Matlab Example Script

Table of Contents

Introduction 1
Publishing Expectations 1
Attributions 1
Markup 2
Publishing 2
Plots 2
Linear Simulation 5

Introduction

- Author: Ethan Cuka
- Class: ESE 351
- Date: Created 9/4/2020, Last Edited 9/8/2020
- With contributions from: Dr. Jason Trobaugh

This document contains formatting and code that demonstrates the expectations for MATLAB work for this class. It also includes an example usage of the `lsim()` function that you may find useful for this course.

For more assistance in learning MATLAB, consider completing the MATLAB on-ramp or consulting the MATLAB documentation.

Publishing Expectations

All MATLAB assignments for this class should be submitted to canvas as *both* a matlab .m script *and* a published .pdf file and should meet the following guidelines:

- Clear, functional code, with occaisional comments to aid readability
- Plots clearly labeled with title and axis labels, as well as a legend if necessary
- Output of important statements not suppressed by semicolons. (Basically we need to be able to see from your pdf that your code did what it was supposed to without opening and running your script)
- Proper credit/attribution given to students and outside sources you used for the assignment
- Any custom functions written for the assignment included either at the bottom of the script or as separate scripts

Attributions

Any time you receive help on an assignment from another student or an outside source, make sure you provide proper attribution: list the names of any students who helped you, and the location of any sources you used, such as Stack Exchange or other help platforms.

One acceptable format for attribution at the start of your assignment is:

- Author: (Your name)
- Class: (Class name or number)
- Date: (Date of creation or last edit)
- Contributions from: (Names of anyone who helped you with the assignment)

You can also leave comments next to individual bits of code providing attribution for them:

```
% This code provided by John Doe via madeupwebsite.com
x = 0;
for i = 1:10
    x = x+1;
end
```

If you're not sure what needs to be attributed? When in doubt, cite it. Do your do diligence and attribute work in good faith, and you'll be fine.

Markup

MATLAB has a simple markup language which will be useful for making nice-looking published worksheets. Learning to use this language will make your work prettier and more readable, as well as fill you with a sense of accomplishment. You won't be graded on how pretty your work is as long as it's readable, but can you really assign a grade to beauty?

Check out the script this pdf is made from, 'examplescript.m', to see examples of how to use various aspects of MATLAB's markup language.

Check out MATLAB's documentation for more detail on MATLAB's markup language.

Publishing

There are two ways to publish your completed matlab script. The quickest is using the command line:

```
publish('examplescript.m', 'pdf')
```

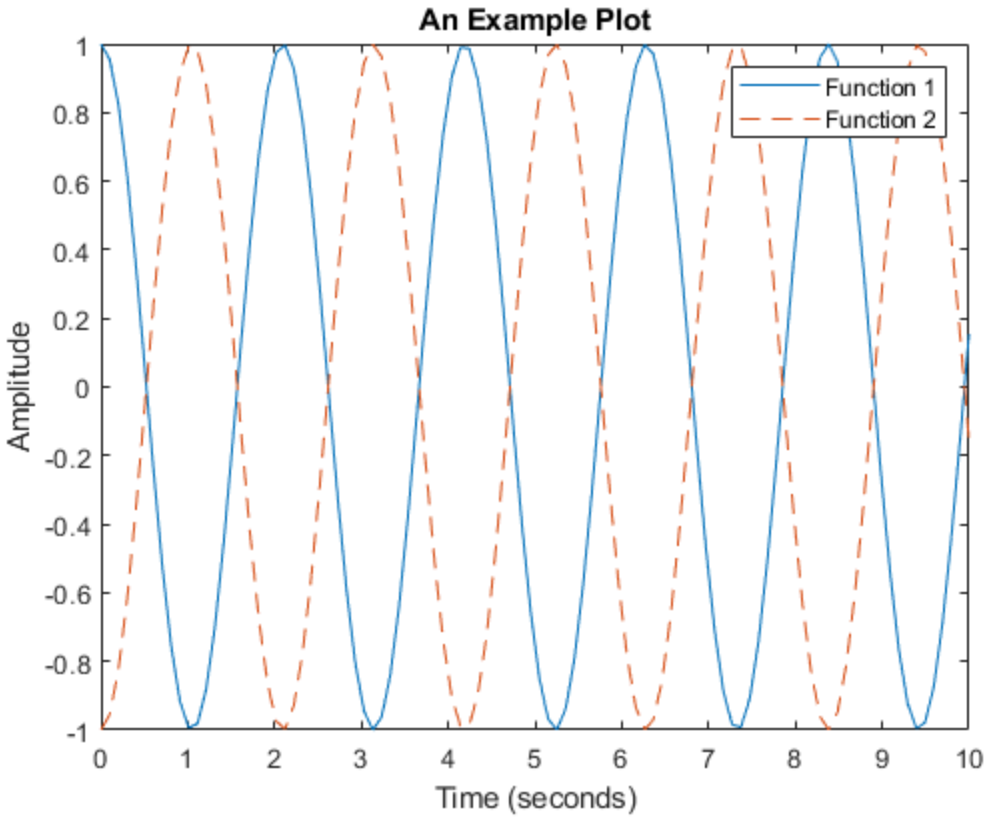
Alternatively, open the "publish" tab in the menu at the top of your MATLAB window. Use the drop-down arrow below the "publish" button to select "open publishing options" and change the "output file format" to "pdf." Then either click "publish" at the bottom of the window or press the "publish" button in the menu ribbon.

Plots

All plots created for this class should be labeled using a title, axis labels. If multiple functions are plotted at once, either include a legend or make use of the `subplot` function. Below are some example plots.

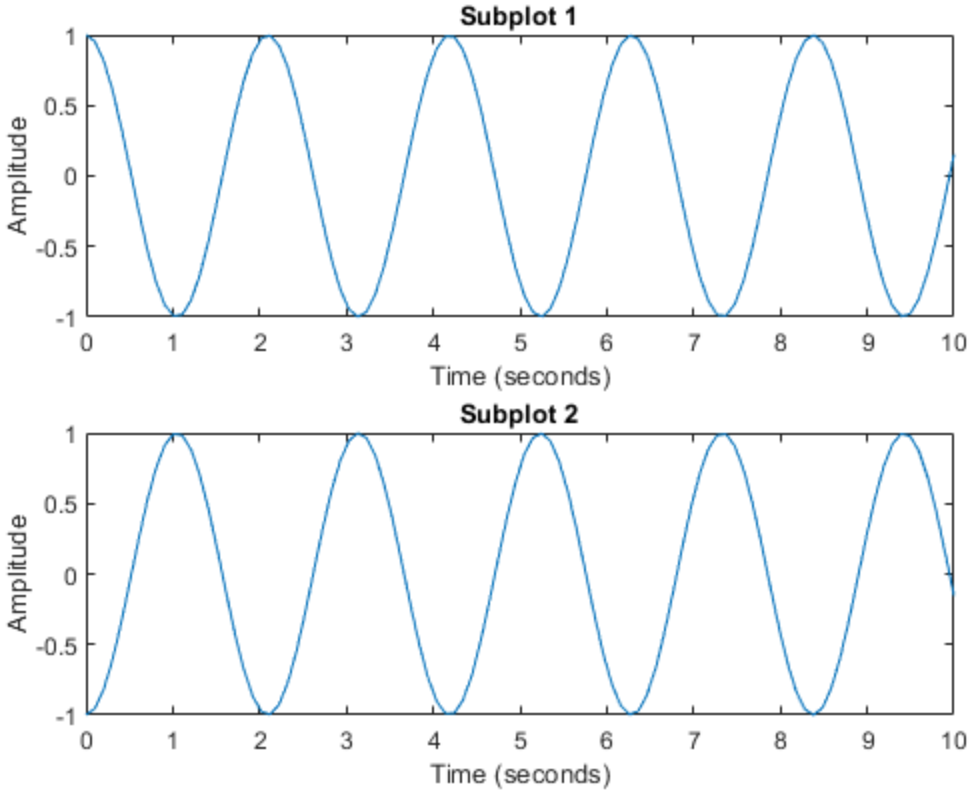
```
t = linspace(0,10,100); % Define time vector
y_1 = cos(3*t);         % Define some signals in terms of t
y_2 = cos(3*t+pi);

% You can name your figure window, but you don't have to!
figure("Name", "An Example Figure Window");

% You can add plot specifications to a plot via commas...
plot(t,y_1, '-'), title("An Example Plot");
```

```
% Or as a separate line
xlabel("Time (seconds)")
ylabel("Amplitude")

% The "hold on" function tells MATLAB not to overwrite the existing
% axis with a new one. You'll need it anytime you want to plot
% multiple functions on the same axis.
hold on;
plot(t,y_2, '--')

% Legend specification must go after all plots have been added
legend(["Function 1" "Function 2"])
% You don't necessarily need to toggle hold off when you're done
hold off;
```



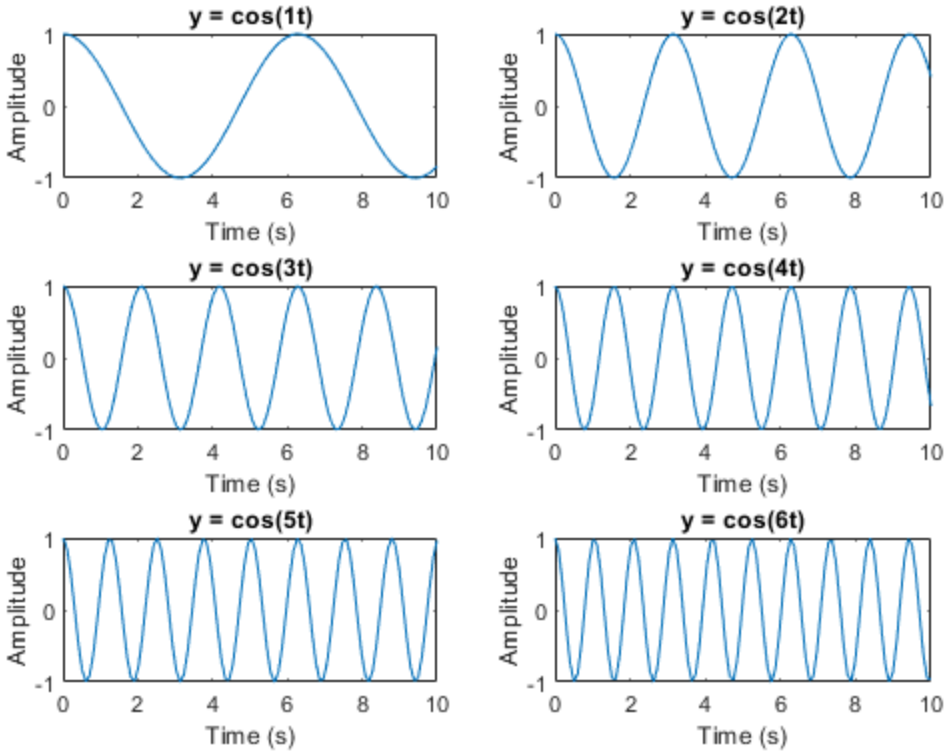
Below is an example of the usage of `subplot`. Subplot takes 3 arguments: a number of rows, a number of columns, and an index, respectively. It then creates a grid within the figure window with the specified number of rows and column. Each space on the grid becomes its own plot which can be created and edited by calling subplot again with the correct index.

```
figure("Name", "A Second Example Figure Window");
subplot(2,1,1), plot(t,y_1), title("Subplot 1"), xlabel("Time
 (seconds)"), ylabel("Amplitude")
subplot(2,1,2), plot(t,y_2), title("Subplot 2"), xlabel("Time
 (seconds)"), ylabel("Amplitude")
```



Note: You can call subplot in a "for" loop!

```
figure("Name", "A Third Example Figure Window");
for i = 1:6
    subplot(3,2,i)
    plot(t, cos(i*t))
    title("y = cos(" + i + "t)")
    xlabel("Time (s)")
    ylabel("Amplitude")
end
```



Linear Simulation

`lsim()` is a MATLAB function that is very useful for simulating the output of a continuous linear system to a specified input over a specified time period. Consider a linear, constant-coefficient differential equation of the form below:

$$a_1\frac{d^n y}{dt^n} + a_2\frac{d^{n-1} y}{dt^{n-1}} + ... + a_n\frac{dy}{dt} + a_{n+1}y = b_1\frac{d^m u}{dt^n} + b_2\frac{d^{m-1} u}{dt^{m-1}} + ... + b_m\frac{dm}{dt} + b_{m+1}u$$

In other words, a system in canonical form, in which all derivatives of the output are on one side and all derivatives of the input are on the other. An equation in this form can be simulated using `lsim` as follows:

```
lsim(b, a, u, t)
```

Where `b` is a vector of the coefficients on the input and `a` is a vector of the coefficients on the output. The coefficients must be in descending order, with coefficients for higher-order derivatives first.

For an example of `lsim` in action, consider a simple damped oscillator system modeled by the following differential equation:

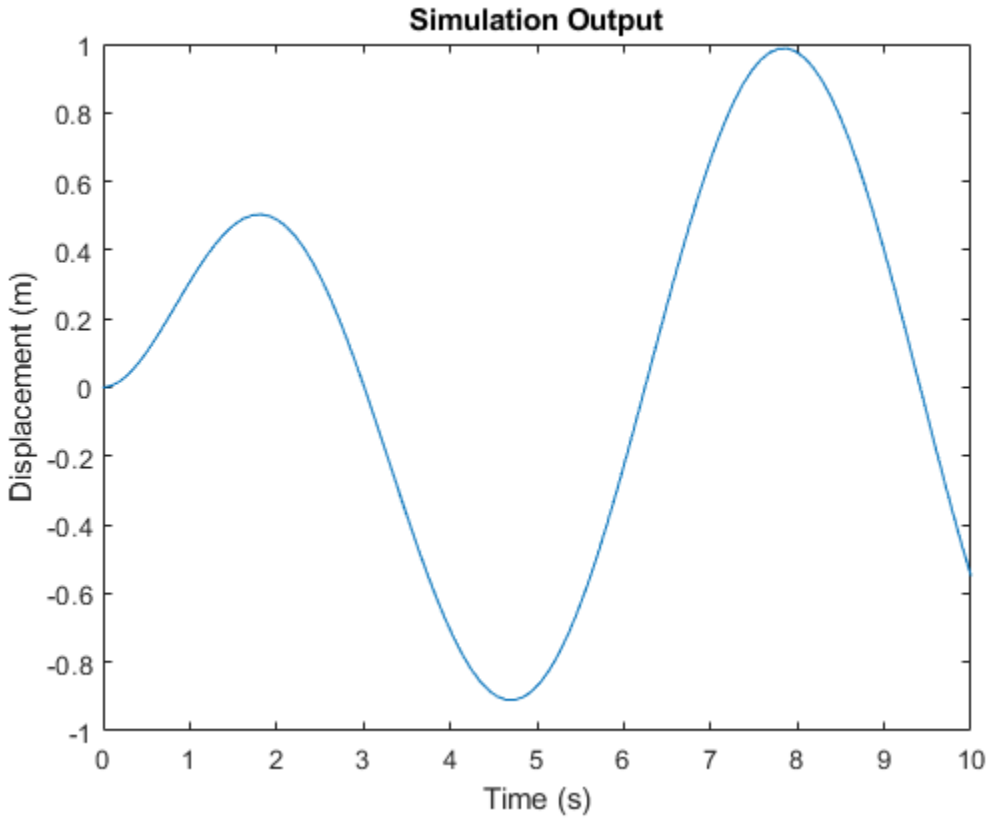
$$m\frac{d^2 y}{dt^2} + c\frac{dy}{dt} + ky = u$$

.

We can simulate this system's response to the input $u = cos(\omega t)$ using `lsim()`:

```
m = 1;      % Mass in kg
```

```
k = 1;      % Spring constant in N/m
c = 1;      % Damping coefficient in Ns/m
omega = 1;  % Frequency of input signal in rads/s

t = linspace(0,10,100);         % Define time vector
u = cos(omega*t);               % Define input vector

% This system has the coefficients {m, c, k} on the left hand side and
% the coefficient {1} on the right hand side.
y = lsim([1], [m c k], u, t);   % Simulate system

figure;                         % Plot results
plot(t, y), title("Simulation Output"), xlabel("Time (s)"),
 ylabel("Displacement (m)")
```



Published with MATLAB® R2020a