sssMOR livedemo - control design using reduced order models

This file is part of `sssMOR`, a Sparse State-Space, Model Order Reduction and System Analysis Toolbox developed at the Chair of Automatic Control, Technische Universitaet Muenchen. For updates and further information please visit www.rt.mw.tum.de/?sssMOR.

Author: Alessandro Castagnotto

Last Change: 09 Mar 2017

Introduction to the Control System Toolbox

In this demo, we will use MATLAB functionality available in the *Control System Toolbox,* designed to systematically analyze, design, and tune linear control systems.

First, **load the system matrices** and define a state space model of our system

```
clear, close all, clc

load('building') %file is provided as a benchmark example within sss
```

Notice matrices `A, B, C` (and other data) have been loaded to your workspace. The size of `A` is **48x48**, so this model is fairly small. `B` and `C` are vectors, hence the model is single-input, single-output (**SISO**)

This model describes the mechanical behaviour of a building in the Los Angeles University Hospital with 8 floors, each having 3 degrees of freedom (rotation only on one axis) [1]

Now, define a **dynamic state-space model** using `ss(A,B,C,D)`

```
sys = ss(A,B,C,[]);
```

```
Warning: The "a" matrix was converted from sparse to full.
```

(you can disregard the warning for the moment)

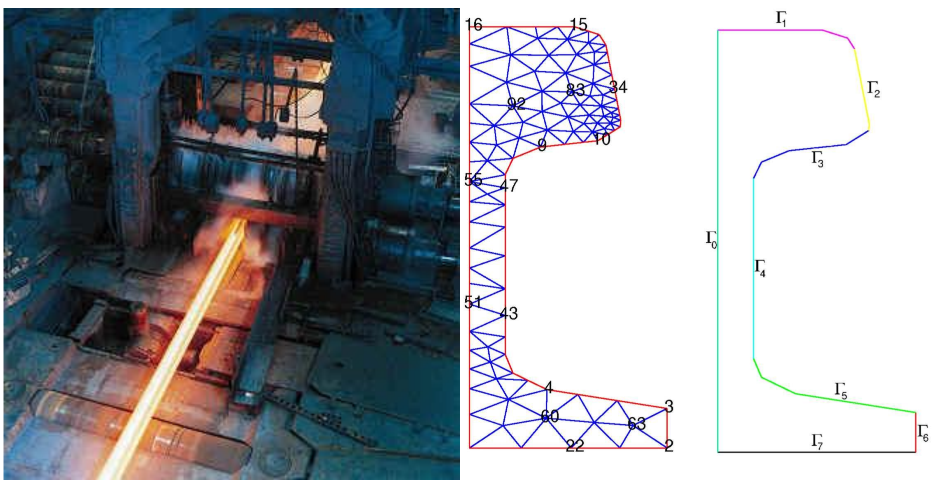
This model can be **analyzed** using all tools provided within the *Control System Toolbox*, such as `bode, impulse, step, norm,` ..., and a **controller** to meet specific requirements can be designed, for example using the `controlSystemDesigner:`

```
% controlSystemDesigner('bode',sys);
```

For this model, we could design e.g. a disturbance rejection controller that regulates an impulse disturbance.

Optimal cooling of a steel profile

Now, we are given a new model that represents the heat transfer process for optimal cooling of a steel profile in an industrial rail production process [2].



This model is available in different sizes, depending on the granularity of the mesh used. In the following, we will use the smallest model with `N=1357` (to be able to analyze it with built-in MATLAB).

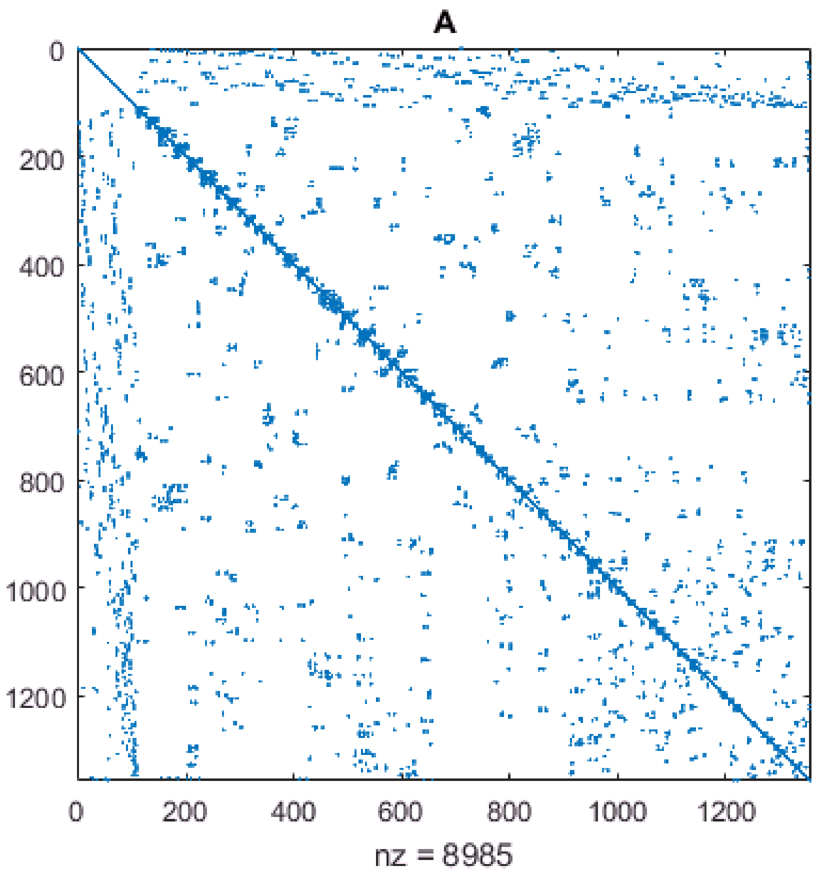
As in the previous example, we can load the system matrices and define a **state-space model**. As the model is given in implicit (**descriptor**) form, we must use the function `dss(A,B,C,D,E)`

```
clear, load('rail_1357')
sys = dss(A,B,C,[],E);
```

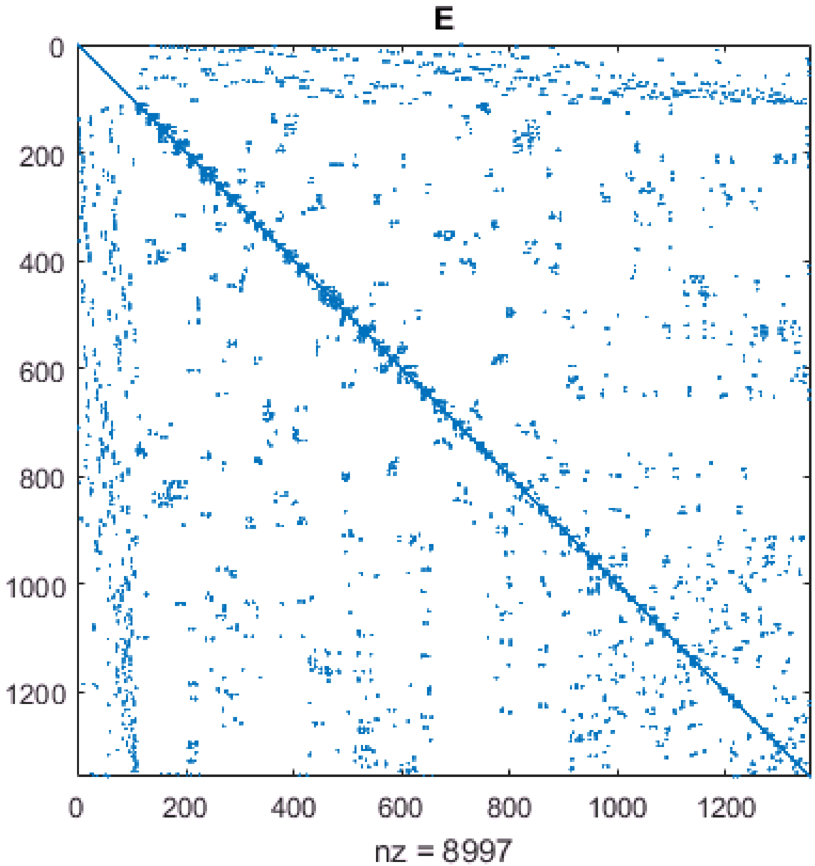
```
Warning: The "a" matrix was converted from sparse to full.
Warning: The "b" matrix was converted from sparse to full.
Warning: The "c" matrix was converted from sparse to full.
Warning: The "e" matrix was converted from sparse to full.
```

Notice again the **warning**: The system matrices we loaded were *sparse*

```
figure; spy(A); title('A');
```




```
figure; spy(E); title('E');
```



MATLAB's `ss` objects do not preserve the sparsity of the system matrices, therefore limiting the usage to models of order $\mathcal{O}(N) = 10^4$ at most.

We can use the functions `disp` and `whos` to display some informations about the model.

```
disp(sys)
```

```
6×7 ss array with properties:

                A: [1357×1357 double]
                B: [1357×7 double]
                C: [6×1357 double]
                D: [6×7 double]
                E: [1357×1357 double]
           Scaled: 0
        StateName: {1357×1 cell}
        StateUnit: {1357×1 cell}
    InternalDelay: [0×1 double]
       InputDelay: [7×1 double]
      OutputDelay: [6×1 double]
               Ts: 0
         TimeUnit: 'seconds'
        InputName: {7×1 cell}
        InputUnit: {7×1 cell}
       InputGroup: [1×1 struct]
       OutputName: {6×1 cell}
       OutputUnit: {6×1 cell}
      OutputGroup: [1×1 struct]
             Name: ''
```

```
           Notes: {}
        UserData: []
    SamplingGrid: [1×1 struct]
```

```
whos sys
```

```
  Name      Size                Bytes  Class    Attributes

  sys       6x7              29605314  ss
```

The model has 6 outputs and 7 inputs (**MIMO**), the inputs corresponding to the temperature boudary conditions in the domains Γ_i in the figure above.

To simplify the analysis, we will take a **SISO subsystem** in the following, e.g. from the 1st input to the 1st output:

```
sys = sys(1,1);
whos sys
```

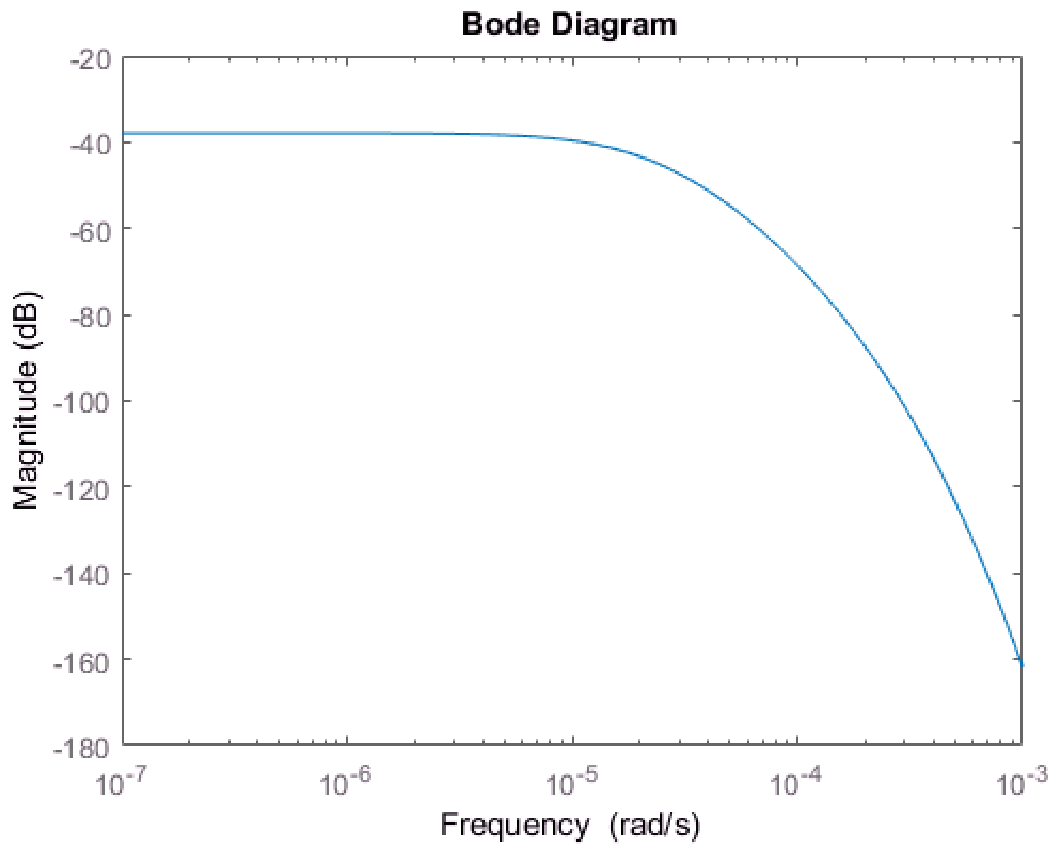
```
  Name      Size                Bytes  Class    Attributes

  sys       1x1              29485482  ss
```

This model is small enoug to be defined as an `ss` object, so we can proceed and analyze it. In the frequency domain, this is commonly done through a **Bode plot**:

```
fh.Bode = figure;
w = {1e-7,1e-3}; %frequency range of interest
tic; bodemag(sys,w); t.Bode = toc
```

```
t = 
    Bode: 42.8605
```



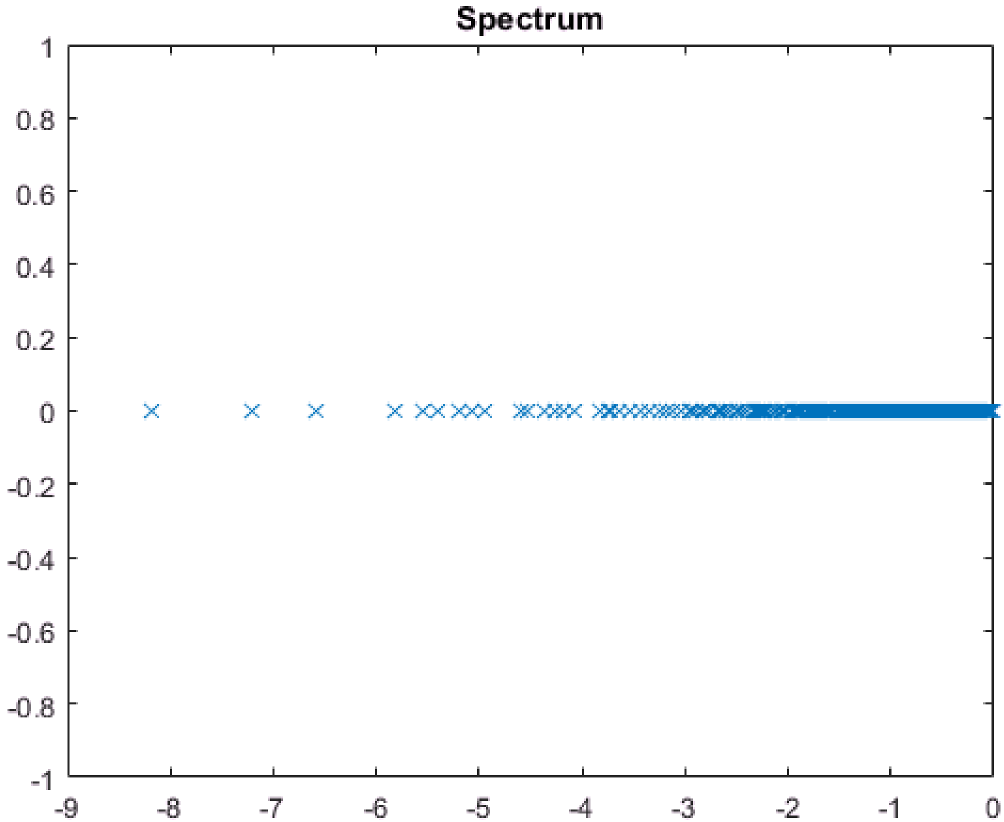
Although we can store the system as an `ss` object, the **computations** performed by built-in MATLAB will not exploit the sparsity of the system matrices, losing efficiency.

Another important quantity when analyzing a linear dynamical system is given by the **spectrum of eigenvalues**

```
fh.Eig = figure;
    tic, lambda = eig(sys); t.Eig = toc
```

```
Warning: Accuracy may be poor in parts of the frequency range. Use the "prescale" command to
maximize accuracy in the range of interest.
t =
    Bode: 42.8605
     Eig: 36.6543
```

```
    plot(complex(lambda),'x'); title('Spectrum');
```



As it can be seen already from this few steps, using `ss` objects for analysis and control design of large-scale models becomes a challenging task due to the **high computational burden**. An interactive control design as shown with the building model becomes almost impossible.

sss - analysis of large-scale, sparse state-space models

To preserve the sparsity of the system matrices in the model and explot it during analysis, we have developed the **sss** toolbox.

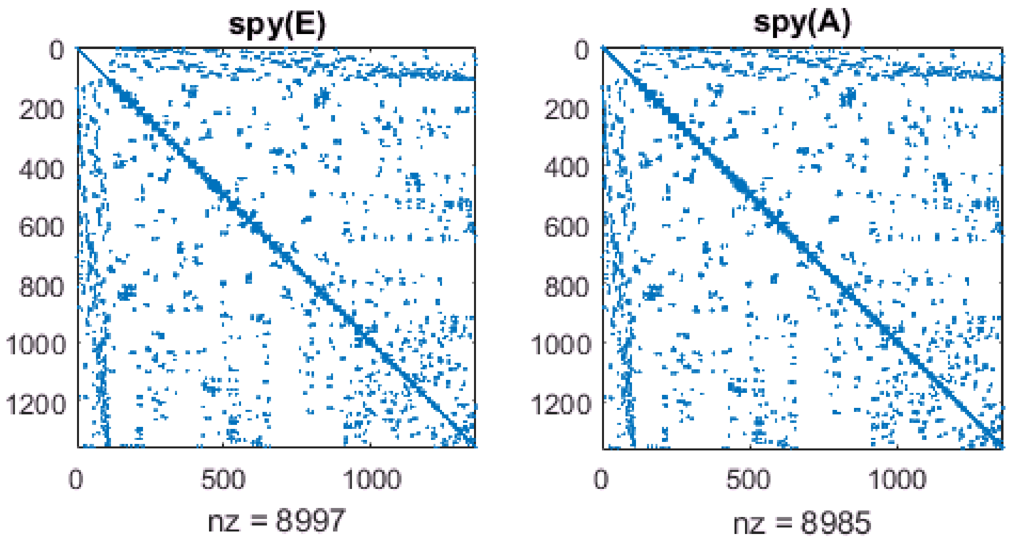
```
doc
```

Defining a **sparse state space model** is as easy as in the built-in case, you just need to add an "s"

```
sysSS = sys; %store the built-in model for comparison
sys = sss(A,B,C,[],E); clear A B C E
```

You can get a first idea about the model at hand for example by inspecting the **sparsity pattern**

```
figure; spy(sys)
```



or by using the function `disp`

```
disp(sys)
```

```
  (DSSS)(MIMO)
  1357 state variables, 7 inputs, 6 outputs
  Continuous-time state-space model.
```

Also in this case we simplify the analysis by taking a SISO subsystem

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

```
sys =
  (DSSS)(SISO)
  1357 state variables, 1 inputs, 1 outputs
  Continuous-time state-space model.
```

Now, let's compare the `ss` and `sss` models in terms of **storage requirements**

```
whos sysSS sys
```

```
  Name        Size                 Bytes  Class     Attributes
```

```
  sys         1x1                321120  sss
  sysSS       1x1              29485482  ss
```

In addition, sss objects include a list of properties that characterize the model, for example **symmetry** of E and A

```
sys.isSym
```

```
ans =
   1
```

or regularity of the E matrix

```
sys.isDae
```

```
ans =
   0
```

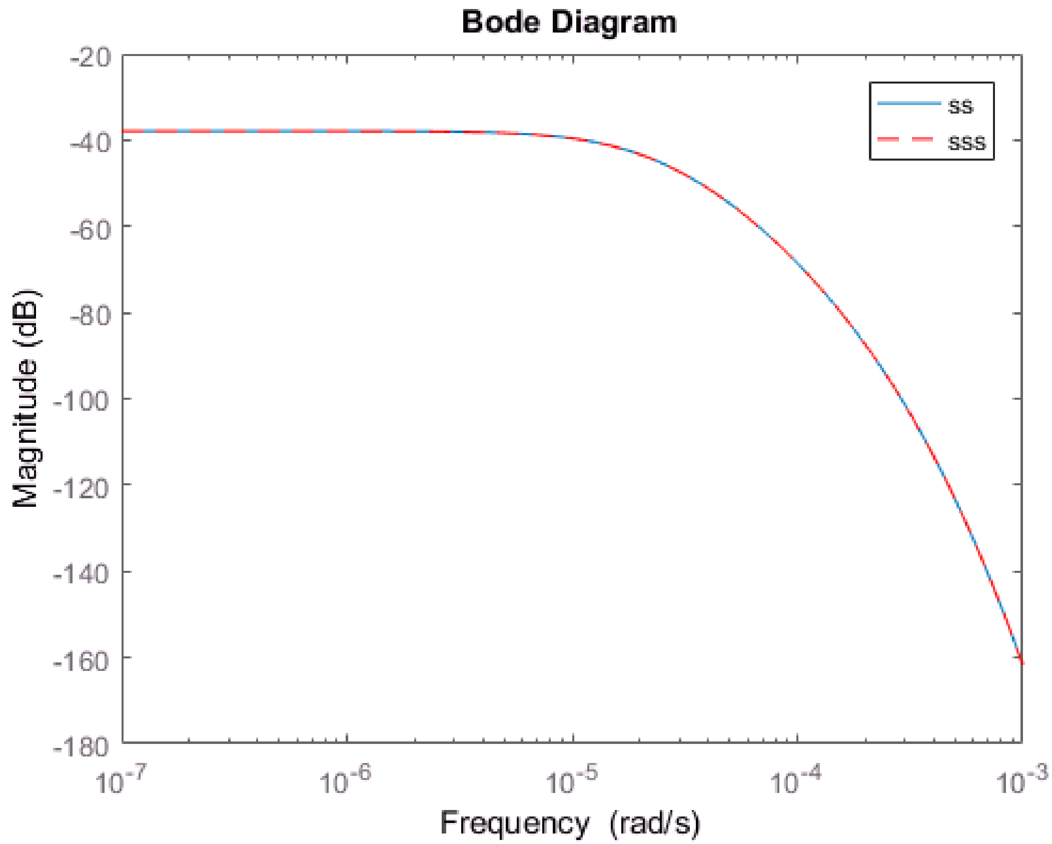
The sss toolbox contains many of the **analysis functions** available in the *Control System Toolbox* and some additions. All functions are programmed to **exploit** whenever possible **the sparsity** of the system matrices.

For example, we can compare the frequency response evaluated with the sss `bode` function.

```
figure(fh.Bode), hold on;
    tic; bodemag(sys,w,'r--'); ts.Bode = toc
```

```
ts =
    Bode: 0.8935
```

```
    legend('ss','sss')
```



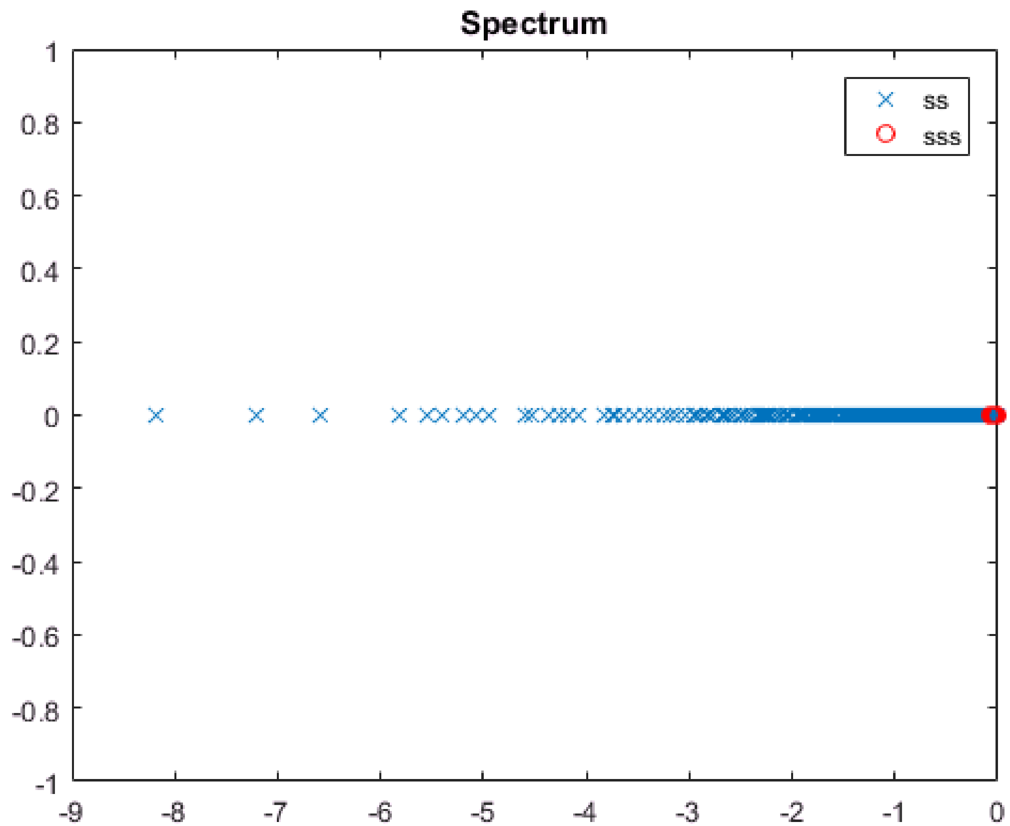
In addition, the computation of the whole eigenvalue spectrum is a dauting task for large-scale models, as it requires dense computations. However, the computation of a small subset can be achieved by using the function `eigs.`

`For example, we can compute the first 300 eigenvalues of smallest magnitude:`

```
figure(fh.Eig), hold on;
tic, lambdaSp = eigs(sys,3e2,'sm'); ts.Eig = toc
```

```
ts =
    Bode: 0.8935
     Eig: 0.8282
```

```
plot(complex(lambdaSp),'ro'); legend('ss','sss')
```



Note the significant time saving that can be achieved by using `sss` objects.

```
t
```

```
t =
    Bode: 42.8605
     Eig: 36.6543
```

```
ts
```

```
ts =
    Bode: 0.8935
     Eig: 0.8282
```

sssMOR - Classical and state-of-the-art model reduction

To obtain a reduced order model that preserves the dominant input-output dynamics we have developed the **sssMOR** toolbox.

```
doc
```

In order to make the control design process more efficient, we now look for a low-order approximation of the original model using model reduction techniques.

For example, we may use the **balanced truncation** method through the function `tbr`. If called only with one input as `syr = tbr(sys)`, the function plots the Hankel Singular Values and waits for user defined order.

This interactive feature is not available in MATLAB live editor, so we will pick a reuced order of n=`10`

```
n = 10;
sysr1 = tbr(sys,10);
```

```
Warning: rctol is not satisfied for S: 1.966493e-05 > rctol (1.000000e-09).
Warning: Maximum number of ADI iterations reached (maxiter = 150). rctol is not satisfied
for R: 3.630029e-05 > rctol (1.000000e-09).
```

Note: tbr finds a **low-rank approximation** of the Gramian matrices using the *M-M.E.S.S.* toolbox developed at the Max Planck Institue in Magdeburg [3]. To notice the difference, try using MATLAB's built-in `balred`.

Reduced order models are generally not sparse, so they have their own class called **`ssRed.`** These objects are a **subclass of ss** and contain information about the reduction.

```
disp(sysr1)
```

```
  (DssRed)(SISO)
  10 state variables, 1 inputs, 1 outputs
  Continuous-time state-space model.
  Reduction Method(s): tbr
  Original order: 1357
```

```
sysr1.reductionParameters.params
```

```
ans =
    originalOrder: 1357
             type: 'adi'
           redErr: 0
           hsvTol: 1.0000e-15
              lse: 'gauss'
              hsv: [57×1 double]
```

We compute a second reduced order model using the iterative rational Krylov algorithm (**IRKA**)

```
sysr2 = irka(sys,10);
```

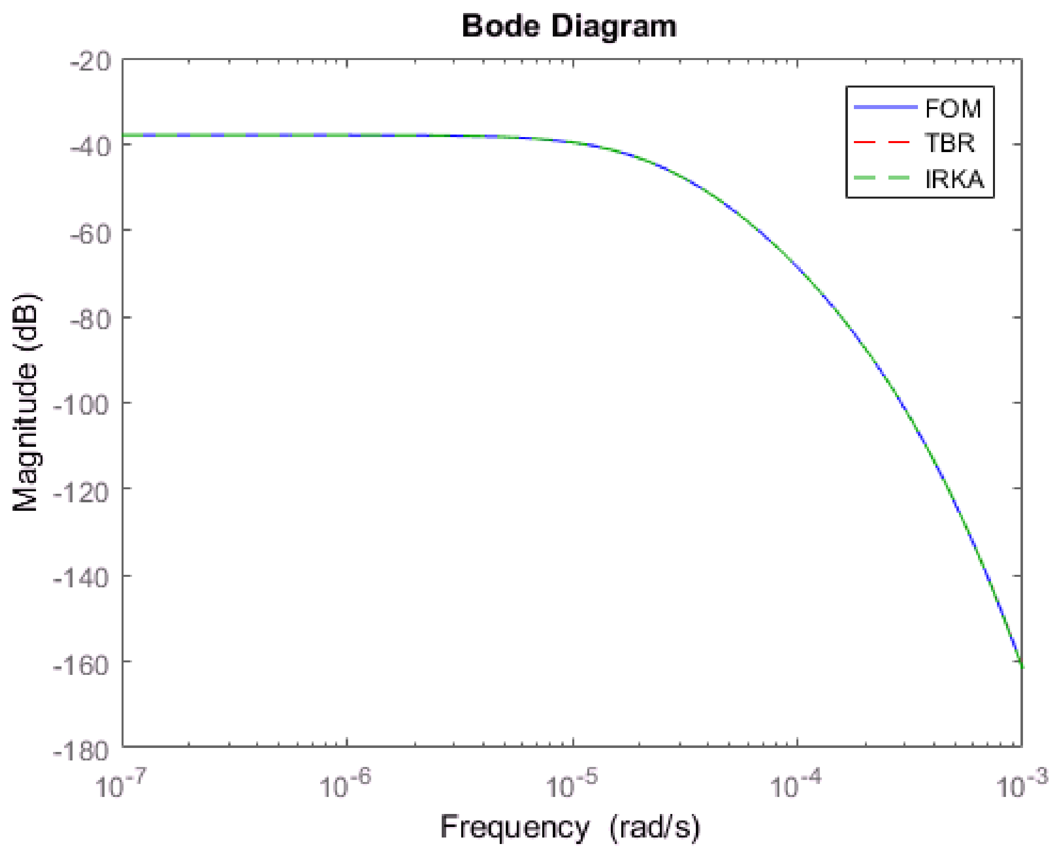
```
IRKA step 004 - Convergence (combAny):  6.2e-01 1.0e-06
```

```
disp(sysr2)
```

```
(DssRed)(SISO)
10 state variables, 1 inputs, 1 outputs
Continuous-time state-space model.
Reduction Method(s): irka
Original order: 1357
```

Using the compatibility between `sss` and `ss`, we can now **analyze the reduction results**

```
figure;
    bodemag(sys,'-b',sysr1,'r--',sysr2,'g--',w)
    legend('FOM','TBR','IRKA')
```



Note that many more reduction functions are available in sssMOR, such as `modalMor, rk, cirka, cure, isrk, rkIcop, spark, ...`

Control design

With the reduced order models, it is now possible to efficiently design a controller that achieves a desired reference temperature `r` within the domain.

```
% controlSystemDesigner('bode',sysr1);
```

References

[1] Chahlaoui, Y. and Van Dooren, P.: "A collection of benchmark examples for model reduction of linear time invariant dynamical systems" (2002) http://slicot.org/20-site/126-benchmark-examples-for-model-reduction

[2] Saak, J.: "Effiziente numerische Lösung eines Optimalsteuerungsproblems für die Abkühlung von Stahlprofilen." (2003)

[3] Saak, J., Köhler, M. and Benner, P.: "M-M.E.S.S.-1.0.1 -- The Matrix Equations Sparse Solvers library" (2016) https://gitlab.mpi-magdeburg.mpg.de/mess/mmess-releases