PSYLAB – Documentation
Version 2.9

Martin Hansen
Institut für Hörtechnik + Audiologie
Jade Hochschule
Oldenburg, Germany

March 12, 2019

Contents

1 Introduction

1.1 What is PsyLab?

'PsyLab' is a collection of scripts, written in Matlab, for designing and controlling interactive psychoacoustical listening experiments in a uniform and quick manner.

Currently, n-AFC detection and discrimination experiments ($n = 2, 3, 4$) and 2-AFC matching experiments are readily supported. The stimulus variable can either be controlled adaptively according to x-up-y-down algorithms ($x, y \in \{1, 2, 3\}$)) and weighted-up-down algorithms. Alternatively, the method of constant stimuli may be used for detection and discrimination experiments.

1.2 History of PsyLab

PsyLab is based on many ideas of the programs 'SI/SISG' which were written at "Drittes Physikalisches Institut", Universität Göttingen, during the years 1989 to 1991, and which were distributed under the GNU General Public License (GNU GPL). SI ("Signalverarbeitung interaktiv") was programmed in Fortran by Dirk Püschel and René Koch and further colleagues for the purpose of digital signal processing and analysis. SISG, built on SI, was written by Ralf Fassel and was used for development, control and analysis of psychoacoustical experiments. The philosophy and organisation of PsyLab was inspired by the paradigms of "SISG" in many ways.

PsyLab has been written by Martin Hansen. It is regularly used for education and research at Institut für Hörtechnik und Audiologie (IHA), Jade Hochschule in Oldenburg, Germany.

One aim of PsyLab has been to provide the students in the psychoacoustic courses with a uniform and simple starting point for designing one's own listening experiments. So far, roughly 400 students at IHA have used PsyLab in their psychoacoustics assignments, for designing and carrying out individual listening experiments. Their experience and comments have continuously contributed to new features and bug fixes of PsyLab.

1.3 License of PsyLab

PsyLab is "free software" and is distributed in OpenSource format under the terms of the GNU General Public License (GPL). This means, amongst others, that copies of the PsyLab-software may be distributed without asking for permission, given that the terms of the GNU GPL are obeyed to.

For more details, see `http://www.gnu.org/licenses/` or look at a verbose copy of the GNU GPL on page 34.

1.4 No warranty, no liability

PsyLab is distributed without any kind of warranty. Have a look at the file "`NO-WARRANTY`" in the root directory of PsyLab before using it!

The use of PsyLab, for whatever purpose, is under the complete and sole responsibility of the user. **Special attention is drawn to the fact that PsyLab does *not* comprise an automatic mechanism to prevent the delivery of too loud sound pressure levels.** It is the explicit and sole responsibility of the user of PsyLab to make sure that the generated stimuli, in combination with any further equipment (sound card, amplifier, headphone or loudspeaker, etc.), will yield the desired sound level.

1.5 Quick start: example experiments

A hands-on explanation of the way how PsyLab works can be found in the subdirectory '`examples`' of the PsyLab-distribution. It contains several example experiments, ready for giving them a try.

Each experiment is comprised of three scripts with similar names, i.e., with the same first part of the name. The name of the "main script" of an experiment can be chosen freely, e.g. '`myscript.m`'. The two accompanying scripts *must* then carry the names '`myscriptset.m`' and '`myscriptuser.m`'. The main script describes and controls the measurements while the other two scripts are used for stimulus generation during the experiment. For a quick start, run one of the example experiments by running the corresponding main script.

The example in the file 'sam_sincarrier_detect.m' is an experiment which measures the detection threshold of the modulation degree of sinusoidal amplitude modulation ("SAM") as a function of modulation frequency. The measurements are performed for a number of different carrier frequencies of the sinusoidal carrier.

The example in the file jnd_frequency.m is an experiment which measures frequency discrimination of pure sinusoids as a function of the reference tone frequency and its sound level.

The example in the file match_freq_binaural.m is a matching experiment which measures the frequency for binaural pitch matching between left and right ear for pure tones as a function of the reference frequency.

The example in the file tone_in_broadbandnoise.m is an experiment with interleaved tracks for measuring the detection threshold of the level of a sinusoid in broad band noise as a function of the tone frequency.

2 Philosophy and structure of PsyLab

2.1 Coding style in PsyLab

The coding style in PsyLab may lead to some debate for some people. However, keep in mind that PsyLab was not designed to serve as an example for one or another coding style.

For several reasons, amongst others easy accessibility for students with limited prior experience in programming/coding, most of the PsyLab source files are written as *scripts/m-files*, i.e. not as functions. This means that all variables controlling PsyLab's behaviour reside in the global workspace of Matlab. This makes it especially important for the user to be informed about the specific naming convention for files and variables. This is explained in section 2.2.

2.2 Conventions in PsyLab: reserved names for files and variables

An important convention respectively requirement of PsyLab is that every experiment must be made up of three (Matlab) scripts. These are the "main script", the "set script", and the "user script". These three files must be pure scripts, i.e. they may *not* be Matlab *functions*! As a consequence, the scripts can access all variables of the global Matlab-workspace. The scripts can of course call further scripts or functions. By defining the file name of the main script, the names of the set script and the user script are determined, too: Their names result from concatenating 'set' respectively 'user' directly after the name of the main script, e.g. testnameset.m respectively testnameuser.m if the name of the main script is testname.m. Note that no single character is allowed between the 'set' or the 'user' and the name of the main script.

Another convention applies to the use of certain variables with reserved names, e.g. M.VAR, M.PARAM, m_ref, m_test. See sections 3.1, 3.2, 3.3, 5.

2.3 Terminology and definitions

This section explains some terms and definitions used inside PsyLab. In addition, the principal course of psychoacoustical experiments realised with PsyLab is explained. Some of the terms are illustrated in Fig. 1.

interval, signal, stimulus During an experiment, the human test subject is presented with one or more stimuli. The individual stimuli are also called intervals or signals. In an n-AFC-experiment, n intervals are presented in succession. Most often, these signal intervals will be separated by short pauses (of silence). It is possible to visually mark the individual intervals.

trial A trial is one presentation of the n stimulus intervals that pertain to an n-AFC experiment. After each trial the subject is required to answer respectively respond to a question, for example one of the kind "which interval contained the test signal?".

(stimulus) variable The "variable" inside PsyLab takes the role of that stimulus dimension/magnitude which is varied from trial to trial (for example adaptively, or using a method of constant stimuli) in order to find the subject's threshold. The "variable" carries the reserved name M.VAR inside PsyLab.

(stimulus) step size The step size is the magnitude by which the stimulus variable gets changed from trial to trial during an adaptive experiment. It is measured in the same units of the variable `M.VAR`. The step size is typically larger in the beginning of the experiment and is then progressively reduced until some minimal step size has been reached. The step size carries the reserved name `M.STEP` inside PSYLAB, and the minimal step size carries the name `M.MINSTEP`.

The step size doesn't have a meaning and isn't used in the "method of constant stimuli".

run A run is the succession of all trials that are necessary to reach and determine one threshold of the varying stimulus variable.

(stimulus) parameter The parameter inside PSYLAB has the meaning of the arbitrary (but then fixed) stimulus dimension/magnitude which is usually kept constant during one run until threshold has been established. In the next run the stimulus parameter would then usually be set to another value and the corresponding threshold would again be established, and so on. The "parameter" carries the reserved name `M.PARAM`.

2nd, 3rd, ... (stimulus) parameter If an experiment requires more than only one stimulus parameter, PSYLAB offers the possibility to use (and organise) any number of independent parameters. In that case, `M.PARAM` will just be a vector instead of a scalar, and `M.PARAM(1)` will be the 1st parameter value, `M.PARAM(2)` the 2nd, `M.PARAM(3)` the 3rd, etc.

reference signal In a trial of a n-AFC experiment, n intervals are presented in succession, and in random order. There are $n-1$ intervals among them which contain the reference signal, i.e. they do *not* contain "the signal" or the "cue" that the subject is required to detect. The reference signal must carry the reserved name `m_ref` *or* `m_ref1`, `m_ref2` etc. inside PSYLAB.

test signal The test signal is the signal which *does* contains that special "cue" that the subject is required to detect, contrary to the reference signal which does not. The test signal will be presented at random in one of the n intervals of a n-AFC trial. The test signal must carry the reserved name `m_test` inside PSYLAB.

familiarisation phase At the beginning of a new run with an adaptive control of the stimulus variable, the test signal will typically be presented clearly above threshold in order to orient the subject about the kind of the signal respectively the "cue" that is to be detected, and to familiarise the subject with the measurement procedure. The step size will be larger at the beginning of this familiarisation phase so that correct responses of the subject will carry the stimulus variable towards the region of threshold more or less rapidly. At the same time the step size will be reduced progressively until it has reached the minimal step size which the user has to specify. The trials up to this point are only used for familiarisation of the subject, and to approach a value of the stimulus variable in the vicinity of the threshold. The trials of the familiarisation phase will however *not* be included in the determination of the threshold.

When using the "method of constant stimuli", there is no familiarisation phase.

measurement phase In an adaptive run, the measurement phase starts at the end of the familiarisation phase. The subject should have approached a point relatively close to the threshold and the stimulus variable changes now only with the minimal step size. The measurement phase lasts until a certain number of reversals (see below) has been reached. This maximum number of reversals serves as a stop criterion and needs to be specified by the user. The threshold is then estimated based on all values that the variable had taken on during the measurement phase.

reversal / turning point In a transformed-up-down experiment [Levitt, 1971], the stimulus variable and its direction of change depend on the subject's responses. The direction alternates back and forth between "going down" and "going up".
NB: Until version 2.5, PSYLAB counted only an upper point, i.e. the change from an up-phase to the next down-phase as 1 "reversal".
NB: Since version 2.6, PSYLAB counts both an upper (from-up-to-down) and a lower (from-down-to-up) point as 1 "reversal".

In other words, as of version 2.6, the occurrence of a local maximum, and likewise of a local minimum, of the stimulus variable `M.VAR` as a function of the trial number is defined as a reversal. This definition inside PSyLAB is in line with the definition of "reversals" respective "turning points" in the majority of the literature.

The number of reversals serves as a stop criterion for a adaptive run. When the number of reversals within the measurement phase has reached the maximum number of reversals, as specified by `M.MAXREVERSAL`, that run is terminated.

If you re-use an older experiment, implemented with PSyLAB version up to 2.5 using the old definition of reversals, there will be a (limited) automatic check for meaningful values of the variable `M.MAXREVERSAL`.
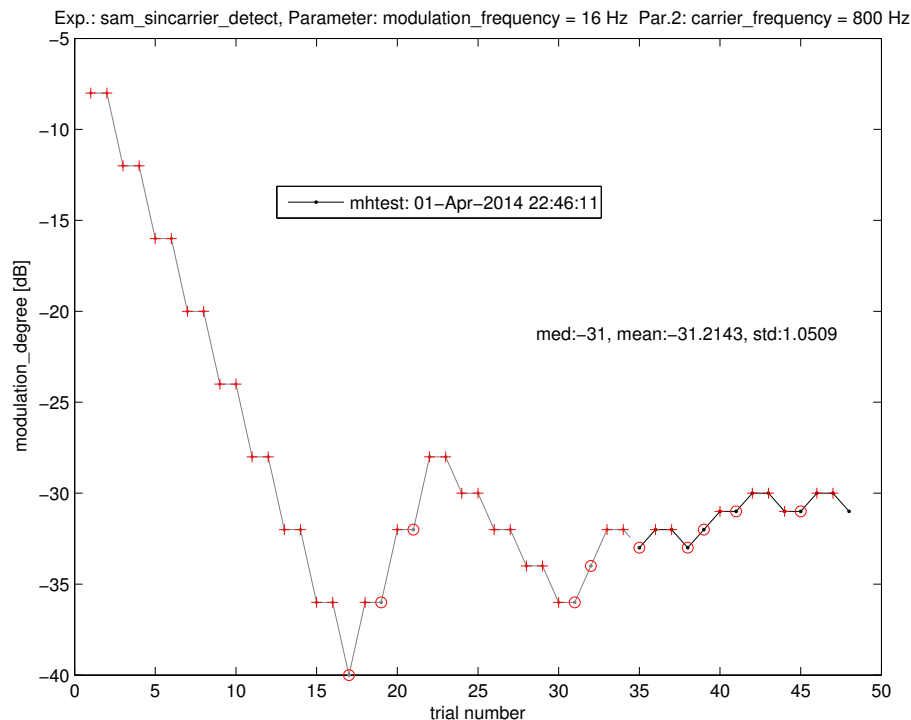

Figure 1: Example of the course of one experimental run with PSyLAB. The plus-signs (+) mark a correct response of the subject, the circles (∘) mark a wrong one. The familiarisation phase (depicted in gray) lasted from trial 1 up to trial 34. During that phase reversals occurred at trial 17, 23, 31, and 34. At each upper reversal, the step size was halved until the minimum step size of 1 dB had been reached. At that point, the measurement phase started (at trial 35, depicted in black) and it lasted for 6 reversals (at trials 35, 37, 38, 43, 35, and 47) which had been set as the maximum number of reversals. The last reversal of those 6 was reached in trial 47 and the run terminated there. The very last data point, trial 48, *would* have been presented next with a modulation degree as indicated, hadn't the run stopped.

Figure 1 shows an example run of the experiment `sam_sincarrier_detect.m` which can be found in the subdirectory `examples` of the PSyLAB-distribution. The data show the value of the stimulus variable as a function of the trial number. The plot can be generated automatically with the PSyLAB-script `mpsy_display_psydat.m`, see section 4.2. The run started with the familiarisation phase with the settings `M.VAR = -8; M.STEP = 4;`. Both values are stated in dB re. 1. The first reversal was reached at trial 17, the step size remained at 4 dB. At trial 23 the second reversal was reached. As this was an upper reversal, the step size was halved to 2 dB. At trial 31 the third reversal was encountered and the step size remained at 2 dB. At trial 34 the fourth reversal was reached and the step size was halved to 1 dB. As this step size had been defined as the minimal step size, the familiarisation phase ended here and the measurement phase started at the next trial 35. From there, the number of reversals starts again counting from scratch. The run terminated when a number of `M.MAXREVERSAL` reversals had been

reached. In the example with `M.MAXREVERSAL=6;` this point was reached after trial 47. Note that the plot shows a data point at trial 48 which was *not* presented to the subject. The corresponding stimulus *would* have been presented to the subject, hadn't the run just been terminated. The threshold is calculated as the *median* of all values of `M.VAR` during the measurement phase, a value of -31 dB in the example. The standard deviation of the threshold is calculated from the same data points and amounted to 1.05 dB in the example. The mean of all data points in the measurement phase is also stated, but only in the plot.

2.4   Types of experiments in PSYLAB

PSYLAB offers to perform various kinds of experiments. One class of experiments are detection and discrimination experiments using a N-AFC transformed-up-down method. These experiments can be performed as a sequence of single runs, one run at a time, or as a group of interleaved runs. These are described in sections 2.4.1 resp. 2.4.4.

Slightly different are matching experiments, like for example employed in pitch matching or loudness matching. The difference relative to N-AFC detection experiments is that the answer of the subject can not be judged to be correct or wrong. Matching experiments are described in section 2.4.2. Matching experiments may also be carried out in an interleaved-runs fashion.

2.4.1 N-AFC experiment

The script `mpsy_afc_main.m` is a central part of PSYLAB for controlling adaptive transformed-up-down n-AFC experiments. Use it if you intend to measure one experimental run at a time, i.e. the next run starts only after the previous run has been completed, i.e. threshold has been reached.

It should be unnecessary for the user to change this file. However, in order to work properly, your own experiment scripts need to conform to a number of conventions, see below.

The file `sam_sincarrier_detect.m` in the subdirectory `examples` shows an example for such a 1-up-2-down 3-AFC experiment which measures the modulation detection threshold for sinusoidal amplitude modulation of sinusoidal carriers. Details are explained in section 3.

2.4.2   Matching experiments

The script `mpsy_match_main.m` is a central part of PSYLAB for controlling adaptive transformed-up-down matching experiments where two signal intervals are presented per trial and the subject is asked to adjust some stimulus dimension of one of them, with the aim to "match" the stimuli regarding a given perceptual aspect.

The file `match_freq_binaural.m` in the subdirectory `examples` shows an example for such an experiment where a binaural pitch match is performed for sinusoids presented monaurally left respectively right.

2.4.3   Available adaptive methods

PSYLAB offers a number of adaptive methods to choose from. The purpose of these methods is to adaptively control the value of the stimulus variable, depending on the subject's previous answer(s).

The most well-known of these methods are probably the "*transformed up-down methods*" [Levitt, 1971], and the 1up-1down, 1up-2down, 2up-1down, and 1up-3down methods are readily available in PSYLAB. The method is specified by the variable `M.ADAPT_METHOD` which should hold the value '`1up-2down`' in order to use the 1up-2-down method.

Two other possible methods/values are '`wud`' for the "*weighted-up-down*" method [Kaernbach, 1991], and '`uwud`' for the "unforced weighted up-down" method [Kaernbach, 2001]. These two methods offer the possibility to (more or less) freely select the percentage correct towards which the adaptive method shall converge. This percentage, e.g. 0.75, must be specified in the variable `M.PC_CONVERGE`. For more information see the description of the variables `M.ADAPT_METHOD` and `M.PC_CONVERGE` in table 1 on page 29.

When using any of the adaptive methods mentioned above, the common behaviour is that wrong answers of the subject lead to an *up*-ward change of the stimulus variable (thereby making the task easier), while correct answers lead to a *down*-ward change (making the task more difficult). This behaviour can be changed using the variable `M.REVERSED_UP_AND_DOWN`. Section 3.4 explains this in more detail.

2.4.4　Experiments with interleaved tracks

PsyLab offers the feature to run an N-AFC experiment or a matching experiment with "interleaved tracks", which means that several experimental runs (usually with different sets of parameter values) are running at the same time in an interleaved fashion. Each run is controlled as if it was a single run (see section 2.4.1) but the current trial is chosen randomly from all incomplete runs, i.e. those runs which have not yet reached threshold.

The script `mpsy_intrlv_afc_main.m` is a central part of PsyLab for controlling such interleaved-tracks adaptive transformed-up-down n-AFC experiments. The script `mpsy_intrlv_match_main.m` does the corresponding job for interleaved matching experiments. It should be unnecessary for the user to change these files. The same requirements and conventions apply as for single-run experiments (see section2.4.1).

The file `tone_in_broadbandnoise.m` in the subdirectory `examples` shows an example for a 1-up-2-down 3-AFC experiment with interleaved tracks, which measures the detectability of a sinusoidal tone in broad band noise. In this example, the threshold for tones of different frequencies is measured in an interleaved-tracks fashion.

The file `match_freq_binaural_intrlv.m` in the same subdirectory shoes an example for an interleaved matching experiment, where two tracks with two different adaptive rules (1up-2down and 2up-1down) are interleaved.

Note that interleaved-tracks experiments may require some extra amount of testing and debugging, especially for the inexperienced programmer. For details see subsection 3.8. Use the force – read the source!

Experiments with interleaved tracks are not yet available for the method of constant stimuli.

2.4.5　Method of constant stimuli

As an alternative to the adaptive control of the stimulus variable, as explained in section 2.4.3, PsyLab also offers the method of constant stimuli. With this method, the stimuli (respectively the values of the stimulus variable) that are going to be presented is known/fixed prior to the test subject's arrival at the test booth, hence the name "constant".

A number of different values of the stimulus variable is chosen and must be saved in the PsyLab variable `M.CONSTSTIM_ALLVARS`. The number of (repeated) presentations must be specified in the variable `M.CONSTSTIM_NUM_PRESENTATIONS`.

The total number of (`M.CONSTSTIM_ALLVARS` $\cdot$ `M.CONSTSTIM_NUM_PRESENTATIONS`) corresponding stimuli are then presented in a randomized/permuted order, with a new permutation for each new experimental run and for each subject.

The file `jnd_frequency.m` in the subdirectory `examples` shows an example for such a constant stimuli experiment where the just noticeable difference in frequency for two sinusoidal stimuli is measured.

Note that an experiment with an adaptive method (like for example 1-up-2-down) has an in-built familiarisation phase that takes place prior to the collection of those data points from which the threshold will be derived. Such a familiarisation phase is not normally part of a constant stimuli experiment. However, it is easy for you to implement some training phase, for which the results shall be discarded: You might always execute the same constant stimuli experiment twice, first with a temporary different subject name and with a smaller number of stimulus repetitions (`M.CONSTSTIM_NUM_PRESENTATIONS`), and the in the second run with the original subject name and number of repetitions.

3　Designing your own experiment

This section starts with the description how to use PsyLab for non-interleaved experiments. Subsection 3.8 explains specific differences that apply for interleaved-tracks experiments.

Designing your own psychoacoustical experiment with PsyLab is easy: Just write your main script, set script, and user script – and start to listen. This section shows you how to write these scripts. You might also want to take a look at the example experiments in the subdirectory `examples` to get a demonstration, or to use them as a starting point.

3.1   The main-script

The purpose of the main script is to assign proper values to a number of principal PSYLAB-variables which define your experiment. These variables are all struct fields of the variable M in the global Matlab-workspace. The name 'M' is *fixed* within PSYLAB, it is a *reserved name* that cannot be used for other purposes. The name 'M' is due to historical reasons; think of it as originating from 'M'easurement.

Section 5 explains how the different struct fields of M control the behaviour of PSYLAB.

Your main script should start with the following line

```
mpsy_init;
```

This call of `mpsy_init.m` *prior* to assigning values to further PSYLAB-variables is mandatory, as the code in `mpsy_init.m` initialises some PSYLAB-variables to their proper start values.

`mpsy_init.m` will, amongst other things, also call a splash screen and query the users consent to being responsible for the control of the sound output level. If the users wishes so, she can disable the splash screen and the query by editing the file `mpsy_querysplash.m`. Have a look at the 1-line-code in that file.

One important variable resp. struct field of M is `M.EXPNAME`, which carries the name of the experiment, as a string. This name *must* be identical to the file name of the main script. Do not try to alter this behaviour! `M.EXPNAME` *must* be defined in the main script. Therefore, it is good practice to use the following line of code (see the help for `mfilename`) in the main script:

```
M.EXPNAME = mfilename;  % important to be set correctly!
```

This implies of course that your main script should bear a good telling name.

A number of further variables resp. struct fields of M need to be set in the main script, which can best be explained by an example. The specific values of the following example are taken from the file `examples/sam_sincarrier_detect.m` of the PSYLAB-distribution. It implements a modulation detection experiment as described by [Kohlrausch et al., 2000]

```
M.NUM_PARAMS    = 2;   % number of parameters
M.PARAMNAME(1)  = {'modulation_frequency'};
M.PARAMUNIT(1)  = {'Hz'};
M.PARAMNAME(2)  = {'carrier_frequency'};
M.PARAMUNIT(2)  = {'Hz'};
M.VARNAME       = 'modulation_degree';
M.VARUNIT       = 'dB';
M.MINSTEP       = 1 ;  % minumum step size, in units of M.VARUNIT
M.NAFC          = 3 ;  % number of forced choice intervals
M.ADAPT_METHOD  = '1up_2down';   % select adaptive method: transformed 1-up-2-down

M.MAXREVERSAL   = 8 ;  % number of reversals required as stop criterion
M.FEEDBACK      = 1 ;  % provide feedback (correct / not correct) to subject
M.INFO          = 1 ;  % provide intermediate info for the subject
M.TASK          = 'which interval contained the test signal (1,2,3)?  ';

M.FS             = 48000; % sampling frequency
M.USE_GUI        = 1;     % use a GUI for user input
M.VISUAL_INDICATOR = 1;   % use visual interval indication
M.SAVERUN        = 1;     % save individual subject responses in psylab file
```

You also need to assign the variable `M.SNAME` with the name (or better: initials) of the test subject. This could be achieved by

```
M.SNAME = input('please type your initials (no spaces, please)','s');
```

It is required by PSYLAB that `M.SNAME` must *not* contain any whitespace characters! You should urge your subjects to always use the same, identical subject name for all measurements. Otherwise the automatic data analysis might become much more complicated (for you as the investigator) than necessary.

Next, your main script would assign proper value(s) for your stimulus parameter(s) stored in the PSYLAB-variable M.PARAM, and then call the PSYLAB-script mpsy_afc_main once for each set of parameter value(s). The script mpsy_afc_main.m is one of the central parts of PSYLAB. It performs the automated control of one complete experiment run until threshold is established.

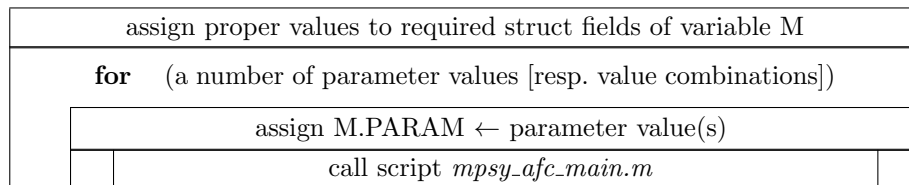To summarize, the principle design of your main script is depicted in Fig. 2.

main script

assign proper values to required struct fields of variable M		
for (a number of parameter values [resp. value combinations])		
	assign M.PARAM ← parameter value(s)	
	call script *mpsy_afc_main.m*	

Figure 2: Principle structure of the main script of a PSYLAB experiment.

Assigning parameter values to M.PARAM (as a scalar in case of one parameter, or as a vector in case of ≥ 2 parameters) and the subsequent call of mpsy_afc_main inside the main script could be achieved in a loop as follows. (The example is again taken from the experiment sam_sincarrier_detect.m).

```
carrier_freqs = [ 800 1600 3200 ];
mod_freqs     = [ 4 16 64 256 ];

for kk = 1:length(carrier_freqs),
    M.PARAM(2) = carrier_freqs(kk);
    for ll = 1:length(mod_freqs),
        M.PARAM(1) = mod_freqs(ll);
        mpsy_afc_main;
    end
end
```

In this example, the carrier frequency is the second parameter. It will be set to 800, 1600 and 3200 Hz, respectively. The modulation frequency is the first parameter which will be set to 4, 16, 64, and 256 Hz. This way, the experiment consists of 12 runs, one for each of the 12 combinations of values of the two parameters. Of course, one could imagine that you would like to measure only 2 or 3 trials today, and continue measuring the remaining parameters the other day. This could be achieved as in the following example:

```
M.PARAM(2)  = 1600;
M.PARAM(1)  = 4;
mpsy_afc_main;
M.PARAM(1)  = 64;
mpsy_afc_main;
```

As mentioned earlier, the PSYLAB-script mpsy_afc_main.m controls one complete run until threshold is reached. The principle flow diagram of the mpsy_afc_main.m is depicted in Fig. 3. As can be seen, mpsy_afc_main will call the set script of the experiment *once* in the beginning. Then, in a loop running until threshold has been reached, the user script of the experiment will be called once for each trial. The natural purpose of the user script is to generate (new) realisations of the test signal m_test and the reference signal(s) m_ref (or m_ref1, m_ref2, etc.) which will then be presented to the subject in a n-AFC manner. Subsequently, the subject's response will be evaluated and the stimulus variable will be changed according to the adaptive rule.

3.2   The set-script

As already noted in section 3.1, the set script will be called *once at the beginning* of any new run. This will happen before the first trial with stimulus presentation, but after a new parameter value, respectively a set of values, has been assigned to the PSYLAB-variable M.PARAM in the main script. The call of the set
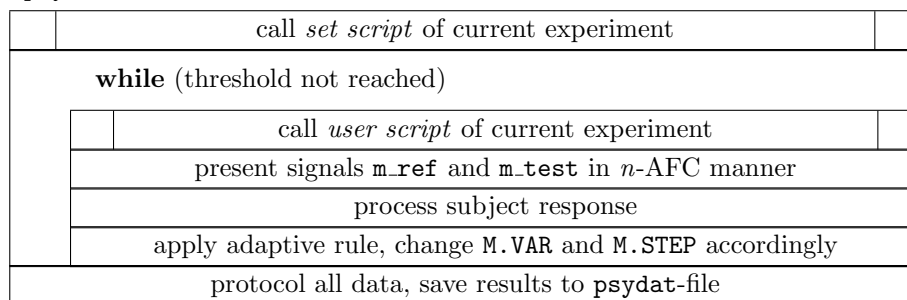
`mpsy_afc_main.m`

	call *set script* of current experiment	
while (threshold not reached)		
	call *user script* of current experiment	
	present signals m_ref and m_test in n-AFC manner	
	process subject response	
	apply adaptive rule, change M.VAR and M.STEP accordingly	
	protocol all data, save results to psydat-file	

Figure 3: Principle flow diagram of one run, as controlled by the script `mpsy_afc_main.m`.

script is performed automatically by the PSYLAB-script `mpsy_afc_main.m`, or `mpsy_match_main.m`. The user does not need to bother with this.

The purpose of the set script is to set some PSYLAB-variables to a proper start value. At least the variables M.VAR and M.STEP should be assigned with new start values. Also, any of those calculations that are of high computational load but only need to be performed once, should as well be placed into the set script. Finally, the set script is the appropriate place to pre-generate stimulus signals or parts of signals that will not change during the run. These include, for example, the special signal vectors with reserved names m_quiet, m_presig, and m_postsig which will be placed between, respectively prior, or after the stimulus intervals (see section 3.3).

For an example, see the set script `sam_sincarrier_detectset.m` in the subdirectory `examples`.

3.3   The user-script

The user script will be called once prior to each new trial. This is performed automatically by the PSYLAB-script `mpsy_afc_main`, or `mpsy_match_main.m`.

The purpose of the user script is to generate the (new) current stimuli that are to be presented in the current trial. Typically, these stimuli depend on the current value of the stimulus variable M.VAR, and most probably also on the value, respectively set of values of M.PARAM. The stimuli generated by the user script will then be taken for further use of the calling script, that means `mpsy_afc_main`, or `mpsy_match_main.m`. Therefore, the stimuli are required to have certain *fixed, reserved names*.

The following signal vectors must reside as Matlab variables inside the global Matlab-workspace at the end of the user script:

m_test   The signal vector that contains the stimulus of the test interval. The test interval contains "the signal" to be detected by the subject, respectively the signal deviation to be discriminated, which the reference signal does not.

m_ref   The signal vector that contains the stimulus of the reference interval(s). If the variable m_ref exists in the Matlab workspace, *this* signal is presented in the $(n-1)$ reference intervals of an n-AFC experiment, which means that all reference intervals are identical. This is for example illustrated in the example experiments `jnd_frequency.m` and `sam_sincarrier_detect.m`.

If m_ref exists in the workspace, then m_ref1, m_ref2, ... (see item described next) *must not* exist in the Matlab workspace.

m_ref1, m_ref2, ...   The $(n-1)$ signals vectors that contain the reference intervals in case that all reference intervals should be different from each other. This is, for example, required if independent realisations of a running noise shall be used for each reference interval. In this case, the variable m_ref must not exist in the workspace. The $(n-1)$ reference signal vectors m_ref1, m_ref2, ... will be presented in random order within each trial (on top of the random placement of the test signal within the n intervals). This is for example illustrated in the example experiment `tone_in_broadbandnoise.m`.

If m_ref1, m_ref2, ... exist in the workspace, then m_ref (see item described before) *must not* exist in the Matlab workspace.

Additionally, the following signal vectors with reserved names can (but need not) reside in the Matlab-workspace after leaving the user script respectively the set script:

m_quiet The signal which is presented between two adjacent stimulus intervals. Most often this will be a quiet (sic!) pause of silence, i.e. a vector of zeroes. Alternatively the signal could be an empty vector (`m_quiet=[]`) which would mean a direct succession of the n intervals of a n-AFC trial.

m_presig The signal which is presented before the first stimulus interval. This could also be silence, or for example a certain marker stimulus.

m_postsig The signal which is presented after the last of the n stimulus intervals.

m_background A signal which is presented simultaneously during the whole duration of a trial, i.e. from the beginning of the pre-signal until the end of the post-signal. This could, for example, hold a continuous noise background signal.

If one or more of the three signals `m_quiet`, `m_presig` or `m_postsig` are left unspecified, i.e. are nonexistent as a variable in the workspace, they default to the empty vector. This is automatically taken care of by the PSYLAB-script `mpsy_check.m`.

If the signal `m_background` is unspecified, i.e. nonexistent as a variable in workspace, it is neglected and no background signal will be added.

If any of the variables `m_quiet`, `m_presig`, and `m_postsig` are assigned a scalar value in the main script or in the set script, then this will be replaced by a silence signal with a duration as specified by that value (in seconds). Example: `m_quiet = 0.3;` will automatically be overwritten by the vector `m_quiet = zeros(round(0.3*M.FS), numchannels);` where the number of channels (1 for mono, 2 for stereo, ...) will be deduced automatically.

Note that this convenience will only work if the scalar duration is specified in the main script or in the set script, but not in the user script.

3.4 Need to go *"up"* after *correct* responses?

In normal adaptive methods, like 1-up-2-down etc., the stimulus variable goes down (is reduced) after one ore more correct answers, thereby making the task more difficult for the test subject. Likewise, the stimulus variable goes up (is increased) after one or more wrong answers, thereby making the task easier for the test subject. A classical example for this is the detection of a test tone in a noise masker, where the masker level is kept fixed and the tone level is the stimulus variable. When the tone level goes up after wrong answers it should get easier to detect the tone.

Sometimes, however, the opposite up/down-behaviour is necessary: The stimulus variable should go up after correct responses and it should go down after correct responses. An example of this is the same test of tone detection in a noise masker like above, but with a fixed tone level and the stimulus variable being the masker level. After correct answers, the masker level should go further up to make the task more difficult.

This reversed-up/down behaviour can be used in PSYLAB by setting `M.REVERSED_UP_AND_DOWN` as a special flag-variable to a non-zero value (meaning "true"). Additionally to this, your step size `M.STEP` and your minimal step size `M.MINSTEP` both have to take *negative* values, like in the following code example:

```
M.REVERSED_UP_AND_DOWN  = 1;   % enable reversed up/down behaviour
M.MINSTEP               = -1;  % dB final step size
M.STEP                  = -8;  % dB initial step size
```

Using this code, the *size* of initial steps will be 8 dB (but in opposite direction compared to the normal case), and the step size will be halved successively after each second reversal until it reaches the minimal *size* of 1 dB.

3.5 Required type and value of PSYLAB-variables

For the correct functioning of PSYLAB there are certain restrictions imposed on the possible value or type of some PSYLAB-variables. These are:

1. The following PSYLAB-variables must be *numeric* in type:

   `M.VAR`, `M.STEP`, `M.NUM_PARAMS`, `M.PARAMS`

2. The following PSYLAB-variables must be of type *string* and must *not contain whitespace* characters:

   `M.SNAME`, `M.VARNAME`, `M.VARUNIT`

3. The following PSYLAB-variables must be of type "*cell string*" (instead of a string):

   `M.PARAMNAME`, `M.PARAMUNIT`

   This is mandatory *even in case they only contain one single entry.*
   Example: `M.PARAMNAME = {'frequency'}; M.PARAMUNIT = {'Hz'};`

   Additionally, the different elements of these variables must *not contain whitespace* characters either.

Not sticking to these rules would probably let you run your experiment without problems (at first) but would put you into trouble when analysing your data later on, see section 4.1 and 4.2. Therefore, an automatic check of these variables is performed by the PSYLAB-script `mpsy_check.m`.

3.6   Test subject interface

PSYLAB offers a choice of different user interfaces for the test subject. One simple interface is to answer via a keyboard. Another is to provide a graphical user interface to be controlled via a mouse and/or key presses. The former has the advantage that an extra monitor in the listening booth could be omitted. The latter may be more informative for the subject and can provide some extra functionality, of course.

In an n-AFC test, a subject answer of 1, 2, ..., n indicates the subject's decision for interval 1, 2, ..., n. These answers can be indicated by keyboard or by corresponding mouse clicks.
Additionally, the following subject answers are possible which carry special meanings:

- The answer 0 indicates a deliberate wrong answer. It is not possible to give this answer via mouse click.

- The answer 8 means: terminate the current run.

- The answer 9 means: terminate the whole experiment.

The last two answers are also available from the "Quit"-menu of PSYLAB's answer GUI.

The type of interface, including the amount of information and feedback that is provided to the subject is controlled by these variables:

M.FEEDBACK   A value of 0 means: do not give feedback to the subject.
A value of 1 means: Give feedback to the subject, i.e. inform the subject whether or not the last answer was correct.
A value larger than 1 means: additionally save a plot of the current run's result after threshold has been reached. The resulting eps-file will always have the same filename, generated as `['plot_mvars_' M.SNAME]`. It will thus we overwritten after each completed run.
A value larger than 2 means: same as before, but create a new filename including the date/time information for each eps-file.

M.INFO   A value of 1 means: Provide additional information to the subject, for example, about parameters of next run, etc.

M.USE_GUI   A value of 0 means: use a pure keyboard interface. The subject's answer consists of a keypress, *followed by* `RET`.
A value of 1 means: use a graphical user interface. This can be controlled via button clicks by a mouse. Alternatively, answering by a single keypress, *without* hitting `RET`, is possible, too. However, in order for this to work properly, the background of the answer GUI must have the focus (rather than, for example, any of the answer buttons). Of course, the keypress needs to be a reasonable subject answer (numbers 1, through n, and 0, 8, 9, see above). Fig. 4 shows the PSYLAB-GUI.

M.VISUAL_INDICATOR This variable only takes effect in case of `M.USE_GUI==1`.

A value of 1 means: mark each stimulus interval visually by highlighting its corresponding answer button. This feature will only work correctly when all n stimulus intervals are of the same temporal duration. However, you should be able to achieve this for most experiments.
A value of 0 means: no visual interval indication.

On some operating systems and/or matlab versions, the visual interval indication is only supported when the built-in function "`sound`" works asynchronously. When this is not the case, then the visual interval indication will only work when "`msound`" is used for audio output, instead of the built-in "`sound`", see section 3.9.

Whether the built-in `sound` works synchronously or asynchronously is automatically checked by `mpsy_init_run.m`.
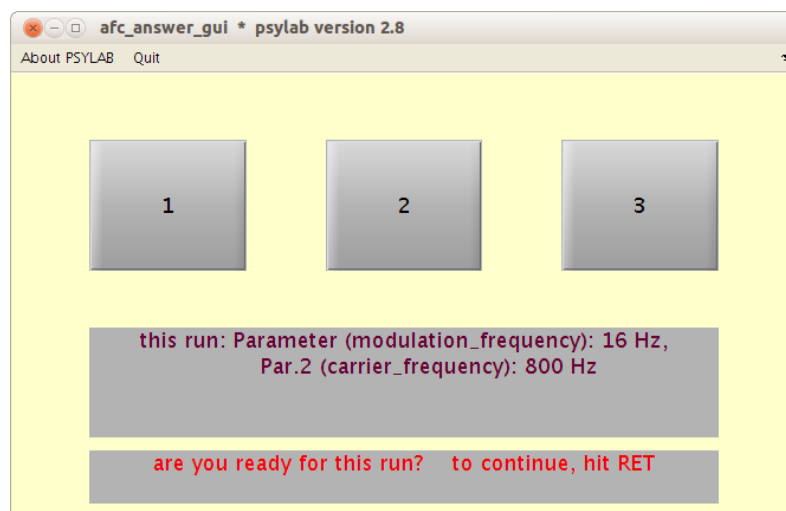
Figure 4: Example of the graphical user interface for the subject.

3.7 Running your experiment

Once you have set up your three experiment files correctly, running your experiment means running your main script. You might want to instruct your subjects to always identify themselves by the identical user ID, for example, their initials, which must not contain any white-spaces. All data gathered from a subject will be saved in a cumulative fashion, on a one-separate-file-per-subject basis, see section 4.1.

The subjects can run the experiment at their own pace. They may answer in one of the ways described in section 3.6, as specified by the experimenter.

Analysis of your data can be performed at any later time, independently from the subject performing the measurement.

3.8 Setting up interleaved-tracks experiments

Designing an interleaved-tracks experiment requires to write a main script, a set script, and a user script as well. Most things are just same as for a single-run non-interleaved experiment, but the following exceptions apply:

1. The variable `M` needs to be an *array*, instead of a scalar variable, the number of elements of `M` being equal to the desired number of simultaneous interleaved tracks of the experiment.

   In the main script, it is *not* necessary to copy all fields of the struct variable `M(1)` to those of `M(2)`, `M(3)`, ... for the 2nd, 3rd, ... concurrent run. Instead, it is sufficient to only assign the "relevant" fields of `M(2)`, `M(3)`, ... , `M(end)`, i.e. those fields with a different content for a different run. When doing this, any fields of `M(2:end)` that are left unspecified will be assigned with the same values as found in `M(1)` automatically. (This is a specific feature of PSYLAB, not of Matlab.)

2. The script `mpsy_intrlv_afc_main.m` is called to control the whole interleaved AFC experiment, instead of `mpsy_afc_main.m`.

   Likewise, the script `mpsy_intrlv_match_main.m` is called to control the whole interleaved matching experiment, instead of `mpsy_match_main.m`.

`mpsy_intrlv_afc_main.m`

rename variable `M` into `MI`,   clear variable `M`
for `i=1` to (number of interleaved tracks)
assign `M ← MI(i)`
call *set script* of current experiment
assign `MI(i) ← M`
while (not all tracks completed)
select random track number *i* among all incomplete tracks.
assign `M ← MI (i)`
call *user script* of current experiment
present signals `m_ref` and `m_test` in *n*-AFC manner
process subject response
apply adaptive rule, change `M.VAR` and `M.STEP` accordingly
check whether current track is completed
assign `MI(i) ← M`
protocol all data, save results to `psydat`-file

Figure 5: Principle flow diagram of an interleaved-tracks experiment, as controlled by the script `mpsy_intrlv_afc_main.m`

The following example code illustrates how to set up an interleaved-tracks experiment. The example shows the main script for the tone-in-broadband-noise experiment (see in the subdirectory `examples`) in which a sinusoidal tone of different frequencies is to be detected in a running broad band noise of fixed bandwidth and level.

```
mpsy_init;

M.EXPNAME       = mfilename;    % experiment name, very important to be set correctly!
M.NUM_PARAMS    = 2;           % number of parameters
M.PARAMNAME(1) = {'testtone_frequency'};
M.PARAMUNIT(1) = {'Hz'};
M.PARAMNAME(2) = {'noise_level'};
M.PARAMUNIT(2) = {'dB'};
M.VARNAME       = 'testtone_level';
M.VARUNIT       = 'dB';
M.TASK          = 'which interval contained the test signal (1,2,3)?  ';
M.MINSTEP       = 1 ;  % minumum step size, in units of M.VARUNIT
M.NAFC          = 3 ;  % number of forced choice intervals
M.ADAPT_METHOD = '1up_2down';   % select adaptive method: transformed 1-up-2-down
M.MAXREVERSAL   = 4 ;  % number of reversals required as stop criterion
M.FEEDBACK      = 1 ;  % 1/0  = yes/no
(...)
M.FS              = 48000; % sampling frequency
M.USE_GUI         = 1;     % use a GUI for user input

M.SNAME = input('\n\n please type your name (initials, no spaces, press RETURN at end) ','s');
```

```
tone_freqs = [ 200 400 800 1600 3200 6400 ];

% assign different parameter sets for interleaved tracks experiment
for kk = 1:length(tone_freqs),
    M(kk).PARAM(1) = tone_freqs(kk);
    M(kk).PARAM(2) = -30;              % use the same noise level for all freq.'s
end

mpsy_intrlv_afc_main;
```

The first part looks exactly the same as for a single-run experiment. The difference starts in the for-loop, where M(kk).PARAM(1) and M(kk).PARAM(2) are assigned, thereby rendering M into an array with length(tone_freqs) elements (in this case 6 elements). After the for-loop, most fields of M(2:6) are still empty (by standard Matlab behaviour). These fields will be filled in automatically by way of the script mpsy_intrlv_init.m which is called inside mpsy_intrlv_afc_main.

The script mpsy_intrlv_afc_main will start an experiment with a number of 6 interleaved tracks (in this example). It needs to be called only once and it will exit after the last of the concurrent interleaved tracks has finished. The principle organisation of an interleaved-tracks experiment is illustrated by the flow diagram in Fig. 5.

Note that the set script and the user script of an experiment can sometimes stay exactly the same as for a single-run experiment. However, all variables (including signals) that need to be different in the different tracks (i.e. different parameter settings) must become a field variable of the current variable M. Then, these fields will be saved into the variable MI, as shown in the flow diagram in Fig. 5. For more details, consult the source code of the files mpsy_intrlv_*.m dealing with interleaved-tracks experiments.

The example of an interleaved-tracks experiment given on page 15 illustrated the use of a number of concurrent parameter value(s) sets, i.e. different tone frequencies in that case. However, this is not the only possibility offered by interleaved-tracks experiments in PSYLAB. Another example is given in the file(s) match_freq_binaural_intrlv*.m in the examples subdirectory.

When designing your own interleaved experiment with PSYLAB you will probably need to study the files mpsy_intrlv_*.m in more detail. For example, specifying the following lines

```
M(1).ADAPT_METHOD = '1up_2down';
M(2).ADAPT_METHOD = '2up_1down';
M(6).ADAPT_METHOD = '2up_1down';
```

in the main script of some 6-interleaved-tracks experiment with different adaptive rules in the different tracks would lead to an auto-filled-in value of '1up_2down' for the field ADAPT_METHOD of M(3) through M(5) as these fields are empty so far. This is carried out by mpsy_intrlv_init.m which will always set empty fields to the corresponding value of M(1), being '1up_2down' in the above example.

3.9   Avoid the use of built-in function sound for playback

Unfortunately, in Version 2011 of Matlab the audio handling was redesigned in a way that made the built-in function sound almost non-functioning.

While this bug was fixed again by Matlab, a somewhat nicer and independent work-around for this annoying problem (and for others as well) is to use an alternative method to send your audio samples to your soundcard: "msound". msound is a mex-file, also developed at the IHA, that uses the PortAudio-Interface (of http://www.portaudio.com/) for communicating with your sound card. More information about msound can be found at the website of the Institute for Hearing Technology and Audiology (http://www.hoertechnik-audiologie.de/, then look for "Institut für Hörtechnik und Audiologie" → "Software"), and on github.

The msound mex-file is now also included in the PSYLAB-distribution. It comes in several pre-compiled versions for win, gnu/linux-systems and mac, mostly in 32 and 64bit format. The compiled binary files can also be found on github under "releases". The 'dll'-format (not on github any longer) is needed for older Matlab versions on win32-system. If you encounter a warning about a "shadowing" effect of some msound-file over another, e.g. the 'mexw32' version shadowing the 'dll' version, you may just delete the 'dll' version. In order to enable sound output via msound, only a few additional PSYLAB-variables need to be set:

```
M.USE_MSOUND       = 1;      % flag whether or not to use msound (1 or 0)
M.MSOUND_DEVID     = 0;      % device ID for msound (choose 0 or [] for default device)
M.MSOUND_FRAMELEN  = 1024;   % framelen for msound (number of samples)
M.MSOUND_NCHAN     = 1;      % number of channels, must match  size(m_outsig,2)
```

The variable `M.USE_MSOUND` works as a flag-variable whether or not to use `msound` instead of `sound`.

The frame length `M.MSOUND_FRAMELEN` is the number of samples within each frame of audio, also called block size, that is sent to the soundcard. You should test/experiment, which `M.MSOUND_FRAMELEN` works best on your own system. The value of 1024 worked well for most purposes on many machines/soundcards, but maybe not for yours.

It is important to set the `M.MSOUND_NCHAN` to the number of channels of your stimuli, i.e. 1 for mono/diotic and 2 for stereo/dichotic signals. If `M.MSOUND_NCHAN` does not match `size(m_outsig,2)`, an error will occur.

When using sound output via `msound` in PSYLAB, it is possible to use the visual interval indication. Put another way round, on some systems it will only be possible to use the visual interval indication *if* you use `msound` for sound output.

3.10   Debugging your experiment

For purposes of debugging, set the variable `M.DEBUG` to a value larger than zero (1, 2, 3, 4), to get an increasing amount of temporary information and plots. Stick to the rule: "Use the force – read the source" to see the behaviour in case of `M.DEBUG`>0.

When gathering real data with test subjects, however, you will want to set `M.DEBUG=0;`

4   Saving and analysing your exerimental data

4.1   How your data are saved: the 'psydat' file

Each time when one run has been completed, i.e. a threshold has been reached in an adaptive experiment or a block of trials in a constant stimulus experiment has been finished, the result of that run is automatically saved in a result file in ASCII-format. The name of that file is always formed by the common prefix 'psydat_' followed by the value of `M.SNAME`, i.e. the subject name. It is important for correct functioning of PSYLAB to avoid any whitespace characters in the subject name. Example: `M.SNAME = 'mh'` yields the filename `psydat_mh`.

The psydat-file is written in a special fixed format. One important aspect of this format is the fact that all data entries in the file are delimited by whitespace characters. Consequently, this requires that certain PSYLAB-variables (for example, parameter names or units) *must not contain whitespace characters*. You should not try to change this behaviour, as this would likely interfere with the mechanism for analysis and display of (all) results. It is mandatory that the precise format of the psydat file is maintained. This means that comments, extra lines, or missing lines are not allowed as these would very likely crash the mechanism for automatic data extraction from the psydat file (see section 4.2).

4.1.1   Adaptive experiments

The following data will be saved in the psydat-file after each completion of one run: experiment name; subject name; date and time; number of stimulus parameters; name, value, and unit of each parameter; name of adaptive rule; name of stimulus variable, threshold value, its standard deviation and minimal and maximal value, unit of stimulus variable. The result entry of one run will span across several lines in the psydat-file. Have a look into one of the example psydat-files in the `examples` subdirectory. One example entry in the psydat file belonging to the experiment `sam_sincarrier_detect.m` is the following:

```
##adapt## sam_sincarrier_detect mh 08-Jan-2006__18:46:03 npar 2 ####
%%----- PAR1: modulation_frequency 16.000000 Hz
%%----- PAR2: carrier_frequency 400.000000 Hz
%%----- ADAPT: 1up_2down
  modulation_degree -31.000000 1.278275 -34.000000 -29.000000 dB
```

This entry informs you that the subject named 'mh' performed the *adapt*ive experiment 'sam_sincarrier_detect' on the 8th of January 2006 in the early evening. The experiment had 2 parameters which are specified in rows 2 and 3, and it used the adaptive 1-up-2-down rule. The result was a threshold value at $m=$ -31 dB, with a standard deviation of 1.28 dB. The minimal respectively maximal values of the modulation degree during the measurement phase were -34 dB respectively -29 dB.

The default behaviour of PSYLAB is to calculate the *threshold* of an adaptive run as the *median* of all values of M.VAR during the measurement phase, and to output this value into the psydat file. However, by setting the variable M.SAVEMEAN to 1 (meaning *true*), you can change this and specify that the threshold should rather be calculated as the *mean* of all values of M.VAR during the measurement phase. The default behaviour can be achieved by letting M.SAVEMEAN be an empty field or by setting it to the value 0 (meaning *false*).

The default behaviour of PSYLAB is to output only the resulting threshold value of an adaptive run to the psydat-file. However, by setting the variable M.SAVERUN to 1, you can change this and specify that also all values of M.VAR during the entire adaptive run, including the corresponding answers of the subject, should be output to the psydat-file in a separate line. One entry in the file could then look like this:

```
##adapt## sam_sincarrier_detect mh  22-Nov-2016__17:14:50 npar 2 ####
%%----- PAR1: modulation_frequency 16.000000 Hz
%%----- PAR2: carrier_frequency 800.000000 Hz
%%----- ADAPT: 1up_2down
%%----- VAL: -8 1 -8 1 -12 1 -12 1 -16 1 -16 1 -20 1 -20 1 -24 1 -24 1 -28 0 -24 1 -24 1 -26 0
-24 1 -24 1 -25 1 -25 1 -26 0 -25 1 -25 1 -26 0 -25 1 -25 1 -26 0 -25 1 -25 1
  modulation_degree -25.000000 0.492366 -26.000000 -25.000000 dB
```

The line starting with %%----- VAL: informs you that the subjects listened to stimuli with the following variable values resp. modulations degrees (and gave answers in parentheses) during this run:
-8 dB (correct); -8 dB (correct); -12 dB (correct); -12 dB (correct); -16 dB (correct); etc. The run consisted of 27 trials, of which the last three were at -26 dB (wrong); -25 dB (correct); -25 dB (correct).

4.1.2   Constant stimuli experiments

In case of a constant stimuli experiment, one experimental run will produce several entries in the psydat-file, one per variable value. An example entry, performed with the experiment jnd_frequency.m, looks like this:

```
##const## jnd_frequency mh4 22-Mar-2017__12:55:13 npar 2 ####
%%----- PAR1: reference_frequency 250.000000 Hz
%%----- PAR2: tone_level -10.000000 dB
%%----- CONST: num_presentations 5
  rel_frequency_increment 4.000000 cent   prob_correct 0.600000
##const## jnd_frequency mh4 22-Mar-2017__12:55:13 npar 2 ####
%%----- PAR1: reference_frequency 250.000000 Hz
%%----- PAR2: tone_level -10.000000 dB
%%----- CONST: num_presentations 5
  rel_frequency_increment 8.000000 cent   prob_correct 0.800000
##const## jnd_frequency mh4 22-Mar-2017__12:55:13 npar 2 ####
%%----- PAR1: reference_frequency 250.000000 Hz
%%----- PAR2: tone_level -10.000000 dB
%%----- CONST: num_presentations 5
  rel_frequency_increment 16.000000 cent   prob_correct 0.800000
##const## jnd_frequency mh4 22-Mar-2017__12:55:13 npar 2 ####
%%----- PAR1: reference_frequency 250.000000 Hz
%%----- PAR2: tone_level -10.000000 dB
%%----- CONST: num_presentations 5
  rel_frequency_increment 32.000000 cent   prob_correct 1.000000
```

Note that these four entries were written to the psydat file at the same time, namely after all 5 presentations for each of the four different values of the variable (4, 8, 16, and 32 cent) were completed. These

four data points can form a part of the psychometric function. Any calculation of thresholds or fitting of a psychometric function from theses data is left to the user of PsyLab.

For a constant stimulus experiment, the default behaviour of PsyLab is to output the resulting pairs of variable/$P_{correct}$ to the psydat-file. As for an adaptive experiment, you can change this by setting the variable M.SAVERUN to 1 to specify that also all values of M.VAR during the entire constant stimuli run, including the corresponding answers of the subject, should be output to the psydat-file in a separate line. One set of entries in the file could then look like this:

```
##const## jnd_frequency mh 12-Mar-2019__15:40:42 npar 2 ####
%%----- PAR1: reference_frequency 250.000000 Hz
%%----- PAR2: tone_level -10.000000 dB
%%----- CONST: num_presentations 3
  rel_frequency_increment 4.000000 cent    prob_correct 0.333333
##const## jnd_frequency mh 12-Mar-2019__15:40:42 npar 2 ####
%%----- PAR1: reference_frequency 250.000000 Hz
%%----- PAR2: tone_level -10.000000 dB
%%----- CONST: num_presentations 3
  rel_frequency_increment 16.000000 cent   prob_correct 1.000000
##const## jnd_frequency mh 12-Mar-2019__15:40:42 npar 2 ####
%%----- PAR1: reference_frequency 250.000000 Hz
%%----- PAR2: tone_level -10.000000 dB
%%----- CONST: num_presentations 3
%%----- VAL: 32 1 16 1 32 1 4 0 16 1 16 1 32 1 4 0 4 1
  rel_frequency_increment 32.000000 cent   prob_correct 1.000000
```

In this short example, three different variable values (being 4, 16, and 32 cent) were presented three times each. Note that the line starting with `%%----- VAL:` is present in the file only once, close to the end of the last of the three entries. However, it informs you about all the 3·3=9 presented stimuli that produced the three entries, one for 4 cent, 16 cent, and 32 cent as the stimulus variable.

Should you ever want to perform the same PsyLab experiment both using an adaptive method and also using the constant stimuli method, then this is possible by just changing a few lines in the main script of the experiment. However, you should then use two different subject names for the same subject, e.g. "mh**a**" for subject "mh" in the **a**daptive experiment and "mh**c**" in the **c**onstant stimuli experiment. This will allow the functions described in section 4.2 to keep the two different kinds of results separate.

4.1.3   psydat-file version

The data format of the psydat-file has seen several versions. The current psydat-file-version "3" has been in use since PsyLab-version 2.8. The functions for reading the psydat-file described in section 4.2 need to match with the format version of the psydat-file. Section 4.2.6 explains how to read data from an "old" psydat file that was produced with PsyLab older than version 2.8.

4.1.4   Temporary files

In addition to the psydat-file in ASCII format, the variables M and M_* can also get saved as a binary mat-file each time a run has been completed. This behaviour will only be turned on if M.DEBUG>0. However, this is mainly for rescue and special purposes. They will not be used by PsyLab for later data analysis, and so the binary files may as well be deleted. If you do not need these temporary files, just set M.DEBUG=0;. When M.FEEDBACK>1, then the functions mpsy_plot_feedback.m, mpsy_plot_thresholds.m, and mpsy_plot_psycfunc.m will append a copy of the resulting figure in eps format to a file named 'plot_*_<*subjectname*>'

4.2   Processing and displaying your results

Every time when a subject has completed one run, that run's result is documented in the "psydat" file, see section 4.1. All results can later be analysed and displayed in several ways by using one of the following PsyLab functions:

read_psydat.m Extract from the psydat file all those data points that belong to one subject and one experiment.

display_psydat_raw.m Plot all individual data points for one subject and one experiment.

display_psydat.m Plot all threshold results for one subject and one experiment, individually averaged across all repetitions for identical parameter value combinations.

These three functions are explained in the following subsections.

4.2.1   read_psydat.m

The script `read_psydat.m` performs the action to read the corresponding psydat file of the subject and to extract from it all those lines that belong to the given experiment name (and of course subject name). Lines that pertain to a different experiment are simply skipped by `read_psydat`. The extracted data are returned in two different data formats. After the call

```
[x,y] = read_psydat(s_name, exp_name)
```

variable `x` contains a struct variable with all data extracted from the psydat file. Each struct field is a vector or cell string with as many elements as the number of data points in the psydat file. A lot of information contained in `x` is therefore quite redundant. Variable `y` stores only the subset of numerical data of `x`, in a matrix format which is explained in the verbose output of `read_psydat`. See also the following section.

4.2.2   display_psydat_raw.m

You can have a look at all individual data points that were accumulated over time for one subject in an experiment. In order to plot *all* data points *individually*, use the script `display_psydat_raw.m`. This script makes use of `read_psydat.m` internally. Aside from plotting, `display_psydat_raw.m` also returns the data that were extracted from the psydat file in the same way as done by `read_psydat`.

The output from calling `display_psydat_raw.m` might look like this and produce a plot as shown in Fig. 6:

```
>> [x,y] = display_psydat_raw('mh', 'sam_sincarrier_detect')
*** info:  found 19 matching entries in psydat_mh
*** info:  meaning of the 6 cols of y:
        threshold, thres_sd, thres_min, thres_max, par1, par2
x =
          date: {1x19 cell}
           par: [1x2 struct]
       varname: {1x19 cell}
     threshold: [-31 -35 -48 -34 -29 -50 -31 -32 -32 -32 -34.5 -31 -40 -54 -32 -52 -32 -34 -31]
      thres_sd: [1x19 double]
     thres_min: [-34 -36 -50 -36 -30 -52 -33 -33 -36 -35 -35 -33 -42 -55 -33 -54 -34 -35 -32]
     thres_max: [-29 -32 -47 -32 -27 -49 -30 -28 -24 -29 -33 -30 -37 -53 -31 -51 -31 -33 -30]
        varunit: {1x19 cell}
y =
  1.0e+003 *
  -0.0310    0.0013   -0.0340   -0.0290    0.0160    0.4000
  -0.0350    0.0010   -0.0360   -0.0320    0.0640    0.4000
  -0.0480    0.0010   -0.0500   -0.0470    0.2560    0.4000
  -0.0340    0.0012   -0.0360   -0.0320    0.0160    1.6000
  -0.0290    0.0010   -0.0300   -0.0270    0.0640    1.6000
  -0.0500    0.0008   -0.0520   -0.0490    0.2560    1.6000
  -0.0310    0.0008   -0.0330   -0.0300    0.0160    3.2000
  -0.0320    0.0014   -0.0330   -0.0280    0.0640    3.2000
  -0.0320    0.0034   -0.0360   -0.0240    0.2560    3.2000
  -0.0320    0.0017   -0.0350   -0.0290    0.0160    0.4000
```

```
-0.0345    0.0008   -0.0350   -0.0330    0.0160    1.6000
-0.0310    0.0008   -0.0330   -0.0300    0.0160    0.4000
-0.0400    0.0016   -0.0420   -0.0370    0.0640    0.4000
-0.0540    0.0007   -0.0550   -0.0530    0.2560    0.4000
-0.0320    0.0006   -0.0330   -0.0310    0.0640    1.6000
-0.0520    0.0009   -0.0540   -0.0510    0.2560    1.6000
-0.0320    0.0009   -0.0340   -0.0310    0.0160    3.2000
-0.0340    0.0007   -0.0350   -0.0330    0.0640    3.2000
-0.0310    0.0007   -0.0320   -0.0300    0.2560    3.2000
```

As the experiment of this example comprised two parameters, the output `y` has 4+2 columns. These are: threshold, standard deviation of threshold, the minimal and maximal value of the stimulus variable that occurred during the measurement phase, and the values of parameter 1 and parameter 2. Figure 6 shows an errorbar-plot of the first and second column of `y` as a function of the 5th column (parameter 1), separately for each value of the 6th column (parameter 2), while columns 3 and 4 are not shown in the plot.
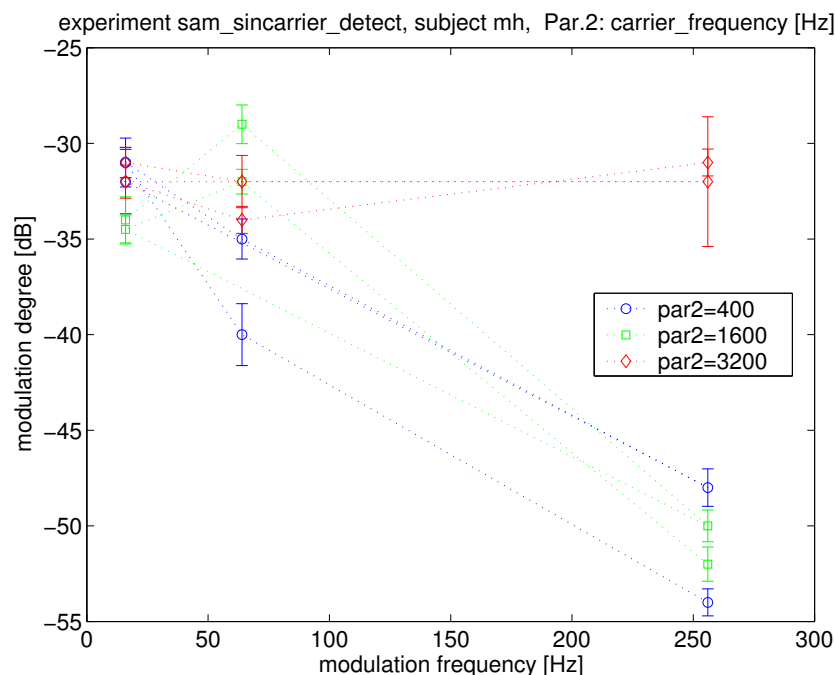
Figure 6: Example plot resulting from calling `display_psydat_raw`. Every single data point (threshold) obtained by the subject is displayed separately.

4.2.3   display_psydat.m

The normal way to process and take a look at your experimental data results is to use the function `display_psydat.m`. This function makes use of `read_psydat.m` to extract all corresponding data from the psydat file. Subsequently, the data handling differs for adaptive experiments and constant stimuli experiments.

Adaptive experiments

For an adaptive experiment, all thresholds that were measured repeatedly are first averaged, individually per each (combination of) different parameter value(s), and then plotted. The first and second argument of `display_psydat` specify the subject name and experiment name.

Calling `display_psydat` with the same input arguments as for `display_psydat_raw` produces four return arguments:

```
[x_all, y_average, y_ave2, hplot] = display_psydat(s_name, exp_name)
```

The output x_all contains the same data as described for output x from read_psydat, see above, but sorted by the value of the 2nd parameter (and 3rd, 4th parameter, if present). Output y_average contains the same data as output y of read_psydat, but *averaged* across repeated measurements with the same set of parameter values. Output y_ave2 contains the same data as y_average, but in a matrix format, where each final data point to be plotted is contained in one row. The last output hplot contains an array of handles to all plot lines in the figure.
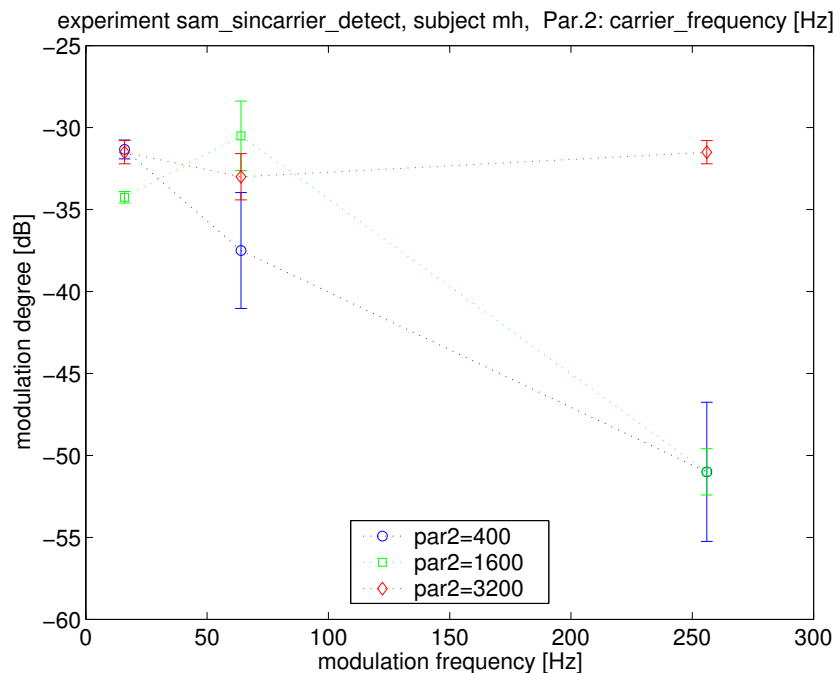
Figure 7: Example plot resulting from calling display_psydat for the same experiment and subject as shown in Fig. 6. All data points (thresholds) pertaining to repeated measurements with the same set of parameter values have been averaged across prior to plotting.

The output from calling display_psydat.m might look like this and produce a plot as shown in Fig. 7. Compare this output and the data to those produced by display_psydat_raw.m (see page 20).

```
>> [x_all,y_average,y_ave2] = display_psydat('mh', 'sam_sincarrier_detect')
*** info:  found 19 matching entries in psydat_mh
*** info:  meaning of the 6 cols of y:
           threshold, thres_sd, thres_min, thres_max, par1, par2
x_all =
1x3 struct array with fields:
    psydata
y_average =
1x3 struct array with fields:
    psydata
y_ave2 =
  1.0e+003 *
   -0.0313    0.0006   -0.0340   -0.0293    0.0160    0.4000
   -0.0375    0.0035   -0.0390   -0.0345    0.0640    0.4000
   -0.0510    0.0042   -0.0525   -0.0500    0.2560    0.4000
   -0.0343    0.0004   -0.0355   -0.0325    0.0160    1.6000
   -0.0305    0.0021   -0.0315   -0.0290    0.0640    1.6000
   -0.0510    0.0014   -0.0530   -0.0500    0.2560    1.6000
   -0.0315    0.0007   -0.0335   -0.0305    0.0160    3.2000
```

```
    -0.0330     0.0014    -0.0340    -0.0305     0.0640     3.2000
    -0.0315     0.0007    -0.0340    -0.0270     0.2560     3.2000
```

A comparison of `x_all(1).psydata` and `y_average(1).psydata` respectively the first three rows of `y_ave2` illustrate the work of `display_psydat`:

```
>> x_all(1).psydata
ans =
   -31.0000     1.2783    -34.0000    -29.0000     16.0000    400.0000
   -32.0000     1.6855    -35.0000    -29.0000     16.0000    400.0000
   -31.0000     0.7878    -33.0000    -30.0000     16.0000    400.0000
   -35.0000     1.0463    -36.0000    -32.0000     64.0000    400.0000
   -40.0000     1.6204    -42.0000    -37.0000     64.0000    400.0000
   -48.0000     0.9783    -50.0000    -47.0000    256.0000    400.0000
   -54.0000     0.7050    -55.0000    -53.0000    256.0000    400.0000
>> y_average(1).psydata
ans =
   -31.3333     0.5774    -34.0000    -29.3333     16.0000    400.0000
   -37.5000     3.5355    -39.0000    -34.5000     64.0000    400.0000
   -51.0000     4.2426    -52.5000    -50.0000    256.0000    400.0000
```

Here, the values -31.3333 and 0.5774 in the first row of `y_average(1).psydata` (or of `y_ave2`) are the mean and the standard deviation across the thresholds that belong to the three repetitions of the experiment with 1st parameter=16 Hz and 2nd parameter=400 Hz. The values -37.5000 and 3.5355 are the mean and the standard deviation of the two repetitions belonging to 1st parameter=64 Hz and 2nd parameter=400 Hz, etc.

Note that the standard deviation of each experimental run, appearing in the 2nd column of the output argument `x_all(:).psydata`, is *neglected* by `display_psydat` regarding the averaging across repetitions.

Note also that the information about the adaptive rule belonging to each data point is *neglected* by `display_psydat` regarding the averaging across repetitions. If you design an experiment where you employ more than one adaptive rule, e.g. 1-up-2-down and 2-up-1-down (with the aim to average the separate thresholds belonging to the target probabilities of 70.7% and 29.3%) then you need to store that piece of information in a different way. It can be done by explicitly using a further parameter, as thresholds belonging to different parameter values will be averaged separately by `display_psydat`. As an example, the experiment `match_freq_binaural_singletrack` shows how this extra parameter can be used.

Constant stimuli experiments

Also for a constant stimuli experiment, the first and second argument of `display_psydat` specify the subject name and experiment name. `display_psydat` will first average across repeatedly measured data (i.e. percentage correct values), individually per each (combination of) different parameter value(s), and then plot them. However, the averaging procedure and the output is different from that for adaptive experiments.

One entry respectively one data set in the psydat-file contains the "score" p, i.e. the probability for a correct answer, and the number N of presentations from which this probability was obtained.

When a score p was measured repeatedly, e.g. in repeated sessions on different days, the individual scores are taken as indenpedent measurements, and averaged using a weighting according the number of presentations: Let the score p_i be based on N_i stimulus presentations in measurement number i, with $i = 1, \ldots, n$. The averaged score $\bar{p}$ is then calculated as $\bar{p} = \frac{\sum_i N_i \cdot p_i}{\sum_i N_i}$. That means $\bar{p}$ is interpreted as being based on a total number of $N_{tot} = \sum_i N_i$ stimulus presentations. A higher number of presentations will lead to a better acuity of the estimate $\bar{p}$, e.g. as expressed by a smaller standard error. For a constant stimuli experiment, `display_psydat` will calculate both $\bar{p}$ and also the standard error associated with $\bar{p}$, according to the following reasoning:

Consider N stimulus presentations where the stimulus variable was held constant. For each presentation, the subject's answer can either be correct or wrong and it is assumed that the subjects's performance is constant over time, which means that the probability p for a correct answer stays constant. Let X be the (random) number of correct responses, then X is distributed according to a Binomial or "Bernoulli"

distrubtion with parameters N and p: $X \sim Bi(N, p)$, with expectation value $E(X) = Np$ and variance $Var(X) = Np(1-p)$. Value p is fixed but unknown, and shall therefore be estimated. The estimator of p is then $\hat{p} = X/N$, which is the relative occurrence of correct answers amoung all N presentations. I can be shown that $\hat{p}$ is an unbiased and consistent estimator for p. Also for repeated estimations of p, the averaged score $\bar{p}$ is identical to the optimal estimator $\hat{p}$ for the real (but unknown) value of p. The random variable $\hat{p} = \bar{p}$ can be interpreted as the *average* of a total number of N_{tot} ones and zeros, with X their sum. The expectation value of $\bar{p}$ is then $E(\hat{p}) = E(\bar{p}) = p$ and the associated standard error, i.e. the standard deviation of $\bar{p}$, is $\sigma_{\bar{p}} = \sqrt{\frac{1}{N^2} Var(X)} = \sqrt{p(1-p)/N}$. These two values are output by display_psydat, separately for each different combination of values of stimulus variable and stimulus parameter(s).

The following example data from a constant stimuli experiment with one parameter demonstrates this behaviour. The psydat-file contains the following data:

```
[x,y] = read_psydat('mm', 'demo_experiment')
*** info:  read_psydat found 37 matching entries in psydat_mm
*** info:  meaning of the 4 cols of y:
        prob_correct, num_presentations, variable, par1
x =
                date: {1x37 cell}
                 par: [1x1 struct]
   num_presentations: [5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5]
             varname: {1x37 cell}
                 var: [1x37 double]
             varunit: {1x37 cell}
        prob_correct: [1x37 double]
y =
    0.8000    5.0000  -35.0000    0.0300
    0.8000    5.0000  -40.0000    0.0300
    1.0000    5.0000  -35.0000    0.0300
    0.8000    5.0000  -40.0000    0.0300
    0.6000    5.0000  -45.0000    0.0300
    0.8000    5.0000  -40.0000    0.0300
    0.8000    5.0000  -45.0000    0.0300
    0.8000    5.0000  -35.0000    0.0300
    0.6000    5.0000  -40.0000    0.0300
    0.4000    5.0000  -45.0000    0.0300
    1.0000    5.0000  -35.0000    0.0300
    0.4000    5.0000  -40.0000    0.0300
    0.8000    5.0000  -45.0000    0.0300
    0.8000    5.0000  -35.0000    0.0300
    0.6000    5.0000  -40.0000    0.0300
    0.6000    5.0000  -45.0000    0.0300
    1.0000    5.0000  -35.0000    0.0300
    0.6000    5.0000  -40.0000    0.0300
    0.8000    5.0000  -35.0000    0.0300
    1.0000    5.0000  -35.0000    0.0300
    1.0000    5.0000  -35.0000    0.0300
    1.0000    5.0000  -40.0000    0.0300
    0.4000    5.0000  -45.0000    0.0300
    1.0000    5.0000  -35.0000    0.0300
    0.8000    5.0000  -40.0000    0.0300
    0.6000    5.0000  -45.0000    0.0300
    1.0000    5.0000  -35.0000    0.0300
    1.0000    5.0000  -40.0000    0.0300
    0.2000    5.0000  -45.0000    0.0300
         0    5.0000  -50.0000    0.0300
    0.8000    5.0000  -38.0000    0.0300
```

```
        0     5.0000   -48.0000      0.0300
   1.0000     5.0000   -34.0000      0.0300
   0.8000     5.0000   -38.0000      0.0300
   0.8000     5.0000   -42.0000      0.0300
   0.4000     5.0000   -46.0000      0.0300
   0.4000     5.0000   -48.0000      0.0300
```

The content of output variable y shows that alls data point were always measured with the same number of 5 stimulus presentations (2nd column in y) and all measurements were made with the same value of 0.03 for the first parameter M.PARAM(1) (4th column in y). The scores (which are proportions, resp. estimated probabilities, all ranging between 0 and 1) are given in the 1st column und the values of the stimulus variable (ranging between -50 and -34) are given in the 3rd column.

When these data are analzed by `display_psydat_raw`, the number N of presentations and the score p for each data point is taken to calculate the standard error $\sigma_{\bar{p}}$ of the score p, as described above. The output then looks like this:

```
[x,y] = display_psydat_raw('mm', 'premasking_sinusoid_const')
*** info:  read_psydat found 37 matching entries in psydat_mm
*** info:  read_psydat: meaning of the 4 cols of 2nd output:
           prob_correct, num_presentations, variable, par1
*** info:  display_psydat_raw: meaning of the 2nd column of 2nd output changed from
           number of presentations to estimated standard error of prob_correct
x =
                    date: {1x37 cell}
                     par: [1x1 struct]
                 varname: {1x37 cell}
                     var: [1x37 double]
                 varunit: {1x37 cell}
            prob_correct: [1x37 double]
    std_err_prob_correct: [1x37 double]
y =
    0.8000    0.1789   -35.0000      0.0300
    0.8000    0.1789   -40.0000      0.0300
    1.0000         0   -35.0000      0.0300
    0.8000    0.1789   -40.0000      0.0300
    0.6000    0.2191   -45.0000      0.0300
...
```

Only the first 5 lines of y are shown here. As an example, the value 0.1789 in the first line was calculated as $\sqrt{0.8 \cdot (1 - 0.8)/5}$, where 0.8 was the score and 5 was the number of presentations (see first line of y in the output of read_psydat).

For the same psydat-file, the output of `display_psydat` then looks like this:

```
[y_all, y_ave, y_ave2] = display_psydat('mm', 'premasking_sinusoid_const')
*** info:  read_psydat found 37 matching entries in psydat_mm
*** info:  read_psydat: meaning of the 4 cols of 2nd output:
           prob_correct, num_presentations, variable, par1

*** info:  display_psydat: NOTE that for a constant stimulus experiment the output
           colums of read_psydat and display_psydat have different meanings, and
           output y_all, y_average have different contents in their 2nd column:
            2nd col in y_all (1st output argument) contains
           "num_presentations" of indiv. data, and
            2nd col in y_average (2nd output argument) contains
           "estimated std.err." of averaged data
y_all =
    psydata: [37x4 double]
y_ave =
```

```
    psydata: [9x4 double]
y_ave2 =
         0          0   -50.0000     0.0300
    0.2000     0.1265   -48.0000     0.0300
    0.4000     0.2191   -46.0000     0.0300
    0.5500     0.0787   -45.0000     0.0300
    0.8000     0.1789   -42.0000     0.0300
    0.7400     0.0620   -40.0000     0.0300
    0.8000     0.1265   -38.0000     0.0300
    0.9273     0.0350   -35.0000     0.0300
    1.0000          0   -34.0000     0.0300
```

4.2.4 Special effects: permute/change the look of your data with display_psydat.m

`display_psydat.m` offer an optional third and fourth argument. They only take effect for experiments with more than one parameter: the third argument specifies the "plot style". A value of 1 means: plot all data into one single figure. A value of 2 means: use a new figure for each value of the 2nd parameter (resp. for each unique combination of values of the 2nd, 3rd, ..., nth parameter).

The optional fourth argument of `display_psydat` specifies a permutation of parameters prior to further analysing and plotting them. The meaning of the fourth argument can best be explained by the following examples. Consider the modulation detection experiment of Dau *et al.*[Dau et al., 1999]. The Variables and Parameters could have been defined in the main script as follows:

```
M.EXPNAME       = 'sam_nbncarrier_detect';
M.NUM_PARAMS    = 2;
M.PARAMNAME(1) = {'modulation_frequency'};
M.PARAMUNIT(1) = {'Hz'};
M.PARAMNAME(2) = {'carrier_bandwidth'};
M.PARAMUNIT(2) = {'Hz'};
M.VARNAME       = 'modulation_degree';
M.VARUNIT       = 'dB';
```

You might have measured the same thresholds as Dau *et al.*, and calling `display_psydat` like in

```
display_psydat('demoparam', 'sam_nbncarrier_detect');
```

results in a plot as shown in Fig. 8. Note here that threshold data (values of `M.VAR`) are plotted as a function of the first parameter (values of `M.PARAM(1)`) in Fig. 8, and each value of the second parameter (`M.PARAM(2)`) yields a separate curve. This is the standard behaviour of `display_psydat`. There are 12 curves for the 12 individual values of the 2nd parameter (carrier bandwidth).

If you would like to replot your data so that threshold is plotted as a function of carrier bandwidth, then you would need to permute the numbering of your two parameters, prior to using `display_psydat`. This can in fact be specified by the fourth argument of this function, which must be a permutation of the numbers 1 through n, where n is the number of parameters of your experiment, i.e. `M.NUM_PARAMS`. Calling `display_psydat` like in

```
display_psydat('demoparam', 'sam_nbncarrier_detect', 1, [2 1])
set(gca, 'Xscale', 'log')
```

specifies that the original order of the parameters shall be permuted into `[2 1]`, i.e. the reverse order of the two parameters. The third argument, '1', specifies that all data should go into one single plot figure. The result is a plot like shown in Fig. 9. Note here that only 3 curves are shown, namely one for each value of the (originally first) parameter "modulation frequency", and each curve is now made of 12 data points. The modulation bandwidth now acts as the first parameter in this plot and its title mentions the modulation frequency as the second parameter. If the third argument ("plot style") had been changed into 2 instead of 1, then three separate figures would have resulted instead of the single one shown in Fig. 9.
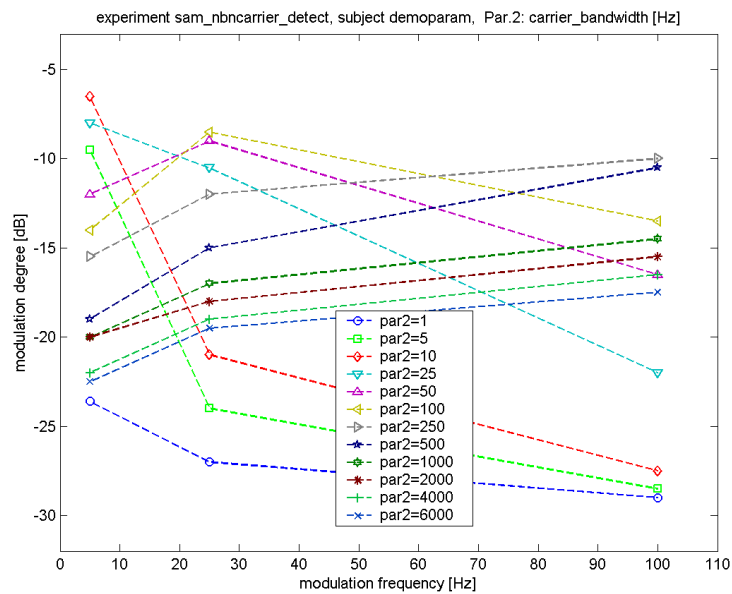
Figure 8: Example data of a modulation detection experiment as described in [Dau et al., 1999] (see text). These data have been taken from Fig. 1 in Dau *et al.* For clarity, standard deviations have been removed.
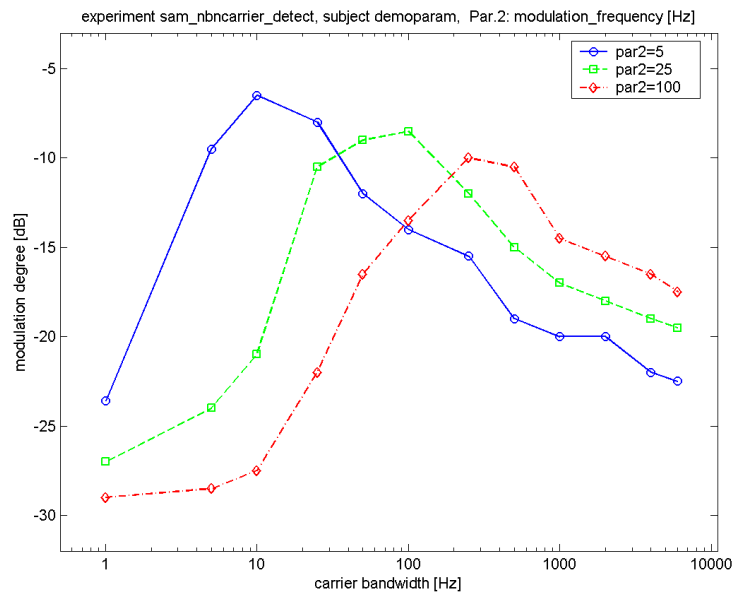
Figure 9: Replot of the same data as shown in Fig. 8 (See also Fig. 2 of Dau *et al.* (1999)). This plot was made by `display_psydat` after permuting the order of the two parameters of the PsyLab-experiment, see text.

4.2.5 psydat_helper.m

After measuring a lot with PSYLAB, you will end up with a larger number of psydat-files, or with very long psydat-files. Eventually you may loose an overview: Which data are stored where? What was the experiment called? Which other experiments are stored in this psydat-file? When did the subject perform those measurements?

   In this case you may get help from the tool `psydat_helper.m`. It opens a GUI and, after a click on the "find all subject names"-button, display the subject names belonging to *all* psydat-files in the current directory. Upon selecting one of the subjects, a second list will pop up, showing *all* experiments that this subject has performed. After selecting one of them, clicking the "plot all data"-button will extract and plot all data belonging to this subject and experiment. As an alternative, you may select a subset of the data, ordered by the date of measurement as shown in the rightmost list. A click on the "extract these and plot"-button below will plot only the subset of data. Data handling and plotting is performed using the regular `display_psydat.m`, as described earlier in this section. A screenshot of the GUI is shown in figure 10.
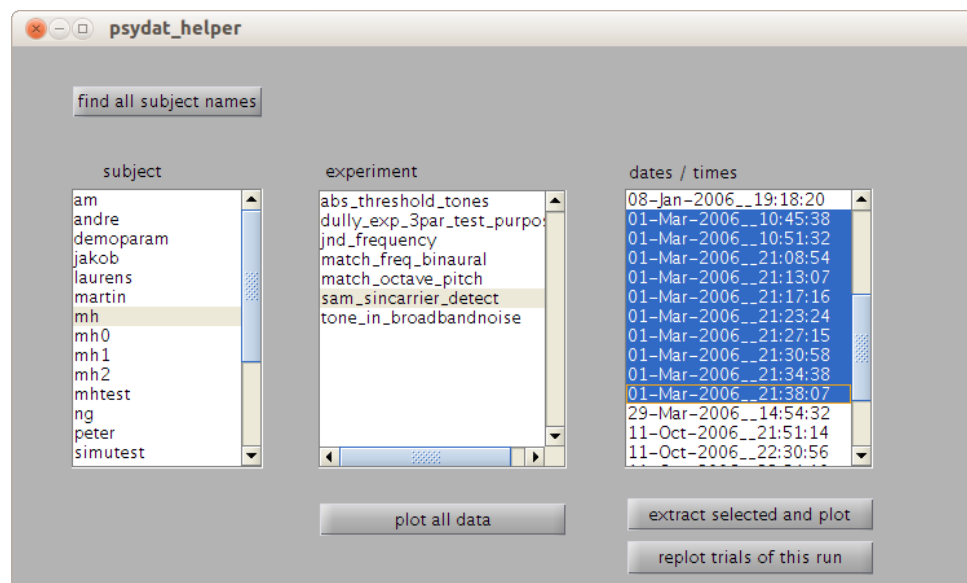
Figure 10: Screenshot of the `psydat_helper` GUI

4.2.6 Reading old psydat files

Should you need to read a psydat file generated with an older version of PSYLAB then the functions read_psydat_v2.m, display_psydat_raw_v2.m, and display_psydat_v2.m are available for reading the data in the older psydat-file-version 2 (cf. section 4.1.3).

5 Variables controlling PSYLAB

PSYLAB is controlled by a number of variables with a fixed, reserved name. All of these variables are struct fields of the variable with reserved name 'M'. Some of these variables are for internal purposes, while others need to be specified by the user according to the needs of the current experiment.

   Table 1 contains a list of all PSYLAB-variables, and explains their use. It is important to remember that some variables are required to be of a certain type, for example, a string, a scalar or a cell string, or must not contain white-space characters. This is described in section 3.5. The required type of each variable is mentioned in parentheses after its name in Table 1.

user variables

Name (type)	Meaning
M.SNAME (s)	subject name, as a string. Preferably only initials. *Important*: must not contain any white-space characters.
M.EXPNAME (s)	experiment name. The experiment name must be identical to the name of the main script (except for the extension '.m').
M.VAR (n)	The value of the stimulus variable, as a scalar number, which is adaptively changed from trial to trial to find the subject's threshold.
M.VARNAME (s)	Name of the stimulus variable M.VAR, e.g. 'test_tone_level'.
M.VARUNIT (s)	Unit of the stimulus variable M.VAR, e.g. 'dB_SPL'.
M.NUM_PARAMS (n)	The number of stimulus parameters. Must be equal to the length of M.PARAM.
M.PARAM (v)	The value resp. values, as a vector, of the stimulus parameter(s) which is/are kept constant during one run until threshold is reached. In case of only one parameter, M.PARAM is of course a scalar.
M.PARAMNAME (c)	Name(s) of the parameter(s), as cell strings, e.g. 'frequency'.
M.PARAMUNIT (c)	Unit(s) of the parameter(s), e.g. 'Hz'.
M.STEP (n)	Value of the current step size by which M.VAR will be changed adaptively depending on the subject's answer. M.STEP is specified in the same unit as M.VAR.
M.MINSTEP (n)	Minimal value of the step size. As soon as the condition M.STEP == M.MINSTEP has been reached, then M.STEP will not be reduced (halved) any longer and the measurement phase starts. At that time, also the counter M.REVERSAL will be reset to zero.
M.MAXREVERSAL (n)	Number of reversals == criterion for the termination of a run. If the condition M.MAXREVERSAL == M.REVERSALS has been reached after the measurement phase has begun, the run is terminated.
M.NAFC (n)	The number of n of intervals in the n-AFC-task.
M.ADAPT_METHOD (s)	The name of the adaptive method of the up-down algorithm. This name must be the suffix of one of the existing mpsy_adapt_*.m PSYLAB scripts, e.g. "1up_2down" for 1-up-2-down according to [Levitt, 1971], "wud" for the "weighted up-down" [Kaernbach, 1991], or "uwud" for the "unforced weighted up-down" [Kaernbach, 2001].
M.PC_CONVERGE (n)	The percentage correct towards which the adaptive methods "wud" and "uwud" (see previous variable and subsection 2.4.3) shall converge. Note, that a number between 0 and 1 is expected, not between 0 and 100! This variable is only used for the "wud" and "uwud" methods.
M.CONSTSTIM_ALLVARS (v)	Vector containing all different (but constant) values of M.VAR during the constant stimulus method.
M.CONSTSTIM_NPRESENTS (n)	Number of repeated presentations, per each value in M.CONSTSTIM_ALLVARS, during the constant stimulus method.
M.FS (n)	The sampling frequency, in Hz.
M.CALIB (v)	Calibration constant = RMS level, in dB SPL, of a full-scale square wave. Note that automatic calibration of your equipment is not yet part of PSYLAB.
M.EARSIDE (n)	Variable specifying the presentation side in case the signals are generated purely mono. Can take one of three predefined values: M_BINAURAL (default value, meaning *diotic* presentation), M_LEFTSIDE, or M_RIGHTSIDE.
M.TASK (s)	Task of the subject, as a string. Typically something like "Which interval contained the test signal?"
M.FEEDBACK (n)	Flag variable (1 or 0), whether or not to provide feedback to the subject (e.g. correct'/not correct') after each answer.
M.USE_GUI (n)	Flag variable (1 or 0), whether or not to use a graphical user interface for the subject's answers instead of keyboard input.

M.VISUAL_INDICATOR (n)	Flag variable (1 or 0), whether or not to provide a visual indication of the individual n intervals of each trial by colour-flashing the individual answer buttons. Works only in case that M.USE_GUI has been set, too.
M.INFO (n)	Flag variable to indicate whether or not additional information about the experiment should be provided to the subject.
M.RESULTSTYLE (n)	Flag variable determining the style of resulting data plots directly after new data have been gathered. A value of 1 (default) means: Plot all data into one figure. Value 2 means: use individual figures for each unique combination of parameter values.
M.SAVEMEAN (n)	Flag variable to indicate that the threshold value should be calculated as the *mean* across M.VAR for trials within the measurement phase, instead of the default behaviour, i.e., calculate their *median*
M.SAVERUN (n)	Flag variable to indicate that also all values of M.VAR and the corresponding answers of the subject should be output to the psydat-file
M.DEBUG (n)	Flag variable (0, or larger) to indicate the amount of intermediate information presented for debug purposes. Should be set to 0 during real work measurements.
M.USE_MSOUND (n)	Flag variable (0=default, or 1) to indicate that 'msound' should be used for sound output instead of Matlab's builtin 'sound'
M.MSOUND_DEVID (n)	The device ID of the sound card (i.e., a number) for use with msound.
M.MSOUND_FRAMELEN (n)	The number of samples of one frame/block for use with msound. msound uses this length for internal buffering and the audio output is sent to the sound card in frames/blocks of this many samples.
M.MSOUND_NCHAN (n)	Number of channels for use with msound. Mono signals have 1 channel, stereo signals have 2, etc. This value must match the value of size(m_outsig,2), otherwise an error will result.

internal variables - not to be manipulated by the user

Name (type)	Meaning
M.QUIT (n)	Flag variable indicting a complete termination of the current experiment.
M.ACT_ANSWER (n)	Current answer of the subject (1=correct, 0=wrong).
M.LAST_ANSWER (n)	Previous answer of the subject (1=correct, 0=wrong).
M.REVERSAL (n)	Number of reversals in the current run. Reversals are detected and counted automatically. Counting starts at 0 again upon start of the measurement phase.
M.REV_IDX (v)	Indices (trial numbers of the current run) at which reversals are detected.
M.UA (n)	Current user answer in AFC experiments, i.e. the interval number, or special values requesting a quit of the run or the experiment.
M.UD (n)	Current user answer in up-down experiment like in matching experiments. Value 1 means 'down', value 2 means 'up'.
M.ANSWERS (v)	Vector with all answers of the subject during the current run.
M.STEPS (v)	Vector with all values of M.STEP during the current run.
M.VARS (v)	Vector with all values of M.VAR during the current run.
M.ALLPARAM (v)	Vector resp. matrix with all values of M.PARAM during the current experiment. Each row of M.ALLPARAM contains the value(s) of M.PARAM pertaining to one run.
M.ALLTHRES_MED (v)	Vector with all threshold values of the current experiment. Each threshold is calculated from the values of M.VARS during each run.
M.ALLTHRES_STD (v)	Vector with all value of the standard deviation of the thresholds during the current experiment. Calculated from the values of M.VARS during each run.
M.DIRECTION (n)	Flag variable for the previous direction of change when applying adaptive up-down-rules. Can take one of the predefined values M.UP, M_DOWN, or M_STAY.

Table 1: List of name and meaning of the PSYLAB-variables,
to be specified by the user or for internal use. The character
in parentheses indicates the required variable type, i.e. s=string,
n=scalar number, v=vector, c=cell string.

6 Helpful scripts in PSYLAB

The PSYLAB distribution contains a few small additional functions that are not needed directly by
PSYLAB, but that can come in handy for signal generation etc. The following scripts/functions are also
described in their `help`.

rms.m Calculate RMS value of a signal vector.

gensin.m Generate sinusoidal tone.

hanwin.m Perform temporal windowing of a signal with a Hanning window resp. generate Hanning-
window-shaped temporal ramps for onset- and offset of a signal.

fft_rect_filt.m Function for very steep bandpass/notch filtering via FFT, by way of setting all Fourier
components outside the passband to zero.

7 New contributions to PSYLAB

Starting with version 2.1, a new directory `contrib` has been added to the PSYLAB distribution. It
contains newly provided contributions to PSYLAB. These contributions comprise additional features for
PSYLAB, like new experimental paradigms, and accompanying example experiments.

The contents of the `contrib` directory should be considered as "experimental" or beta-level-tested.
As time goes by and new versions of PSYLAB are released, the new features might migrate to become
part of the regular PSYLAB-distribution.

The first two contributions were provided by students of "Institut für Hörtechnik und Audiologie".
They implemented two new experimental designs in PSYLAB, namely

1. A procedure to estimate several points on the psychometric function (that means: not only one
   threshold value) by repeatedly presenting stimuli at several fixed values of the stimulus variable, and
   recording the corresponding percent correct responses. The accompanying example experiment is a
   replication of the binaural pitch detection and discrimination experiment by [Santurette and Dau, 2007]

   This has been replaced by new and extended code for the constant stimuli method.

2. A closed-set 1-out-of-n identification procedure. The accompanying example experiment is a repli-
   cation of the melody recognition experiment by [Akeroyd et al., 2001]

Please note that the content of the contrib-directory is checked for consistency with the remaining
part of PSYLAB *far less* frequently!

References

[Akeroyd et al., 2001] Akeroyd, M., Moore, B., and Moore, G. (2001). Melody recognition using three
types of dichotic-pitch stimulus. *J. Acoust. Soc. Am.*, 110(3):1498–1504.

[Dau et al., 1999] Dau, T., Verhey, J., and Kohlrausch, A. (1999). Intrinsic envelope fluctuations and
modulation-detection thresholds for narrow-band noise carriers. *J. Acoust. Soc. Am.*, 106(5):2752–2760.

[Kaernbach, 1991] Kaernbach, C. (1991). Simple adaptive testing with the weighted up-down method.
Perception & Psychophysics, 49 W:227–229.

[Kaernbach, 2001] Kaernbach, C. (2001). Adaptive threshold estimation with unforced-choice tasks. *Perception & Psychophysics*, 63(8):1377–1388.

[Kohlrausch et al., 2000] Kohlrausch, A., Fassel, R., and Dau, T. (2000). The influence of carrier level and frequency on modulation and beat detection threshold for sinusoidal carriers. *J. Acoust. Soc. Am.*, 108(2):723–734.

[Levitt, 1971] Levitt, H. (1971). Transformed up-down procedures in psychoacoustics. *J. Acoust. Soc. Am.*, 49:467–477.

[Santurette and Dau, 2007] Santurette, S. and Dau, T. (2007). Binaural pitch perception in normal-hearing and hearing-impaired listeners. *Hear. Res.*, 223:29–47.

Index

The GNU GPL

to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year> <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  'Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.