Brian Williams
AUP Final Report
Dr. Kent Lundberg

Educational Java Applet for Linear System Responses

Abstract

This paper describes an educational web applet for students studying linear systems and their frequency and time responses.

Introduction

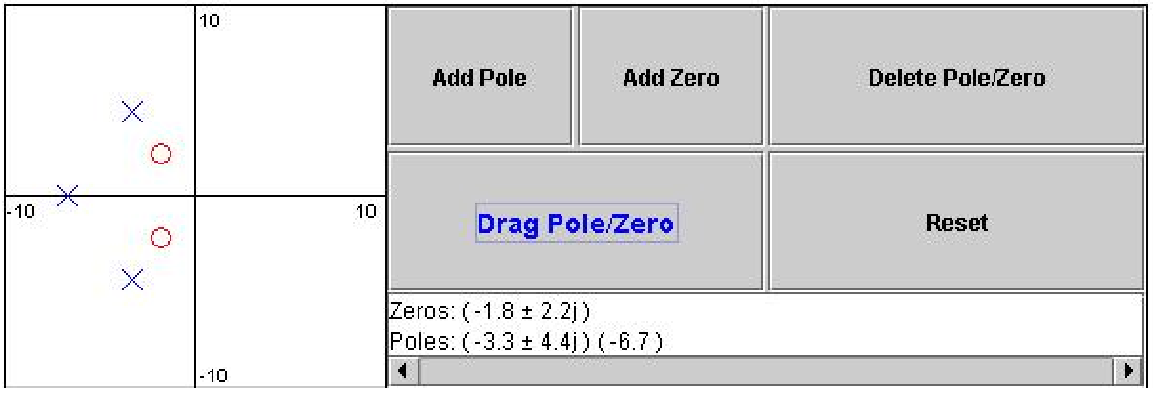
The applet described in this paper is designed for students learning about linear systems and feedback controls, specifically interactions with the s-plane and the resulting system responses. The four responses currently displayed are Bode plots, Nichols plots, Nyquist diagrams and step responses. These plots are created and modified as the student manipulates a small graph area representing the s-plane. Poles and zeros can be added, deleted and independently moved on the s-plane, with the resulting plots changing dynamically as the state of the s-plane changes. This allows students to visually connect changes with poles or zeros with the response of the system.

The output frame itself only displays one response at a time, but the student can switch between responses without affecting the current state of the s-plane. This feature allows students to see the relationships between and one system and the different plots of frequency and time responses.

The applet is written entirely in Java and a student can load it with only the student an up-to-date version of Sun's free JVM. Matlab, Mathematica, and Maple all provide more detailed tools to see these same responses, but this applet can be loaded on any machine for free and has no learning curve.

How to Use the Applet

There are three sections of the applet the student interacts with: the s-plane, the output graph selection tabs, and the "mode" of the applet. The s-plane is where poles and zeros are added, deleted, and dragged. A click in the s-plane will do one of those three actions, depending on the mode of the applet. The mode is set by the buttons to the right of the s-plane and stays highlighted in blue. Figure 1 shows the s-plane and the buttons to set the mode.



Zeros: (-1.8 ± 2.2j)
Poles: (-3.3 ± 4.4j) (-6.7)

Figure 1 - S-Plane, Current State and Mode Buttons

In Figure 1, complex pairs of poles and zeros have been added along with a single pole on the real axis, and the current mode is “dragging”. In dragging mode, the student can drag a pole or zero around by clicking and holding down the left mouse button near any feature. The display of the current state of the s-plane will change once per pixel, and the output graphs will also change as the mouse moves.

The applet treats a complex pair of poles or zeros as one entity, so when one of a pair is added, dragged, or moved, the conjugate pole/zero is also affected. Also, when a pole or zero is on the axis, it is treated as a single real pole/zero, not as a pair. Multiple poles or zeros can occupy the same coordinate by adding or dragging additional poles to that location. They will only look like one pole, but the text list will show each one independently.

The applet also treats any pole or zero added or dragged within 0.5 of the real axis as though it was on the axis. Even though a pole/zero is being dragged will snap to the axis when it comes within that tolerance, it is still actively being moved and will “un-snap” if the mouse moves away from the axis.

To view the resulting graphs, the student can select one of the four tabs in the middle of the applet. Each tab will bring the corresponding graph to view, with the initial plot representing the output from the current state of the s-plane. Figure 2 shows the layout of the tabs.

Figure 2 - Selection Tabs for Output Graphs

The output plots themselves are not interactive, they simply plot their respective responses based on the state of the s-plane.

Mathematics Used in the Applet

The four plots are created independently using approximations to the real shape of the graphs. Each approximation is nearly exact and the difference will not be noticeable except in a few select cases. The Bode plot, Nichols plot, and Nyquist plot all use the same frequency data generated with each change to the s-plane. Pseudo-code for the calculations is in Appendix A.

Frequency Response

The numerical method the applet uses begins by selecting 400 representative frequencies between 0.01 and 100, evenly spaced along a log scale. Since the breakpoints from a pole or zero must fall between 0.1 and 10, that particular range of frequencies will capture all the interesting parts of the plots. For each frequency, it calculates the magnitude and phase by combining the effects of each pole or zero in the s-plane.

The frequencies, magnitudes and phases are stored in separate arrays, with the magnitude and phases at location n in each array corresponding to the value at the same location n in the frequency array.

There is one current issue with the implementation of the frequency response. Often, the resonant frequency does not fall on one of the selected 400 points, so the full value of the peak is not always displayed. The work-around consists of changing the point closest to the resonant frequency to be directly on that frequency before calculating the magnitude and phase. Then when the graph is drawn, it will display the peak magnitude. This is not yet implemented, but could be in a future update to the applet.

Step Response

The numerical method for calculating the step response consists of calculating the polynomials of the transfer function from the poles and zeros then using a fourth-order Runge-Kutta algorithm to approximate the step response.

Calculating a polynomial from a set of either poles or zeros is straightforward multiplication of terms, with the result stored in an array. The value at any particular index in the array is the coefficient for the term of the same power. Complex conjugate poles or zeros are multiplied as a second-order polynomial to eliminate complications from imaginary numbers. Special information is also kept when a pole or zero is at the origin and is used to shift the values in the array after the polynomial is calculated without figuring in the points at the origin.

With the two polynomials, the Runge-Kutta algorithm can calculate the initial conditions and step response. The gain is also calculated using the lowest non-zero coefficients of the transfer function polynomials. The gain is used to adjust the matrices used in the algorithm so the steady-state value of the response is one.

When there are more zeros than poles in the system, the graph displays an error message to let the user know of this condition. The graph also displays the current gain of the system below the x-axis.

Description of Applet Structure in Java

The design of the applet can be split into three major units: the input section, the output graphs, and the data structure holding system responses. The input section consists of the s-plane graph, the set of buttons, and data structures to hold the state of the s-plane (poles and zeros). The output graphs section contains a panel for each type of plot, each one having a graph title, axis labels, and the actual plot. The data structure for frequency and step responses, the LinearSystemHolder class, is the layer separating the two other sections.

LinearSystemHolder

The LinearSystemHolder class holds information about one or more LinearSystem objects. LinearSystem objects hold a frequency response, a step response, and, if supplied, the poles and zeros of one linear system. LinearSystem objects are not created directly, but by the `addSystem()` method in the LinearSystemHolder.

A system can be added in multiple ways. The first way is to pass the poles, zeros, and a range of frequencies to the LinearSystemHolder. Then, the LSH creates a new LinearSystem with that information. Internally, the LinearSystem calculates the frequency and step responses of the system represented by the poles and zeros. A LinearSystemHolder can also accept the frequency and step response data directly, and just store it in a LinearSystem for later retrieval. The LinearSystemHolder only allows for deleting all systems, not specific single systems, since it does not necessarily keep the systems in any particular order.

When the applet is started, it creates a single LinearSystemHolder. With each change to the s-plane, the applet deletes all the current systems and passes the LSH the new set of poles and zeros to create a new LinearSystem. See Figure 3 for how the applet interacts with the LinearSystemHolder.

The LinearSystemHolder only returns a set of all frequency responses or a set of all step responses, one from each LinearSystem it contains. The extraction will be explained in the next section about the output graphs.

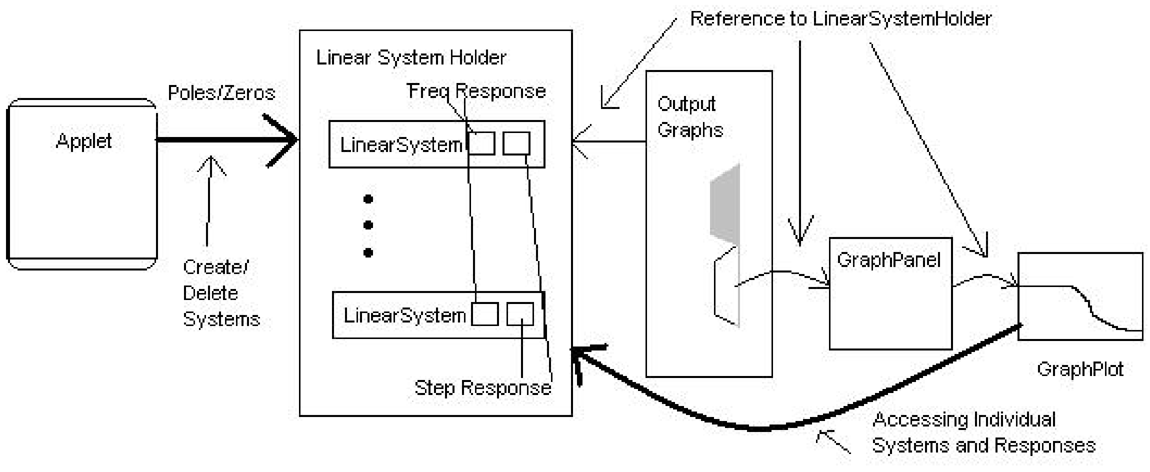


Figure 3 - LinearSystemHolder dividing input from output

Output Graphs

The output graph panel is a Swing JTabbedPane. The panel contains a tab for each graph in the applet and methods for updating the actively displayed graph.

Each panel in the output graphs is an extension of the abstract class GraphPanel. A GraphPanel is a JPanel with a specific format for the title, axis label, and graph plot. The graph plot within the GraphPanel is an instance of the abstract class GraphPlot. The GraphPlot class defines the actual graph, gridlines, colors, size and labels to be used in drawing each graph.

Both GraphPanel and GraphPlot create layers, allowing for easy addition or subtraction of new types of output graphs. By using these standardized classes, a new implementation of GraphPanel just needs to define the labels, axes, title, size, etc., and a GraphPlot to draw the graph.

The applet passes a reference to the LSH to the output graph panel, which passes it on to the actively selected GraphPanel, which then passes it to the GraphPlot to extract the necessary information to draw the graph. Each new GraphPlot needs to access either the frequency or step response data within the LinearSystemHolder to draw the new graph.

S-Plane Graph and Buttons

The input section is split into three sub-sections, the s-plane graph, the text listing of poles and zeros, and the buttons.

The s-plane graph is a JPanel that implements MouseListener and MouseMotionListener to react to the student using the applet. The action taken by the s-plane graph depends on the mode of the applet. When in add or delete mode, the panel only reacts to single clicks on the graph. When in dragging mode, the student must click near a pole or zero then hold down the mouse button to move the pole or zero on the graph.

The s-plane updates the state of the applet by deleting poles/zeros and adding poles/zeros. The dragging feature works by deleting the last location of the pole/zero and adding a new pole/zero at the new location. Once an update has finished, the s-plane signals to the applet that it should now redraw the output graphs

The buttons are JButtons and only change the mode of the applet or reset the applet. Clicking to change a mode calls `setMode()` in the applet and changes the mode to the setting for each button. Resetting the applet deletes all the poles and zeros and tells the applet to redraw the s-plane and output graphs.

Future Possible Additions to the Applet

The applet has a few areas where it has been designed for extension.

One potential feature is the ability to save and graph the data from multiple systems at once. The LinearSystemHolder is capable of holding many systems and GraphPlot requests all the frequency or step responses from the LinearSystemHolder, if more than one is stored. I tested that the GraphPlot can graph more than one line at a time, and I have even defined a set of colors to cycle through when the LinearSystemHolder returns more than one set of response data.

At this point, when a change is made to the s-plane, the LinearSystemHolder is emptied and one new LinearSystem is added. However, a possible feature could allow students to “save” a specific LinearSystem and continue to draw that system while updating another LinearSystem to represent any changes to the s-plane since that state was saved. This could allow better visual comparisons between two states of the s-plane.

Another possible addition to the applet is a root-locus plot. The calculations to draw the lines are fairly complicated, but the addition of one more tab and setting up the panel containing the plot would be simple. The frequency response class also stores the poles and zeros of a system (if provided at creation), so the root-locus plot could access that data. The only difficult part would be calculating all the lines for drawing the plot itself.

Javadoc Documentation (Available Online)

The current host of the applet is: http://web.mit.edu/6.302/www/pz/. The documentation is linked from that page and is also included in the source JAR file.

Appendix A – Frequency data generation pseudo-code

```
// Step 1 – Create a list of frequencies from 10^-2 to 10^2
decades = 4; // log(High frequency) – log(low frequency)
for (100 values per decade*decades+1) {
    // Create 100 sample frequencies per decade, evenly spaced on a log scale
    frequencies[i] = 10^-2 * 10^(i/100)
}

// Step 2 – Calculate Magnitude at each frequency using s-plane state
magnitudes[401] // One for each frequency point
for (each frequency) {
    jw = frequency[i]*j;  // Set numerical value of j*omega
    for (each pole and zero)  {
        temp = 0.0;
        // Sum effects from each pole and zero using log values
        if (pole at origin)
            temp = temp + log(1/(jw));
        if (zero at origin)
            temp = temp – log(1/(jw));
        if (pole not at origin)
            temp = temp – log((jw/pole) + 1);
        if (zero not at origin)
            temp = temp + log((jw/zero) +1);
    }
    magnitudes[i] = temp;  // magnitude[i] corresponds to frequency[i]
}

// Step 3 – Calculate Phase at each frequency using s-plane state
phases[401] // One for each frequency point
for (each frequency) {
    jw = frequency[i]*j;  // Set numerical value of j*omega
    for (each pole and zero)  {
        temp = 0.0;  // Sum effects from each pole and zero
        if (pole at origin)
            temp = temp – pi/2;
        if (zero at origin)
            temp = temp + pi/2;
        if (pole not at origin)
            temp = temp – arctan(pole.imaginary/pole.real);
        if (zero not at origin)
            temp = temp + arctan(zero.imaginary/zero.real);
    }
    phases[i] = temp;  // phases[i] corresponds to frequency[i]
}
```