General Introduction to AARAE

Audio and Acoustical Response Analysis Environment for Matlab

Densil Cabrera
Daniel Ricardo Jimenez Pinilla

Version 0.02
January 2014

Table of Contents

1 What is AARAE? 2

2 AARAE's authors and contributors 3

3 AARAE licence 4

4 Using AARAE: Quick start guide 5

4.1 Requirements 5
4.2 Installing and opening AARAE 5
4.3 Working with AARAE 5
4.3.1 Generating a test signal 5
4.3.2 Importing audio from a wav file 6
4.3.3 Importing audio from a mat file 6
4.3.4 Recording audio 6
4.3.5 Generating audio from a calculator 7
4.4 Processing audio in AARAE 7
4.5 Analysing audio in AARAE 8
4.6 Calibration 9
4.7 Editing audio 10
4.8 Saving audio and other data from AARAE 11
4.9 Deleting data from the AARAE workspace 11
4.10 Convolving `.audio` with `.audio2` 11
4.11 Playing and viewing audio from the main AARAE GUI 12

5 Examples of using AARAE 13

5.1 Measuring room acoustics 13
5.1.1 Measuring a calibration tone 13
5.1.2 Measuring background noise 13
5.1.3 Measuring a room impulse response 13
5.1.4 Calculating reverberation time 13
5.1.5 Calculating speech transmission index 14

6 Adding to AARAE 15

6.1 General aspects of adding code to AARAE 15
6.2 Name and Description 16
6.3 AARAE audio data format as input 16
6.4 Inputting additional audio 18
6.5 Creating a dialog box for input parameters 18
6.6 Calling other AARAE functions from your function 19
6.7 The format of output data 20
6.7.1 Figures, tables and other output 20
6.7.2 Tables 20
6.8 Adding a license 21
6.9 Adding the function to AARAE 22
6.10 Sharing your code 22

1 What is AARAE?

AARAE is a Matlab-based measurement, processing and analysis environment (incorporating a graphical user interface) for audio and acoustic system responses. It is intended primarily for use in education and research. AARAE is open source and is available at no cost.

AARAE is designed primarily for flexibility and extensibility, but also to provide a reasonably intuitive user interface. Rather restricting measurement and analysis to fixed procedures, AARAE aims to provide a variety of approaches. This flexibility is likely to be useful in both education and research: in education users can learn more if they can try more than one approach; and in research flexibility can be necessary when new, or non-standard, approaches to measurement and analysis are required. AARAE aims to be extensible by minimizing the additional coding required for adding working functions to AARAE. After a little additional coding, it is simply a matter of dragging and dropping a pre-existing function into the appropriate folder. Guidelines on how to add functions to AARAE are given in this document. The AARAE user interface is designed to support these features.

The AARAE GUI is likely to be convenient both to experienced Matlab users and beginners. For beginners it should provide a reasonably intuitive tool for measurement and analysis, which may spur interest in learning to use Matlab more directly. Of course, the underlying functions in AARAE can also be accessed directly from Matlab (without the GUI) for particular analysis needs (e.g. large scale data processing), and used in this way AARAE can be seen as a Matlab toolbox.

AARAE will always be a work-in-progress, as contributions are added and existing functions are refined. Hence it is always likely to have issues to resolve, work with, or work around. It should be very useful to people who want to try things out in the field of audio and acoustical measurements and associated analysis. It is an exploratory tool.

To check for AARAE updates, refer to http://aarae.org

2 AARAE's authors and contributors

The AARAE project was created by Densil Cabrera in mid 2013. The core code for AARAE was written by Daniel Ricardo Jimenez Pinilla.

At the time of writing, AARAE has various contributors, but we hope that the number of contributors will grow considerably in the future. Currently contributors include:
Grant Cuthbert (linear and non-linear reverberation parameter analysers)
Doheon Lee (STI, SII and loudness-based reverberation parameter analysers)

3 AARAE licence

AARAE uses the BSD 3-Clause license (see below).

Individual files and sets of files that are contributed to AARAE may have their own license (please check the relevant files).

If you are contributing a file to AARAE, it should preferably use the BSD 3-Clause license, but with the appropriate year, owner(s) and organisation (clause 3).

4 Using AARAE: Quick start guide

AARAE is distributed as a folder, containing Matlab functions and related documents in the main folder and subfolders.

4.1 Requirements

AARAE is **not** a stand-alone program. You must have Matlab to run AARAE. You will need Matlab's Signal Processing Toolbox to use AARAE satisfactorily, and other toolboxes are also used by particular functions (e.g. Statistics Toolbox and Curve Fitting Toolbox).

AARAE had been developed on Windows and Macintosh operating systems, using Matlab 2013b.

Plenty of memory is likely to be helpful in using AARAE.

4.2 Installing and opening AARAE

To install AARAE, put the main AARAE folder with all of its contents somewhere on your computer's disc (or other storage). It may help to not put the AARAE folder in your main Matlab folder, so as to avoid possible conflicts between function names on your Matlab path.

Open Matlab, and navigate to the AARAE folder.

Type `aarae` into the Matlab command window. This should start the graphical user interface (GUI).

You may notice that Matlab is 'busy' when AARAE is running, so it is important to terminate AARAE when you have finished using it. Do this by clicking on the 'Finish' button on the lower right-hand corner of the user interface.

4.3 Working with AARAE

The first step in working with AARAE is to acquire some data. The tools for acquiring data are available as buttons in the 'Start menu' of the GUI. Data can be acquired by:

(i) generating a test signal;
(ii) importing audio data from a wav file or a mat file;
(iii) recording audio; or
(iv) generating audio from one of the 'calculators' (not all calculators generate audio, but some do).

4.3.1 Generating a test signal

Various test signals can be generated by clicking on the 'Generate Audio' button. For example, a swept sinusoid can be generated using a function within the 'Sweeps' folder that can be accessed from the Generate Audio dialog box. In that case, an inverse filter of the sweep

is also generated, and stored in the same AARAE structure as the sweep – so that they can be used to measure an impulse response.

Generators are accessed via the 'Generate audio' button within the 'Start' section of the main AARAE GUI.

4.3.2 Importing audio from a wav file

Audio can be imported from a wav file by clicking on the 'load audio' button, and navigating to select a wav file. Once imported, the audio will appear in the 'Test Signals' section of the data tree on the left of the GUI. The audio waveform(s) and sampling rate are imported (and stored as fields within the AARAE audio structure).

To import audio (from a wav or mat file), click on the 'Load audio' button in the 'Start' section of the main AARAE GUI:

4.3.3 Importing audio from a mat file

Audio data that was previously saved by AARAE as a mat file is suitable for import. Such data is saved as a structure, and it must at least have the `.audio` field (containing the waveform), and should have other fields used by AARAE (especially `.fs` , `.nbits` and `.datatype`).

By using the AARAE field names, you can easily create a structure in Matlab (without using AARAE) that is suitable for saving as a mat file and being imported into AARAE.

An audio waveform that is not in a structure, saved in a mat file, can also be imported. In this case the sampling rate and bit depth need to be entered in a dialog box as the waveform is imported. Note that AARAE can accept waveforms in a 3-dimensional matrix: dimension 1 is for time, dimension 2 for channels, and dimension 3 for bands (e.g. audio filtered into octave bands).

4.3.4 Recording audio

Audio can be recorded using the computer's audio interface. The 'Record audio' button initiates an interface for recording (and playing, if audio is selected within AARAE when the button is clicked).

4.3.5 Generating audio from a calculator

Calculators are the miscellaneous category of extensible AARAE function, and some calculators generate audio. For example a calculator may generate an artificial impulse response, which could then be processed and analysed within AARAE in the same way as measured impulse responses. Audio from calculators is put in the 'Results' part of the AARAE data tree (on the left-hand side of the GUI).

To select and run a calculator, press the 'Calculators' button in the 'Start' section of the main AARAE GUI:

4.4 Processing audio in AARAE

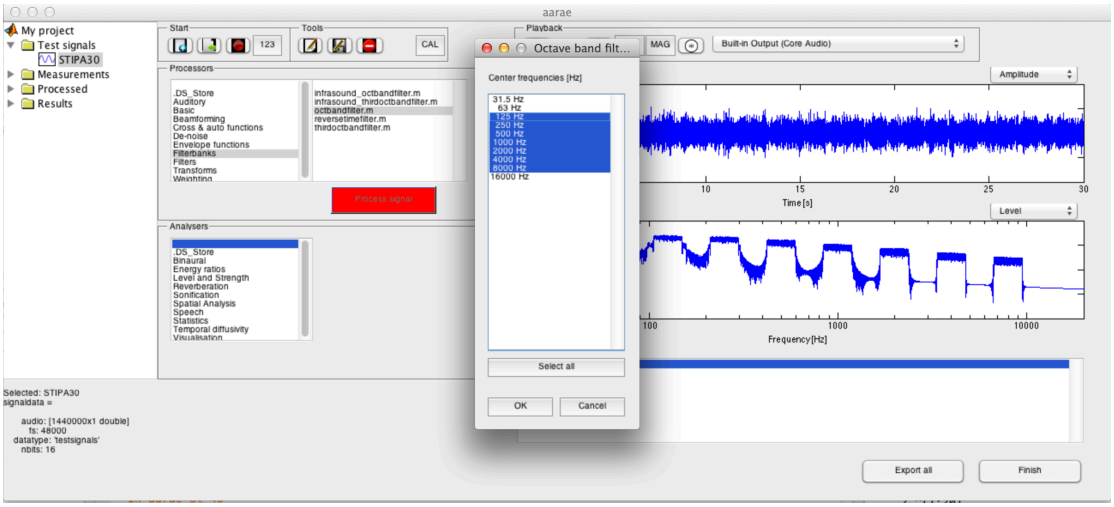
Once audio is present and selected in AARAE, 'Processors' and 'Analysers' are available for use. A processor performs an operation on the audio, deriving a new audio waveform (or waveforms). For example, processors exist to filter the audio, which in some cases could be useful prior to analysis.

Audio can be processed by:

(i) selecting the audio using the data tree on the left of the GUI; and then

(ii) selecting a processor, and clicking on the 'process signal' button.

Often processors have their own dialog box or boxes tor processing settings. The following figure shows one of the filterbank processors, with its list dialog box (in which particular octave band centre frequencies are selected prior to processing).



Note that some processors may be slow. AARAE does not have a progress bar display (since each processor is an independent Matlab function). If you are worried that nothing is happening, check the Matlab command line window for error messages, and wait a little. For slow processors, it could be helpful to get a feel for their timing by initially testing them on small audio files (short duration, small number of channels and bands, e.g. 1).

The output of processors is put in the 'Processed' part of the AARAE data tree (on the left-hand side of the GUI).

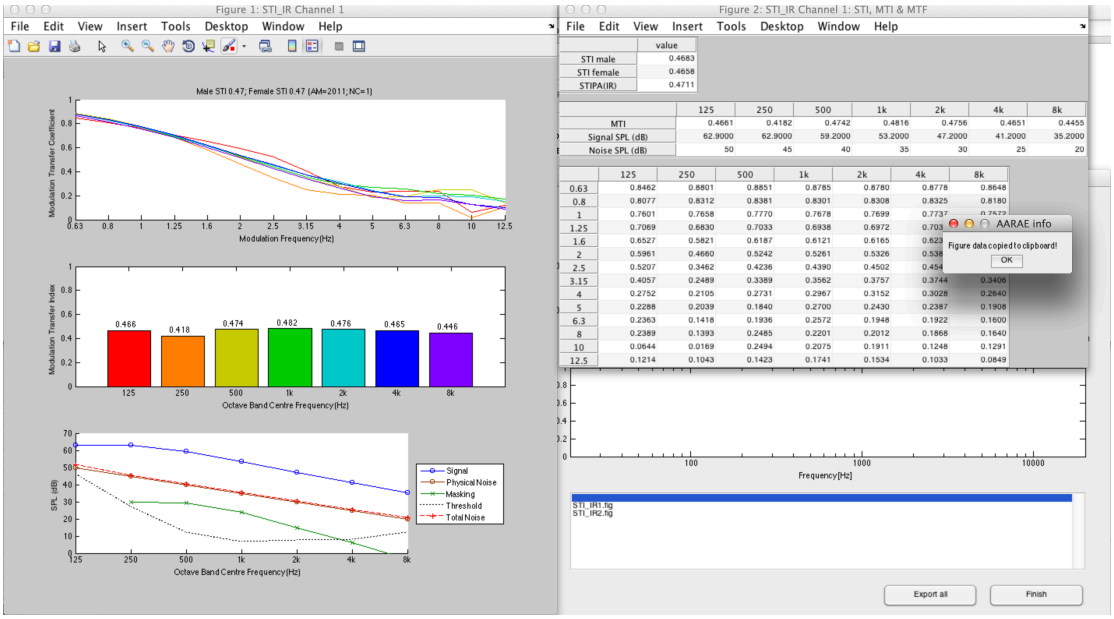
4.5 Analysing audio in AARAE

The analysers in AARAE are primarily concerned with providing graphical and/or numeric results that describe the features of the analysed audio. In some cases they may also provide audio output, although this is not their main purpose.
Analysing audio is done the same way as processing audio (except that an analyser is chosen instead of a processor).

Note that, like some processors, some analysers may be slow.

Most analysers produce figures, which can be edited, saved and exported like any other Matlab figure. When closed, these figures are stored by AARAE (while it is running), so that they can be recalled later. Numeric data from analysers can be in many formats, and at present AARAE does not necessarily provide a means of displaying it (e.g., if the dataset is large or multidimensional). Numeric data from analysers is saved as MAT files, which can then be accessed directly from Matlab. A well-written analyser should present the key results as figures (tables are also a type of figure in Matlab), and users can then examine the results further by accessing them directly in Matlab.

The following figure shows the output of an analyser (STI_IR), which includes charts and tables. Clicking on the background area of the table copies the table contents to clipboard (e.g., for pasting into a spreadsheet). The table and figure are kept within the AARAE workspace after they are closed, and can be regenerated by double-clicking on the appropriate item in the list in the box on the lower right of the AARAE interface.



4.6 Calibration

For some types of measurement and analysis, calibration is necessary. AARAE data structures can include a `.cal` field, which is an offset (or gain) in decibels that defines the relationship between the wave's amplitude and its level in decibels. The following equation expresses the relationship between amplitude (a), level (L) and the `.cal` field value:

$$L = 10\log_{10}(a^2) + \text{cal}$$

Level could refer to sound pressure level, or full scale level, or something else (that depends on how the `.cal` value is used – and what the purpose of the analysis is).

Note that when a waveform is calibrated in AARAE, no change is made to the waveform (it is not amplified or attenuated). Instead the `.cal` field is created (or changed) for potential use within processors and analysers. It is up to the writers of processors and analysers to use the `.cal` field where appropriate.

For sound pressure level, it is common for a calibration tone to be recorded on the measurement microphone. The calibration tone has a

known sound pressure level (e.g. 94 dB), and this information can be used to calibrate a sound recording that was made using the same gain structure (or a different gain structure for which the difference is known).

Calibration is done via the 'cal' button in the AARAE tools:

There are three ways of calibrating an audio recording in AARAE:

(i) If a recording in AARAE has a known level (e.g. sound pressure level), it can be used to calibrate other recordings (or indeed itself). In this case, the r.m.s. level of the calibration signal is entered by the user.

(ii) If an audio recording has been imported into AARAE from a wav file, then it is likely that the calibration tone is also a wav file – which can be used for calibration. In this case, the rms level of the calibration signal is entered by the user.

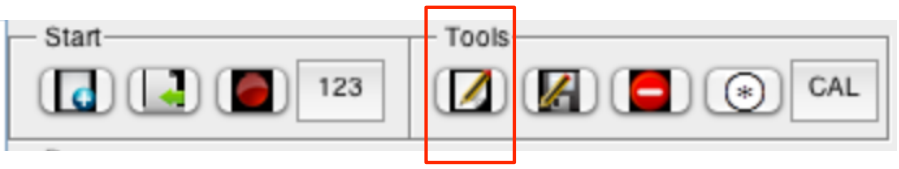
(iii) The value of the `.cal` field can be entered directly by the user.

In the case of multichannel recordings, either a single channel calibration tone, or a multichannel calibration tone (with the same number of channels as the audio being calibrated) may be used. A single channel calibration tone yields the same calibration offset on all channels, whereas a multichannel calibration tone allows for different offsets on each channel.

Note that calibration tones must be clean recordings (containing only the calibration tone). In some cases this may require editing of the calibration tone recordings prior to using them in AARAE (e.g., to remove silence and impulsive noise, and/or to align multichannel calibration tones).

4.7 Editing audio

AARAE is not an audio editor, but it is possible to do some minor editing of audio within AARAE. The edit audio button opens an interface to do operations such as truncation, gain change, time reversal, etc. The functions that are available within this window are those in the /processors/basic directory – and so it is a simple matter to add to the available functions. Operations performed within the editing interface can be undone while the interface is open (unlike when they are performed from the main AARAE interface).



4.8 Saving audio and other data from AARAE

Audio data may be saved using the 'Save signal' button either as mat or wav. However, multiband audio is not compatible with the wav format (because wav only has two dimensions and the multiband signal uses the third dimension).

All types of data can be saved using the 'Export all' button on the lower right of the main AARAE GUI. The data is saved within the 'Projects' folder of AARAE (in a sub-folder named by the user). Figures are saved as .fig files, which can be opened, edited and exported via Matlab.

A log file is automatically generated whenever AARAE is run, which records actions taken within AARAE. The log file is saved in the 'Log' folder of AARAE.

4.9 Deleting data from the AARAE workspace

Data may be removed from the AARAE workspace using the 'Delete signal' button.

4.10 Convolving `.audio` with `.audio2`

Some generators create an additional audio field, called `.audio2`. In some cases, `.audio2` is the inverse filter of `.audio`, and can therefore be used to derive impulse responses when `.audio` is used as a test signal. The convolve button in the tools section can be used for this purpose.

4.11 Playing and viewing audio from the main AARAE GUI

The main AARAE GUI includes two charts for visual display of audio. These are currently undergoing substantial development, and so are not described in this version of the documentation.

Audio may also be played using controls in the 'Playback' section of the GUI. These controls allow for normal playback, reverse time playback, random phase (steady state) playback, flat magnitude playback, and playing the selected audio convolved with REFERENCE_AUDIO.wav (which is in the 'Audio' directory). By default, this reference audio file is a short speech recording, but it can be replaced with another wav file with the same name if desired.

5 Examples of using AARAE

The following examples or tutorials will probably be written in 2014.

5.1 Measuring room acoustics

This example considers how measurements might be made of the acoustics of a room, taking reverberation and background noise into account, and calculating room acoustical parameters such as reverberation time and speech transmission index. Note that this example is about using the software, and so it does not deal with aspects of the measurement that are not concerned with using the software (e.g. setting up equipment, spatial variation).

5.1.1 Measuring a calibration tone

A calibrator is placed on the measurement microphone, and its signal is recorded by AARAE. It is best to start and stop the recording with the calibrator already in place and operating, so as to avoid noise in the recording. The sound pressure level of the calibrator (and, if relevant, the gain settings of the microphone preamplifier and interface) should be noted.
The calibration tone recording will appear in AARAE in the 'Measurements' part of the data tree.
The recording should be inspected, and if necessary, edited (truncated) to remove noise and/or silence.

5.1.2 Measuring background noise

The background noise in the room is recorded. The recording duration would depend on the characteristics of the noise (especially time-variance), but it might be 60 s if the noise is close to steady state.

The background noise recording can be calibrated using AARAE's 'CAL' button.

The octave band sound pressure level of the background noise can be calculated using XXXX.

5.1.3 Measuring a room impulse response

A test signal is generated – in this case the exponential sinusoidal sweep.
Once generated, the sweep is played and the result recorded.
The recorded sweep is transformed into an impulse response.
The impulse response is truncated.

5.1.4 Calculating reverberation time

The

5.1.5 Calculating speech transmission index

6 Adding to AARAE

This part of the AARAE documentation assumes some familiarity with Matlab. Nevertheless, if you are not familiar with Matlab, but are interested in writing AARAE code, it may not be too difficult if you use a combination of Matlab tutorials, Matlab help, using the templates in AARAE, looking at pre-existing AARAE calculators, generators, processors and analysers, and persistently and patiently trying code out using breakpoints where helpful.

6.1 General aspects of adding code to AARAE

AARAE is designed to make it relatively simple to add *calculators*, *generators*, *processors* and *analysers*. If you have previously written a Matlab function that fits into one of these categories, then it is probably quite simple to adapt it for AARAE. You will need to consider:

- The function's name and description;
- the input and output data formats;
- a dialog box for parameters (if relevant);
- the possibility of calling other AARAE functions from your function;
- figures and tables (if relevant);
- the license

Code that is added should either be written by the contributor, or have a license that allows it to be incorporated, used and redistributed (in such cases, the license must be included in the contribution).

In many cases, added code will consist of a single document (in an .m file), with one or more functions. This is a good approach for code that is not very complex. An alternative is to have a core function (a 'calling function') which calls other functions within a further subfolder. This further subfolder can be in the same folder as the calling function but will not be visible in the AARAE GUI. Hence the calling function's purpose is to interface between AARAE and the functions within the additional folder. This second approach is convenient for interfacing pre-existing toolboxes with AARAE, or for adding pre-existing code without changing it in any way (which may be a license requirement), or for adding complex analysers that are best stored using multiple files.

Regardless of which approach is taken the same issues in interfacing with AARAE need to be addressed.

AARAE has a folder called 'Templates', which contains templates for calculators, generators, processors and analysers. These, together with the guidance in the following sections, should make the writing of new content for AARAE relatively straightforward.

6.2 Name and Description

The name of the function (or calling function) is what a user sees within the GUI, so contributed functions need to have helpful names. In some cases, there may be the potential for several functions that do similar things (e.g. different ways of estimating reverberation time from an impulse response), and so the function name should be specific enough to accommodate the alternative functions. In other words, it is not a good idea to call a function 'reverbtime', but instead the name should have more particular information (such as the algorithm used, etc).

Within the GUI, hovering the mouse over the name of the function will bring up a description of the function (if one has been written). This description is the continuous comments within the function immediately following the function declaration. Please consider what text should be included there - such as a brief statement of what the function does, a reference to a publication (if relevant), and some guidance on how to use the analyser. It is a good idea to include the author, version and date in this text.

6.3 AARAE audio data format as input

Calculators and generators do not have audio input. Control of them via input parameters is outlined later in this document.

Processors and analysers do have audio input, and AARAE sends audio to them as a structure. **A processor or analyser for AARAE should have this structure as its first input argument.**

The following structure fields could be (but are not necessarily) used:

Field	Description and comment
`.audio`	This contains the audio waveform, varying in time down the columns. Dimension 2 is used for channels, and dimension 3 is used for bands (e.g. octave-band filtered data). Processors and analysers should be written to anticipate a potentially 3-dimensional waveform matrix, rather than returning an error if the input is multi-channel and/or multi-band.
`.audio2`	This is a complementary audio waveform, such as an inverse filter used to transform a recorded sweep into an impulse response. Some AARAE generators (but not all) create this field.
`.fs`	This is the audio sampling rate in Hz.
`.nbits`	This is the bit depth of the audio when playing, recording or writing audio to a wave file.

`.bandID`	This is a list of frequencies (or numbers of some type) associated with each band.
`.chanID`	This is a list of channel identifiers (e.g. angles in the case of beam-formed audio) - which has not been implemented yet.
`.cal`	This is a calibration offset in decibels such that `10*log10(wave.^2)+cal` yields the appropriate level (e.g., sound pressure level).
`.datatype`	This is a string describing the type of signal, which is used to determine in which part of the user interface tree the signal appears.

Other fields could also exist.

It is best practice to write a processor or analyser function to accept structure input, as well accepting as a set of discrete input arguments (which can be more convenient if the function is not being called via the GUI, for example if it is doing a large scale scripted analysis during which dialog boxes would be a nuisance). This can be achieved using an `isstruct()` test, for example:

```
function dosomething(in, fs, cal)
if isstruct(in)
       audio = in.audio; % audio wave data
       fs = in.fs; % audio sampling rate in Hz
      cal = in.cal; % calibration offset in dB
else
       audio = in;
end
% then do something with the data and metadata ...
```

In the above example, the second and third input arguments are ignored if the first input argument is a structure. However if the first input argument is a vector or matrix, it will be used directly as the audio waveform, and the second and third input arguments are required (notwithstanding further code to set default values).

Considering that the audio waveform might be 1, 2 or 3-dimensional, it may be important to be able to read its size. The following code provides this information:

```
[len, chans, bands] = size(audio);
```

Having obtained this information, we then need to decide what to do with it. Some analysers will work perfectly well on any number of channels and bands, while others may be designed only for 1 channel, 2 channels, 1 band, or some other constraints. One option is to return an error if the input is incorrectly dimensioned. Another option is to

truncate and/or mix the data as required. For example:

```
if bands > 1
% mix all bands together (removing the third dimension)
       audio = mean(audio,3);
       disp('Multiband audio has been mixed into a single band.')
end
```

or

```
 if chans > 2
        audio = audio(:,1:2,:); % use only the first two channels
        disp('Only the first two channels have been used.')
 end
```

Sometimes a field that would be used by the processor or analyser is not present in the input structure. For example, the calibration offset might not be present if the audio has not been calibrated. The following example shows how this can be dealt with.

```
 if isfield(in, 'cal')
       cal = in.cal; % get the calibration offset from the input structure
 else
       cal = 0; % set to default value
       % (but it would be more useful to get a value via a dialog box)
 end
```

6.4 Inputting additional audio

Sometimes a processor or analyser may require more than one audio signal as input. For example, a processor that convolves one waveform with another would need to access both waveforms to do this task.

AARAE has a utility function to facilitate this: `choose_audio` (it is in the Utilities folder). It can be run within a processor or analyser (or indeed a generator or calculator) simply by calling it – for example:

```
audio2 = choose_audio;
```

In the above example, `audio2` becomes a structure, with at least some of the fields previously described (including `.audio` and `.fs`). This audio can come from within AARAE, or can be imported from a file.

This also is the way for have audio input to calculators and generators (although normally calculators and generators do not have any audio input).

6.5 Creating a dialog box for input parameters

When running a calculator, generator, processor or analyser from the AARAE GUI, the user-controlled settings of the function need to be entered using a dialog box. Fortunately it is quite easy to add a rudimentary dialog box to a Matlab function, which is often all that is

required. Matlab's `inputdlg()` is one way of achieving this (refer to Matlab's help). The following is an example of a dialog box with three user-set parameters:

```
% Default values of three user-set parameters
startthresh = -20;
bpo = 1;
doplot = 1;

def = {num2str(startthresh), num2str(bpo), num2str(doplot)};

% Prompt for the three parameters:
prompt = {'Threshold for IR start detection (dB)', ...
          'Bands per octave (1 | 3)', ...
          'Plot (0 | 1)'};

% Title of the dialog box
dlg_title = 'Settings';

num_lines = 1;

answer = inputdlg(prompt,dlg_title,num_lines,def);


% Set function variables from the values returned from the dialog box
if ~isempty(answer)
          startthresh = str2num(answer{1,1});
          bpo = str2num(answer{2,1});
          doplot = str2num(answer{3,1});
end
```

In the code example above, the default values of the parameter variables are set prior to setting up the dialog box, so that if the box returns no values then the calculation can proceed with default values. In the subsequent code, further consideration should be given to the possibility of unexpected (or meaningless) values. The dialog box generated by the code is shown below:

Matlab has other simple options for input, such as `listdlg()` and `menu()`. Refer to Matlab's help for more information.

6.6 Calling other AARAE functions from your function

It is worth bearing in mind that any function in the AARAE folder and subfolders can be called from an AARAE function. However, while some functions are likely to be stable (they perform core functions),

others may evolve, which might cause problems for your code in the future.

The octave and 1/3-octave band filterbank functions are useful ones to call (and were written partly for this purpose). If a common sampling rate is used (44.1, 48, 96, 192 kHz), these will run faster than if you were to generate the filters within your own function.

6.7 The format of output data

For use within AARAE, data returned by the functions output argument(s) should be in a structure.

6.7.1 Figures, tables and other output

It is worth thinking about how the result of running the function can be best represented. In the case of analysers, charts are usually helpful, and other options include numerical tables, sonification, etc.

Sometimes a while loop could make it easier to work with multiple output options (so the user can choose whether or not to generate each output type, and indeed to discard the analysis results).

Data can be output to Matlab's command line window (e.g., by using `disp()`). You should not assume that a user will necessarily notice this, but it can still be a useful way of returning numeric data and messages.

6.7.2 Tables

Sometimes a table, or a collection of tables is a very useful representation of an analysis. While you can create and format your own tables relatively easily using Matlab's `uitable()`, usually trial and error is required to get the dimensions right. Therefore AARAE has a utility function included – called `disptables()` - to automatically place and dimension tables so that the tedious trial and error process can be mostly avoided. Tables generated using `disptables()` have additional functionality: by clicking on the grey background of the table, all of the table contents is copied to clipboard (tab-delimited) for possible use in another program (e.g. a spreadsheet).

Here is an example, creating three tables within a figure:

```
f = figure('Name',['STI_IR Channel ',num2str(ch),': STI, MTI & MTF'], ...
'Position',[200 200 630 540]);
%[left bottom width height]

dat1 = MTF_ch;
cnames1 = {'125', '250', '500', '1k', '2k', '4k', '8k'};
rnames1 = {'0.63', '0.8', '1', '1.25', '1.6', '2',...
           '2.5', '3.15','4','5','6.3','8','10','12.5'};
t1 =uitable('Data',dat1,'ColumnName',cnames1,'RowName',rnames1);

dat2 = [MTI(ch,:);Lsignal(ch,:);Lnoise(ch,:)];
```

```
cnames2 = {'125', '250', '500', '1k', '2k', '4k', '8k'};
rnames2 = {'MTI', 'Signal SPL (dB)', 'Noise SPL (dB)'};
t2 =uitable('Data',dat2,'ColumnName',cnames2,'RowName',rnames2);


dat3 = [M_STI(ch);F_STI(ch);STIPA(ch)];
cnames3 = {'value'};
rnames3 = {'STI male','STI female','STIPA(IR)'};
t3 =uitable('Data',dat3,'ColumnName',cnames3,'RowName',rnames3);

disptables(f,[t3 t2 t1]);
```

6.8 Adding a license

Contributed code in AARAE should include a license, because this makes clear how the code can be used.

AARAE uses the BSD 3-Clause license for the project as a whole, and we encourage contributors to use the same license so as to minimise license conflicts.

To use the BSD 3-Clause license, add the following text to your Matlab function as a comment. Change the YEAR, OWNER and ORGANISATION fields as appropriate. The license comment could be at the end of the m file.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Copyright (c) YEAR, OWNER
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are
% met:
%
%  * Redistributions of source code must retain the above copyright notice,
%    this list of conditions and the following disclaimer.
%  * Redistributions in binary form must reproduce the above copyright
%    notice, this list of conditions and the following disclaimer in the
%    documentation and/or other materials provided with the distribution.
%  * Neither the name of the ORGANISATION nor the names of its contributors
%    may be used to endorse or promote products derived from this software
%    without specific prior written permission.
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
% "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
% TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
% PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
% OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
% EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
% PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
% PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
% LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
% NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
% SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

If your contributed code uses a different license, please ensure that the

license is easily found by a user, and that the license is not incompatible with that of AARAE. In particular, the license must permit redistribution.

6.9 Adding the function to AARAE

Specific aspects of preparing calculators, generators, processors and analysers are described further on in this document. Once all aspects have been dealt with, it may be the time to add the function to AARAE.

To add the function to AARAE, it is simply a matter of putting it in the appropriate folder. It must be in a subfolder within the 'Calculators', 'Generators', 'Processors' or 'Analysers' folder (as appropriate). If there is not an appropriately named subfolder, one can be created (but please exercise restraint – we don't want too many categories).

Make sure that the analyser works as intended (both as a stand-alone function, and via the AARAE GUI). If you are hoping that other people will find the analyser helpful, make sure that it is adequately documented (at least by comments within the m file).

If you wish to add further documentation, it can be added to the 'Documents' folder.

6.10 Sharing your code

If you would like to contribute code to the main AARAE distribution, the current method is to send it to Densil Cabrera (densil.cabrera@sydney.edu.au).
If you are likely to be a regular contributor, then it would be best to access the AARAE source repository on Google Code (via svn).
https://code.google.com/p/aarae-source/source/checkout