MAX PLANCK INSTITUTE FOR DYNAMICS OF COMPLEX TECHNICAL SYSTEMS MAGDEBURG

CSC COMPUTATIONAL METHODS IN SYSTEMS AND CONTROL THEORY

DFG Deutsche Forschungsgemeinschaft

System-Theoretic Model Reduction in pyMOR

L. Balicki[1], P. Benner[1], R. Fritze[2], P. Mlinarić[1], M. Ohlberger[2], S. Rave[2], J. Saak[1], F. Schindler[2]

[1]Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany
[2]University of Münster, Münster, Germany

Overview

pyMOR is a software library for building model order reduction (MOR) applications with the Python programming language. Some of the features are:

- reduced basis and system-theoretic MOR methods,
- integration with external PDE solver packages,
- support for MPI distributed computing,
- permissive open source license (BSD-2-clause).

History

- pyMOR development started in late 2012 at WWU Münster, focusing on reduced basis methods for parameterized PDEs
- version 0.1 released in Apr 2013
- contributions from MPI Magdeburg towards adding system-theoretic methods started in 2015
- pyMOR's design philosophy paper published in 2016:

  R. Milk, S. Rave, F. Schindler
  pyMOR – Generic Algorithms and Interfaces for Model Order Reduction
  SIAM J. Sci. Comput., 38(5), pp. S194–S216, 2016

- DFG project "pyMOR — Sustainable Software for Model Order Reduction" started in Jan 2019
- version 0.5 released in Jan 2019 (the first version to include system-theoretic methods)
- version 2019.2 released in Dec 2019

Input-output systems

The most work went into continuous-time, linear time-invariant systems

$$E\dot{x}(t) = Ax(t) + Bu(t), \quad x(0) = 0,$$
$$y(t) = Cx(t) + Du(t),$$

where u, x, and y are respectively the input, state, and output. In the frequency-domain, we have

$$Y(s) = H(s)U(s), \quad H(s) = C(sE - A)^{-1}B + D,$$

where H is the transfer function. Many methods for first-order systems were extended to second-order systems

$$M\ddot{x}(t) + E\dot{x}(t) + Kx(t) = Bu(t), \quad x(0) = 0, \dot{x}(0) = 0,$$
$$y(t) = C_p x(t) + C_v \dot{x}(t) + Du(t),$$

and some for linear time-delay systems

$$E\dot{x}(t) = \sum_{i=1}^{q} A_i x(t - \tau_i) + Bu(t), \quad x(t) = 0, \ t \leqslant 0,$$
$$y(t) = Cx(t) + Du(t).$$

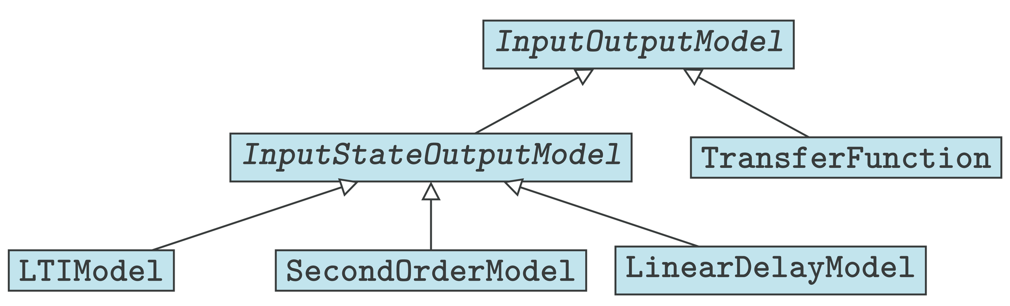


Figure 1: Class diagram of input-output systems in pyMOR 2019.2

System-theoretic methods

- balancing-based and interpolatory methods for first-order and second-order systems
- interpolatory methods for time-delay systems and transfer functions

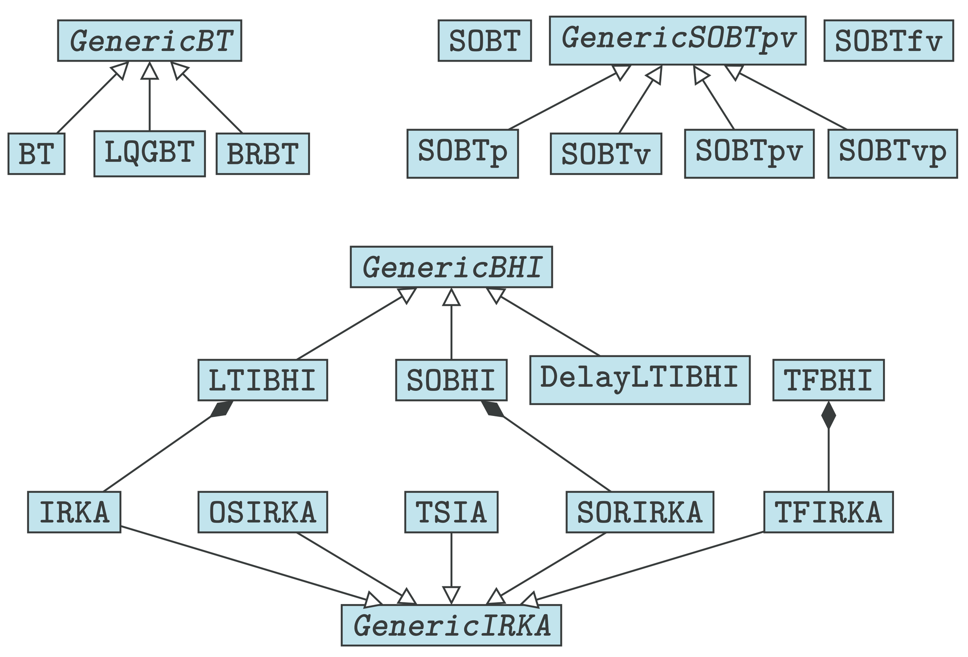


Figure 2: Class diagram of system-theoretic reductors in pyMOR 2019.2

Example

We discretize a heat equation over a cross section of a heat sink using FEniCS, giving a model of order 12296. For the input, we chose heating over the bottom boundary, while for the output the adjoint of the input operator. Figure 3 shows the solution snapshot at $t = 10$, starting from the zero initial condition and with input $u(t) = \sin(\frac{\pi}{3}t)^2$.

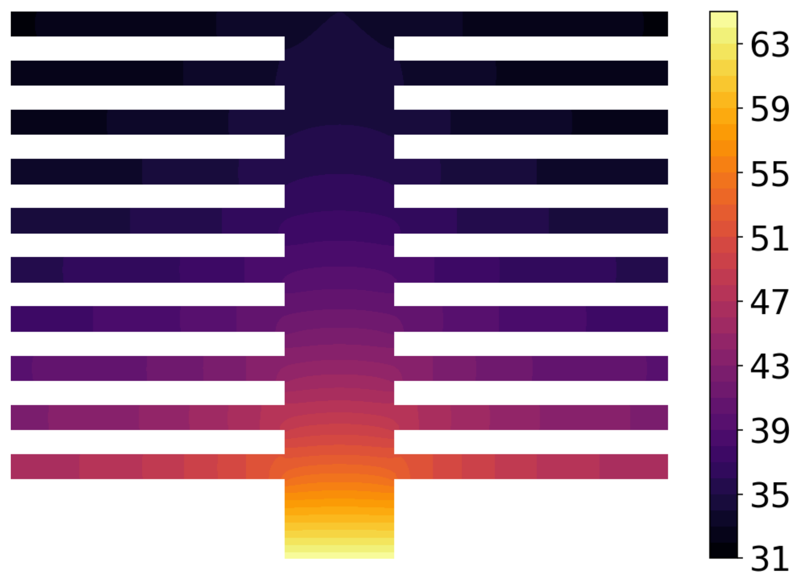


Figure 3: State of the full-order model at $t = 10$

After computing Hankel singular values (which is independent of the chosen input function), using a low-rank Lyapunov equation solver, we can determine upper and lower bounds for the relative $\mathcal{H}_\infty$-error when using balanced truncation. Figure 4 shows these bounds.

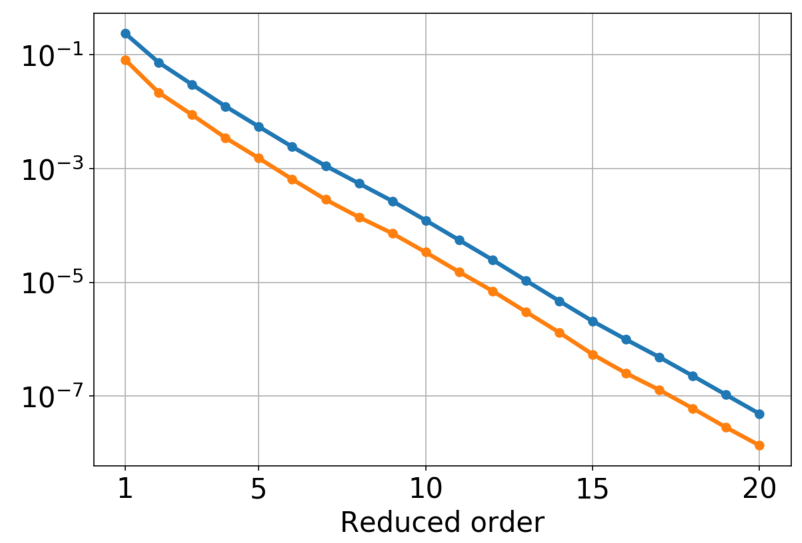


Figure 4: Relative $\mathcal{H}_\infty$-error upper and lower bounds for balanced truncation

We chose order 10 for the reduced-order model, for which the bounds for the relative $\mathcal{H}_\infty$-error are 3.38×10^{-5} and 1.23×10^{-4}. The relative $\mathcal{H}_2$-error can be computed and its value is 7.37×10^{-3}.

In Figures 5 and 6, we compare the full-order and reduced-order model. We see that the output is approximated better than the state.

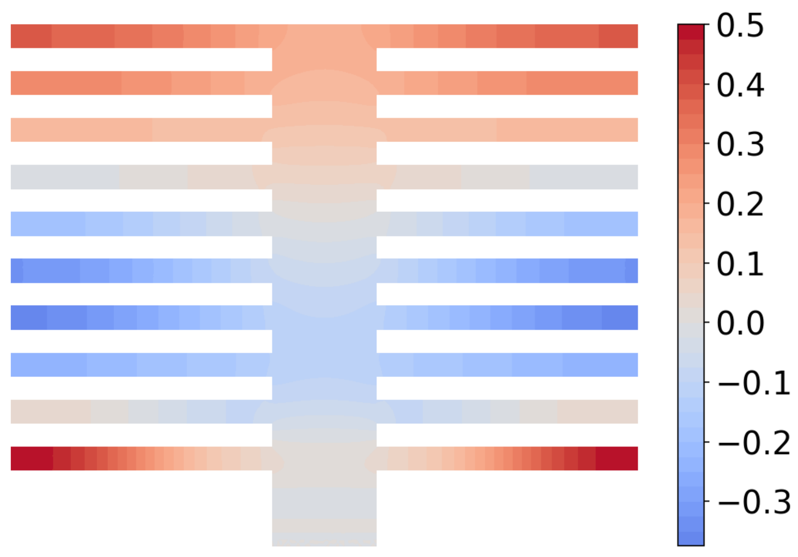


Figure 5: State error between the full-order and reduced-order model at $t = 10$

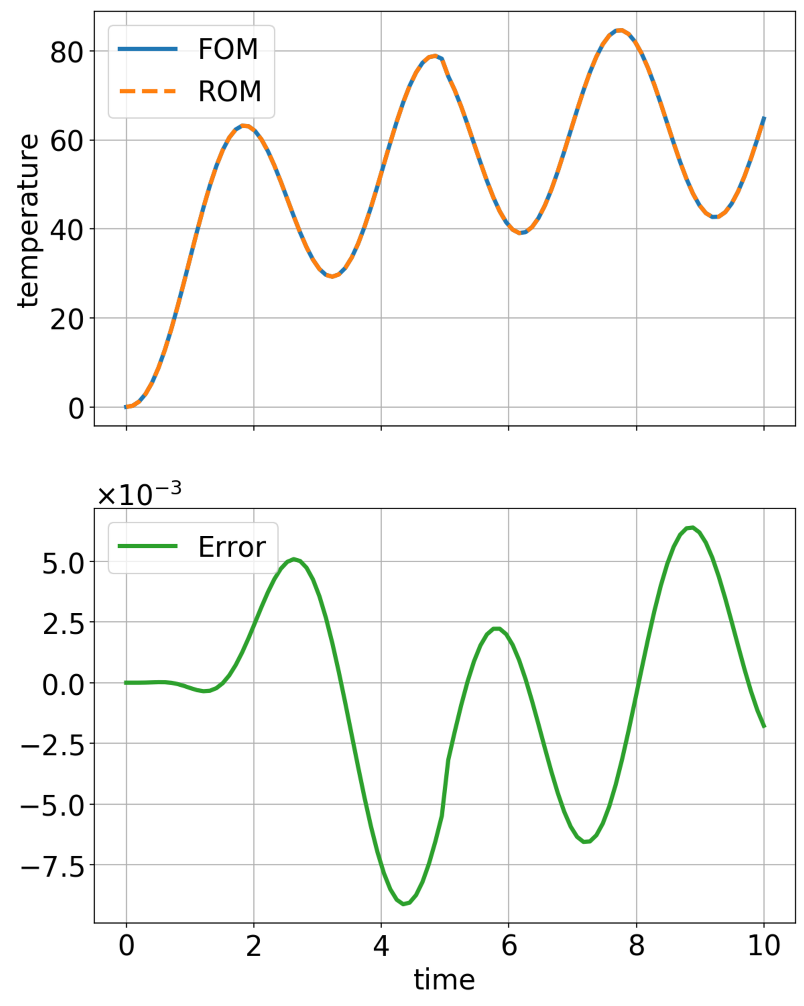


Figure 6: Output of the full-order model (FOM), reduced-order model (ROM), and the error, starting from zero initial condition and with input $u(t) = \sin(\frac{\pi}{3}t)^2$

Building a model

From matrices or files

It is possible to create an `LTIModel` object from NumPy/SciPy matrices or files in Matrix Market or MATLAB format:

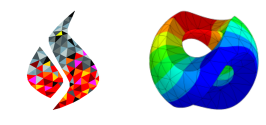
```
from pymor.models.iosys import LTIModel
fom = LTIModel.from_matrices(A, B, C, D, E)
fom = LTIModel.from_abcde_files('file')
fom = LTIModel.from_mat_file('file')
```

By integrating with an external PDE solver package

The specialty of pyMOR is the possibility to link to a PDE solver. Currently supported PDE solver packages are deal.II, DUNE, FEniCS, and NGSolve.

The benefits of this approach are:

- passing of high-dimensional data is not necessary and
- linear systems solvers from PDE packages can be used.



Reductors

Reductors are classes with a `reduce` method. Here is a way to apply balanced truncation (BT):

```
from pymor.reductors.bt import BTReductor
rom = BTReductor(fom).reduce(10)
```

and similarly for the iterative rational Krylov algorithm (IRKA):

```
from pymor.reductors.h2 import IRKAReductor
rom = IRKAReductor(fom).reduce(10)
```

Use the QR code on the right for interactive Jupyter notebooks demonstrating system-theoretic methods available in pyMOR 2019.2 (as shown in Figure 2). It uses `mybinder.org`, which might take a few minutes to build the Docker image.

Matrix equations

Solvers for the following matrix equations types are available in pyMOR:

Lyapunov: $AXE^T + EXA^T + BB^T = 0,$
Riccati: $AXE^T + EXA^T + EXC^TCXE^T + BB^T = 0,$
Sylvester: $AX\widehat{E}^T + EX\widehat{A}^T + B\widehat{B}^T = 0.$

Lyapunov equations appear in balanced truncation and $\mathcal{H}_2$-norm computation, Riccati equations in variants of balanced truncation, and sparse-dense Sylvester equations in the two-sided iteration algorithm (TSIA).

Solvers implemented in pyMOR 2019.2 are:

- low-rank alternate direction implicit method for Lyapunov,
- low-rank RADI method for Riccati, and
- direct solver for sparse-dense Sylvester equations.

Furthermore, bindings to large-scale Lyapunov and Riccati equation solvers in Py-M.E.S.S. are available. Solvers for small dense matrix equations are available through bindings for Py-M.E.S.S. and Slycot.

Installation and references

Supported Python versions
3.6 and above

PyPI
`pip install pymor`

Conda
`conda install -c conda-forge pymor`

Source code
`https://github.com/pymor/pymor`

Documentation
`https://docs.pymor.org`

DOI
10.5281/zenodo.592992

pymor.org