Exploits in DSP

Murat Tanyel
Department of Engineering
Geneva College
Beaver Falls, PA 15010
Email: mtanyel@geneva.edu

Abstract

Revision of a course is an exciting opportunity to develop new examples and demonstrations. Last year's revision of my Digital Signal Processing (DSP) course allowed me to take advantage of a new textbook's rich supply of MATLAB examples by merging them with my decades of experience in utilizing LabVIEW in DSP. I was able to offer students both MATLAB and LabVIEW versions of exercises, exposing them to two leading software environments in the field simultaneously. Utilizing LabVIEW's incorporation of Mathscript in its programming environment, we were able to develop hybrid MATLAB-LabVIEW code or write entire MATLAB programs in LabVIEW VIs. This paper will share some of these examples and compare results obtained from the two different environments. It will highlight the relative strengths of the two environments and conclude with student reactions to this approach.

Introduction

These days, digital signal processing has become ubiquitous. My informal interactions with students in Signals & Systems and DSP classes has revealed that a technologically savvy college student has become aware of DSP in electronic music synthesizers, sound cards in PCs, equalizers in car stereos. Students may have even heard about DSP chips, oversampling digital filters, 1-bit A/D and D/A converters, wavetable sound synthesis, audio effects processors, or all-digital audio studios if they have had an interest in music or audio recording. By the time they are seniors or juniors, which is when a typical Geneva College engineering student takes DSP, they are ready and eager to learn more about DSP. As the instructor of their introductory DSP class, I hold myself responsible for not only enlightening them in the theoretical background that gives rise to these "cool" devices buts also keeping their interest through relevant examples and demonstrations.

The textbook chosen for the course usually sets the tone in what happens in the classroom. I remember my experience as a teaching assistant for the senior level DSP course which used a classical textbook in the field[1], in which students were lost in the sea of contour integrations finding inverse z-transforms. My aim, therefore, in choosing the textbook is to provide students with a reference book which will offer a solid theoretical background while also including relevant examples that may serve as building blocks for applications in their future careers. To that end, I have used textbooks by Orfanidis[2] and Mitra[3] in the past. My choice of a new textbook by Tan and Jian[4] last academic year was premature, but the rich array of real-life examples in the textbook, coupled with data files for those examples in the instructors' resources website convinced me to make the switch.

As most textbooks in DSP, Tan & Jian utilize MATLAB as their computational tool. As I have explained in previous publications[5,6,7], I use LabVIEW as the computational tool in DSP. Since both MATLAB and LabVIEW have become industry standards, I decided to introduce both of these computing environments in the most recent (Fall '16) offering of DSP. I was able to combine these two tools thanks to LabVIEW's MathScript Node. LabVIEW's help file describes this tool as follows: "Executes LabVIEW MathScripts and your other text-based scripts using the MathScript RT Module engine." In other words, the user can type in MATLAB scripts in this node and LabVIEW will execute those scripts. Armed with this tool, I prepared my class demos in two versions, one with MathScript Node running MATLAB code and one with LabVIEW's equivalent function(s) from its signal processing toolkit. In the lab, the students were asked to try out both methods in introductory exercises and were left free to choose either method in more open ended assignments. In the next section, I will provide a sample of examples demonstrating this approach.

Examples

In the first example we will review a problem for which LabVIEW does not provide a tool. We will see how such a problem can be solved quickly without leaving the LabVIEW platform utilizing the MathScript node.

In digital filter design, we sometimes resort to the results of analog filter design. We start with the transfer function of a known analog filter and apply bilinear transformation to obtain the coefficients of the corresponding digital filter. Therefore, even students of digital signal processing should know a little about analog filter design. LabVIEW provides utilities to design digital filters directly and therefore does not offer any analog filter tools. MATLAB, on the other hand, offers such tools. One of those tools is the MATLAB function **freqs()**, which can be used to plot analog filter frequency responses for verification.

Example 8.2 of the textbook demonstrates the application of lowpass prototype transformation design method, in which an analog lowpass filter with a cutoff frequency of 1 radian per second, called the lowpass prototype, is converted to desired analog lowpass, highpass, bandpass or bandstop filters with specified frequencies. In the example, a lowpass prototype with the transfer function

$$H_p(s) = \frac{1}{s+1} \quad \text{(Eq. 1)}$$

is converted into (a) a highpass filter with a cutoff frequency of 40 radians per second and (b) a bandpass filter with a center frequency of 100 radians per second and bandwidth of 20 radians per second. The results of the example can be generalized to yield the transfer functions

$$H_{HP}(s) = \frac{s}{s+\omega_c} \qquad \text{(Eq. 2)}$$

and

$$H_{BP}(s) = \frac{\omega_{BW} s}{s^2 + \omega_{BW} s + \omega_0^2} \qquad \text{(Eq. 3)}$$

where $H_{HP}(s)$, $H_{BP}(s)$, ω_c, ω_{BW} and ω_0 are the highpass filter transfer function, highpass cutoff frequency, bandpass filter transfer function, bandwidth and center frequency of the bandwidth filter, respectively.

Figure 1 depicts the front panel of a VI which plots the magnitude responses of the highpass and bandpass filters of the textbook example 8.2. The front panel employs LabVIEW's standard XY graph tool and two knobs (wc for cutoff frequency and BW for ω_{BW}). A study of the block diagram that gave rise to these plots (Figure 2) reveals that it is actually a Mathscript node, which executes MATLAB commands, particularly the **freqs()** function, that is responsible for these plots. In essence, the whole VI is MATLAB dressed in LabVIEW's visual interface.

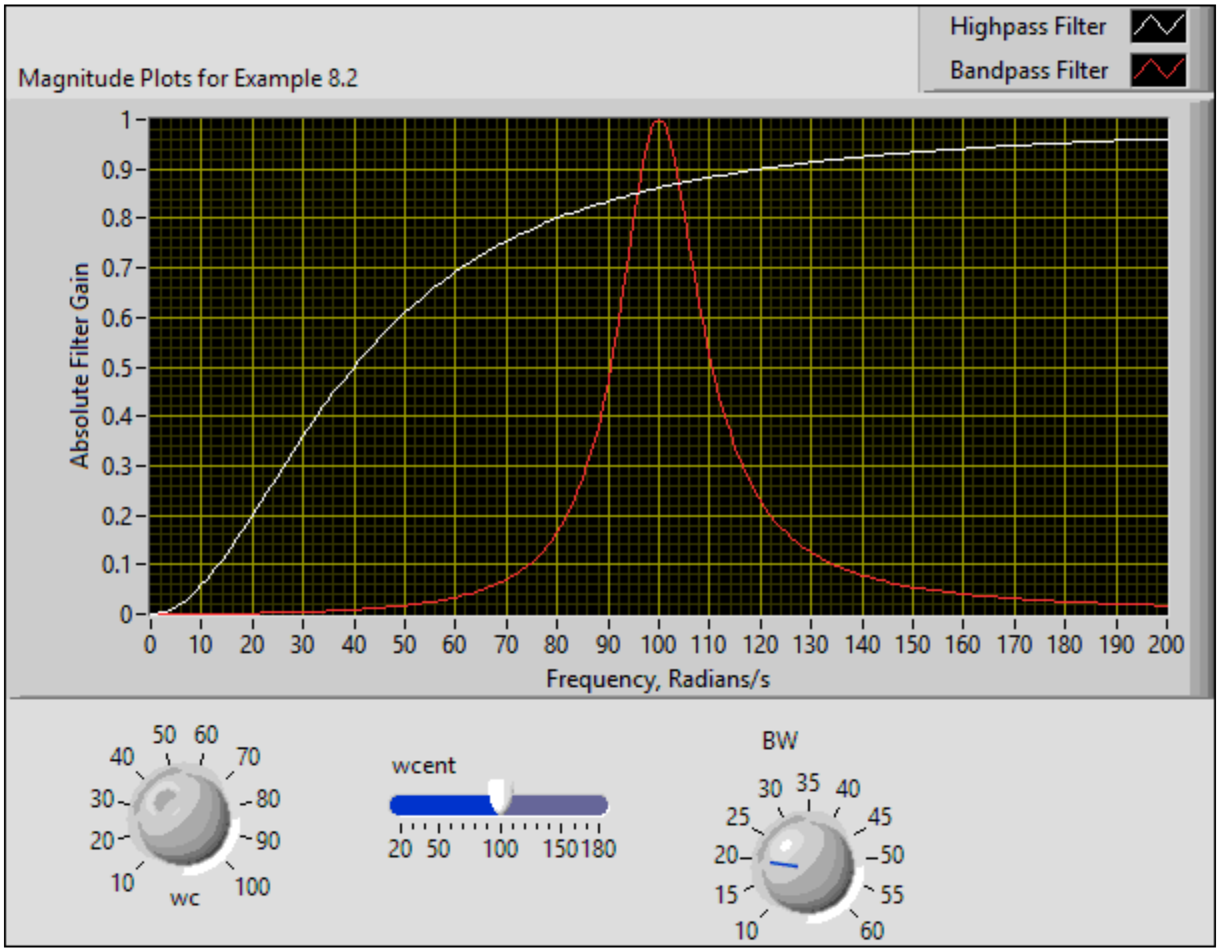


Figure 1: LabVIEW VI front panel showing the frequency responses of an analog highpass filter with a cut-off frequency of 40 radians per second and an analog bandpass filter of center frequency 100 and bandwidth of 20 radians per second.

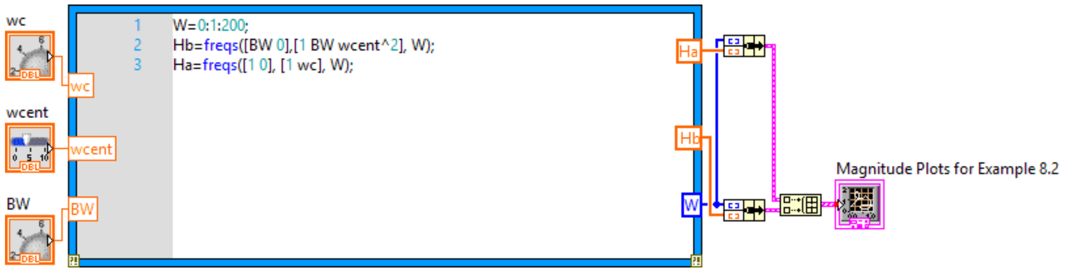

```
W=0:1:200;
Hb=freqs([BW 0],[1 BW wcent^2], W);
Ha=freqs([1 0], [1 wc], W);
```

Figure 2: The LabVIEW block diagram for the VI of Figure 1.

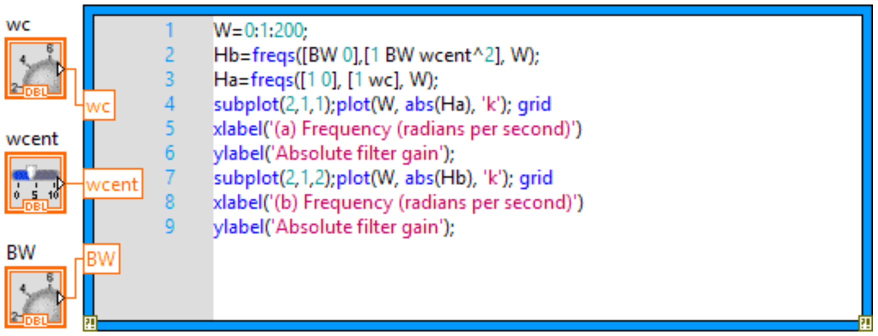

```
W=0:1:200;
Hb=freqs([BW 0],[1 BW wcent^2], W);
Ha=freqs([1 0], [1 wc], W);
subplot(2,1,1);plot(W, abs(Ha), 'k'); grid
xlabel('(a) Frequency (radians per second)')
ylabel('Absolute filter gain');
subplot(2,1,2);plot(W, abs(Hb), 'k'); grid
xlabel('(b) Frequency (radians per second)')
ylabel('Absolute filter gain');
```

Figure 3: MathScript node modified to incorporate MATLAB plotting functions eliminating the need for LabVIEW's XY Graph tool and its obligatory functions.

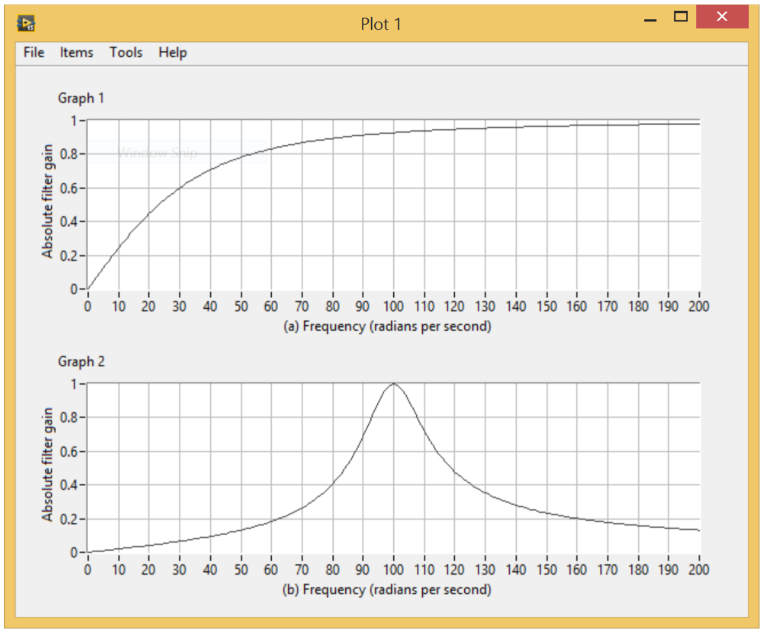


Figure 4: The Plot window which appears upon execution of the block diagram in Figure 3.

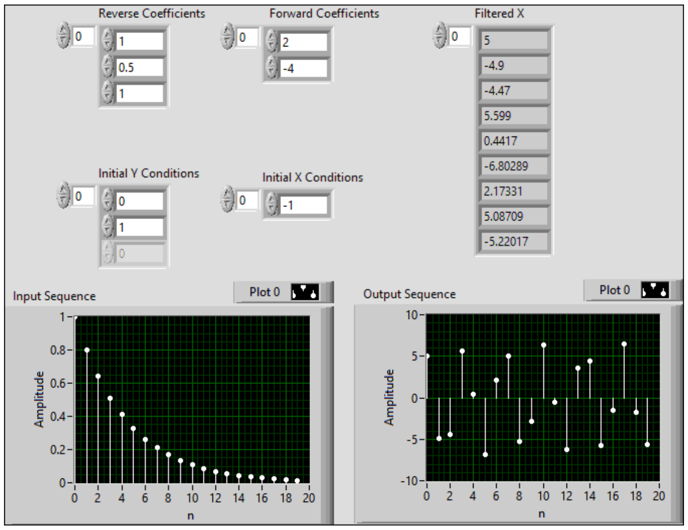
For those who would rather not deal with LabVIEW's XY graph and the Bundle, Build Array functions that it requires, the MathScript node even recognizes MATLAB's plotting commands and, when executed, generates a separate window with the plots (Figures 3 and 4).

In the second example we will compare LabVIEW's and MATLAB's filter functions. In Example 6.2 of the textbook, we are given a DSP system described by the difference equation:

$$y[n] = 2x[n] - 4x[n-1] - 0.5y[n-1] - y[n-2] \qquad \text{(Eq. 4)}$$

with the initial conditions $y[-2] = 1$, $y[-1] = 0$, $x[-1] = -1$. We are told the input $x[n] = 0.8^n u[n]$ and asked to compute the response $y[n]$ for 20 samples.

The MATLAB function **filter()** can be used to perform the digital filtering in conjunction with the function **filtic()**, which obtains initial states from initial conditions in the difference equation. LabVIEW achieves the same result with its **IIR Filter with I.C.** function.


Reverse Coefficients: 0 | 1, 0.5, 1

Forward Coefficients: 0 | 2, -4

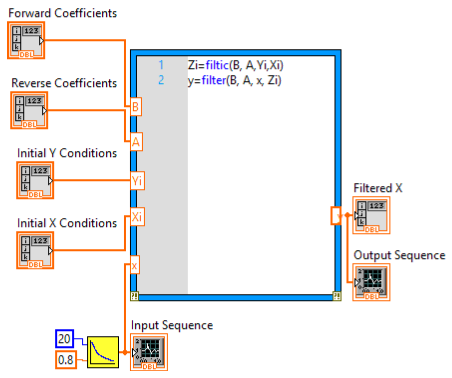
Filtered X: 0 | 5, -4.9, -4.47, 5.599, 0.4417, -6.80289, 2.17331, 5.08709, -5.22017

Initial Y Conditions: 0 | 0, 1, 0

Initial X Conditions: 0 | -1

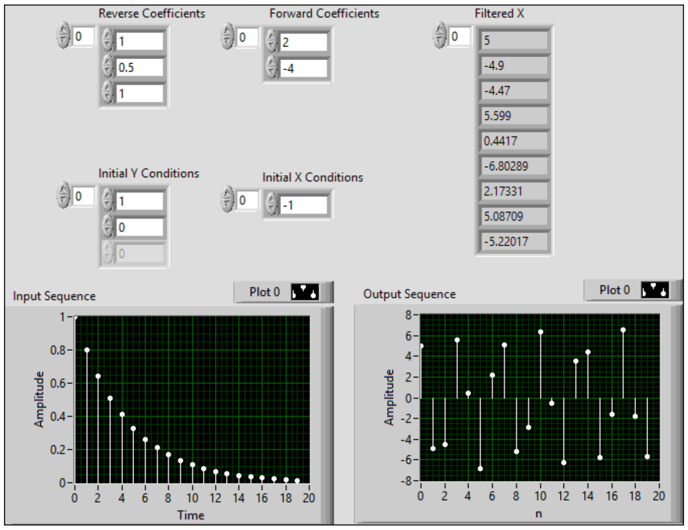
Figure 5: Front panel of the VI implementing Example 6.2 of the textbook with the MathScript node.

Figures 5 and 6 depict the front panel and block diagram of the VI which implements this example with MATLAB commands. The block supplying the input sequence is a subVI generating an exponentially decaying sequence $\alpha^n u[n]$ whose length is set to 20 with $\alpha = 0.8$.



```
Zi=filtic(B, A,Yi,Xi)
y=filter(B, A, x, Zi)
```

Figure 6: MathScript node implementing Example 6.2 of the textbook.



Reverse Coefficients: 1, 0.5, 1

Forward Coefficients: 2, -4

Filtered X: 5, -4.9, -4.47, 5.599, 0.4417, -6.80289, 2.17331, 5.08709, -5.22017

Initial Y Conditions: 1, 0, 0

Initial X Conditions: -1

Input Sequence

Output Sequence

Figure 7: Front panel of the VI implementing Example 6.2 of the textbook with LabVIEW's **IIR with I.C.** function.

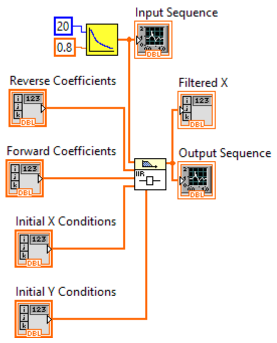


Figure 8: LabVIEW's **IIR with I.C.** function implementing Example 6.2 of the textbook.

Figures 7 and 8 depict the front panel and the block diagram of the VI which implements the same example utilizing LabVIEW's **IIR Filter with I.C.** function. Comparison of the two front panels will reveal that MATLAB's **filtic()** function requires it initial condition inputs in the reverse order of LabVIEW's **IIR Filter with I.C.** function.

Discussion & Conclusions

The engineering department at Geneva College offers an ABET-accredited engineering major with civil, environmental, chemical, mechanical, electrical, computer, biomedical engineering concentrations well as serving a non-accredited chemical engineering major provided by the Department of Chemistry. Digital Signal Processing is a senior/junior level elective for electrical, and computer engineering students. It is also cross-listed as BME 440, an elective for biomedical engineering students. This offering of the course was taken by 3 computer, 1 biomedical, 1 double major electrical and mechanical, and 5 electrical engineering students. The student surveys conducted at the end of the semester showed that the course with the new textbook had been regarded more favorably than its previous offering. One student noted that the instructor was having fun with the demonstrations. Several students remarked that the labs, based on the exercises in the textbook, were helpful in reinforcing the theory. This kind of reaction from the students reinforced my conviction that the change of the textbook was a beneficial move.

As expected, some students preferred the textual interface of MATLAB to the block diagrams of LabVIEW. These students incorporated Mathscript nodes in their LabVIEW code, using the graphical interface for input/output or simple calculations when left free to do so. One student who had prior experience in MATLAB used the technique of the example depicted in Figure 3 and 4, completely bypassing LabVIEW utilities in exercises that did not dictate the solution tools.

In most examples, LabVIEW's signal processing tools and Mathscript node delivered results which did not deviate under cursory observation. The most noticeable difference was observed in window functions and related FIR windowed filter designs. Although both results met the specs, slight differences in frequency responses were noticeable in the plots. Further study in this behavior is warranted.

Based on the student reaction in end-of-the-semester survey and behavior in open ended assignments, offering both MATLAB and LabVIEW tools in the same course has merit and I intend to repeat this approach next time I offer DSP.

References:

[1] Oppenheim, A. V., Schafer, R.W., *Digital Signal Processing 1st Edition*, Upper Saddle River, NJ: Prentice Hall (1975)
[2] Orfanidis, S. J., *Introduction to Signal Processing*, Upper Saddle River, NJ: Prentice Hall (1996).
[3] Mitra S. K., *Digital Signal Processing 4th Edition*, McGraw Hill Higher Education (2011)
[4] Tan, L., Jian, J., *Digital Signal Processing Fundamentals and Applications*, Boston, Academic Press (2013)

[5] Viss, M., Tanyel, M., “From Block Diagrams to Graphical Programs in DSP”, *2001 ASEE Annual Conference and Exposition Proceedings*, Albuquerque, N.M., June 24-27 2001

[6] Tanyel, M., “Enhancing the DSP Toolkit of LabVIEW”, *2002 ASEE Annual Conference and Exposition Proceedings*, Montréal, QC, June 16-19 2002

[7] Tanyel, M., “Putting Bells & Whistles on DSP Toolkit of LabVIEW”, *2011 ASEE Annual Conference and Exposition Proceedings*, Vancouver, BC, June 26-29 2011