
AC 2012-4561: MATHEMATICAL MODELING AND SIMULATION USING LABVIEW AND LABVIEW MATHSCRIPT

Dr. Nikunja Swain, South Carolina State University

Nikunja Swain is a professor in the College of Science, Mathematics, Engineering and Technology at South Carolina State University. He is involved in teaching various courses in engineering technology and computer science. He holds a Ph.D. in electrical/energy engineering, a M.S. in electrical and computer engineering, and a M.S. in electrical engineering. He is a member of ACM, ACM SIGITE, IEEE, IEEE CS, and ASEE. He is a registered Professional Engineer (PE) in South Carolina and TAC/ABET evaluator for Computer engineering technology and electrical engineering technology.

Mathematical Modeling and Simulation using LabVIEW and LabVIEW MathScript

Abstract

There are numerous uses of simulation, starting from simulation of simple electric circuits to complex tasks such as electromagnetic fields, heat transfer through materials, networking, computer circuits, game programming, electron flow in semiconductors, or beam loading with the ultimate objective of providing illustrations of concepts that are not easily visualized and difficult to understand. Simulators are also used as an adjunct to and, in some cases such as distance learning courses, as a substitute for actual laboratory experiments. LabVIEW and LabVIEW MathScript are currently used in a number of science, engineering and technology programs and industries for simulation and analysis. This paper will discuss design and development of interactive instructional modules for Control Systems and Numerical Analysis Courses using LabVIEW and LabVIEW MathScript.

Introduction

Simulation exercises are integral part of the science, engineering, and technology programs. Simulation exercises provide verification of the basic theory and reinforcement of the underlying principles; greater attention to the theoretical limitations; application of logical analysis to solve real world problems. There are number of use of simulation, starting from simulation of simple electric circuits to complex tasks such as electromagnetic fields, heat transfer through materials, modeling and evaluation of energy systems, networking, computer circuits, game programming, and electron flow in semiconductors or beam loading with the ultimate objective of providing illustrations of concepts that are not easily visualized, and difficult to understand. Simulators are also used as an adjunct to and in some cases (distance learning courses) as a substitute for actual laboratory experiments. In many instances the student are required to verify their theoretical design through simulation before building and testing the circuit in the laboratory. Studies show that students who used simulation prior to conducting actual experiment performed better than the students who conducted the laboratory experiments without conducting simulation first.

Currently, science and engineering instructors use a number of software packages to achieve this objective. In this paper, we propose using LabVIEW and MathScript software (National Instruments Corp., Austin, Texas) to design virtual interactive instructional modules to teach mathematics and control systems courses. LabVIEW is based on graphical programming and easy to use. The virtual interactive problem-solving environment enables students to analyze, visualize, and document real-world science and engineering problems. MathScript is a high-level, text- based programming language. MathScript includes more than 800 built-in functions and the syntax is similar to MATLAB. MathScript is an add-on module to LabVIEW but one doesn't need to know LabVIEW programming in order to use MathScript. In addition to the MathScript built-in functions, the *LabVIEW Control Design and Simulation Module* and *LabVIEW Digital Filter Design Toolkit* have lots of additional functions.

Using LabVIEW and MathScript, students can be exposed to applied technology and science experiments, based on theoretical science and mathematics concepts, in introductory engineering, technology, and science courses. In their junior and senior years, students would be introduced to advanced experiments. The interactive virtual modules present a wide variety of hypothetical and real-world problems with examples for students to solve. Instructors would introduce students to the lab manuals and the steps involved in the design process; such as, solving the problem theoretically, selecting proper methodologies, designing the experimental setup, using the required software package to simulate the experiment, and to compare the theoretical and simulation results. This would eliminate the cookbook approach that is the mainstay of most of engineering, technology, and science departments.

Example of a LabVIEW and Mathscript Numerical Analysis Module

LabVIEW is extremely flexible and some of the application areas of LabVIEW are Simulation, Data Acquisition and Data Processing. The Data Processing library includes signal generation, digital signal processing (DSP), measurement, filters, windows, curve fitting, probability and statistics, linear algebra, numerical methods, instrument control, program development, control systems, and fuzzy logic. These features of LabVIEW and Mathscript have helped us in providing an Interdisciplinary Integrated Teaching and Learning experiences that integrates team-oriented, hands-on learning experiences throughout the engineering technology and sciences curriculum and engages students in the design and analysis process beginning with their first year.

Bisection Method

One of the first numerical methods developed to find the root of a nonlinear equation $f(x) = 0$ was the bisection method (also called *binary-search* method). Since the method is based on finding the root between two points, the method falls under the category of bracketing methods. Given a closed interval $[a, b]$ on which $f(x)$ changes sign, we divide the interval in half and note that f must change sign on either the right or the left half (or be zero at the midpoint of $[a, b]$.) We then replace $[a, b]$ by the half-interval on which f changes sign. This process is repeated until the interval has total length less than ϵ . In the end we have a closed interval of length less than ϵ on which f changes sign. The Intermediate Value Theorem (IVT) guarantees that there is a zero of f in this interval. The endpoints of this interval, which are known, must be within ϵ of this zero.

The method is based on the following algorithm:

Initialization: The bisection method is initialized by specifying the function $f(x)$, the interval $[a, b]$, and the tolerance $\epsilon > 0$.

We also check whether $f(a) = 0$ or $f(b) = 0$, and if so return the value of a or b and exit.

Loop: Let $m = (a + b)/2$ be the midpoint of the interval $[a, b]$. Compute the signs of $f(a)$, $f(m)$, and $f(b)$. If any are zero, return the corresponding point and exit.

Assuming none are zero, if $f(a)$ and $f(m)$ have opposite sides, replace b by m , else replace a by m . If the length of the $[a,b]$ is less than ϵ , return the value of a and exit.

LabVIEW and Mathscript modules for Bisection

The algorithm presented above is used to design and test virtual instrument module to find a root of the equation $f(x) = x - 0.2\sin x - 0.5$ in the interval 0 to 1. The results of the module are consistent with the theoretical calculations. Figure 1 shows the front panel and Figure 2 shows the diagram panel of this module. Figure 3 shows the Mathscript code and result.

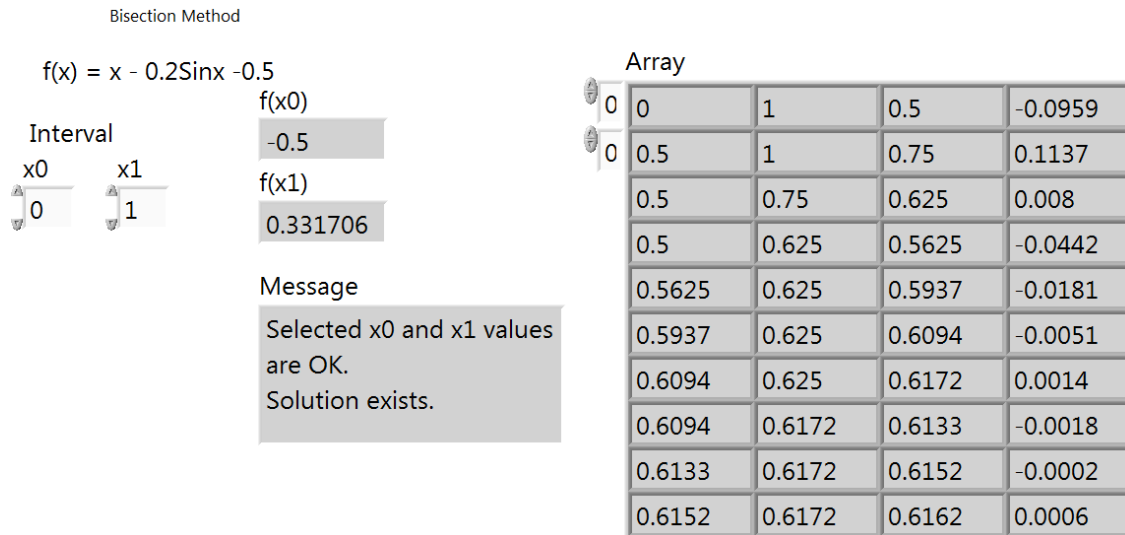


Figure 1 – Front Panel of Bisection virtual instruments module

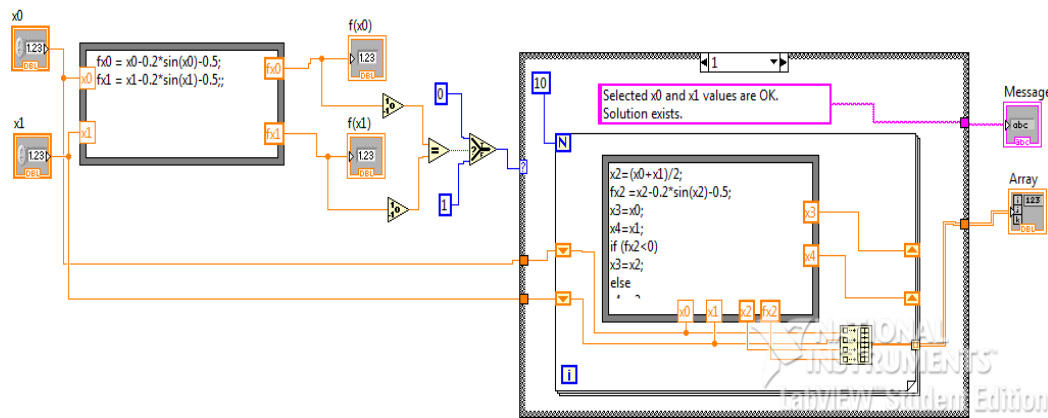


Figure 2 – Front Panel of Bisection virtual instruments module

Mathscript m file for Bisection Method

```
function approx_root = bisect ( a, b )
% bisect finds an approximate root of the function cosy using bisection

fa = a - 0.2*sin(a) - 0.5;
fb = b - 0.2*sin(b) - 0.5;
```

```

while ( abs ( b - a ) > 0.000001 )

    c = ( a + b ) / 2;
    approx_root = c;
    fc = c - 0.2*sin(c) - 0.5;

% What follows is just a nice way to print out a little table.  It does not add to
% the algorithm itself, it only makes it easier to see what is going on at runtime.
%
[ a, c, b;
  fa, fc, fb ]

if ( sign(fb) * sign(fc) <= 0 )
    a = c;
    fa = fc;
else
    b = c;
    fb = fc;
end

end

```

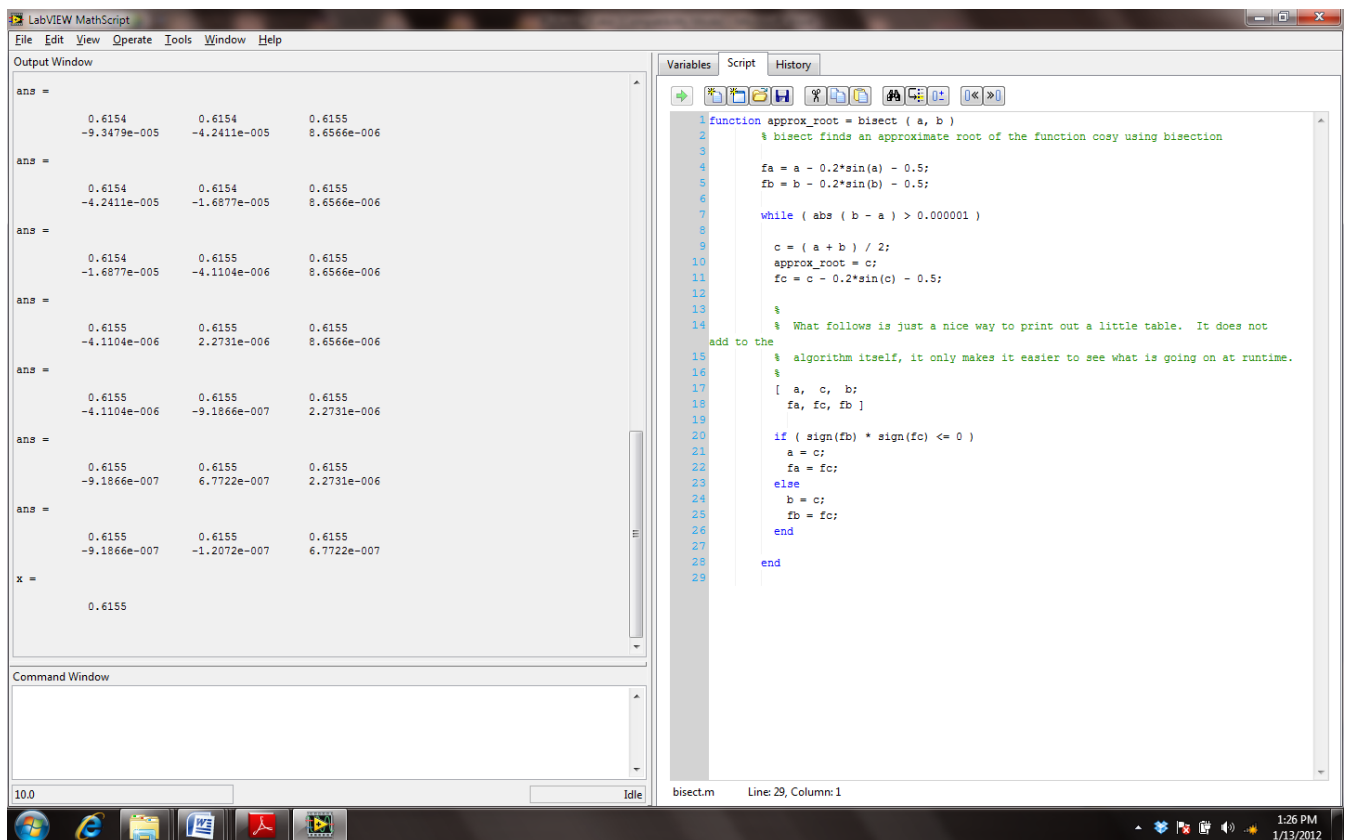


Figure 3 – Mathscript code and result

Newton's Method

Methods such as the bisection method and the false position method of finding roots of a nonlinear equation $f(x) = 0$ require bracketing of the root by two guesses. Such methods are called *bracketing methods*. These methods are always convergent since they are based on reducing the interval between the two guesses so as to zero in on the root of the equation. In the Newton-Raphson method, the root is not bracketed. In fact, only one initial guess of the root is needed to get the iterative process started to find the root of an equation. The method hence falls in the category of *open methods*. Convergence in open methods is not guaranteed but if the method does converge, it does so much faster than the bracketing methods.

Algorithm

The steps of the Newton-Raphson method to find the root of an equation $f(x) = 0$ are

1. Evaluate $f'(x)$ symbolically
2. Use an initial guess of the root, x_i , to estimate the new value of the root, x_{i+1} , as

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

3. Find the absolute relative approximate error $|\epsilon_a|$ as

$$|\epsilon_a| = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \times 100$$

Compare the absolute relative approximate error with the pre-specified relative error tolerance, ϵ_s . If $|\epsilon_a| > \epsilon_s$, then go to Step 2, else stop the algorithm. Also, check if the number of iterations has exceeded the maximum number of iterations allowed. If so, one needs to terminate the algorithm and notify the user.

LabVIEW module for Newton's Method

The algorithm presented above is used to design and test virtual instrument module to find a root of the equation $f(x) = x^3 - 2x - 5 = 0$ starting at $x = 3$. The results of the module are consistent with the theoretical calculations. Figure 4 shows the front panel and Figure 5 shows the diagram panel of this module. In Figure 4, column 1 represents number of iteration, column 2 represents x_i , column 3 represents $f(x_i)$, column 4 represents $f'(x)$, and column 5 represents x_{i+1} .

Equation: $x^3 - 2x - 5 = 0$
 Method: Newton-Raphson Method
 Let $f(x) = x^3 - 2x - 5$; $f'(x) = 3x^2 - 2$
 $x_{n+1} = x_n - f(x_n)/f'(x_n)$ Starting $x = 3$

Starting x	Results				
3	0	3	16	25	2.36
	1	2.36	3.4243	14.7088	2.1272
	2	2.1272	0.3711	11.5749	2.0951
	3	2.0951	0.0065	11.1688	2.0946
	4	2.0946	2.1461E-6	11.1614	2.0946
	5	2.0946	2.3205E-13	11.1614	2.0946
	6	2.0946	-9.116E-16	11.1614	2.0946
	7	2.0946	-9.116E-16	11.1614	2.0946
	8	2.0946	-9.116E-16	11.1614	2.0946
	9	2.0946	-9.116E-16	11.1614	2.0946

Figure 4 – Front Panel of Newton’s method module

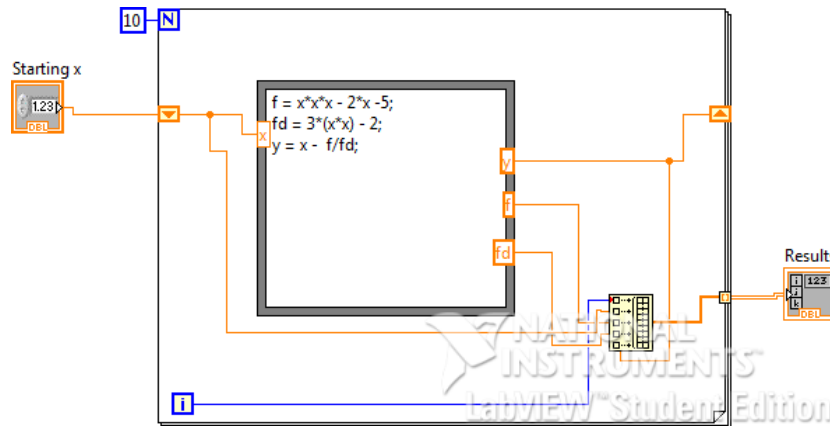


Figure 5 – Diagram Panel of Newton’s method module

Interpolation Method

Polynomial interpolation involves finding a polynomial of order n that passes through the $n + 1$ points. One of the methods of interpolation is called the direct method. Other methods include Newton’s divided difference polynomial method and the Lagrangian interpolation method. Our discussion will be limited to the Direct Method.

The direct method of interpolation is based on the following premise. Given $n + 1$ data points, fit a polynomial of order n as given below

$y = a_0 + a_1x + \dots + a_nx^n$, through the data, where a_0, a_1, \dots, a_n are $n + 1$ real constants. Since $n + 1$ values of y are given at $n + 1$ values of x , one can write $n + 1$ equations. Then the $n + 1$ constants, a_0, a_1, \dots, a_n , can be found by solving the $n + 1$ simultaneous linear equations. To find the value of y at a given value of x , simply substitute the value of x in Equation.

LabVIEW module for Direct Method

The algorithm presented above is used to design and test virtual instrument module for direct method of interpolation. The results of the module are consistent with the theoretical calculations. Figure 6 shows the front panel and Figure 7 shows the diagram panel of this module.

USER INPUTS/SELECTION

Example 1

The upward velocity of a rocket is given as a function of time in Table 1.

Table 1 Velocity as a function of time.

t (s)	$v(t)$ (m/s)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

Selected Times/Corresponding velocity

t0	t1	t2
10	15	20
v(t0)	v(t1)	v(t2)
227.04	362.78	517.35

Ring

Direct Interpolation

t desired

16

Determine the value of the velocity at $t = 16$ seconds.

RESULTS

A Matrix	B Matrix	Result	a0	a1	a2															
<table border="1"> <tbody> <tr> <td>1</td> <td>10</td> <td>100</td> </tr> <tr> <td>1</td> <td>15</td> <td>225</td> </tr> <tr> <td>1</td> <td>20</td> <td>400</td> </tr> </tbody> </table>	1	10	100	1	15	225	1	20	400	<table border="1"> <tbody> <tr> <td>227.04</td> </tr> <tr> <td>362.78</td> </tr> <tr> <td>517.35</td> </tr> </tbody> </table>	227.04	362.78	517.35	<table border="1"> <tbody> <tr> <td>12.05</td> </tr> <tr> <td>17.733</td> </tr> <tr> <td>0.3766</td> </tr> </tbody> </table>	12.05	17.733	0.3766	12.05	17.733	0.3766
1	10	100																		
1	15	225																		
1	20	400																		
227.04																				
362.78																				
517.35																				
12.05																				
17.733																				
0.3766																				
			vt at t = 16																	
			392.188																	

Figure 6 – Front Panel of Direct method of Interpolation

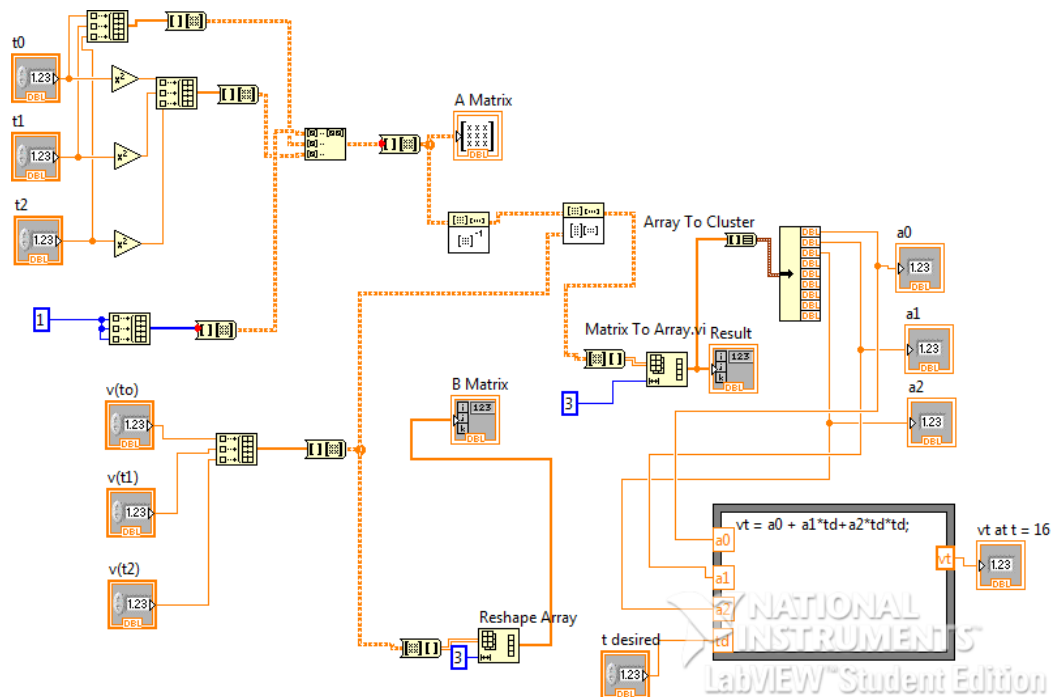


Figure 7 – Diagram Panel of Direct method of Interpolation

Example of a LabVIEW and Mathscript Control Systems Module

The frequency response is a representation of the system's response to sinusoidal inputs at varying frequencies. The output of a linear system to a sinusoidal input is a sinusoid of the same frequency but with a different magnitude and phase. The **frequency response** is defined as the magnitude and phase differences between the input and output sinusoids. In this tutorial, we will see how we can use the open-loop frequency response of a system to predict its behavior in closed-loop.

The **gain margin** is defined as the change in open loop gain required to make the system unstable. Systems with greater gain margins can withstand greater changes in system parameters before becoming unstable in closed loop.

The **phase margin** is defined as the change in open loop phase shift required to make a closed loop system unstable. The phase margin also measures the system's tolerance to time delay. If there is a time delay greater than $180/W_{pc}$ in the loop (where W_{pc} is the frequency where the phase shift is 180 deg), the system will become unstable in closed loop. The time delay can be thought of as an extra block in the forward path of the block diagram that adds phase to the system but has no effect the gain. That is, a time delay can be represented as a block with magnitude of 1 and phase $w*time_delay$ (in radians/second).

The bode plot, phase margin, and gain margin of the closed loop system $T(S) = 2/(s^3 + 3s^2 + 2s + 2)$ is calculated using LabVIEW and mathscript. The results of the simulation are consistent with the theoretical calculations. Figure 8 represents the Front panel, Figure 9 represents the Diagram panel of LabVIEW simulation. Figure 10 represents the corresponding Mathscript simulation.

Front Panel of Gain_Phase Margin Virtual Instrument Module

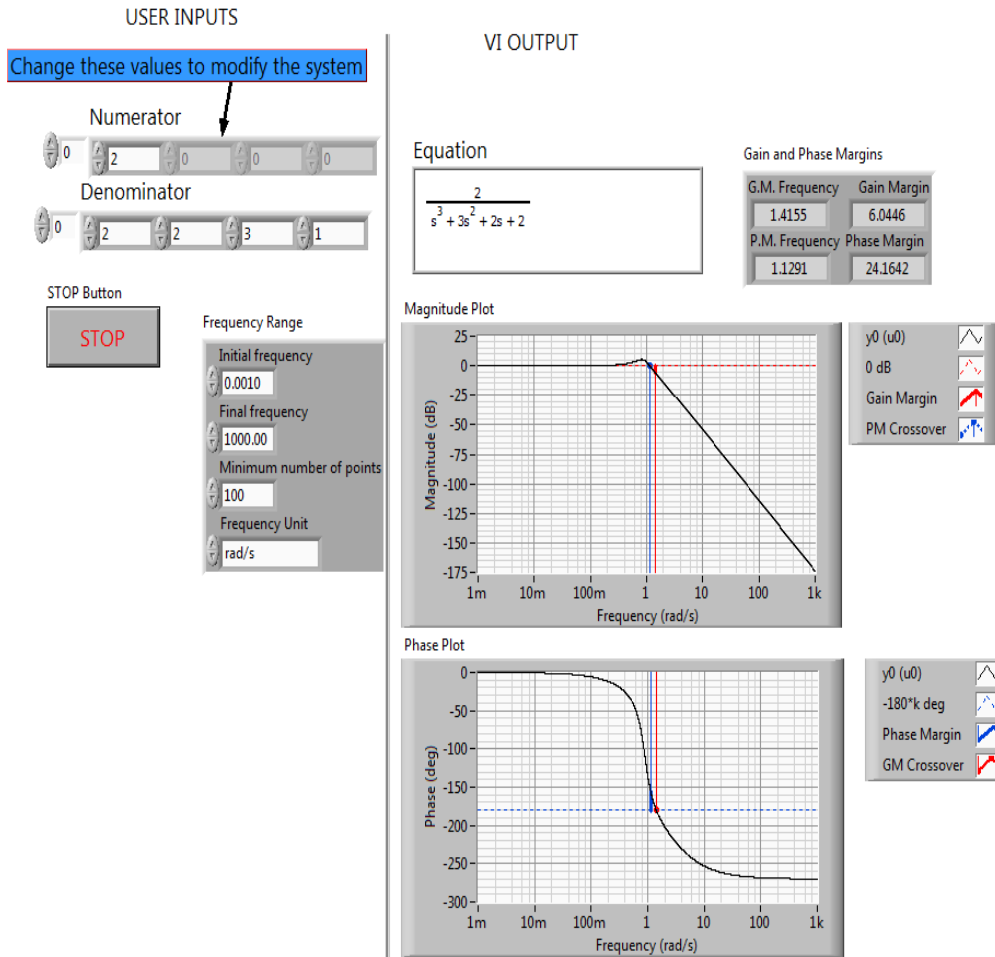


Figure 8 – Front Panel of Phase and Gain Margin Module

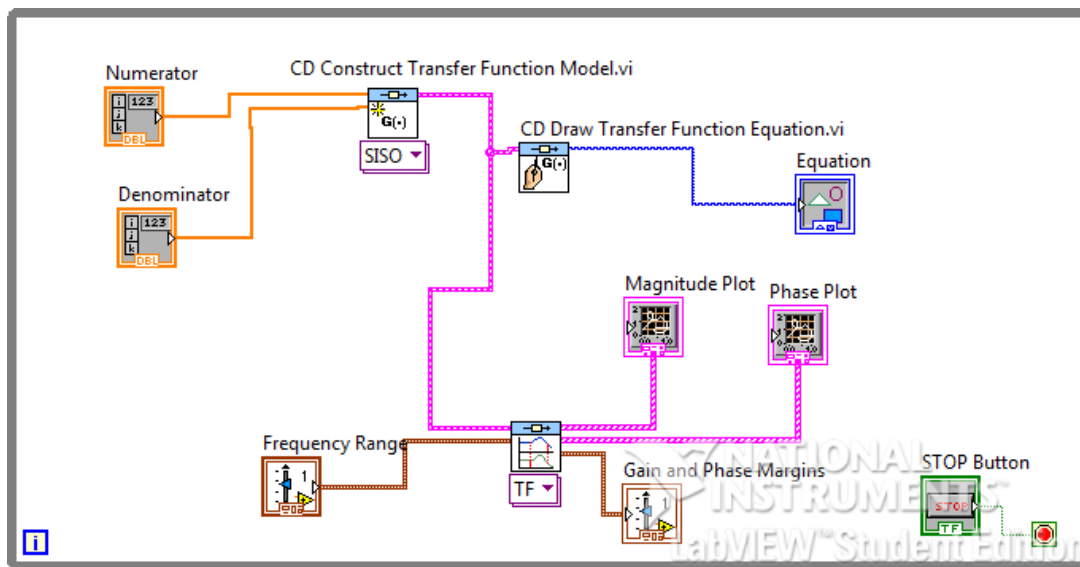


Figure 9 – Diagram Panel of Phase and Gain Margin Module

MathScript Code for Phase-Gain Margins

```
num = 2;  
den = [1 3 2 2];  
sys = tf(num,den);  
margin(sys)
```

MathScript Simulation Results

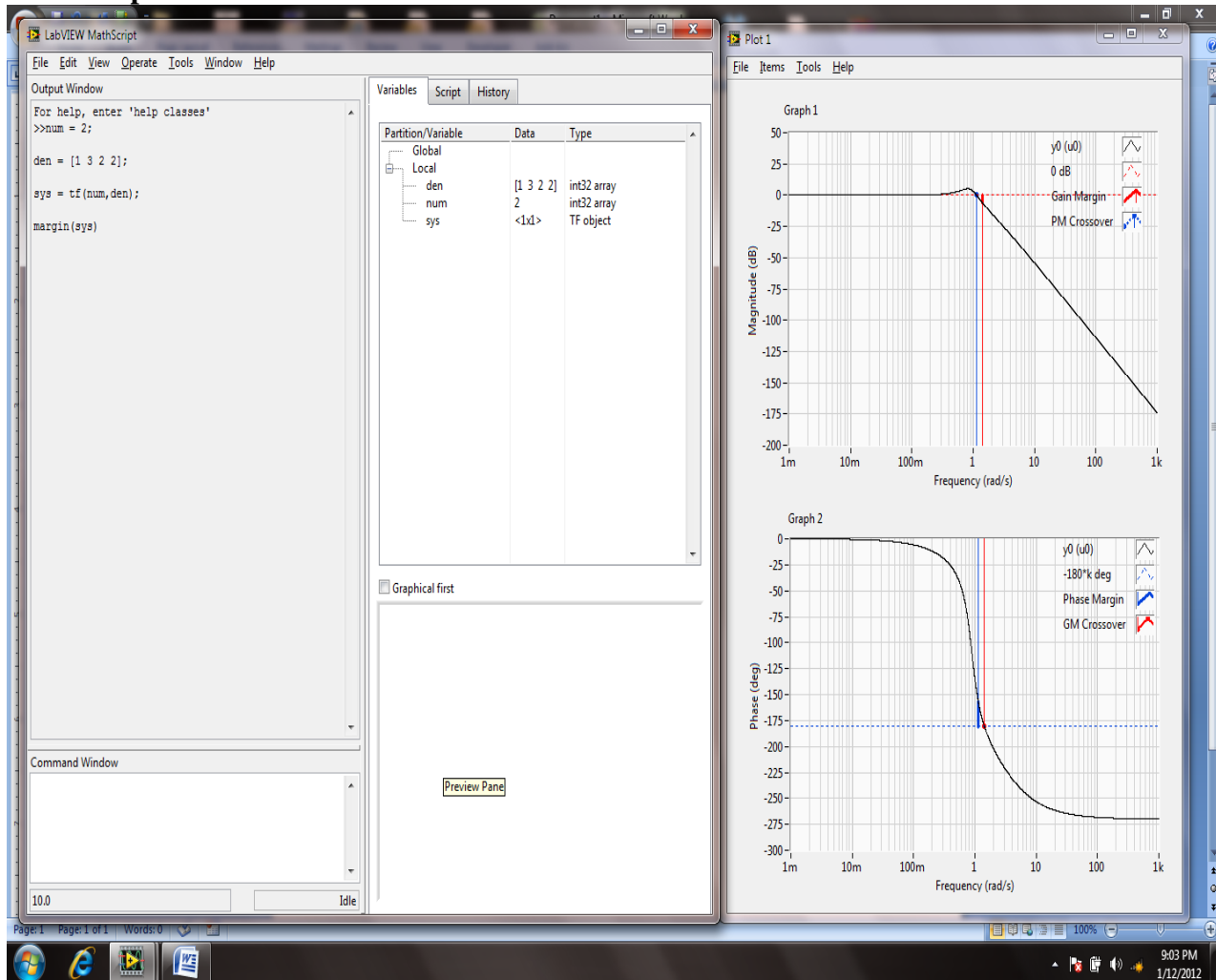


Figure 10 – MathScript Simulation Results for Phase and Gain Margin

Summary and Conclusions

The sample modules presented above are user friendly and performed satisfactorily under various input conditions. These and other modules helped the students to understand the concepts in more detail. These modules can be used in conjunction with other teaching aids to enhance student learning in various courses and will provide a truly modern environment in which students and faculty members can study engineering, technology, and sciences at a level of detail.

The LabVIEW modules are designed using built in Mathematics and other libraries and can be easily extended to accommodate more complex problems. The Control Design and Simulation Module in LabVIEW contains number of Toolbox consists of number of functions related to classical and modern control systems design and analysis. National Instruments website provides numerous tutorials and application examples on the use of these modules in real world situations. These will be definitely helpful to students as well as faculty to design and develop virtual instrument modules for various applications.

MathScript is a high-level, text- based programming language and is easy to use. MathScript is an add-on module to LabVIEW but one doesn't need to know LabVIEW programming in order to use MathScript. MathScript includes more than 800 built-in functions and the syntax is similar to MATLAB. LabVIEW with MathScript may be enough to address many of the simulation needs of a technology program.

Bibliography

1. Elaine L., Mack, Lynn G. (2001), "Developing and Implementing an Integrated Problem-based Engineering Technology Curriculum in an American Technical College System" Community College Journal of Research and Practice, Vol. 25, No. 5-6, pp. 425-439.
2. Kellie, Andrew C., And Others. (1984), "Experience with Computer-Assisted Instruction in Engineering Technology", Engineering Education, Vol. 74, No. 8, pp712-715.
3. Lisa Wells and Jeferey Travis, LabVIEW for Everyone, Graphical Programming Even Made Easier, Prentice Hall, NJ 07458, 1997.
4. URL: <http://numericalmethods.eng.usf.edu/>
5. URL: <http://www.ni.com>
6. Dorf, Richard and Bishop, Robert, Modern Control Systems, Prentice Hall, NJ 07458, 2008.