A modular framework for running psychoacoustic experiments and computational perception models

AFC for Mathwork's MATLAB
Version 1.40.0
July 05, 2014

Author: Stephan Ewert
Email: stephan.ewert@uni-oldenburg.de
www.aforcedchoice.com
www.medi.uni-oldenburg.de/afc/

Development was conducted partly during my time at:

Medizinische Physik
Universität Oldenburg

Centre for Applied Hearing Research
Technical University of Denmark

Research Lab of Electronics
Massachusets Institute of Technology

Special thanks to:

Torsten Dau, Andrew J. Oxenham, Ralph Peter Derleth, Jesko L. Verhey, Jörg Damaschke, Oliver Fobel, Darren Reed, and Birger Kollmeier

Contents

I. Introduction

AFC is a psychoacoustic measurement toolbox for Mathworks MATLAB offering a modular framework for entirely free configurable measurement setups, including the alternative forced choice procedure, the constant stimuli method, interleaved setups, and customizable language settings. AFC can be used and modified for own adaptive and non-adaptive psychophysical measurements which can be derived from a set of examples. In general, an experiment requires the definition of three MATLAB m-files (called m-files in the following), in which the entire configuration and the signal generation and/or processing is scripted. In most cases, starting a measurement will open a response window guiding the subject through the experiment. The subject can be presented with a sound stimulus or a sequence of sound stimuli referred to as a trial. When a trial presentation is finished, the subject is asked to respond to a certain question, e.g., which of three sounds was different. The response is collected by use of buttons on the keyboard, mouse interaction with the response window, or by using a touch screen. Completion of a measurement is indicated in the response window. The experimental results and all information required to reconstruct the entire measurement are automatically saved to disc. AFC can perform the same experiment for different parameters or different experiments in specific predefined order (referred to as a session) for individual subjects. Data is automatically logged such that experiments can be interrupted, split into different sessions, and continued at the last exit point.

AFC can also interface with perceptual models to perform psychoacoustic experiments in exactly the same framework as the measurements with human listeners. Example models are provided.

AFC follows a modular structure with many modules, e.g., the response window, the messages, the sound output command, and the measurement procedures, being overloadable. Thus users can customize AFC to a maximum degree of freedom in order to meet the requirements of their experiments, e.g., by designing a new response window, by customizing messages displayed to the subject, or by integrating arbitrary third-party sound output commands.

II. General remarks

A. What is new in AFC 1.40.0

If you are using AFC for the first time, you might want to skip this paragraph.

For users of previous versions, there are a number of structural changes in the AFC code and some new additions that should be mentioned. Nevertheless, the current AFC can still be used as any previous version.

- p-files for afc_main.m and afc_control.m are gone, all files are included as human readable m-files, allowing for a better understanding of AFC.
- A new top-level function afc.m was added allowing conduction of newly added experimental sessions or single experiments as before; alternatively afc_main.m can still be used as before.
- Support for custom measurement procedures was added (not documented yet). AFC's officially supported stock procedures remain in afc_control.
- A new logic to interface to custom sound commands was added (see addons\sound_exampleAudioplayer.m)
- In the examples the structure set was renamed to setup

- A free, AFC-specific version of Hörtech's powerful SoundMexPro is included for exclusive use with the AFC package. If you want to benefit from SoundMexPro's full feature set or if you want to indirectly support AFC, I encourage you to get a licensed version of SoundMexPro from www.hoertech.de.
- AFC is now licensed under Creative Commons (CC BY-NC-ND 4.0) for its core files (in folders `\base` and `\gui` or with `afc` in the file name) and under (CC BY 4.0) for the remaining files which can be modified by the user, see Sec II.C.

B. What to do and not to do

AFC is a measurement and modeling tool for "lab" use and as such users are encouraged to add or derive their own experiments, dig into the code and come up with new suggestions, new ideas or even complete own measurement procedures. These changes and own derivations as well as sharing them publicly are explicitly granted by AFC's license, regarding most content of the software distribution.

Changes to AFC's core files (including all files in the `\base` and `\gui` folders and all files with `afc` in the file name) are not permitted by AFC's license (see Sec. II.C.). Independent of the license, it is strongly recommended to not touch any of these files in order to keep compatibility with future releases. If you encounter problems, or if you have suggestions for improvements of AFC's core files, please report them to the author. This will also make sure that all users can benefit from improvements in future versions.

In order to let others benefit from your own derived experiments or other addons, it is recommended to stay as close as possible to AFC's structure and conventions which become obvious from the included examples.

AFC has gained a lot from third-party suggestions and bug reports in the past 15 years and the goal is to continue improving the software. Whenever you find bugs or feel that you could significantly contribute with a new suggestion, please contact the author.

C. License

Unless otherwise stated, the AFC distribution, including all files and the distribution file itself is licensed under Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International (CC BY-NC-ND 4.0; http://creativecommons.org/licenses/by-nc-nd/4.0). In short, this means that you are free to use and share (copy, distribute and transmit) the original AFC distribution "as is" giving credit to the author and in a noncommercial way.

All files inside the `\base` and `\gui` folders and all files with `afc` in the file name, including the distribution package are licensed under CC BY-NC-ND 4.0. You may not alter, reverse engineer, transform, re-pack, or build upon these files.

All files outside the `\base` and `\gui` folders and except for files with `afc` in the file name, including all experiment files, model files, calibration files, interface files to extern hardware and custom sound commands, measurement procedure files, their interfaces and their structure, as defined by the included examples, are licensed under Creative Commons Attribution 4.0 International (CC BY 4.0; http://creativecommons.org/licenses/by/4.0).

Users are thus explicitly permitted to create custom experiments, models, calibration files, and measurements procedures or to derive them from included examples, and to distribute their own creations independent of AFC provided they are attributed as custom content for AFC and acknowledge the original author.

D. How to cite AFC

If you have used AFC in you want to give credits to the author or reference him in your scientific work, you should cite AFC in the following way:

Ewert, SD. (2013) “AFC - A modular framework for running psychoacoustic experiments and computational perception models,” in Proceedings of the International Conference on Acoustics AIA-DAGA 2013, Merano, Italy, pp. 1326-1329.

E. Disclaimer

No warranties. To the maximum extent permitted by applicable law, the software is provided "as is" and the author Stephan Ewert and its suppliers disclaim all warranties, either express or implied, including, but not limited to, implied warranties of fitness for a particular purpose.

No liability for consequential damages. To the maximum extent permitted by applicable law, in no event shall Stephan Ewert be liable for any special, incidental, indirect, or consequential damages whatsoever (including, without limitation, hearing damage, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use the software, even if Stephan Ewert has been advised of the possibility of such damages.

In short, AFC is provided “as is” and you use at your own risk. If you melt your computer, trash your headphones, or even worse, damage your hearing, it is your own responsibility.

F. Credits

I would like to thank the many people that beta tested earlier versions of AFC, found bugs or made suggestions for improvements. In particular I would like to thank the following people for their valuable contributions to AFC and their support:

Torsten Dau &Andrew Oxenham	- for the confidence to use my first “one-weekend version” even before I used it
Andrew Oxenham	- for many valuable suggestions
Jörg Damaschke	- for suggestions and discussion on features
Ralph Peter Derleth	- for improvements of the initial version of the SIM (model) procedure
Oliver Fobel (geb. Wegner)	- for help with MATLAB issues/improvements
Jesko Verhey	- for improvements (afc_show)
Thomas Ulrich Christiansen	- for danisch message translation
Christian Fuellgrabe	- for french message translation
Daniel Berg (Hoertech)	- for continued support with soundMexPro
Darren Reed	- for suggestions on the documentation

III. Installation

A. System and hardware requirements

AFC requires Mathwork's MATLAB version 5.3 or higher on Windows, Mac OSX or Linux. The toolbox requires no specific hardware. Sound output uses the internal MATLAB sound commands in the default settings and can be configured to use third-party sound output tools for MATLAB.

B. Installing the AFC package for MATLAB

Unzip the AFC distribution to your MATLAB directory or a directory of choice (e.g, `c:\usr\mylab`). Unpacking has to be performed including folder names. This will create the subdirectory `\afc` containing all required files and subfolders.

To use AFC, change the operating path in MATLAB to the `\afc` subdirectory.

If you want to use the included free version of soundMexPro exclusively with AFC, unpack the password protected archive `soundmexpro4afc` located in the folder `\soundmexpro`. For details and the license see `soundmexpro4afc.txt`. Make sure a subfolder `\bin` was created in folder `\soundmexpro`.

C. Configuring AFC

A configuration window to setup AFC and to select the default language is automatically opened the first time upon using the AFC package in MATLAB or if the setup was not finalized before. This window also allows users to optionally select different versions of soundmex (www.soundmexpro.de) to be used on Windows systems with AFC. For using MATLAB's internal sound do not select any of the soundmex versions.

The setup can be manually opened at any time by typing the following command in the MATLAB command window:

```
afc('setup')
```

AFC temporally adds all required folders to the MATLAB search path. The required folders are defined in `afc_addpath.m`.

D. Calibration of sound output level

In the default calibration used by the example experiments, AFC assumes a sound pressure level (SPL) of 100 dB SPL reached for a digital stimulus with an amplitude of 1 (full scale, FS) equal to 0 dB FS. This assumption usually avoids overly loud stimulus levels when trying the examples on average, non-calibrated hardware. **Note that sound output might still be at a very high level and has the potential to damage your ears**. It is, therefore, recommended to first turn down the volume of your system and to increase it during AFC's sound output to moderate levels. A detailed description how to adapt AFC's calibration files to your system in provided below in the calibration section.

IV. Quick start guide

The AFC distribution comes with a number of example experiments in the folder `\experiments`. The examples can be used as a guideline for designing own experiments. Examples are given for a basic adaptive threshold tracking (`example_*.m`), the estimation of a psychometric function (`exampleConstantStimuli_*.m`), or a matching experiment with interleaved runs, where different stimulus configurations are tested simultaneous (`exampleInterleaved_*.m`). In addition, `exampleCalscript_*.m` is a given as an example how to use the inbuilt calibration features. The basic example experiment (`example_*.m`) is described here.

It is assumed that you are familiar with the basics of psychoacoustic measurement procedures, particularly alternative forced-choice tasks. The procedure is not described in detail here. In this document the following terminology will be utilized:

- The term "**run**" or "**measurement run**" refers to the estimation of a single data point, i.e., from the start of a measurement until a result for a single experimental configuration is gained. This might be the detection threshold for a tone in a specific experimental condition, derived using adaptive tracking, or the corresponding psychometric function derived with the method of constant stimuli.
- An actual "**measurement**" or "**session**" might consist of multiple runs, e.g., estimating the detection threshold for a tone three times at several different frequencies (independent variable) distributed over two sessions on two different days.
- The variable of the experiment which is adjusted during an experimental run, is referred to as "**experimental variable**" or "**tracking variable**". The outcome of the experiment, e.g., a threshold estimate gained from the values of the experimental variable during performing the experiment is the dependent variable of the experiment.

A. Starting the example experiment

The example experiment estimates the modulation-detection threshold for a sinusoidal amplitude modulation at 8 Hz and 16 Hz (independent variable) in two consecutive runs applied to a broadband Gaussian noise carrier. An adaptive, 3-interval 3-AFC method (1-up-2-down rule) is used. There are two ways to start the example. Either by using AFC's top-level function (recommended) in the command window:

```
afc('main','example','subjectName','condition');
```

or (like in previous AFC versions) by directly using AFC's main command:

```
afc_main('example','subjectName','condition');
```

If `afc_main` is used, `afc_addpath` has to be executed first to add all required paths to MATLAB's search path. The top level function `afc` has further functionality which will be explained later and automatically adds the required paths. If the `afc` function is called with the first argument 'main' as shown above, it will essentially call the `afc_main` function as stated above, after having updated the MATLAB search path.

Thus, identical in both function calls are the arguments 'example', which is the name of the experiment, 'subjectName', which is the name or ID of the listener (e.g., 'JD78' for John Doe born in 1978), and the experiment dependent 'condition', which can be 'cd1' or 'cd2' in case of the example experiment. With these both conditions the same experiment is performed

at two different overall stimulus levels (With ’cd2’ the overall level is 10 dB lower than with ’cd1’). Before staring the experiment make sure the sound output volume is adjusted to a reasonable level at your system.

If AFC is started the first time, a setup window will open (see III.C). This window allows users to optionally select different versions of soundmex (www.soundmexpro.de) to be used on Windows systems with AFC. For using MATLAB’s internal sound do not select any of the soundmex versions, select your preferred language and press ‘ok’.

AFC opens a modal response window during the run of the experiment. Instructions are displayed guiding you to the experiment. Please follow the instructions by pressing the number of the interval on your keyboard in which the amplitude modulation was presented. After finishing a run, the next measurement can be started by pressing the ’s’ button on your keyboard or the whole measurement can be terminated by pressing the ’e’ button. After having terminated AFC, the experiment can be started again and AFC will continue with the next run. After finishing two threshold runs, the example measurement is completely finished. Now, only the end button ’e’ can be used to end the experiment. If you attempt to start the experiment again using one of the two function calls described above, AFC will remember that the specific experiment is finished for the specific subject. The response window will only offer ’e’ to end the session.

You are able to terminate AFC at any time by closing the response box. In this case, only the currently running and unfinished measurement run is lost. Usually this should only be done during development. Subjects should not be advised to terminate the measurement by closing the response window during an experimental run.

a. A quick view at the results

AFC stores the experimental results in files with a name determined by the parameters passed to the AFC function call. For the example experiment, the ASCII file

```
example_subjectName_cd1.dat
```

has been written to current directory if an experimental run was finished. The ASCII file contains the measurement results in the following format:

```
%   exppar1[Hz]   expvar[dB]   std(expvar)[dB]
   8.00000000   -6.50000000   0.50000000
```

In the first column, the value for the independent variable for which the experiment was performed is given. The independent variables are referred to as `exppar1 … expparN` in AFC. The example experiment has only one (`exppar1`) being the modulation rate in Hz. The second column holds the dependent variable of the experiment, in this case mean value at threshold of the experimental variable (`expvar`) and the third column its standard deviation.

After each finished run a new line with the results is appended. Examples how to import the files in MATLAB are given in `\docs\dataImport.txt`

b. Protocol and control file

AFC has automatically written two further ASCII files to the same directory:

```
example_protocol.pro
control_example_subjectName_cd1.dat
```

The first file is referred to as protocol file in the following and is generated by AFC for each given experiment. All information required to reconstruct the measurement is appended as a new item at the end of the protocol file after finishing an experimental run. After finishing one experimental run for the example, the protocol file `example_protocol.pro` looks something like:

```
% standard protocol file generated by matlab afc procedure
%
% new entry (date/experiment/exppar1[Hz]/signalpos/response/expvar[dB]/measurement points/mean+std)
31-5-2013 14:50:59
example_subjectName_cd1
8.00000000
2  1  1  2  2  1  3  2  3  1  2  1  2  3  1  2  2
2  1  2  2  2  2  3  2  2  1  2  2  2  3  2  2  2
-6.00000000    -6.00000000    -10.00000000    -6.00000000    -6.00000000    -8.00000000    -6.00000000
-7.00000000    -6.00000000    -7.00000000    -6.00000000    -7.00000000    -6.00000000
-6.50000000   0.50000000
```

The second file is referred to as control file in the following and is generated individually for each unique set of parameters passed to the `afc_main` function. For each results file, there will always be a control file with the same name except for the header ‘control_’. The control file contains a list of (combinations of) the independent variables for which the experiment is going to be performed for each subject. AFC uses the control file to track which conditions are already finished and were to continue an experiment when AFC is started again. If an experiment is started for a new subject for the first time, the control file is written based on the configuration of the experiment. After having started the example, the configuration file `control_example_subjectName_cd1.dat` looks like:

```
% run#
% track1: exppar1;
       1
      8.00000000
     16.00000000
```

Below two comment lines, the first integer entry in this file is the current run number of the experiment. The entries below the counter are listing the independent variables for which the experiment is performed in consecutive runs (top-bottom order). The counter serves as an index to the list of values of the independent variable at which the experiment should be continued on next start. After a successful completion of a run, the counter is incremented. To add a new run or to change the order of runs, the value of the independent variables can be edited or new lines can be added to the list.

Note: if a control file for a specific experiment and subject name exist, AFC retrieves the values for the independent variables (referred to as `exppar1 … expparN`) from this file and does not write a new control file.

B. Setting up your own experiment

This section tells you how to configure the necessary files to run your own experiment. It can be useful to first try the example in the previous ‘Quick Start’ section to better understand particular settings of interest. This section assumes basic knowledge of MATLAB.

a. The experiment files

Each AFC experiment is defined by three files, in case of the example experiment by the three files:

`example_cfg.m`
`example_set.m`
`example_user.m`

These three files with the ending ‘_cfg’, ‘_set’, ‘_user’ are mandatory for any experiment in AFC. The files have the following function:

The experimental configuration file `example_cfg.m` contains the whole basic configuration of the experiment stored in the AFC specific structure ‘def’. Here the measurement procedure, the number of intervals, settings for the measurement procedure, the path where to write the results, the duration of the stimuli, the samplerate, etc. are defined.

```
% general measurement procedure
def.measurementProcedure = 'transformedUpDown'; % measurement procedure
def.intervalnum = 3;                            % number of intervals
def.repeatnum = 1;                              % number of repeatitions of the experiment

% tracking variable (dependent variable)
def.startvar = -6;                              % starting value of the tracking variable
def.expvarunit = 'dB';                          % unit of the tracking variable
def.expvardescription = 'AM depth';             % description of the tracking variable

% limits for tracking variable
def.minvar = -100;                              % minimum value of the tracking variable
def.maxvar = 0;                                 % maximum value of the tracking variable
def.terminate = 1;                              % terminate execution on min/maxvar hit: 0 = warning,
def.endstop = 3;                                % Allows x nominal levels higher/lower than the limits

% experimental variable (independent variable)
def.exppar1 = [8 16];                           % vector containing experimental parameters for which
def.exppar1unit = 'Hz';                         % unit of experimental parameter
def.exppar1description = 'AM frequency';        % description of the experimental parameter

% experimental variable 2...N (independent variable)
% add here if required

% interface, feedback and messages
def.afcwin = 'afc_win';                 % overload response window
def.windetail = 1;                      % displays additional information in the response window
def.mouse = 0;                          % enables mouse/touch screen control (1), or disables (0)
def.feedback = 1;                       % no feedback
def.messages = 'default';               % message configuration file, if 'autoSelect' AFC
def.markinterval = 1;                   % disable all button marking

% save paths and save function
def.result_path = '';                   % where to save results
```

```
def.control_path = '';                  % where to save control files
def.savefcn = 'default';                % function which writes results to disk

% samplerate, number of channels and sound output
def.samplerate = 44100;                 % sampling rate in Hz
def.bits = 16;                          % output bit depth: 8 or 16 see def.externSoundCommand for 32
```

Only the configuration variables required for the example experiment are defined in the config file. AFC has many more variables stored in ‘def’ which are automatically set to their default values. Defaults and explanations for all possible configuration variables are given in `docs\configurationVariableList.txt`.

The setup file `example_set.m` is a function and is executed once at each start of a new measurement run. It is typically used to setup condition dependent variables and to pre-generate stimuli that are not linked to the experimental variable of the experiment and thus do not change during a measurement run. The config structure `def` and the AFC specific structure `work` are declared global at the beginning. In `def` configuration variables defined in the config file can be accessed. `work` is initialized and updated by AFC and holds a number of variables that change during the experiment or that were passed to the `afc_main` function call, e.g., `work.condition` holding the last argument with which `afc_main` was executed (in this case ‘cd1’ or ‘cd2’). A third optional global structure `setup` is declared global in the example that is used as a container for signals generated here.

```
function example_set

global def
global work
global setup

% make condition dependend entries in structure set

switch work.condition
case 'cd1'
   setup.level=1;
case 'cd2'
   setup.level=0.25;
otherwise
   error('condition not recognized');
end

% define signals in structure setup

setup.modsine = sin([0:def.intervallen-1]'*2*pi*work.exppar1/def.samplerate);
setup.hannlen = round(0.05*def.samplerate);
setup.window = hannfl(def.intervallen, setup.hannlen, setup.hannlen);
```

The user file `example_user.m` is a function and is executed prior to each stimulus presentation. The stimuli which are presented and are dependent on the tracking variable in the consecutive presentation intervals of a measurement run are generated here. The same three globals as in `example_set.m` are declared. The experimental variable is `work.expvaract`, and is automatically updated by AFC. Configuration variables are again retrieved from the `def` structure and pre-generated signals can be accessed from the setup structure. It is mandatory that the user file provides or updates the four fields `work.signal`, `work.presig`, `work.postsig`, and `work.pausesig` from which AFC assembles the sound output.

```
function example_user

global def
global work
global setup

% 1000 ... 4000 Hz bandlimited Gaussian noise
% The experiment is meant to be 3-interval 3-alternative forced choice, so we need 3 signals.
% One signal holds the target amplitude modulation.
% work.expvaract holds the current value of the tracking variable of the experiment.

tref1 = real(ifft(scut(fft(randn(def.intervallen,1)),1000,4000,def.samplerate))) * setup.level/10;
tref2 = real(ifft(scut(fft(randn(def.intervallen,1)),1000,4000,def.samplerate))) * setup.level/10;
tuser  =  real(ifft(scut(fft(randn(def.intervallen,1)),1000,4000,def.samplerate)))  *  setup.level/10  .*
...
   (1 + (10^(work.expvaract/20) * set.modsine));

% Hanning windows
tref1 = tref1 .* set.window;
tref2 = tref2 .* set.window;
tuser = tuser .* set.window;

% pre-, post- and pausesignals (all zeros)
presig = zeros(def.presiglen,2);
postsig = zeros(def.postsiglen,2);
pausesig = zeros(def.pauselen,2);

% make required fields in work
work.signal = [tuser tuser tref1 tref1 tref2 tref2];    % left = right (diotic) first two columns
work.presig = presig;                                   % must contain the presignal
work.postsig = postsig;                                 % must contain the postsignal
work.pausesig = pausesig;                               % must contain the pausesignal
```

b. Modifying the experiment

At this point the very basics of AFC are explained and you can start modifying the example experiment or deriving an own experiment from the provided examples.

In the example experiment, the independent variable ’exppar1’ has a value of 8 and 16 Hz and the measurement is performed a single time as defined by `def.repeatnum` in `example_cfg.m`. Both values of the parameter are presented in random order (`def.parrand = 1`). Based on these definitions, the control file was written when AFC was first started. The control file might be edited with a text editor or in the MATLAB editor (‘Open as text’) to repeat `exppar1 = 8` a further time or to add `exppar1 = 32`:

```
% run#
% track1: exppar1;
       1
      8.00000000
     16.00000000
      8.00000000
     32.00000000
```

When AFC is now started again, these two further measurement runs will be added and the response window indicates the updated overall and remaining number of measurement runs.

To develop an own experiment, the user is advised to save the example experiment files under a new name and to derive the experiment from there.

C. Setting up an experimental session

Experimental sessions are predefined lists of different experiments or experimental conditions which can be run in AFC. An experimental session is defined in a session file located in the folder `\session`. An example is provided in example `session_example.m`. A session is started by:

```
afc('session','sessionName','subjectName');
```

D. The example experiment and example model

The folders `\experiments` and `\models` hold a number of different example experiments and models using a variety of features provided by AFC. You might want to run the example experiment with a model instead of a human subject. Use the command: `afc_main('example','exampleModel','cd1')`; In this case the configuration file `exampleModel_cfg.m` tells AFC that the "subject" 'exampleModel' is a model and does not need a response window nor sound output. It is also possible to run a model with activated response window and sound output. In this case, the model response is indicated by highlighted buttons. User interaction via mouse or keyboard is disabled. This feature might be useful for demonstration purposes. Please browse to the respective m-files for more information. Generally, the configuration file `subjectName_cfg.m` is used for subject specific configurations, independent whether the subject is human or model. In case of human subjects this might be the language setting of the response window (e.g. `def.language = DE` for German).

Please have a look at the other examples and models to become familiar with AFC.

V. General structure and functionality of AFC

Figure 1 shows a block diagram of AFC's basic structure. AFC is executed from the MATLAB command window using the top-level function `afc.m`. Two modes of operation are available, either starting an experimental session or a single experiment. An experimental session is a list of (different) experiments that is to be conducted by a given subject. For each of the experiments in a session, the number of experimental runs to be conducted (each yielding, e.g., a threshold estimate) can be specified. The session is defined by an ASCII text file (`session_sessionName.dat` in Fig. 1). After having started an experiment as part of a session or directly, AFC's core function `afc_main.m` handles the initialization by loading a number of configuration parameters in a hierarchical order (see block "Configuration" in Fig. 1). If the subject is defined to be a computer model (in `subject_cfg.m`), the "Model module" is loaded and configured. AFC then calls the configuration function of the experiment (`expName_cfg.m` in block "Experiment definition") and enters its main loop (`afc_work.m`) to conduct the experiment. Depending on the configuration, a response window is opened informing the subject about the just started experiment. Additionally, the sound card and optional external hardware (e.g., software controllable amplifiers) are initialized as indicated by the block "Hardware & interfaces". Upon the subject's response, an experimental run of the experiment is started calling the setup function of the experiment (`expName_set.m`) once, and `expName_user.m` repeatedly after each response until the experimental run is finished. The measurement procedure is managed in `afc_control.m` and updates the experimental (or tracking) variable which is used in `expName_user.m` to generate the stimuli. Once an experimental run is finished, the data is written to the ASCII text file `subject_expName_boothID.dat` and for safety a new entry in the ASCII protocol file `expName_protocol.pro` is appended. Upon initial start of a given experiment for a given subject, a list (of all combinations) of the independent variables for which the experiment is to be conducted was written to disk as an ASCII file, `control_subject_expName_boothID.dat`. The automated generation of this control file depends on the independent variables of the experiment and their configuration defined in `expName_cfg.m`. Alternatively to automatic generation of the control file, it can be manually generated if specific requirements exist for order and combination of independent variables within or across subjects (e.g., Latin squares). The progress of the experiment is stored in the control file after each completed experimental run. This makes it possible to stop and resume the experiment at any time, or to append certain experimental conditions for which the experiment has to be repeated at a later time.

In most cases, the AFC user is only concerned with the block “Experiment definition” to design an own experiment, with the block “Configuration” for, e.g., calibration, with the inspection of the experimental results, or the optional definition of a session file, and in some cases with manual modification of the control file. The experiment can, e.g. overload the default response window (`afc_win`) or a custom sound command can be used.

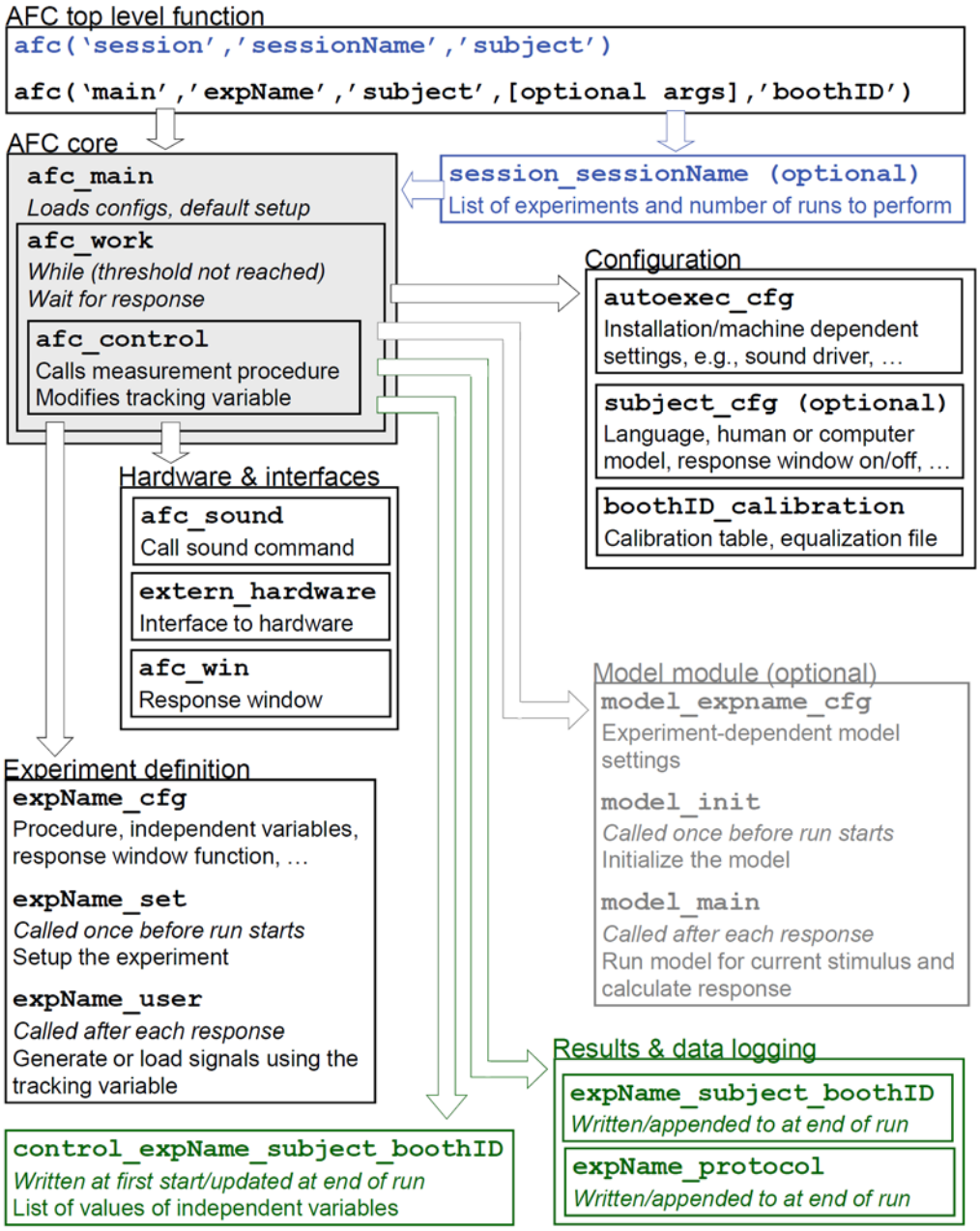

Figure 1: Block diagram of the AFC software framework. AFC is started by the top-level function `afc.m` with loads the AFC core routines for conducting an experiment or optionally a session. The AFC core loads all configurations and opens the response window as well as the hardware interfaces (sound commands, optional third-party hardware functions). The experiment files are loaded, the experiment is initialized, and performed depending on either human responses or responses by the optional computer model. At completion of an experimental run, results are written to hard disc in human readable text (ASCII) format, including all information required to reconstruct the measurement. After each experimental run, the experiment can be continued, or terminated and resumed at any time.

VI. Experiment design

AFC offers a range of example experiments which can be used as a starting point for the design of own experiments. For an own experiment (named, e.g., `myExp`) three m-files have to be generated as shown in the block "Experiment definition" of Fig. 1: `myExp_cfg.m`, `myExp_set.m`, `myExp_user.m`. Optionally, depending on the degree of customization, further optional m-files (see Sec. VIII): For a customized response window `myExp_win.m`, for customized messages and button labels `myExp_msg.m`.

Typically the definition or modification of the configuration file from one of the examples is the first step. Among others, in `myExp_cfg.m`, the measurement procedure, the starting value, units, and valid range of the experimental variable, as well as the independent variables for which the experiment is to be conducted are defined. The experiment can have any number of independent variables (typically only a few are used). These can be defined in `myExp_cfg.m` allowing an automated conduction of the experiment for given independent variables and repetitions. The list of values of the independent variables for which the experiment is to be conducted is written to the control file on first start of the experiment. Alternatively, values for independent variables can be passed in the function call as optional arguments (see Fig. 1). If optional arguments are passed in the function call, the naming of the control and results file will automatically include the optional arguments.

The second step is the definition of the setup file `myExp_set.m`. Typically all calculations, and generating or loading stimuli that only depend on the independent variables, and thus will not change during the run of the experiment, are performed here. For example, `myExp_set.m` might load one of a number of 30-s wav files with a name defined by the value of the independent variable for which the current experiment run is performed. As another example, often a vector containing a window function with raised-cosine on and off ramps is defined here, which is then applied to the stimuli in `myExp_user.m`.

The third file, `myExp_user.m`, contains the main stimulus generation of the experiment and returns those stimuli to the AFC core. Typically, a number of reference stimuli are generated (e.g., in case of forced-choice experiment) and a target stimulus depending on the current value of the experimental variable. This current value is automatically updated and passed to `myExp_user.m` by AFC during the experiment run and the user file is executed after each response of the subject. Typically, stimuli are generated "on the fly" in AFC, meaning that the MATLAB code in the user file contains expressions to calculate the stimuli. In some cases stimuli are loaded from disk, or random parts of a long-duration, pre-loaded file, like the 30-s wav file mentioned in the above example of `myExp_set.m`, are cut and assigned as, e.g., masker to the reference and target stimuli.

VII. Computer model module

AFC can optionally retrieve responses from a computer model, e.g., `myModel`. In this case, `myModel` is passed as subject name in the `afc.m` function call. `myModel_cfg.m` has to exist and has to define "subject" `myModel` as computer model. Optionally it disables sound output and the response window. For teaching purposes, AFC can also run a model with sound output and response window enabled, in which case the button pressed by the model can be highlighted. The computer model `myModel` itself is defined by at least three further m-files as indicated in the block "Model module" of Fig. 1: `myModel_expName_cfg.m`, `myModel_init.m`, `myModel_main.m`.

The first file `myModel_expName_cfg.m` can be optionally used to overload the general configuration from `myModel_cfg.m` depending on the experiment (here with name `expName`).

`myModel_init.m` is comparable to the setup file of an experiment and should hold all MATLAB code that is executed once prior to an experimental run. For example, filter coefficients for a filter bank used in the model are calculated.

`myModel_main.m` is the counterpart of the user file of an experiment and is called by AFC after the stimuli are generated. The model's main function has to pass the model's response to AFC, replacing the human response.

VIII. Customization

In most cases, the definition of an experiment by the described three files offers sufficient freedom to the user and the many configuration variables of AFC assessable in the configuration file cover most requirements. A list of all configuration variables and their explanation is provided in `docs\configurationVariableList.txt`. Further customization is possible by overloading default AFC functions (e.g., the save function which writes results to disc) with own derivations of the original functions. The two most commonly overloaded default functions allow for:

A. Customization of the response window

AFC's default response window function `afc_win.m` automatically shows a horizontal row of response buttons, according to the number of intervals defined in the configuration of the experiment. Buttons are labelled by numbers as defined in the default (localized) message file `default_EN_msg.m`. In addition, message and status boxes are shown in the response window depending on configuration, and optionally start and end buttons (for, e.g., touch screen operation) are shown. The window function to be loaded is specified in the configuration file of the experiment and should be replaced by `myExp_win.m`, if an experiment-dependent customization is required. The example experiments `exampleIdentification` and `exampleCustomized` use custom response windows which can be used as a reference for own creations.

B. Customization of messages and button labels

AFC's default messages and button labels are available in four languages (Danish, English, French, German) and are defined in localized message files, e.g., `default_EN_msg.m` in AFC's base folder. Other languages can be easily added by generating the according localized message files. Experiment-specific messages and button labels should be defined in a custom message file `myExp_msg.m`. If an experiment-specific message file exists, AFC automatically overloads the default message file if `def.messages = 'autoSelect'`is stated in the experiment's configuration file. Custom message files can also be localized, e.g., `myExp_DE_msg.m` for German. If localized messages do not exist for a specific language, AFC falls back to `myExp_EN_msg.m` and alternatively to the unlocalized message file `myExp_msg.m`. The example experiments `example3I2AFC`, `exampleConstanStimuli`, `exampleIdentification`, `exampleInterleaved`, and `exampleCustomized` have custom message files from which own creations can be derived.

C. Customization of the data save function

AFC's default function to save the measurement results depends on the measurement procedure and is specified by `def.savefcn = 'default'` in the experiment configuration file. For the transformed up-down procedure AFC uses `\base\adaptive_mean_savefcn.m` as default. If a custom save function should be used, `def.savefcn` has to be changed accordingly. In `exampleCustomized` the custom save function `exampleCustomized_savefcn.m` is used as specified by `def.savefcn = 'exampleCustomized'` in `exampleCustomized`_cfg.m. `exampleCustomized_savefcn.m` may be used to derive own creations.

IX. Sound output and interfacing other hardware

A. Sound output

AFC supports the internal MATLAB sound commands `audioplayer`, `sound`, and `wavplay` as well as third-party (external) sound commands. The recommended external sound command on Windows systems is SoundMexPro (www.soundmexpro.de), which is included in the release version in a free and restricted (2-channel) AFC-specific version. SoundMexPro offers ASIO support avoiding drop outs with USB sound cards, up to 32-bit output, and continuous playback (with additive or multiplicative mixing in the presentation intervals). A number of other third-party sound commands are supported. The third-party sound command to be used is specified in the configuration variable `def.externSoundCommand` either for specific experiment in, e.g. `myExp_cfg.m` or globally in `autoexec_cfg.m`. Use of SoundmexPro can be activated by `afc('setup')` which modifies `autoexec_cfg.m`.

If `def.externSoundCommand` is set to an empty string (default), MATLABs internal sound command is selected by `def.internSoundCommand` (either `'audioplayer'`, `'sound'`or `'wavplay'`). As default AFC uses MATLAB's intern sound command `audioplayer`.

It is recommended to configure the sound command in `autoexec_cfg.m`. While `afc('setup')` can be used to select versions of soundmexPro, other sound commands have to be configured manually in `autoexec_cfg.m` If a specific sound command should be used for a specific experiment, the configuration in `autoexec_cfg.m` can be overloaded by definition of another sound command in the respective experiment's `_cfg` file. If you experience problems with sound output or synchronization of button highlighting, have a look at section XIII. In this case it might be helpful to additionally specify the delay for the button highlighting using `def.markIntervalDelay` or the sound device with `def.deviceID`.

Support for further third-party sound commands (e.g., playrec or msound) can be easily added by definition of a custom sound script, e.g. `sound_mySoundCommand.m`. An example is provided as `\addons\sound_exampleAudioplayer.m` which uses MATLAB's internal audioplayer command and which can be used as basis for own custom sound scripts. To use a custom sound script, `def.externSoundCommand` has to be set to the name of the sound script, e.g. `def.externSoundCommand = 'exampleAudioplayer'` for usage of `sound_exampleAudioplayer.m`. The example experiment `\experiments\exampleCustomized` is can be used as a reference for using a custom sound command. If a custom sound script with a name identical to one of AFC's stock methods is created, e.g. `sound_soundMexPro.m`, it overrides the sound command call from `afc_sound.m`. Thus custom sound scripts can be used to add support for new sound commands or to overload stock sound commands.

The bit depth to be used is specified by def.bits which defaults to 16. Note that the supported bit depths depend on the sound command.

B. Interfacing external hardware

AFC's default interface to external, third-party hardware is located in the function `extern_defaulthardware.m`, which can be overloaded and adapted to the meet own requirements by the customer. The default hardware function plays a short silent (zero) sound output during initialization. The typical use case is initialization and update of software-controlled amplifiers during an experiment. The default hardware function can be overloaded

and adapted to the meet own requirements by the customer. An example is provided by `\addons\extern_exampleTDT.m` which can be used as reference for own modifications. A custom hardware function is called by specifying, e.g. `def.extern_hardware = 'exampleTDT'` in the experiment configuration or in `autoexec_cfg.m`.

X. Calibration of sound output

For calibration of the sound output levels, it is recommended to determine the root-mean-square (rms) sound pressure level (SPL) produced by the equipment used (e.g., headphone on an artificial ear) for a digital output signal with an rms full-scale (FS) level of one. This calibration constant can be determined with pure tones at a certain digital level (e.g., -20 dB FS rms) for a number of frequencies. The resulting sound-pressure level in dB SPL rms is gained using standard calibration equipment. The values are then converted into dB-SPL-rms for a zero-dB-FS-rms output and stored in a calibration table for each frequency and audio channel in AFC's calibration script. Note to always compare either rms values for both types of signals (dB FS rms to dB SPL rms) or peak values, respectively. It is recommended to name the calibration script according to the measurement site for which the calibration was performed, e.g., `boothID_calibration.m`. The included `default_calibration.m` assumes an output of 100 dB SPL rms for 0 dB FS rms to avoid overly high output levels on un-calibrated systems. In addition to the calibration table, the calibration script can also specify a FIR compensation filter for the headphones or speakers, or can optionally calculate a compensation filter from the calibration table.

AFC provides a script to generate and play pure tones of arbitrary frequency including an optional FIR compensation filter to conduct the calibration. In addition, a script to validate calibration files is included.

All stimuli provided by the user script of the experiment must be scaled according to their desired sound-pressure level, e.g., a pure tone stimulus with a desired peak level on 60 dB SPL has to be scaled to a peak value of 10000.

How to use the inbuilt calibration functionality is demonstrated in `experiments\exampleCalscript_*.m`. The experiment uses the default calibration script `base\default calibration.m`. Typically, the user should define a specific calibration file for a certain setup, e.g., `calibration\boothID_calibration.m`, or for an experiment, e.g., `calibration\myexperiment_calibration.m`. The calibration file that is used for a specific experiment can be specified in different ways. def.calScript = 'boothID' in `myexperiment_cfg.m` would force AFC to use `boothID_calibration.m`. If def.calScript is set to 'autoSelect' either a calibration file with the name of the experiment (in this case `myexperiment_calibration.m`) is loaded or the last argument of the afc main function call is used to reference a calibration file, e.g., `afc_main('exampleCalscript','myname','boothID')`; This is useful if an experiment is conducted with different setups. If def.calScript is set to 'autoSelect' and no fitting calibration file is found, `base\default_calibration.m` is loaded. Therefore the user should customize the default file to give meaningful values for the specific setups. Please open the example calibration file `calibration\example_calibration.m` for details about the calibration and as a starting point for own calibration files. Note to always compare either rms (root-mean square) values for both types of signals (dB FS rms to dB SPL rms) or peak values, respectively. The calibration values are then entered in def.calTable in, e.g, `calibration\mysetup_calibration.m`.

Instead of using AFC's inbuilt tools for the calibration using `def.calScript`, the user can take care of the calibration in an alternative way. In this case it is recommended that the user specifies the calibration level in the variable `work.currentCalLevel` in, e.g., `experimentName_set.m`. The target level during the experiment is then reached by appropriate attenuation of the digital signal in MATLAB in the `_user` file. This method is used most of the included examples, e.g. `exampleInterleaved_set.m`, where the experimental condition 'cd1' or 'cd2' is actually used to switch between two setups.

If external hardware, e.g. Tucker-Davis equipment is configured using the extern-hardware scripting (default is `extern_defaulthardware.m`, calibration settings might also fit here.) Then, for different measurement units, the extern hardware script might be customized for the specific experimental setup the computer is running with. (Note, the `def.extern_hardware` configuration variable can be used to execute customized hardware scripts, as in, e.g. `\experiments\exampleCustomized_cfg.m`.)

XI. Hints for experiment debugging and development

During the development of an experiment it is often useful to display the output time signal or spectra. AFC is able to display the time signal and the spectra as well as the development of the tracking variable during a run. These features are enabled with the `'def.show*'` configuration variables in `experimentName_cfg.m`. It might be convenient to define a specific "debugging subject" in, e.g., `debugSubject_cfg.m`. See `docs\configurationVariableList.txt` for more details.

XII. Additional help

For additional help is provided by typing `help afc main` in the MATLAB command window and also in the comments in `example*_cfg.m`, `example*_set.m`, `example*_user.m`. Furthermore please have a look at `docs\configurationVariableList.txt` and `docs\dataImport.txt` or at `docs\FAQs.txt` for frequently asked questions.

XIII. Known issues

Since at least MATLAB's 2011 version the internal sound command `'sound'` is no longer asynchronous and can thus no longer be used for visual indication of the intervals by highlighting the buttons as in earlier versions (`def.markinterval = 1`).

The default internal sound command is now `'audioplayer'`. This brings other issues as `'audioplayer'` is not well suited either for visual button marking because it introduces system and MATLAB version dependent delays. Therefore it is recommended to use the tweak variable `def.markIntervalDelay = 0.5`, which is automatically initialized with this value when using `'audioplayer'`. The best value may depend on your system and might be `def.markIntervalDelay = 0`. Modify the delay in `'autoexec_cfg.m'` to match your needs.

For MATLAB versions until 2013 on Windows systems, `'wavplay'` is working well. If you experience weird delays for the button marking, set `def.markIntervalDelay = 0` in `'autoexec_cfg.m'`.

If you use `'sound'`, a warning is issued in newer versions in combination with button highlighting (`def.markinterval = 1`).

If you use the included free AFC version of soundmexPro buttons can be highlighted by stating `def.markinterval = 1` and `def.soundmexMark = 1`. In MATLAB 2014 there seems to be an issue that whole parts of the response window are highlighted and not the individual buttons.

If you use an older full version of soundmexPro you might run into issues with ASIO4All (www.asio4all.com) 2.11 drivers, so it is recommended to update soundmexPro or to use ASIO4All 2.10 or earlier.

XIV. FAQ - Frequently asked questions

Q: I have changed `expparX` values in my experiment `_cfg` file, but it doesn't seem to be used when I run the experiment.

A: Once you start your experiment, AFC generates a control file (which is in the folder you specified with `def.control_path` in `_cfg` or in the current folder (type `'pwd'` in MATLAB) and uses the `expparX` values in the `_cfg` file. It has the name `'control_myExperiment_XYZ.dat'`. If the control file already exists, it is not overwritten. You have to delete it in order to let AFC generate a new one with the current values you specified in `_cfg`.

Q: Where are my data files?

A: You probably did not specify a `def.results_path` in your experiment `_cfg`. In this case the results were written to your current working directory.

Q: I get the error: def.parrand dimension mismatch

A: def.parrand has to be a row vector with the number of elements equal to the number of exppars.

Q: I get the error: exppar dimension mismatch, incorrect number of tracks

A: One of your exppars has more than one but less columns than def.interleavenum. When interleaved, all exppars have to have def.interleavenum columns or only one column, in which case it is assumed that the specific exppar is the same for all interleaved tracks.

Q: How do I set up the exppar variables for interleaved?

A: When running an interleaved experiment, the exppar variables which should be interleaved in one run need to be in one column and def.interleavenum needs to equal the number of rows. The exppars in different columns are used in different runs. The exppars in different rows are used in the interleaved tracks of one run.

Q: I would like to have different startvars for the elements of expparN. Is it possible?

A: It is possible to use an additional exppar to define the startvars and to overwrite the values from `def.startvar` in the `_set` file of the experiment. The example experiment `exampleInterleaved` documents how this is done. Please have a look at the `exampleInterleaved_cfg.m` and `exampleInterleaved_set.m` file.

Q: The red highlighting of the buttons is not in sync with the sound. How can I tweak this?

A: Add `def.marktIntervalDelay = 0` in autoexec_cfg.m and tweak the value. If you are using MATLAB 2007b a value of 0 fits.

Q: Why is AFC using global variables? I don’t like this and it confuses me.

A: The use of the global structures, e.g. work and def for AFCs internal communication has mainly historical reasons. Back in 1999 it was the only way to let MATLAB do ‘pointer operations’ on the (back then) large sound data arrays to enable fast enough processing for running AFC.