APPENDIX H

Control System Computational Aids

To Accompany

Control Systems Engineering
3rd Edition

By

Norman S. Nise

John Wiley & Sons

ISBN 0-471-36601-3

John Wiley & Sons, Inc.
605 Third Avenue
New York, NY 10158
USA

A P P E N D I X H

Control System Computational Aids

H.1 Step Response of a System Represented in State Space

In this section we will discuss how to obtain the step response of systems represented in state space. We will begin by discussing how state equations can be used to program a digital computer and progress to a computer program that you can use to perform step-response simulations.

Using State Equations for Computer Simulations

One advantage of state equations is the ability to use this representation to simulate control systems on the digital computer. This section is devoted to demonstrating this concept. Consider the system represented in state-space by Eqs. (H.1).

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \tag{H.1a}$$

$$y(t) = \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{H.1b}$$

$$\begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \tag{H.1c}$$

This system is represented in phase-variable form and has a unit step input, $u(t)$. We are about to formulate a solution for the system output, $y(t)$, by numerically integrating the differential equation on the digital computer. We will use a method called *Euler's approximation* , where the area to be integrated is approximated as a rectangle. The solution obtained on the computer is an actual time waveform plot rather than the closed-form expression we arrived at via the Laplace transform.

Writing the state equations explicitly, we have

$$\frac{dx_1}{dt} = x_2 \tag{H.2a}$$

$$\frac{dx_2}{dt} = -2x_1 - 3x_2 + 1 \tag{H.2b}$$

If we approximate dx by Δx and dt by Δt, and multiply through by Δt

$$\Delta x_1 = x_2 \Delta t \tag{H.3a}$$

$$\Delta x_2 = (-2x_1 - 3x_2 + 1)\Delta t \tag{H.3b}$$

We can say that the value at the next interval for either state variable is approximately the current value plus the change. Thus,

$$x_1(t + \Delta t) = x_1(t) + \Delta x_1 \tag{H.4a}$$

$$x_2(t + \Delta t) = x_2(t) + \Delta x_2 \tag{H.4b}$$

Finally, from the output equation (H.1b), $y(t)$ at the next time interval, $y(t + \Delta t)$, is

$$y(t + \Delta t) = 2x_1(t + \Delta t) + 3x_2(t + \Delta t) \tag{H.5}$$

Let us see how this would work on the digital computer. From the problem statement, x_1 and x_2 begin at $t = 0$ with values 1 and –2, respectively. If we assume a Δt interval of 0.1 second[1], the change in x_1 and x_2 from 0 to 0.1 second is found from Eqs. (H.3) to be,

$$\Delta x_1 = x_2(0)\Delta t = -0.2 \tag{H.6a}$$

$$\Delta x_2 = [-2x_1(0) - 3x_2(0) + 1]\Delta t = 0.5 \tag{H.6b}$$

from which the state variables at $t = 0.1$ second are found from Eqs. (H.4) to be

$$x_1(0.1) = x_1(0) + \Delta x_1 = 0.8 \tag{H.7a}$$

$$x_2(0.1) = x_2(0) + \Delta x_2 = -1.5 \tag{H.7b}$$

Finally, the output at $t = 0.1$ second is calculated from Eq. (H.5) to be,

$$y(0.1) = 2x_1(0.1) + 3x_2(0.1) = -2.9 \tag{H.8}$$

[1] Δt is selected to be small and, initially, at least an order of magnitude less than the system's time constants. In order to determine, empirically, how small Δt should be, the value of Δt can be successively reduced after each response has been calculated by the computer until the difference between the current response and the previous response is negligible. If Δt is too large, then error results from inaccurately representing the area under the state variable curve. If Δt is too small, then round-off error will accumulate during the computation because of the numerous calculations.

The values of the state variables at $t = 0.1$ second are used to calculate the values of the state variables and the output at the next interval of time, $t = 0.2$ second. Once again the changes in x_1 and x_2 are,

$$\Delta x_1 = x_2(0.1)\Delta t = -0.15 \tag{H.9a}$$

$$\Delta x_2 = [-2x_1(0.1) - 3x_2(0.1) + 1]\Delta t = 0.39 \tag{H.9b}$$

from which the state variables at $t = 0.2$ second are found to be

$$x_1(0.2) = x_1(0.1) + \Delta x_1 = 0.65 \tag{H.10a}$$

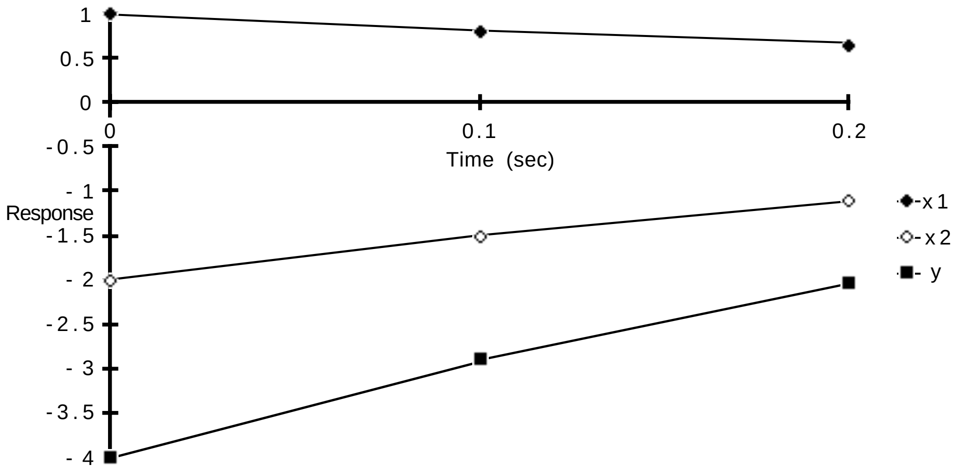
$$x_2(0.2) = x_2(0.1) + \Delta x_2 = -1.11 \tag{H.10b}$$

Finally, the output at $t = 0.2$ second is calculated as,

$$y(0.2) = 2x_1(0.2) + 3x_2(0.2) = -2.03 \tag{H.11}$$

The results are summarized in Figure H.1. Continuing in like manner until $t = t_f$, the maximum desired time, the response for $0 \le t \le t_f$ can be obtained.

Figure H.1 State variables and output for the system of Eqs. H.1



Computer Program for Step Response

In this subsection we will design a computer program that simulates a system's step response using state equations. The code was developed using Visual Basic® Version 6 and converted to a stand-alone application that runs on a PC[2]. The resulting application, recommended for readers who do not have access to MATLAB®, can be found in the Appendix H folder on this CD-ROM[3]. To run the setup program, open the folder labeled Step Response inside the Appendix H folder and double-click on setup icon. For directions on running the program see the README file inside the Appendix H folder. Let us now summarize the design of the step response software.

First, we enumerate the software requirements as follows:

1. The user will input (1) system order, (2) components of the system, input, and output matrix.
2. The user will input the initial conditions.
3. The user will input the following plot parameters: (1) iteration interval, (2) plot interval, and (3) maximum time.
4. The program will plot the step response as well as listing the response data.
5. The program will replot the step response after allowing the user to change the initial conditions as well as the plot parameters without reentering the system.

The program plots the step response of a system represented in state space and permits the user to choose an iteration interval. A helpful technique of finding the iteration interval is to run the program with successively diminishing iteration intervals until reaching an iteration interval below which there is no appreciable change in the results.

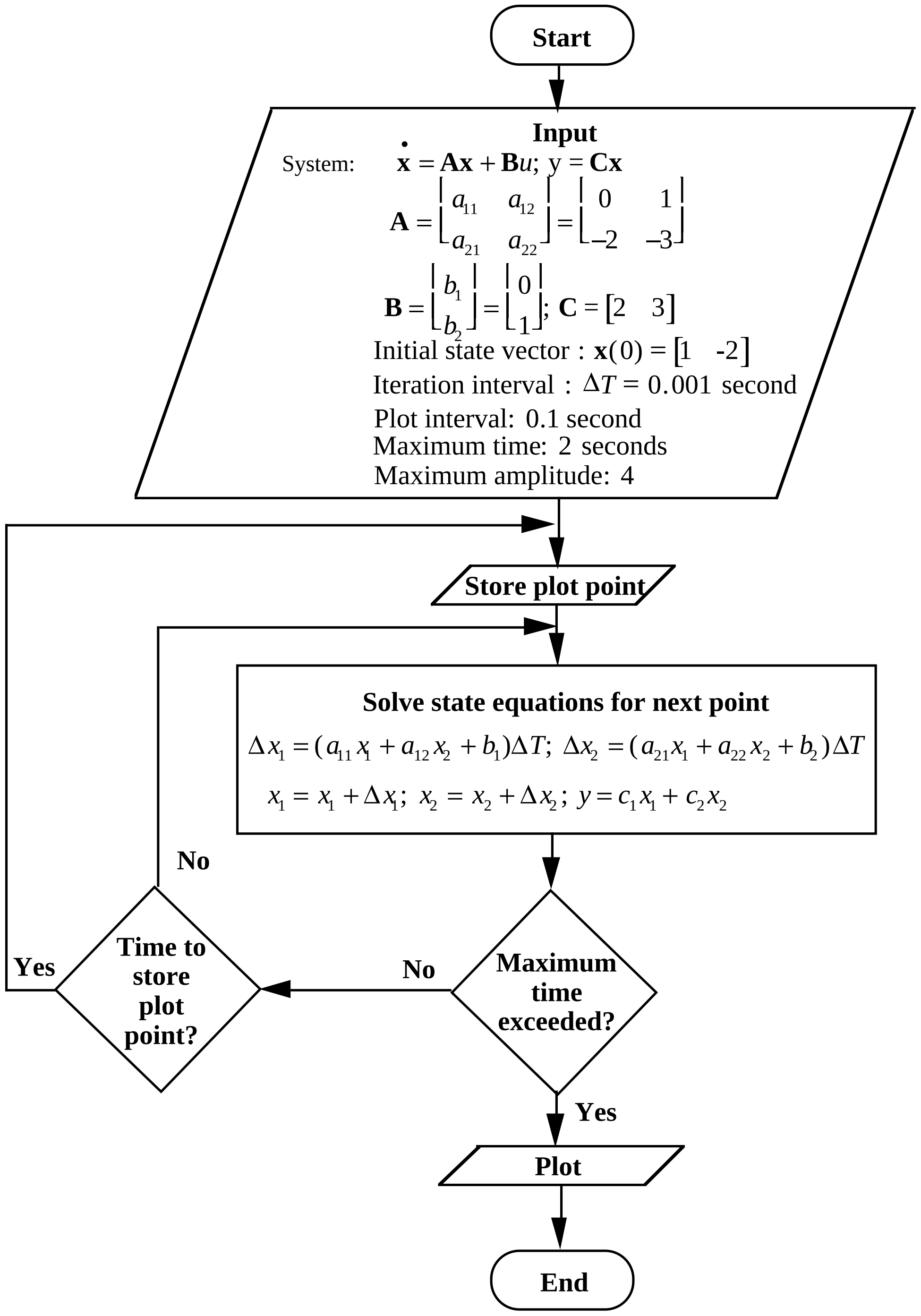
Another parameter the user can select is the print interval which allows the user to print at a larger time interval than the iteration interval.

The execution time of the program is also an input parameter. The user should choose a time for which the output has already reached a steady-state value.

A simplified flow-chart for the program is shown in Figure H.2 and uses the system of Eqs. (H.1).

[2] Visual Basic is a registered trademark of Microsoft Corporation.
[3] MATLAB is a registered trademark of The MathWorks, Inc.

Figure H.2 Flow-chart for Step Response program


Start
Input
System: $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u$; $y = \mathbf{C}\mathbf{x}$
$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$
$\mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$; $\mathbf{C} = [2 \quad 3]$
Initial state vector : $\mathbf{x}(0) = [1 \quad -2]$
Iteration interval : $\Delta T = 0.001$ second
Plot interval: 0.1 second
Maximum time: 2 seconds
Maximum amplitude: 4
Store plot point
Solve state equations for next point
$\Delta x_1 = (a_{11}x_1 + a_{12}x_2 + b_1)\Delta T$; $\Delta x_2 = (a_{21}x_1 + a_{22}x_2 + b_2)\Delta T$
$x_1 = x_1 + \Delta x_1$; $x_2 = x_2 + \Delta x_2$; $y = c_1x_1 + c_2x_2$
No
Yes
Time to store plot point?
No
Maximum time exceeded?
Yes
Plot
End

Code Module

We now present a sample implementation of the flow-chart of Figure H.2. The routine can run independently, as part of a Visual Basic Code Module, or tailored to another programming language or other machines, such as hand-held calculators.

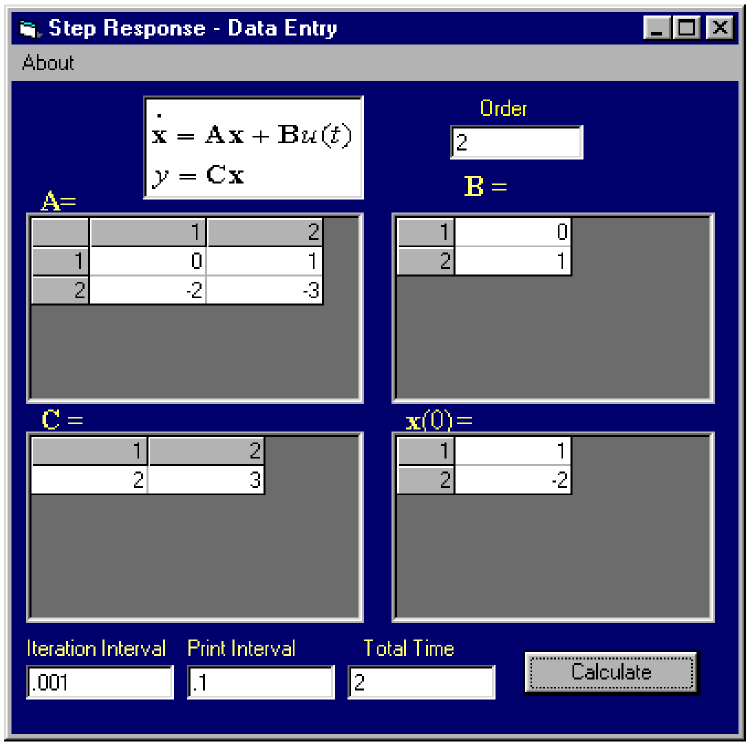
The routine can obtain its input variable values through the Visual Basic GUI interface, as presented in the sample run below, or through another program written to pass this code the input variables. The same is true of the output variables. In the sample run below, output variables are passed to the Visual Basic GUI interface for display, but could just as well be passed to another program.

We now list the sample subroutine, which we call CalcStateSpace :

```
'**********************Input Variables**********************
'Although the following arrays are being dimensioned for a
'100th order system, only a portion of the array, defined by
'the sys_order variable, is used.

Public X(100) As Single           'X vector input.
Public A(100, 100) As Single      'A matrix Input.
Public B(100) As Single           'B matrix Input.
Public C(100) As Single           'C vector Input

Public PRNT_Int                   'Print interval input.
Public sys_order                  'System order input.
Public DELTAT                     'Delta time input.
Public MAXTIME                    'Total run-time input.

'*********************Output Variables*********************
Public DELTAX(1000) As Single     'Array holding the time for
                                  'each point calculated.
Public Y(1000) As Single          'Array holding the output
                                  'response value for each
                                  'point calculated.

'****************Subroutine CalcStateSpace*****************
Public Sub CalcStateSpace()
On Error GoTo errorHandle

'***************Store initial value for plot***************
      Let cx = 0
      For i = 1 To sys_order
           cx = cx + C(i - 1) * X(i - 1)
      Next i
      Y(0) = cx

'*******************Start plot loop************************
    For K = 1 To CInt(MAXTIME / PRNT_Int) Step 1
                                  'Index for Printing interval

'****************Start iteration loop**********************
        For n = 1 To CInt(PRNT_Int / DELTAT) Step 1
                                  'Index for iteration interval
            For i = 1 To sys_order
                Let ax = 0
                For j = 1 To sys_order
                   ax = ax + A(i - 1, j - 1) * X(j - 1)
                Next j
                DELTAX(i - 1) = (ax + B(i - 1)) * DELTAT
                                   ' Calculate delta X1
            Next i
            For i = 1 To sys_order
                   X(i - 1) = X(i - 1) + DELTAX(i - 1)
                                   'Calculate next x
            Next i
            Let cx = 0
            For i = 1 To sys_order
               cx = cx + C(i - 1) * X(i - 1)
            Next i
        Next n

'*******************End iteration loop*********************
        Y(K) = cx

    Next K

'**********************End plot loop***********************
    Exit Sub
errorHandle:
        message = "System Error: " + Err.Description
        MsgBox (message)
        On Error GoTo 0
End Sub
```

As an example, data entry and results for the code shown above are via a graphical user interface (GUI) developed in Visual Basic 6 and produced by the stand-alone application enclosed in this folder. Figure H.3 shows the GUI interface for data entry using the system of Eqs. H.1 as an example. Figure H.4 shows the output window for the example.

Figure H.3 Step Response Program: GUI interface for data entry;



$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u(t)$$
$$y = \mathbf{C}\mathbf{x}$$

Order: 2

A=

	1	2
1	0	1
2	-2	-3

B =

1	0
2	1

C =

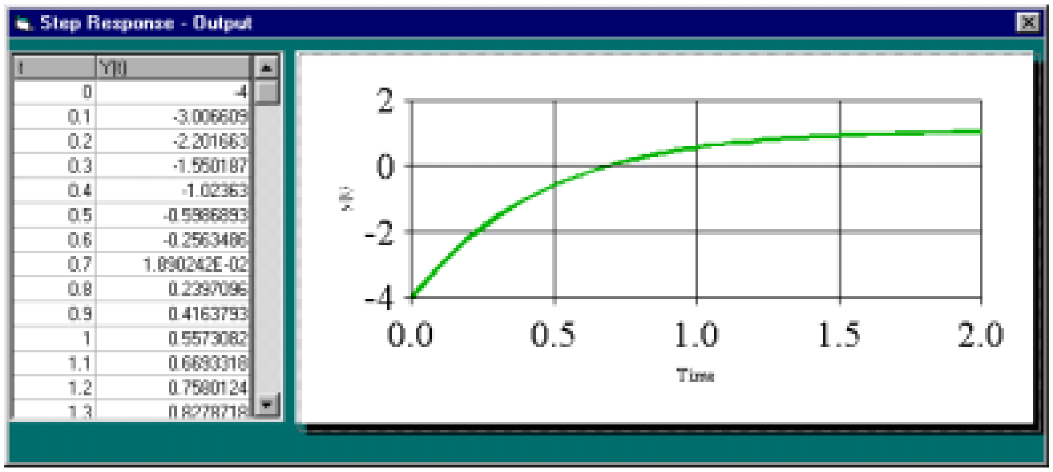
1	2
2	3

x(0)=

1	1
2	-2

Iteration Interval: .001
Print Interval: .1
Total Time: 2

Figure H.4 Step Response Program: output window



t	Y(t)
0	-4
0.1	-3.006609
0.2	-2.201663
0.3	-1.550187
0.4	-1.02363
0.5	-0.5986893
0.6	-0.2563486
0.7	1.090242E-02
0.8	0.2397096
0.9	0.4163793
1	0.5573082
1.1	0.6693318
1.2	0.7580124
1.3	0.8278718

H.2 Root Locus and Frequency Response

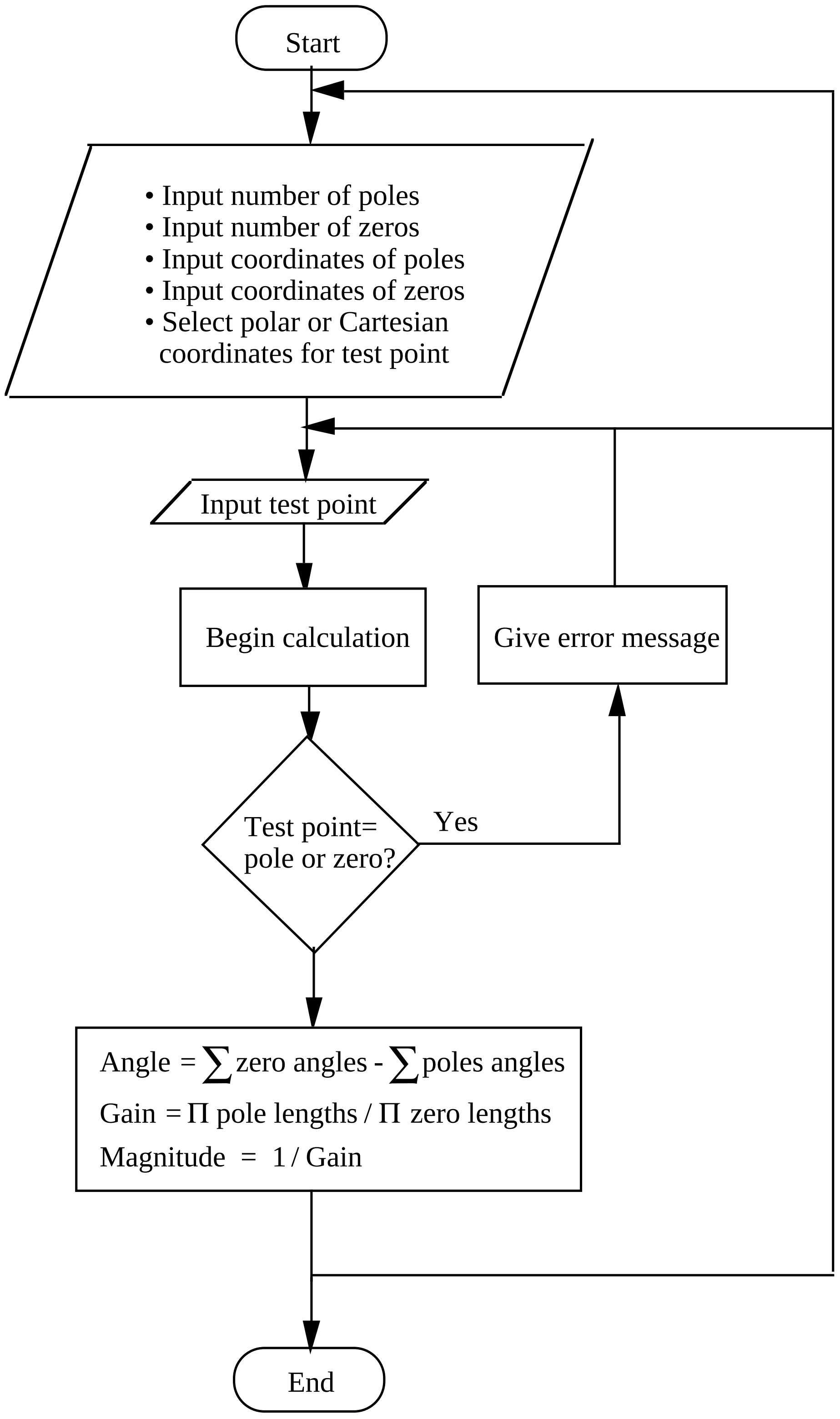
In this section we will develop a computer program that can be used as an alternative to MATLAB to search for points on the root locus and obtain magnitude and phase frequency response data. The code was developed using Visual Basic® Version 6 and converted to a stand-alone application that runs on a PC. The resulting application, recommended for readers who do not have access to MATLAB, can be found in the Appendix H folder on this CD-ROM. To run the setup program, open the folder labeled Root Locus inside the Appendix H folder and double-click on setup icon. For directions on running the program see the README file inside the Root Locus folder. The program also can be adapted to hand-help calculators. Let us now summarize the design of the step response software.

First, we enumerate the software requirements as follows:

1. The user will input the number of open-loop poles and zeros.
2. The user will input the values of the open-loop poles and zeros.
3. The user will select polar or Cartesian coordinates for the test point.
4. The user will input the coordinates of the test point.
5. The user will initiate the calculation.
6. The program will display the angle and magnitude of the open-loop transfer function at the test point as well as the gain.
7. The user can change the open-loop poles and zeros and the test point before initiating another calculation.

A simplified flow-chart for the program is shown in Figure H.5.

Figure H.5 Flow-chart for Root Locus and Frequency Response Program


Start
• Input number of poles
• Input number of zeros
• Input coordinates of poles
• Input coordinates of zeros
• Select polar or Cartesian coordinates for test point
Input test point
Begin calculation
Give error message
Test point= pole or zero?
Yes
Angle = ∑ zero angles - ∑ poles angles
Gain = Π pole lengths / Π zero lengths
Magnitude = 1 / Gain
End

Code Module

We now present a sample implementation of the flow-chart of Figure H.5. The routine can run independently, as part of a Visual Basic Code Module, or tailored to another programming language or other machines, such as hand-held calculators.

The routine can obtain its input variable values through the Visual Basic GUI interface, as presented in the sample run below, or through another program written to pass this code the input variables. The same is true of the output variables. In the sample run below, output variables are passed to the Visual Basic GUI interface for display, but could just as well be passed to another program.

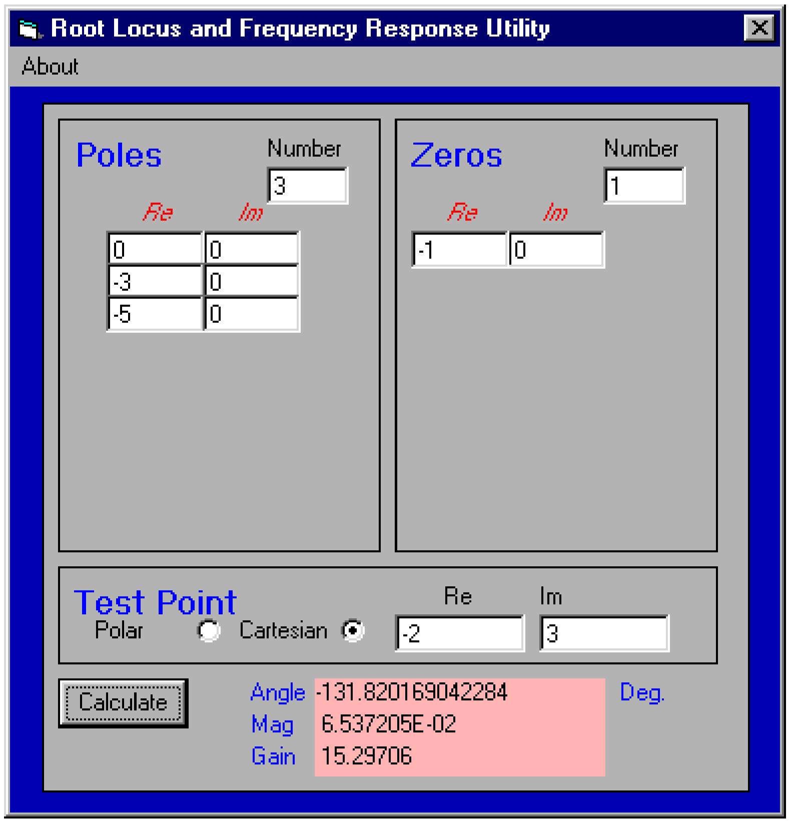
We now list the sample subroutine, which we call RootLocusCalc :

```
'**********************Input Variables**********************
Public Polar        'True or False. Determines if input test
                    'point is interpreted as polar coordinates
                    '(Polar = True or Cartesian Coordinates
                    '(Polar = False).

Public testAVal     'Test point x coordinate (Polar = False) or
                    'test point magnitude (Polar = True).
Public testBVal     'Test point y coordinate (Polar = False) or
                    'test point angle (Polar = True).

Public NumPolesVal  'Number of poles in system (0 to 10).
Public NumZerosVal  'Number of Zeros in system (0 to 10).

'The following variables have enough space for 10 poles,
'but only the number of data points referred to by NumPolesVal
'are stored in the array starting at the 0th element.

Public RLPoleRe(10)  'Stores each open-loop poles's real part.
Public RLPoleI(10)   'Stores each open-loop poles's imaginary part.

'The following variables have enough space for 10 zeros,
'but only the number of data points referred to by NumZerosVal
'are stored in the array starting at the 0th element.

Public RLZeroRe(10)  'Stores each open-loop zero's real part.
Public RLZeroIm(10)  'Stores each open-loop zero's imaginary part.

'*********************Output Variables**********************
Public RLKval        'Returns the Gain at the given test point.
Public RLMagVal      'Returns the magnitude at the given test point.
Public RLAngleVal    'Returns the angle in degrees at the given
                     'test point.

Public ErrorFlag     'If this variable is set to True, then
                     'an error occurred during calculation
                     'and the output data is disregarded.

'******************Subroutine RootLocusCalc******************
Const pi = 3.14159265358979

Public Sub RootLocusCalc()
      Dim deltaX As Single
      Dim deltaY As Single
      ErrorFlag = False
      RecGain = 1
      angle = 0

If Polar = True Then  'Convert polar test point to Cartesian.
          testReVal = testAVal * Cos(testBVal * pi / 180)
          testImVal = testAVal * Sin(testBVal * pi / 180)
      Else            'Test point is Cartesian - use as is.
          testReVal = testAVal
          testImVal = testBVal
      End If

      For K = 0 To NumPolesVal - 1
        ReVal = RLPoleRe(K)
        ImVal = RLPoleI(K)

        deltaX = testReVal - ReVal
        deltaY = testImVal - ImVal

        If Abs(deltaX) < 0.0000000001 And _
           Abs(deltaY) < 0.0000000001 Then
            MsgBox ("ERROR: Test point is the same as an " & _
                    "open-loop pole. Enter new test point.")
            ErrorFlag = 1
            GoTo exitrootlocus
        Else
            RecGain = RecGain * CartToMag(deltaX, deltaY)
            angle = angle - CartToAngle(deltaX, deltaY)
        End If
      Next K

     For K = 0 To NumZerosVal - 1
        ReVal = RLZeroRe(K)
        ImVal = RLZeroIm(K)

        deltaX = testReVal - ReVal
        deltaY = testImVal - ImVal

        If Abs(deltaX) < 0.0000000001 And _
           Abs(deltaY) < 0.0000000001 Then
            MsgBox ("ERROR: Test point is the same as an " & _
                    "open-loop zero. Enter new test point.")
            ErrorFlag = 1
            GoTo exitrootlocus
       Else
            RecGain = RecGain / CartToMag(deltaX, deltaY)
            angle = angle + CartToAngle(deltaX, deltaY)
       End If
      Next K
      angle = angle * 180 / pi
      angle = (angle / 360 - Fix(angle / 360)) * 360
exitrootlocus:
      If ErrorFlag <> 1 Then
RLKval = RecGain
  RLMagVal = 1 / RecGain
        RLAngleVal = angle
      End If

      End Sub
```

```
'*********************Function CartToMag*********************
Public Function CartToMag(X As Single, Y As Single) As Single
  CartToMag = Sqr(Abs(X ^ 2 + Y ^ 2))
End Function

'********************Function CartToAngle********************
Public Function CartToAngle(deltaX, deltaY) As Single
   If deltaX = 0 Then angle = pi / 2 _
   Else angle = Atn(Abs(deltaY) / Abs(deltaX))
   If deltaY >= 0 And deltaX >= 0 Then angle = angle
   If deltaY >= 0 And deltaX < 0 Then angle = (pi - angle)
   If deltaY < 0 And deltaX <= 0 Then angle = -(pi - angle)
   If deltaY < 0 And deltaX > 0 Then angle = -angle
   CartToAngle = angle
End Function
```

Data entry and results for the code shown above are via a graphical user interface (GUI) developed in Visual Basic 6 and produced by the stand-alone application included in the Appendix H folder. Figure H.6 shows the GUI interface for data entry and results using $G(s) = \frac{(s+1)}{s(s+3)(s+5)}$ and a test point = -2 + j3 as an example.

Figure H.6 Root Locus Program: GUI interface for data entry and results



Root Locus and Frequency Response Utility

About

Poles — Number: 3

Re	Im
0	0
-3	0
-5	0

Zeros — Number: 1

Re	Im
-1	0

Test Point — Polar ○ Cartesian ● — Re: -2 Im: 3

Calculate

Angle -131.820169042284 Deg.
Mag 6.537205E-02
Gain 15.29706