

Introduction to Matlab

September 26, 2007
Control Systems – 520.353

Frequently Asked Questions about Matlab

Q: What is Matlab good for?

A: Matlab can solve any problem that can be expressed quantitatively.

Q: Matlab seems really slow. Is this really the best way for engineers to get things done?

Q: Big deal that Matlab can do these computations – I can do all this stuff in

C++/Java/PHP/Python/whatever. What's so special about Matlab?

A: It's true that Matlab has more overhead compared to some other programming environments, and it's also true that Matlab can't do anything that you can't do some other way. Matlab can, however, save you time because its extensive library of built-in functions and add-on toolboxes can help you write useful code a lot faster than you could in, say, C++. Matlab offers tremendous savings in development time in exchange for somewhat longer execution time.

Q: Why are we learning Matlab in Control Systems?

A: The Control Systems toolbox makes it really easy to play around with transfer functions, analyze systems, and do a lot of stuff related to what we're learning in class.

Getting Started

- Double-click on Matlab R2007a (on the desktop) to open it.
- Examine what you see in Matlab. Locate the Command Window, the Command History Window, and the Current Directory Window. Notice how you can access the Current Directory Window or the Workspace using the tab. Switch to the Workspace.

Commands and Basic Variables

- In the Command window, type: $x = 9$ and hit Enter.
- Observe what happened. The output from your command appears in the Command Window. The command you typed appears in the History Window. And an entry for x appears in the Workspace window, shown to be assigned the value 9.
- Try something different: type $x = 7;$ and hit Enter. Notice that the semicolon suppresses the output in the Command Window. The new command appears in the History Window, and the value of x changes to 7 in the Workspace.

Vectors and Matrices

- Matlab is short for Matrix Laboratory, named as such because it is optimized to perform operations on matrices quickly and efficiently. Let's explore some of Matlab's basic matrix capabilities.
- Create a row vector: type $rv = [1 \ 2 \ 3]$ and hit Enter.

- Create a column vector: type `cv = [4; 5; 6]` and hit Enter. Notice how these two commands produce different-looking results in both the Command Window and the Workspace.
- Multiply these vectors together: try `rv*cv` and `cv*rv`. Before you try this, predict what will happen. Do the results agree with what you predicted? Note what happens if you do a calculation but don't assign the result to any variable. Locate the `ans` variable in the Workspace.
- The `'` (apostrophe) operator transposes a vector or matrix. See what happens when you type `cv'` and `rv'`.
- Create a matrix: `M = [0 0 2; 0 2 0; 2 0 0]`
Notice how that appears in the Workspace.
- Multiply `cv2 = M*cv` and `rv2 = rv*M`. Predict what's going to happen. Is your prediction correct? What happens if you do `cv*M` or `M*rv`?
- The `.*` operator performs elementwise multiplication. Do that to the column vectors `cv` and `cv2`. Do you see how this is different from the `*` operator? Also try addition and subtraction: `cv + cv2` and `cv - cv2`.

Eigenstuff

- Create a matrix `A = [4 1 0; 3 2 8; 7 6 5]`.
- Figure out how Matlab comes up with eigenvectors and eigenvalues: type `help eigs`.
- Recall from linear algebra that if a matrix has a complete set of linearly independent eigenvectors, there exists a diagonal matrix D similar to the original matrix A , i.e. $D = P^{-1}AP$, where P is a matrix formed by the eigenvectors of A . Use the `eigs` command to come up with the D and P matrices for the matrix A that you just created, then verify that $D = P^{-1}AP$. (Hint: `inv(P)` is the command to find P^{-1} .)

Creating Plots

- Create an evenly distributed time vector: `t = 0:0.1:10;`
- Notice how this appears in the Workspace. Double-click on the `t` variable in the Workspace to see what it looks like. Close out the Array Editor when you're done.
- Create a vector `x1 = sin(t);`
- Plot `x1` versus time: `plot(t,x1)`
- Create another vector `x2 = exp(-2*t)*sin(t);`
- Why doesn't this command work? (Hint: think about vectors.) Fix it and plot `x2` versus time.

CHECKPOINT #1

Control Systems Toolbox: Transfer Functions

- The `tf` command creates a transfer function. Its usage is as follows: to create a transfer

function $H(s) = \frac{b_n s^n + b_{n-1} s^{n-1} + \dots + b_1 s + b_0}{a_m s^m + a_{m-1} s^{m-1} + \dots + a_1 s + a_0}$, type:

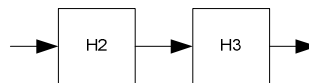
`H = tf([b(n) b(n-1) ... b(1) b(0)], [a(n) a(n-1) ... a(1) a(0)])`

- Using the `tf` command, create a transfer function $H_1(s) = \frac{1}{s^2 + 3s + 4}$.
- Plot the impulse response and the step response using the `impz` and `step` commands, respectively. Note the dotted line indicating the steady-state value of the responses.

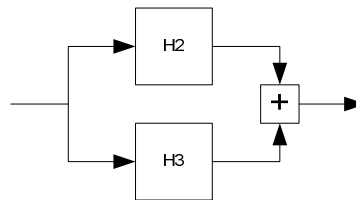
CHECKPOINT #2

Connecting Transfer Functions Together

- Define transfer functions $H_2(s) = \frac{1}{s-1}$ and $H_3(s) = \frac{s^2 - 6s + 5}{s^2 + 5s + 6}$. Use the `impz` command to check the stability of each.
- Use the `help` command to find out how to use the `series` command. Define $H_s(s)$ as the series connection of $H_2(s)$ and $H_3(s)$. Check the impulse response of $H_s(s)$. Why does this system appear to be stable? (If you don't know why, use the `why` command to find out.)



- Use the `help` command to find out how to use the `parallel` command. Define $H_p(s)$ as the parallel connection of $H_2(s)$ and $H_3(s)$. Check the impulse response of $H_p(s)$.



- Use the `help` command to find out how to use the `feedback` command. Define $H_F(s)$ as $H_2(s)$ and $H_3(s)$ connected in a negative feedback configuration, with $H_2(s)$ in the forward path and $H_3(s)$ in the feedback path. Check the impulse response of $H_F(s)$.

- If you have time, find a transfer function that you could put in the feedback path such that the resulting system would be stable. (If you don't have time, this would be a useful thing to do later.) Check the impulse response of the resulting system to see if you were right.

CHECKPOINT #3

Poles and Zeros

- Use the `pole` and `zero` commands to find the poles and zeros, respectively, of $H_S(s)$, $H_P(s)$, and $H_F(s)$. (Does this shed any light on why $H_S(s)$ appears to be stable? Is it internally stable?)
- Construct the transfer function $H_{osc}(s) = \frac{s+1}{(s^2+4)(s+6)} = \frac{s+1}{s^3+6s^2+4s+24}$. Use the `pzmap` command to represent graphically where the poles and zeros are.

More Fun with (In)Stability

- Plot the impulse response of $H_{osc}(s)$. Is this system stable?
- Find the response of $H_{osc}(s)$ to an input $U(s) = \frac{5}{s^2+4}$. (Hint: make a transfer function $U(s)$, then put $U(s)$ and $H_{osc}(s)$ in series and find the impulse response. Do you see why this works?) Using this example, explain what is bad about a marginally stable system.

CHECKPOINT #4

The End

Assignment: Homework #3, problem 3 asks you to use Simulink to solve a feedback problem. We're going to learn Simulink next week, and we'll revisit this problem then. This week, please do this problem using Matlab and the things we learned today about the Control Systems toolbox. Plot the step response of the system at each place in the diagram where there is a Scope. You should turn in two of these plots, each labeled with the transfer function from the input to the output. *This assignment, like the rest of the homework, should be done on your own.*