Computer Exercises - Just for Practice with the SITB
Course on System Identification, Spring 2016
Lennart Ljung

January 11, 2016

Some General Comments

- There are easily more exercises here than you will have time to during the course. Concentrate on the exercises marked with an asterisk (*), and then you can save the others for later training.
- The exercises are intended for MATLAB version 8.4 and the System Identification Toolbox version 9.1 (R2014b).
- It is recommended to run Exerises 1.1 – 2.8 and 3.7 with the GUI – the System Identification app. Most of the problems in Exercises 3 and 4 are better suited for command line commands.

1 Exercise #1: The System Identification Toolbox and Simple Models

In these exercises we shall use the SYSTEM IDENTIFICATION TOOLBOX (SITB) which is a collection of m-files, written in MATLAB. It can be run both in command mode and in a Graphical User Interface (GUI) mode, or in any combination of these.

1.1 * The GUI

In this first exercise we shall get familiar with the basic functions of the SITB in its GUI mode.

Type `systemIdentification` to start the GUI or select the APP System identification under the top toolstrip APPS in the MATLAB Window. Then select the "interactive walkthrough", available under the **Help** menu. This demo deals with real data from a "hair-dryer". This is a laboratory process in which air is let into a tube. At the inlet it passes a mesh of resistor wires, which are heated electrically. The input (u) is the power applied to these. After a transportation delay the temperature of the air is measured. This temperature is the output (y).

The sampling interval is 80 ms, and the input was chosen as a binary random signal.

Follow the demo, and then try to find good models yourself.

1.2 A Simulated System

Let's simulate our own system, and try out various methods with the data: The system we generate and simulate is

$$y(t)-1.5y(t-1)+0.7y(t-2) = u(t-1)+0.5u(t-2)+e(t)-e(t-1)+0.2e(t-2)$$

where $e(t)$ is white Gaussian noise with variance 0.25.

To generate this model and the data, do the following in the MATLAB command window:

```
a=[1 -1.5 0.7]; % The A-polynomial above

b=[0 1 0.5]; % The B-polynomial.  Note that the delay is denoted

    % by the leading zero, corresponding to the term u(t)

c=[1 -1 0.2]; % The C-polynomial

m0 = idpoly(a,b,c);

m0.NoiseVariance = 0.25;

u = sign(randn(300,1)); % The input
```

```
y = sim(u,m0,simOptions('AddNoise',true)); % The output
```

```
It is often useful to pack the data as an iddata object:
```

```
z = iddata(y,u,Ts).  (Ts is the sampling interval.)
```

Import the data `u` and `y` using the **Import Data** pop-up menu, and also import the model `m0` using the **Import Model** pop-up menu.

Then, first try to estimate a model in the "correct" structure, which is `armax` with `nn=[2 2 2 1]`, and evaluate the model's properties (zeros and poles, the frequency function, the step response and so on), and compare with the true, simulated system's properties.

Also try different model structures with `oe, armax, arx, bj` to see how well the parameters, the zeros and poles, the step response, and the frequency function of the system can be estimated in other structures.

1.3 * The Hair-Dryer: Choice of Model Structure

How did we come up with the structure for the hair-dryer `arx, [2 2 3]` in Section 1.1? A suitable path is the following one:

First try correlation analysis to get a feeling for the delay and basic dynamics. That is, use **Correlation Model** in the **Estimate** pop-up and look at the transient response. It is good to look at the impulse response as a stem plot and add confidence interval lines. Use the menus in the transient response window to achieve this.

We see fairly clearly that the delay is 3 samples. (Note that the first element of the impulse response corresponds to delay 0.)

Second, let us get a quick feeling for whether it will be "easy" to find a good linear black box model by trying a fourth order ARX model with the delay we found by correlation analysis. To evaluate its properties we just simulate it with the validation set:

Third, We try many ARX-models (under the estimate-menu item **polynomial models**) to find out which are the best orders and delays (in terms of being able to do well in one-step ahead prediction on the validation data set): Use the **Order Selection** button in the **ARX** panel (`Estimate -> Polynomial Models -> ARX`), or enter any model orders you like.

Suppose we go for the best fit which is obtained for `[4 4 3]`. (This is however only slightly better than `[2 2 3]`).

Then, *fourth* we will check whether the high order model actually is necessary to describe the dynamics from u to y, or whether it is the noise properties that have pushed the need for more poles. This we do by checking possible pole-zero cancellations in the transfer function from u to y. Look at poles and zeros with confidence intervals. The significance of a higher order model is also revealed by the discrepancy between the second and fourth order models in the Bode Plot and in the Model Output.

There is a clear cancellation for two poles/zeros. We could then go for a second order dynamics, with some extra freedom for the noise model. Let's compare that with the other models obtained by `armax` and `oe`. Check out other properties of the model, like its ability to predict and simulate the validation data.

To illustrate how **residual analysis** will reveal bad choices of `nk` and `nb` you can try ARX models of orders `n=[2 2 4]` and `n=[2 1 3]` and look at these models' residual analysis.

1.4 Some Simulated Systems

You have available eight data sets:

`iddata1`

`iddata2`

⋮

`iddata8`

The data object in each of them is called `z1, z2, .. z8`. These are simulated data. There are thus "correct solutions" to them. (These are revealed by `z1.note, z2.note` etc. Note that `idpoly(A,B,C,D,F)` is the polynomial linear model $Ay = B/Fu + C/De$)

Examine several of them, and try to figure out how they have been generated.

If you like, you can use the four step start-up procedure that we outlined in Section 1.3.

1.5 * Real Process Data

Load the `mat`-file `realdata.mat`. It contains several data sets from real applications. Select a few and try to build good models.

`dry2`: Data from a “hairdryer”. The input is the power of the heater and the output is the temperature of the air at the outlet of the dryer. The sampling interval is 0.08 seconds.

`dcservo`: Real data from a laboratory scale DC-servo (Feedback). The output is the angular position of the motor shaft, and the input is the applied voltage to the motor. The sampling interval is 50 ms. In order for the angle to stay within plus minus π, the input had to be determined by feedback. A proportional regulator was applied and the reference signal was chosen as a binary random signal Note that the output has first been measured at time t and then the input with time label t has been applied, so there is no logical possibility that $u(t)$ affects $y(t)$. That means that we shall not use a direct term from input to output (no delay, `nk=0`),. We shall therefore be confined to `nk>0`. Note also that there is an integrator in the system, so it may be difficult to keep the levels in simulations. Therefore it is advisable to use **model output** with, say, 15 step ahead predictions rather than pure simulations.

`steamdat`: Data from the Control Division’s model steam engine driving an electric generator. This process has two inputs and two outputs. The inputs are (1) Pressure of the ”steam” (compressed air) and (2) the magnetization voltage to the generator. The outputs are (1) the frequency and (2) the voltage of the generated current. The experiment was performed in open loop and the sampling interval was 50 ms.

`ecnmtr`: These are the sales data, given by Box and Jenkins (1970). The input is the leading sales indicator, while the output is the actual sales.

`fouga`: These are the gas furnace data, given by Box and Jenkins(1970). The input is the methane (CH_4) flow, while the output is the percentage of the carbondioxide (CO_2) in the fumes.

`glass`: This data sets is taken from a glass drawing manufacturing company in France. The `glass` data contains the drawing speed as input and the thickness of the manufactured glass tube as output. The drawing speed is the speed at which the glass tube is drawn at the production line.

`jas`: These are data from one of the test flights of JAS 39 Gripen. The output is the pitch rate and the three inputs are various control surfaces on the wings.

`biom`: This is a biomethanization process. The input is the dilution rate of the input flow of organic load. The output is the methane gas flow produced by the methanization reaction.

Exercise # 2: Special Methods for Linear System Models

2.1 * Estimation of Spectra

Simulate a signal of AR-type

$$y(t) - 1.5y(t-1) + 0.9y(t-2) = e(t)$$

by

```
m0 = idpoly([1 -1.5 0.9],[]);
```

```
y = sim(randn(500,1),m0);
```

Import the signal and the true system into the GUI as before.

Let's now try and estimate the spectrum of this signal by **Spectral Model**. Use different window sizes and methods and examine the resulting spectra. Compare with the true spectrum.

The mat file `current` contains the variable `i4r`. It contains the current signal in the R-phase when a 400 kV transformer is energized. The sampling interval is 1ms. The signal is very oscillatory and quite wide time windows may be needed.

Of course, spectra can also be estimated by parametric methods using AR and ARMA models.

If time permits, try and get a good estimate of the spectrum of `current` via AR-models.

2.2 * Spectral Analysis

Load any of the simulated data sets `iddata1 ... iddata8` that you worked with in Exercise 1.4. Use (in the GUI) `Estimate>Spectral Models` for spectral analysis estimates of the transfer function. Try different window sizes. Try also a "raw" (unsmoothed) ETFE (by choosing ETFE with default resolution) Compare with what parametric models give!

2.3 Pre-filtering

Load again the signal `i4r` from `current.mat`. First try AR-models with increasing orders and look at the spectra. You will need orders around 20 before a good spectrum with most of the harmonics is obtained. (You may have to increase the plotting resolution of the spectra – using the Options menu) to see all details.

Then try out pre-filtering of the signal, and see if you can estimate certain parts of the spectrum well, using lower order models.

Estimate, with parametric techniques, the 4th and 5th harmonics of the signal!

2.4 * The IV Method and the N4SID Method

Load one of the simulated systems in exercise 1.4. Try the IV-method (found under `Estimate -> Polynomial models -> ARX`) and the N4SID method (found under `Estimate -> State Space models -> Estimation Options`). Study how they compare with the prediction error methods for the same structures.

In particular you may try the ARX orders [2 2 1] for iddata1 with ARX and IV (why is there a difference?).

If time allows and you have interest you may study how the N4SID method behaves with different "horizons" (under `Estimation Options`; see page 531 in textbook – these are the orders of the ARX models used to compute the states).

2.5 Multi-output Systems

Load the data set `marx22.mat`. It contains simulated data (called `z` with two outputs and two inputs. Find a good ARX-model for these data. Split the data set into estimation and validation sets in the usual way. What is the best structure? Try also `N4SID` on these data! Evaluate frequency functions, zeros and poles, simulation behavior, residuals and the like!

2.6 Multi-output Systems, cont'd

Simulate the system

$$y_1(t)-1.5y_1(t-1)-0.5y_2(t-1)+0.7y_1(t-2) = u_1(t-1)+0.5u_2(t-1)+0.5u_1(t-2)$$

$$y_2(t)+0.5y_1(t-1)+0.5y_2(t-1)+0.7y_2(t-2) = u_2(t-1)+0.5u_1(t-1)+0.5u_2(t-2)$$

without noise and with inputs that are shifting randomly between -1 and $+1$. The commands `idpoly` and `sim` are suitable for this.

Add to the simulated outputs white Gaussian measurement noise with zero mean and unit variance.

Estimate models from these data using the `arx` command. Evaluate their properties. Why are they not so good?

Try also `n4sid` on the same data. Conclusions?

2.7 * Frequency domain data

Load the file `frfdata`. It contains a rough estimate of a system's frequency function (as an `idfrd` object). You can look at this complex-valued function as a bode plot by the command `bode(frfdata)`. Import this data set into the GUI (by using `import data object`). Make it the working data and validation data and proceed as before to estimate any model. Note that noise property models cannot be estimated from frequency domain data.

Hint: the data is actually the ETFE of the iddata1 data.

2.8 * Regularization

Load the file `Reg`. It contains the data object `eData` and the true system that generated it, `trueSys`. Import both into the GUI. Try FIR models (i.e. ARX models with na=0) to estimate the impulse response for different orders ([0 nb 0]) using `Estimate -> Polynominal Models -> ARX`. Compare with the true system's impulse response (using the model view `Transient Response, Option: Impulse Response`); turn the confidence region on and off.

Why is it difficult to get a good fit? Try the regularization option offered in the estimation panel (e.g. with the TC kernel) also for quite high nb-orders (like 100).

Although the system is of quite low order it is not very well excited for high order FIR models. Try also `oe([3 2 0])` for a “correct model order”.

Exercise # 3: Grey-box and Nonlinear Models

3.1 * Linear Grey-box Models: Data from a DC motor

[It may be easier to do this exercise in command mode, than in the GUI.]

A simple DC motor with outputs angle and angular velocity can be modeled as discussed on page 95 in the textbook. The dynamics of the motor can – without noise – be described by

$$\frac{d}{dt}x(t) = \begin{pmatrix} 0 & 1 \\ 0 & \theta_1 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ \theta_2 \end{pmatrix} u(t)$$

$$y(t) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} x(t)$$

Use `idss` and its object properties to create a model structure for this. Use the initial/nominal values $\theta_1 = -1$ and $\theta_2 = 0.28$.

You can create this model in continuous time using the `idss` object command:

```
m = idss(A,B,C,D,'ts',0)
```

3.1.1 Simulated Data

In the MAT-file `dcsim` there is a simulated data set from this DC motor. The first output is the angular position of the motor shaft and the second output is the angular velocity. The input is the applied voltage. The sampling interval is 0.1 seconds. As usual, split the data into estimation and validation data:

```
ze = dcsim(1:200), zv = dcsim(201:end)
```

Check how the nominal model describes the data:

```
compare(dcsim,m)
```

Use `arx`, `n4sid`, and `ssest` with suitable structures to determine a good model of the system from the estimation data. Check poles and zeros, frequency functions and simulation and prediction behavior on the validation data.

Then build a grey box model using the `idss` structure with certain parameters fixed according to the model above.

Hint: The model parameters can be fixed and unfixed with commands as

```
m.Structure.a.Free = [0 0; 0 1]
```

Use `ssest` to estimate such a grey box model with parameters fixed according to the above structure.

3.1.2 Real DC Motor Data

Now load the real data in `dcreal` and try the same model structures. Any problems?

Think of the following

- The data consist of "raw" voltages. Is the angular position measurement then necessarily the integral of the velocity measurements? Free an appropriate matrix element!
- The initial value of the position may not be zero. Can it be estimated?

  [`ssest(ze,m,ssestOptions('InitialState','Estimate'));`]
- Output error methods are a bit tricky in this case when the system is not asymptotically stable. How can we "let loose" a noise model?

  [`ssest(ze,m,'DisturbanceModel','Estimate',...`

  `ssestOptions('InitialState','Estimate'));`]
- Due to the integration it may be difficult to require the levels in model and measured angular position to be the same. It is suitable to use `compare` with prediction (say 25 samples ahead) rather than with simulation.

3.1.3 Customized Models

Consider again the simulated data `dcsim`. Suppose that we accurately know the static gain of the DC motor (from input voltage to angular velocity), for example from a previous step-response experiment. If the static gain is G and the time constant of the motor is τ, then the state-space model becomes

$$\frac{d}{dt}x(t) = \begin{pmatrix} 0 & 1 \\ 0 & -1/\tau \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ G/\tau \end{pmatrix} u(t)$$

$$y(t) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} x(t)$$

With G known, there is a dependence between the entries of the different matrices. In order to describe that, the earlier used way to free and fix parameters will not be sufficient.

Write your own m-file that describes this new model structure. Let the static gain G be "an auxiliary argument" (so that it can be changed without editing the file).

[Help: There is such a function with the name `motor` in the toolbox. Type "type motor" to see it.]

Then use `idgrey` to define the structure. Study `help idgrey` for the format.

Estimate the parameter τ under the assumption that the static gain is $G = 0.25$. Compare the obtained model with the models obtained in the other structures.

3.2 Linear Grey-box: A Vibrational System

At Chalmers University an experimental vibrational system has been set up. It consists of a force source (an "exciter") that acts on a table that is fixed to the ground via four springs. The exciter causes the table to vibrate, and the vibrations are measured with a number of accelerometers. The experiment mimics the ones that are used for automobiles and aircraft to investigate vibrational modes.

Some measured data are given in the file `zvib`. The output is the translatational acceleration of the table (in 10^3 cm/ms^2). The input is the applied force (N). The sampling frequency is 300 Hz.

Let y be the position of the table. Let m be its mass, k the spring constant and b the damping factor (viscous damping). Then we have

$$\ddot{y}(t) = -\frac{b}{m}\dot{y}(t) - \frac{k}{m}y(t) + \frac{1}{m}F(t)$$

With the state vector

$$x(t) = \begin{pmatrix} y(t) \\ \dot{y}(t) \end{pmatrix}$$

we have

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ -k/m & -b/m \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1/m \end{pmatrix} F(t)$$

$$\ddot{y}(t) = \begin{pmatrix} -k/m & -b/m \end{pmatrix} x(t) + \frac{1}{m} F(t)$$

For the experimental setup, m is of the order of magnitude 1, k is of the order of magnitude 100000 and b is of the order of magnitude 100.

Estimate m, k and b from the data! Evaluate the obtained model, and compare it with what `arx` and `n4sid` and other input-output models give!

3.3 Nonlinear Grey-Box Model: Static Friction

A well known friction model proposed by Makkar, et al (2005), links the slip speed v(t) of a body in contact with another body to the friction force f(t) via the static relationship

```
f(t) = g(1)*(tanh(g(2)*v(t) - tanh(g(3)*v(t)) + g(4)*tanh(g(5)*v(t))
+ g(6)*v(t)
```

where g(1), ..., g(6) are 6 unknown positive parameters.

At our disposal are 2 different (simulated) data sets where the input slip speed was swept from -10 m/s to 10 m/s in a ramp-type manner. We load the data and create two IDDATA objects for our identification experiments, ze for estimation and zv for validation purposes. The data sets are given in the mat-file `friction`.

Estimate the 6 unknown parameters from these data!

Hints: An m-file that describes the friction model is given by `friction-m` is available in the data folder. This m-file can be packaged into an `idnlgrey` object by

```
FileName      = 'friction_m';          % File describing the
                                       % model structure.
Order         = [1 1 0];               % Model orders [ny nu nx].
Parameters    = {[0.20; 90; 11; ...
```

```
                  0.12; 110; 0.015]};  % Initial parameters.
InitialStates = [];                    % Initial initial states.
Ts            = 0;                     % Time-continuous system.
nlgr = idnlgrey(FileName, Order, Parameters, InitialStates, Ts,    ...
                'Name', 'Static friction model',                   ...
                'InputName', 'Slip speed', 'InputUnit', 'm/s',     ...
                'OutputName', 'Friction force', 'OutputUnit', 'N', ...
                'TimeUnit', 's');
nlgr = setpar(nlgr, 'Minimum', {zeros(6, 1)});   % All
                                     % parameters must be >= 0.
```

To save some time for you, this idnlgrey object was also contained in the mat-file `friction`. Use `compare` to check how well the initial model `nlgrey` describes the validation data. Use `pem` to estimate the model parameters from the estimation data and check if the fit to validation data is improved. Check with `m.parameters.Value` how much the parameters have been adjusted.

In this case the initial parameters were in reasonable vicinity of the true ones. Test different initial values by

`m.parameters.Value=[1 2 3 4 5 6]'` e.g.

and fixing parameters with

`m.parameters.Fixed=[1 0 1 1 0 0]'` e.g.

how good prior knowledge you need to get reasonable estimates.

3.4 Nonlinear Grey-Box Model: A non-adiabatic continuous stirred tank reactor

Type `iddemo` in the command window and select nonlinear grey box examples - Example # 9. Go through the demo by clicking the `next` button.

3.5 Linear Regressions With Physical Insight: Solar Heated House

First load the data from the solar heated house in the file `solardat.mat`. The iddata object `z` contains the temperature of the heat storage, the pump velocity (on/off), and the solar radiation. The sampling interval is 10 minutes. Examine the data and try out some standard (linear) models!

Then test the non-linear transformation of raw data suggested in Ljung (1999). (Note that the temperature should be measured as the difference to the outdoor temperature for the physics to be consistent. Note also that even when the pump is off we should assume a certain flow, like 10 percent of the total one, due to convection.) If you like, there is an m-file `nldattr` to help you with the transformation.

Estimate models based on the transformed variables. To use `compare` now to test the model properties is not so revealing, since the new "inputs" (ϕ_i) contain information about the actual, measured temperature. Instead the whole, non-linear model should be simulated using only past measurements of pump flow and solar radiation. The m-file `sunsim` will help you with this.

Do the physical transformations take care of all effects? Look at the "bump" in the temperature around hour 12-15 (sample 72-90). All models have difficulties to explain that. What do you think is the cause of this phenomenon?

3.6 Semiphysical Modelling: Another Heating Process

Consider the following process. An electric heater is put into a fluid, whose temperature varies (basically due to the influx of new fluid). The internal resistance of the heater is 5 ohms. The temperature of the fluid (tf), the temperature of the heater (y) and the applied voltage to the heater (u) is measured. Construct a model of the system (i.e. how does y depend on u and tf?) based on the (simulated) data in the file `zheat`!. Use as few parameters as possible!

3.7 * Data from a Nonlinear system

Load the file `nldata`. It contains the data object `znl`. Import it into the GUI and find a nonlinear model that describes a validation part of the data as well as possible.

Hints: `Estimate -> Nonlinear models` shows rather well what is available for black-box nonlinear models. Note the different options, e.g. the possibility to randomly restart the search for neural network models to come to grips with local minima.

Exercise # 4: Recursive Algorithms

4.1 * Testing the Algorithms

Load the data in the mat-file `recdata`. The data is called `zrec`. It has been simulated by the system

$$(1 - 1.5q^{-1} + 0.7q^{-2})y(t) = (q^{-1} + 0.5q^{-2})u(t) + (1 - q^{-1} + 0.2q^{-2})e(t)$$

Estimate the parameters of this model by various of the recursive algorithms. Try, e.g., `roe`, `rplr`, `rarmax`. Try various adaptation mechanisms `'ff', 'ug', 'ng', 'kf'` and adaptation gains. The time variation of all the parameters are easily plotted by `plot(thm)`, but it may be easier to concentrate on a few of the parameters.

4.2 On-Line Applications

In on-line applications the estimates are required at the same time as the data are observed. The recursive algorithms in the SITB can be used also in that way, provided relevant information is stored from sample to sample. The conceptual algorithm becomes

1. Wait for measurements y and u.
2. Update:

   ```
   [th,yh,P,phi] = rarx([y u],[na,nb,nk],'ff',0.98,th',P,phi);
   ```
3. Use `th` for whatever on-line application required.
4. Go to 1.

Thus the previous estimate `th` is fed back into the algorithm along with the previous value of the matrix `P` and the data vector `phi`.

Try out this mode by plotting the estimates sequentially. It is useful to fix the axis of the plot by a separate first step, which also produces start-up values of `th, P` and `phi`:

```
[th,yh,P,phi] = rarx(z(1,:),[2 2 1],'ff',0.98);
```

```
axis([1 50 -2 2])

plot(1,th(1),'*',1,th(2),'+',1,th(3),'o',1,th(4),'*'),hold

for k=2:50

   [th,yh,P,phi]=rarx(z(k,:),[2 2 1],'ff',0.98,th',P,phi);

   plot(k,th(1),'*',k,th(2),'+',k,th(3),'o',k,th(4),'*')

end
```

If you prefer, you may write an m-file function for this (taking for example arguments for the adaptation mechanism and adaptation gain), so that you easily can test different variants.

4.3 * Failure Detection by Recursive Identification

Load the data in the mat-file `fail`. They are called `zfail`. The data are generated by a first order system that at some time point undergoes a "failure" in that the delay `nk` changes from 1 to 2. Models of ARX-type will do well. Test how this failure can be detected by recursive identification. Try out different adaptation mechanisms and adaptation gains that allow "quick and certain" detection. If you prefer, you can use on-line plotting as in Exercise 4.2. This will increase the "realism" (but is somewhat slower).

5 Command-Line Commands

Most execises can be run in the System Identication APP ("the GUI"). But some are better suited for command line work. This short list may help you find relevant commands. Use "Help command" or "doc command" for syntax information.

Most execises can be run in the System Identication APP ("the GUI"). But some are better suited for command line work. This short list may help you find relevant commands. Use "Help command" or "doc command" for syntax information.

5.1 Commands for model estimation

- `ssest, n4sid, ssregest, greyest`: commands for estmating state space models.
- `tfest`: command for estimating tranfer function models
- `polyest, arx, or, iv, bj, armax`: commands for estimating polynomial models
- `impulseest`: command for estimating the impulse response.
- `spa, etfe, spafdr`: Commands for estimating frequency functions and spectra
- `nlarx, nlhw`: Commands for estimating non-linear ARX models and Hammerstein-Wiener models.
- `iddata`: Data container for identification

5.2 Commands for evaluating models

- `compare`. Compare model on (validation) data
- `zpkdata, iopzmap`: Compute and plot poles and zeros (same as CSTB)
- `bode, step, impulse`: Plot the corresponding property (same as CSTB).