Spectrum Analysis Using Java

In partial fulfillment of the requirements for

EE 4012 Senior Design

By:

Janna M. Shaffer
shaffer@isip.msstate.edu

Instructor:

Dr. Bert Nail

Department of Electrical & Computer Engineering
Mississippi State University
Mississippi State, MS 39762
Spring Semester 1997

Table of Contents

Spectrum Analysis Using Java

Janna M. Shaffer

EE 4012 -- Senior Design Project
Department of Electrical and Computer Engineering
Mississippi State University
Mississippi State, Mississippi 39762
shaffer@isip.msstate.edu

ABSTRACT

Signal spectrum analysis is considered to be a difficult and unintuitive concept by most students of Electrical Engineering. We have developed a graphical learning tool to assist in the explanation and visualization of spectral analysis concepts (such as frequency response) for an educational environment. The Java-based graphical user interface (GUI) for this tool has been refined using human factors research to minimize the learning curve for the novice user. The object-oriented design and platform-independence provided by Java make this tool widely portable and easily accessible to students and provide them with a powerful learning mechanism for a complex subject.

1. INTRODUCTION

Spectrum Analysis is not an intuitive process. One must go through the mathematical process of the Fourier Transform to understand the frequency response of a signal. It would be much more useful if a student could see the frequency response of a signal rather than trying to visualize it from abstract mathematical equations. Thus a spectrum analysis visualization package would be a much needed educational tool.

This need for visualization of a concept and Java's graphical environment make Java a good programming language to develop such a tool in. Portability and accessibility also make Java a good

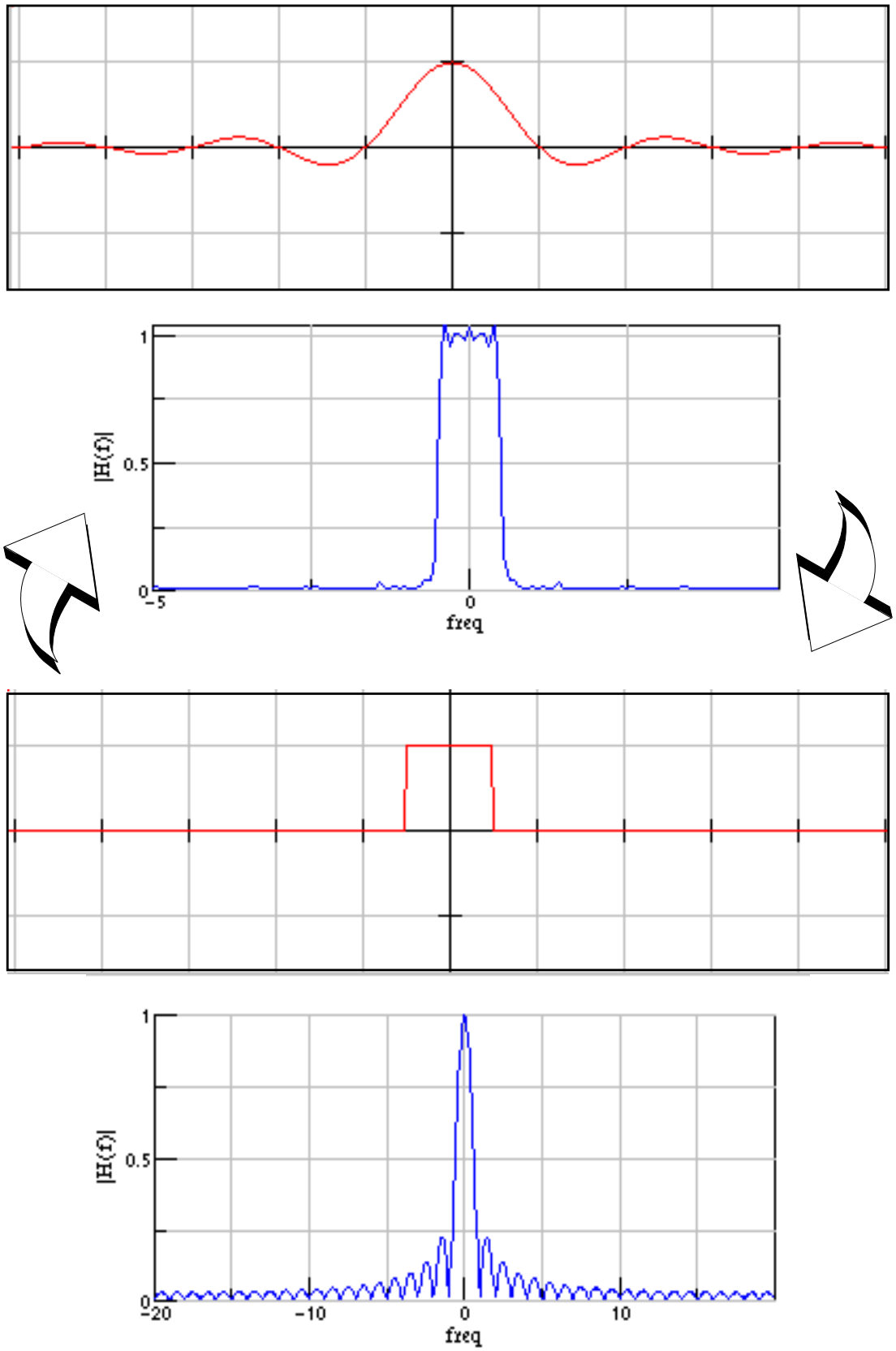

Figure 1. Duality as Shown Through the Spectrum Analysis Tool

choice. With this use of this Spectrum Analysis tool, the frequency response of a signal should become an easier concept for students to understand

2. MOTIVATION

The main objective for this spectrum analysis tool is education. Its purpose is to help clarify concepts that are often hard for students to grasp such as the frequency response of a signal. Because this tool is written in Java, it is easily accessible to students. The applet can be run in the web browser itself or downloaded and run in an applet viewer. An example of one of the concepts that can be learned from this tool is in Figure 1. This figure shows that the magnitude response of a unit pulse is a sinc function and that the magnitude response of a sinc function is a unit pulse. This illustrates the concept of duality.

3. DESCRIPTION

The spectrum analysis tool was created using Java. Its use will be to clarify concepts of magnitude response, phase response, and windowing of a signal. The tool has a graphical user interface for ease of use. The input signal area of the tool can be directly drawn on or a signal selected from a predefined list may be chosen to plot on this area. The window function area has the same setup as does the input signal area. There is a control bar that contains components to control the interactive drawing of input signals. On this control bar is the button which will execute the analysis of the input signal and window function if there is one defined. There is also a control bar that contains the predefined list of input signals and user defined graphing parameters. The tool also contains a graph for the magnitude response and phase response of the input signal. This graph is zoomable for grater resolution of the frequency response. All of these components work together to implement a software tool that helps students learn the concept of

frequency response.

4. DESIGN AND IMPLEMENTATION

The design process was taken one obstacle at a time. The first step was designing the graphical user interface (GUI). The next step was to decide on the basic functionality for input and output to the applet. After this major step was accomplished, details of the software structure were worked out and implemented. When each of these steps were executed, a working software tool resulted.

4.1. Evaluation of the Problem

Before any code was written, an idea of what would be needed for input and output, and how this would be displayed had to be considered. A canvas for input signals would be necessary as to specify the input to the system. Also a canvas for the window function, magnitude response, and phase response would be needed. It could also be concluded that a control bar would be needed to control the input/ output of the applet. Once the basics for the applet were determined, a preliminary layout could evolve.

4.2. Graphical User Interface (GUI)

Java's built in graphical interface classes were utilized to design the GUI for this tool. The actual GUI for this tool can be seen in Figure 2. The layout of the GUI was divided into input, output, and the window function. The input area for this tool resides in the top half of the applet. The control bars controlling the drawing and other parameters for the input surround it. The lower half of the applet contains the window function, phase response, and magnitude response. The magnitude and phase response lie below the input canvas as to keep with a basic flow from input to output. The magnitude response area lies above the phase response area for that is the usual practice

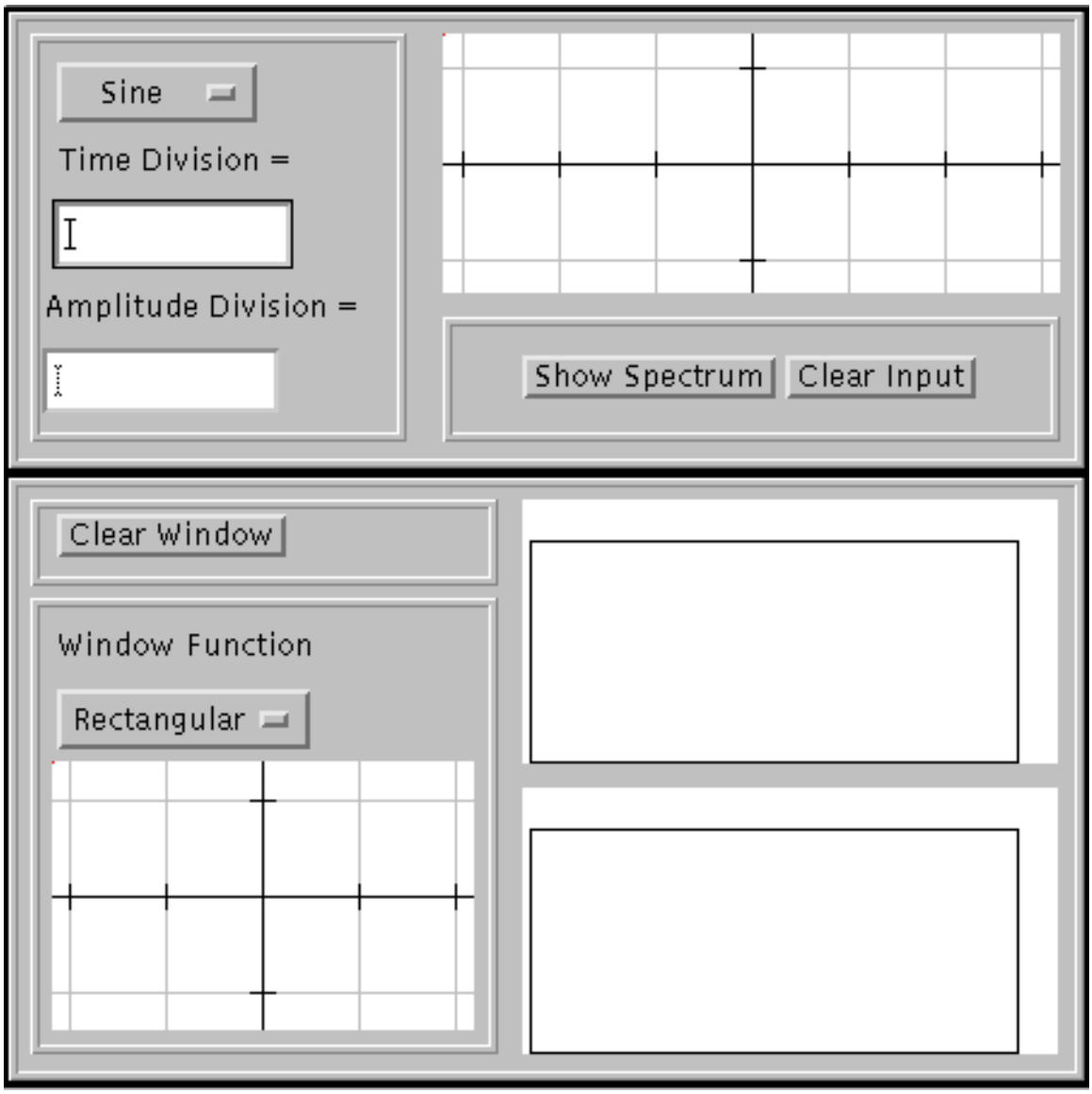

Figure 2. Graphical User Interface

for displaying frequency response graphs. With the GUI completed, the details of how each area would work independently and also how they would interconnect could be decided upon.

The graphics classes themselves are subclassed off basic Java components. The two main components used were the canvas and the panel. From the Panel class, a framed panel was subclassed to draw a three-dimensional border around

the panel. From the framed panel, a sub-panel was subclassed to set the layout and basic colors for the canvas. The input panel, output panel, window function panels, and the input control bar were subclassed from the sub-panel. The canvas class in Java was the other class widely used throughout the code. From the Canvas class was subclassed a drawing canvas. This drawing canvas allowed for drawing directly onto the canvas. It also provided functionality for drawing the grid and setting parameters for the graph. From the drawing canvas class were derived the input drawing canvas class and the window function class. Each class further specialized the drawing canvas by adding the functionality to choose default signals to plot based on the need for that particular class.

4.3. Input Classes

The input class is the first of the two main sections in the structure of this code. It contains instances of the input drawing canvas, the control bar, and the drawing control bar. The input drawing canvas is a canvas that allows the user to draw directly on it. It has two main axes: the horizontal axis represents time while the vertical axis represents amplitude. The two control bars surround the input drawing canvas. The control bar contains a pull-down list of predefined signals that can be plotted onto the input drawing canvas. It also contains two input areas that control the time division and amplitude division for the input drawing canvas. The drawing control bar contains two buttons. One button clears the input drawing canvas. The other button signifies that the input is ready for analysis.

4.4. Output Classes

The second of the two main sections in the structure of this code is the output class. The output class contains instances of the magnitude response graph, the phase

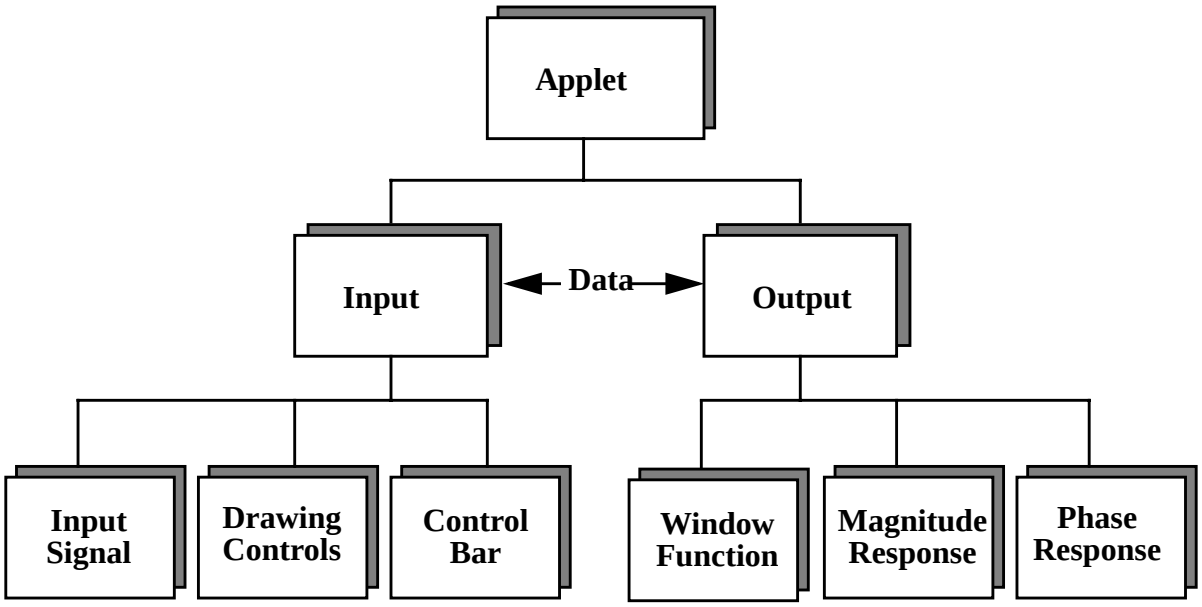

Figure 3. Software Structure

response graph, and the window function canvas. The window function canvas area also includes a button that clears the drawing area. The window function is included in the output class for visual purposes only, although it is actually an input characteristic. The magnitude and phase response graphs are entirely controlled by the output class.

4.5. Data Class

Before the data class was created, there was no way for communication between the input and output classes. The data class provided this bridge between the input and output classes. This class is passed in to all classes of input and output. It contains an array for the input values, flags, and functions that manipulate the data. The flags are the primary form of communication between input and output. When a certain task has been performed somewhere else in the code, the appropriate flag is set to alert the other sections of this change. The data class allowed for more efficient

communication between classes.

4.6. Threading

After the initial layout of the software was complete, details of the interworkings of the software had to be worked out. The threading of classes played an important role in the functionality of the software. By using threads in conjuction with the data class flags, a class can constantly check to see if it needs to update or perform a particular task. For example, the output class checks to see if the flag that signifies that input data is ready for analysis is set. When this flag is set, the output then performs the FFT analysis. The beauty of this the output class being threaded is that while the output is being calculated, the rest of the classes can go on and do their regular tasks, thus their is no dead time during analysis. Several classes work in a manner similar to this, such as the input drawing canvas and the window function.

4.7. Bringing It All Together

With the GUI in place, the class structure determined, and threading implemented, the last step in the design process was to make sure that each step was integrated correctly and the output of the tool was correct. Integrating the steps was not a difficult task because of the forethought in laying out the software design. Input could talk to the output and window function by means of threads which utilized the data class. Implementing the actual code to perform the analysis was also integrated swiftly. With the initial design in place, the next step in the design process would be testing.

5. TESTING AND VALIDATION

This program was developed on the Unix operating system. The majority of the testing was done using the Unix version of Netscape Navigator Gold 3.0 and an appletviewer. On this platform, the majority of this tool's features worked properly. The

main problem with this Spectrum Analysis tool is the phase response. To date, there is a problem outputting correct phase. This problem can and will be fixed in future versions of the software. The problem stems from the input data starting at negative time rather than at time zero. The other significant problem came from a problem with Java. GUI components appear different depending upon where the applet is run. The GUI looked different in Netscape than it did in the appletviewer. The main difference was how the panels were sized and aligned. Follow-up testing was performed using the Windows 95 version of Netscape Navigator 3.0. The same problems that occurred on the Unix platform were also present on the Windows platform.

6. CONCLUSION

This paper describes the functionality and design process of a Spectrum Analysis tool written in Java. This tool should provide a learning mechanism for students trying to understand the often unintuitive concept of spectrum analysis.. This design, however, is not in its final stage. There is still some functionality that must be added. The design process will continue in a circular fashion while features are being added in subsequent versions of this tool.

REFERENCES

1 Flanagan, David. *Java in a Nutshell*. Sebastopol, California: O'Reilly and Associates, Inc., 1996.

2 Naughton, Patrick and Herbert Schildt. *Java: The Complete Reference*. Berkeley, California: Osborne McGraw-Hill, 1997.

3 Proakis, John G. and Dimitris G. Manolakis. *Digital Signal Processing*, 3rd ed. Upper Saddle River, New Jersey: Prentice Hall, 1996.

4 Ziemer, Rodger E., William H. Tranter,

and D. Ronald Fanin. *Signals and Systems: Continuous and Discrete*, 3rd ed. New York: Macmillan Publishing Company, 1993.

APPENDIX A: SOURCE CODE DOCUMENTATION

file: Spectrum.java

class Spectrum extends Applet

Spectrum is the instance of the applet itself.

public void init()

init is called when applet Spectrum begins. Sets layout of the applet and adds an input and output panel to the applet.

file: FramedPanel.java

class FramedPanel extends Panel

Draws 3-D frame around a panel.
Hierarchy: Panel->FramedPanel

public Insets insets()

Insets ensures that no Component is placed on top of the frame. Overrides the Insets method of the Panel class.

public void paint(Graphics g)

paint draws the frame at this panel's edges.

file: SubPanel.java

class SubPanel extends FramedPanel

SubPanel is a FramedPanel with the layout manager of GridBagLayout.
Hierarchy: Panel->FramedPanel->SubPanel

SubPanel()

Constructor that sets size, background properties, and layout manager.

public boolean constrain(Container container, Component component,

constrain creates the constraints of a component in a container and then adds the component to the container. it is based on code from "Java in a Nutshell" by David Flanagan. Published by O'Reilly and Associates incorporated.

file: InputPanel.java

class InputPanel extends SubPanel

InputPanel creates a basic canvas for which a drawing canvas, a control bar for the input parameters, and a control bar for drawing are added.
Class hierarchy: Panel->FramedPanel->SubPanel>InputPanel

Imputable(Data d)

Constructor sets size, background properties.

void add_components()

add_components adds a control panel, a drawing controls panel, and a canvas to the input panel.

file: OutputPanel.java

class OutputPanel extends SubPanel implements Runnable

OutputPanel creates area for phase and magnitude response, and window function panels to be added to the applet. Controls the execution of the FFT.

OutputPanel(Data d)

Constructor sets size, background properties, layout manager, and adds components. Initializes thread for running the output.

void add_components()

add_components adds a phase canvas, magnitude canvas, and a place holder for the controls and window function to the output panel.

void control_layout()

control_layout adds the window function panel and a control bar to a panel

void place_controls()

place_controls places individual controls on the control panel.

public void run()

run is the body of the thread. It will update the display area when necessary. The necessity is based on the condition of the calculate_flag.

boolean sr_init(double sr_wr_d[], double sr_wi_d[])

sr_init creates lookup tables for sin and cosine terms.

boolean Srfft(double[] output_a, double[] input_a)

Srfft is used to compute the real split-radix FFT.
This code is based on the code by G. A. Sitton of the Rice University. The theory behind the implementation closely follows the description of the split-radix algorithm in: J. G. Proakis, D. G. Manolakis, "Digital Signal Processing —Principles, Algorithms, and Applications" 2nd Edition, Macmillan Publishing Company, New York, pp. 727-730, 1992.

file: FunctPanel.java

class FunctPanel extends SubPanel

Creates area for a window function drawing canvas and its controls to be placed. Allows for user interaction with the window function panel.

FunctPanel(Data d)

Constructor calls parent constructor and creates instance of the window function drawing panel and passes it the data class to write to.

void ClearDrawingArea()

ClearDrawingArea clears the window function drawing area.

public boolean action(Event e, Object arg)

action handles any user event.

void add_components()

add_components adds a control panel and a canvas to the input panel.

file: Data.java

class Data

Data contains data in array format for the input and output values also contain flags to be checked before and after performing FFT analysis.

Data()

Constructor initializes size of FFT depending on size of DrawingCanvas.

void InitializeWindowFunction()

InitializeWindowFuction writes all ones into the window function so as not to affect input if no window is applied.

void ApplyWindow()

Apply_Window multiplies the input_data by the window_function.

void time_shift()
time_shift shifts starting point of the fft window to time=0.

file: DrawingCanvas.java

class DrawingCanvas extends Canvas implements Runnable
Drawing Canvas adds interactive drawing capabilities.

DrawingCanvas(Data d)
DrawingCanvas()
Constructor contructs a canvas for drawing waveforms onto the canvas.

public boolean update_screen() {
update_screen sets the update_flag causing a repaint of the canvas.

void DetermineDimensions()
DetermineDimensions determines the dimensions of the canvas.

public void run()
run is the body of the thread. It will update the display area when necessary. The necessity is based on the condition of the update_flag.

public boolean handleEvent(Event e) {
handleEvent controls the handling of the mouse on the screen during drawing.

public void paint(Graphics g)
paint displays a grid on the canvas and enables drawing directly onto the canvas using lines.

void fillArray()
fillArray fills the array that contains pixel data with the pixels of the curve drawn.

public void DrawGrid(Graphics g)
DrawGrid draws default grid onto the canvas: includes main axes, grid map, and tick marks.

void DrawWaveform(int tempArray[])
DrawWaveform plots pixel values in the pixel array to the canvas.

void WriteData(int curveArray[])
WriteData write actual curve data to data class so fft can be performed set flag after data written.

void SetTimeDivision(double new_time_div)
SetTimeDivision sets time division from user input.

void SetAmplitudeDivision(double new_amp_div)
SetAmplitudeDivision sets amplitude division from user input.

void ClearDrawingArea() {
ClearDrawingArea removes drawing elements from the canvas but leaves the grid.

file: WindowFunction.java

class WindowFunction extends DrawingCanvas {
WindowFunction creates an interactive drawing canvas for window functions.
Hierarchy: Canvas->DrawingCanvas->WindowFunction

WindowFunction(Data d) {
Constructor contructs a canvas for drawing waveforms.

void drawRectangular()
drawRectanglular draws rectangular window to drawing canvas.

void drawHanning()
drawHanning draws Hanning window to drawing canvas.

void drawBartlett()
drawBartlett draws Bartlett window to drawing canvas.

file: InputDrawingCanvas.java

class InputDrawingCanvas extends DrawingCanvas
InputDrawingCanvas allows for interactive drawing on a canvas and also specifies specific input signal that can be plotted.
Hierarchy: Canvas->DrawingCanvas->InputDrawingCanvas

InputDrawingCanvas(Data d)
Constructor constructs a canvas for drawing.

void drawSine()
drawSine draws sine wave to drawing canvas.

void drawCosine()
drawCosine draws cosine wave to drawing canvas.

void drawSinc()
drawSinc draws sinc wave to drawing canvas.

void drawUnit()
drawUnit draws unit pulse function centered about the origin.

void drawTriangle()
drawTriangle draws triangle function centered about the origin.

file: DrawControls.java

class DrawControls extends SubPanel implements Runnable
Threaded control bar that controls drawing capabilities of input.
Hierarchy: Panel->FramedPanel->SubPanel->DrawControls

DrawControls(DrawingCanvas wp, Data d)
Constructor that initializes the local variables with instances of the InputDrawingCanvas and FFT data.

public void run()
run is the body of the thread. It will update the display area when necessary. The necessity is based on the condition of the update_flag.

public boolean action(Event e, Object arg)
action handles any action performed by the user.

void place_drawing_controls()
place_drawing_controls positions individual controls on the drawing control panel.

file: ControlBar.java

class ControlBar extends SubPanel
ControlBar controls the InputDrawingCanvas. It allows for the user to pick a waveform and set amplitude and time divisions.

ControlBar(InputDrawingCanvas wp)
Constructor calls parent constructor.

public boolean action(Event e, Object arg)
action handles any user input

public boolean keyUp(Event e, int key)
keyUp determines the key released if the key is an arrow key then it refreshes the output screen.

void place_controls()(
place_controls positions individual controls on the Control Panel.

file: MagResponse.java

public class MagResponse extends G2Dint implements Runnable
MagResponse implements a canvas which holds the magnitude response of a signal. The class it is subclassed from, G2Dint, adds zooming capabilites to the Graph class.
Hierarchy: Graph->G2Dint->MagResponse

public MagResponse(Data fft_input)
Constructor constructs a magnitude response graph area.

public boolean draw_response()
draw_response draws the magnitude response of the signal on the screen.

public boolean update_screen()
update_screen sets the update_flag causing a repaint of the canvas.

public void run()
run is the body of the thread. it will update the display area when necessary. The necessity is based on the condition of the update_flag.

file: PhaseResponse.java

public class PhaseResponse extends G2Dint implements Runnable
PhaseResponse implements a canvas which holds the magnitude response of a signal.
Hierarchy: Graph->G2Dint->PhaseResponse

public PhaseResponse(Data fft_input)
Constructor constructs a phase response graph area.

public boolean draw_response()
draw_response draws the magnitude response of the signal on the screen.

double GetAngle(double z_imag, double z_real)
GetAngle returns the angle of a complex number.

public boolean update_screen() {
update_screen sets the update_flag causing a repaint of the canvas.

public void run()
run is the body of the thread. it will update the display area when necessary. The necessity is based on the condition of the update_flag.