Worcester Polytechnic Institute

Rho Beta Epsilon

Robotics Engineering

C23 MATLAB Workshop

Created by:
Kalina Bonofiglio
Tuomas Pyorre
Zachary Rivernider
Brandon Simpson

January 26, 2023

Contents

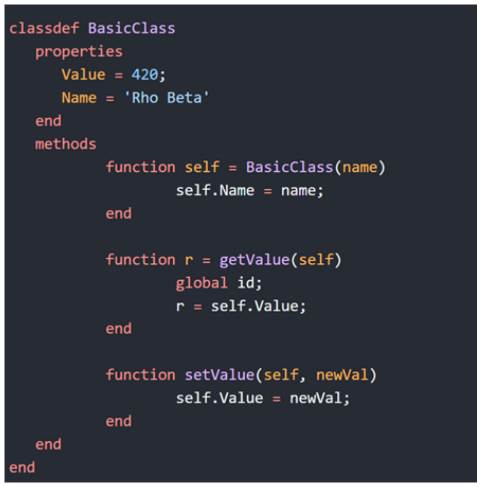
1 Class Familiarity

Did you know that MATLAB is more than just a fancy calculator? It's actually a powerful programming language and IDE (Integrated Development Environment).

1.1 Create a Class

Just like any powerful programming language, you can create classes within MATLAB (if you aren't super familiar with Object Oriented Programming (oop), refresh yourself here). To learn the basics on how to create a class in MATLAB read through this documentation from Mathworks. The easiest way to create a class in MATLAB is to:

1. Click the drop down under New
2. Click class



```
classdef BasicClass
   properties
      Value = 420;
      Name = 'Rho Beta'
   end
   methods
          function self = BasicClass(name)
                 self.Name = name;
          end

          function r = getValue(self)
                 global id;
                 r = self.Value;
          end

          function setValue(self, newVal)
                 self.Value = newVal;
          end
   end
end
```

Figure 1: Sample class generated for this Workshop

1.1.1 Procedure

1. Create a class using any of the methods described above.

1.2 Global Variables and Constructors

In order to create a class constructor you must instantiate variable in front of it, as is seen in the code snippet shown in Figure 1 with the variable "self". The usage of self in MATLAB (and also used in Python) is to denote the instance of the class. For example, the functions **getValue** and **setValue** are called upon the instance of the class, so the function definition must take self in as a parameter. **Not including this variable will cause errors in MATLAB.**

When variables are instantiated in the **properties** section of the class, they are global to the class. Each instance of the class will have their own set of these variables and they can be used throughout the class.

1.2.1 Procedure

1. Create an empty constructor for your class.
2. Create a global variable that is stored within the class

1.3 Create Methods

Because of the nature of OOP, you cannot always directly access the attributes of the object (instantiated instance of a class). MATLAB by default does not add this restriction.

Helpful Debugging Tip: MATLAB class files cannot be run like a normal script. In order to create an instance of your class, run your constructor in the terminal. For example, bc = BasicClass('Rob Test') will create an object called bc of the basic class. Then call your getter and setter on it to see them work.

1.3.1 Procedure

1. Create a getter function to return the global variable created in Section 1.2.1
2. Create a setter function to change the global variable created in Section 1.2.1
3. Verify that the class you made works and that it is changing and returning the proper value. (The helpful tip above may help with running the class)

2 Static Plots

2.1 Plot Position

When plotting in MATLAB, there are a few characteristics that are fundamental to every plot that is created. Shown in the image below is a side-by-side comparison of the same data with and without the proper attributes.

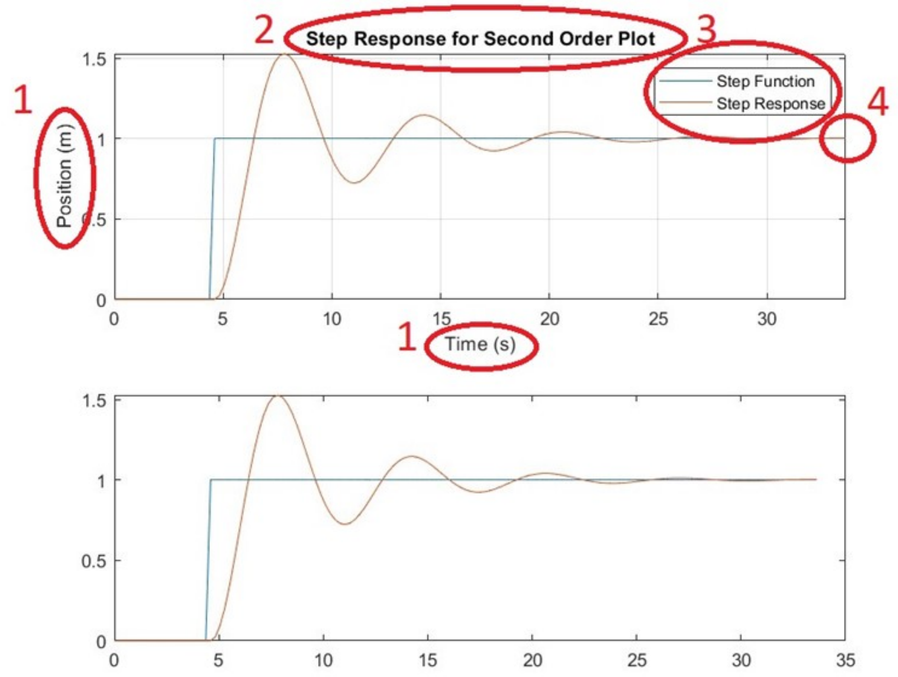


Figure 2: Side by side comparison for good plotting strategies.

1. **Axis titles:** The specific units of the plot are critical to fully understanding what the figure is trying to represent. These are appended to a plot through the x-axis and y-axis labeling functions.
2. **Main title:** All plots should have a descriptive title to explain the general purpose of the data it is representing. These can be added through the MATLAB title function.

3. **Legend:** When plots have multiple lines, a legend is incredibly important to assure that the reader is able to distinguish between the data. MATLAB's legend function makes adding a legend very straightforward.

4. **Axis limits on the x and y axis:** In general, MATLAB auto-sizes data, but that constraint does not always look the cleanest. These plots can be trimmed with the x-limits function and the y-limits function.

5. **Grid lines:** They can make the plot clearer and easier match a line to a value. Grid lines can be added with the grid function in MATLAB.

In addition to these graph characteristics, knowing how to add multiple lines to a plot is also a very valuable skill. In MATLAB, when a plot is created it generates a figure window and displays the new plot. In the case of the user calling another plot, the previous plot would be overwritten. To overlap data, the hold function must be enabled.

Helpful Debugging Tip: When working with classes the data does not get saved in the workspace. This could complicate troubleshooting and make finding errors difficult. Breakpoints can be placed in a script by clicking the line number for the desired stopping point. This will cause the number to have a red box and when the program is run, it will stop at that point. When it stops, the user can either continue until the next breakpoint, step through the function, or type directly into the terminal to see the current variable state. To remove the breakpoint, just click the red box and it will return to normal.

2.1.1 Procedure

1. Import the .mat file named "MATLABworkshopData.mat". In this file are three subsections of data that were prepared in advance.
   (a) **stepFunction** is a step function shifted to the right. A step function is at a value of zero that then steps to 1.
   (b) **stepResponse** outside of this workshop, a transfer function for a second order system was designed. This variable holds the output of that system when given a step input. Also known as the step response.
   (c) **time** This is the global time stamp that is universal to all of plots.

2. Create a new function in your class that takes in two inputs
   (a) The first input should be the data itself. Must combine all of the data from the previous step into one variable.
   (b) The second input should be the labels for the graph, formatted in whichever way that makes the most sense.
      i. One sample format would be to have a 3x3 matrix where the first row is the title, x axis, and y axis label, the remaining values of the first row represent the titles for the legend and the remaining elements are empty

3. Plot both the step response and the step function in response to time on the **same graph**.

4. Add all of the proper graph attributes mentioned above.

5. Experiment with breakpoints by placing a breakpoint after you're done plotting, but before you add your graph characteristics. Then, step through the program and observe all of the attributes as they appear on the plot.

2.1.2 Challenge

1. For an extra challenge, make the function plot "n" amount of data sets where n is the amount of data sets in the first input

2.2 Plot Derivatives

Now that we have a position plot, let's use this data to create a velocity plot. There are a few ways to calculate the discrete derivative of data-sets in MATLAB. One of the most common ways is to use MATLAB's built in gradient function. This function takes in a single array of data and calculates the difference between each element and the next. Recall that a derivative is yx. You'll need to calculate the gradient of both the y vector and the x vector, and then divide them appropriately. If you don't want to use a built-in function, you can also create this code on your own by taking the change and x and y for each element of your array and dividing them.

Once you have calculated the discrete derivative of position, you can plot it in the same fashion as Section 2.1. Do this and visually verify that you have calculated the derivative correctly. (The velocity plot should show the slope of the position plot).

2.2.1 Procedure

1. Create a new function in your class that takes in two inputs that are similar to the previous step.
   (a) The first input is the original data that will be used to calculate the derivative
   (b) The second input is once again the graph characteristics. It would be beneficial to keep the same input as the previous step
2. Calculate the derivative of both the step response and the step function in reference to time
3. Plot the derivative of each data set in reference to time
4. Add the proper attributes mentioned in Section 2.1
5. Have the function return the derivative of the first input. The output should mirror the input where one section is the unchanged time, and two derivatives of the input data set.

2.2.2 Challenge

1. For an extra challenge, make the function find the derivative and plot "n" amount of data sets where n is the amount of data sets in the first input. The same challenge as the previous step, but with derivatives.

2.3 Position and Derivative Subplots

Instead of having two separate plots for position and velocity, we want to display both the original function and the derivative of the function in the same figure as two separate graphs. This is called creating subplots. You can think of a figure (the window that MATLAB pops up whenever you tell it to make a graph) as a blank canvas. The subplot() function simply tells MATLAB how to size and place your graph on this blank canvas. The most common form of the subplot function is subplot(m,n,p), where m is the number of rows of graphs in your plot, n is the number of columns of graphs in your plot, and p is the index of the plot. Indices increase from top to bottom and then left to right. So if you have a 2x2 subplot the first index will be the top left, the second index will be the bottom left, the third index will be the top right, and the fourth index will be the bottom right.

For example, to create a subplot with a single column and two rows, add subplot(2,1,1) above your first plot instruction, and subplot (2,1,2) above your second plot instruction.

2.3.1 Procedure

1. Create a new function in your class that takes in three inputs.
   (a) The first input should be the original data formatted the same way as the previous steps
   (b) The second input should be the axis and title information for each subplot.
   (c) The third input should be a single row or column vector that has the labels for the data set descriptions (the information you want expressed in the legend)

2. Using the position and derivative functions made earlier, create a subplot with the original position data, first order derivative, and second order derivative.
   (a) Take advantage of the derivative function returning the derivative of the input data
3. Make sure all plots have the proper attributes mentioned in section 2.1
   (a) This may require reformatting the input data so that it follows the naming structure of the previous functions

2.3.2 Challenge

1. Extend this function to also exist for "n" amount of data sets
2. Manipulate the function so that for each subplot mentioned in the second plot, it will create that many derivatives. In other words, expand the function to create "k" of derivatives and place them all in a subplot where "k" is an arbitrary value.

3 Tex Interpreter (Optional)

If you ever need to use a fraction, other equation notation, or a Greek letter in a graph axis or title, you can use MATLAB's built-in Tex interpreter to convert Tex symbols and equations into displayable text. For example, if you would like your x-axis label to be the character Lambda, you can write xlabel("Here is lambda: \lambda"), and it will automatically convert this into the actual greek letter lambda and print ("Here is λ"). As shown with the link, this built-in interpreter requires no extra work and can even superscript, subscript, bold, etc.

3.1 Procedure

1. Experiment with the various options and adding characters
2. Duplicate the format of the fraction shown in Figure 3 using superscripts and subscripts.
   (a) This is an extra layer of detail that is not required, but elevates plots and helps them standout.

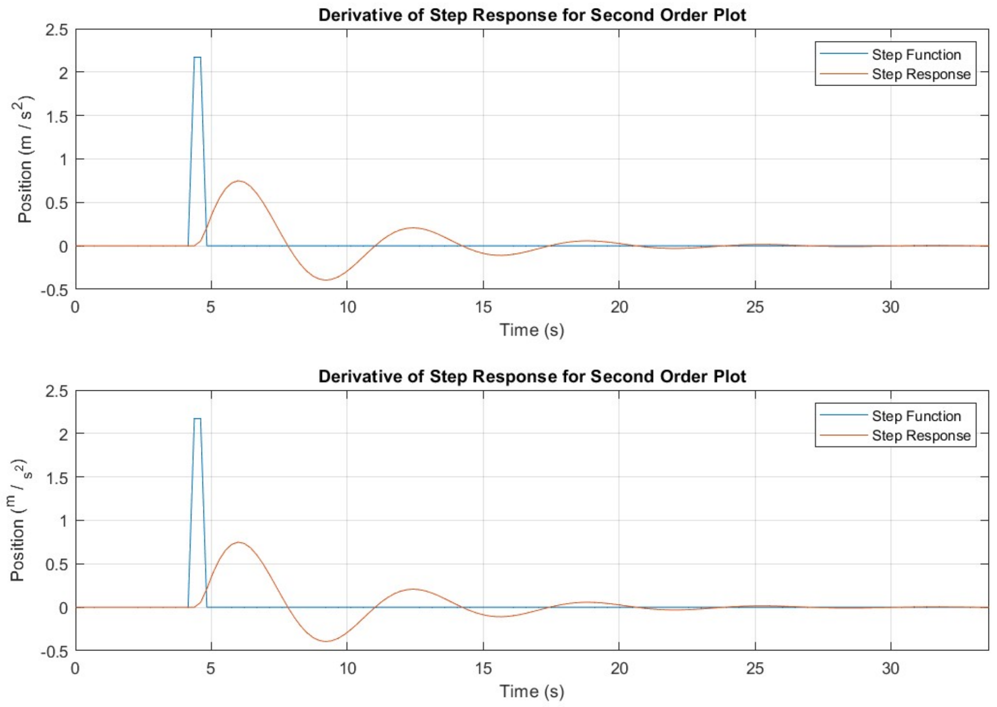


Figure 3: Side by side view of Tex and Non-Tex commands on the derivative plot

4 Live Plot

For this section we will be creating the same data as in Section 2, except instead of a static image we will be actively updating the image. So, we need to start using for loops to iterate through our data, and store the data with some method to plot it. We want to update our plots with new data while keeping the old data, and keep the axis' stable and not constantly creating new plots.

On top of this, as the for loops compute very quickly, we also need to slow the plot down a little bit. We will be using the pause(t)(t is in seconds) operator, just so we can view the plot updating live. You can play with the tuning of t to see what refresh rate looks good for the live update. If you want to see an example of what this plot should look like, click here.

We can reuse a lot from what we learn in Section 2, we now just need to store data, which we have a few methods for. First one is that we make a matrix of zeros that is the size of the data and the timestamps included in the data. The second one, is that we initialize an empty matrix [], and with each iteration we update our data, by saying plotData = [plotData, data(:, timestamp)]. The second method may be more computationally expensive to run, although it makes the plotting a bit easier, as then your animation won't have a line from the end of the current plot to (0, 0). Now that we know how to store the data, we can plot it by iterating through it.

For our final step, we can also take video of the frames in the plots. For this, we need to know a few commands. First, we need to start writing the video with v = VideoWriter('name of video.avi'). Afterwards we open the video for recording with open(v). Then, we can save the frames of each plot using f = getframe(gfc), which simply stores the current stage of the plot as a frame. Now we write it to our video using writeVideo(v, f). Finally, after we have gone through the plot, we can end the recording with close(v).

4.1 Procedure

1. Create a new function in your class that takes three inputs and plots the data.
    (a) The first input is the data, up to what timestamp we are on in the for loop.
    (b) The second input is once again the graph characteristics.
    (c) The third input will be used for setting the x-limit, so we need to give it
2. Now create the for loop in your main script, and have it iterate through the data while storing it by using one of the methods mentioned(or your own method).
3. Make sure the axis limits do not change while plotting, and that all the plotting is done in one figure.

4.2 Challenge

1. Take a video of the live plot in Matlab, and save it as an .avi file.
2. Add in four more seconds to the end of the video of the static plot, to make the video more viewable.