DEVELOPMENT OF A TOOL FOR WEB BASED CONTROL ENGINEERING EDUCATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HÜSEYİN CİĞEROĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2004

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Mübeccel DEMİREKLER
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. Önder YÜKSEL
Supervisor

Examining Committee Members

Prof. Dr. Mübeccel DEMİREKLER	(METU, EE)	_____
Prof. Dr. Önder YÜKSEL	(METU, EE)	_____
Prof. Dr. Kemal LEBLEBİCİOĞLU	(METU, EE)	_____
Prof. Dr. Hasan GÜRAN	(METU, EE)	_____
Tolga ÇAMLIKAYA	(BİLTEN)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Hüseyin Ciğeroğlu

ABSTRACT

DEVELOPMENT OF A TOOL FOR WEB BASED CONTROL ENGINEERING EDUCATION

CİĞEROĞLU, Hüseyin
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Önder YÜKSEL

August 2004, 124 pages

It is obvious that learning is more productive with visual mediums and simulations. Especially in technical subjects, this approach is more important. Visual modification of parameters in a control system provides many benefits both in analyzing the system and in learning process. Additionally if this material is published on the internet, students can reach anywhere anytime to this material.

This thesis describes a Web-based system developed for control engineering education for both the instructor and the student. The system will generate learning material according to the instructor's requests. Instructors will design the system and define the borders to help students to learn rapidly the subject of the lesson. They will decide on the functions and which variables can be played with and present them

to the students. This work will help the students that take the basic courses of control engineering. Students will interactively experiment with the system. They will see the effect by changing the variables via sliders of certain functions (e.g. step, bode, root locus...). The system is developed with the programming language JAVA to run over the internet and to be platform independent.

Keywords: Educational Software, Control Systems, JAVA, Online Education, Interactive Experimentation

ÖZ

WEB TABANLI KONTROL MÜHENDİSLİĞİ EĞİTİMİ AMAÇLI BİR ARACIN GELİŞTİRİLMESİ

CİĞEROĞLU, Hüseyin
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Danışmanı: Prof. Dr. Önder YÜKSEL

Ağustos 2004, 124 sayfa

Eğitimin görsel araçlar ve simülasyonlarla daha verimli olduğu açıktır. Özellikle teknik konularda bu daha önemli bir hale gelir. Bir kontrol sisteminde değişkenlerin görsel olarak değiştirilmesi hem sistem analizinde hem de öğrenme sürecinde birçok yarar sağlar. Özellikle bu materyal internet üzerinden sunulursa öğrenciler her an her yerde bu materyale ulaşabilirler.

Bu tezde, hem öğretici hem de öğrenciye hizmet edecek kontrol mühendisliği eğitimi amaçlı internet tabanlı bir sistem tanımlanmıştır. Sistem öğreticinin istekleri doğrultusunda eğitim materyali üretecektir. Öğreticiler öğrencinin dersin konusunu hızlıca anlamaları için sistemi tasarlayacak ve sınırlarını belirleyeceklerdir. Hangi fonksiyonların çalıştırılabileceğine ve hangi değişkenlerle oynanabileceğine karar

verip bunu öğrenciye sunacaklardır. Bu çalışma temel kontrol mühendisliği dersini alan öğrencilere yardımcı olacaktır. Öğrenciler sistem sayesinde etkileşimli deney yapabileceklerdir. Öğrenciler bu tezde bahsedilen fonksiyonların (step, bode, root locus...) grafiklerindeki değişimleri değişkenleri değiştirerek görebileceklerdir. Sistemin internet üzerinden erişilebilir olması ve her platformda çalışabilmesi için programlama dili olarak JAVA seçilmiştir.

Anahtar Kelimeler: Eğitim Yazılımı, Kontrol Sistemleri, JAVA, Çevrimiçi Eğitim, Etkileşimli Deney

ACKNOWLEDGMENTS

I would like to express my gratitude to my thesis supervisor Prof. Dr. Önder Yüksel for his guidance, advice and supervision throughout the research.

I like to thank my wife Buket for her support and patience.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

1.1 Productive learning

Traditional lectures have limitations in teaching complex subjects to the students. Experiments and visual media are not always present or enough in the classroom. The need to answer student needs outside this environment is obvious. To improve the learning process, alternative methods must be found. There are various tools to achieve this idea. Tutorials and visual interactive examples in a web lesson is a good example as a supplementary material.

Often mentioned as independent study, self-paced and self-structured learning allow one to work on his/her own outside the classroom. There are no instructors and usually there is not a specified start and end time. Also free-experimentation is often the best way to learn. Since it involves try and see, it is hard to forget anything learned by this way.

1.2 Education on the Web

The World Wide Web has drastically improved the ability to train and educate students electronically. Whether the materials are a static tutorial or an interactive on-line workshop, the Web provides significant new functionality in conducting information to the student and providing discussion lists for knowledge exchange.

Many universities have started online education on the web. There are many benefits to publish educational material on the internet. Since the courses are asynchronous, students may study on the right time that they feel ready to work and at the pace it suits them. In addition, many supplementary resources can be provided to the student. Interactive examples and visual tools can be presented to accelerate the learning process.

Nowadays it is very easy to reach information. Any information can be found by entering a few keywords to search. But it is still a problem to find valuable resources related to your search. If the concept addresses a small audience you are sometimes limited to only a few sources.

1.3 Control Engineering Education on the Web

Control engineering education must combine theory and practice in each lesson. Theory can be taught in the classroom. Practical experience can be learned in laboratories. Nothing will give the same feel to a student sitting in a laboratory. But web education can be close to reality if it can simulate the lab environment satisfactorily.

When integrated with off-line design tools for control systems such as MATLAB or similar real-time tools, the Web can increase students' involvement in the education experience in computer-aided control system design lectures. The Web provides an effective platform for integrating design programs into a single interface and is an ideal place for information which is itself quickly changing. The Web is altering some areas of study through its advantages. It gives new opportunities for learning and reaching information.

Ali Abid from Ruhr University expresses his thoughts about control engineering education with these words: “Education in engineering (especially control engineering) faces two major problems. Firstly, there is a persistent lack of motivation from the student’s side; secondly, the ideas and phenomena involved in system theory are too complex to be illustrated using conventional means of demonstration and instruction.” [2]

MATLAB is a widely used, almost-standard tool and is a great help for control engineers. However, it is hard to deploy examples that use MATLAB on the web. There are two possible approaches. First, it can be assumed that MATLAB is installed on the students' computers. Second, MATLAB is running on a server and serve the students that do not have MATLAB installed. In the first case, all users must install MATLAB to use the examples prepared. In the second case, the server should have a good configuration to serve many users. If one is going to teach across MATLAB, it must be assured that there are no license problems.

1.4 Why JAVA?

In the sequel we will explain why Java is chosen to build a web based education tool by summarizing the article by Matt Curtin, published by the interhack web site on 1998 [3].

Java is a platform-neutral language. That means, Java programs would be written to run on a Virtual Machine instead of a physical computer. Then the virtual machine would run on a real computer. Because Java's Virtual Machine has to strictly adhere to interface and behavior definitions, a programmer can develop a program for the Java Virtual Machine on the sort of computer that he likes, and expect it to run on the sort of computers that the program users like, inside their Java Virtual Machines. This concept is named "write once, run anywhere". This means that when you go to buy an application, you do not buy the "Linux version" or the "Windows version"; you buy the "Java version" instead. In addition, as long as you have the Java Virtual Machine, which is free, and available throughout the internet, you can buy the program without having to worry whether it is going to run on your computer.

What is important here is an understanding that "Write Once, Run Anywhere" is a promise that Java will work the way computers should work. Specifically, that they should be able to work together, as their users want them to, regardless of who makes them or the software they run.

Much of the promise of Java has been delivered. But it is going to take a lot of work by a lot of people to take Java from being a relatively immature tool for a variety of tasks to a mature, robust system which can be applied to almost any computing problem.

Though Java is not the first language to offer runtime platform independence, it is significant that Java has gotten the support of vendors of computing devices from supercomputers to embedded systems and smartcards. Never before has such a diverse set of companies worked so hard together to make their systems be able to run the same code. By and large, developers want to make "Write Once, Run Anywhere" a reality. This will be a good thing for computing, the people who use the technology, and even those who use the services of organizations that use the technology.

Sun, in promoting Java philosophy, have amassed a tremendous amount of support. Without question, the level of support Java enjoys is unprecedented in this industry. Because of this support, organizations have poured resources into Java, resulting in a rapidly maturing environment.

A sophisticated language like Java, after having time to mature, running in a secure and scalable environment like the Java Virtual Machine holds the potential to let us into a new age of software. An age where information technology managers can intelligently provide for the needs of their users.

1.5 The Thesis

In this thesis, a web-based tool, which enables the user to generate interactive examples to supplement basic control engineering instruction, is considered. With the algorithms presented, various responses to transfer functions can be studied by changing their variables.

The instructor can easily develop new systems and GUI's (Graphical User Interface) by changing the parameters in the parameter file. It is easy to create new transfer functions and to determine what part can be controlled by the students. Since

the system was developed with an object-oriented language in a modular way, it is simple to add new functionality to the system.

For the student, there are tabbed panes to easily navigate between functions. Every button and slider is explained within the GUI. There is no need for any help document.

The whole program is written in JAVA and works as an applet. It can be executed on every computer that has a JAVA virtual machine installed. This was one of the biggest aims to use the JAVA programming language.

In chapter 2 the technical background and the developed algorithms are presented. In chapter 3, the instructor part to develop learning material is described. The usage of the system for students is given in chapter 4. Chapter 5 consists of conclusion and further studies.

CHAPTER 2

TECHNICAL BACKGROUND AND ALGORITHMS

2.1 General

The program is developed using the Java programming language and the library J2SE (Java Standard Edition). It is written as an applet with Borland JBuilder 10. All the files are gathered in a JAR file named "`project.jar`" which is a container for JAVA projects. The classes in the archive can be seen by opening the file with standard zip programs. Table 1 show the files used in the project. To execute the code, it is enough to put an applet line into an html file. By opening the html file in a browser, the tool will automatically execute.

The layout consists of frames. The first screen has two frames. The left frame is for operations and data entering. The right frame contains the graph and time slider.

Table 1 The folders and files structure of the project

controlapplet	
	BodeFrame.java
	ComplexNumber.java
	ControlSystemApplet.java
	LeftPanel.java
	MainPanel.java
	newRK4.java
	NyquistFrame.java
	PZSelectPanel.java

Table 1 (cont'd)

	Rlocus.java	
	Tf2ss.java	
	Zp2tf.java	
Jama		
	EigenvalueDecomposition.java	
	Matrix.java	
	Util	
		Maths.java
parameter		
	params.java	
scientific		
	Graphics	
		InvalidPlotValueException.java
		JPlot2D.java
	Io	
		Fmt.java
	Math	
		Array.java
		Complex.java
		InvalidArraySizeException.java
		Math1.java

JavaDoc is a program to automatically generate documentation for the source code. By entering appropriate text to all variables and methods within the code, JavaDoc parses the files and generates easily browsable help documentation for the developer in html format. The help documentation can be found in appendix C.

2.2 Modules

This section will present the modules of the system. The module map is given in Figure 1. The following sub sections describe the modules, their algorithms and interactions with other modules.

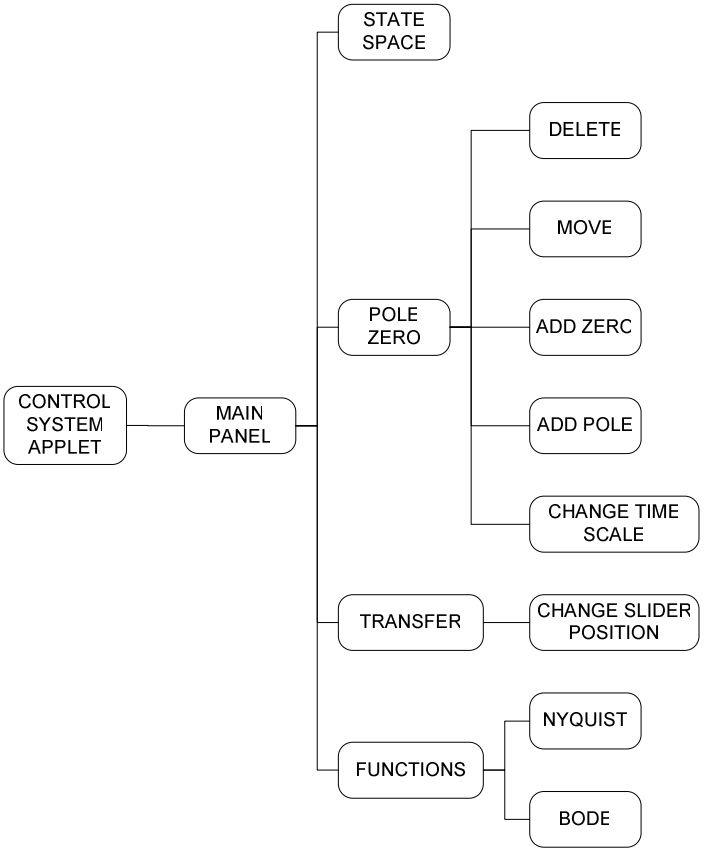

Figure 1 Module Map

2.2.1 ControlSystemApplet Module

Function : Program entry, constructs the applet.

Called by : Called firstly after running the software.

Calls : MainPanel

Inputs : None.

Outputs : None.

Algorithm : Display applet

Create Main Panel

2.2.2 MainPanel Module

Function : Constructs the tabbed pane and graph.

Called by : Called by ControlSystemApplet.

Calls : State Space, PoleZero, Transfer, Functions, Change Time Scale Slider.

Inputs : None.

Outputs : None.

Algorithm : Display tabbed pane, graph panel and time scale slider.

Switch (user input)

Case PZ{ XE "Confirm" }:

Display Pole Zero tab.

Case TF:

Display Transfer Function tab.

Case SS:

Display State Space tab.

Case FUNC:

Display Functions tab.{ XE "Int. 1." }

2.2.3 PoleZero Module

Function : Constructs pole zero tab.

Called by : Called by ControlSystemApplet module.

Calls : Add Pole, Add Zero, Delete Root, Move Root.

Inputs : Cursor information.

Outputs : Root locus and step graphs.

Algorithm : Get user input.

Case Add Pole{ XE "Confirm" }:

Add pole to the system.

Display updated root locus and step response graph.

Case Add Zero:

Add zero to the system.

Display updated root locus and step response graph.

Case Delete Root:

Delete root from system.

Display updated root locus and step response graph.

Case Move root:

Move root in the system.

Display updated root locus and step response graph.

2.2.4 Transfer Module

Function : Constructs the transfer function tab.

Called by : Called by ControlSystemApplet module.

Calls : Change Slider Position.

Inputs : Slider positions.

Outputs : Step graph.

Algorithm : Get slider position.

Update transfer function and step response graph.

2.2.5 State Space Module

Function : Constructs the state space tab.

Called by : Called by ControlSystemApplet module.

Calls : None.

Inputs : State space representation.

Outputs : None.

Algorithm : None.

2.2.6 Functions Module

Function : Constructs the functions tab.

Called by : Called by ControlSystemApplet module.

Calls : Nyquist and Bode modules.

Inputs : Current transfer function.

Outputs : None.

Algorithm : Gets the choice from user and runs the appropriate module.

2.2.7 Nyquist Module

Function : Constructs the Nyquist graph.

Called by : Called by Functions module.

Calls : None.

Inputs : Transfer function.

Outputs : Nyquist graph points array.

Algorithm : Gets the transfer function and constructs the Nyquist graph.

2.2.8 Bode Module

Function : Constructs the Bode graph.

Called by : Called by Functions module.

Calls : None.

Inputs : Transfer function.

Outputs : Bode graph points array.

Algorithm : Gets the transfer function and constructs the Bode graph.

2.3 Technical Background

In this part the functions and their algorithms are presented. Most of these algorithms are developed by the author, while certain modules taken from exterior sources are indicated by references. The functions that are developed are step response, bode plot, nyquist plot, root-locus graphics and pole-zero to transfer function, transfer function to state space conversions.

2.3.1 Step Response Algorithm

Step response graphs are very important in control engineering. They give information like settling time, rise time, steady state, peak response, overshoot parameters of the system. In addition, the system stability can be measured.

If u(t) is a step function such that

$$u(t) = \begin{Bmatrix} 1 & t \geq 0 \\ 0 & otherwise \end{Bmatrix}, \tag{2.1}$$

the system F's step response is defined as

$$g(t) = F[u(\cdot)]. \tag{2.2}$$

The step response algorithm is implemented with the use of Runge-Kutta algorithm. It is a numerical integration algorithm. For numerical computing there are many different numerical integration algorithms available, but the fourth order Runge-Kutta (RK4) is commonly used in control engineering design. RK4 is used to solve numerical solution to the kind of differential equations with initial value expressed as:

$$\begin{aligned} &\frac{dy(t)}{dt} = f(t, y), \\ &a \leq t \leq b, \\ &y(a) = \alpha \end{aligned} \tag{2.3}$$

It is a method of numerically integrating ordinary differential equations by using a trial step value at the midpoint of an interval to cancel out lower order error terms. The second order formula is

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\
y_{n+1} &= y_n + k_2 + O(h^3),
\end{aligned}
\tag{2.4}
$$

and the fourth order formula is

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\
k_3 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \\
k_4 &= hf(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5)
\end{aligned}
\tag{2.5}
$$

This method is reasonably simple and strong and is a good convenient candidate for numerical solution of differential equations when combined to an intelligent adaptive step-size algorithm. The flow of the step response algorithm is given in Figure 2.

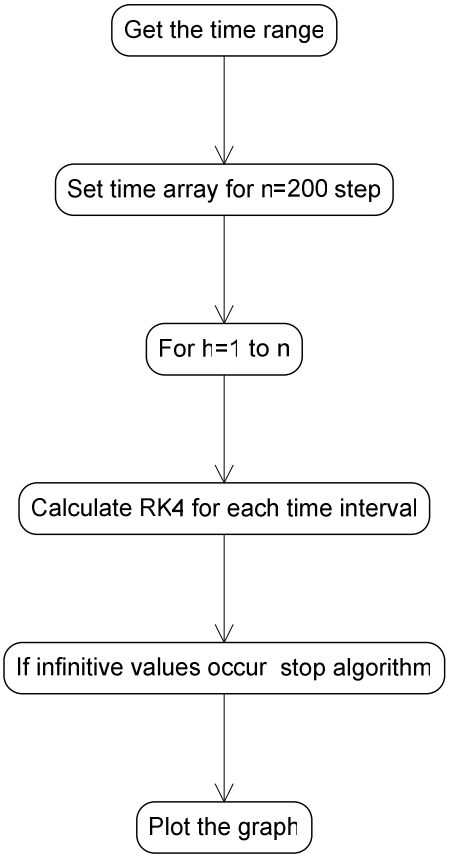

Figure 2 Step response algorithm

The code for the step response is given in Appendix B.

2.3.2 Bode Plot Algorithm

To analyze and design feedback loops, bode plots are very useful. It is a graphical technique for analyzing and designing control systems in the frequency domain. These plots contain two curves, namely the gain and phase versus the frequency. The frequency axis is a log axis which increases with the powers of ten. So we obtain a semilog graph (the gain is also the log of gain, but its usually shown linearly in decibels).

It is easy to calculate bode plots for a given transfer function. Since no implicit equations are involved; it simply consists of function evaluation. If we have a system that we want to calculate bode plot for, according to the gain of the v_{out} and v_{in} parameters, we can write:

$$dB = 20 \cdot \log \left| \frac{V_{out}(w)}{V_{in}(w)} \right| \tag{2.6}$$

The phase is found by calculating

$$\begin{aligned} &\frac{V_{out}(w)}{V_{in}(w)} = R(w) + jX(w) \\ &\phi = \tan^{-1}\left[I(w) / R(w) \right] \end{aligned} . \tag{2.9}$$

Figure 3 shows the flowchart of the algorithm. The source of the algorithm is given in Appendix B.

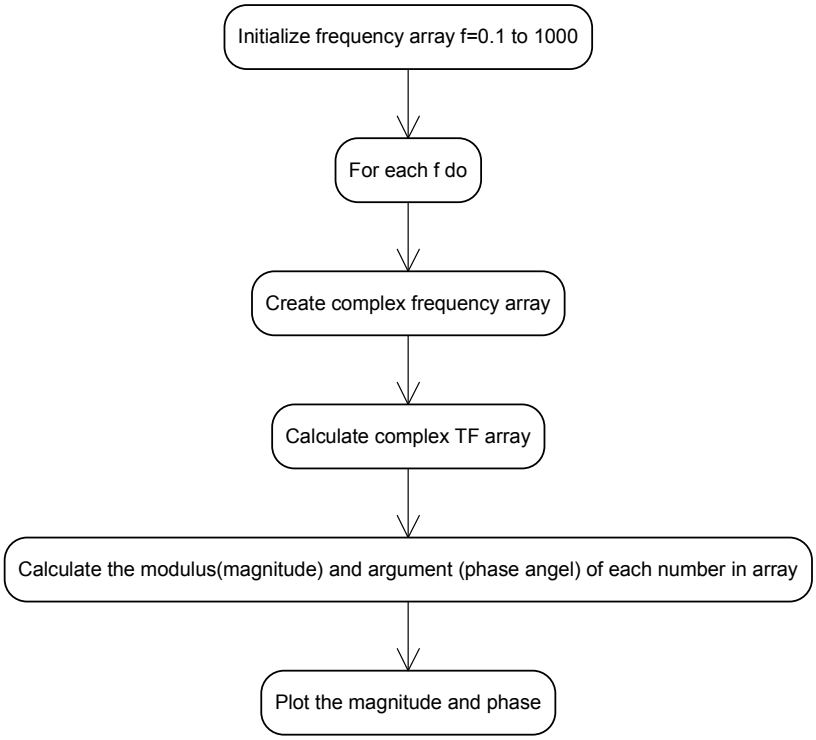

Figure 3 Bode algorithm flowchart

2.3.3 Nyquist Plot Algorithm

The Nyquist plot is very useful in looking at the stability of a negative feedback system. The minimum gain and phase margins can be obtained from the Nyquist diagram. The Nyquist algorithm can be seen in
Figure 4. The source code of the algorithm is given in Appendix B.

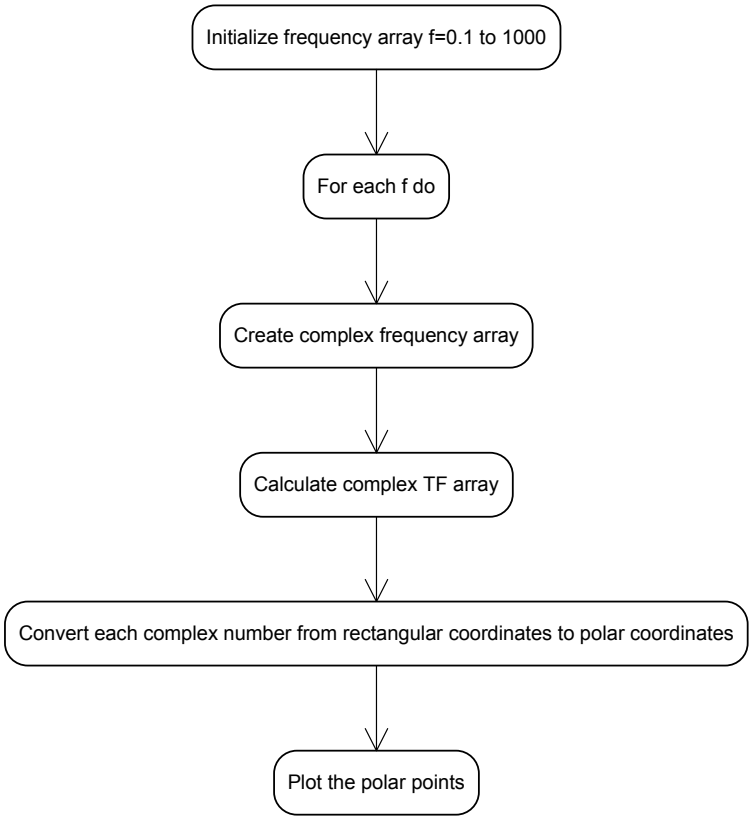

Figure 4 Nyquist algorithm flowchart

2.3.4 Root Locus Algorithm

The root-locus is a powerful graphical tool which gives insight to the complete dynamic response under the effect of a variable parameter. The root locus method helps the designer of a control system to understand the gain of the controller at an operating point.

Root locus is a graphic method for analysis and design of control systems. Given a feedback control system, the root locus illustrates how the pole-zero pattern of the closed-loop system vary with the closed-loop gain.

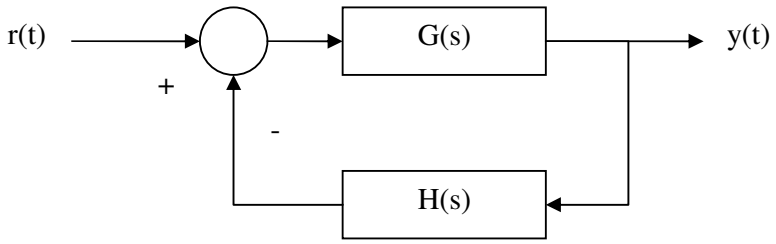

Figure 5 Basic feedback system

The feedback configuration for a basic control system is shown in Figure 5. G(s) represents the open loop system and H(s) is the controller in the feedback loop. The overall transfer function is then obtained as

$$\frac{Y(s)}{R(s)} = \frac{G(s)}{1+G(s)H(s)}. \tag{2.10}$$

So the characteristic equation of the closed-loop system is:

$$1+G(s)H(s) = 0. \tag{2.11}$$

If we can write the term G(s)H(s) as

$$G(s)H(s) = \frac{KQ(s)}{P(s)}, \tag{2.12}$$

where Q(s) and P(s) are polynomials. So we can write the characteristic equation as

$$1+\frac{KQ(s)}{P(s)}=\frac{P(s)+KQ(s)}{P(s)}=0$$

$$P(s)+KQ(s)=0 \quad . \tag{2.13}$$

The trajectories of the roots of the last equation as a function of K give us the locus of the system.

The root locus algorithm in the program first converts the transfer function to its state-space representation (A,B,C,D) in controllable canonical form. There is a fixed gains vector k for which the root-locus will be calculated. Then, the algorithm calculates the eigenvalues of A-B·C·k. Finally the output is plotted with different colors to differentiate the paths of the roots. The eigenvalues are calculated by the EigenvalueDecomposition class file which comes with the JAMA package [4]. The root locus algorithm can be seen in Figure 6. The source code of the algorithm is given in Appendix B.

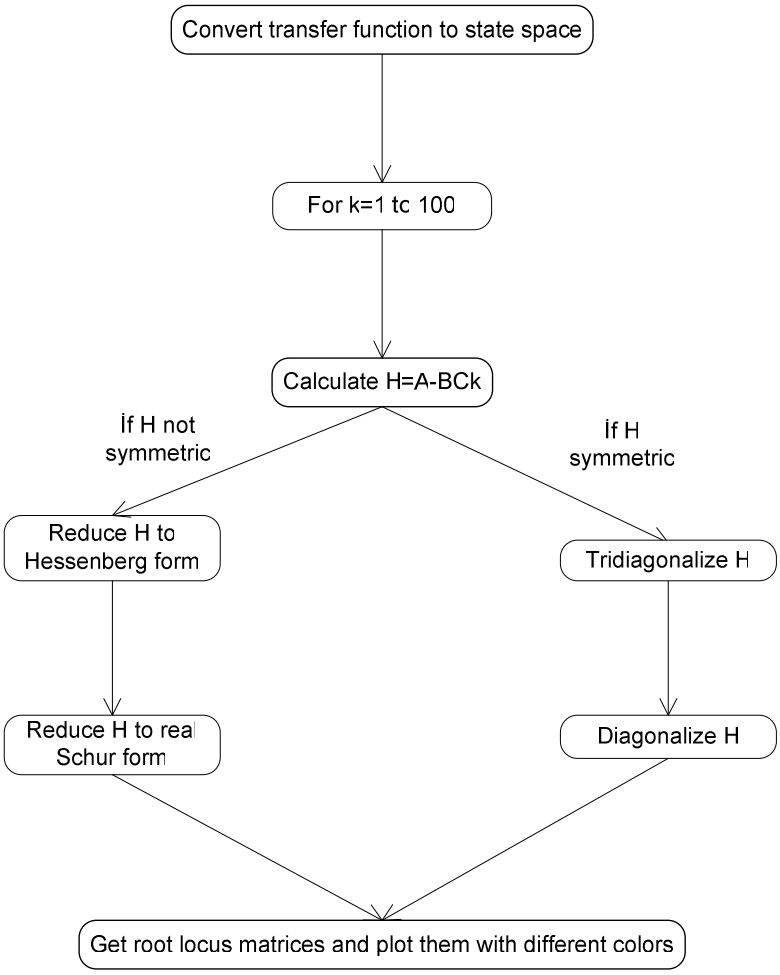

Figure 6 Root locus algorithm

2.3.5 Zero-Pole to Transfer Function Algorithm

The algorithm of converting zero-pole representation to transfer function representation is straightforward. To find the numerator coefficients, the zeros; and to find the denominator coefficients, the poles are given as parameters to the `constructPoly` function. This function converts the multiplying terms to a polynomial and thus converts the transfer function representation from

$$\frac{(s-z_1)(s-z_2)...(s-z_n)}{(s-p_1)(s-p_2)...(s-p_m)} \tag{2.14}$$

to

$$\frac{s^n + as^{n-1} + ... + a_{n-1}}{s^m + bs^{m-1} + ... + b_{m-1}}. \tag{2.15}$$

The listing of this function is given in Appendix B.

2.3.6 Transfer Function to State-Space Algorithm

This algorithm converts the transfer function to the controllable canonical state space representation. This representation,

$$\frac{s^n + as^{n-1} + ... + a_{n-1}}{s^m + bs^{m-1} + ... + b_{m-1}} \tag{2.16}$$

is converted to

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}. \tag{2.17}$$

2.3.7 Graphical Display Library

The Jplot2D class is being used to draw the two dimensional graphical outputs. This component was taken from the Chapman Package [6]. The explanation of the package is given in Appendix E.

CHAPTER 3

INSTRUCTOR PART OF THE SYSTEM

3.1 Aim

The main idea is to develop a tool that the instructor can easily prepare educational material for specific subjects. According to the chosen topic, the related modules should be selected and the desired parameters be changed, to produce a student part.

The instructor part does not involve programming of code and complex operations. Almost every parameter, slider, button can be controlled as to whether they will appear in front of the student and the student may change and use them.

3.2 Defining a New System

The instructor can prepare custom study material for the student by changing various parameters of the program. The initial function, sliders and their interval and visibility, the time scale and briefly everything can be defined by the instructor. The required functions can be opened and the others can be closed so the student can focus on a certain concept. All these parameters are defined in a file named "`params.java`" that can be found in appendix D. After opening and defining a new system in this file, it can be compiled and converted to "`params.class`" file with the javac.exe program. Adding the class file into the "`project.jar`" with a

zip program will make the new system ready to use. The following is a list of what values can be given initially.

3.2.1 System

The system can be defined by the two variables POLES and ZEROS. To enter the real or imaginary point it is important to put a comma between the values. A blank must be placed between different points. There is no need to enter the complex conjugate of a point, since it will be created automatically. Below is an example:

```
public final static String POLES = "5 -5,4j";
public final static String ZEROS = "-5";
```

These values refer to the poles at (5+0j), (-5+4j), (-5-4j) and the zero at (-5+0j) and hence describes the transfer function

$$\frac{s+5}{(s-5)(s+5+4j)(s+5-4j)} \quad . \tag{3.1}$$

3.2.2 Point Operations

Point operations involve adding new poles and zeros, moving points and deleting points. These operations can be given or not given to the students. The only thing is to set the value to true or false. Here is an example:

```
public final static boolean ADD_POLE = true;
public final static boolean ADD_ZERO = true;
public final static boolean PZ_MOVE = true;
public final static boolean PZ_DELETE = false;
```

This example gives the opportunity to add poles and zeros and move them but prohibits the deletion of points.

3.2.3 Time Scale

Time scale variables control the properties of the time slider of the main graph. The variables are,

```
public static boolean SHOW_TIME_SCALE = true;
public static int TIME_AXIS_CURRENT_VALUE = 20;
public final static int TIME_AXIS_MAX_VALUE = 30;
```

These variables define a slider with a maximum value of 30 and current value of 20. If the SHOW_TIME_SCALE variable is set to false no time slider will be available for the student.

3.2.4 Tabs

The system has four tabs: zero-pole representation, transfer function representation, state space representation and functions tab. Each of these tabs can be selected to be displayed or not by setting their variables' value to true or false.

```
public static boolean SHOW_ZP_TAB = true;
public static boolean SHOW_TF_TAB = true;
public static boolean SHOW_SS_TAB = true;
public static boolean SHOW_FUNC_TAB = true;
```

3.2.5 Main Frame

These variables set the initial width and height of the systems window. The actual values are 750 pixels for the width and 625 pixels for the height.

```
public static int FRAME_X = 750;
public static int FRAME_Y = 625;
```

3.2.6 Feedback

The variables define whether to use feedback or not, the initial feedback value and the minimum and maximum values of the slider that controls the feedback value. Also the closed loop transfer function numerator and denominator are stored in the feed_num and feed_den variables.

```
public static boolean feedback = false;
public static int feedbackValue = 0;
public static int FEEDBACK_MIN=-30;
public static int FEEDBACK_MAX=30;
public static double[] feed_num;
public static double[] feed_den;
```

3.2.7 Sliders

The sliders are available for the transfer function representation. With the variables below, it can be chosen to display or not the variables.

```
public static boolean SHOW_NUM_SLIDER1 = true;
public static boolean SHOW_NUM_SLIDER2 = true;
public static boolean SHOW_NUM_SLIDER3 = true;
public static boolean SHOW_NUM_SLIDER4 = true;

public static boolean SHOW_DEN_SLIDER1 = true;
public static boolean SHOW_DEN_SLIDER2 = true;
public static boolean SHOW_DEN_SLIDER3 = true;
public static boolean SHOW_DEN_SLIDER4 = true;
```

Also initial minimum and maximum values can be given with these variables.

```
public static int NUM_SLIDER1_MIN=-30;
public static int NUM_SLIDER1_MAX=30;
public static int NUM_SLIDER2_MIN=-30;
public static int NUM_SLIDER2_MAX=30;
public static int NUM_SLIDER3_MIN=-30;
public static int NUM_SLIDER3_MAX=30;
public static int NUM_SLIDER4_MIN=-30;
public static int NUM_SLIDER4_MAX=30;

public static int DEN_SLIDER1_MIN=-30;
public static int DEN_SLIDER1_MAX=30;
public static int DEN_SLIDER2_MIN=-30;
public static int DEN_SLIDER2_MAX=30;
public static int DEN_SLIDER3_MIN=-30;
public static int DEN_SLIDER3_MAX=30;
public static int DEN_SLIDER4_MIN=-30;
public static int DEN_SLIDER4_MAX=30;
```

3.3 Sample

This part will explain step by step how to prepare a simple study material for the student. The concept is chosen as the effect of the poles on the step function.

3.3.1 Creation of a Sample HTML File

First an HTML file is written. The page is given in Figure 7. This page contains theoretical information and the button to start the tool. The button will show the tool once clicked, and with another click it will hide the tool. Below the button, there are some instructions for the student to complete. These instructions are tasks to complete with using the tool.

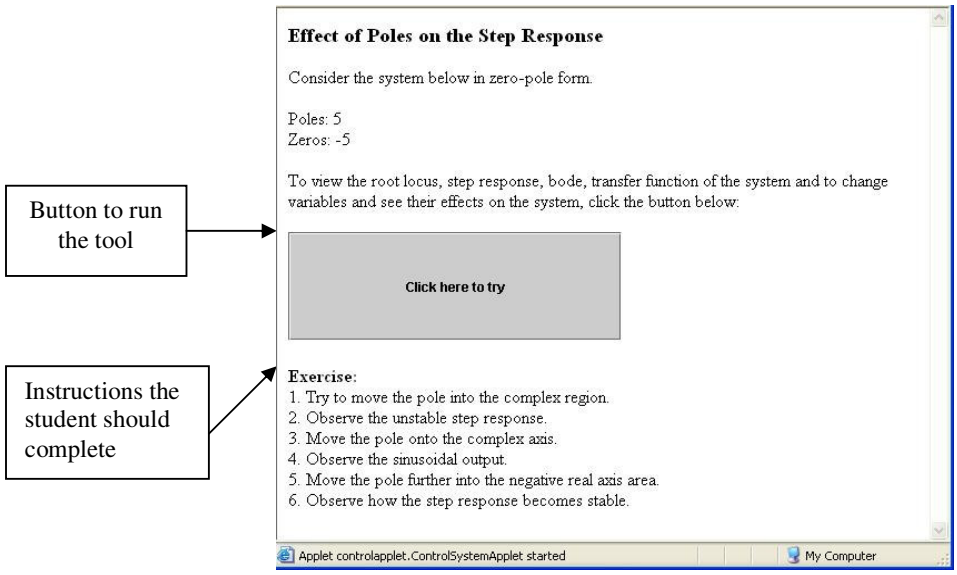

Figure 7 Sample HTML file

3.3.2 Preparing the Parameters File

We have a sample pole-zero system which has a pole at 5 and a zero at -5. The instructor should enter these values into the parameters file "`params.java`" as described in the previous section:

```
public final static String POLES = "5";
public final static String ZEROS = "-5";
```

Then the parameters file should be copied into the system. The container of the system's files is named "`project.jar`". With any zip program this file can be copied into the jar archive.

3.3.3 Binding the Applet

After preparing the parameters, the applet should be inserted into the HTML file. Here is a sample:

```
<APPLET CODE=controlapplet.ControlSystemApplet.class archive="project.jar"
WIDTH=300 HEIGHT=100></APPLET>
```

The "archive" parameter which is named "`project.jar`" is our container for the files of the system.

The HTML file and our tool file can be copied to any web server so students can open this system in their browsers. If students do not have internet connection, the files can be distributed by any medium so students can copy them to their computers and work standalone.

Various samples can be developed about different subjects. The instructor only arranges the parameter file and develops a Web page. The usage of the sample is shown in the student part.

CHAPTER 4

STUDENT PART OF THE SYSTEM

4.1 Aim

The student part aims to teach certain subjects with an interactive graphical user interface. It should be web based so everyone with an internet connection can reach the system and work on it. For a short download time the system is about 224 KB. The views are divided into tabs, the selections consist of radio buttons, and parameters can be varied with sliders or can be entered with just a mouse click. Therefore, the learning time to use the system is very short.

4.2 General

If java runtime engine is installed prior to open the program, there is nothing else that the program needs to run. If java is not installed, it can be found from Sun's website or it can be installed from the given web page in the references section [8].

The first view consists of three parts: a tabbed component, a graph and a slider. The graph plots the step response of the current function. The slider controls the time parameter of the graph. The tabbed pane is used to enter various values, change sliders and to run other commands.

Figure 8 is a sample of the main screen.

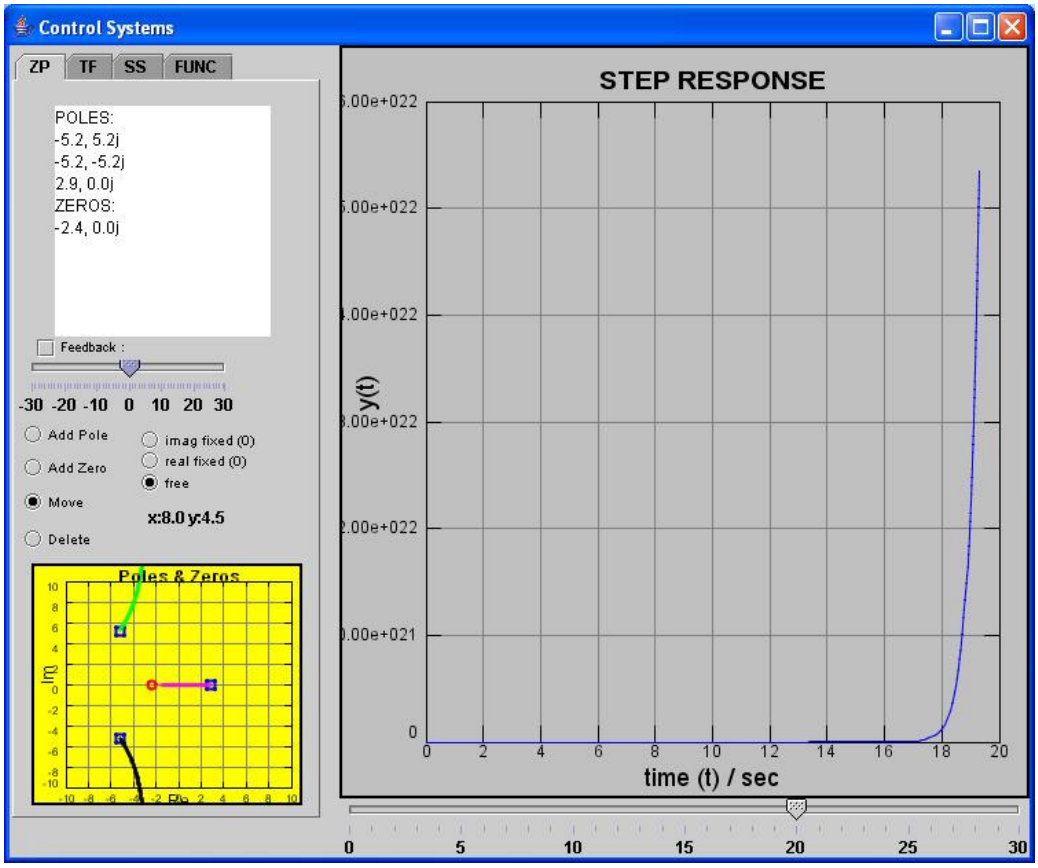

Figure 8 Main screen of the student part

4.2.1 Zero-Pole Representation

This representation has three parts: the text area, radio buttons for certain functions and the pole-zero-root locus graph. The upper part is a list that shows the pole and zero points.

In the middle there are four main radio buttons to select the operation. The first two operations add a pole or zero to the function. After selecting one of the add radio buttons a point can be added by a left mouse click on the graph. While the mouse is over the poles & zeros graph the x and y value can be seen above the graph.

If the “free” option is selected of the upper radio buttons, a point with desired real and imaginary values can be created. If “imaginary axis fixed” is selected a real pole or zero will be created. If “real axis fixed” option is selected, a complex conjugate pair, whose real value is zero, will be created.

The third radio button is to move the points. After selecting that button, a click on one of the points on the graph will catch that point until the left mouse button is released. While holding the left mouse button, the mouse can be dragged and the point will move to the desired position.

The fourth radio button deletes the points. After selecting that option, a left mouse click on the graph is enough to delete the selected point.

The feedback check box plots the closed loop root locus. The k value in the feedback path can be determined by changing the slider value.

The graph will plot the root locus of the system automatically and in synchronization with the operations.

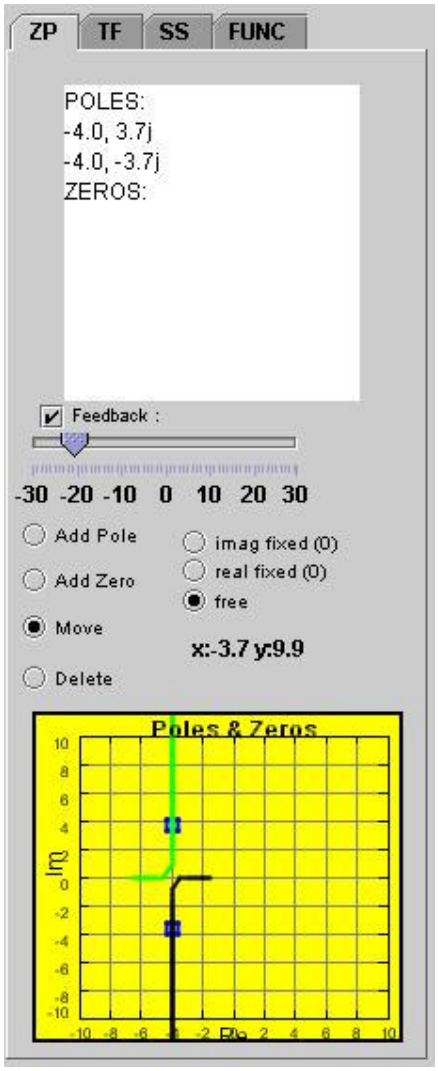

Figure 9 Pole-zero representation tab

4.2.2 Transfer Function Representation

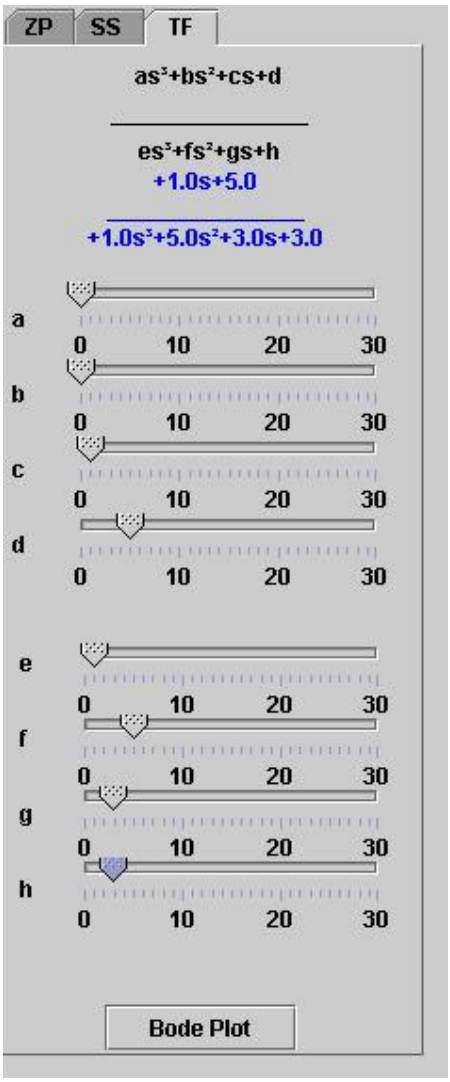

Figure 10 Transfer function representation tab

The TF tab is used to change the coefficients of the transfer function representation of the pole-zero system. The transfer function is displayed in this view and its coefficients can be changed via the sliders. The sliders' initial minimum and

maximum values are defined by the instructor. On the uppermost part of the view there is a legend about which slider controls which value.

4.2.3 State Space Representation

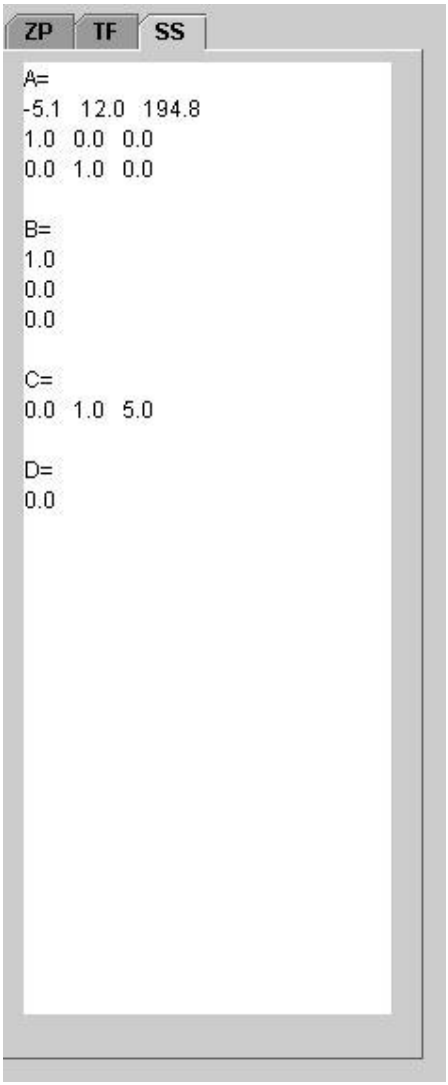

Figure 11 State space representation tab

This is the state space system which shows the A,B,C and D matrices. This view is updated automatically. There is no need to press a button.

4.2.4 Functions

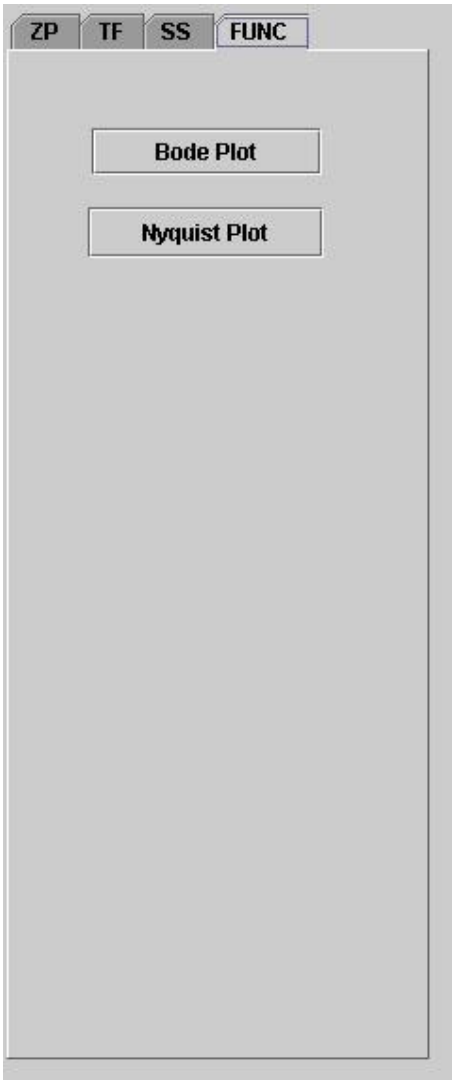

Figure 12 Functions tab

In this view, the functions can be executed for the current system. To execute a function a click on that button is enough.

4.3 Usage of the System

The program can be started by opening the accompanying html file. Since Java is platform independent it can run on every platform. The student can play with many variables and sliders in the system. By changing these values (s)he can see the effects on the graphics. Some of the function usages are explained below.

4.3.1 Step Function Graph

In the pole-zero representation tab one of the add pole, add zero, move and delete operations can be selected. By changing the pole zero graph, the effect can be seen on the right side in the step function graph. The effects on the step response can also be seen by changing the sliders on the transfer function representation tab. Below is a sample output of the step function.

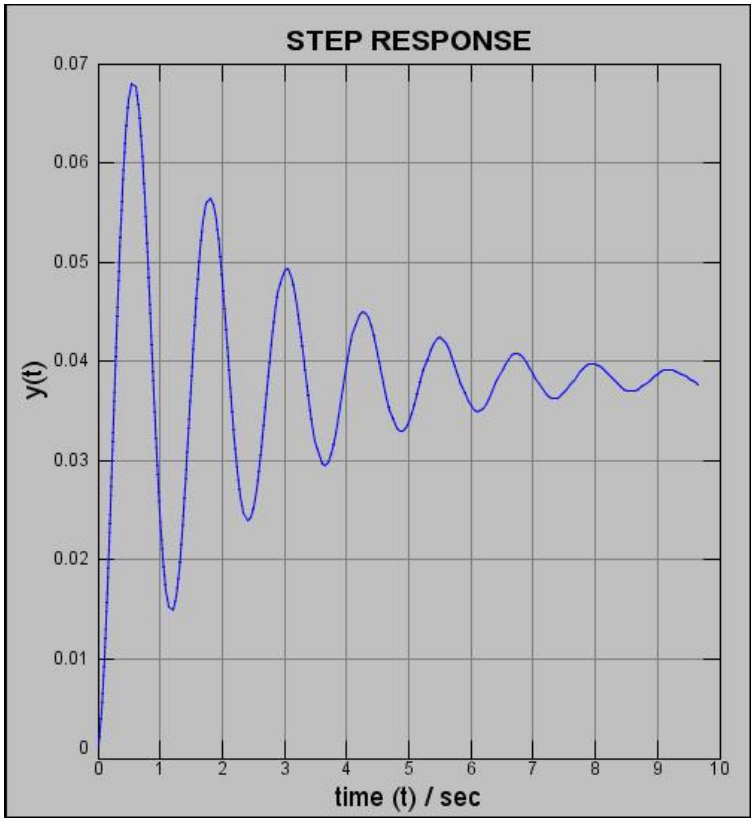

Figure 13 Step response graph

4.3.2 Root Locus Graph

The root locus is automatically drawn to the poles and zeros view. When one of the operations add pole, add zero, move or delete is selected and applied to the poles and zeros view via the mouse, the resulting root locus can be seen in the same view.

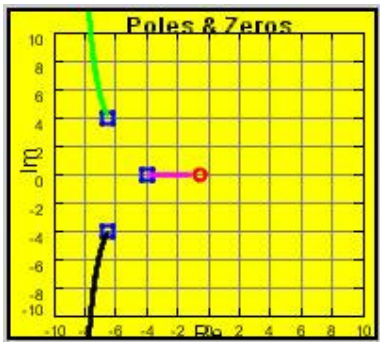

Figure 14 Root locus graph

4.3.3 Bode Graph

The bode graph can be seen by clicking the "Bode Plot" button on the functions tab. When the button is clicked, the bode graph of the active function is plotted.

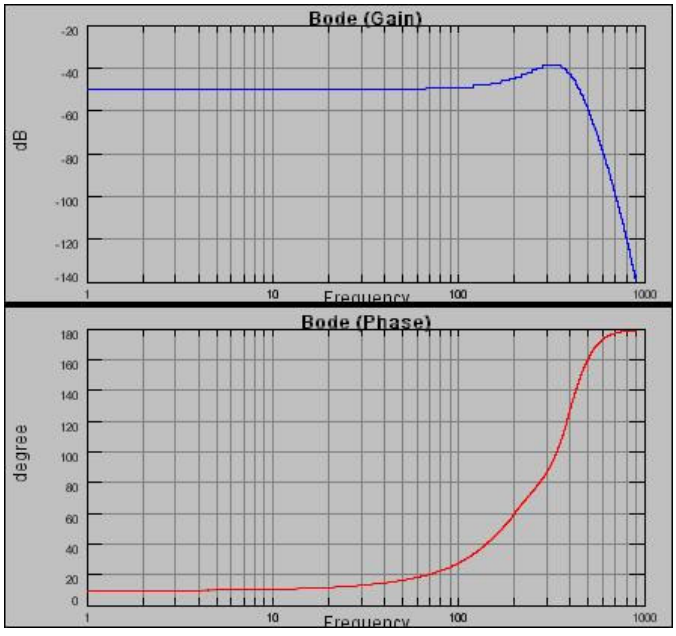

Figure 15 Bode Graph

4.3.4 Nyquist Graph

The nyquist graph can be seen by clicking the “Nyquist Plot” button on the functions tab. When the button is clicked, the nyquist graph of the active function is plotted.

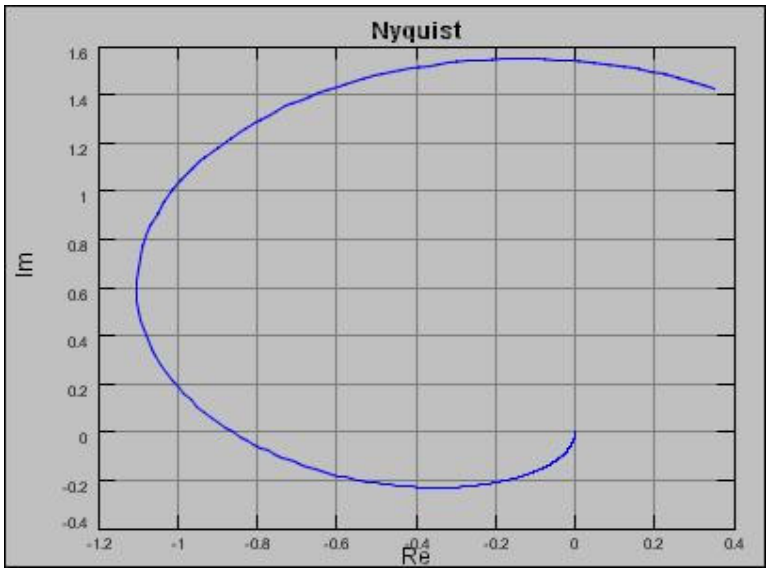

Figure 16 Nyquist Graph

4.4 Sample

In section 3.3 a sample application was developed by the instructor. This section will explain how to use the system. When the files are copied to a web server or distributed to the students, the HTML file should be opened. After reading the theoretical background in the web page, by pressing the button the tool will appear as seen in Figure 17.

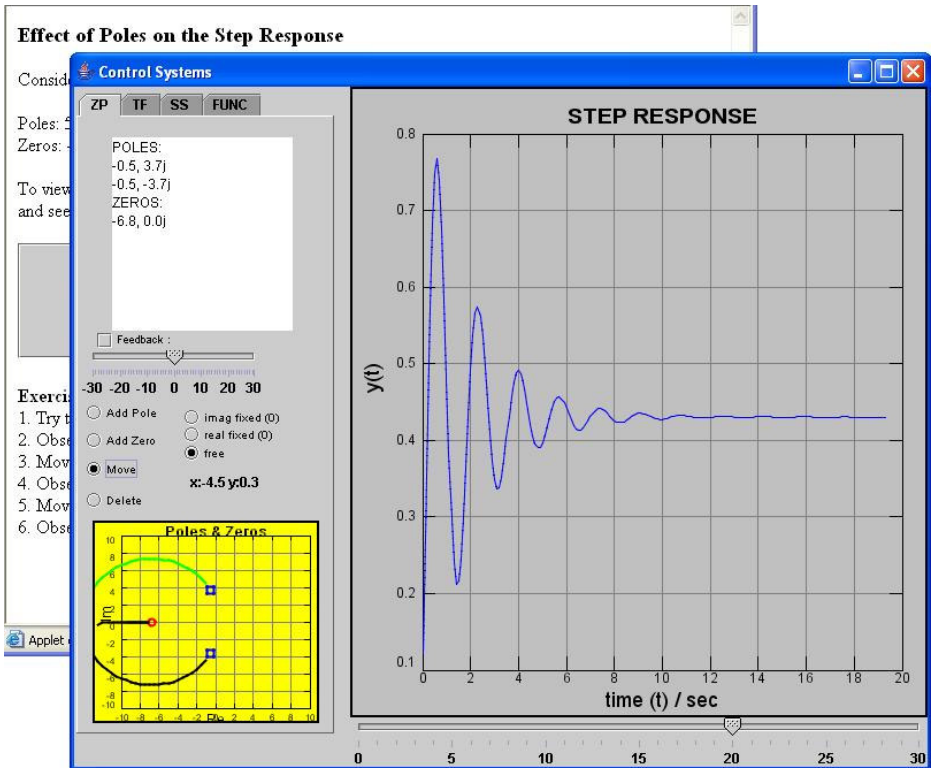

Figure 17 Sample application

There are instructions for the student on the HTML page. Here are the step by step screen outputs of the instructions:

1. Try to move the pole into the complex region: The pole is real. When the student tries to move it outside the real axis, automatically the complex conjugate will be created and shown.

2. Observe the unstable step response: The step response is given in Figure 18. The poles and the step response output is shown in the same screen.

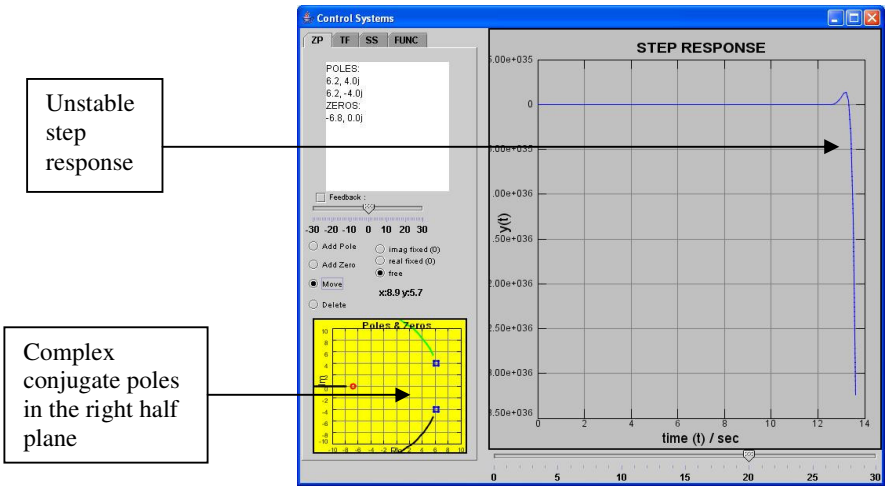

Figure 18 Unstable step response

3. Move the pole onto the complex axis.

4. Observe the sinusoidal output: When the pole is pure imaginary the step response becomes sinusoidal (Figure 19).

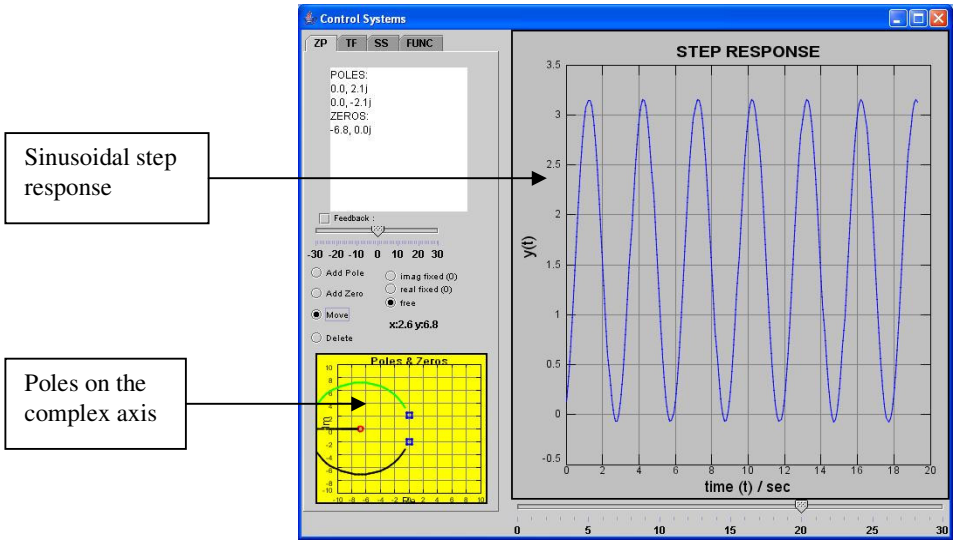

Figure 19 Sinusoidal step response

5. Move the pole further into the negative real axis area.

6. Observe how the step response becomes stable: Now the poles are in the right half plane, and the step response is stable.

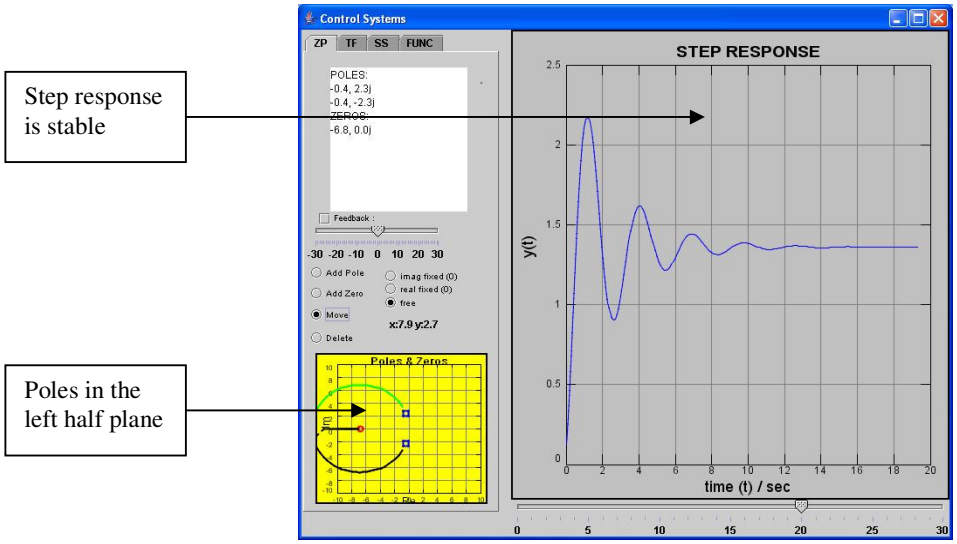

Figure 20 Stable step response

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

Today, information is on our fingertips. The World Wide Web gives the possibility to reach information anywhere anytime. Preparation of visual education material that is published on the internet is not easy. The tool in this thesis was intended to easily prepare customized study material for the student. The changes are made in a text file that we can name as a wizard. There are many options to develop a customized system. The parameters file can be increased to add new controllability options for the instructor.

The tool enables to teach students n'th order transfer functions, the responses of certain functions to the chosen transfer function and the conversions between different representations. More importantly, by changing many sliders and mouse controllable variables the student can see the effects of the modifications on certain parameters and hence gain valuable insight and on-line design experience. For example the sample in the thesis shows how the step response is effected by changing the roots with the mouse on a root locus graph. In the same time, the root locus graph changes dynamically.

The tool does not need any other programs to run except the Java virtual machine. It is device independent. To run the tool, opening a web page is enough. It can be copied to other computers to execute it standalone.

5.2 Future Work

In this thesis algorithms for some certain functions and concepts are developed. These algorithms can be a base library for many control engineering fields. The algorithm library can be extended by addition of other control engineering functions. Same work for advanced control engineering (e.g. optimization) can be done.

The material in this thesis can be changed to cover a whole basic control engineering course. So all the subjects can be thought interactively and graphically.

Visual part can be extended to cover visual objects like circuit elements. After designing a system with drag and drop actions, the transfer function can be obtained automatically from the system.

Since the code is written with the object oriented Java programming language and it contains comments to easily understand the algorithms in it, the tool can be extended in every way. Appendix A contains the UML diagrams of the classes in the program to understand the interactions between classes. In Appendix B source code for the system and algorithms are given.

REFERENCES

[1] Rajni V. Patel, Alan J. Laub, Paul M. Van Dooren. Numerical Linear Algebra Techniques for Systems and Control, IEEE Press, 1993.

[2] Automatic Control Annual Report, Control Engineering Laboratory Department of Electrical Engineering and Information Sciences Ruhr-University, Bochum, 1999.

[3] Interhack (retrieved on Sep 28, 2004), Write Once Run Anywhere: Why It Matters, http://www.interhack.net/people/cmcurtin/rants/write-once-run-anywhere/write-once-run-anywhere.html, last update 1998

[4] National Institute of Standarts and Technology (retrieved on Sep 28, 2004), NIST JAMA Package, http://math.nist.gov/javanumerics/jama/, last update 1999

[5] MathMedics (retrieved on Sep 28, 2004), Finding Roots of Polynomials Graphically and Numerically, http://www.sosmath.com/algebra/factor/fac03/fac03.html, last update 1997

[6] University of Virginia Chemistry Department, 2004, Chapman Package, http://lab3d.chem.virginia.edu/docs/external/chapman/chapman/graphics/JPlot2D.html, last update 2002

[7] Javalobby (retrieved on Sep 28, 2004), Introduction to Java Algorithms in Control Systems, http://www.javalobby.org/kb/entry.jspa?entryID=103&categoryID=49, last update 2004

[8] Sun Microsystems (retrieved on Sep 28, 2004), Sun J2SE Download, http://java.sun.com/j2se/1.3/download.html, last update 2004

[9] Mathworks Corp (retrieved on Sep 28, 2004), MATLAB Documentation, http://www.mathworks.com /access/helpdesk/help/helpdesk.html, last update 2004

APPENDIX A

UML REPRESENTATION OF PACKAGES AND CLASSES IN THE PROGRAM

UML (Unified Modeling Language) is the worldwide standard for representing object-oriented analysis and design diagrams in programming. This appendix will introduce the class diagrams that the author has developed in uml notation.

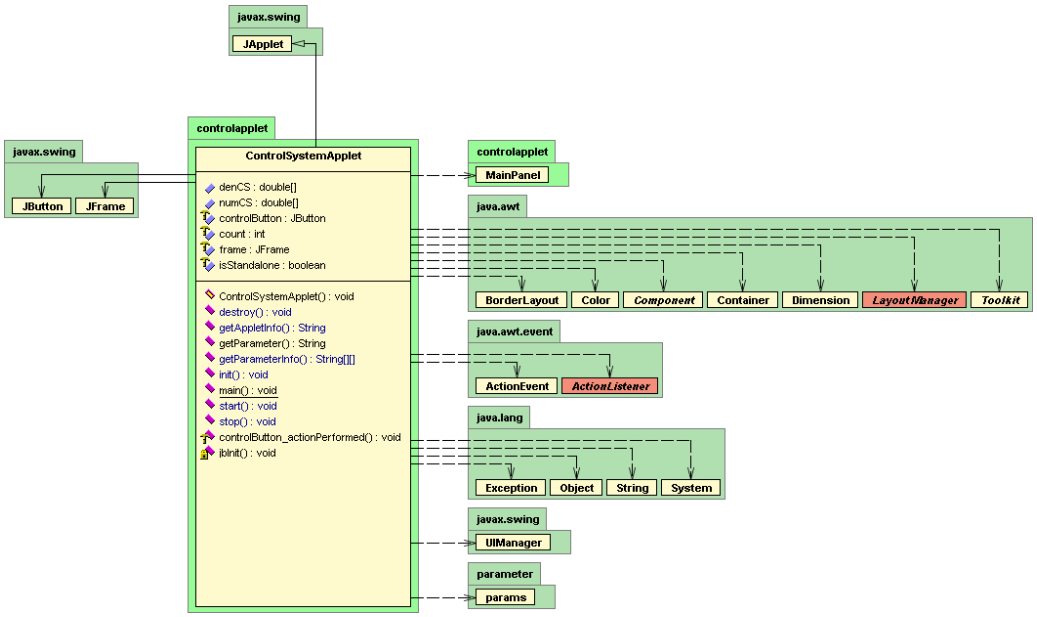

Figure 21 ControlSystemApplet class in UML notation

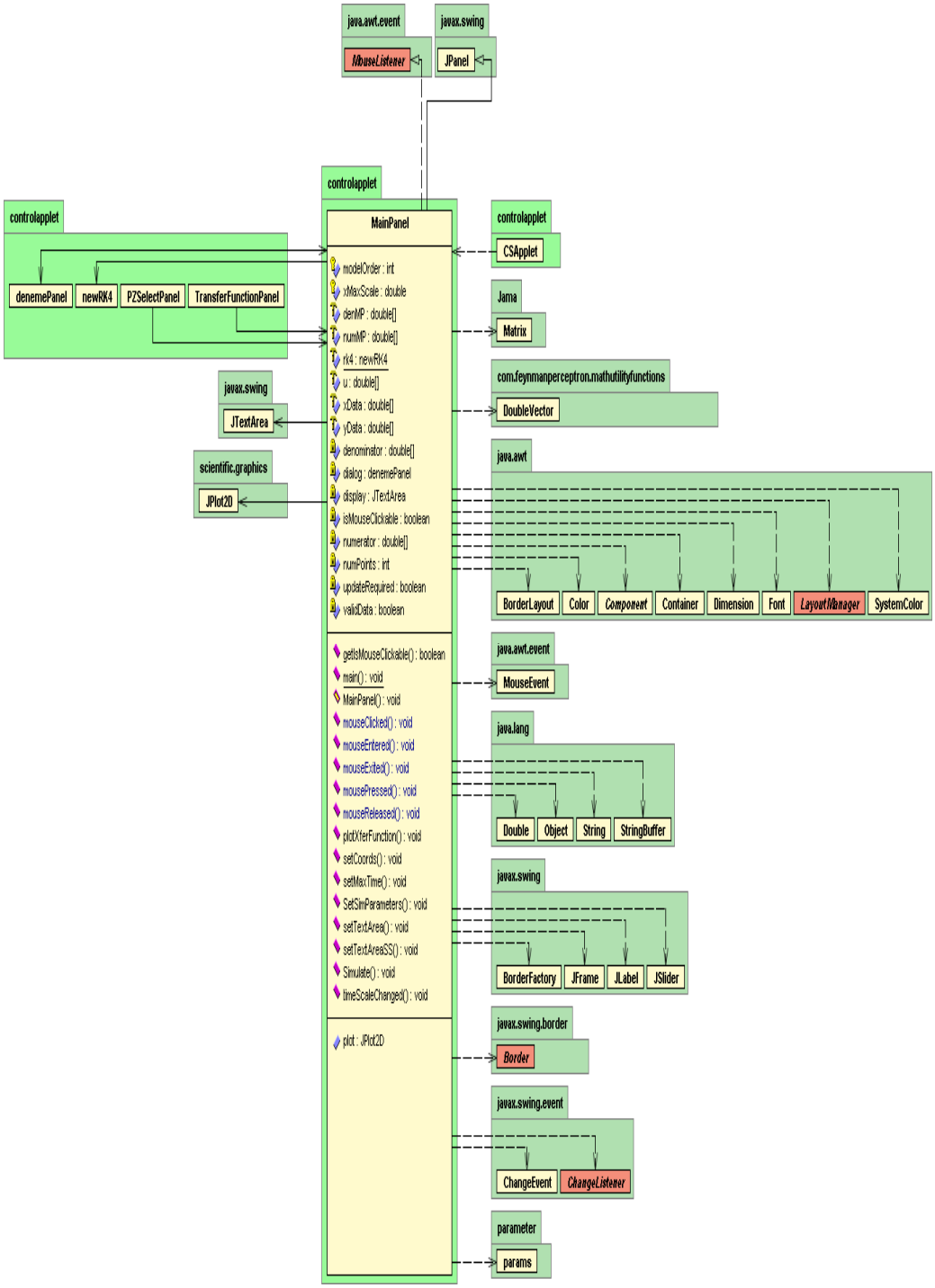

Figure 22 MainPanel class in UML notation

Figure 23 TansferFunctionPanel class in UML notation

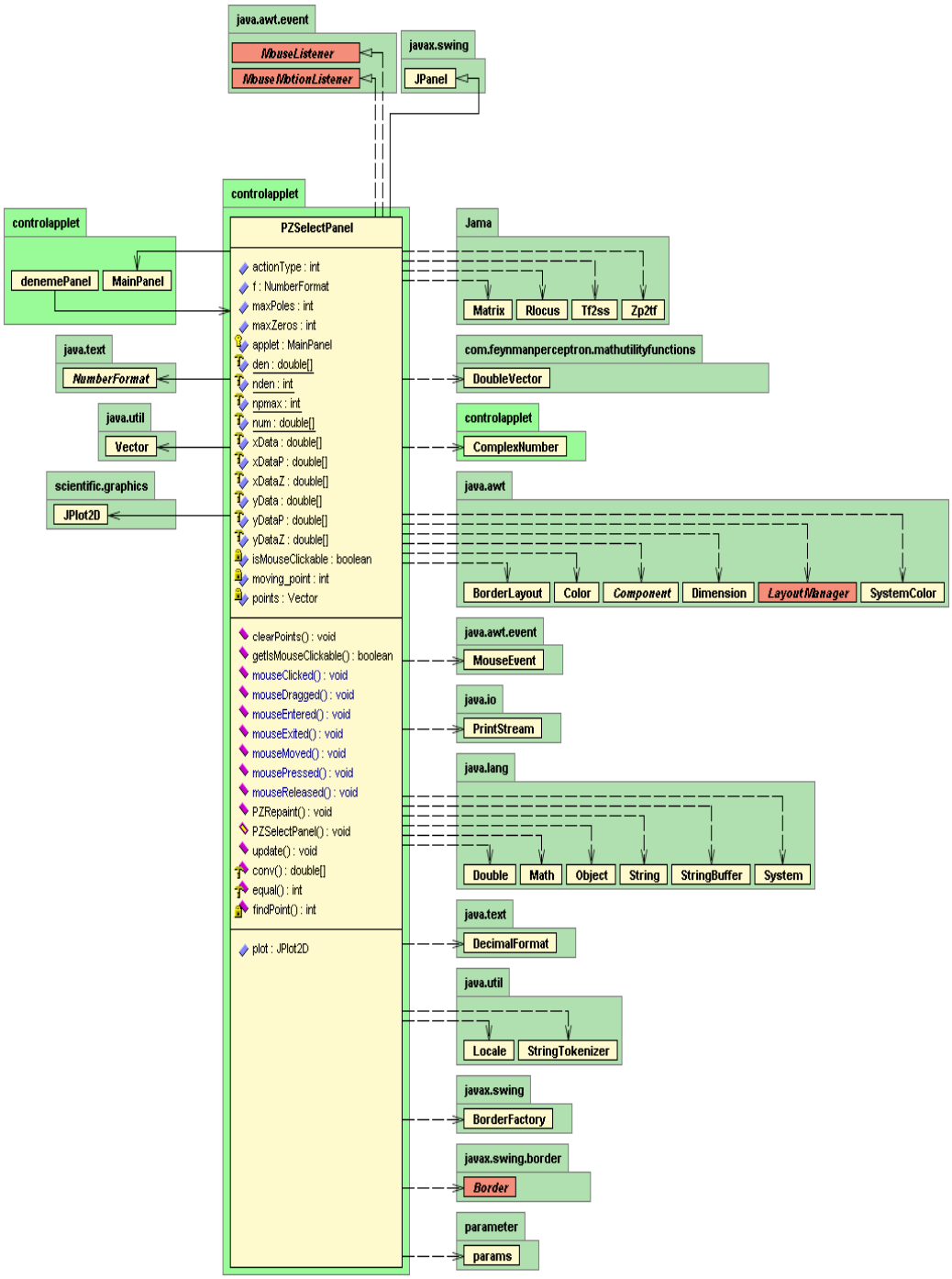

Figure 24 PZSelectPanel class in UML notation

APPENDIX B

SOURCE CODES OF THE PROGRAM

In this part the source codes of the classes that the author has developed will be listed.

1. ControlSystemApplet.java

```
package controlapplet;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

/**
 * Implements the applet.
 *
 * @author Hüseyin CİĞEROĞLU
 * @version 1.0
 *
 */
public class ControlSystemApplet
    extends JApplet {

  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) : getParameter(key) == null ?
        def : getParameter(key);
  }

  public ControlSystemApplet() {
    isStandalone = false;
    count = 0;
    controlButton = new JButton();
  }

  public void init() {
    try {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
```

```
/**
 * Creates (type MainPanel) and frame(type JFrame)
 * (frame is the container for geometry). Also creates
 * controlButton to make the frame visible.
 *
 * @throws Exception
 */
private void jbInit() throws Exception {
  MainPanel geometry = new MainPanel();
  geometry.setPreferredSize(new Dimension(80, 500));
  frame = new JFrame("Control Systems");
  frame.getContentPane().setLayout(new BorderLayout());
  frame.pack();
  frame.setSize(new Dimension(parameter.params.FRAME_X,
                          parameter.params.FRAME_Y)); //main frame size
  Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
  frame.setLocation( (d.width - frame.getSize().width) / 2,
                  (d.height - frame.getSize().height) / 2);
  controlButton.setForeground(new Color(0, 0, 0));
  controlButton.setPreferredSize(new Dimension(120, 30));
  controlButton.setText("Click here to try");

  controlButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      controlButton_actionPerformed(e);
    }
  });
  controlButton.setVisible(true);

  setSize(new Dimension(270, 200));
  frame.getContentPane().add(geometry, BorderLayout.CENTER);
  this.getContentPane().add(controlButton, BorderLayout.CENTER);
  frame.setVisible(false);
}

public void start() {
}

public void stop() {
}

public void destroy() {
}

public String getAppletInfo() {
  return "Control System Applet";
}

public String[][] getParameterInfo() {
  return null;
}

/**
 * Creates applet(type CAApplet) and frame(type JFrame)
 * (frame is the container for geometry). Also creates
 * controlButton to make the frame visible.
 *
 * @params args
 */
public static void main(String args[]) {
  ControlSystemApplet applet = new ControlSystemApplet();
  applet.isStandalone = true;
  JFrame frame = new JFrame();
```

```
    frame.setDefaultCloseOperation(3);
    frame.setTitle("Control Systems");
    frame.getContentPane().add(applet, "Center");
    applet.init();
    applet.start();
    frame.setSize(new Dimension(400, 300));
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation( (d.width - frame.getSize().width) / 2,
                       (d.height - frame.getSize().height) / 2);
    frame.setVisible(false);
  }

  void controlButton_actionPerformed(ActionEvent e) {
    if (count % 2 == 0) {
      frame.setVisible(true);
      controlButton.setText("Click here to hide");
    }
    else {
      frame.setVisible(false);
      controlButton.setText("Click here to try");
    }
    count++;
  }

  boolean isStandalone;
  int count;
  JButton controlButton;
  JFrame frame;

  static {
    try {
      UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    }
    catch (Exception exception) {}
  }

  public double[] numCS;
  public double[] denCS;

}
```

2. MainPanel.java

```
package controlapplet;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import scientific.graphics.JPlot2D;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import parameter.params;
import Jama.Matrix;

/**
 * Implements the main screen of the applet.
 *
 * @author Hüseyin CİĞEROĞLU
 * @version 1.0
 *
```

```
 */
public class MainPanel
    extends JPanel
    implements MouseListener {

  static newRK4 rk4;
  private JTextArea display;
  private boolean updateRequired;
  private boolean validData;
  private boolean isMouseClickable;
  private double numerator[];
  private double denominator[];
  protected double xMaxScale;
  private int numPoints;
  double u[];
  double xData[];
  double yData[];
  double numMP[];
  double denMP[];
  protected int modelOrder;

  private LeftPanel dialog;
  protected JPlot2D plot;
  public MainPanel() {
    isMouseClickable = false;
    numPoints = 1000;
    updateRequired = false;
    validData = false;
    xMaxScale = 10D;
    setLayout(new BorderLayout());
    dialog = new LeftPanel(this);
    plot = new JPlot2D();
    SetSimParameters(numMP, denMP);
    plot.setValues(xData, yData);
    plot.setGridState(true);
    plot.setXLabel("time (t) / sec");
    plot.setYLabel("y(t)");
    plot.setTitle("STEP RESPONSE");
    plot.setBackground(SystemColor.lightGray);
    plot.setBorder(BorderFactory.createLineBorder(Color.black, 2));
    JPanel centerPanel = new JPanel(new BorderLayout());
    JSlider timeScale = new JSlider();
    centerPanel.add(plot, "Center");

    timeScale.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        timeScaleChanged(e);
      }
    });

    timeScale.setMajorTickSpacing(5);
    timeScale.setValue(params.TIME_AXIS_CURRENT_VALUE);
    timeScale.setMaximum(params.TIME_AXIS_MAX_VALUE);
    timeScale.setMinorTickSpacing(1);
    timeScale.setPaintLabels(true);
    timeScale.setPaintTicks(true);
    timeScale.setFont(new java.awt.Font("Dialog", 0, 10));
    if (parameter.params.SHOW_TIME_SCALE) {
      centerPanel.add(timeScale, "South");
    }
    add(centerPanel, "Center");
    add(dialog, "West");
    addMouseListener(this);
```

```
  plot.addMouseListener(this);
}

public void timeScaleChanged(ChangeEvent e) {
  JSlider source = (JSlider) e.getSource();
  int fps = (int) source.getValue();
  setMaxTime(fps);
  dialog.refreshGraph();
}

public boolean getIsMouseClickable() {
  return isMouseClickable;
}

public void plotXferFunction() {
  if (plot != null) {
  }
}

public void setMaxTime(double t) {
  parameter.params.TIME_AXIS_CURRENT_VALUE = (int) t;
  xMaxScale = t;
}

public void Simulate() {
  int numOrder = 0;
  int denOrder = 0;
  for (int order = 0; order < numMP.length; order++) {
    if (numMP[order] != 0) {
      numOrder = order + 1;
    }
    if (denMP[order] != 0) {
      denOrder = order + 1;
    }
  }

  double[] nn = new double[13];
  double[] dd = new double[13];
  for (int order = 0; order < parameter.params.num.length; order++) {
    nn[order] = parameter.params.num[order];
  }
  for (int order = 0; order < parameter.params.den.length; order++) {
    dd[order] = parameter.params.den[order];
  }

  denOrder = denominator.length;
  xData = new double[194];
  yData = new double[194];
  rk4 = new newRK4(0.0F, parameter.params.TIME_AXIS_CURRENT_VALUE); //);
  rk4.intRK4(0, parameter.params.TIME_AXIS_CURRENT_VALUE, denOrder - 1, nn,
            dd);

  double xaf[][] = new double[newRK4.nstep + 1][newRK4.mxplot];
  double xaf1[] = new double[newRK4.nstep + 1];
  int infinitive = 0;
  for (int k = 0; k < newRK4.nstep - 6; k++) {
    xaf1[k] = newRK4.xst[k];
    xData[k] = newRK4.xst[k];
    for (int i = 0; i <= 0; i++) {
      xaf[k][i] = newRK4.yst[k][i];
      if ( (Double.isNaN(newRK4.yst[k][i])) ||
          (Double.isInfinite(newRK4.yst[k][i]))) {
        if (infinitive == 0) infinitive = k;
```

```
      }
      else
        yData[k] = newRK4.yst[k][i];
    }
  }
  double[] newx;
  double[] newy;
  if (plot != null) {
    if (infinitive != 0) {
      newx = new double[infinitive];
      newy = new double[infinitive];
      for (int i = 0; i < infinitive; i++) {
        newx[i] = xData[i];
        newy[i] = yData[i];
      }
    }
    else {
      newx = new double[194];
      newy = new double[194];
      for (int i = 0; i < 194; i++) {
        newx[i] = xData[i];
        newy[i] = yData[i];
      }
    }
    plot.setValues(newx, newy);
    plot.repaint();
  }

  updateRequired = false;
  validData = true;
  if (dialog != null) {
    for (int order = 0; order < denMP.length; order++) {
      dialog.numerator[order] = numMP[order];
      dialog.denominator[order] = denMP[order];
    }
    dialog.writeTF();
  }
}

public synchronized void SetSimParameters(double num[], double den[]) {
  this.numMP = new double[den.length];
  for (int k = 0; k < num.length; k++)
    numMP[k] = num[k];
  this.denMP = den;

  int denOrder = den.length - 1;
  denominator = new double[den.length];
  for (int i = 0; i < den.length; i++) {
    denominator[i] = den[i];
  }

  numerator = new double[den.length];
  for (int i = 0; i < den.length; i++) {
    numerator[i] = 0;
  }
  for (int i = 0; i < num.length; i++) {
    numerator[i] = num[i];
  }

  if (denOrder == 0) {
    denOrder = 1;
    denominator[1] = 0.0D;
  }
```

```
  Simulate();
}

public JPlot2D getPlot() {
  return plot;
}

public static void main(String args[]) {
  MainPanel geometry = new MainPanel();
  geometry.setPreferredSize(new Dimension(300, 150));
  JFrame frame = new JFrame("Control Systems");
  frame.getContentPane().setLayout(new BorderLayout());
  frame.getContentPane().add(geometry, "Center");
  frame.pack();
  frame.show();
}

public void mouseClicked(MouseEvent e) {
  Object ob = e.getSource();
  if (getIsMouseClickable() && (ob instanceof JPlot2D)) {
    getPlot().mouseClicked(e);
    if (getPlot().getCurrentPoint() != null) {
      StringBuffer sb = new StringBuffer("".concat(String.valueOf(String.
          valueOf(display.getText()))));
      String str = String.valueOf(String.valueOf( (new StringBuffer(" t = ")).
          append(getPlot().getCurrentPoint().getVecX()).append("\n").append(
          " y = ").append(getPlot().getCurrentPoint().getVecY()).append(
          "\n\n")));
      sb.append(str);
      display.setText(sb.toString());
    }
  }
}

public void mousePressed(MouseEvent mouseevent) {
}

public void mouseReleased(MouseEvent mouseevent) {
}

public void mouseEntered(MouseEvent mouseevent) {
}

public void mouseExited(MouseEvent mouseevent) {
}

public void setTextArea(double xP[], double yP[], double xZ[], double yZ[]) {
  if (dialog != null) {
    dialog.jTextArea1.setText("");
    dialog.jTextArea1.setLineWrap(true);
    dialog.jTextArea1.removeAll();
    dialog.jTextArea1.append("POLES:\n");
    for (int x = 0; x < 12; x++) {
      if (xP[x] != 0 || yP[x] != 0) {
        dialog.jTextArea1.append(xP[x] + ", " + yP[x] + "j\n");
      }
    }
    dialog.jTextArea1.append("ZEROS:\n");
    for (int x = 0; x < 12; x++) {
      if (xZ[x] != 0 || yZ[x] != 0) {
        dialog.jTextArea1.append(xZ[x] + ", " + yZ[x] + "j\n");
      }
    }
```

```
  }
 }

 public void setTextAreaSS(Matrix A, Matrix B, Matrix C) {
  if (dialog != null) {
   dialog.textAreaSS.setText("");
   dialog.textAreaSS.setLineWrap(true);
   dialog.textAreaSS.removeAll();
   dialog.textAreaSS.append("A=" + "\n");

   String row = "";
   for (int x = 0; x < A.getRowDimension(); x++) {
    row = "";
    for (int y = 0; y < A.getColumnDimension(); y++) {
     row = row + Double.toString(A.get(x, y)).substring(0,
         Double.toString(A.get(x, y)).indexOf(".") + 2) + "    ";
    }
    dialog.textAreaSS.append(row + "\n");
   }

   dialog.textAreaSS.append("\nB=" + "\n");
   for (int x = 0; x < B.getRowDimension(); x++) {
    row = "";
    for (int y = 0; y < B.getColumnDimension(); y++) {
     row = row + Double.toString(B.get(x, y)).substring(0,
         Double.toString(B.get(x, y)).indexOf(".") + 2) + "    ";
    }
    dialog.textAreaSS.append(row + "\n");
   }

   dialog.textAreaSS.append("\nC=" + "\n");
   for (int x = 0; x < C.getRowDimension(); x++) {
    row = "";
    for (int y = 0; y < C.getColumnDimension(); y++) {
     row = row + Double.toString(C.get(x, y)).substring(0,
         Double.toString(C.get(x, y)).indexOf(".") + 2) + "    ";
    }
    dialog.textAreaSS.append(row + "\n");
   }

   dialog.textAreaSS.append("\nD=" + "\n" + "0.0");
  }
 }

 public void setCoords(double x, double y) {
  dialog.jLabel14.setText("x:" + x + " y:" + y);
 }

}
```

3. BodeFrame.java

```
package controlapplet;

import java.awt.*;
import javax.swing.*;
import scientific.graphics.JPlot2D;

/**
 * Implements the bode graph algorithm given the transfer function.
 *
```

```
 * @author Hüseyin CİĞEROĞLU
 * @version 1.0
 *
 */
public class BodeFrame {
  protected JPlot2D plotMag;
  protected JPlot2D plotPhase;
  Complex[] theResult;
  double[] theFreqs;
  String numStr;
  String denStr;
  double maxDB = 0.0;

  public BodeFrame(double x[], double y[]) {
    JFrame frame = new JFrame("Bode Plot");
    frame.getContentPane().setLayout(new BorderLayout());
    plotMag = new JPlot2D();
    plotPhase = new JPlot2D();

    double xDataP[] = new double[800];
    double yDataP[] = new double[800];
    double xDataZ[] = new double[800];
    double yDataZ[] = new double[800];
    plotMag.setGridState(true);
    plotMag.setXLabel("Frequency");
    plotMag.setYLabel("dB");
    plotMag.setTitle("Bode (Gain)");

    plotPhase.setGridState(true);
    plotPhase.setXLabel("Frequency");
    plotPhase.setYLabel("degree");
    plotPhase.setTitle("Bode (Phase)");

    plotMag.setPlotType(plotMag.SEMILOGX);
    plotMag.setLineColor(Color.blue);
    plotMag.setMarkerColor(Color.blue);
    plotMag.setMarkerState(true);
    plotMag.setLineState(plotMag.LINE_ON);

    plotPhase.setPlotType(plotPhase.SEMILOGX);
    plotPhase.setLineColor(Color.red);
    plotPhase.setMarkerColor(Color.red);
    plotPhase.setMarkerState(true);
    plotPhase.setLineState(plotPhase.LINE_ON);


    int index = 0;

    theFreqs = new double[800];
    theFreqs[0] = 0.1;
    for (index = 0; index < 799; index++) {
      theFreqs[index + 1] = Math.pow(10.0, 0.005) * theFreqs[index];
    }

    theResult = plotAll(x, y);

    double[] dBResult = new double[theResult.length];
    maxDB = -100.0;
    for (index = 0; index < theResult.length; index++) {
      double theMag = theResult[index].mod();
      theMag = 20.0 * Math.log(theMag); //hus / Math.log(10.0);
      dBResult[index] = theMag;
      maxDB = Math.max(maxDB, theMag);
```

```
    }

    int oldy = 0;
    int newy = 0;
    for (index = 0; index < theResult.length; index++) {
      xDataP[index] = theFreqs[index];
      yDataP[index] = theResult[index].arg(); //newy;//*57.297;
    }

    for (index = 0; index < theResult.length; index++) {
      xDataZ[index] = theFreqs[index];
      yDataZ[index] = dBResult[index];
    }
  //phase:
    plotMag.addCurve(xDataZ, yDataZ);
    plotMag.setLineColor(Color.blue);
    plotMag.setLineStyle(plotMag.LINESTYLE_SOLID);
    plotMag.setLineState(plotMag.LINE_ON);
  //gain:
    plotPhase.addCurve(xDataP, yDataP);
    plotPhase.setLineColor(Color.red);
    plotPhase.setLineStyle(plotMag.LINESTYLE_SOLID);
    plotPhase.setLineState(plotMag.LINE_ON);

    plotMag.setPreferredSize(new Dimension(400, 300));
    plotMag.setBackground(SystemColor.lightGray);
    plotMag.setBorder(BorderFactory.createLineBorder(Color.black, 2));
    plotMag.setVisible(true);

    plotPhase.setPreferredSize(new Dimension(400, 300));
    plotPhase.setBackground(SystemColor.lightGray);
    plotPhase.setBorder(BorderFactory.createLineBorder(Color.black, 2));
    plotPhase.setVisible(true);

    frame.getContentPane().add(plotMag, "North");
    frame.getContentPane().add(plotPhase, "Center");
    frame.pack();
    frame.show();

  }

  public Complex[] plotAll(double[] nums, double[] dens) {
    if (nums.length == 0) {
      return null;
    }
    if (dens.length == 0) {
      return null;
    }
    if (theFreqs.length == 0) {
      return null;
    }

    Complex[] thePlot = new Complex[theFreqs.length];

    int outer = 0;
    for (outer = 0; outer < theFreqs.length; outer++) {
      Complex cFreq = new Complex(0, theFreqs[outer]);

      Complex cMult = new Complex(1, 0);

      Complex cNum = new Complex(0, 0);
      int inner = 0;
      for (inner = 0; inner < nums.length; inner++) {
```

```
      Complex cCoef = new Complex(nums[nums.length - 1 - inner], 0);

      cCoef.mpy(cMult);
      cNum.add(cCoef);

      cMult.mpy(cFreq);
     }

     cMult = new Complex(1, 0);

     Complex cDen = new Complex(0, 0);
     for (inner = 0; inner < dens.length; inner++) {
      Complex cCoef = new Complex(dens[dens.length - 1 - inner], 0);

      cCoef.mpy(cMult);
      cDen.add(cCoef);

      cMult.mpy(cFreq);
     }

     thePlot[outer] = cNum.div(cDen);
    }

   // return result
   return thePlot;
  }

}
```

4. NyquistFrame.java

```
package controlapplet;

import java.awt.*;
import javax.swing.*;
import scientific.graphics.JPlot2D;

/**
 * Implements the bode graph algorithm given the transfer function.
 *
 * @author Hüseyin CİĞEROĞLU
 * @version 1.0
 *
 */
public class NyquistFrame {
  protected JPlot2D plotMag;
  Complex[] theResult;
  double[] theFreqs;
  String numStr;
  String denStr;
  double maxDB = 0.0;

  public NyquistFrame(double x[], double y[]) {
    JFrame frame = new JFrame("Nyquist Plot");
    frame.getContentPane().setLayout(new BorderLayout());
    plotMag = new JPlot2D();

    double xDataP[] = new double[800];
    double yDataP[] = new double[800];
    double xDataZ[] = new double[800];
    double yDataZ[] = new double[800];
```

```
    plotMag.setGridState(true);
    plotMag.setXLabel("Re");
    plotMag.setYLabel("Im");
    plotMag.setTitle("Nyquist");

    plotMag.setLineColor(Color.blue);
    plotMag.setMarkerColor(Color.blue);
    plotMag.setMarkerState(true);
    plotMag.setLineState(plotMag.LINE_ON);

    int index = 0;

    theFreqs = new double[800];
    theFreqs[0] = 0.1;
    for (index = 0; index < 799; index++) {
      theFreqs[index + 1] = Math.pow(10.0, 0.005) * theFreqs[index];
    }

    theResult = plotAll(x, y);

    double[] xp = new double[theResult.length];
    double[] yp = new double[theResult.length];
    maxDB = -100.0;
    for (index = 0; index < theResult.length; index++) {
      yp[index] = theResult[index].real();
      xp[index] = theResult[index].imag();
    }

  //phase:
    plotMag.addCurve(yp, xp);
    plotMag.setLineColor(Color.blue);
    plotMag.setLineStyle(plotMag.LINESTYLE_SOLID);
    plotMag.setLineState(plotMag.LINE_ON);

    plotMag.setPreferredSize(new Dimension(600, 400));
    plotMag.setBackground(SystemColor.lightGray);
    plotMag.setBorder(BorderFactory.createLineBorder(Color.black, 2));
    plotMag.setVisible(true);

    frame.getContentPane().add(plotMag, "Center");
    frame.pack();
    frame.show();

  }

  public Complex[] plotAll(double[] nums, double[] dens) {
    if (nums.length == 0) {
      return null;
    }
    if (dens.length == 0) {
      return null;
    }
    if (theFreqs.length == 0) {
      return null;
    }

    Complex[] thePlot = new Complex[theFreqs.length];

    int outer = 0;
    for (outer = 0; outer < theFreqs.length; outer++) {
      Complex cFreq = new Complex(0, theFreqs[outer]);

      Complex cMult = new Complex(1, 0);
```

```
    Complex cNum = new Complex(0, 0);
    int inner = 0;
    for (inner = 0; inner < nums.length; inner++) {
     Complex cCoef = new Complex(nums[nums.length - 1 - inner], 0);

     cCoef.mpy(cMult);
     cNum.add(cCoef);

     cMult.mpy(cFreq);
    }

    cMult = new Complex(1, 0);

    Complex cDen = new Complex(0, 0);
    for (inner = 0; inner < dens.length; inner++) {
     Complex cCoef = new Complex(dens[dens.length - 1 - inner], 0);

     cCoef.mpy(cMult);
     cDen.add(cCoef);

     cMult.mpy(cFreq);
    }

    thePlot[outer] = cNum.div(cDen);
   }

   return thePlot;
  }

}
```

5. RK4.java

```
package controlapplet;

class RK4 {
  RK4(float f, float f1) {
    k = new double[ny][6];
    yval = new double[ny];
    yvalp = new double[nd];
    ytemp = new float[ny];
    u = new double[nd];
    xmin = f;
    xmax = f1;
  }

  void intRK4(int maxTime, int denOrder, double numx[], double denx[]) {
    num1 = numx;
    den1 = denx;
    int i = 0;
    float f = 0;
    float f1 = maxTime;
    nyy = denOrder - 1;
    float af[] = new float[ny];
    for (int j = 0; j < nyy; j++)
     af[j] = 0.0F;

    af[0] = 0.0F;
    hstep = (f1 - f) / (float) nstep;
```

```
  for (int l = 0; l < nyy; l++)
    yval[l] = af[l];

  for (int i1 = 0; i1 < nd; i1++) {
    yvalp[i1] = 0.0D;
    u[i1] = 0.0D;
  }

  for (float f2 = f; f2 <= f1; f2 += hstep) {
    ytemp = func(f2);
    float f3 = ytemp[nyy - 1] + (float) num1[denOrder - 1];
    yst[i][0] = f3;
    xst[i] = f2;
    i++;
    for (int j1 = 0; j1 < nd - 1; j1++) {
      yvalp[nd - 1 - j1] = yvalp[nd - 2 - j1];
      u[nd - 1 - j1] = u[nd - 2 - j1];
    }

    yvalp[0] = f3;
  }
}

float[] func(float f) {
  float af[] = new float[nyy];
  ctime = f;
  sumk(1, 0.0F, 0.0D, 0.0D, 0.0D, 0.0D, 0.0D);
  sumk(2, 0.25F, 0.25D, 0.0D, 0.0D, 0.0D, 0.0D);
  sumk(3, 0.375F, 0.09375D, 0.28125D, 0.0D, 0.0D, 0.0D);
  sumk(4, 0.9230769F, 0.8793809740555 3028D, -3.2771961766044608D,
       3.3208921256258535D, 0.0D, 0.0D);
  sumk(5, 1.0F, 2.0324074074074074D, -8D, 7.1734892787524362D,
       -0.20589668615984405D, 0.0D);
  sumk(6, 0.5F, -0.2962962962962962 8D, 2D, -1.3816764132553607D,
       0.45297270955165692D, -0.27500000000000002D);
  ctime = ctime + hstep;
  sumk(0, 0.0F, 0.11574074074074074D, 0.0D, 0.54892787524366471D,
       0.53533138401559455D, -0.20000000000000001D);
  for (int i = 0; i < nyy; i++)
    af[i] = (float) yval[i];

  return af;
}

void sumk(int i, float f, double d, double d1, double d2, double d3,
          double d4) {
  double ad[] = new double[nyy];
  float af[] = new float[nyy];
  for (int j = 0; j < nyy; j++) {
    double sum=yval[j] + d * k[j][0] + d1 * k[j][1] + d2 * k[j][2] + d3 * k[j][3] +
        d4 * k[j][4];
    if (sum>1e30)
      ad[j] = 1e30;
    else
      ad[j] = sum;
    if (i == 0)
      yval[j] = ad[j];
  }

  if (i == 0)
    return;
  evalf(ctime + f * hstep, ad, af);
  for (int l = 0; l < nyy; l++)
```

```
      k[l][i - 1] = hstep * af[l];

    }

    void evalf(float f, double ad[], float af[]) {
      int i = nyy - 1;
      float sum=(float) ( (double) ( -den1[0]) * ad[i] + (double) num1[0]);
      if (sum>1e30)
        af[0] = (float)1e30;
      else
        af[0]=sum;
      for (int j = 1; j <= i; j++) {
        af[j] = (float) ( (ad[j - 1] - (double) den1[j] * ad[i]) +(double) num1[j]);
        if (sum>1e30)
          af[j] = (float)1e30;
        else
          af[j]=sum;

      }
    }

    static int nstep;
    static int mxplot;
    static int ny = 13;
    static int nd = 2;
    static double yst[][];
    static double xst[];
    static float xmin;
    static float xmax;
    int nyy;
    float ctime;
    float hstep;
    double k[][];
    double yval[];
    double yvalp[];
    float ytemp[];
    double u[];
    double num1[];
    double den1[];

    static {
      nstep = 200;
      mxplot = 1;
      yst = new double[nstep + 1][mxplot];
      xst = new double[nstep + 1];
    }
}
```

6. Tf2ss.java

```
package controlapplet;

/**
 * Implements the transfer function to state space conversion.
 *
 * @author Hüseyin CİĞEROĞLU
 * @version 1.0
 *
 */
public class Tf2ss {
```

```
/**
 * Constructor of the Zp2tf class.
 */
public Tf2ss() {
}

/**
 * Converts transfer function systems to state space systems.
 *
 * @param num numerator of the transfer function
 * @param den denominator of the transfer function
 * @return true : if success
 */
public boolean calculateTF2SS(double[] num, double[] den) {
  int mnum = 1;
  int nnum = num.length;

  int mden = 1;
  int n = den.length;

  double numNew[] = new double[n];
  double denNew[] = new double[n - 1];

  //if numerator size is bigger then denominator return false
  if (nnum > n) {
    return false;
  }

  //make numerator same size with denominator
  if (n - nnum > 0) {
    for (int k = nnum; k < n; k++)
      numNew[k] = 0;
  }

  //normalize numerator to denominator(0)
  for (int k = 0; k < num.length; k++)
    numNew[k] = num[k] / den[n - 1];

    // the d matrix
  D = new Matrix(1, 1);
  D.set(0, 0, numNew[n - 1]);

  //special case
  if (n == 1) {
    return true;
  }

  //normalize den
  A = new Matrix(n - 1, n - 1);
  B = new Matrix(n - 1, 1);
  C = new Matrix(1, n - 1);
  for (int k = 0; k < n - 1; k++) {
    for (int l = 0; l < n - 1; l++) {
      A.set(k, l, 0);
      B.set(k, 0, 0);
      C.set(0, k, 0);
    }
  }

  B.set(0, 0, 1);

  for (int k = 0; k < n - 1; k++) {
    denNew[k] = den[k] / den[n - 1];
```

```
      A.set(0, n - k - 2, -denNew[k]);
    }

    //put the identity matrix in the remaining rows
    for (int k = 0; k < n - 2; k++) {
      A.set(k + 1, k, 1);
    }

    if (nnum > 0) {
      for (int k = 1; k < n; k++)
        C.set(0, k - 1, numNew[n - k - 1] - numNew[n - 1] * denNew[n - k - 1]);
    }

    return true;
  }

  /**
   * Resulting A matrix.
   */
  public Matrix A;

  /**
   * Resulting B matrix.
   */
  public Matrix B;

  /**
   * Resulting C matrix.
   */
  public Matrix C;

  /**
   * Resulting D matrix.
   */
  public Matrix D;

}
```

7. Zp2tf.java

```
package controlapplet;

import java.util.Vector;
import controlapplet.ComplexNumber;

/**
 * Implements the zero-pole to transfer function conversion.
 *
 * @author Hüseyin CİĞEROĞLU
 * @version 1.0
 *
 */
public class Zp2tf {

  /**
   * Constructor of the Zp2tf class.
   */
  public Zp2tf() {
  }

  /**
```

```
 * Converts zero-pole systems to transfer function systems.
 *
 * @param zpPoints the zeros and poles of the system
 * @param k the gain
 * @return true : if success
 */
public boolean calculateZP2TF(Vector zpPoints, int k) {
  k = 1;
  int numb = zpPoints.size();
  int numberZ = 0;
  int numberP = 0;

  for (int u = 0; u < numb; u++) {
    if ( ( (ComplexNumber) zpPoints.elementAt(u)).getGain() == -1) {
      if ( ( (ComplexNumber) zpPoints.elementAt(u)).getImag() != 0) {
        numberZ++;
      }
      numberZ++;
    }
    else {
      if ( ( (ComplexNumber) zpPoints.elementAt(u)).getImag() != 0) {
        numberP++;
      }
      numberP++;
    }
  }

  Vector p = new Vector(numberP);
  Vector z = new Vector(numberZ);

  for (int u = 0; u < numb; u++) {
    if ( ( (ComplexNumber) zpPoints.elementAt(u)).getGain() == -1) {
      z.addElement(zpPoints.elementAt(u));
      if ( ( (ComplexNumber) zpPoints.elementAt(u)).getImag() != 0) {
        ComplexNumber c = (ComplexNumber) zpPoints.elementAt(u);
        c.setImag(c.getImag() * -1);
        z.addElement(c);
      }
    }
    else {
      p.addElement(zpPoints.elementAt(u));
      if ( ( (ComplexNumber) zpPoints.elementAt(u)).getImag() != 0) {
        ComplexNumber c = new ComplexNumber( ( (ComplexNumber) p.elementAt(p.
            size() - 1)).getReal(),
                                    ( (ComplexNumber) p.elementAt(p.
            size() - 1)).getImag() * -1);
        p.addElement(c);
      }
    }
  }

  den = constructPoly(p);
  num = constructPoly(z);
  return true;
}

/**
 * Constructs polynomials from given roots.
 *
 * @param a : A vector consisting of the roots
 */
public double[] constructPoly(Vector a) {
  int order = a.size();
```

```
    ComplexNumber den1[] = new ComplexNumber[order + 1];
    for (int t = 0; t < order + 1; t++) {
      den1[t] = new ComplexNumber();
    }
    den1[order].setReal(1);
    for (int t = 0; t < order; t++) {
      for (int l = t; l >= 0; l--) {
        den1[order - l - 1] = den1[order - l -
            1].minus( ( (ComplexNumber) a.
                    elementAt(t)).mul(new ComplexNumber(den1[order - l])));
      }
    }

    double polynom[] = new double[order + 1];
    for (int t = 0; t < order + 1; t++) {
      polynom[t] = (den1[t]).getReal();
    }

    return polynom;
  }

  /**
   * Resulting denominator.
   */
  public double[] den;

  /**
   * Resulting numerator.
   */
  public double[] num;
}
```

8. Rlocus.java

```
package controlapplet;

/**
 * Implements the root locus finding algorithm.
 *
 * @author Hüseyin CİĞEROĞLU
 * @version 1.0
 *
 */
public class Rlocus {

  /**
   * Constructor of the Rlocus class.
   */
  public Rlocus() {
  }

  /**
   * Calculates the root locus of the given state space system.
   *
   * @param a the A matrix
   * @param b the B matrix
   * @param c the C matrix
   * @param d the D matrix
   * @param k the gain array to calculate
   * @return true : if success
   */
```

```
  public boolean findRlocus(Matrix a, Matrix b, Matrix c, double d, double[] k) {
    int ns = b.getRowDimension();
    int nu = b.getColumnDimension();
    int nk = k.length;
    Matrix r = new Matrix(nu, nk);//(ns, nk);

    b = b.times(c);
    int i = 0;

    for (int t = 0; t < k.length; t++) {
      k[t] = k[t] / (1 + k[t] * d);
    }

    rReal = new Matrix(a.getColumnDimension(), k.length);//Row
    rImag = new Matrix(a.getColumnDimension(), k.length);

    for (int t = 0; t < k.length; t++) {
      double im[];
      double re[];
      i++;
      EigenvalueDecomposition eigens = (a.minus(b.times(k[t]))).eig();
      im = eigens.getImagEigenvalues();
      re = eigens.getRealEigenvalues();
      for (int l = 0; l < re.length; l++) { //im.length
        rReal.set(l, t, re[l]);
        rImag.set(l, t, im[l]);
      }
    }
    return true;
  }

  /**
   * The imaginary part of the root locus.
   */
  public Matrix rImag;

  /**
   * The real part of the root locus.
   */
  public Matrix rReal;
}
```

9. PZSelectPanel.java

```
package controlapplet;

import scientific.graphics.JPlot2D;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.util.StringTokenizer;
import parameter.params;

/**
 * Implements the pole zero and root locus containing graph.
 * Poles and zeros can be entered via mouse and root locus is
 * calculated automatically.
 *
 * @author Hüseyin CİĞEROĞLU
 * @version 1.0
 *
```

```
 */
public class PZSelectPanel
    extends JPanel
    implements MouseListener, MouseMotionListener {
  public PZSelectPanel(MainPanel app) {
    applet = app;
    moving_point = -1;
    plot = new JPlot2D();
    points = new Vector(16, 3);
    f = java.text.NumberFormat.getInstance(java.util.Locale.US);
    if (f instanceof java.text.DecimalFormat)
      ( (java.text.DecimalFormat) f).setMaximumFractionDigits(1);

    StringTokenizer st = new StringTokenizer(params.POLES);
    while (st.hasMoreTokens()) {
      String pValue = st.nextToken();
      if (pValue.indexOf("j") == -1) {
        ComplexNumber controlpoint;
        points.addElement(controlpoint = new ComplexNumber(
            new ComplexNumber(Double.parseDouble(pValue), 0), 1));
      }
      else {
        if (pValue.indexOf(",") == -1) {
          ComplexNumber controlpoint;
          points.addElement(controlpoint = new ComplexNumber(
              new ComplexNumber(0, Double.parseDouble(pValue.substring(0,
              pValue.length() - 1))), 1));
        }
        else {
          ComplexNumber controlpoint;
          points.addElement(controlpoint = new ComplexNumber(
              new ComplexNumber(Double.parseDouble(pValue.substring(0,
              pValue.indexOf(","))),
                        Double.parseDouble(pValue.substring(pValue.
              indexOf(",") + 1,
              pValue.length() - 1))), 1));
        }
      }
    }

    st = new StringTokenizer(params.ZEROS);
    while (st.hasMoreTokens()) {
      String zValue = st.nextToken();
      if (zValue.indexOf("j") == -1) {
        ComplexNumber controlpoint;
        points.addElement(controlpoint = new ComplexNumber(
            new ComplexNumber(Double.parseDouble(zValue), 0), -1));
      }
      else {
        if (zValue.indexOf(",") == -1) {
          ComplexNumber controlpoint;
          points.addElement(controlpoint = new ComplexNumber(
              new ComplexNumber(0, Double.parseDouble(zValue.substring(0,
              zValue.length() - 1))), -1));
        }
        else {
          ComplexNumber controlpoint;
          points.addElement(controlpoint = new ComplexNumber(
              new ComplexNumber(Double.parseDouble(zValue.substring(0,
              zValue.indexOf(","))),
                        Double.parseDouble(zValue.substring(zValue.
              indexOf(",") + 1,
              zValue.length() - 1))), -1));
```

```
    }
   }
  }


  xDataP = new double[12];
  yDataP = new double[12];
  xDataZ = new double[12];
  yDataZ = new double[12];

  this.setLayout( (new BorderLayout()));
  plot.setGridState(true);

  plot.setXLabel("Re");
  plot.setYLabel("Im");
  plot.setTitle("Poles & Zeros");
  double a[] = new double[2];
  a[0] = -10;
  a[1] = 10;
  plot.setXScale(a);
  plot.setYScale(a);

  plot.setLineColor(Color.blue);
  plot.setMarkerColor(Color.blue);
  plot.setMarkerState(true);
  plot.setMarkerStyle(plot.MARKER_SQUARE);
  plot.setLineState(plot.LINE_OFF);
  plot.addCurve(xDataP, yDataP);

  plot.setLineColor(Color.red);
  plot.setMarkerColor(Color.red);
  plot.setMarkerState(true);
  plot.setMarkerStyle(plot.MARKER_CIRCLE);
  plot.setLineState(plot.LINE_OFF);

  plot.addCurve(xDataZ, yDataZ);

  plot.setBackground(SystemColor.yellow);

  plot.setBorder(BorderFactory.createLineBorder(Color.black, 2));
  plot.setPreferredSize(new Dimension(100, 100));
  plot.setVisible(true);
  plot.setAllowMouseClick(isMouseClickable);
  plot.setMouseCursorChange(isMouseClickable);

  plot.addMouseListener(this);
  plot.addMouseMotionListener(this);
  this.add(plot, BorderLayout.CENTER);
 }

 public void PZRepaint() {
  plot.repaint();
 }

 public boolean getIsMouseClickable() {
  return isMouseClickable;
 }

 public void mouseClicked(MouseEvent e) {
  Object ob = e.getSource();
  if (getIsMouseClickable() && (ob instanceof JPlot2D)) {
    getPlot().mouseClicked(e);
    if (getPlot().getCurrentPoint() != null) {
```

```
      PZRepaint();
    }
  }
}

public void mousePressed(MouseEvent mouseevent) {

  getPlot().mouseClicked(mouseevent);
  double pointX = Double.parseDouble(f.format(getPlot().getCurrentPoint().
                                      getVecX()));
  double pointY = Double.parseDouble(f.format(getPlot().getCurrentPoint().
                                      getVecY()));

  //if MOVE action, then find the point and set it
  if (actionType == 3) {
    if (findPoint(pointX, pointY) >= 0)
      moving_point = findPoint(pointX, pointY);
  }

  //if DELETE
  else if (actionType == 4) {
    int l2 = findPoint(pointX, pointY);
    if (l2 >= 0) {
      points.removeElementAt(l2);
      update();
    }
  }
  //if ADDPOLE or ADDZERO then add it
  else if ( (actionType == 1) || (actionType == 2)) {
    if (parameter.params.IMAG_FIXED) {
      pointY = 0;
    }
    if (parameter.params.REAL_FIXED) {
      pointX = 0;
    }
    int numberOfPoints = points.size();
    int counter = 1;

    switch (actionType) {
      case 1: { //add pole
        for (int j2 = 0; j2 < numberOfPoints; j2++) {
          ComplexNumber controlpoint5 = (ComplexNumber) points.elementAt(j2);
          if (pointX == controlpoint5.a && pointY == controlpoint5.b &&
              controlpoint5.gain > 0)
            counter++;
        }

        ComplexNumber controlpoint;
        points.addElement(controlpoint = new ComplexNumber(new ComplexNumber(
            pointX, pointY
            ), counter, parameter.params.IMAG_FIXED,
                    parameter.params.REAL_FIXED));
      }
      break;
      case 2: { //add zero
        for (int k2 = 0; k2 < numberOfPoints; k2++) {
          ComplexNumber controlpoint6 = (ComplexNumber) points.elementAt(k2);
          if (pointX == controlpoint6.a && pointY == controlpoint6.b &&
              controlpoint6.gain < 0)
            counter++;
        }

        ComplexNumber controlpoint1;
```

```
        points.addElement(controlpoint1 = new ComplexNumber(new ComplexNumber(
            pointX, pointY
            ), -counter, parameter.params.IMAG_FIXED,
                    parameter.params.REAL_FIXED));
      }
      break;

    }

    update();
  }
}

public void update() {
  int i1 = 0;
  int j1 = 0;
  int i = points.size();

  plot.removeAll();
  plot.setGridState(true);
  plot.setXLabel("Re");
  plot.setYLabel("Im");
  plot.setTitle("Poles & Zeros");
  plot.setBackground(SystemColor.yellow);

  //poles
  for (int c = 0; c < 12; c++) {
    xDataP[c] = 0;
    yDataP[c] = 0;
    xDataZ[c] = 0;
    yDataZ[c] = 0;
  }

  for (int i3 = 0; i3 < i; i3++) {
    ComplexNumber controlpoint1 = (ComplexNumber) points.elementAt(i3);
    if (controlpoint1.gain > 0) {
      xDataP[i1] = controlpoint1.a;
      yDataP[i1] = controlpoint1.b;
      i1++;

      if (controlpoint1.b != 0) {
        xDataP[i1] = controlpoint1.a;
        yDataP[i1] = ( -1 * controlpoint1.b);
        i1++;
      }
    }
  }

  //zeros
  for (int j3 = 0; j3 < i; j3++) {
    ComplexNumber controlpoint1 = (ComplexNumber) points.elementAt(j3);
    if (controlpoint1.gain < 0) {
      xDataZ[j1] = controlpoint1.a;
      yDataZ[j1] = controlpoint1.b;

      j1++;
      if (controlpoint1.b != 0) {
        xDataZ[j1] = controlpoint1.a;
        yDataZ[j1] = ( -1 * controlpoint1.b);
        j1++;
      }
    }
  }
```

```
int pointN = 0;
for (int numP = 0; numP < 12; numP++) {
  if (xDataP[numP] != 0 || yDataP[numP] != 0)
    pointN++;
}
//create dynamically new array, so that no image appears at origin
//for poles
double xDataPr[] = new double[pointN];
double yDataPr[] = new double[pointN];
for (int numP = 0; numP < pointN; numP++) {
  xDataPr[numP] = xDataP[numP];
  yDataPr[numP] = yDataP[numP];
}
plot.addCurve(xDataPr, yDataPr);
plot.setLineColor(Color.blue);
plot.setMarkerColor(Color.blue);
plot.setMarkerState(true);
plot.setMarkerStyle(plot.MARKER_SQUARE);
plot.setLineState(plot.LINE_OFF);
getPlot().repaint();
//for zeros
pointN = 0;
for (int numP = 0; numP < 12; numP++) {
  if (xDataZ[numP] != 0 || yDataZ[numP] != 0)
    pointN++;
}
double xDataZr[] = new double[pointN];
double yDataZr[] = new double[pointN];
for (int numP = 0; numP < pointN; numP++) {
  xDataZr[numP] = xDataZ[numP];
  yDataZr[numP] = yDataZ[numP];
}
plot.addCurve(xDataZr, yDataZr);
plot.setLineColor(Color.red);
plot.setMarkerColor(Color.red);
plot.setMarkerState(true);
plot.setMarkerStyle(plot.MARKER_CIRCLE);
plot.setLineState(plot.LINE_OFF);
getPlot().repaint();
getPlot().setVisible(true);

//find tf from zp, set num and den

double af[] = new double[13];
double af1[] = new double[13];
double af2[] = new double[3];
double af3[] = {
    1.0F
};
i = points.size();
nden = equal(den, af3);
int j3 = equal(num, af3);
for (int j4 = 0; j4 < i; j4++) {
  ( (ComplexNumber) points.elementAt(j4)).a = ( (ComplexNumber) points.
      elementAt(j4)).a * -1;
  ( (ComplexNumber) points.elementAt(j4)).b = ( (ComplexNumber) points.
      elementAt(j4)).b * -1;
}

 for (int j4 = 0; j4 < i; j4++) {
   ( (ComplexNumber) points.elementAt(j4)).a = ( (ComplexNumber) points.
       elementAt(j4)).a * -1;
```

```
  ( (ComplexNumber) points.elementAt(j4)).b = ( (ComplexNumber) points.
      elementAt(j4)).b * -1;
 }

Vector vector3 = new Vector(1);
Vector vector4 = new Vector(4);

for (int n = 0; n < 13; n++) {
  vector3.addElement(new ComplexNumber(num[n], 0));
  vector4.addElement(new ComplexNumber(den[n], 0));
}

Zp2tf con = new Zp2tf();
con.calculateZP2TF(points, 1);

Tf2ss calc = new Tf2ss();
calc.calculateTF2SS(con.num, con.den);
parameter.params.num = con.num;
parameter.params.den = con.den;

double k[] = new double[200];
for (int o = 0; o < 200; o++) {
  k[o] = o;
}
if ( (points.size() == 2) &&
    ( ( ( (ComplexNumber) (points.elementAt(1))).gain == -1)
     && ( ( (ComplexNumber) (points.elementAt(0))).gain == 1) ||
     ( ( (ComplexNumber) (points.elementAt(0))).gain == -1)
     && ( ( (ComplexNumber) (points.elementAt(1))).gain == 1))
    && ( ( (ComplexNumber) (points.elementAt(0))).b == 0)
    && ( ( (ComplexNumber) (points.elementAt(1))).b == 0)) {
  double[] xLocus = new double[199];
  double[] yLocus = new double[199];
  for (int q1 = 0; q1 < 199; q1++) {
    xLocus[q1] = ( (ComplexNumber) (points.elementAt(0))).a -
        ( ( (ComplexNumber) (points.elementAt(0))).a -
         ( (ComplexNumber) (points.elementAt(1))).a) * q1 / 199;
    yLocus[q1] = 0;
  }
  plot.setPlotType(plot.LINEAR);
  plot.setLineState(plot.LINE_ON);
  plot.addCurve(xLocus, yLocus);
  plot.setLineColor(Color.GREEN);
  plot.setLineWidth(2);
}
else {
  Rlocus rlocus = new Rlocus();

  applet.setTextAreaSS(calc.A, calc.B, calc.C);
  rlocus.findRlocus(calc.A, calc.B, calc.C, 0, k);

  double[] xLocus = new double[199];
  double[] yLocus = new double[199];
  ;
  double[] xLocus1 = new double[199];
  double[] yLocus1 = new double[199];
  ;
  double[] xLocus2 = new double[199];
  double[] yLocus2 = new double[199];
  ;
  double[] xLocus3 = new double[199];
  double[] yLocus3 = new double[199];
  ;
```

```
double[] xLocus4 = new double[199];
double[] yLocus4 = new double[199];
;
double[] xLocus5 = new double[199];
double[] yLocus5 = new double[199];
;
double[] xLocus6 = new double[199];
double[] yLocus6 = new double[199];
;
double[] xLocus7 = new double[199];
double[] yLocus7 = new double[199];
;

for (int q = 0; q < rlocus.rReal.getRowDimension(); q++) {
 for (int w = 0; w < rlocus.rReal.getColumnDimension() - 1; w++) {
  if (q == 0) {
   xLocus[w] = rlocus.rReal.get(q, w + 1);
   yLocus[w] = rlocus.rImag.get(q, w + 1);
  }
  if (q == 1) {
   xLocus1[w] = rlocus.rReal.get(q, w + 1);
   yLocus1[w] = rlocus.rImag.get(q, w + 1);
  }
  if (q == 2) {
   xLocus2[w] = rlocus.rReal.get(q, w + 1);
   yLocus2[w] = rlocus.rImag.get(q, w + 1);
  }
  if (q == 3) {
   xLocus3[w] = rlocus.rReal.get(q, w + 1);
   yLocus3[w] = rlocus.rImag.get(q, w + 1);
  }
  if (q == 4) {
   xLocus4[w] = rlocus.rReal.get(q, w + 1);
   yLocus4[w] = rlocus.rImag.get(q, w + 1);
  }
  if (q == 5) {
   xLocus5[w] = rlocus.rReal.get(q, w + 1);
   yLocus5[w] = rlocus.rImag.get(q, w + 1);
  }
  if (q == 6) {
   xLocus6[w] = rlocus.rReal.get(q, w + 1);
   yLocus6[w] = rlocus.rImag.get(q, w + 1);
  }
  if (q == 7) {
   xLocus7[w] = rlocus.rReal.get(q, w + 1);
   yLocus7[w] = rlocus.rImag.get(q, w + 1);
  }
 }
 if (q == 0) {

  plot.setPlotType(plot.LINEAR);
  plot.setLineState(plot.LINE_ON);
  plot.addCurve(xLocus, yLocus);
  plot.setLineColor(Color.GREEN);
  plot.setLineWidth(2);
 }
 if (q == 1) {
  plot.setPlotType(plot.LINEAR);
  plot.setLineState(plot.LINE_ON);
  plot.addCurve(xLocus1, yLocus1);
  plot.setLineColor(Color.BLACK);
  plot.setLineWidth(2);
 }
```

```
      if (q == 2) {
        plot.setPlotType(plot.LINEAR);
        plot.setLineState(plot.LINE_ON);
        plot.addCurve(xLocus2, yLocus2);
        plot.setLineColor(Color.MAGENTA);
        plot.setLineWidth(2);
      }
      if (q == 3) {
        plot.setPlotType(plot.LINEAR);
        plot.setLineState(plot.LINE_ON);
        plot.addCurve(xLocus3, yLocus3);
        plot.setLineColor(Color.CYAN);
        plot.setLineWidth(2);
      }
      if (q == 4) {
        plot.setPlotType(plot.LINEAR);
        plot.setLineState(plot.LINE_ON);
        plot.addCurve(xLocus4, yLocus4);
        plot.setLineColor(Color.BLUE);
        plot.setLineWidth(2);
      }
      if (q == 5) {
        plot.setPlotType(plot.LINEAR);
        plot.setLineState(plot.LINE_ON);
        plot.addCurve(xLocus5, yLocus5);
        plot.setLineColor(Color.PINK);
        plot.setLineWidth(2);
      }
      if (q == 6) {
        plot.setPlotType(plot.LINEAR);
        plot.setLineState(plot.LINE_ON);
        plot.addCurve(xLocus6, yLocus6);
        plot.setLineColor(Color.LIGHT_GRAY);
        plot.setLineWidth(2);
      }
      if (q == 7) {
        plot.setPlotType(plot.LINEAR);
        plot.setLineState(plot.LINE_ON);
        plot.addCurve(xLocus7, yLocus7);
        plot.setLineColor(Color.ORANGE);
        plot.setLineWidth(2);
      }
     }
    }
    getPlot().repaint();
    getPlot().setVisible(true);

    double[] numTF = new double[num.length];
    double[] denTF = new double[den.length];
    for (int order = 0; order < num.length; order++) {
      numTF[order] = num[order];
    }
    for (int order = 0; order < den.length; order++) {
      denTF[order] = den[order];
    }

    applet.setTextArea(xDataP, yDataP, xDataZ, yDataZ);
    applet.SetSimParameters(con.num, con.den);
  }

  public void mouseDragged(MouseEvent mouseevent) {
    int i = points.size();
    getPlot().mouseDragged(mouseevent);
```

```
double pointX = Double.parseDouble(f.format(getPlot().getCurrentPoint().
                                    getVecX()));
double pointY = Double.parseDouble(f.format(getPlot().getCurrentPoint().
                                    getVecY()));
applet.setCoords(pointX, pointY);

if (moving_point >= 0) {
 //control, whether we are within the graph:
 if (pointX > 10)
  pointX = 10;
 if (pointX < -10)
  pointX = -10;
 if (pointY > 10)
  pointY = 10;
 if (pointY < -10)
  pointY = -10;
 ComplexNumber controlpoint2 = (ComplexNumber) points.elementAt(
    moving_point);
 double i3 = controlpoint2.a;
 double j3 = controlpoint2.b;

 if (controlpoint2.imagFixed) {
  pointX = 0;
 }
 if (controlpoint2.realFixed) {
  pointY = 0;
 }

 int k3 = (int) controlpoint2.gain;
 if (actionType == 3) {
  controlpoint2.a = pointX;
  controlpoint2.b = pointY;
 }
 else

 if (j3 == 0) {
  if (actionType == 3 && Math.abs(k3) == 1)
   controlpoint2.b = 0;
  else
  if (controlpoint2.b != 0) {
   int l3 = 1;
   for (int j4 = 0; j4 < l3; j4++) {
    int i5 = findPoint(i3, j3);
    if (i5 >= 0 && i > 1) {
     points.removeElementAt(i5);
     i = points.size();
    }
   }

   moving_point = findPoint(i3, j3);
   if (moving_point >= 0) {
    ComplexNumber controlpoint3 = (ComplexNumber) points.elementAt(
       moving_point);
    controlpoint3.a = controlpoint2.a;
    controlpoint3.b = controlpoint2.b;
    controlpoint3.gain = k3 <= 0 ? k3 + l3 : k3 - l3;
   }
  }
 }
 else
 if (controlpoint2.b == 0 && i < npmax - 1) {
  int l1 = (int) (2 * k3);
  for (int i4 = 0; i4 < Math.abs(l1 - k3); i4++) {
```

```
      int k4 = k3 <= 0 ? - (i4 + 1) : i4 + 1;
      points.addElement(controlpoint2 = new ComplexNumber(new ComplexNumber(
          controlpoint2.a,
          controlpoint2.b), k4));
      i = points.size();
     }

     moving_point = findPoint(controlpoint2.a, controlpoint2.b);
    }
    int i2 = 1;
    boolean flag = false;
    if (controlpoint2.b != 0 &&
        controlpoint2.gain - (controlpoint2.gain / 2) * 2 != 0)
     flag = true;
    for (int l4 = 0; l4 < i; l4++)
     if (l4 != moving_point) {
      ComplexNumber controlpoint7 = (ComplexNumber) points.elementAt(l4);
      if (controlpoint2.a == controlpoint7.a &&
          controlpoint2.b == controlpoint7.b &&
          controlpoint7.gain * controlpoint2.gain > 0) {
       if (controlpoint7.gain > 0)
        controlpoint7.gain = i2;
       else
        controlpoint7.gain = -i2;
       i2++;
      }
     }

    if (controlpoint2.gain > 0)
     controlpoint2.gain = i2;
    else
     controlpoint2.gain = -i2;
    repaint();
    update();
   }
  }

  public void mouseMoved(MouseEvent mouseevent) {
   getPlot().mouseMoved(mouseevent);
   if (getPlot().getCurrentPoint() != null) {
    double pointX = Double.parseDouble(f.format(getPlot().getCurrentPoint().
                                        getVecX()));
    double pointY = Double.parseDouble(f.format(getPlot().getCurrentPoint().
                                        getVecY()));
    applet.setCoords(pointX, pointY);
   }
  }

  public void mouseReleased(MouseEvent mouseevent) {
   if (actionType == 3) {
    moving_point = -1;
   }
  }

  public void mouseEntered(MouseEvent mouseevent) {
  }

  public void mouseExited(MouseEvent mouseevent) {
  }

  public JPlot2D getPlot() {
   return plot;
  }
```

```
private int findPoint(double i, double j) {
  int k = points.size();
  int l = -1;
  double i1 = 1;
  for (int j1 = 0; j1 < k; j1++) {
    ComplexNumber controlpoint = (ComplexNumber) points.elementAt(j1);
    if ( (controlpoint.within(i, j) ||
          controlpoint.within(i, -j)) &&
        Math.abs(controlpoint.gain) >= i1) {
      i1 = Math.abs(controlpoint.gain);
      l = j1;
    }
  }
  return l;
}

int equal(double af[], double af1[]) {
  int i = -1;
  for (int j = af.length - 1; j >= af1.length; j--)
    af[j] = 0.0F;

  for (int k = af1.length - 1; k >= 0; k--) {
    if (af1[k] != 0.0F && i < 0)
      i = k;
    af[k] = af1[k];
  }
  return i;
}

double[] conv(double af[], double af1[], int i) {
  double af2[] = new double[13];
  int l = af1.length;
  if (af1[l - 1] == 0.0F)
    l--;
  for (int i1 = 0; i1 < i + l; i1++) {
    af2[i1] = 0.0F;
    int j = Math.max(0, (i1 - l) + 1);
    int k = Math.min(i, i1);
    for (int j1 = j; j1 <= k; j1++)
      af2[i1] = af2[i1] + af[j1] * af1[i1 - j1];
  }
  return af2;
}

public void clearPoints() {
  points.removeAllElements();
}

protected MainPanel applet;
protected JPlot2D plot;
double xData[], yData[];
double xDataP[], yDataP[];
double xDataZ[], yDataZ[];
private boolean isMouseClickable = true;
public int actionType = 1;
private Vector points;

public int maxZeros = 6;
public int maxPoles = 6;

private int moving_point;
```

```
  static int npmax;
  static double den[];
  static double num[];
  static int nden;

  public java.text.NumberFormat f;

  static {
    npmax = 6;
    den = new double[npmax * 2 + 1];
    num = new double[npmax * 2 + 1];
  }
}
```

10. EigenValueDecomposition.java

```
package Jama;
import Jama.util.*;

/** Eigenvalues and eigenvectors of a real matrix.
<P>
   If A is symmetric, then A = V*D*V' where the eigenvalue matrix D is
   diagonal and the eigenvector matrix V is orthogonal.
   I.e. A = V.times(D.times(V.transpose())) and
   V.times(V.transpose()) equals the identity matrix.
<P>
   If A is not symmetric, then the eigenvalue matrix D is block diagonal
   with the real eigenvalues in 1-by-1 blocks and any complex eigenvalues,
   lambda + i*mu, in 2-by-2 blocks, [lambda, mu; -mu, lambda].  The
   columns of V represent the eigenvectors in the sense that A*V = V*D,
   i.e. A.times(V) equals V.times(D).  The matrix V may be badly
   conditioned, or even singular, so the validity of the equation
   A = V*D*inverse(V) depends upon V.cond().
**/

public class EigenvalueDecomposition implements java.io.Serializable {

/* ------------------------
   Class variables
 * ------------------------ */

   /** Row and column dimension (square matrix).
   @serial matrix dimension.
   */
   private int n;

   /** Symmetry flag.
   @serial internal symmetry flag.
   */
   private boolean issymmetric;

   /** Arrays for internal storage of eigenvalues.
   @serial internal storage of eigenvalues.
   */
   private double[] d, e;

   /** Array for internal storage of eigenvectors.
   @serial internal storage of eigenvectors.
   */
   private double[][] V;
```

```
   /** Array for internal storage of nonsymmetric Hessenberg form.
   @serial internal storage of nonsymmetric Hessenberg form.
   */
   private double[][] H;

   /** Working storage for nonsymmetric algorithm.
   @serial working storage for nonsymmetric algorithm.
   */
   private double[] ort;

/* ------------------------
   Private Methods
 * ------------------------ */

   // Symmetric Householder reduction to tridiagonal form.

   private void tred2 () {

   //  This is derived from the Algol procedures tred2 by
   //  Bowdler, Martin, Reinsch, and Wilkinson, Handbook for
   //  Auto. Comp., Vol.ii-Linear Algebra, and the corresponding
   //  Fortran subroutine in EISPACK.

      for (int j = 0; j < n; j++) {
        d[j] = V[n-1][j];
      }

      // Householder reduction to tridiagonal form.

      for (int i = n-1; i > 0; i--) {

        // Scale to avoid under/overflow.

        double scale = 0.0;
        double h = 0.0;
        for (int k = 0; k < i; k++) {
          scale = scale + Math.abs(d[k]);
        }
        if (scale == 0.0) {
          e[i] = d[i-1];
          for (int j = 0; j < i; j++) {
            d[j] = V[i-1][j];
            V[i][j] = 0.0;
            V[j][i] = 0.0;
          }
        } else {

          // Generate Householder vector.

          for (int k = 0; k < i; k++) {
            d[k] /= scale;
            h += d[k] * d[k];
          }
          double f = d[i-1];
          double g = Math.sqrt(h);
          if (f > 0) {
            g = -g;
          }
          e[i] = scale * g;
          h = h - f * g;
          d[i-1] = f - g;
          for (int j = 0; j < i; j++) {
            e[j] = 0.0;
```

```
      }

      // Apply similarity transformation to remaining columns.

      for (int j = 0; j < i; j++) {
        f = d[j];
        V[j][i] = f;
        g = e[j] + V[j][j] * f;
        for (int k = j+1; k <= i-1; k++) {
          g += V[k][j] * d[k];
          e[k] += V[k][j] * f;
        }
        e[j] = g;
      }
      f = 0.0;
      for (int j = 0; j < i; j++) {
        e[j] /= h;
        f += e[j] * d[j];
      }
      double hh = f / (h + h);
      for (int j = 0; j < i; j++) {
        e[j] -= hh * d[j];
      }
      for (int j = 0; j < i; j++) {
        f = d[j];
        g = e[j];
        for (int k = j; k <= i-1; k++) {
          V[k][j] -= (f * e[k] + g * d[k]);
        }
        d[j] = V[i-1][j];
        V[i][j] = 0.0;
      }
    }
    d[i] = h;
  }

  // Accumulate transformations.

  for (int i = 0; i < n-1; i++) {
    V[n-1][i] = V[i][i];
    V[i][i] = 1.0;
    double h = d[i+1];
    if (h != 0.0) {
      for (int k = 0; k <= i; k++) {
        d[k] = V[k][i+1] / h;
      }
      for (int j = 0; j <= i; j++) {
        double g = 0.0;
        for (int k = 0; k <= i; k++) {
          g += V[k][i+1] * V[k][j];
        }
        for (int k = 0; k <= i; k++) {
          V[k][j] -= g * d[k];
        }
      }
    }
    for (int k = 0; k <= i; k++) {
      V[k][i+1] = 0.0;
    }
  }
  for (int j = 0; j < n; j++) {
    d[j] = V[n-1][j];
    V[n-1][j] = 0.0;
```

```
  }
  V[n-1][n-1] = 1.0;
  e[0] = 0.0;
}

// Symmetric tridiagonal QL algorithm.

private void tql2 () {

//  This is derived from the Algol procedures tql2, by
//  Bowdler, Martin, Reinsch, and Wilkinson, Handbook for
//  Auto. Comp., Vol.ii-Linear Algebra, and the corresponding
//  Fortran subroutine in EISPACK.

  for (int i = 1; i < n; i++) {
    e[i-1] = e[i];
  }
  e[n-1] = 0.0;

  double f = 0.0;
  double tst1 = 0.0;
  double eps = Math.pow(2.0,-52.0);
  for (int l = 0; l < n; l++) {

    // Find small subdiagonal element

    tst1 = Math.max(tst1,Math.abs(d[l]) + Math.abs(e[l]));
    int m = l;
    while (m < n) {
      if (Math.abs(e[m]) <= eps*tst1) {
        break;
      }
      m++;
    }

    // If m == l, d[l] is an eigenvalue,
    // otherwise, iterate.

    if (m > l) {
      int iter = 0;
      do {
        iter = iter + 1;  // (Could check iteration count here.)

        // Compute implicit shift

        double g = d[l];
        double p = (d[l+1] - g) / (2.0 * e[l]);
        double r = Maths.hypot(p,1.0);
        if (p < 0) {
          r = -r;
        }
        d[l] = e[l] / (p + r);
        d[l+1] = e[l] * (p + r);
        double dl1 = d[l+1];
        double h = g - d[l];
        for (int i = l+2; i < n; i++) {
          d[i] -= h;
        }
        f = f + h;

        // Implicit QL transformation.

        p = d[m];
```

```
            double c = 1.0;
            double c2 = c;
            double c3 = c;
            double el1 = e[l+1];
            double s = 0.0;
            double s2 = 0.0;
            for (int i = m-1; i >= l; i--) {
              c3 = c2;
              c2 = c;
              s2 = s;
              g = c * e[i];
              h = c * p;
              r = Maths.hypot(p,e[i]);
              e[i+1] = s * r;
              s = e[i] / r;
              c = p / r;
              p = c * d[i] - s * g;
              d[i+1] = h + s * (c * g + s * d[i]);

              // Accumulate transformation.

              for (int k = 0; k < n; k++) {
                h = V[k][i+1];
                V[k][i+1] = s * V[k][i] + c * h;
                V[k][i] = c * V[k][i] - s * h;
              }
            }
            p = -s * s2 * c3 * el1 * e[l] / dl1;
            e[l] = s * p;
            d[l] = c * p;

            // Check for convergence.

          } while (Math.abs(e[l]) > eps*tst1);
        }
        d[l] = d[l] + f;
        e[l] = 0.0;
      }

      // Sort eigenvalues and corresponding vectors.

      for (int i = 0; i < n-1; i++) {
        int k = i;
        double p = d[i];
        for (int j = i+1; j < n; j++) {
          if (d[j] < p) {
            k = j;
            p = d[j];
          }
        }
        if (k != i) {
          d[k] = d[i];
          d[i] = p;
          for (int j = 0; j < n; j++) {
            p = V[j][i];
            V[j][i] = V[j][k];
            V[j][k] = p;
          }
        }
      }
    }

    // Nonsymmetric reduction to Hessenberg form.
```

```
private void orthes () {

  //  This is derived from the Algol procedures orthes and ortran,
  //  by Martin and Wilkinson, Handbook for Auto. Comp.,
  //  Vol.ii-Linear Algebra, and the corresponding
  //  Fortran subroutines in EISPACK.

  int low = 0;
  int high = n-1;

  for (int m = low+1; m <= high-1; m++) {

    // Scale column.

    double scale = 0.0;
    for (int i = m; i <= high; i++) {
      scale = scale + Math.abs(H[i][m-1]);
    }
    if (scale != 0.0) {

      // Compute Householder transformation.

      double h = 0.0;
      for (int i = high; i >= m; i--) {
        ort[i] = H[i][m-1]/scale;
        h += ort[i] * ort[i];
      }
      double g = Math.sqrt(h);
      if (ort[m] > 0) {
        g = -g;
      }
      h = h - ort[m] * g;
      ort[m] = ort[m] - g;

      // Apply Householder similarity transformation
      // H = (I-u*u'/h)*H*(I-u*u')/h)

      for (int j = m; j < n; j++) {
        double f = 0.0;
        for (int i = high; i >= m; i--) {
          f += ort[i]*H[i][j];
        }
        f = f/h;
        for (int i = m; i <= high; i++) {
          H[i][j] -= f*ort[i];
        }
      }

      for (int i = 0; i <= high; i++) {
        double f = 0.0;
        for (int j = high; j >= m; j--) {
          f += ort[j]*H[i][j];
        }
        f = f/h;
        for (int j = m; j <= high; j++) {
          H[i][j] -= f*ort[j];
        }
      }
      ort[m] = scale*ort[m];
      H[m][m-1] = scale*g;
    }
  }
```

```
  // Accumulate transformations (Algol's ortran).

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      V[i][j] = (i == j ? 1.0 : 0.0);
    }
  }

  for (int m = high-1; m >= low+1; m--) {
    if (H[m][m-1] != 0.0) {
      for (int i = m+1; i <= high; i++) {
        ort[i] = H[i][m-1];
      }
      for (int j = m; j <= high; j++) {
        double g = 0.0;
        for (int i = m; i <= high; i++) {
          g += ort[i] * V[i][j];
        }
        // Double division avoids possible underflow
        g = (g / ort[m]) / H[m][m-1];
        for (int i = m; i <= high; i++) {
          V[i][j] += g * ort[i];
        }
      }
    }
  }
}


// Complex scalar division.

private transient double cdivr, cdivi;
private void cdiv(double xr, double xi, double yr, double yi) {
  double r,d;
  if (Math.abs(yr) > Math.abs(yi)) {
    r = yi/yr;
    d = yr + r*yi;
    cdivr = (xr + r*xi)/d;
    cdivi = (xi - r*xr)/d;
  } else {
    r = yr/yi;
    d = yi + r*yr;
    cdivr = (r*xr + xi)/d;
    cdivi = (r*xi - xr)/d;
  }
}


// Nonsymmetric reduction from Hessenberg to real Schur form.

private void hqr2 () {

  //  This is derived from the Algol procedure hqr2,
  //  by Martin and Wilkinson, Handbook for Auto. Comp.,
  //  Vol.ii-Linear Algebra, and the corresponding
  //  Fortran subroutine in EISPACK.

  // Initialize

  int nn = this.n;
  int n = nn-1;
  int low = 0;
```

```
int high = nn-1;
double eps = Math.pow(2.0,-52.0);
double exshift = 0.0;
double p=0,q=0,r=0,s=0,z=0,t,w,x,y;

// Store roots isolated by balanc and compute matrix norm

double norm = 0.0;
for (int i = 0; i < nn; i++) {
  if (i < low | i > high) {
    d[i] = H[i][i];
    e[i] = 0.0;
  }
  for (int j = Math.max(i-1,0); j < nn; j++) {
    norm = norm + Math.abs(H[i][j]);
  }
}

// Outer loop over eigenvalue index

int iter = 0;
while (n >= low) {

  // Look for single small sub-diagonal element

  int l = n;
  while (l > low) {
    s = Math.abs(H[l-1][l-1]) + Math.abs(H[l][l]);
    if (s == 0.0) {
      s = norm;
    }
    if (Math.abs(H[l][l-1]) < eps * s) {
      break;
    }
    l--;
  }

  // Check for convergence
  // One root found

  if (l == n) {
    H[n][n] = H[n][n] + exshift;
    d[n] = H[n][n];
    e[n] = 0.0;
    n--;
    iter = 0;

  // Two roots found

  } else if (l == n-1) {
    w = H[n][n-1] * H[n-1][n];
    p = (H[n-1][n-1] - H[n][n]) / 2.0;
    q = p * p + w;
    z = Math.sqrt(Math.abs(q));
    H[n][n] = H[n][n] + exshift;
    H[n-1][n-1] = H[n-1][n-1] + exshift;
    x = H[n][n];

    // Real pair

    if (q >= 0) {
      if (p >= 0) {
        z = p + z;
```

```
      } else {
        z = p - z;
      }
      d[n-1] = x + z;
      d[n] = d[n-1];
      if (z != 0.0) {
        d[n] = x - w / z;
      }
      e[n-1] = 0.0;
      e[n] = 0.0;
      x = H[n][n-1];
      s = Math.abs(x) + Math.abs(z);
      p = x / s;
      q = z / s;
      r = Math.sqrt(p * p+q * q);
      p = p / r;
      q = q / r;

      // Row modification

      for (int j = n-1; j < nn; j++) {
        z = H[n-1][j];
        H[n-1][j] = q * z + p * H[n][j];
        H[n][j] = q * H[n][j] - p * z;
      }

      // Column modification

      for (int i = 0; i <= n; i++) {
        z = H[i][n-1];
        H[i][n-1] = q * z + p * H[i][n];
        H[i][n] = q * H[i][n] - p * z;
      }

      // Accumulate transformations

      for (int i = low; i <= high; i++) {
        z = V[i][n-1];
        V[i][n-1] = q * z + p * V[i][n];
        V[i][n] = q * V[i][n] - p * z;
      }

    // Complex pair

    } else {
      d[n-1] = x + p;
      d[n] = x + p;
      e[n-1] = z;
      e[n] = -z;
    }
    n = n - 2;
    iter = 0;

  // No convergence yet

  } else {

    // Form shift

    x = H[n][n];
    y = 0.0;
    w = 0.0;
    if (l < n) {
```

```
   y = H[n-1][n-1];
   w = H[n][n-1] * H[n-1][n];
}

// Wilkinson's original ad hoc shift

if (iter == 10) {
   exshift += x;
   for (int i = low; i <= n; i++) {
      H[i][i] -= x;
   }
   s = Math.abs(H[n][n-1]) + Math.abs(H[n-1][n-2]);
   x = y = 0.75 * s;
   w = -0.4375 * s * s;
}

// MATLAB's new ad hoc shift

if (iter == 30) {
    s = (y - x) / 2.0;
    s = s * s + w;
    if (s > 0) {
        s = Math.sqrt(s);
        if (y < x) {
           s = -s;
        }
        s = x - w / ((y - x) / 2.0 + s);
        for (int i = low; i <= n; i++) {
           H[i][i] -= s;
        }
        exshift += s;
        x = y = w = 0.964;
    }
}

iter = iter + 1;   // (Could check iteration count here.)

// Look for two consecutive small sub-diagonal elements

int m = n-2;
while (m >= l) {
   z = H[m][m];
   r = x - z;
   s = y - z;
   p = (r * s - w) / H[m+1][m] + H[m][m+1];
   q = H[m+1][m+1] - z - r - s;
   r = H[m+2][m+1];
   s = Math.abs(p) + Math.abs(q) + Math.abs(r);
   p = p / s;
   q = q / s;
   r = r / s;
   if (m == l) {
      break;
   }
   if (Math.abs(H[m][m-1]) * (Math.abs(q) + Math.abs(r)) <
      eps * (Math.abs(p) * (Math.abs(H[m-1][m-1]) + Math.abs(z) +
      Math.abs(H[m+1][m+1])))) {
         break;
   }
   m--;
}

for (int i = m+2; i <= n; i++) {
```

```
      H[i][i-2] = 0.0;
      if (i > m+2) {
         H[i][i-3] = 0.0;
      }
   }

   // Double QR step involving rows l:n and columns m:n

   for (int k = m; k <= n-1; k++) {
      boolean notlast = (k != n-1);
      if (k != m) {
         p = H[k][k-1];
         q = H[k+1][k-1];
         r = (notlast ? H[k+2][k-1] : 0.0);
         x = Math.abs(p) + Math.abs(q) + Math.abs(r);
         if (x != 0.0) {
            p = p / x;
            q = q / x;
            r = r / x;
         }
      }
      if (x == 0.0) {
         break;
      }
      s = Math.sqrt(p * p + q * q + r * r);
      if (p < 0) {
         s = -s;
      }
      if (s != 0) {
         if (k != m) {
            H[k][k-1] = -s * x;
         } else if (l != m) {
            H[k][k-1] = -H[k][k-1];
         }
         p = p + s;
         x = p / s;
         y = q / s;
         z = r / s;
         q = q / p;
         r = r / p;

         // Row modification

         for (int j = k; j < nn; j++) {
            p = H[k][j] + q * H[k+1][j];
            if (notlast) {
               p = p + r * H[k+2][j];
               H[k+2][j] = H[k+2][j] - p * z;
            }
            H[k][j] = H[k][j] - p * x;
            H[k+1][j] = H[k+1][j] - p * y;
         }

         // Column modification

         for (int i = 0; i <= Math.min(n,k+3); i++) {
            p = x * H[i][k] + y * H[i][k+1];
            if (notlast) {
               p = p + z * H[i][k+2];
               H[i][k+2] = H[i][k+2] - p * r;
            }
            H[i][k] = H[i][k] - p;
            H[i][k+1] = H[i][k+1] - p * q;
```

```
          }

          // Accumulate transformations

          for (int i = low; i <= high; i++) {
            p = x * V[i][k] + y * V[i][k+1];
            if (notlast) {
              p = p + z * V[i][k+2];
              V[i][k+2] = V[i][k+2] - p * r;
            }
            V[i][k] = V[i][k] - p;
            V[i][k+1] = V[i][k+1] - p * q;
          }
        }  // (s != 0)
      }  // k loop
    }  // check convergence
  }  // while (n >= low)

  // Backsubstitute to find vectors of upper triangular form

  if (norm == 0.0) {
    return;
  }

  for (n = nn-1; n >= 0; n--) {
    p = d[n];
    q = e[n];

    // Real vector

    if (q == 0) {
      int l = n;
      H[n][n] = 1.0;
      for (int i = n-1; i >= 0; i--) {
        w = H[i][i] - p;
        r = 0.0;
        for (int j = l; j <= n; j++) {
          r = r + H[i][j] * H[j][n];
        }
        if (e[i] < 0.0) {
          z = w;
          s = r;
        } else {
          l = i;
          if (e[i] == 0.0) {
            if (w != 0.0) {
              H[i][n] = -r / w;
            } else {
              H[i][n] = -r / (eps * norm);
            }

          // Solve real equations

          } else {
            x = H[i][i+1];
            y = H[i+1][i];
            q = (d[i] - p) * (d[i] - p) + e[i] * e[i];
            t = (x * s - z * r) / q;
            H[i][n] = t;
            if (Math.abs(x) > Math.abs(z)) {
              H[i+1][n] = (-r - w * t) / x;
            } else {
              H[i+1][n] = (-s - y * t) / z;
```

```
          }
        }

        // Overflow control

        t = Math.abs(H[i][n]);
        if ((eps * t) * t > 1) {
          for (int j = i; j <= n; j++) {
            H[j][n] = H[j][n] / t;
          }
        }
      }
    }

  // Complex vector

  } else if (q < 0) {
    int l = n-1;

    // Last vector component imaginary so matrix is triangular

    if (Math.abs(H[n][n-1]) > Math.abs(H[n-1][n])) {
      H[n-1][n-1] = q / H[n][n-1];
      H[n-1][n] = -(H[n][n] - p) / H[n][n-1];
    } else {
      cdiv(0.0,-H[n-1][n],H[n-1][n-1]-p,q);
      H[n-1][n-1] = cdivr;
      H[n-1][n] = cdivi;
    }
    H[n][n-1] = 0.0;
    H[n][n] = 1.0;
    for (int i = n-2; i >= 0; i--) {
      double ra,sa,vr,vi;
      ra = 0.0;
      sa = 0.0;
      for (int j = l; j <= n; j++) {
        ra = ra + H[i][j] * H[j][n-1];
        sa = sa + H[i][j] * H[j][n];
      }
      w = H[i][i] - p;

      if (e[i] < 0.0) {
        z = w;
        r = ra;
        s = sa;
      } else {
        l = i;
        if (e[i] == 0) {
          cdiv(-ra,-sa,w,q);
          H[i][n-1] = cdivr;
          H[i][n] = cdivi;
        } else {

          // Solve complex equations

          x = H[i][i+1];
          y = H[i+1][i];
          vr = (d[i] - p) * (d[i] - p) + e[i] * e[i] - q * q;
          vi = (d[i] - p) * 2.0 * q;
          if (vr == 0.0 & vi == 0.0) {
            vr = eps * norm * (Math.abs(w) + Math.abs(q) +
            Math.abs(x) + Math.abs(y) + Math.abs(z));
          }
```

```
                cdiv(x*r-z*ra+q*sa,x*s-z*sa-q*ra,vr,vi);
                H[i][n-1] = cdivr;
                H[i][n] = cdivi;
                if (Math.abs(x) > (Math.abs(z) + Math.abs(q))) {
                  H[i+1][n-1] = (-ra - w * H[i][n-1] + q * H[i][n]) / x;
                  H[i+1][n] = (-sa - w * H[i][n] - q * H[i][n-1]) / x;
                } else {
                  cdiv(-r-y*H[i][n-1],-s-y*H[i][n],z,q);
                  H[i+1][n-1] = cdivr;
                  H[i+1][n] = cdivi;
                }
              }

              // Overflow control

              t = Math.max(Math.abs(H[i][n-1]),Math.abs(H[i][n]));
              if ((eps * t) * t > 1) {
                for (int j = i; j <= n; j++) {
                  H[j][n-1] = H[j][n-1] / t;
                  H[j][n] = H[j][n] / t;
                }
              }
            }
          }
        }
      }

      // Vectors of isolated roots

      for (int i = 0; i < nn; i++) {
        if (i < low | i > high) {
          for (int j = i; j < nn; j++) {
            V[i][j] = H[i][j];
          }
        }
      }

      // Back transformation to get eigenvectors of original matrix

      for (int j = nn-1; j >= low; j--) {
        for (int i = low; i <= high; i++) {
          z = 0.0;
          for (int k = low; k <= Math.min(j,high); k++) {
            z = z + V[i][k] * H[k][j];
          }
          V[i][j] = z;
        }
      }
    }


/* ------------------------
   Constructor
 * ------------------------ */

   /** Check for symmetry, then construct the eigenvalue decomposition
   @param A    Square matrix
   @return     Structure to access D and V.
   */

   public EigenvalueDecomposition (Matrix Arg) {
     double[][] A = Arg.getArray();
     n = Arg.getColumnDimension();
```

```
      V = new double[n][n];
      d = new double[n];
      e = new double[n];

      issymmetric = true;
      for (int j = 0; (j < n) & issymmetric; j++) {
        for (int i = 0; (i < n) & issymmetric; i++) {
          issymmetric = (A[i][j] == A[j][i]);
        }
      }

      if (issymmetric) {
        for (int i = 0; i < n; i++) {
          for (int j = 0; j < n; j++) {
            V[i][j] = A[i][j];
          }
        }

        // Tridiagonalize.
        tred2();

        // Diagonalize.
        tql2();

      } else {
        H = new double[n][n];
        ort = new double[n];

        for (int j = 0; j < n; j++) {
          for (int i = 0; i < n; i++) {
            H[i][j] = A[i][j];
          }
        }

        // Reduce to Hessenberg form.
        orthes();

        // Reduce Hessenberg to real Schur form.
        hqr2();
      }
    }

  /* ------------------------
     Public Methods
   * ------------------------ */

    /** Return the eigenvector matrix
    @return     V
    */

    public Matrix getV () {
      return new Matrix(V,n,n);
    }

    /** Return the real parts of the eigenvalues
    @return     real(diag(D))
    */

    public double[] getRealEigenvalues () {
      return d;
    }

    /** Return the imaginary parts of the eigenvalues
```

```
   @return     imag(diag(D))
   */

   public double[] getImagEigenvalues () {
      return e;
   }

   /** Return the block diagonal eigenvalue matrix
   @return     D
   */

   public Matrix getD () {
      Matrix X = new Matrix(n,n);
      double[][] D = X.getArray();
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            D[i][j] = 0.0;
         }
         D[i][i] = d[i];
         if (e[i] > 0) {
            D[i][i+1] = e[i];
         } else if (e[i] < 0) {
            D[i][i-1] = e[i];
         }
      }
      return X;
   }
}
```

APPENDIX C

JAVADOC

Javadoc help files are generated automatically if appropriate convention is used while developing the code. These documents help the author and other developers to quickly understand the code and trace it easily. It is a good way to understand the code for adding new modules and functions to the project. The JavaDoc files are given below.

Class ControlSystemApplet

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Panel
              └ java.applet.Applet
                  └ javax.swing.JApplet
                      └ controlapplet.ControlSystemApplet
```

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, javax.swing.RootPaneContainer, java.io.Serializable

---

public class **ControlSystemApplet**
extends javax.swing.JApplet

Implements the applet.

Version:

1.0

Author:

Hüseyin CİĞEROĞLU

See Also:

Serialized Form

Nested Class Summary

Nested classes inherited from class javax.swing.JApplet
javax.swing.JApplet.AccessibleJApplet

Nested classes inherited from class java.applet.Applet
java.applet.Applet.AccessibleApplet

Nested classes inherited from class java.awt.Panel
java.awt.Panel.AccessibleAWTPanel

Nested classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

(package private) javax.swing.JButton	**controlButton** Button to hide or show the application.
(package private) int	**count** Variable to hold the hide or show state.
double[]	**denCS**
(package private) javax.swing.JFrame	**frame** Main container frame

(package private) boolean	**isStandalone**
double[]	**numCS**

Fields inherited from class javax.swing.JApplet
accessibleContext, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Container

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

ControlSystemApplet()

Method Summary

(package private) void	**controlButton_actionPerformed**(java.awt.event.ActionEvent e) Displays or hides the main panel when clicked
void	**destroy**()

java.lang.String	**getAppletInfo**()
java.lang.String	**getParameter**(java.lang.String key, java.lang.String def)
java.lang.String[][]	**getParameterInfo**()
void	**init**() Calls init function of the applet.
static void	**main**(java.lang.String[] args) Creates applet(type CAApplet) and frame(type JFrame) (frame is the container for geometry).
void	**start**()
void	**stop**()

Methods inherited from class javax.swing.JApplet
addImpl, createRootPane, getAccessibleContext, getContentPane, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isRootPaneCheckingEnabled, paramString, remove, setContentPane, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.applet.Applet
getAppletContext, getAudioClip, getAudioClip, getCodeBase, getDocumentBase, getImage, getImage, getLocale, getParameter, isActive, newAudioClip, play, play, resize, resize, setStub, showStatus

Methods inherited from class java.awt.Panel
addNotify

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

isStandalone
`boolean` **`isStandalone`**

---

count
`int` **`count`**

Variable to hold the hide or show state.

---

controlButton
`javax.swing.JButton` **`controlButton`**

Button to hide or show the application.

---

frame
`javax.swing.JFrame` **`frame`**

Main container frame

---

numCS
`public double[]` **`numCS`**

---

denCS
`public double[]` **`denCS`**

Constructor Detail

ControlSystemApplet
`public` **`ControlSystemApplet`**`()`

Method Detail

getParameter

```
public java.lang.String getParameter(java.lang.String key,
                                      java.lang.String def)
```

---

init
`public void` **`init`**`()`

Calls init function of the applet.

Returns:
void

---

start
`public void` **`start`**`()`

---

stop
`public void` **`stop`**`()`

---

destroy
`public void` **`destroy`**`()`

---

getAppletInfo
`public java.lang.String` **`getAppletInfo`**`()`

---

getParameterInfo
`public java.lang.String[][]` **`getParameterInfo`**`()`

---

main
`public static void` **`main`**`(java.lang.String[] args)`

Creates applet(type CAApplet) and frame(type JFrame) (frame is the container for geometry). Also creates controlButton to make the frame visible.

---

controlButton_actionPerformed
`void` **`controlButton_actionPerformed`**`(java.awt.event.ActionEvent e)`

Displays or hides the main panel when clicked

Class newRK4

`java.lang.Object`
  `└` **`controlapplet.newRK4`**

---

class **newRK4**
extends java.lang.Object

Field Summary

(package private) float	**ctime**
(package private) double[]	**den** Denominator of the transfer function
(package private) float	**hstep**
(package private) double[][]	**k**
(package private) static int	**mxplot**
(package private) static int	**nd**
(package private) int	**nden**
(package	**nstep**

private) static int	
(package private) double[]	**num** Numerator of the transfer function
(package private) static int	**ny**
(package private) int	**nyy**
(package private) float	**Pymax**
(package private) float	**Pymin**
(package private) double[]	**u**
(package private) static float	**xmax**
(package private) static float	**xmin**
(package private) static float[]	**xst**
(package private) static float	**ymax**
(package private) static float	**ymin**
(package private) static float[][]	**yst**
(package private) float[]	**ytemp**
(package private) double[]	**yval**
(package private) double[]	**yvalp**

Constructor Summary

(package private)	**newRK4**(float startT, float endT) Constructor of the RK4 function

Method Summary

(package private) void	**calculateRK4**(float f, float f1, int denOrder, double[] numerator, double[] denominator) Calculates the RK4 function
(package private) void	**evalf**(float f, double[] ad, float[] af) Evaluates the function.
(package private) float[]	**func**(float f) Calculates the steps.
(package private) void	**sumk**(int i, float f, double d, double d1, double d2, double d3, double d4) Adds the steps.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

nstep
static int **nstep**

---

mxplot
static int **mxplot**

---

ny
static int **ny**

---

nd
static int **nd**

---

yst
static float[][] **yst**

---

xst
`static float[] xst`

---

ymin
`static float ymin`

---

ymax
`static float ymax`

---

xmin
`static float xmin`

---

xmax
`static float xmax`

---

nyy
`int nyy`

---

ctime
`float ctime`

---

hstep
`float hstep`

---

k
`double[][] k`

---

yval
`double[] yval`

---

yvalp
`double[] yvalp`

---

ytemp
`float[] ytemp`

---

u
`double[] u`

---

num
`double[] num`

Numerator of the transfer function

---

den
`double[] den`

Denominator of the transfer function

---

nden
`int nden`

Pymax
`float` **`Pymax`**

Pymin
`float` **`Pymin`**

Constructor Detail

newRK4
```
newRK4(float startT,
       float endT)
```
Constructor of the RK4 function

Parameters:

`startT` - start time

`endT` - end time

Method Detail

calculateRK4
```
void calculateRK4(float f,
                  float f1,
                  int denOrder,
                  double[] numerator,
                  double[] denominator)
```
Calculates the RK4 function

Parameters:

`f` - start time

`f1` - end time

`denOrder` - order of the denominator

`numerator` - numerator of the transfer funtion

Returns:

void

---

func
```
float[] func(float f)
```
Calculates the steps.

Parameters:

`f` -

Returns:

floating number

---

sumk
```
void sumk(int i,
          float f,
          double d,
          double d1,
          double d2,
          double d3,
          double d4)
```
Adds the steps.

Parameters:

`i` -

f -
d -
d1 -
d2 -
d3 -
d4 -

Returns:
void

---

evalf

```
void evalf(float f,
           double[] ad,
           float[] af)
```

Evaluates the function.

Parameters:
ad -
af -

Returns:
void

Class Rlocus

```
java.lang.Object
  └ controlapplet.Rlocus
```

---

public class **Rlocus**
extends java.lang.Object

Implements the root locus finding algorithm.

Version:
1.0

Author:
Hüseyin Ciğeroğlu

Field Summary	
Matrix	**rImag** The imaginary part of the root locus.
Matrix	**rReal** The real part of the root locus.

Constructor Summary

Rlocus()	Constructor of the Rlocus class.

Method Summary

boolean	**findRlocus**(Matrix a, Matrix b, Matrix c, double d, double[] k) Calculates the root locus of the given state space system.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

rImag

public Matrix **rImag**

The imaginary part of the root locus.

---

rReal

public Matrix **rReal**

The real part of the root locus.

Constructor Detail

Rlocus

public **Rlocus**()

Constructor of the Rlocus class.

Method Detail

findRlocus

```
public boolean findRlocus(Matrix a,
                          Matrix b,
                          Matrix c,
                          double d,
                          double[] k)
```

Calculates the root locus of the given state space system.

Parameters:

a - the A matrix

b - the B matrix

c - the C matrix
d - the D matrix
k - the gain array to calculate
Returns:
true : if success

Class BodeFrame

java.lang.Object
  └ **controlapplet.BodeFrame**

---

public class **BodeFrame**
extends java.lang.Object

Implements the bode graph algorithm given the transfer function.

Version:
1.0
Author:
Hüseyin CİĞEROĞLU

Field Summary

(package private) double	**maxDB** The maximum magnitude value.
protected JPlot2D	**plotMag** The Bode magnitude graph.
protected JPlot2D	**plotPhase** The Bode phase graph.
(package private) double[]	**theFreqs** The frequency array.
(package private) ComplexNumber[]	**theResult** The complex transfer function.

Constructor Summary

BodeFrame(double[] n, double[] d) Constructor of the BodeFrame class.

Method Summary

ComplexNumber[]	**calculateValues**(double[] nums, double[] dens) Calculates the transfer function in complex values given the frequency array.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

plotMag

protected JPlot2D **plotMag**

The Bode magnitude graph.

---

plotPhase

protected JPlot2D **plotPhase**

The Bode phase graph.

---

theResult

ComplexNumber[] **theResult**

The complex transfer function.

---

theFreqs

double[] **theFreqs**

The frequency array.

---

maxDB

double **maxDB**

The maximum magnitude value.

Constructor Detail

BodeFrame

public **BodeFrame**(double[] n,
double[] d)

Constructor of the BodeFrame class.

Parameters:

n - the numerator of the tansfer function

d - the denominator of the transfer function

Method Detail

calculateValues

```
public ComplexNumber[] calculateValues(double[] nums,
                                       double[] dens)
```

Calculates the transfer function in complex values given the frequency array.

Parameters:

`nums` - the numerator of the tansfer function

`dens` - the denominator of the transfer function

Returns:

Complex array : the resultant Complex array of the transfer function.

Class NyquistFrame

```
java.lang.Object
  └ controlapplet.NyquistFrame
```

---

public class **NyquistFrame**
extends java.lang.Object

Implements the bode graph algorithm given the transfer function.

Version:

1.0

Author:

Hüseyin CİĞEROĞLU

Field Summary

protected JPlot2D	**plotNyquist** The Nyquist graph.
(package private) double[]	**theFreqs** The frequency array.
(package private) ComplexNumber[]	**theResult** The complex transfer function.

Constructor Summary

NyquistFrame(double[] n, double[] d) Constructor of the NyquistFrame class.

Method Summary

ComplexNumber[]	**plotAll**(double[] nums, double[] dens) Calculates the transfer function in complex values given the frequency array.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

plotNyquist

protected JPlot2D **plotNyquist**

The Nyquist graph.

---

theResult

ComplexNumber[] **theResult**

The complex transfer function.

---

theFreqs

double[] **theFreqs**

The frequency array.

Constructor Detail

NyquistFrame

public **NyquistFrame**(double[] n,
double[] d)

Constructor of the NyquistFrame class.

Parameters:

n - the numerator of the tansfer function

d - the denominator of the transfer function

Method Detail

plotAll

public ComplexNumber[] **plotAll**(double[] nums,
double[] dens)

Calculates the transfer function in complex values given the frequency array.

Parameters:

nums - the numerator of the tansfer function

dens - the denominator of the transfer function

Returns:

Complex array : the resultant Complex array of the transfer function.

Class Tf2ss

```
java.lang.Object
  └ controlapplet.Tf2ss
```

---

public class **Tf2ss**
extends java.lang.Object

Implements the transfer function to state space conversion.

Version:
1.0
Author:
Hüseyin CİĞEROĞLU

Field Summary

Matrix	**A** Resulting A matrix.
Matrix	**B** Resulting B matrix.
Matrix	**C** Resulting C matrix.
Matrix	**D** Resulting D matrix.

Constructor Summary

Tf2ss() Constructor of the Zp2tf class.

Method Summary

boolean	**calculateTF2SS**(double[] num, double[] den) Converts transfer function systems to state space systems.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll,

toString, wait, wait, wait

Field Detail

A

public Matrix **A**

Resulting A matrix.

B

public Matrix **B**

Resulting B matrix.

C

public Matrix **C**

Resulting C matrix.

D

public Matrix **D**

Resulting D matrix.

Constructor Detail

Tf2ss

public **Tf2ss**()

Constructor of the Zp2tf class.

Method Detail

calculateTF2SS

public boolean **calculateTF2SS**(double[] num,
double[] den)

Converts transfer function systems to state space systems.

Parameters:

num - numerator of the transfer function

den - denominator of the transfer function

Returns:

true : if success

Class Zp2tf

java.lang.Object
  └ **controlapplet.Zp2tf**

---

public class **Zp2tf**
extends java.lang.Object

Implements the zero-pole to transfer function conversion.

Version:
1.0
Author:
Hüseyin CİĞEROĞLU

Field Summary

double[]	**den** Resulting denominator.
double[]	**num** Resulting numerator.

Constructor Summary

Zp2tf() Constructor of the Zp2tf class.

Method Summary

boolean	**calculateZP2TF**(java.util.Vector zpPoints, int k) Converts zero-pole systems to transfer function systems.
double[]	**constructPoly**(java.util.Vector a) Constructs polynomials from given roots.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

den
public double[] **den**
Resulting denominator.

---

num
public double[] **num**
Resulting numerator.

Constructor Detail

Zp2tf

```
public Zp2tf()
```

Constructor of the Zp2tf class.

Method Detail

calculateZP2TF

```
public boolean calculateZP2TF(java.util.Vector zpPoints,
                              int k)
```

Converts zero-pole systems to transfer function systems.

Parameters:

`zpPoints` - the zeros and poles of the system

`k` - the gain

Returns:

true : if success

---

constructPoly

```
public double[] constructPoly(java.util.Vector a)
```

Constructs polynomials from given roots.

Parameters:

`a` - : A vector consisting of the roots

Class EigenvalueDecomposition

```
java.lang.Object
  └ Jama.EigenvalueDecomposition
```

All Implemented Interfaces:

java.io.Serializable

---

public class **EigenvalueDecomposition**
extends java.lang.Object
implements java.io.Serializable

This class was taken from the NIST JAMA package.

Eigenvalues and eigenvectors of a real matrix.

If A is symmetric, then A = V*D*V' where the eigenvalue matrix D is diagonal and the eigenvector matrix V is orthogonal. I.e. A = V.times(D.times(V.transpose())) and V.times(V.transpose()) equals the identity matrix.

If A is not symmetric, then the eigenvalue matrix D is block diagonal with the real eigenvalues in 1-by-1 blocks and any complex eigenvalues, lambda + i*mu, in 2-by-2 blocks, [lambda, mu; -mu, lambda]. The columns of V represent the eigenvectors in the sense that A*V = V*D, i.e. A.times(V) equals V.times(D). The matrix V may

be badly conditioned, or even singular, so the validity of the equation A = V*D*inverse(V) depends upon V.cond().

See Also:

Serialized Form

Constructor Summary

EigenvalueDecomposition(Matrix Arg)	Check for symmetry, then construct the eigenvalue decomposition

Method Summary

Matrix	**getD**() Return the block diagonal eigenvalue matrix
double[]	**getImagEigenvalues**() Return the imaginary parts of the eigenvalues
double[]	**getRealEigenvalues**() Return the real parts of the eigenvalues
Matrix	**getV**() Return the eigenvector matrix

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EigenvalueDecomposition

public **EigenvalueDecomposition**(Matrix Arg)

Check for symmetry, then construct the eigenvalue decomposition

Method Detail

getV

public Matrix **getV**()

Return the eigenvector matrix

Returns:

V

---

getRealEigenvalues

```
public double[] getRealEigenvalues()
```

Return the real parts of the eigenvalues

Returns:

real(diag(D))

---

getImagEigenvalues

```
public double[] getImagEigenvalues()
```

Return the imaginary parts of the eigenvalues

Returns:

imag(diag(D))

---

getD

```
public Matrix getD()
```

Return the block diagonal eigenvalue matrix

Returns:

D

APPENDIX D

PARAMETER FILE

This code is the parameter file which is the place where the instructor can define a new system. The variable names are self-explanatory.

```
package parameter;

public class params {

  //INITIAL POLES AND ZEROS
  //sample for p or z : "2,-5 3,0" = 2-5j , 2+5j , 3
  //write only one of the complex conjugates
  public final static String POLES = "5";
  public final static String ZEROS = "-5";

  //TIME SCALE
  public static boolean SHOW_TIME_SCALE = true;
  public static int TIME_AXIS_CURRENT_VALUE = 20;
  public final static int TIME_AXIS_MAX_VALUE = 30;

  //POLE AND ZERO OPERATIONS
  //true means these operations can be done by the student
  public final static boolean ADD_POLE = true;
  public final static boolean ADD_ZERO = true;
  public final static boolean PZ_MOVE = true;
  public final static boolean PZ_DELETE = true;

  //PZ real or imaginary, axis fixed or not?
  public static boolean IMAG_FIXED = false;
  public static boolean REAL_FIXED = false;
  public static boolean IMAGREAL_FREE = true;

  //SHOW TABS
  public static boolean SHOW_ZP_TAB = true;
  public static boolean SHOW_TF_TAB = true;
  public static boolean SHOW_SS_TAB = true;
  public static boolean SHOW_FUNC_TAB = true;

  //MAIN FRAME
  public static int FRAME_X = 750;
  public static int FRAME_Y = 625;
```

```
  //TF SLIDERS
  public static boolean SHOW_NUM_SLIDER1 = true;
  public static boolean SHOW_NUM_SLIDER2 = true;
  public static boolean SHOW_NUM_SLIDER3 = true;
  public static boolean SHOW_NUM_SLIDER4 = true;

  public static boolean SHOW_DEN_SLIDER1 = true;
  public static boolean SHOW_DEN_SLIDER2 = true;
  public static boolean SHOW_DEN_SLIDER3 = true;
  public static boolean SHOW_DEN_SLIDER4 = true;

  public static int NUM_SLIDER1_MIN=-30;
  public static int NUM_SLIDER1_MAX=30;
  public static int NUM_SLIDER2_MIN=-30;
  public static int NUM_SLIDER2_MAX=30;
  public static int NUM_SLIDER3_MIN=-30;
  public static int NUM_SLIDER3_MAX=30;
  public static int NUM_SLIDER4_MIN=-30;
  public static int NUM_SLIDER4_MAX=30;

  public static int DEN_SLIDER1_MIN=-30;
  public static int DEN_SLIDER1_MAX=30;
  public static int DEN_SLIDER2_MIN=-30;
  public static int DEN_SLIDER2_MAX=30;
  public static int DEN_SLIDER3_MIN=-30;
  public static int DEN_SLIDER3_MAX=30;
  public static int DEN_SLIDER4_MIN=-30;
  public static int DEN_SLIDER4_MAX=30;

  //FEEDBACK
  public static boolean feedback = false;
  public static int feedbackValue = 0;
  public static int FEEDBACK_MIN=-30;
  public static int FEEDBACK_MAX=30;
  public static double[] feed_num;
  public static double[] feed_den;

  public static double[] num;
  public static double[] den;
}
```

APPENDIX E

JPLOT2D LIBRARY

This is the explanation of the package, taken from its class help file:

Class JPlot2D contains methods to plot (x, y) data values. The plots can be linear-linear, log-linear, linear-log, log-log, polar, or bar. This class implements the Printable interface, and supports a pop-up menu to enable the plot to be sent to a printer.

This is a lightweight component suitable for use with Swing GUI components. The corresponding heavyweight component suitable for use with AWT GUI components is chapman.graphics.Plot2D.

This class supports the input of a single double array, two double arrays, or single Complex array. If a single double array is supplied, the elements of that array will be plotted as a function of their array indices. If two single double arrays are supplied, the first array will provide the x coordinate and the second array will provide the y coordinate. If a single Complex array is supplied, the real part of the array will be plotted on the x axis and the imaginary part of the array will be plotted on the y axis.

Each JPlot2D object can display up to 16 separate curves. The line and marker styles and colors associated with each curve can be controlled separately, with the methods setLineState and setMarkerState controlling whether or not lines and markers are drawn for each curve. By default, lines are drawn between points on the curve, and markers are not drawn at data points, but this behavior can be changed with the above-mentioned methods. A user may specify one of four possible line styles: LINESTYLE_SOLID, LINESTYLE_DOT, LINESTYLE_LONGDASH, and LINESTYLE_SHORTDASH. These styles rotate by default on successive curves within a single plot. A user may specify one of five possible marker styles:

MARKER_SQUARE, MARKER_CIRCLE, MARKER_TRIANGLE, MARKER_DIAMOND, and MARKER_DOWNSQUARE. These styles rotate by default on successive curves within a single plot.

Features of this class include

1. Up to 16 curves per plot.
2. User-settable line colors.
3. User-settable line widths.
4. User-settable line styles.
5. User-settable marker styles.
6. Optional grid lines.
7. User-settable tic mark locations.
8. User-settable title and axis labels.
9. The ability to add additional annotations (except for polar plots).
10. The ability to print the graph using a pop-up menu.

Since this class extends JComponent, it can be used with either applications or applets.

Note 1: If the plot type is polar, then the (x, y) values are interpreted as (r, theta), where r is the radial distance from the origin of the plot, and theta is the angle in radians, counterclockwise from the positive x-axis. Also, in the case of polar plots, the input angles must be specified in radians, but the angle labels on the plot are in degrees.

Note 2: If an x or y axis is logarithmic, then zero or negative values on that axis will produce an InvalidPlotValueException.

Note 3: If the plot is a bar plot, then the y values must all be non-negative. A negative value will produce an InvalidPlotValueException. [6]