AMT 1.0: the toolbox for reproducible research in auditory modeling

Piotr Majdak, Clara Hollomey, Robert Baumgartner

The auditory modeling toolbox (AMT) is a Matlab/Octave toolbox for the development and application of auditory computational models with a particular focus on binaural hearing. The AMT aims for a consistent and structured implementation of auditory models, well-structured in-code documentation, inclusion of auditory data required to run the models, ability to reproduce the model predictions, and user-friendly access in order to allow students and researchers to work with and to advance existing models. Model implementations can be evaluated in two stages, by running so-called demonstrations which are quick presentations of a model and by starting so-called experiments aimed at reproducing results from the corresponding publications. The AMT 1.0 includes over 50 models and is freely available as an open-source package from www.amtoolbox.org.

1. Introduction

Our understanding of the world relies on observations. These observations drive the development of models and models explain the world [1]. In the domain of hearing research, auditory models are informative representations of processes describing the hearing system. Auditory models of conceptual nature qualitatively describe behavioral or neural outcomes. Computational models usually build upon such conceptual models and consist of algorithms designed to numerically process sound stimuli and to output a measurable quantitative metric [2]. Computational models are extremely important to reproduce results and confirm conclusions [3]. In this article, we describe the auditory modeling toolbox (AMT) version 1.0, which is a collection of computational models of the human hearing system with the focus on reproducible research.

The motivation for the development of auditory models is widely spread. Auditory models can help to evaluate how a deficit in one or more components affects the overall operation of the hearing system. They also can be useful in technical applications, such as the improvement of human-machine communication, as well as in clinical applications, such as the development of new processing strategies in hearing-assistive devices. An important property of auditory models is to represent results from an auditory experiment in order to explain the functioning of the targeted part of the auditory system. To this end, new auditory models are based on already existing ones, and the development of an auditory model begins with the process of comprehending and reproducing results of previously published models [4].

Unfortunately, when a new model is published in the usual form of a journal article, the model description and discussion about its properties is often not sufficient to reproduce the results. In addition, publishing the corresponding model implementation, i.e., a computer-readable sequence of commands to be executed, is required to ensure direct reproducibility. Thus, it is not surprising that articles describing computational algorithms have been sometimes described as “advertisement of scholarship” [5] whereas the scholarship itself is then represented by computer-executable code, input data, and parameters [6]. Hence, the spectrum of reproducibility is wide [7] – model implementations linked with the corresponding publication and data are vital prerequisites for a full replication of the research [8]. The AMT directly addresses the full research replication by providing executable code and data, both linked to extensive documentation referencing the corresponding publications (see Sec. 5).

Computational models of auditory processes can focus on various levels of details. For example, there are physiological models based on the description of cochlear mechanics or neural firing mechanisms. On the other end of the spectrum, there are phenomenological (or functional) models which describe an input-output relation by a high-level function without a strict relation to the underlying bio-

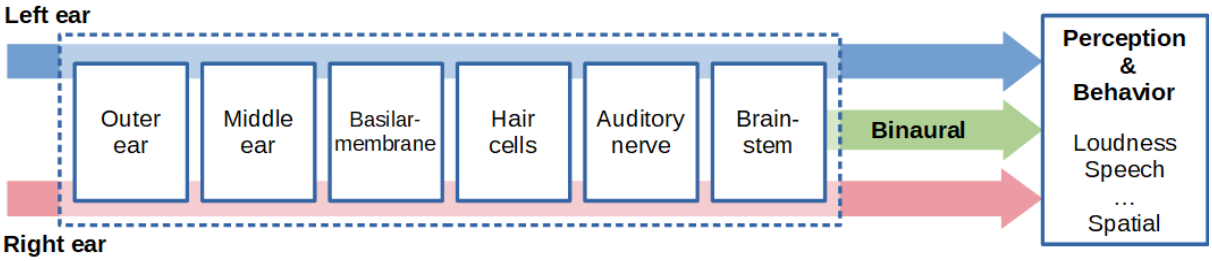


Figure 1: Functional structure of the AMT with stages reflecting the monaural processing stages of the auditory periphery (Left ear, Right ear), followed by an optional stage of binaural interaction (Binaural) and stages modeling perceptual or even behavioral outcomes (Perception & Behavior).

logical structures. Most computational auditory models follow a common functional structure in line with the ascending auditory pathway, which is reflected in the AMT in the way shown in Fig. 1: The sound entering and filtered by the outer ear is transmitted by the middle ear, frequency-decoded in the cochlea, neurally encoded by hair cells, transmitted via auditory nerves, and monaurally processed in the first nuclei of the brainstem. The monaural output of those nuclei is combined with that coming from the other ear (**Binaural** in Fig. 1). All three representations, from the left ear, the right ear, and the binaural stage are further processed in higher-level stages dealing with the modeling of perception and behavior. In the AMT, various types of models deal with various types of percepts such as loudness, speech, or space.

There are many auditory models with publicly available implementations. For example, ModelDB, one of the websites storing and listing computational neuroscience models [9], lists 65 implementations for the search word "auditory". While model sharing is common in the neuroscience community [10], most of the implementations focus on specific properties or stages of the auditory system, such as modeling the frequency selectivity in cochlear processing [11] or brainstem activities [12][1], respectively. Other types of implementations simulate multiple parts of the auditory pathway such as processing up to the auditory nerve [13][2],[14][3], auditory-cortical processing [15][4], or everything relevant to a specific percept such as loudness [16][5].

In contrast, the AMT is a *collection* of *auditory* models. It is implemented within the environment of Matlab [17] and Octave [18]. Besides the AMT, there are also other publicly available auditory model collections. The Auditory Toolbox is one of the earliest freely available collections and includes three cochlear models [19]. It is written for Matlab, but the development seems to have stopped in 1993. AIM [20] is a collection of models aiming to describe the formation of auditory events along the auditory pathway. Its development started in 1995 and the maintenance lasted until 2011 and 2013

1 https://www.urmc.rochester.edu/labs/carney/publications-code/auditory-models.aspx
2 https://github.com/HearingTechnology/Verhulstetal2018Model
3 https://www.ece.mcmaster.ca/~ibruce/zbcANmodel/zbcANmodel.htm
4 http://nsl.isr.umd.edu/downloads.html
5 https://www.psychol.cam.ac.uk/files/tv2018matlab.zip

for the offline and real-time version, respectively. Other notable but stagnated collections of auditory models are the HUTear [21] (developed until 2000), SOMA (until 2011)[6], EarLab [22] (until 2016), and Cochlea [23] (until 2017). DSAM is a potentially still maintained model collection that includes multiple auditory nerve models, neural cell models, and utilities [24]. While the main development happened before 2013, some activities on re-writing the code to C++ have been restarted in 2020.[7] Brian Hears, a collection of cochlear models [25], was developed until 2011 and has been recently upgraded to Brian2Hears and ported to be based on Brian2, an open-source simulator for spiking neural networks being under active development [26]. At its current development, Brian2Hears includes auditory models developed with a focus on computational efficiency.[8] Most of the auditory models from all those collections are also available in the AMT, which is under long-term active development and aims as a one-stop shop for auditory models within multiple programming environments (see Sec. 2.2) and offered under multiple licenses (see Sec. 5.2).

Besides these general collections, there are also collections of auditory models wrapped up for special applications. Examples of such collections are Eidos that targets speech analysis [27] or Two!Ears that targets the simulation of robotic hearing including basic cognitive functions [28]. As an interesting side note, the Two!Ears toolbox is based on an early version of the AMT, demonstrating the importance of freely available and reproducible implementations of auditory models.

Beginning with the first AMT release (the version 0.003 released as a draft in 2010), the continuous contributions from the auditory community have helped us to extend the AMT. Through the AMT versions 0.x (described in [29] and released between 2013 and 2020), the AMT has matured to a large collection of auditory models. The release of the AMT 1.0 offers new features and important structural changes. In this article, we describe the AMT as of the version 1.0, referring to it as the AMT. Following the semantic versioning[9], this description applies to all future AMT 1.x versions. In the next sections, we describe the environment of the AMT, the available models, their tools and status, and general information such as the documentation system. Finally, we provide some practical tips for getting started to work with and contribute to the AMT.

2. Environment

The AMT has been developed within the environment of Matlab 2016a [17] and Octave 6.2 [30]. Figure 2 shows the logical structure of the AMT. The auditory models are embedded in model-independent resources such as core functions, common functions, a caching mechanism, and access to

6 https://code.soundsoftware.ac.uk/projects/soma
7 https://sourceforge.net/p/dsam/git/ci/1619b6db26b0b862c5c323c75e1fed630df2e8fd/
8 https://brian2hears.readthedocs.io/en/stable/index.html
9 https://semver.org/

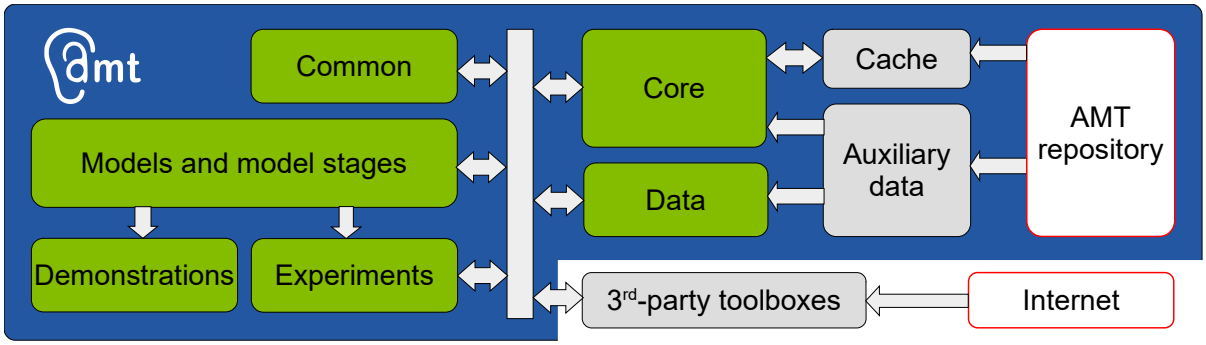


Figure 2: Logical structure of the AMT. **Green**: Code consisting of functions and algorithms. **Grey**: Data. **White**: Information available online for download.

auxiliary data with an online data repository. Further, the AMT uses various third-party toolboxes, mostly to accommodate individual model requirements.

2.1. Core functions

The core functions control the configuration and the workflow of the AMT. The most essential function is `amt_start` that installs toolboxes when required and sets up a default configuration for the work with other AMT functions. This function supports flags to control the global behavior of the AMT such as the behavior of the cache (see Sec. 2.3) and the message display control. For example, when the AMT is started in the default `verbose` mode, messages will always be displayed. When the AMT is started in the `silent` mode instead, display of all messages will be suppressed.

The AMT configuration can be retrieved with `[f,k]=amt_configuration`, where `f` returns the configuration flags and `k` returns the status, paths of toolboxes (see Sec. 2.2), base path of the AMT, paths of auxiliary data (see Sec. 2.4) and cache (see Sec. 2.3), as well as names of the current and previous AMT versions. The configuration can be displayed by calling `amt_info`. The flags can also be obtained by the function `amt_flags`. Most of the configuration parameters can also be obtained by the functions `amt_basepath`, `amt_auxdatapath` and `amt_auxdataurl`, `amt_cache`, and `amt_version`. The AMT can be stopped with `amt_stop`, which removes the configuration from the system, but does not delete the user's variables.

The AMT core function `amt_emuexp` provides a functionality to emulate psychophysical experiments by simulating the underlying processes. In a typical psychophysical experiment (e.g., Fig. 3, top), after an initialization of the procedure, signals are generated and provided to a subject. Then, the subject processes them and based on cognitive mechanisms provides a decision. This decision triggers the procedure to continue with the experiment until it finishes with a result. The function `amt_emuexp` emulates all those components (Fig. 3, bottom). The initialization phase consists of setting up `amt_emuexp` by separately providing parameters for the experiment, signal generation, auditory model, and decision stage. Each of the four initializations are triggered by calling `amt_emuexp`

with the corresponding init parameter. The experiment is started by calling `amt_emuexp` with the run parameter. Then, signals are generated, their processing by the auditory model is triggered, and the model outputs are provided to the decision stage. The decision output triggers the `amt_emuexp` procedure to continue with the simulation until it finishes with a result. Note that functions for creating the signals, modeling the auditory system, and providing the decision are not part of the `amt_emuexp` and can be any functions of the AMT environment. An example of using `amt_emuexp` can be found in `demo_breebaart2001` demonstrating a three-alternative forced choice experiment from [31]. The functionality of `amt_emuexp` also supports emulating experiments following the interface proposed in [32]. This interface integrates the experiment software, an auditory pathway model, and task-dependent decision stages – on potentially remote computers irrespective of their underlying programming language. In `amt_emuexp`, this mode is initiated by initializing the `amt_emuexp` experiment with the parameter `interface` set to `ModelInitiative`. An example of performing an experiment using the model initiative of [32] can be found in `exp_breebaart2001` replicating Fig. 3 from [31].

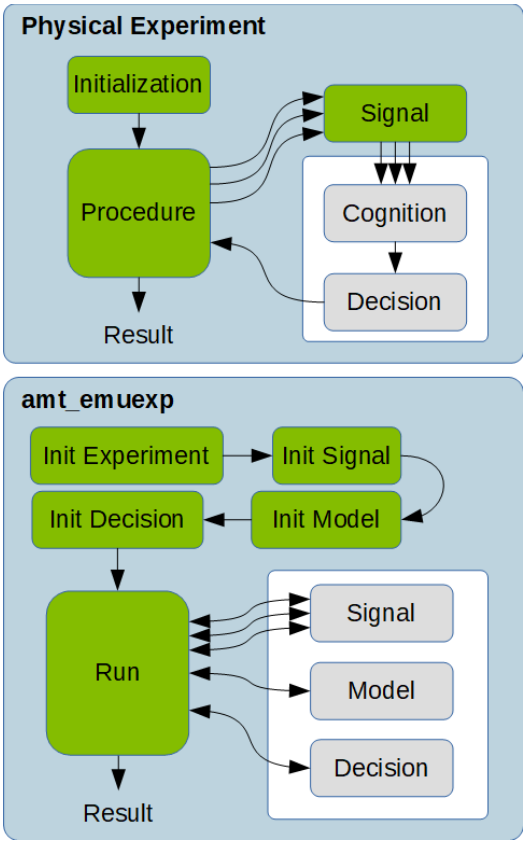


Figure 3: Example of a typical psychophysical experiment (top) and its emulation within the AMT (bottom). **Top**: Physical experiment with an initialization the main procedure triggering signal generation provided to the subject, who receives the signals, processes them and provides a decision, terminating with a result after sufficient number of iterations. **Bottom**: Emulation of that experiment by `amt_emuexp`: Initialization of the corresponding components (`Init`), procedure stage (`Run`) controlling the signal generation, modeling the cognition, triggering the decision stage, and calculating the result. **Green**: integral parts of the experiment procedure (top) and `amt_emuexp` (bottom). **Grey**: Experiment parts representing the subject (top) and functions outside the `amt_emuexp` (bottom).

The AMT core function `amt_disp` is used for displaying text in the command window, specifically targeting the AMT configuration. It is an obligatory replacement for Matlab's built-in function `disp`. When called without further flags, `amt_disp` outputs the message in a similar way to `disp`, with the behavior depending on the global verbose mode of the AMT (see details of `amt_start` for further explanation on the start-up configurations). When called with the parameter `volatile`, progress can be displayed in loops, i.e., by calling `amt_disp(..,'volatile');` in a loop and calling `amt_disp();` after the loop. For messages

6

showing results important for the online documentation, `amt_disp` can be called with the parameter `documentation`.

The AMT core functions further warrant proper functioning through compilation of binaries on the user's system (`amt_mex`), calling functions from external environments (`amt_extern`), or handling recursive search within directories (`amt_subdir`), among others.

2.2. Third-party toolboxes and interfaces to external environments

The AMT uses various third-party toolboxes and provides interfaces with other programming environments. Third-party toolboxes are packages of code developed by others independently of the AMT. They have their own but AMT-compatible license and are used by the AMT but not owned by the AMT team. These toolboxes are stored in the directory `thirdparty`. Alternatively, a user can also store them anywhere on the system and make them available within the environment's search path.

We distinguish between essential and model-dependent toolboxes. The large time-frequency analysis toolbox (LTFAT) is an essential toolbox, which means that the AMT will not run without it. The AMT uses the LTFAT's core functions such as parsing the input parameters (LTFAT function `ltfatarghelper`) and signal-processing matrix functions such as `assert_sigreshape_pre`. If locally not available, the LTFAT will be automatically downloaded and installed. The AMT will terminate with an error if that procedure fails.

The model-dependent toolboxes are not required to run the AMT, however, they are required when executing specific models. There is a variety of such toolboxes used within the AMT. For example, the application programming interface (API) for the spatially oriented format for acoustics (SOFA) handles head-related transfer functions (HRTFs) stored as SOFA files and provides general functionality to analyze, process, and display such data [33]. In the AMT, the SOFA API is used by models requiring HRTFs. Another example is the sound field synthesis toolbox (SFS), which is used by `wierstorf2011` in order to approximate the sound field provided by loudspeaker arrays before modeling its auditory processing [34]. These toolboxes are provided within the AMT full release package (see Sec. 4.1).

The AMT supports interfaces to external environments. This way, the AMT is not limited to run models written for Matlab or Octave only – the model processing is triggered by `amt_extern` within the AMT environment, but these models are actually run outside of Matlab or Octave. These external environments can be installed anywhere on the user's system as long as they are accessible within the AMT environment. This can be done by setting corresponding paths on the user's system. Files in-

tended to be executed by an external environment are stored in the directory `environments`. Currently, two environments are in use.

Models implemented in Python can be executed by calling the AMT function `amt_extern`. In the AMT 1.0, models `verhulst2012`, `verhulst2015`, and `verhulst2018` use Python version 3 installed with packages `numpy` and `scipy`.

Models implemented in low-level programming languages such as C and C++ can be compiled to binary executable files, which are then executed on the user's machine. These files are compiled as by `amt_mex` which compiles C++ files into MEX binaries which shadow Matlab or Octave files for faster processing. These files are stored in the directory `mex` and `oct` to work in Matlab and Octave, respectively. Further, `amt_mex` executes `make.bat` and `makefile`, on Windows and Linux, respectively, to compile native binaries used by external environments and stored in the AMT directory `environments`. For compiling, the AMT requires the GNU compiler collection (GCC).

2.3. Cache

The AMT uses a two-level caching mechanism for storing pre-calculated results. The first level of the cache is locally stored in the AMT directory `cache`. That local cache has a read/write access. This means after having some data calculated, the results can be stored in that local cache and be accessed later on. The second-level cache is integrated in the online repository of the AMT, in which the cache data is stored for each AMT version separately. This online cache is read-only. Hence, the AMT can pull data from it and store it in the local cache. However, only the AMT team can push new data to the online cache. This combination of two cache levels avoids recalculations at the local level of a user and ensures a valid online cache data controlled by the AMT team. The online cache uses incremental versioning. Hence, only the data different from the previous version are stored.

The AMT cache mechanism is controlled by the function `amt_cache` that supports the commands `get` for accessing cached data as well as `set` for storing calculated data in the local cache (and further commands controlling the behavior of the cache system, see the documentation of `amt_cache`). When accessing cached data, first the data are searched in the local cache, and if not available, the online cache is accessed. The data are stored in the AMT's cache directory and subsequent directories named by the caller function, in files named by the user. For example, when the file `example.m` executes the command `amt_cache('set','xyz', a, b, c)`, the AMT will create a directory named example and store the variables `a`, `b`, and `c` in the file `xyz.mat`. The access to such a cached file is then given by calling `[a, b, c] = amt_cache('get', 'xyz')`. If the cached file (`xyz.mat`) does not exist, neither locally nor online, the output variables (`a`, `b`, `c`) will be empty indicating that a recalculation should be triggered. For more details, see the help section of `amt_cache`.

The AMT supports four global caching modes helping to control the access to the local and online repository. When calling `amt_cache` in the `normal` mode, the local cache will be used. If the data is not available in the local cache, it will be downloaded from the internet. If it is remotely not available, recalculation of the data will be enforced. Note that this method may by-pass the actual processing and thus does not always test the actual functionality of a model. It is, however, convenient for fast access of results like plotting figures. In contrast, the `redo` mode enforces the recalculation of the data without even checking the cache. In the `redo` mode, the `get` command of `amt_cache` always outputs empty variables, triggering the recalculation. The `cached` mode is the opposite to the `redo` mode and enforces `amt_cache` to always use the cached data. If the cached data are not available, neither locally nor remotely, an error will be thrown. In the fourth cache mode `localonly`, cached results will be loaded from the local cache, but will be recalculated if locally not available. This mode is intended for running the AMT without access to the internet.

All these four caching modes are supported by the get command of `amt_cache`, allowing the user individually to force the cache behavior of specific AMT functions. For example, `exp_lindemann1986('fig6')` plots Figure 6 from [35] based on cached results, whereas `exp_lindemann1986('fig6','redo')` calculates the data and then plots that figure.

2.4. Data

Most of the models require data to run and test them. The AMT provides various mechanisms to access the data. We distinguish between auxiliary data, data functions, and the access to HRTFs.

The auxiliary data are large chunks of data that are not provided with the AMT code. This data can be accessed with the function `amt_load`. The data are retrieved from the directory `auxdata`, where they are locally stored and structured by the model name. Correspondingly, `amt_load` requires two parameters: the model and the name of the dataset. An optional third parameter can be used to load only a single variable from a larger dataset. If the requested dataset is locally not available, it will be downloaded from the AMT online auxiliary data repository, and locally stored in the AMT directory `auxdata` for future usage. Note that the local `auxdata` directory contains data for the particular AMT version only and the online `auxdata` repository contains data for each AMT version separately based on incremental versioning. Hence, only the data different from the previous version are stored. Further note that `amt_load` loads MAT files per default, but it can also be used to load audio files in WAV format, in which case, `amt_load` returns two variables: the audio data and the sampling rate.

Data directly referring to a publication can be accessed by the so-called data functions which have the prefix `data_`, e.g., `data_majdak2010` returns the localization responses from [36]. Data functions provide a more intuitive access because they can provide a documentation within the function's

in-code documentation and refer to the corresponding publication. Note that some of the data functions internally use `amt_load` to load larger chunks of auxiliary data.

The third data category contains HRTFs, describing the acoustic filtering of the sound by the listener's body, in particular, the head, torso, and ears [37]. The AMT stores HRTFs in the directory `hrtfs` as so-called SOFA files, and uses the SOFA API for Matlab/Octave for their handling. Similarly to `amt_load`, the API's function `SOFAload` for loading an HRTF dataset supports caching of the HRTFs, which are, if locally not found, automatically downloaded from the AMT online HRTF repository.

2.5. Common functions

Common functions are helpers and converters used by models and model stages. A common function represents an algorithm with an established functionality within the auditory community, has a technical background, and can be used among various models. They are not part of the AMT structural core (in contrast to `amt_` core functions), they are also not model-specific (in contrast to model stages), and they do calculations (in contrast to data functions). Common functions are stored in the directory `common` and usually created by the AMT team as soon as multiple models use a similar functionality that can be integrated to a single stand-alone function.

In the AMT, we have common functions that perform filter bank processing, envelope extraction, fading of signals, frequency-scale conversions, level conversions, and much more. Common functions can also calculate important parameters such as standardized hearing thresholds. In contrast to data functions, common functions perform some calculations, e.g., by interpolating or numerically evaluating a formula.

An important property of common functions is their model independence. This is obvious when considering functions such as `sph2horpolar` that converts between spatial coordinate systems. Still, model-dependent functionality of a common auditory function can be triggered by using model-dependent flags. An example is `ihcenvelope` that implements a widely used model of the inner hair cell in terms of signal rectification followed by low-pass filtering. Model specific parametrization of `ihcenvelope` can be triggered by using model-specific flags supported by `ihcenvelope`. For instance, in order to use the inner hair cell processing with parameters from [38], the flag `ihc_bernstein1999` can be used, i.e., `ihcenvelope(insig, fs, 'ihc_bernstein1999')`.

3. Models

The auditory models are implemented as model functions within the AMT and have their associated model stages. Further they are accompanied by model-specific plotting functions, signal generators,

demonstrations, and experiments for result reproduction. Also, the status of each model implementation is tracked in order to provide an estimation of the quality of the integrated models.

3.1. Models and their stages

In the AMT, a model is a stand-alone and testable algorithm publicly described in a scientific article discussing model parameters and providing evaluation results. Model implementations are stored as functions in the AMT directory `models`. They are named by the last name of the first author and the year of the corresponding publication, e.g., `dietz2011` for [39]. While this naming convention may appear unfair to the remaining contributing researchers, it is simple and provides great visibility to the principal author who is in most of the cases also responsible for the model implementation. If there are multiple publications with the same last name and year, a short but descriptive postfix is appended after the year, e.g., `exp_baumgartner2015binweight` distinguishes from `exp_baumgartner2015`.[10]

A model function can further depend on other functions explicitly linked with the model. These so-called model stages are usually not stand-alone, i.e., they are part of a model and they need a model in order to be tested. Only model stages belonging to a model can be included in the AMT. Model stages are stored in the directory `modelstages` and, in order to pronounce the link to the model, model stages have the prefix of the model function followed by an underscore and stage description, e.g., `dietz2011_interauralfunctions`.[11]

For some model stages of the auditory pathway, established approaches are available and are used by a variety of other models. Such model stages do not have an explicit link to a specific model and are thus integrated as common functions, see Sec. 2.5. In the following we briefly describe models and common functions following the functional structure shown in Fig. 1.

The processing of the outer ear is supported by the common functions `headphonefilter`, and `hrtf2dtf`, combined with model-dependent HRTFs and the corresponding SOFA functions enabling the modeling of HRTF effects. The common function `itdestimator` collects commonly used approaches to extract timing information from HRTFs, which can be checked by `ziegelwanger2014` [40] for their geometrical consistency. The transmission properties of the middle ear can be modeled by the common function `middleearfilter`.

Signal processing approaches to approximate cochlear processing consist of modeling the basilar membrane excitation and subsequent transmission by inner hair cells. Modeling the basilar membrane

10 Note the lack of underscore between the year and the postfix showing that `baumgartner2015binweight` is not part of `baumgartner2015`.

11 Note the underscore between the year and the postfix showing that `interauralfunctions` is part of `dietz2011`.

velocity as a function of frequency can be done by `lopezpoveda2001` [41], `hohmann2002` [42], `lyon2011` [11], and `verhulst2012` [43] as well as the common functions `auditoryfilterbank`, `gammatone`, `gammachirp`, or `ufilterbankz`. These approaches include the active feedback of the outer hair cells in various ways. The transmission of the inner hair cells is supported by the common function `ihcenvelope`, which can be parametrized depending on the targeted behavior.

The auditory nerve (AN) is often linked to the models of the basilar membrane and hair cells. Hence, the models `zilany2007` [44], `zilany2014` [45], and `bruce2018` [14] implement the complete chain from sound pressure to spike rates of AN fibers. The AN functionality alone can be modeled with the common function `adaptloop` [46], which can be parametrized to simulate various approaches [31, 46–49].

An important property of the neural auditory pathway from the cochlear nucleus to the inferior colliculus is the sensitivity to temporal modulations, which is supported by the models `ewert2000` [50] and `carney2015` [12] as well as by the common function `modfilterbank` [51]. Models that also integrate the more peripheral stages with some processing of higher neural stages are `dau1996` [46], `dau1997` [47], `roenne2012` [52], `verhulst2015` [53], `verhulst2018` [13], `relanoiborra2019` [48], and `king2019` [54]. Note that some models output individual neural spikes but others output spike rates only or even more abstract measures of neural activities.

Binaural processing is supported by the models `lindemann1986` [35], `breebaart2001` [31], `dietz2011` [55], and `takanen2013` [56]. The common function `itd2angle` further provides a simple solution to create a relationship between the processed binaural timing cues and sound incidence angle.

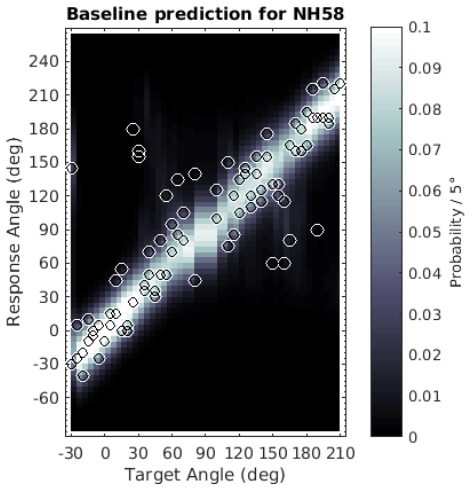
The last section involves modeling various aspects of perception. In the AMT, we have loudness models represented by `moore1997` [57], `glasberg2002` [58], and `chen2011` [59] as well as a loudness model considering binaural inhibition, `moore2016` [60]. We have monaural speech perception models such as `joergensen2011` [61], `taal2011` [62], and `joergensen2013` [63] and those considering binaural speech processing such as `culling2004` [64], `jelfs2011` [65], and `hauth2020` [66]. For perceptual similarity, we have `osses2021` [67] as a monaural model and `mckenzie2021` [68] as a binaural model. Last but not least, we have many models of spatial perception: `zakarauskas1993` [69], `langendijk2002` [70], `may2011` [71], `baumgartner2013` [72], `georganti2013` [73], `wierstorf2013` [34], `baumgartner2014` [74], `reijniers2014` [75], `kelvasa2015` [76], `baumgartner2016` [39], `hassager2016` [77], `baumgartner2017` [78], `li2020` [79], `baumgartner2021` [80], `barumerli2021` [81], and `mclachlan2021` [82]. Those models of spatial perception focus on either the direction, externalization, or distance of the sound source. The most recent models apply Bayesian inference to account for higher cognitive processes of

information integration across cues, modalities, and time.

3.2. Model-dependent plotting and signal generation

While Matlab and Octave provide a huge variety of functions for signal generation and plotting, the AMT provides functions specifically supporting the needs of auditory models. Signal generators have the prefix `sig_` and are stored in the `signal` directory. Plotting functions have the prefix `plot_` and are stored in the `plot` directory.

For the file names, we follow the same pattern as for other parts of AMT functionality. Generators and plotting functions, which have been made specifically for a particular publication, have the author-year identifier in their file names, e.g., `sig_hartmann1996` generates the signals tested in [83]. Functions which are used more widely across multiple publications and represent more general plotting and signal generation, have their functionality in the filename, e.g., `sig_ildsin` generates a binaural sine wave with an interaural level difference.




```
Performance Predictions:
------------------------
Quadrant errors (%)              4.8
Local polar RMS error (deg)     23.8
Local polar error rate (%)       13.6
```

Figure 4: Output of the demonstration `demo_baumgartner2014` consisting of a panel shown in a figure and alphanumeric output to the command window. **Panel:** The circles show the actual responses of the listener NH58 from [36] localizing noise bursts presented from various target angles along the median plane. Brightness encodes the probability to respond at an angle predicted by `baumgartner2014` [74]. **Text:** predicted localization performance for that listener.

3.3. Demonstrations and experiments

Demonstrations are scripts which can be run without any parameters in order to demonstrate the functionality of a model or data. Demonstrations have the prefix `demo_` and are stored in the directory `demos`. Demonstrations provide a visual representation of a model output. Figure 4 shows an example of a demonstration. Demonstrations are scripts, not functions, thus, they finish with all used variables in the user's workspace, ready to be inspected by the user for easily getting into the model's functionality.

Experiments address the reproducible-research objective of the AMT. They aim at reproducing model results from publications related to the models. Ideally, they produce exactly the same results as that from the corresponding article. By visually comparing the experiment output, the quality of the model can be estimated, see the following section on the model status. Experiments are functions (not scripts) with the prefix `exp_` followed by the last name of the first author and the year of the publica-

13

tion reporting the simulation results. Most of the AMT's experiments are those replicating the outcome of a publication describing a model, e.g., `exp_bruce2018` replicates figures from [14] obtained by running `bruce2018`. A parameter with the prefix `fig` is used to produce a figure, e.g., `exp_baumgartner2014('fig2')` reproduces the Fig. 2 from [74]. Prefix `tab` is used to produce a table, e.g., `exp_baumgartner2014('tab2')`.

In addition to the experiments replicating the model publication, further experiments replicate the outcome of secondary publications, i.e., those which apply an already published model. In the AMT, the naming convention is to use the naming of the secondary publication. At the moment of writing, in the AMT, we have only a few of such cases: `exp_baumgartner2015` replicating the outcome from [84] and `exp_baumgartner2015binweight` replicating the outcome from [85], both applying `baumgartner2014` from [74]; `exp_engel2021` from [86] applying both `reijniers2014` and `baumgartner2021`; `exp_osses2022` from [86] applying `dau1997`, `verhulst2015`, `verhulst2018`, `bruce2018`, `king2019`, `relanoiborra2019`, and `osses2021`; `exp_roettges2022` from [87] applying `hauth2020`; and `exp_steidle2019` from [88] applying `itdestimator` and `ziegelwanger2014`.

Experiments can also be used to pass the reproduced results to the caller functions for further processing. Their difference to the functions providing auxiliary data (`data_`) is that experiments process some data by the models, whereas the data functions just load existing data. Still, the focus of the experiments is to reproduce model results. If some experiment output is used frequently by others, this output is a good candidate for a transfer to a data function in the future.

3.4. Model status

The AMT tracks the status of each model implementation in order to provide an estimation of the quality of the models available in the AMT. The model status describes the quality of the model's source code and documentation, as well as verification. The verification considers the correspondence between the results shown in the corresponding publication and the results provided by the AMT implementation, usually implemented within the experiment functions.

The status of the code and the documentation distinguishes between four states (with letter score in brackets):

- **Submitted (D)**: The model has been submitted to the AMT team and the code has been included in the source-code repository as submitted, but it has not been integrated yet (no documentation, potential compilation errors, libraries missing, etc). In the release version, the model neither appears on the website nor is available for download. The current state of the integration can be provided upon request.

- **Satisfactory (C)**: The code/documentation fits the AMT conventions just enough for being available in the release version for download. The model and its documentation appear on the website, but major work may still be required.
- **Good (B)**: The code/documentation follows the AMT conventions, but there are still open issues.
- **Perfect (A)**: The code/documentation fully complies with the AMT conventions, no open issues.

Note that after reaching the status "Perfect", that model status remains even when a minor issue appears (most probably raised by an AMT user).

The status of the verification consists of four states:

- **Unknown (D)**: The AMT can not run experiments for this model and can not produce any results usable for verification. This is the default state right after having a model implementation provided to the AMT team.
- **Untrusted (C)**: The verification code is available but the experiments do not reproduce the relevant parts of the publication. The AMT team is seeking for a solution to reveal the problems in the discrepancy between the publication and the implementation.
- **Qualified (B)**: The experiments produce similar results as in the publication in terms of showing trends and explaining the effects, but not necessarily matching the numerical results. Explanation for the differences can be provided, for example, not all original data being available, or publication being affected by a known and documented bug.
- **Verified (A)**: The experiments produce the same results as in the publication. Minor differences are allowed if a plausible explanation is provided, e.g., layout issues in the graphical representations or randomness introduced by noise or other probabilistic modeling approaches.

A table providing an overview of the available models and their status can be found on the AMT website[12]. Note that the status is only a snapshot of the development because the implementations in the AMT are continuously developed, evaluated, and improved. Any feedback is appreciated either via email or ticket created on SourceForge.

4. Working with the AMT

4.1. Getting started

The AMT 1.0 has been developed and tested with Matlab 2016a and Octave 6.2 under Windows 10 Pro (2004) and Linux Ubuntu Focal Fossa (20.04 LTS). While most of the models will work with even older versions of Matlab and Octave, we recommend using one of the tested environments. In

12 http://amtoolbox.org/

addition to the pure Matlab installation, additional toolboxes, such as the “Signal processing toolbox” and the “Statistics and machine learning toolbox” may be required to run some of the models. With Matlab 2016a or later, these toolboxes can be installed by using the Add-On Explorer within the Matlab user interface. Similarly, when run in Octave, some models require additional packages such as `signal`, `statistics`, and `netcdf`, which need to be installed by the user before starting the AMT.

The AMT also requires third-party toolboxes (see Sec. 2.2). The toolboxes can be obtained in several ways. The easiest and recommended method is to use the AMT releases marked as `amtoolbox-full` packages, which contain all those toolboxes. With a full package downloaded and unzipped on the user’s system, `amt_start` starts the AMT.

The second method is to let the AMT download the toolboxes during the first start of the AMT. This is required when the non-full package of the AMT release is available. The download and toolbox installation can be then executed by `amt_start('install')`, which triggers an interactive inquiry of the toolboxes to be downloaded and installed. Note that, if none of the toolboxes are available, `amt_start` per default downloads the LTFAT and terminates with an error if LTFAT installation fails because the LTFAT is an essential toolbox. But, `amt_start` does not download other toolboxes because they are not essential to a large part of the AMT functionality.

The third method is the manual installation of the third-party toolboxes by the user. This method provides the most flexibility at the price of most effort and required knowledge about the user’s system.

Some of the AMT models need to be compiled for the particular system of the user. For users’ convenience, some pre-compiled binaries are provided with the release of the AMT, however, binaries compiled for the user’s system may be required in some cases. This can be done by executing the command `amt_mex`, which 1) compiles corresponding Matlab/Octave files to the so-called MEX binaries, and 2) compiles the C and C++ files provided with the AMT to binary files. In order to run `amt_mex`, the GNU compiler collection (GCC) must be available as the command `gcc` within the AMT environment. The particular package depends on the user’s operation system and can be downloaded from many sources on the internet. The availability can be checked by two means: 1) calling `mex -setup` shows the compiler available to compile MEX files, or 2) calling `system('gcc')` terminates with an error of no input files provided only if GCC is available to compile C and C++ files. Note that `amt_start('install')` also executes `amt_mex` in order to provide compiled models.

The AMT configuration and status of available toolboxes is handled by the command `amt_configuration` (see Sec. 2.1). Information on which toolboxes are required for running a specific model can be queried by executing `amt_info` with the model name as parameter.

4.2. Contributing to the AMT

Each AMT user is warmly welcome to contribute to the AMT. The motivation for a contribution can be manifold. For example, an AMT user has implemented a model and wants the findings to be available and accessible for future research as this increases its potential impact. Or, an AMT user has applied a model from the AMT and has implemented experiments or demonstrations displaying the model functionality. Or, an AMT user has a general urge to support reproducible research and learn about open-source projects and auditory modeling.

In any case, we highly encourage any AMT user – before locally modifying AMT files – to retrieve the most recent version from the source repository[13] and integrate the modifications into that repository. This way, modifications can be uploaded to the online repository, will not be forgotten on the local computer, and can be spread among the AMT community. To this end, we recommend the following workflow (we assume a general knowledge of git, a distributed version-control system for collaborative development of software, [89]):

Retrieve the AMT repository and clone the code using `git clone https://git.code.sf.net/p/amtoolbox/code amtoolbox-code`

- Create a branch named by your last name and the year of your contribution, e.g., `smith2021` for Ms. Smith contributing in 2021. The corresponding git command would be `git branch smith2021` in that example.
- Switch to that branch, e.g., `git checkout smith2021`
- Write the code and add all your files to the repository, `git add your_files`.
- Commit your changes to your local git repository and describe the changes.

Note that by using an own branch, no harm can be done to the "official" AMT code at all.

In the next step, the modifications can be provided to the AMT team by pushing the local branch to the AMT repository. To this end, the user needs to obtain write access to the repository which is provided by the AMT team after approaching us via email. We will include the user's SourceForge[14] account to the list of AMT developers, enabling the user to push the files to the online AMT repository. Then, the AMT team will review the users' contribution aiming at integrating that contribution in the main AMT code. For contributions written in programming languages other than Matlab or Octave, the AMT team will provide the means for their integration (e.g., see Sec. 2.2). In order to make the integration of changes as smooth as possible, we ask to follow the following requirements.

13 https://sourceforge.net/p/amtoolbox/code/
14 https://sourceforge.net/

4.2.1. Environment

The AMT is aimed to be compatible with five-years old Matlab versions, i.e., Matlab 2016a for the AMT 1.0. This needs to be considered when developing and integrating own functions. Further, we aim at using as few additional dependencies as possible. When proposing a new Matlab or third-party toolbox as a requirement, we ask to check whether the required functionality is provided with the AMT and/or currently available toolboxes. When proposing an additional third-party toolbox, we ask to check whether it is freely available and its license is GPL-compatible. We also ask to check for further dependencies of those toolboxes because third-party toolboxes need to be self-contained, i.e., not depending on further toolboxes and consist of code only (no data in the toolboxes). When adding a new third-party toolbox, we ask to locally store it in the directory `thirdparty` and notify us. We will then integrate its functionality and usage within the AMT environment. Note that it is not allowed to include third-party toolboxes in the AMT code repository.

4.2.2. Directory structure

The files are stored in directories reflecting the AMT environment (see Sec. 2). For the new contribution, the model implementation goes as a single function into the directory `models`. This model function can be complemented by (multiple) model stages stored in the directory `modelstages`. Model-specific data, plotting functions, and signal generators go to `data`, `plot` and `signals`, respectively. Model demonstrations and experiments reproducing results go to `demos` and `experiments`, respectively. Large chunks of data can be locally stored as auxiliary data in `auxdata`; they need to be accessed by `amt_load` and provided to the AMT team (link via email) after having the code submitted to the remote repository.[15] In demonstrations and experiments, if the processing duration is beyond a few minutes, we encourage to cache the results by integrating `amt_cache`.

4.2.3. Function and file names

Underscore is a reserved character in the AMT environment and used only to distinguish structural parts in the AMT. All function names are lowercase. This avoids a lot of confusion because 1) some computer architectures handle the casing with various relevance and 2) in Matlab or Octave documentation, function names are traditionally converted to uppercase. Function names indicate what they do, rather than the algorithm they use, or the person who programmed or invented it. For reasons pointed out earlier in Sec. 3, prominent exceptions represent the model functions and their demonstrations as well as experiments, which are named by the last name of the first author followed by the year of the corresponding publication.

If the new model consists of several functions, the model function contains the main functionality and the remaining parts are covered by model stages. Stand-alone model stages without a corresponding model are not allowed; a model stage needs a model. Local functions, i.e., functions within a file

15 Commit of data to the code repository is not allowed.

containing the main function, have the prefix `local_` in order to easily be distinguished from other AMT functions. The function `amt_disp` is used to display text in the command window, offering functionality such as volatile display of progress and pushing results to the online documentation. For the sake of simplicity, object-oriented programming and use of own classes are not recommended.

4.2.4. Variable names and default parameters

Within each function, variable names are allowed to be both lower and upper case, depending on the author's personal style. Use of global variables is not allowed because they make the code harder to debug and to parallelize. In matrices, the first dimension is time (or the sampled time interval).

For the handling of default and optional parameters, we use the functionality provided by the function `ltfatarghelper` from the LTFAT[16]. When creating a new model, it is required to store all model's default parameters in a separate file placed in the directory defaults and named by model's name with the prefix `arg_`. This file needs to be a function with `definput` as input and output, in which flags are stored in the structure `definput.flags` and key-value pairs are stored in the structure `definput.keyvals` – see arbitrary `arg_` function for more details. Then, in the model, for the parameter parsing, the function `ltfatarghelper()` from the LTFAT is used in the following form:

```
definput.import = {'model2021'}; % load defaults from arg_model2021
[f,kv] = ltfatarghelper({},definput,varargin);
```

resulting with flags and key-value pairs stored in the structures `f` and `kv`, respectively. Note the `varargin` input to the `ltfatarghelper`, which passes the optional parameters provided to a model to be processed. This way default parameters from the `arg_` function will be overwritten by optionally provided parameters to the model. While `ltfatarghelper` supports complex passing of parameters, in order to provide a clearly structured handling of the default parameters, we discourage from loading multiple default parameter files.

4.2.5. Signal levels

Auditory models can be nonlinear and the numeric representation of physical quantities like the sound pressure level (SPL) must be well-defined. In the AMT, an audio signal represents sound pressure in Pascal and is represented on the logarithmic dB scale *re* 20 µPa. Thus, an audio signal having a root-mean square (RMS) level of 1 (e.g., a square signal with the amplitude between -1 and 1, or a sine with the amplitude between $\pm\sqrt{2}$) corresponds to an SPL of 93.9794 dB. This level convention reflects the SI system and is the default level convention in the AMT[17]. However, AMT models have been developed with a variety of level conventions and a level conversion is sometimes required. To this end, the AMT common function `dbspl` calculates the SPL in dB of an audio signal considering

16 See http://ltfat.github.io/doc/base/ltfatarghelper.html

17 Note that this is a non-compatible change with respect to the AMT 0.x.

the AMT level convention. While this function is similar to the Matlab/Octave function `rms`, `dbspl` additionally converts to the logarithmic dB scale and considers the AMT level convention. In some cases, an audio signal needs to be scaled to a given SPL considering the AMT level convention. This can be done with the function `out=scaletodbspl(in,spl)`, which scales `in` such that the SPL of `out` is `spl`. Note that when a linear model such as the linear Gammatone filterbank is applied, the level convention can be ignored. Note further that because of historical reasons, previous AMT versions used a different level convention as the default.

While we are happy about contributions with perfectly structured code, clear parameter passing, and naming exactly following the AMT conventions, we encourage all programmers and researchers to contribute their code *as it is* and *as soon as possible*. "Publish your code: it's good enough" [90].

5. General information

5.1. Documentation

The AMT uses an in-code documentation system, i.e., the documentation text is embedded in the source-code files of the implementation. This way, a high level of integrity between the code and documentation can be provided and still a human-readable documentation can be generated. When releasing a new AMT version, this documentation is compiled by the compiler mat2doc[18] to an offline documentation available when calling help within the Matlab or Octave environment, e.g., `help exp_hassager2016`, and an online documentation published at the AMT documentation website.

The syntax of mat2doc is based on reStructuredText[19], a widely used markup syntax and parser component of Docutils[20]. The documentation based on reStructuredText can be compiled online.[21] In mat2doc, relevant differences to reStructuredText are 1) the comment character `%` in each line, 2) the first line representing a brief description of the function, e.g.; `%AMT_CACHE Caches variables`, according to Matlab tradition, and 3) in all other lines, applying the three-blank rule between `%` and the first letter of the text, e.g., `%   This is an example.`

In addition, mat2doc adds some environment-specific features triggered by keywords. The keyword `Usage:` appended by multiple lines shows how the corresponding function can be called, e.g., `% Usage:  amt_start;`. The keywords `Input parameters:` and `Output parameters:` appended by multiple lines can be used to explain the input and output parameters of a function. The keyword `References:` appended by a single-line list of BibTex[22] identifiers includes a list of refer-

18 http://mat2doc.sourceforge.net/
19 https://docutils.sourceforge.io/rst.html
20 https://docutils.sourceforge.io/index.html
21 http://rst.ninjs.org/
22 http://www.bibtex.org/

ences used to cite publications within the documentation. The corresponding references are stored in the file `project.bib` in the AMT directory `mat2doc`. The keyword `See also:` appended by a single-line list of other AMT functions includes a list of links to other relevant AMT functions. Note that these keywords must be used exactly as given, including the casing and colon.

Further, the in-code documentation supports the so-called anchors, which are keywords to provide additional and machine-readable information about the authors, requirements, the license, and model status (see `amt_info`). The anchors are encoded as `%  #Anchor:` (three blanks between `%` and `#`) and are provided in a comment block of the in-code documentation. The author information is encoded by the anchor `#Author` and can be provided in multiple lines (each of them beginning with `%  #Author:`), enabling various author contributions over the course of time. The information about requirements uses `#Requirements` and represents the required environment (either `Matlab` or `Octave`, never both), internal packages (`M-signal` and/or `M-statistics` for Matlab toolboxes), third-party toolboxes (`SOFA`, `SFS`, and/or `CircStat`), and additionally required environments (`Python`, `Binary`, and/or `MEX`). The anchor `#License` enables the multi-licensing feature of the AMT (see the following section). The model status is encoded by the anchors `#StatusDoc`, `#StatusCode`, and `#Verification` followed by the short name of the status as enlisted in Sec. 3.4.

5.2. Licensing

The code written by the AMT core team is licensed under the GNU general public license (GPL) version 3, which basically allows users to run, study, share, and modify the software. Also the code written by other contributors for the AMT is licensed under GPL. By committing code to the AMT repository, contributors agree to use that license and to transfer the ownership to the AMT team.

While the AMT generally follows the GPL, in order to include implementations not licensed under the GPL, the AMT also supports multiple licensing. To this end, model implementations provided by the researchers and integrated in the AMT can be licensed under a license different from the GPL. A researcher may choose a separate license regulating the usage of that model, and while the author grants the AMT team the permission to distribute the model to third parties without a prior written authorization, the model ownership remains with the author. This information is clearly described within the corresponding files.

Licensing also involves third-party toolboxes. For a clear legal integration, only toolboxes with licenses compatible for working with the GPL are used in the AMT. The license information is stored in the toolboxes' corresponding directories as provided by the toolbox authors. The ownership of third-party toolboxes remains with the toolbox authors.

For models deviating from GPL v3, at their first usage, the AMT displays a boilerplate, i.e., a brief note about the separate license and the most important terms such as terms of usage. That boilerplate

and the license type can also be displayed any time by executing `amt_info` with the model name as parameter. For example, a model can be restricted to be used in non-commercial applications only. In such a case, while the AMT is allowed to be used in commercial projects (as a consequence of the GPL), the user will be warned that by using that particular model, commercial usage is prohibited.

Note the difference between a model deviating from GPL v3 in the AMT and a third-party toolbox. A toolbox is integrated without any modification and the ownership remains with the authors. A model, even when integrated under a license different to GPL, requires code modifications. To this end, we seek permission from the researchers allowing us to edit and integrate their code. By having the model integrated, it has joint ownership (unless the researchers have transferred the ownership to us) and remains under the researcher's preferred license.

The license information is provided by the following mechanism: 1) The AMT directory `licenses` stores two plain-text files per license: the full license text and a license's boilerplate, named as `X_license.txt` and `X_boilerplate.txt`, respectively, with `X` being the short keyword describing the license; 2) File-specific information about the license by means of in-code documentation, with the license anchor followed by the license keyword (casing ignored). Files without the license information are licensed under the standard AMT license. For example, the `licenses` directory contains the files `ugent_license.txt` as well as `ugent_boilerplate.txt` and by using `%   License: UGent` in any file, that file will be licensed under the license of the University Gent.

5.3. Acknowledging researchers for their contribution

Development of new models is much work. And publishing the model implementation is easy via the internet. Thus, researchers may ask for the motivation to permit their work to be integrated in the AMT or even more, to even put more effort and integrate it in the AMT by themselves. To address this issue, the AMT provides a variety of ways to acknowledge researcher's work and display researcher's contribution.

First, the models are named after the last name of the first author of the publication describing the model, providing great visibility to the main author of the model. Second, the publication describing the model is clearly cited on the AMT website. This promotes the researcher and the publication beyond the journal publisher's common promoting channels. Further, the publication describing the model is cited in the AMT in-code documentation, which is visible in both the AMT online documentation and within the Matlab/Octave help system. Last but not least, the models integrated in the AMT are cited by publications describing the AMT. This is an important means of scientific recognition, emphasizing the significance of the researcher's contribution to a better understanding of the auditory system.

Not only model developers, but also researchers applying AMT models in their publications can provide experiments reproducing their results to the AMT. The contribution of these researchers will be visible by having their own `exp_` function named by their last name and with a reference to their publication, acknowledging their effort of contributing to the AMT.

Finally, even programmers, neither contributing a full model nor an experiment, but improving the AMT code, get acknowledged by noting their names in the sources and online code repository. The AMT website lists all AMT contributors so far.

6. Conclusions

The AMT 1.0 implements over 50 auditory models and integrates eleven publication-specific and 29 general datasets. It is available (with all the required third-party toolboxes) from SourceForge[23] as a free and open-source software package for Matlab and Octave. Most of the AMT's models and data are well-documented and verified, as reflected in the model status of the AMT's documentation web page[24]. The models are accompanied by "demonstrations" providing a simple access to a model's implementation and "experiments" aiming at reproducing the models published output. An online data repository helps to keep the AMT compact, while still having access to all data required to reproduce each model's output. The open-source code repository combined with a comprehensive documentation system, multi-licensing, and contribution reward aims at helping others to contribute to the AMT at a low entry threshold.

With the release of AMT 1.0, the AMT has matured to a large collection of auditory models. It now includes new models such as those based on Bayesian inference, statistical signal processing, and predicting speech intelligibility. The AMT would not be that comprehensive without the many contributions from various researchers from the auditory community, for which we are wholeheartedly grateful.

By integrating comprehensive monaural processing stages with models of binaural and spatial hearing, the AMT paves the road towards more complex cognitive auditory models. Researchers from the auditory cognitive sciences are invited to pick them up and extend them towards more encompassing models of auditory or multimodal cognition.

7. References

1. Frigg, R., and Hartmann, S. (**2012**). "Models in Science," In E. N. Zalta (Ed.), Stanf. Encycl. Philos. Fall 2012.

2. Meddis, R., Lopez-Poveda, E., Fay, R. R., and Popper, A. N. (Eds.) (**2010**). *Computational Models of the Au-*

23 https://sourceforge.net/projects/amtoolbox

24 http://amtoolbox.org/

ditory System, Springer Handbook of Auditory Research Springer US.

3. Jasny, B. R., Chin, G., Chong, L., and Vignieri, S. (**2011**). "Again, and Again, and Again …," Science **334**, 1225–1225.

4. Vandewalle, P., Kovacević, J., and Vetterli, M. (**2009**). "Reproducible research in signal processing: What, why, and how," IEEE Signal Process. Mag. **26**, 37–47.

5. Schwab, M., Karrenbach, N., and Claerbout, J. (**2000**). "Making scientific computations reproducible," Comput. Sci. Eng. **2**, 61–67. Presented at the Computing in Science Engineering.

6. Mesirov, J. P. (**2010**). "COMPUTER SCIENCE Accessible Reproducible Research," Science , doi: 10.1126/science.1179653.

7. Peng, R. D. (**2011**). "Reproducible Research in Computational Science," Science **334**, 1226–1227.

8. Claerbout, J. F., and Karrenbach, M. (**1992**). "Electronic documents give reproducible research a new meaning," SEG Tech. Program Expand. Abstr. 1992, SEG Technical Program Expanded Abstracts Society of Exploration Geophysicists, Vols. 1-0, pp. 601–604.

9. Peterson, B. E., Healy, M. D., Nadkarni, P. M., Miller, P. L., and Shepherd, G. M. (**1996**). "ModelDB: an environment for running and storing computational models and their results applied to neuroscience," J. Am. Med. Inform. Assoc. JAMIA **3**, 389–398.

10. Morse, T. M. (**2007**). "Model sharing in computational neuroscience," Scholarpedia **2**, 3036.

11. Lyon, R. (**2011**). "Cascades of two-pole–two-zero asymmetric resonators are good models of peripheral auditory function," J Acoust Soc Am **130**, 3893–3904.

12. Carney, L. H., Li, T., and McDonough, J. M. (**2015**). "Speech Coding in the Brain: Representation of Vowel Formants by Midbrain Neurons Tuned to Sound Fluctuations,," eNeuro , doi: 10.1523/ENEURO.0004-15.2015.

13. Verhulst, S., Altoè, A., and Vasilkov, V. (**2018**). "Computational modeling of the human auditory periphery: Auditory-nerve responses, evoked potentials and hearing loss," Hear. Res., Computational models of the auditory system **360**, 55–75.

14. Bruce, I. C., Erfani, Y., and Zilany, M. S. A. (**2018**). "A phenomenological model of the synapse between the inner hair cell and auditory nerve: Implications of limited neurotransmitter release sites," Hear. Res., Computational models of the auditory system **360**, 40–54.

15. Ru, P. (**2001**). *Multiscale Multirate Spectro-Temporal Auditory Model* (PhD Thesis), University of Maryland College Park.

16. Moore, B. C. J. (**2014**). "Development and Current Status of the 'Cambridge' Loudness Models," Trends Hear. **18**, 2331216514550620.

17. Higham, D. J., and Higham, N. J. (**2016**). *MATLAB guide*, Siam, Vol. 150.

18. Eaton, J. W., Bateman, D., and Hauberg, S. (**2002**). *GNU Octave Manual*, Network Theory, Ltd.

19. Malcolm Slaney (**1998**). *Auditory Toolbox: A MATLAB Toolbox for Auditory Modeling Work* (No. #1998-010), Interval Research Corporation.

20. Patterson, R. D., Allerhand, M. H., and Giguère, C. (**1995**). "Time-domain modeling of peripheral auditory processing: a modular architecture and a software platform," J. Acoust. Soc. Am. **98**, 1890–1894.

21. Härmä, A., and Palomäki, K. (**1999**). "HUTear - A Free Matlab Toolbox for Modeling of Human Auditory

System," Presented at the Matlab DSP Conference, 96–99.

22. Mountain, D. C., Anderson, D. A., Bresnahan, G. J., Deligeorges, S. G., Hubbard, A. E., and Vajda, V. (**2006**). "EarLab: A modular approach to auditory simulation," J. Biomech., Abstracts of the 5th World Congress of Biomechanics **39**, S434.

23. Rudnicki, M., Schoppe, O., Isik, M., Völk, F., and Hemmert, W. (**2015**). "Modeling auditory coding: from sound to spikes," Cell Tissue Res. **361**, 159–175.

24. O'Mard, L. P. (**2012**). *Development System for Auditory Modelling (DSAM)*, Centre for the Neural Basis of Hearing (CNBH).

25. Fontaine, B., Goodman, D. F. M., Benichoux, V., and Brette, R. (**2011**). "Brian Hears: Online Auditory Processing Using Vectorization Over Channels," Front. Neuroinformatics , doi: 10.3389/fninf.2011.00009.

26. Stimberg, M., Brette, R., and Goodman, D. F. (**2019**). "Brian 2, an intuitive and efficient neural simulator," eLife **8**, e47314.

27. Gutkin, A. (**2020**). "Eidos: An Open-Source Auditory Periphery Modeling Toolkit and Evaluation of Cross-Lingual Phonemic Contrasts," Proc. 1st Jt. Workshop Spok. Lang. Technol. -Resour. Lang. SLTU Collab. Comput. -Resour. Lang. CCURL European Language Resources association, Marseille, France, 9–20.

28. Two!Ears Team (**2018**). *Two!Ears Auditory Model 1.5*, Zenodo.

29. Søndergaard, P., and Majdak, P. (**2013**). "The Auditory Modeling Toolbox," In J. Blauert (Ed.), Technol. Binaural List. Springer, Berlin-Heidelberg, Germany, pp. 33–56.

30. Eaton, J. W., Bateman, D., Hauberg, S., and Wehbring, R. (**2016**). *GNU Octave version 4.2.0 manual: a high-level interactive language for numerica l computations*,.

31. Breebaart, J., van de Par, S., and Kohlrausch, A. (**2001**). "Binaural processing model based on contralateral inhibition III Dependence on temporal parameters," J Acoust Soc Am **110**, 1105–17.

32. Dietz, M., Lestang, J.-H., Majdak, P., Stern, R. M., Marquardt, T., Ewert, S. D., Hartmann, W. M., et al. (**2018**). "A framework for testing and comparing binaural models," Hear. Res. **360**, 92–106.

33. Majdak, P., Carpentier, T., Nicol, R., Roginska, A., Suzuki, Y., Watanabe, K., Wierstorf, H., et al. (**2013**). "Spatially Oriented Format for Acoustics: A Data Exchange Format Representing Head-Related Transfer Functions," Proc. 134th Conv. Audio Eng. Soc. AES Roma, Italy, Convention Paper 8880.

34. Wierstorf, H., Raake, A., and Spors, S. (**2013**). "Binaural Assessment of Multichannel Reproduction," In J. Blauert (Ed.), Technol. Binaural List. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 255–278.

35. Lindemann, W. (**1986**). "Extension of a binaural cross-correlation model by contralateral inhibition I Simulation of lateralization for stationary signals," J. Acoust. Soc. Am. **80**, 1608–1622.

36. Majdak, P., Goupell, M. J., and Laback, B. (**2010**). "3-D localization of virtual sound sources: effects of visual environment, pointing method, and training," Atten. Percept. Psychophys. **72**, 454–69.

37. Møller, H., Sørensen, M. F., Hammershøi, D., and Jensen, C. B. (**1995**). "Head-related transfer functions of human subjects," J Audio Eng Soc **43**, 300–321.

38. Bernstein, L. R., van de Par, S., and Trahiotis, C. (**1999**). "The normalized interaural correlation: Accounting for NoSπ thresholds obtained with Gaussian and \"low-noise\" masking noise," J. Acoust. Soc. Am. **106**, 870.

39. Baumgartner, R., Majdak, P., and Laback, B. (**2016**). "Modeling the Effects of Sensorineural Hearing Loss

on Sound Localization in the Median Plane," Trends Hear. **20**, 2331216516662003.

40. Ziegelwanger, H., and Majdak, P. (**2014**). "Modeling the direction-continuous time-of-arrival in head-related transfer functions," J. Acoust. Soc. Am. **135**, 1278–93.

41. Lopez-Poveda, E. A., and Meddis, R. (**2001**). "A human nonlinear cochlear filterbank," J Acoust Soc Am **110**, 3107–18.

42. Hohmann, V. (**2002**). "Frequency analysis and synthesis using a Gammatone filterbank," Acta Acust. United Acust. **88**, 433–442.

43. Verhulst, S., Dau, T., and Shera, C. (**2012**). "Nonlinear time-domain cochlear model for transient stimulation and human otoacoustic emission," J. Acoust. Soc. Am. **132**, 3842–3848.

44. Zilany, M. S. A., and Bruce, I. C. (**2007**). "Representation of the vowel $/\epsilon/$ in normal and impaired auditory nerve fibers: Model predictions of responses in cats," jasa **122**, 402–248.

45. Zilany, M. S. A., Bruce, I. C., and Carney, L. H. (**2014**). "Updated parameters and expanded simulation options for a model of the auditory periphery," J. Acoust. Soc. Am. **135**, 283–286.

46. Dau, T., Püschel, D., and Kohlrausch, A. (**1996**). "A quantitative model of the 'effective' signal processing in the auditory system I Model structure," J Acoust Soc Am **99**, 3615–22.

47. Dau, T., Kollmeier, B., and Kohlrausch, A. (**1997**). "Modeling auditory processing of amplitude modulation I Detection and masking with narrow-band carriers," J. Acoust. Soc. Am. **102**, 2892–2905.

48. Relaño-Iborra, H., Zaar, J., and Dau, T. (**2019**). "A speech-based computational auditory signal processing and perception model," J. Acoust. Soc. Am. **146**, 3306–3317.

49. Jepsen, M. L., Ewert, S. D., and Dau, T. (**2008**). "A computational model of human auditory signal processing and perception," J Acoust Soc Am **124**, 422–38.

50. Ewert, S., and Dau, T. (**2000**). "Characterizing frequency selectivity for envelope fluctuations," J Acoust Soc Am **108**, 1181–1196.

51. Viemeister, N. F. (**1979**). "Temporal modulation transfer functions based upon modulation thresholds," J. Acoust. Soc. Am. **66**, 1364–1380.

52. Rønne, F. M., Dau, T., Harte, J., and Elberling, C. (**2012**). "Modeling auditory evoked brainstem responses to transient stimuli," J. Acoust. Soc. Am. **131**, 3903–3913.

53. Verhulst, S., Bharadwaj, H. M., Mehraei, G., Shera, C. A., and Shinn-Cunningham, B. G. (**2015**). "Functional modeling of the human auditory brainstem response to broadband stimulation," J. Acoust. Soc. Am. **138**, 1637–1659.

54. King, A., Varnet, L., and Lorenzi, C. (**2019**). "Accounting for masking of frequency modulation by amplitude modulation with the modulation filter-bank concept," J. Acoust. Soc. Am. **145**, 2277–2293.

55. Dietz, M., Ewert, S. D., and Hohmann, V. (**2011**). "Auditory model based direction estimation of concurrent speakers from binaural signals," Speech Commun. **53**, 592–605.

56. Takanen, M., Santala, O., and Pulkki, V. (**2013**). "Binaural assessment of parametrically coded spatial audio signals," In J. Blauert (Ed.), Technol. Binaural List. Springer, Berlin, Germany, pp. 333–358.

57. Moore, B. C. J., Glasberg, B. R., and Baer, T. (**1997**). "A Model for the Prediction of Thresholds, Loudness, and Partial Loudness," J Audio Eng Soc **45**, 224–240.

58. Glasberg, B. R., and Moore, B. C. J. (**2002**). "A Model of Loudness Applicable to Time-Varying Sounds," J

Audio Eng Soc **50**, 331–342.

59. Chen, Z., Hu, G., Glasberg, B. R., and Moore, B. C. J. (**2011**). “A new model for calculating auditory excitation patterns and loudness for cases of cochlear hearing loss,” Hear. Res. **282**, 69–80.

60. Moore, B. C. J., Glasberg, B. R., Varathanathan, A., and Schlittenlacher, J. (**2016**). “A Loudness Model for Time-Varying Sounds Incorporating Binaural Inhibition,” Trends Hear. , doi: 10.1177/2331216516682698.

61. Jørgensen, S., and Dau, T. (**2011**). “Predicting speech intelligibility based on the signal-to-noise envelope power ratio after modulation-frequency selective processing,” J. Acoust. Soc. Am. **130**, 1475–1487.

62. Taal, C. H., Hendriks, R. C., Heusdens, R., and Jensen, J. (**2011**). “An Algorithm for Intelligibility Prediction of Time–Frequency Weighted Noisy Speech,” IEEE Trans. Audio Speech Lang. Process. **19**, 2125–2136. Presented at the IEEE Transactions on Audio, Speech, and Language Processing.

63. Jørgensen, S., Ewert, S. D., and Dau, T. (**2013**). “A multi-resolution envelope-power based model for speech intelligibility,” J. Acoust. Soc. Am. **134**, 436–446.

64. Culling, J. F., Hawley, M. L., and Litovsky, R. Y. (**2004**). “The role of head-induced interaural time and level differences in the speech reception threshold for multiple interfering sound sources,” J Acoust Soc Am **116**, 1057–65.

65. Jelfs, S., Culling, J. F., and Lavandier, M. (**2011**). “Revision and validation of a binaural model for speech intelligibility in noise,” Hear. Res. , doi: 10.1016/j.heares.2010.12.005.

66. Hauth, C. F., Berning, S. C., Kollmeier, B., and Brand, T. (**2020**). “Modeling Binaural Unmasking of Speech Using a Blind Binaural Processing Stage,” Trends Hear. **24**, 2331216520975630.

67. Osses, A., and Kohlrausch, A. (**2021**). “Perceptual similarity between piano notes: Simulations with a template-based perception model,” J. Acoust. Soc. Am.

68. McKenzie, T., Armstrong, C., Ward, L., Murphy, D., and Kearney, G. (**2021**). “A Perceptually Motivated Spectral Difference Model for Binaural Signals,” Acta Acust.

69. Zakarauskas, P., and Cynader, M. S. (**1993**). “A computational theory of spectral cue localization,” J Acoust Soc Am **94**, 1323–1331.

70. Langendijk, E. H. A., and Bronkhorst, A. W. (**2002**). “Contribution of spectral cues to human sound localization,” J Acoust Soc Am **112**, 1583–96.

71. May, T., van de Par, S., and Kohlrausch, A. (**2011**). “A probabilistic model for robust localization based on a binaural auditory front-end,” IEEE Trans Audio Speech Lang Proc **19**, 1–13.

72. Baumgartner, R., Majdak, P., and Bernhard, L. (**2013**). “Assessment of sagittal-plane sound localization performance in spatial-audio applications,” In J. Blauert (Ed.), Technol. Binaural List. Springer, Berlin, Heidelberg, pp. 93–119.

73. Georganti, E., May, T., van de Par, S., and Mourjopoulos, J. (**2013**). “Sound Source Distance Estimation in Rooms based on Statistical Properties of Binaural Signals,” Audio Speech Lang. Process. IEEE Trans. On **21**, 1727–1741.

74. Baumgartner, R., Majdak, P., and Laback, B. (**2014**). “Modeling sound-source localization in sagittal planes for human listeners,” J. Acoust. Soc. Am. **136**, 791–802.

75. Reijniers, J., Vanderelst, D., Jin, C., Carlile, S., and Peremans, H. (**2014**). “An ideal-observer model of human sound localization,” Biol. Cybern. **108**, 169–181.

76. Kelvasa, D., and Dietz, M. (**2015**). “Auditory Model-Based Sound Direction Estimation With Bilateral Cochlear Implants,” Trends Hear. , doi: 10.1177/2331216515616378.

77. Hassager, H. G., Gran, F., and Dau, T. (**2016**). “The role of spectral detail in the binaural transfer function on perceived externalization in a reverberant environment,” J. Acoust. Soc. Am. **139**, 2992–3000.

78. Baumgartner, R., Reed, D. K., Tóth, B., Best, V., Majdak, P., Colburn, H. S., and Shinn-Cunningham, B. (**2017**). “Asymmetries in behavioral and neural responses to spectral cues demonstrate the generality of auditory looming bias,” Proc. Natl. Acad. Sci. **114**, 9743–9748.

79. Li, S., Baumgartner, R., and Peissig, J. (**2020**). “Modeling perceived externalization of a static, lateral sound image,” Acta Acust. **4**, 21.

80. Baumgartner, R., and Majdak, P. (**2021**). “Auditory cue combination in perceptual externalization of static sound sources: a model-based meta analysis,” Acta Acust.

81. Barumerli, R., Majdak, P., Baumgartner, R., Geronazzo, M., and Avenzini, F. (**2021**). “Predicting human spherical sound-source localization based on Bayesian inference,” Acta Acust.

82. McLachlan, G., Majdak, P., Reijniers, J., and Peremans, H. (**2021**). “Towards modelling active dynamic sound localisation based on Bayesian inference,” Acta Acust.

83. Hartmann, W. M., and Wittenberg, A. (**1996**). “On the externalization of sound images,” J Acoust Soc Am **99**, 3678–88.

84. Baumgartner, R., and Majdak, P. (**2015**). “Modeling Localization of Amplitude-Panned Virtual Sources in Sagittal Planes,” J Audio Eng Soc **63**, 562–569.

85. Baumgartner, R., Majdak, P., and Laback, B. (**2015**). “The Reliability of Contralateral Spectral Cues for Sound Localization in Sagittal Planes,” Presented at the Midwinter Meeting of the Association for Research in Otolaryngology, Baltimore, MD, USA.

86. Engel, I., and Picinali, L. (**2021**). “Perceptual models as a mean of assessment of Ambisonics-based binaural rendering methods,” Acta Acust.

87. Röttges, S., Hauth, C. F., Brand, T., and Rennies-Hochmuth, J. (**2021**). “Challenging a non-intrusive EC-mechanism: Modelling the Interaction between binaural and temporal speech processing,” Acta Acust.

88. Steidle, L., and Baumgartner, R. (**2019**). “Geometrical Evaluation of Methods to approximate Interaural Time Differences by Broadband Delays,” Fortschritte Akust. Rostock, 368–370.

89. Chacon, S., and Straub, B. (**2014**). *Pro git*, Apress.

90. Barnes, N. (**2010**). “Publish your computer code: it is good enough,” Nature **467**, 753.

706