Project ID: XXXX

Generic MATLAB Toolbox v1.1

Software User manual

Table of Contents

[2 Introduction and Background - 2 -](#_Toc52962441)

[2.1 Goals of the GMT - 2 -](#_Toc52962442)

[2.2 List of abbreviations - 2 -](#_Toc52962443)

[3 Sound coding with the GMT - 3 -](#_Toc52962444)

[3.1 Demo - 3 -](#_Toc52962445)

[3.2 Initialisation of a GMT script - 3 -](#_Toc52962446)

[3.3 Add a processing block - 4 -](#_Toc52962447)

[3.4 Basic block structure - 5 -](#_Toc52962448)

[3.5 Defining connections between blocks - 6 -](#_Toc52962449)

[3.6 Setting and accessing data - 7 -](#_Toc52962450)

[3.7 Running a GMT coding strategy - 8 -](#_Toc52962451)

[4 Essential Processing Blocks - 9 -](#_Toc52962452)

[4.1 SpecRes Processing Blocks - 9 -](#_Toc52962453)

[4.2 Fundamental Strategy Parameters - 11 -](#_Toc52962454)

[4.3 Visualisation tools - 12 -](#_Toc52962455)

[4.4 Noise Vocoder CI Simulation - 12 -](#_Toc52962456)

[5 Contact - 14 -](#_Toc52962457)

[6 References - 14 -](#_Toc52962458)

# Introduction and Background

The Generic MATLAB Toolbox (GMT) is a collection of MATLAB scripts simulating the processing steps of the Advanced Bionics (AB) cochlear implant (CI) system. This document gives an overview over the goals and structure of the GMT, guidelines how to use the provided environment, and an annotated demonstrations of a simple CI coding strategy.

## Goals of the GMT

The target group of the GMT are researchers and signal processing engineers familiar with the MATLAB programming environment (The MathWorks, Inc., Natick, MA) who want to investigate and develop sound coding algorithms for AB CI speech processors. Specifically, the goal of this toolbox is to facilitate

* **Algorithm Development**  
  The structure of the GMT provides easy, highly modular access to all steps of the CI processing chain, enabling the implementations of new or improved sound coding blocks at any stage with minimal structural impact on the remaining system. The MATLAB environment with its high-level language and extensive numerical computation libraries allows for fast prototyping without having to worry about particular restrictions of the various product DSP platforms.
* **Demonstration and Education**   
  Based on the example scripts that are provided with the toolbox, the GMT allows to create a wide range of visual and aural demonstrations of the operations of CI sound coding strategies and their constituent processing steps. The provided vocoder simulation provides a valuable tool for demonstrating the acoustic percept of CI users to normal-hearing listeners.
* **Perceptual Evaluation**   
  FFT re-synthesis or vocoder simulations may be used for perceptual and performance evaluations of CI sound coding algorithms with normal-hearing listeners in order to gauge their potential benefit for CI users during early phases of development.

## List of abbreviations

|  |  |
| --- | --- |
| Abbreviation | Description |
| AB | Advanced Bionics |
| CI | Cochlear implant |
| FFT | Fast Fourier transform |
| FT | Forward telemetry |
| M, M-level | Electric stimulation level evoking the most comfortable loudness percept |
| pps | Pulses per second |
| STFT | Short-time Fourier transform |
| T, T-level | Electric stimulation level at perceptual hearing threshold |

# Sound coding with the GMT

The GMT was developed to provide an easily accessible and flexible way to simulate and experiment with new developments in the CI processing chain based on the foundation of AB’s F120 coding strategy and its experimental predecessor, SpecRes. In this chapter, a brief overview is given and the general structure of the GMT and its use is introduced by describing the sections of the script demo1\_SpecRes.m which is delivered with the GMT. To follow the explanation in this chapter, it is recommended to open the file demo1\_SpecRes.m with in the MATLAB editor. Familiarity with the basic concepts of object-oriented programming and their implementation in MATLAB is assumed and will greatly enhance the ability to adapt and generalize from the provided examples.

## Demo

The demo can be executed by typing demo1\_SpecRes in the command window of MATLAB. To be able to execute the demo, it is recommended to set an environment variable in windows called GMTROOT with the directory as variable value. The output is the electrodogram shown at the bottom in Figure 1 and the signal flow chart in Figure 2. In this example, all default values are used to generate the electrodogram.

## Initialisation of a GMT script

clear all;

clear classes;

After clearing the workspace to avoid any influence of former MATLAB processing, the initGmtClassPath function reads the GMT base directory from the system environment variable GMTROOT.

initGmtClassPath;

file = 'yes.wav';

The file that is used as the input in this script is called ‘yes.wav’. The input file has to be a wav-file. The sample rate is not important because the file gets resampled to the sample rate of the AB speech processor in the first stages of the processing.

strat = FftStrategy();

Here, an instance of the the FftStrategy class is created and its handle assigned to the variable strat. The FftStrategy class itself does not implement any of the actual signal processing functionality, but provides first and foremost a backbone structure for managing the signal flow between arbitrary processing steps that will be added in the following. FftStrategy also defines some basic attributes of an FFT-based coding strategy (such as SpecRes or F120) to be shared across all those processing steps. For non-FFT-based strategies, the simpler Strategy class may be used instead (of which FftStrategy is a sub-class).



Figure 1: Time domain signal (top) of the word "yes" and the corresponding electrodogram (bottom) generated with the GMT. The height of the lines represents the amount of current that is emitted by the respective electrode.

## Add a processing block

A specific processing block (or, unit) is added to a coding strategy by instantiating an object of the respective class (e.g. ReadWavUnit), which needs to be a sub-class of the ProcUnit class. ProcUnit itself defines the overall interface and signal-flow logic interacting with the parent coding strategy object. All specific functionality is implemented in its sub-classes, a number of which are used in demo1\_SpecRes.m. As parameters, the handle of the containing strategy object (here: strat) and a label string for the block (e.g. 'SRC’) must be provided when instantiating any processing block. Further parameters may be required depending on the specific processing block being instantiated. The overall structure for a block called “Example” is as follows:

block = ExampleUnit(parent Strategy object, block label, additional parameter 1, additional parameter 2, …)

Typically, a coding strategy will encompass a number of different processing blocks. Note that blocks do not need to be instantiated in order of execution (and conversely, the order in which blocks are defined does not fully define the order in which they will be executed later on; cf. section ‎3.7).

In the example script demo1\_SpecRes.m, the following blocks are added

src = ReadWavUnit(strat, 'SRC', 'Sounds\AzBio\_3sent.wav');

mix = AudioMixerUnit(strat, 'MIX', 1, 65, 'rms');

pre = HarmonyPreemphasisUnit(strat, 'PRE');

agc = DualLoopTdAgcUnit(strat, 'AGC');

wb = WinBufUnit(strat, 'WB');

fftfb = FftFilterbankUnit(strat, 'FFT');

env = HilbertEnvelopeUnit(strat, 'ENV');

spl = SpecPeakLocatorUnit(strat, 'SPL');

csw = CurrentSteeringWeightsUnit(strat, 'CSW');

csynth = CarrierSynthesisUnit(strat, 'CSYNTH');

map = F120MappingUnit(strat, 'MAP');

plotter = PlotF120ElectrodogramUnit(strat, 'PLT');

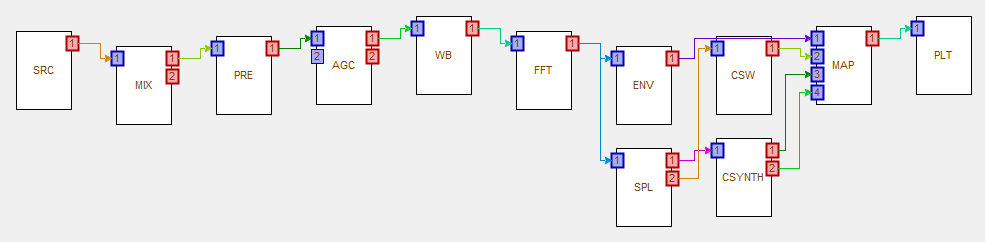


Figure 2: Flowchart of the GMT implementation of the SpecRes strategy.

Figure 2 depicts the blocks that were added in this example, including connections from output data ports (red) to input data port (blue) established as described in section ‎3.5.

## Basic block structure

The GMT follows an object oriented approach in its basic block structure. Per convention, each block consists of at least two scripts: ExampleUnit.m and exampleFunc.m. ExampleUnit.m contains the actual class definition including the class attributes, the constructor method and a specific implementation of the run() method defined by the inherited ProcUnit interface. In many processing blocks provided with the GMT, the run() method will simply collect the block’s inputs and then call the function exampleFunc(…) which implement the actual, specific functionality of that block. This convention is employed to allow users to access the functionality of each block without being forced to use the object-oriented framework, and is highly encouraged for any future additions to the library of blocks. exampleFunc should take an argument of type ExampleUnit that contains all relevant configuration parameters. When used outside the object oriented framework, the user can simply replace the parameter object (of type ExampleUnit) by a struct with matching fields. For documentation purposes, it is important that both the class and the function are well-documented in terms of functionality, input/output signals and parameters incl. units and default values. Here is an example for how a class and its corresponding function should be documented:

FftFilterbankUnit.m

% FftFilterbankUnit < ProcUnit

% Compute FFT of buffered signal segments.

%

% FftFilterbankUnit properties

% combineDcNy - Combine DC and Nyquist bins into single complex 1st bin? [boolean] [false]

% if true, then bin #1 = .5\*(DC+NY + i(DC-NY)) and bin #NFFT/2+1 = 0

% compensateFftLength - Divide FFT coefficients by nFft/2? [boolean] [false]

% includeNyquistBin - return bin #nFft/2+1 in output? [boolean] [false]

%

% Input Ports:

% #1 - buffered signal frames, nFft x nFrames

%

% Output Ports:

% #1 - FFT coefficient matrix, (nFft/2) x nFrames (default) or (nFft/2+1) x nFrames,

% depending on includeNyquistBin

%

% See also: fftFilterBankFunc.m, ProcUnit.m

fftFilterbankFunc.m

% X = fftFilterbankFunc(par, buf)

% Compute FFT along each column of a matrix buffered signal segments.

%

% INPUT:

% par - parameter object / struct

% buf - nFft x nFrames matrix of windowed signal buffers

%

% FIELDS FOR par:

% parent.nFft - FFT length [int > 0]

% combineDcNy - Combine DC and Nyquist bins into single complex 1st bin? [boolean]

% if true, then bin #1 = .5\*(DC+NY + i(DC-NY)) and bin #(nFfft /2+1) = 0

% compensateFftLength - Divide FFT coefficients by nFft/2? [boolean]

% includeNyquistBin - return bin #nFft/2+1 in output? [boolean]

%

% OUTPUT:

% X - FFT coefficient matrix, (nFft/2) x nFrames or (nFft/2+1) x nFrames depending on includeNyquistBin

## Defining connections between blocks

After a block has been added to a coding strategy object in the manner described above, its signal input and output ports can and typically need to be connected to the ports of other blocks. By calling strat.connect(…) it is possible to connect the output port of a source block to the input port of a destination block:

strat.connect(block\_src [, port\_src], block\_dest [, port\_dest]).

If no input or output port number are provided, port number 1 will be assumed implicitly. block\_src and block\_dest can be either be object handles or the label string of processing blocks contained in the *same* coding strategy. Connections between units contained in different coding strategies are not supported. The units and connections within a strategy have to form a directed acyclic graph, i.e. loops in the network of connections (spanning any number of units) are not allowed. Note that connections can be added in arbitrary order, and the instantiation of block and the definition of block-to-block connection may be freely interspersed (cf. section ‎3.7).

In demo1\_SpecRes.m, the connections between the processing blocks are defined as follows:

strat.connect(src, mix);

strat.connect(mix, pre);

strat.connect(pre, agc);

strat.connect(agc, wb);

strat.connect(wb, fftfb);

strat.connect(fftfb, env);

strat.connect(fftfb, spl);

strat.connect(spl, csynth);

strat.connect(spl, 2, csw);

strat.connect(env, map);

strat.connect(csw, map, 2);

strat.connect(csynth, map, 3);

strat.connect(csynth, 2, map, 4);

strat.connect(map, plotter);

Figure 2 shows the connections between the blocks. Arrows in different colors indicate that there is a crossing between two lines. To avoid confusions in the signal flow chart, different colors are used for the respective arrows.

## Setting and accessing data

Before executing an entire coding strategy, it is required that the input(s) of any block which is not itself a signal source be connected (and hence, populated automatically during execution) to the output port of another block, or set manually in advance. It is possible to set the desired input to a block by calling the parameter of a block as follows:

block.setInput(input\_port\_nr, input\_data);

Accessing the output data of each block is very similar to setting data. In order to read the most recent output(s) of any processing block, use the following function call:

data\_out = block.getOutput(output\_port\_nr)

## Running a GMT coding strategy

After all processing blocks are instantiated, their parameters set, their inter-connections established and any required input signals defined, the entire coding strategy can be executed by calling

strat.run();

The strategy’s run() method will in turn execute the run() method of every processing block it contains and forwarded all generated outputs according to the pre-defined connections between the units. The appropriate order of execution amongst the units is determined automatically by the coding strategy based on the connection structure, ensuring that all inputs of a block (provided through connections within the strategy) are computed and made available before calling the block’s run() method. The acyclic structure of connections ensures that such an order can always be determined. This means that the execution order cannot be freely defined or controlled by the user, but the order in which units are instantiated may influence the execution order (amongst units of the same depth in the connection graph) to some degree.

# Essential Processing Blocks

All building blocks necessary to faithfully emulate the SpecRes processing strategy (Nogueira et al, 2009) are supplied with the GMT. Figure 2 shows the block diagram of SpecRes as implemented in the GMT. The processing blocks are described in the following.

## SpecRes Processing Blocks



Figure 3: Output of the blocks ReadWavUnit (row 1), FftFilterbankUnit (row 2), HilbertEnvUnit (row 3) and PlotF120Electrodogram (row 4) for the speech token "yes".

ReadWavUnit

In ReadWavUnit (‘SRC’), the input wav-file is read and resampled to the sampling rate of the CI speech processor. The output is the resampled input wav-file. In Figure 3, the output of the block RW\_1 to the input wav file “yes” is shown in the first row.

AudioMixerUnit

In AudioMixerUnit (‘MIX’), a number of input signals are scaled to a desired target level and added, allowing for numerous options regarding onset delay and wrapping behavior.

HarmonyPreemphasisUnit

In HarmonyPreemphasisUnit (‘PRE’) the input signal is high-pass filtered, with filter characteristics matching the pre-emphasis filter applied to the audio inputs (microphones and auxiliary) of the Harmony speech processor.

DualLoopTdAgcUnit

The DualLoopTdAgcUnit (‘AGC’) implements a broadband dual-loop adaptive gain control (AGC) algorithm as described in Boyle et al., 2009. Signals above a certain threshold input level, the knee-point, are compressed with a high compression ratio and slow speed (maintaining the natural crest factor of speech), except when an abrupt increases in signal level triggers the fast loop designed to protect the CI user from sudden loudness discomfort. Outputs are the compressed audio signal and the gain applied.

WinBufUnit

In WinBufUnit (‘WB’), the sampled input signal is buffered into frames (according the FFT length and hop-size defined in the parent FftStrategy object) and each frame multiplied with a time-domain window. Different window functions are available. The output is a matrix containing the windowed time-domain signal frames.

FftFilterbankUnit

In FftFilterbankUnit (‘FFT’), the FFT of the windowed signal buffers is computed for each time frame. Hence, its output is a short-time FFT (STFT) representation of the original signal. Only the content of the nFFt/2 positive frequencies is provided (due to the FFT symmetry for real-valued signals). In Figure 3, the output of the block FFT for the word “yes” is shown in the second row.

HilbertEnvelopeUnit

In HilbertEnvelopeUnit (‘ENV’), the Hilbert envelope for a number of channels is extracted at the mid-point of each signal frame by summation of the short-time FFT bins according to Nogueira et al., 2009. The number of channels are determined by nChan, the allocation table by nBinLims, and the lowest bin that is taken into account in the allocation table by startBin, all of which are defined as attributes in the FftStrategy class (cf. section ‎4.2). In Figure 3, the output of the block ENV of the word “yes” is shown in the third row.

SpecPeakLocatorUnit

SpecPeakLocatorUnit (‘SPL’) takes the STFT coefficients as input and estimates the dominant peak frequency for each channel and the corresponding electrode location along the cochlear axis. For more details on the frequency estimation, the reader is referred to Nogueira et al, 2009.

CurrentSteeringWeightsUnit

In CurrentSteeringWeightsUnit (‘CSW’), the current steering weights for the pairs of electrodes are determined. The estimates of the frequency-dependent cochlea position are used as input to this block. The output are the current weights for all electrodes.

CarrierSynthesisUnit

In contrast to the other blocks so far, CarrierSynthesisUnit (‘CSYNTH’) operates at the forward-telemetry rate, i.e. the rate corresponding to one full implant stimulation cycle across all channels. The FT rate is patient- and program-specific and is determined by parameters such as the stimulation pulse-width and number of active electrodes. For low-frequency channels, the carrier is square-wave modulated with a rate determined by the current peak-frequency estimates in the respective channels (Nogueira et al., 2009), thereby providing a matching temporal fine-structure cue. In order for subsequent blocks to be able to associates signals sampled at FT rate with those sampled at audio frame rate, an index vector is also generated by this block, mapping FT frame indices to the corresponding audio frame indices.

F120MappingUnit

In F120MappingUnit (‘MAP’), the carrier signal for each channel is modulated with the corresponding Hilbert envelope. The current weights are applied to determine how the distribute the resulting stimulation level across the two electrodes constituting that channel. Finally, the actual stimulation current amplitudes (in µA) are determined for each electrode based on a number of electrode-specific mapping parameters including the electric stimulation threshold (T) and most comfortable loudness (M) levels, the input dynamic range (IDR) and an additional gain (Nogueira et al., 2009). The output of this block is one pair of stimulation current levels for each channel, sampled at FT rate. In the actual CI system, these vectors (of pair-wise current amplitudes) are transmitted to the implanted RF receiver, where the implant logic generates the analog current waveforms based on stored pulse-shape templates and timing patterns.

## Fundamental Strategy Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Default value | Property of |
| nChan | Number of analysis channels | 15 | Strategy |
| fs | Sampling frequency | 17400 Hz | Strategy |
| pulseWidth | Duration of a single stimulation phase | 18 µs | Strategy |
| nFft | FFT length | 256 | FftStrategy |
| nHop | Hop size | 20 | FftStrategy |
| windowType | Analysis window for FFT | blackHann | FftStrategy |
| startBin | First FFT bin considered in filterbank | 6 | FftStrategy |
| nBinLims | Number of bins allocated in the respective channel | [2, 2, 1, 2, 2, 2, 3, 4, 4, 5, 6, 7, 8, 10, 56] | FftStrategy |

## Visualisation tools

csViewer

csViewer depicts the defined inputs and outputs of the blocks in the process chain and their relationship with each other. The signal flow is shown with arrows. Arrows in different colors indicate overlapping pathways in the process chain. When csViewer is called before execution of the strategy, the figure gets updated while running the script by marking completed stages in bold when parameter listen is set to 1. Figure 2 is generated with csViewer.

PlotF120ElectrodogramUnit

PlotF120ElectrodogramUnit (‘PLT’) can be added to the process chain like a processing block by connecting it to the output of an F120MappingUnit. It creates a figure showing an electrodogram, i.e. the excitation on each electrode across time. The height of the bar represents the current amplitude emitted by the respective electrode. Note that, for the sake of simplicity and generality, only the (positive) amplitude and onset time of each bi-phasic pulse is shown for each stimulation frame, but not its full time-course throughout the frame. Figure 3 (row 4) depicts the electrodogram of the word “yes” processed with the GMT implementation of the SpecRes strategy.

## Noise Vocoder CI Simulation

Noise vocoder CI simulations are often used in a first step of the evaluation of new sound coding algorithms with normal hearing listeners due to the easier recruitment of subjects and reduced variability. It was shown that results of CI simulations correlate with speech intelligibility results produced by high-performing CI performers. A noise vocoder CI simulation is also implemented in the GMT and allows the developer to listen to the output signal of the sound coding strategy before it is subjected to the processing that is done to obtain the electrical stimuli. Therefore, the blocks belonging to the noise vocoder CI simulation are implemented often after the envelope extraction stage.

VocoderUnit

In VocoderUnit, a FFT-based vocoder is implemented. The extracted envelopes are used to modulate the amplitudes of short-time Fourier transform (STFT) coefficients with the respective frequency content. The overlap between the pass-bands of synthesis filters can be configured to model different degrees of cross-talk between neighboring channels (e.g. due to variance in the distances between electrodes and neural tissue and the extent of current spread in the cochlear fluids and tissues). Phase synthesis can be configured to produce either (harmonic complex) tones or band-limited noise as carrier signal for each channel. Output of the vocoder is a matrix of complex STFT coefficients.

FftSynthesisUnit

In FftSynthesisUnit, an acoustic signal is generated from a STFT representation of the signal using weighted overlap-add (WOLA) synthesis. It is used at the end of the vocoder processing to obtain an acoustic output signal. However, it can also be used at various stages of the processing when a signal in the STFT domain has to be transformed to the time domain.

# Contact

The GMT was developed as a tool to facilitate experimentation with advanced sound coding concepts. When questions arise or bugs are detected, please contact AB and send a message to

[phillipp.hehrmann@advancedbionics.com](mailto:phillipp.hehrmann@advancedbionics.com)

or

raphael.koning@advancedbionics.com

We will get in contact as soon as possible to answer your request and take suggestions and remarks into account.

# References

P. Boyle, , A. Büchner, M. A. Stone, T. Lenarz, and B. C. J. Moore, “Comparison of Dual-Time-Constant and Fast-Acting Automatic Gain Control (AGC) Systems in Cochlear Implants,” International Journal of Audiology, 48(4): 211–221, 2009.

M. Brendel, A. Büchner, B. Krüger, C. Frohne-Büchner, and T. Lenarz, “Evaluation of the Harmony soundprocessor in combination with the speech coding strategy HiRes 120,“ Otology & Neurotology, 29(2):199-202, 2008.

W. Nogueira, L. Litvak, B. Edler, J. Ostermann, and A. Büchner, “Signal Processing Strategies for Cochlear Implants Using Current Steering,” EURASIP Journal on Advances in Signal Processing, 2009.

Copyright (c) 2014 - 2020 Advanced Bionics. All rights reserved.