**A Manual of Audapter**

**Version 2.1**

Shanqing Cai

[scai@bu.edu](mailto:scai@bu.edu) or [shanqing.cai@gmail.com](mailto:shanqing.cai@gmail.com)

Speech Laboratory,

Department of Speech, Language and Hearing Sciences,

Sargent College of Health and Rehabilitation Sciences,

Boston University

January 2014

1. Overview: What is Audapter?[[1]](#footnote-2)

Audapter is a software package for configurable real-time manipulation of acoustic parameters of speech that runs on general-purpose computers. It is designed for research on auditory-motor interactions in speech production, but may also be of use for certain speech signal processing applications. The current version of Audapter supports manipulation (i.e., perturbation) of the following acoustic parameters:

1. Formant frequencies (F1 and F2), in both static and time-varying ways
2. Fundamental frequency (F0, or pitch)
3. Local timing, through time-warping
4. Local intensity
5. Global time delay (delayed auditory feedback)
6. Global intensity

Perturbation types a – d can be automatically gated in on specific preselected parts of a given utterance, through a set of heuristic rules for online status tracking (OST, see Sect. X.X). Certain combinations of perturbation types (e.g., a and c, b and e) can be delivered simultaneously.

As a package, Audapter includes both the core algorithms for real-time speech signal processing and MATLAB wrap-arounds supporting psychophysical experiments. The real-time signal processing algorithms are coded in C++ and implemented as a MEX interface for MATLAB. A set of MATLAB scripts and programs are available for calling Audapter and utilizing it in various types of auditory feedback perturbation (AFP) experiments. These includes graphical user interfaces for stimulus presentation, experimental workflow control, data preprocessing, as well as basic analysis of ths data. Although Audapter has only be thoroughly used and tested on Windows PCs (both 32- and 64-bit), it should be portable to other platforms, including Mac.

Audapter was developed at the Speech Communication Group, Research Laboratory of Electronics (RLE), Massachusetts Institute of Technology (MIT) as well as the Speech Laboratory of Boston University. Marc Boucek (Ref) and Satrajit Ghosh originated the MEX C++ project. This code was partly based on algorithms written on DSP platforms by Virgilio Villacorta and Kevin J. Riley in earlier AFP experiments. Since 2007, Shanqing Cai, the author of this document, made extensive modifications to Audapter and added many new functions. Cai is currently the primary maintainer of this software package.

This manual will serve as a general guide to the usage of Audapter.

1.1. How to cite Audapter?

To cite the Audapter as a software package, use the following references:

Cai S, Boucek M, Ghosh SS, Guenther FH, Perkell JS. (2008). A system for online dynamic perturbation of formant frequencies and results from perturbation of the Mandarin triphthong /iau/. In *Proceedings of the 8th Intl. Seminar on Speech Production, Strasbourg, France*, Dec. 8 - 12, 2008. pp. 65-68.

Tourville JA, Cai S, Guenther FH (2013) Exploring auditory-motor interactions in normal and disordered speech. *Proceedings of Meeting on Acoustics*. 9:060180. Presented at the *165th Meeting of the Acoustical Society of America*, Montreal, Quebec, Canada, June 2 – June 7, 2013.

Published experimental studies based on Audapter from the MIT Speech Communication Group and Boston University Speech Lab include:

Cai S, Beal DS, Ghosh SS, Guenther FH, Perkell JS. (In press). Impaired timing adjustments in response to time-varying auditory perturbation during connected speech production in persons who stutter. Brain Lang.

Cai S, Ghosh SS, Guenther FH, Perkell JS. (2010). Adaptive auditory feedback control of the production of the formant trajectories in the Mandarin triphthong /iau/ and its patterns of generalization. J. Acoust. Soc. Am. 128(4):2033-2048.

Cai S, Ghosh SS, Guenther FH, Perkell JS. (2011). Focal manipulations of formant trajectories reveal a role of auditory feedback in the online control of both within-syllable and between-syllable speech timing. J. Neurosci. 31(45):16483-16490.

Cai S, Beal DS, Ghosh SS, Tiede MK, Guenther FH, Perkell JS. (2012). Weak responses to auditory feedback perturbation during articulation in persons who stutter: Evidence for abnormal auditory-motor transformation. PLoS ONE. 7(7):e41830.

When appropriate, these references can be also be cited.

2. Getting Started - Running Offline Demos

The Audapter package comes with a set of demo scripts that show you the basic capacity of the software as well as serve as examples for programming your own Audapter applications.

Details on how to obtain the code can be found in Appendix 1. Instructions on how to building the MEX program of Audapter in Microsoft Visual C++ can be found in Appendix 2. To set up the environment properly, you need to add path to Shanqing Cai's MATLAB toolkit, by entering in MATLAB a command such as the following:

addpath [e:/speechres/commonmcode](file:///e:\speechres\commonmcode);

Then, use the cds script in the toolkit to set up the paths and environment automatically:

cds('ape');

You can check that the path to the Audapter MEX program has been set up correctly by entering command:

which Audapter;

2.1. Offline Demos

These offline demos can be run in MATLAB without an ASIO-compatible sound card, because these demo scripts utilize the offline processing option of Audapter, i.e., the “runFrame option” (se Sect. X1).

2.1.1. Offline Demo 1: Formant perturbation

The command for bringing up this demo is:

test\_audapter('formant', '--play');

This command brings up two windows in MATLAB, each showing a spectrogram. The first window shows the spectrogram of the input signal, which is the English phrase “test a pepper” uttered by an adult male speaker. Overlaid in the spectrogram are the F1 and F2 tracks calculated by Audapter during the supra-threshold intervals, as well as a black curve showing the OST status (multiplied by 500 for visualization purpose). The second figure shows the spectrogram of the output, i.e., perturbed, speech signal. The F1 and F2 during the word “a” and the first syllable of the word “pepper” are altered. The new formant values are shown by the green curves in this figure. This demo program also plays the input and output signals, due to the inclusion of --play in the input argument. The consequence of this joint downward F1 and upward F2 is that the word “pepper” sounds more similar to the word “paper”.

This simple demo demonstrates three aspects of Audapter’s capacity: 1) formant tracking, 2) formant perturbation and 3) OST for tracking the progress of the sentence and delivering the perturbation at specific part of a multisyllabic utterance.





Figure 1. Graphical output of the demo command: test\_audapter('formant', '--play');

2.1.2. Offline Demo 2: F0 perturbation

To see the demo of Audapter’s F0 (pitch) perturbation capacity, use the following command:

test\_audapter('pitch', '--play');

The graphical output of this command is similar in format to the formant-perturbation demo. This example is based on the same recording as in Demo 1. However, unlike the formant perturbation example, the fundamental frequency (F0) is shifted up during the word “a” and the first syllable of the word “pepper”.

2.1.3. Offline Demo 3: Time warping

The following command brings up an example of time-warping perturbation:

test\_audapter('timeWarp', '--play');

Comparing the two spectrograms, you can see change in the timing of various parts of the utterance. Specifically, two time-warping events were included in this example. The first event lengths the duration of the [s] sound in the word “test” and delays the onset of the final [t] sound. It also delays the onset of the word “a”. This warping event ends at approximately the beginning of the first syllable in “pepper”. The second warping event starts during the silent interval before the onset of the second [p] sound in “pepper”. It lengthens this silent interval and thereby delays the onset of the noise release in the following [p] sound. As can be seen in this example, more than one time-warping events can be included in the same utterance.

2.1.4. Offline Demo 4: Dynamic perturbation of a Standard Chinese triphthong

In this demo, we show how audapter can impose a time-varying F1 perturbation during a triphthong [iau] in Standard Chinese. To run this demo, enter command:

audapterDemo\_triphthong('--play');



Figure 2. Graphical output of the demo command: audapterDemo\_triphthong('--play'). It shows the time-varying F1 perturbation during the Standard Chinese triphthong [iau].

The “--play" option leads to playback of the input and output waveforms. The script also shows a figure with two subplots, the left of which shows the input (unperturbed) spectrogram and the right of which illustrates the perturbed output. The white curves overlaid on these spectrograms are the original F1 and F2 values in the female speaker’s production, while the dashed green curves are the perturbed F1 and F2 values. The green F2 curve overlaps with the solid white F2 curve because the perturbation was limited to F1 in this example.

This type of time-varying perturbation is what is used in Cai et al. (2010). It is achieved by supplying proper values to these following Audapter parameters that support this type of OST-independent, time-varying formant manipulation: *f2Min*, *f2Max*, *f1Min*, *f1Max*, *LBk*, *LBb*, *pertF2*, *pertAmp* and *pertPhi*. Together, these nine parameters forms a structure called the perturbation field in the plane of F1 and F2. In this perturbation field, the amount of perturbations to F1 and F2 can vary as a function of the unperturbed value of F2. This also forms the basis for the “spatial” perturbation type used in Cai et al. (2011). Section W1 contains further details about the design and usage of the perturbation field. Detailed descriptions of the nine Audapter parameters can also be found in Section Y1.

2.2. Online demos

The following demos show the real-time speech perturbation capacity of Audapter. They use the real-time processing options (“start” and “stop”; see Section X1). In order to run them successfully, you need to have an ASIO-compatible audio interface attached to the computer. These demo scripts assume that the audio interface has the following settings: sampling rate = 48000 Hz; buffer length (frame size) = 96.

2.2.1. Online Demo 1. Persistent formant shift

audapterDemo\_online('persistentFormantShift', gender);

wherein gender is the gender of the user ('male', or 'female').

Mute mode. Noise feedback mode. Voice + noise mixing mode. Speech modulated noise.

2.2.2. Online Demo 2. Persistent pitch shift

audapterDemo\_online('persistentPitchShift');

2.2.3. Online Demo 3. Two fixed-delay, fixed-duration short pitch shifts in one utterance

Demo command line:

audapterDemo\_online('twoShortPitchShifts');

This demo includes two short pitch shifts following voicing onset, each of which lasts 200 ms. The second shift begins 300 ms after the end of the first shift. This timing control is achieved through the OST file in ../example\_data/two\_blips.ost. Studying this ost file and the associated PCF file (../example\_data/two\_pitch\_shifts.pcf) can give you a basic idea of how to use the OST and PCF capacities to perform fixed-duration, fixed-delay perturbations. The effect of this demo time warping will be the most salient if you can utter fast-changing sounds with abrupt onsets, such as “puh puh puh…”

2.2.4. Online Demo 4. Focal formant shift

2.2.5. Online Demo 5. Time warping

Command line:

audapterDemo\_online('timeWarp');

In this demo, Audapter waits for the onset of voicing, as detected by the INTENSITY\_RISE\_HOLD mode of OST and a hard-coded intensity threshold of 0.02. Then it waits for another 100 ms before initiating a time warping event. This time warping event is specified in the pcf file ../example\_data/time\_warp\_demo.pcf.

2.2.6. Online Demo 6. Globally delayed auditory feedback

2.2.7. Online Demo 7. Continuous sine wave generation

To run this demo, enter command:

audapterDemo\_online('playTone');

You will hear four notes (A, B, C#, A) played in a sequence. Even though this function may seem very similar to the tone sequence generation function (Sect. 2.2.9), there is an important difference. Notice that the last tone will play continuously and keep going, until the user hits Enter to trigger the Audapter(‘stop’) command. The tone sequence generator is not capable of producing continuous tones. Another difference is in the initial phases of the individual tones. You may be able to hear the discontinuities (clicks) in the sound produced by this example. This is because the tones produced under the “playTone” mode of Audapter do not have on/off ramps. The tone sequence generator, however, is capable of imposing ramps on the tones.

2.2.8. Online Demo 8. Waveform playback

This demo can be brought up by the command:

audapterDemo\_online('playTone');

You will hear an utterance being played from the output channel of the audio interface. This option is based on the “playWave” mode of Audapter. The waveform for playback is supplied to Audapter with the following syntax (see the script):

Audapter('setParam', 'datapb', sigInRS);

sigInRS must have the same sampling frequency as the audio interface’s hardware sampling frequency before downsampling. In addition, it’s length must not exceed the maximum playback sample count maxPBSize, which can be obtained through the command:

maxPBSize = Audapter('getMaxPBLen');

2.2.9. Online Demo 9. Tone sequence generation

X1. Basic Command Line Usage of Audapter

Audapter is a C++ MEX program that can be called from MATLAB. If you enter the command:

Audapter;

without any input arguments in MATLAB (after setting up the environment and paths correctly), you will see the help message in the MATLAB console. The help message lists the types of commands that you can send to Audapter. Each command can be called in two different ways, either as a command number (e.g.,1, 2) or as the corresponding character-based command name (e.g., start, stop).

To list the currently attached ASIO audio interfaces (i.e., sound cards), use:

Audapter('info');

To start a trial, which involves real-time audio processing, do:

Audapter('start');

To stop a trial, do:

Audapter('stop');

Note: to let Audapter work properly during the real-time processing trial, you need to have its parameters set properly, and optionally, have the online status tracking (OST) and perturbation configuration (PCF) files loaded properly. See the following sections (Sect. XX, XX and XX) on details of these settings. Among the parameters of Audapter, the most basic ones for ensuring the crash-free functioning include

1. downsampling factor (parameter “downFact”)
2. sampling rate (parameter “srate”),
3. frame length (parameter “frameLen”)

It is important to note that srate and frameLen should be the actual hardware sampling rate and buffer length divided by downFact. The downsampling factor, typically set to 3 or 4, is for reducing the computational load on CPUs for ensuring real-time processing. For example, if your audio interface has a sampling rate of 48000 Hz and a buffer length of 96, given that you’ve specified a downFact of 3, the values of srate and frameLen you should use are 16000 and 32, respectively.

To set a parameter of Audapter, use the setParam option:

Audapter('setParam', paramName, paramVal, bVerb);

In this command syntax, the second and third input arguments are the name of the parameter and the value you with set it to, respectively. The optional, fourth argument is a Boolean (0/1) variable that indicates whether the verbose mode is selected. If you do not include this argument, the verbose mode is set by default. For example, the following command

Audapter('setParam', 'bCepsLift', 0, 0);

sets the parameter bCepsLift (see Sect. XX) to 0 (i.e., false) under the silent (i.e., non-verbose) mode.

Oftentimes, for debugging and data analysis, you may want to run Audapter on pre-recorded speech sounds, in an offline fashion. The option “runFrame” allows you to do that. In fact, the offline demos you have seen rely on this option. In this offline mode, you supply Audapter with signal frames (i.e., buffers) of speech sound at a time, with the following syntax:

Audapter('runFrame', signalFrame);

wherein signalFrame is a vector of speech samples whose length matches frameLen \* downFact of Audapter. In other words, signalFrame is a frame of audio *before* downsampling. You can call Audapter in this way repeatedly, but with different signalFrames, to simulate the consecutive buffers that come in during an online, real-time trials. The test\_audapter script for the offline demos contains an example of how this is achieved.

The option “getData” of Audapter allows the user to extract audio and associated data from the last trial. This applies to either real-time trials triggered by options “start” and “stop” and offline trials triggered by option “runFrame”:

[sig, dat] = Audapter('getData');

In the output, sig is a N×2 matrix, in which N is the number of samples in the last online or offline trial after the downsampling. The first column is the input signal; the second one is the output (potentially perturbed) signal. The second output “dat” is a N×M matrix containing various data derived from the audio input, such as calculated formant frequencies, LP coefficients, short-time RMS intensity values, OST status numbers, etc. Each of the M columns is a different type of derived data. The matrix is not annotated and is not meant to be used directly by the user. Instead, there is a MATLAB script that wraps around the “getData” option of Audapter and generates much more readable data. It can be called in the following way (see the demo script: test\_audapter.m, for an example):

data = AudapterIO('getData');

The output data includes both the input / output signals and the derived data. Section XX contains a detailed description of all the fields of the output “data”.

The “reset” option in Audapter allows the user to reset the status of the temporary data fields in Audapter, so as to prepare for the next incoming trial. It can be called as:

Audapter('reset');

which is equivalent to the calling the “reset” option in the AudapterIO wrap-around:

AudapterIO('reset');

This resetting does not alter the parameter values. Instead, it sets memory fields that hold past audio signals, past formant values, etc., as well as the status of the OST tracker to zero or other proper initial values, so that a new trial can start without any influence from the previous trial. This resetting action should be performed prior to the onset of any new utterance in online and offline processing. The code in test\_audapter.m demo script shows that.

The “ost” and “pcf” options allows the loading of OST and PCF into Audapter, respectively, for specifying the details of online word tracking rules and perturbation to be delivered during the utterance. Details on how to use these options can be found in Sections X3 and X4.

The options of Audapter listed above are for speech signal processing. There are a number of other options in Audapter that support signal generation and playback functions that might be useful during psychophysical experiments, as listed below.

The “playTone” option lets Audapter generate a continuous sine wave, of which the frequency, amplitude and initial phase cangle can be specified in parameters wgFreq, wgAmp and wgTime, respectively. See demo script **test\_audpater\_sine\_wave.m** (Sect. XX) for an example on how to use this option.

Apart from generating a continuous sine wave, the user can also load an existing waveform of which the sampling rate equals srate\*downFact and it back by using the “playWave” option. See demo script **test\_audapter\_play\_wav.m** (Sect. XX) for an example on how to use the playWave option.

In addition, Audapter can also generate a sequence of short tone blips of adjustable durations, frequencies, amplitudes, onset/offset ramps and inter-tone intervals, through the “playToneSeq” option. Audapter can also write the waveform of the generatd tone sequence to a .wav file through the “writeToneSeq” option. See demo script **test\_audapter\_tone\_seq.m** (Sect. XX) for further details and examples of using these options.

The last, but the not least important, command-line option of Audapter covered is the “deviceName” option. It is used to select an audio interface to use. It should be especially useful when you have multiple ASIO-compatible sound cards attached to your computer. When Audapter starts a real-time operation, such as “start”, “playTone” or “playWave”, it searches for the sound card with name matching the value of the pre-set deviceName. If it fails to find such a device, it will report error and stop. This option can be called with the following syntax example.

Audapter('deviceName', 'MOTU MicroBook');

“MOTU MicroBook” is the default value of deviceName. If you use a different sound card, you’ll have to set it properly yourself.

Y1. Adjustable Parameters of Audapter

Table Y1 provides a list of all configurable parameters of Audapter. The values of these parameters can be set with the “setParam” option of Audapter. (see Sect. X1).

Table Y1. Adjustable parameters of Audapter

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter Name** | **Parameter Type** | **Description** | **Default value[[2]](#footnote-3)** |
| ***Part 1. Basic audio interface settings*** | | | |
| *downFact* | int | Downsampling factor. The downsampling is for reducing the computational load for meeting the real-time constraint. | 3 |
| *srate* | int | Sampling rate (Hz), after downsampling | 16000[[3]](#footnote-4) |
| *frameLen* | int | Frame length in number of samples, after downsampling. This value should be an integer power of two. | 32 |
| *nDelay* | int | Processing delay in number of frames. The delay is due to the way in which Audapter forms an internal processing window: an internal window consists of (2 \* nDelay - 1) input frames. During formant perturbation, the value of nDelay determines the feedback latency. Note that if other types of perturbation, such as pitch shifting and time warping is involved, the feedback latency may depend on other phase-vocoder-related settings. | 7 |
| *nWin* | int | Number of windows per frame. Each incoming frame is divided into nWin windows. | 1 |
| *fb* | int | Feedback mode.  0: mute (play no sound)  1: normal (speech only)  2: noise only  3: speech + noise.  4: speech-modulated noise.  Note: these options work only under real-time processing mode of Audapter, invoked through Audapter(1) or Audapter(‘start’). | 1 |
| *stereoMode* | Int | Two-channel audio output mode. This applies only to Audapter’s real-time processing mode.  0: Audio signal in left channel only; right channel muted  1: Identical audio signals in left and right channels  2: Audio signal in left channel; simulated TTL pulses for indicating pitch perturbation intervals in right channel. | 1 |
| *scale* | double | Output scaling factor. This can be used as a global (i.e., time-invariant) gain control. | 1.0 |
| ***Part 2. Basic signal processing and intensity calculations*** | | | |
| *preemp* | double | Pre-emphasis factor | 0.98 |
| *rmThr* | double | Short-time RMS threshold. This threshold is used to determine during which input frames the formants are tracked and shifted. See rows “data.rms” and “data.fmts” in Table X2 for further details. | 0.02. Note that this default value is by no means generalizable. It is selected more or less arbitrarily. The proper value of rmsThr depends on many factors such as microphone gain, speaker volume, identity of the vowel, etc. |
| *rmsRatio* | double | Threshold for short-time ratio between the smoothed unfiltered intensity value and the smoothed high-passed intensity value. Together with rmsThr, this parameter is involved in vowel detection for determining when formants are tracked and shifted. See rows “data.rms” and “data.fmts” in Table X2 for further details. | 1.3 |
| *rmsFF* | double | Forgetting factor (FF) for smoothing of short-time RMS intensity (rms\_o) to obtain the smoothed intensity (rms\_s). | 0.95 |
| ***Part 3. Formant tracking and shifting settings*** | | | |
| *nLPC* | int | Order of linear prediction (LPC). The number of LP coefficients will be nLPC + 1. | 15. For LP formant tracking to work properly, this value needs to be adjusted according to the sampling rate and the vocal-tract length of the speaker. Under 16000 Hz sampling rate (following downsampling), values 15 and 17 are recommended for female and male adult speakers, respectively. See mcode/ getAudapterDefaultParams. |
| *nFmts* | int | Number of formants to be shifted. | 2 |
| *nTracks* | int | Number of tracked formants. The 1st to the nTracks-th formants will be tracked. | 4 |
| *avgLen* | int | Length of the formant-frequency smoothing window (in number of frames). To disable smoothing of formant frequencies, use avgLen = 1. | 10. Ideally, the smoothing window width should be approximately equal to the pitch cycle. |
| *cepsWinWidth* | int | Low-pass cepstral liftering window size | Depends on the F0 of the speaker . See Section WW. |
| *minVowelLen* | int | Minimum allowed vowel duration (in number of frames). This is a somewhat obsolete parameter. It was used during prior single CVC syllable vowel formant perturbation experiments for preventing premature termination of perturbations. This capacity should have largely been superseded by OST (Sect. X3). | 60 |
| *f2Min* | double | Lower boundary of the perturbation field (Section W1) | 0.0 |
| *f2Max* | double | Upper boundary of the perturbation field (Section W1) | 0.0 |
| *f1Min* | double | Left boundary of the perturbation field (Section W1) | 0.0 |
| *f1Max* | double | Right boundary of the perturbation field (Section W1) | 0.0 |
| *LBk* | double | Slope of the tilted left boundary of the perturbation field (Section W1) | 0.0 |
| *LBb* | double | Intercept of the tilted right boundary of the perturbation field (Section W1) | 0.0 |
| *pertF2* | 1×257 double  array | The independent variable of the perturbation vector field (unit: mel or Hz, dependent on *bmelshift*). See Section W1. | All 0.0 |
| *pertAmp* | 1×257 double  array | The 1st dependent variable of the perturbation field: amplitude of the vectors.  When *bratioshift* = 0, *pertAmp* specifies the absolute amout of formant shifting (in either Hz or mel, depending on *bmelshift*).  When *bratioshift* = 1, *pertAmp* specifies the relative amount of formant shifting. See Section 1.2. (Section W1) | All 0.0 |
| *pertPhi* | 1×257  double array | The 2nd dependent variable of the perturbation field: orientation angle of the vectors (radians). See Section 1.2. | All 0.0 |
| ***Part 4. Sine wave (pure tone) generation and waveform playback*** | | | |
| *wgFreq* | double | Sine-wave generator frequency in Hz | 1000 |
| *wgAmp* | double | Sine-wave generator frequency (wav amp) | 0.1 |
| *wgTime* | double | Sine-wave generator initial time, used to set the initial phase. | 0 |
| *datapb* | double, 1×L array | Arbitrary sound waveform for playback. The sampling rate of the playback is srate \* downFact. Under the default setting, this sampling rate is 16000 × 3 = 48000 Hz. Therefore TransShiftMex can playback 4.8 seconds of sound. L must less than or equal to the maximum allowable playback length in # of samples, which can be obtained from Audapter by using syntax: Audapter('getMaxPBLen') | zeros(1, Audapter('getMaxPBLen')) |
|  | | | |
| *triallen* | double | Length of the trial in sec. “triallen” seconds past the onset of the trial, the playback gain is set to zero. | 2.5 |
| *ramplen* | double | Length of the onset and offset linear ramps in sec. | 0.05 |
| *afact* | double | α factor of the penalty function used in formant tracking. It is the weight on the bandwidth criterion (see Section 1.4). | 1 |
| *bfact* | double | β factor of the penalty function used in formant tracking. It is the weight on the a priori knowledge of the formant frequencies (see Section 1.4).. | 0.8 |
| *gfact* | double | γ factor of the penalty function used in formant tracking. It is the weight on the temporal smoothness criterion (see Section 1.4).. | 1 |
| *fn1* | double | A priori expectation of F1 (Hz) | 591 for male speakers; 675 for female speakers. (Note these values were selected for the Mandarin triphthong /iau/.) |
| *fn2* | double | A priori expectation of F2 (Hz) | 1314 for male speakers; 1392 for female speakers. (Note these values were selected for the Mandarin triphthong /iau/.) |
| *bgainadapt* | Boolean | A flag indicating whether gain adaptation is to be used (See Section 1.6) | 0 |
| *bshift* | Boolean | A flag indicating whether formant frequency shifting is to be used. Note: the following parameters must be properly set beforehand in order for the shifting to work: rmsthresh, rmsratio, f1min, f1max, f2min, f2max, lbk, lbb, pertf2, pertAmp, pertPhi, bdetect. | 1 |
| *btrack* | Boolean | A flag indicating whether the formant frequencies are tracked. It should almost always be set to 1. | 1 |
| *bdetect* | Boolean | A flag indicating whether TransShiftMex is to detect the time interval of a vowel. It should be set to 1 whenver bshift is set to 1. | 1 |
| *bweight* | Boolean | A flag indicating whether TransShiftMex will smooth the formant frequencies with an RMS-based weighted averaging. | 1 |
| *bcepslift* | Boolean | A flag indicating whether TransShiftMex will do the low-pass cepstral liftering. Note: cepswinwidth needs to be set properly in order for the cepstral liftering to work. | 1 |
| *bratioshift* | Boolean | A flag indicating whether the data in *pertAmp* are absolute (0) or relative (1) amount of formant shifting. See Section 1.2. | 0 |
| *bmelshift* | Boolean | A flag indicating whether the perturbation field is defined in Hz (0) or in mel (1). See Section 1.2. | 1 |

W1. The perturbation field



Figure W1. A schematic drawing of the perturbation field. The dashed lines shows the boundaries of the perturbation field. The arrows show the perturbation vectors. The shaded region is the perturbation field. A and θ are the magnitude and angle of the vector, which are both functions of F2.

The perturbation field is a region in the F1-F2 plane wherein F1 and F2 can be perturbed in a F2-dependent way. This mode of formant perturbation is currently incompatible with the OST and PCF files, described in Sections X3 and X4. In particular, OST and PCF take precedence over the perturbation field. If you have OST and PCF files loaded into Audapter, the software will use the information in those configuration files for determine the amount of F1 and F2 perturbation. Therefore, to prevent OST and PCF configurations from overriding the perturbation field, OST and PCF settngs need to be nullified prior to the onset of a trial that utilizes the perturbation field. This can be achieved through the commands:

Audapter('ost', '', 0);

Audapter('pcf', '', 0);

As shown schematically shown in Fig. W1, the location of the field is defined by five boundaries specified by six of Audapter’s adjustable parameters (Table Y1):

F1 ≥ *f1Min*; (1)

F1 ≤ *f1Max*; (2)

F2 ≥ *f2min*; (3)

F2 ≤ *f2max*; (4)

F2 ≥ *lbk* × F1 + *lbb*, if *lbk* ≤ 0; or F2 ≤ *lbk* × F1 + *lbb*, if *lbk* > 0; (5)

The units of f1min, f1max, f2min, f2max, lbb and lbk are either Hz or mel depends on another parameter, *bMelShift*. If *bMelShift* = 1, their units are mel; when *bmelshift* = 0; their units are Hz. In addition, the short-time intensity and spectrum need to satisfy certain conditions for the formant perturbation to happen. The “data.fmts” row in Table Y1 contains the definition of these conditions.

Detection of a vowel and shifting its formant frequencies is contingent on simultaneous satisfaction of the the intensity/spectrum condition and Equations (1) – (5) and. The boundary defined by Equation (5) is in general a tilted line (see Fig. W1), and may seem a little bit peculiar. It was added because it was found to improve triphthong detection reliability in the Standard Chinese triphthong perturbation study. This boundary can be disabled by setting both *lbb* and *lbk* to zero. Similarly, if your project is concerned with only a fixed amount perturbation to a steady-state vowel, you may wish not to use the boundaries *f1Min*, *f1Max*, *f2Min*, and *f2Max*, and rely only on the RMS criteria in Eqn. (6). This can be achieved by simply setting *f1Min* and *f2Min* to 0 and *f1Max* and *f2Max* to sufficiently large values (e.g., 5000).

The perturbation field is a vector field (arrows in Fig. W1). Each vector specifies how much F1 and F2 will be perturbed, respectively. Each vector is defined by a magnitude A and an angle φ, which corresponds to *pertAmp* and *pertPhi* in the list of adjustable parameters (Table W1). Both A and φ are functions of F2. *pertAmp* can be either an absolute amount of formant shift or a relative ratio for formant shift, depending on whether *bRatioShift* is set to 0 or 1. The angle pertPhi has a unit of radians and starts from the positive horizontal axis and increases in the counterclockwise dierection, in a fashion analogous to the complex plane. For example, if *bmelshift* = 0, *bratioshift* = 1, *pertAmp* = all 0.3’s and *pertPhi* = all 0’s, then the perturbation will be a uniform 30% increase in F1 of the vowel.

The mappings from F2 to A and φ are specified in the form of look-up tables (LUT) by the three parameters *pertf2*, *pertAmp* and *pertPhi*, which are all 1×257 vectors. The hard-coded number 257 may look a little peculiar. It is selected to enable efficient binary search for mapping unperturbed F2 values to perturbation vectors. Linear interpolation is used to calculate the magnitude and angle of the perturbation vectors. See the demo script “audapterDemo\_triphthong.m” for an example of how to use the perturbation field (see also Sect. 2.1.4)

The design of the perturbation is general enough to allow flexible F2-dependent perturbations. However, your project may concern with only fixed perturbation to a steady-state vowel, and hence not require this flexible setup. If that’s the case, you can simply set both *pertAmp* and *pertPhi* as constant. For example, if you want to introduce a 300-mel downward shift to the F1 of a steady-state vowel (e.g., /ε/), you can simply set *bmelshift* = 1, *bratioshift* = 0, and let *pertAmp* be a 1×257 vector of all 300’s and let *pertPhi* be a 1×257 vector of all π’s. Here, *pertf2* should be a 1×257 linear ramp from *f2min* to *f2max*.

You should also keep in mind that the parameters *f1min*, *f1max*, *f2min, f2max, lbk,* *lbb, pertf2,* and *pertAmp* all have units that are dependent on *bmelshift*, despite the fact that the formant frequency outputs in dataMat (See Section 1.1) and other parameters of TransShiftMex (e.g, *srate*, *fn1*, *fn2*, *wgfreq*, see Table 1) always have the unit of Hz.

X2. Data structure of the .mat files

The runExperiment script generates a .mat file for each trial. Each of those .mat files contains a variable by the name of “data”. This variable is obtained through the AudapterIO script. It is a structure containing a number of fields. The following is a description of the meaning of the fields.

Table X2. Data fields and their meanings in the output of AudapterIO('getData') and the saved data files

|  |  |
| --- | --- |
| **Field name** | **Meaning** |
| data.signalIn | Input signal after downsampling. This can either be from the audio interface for an online trial, or from the “runFrame” option for an offline trial. |
| data.signalOut | Output signal, before upsampling, possibly perturbed, depending on the perturbation configuration. If an online trial is involved, this is the signal delivered to the output channel of the audio interface, i.e., to the headphone |
| data.intervals | The onset sample numbers (integers) for the data frames. |
| data.rms | Column 1: Short-time root-mean-square (RMS) intensity values, smoothed, denoted **rms\_s**. The amount of smoothing is determined by the value of the parameter “rmsFF” (RMS forget factor).  Column 2: Short-time RMS intensity, pre-emphasized (i.e., high-pass filtered) and smoothed, denoted **rms\_p**. The amount of pre-empahsis (i.e., high-pass filtering) is determined by parameter “preemp”, whose value is 0.98 by default. The amount of smoothing is the same as in column 1.  Column 3: Non-smoothed, non-pre-emphasized short-time RMS intensity, denoted **rms\_o.** |
| data.fmts | Formant frequencies. The values are non-zero only for the moments in which the smoothed short-time RMS intensity rms\_s and the ratio between rms\_s and rms\_p (i.e., rms\_ratio) satisfy the following condition:  (rms\_s > 2 \* rmsThr) && (rms\_ratio > rmsRatioThresh / 1.3)  ||  (2 \* rmsThr > rms\_s > rmsThr) && (rms\_ratio > rmsRatioThresh)  The use of rmsRatioThresh in this condition is for excluding the unvoiced segments from the formant tracking. The value of rms\_ratio should be greater during vowels and other voiced sounds than during unvoiced sounds such as fricatives. If you do not wish to include this ratio in the determination of formant-tracking intervals, you can set rmsRatioThresh to 0. |
| data.rads | The phase angles of the poles in the Z-plane in the LP results. |
| data.dfmts | Rate of change (velocity) of the formants. |
| data.sfmts | Formant frequencies in the output signal (i.e., auditory feedback). The values are non-zero during and only during the time intervals involving non-zero formant shifts. |
| data.rms\_slope | Short-time slope of rms\_s (the smoothed short-time RMS intensity values). This slope has a unit of s-1, and is obtained through Pearson linear regression of the rms\_s values against the time. The size of the regression window is adjustable in the first part of an OST file (see Sect. XX). The reason why this parameter is configured in OST files is because rms\_slope is used in certain heuristic rules in OST. |
| data.ost\_stat | OST status number, determined by using the configurations in the OST file loaded into Audapter. If no OST file has been loaded, the values in ost\_stat will be all zero. |
| data.pitchShiftRatio | The ratio of pitch shift as a function of time. 1.0 - no shift; >1.0 - upshift; <1.0 downshift. |
| data.params | Parameter settings during the trial. See Table XX for a full description of the parameters. |
| data.vowelLevel (may or may not exist depending on the version of runExperiment.m) | Mean intensity of the vowel (in dB SPL A). |
| data.uiConfig (may or may not exist depending on the version of runExperiment.m) | GUI configuration during this trial |
| data.timeStamp | A time stamp created shortly after the end of the trial |

X3. Online Status Tracking (OST)

For certain psychophysical AFP applications, you may wish to use a multisyllabic speech utterance and impose the perturbation during specific sounds or syllables of the utterance. Online status tracking (OST) is a functionality of Audapter that serves this purpose. You can design a set of heuristic rules based on signal properties such as intensity to detect the onset and offset of various sounds in the utterance. With OST, Audapter assigns an integer status number to each input frame in real time. In post-processing, these state numbers are stored in data.ost\_stat (see Sect. X.X). You can map these state numbers to various types of perturbations in by using perturbation configuration (PCF) files, a topic covered in Sect. X.X. Therefore OST and PCF work together to enable the online automatic triggering of perturbation events.

An OST file is an ASCII text file that configures the set of heuristic rules for tracking the progress of a speech utterance. It can be loaded into Audpater with the 'ost' option:

Audapter('ost', ost\_fn, 0);

The second input argument is the name of the OST configuration file. The third argument is a Boolean flag for verbose mode.

Code Sample X below is an example OST file. You should follow this formant when creating your own OST files. This file consists of three parts. Part 1 is a single line that begins with rmsSlopeWin =. This configures the window size (in seconds) for computing the slopes of short-time RMS intensity. Part 2 begins with a line such as n = 3. This compulsory line specifies the number of OST rules in the OST file. The number of following lines in this part must match the value of n. Each of the following lines specifies a tracking rule. These rules are engaged sequentially during an online trial.

**Code Sample 1.** An example online status tracking (OST) configuration file.

|  |
| --- |
| # Online status tracking (OST) configuration file  rmsSlopeWin = 0.030000  # Main section: heuristic rules for tracking  n = 3  0 INTENSITY\_RISE\_HOLD 0.02 0.0200 {} # Detect the onset of the first word  2 INTENSITY\_FALL 0.01 0.0100 {} # Detect the end of the first word  4 OST\_END NaN NaN {}  # maxIOICfg  n = 1  2 0.2 4 |

Each line of OST rule consists of five fields that are words or numbers, separated by single spaces. The first field is the starting state (ost\_stat) value. The second field selects the mode of tracking. It can be either a number from the first column of Table 2 or an all-upper-case string from the second column of the same table. Table 2 lists the currently supported modes of tracking. They are based mostly on short-time intensity, its rate of change (slope), and the ratio of spectral intensity in high- and low-freuency bands (e.g., mode numbers 30 and 31). If you wish to include new and/or more sophisticated modes of tracking, changes to the C++ source code of Audapter will have to be made. Specifically, the OST functions are package in header and source files ost.h and ost.cpp. The third and fourth fields of the line are the two mode-specific parameters that can be configured by the user. For example, in the tracking INTENSITY\_RISE\_HOLD, the user needs to set the intensity threshold and the hold duration in the third and fourth fields, respectively. The fourth column of Table 2 contains descriptions of these parameters. Note that some tracking modes are associated with two parameters, while others are associated with one or none. In the cases wherein fewer than two parameters are required, use the first several ones of the third and fourth fields, and leave the rest at NaN or arbitrary values. The fifth field of the line is a pair of curly brackets. This field serves no purpose in the current version of Audapter, but are reserved for potential future uses.

Each tracking mode is associated with a fixed increment in status number at the end of the mode. For example, the mode INTENSITY\_RISE\_HOLD involves an increment of 2 from the beginning to the end of the tracking. The last column of Table 2 lists these increment amounts. In Part 2 of the OST file, the first fields of the consecutive lines must match these increment values, otherwise unexpected and unpredictable tracking errors may occur. In other words, if the onset ost\_stat value of an INTENSITY\_RISE\_HOLD rules is 0, for example, then the beginning ost\_value of the next rule, specified in the following line, must be 2. It should also be noted that each set of OST rules must end with a rule of the OST\_END tracking mode (e.g., see the code sample above).

Part 3 of the OST file is for the maximum-inter-onset-interval (maxIOI) mode of tracking. The maxIOI mode of tracking is a quite ad hoc way of dealing with possible tracking failure. It is essentially a way of telling the OST module of Audapter that you should proceed to a different state forcefully, regardless of the tracking rule, if a certain amount of time has elapsed from the onset of a given state. As you probably have come to realize, this is not an elegant way of approaching the tracking problem and should be used only as a last resort when necessary.

This part begins with a line which specifies the number of maxIOI rules. The number of the trailing lines in this section must match the value of n in this first line. In each of the trailing lines, there are three numbers. The first number is the onset ost\_stat number. The second one is the maximum wait time, in seconds. The third onset is the value of ost\_stat that Audapter will automatically jump to when this wait period has elapsed.

Consecutive parts in the OST file are separated by blank lines. You can insert comment lines in OST file. These comment lines should begin with the hash (#) character. You can also add comments to the end of uncommented lines (as in certain programming languages such as Python or MATLAB).

**Table 2.** A list of supported heuristic modes for online status tracking (OST)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mode Number** | **Mode Name** | **Description and Example Usage** | **Parameters** | **Increment in OST state number** |
| 0 | OST\_END | Serves as an ending rule. Once a rule of this mode is reached, OST halts and the status number freezes at the current value until the end of the trial. Each OST file **must** end with this rule. | (None) | 0 |
| 1 | ELAPSED\_TIME | Elapsed time from previous state  Example: Wait for a fixed amount of time (e.g., 100 ms) after voicing onset | prm1: duration | +1 |
| 5 | INTENSITY\_RISE\_HOLD | Crossing an intensity (RMS) threshold from below and hold  Example: Detect the onset of a vowel or a voiced consonant-vowel cluster | prm1: rmsThresh  prm2: minDur (s) | +2 |
| 6 | INTENSITY\_RISE\_HOLD\_POS\_SLOPE | Crossing an intensity threshold from below and hold, during positive RMS slope  Example: Detect the onset of a vowel, with added security | prm1: rmsThresh  prm2: minDur (s) | +2 |
| 10 | POS\_INTENSITY\_SLOPE\_STRETCH | Stretch of positive intensity slope, with only a stretch count threshold  Example: This is still another way to detect the onset of a vowel or voiced consonant | prm1: stretchCntThresh | +2 |
| 11 | NEG\_INTENSITY\_SLOPE\_STRETCH\_SPAN | Stretch of negative intensity slope, with a stretch count threshold and a stretch span threshold  Example: Detect the end of a vowel or the silent interval between a vowel and a stop consonant | prm1: stretchCntThresh  prm2: stretchSpanThresh | +2 |
| 20 | INTENSITY\_FALL | Fall from a certain intensity threshold  Example: Detect the end of a vowel | prm1: rmsThresh  prm2: minDur (s) | +1 |
| 30 | INTENSITY\_RATIO\_RISE | Intensity ratio (TODO: Explain) cross from below and hold  Example: Detect the onset of a sibilant (e.g., [s]) | prm1: rmsRatioThresh  prm2: minDur (s) | +2 |
| 31 | INTENSITY\_RATIO\_FALL\_HOLD | Intensity ratio: fall from a threshold and hold  Example: Detect the end of a sibilant (e.g., [s]) | prm1: rmsRatioThresh  prm2: minDur (s) | +2 |

X4. Perturbation configuration (PCF) files

Once you have the OST heuristics configured, the next step in enabling focal perturbation of AF is supplying Audapter with a perturbation configuration (PCF) file. Similar to OST files, you can input a PCF into Audapter by using the following syntax:

Audapter('pcf', pcf\_fn, 0);

The perturbation settings in a PCF file is divided into two parts:

1. Time warping settings
2. Settings for pitch, intensity and formant perturbations to be delivered at each specific state number

This two-part organization is reflected in the structure of the PCF files. See the following code example:

Code Sample X2. An example online status tracking (OST) configuration file.

**Code Sample 2.** An example perturbation configuration (PCF) file.

|  |
| --- |
| # Section 1 (Time warping): (state number), tBegin, rate1, dur1, durHold, rate2  2  0.94, 0.1, 0.1, 0.1, 1.5 # Time warping 1  1.502, 0.1, 0.1, 0.1, 1.5 # Time warping 2  # Section 2: stat pitchShift(st) gainShift(dB) fmtPertAmp fmtPertPhi(rad)  6  0, 0.0, 0, 0, 0  1, 0.0, 0, 0, 0  2, 0.0, 0, 0, 0  3, 0.0, 0, 0, 0  4, 0.0, 0, 0, 0  5, 2.0, 0, 0, 0 # Two-semitone upward pitch perturbation during the last word |

This example PCF file defines two types of perturbations during a single utterance: two temporally non-overlapping time warps in Section 1 and a two-semitone pitch shift in Section 2.

The syntax of Section 1 (time warping) is as follows. You begin by including a line consisting of a single positive integer, specifying the number of time warping events in the utterance. Following this line, the correct number of lines need to be entered, defining details of each time-warping event. There are two possible formatting for each line. In the first format, five numbers are included in the line. These five numbers provide Audapter with the following pieces of information, respectively,

1. *tBegin*: The onset time of the warp event (relative to utterance onset)
2. *rate1*: The rate of initial time warping, with <1 being time dilation. In fact, this number has to be ≤1.0 in order for the system to be causal.
3. *dur1*: Duration of the initial warping at *rate1*.
4. *durHold*: Duration of the hold (i.e., no-warping) period following *dur1*
5. *rate2:* rate of the time warping in the catch-up (or recovery) period. This number has to be ≥1.0.

In total, a time warping event configured in this format lasts for a total duration of

tBegin + dur1 + durHold + dur2

wherein

dur2 = (1 - rate1) / (rate2 - 1) × dur1

When more than one warping events are specified in this format, Audapter will check to make sure that there are no temporal overlap between them. It will report error if an overlap exists.

In format 2, six, instead of five, numbers are included in each line. The first number should be an integer and it specifies the OST status number the time-warping event resides in. In this format, the onset timing of the warping event is relative to the onset time of the specified status number, not the onset of the utterance. The following five numbers have the same meaning as the numbers in line format 1. As in formant 1, Audapter will look for temporal overlaps between time-warping events of the same OST status number and report an error if it finds any. However, because the onset timing of different OST status numbers cannot be predicted beforehand, Audapter will not attempt to check overlaps between time-warping events between different OST numbers or between time-warping events specified with different formats.

Note that the sample PCF file above includes only format 1.

In Section 2 of the PCF file, you define the amount of the following three types of non-time-warping perturbations at each OST status number. This section needs to start with a line consisting of a single integer that specifies the total number of different OST status numbers. Since OST status numbers always begin at 0, this integer should be one plus the maximum OST status number in the OST file you are using.

The following lines have a fixed format, namely five numbers separated by commas and/or spaces. These five numbers, in order, define the following perturbation settings:

1. 1st number: The OST status number. Note that this has to be sequential. You cannot skip status numbers or include status numbers that are outside the possible range.
2. 2nd number: The amount of pitch shifting, in semitones. Positive values correspond to upward pitch shifts, while negative ones corresponds to downward shifts.
3. 3rd number: The amount of intensity perturbation, in dB.
4. 4th number: The magnitude of joint F1-F2 perturbation vector, in the formant plane spanned by F1 and F2. The unit of this depends on the bRatioShift parameter set in Audapter (see Section XX and Table XX). If it is set to 0 (false), the unit will be Hz. Otherwise this number is dimensionless and specifies the ratio (fraction) of formant shifts.
5. 5th number: The angle of the formant perturbation vector, in radians. For example, an angle of 0 leads to a pure upward F1 perturbation. An angle of -π/2 leads to a pure downward F2 perturbation. An angle of π/4 is used to specify equal amounts of perturbation to F1 and F2.

As in an OST file, you can add comments to the PCF file with the “#” character (see Code Sample X2).

References

Boucek M. (2007). The nature of planned acoustic trajectories. Unpublished M.S. thesis. Universität Karlsruhe.

Cai S, Boucek M, Ghosh SS, Guenther FH, Perkell JS. (2008). A system for online dynamic perturbation of formant frequencies and results from perturbation of the Mandarin triphthong /iau/. In *Proceedings of the 8th Intl. Seminar on Speech Production, Strasbourg, France*, Dec. 8 - 12, 2008. pp. 65-68.

Cai S, Beal DS, Ghosh SS, Guenther FH, Perkell JS. (In press). Impaired timing adjustments in response to time-varying auditory perturbation during connected speech production in persons who stutter. Brain Lang.

Cai S, Beal DS, Ghosh SS, Tiede MK, Guenther FH, Perkell JS. (2012). Weak responses to auditory feedback perturbation during articulation in persons who stutter: Evidence for abnormal auditory-motor transformation. PLoS ONE. 7(7):e41830.

Cai S, Ghosh SS, Guenther FH, Perkell JS. (2010). Adaptive auditory feedback control of the production of the formant trajectories in the Mandarin triphthong /iau/ and its patterns of generalization. J. Acoust. Soc. Am. 128(4):2033-2048.

Cai S, Ghosh SS, Guenther FH, Perkell JS. (2011). Focal manipulations of formant trajectories reveal a role of auditory feedback in the online control of both within-syllable and between-syllable speech timing. J. Neurosci. 31(45):16483-16490.

Tourville JA, Cai S, Guenther FH (2013) Exploring auditory-motor interactions in normal and disordered speech. *Proceedings of Meeting on Acoustics*. 9:060180. Presented at the *165th Meeting of the Acoustical Society of America*, Montreal, Quebec, Canada, June 2 – June 7, 2013.

Xia K, Espy-Wilson C. (2000). A new strategy of formant tracking based on dynamic programming. In ICSLP2000, Beijing, China, October 2000.

Appendix 1. Instructions on obtaining and setting up the Audapter package

Appendix 2. Instructions on building the core MEX program of Audapter in Microsoft Visual C++

1. Download the C++ code from [TODO].   
   (Courtesy notice: Please check with the author of the manual before sharing this link with other labs or research groups.)
2. Extract the Audapter-2.0 directory in the zip archive to C:/speechres/audapter
3. Open the solution in Visual C++ 2010 or later
4. In Visual C++, select the correct architecture (win32 or x64). Note that you may need to manually set the linker output format to either mexw32 or mexw64, depending on your architecture.
5. There are a number of configurations in the solution, such as “Release”, “Release\_NWU” and so on. The main difference between these configurations are the include and library paths. You can use an existing configuration and make necessary modifications to it. You can modify the configuration by right-clicking “Audapter” in the Solution Explorer and select “properties”.  
   Below are the most important settings for ensuring successful debugging and compiling:
   1. General / Configuration type = Dynamic Library (.dll)
   2. C/C++ / Additional include directories should contain $(SolutionDir)\audioIO, $(SolutionDir)\SibShift and C:\Program Files\MATLAB\R2011a\extern\include. The last directory may vary depending on your MATLAB installation path.
   3. Linker / General / Additional library dependencies should include C:\Program Files\MATLAB\R2011a\extern\lib\win32\microsoft. This directory may vary depending on your MATLAB installation path and your CPU architecture.
   4. Linker / Input / Additional dependencies should include libmx.lib, libmex.lib and libmax.lib
   5. Other settings that are described in this helpful webpage for guiding beginners through MEX building in VC++: <http://coachk.cs.ucf.edu/GPGPU/Compiling_a_MEX_file_with_Visual_Studio2.htm>
6. Use menu option: “Build 🡪 Rebuild” to rebuild both the Audapter and audioIO projects in the solution. A number of warning messages are expected. Most of them should be harmless and can be neglected.

1. List of abbreviations: F0 – fundamental frequency; F1- 1st formant frequency; F2 – 2nd formant frequency; FF – Forgetting factor; LP – Linear prediction; OST – Online status tracking; PCF – Perturbation configuration; RMS – Root mean square. [↑](#footnote-ref-2)
2. These default values of downFact, srate and frameLen are the parameter values that Audapter automatically take after construction. It can be found in the constructor of Audapter (Audapter::Audapter) in the C++ source code. [↑](#footnote-ref-3)
3. 48000 Hz downsampled by a factor of 3. Note that this value is set for MOTU MicroBook. For other audio interfaces, other values may need to be used. [↑](#footnote-ref-4)