Chair of Automatic Control

Technische Universität München

sss & sssMOR: Analysis & Reduction of Large-Scale Dynamic Systems with MATLAB*

Alessandro Castagnotto, Maria Cruz Varona, Boris Lohmann

The accurate modeling of dynamic systems often results in a large number (>10^4) of differential equations describing the evolution of the system in time. The system matrices can easily become too large for computations or even storage of state-space (ss) objects in MATLAB†. In this contribution, we present two toolboxes that exploit the sparsity of large-scale systems by defining **sparse state-space (sss) objects** and implement both classic and state-of-the-art **model reduction** algorithms.

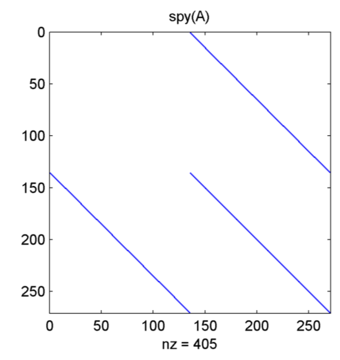
sss

Exploiting sparsity of the system matrices

Linear time-invariant dynamical systems are often given as state-space representations. In a large-scale setting, i.e., when the order N is high ($N \gg 10^3$), the matrices are generally sparse, i.e., the number of nonzero entries is small compared to N^2.

$$\boxed{E}\,\boxed{\dot{x}} = \boxed{A}\,\boxed{x} + \boxed{B}\,\boxed{u} \quad \Big\} \; x \in \mathbb{R}^N$$

$$\boxed{y} = \boxed{C}\,\boxed{x} + \boxed{D}\,\boxed{u}$$



Sparsity pattern of the ISS model‡, used as a benchmark for the example in this poster ($N = 270$)

Unfortunately, MATLAB's Control System Toolbox converts the matrices to "full". For this reason, the definiton of state-space systems by calling

```
sys = ss(A,B,C,D)   or   sys = dss(A,B,C,D,E)
```

is only feasible up until an order of magnitude $\mathcal{O}(10^4)^§$. In fact, note that storing an identity matrix of size 10^5 as "full" requires about 80GB, in the sparse case only 2.4MB!

Functionality

With **sss**, you can exploit sparsity when defining and manipulating dynamical systems. All you need to do is define the system as

```
sys = sss(A,B,C,D,E)
```

to start using most of the tools you are used to, like …

Frequency domain analysis:

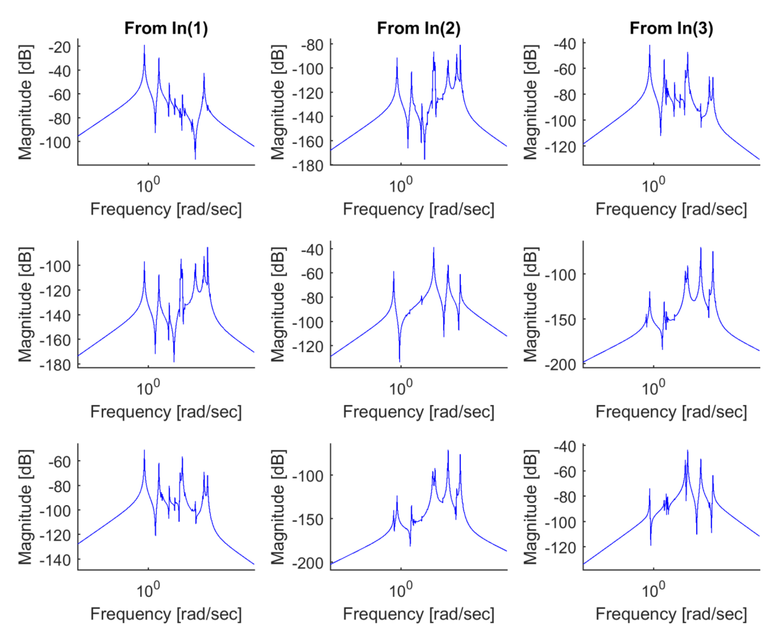
```
>> bode(sys), sigma(sys),…
```

Time domain analysis:

```
>> impulse(sys), step(sys),…
```

Further properties:

```
>> norm(sys), isstable(sys),…
```



… as well as **new** functions such as

```
>> eigs(sys), spy(sys), diag(sys),…
```

Whenever possible, these functions are adapted to exploit sparsity of **sss** objects.

Performance

The following table summarizes a comparison between ss/dss and **sss** computations:

	MATLAB built-in	sss toolbox	improvement factor
storage of `sys`	597.05 KB	24.99 KB	≈ 25
`bode(sys)`	1.83 s	0.64 s	≈ 3
`sigma(sys)`	1.67 s	0.97 s	≈ 2
`c2d(sys)`	0.02 s	< 0.001 s	≈ 20
`residue(sys)`	not feasible	0.12 s	∞

sssMOR

Capturing the relevant dynamics with reduced order models

Even when using the sss toolbox, computations with large-scale models will be expensive. For this reason, we often seek reduced order models as good approximations of much smaller order $n \ll N$. For linear systems, the standard reduction framework is given by Petrov-Galerkin projections of the form

$$\overbrace{W^\top E V}^{E_r}\,\dot{x}_r = \overbrace{W^\top A V}^{A_r}\,x_r + \overbrace{W^\top B}^{B_r}\,u \quad \Big\} \; x_r \in \mathbb{R}^n$$

$$y_r = \underbrace{C V}_{C_r}\,x_r + \underbrace{D}_{D_r}\,u$$

where the projection matrices V, W can be computed with different methods depending on what properties of the original model should be preserved. Classical methods include *modal reduction*, *truncated balanced realizations* and *rational Krylov* methods, while state-of-the-art algorithms include, for instance, *IRKA* and *CUREd SPARK*.

Functionality

Model reduction in **sssMOR** is performed by passing an **sss** object of the original model to the appropriate function, together with some additional parameters.

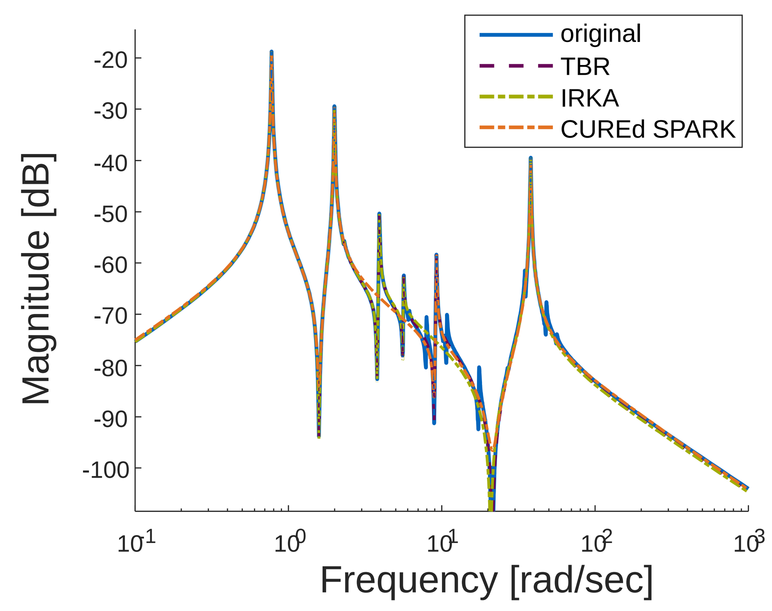
functions	description
`modalMor(sys,n)`	Modal reduction preserving predominant eigenvalues
`tbr(sys,n)`	Truncated balanced realization, retaining dominant Hankel singular values
`rk(sys,sIn,sOut)`	Rational Krylov subspace methods, matching some Taylor series coefficients of the transfer function
`irka(sys,s0)`	Iterative rational Krylov algorithm for $\mathcal{H}_2$-optimal reduction
`cure(sys)`	Cumulative Reduction framework with $\mathcal{H}_2$-pseudo-optimal reduction and adaptive choice of reduced order

Results

For illustration purposes, the reduction is performed on the first element of the transfer matrix only (SISO). This system can be extracted from the **sss** object by calling

```
sys = sysMIMO(1,1)
```

All reduced models shown in the plot below are of order $n = 12$.



red. method	red. time
`modalMor(sys,n)`	0.34 s
`tbr(sys,n)`	0.31 s
`rk(sys,sIn,sOut)`	0.06 s
`irka(sys,s0)`	0.52 s
`cure(sys)`	0.66 s

Notes

sss and **sssMOR** are open-source toolboxes distributed under GPLv2 to foster the academic exchange on software for large-scale applications and model reduction. For more infos, visit www.rt.mw.tum.de/?sssMOR or mail us at sssMOR@rt.mw.tum.de.

*Part of this work is supported by the German Research Foundation (DFG), Grant LO408/19-1.

†MATLAB and Control System Toolbox (Release 2015b) are registered trademarks of The MathWorks, Inc., Natick, Massachusetts, United States.

‡SLICOT benchmark examples: http://slicot.org/20-site/126-benchmark-examples-for-model-reduction

§All computations were conducted on an Intel Core i7-2640M CPU @ 2.80 GHz with 8.00 GB RAM.

Lehrstuhl für Regelungstechnik
Prof. Dr.-Ing. habil. Boris Lohmann

www.rt.mw.tum.de
{a.castagnotto, maria.cruz, lohmann}@tum.de