MATAA – Mat's Audio Analyzer

MATAA is a free audio analysis tool
for use with MATLAB or GNU Octave

https://github.com/mbrennwa/mataa (`https://github.com/mbrennwa/mataa`)

Manual version: 3 January 2016

Matthias Brennwald `matthias@audioroot.net`

Table of Contents

1 Overview

MATAA (`https://github.com/mbrennwa/mataa`) is a highly flexible and versatile audio analysis system. MATAA uses the computer soundcard (or an external audio interface) to feed a test signal to the device under test (DUT) and to simultaneously record the response signal of the DUT. The response signal is then analysed using one or more of the many tools provided by MATAA. MATAA is extremely flexible and extensible, so that you can make it do exactly what you need (it won't make coffee, though). MATAA runs on all sorts of computer platforms and operating systems (Mac OS X, Windows, Linux, etc.). And, most important, MATAA is free software!

However, MATAA is not just another audio analyser, such as e.g. MLSSA (`http://www.mlssa.com`), CLIO (`http://www.audiomatica.com`), IMP (`http://www.libinst.com`), LAUD (`http://www.libinst.com`), Praxis (`http://www.libinst.com`), Hobby Box (`http://www.audio-software.de`), ARTA (`http://www.fesb.hr/~mateljan/arta`), or MacSpeaker (`http://www.audioroot.net/analysis/MacSpeaker.html`), etc. MATAA is rather a collection of small programs (I will call them *'MATAA tools'* or just *'tools'* from now on) that accomplish small (but sometimes difficult or tedious) tasks to acquire, process, transform, and visualise audio data. These tools are written in the standard and easy-to-understand but very powerful programming language of MATLAB (`http://www.mathworks.com`), a numerical computing environment. Instead of MATLAB, you can also use GNU Octave (`http://www.gnu.org/software/octave`), which is a free MATLAB clone. MATAA runs just as well under either MATLAB or Octave.

The strength of MATAA over other audio analysers is its flexibility. The various MATAA tools can be combined in any way you like. In addition, you can use MATLAB/Octave scripts to to automate a measurement according to your needs and setup, or to expand on the features of MATAA. Several pre-defined scripts to automate 'typical' analyses are provided with MATAA (e.g. measuring the impulse response of a loudspeaker, removing room echoes, and calculating the anechoic frequency response). To get a feeling for MATAA, I recommend you use these scripts as a starting point and modify them as required for your needs. If you are new to MATLAB/Octave, I recommend you take a look at Chapter 6 [Getting started with MATLAB or Octave], page 64. The approach of having to write MATLAB/Octave commands and scripts may seem cumbersome in comparison to interacting with MATAA using some whizz-bang graphical user interface. However, my experience with writing commands and scripts is that it makes me think twice about how my measurement works, which in turn results in a deeper understanding of the data I acquired with MATAA. Also, once a script works as desired, it is easy and very fast to repeat a given measurement procedure.

One noteable advantage of using MATLAB/Octave as a basis for MATAA is that we can use all the available MATLAB/Octave tools for processing, analysis, and plotting of data. MATLAB/Octave can import and export data in various formats, which greatly simplifies the data exchange between MATAA and other software. In addition, MATLAB/Octave provide powerful tools for plotting data, and to export these plots in various graphics file formats.

While MATLAB and Octave both run on a wide variety of operating systems and computer platforms, their audio input/output routines do not work the same on different environments, and, as of this writing, they do not work at all on some operating systems (e.g.

Mac OS X). I therefore designed MATAA such that the audio input/output is handled by one single tool that works differently depending on the computer environment. The user therefore does not need to worry about the audio differences of different plaforms. Furthermore, I wrote a program that handles the audio input/output on Mac OS X. So far, the audio input/output of MATAA has been tested on Mac OS X and on Windows. Linux users reported that audio input/output can be compiled successfully, but I cannot provide specific compilation instructions. Chapter 2 [Installation and Setup], page 3 provides more information on the specific requirements of MATAA regarding the audio hardware and operating systems.

To find out more about MATAA, go to the MATAA homepage at https://github.com/mbrennwa/mataa (`https://github.com/mbrennwa/mataa`).

2 Installation and Setup

Before digging in, I believe the following note is in order: installing MATAA and MATLAB/Octave may be difficult for those who are not experienced computer buffs. If you need help, ask a wizard. If you don't have a wizard at hand, try asking me at `matthias@audioroot.net`.

2.1 MATLAB/Octave

To run MATAA, you need to install either MATLAB or Octave. I leave it up to you to decide on either of those. You can also install both MATLAB and Octave, they can peacefully co-exist on the same computer. MATLAB is an expensive commercial product, and you get what you pay for (see `http://www.mathworks.com` for details). In contrast, Octave is free software, but you still get a lot from it (more than enough for MATAA), see `http://www.octave.org` for details. Furthermore, there is a very helpful mailing list where you can get help and assistance with Octave, see `http://www.octave.org/help`. Depending on your computer platform and operating system, the installation of MATLAB or Octave will be different. Please follow the instructions that come with MATLAB/Octave.

If you decide to run MATAA using Octave, I highly recommend to use Octave 3.0 or later. While earlier MATAA versions were able to run on Octave 2.1 or 2.9, version 3.0 incorporates a large part of Matlab's handle-graphics system. To simplifiy further development of MATAA on both Octave and Matlab, I therefore decided to drop support for the older gnuplot-oriented graphics system in Octave. The plotting routines of current versions of MATAA therefore rely on Octave 3.0 or later.

I recommend to keep all your MATLAB/Octave code and packages in one directory (which may of course contain several subdirectories). This greatly helps MATLAB/Octave to find your files. For MATLAB, the default path for this is `~/MATLAB` (where `~` indicates your home directory). For Octave, there is no default (I believe), but I recommend to use either `~/Octave/` or, if you have both MATLAB and Octave installed and want to keep the MATLAB/Octave files in the same directory, `~/MATLAB/`.

2.2 MATAA

First of all, download MATAA. There are two possiblities:

- Download a recent package file from `https://github.com/mbrennwa/mataa` and expand it if your internet browser or computer didn't do so already. If you later need to update to a more current version, download the most current package file, expand it and replace your previous version with the new one.
- Download the most current version using subversion with the following command:
  `svn checkout https://github.com/mbrennwa/mataa/trunk`
  Then rename `trunk` to `mataa`. On Linux or Mac OS X:
  `mv trunk mataa`
  If you later need to update to the current version, use the following command:
  `svn update`

You should now have a directory `mataa` containing several sub-directories. Make sure `mataa` (and its subdirectories) is located in your default MATLAB/Octave path, which is assumed to be ~/MATLAB/ from now on (see Section 2.1 [Installing MATLAB/Octave], page 3). Your MATAA setup should now look like this (in alphabetical order):

- `~/MATLAB/mataa/documentation/`: This directory contains the MATAA documentation and manual in various formats.
- `~/MATLAB/mataa/mataa_scripts/`: This directory contains various demo and test scripts.
- `~/MATLAB/mataa/mataa_tools/`: This directory contains the MATAA 'tools' (see [MATAA tools], page 1).
- `~/MATLAB/mataa/microphone_data/`: This directory contains files with information on the characteristics of measurement microphones (this data will be used to correct for the microphone characteristics, e.g. for loudspeaker testing).
- `~/MATLAB/mataa/test_signals`: This directory contains various test-signal files.
- `~/MATLAB/mataa/TestTone`: This directory contains the TestTone and TestDevices programs (binaries for Mac OS X and Windows, as well as the source code if you want to compile for other platforms.).
- `~/.mataa_settings.mat`: This file is used to store the 'preferences' of MATAA (e.g. the color to be used for data plotting). Don't worry if this file is missing—MATAA will create it for you.

In addition to these files and paths, you might consider to create an additional path to keep your custom MATAA scripts. I highly recommend to keep this path outside the main MATAA path. Otherwise it will be difficult to upgrade to a newer version of MATAA and you increase the risk of accidentally loosing your custom files during the upgrade process. For instance, I keep my custom MATAA scripts in `~/MATLAB/mataa_user_scripts/`.

If everything set up as outlined above, you are ready to use MATAA from within MATLAB/Octave. However, MATLAB/Octave will (most propably) not find the MATAA files. To tell MATLAB/Octave where the MATAA files are, you can use the `addpath`with older versions of Octave, you may have to use `path`nstead). To automate this task, I recommend to put the necessary commands into the so-called startup file of MATLAB/Octave. This file is executed by MATLAB/Octave everytime MATLAB/Octave is started. You can edit the startup file, which is an ASCII text file, using your preferred text editor:

- For MATLAB, the startup file is `~/MATLAB/startup.m`
- For Octave, the startup file is `~/.octaverc` (note the dot in the file name)

For example, assume you have installed MATAA to `~/MATLAB/mataa/`. Add the following lines to the end of this file:

```
addpath ("~/MATLAB/mataa/mataa_tools");
addpath ("~/MATLAB/mataa/mataa_scripts");
addpath ("~/MATLAB/mataa/test_signals");
```

If you created a directory `~/MATLAB/mataa_user_scripts/` to store your custom MATAA stuff, you may add the following line to let MATLAB/Octave know about this::

```
addpath ("~/MATLAB/mataa_user_scripts");
```

If the path to your MATAA files contains spaces, you will need to add a backslash in front of the space(s). Otherwise MATLAB/Octave will not recognize the space(s) and the `path` commands will fail. For example, if your MATAA files are in `~/My Octave files/mataa`, the above lines would read `addpath ("~/My\ Octave\ files/mataa/mataa_tools/");`, etc.

Also, it is not recommended to install the MATAA files (or any of your personal MATLAB/Octave files) to the path where the MATLAB/Octave program is installed. Once you update your MATLAB/Octave software to a later version, the previous program files may be deleted, and hence your MATAA (or other personal MATLAB/Octave files) will be deleted, too.

2.3 Hardware Setup

The way your sound hardware (soundcard or audio interface) needs to be set up for use with MATAA will depend on its features, on your computer platform, and on the type of measurement you want to make. Furthermore, additional devices (e.g. amplifiers, filters, microphones, etc.) may be needed for certain measurements. Hence, the hardware setup will vary with the type of measurement and the specifics your equipment. This manual therefore aims to provide rather general advice and background on what to watch out for.

That said, you should also remember that a measurement can only be as good as the audio hardware you use!

2.3.1 The Building Blocks of the Measurement Setup

The basic procedure followed during a MATAA measurement is that MATAA feeds a test signal to the soundcard, which is connected to the DUT. The response of the DUT to this test signal is recorded by the soundcard. The response signal is then loaded back into MATAA for further analysis.

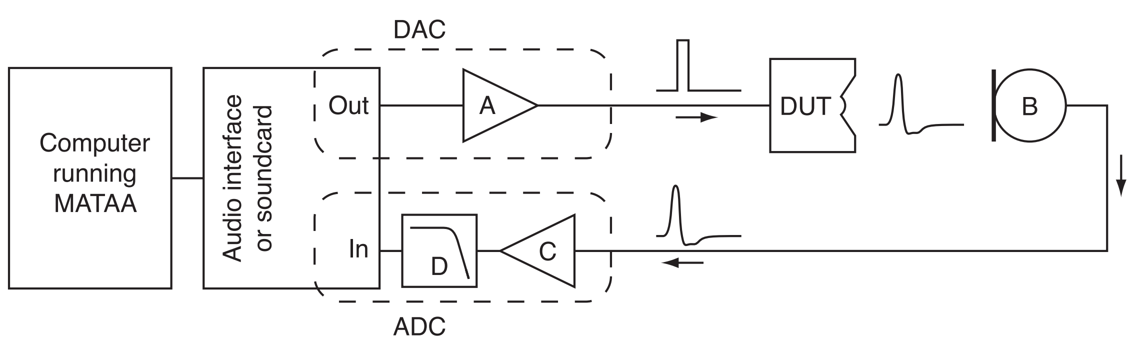

Figure 2.1: 'Generalised' measurement setup. A: buffer or amplifier to drive the DUT (optional), B: sensor to record the DUT's response signal (e.g. microphone), C: output signal buffer or amplifier (e.g. microphone amplifier), D: anti-alias filter (may be omitted in special cases, see text). To calibrate the signal levels, MATAA takes into account the gain and transfer functions of the soundcard (input and output), amplifiers, and the detector; The DAC (digital-to-analog conversion including buffer or amplifier), the sensor (B), and the ADC (analog to digital conversion including buffer or amplifier) are considered separately for calibration (see text).

Figure 2.1 shows the measurement setup, which consists of several building blocks. Depending on the type of measurement and setup, some (or most) of these building blocks are obsolete. In Figure 2.1, the test signal travels through the following blocks:

- The audio output of the soundcard. The the sound hardware is usually set up via the operating system. The output level of the soundcard should be set as high as possible to maximise the signal/noise ratio (SNR). Apart from thatz, the quality of the test signal will depend on the quality of the soundcard (e.g. the D/A converter). Today, most soundcards support sampling rates of 44.1 kHz and bit depths of 16 bits (CD quality), which is fine for many types of measurements. Many soundcards allow sampling rates of 96 kHz or even 192 kHz and sampling depths of 24 bits, therefore providing more headroom with respect to SNR and the upper frequency limit. In some applications (e.g. for low-frequency analyses utilising long test signals), however, low sampling rates are preferable to minimise memory and computing time. Most soundcards allow sampling frequencies as low as 8 kHz. Another aspect of the souncard output is the output impedance and output power. The output impedance should be much lower than the input impedance of the next stage. While most soundcards can easily drive high-impedance headphones, the output impedance may be too high and the output power too low to directly drive a loudspeaker or other low-impedance DUT.
- A buffer or amplifier (A in Figure 2.1). Depending on the DUT, you will need a buffer or amplifier to match the impedance and power loevel to the DUT.
- The device under test (DUT). In principle, this can be anything accepting an electrical sound at its input. Typical MATAA applications include loudspeakers and speaker crossover filters, as well as active devices such as active filters or amplifiers.
- A sensor (B in Figure 2.1) to convert the output signal of the DUT to an electrical signal. For instance, this sensor may be a microphone or an accelerometer (e.g. for loudspeaker testing). If the output of the DUT is electrical (e.g. in case of a filter circuit or an amplifier), the DUT's output should be terminated by a resistor, which can be considered to act as a sensor. This resistor should have the same value as the impedance of the device that would otherwise be connected to the output of the DUT. For testing loudspeaker crossover filters, consider connecting the filter output to the speaker driver(s) rather than a resistor, because the behaviour of the filter may depend on the complex impedance of the driver(s). If the signal voltage from the sensor (or the DUT) is higher than the maximum voltage of the next stage, you will need to attenuate the signal, e.g. using a voltage divider. In some cases (e.g. to analyse high-voltage signals in tube amplifiers), I strongly recommend to add further over-voltage protection to avoid destroying anything!
- A buffer or amplifier (C in Figure 2.1) to match the signal amplitude and impedance level of the DUT response to the input of the soundcard. If the DUT's respons was recorded with a microphone, this will be a microphone amplifier. In many other cases, this buffer/amplifier can be omitted, provided the output impedance of the previous stage (the sensor) is much lower than the input impedance of the next stage.
- The anti-aliasing filter (D in Figure 2.1) removes high-frequency components from the DUT's response signal. If the the DUT response contains signal components with frequencies higher than the Nyquist frequency (half the sampling frequency) of the sound input's analog-to-digital (A/D) converter, these signal components will be aliased to lower frequencies during A/D conversion. This signal 'contamination' can be avoided

(or at least constrained) by removing the signal components higher than the Nyquist frequency *before*/D conversion. Many soundcards have a built-in anti-aliasing filter with a cut-off frequency that is automatically adjusted to the sampling rate. You can check for the presence of an anti-aliasing filter by applying sine signals with frequencies higher than the Nyquist frequency (e.g. using an analog signal generator). Then check the digitized signal for alias signals in the frequency range below the Nyquist frequency. Furhter, if the signal from the DUT is (virtually) free of frequencies higher then the Nyquist frequency, you can omit the anti-aliasing filter. Vice versa, you can omit the anti-aliasing filter, if the soundcard samples the test signal with a sampling rate of at least twice the highest frequency contained in the test signal. For instance, loudspeakers and test microphones rarely extend to frequencies higher than 40 kHz. Thus, if your soundcard allows setting the sampling rate to 80 kHz or higher (e.g. 96 kHz or 192 kHz), you can omit the anti-aliasing filter by using a sampling rate of at least 80 kHz.

- The Soundcard audio input: Here, the same applies as with the audio input, with a few exceptions. Firstly, a high input impedance is preferable so that the previous stage can easily drive the audio input. Secondly, the sensitivity of the analog-to-digital (A/D) converter should be set as high as possible (to maximise SNR), but not too high (to avoid clipping of the signal).
- Signal calibration: MATAA takes into account the sensitivity and gain of the DAC and ADC blocks, as well as the sensitivity and the potentially frequency-dependent transfer function of the signal sensor. See ⟨undefined⟩ [Data calibration], page ⟨undefined⟩.

2.3.2 Soundcard setup, TestTone, and TestDevices

MATAA talks to the soundcard using the TestTone and TestDevices programs (which are part of the MATAA package, see Section 2.2 [Installing MATAA], page 3). If your computer has more than one device for sound input or output, MATAA uses the default device set for your computer.

A few notes:

- The Windows versions of TestTone and TestDevices only work with ASIO drivers (WMME and DirectSound are not supported). If your soundcard did not come with an ASIO driver, check out ASIO4ALL (`http://www.asio4all.com`). The Windows binaries were compiled by Shu Sang (`sangshu@hotmail.com`) – thank you Shu! Please note that Shu used Microsoft Visual Studio to compile TestTone and TestDevices. Therefore, if you experience problems with sound input or output, you may need to install the Microsoft Visual C++ 2005 SP1 Redistributable Package (x86) to make TestTone and TestDevices work properly. You can download the package here: `http://www.microsoft.com/downloads/details.aspx?familyid=200B2FD9-AE1A-4A14-984D-389C36F85647&displaylang=en` (thanks to Gabe for this hint!).
- The Mac OS X versions of TestTone and TestDevices rely on CoreAudio, Apples application programming interface for sound on Mac OS X. CoreAudio provides automatic sample-rate conversion. It is therefore possible to use sample rates with MATAA that are not directly supported by the hardware.
- The Linx versions of TestTone and TestDevices are available for Linux running on Intel and PowerPC machines. If you need to compile your own binaries using the

source code included in the MATAA distribution, see Section 2.6 [Compiling TestTone and TestDevices on Linux], page 12.

2.3.3 Sound channel allocation

Most soundcards have at least two sound channels for stereo sound. While many measurements can be made using only one channel, there are a few cases where the second channel is needed to record a reference signal (e.g. impedance measurements, ⟨undefined⟩ [Impedance measurement], page ⟨undefined⟩). In most other cases, using the second channel to record a calibration signal will allow you to correct for artifacts that may be introduced by the test equipment, which will improve the precision and the quality of the measurement.

While Figure 2.1hows the path of the test signal to and from the DUT, it does not show the path reference signal. The reference-signal path will depend strongly on the type of measurement and the test equipment used. Figure 2.2 is an attempt to illustrate some typical examples.

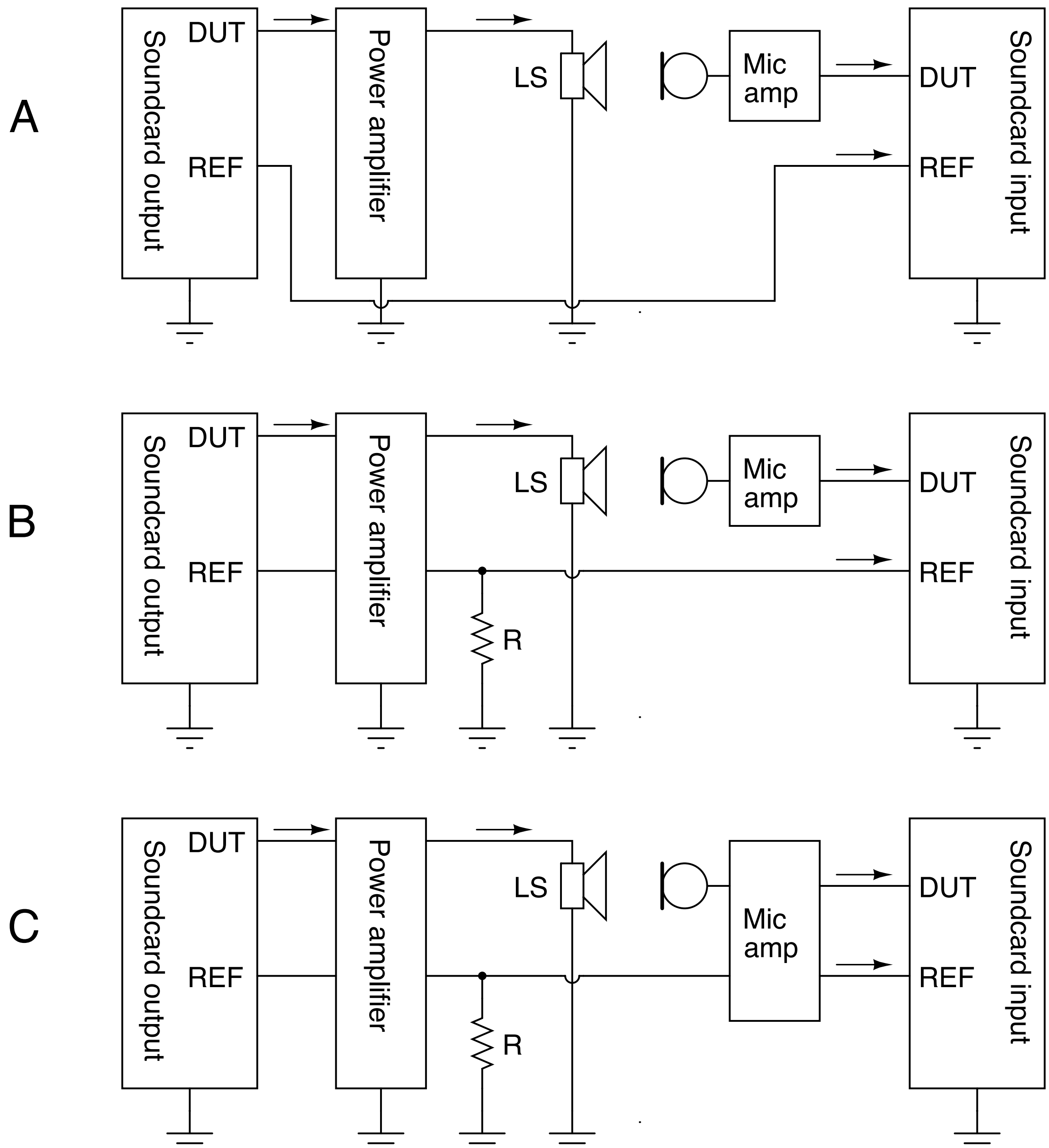

Figure 2.2: Some examples of how to use the second sound channel ('REF') of a stereo soundcard. A: both the power amplifier and the microphone amplifier are mono: wire the REF output directly to the REF input, B: the power amplifier has two (stereo) channels, but the microphone amplifier is mono: use the second channel of the power amplifier to calibrate for its characteristics, C: both the power amplifier and the microphone amplifier are stereo: use the second channel of the power amplifier and the microphone amplifier to calibrate for the characteristics of both amplifiers.

By default, MATAA uses the left channel to record the test signal from the DUT, and the right channel to record the reference signal. If your soundcard uses 3.5 mm jacks, the DUT channel (left) should be on the tip of the 3.5 mm jack. The reference channel (right) should be on the ring in the middle of the jack. The ground (common to both channels) is on the contact closest to the body of the jack Figure 2.3. If the left and right channels are revesed on the connectors of your soundcard, you can adjust the channel allocation using the `mataa_settings` command.

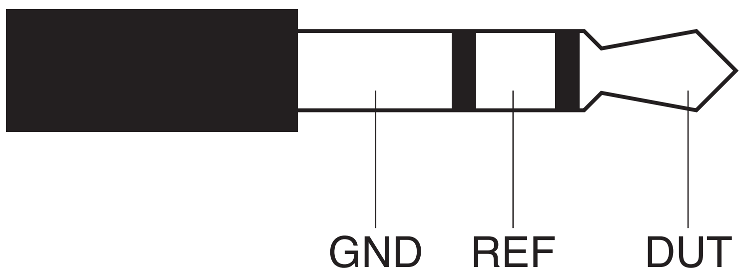

Figure 2.3: Pinout of 3.5 mm jack

2.3.4 Interchannel delay

With some (lesser) soundcards, the data recorded in one channel may be offset in time with respect to the other up to to several tens of microseconds. This effect is called “interchannel delay”. Interchannel delay can result in wrong results from impedance measurements using the sine-sweep method as described in “MATAA: A Free Computer-Based Audio Analysis System” (article in audioXpress (7), 2007).

Therefore, interchannel delay must be removed from the measured data before calculating impedance function from the data. The `mataa_measure_impedance` command, which automates impedance measurement using the mentioned sine-sweep method, takes care of interchannel delay by shifting the measured data in time. The information on the amount of interchannel delay is taken from the MATAA settings file (the `interchannel_delay` field specifies the interchannel delay in seconds). By default, the interchannel delay is set to zero. You can adjust this value using the `mataa_settings` command (see Section 5.62 [mataa_settings], page 52). For instance, with a soundcard exhibiting an interchannel delay of 17 microseconds, the interchannel delay parameter would be set by:

```
mataa_settings('interchannel_delay',17E-6);
```

To test if your soundcard exhibits interchannel delay, it is best to measure the impedance of a resistor with a purely ohmic impedance (i.e. with constant resistance for all frequencies) using the method described in “MATAA: A Free Computer-Based Audio Analysis System” (article in audioXpress (7), 2007). If this measurement gives a flat impedance reading, your soundcard is not affected by interchannel delay (or the interchannel delay is already adjusted properly in the MATAA settings). Otherwise, you need to adjust the interchannel delay setting until you get a flat impedance reading.

2.4 Setting up the soundcard of an Apple Macintosh computer running Mac OS X

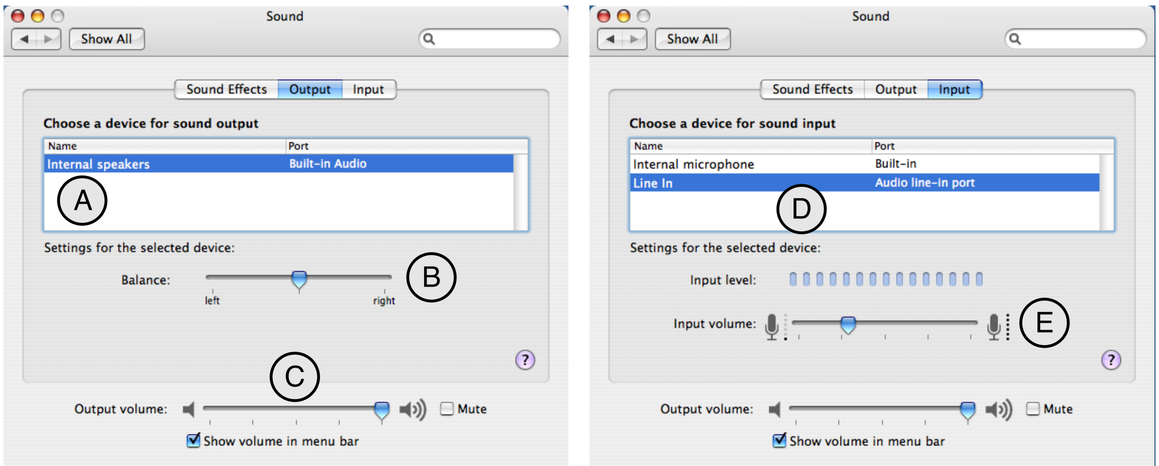

Figure 2.4: Audio hardware setup in Mac OS X (left: audio input, right: audio output). A: select the device to be used for audio output, B: set channel balance to 'balanced', C: set output level as high as possible, D: select the device to be used for audio input, E: set input sensitivity as high as possible, but low enough to avoid clipping of the input signal.

To set up the audio hardware in Mac OS X, choose 'System Preferences' in the Apple menu. Then, click on 'Sound', and follow the instructions in Figure 2.1.

2.5 Setting up the soundcard of a computer running Linux

On Linux, I strongly recommend using ALSA (the Advanced Linux Sound Architecture). Choosing the right sound architecure and sound devices as the default devices (i.e. the devices that will be used by MATAA) depends on the Linux distribution used. Read the documentation for your system. Apart from that, the following terminal commands may be useful to find the relevant information on the sound devices available on your system:

- To display a list of the ALSA sound cards for sound output, including the current default device that will be used by MATAA:
  `aplay -L`
- To display a list of the ALSA sound cards for sound input, including the current default device that will be used by MATAA:
  `arecord -L`
- To display a list of the ALSA sound output devices:
  `aplay -l`
- To display a list of the ALSA sound input devices:
  `arecord -l`
- As an alternative, the following command displays information on the available ALSA devices:
  `cat /proc/asound/devices`

Also, reading the ALSA instructions on http://www.alsa-project.org or http://seehuhn.de/pages/alsa will be helpful. From reading these documents, I found that all I needed to do on my system was to create a file ~/.asoundrc, which specifices the default. As an illustration, this is how the file looks on one of my systems:

```
pcm.!default {
    type hw
    card 0
}
ctl.!default {
    type hw
    card 0
}
```

Without this file, the default device on this system is set to something that thinks that there are 128 sound channels, both for input and output. MATAA therefore produces data for all 128 channels, both for input and output. Because my hardware only has two channels, the remaining 126 channels are somehow merged into two real channels, which takes a lot of CPU power. Sound input and output is therefore very time consuming, and I believe merging 128 data channels into the two sound channels of the hardware is not good for the data integrity. I therefore always double check if MATAA uses the right sound device using the `mataa_audio_info` command. On most systems, the number of sound channels should be 2, and probably not 128.

2.6 Compiling TestTone and TestDevices on Linux

The following instructions may be useful if you need to compile TestTone and TestDevices on Linux. The commands given may need to be adapted to a specific Linux environment, however.

- Download a recent release of the portaudio source code from http://portaudio.com (pick the one marked as 'recommended'). The files are packaged in `*.tgz` file. Extract the files from the `*.tgz` file. In the following example, I stored the portaudio files on my Desktop (`~/Desktop/portaudio/`).
- Compile portaudio with support for the ALSA backend only. I did this using the following commands (other backends, such as OSS, are not recommended for use with MATAA):
```
cd ~/Desktop/portaudio
./configure --with-alsa=yes --with-jack=no --with-oss=no
make
```
- Copy the portaudio library you just compiled and `portaudio.h` to the path where the TestTone source code lives, e.g.:
```
cp lib/.libs/libportaudio.a ~/matlab/mataa/TestTone/source/
cp include/portaudio.h ~/matlab/mataa/TestTone/source/
```
- Compile TestTone and TestDevices using the following commands:
```
cd ~/matlab/mataa/TestTone/source/
gcc -lrt -lasound -lpthread -o TestTonePA19 TestTonePA19.c libportaudio.a
```

gcc -lrt -lasound -lpthread -o TestDevicesPA19 TestDevicesPA19.c libportaudio.a

- Finally, move the binaries you just compiled to the path where MATAA expects to find them (i.e. ~/matlab/mataa/TestTone/LINUX_X86-32, ~/matlab/mataa/TestTone/LINUX_X86-64 or ~/matlab/mataa/TestTone/LINUX_PPC):
  mv TestTonePA19 ../LINUX_X86-64/
  mv TestDevicesPA19 ../LINUX_X86-64/

2.7 Testing the Installation

To test your software installation, first start MATLAB or Octave. Then, type `mataa_selftest` to the MATLAB/Octave command prompt to run a MATAA self-test. `mataa_selftest` is a MATAA script in `.../mataa/mataa_scripts/` that runs several tests, that will display various messages on the success of the tests. Some tests may fail, but that does not necessarily mean that your MATAA installation is broken. If in doubt, carefully read the error or warning messages. If still in doubt, contact me at `matthias@audioroot.net`.

The self-test script also includes a test of the hardware for sound input and output. Details on the setup of the sound hardware are given in Section 2.3 [Hardware Setup], page 5. For now, it will suffice to run the self test with the soundcard input(s) connected directly to the output(s).

If MATLAB/Octave cannot find the `mataa_selftest` script, this most likely indicates that the script file is not on the search path of MATLAB/Octave. Double check the path settings outlined in Section 2.2 [Installing MATAA], page 3. You can also type `path` to the MATLAB/Octave prompt to display the current search path.

3 Working with MATAA

This manual assumes you know what kind of measurements you are after, and why. This manual is not an introduction to acoustic measurement principles. Please refer to other sources to find background information on techniques and methods for measurements in electroacoustic systems. Some documents that I can recommend:

1. J. d'Appolito: Testing Loudspeakers, Audio Amateur Press, Peterborough, New Hampshire, USA, 1987.
2. J. d'Appolito: Testing Loudspeakers: Which Measurements Matter (Parts 1 and 2), audioXpress (9,10), 2008.
3. L. Olson: A MLSSA Gallery, 2006, `http://www.nutshellhifi.com/MLS` (last checked 5 May 2007)
4. M.S. Brennwald: MATAA: A Free Computer-Based Audio Analysis System, audioXpress (7), 2007. Copies of this article are distributed together with MATAA, and are available online, too. The original article is at `http://www.audioxpress.com/magsdirx/ax/addenda/media/brennwald2806.pdf` (last checked 12 Aug 2007), and a version that is somewhat easier to read is at `http://www.audioroot.net/analysis/MATAA_aX_original.pdf` (last checked 12. Aug 2007).

The workflow with MATAA can be separated into two parts. First, you need to figure out how to setup the connections between the DUT and the soundcard (see Section 2.3 [Hardware Setup], page 5). Second, you will type commands into MATLAB/Octave telling MATAA to carry out the tests, process the data, or plot the results. This second step requires you to know how to work with MATLAB/Octave (see Chapter 6 [Getting started with MATLAB or Octave], page 64). You will also need to know the names of the MATAA commands and how they work.

Information on the MATAA commands is available in the Chapter 5 [MATAA tools reference], page 16. You may also use the online help system on MATLAB/Octave by typing `help <command>` into the MATLAB/Octave command prompt. For instance, if you need to know how to use the signal generator command `mataa_signal_generator`, type `help mataa_signal_generator`. This help system is not limited to the MATAA commands, but works for all MATLAB/Octave commands (for example, if you want to find out how to save data from within Matab/Octave to disk, type `help save`).

For a few worked examples on how to use MATAA in real-world applications, please refer to the MATAA article published in audioXpress ("MATAA: A Free Computer-Based Audio Analysis System", included with the MATAA package).

4 Data Calibration

THIS CHAPTER IS UNDER CONSTRUCTION

4.1 Calibration files

4.2 Loopback

5 MATAA tools reference

This section contains a list of the MATAA tools and their usage information as of 03-Jan-2016.

5.1 mataa_audio_guess_latency

file: ...mataa_tools/mataa_audio_guess_latency.m

function latency = mataa_audio_guess_latency (fs,maxLatency);

DESCRIPTION:
This function measures the latency of the audio hardware at sampling frequency fs, including the connected DUT.

The latency is defined as follows:
t1: the time needed by the audio output device to process the signal
t2: the time needed by the signal to travel from the audio output to the audio input of the computer (this will be determined by the analytical setup. In case of loudspeaker analysis, t2 will be deteremined mainly by the distance between microphone and loudspeaker).
t3: the time needed by the audio input device to process the signal

Then: latency = t1 + t2 + t3

INPUT:
fs: sampling frequency to be used for audio I/O (in seconds)
maxLatency (optional): the expected maximum of the latency (in seconds). If not specified, the user will be asked to supply a value.

OUTPUT:
latency: the latency of the system, as defined above (in seconds)

5.2 mataa_audio_info

file: ...mataa_tools/mataa_audio_info.m

function audioInfo = mataa_audio_info;

DESCRIPTION:
This function returns a struct (audioInfo) containing information on the default devices for audio input and output. Note: the list of supported sample rates reflects the 'standard' rates offered by the operating system. This is not necessarily identical to the rates supported by hardware itself, as the operating system may provide other rates, e.g. by (automatic) sample-rate conversion (such as in the case of Mac OS X / CoreAudio). Also, the list of supported sample rates may be incomplete, because the TestDevices programs checks

for 'standard' rates only. It may therefore be possible to use other sample rates than those returned from this function (check the description of your audio hardware if you need to know the rates supported by the hardware). This function checks for full and half duplex operation (i.e. if the input and output devices are the same), and returns the list of supported sample rates depending on full or half duplex operation (they may be different, e.g. if a high sampling rate is only available with half duplex due to limits in the data transfer rates).

EXAMPLE:
(get some information on the audio hardware):
> info = mataa_audio_info;
> info.input % shows information about the input device
> info.output % shows information about the output device

5.3 mataa_computer

file: ...mataa_tools/mataa_computer.m

function platform = mataa_computer;

DESCRIPTION:
Returns the current computer platform.

INPUT:
(none)

OUTPUT
platform: string indicating the computer platform:
MAC: Mac OS X (Darwin)
PCWIN: MS Windows
LINUX_X86-32: Linux on x86 / 32 Bit platform
LINUX_X86-64: Linux on AMD / 64 Bit platform
UNKNOWN: unknown platform (unknown to MATAA)

5.4 mataa_convolve

file: ...mataa_tools/mataa_convolve.m

function z = mataa_convolve(x,y);

DESCRIPTION:
This function convolves two data series x and y. The convolution is done using the fourier-transform method. x and y should have the same length (pad zeroes, if necessary). The

result of the convolution (z) will also be of the same length as x and y.

see also http://rkb.home.cern.ch/rkb/AN16pp/node38.html

EXAMPLE:
T = 1; fs = 44100; f0 = 10;
t = [1/fs:1/fs:T];
x = sin(2*pi*f0*t);
y = zeros (size(x));
y(1000) = -1.5;
z = mataa_convolve (x,y);
plot (t,x,'r',t,y,'k',t,z,'b')

5.5 mataa_deConvolve

file: ...mataa_tools/mataa_deConvolve.m

function [y] = mataa_deConvolve(z,x);

DESCRIPTION:
This function deconvolves z from x. In other words: if z = x*y ('z is the convolution of x and y'), then this function calculates y from z and x. The deconvolution is done using the fourier-transform method. z and x should have the same length (pad zeroes, if necessary).

see also http://rkb.home.cern.ch/rkb/AN16pp/node38.html

Example (calculate impulse response of a loudspeaker or other DUT):
x: the input signal sent to the speaker (known), length(x) = Lx
y: the impulse response of the speaker (not known), length(y) = Ly
z: the measured response of the speaker to signal x (known), length(z) = Lz
then: z = x*y
note: Lz = Lx + Ly -1

then: Z = XY (where the uppercase letters denote the complex fourier transforms of x, y, and z)
or: fft(z) = fft(x) fft(y), where x and y are padded with zeros to length Lz
hence fft(y) = fft(z) / fft(x), or y = ifft(fft(z) / fft(x))

5.6 mataa_export_FRD

file: ...mataa_tools/mataa_export_FRD.m

function mataa_export_FRD (f,mag,phase,comment,file);

DESCRIPTION:
Export frequency-domain data to a FRD file.
(see also http://www.pvconsultants.com/audio/frdis.htm)
An FRD file is essentially an ASCII file containing three columns of data: frequency, magnitude, and phase. A detailed description of the FRD file format is given below.

INPUT:
f: frequency values (Hz)
mag: magnitude values (usually in dB)
phase: phase (in degrees, usually wrapped to the range -180...+180 degrees)
file: string containing the name of the file to be written (may contain a complete path. If no path is given, the file will be written to the current working directory)
comment: string containing a comment to be saved with the data, e.g. a description of the data. Use comment = " if you do not want a comment in the data file.

OUTPUT:
(none)

DESCRIPTION OF THE FRD FILE FORMAT
The following is a detailed description of the FRD format (taken from the website given above):

What is an FRD File?

A Frequency Response Data file is a human readable text file that contains a numerical description of Frequency and Phase Response. The purpose of an FRD file to represent measurements or targets or corrections of acoustic items, like loudspeakers and/or crossovers or room effects. The reason for using FRD files is to pass information between different design programs and thus to get the programs to share data and work together to achieve a complete finished design.

Structurally, an FRD file is very simple. An * is placed in the first character position of any line that is a comment, so the remainder of that line is ignored. Comments can only be added at the beginning of an FRD file and not embedded once the data starts.

After the comment, the data block is composed of three numerical values per line separated by either one or more spaces or a tab. Each line is a single measurement or value instance. The numerical values, in order, per line, correspond to Frequency, Magnitude and Phase. The frequency data should start at the low end of the response and proceed to the higher end with no directional reversals or overlapping repeating regions in the frequency progression. That is all. It should look something like this:

* Seas T25-001.frd
* Freq(Hz) SPL(db) Phase(deg)

```
*
10 21.0963 158.4356
10.1517 21.0967 158.4363
10.3056 21.3305 158.7836
10.4619 21.5644 159.1299
10.6205 21.7983 159.2452
10.7816 22.032 159.3599
10.9451 22.2658 159.4099
11.1111 22.4996 159.4597
11.2796 22.7335 159.4832
11.4507 22.9672 159.5065
11.6243 23.2011 159.5171
11.8006 23.4349 159.5276
11.9795 23.6687 159.5308
12.1612 23.9025 159.534
```

The comment field mentioned above is sometimes required, even if the data in it is never used, or at least we have encountered programs that will not load the FRD file if the Comment field is not there. We have also found the opposite, programs that get confused about the comment field and work better if there was none. In general the comments are useful to the human reader and specific to the last program to output the data. So box modelers may have the conditions used to create the curve, like Vb, Driver name and T/S parameters, etc.

It is usually better that the data blocks have boundaries on the numbers used. Although Scientific Notation is permitted, it is usually better, more accurate and much more readable if the numbers used have exactly four decimal places below the dot (greater accuracy is really not helpful and less has been show to induce jitter from Group Delay derived or other secondary processing). In addition, it greatly simplified the operation of any subsequent program if the Frequency spacing is even and progresses in a log spacing format. This tends to spread the samples evenly over the frequency segment.

The Magnitude number is log gain and in db values. The scale can be SPL wattage distance format (hovering about 90) or a unity aligned offset (usually just above zero for diffraction or starting at and diving below zero steeply for box models and crossover functions). The Phase data is best if in degrees, from –180 to +180 wrapping.

In general, there are good reasons to keep the frequency sampling density high enough to accurately represent a complex waveform sequence (without losing detail) but not so dense as to generate large amounts of extra sample data. Usually between 200 to 250 samples per decade, which is about 60 to 75 samples per octave, works very well.

When processing files and using the resultants, there are also good reasons to have the response extend at least one octave and preferably 2 or more octaves beyond the region of interest (above and below) so as to keep phase tracking error very low. This is especially important when deriving Minimum Phase or Optimizing crossovers downstream. A good standard to target is the internal default one of the Frequency Response Combiner program, which was selected for those reasons above (sample density and frequency extension) and for a close adherence to digital sound cards sampling rates, and also that the sample set was easily sub-divided into many equal sized integer count pieces (2, 3, 4, 6, 7, 8, 14, 16, 21, 24). The FRC program default standard for internal FRD data calculation is 2 Hz to 96,000 Hz with 1176 equal log spaced samples or about 251 samples per decade.

5.7 mataa_export_TMD

file: ...mataa_tools/mataa_export_TMD.m

function mataa_export_TMD (t,s,comment,file);

DESCRIPTION:
Export time-domain data to a TMD file (or, in other words: export the samples a signal s(t) to an ASCII file). A TMD file is essentially an ASCII file containing two columns of data: time and signal samples. The 'TMD format' is modelled after the FRD format for frequency-domain data (see mataa_export_FRD for more information).

INPUT:
t: time values (seconds)
s: signal samples
comment: string containing a comment to be saved with the data, e.g. a description of the data. Use comment = '' if you do not want a comment in the data file.

OUTPUT:
(none)

5.8 mataa_file_default_name

file: ...mataa_tools/mataa_file_default_name.m

function name = mataa_file_default_name;

DESCRIPTION:
This function returns a file name that can be used to save MATAA data. If 'ask' is nonzero, the user is asked to enter a file name. If no answer is given or if 'ask' is zero, a default file name made up of the current date and time of day is returned.

INPUT:
ask: flag to specify if the user should be asked for a file name. If 'ask' is not specified, ask=0 is assumed.

OUTPUT:
name: file name

5.9 mataa_FR_extend_LF

file: ...mataa_tools/mataa_FR_extend_LF.m

function [mag,phase,f] = mataa_FR_extend_LF (fh,mh,ph,fl,ml,pl,f1,f2);;

DESCRIPTION:
Extend frequency response (e.g. from an anechoic analysis of a loudspeaker impulse response measured in the far field) with low-frequency data (e.g. from a near-field measurement). The frequency ranges of the two frequency responses need to overlap, and the common data in the frequency range [f1,f2] is used to determine the offsets in the magnitude and phase of the two frequency-response data sets. The low-frequency magnitude and phase (ml, pl) is adjusted to fit the high-frequency data (mh, ph). The phase data (ph, pl) may either be wrapped (e.g. to a range of -180..+180 deg) or unwrapped. After adjusting the relative offsets, the resulting response in the overlap band is computed as the weighted mean of the low and high frequency data, where the weight of the high-frequency data increases linearly from 0 at f1 to 1 at f2 (and vice versa for the low-frequency data).

INPUT:
mh, ph, fh: magnitude (dB), phase (deg.) and frequency (Hz) data of the frequency response covering the high-frequency range
ml, pl, fl: magnitude (dB), phase (deg.) and frequency (Hz) data of the frequency response covering the low-frequency range
f1, f2: [f1,f2] is the frequency range used to determine the offsets of the low-frequency magnitude and phase (ml, pl) relative to the high-frequency data (mh, ph).

OUTPUT:
mag, phase, f: magnitude (dB), phase (deg, unwrapped) and frequency (Hz) of the combined frequency response. The data with f > f2 are identical to (mh,ph,fh), those with f < f1 correspond to (ml,pl,fl) with the magnitude and phase offsets removed. The data in the range [f1,f2] corresponds to the combination of the data of both data sets.

5.10 mataa_FR_smooth

file: ...mataa_tools/mataa_FR_smooth.m

function [mag,phase,f] = mataa_FR_smooth (mag,phase,f,smooth_interval);

DESCRIPTION:
Smooth frequency response in octave bands.

INPUT:
mag: magnitude data
phase: phase data
f: frequency
smooth_interval: width of octave band used for smoothing

OUTPUT:
mag: smoothed frequency response (magnitude)
phase: smoothed frequency response (phase)
f: frequency values of smoothed frequency response data

EXAMPLE:
> [h,t] = mataa_IR_demo;
> [mag,phase,f] = mataa_IR_to_FR(h,t); % calculates magnitude(f) and phase(f)
> [magS,phaseS,fS] = mataa_FR_smooth(mag,phase,f,1/4); % smooth to 1/4 octave resolution
> semilogx (f,mag , fS,magS); % plot raw and smoothed data

fractional octave between last and second-last data point:

5.11 mataa_f_to_t

file: ...mataa_tools/mataa_f_to_t.m

function t = mataa_f_to_t (f);

DESCRIPTION:
returns the time bins of the inverse fourier spectrum sampled at frequencies f (f is assumed to be evenly spaced!)

INPUT:
f: frequency-value vector (in Hz). Values must be sorted and evenly spaced.

OUTPUT:
t: time values (vector, in seconds)

5.12 mataa_gnuplot

file: ...mataa_tools/mataa_gnuplot.m

function mataa_gnuplot (cmd);

DESCRIPTION:
This function executes the gnuplot command 'cmd' by calling __gnuplot_raw__(cmd). This

only makes sense with Octave if gnuplot is used as the plotting engine. IMPORTANT: THIS FUNCTION SHOULD NOT BE USED ANYMORE, BECAUSE THE GNUPLOT INTERFACE TO OCTAVE HAS CHANGED CONSIDERABLY IN OCTAVE 2.9.X. IT WILL PROPABLY BE CHANGED FURTHER, BREAKING THIS FUNCTION.

INPUT:
cmd: string containing the gnuplot command.

5.13 mataa_guess_IR_start

file: ...mataa_tools/mataa_guess_IR_start.m

function [t_start,t_rise] = mataa_guess_IR_start (h,t,fc,verbose);

DESCRIPTION:
Try to determine the start and and rise time of an impulse response signal.

Note: this function calculates the analytic signal to determine the envelope function of h(t), and then analyses the envolope curve to find t_start and t_rise. See, for instance: http://en.wikipedia.org/wiki/Analytic_signal .

INPUT:
h: impulse response
t: time-values vector of impulse response samples (vector, in seconds), or, alternatively, the sampling frequency of h(t) (scalar, in Hz, the first sample in h is assumed to correspond to time t(1)=0).
fc (optional): cut-off frequency of high pass filter applied to h(t) before finding the impulse. This is useful if h(t) is masked by low-frequency noise. If fc is not empty, a 4th order Butterworth high-pass filter will be applied to h(t) to remove low-frequency noise.
verbose (optional): if verbose=0, no user feedback is given. If not specified, verbose ~= 0 is assumed.

OUTPUT:
t_start: 'beginning' of h(t) (seconds)
t_rise: rise time of h(t) (seconds)

EXAMPLE:
> [h,t] = mataa_IR_demo; % load demo data of an loudspeaker impulse response.
> mataa_plot_IR(h,t); % plot the fake signal
> [t_start,t_rise] = mataa_guess_IR_start(h,t,20)

This gives t_start = 0.288 ms and t_rise = 0.0694 ms. In this example might therefore safely discard all data with t < t_start. In real-world use (with noise and Murphy's law against us), however, it might be worthwile to add some safety margin, e.g. using t_rise: discard all data with t < t_start - t_rise.

5.14 mataa_hilbert

file: ...mataa_tools/mataa_hilbert.m

function y = mataa_hilbert (x)

DESCRIPTION:
Calculates the Hilbert transform of x.

This code was modelled after the Hilbert transform function 'hilbert.m' available from Octave-Forge

INPUT:
x: input signal (column vector). If x contains complex values, only the real part of these values will be used.

OUTPUT:
y: hilbert transform of x

5.15 mataa_impedance_fit_speaker

file: ...mataa_tools/mataa_impedance_fit_speaker.m

function [Rdc,f0,Qe,Qm,L1,L2,R2] = mataa_impedance_fit_speaker (f,mag,phase);

DESCRIPTION:
Fits the impedance model of mataa_impedance_speaker_model to the impedance data mag(f) and phase(f). This can be useful in determining Thielle/Small parameters from impedance measurements.

INPUT:
f: frequency values of the impedance data
mag: magnitude of impedance data (Ohm)
phase: phase of impedance data (degrees)

OUTPUT:
Rdc, f0, Qe, Qm, L1, L2, R2: see mataa_impedance_speaker_model (input parameters)

5.16 mataa_impedance_speaker_model

file: ...mataa_tools/mataa_impedance_speaker_model.m

function [mag,phase] = mataa_impedance_speaker_model (f,Rdc,f0,Qe,Qm,L1,L2,R2)

DESCRIPTION:
Calculate speaker impedance (magnitude and phase) as a function of frequency f according to the MLSSA model (see Figure 7.16 in J. d'Appolito, "Testing Loudspeakers", Audio Amateur Press). This model essentially consists of a combination of three impedance elements connected in series (where w = 2*pi*f, w0 = 2*pi*f0):
(a) The DC resistance of the voice coil (Rdc)
(b) A parallel LCR circuit, reflecting the the low-frequency part of the impedance curve (resonance peak).
(c) L1 in series with a parallel combination of R2 and L2. L1, L2, and R2 reflect the high-frequency part of the impedance curve. For L2 = 0 and R2 = Inf, this model reduces to the simpler concept where the voice-coil inductance Le is constant with frequency (and L1 = Le).

INPUT:
f: frequency values for which impedance will be calculatedq
Rdc: DC resistance of the voice coil (Ohm)
f0: resonance frequency of the speaker (Hz)
Qe: electrical quality factor of the speaker (at resonance)
Qm: mechanical quality factor of the speaker (at resonance)
L1, L2, R2 (optional): see above (in H or Ohm, respectively)

OUTPUT:
mag: magnitude of impedance (Ohm)
phase: phase of impedance (degrees)

NOTES:
- The ratio Qm/Qe reflects the height of the impedance peak. If Zmax is the impedance maximum (at resonance) then Zmax/Rdc = Qm/Qe-1.
- Qe reflects the width of the impedance peak (at least I think so; large Qe corresponds to a narrow peak)

EXAMPLE:
The following gives a good approximation of the data shown in Fig. 7.18 in J. d'Appolito, "Testing oudspeaker" on page 122:
f = logspace(1,4,100);
[mag,phase] = mataa_impedance_speaker_model (f,7.66,33.22,0.45,3.4,0.4e-3,1.1e-3,13);
semilogx (f,mag,f,phase)

5.17 mataa_import_AIFF

file: ...mataa_tools/mataa_import_AIFF.m

function [t,s] = mataa_import_AIFF (file)

DESCRIPTION:
Import time-domain data from an AIFF file. This function requires the sndfile-convert utiliy, which is part of libsndfile (http://www.mega-nerd.com/libsndfile).

INPUT:
file: string containing the name of the file containing the data to be imported. The string may contain a complete path. If no path is given, the file is assumed to be located in the current working directory.

OUTPUT:
t: time values (s)
s: signal samples

5.18 mataa_import_FRD

file: ...mataa_tools/mataa_import_FRD.m

function [f,mag,phase,comments] = mataa_import_FRD (file);

DESCRIPTION:
Import frequency-domain data from a FRD file.
(see also mataa_export_FRD).

INPUT:
file: string containing the name of the file containing the data to be imported. The string may contain a complete path. If no path is given, the file is assumed to be located in the current working directory.

OUTPUT:
f: frequency values (Hz)
mag: magnitude values
phase: phase
comments: cell string containing the comments in the data file (if any)

HISTORY:
9. January 2008 (Matthias Brennwald): first version

5.19 mataa_import_mlssa

file: ...mataa_tools/mataa_import_mlssa.m

function [mlsvec,mlsfs,stimulus_amp,mlsdf] = mataa_import_mlssa (File,Outfile,Withir);

Reads a MLSSA .TIM or .FRQ file and extracts all data from it. Note that this function has been designed using Matlab only (i.e. it might not work as well with Octave).

INPUT:
File (optional): should contain the filename, including path and extension (.TIM or .FRQ). If File is empty, a file dialog is presented.
Outfile: should contain a filename, including path but no extension (will be given.mat). The output data will be saved in this file.
Withir (optional): parameter, should be included and with the text 'Withir' if the impulse response (or transfer function) mlsvec should be included in the Output file.

OUTPUT:
mlsvec the impulse response (for .TIM files) or the transfer function (for .FRQ files; containing nfft/2 + 1 complex values).
mlsfs the sampling frequency
stimulus_amp the stimulus amplitude used during the measurement
mlsdf the frequency increment (only for .FRQ files)

Comment 1: Note that an MLS file (.TIM or .FRQ) is half the size of the corresponding Matlab file (MLSSA uses single precision whereas Matlab uses double precision). Thus the MLS files can be used and opened every time data is needed, instead of creating a Matlab copy of the file.

Comment 2: The output parameter stimulus_amp might be needed to scale the impulse response correctly. MLSSA does not scale the impulse versus the stimulus_amp so that if different stimulus_amp have been used, the corresponding impulse responses will display different amplitudes. The transfer functions (.FRQ) are however scaled correctly.

Comment 3: The impulse response can be retrieved from the transfer function by inserting the values for negative frequencies:
[mlsvec,mlsfs,stimulus_amp,mlsdf] = readmls('TEST.FRQ',Outfile);
npoints = length(mlsvec);
mlsvec = [mlsvec; conj(mlsvec(npoints-1:-1:2))];
ir = real(ifft(mlsvec)); % ir should be a real quantity. Any remaining
% imaginary values will reflect numerical errors
% or an incorrect transfer function.
Note however that if a window was used before calculating the transfer function the windowed impulse response will be extracted.

Comment 4: The MLSSA files contain a large number of auxilliary parameters that are saved in
the Outfile. Refer to the appendix of the MLSSA manual for information about these
parameters, which are those in the setup of the MLSSA measurements. According to
the manual, this setup structure can be changed in future versions. This one is
valid for version 9.0.

The program is based on code written by Peter Svensson (svensson[at]iet.ntnu.no) available at http://www.iet.ntnu.no/~svensson/readmls.m. Peter Svensson explicitly agreed to provide his work for inclusion in MATAA.

5.20 mataa_import_TMD

file: ...mataa_tools/mataa_import_TMD.m

function [t,s,comments] = mataa_import_TMD (file,timefix)

DESCRIPTION:
Import time-domain data from a TMD file (see also mataa_export_TMD).

INPUT:
file: string containing the name of the file containing the data to be imported. The string may contain a complete path. If no path is given, the file is assumed to be located in the current working directory.
timefix (optional): flag indicating if (and how) mataa_import_TMD should try to make time values evenly spaced. If timefix > 1: t = timefix * round (1/mean(diff(t))/timefix)

OUTPUT:
t: time values (s)
s: signal samples
comments: cell string containing the comments in the data file (if any)

EXAMPLE:

> [t,h,comments] = mataa_import_TMD ('scanspeaker_0deg_no_filter_tweeter.tmd',10);

5.21 mataa_interp

file: ...mataa_tools/mataa_interp.m

function y = mataa_interp (xi,yi,x);

DESCRIPTION:
Linear interpolation of y(x) from yi(xi)

if x is outside the range of xi, mataa_interp returns a linear extrapolation of the yi

Linear interpolation is of course available in Matlab and Octave-Forge as interp1. However, it's not available in plain-vanilla Octave, which is a shame, I think (this was fixed a while ago, so mataa_interp is obsolete and may be removed in the future). I therefore provided this function for MATAA so that I don't have to worry about interp1 missing in Octave while still being able to easily write code that is compatible with both Matlab and Octave.

FIXME: THIS CODE IS AS INEFFICIENT AS IT GETS!

5.22 mataa_IR_demo

file: ...mataa_tools/mataa_IR_demo.m

function [h,t] = mataa_IR_demo (IRtype)

DESCRIPTION:
This function returns the an impulse response h(t), specified by 'IRtype'.

INPUT:
type (optional): string describing the type of impulse response (see below). If not specified, type = 'DEFAULT' is used.

valid choices for 'IRtype':

FE108: impulse response of a Fostex FE108Sigma full-range driver, sampled at a rate of 96 kHz.

DIRAC: dirac impulse (first sample is 1, all others are zero), with a length of 1 second, sampled at 44.1 kHz.

EXP: exponential decay (f(t) = exp(-t/tau), with tau=1E-2 seconds), with a length of 1 second, sampled at 44.1 kHz.

DEFAULT: same as 'FE108'.

OUTPUT:
h: impulse response samples
t: time coordinates of samples

5.23 mataa_IR_remove_echo

file: ...mataa_tools/mataa_IR_remove_echo.m function [h,t] = mataa_IR_remove_echo (h,t,t_echo_start,t_echo_end);

DESCRIPTION:
This function removes echos from an impulse response. The echos are replaced by data calculated by linear interpolation.

INPUT:
h: values impulse response (vector)
t: time values of samples in h (vector)
t_echo_start: start time of echo
t_echo_end: end time of echo

OUTPUT:
h: values impulse response with echo removed
t: time values of samples in h

5.24 mataa_IR_to_CSD

file: ...mataa_tools/mataa_IR_to_CSD.m

function [spl,f,t] = mataa_IR_to_CSD (h,t,T,smooth_interval);

DESCRIPTION:
This function calculates cumulative spectral decay (CSD) data (SPL-responses spl at frequencies f and delay times d).

INPUT:
h: values impulse response (vector)
t: time values of samples in h (vector, in seconds) or sampling rate of h (scalar, in Hz)
T: desired delay times (should be evenly spaced)
smooth_interval (optional): if supplied, the SPL curves are smoothed using mataa_IR_to_FR_smooth

OUTPUT:
spl: CSD data (dB)
f: frequency (Hz)
d: delay of CSD data (seconds)

EXAMPLE:
[h,t] = mataa_IR_demo ('FE108');
T = [0:1E-4:4E-3];
[spl,f,t] = mataa_IR_to_CSD (h,t,T,1/24);
mataa_plot_CSDt (spl,f,t,50);

5.25 mataa_IR_to_ETC

file: ...mataa_tools/mataa_IR_to_ETC.m

function [etc,t] = mataa_IR_to_ETC (h,t);

DESCRIPTION:
This function calculates the energy-time-curve (ETC) from the impulse response h(t).
The ETC is the envelope (magnitude) of the analytic signal of h (see D'Appolito, J.: Testing Loudspeakers, p. 125)

INPUT:
h: impulse response (in volts)
t: time coordinates of samples in h (vector, in seconds) or sampling rate of h (scalar, in Hz)

OUTPUT:
etc: energy-time curve
t: time coordinates of etc (in seconds)

EXAMPLE:
> [h,t] = mataa_IR_demo;
> [etc,t] = mataa_IR_to_ETC(h,t);
> mataa_plot_ETC_lin(etc,t)

5.26 mataa_IR_to_FR

file: ...mataa_tools/mataa_IR_to_FR.m

function [mag,phase,f] = mataa_IR_to_FR (h,t,smooth_interval,unit);

DESCRIPTION:
Calculate frequency response (magnitude in dB and phase in degrees) of a system with impulse response h(t)

INPUT:
h: impulse response (in volts)
t: time coordinates of samples in h (vector, in seconds) or sampling rate of h (scalar, in Hz)
smooth_interval (optional): if specified, the frequency response is smoothed over the octave interval smooth_interval.
unit (optional): unit of h.

OUTPUT:
mag: magnitude of frequency response (in dB). If unit of h is 'Pa' (Pascal), then mag is referenced to 20 microPa (standard reference sound pressure level).
phase: phase of frequency response (in degrees). This is the TOTAL phase including the

'excess phase' due to (possible) time delay of h(h). phase is unwrapped (i.e. it is not limited to +/-180 degrees, and there are no discontinuities at +/- 180 deg.)
f: frequency coordinates of mag and phase

EXAMPLE:
> [h,t] = mataa_IR_demo;
> [mag,phase,f] = mataa_IR_to_FR(h,t); % calculates magnitude(f) and phase(f)
> [mag,phase,f] = mataa_IR_to_FR(h,t,1/8); % same as above, but smoothed to 1/8 octave resolution
(use mataa_plot_FR(mag,phase,f) to plot the results)

5.27 mataa_IR_to_SR

file: ...mataa_tools/mataa_IR_to_SR.m

function [s,t] = mataa_IR_to_SR (h,t);

DESCRIPTION:
calculates the step response of a system with impulse response h(t)

INPUT:
h: impulse response (in volts)
t: time coordinates of samples in h (vector, in seconds) or sampling rate of h (scalar, in Hz)

OUTPUT:
s: step response
t: time (seconds)

5.28 mataa_IR_to_TBES

file: ...mataa_tools/mataa_IR_to_TBES.m

function [A,tau,f] = mataa_IR_to_TBES (h,t,f);

DESCRIPTION:
Calculate tone burst energy storage (TBES) data. The impulse response is convolved with shaped tone burst(s) to analyze the transient response and energy storage of the DUT at different frequencies. Tone burst signals used are 4 cycles of pure sine with a Blackman envelope.
The method is based on the ideas of Siegfried Linkwitz (see http://www.linkwitzlab.com/frontiers_2.htm#M
) and Jochen Fabricius.

INPUT:
h: values impulse response (vector)

t: time values of samples in h (vector, in seconds) or sampling rate of h (scalar, in Hz)
f: frequency value(s) of tone burst (Hz)

OUTPUT:
A: amplitude envelope (dB, relative to max value)
tau: dimensionless time value (time normalized by period of burst frequency)
f: frequency values (same values as in input, useful for plotting TBES results)

EXAMPLE:
> [h,t] = mataa_IR_demo ('FE108');
> f = logspace (2,4,50);
> [A,tau,f] = mataa_IR_to_TBES (h,t,f);

5.29 mataa_load_calibration

file: ...mataa_tools/mataa_load_calibration.m

function cal = mataa_load_calibration (calfile)

DESCRIPTION:
Load calibration data for test devices from calibration file.

INPUT:
calfile: name of calibration file (e.g., "Behringer_ECM8000.txt")

OUTPUT:
cal: struct with calibration data.

EXAMPLE:
To load the (generic) calibration data for a Behringer ECM8000 microphone:
c = mataa_load_calibration ('BEHRINGER_ECM8000_D1303397118_MICROPHONE.txt');

5.30 mataa_measure_HD

file: ...mataa_tools/mataa_measure_HD.m

function [THD,kn,l0DUT,l0REF] = mataa_measure_HD (f1,T,fs,N);

DESCRIPTION:
This function measures harmonic distortion using a sine wave with a given frequency.

INPUT:
f1: base frequency in Hz.

T: sine-signal length in seconds.
fs: sampling frequency in Hz
N (optional): number of harmonics to be analyzed. By default, N=12 is assumed.

OUTPUT:
THD = total harmonic distortion, see below.
kn: harmonic distortion spectrum, in voltage units (not power). kn is a vector containing the harmonic components (k1, k2, k3, ... kN), where k1 corresponds to f1. The spectrum is normalised such that k1 is equal to one.
f1: true value of f1 used for analyses (value may be adjusted slightly to fit in the resolution of the fourier spectrum).
l0DUT, l0REF: RMS level of the DUT and REF signals at the soundcard input (useful for repeated tests at different levels)

NOTE 1: THD is computed WITHOUT the noise in the spectrum ranges between the harmoics.

NOTE 2: There exist different definitions of THD (see e.g. http://en.wikipedia.org/wiki/THD and the external links cited there for some of these definitions). Here, the following definition is used:
THD = sqrt(k2^2 + k3^2 + ... + kN^2) / k1

NOTE 3: THD is returned in relative units, not percentage or dB. For instance, THD = 0.02 corresponds to 2% THD.

NOTE 4: Only the harmonic components up to kN are analysed. Signal components in between the harmonic components (noise, hum, etc.) are NOT included in THD. The result is therefore NOT THD + noise !

EXAMPLE:
> [thd,k] = mataa_measure_HD(1000,1,96000); % measure THD and harmonic power distortion spectrum for a base-frequency of 1 kHz.
> mataa_plot_HD(k,'f1: 1kHz'); % plot the distortion spectrum

5.31 mataa_measure_impedance

file: ...mataa_tools/mataa_measure_impedance.m

function [Zmag,Zphase,f] = mataa_measure_impedance (fLow,fHigh,R,fs,resolution);

DESCRIPTION:
Measures the complex, frequency-dependent impedance Z(f) in the frequency range [fLow,fHigh].
The measurement relies on the setup described in the MATAA manual.

INPUT:
fLow: lower limit of the frequency range (Hz)
fHigh: upper limit of the frequency range (Hz)
R: resistance of the reference resistor (Ohm)
fs (optional): sampling frequency to be used for sound I/O. If not value is given, the lowest possible sampling frequency will be used.
resolution (optional): frequency resolution in octaves (example: resolution = 1/24 will give 1/24 octave smoothing). Default is resolution = 1/48. If you want no smoothing at all, use resolution = 0.

OUTPUT:
Zabs: impedance magnitude (Ohm)
Zphase: impedance phase (degrees)
f: vector of frequency values

5.32 mataa_measure_IR

file: ...mataa_tools/mataa_measure_IR.m

function [h,t,unit] = mataa_measure_IR (input_signal,fs,N,latency,loopback,cal);

DESCRIPTION:
This function measures the impulse response h(t) of a system using sample rate fs. The sampling rate must be supported by the audio device and by the TestTone program. See also mataa_measure_signal_response. h(t) is determined from the deconvolution of the DUT's response and the original input signal (if no loopback is used) or the REF channel (with loopback). The allocation of the DUT (and REF) channel is determined using mataa_settings ('channel_DUT') (and mataa_settings ('channel_REF')).

INPUT:
input_signal: input signal, vector of signal samples or name to file with sample data. Files must be in ASCII format and contain a one-column vector of the signal samples, where +1.0 is the maximum and -1.0 is the minimum value. The file should be in the 'test_signals' path. NOTE: it can't hurt to have some zeros padded to the beginning and the end of the input_signal. This helps to avoid that the DUT's response is cut off due to the latency of the audio hardware (and possibly the 'flight time' of the sound from a loudspeaker to a microphone).
N (optional): the impulse response is measured N times and the mean response is calculated from these measurements. N = 1 is used by default.
latency: see mataa_measure_signal_response
loopback (optional): flag to control the behaviour of deconvolution of the DUT and REF channels. If loopback = 0, the DUT signal is not deconvolved from the REF signal (no loopback calibration). Otherwise, the DUT signal is deconvolved from the REF channel. The allocation of the DUT and REF channels is taken from mataa_settings('channel_DUT') and mataa_settings('channel_REF'). Default value (if not specified) is loopback = 0.

cal (optional): calibration data (struct or (cell-)string, see mataa_load_calibration and mataa_signal_calibrate)

OUTPUT:
h: impulse response
t: time
unit: unit of data in h

EXAMPLES:

A. Measure impulse response using a sweep test signal (without any data calibration):
> s = mataa_signal_generator ('sweep',44100,1,[50 20000]); % create test signal (sine sweep from 50 to 20000 Hz, 1 s long, with 44.1 kHz sampling frequency
> [h,t] = mataa_measure_IR (s,44100,1,0.1); % measure impulse response using test signal s, allowing for 0.1 s latency of sound in/out
> plot (t,h) % plot result

B. same, with calibration using a looback connection on second channel:
> s = mataa_signal_generator ('sweep',44100,1,[50 20000]);
> [h,t] = mataa_measure_IR (s,44100,1,0.1,1); % with loopback deconvolution
> plot (t,h)

5.33 mataa_measure_IR_old_noLoopback

file: ...mataa_tools/mataa_measure_IR_old_noLoopback.m

function [h,t,unit] = mataa_measure_IR (input_signal,fs,N,latency,cal);

DESCRIPTION:
This function measures the impulse response h(t) of a system using sample rate fs. The sampling rate must be supported by the audio device and by the TestTone program. See also mataa_measure_signal_response. h(t) is determined from the deconvolution of the DUT's response and the original input signal.

INPUT:
input_signal: input signal, vector of signal samples or name to file with sample data. Files must be in ASCII format and contain a one-column vector of the signal samples, where +1.0 is the maximum and -1.0 is the minimum value. The file should be in the 'test_signals' path. NOTE: it can't hurt to have some zeros padded to the beginning and the end of the input_signal. This helps to avoid that the DUT's response is cut off due to the latency of the audio hardware (and possibly the 'flight time' of the sound from a loudspeaker to a microphone).
N (optional): the impulse response is measured N times and the mean response is calculated from these measurements. N = 1 is used by default.
latency: see mataa_measure_signal_response

cal (optional): calibration data (struct or string, see mataa_load_calibration and mataa_signal_calibrate)

OUTPUT:

h: impulse response
t: time
unit: unit of data in h

5.34 mataa_measure_signal_response

file: ...mataa_tools/mataa_measure_signal_response.m

function [responseSignal,inputSignal,t,unit] = mataa_measure_signal_response (input_signal,fs,latency,verbose,channels,cal);

DESCRIPTION:
This function feeds one or more test signal(s) to the DUT(s) and records the response signal(s).

INPUT:
input_signal: this is either a matrix containing the samples of the test signal, or a string containing the name of a TestTone file containing the test signal. See mataa_signal_to_TestToneFile for the format of the matrix containing the test signal samples. If a data file is given as the input, and if the file name is given without the full path of the file, the file is assumed to reside in the MATAA signals-path (you can retrieve the signals path with the command mataa_path('signals')).

fs: the sampling rate to be used for the audio input / output (in Hz). Only sample rates supported by the hardware (or its driver software) are supported.

latency: if the signal samples were specified rather than a file name/path, the signal is padded with zeros at its beginning and end to avoid cutting off the test signals early due to the latency of the sound input/output device(s). 'latency' is the length of the zero signals padded to the beginning and the end of the test signal (in seconds). If a file name is specified instead of the signal samples, the value of 'latency' is ignored.

verbose (optional): If verbose=0, no information or feedback is displayed. Otherwise, mataa_measure_signal_response prints feedback on the progress of the sound in/out. If verbose is not specified, verbose ~= 0 is assumed.

channels (optional): index to data channels obtained from the ADC that should be processed and returned. If not specified, all data channels are returned.

cal (optional): calibration data for use with mataa_signal_calibrate (see mataa_signal_calibrate for details). If different audio channels are used with different hardware (e.g., a microphone in the DUT channel and a loopback without microphone in the REF channel), separate structs describing the hardware of each channel can be provided in a cell array.

OUTPUT:
inputSignal: matrix containing the input signal(s). This may be handy if the original test-signal data are stored in a file, which would otherwise have to be loaded into into workspace to be used.

responseSignal: matrix containing the signal(s) from the audio input device. This will contain the data from all channels used for signal recording, where each matrix colum corresponds to one channel.

t is vector containing the times corresponding the samples in responseSignal and inputSignal (in seconds)

unit: unit of data in responseSignal. If the signal has more than one channel, signal_unit is a cell string with each cell reflecting the units of each signal channel.

FURTHER INFORMATION:
The signal samples range from -1.0 to +1.0).
The TestTone program feeds the input_signal to both stereo channels of the output device, and records from both stereo channels of the input device (assuming we have a stereo device). Therefore, the response signal has two channels. As an example, channel 1 is used for for the DUT's response signal and channel 2 can be used to automatically calibrate for the frequency response / impulse response of the audio hardware (by directly connecting the audio output to the audio input). Channel allocation can be set using mataa_settings.

EXAMPLE:
Feed a 20Hz square-wave signal to the DUT and compare the input and response signals:
> [out,in,t] = mataa_measure_signal_response('squareburst_96k_1s_20Hz.in',96000);
> plot(t,in,t,out)

5.35 mataa_menu

file: ...mataa_tools/mataa_menu.m

function out = mataa_menu (title, varargin)

DESCRIPTION:
This function prints a menu and asks the user to choose a command from the menu.

title: the tile of the menu (string)
varargin: a list of menu entries as described in the below example
out: the command chosen by the user

EXAMPLE:

To print a menu with the title 'Main menu' and the commands 'measure', 'plot', 'save' and 'exit':
choice = mataa_menu('Main menu','m','measure','p','plot','s','save','e','exit');

The result will look like this:
————
Main menu:
[m] measure – [p] plot – [s] save – [e] exit

Choose a command:
————
The user then chooses one of the four commands by entering 'm', 'p', 's' or 'e'. If he/she enteres something else, an error message will be shown, and the menu is displayed again.

5.36 mataa_minimum_phase

file: ...mataa_tools/mataa_minimum_phase.m

function min_phase = mataa_minimum_phase (mag,f);

DESCRIPTION:
Calculates minimum phase from magnitude frequency response using the Hilbert transform
(see http://en.wikipedia.org/wiki/Minimum_phase#Relationship_of_magnitude_response_to_phase_response).

INPUT:
mag: magnitude of frequency response (in dB)
f: frequency coordinates of mag (in Hz)

OUTPUT:
min_phase: minimum phase at frequnecies f (unwrapped, in degrees)

%% % calculate minimum phase using the Hilbert transform:
%% % see: http://www.fourelectronics.com/Hilbert-transform-to-calculate-Magnitude-from-Phase-10052397.html
%% % and: http://www.dsprelated.com/showmessage/29416/1.php
%% % this should use the NATURAL log, and 'abs(p)' rather than '10*abs(p)'!
convert mag from dB to natural units:

5.37 mataa_octave_version

file: ...mataa_tools/mataa_octave_version.m

function [version,subversion,subsubversion] = mataa_octave_version

DESCRIPTION:
Returns the Octave version. If called with Matlab, the output values are set to NaN.

INPUT:
(none)

OUTPUT:
version: main version
subversion: subversion
subsubversion: subsubversion

EXAMPLE:
With Octave 2.1.73, the output is:
version = 2
subversion = 1
subsubversion = 73

5.38 mataa_path

file: ...mataa_tools/mataa_path.m

function path = mataa_path (whichPath);

DESCRIPTION:
This function returns the Matlab / MATAA paths as specified by 'whichPath'

INPUT:
whichPath (optional): a string specifying which path should be retrieved.
whichPath can be one of the following:
'main' (default) the main MATAA path
'signals' the path where the test signal data is stored
'tools' the path where the MATAA 'tools' routines are stored (the MATAA toolbox)
'TestTone' the path to the TestTone program
'TestDevices' the path to the TestDevices program
'mataa_scripts' the path to the MATAA scripts
'microphone' the path to the microphone-data files
'settings' the path where the MATAA settings are stored
'calibration' the path where calibration files are stored (microphones, audio interfaces /

soundcards, etc.)

If whichPath is not specified, it is set to 'main' by default.

OUTPUT:
path: the MATAA path as indicated by whichPath (string)

5.39 mataa_phase_remove_delay

file: ...mataa_tools/mataa_phase_remove_delay.m

function [phase,f] = mataa_phase_remove_delay (phase,delay);

DESCRIPTION:
This function removes excess phase due to time delay.

INPUT:
phase: phase, including excess phase due to time delay (unwrapped, in degrees)
f: frequency coordinates of phase (in Hz)
delay: time delay to be removed from the phase (in seconds)

OUTPUT:
phase: phase with excess phase corresponding to delay removed (unwrapped, in degrees)

5.40 mataa_phase_remove_trend

file: ...mataa_tools/mataa_phase_remove_trend.m

function [phase,delay] = mataa_phase_remove_trend (phase,f,f1,f2);

DESCRIPTION:
Remove linear trend in phase(f), e.g. excess phase due to time delay.

INPUT:
phase: phase, including excess phase due to time delay (unwrapped, in degrees)
f: frequency coordinates of phase (in Hz)
f1, f2 (optional, in Hz): if both f1 and f2 are specified, the linear trend in phase(f1<f<f2) is removed from phase(f). If both f1 and f2 are not specified, the full range of f is used from trend analysis.

OUTPUT:
phase: phase with excess phase corresponding to delay removed (unwrapped, in degrees)
delay: time delay corresponding the the removed phase trend (in seconds)

5.41 mataa_plot_CSDt

file: ...mataa_tools/mataa_plot_CSDt.m

function mataa_plot_CSDt (spl,f,t,spl_range,annote,opts);

DESCRIPTION:
Plot cumulative spectral decay (CSD) data from mataa_IR_to_CSD(...) in a 3D diagram using slices of constant time t ('waterfall plot'). The argument 'annote' is optional, and can be used to specify annotations to be added to the titles of the plots.

INPUT:
spl,f,t: see description of output of mataa_IR_to_CSD
spl_range: the range covered on the y axis of the waterfall diagram (in dB)
annote: annotations to the plot title (string, optional)
opts: plot opts (sting or cell string containing multiple opts, optional). Currently, the following opts are available (for Octave 2.9.10 or newer):
opts = 'contours' : plot contours of waterfall diagram below the waterfall
opts = 'countours2': plot contours (lines) only in a 2-D plot
opts = 'shaded2': similar to 'contours2', but fills the areas in between the contours with a solid color)

EXAMPLE:
[h,t] = mataa_IR_demo ('FE108');
T = [0:1E-4:4E-3];
[spl,f,t] = mataa_IR_to_CSD (h,t,T,1/24);
mataa_plot_CSDt (spl,f,t,50);

5.42 mataa_plot_defaults

file: ...mataa_tools/mataa_plot_defaults.m

function mataa_plot_defaults

DESCRIPTION:
In earlier version of MATAA, this function sets default gnuplot state for MATAA plots in Octave. With the current version of MATAA, this function has no effect.

HISTORY:
26. December 2007 (Matthias Brennwald): commented out all commands so they have no effect anymore. Leave setting of plotting options to the user.
first version: 7. November 2006, Matthias Brennwald
%% if exist('OCTAVE_VERSION')
%% % do Octave specific stuff here
%% else

%% % do Matlab specific stuff here
%% %%% fh = gcf;
%% %%% p = get(fh,'Position');
%% %%% if p([3,4]) == [560 420];
%% %%% % make plots somewhat smaller than default
%% %%% p([3,4]) = [450 280];
%% %%% set(fh,'Position',p);
%% %%% end
%% %%% set(fh,'PaperPositionMode','auto'); % use same plot size for saving files as for plotting on screen
%% end
%% if mataa_settings('plotHoldState')
%% hold on
%% end
%%
%% % otherwise leave the plot state as it is (the user may have typed 'hold on' or something

5.43 mataa_plot_ETC_dB

file: ...mataa_tools/mataa_plot_ETC_dB.m

function mataa_plot_ETC_log (etc,t,annote,dB_range);

DESCRIPTION:
Same as mataa_plot_ETC, but uses a dB scale for the vertical axis.
The 'dB_range' parameter (optional) can be given to specify the dB range to be plotted. If not specified, a default value of 60 dB is used

5.44 mataa_plot_ETC_lin

file: ...mataa_tools/mataa_plot_ETC_lin.m

function mataa_plot_ETC_lin (etc,t,annote);

DESCRIPTION:
Plots the energy-time-curve (ETC) etc(t), using a linear y-axis scale.

INPUT:
etc: values of the energy-time curve (vector)
t: time values (vector)
annote (optional): annotation to the plot title (string)

OUTPUT:
(none)

EXAMPLE:
> t = [0:100]/1000; h = sin(200*t).*exp(-70*t);
> etc = mataa_IR_to_ETC(h,t);
> mataa_plot_ETC(t,etc, 'damped sine');

5.45 mataa_plot_FR

file: ...mataa_tools/mataa_plot_FR.m

function mataa_plot_FR (mag,phase,f,annote,fNorm,phaseUnwrap);

DESCRIPTION:
Plots frequency response magnitude, and phase (optional)

INPUT:
mag: magnitude of frequency response (in dB)
phase (optional): phase of frequency response (in degrees). If you don't want to plot phase, but other optional arguments below are required, use phase = [].
f: frequency coordinates of mag and phase (in Hz)
annote (optional): text note to be added to the plot title. If you don't want to add a note, but other optional arguments below are required, use annote = ''.
fNorm (optional): frequency to which the magnitude plot is normalised. If you don't want to normalise the plot, but other optional arguments below are required, use fNorm = [].
phaseUnwrap (optional): if phaseUnwrap is not zero, the phase is unwraped (so that discontinuities at +/- 180 deg. are avoided). Otherwise, phase is wrapped to +/- 180 deg.

EXAMPLE(S):
> [h,t] = mataa_IR_demo;
> [mag,phase,f] = mataa_IR_to_FR(h,t,1/12);
> mataa_plot_FR(mag,[],f); % plain vanilla plot of magnitude vs. frequency (without phase)
> mataa_plot_FR(mag,[],f,'demo',1000); % plots magnitude with an annotation to the plot title and normalizes mag by mag(f=1000).
> mataa_plot_FR(mag,phase,f,'demo again',80,1); % plots magnitude and phase with an annotation to the plot title. Magnitude is normalised such that mag(f=80) = 0 dB, and phase is unwrapped.

5.46 mataa_plot_HD

file: ...mataa_tools/mataa_plot_HD.m

function mataa_plot_HD (kn, annote);

DESCRIPTION:
This function plots the harmonic distortion spectrum in kn.

INPUT:
kn = [k1 k2 k3 ... kn] is the normalised distortion spectrum.
k1 corresponds to the fundamental frequency or first harmonic (k1 = 1, not plotted), k2 the component of second harmonic relative to the fundamental, k3 that of the third harmonic, etc.
annote (optional): optional annotation to be added to the plot title

EXAMPLE:
> [thd,k] = mataa_measure_thd(1000,1,96000); % measure THD and harmonic distortion spectrum
> mataa_plot_HD(k,'f0: 1kHz'); % plot the distortion spectrum

5.47 mataa_plot_impedance

file: ...mataa_tools/mataa_plot_impedance.m

function mataa_plot_impedance (mag,phase,f,annote);

DESCRIPTION:
Plots impedance (magnitude and phase) versus frequency.

INPUT:
mag: impedance magnitude (Ohm)
phase: impedance phase (degrees)
f: frequency (Hz)
annote (optional): text note to be added to the plot title.

OUTPUT:
(none)

5.48 mataa_plot_IR

file: ...mataa_tools/mataa_plot_IR.m

function mataa_plot_IR (h,t,annote);

DESCRIPTION:
This function plots the impulse response h(t).

INPUT:
h: impulse response samples
t: time coordinates of impulse response samples (vector, in seconds), or, alternatively, the sampling frequency of h(t) (scalar, in Hz)

annote (optional): text note to be added to the plot title.

EXAMPLE:
> [h,t] = mataa_IR_demo;
> mataa_plot_IR(h,t,'demo impulse response');

5.49 mataa_plot_one

file: ...mataa_tools/mataa_plot_one.m

function h = mataa_plot_one (x,y,figNum,plottit,xtit,ytit);

DESCRIPTION:
Plots y vs. x.

INPUT:
x: x values
y: y values to be plotted vs. x.
figNum: number (handle) of the figure window to be used for the plot. Use figNum = [] if the default window is to be used (e.g. the current plot window)
plottit: plot title.
xtit: x-axis label
ytit: y-axis label

OUTPUT:
h: handle to the axes of the plot.

5.50 mataa_plot_save

file: ...mataa_tools/mataa_plot_save.m

function mataa_plot_save (fileName);

DESCRIPTION:
Saves the last plot to an EPS (encapsulated post script) file.
'fileName' is the name (and path) of the file. If it does not include a path, the file is saved to the current directory (type 'pwd' to see the current directory).

5.51 mataa_plot_SR

file: ...mataa_tools/mataa_plot_SR.m

function mataa_plot_SR (h,t,annote);

DESCRIPTION:
This function plots the step response h(t).

INPUT:
h: step response samples
t: time coordinates of response response samples (vector), or, alternatively, the sampling frequency of h(t) (scalar)
annote (optional): text note to be added to the plot title.

EXAMPLE:
> [h,t] = mataa_IR_demo;
> [h,t] = mataa_IR_to_SR(h,t);
> mataa_plot_SR(h,t,'demo step response');

5.52 mataa_plot_TBESf

file: ...mataa_tools/mataa_plot_TBESf.m

function mataa_plot_TBESf (f,tau,A,ARange,tauRange,annote);

DESCRIPTION:
Plot tone burst energy storage data (as obtained from mataa_IR_to_TBES(...) in a 3D diagram using slices of constant frequency t.

INPUT:
f,tau,A: see description of output of mataa_IR_to_TBES
ARange: range of A axis
annote: annotations to the plot title (string, optional)

EXAMPLE:
> [h,t] = mataa_IR_demo ('FE108');
> f = logspace (2,4,50);
> [A,tau,f] = mataa_IR_to_TBES (h,t,f);
> mataa_plot_TBESf (f,tau,A,40,8,'FE108');

5.53 mataa_plot_time_signal

file: ...mataa_tools/mataa_plot_time_signal.m

function mataa_plot_time_signal (s,t,plottit,xtit,ytit,plotWindow);

DESCRIPTION:
This function plots the signal s(t).

INPUT:
s: signal samples
t: time values (vector, in seconds), or, alternatively, the sampling frequency of the signal (scalar, in Hz)
plottit: plot title.
xtit, ytit: labels for the x-axis and y-axis
plotWindow: number (handle) of the figure window to be used for the plot. Use plotWindow = [] if the default window is to be used (e.g. the current plot window)

5.54 mataa_plot_two

file: ...mataa_tools/mataa_plot_two.m

function h = mataa_plot_two (x,y1,y2,figNum,plottit,xtit,y1tit,y2tit);

DESCRIPTION:
Plots y1 and y2 vs. x.

INPUT:
x: x values
y1, y2: y values to be plotted vs. x. y2 may be empty (y2 = []), which will result in a single plot of y1 vs x.
figNum: number (handle) of the figure window to be used for the plot. Use figNum = [] if the default window is to be used (e.g. the current plot window)
plottit: plot title.
xtit: x-axis label
y1tit, y2tit: y-axis label of the y1 and y2 data

OUTPUT:
h: a 2-vector containig the handles to the axes of the two plots. If the second plot is omitted h(2) will be set to NaN,

5.55 mataa_plot_two_logX

file: ...mataa_tools/mataa_plot_two_logX.m

function h = mataa_plot_two_log (x,y1,y2,figNum,plottit,xtit,y1tit,y2tit);

DESCRIPTION:
Same as mataa_plot_two, but with logarithmic x axes.

INPUT:
(see mataa_plot_two)

OUTPUT:
(see mataa_plot_two)

5.56 mataa_realFT

file: ...mataa_tools/mataa_realFT.m

function [S,f] = mataa_realFT (s,t);

DESCRIPTION:
Identical to mataa_realFT0, but without the component corresponding to f=0.

INPUT:
(see mataa_realFT0)

OUTPUT:
(see mataa_realFT0)

5.57 mataa_realFT0

file: ...mataa_tools/mataa_realFT0.m

function [S,f] = mataa_realFT0 (s,t);

DESCRIPTION:
Calculates the complex fourier-spectrum S of a real signal s for frequencies f >= 0. Only the half spectrum corresponding to positive frequencies is returned, because for a real signal S(-f)=S*(f). S is normalized to length of s. The fourier spectrum S therefore does not depend on the sample rate used to digitize a given signal (i.e. S does not depend on the length of the signal). s can be of any length (no padding to length of 2n or even length necessary). In order to avoid frequency leakage, mataa_realFT does NOT pad s to even length. Each column of s represents one audio channel.

INPUT:
s: signal samples (vector containing the real-valued samples)
t: time values of the signal samples (vector, with evenly spaced values) or sample rate (scalar)

OUTPUT:
S: complex fourier spectrum of s ('positive' half, see also DESCRIPTION).
f: frequency values (vector)

5.58 mataa_realIFT

file: ...mataa_tools/mataa_realIFT.m

function [s,t] = mataa_realIFT (S,f);

DESCRIPTION:
Same as mataa_realIFT0, but without f=0.

INPUT:
S: complex fourier spectrum of the signal ('positive' half, see also DESCRIPTION).
f: frequency values (vector)

OUTPUT:
s: signal samples (real-valued samples)
t: time values of the signal

5.59 mataa_realIFT0

file: ...mataa_tools/mataa_realIFT0.m

function [s,t] = mataa_realIFT0 (S,f);

DESCRIPTION:
Calculates the inverse Fourier transform of a spectrum S(f) of a signal with real-valued samples. Only the 'positive' half of the spectrum is used, i.e. only positive frequencies (including f=0) must be given as input. See also mataa_realFT0.

INPUT:
S: complex fourier spectrum of the signal ('positive' half, see also DESCRIPTION).
f: frequency values (vector)

OUTPUT:
s: signal samples (real-valued samples)
t: time values of the signal

5.60 mataa_running_mean

file: ...mataa_tools/mataa_running_mean.m

function y = mataa_running_mean (x,n,w);

DESCRIPTION:
Returns a running mean of a data series x.

INPUT:
x: vector conaining the original data series
n: width of the smoothing window (number of samples, should be an odd number, n > 0)
w (optional): name of window type to be used. Default is 'rectangular', for other window types see mataa_signal_window

OUTPUT:
y: running mean of y, length(ym) = length(y)

EXAMPLE:
> N=1000; f0=500; fs=96000; t=[0:N-1]/fs; s = sin(2*pi*f0*t); % prepare a 500-Hz sine
> x = s+randn(size(s))/10; % create a noisy version of s
> y = mataa_running_mean(x,41,'hamm'); % remove the noise using a 41 samples wide Hamming window
> plot(t,x,'k',t,s,'g',t,y,'r') % plot the different versions of s

5.61 mataa_select_signal_window_time

file: ...mataa_tools/mataa_select_signal_window_time.m

function [t_start,t_end] = mataa_select_signal_window_time;

DESCRIPTION:
Interactively select start and end times of a signal.

INPUT:
(none)

OUTPUT:
t_start: start of selected signal range
t_end: end of selected signal range

input('Make shure that the window showing the signal-plot is active, and the zoom is set accordingly (press ENTER to confirm)...')

5.62 mataa_settings

file: ...mataa_tools/mataa_settings.m

function val = mataa_settings (field,value)

DESCRIPTION:
Retrieve and set MATAA settings.

mataa_settings with no arguments returns all the settings
mataa_settings(field) returns the value of the setting of 'field'
mataa_settings(field,val) sets the value of the setting 'field' to 'val'.
mataa_settings('reset') resets the settings to default values

EXAMPLES:
** get the current settings (this also shows you the available fields):
> mataa_settings

** get the current plot color:
> mataa_settings('plotColor')

** set the plot color to red:
> mataa_settings('plotColor','r')

** In principle, you can store anything in the MATAA settings file. For instance, you can store the birhtday of your grandmother, so you'll never forget that:
> mataa_settings('BirthdayOfMyGrandmother','1st of April 1925');

5.63 mataa_signal_analytic

file: ...mataa_tools/mataa_signal_analytic.m

function a = mataa_signal_analytic (s);

DESCRIPTION:
Calculate analytic signal a of signal s.

INPUT:
s: vector containing the samples values of the signal.

OUTPUT:
a: vector containing the analytic signal of s.

EXAMPLE:
calculate the amplitude envelope of the impulse response of a loudspeaker
> [h,t] = mataa_IR_demo; % load demo impulse response
> a = mataa_signal_analytic(h); % calculate analytic response
> a = abs(a); % abs(a) is the amplitude envelope of impulse response
> plot(t,a);

5.64 mataa_signal_autocorr

file: ...mataa_tools/mataa_signal_autocorr.m

function [c,T] = mataa_signal_autocorr (s,t);

DESCRIPTION:
Autocorrelation c(T) of signal s(t), for positive delays (T>=0).

INPUT:
s: vector containing the samples values of the signal.
t: time values of the signal samples (vector, in seconds, with evenly spaced values) or sample rate (scalar, in Hz).

OUTPUT:
c: vector containing the autocorrelation of s.
T: time lag (vector).

5.65 mataa_signal_calibrate

file: ...mataa_tools/mataa_signal_calibrate.m

function [h_corr,t,unit] = mataa_signal_calibrate (h,t,cal)

DESCRIPTION:
This function calibrates a signal h(t) (reflecting a DUT transfer function) using the given calibration data (e.g., for a specific audio interface, microphone, sensor, etc), and it will also (try to) determine the unit of the calibrated data. If only magnitude of the transfer function is given without phase information, phase is calculated by assuming the device to be minimum phase.
If h has more than one channel, different calibration information can be specified for the different channels.
See also mataa_load_calibration.

EXAMPLE with loudspeaker (DUT) tested using a microphone:

MATAA / COMPUTER —-> DAC (+BUFFER) —-> DUT —-> SENSOR —-> ADC (+PREAMP) —-> MATAA / COMPUTER
(dimensionless) (nodim -> V) (V -> Pa) (Pa -> V) (V -> nodim) (dimensionless)

===> unit of DUT transfer function is Pa/V

INPUT:
h: signal samples (unit: dimensionless data as obtained by ADC / soundcard)

t: time coordinates of samples in h (vector, in seconds) or sampling rate of h (scalar, in Hz)
cal: name of calibration file (e.g., 'Behringer_ECM8000_transfer.txt') or calibration data (struct object as obtained from mataa_load_calibration). For calibration of more than one data channels, cal can be specified as a cell array, whereby each cell element is used for the corresponding data channel.

OUTPUT:
h_corr: corrected signal
t: time coordinates of samples in h
unit: unit of h_corr (string), i.e. the unit of the calibrated DUT transfer function

helper function for calibration of various units

5.66 mataa_signal_clipcheck

file: ...mataa_tools/mataa_signal_clipcheck.m

function n = mataa_signal_clipcheck (s,N);

DESCRIPTION:
Returns the number of samples with amplitude less than N percent% lower than the maximum amplitude of the signal (absolute values).

INPUT:
s: vector of signal samples
N (optional): percentage of deviation from maximum amplitude. Default value is N = 1 (i.e. 1%).

OUTPUT:
n: number of samples with amplitude less than 1% lower than the maximum amplitude of the signal (absolute values).

EXAMPLES:
* White-noise signal (not clipped):
> wn = mataa_signal_generator('pink',1000,1); % a white-noise signal with 1000 samples (with sample ranges distributed in the range between -1...+1).
> n = mataa_signal_clipcheck(wn,0.1); % find number of samples with (absolute) amplitudes that are within 0.1% of the maximum (absolute) amplitude. This will result in a low value of n (i.e. n=1, 2, or 3, but higher values are unlikely).

* Clipped white-noise signal:
> wn = 2.5*mataa_signal_generator('pink',1000,1); % a white-noise signal with 1000 samples (with sample ranges distributed in the range between -2.5...+2.5).
> wn(wn > 1) = 1; wn(wn < -1) = -1; % fake clipping, i.e. truncate the samples to the range

(-1...+1).
> n = mataa_signal_clipcheck(wn,0.1); % find number of samples with (absolute) amplitudes that are within 0.1% of the maximum (absolute) amplitude. This will result in a much higher value of n than in the previous example (n ~ 200).

* Square-wave signal:
> sq = mataa_signal_generator('square',10000,0.1,1000); % a square wave signal with 1000 samples (i.e. a signal with sample values of either +1 or -1).
> n = mataa_signal_clipcheck(sq,0.01); % find number of samples with (absolute) amplitudes that are within 0.01% of the maximum (absolute) amplitude. This results in n=1000, because the amplitude of all samples is equal to 1.

5.67 mataa_signal_crop

file: ...mataa_tools/mataa_signal_crop.m

function [s,t] = mataa_signal_crop (s,t,t_start,t_end);

DESCRIPTION:
This function crops out the part of the signal s(t) in the range t = t_start...t_end

INPUT:
s: siglal samples
t: time coordinates of impulse response samples (vector, in seconds), or, alternatively, the sampling frequency of s(t) (scalar, in Hz)

OUPTUT:
s: signal samples of cropped signal
t: time coordinates of cropped signal (in seconds)

5.68 mataa_signal_generator

file: ...mataa_tools/mataa_signal_generator.m

function [s,t,info] = mataa_signal_generator (kind,fs,T,param);

DESCRIPTION:
This function creates a signal s(t) of a specified type.

INPUT:
kind: kind of signal (see below)
fs: sampling rate (in Hz)
T: length of the signal (in seconds)
param: Some signals require additional information, which can be specified in 'param' (a

vector or structure containing the required parameters, depending on the signal kind, see below)

kind can be one of the following:
'white': White noise (no additional parameters required)
'pink': Pink noise (no additional parameters required)
'MLS': Maximum length sequence (MLS). The 'T' parameter is ignored, and param = n is the number of taps to be used for the MLS. The length of the MLS will be 2^n-1 samples.
'sine','sin': Sine wave (param = frequency in Hz)
'cosine','cos': Cosine wave (param = frequency in Hz)
'sweep','sweep_log':Sine sweep, where frequency increases exponentially with time (param = [f1 f2], where f1 and f2 are the min. and max frequencies in Hz)
'sweep_lin': Sine sweep, where frequency increases linearly with time (param = [f1 f2], where f1 and f2 are the min. and max frequencies in Hz)
'sweep_smooth','sweep_log_smooth': Same as 'sweep' and 'sweep_log', but with a smooth fade-in and fade-out (to reduce high-frequency clicks at beginning and end)
'stepsweep','stepsweep_log': Stepped sine sweep; a series of time-shaped sine bursts, whereby the frequency is constant throughout each burst, and increases exponentially from one burst to the next. Bursts are shaped by a Blackman envelope for smooth transition from one burst to the next. The length of each burst is such that all burst contain the same number of sine cycles. param(1): frequency of first burst, param(2): frequency of last burst, param(3): number of bursts, param(4): fractional length of burst envelope with full amplitude [optional, default value: 0.7]. info.f: frequencies of bursts, info.i_end: indices to last sample in each burst, info.Nc: number of cycles in each burst
'square': Square (rectangle) wave (param = frequency in Hz)
'rectangle','rect: Same as 'square'
'sawtooth','saw': Sawtooth wave (param = frequency in Hz)
'triangle','tri': Triangle wave (param = frequency in Hz)
'dirac': Dirac signal (First sample 1, zeroes otherwise)
'zero': Zero signal ('silence')

OUTPUT:
s: vector containing the signal samples (tha values in s can range from -1...+1)
t: vector containing the sample times (in seconds)
info: additional information about the signal (empty in for most signal types; see 'kind' input above).

Examples:
1. Create a 1-second pink-noise signal 96kHz sample rate:
> [pink,t] = mataa_signal_generator('pink',96000,1);
> plot(t,pink)

2. Create a 0.1-second 1-kHz square-wave signal with 10 kHz sample rate:
> [sq,t] = mataa_signal_generator('square',10000,0.1,1000);
> plot(t,sq)

3. Create a 1-kHz sine burst windowed by a Hanning window:
> [burst,t]=mataa_signal_generator('sin',96000,0.01,1000);
> burst = mataa_signal_window(burst,'hann');
> plot(t,burst)

FURTHER READING:
- different kinds of noise: http://en.wikipedia.org/wiki/Colors_of_noise
- pink noise generation: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=5091&

- sine sweeps (chirp signals): http://en.wikipedia.org/wiki/Chirp

5.69 mataa_signal_pad_Zeros

file: ...mataa_tools/mataa_signal_pad_Zeros.m

function [s,t] = mataa_signal_pad_Zeros (s0,t0,T);

DESCRIPTION:
This function pads a signal s0(t0) with zeroes, i.e. replaces signal s0(t0) with s(t), where...
...s(t=t0) = s0(t0)
...s(t>max(t0) and t<T) = 0

The new signal s(t) therefore has length T

5.70 mataa_signal_removeHF

file: ...mataa_tools/mataa_signal_removeHF.m

function [s,t] = mataa_signal_removeHF (s,t,fc);

DESCRIPTION:
Removes signal components with frequencies higher than fc from s(t) by repeated convolution of s with a Hann window.

INPUT:
s: signal samples
t: time (vector, in seconds) or sampling frequency (scalar, in Hz)
fc: cut-off frequency (in Hz)

OUTPUT:
s: filtered signal samples
t: time

5.71 mataa_signal_save

file: ...mataa_tools/mataa_signal_save.m

function mataa_signal_save (s,fs,file,description);

DESCRIPTION:
Saves the signal s(t) to an binary file (Matlab 6 format).

INPUT:
...

OUTPUT:
...

5.72 mataa_signal_spectrogram

file: ...mataa_tools/mataa_signal_spectrogram.m

function [m,t,f] = mataa_signal_spectrogram (s,t,dt,smooth);

DESCRIPTION:
Calculate spectrogram (aka sonogram) of the signal s(t).

INPUT:
s: vector containing the samples values of the signal.
t: time values of samples in h (vector, in seconds) or sampling rate of h (scalar, in Hz)
dt: width time chunks used to calculate of spectrogram lines
smooth (optional): if specified, the data is smoothed in the frequency domain over the octave interval smooth_interval.

OUTPUT:
m: magnitude values in dB (matrix)
t: time values
f: frequency values

EXAMPLE:
fs = 44100; L = 3;
[s1,t] = mataa_signal_generator ("sweep_lin",fs,L,[1000 20000]);
s2 = mataa_signal_generator ("sweep_log",fs,L,[1000 20000]);
s3 = s1+s2;
[M1,T1,F1] = mataa_signal_spectrogram (s1,t,0.05);
[M2,T2,F2] = mataa_signal_spectrogram (s2,t,0.05);
[M3,T3,F3] = mataa_signal_spectrogram (s3,t,0.05);
subplot (3,1,1); surf (T1,F1/1000,M1); shading interp; view (0,90); ylabel ('Frequency

(kHz)');
subplot (3,1,2); surf (T2,F2/1000,M2); shading interp; view (0,90); ylabel ('Frequency (kHz)');
subplot (3,1,3); surf (T3,F3/1000,M3); shading interp; view (0,90); xlabel ('Time (s)'); ylabel ('Frequency (kHz)');

5.73 mataa_signal_to_TestToneFile

file: ...mataa_tools/mataa_signal_to_TestToneFile.m

function pathToFile = mataa_signal_to_TestToneFile (s,pathToFile,zeroTime,fs);

DESCRIPTION:
Saves the test signals in matrix s to a file on disk (for use with TestTone). Optionally, the signals are padded with zeroes at the beginning and the end.

INPUT:
s: the signal samples (in the range of [-1..+1]). In general, s is a matrix with each column corresponding to one data channel, and each row corresponding to a signal frame (i.e. all samples corresponding to the same time step). For single-channel data (i.e. mono signals), s is a column vector. A warning will be printed if s has more columns than rows.

pathToFile (optional): the path (including the file name) of the destination file. If not specified, a temporary file will be used. If you want to specify zeroTime and fs, but not pathToFile, use pathToFile = '';

zeroTime (optional): duration of 'zero signal' to be padded to the beginning and the end of the signal (in seconds). If not specified, no zeros will be padded to the signal.

fs (only if zeroTime is specified): the sample rate of the signal (in Hz). This is required to determine the number of 'zero samples'.

OUTPUT:
pathToFile: the path (including the file name) of the file to which the data was written.

NOTE 1: TestTone assumes that all information regarding the sample rate / time interval in between the samples is handled appropriately. mataa_signal_to_TestToneFile therefore does NOT handle any sample timing information. Only the sample VALUES are written to disk.

NOTE 2: the data in s should be padded with zeros at the beginning and the end of the signal to avoid problems with sound-I/O latency. If s does not include zeros at the beginning and the end, use the zeroTime option.

check format of input data:

5.74 mataa_signal_window

file: ...mataa_tools/mataa_signal_window.m

function s = mataa_signal_window (s0,window,par,len);

DESCRIPTION:
Multiplies the signal s0 by the window function with the name 'window', and returns the result in s.
Some window functions rely on a parameter, which can be specified by par (par can be omitted for those functions that don't rely on an extra parameter)

The following window functions are available (see e.g. http://en.wikipedia.org/wiki/Window_function for a description of these functions):
'rectangular', 'rect', 'nowindow' : rectangular window (i.e. no window at all)
'gauss': gauss window, whith shape parameter sigma = par (par <= 0.5)
'sin', 'cos','sine','cosine': sine / cosine window
'hamming', 'hamm': Hamming window
'hann': Hann window (cosine window). Note: in anology to the 'Hamming' window, this is often wrongly referred to as 'Hanning'. However, the name relates to a guy called Julius von Hann.
'bartlett','bart','triangular': Bartlett (triangular) window.
'blackman', 'black': Blackman window
'kaiser': Kaiser window with parameter alpha = par
'bingham': Bingham window with parameter par (par = 0 –> rectangular window, par = 1 –> Hann window).

Also, 'half' windows may be used, whereby the second half of the window is used. This is done by appending '_half' to the window name. This is useful, for instance, to attenuate echoes towards the end in an impulse response, while retaining the information at the beginning of the signal.

Furthermore, mataa_signal_window can also be used to return the window function itself, see example below.

INPUT:
s0: vector containing the samples values of the original signal (i.e. the signal that will be windowed).
window: string contining the name of the window type to be used (see above).
par: parameter(s) to further specify the window function. Depending on the window type, par may not be required (and will be ignored in these cases).

len: fractional length of full-amplitude range inserted between rise / fall of window slopes (optional, default: len = 0)

OUTPUT:
s: vector containing the sample value of the windowed signal.

EXAMPLES:

> s = mataa_signal_window(s,'hamming'); replaces s by a hamming-windowed version of itself

> s = mataa_signal_window(s,'hamming_half'); replaces s by a version of s windowed by the second half of a hamming window

> s = mataa_signal_window(repmat(1,1,1000),'gauss',0.4); returns just the gauss %

5.75 mataa_smooth_log

file: ...mataa_tools/mataa_smooth_log.m

function [y,x] = mataa_smooth_log (yRaw,xRaw,step)

THIS FUNCTION IS OBSOLETE. USE mataa_FR_smooth instead.

5.76 mataa_tempfile

file: ...mataa_tools/mataa_tempfile.m

function filepath = mataa_tempfile;

DESCRIPTION:
returns a path to a tempfile to be used with MATAA

INPUT:
(none)

OUTPUT:
filepath: string containing the path to the tempfile (including the file name).

5.77 mataa_t_to_f

file: ...mataa_tools/mataa_t_to_f.m

function f = mataa_t_to_f (t);

DESCRIPTION:
Same as mataa_t_to_f0, but the component corresponding to f=0 is removed from the output.

INPUT:
(see mataa_t_to_f0).

OUTPUT:
(see mataa_to_f0).

5.78 mataa_t_to_f0

file: ...mataa_tools/mataa_t_to_f0.m

function f = mataa_t_to_f0 (t);

DESCRIPTION:
This function returns the frequency bins of the fourier spectrum of a signal sampled at times t (vector). t must be be sorted and evenly spaced for this.

INPUT:
t: time values (vector, in seconds) of the signal

OUTPUT:
f: vector of the fourier-frequency bins (in Hz)

6 Getting started with MATLAB or Octave

MATLAB and Octave are powerful number crunching tools. While MATLAB is a commercial product, Octave is free and largely compatible with MATLAB. Both MATLAB and Octave run on various computer platforms. The name 'MATLAB' (*Mat*rix *lab*oratory) indicates that MATLAB (and therefore also Octave) basically work with matrices (for the non-mathematicians out there: a matrix is nothing more than a collection of numbers arranged in a rectangular way). This also includes scalars (i.e. a 1 x 1 matrix) and vectors (e.g. a 3 x 1 matrix for a vector containing 3 elements).

MATAA uses mostly scalars and vectors rather than 'full-blown' matrices. For example, consider a test signal made up by 2000 samples. This test signal would be stored in a vector with 2000 elements, or, in MATLAB terminology, in a 2000 x 1 matrix. Such a vector is also called a *column* vector, because its elements are arranged vertically. The same test signal might as well be represented by a 1 x 2000 matrix, which whould then be called an *row* vector, because its elements are arranged horizontally. I leave it to the MATAA user to choose between column and row vectors. However, I usually prefer to store test signals and the like in column vectors, because if the data is stored in a text file, I find it easier to read with a text editor.

In some later version of this manual I may add more information on using MATLAB and Octave. For the time beeing, please refer to the excellent tutorials listed below. These tutorials should get you started with MATLAB or Octave, but beware: you will not need to read (or even understand) every detail in these documents to run MATAA. Also note that 'MATLAB tutorials' are also useful for Octave users, and vice versa. Finally, typing `help` at the MATLAB/Octave prompt will display an overview of the MATLAB/Octave environment. Also, `help`an be used to get help on a specific command, e.g.: `help fft` will display information on the `fft` command (fast Fourier transform).

- Kermit Sigmon wrote an excellent MATLAB tutorial. It is available in HTML format for online viewing (`http://www.mines.utah.edu/gg_computer_seminar/MATLAB/MATLAB.html`) and as a post-script file for printing (`http://www.mines.utah.edu/gg_computer_seminar/MATLAB/primer.ps`).
- Mark Gockenbach wrote another good introduction to MATLAB. It is available in HTML format for online viewing (`http://www.math.mtu.edu/~msgocken/intro/intro.html`) and as a post-script file for printing (`http://www.math.mtu.edu/~msgocken/intro/intro.ps`).
- Henri Gavin has compiled a list of various MATLAB tutorials and books: `http://www.duke.edu/~hpgavin/MATLAB.html`
- The full documentation for Octave (by John W. Eaton) is available online : `http://www.gnu.org/software/octave/doc/interpreter/`
- Another Octave tutorial is available online at `http://homepages.nyu.edu/~kpl2/dsts6/octaveTutorial.html`
- Henri Amuasi, Carl Scheffler and Mike Pickles also wrote a nice Octave tutorial, which is available online at `http://www.aims.ac.za/resources/tutorials/octave/`
- There are also some Octave Wikis: `http://wiki.octave.org/` and `http://www.aims.ac.za/wiki/index.php/Octave`

- Finally, Wikipedia has nice pages on both MATLAB (http://en.wikipedia.org/wiki/MATLAB) and Octave (`http://en.wikipedia.org/wiki/GNU_Octave`).

Appendix A Licences

A.1 GNU General Public License (Version 2, June 1991)

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing 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

A.2 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.2.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept index

MATAA tools index