Acta Acustica 2022, **6**, 19

https://doi.org/10.1051/aacus/2022011

Available online at:
https://acta-acustica.edpsciences.org

Topical Issue - Auditory models: from binaural processing to multimodal cognition

TECHNICAL & APPLIED ARTICLE

OPEN ACCESS

AMT 1.x: A toolbox for reproducible research in auditory modeling

Piotr Majdak*, Clara Hollomey, and Robert Baumgartner

Austrian Academy of Sciences, Acoustics Research Institute, Vienna, Austria

Received 31 May 2021, Accepted 18 March 2022

Abstract – The Auditory Modeling Toolbox (AMT) is a MATLAB/Octave toolbox for the development and application of computational auditory models with a particular focus on binaural hearing. The AMT aims for a consistent implementation of auditory models, well-structured in-code documentation, and inclusion of auditory data required to run the models. The motivation is to provide a toolbox able to reproduce the model predictions and allowing students and researchers to work with and to advance existing models. In the AMT, 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. Here, we describe the tools and mechanisms available within the framework of all AMT 1.x versions. The recently released AMT 1.1 includes over 60 models and is freely available as an open-source package from https://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 form a powerful tool to formalize hypotheses, generate testable predictions, reproduce results, and confirm conclusions [3]. In this article, we describe the Auditory Modeling Toolbox (AMT), which is a framework for computational models of the human hearing system emphasizing reproducible research.

The motivation for the development of auditory models is widely spread across different domains and applications. Auditory models can help to evaluate how a deficit in one or more parts of the hearing system affects its overall operation. They can help to prototype novel hearing-related algorithms in early prototyping stages, effectively reducing the amount of listening tests. By doing so, they facilitate advances 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]. In some cases, such a family tree is clearly evident. For example, building upon a peripheral model for the processing of amplitude modulation [5], a model characterizing the frequency selectivity for envelope fluctuations has been developed [6], which then led to the development of an envelope-power based model for speech intelligibility [7].

Unfortunately, when a new model is published in the usual form of a journal article, the model description and the discussion about its properties are 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 sometimes been described as "advertisement of scholarship" [8] whereas the scholarship itself is then represented by computer-executable code, input data, and parameters [9]. Hence, the spectrum of reproducibility is wide [10] – model implementations linked with the corresponding publication and data are vital prerequisites for a full replication of the research [11]. The AMT directly addresses the full research replication by providing executable code and data, both linked to an extensive documentation referencing the corresponding publications (see Sect. 5.2). Further, the AMT facilitates publishing of researchers' models combined with the implementation, improving their chances of generating impact within the community (see Sect. 5.4).

*Corresponding author: piotr.majdak@oeaw.ac.at

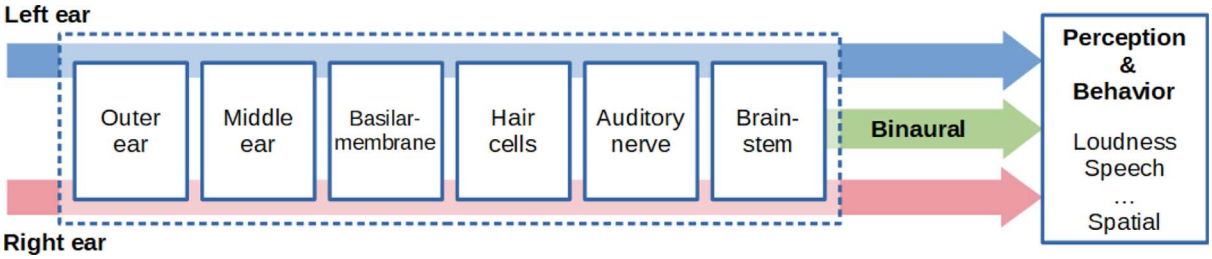


Figure 1. Functional structure of the AMT with stages reflecting the monaural processing stages of the auditory periphery (Left ear, Right ear), followed by an optional stage of binaural interaction (Binaural) and stages modeling perceptual or even behavioral outcomes (Perception & Behavior).

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 biological 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 Figure 1: The sound entering and filtered by the outer ear is transmitted by the middle ear, frequency-decoded in the cochlea, neurally encoded by the 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 databases listing computational neuroscience models [12], lists 65 implementations for the search word "auditory". While model sharing is common in the neuroscience community [13], most of the implementations focus on specific properties or stages of the auditory system, such as modeling the frequency selectivity in cochlear processing [14] or brainstem activities [15][1], respectively. Other types of implementations simulate multiple parts of the auditory pathway such as processing up to the auditory nerve [16][2], [17][3], auditory-cortical processing [18][4], or everything relevant to a specific percept such as loudness [19][5].

In contrast, the AMT is a *collection* of auditory models. In contrast to a database, it provides executable implementations of the models in a single software package. It is implemented within the environment of MATLAB [20] and Octave [21]. 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 [22]. It is written for MATLAB, but the development seems to have stopped in 1993. AIM [23] 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 for the offline version and until 2013 for the real-time version. Other notable but stagnated collections of auditory models are HUTear [24] (developed until 2000), SOMA (until 2011)[6], EarLab [25] (until 2016), and Cochlea [26] (until 2017). Development System for Auditory Modelling (DSAM) is a potentially still maintained model collection that includes multiple auditory nerve models, neural cell models, and utilities [27]. 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 [28], 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 [29]. 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 at being a one-stop shop for auditory models within multiple programming environments (see Sect. 3.2) and offered under multiple licenses (see Sect. 5.3).

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 [30], Faame[9] that targets model evaluation [31], or Two!Ears that targets the simulation of robotic hearing including basic cognitive functions [32]. As an interesting side note, the Two!Ears toolbox is based on an early version of the AMT, demonstrating the importance of freely

[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.
[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://www.faame4u.com.

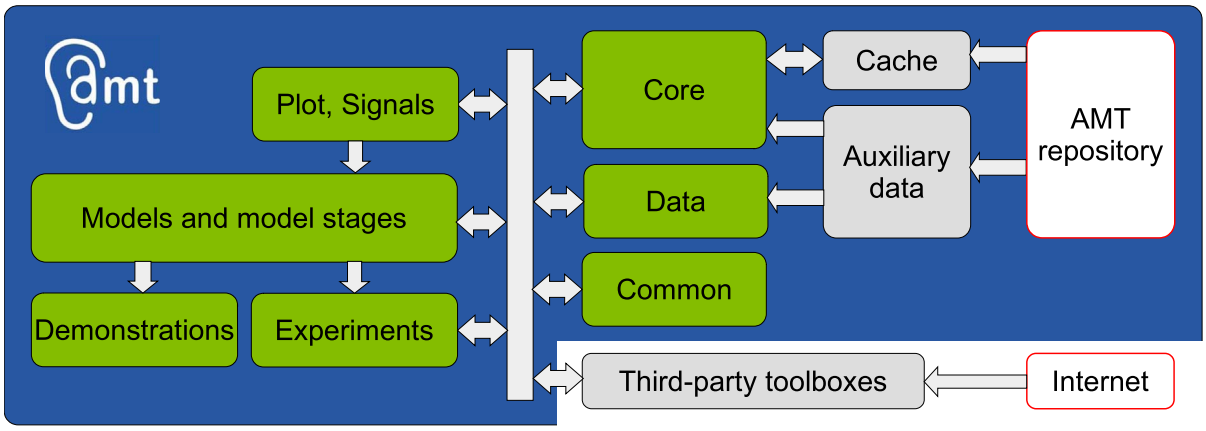


Figure 2. Logical structure of the AMT 1.x. *Green*: AMT's code consisting of functions and algorithms. *Grey*: Data and third-party toolboxes. *White*: Information available online for download. *Left side* (Plot, Signals, Models and model stages, Demonstrations, and Experiments): model-dependent AMT parts discussed in Section 2. *Right side* (Core, Data, Common, Cache, Auxiliary data, Third-party toolboxes): AMT's environment described in Section 3.

available and reproducible implementations of auditory models. On the other hand, the AMT also gains from such freely available model implementations, some of which have been integrated in the AMT. 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 [33] and released between 2013 and 2020), the AMT has matured to a large collection of auditory models.

However, the AMT 1.x is more than a collection of auditory model implementations – it is rather a framework providing sophisticated mechanisms supporting the models and the users. Figure 2 shows the logical structure of the AMT 1.x, in which we distinguish between the code, the data, and further information available online for download. The AMT 1.x integrates third-party toolboxes, mostly to provide some basic functionalities and to accommodate individual model requirements. Further, the AMT 1.x provides model-independent resources such as core and common functions, a caching mechanism, and access to auxiliary data with an online data repository. The auditory models can be accessed directly as functions, and their work can also be reproduced in the so-called experiments and demonstrated in the so-called demonstrations.

The recent releases of the AMT 1.0 and 1.1 (June and December 2021, respectively) offer new features and important structural changes. As compared to the description of the AMT versions 0.x [33], in this article, we focus on the new features of the AMT[10], the general AMT mechanics, and provide comprehensive descriptions from the perspectives of the AMT users and contributors.

[10] Following the semantic versioning (https://semver.org/), all AMT 1.x versions are aimed to be backwards compatible within the major version number. Thus, while in this article, we describe the AMT as of the version 1.1, our description applies to all AMT 1.x versions, to which, for the sake of clarity, we refer to as the AMT (see also the section "Data availability statement").

In the next section, we briefly enlist the available models (as of the AMT 1.1) and describe model-dependent AMT parts, including the system used to track the quality of model implementations. Then, we describe the AMT environment in greater detail, providing general information about the AMT such as the documentation system. Finally, we provide some practical tips for getting started with and contributing to the AMT.

2 Models

Within the AMT, auditory models are implemented as model functions and have their associated model stages. They are accompanied by model-specific plotting functions, signal generators, demonstrations, and experiments for the reproduction of published results. The status of each model implementation is tracked in order to provide an estimation of the quality of the integrated models. In this section, we describe the models, model stages, demonstrations, and experiments as released in AMT version 1.1. The model-dependent parts of the AMT structure are shown in the left part of Figure 2.

2.1 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. There are many models implemented in the AMT and the list is dynamic and growing. In this manuscript, we focus on the mechanisms behind the model implementations and only briefly describe the model functionalities. A complete list of models can be found at the AMT website. A detailed description of each model's functionalities and properties can be found in the corresponding publications.

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 Dietz et al. (2011) [34]. 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 year distinguishing e.g. `vicente2020` from `vicente2020nh`[11]. While a model's input parameters heavily depend on the model, the first input parameter is always the auditory signal and in matrices, the first dimension is time (or the sampled time interval).

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 1.x. Model stages are stored in the directory command "`model-stages`", 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`[12].

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 Sect. 3.5). In the following, we briefly describe models and common functions according to the functional structure shown in Figure 1.

2.1.1 Outer and middle ear

The processing of the outer ear is supported by the common functions `headphonefilter` and `hrtf2dtf`, combined with model-dependent head-related transfer functions (HRTFs) and the corresponding third-party toolbox 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` [35] for their geometrical consistency. The common function `itd2angle` further provides a simple solution to create a relationship between the processed binaural timing cues and sound incidence angle. The transmission properties of the middle ear can be modeled by the common function `middleearfilter`.

2.1.2 Basilar membrane and hair cells

Signal processing approaches approximating cochlear processing consist of modeling the basilar membrane excitation and the subsequent transmission by inner hair cells. Modeling the basilar membrane velocity as a function of frequency can be done by `lopezpoveda2001` [36], `hohmann2002` [37], `lyon2011` [14], and `verhulst2012` [38], 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.

2.1.3 Auditory nerve

The auditory nerve (AN) is often linked to the models of the basilar membrane and hair cells. Hence, the models `zilany2007` [39], `zilany2014` [40], and `bruce2018` [17] 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` [41], which can be parametrized to simulate non-linear AN properties by various approaches [5, 41–44].

2.1.4 Brainstem

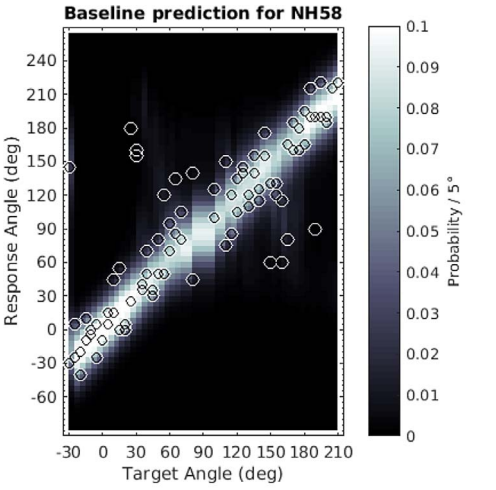
An important property of the neural auditory pathway from the cochlear nucleus to the inferior colliculus is the sensitivity to temporal modulations, which is supported by the models `ewert2000` [6] and `carney2015` [15] as well as by the `modulationfilterbank` [5] based on the temporal modulation transfer functions, both implemented [45] in the common function `modfilterbank`. Models that also integrate the more peripheral stages with some processing of higher neural stages are `dau1996` [41], `dau1997` [5], `roenne2012` [46], `verhulst2015` [47], `verhulst2018` [16], `relanoiborra2019` [43], and `king2019` [48]. Note that while some models output individual neural spikes, others output spike rates or even more abstract measures of neural activities.

Binaural processing is supported by the models `lindemann1986` [49], `breebaart2001` [42], `dietz2011` [34], and `takanen2013` [50].

2.1.5 Perception and behavior

The last section involves models describing various aspects of perception. In the AMT 1.1, we have loudness models represented by `moore1997` [52], `glasberg2002` [53], and `chen2011` [54] as well as a loudness model considering binaural inhibition, `moore2016` [55]. We have monaural speech perception models such as `joergensen2011` [56], `taal2011` [57], and `joergensen2013` [7], and models considering binaural speech processing such as `culling2004` [58], `jelfs2011` [59], `leclere2015` [60], `hauth2020` [61], `prudhomme2020` [62], `vicente2020nh` [63], `vicente2020` [64], and `lavandier2022` [65]. For perceptual similarity, we have `osses2021` [66] as a monaural model and `llado2022` [67] as a binaural model considering the effect of head-worn devices. Last but not least, we have many models of spatial perception: `zakarauskas1993` [68], `langendijk2002` [69], `may2011` [70], `baumgartner2013` [71], `georganti2013` [72], `wierstorf2013` [73], `baumgartner2014` [74], `reijniers2014` [75], `kelvasa2015` [76], `baumgartner`

[11] Note the lack of underscore between the year and the postfix showing that `vincent2020nh` is not part of `vincent2020`.
[12] Note the underscore between the year and the postfix showing that `interauralfunctions` is part of `dietz2011`.




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

Figure 3. Output generated by the AMT function `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 [95] localizing noise bursts presented from various target angles along the median plane. Brightness encodes the probability to respond at an angle as predicted by `baumgartner2014` [74]. *Text*: Predicted metrics of localization performance for that listener

`2016` [34], `hassager2016` [77], `baumgartner2017` [78], `li2020` [79], `baumgartner2021` [80], `barumerli 2021`[13] [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.

2.2 Demonstrations and experiments

Demonstrations are scripts which can be run without any parameters in order to demonstrate the functionality of a model or data set. Demonstrations have the prefix `demo_` and are stored in the directory demos. Demonstrations provide a visual representation of a model output. Figure 3 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 insights 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 those from the corresponding article. By visually comparing the experiment output, the quality of the model can be estimated (see Sect. 2.4). Figure 4 shows an example of the output of such an experiment function, along with the actual graphic from the original publication. Experiments are functions (not scripts) with the prefix `exp_` followed by the last name of the first author and the year of the publication reporting the simulation results. Most of the AMT experiments are those replicating the outcome of a publication describing a model; e.g., in the AMT 1.1, `exp_li2020` replicates figures from Li et al. (2020) [79] obtained by running the model `li2020`. A parameter with the prefix `fig` is used to reproduce a figure; e.g., `exp_li2020('fig2')` reproduces Figure 2 from Li et al. (2020) [79] as shown in our Figure 4. Similarly, a parameter with the prefix `tab` is used to reproduce the results of a table, e.g., `exp_baumgartner2014 ('tab2')` reproduces Table 2 from Baumgartner et al. (2014) [74].

In addition to the experiments directly replicating the original model publication, applications of a model can be implemented as additional experiments. If such an application is published in a secondary publication, in the AMT, the naming convention is to use the naming of the secondary publication. In the AMT 1.1, we have a few of such cases: `exp_baumgartner2015` [83] and `exp_baumgartner 2015binweight` [84], both applying `baumgartner2014` [74]; `exp_engel2021`[14] [85] applying three models, namely `jelfs2011`, `reijniers2014` and `baumgartner 2021`; `exp_osses2022` [86] applying `daul997`, `verhulst2015`, `verhulst2018`, `bruce2018`, `king 2019`, `relanoiborra2019`, and `osses2021`; `exp_roettges2022` [87] applying `hauth2020`; as well as `exp_steidle2019` [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.

2.3 Model-dependent plotting and signal generation

While MATLAB and Octave provide a variety of functions for signal generation and plotting, the AMT provides functions specifically supporting the requirements 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 file names, we follow the same pattern as for other parts of the 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

[13] Will be replaced by `barumerli2022` in future releases to reflect the actual publication year of the article.

[14] Will be replaced by `exp_engel2022` in future releases to reflect the actual publication year of the article.

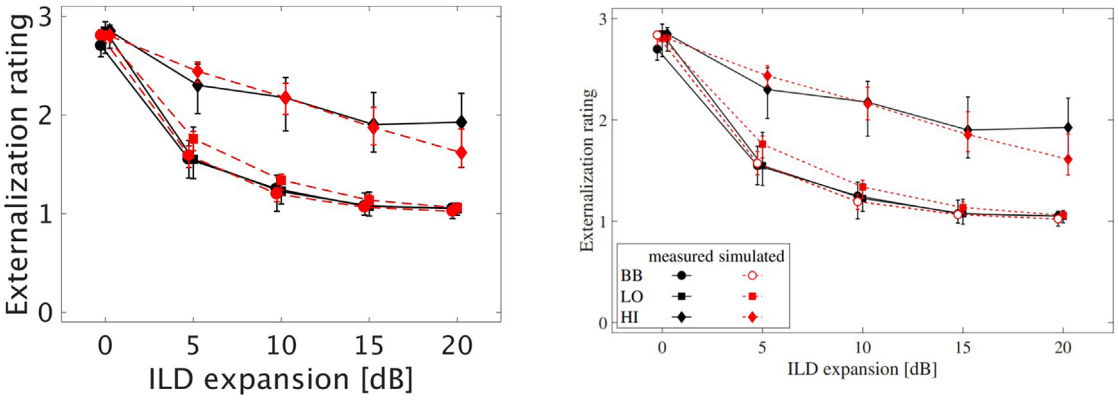


Figure 4. *Left*: Output generated by the AMT function `exp_li2020('fig2');`, reproducing Figure 2 from Li et al., (2020) [79] showing the actual and predicted externalization rating as a function of ILD variation. *Right*: The original Figure 2 from Li et al. (2020) [79].

tested in Hartmann and Wittenberg (1996) [89]. Functions that 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.

2.4 Model status

The AMT team tracks the status of each model implementation in order to provide an estimation of the quality of all model implementations available. The model status describes the quality of the model's source code and documentation, as well as its 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):

- *Perfect (A)*: The code/documentation fully complies with the AMT conventions; there are no open issues.
- *Good (B)*: The code/documentation follows the AMT conventions, but there are still open issues.
- *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.
- *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 it available for download. The current state of the integration can be provided upon request.

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:

- *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.
- *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. Explanations for the differences can be provided, for example, not all original data being available, or the publication being affected by a known and documented bug.
- *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.
- *Unknown (D)*: The current AMT version cannot run experiments for this model and cannot produce any results usable for verification. This is the default state right after having a model implementation provided to the AMT team.

A table providing an overview of the available models and their status can be found on the AMT website[15]. Note that the status is only a snapshot of the development because the implementations in the AMT are continuously developed, evaluated, and improved. The status is not fixed at any time and can be appealed by the authors. Any feedback is appreciated either via email or ticket created on the source-code repository system at SourceForge[16].

3 Environment

Figure 2 shows the logical structure of the AMT 1.x. The auditory models are complemented by model-independent resources such as core functions, common

[15] http://amtoolbox.org/.
[16] https://sourceforge.net/.

functions, a caching mechanism, and access to auxiliary data with an online data repository. Further, the AMT uses various third-party toolboxes, mostly to accommodate individual model requirements. The AMT uses MATLAB [20] or Octave [90] as the environment (for more details on the requirements, see Sect. 4). Thus, we use MATLAB's syntax in the following sections when describing the AMT functionality.

3.1 Core functions

The core functions control the configuration and the workflow of the AMT. The most essential function is `amt_start`, which installs toolboxes when required and sets up a default configuration for the work with other AMT functions. Other AMT core functions further warrant proper functioning, for example, through compilation of binaries on the user's system (`amt_mex`), calling functions from external environments (`amt_extern`), handling the cache (`amt_cache`, see Sect. 3.3), loading auxiliary data (`amt_load`, see Sect. 3.4), or running experiments (`amt_emuexp`).

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 Sect. 3.2), base path of the AMT, paths of auxiliary data (see Sect. 3.4) and cache (see Sect. 3.3), as well as the 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_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 showing results that are important for the online documentation, `amt_disp` can be called with the parameter `documentation`.

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 the models are actually run outside of MATLAB or Octave. The 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 intended 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`. The AMT 1.x uses Python version 3 installed with packages NumPy and SciPy. Models implemented in interoperable 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 by `amt_mex`, which produces MEX binaries that 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` on Windows or `makefile` on Linux 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).

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. 5, top), after an initialization of the procedure, signals are generated and provided to a human listener. Then, the listener 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. 5, bottom). The initialization phase consists of setting up `amt_emuexp` by separately providing parameters for the experiment, the signal generation, the auditory model, and the 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, the 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 `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 [42]. The functionality of `amt_emuexp` is similar to that from AFC[17], [91]; both support the emulation of experiments following the interface proposed for testing and comparing models called "model initiative" [92]. 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`, see `exp_breebaart2001` for an example.

[17] http://www.aforcedchoice.com.

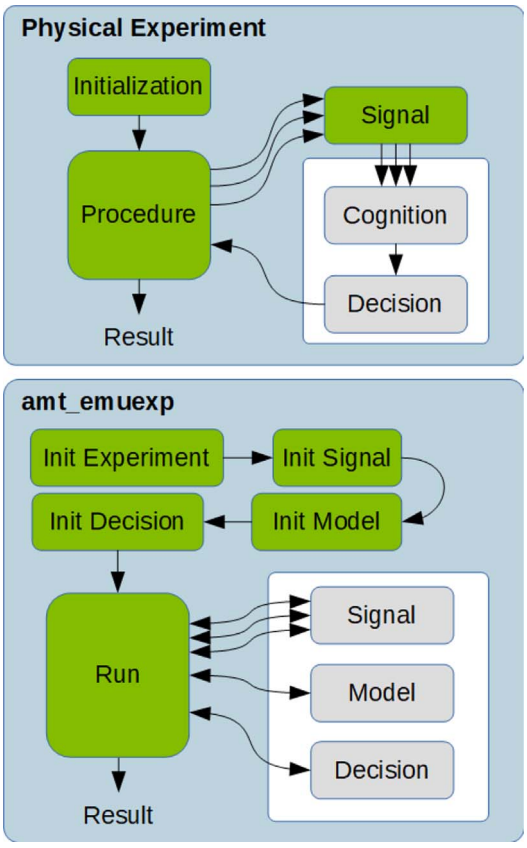


Figure 5. Example of a typical psychophysical experiment (top) and its emulation within the AMT (bottom). *Top*: In a physical experiment, after initialization, the main procedure triggers signal generation. The subject, who receives the signals, processes them and provides a decision, terminating with a result after a 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 cognitive processes, 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 participant (top) and functions outside the `amt_emuexp` (bottom).

3.2 Dependencies and third-party toolboxes

The AMT uses various third-party toolboxes, which 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, the 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) [93] is an essential toolbox, which means that the AMT will not run without it. The AMT uses the LTFAT's core functions for 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 HRTFs stored as SOFA files and provides general functionality to analyze, process, and display such data [94]. 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 [72]. These toolboxes are, along with the Circular Statistics and the Binaural Spherical Harmonics toolbox, provided within the AMT 1.1 full release package (see Sect. 4).

3.3 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 that after having some data calculated, the results can be stored in the 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 valid online cache data controlled by the AMT team. The online cache uses incremental versioning. Hence, only data that differs from previous versions 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 files named by the user within the AMT's cache directory and subsequent directories named by the caller function. 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. The `global` mode is set up during the startup by `amt_start`, which can be called with one of the cache modes as an input argument. 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 the fast access of results and 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 they are not available locally. 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 to selectively control the cache behavior of specific AMT functions. For example, `exp_lindemannl986('fig6')` plots the respective figure from Lindemann et al. (1986) [49] based on cached results, whereas `exp_lindemannl986 ('fig6','redo')` first calculates the data and then plots that figure.

3.4 Data functions and auxiliary data

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

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 that differs from previous versions 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 functions access data referring to a specific publication. They have the prefix `data_`, e.g., `data_majdak2010` returns the localization responses from Majdak et al. (2010) [95]. Data functions provide 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 amounts of auxiliary data.

HRTFs form a separate data category. They describe the acoustic filtering of the sound by the listener's body, in particular, the head, torso, and ears [96]. 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 the caching of the HRTFs, which are, if not found locally, automatically downloaded from the AMT online HRTF repository.

3.5 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), and neither are they model-specific (in contrast to model stages), but 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. Moreover, there are common functions calculating 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, the model-dependent functionality of a common auditory function can be triggered by using model-dependent flags. An example is `ihcenvelope` implementing the widely used model of the inner hair cells applying signal rectification followed by low-pass filtering to the input signal. The model specific parametrization of `ihcenvelope` can be triggered by using model-specific flags. For instance, in order to use the inner hair cell processing with parameters from Bernstein et al. (1999) [97], the flag `ihc_bernsteinl999` can be employed, i.e., `ihcenvelope(insig, fs, 'ihc_bernsteinl999')`.

4 Getting started with the AMT

The AMT 1.1 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). Other AMT 1.x versions are being developed and tested with up to five-years old MATLAB versions. While most of the models will work with even older versions of MATLAB and Octave, we recommend using one of the tested environments. When using MATLAB, 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. Finally, the AMT 1.x requires third-party toolboxes (see Sect. 3.2), which can be obtained in several ways, depending on the installation method.

The easiest and recommended method to install the AMT is to use a release marked as `amtoolbox-full` package, which contains all the third-party toolboxes. With a full package downloaded and unzipped on the user's system, the AMT is installed and can be started with `amt_start`.

The second method is to let the AMT download the third-party toolboxes during the first start of the AMT. This is required when the full package of the AMT release is not available. This installation method can then be triggered with `amt_start('install')`, which performs an interactive inquiry of the toolboxes to be downloaded and installed.

The third method is the manual installation of the third-party toolboxes by the user. This method provides the most flexibility, but requires more effort and knowledge about the user's system.

Regardless of the installation method, if the LTFAT is not available on the start of `amt_start`, `amt_start` will per default download and install it (or terminate with an error if the install fails), because the LTFAT is an essential toolbox for the AMT. However, `amt_start` does not download other toolboxes because they are not essential to the AMT's core functionality.

Some of the AMT models need to be compiled for the particular system of the user. For the user's 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 the 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 compilers available to compile MEX files; or 2) calling `system('gcc -v')` displayes the installed GCC version if available. Note that `amt_start('install')` also executes `amt_mex` in order to provide compiled models.

The AMT configuration and the status of the available toolboxes are handled by the command `amt_configuration` (see Sect. 3.1). Information on which toolboxes are required for running a specific model can be queried by executing `amt_info` with the model name as an input argument.

5 Contributing to the AMT

Each AMT user is warmly welcome to contribute to the AMT. The motivations 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 to increase its potential impact. Or, an AMT user has applied a model from the AMT and has implemented experiments or demonstrations displaying its functionality. Or, an AMT user has a general urge to support reproducible research and to learn about open-source projects and auditory modeling.

5.1 Coding

Before locally modifying AMT files, we highly encourage contributors to retrieve the most recent version from the sourcecode repository[18] and to integrate their modifications there. 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 the collaborative development of software, [98]):

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

[18] https://sourceforge.net/p/amtoolbox/code/.

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. The user's SourceForge[19] account will be included 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 user's 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 Sect. 3.1). 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* [99]. Still in order to make the integration of changes as smooth as possible, the contributors are asked to consider the following rules.

5.1.1 Environment

The aim is to warrant compatibility of the AMT 1.x with up to five-year old MATLAB versions, i.e., MATLAB 2016a for the AMT 1.1. 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 already provided by the current AMT version and/or its third-party toolboxes. When proposing an additional third-party toolbox, we ask to check whether it is freely available and if 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 need to 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`, confirm its functioning within the AMT environment, and notify us. We will then integrate its functionality and usage within the AMT environment. Note that, to keep the repository compact, it is not allowed to include third-party toolboxes in the AMT code repository.

5.1.2 Directory structure

The files are stored in directories reflecting the AMT environment (see Sect. 3). The model contribution is represented by a single function in the directory `models`. This model function can be complemented by (multiple) model stages stored in the directory `modelstages`. Model-specific data go to `data`, plotting functions go to `plot`, and signal generators go to `signals`. Model demonstrations go to `demos` and experiments reproducing results go to `experiments`. Large 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[20].

[19] https://sourceforge.net/.

[20] Commit of data to the code repository is not allowed.

In demonstrations and experiments, if the processing duration is beyond a few minutes, we encourage to cache the results via `amt_cache`.

5.1.3 Function and file names

Underscore is a reserved character in the AMT environment and used only to distinguish structural parts. All function names are lowercase. This avoids a lot of confusion because 1) the handling of the casing depends on the operating system, and 2) in the MATLAB and Octave documentations, function names are traditionally converted to uppercase. Function names indicate what they do, rather than which algorithm they use, or the person who programmed or invented it. For the reasons pointed out earlier in Section 2, 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 containing the main function, have the prefix `local_` in order to be easily distinguished from other AMT functions. The function `amt_disp` is used to display text in the command window, offering functionality such as the volatile display of computational progress and the singling out of specific results for publication in the online documentation. For the sake of simplicity, object-oriented programming and the usage of own classes are not recommended.

5.1.4 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. Global variables are 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[21]. When creating a new model, all its default parameters can be stored in a separate file placed in the directory `defaults` and named by the model's name with the prefix `arg_`. This file needs to be a function with `definput` as an input and output parameter, in which flags are stored in the structure `definput.flags` and key-value pairs are stored in the structure `definput.keyvals` – see any arbitrary `arg_` function for more details. Then, inside the model function, `ltfatarghelper()` is used for the parameter parsing as follows:

```
definput.import = {'model2021'}; [f,kv]
= ltfatarghelper({},definput,varargin);
```

[21] See http://ltfat.github.io/doc/base/ltfatarghelper.html.

resulting in flags and key-value pairs being stored in the structures `f` and `kv`, respectively. The `varargin` input passes the optional parameters provided through the model call to `ltfatarghelper`. There, the input arguments are processed from right to left. As a consequence, the default parameters from the `arg_` function will be overwritten by the optionally provided parameters. While `ltfatarghelper` supports more complex passing of parameters, in order to provide a clearly structured handling of the default parameters, we discourage from loading multiple default parameter files.

5.1.5 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 the 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 a peak amplitude of $\pm\sqrt{2}$) corresponds to an SPL of 93.9794 dB. This level convention reflects the SI system and is the default level convention in the AMT[22]. 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 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`. The level convention can be ignored if only linear models, such as the linear Gammatone filterbank, are applied. Note that because of historical reasons, previous AMT versions used a different level convention as the default.

5.2 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 its documentation can be provided while still generating a human-readable documentation. When releasing a new AMT version, this documentation is compiled by the compiler mat2doc[23] 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[24], a widely used markup syntax and parser component of Docutils[25]. The documentation based on reStructuredText can be compiled online[26]. 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[27] identifiers includes a list of references 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 function names 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, the requirements for running the models, their licenses, and 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`, `BinauralSH`, 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 Section 2.4.

5.3 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 the GPL. By committing code to the AMT repository, contributors agree to use

[22] Note that this is a non-compatible change with respect to the AMT 0.x.
[23] http://mat2doc.sourceforge.net/.
[24] https://docutils.sourceforge.io/rst.html.
[25] https://docutils.sourceforge.io/index.html.
[26] http://rst.ninjs.org/.
[27] http://www.bibtex.org/.

that license and to transfer the ownership to the AMT team, unless other licensing has been agreed.

While the AMT generally follows the GPL, the AMT 1.x also supports multiple licensing. Hence, 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. Note that some licenses (or even patents) may restrict the usage of the model implementation. The code can be included in the AMT as long as the license allows us to distribute that code within the AMT.

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 the 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 their 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) The in-code documentation provides file-specific information about the license via the license anchor followed by the license keyword (casing ignored). 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. Files without the license information are licensed under the standard AMT license.

5.4 Acknowledging researchers for their contribution

Developing new models is much work, while publishing the model implementation online is easy. Thus, researchers may ask for the motivation to permit their work to be integrated in the AMT or to put in even more effort and integrate it in the AMT by themselves. To address this issue, the AMT provides a variety of ways to acknowledge the researcher's work and display their 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 promotion 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, programmers, who provide a significant improvement to the AMT code (but neither contribute a full model nor an experiment) get acknowledged by noting their names in the sources and the online code repository. The AMT website lists all AMT contributors so far.

6 Conclusions

The AMT 1.x implements a variety of peripheral and higher-level auditory models and integrates both publication-specific and general datasets. Its most recent version is available (with all the required third-party toolboxes) from SourceForge[28] 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[29]. 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

[28] https://sourceforge.net/projects/amtoolbox.
[29] http://amtoolbox.org/.

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 1.1, the AMT has matured to a collection of over 60 auditory models. It now includes new models such as those based on Bayesian inference, statistical signal processing, and on speech intelligibility predictions. 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.

Conflict of interest

The authors declare that they have no conflict of interest.

Acknowledgments

We would like to thank Peter Søndergaard for initiating the AMT as a project for facilitating reproducible research by collecting auditory models and making them available in a common framework. The AMT is the umbrella project of the group "Aural Assessment By means of Binaural Algorithms" (AABBA), which is an intellectual group of scientists collaborating on the development and applications of models of human spatial hearing. We thank Jens Blauert for the initiative to create AABBA and we thank all AABBA members for their long-standing support. AABBA is the driving force behind the development of the AMT. Finally, the AMT would not be that comprehensive without the many contributions from various researchers from the auditory community. We are wholeheartedly grateful to all the contributors. This work was supported by the European Union (EU) within the project SONICOM (grant number: 101017743, RIA action of Horizon 2020) and by the Austrian Science Fund (FWF) within the project Dynamates (grant number: ZK66).

Data availability statement

This article describes computer code of the Auditory Modeling Toolbox (AMT) version 1.1.0. Because the AMT 1.x is continuously being updated, we encourage to use the most recent release of the AMT 1.x available at https://sf.net/projects/amtoolbox/files/AMT%201.x/ [100]. The source code of all AMT 1.x versions can be found at https://sf.net/p/amtoolbox/code/ci/AMT_1.x/tree/.

The documentation of *all* AMT versions can be found at http://amtoolbox.org/doc.php [101]. The general AMT landing page containing all AMT-related information is located at http://amtoolbox.org/ [102].

References

1. R. Frigg, S. Hartmann: Models in Science, in: E.N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, Fall, 2012.
2. R. Meddis, E. Lopez-Poveda, R.R. Fay, A.N. Popper (Eds.): Computational Models of the Auditory System, Springer, US, 2010. https://doi.org/10.1007/978-1-4419-5934-8.
3. B.R. Jasny, G. Chin, L. Chong, S. Vignieri: Again, and again, and again …. Science 334, 6060 (2011) 1225. https://doi.org/10.1126/science.334.6060.1225.
4. P. Vandewalle, J. Kovacević, M. Vetterli: Reproducible research in signal processing: What, why, and how. IEEE Signal Processing Magazine 26, 3 (2009) 37–47. https://doi.org/10.1109/MSP.2009.932122.
5. T. Dau, B. Kollmeier, A. Kohlrausch: Modeling auditory processing of amplitude modulation. I. Detection and masking with narrow-band carriers. Journal of the Acoustical Society of America 102, 5 (1997) 2892–2905.
6. S. Ewert, T. Dau: Characterizing frequency selectivity for envelope fluctuations. Journal of the Acoustical Society of America 108 (2000) 1181–1196. https://doi.org/10.1121/1.1288665.
7. S. Jørgensen, S.D. Ewert, T. Dau: A multi-resolution envelope-power based model for speech intelligibility. Journal of the Acoustical Society of America 134, 1 (2013) 436–446. https://doi.org/10.1121/1.4807563.
8. M. Schwab, N. Karrenbach, J. Claerbout: Making scientific computations reproducible. Computing in Science & Engineering 2, 6 (Nov. 2000) 61–67. https://doi.org/10.1109/5992.881708.
9. J.P. Mesirov, Accessible Reproducible Research, Science 327, 5964 (2010) 415–416. https://doi.org/10.1126/science.1179653.
10. R.D. Peng, Reproducible research in computational science, Science 334, 6060 (2011) 1226–1227. https://doi.org/10.1126/science.1213847.
11. J.F. Claerbout, M. Karrenbach: Electronic documents give reproducible research a new meaning, in: SEG Technical Program Expanded Abstracts 1992, Society of Exploration Geophysicists. 1992, pp. 601–604. https://doi.org/10.1190/1.1822162.
12. B.E. Peterson, M.D. Healy, P.M. Nadkarni, P.L. Miller, G. M. Shepherd: ModelDB: an environment for running and storing computational models and their results applied to neuroscience. Journal of the American Medical Informatics Association: JAMIA 3, 6 (1996) 389–398. https://doi.org/10.1136/jamia.1996.97084512.
13. T.M. Morse: Model sharing in computational neuroscience. Scholarpedia 2, 4 (2007) 3036. https://doi.org/10.4249/scholarpedia.3036.
14. R. Lyon: Cascades of two-pole–two-zero asymmetric resonators are good models of peripheral auditory function. Journal of the Acoustical Society of America 130 (2011) 3893–3904. https://doi.org/10.1121/1.3658470.
15. L.H. Carney, T. Li, J.M. McDonough: Speech coding in the brain: representation of vowel formants by midbrain neurons tuned to sound fluctuations. eNeuro 2, 4 (2015). https://doi.org/10.1523/ENEURO.0004-15.2015.
16. S. Verhulst, A. Altoè, V. Vasilkov: Computational modeling of the human auditory periphery: Auditory-nerve responses, evoked potentials and hearing loss. Hearing Research 360 (2018) 55–75. https://doi.org/10.1016/j.heares.2017.12.018.
17. I.C. Bruce, Y. Erfani, M.S.A. Zilany: A phenomenological model of the synapse between the inner hair cell and auditory nerve: Implications of limited neurotransmitter release sites. Hearing Research 360 (2018) 40–54. https://doi.org/10.1016/j.heares.2017.12.016.

18. P. Ru: Multiscale Multirate Spectro-Temporal Auditory Model. PhD Thesis, University of Maryland College Park, 2001. [Online]. Available: http://nsl.isr.umd.edu/downloads.html
19. B.C.J. Moore: Development and current status of the "Cambridge" loudness models. Trends in Hearing 18 (2014) 2331216514550620. https://doi.org/10.1177/2331216514550620.
20. D.J. Higham, N.J. Higham: MATLAB guide, vol. 150, Siam.
21. J.W. Eaton, D. Bateman, S. Hauberg: GNU Octave Manual, Network Theory Ltd.. 2002.
22. Malcolm Slaney: Auditory Toolbox: A MATLAB toolbox for auditory modeling work, Interval Research Corporation, Technical Report #1998-010. 1998. Accessed: Mar. 26, 2021. [Online]. Available: https://engineering.purdue.edu/~malcolm/interval/1998-010/AuditoryToolboxTechReport.pdf
23. R.D. Patterson, M.H. Allerhand, C. Giguère: Time-domain modeling of peripheral auditory processing: a modular architecture and a software platform. Journal of the Acoustical Society of America 98, 4 (1995) 1890–1894. https://doi.org/10.1121/1.414456.
24. A. Härmä, K. Palomäki: HUTear – A Free MATLAB Toolbox for Modeling of Human Auditory System. 1999, pp. 96–99. Accessed: Mar. 26, 2021. [Online]. Available: http://legacy.spa.aalto.fi/software/HUTear/.
25. D.C. Mountain, D.A. Anderson, G.J. Bresnahan, S.G. Deligeorges, A.E. Hubbard, V. Vajda: EarLab: A modular approach to auditory simulation. Journal of Biomechanics 39 (2006) S434. https://doi.org/10.1016/S0021-9290(06)84771-8.
26. M. Rudnicki, O. Schoppe, M. Isik, F. Völk, W. Hemmert: Modeling auditory coding: from sound to spikes. Cell and Tissue Research 361, 1 (2015) 159–175. https://doi.org/10.1007/s00441-015-2202-z.
27. L.P. O'Mard: Development System for Auditory Modelling (DSAM), Centre for the Neural Basis of Hearing (CNBH), 2012. [Online]. Available: http://dsam.org.uk.
28. B. Fontaine, D.F.M. Goodman, V. Benichoux, R. Brette: Brian hears: Online auditory processing using vectorization over channels. Frontiers in Neuroinformatics 5 (2011). https://doi.org/10.3389/fninf.2011.00009.
29. M. Stimberg, R. Brette, D.F. Goodman: Brian 2, an intuitive and efficient neural simulator. eLife 8 (2019) e47314. https://doi.org/10.7554/eLife.47314.
30. A. Gutkin: Eidos: an open-source auditory periphery modeling toolkit and evaluation of cross-lingual phonemic contrasts, in: Proceedings of the 1st Joint Workshop on Spoken Language Technologies for Under-resourced languages (SLTU) and Collaboration and Computing for Under-Resourced Languages (CCURL), Marseille, France, May 2020, pp. 9–20. Accessed: Mar. 26, 2021. [Online]. Available: https://www.aclweb.org/anthology/2020.sltu-1.2
31. T. Biberger, H. Schepker, F. Denk, S.D. Ewert: Instrumental quality predictions and analysis of auditory cues for algorithms in modern headphone technology. Trends in Hearing 25 (2021) 23312165211001220. https://doi.org/10.1177/23312165211001219.
32. Two!Ears Team: Two!Ears Auditory Model 1.5, Zenodo, 2018. https://doi.org/10.5281/zenodo.1458420.
33. P. Søndergaard, P. Majdak: The Auditory Modeling Toolbox. In: J. Blauert (Ed.), The Technology of Binaural Listening, Berlin-Heidelberg, Germany: Springer, 2013, pp. 33–56.
34. M. Dietz, S.D. Ewert, V. Hohmann: Auditory model based direction estimation of concurrent speakers from binaural signals. Speech Communication 53 (2011) 592–605.
35. H. Ziegelwanger, P. Majdak: Modeling the direction-continuous time-of-arrival in head-related transfer functions. Journal of the Acoustical Society of America 135, 3 (2014) 1278–1293. https://doi.org/10.1121/1.4863196.
36. E.A. Lopez-Poveda, R. Meddis: A human nonlinear cochlear filterbank. Journal of the Acoustical Society of America 110, 6 (2001) 3107–3118. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/11785812.
37. V. Hohmann: Frequency analysis and synthesis using a Gammatone filterbank. Acta Acustica united with Acustica 88, 3 (2002) 433–442.
38. S. Verhulst, T. Dau, C. Shera: Nonlinear time-domain cochlear model for transient stimulation and human otoacoustic emission. Journal of the Acoustical Society of America 132 (2012) 3842–3848.
39. M.S.A. Zilany, I.C. Bruce: Representation of the vowel ε in normal and impaired auditory nerve fibers: Model predictions of responses in cats. Journal of the Acoustical Society of America 122 (2007) 402–417.
40. M.S.A. Zilany, I.C. Bruce, L.H. Carney: Updated parameters and expanded simulation options for a model of the auditory periphery. Journal of the Acoustical Society of America 135, 1 (2014) 283–286. https://doi.org/10.1121/1.4837815.
41. T. Dau, D. Püschel, A. Kohlrausch: A quantitative model of the "effective" signal processing in the auditory system. I. Model structure. Journal of the Acoustical Society of America 99, 6 (1996) 3615–3622. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/8655793.
42. J. Breebaart, S. van de Par, A. Kohlrausch: Binaural processing model based on contralateral inhibition. III. Dependence on temporal parameters. Journal of the Acoustical Society of America 110, 2 (2001) 1105–1117. [Online]. Available: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=retrieve&db=pubmed&dopt=abstract&list_uids=11519578
43. H. Relaño-Iborra, J. Zaar, T. Dau: A speech-based computational auditory signal processing and perception model. Journal of the Acoustical Society of America 146, 5 (2019) 3306–3317. https://doi.org/10.1121/1.5129114.
44. M.L. Jepsen, S.D. Ewert, T. Dau: A computational model of human auditory signal processing and perception. Journal of the Acoustical Society of America 124, 1 (2008) 422–438. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/18646987.
45. N.F. Viemeister: Temporal modulation transfer functions based upon modulation thresholds. Journal of the Acoustical Society of America 66, 5 (1979) 1364–1380. https://doi.org/10.1121/1.383531.
46. F.M. Rønne, T. Dau, J. Harte, C. Elberling: Modeling auditory evoked brainstem responses to transient stimuli. Journal of the Acoustical Society of America 131, 5 (2012) 3903–3913. https://doi.org/10.1121/1.3699171.
47. S. Verhulst, H.M. Bharadwaj, G. Mehraei, C.A. Shera, B.G. Shinn-Cunningham: Functional modeling of the human auditory brainstem response to broadband stimulation. Journal of the Acoustical Society of America 138, 3 (2015) 1637–1659. https://doi.org/10.1121/1.4928305.
48. A. King, L. Varnet, C. Lorenzi: Accounting for masking of frequency modulation by amplitude modulation with the modulation filter-bank concept. Journal of the Acoustical Society of America 145, 4 (2019) 2277–2293.
49. W. Lindemann: Extension of a binaural cross-correlation model by contralateral inhibition. I. Simulation of lateralization for stationary signals. Journal of the Acoustical Society of America 80, 6 (1986) 1608–1622. https://doi.org/10.1121/1.394325.

50. M. Takanen, O. Santala, V. Pulkki: Binaural assessment of parametrically coded spatial audio signals. In: J. Blauert (Ed.), The technology of binaural listening, Berlin, Germany: Springer. 2013, pp. 333–358.
51. B.C.J. Moore, B.R. Glasberg, T. Baer: A model for the prediction of thresholds, loudness, and partial loudness. Journal of the Audio Engineering Society 45, 4 (1997) 224–240. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=10272
52. B.R. Glasberg, B.C.J. Moore: A model of loudness applicable to time-varying sounds. Journal of the Audio Engineering Society 50, 5 (2002) 331–342. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=11081
53. Z. Chen, G. Hu, B.R. Glasberg, B.C.J. Moore: A new model for calculating auditory excitation patterns and loudness for cases of cochlear hearing loss. Hearing Research 282, 1 (2011) 69–80. https://doi.org/10.1016/j.heares.2011.09.007.
54. B.C.J. Moore, B.R. Glasberg, A. Varathanathan, J. Schlittenlacher: A loudness model for time-varying sounds incorporating binaural inhibition. Trends in Hearing 20 (2016). https://doi.org/10.1177/2331216516682698.
55. S. Jørgensen, T. Dau: Predicting speech intelligibility based on the signal-to-noise envelope power ratio after modulation-frequency selective processing. Journal of the Acoustical Society of America 130, 3 (2011) 1475–1487. https://doi.org/10.1121/1.3621502.
56. C.H. Taal, R.C. Hendriks, R. Heusdens, J. Jensen: An algorithm for intelligibility prediction of time-frequency weighted noisy speech. IEEE Transactions on Audio, Speech, and Language Processing 19, 7 (2011) 2125–2136. https://doi.org/10.1109/TASL.2011.2114881.
57. J.F. Culling, M.L. Hawley, R.Y. Litovsky: The role of head-induced interaural time and level differences in the speech reception threshold for multiple interfering sound sources. Journal of the Acoustical Society of America 1162 (2004) 1057–1065. [Online]. Available: http://www.ncbi.nih.gov/entrez/query.fcgi?cmd=retrieve&db=pubmed&dopt=abstract&list_uids=15376672.
58. S. Jelfs, J.F. Culling, M. Lavandier: Revision and validation of a binaural model for speech intelligibility in noise. Hearing Research 275, 1–2 (2011). https://doi.org/10.1016/j.heares.2010.12.005.
59. T. Leclere, M. Lavandier, J.F. Culling: Speech intelligibility prediction in reverberation: Towards an integrated model of speech transmission, spatial unmasking, and binaural dereverberation. Journal of the Acoustical Society of America 137, 6 (2015) 3335–3345. https://doi.org/10.1121/1.4921028.
60. C.F. Hauth, S.C. Berning, B. Kollmeier, T. Brand: Modeling binaural unmasking of speech using a blind binaural processing stage. Trends in Hearing 24 (2020) 2331216520975630. https://doi.org/10.1177/2331216520975630.
61. L. Prud'homme, M. Lavandier, V. Best: A harmonic-cancellation-based model to predict speech intelligibility against a harmonic masker. Journal of the Acoustical Society of America 148, 5 (2020) 3246–3254. https://doi.org/10.1121/10.0002492.
62. T. Vicente, M. Lavandier: Further validation of a binaural model predicting speech intelligibility against envelope-modulated noises. Hearing Research 390 (2020) 107937. https://doi.org/10.1016/j.heares.2020.107937.
63. T. Vicente, M. Lavandier, J.M. Buchholz: A binaural model implementing an internal noise to predict the effect of hearing impairment on speech intelligibility in non-stationary noises. Journal of the Acoustical Society of America 148, 5 (2020) 3305–3317. https://doi.org/10.1121/10.0002660.
64. M. Lavandier: A series of speech intelligibility models in the auditory modeling toolbox. Submitted to Acta Acustica (2022) .
65. A. Osses Vecchi, A. Kohlrausch: Perceptual similarity between piano notes: Simulations with a template-based perception model. Journal of the Acoustical Society of America 149, 5 (2021) 3534–3552. https://doi.org/10.1121/10.0004818.
66. P. Lladó, P. Hyvärinen, V. Pulkki: Auditory model-based estimation of the effect of head-worn devices on frontal horizontal localisation. Acta Acustica 6 (2022) 1. https://doi.org/10.1051/aacus/2021056.
67. P. Zakarauskas, M.S. Cynader: A computational theory of spectral cue localization. Journal of the Acoustical Society of America 94 (1993) 1323–1331.
68. E.H.A. Langendijk, A.W. Bronkhorst: Contribution of spectral cues to human sound localization. Journal of the Acoustical Society of America 112, 4 (2002) 1583–1596. https://doi.org/10.1121/1.1501901.
69. T. May, S. van de Par, A. Kohlrausch: A probabilistic model for robust localization based on a binaural auditory front-end. IEEE Transactions on Audio, Speech, and Language Processing 19 (2011) 1–13.
70. R. Baumgartner, P. Majdak, L. Bernhard: Assessment of sagittal-plane sound localization performance in spatial-audio applications. In: J. Blauert (Ed.), The Technology of Binaural Listening, Berlin, Heidelberg: Springer. 2013, pp. 93–119.
71. E. Georganti, T. May, S. van de Par, J. Mourjopoulos: Sound source distance estimation in rooms based on statistical properties of binaural signals. IEEE Transactions on Audio, Speech, and Language Processing 21, 8 (2013) 1727–1741. https://doi.org/10.1109/TASL.2013.2260155.
72. H. Wierstorf, A. Raake, S. Spors: Binaural assessment of multichannel reproduction. In: J. Blauert (Ed.), The Technology of Binaural Listening, Berlin, Heidelberg: Springer, Berlin Heidelberg, 2013, pp. 255–278. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37762-4_10.
73. R. Baumgartner, P. Majdak, B. Laback: Modeling sound-source localization in sagittal planes for human listeners. Journal of the Acoustical Society of America 136 (2014) 791–802. https://doi.org/10.1121/1.4887447.
74. J. Reijniers, D. Vanderelst, C. Jin, S. Carlile, H. Peremans: An ideal-observer model of human sound localization. Biological Cybernetics 1082 (2014) 169–181. https://doi.org/10.1007/s00422-014-0588-4.
75. D. Kelvasa, M. Dietz: Auditory model-based sound direction estimation with bilateral cochlear implants. Trends in Hearing 19 (2015). https://doi.org/10.1177/2331216515616378.
76. R. Baumgartner, P. Majdak, B. Laback: Modeling the effects of sensorineural hearing loss on sound localization in the median plane. Trends in Hearing 20 (2016) 2331216516662003. https://doi.org/10.1177/2331216516662003.
77. H.G. Hassager, F. Gran, T. Dau: The role of spectral detail in the binaural transfer function on perceived externalization in a reverberant environment. Journal of the Acoustical Society of America 139, 5 (2016) 2992–3000. https://doi.org/10.1121/1.4950847.
78. R. Baumgartner, D.K. Reed, B. Tóth, V. Best, P. Majdak, H.S. Colburn, B. Shinn-Cunningham: Asymmetries in behavioral and neural responses to spectral cues demonstrate the generality of auditory looming bias. Proceedings of the National Academy of Sciences of the United States of America 114, 36 (2017) 9743–9748. https://doi.org/10.1073/pnas.1703247114.

79. S. Li, R. Baumgartner, J. Peissig: Modeling perceived externalization of a static, lateral sound image. Acta Acustica 4, 5 (2020) 5. https://doi.org/10.1051/aacus/2020020.
80. Robert Baumgartner, Piotr Majdak: Decision making in auditory externalization perception: model predictions for static conditions. Acta Acustica 5 (2021) 59. https://doi.org/10.1051/aacus/2021053.
81. R. Barumerli, P. Majdak, R. Baumgartner, M. Geronazzo, F. Avenzini: Predicting human spherical sound-source localization based on Bayesian inference. Submitted to Acta Acustica (2022).
82. G. McLachlan, P. Majdak, J. Reijniers, H. Peremans: Towards modelling active sound localisation based on Bayesian inference in a static environment. Acta Acustica 5 (2021) 45. https://doi.org/10.1051/aacus/2021039.
83. R. Baumgartner, P. Majdak: Modeling localization of amplitude-panned virtual sources in sagittal planes. Journal of the Audio Engineering Society 63, 7/8 (2015) 562–569. https://doi.org/10.17743/jaes.2015.0063.
84. R. Baumgartner, P. Majdak, B. Laback: The reliability of contralateral spectral cues for sound localization in sagittal planes, in: Presented at the Midwinter Meeting of the Association for Research in Otolaryngology, Baltimore, MD, USA, 2015.
85. J. Engel Alonso Martinez, D. Goodman, L. Picinali: Assessing HRTF preprocessing methods for Ambisonics rendering through perceptual models. Acta Acustica 6 (2022) 4. https://doi.org/10.1051/aacus/2021055.
86. A. Osses Vecchi, L. Varnet, L.H. Carney, T. Dau, I.C. Bruce, S. Verhulst, P. Majdak: A comparative study of eight human auditory models of monaural processing. Acta Acustica 6 (2022) 17. https://doi.org/10.1051/aacus/2022008.
87. S. Röttges, C.F. Hauth, T. Brand, J. Rennies-Hochmuth: Challenging a non-intrusive EC-mechanism: Modelling the Interaction between binaural and temporal speech processing. Submitted to Acta Acustica (2022).
88. L. Steidle, R. Baumgartner, Geometrical evaluation of methods to approximate interaural time differences by broadband delays, in: Fortschritte der Akustik, Rostock. 2019, 368–370.
89. W.M. Hartmann, A. Wittenberg: On the externalization of sound images. Journal of the Acoustical Society of America 99, 6 (1996) 3678–3688. [Online]. Available: http://www.ncbi.nih.gov/entrez/query.fcgi?cmd=retrieve&db=pubmed&dopt= abstract&list_uids=8655799.
90. J.W. Eaton, D. Bateman, S. Hauberg, R. Wehbring: GNU Octave version 6.1.0 manual: a high-level interactive language for numerica l computations (2020). [Online]. Available: http://www.gnu.org/software/octave/doc/interpreter.
91. S.D. Ewert: AFC – A modular framework for running psychoacoustic experiments and computational perception models, in: Proceedings of the International Conference on Acoustics AIA-DAGA, Merano, Italy, 2013, pp. 1326–1329.
92. M. Dietz, J.H. Lestang, P. Majdak, R.M. Stern, T. Marquardt, S.D. Ewert, W.M. Hartmann, D.F. Goodman: A framework for testing and comparing binaural models. Hearing Research 360 (2018) 92–106. https://doi.org/10.1016/j.heares.2017.11.010.
93. Z. Průša, P.L. Søndergaard, N. Holighaus, C. Wiesmeyr, P. Balazs: The large time-frequency analysis toolbox 2.0, in: Sound Music, and Motion, Cham, 2014, pp. 419–442. https://doi.org/10.1007/978-3-319-12976-1_25.
94. P. Majdak, Y. Iwaya, T. Carpentier, R. Nicol, M. Parmentier, A. Roginska, Y. Suzuki, K. Watanabe, H. Wierstorf, H. Ziegelwanger, M. Noisternig: Spatially oriented format for acoustics: a data exchange format representing head-related transfer functions, in: Proceedings of the 134th Convention of the Audio Engineering Society (AES), Roma, Italy, 2013, Convention Paper 8880.
95. P. Majdak, M.J. Goupell, B. Laback: 3-D localization of virtual sound sources: effects of visual environment, pointing method, and training. Attention, Perception, & Psychophysics 72, 2 (2010) 454–469. https://doi.org/10.3758/APP.72.2.454.
96. H. Møller, M.F. Sørensen, D. Hammershøi, C.B. Jensen: Head-related transfer functions of human subjects. Journal of the Audio Engineering Society 43 (1995) 300–321.
97. L.R. Bernstein, S. van de Par, C. Trahiotis: The normalized interaural correlation: Accounting for NoSπ thresholds obtained with Gaussian and \"low-noise\" masking noise. Journal of the Acoustical Society of America 106 (1999) 870.
98. S. Chacon, B. Straub: Pro git. 2nd ed., Apress, 2014.
99. N. Barnes: Publish your computer code: it is good enough. Nature 467, 7317 (2010) 753. https://doi.org/10.1038/467753a.
100. The AMT Team: The Auditory Modeling Toolbox 1.x Full Packages. https://sourceforge.net/projects/amtoolbox/files/AMT%201.x/amtoolbox-full-1.0.0.zip/download (accessed Mar. 17, 2022).
101. The AMT Team: Documentation of the Auditory Modeling Toolbox (AMT). http://amtoolbox.org/doc.php (accessed Mar. 17, 2022).
102. The AMT Team: The Auditory Modeling Toolbox (AMT). http://amtoolbox.org/ (accessed Mar. 17, 2022).

Cite this article as: Majdak P. Hollomey C. & Baumgartner R. 2022. AMT 1.x: A toolbox for reproducible research in auditory modeling. Acta Acustica, **6**, 19.