NTNU

Norwegian University of
Science and Technology

Matlab and Simulink - tips & tricks

Linear System Theory - TTK4115

version 1.0 DØH

Daniel Ørnes Halvorsen
Department of Engineering Cybernetics, NTNU

Intro

MATLAB is numeric computing platform used by engineers and scientists to analyze data, develop algorithms, and create models. Simulink is a block diagram environment for multidomain simulation and model-based design. Combined, they allow you to design systems in a simulation environment through textual and graphical programming.

This text lists a few commands or tools that you might need in the course, TTK4115. Relevant theory and required assumptions from the lectures are not repeated here, only the commands. Contact the email adress below if you find any errors in the text or if you have suggestions of things that could be added.

esmee.oudijk@ntnu.no

1. Matlab

1.1. **Matrix.** To declare the matrix

$$A = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \dots & x_{dn} \end{bmatrix}$$

the elements in each row are entered with a space or comma separation, and each row is separated with a semi-colon. This matrix is a $d \times n$ matrix.

```
A = [x11  x12  x13 ... x1n ;
     x21  x22  x23 ... x2n ;
      ⋮
     xd1  xd2  xd3 ... xdn]
```

1.2. **Vector.** To declare a $d \times 1$ column-vector

$$v = \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ \vdots \\ x_{d1} \end{bmatrix}$$

each entry is separated by a semi colon.

```
v = [x11 ; x21 ; x31 ; ... ; xd1]
```

Alternatively, a row vector $(d \times 1)$ can be declared through

```
row_vec = [start:step:stop]
```

where `start` indicates the first value, `stop` indicates the last value, and `step` is the step-size between vector entries. Pay attention to the orientation of column-vectors and row-vectors and only implement what is required. You can transpose matrices and vectors using a single quote $'$.

```
A_transposed = A'
```

1.3. **Determinant of matrix.** To compute the determinant of a square matrix A,

```
d = det(A)
```

1.4. **Inverse of a matrix A.** To compute the inverse of a matrix A,

```
Y = inv(A)
```

`Y=A^(-1)` works as well

1.5. **Eigenvalues.** Eigenvalues of a matrix A can be computed through,

```
[q,d]=eig(A)
```

where q is the eigenvector matrix whose columns are eigenvectors normalized to length 1. d is a diagonal matrix whose diagonal elements are the eigenvalues of A and corresponds to each eigenvector in q.

1.6. **Equivalence transformation.** Given a $n \times n$ real non-singular matrix P, the equivalence transformation due to P can be computed through

```
[ab,bb,cb,db]=ss2ss(a,b,c,d,P)
```

1.7. **User defined functions.** Declare your own functions using the `function` call.

```
function [y1,...,yn] = myfun(x1,...,xm)

% your code here..

end
```

where `[y1,..,yn]` are the outputs from your function, `myfun` is the name of your function and should be named descriptively, `(x1,..,xm)` are the function arguments or the inputs to your function.

1.8. **If, elseif, else.** `If expression, statements, end` evaluates an expression, and executes a group of statements when the expression is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

The `elseif` and `else` blocks are optional. The `statements` execute only if previous expressions in the `if`...`end` block are false. An `if` block can include multiple `elseif` blocks.

```
if expression
    statements
elseif expression
    statements
else
    statements
end
```

1.9. **State space equations.** To declare a system described by the state space equations a, b, c and d

```
system = ss(a,b,c,d)
```

A simulated response from this system can be computed numerically given discrete time points t, input u and initial state x_0

```
y = lsim(system,u,t,x0)
```

The system response from a step or impulse input can be computed numerically by

```
y_step = step(a,b,c,d)
y_impulse = impulse(a,b,c,d)
```

Go from continous state-space equation matrices, a and b, to discrete state-space equation matrices ad and bd,

```
[ad,bd]=c2d(a,b,T)
```

where T is the sample time.

1.10. **Transfer function.** The transfer function can be computed through the state-space equations by

```
system_ss = ss(a,b,c,d)
system_tf = tf(system_ss)
```

or through the numerator and denominator of the transfer function, n and de,

```
system_tf = tf(n,de)
```

Given the transfer function we can compute numerically the step and impulse response

```
y_step = step(n,de)
y_impulse = impulse(n,de)
```

1.11. **Canonical form.** Equivalence transformation of state-space equations to canonical form with the specified types `type='companion'` or `type='modal'`.

```
[ab,bb,cb,db] = canon(a,b,c,d,type)
```

1.12. **Controllability.** Controllability matrix

```
C = ctrb(a,b)
C = ctrb(system)
```

Gramian matrix

```
Wc = gram(system)
```

Compute rank of the above matrices to verify controllability

```
rk = rank(C)
```

1.13. **Observability.** Observability matrix

```
O = obsv(a,c)
O = obsv(system)
```

1.14. **Kalman decomposition.** The Kalman decomposition of a system can be computed by,

```
[Abar,Bbar,Cbar,T,k] = ctrbf(A,B,C)
```

Minimal realization, eliminates uncontrollable or unobservable state in state-space models, or cancels pole-zero pairs in transfer functions or zero-pole-gain models. The output sysr has minimal order and the same response characteristics as the original model sys.

```
sysr = minreal(system)
```

1.15. **Regulation.** Compute the state-feedback gain matrix K, given the system matrices a and b, and desired poles P.

```
K_gain = place(a,b,P)
```

Compute the optimal state-feedback gain matrix K-optimal by minimizing a quadratic cost function

```
K = lqr(a,b,Q,R)
```

where Q is related to the cost of state deviations, and R is related to cost for control input.

2. Simulink

2.1. **Library browser / list of blocks.** Alternatively, click anywhere in your Simulink environment and start typing the name of the desired block

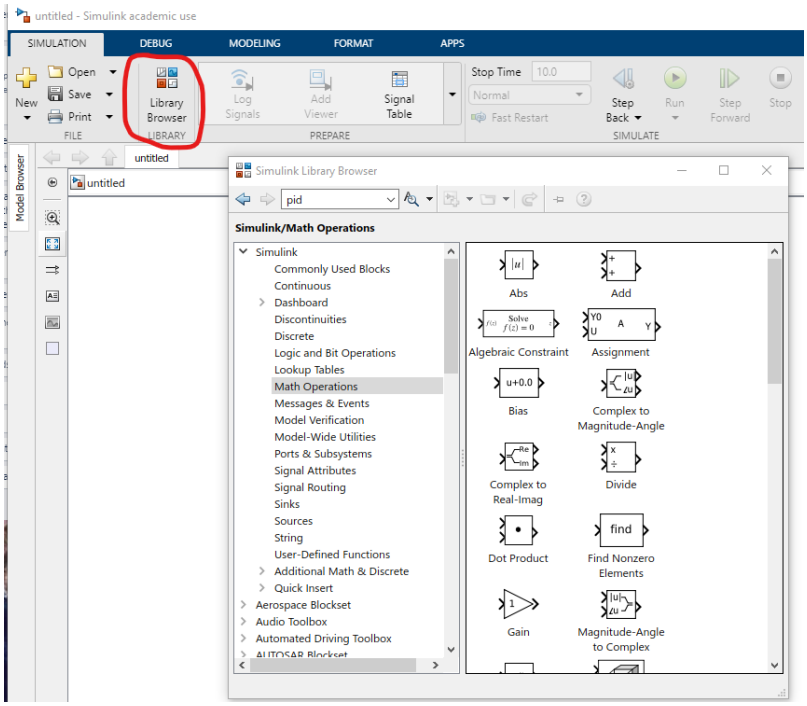

Figure 1. The library browser, marked with a red circle, lets you quickly locate and copy library blocks into a model. Scroll through the categories or search or the relevant block name.

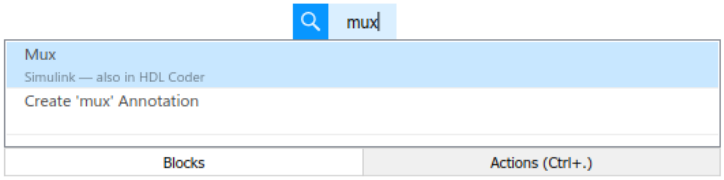

Figure 2. Click anywhere in the Simulink environment and type in the name of your desired block for quicker access than through the Library Browser.

2.2. **Mux, de-mux.**

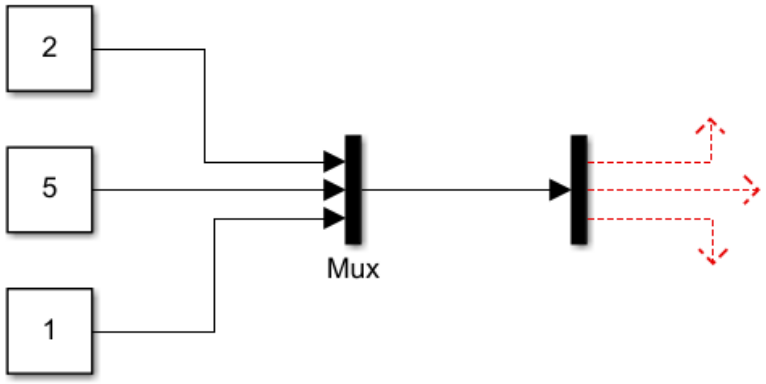

Figure 3. Combine multiple input signals into one through the mux-block. Split the constituents of a signal using the de-mux block.

2.3. **Scope.** The scope-block displays time domain signals. It allows you to adjust the amount of time and the range of input values displayed. Display more signals simultaneously by, e.g. merging signals through the mux-block.

Figure 4. Scope-block.

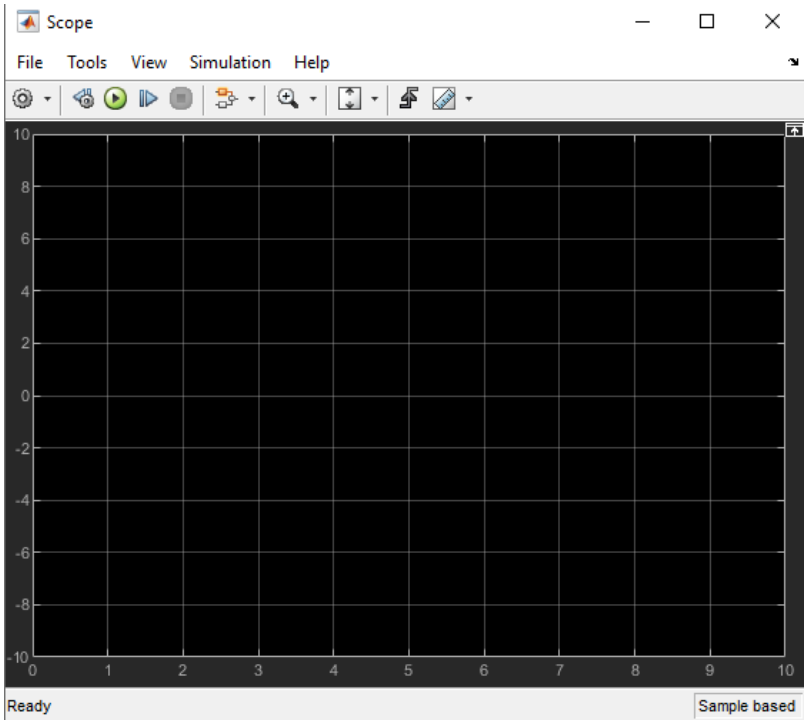

Figure 5. Scope output.

2.4. **Recording data.** To save measured data from the system, you may use a "To File" or "To Workspace" block. The standard time-series format does not work. Instead use one of the others, such as “Array” to export your data; see Fig. 6. You can load the stored data into the Matlab Workspace by using the command `load xxxx`, where `xxxx.mat` is the name of the stored MAT file. Note that the first row of the array in which the data is stored contains the time sequence; the other rows contain the stored signals. You can store multiple signals in one variable by muxing the signals with the Mux block in the Simulink Library Browser. Due to a limit in the QuaRC driver, the following steps has to be made to store data sequences longer than 10 seconds: `Hardware` → `Control Panel` → `Signal & Triggering` → `Trigger options`, you should increase the duration to an appropriate value.

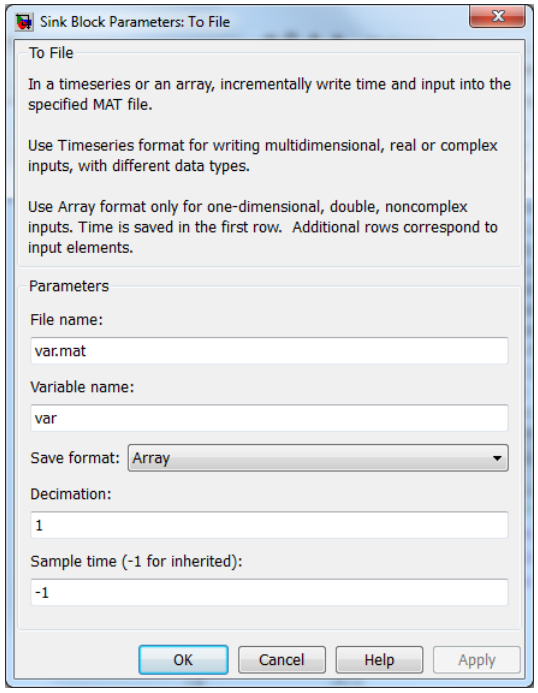

Figure 6. Sending data to a file

2.5. **Change color of blocks.** Click the block you wish to change the color of, then `Format` → `Background` → select a color.

2.6. **Signal and dimensions.** It can be useful to verify the dimensions of the signal input and output of a block. This can be done by pressing `Debug` → `Information Overlays` → `Signal Dimensions`.

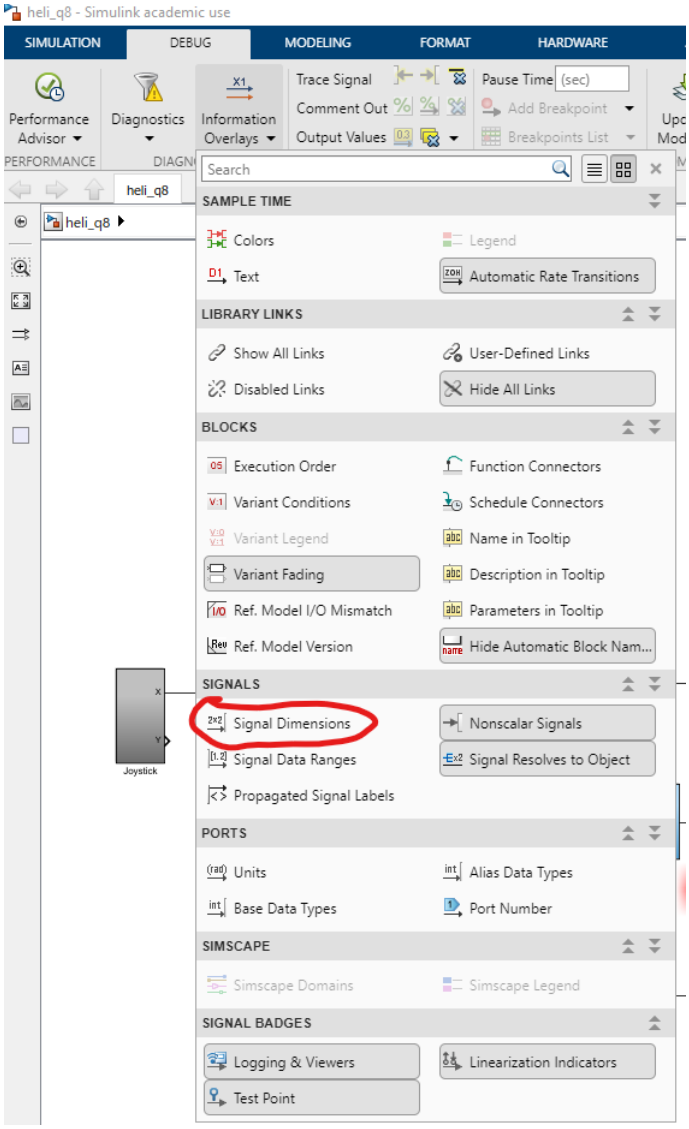

Figure 7. Display the dimensions of input and output signals.

2.7. **Step function as input signal.** Use a step function as input signal to the controllers, instead of the joystick reference signal. This makes it easier to compare different pole configurations. The plot of the comparisons will also look more systematic.

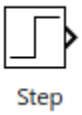

Figure 8. Use a step-function as input instead of the joystick to do repeatable experiments and systematic plots.