AEROSPACE SYSTEMS AND CONTROL THEORY MATLAB COMMANDS

BASED ON LECTURES BY DR R C DE VISSER

SAM VAN ELSLOO

MAY - JUNE 2017

言葉を失いました

1 Basics

Some of the most basic stuff, not really directly related to control theory.

1.1 Indexing

Suppose you have a $N \times M$ matrix. If you'd want the entry in the fifth row and seventh column, you'd write

```
A(5,7)
```

Note that the first number is the row, the second number the column. If you'd want the third entry in the final column (if you don't know the size for example), you can easily retrieve this using

```
A(3,end)
```

If you want the fourth entry of the third-to-last column, you'd write

```
A(4,end-2)
```

If you want only a certain column vector out of the matrix (but all the values in that column), you'd write

```
A(:,3)
```

This tells Matlab to get the entries on all rows, but in the third column.

1.2 Matrix properties

If you have a simple row or column vector, then you can find the minimum, maximum, length and sum of it easily:

```
min(y)
max(y)
length(y)
sum(y)
```

If you have a $N \times M$ matrix, then each of above commands will only give the respective values columns, meaning you get your outputs stored in a $1 \times M$ matrix. You can find the properties for the entire matrix by writing

```
min(min(A))
max(max(A))
sum(sum(A))
```

Note that the size of matrix A can better be retrieved using `size(A)`. This stores it as a 1×2 matrix; the first entry is the number of rows N and the second the number of columns M. So, if you'd want to find the number of rows in A, the best solution would be to write

```
sizeA=size(A)
N=sizeA(1)
```

1.3 Finding stuff in matrices

If you want to find a list of the indices of entries that fulfil a certain property (e.g. you want to find a list of indices for which the entries are larger than 5), you can do so as follows:

```
idx=find(A>5)
```

where `A` is a previously defined matrix. If you then want a row vector containing the associated entries, you write

```
idx=find(A>5)
B=A(idx)
```

This will always return a row-vector of the entries of A that satisfy the condition (even if A is a matrix). There is nothing you can do about that. However, I can't think of a case where you would want it to be a matrix, so don't worry.

1.4 Logical operators

For example, suppose want to create a list of indices of all entries that are either larger than 3 or smaller than -2. This can be written as follows

```
idx=find(A>3 | A<-2)
```

Matlab's way of writing OR is by using that | thing (don't know what's it called in real life). The equivalent of the NOT statement is

```
idx=find(~A>3)
```

which will find the indices of the entries that are NOT larger than 3. Finally, for AND we simply have

```
idx=find(A>3 & mod(a,5)=0)
```

which returns the indices of the entries of A that are larger than 3 and are divisible by 5 (you don't need to know what `(a,5)=0` does I just wanted to give a logical AND statement).

SUMMARY OF INTRODUCTION

1. Use `A(5,7)` to find certain entries (5th row, 7th column). Use `end` to select the last column/row; use `end-1` to find the row/column before that. Select an entire row/column by using : (if you want an entire column, write `A(:,3)`).
2. For row (or column) vectors, simply use `min(y)` etc.; for matrices, use `min(min(y))` etc.
3. Find a list of indices of entries satisfying some condition by writing `idx=find(A>5)`.
4. Make a new row vector containing only the corresponding entries by using `B=A(idx)`.
5. | is the OR operator, ~ is the NOT operator and & is the AND operator.

2 Transfer functions

Stuff that basically exclusively applies to transfer functions (and not to state space systems as well).

2.1 Defining transfer functions

Defining a transfer function is easy peazy:

```
s=tf('s')
h=(3*s+1)/(s*(s^2+2*s+4))
```

It can never hurt to use the `minreal()` command to clean simplify fractions; I don't think there's any way it can hurt your results, but leaving it out can have negative effects (quite frequently actually), so it's better to be safe than sorry in its use.

```
s=tf('s')
h1=minreal(3*s+4)/(2*s+1)
h2=minreal(4*s^3+s)/(2*s^2+3*s+1)
h3=minreal(h1/h2)
```

2.2 Feedback loops

For a transfer function from an input $U(s)$ to output $Y(s)$, the transfer function is given by

$$H(s) = \frac{Y(s)}{U(s)} = \frac{\text{feedforward path}}{1 + \text{feedback path}} \tag{2.1}$$

For example, for the block diagram of figure 2.1 we can deduce the following transfer functions. Between y as

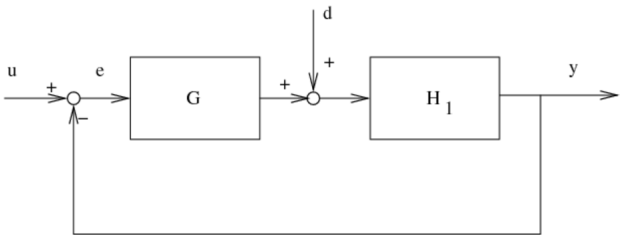


Figure 2.1: Block diagram.

output and u as input, we have

$$H_1(s) = \frac{Y(s)}{U(s)} = \frac{\text{feedforward path}}{1 + \text{feedback path}} = \frac{G \cdot H_1}{1 + G \cdot H_1}$$

Between e as output and u as input, we have

$$H_2(s) = \frac{E(s)}{U(s)} = \frac{\text{feedforward path}}{1 + \text{feedback path}} = \frac{1}{1 + G \cdot H_1}$$

Denote the outcome of G as a; the transfer function between a as output and u as input is

$$H_3(s) = \frac{A(s)}{U(s)} = \frac{\text{feedforward path}}{1 + \text{feedback path}} = \frac{G}{1 + G \cdot H_1}$$

The transfer function from d (the input) to y (the output) is

$$H_4(s) = \frac{Y(s)}{D(s)} = \frac{\text{feedforward path}}{1 + \text{feedback path}} = \frac{H_1}{1 + G \cdot H_1}$$

2.3 Finding responses to inputs

Please note that most of what's written below is applicable only to transfer functions; it's all very similar for statespace systems but then you need to make some changes so please remember the distinction between them.

2.3.1 Time-vector

Creating the time vector is the first step:

```
dt=0.01
t=0:dt:20
```

Usually a timestep of 0.01 will do it, although you can decrease it to 0.001 to improve accuracy. Plot the results to make sure your time vector is large enough (although in nearly all of the E-lectures, 20 seconds has been enough).

2.3.2 Determining the response to certain inputs

If we have a certain transfer function, the response to a step respectively impulse (Dirac delta) can straightforwardly be determined:

```
s=tf('s');
H=1/(2*s^2+5*s+20);
dt=0.01;
t=0:dt:60;
y1=step(H,t);
y2=impulse(H,t);
```

`y1` and `y2` are then row vectors of the same size as the time vector.

The response to an arbitrary input is found by first setting up a row-vector of the input and then using `lsim()`. For example, if we want a pulse (i.e. $u(t) = 1$ for $0 \leq t \leq 1$ and $u(t) = 0$ for $1 < t \leq 20$ then we get

```
s=tf('s');
H=1/(2*s^2+5*s+20);
dt=0.01;
t=0.01:dt:60;
u=[ones(1,1/dt) zeros(1,19/dt)];
ylsim(y,u,t)
```

Note how the input vector is created: `ones(1,1/dt)` creates a row-vector full of ones, with length 1/dt; this makes sure it's as long as one second. Similarly, `zeros(1,19/dt)` makes the zeroes last as one second. Finally, note that if you for example write

```
t=0:dt:20
u=sin(t)
```

then u will automatically be a row-vector of the same size, containing the function $sin(t)$ as you'd expect.

2.3.3 Plotting results

Plotting results is very easy; just write

```
plot(t, y)
```

If you want to plot multiple stuff at once, write

```
plot(t, [y1 y2])
```

2.3.4 *Time domain response criteria*

Overshoot can be determined using the following line of code:

```
overshoot = (max(y1) - y1(end))/y1(end) * 100
```

Peak time can be determined using:

```
tpeak=t(find(y1==max(y1)))
```

Settling time can be determined using:

```
idx=find(y1>y1(end)*1.05 | y1 < y1(end)*0.95)
tsettling=t(idx(end)+1)
```

Delay time can be determined using:

```
idx=find(y1>=0.1*y1(end))
tdelay=t(idx(1))
```

Rise time can be determined using:

```
idx1=find(y1>=0.1*y1(end))
idx2=find(y1>=0.9*y1(end))
trise=t(idx2(1))-t(idx(1))
```

You have to remember these by heart. Another way of determining peak time and settling time is by use of

```
S=lsiminfo(y,t,'SettlingTimeTreshold',0.05)
```

where the last value denotes the threshold to be used (Matlab uses 0.02 as a standard so you need to manually change it). This prints the settling time and the time at which the maximum value is reached (amongst other).

SUMMARY OF TRANSFER FUNCTIONS

1. Transfer functions are easily defined in Matlab. Don't forget to use `minreal()` every now and then; never hurts but lack of it can cause stuff to fuck up.
2. Feedback loops are given by

$$H(s) = \frac{Y(s)}{U(s)} = \frac{\text{feedforward path}}{1 + \text{feedback path}} \tag{2.2}$$

3. Creating time vectors is done by `t=0:dt:20` where `dt` is defined before. Computing the response to a step or impulse input is as simple as `step(H,t)` and `impulse(H,t)`; the results are stored in a row vector. Responses to arbitrary inputs require you to define the input vector. One thing to keep in mind is that for example, if you want your input to equal a certain value (e.g. 2) for a certain amount of time (e.g. 15 seconds), you can do so by writing `u=[2*ones(1,15/dt)]`.
4. Plotting can be done by writing `plot(t,y)` or (if you want to plot multiple things) `plot(t...,[y1 y2]`. It can be advised to write `clf()` above the line where you plot; this will clear the figure from previous plots.
5. Use the the code shown below this text box to find overshoot, peak time, settling time, delay time and rise time, respectively. One can also use `S=lsiminfo(y,t,'SettlingTimeTreshold...',0.05)` to find the settling time and the peak time.

Code below shows the code for the response properties.

```
overshoot = (max(y1) - y1(end))/y1(end) * 100
tpeak=t(find(y1==max(y1)))
idx=find(y1>y1(end)*1.05 | y1 < y1(end)*0.95)
tsettling=t(idx(end)+1)
idx=find(y1>=0.1*y1(end))
tdelay=t(idx(1))
idx1=find(y1>=0.1*y1(end))
idx2=find(y1>=0.9*y1(end))
trise=t(idx2(1))-t(idx(1))
```

3 State-space systems

3.1 Converting to state-space systems

There are three ways they can ask you to go to state-space systems: they give you a (set) of ODEs, they give you a block diagram or they give you a set of transfer functions.

3.1.1 Starting with a set of ODEs

Consider they give you the ODE

$$\begin{aligned} m\ddot{y} + b\dot{y} + ky &= F \\ J\ddot{\theta} + B\dot{\theta} + K\theta &= T \end{aligned}$$

In this case, we have two second order ODEs; this means that our states will become $x_1 = y$, $x_2 = \dot{y}$; $x_3 = \theta$ and $x_4 = \dot{\theta}$ (you defined states up until (so not including) the highest order derivative; thus we do not include something like $x_3 = \ddot{y}$ but stop at $x_2 = \dot{y}$). Our aim is to find equations for $\dot{x}_1$, $\dot{x}_2$, $\dot{x}_3$ and $\dot{x}_4$. For $\dot{x}_1$ and $\dot{x}_3$, the equations are easy[1]

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_3 &= x_4 \end{aligned}$$

For $\dot{x}_2$ and $\dot{x}_4$, the equations are found by plugging in $\ddot{y} = \dot{x}_2$, $\dot{y} = x_2$, $y = x_1$, $\ddot{\theta} = \dot{x}_3$, $\dot{\theta} = x_3$ and $\theta = x_4$. In the first equation, we then get

$$\begin{aligned} m\dot{x}_2 + bx_2 + kx_1 &= F \\ \dot{x}_2 &= -\frac{b}{m}x_2 - \frac{k}{m}x_1 + \frac{F}{m} \end{aligned}$$

For the second equation, we get

$$\begin{aligned} J\dot{x}_4 + Bx_4 + Kx_3 &= T \\ \dot{x}_4 &= -\frac{B}{J}x_4 - \frac{K}{J}x_3 + \frac{T}{J} \end{aligned}$$

Thus, we have the system of equations

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{k}{m}x_1 - \frac{b}{m}x_2 + \frac{F}{m} \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= -\frac{K}{J}x_3 - \frac{B}{J}x_4 + \frac{T}{J} \end{aligned}$$

and thus our state equation becomes

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + B\mathbf{u} \\ \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k}{m} & -\frac{b}{m} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{K}{J} & -\frac{B}{J} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{1}{m} & 0 \\ 0 & 0 \\ 0 & \frac{1}{J} \end{bmatrix} \begin{bmatrix} F \\ T \end{bmatrix} \end{aligned}$$

[1] These are the states that do not correspond to the highest-order derivative that is included as state ($x_2 = \dot{y}$ is the state that corresponds to $\dot{y}$, which is the highest order derivative of y that we have made a state). Same can be said about x_4 but then with respect to θ.

Thus, now we have found our A and B matrices. Suppose we are interested in the outputs $\dot{\theta}$, y and $r = y - \theta$ (in that order). To find our C and D matrices then, we have to relate them to the states and inputs. For $\dot{\theta}$ and y, these relations are straightforward: $\dot{\theta} = x_4$ and $y = x_1$. For r, we get $r = y - \theta = x_1 - x_3$, thus we get the following:

$$\begin{aligned}
\dot{\theta} &= x_4 \\
y &= x_1 \\
r &= x_1 - x_3 \\
\mathbf{y} &= C\mathbf{x} + D\mathbf{u} \\
\begin{bmatrix} \dot{\theta} \\ y \\ r \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} F & T \end{bmatrix}
\end{aligned}$$

This defines the C and D matrices. You can easily write them down in Matlab. Once you've written down, write

```
sys=ss(A,B,C,D)
```

to make Matlab realize it's a system so that you can do stuff with it.

3.1.2 *Converting block diagrams to state-space: append and connect*

The easiest way to convert block diagrams to state-systems is by use of `append` and `connect`. Consider the block diagram of figure 3.1.

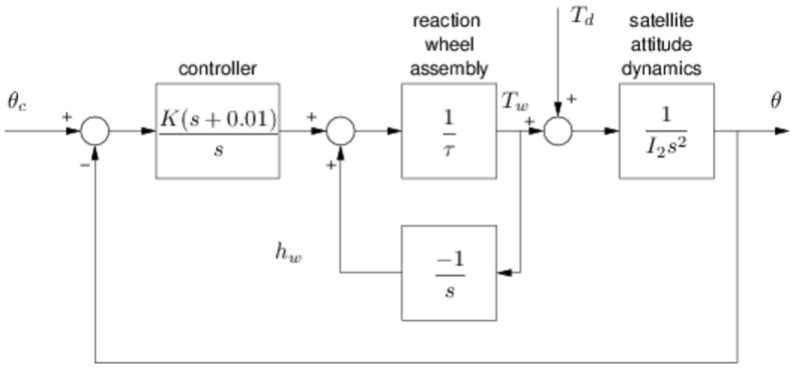


Figure 3.1: Block diagram.

The first step is to define everything as block as a subsystem with its own input and own output; see figure 3.2.

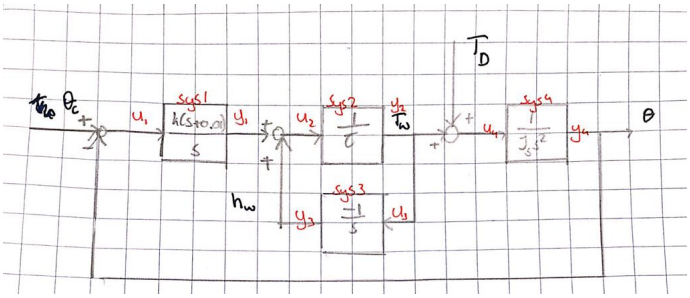


Figure 3.2: Inputs and outputs.

Then, you plug in all of these transfer functions into Matlab and use `append`:

```
s=tf('s')
h1=k*(s+0.01)/s
h2=1/t
h3=-1/s
h4=1/(is*s^2)
sys=append(h1,h2,h3,h4)
```

Then, you need to set up the Q matrix: we have the relations between inputs and outputs:

$$\begin{aligned} u_1 &= -y_4 \\ u_2 &= y_1 + y_3 \\ u_3 &= y_2 \\ u_4 &= y_2 \end{aligned}$$

so that

$$Q = \begin{bmatrix} 1 & -4 & 0 \\ 2 & 1 & 3 \\ 3 & 2 & 0 \\ 4 & 2 & 0 \end{bmatrix}$$

Furthermore, we have the inputs θ_c and T_d; θ_c is in front of the first block; T_d is front of the fourth block, thus we get `Inputs=[1 4]`. If we want the outputs θ, T_w and h_w, we see that θ is after the fourth block; T_w after the second block and h_w after the third block. Thus, `Outputs=[4 2 3]`.

If you define Q, Inputs and Outputs correctly, we then can create the state space model using

```
system=connect(sys, Q, Inputs, Outputs)
```

3.1.3 *Converting transfer functions to state-space*

They can also give you a set of transfer functions and ask you to convert them to a state-space system. Consider a system with 3 outputs and 2 inputs, with transfer functions

$$\begin{aligned} H_1(s) &= \frac{Y_1(s)}{U_1(s)} = \frac{s+5}{s\left(s^2+5s+4\right)} \\ H_2(s) &= \frac{Y_2(s)}{U_1(s)} = \frac{6s+1}{s+4} \\ H_3(s) &= \frac{Y_3(s)}{U_1(s)} = \frac{4s^2+1}{3s^3+2s} \\ H_4(s) &= \frac{Y_1(s)}{U_2(s)} = \frac{1}{s^2+5s+4} \\ H_5(s) &= \frac{Y_2(s)}{U_2(s)} = \frac{1}{s+4} \\ H_6(s) &= \frac{Y_3(s)}{U_2(s)} = \frac{4s^2+1}{3s^2+2} \end{aligned}$$

We can convert this to a state-space system by remembering that in Matlab, we use the following matrix:

$$\frac{Y(s)}{U(s)} = \begin{bmatrix} \frac{Y_1(s)}{U_1(s)} & \frac{Y_1(s)}{U_2(s)} & \cdots & \frac{Y_1(s)}{U_3(s)} \\ \frac{Y_2(s)}{U_1(s)} & \frac{Y_2(s)}{U_2(s)} & \cdots & \frac{Y_2(s)}{U_3(s)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{Y_m(s)}{U_1(s)} & \frac{Y_m(s)}{U_2(s)} & \cdots & \frac{Y_m(s)}{U_3(s)} \end{bmatrix}$$

Thus, in our case

$$\frac{Y(s)}{U(s)} = \begin{bmatrix} \frac{Y_1(s)}{U_1(s)} & \frac{Y_1(s)}{U_2(s)} \\ \frac{Y_2(s)}{U_1(s)} & \frac{Y_2(s)}{U_2(s)} \\ \frac{Y_3(s)}{U_1(s)} & \frac{Y_3(s)}{U_2(s)} \end{bmatrix} = \begin{bmatrix} H_1(s) & H_4(s) \\ H_2(s) & H_5(s) \\ H_3(s) & H_6(s) \end{bmatrix}$$

This means that in Matlab, you'd have to plug in the following:

```
H=[H1 H4;
   H2 H5;
   H3 H6];
sys=ss(H)
```

This will make the state space system with the correct inputs and outputs. Note that usually, they won't state it as explicitly as I did which transfer functions belong to which input and which output[2]

CONVERTING STUFF TO STATE-SPACE SYSTEMS

1. Starting with set of ODEs: if equation is of Nth order, introduce state variables up until $N-1$th order derivative. For the first $N-2$ equations, the equations are then simply $\dot{x}_1 = x_2$, $\dot{x}_2 = x_3$, etc. For the last equation, substitute all state variables into the equation and rearrange and solve for the last state. For the output equation, look how the outputs are related to the states and inputs. Write down the matrices in Matlab and use `sys=ss(A,B,C,D)` to make it a proper system.
2. Starting with a block diagram: define each block as a subsystem with its own input and output. Write the transfer functions in Matlab and put them all together using `sys=append(h1,...h2,h3,h4,h5)`. Then, find the matrix Q and write it down in Matlab. Then, check for each input in *front* of which block it occurs; this will form your `Input=[]`. Be sure to use the correct order of inputs. Finally, check for each output **after** which block it occurs; this will form your `Output=[]`. Again, use the correct order. Finally, put everything together with `system=...connect(sys, Q, Inputs, Outputs)`.

3.2 Determining responses

3.2.1 Response to initial conditions

Determining the response to initial conditions is straightforward. Note that the initial conditions are applied to the states of the system. Suppose we have a system with 4 states, 2 outputs and 3 inputs. The response to initial conditions can be determined as follows (the first line just sets up a random system of 4 states, 2 outputs and 3 inputs):

```
sys=rss(4,2,3)
t=0:0.1:20
x0=[2 3 1 2]
y=initial(sys,x0,t)
```

Note that `x0` has four entries as we have four states. Furthermore, as we have two outputs, `y` will have two columns. This means that if you need only the response of one output to the initial condition (for example to plot output #1 vs. time following the initial condition), you have to write `y(:,1)` or `y(:,2)`, depending on which output you want.

3.2.2 Response to step and impulse inputs

The response to step and impulse inputs are almost as easy as before: again, again we use the commands

```
y1=step(sys,t)
y2=impulse(sys,t)
```

However, once again affairs are complicated by the fact there are multiple inputs and multiple outputs. There are multiple ways of dealing with this.

[2] In the sense that for example they'd say, the inputs are the force F and the moment M (in that order) and the outputs elevator angle δ_e, rudder angle δ_r, and aileron deflection δ_a (again in that order); the transfer function between the elevator angle and force is this; between the elevator angle and moment it's this etc. etc. Just read carefully and you'll be fine.

For me, the easiest way to do is follows. Suppose we have X states, Y outputs and Z inputs. The thing is, `step()` and `impulse()` both produce multiple things: they first calculate the response when only the first input is a step input; then they calculate the response when only the second input is a step input, etc. It doesn't calculate it for when everything is a step input at once; it only calculates it for each individual input being a step input separately. This means that the response `y1`/`y2` is saved as a $N \times Y \times Z$ array, with N being the number of points in time that's being solved for (i.e. the length of the time vector). This means that if you for example only want to consider the response of the third output to a step input of the second input (for example to plot stuff), you have to use `y1(:,3,2)`. Note that if you have single output and/or single input, this still works: `y1(:,3,1)` will still work fine even if you have only one input (which means you'd only get a $N \times Y$ array). Even if you have single input and single output (which means that `y` is already a simple column vector that you could plot), `y(:,1,1)` doesn't break it so it's not a bad idea to just always use it.

3.2.3 *Response to arbitrary inputs*

The response to arbitrary inputs is similar to before; however, since we now often have multiple inputs, it is important to make sure that the input is defined for *all* inputs. Again, if the timevector contains N entries, and there are Z inputs, then you need to make sure you have `u`, the input matrix, be an $Z \times N$ matrix. Note that the response `y` will be a $N \times M$ matrix. If you want the response of the second output, you'll again have to write `y(:,2)`, etc. The code will look like

```
y=lsim(sym,u,t)
```

3.3 *Plotting responses*

Plotting responses is done exactly as during the plotting of transfer functions. However, you do need to take into account that you need to select the correct output, thus you need to select the correct columns based on what I told you before. For example, if we want to plot the response of the third output to a step input of the fourth input, we'd write:

```
t=0:0.01:20
y=step(sys,t)
clf()
plot(t, y(:,3,4)
```

3.4 *Time domain response criteria*

Same as before, just remember to select the correct columns. We now get (for the second output to third input):

Overshoot can be determined using the following line of code:

```
overshoot = (max(y1(:,2,3)) - y1(end,2,3))/y1(end,2,3) * 100
```

Peak time can be determined using:

```
tpeak=t(find(y1(:,2,3)==max(y1(:,2,3))))
```

Settling time can be determined using:

```
idx=find(y1(:,2,3)>y1(end,2,3)*1.05 | y1(:,2,3) < y1(end,2,3)*0.95)
tsettling=t(idx(end)+1)
```

Delay time can be determined using:

```
idx=find(y1(:,2,3)>=0.1*y1(end,2,3))
tdelay=t(idx(1))
```

Rise time can be determined using:

```
idx1=find(y1(:,2,3)>=0.1*y1(end,2,3))
idx2=find(y1(:,2,3)>=0.9*y1(end,2,3))
trise=t(idx2(1))-t(idx(1))
```

SUMMARY OF DETERMINING RESPONSES

If we have N time steps, X states, Y outputs and Z inputs:

1. Determine the response to initial conditions using `y=initial(sys,x0,t)`, where `x0` specifies the initial conditions of the states. This creates a $N \times Y$ matrix.
2. Determine the response to step and impulse inputs using `y=step(sys,t)` or `y=impulse(...sys,t)`. This creates a $N \times Y \times Z$ array.
3. Determine the response to arbitrary input using `y=lsim(y,u,t)`. Make sure `u` is a $Z \times N$ matrix.

For all above points, note that you need to select the correct columns every time if you want to use the responses (e.g. to plot it, or to find system properties). That means that for the first and third type of response, you for example need to write `y(:,2)` if you want the response of the second output. For the second type of response, you also need to select to which input you want the response to; i.e. if you want the response of the third output to a step input of the fourth input, you need to write `y(:,3,4)`.

1. Plotting is exactly the same as before, as long as you select the correct columns for the response.
2. Same goes for time domain response criteria.

3.5 Adding feedback to state-space systems and trivial affairs

3.5.1 Adding feedback

Adding feedback loops can be quite tricky, as we now have a feedback matrix to implement. In general, if we have Y outputs and Z inputs, then the feedback matrix will be of size $Z \times Y$. Then, if you for example are asked to compare the third output to the second input, with a gain of K in between, you make a matrix full of zeros of size $Z \times Y$, but with with the entry of the second row, third column equal to K. For example, if we have 4 outputs, 3 inputs, and we use feedback between the 2nd input and 1st output, we write

```
Kmatrix=[0 0 0 0;
         K 0 0 0;
         0 0 0 0];
system=feedback(sys,Kmatrix)
```

3.5.2 Poles and damping

Applicable to both transfer functions and state-space systems (didn't know where to put it before):

```
pole(H)
pole(sys)
```

will give the poles of a system. You can request even more information by asking for

```
damp(H)
damp(sys)
```

which also gives the damping ratio, natural frequency and time constant of each pole (time constant eing equal to T in $e^{-t/T}$.

SUMMARY OF FEEDBACK AND TRIVIAL AFFAIRS

1. With X states, Y outputs and Z inputs, the feedback gain matrix must be of size $Z \times Y$. If you are interested in the relation between the pth output and qth input, then this matrix will be full of zeros, except for the entry of qth row and pth column (which will have the gain K in there).
2. The commands `pole()` and especially `damp()` can give you valuable information about poles (and for `damp`, also about the damping coefficients, natural frequencies and time constants).

4 Design: root-loci

4.1 General characteristics of root-locus

Know these by heart:

- Let there be n poles and m zeros. Then there will be $q = n - m$ asymptotes, which will have angles

$$\phi_a = \frac{2q + 1}{n - m} \tag{4.1}$$

- m of the poles will move towards the zeros as $K \to \infty$, the remaining $n - m$ poles will move along the asymptotes.
- The root-locus can only exist (when counting from the right) between the first and second pole/zero, third and fourth pole/zero, fifth and sixth pole/zero etc.
- Once one of the poles moves into the right-half plane, the system becomes unstable.
- The root-locus is basically always symmetric w.r.t. the real axis.

4.2 Root-loci for transfer functions

Root-locus can be accessed by writing

```
rltool(H)
```

Once it's plotted, go to the other tab (it first show a tab with the response to a step input). You can vary the gain by clicking on one of the neon-pink points and moving it around. In the bottom right you can see the current location of it, and (if it's at a complex location), the associated natural frequency and damping coefficient. The corresponding gain can be found by clicking on C in the top-left window.

More advanced options are:

- If you want a certain damping ratio, right-click on the plot, go to 'design requirement', go to 'new', and go to 'Design requirement type: Damping ratio'. You can plug in the desired value. In the plot there are now two clear black lines; the white area has ζ larger than the requirement. The light yellow area has ζ smaller than the requirement.
- Although it is allowed to implement the gain, compute the closed-loop transfer function and calculate the settling time from there, another trick is to go to the other tab (the one that automatically showed up when opening root-lcous). Right-click on the plot, go to Characteristics and click Settling Time. Then, right-click again, go to Properties... (bottom of menu), go to Options and change Show settling time within to 5% (instead of 2%). Now go back to the plot and click on the blue dot, it'll show you the settling time.

4.3 Root-loci for state space systems

For state-space systems, not much is different. The thing that you need to remember is that you'll be designing the gain used between one input and one output. Thus, you need to select the correct transfer function for this. Implementing the gain has been explained in section 3.5.

SUMMARY OF ROOT-LOCUS

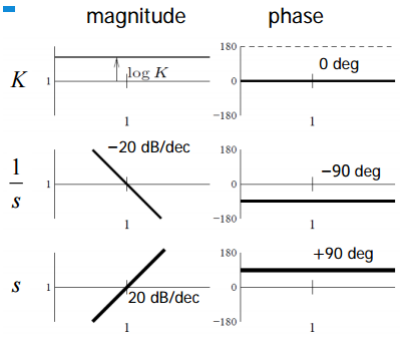
1. If there are n poles and m zeros, then m of the poles will go to the open-loop zeros (so they all get filled up with a pole). The other poles will move along asymptotes. The root-locus can only exist (when counting from the right) between the first and second pole/zero, third and fourth pole/zero, etc. Once one of the poles moves into the right-half plane, the system becomes unstable.

2. Access root-locus by writing `rltool(H)`. Move the neon-pink dots around to change the gain. The gain can be found from the left-top window by clicking C. Design-requirements can be easily put in, and settling time can also be easily calculated.
3. For state-equations, remember how to convert to the correct transfer function.

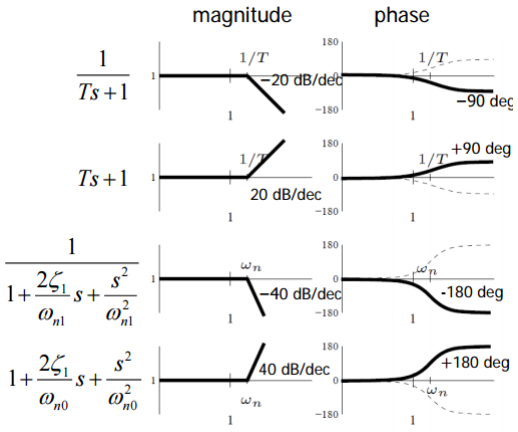
5 Design: bode

5.1 Shape of Bode plot

The Bode plot is obtained by substituting $j\omega$ in for each instance of s in the transfer function. Characteristics of various components are shown in figure 5.1.



(a) Gain and phase of various components. Note that for the second picture, if you e.g. have $1/s^2$ then you get a gain of -40 dB/dec for the magnitude; the phase would become -180 degree. The same holds for the third figure.



(b) Gain and phase of various components. Note that for all of them, the corner frequencies are $1/|T|$ and $1/|\omega_n|$. Furthermore, the dotted line is what happens when T or ω_n are negative.

Figure 5.1: Gain and phase of various components.

From this, important things can be said: first, for type-N systems[1]:

- A type-0 system starts at 0 slope in the magnitude plot, and at 0° in the phase plot.
- A type-1 system starts at -20 dB/dec slope in the magnitude plot, and at −90° in the phase plot.
- A type-2 system starts at -40 dB/dec slope in the magnitude plot, and at −180°.

A non-minimum phase system[2] on the other hand, will produce 90° of phase lag instead of phase lead compared to the situation where the zero would have been placed in the left-half plane.

Some important criteria are:

- −3 dB bandwidth: the lowest frequency at which the magnitude plot goes below -3 dB.
- −45° bandwidth: the lowest frequency at which the phase plot goes below -45°.

[1] A type-N system has N poles in the origin. For example,

$$H(s) = \frac{1}{s^2(s+3)}$$

is a type-II system.

$$H(s) = \frac{1}{s^2+4s+6}$$

is a type-0 system.

[2] A non-minimum phase system is a system which has at least one zero in the right-half plane.

5.2 Polar plot

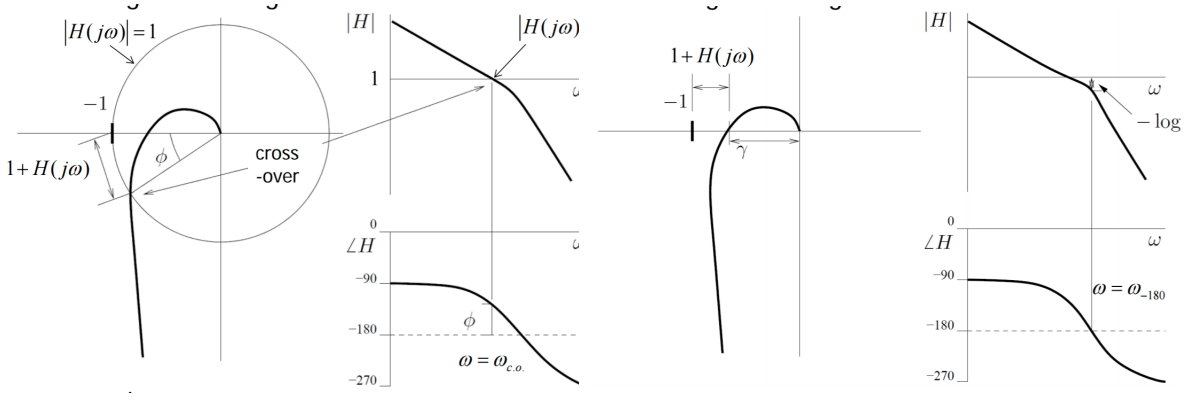
If a polar plot encloses the critical point (-1,0), then the system is unstable. If a polar plot does not enclose it, then the system is stable. Enclosing is the portion of the graph that if you walk from $\omega = 0$ to $\omega = \infty$ is located to the right of you. Alternatively, typically speaking, if a polar plot intersects the real axis to the left of (-1,0), it is unstable; if it intersects it to the right of it, it is stable.

Additionally, the type of the system has influence on the general shape:

- Type-0 systems start on the real axis, a finite distance away from the origin.
- Type-1 systems start on the imaginary axis, along an asymptote.
- Type-2 systems start on the real axis, an infinite distance away from the origin, along an asymptote.

Note that the values of $\omega = 0$ and $\omega = \infty$ are usually visible from the polar plot.

Phase and gain margin are shown in figure 5.2.



(a) Phase margin.

(b) Gain margin.

Figure 5.2: Phase and gain margin.

These can also be deduced from the Bode plot; see the right parts of figure 5.2.

5.3 Programming into Matlab

Bode plots and polar plots can be easily obtained. Use

```
bode(H)
margin(H)
grid on
```

To draw a Bode plot of H, with margins indicated. Polar plots are obtained using

```
nyquist(H)
```

this plots the polar plot both for negative and positive frequencies. You only have to bother with positive frequencies; this is the plot where the arrow is in the direction of the origin.

SUMMARY OF FREQUENCY ANALYSIS

1. In the Bode plot:
   - A type-0 system starts at 0 slope in the magnitude plot, and at 0° in the phase plot.
   - A type-1 system starts at -20 dB/dec slope in the magnitude plot, and at −90° in the phase plot.
   - A type-2 system starts at -40 dB/dec slope in the magnitude plot, and at −180°.
2. A non-minimum phase system has 90° lag instead of 90° lead for a zero in the left-half plane.
3. In the polar plot:
   - Type-0 systems start on the real axis, a finite distance away from the origin.

- Type-1 systems start on the imaginary axis, along an asymptote.
- Type-2 systems start on the real axis, an infinite distance away from the origin, along an asymptote.

4. Bode plots and margins are easily determined using `bode(H)` and `margin(H)`.
5. A polar plot is drawn using `nyquist(H)`; only consider the half of the plot where the direction of the arrow is towards the origin.
6. Especially nice stuff happens when you use `sisotool(H)`.

6 Other stuff that I don't know where to put

6.1 Laplace initial-value and final-value theorem

If you want to to calculate the initial-value or final-value of $f(t)$, use

$$f(0) = \lim_{s\to\infty} \{sF(s)\} \tag{6.1}$$

$$\lim_{t\to\infty} f(t) = \lim_{s\to 0} \{sF(s)\} \tag{6.2}$$

Here, $F(s)$ is the Laplace transform of the thing you're interested in; this could either be the output $Y(s)$ or the error, $E(s)$. In both cases, typically speaking you will not know $Y(s)$ or $E(s)$ itself; then, write it like

$$y(0) = \lim_{s\to\infty} \{sY(s)\} = \lim_{s\to\infty} \left\{ sU(s) \frac{Y(s)}{U(s)} \right\}$$

where $Y(s)/U(s)$ is the transfer the function (of course, you'd use $E(s)$ if you wanted the error).

If you want values of the derivative, use $s^2F(s)$ instead; 2nd order derivative would be $s^3F(s)$.

Note that $s = 0$ and $s = \infty$ can usually be determined from polar and bode plots.

- The position error is the steady-state error that occurs in response to a step input, $U(s) = 1/s$.
- The velocity error is the steady-state error that occurs in response to a ramp input, $U(s) = 1/s^2$.
- The acceleration error is the steady-state error that occurs in response to an acceleration input, $U(s) = 1/s^3$.

Furthermore:

- Type-0 systems have non-zero, but finite position errors, and infinite velocity and acceleration errors.
- Type-1 systems have zero position errors, finite but non-zero velocity errors, and infinite acceleration errors.
- Type-2 systems have zero position and velocity errors, and finite but non-zero acceleration errors.

Non-minimum phase systems have a zero in the right-half plane; the initial response will be in the opposite direction of what you'd expect.