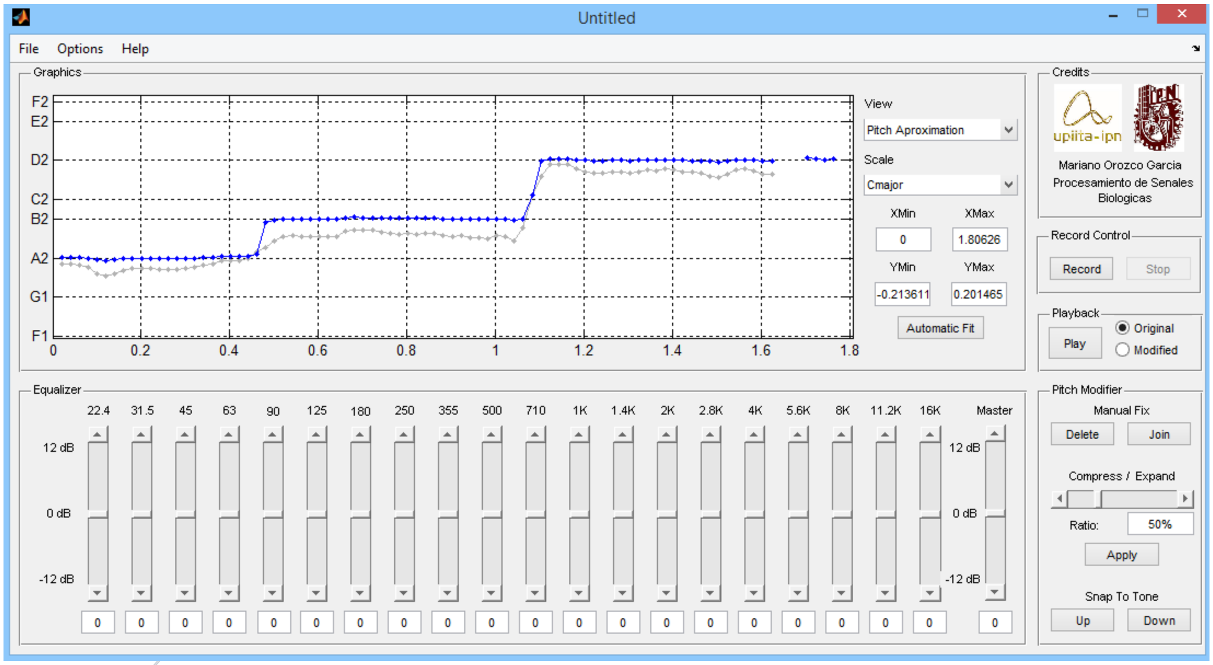
18-12-2015

Audio Tuner

Pitch corrector



View: Pitch Aproximation

Scale: Cmajor

XMin	XMax
0	1.80626

YMin	YMax
-0.213611	0.201465

Mariano Orozco Garcia
Procesamiento de Senales
Biologicas

● Original
○ Modified

22.4	31.5	45	63	90	125	180	250	355	500	710	1K	1.4K	2K	2.8K	4K	5.6K	8K	11.2K	16K	Master
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ratio: 50%

Mariano Orozco García
PROCESAMIENTO DE SEÑALES BIOLÓGICAS, 3BM4
Tovar Corona Blanca

Procesamiento de señales biológicas
3BM4

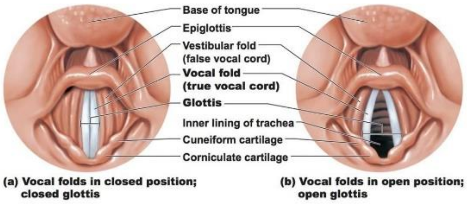
Tovar Corona Blanca
Orozco García Mariano

Audio Tuner – Pitch corrector

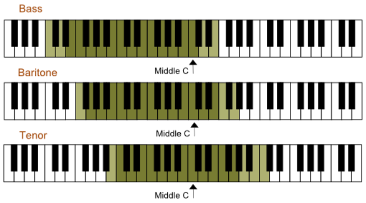
INTRODUCTION:

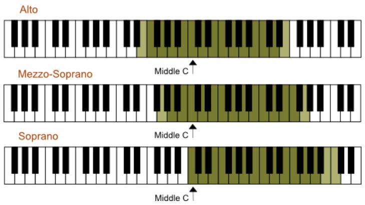
Human Voice

It is produced by the diaphragm action when it pushes air from the lungs through the vocal folds. This produces a periodic train of air that is shaped by the resonances of the vocal tract. The air make the vocal folds vibrate and this vibrations generate a sound wave that is a combination of several frequencies and their harmonics. The basic resonances, called vocal formants, can be changed by the action of the articulators to produce distinguishable voice sounds, like the vowel sounds.



In men and women different pitches can be archived. From bass to soprano voices the frequencies that the human voice can reach go from 82 Hz to 1056Hz.





Pitch correction

Pitch correction is an electronic effect that changes the intonation (highness or lowness in pitch) of an audio signal so that the pitches will be notes that correspond with the desired pitch of the song. Any pitch correction system first detects the pitch of an audio signal and then it calculates the desired change and modifies the audio signal.

The most common use of pitch correctors is to fix wrong intonation of notes sung by vocalists in popular music sound recordings. The use of pitch correction speeds up the recording process, because singers do not need to keep singing a song or vocal line and re-recording it until the pitches are correct. The pitch correction software can correct any pitch errors in the singing without the need for overdubbing or re-recording. However, it can also be used to fix intonation in recorded instrumental parts such as violin, cello or trumpet.

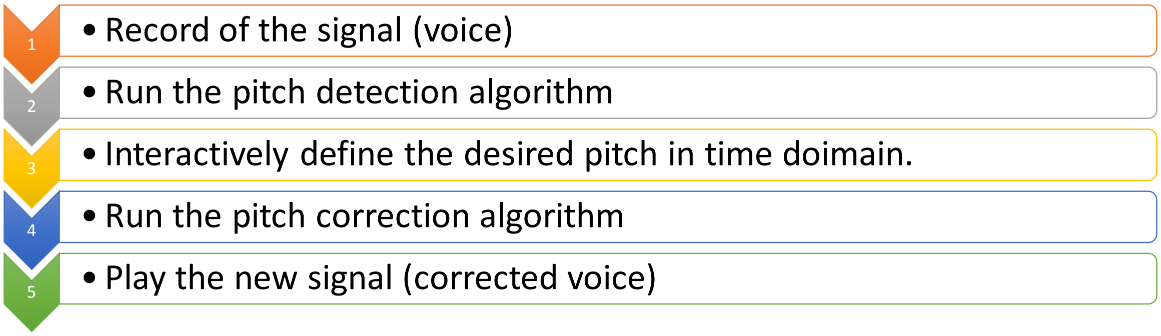
The use of pitch correction tools is controversial in music industry because it can make perfectly in-tune performances from a vocalist who is otherwise not skilled enough to give one. Because of this some artist make public that in their records or concerts no pitch corrector was used.

OBJECTIVE:

To implement a pitch corrector that add several effects to the human voice and adjust the original wave to the desired parameters in an interactive way.

DEVELOPMENT:

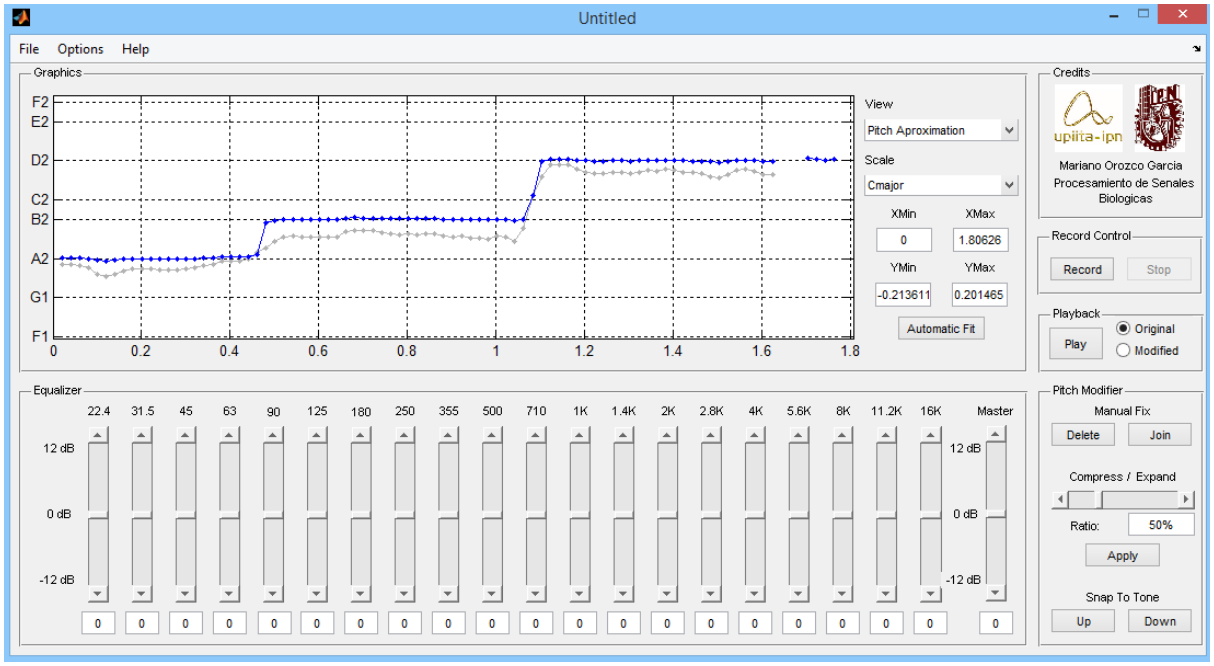
Block Diagram



1. Record of the signal (voice)
2. Run the pitch detection algorithm
3. Interactively define the desired pitch in time doimain.
4. Run the pitch correction algorithm
5. Play the new signal (corrected voice)

Implementation

In order to implement this tool we will use a GUI to make it interactive and user-friendly.



View: Pitch Aproximation
Scale: Cmajor
XMin: 0
XMax: 1.80626
YMin: -0.213611
YMax: 0.201465
Equalizer values: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Mariano Orozco Garcia
Procesamiento de Senales Biologicas
Playback: ● Original ○ Modified
Ratio: 50%

Figure1. *GUI used for the pitch corrector.*

As we can see the right panels are the ones that we will focus along with the graphics panel but we will omit the equalizer panel because it is beyond this particular job.

In the *graphics panel* we can choose between seen the pitch approximation view and the signal in the time domain, we can change the musical scale in which we want to work (mayor or minor of any key) and set the visualization window that we want to use or press the automatic Fit button. We can also use the graphics interactively in the pitch approximation view option. This allows us to use the mouse to select the points that are generated by the pitch detector algorithm and modify them using the pitch modifier panel that we will explain later in this section. To use the interactive plots we use two overlapped axes items in the GUI one to show the original signal in one and to move the items with the help of the mouse and the pitch corrector options in the other.

The *credits panel* show us information about the program.

The *record control* panel gives us the buttons to start recording or stop the record, only one is active at the same time for program flow purposes.

The *playback panel* offer us two options: to play the original record or the modified one. This is used to compare how the sounds are made and the program is stopped while the signal is being played.

The *pitch modifier panel* is the principal interest of this project. This includes the delete and join detected points in case to be necessary to manually correct the mistakes that the pitch detector could make. This is shown in figure 2.

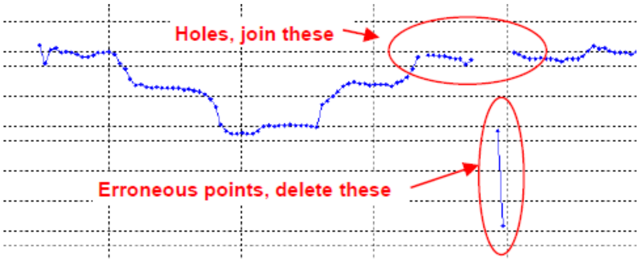


Figure2. *Use of delete and join buttons.*

The compress or expand option sets the compression rate (whose default is 50%) and modifies the signal when apply button is pressed. This compress or expand the selected points geometrically around the arithmetic media of the selected group. We can see an example in figure 3 of this process.

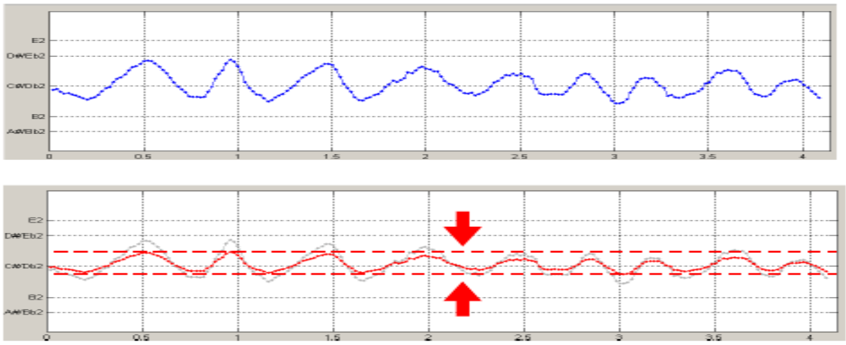
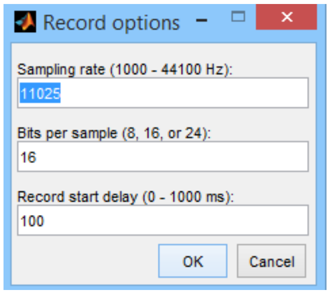


Figure3. *Compression example.*

Finally up and down buttons in the snap option will place the arithmetic media of the selected group exactly in the tone that is above or below the current value and move all the selected points when accordingly. This will compose the desired frequency vector that will be used in the pitch corrector algorithm.

In the file menu we will find some other options to export and import audio or save the plots as images (not implemented yet). We can also save or load the entire project, all the values of the options as well as the original and modified signals (already implemented).

In the options menu we can modify the recording options and the pitch corrector options. Record options include: the sampling rate, the bits per sample and the start delay time. The start delay time is used because of the time the program need to open the microphone and actively start recording. Default values are shown in figure 4. Pitch corrector options will be explained in detail in algorithm explanation section.


Record options

Sampling rate (1000 - 44100 Hz):
11025

Bits per sample (8, 16, or 24):
16

Record start delay (0 - 1000 ms):
100

Figure4. *Record options.*

In the help options we can find information about the GUI and the link to this file.

Algorithm explanation

The pitch detector algorithm works using an autocorrelation in time domain. First we separate de signals in blocks that will be analyzed with the correlation function and then with the help of an amplitude threshold we measure the frequency of the resulting signal. The components that have more amplitude that are minimum in the range where the pitch detector works and have harmonics that are also big in amplitude (with some tolerances) will be considered as the principal frequency.

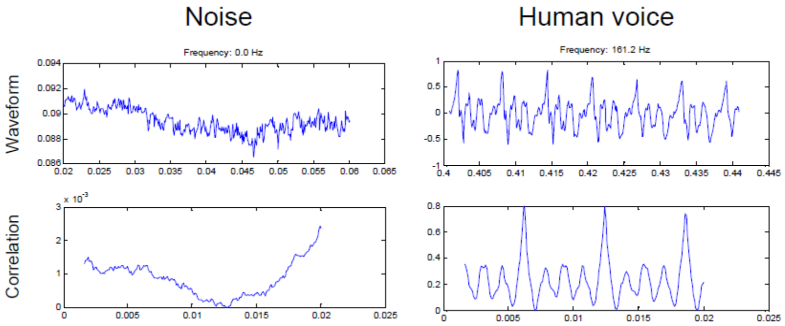
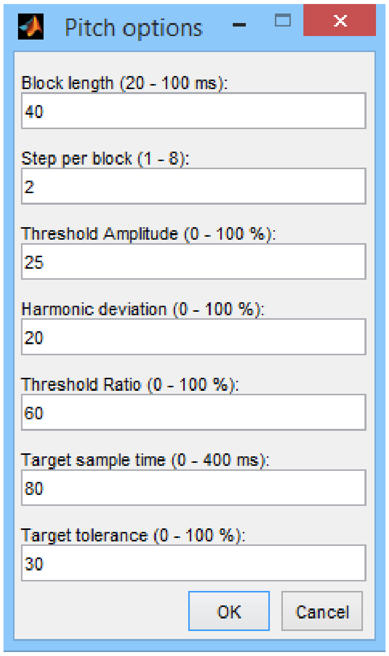


Figure5. *Correlation of signals.*

The frequencies that we obtain in the block will be saved and then we will choose the next block by moving certain amount of samples (step). The algorithm is made in a way that the step is shorter than the block so the blocks overlap and this gave us a better understanding of how the principal frequency is acting along the time.

Finally this vector is averaged in sets. If the previous and the next frequency are close this is considered as the same note. If they differ we consider this as the next note. We have a target sample time for the groups that will appear in the graphics, this ones are obtained by averaging the principal frequencies that we found along this period of time.

This algorithm requires some parameters to work whose default values are shown in figure 6. Block length is the size in time of the block. The step per block is the time that will be skipped from the previous block start to the next one and should be shorter than the block length. The threshold amplitude is used to reduce noise and we will only include in the analysis those frequencies that are above the desired ratio in amplitude. Harmonic deviation and threshold ratio are parameters to detect and eliminate the harmonics of other frequencies in order to not consider them as the principal one. Target sample time is used to define the amount of time that will contain a desired frequency that is representative for that lapse. The target sample time should be bigger than the block length. Finally target tolerance defines if in that target sample time there is a representative frequency or not, this depends on how disperse or compact the data is.



Pitch options

Block length (20 - 100 ms):
40

Step per block (1 - 8):
2

Threshold Amplitude (0 - 100 %):
25

Harmonic deviation (0 - 100 %):
20

Threshold Ratio (0 - 100 %):
60

Target sample time (0 - 400 ms):
80

Target tolerance (0 - 100 %):
30

Figure6. *Pitch options*

The pitch corrector algorithm uses the ratio between the desired frequency and the original one to interpolate or extrapolate the wave in the time domain by compressing or expanding it and then average or add the missing points in order to maintain the same sample rate of the original signal.

RESULTS:

Human voice can be modified effectively and pitch correction can be clearly heard with a medium trained ear.

Tests suggest that maximum record time should be around 20 seconds since buffer is required to keep data. Time of processing and pitch algorithms get unpractical beyond this point.

The pitch detector have some mistakes and must be manually corrected in the parameters or even graphically in some cases. However this algorithm is quite stable for any voice (male or female) and the modifications that should be made manually are minimal.

The audio loses quality and can be perceived as more artificial when you modify the signal when the ratios between the desired and original frequency is very high or very low. This is caused for the approximations made in the interpolation or extrapolation of the resulting signals that modify and loose data with reference to the original one.

PERSONAL CONCLUSIONS:

It is possible to create a pitch corrector tool of the human voice that detects any voice. This algorithm even work with instruments or other sounds.

Frequencies have to be set and changed in the pitch detector algorithm for woman or men since they have different spectra. Or for more general (but less accurate) purposes, set a bandwidth that involves both woman and man.

Noise in the record can substantially affect the results of the pitch detector. A good audio record and a controlled ambient should be generated in order to get good results.

If the algorithm is implement in a dedicated software or a faster platform we can extend its purposes for longer records, with more definition or with some modifications we could even implement it for real-time pitch corrector system. This real-time can be only made if we know the pitch we want to reach before we start to record.

REFERENCES:

[1] Jürgen Stutzki, Convolution, Autocorrelation, Cross-correlation, Power Spectrum: Fourier Transform and its Applications. Soummersemester 2007. Recovered from: [http://hera.ph1.uni-koeln.de/~stutzki/teaching/FT_appl_2.pdf].

[2] R. Nave, *Vocal sound production,* Hyper Physics, recovered from: [http://hyperphysics.phy-astr.gsu.edu/hbase/music/voice.html].

[3] *How the voice works,* American Academy of otolaryngology. Recovered from: [http://www.entnet.org/content/how-voice-works].

[4] Tonya Reiman, *The human Voice – Pitch*, Body Language University, Recovered from: [http://www.bodylanguageuniversity.com/public/203.cfm].

APPENDIX:

Code in Matlab:

```
function varargout = AudioTuner(varargin)
    % Initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename,
...
                       'gui_Singleton',
gui_Singleton, ...
                       'gui_OpeningFcn',
@AudioTuner_OpeningFcn, ...
                       'gui_OutputFcn',
@AudioTuner_OutputFcn, ...
                       'gui_LayoutFcn',  [] , ...
                       'gui_Callback',   []);
    if nargin && ischar(varargin{1})
        gui_State.gui_Callback =
str2func(varargin{1});
    end
    if nargout
        [varargout{1:nargout}] =
gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end


% --- Opening and Output Functions
function AudioTuner_OpeningFcn(hObject, ~, handles,
varargin)
    % This function has no output args, see
OutputFcn.
    % hObject    handle to figure
    % handles    structure with handles and user
data (see GUIDATA)
    % varargin   command line arguments to
AudioTuner (see VARARGIN)
    handles.output = hObject; % Choose default
command line output
    % Axes1 defaults
    set(handles.axes1,'XTickLabel',{});
    set(handles.axes1,'YTickLabel',{});
    % Axes2 is just for mouse click input
    set(handles.axes2,'XTick',[]);
    set(handles.axes2,'YTick',[]);
    set(handles.axes2,'XTickLabel',{});
    set(handles.axes2,'YTickLabel',{});

set(handles.axes2,'ButtonDownFcn','AudioTuner(''mou
seclick'',gcbo,[],guidata(gcbo))');
    % Set sound
    handles.sound.A = [];
    handles.sound.A_corrected = [];
    handles.sound.Fs = [];
    handles.sound.t = [];
    handles.sound.f0 = [];
    handles.sound.f0_save = [];
    handles.sound.f0_corrected = [];
    handles.sound.f0_corrected_save = [];
    handles.sound.scale_factor = [];
    handles.sound.tcalc = [];
    handles.sound.selected_points = logical([]);
    handles.sound.selected_points_save =
logical([]);
    % Set status
    handles.status.isrecording = false;
    handles.status.filename = '';
    handles.status.Xlim = [];
    handles.status.Ylim = [];
    % Set recording default options
    handles.record_options.Fs = 11025;
% Sampling frequency (Hz)
    handles.record_options.nbits = 16;
% Bits of precision
    handles.record_options.initial_trim = 0.1;
% Always trim off the initial 0.1 sec
    % Pitch detection defaults
    handles.pitch_options.Fmin = 50;
% Min frequency to search
    handles.pitch_options.Fmax = 600;
% Max frequency to search
    handles.pitch_options.block_length = 0.04;
% Length of each chuck to analyze for frequency
    handles.pitch_options.step_per_block = 2;
% What fraction of the block width to step for each
pitch detect
    handles.pitch_options.threshold_amp = 0.25;
% Threshold of autocorr to be called a peak
    handles.pitch_options.harmonic_deviation = 0.2;
% Max deviation from multiple fundamental to be
considered a harmonic (0.2 = +/- 20%)
    handles.pitch_options.threshold_ratio = 0.6;
% Max ratio in heights of autocorr allowed between
fundamental and harmonics
    handles.pitch_options.target_sample_time =
0.08; % Over what time interval to sample previous
freq's to determine next target f0
    handles.pitch_options.target_tol = 0.3;
% Max tolerance from f0 target (0.2 = +/-20%)
    handles.pitch_options.compress_ratio = 0.5;
% Ratio to compress tone (0-100 %)
    % Scale options
    handles.scale_options.scale = 'Cmajor';
% Scale for pitch correction
    handles.scale_options.indices = [];
    handles.scale_options.freqs = [];
    handles.scale_options.notes = {};
    handles.scale_options.fund_index = [];
    [handles.scale_options.indices,...
    handles.scale_options.freqs,...
    handles.scale_options.notes,...
    handles.scale_options.fund_index] = ...
        get_scale(handles.scale_options.scale);
    % Other options
    handles.record_obj = [];
    % Update handles structure
    handles = update_GUI(handles);
    guidata(hObject, handles);

function varargout = AudioTuner_OutputFcn(~, ~,
handles)
    % varargout  cell array for returning output
args (see VARARGOUT);
    % handles    structure with handles and user
data (see GUIDATA)
    % Get default command line output from handles
structure
    varargout{1} = handles.output;

function axes3_CreateFcn(hObject, ~, ~)
    imshow(imread('UPIITA.jpg'));

function axes4_CreateFcn(hObject, ~, ~)
    imshow(imread('IPN.jpg'));

% --- Equalizer Create Sliders
function B1_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B2_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
```

```
function B3_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B4_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B5_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B6_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B7_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B8_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B9_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B10_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B11_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B12_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B13_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B14_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B15_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B16_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B17_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B18_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B19_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function B20_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function Volume_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end


% --- Equalizer Create Text Box For values
function V1_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V2_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V3_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V4_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V5_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V6_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
```

```
function V7_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V8_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V9_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V10_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V11_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V12_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V13_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V14_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V15_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V16_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V17_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
```

```
function V18_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V19_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function V20_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function VVolume_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end


% --- Graphics Create Options
function view_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function Scale_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function XMin_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function XMax_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function YMin_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function YMax_CreateFcn(hObject, ~, ~)
    if ispc &&
isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
```

```
% --- Pitch Create Functions
function CompExp_CreateFcn(hObject, ~, ~)
    if isequal(get(hObject,'BackgroundColor'), 
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

function VCompExp_CreateFcn(hObject, ~, handles)
    if ispc && 
isequal(get(hObject,'BackgroundColor'), 
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end


% --- Equalizer Callback Sliders
function B1_Callback(hObject, ~, handles)
    set(handles.V1,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B2_Callback(hObject, ~, handles)
    set(handles.V2,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B3_Callback(hObject, ~, handles)
    set(handles.V3,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B4_Callback(hObject, ~, handles)
    set(handles.V4,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B5_Callback(hObject, ~, handles)
    set(handles.V5,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B6_Callback(hObject, ~, handles)
    set(handles.V6,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B7_Callback(hObject, ~, handles)
    set(handles.V7,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B8_Callback(hObject, ~, handles)
    set(handles.V8,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B9_Callback(hObject, ~, handles)
    set(handles.V9,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B10_Callback(hObject, ~, handles)
    set(handles.V10,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B11_Callback(hObject, ~, handles)
    set(handles.V11,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B12_Callback(hObject, ~, handles)
    set(handles.V12,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B13_Callback(hObject, ~, handles)
    set(handles.V13,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B14_Callback(hObject, ~, handles)
    set(handles.V14,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B15_Callback(hObject, ~, handles)
    set(handles.V15,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B16_Callback(hObject, ~, handles)
    set(handles.V16,'String',get(hObject,'Value'))
    guidata(hObject, handles);
```

```
 function B17_Callback(hObject, ~, handles)
    set(handles.V17,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B18_Callback(hObject, ~, handles)
    set(handles.V18,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B19_Callback(hObject, ~, handles)
    set(handles.V19,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function B20_Callback(hObject, ~, handles)
    set(handles.V20,'String',get(hObject,'Value'))
    guidata(hObject, handles);

function Volume_Callback(hObject, ~, handles)

set(handles.VVolume,'String',get(hObject,'Value'))
    guidata(hObject, handles);

% --- Graphics Callback
function view_Callback(hObject, ~, handles)
    if isempty(handles.sound.A_corrected)
        handles = run_pitch_correction(handles);
    end
    handles = update_plot(handles);
    guidata(hObject, handles);

function Scale_Callback(hObject, ~, handles)
    scale = get(handles.Scale,'String');
    scale = scale{get(handles.Scale,'Value')};
    if ~strcmp(scale,'CUSTOM')
        handles.scale_options.scale = scale;
        [handles.scale_options.indices,...
            handles.scale_options.freqs,...
            handles.scale_options.notes,...
            handles.scale_options.fund_index] = ...
            get_scale(handles.scale_options.scale);
    end
    handles = update_plot(handles);
    guidata(hObject, handles);

function XLim_Callback(hObject, ~, handles)
    if ~isempty(handles.sound.t)
        mi = 
max(str2num(get(handles.XMin,'String')),min(handles
.sound.t));
        ma = 
min(str2num(get(handles.XMax,'String')),max(handles
.sound.t));
        if mi>=ma
            mi = min(handles.sound.t);
            ma = max(handles.sound.t);
        end
        handles.status.Xlim = [mi,ma];
        set(handles.XMin,'String',mi);
        set(handles.XMax,'String',ma);
        guidata(hObject, handles);
    else
        disp('error');
    end

function YLim_Callback(hObject, ~, handles)
    if ~isempty(handles.sound.t)
        mi = 
max(str2num(get(handles.YMin,'String')),min(handles
.sound.A));
        ma = 
min(str2num(get(handles.YMax,'String')),max(handles
.sound.A));
        if mi>ma
            mi = min(handles.sound.A);
            ma = max(handles.sound.A);
        end
        handles.status.Ylim = [mi,ma];
        set(handles.YMin,'String',mi);
        set(handles.YMax,'String',ma);
        guidata(hObject, handles);
    else
        disp('error');
    end

function AutomaticFit_Callback(hObject, ~, handles)
    handles.status.Xlim =
[min(handles.sound.t),max(handles.sound.t)];
    handles.status.Ylim =
[min(handles.sound.A),max(handles.sound.A)];
    handles = update_plot(handles);
    guidata(hObject, handles);

% --- Record Control
function Record_Callback(hObject, ~, handles)
    try
        handles.record_obj =
audiorecorder(handles.record_options.Fs,...
            handles.record_options.nbits,1);
        record(handles.record_obj);
        handles.status.isrecording = true;
        handles = update_GUI(handles);
        guidata(hObject, handles);
    catch
        h = errordlg('Error starting audio input
device! Check sound card and microphone!','ERROR');
        waitfor(h);
    end

function Stop_Callback(hObject, ~, handles)
    if handles.status.isrecording
        % Stop recording
        stop(handles.record_obj);
        handles.status.isrecording = false;
        handles = update_GUI(handles);
        % Save values
        handles.sound.A =
getaudiodata(handles.record_obj,'single');
        Nstart =
round(handles.record_options.initial_trim*...
            handles.record_options.Fs); % Trim
initial samples
        handles.sound.A =
handles.sound.A(Nstart:end);
        handles.sound.Fs =
handles.record_options.Fs;
        handles.sound.A_corrected =
handles.sound.A;
        handles.sound.t =
(0:length(handles.sound.A)-
1)./handles.record_options.Fs;
        % Clear old frequency data
        handles.sound.f0 = [];
        handles.sound.f0_save = [];
        handles.sound.f0_corrected = [];
        handles.sound.f0_corrected_save = [];
        handles.sound.selected_points =
logical([]);
        handles.sound.selected_points_save =
logical([]);
        % Clear plot limits
        handles.status.Xlim = [];
        handles.status.Ylim = [];
        % Update
        handles = run_pitch_detection(handles);

set(handles.XMin,'String',min(handles.sound.t));

set(handles.XMax,'String',max(handles.sound.t));
        handles.status.Xlim =
[min(handles.sound.t),max(handles.sound.t)];

set(handles.YMin,'String',min(handles.sound.A));

set(handles.YMax,'String',max(handles.sound.A));
        handles.status.Ylim =
[min(handles.sound.A),max(handles.sound.A)];
        set(handles.view,'Value',3)
        handles = update_plot(handles);
        Scale_Callback(hObject, [], handles)
```

```
        handles = update_GUI(handles);
        guidata(hObject, handles);
    end


% --- Playback
function Play_Callback(hObject, ~, handles)
    % Decide which to play
    choice =
get(get(handles.PlayChoice,'SelectedObject'),'Strin
g');
    choice = strcmp(choice,'Original');
    if choice
        if isempty(handles.sound.A)
            return
        end
        A = handles.sound.A;
    else
        if isempty(handles.sound.A_corrected)
            handles =
run_pitch_correction(handles);
        end
        A = handles.sound.A_corrected;
    end
    wavplay(A,handles.sound.Fs);
    axes(handles.axes2);


% --- Pitch Modifier
function Delete_Callback(hObject, ~, handles)
    if isempty(handles.sound.f0_corrected) ||
~any(handles.sound.selected_points)
        return
    end
    % Save for undoing
    handles.sound.f0_save(end+1,:) =
handles.sound.f0;
    handles.sound.f0_corrected_save(end+1,:) =
handles.sound.f0_corrected;
    handles.sound.selected_points_save(end+1,:) =
handles.sound.selected_points;
    % Pull out the frequency and time of the
selected points

handles.sound.f0_corrected(handles.sound.selected_p
oints) = 0;
    handles.sound.f0(handles.sound.selected_points)
= 0;
    handles.sound.selected_points =
false(size(handles.sound.f0));
    handles.sound.A_corrected = [];
    handles = update_plot(handles);
    Scale_Callback(hObject, [], handles)
    handles = update_GUI(handles);
    guidata(hObject, handles);

function Join_Callback(hObject, ~, handles)
    if isempty(handles.sound.f0_corrected) ||
~any(handles.sound.selected_points)
        return
    end
    % Save for undoing
    handles.sound.f0_save(end+1,:) =
handles.sound.f0;
    handles.sound.f0_corrected_save(end+1,:) =
handles.sound.f0_corrected;
    handles.sound.selected_points_save(end+1,:) =
handles.sound.selected_points;
    % Pull out the frequency and time of the
selected points
    any_changes = false;
    I = find(handles.sound.selected_points);
    f0 = handles.sound.f0;
    for n = 1:length(I)-1
        if I(n+1)-I(n) > 1
            if f0(I(n)) > 1 && f0(I(n+1)) > 1 &&
all(f0(I(n)+1:I(n+1)-1) < 1)
```

```
                f0(I(n):I(n+1)) =
logspace(log10(f0(I(n))),log10(f0(I(n+1))),length(I
(n):I(n+1)));
handles.sound.selected_points(I(n):I(n+1)) = true;
                any_changes = true;
            end
        end
    end
    handles.sound.f0 = f0;
    f0 = handles.sound.f0_corrected;
    for n = 1:length(I)-1
        if I(n+1)-I(n) > 1
            if f0(I(n)) > 1 && f0(I(n+1)) > 1 &&
all(f0(I(n)+1:I(n+1)-1) < 1)
                f0(I(n):I(n+1)) =
logspace(log10(f0(I(n))),log10(f0(I(n+1))),length(I
(n):I(n+1)));
                any_changes = true;
            end
        end
    end
    if ~any_changes
        return
    end
    handles.sound.f0_corrected = f0;
    handles.sound.A_corrected = [];
    handles = update_plot(handles);
    Scale_Callback(hObject, [], handles)
    handles = update_GUI(handles);
    guidata(hObject, handles);

function CompExp_Callback(hObject, ~, handles)

set(handles.VCompExp,'String',[num2str(get(hObject,
'Value')),'%'])
    handles.pitch_options.compress_ratio =
get(hObject,'Value')/100;
    guidata(hObject, handles);

function Apply_Callback(hObject, ~, handles)
    if isempty(handles.sound.f0_corrected) ||
~any(handles.sound.selected_points)
        return
    end
    % Pull out the frequency and time of the
selected points
    f0 =
handles.sound.f0_corrected(handles.sound.selected_p
oints);
    I = find(handles.sound.selected_points);
    t = handles.sound.tcalc(I(1):I(end));
    t = t-t(1);
    % Come up with a envelope for compressing that
goes from 1 down to the desired compression factor
over the desired time interval
    factor =
ones(size(t)).*handles.pitch_options.compress_ratio
;
    factor =
factor(handles.sound.selected_points(I(1):I(end)));
    % Scale the deviations around the mean in a log
sense
    f0 = 10.^(mean(log10(f0)) + factor.*(log10(f0)-
mean(log10(f0))));
    handles.sound.f0_corrected_save(end+1,:) =
handles.sound.f0_corrected;
    handles.sound.f0_save(end+1,:) =
handles.sound.f0;
    handles.sound.selected_points_save(end+1,:) =
handles.sound.selected_points;

handles.sound.f0_corrected(handles.sound.selected_p
oints) = f0;
    handles.sound.A_corrected = [];
    handles = update_plot(handles);
    Scale_Callback(hObject, [], handles);
    handles = update_GUI(handles);
    guidata(hObject, handles);
 function Snap_Callback(hObject, ~, handles)
    if isempty(handles.sound.f0_corrected) ||
~any(handles.sound.selected_points) ||...
            isempty(handles.scale_options.freqs);
        return
    end
    snap_direction = get(gcbo,'String');
    % Pull out the frequency and time of the
selected points
    I = find(handles.sound.selected_points);
    sp =
handles.sound.selected_points(I(1):I(end));
    f0 = handles.sound.f0_corrected(I(1):I(end));
    t = handles.sound.tcalc(I(1):I(end));
    t = t-t(1);
    where = 1;
    tmp = f0(where:end);
    mean_f0 = 10^mean(log10(tmp(sp(where:end))));
    if strcmp(snap_direction,'Down')
        Iu = find(handles.scale_options.freqs <
0.99*mean_f0);
        mean_new =
max(handles.scale_options.freqs(Iu));
    elseif strcmp(snap_direction,'Up')
        Iu = find(handles.scale_options.freqs >
1.01*mean_f0);
        mean_new =
min(handles.scale_options.freqs(Iu));
    end
    factor = mean_new/mean_f0.*ones(size(t));
    if where > 1
        factor(1:where) =
linspace(1,factor(end),where);
    end
    % Scale the deviations around the mean in a log
sense
    f0 = f0.*factor;
    handles.sound.f0_corrected_save(end+1,:) =
handles.sound.f0_corrected;
    handles.sound.f0_save(end+1,:) =
handles.sound.f0;
    handles.sound.selected_points_save(end+1,:) =
handles.sound.selected_points;

handles.sound.f0_corrected(handles.sound.selected_p
oints) = f0(sp);
    handles.sound.A_corrected = [];
    handles = update_plot(handles);
    Scale_Callback(hObject, [], handles);
    handles = update_GUI(handles);
    guidata(hObject, handles);

% --- Menu
function File_Callback(hObject, ~, handles)
    % File Menu

function Save_Callback(hObject, ~, handles)
    % Read data
    data.sound = handles.sound;
    data.status = handles.status;
    data.record_options = handles.record_options;
    data.pitch_options = handles.pitch_options;
    % Verify if the file has a name
    if ~isempty(handles.status.filename)
        FilterSpec = handles.status.filename;
    else
        FilterSpec = 'untitled.prj';
    end
    % Choose path and name for file then save
    [FileName,PathName,FilterIndex] =
uiputfile(FilterSpec,'Save');
    if ischar(FileName)
        handles.status.filename =
fullfile(PathName,FileName);
        data.status.filename =
fullfile(PathName,FileName);
        save(fullfile(PathName,FileName), 'data','-
mat');
    end
    guidata(hObject, handles);
```

```
function Load_Callback(hObject, ~, handles)
    % Warning not to loose current work
    button = questdlg('Any changes since last save
will be lost! Do you want to continue?',...
        'Confirm load','Yes','No','Yes');
    if strcmp(button,'No')
        return
    end
    % Verify if the file has a name
    if ~isempty(handles.status.filename)
        FilterSpec = handles.status.filename;
    else
        FilterSpec = '*.prj';
    end
    % Choose path and name for file then load
    [FileName,PathName,FilterIndex] =
uigetfile(FilterSpec,'Load');
    if ischar(FileName)
        load('-mat',fullfile(PathName,FileName));
        handles.status.filename =
fullfile(PathName,FileName);
        handles.sound = data.sound;
        handles.status = data.status;
        handles.record_options =
data.record_options;
        handles.pitch_options = data.pitch_options;
        % handles.scale_options =
data.scale_options;
        % handles.equalizer_options =
data.equalizer_options;
        handles = update_plot(handles);
        guidata(hObject, handles);
    end

function Export_Callback(hObject, ~, handles)
    % Export Submenu

function Image_Callback(hObject, ~, handles)
    % Image

function Audio_Callback(hObject, ~, handles)
    % Audio

function Import_Callback(hObject, ~, handles)
    % Import Audio

function Options_Callback(hObject, ~, handles)
    % Options Menu

function RecordOptions_Callback(hObject, ~,
handles)
    % Read actual values:
    Fs = handles.record_options.Fs;
    nbits = handles.record_options.nbits;
    initial_trim =
handles.record_options.initial_trim*1000;
    % Create input dialog box
    prompt = {
        'Sampling rate (1000 - 44100 Hz):'
        'Bits per sample (8, 16, or 24):'
        'Record start delay (0 - 1000 ms):'
        };
    dlg_title = 'Record options';
    num_lines = [1 35];
    def = {
        num2str(Fs)
        num2str(nbits)
        num2str(initial_trim)
        };
    answer =
inputdlg(prompt,dlg_title,num_lines,def);
    % Save values if they are correct
    empty_error = false;
    input_error = false;
    if ~isempty(answer)
        if ~isempty(answer{1})
            temp = round(str2num(answer{1}));
            if temp < 1000 || temp > 44100
                input_error = true;
            else
                Fs = temp;
            end
        else
            empty_error = true;
        end
        if ~isempty(answer{2})
            temp = round(str2num(answer{2}));
            if ~ismember(temp,[8,16,24])
                input_error = true;
            else
                nbits = temp;
            end
        else
            empty_error = true;
        end
        if ~isempty(answer{3})
            temp = round(str2num(answer{3}));
            if temp < 0 || temp > 1000
                input_error = true;
            else
                initial_trim = temp;
            end
        else
            empty_error = true;
        end
        if empty_error
            hwarn = warndlg('One or more values are
empty, previous values are used!',...
                'WARNING');
            waitfor(hwarn);
        end
        if input_error
            hwarn = warndlg('One or more values out
of range, previous values are used!',...
                'WARNING');
            waitfor(hwarn);
        end
        % Save values
        handles.record_options.Fs = Fs;
        handles.record_options.nbits = nbits;
        handles.record_options.initial_trim =
initial_trim/1000;
        % Update state of GUI
        handles = update_plot(handles);
        guidata(hObject, handles);
    end

function PitchOptions_Callback(hObject, ~, handles)
    % Read actual values:
    block_length =
handles.pitch_options.block_length*1000;
    step_per_block =
handles.pitch_options.step_per_block;
    treshold_amp =
handles.pitch_options.threshold_amp*100;
    harmonic_deviation =
handles.pitch_options.harmonic_deviation*100;
    threshold_ratio =
handles.pitch_options.threshold_ratio*100;
    taget_sample_time =
handles.pitch_options.target_sample_time*1000;
    target_tol =
handles.pitch_options.target_tol*100;
    % Create input dialog box
    prompt = {
        'Block length (20 - 100 ms):'
        'Step per block (1 - 8):'
        'Threshold Amplitude (0 - 100 %):'
        'Harmonic deviation (0 - 100 %):'
        'Threshold Ratio (0 - 100 %):'
        'Target sample time (0 - 400 ms):'
        'Target tolerance (0 - 100 %):'
    };
    dlg_title = 'Pitch options';
    num_lines = [1 35];
```

```
   def = {
        num2str(block_length)
        num2str(step_per_block)
        num2str(treshold_amp)
        num2str(harmonic_deviation)
        num2str(threshold_ratio)
        num2str(taget_sample_time)
        num2str(target_tol)
    };
    answer =
inputdlg(prompt,dlg_title,num_lines,def);
    % Save values if they are correct
    empty_error = false;
    input_error = false;
    if ~isempty(answer)
        if ~isempty(answer{1})
            temp = round(str2num(answer{1}));
            if temp < 20 || temp > 100
                input_error = true;
            else
                block_length = temp;
            end
        else
            empty_error = true;
        end
        if ~isempty(answer{2})
            temp = round(str2num(answer{2}));
            if temp < 1 || temp > 8
                input_error = true;
            else
                step_per_block = temp;
            end
        else
            empty_error = true;
        end
        if ~isempty(answer{3})
            temp = round(str2num(answer{3}));
            if temp < 0 || temp > 100
                input_error = true;
            else
                treshold_amp = temp;
            end
        else
            empty_error = true;
        end
        if ~isempty(answer{4})
            temp = round(str2num(answer{4}));
            if temp < 0 || temp > 100
                input_error = true;
            else
                harmonic_deviation = temp;
            end
        else
            empty_error = true;
        end
        if ~isempty(answer{5})
            temp = round(str2num(answer{5}));
            if temp < 0 || temp > 100
                input_error = true;
            else
                threshold_ratio = temp;
            end
        else
            empty_error = true;
        end
        if ~isempty(answer{6})
            temp = round(str2num(answer{6}));
            if temp < 0 || temp > 400
                input_error = true;
            else
                taget_sample_time = temp;
            end
        else
            empty_error = true;
        end
        if ~isempty(answer{7})
            temp = round(str2num(answer{7}));
```

```
        if temp < 0 || temp > 100
                input_error = true;
            else
                target_tol = temp;
            end
        else
            empty_error = true;
        end
        if empty_error
            hwarn = warndlg('One or more values are
empty, previous values are used!',...
                'WARNING');
            waitfor(hwarn);
        end
        if input_error
            hwarn = warndlg('One or more values out
of range, previous values are used!',...
                'WARNING');
            waitfor(hwarn);
        end
        % Save values
        handles.pitch_options.block_length =
block_length/1000;
        handles.pitch_options.step_per_block =
step_per_block;
        handles.pitch_options.threshold_amp =
treshold_amp/100;
        handles.pitch_options.harmonic_deviation =
harmonic_deviation/100;
        handles.pitch_options.threshold_ratio =
threshold_ratio/100;
        handles.pitch_options.target_sample_time =
taget_sample_time/1000;
        handles.pitch_options.target_tol =
target_tol/100;
        % Update state of GUI
        handles = run_pitch_detection(handles);
        handles = update_plot(handles);
        guidata(hObject, handles);
    end

function Help_Callback(hObject, ~, handles)
    % Help Menu

function About_Callback(hObject, ~, handles)
    h = msgbox({
        '                           Audio Tuner'
        ''
        ' This program can add effects to voice
records'
        '      using a pitch corrector and an
equalizer.'
        ''
        '         Created by: Mariano Orozco
Garcia'
        '             morozcog1101@alumno.ipn.mx'
    },'About');
    waitfor(h);

function HowTo_Callback(hObject, ~, handles)
    try
        open('AudioTuner.pdf');
    catch
        h = errordlg({
            'Cannot open Audio Tuner.pdf.'
            'Please locate and open file manually
for instructions'
            },'ERROR OPENING HELP FILE');
        waitfor(h)
    end
```

```
% --- Pitch Functions
function handles = run_pitch_detection(handles)
    % Runs only if theres a sound clip
    if isempty(handles.sound.A)
        return
    end
    % Get variables out of the handles structure
    block_length =
handles.pitch_options.block_length;
    step_per_block =
handles.pitch_options.step_per_block;
    Fmin = handles.pitch_options.Fmin;
    Fmax = handles.pitch_options.Fmax;
    target_sample_time =
handles.pitch_options.target_sample_time;
    target_tol = handles.pitch_options.target_tol;
    harmonic_deviation =
handles.pitch_options.harmonic_deviation;
    threshold_ratio =
handles.pitch_options.threshold_ratio;
    threshold_amp =
handles.pitch_options.threshold_amp;
    A = handles.sound.A;
    t = handles.sound.t;
    Fs = handles.sound.Fs;
    % Variables for size of analysis
    block =
step_per_block*round(block_length*Fs/step_per_block
);     % Size of each block to find pitch
    step = block/step_per_block;
% Step (blocks are 4 times larger than "steps"
    N = floor((length(A)-block)/step);      %
Number of frequency computations
    % Initialize variables for storing results
    f0 = zeros(1,N);                        %
Initialize vector for storing frequencies
    tcalc = zeros(1,N);                     % The
time at which that frequency calculation is valid
    f0_target = [];                         % For
keeping track of the target for the next
calculation
    f0_samples =
floor(target_sample_time/(step/Fs)); % Figure out
how many f0 samples there will be
    % Waitbar
    hwait = waitbar(0,'Determining pitch...');
    set(hwait,'Name','Please Wait');
    waitbar_count = 0;
    waitbar_update = round(N/10);
    pause(0.001);
    % Algorithm for pitch detection
    I = 1;
    for n = 1:N
        % Update waitbar
        if waitbar_count > waitbar_update
            waitbar(n/N,hwait);
            waitbar_count = 0;
        end
        waitbar_count = waitbar_count + 1;
        % Extract a block of the wave file
        Atemp = A(I:I+block-1);
        ttemp = t(I:I+block-1);
        I = I+step;
        % Do the autocorrelation to find the
frequency
        [f0(n),acorr,Nshifts,Tshifts] = ...

find_f0_timedomain2(Atemp,Fs,Fmin,Fmax,...

threshold_amp,threshold_ratio,harmonic_deviation,..
.
            f0_target,target_tol);
        tcalc(n) = median(ttemp);
```

```
% If the previous and current are invalid, then
make the last one invalid
        % as well
        if n > 2
            if f0(n-2) < 1 && f0(n) < 1
                f0(n-1) = 0;
            end
        end
        % Look at the last few samples to determine
the target frequency for
        % the next iteration
        if n >= f0_samples
            f0_chunk = f0(n-f0_samples+1:n);
            f0_target = f0_chunk(f0_chunk>0);
            if ~isempty(f0_target)
                f0_target = mean(f0_target);
            end
        end
    end
    % Close waitbar
    if ishandle(hwait)
        close(hwait);
    end
    % Save calculation
    handles.sound.A_corrected = [];
    handles.sound.f0 = f0;
    handles.sound.f0_save = [];
    handles.sound.f0_corrected = f0;
    handles.sound.f0_corrected_save= [];
    handles.sound.tcalc = tcalc;
    handles.sound.selected_points =
false(size(f0));
    handles.sound.selected_points_save=
false(size(f0));
    handles.sound.block = block;
    handles.sound.step = step;

function handles = run_pitch_correction(handles)
    % Runs only if theres a sound clip
    if isempty(handles.sound.A)
        return
    end
    % Get variables out of the handles structure
    A = handles.sound.A;
    t = handles.sound.t;
    f0 = handles.sound.f0;
    f02 = handles.sound.f0_corrected;
    Fs = handles.sound.Fs;
    block = handles.sound.block;
    step = handles.sound.step;
    % Variables for size of analysis
    scale_factor = zeros(size(f0));
    A_corrected = zeros(size(A));
    N = length(f0);
    max_acorr_shift = 0;
    max_acorr_amp = 0;
    % Waitbar stuff
    hwait = waitbar(0,'Correcting pitch...');
    set(hwait,'Name','Please Wait');
    waitbar_count = 0;
    waitbar_update = round(N/10);
    pause(0.001);
    % Algorithm for correcting pitch
    I = 1;
    for n = 1:N
        % Update waitbar
        if waitbar_count > waitbar_update
            waitbar(n/N,hwait);
            waitbar_count = 0;
        end
        waitbar_count = waitbar_count + 1;
        % Pull out a chunk of the signal
        Atemp = A(I:I+block-1);
        ttemp = t(I:I+block-1);
        if f0(n) > 0 && ~isnan(f0(n))
```

```
% Calculate the scale factor by which to shift the
frequency
            scale_factor(n) = f02(n)/f0(n);
            % Interpolate
            tp = mean(ttemp) + (ttemp-
mean(ttemp)).*scale_factor(n);
            Ainterp = interp1(ttemp,Atemp ,tp)';
            Ivalid = find(~isnan(Ainterp));
            Ainterp(isnan(Ainterp)) = 0;
            Nperiod = ceil(1/(f02(n)/Fs));
        else
            % No frequency shift
            scale_factor(n) = 1;
            Ainterp = Atemp;
            Ivalid = find(~isnan(Ainterp));
            Nperiod =
ceil(1/(handles.pitch_options.Fmin/Fs));
        end
        if n == 1
            A_corrected(I:I+block-1) = Ainterp;
        else
            % Pull out one period of the new
waveform
            Achunk =
Ainterp(Ivalid(1):Ivalid(1)+Nperiod-1);
            factor = sum(abs(Achunk))*2;
            if ~all(scale_factor(n-1:n) == 1)
                % Start doing correlation
                max_acorr_amp = 0;
                max_acorr_shift = 0;
                for Nshift = -round(Nperiod/2)+
(1:Nperiod)
                    % Calculate makeshift
autocorrelation
                    acorr = 1-sum(abs(Achunk -
A_corrected((I:I+Nperiod-1) + Nshift +
Ivalid(1))))./factor;
                    if acorr > max_acorr_amp
                        max_acorr_amp = acorr;
                        max_acorr_shift = Nshift;
                    end
                end
            end
            [what,where] = min(abs(Achunk - ...
                A_corrected((I:I+Nperiod-1) +
max_acorr_shift + Ivalid(1))));
            A_corrected((I+where-
1:I+length(Ivalid)-1) + max_acorr_shift +
Ivalid(1)) = ...
                Ainterp(Ivalid(where:end));
        end
        I = I+step;
    end
    % Save values on handles
    handles.sound.A_corrected = A_corrected;
    handles.sound.scale_factor = scale_factor;
    % Close waitbar
    if ishandle(hwait)
        close(hwait);
    end

function [f0,acorr,Nshifts,Tshifts] =
find_f0_timedomain2(A,Fs,Fmin,Fmax,...
threshold_amp,threshold_ratio,harmonic_deviation,ta
rget_f0,target_tol)
    % This function calculates the fundamental
frequency of a signal in the time domain
    % OUTPUTS:
    %   f0:         The interpolated fundamental
freqency (Hz)
    %   acorr:      The vector of "autocorrelation"
values
    %   Nshifts:    The vector of shift indices
associated with the values in
    %               acorr
    %   Tshifts:    The vector of time shifts
associated with the values in
    %               acorr
    % INPUTS:
    %   A:          The input signal A(t), sampled
at an interval Ts
    %   Fs:         The sample frequency (Hz)
    %   Fmin:       (Optional) The minimum expected
frequency (Hz)
    %   Fmax:       (Optional) The maximum expected
freqency (Hz)
    %   threshold_amp: (Optional) The min autocorr
amplitude considered peak
    %   threshold_ratio: (Optional) The min ratio
between max autocorr peak and
    %                   a given peak to be
considered as a possible peak
    %   harmonic_deviation: (Optional) Tolerance
for looking for another peak
    %                   at half the frequency
of the highest
    %                   autocorrelation peak
    %   target_f0:  (Optional) The probable target
frequency
    %   target_tol: (Optional) The max error
between target_f0 and current freq
    %                   (0.1 = 10%)
    if nargin < 9 || isempty(target_tol)
        target_tol = 0.1;
    end
    if nargin < 8 || isempty(target_f0)
        target_f0 = [];
    end
    if nargin < 7 || isempty(harmonic_deviation)
        harmonic_deviation = 0.03;
    end
    if nargin < 6 || isempty(threshold_ratio)
        threshold_ratio = 0.7;
    end
    if nargin < 5 || isempty(threshold_amp)
        threshold_amp = 0.3;
    end
    if nargin < 4 || isempty(Fmax)
        Fmax = 800;
    end
    if nargin < 3 || isempty(Fmin)
        Fmin = 40;
    end
    f0 = 0;
    acorr = [];
    Nshifts = [];
    Tshifts = [];
    % Calculate index of min and max shift
    Nshift_min = floor(1/(Fmax/Fs));
    Nshift_max = ceil(1/(Fmin/Fs));
    if Nshift_max+Nshift_min > length(A)  ||
Nshift_min >= Nshift_max
        disp('Error in find_f0_timedomain2.m: Chunk
size must be larger!')
        return
    end
    % Pull out the chunk of the signal and
calculate a scale factor
    % The scale factor is the probable maximum
value
    Achunk = A(1:Nshift_max);
    scale_factor = sum(abs(Achunk))*2;
    % Calculate a vector of all shifts
    Nshifts = Nshift_min:Nshift_max;
    Tshifts = Nshifts / Fs;
    % Shift and calculate a makeshift
autocorrelation
    acorr = zeros(1,length(Nshifts));
    index = 1;
    for Nshift = Nshifts
        % Break out if we are shifting the chunk
beyond the end of the vector A
        if Nshift_max + Nshift > length(A)
            Nshifts = Nshifts(1:index-1);
            Tshifts = Tshifts(1:index-1);
            acorr = acorr(1:index-1);
            break
        end
        % Calculate makeshift autocorrelation
        acorr(index) = 1-sum(abs(Achunk - 
A([1:Nshift_max] + Nshift)))./scale_factor;
        index = index + 1;
    end
    % Try to make the autocorrelation "level" and 
with the minimum at zero
    acorr = acorr - 
polyval(polyfit(Nshifts,acorr,1),Nshifts);
    acorr = acorr - min(acorr);
    % Find a list of all of the peaks above the 
threshold
    [maxX,maxY,Imax,maxX_fit,peakX,peakY] = 
peakfind(Tshifts,acorr,5,1/Fmax,[],'',false);
    % Keep only those harmonics that are within a 
certain height of the largest
    % peak and whose height is above the threshold 
amplitude
    % Make a counter to keep track of which peaks 
remain after each criteria is
    % applied below...
    Ipeak = 1:length(maxX_fit);
    % Keep those whose heights are above the 
threshold
    Ipeak = Ipeak(maxY > threshold_ratio*max(maxY) 
& maxY > threshold_amp);
    maxX_fit = maxX_fit(Ipeak);
    % If only 1 or zero peaks remain, return
    N = length(Ipeak);
    if N < 2
        return
    end
    % Keep only those harmonics whose frequency is 
less than a certain
    % deviation around a multiple of the 
fundamental
    [maxX_fit,Ix] = sort(maxX_fit);
    Ipeak = Ipeak(Ix);
    possible_f0 = zeros(1,N);
    for n = 1:N-1
        if any(abs(1 - 
maxX_fit(n+1:N)./(2*maxX_fit(n))) < 
harmonic_deviation);
            if isempty(target_f0)
                f0 = 1./maxX_fit(n);
                Ipeak = Ipeak(n);
                break
            else
                possible_f0(n) = 1./maxX_fit(n);
            end
        end
    end
    % Now keep only the harmonic that is closest to 
the desired harmonic
    if ~isempty(target_f0)
        f0_error = abs(possible_f0./target_f0 - 1);
        [what,where] = min(f0_error);
        if what < target_tol
            f0 = possible_f0(where);
            Ipeak = Ipeak(where);
        end
    end
    % Now "fine tune" the frequency around the peak 
using a parabolic fit
    if f0 > 0
        p = 
polyfit(peakX(Ipeak,:),peakY(Ipeak,:),2);
        Xfit = -p(2)/2/p(1);
        f0 = 1/Xfit;
    end

function [maxX,maxY,Imax,maxX_fit,peakX,peakY] = 
peakfind(X,Y,Nmax,Xsep,Ymin,sortmethod,peakfit)
    % This function finds the maxima of the given 
function, and
    % returns the first Nvalues, sorted in 
decending order
    % INPUTS:
    %  X = a vector of X-values for the signal
    %  Y = a fector of Y-values for the signal
    %  Nmax = maximum number of values to return
    %  Xsep = the minimum acceptable separation 
between "peaks"
    %  Ymin = the minimum Y value to return
    %  sortmethod = how to sort the peaks...
    %               'maxY' = Sort in decending 
order starting with maximum Y-valued peak
    %               'minX' = Sort in ascending 
order starting with minimum
    %               X-valued peak
    %  peakfit = set to 'true' to do a parabolic 
fit and refine the maxX values
    % OUTPUTS:
    %  maxX = the X-coordinates of all of the N 
peaks
    %  maxY = the Y-heights of all of the N peaks
    %  Imax = indices of all of the N peaks
    %  maxX_fit = a more refined list of the X-
coordinates based on a parabolic
    %            fit to each peak
    %  peakX = a Nx5 matrix with each row 
containing the 5 X-values around
    %            each peak
    %  peakY = a Nx5 matrix with each row 
containing the 5 Y-values around each
    %            peak
    if nargin < 7 || isempty(peakfit)
        peakfit = true;
    end
    if nargin < 6 || isempty(sortmethod)
        sortmethod = 'maxY';
    end
    if nargin < 5
        Ymin = [];
    end
    if nargin < 4 || isempty(Xsep)
        Xsep = 0;
    end
    if nargin < 3 || isempty(Nmax)
        Nmax = 10;
    end
    % Differentiate, and find change in sign
    Ydiff = diff(Y);
    Imax1 = find(Ydiff(1:end-1) > 0 & Ydiff(2:end) 
< 0);
    Imax2 = find(Ydiff(1:end-2) > 0 & Ydiff(2:end-
1) == 0 & Ydiff(3:end) < 0);
    Imax3 = find(Ydiff(1:end-3) > 0 & Ydiff(2:end-
2) == 0 & Ydiff(3:end-1) == 0 & Ydiff(4:end) < 0);
    % Concatenate all of the possible peaks
    maxX = [X(Imax1+1) X(Imax2+1) X(Imax3+2)];
    maxY = [Y(Imax1+1) Y(Imax2+1) Y(Imax3+2)];
    Imax = [(Imax1+1) (Imax2+1) (Imax3+2)];
    % Re-sort if desired
    if strcmp(sortmethod,'minX')
        [maxX,IX] = sort(maxX,'ascend');
        maxY = maxY(IX);
        Imax = Imax(IX);
    elseif strcmp(sortmethod,'maxY')
        [maxY,IY] = sort(maxY,'descend');
        maxX = maxX(IY);
        Imax = Imax(IY);
    end
    % Remove any peaks that are below the min peak 
value allowed
    if ~isempty(Ymin)
        Irem = find(maxY < Ymin);
        maxY(Irem) = [];
        maxX(Irem) = [];
        Imax(Irem) = [];
    end
    % Remove any peaks that are closer than the 
minimum allowed peak seperation
    % in X
    if Xsep > 0
        Iremove = zeros(1,length(maxX));
        I = 0;
        for n = 2:length(maxX)
```

```
            if abs(maxX(n) - maxX(n-1)) < Xsep
                maxX(n) = maxX(n-1);
                maxY(n) = maxY(n-1);
                I = I+1;
                Iremove(I) = n;
            end
        end
        Iremove = Iremove(1:I);
        maxX(Iremove) = [];
        maxY(Iremove) = [];
        Imax(Iremove) = [];
    end
    % Remove any peaks beyond the max number to
return
    if length(maxX) > Nmax
        maxX = maxX(1:Nmax);
        maxY = maxY(1:Nmax);
        Imax = Imax(1:Nmax);
    end
    maxX_fit = maxX;
    peakX = zeros(length(Imax),5);
    peakY = zeros(length(Imax),5);
    % Do a 2nd-order poly fit around the peak to
pinpoint the peak location
    for n = 1:length(Imax)
        if Imax(n) > 3 && Imax(n) < length(X)-2
            I = Imax(n)+[-2:2];
            if peakfit
                p = polyfit(X(I),Y(I),2);
                maxX_fit(n) = -p(2)/2/p(1);
            end
            peakX(n,1:5) = X(I);
            peakY(n,1:5) = Y(I);
        end
    end

% --- Plot Functions
function handles = update_plot(handles)
    val = get(handles.view,'value');
    axes(handles.axes1);
    if val == 1
        if ~isempty(handles.sound.t) &&
~isempty(handles.sound.A)
            hplot = [];
            legends = {};
            if ~isempty(handles.sound.A_corrected)
                hplot(1) =
plot(handles.axes1,handles.sound.t,handles.sound.A,
'color',[0.7 0.7 0.7]);
            else
                hplot(1) =
plot(handles.axes1,handles.sound.t,handles.sound.A,
'color','b');
            end
            legends{1} = 'Original';
            if ~isempty(handles.sound.A_corrected)
                hold on
                hplot(2) =
plot(handles.axes1,handles.sound.t,handles.sound.A_
corrected,'color','b');
                legends{2} = 'Modified';
                hold off
            end
            if isempty(handles.status.Xlim)

set(handles.axes1,'Xlim',[min(handles.sound.t)
max(handles.sound.t)]);
            else

set(handles.axes1,'Xlim',handles.status.Xlim);
            end
            dA = max(handles.sound.A)-
min(handles.sound.A);

set(handles.axes1,'Ylim',[min(handles.sound.A)-
0.1*dA max(handles.sound.A)+0.1*dA]);
        else
            cla
        end
```

```
    elseif val == 3
        if ~isempty(handles.sound.tcalc) &&
~isempty(handles.sound.f0)
            hplot = [];
            f0 = handles.sound.f0;
            f0(f0 < 1) = NaN;
            if all(isnan(f0))
                cla
                hwarn = warndlg('No pitch detected
with current settings!','WARNING');
                waitfor(hwarn);
            else
                hplot(1) =
semilogy(handles.axes1,handles.sound.tcalc,f0,'.-
','color',[0.7 0.7 0.7]);
                %legends{1} = 'Original';
                if
~isempty(handles.sound.f0_corrected)
                    hold on
                    f02 =
handles.sound.f0_corrected;
                    f02(f02 < 1) = NaN;
                    hplot(2) =
semilogy(handles.axes1,handles.sound.tcalc,f02,'.-
','color','b');
                    %legends{2} = 'Modified';

                    if
any(handles.sound.selected_points)
                        f03 = f02;

f03(~handles.sound.selected_points) = NaN;
                        semilogy(handles.axes1,...

handles.sound.tcalc,f03,'.-','color','r');
                    end
                    hold off
                end
                grid on
                Ylim = [min(f0)/1.2 max(f0)*1.2];
                set(handles.axes1,'Ylim',Ylim);

set(handles.axes1,'Ytick',handles.scale_options.fre
qs);

set(handles.axes1,'YtickLabel',handles.scale_option
s.notes);
                if isempty(handles.status.Xlim)

set(handles.axes1,'Xlim',[min(handles.sound.t)
max(handles.sound.t)]);
                else

set(handles.axes1,'Xlim',handles.status.Xlim);
                end
            end
        else
            cla
        end
    end
    % Invisible axes for getting mouse information

set(handles.axes2,'Xlim',get(handles.axes1,'Xlim'))
;

set(handles.axes2,'Ylim',get(handles.axes1,'Ylim'))
;

set(handles.axes2,'Yscale',get(handles.axes1,'Yscal
e'));

set(handles.axes2,'ButtonDownFcn','AudioTuner(''mou
seclick'',gcbo,[],guidata(gcbo))');
    set(handles.axes2,'Color','none');
    axes(handles.axes2);

function nokeyresponse(hObject, eventdata, handles)
    % Makes nothing happen when you press a key
```

```
function mouseclick(hObject, ~, handles)
    % Executes when the mouse is clicked on the
plot
    % Executes only if pich correction is made
    if isempty(handles.sound.f0)
        return
    end
    % Executes only on graph 3
    if get(handles.view,'value') == 3
        clicktype = get(gcf,'SelectionType');
        if strcmp(clicktype,'normal') ||
strcmp(clicktype,'alt')
            point1 = get(gca,'CurrentPoint');    %
button down detected
            finalRect = rbbox;                   %
return figure units
            point2 = get(gca,'CurrentPoint');    %
button up detected
            point1 = point1(1,1:2);              %
extract x and y
            point2 = point2(1,1:2);
            p1 = min(point1,point2);             %
calculate locations
            offset = abs(point1-point2);         %
and dimensions
            xminmax = [p1(1) p1(1)+offset(1)];
            yminmax = [p1(2) p1(2)+offset(2)];
            if strcmp(clicktype,'normal')
                handles.sound.selected_points(:) =
false;
            end
            handles.sound.selected_points(...
                handles.sound.f0_corrected >
yminmax(1) & ...
                handles.sound.f0_corrected <
yminmax(2) & ...
                handles.sound.tcalc > xminmax(1) &
...
                handles.sound.tcalc < xminmax(2) &
...
                handles.sound.f0_corrected > 0) =
true;
        elseif strcmp(clicktype,'extend')
            % Save for undoing
            handles.sound.f0_save(end+1,:) =
handles.sound.f0;

handles.sound.f0_corrected_save(end+1,:) =
handles.sound.f0_corrected;

handles.sound.selected_points_save(end+1,:) =
handles.sound.selected_points;
            % Get initial click point
            point1 = get(gca,'CurrentPoint');    %
button down detected
            point1 = point1(1,1:2);
            handles.point1 = point1;
            handles.single_point = false;
            % If no points have previously been
selected, select the closest point
            if all(~handles.sound.selected_points)
                a = axis;
                tnorm = (handles.sound.tcalc -
a(1))/(a(2)-a(1));
                fnorm = (handles.sound.f0_corrected
- a(3))/(a(4)-a(3));
                xnorm = (point1(1) - a(1))/(a(2)-
a(1));
                ynorm = (point1(2) - a(3))/(a(4)-
a(3));
                [what,where] = min((tnorm-
xnorm).^2+(fnorm-ynorm).^2);

handles.sound.selected_points(where) = true;
                handles.single_point = true;
            end
            guidata(hObject, handles);
            pause(0.05);
            % Set function to track mouse position
            handles.tstart = now;
            guidata(hObject, handles);

set(gcf,'WindowButtonMotionFcn','AudioTuner(''wbmcb
'',gcbo,[],guidata(gcbo))');

set(gcf,'WindowButtonUpFcn','AudioTuner(''wbucb'',g
cbo,[],guidata(gcbo))');
        end
    end
    update_plot(handles);
    Scale_Callback(hObject, [], handles)
    guidata(hObject, handles);

function wbmcb(hObject, ~, handles)
    % Note: added in a timer so that the figure
won't try to refresh too
    % quickly and crash
    if (now-handles.tstart)*24*60*60 > 0.08
        handles.tstart = now;
        cp = get(gca,'CurrentPoint');
        ydiff = cp(1,2)/handles.point1(2);

handles.sound.f0_corrected(handles.sound.selected_p
oints) = ...

handles.sound.f0_corrected_save(end,handles.sound.s
elected_points).*ydiff;
        handles.sound.A_corrected = [];
        guidata(hObject, handles);
        update_plot(handles);
    end

function wbucb(hObject, ~, handles)
    if handles.single_point
        handles.sound.selected_points(:) = false;
    end
    guidata(hObject, handles);
    update_plot(handles);
    set(gcf,'WindowButtonMotionFcn','');
    set(gcf,'WindowButtonUpFcn','');

function [indices,freqs,notes,fund_index] =
get_scale(scale)
    indices = [];
    freqs = [];
    notes = {};
    fund_index = [];
    major_indices = [0 2 4 5 7 9 11];
    minor_indices = [0 2 3 5 7 8 10];
    fund_indices = [0 2 3 5 7 8 10];
    fund_notes = {'A' 'B' 'C' 'D' 'E' 'F' 'G'};
    scale_types = {'major','minor'};
    if nargin < 1 || isempty(scale)
        % Return all possiblities of scales
        indices = {};
        for n = 1:length(fund_notes)
            for m = 1:2
                indices{end+1} = [fund_notes{n}
scale_types{m}];
            end
        end
        return
    else
        fund_index =
fund_indices(strcmp(scale(1),fund_notes));
        if strcmp(scale(2:end),'major')
            indices = major_indices+fund_index;
        elseif strcmp(scale(2:end),'minor')
            indices = minor_indices+fund_index;
        else
            disp('ERROR in get_scale.m: unknown
scale!')
            return
        end
    end
    indices = [indices-12 indices indices+12
indices+24 indices+36];
    indices = indices(indices >=0 & indices <=48);
    [indices,freqs,notes] =
get_note_matrix(indices);

function [indices,freqs,notes] =
get_note_matrix(note)
    if nargin < 1
        note = '';
    end
    indices = (0:48)';
    freqs = 55.*2.^(indices./12);
    notes = {
        'A1'
        'A#/Bb1'
        'B1'
        'C1'
        'C#/Db1'
        'D1'
        'D#/Eb1'
        'E1'
        'F1'
        'F#/Gb1'
        'G1'
        'G#/Ab1'
        'A2'
        'A#/Bb2'
        'B2'
        'C2'
        'C#/Db2'
        'D2'
        'D#/Eb2'
        'E2'
        'F2'
        'F#/Gb2'
        'G2'
        'G#/Ab2'
        'A3'
        'A#/Bb3'
        'B3'
        'C3'
        'C#/Db3'
        'D3'
        'D#/Eb3'
        'E3'
        'F3'
        'F#/Gb3'
        'G3'
        'G#/Ab3'
        'A4'
        'A#/Bb4'
        'B4'
        'C4'
        'C#/Db4'
        'D4'
        'D#/Eb4'
        'E4'
        'F4'
        'F#/Gb4'
        'G4'
        'G#/Ab4'
        'A5'
        };
    if ~isempty(note)
        if ischar(note)
            I = find(strcmp(notes,note));
            if ~isempty(I);
                indices = indices(I);
                freqs = freqs(I);
                notes = notes(I);
            else
                indices = [];
                freqs = [];
                notes = '';
            end
        else
            note = note(note>=0 & note<=48);
            indices = note;
            freqs = freqs(note+1);
            notes = notes(note+1);
        end
    end

function handles = update_GUI(handles)
    % Change button if is recording or not
    if handles.status.isrecording
        set(handles.Play,'enable','off');
        set(handles.Record,'enable','off');
        set(handles.Stop,'enable','on');
    else
        if ~isempty(handles.sound.A)
            set(handles.Play,'enable','on');
        else
            set(handles.Play,'enable','off');
        end
        set(handles.Record,'enable','on');
        set(handles.Stop,'enable','off');
    end
    % AutomaticFit
    if isempty(handles.status.Xlim) &&
isempty(handles.status.Ylim)
        set(handles.AutomaticFit,'enable','off');
    else
        set(handles.AutomaticFit,'enable','on');
    end
```