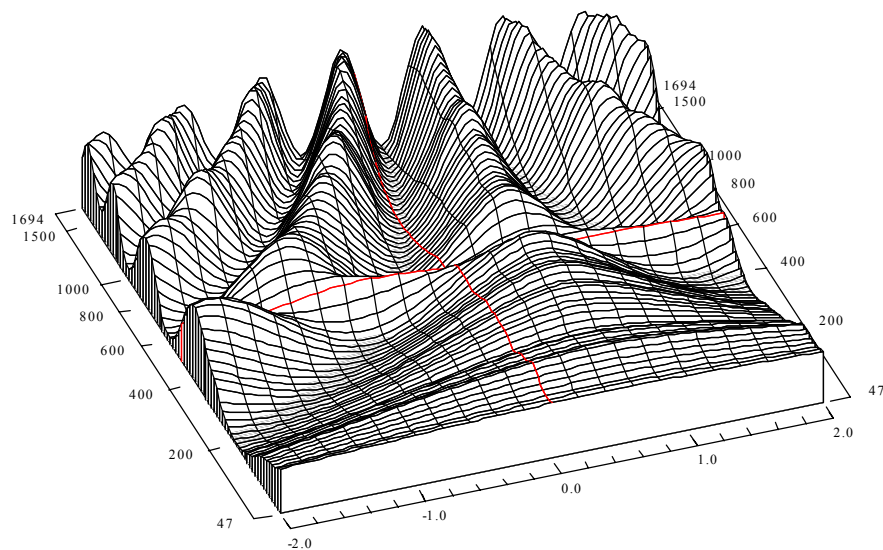


A Binaural Cross-correlogram Toolbox for MATLAB

Michael A. Akeroyd

Department of Neuroscience,
University of Connecticut Health Center,
263 Farmington Avenue,
Farmington, CT 06030, USA.

Laboratory of Experimental Psychology,
School of Biological Sciences,
University of Sussex,
Falmer, Brighton, BN1 9QG, UK.



Binaural correlogram of a 500-Hz Huggins pitch carried on a broadband noise of 500- μ s ITD.

February 2nd, 2001

Contents

1	Introduction	4
1.1	Preface	4
1.2	Hardware/software requirements	4
1.3	Installation	5
1.4	Acknowledgements	5
2	Overview of the toolbox	6
2.1	What is in the toolbox	6
2.2	Help	6
2.3	Functions in the toolbox	7
2.4	Typographic conventions	8
2.5	Data formats	8
2.5.1	‘wave’ format	8
2.5.2	‘correlogram’ format	10
2.6	The “infoflag” parameter	12
3	Generating a signal	13
3.1	Dichotic bandpass noise	13
3.2	Plotting a ‘wave’ signal	15
3.3	Playing a ‘wave’ signal	17
3.4	Saving a ‘wave’ signal	18
3.5	Adding two ‘wave’ signals	19
3.6	Concatenating two ‘wave’ signals	19
3.7	Other signal-generation functions	20
3.7.1	Bandpass noises	20
3.7.2	Interaurally-decorrelated bandpass noises	21
3.7.3	Dichotic pitches	23
3.7.4	Pure tones	24
3.7.5	Complex tones	25
3.7.6	Any pre-made signals	28
3.8	Obtaining an FFT of a waveform	29
4	Creating a binaural cross-correlogram	31
4.1	Calculating the correlogram	31
4.2	Plotting a correlogram	36
4.3	Models of neural transduction	38
4.4	Types of binaural processing	40
4.5	More examples of binaural correlograms	42
4.5.1	500-Hz pure tone with 500- μ s ITD	42
4.5.2	Interaurally-decorrelated broadband noise	43
4.5.3	600-Hz, 16% Huggins pitch	44

5 Post-processing a binaural cross-correlogram	45
5.1 Frequency weighting	45
5.2 Delay weighting	46
5.3 Illustrations of the weighting functions	47
5.4 Lateralization predictions: The centroid	49
5.5 Lateralization predictions: The best peak and the largest peak	50
 6 Displaying a correlogram in Windows	 53
6.1 Basic options	55
6.2 “Drawing-type” dialog box	60
6.3 “Info Window”	60
6.4 Temporary files	61
6.5 Copying/pasting a correlogram into Powerpoint	62
6.6 Keyboard shortcuts	63
 Appendix 1 Frequency Weighting	 64
1.1 Raatgever’s function	64
1.2 Stern et al’s function	64
1.3 The “MLD” function	65
1.4 Comparative Illustrations	65
 Appendix 2 Delay Weighting	 66
2.1 Colburn’s function	66
2.2 Shear’s function	66
2.3 Shackleton et al.’s function	67
2.4 Comparative Illustrations	68
 Appendix 3 The ERB function	 69
3.1 The ERB at a given center frequency	69
3.2 Expressing a frequency in ERB number	70
3.3 A useful filterbank	71
3.4 Illustrations	72
 References	 73
 Further reading	 74

Chapter 1

Introduction

1.1 Preface

This manual describes a MATLAB toolbox for computational modeling of binaural auditory processing. My goals were (1) to develop MATLAB software for calculating the binaural cross-correlogram of a sound and for then determining the lateralization of the sound, and (2) to develop Windows 98 software for displaying and post-processing a binaural cross-correlogram. The toolbox was written to support my own research but I am making it available in the hope that it may prove useful.

The manual does not explain either how to use MATLAB or how to do binaural modeling, and so a familiarity with the concepts of binaural hearing is assumed. I can only offer limited support for the toolbox, but I will try to mend any bugs that are present and help with its use. Although I use the toolbox continually, I tend to use the same functions and options and so might not have found any some bugs (in this sense the toolbox should be regarded as "Beta" code). I will try to incorporate any suggestions for improvements, changes, or additions to the toolbox or to this manual. I will also try to—but cannot promise to—answer any questions on binaural modeling that arise out of the use of this toolbox.

If you find the toolbox useful, I would appreciate it if you would send me an email to maa@biols.susx.ac.uk. I will then put your name onto a mailing list so I can let you know of new functions, bug corrections, etc.

The toolbox was written whilst I was a MRC Research Fellow at the University of Sussex and the University of Connecticut Health Center. Any for-profit use or redistribution is prohibited. No warranty is expressed or implied.

1.2 Hardware/Software requirements

I use this software on a Windows-98 PC running MATLAB 5.11 (specifically version 5.3.1.29215a). It also works on MATLAB 6 (version 6.0.0.88; release 12); the only known inconsistency is that `mccgrampl02dsqrt` does not plot the correlogram properly if either the *symbolsize* or *linewidth* parameters are set to 0. I do not know the degree to which this toolbox will work on earlier versions of MATLAB or on non-PC platforms. If, however, you find that it does (or does not) work, then I would appreciate it if you would let me know.

With one exception all of the MATLAB functions used in the toolbox are part of the standard release of MATLAB. The exception is the function `hilbert`, which computes the Hilbert Transform used in the envelope-compression algorithm. This function is part of the Signal Processing toolbox, which can be obtained from the Mathworks.

The display program `ccdisplay.exe` is a Windows 95/98 executable. It is written in Borland C++ Builder (version 3.0). The source code is available on request from myself.

1.3 Installation

All of the software in the toolbox should be copied into a single directory whose name is then added to the MATLAB path (see `pathdef.m` in (on my system) `matlab\toolbox\local\`). The name of the directory does not matter so something like "binauraltoolbox" will suffice. The location of the directory is also immaterial: the only requirement is that it can be accessed by MATLAB.

The Windows program `ccdisplay.exe` does not require separate installation as all of the libraries it needs are compiled with it. It should be placed in the same directory as the remainder of the toolbox.

1.4 Acknowledgements

I am extremely grateful to Klaus Hartung of the University of Connecticut Health Center for his help with MATLAB, for many improvements to the toolbox (especially with the Meddis hair cell), and for comments on earlier versions of this manual. The gammatone-filter code is taken from Malcolm Slaney's "Auditory Toolbox" (<http://rvl4.ecn.purdue.edu/~malcolm/interval/1998-010>). The code for the envelope-compression algorithm, the normalized-correlation calculation, the one-sided FFT/inverse FFT, and the basic code for generating a noise of given ITD and IPD are all taken from functions written by Les Bernstein of the University of Connecticut Health Center. The code used in the two-dimensional plot of the correlogram (`mccgram2dsqrt`) was supplied by Chris Darwin of the University of Sussex.

Contact address:

Dr Michael A Akeroyd,
Laboratory of Experimental Psychology,
University of Sussex,
Falmer,
Brighton, BN1 9QG,
United Kingdom.

Email: maa@biols.susx.ac.uk

Webpage: http://www.biols.susx.ac.uk/Home/Michael_Akeroyd/

This manual © Michael A. Akeroyd (February 3, 2001).

Chapter 2

Overview of the toolbox

2.1 What is in this manual

Chapters 3-6 are primarily a tutorial in the use of the toolbox. Chapter 3 describes how to make a bandpass noise. It also describes how to make other signals, and how to plot, play, and save a signal. Chapter 4 describes how to generate a binaural cross-correlogram. Chapter 5 describes how to apply frequency or delay weightings to a correlogram, as well as how to use a correlogram to predict the lateralization of a signal. Chapter 6 describes how to use the Windows program `ccdisplay` for displaying and processing a correlogram.

Appendices 1 and 2 outline the various frequency-weighting functions and delay-weighting functions that are available. Appendix 3 describes the ERB function used to calculate the bandwidth and frequency spacing of the filters in the gammatone filterbank.

The remainder of this chapter summarizes the functions provided in the toolbox, describes the data formats used by the functions, and briefly describes the *infoflag* parameter used in most of the functions.

2.2 Help

Online help for all the functions can be obtained by typing '`help functionname`'. For example, to see the help page for the function `mcreatetone`, type:

```
» help mcreatetone
```

Further documentation will be found in the comments to the code in each function.

2.3 Functions in the toolbox

The next page lists all the user functions. The other functions in the toolbox are used internally by these functions.

Signal generators (see Chapter 3)

mcreatetone	Dichotic pure tone.
mcreatecomplextone	Dichotic complex tone (components defined in an additional text file).
mcreatenoise1	Dichotic bandpass noise (defined by center frequency and bandwidth).
mcreatenoise2	Dichotic bandpass noise (defined by lowpass and highpass cutoff frequencies).
mcreatenoise1rho	Interaurally-decorrelated dichotic bandpass noise (defined by c.f. and bandwidth).
mcreatenoise2rho	Interaurally-decorrelated dichotic bandpass noise (defined by low/high frequencies).
mcreatehuggins1	Huggins pitch (carrier noise defined by center frequency and bandwidth).
mcreatehuggins2	Huggins pitch (carrier noise defined by lowpass and high-pass cutoff frequencies).
mwavcreate	Convert any pre-made signal to the 'wave' format.

Signal processing (see Chapter 3)

mwaveadd	Add two 'wave' signals.
mwavecat	Concatenate two 'wave' signals together.
mwaveplay	Play a 'wave' signal through the PC speakers.
mwavesave	Save a 'wave' signal as a .wav file.
mwaveplot	Plot a 'wave' signal.
mfft1side	Calculate and plot the FFT of a monaural waveform.

Binaural cross-correlograms (see Chapters 4 and 5)

mcorrelogram	Calculate and plot a correlogram of a 'wave' signal.
mccgramdelayweight	Apply delay weighting (the $p(\tau)$ function) to a correlogram.
mccgramfreqweight	Apply frequency weighting to a correlogram.
mccgrampeak	Find the location of a peak in the across-frequency average of a correlogram.
mccgramcentroid	Find the location of the centroid in the across-frequency average of a correlogram.
mccgramplot4panel	Plot a four-panel picture of a correlogram.
mccgramplot3dmesh	Plot a correlogram as a 3-dimensional mesh.
mccgramplot3dsurf	Plot a correlogram as a 3-dimensional surface.
mccgramplot2dsqrt	Plot a correlogram as a 2-dimensional plot (incorporates a square-root transformation of the correlogram values).
mccgramplotaverage	Plot the across-frequency average of a correlogram.

Displaying a correlogram in Windows (see Chapter 6)

mcallccdisplay	Display a previously-made correlogram using the Windows program ccdisplay .
ccdisplay.exe	A Windows program for displaying and transforming a correlogram.

ERB functions (see Appendix 3)

merb	Calculate the ERB at a given center frequency.
mhztoerb	Convert a frequency from units of Hz to units of ERB number.
merbtohz	Convert a frequency from units of ERB number to units of Hz.

2.4 Typographic conventions

In the text of this manual the names of functions and the names of variables are printed in bold Courier:

e.g., **mcreatetone.m**

Example command-lines are printed in bold Courier and are preceded by **»**. Apart from the **»**, which represents the MATLAB prompt and should not itself be typed, the rest of the line should be typed exactly. For example:

```
» mwaveplay(n, -1, 'stereo', 1);
```

Information reported by the functions to the MATLAB terminal window is printed in normal Courier. For example:

```
input waveform = n
duration = 6000 samples = 300.0 msecs
'stereo': leftchannel in leftear and rightchannel in rightear
auto-scaling amplitude to +1...-1
playing using 'sound' ...
```

2.5 Data formats

The toolbox uses two special structure arrays to hold data:

wave	Signal waveforms and associated statistics.
correlogram	Binaural cross-correlograms and associated statistics.

2.5.1 'wave' format

Chapter 3 describes how to make a noise stimulus and store it in a workspace variable, called **n**, using the 'wave' format. Typing **n** alone at the MATLAB terminal will list the fields of the 'wave' format and the values they contain:

```
>> n
n =
    generator: 'mcreatenoise1'
    leftwaveform: [5000x1 double]
    rightwaveform: [5000x1 double]
    samplefreq: 20000
    duration_samples: 5000
    duration_ms: 250
    leftmax: 7.2591e+003
    leftmin: -6.9245e+003
    leftrms: 1.8905e+003
    leftpower_db: 6.5532e+001
    leftenergy_db: 5.9511e+001
    rightmax: 7.2591e+003
    rightmin: -6.9245e+003
    rightrms: 1.8887e+003
```



```

rightpower_db: 6.5523e+001
rightenergy_db: 5.9503e+001
overallmax: 7.2591e+003
normalizedrho: 2.2596e-002

```

The fields are:

<i>generator</i>	Name of the function used to make the signals.
<i>leftwaveform</i>	Vector of sample values for the waveform of the left channel.
<i>rightwaveform</i>	Vector of sample values for the waveform of the right channel.
<i>samplefreq</i>	Sampling frequency (Hz).
<i>duration_samples</i>	Duration of the signal (samples).
<i>duration_ms</i>	Duration of the signal (milliseconds).
<i>leftmax</i>	Maximum sample value of the left channel.
<i>leftmin</i>	Minimum sample value of the left channel.
<i>leftrms</i>	Root-mean-square sample value of the left channel.
<i>leftpower_db</i>	Power of the left channel (dB).
<i>leftenergy_db</i>	Energy of the left channel (dB).
<i>rightmax</i>	Maximum sample value of the right channel.
<i>rightmin</i>	Minimum sample value of the right channel.
<i>rightrms</i>	Root-mean-square sample value of the right channel.
<i>rightpower_db</i>	Power of the right channel (dB).
<i>rightenergy_db</i>	Energy of the right channel (dB).
<i>overallmax</i>	Overall largest sample value in both channels.
<i>normalizedrho</i>	Normalized correlation of the signal.

Most of the fields are self-explanatory. One exception is the *normalizedrho* field. This field contains the value of the ‘normalized correlation’ of the signal, which is described in Bernstein and Trahiotis (1996a, equation 1). They, in that article and two subsequent ones (Bernstein and Trahiotis, 1996b; Bernstein et al., 1999), found the normalized correlation to be useful in predicting No π masking-level differences as a function of frequency and type of masking noise. The equation for the normalized correlation ρ is:

$$\rho = \frac{\sum_{n=1}^{n=N} x_n y_n}{\sqrt{\sum_{n=1}^{n=N} x_n^2} \sqrt{\sum_{n=1}^{n=N} y_n^2}}$$

where n is sample number, N is the duration of the signal in samples, and x_n and y_n are the left and right waveforms of the signal.

The functions `mcreatetone`, etc., all store signals using the ‘wave’ format. Also, the function for calculating the binaural cross-correlogram (`mcorrelogram`) assumes the signal is in the ‘wave’ format. A separate function (`mwavecreate`) will store *any* previously-made two-channel signal in the ‘wave’ format.

Note that the left and right waveforms are stored in one-dimensional vectors. They can therefore be manipulated in the same way as any other MATLAB vector. For example, to invert every sample in the left waveform of a 'wave' signal, type:

```
>> newwaveform = wavel.leftwaveform * -1;
```

For a second example, to play the left waveform using the MATLAB function `soundsc` and at the correct sampling frequency, type:

```
>> soundsc(wavel.leftwaveform, wavel.samplefreq)
```

An easy way of converting the transformed waveform to the 'wave' format is to use the `mwavcreate` function. For example, to invert the left waveform, type:

```
>> newwaveform1 = wavel.leftwaveform * -1;
>> wave2 = mwavcreate(newwaveform, wavel.rightwaveform, wavel.samplefreq, 1);
```

Note that many MATLAB functions can be condensed into one line. For example, the preceding example is the same as

```
>> wave2 = mwavcreate(wavel.leftwaveform * -1, wavel.rightwaveform,...
                    wavel.samplefreq, 1);
```

2.5.1 'correlogram' format

Chapter 4 describes how to generate the binaural cross-correlogram of a 'wave' signal and store it in a workspace variable, called `cc1`, using the 'correlogram' format. Typing `cc1` alone at the MATLAB terminal will list the fields of the 'correlogram' format and the values they contain:

```
>> cc1
cc1 =
    title: 'first-level correlogram'
    type: 'binauralcorrelogram'
    modelname: 'mcorrelogram'
    transduction: 'hw'
    samplefreq: 20000
    mincf: 200
    maxcf: 1000
    density: 2
    nfilters: 21
    q: 9.2789e+000
    bwmin: 2.4673e+001
    mindelay: -3500
    maxdelay: 3500
    ndelays: 141
    freqaxisHz: [21x1 double]
    freqaxiserb: [21x1 double]
    powerleft: [21x1 double]
```

```

    powerright: [21x1 double]
    delayaxis: [1x141 double]
    freqweight: 'null'
    delayweight: 'null'
    data: [21x141 double]

```

The fields are:

<i>title</i>	Title/information on correlogram.
<i>type</i>	Type of correlogram (usually ‘binauralcorrelogram’).
<i>modelname</i>	Name of function used to create the correlogram.
<i>transduction</i>	Name of model for neural transduction.
<i>samplefreq</i>	Sampling frequency of the signal (Hz).
<i>mincf</i>	Lowest filter frequency in the filterbank (Hz).
<i>maxcf</i>	Highest filter frequency in the filterbank (Hz).
<i>density</i>	Spacing of filters in the filterbank (filters per ERB).
<i>nfilters</i>	Number of filters in the filterbank.
<i>q</i>	'q' factor used in calculating the bandwidth of each filter.
<i>bwmin</i>	Minimum-bandwidth factor used in calculating the bandwidth of each filter.
<i>mindelay</i>	Smallest (most-negative) internal delay τ used in the correlogram.
<i>maxdelay</i>	Largest (most-positive) internal delay τ used in the correlogram.
<i>freqaxishz</i>	Vector of the center frequencies of each filter in the filterbank (Hz).
<i>freqaxiserb</i>	Vector of the center frequencies of each filter in the filterbank (ERB number).
<i>powerleft</i>	Vector of the power in each filter for the left channel.
<i>powerright</i>	Vector of the power in each filter for the right channel.
<i>delayaxis</i>	Vector of the internal delays τ in the correlogram (μ s).
<i>freqweight</i>	Whether frequency weighting has been applied or not.
<i>delayweight</i>	Whether delay weighting has been applied or not.
<i>data</i>	Two-dimensional (frequency x internal delay) matrix of the correlogram values.

The parameters *q* and *bwmin* control the bandwidth of the gammatone filters. They, as well as how to convert filter center frequencies from Hz to ERB number and back again, are described in Appendix 3.

Note that the correlogram itself is stored in the two-dimensional matrix *data*, which can be transformed in the same way as any other MATLAB matrix. For example, to square-root every value in the correlogram, type:

```
>> newdata = sqrt(cc1.data);
```

In order to store the transformed data in the 'correlogram' format, first copy the original correlogram, so preserving all the information in the other fields, and then to copy the transformed data into the *data* field directly. For example:

```

>> cc2 = cc1;
>> cc2.data = sqrt(cc1.data);

```

2.6 The *infoflag* parameter

The last parameter of most of the functions is *infoflag*, whose value determines the amount of information reported to the MATLAB terminal window. It can take values of 0, 1, or 2:

- 0 Do not report any information or plot any pictures.
- 1 Report some information as the function runs. I use this value most of the time as I like to watch the progress of the functions.
- 2 In addition to reporting information, also plot figures.

Note that a value of 2 is only meaningful for those functions that can plot pictures, of which the primary examples are `mfft1side` and `mcorrelogram`. Values of 0 and 1 apply to all the functions.

Chapter 3

Generating a signal

3.1 Dichotic bandpass noise

This part of the tutorial shows how to create a dichotic bandpass noise.

The function `mcreatenoise1` will synthesize a dichotic bandpass noise, by first creating two bands of noise in the spectral domain and then applying an inverse-FFT to create two waveforms. One band of noise is for the left channel, the other for the right channel. Both bands have the same center frequency, bandwidth, and duration, but can differ in phase or level so giving an ITD/IPD or an IID. The full syntax is

```
outputwave = mcreatenoise1(centerfreq, bandwidth,
                           spectrumlevelleft, spectrumlevelright, itd, ipd,
                           duration, gateduration, samplefreq, infoflag);
```

where the parameters are

<code>centerfreq</code>	Center frequency of the passband of the noise (Hz).
<code>bandwidth</code>	Bandwidth of the passband of the noise (Hz).
<code>spectrumlevelleft</code>	Spectrum level of the left channel of the passband of the noise (dB).
<code>spectrumlevelright</code>	Spectrum level of the right channel of the passband of the noise (dB).
<code>itd</code>	Interaural time delay (ITD) of the passband of the noise (microseconds).
<code>ipd</code>	Interaural phase delay (IPD) of the passband of the noise (degrees).
<code>duration</code>	Overall duration (milliseconds).
<code>gateduration</code>	Duration of raised-cosine onset/offset gates (milliseconds).
<code>samplefreq</code>	Sampling frequency (Hz).
<code>infoflag</code>	1 (report useful information) or 0 (do not report useful information).

For example, typing in this command will create a bandpass noise of 500-Hz center frequency, 400-Hz bandwidth, 40-dB spectrum level for both channels, 500- μ sec ITD, 0° IPD, 300-ms duration, 10-ms raised-cosine gates, and using a sampling frequency of 20000 Hz. The signal is stored in the workspace variable `n` :

```
>> n = mcreatenoise1(500,400,40,40,500,0,300,10,20000,0);
```

The last parameter is `infoflag`. This can be either 0 or 1. If it is equal to 0 then the program runs but does not report anything to the terminal window (as in the above example). If instead it is equal to 1 then the program reports the following information to the MATLAB terminal window as it runs

(although note that the line numbers in parentheses are *not* displayed; I added those for this tutorial). This is shown with the next example command:

```
>> n = mcreatenoise1(500,400,40,40,500,0,300,10,20000,1);
```

```
(1) This is mcreatenoise1.m
(2) creating 6000-point FFT buffer with 20000 sampling rate ...
(3) FFT resolution = 3.33 Hz
(4) center frequency: 500.0 Hz   bandwidth: 400 Hz
(5) lowest frequency : 300.0 Hz (rounded to 300.0 Hz)
(6) highest frequency: 700.0 Hz (rounded to 700.0 Hz)
(7) number of FFT components included = 121
(8) creating random real/imag complex pairs ...
(9) inverse ffting for waveform ...
(10) normalizing power ...
(11) getting phase spectrum ...
(12) time-delaying phase spectrum of right channel by 500 usecs ...
(13) phase-shifting phase spectrum of right channel by 0 degs ...
(14) inverse-FFTing left and right channels to get waveforms ...
(15) applying 10.0-ms raised cosine gates ...
(16) setting spectrum level of left channel to 40.0 dB (overall level=66.0 dB)...
(17) setting spectrum level of right channel to 40.0 dB (overall level=66.0 dB)...
(18) transposing left waveform ...
(19) transposing right waveform ...
(20) creating 'wave' structure ...
(21) waveform statistics :
(22)   samplingrate           = 20000 Hz
(23)   power (left, right)    = 66.3 dB   66.3 dB
(24)   energy (left, right)   = 61.1 dB   61.1 dB
(25)   maximum (left, right)  = 6637.9    6637.9
(26)   minimum (left, right)  = -7366.0   -7366.0
(27)   rms amplitude (left, right) = 2065.0    2063.7
(28)   duration              = 6000 samples = 300.00 msecs
(29)   normalized correlation = 0.0254
(30) storing waveform to workspace as wave structure ..
```

Line 1 reports the name of the program. Lines 2 and 3 report the size of the buffer used for the FFT. As MATLAB can perform non-power-of-2 FFTs this buffer is equal to the duration of the noise in samples. Lines 4-6 report the frequency parameters of the passband of the noise in Hz and when rounded to the closest FFT frequency (this rounding is necessary as the FFT does not necessarily use integer-spaced frequencies; in this example the frequencies are spaced at 3.33-Hz steps (see line 3)). Line 7 reports how many of the FFT components are included in the passband, as both the requested values and rounded to the closest FFT frequencies. Lines 8-15 report stages in the generation of the noise. Lines 16-17 report the spectrum and overall levels of the right channels. Lines 18-20 report further stages in the generation of the noise. Lines 22-29 report some statistics on the noise; note that the power (line 23) is equal to the requested spectrum level (lines 16/17) in dB plus $10\log_{10}(\text{bandwidth})$, where the value of the bandwidth is in Hz. These values will not be exactly the same as the requested value as each individual noise is a different random process.

`mcreatenoise1` creates a 'wave' signal, which contains the left and right waveforms of the signal as well as a variety of statistics on those waveforms. The components of the structure array can be shown by typing in the name of the variable (here `n`) at the MATLAB command line:

```

>> n
n =

      generator: 'mcreatenoise1'
    leftwaveform: [6000x1 double]
    rightwaveform: [6000x1 double]
      samplefreq: 20000
duration_samples: 6000
    duration_ms: 300
      leftmax: 6.6379e+003
      leftmin: -7.3660e+003
      leftrms: 2.0650e+003
    leftpower_db: 6.6298e+001
    leftenergy_db: 6.1070e+001
      rightmax: 6.6379e+003
      rightmin: -7.3660e+003
      rightrms: 2.0637e+003
    rightpower_db: 6.6293e+001
    rightenergy_db: 6.1064e+001
      overallmax: 7.3660e+003
    normalizedrho: 2.5356e-002

```

Each field of the 'wave' format is described Section 2.5.1.

3.2 Plotting a 'wave' signal

The left and right waveforms of the noise can be plotted using `mwaveplot`. The syntax of `mwaveplot` is

```
mwaveplot(wave, channelflag, starttime, endtime)
```

where the parameters are

<code>wave</code>	Signal to be plotted.
<code>channelflag</code>	Whether to plot the left channel, right channel or both channels.
<code>starttime</code>	Sample time at which to start plot.
<code>endtime</code>	Sample time at which to end plot.

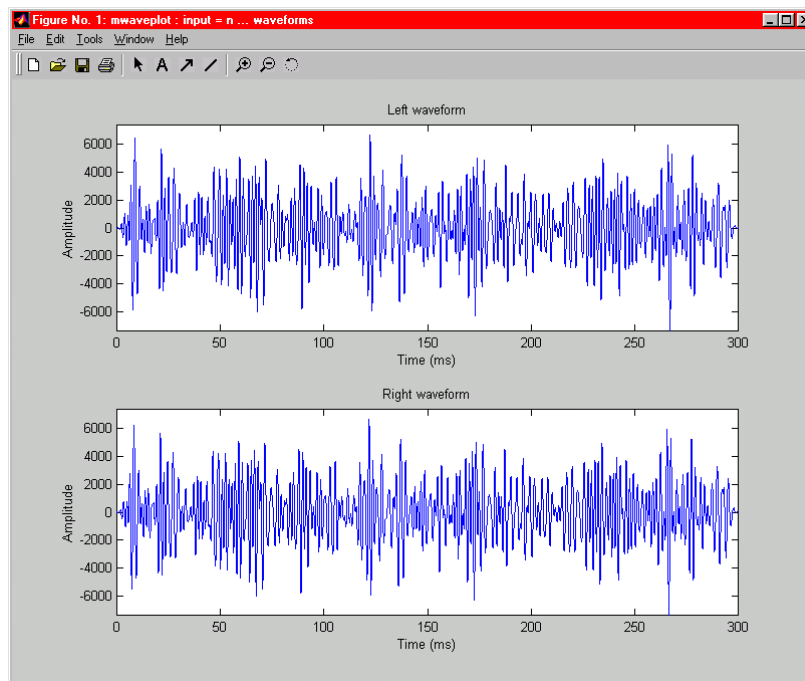
For example, to plot the full waveforms of both channels of `n`, type:

```
>> mwaveplot(n, 'stereo', -1, -1);
```

The resulting figure is shown at the top of the next page. The two parameters `-1` and `-1` mean, respectively, start the plot at the beginning of the signal and end the plot at the finish of the sound.

The `channelflag` parameter can take three values (note that the quote marks must be included):

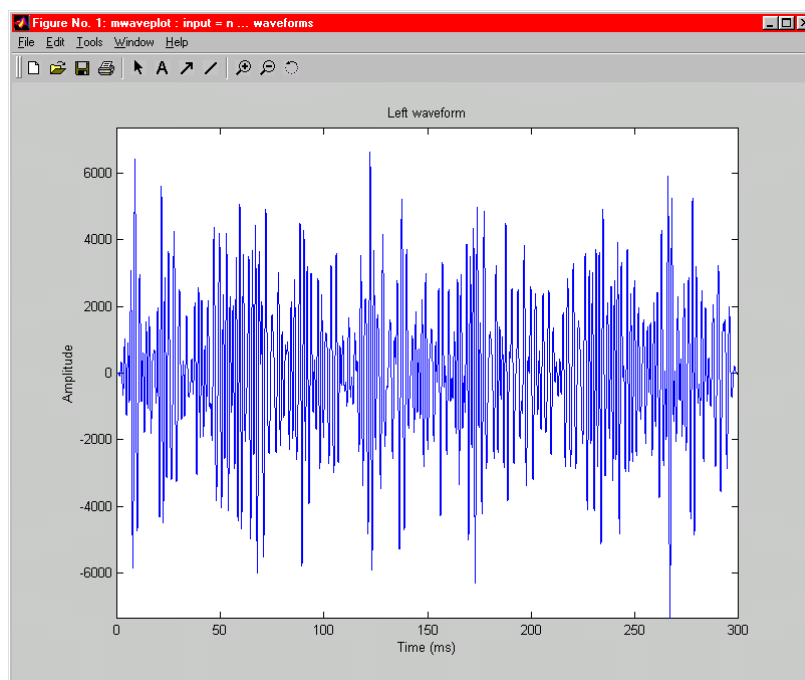
<code>'stereo'</code>	Plot both channels.
<code>'left'</code>	Plot the left channel only.
<code>'right'</code>	Plot the right channel only.



For example, to plot the full waveform of the left channel of **n**, type:

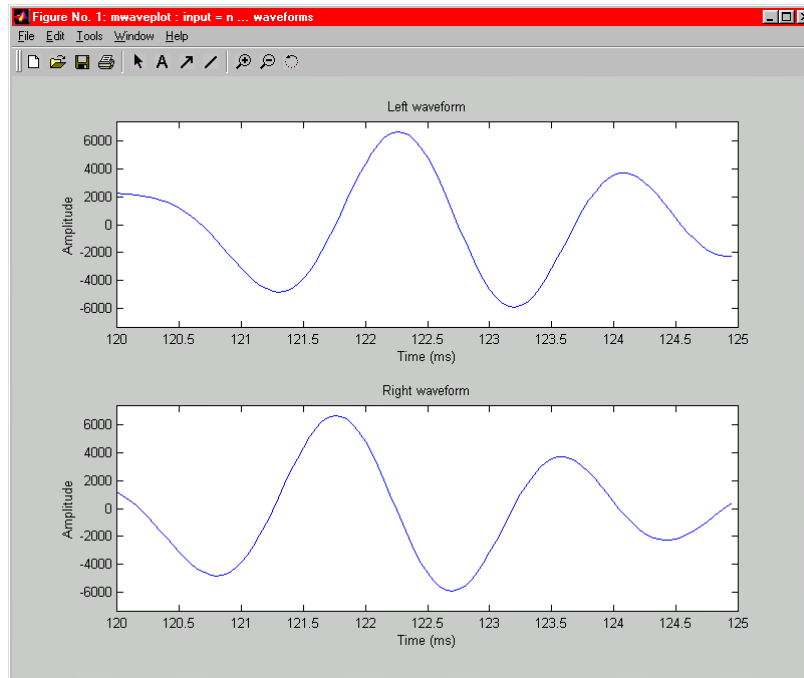
```
» mwaveplot(n, 'left', -1, -1);
```

The resulting figure is shown below.



The duration of the plotted waveforms is controlled by the third and fourth parameters. Their values define the start time and end time of the plots, with the exception that values of -1 (as used in the above examples) mean start-plot-at-beginning-of-signal (third parameter) and end-plot-at-finish-of-signal (fourth parameter). For example, to plot the signal between $t = 120$ ms and $t = 125$ ms, type;

```
» mwaveplot(n, 'stereo', 120, 125);
```



Note that this plot shows that the right waveform leads the left waveform by 0.5 ms. This is because the noise **n** was made with an ITD of 500 μ s.

3.3 Playing a 'wave' signal

The function **mwaveplay** will play a 'wave' signal through the PC speakers. The normal situation is to play the both channels of the signal at maximum amplitude. For example, to play the noise **n**, type:

```
» mwaveplay(n, -1, 'stereo', 1);
```

```
input waveform = n
duration = 6000 samples = 300.0 msecs
'stereo': leftchannel in leftear and rightchannel in rightear
auto-scaling amplitude to +1...-1
playing using 'sound' ...
```

The first parameter (here `n`) is the 'wave' signal to be played.

The second parameter (here `-1`) is a scaling factor that sets the level of the signal. When MATLAB plays a sound it assumes that the maximum amplitude range of the signal is -1 to $+1$; any samples with a value outside this range are clipped. If the second option of `mwaveplay` is set to `-1` then `mwaveplay` will automatically scale the signal so that the *overallmax* field of the 'wave' signal is equal to $+1$. This is done by dividing all the sample values by *overallmax*. This value therefore sets the level to be the maximum without clipping. If instead the value of the second parameter is equal to anything other than `-1` then `mwaveplay` will divide the sample values by that value and then play the signal. To ensure no clipping, this number should be large enough so that the resulting values are all in the range -1 to $+1$.

The third parameter (here `'stereo'`) is a switch controlling which channels are played. The options are (again the quote marks must be included):

<code>'stereo'</code>	Play both channels (as in the above example).
<code>'swap'</code>	Play both channels but with the left and right channels swapped.
<code>'random'</code>	Use one of <code>'stereo'</code> or <code>'swap'</code> , chosen at random each time the function is called.
<code>'left'</code>	Play left channel only.
<code>'right'</code>	Play right channel only.

For example, to play the left channel only, type:

```
» mwaveplay(n, -1, 'left', 1);
```

The fourth parameter (here `1`) is *infoflag*. If it is equal to `1` then the function reports the running information; if it is equal to `0` then nothing is reported but the function still plays the signal.

3.4 Saving a 'wave' signal

The function `mwavesave` will save the signal as a `.wav` file. The syntax is the same as that for `mwaveplay` except that an additional parameter specifies the filename. For example:

```
» mwavesave('sound1.wav', n, -1, 'stereo', 1);
```

```
input waveform =
duration = 6000 samples = 300.0 msecs
'stereo': leftchannel in leftear and rightchannel in rightear
auto-scaling amplitude to +1...-1
saving to file sound1.wav using 'wavwrite' (16-bit resolution)...
```

This example will save the 'wave' signal `n` in the file `'sound1.wav'`, with automatic setting of the amplitude range to -1 to $+1$ and using the `'stereo'` option (i.e., both channels). The amplitude scaling factor (here `-1`) and the channel switch (here `'stereo'`) are the same as those described above for `mwaveplay`.

3.5 Adding two 'wave' signals

The function `mwaveadd` will add together two 'wave' signals. For example, to add a noise `n1` to a second noise `n2` and store the result in `n3`, type:

```
» n3 = mwaveadd(n1, n2, 1);
```

```
adding waves ...
creating 'wave' structure ...
waveform statistics :
  samplingrate           = 20000 Hz
  power (left, right)    = 72.3 dB  72.3 dB
  energy (left, right)   = 67.1 dB  67.1 dB
  maximum (left, right)  = 13275.9    13275.9
  minimum (left, right)  = -14732.0   -14732.0
  rms amplitude (left, right) = 4129.9    4127.3
  duration               = 6000 samples = 300.00 msecs
  normalized correlation = 0.0254
```

The two signals should have the same duration and sampling frequency.

3.6 Concatenating two 'wave' signals

The function `mwavecat` will concatenate two 'wave' signals together. The syntax is:

```
outputwave = mwavecat(wave1, wave2, silence_ms, infoflag);
```

where the parameters are:

```
wave1      First 'wave' signal.
wave2      Second 'wave' signal.
silence_ms Duration of silent burst to put in between the two signals (milliseconds).
infoflag    1 or 0.
```

The two signals should have the same sampling rate.

For example, to create one diotic noise of 250-ms duration, a second diotic noise of 50-ms duration, and then to concatenate them together (with 100-ms of silence between them) and store the result in `n3`, type:

```
» n1 = mcreatenoise1(500,400,40,40,0,0,250,10,20000,0);
» n2 = mcreatenoise1(500,400,40,40,0,0,50,10,20000,0);
» n3 = mwavecat(n1, n2, 100, 1);
```

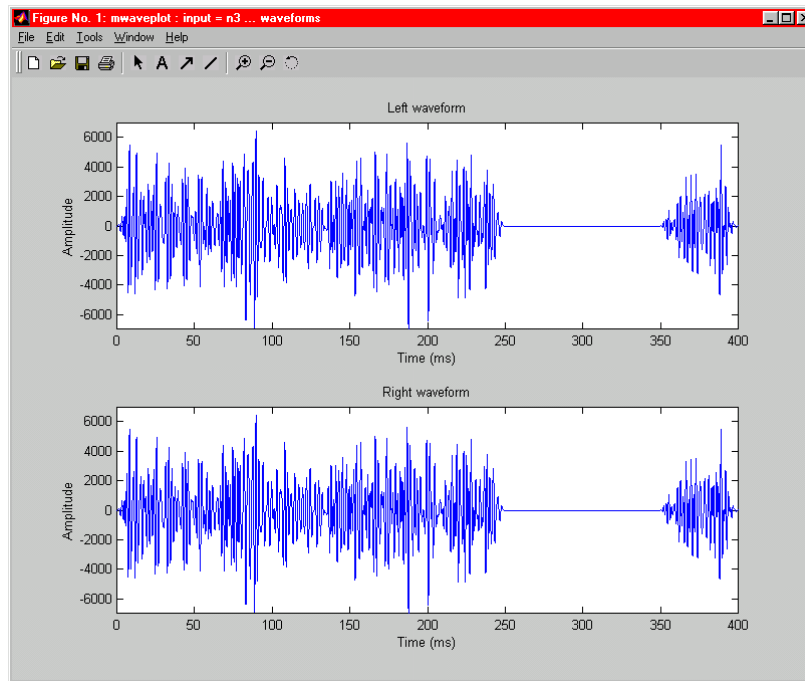
```
concatenating waves ...
creating 'wave' structure ...
waveform statistics :
  samplingrate           = 20000 Hz
```

```

power (left, right)          = 64.5 dB   64.5 dB
energy (left, right)         = 60.5 dB   60.5 dB
maximum (left, right)        = 6390.7    6390.7
minimum (left, right)        = -6954.1   -6954.1
rms amplitude (left, right)   = 1680.1    1680.1
duration                     = 8000 samples = 400.00 msecs
normalized correlation        = 1.0000

```

The next picture shows the plot of `n3` using `mwaveplot`; note that `n3` consists of the 250-ms noise, a 100-ms silent gap, and then the 50-ms noise:



3.7 Other signal-generation functions

3.7.1 Bandpass noises

The function `mcreatenoise2` is similar to `mcreatenoise1` but the first two parameters specify the lower and higher cutoff frequencies instead of the center frequency and bandwidth. The other parameters are the same. The full syntax is

```

outputwave = mcreatenoise2(lowfrequency, highfrequency,
                           spectrumlevelleft, spectrumlevelright, itd, ipd,
                           duration, gateduration, samplefreq, infoflag);

```

where the parameters are

`lowfrequency` Lower cutoff frequency of the passband of the noise (Hz).

<code>highfrequency</code>	Higher cutoff frequency of the passband of the noise (Hz).
<code>spectrumlevelleft</code>	Spectrum level of the left channel of the passband of the noise (dB).
<code>spectrumlevelright</code>	Spectrum level of the left channel of the passband of the noise (dB).
<code>itd</code>	Interaural time delay (ITD) of the passband of the noise (microseconds).
<code>ipd</code>	Interaural phase delay (IPD) of the passband of the noise (degrees).
<code>duration</code>	Overall duration (milliseconds).
<code>gateduration</code>	Duration of raised-cosine onset/offset gates (milliseconds).
<code>samplefreq</code>	Sampling frequency (Hz)
<code>infoflag</code>	1 or 0.

For example, the noise described in Section 3.1 had a center frequency of 500 Hz and a bandwidth of 400 Hz. The passband therefore extends from 300 Hz to 700 Hz. So, to make this noise using `mcreatenoise2` instead of `mcreatenoise1`, type:

```
>> n = mcreatenoise2(300,700,40,40,500,0,300,10,20000,0);
```

For a second example, to create a diotic noise with a passband from 0 Hz to 1000 Hz, type:

```
>> n = mcreatenoise2(0,1000,40,40,0,0,300,10,20000,0);
```

3.7.2 Interaurally-decorrelated bandpass noises

The pair of functions `mcreatenoise1rho` and `mcreatenoise2rho` synthesize an interaurally-decorrelated noise. The interaural correlation ρ (rho) of the noise is specified instead of the ITD or IPD. The numbers 1 and 2 in the function names are the same as for `mcreatenoise1` and `mcreatenoise2`: in `mcreatenoise1rho` the center frequency and bandwidth are specified, and in `mcreatenoise2rho` the lower and higher cutoff frequencies are specified.

The full syntax of `mcreatenoise1rho` is:

```
outputwave = mcreatenoise1rho(centerfreq, bandwidth,
                               spectrumlevelleft, spectrumlevelright, rho,
                               duration, gatelength, samplefreq, infoflag)
```

where the parameters are

<code>centerfreq</code>	Center frequency of the passband of the noise (Hz).
<code>bandwidth</code>	Bandwidth of the passband of the noise (Hz).
<code>spectrumlevelleft</code>	Spectrum level of the left channel of the passband of the noise (dB).
<code>spectrumlevelright</code>	Spectrum level of the left channel of the passband of the noise (dB).
<code>rho</code>	Interaural correlation of the noise.
<code>duration</code>	Overall duration (milliseconds).
<code>gateduration</code>	Duration of raised-cosine onset/offset gates (milliseconds).
<code>samplefreq</code>	Sampling frequency (Hz).
<code>infoflag</code>	1 or 0.

The syntax of `mcreatenoise2rho` is the same but the first two parameters specify the lower and higher cutoff frequencies.

For example, to create a noise with an interaural correlation of 0.0 (i.e., perfectly uncorrelated, and so commonly referred to as "Nu"), but whose other parameters are the same as those used in Section 3.1, type:

```
>> n = mcreatenoise1rho(500,400,40,40,0,300,10,20000,1);

(1) This is mcreatenoise1rho.m
(2) creating common noise: relative power          = 0.00 ...
(3) creating first independent noise: relative power = 1.00 ...
(4) creating second independent noise: relative power = 1.00 ...
(5) creating 'wave' structure ...
(6) waveform statistics :
(7)   samplingrate          = 20000 Hz
(8)   power (left, right)   = 65.6 dB  65.4 dB
(9)   energy (left, right)  = 60.4 dB  60.1 dB
(10)  maximum (left, right) = 6935.1    6493.2
(11)  minimum (left, right) = -6436.5    -6791.9
(12)  rms amplitude (left, right) = 1915.4    1854.7
(13)  duration              = 6000 samples = 300.00 msecs
(14)  normalized correlation = -0.0165
(15) storing waveform to workspace as wave structure ..
```

Note that the *normalized correlation* field (line 14) of the signal is approximately 0; it is not exactly 0.0 because of random fluctuations inherent to all noises.

For a second example, to create the same noise but with an interaural correlation of -1.0 (i.e., a IPD of π radians, and so commonly referred to as "N π "), type:

```
>> n = mcreatenoise1rho(500,400,40,40,-1,300,10,20000,1);

(1) This is mcreatenoise1rho.m
(2) creating common noise: relative power          = 1.00 ...
(3) inverting one channel of common noise to get negative correlation ...
(4) creating first independent noise: relative power = 0.00 ...
(5) creating second independent noise: relative power = 0.00 ...
(6) creating 'wave' structure ...
(7) waveform statistics :
(8)   samplingrate          = 20000 Hz
(9)   power (left, right)   = 65.9 dB  65.9 dB
(10)  energy (left, right)  = 60.7 dB  60.7 dB
(11)  maximum (left, right) = 5683.6    6491.9
(12)  minimum (left, right) = -6491.9    -5683.6
(13)  rms amplitude (left, right) = 1969.2    1969.2
(14)  duration              = 6000 samples = 300.00 msecs
(15)  normalized correlation = -1.0000
(16) storing waveform to workspace as wave structure ..
```

In this example the normalized correlation is exactly -1.0.

3.7.3 Dichotic pitches

The pair of functions `mcreatehuggins1` and `mcreatehuggins2` synthesize a “Huggins” dichotic-pitch imposed on a bandpass noise. The functions apply a linear transition in interaural phase, from 0 radians to 2π radians. If made sufficiently narrow, then this transition gives rise to the sensation of pitch when the stimulus is presented binaurally over headphones.

The parameters of `mcreatehuggins1` and `mcreatehuggins2` are the same as those in `mcreatenoise1` and `mcreatenoise2`, except that two additional parameters are used which define the center frequency and bandwidth of the transition in interaural phase. The numbers 1 and 2 in the function names are the same as for `mcreatenoise1` and `mcreatenoise2`: in `mcreatehuggins1` the center frequency and bandwidth of the bandpass noise are specified, and in `mcreatehuggins2` the lower and higher cutoff frequencies of the bandpass noise are specified.

The full syntax of `mcreatehuggins1` is:

```
outputwave = mcreatehuggins1(transitioncf, transitionbw,
                             centerfreq, bandwidth,
                             spectrumlevelleft, spectrumlevelright, itd, ipd,
                             duration, gatelength, samplefreq, infoflag)
```

where the parameters are the same as `mcreatenoise1` apart from the first two:

<code>transitioncf</code>	Center frequency of interaural phase transition (Hz).
<code>transitionbw</code>	Bandwidth of interaural phase transition (% of center frequency).

The syntax of `mcreatehuggins2` is the same as `mcreatehuggins1` but parameters three and four specify the lower and higher cutoff frequencies.

For example, to create a Huggins pitch at 600 Hz and of 16% bandwidth, carried on a noise of 500-Hz center frequency, 1000-Hz bandwidth, 40-dB spectrum level, 0- μ s ITD, 0-degrees IPD, 250-ms duration, 10-ms raised-cosine gates and using a sampling frequency of 20000 Hz, type:

```
>> n = mcreatehuggins1(600, 16, 500, 1000, 40, 40, 0, 0, 250, 10, 20000, 1);
```

The printed output is mostly the same as `mcreatenoise1` but includes some additional lines describing the transition in interaural phase (lines 14-18):

```
(1) This is mcreatehuggins1.m
(2) creating 5000-point FFT buffer with 20000 sampling rate ...
(3) FFT resolution = 4.00 Hz
(4) center frequency: 500.0 Hz  bandwidth: 1000 Hz
(5) lowest frequency : 0.0 Hz (rounded to 0.0 Hz)
(6) highest frequency: 1000.0 Hz (rounded to 1000.0 Hz)
(7) number of FFT components included = 251
(8) creating random real/imag complex pairs ...
(9) inverse FFTing for waveform ...
(10) normalizing power ...
(11) getting phase spectrum ...
```

```

(12) time-delaying phase spectrum of right channel by 0 usecs ...
(13) phase-shifting phase spectrum of right channel by 0 degs ...
(14)   creating 0-2pi phase-shift transition in left channel ...
(15)   bottom freq = 552.0 Hz (=0 radians)
(16)   middle freq = 600.0 Hz (=pi radians)
(17)   top freq    = 648.0 Hz (=2pi radians)
(18)   range       = 96.0 Hz = 24 FFT points)
(19) applying 10.0-ms raised cosine gates ...
(20) setting spectrum level of left channel to 40.0 dB (overall level = 70.0 dB)
(21) setting spectrum level of right channel to 40.0 dB (overall level = 70.0 dB)
(22) transposing left waveform ...
(23) transposing right waveform ...
(24) creating 'wave' structure ...
(25) waveform statistics :
(26)   samplingrate           = 20000 Hz
(27)   power (left, right)    = 70.3 dB   70.3 dB
(28)   energy (left, right)   = 64.3 dB   64.2 dB
(29)   maximum (left, right)  = 11350.9    11003.6
(30)   minimum (left, right)  = -12935.6   -10198.1
(31)   rms amplitude (left, right) = 3264.2    3259.7
(32)   duration              = 5000 samples = 250.00 msec
(33)   normalized correlation = 0.8959
(34) storing waveform to workspace as wave structure ..

```

3.7.4 Pure tones

The function `mcreatetone` will synthesize a dichotic pure tone. The syntax is:

```
outputwave = mcreatetone(freq, powerleft, powerright, itd, ipd,
                        duration, gatelength, samplefreq, infoflag)
```

where the parameters are:

<code>freq</code>	Frequency of the tone (Hz).
<code>powerleft</code>	Power of the left channel of the tone (dB).
<code>powerright</code>	Power of the right channel of the tone (dB).
<code>itd</code>	Interaural time delay (ITD) of the tone (microseconds).
<code>ipd</code>	Interaural phase delay (IPD) of the tone (degrees).
<code>duration</code>	Overall duration (milliseconds).
<code>gateduration</code>	Duration of raised-cosine onset/offset gates (milliseconds).
<code>samplefreq</code>	Sampling frequency (Hz).
<code>infoflag</code>	1 or 0.

For example, to create a tone of 750-Hz frequency, 60-dB level for both left and right channels, 500- μ sec ITD, 0° IPD, 300-ms duration, 0-ms raised-cosine gates, and using a sampling frequency of 20000 Hz, type:

```
>> t = mcreatetone(750,60,60,500,0,300,0,20000,1);
```



```

(1) This is mcreatetone.m
(2) frequency =750.0 Hz
(3) itd = 500.0 usecs ipd = 0.0 degs => trueitd: 500.000 usecs
(4) left channel : level = 60.00 dB amplitude = 1414.2 samples
(5) right channel : level = 60.00 dB amplitude = 1414.2 samples
(6) left channel : starting phase (sin) = 0.000 cycles
(7) right channel : starting phase (sin) = 0.375 cycles
(8) creating sinwaves ...
(9) applying 10.0-ms raised cosine gates ...
(10) transposing left waveform ...
(11) transposing right waveform ...
(12) creating 'wave' structure ...
(13) waveform statistics :
(14) samplingrate = 20000 Hz
(15) power (left, right) = 60.0 dB 60.0 dB
(16) energy (left, right) = 54.8 dB 54.8 dB
(17) maximum (left, right) = 1414.2 1414.2
(18) minimum (left, right) = -1414.2 -1414.2
(19) rms amplitude (left, right) = 1000.0 1000.0
(20) duration = 6000 samples = 300.00 msecs
(21) normalized correlation = -0.7071
(22) storing waveform in workspace as 'wave' ..

```

Note that the root-mean-square amplitude of the tone is 1000 (line 19). The maximum sample value is therefore 1414 (line 17), as, for a pure tone, these values are related by a factor of $\sqrt{2}$. Furthermore, the requested power was 60 dB, which is equal to $20\log_{10}(1000)$.

Also, note that the root-mean-square amplitude is calculated over the full duration of the signal. Thus it will be smaller if raised-cosine gates are incorporated at the onset and offset of the signal. For example, if 25-ms gates are used, then the root-mean-square amplitude falls to 946:

```
>> t = mcreatetone(750,60,60,500,0,300,25,20000,0);
```

```

This is mcreatetone.m
...
maximum (left, right) = 1414.2 1414.2
minimum (left, right) = -1414.2 -1414.2
rms amplitude (left, right) = 946.4 946.4

```

3.7.5 Complex tones

The function `mcreatecomplextone` will synthesize a dichotic complex tone. The syntax is

```
outputwave = mcreatecomplextone(parameterfile, overallgain_db, gatelength_ms,
                                samplefreq, infoflag)
```

where the parameters are:

```
parameterfile    Name of text file defining the components of the complex tone.
overallgain_db   Overall gain applied to all components (dB).
```

gatelength_ms	Duration of raised-cosine onset/offset gates applied to <u>each</u> component (msecs).
samplefreq	Sampling frequency (Hz).
infoflag	1 or 0.

The text file must specify all the parameters of the components in a special format. The supplied example is called **complextonefile1.txt** and is shown next:

```
% Parameter file for specifying a complex tone
% Read by mcreatecomplextone.m
%
% ordering of values in each line is:
%   freq           Hz
%   level(left)    dB
%   level(right)   dB
%   phase          degrees (assumes 'sin' generator) (-999 is code for random)
%   starttime      msecs
%   end            msecs
%   itd            usecs
%   ipd            degrees
%
% All lines beginning with '%' are ignored
%
% This example makes a complex-tone similar to that used by Hill
% and Darwin (1996; JASA, 100, 2352-2364): a 1000-ms duration
% complex tone with 1500-us ITD but with the 500-Hz component
% starting after 400-ms and only lasting for 200 ms
% (cf Hill and Darwin, Exp 1)
%
% Example of MATLAB call:
% >> wavel = mcreatecomplextone('complextonefile1.txt', 0, 20, 20000, 1);
%
%
% version 1.0 (Jan 20th 2001)
% MAA Winter 2001
%-----

200  60 60   90   0 1000   1500 0
300  60 60   90   0 1000   1500 0
400  60 60   90   0 1000   1500 0

600  60 60   90   0 1000   1500 0
700  60 60   90   0 1000   1500 0
800  60 60   90   0 1000   1500 0

500  60 60   90  400 600   1500 0

% the end!
```

In the text file all lines beginning with % are ignored by the parser in `mcreatecomplextone`. All other lines are assumed to specify a separate frequency component. The format of each line is:

frequency	power_left	power_right	startingphase	start_time	end_time	ITD	IPD
(Hz)	(dB)	(dB)	(degrees)	(ms)	(ms)	(us)	(deg)

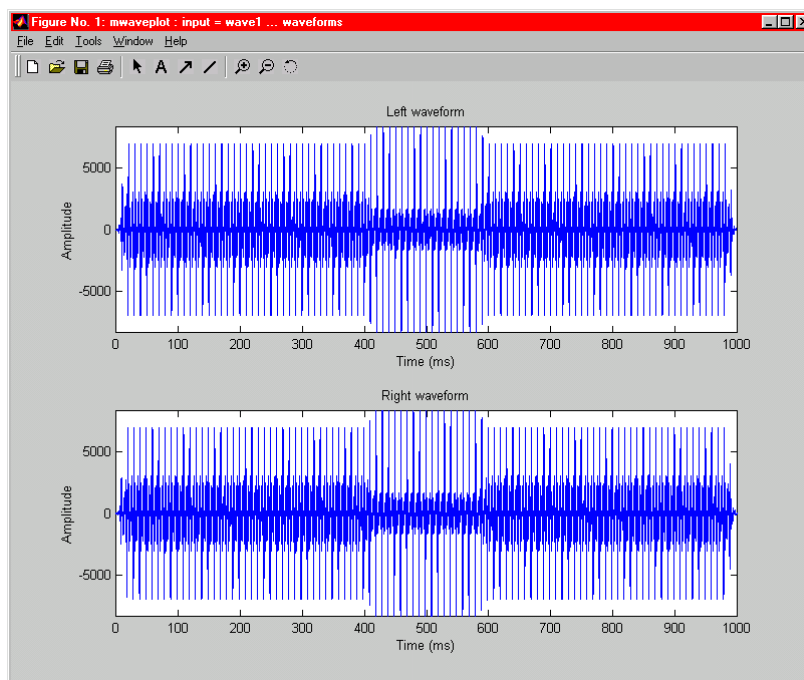
Most of the parameters are the same as those used in `mcreatetone`. The three that are not are:

<code>startingphase</code>	The starting phase of each component: 0° corresponds to 'sin' phase and 90° to 'cos' phase. A value of -999 means that a random starting phase is used.
<code>start_time</code>	When to start the component, relative to the start of the signal (ms).
<code>end_time</code>	When to end the component, relative to the start of the signal (ms).

Note that (1) the components can be specified in any order and (2) `mcreatecomplextone` can create asynchronous components. For example, in the above file `complextonefile1.txt`, the 500-Hz component starts after 400 ms and lasts 200 ms. To create this signal, type:

```
>> t = mcreatecomplextone('complextonefile1.txt', 0, 20, 20000, 1);
This is mcreatecomplextone.m
  freq (left|right starting phase) (left|right level, gain) itd|ipd start|end times
1: 200 Hz (0.000 1.885 rads) (60.00 60.00 +0.00 dB) 1500|0 us|degs start|end = 0 1000 ms
2: 300 Hz (0.000 2.827 rads) (60.00 60.00 +0.00 dB) 1500|0 us|degs start|end = 0 1000 ms
3: 400 Hz (0.000 3.770 rads) (60.00 60.00 +0.00 dB) 1500|0 us|degs start|end = 0 1000 ms
4: 500 Hz (0.000 4.712 rads) (60.00 60.00 +0.00 dB) 1500|0 us|degs start|end = 400 600 ms
5: 600 Hz (0.000 5.655 rads) (60.00 60.00 +0.00 dB) 1500|0 us|degs start|end = 0 1000 ms
6: 700 Hz (0.000 6.597 rads) (60.00 60.00 +0.00 dB) 1500|0 us|degs start|end = 0 1000 ms
7: 800 Hz (0.000 7.540 rads) (60.00 60.00 +0.00 dB) 1500|0 us|degs start|end = 0 1000 ms
transposing left waveform ...
transposing right waveform ...
creating 'wave' structure ...
waveform statistics :
  samplingrate          = 20000 Hz
  power (left, right)    = 67.8 dB 67.8 dB
  energy (left, right)   = 67.8 dB 67.8 dB
  maximum (left, right)  = 8285.6 8285.6
  minimum (left, right)  = -8285.6 -8285.6
  rms amplitude (left, right) = 2454.5 2454.5
  duration               = 20000 samples = 1000.00 msecs
  normalized correlation = 0.0000
storing waveform to workspace as wave structure ..
```

The asynchronous 500-Hz component generates the change in the waveform visible from 400 to 600 ms shown in the next picture.



3.7.6 Any pre-made signals

The function `mwavecreate` will convert any two MATLAB vectors to the 'wave' format. This function therefore allows any premade signal to be used. The syntax is:

```
outputwave = mwavecreate(leftwaveform, rightwaveform, samplefreq, infoflag);
```

where the parameters are:

<code>leftwaveform</code>	Vector containing the left waveform.
<code>rightwaveform</code>	Vector containing the right waveform.
<code>samplefreq</code>	Sampling frequency (Hz).
<code>infoflag</code>	1 or 0.

For example, if one vector is made which contains 1 cycle of a sinusoid of unit frequency:

```
>> leftwaveform = sin(0:0.01:2*pi);
```

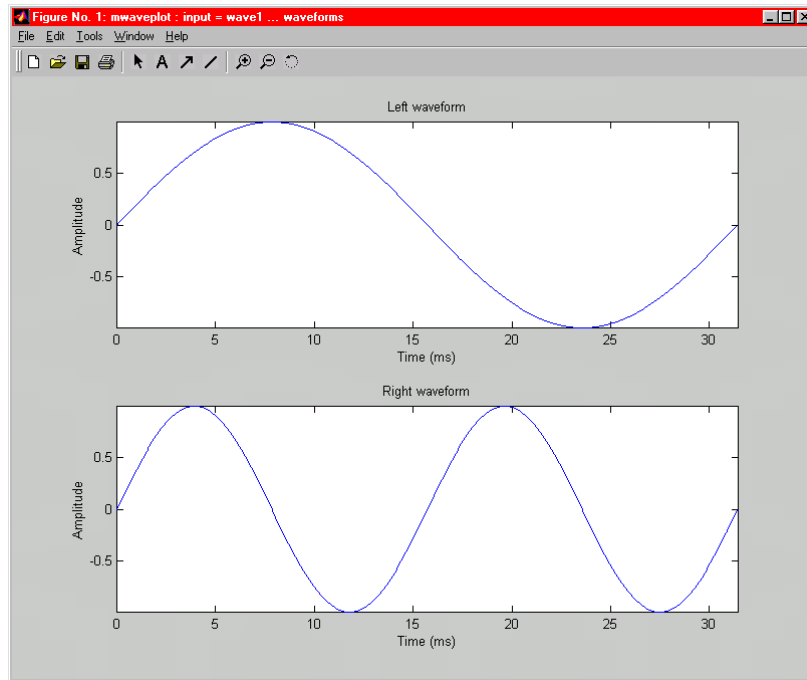
and a second vector is made which contains 2 cycles of a sinusoid of twice the frequency:

```
>> rightwaveform = sin((0:0.01:2*pi)*2);
```

then typing this will combine them into a 'wave' variable:

```
>> wave1 = mwavecreate(leftwaveform, rightwaveform, 20000, 1);
```

which has the unit-frequency sinusoid in the left channel and the twice-frequency sinusoid in the right channel:



3.8 Obtaining an FFT of a waveform

The function `mfft1side` will calculate the magnitude and phase spectrum of a waveform. The function (and the corresponding `minversefft1side`) are used internally in `mcreatenoise1`, etc., but is briefly described here in case it is useful.

To obtain the magnitude and phase spectrum of a waveform, type:

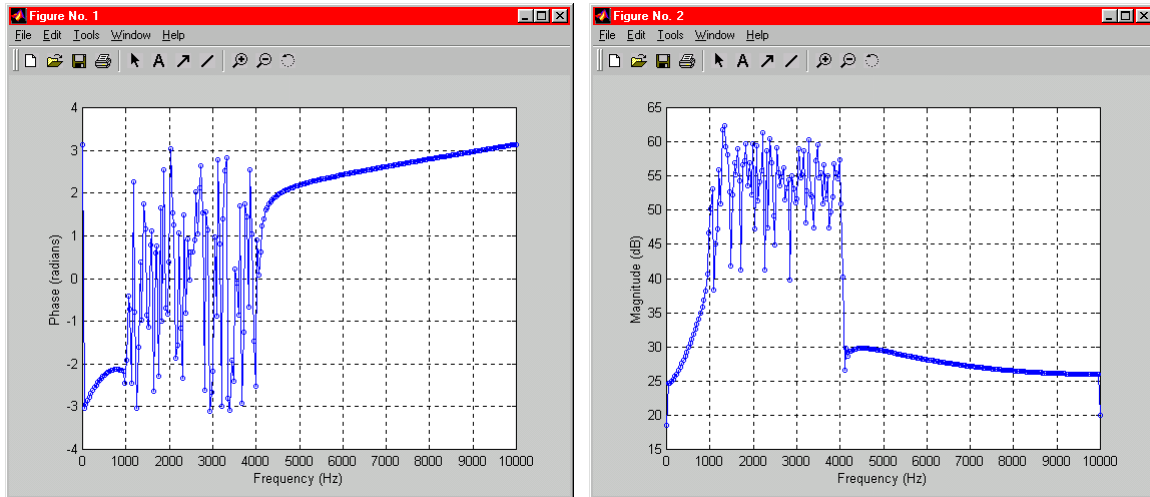
```
» fftmatrix = mfft1side(wave1.leftwaveform, 20000, 5000, 2);
```

where the first parameter is the waveform, the second parameter is the sampling frequency, the third is the number of points in the FFT and the fourth is the value of the *infoflag*. With an *infoflag* of 1 or 2 the function reports this:

```
creating 512-point FFT buffer with FFT resolution = 39.06 Hz
FFTing ...
discarding negative frequencies ...
doubling magnitudes...
scaling by number of points in FFT ...
plotting phase spectrum in figure 1 ...
plotting magnitude spectrum in figure 2 ...
storing answers as 257x3 matrix ...
```

(257 frequencies = $512/2 + 1$ (for 0-Hz component))

If *infoflag* is set to 2 then two figures are also plotted: figure 1 plots the phase spectrum and figure 2 plots the magnitude spectrum:



The output `fftmatrix` is a two-dimensional matrix with one row for each frequency in the FFT and three columns: column 1 contains the value of the frequency (in Hz), column 2 contains the value of the magnitude spectrum (in linear units not dB), and column 3 contains the phase (in radians).

Note that the first parameter of `mfft1side` is a waveform vector, not a complete 'wave' signal. This is because, at the point in `mcreatenoise1` at which it is used, only the monaural waveforms are available.

Chapter 4

Creating a binaural cross-correlogram

4.1 Calculating the correlogram

This part of the tutorial uses the dichotic bandpass noise **n** made in Section 3.1 to demonstrate the creation of a binaural cross-correlogram.

The function **mcorrelogram** will calculate the binaural cross-correlogram of a ‘wave’ signal. The full syntax is:

```
correlogram = mcorrelogram(lowfreq, highfreq, filterdensity,
                           mindelay, maxdelay,
                           transduction, binauralswitch, wave, infoflag)
```

where the parameters are:

lowfreq	Actual lower limit of the filterbank (Hz).
highfreq	Nominal upper limit of the filterbank (Hz).
filterdensity	Density of filters in the filterbank (filters per ERB).
mindelay	Minimum (most-negative) limit of the internal delay τ dimension (μ s).
maxdelay	Maximum (most-positive) limit of the internal delay τ dimension (μ s).
transduction	String controlling the type of neural transduction (see below).
binauralswitch	String controlling the type of binaural processing (see below).
wave	‘wave’ signal to be modeled.
infoflag	2 (plot pictures and report information), or 1 (report information), or 0 (do not plot or report anything).

For example, to generate the binaural cross-correlogram of the noise **n**, using a filterbank from 200 Hz to 1000 Hz with filters spaced at 2 per ERB, using a τ dimension from -3500 to 3500 μ s, using halfwave rectification as the model of neural transduction, and using cross-products as the binaural processing, type:

```
» cc1=mcorrelogram(200,1000,2,-3500,3500, 'hw', 'cp', n, 2);
```

If *infoflag* is equal to 2 (as in this example) or 1 then **mcorrelogram** will report this running information to the MATLAB terminal (although I added the line numbers for the purposes of this tutorial)

```
(1) This is mcorrelogram ...
(2) waveform length = 6000 samples = 300.0 ms
(3) sampling rate   = 20000 Hz
(4) one sample     = 50.0 us
```

```

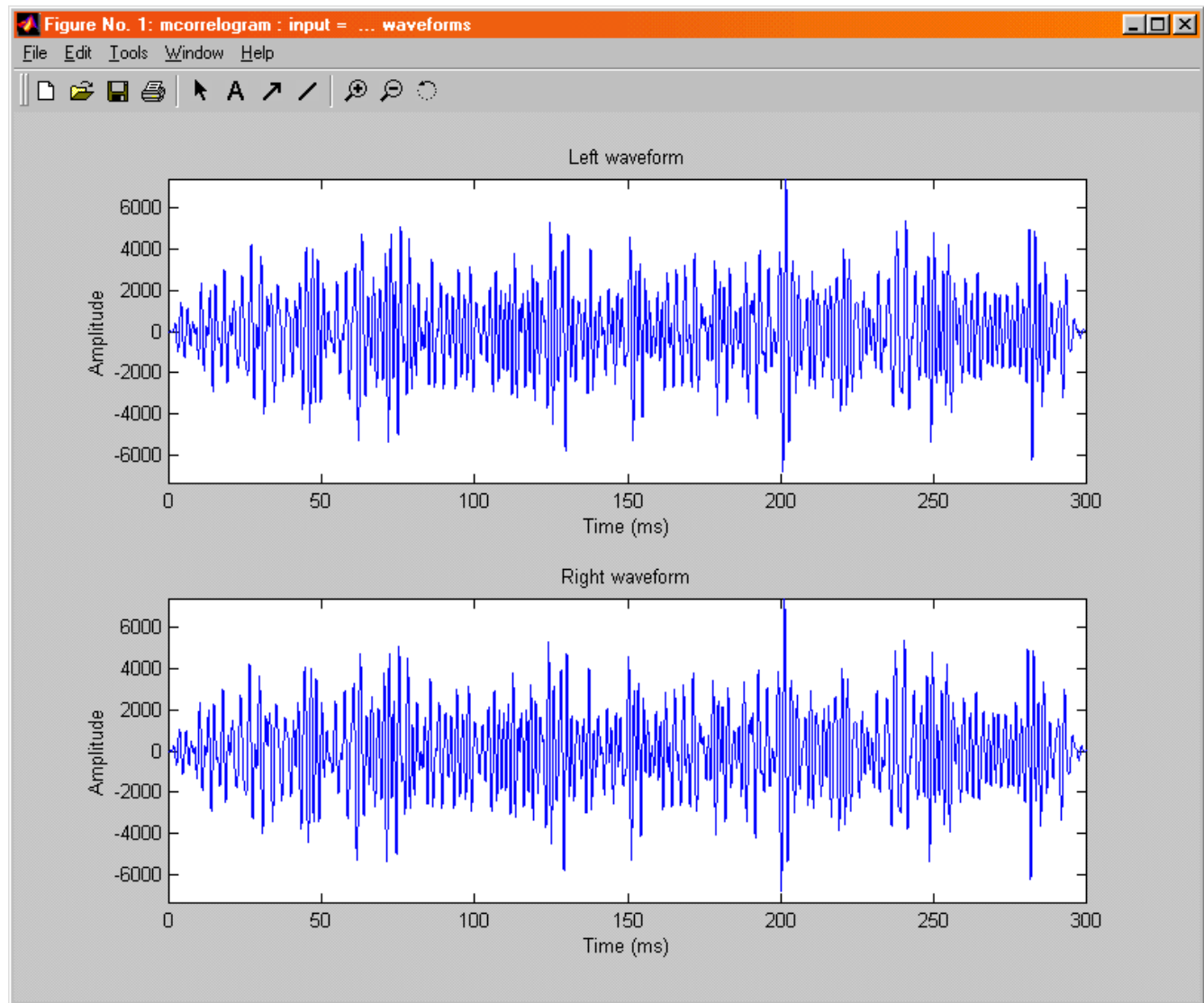
(5)
(6) applying filterbank to left waveform ...
(7) gammatone filtering from 200.0 to 1031.3 Hz (=5.83 to 15.83 ERBs)
(8) number of channels = 21 channels
(9) density = 2.0 channels per ERB
(10) q-factor = 9.27888 bwmin = 24.673
(11) applying same filterbank to right waveform ...
(12)
(13) applying transduction to the multichannel filter outputs (left channel) ...
(14) 'hw' = halfwave rectification only ...
(15) applying same transduction to the multichannel filter outputs (right channel)
(16)
(17) binaural processing ...
(18) 'cp' = measuring left x right crossproducts in each frequency channel ...
(19) (not normalized by power in each channel)
(20) (final values = unweighted averaged across full stimulus duration)
(21) number of delays = 141
(22) limits = -3500 to 3500 microsecs
(23) doing delay # 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
(24) 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
(25) 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
(26) 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
(27) 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
(28) 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
(29) 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138
(30) 139 140 141
(31)
(32) plotting waveforms in figure 1
(33) plotting monaural excitation patterns in figure 2
(34) plotting 3-dimensional correlogram in figure 3.1 using mccgramplot3dmesh
(35) plotting 3-dimensional correlogram in figure 3.2 using mccgramplot3dsurf
(36) plotting 2-dimensional correlogram in figure 3.3 using mccgramplot2dsqrt
(37) plotting across-freq average in figure 3.4 using mccgramplotaverage
(38) saving correlogram of size 21 x 141 to workspace ...

```

Lines 2-4 report the duration and sampling frequency of the input sound. Lines 6-11 report the parameters of the gammatone filterbank; in particular, the frequency range of the filterbank (line 7), the number of channels (line 8) and the q-factor and minimum-bandwidth factor of the filterbank (line 10). Lines 13-15 report the model of neural transduction. Lines 17-30 report information on the binaural processing itself, notably the type of binaural processing (line 18-20) and the size and length of the internal delay axis (lines 21-22). The numbers in lines 23-30 represent each delay along the internal delay axis and are printed in order as the calculations reach each delay. Lines 32-37 summarize the figures. Line 38 reports the size of the correlogram (number of frequency channels x number of internal delays) stored in the workspace.

If a value of 2 is used for *infoflag* than **mccorrelogram** will create three figures. These are shown on the following pages.

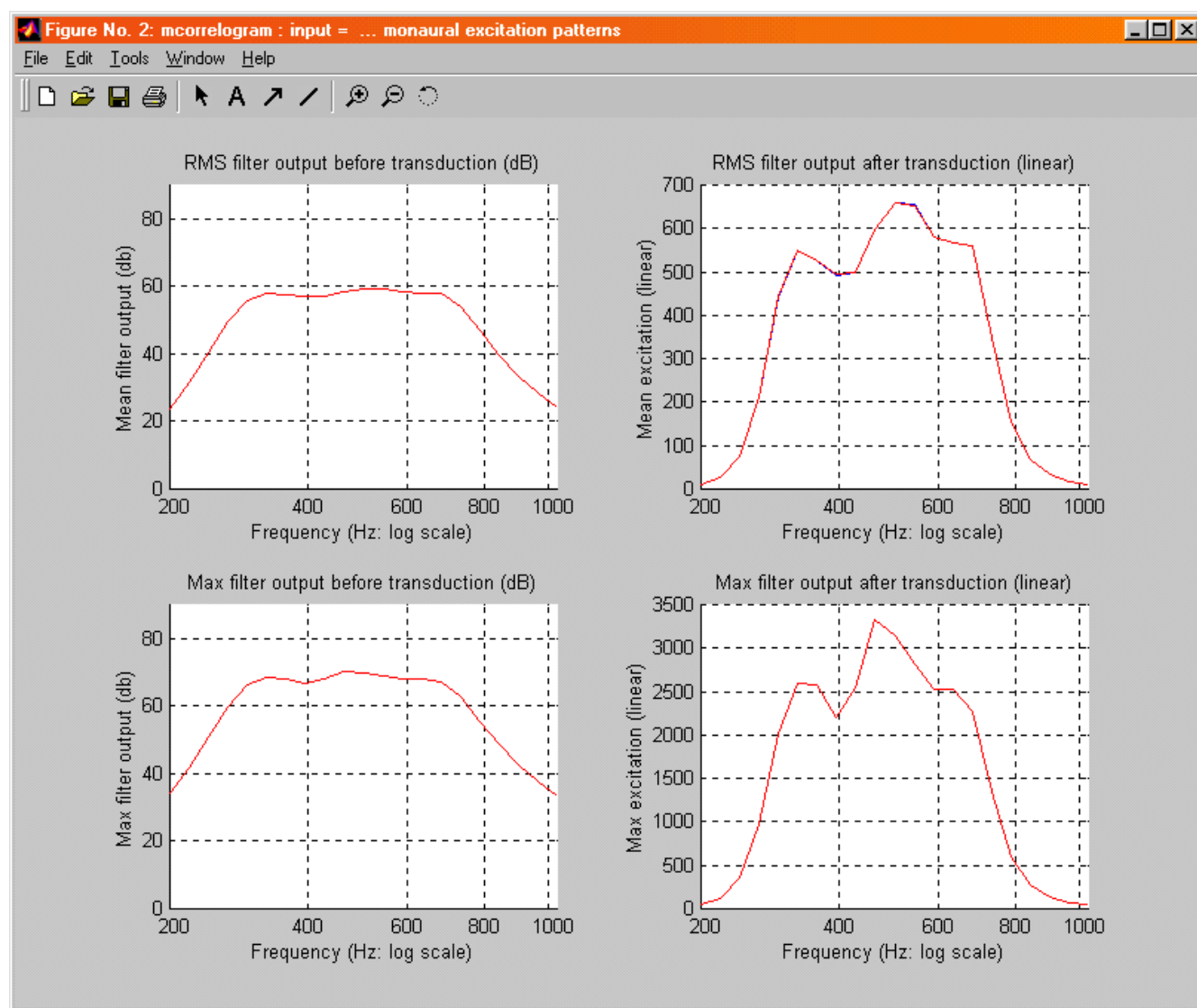
Figure 1 shows the left and right waveforms of the signal (see `mwaveplot`; Section 3.2).



The four panels of Figure 2 show the mean and maximum levels in each frequency channel of the filterbank, both before and after neural transduction:

top-left panel	Mean levels after gammatone filtering (dB y-axis).
top-right panel	Mean levels after gammatone filtering and neural transduction (linear y-axis).
bottom-left panel	Maximum level after gammatone filtering (dB y-axis).
bottom-right panel	Maximum level after gammatone filtering and neural transduction (linear y-axis).

In all the panels the blue line plots the levels for the left channel and the red line plots the levels for the right channel. In this example the levels of the left and right channels are identical and so the blue line is hidden behind the red line.

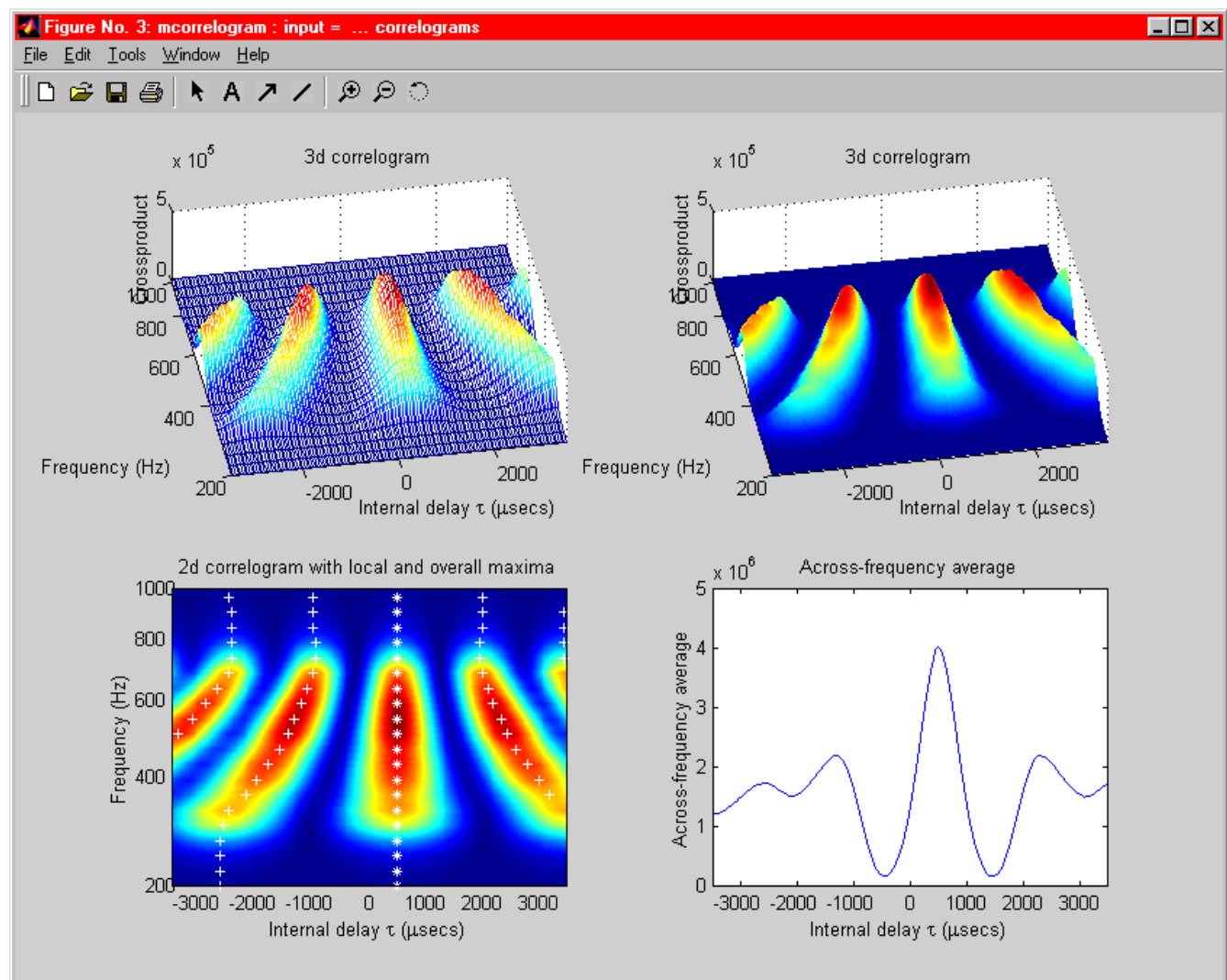


The four panels of Figure 3 show a variety of views of the binaural cross-correlogram.

top-left panel	Three-dimensional 'mesh' view.
top-right panel	Three-dimensional 'surf' (surface) view.
bottom-left panel	Two-dimensional view (looking down on the correlogram from above).
bottom-right panel	Across-frequency average of the correlogram.

The two-dimensional plot (bottom-left panel) also shows white crosses and asterisks. The crosses show the location of all the local maxima in the cross-correlation functions in each frequency channel. The asterisks show the location of the largest (overall) maximum in each cross-correlation function.

The colours are set so that dark blue is least activity, light blue and orange is intermediate, and red is most activity.



In a correlogram the peaks occur wherever the left and right filter outputs are in phase. This occurs where:

- 1 The internal delay matches the ITD of the stimulus.
- 2 Wherever the internal delay is equal to that value ± 1 period of the filter center frequency, ± 2 periods, ± 3 periods, and so on.

In the above example the noise **n** has an ITD of 500 μ s. Hence there is a across-frequency peak at $\tau=500 \mu$ s. This peak generates the "ridge" that is shown in the three colour plots. It also generates the single large peak in the plot of the across-frequency average. The other ridges in the colour plots represent the internal delays corresponding to ± 1 periods and ± 2 periods. These ridges are curved because the period of each frequency channel is inversely proportional to the center frequency of the channel.

Chapter 5 outlines how the lateralization of a sound is calculated from the correlogram.

4.2 Plotting a correlogram

Each of the panels shown in figure 3 of **mccorrelogram** can also be plotted separately, as the code that draws them is provided in separate functions:

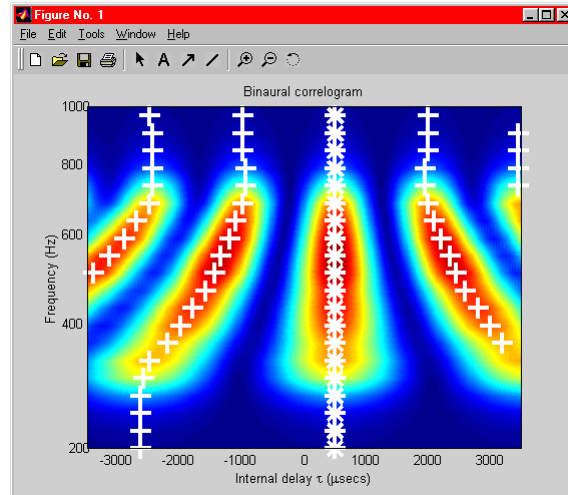
mccgramplot4panel	Plot all 4 panels in one figure (as illustrated above).
mccgramplot3dmesh	Plot the three-dimensional 'mesh' view (uses the MATLAB function mesh).
mccgramplot3dsurf	Plot the three-dimensional 'surf' view (uses the MATLAB function surf).
mccgramplot2dsqrt	Plot the two-dimensional view (uses the MATLAB function pcolor).
mccgramplotaverage	Plot the across-frequency average of the correlogram.

For all of those five functions the only parameter is the name of the correlogram (note that these functions do not have an *infoflag* parameter). For example, to plot the correlogram **cc1** as a 3d-mesh, type:

```
>> mccgramplot3dmesh(cc1)
```

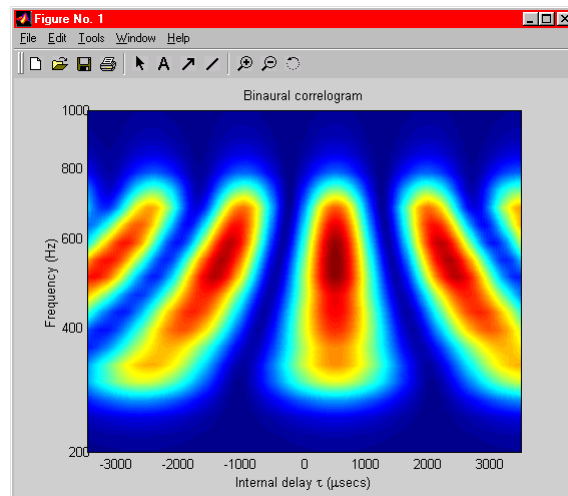
The function `mccgramplot2dsqrt` can have two optional parameters that control the size (parameter #2) and linewidth (parameter #3) of the symbols. For example, to have super-big and super-wide symbols, type:

```
» mccgramplot2dsqrt(cc1, 15, 3)
```



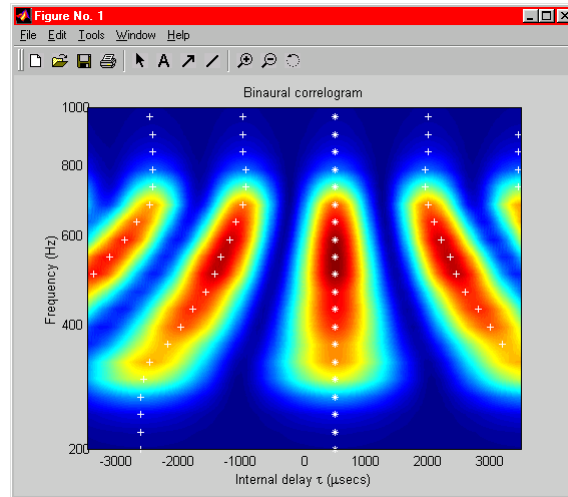
If values of 0 are used for both parameters #2 and #3 then the symbols are not plotted:

```
» mccgramplot2dsqrt(cc1, 0, 0)
```



To use the default values (`symbolsize = 5`, `linewidth = 1`), do not specify the parameters:

```
» mccgramplot2dsqrt(cc1)
```



These parameters can also be specified in `mccgramplot4panel` which will then use them in its call to `mccgramplot2dsqrt`.

4.3 Models of neural transduction

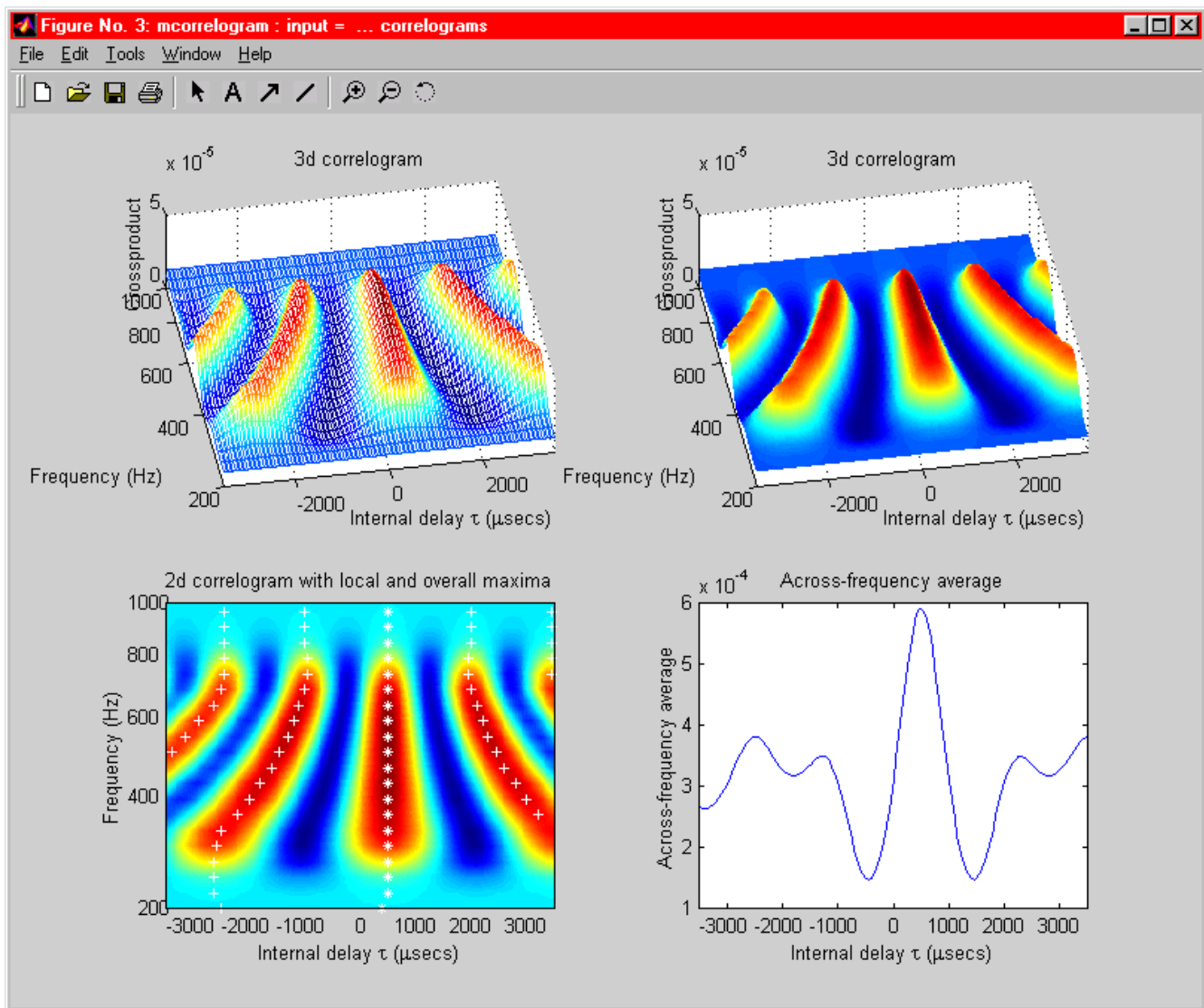
A variety of different models of neural transduction are available (parameter #6 of `mccorrelogram`). They are:

<code>'linear'</code>	No extra processing; use the actual output of the gammatone filters.
<code>'hw'</code>	Apply halfwave rectification to the filter outputs.
<code>'log'</code>	Apply halfwave rectification to the filter outputs and then apply logarithmic compression.
<code>'power'</code>	Apply halfwave rectification to the filter outputs and then power-law compression (using an exponent of 0.4).
<code>'v=3'</code>	Apply halfwave rectification to the filter outputs and then power-law expansion (using an exponent of 3; see Shear, 1987, and Stern and Shear, 1996).
<code>'envelope'</code>	Apply the Bernstein et al. envelope-compression algorithm to the filter outputs (see Bernstein and Trahiotis, 1996b; Bernstein et al., 1999).
<code>'meddishigh'</code>	Apply the Meddis haircell model to the filter outputs, using a 'high'-spontaneous rate fiber (see Meddis et al., 1990, Table II).
<code>'meddismedium'</code>	Apply the Meddis haircell model to the filter outputs, using a 'medium'-spontaneous rate fiber (see Meddis et al., 1990, Table II).

The `'linear'`, `'hw'`, `'log'`, `'power'` and `'v=3'` options are all reasonably simple (although note that `'v=3'` does an expansion not a compression). The `'envelope'` option is more complicated and does this: the envelope of the signal is extracted using a Hilbert transform, compressed to the power of 0.23, then halfwave rectified, then expanded to the power of 2.0, and finally lowpass filtered at a corner frequency of 425 Hz. The envelope compression itself is described in Bernstein et al. (1999); the lowpass filtering is described in Bernstein and Trahiotis (1996b). The `'meddishigh'` and `'meddislow'` options are new implementations of the BASIC code in Meddis et al. (1990) for simulating the probability of firing of an inner hair cell. The two sets of parameters correspond to the two columns in Table II of that paper. Note that both `'meddishigh'` and `'meddislow'` add a short duration of silence before and after the signal.

For example, the next picture shows the same correlogram but calculated using `'meddishigh'` instead of `'hw'`:

```
>> cc1=mcorrelogram(200,1000,2,-3500,3500, 'meddishigh', 'cp', n, 2);
```



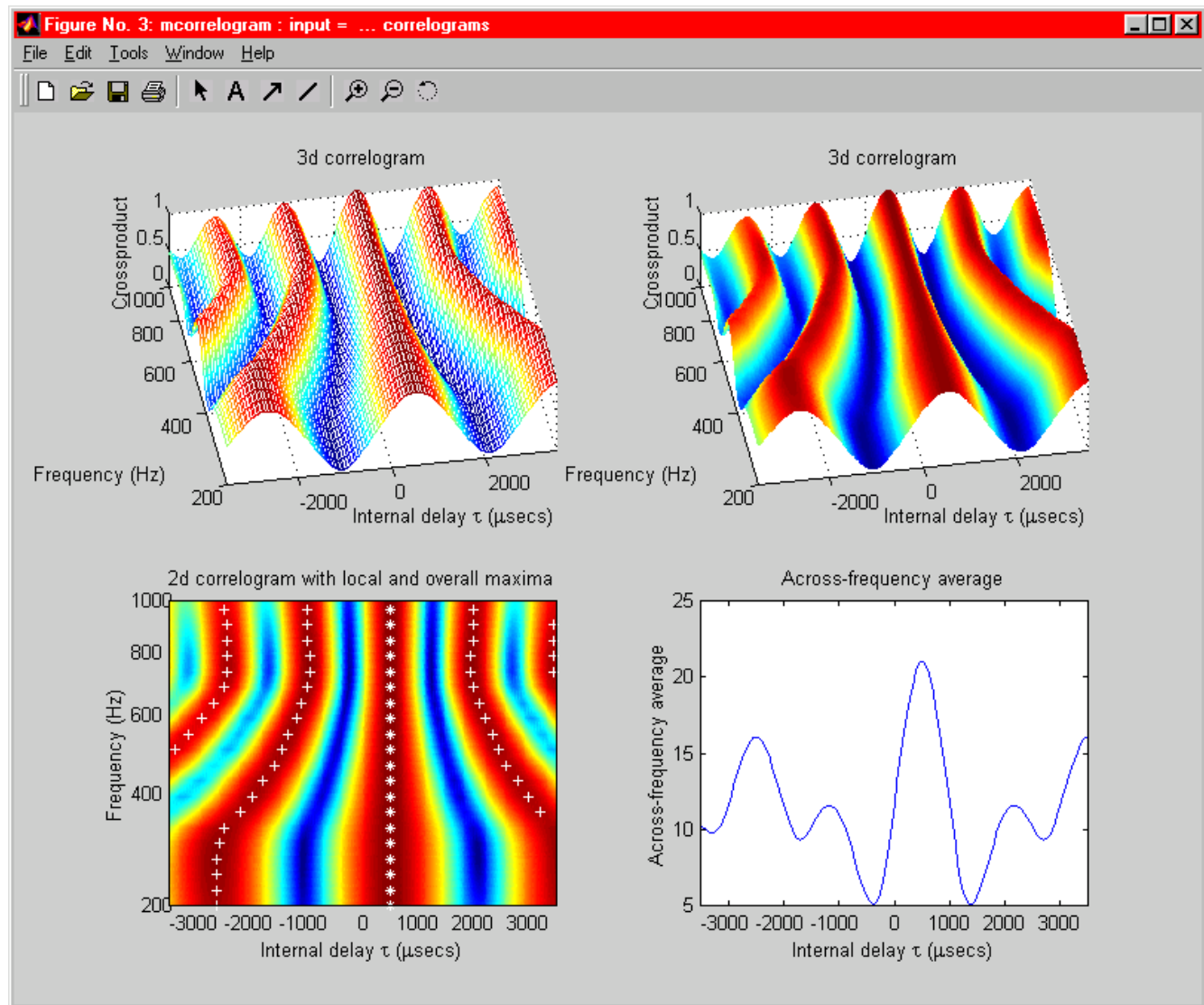
4.4 Types of binaural processing

Three different types of binaural processing are available (parameter #7 of `mcorrelogram`). They are

- 'cp' Cross product: Cross-multiply left and right filter .
- 'cc' Cross correlation: Cross-multiply left and right filter outputs and then normalize by dividing by the average power in the two filter outputs.
- 's' Subtraction: Subtract the right filter output from the left filter output.

The next picture shows the correlogram of the same noise `n` as before but calculated using cross-correlation ('cc') instead of cross-products ('cp')

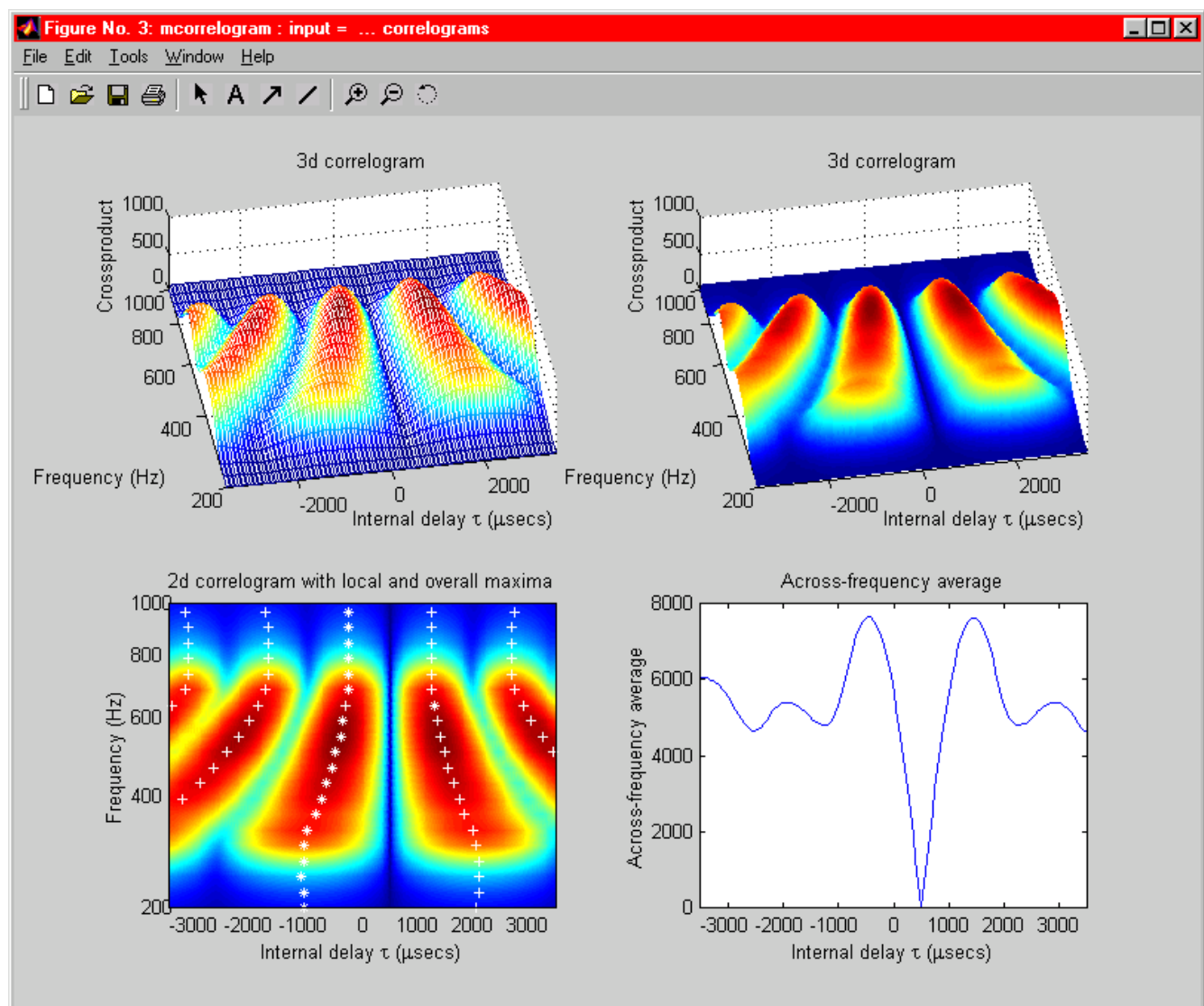
```
>> cc1=mcorrelogram(200,1000,2,-3500,3500, 'hw', 'cc', n, 2);
```



Subtraction ('s') is a different process. In cross-correlograms made using 'cp' or 'cc' the peaks occur whenever the left and right filter outputs are in phase, and the valleys occur where the filter outputs are out of phase. In subtraction the situation is reversed: the peaks occur where the filter outputs are out of phase, and so the valleys occur where the filter outputs are in phase. This is shown in the next figure, which shows the correlogram of the same noise *n* but generated using the subtraction option: note that there is now a deep valley at an internal delay of 500 μ s.

```
» cc2=mcorrelogram(200,1000,2,-3500,3500, 'hw', 's', n, 2);
```

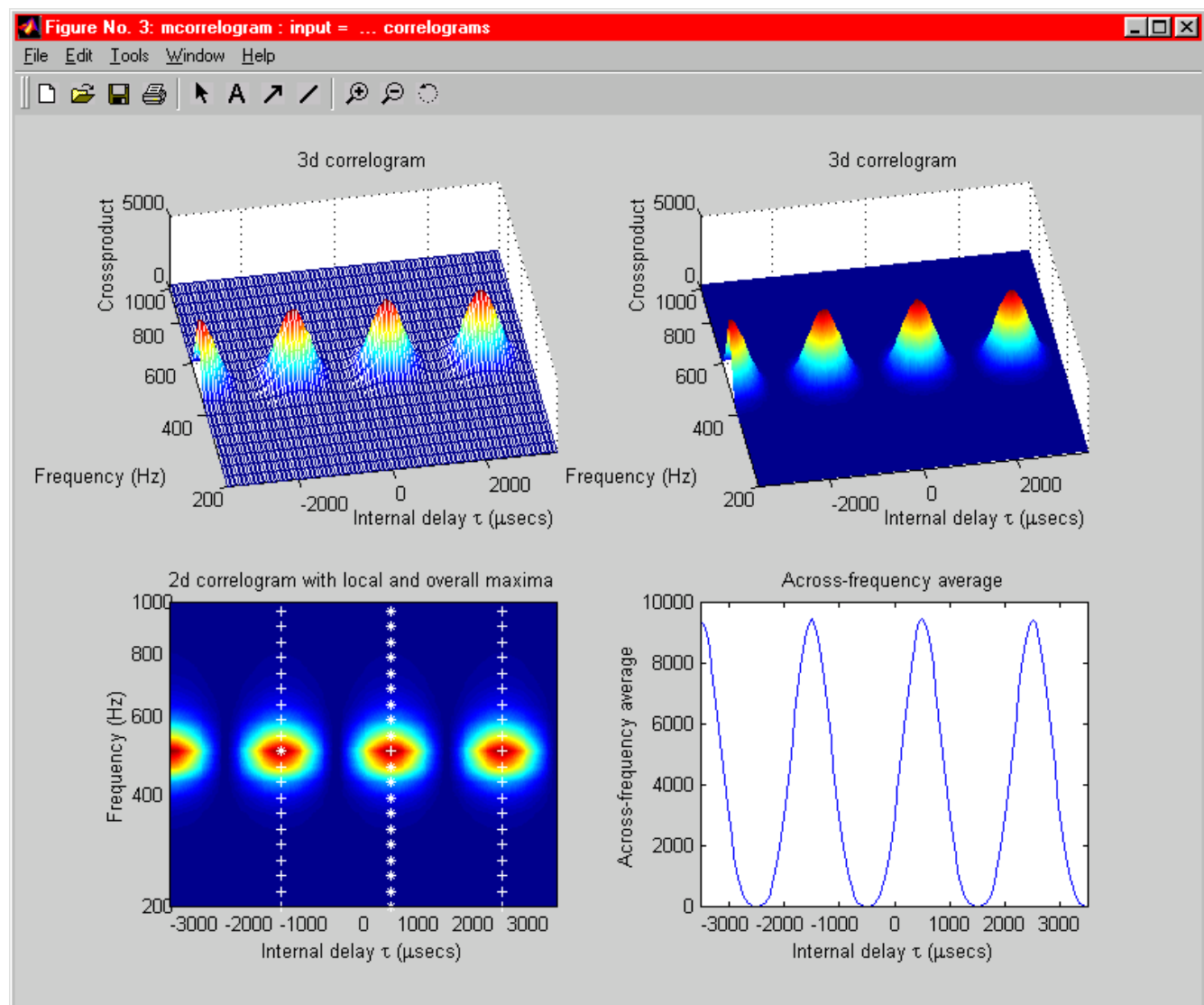
The subtraction option is not intended for measuring lateralization. Instead, it is intended as a step in the calculation of the recovered spectrum of the modified Equalization-Cancellation model, which is one of the functions performed by the Windows program `ccdisplay.exe` (see Chapter 6).



4.5 More examples of binaural correlograms

4.5.1 500-Hz pure tone with 500- μ s ITD

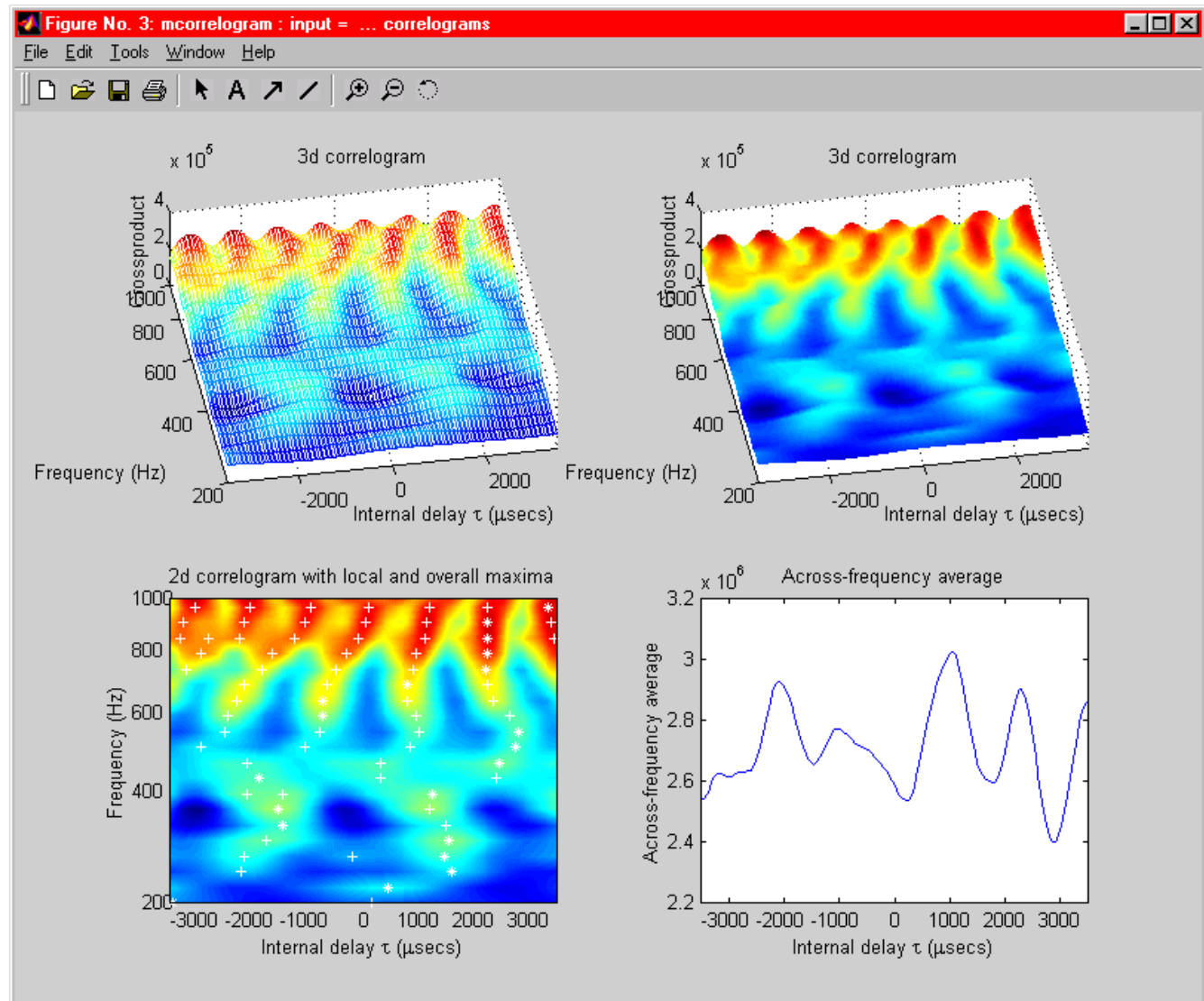
```
» wave1=mcreatetone(500,40,40,500,0,250,10,20000,0);
» cc1=mcorrelogram(200,1000,2,-3500,3500, 'hw', 'cp', wave1, 2);
```



4.5.2 Interaurally-decorrelated broadband noise

```
» wavel=mcreatenoise2rho(0, 2000, 40, 40, 0, 500, 10, 20000, 0);
» ccl=mccorrelogram(200,1000,2,-3500,3500, 'hw', 'cp', wavel, 2);
```

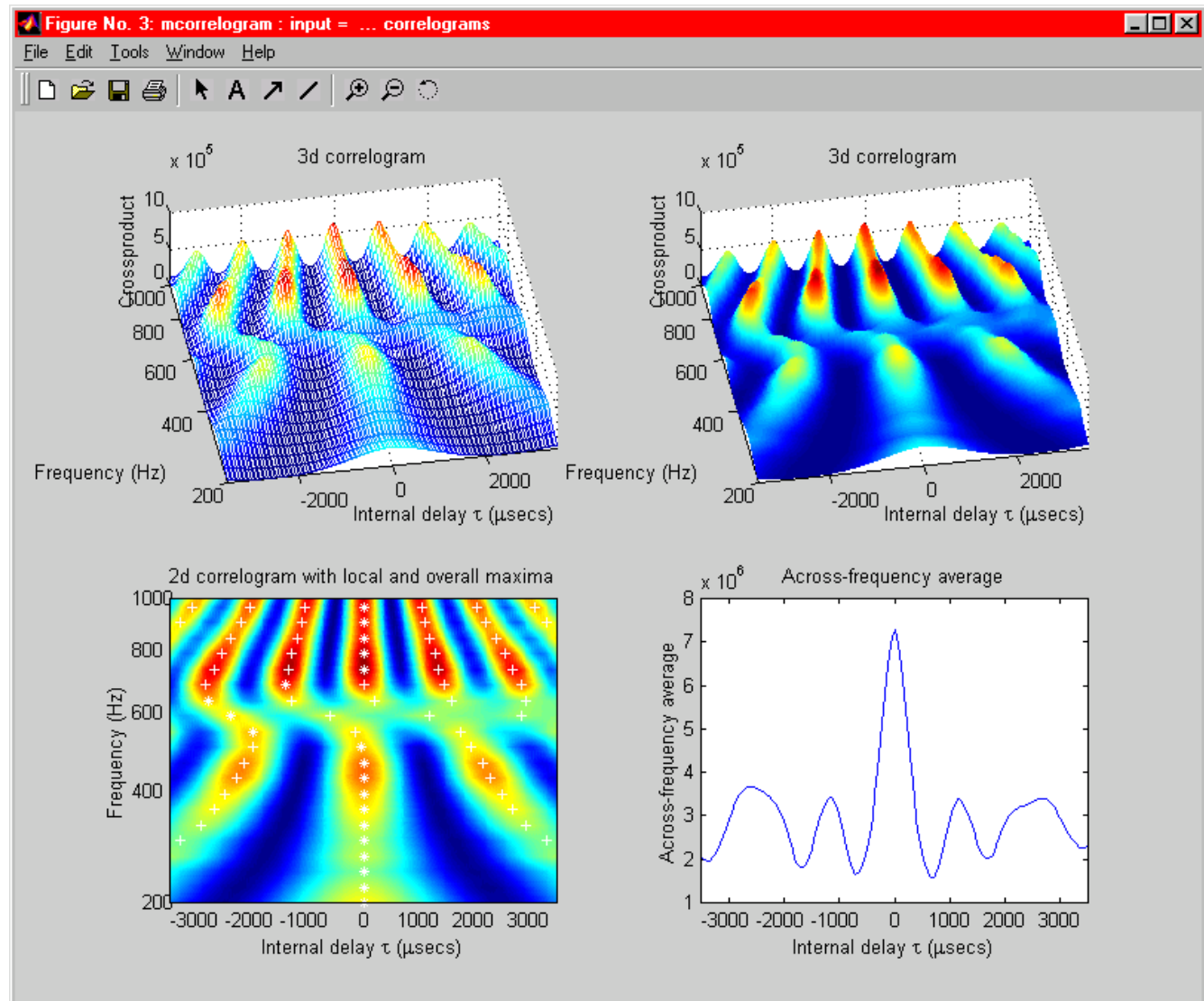
The correlogram is effectively random.



4.5.3 600-Hz, 16% Huggins pitch

```
» wavel=mcreatehuggins2(600, 16, 0, 2000, 40, 40, 0, 0, 250, 10, 20000, 0);
» ccl=mcorrelogram(200,1000,2,-3500,3500, 'hw', 'cp', wavel, 2);
```

The transition in interaural phase centered at 600 Hz creates the shift in the tracks near 600 Hz.



Chapter 5

Post-processing a binaural cross-correlogram.

This part of the tutorial describes how a prediction of the lateralization of a sound is obtained from the binaural cross-correlogram. It begins with a discussion of frequency and delay weighting, as such weightings are often applied to the correlogram—*before* a prediction of lateralization is made—in order to emphasize certain frequencies and certain delays.

5.1 Frequency weighting

The function `mccgramfreqweight` will weight the correlogram in frequency, so that certain frequencies (near 300 Hz or 600 Hz) are emphasized more than other frequencies. The syntax is

```
[output_correlogram, output_freqweight] = mccgramfreqweight(correlogram,
                                                             fswitch, infoflag);
```

where the parameters are:

<code>correlogram</code>	Correlogram to be weighted.
<code>fswitch</code>	Which weighting-function to apply (see below).
<code>infoflag</code>	1 or 0.

For example, to apply the Stern et al. (1988) weighting function, type:

```
> [cc2, ccw] = mccgramfreqweight(cc1, 'stern', 1);

creating stern et al's freq weighting function ...
applying function ...
```

Note that the function has two outputs. The first (here `cc2`) is the frequency-weighted correlogram. The second (here `ccw`) is the frequency-weighting function. Both outputs are stored in the 'correlogram' format.

Three types of frequency weighting are available:

<code>'stern'</code>	Stern et al.'s (1980) fit to Raatgever's (1980) data on the dominance of certain frequencies in lateralization. The function has a maximum at about 600 Hz.
<code>'raatgever'</code>	Raatgever's (1980) own fit to his data on the dominance of certain frequencies in lateralization. The function has a maximum at about 600 Hz.

'mld' A roex (rounded-exponential) fit to the dependence of the N0S0-N0S π masking level difference ("MLD") as a function of frequency. This function was developed by Akeroyd and Summerfield (1999) in a temporal model of dichotic pitch. It has a peak at 300 Hz.

These functions are described further in Appendix 1.

Note that `mccgramfreqweight` fills in the *freqweight* field of the correlogram structure with the name of the weighting function:

```
» cc2
cc2 =
        title: 'first-level correlogram'
        ...
        freqweight: 'stern'
        ...
```

Illustrations of frequency-weighted correlograms are shown in Section 5.3.

5.2 Delay weighting

The function `mccgramdelayweight` will weight the correlogram by internal delay, so that values at internal delays near 0 are emphasized and values at internal delays distant from 0 are attenuated. The delay-weighting function is often referred to as the " $p(\tau)$ " function. The syntax is:

```
[output_correlogram, output_freqweight] = mccgramdelayweight(correlogram,
                                                             ptswitch, infoflag);
```

where the parameters are:

<code>correlogram</code>	Correlogram to be weighted.
<code>ptswitch</code>	Which weighting-function to apply (see below).
<code>infoflag</code>	1 or 0.

For example, to apply the Shear (1987) delay-weighting function, type:

```
» [cc2, ccw] = mccgramdelayweight(cc1, 'shear', 1);

creating delay-weighting function: Shear/Stern, normalized so area = 1:
Kh = 3000 sec-1  flattop = +/- 200 usecs  lf = 0.1  lp = -1.1
applying function ...
```

As with `mccgramfreqweight`, two outputs are created: the first (here `cc2`) is the delay-weighted correlogram. The second (here `ccw`) is the delay-weighting function. Again both outputs are stored in the 'correlogram' format.

Three types of delay-weighting functions are available

'colburn'	Colburn's (1977) frequency-independent function.
'shear'	Shear's (1987; Stern and Shear, 1996) frequency-dependent function.
'shackleton'	Shackleton et al's (1992) frequency-independent function.

All of these function have a maximum at $\tau=0$ but differ in their shape and width. They are described further in Appendix 2.

`mccgramdelayweight` fills in the *delayweight* field of the correlogram structure with the name of the weighting function:

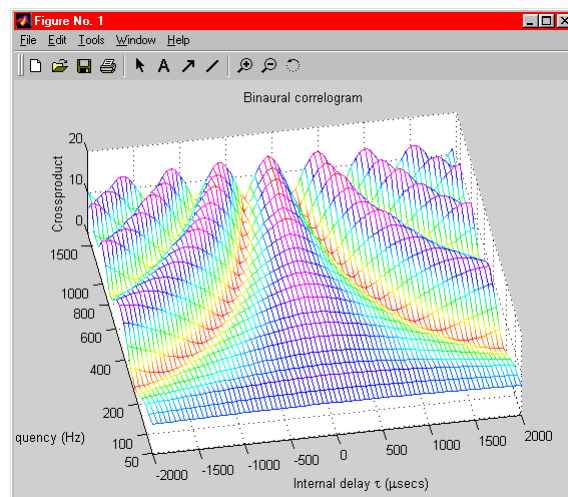
```
>> cc2
cc2 =
    title: 'first-level correlogram'
    ...
    delayweight: 'shear'
    ...
```

Illustrations of delay-weighted correlograms are shown in Section 5.3.

5.3 Illustrations of the weighting functions

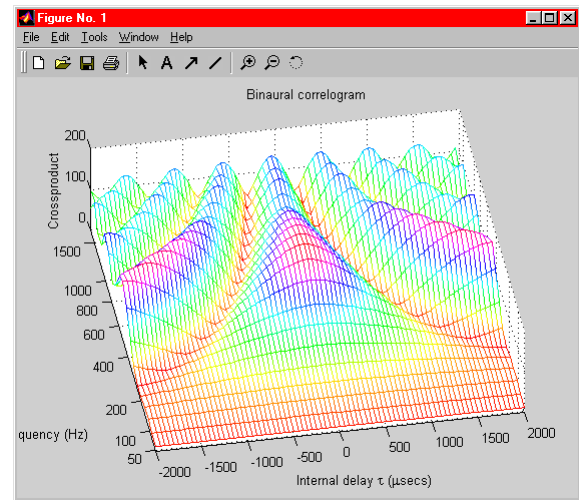
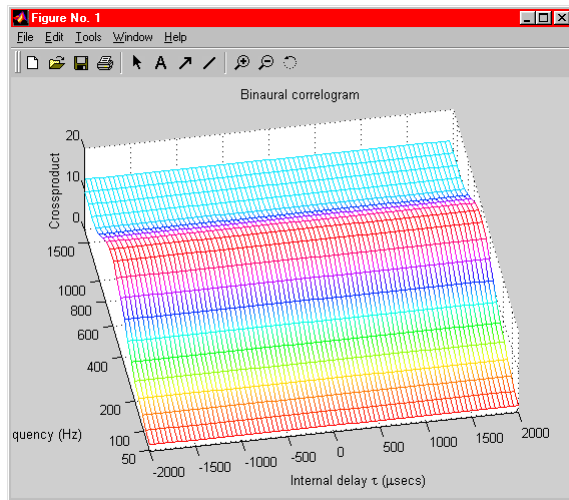
The effects of these weightings are shown in the following pictures. First, the correlogram of a broadband diotic noise was constructed thus:

```
>> n = mcreatenoise2(0, 2000, 40, 40, 0, 0, 250, 10, 20000, 0);
>> cc1 = mcorrelogram(47.4, 1700, 1, -2000, 2000, 'power', 'cp', n, 0);
>> mccgramplot3dmesh(cc1);
```



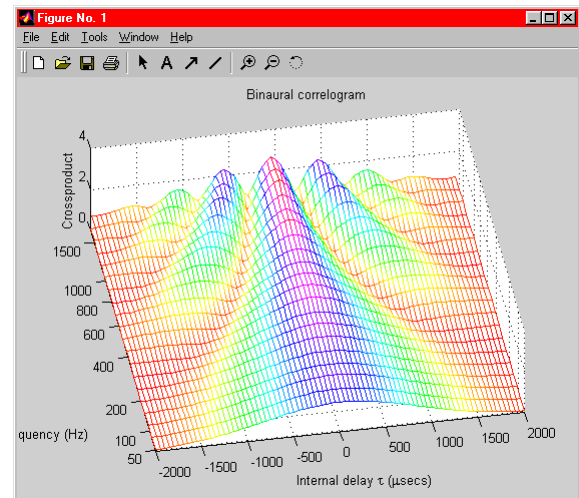
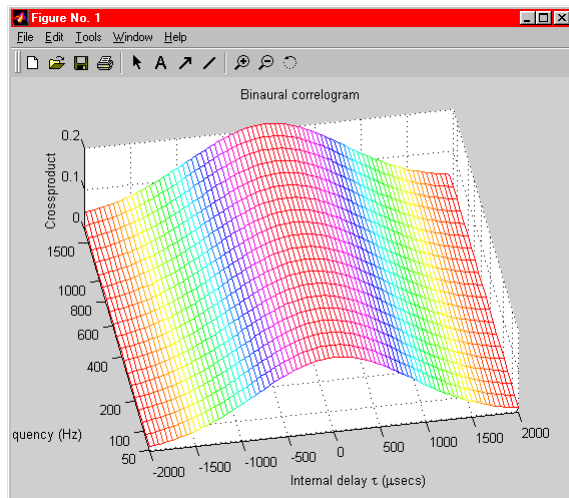
The next two pictures show the effect of applying the **'stern'** frequency weighting. The left panel shows the frequency-weighting function and the right panel shows the frequency-weighted correlogram. Note that frequencies near 600 Hz have been emphasized and that other frequencies, especially below about 200 Hz, have been attenuated.

```
» [cc2 ccw] = mccgramfreqweight(cc1, 'stern', 0);
» mccgramplot3dmesh(ccw);
» mccgramplot3dmesh(cc3);
```



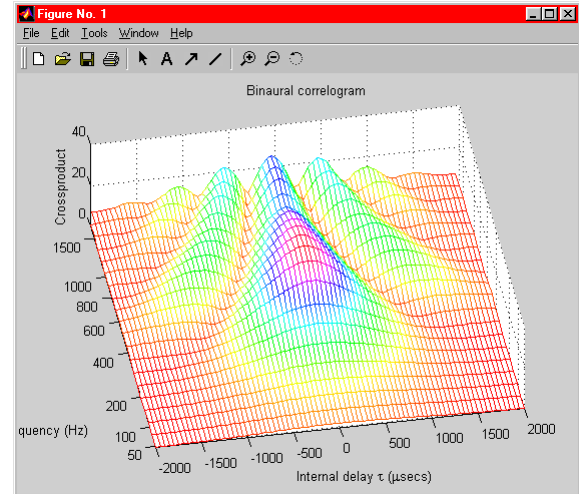
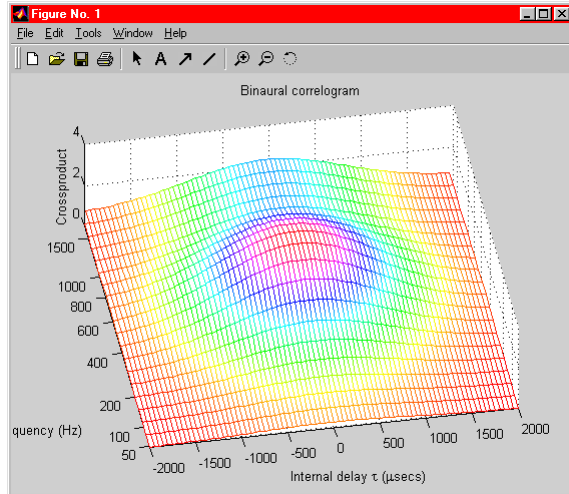
The next two pictures show the effect of applying the **'shackleton'** delay weighting. The left panel shows the delay-weighting function and the right panel shows the delay-weighted correlogram. Note that internal delays near 0 have been emphasized.

```
» [cc2 ccw] = mccgramdelayweight(cc1, 'shackleton', 0);
» mccgramplot3dmesh(ccw);
» mccgramplot3dmesh(cc3);
```



The final two pictures show the effect of applying the '**shackleton**' delay weighting in conjunction with the '**stern**' frequency weighting. The left panel shows the combined-weighting function and the right panel shows the frequency-&-delay-weighted correlogram.

```
» [cc2 ccw] = mccgramdelayweight(cc1, 'shackleton', 0);
» cc3 = mccgramdelayweight(cc2, 'shackleton', 0);
» ccww = mccgramdelayweight(ccw, 'shackleton', 0);
» mccgramplot3dmesh(ccww);
» mccgramplot3dmesh(cc3);
```



5.4 Lateralization predictions: The centroid

A major use of the cross-correlogram is to predict the lateralization of a sound. The most common measures of lateralization are

- The centroid of the across-frequency average of the correlogram.
- The position of the largest peak in the across-frequency average of the correlogram.
- The position of the “best” (closest to $\tau=0$) peak in the across-frequency average of the correlogram.

The function `mccgramcentroid` will calculate the centroid of the across-frequency average of a correlogram. For example,

```
» p = mccgramcentroid(cc1, 1);
```

where `cc1` is a previously made correlogram and 1 is the *infoflag*. The value of the centroid (in μ s) is returned in the workspace variable `p`.

In this first example the noise is diotic and so the centroid is at 0 μ s:

```
» n = mcreatenoise1(500,400,40,40,0,0,250,10,20000,0);
» cc1 = mcorrelogram(47.4, 1690, 1, -3500, 3500, 'power', 'cp', n, 0);
» cc2 = mccgramfreqweight(cc1, 'stern', 0);
» cc3 = mccgramdelayweight(cc2, 'colburn', 0);
» p = mccgramcentroid(cc3, 1);
```

Calculating centroid = $\text{sum}(t * \text{correlogram}(t,f)) / \text{sum}(\text{correlogram}(t,f))$
(sums are over delay t and frequency f)

centroid = -0.000 usecs

In this second example the noise is given an ITD of 100 μ s. The centroid is at 29 μ s.

```
» n = mcreatenoise1(500,400,40,40,100,0,250,10,20000,0);
» cc1 = mcorrelogram(47.4, 1690, 1, -3500, 3500, 'power', 'cp', n, 0);
» cc2 = mccgramfreqweight(cc1, 'stern', 0);
» cc3 = mccgramdelayweight(cc2, 'colburn', 0);
» p = mccgramcentroid(cc3, 1);
```

Calculating centroid = $\text{sum}(t * \text{correlogram}(t,f)) / \text{sum}(\text{correlogram}(t,f))$
(sums are over delay t and frequency f)

centroid = 29.392 usecs

In this third example the noise is given an ITD of 200 μ s. The centroid is at 56 μ s. This value is slightly less than twice the previous value, even though the ITD was increased by a factor of 2: for small ITDs, the centroid is approximately proportional to the ITD.

```
» n = mcreatenoise1(500,400,40,40,500,0,250,10,20000,0);
» cc1 = mcorrelogram(47.4, 1690, 1, -3500, 3500, 'power', 'cp', n, 0);
» cc2 = mccgramfreqweight(cc1, 'stern', 0);
» cc3 = mccgramdelayweight(cc2, 'colburn', 0);
» p = mccgramcentroid(cc3, 1);
```

Calculating centroid = $\text{sum}(t * \text{correlogram}(t,f)) / \text{sum}(\text{correlogram}(t,f))$
(sums are over delay t and frequency f)

centroid = 56.036 usecs

5.5 Lateralization predictions: The best peak and the largest peak.

The function `mccgrampeak` will calculate the position of the peak in the across-frequency average of a correlogram. The function will calculate either the “best” peak or the largest peak.

The “best” peak is whichever peak is closest to $\tau=0$. For example:

```
>> p = mccgrampeak(cc1, 'best', 1);
```

where `cc1` is a previously made correlogram and 1 is the *infoflag*. The location of the peak (in μs) is returned in the workspace variable `p`.

The “largest” peak is whichever peak has the largest value no matter what its location along the internal-delay axis. For example:

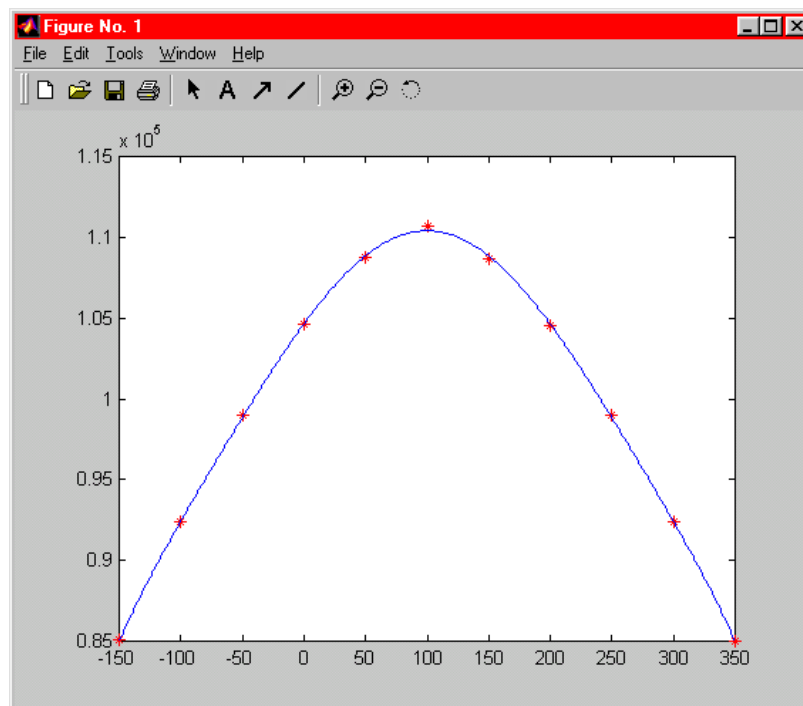
```
>> p = mccgrampeak(cc1, 'largest', 1);
```

where `cc1` is a previously made correlogram and 1 is the *infoflag*. The location of the peak (in μs) is returned in the workspace variable `p`.

`mccgrampeak` uses the MATLAB function `polyfit` to interpolate the position of the peak to better resolution than the sampling resolution of the internal delay axis. A 6th-order function is used, and the range of points in the fit is $\pm 250 \mu\text{s}$ about the peak. The fit can be plotted by setting *infoflag* to be 2; for example:

```
>> p = mccgrampeak(cc1, 'largest', 2);
```

In the plot the data points are shown as asterisks and the interpolated function from `polyfit` is shown as the solid line:



In this first example the noise is given an ITD of 100 μ s. Both the largest peak and the best peak are at 100 μ s.

```
» n = mcreatenoise1(500, 400, 40, 40, 100, 0, 250, 10, 20000, 0);
» cc1 = mcorrelogram(47.4, 1690, 1, -3500, 3500, 'power', 'cp', n, 0);
» cc2 = mccgramfreqweight(cc1, 'stern', 1);

» p = mccgrampeak(cc2, 'largest', 1);
averaging across frequency ...
finding largest peak ...
location: 73 samples = 100 usecs
interpolating using 6 order 'polyfit' ...
interpolated position = 100 usecs

» p = mccgrampeak(cc2, 'best', 1);
averaging across frequency ...
finding best (most-central) peak ...
location: 73 samples = 100 usecs
interpolating using 6 order 'polyfit' ...
interpolated position = 100 usecs
```

In this second example the noise is given an ITD of 1525 μ s. The largest peak is at 1526 μ s and so marks the correct ITD (note that the `polyfit` interpolation has almost perfectly reconstructed the correct delay, even though it falls between samples along the internal delay dimension). The best peak, however, represents one of the slip cycles generating one of the other ridges in the correlogram (see notes on p. 36). It is at -236 μ s, and so is (1) closer to midline and (2) on the other side than the largest peak.

```
» n = mcreatenoise1(500, 400, 40, 40, 1525, 0, 250, 10, 20000, 0);
» cc1 = mcorrelogram(47.4, 1690, 1, -3500, 3500, 'power', 'cp', n, 0);
» cc2 = mccgramfreqweight(cc1, 'stern', 1);

» p = mccgrampeak(cc2, 'largest', 2);
averaging across frequency ...
finding largest peak ...
location: 102 samples = 1550 usecs
interpolating using 6 order 'polyfit' ...
interpolated position = 1526 usecs

» p = mccgrampeak(cc2, 'best', 1);
averaging across frequency ...
finding best (most-central) peak ...
location: 66 samples = -250 usecs
interpolating using 6 order 'polyfit' ...
interpolated position = -236 usecs
```

Chapter 6

Displaying a Correlogram in Windows

A separate Windows 95/98 program `ccdisplay.exe` is provided for displaying binaural cross-correlograms. To run the program from MATLAB, type:

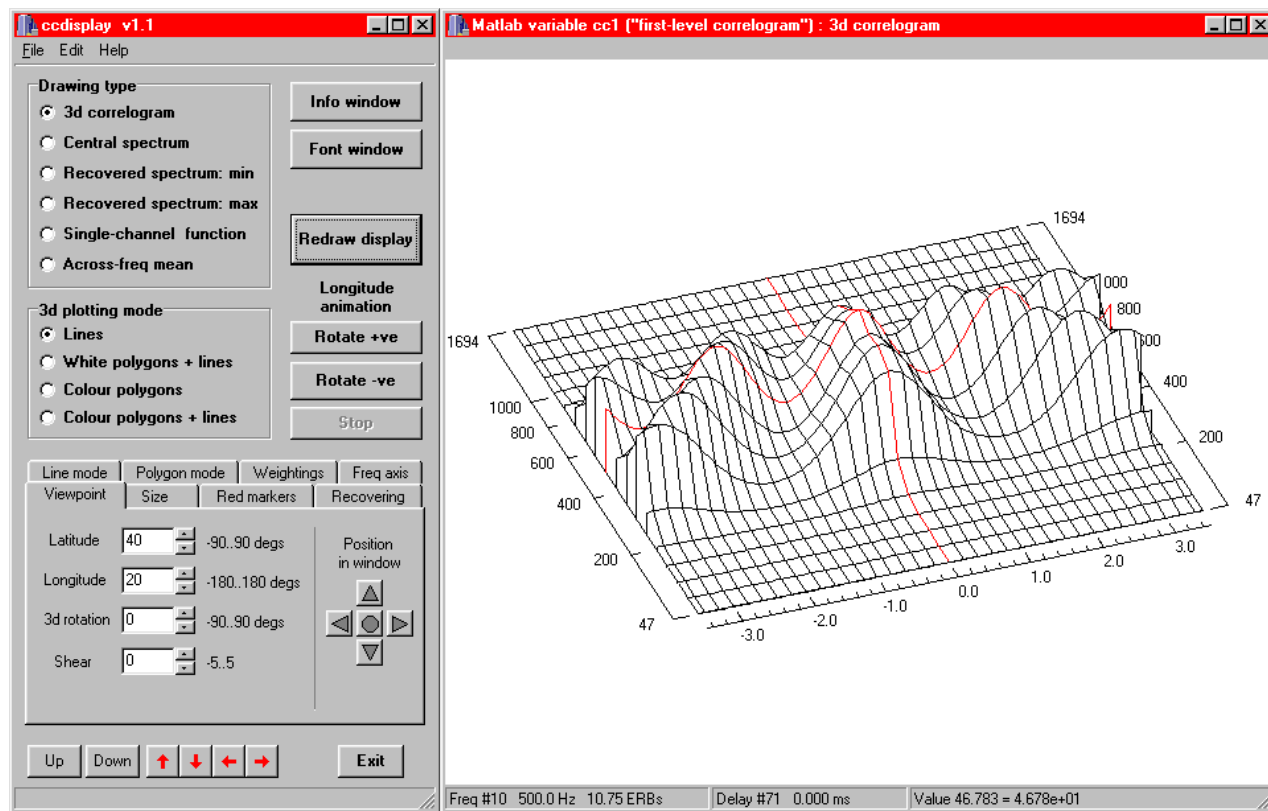
```
» mcallccdisplay(cc1);
```

where `cc1` is a previously made correlogram. This program saves the correlogram (using a special file format that can be read by `ccdisplay`) and then calls `ccdisplay` using the `dos` function in MATLAB.

`mcallccdisplay` will report this information to the MATLAB window:

```
writing header to tempmatlab.bcc ...  
(including filter center frequencies in output file)  
writing 21x141 data points to tempmatlab.bcc ...  
  
calling Windows-95 program ccdisplay.exe ...  
(using path d:\uchc\export\release6\)  
  
returning to MATLAB workspace.
```

Two windows should pop up. The one on the left is the Control Window and the one on the right is the Display Window. These windows are shown on the next page. The program is controlled by the buttons and switches in the Control Window.



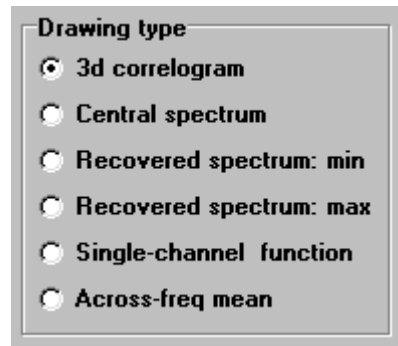
6.1 Basic options

The `ccdisplay` program will:

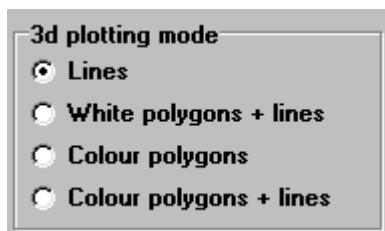
- Display a correlogram in three dimensions from any viewpoint.
- Show only a subset of frequency channels or internal delays.
- Calculate central spectra, recovered spectra, single-channel cross-correlation functions, and the across-frequency average function.
- Apply frequency or delay weightings.
- Report the value at any point in the cross-correlogram.

My goal in writing `ccdisplay` was to have a program that could display a correlogram in a variety of ways, that could perform simple transformations on the correlogram, and that could report the values of the numbers in the correlogram. All of the displays and functions were incorporated because, at some point, I needed to do them in analysing particular correlograms. The result is the Control Window has a lot of buttons, switches, and dialog boxes. Clicking most of these will immediately show what they do. My favourite is the `Rotate +ve` button, which, when clicked, starts rotating the correlogram anticlockwise (in this respect, "+ve" means increasing longitude). This animation continues until the `Stop` button is clicked. The `Rotate -ve` is similar but rotates the correlogram clockwise.

The majority of controls apply to the 3d-plot of the correlogram. This plot is selected using the `3d-correlogram` option of the `Drawing Type` box. This box is further described in Section 6.2 below.



The switches in the `3d-plotting mode` box determine if it is drawn as lines (which is the fastest of the various 3d modes), or polygons filled with white (which is intermediate in speed), or polygons filled using a grayscale representing the value of each point in the correlogram (which is the slowest). The `Line Mode` page (see p 58) controls the number of lines plotted. The `Polygon Mode` page (see p 58) controls the size of the polygons and the colourmap used for the polygons.



The **viewpoint** page determines the position from which the 3d correlogram is viewed. **Latitude** determines the degree to which the correlogram is tilted towards or away from the plane of the monitor screen. **Longitude** determines the degree to which it is turned along the plane of frequency x delay (the **Rotate +ve** and **Rotate -ve** buttons that were mentioned earlier animate the correlogram by rotating it in Longitude). **3d rotation** determines the degree to which it is rotated in the plane of the monitor screen. The values of Latitude, Longitude, and Rotation are all in degrees. **Shear** does not vary the viewpoint; instead it applies a shearing transformation to the correlogram, so that the higher-frequency channels are displaced across from the lower-frequency channels. Values of Shear of between -5 and +5 give good results. The arrow buttons move the correlogram within the Display Window. The circle button moves the correlogram to the center of the Display Window.

The screenshot shows a control panel with several tabs: "Line mode", "Polygon mode", "Weightings", "Freq axis", "Viewpoint", "Size", "Red markers", and "Recovering". The "Viewpoint" tab is selected. It contains four input fields with spinners: "Latitude" (40, range -90..90 degs), "Longitude" (20, range -180..180 degs), "3d rotation" (0, range -90..90 degs), and "Shear" (0, range -5..5). To the right of these fields is a section labeled "Position in window" containing five arrow buttons (up, down, left, right) and a central circle button.

The **size** page determines the overall size of the correlogram and the range of frequency channels and delays that are plotted. The overall size is expressed as a percent of the size of the Display Window. The range are expressed as channel numbers (for the frequency axis) or delay numbers (for the delay axis). The **Max Range** button will reset these values to the full range of the correlogram. The **top** dialog box controls how the vertical scale of the correlogram is plotted. The value of **top** is the value of the correlogram that is plotted at maximum height, so if **top** is large, then the more compressed the correlogram peaks will look, and if **top** is small, then the more expanded the correlogram peaks will look. The value of **top** can also be controlled from the keyboard by pressing 'u' (for up) or 'd' (for down) when the keyboard focus is in the display window.

The screenshot shows the "Size" tab selected in the same control panel. It features two sections: "Overall size (%)" and "Top". Under "Overall size (%)", there are input fields for "Frequency" (65) and "Delay" (65). The "Top" section has an input field with the value 46.78. Below these are two range settings: "Frequency range" from 0 to 18 and "Delay range" from 0 to 140. Each range setting has a "Max range" button next to it.

The **Red Markers** page determines the location of the red lines in the correlogram. These red lines mark the selected frequency or delay that are used for the plots of the central spectrum, recovered spectrum, or delay function (see section 6.2 below). The positions of the markers are controlled by either the dialog boxes in the **Red Markers** page, the arrow buttons at the bottom of the control window, or (when the keyboard focus is in the display window) by the arrow keys on the keyboard. The **Cut Display** dialog boxes will slice the correlogram in two, leaving the actual values above the cut but zeros below the cut. The **Colour Channel** dialog boxes determine if the markers are coloured (in red) or not. The **Draw Zero Line** dialog boxes determine if the markers are also projected onto the plane of frequency x delay; i.e., the plane of zero values in the correlogram.

Selected channel		Selected spectrum	
Freq (Hz)	499	Delay (ms)	0.000
(rounded to 499 Hz)		(rounded to 0.000 ms)	
Cut display	<input type="checkbox"/>	Cut display	<input type="checkbox"/>
Colour channel	<input checked="" type="checkbox"/>	Colour spectrum	<input checked="" type="checkbox"/>
Draw zero line	<input type="checkbox"/>	Draw zero line	<input type="checkbox"/>

The **Recovering** page determines the delay range over which the recovered spectrum is calculated. This range can be either the full internal-delay axis or can be limited to just one period of the center-frequency of the channel (the reasoning here is that most cross-correlation function are essentially periodic, with a period equal to the inverse of the center frequency of the channel). The recovered spectrum itself is described in Section 6.2.

Calculation of recovered spectra	
<input checked="" type="radio"/>	use full delay range
<input type="radio"/>	use +/- 0.5 cycles only

The **Line mode** page determines in more detail the plotting when the correlogram is plotted as lines. The dialog boxes control whether frequency-channel and delay lines (spectra) are drawn, the spacing of the lines, and whether a simple hidden-line algorithm is used (which, at the moment, works on frequency-channel lines only).

Viewpoint Size Red markers Recovering
Line mode Polygon mode Weightings Freq axis

Individual channels **Individual spectra**

Plot ☒ Plot ☒

Spacing 1 Spacing (ms) 0.200

Remove hidden lines ☒

The **Polygon mode** page determines in more detail the plot of polygons. The dialog boxes determine the size of each polygon and the range of the grayscale used if the Grayscale modes are selected. The default values are chosen because they work well on my computer. Note that the hidden-polygon algorithm appears to misplace 1 or 2 polygons; I do not know why.

Viewpoint Size Red markers Recovering
Line mode Polygon mode Weightings Freq axis

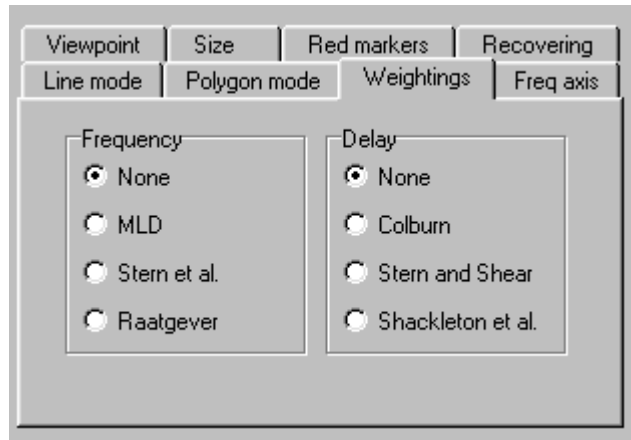
Colourmap **Polygon size**

No. of colours 256 Filters 2

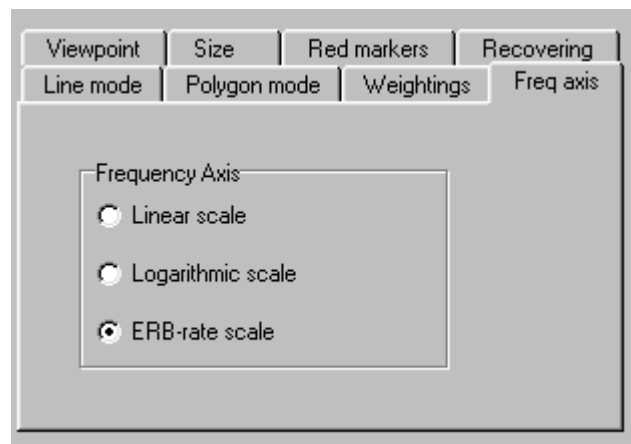
Max colourvalue 255 Delays 2

Min colourvalue 128

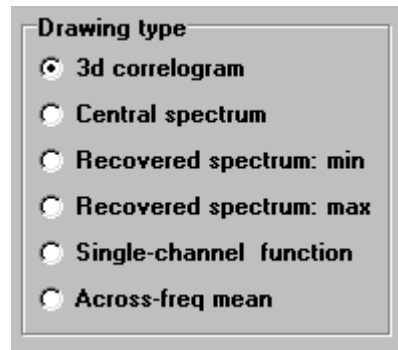
The **weightings** page determines if frequency or delay weighting functions are applied to the correlogram. The frequency functions are: '**MLD**' (a fit to the N0S0-N0S π masking level difference as a function of frequency), '**Stern et al.**' (the frequency-dominance function from Stern et al., 1988), and '**Raatgever**' (the frequency dominance function from Raatgever, 1980). These functions are the same as those described earlier in Section 5.1. The delay functions are: '**Colburn**' (the $p(\tau)$ function from Colburn, 1977), '**Stern and Shear**' (the $p(\tau)$ function from Shear, 1987, and Stern and Shear, 1996), and '**Shackleton et al.**' (the $p(\tau)$ function used by Shackleton et al., 1992). These functions are the same as those described earlier in Section 5.2.



The **Freq axis** page determines how the frequency axis of the correlogram is plotted. The options set the plotted spacing of the frequency channels so that an equal distance on the screen corresponds to equal distances in Hz ('**linear scale**'), or equal distances in log Hz ('**logarithmic scale**'), or equal distances in ERB number ('**ERB-rate scale**'). Note that this page does not change the center frequencies of each channel; they are set at the time of the original call to **mcorrelogram**.



6.2 "Drawing type" dialog box



The **3d correlogram** is the default view and shows the whole correlogram.

The **Central spectrum** is a single slice of the correlogram taken at a constant value of internal delay. Hence it is a plot of correlogram level vs. frequency for constant delay. The position of the selected delay is determined by the **Red Marker** page described above. There is one central spectrum for each value of delay. The central spectrum is incorporated as it was used by Bilsen, Raatgever and their colleagues in their model of dichotic pitches (see *Further Reading: Central Spectra and the Dichotic Pitches*).

The **Single Channel Function** is a single slice of the correlogram taken at a constant value of frequency. Hence it is a plot of correlogram level vs. delay for constant frequency. It too is determined by the **Red Marker** page. There is one such function for each frequency channel.

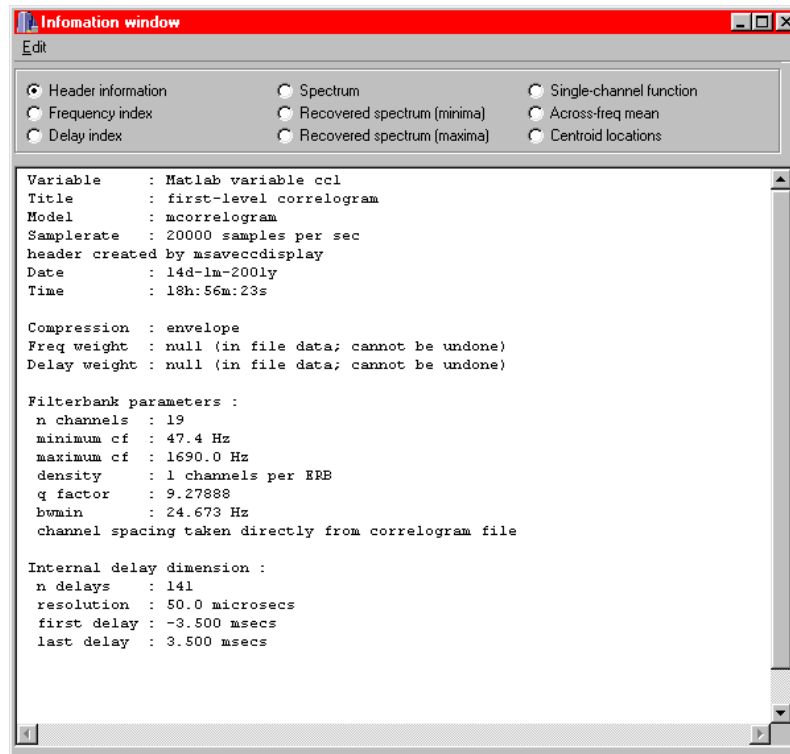
The **Recovered Spectrum: Min** is a plot of the minimum level in each frequency channel versus frequency. There is only one such spectrum for a correlogram. The **Recovered Spectrum: Max** is similar but is a plot of the maximum level not the minimum level. The delay range over which these values are calculated is determined by the **Recovering** page. The recovered spectrum (in particular, the minimum version) is incorporated as it was used by Culling and Summerfield in their model of dichotic pitches (see *Further Reading: Recovered Spectra and the Dichotic Pitches*).

The **Across-Freq Mean** is a plot of the level at each delay averaged across frequency.

6.3 "Info Window"

Clicking the **Info Window** button will bring up a new text window that reports numerical information on the correlogram. The various options within the window determine what information is reported. Note that the values reported by **Centralspectrum** and **Single-Channel Function** will depend on, respectively, which central spectrum or single-channel function is chosen (see the **Red Markers** page). All of the values take account of any frequency or delay weights that are applied.

Basic textual information --- chosen frequency channel, delay and the level of the correlogram at their intersection --- is also reported on the bottom of the Display Window.



6.4 Temporary files

Two temporary files are created. `mcallccdisplay` creates a file called `tempmatlab.bcc`, which contains an ASCII header describing the parameters of the correlogram followed by a binary data structure containing the values of the correlogram as floating-point numbers. This format can be read by `ccdisplay`. `ccdisplay` itself creates a file called `bccdisplay.tmp`, which is a temporary file used in loading the data (it is included in order to be backwardly compatible with an earlier UNIX version of the binaural model, in which the data had to be byte-swapped before loading on a PC). The called `bccdisplay.tmp` file should not be used for storing important data as it is overwritten each time `ccdisplay` is called.

The function `msaveccdisplay` will save a correlogram in a format that can be read by `ccdisplay`. For example, typing this will save a previously-made correlogram `cc1` in a file called 'correlogram1.bcc' and label the correlogram as '#1':

```
>> msaveccdisplay(cc1, 'correlogram1.bcc', '#1', 1)
```

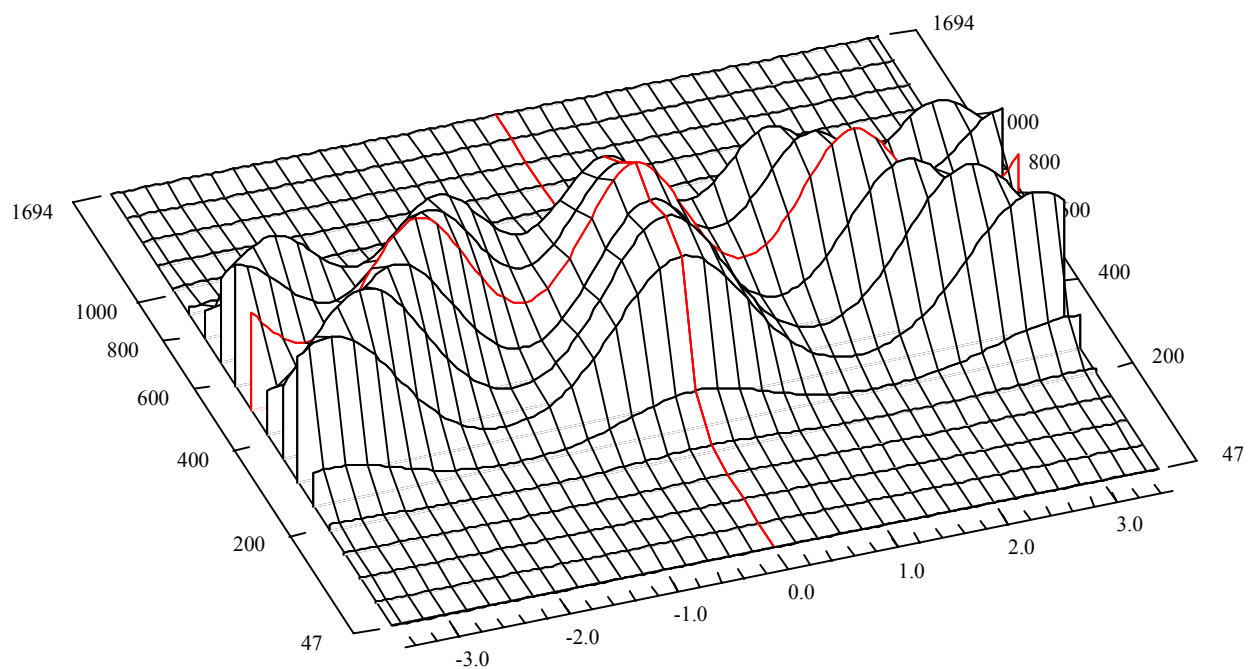
Saved files can be loaded into `ccdisplay` using `open` from the **File** menu of the Control Window.

6.5 Copying/pasting a correlogram into Powerpoint

Pressing **copy** from the **Edit** menu of the Control Window will copy the current view of the correlogram as a Windows Metafile, suitable for pasting into Microsoft Powerpoint 97. Note that the image should not be pasted using **Paste** in the **Edit** menu of Powerpoint, as then the image is pasted as a very small picture thus:



Instead, it should be pasted using **Paste Special** in the **Edit** menu of Powerpoint and specifying the **Picture (Enhanced Metafile)** option, thus:

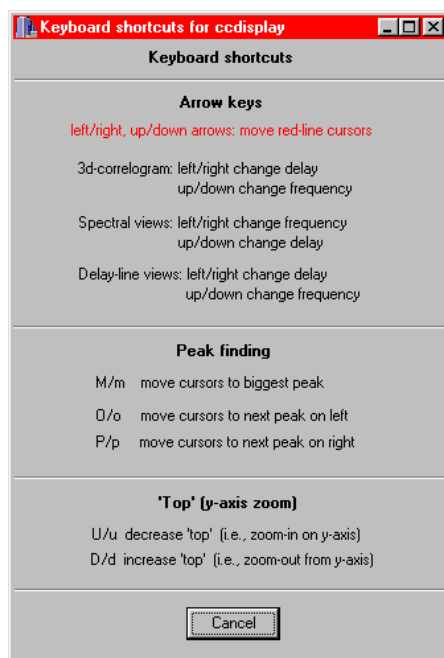


6.6 Keyboard shortcuts

Various keyboard shortcuts are available for controlling some aspects of `ccdisplay`. These can:

- **u, d** Control the value of 'top' (see **size** page of the Control Window).
- **arrow keys** Move the location of the frequency-channel and delay markers.
- **m, o, p** Find peaks in the various two-dimensional views.

All the shortcuts are listed in the **Keyboard Shortcuts** part of the **Help** menu:



Appendix 1

Frequency Weighting

Appendix 1.1 Raatgever's function

Raatgever's (1980; Eq. IV.1, p. 64) weighting function is:

$$w(f) = \frac{1}{e^{((f_{Hz}/300)-2)^2}} \quad f_{Hz} < 600$$

$$w(f) = \frac{1}{e^{((f_{Hz}/600)-1)^2}} \quad f_{Hz} \geq 600$$

where f_{Hz} is the frequency in Hz.

Appendix 1.2 Stern et al.'s function

Stern et al.'s (1988, $q(f)$ function, p. 160) weighting function is:

$$w(f) = 10 \left(\frac{-b_1 f_{Hz} - b_2 f_{Hz}^2 - b_3 f_{Hz}^3}{10} \right) \quad f_{Hz} \leq 1200$$

$$w(f) = 10 \left(\frac{-b_1 1200 - b_2 1200^2 - b_3 1200^3}{10} \right) \quad f_{Hz} > 1200$$

where f_{Hz} is the frequency in Hz and the constants b_1 , b_2 , and b_3 are equal to:

$$b_1 = -0.0938272$$

$$b_2 = 0.000112586$$

$$b_3 = -0.0000000399154$$

Appendix 1.3 The ‘MLD’ function

Akeroyd and Summerfield's (1999) fit to the variation with frequency of the N0S0-N0S π masking level difference is:

$$w(f) = (1-r)(1 + (f_{Hz} - f_0)p)e^{((f_0 - f_{Hz})p)} + r$$

where f_{Hz} is the frequency in Hz, and the values of p , r and f_0 are equal to

$$p = 0.0046$$

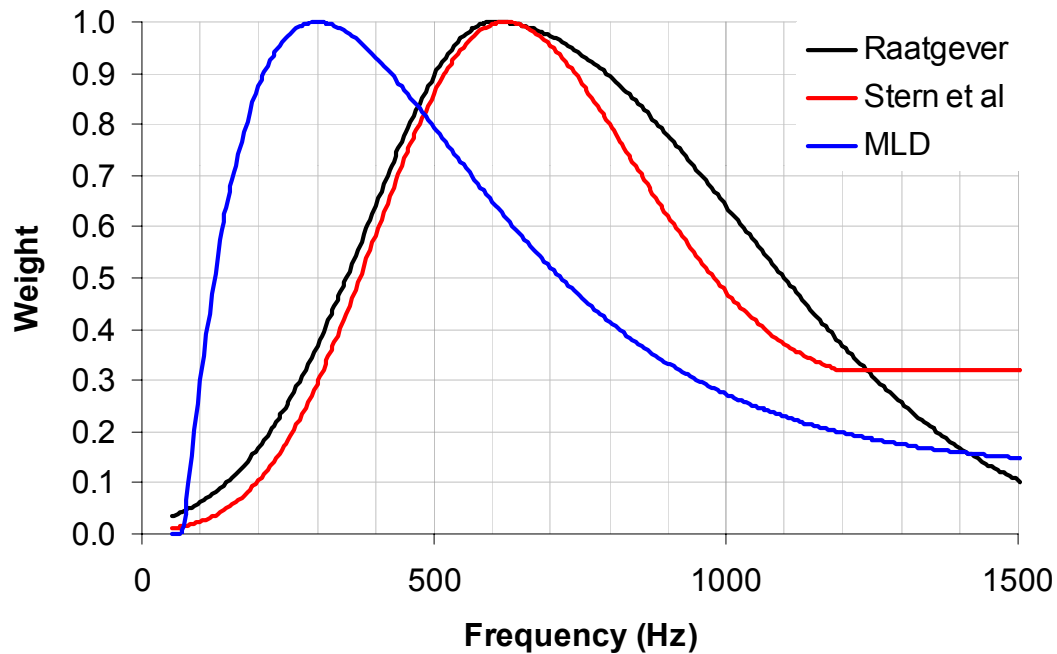
$$r = 10^{(-9/10)} = 0.125893$$

$$f_0 = 300$$

Appendix 1.4 Comparative Illustrations

These functions are illustrated in the next picture. The location of the peak of each function is:

Raatgever	600 Hz
Stern et al.	623 Hz
MLD	300 Hz



Appendix 2

Delay Weighting

Appendix 2.1 Colburn's function

Colburn's (1977, Eq. 3) delay-weighting function is:

$$\begin{aligned}
 p(\tau_{\mu s}) &= C & |\tau_{\mu s}| &\leq 150 \\
 p(\tau_{\mu s}) &= Ce^{\left(\frac{150-|\tau_{\mu s}|}{600}\right)} & 150 < |\tau_{\mu s}| &\leq 2200 \\
 p(\tau_{\mu s}) &= 0.033Ce^{\left(\frac{2200-|\tau_{\mu s}|}{2300}\right)} & |\tau_{\mu s}| &> 2200
 \end{aligned}$$

where $\tau_{\mu s}$ is the internal delay in μsecs and C is a constant. The value of C is set so that the area under the delay-weighting function is equal to 1.0, i.e:

$$\int_{-\infty}^{\infty} p(\tau_{\mu s}) d\tau_{\mu s} = 1$$

Appendix 2.2 Shear's function

Shear's (1987, Eqs. 4.14, 4.15; Stern and Shear, 1996, Eqs. 6, 7) delay-weighting function is:

$$\begin{aligned}
 p(\tau_{\mu s}) &= \frac{C}{|200|} \left(e^{-2\pi k_L |200|} - e^{-2\pi k_H |200|} \right) & \tau_{\mu s} &\leq 200 \\
 p(\tau_{\mu s}) &= \frac{C}{|\tau_{\mu s}|} \left(e^{-2\pi k_L |\tau_{\mu s}|} - e^{-2\pi k_H |\tau_{\mu s}|} \right) & \tau_{\mu s} &> 200
 \end{aligned}$$

where $\tau_{\mu s}$ is the internal delay in μsecs and the parameters k_L and k_H are given by:

$$\begin{aligned}
 k_L &= \frac{0.1}{1000000} f_{Hz}^{1.1} & f_{Hz} \leq 1200 \\
 &= \frac{0.1}{1000000} 1200^{1.1} & f_{Hz} > 1200
 \end{aligned}$$

$$k_H = \frac{3000}{1000000}$$

where f_{Hz} is the frequency in Hz. Note that the value of k_L is dependent on frequency but the value of k_H is fixed. Again the constant C is chosen so that the area is equal to 1.0; because the value k_L changes with frequency the value of C will also.

Appendix 2.3 Shackleton et al.'s function

Shackleton et al.'s (1992) delay-weighting function is:

$$p(\tau_{\mu s}) = \frac{1}{\sigma_{\mu s} \sqrt{2\pi}} e^{-\left(\frac{\tau_{\mu s}^2}{2\sigma_{\mu s}^2}\right)}$$

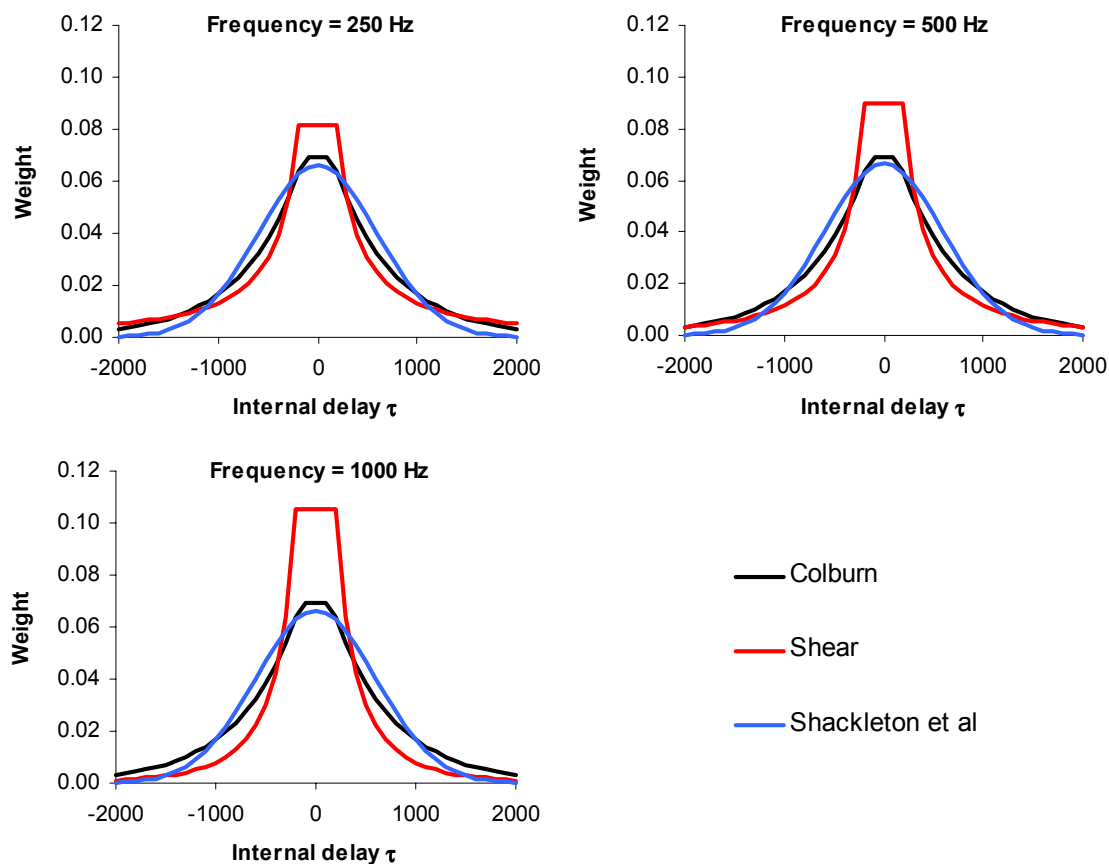
where the standard deviation $\sigma_{\mu s}$ is

$$\sigma_{\mu s} = 600$$

As this function is a Gaussian of standard deviation $\sigma_{\mu s}$ its area is equal to 1.0.

Appendix 2.4 Comparative Illustrations

Colburn's and Shackleton et al.'s functions are the same at all frequencies, whereas Shear's function narrows as the frequency is increased. The narrowing can be seen in the additional emphasis applied to delays near zero in the pictures below.



Appendix 3

The ERB function

Appendix 3.1 The ERB at a given center frequency

Glasberg and Moore (1990) gave this function for describing the relationship between the equivalent rectangular bandwidth (ERB) of the auditory filter and its center frequency:

$$ERB = 24.7(4.37F_{kHz} + 1)$$

where F_{kHz} is the center frequency in kHz. The function is illustrated at the end of this Appendix. In the gammatone-filterbank software incorporated in the Auditory Image Model (see *Further Reading: The gammatone filterbank and the Auditory Image Model*) a two-parameter version of this equation was used:

$$ERB = b_{\min} \left(\frac{f_{Hz}}{qb_{\min}} + 1 \right)$$

where f_{Hz} is the center frequency in Hz, q is the q-factor of the filter and b_{\min} is a parameter representing the minimum bandwidth of the filter. These values of q and b_{\min} give Glasberg and Moore's equation:

$$\begin{aligned} b_{\min} &= 24.673 \\ q &= \frac{1000}{4.37b_{\min}} \\ &= 9.2789 \end{aligned}$$

These values are returned by the function `mstandarderbparameters`. For example:

```
>> [q bmin] = mstandarderbparameters;
>> q

q =
    9.2789e+000

>> bmin

bmin =
    2.4673e+001
```

The function **merb** will return the ERB at a frequency. For example, to obtain the ERB at 500-Hz using the standard values of q and b_{min} , type:

```
>> erb = merb(500, 9.2789, 24.673, 1);
cf = 500.0 Hz: ERB = 78.56 Hz

>> erb

erb =
    7.8559e+001
```

Appendix 3.2 Expressing a frequency in ERB number

In auditory modeling it is convenient to express the center frequency of a filter using a frequency-related scale based on units of the ERB. The function which transforms center frequency from a value in Hz to a value in ERB number can be obtained by integrating the inverse of the ERB function (Glasberg and Moore, 1990), giving:

$$\begin{aligned} f_{erb} &= \frac{1000 \log_e(10)}{24.7 \times 4.37} \log_{10} \left(\frac{4.7 f_{Hz}}{1000} + 1 \right) \\ &= 21.3 \log_{10} \left(\frac{4.7 f_{Hz}}{1000} + 1 \right) \\ &= q \log_e \left(\frac{f_{Hz}}{q b_{min}} + 1 \right) \end{aligned}$$

where f_{Hz} is the center frequency in Hz and f_{erb} is the center frequency in ERB numbers (see also Hartmann, 1997, p. 251, Eq 10.29). This function is illustrated at the end of this Appendix.

The function **mhztoerb** will convert a frequency from Hz to ERB number. For example, to convert 500 Hz into ERB numbers using the standard values of q and b_{min} , type:

```
>> f_erb = mhztoerb(500, 9.2789, 24.673, 1);
500.0 Hz -> 10.746 ERBs

>> f_erb

f_erb =
    1.0746e+001
```

The function **merbtohz** will convert back. For example:

```
>> f_hz = merbtohz(10.746, 9.2789, 24.673, 1);
10.7 ERBs -> 499.982 Hz
```

```
>> f_hz
```

```
f_hz =  
4.9998e+002
```

This value is not quite exactly 500 Hz because of rounding errors in the value of 10.746 (a more accurate value of 10.7462331 ERB number gives 500.000 Hz).

Appendix 3.3 A useful filterbank

The gammatone filterbank used in the binaural toolbox spaces filters equally in ERB number. In my recent modeling (e.g., Trahiotis et al., 2001) I have used a filterbank ranging from 47.4 to 1694 Hz:

47 Hz = 1.75 ERB number = 9 ERB numbers below 500 Hz.
500 Hz = 10.75 ERB number
1694 Hz = 19.75 ERB number = 9 ERB numbers above 500 Hz.

Such a filterbank therefore extends by the same range above and below 500 Hz. In the toolbox this filterbank should be specified as 47.4 – 1690 Hz, as the highest filter is placed just *above* the nominal upper frequency. For example:

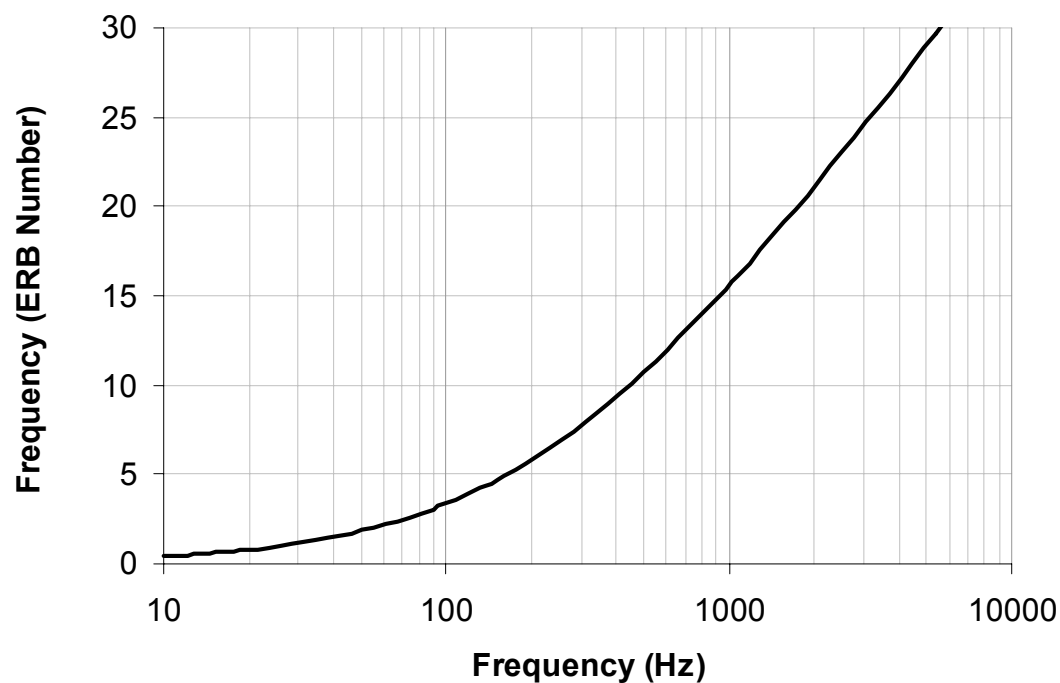
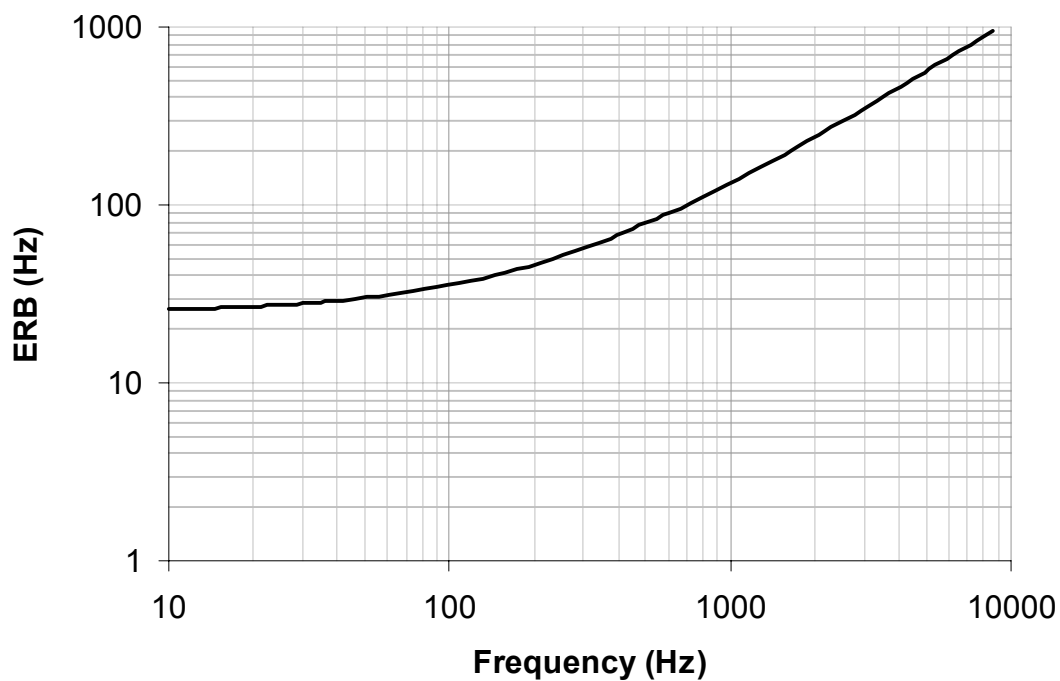
```
>> ccl = mcorrelogram(47.4, 1690, 1, -3500, 3500, 'envelope', 'cp', wavel, 2);
```

This example gives a 19-channel filterbank with these center frequencies and bandwidths. Note that the frequency in Hz and ERB number is stored in the *freqaxiserb* and *freqaxisHz* fields of the ‘correlogram’ structure.

#	Freq (Hz)	Freq (ERB number)	ERB (Hz)
1	47.40	1.75	29.8
2	78.85	2.75	33.2
3	113.87	3.75	36.9
4	152.88	4.75	41.1
5	196.33	5.75	45.8
6	244.72	6.75	51.0
7	298.62	7.75	56.9
8	358.65	8.75	63.3
9	425.51	9.75	70.5
10	499.99	10.75	78.6
11	582.93	11.75	87.5
12	675.32	12.75	97.5
13	778.22	13.75	108.5
14	892.82	14.75	120.9
15	1020.47	15.75	134.7
16	1162.65	16.75	150.0
17	1321.00	17.75	167.0
18	1497.37	18.75	186.0
19	1693.82	19.75	207.2

Appendix 3.4 Illustrations

The first picture shows the relationship between a frequency and the ERB at that frequency. The second picture shows the relationship between a frequency in units of Hz and in units of ERB number.



References

- Akeroyd MA and Summerfield AQ (1999). "A fully temporal account of the perception of dichotic pitches," *Br. J. Audiol.*, 33(2), 106-107.
- Colburn HS (1977). "Theory of binaural interaction based on auditory-nerve data. II. Detection of tones in noise," *J. Acoust. Soc. Am.*, 61, 525-533.
- Bernstein LR and Trahiotis C (1996a). "On the use of the normalized correlation as an index of interaural envelope correlation", *J. Acoust. Soc. AM.*, 100, 1754-1763.
- Bernstein LR and Trahiotis C (1996b). "The normalized correlation: Accounting for binaural detection across center frequency", *J. Acoust. Soc. AM.*, 100, 3774-3784.
- Bernstein LR, van de Par S and Trahiotis C (1999). "The normalized correlation: Accounting for N0Sp thresholds obtained with Gaussian and "low-noise" masking noise", *J. Acoust. Soc. Am.*, 106, 870-876.
- Glasberg BR and Moore BCJ (1990). "Derivation of auditory filter shapes from notched-noise data," *Hearing Research*, 47, 103-138.
- Hartmann, WM (1997). *Signals, Sound, and Sensation* (American Institute of Physics, Woodbury, New York).
- Meddis R (1986). "Simulation of mechanical to neural transduction in the auditory receptor," *J. Acoust. Soc. Am.* 79, 702-711.
- Meddis R (1988). "Simulation of auditory-neural transduction: Further studies," *J. Acoust. Soc. Am.* 83, 1056-1063.
- Meddis R, Hewitt M, and Shackleton TM (1990). "Implementation details of a computational model of the inner-haircell/auditory-nerve synapse," *J. Acoust. Soc. Am.* 87, 1813-1816.
- Raatgever J (1980). *On the binaural processing of stimuli with different interaural phase relations*, (Doctoral dissertation, Delft University of Technology, The Netherlands).
- Shackleton TM, Meddis R and Hewitt MJ (1992). "Across frequency integration in a model of lateralization," *J. Acoust. Soc. Am.*, 91, 2276-2279.
- Shear GD (1987). "Modeling the dependence of auditory lateralization on frequency and bandwidth," (Masters thesis, Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh).
- Stern RM, Zeiberg AS, and Trahiotis C (1988). "Lateralization of complex binaural stimuli: A weighted image model," *J. Acoust. Soc. Am.*, 84, 156-165 (erratum: *J. Acoust. Soc. Am.*, 90, 2202).
- Stern RM and Shear GD (1996). "Lateralization and detection of low-frequency binaural stimuli: Effects of distribution of internal delay," *J. Acoust. Soc. Am.*, 100, 2278-2288.
- Trahiotis C, Bernstein LR, and Akeroyd MA (2001). "Manipulating the patterns of interaural-cross correlation affects listeners' sensitivity to changes in interaural delay," *J. Acoust. Soc. Am.*, 109 (1), 321-330.

Further Reading

All of these are listed in date order.

Books and review chapters on lateralization

Durlach NI and Colburn HS (1978). "Binaural phenomena", in *Handbook of Perception: Volume 4*, edited by EC Carterette and MP Friedman (Academic, New York).

Colburn HS and Durlach NI (1978). "Models of binaural interaction", in *Handbook of Perception: Volume 4*, edited by EC Carterette and MP Friedman (Academic, New York).

Stern RM and Trahiotis C (1995). "Models of binaural interaction," in *Hearing*, edited by BCJ Moore (Academic, San Diego).

Colburn HS (1995). "Computational models of binaural processing," in *Auditory Computation*, edited by H Hawkins and T McMullin (Springer-Verlag, New York).

Stern RM and Trahiotis C (1997). "Models of binaural perception," in *Binaural and Spatial Hearing in Real and Virtual Environments*, edited by RH Gilkey and TR Anderson (Lawrence Erlbaum Associates, Mahwah, New Jersey).

Blauert J (1997). *Spatial Hearing: The Psychophysics of Human Sound Localization* (MIT, Cambridge).

Models of lateralization and binaural processing

Stern RM and Colburn HS (1978). "Theory of binaural interaction based on auditory nerve data. IV. A model for subjective lateral position," *J. Acoust. Soc. Am.*, 84, 127-140.

Shear GD (1987). "Modeling the dependence of auditory lateralization on frequency and bandwidth," (Masters thesis, Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh).

Stern RM and Colburn HS (1985). "Lateral position based models of interaural discrimination," *J. Acoust. Soc. Am.*, 77, 753-755.

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

Lindemann W (1986b). "Extension of a binaural cross-correlation model by means of contralateral inhibition. II. The law of the first wave front," *J. Acoust. Soc. Am.*, 80, 1623-1630.

Stern RM, Zeiberg AS, and Trahiotis C (1988). "Lateralization of complex binaural stimuli: A weighted image model," *J. Acoust. Soc. Am.*, 84, 156-165 (erratum: *J. Acoust. Soc. Am.*, 90, 2202).

Trahiotis C and Stern RM (1989). "Lateralization of bands of noise: Effects of bandwidth and differences of interaural time and intensity," *J. Acoust. Soc. Am.*, 86, 1285-1293.

- Stern RM, Zeppenfeld T, and Shear GD (1991). "Lateralization of rectangularly-modulated noise: Explanations for counterintuitive reversals," *J. Acoust. Soc. Am.*, 90, 1901-1907.
- Shackleton TM, Meddis R, and Hewitt MJ (1992). "Across frequency integration in a model of lateralization," *J. Acoust. Soc. Am.*, 91, 2276-2279.
- Stern RM and Trahiotis C (1992). "The role of consistency of interaural timing over frequency in binaural lateralization," in *Auditory Physiology and Perception*, edited by Y Cazals, K Horner and L Demany (Pergamon Press, Oxford).
- Gaik W (1993). "Combined evaluation of interaural time and intensity differences: Psychoacoustic results and computer modeling," *J. Acoust. Soc. Am.*, 94, 98-110.
- Trahiotis C and Stern RM (1994). "Across frequency interaction in lateralization of complex binaural stimuli," *J. Acoust. Soc. Am.*, 96, 3804-3806.
- Stern RM and Shear GD (1996). "Lateralization and detection of low-frequency binaural stimuli: Effects of distribution of internal delay," *J. Acoust. Soc. Am.*, 100, 2278-2288.
- Stern RM and Trahiotis C (1998). "Binaural mechanisms that emphasize consistent interaural timing information over frequency," in *Psychophysical and Physiological Advances in Hearing*, edited by AR Palmer, A Rees, AQ Summerfield, and R Meddis (Whurr, London).
- Shackleton TM (1998). "Comment on Stern RM and Trahiotis C: 'Binaural mechanisms that emphasize consistent interaural timing information over frequency'," in *Psychophysical and Physiological Advances in Hearing*, edited by AR Palmer, A Rees, AQ Summerfield, and R Meddis (Whurr, London).
- Akeroyd MA and Summerfield AQ (2000). "The lateralization of simple dichotic pitches," *J. Acoust. Soc. Am.*, 108, 316-334.
- Akeroyd MA and Summerfield AQ (2001). "A computational auditory model of the lateralization of the Huggins pitch," in *Computational Models of Auditory Function*, edited by S Greenberg and M Slaney (to be published by IOS Press).
- Trahiotis C, Bernstein LR, and Akeroyd MA (2001). "Manipulating the patterns of interaural-cross correlation affects listeners' sensitivity to changes in interaural delay," *J. Acoust. Soc. Am.*, 109 (1), 321-330.
- Hartung K and Trahiotis C (2001). "Peripheral auditory processing and the precedence effect," manuscript submitted for publication.

Recovered spectra and the dichotic pitches

- Culling JF, Summerfield AQ, and Marshall DH (1998a). "Dichotic pitches as illusions of binaural unmasking. I. Huggins' pitch and the 'binaural edge pitch'," *J. Acoust. Soc. Am.*, 103, 3509-3526.
- Culling JF, Marshall DH, and Summerfield AQ (1998b). "Dichotic pitches as illusions of binaural unmasking. II. The Fourcin pitch and the dichotic repetition pitch," *J. Acoust. Soc. Am.*, 103, 3527-3539.

Culling JF (2000). "Dichotic pitches as illusions of binaural unmasking. III. The existence region of the Fourcin pitch," *J. Acoust. Soc. Am.*, 107, 2201-2208.

Akeroyd MA and Summerfield AQ (2000). "The lateralization of simple dichotic pitches," *J. Acoust. Soc. Am.*, 108, 316-334

Central spectra and the dichotic pitches

Bilsen FA and Goldstein JL (1974). "Pitch of dichotically delayed noise and its possible spectral basis," *J. Acoust. Soc. Am.*, 55, 292-296.

Bilsen FA (1977). "Pitch of noise signals: Evidence for a 'central spectrum'," *J. Acoust. Soc. Am.*, 61, 150-161.

Raatgever J and Bilsen FA (1977). "Lateralization and dichotic pitch as a result of spectral pattern recognition," in *Psychophysics and Physiology of Hearing*, edited by EF Evans and JP Wilson (Academic, London).

Raatgever J (1980). *On the binaural processing of stimuli with different interaural phase relations*, (Doctoral dissertation, Delft University of Technology, The Netherlands).

Raatgever J and Bilsen FA (1986). "A central spectrum theory of binaural processing. Evidence from dichotic pitch," *J. Acoust. Soc. Am.*, 80, 429-441.

Frijns JHM, Raatgever J and Bilsen FA (1986). "A central spectrum theory of binaural processing. The binaural edge pitch revisited," *J. Acoust. Soc. Am.*, 80, 442-451.

Bilsen FA (1995). "What do dichotic pitch phenomena tell us about binaural hearing," in *Advances in Hearing Research*, edited by GA Manley, GM Klump, C Koppl, and H Fastl (World Scientific, London)

Bilsen FA, van der Meulen AP, and Raatgever J (1998). "Salience and JND of pitch for dichotic noise stimuli with scattered harmonics: grouping and the central spectrum theory," in *Psychophysical and Physiological Advances in Hearing*, edited by AR Palmer, A Rees, AQ Summerfield and R Meddis (Whurr, London).

Bilsen FA and Raatgever J (2000). "On the dichotic pitch of simultaneously presented interaurally delayed white noises. Implications for binaural theory," *J. Acoust. Soc. Am.*, 108, 272-284.

Bilsen FA (2001). "The case of the missing central spectra," in *Physiological and Psychophysical Bases of Auditory Function*, edited by AJM Houtsma, A Kohlrausch, VF Prijs and R Schoonhoven (to be published by Shaker Publishing, Maastricht, The Netherlands).

The gammatone filterbank and the Auditory Image Model.

Holdsworth JW, Nimmo-Smith I, Patterson RD, and Rice P (1988). Annex C of *Spiral VOS Final Report: Part A: The Auditory Filterbank* (MRC Applied Psychology Unit, Cambridge, UK, Report #2341).

Patterson RD, Holdsworth JW, Nimmo-Smith I, and Rice P (1988). *Spiral VOS Final Report: Part A: The Auditory Filterbank* (MRC Applied Psychology Unit, Cambridge, UK, Report #2341).

Patterson RD, Robinson K, Holdsworth J, McKeown D, Zhang C, and Allerhand M (1992). “Complex sounds and auditory images”, in *Auditory Physiology and Perception*, edited by Y Cazals, K Horner and L Demany (Pergamon Press, Oxford).

Patterson RD and Holdsworth J (1994). “A functional model of neural activity patterns and auditory images”, in *Advances in Speech, Hearing and Language Processing*, edited by WA Ainsworth (JAI, London).

Patterson RD (1994). “The sound of a sinusoid: Spectral models”, *J. Acoust. Soc. Am.*, 96, 1409-1418.

Patterson RD (1994). “The sound of a sinusoid: Time-interval models”, *J. Acoust. Soc. Am.*, 96, 1419-1428.

Patterson RD, Allerhand MH and Giguere C (1995). “Time-domain modeling of peripheral auditory processing: A modular architecture and a software platform”, *J. Acoust. Soc. Am.*, 98, 1890-1894.

Two other computational auditory models

O'Mard LP, Hewitt MJ, and Meddis R (1997). *LUTEar 2.0.9 Manual* (University Of Essex: <http://www.essex.ac.uk/psychology/hearinglab/lutear/manual/Manual.html>). This report describes the Core Routines Library of the LUTEar auditory model. The model has recently been combined with Patterson's Auditory Image Model and renamed the Development System for Auditory Models (“DSAM”). A beta version of DSAM can be obtained from: <ftp://ftp.essex.ac.uk/pub/omard/dsam/>

Slaney M (1998). *Auditory Toolbox: Version 2* (Technical Report #1998-010, Interval Research Corporation, <http://rvl4.ecn.purdue.edu/~malcolm/interval/1998-010>). The gammatone functions in the present toolbox are taken from here.