Signal Processing Laboratories

D.G. Bailey and K.A. Mercer

Institute of Information Sciences and Technology, Massey University,
Private Bag 11222, Palmerston North

E-mail: D.G.Bailey@massey.ac.nz

Abstract: The philosophy behind teaching the 3rd year signal processing paper is to concentrate on the concepts rather than detailed mathematics. In support of this, the laboratories place an emphasis on demonstrating and reinforcing basic principles. There are four sets of experiments, closely coupled with the four modules of the theory. Each set of experiments has MATLAB based simulations, and a practical component.

Keywords: Signal Processing, DSP, aliasing, digital filters, MATLAB

1. INTRODUCTION

Many signal processing texts concentrate on the mathematical basis of signal processing. While this is important, it is essential that the students develop an intuitive understanding of the underlying concepts to be able to apply the appropriate mathematics correctly.

The philosophy taken within the 3rd year signal processing paper is to concentrate on signal processing concepts, rather than rigorous mathematical detail. Once the concepts are understood, the necessary mathematics can be applied intelligently to signal processing problems. To enable the students to build the necessary conceptual framework, the paper has a strong laboratory component, which places an emphasis on demonstrating and reinforcing basic principles.

1.1 Structure of Paper

Most of the students coming into the paper have covered basic electronics and calculus. As a result, they are familiar with the concepts of signals, frequency, and filters. In their mathematics, they have covered Fourier series, and Laplace transforms in the context of solving differential equations. One of the goals in the early part of the course is to relate these techniques to signal processing.

The course is split into four modules, with each module consisting of 10 lectures, 3 tutorials and two 3-hour laboratory sessions. Each module builds on the material and concepts from the previous modules. The text for the paper was Mitra's [1] "Digital Signal Processing: A Computer-Based Approach". This provided an excellent mathematical and theoretical basis for the paper, primarily for the last 3 modules, while the lectures concentrated on developing the conceptual framework. An outline of the topics covered by the paper is as follows:

1.1.1 Analogue Signal Processing. The main concept covered in the first module is that of linear time-invariant systems. It is shown how an LTI system allows the decomposition of its input signal into basic component signals, and the output is reconstructed from the outputs from each of the basic input signals. Important decompositions covered are: as impulses, leading to the concepts of impulse response and the convolution integral; as sinusoids, leading to the Fourier series, Fourier transform, and frequency response; and as complex exponentials, leading to the Laplace transform, system function, and pole-zero representation. In the laboratory, the relationship between these different decompositions and representations is investigated.

1.1.2 Discrete Time Systems. The second module looks at how signals may be converted between the

analog and digital domains, and introduces the basic operations used by discrete time systems. A significant section of this module covers sampling, aliasing, and the associated anti-alias and reconstruction filters. The principles of linear time invariant systems are covered again from a discrete time perspective as discrete convolution, discrete Fourier transform, and z-transform.

1.1.3 Digital Filters. Building on the previous module, this module covers filter design from two perspectives. Low order filters are designed directly from their pole-zero representations in the z-domain. The other approach proceeds from the desired frequency response using traditional methods: windowed Fourier series, frequency sampling, and optimisation methods for FIR design, and impulse invariant and bilinear transformations for IIR design. Important properties and implementations of both FIR and IIR filters are also discussed, as are important classes of filters such as linear-phase, all-pass, and comb filters. Transformation techniques are introduced for converting a prototype low-pass filter into low-pass, high-pass, band-pass, and band-stop forms.

1.1.4 Implementation and Applications. The final module looks at some of the important considerations in implementing a signal processing algorithm on a DSP. Important considerations are realisability (eliminating delay free loops) and the effects of quantisation on both the signal and on filter coefficients. At this stage, random signals and noise are introduced. The ideas behind the fast Fourier transform are described, the short-time Fourier transform is discussed. The tradeoff between resolution in time and frequency leads to an introduction to wavelets and multirate processing.

2. EXPERIMENTAL WORK

The experimental work for this paper closely parallels the lecture material. The overall aim is to encourage the students to develop an appropriate understanding of the basic principles through practical experience. While Mitra publishes a companion laboratory manual [2] that gives a broad range of MATLAB experiments, the focus of these is less on developing the concepts, and is completely MATLAB based. Therefore a separate set of experiments was developed for this course.

Within the practical work for this paper, there are four sets of experiments, corresponding to each of the four modules. Each set of experiments has two components: a MATLAB based simulation or design exercise and measurements made on a physical system. The class was split into two for each laboratory session, with half of the class working in a computer lab with MATLAB and the other half working in the electronics lab. The following week, the two groups were swapped. Time and equipment constraints meant that within each laboratory, students had to work in groups of three or four. At the end of each module, the students submitted a group report.

A description of the experimental work for each of the modules will now be described:

2.1 Analogue Signal Processing

The aim of the first module laboratories was to reinforce the concept of different signal decompositions by investigating the relationships between the system function, frequency response, impulse response, and step response.

In the electronics laboratory, students were given a band-pass amplifier with the characteristics shown in figure 1. Students then measured the frequency response, impulse response, and step response of the amplifier.

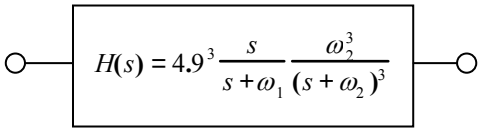

Figure 1: System response of the band-pass analogue amplifier investigated; ω_1=6667, and ω_2=66667.

Using MATLAB, the students took the system response provided, and plotted the theoretical frequency response. The impulse response was then calculated by taking the inverse Fourier transform of the frequency response. The impulse response was again calculated directly from the system function via partial fraction expansion and inverse Laplace transform. Finally, the step response was calculated by convolving the impulse response with a unit step.

In their group report, the students compared their practical measurements with their simulated results,

and discussed the relationships between the different measurements made.

2.2 Sampling and Aliasing

The focus of the second module laboratories was on sampling and reconstruction. The approach taken was to get the students to both listen to audio signals and view them on an oscilloscope, relating what they observed and heard to the original signal.

In the electronics laboratory, the students investigated the sample-and-hold circuit shown in figure 2. The first stage implements a track-and-hold, and with a square wave clock, it requires a second stage to emulate reconstruction by a zero-order hold. After constructing the circuit on breadboard it was used with a sine wave input, and with both a speaker and oscilloscope on the output. This enabled investigation of aliasing, and also signal reconstruction using a zero-order hold. Both aliasing, and incomplete removal of the spectral images by the zero-order hold were clearly evident.

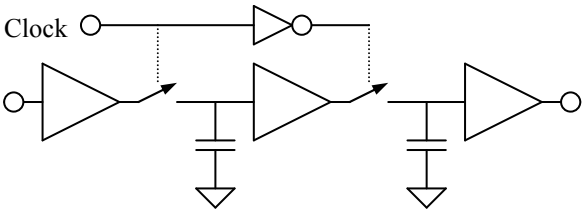

Figure 2: Sample-and-hold circuit used in module 2.

Within MATLAB, use was made of the ability to play sound samples through the PC's multimedia card. The goal was to test the student's understanding of aliasing by investigating progressively more complex signals. While students initially had a reasonable understanding of what happened when a sine-wave was sampled, a square-wave with its high frequency harmonics initially caused confusion (see figure 3). There were considerably more frequency components than the students were expecting, and the position of the components changed significantly as the relationship between the square-wave frequency and sample frequency was varied.

Finally, students were able to record a one second speech segment, and resample and replay the sound at different sample rates. By considering the intelligibility of the resampled speech, the students were able to investigate the effect of aliasing on complex signals.

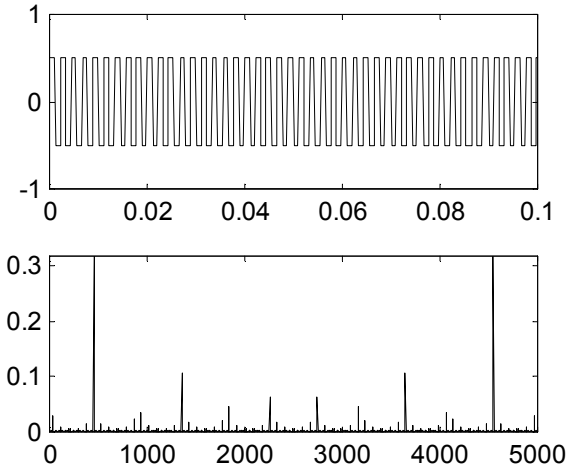

Figure 3: Screen shot of a 451 Hz square wave sampled at 5 kHz, and its frequency spectrum

The MATLAB m-files provided to the students for this module are listed in Appendix 1.

2.3 Digital Filter Design

The purpose of the third module laboratories was to give some experience in designing and implementing digital filters, and to provide an appreciation for the advantages and disadvantages of some of the different types of filters. All of the students started by using MATLAB to design both FIR and IIR filters to meet the following specifications:

Sample rate: 8 kHz
Pass band edge: 1000 Hz
Maximum pass band ripple: 1 dB
Stop band edge: 1500 Hz
Minimum stop band attenuation: 20 dB

After designing the filters, the coefficients were quantised and the filters implemented using a TMS320C50 DSP. Students were provided with the code implement the filters (in appendix 2) and only had to enter their scaled and quantised coefficients.

For simplicity, both FIR and IIR filters were implemented using a direct form representation (see figures 4 and 5 for the filter structures). While this is appropriate for FIR filters, a direct form implementation of IIR filters can give problems with stability as a result of both overflow, and coefficient quantisation. With the low order designs from this laboratory, quantisation effects are not a serious restriction. However, overflow can make the IIR filter unstable with large amplitude signals.

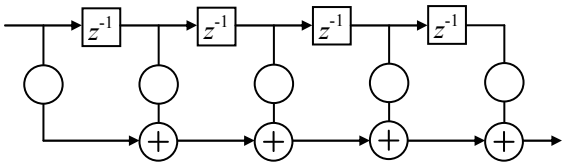

Figure 4: Direct form FIR filter representation.

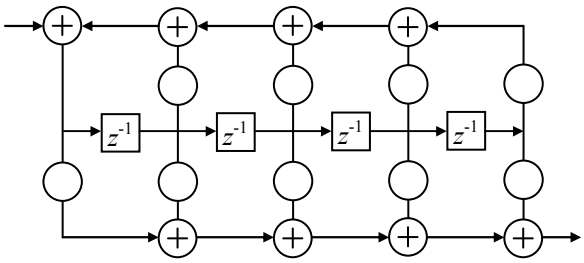

Figure 5: Direct form IIR filter representation.

In their report, students compare the frequency response of their DSP-based filters with the ideal response calculated in MATLAB.

2.4 Project

The project associated with the fourth module is less directly related to the lecture material. The aim of the project is to integrate a range of concepts as they design and test a more complex digital signal processing system.

For their project students had to design a signal processing system for decoding numbers entered on a touch-tone phone. Although their design is based on MATLAB, consideration must be given to how the final design would be implemented on a DSP. Some of the issues covered in the design are:

- Coping with variable strength signals (AGC)
- Designing filters for each of the tones
- Detecting the presence or absence of a tone (envelope detecting)
- Decoding the tones to give the digit dialled
- Distinguishing valid tones, voice, and noise

In their report, students had to describe their system, and present the response of their system to real sound samples recorded through a microphone.

3. DISCUSSION AND CONCLUSIONS

In the early modules, students were overwhelmed by the number of concepts and techniques they had to learn. This was exacerbated by the rapid pace at which material was covered as a result of semesterisation. The focus on basic signal processing principles and concepts rather than mathematical details was crucial to the overall understanding of the material. In this regard, the structure of the laboratories was invaluable in reinforcing the concepts covered during lectures.

In the module 1 laboratories, students were encouraged when they obtained virtually identical results from the different approaches followed. Some students had difficulty with direct measurement of the impulse response with the amplifier saturating giving a distorted output. With the peak of the unit impulse response just below $2x10^6$ Volts, this necessitated using a very short (in both time and amplitude) pulse, and scaling the results. Once discovered, this problem was easily fixed by reducing the length of the impulse and repeating the measurement.

The module 2 MATLAB laboratory thoroughly demonstrated the spectral folding that results from sampling. The use of a square wave meant that students had to know exactly what was happening to explain the amplitude and position of every spectral peak. In the electronics laboratory, there was still some confusion between aliasing and the spectral images that had not been completely removed by the zero-order hold. Although the cause of the two phenomena is the same, it was not clear that the higher frequency images could be removed with an appropriate low pass filter.

Students generally obtained good results for their filter design, provided they kept the input amplitude sufficiently low to avoid overflow instabilities with the IIR filter. In the stop band, the nulls were slightly shifted because the filter clock was at 7.95kHz rather than the 8kHz the students used for the design. The practical experience in designing and testing their digital filters solidly reinforced the lecture materials.

The initial goal for the project was to have the students implement and test their designs on the DSPs. However, after the first laboratory session it became clear that this would not be practical in the time available so the scope of the project was altered to completing the project in MATLAB and discussing what a DSP implementation would involve in their report. It was encouraging to see several groups use multirate techniques to reduce the amount of processing required. One of the most valuable

practical lessons learned was the tradeoff between frequency and time resolution. Initially some groups used very narrow band, high Q filters to select the tones. While this gave excellent frequency discrimination, the filter output had very slow rise and fall times and the amplitude response was less than that of a wider bandwidth filter.

Overall, the students have found the approach taken with the laboratories to be invaluable for reinforcing the important concepts. Several times during the laboratory sessions students could be heard exclaiming "Ah so that's why you do that!" or similar. Once the basic principles and concepts were understood, the correct application of the appropriate mathematics invariably followed.

4. ACKNOWLEDGEMENTS

The authors would like to acknowledge the contributions of Professor Bob Hodgson in teaching the sections on discrete time convolution, sampling and reconstruction, random signals and noise.

5. REFERENCES

[1] S.K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, McGraw Hill, International Edition, Singapore, 1998.

[2] S.K. Mitra, *Signal Processing Laboratory using MATLAB*, McGraw Hill, International Edition, Singapore, 2000.

APPENDIX 1: MATLAB CODE

MatLAB m-files for module 2, investigating sampling and aliasing.

A1.1 Note.m

```
function note( pitch, rate );
% NOTE( frequency, rate );
%
% Plays 1 second of a sine wave at the specified
% sample rate.
% frequency  is the frequency of the note (Hz)
% rate       is the sample rate used (Hz)
% The first 0.1 second of the note is plotted,
% along with the frequency spectrum of the note.
%
% See also  SQUARE
t=[0:rate];
n=sin(2*pi*pitch/rate*t);  % Get the sine sample
if rate < 5000             % Upsample to approx
   ups=round(11025/rate);  % 11kHz
   p=interp( n, ups );
else
   ups=1;
   p=n;
end
wavplay( p, rate*ups );    % Play the sound
tp=[0:length(p)-1];

subplot(2,1,1);
plot(tp/rate/ups, p );     % Plot first 0.1 s
axis([0,0.1,-1,1]);
subplot(2,1,2);
frq=fft(n);                % Get spectrum
frq=abs(frq)/rate;
plot(t, frq);
axis([0,rate,0,max(frq)]);
```

A1.2 Square.m

```
function square( pitch, rate );
% SQUARE( frequency, rate );
%
% Plays one second of a square wave signal at the
% specified sample rate.
% frequency   is the frequency of the note (Hz)
% rate        is the sample rate used (Hz)
% The first 0.1 second of the note is plotted,
% along with the frequency spectrum of the note.
%
% See also  NOTE
t=[0:rate-1];              % Create a square wave
n=(sin(2*pi*pitch/rate*t+.001)>0)-0.5;
if rate < 100              % Upsample to approx
   ups=round(11025/rate);  %11 kHz
   p=interp( n, ups );
else
   ups=1;
   p=n;
end
wavplay( p, rate*ups );    % Play the sound
tp=[0:length(p)-1];

subplot(2,1,1);
plot(tp/rate/ups, p );     % Plot the first 0.1 s
axis([0,0.1,-1,1]);
subplot(2,1,2);
frq=abs(fft(n))/rate;      % Get the spectrum
plot(t, frq);
axis([0,rate,0,max(frq)]);
```

A1.3 Record.m

```
function record;
% RECORD;
%
% Records 1 second of input through the microphone
% and scales it to give a normalised signal.
% The signal is sampled at 22050 Hz, and then
% replayed immediately at the same frequency.
%
% To play back at a different sample rate, see
% SAMPLE
global voice;
voice=wavrecord(22050,22050); % Record 1 second
mn=min(voice);
mx=max(voice);
voice=voice*(2/(mx-mn));   % Autoscale it
wavplay(voice,22050);      % Replay the sound
```

A1.4 Sample.m

```
function sample( rate );
% SAMPLE( rate );
%
% Resamples and replays the previously recorded
% sound signal using a lower sample rate.
%
```

```
% rate   is the desired playback sample rate (Hz).
%        The sample rate actually used may be
%        slightly different because it will be an
%        integer submultiple of the original sample
%        rate of 22050 Hz.
%
% See also  RECORD
global voice;
sr = round( 22050 / rate );% Integer change
if sr < 1
   return;          % Can't go to higher freq.
end
samplerate=22050/sr;       % True sample rate

v = voice(1:sr:22050);     % Downsample the signal
t=[0:length(v)-1];
wavplay( v, samplerate );  % Play the sound

subplot(2,1,1);
plot(t/length(v),v);       % Plot the signal
subplot(2,1,2);
frq=abs(fft(v))/length(v); % and its spectrum
plot(t*(samplerate/length(v)),frq);
axis([0,4000,0,max(frq)]);
```

APPENDIX 2: DSP FILTER CODE

The students tested their filters on the TMS320C50 evaluation kits using the following code. In the interests of space, the initialisation code has been omitted. The FIR and IIR filters both use direct form structures, with a block of memory reserved in the data space for storing the internal filter state. The filters are called with the input data in the accumulator, and return with the filtered output in the accumulator. One or other of these subroutines is called by the code that reads the samples from the analog interface chip. After the filter subroutine returns, a first order sinc correction filter is applied, and the output written to the analog interface chip.

Contact the authors directly if you wish to obtain an electronic copy of the complete programs.

```
       .ds    0F00h  ; Data space
taps   .word  0      ; Taps of shift register

       .ps    0A00h  ; Program space
;================================================
; For FIR filter, multiply all coefficients by 2^15
; and round. The FIRSC is 15 (need to shift by 2^15
; to undo the scaling). Then set the filter order
; and provide the coefficients in memory, replacing
; those below. The filter order is 1 less than the
; number of coefficients. The coefficients should
; be entered in reverse order. With a linear phase
; filter which is symmetrical this shouldn't
; matter.  For example a 4 point moving average
; filter (3rd order):
; H(z) = 0.25 + 0.25z^-1 + 0.25z^-2 + 0.25z^-3
; Scale by 2^15 to give coefficients
; FIRCOEF:  8192, 8192, 8192, 8192
;------------------------------------------------
FIRORD .set   3      ; Filter order (# coeff - 1)
FIRSC  .set   15     ; Scaling of the coefficients
FIRCOEF              ; Entered in reverse order
       .word  8192,8192,8192,8192

;------------------------------------------------
FIR    lar    AR0,#taps     ; Set AR0 to filter taps
       mar    *,AR0         ; Select AR0
       sacl   *             ; Store sample at start
       adrk   #FIRORD       ; Point AR0 to the end
       zap
       rpt    #FIRORD       ; Apply filter, moving
       macd   FIRCOEF,*-    ; data it filters
       apac                 ; Last sum
       bsar   FIRSC         ; Scale to give output
       ret

;================================================
; IIR filters are harder to set up - it is
; important to make sure that the feedback section
; doesn't overload the filter taps. Check the
; maximum gain of the feedback section on its own,
; and scale the input by the nearest power of 2 to
; minimise overflow problems.
; Need to enter the filter order and the numerator
; and denominator coefficients. The numerator
; coefficients must be multiplied by the scale
; factor to compensate for the scaling of filter
; taps. All the denominator coefficients (apart
; from the leading 1) must be made negative to give
; negative feedback.
; For example a 4th order filter:
; num = 0.1417 + 0.3001z^-1 + 0.4120z^-2
;         + 0.3001z^-3 + 0.1417z^-4
; den = 1.0000 - 0.5095z^-1 + 1.1505z^-2
;         - 0.4339z^-3 + 0.2588z^-4
; Denominator on its own has maximum gain of 7.6 so
; the input should be divided input by 8 to avoid
; internal overflow problems. Maximum denominator
; is 1.15, so scale by 2^14 to give max resolution
; to denominator coefficients.
; den => 16384, -8347, 18850, -7110, 4241
; Next step is to divide input by 8 by scaling the
; first coefficient, and to change the sign of the
; remaining filter coefficients
; IIRDEN: 2048, 8347, -18850, 7110, -4241
; Numerator needs to by multiplied by 8 to
; compensate for the input scaling
; num => 1.1772, 2.4009, 3.2959, 2.4009, 1.1772
; Max value is 3.3, so scale by 2^13 to give
; maximum resolution to the numerator coefficients
; IIRNUM: 9643, 19668, 27000, 19668, 9643
;------------------------------------------------
IIRORD .set   4      ; Filter order (# coeff - 1)
DENSC  .set   14     ; Scaling for the denominator
IIRDEN               ; Denominator in normal order
       .word  2048, 8347, -18850, 7110, -4241
NUMSC  .set   13     ; Scaling for the numerator
IIRNUM               ; Numerator, in reverse order
       .word  9643, 19668, 27000, 19668, 9643

;------------------------------------------------
IIR    lar    AR0,#taps     ; Set AR0 to filter taps
       mar    *,AR0         ; Select AR0
       sacl   *             ; Put sample in place of
       zap                  ; 0th tap to simplify
       rpt    #IIRORD
       mac    IIRDEN,*+     ; Apply denom filter
       apac                 ; Last sum
       bsar   DENSC         ; Divide by den scale
       sacl   taps          ; Put O/P from feedback
                            ; into 0th tap for num
       mar    *-,AR0        ; Select the last tap
       zap
       rpt    #IIRORD       ; Apply numerator filter
       macd   IIRNUM,*-     ; moving data as it goes
       apac
       bsar   NUMSC         ; Divide by num. scale
       ret
```