Design of the Auditory Modelling Toolbox

December 31, 2015

1 How to start

- amtstart: start in demo mode, presents cached results (where previously precalculated), display all messages. Corresponds to amtstart('verbose','download');. Use flag 'redo' in cases where the data need to be calculated again.
- Messages:
  - 'silent': supress all messages. Nice for usage in other toolboxes.
  - 'documentation': supress the messages about calculation progress. Used when creating the documentation.
  - 'verbose': display all messages
- Calculation of experiments and data:
  - 'normal': use local cache. If locally not available, download precalculated results from the remote cache at the internet. If not available at the internet, recalculate the data. This is the default.
  - 'cached': Never recalculate: use the cache (local cache or if locally not available, download precalculated results from the remote cache at the internet). If not available, throw an error.
  - 'localonly': Use the local cache and recalculate if not available. Never connect to the remote cache at the internet.
  - 'redo': always redo. Overwrite the local cache when done.

2 Structure

2.1 Models, demos, experiments, and data (update me!)

- Demo must run without requiring user input, neither from the keyboard or from calling them as functions. This is because the documenation system must be able to run them in the background.

- A demo can be a function, as long as the function can be run without taking any inputs, see above.
- A demo, experiment or example in a function may not use the Matlab GUI or the waitbar function: The documentation system cannot handle this in the background.
- You cannot load data like "load mydata", because you cannot assume anything about the path. All loads must use the full path to the function. This can easily be done by using "mfilename" (see the code for examples) or "amtbasepath"

2.2 Directory structure

`arg` Directory with default parameters of various AMT functions.

`auxdata` Auxiliary data used by AMT functions. Auxdata are data too large to be provided within the AMT package and will be downloaded from the internet when requested by a function. Auxdata are structured by models and can be accessed from any function. The loading and storage are controlled by amtload and amtsave.

`cache` Data cached for later. Some results need much time to be calculated. They can be stored here and re-loaded when requested again. Cached data are structured by the functions creating the data and can be accessed by the creator only. Cache is controlled by amtcache.

`comp` Computational routines. They are all prefixed by `comp_` Computational subroutines does not need to check input arguments, they never take a variable number of input arguments, and in many cases they are shadowed by a C/Mex/Oct implementation found in directoroes mex and/or oct. Currently, also default parameters for some models/functions are stored in this directory as well. The functions providing the default parameter have the prefix arg_

`demos` Simple demos describing how to use the toolbox. The demos usually generates some plot, which are automatically put on the homepage. The are prefixed by demo_.

`experiments` Descriptions of experiments that reproduce results presented in a paper (figure, table, etc). The functions providing the results are named by the corresponding model, have the prefix exp_, and use the figure number as parameter.

`monaural` Monaural models

`binaural` Binaural models

`filters` Functions with auditory filters are stored here.

`general` This directory contains general functions that cannot be placed elsewhere.

`hrtfs` HRTFs and other spatially oriented data are stored here. They are structured by models and use SOFA as file format. HRTFs are handled by the SOFA API and can be loaded with SOFAload.

`humandata` Function for creating or loading recorded data (exception: audio signals, see the directory signals). The model related functions are prefixed by data_.

`mat2doc` Files related to the automatic documentation generator (mat2doc).

`mex` Matlab files shadowed by `.c` and `.cpp` files which can be compiled by Matlab's `mex` command for faster processing.

`oct` C++ (`.cc`) files which can be compiled by gcc for faster processing in Octave. The corresponding Octave files (.m) must be stored in the mex directory.

`plot` Model-related plot functions

`signals` Functions for creating/loading audio signals.

`speech` Speech and speech perception related models.

`src` Source codes of models which require compilation.

`testing` Simple test scripts to verify that new and old code produce the same result, and to perform other tests. Each test function is prefixed by `test_` A test function takes no input, and returns a single variable indicating how many tests failed.

3 Coding standards (update me!)

3.1 Goals of the software

- High quality code: Easy to read and maintain
- High quality documentation in the code: This is the best place to keep the documentation, as it is easier to keep up to date when the code changes. The documentation can be extracted in HTML and TEX formats.
- Simplicity: Eliminate unneccesary configuration options. Always choose the simplest possible solution. This makes it much easier to read and modify the code.

3.2 Data in general

It is generally assumed that all data is a regular sampling of a continuous phenomena at a certain sampling frequency `fs`. Many routines will therefore ask for both a signal and a sampling frequency. By the same reasoning, frequencies are usually specified in Hz because we assume the sampling frequency to be known (so don't do normalized frequencies between 0 and 0.5 as Matlab sometimes does).

3.3 Definition of the input to the model

If nothing else is noted, this should be the description of sound input to any model:

An acoustical signal is representated by a column vector of numbers. The numbers are obtained by sampling the air pressure of the acoustical signal at a constant sampling rate. The numbers are scaled such that an acoustical signal with a level of 100 dB SPL corresponds to a digital signal with an RMS value of 1.

3.4 General data structures

It is probably impossible to list the relevant data structures before developing the software, but the few ones listed here should always be used.

- A single channel signal is a column vector.
- A multi channel signal is a column matrix.
- A multi-channel signal stemming from a filterbank has time as first dimension, channel number as second, and original signal channel (left/right) as third dimension.

3.5 Specific data structures

- The output from the modulation filterbank has time as first dimension, frequency channel number as second, modulation channel number as third and original signal channel (left/right) as fourth.

3.6 Matlab specific coding standards

3.7 C and Fortran specific coding standards

- Variables name are allowed to be both lower and upper case. This convention is called *camel casing*, see http://en.wikipedia.org/wiki/CamelCase. The general rule is that the word starts with a lower case letter, and then the following words are starts will upper case letters.

3.8 Function names

- All function names in Matlab must be lowercase. This avoids a lot of confusion because some computer architectures respect upper/lower casing and others do not. Furthermore, function names are traditionally written in uppercase in Matlab documentation.
- Models are named after the paper in which they first appeared, as in dau96, or after their commonly accepted name if theres is no doubt as to which model is the name refers.
- *As much as possible*, functions are named after the function they perform, rather than the algorithm they use, or the person who invented it.
- It is not allowed to use underscores in function names. They are reserved for structural purposes, i.e. as in `demo_gammatone` or `test_dau96`.
- If a model consist of several files, the must all be prefixed by the model name, as in `takanen2013mso` and `takanen2013lso`.

3.9 Variable names

- Variable should not have the same name as an already existing function is Matlab. This makes the code easier to read and less prone to errors. However, as almost all short names are taken by a Matlab function, this can be hard to obtain in practice.
- No global variables. Global variables makes it harder to debug, and the code cannot be parallized.
- Never use `i` or `j` as a variable name in Matlab, as they are used for the imaginary unit. This creates a great deal of confusion when reading other peoples code. Please use `ii` and `jj` instead, or something completely different. Using `i` and `j` are allowed in C, which does not have an imaginary unit.

The following is a list of common variables.

`insig` Input signal

`outsig` Output signal

`inoutsig` Some simple functions modify the signal in place, so the signal is both input and output. This may save some memory and processing time. Please write 'insig' and 'outsig' in the documentation, as the user should not care about this detail.

`fs` Sampling frequency.

`siglen` Length of signal

`nsigs` Number of signals

`fc` Center frequency/frequencies of filter/filter-bank.

`flow,fhigh` Generic low, high frequencies determining a range of frequencies.

`a,b` Filter coefficients to IIR filters.

3.10 The init file

The `mydirinit.m` gets executed when the main upstart routine `amtstart` is run. The upstart routine adds the directory `mydir` to Matlab's search path before executing the routine.

The `mydirinit` script must define a variable called `status` and set it to 1 (one) if the toolbox should be loaded. Any other value of status will case the `mydir` directory to be removed again from the search path.

This simple mechanism enables you to check if everything it correct, and refuse loading the directory if something is not.

Similarly, it is possible to set a variable `module_version` to any number, and this will be the version number of the module. Otherwise, the global version number will be used.

So in the most common case, the `mytoolboxinit.m` file will contain just a single line:

```
status=1;
```

4 Auxiliary data

Data which are neither code or algorithm, or which are simple to large are not be provided within the AMToolbox release file. Examples for such are signals, audio files, calibration parameters, larger sets of model parameters, or experimental results. In the AMToolbox, these data are called auxiliary data and stored in a separate directory.

The directory with auxiliary data is handled by `amtauxdatapath`, which is per default pointing to the directory `auxdata` in the AMToolbox.

If locally not available, the auxiliary data will be automatically downloaded from the internet. The URL of the auxiliary data is handled by `amtauxdataurl`, which is per default pointing to `http://www.sofacoustics.org/data/amt-$version$/auxdata`, with `$version$` as the version of the AMToolbox release.

4.1 Loading

The auxiliary data are structured by the model. For each model, the data are stored in files. The auxiliary data can be accessed from any function by calling `amtload` with the model name and the filename as parameters, e.g. `mydata = amtload('majdak2010','data.mat');`

`amtload` handles data differently depending on the type of the file:

- `.mat`: amtload will load the MAT-file and return its content in a single variable, similarly to Matlab's function load
- `.wav`: amtload will load the WAV-file and return two parameters: signal, and sampling rate.

4.2 Saving

When a function creates data which is intended to be accessed as auxiliary data, the following statement can be used:

```
save(fullfile(amtauxdatapath, 'mymodel', 'myfile.mat'), 'myvariable1',
'myvariable2', ..., 'myvariableN');
```

In order to provide the auxiliary data for download from the internet, contact the developer team.

5 Caching the data

Some models take a very long time to complete, and it is therefore advantegous to cache the result after a first run. In order to provide a uniform mechanism for handling this problem, AMToolbox provides the `amtcache` function. `amtcache` allows to store variables in packages and return the package if requested later. The following example illustrates the caching:

- Ask for variables `x, y, z`, stored in the package `'fig1'`:

  ```
  [x,y,z]= amtcache('get', 'fig1');
  ```

- Check whether the variables have been loaded:

  ```
  if isempty(x)
  ```

- Within the if-statement, because `x` could not be loaded, do the computation of the variables, e.g.:

  ```
  x = 12;
  y = rand(10,100);
  z = 'my funny string';
  ```

- Save the variables in cache as a package, for example `'fig1'`:

  ```
  amtcache('set', 'fig1', x, y, z);
  ```

- Close the if-statement

  ```
  end
  ```

- The variables `x,y,z` are now in your workspace. Process as usual.

Note the following:

- Each functions has its own packages. The packages are stored in the directory `cache` of the AMToolbox. Write access must be granted.
- In `'get'` and `'set'`, the order of the variables must be the same. We encourage, however, to use a single structure instead of multiple variables.
- All variables will be loaded when requrested for a package.

Optionally, the mode of cache operation can be changed by using a `flag`, e.g., `[..] = amtcache('get', [..] , flag);`. There are several cache modes:

- `'normal'`: package will be recalculated when locally not available.
- `'redo'`: enforce the recalculation of the package. `[..] = amtcache('get', [..])` outputs empty variables always.
- `'cached'`: enforce using cached package. If the cached package is locally not available, it will be downloaded from the internet. If it is remotely not available, warning will be thrown and the package will be recalculated. Note that this method may by-pass the actual processing and thus does not test the corresponding functionality. It is, however, very convenient for fast access of results like plotting figures. On the internet, the cached packages are available only for the models from release version of the AMToolbox.
- `'global'`: use global AMToolbox settings for the cache mode. The global settings are defined at the start of the AMToolbox.
- If the flags should be provided by the user and `ltfatargehlper` is used to parse the flags, the amtcache flags can be obtained by `definput.import={'amtcache'};`. After `[flags,keyvals] = ltfatarghelper({},definput,varargin);` the amtcache flags will in `flags.cachemode`.

amtcache supports other commands:

- `'clearAll'`: clears the cache directory. This command asks for a confirmation.
- `'getURL'`: returns the internet URL of the cache.
- `'setURL'`: sets the internet URL of the cache.
- `'remove'`: removes a package from the cache. Parameter: Package. NA yet.
- `'delete'`: deletes all packages of a function from the cache. Parameter: Token encoding the function. NA yet.

Furter examples on usage of cache mechanism are provided in `test_amtcache`.

6 Structure of a human data file (update me!)

```
function [outx,outy]  = data_soendergaard2099(varargin)
%DATA_SOENDERGAARD2099  This is the headline that describes the function
%   Usage: [outx,outy] = data_soendergaard2099()
%
%   `[outx,outy]=data_soendergaard2099(flag) returns data points from the Soendergaard
%   of:
%
%     'noplot'        Don't plot, only return data. This is the default.
%
%     'plot'          Plot the data.
%
%     'figXXX'        Put in a description of the output data.
%
%     'figYYY'        Put in a description of the output data.
%
%   Examples:
%   ---------
%
%   Figure 5 can be displayed using :::
%
%     [out1,out2] = data_soendergaard2099('fig5','plot');
%
%   Figure 7 can be displayed using :::
%
%     [out1,out2] = data_soendergaard2099('fig7','plot');
%
%   References: soendergaard2099universe
definput.flags.plot = {'noplot','plot'};
definput.flags.type={'figXXX,'figYYY'};
[flags,keyvals]  = ltfatarghelper({},definput,varargin);
if flags.do_figXXX
  outx = [0, 1.3750, 1.3750, 1.6250, 1.8750, 2.7083];
  outy = [0.4513, 0.7500, 0.2500, 0.4590, 0.3436, 0.5159];
  if flags.do_plot

    % Put the visualization of the data here
  end;
end;
% Do the same for the next figure
if flags.do_figYYY
  ...
end;
```

7 Anchors

An anchor is a capitalized word in the code, that can be extracted by a script. Each file should contain the following anchors:

AUTHOR The line following this anchor indicates who the authors are.

For debugging it is possible to insert XXX, FIXME, BUG, TODO into the code, followed by a description of the problem.