Lab 07 - Reverb Analysis and Synthesis

Ron Guglielmone

INTRODUCTION

The supplied plugin, 'Reverb', was modified to incorporate new filters-- these modify the decay time and output equalization.

PROBLEM 1: PART A

Give the expression of an analog, first-order, low-pass filter, with cut-off ρ (radian frequency), and gain g_1 at ω = 0.

$$H(s) = \frac{\frac{s}{\rho} + g_1}{\frac{s}{\rho} + 1}$$

Give the poles and zeros of this low-pass filter.

$$\text{Pole, } s_p = -\sqrt{\frac{1}{g_1}} \text{ and Zero, } s_z = -\sqrt{g_1}$$

Sketch bode plots for magnitude and phase and define ρ in terms of g_1.

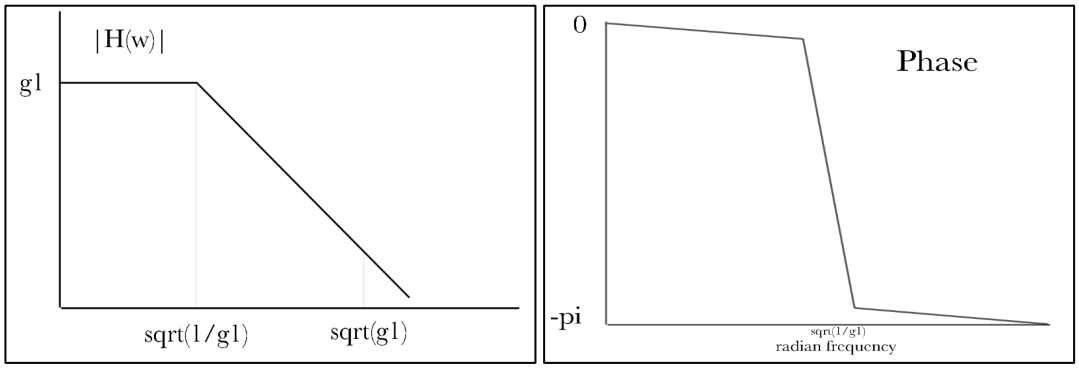

Figure 1.1, magnitude and phase response sketch for low pass filter.

$$\rho = \sqrt{\frac{1}{g_1}}$$

PROBLEM 1: PART B

Give the expression of an analog, first-order, high-pass filter, with cut-off ρ (radian frequency), and gain g_1 at ω = 0.

$$H(s) = \frac{\frac{g_1 s}{\rho} + 1}{\frac{s}{\rho} + 1}$$

Give the poles and zeros of this high-pass filter.

$$\text{Zero } s_z = -\sqrt{\frac{1}{g_1}} \text{ and Pole } s_p = -\sqrt{g_1}$$

Sketch bode plots for magnitude and phase and define ρ in terms of g_1.

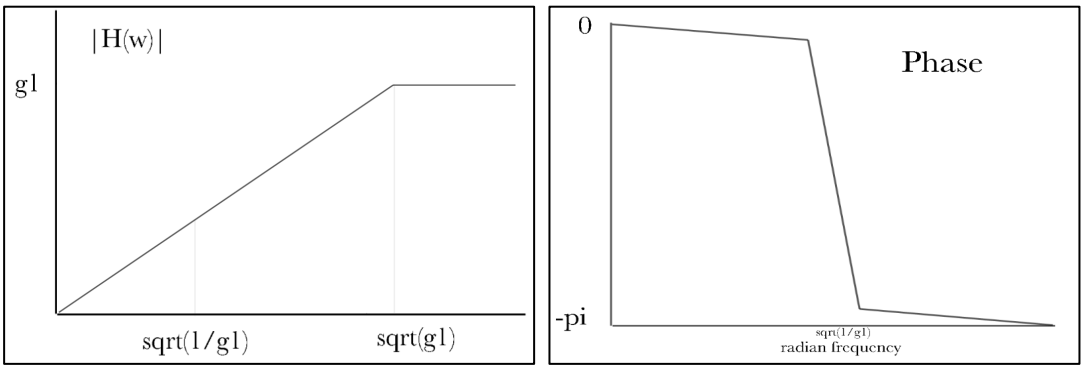

Figure 1.2, magnitude and phase response sketch for high pass filter.

$$\rho = \sqrt{g_1}$$

PROBLEM 1: PART C

Give the expression of an analog, first-order, shelving filter, with cut-off ρ (radian frequency), gain l_0 at $\omega = 0$, and gain l_∞ at $\omega = \infty$.

$$H(s) = l_\infty \frac{\frac{s}{\rho} + l_0}{\frac{s}{\rho} + 1}$$

Give the poles and zeros of this high-pass filter.

$$\text{Pole } s_p = -\sqrt{\frac{l_\infty}{l_0}} \text{ and Zero } s_z = -\sqrt{\frac{l_0}{l_\infty}}$$

Sketch bode plots for magnitude and phase.

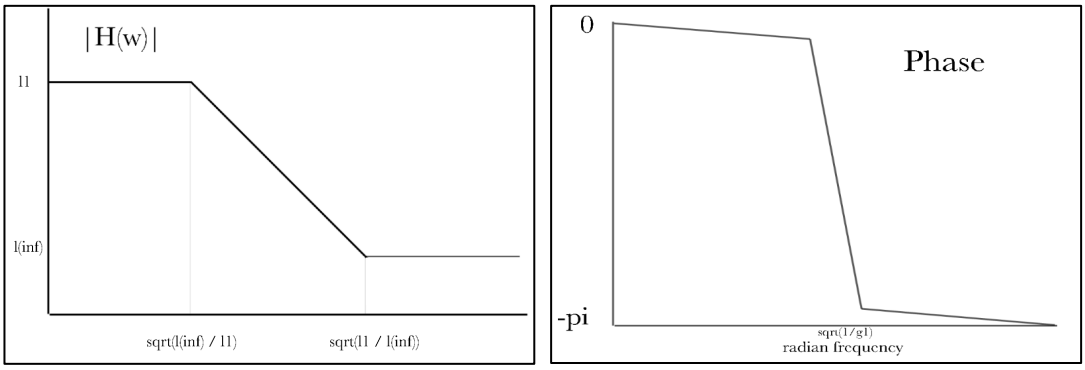

Figure 1.3, magnitude and phase response sketch for shelf filter.

(All of the above answers were derived from the course notes, page 250).

PROBLEM 1: PART D

Give expressions of l_0, l_∞ and ρ as functions of τ_0, τ_∞, ω_c and T in order to achieve a feedback loop with decay time τ_0 at low frequencies, and τ_∞ at high frequencies.

From the course reader, we know

$$T_{60}(\omega) = -\frac{60 \cdot \tau / f_s}{20 \log_{10} |\alpha(\omega)|}$$

Where

$$|\alpha_i(\omega)| = \exp\{\ln(0.001)\, \tau / [f_s \cdot T_{60}(\omega)]\}$$

PROBLEM 1: PART E

The following modifications to the code were made (Figure 1.4).

```
double gammaH = pow(10, -3*theLength / fs / t60high);
double gammaL = pow(10, -3*theLength / fs / t60low);

b0 = gammaL;
b1 = gammaH / transition;
a1 = 1.0 / transition;

norm=1.0 + a1*2*fs;
b0z= 1.0*(b0 + b1*2*fs)/norm;
b1z= 1.0*(b0 - b1*2*fs)/norm;
a1z= 1.0*(1 - a1*2*fs)/norm;

*(pcoefs) = b0z;
*(pcoefs+1) = b1z;
*(pcoefs+2) = a1z;
```

Figure 1.4, code modifications for Part E.

PROBLEM 1: PART F

The plugin was run with high frequency delay of 0.5 sec and low frequency delay of 2 seconds. An impulse response measurement was conducted. The spectrograph of the impulse response is presented below in Figure 1.5

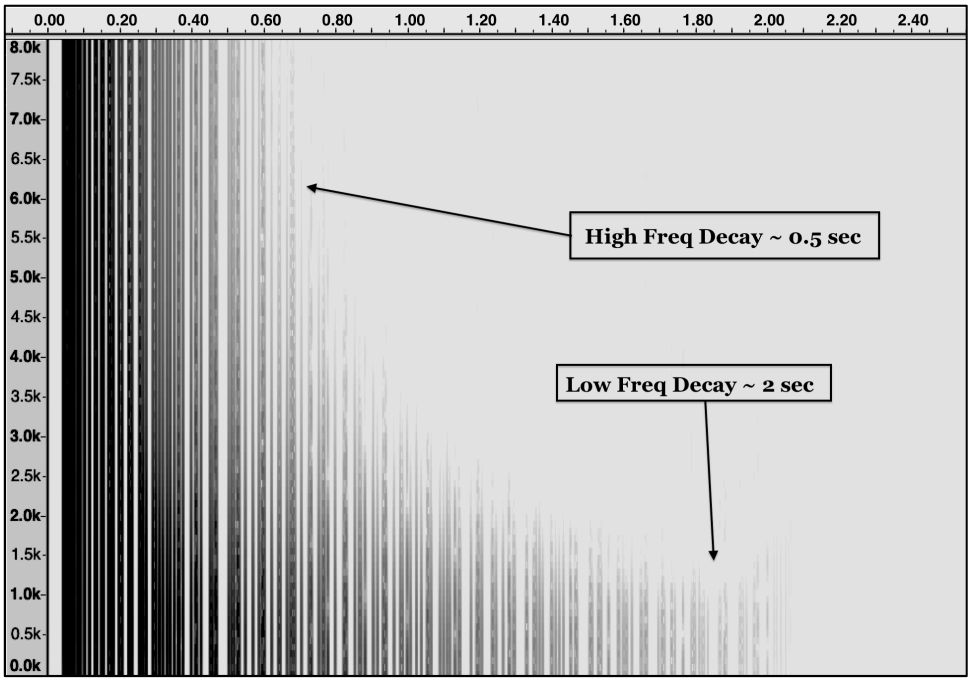

Figure 1.5, spectrogram of plugin impulse response.

PROBLEM 1: PART G

The default identity matrix was modified with the following mixing matrix (Figure 1.6).

```
// Mixing matrix:
    {-0.0237,   -0.0084,    0.0035,    0.0111,    0.0157,    0.0019, -0.0237,   -0.0084,    0.0035,    0.0111,    0.0157,    0.0019,},
    {-0.0083,   -0.0022,    0.0163,    0.0051,    0.0072,    0.0057, -0.0083,   -0.0022,    0.0163,    0.0051,    0.0072,    0.0057,},
    {0.0417,    0.0314,    0.0368,   -0.0071,   -0.0033,   -0.0068,    0.0417,    0.0314,    0.0368,   -0.0071,   -0.0033,   -0.0068,},
    {-0.0697,   -0.0304,   -0.0524,    0.0052,    0.0174,   -0.0026, -0.0697,   -0.0304,   -0.0524,    0.0052,    0.0174,   -0.0026,},
    {-0.1913,   -0.0466,   -0.0260,    0.0260,    0.0336,    0.0027, -0.1913,   -0.0466,   -0.0260,    0.0260,    0.0336,    0.0027,},
    {0.0734,   -0.0268,    0.0119,    0.0177,    0.0146,    0.0039,    0.0734,   -0.0268,    0.0119,    0.0177,    0.0146,    0.0039,},
    {0.9082,    0.2591,    0.2211,   -0.0226,   -0.0492,   -0.0350,    0.9082,    0.2591,    0.2211,   -0.0226,   -0.0492,   -0.0350,},
    {-0.3336,    0.5559,    0.7424,    0.1570,    0.0028,   -0.0193, -0.3336,    0.5559,    0.7424,    0.1570,    0.0028,   -0.0193,},
    {0.0988,   -0.7301,    0.4967,    0.4543,   -0.0228,    0.0377,    0.0988,   -0.7301,    0.4967,    0.4543,   -0.0228,    0.0377,},
    {0.0554,    0.2776,   -0.3424,    0.7932,    0.2608,    0.3236,    0.0554,    0.2776,   -0.3424,    0.7932,    0.2608,    0.3236,},
    {0.0303,   -0.0866,    0.1740,   -0.3716,    0.5341,    0.7333,    0.0303,   -0.0866,    0.1740,   -0.3716,    0.5341,    0.7333,},
    {-0.0304,    0.0360,   -0.0225,    0.0013,   -0.8012,    0.5954, -0.0304,    0.0360,   -0.0225,    0.0013,   -0.8012,    0.5954,},
```

Figure 1.6, Mixing matrix code for blending delay lines.

We see several differences in the spectrograph (Figure 1.7).

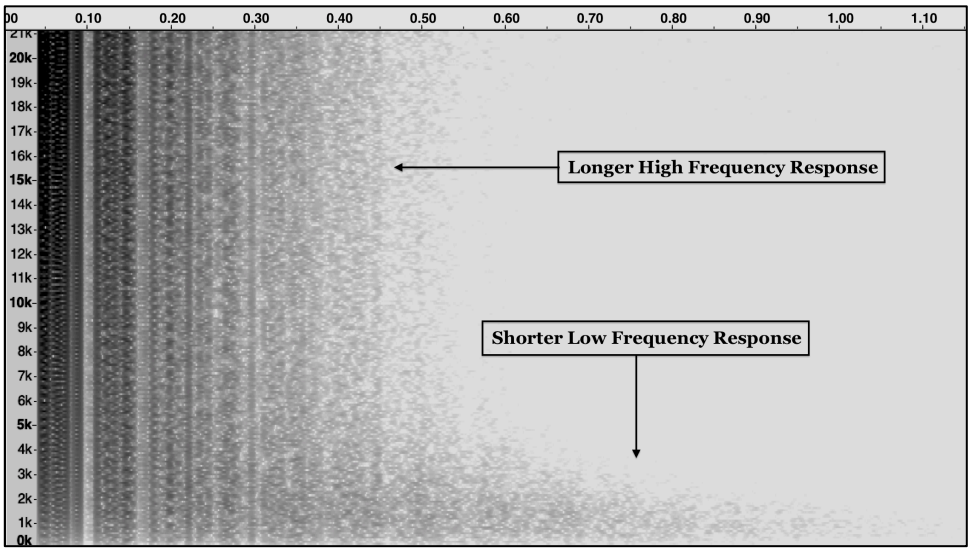

Figure 1.7, spectrograph of mixed feedback delay line IR.

First of all, the harsh lines have been diffused. Also, the decay times have been obscured with the low frequency decay shortening, and the high frequency decay lengthening.

PROBLEM 1: PART H

The following code was added in designParametric (Figure 1.8).

```
double b0, b1, b2, a0, a1, a2;
double acoefs[6];
double omega = 2*pi*center;
double omegaSq = omega*omega;

b0 = 1.0;
b1 = gain / (qval*omega);
b2 = 1.0 / (omegaSq);
a0 = 1.0;
a1 = 1.0 / (qval*omega);
a2 = 1.0 / (omegaSq);
```

Figure 1.8, designParametric function code.

And the bilinear transform was also attempted (Figure 1.9).

```
double T = 1.0 / fs;
double Tsq = T*T;

az0 = ((4*a2) + (2*T*a1) + (Tsq*a0));
az1 = ((-8*a2) + (2*Tsq*a0)) / az0;
az2 = ((4*a2) - (2*T*a1) + (Tsq*a0)) / az0;
bz0 = ((4*b2) + (2*b1*T) + (b0*Tsq)) / az0;
bz1 = ((-8*b2) + (2*b0*Tsq)) / az0;
bz2 = ((4*b2) - (2*b1*T) + (b0*Tsq)) / az0;

az0 = 1;
```

Figure 1.9, bilinear transform code.

But I'm not sure if these are working properly.

PROBLEM 1: PART I

I couldn't get my EQ working, and this part of the problem hinged on that...

But for $T_{60,}$ I'd have tried about 4 seconds. For the EQ, I'd have put a slight bump in the mid-range frequencies (around 800-1,200 Hz).

PROBLEM 2: PART A

The impulse response of a Sansui RA-700 spring reverb has been analyzed. Its spectrogram is shown below in Figure 2.1.

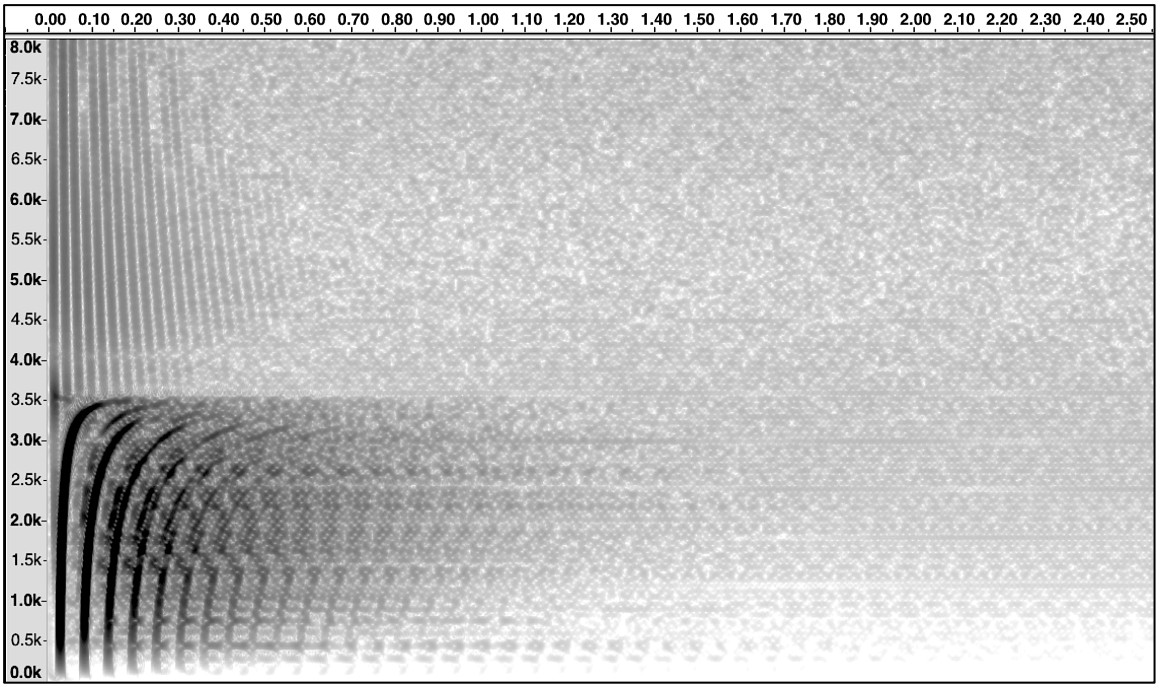

Figure 2.1, spectrogram of an impulse response from a spring reverb.

We can see from the spectrogram that ω_c, the asymptote of the feather-like features, is somewhere between 3.5 kHz and 3.6 kHz. These features are about 55 ms apart from each other, therefore $\tau_0 = 0.055$ seconds (Figure 2.2). (Note: Jonathan Abel's paper on this topic mentions a frequency of 3625 Hz and delay of 52.25 ms, which is quite close to what I've observed independently).

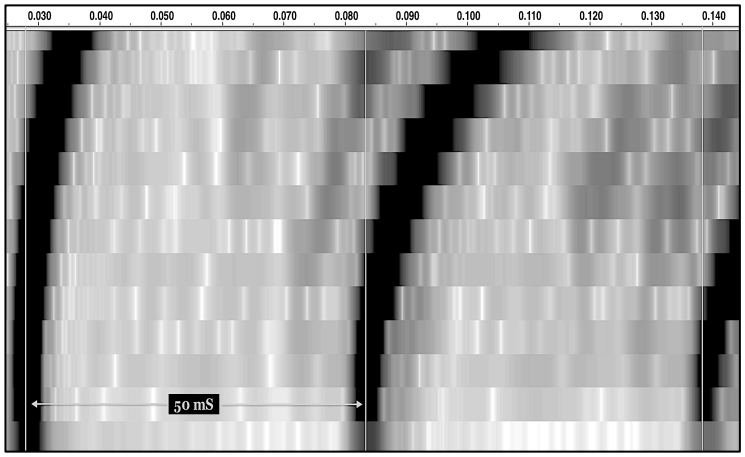

Figure 2.2, spacing between features, τ_0.

PROBLEM 2: PART B

The following method was attempted to solve for frequency dependent T_{60} values, but it has a bug (Figure 2.3).

```
for i = 1 : length(omegaM)

    % Define bounds for bandpass:
    center = omegaM(i);             % center of bin [Hz]
    top = center*2;                 % upper bound of bin
    top = (2*pi*top)/fs;            % ...normalized
    bottom = center/2;              % lower bound of bin
    bottom = (2*pi*bottom)/fs;      % ...normalized

    % Keep in bounds for butter():
    if (top >= 1); top = 0.999; end
    if (top <= 0); top = 0.001; end
    if (bottom >= 1); bottom = 0.999; end
    if (bottom <= 0); bottom = 0.001; end

    % Design band-pass using LP and HP:
    [b1, a1] = butter(2,bottom,'high');
    [b2, a2] = butter(2,top,'low');

    % Filter IR into 1/2 oct band:
    h1 = filter(b1,a1,h);           % high-passed IR
    h2 = filter(b2,a2,h1);          % LP and HP IR
    WN = 1000;                      % RMS Window [samp]
    OL = 500;                       % RMS Overlap [samp]
    ZP = 1;                         % Flag for ZP
    h3 = rms(h2,WN,OL,ZP);          % RMS of bin

    % Find T60 Value for bin:
    A0 = max(h);                    % initial value
    A60 = A0;                       % T60 amplitude
    Tval = A60/A0;                  % ratio for T60
    k = 1;                          % index for loop
    index = 1;
    index0 = 5;                     % t60 value index

    % Make sure index is in bounds:
    if (length(h3) > index0)

        % Find T60 index:
        index = index0;
        while (Tval > 0.001 && length(h3) < index)
            A60 = h3(index);
            Tval = A60/A0;
            index = index + 1;
        end

        T60Times(i) = index/fs;     % T60 in seconds
    end

    % If index is out of bounds, default:
    if (length(h3) < index0)
        T60Times(i) = 0;
    end
end
```

Figure 2.3, first attempt at calculating T_{60} times.

In 'A Modal Architecture for Artificial Reverberation with Application to Room Acoustics Modeling ', Jonathan Abel just uses a constant value of 1 second. So, if I can't resolve the bugs in the above script, I will also use a constant T_{60} value of 1 second.

PROBLEM 2: PART C

The following MATLAB script was written to calculate values of γ_m (Figure 2.4).

```
% Grab the signal:
[h, fs] = audioread('springRA700.wav');

% Build spring mode frequencies:
omegaCHz = 3555;                    % Frequency limit [Hz]
wCRadSec = (omegaCHz*(2*pi));       % Frequency [radians]
wCRadSam = (2*pi*omegaCHz)/fs;      % Frequency [rad/samp]
T0 = 0.055;                         % Time ripple [sec]
k0 = (2*pi)/(wCRadSec*T0);          % Spring const(ish)
numM = floor(pi/(2*k0));            % Number of modes
M = 1:1:numM;                       % Index vector
omegaM = omegaCHz.*sin(k0.*M);      % Mode frequencies

% Calculate gammas:

gammas = [];

for i = 1 : length(omegaM)

    % Define bounds for bandpass:
    center = omegaM(i);             % center of bin
    top = center*2;                 % upper bound of bin
    top = (2*pi*top)/fs;            % ...normalized
    bottom = center/2;              % lower bound of bin
    bottom = (2*pi*bottom)/fs;      % ...normalized

    % Keep in bounds for butter():
    if (top >= 1); top = 0.999; end
    if (top <= 0); top = 0.001; end
    if (bottom >= 1); bottom = 0.999; end
    if (bottom <= 0); bottom = 0.001; end

    % Design band-pass using LP and HP:
    [b1, a1] = butter(2,bottom,'high');
    [b2, a2] = butter(2,top,'low');

    % Filter IR into 1/2 oct band:
    h1 = filter(b1,a1,h);           % high-passed IR
    h2 = filter(b2,a2,h1);          % LP and HP IR
    WN = 1000;                      % RMS Window [samp]
    OL = 500;                       % RMS Overlap [samp]
    ZP = 1;                         % Flag for ZP
    h3 = rms(h2,WN,OL,ZP);          % RMS of bin
    hdB = 20*log10(h3);             % Decible IR
    l = floor(length(hdB)/2);       % Truncated IR leng
    hdB = hdB(100:l);               % Truncated IR
    l2 = length(hdB);               % New length
    t = 1:1:length(hdB);            % T for poly
    p = polyfit(t,hdB,1);           % Least sqrs.
    gammas(i) = db2mag(polyval(p,0));
```

Figure 2.4, MATLAB script for finding gammas.

My values for gamma seem much too small. Some sample values are shown in Figure 2.5 below. Perhaps this is the source of errors encountered in Part D(?).

```
gammas =

   1.0e-03 *

  Columns 1 through 6

    0.0568    0.1156    0.1703    0.2132    0.2447    0.2693

  Columns 7 through 12

    0.2913    0.3129    0.3352    0.3585    0.3825    0.4069

  Columns 13 through 18

    0.4314    0.4556    0.4792    0.5019    0.5235    0.5439

  Columns 19 through 24

    0.5629    0.5804    0.5964    0.6110    0.6241    0.6359

  Columns 25 through 30

    0.6463    0.6556    0.6637    0.6707    0.6768    0.6819
```

Figure 2.5, sample output of gamma values-- note all values are 1.0E-03.

PROBLEM 2: PART D

The following MATLAB code was written to synthesize the impulse response, but I couldn't get it to work (Figure 2.6). I suspect there was something wrong with my calculations of gamma that spilled over.

```
duration = 6;                  % IR duration in sec
t = 0:1:fs*duration;           % time vector

hmFinal = zeros(length(t),1)';    % Vec for final IR

for i = 1 : length(gammas)

hm = abs(gammas(i))* ...          % From J.A's paper
   exp((1i*omegaM(i) - 1).*t);

 hmFinal = hmFinal + real(hm);    % Sum them up

end
```

Figure 2.6, broken MATLAB code for IR synthesis.