The University of Sydney

Faculty of Architecture, Design and Planning

DESC 9115: Digital Audio Systems

Laboratory 2 (2011): Parametric Filters and Their Applications for Room Modes

Instructor:	William L. Martens
Tutor:	Luis A. Miranda J.
Student Name:	Renzo Arango
SID:	199912753
Date:	Monday 30th May 2011

Table of Contents

1. Introduction

As requested in the laboratory 2 brief, this report includes the functions developed for assignment 2, the help text included in the functions which are invoked when using the *help* Matlab command, and the help page which corresponds to each of these functions.

The help pages are included in the main body of this report. The functions and their help texts are included in Appendix A of this report.

Please note these functions are invoked from a main script. A help text is included in this script. As per the functions, both the main script and its help text, are included in Appendix A.

2. Help Pages

2.1. Function *roomcomp_RA_2*

Function which invokes equalization procedures for loudspeaker – room impulse response (see functions *parfiltid* and *parfilt*), and convolves equalized and unequalised responses with a “dry signal”.

2.1.1. Syntax

roomcomp_RA_1(freq,IR_soundfilename,DS_soundfilename,ES_soundfilename), where:

- *freq* is the array which contains the pole locations to be used in the equalization procedure.
- *IR_soundfilename* is the file name (without extension) of the WAV file which contains the measured loudspeaker – room impulse response and which will subject to the equalization.
- *DS_soundfilename* is the file name (without extension) of the WAV file which contains the “dry signal” (i.e. anechoic room recording) and which will be subject to the convolution with the equalized and unequalised impulse responses.
- *ES_soundfilename* is the WAV file name (without extension) on to which the convolution product between the equalized impulse response and the dry signal recording will be saved.

2.1.2. Description

This function manages the input and output parameters of functions *parfiltid* and *parfilt*. For function *parfiltid* it sets out as input parameters the following:

- Pole locations (input parameter of function *roomcomp_RA_1*).
- Filtered output target array which is obtained by applying a 4th order high pass Butterworth filter to a Delta function which has the same size as per the minimum phase reconstructed signal.

- Minimum phase reconstructed signal (obtained via a cepstrum analysis).
- Number of taps for FIR filters (optional).

The output parameters from this function are then used to implement a parallel filter procedure on the loudspeaker – room impulse response, via function *parfilt.* The result is an equalized impulse response.

Finally a “dry signal” is convolved with the unequalised impulse response, and with the equalized impulse response (by invoking function *conv_mono_RA_1*). The products of both convolutions are played back and saved in individual WAV files. By default the convolution product between the measured (i.e. unequalized) impulse response and the dry signal recording will be saved in WAV file called "un-equalized_wetsignal.wav". The latter convolution product will be saved in a WAV file whose filename is as per parameter *ES_soundfilename*.

2.1.3. References

Balazs, Bank; *Transfer Function Modelling and Equalization by Fixed Pole Parallel Filter* [Online] Available http://home.mit.bme.hu/~bank/parfilt/#matlabcode, 31st March 2011.

Balazs, Bank (2008); *Perceptually Motivated Audio Equalization Using Fixed-Pole Parallel Second-Order Filters*; IEEE Signal Processing Letters, Volume 15.

2.2. Function *parfiltid*

Function calculates pole and zero coefficients for parallel IIR 2nd order filters, as well as coefficients for FIR filters if required (optional).

2.2.1. Syntax

[Bm,Am,FIR] = parfiltid(in,out,p,NFIR), where:

- *in* is the array containing the minimum phase reconstructed signal.
- *out* is the filtered output target array.
- *p* is the array containing the pole locations.
- *NFIR* is the array containing the number of taps for the FIR filters (optional).
- *Bm* is the array containing the zero coefficients for the 2nd order IIR parallel filters.
- *Am* is the array containing the pole coefficients for the 2nd order IIR parallel filters.
- *FIR* is the array containing the coefficients for the parallel FIR filters.

2.2.2. Description

To estimate the impulse response of each parallel filter, this function filters the minimum phase reconstructed signal (MPRS) through a series of 2nd order filters with no zero coefficients and pole coefficients obtained by the linear least square method. The filtered MPRS is stored in the impulse response matrix M.

The B_M array is then calculated from the P_{opt} parameter which is estimated as per the following equation: $P_{opt} = (M^H M)^{-1} M^H h_t$, where M^H is the conjugate transpose of M, and h_t is the filtered output target array. The A_M array is estimated by producing polynomials which have roots based on the pole locations contained in array *p*.

2.2.3. References

Balazs, Bank; *Transfer Function Modelling and Equalization by Fixed Pole Parallel Filter* [Online] Available http://home.mit.bme.hu/~bank/parfilt/#matlabcode, 31st March 2011.

Balazs, Bank (2008); *Perceptually Motivated Audio Equalization Using Fixed-Pole Parallel Second-Order Filters*; IEEE Signal Processing Letters, Volume 15.

2.3. Function *parfilt*

Function implements parallel filter procedure on an input signal (i.e. loudspeaker – room impulse response)

2.3.1. Syntax

y = parfilt(Bm,Am,FIR,x), where:

- *Bm* is the array containing the zero coefficients of the 2nd order IIR filters.
- *Am* is the array containing the pole coefficients of the 2nd order IIR filters.
- *FIR* is the array containing the coefficients of the FIR filters (optional).
- *x* is the array containing the signal to be filtered.
- *y* is the array containing the filtered signal.

2.3.2. Description

Using a *for* loop, this function implements a parallel filter routine using the zero and pole coefficients entered as input parameters (in the function *RoomModes_RA_2*, these parameters were estimated by function *parfiltid*).

2.3.3. References

Balazs, Bank; *Transfer Function Modelling and Equalization by Fixed Pole Parallel Filter* [Online] Available http://home.mit.bme.hu/~bank/parfilt/#matlabcode, 31st March 2011.

Balazs, Bank (2008); Perceptually Motivated Audio Equalization Using Fixed-Pole Parallel Second-Order Filters; IEEE Signal Processing Letters, Volume 15.

2.4. Function *RoomModes_RA_2*

Function calculates room mode frequencies for rectangular rooms.

2.4.1. Syntax

R = RoomModes_RA_2(x,y,z,c,o,showmodes,axial_opt), where:

- *x, y, z* are the physical room dimensions corresponding to length, width and height respectively (in metres).
- *c* is the speed of sound (343 m/s by default).
- *o* is the highest room order
- *showmodes* is an option to visualize the room mode frequencies and corresponding room modes (0 corresponds to disable this option, any other number to activate this option).
- *axial_opt* is an option to only calculate axial room mode frequencies (0 corresponds to disable this option, any other number to activate this option).
- *R* is the array containing, in ascending order, all calculated room mode frequencies.

2.4.2. Description

This function uses the following equation to estimate room mode frequencies for rectangular rooms:

$$f_{(n_l,n_w,n_h)} = \frac{c}{2}\sqrt{[(\frac{n_l}{l})^2 + (\frac{n_w}{w})^2 + (\frac{n_h}{h})^2]}$$, where:

- c is the speed of sound.
- n_l, n_h, n_w are integer values which define the mode number.
- l is the room length in metres.
- w is the room width in metres.
- h is the room height in metres.

Please note the following if using this function to generate pole locations which will be used in functions *roomcomp_RA_1*, *parfiltid* and *parfilt*:

- Recommended highest room mode order which is used to calculate all type of room mode frequencies (i.e. tangential, axial and oblique) is 1. A higher room order could produce room mode frequencies which could produce inaccuracies in the calculations of zero and pole coefficients of 2nd order IIR parallel filters and parallel FIR filters. This might be due to the close proximity of the estimated room mode frequencies corresponding to higher room mode orders.

- If calculating frequencies corresponding only to axial modes, then any high room order could be used.

2.4.3. References

Bies, David A.; Hansen, Colin H. (2003) *Engineering Noise Control – Theory and Practice*, 3rd Edition, Spon Press.

2.5. Function *conv_mono_RA_1*

Function performs a convolution between two signals (both should be mono signals).

2.5.1. Syntax

output = conv_mono_RA_1(dryspeech,simir), where:

- *dryspeech* is the output array produced from using *wavread* Matlab function on a WAV file. If used within function *roomcomp_RA_1* then this array should correspond to the "dry signal".

- *simir* is produced in the same manner as per parameter *dryspeech*. However if used within function *roomcomp_RA_1* then this array should correspond to the impulse response.

- *output* is the array containing the result of the convolution procedure.

2.5.2. Description

This function produces a convolution between two signals as per the following procedure:

- Calculates the Fast Fourier Transform (FFT) of each input signal.

- Multiplies the FFTs.

- Performs an Inverse Fourier Transform (IFFT) for the product of the above.

2.5.3. References

Smith, Stephen W. (1997) *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, California: California Technical Publishing.

2.6. Function *norm_RA_1*

Function normalizes a signal to amplitude values between 1 and -1.

2.6.1. Syntax

norm_signal=norm_RA_1(input), where:

- *input* is the array containing the signal to be normalized.
- *norm_signal* is the array containing the normalized signal.

2.6.2. Description

This function normalizes a signal as per the following procedure:

- Estimates the absolute value for all elements in the input array.
- Determines the maximum absolute value.
- Defines a gain factor by dividing 0.999 with the maximum absolute value.
- The normalized signal is calculated by multiplying the input signal with the gain factor.

3. Appendix A: Matlab Scripts (Including Help Texts)

3.1. Main Script

```
%% Main Script
%This script is the interface between the user and the MATLAB functions
%It requests all the user input requirements which will be used by the functions
%It starts by requesting the frequencies which will be used for
%equalization, hence the user can opt to calculate room mode frequencies OR
%enter particular frequencies.
%Then the equalization procedure is performed.

clc
clear all
close all

freq_opt=input('Do you want to calculate room mode frequencies for equalization (0↙
for NO, any number for YES): ');
if ~isnumeric(freq_opt)
    freq_opt=input('Do you want to calculate room mode frequencies for equalization↙
(0 for NO, any number for YES): ');
else
    if freq_opt>0
        c=343;
        x=input('Enter room length: ');
        if ~isnumeric(x)
            error('Room length is not a number, aborting');
        end
        y=input('Enter room width: ');
        if ~isnumeric(y)
            error('Room width is not a number, aborting');
        end
        z=input('Enter room height: ');
        if ~isnumeric(z)
            error('Room height is not a number, aborting');
        end
        axial_opt=input('Do you want to calculate axial modes ONLY (0 for NO, any↙
number for YES): ');
        if ~isnumeric(axial_opt)
            axial_opt=input('Do you want to calculate axial modes ONLY (0 for NO, any↙
number for YES): ');
        end
        fprintf('\nPlease note highest order room mode should be 1 if all type of↙
modes (i.e. tangential, axial and oblique) are calculated for equalization↙
procedure');
        fprintf('\nAny highest order room mode could be used if only calculating↙
axial room modes for equalization procedure \n');
        high_mode=input('Please enter highest order room mode: ');
        if ~isnumeric(high_mode)
            error('Highest order room mode is not a number, aborting');
        end
        rm_md_freq=RoomModes_RA_2(x,y,z,c,high_mode,0,axial_opt);
    else
        rm_md_freq_lgth=input('How many frequencies do you want to equalize for ?:↙
');
        if ~isnumeric(rm_md_freq_lgth)
            rm_md_freq_lgth=input('How many frequencies do you want to equalize for↙
?: ');
        end
        for i=1:rm_md_freq_lgth
            rm_md_freq(i)=input(['Enter frequency ',num2str(i),' for equalization:↙
']);
            if ~isnumeric(rm_md_freq(i))
                rm_md_freq(i)=input(['Enter single frequency ',num2str(i),' for↙
equalization: ']);
            end
        end
    end
end

IR_filename=input('Enter filename containing loudspeaker - receiver impulse response↙
(do not enter the WAV extension): ','s');
DS_filename=input('Enter filename containing dry signal recording (do not enter the↙
WAV extension): ','s');
ES_filename=input('Enter filename of WAV file which will contain convolution product↙
of dry signal and equalized impulse response (do not enter the WAV extension):↙
','s');

roomcomp_RA_2(rm_md_freq,IR_filename,DS_filename,ES_filename);
```

3.2. Function *roomcomp_RA_2*

```
function roomcomp_RA_2(freq,IR_soundfilename,DS_soundfilename,ES_soundfilename)
%roomcomp_RA_1: Sets out input parameters for function partfiltid such as
%pole locations (based on frequencies selected by the user), filtered
%output target array, minimum phase reconstructed signal (obtained from
%cepstrum analysis of measured loudspeaker - room impulse response), and
%number of taps for FIR filters (optional).
%Outputs from this function are then transferred to function partfilt where
%the measured loudspeaker - room impulse response is equalized via parallel
%2nd order IIR filters and optional FIR filters.
%Finally, it convolves dry signal (i.e. recording in an anechoic chamber) with
%the unequalized and equalized loudspeaker-room impulse
%responses. The result of each convolution process is played back and
%saved in a WAV file.
%By default the convolution product between the measured (i.e. unequalized)
%impulse response and the dry signal recording will be saved in WAV
%file called "un-equalized_wetsignal.wav".
%SYNTAX: roomcomp_RA_1(freq,IR_soundfilename,DS_soundfilename)
%Input parameters:
%freq: Frequencies (as selected by user) at which poles will be located
%IR_soundfilename: Name of WAV file (without extension) which contains the
%measured loudspeaker - room impulse response
%DS_soundfilename: Name of WAV file (without extension) which contains the
%dry signal recording
%ES_soundfilename: Name of WAV file (without extension) which will contain
%the convolution product between the equalized impulse response and the dry
%signal recording
%Originally written by:  C. Balazs Bank, Helsinki University of Technology,
%2007.
%Based on following paper: Balázs Bank: Perceptually Motivated Audio Equalization
%Using Fixed-Pole Parallel Second-Order Filters
%IEEE Signal Processing Letters, 2008

clc

filetype = '.wav';
IR_filename = strcat(IR_soundfilename,filetype);

[x,Fs,nbits]=wavread(IR_filename);
x=x(1405:1.6e4); % this line might be adjusted depending on length of recorded IR
data=x;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%logarithmic pole positioning

Fs=44100;

%R=0.5; %you may need to change this depending on the number of poles (more poles:↙
larger, less poles: smaller)
% fplog=[logspace(log10(30),log10(200),13) logspace(log10(250),log10(18000),12)]; %↙
two sets of log. resolution

R=1.5;
%fplog=[logspace(log10(90),log10(95),10)]; %two sets of resolution
%fplog=90.5;
```

```
fplog=freq';

wp=2*pi*fplog/Fs;
p=R.^(wp/pi).*exp(j*wp);
plog=[p conj(p)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%preparing data

% data=impresp(225:100224); %neglecting the delay part

%[cp,minresp]=rceps(data(1:100000)); %making the measured response minumum-phase
[cp,minresp]=rceps(data);
output=zeros(length(minresp),1);
output(1)=1; %target

[Bf,Af]=butter(4,30/(Fs/2),'high');
outf=filter(Bf,Af,output); %making the target output a 30 Hz highpass

imp=zeros(1,length(data));
imp(1)=1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%filter design

[Bm,Am,FIR]=parfiltid(minresp,outf,plog,1); %Parallel filter design

equalizedresp=parfilt(Bm,Am,FIR,data); %equalized loudspeaker response - filtering↙
the
                        %measured transfer function by the parallel filter

equalizer=parfilt(Bm,Am,FIR,imp); %equalizer impulse response - filtering a unit↙
pulse

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DS_filename = strcat(DS_soundfilename,filetype);
ES_filename = strcat(ES_soundfilename,filetype);

[ds,fs_ds,nb_ds]=wavread(DS_filename);

un_ed_conv_ds=conv_mono_RA_1(ds,data);
disp('Listening to un-equalized wetsignal');
sound(un_ed_conv_ds,fs_ds);
n_un_ed_conv_ds=norm_RA_1(un_ed_conv_ds);
wavwrite(n_un_ed_conv_ds,fs_ds,'un-equalized_wetsignal.wav');
keyboard

ed_conv_ds=conv_mono_RA_1(ds,equalizedresp);
disp('Listening to equalized wetsignal');
sound(ed_conv_ds,fs_ds);
n_ed_conv_ds=norm_RA_1(ed_conv_ds);
wavwrite(n_ed_conv_ds,fs_ds,ES_filename);
```

3.3. Function *parfiltid*

```
function [Bm,Am,FIR]=parfiltid(in,out,p,NFIR);
%parfiltid: Calculates pole and zero coefficients for IIR 2nd order
%filters, as well as pole coefficients for FIR filters (optional), for a
%certain number of pole locations as selected by the user.
%SYNTAX: [Bm,Am,FIR]=parfiltid(in,out,p,NFIR)
%Input parameters:
%in: minimum phase reconstructed signal (as calculated in function
%roomcomp_RA_1)
%out: filtered output target array (as calculated in function
%roomcomp_RA_1)
%p: pole locations (as selected by the user)
%NFIR: number of taps for paraller FIR filters (default number is 1)
%Output parameters:
%Bm: zero coefficients of 2nd order IIR parallel filters
%Am: pole coefficients of 2nd order IIR parallel filters
%FIR: coefficients for parallel FIR filters (optional)
%Originally written by: C. Balazs Bank, Helsinki University of Technology,
%2007.
%Based on following papers:
%   Balazs Bank, "Perceptually Motivated Audio Equalization Using Fixed-Pole Parallel
%   Second-Order Filters", IEEE Signal Processing Letters, 2008.
%   http://www.acoustics.hut.fi/go/spl08-parfilt
%
%   Balazs Bank, "Direct Design of Parallel Second-order Filters for
%   Instrument Body Modeling", International Computer Music Conference,
%   Copenhagen, Denmark, Aug. 2007.
%   http://www.acoustics.hut.fi/go/icmc07-parfilt

if nargin==3,
    NFIR=1;
end;

p=p(find(p)); %we don't want to have any poles in the origin - for that we have the↙
parallel FIR part
for k=1:length(p), %making the filter stable by flipping the poles into the unit↙
cirle
   if abs(p(k))>1
      p(k)=1/conj(p(k));
  end;
end;

p=cplxpair(p); %order it to complex pole pairs + real ones afterwards

%in order to have second-order sections only (i.e., no first order)
pnum=length(p); %number of poles
ppnum=2*floor(pnum/2); %the even part of pnum
ODD=0;
if pnum>ppnum, %if pnum is odd
    ODD=1;
end;


in=in(:); %making column vectors
out=out(:);
```

```
OUTL=length(out);
INL=length(in);

%making input the same length as the output

if INL>OUTL,
   in=in(1:OUTL);
end;

if INL<OUTL,
   in=[in; zeros(OUTL-INL,1)];
end;


L=OUTL;

%constructing the modeling signal matrix

for k=1:2:ppnum-1, %second-order sections
    resp=filter(1,poly(p(k:k+1)),in); %impluse response of the two-pole filter
    M(:,k)=resp;
    M(:,k+1)=[0 ;resp(1:L-1)]; %the response delayed by one sample
end;

if ODD, %if the number of poles is odd, we have a first-order section
    resp=filter(1,poly(p(pnum)),in);
    M(:,pnum)=resp;
end;

for k=1:NFIR, %parallel FIR part
    M(:,pnum+k)=[zeros(k-1,1); in(1:L-k+1)];
end;

y=out; %making it column vector

%Looking for min(||y-M*par||) as a function of par:
%least squares solution by equation solving
A=M'*M;
b=M'*y;
par=A\b;

%constructing the Bm and Am matrices
for k=1:ppnum/2,
    Am(:,k)=poly(p(2*k-1:2*k))';
    Bm(:,k)=par(2*k-1:2*k);
end;
if ODD, %we extend the first-order section to a second-order one by adding zero↙
coefficients
    Am(:,k+1)=[poly(p(pnum))'; 0];
    Bm(:,k+1)=[par(pnum); 0];
end;

FIR=0;
%constructing the FIR part
```

```
if NFIR>0,
    FIR=par(pnum+1:pnum+NFIR);
end;
```

3.4. Function *parfilt*

```
function y=parfilt(Bm,Am,FIR,x);
%parfilt: Implements parallel filter routine based on filter coefficients
%for IIR and FIR filters (as calculated in function parfiltid).
%SYNTAX: y=parfilt(Bm,Am,FIR,x)
%Input parameters:
%Bm: zero coefficients of 2nd order IIR filters
%Am: pole coefficients of 2nd order IIR filters
%FIR: FIR filter coefficients
%x: signal to be filtered (i.e. measured loudspeaker - room impulse
%response
%Output parameters:
%y: Filtered signal (i.e. equalized loudspeaker - room impulse response
%Originally written by: C. Balazs Bank, Helsinki University of Technology,
%2007.
%Based on following papers:
%    Balazs Bank, "Perceptually Motivated Audio Equalization Using Fixed-Pole↙
Parallel
%   Second-Order Filters", IEEE Signal Processing Letters, 2008.
%   http://www.acoustics.hut.fi/go/spl08-parfilt
%
%   Balazs Bank, "Direct Design of Parallel Second-order Filters for
%   Instrument Body Modeling", International Computer Music Conference,
%   Copenhagen, Denmark, Aug. 2007.
%   http://www.acoustics.hut.fi/go/icmc07-parfilt

y=zeros(size(x));
s=size(Am);
for k=1:s(2),
    y=y+filter(Bm(:,k),Am(:,k),x);
end;
y=y+filter(FIR,1,x);
```

3.5. Function *RoomModes_RA_2*

```
function R=RoomModes_RA_2(x,y,z,c,o,showmodes,axial_opt)
%RoomModes_RA_2: Calculates room mode frequencies for rectangular rooms,
%including axial, tangential and oblique modes. Highest mode order is
%specified by parameter "o".
%Room mode frequency calculation is based on following equation:
%                        _________________________
%                      /    2        2        2
% R          = c      / /qx\     /qy\     /qz\
%  qx,qy,qz  ---     / (----) + (----) + (----)
%             2 \   /   \x /     \y /     \z /
%                  \/
%where qx, qy, qz define the room mode
%SYNTAX: R = RoomModes_RA_2(x,y,z,c,o,showmodes,axial_opt)
%Input Parameters:
%x,y,z: Room dimensions where x is length, y is width, z is height
%c: speed of sound in the room. (Default = 343 m/s).
%o: highest room mode order which is used to calculate room mode frequencies.
%showmodes: any number (not zero) to display modes or 0 to just show results.
%axial_opt: any number (not zero) to calculate and/or display axial modes
%only or 0 for otherwise (Default = 0).
%Output Parameters:
%R: Array of room mode frequencies, arranged in ascending order
%Orginally written by : André Lundkvist, also known as Jiisuki,
%Website: http://www.jiisuki.net

clc
% tic;
%Check arguments.
if nargin <= 5
    fprintf('\nOptional display of modes disabled.\n')
    %Set mode-display false.
    showmodes = 0;
end
if nargin <= 4
    fprintf('\nUsing default order number = 2.\n')
    %Set default order.
    o = 2;
end
if nargin <= 3
    fprintf('\nUsing a normal speed of sound in air of 343 m/s.\n')
    %Set normal speed of sound in air, 20 degrees celsius.
    c = 343;
end
if nargin <= 2
    error('Function must have at least one dimension to calculate!')
end
%Check for invalid order input.
if o < 1
    fprintf('\nHighest order must be over 1, setting default order = 2\n')
    o = 2;
end
%Reset matrix.
if axial_opt==0
    R=zeros((o+1)^3,1);
else
```

```
    R=zeros((o*3)+1,1);
end

if c == 0
    disp('You don´t want to use vacuum, right? Setting speed to 343 m/s.')
    c = 343;
end
%Check for errors in dimension of the room.
if x == 0
    error('Invalid X dimension entered, aborting.')
else
x=abs(x);
end
if y == 0
    error('Invalid Y dimension entered, aborting.')
else
y=abs(y);
end
if z == 0
    error('Invalid Z dimension entered, aborting.')
else
z=abs(z);
end

%Reset calculations counter, used for output statistics.
k=0;
%Starts the loops that will cycle through all combinations of qx, qy, qz
%and save them in a 3d matrix, R(qx+1,qy+1,qz+1).
fprintf('\nStarting calculations.\n')
if showmodes ~= 0
fprintf('\n')
end
% keyboard
if axial_opt == 0
    for qz=0:o
        for qy=0:o
            for qx=0:o
                R(k+1)=(c/2)*sqrt(((qx)/x)^2+((qy)/y)^2+((qz)/z)^2);
                if showmodes ~= 0
                    fprintf('f(%i,%i,%i) = %g\n',qx,qy,qz,R(k+1))
                end
            k=k+1;
            end
        end
    end
else
    for qx=0:o
        qy=0;
        qz=0;
        R(k+1)= (c/2)*sqrt(((qx)/x)^2+((qy)/y)^2+((qz)/z)^2);
        % keyboard;
        if showmodes ~= 0
            fprintf('f(%i,%i,%i) = %g\n',qx,qy,qz,R(k+1))
        end
        k=k+1;
    end
    for qy=1:o
        qx=0;
        qz=0;
        R(k+1)= (c/2)*sqrt(((qx)/x)^2+((qy)/y)^2+((qz)/z)^2);
        % keyboard;
        if showmodes ~= 0
            fprintf('f(%i,%i,%i) = %g\n',qx,qy,qz,R(k+1))
        end
        k=k+1;
    end
    for qz=1:o
        qx=0;
        qy=0;
        R(k+1)= (c/2)*sqrt(((qx)/x)^2+((qy)/y)^2+((qz)/z)^2);
        % keyboard;
        if showmodes ~= 0
            fprintf('f(%i,%i,%i) = %g\n',qx,qy,qz,R(k+1))
        end
        k=k+1;
    end
end

R=sort(R);

%Easy output format for frequencies.
format short g
%Return success information
if showmodes ~= 0
    fprintf('\n%i calculations completed successfully.\nDisplaying modes list.\n',k)
else
    fprintf('\n%i calculations completed successfully.\nDisplaying quick sorted list.↙
\n',k)
end
% toc
```

3.6. Function *conv_mono_RA_1*

```
function output=conv_mono_RA_1(dryspeech,simir)
%conv_mono_RA_1: Conducts a convolution between a "dry signal" (i.e.
%anechoic room recording) with a selected impulse response. Convolution is
%performed by calculating the inverse FFT of the product between the FFT of
%the "dry signal" with the FFT of the impulse response.
%SYNTAX: output=conv_mono_RA_1(dryspeech,simir)
%Input parameters:
%dryspeech: Read output of "dry signal" WAV file using WAVREAD Matlab function
%simir: Read output of impulse response WAV file using WAVREAD Matlab function
%Output parameters:
%output: Product of convolution between "dry signal" and selected impulse
%response
%Based on MATLAB script convolvefftfiltcomparison.m (originally provided
%for DESC 9115 Tutorial 6)

lendryspeech = length(dryspeech);

lensimir = length(simir);

outputlength = lendryspeech + lensimir - 1;

output = ifft(fft(dryspeech, outputlength) .* fft(simir, outputlength));
```

3.7. Function *norm_RA_1*

```
function norm_signal=norm_RA_1(input)
%norm_RA_1: Normalizes a signal to amplitude values between 1 and -1
%SYNTAX: norm_signal=norm_RA_1(input)
%Input parameters:
%input: Signal to be normalized
%Output parameters:
%norm_signal: Normalized signal

input_abs=abs(input);
s_max=max(input_abs);
g_factor=0.999/s_max;
norm_signal=input.*g_factor;
```