

Transfer Function Generator

Christos Bohoris

BEng. Electrical and Electronic Engineering

Table of Contents

| | |
|---|----------|
| Table of Contents..... | 2 |
| PROJECT SPECIFICATION..... | 4 |
| Summary..... | 5 |
| Chapter 1 - Control Project..... | 7 |
| 1.1 Introduction..... | 7 |
| 1.2 System Requirements | 7 |
| 1.3 Code Implementation..... | 7 |
| Chapter 2 - Basic Graph Design | 11 |
| 2.1 General..... | 11 |
| 2.2 Theory - Calculating Graph data | 11 |
| 2.3 Relationships of Class CGraph..... | 13 |
| 2.4 Declaration of Class CGraph..... | 14 |
| 2.5 Implementation of Class CGraph | 15 |
| Chapter 3 - Bode Plots..... | 23 |
| 3.1 General..... | 23 |
| 3.2 Theory - Bode Plots | 23 |
| 3.3 Relationships of Class CBode | 24 |
| 3.4 Declaration of Class CBode..... | 25 |
| 3.5 Implementation of Class CBode..... | 26 |
| 3.6 The Interface..... | 27 |
| Chapter 4 - Nyquist Diagrams | 32 |
| 4.1 General..... | 32 |
| 4.2 Theory - Nyquist Diagram..... | 32 |
| 4.3 Relationships of Class CNyquist | 32 |
| 4.4 Declaration of Class CNyquist | 33 |
| 4.5 Implementation of Class CNyquist..... | 34 |
| 4.6 The Interface..... | 35 |
| Chapter 5 - Nichols Charts | 38 |
| 5.1 General..... | 38 |
| 5.2 Theory - Nichols Charts..... | 38 |
| 5.3 Relationships of Class CNichols | 38 |
| 5.4 Declaration of Class CNichols..... | 39 |
| 5.5 Implementation of Class CNichols..... | 40 |
| 5.6 The Interface..... | 42 |
| Chapter 6 - Inverse Nyquist Diagrams | 45 |
| 6.1 General..... | 45 |
| 6.2 Theory - Inverse Nyquist Diagrams | 45 |
| 6.3 Code Implementation..... | 45 |
| 6.4 The Interface..... | 46 |
| Chapter 7 - Routh-Hurwitz Criteria..... | 49 |
| 7.1 General..... | 49 |
| 7.2 Theory - Routh-Hurwitz Criteria..... | 49 |
| 7.3 Routh-Hurwitz Criteria in Control Project | 51 |
| 7.4 The Interface..... | 52 |

| | |
|---|-----|
| Chapter 8 - Horner's Method..... | 54 |
| 8.1 General..... | 54 |
| 8.2 Theory - Horner's Method..... | 54 |
| 8.3 Horner's Method in Control Project..... | 56 |
| 8.4 The Interface..... | 57 |
| Chapter 9 - The User Interface | 59 |
| 9.1 General..... | 59 |
| 9.2 Declaration of Class TCProjApp..... | 59 |
| 9.3 Implementation of Class TCProjApp..... | 60 |
| Discussion - Conclusions..... | 68 |
| References..... | 71 |
| Appendix..... | 73 |
| A.1 File: CP.CPP | 73 |
| A.2 File: CAPP.CPP | 100 |
| A.3 File: CAPP.H | 118 |
| A.4 File: CPH.TXT..... | 121 |

CARDIFF SCHOOL OF ENGINEERING

PROJECT SPECIFICATION

Agreed Title: Transfer Function Generator

Aims and Objectives: The aim of this project is to design and develop techniques to obtain a transfer function of a system (first or second order only) from a block diagram and then analyze the transfer function and plot the response using C programming language.

Supervisor
Dr. T. Meydan

2nd Examiner
Dr. J. Liu

Student
C. Bohoris

Date: 30-10-1996

Report Submission Date: 23-4-1997

Oral Presentation Date: 6-5-1997

Summary

Control Project v1.0 is a powerful program that can be used to analyze a control system. Using Control Project the user can easily obtain the Bode plot, the Nyquist diagram, the Nichols chart or the inverse Nyquist diagram of a system by just entering its transfer function. Control Project also allows you to analyze a control system by using the Routh-Hurwitz Criteria or Horner's method. The program was written in C++ using object-oriented techniques. It has a user friendly interface, with full keyboard and mouse support which along with an efficient help system provides an ideal tool for any user.

Chapter 1 - Control Project

1.1 Introduction

The initial aim of Control project was the design of a system that would analyze a transfer function and then it would draw its Bode plot. Version 1.00 of Control Project not only achieved this goal but also includes many other features that promote it to a powerful Control tool. With Control Project anyone can easily draw a Bode plot, a Nyquist diagram, a Nichols chart or an inverse Nyquist diagram of a control system. All graphs can easily be printed by just pressing the keyboard's 'Print Screen' button. The user can change the default settings of all these graphs with just a simple mouse click. The program also allows you to test the system with two powerful control methods, the Routh-Hurwitz criteria and the Horner's method. Control project has a simple and user friendly desktop environment written in C++ using the Turbo Vision library classes. The environment has full keyboard and mouse support at any level. A specially made help file provides structured help on really anything on the desktop from menu items to dialog boxes and the Control Project desktop itself. Every good program has to have a future. In that respect the first lines of code of Control Project version 2.0 for 32-bit Windows have already been written while the complete program will be ready by September 1997.

1.2 System Requirements

Control Project version 1.0 can run in an Intel 8086 although an 80386 or higher processor is recommended. Other requirements include Microsoft's MS-DOS operating system or compatible (Digital Research DR-DOS, IBM's PC-DOS etc.), 2MB of RAM, a VGA compatible display card and a PS/2 compatible mouse (optional).

1.3 Code Implementation

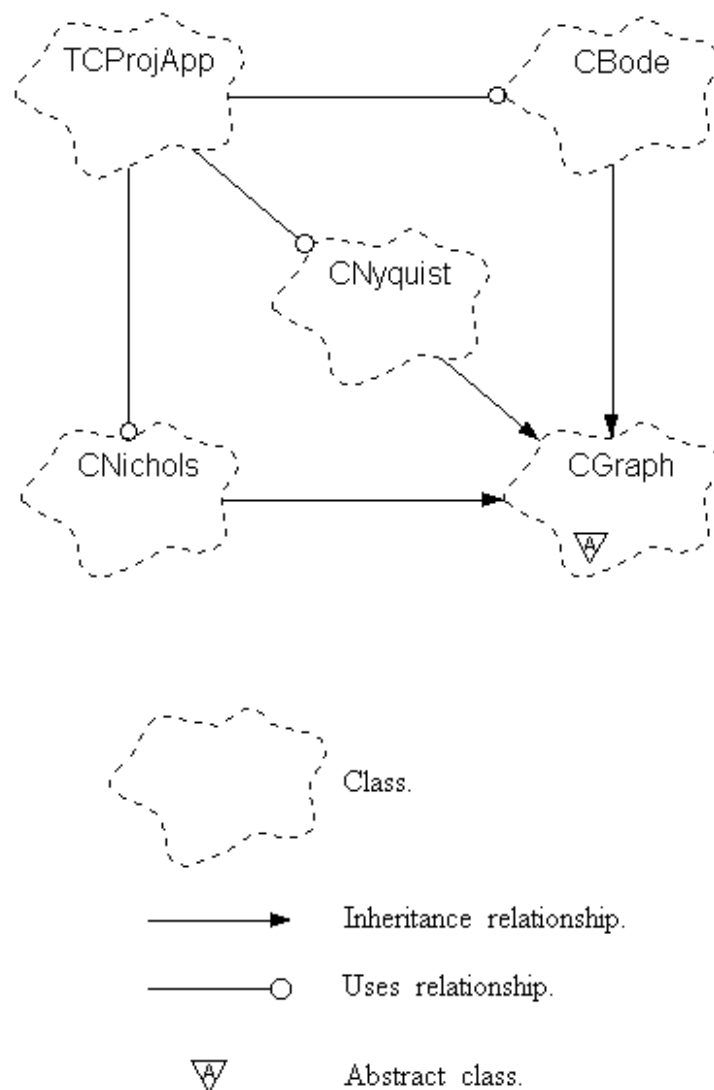
Control Project was written in C++ using the Borland C++ v3.0 compiler. The structure of the code consists of the following classes,

- CGraph
- CBode
- CNyquist
- CNichols

- TProjApp

In the first stage of the development the relationships of these classes were decided and analyzed using Booch's object-oriented techniques. Various class diagrams were also sketched with the aid of an object oriented software engineering program (Rational Rose v3.02 for Windows). The following class diagram shows the basic relationships between the classes of the program.

Figure 1.1 - Basic class relationships and list of symbols used.



As we can see the CGraph class is an abstract class which means that it has no instances and it only exists for inheritance purposes. Class CGraph contains all these functions that perform calculations or set the basic graphics operations and are shared by the other classes of the program. Classes CBode, CNyquist and CNichols inherit the characteristics of CGraph and are responsible for obtaining Bode plots, Nyquist diagrams (inverse or normal) and Nichols charts respectively. Class TCProjApp uses the above three classes (Figure 1.1) and also provides a link between them and the environment of Control Project. This class also contains the implementation of the inverse Nyquist diagrams (using class CNyquist), the Routh-Hurwitz method and Horner's method as well as the functions used for the design of the interface.

Chapter 2 - Basic Graph Design

2.1 General

This chapter is dedicated to the analysis of the CGraph class. The CGraph class is an abstract class used as a base class for other classes to inherit. It shares all the common characteristics, attributes and behaviors that all three graph classes, CBode for Bode plots, CNyquist for Nyquist diagrams, and CNichols for Nichols charts can share.

2.2 Theory - Calculating Graph data

Let the open-loop transfer function represented by the rational polynomial,

$$G(s) = \frac{N(s)}{D(s)} \quad (\text{Equation 2.1})$$

where $N(s)$ and $D(s)$ are polynomials of the form,

$$D(s) = a_0 s^n + a_1 s^{n-1} + \dots + a_{n-1} s + a_n \quad (\text{Equation 2.2})$$

$$N(s) = b_0 s^m + b_1 s^{m-1} + \dots + b_{m-1} s + b_m \quad (\text{Equation 2.3})$$

For a given frequency ω , the corresponding complex open-loop data is obtained by setting $s = j\omega$,

$$G(j\omega) = \frac{N(j\omega)}{D(j\omega)} \quad (\text{Equation 2.4})$$

where,

$$D(j\omega) = a_0 (j\omega)^n + a_1 (j\omega)^{n-1} + \dots + a_{n-1} (j\omega) + a_n \quad (\text{Equation 2.5})$$

$$N(j\omega) = b_0 (j\omega)^m + b_1 (j\omega)^{m-1} + \dots + b_{m-1} (j\omega) + b_m \quad (\text{Equation 2.6})$$

This may be separated into real and imaginary parts,

$$\text{Re}(D(j\omega)) = a_n - \omega^2 a_{n-2} + \omega^4 a_{n-4} - \omega^6 a_{n-6} + \dots \quad (\text{Equation 2.7})$$

$$\text{Im}(D(j\omega)) = a_{n-1} - \omega^3 a_{n-3} + \omega^5 a_{n-5} - \omega^7 a_{n-7} + \dots \quad (\text{Equation 2.8})$$

$$\operatorname{Re}(N(j\omega)) = a_m - \omega^2 a_{m-2} + \omega^4 a_{m-4} - \omega^6 a_{m-6} + \dots \quad (\text{Equation 2.9})$$

$$\operatorname{Im}(N(j\omega)) = a_{m-1} - \omega^3 a_{m-3} + \omega^5 a_{m-5} - \omega^7 a_{m-7} + \dots \quad (\text{Equation 2.10})$$

To find the open-loop real and imaginary coordinates, the normal rules of complex algebra are applied to give,

$$G(j\omega) = \frac{\operatorname{Re}(N(j\omega)) + j \operatorname{Im}(N(j\omega))}{\operatorname{Re}(D(j\omega)) + j \operatorname{Im}(D(j\omega))} \quad (\text{Equation 2.11})$$

From which,

$$\operatorname{Re}(G(j\omega)) = \frac{\operatorname{Re}(N(j\omega)) \operatorname{Re}(D(j\omega)) + \operatorname{Im}(N(j\omega)) \operatorname{Im}(D(j\omega))}{\operatorname{Re}(D(j\omega))^2 + \operatorname{Im}(D(j\omega))^2} \quad (\text{Equation 2.12})$$

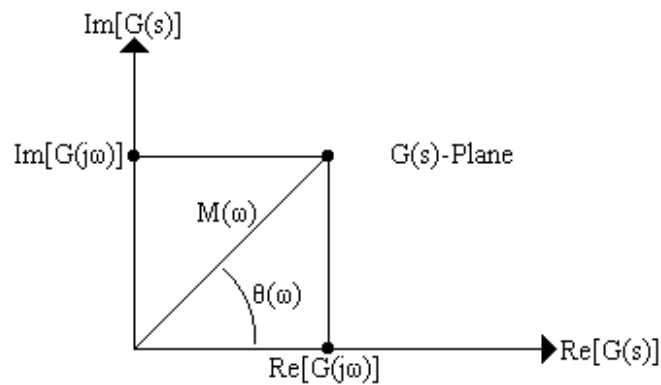
$$\operatorname{Im}(G(j\omega)) = \frac{\operatorname{Im}(N(j\omega)) \operatorname{Re}(D(j\omega)) - \operatorname{Re}(N(j\omega)) \operatorname{Im}(D(j\omega))}{\operatorname{Re}(D(j\omega))^2 + \operatorname{Im}(D(j\omega))^2} \quad (\text{Equation 2.13})$$

If the real and imaginary parts of $G(j\omega)$ are plotted on an Argand diagram (in rectangular coordinates), then the locus of points obtained for $0 \leq \omega \leq \infty$ is called the Polar or Nyquist plot.

The required graph data are taken by converting $G(j\omega)$ into the Euler form,

$$G(j\omega) = M(\omega) e^{j\theta(\omega)} \quad (\text{Equation 2.14})$$

As shown in figure 2.1.

Figure 2.1 - A Point in the $G(s)$ Plane.

This figure shows that,

$$M(\omega) = \sqrt{\text{Re}(G(j\omega))^2 + \text{Im}(G(j\omega))^2} \quad (\text{Equation 2.15})$$

$$\theta(\omega) = \tan^{-1} \frac{\text{Im}(G(j\omega))}{\text{Re}(G(j\omega))} \quad (\text{Equation 2.16})$$

To produce appropriate data, $M(\omega)$ is converted to dB,

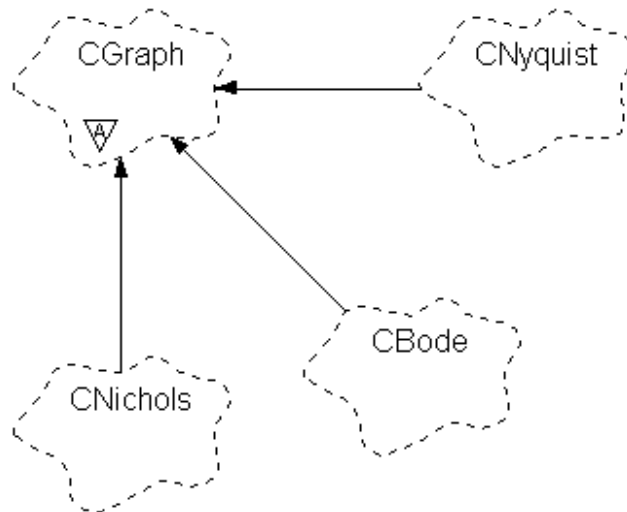
$$M(\omega)_{dB} = 20 \log_{10} (M(\omega)) \quad (\text{Equation 2.17})$$

Also, the inverse tangent function will give the angle in radians. This must be converted to degrees. Care was taken to ensure that the angle $\theta(\omega)$ is measured from the positive real axis in an anti-clockwise direction. As we will see later the above procedure is implemented in the 'calculate' function of the CGraph class.

2.3 Relationships of Class CGraph

The following diagram (Figure 2.2) shows the relationships of the CGraph class inside the program. As we can see CGraph is a base, abstract class that is used for inheritance purposes.

Figure 2.2 - Relationships of Class CGraph.



The classes that inherit from CGraph are the CBode class, the CNyquist class and the CNichols class.

2.4 Declaration of Class CGraph

Class

CGraph

Operations

| No. | Function: (Group A) | Name: |
|-----|---------------------------------|--------------------|
| 1. | Calculate data. | calculate |
| 2. | Draw a line. | gPlot |
| 3. | Set the current position. | gMove |
| 4. | Draw a vertical line. | vertLine |
| 5. | Draw a horizontal line. | horizLine |
| 6. | Draw a circle. | gCircle |
| 7. | Write left-justified text. | gWriteL |
| 8. | Write right-justified text. | gWriteR |
| 9. | Write center-justified text. | gWriteC |
| 10. | Set the graphic mode. | setgraph |
| 11. | Release driver memory. | freeDriverMem |
| 12. | Load graphics driver. | graphAppLoadDriver |
| 13. | Initialize graphics mode. | graphAppInit |
| 14. | Close graphics mode. | graphAppDone |
| 15. | Check for active graphics mode. | graphicsActive |
| 16. | Start graphics mode. | graphicsStart |

| | |
|---------------------------------|--------------|
| 17. Exit graphics mode. | graphicsStop |
| 18. Convert degrees to radians. | rad |
| 19. Convert a real to integer. | round |
| 20. Get the sign of a number. | sgn |
| 21. Round a real number. | trunk |

Data Items

| Item. | Data: | Type: | Name: |
|-------|-----------------------|-----------------|--------|
| A. | X coordinate. | Real number. | xScale |
| B. | Y coordinate. | Real number. | yScale |
| C. | Maximum x coordinate. | Integer number. | xMax |
| D. | Maximum y coordinate. | Integer number. | yMax |
| E. | Value of magnitude. | Real number. | db |
| F. | Value of phase. | Real number. | phi |
| G. | Value of frequency. | Real number. | freq |
| H. | Linear value. | Real number. | Flin |

2.5 Implementation of Class CGraph

In the title of the following descriptions the code *Letter.Number* indicates the group letter and the number of the function as mentioned in the declaration table above.

Function A.1

```
void calculate(InputData iData);
```

Description

Calculates magnitude and phase for a given frequency ω . The function uses the procedure described in part 2.2.

Parameters

Structure iData.

Return Value

NONE.

Function A.2

```
void gPlot(int xp, int yp);
```

Description

Draws a line from current position to "xp", "yp". The graphics of IBM PCs have the origin at the top left hand side. This function inverts "yp" and scales "xp" and "yp" to the maximum position "Xmax", "Ymax". The operation of this function is handled by the *lineto* function declared in the header file *graphics.h*.

Parameters

X and Y coordinate position "xp" and "yp" as integers.

Return Value

NONE.

Function A.3

```
void gMove(int xp, int yp);
```

Description

Moves from current position to "xp", "yp". The graphics on IBM PCs have the origin at the top left hand side. This function inverts "yp" and scales "xp" and "yp" to the maximum position "Xmax", "Ymax". The operation of this function is handled by the *'moveto'* function declared in the header file *'graphics.h'*.

Parameters

X and Y coordinate position "xp" and "yp" as integers.

Return Value

NONE.

Function A.4

```
void vertLine(int xStart, int yStart, int length);
```

Description

Draws a vertical line of a given length starting from "xStart", "yStart". This avoids a one-pixel possible kink in the line which may occur in the *'gPlot'* function. The x-y positions and length are scaled to 640 by 480 pixels and the y position is inverted to give the 0, 0 origin at the bottom left of the screen. The function first moves the current position to "xStart", "yStart" using the *'moveto'* function and then uses *'lineto'* function to draw a vertical line of the desired length.

Parameters

"xStart", "yStart" starting point coordinates and length of line "length" as integers.

Return Value

NONE.

Function A.5

```
void horizLine(int xStart, int yStart, int length);
```

Description

Draws a horizontal line of a given length starting from "xStart", "yStart". This avoids a one-pixel possible kink in the line which may occur in the *'gPlot'* function. The x-y positions and length are scaled to 640 by 480 pixels and the y position is inverted to give the 0, 0 origin at the bottom left of the screen. The function first moves the current position to "xStart", "yStart" using the *'moveto'*

function and then uses *'lineto'* function to draw a horizontal line of the desired length.

Parameters

"xStart", "yStart" starting point coordinates and length of line "length" as integers.

Return Value

NONE.

Function A.6

```
void gCircle(int xCentre, int yCentre, int radius);
```

Description

Draws a circle in 40 segments scaled to the current graphics mode centered on "xCentre", "yCentre" with given radius on screen based on nominal 640 pixels horizontally and 480 pixels vertically. The function uses *'gplot'* to draw a series of short lines that all together form the shape of a circle. The `M_PI` constant used is actually the value of π and it is defined in the header file *'math.h'*.

Parameters

"xCentre", "yCentre" center point coordinates and the radius "radius" of the circle.

Return Value

NONE.

Function A.7

```
void gWriteL(int x, int y, char* text);
```

Description

Writes a text string "text" to graphics screen nominal 640 by 480 pixels. Text is left-justified to scaled X, Y position. The text is justified by using the appropriate argument in function *'settextjustify'*. *'settextstyle'* defines normal horizontal text which is finally displayed by using the *'outtextxy'* function. All these functions are defined in the header file *'graphics.h'*.

Parameters

"x", "y" starting point coordinates and a text string "text".

Return Value

NONE.

Function A.8

```
void gWriteR(int x, int y, char* text);
```

Description

Writes a text string "text" to graphics screen nominal 640 by 480 pixels. Text is right-justified to scaled X, Y position. The text is justified by using the appropriate argument in function *'setttextjustify'*. *'setttextstyle'* defines normal horizontal text which is finally displayed by using the *'outtextxy'* function. All these functions are defined in the header file *'graphics.h'*.

Parameters

"x", "y" starting point coordinates and a text string "text".

Return Value

NONE.

Function A.9

```
void gWriteC(int x, int y, char* text);
```

Description

Writes a text string "text" to graphics screen nominal 640 by 480 pixels. Text is center-justified to scaled X, Y position. The text is justified by using the appropriate argument in function *'setttextjustify'*. *'setttextstyle'* defines normal horizontal text which is finally displayed by using the *'outtextxy'* function. All these functions are defined in the header file *'graphics.h'*.

Parameters

"x", "y" starting point coordinates and a text string "text".

Return Value

NONE.

Function A.10

```
void setgraph(void);
```

Description

Finds the type of screen of the computer and sets graphics mode. Also sets the scale factors "xScale" and "yScale" used in *'gplot'* and *'gmove'*. It uses *'getmaxx'* and *'getmaxy'* functions defined in header file *'graphics.h'* to determine the screen dimensions in pixels.

Parameters

NONE.

Return Value

NONE.

Function A.11

```
void freeDriverMem(void);
```

Description

Release memory occupied by driver pointer. If the pointer exists it is deleted by using the *'delete'* operator.

Parameters

NONE.

Return Value

NONE.

Function A.12

```
Boolean graphAppLoadDriver(int driverNum);
```

Description

Loads the graphics driver. The function searches the current directory to locate the appropriate BGI file, then opens and loads this file for use by the program.

Parameters

The value of the selected driver constant "driverNum".

Return Value

A Boolean value which is true if the operation is successful.

Function A.13

```
Boolean graphAppInit(int aDriver, int aMode,  
char* aBGIPath, Boolean loadAtInit);
```

Description

Initializes a BGI file. If 'loadAtInit' is true, it tries to locate and load the driver. Returns true if 'loadAtInit' succeeds or is set to false. The function does not "own" 'bgiPath', but instead this is passed as a pointer to a string that is allocated elsewhere. 'graphAppInit' does not de-allocate 'bgiPath' when finished.

Parameters

Integer Constants "aDriver", "aMode", the driver's path "aBGIPath" and a Boolean "loadAtInit".

Return Value

Boolean value which is true if no error occurs.

Function A.14

```
void graphAppDone(void);
```

Description

Closes graphics mode and releases the occupied memory. Uses the function *'closegraph'* to close the graphics mode and the function *'freeDriverMem'* to release the memory.

Parameters

NONE.

Return Value

NONE.

Function A.15

Boolean graphicsStart(void);

Description

Starts the graphics mode. It uses the function *'initialize'* to open a graphics mode window and the function *'setgraph'* to set the required settings.

Parameters

NONE.

Return Value

Boolean value which is true if no error occurs.

Function A.16

Boolean graphicsActive(void);

Description

Checks if graphics mode is active. If the *'graphActive'* variable is true then graphics mode is active, if it is false then graphics mode is inactive.

Parameters

NONE.

Return Value

Boolean value which is true if graphics mode is active.

Function A.17

void graphicsStop(void);

Description

Exits graphics mode and returns to the Turbo Vision environment of Control Project. It uses the function *'closegraph'* to exit from graphics mode and the method *'TProgram::application→redraw'* to display the Turbo Vision environment.

Parameters

NONE.

Return Value

NONE.

Function A.18

```
double rad(double x);
```

Description

Returns the radian equivalent of an angle.

Parameters

The angle "x".

Return Value

The equivalent of the angle in radians.

Function A.19

```
int round(double x);
```

Description

Converts a real number to an integer.

Parameters

A real number "x".

Return Value

The rounded integer.

Function A.20

```
int sgn(double x);
```

Description

Returns 1 if "x" is positive, 0 if "x" is zero and -1 if "x" is negative.

Parameters

Real number "x".

Return Value

1 if $x > 0$, 0 if $x = 0$, -1 if $x < 0$.

Function A.21

```
int trunk(double x);
```

Description

Rounds up if "x" is positive, else rounds down.

Parameters

A real number "x".

Return Value

The rounded integer.

Chapter 3 - Bode Plots

3.1 General

This chapter is dedicated to the analysis of the class `CBode`. This class takes the input data, calculates the required graph data and plots them in a graphics mode window. Using the `CBode` class Control Project allows you to easily obtain the Bode plot of a transfer function. The user has also the choice of changing the settings of the graph if this is required.

3.2 Theory - Bode Plots

The log - magnitude and phase frequency response curves as functions of $\log \omega$ are called Bode plots. These are two plots in rectangular coordinates in which the magnitude is expressed in decibels (dB), and the phase angle in degrees, both plotted as functions of the logarithm of frequency in rad / unit time. One of the main advantages of using a logarithmic scale for the magnitude ratio is the ease with which the dynamic elements in a control loop can be manipulated. At a given frequency, the magnitude ratio of an open - loop transfer function is obtained by multiplying together the individual angles. By using Bode plots, both these tasks are accomplished by graphical summation. Consider the following transfer function,

$$G(s) = \frac{K (s + z_1) (s + z_2) \dots (s + z_k)}{s^m (s + p_1) (s + p_2) \dots (s + p_n)} \quad (\text{Equation 3.1})$$

The magnitude frequency response is the product of the magnitude frequency responses of each term,

$$G(s) = \frac{K |(s + z_1)| |(s + z_2)| \dots |(s + z_k)|}{|s^m| |(s + p_1)| |(s + p_2)| \dots |(s + p_n)|} \Big|_{s \rightarrow j\omega} \quad (\text{Equation 3.2})$$

Thus, if we know the magnitude response of each pole and zero term, we can find the total magnitude response. The process is simplified by working with the logarithm of the magnitude since the zero terms' magnitude responses would be added and the pole terms' magnitude responses subtracted, rather than respectively multiplied or divided, to yield the logarithm of the total magnitude response. By converting the magnitude response into dB, we obtain,

(Equation 3.3)

$$\begin{aligned} \text{Magnitude} &= 20 \log K + 20 \log|(s + z_1)| + 20 \log|(s + z_2)| + \dots \\ \dots - 20 \log|s^m| - 20 \log|(s + p_1)| - \dots \quad |s \rightarrow j\omega \end{aligned}$$

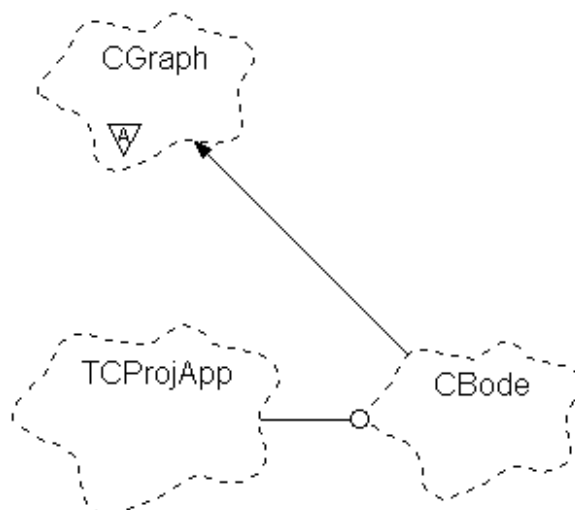
Therefore, if we know the response of each term, the algebraic sum would yield the total response in dB. From equation 3.2 the phase frequency response is the sum of the phase frequency response curves of the zero terms minus the sum of the phase frequency response curves of the pole terms,

$$\begin{aligned} \text{Phase} &= \tan K + \tan|(s + z_1)| + \tan|(s + z_2)| + \dots \\ \dots - \tan|s^m| - \tan|(s + p_1)| - \dots \quad |s \rightarrow j\omega \end{aligned} \quad (\text{Equation 3.4})$$

3.3 Relationships of Class CBode

The CBode class defines the properties, actions and characteristics of the CBode objects. The following diagram shows the relationships of the CBode class inside the program.

Figure 3.1 - CBode class relationships.



As it can be seen CBode inherits from the abstract class CGraph. The interface class TCProjApp uses CBode so that the obtained graph can be displayed from the Turbo Vision environment.

3.4 Declaration of Class CBode

Class

CBode

Operations

| No. | Function: (Group B) | Name: |
|-----|--------------------------------------|---------|
| 1. | Draw a Bode grid. | grid |
| 2. | Set positions on the magnitude plot. | dbxy |
| 3. | Set positions on the phase plot. | phixy |
| 4. | Plot magnitude. | dbplot |
| 5. | Plot phase. | phiplot |
| 6. | Draw Bode plot. | draw |

Data Items

NONE.

3.5 Implementation of Class CBode

Function B.1

```
void grid(void);
```

Description

Draws the Bode grid outline. One degree is 0.6 pixels and one dB is three pixels.

Parameters

NONE.

Return Value

NONE.

Function B.2

```
void dbxy(double& db, double step, int decade,
          int& xPos, int& yPos);
```

Description

Converts decibel and frequency into X and Y positions on the Bode Plot.

Parameters

The magnitude "db", the frequency "step" and the position "xPos", "yPos".

Return Value

NONE.

Function B.3

```
void phixy(double& phi, double step, int decade,
           int& xPos, int& yPos);
```

Description

Converts phase shift and frequency into X & Y positions on the Bode Plot.

Parameters

The phase "phi", the frequency "step" and the position "xPos", "yPos".

Return Value

NONE.

Function B.4

```
void dbplot(double db, double step, int decade,
            Boolean first);
```

Description

Draws a line from current position to "db" on the Bode grid.

Parameters

The magnitude "db", the frequency "step" and a boolean variable "first". When "first" is true 'dbplot' will plot the first point of the graph.

Return Value

NONE.

Function B.5

```
void phiplot(double phi, double step, int decade,
            Boolean first);
```

Description

Draws a line from current position to "phi" on the Bode grid.

Parameters

The phase "phi", the frequency "step" and a boolean variable "first". When "first" is true 'phiplot' will plot the first point of the graph.

Return Value

NONE.

Function B.6

```
void draw(InputData iData);
```

Description

Draws a Bode Plot.

Parameters

NONE.

Return Value

NONE.

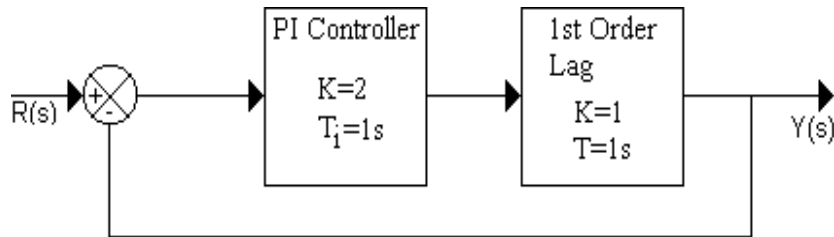
3.6 The Interface

To obtain a Bode plot in Control Project, select '*Bode Plot*' from the '*Graph*' menu by either using a mouse or keyboard by pressing 'Alt + G' and then 'B'. The Input dialog of

Bode plots appears (Figure 3.3). This dialog allows you to enter a transfer function by typing its numerator and denominator coefficients.

Let us consider the following system represented in the block diagram of figure 3.2.

Figure 3.2 - Block diagram of a control system.



The system consists of a proportional plus integral (PI) controller and a first order lag. The PI controller has a gain $K=2$ and an integral time $T_i=1s$. The general transfer function of a PI controller is,

$$\frac{K (s + \frac{1}{T_i})}{s} \quad (\text{Equation 3.5})$$

The first order lag of our system (Figure 3.2) has a gain $K=1$ and a time constant $T=1s$. The general transfer function of a first order lag is,

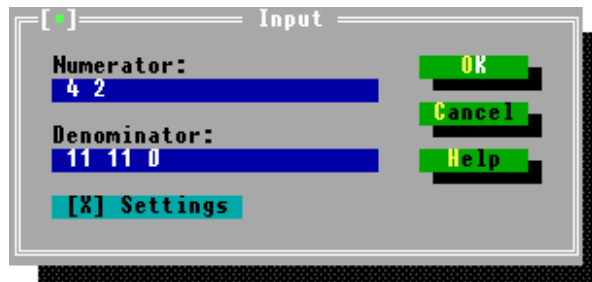
$$\frac{K/T}{s + 1/T} \quad (\text{Equation 3.6})$$

By substituting the values of the system in equations 3.5 and 3.6 and multiplying them together we obtain the open loop transfer function $G(s)$ of the system which is,

$$G(s) = \frac{4s + 2}{11s^2 + 11s} \quad (\text{Equation 3.7})$$

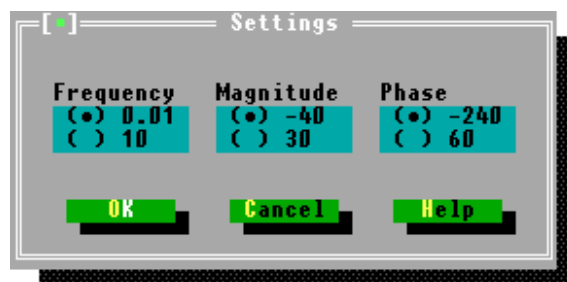
The input that defines $G(s)$ in the Input dialog of Control Project is shown in figure 3.3.

Figure 3.3 - Input window of Bode plots.

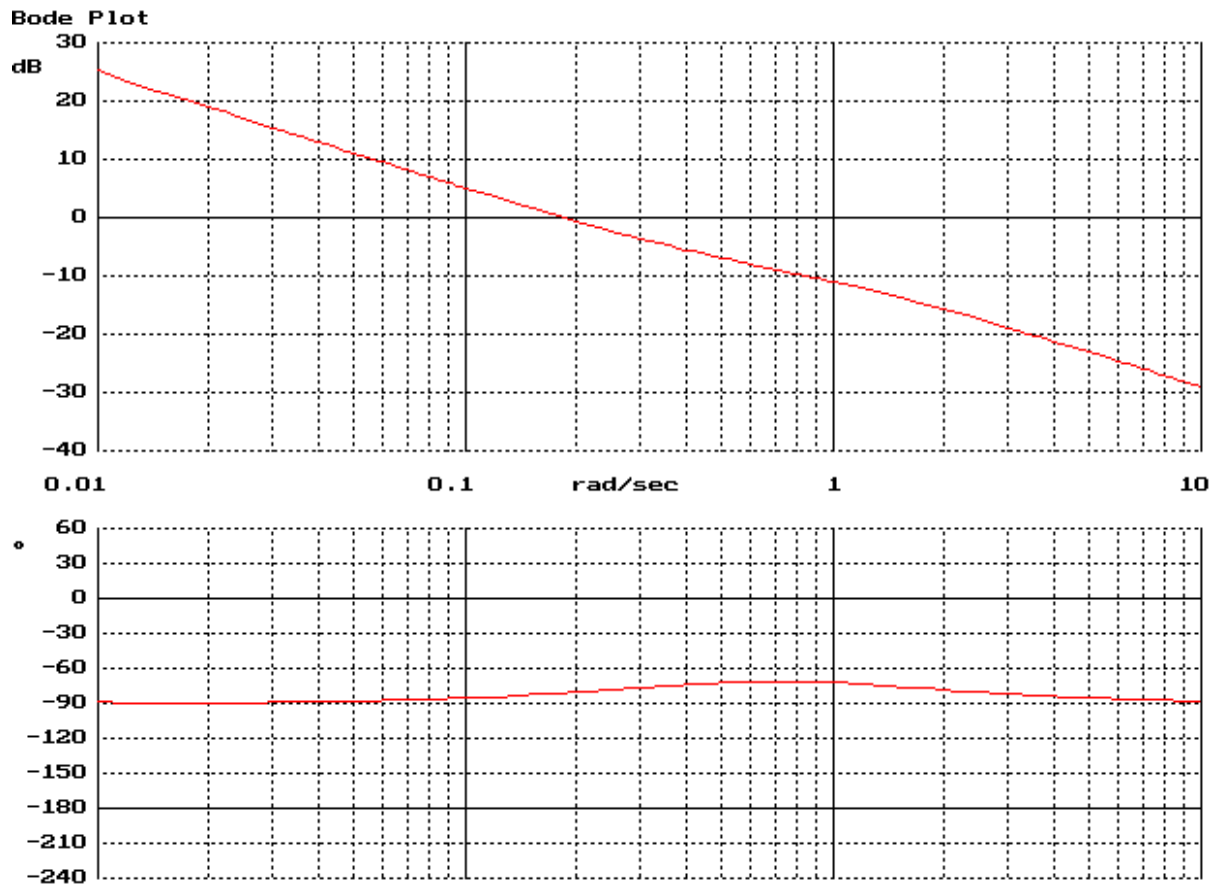


In figure 3.3 it can be seen that the user has clicked the 'Settings' check box that enables the user to change the settings of the plot. Pressing 'OK' displays the settings window seen in figure 3.4.

Figure 3.4 - Settings window of Bode plots.



Pressing the 'OK' button of the settings window will display the Bode plot of $G(s)$ which can be seen in figure 3.5,

Figure 3.5 - Bode Plot output of $G(s)$.

From this Bode plot we can see that the magnitude is decreasing as the frequency increases from 0.01 to 10 rad/sec. Also, the phase is constant to a value of -90° for frequencies of about 0.012 rad/sec to 0.029 rad/sec. The unity gain on a Bode plot corresponds to 0dB, so the criterion for stability is the gain curve must cross the zero dB axis before the phase shift reaches -180° . This conclusion can be wrong if the phase shift curve crosses the -180° line several times. That's why Bode plots allow us to make only a prediction on the stability of the system. The phase margin of a system is the number of degrees by which the phase angle is numerically smaller than the critical angle of -180° at gain crossover. Practically on a Bode plot it is the vertical distance

in degrees between the phase curve and the -180° axis when magnitude is zero. The gain margin is the factor by which the magnitude ratio must be multiplied at phase crossover to make it unity. On a Bode plot the gain margin of the system is the vertical distance in dB between the magnitude curve and the 0 dB axis when phase is -180° .

Chapter 4 - Nyquist Diagrams

4.1 General

This chapter is dedicated to the analysis of the class C_Nyquist. This class takes the input data, calculates the required graph data and plots them in a graphics mode window. Using the C_Nyquist class Control Project allows you to easily obtain the Nyquist diagram of a transfer function. The user has also the choice of changing the settings of the graph, if this is required.

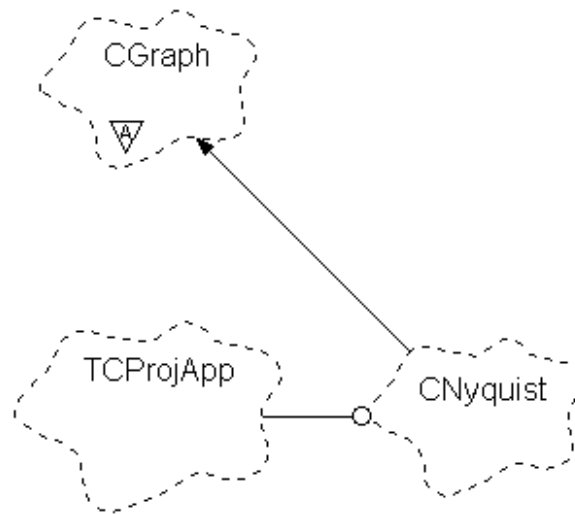
4.2 Theory - Nyquist Diagram

Nyquist diagrams, named after Harry Nyquist (1889-1976), are mappings of the Nyquist contour from the s -plane onto the $G(s)$ - plane of a transfer function $G(s)$. Since the Nyquist contour includes the imaginary s - plane axis, and mappings along this axis produce the polar plot, then polar plots form part of the Nyquist diagram. Usually after testing the stability of a system only the polar plot portion of the diagram is used for system design work. The polar plot is a plot of the gain (magnitude) and phase locus for all frequencies. Nyquist diagrams are particularly useful as they can be used to determine the closed - loop stability of a system directly from the open - loop polar plot.

4.3 Relationships of Class C_Nyquist

The C_Nyquist class defines the properties, actions and characteristics of the C_Nyquist objects. The following diagram shows the relationships of the C_Nyquist class inside the program,

Figure 4.1 - CNyquist class relationships.



As it can be seen CNyquist inherits from the abstract class CGraph. The interface class TCProjApp uses CNyquist so that the obtained graph can be displayed from the Turbo Vision environment.

4.4 Declaration of Class CNyquist

Class

CNyquist

Operations

| No. | Function: (Group C) | Name: |
|-----|---------------------------------------|-------|
| 1. | Draw the Nyquist grid. | grid |
| 2. | Set x, y coordinates on Nyquist grid. | xy |
| 3. | Plot current position. | plot |
| 4. | Draw Nyquist diagram. | draw |

Data Items

NONE.

4.5 Implementation of Class *CNyquist*

Function C.1

```
void grid(void);
```

Description

Draws the Nyquist grid outline. The grid consists of one outer and one inner circle drawn by using the 'gCircle' function and lines passing through them. A small blue circle in the one side of the grid (at -180°) marks the unity gain.

Parameters

NONE.

Return Value

NONE.

Function C.2

```
void xy(int& x, int& y);
```

Description

Sets the x,y coordinates on the Nyquist grid.

Parameters

The current coordinates "x", "y".

Return Value

NONE.

Function C.3

```
void plot(Boolean first);
```

Description

Draws a line from the current position to x,y.

Parameters

Boolean variable "first".

Return Value

NONE.

Function C.4

```
void draw(InputData iData);
```

Description

Draws a Nyquist diagram.

Parameters

An InputData structure "iData".

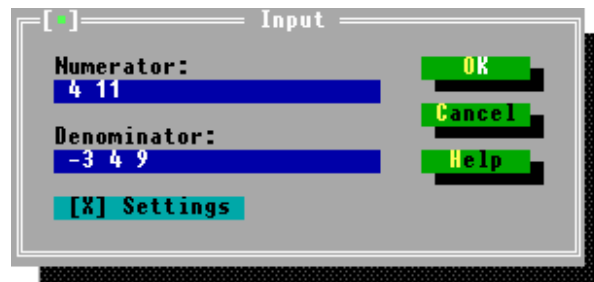
Return Value

NONE.

4.6 The Interface

To obtain a Nyquist diagram in Control Project select '*Nyquist diagram*' from the '*Graph*' menu by either using a mouse or keyboard by pressing 'Alt + G' and then 'q'. The Input dialog of figure 4.2 appears. This dialog allows you to enter a transfer function by typing its numerator and denominator coefficients.

Figure 4.2 - Input window of Nyquist diagrams.

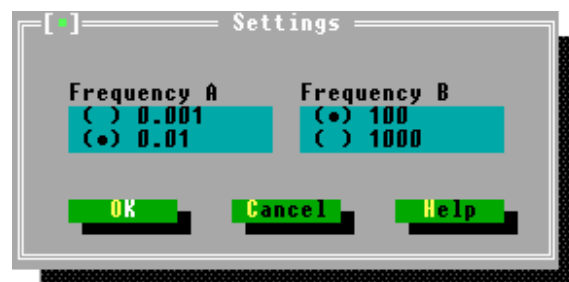


The input shown in figure 4.2 represents the transfer function,

$$G(s) = \frac{4s + 11}{-3s^2 + 4s + 9} \quad (\text{Equation 4.1})$$

In figure 4.2 it can be seen that the user clicked the 'settings' button which allows the change of the diagram's defaults. Pressing 'OK' causes the following settings window to appear.

Figure 4.3 - Nyquist diagrams settings window.



As it can be seen in the above figure the user set a starting frequency of 0.01 rad/sec and a finishing frequency of 100 rad/sec.

Pressing the 'OK' button of the settings window will display the Nyquist diagram of $G(s)$ which can be seen in figure 4.4,

Figure 4.4 - Nyquist diagram output of $G(s)$.

Nyquist Diagram

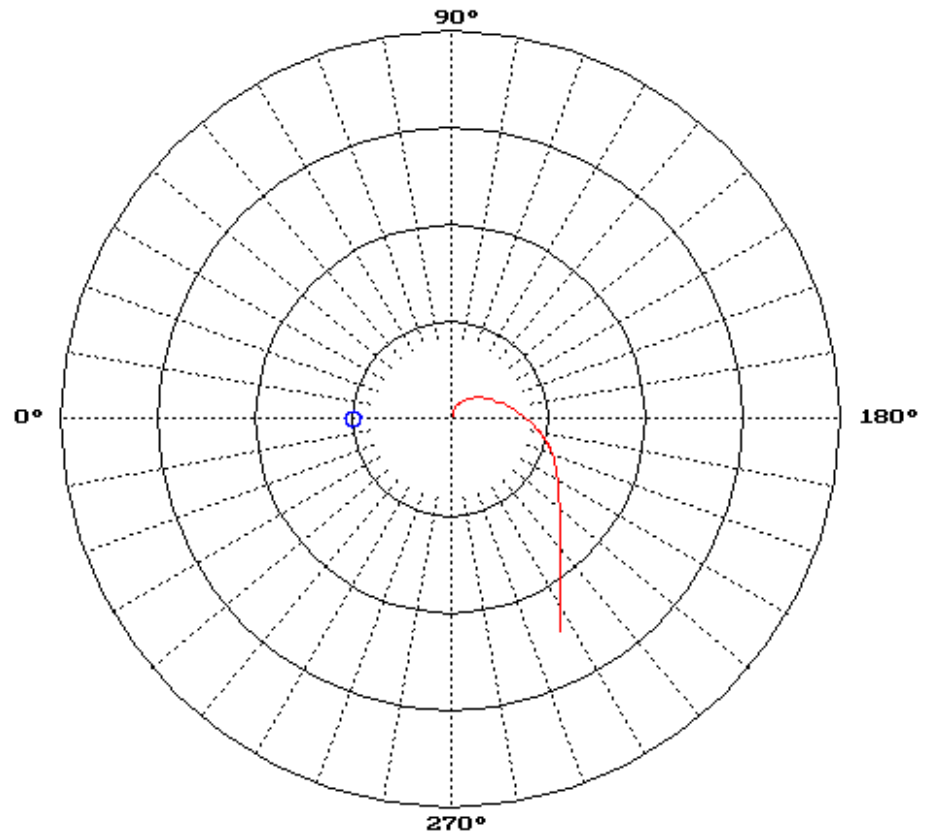


Figure 4.4 shows the Nyquist diagram of $G(s)$ (Equation 4.1) for frequencies from 0.01 to 100 rad/sec. The blue circle on the grid marks the unity gain of the diagram. In general for a system to be stable the curve of the Nyquist diagram must not enclose the unity gain point (at -1 and -180°). Gain and phase margins can also be read from a Nyquist diagram with the aid of a unity radius circle constructed on the origin.

Chapter 5 - Nichols Charts

5.1 General

A Nichols chart is a plot of gain, in decibels, against phase shift. Nichols charts are a very effective method of displaying frequency response. Using the CNichols class Control Project allows you to easily obtain the Nichols chart of a transfer function. The CNichols class takes the input data, calculates the required graph data and plots them in a graphics mode window.

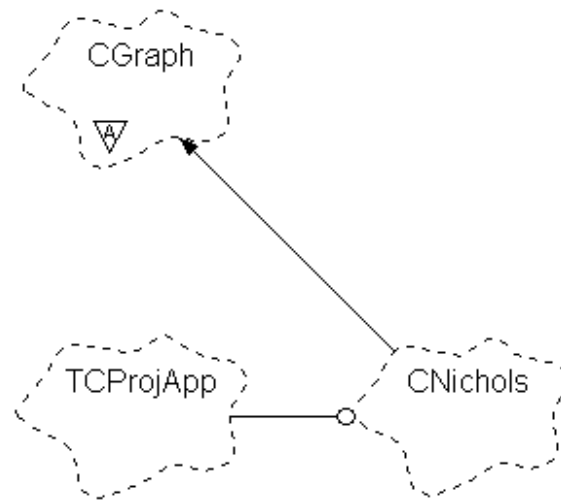
5.2 Theory - Nichols Charts

In principle, a Nichols chart may be used for adjusting a system's open loop gain and phase margins, as well as modifying its closed loop performance specifications. In practice, Nichols charts are used to check the closed-loop characteristics and to make any necessary gain adjustments. A Nichols chart is essentially a transformation of M and N circles on the polar plot into non-circular M and N contours on a plot of dB magnitude versus phase in rectangular coordinates. On a Nichols chart the phase margin is the horizontal distance in degrees between our curve and the -180° axis when magnitude is zero. Similarly, the gain margin of the system is the vertical distance in dB between our curve and the 0 dB axis when phase is -180° .

5.3 Relationships of Class CNichols

The CNichols class defines the properties, actions and characteristics of the CNichols objects. The following diagram shows the relationships of the CNichols class inside the program.

Figure 5.1 - CNichols class relationships.



As it can be seen CNichols inherits from the abstract class CGraph. The interface class TCProjApp uses CNichols so that the obtained chart can be displayed from the Turbo Vision environment.

5.4 Declaration of Class CNichols

Class

CNichols

Operations

| No. | Function: (Group D) | Name: |
|-----|------------------------|---------|
| 1. | Set point coordinates. | xy |
| 2. | Draw a point on chart. | plot |
| 3. | Draw +3dB line. | threedb |
| 4. | Draw 0dB line. | zerodb |
| 5. | Draw Nichols grid. | grid |
| 6. | Draw a Nichols chart. | draw |

Data Items

NONE.

5.5 Implementation of Class CNichols

Function D.1

```
void xy(double ndb, double nphi, int& x, int& y);
```

Description

Set coordinates x, y of a point on Nichols chart.

Parameters

Magnitude "ndb", phase "nphi", coordinates "xp", "yp".

Return Value

NONE.

Function D.2

```
void plot(double ndb, double nphi, Boolean first);
```

Description

Draws a line from current position to "ndb"/"nphi" on the Nichols grid.

Parameters

Magnitude "ndb", phase "nphi", coordinates "xp", "yp".

Return Value

NONE.

Function D.3

```
void threedb(void);
```

Description

Draws the +3dB line of the Nichols grid.

Parameters

NONE.

Return Value

NONE.

Function D.4

```
void zerodb(void);
```

Description

Draws the 0dB line of the Nichols grid.

Parameters

NONE.

Return Value

NONE.

Function D.5

```
void grid(void);
```

Description

Draws a Nichols grid outline scaling 1.5 pixels per degree in x direction and 6 pixels per dB in y direction.

Parameters

NONE.

Return Value

NONE.

Function D.6

```
void draw(InputData id);
```

Description

Draws a Nichols chart. This function takes the starting and finishing frequencies selected by the user (or otherwise the defaults) and for each value of frequency in that range draws the corresponding points that form the Nichols Chart.

Parameters

An InputData structure "id".

Return Value

NONE.

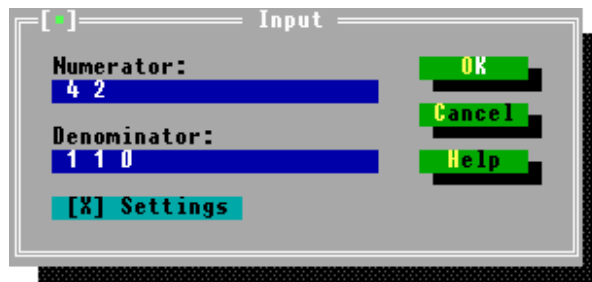
5.6 The Interface

To obtain a Nichols chart in Control Project select '*Nichols Chart*' from the '*Graph*' menu by either using a mouse or keyboard by pressing '*Alt + G*' and then '*N*'. The Nichols chart Input dialog then appears. This dialog allows you to enter a transfer function by typing its numerator and denominator coefficients. Let's consider the transfer function,

$$G(s) = \frac{4s + 2}{s^2 + s} \quad (\text{Equation 5.1})$$

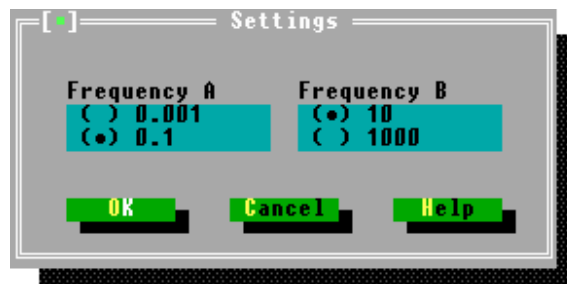
The input that defines $G(s)$ in the Input dialog of Control Project is the one shown in the figure 5.2.

Figure 5.2 - Nichols chart Input window.

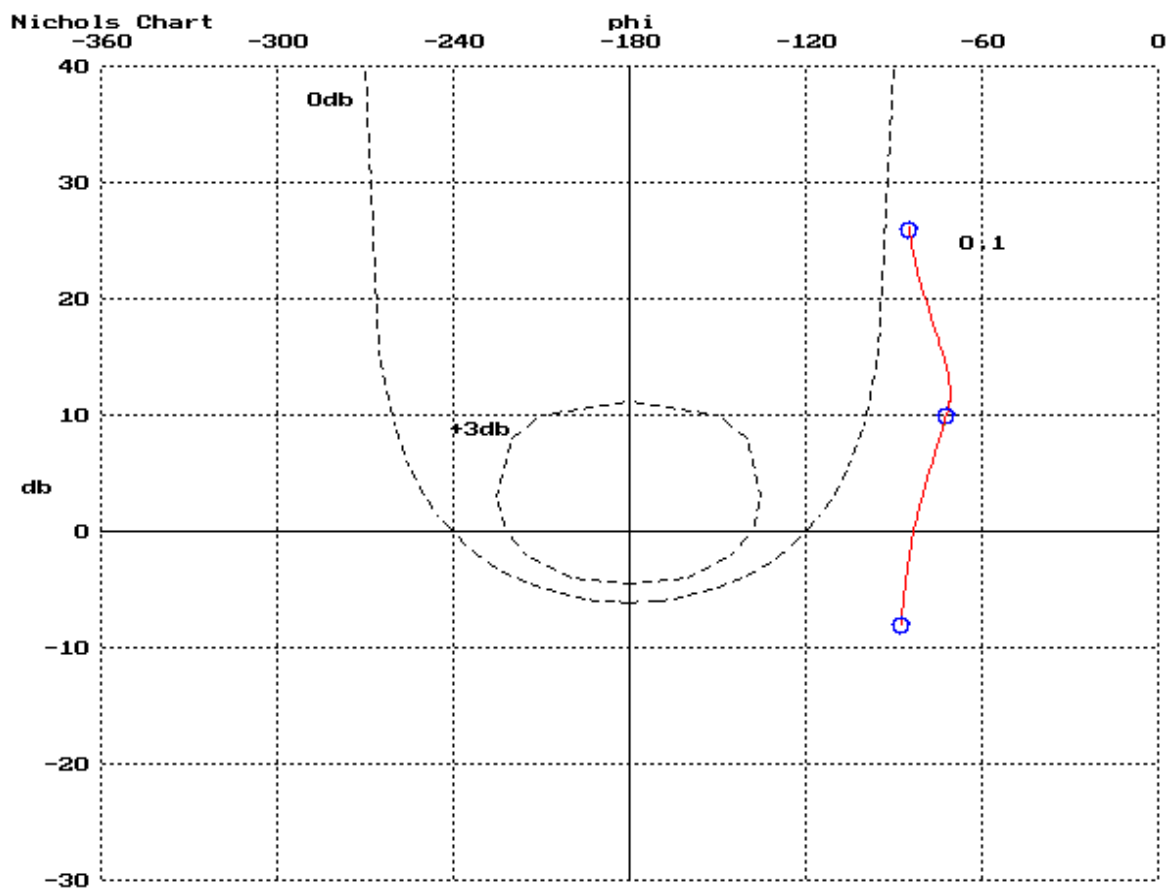


In figure 5.2 it can be seen that the user has clicked the 'Settings' check box that enables the user to change the settings of the chart. Pressing 'OK' displays the settings window seen in figure 5.3.

Figure 5.3 - Nichols Chart Settings



Pressing the 'OK' button of the settings window will display the Nichols Chart of $G(s)$ which can be seen in figure 5.4,

Figure 5.4 - Nichols Chart output of $G(s)$.

The first blue circle of the diagram indicates that the point corresponds to the starting frequency of 0.1 rad/sec as also stated in the graph while the next blue circles represent the points with frequencies 1 and 10 rad/sec (finishing frequency). On a Nichols chart, the critical unity gain point is simply the intercept of the 0 dB and the -180° axis. For stability, the critical point must be to the right of the curve when traversed in the direction of increasing frequency.

Chapter 6 - Inverse Nyquist Diagrams

6.1 General

An inverse Nyquist diagram shows the reciprocal of a function, the locus of the inverse of the gain and negative phase for all frequencies. Every inverse Nyquist diagram also contains an inverse polar plot.

6.2 Theory - Inverse Nyquist Diagrams

Inverse Nyquist diagrams are particularly useful when the system has a minor feedback loop, or if feedback compensation (the deliberate introduction of dynamic elements into the feedback path in order to meet a particular design specification) is required. Familiarity with inverse Nyquist diagrams is a prerequisite for a frequency domain study of multivariable systems (systems with several inputs and outputs). In principle, the inverse Nyquist diagram may be used to achieve the same objectives as direct Nyquist diagrams. The Nyquist stability criterion, gain margin, steady-state error checks and the various closed-loop frequency response measures all have their counterpart in the inverse plane. Lets now assume a transfer function,

$$G(j\omega) = \frac{N(j\omega)}{D(j\omega)} \quad (\text{Equation 6.1})$$

The inverse Nyquist diagram is a plot in rectangular coordinates of $G(j\omega)^{-1}$,

$$G(j\omega)^{-1} = \frac{D(j\omega)}{N(j\omega)} \quad (\text{Equation 6.2})$$

6.3 Code Implementation

Inverse Nyquist diagrams are implemented in Control Project by a member function of the TCProjApp class, named '*iNyquistData*'. This function is described below,

```
Name: void iNyquistData(void);
```

Member of class TCProjApp.

Description

Takes input data and draws an inverse Nyquist diagram. The function uses the *'plotNyquist'* function to plot inverse Nyquist diagrams. The data used for the plot are inverted to obtain the reciprocal of the function.

Parameters

NONE.

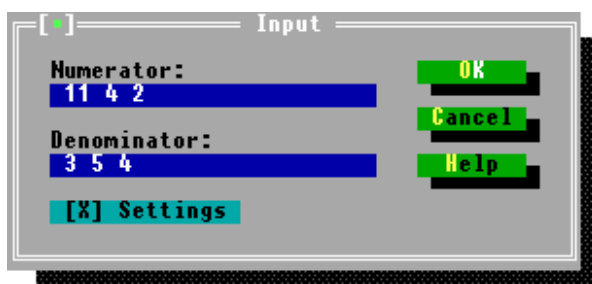
Return value

NONE.

6.4 The Interface

With Control Project one can easily obtain an inverse Nyquist diagram from a transfer function by first selecting the *'Graph'* menu and then *'Inverse Nyquist Diagram'*. This can be done by either using a mouse or pressing *'ALT-G'* and *'a'*. The input dialog of figure 6.1 appears.

Figure 6.1 - Input window of Inverse Nyquist diagram.

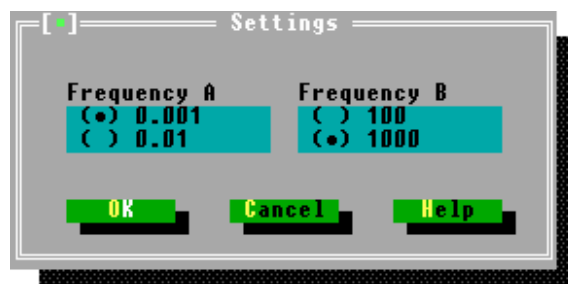


In this dialog box the user entered the coefficients of a polynomial as it can be seen in figure 6.1 where the coefficients represent the transfer function ,

$$G(s) = \frac{11s^2 + 4s + 2}{3s^2 + 5s + 4} \quad (\text{Equation 6.3})$$

Also, in the same figure we can see that the user clicked the 'Settings' check box which allows you to change the default settings of the diagram. Clicking the 'OK' button with the mouse or pressing 'ALT-O' displays the following settings window seen in figure 6.2.

Figure 6.2 - Inverse Nyquist diagram settings.



As it can be seen from figure 6.2 the user has selected a starting frequency A of 0.001 rad/sec and a finishing frequency B of 1000 rad/sec for the inverse Nyquist diagram. Pressing 'OK' displays the inverse Nyquist diagram of equation 6.3 that can be seen in figure 6.3.

Figure 6.3 - Inverse Nyquist diagram of $G(s)$ (Equation 6.3).

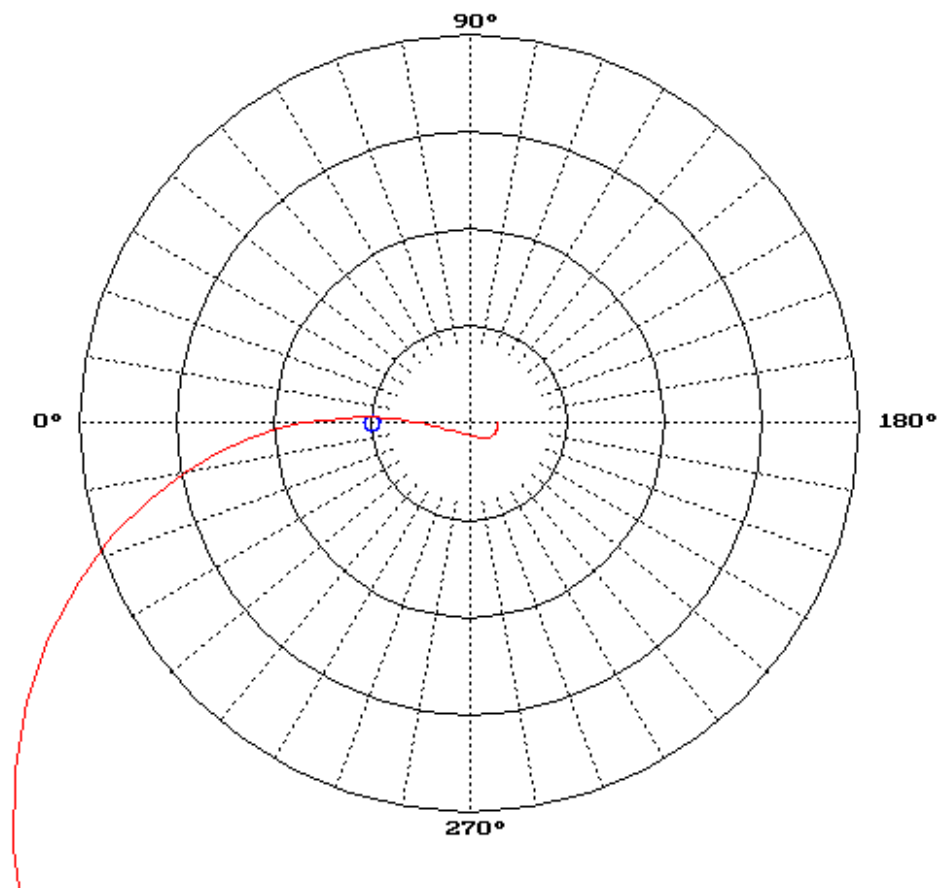
Inverse Nyquist Diagram

Figure 6.2 shows the inverse Nyquist diagram of $G(s)$ (Equation 6.3) for frequencies from 0.001 to 1000 rad/sec. The blue circle on the grid marks the unity gain of the diagram.

Chapter 7 - Routh-Hurwitz Criteria

7.1 General

This chapter discusses the Routh-Hurwitz criteria, a method of determining the stability of a system using its characteristic equation. It involves calculating the number (not the location) of characteristic roots within the unstable right half s-plane. The number of roots in the stable left half s-plane and the number of roots on the imaginary axis may also be found. Apart from the stability checks, usually carried out before determining root locations, the method may also be used to establish the limiting values for a variable parameter beyond which a system would become unstable.

7.2 Theory - Routh-Hurwitz Criteria

Absolute stability requires only that all roots of the system characteristic equation,

$$a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0 = 0 \quad (\text{Equation 7.1})$$

(that is, the poles of its transfer function) lie in the left half s-plane. As it is known, if any of the coefficients are zero or if not all coefficients have the same sign, there will be roots on or to the right of the imaginary axis. If all coefficients are present and have the same sign, which can be taken to be positive without loss of generality, the Routh-Hurwitz criteria provides a quick method for determining absolute, but not relative, stability from the coefficients, without calculating the roots. It shows how many system poles are in the right half s-plane or on the imaginary axis. The general form of a characteristic equation (Equation 7.1), produces a Routh array arranged as follows,

| | | | | | |
|-----------|-----------|-----------|-----------|----------|----------|
| s^n | a_n | a_{n-2} | a_{n-4} | \dots | 0 |
| s^{n-1} | a_{n-1} | a_{n-3} | a_{n-5} | \dots | 0 |
| s^{n-2} | b_1 | b_2 | b_3 | \dots | 0 |
| s^{n-3} | c_1 | c_2 | c_3 | \dots | 0 |
| \vdots | \vdots | \vdots | \vdots | \vdots | \vdots |
| s_1 | y_1 | y_2 | 0 | 0 | 0 |
| s_0 | z_1 | 0 | 0 | 0 | 0 |

Where,

$$b_1 = \frac{1}{a_{n-1}}(a_{n-1} a_{n-2} - a_n a_{n-3}), \quad b_2 = \frac{1}{a_{n-1}}(a_{n-1} a_{n-4} - a_n a_{n-5}) \dots$$

$$c_1 = \frac{1}{b_1}(b_1 a_{n-3} - b_2 a_{n-1}), \quad c_2 = \frac{1}{b_1}(b_1 a_{n-5} - b_3 a_{n-1}) \dots$$

Calculations in each row are continued until only zero elements remain. In each of the last two rows the second and following elements are zero. It can be shown that the elements in any row can be multiplied by an arbitrary positive constant without affecting the results. This can be used to simplify the arithmetic. The Routh-Hurwitz criteria states,

1. A necessary and sufficient condition for stability is that there are no changes of sign in the elements of the first column of the array.
2. The number of these sign changes is equal to the number of roots in the right-half s-plane.
3. If the first element in a row is zero, it is replaced by a very small positive number ε , and the sign changes when $\varepsilon \rightarrow 0$ are counted after completing the array.
4. If all elements in a row are zero, the system has poles in the right-half plane or on the imaginary axis.

7.3 Routh-Hurwitz Criteria in Control Project

Routh-Hurwitz criteria is implemented in Control Project by two member functions of the TCProjApp class, named '*routh*' and '*routhData*'. The following description of these functions is now given.

Name: **void routh(void);**

Member of class TCProjApp.

Description

Sets and calculates the elements, of the Routh array. It also makes a decision on the stability of the system.

Parameters

NONE.

Return value

NONE.

Name: **void routhData(void);**

Member of class TCProjApp.

Description

Draws the input window, takes input data and determines the stability of the system using the '*routh*' function.

Parameters

NONE.

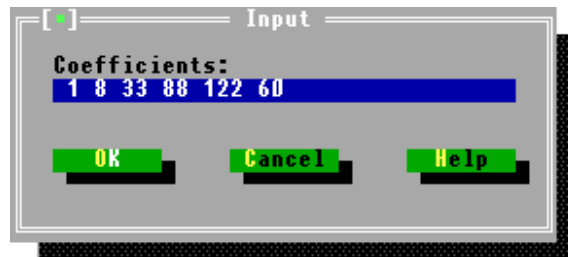
Return value

NONE.

7.4 The Interface

With Control Project anyone can easily apply the Routh-Hurwitz criteria to a characteristic equation by first selecting the 'Method' menu and then 'Routh-Hurwitz'. This can be done by either using a mouse or pressing 'ALT-M' and 'R'. The input dialog of figure 7.1 appears.

Figure 7.1 - Input window of Routh-Hurwitz criteria.

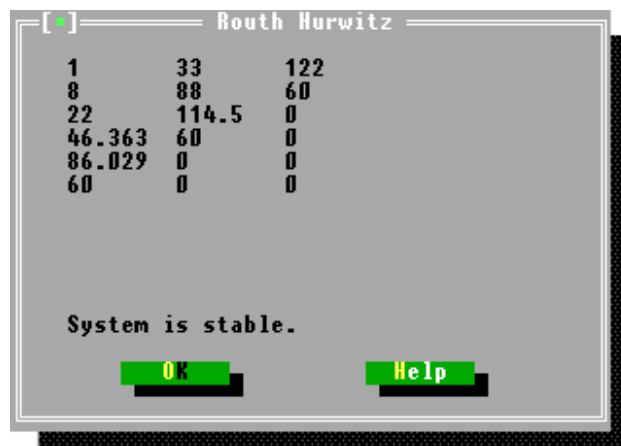


In the input window the user can enter the coefficients of a characteristic equation as in the case of figure 7.1 were the coefficients represent the characteristic equation ,

$$s^5 + 8s^4 + 33s^3 + 88s^2 + 122s + 60 = 0 \quad (\text{Equation 7.2})$$

Clicking the 'OK' button with the mouse or pressing 'ALT-O' displays the following output window seen in figure 7.2.

Figure 7.2 - Output of Routh-Hurwitz criteria.



This window displays the Routh table as well as the decision of the program on the stability of the system. As we can see, in the first column of the Routh table we don't

have a sign change. This means that the system is stable as it is also stated by the program.

Chapter 8 - Horner's Method

8.1 General

This chapter discusses the Horner's method, a method that gives the form of a polynomial if we were to move its imaginary axis from zero to a chosen shift value.

8.2 Theory - Horner's Method

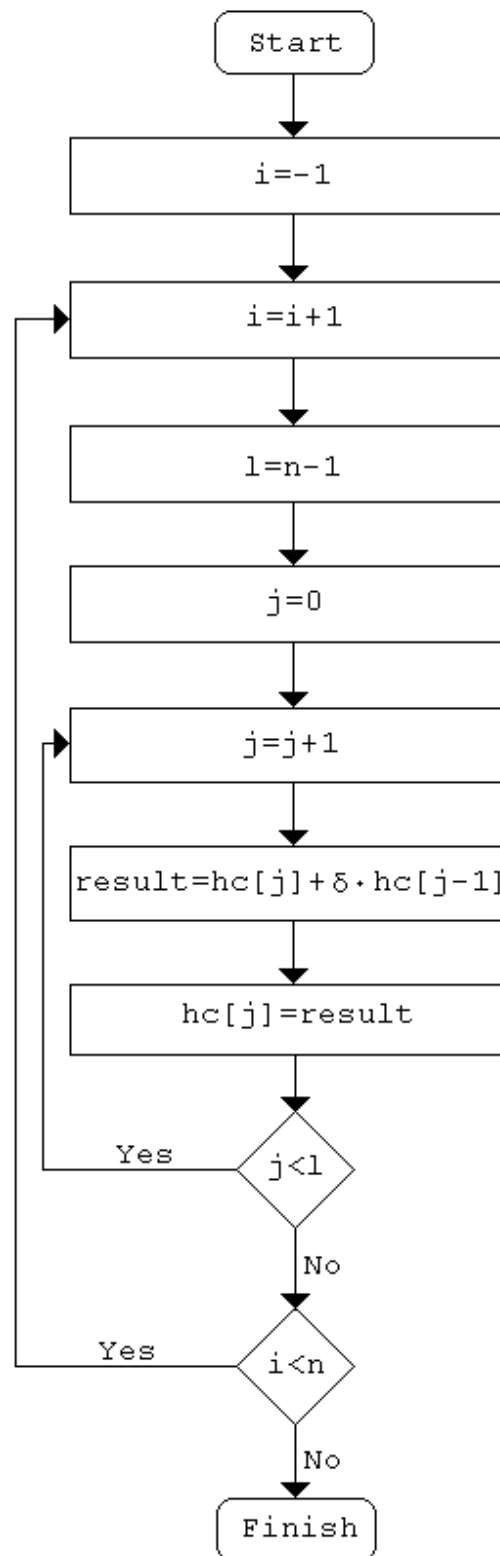
In all control systems the minimum time constant performance specification can be drawn as a line parallel to, but to the left of, the imaginary axis in the s-plane. If all the system's poles are to the left of this performance line, then the slowest rate of decay for any transient is established. The system's absolute stability may be generally defined as the distance δ of this performance line from the imaginary axis. In many control systems it is not sufficient to state merely that the system is stable. Often it is desirable to know a system's margin of absolute stability - that is, how fast the transients of a system will decay after some forcing disturbance. Typically, this requires all the roots of the system's characteristic equation to lie to the left of a line parallel to the imaginary axis and passing through the point $s = -\delta$, where δ defines the slowest permissible rate of decay.

Translation of the imaginary axis is easily carried out, using Horner's method as follows. Let the coefficients of the shifted polynomial $D(s-\delta)$ be b_0, b_1, \dots, b_n and the coefficients of $D(s)$ be a_0, a_1, \dots, a_n . To find the coefficients of $D(s-\delta)$, initially let,

$$b_i = a_i, \text{ for } i = 0, 1, \dots, n$$

The procedure shown as a flow diagram in figure 8.1 is then followed and the resulting coefficients b_0, b_1, \dots, b_n will be the coefficients of the shifted polynomial $D(s-\delta)$.

Figure 8.1 - Flow diagram of Horner's method algorithm.



8.3 Horner's Method in Control Project

Horner's method is implemented in Control Project by two member functions of the TCProjApp class, named '*horner*' and '*hornerData*'. These functions are described below,

Name: **void horner(int shift);**

Member of class TCProjApp.

Description

Takes input data and determines the coefficients of the shifted polynomial. The algorithm used to determine these coefficients can be seen in the flow chart of figure 8.1.

Parameters

The chosen value of shift of the imaginary axis '*shift*' as integer.

Return value

NONE.

Name: **void hornerData(void);**

Member of class TCProjApp.

Description

Draws the input window, takes input data and determines the new coefficients using the '*horner*' function.

Parameters

NONE.

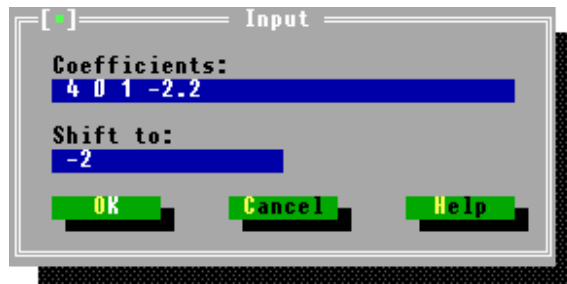
Return value

NONE.

8.4 The Interface

With Control Project one can easily apply the Horner's method to a polynomial by first selecting the 'Method' menu and then 'Horner'. This can be done by either using a mouse or pressing 'ALT-M' and 'e'. The input dialog of figure 8.2 appears.

Figure 8.2 - Input window of Horner's method.



In this dialog box the user enters the coefficients of a polynomial as in the case of figure 8.2 were the coefficients represent the polynomial ,

$$4s^3 + s - 2.2 = 0 \quad (\text{Equation 8.1})$$

Also, in the same figure we can see that the user entered a value of -2 for the shift of the imaginary axis. Clicking the 'OK' button with the mouse or pressing 'ALT-O' displays the following output window seen in figure 8.3.

Figure 8.3 Output of Horner's method.



| Coefficient: | Order: |
|--------------|--------|
| 4 | 3 |
| -24 | 2 |
| 49 | 1 |
| -36.2 | 0 |

This window displays the resulting coefficients of the new polynomial, as obtained by applying Horner's method. In our case (Equation 8.1) the resulting coefficients are those seen in figure 8.3 and they represent the polynomial,

$$4 s^3 - 24 s^2 + 49 s - 36.2 = 0 \quad (\text{Equation 8.2})$$

Equation 8.2 is actually the polynomial obtained by shifting the imaginary axis of Equation 8.1 from 0 to -2 .

Chapter 9 - The User Interface

9.1 General

This chapter is dedicated to the analysis of the TCProjApp class which draws the Turbo Vision environment of Control Project and connects it with the graphics mode used for some of the program's functions. It uses the three graph classes, CBode for Bode plots, CNyquist for Nyquist diagrams, and CNichols for Nichols charts. It also contains the member functions that are responsible for the Inverse Nyquist diagrams, the Routh - Hurwitz and Horner methods as described in chapters 6, 7 and 8 respectively.

9.2 Declaration of Class TCProjApp

Class

TCProjApp

Operations

| No. | Function: (Group E) | Name: |
|-----|--|-------------|
| 1. | Constructor. | TCProjApp |
| 2. | Get an event. | getEvent |
| 3. | Set application's colors. | getPalette |
| 4. | Draw the 'about' box. | about |
| 5. | Draw the Bode plot Settings box. | bodeMore |
| 6. | Draw the Bode plot Input box. | bodeData |
| 7. | Display a Bode plot. | plotBode |
| 8. | Draw the Nyquist diagram Settings box. | nyquistMore |
| 9. | Draw the Nyquist diagram Input box. | nyquistData |
| 10. | Display a Nyquist diagram. | plotNyquist |
| 11. | Draw the Nichols chart Settings box. | nicholsMore |
| 12. | Draw the Nichols chart Input box. | nicholsData |
| 13. | Display a Nichols chart. | plotNichols |
| 14. | Handles defined events. | handleEvent |

| | |
|--------------------------|----------------|
| 15. Draws a status line. | initStatusLine |
| 16. Draws the menu bar. | initMenuBar |
| 17. Destructor. | ~TCProjApp |

Data Items

| Item. | Data: | Type: | Name: |
|-------|---------------|---------------------|-------|
| A | Input values. | Structure InputData | iData |

9.3 Implementation of Class TCProjApp

In the title of the following descriptions the code *Letter.Number* indicates the group letter and the number of the function as mentioned in the declaration table.

Function E.1

```
TCProjApp(void);
```

Description

TCProjApp Constructor. Sets the class and initializes input data.

Parameters

Inherits TProgInit.

Return Value

NONE.

Function E.2

```
void getEvent(TEvent& event);
```

Description

Configures system events. Sets keyboard and mouse events as well as the help file of the program.

Parameters

A TEvent type "event".

Return Value

NONE.

Function E.3

```
TPalette& getPalette(void) const;
```

Description

Set the application's colors. Gray boxes, green highlighted menus etc.

Parameters

NONE.

Return Value

The application's colors.

Function E.4

```
void about(void);
```

Description

Draws the 'about' window which contains version and copyright information of the program.

Parameters

NONE.

Return Value

NONE.

Function E.5

```
void bodeMore(void);
```

Description

Displays the Bode plot settings window. This dialog box allows you to change the parameters of the graph.

Parameters

NONE.

Return Value

NONE.

Function E.6

```
void bodeData(void);
```

Description

Displays the Bode plot input dialog box. This dialog box takes the input data and draws a Bode plot.

Parameters

NONE.

Return Value

NONE.

Function E.7

```
void plotBode(void);
```

Description

Draws a Bode plot in a graphics window inside the Turbo Vision environment of Control Project.

Parameters

NONE.

Return Value

NONE.

Function E.8

```
void NyquistMore(void);
```

Description

Displays the Nyquist diagram settings dialog box. This window allows you to change the parameters of the graph.

Parameters

NONE.

Return Value

NONE.

Function E.9

```
void NyquistData(void);
```

Description

Displays the Nyquist diagram input dialog box. This dialog box takes the input data and draws a Nyquist diagram.

Parameters

NONE.

Return Value

NONE.

Function E.10

```
void plotNyquist(void);
```

Description

Draws a Nyquist diagram in a graphics window inside the Turbo Vision environment of Control Project.

Parameters

NONE.

Return Value

NONE.

Function E.11

```
void NicholsMore(void);
```

Description

Displays the Nichols chart settings dialog box. This window allows you to change the parameters of the graph.

Parameters

NONE.

Return Value

NONE.

Function E.12

```
void NicholsData(void);
```

Description

Displays the Nichols chart input dialog box. This dialog box takes the input data and draws a Nichols chart.

Parameters

NONE.

Return Value

NONE.

Function E.13

```
void plotNichols(void);
```

Description

Draws a Nichols chart in a graphics window inside the Turbo Vision environment of Control Project.

Parameters

NONE.

Return Value

NONE.

Function E.14

```
void handleEvent(TEvent& event);
```

Description

Defines the system's events or otherwise what the respond of the system would be in a certain mouse click or key press.

Parameters

A TEvent object "event".

Return Value

NONE.

Function E.15

```
TStatusLine* initStatusLine(TRect r);
```

Description

Draws the system's status line bar which appears on the lower part of the desktop.

Parameters

A TRect object "r".

Return Value

The value of status line pointer.

Function E.16

```
TMenuBar* initMenuBar(TRect r);
```

Description

Draws the program's menu bar.

Parameters

A TRect object "r".

Return Value

The value of Menu bar pointer.

Function E.17

```
~TCProjApp(void);
```

Description

TCProjApp destructor. Deletes TCProjApp class and the data pointers used.

Parameters

NONE.

Return Value

NONE.

Discussion - Conclusions

Control Project v1.0 is a very useful control engineering tool that introduces some of the most common control engineering techniques available. Its code is written in the C++ programming language, structured using object-oriented techniques. Every class defines the properties, characteristics and behaviors of an object that is responsible for a specific job. For the development of Control Project v1.0, the first step was a software engineering analysis of the project through diagrams and examination of the relationships of the system's objects. Software Engineering tools were also used for these purposes. The whole analysis proved extremely useful because it clarified certain difficulties, it simplified the programming part of the design and it ensured that an optimum code structure was selected. The second step of the design was the programming part where the code of Control Project was written. Control Project v1.0 allows you to examine the behavior of a control system by just entering its transfer function.

Bode plots as those that can be obtained in Control Project, provide a very efficient and accurate method of control design. The logarithmic representation approach of Bode plots has many advantages. They are in general simple to obtain and because the logarithmic scale used is non-linear, it enables us to cover a greater range of frequencies.

Anyone using Control Project can easily obtain the Nyquist diagram of a control system. Every Nyquist diagram also includes the polar plot of the system. Nyquist diagrams are particularly useful as they can be used to determine the closed loop stability of a system directly from the open-loop polar plot.

Control Project allows the user to obtain the Nichols chart of a control system. The Nichols chart is a particularly effective method of displaying frequency response as it is easy to visualize the effect of adjusting controller parameters. In our days, a Nichols chart is commonly used for design though not as much as in the past.

Another type of graph that can easily be obtained using Control Project is the Inverse Nyquist diagram. Every inverse Nyquist diagrams also includes the inverse polar plot of the system. Inverse Nyquist diagrams are particularly useful when the system has a

minor feedback loop, or when feedback compensation is required. They are also useful for the study of multivariable systems (systems with several inputs and outputs).

The user can also examine a control system by using Control Project and the Routh-Hurwitz Criteria or the Horner's method.

Anyone can apply the Routh-Hurwitz criteria easily using Control Project. Routh-Hurwitz criteria generally provides a quick and easy method of establishing a system's stability. A control system can also be tested using Horner's Method which is a very effective way of finding a system's margin of absolute stability.

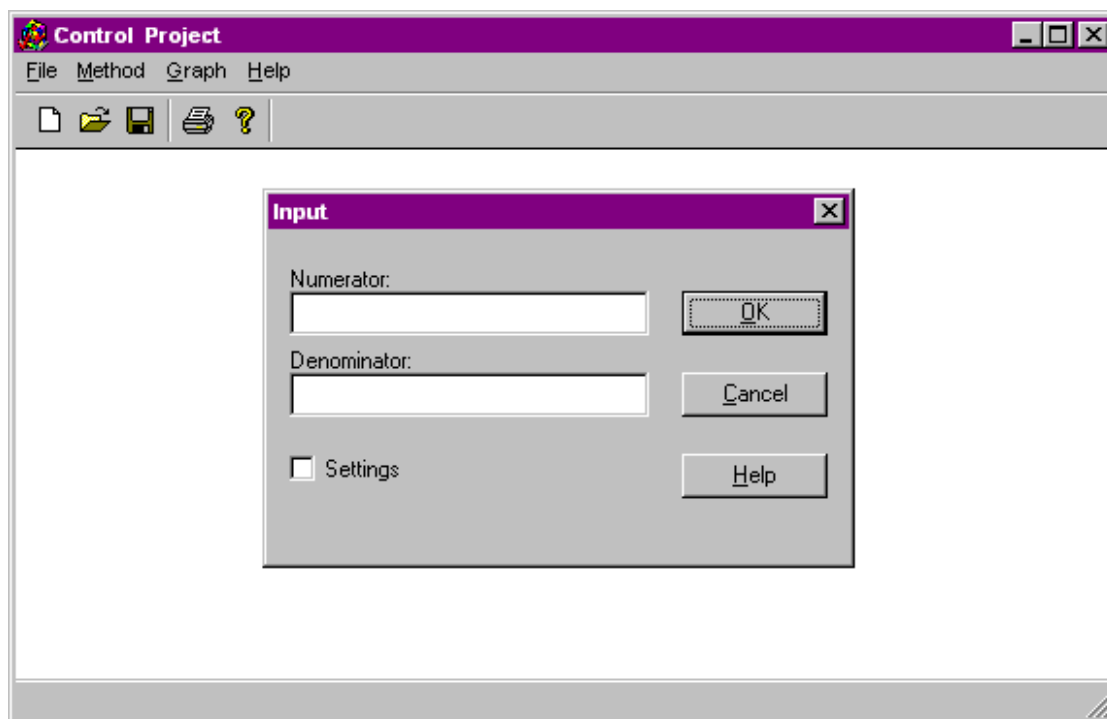
Control Project's environment ensures that all the above methods and graphs can be used efficiently. A specially designed help file provides quick help at any time and for anything on the desktop.

As we have seen previously, Control Project v1.0 uses object oriented code. One of the biggest advantages of object oriented code is that it is easily re-usable and adaptable. In that respect, a new version of Control Project for 32-bit Windows operating systems (Windows 95/Windows NT) is already under construction. At the time of writing the interface development part has already finished (see figure A) while the complete program is planned to be ready by September 1997. Control Project version 2.0 is a single document application and is going to be entirely written using Microsoft Visual C++ v5.0 developer's studio. Some of the features of this version will be,

- A new Office '97 style toolbar for easy access of menu commands.
- The ability to save an obtained graph in a special file which can be opened again at a later time.
- Fully 32-bit code with complete multitasking features available.
- Full graph and results printing using Print Manager.
- All the features of version 1.0. Bode plots, Nyquist diagrams, Nichols charts, inverse Nyquist diagrams, Routh-Hurwitz Criteria and Horner's method.
- Context sensitive help supported by a new and revised 32-bit Help file.

Hopefully by September 1997 anyone would be able to obtain Control Project v2.0 from shareware distribution companies in the Internet like shareware.com or windows95.com.

Figure A - Control Project version 2.0



References

- '*Control Engineering*' - First Edition, 1996
E. A. Parr
Butterworth-Heinemann Publishing.
- '*Control Systems Engineering & Design*' - First Edition, 1989
S. Thomson
Logman Scientific & Technical.
- '*Feedback Control Systems*' - Third Edition, 1994
J. Van De Vegte
Prentice Hall International Editions.
- '*Control Systems*' - First Edition, 1986
N. K. Sinha
CBS College Publishing.
- '*Modern Control System Theory and Application*' - Second Edition, 1980
S. M. Shinnars
Addison - Wesley Publishing Company.
- '*Automatic Control Engineering*' - Fourth Edition, 1987
F. H. Raven
McGraw - Hill Book Company.
- UWCC, EN2058 '*Control Engineering*' notes, 1995/96
T. Meydan and P. Channon.

- '*Object Oriented Programming in C++*' - Second Edition, 1995
R. Lafore
Waite Group Press.
- '*C++, The Complete Reference*' - Second Edition, 1995
H. Schildt
Osborne McGraw - Hill.
- '*Object - Oriented Analysis and Design*' - Second Edition, 1994
G. Booch
The Benjamin / Cummings Publishing Company, inc.

Appendix

A.1 File: CP.CPP

```

#include "capp.h"

Boolean status;

typedef void (* DriverPtrFunction)();
typedef void (* function)();

const MAXSIZE = 80;
const int lastDriver = 10;
char driverName[lastDriver][10] =
    {"CGA",           // 1. CGA
     "CGA",           // 2. MCGA
     "EGAVGA",        // 3. EGA
     "EGAVGA",        // 4. EGA64
     "EGAVGA",        // 5. EGAMONO
     "IBM8514",       // 6. IBM8514
     "HERC",          // 7. HercMono
     "ATT",           // 8. ATT400
     "EGAVGA",        // 9. VGA
     "PC3270"};

Boolean graphActive = False;
DriverPtrFunction driverPtr = 0;
unsigned int driverSize = 0;
char emptyString[] = "";
char* bgiPath = emptyString;
int driver = 0;
int mode = 0;

int vssf(char buffer[40], char* fmt, ...);

struct DialogData
{
    char denominatorData[128];
    char numeratorData[128];
    ushort xDialogData;
};

struct Settings
{

```

```

    ushort aDialogData;
    ushort bDialogData;
    ushort cDialogData;
};

DialogData* inDialogData;
Settings* inSet;

class TCProjApp : public TApplication
{
public:
    InputData iData;
    TCProjApp(void);
    virtual void getEvent(TEvent& event);
    virtual TPalette& getPalette(void) const;
    void about(void);

    void bodeMore(void);
    void bodeData(void);
    void plotBode(void);

    void nyquistMore(void);
    void nyquistData(void);
    void plotNyquist(void);

    void iNyquistData(void);

    void nicholsMore(void);
    void nicholsData(void);
    void plotNichols(void);

    void routh(void);
    void routhData(void);

    void horner(int shift);
    void hornerData(void);

    virtual void handleEvent(TEvent& event);
    static TStatusLine* initStatusLine(TRect r);
    static TMenuBar* initMenuBar(TRect r);
    ~TCProjApp(void);
};

int main(void)
{
    TCProjApp myApp;

```

```

myApp.run();

return 0;
}

//      FUNCTION: TProjApp(void);
//      DESCRIPTION: TProjApp class constructor.
//      PARAMETERS: Inherits TProgInit.
//      RETURN VALUE: NONE.
TProjApp::TProjApp(void):
    TProgInit(& TProjApp::initStatusLine,
              & TProjApp::initMenuBar,
              & TProjApp::initDeskTop);
{
    inDialogData = new DialogData;
    inSet = new Settings;
    // Initialize input data.
    strcpy(inDialogData->denominatorData, "");
    strcpy(inDialogData->numeratorData, "");
    inDialogData->xDialogData = 0;
    inSet->aDialogData = 0;
    inSet->bDialogData = 0;
    inSet->cDialogData = 0;
}

//      FUNCTION: void getEvent(TEvent& event);
//      DESCRIPTION: Configures system events.
//      PARAMETERS: A TEvent type "event".
//      RETURN VALUE: NONE.
void TProjApp::getEvent(TEvent& event)
{
    TWindow* w;
    THelpFile* hFile;
    fpstream* helpStrm;
    static Boolean helpInUse = False;

    TApplication::getEvent(event);
    switch (event.what)
    {
        case evCommand:
            if ((event.message.command == cmHelp) &&
                (helpInUse == False))
            {
                helpInUse = True;
                helpStrm = new fpstream("CPH.HLP",
                                        ios::in|ios::binary);
                hFile = new THelpFile(* helpStrm);
            }
    }
}

```

```

if (!helpStrm)
{
    messageBox("COULD NOT OPEN HELP FILE.",
               mfError | mfOKButton);
    delete hFile;
}
else
{
    w = new THelpWindow(hFile, getHelpCtx());
    if (validView(w) != 0)
    {
        execView(w);
        destroy(w);
    }
    clearEvent(event);
}
helpInUse = False;
}
break;
case evMouseDown:
if (event.mouse.buttons != 1)
    event.what = evNothing;
break;
}
}

//      FUNCTION: TPalette& getPalette(void) const;
//      DESCRIPTION: Set the application's colors.
//      PARAMETERS: NONE.
//      RETURN VALUE: The application's colors.
TPalette& TCProjApp::getPalette(void) const
{
    static TPalette newcolor (cpColor cHelpColor,
                              sizeof(cpColor cHelpColor)-1 );
    static TPalette newblackwhite(cpBlackWhite
                                   cHelpBlackWhite,
                                   sizeof(cpBlackWhite cHelpBlackWhite)-1 );
    static TPalette newmonochrome(cpMonochrome
                                   cHelpMonochrome,
                                   sizeof(cpMonochrome cHelpMonochrome)-1 );
    static TPalette* palettes[] =
    {
        &newcolor,
        &newblackwhite,
        &newmonochrome
    };
    return* (palettes[appPalette]);
}

```

```

}

//      FUNCTION: void about(void);
//      DESCRIPTION: Draws rhe about dialog box.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::about(void)
{
    TDialog* aboutBox = new TDialog(TRect(0, 0, 44, 10), "About");
    aboutBox->helpCtx = hcHAbout;
    aboutBox->insert(new TStaticText(TRect(14, 2, 42, 3), "CONTROL
PROJECT"));
    aboutBox->insert(new TStaticText(TRect(16, 3, 42, 4), "Version
1.0"));
    aboutBox->insert(new TStaticText(TRect(3, 5, 42, 6),
        "Copyright (C) 1997 by Christos Bohoris"));

    aboutBox->insert(new TButton(TRect(16, 7, 26, 9), "OK", cmOK,
bfDefault));
    aboutBox->options |= ofCentered;
    deskTop->execView(aboutBox);
    destroy(aboutBox);
}

//      FUNCTION: void bodeMore(void);
//      DESCRIPTION: Displays the Bode plot settings dialog.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::bodeMore(void)
{
    TDialog* bodeMWin = new TDialog(TRect(20, 5, 60, 16),
"Settings");
    bodeMWin->helpCtx = hcHBodeSettings;
    if (bodeMWin)
    {
        TView* b = new TRadioButtons(TRect(3, 4, 13, 6),
            new TSIItem("0.01",
            new TSIItem("10", 0)));
        bodeMWin->insert(b);
        bodeMWin->insert(new TLabel(TRect(2, 3, 12, 4),
            "Frequency", b));

        b = new TRadioButtons(TRect(15, 4, 25, 6),
            new TSIItem("-40",
            new TSIItem("30", 0)));
        bodeMWin->insert(b);
        bodeMWin->insert(new TLabel(TRect(14, 3, 24, 4),

```

```

        "Magnitude", b));

b = new TRadioButtons(TRect(27, 4, 37, 6),
new TSItem("-240",
new TSItem("60", 0)));
bodeMWin->insert(b);
bodeMWin->insert(new TLabel(TRect(26, 3, 36, 4),
        "Phase", b));

bodeMWin->insert(new TButton(TRect(15, 8, 25, 10),
        "~C~ancel", cmCancel, bfNormal));
bodeMWin->insert(new TButton(TRect(27, 8, 37, 10),
        "~H~elp", cmHelp, bfNormal));
bodeMWin->insert(new TButton(TRect(3, 8, 13, 10),
        "~O~K", cmOK, bfDefault));

bodeMWin->setData(inSet); // Save dialog data ...
ushort control = deskTop->execView(bodeMWin);
if (control != cmCancel) // ... and read it back ...
// ... when the dialog ...
{
    // ... box is successfully closed.
    bodeMWin->getData(inSet);

    if (inSet->aDialogData == 0)
        iData.wInit = 0.01;
    if (inSet->aDialogData == 1)
        iData.wInit = 10.0;

    if (inSet->bDialogData == 0)
        iData.mInit = -40;
    if (inSet->bDialogData == 1)
        iData.mInit = 30;

    if (inSet->cDialogData == 0)
        iData.pInit = -240;
    if (inSet->cDialogData == 1)
        iData.pInit = 60;

    plotBode();
}
destroy(bodeMWin);
}
}

// FUNCTION: void bodeData(void);
// DESCRIPTION: Gets input and draws Bode plot.
// PARAMETERS: NONE.

```

```

// RETURN VALUE: NONE.
void TCProjApp::bodeData(void)
{
    TDialog* bodeWin = new TDialog(TRect(19, 5, 61, 16),
                                   "Input");
    bodeWin->helpCtx = hcHBodeInput;
    if (bodeWin)
    {
        inDialogData->xDialogData = 0;
        TView* b = new TInputLine(TRect(3, 6, 27, 7), 128);
        bodeWin->insert(b);
        bodeWin->insert(new TLabel(TRect(2, 5, 26, 6),
                                   "Denominator:", b));

        bodeWin->insert(new TButton(TRect(29, 2, 39, 4),
                                   "~O~K", cmOK, bfDefault));
        bodeWin->insert(new TButton(TRect(29, 4, 39, 6),
                                   "~C~ancel", cmCancel, bfNormal));
        bodeWin->insert(new TButton(TRect(29, 6, 39, 8),
                                   "~H~elp", cmHelp, bfNormal));
        b = new TInputLine(TRect(3, 3, 27, 4), 128);
        bodeWin->insert(b);
        bodeWin->insert(new TLabel(TRect(2, 2, 26, 3),
                                   "Numerator:", b));

        b = new TCheckBoxes(TRect(3, 8, 17, 9),
                             new TItem("Settings",0));
        bodeWin->insert(b);

        bodeWin->setData(inDialogData); // Save dialog data.
        ushort control = deskTop->execView(bodeWin);

        if (control != cmCancel) // Read it back ...
                                // ... when the dialog ...
        {
            // ... box is successfully closed.
            bodeWin->getData(inDialogData);
            iData.wInit = 0.01;
            iData.mInit = -40;
            iData.pInit = -240;
            iData.numOrder = vssf(inDialogData->numeratorData,
                                  "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
                                  &iData.num[0], &iData.num[1], &iData.num[2],
                                  &iData.num[3], &iData.num[4], &iData.num[5],
                                  &iData.num[6], &iData.num[7], &iData.num[8],
                                  &iData.num[9]);
            iData.denOrder = vssf(inDialogData->denominatorData,
                                  "%lf %lf %lf %lf %lf %lf %lf %lf %lf",

```

```

        &iData.den[0], &iData.den[1], &iData.den[2],
        &iData.den[3], &iData.den[4], &iData.den[5],
        &iData.den[6], &iData.den[7], &iData.den[8],
        &iData.den[9]);
    if (inDialogData->xDialogData == 1)
        bodeMore();
    else
        plotBode();
}
destroy (bodeWin);
}
}

//      FUNCTION: void plotBode(void)
//      DESCRIPTION: Draws a Bode plot in a graphics window
//              inside the Turbo Vision environment of
//              Control Project.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::plotBode(void)
{
    char errorMsg[MAXSIZE];
    CBode* bp = new CBode;
    suspend();

    if (bp->graphicsStart() == False)
    {
        strcpy(errorMsg, grapherrmsg(graphresult()));
        strcat(errorMsg, ".");
        messageBox(errorMsg, mfError | mfOKButton);
    }
    else
    {
        bp->setgraph();
        bp->draw(iData);
        getch();
        bp->graphicsStop();
    }
    delete bp;
    resume();
}

//      FUNCTION: void nyquistMore(void);
//      DESCRIPTION: Display the Nyquist diagram settings
//              dialog.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.

```



```

void TCProjApp::nyquistMore(void)
{
    TDialog* nyquistMWin = new TDialog(TRect(20, 5, 60, 16),
    "Settings");
    nyquistMWin->helpCtx = hcHNyquistSettings;
    if (nyquistMWin)
    {
        TView* b = new TRadioButtons(TRect(4, 4, 19, 6),
            new TSItem("0.001",
            new TSItem("0.01", 0)));
        nyquistMWin->insert(b);
        nyquistMWin->insert(new TLabel(TRect(3, 3, 19, 4),
            "Frequency A", b));

        b = new TRadioButtons(TRect(21, 4, 36, 6),
            new TSItem("100",
            new TSItem("1000", 0)));
        nyquistMWin->insert(b);
        nyquistMWin->insert(new TLabel(TRect(20, 3, 36, 4),
            "Frequency B", b));

        nyquistMWin->insert(new TButton(TRect(15, 8, 25, 10),
            "~C~ancel", cmCancel, bfNormal));
        nyquistMWin->insert(new TButton(TRect(27, 8, 37, 10),
            "~H~elp", cmHelp, bfNormal));
        nyquistMWin->insert(new TButton(TRect(3, 8, 13, 10),
            "~O~K", cmOK, bfDefault));

        nyquistMWin->setData(inSet); // Save dialog data.
        ushort control = deskTop->execView(nyquistMWin);
        if (control != cmCancel) // Read it back ...
            // ... when the dialog ...
        {
            // ... box is successfully closed.
            nyquistMWin->getData(inSet);

            if (inSet->aDialogData == 0)
                iData.wInit = 0.001;
            if (inSet->aDialogData == 1)
                iData.wInit = 0.01;

            if (inSet->bDialogData == 0)
                iData.wEnd = 100.0;
            if (inSet->bDialogData == 1)
                iData.wEnd = 1000.0;

            plotNyquist();
        }
    }
}

```

```

    destroy(nyquistMWin);
}
}

//      FUNCTION: void nyquistData(void);
//      DESCRIPTION: Gets Input and draws Nyquist diagram.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::nyquistData(void)
{
    TDialog* nyquistWin = new TDialog(TRect(19, 5, 61, 16),
                                     "Input");
    nyquistWin->helpCtx = hcHNyquistInput;
    if (nyquistWin)
    {
        inDialogData->xDialogData = 0;
        TView* b = new TInputLine(TRect(3, 6, 27, 7), 128);
        nyquistWin->insert(b);
        nyquistWin->insert(new TLabel(TRect(2, 5, 26, 6),
                                       "Denominator:", b));

        nyquistWin->insert(new TButton(TRect(29, 2, 39, 4),
                                       "~O~K", cmOK, bfDefault));
        nyquistWin->insert(new TButton(TRect(29, 4, 39, 6),
                                       "~C~ancel", cmCancel, bfNormal));
        nyquistWin->insert(new TButton(TRect(29, 6, 39, 8),
                                       "~H~elp", cmHelp, bfNormal));
        b = new TInputLine(TRect(3, 3, 27, 4), 128);
        nyquistWin->insert(b);
        nyquistWin->insert(new TLabel(TRect(2, 2, 26, 3),
                                       "Numerator:", b));

        b = new TCheckBoxes(TRect(3, 8, 17, 9),
                             new TItem("Settings",0));
        nyquistWin->insert(b);

        nyquistWin->setData(inDialogData);    // Save data.
        ushort control = deskTop->execView(nyquistWin);

        if (control != cmCancel) // Read it back ...
                                // when the dialog ...
        {
            // ... box is successfully closed.
            nyquistWin->getData(inDialogData);
            iData.wInit = 0.01;
            iData.wEnd = 100.0;
            iData.numOrder = vssf(inDialogData->numeratorData,
                                  "%lf %lf %lf %lf %lf %lf %lf %lf %lf",

```

```

        &iData.num[0], &iData.num[1], &iData.num[2],
        &iData.num[3], &iData.num[4], &iData.num[5],
        &iData.num[6], &iData.num[7], &iData.num[8],
        &iData.num[9]);
    iData.denOrder = vssf(inDialogData->denominatorData,
        "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
        &iData.den[0], &iData.den[1], &iData.den[2],
        &iData.den[3], &iData.den[4], &iData.den[5],
        &iData.den[6], &iData.den[7], &iData.den[8],
        &iData.den[9]);
    status = True;
    if (inDialogData->xDialogData == 1)
        nyquistMore();
    else
        plotNyquist();
}
destroy (nyquistWin);
}
}

//      FUNCTION: void plotNyquist(void);
//      DESCRIPTION: Draws a Nyquist diagram in a graphics
//                      window inside the Turbo Vision
//                      environment of Control Project.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::plotNyquist(void)
{
    char errorMsg[MAXSIZE];
    CNyquist* nd = new CNyquist;
    suspend();

    if (nd->graphicsStart() == False)
    {
        strcpy(errorMsg, grapherrmsg(graphresult()));
        strcat(errorMsg, ".");
        messageBox(errorMsg, mfError | mfoKButton);
    }
    else
    {
        nd->setgraph();
        setcolor(WHITE);
        if (status == True)
            outtextxy(1, 1, "Nyquist Diagram");
        else
            outtextxy(1, 1, "Inverse Nyquist Diagram");
        nd->draw(iData);
    }
}

```

```

    getch();
    nd->graphicsStop();
}
delete nd;
resume();
}

//      FUNCTION: void iNyquistData(void);
//      DESCRIPTION: Gets data and draws an inverse Nyquist
//      diagram.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::iNyquistData(void)
{
    TDialog* nyquistWin = new TDialog(TRect(19, 5, 61, 16),
                                     "Input");
    nyquistWin->helpCtx = hcHiNyquistInput;
    if (nyquistWin)
    {
        inDialogData->xDialogData = 0;
        TView* b = new TInputLine(TRect(3, 6, 27, 7), 128);
        nyquistWin->insert(b);
        nyquistWin->insert(new TLabel(TRect(2, 5, 26, 6),
                                       "Denominator:", b));

        nyquistWin->insert(new TButton(TRect(29, 2, 39, 4),
                                       "~O~K", cmOK, bfDefault));
        nyquistWin->insert(new TButton(TRect(29, 4, 39, 6),
                                       "~C~ancel", cmCancel, bfNormal));
        nyquistWin->insert(new TButton(TRect(29, 6, 39, 8),
                                       "~H~elp", cmHelp, bfNormal));
        b = new TInputLine(TRect(3, 3, 27, 4), 128);
        nyquistWin->insert(b);
        nyquistWin->insert(new TLabel(TRect(2, 2, 26, 3),
                                       "Numerator:", b));

        b = new TCheckBoxes(TRect(3, 8, 17, 9),
                             new TItem("Settings",0));
        nyquistWin->insert(b);

        nyquistWin->setData(inDialogData);    // Save data.
        ushort control = deskTop->execView(nyquistWin);

        if (control != cmCancel) // Read it back ...
                                // ... when the dialog ...
        {
            // ... box is successfully closed.
            nyquistWin->getData(inDialogData);
        }
    }
}

```

```

iData.wInit = 0.1;
iData.wEnd = 1000.0;
iData.numOrder = vssf(inDialogData->numeratorData,
    "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
    &iData.den[0], &iData.den[1], &iData.den[2],
    &iData.den[3], &iData.den[4], &iData.den[5],
    &iData.den[6], &iData.den[7], &iData.den[8],
    &iData.den[9]);
iData.denOrder = vssf(inDialogData->denominatorData,
    "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
    &iData.num[0], &iData.num[1], &iData.num[2],
    &iData.num[3], &iData.num[4], &iData.num[5],
    &iData.num[6], &iData.num[7], &iData.num[8],
    &iData.num[9]);
    status = False;
    if (inDialogData->xDialogData == 1)
        nyquistMore();
    else
        plotNyquist();
}
destroy (nyquistWin);
}
}

//      FUNCTION: void nicholsMore(void);
//      DESCRIPTION: Displays the Nichols Chart settings
//      dialog.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::nicholsMore(void)
{
    TDialog* nicholsMWin = new TDialog(TRect(20, 5, 60,16),
        "Settings");
    nicholsMWin->helpCtx = hcHNicholsSettings;
    if (nicholsMWin)
    {
        TView* b = new TRadioButtons(TRect(4, 4, 19 , 6),
            new TSItem("0.001",
            new TSItem("0.1", 0)));
        nicholsMWin->insert(b);
        nicholsMWin->insert(new TLabel(TRect(3, 3, 19, 4),
            "Frequency A", b));

        b = new TRadioButtons(TRect(21, 4, 36, 6),
            new TSItem("10",
            new TSItem("1000", 0)));
        nicholsMWin->insert(b);
    }
}

```

```

nicholsMWin->insert(new TLabel(TRect(20, 3, 36, 4),
    "Frequency B", b));

nicholsMWin->insert(new TButton(TRect(15, 8, 25, 10),
    "~C~ancel", cmCancel, bfNormal));
nicholsMWin->insert(new TButton(TRect(27, 8, 37, 10),
    "~H~elp", cmHelp, bfNormal));
nicholsMWin->insert(new TButton(TRect(3, 8, 13, 10),
    "~O~K", cmOK, bfDefault));

nicholsMWin->setData(inSet);    // Save data.
ushort control = deskTop->execView(nicholsMWin);
if (control != cmCancel) // Read it back ...
    // ... when the dialog ...
{
    // ... box is successfully closed.
    nicholsMWin->getData(inSet);

    if (inSet->aDialogData == 0)
        iData.wInit = 0.001;
    if (inSet->aDialogData == 1)
        iData.wInit = 0.1;

    if (inSet->bDialogData == 0)
        iData.wEnd = 10.0;
    if (inSet->bDialogData == 1)
        iData.wEnd = 1000.0;

    plotNichols();
}
destroy(nicholsMWin);
}
}

//    FUNCTION: void nicholsData(void);
//    DESCRIPTION: Takes data and draws Nichols chart.
//    PARAMETERS: NONE.
//    RETURN VALUE: NONE.
void TCProjApp::nicholsData(void)
{
    TDialog* nicholsWin = new TDialog(TRect(19, 5, 61, 16),
        "Input");
    nicholsWin->helpCtx = hcHNicholsInput;
    if (nicholsWin)
    {
        inDialogData->xDialogData = 0;
        TView* b = new TInputLine(TRect(3, 6, 27, 7), 128);
        nicholsWin->insert(b);
    }
}

```

```

nicholsWin->insert(new TLabel(TRect(2, 5, 26, 6),
                             "Denominator:", b));

nicholsWin->insert(new TButton(TRect(29, 2, 39, 4),
                              "~O~K", cmOK, bfDefault));
nicholsWin->insert(new TButton(TRect(29, 4, 39, 6),
                              "~C~ancel", cmCancel, bfNormal));
nicholsWin->insert(new TButton(TRect(29, 6, 39, 8),
                              "~H~elp", cmHelp, bfNormal));
b = new TInputLine(TRect(3, 3, 27, 4), 128);
nicholsWin->insert(b);
nicholsWin->insert(new TLabel(TRect(2, 2, 26, 3),
                             "Numerator:", b));

b = new TCheckBoxes(TRect(3, 8, 17, 9),
new TItem("Settings",0));
nicholsWin->insert(b);

nicholsWin->setData(inDialogData); // Save data.
ushort control = deskTop->execView(nicholsWin);

if (control != cmCancel) // Read it back ...
                        // ... when the dialog ...
{
    // ... box is successfully closed.
    nicholsWin->getData(inDialogData);
    iData.wInit = 0.1;
    iData.wEnd = 10.0;
    iData.numOrder = vssf(inDialogData->numeratorData,
        "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
        &iData.num[0], &iData.num[1], &iData.num[2],
        &iData.num[3], &iData.num[4], &iData.num[5],
        &iData.num[6], &iData.num[7], &iData.num[8],
        &iData.num[9]);
    iData.denOrder = vssf(inDialogData->denominatorData,
        "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
        &iData.den[0], &iData.den[1], &iData.den[2],
        &iData.den[3], &iData.den[4], &iData.den[5],
        &iData.den[6], &iData.den[7], &iData.den[8],
        &iData.den[9]);

    if (inDialogData->xDialogData == 1)
        nicholsMore();
    else
        plotNichols();
}
destroy (nicholsWin);
}

```

```

}

//      FUNCTION: void plotNichols(void);
//      DESCRIPTION: Draws a Nichols chart in a graphics
//                      window inside the Turbo Vision
//                      environment of Control Project.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::plotNichols(void)
{
    char errorMsg[MAXSIZE];
    CNichols* nc = new CNichols;
    suspend();

    if (nc->graphicsStart() == False)
    {
        strcpy(errorMsg, grapherrmsg(graphresult()));
        strcat(errorMsg, ".");
        messageBox(errorMsg, mfError | mfOKButton);
    }
    else
    {
        nc->setgraph();
        nc->draw(iData);
        getch();
        nc->graphicsStop();
    }
    delete nc;
    resume();
}

//      FUNCTION: void routh(void);
//      DESCRIPTION: Takes data and determines stability
//                      using the Routh Hurwitz method.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::routh(void)
{
    double routh_Table[10][10] = {0.0};
    char s[30];
    int r = 0, c = 0, i, j, last_c;

    for (i = (iData.denOrder+1); i <=9; i++)
    {
        iData.den[i] = 0.0;
    }
    for (i = 0; i <= (iData.denOrder-1); i++)

```



```

{
    routh_Table[r][c] = iData.den[i];
    last_c = c;
    if (r == 1)
    {
        r = 0;
        c++;
    }
    else
        r = 1;
}
for (r = 2; r <= iData.denOrder; r++)
    for (c = 0; c <= last_c; c++)
    {
        if (routh_Table[r - 1][0] == 0)
            routh_Table[r - 1][0] = 0.1;
        routh_Table[r][c] = - (routh_Table[r-2][0] *
                               routh_Table[r-1][c+1] -
                               routh_Table[r-1][0] *
                               routh_Table[r-2][c+1]) /
                               routh_Table[r - 1][0];
        if ((routh_Table[r][c] <= 0.5) &&
            (routh_Table[r][c] >= -0.5))
            routh_Table[r][c] = 0.0;
    }

j = 0;
Boolean stability = True;
for (i = 0; i <= (iData.denOrder-1); i++)
{
    if (((routh_Table[i][j] < 0) &&
        (routh_Table[i + 1][j] > 0)) ||
        ((routh_Table[i][j] > 0) &&
        (routh_Table[i + 1][j] < 0)))
        stability = False;
    if (stability == False)
        strcpy(s, "System is unstable.");
    else
        strcpy(s, "System is stable.");
}

TDialog* routhWin = new TDialog(TRect(18, 2, 62, 20),
                               "Routh Hurwitz");
routhWin->helpCtx = hcHRouthDialog;
int iStyle = 0, jStyle = 0;
if (routhWin)
{

```

```

routhWin->insert(new TStaticText(TRect(4, 13, 24,
                                     14), s));
for (i = 0; i <= (iData.denOrder - 1); i++)
{
    for (j = 0; j <= last_c; j++)
    {
        strcpy(s, "");
        gcvt(routh_Table[i][j], 6, s);
        routhWin->insert(new TStaticText(Trect
                                         (4+iStyle, 2+jStyle,
                                          10+iStyle, 3+jStyle), s));
        iStyle = iStyle + 8;
    }
    jStyle++;
    iStyle = 0;
}
routhWin->insert(new TButton(TRect(7, 15, 17, 17),
                                "~O~K", cmOK, bfDefault));
routhWin->insert(new TButton(TRect(25, 15, 35, 17),
                                "~H~elp", cmHelp, bfNormal));

deskTop->execView(routhWin);
destroy (routhWin);
}
}

//      FUNCTION: void routhData(void);
//      DESCRIPTION: Takes data and determines stability
//      using the Routh Hurwitz method.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::routhData(void)
{
    strcpy(inDialogData->denominatorData, "");
    TDialog* methodWin = new TDialog(TRect(20, 5, 60, 15),
                                     "Input");
    methodWin->helpCtx = hcHRouthInput;
    if (methodWin)
    {
        methodWin->insert(new TButton(TRect(2, 6, 12, 8),
                                        "~O~K", cmOK, bfDefault));
        methodWin->insert(new TButton(TRect(15, 6, 25, 8),
                                        "~C~ancel", cmCancel, bfNormal));
        methodWin->insert(new TButton(TRect(28, 6, 38, 8),
                                        "~H~elp", cmHelp, bfNormal));

        TView* b = new TInputLine(TRect(3, 3, 37, 4), 128);
    }
}

```

```

methodWin->insert(b);
methodWin->insert(new TLabel(TRect(2, 2, 20, 3),
                             "Coefficients:", b));

methodWin->setData(inDialogData); // Save data.
ushort control = deskTop->execView(methodWin);

if (control != cmCancel) // Read it back ...
                        // ... when the dialog ...
{
    // ... box is successfully closed.
    methodWin->getData(inDialogData);
    iData.denOrder = vssf(inDialogData->denominatorData,
                         "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",
                         &iData.den[0], &iData.den[1], &iData.den[2],
                         &iData.den[3], &iData.den[4], &iData.den[5],
                         &iData.den[6], &iData.den[7], &iData.den[8],
                         &iData.den[9]);

    if ((iData.denOrder >= 2) &&
        (iData.denOrder <= 10))
        routh();
    else
        messageBox("System order should be between 1 and 9.",
                   mfError | mfOKButton);
}
destroy (methodWin);
}
strcpy(inDialogData->denominatorData, "");
}

//      FUNCTION: void horner(int shift);
//      DESCRIPTION: Takes data and determines the
//                  coefficients of the shifted polynomial.
//      PARAMETERS: The value of shift.
//      RETURN VALUE: NONE.
void TCProjApp::horner(int shift)
{
    double hc[11] = {0.0}, result;
    char s[10], ord[10];
    int i, j, n, l;

    n = iData.denOrder;
    for (i = 0; i <= (n - 1); i++)
        hc[i+1] = iData.den[i];

    i = 0;

```

```

do
{
    i = i + 1;
    l = (n + 1) - i;
    j = 1;
    do
    {
        j = j + 1;
        result = hc[j] + shift * hc[j - 1];
        hc[j] = result;
    }
    while (j < l);
}
while (i < n - 1);

TDialog* hornerWin = new TDialog(TRect(20, 2, 60, 20),
                                "Horner");
hornerWin->helpCtx = hcHHornerDialog;
int jStyle = 0, temp;
temp = n - 1;
if (hornerWin)
{
    hornerWin->insert(new TStaticText(TRect(5, 2, 17, 3),
                                       "Coefficient:"));

    hornerWin->insert(new TStaticText(TRect(23, 2, 33, 3),
                                       "Order:"));

    for (j = 1; j <= n; j++)
    {
        strcpy(s, "");
        gcvt(hc[j], 6, s);
        strcpy(ord, "");
        itoa(temp, ord, 10);
        temp--;
        hornerWin->insert(new TStaticText(TRect(5,
                                                4+jStyle, 11, 5+jStyle), s));
        hornerWin->insert(new TStaticText(TRect(23,
                                                4+jStyle, 30, 5+jStyle), ord));
        jStyle++;
    }
    hornerWin->insert(new TButton(TRect(7, 15, 17, 17),
                                   "~O~K", cmOK, bfDefault));
    hornerWin->insert(new TButton(TRect(22, 15, 32, 17),
                                   "~H~elp", cmHelp, bfNormal));

    deskTop->execView(hornerWin);
}

```

```

    }
    destroy (hornerWin);
}

//      FUNCTION: void hornerData(void);
//      DESCRIPTION: Takes data and determines coefficients
//      using the Horner's method.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void TCProjApp::hornerData(void)
{
    int shift = -1;
    strcpy(inDialogData->denominatorData, "");
    strcpy(inDialogData->numeratorData, "");
    TDialog* methodWin = new TDialog(TRect(20, 5, 60, 16),
        "Input");
    methodWin->helpCtx = hcHHornerInput;
    if (methodWin)
    {
        methodWin->insert(new TButton(TRect(2, 8, 12, 10),
            "~O~K", cmOK, bfDefault));
        methodWin->insert(new TButton(TRect(15, 8, 25, 10),
            "~C~ancel", cmCancel, bfNormal));
        methodWin->insert(new TButton(TRect(28, 8, 38, 10),
            "~H~elp", cmHelp, bfNormal));

        TView* b = new TInputLine(TRect(3, 3, 37, 4), 128);
        methodWin->insert(b);
        methodWin->insert(new TLabel(TRect(2, 2, 20, 3),
            "Coefficients:", b));

        b = new TInputLine(TRect(3, 6, 20, 7), 128);
        methodWin->insert(b);
        methodWin->insert(new TLabel(TRect(2, 5, 20, 6),
            "Shift to:", b));

        methodWin->setData(inDialogData);    // Save data.
        ushort control = deskTop->execView(methodWin);

        if (control != cmCancel) // Read it back ...
            // ... when the dialog ...
        {
            // ... box is successfully closed.
            methodWin->getData(inDialogData);
            iData.denOrder = vssf(inDialogData->denominatorData,
                "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",
                &iData.den[0], &iData.den[1], &iData.den[2],
                &iData.den[3], &iData.den[4], &iData.den[5],

```

```

        &iData.den[6], &iData.den[7], &iData.den[8],
        &iData.den[9]);

    iData.numOrder = vssf(inDialogData->numeratorData,
        "%d", &shift);

    if ((iData.denOrder >= 2) &&
        (iData.denOrder <= 10))
        horner(shift);
    else
        messageBox("System order should be between 1 and 9.",
            mfError | mfOKButton);
}
destroy (methodWin);
}
strcpy(inDialogData->denominatorData, "");
strcpy(inDialogData->numeratorData, "");
}

//      FUNCTION: void handleEvent(TEvent& event);
//      DESCRIPTION: Defines the program's events.
//      PARAMETERS: A TEvent object "event".
//      RETURN VALUE: NONE.
void TCProjApp::handleEvent(TEvent& event)
{
    TApplication::handleEvent(event);
    if (event.what == evCommand)
    {
        switch(event.message.command)
        {
            case AboutCmd:
                about();
                break;
            case BodeCmd:
                bodeData();
                break;
            case NyquistCmd:
                nyquistData();
                break;
            case NicholsCmd:
                nicholsData();
                break;
            case InvNyquistCmd:
                iNyquistData();
                break;
            case RouthCmd:
                routhData();

```

```

        break;
    case HornerCmd:
        hornerData();
        break;
    default:
        break;
}
clearEvent(event);
}
}

//      FUNCTION: TStatusLine* initStatusLine(TRect r)
//      DESCRIPTION: Draws the status line of the program.
//      PARAMETERS: A TRect object "r".
//      RETURN VALUE: The value of status line Pointer.
TStatusLine* TCProjApp::initStatusLine(TRect r)
{
    r.a.y = r.b.y - 1;
    return new TStatusLine(r,
        *new TStatusDef(0, 0xFFFF) +
        *new TStatusItem("~Alt-X~ Exit", kbAltX, cmQuit) +
        *new TStatusItem("~F1~ Help", kbF1, cmHelp) +
        *new TStatusItem("~F10~ Menu", kbF10, cmMenu));
}

//      FUNCTION: TMenuBar* initMenuBar(TRect r)
//      DESCRIPTION: Draws the system menu.
//      PARAMETERS: TRect object "r".
//      RETURN VALUE: Value of Menu bar pointer.
TMenuBar* TCProjApp::initMenuBar(TRect r)
{
    TSubMenu& subm1 =
        *new TSubMenu("~F~ile", 0, hcHFile) +
        *new TMenuItem("E~x~it", cmQuit, kbAltX, hcHExit,
            "Alt-X");
    TSubMenu& subm2 =
        *new TSubMenu("~M~ethod", 0, hcHMethod) +
        *new TMenuItem("~R~outh Hurwitz", RouthCmd, kbNoKey,
            hcHRouth)+
        *new TMenuItem("Horn~e~r", HornerCmd, kbNoKey,
            hcHHorner);
    TSubMenu& subm3 =
        *new TSubMenu("~G~raph", 0, hcHGraph) +
        *new TMenuItem("B~o~de Plot", BodeCmd, kbNoKey,
            hcHBode)+
        *new TMenuItem("Ny~q~uist Diagram", NyquistCmd,
            kbNoKey, hcHNyquist)+

```

```

    *new TMenuItem("~N~ichols Chart", NicholsCmd,
                  kbNoKey, hcHNichols)+
    *new TMenuItem("Inverse Nyquist Di~a~gram",
                  InvNyquistCmd, kbNoKey, hcHiNyquist);
TSubMenu& subm4 =
    *new TSubMenu("~H~elp", 0, hcHHelp) +
    *new TMenuItem("~A~bout    ", AboutCmd, kbNoKey,
                  hcHAbout);
r.b.y = r.a.y + 1;
return (new TMenuBar(r, subm1 + subm2 + subm3 +
                    subm4));
}

//      FUNCTION: ~TCProjApp(void);
//      DESCRIPTION: TCProjApp destructor.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
TCProjApp::~TCProjApp(void)
{
    delete inDialogData;
    delete inSet;
}

//      FUNCTION: int vssf(char buffer[128], char* fmt,
//                          ...)
//      DESCRIPTION: Extracts variables from a string.
//      PARAMETERS: The string "buffer".
//      RETURN VALUE: Number of variables successfully read.
int vssf(char buffer[128], char* fmt, ...)
{
    va_list argptr;
    int cnt;

    fflush(stdin);
    va_start(argptr, fmt);
    cnt = vsscanf(buffer, fmt, argptr);
    va_end(argptr);

    return(cnt);
}

//      FUNCTION: void freeDriverMem(void);
//      DESCRIPTION: Free memory occupied by driver pointer.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void CGraph::freeDriverMem(void)
{

```



```

if (driverPtr != 0)
    delete (char* ) driverPtr;
driverPtr = NULL;
driverSize = 0;
}

//      FUNCTION: Boolean graphAppLoadDriver(int
//                                     driverNum);
//      DESCRIPTION: Loads the appropriate graphics driver.
//      PARAMETERS: The driver constant "driverNum"
//      RETURN VALUE: True if no error.
Boolean CGraph::graphAppLoadDriver(int driverNum)
{
    char fileName[MAXSIZE];
    int handle, ccode;
    ostringstream ss(fileName, MAXSIZE);

    if (driverNum <= lastDriver)
    {
        if (bgiPath[strlen(bgiPath)-1] != '\\')
            strcat(bgiPath, "\\");
        ss << bgiPath << driverName[driver-1] << ".BGI" << ends;

        ifstream f(ss.str(), ios::in|ios::binary);
        if (f)
        {
            handle = f.rdbuf()->fd();
            driverSize = (unsigned int)filelength(handle);
            f.seekg( 0L, ios::beg);
            if (driverSize < (64 * 1024L - 0xF))
            {
                driverPtr = NULL;
                driverPtr = (DriverPtrFunction) new
                    char[driverSize];
                f.read((char *)driverPtr, ushort(driverSize));
                if (f)
                {
                    ccode = registerfarbgidriver(driverPtr);
                    if (ccode >= 0)
                        return True;
                    else
                        freeDriverMem();
                }
            }
            f.close();
        }
    }
}

```

```

    return False;
}

//      FUNCTION: Boolean CGraph::graphAppInit(
//      int aDriver, int aMode,
//      char* aBGIPath, Boolean loadAtInit);
//      DESCRIPTION: Init BGI. If loadAtInit is true, try to
//      locate and load driver. Returns true if
//      LoadAtInit succeeds or is set to False.
//      Does not "own" bgiPath, but instead is
//      passed a pointer to a string that is
//      allocated elsewhere. Does not de-
//      allocate bgiPath when done.
//      PARAMETERS: Integer "aDriver", "aMode", driver path
//      "aBGIPath" and a Boolean "loadAtInit".
//      RETURN VALUE: True if no error.
Boolean CGraph::graphAppInit(int aDriver, int aMode,
                             char* aBGIPath, Boolean loadAtInit)
{
    if (aBGIPath != 0)
        bgiPath = aBGIPath;
    driver = aDriver;
    mode = aMode;
    freeDriverMem();
    if (loadAtInit == True)
    {
        if (driver == 0)
            detectgraph(&driver, &mode);
        if (driver > 0)
            return graphAppLoadDriver(driver);
        else
            return False;
    }
    return(True);
}

//      FUNCTION: void graphAppDone(void);
//      DESCRIPTION: Close graphics mode and free the
//      occupied memory.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void CGraph::graphAppDone(void)
{
    if (graphActive == True)
        closegraph();
    freeDriverMem();
    graphActive = False;
}

```

```

    bgiPath = emptyString;
    driver = DETECT;
    mode = 0;
}

//      FUNCTION: Boolean graphicsStart(void);
//      DESCRIPTION: Start graphics mode.
//      PARAMETERS: NONE.
//      RETURN VALUE: True if no error.
Boolean CGraph::graphicsStart(void)
{
    if (graphActive == True)
        return(True);

    initgraph(&driver, &mode, bgiPath);

    if (driver < 0)
    {
        graphicsStop();
        return False;
    }
    else
        graphActive = True;
    setgraph();
    return True;
}

//      FUNCTION: Boolean graphicsActive(void);
//      DESCRIPTION: Checks if graphics mode is active.
//      PARAMETERS: NONE.
//      RETURN VALUE: True if graphics mode is active.
Boolean CGraph::graphicsActive(void)
{
    if (graphActive == True)
        return True;
    else
        return False;
}

//      FUNCTION: void graphicsStop(void);
//      DESCRIPTION: Exits graphics mode and returns to Turbo
//                      Vision.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void CGraph::graphicsStop(void)
{
    if (graphActive == True)

```

```

    closegraph();

    graphActive = False;
    TProgram::application->redraw();
}

```

A.2 File: CAPP.CPP

```

#include "capp.h"

//      FUNCTION: double rad(double angle);
//      DESCRIPTION: Returns the radian equivalent of angle.
//      PARAMETERS: The angle "angle".
//      RETURN VALUE: The radian equivalent of angle.
double CGraph::rad(double angle)
{
    return (2.0 * M_PI * angle / 360.0);
}

//      FUNCTION: int round(double x);
//      DESCRIPTION: Converts a float to an integer.
//      PARAMETERS: Real number x.
//      RETURN VALUE: Rounded integer.
int CGraph::round(double x)
{
    return int(ceil(x));
}

//      FUNCTION: int sgn(double x);
//      DESCRIPTION: Returns 1 if "x" is positive, 0 if "x"
//                  is zero and -1 if "x" is negative.
//      PARAMETERS: Real number x.
//      RETURN VALUE: 1:x > 0, 0:x = 0, -1:x < 0.
int CGraph::sgn(double x)
{
    if (x > 0.0)

```

```

    return 1;
else
if (x < 0.0)
    return -1;
else
    return 0;
}

//      FUNCTION: int trunk(double x);
//      DESCRIPTION: Rounds up if "x" is positive, else round
//      down.
//      PARAMETERS: A real number "x".
//      RETURN VALUE: Rounded integer value.
int CGraph::trunk(double x)
{
    if (x >= 0.0)
        return int(ceil(x));
    else
        return int(floor(x));
}

//      FUNCTION: void calculate(InputData iData);
//      DESCRIPTION: Calculates magnitude and phase for a
//      given w.
//      PARAMETERS: Structure iData.
//      RETURN VALUE: NONE.
void CGraph::calculate(InputData iData)
{
    double FA = 57.29578,
           xZ = 0.0,
           yZ = 0.0,
           xP = 0.0,
           yP = 0.0,
           qR = 1.0,
           qI = 0.0,
           qZ;

    for (int i = iData.numOrder; i >= 0; i--)
    {
        xZ = xZ + iData.num[i] * qR; // Real part.
        yZ = yZ + iData.num[i] * qI; // Imaginary part.
        qZ = -(iData.w * qI);
        qI = iData.w * qR;
        qR = qZ;
    }
    qR = 1.0;
    qI = 0.0;
}

```

```

for (i = iData.denOrder; i >= 0; i--)
{
    xP = xP + iData.den[i] * qR; // Real part.
    yP = yP + iData.den[i] * qI; // Imaginaty part.
    qZ = -iData.w * qI;
    qI = iData.w * qR;
    qR = qZ;
}
// Calculate frequency steps.
double wP, aA, dD, aW;

aA = xP * xZ + yZ * yP;
dD = xP * xP + yP * yP;
if (dD == 0.0)
    dD = 1E-10;
aW = aA / dD;
double bW;

bW = (yZ * xP - yP * xZ) / dD;

db = sqrt(aW * aW + bW * bW);
db = 20.0 * log10(db);
double bL;

if (aW == 0.0)
{
    phi = 90.0 * sgn(bW);
    aW == 1E-10;
}
bL = atan(bW / aW);
if (aW > 0.0)
    phi = bL * FA;
else
    if (bW >= 0.0)
        phi = bL * FA + 180.0;
    else
        phi = -180.0 + bL * FA;

aW = (aW * 1000.0 + 0.5) / 1000.0;
bW = (bW * 1000.0 + 0.5) / 1000.0;
db = (db * 1000.0 + 0.5) / 1000.0;
}

//      FUNCTION: void gPlot(int xp, int yp);
//      DESCRIPTION: Draws line from current position to xp,
//                  yp. Because the graphics on IBM PCs has

```

```

//          origin at top LHS, it inverts yp and
//          scales xp and yp to the current screen
//          Xmax and Ymax.
//   PARAMETERS: X and Y positions "xp" and "yp".
//   RETURN VALUE: NONE.
void CGraph::gPlot(int xp, int yp)
{
    // lineto draws a line from the current position (CP)
    // to (x, y).
    lineto(int(xScale * xp), int(yMax - (yScale * yp)));
}

//   FUNCTION: void gMove(int xp, int yp);
//   DESCRIPTION: Moves from current position to xp, yp
//               because the graphics on IBM PCs has
//               origin at top LHS, it inverts yp
//               and scales xp and yp to the current
//               screen Xmax and Ymax.
//   PARAMETERS: X and Y positions "xp" and "yp".
//   RETURN VALUE: NONE.
void CGraph::gMove(int xp, int yp)
{
    // moveto moves the current position (CP) to (x, y).
    moveto(int(xScale * xp), int(yScale * (yMax - yp)));
}

//   FUNCTION: void vertLine(int xStart, int yStart,
//                           int length);
//   DESCRIPTION: Draws a vertical line of given length
//               from "xStart" and "yStart". This avoids
//               a one pixel possible kink in the line
//               which may occur in the gPlot() function.
//               The x/y positions and length are scaled
//               to 640 by 480 pixels and the y position
//               inverted to give the 0, 0 origin at the
//               bottom left of the screen.
//   PARAMETERS: "xStart", "yStart" starting point
//               coordinates and length of line "length".
//   RETURN VALUE: NONE.
void CGraph::vertLine(int xStart, int yStart, int length)
{
    moveto(int(xStart * xScale),
           int(yMax - yScale * yStart));
    lineto(int(xStart * xScale),
           int(yMax - yScale * yStart - length * yScale));
}

```

```

//      FUNCTION: void horizLine(int xStart, int yStart,
//                                int length);
//  DESCRIPTION: Draws a horizontal line of given length
//                from xStart and yStart. This avoids a
//                one pixel possible kink in the
//                line which may occur in the gPlot()
//                function. The x/y positions and length
//                are scaled to 640 by 480 pixels and the
//                y position inverted to give the 0, 0
//                origin at the bottom left of the screen.
//  PARAMETERS: "xStart", "yStart" starting point
//                coordinates and length of line "length".
//  RETURN VALUE: NONE.
void CGraph::horizLine(int xStart, int yStart, int length)
{
    moveto(int(xStart * xScale),
           int(yMax - yScale * yStart));
    lineto(int(xStart * xScale + length * xScale),
           int(yMax - yScale * yStart));
}

//      FUNCTION: void gCircle(int xCentre, int yCentre,
//                              int radius);
//  DESCRIPTION: Draws a circle in 40 segments scaled to
//                the current graphics mode centered on
//                xcent, ycent with given radius on screen
//                based on nominal 640 pixels horizontally
//                and 480 pixels vertically.
//  PARAMETERS: "xCentre", "yCentre" centre point
//                coordinates and the radius "radius".
//  RETURN VALUE: NONE.
void CGraph::gCircle(int xCentre, int yCentre,
                    int radius)
{
    double theta,
           ct, st,
           dx, dy = 0.0,
           xOne;

    gMove(xCentre + radius, yCentre);
    theta = M_PI / 20.0;
    ct = cos(theta);
    st = sin(theta);
    dx = radius * 1.0;
    for (int step = 0; step < 39; step++)
    {
        xOne = dx * ct - dy * st;
    }
}

```



```

    dy = dx * st + dy * ct;
    dx = xOne;
    gPlot(int(xCentre + dx), int(yCentre + dy));
}
gPlot(xCentre + radius, yCentre);
}

//     FUNCTION: void gWriteL(int x, int y, char* text);
//     DESCRIPTION: Writes text string txt to graphics
//                  screen nominal 640 by 480 pixels. Text
//                  is left justified to scaled X, Y
//                  position.
//     PARAMETERS: "x", "y" starting point coordinates,
//                  text string "text".
//     RETURN VALUE: NONE.
void CGraph::gWriteL(int x, int y, char* text)
{
    setttextjustify(LEFT_TEXT, BOTTOM_TEXT);
    setttextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    outtextxy(int(xScale * x), 480 - int(yScale * y),
              text);
}

//     FUNCTION: void gWriteR(int x, int y, char* text);
//     DESCRIPTION: Writes text string txt to graphics
//                  screen nominal 640 by 480 pixels. Text
//                  is right justified to scaled
//                  X, Y position.
//     PARAMETERS: "x", "y" starting point coordinates,
//                  text string "text".
//     RETURN VALUE: NONE.
void CGraph::gWriteR(int x, int y, char* text)
{
    setttextjustify(RIGHT_TEXT, BOTTOM_TEXT);
    setttextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    outtextxy(int(xScale * x), 480 - int(yScale * y),
              text);
}

//     FUNCTION: void gWriteC(int x, int y, char* text);
//     DESCRIPTION: Writes text string txt to graphics
//                  screen nominal 640 by 480 pixels. Text
//                  is centre justified to scaled
//                  X, Y position.
//     PARAMETERS: "x", "y" starting point coordinates,
//                  text string "text".
//     RETURN VALUE: NONE.

```

```

void CGraph::gWriteC(int x, int y, char* text)
{
    setttextjustify(CENTER_TEXT, BOTTOM_TEXT);
    setttextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    outtextxy(int(xScale * x), 480 - int(yScale * y),
              text);
}

//      FUNCTION: void setgraph(void);
//      DESCRIPTION: Finds type of screen on the computer and
//      sets graphics mode. Also sets the scale
//      factors xscale and yscale used in
//      gplot() and gmove().
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void CGraph::setgraph(void)
{
    xMax = getmaxx(); // Find screen dimensions in pixels.
    yMax = getmaxy();
    yScale = yMax / 480.0; // Set scale factor for drawing.
    xScale = xMax / 640.0;
    setbkcolor(BLACK); // Colors for drawing and ...
                       // ... background.

    setcolor(WHITE);
}

//      FUNCTION: void grid(InputData iData);
//      DESCRIPTION: Draws the Bode grid outline. One degree
//      is 0.6 pixels and one db is three
//      pixels.
//      PARAMETERS: Structure iData.
//      RETURN VALUE: NONE.
void CBode::grid(InputData iData)
{
    int dbStep, phiStep, decade,
        xStart = 45, yPhiStart = 30, ydbStart = 250,
        xDecade = 190,
        xPos, yPos, count, angle, decibel;
    char* gString = new char;

    // Draw phase shift grid first.
    for (phiStep = 0; phiStep <= 10; phiStep++)
        // "phiStep" is 30 degrees.
        {
            if ((phiStep == 2) || (phiStep == 8))
                setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
            else

```

```

    setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
    // Solid at zero and -180 degrees.
    yPos = yPhiStart + 18 * phiStep;
    horizLine(xStart, yPos, 3 * xDecade);
    angle = iData.pInit + 30 * phiStep;
    itoa(angle, gString, 10);
    gWriterR(40, yPos - 2, gString);
}
// Now draw decibel grid.
for (dbStep = 0; dbStep < 8; dbStep++)
// db step is 10db.
{
    if (dbStep == 4)
        setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
    else
        setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
// Solid at zero db.
    yPos = ydbStart + 30 * dbStep;
    horizLine(xStart, yPos, 3 * xDecade);
    decibel = iData.mInit + 10 * dbStep;
    itoa(decibel, gString, 10);
    gWriterR(40, yPos - 2, gString);
}
setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
// Draw the vertical log lines.
for (decade = 0; decade <= 2; decade++)
{
    for (count = 1; count <= 9; count++)
    {
        xPos = int(xStart + xDecade * decade + xDecade *
                    log10(count));
        if (count == 1)
            setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
        else
            setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
// Solid at zero db.
        vertLine(xPos, yPhiStart, 180);
        vertLine(xPos, ydbStart, 210);
    }
    setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
    vertLine(xStart + 3 * xDecade, yPhiStart, 180);
    vertLine(xStart + 3 * xDecade, ydbStart, 210);
}
gWriteL(1, 470, "Bode Plot");
gWriteL(1, 445, "dB");
gWriteL(1, 197, "∅");
gWriteL(290, 230, "rad/sec");

```

```

delete gString;
}

//      FUNCTION: void dbxy(double& db, double step,
//                        int decade,
//                        int& xPos, int& yPos);
//      DESCRIPTION: Converts decibel and frequency into X
//                  and Y positions for Bode Plot.
//      PARAMETERS: The magnitude "db", frequency "step" and
//                  the position "xPos", "yPos".
//      RETURN VALUE: NONE.
void CBode::dbxy(double& db, double step, int decade,
                int& xPos, int& yPos, InputData iData)
{
    int blx = 45, bly = 250; // Bottom left of grid.

    if (db > (iData.mInit * 1.0 + 70.0))
        db = (iData.mInit * 1.0 + 70.0);
    if (db < (iData.mInit * 1.0))
        db = (iData.mInit * 1.0);
    xPos = int(blx + 190 * (decade + log10(step)));
    yPos = int(bly + ((-iData.mInit * 1.0) + db) * 3.0);
}

//      FUNCTION: void phixy(double& phi, double step,
//                        int decade,
//                        int& xPos, int& yPos);
//      DESCRIPTION: Converts phase shift and frequency into
//                  X & Y positions for Bode Plot.
//      PARAMETERS: The phase "phi", frequency "step" and
//                  the position "xPos", "yPos".
//      RETURN VALUE: NONE.
void CBode::phixy(double& phi, double step, int decade,
                int& xPos, int& yPos, InputData iData)
{
    int blx = 45, bly = 30; // Bottom left of grid.

    if (phi > (iData.pInit * 1.0 + 300.0))
        phi = (iData.pInit * 1.0 + 300.0);
    if (phi < (iData.pInit * 1.0))
        phi = (iData.pInit * 1.0);
    xPos = int(blx + 190.0 * (decade + log10(step)));
    yPos = int(bly + 0.6 * (phi + (-iData.pInit * 1.0)));
}

//      FUNCTION: void dbplot(double db, double step,
//                        int decade, Boolean first);

```

```

// DESCRIPTION: Draws from current position to db on
// Bode chart.
// PARAMETERS: The magnitude "db", frequency "step" and
// a boolean variable "first". When "first"
// is TRUE bdbplot will plot the first
// point of the graph.
// RETURN VALUE: NONE.
void CNode::bdbplot(double db, double step, int decade,
                    Boolean first, InputData iData)
{
    int x, y;

    dbxy(db, step, decade, x, y, iData);
    if (first == True)
        gMove(x, y);
    else
        gPlot(x, y);
}

// FUNCTION: void phiplot(double phi, double step,
// int decade, Boolean first);
// DESCRIPTION: Draws from current position to phi on
// Bode chart.
// PARAMETERS: The phase "phi", frequency "step" and a
// boolean variable "first". When "first"
// is TRUE bphiplot will plot the first
// point of the graph.
// RETURN VALUE: NONE.
void CNode::phiplot(double phi, double step, int decade, Boolean
first,
                    InputData iData)
{
    int x, y;

    phixy(phi, step, decade, x, y, iData);
    if (first == True)
        gMove(x, y);
    else
        gPlot(x, y);
}

// FUNCTION: void draw(InputData iData);
// DESCRIPTION: Draws a Bode Plot.
// PARAMETERS: NONE.
// RETURN VALUE: NONE.
void CNode::draw(InputData iData)
{

```

```

double step;
char* gString = new char;
int minDec, decade;
Boolean first;

setgraph();
grid(iData);

if (iData.wInit == 0.01)
    iData.wInit = 0.02;
minDec = round(log10(iData.wInit - 0.01));
setcolor(LIGHTRED);
// First draw the gain plot.
first = True;
for (decade = 0; decade <= 2; decade++)
{
    step = 1.0;
    do
    {
        iData.w = step * pow(10.0, decade + minDec);
        calculate(iData);
        dbplot(db, step, decade, first, iData);
        first = False;
        step = step + 0.2;
    }
    while (step <= 10.0);
}
// Draw the phase shift plot.
first = True;
for (decade = 0; decade <= 2; decade++)
{
    step = 1;
    do
    {
        iData.w = step * pow(10.0, decade + minDec);
        calculate(iData);
        phiplot(phi, step, decade, first, iData);
        first = False;
        step = step + 0.2;
    }
    while (step <= 10.0);
}
// Write frequency on the grid.
setcolor(WHITE);
int dec, sign;
for (decade = 0; decade <= 3; decade++)
{

```

```

    freq = pow(10.0, decade + minDec);

    gcvt(freq, 6, gString);
    gWriteR(50 + decade * 190, 230, gString);
}
delete gString;
}

//      FUNCTION: void grid(void);
//      DESCRIPTION: Draws a Nyquist diagram grid.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void CNyquist::grid(void)
{
    int radius;
    double inner = 40.0, outer = 200.0;
    int xp, yp;
    Boolean calc;
    double rad;

    setcolor(WHITE);
    setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
    for (int gain = 1; gain <= 4; gain++)
    {
        radius = 50 * gain;
        gCircle(320, 240, radius);
    }

    setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
    for (int theta = 0; theta <= 35; theta++)
    {
        calc = True;
        // The four if { } below ensure 0, 90, 180 and 270 ...
        // ... lines are square.
        if (theta == 0)
        {
            horizLine(320 + inner, 240, (outer - inner));
            calc = False;
        }
        if (theta == 9)
        {
            vertLine(320, 240 + inner, (outer - inner));
            calc = False;
        }
        if (theta == 18)
        {
            horizLine(320 - outer, 240, (outer - inner));

```

```

    calc = False;
}
if (theta == 27)
{
    vertLine(320, 240 - outer, (outer - inner));
    calc = False;
}
if (calc == True)
{
    rad = 2.0 * M_PI * theta / 36.0;
    xp = int(320.0 + inner * cos(rad));
    yp = int(240.0 + inner * sin(rad));
    gMove(xp, yp);
    xp = int(320.0 + outer * cos(rad));
    yp = int(240.0 + outer * sin(rad));
    gPlot(xp, yp);
}
}
horizLine(320 - inner, 240, 2 * inner);
vertLine(320, 240 - inner, 2 * inner);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
setcolor(LIGHTBLUE);
gCircle(270, 240, 4); // Mark the unity gain, -180. degrees
point.
setcolor(WHITE);
gWriteL(80, 238, " 0ø");
gWriteL(530, 238, "180ø");
gWriteL(312, 444, "90ø");
gWriteL(308, 28, "270ø");
}

//      FUNCTION: void xy(int& x, int& y);
//  DESCRIPTION: Sets the x, y coordinates on the Nyquist
//                grid.
//  PARAMETERS: Coordinates "x", "y".
//  RETURN VALUE: NONE.
void CNyquist::xy(int& x, int& y)
{
    double angle; // phi converted to radians.
    int gainScale = 50; // Pixels for a gain of one.
    int xCentre = 320, yCentre = 240;
    // Scale and centre of diagram.

    angle = (2.0 * M_PI * phi) / 360.0;

    x = round(gainScale * flin * cos(angle)) + xCentre;
    y = round(gainScale * flin * sin(angle)) + yCentre;

```



```

}

//      FUNCTION: void plot(Boolean first);
//      DESCRIPTION: Draws from current position to x, y.
//      PARAMETERS: Boolean value "first".
//      RETURN VALUE: NONE.
void CNyquist::plot(Boolean first)
{
    int x, y;

    xy(x, y); // Convert flin, phi to x, y position.
    if (first == True)
        gMove(x, y);
    else
        gPlot(x, y);
}

//      FUNCTION: void draw(InputData iData);
//      DESCRIPTION: Draws a Nyquist diagram.
//      PARAMETERS: The InputData structure "iData".
//      RETURN VALUE: NONE.
void CNyquist::draw(InputData iData)
{
    double step;
    int minDec, maxDec, decade;
    Boolean first;

    setgraph();
    grid();
    setcolor(LIGHTRED);
    minDec = int(log10(iData.wInit + 0.01));
    maxDec = int(log10(iData.wEnd)) + 1;
    first = True; // Says gmove() rather than gplot().
    for (decade = minDec; decade <= maxDec; decade++)
    {
        step = 1;
        do
        {
            iData.w = step * pow(10.0, decade);
            calculate(iData);
            flin = pow(10.0, db / 20.0);
            plot(first);
            first = False;
            step = step + 0.2;
        }
        while (step <= 9.8);
    }
}

```

```

}

//      FUNCTION: void xy(double ndb, double nphi,
//                        int& x, int& y);
//  DESCRIPTION: Set coordinates x, y of a point on
//                Nichols chart.
//  PARAMETERS: Magnitude "ndb", phase "nphi",
//                coordinates "xp", "yp".
//  RETURN VALUE: NONE.
void CNichols::xy(double ndb, double nphi, int& xp,
                 int& yp)
{
    int blx, bly;
    double phiInc, dbInc;

    if (ndb > 40.0)
        ndb = 40.0;
    if (ndb < -30.0)
        ndb = -30.0;
    if (nphi > 0.0)
        nphi = 0.0;
    if (nphi < -360.0)
        nphi = -360.0;
    blx = 50; // Bottom left of grid.
    bly = 20;
    phiInc = 1.5; // x pixels per degree.
    dbInc = 6.0; // y pixels per db.
    xp = int(blx + (360.0 + nphi) * phiInc);
    yp = int(bly + (30.0 + ndb) * dbInc);
}

//      FUNCTION: void plot(double ndb, double nphi,
//                          Boolean first);
//  DESCRIPTION: Draws from current position to db / phi //
//                on Nichols chart.
//  PARAMETERS: Magnitude "ndb", phase "nphi", boolean //
//                "first".
//  RETURN VALUE: NONE.
void CNichols::plot(double ndb, double nphi,
                   Boolean first)
{
    int x, y;

    xy(ndb, nphi, x, y);
    if (first)
        gMove(x, y);
    else

```

```

    gPlot(x, y);
}

//      FUNCTION: void threedb(void);
//      DESCRIPTION: Draws the +3db line on the Nichols
//      chart.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void CNichols::threedb(void)
{
    int toggle, x, y;

    setlinestyle(DASHED_LINE, 0, NORM_WIDTH);
    toggle = -1;
    do // Simple table of points on 3dB curve.
    {
        xy(11.0, -180.0, x, y);
        gMove(x, y);
        plot(10.0, -180.0 + toggle * 30.0, False);
        plot(8.0, -180.0 + toggle * 40, False);
        plot(3.0, -180.0 + toggle * 45, False);
        plot(0.0, -180.0 + toggle * 42, False);
        plot(-2.0, -180.0 + toggle * 35, False);
        plot(-4.0, -180.0 + toggle * 20, False);
        plot(-4.5, -180.0, False);
        toggle = toggle + 2;
    }
    while (toggle <= 2);
    setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
    gWriter(260, 250, "+3db");
}

//      FUNCTION: void zerodb(void);
//      DESCRIPTION: Draws the zero db line on the Nichols
//      chart.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void CNichols::zerodb(void)
{
    int x, y, angle;
    double theta, nphi, ndb;

    xy(40.0, -90.0, x, y);
    gMove(x, y);
    setlinestyle(DASHED_LINE, 0, NORM_WIDTH);
    for (angle = 1; angle <= 35; angle++)
    {

```

```

    theta = rad(10.0 * angle);
    nphi = -(180.0 + 10.0 * angle) / 2.0;
    ndb = 20.0 * log10(1.0 / (2.0 * sin(theta / 2.0)));
    xy(ndb, nphi, x, y);
    gPlot(x, y);
}
xy(40.0, -270.0, x, y);
gPlot(x, y);
setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
gWriteR(180.0, 420.0, "0db");
}

//      FUNCTION: void grid(void);
//      DESCRIPTION: Draws a nichols grid outline scaling 1.5 //
pixels per degree in x direction and 6 //                pixels
per db in y direction.
//      PARAMETERS: NONE.
//      RETURN VALUE: NONE.
void CNichols::grid(void)
{
    int dbStep, phiStep, angle, decibel,
        xStart = 50, yStart = 20, xPos, yPos;
    char gString[5];

    for (phiStep = 0; phiStep <= 6; phiStep++)
    // "phiStep" represents 60 degrees.
    {
        if (phiStep == 3)
            setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
        else
            setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        xPos = xStart + 90 * phiStep;
        vertLine(xPos, yStart, 420);
        angle = -360 + 60 * phiStep;
        itoa(angle, gString, 10);
        gWriteC(xPos, 450, gString);
    }
    for (dbStep = 0; dbStep <= 7; dbStep++)
    // "dbStep" represents 10db.
    {
        if (dbStep == 3)
            setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
        else
            setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        yPos = yStart + 60 * dbStep;
        horizLine(xStart, yPos, 540);
        decibel = -30 + 10 * dbStep;
    }
}

```

```

        itoa(decibel, gString, 10);
        gWriter(45, yPos - 3, gString);
    }
    gWriteL(310, 460, "phi");
    gWriteL(10, 220, "db");
    gWriteL(5, 460, "Nichols Chart");
    zerodb();
    threedb();
    setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
}

//      FUNCTION: void draw(InputData id);
//      DESCRIPTION: Draws a Nichols chart.
//      PARAMETERS: The Input data "iData".
//      RETURN VALUE: NONE.
void CNichols::draw(InputData iData)
{
    double w, step;
    int minDec, maxDec, decade, x, y, xOffset, yOffset;
    Boolean first;
    char* gString = new char;

    setgraph();
    grid();
    minDec = trunk(log10(iData.wInit + 0.01));
    maxDec = trunk(log10(iData.wEnd));

    first = True; // Says gmove rather than gplot.
    for (decade = minDec; decade <= maxDec; decade++)
    {
        step = 1;
        do
        {
            iData.w = step * pow(10.0, decade);
            // Frequency calculated ...
            // ... from decade and step.
            calculate(iData);
            setcolor(LIGHTRED);
            plot(db, phi, first);
            if (step == 1)
            { // Mark it on the chart.
                xy(db, phi, x, y);
                setcolor(LIGHTBLUE);
                gCircle(x, y, 4);

                setcolor(WHITE);
                gcvt(iData.w, 6, gString);
            }
        }
    }
}

```

```

yOffset = 5; // Offsets place frequency ...
xOffset = 50;
// ... text according to position ...

if (db > 0.0) // ... on the chart, and are ...
    yOffset = -10; // ... changed below if needed.
if (phi < -180.0)
    xOffset = -10;
if (decade == minDec)
    gWriter(x + xOffset, y + yOffset, gString);
gMove(x, y);
}
first = False;
step = step + 0.2;
}
while ((step <= 9.8) && (iData.w < iData.wEnd));
}
delete gString;
}

```

A.3 File: CAPP.H

```

#define Uses_TProgram
#define Uses_TRect
#define Uses_TMenuBar
#define Uses_TSubMenu
#define Uses_TMenuItem
#define Uses_TKeys
#define Uses_fpstream
#define Uses_TView
#define Uses_TPalette
#define Uses_MsgBox
#define Uses_TApplication
#define Uses_TDeskTop
#define Uses_TStaticText
#define Uses_TDialog

```

```
#define Uses_TEventQueue
#define Uses_TEvent
#define Uses_TButton
#define Uses_TStatusLine
#define Uses_TStatusItem
#define Uses_TStatusDef
#define Uses_TScroller
#define Uses_TScrollBar
#define Uses_TSItem
#define Uses_TCheckBoxes
#define Uses_TRadioButton
#define Uses_TLabel
#define Uses_TInputLine
#include "cph.h"
#include <help.h>
#include <stdio.h>
#include <stdarg.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
#include <fstream.h>
#include <io.h>
#include <tv.h>
#include <fstream.h>
#include <strstrea.h>
#include <string.h>
#include <stdlib.h>

const AboutCmd      = 100,
      BodeCmd       = 101,
      NyquistCmd    = 102,
      NicholsCmd    = 103,
      InvNyquistCmd = 104,
      RouthCmd      = 105,
      HornerCmd     = 106;

struct InputData
{
    int numOrder,
        denOrder,
        mInit,
        pInit;
    double num[10],
           den[10],
           wInit,
           w,
           wEnd;
```

```

};

class CGraph
{
public:
    double xScale, yScale;
    int xMax, yMax;

    double rad(double x);
    int round(double x);
    int sgn(double x);
    int trunk(double x);
    double db, phi, freq, flin;

    void calculate(InputData id);
    void gPlot(int xp, int yp);
    void gMove(int xp, int yp);
    void vertLine(int xStart, int yStart, int length);
    void horizLine(int xStart, int yStart, int length);
    void gCircle(int xCentre, int yCentre, int radius);
    void gWriteL(int x, int y, char* text);
    void gWriteR(int x, int y, char* text);
    void gWriteC(int x, int y, char* text);
    void setgraph(void);
    void freeDriverMem(void);
    Boolean graphAppLoadDriver(int driverNum);
    Boolean graphicsActive(void);
    Boolean graphAppInit(int, int, char* , Boolean);
    void graphAppDone(void);
    Boolean graphicsStart(void);
    void graphicsStop(void);
};

class CBode: public CGraph
{
public:
    void dbxy(double& db, double step, int decade, int&
              xp, int& yp, InputData iData);
    void phixy(double& phi, double step, int decade, int&
              xp, int& yp, InputData iData);
    void dbplot(double db, double step, int decade,
                Boolean first, InputData iData);
    void phiplot(double phi, double step, int decade,
                 Boolean first, InputData iData);
    void grid(InputData);
};

```



```

        void draw(InputData id);
};

class CNyquist: public CGraph
{
public:
    void grid(void);
    void xy(int& x, int& y);
    void plot(Boolean first);
    void draw(InputData iData);
};

class CNichols: public CGraph
{
public:
    void xy(double ndb, double nphi, int& xp, int& yp);
    void plot(double ndb, double nphi, Boolean first);
    void threedb(void);
    void zerodb(void);
    void grid(void);
    void draw(InputData iData);
};

```

A.4 File: CPH.TXT

.topic HCP=0

CONTROL PROJECT

Welcome, {:HEmpty}{CONTROL PROJECT:HControlProject} is a program that allows you to get information and graphs from any given transfer function.

Help Contents

Menu Item:

```

{File:HFile}
{Exit:HExit}

```

```

{Method:HMethod}
{Routh Hurwitz:HRouth}
{Horner:HHorner}
{Graph:HGraph}
{Bode Plot:HBode}
{Nyquist Diagram:HNyquist}
{Nichols Chart:HNichols}
{Nyquist Diagram:HiNyquist}
{Help:HHelp}
{About:HAbout}

```

.topic HControlProject

CONTROL PROJECT Version 1.0

CONTROL PROJECT V1.0, Copyright (C)1997, developed by Christos Bohoris. This project was developed as a final year project for a BENG. in Electrical and Electronic Engineering at the University of Wales, College of Cardiff during the academic year 1996-1997.

The menu bar can be accessed by pressing F10. All menu items are accessed by pressing Alt-Z, where Z is the first letter of the menu. For example, the "Help" menu is pulled down by Alt-H.

Press ESC to put this help screen away.

.topic HExit

File|Exit (Alt-X)

The Exit command terminates this program.

.topic HFile

File (Alt-F)

The File menu appears on the far left of the menu bar.

The File menu contains,

```

{:HEmpty}
{Exit:HExit}

```

.topic HRouth

Method|Routh Hurwitz

Choosing the Routh Hurwitz command from the Method menu, allows you to obtain a Routh Hurwitz table and determine the stability of a system.

See also,

```
{:HEmpty}
  {Input-Routh Hurwitz:HRouthInput}
  {Routh Hurwitz:HRouthDialog}
```

.topic HHorner

Method | Horner

Choosing the Horner command from the Method menu, allows you to obtain a linear translation of the imaginary axis relative to the system's poles and zeros using the Horner's method.

See also,

```
{:HEmpty}
  {Input-Horner:HHornerInput}
  {Horner:HHornerDialog}
```

.topic HMethod

Method (Alt-M)

The Method menu offers a choice of control methods.

The Method menu contains,

```
{:HEmpty}
  {Routh Hurwitz:HRouth}
  {Horner:HHorner}
```

.topic HBode

Graph | Bode Plot

Choosing the Bode Plot command from the Graph menu, allows you to obtain the Bode Plot of a transfer function.

See also,

```
{:HEmpty}
  {Input-Bode Plot:HBodeInput}
  {Settings-Bode Plot:HBodeSettings}
```

.topic HNyquist

Graph | Nyquist Diagram

Choosing the Nyquist Diagram command from the Graph menu, allows you to obtain the Nyquist Diagram of a transfer function.

See also,

```
{:HEmpty}
  {Input-Nyquist Diagram:HNyquistInput}
  {Settings-Nyquist Diagram:HNyquistSettings}
```

.topic HNichols

Graph|Nichols Chart █
████████████████████

Choosing the Nichols Chart command from the Graph menu, allows you to obtain the Nichols Chart of a transfer function.

See also,

```
{:HEmpty}
  {Input-Nichols Chart:HNicholsInput}
  {Settings-Nichols Chart:HNicholsSettings}
```

.topic HiNyquist

Graph|Inverse Nyquist Diagram █
████████████████████

Choosing the Inverse Nyquist Diagram command from the Graph menu, allows you to obtain the Inverse Nyquist Diagram of a transfer function.

See also,

```
{:HEmpty}
  {Input-Inverse Nyquist Diagram:HiNyquistInput}
  {Settings-Inverse Nyquist Diagram:HNyquistSettings}
```

.topic HGraph

Graph █ (Alt-G)
████████

The Graph menu offers choices for plotting various graphs of a transfer function.

The Graph menu contains,

```
{:HEmpty}
  {Bode Plot:HBode}
  {Nyquist Diagram:HNyquist}
  {Nichols Chart:HNichols}
  {Inverse Nyquist Diagram:HiNyquist}
```

.topic HAbout

Help|About █
████████████████████

When you choose the About command from the Help menu, a dialog box appears, showing version information.

```
.topic HHelp
Help █ (Alt-H)
█
```

The Help menu offers a choice for obtaining information on CONTROL PROJECT.

```
The Help menu contains,
{:HEmpty}
{About:HAbout}
```

```
.topic HBodeInput
Input-Bode Plot █
█
```

The Input dialog box allows you to enter a transfer function by typing its numerator and denominator coefficients. For example for the transfer function T.F. = $(s + 2) / (3s^2 + 5)$ you should enter,

```
Numerator:
1 2
Denominator:
3 0 5
```

The numerator and denominator must be in the form:

$$C_n s^n + \dots + C_2 s^2 + C_1 s + C_0$$

The system defaults define a starting, frequency of 0.01 rad/sec, magnitude of -40 db and phase of -240° . The maximum allowed order of numerator or denominator is 9.

Checking on the Settings box, allows these defaults to be changed in a second dialog that appears after you click "OK".

```
.topic HBodeSettings
Settings-Bode Plot █
█
```

This dialog allows you to set the starting values of frequency, magnitude and phase of a Bode Plot.

```
.topic HNyquistInput
Input-Nyquist Diagram █
█
```

The Input dialog box allows you to enter a transfer function by typing its numerator and denominator coefficients. For example for the transfer function T.F. = $(s + 2) / (3s^2 + 5)$ you should enter,

```
Numerator:
```

```

1 2
Denominator:
3 0 5

```

The numerator and denominator must be in the form:

$$C_n s^n + \dots + C_2 s^2 + C_1 s + C_0$$

The system defaults define a starting, frequency of 0.1 rad/sec, and a finishing frequency of 10 rad/sec. The maximum allowed order of numerator or denominator is 9.

Checking on the Settings box, allows these defaults to be changed in a second dialog that appears after you click "OK".

```

.topic HNyquistSettings
Settings-(Inverse) Nyquist Diagram

```

This dialog allows you to set the value of the starting frequency A and the finishing frequency B of the diagram.

```

.topic HNicholsInput
Input-Nichols Chart

```

The Input dialog box allows you to enter a transfer function by typing its numerator and denominator coefficients. For example for the transfer function T.F. = $(s + 2) / (3s^2 + 5)$ you should enter,

```

Numerator:
1 2
Denominator:
3 0 5

```

The numerator and denominator must be in the form:

$$C_n s^n + \dots + C_2 s^2 + C_1 s + C_0$$

The system defaults define a starting, frequency of 0.1 rad/sec, and a finishing frequency of 10 rad/sec. The maximum allowed order of numerator or denominator is 9.

Checking on the Settings box, allows these defaults to be changed in a second dialog that appears after you click "OK".

```

.topic HNicholsSettings
Settings-Nichols Chart

```

This dialog allows you to set the value of the starting frequency A and the finishing frequency B of the Nichols chart.

```
.topic HiNyquistInput
Input-Inverse Nyquist Diagram
```

The Input dialog box allows you to enter a transfer function by typing its numerator and denominator coefficients. For example for the transfer function T.F. = $(s + 2) / (3s^2 + 5)$ you should enter,

```
Numerator:
1 2
Denominator:
3 0 5
```

The numerator and denominator must be in the form:

$$C_n s^n + \dots + C_2 s^2 + C_1 s + C_0$$

The system defaults define a starting, frequency of 0.1 rad/sec, and a finishing frequency of 10 rad/sec. The maximum allowed order of numerator or denominator is 9.

Checking on the Settings box, allows these defaults to be changed in a second dialog that appears after you click "OK".

```
.topic HRouthDialog
Routh Hurwitz
```

This dialog displays the Routh Hurwitz table of the system and also the decision of the program on the stability of the system.

```
.topic HRouthInput
Input-Routh Hurwitz
```

This dialog allows you to enter the coefficients of a characteristic equation.

For example for the characteristic equation, C.E. = $2s^2 + 16$ you should enter,



```
Coefficients:
2 0 16
```

The coefficients must be in the form:

$$C_n s^n + \dots + C_2 s^2 + C_1 s + C_0$$

The minimum allowed order of the characteristic equation is 1 and the maximum is 9.

.topic HHornerInput

Input-Horner 


This dialog allows you to enter the coefficients of a polynomial. For example for the polynomial, $P = 21s^2 + 30$ you should enter,

Coefficients:
 21 0 30



The coefficients must be in the form:

$$C_n s^n + \dots + C_2 s^2 + C_1 s + C_0$$

The minimum allowed order of the polynomial is 1 and the maximum is 9.

This dialog also allows you to set the value where the imaginary axis will be shifted.

.topic HHornerDialog

Horner 


This dialog displays the coefficients of the shifted polynomial as determined by the Horner method.

.topic HEmpty

EMPTY