgrid,tgrid,BCtype)

figure(4)
% No movie recording
[¬,zlims] = ...
    Anim1DHeat(CLsim.xesol(1:N,:),spgrid,tgrid,BCtype,0.03,0);

% Movie recording
% [MovAnim,zlims] = ...
    Anim1DHeat(CLsim.xesol(1:N,:),spgrid,tgrid,BCtype,0.03,1);

%movie(MovAnim)

figure(5)
tt = linspace(0,16,500);
plot(tt,yref(tt),'Color',1.1*[0 0.447 0.741],'Linewidth',3);
title('Reference signal ...
    $y_{ref}$','Interpreter','latex','Fontsize',16)
set(gca,'xgrid','on','ygrid','on','tickdir','out','box','off')

% Export movie to AVI

% AnimExport = VideoWriter('Heat1DCase3-animation.mp4','MPEG-4');
% AnimExport.Quality = 100;

% AnimExport = ...
    VideoWriter('Heat1DCase3-animation.avi','Uncompressed AVI');

% AnimExport.FrameRate = 15;
% open(AnimExport);
% writeVideo(AnimExport,MovAnim);
% close(AnimExport);
```

4. Implemented Controller Types

In this section we list the concrete controllers implemented in **RORPack**. The documentation of the code includes additional information on the usage of the construction routines. The controllers are the following:

- The "minimal robust controller" (including only the internal model), based on references [**?**], [**?**, Sec. IV]. Stabilization of the closed-loop system is based on selection of a suitable *low-gain parameter* $\varepsilon > 0$.

  Calling sequence for the construction:

  ```
  LowGainRC(freqsReal,Pvals,epsgain,Sys)
  ```

  where `Sys` contains the parameters of the plant, `freqsReal` contains the frequencies $\{\omega_k\}_{k=0}^q$ of the signals $y_{ref}(t)$ and $w_{dist}(t)$ in (2.3), `epsgain` is the value of the low-gain parameter $\varepsilon > 0$. Finally, `Pvals` is a $(q+1) \times p \times m$ array containing the values $P(i\omega_k) = C_\Lambda R(i\omega_k, A)B + D \in \mathbb{C}^{p\times m}$ of the transfer function of (2.1) at the complex frequencies $\{i\omega_k\}_{k=0}^q$ of the reference and disturbance signals in (2.3). The parameter `epsgain` can alternatively be a vector of length 2 providing minimal and maximal values for ε. The controller construction has a naive functionality for finding an ε to optimize stability margin of the numerically approximated closed-loop system (simply by starting from the minimal value and increasing ε in steps).

- The "observer-based robust controller", based on references [**?**, Sec. 7], [**?**, Sec. VI], [**?**, Sec. 5]. The closed-loop stability is achieved using an observer for the state of the system (2.1).

  Calling sequence for the construction:

  ```
  ObserverBasedRC(freqsReal,Sys,K21,L,
                  IMstabtype,IMstabmarg)
  ```

  where `Sys` contains the parameters of the plant, `freqsReal` contains the frequencies $\{\omega_k\}_{k=0}^q$ of the signals $y_{ref}(t)$ and $w_{dist}(t)$ in (2.3). The parameters `K21` and `L` describe operators K_{21} and L, respectively, such that $A + BK_{21}^\Lambda$ and $A + LC_\Lambda$ generate exponentially stable semigroups.

  Instead of the approach used in [**?**], the "internal model", i.e., the pair (G_1, B_1), in the controller is stabilized using either LQR-based design (`IMstabtype = 'LQR'`) or pole placement (`IMstabtype = 'poleplacement'`) with a stability margin `IMstabmarg`. Note that the variable `IMstabmarg` only determines the stability margin of the internal model, and the stability margin of the closed-loop system also depends on the stability margins of the semigroups generated by $A + BK_2^\Lambda$ and $A + LC_\Lambda$.

- The "The reduced order observer-based robust controller", for *parabolic* PDE systems, especially convection-diffusion-reaction equations. The controller design is based on Galerkin approximations and model reduction via Balanced Truncation, and was introduced in the reference [**?**] (see also [**?**] for systems with boundary control).

  Calling sequence for the construction:

  ```
  ObserverBasedROMRC(freqsReal,sysApprox,alpha1,alpha2,
                     R1,R2,Q0,Q1N,Q2N,ROMorder)
  ```

where `sysApprox` contains the parameters of the Galerkin approximation (A^N, B^N, C^N, D) of the plant parameters (A, B, C, D) (a struct with fields `sysApprox.AN`, `sysApprox.BN`, `sysApprox.CN`, `sysApprox.D`), `freqsReal` contains the (real) frequencies $\{\omega_k\}_{k=0}^q$ of the signals $y_{ref}(t)$ and $w_{dist}(t)$ in (2.3). The parameters `alpha1`, `alpha2`, `R1`, `R2`, `Q0`, `Q1N`, `Q2N` are parameters for the LQR/LQG-based stabilization of the closed-loop system (details below). Finally, `ROMorder` is the desired dimension of the reduced order "observer-part" of the controller. The final controller has dimension which is equal to the dimension of the internal model plus `ROMorder`.

In particular, the parameters `alpha1` and `alpha2` correspond to $\alpha_1, \alpha_2 \geq 0$ in [**?**, Sec. III.A], and are used as design parameters to determine the closed-loop stability margin (the choice $\alpha_1 = \alpha_2 = 0$ is possible). In particular, $(A+\alpha_k I, B, C)$ should be exponentially stabilizable and detectable for $k = 1, 2$. The parameters `R1`, `R2`, and `Q0` correspond to $R_1 \in \mathcal{L}(Y)$, $R_2 \in \mathcal{L}(U)$, and $Q_0 \in \mathbb{R}^{n_{IM} \times n_{IM}}$ (where $n_{IM} \in \mathbb{N}$ is the dimension of the internal model). All these matrices should be chosen to be invertible and positive definite, and for example choices $R_1 = r_1 I$, $R_2 = r_2 I$, $Q_0 = q_0 I$ for some $r_1, r_2, q_0 > 0$ are possible (the dimension n_{IM} of Q_0 can be computed with the routine `IMdim(freqsReal,p)`). Finally, `Q1N` and `Q2N` correspond to the Galerkin approximations (compatible with `sysApprox`) of the parameters $Q_1 \in \mathcal{L}(U_0, X)$ and $Q_2 \in \mathcal{L}(X, Y_0)$ in [**?**, Sec. III.A]. They can be chosen to correspond to the Galerkin approximations of $Q_1 = q_1 I$ and $Q_2 = q_2 I$ for some $q_1, q_2 > 0$.

The model reduction dimension parameter `ROMorder` corresponds to r in [**?**, Sec. III.A] and it determines the final reduced order dimension of the observer-part of the controller. The size of the parameter should be determined to be sufficiently high so that the closed-loop system is exponentially stable. The result [**?**, Thm. III.1] guarantees that closed-loop stability for the original PDE plant and the reduced order controller is achieved if both the Galerkin approximation order N and the model reduction parameter $r \geq N$ are sufficiently high, but (at the moment) there are no concrete lower bounds for these values. Typically, if the decay of the Hankel singular values of (A^N, B^N, C^N) are sufficiently fast, it is possible to achive the closed-loop stability with a fairly small model reduction parameter r.

- The "dual observer-based robust controller", based on references [**?**, Sec. V], [**?**, Sec. 4]. The closed-loop stability is achieved using a complementary controller structure that coincides with observer-based stabilization of the dual of the closed-loop system.

  Calling sequence for the construction:

  ```
  DualObserverBasedRC(freqsReal,Sys,K2,L1,
                      IMstabtype,IMstabmarg)
  ```

  where `Sys` contains the parameters of the plant, `freqsReal` contains the frequencies $\{\omega_k\}_{k=0}^q$ of the signals $y_{ref}(t)$ and $w_{dist}(t)$ in (2.3).

The parameters `K2` and `L1` describe operators K_2 and L_1, respectively, such that $A + BK_2^\Lambda$ and $A + L_1C_\Lambda$ generate exponentially stable semigroups.

Instead of the approach in [**?**], the "internal model", i.e., the pair (C_1, G_1), in the controller is stabilized using either LQR-based design (`IMstabtype = 'LQR'`) or pole placement (`IMstabtype = 'poleplacement'`) with a predefined stability margin `IMstabmarg`. Note that the variable `IMstabmarg` only determines the stability margin of the internal model, and the stability margin of the closed-loop system also depends on the stability margins of the semigroups generated by $A + BK_2^\Lambda$ and $A + L_1C_\Lambda$.

- The "passive minimal controller" based on [**?**, Thm. 1.2], [**?**, Sec. 5.1], [**?**]. For a strongly stable impedance passive system the closed-loop stability can be achieved using passive controller design and a *power preserving interconnection* between the passive control system and the controller.

  Calling sequence for the construction:

  `PassiveRC(freqsReal,Pvals,epsgain,Sys)`

  where `freqsReal` contains the (real) frequencies $\{\omega_k\}_{k=0}^q$ of the signals $y_{ref}(t)$ and $w_{dist}(t)$ in (2.3)`epsgain` is a parameter $\varepsilon > 0$ that controls the norm of K and `Sys` contains the parameters of the plant. In addition, `Pvals` is a $(q+1) \times p \times m$ array containing the values $P(i\omega_k) = C_\Lambda R(i\omega_k, A)B + D \in \mathbb{C}^{p\times m}$ of the transfer function of (2.1) at the complex frequencies $\{i\omega_k\}_{k=0}^q$ of the reference and disturbance signals in (2.3).

  The parameter `epsgain` can alternatively be a vector of length 2 providing minimal and maximal values for ε. The controller construction has a naive functionality for finding an ε to optimize stability margin of the numerically approximated closed-loop system (simply by starting from the minimal value and increasing ε in steps).

  It should be noted that the controller construction routine does not test passivity, and for a non-passive system the resulting closed-loop system will typically be unstable.

4.1. **Comments on the Controller Parameters.** In this section we make some remarks on the choices of the controller parameters.

The gain parameter $\varepsilon > 0$ in `LowGainRC`. The theory in [**?**, **?**] guarantees that for a stable system (A, B, C, D) there exists $\varepsilon^* > 0$ such that for any $0 < \varepsilon < \varepsilon^*$ the closed-loop system is exponentially stable. The stability margin of the closed-loop system (which directly determines the convergence rate of the regulation error $e(t)$) can be optimized with a suitable choice of $\varepsilon > 0$. Finding such an optimal value of $\varepsilon > 0$ for a PDE system is a challenging task, but in numerical simulations one can use a naive approach of tracking the spectrum of the finite-dimensional closed-loop system matrix $A_e(\varepsilon)$. This is the approach taken in the example cases in **RORPack**, though it should be noted that for general PDE systems there is no guarantee that the value $\varepsilon > 0$ obtained this way would optimize the closed-loop

system for the original PDE system. In the example cases we are mainly interested in finding an $\varepsilon > 0$ which achieves a reasonable rate of convergence rate of the error $\|e(t)\|$.

The gain in the passive robust controller `PassiveRC` can analogously be adjusted with a choice of a parameter $\varepsilon > 0$ to optimize the closed-loop stability margin.

The transfer function values. The controllers make use of the values $P(i\omega_k)$, $P_K(i\omega_k)$ and $P_L(i\omega_k)$ of the transfer functions associated to the original or stabilized versions of the PDE system. The **RORPack** class `LinearSystem` has the necessary routines for computing these values based on the matrices `A`, `B`, `C`, and `D`. However, using the same approximation of the PDE for both controller design and simulation corresponds to essentially controlling the approximation as a finite-dimensional system. If possible, to avoid unrealistically positive results, it is therefore better to use two different approximations for controller construction and simulation. This is especially the case if the validity of the approximation for the computation of the parameters can not be guaranteed with absolute certainty. However, there are of course cases, such Galerkin approximations of parabolic systems [**?**], where the values of the controller parameters can be shown to converge with the order of the approximation, and the above concerns are unnecessary.

For some special PDEs, such as 1D heat or wave equations with constant coefficients, the values $P(i\omega_k)$ may have explicit expressions, but these are very limited special cases. For PDEs with spatially varying parameters there are powerful computational methods that can be used to determine $P(i\omega_k)$, such as the **Chebfun** package (https://chebfun.org/) employed in `heat_1d_2.py` in Section 5.1 ("Case 2").

5. PDE Models of the Example Cases

In this section we introduce the PDE models considered in the example cases and review their fundamental properties.

The considered reference and disturbance signals are in each case combinations of trigonometric functions of the form (2.3) with a given set of frequencies. The precise choices of the signals can be seen from the main files of the examples. Similarly, the chosen initial states are visible from the source files, and in several files alternative initial states are provided (these can be used by uncommenting the corresponding lines of code).

5.1. The 1D Heat Equations.

This collection of examples consider 1D heat equations with spatially varying thermal diffusivity on $\Omega = [0, 1]$ with different configurations of control inputs and measured outputs. The first two cases consider control, disturbance, and observation on the boundary, and the final example considers a system with two distributed inputs and outputs. In general, any combination of the above types of inputs and outputs is possible. The main property from the point of view of robust output regulation is whether or not the uncontrolled system is exponentially stable (minimal low-gain controller can be used) or impedance passive (the passive robust controller can be used).

In each of the cases, the semidiscretization of the PDE is completed using Finite Differences.

Case 1: Neumann boundary input at $\xi = 0$, disturbance and output at $\xi = 1$.

Main file name: `heat_1d_1.py`

The model on $\Omega = [0, 1]$ is

$$\begin{aligned} \frac{\partial x}{\partial t}(\xi, t) &= \frac{\partial}{\partial \xi}(c(\xi)\frac{\partial x}{\partial \xi})(\xi, t), \qquad x(\xi, 0) = x_0(\xi) \\ -\frac{\partial x}{\partial \xi}(0, t) &= u(t), \qquad \frac{\partial x}{\partial \xi}(1, t) = w_{dist}(t), \\ y(t) &= x(1, t), \end{aligned}$$

where $c(\cdot) \geq c_0 > 0$ is the spatially varying *thermal diffusivity* of the material. The uncontrolled system is unstable due to the eigenvalue $\lambda = 0$. The example uses the "Observer-Based Robust Controller" and "Dual Observer-Based Robust Controller" to achieve robust output tracking and disturbance rejection.

Case 2: Input, output, and disturbance at $\xi = 0$, Dirichlet at $\xi = 1$.

Main file name: `heat_1d_2.py`

The model on $\Omega = [0, 1]$ is

$$\begin{aligned} \frac{\partial x}{\partial t}(\xi, t) &= \frac{\partial}{\partial \xi}(c(\xi)\frac{\partial x}{\partial \xi})(\xi, t), \qquad x(\xi, 0) = x_0(\xi) \\ -\frac{\partial x}{\partial \xi}(0, t) &= u(t) + w_{dist}(t), \qquad x(1, t) = 0, \\ y(t) &= x(0, t), \end{aligned}$$

where $c(\cdot) \geq c_0 > 0$ is the spatially varying thermal diffusivity of the material. The uncontrolled system is exponentially stable due to the homogeneous Dirichlet boundary condition at $\xi = 1$. The system is also impedance passive since the control input and measured output are collocated, and because of this the robust output regulation problem can be solved using the Passive Robust Controller.

In this example we use the **Chebfun package** [**?**, **?**] (https://chebfun.org/) in computing the values $P(i\omega_k)$ of the transfer function as well as other parameters required in the controller construction. The Chebfun package utilizes spectral methods and provides powerful and easy-to-use tools for the solution of (especially 1D) boundary value problems with accuracies close to machine precision. Because of this, Chebfun has great potential in controller design for PDEs and it is especially perfectly suited for robust output regulation of this class of systems. The only drawback from **RORPack**'s perspective is that at the moment Chebfun is only implemented in Matlab[2]. However, using the Python-to-Matlab interface library "`matlab.engine`" included in Matlab (since R2014b), it is possible to call Matlab scripts and functions directly from Python. This approach is used by default in the current PDE example case (this requires installed versions of Matlab, Chebfun package, and the separate installation `matlab.engine` Python package, see https://se.mathworks.com/help/matlab/matlab-engine-for-python.html

[2]Though partial Python implementations exist, the most important functionality for solving BVPs can only be found in the original Matlab version.

for details). The downside of this approach is that the startup time of the Matlab engine in Python can be extremely slow, and therefore the computations require a relatively long time. However, we get the benefit of very accurate values of the controlled PDE system to be used in the controller design.

Alternate computations of the controller parameters using the Finite Difference approximation (the one used in the main simulation) are commented out in the example code, and can be uncommented to run the `heat_1d_2` example case without the Matlab interface.

Case 3: Distributed input and output, boundary disturbance at $\xi = 0$, Dirichlet at $\xi = 1$.

Main file name: `heat_1d_3.py`

The model on $\Omega = [0, 1]$ is

$$\frac{\partial x}{\partial t}(\xi, t) = \frac{\partial}{\partial \xi}(c(\xi)\frac{\partial x}{\partial \xi})(\xi, t) + b_1(\xi)u_1(t) + b_2(\xi)u_2(t)$$
$$-\frac{\partial x}{\partial \xi}(0, t) = w_{dist}(t), \qquad x(1, t) = 0, \quad x(\xi, 0) = x_0(\xi),$$
$$y(t) = \int_0^1 \begin{bmatrix} c_1(\xi)x(\xi, t) \\ c_2(\xi)x(\xi, t) \end{bmatrix} d\xi \in \mathbb{R}^2$$

where $c(\cdot) \geq c_0 > 0$ is the spatially varying thermal diffusivity of the material and

$$b_1(\xi) = 10\chi_{[.3,.6]}(\xi), \quad b_2(\xi) = 10\chi_{[.6,.7]}(\xi),$$
$$c_1(\xi) = 10\chi_{[.1,.2]}(\xi), \quad c_2(\xi) = 10\chi_{[.8,.9]}(\xi).$$

Here $\chi_{[a,b]}(\cdot)$ is the characteristic function on the interval $[a, b] \subset [0, 1]$, and thus the control inputs act on the intervals $[0.3, 0.4]$ and $[0.6, 0.7]$, and the outputs measure the average temperatures on the intervals $[0.1, 0.2]$ and $[0.8, 0.9]$. The uncontrolled system is stable because of the homogeneous Dirichlet boundary condition at $\xi = 1$.

Figure 2 shows example results of the simulations including plots of the outputs and reference signals, norm of the regulation error, computed control signals, and the state of the controlled system as a function of ξ and t.

5.2. **The 2D Heat Equations on Rectangular Domains.** These examples consider two-dimensional heat equations on $\Omega = [0, 1] \times [0, 1]$ with boundary input and output and boundary disturbances. The inputs and outputs act in an averaged sense on the boundaries.

The reference and disturbance signals are again of the form (2.3) and can be seen from the main files of the simulations.

Case 1: Collocated input and output.

Main file name: `heat_2d_1.py`

This case is considered in [**?**, Sec. VII]. The system on $\Omega = [0, 1] \times [0, 1]$ with two inputs $u(t) = (u_1(t), u_2(t))^T$ and two outputs $y(t) = (y_1(t), y_2(t))^T$

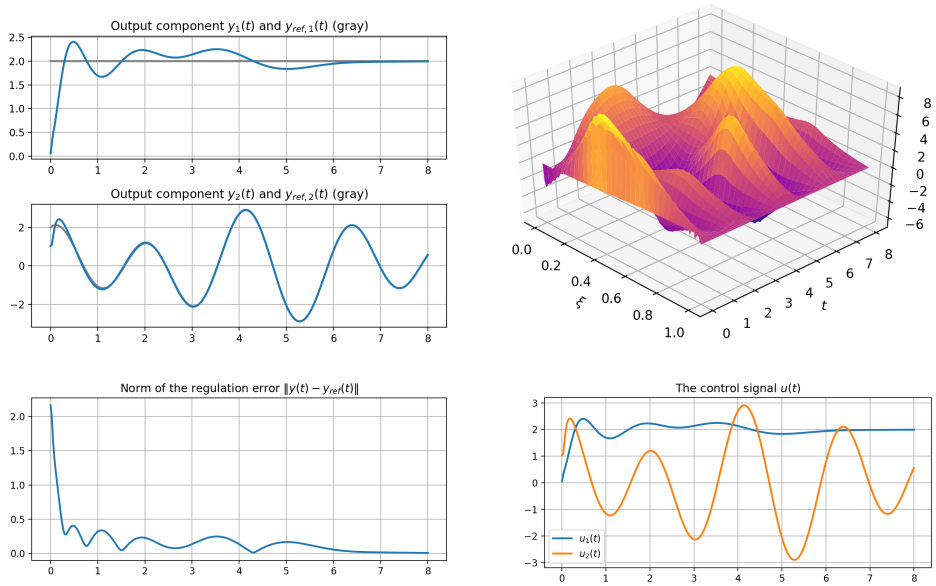

FIGURE 2. Example output of the 1D Heat equation ("Case 3").

is determined by

$$x_t(\xi,t) = \Delta x(\xi,t), \qquad x(\xi,0) = x_0(\xi)$$
$$\frac{\partial x}{\partial n}(\xi,t)|_{\Gamma_1} = u_1(t), \qquad \frac{\partial x}{\partial n}(\xi,t)|_{\Gamma_2} = u_2(t), \qquad \frac{\partial x}{\partial n}(\xi,t)|_{\Gamma_0} = 0$$
$$y_1(t) = \int_{\Gamma_1} x(\xi,t)d\xi, \qquad y_2(t) = \int_{\Gamma_2} x(\xi,t)d\xi.$$

Here the parts Γ_0, Γ_1, and Γ_2 of the boundary $\partial\Omega$ are defined so that $\Gamma_1 = \{\,\xi = (\xi_1, 0) \mid 0 \le \xi_1 \le 1/2\,\}$, $\Gamma_2 = \{\,\xi = (\xi_1, 1) \mid 1/2 \le \xi_1 \le 1\,\}$, $\Gamma_0 = \partial\Omega \setminus (\Gamma_1 \cup \Gamma_2)$. By [**?**, Cor. 2] the heat equation defines a regular linear system with feedthrough $D = 0$. The system is also impedance passive. The uncontrolled system is unstable due to the eigenvalue $\lambda = 0$, but it can be stabilized exponentially with negative output feedback $u(t) = -\kappa y(t)$ for any $\kappa > 0$. In this example we assume the system is pre-stabilized with $\kappa = 1$.

In the simulations, the system is approximated using the eigenmodes of the Laplacian. The system can be controlled either with the low-gain minimal robust controller or the passive controller (with pre-stabilizing negative output feedback), or with one of the observer-based robust controllers.

Case 2: Non-collocated input and output.

Main file name: `heat_2d_2.py`

A similar case (but with collocated inputs and outputs) was considered in [**?**, Sec. 6.3]. The system on $\Omega = [0,1] \times [0,1]$ is

$$x_t(\xi,t) = \Delta x(\xi,t), \qquad x(\xi,0) = x_0(\xi)$$
$$\frac{\partial x}{\partial n}(\xi,t)|_{\Gamma_1} = u(t), \qquad \frac{\partial x}{\partial n}(\xi,t)|_{\Gamma_3} = w_{dist}(t), \qquad \frac{\partial x}{\partial n}(\xi,t)|_{\Gamma_0} = 0$$
$$y(t) = \int_{\Gamma_2} x(\xi,t)d\xi.$$

Here the parts Γ_0, Γ_1, Γ_2, and Γ_3 of the boundary $\partial\Omega$ are defined so that $\Gamma_1 = \{\,\xi = (0,\xi_2) \mid 0 \le \xi_2 \le 1\,\}$, $\Gamma_2 = \{\,\xi = (\xi_1,1) \mid 0 \le \xi_1 \le 1\,\}$, $\Gamma_3 = \{\,\xi = (\xi_1,0) \mid 0 \le \xi_1 \le 1/2\,\}$, $\Gamma_0 = \partial\Omega \setminus (\Gamma_1 \cup \Gamma_2 \cup \Gamma_3)$. By [**?**, Cor. 2] the heat equation defines a regular linear system with feedthrough $D = 0$. The uncontrolled system is unstable due to the eigenvalue $\lambda = 0$.

In the simulations, the system is approximated using Finite Differences with a uniform grid.

5.3. **The 1D Wave Equations.** In these examples we consider one-dimensional undamped and damped wave equations with control and observation at the boundaries and inside the domain.

Case 1: Non-collocated boundary input and output, disturbance near the output.

Main file name: `wave_1d_1.py`

The model on $\Omega = [0,1]$ is

$$\begin{aligned} x_{tt}(\xi,t) &= x_{\xi\xi}(\xi,t), \qquad & x(\xi,0) &= x_0(\xi), \quad & x_t(\xi,0) &= x_1(\xi) \\ -\frac{\partial x}{\partial \xi}(0,t) &= w_{dist}(t), \qquad & \frac{\partial x}{\partial \xi}(1,t) &= u(t), \\ y(t) &= x_t(0,t), \qquad & y_m(t) &= \int_0^1 x(\xi,t)dt \end{aligned}$$

The uncontrolled system is unstable due to lack of damping. It also cannot be stabilized with output feedback due to the non-collocated configuration of the inputs and outputs. The goal is to achieve tracking of the output $y(t)$. The additional measured output $y_m(t)$ is required to achieve closed-loop stability due to the fact that the system is not exponentially detectable with output $y(t)$ (because of the unobservability of the eigenvalue $\lambda = 0$).

In the example case, the second output $y_m(t)$ is used to prestabilize eigenvalue $\lambda = 0$ of the system with preliminary output feedback of the form $u(t) = -\kappa_m y_m(t) + \tilde{u}(t)$ (where $\tilde{u}(t)$ is the new input of the system). There are also other (and better) ways to handle the situation, and these will be implemented in later versions of this example. The pairs (A,B) and (C,A) are stabilized using collocated designs, i.e., we choose $K = -\kappa B^*$ and $L = -\ell C^*$ to stabilize the pairs $A + BK$ and $A + LC$. The main file contains the pre-stabilization gain $K_m > 0$ and the gains $\kappa, \ell > 0$ as design parameters.

The example considers output tracking of the velocity $x_t(0,t)$ to arbitrary 2-periodic reference signals. This is achieved by including frequencies of the form $k\pi$ for $k \in \{1,\dots,q\}$ in the internal model. The reference signal is defined by defining its profile over one period in the variable '`yref1per`'. Note that it is not necessary to compute the amplitudes of the frequency components of $y_{ref}(t)$ (which would be equivalent to finding the Fourier series expansion of the reference signal).

In the simulations the system is approximated using the orthonormal eigenfunctions of the undamped wave operator. Figure 3 shows example results of the simulation for two different periodic reference signals — a nonsmooth triangle signal and signal consisting of semicircles.

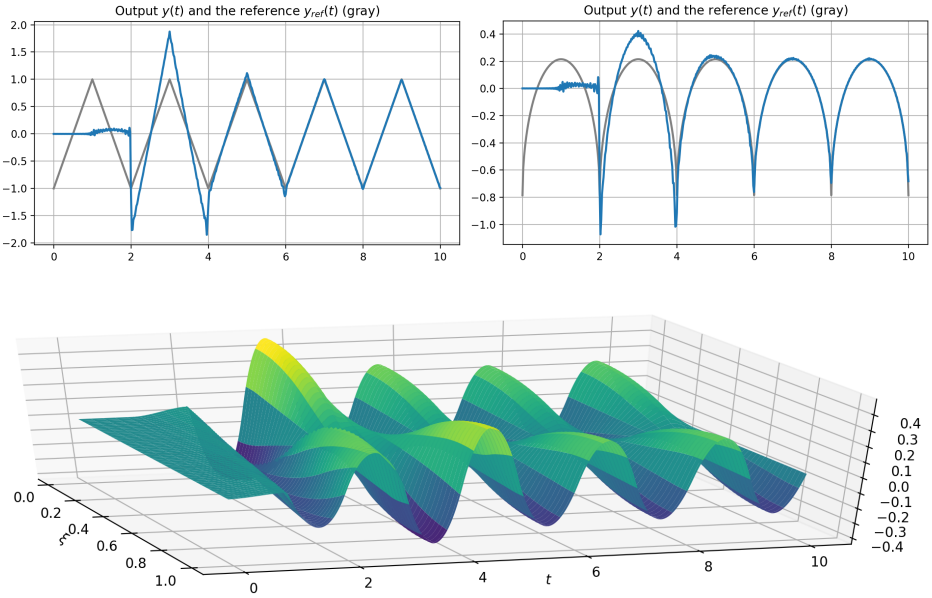

FIGURE 3. Example outputs of the periodic tracking for the 1D wave equation ("Case 1").

Case 2: Collocated distributed input and output.

Main file name: `wave_1d_2.py`

The model on $\Omega = [0, 1]$ is

$$\begin{aligned} x_{tt}(\xi, t) &= x_{\xi\xi}(\xi, t) + b(\xi)u(t) + b_d(\xi)w_{dist}(t) \\ x(0, t) &= 0, \qquad x(1, t) = 0 \\ x(\xi, 0) &= x_0(\xi), \quad x_t(\xi, 0) = x_1(\xi) \\ y(t) &= \int_0^1 b(\xi)x(\xi, t)dt \end{aligned}$$

for some $b(\cdot), b_d(\cdot) \in L^2(0, 1; \mathbb{R})$. The uncontrolled system is unstable due to the lack of damping. Since the input and output are distributed, it is also not exponentially stabilizable or detectable, but instead it is only strongly (or polynomially [**?**]) stabilizable provided that $\langle b(\cdot), \sin(k\pi\cdot)\rangle_{L^2} \neq 0$ for all $k \in \mathbb{N}$. Because of this, it is not possible to use the low-gain controller, and the two observer-based controller designs are not guaranteed to achieve closed-loop stability. However, since the system is impedance passive, the Passive Robust Controller can be used in achieving robust output regulation even in the absence of exponential stability as shown in [**?**, Sec. 5.1]. Due to the sub-exponential closed-loop stability, the regulation error does not converge with any uniform convergence rate, but instead the rate depends on the initial state of the system.

In the simulations the system is approximated using the orthonormal eigenfunctions of the undamped wave operator.

5.4. **The Controlled Beam Equations.** The following example study controller design for controlled beam equation models.

Example: An Euler–Bernoulli Beam with Kelvin–Voigt Damping.

Main file name: `BeamKelvinVoigt_Main.m`

The simulation example is taken from the conference article [**?**]. The goal is to construct a *finite-dimensional* and low-order error feedback controller for output tracking and disturbance rejection of an Euler–Bernoulli beam equation with Kelvin-Voigt damping on $\Omega = (-1, 1)$ [**?**, Sec. 3],

$$\begin{aligned} v_{tt}(\xi,t) + \left(EIv_{\xi\xi} + d_{KV}Iv_{\xi\xi t}\right)_{\xi\xi}(\xi,t) + d_v v_t(\xi,t) \\ = b_1(\xi)u_1(t) + b_2(\xi)u_2(t) + B_{d0}(\xi)w_{dist}(t) \\ v(-1,t) = v_\xi(-1,t) = 0, \\ v(1,t) = v_\xi(1,t) = 0 \\ v(\xi,0) = v_0(\xi), \qquad v_t(\xi,0) = v_1(\xi), \\ y(t) = (v(\xi_1,t), v(\xi_2,t))^T. \end{aligned}$$

The parameters $E > 0$ and $I > 0$ are the (constant) elastic modulus and the second moment of area, respectively, and $d_{KV} > 0$ and $d_v \geq 0$ are the damping coefficients associated to the Kelvin–Voigt damping and viscous damping, respectively. The beam is assumed to have constant density $\rho = 1$.

The system has two control inputs $u(t) = (u_1(t), u_2(t))^T$ and two measured outputs $y(t) = (y_1(t), y_2(t))^T$. The outputs are point observations of the deflection of the beam at distinct points $\xi_1, \xi_2 \in (-1, 1)$. The input profiles $b_1(\cdot), b_2(\cdot) \in C^1(-1, 1; \mathbb{R})$ are fixed input profiles satisfying $b_j(\pm 1) = b_j'(\pm 1) = 0$, $j = 1, 2$, and the disturbance input term has the form $B_{d0}(\xi)w_{dist}(t) = \sum_{k=1}^{n_d} b_{dk}(\xi)w_{dist}^k(t)$ for $w_{dist}(t) = (w_{dist}^k(t))_{k=1}^{n_d}$ and for some fixed but unknown profile functions $b_{dk}(\cdot) \in C^1(-1, 1; \mathbb{R})$ with $b_{dk}(\pm 1) = b_{dk}'(\pm 1) = 0$ for $k \in \{1, \ldots, n_d\}$.

In the simulation example the physical parameters of the model are

$$\rho = 1, \qquad E = 10, \quad I = 1, \quad d_{KV} = 0.01, \quad d_v = 0.4.$$

The point observations are located at $\xi_1 = -0.6$ and $\xi_2 = 0.3$, and the input profiles are

$$b_1(\xi) = \frac{1}{3}(\xi + 1)^2(1 - \xi)^6, \quad \text{and} \quad b_2(\xi) = \frac{1}{3}(\xi + 1)^6(1 - \xi)^2.$$

The system has a single disturbance with a disturbance input profile $b_{d1}(\xi) = (\xi + 1)^2(1 - \xi)^2$.

Because of the Kelvin–Voigt damping, the controlled PDE model satisfies the assumptions required for reduced order internal model based controller design in [**?**]. In this example, the PDE model is approximated using a spectral Galerkin method described in [**?**]. This approximation scheme is used for two purposes, first of all to design the "reduced order internal model based controller" with the routine `ObserverBasedROMRC`, and in addition to approximate the controlled PDE system in the closed-loop simulation (with a higher order approximation).

With the chosen parameters of the beam model the controlled PDE system is in fact exponentially stable (but with a fairly small stability margin). Because of this, it is possible to alternatively use the "low gain internal model based controller" (`LowGainRC`) to achieve output tracking and disturbance

rejection. Construction of this controller is included in the main file (code commented out). However, the small stability margin of the controlled PDE system also severely limits the size of the achievable closed-loop stability margin. On the other hand, in the simulation example the "reduced order internal model based controller" achieves a much higher closed-loop stability margin and convergence rate of the output while only having 4 additional state variables than the low gain controller (whose order is *minimal*), and with only very reasonable increase in the control input gain. For a more detailed comparison of the performances of the two controllers, you can see [**?**, Sec. 4], and complete similar simulations by uncommenting the construction of the low gain controller in the main simulation file.

Figure 4 shows the outputs of the simulation example.

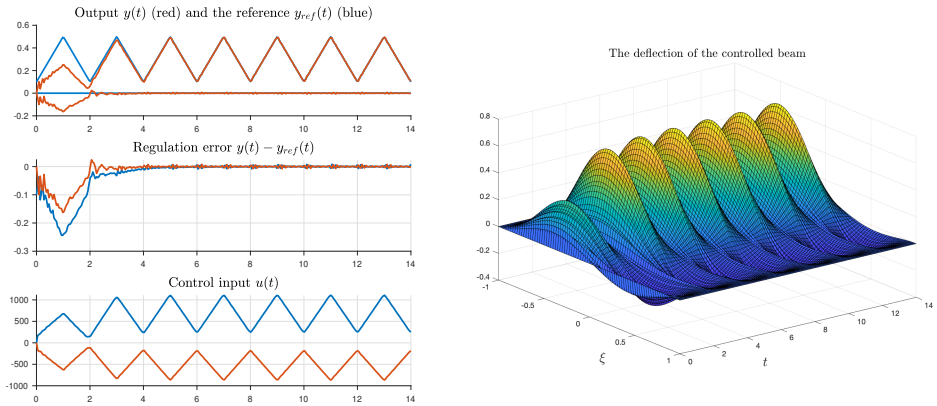

FIGURE 4. Example outputs of the controlled Kelvin–Voigt beam.

Example: A Timoshenko beam with collocated in-domain control and observation.

Main file name: `TimoshenkoLHMNC18_Main.m`

The example is related to the article [**?**]. The controlled Timoshenko beam with constant parameters ρ (mass density), I_ρ rotary moment of inertia, E Young's modulus, I moment of inertia, K shear modulus, is given by

$$\rho \frac{\partial^2 w}{\partial t^2}(\xi,t) = K\frac{\partial}{\partial \xi}\left(\frac{\partial w}{\partial \xi}(\xi,t) - \phi(\xi,t)\right) - d_w \frac{\partial w}{\partial t}(\xi,t)$$

$$I_\rho \frac{\partial^2 \phi}{\partial t^2}(\xi,t) = EI\frac{\partial^2 \phi}{\partial \xi^2} + K\left(\frac{\partial w}{\partial \xi}(\xi,t) - \phi(\xi,t)\right) - d_\phi \frac{\partial \phi}{\partial t}(\xi,t) + \chi_{[1-\eta,1]}(\xi)u(t).$$

Here $w(\xi,t)$ is the displacement of the beam at $\xi \in (0,1)$ and $\phi(\xi,t)$ is the rotation angle of the beam. The d_w and d_ϕ are viscous damping parameters. The control acts through the input profile $\chi_{[1-\eta,1]}(\cdot)$ (the characteristic function of the interval $[1-\eta,1]$ for some $\eta < 1$). The measured output (which is collocated with the control input) is given by

$$y(t) = \int_{1-\eta}^{1} \frac{\partial \phi}{\partial t}(\xi,t)dt.$$

In the simulation example the physical parameters of the model are

$$\rho = 1, \quad I_\rho = 1, \quad E = 1, \quad I = 1, \quad K = 1, \quad d_w = 2, \quad d_\phi = 2, \quad \eta = 0.2.$$

In this example, the PDE model is approximated with Finite Differences scheme. This scheme is very simple and easy to implement, but it should be noted that literature offers several more reliable and advanced numerical approximation methods for this class of PDEs, and some of these are able to, for example, preserve the *port–Hamiltonian* structure of the control system.

By default, the control design used in the example is the "Passive minimal controller", which is constructed using `PassiveRC` (this controller was used in [**?**]). This controller can be used because the system is both exponentially stable and impedance passive. Because the control system has bounded input and output operators, it is also possible to use other control design methods, for example the "observer-based robust controller" (`ObserverBasedRC`) or the "dual observer-based robust controller" (`DualObserverBasedRC`). The former is implemented (and commented out) in the code. For these alternative controller designs, the parameters K_{21} and L (the stabilizing state feedback and output injection gain) can be designed using "collocated design" as $K_{21} = -\kappa B^*$ and $L = -\ell C^*$ for some gain parameters $\kappa, \ell > 0$.

Figure 5 shows the outputs of the simulation example.

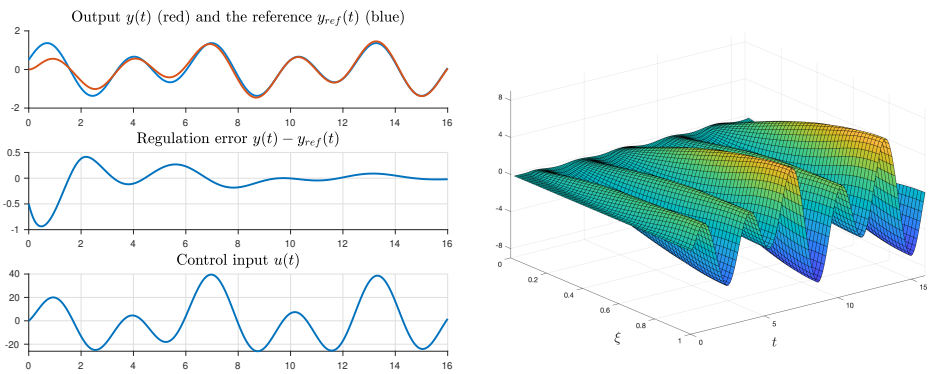

FIGURE 5. Example outputs of the controlled Timoshenko beam.

6. CONTRIBUTORS

Lassi Paunonen: Project leader, developer (2017–)
Jonne Karru: Developer (2021)
Mikko Aarnos: Developer (2018)
Jukka-Pekka Humaloja: Developer (2018–)

APPENDIX A. EXTERNAL LINKS

- https://sysgrouptampere.wordpress.com — Systems Theory Research Group at Tampere University.
- https://github.com/lassipau/rorpack/ — RORPack for Python
- https://github.com/lassipau/rorpack-matlab/ — RORPack for Matlab
- https://lassipau.github.io/rorpack/ — RORPack homepage (hosted by GitHub Pages)

- http://mathesaurus.sourceforge.net/matlab-numpy.html — Useful information on differences in the Matlab and Python/Numpy syntax for vectors and matrices.
- https://chebfun.org — Homepage of the Chebfun Project.

Department of Mathematics, Tampere University, PO. Box 692, 33101 Tampere, Finland

Email address: lassi.paunonen@tuni.fi