DETECTION TUTORIAL

Exploring automatic detection with Ishmael

Oregon State University/NOAA PMEL
Hosted by:
The University of Rhode Island
The Engineering Computer Center, College of Engineering

Detection of Marine Mammal Sounds: A Passive Acoustic Training Workshop
June 17-21, 2018
Sponsored by the Living Marine Resources program, U.S. Navy

Development Team

Cooperative Institute for Marine Resources Studies (CIMRS), Oregon State University, and NOAA Pacific Marine Environmental Laboratory

David K. Mellinger	**David.Mellinger@oregonstate.edu**
Sharon L. Nieukirk	**Sharon.Nieukirk@oregonstate.edu**
Curtis Lending	**Curtis.Lending@oregonstate.edu**
Sara Heimlich	**Sara.Heimlich@oregonstate.edu**

Bio-Waves, Inc.

Elizabeth Ferguson	**eferguson@bio-waves.net**
Julie Oswald	**julie.oswald@bio-waves.net**
Michael Oswald	**mike.oswald@bio-waves.net**
Tom Norris	**thomas.f.norris@bio-waves.net**

San Diego State University

Chris Marsh	**chmarsh76@gmail.com**

Ishmael and ROCCA have been principally supported by the Living Marine Resources (LMR) program, U.S. Navy, and the Office of Naval Research (ONR), U.S. Navy

DETECTION TUTORIAL

Exploring automatic detection with Ishmael

READ THIS FIRST!

- The exercises below use the **Ishmael** software package. Install it on your machine if necessary (you can get it for free at http://www.bioacoustics.us/ishmael.html), then launch it.

- The sound files used below are in a folder, **soundfile_examples**. You can get it at the same website.

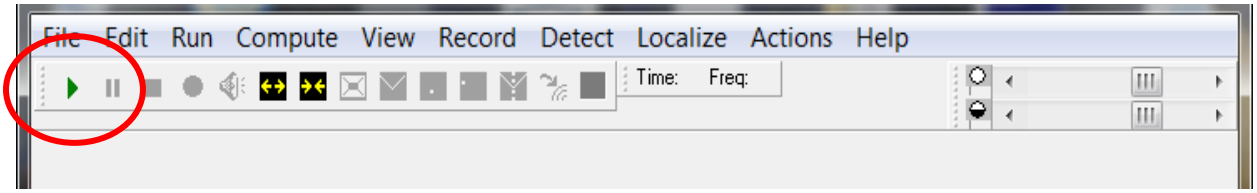
- A lot of the exercises below say "load the file **Ishmael Factory Settings.ipf** to reset your Ishmael settings." This is so any settings you have in Ishmael left over from previous exercise(s) won't interfere with this exercise. IPF stands for Ishmael Preferences File, and .ipf files are how Ishmael stores all your settings. You can load and save your settings for different analysis tasks you're using Ishmael for. The **Ishmael Factory Settings.ipf** file is in the **soundfile_examples** folder; to load it in Ishmael, either drag it from the folder and drop it onto the main Ishmael window, or do **File->Load settings**, find and choose **Ishmael Factory Settings.ipf**, and click OK.

Table of Contents

Basics of sound analysis in Ishmael

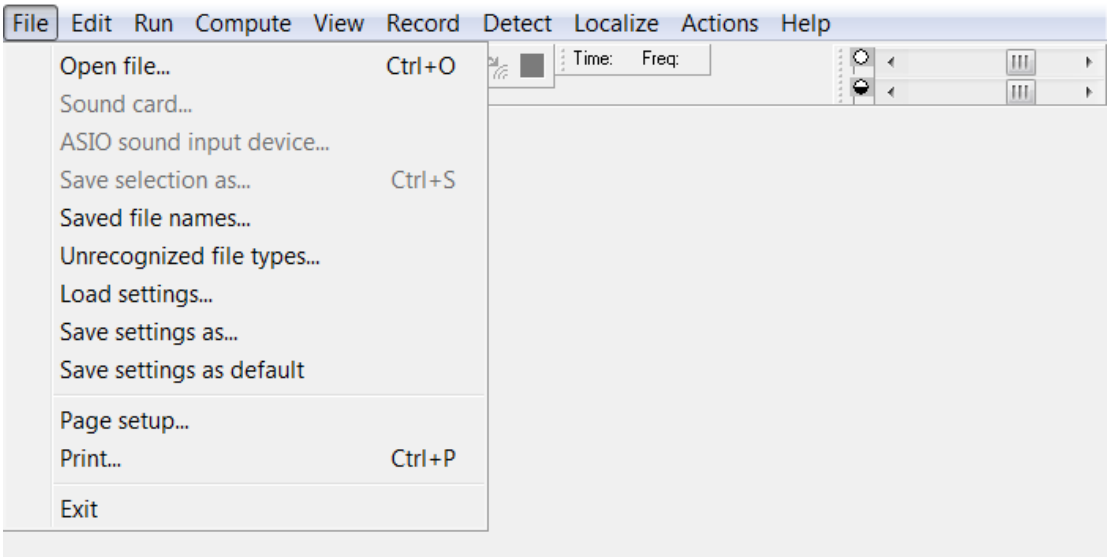
Viewing sound files and making spectrograms

The exercises below require you to adjust some parameters, after which you'll need to run Ishmael to display and process your sound data: click "**Run**" on the menu bar or the **Run** button (the green arrow under the menu bar).



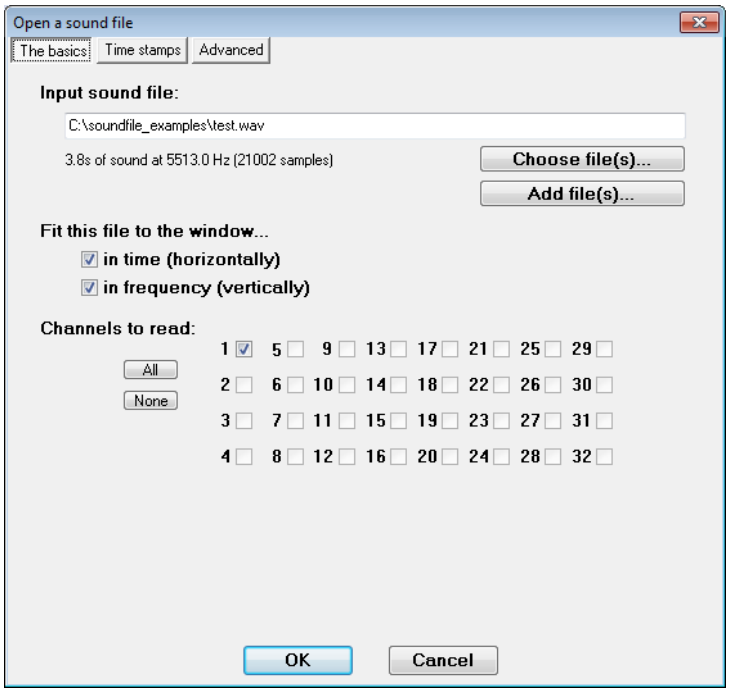
Open a sound file

If you or someone else has been using Ishmael, load **Ishmael Factory Settings.ipf** to reset your Ishmael settings. (See the section "Read This First!" above to do this.) Using **File→Open file**, open the sound file **test.wav**.



Note: You can open a file via the **File→Open**, or by dragging and dropping a file into the Ishmael window.

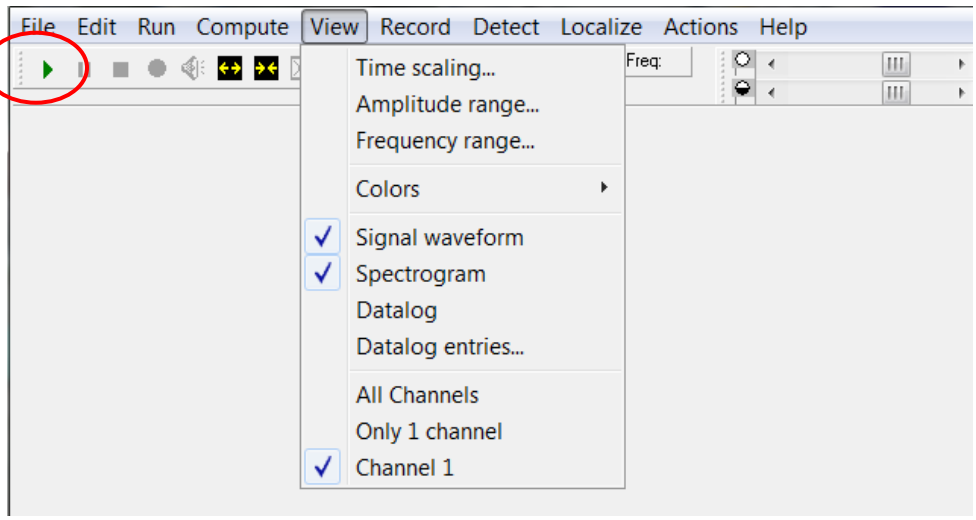
On "**The basics**" tab, you have the option of fitting the sound file to the display in time and frequency ("**Fit this file to the window…**."). Fitting in time is good for relatively short files like the ones in our examples, but if your file is long, the display is so compressed that you can't see any details of the sound in the spectrogram.



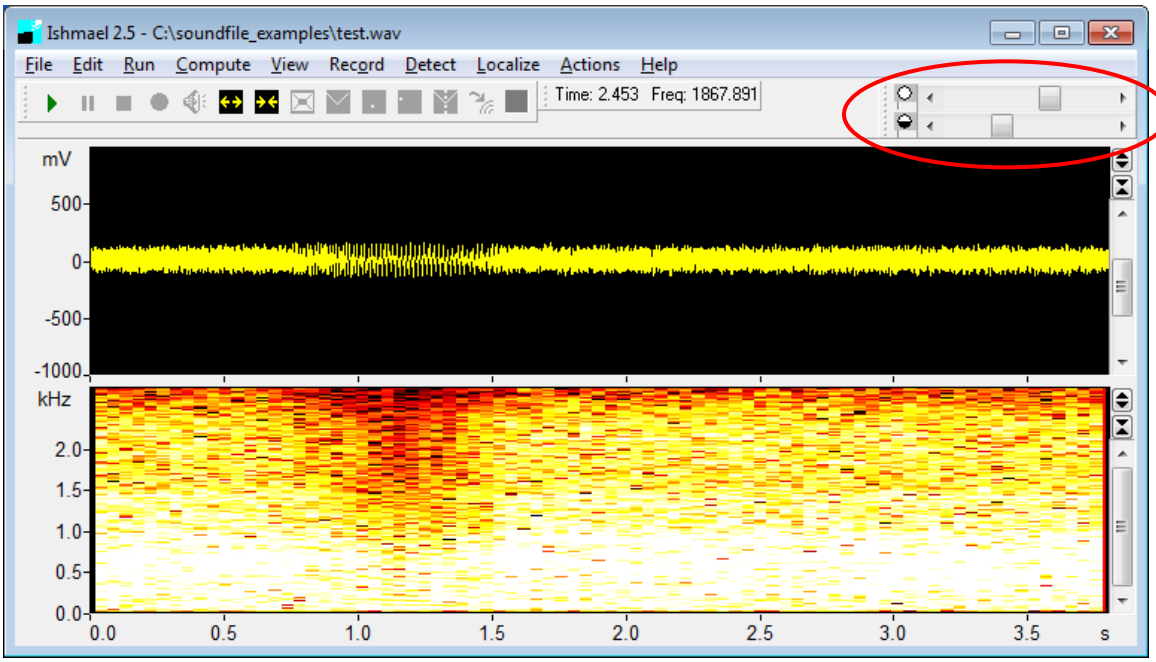
Display a spectrogram:

Look at **View→Spectrogram**; if **Spectrogram** is not already checked, select it and click the green **Run** button. There are two other ways to perform **Run** : selecting **Run** from the menu, or, easiest of all, pressing the space bar on your keyboard.

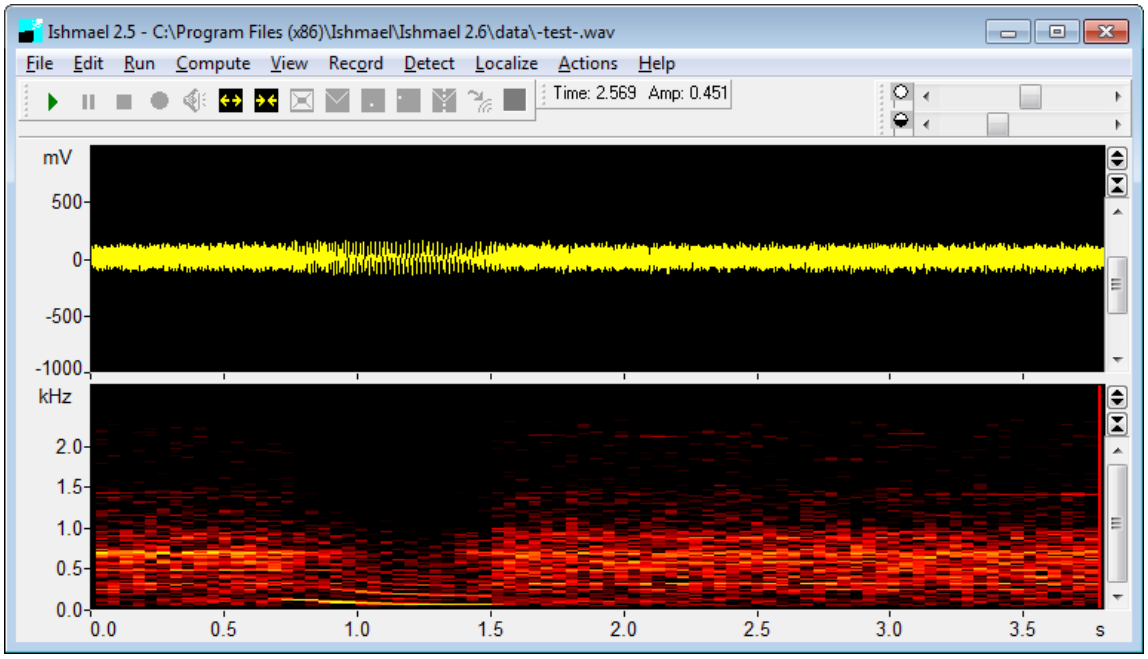
Select **View->Signal waveform**, if it's not already checked, to see the time series (the sound signal) as well.



The file **test.wav** should look something like this:

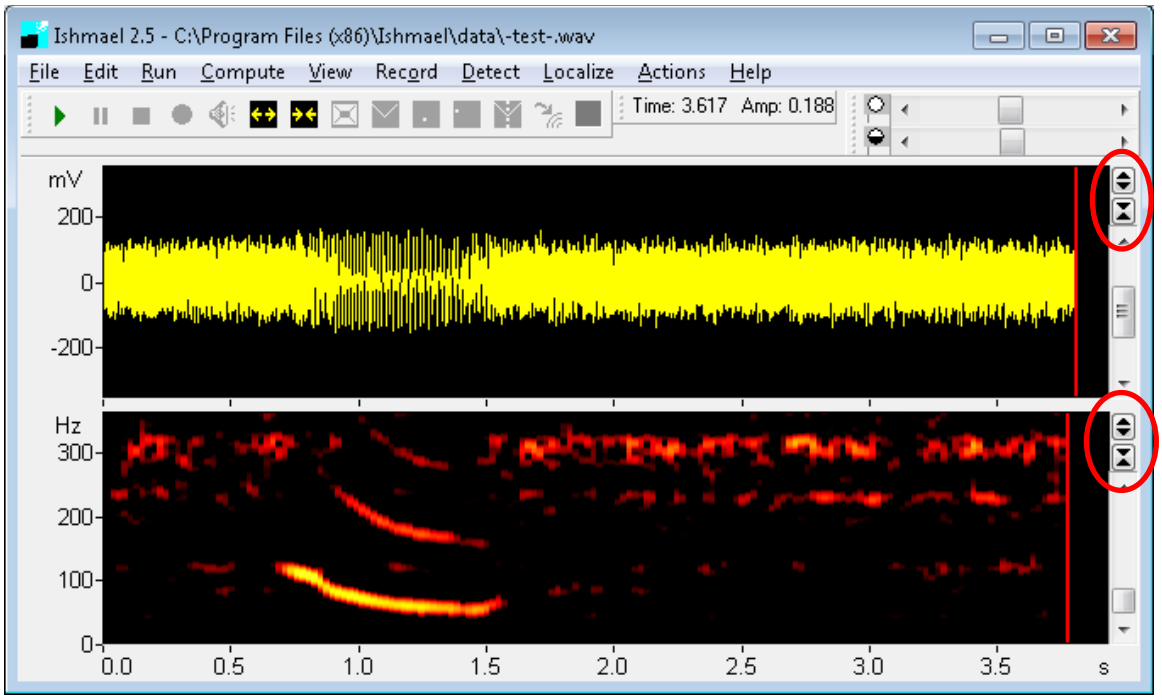


Adjust the spectrogram display using the brightness and contrast controls at the upper right of the main window. The top scrollbar adjusts the brightness and the bottom one adjusts the contrast. Then, click the green **Run** button to re-display the sound. With some exceptions, ***you need to click the green run button (or Run->Run on the menu) whenever you adjust something in Ishmael***, You should get a spectrogram like this:

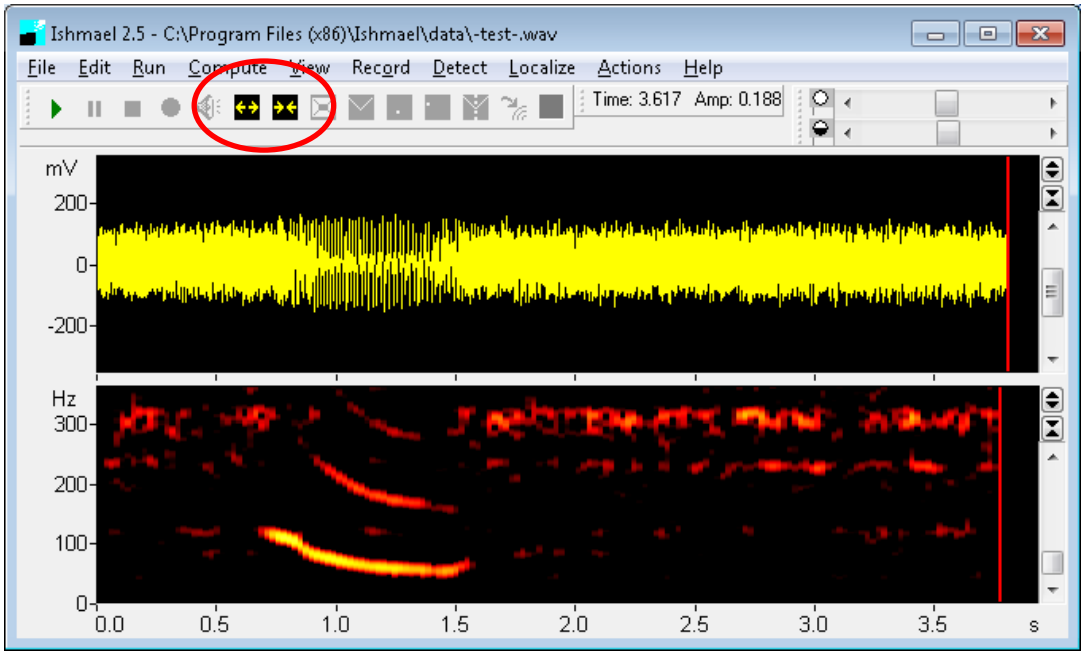


Note: You can only scroll forward in a file in Ishmael. Long files will scroll automatically unless you pause scrolling via hitting the space bar, the pause button on the ribbon bar, or by activating the "pause at end of screen" option.

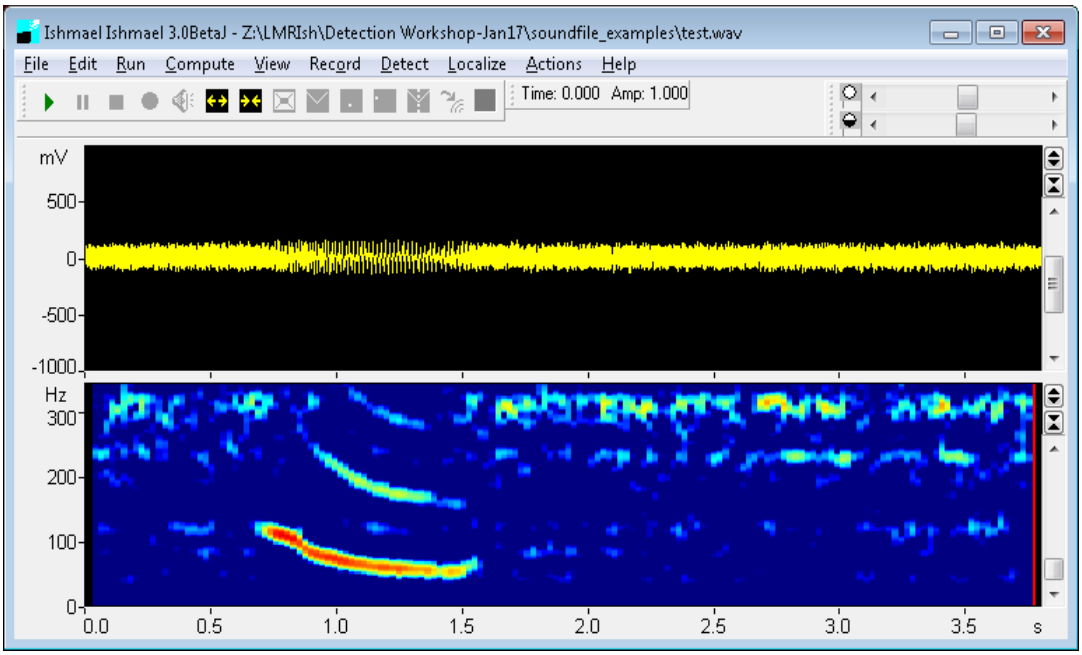
To change the vertical scaling in both the spectrogram and signal waveform, use the vertical shrink and stretch buttons on the right, near the vertical scroll bar. You can change the frequency scaling independently for the spectrogram or signal waveform.



You can now clearly see the downsweeps in this sound file. The horizontal shrink and stretch buttons in the main button bar also change the time scaling in both the spectrogram and signal waveform.



Time and frequency scaling in the spectrogram, and amplitude scaling in the signal waveform, can also be adjusted numerically, using **View->Time scaling** and **View->Frequency range** and **View->Amplitude range** respectively**.** You can also choose a different color pallette for your spectrogram via the **View->Colors** option.



Note: Whenever you open a new file, you usually need to adjust these things – the spectrogram frame size, brightness and contrast, and possibly the time or frequency scaling – to see the spectrogram clearly. Keep this in mind whenever you open a new file..

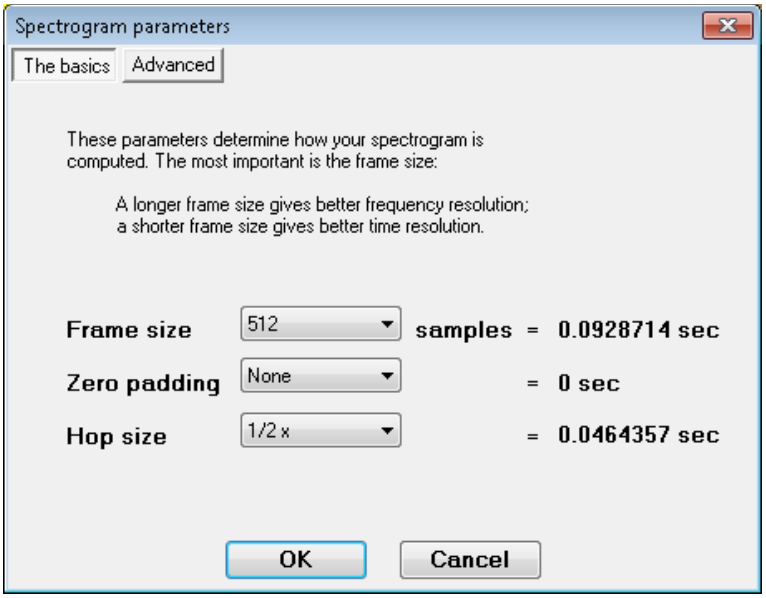
Spectrogram computation

We've just been adjusting how the spectrogram *looks*, but we can also change how it is *computed*, which affects both how it looks and also how it will behave when we get to later steps, like detection.

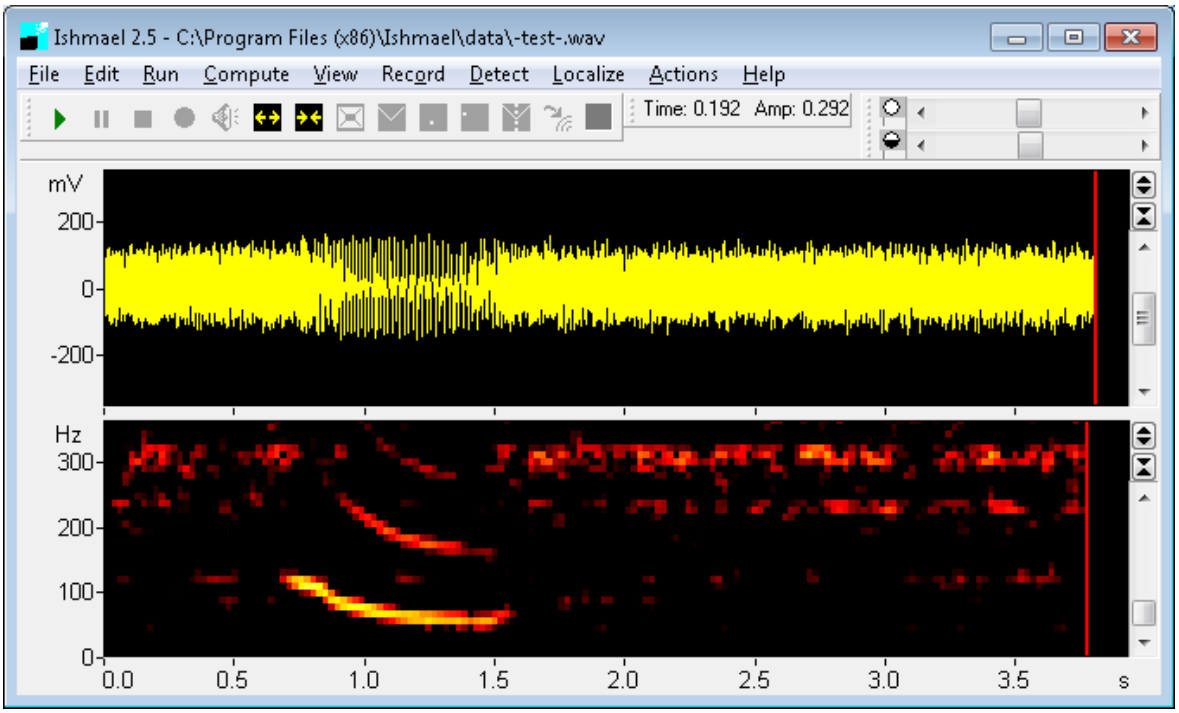
Parameters for the spectrogram computation can be adjusted using **Compute→Spectrogram parameters**. These parameters determine the time and frequency resolution of the spectrogram.

The most important of these parameters is the **Frame size**, which controls how much detail is visible in your sound in both time and frequency – the time and frequency resolutions. These resolutions are inversely related, so that finer time detail will give you coarser frequency detail, and vice versa. In general, a larger frame size provides better frequency resolution but worse time resolution, and vice versa for a smaller frame size.

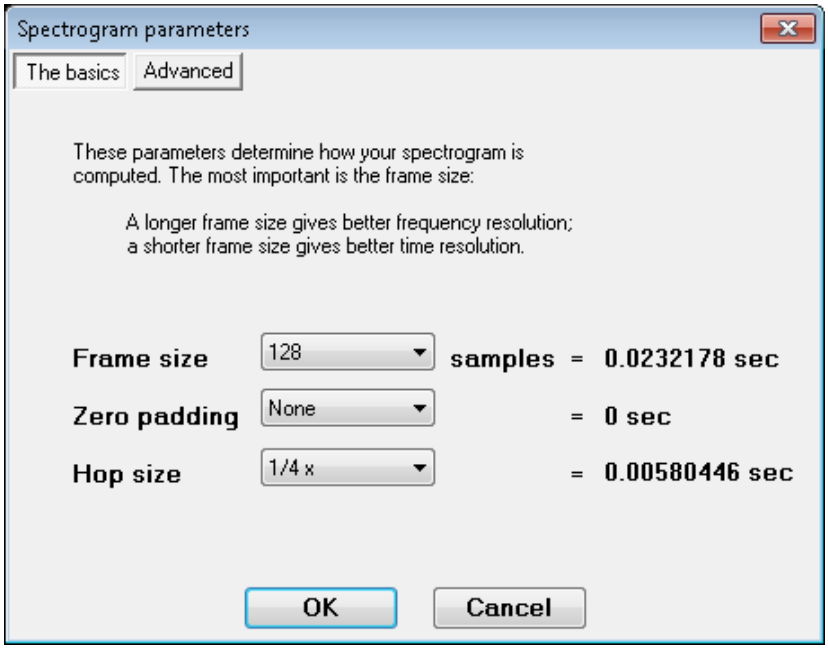
For example, these parameters…



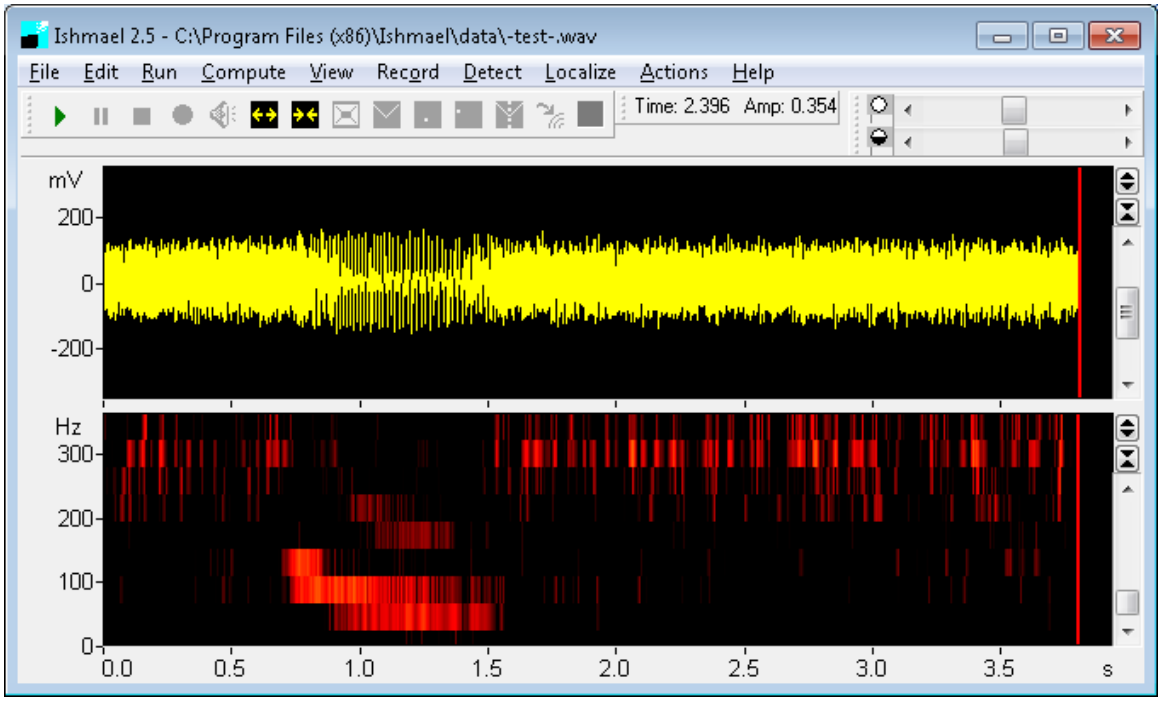
…create a spectrogram that looks like this:



If we shorten the frame size to 128 and remove the zero padding…

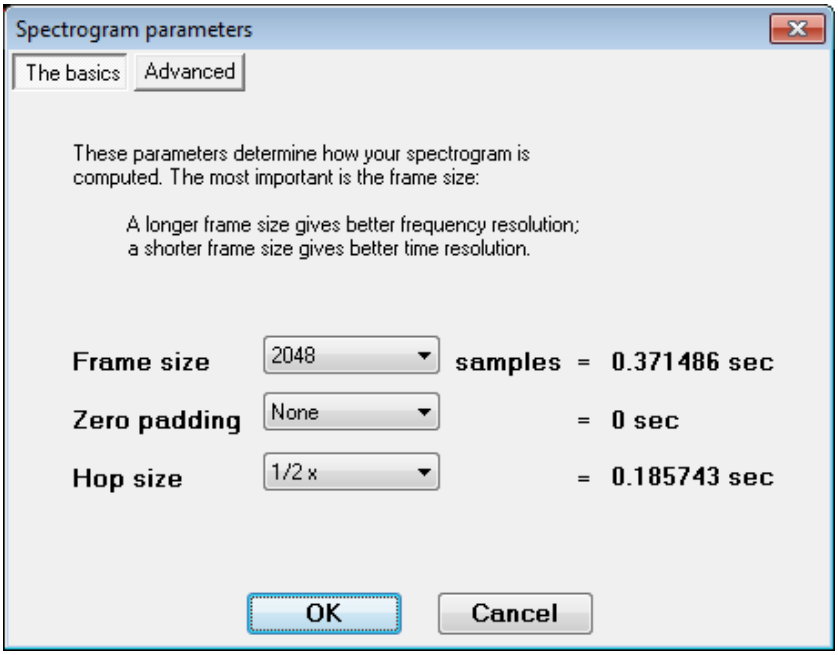


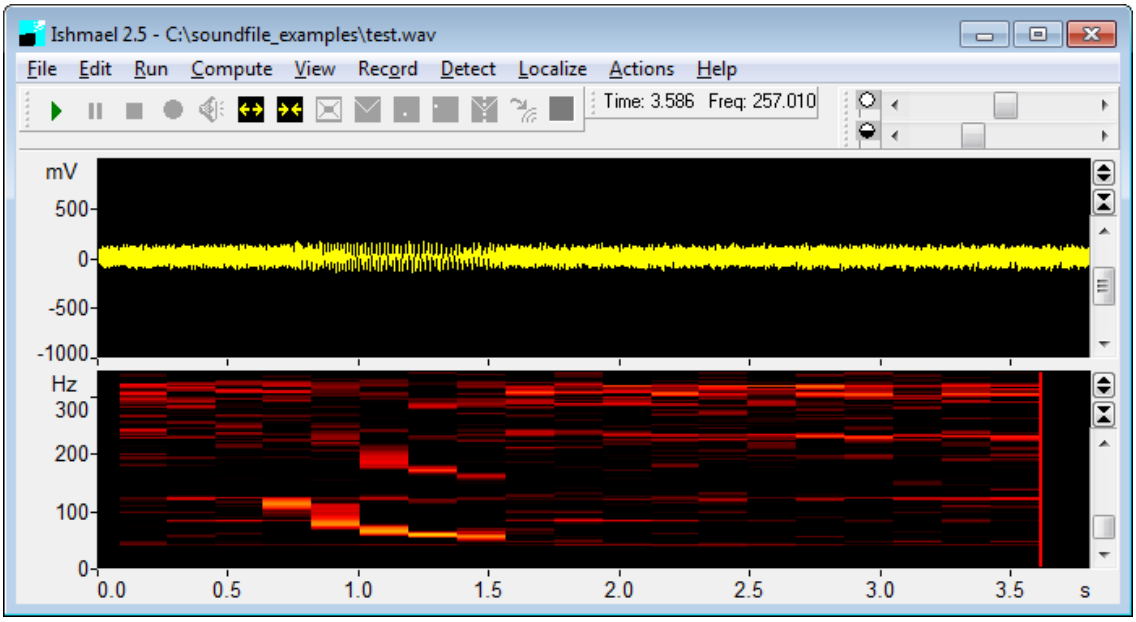
…then the spectrogram gets coarser frequency resolution (fewer horizontal stripes) but very fine time resolution (more vertical stripes):



Note: The time to compute a spectrogram increases as hop size decreases, so choose your hop size carefully!

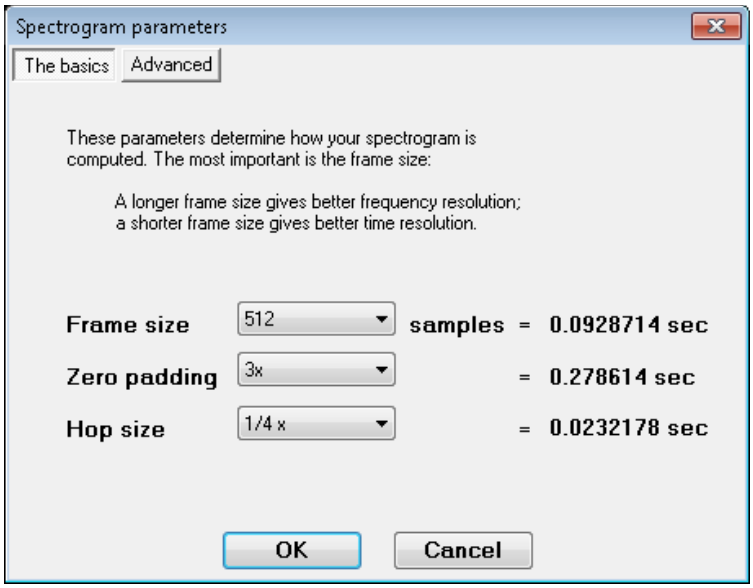
At the other extreme, using a very long frame size of 2048 samples will smear out **test.wav** in frequency, resulting in many horizontal stripes but very few vertical ones:



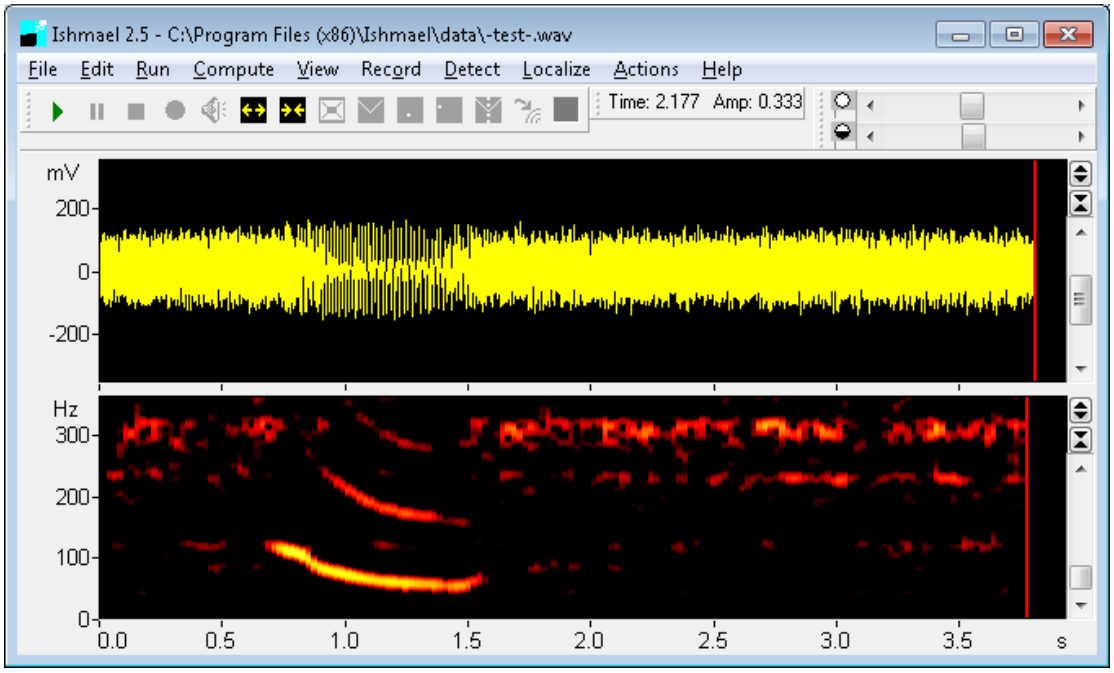


Neither of these extremes works all that well; the happy medium for the frame size in this file is around 512 samples.

Changing other parameters will also change how your spectrogram appears. Adding zero-padding and changing to a shorter hop size will add more cells to your spectrogram, making it look less chunky. For instance, if we go back to a frame size of 512, pad the data by 3x, and change the hop size to ¼…



…we get a spectrogram that looks smoother, like this:



Generally speaking, for a given call type, choose a frame size that makes the shape of your calls apparent, with acceptable resolution in both time and frequency. Judge this just by inspecting your call in the spectrogram; are parts of it smeared together vertically (in frequency)? If so, use a longer frame size to get better frequency resolution. If they're smeared together horizontally (in time), use a shorter frame size.

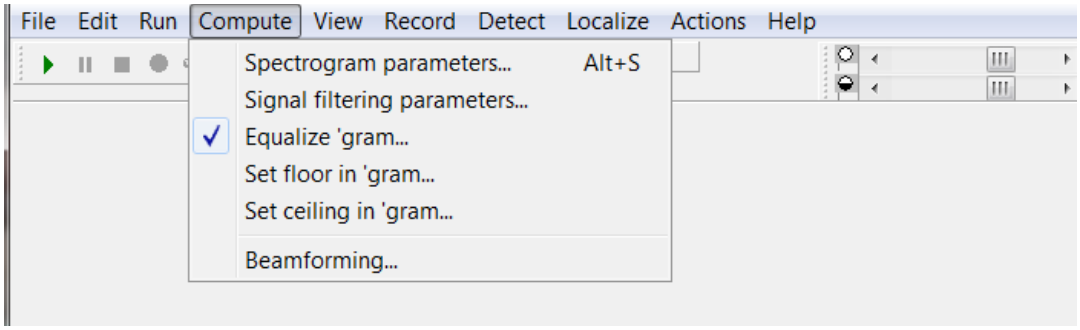
As mentioned above, zero-padding and the hop size can be used to smooth out the appearance of your spectrogram. The downside to using more zero-padding, or a shorter hop size, is that calculating and displaying the spectrogram takes more computation time and more memory. These are worries for long sound files and/or large data sets, so if you don't have either of these conditions, you can use more padding and a short hop size without worries. For zero-padding, 1x or 3x works pretty well, as does 1/4x or 1/8x for hop size.

To sum up:

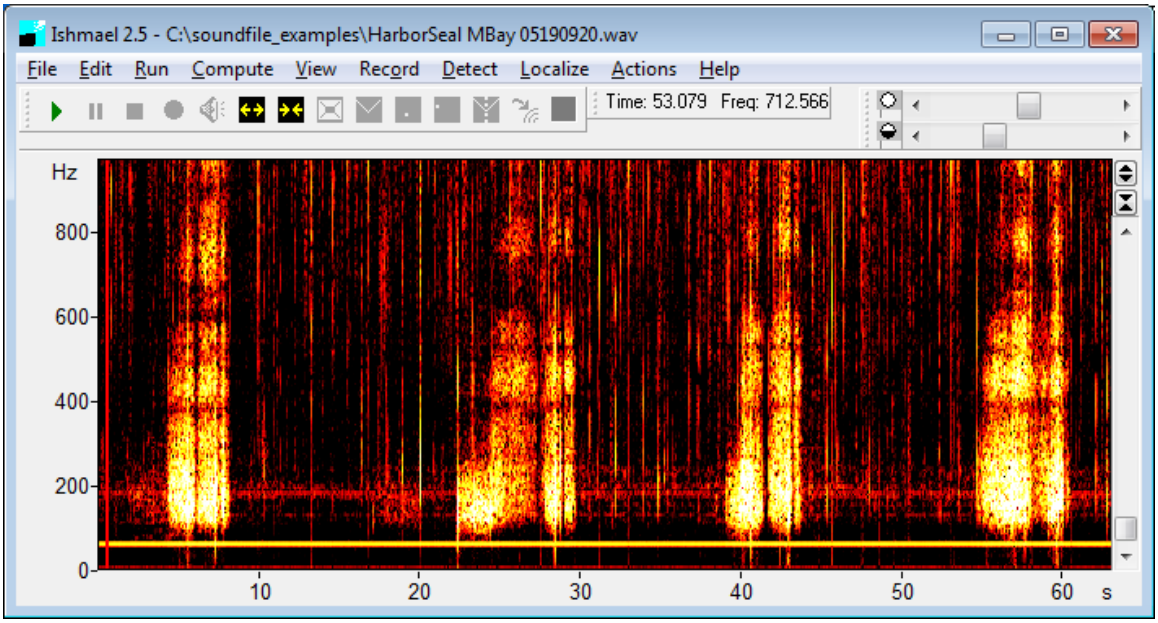
- *The first time you look at a new call type, choose an appropriate frame size.* Try several sizes and click the green **Run** button to see each one. Choose a frame size that gives a reasonably good view of the calls, with acceptable resolution in both time and frequency.
- If you have small sound files or a small amount of data, you can bump up the zero-padding (say to 3x) and decrease the hop size (say to 1/8x) to get smoother spectrograms.
- *Adjust the brightness and contrast* until the background is dark and the calls are visible.
- *Adjust the time and frequency scaling* as you desire.

Equalization / Normalization

Often it's useful to reduce long-duration background noise, such as boat noise and wind noise, before applying a detection method. You can do this in Ishmael with **Compute→Equalize gram**.

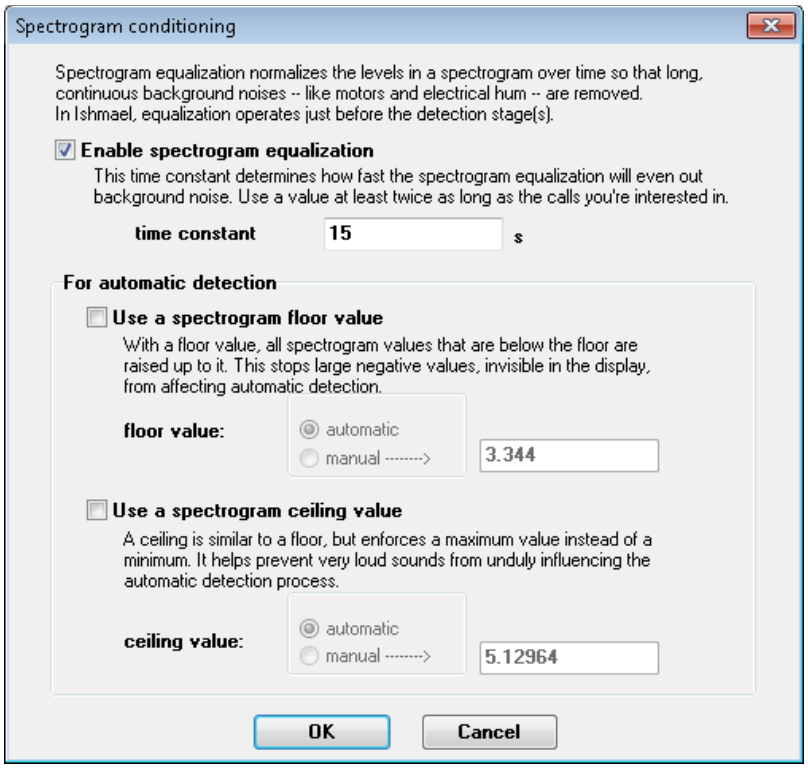


Open the file **HarborSeal MBay 05190920.wav** . Make a spectrogram with a frame size of 8192 samples:

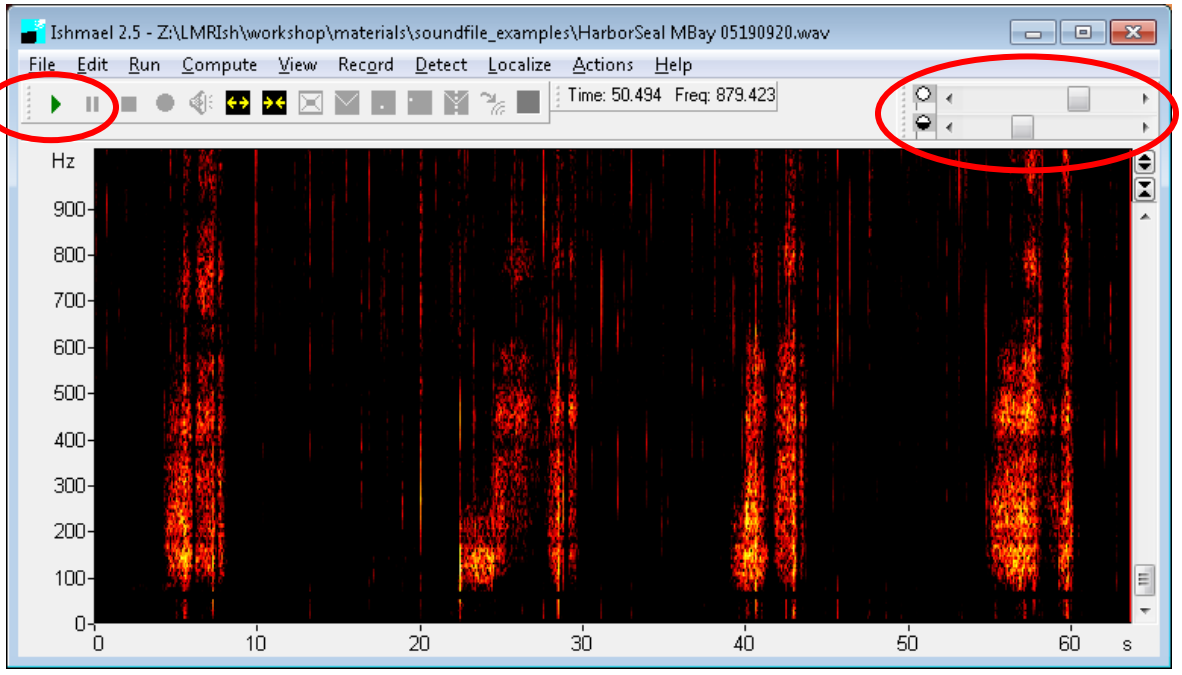


Note the annoying, continuous sound at about 60 Hz, and another at about 180 Hz.

Equalization can be used to eliminate these from the spectrogram: this removes it *only* from the spectrogram, not from the underlying sound signal. We choose a time constant for the equalization that will remove this 60 Hz noise; a good rule of thumb is to choose a value that is 3-5 times as long as the call type(s) you're interested in, so you don't remove the sound of interest. These harbor seal calls are ~5 s long; in this case a 15 s time constant works well.

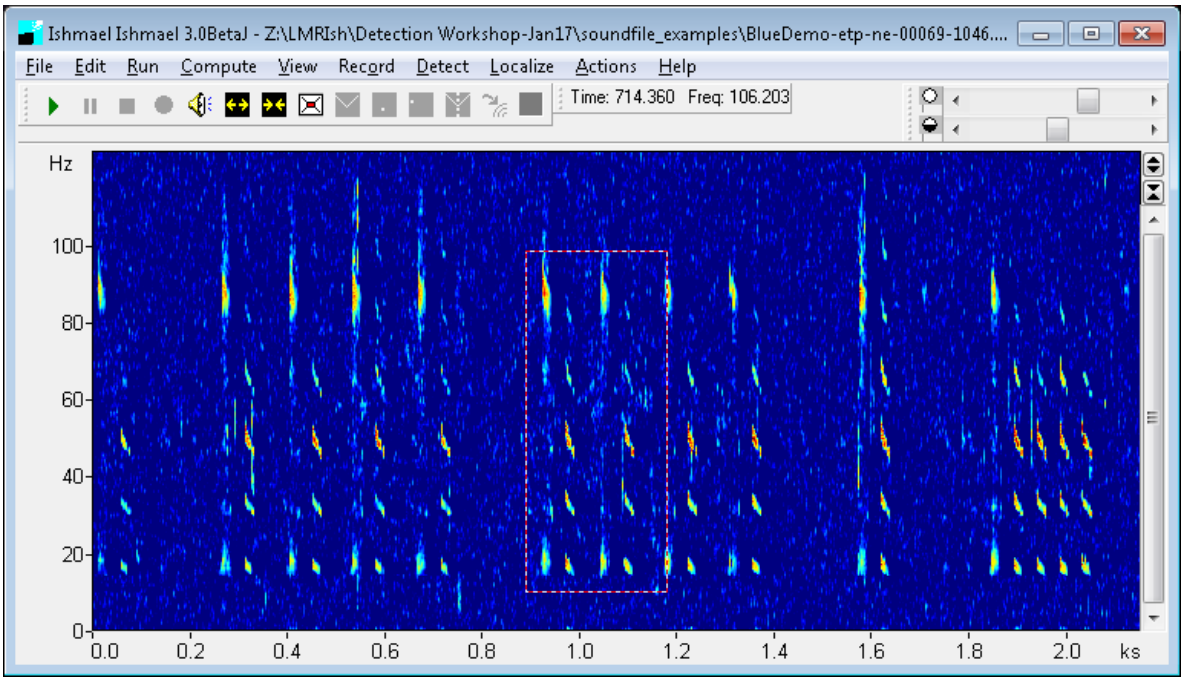


After turning on equalization, you'll need to re-adjust the brightness and contrast using the scrollbars at the upper right, and click the green **Run** button. If the sound that you are trying to remove is variable in length, just do your best to choose a time constant that works and does not remove the target sound. The file is not permanently changed – you can turn equalization on and off and try a few different time constants. One important note: this time constant is applied to all of the data, and may reduce the amplitude of your calls. Also, if you are dealing with a transient sound that comes and goes, you should apply equalization to ALL files so that detector results or other analysis are comparable.



Playing back a sound

Ishmael can play back sounds of interest in your acoustic data. To try this, load the file **BlueDemo-etp-ne-00069-1046.wav**, and adjust the parameters so that you can see the series of blue whale A-B-A-B calls clearly. Hold down the left mouse button and drag the mouse to draw a box around the sound. A dotted box will appear on the spectrogram and/or the signal waveform, and the speaker button () will become active on the menu button bar. Click on the speaker button, and the sound will be played.



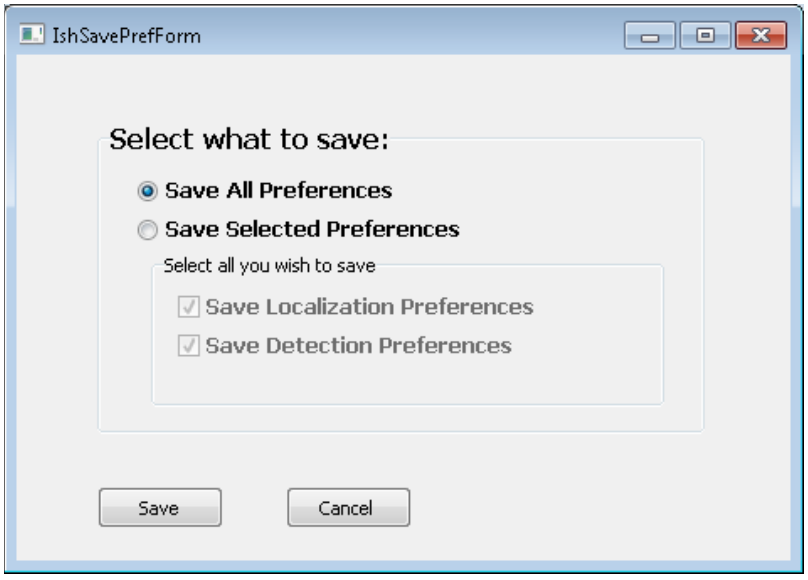
To change the playback sound speed click on the **Run → Playback options** and choose your desired speed, speaker type, etc. Try changing the speed to 10X to hear the details of the blue whale calls. Note that Ishmael will play back all frequencies of the sound, not just those in the box.

Loading and saving Ishmael preferences files

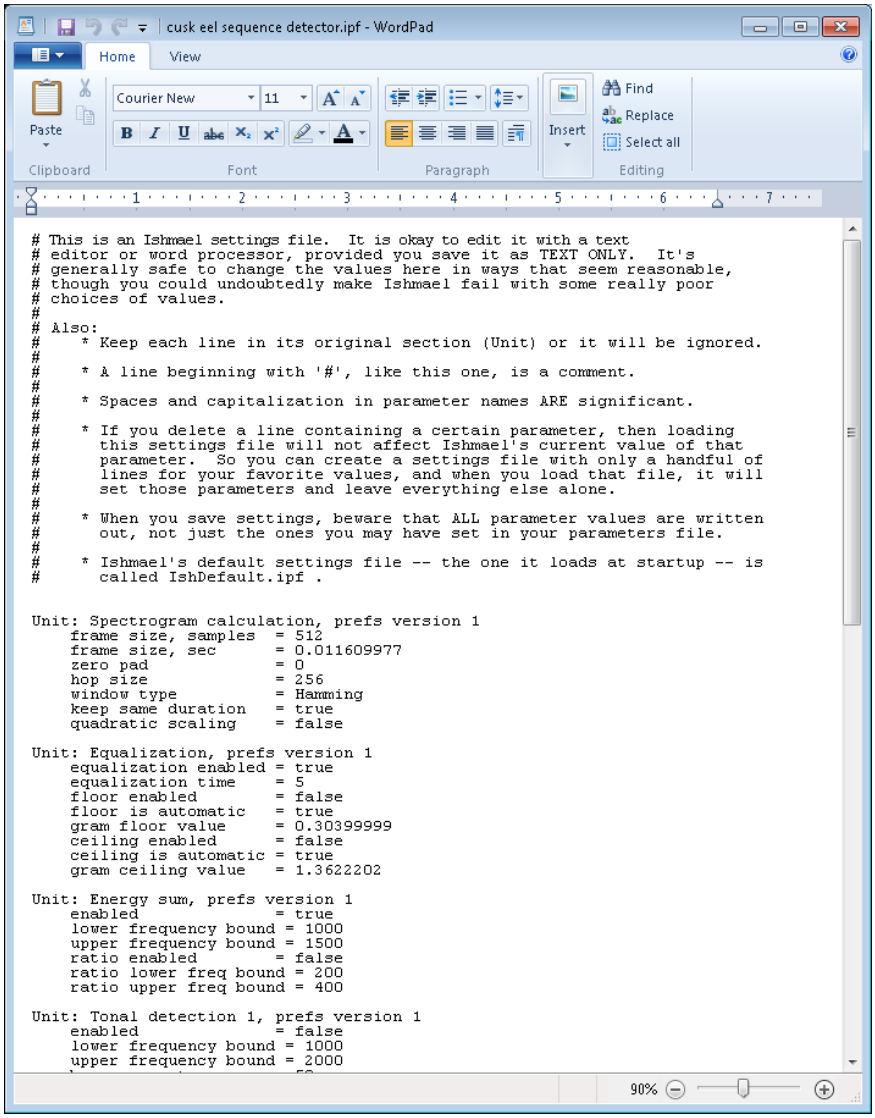
Often, after taking the time to configure Ishmael to build a detector, or to record, view or convert a sound file, you will want to save the settings file, a.k.a. **I**shmael **P**references **F**ile (**IPF**), that contains all of your choices.

Choose the **File→Save settings as** menu option, and then in the dialog box you can either save all your preferences, including detection parameters, your logfile configuration and the location of your files on your computer, or just the detection parameters. Give your IPF file a name that reflects the Ishmael choices you are saving and has the extension ".ipf" (for example "my whale detector.ipf").

Here is an example where we have chosen to save all preferences which results in an IPF file that can be edited with any text editor



You can save your current Ishmael settings in this file by choosing **File→Save settings as default**. Then the next time you start Ishmael, these settings will be loaded and used. You can also drag and drop an IPF file into the Ishmael window and it will load automatically.



Preferences files are just text files, and if you want, you can open them and edit them in your favorite text editor (Notepad, Notepad++, WordPad, MS Word, etc.). Be sure to save the file as plain text.

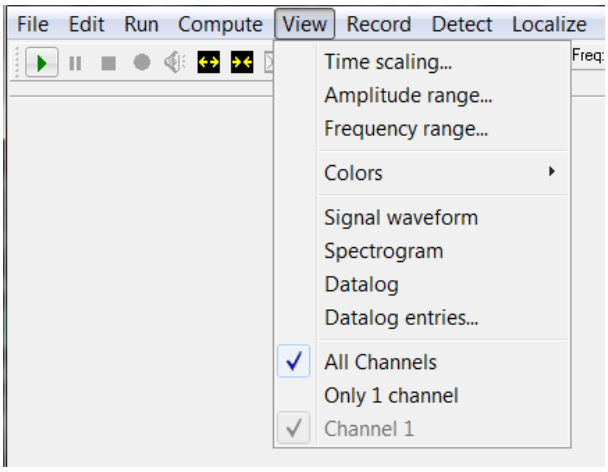
Make sure you pay attention to the notes in the header of the IPF file when manually editing a preferences file.

Now that you have your preferences saved, if you'd like to restart a detector or try it on a different dataset, you can load the saved IPF by choosing the **File→Load settings** option and navigating to your saved IPF file. You can also drag it from the Windows file browser and drop it on Ishmael.

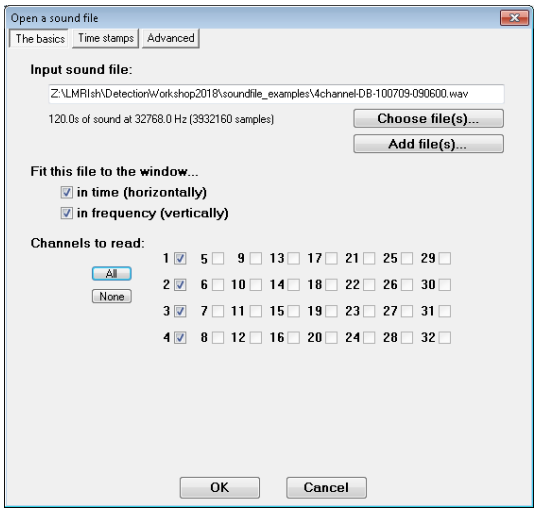
Note: When you save all preferences, this includes the name and location of your logfile. To avoid confusion, make sure you give your logfile a new name with each run of a detector. If you forget to do this – don't panic. Ishmael will not overwrite an existing logfile but will append to it.

Opening a multi-channel file

In a multi-channel file, you have the option of looking at just one or more than one of the channels, which is available via the **View** menu.



Open the file **4channel-DB-100709-090600.wav**. On **the basics** tab, you can choose which channels to read into Ishmael. In this example, we have four channels in this wav file, and Ishmael finds these and automatically chooses all of them. If you want to clear these four channels, click the **None** button, then choose one or two channels, click **OK**, then click the green **Run** button. You will need to adjust the brightness and contrast, and you may need to adjust the time and/or frequency scaling, to see a clear spectrogram.

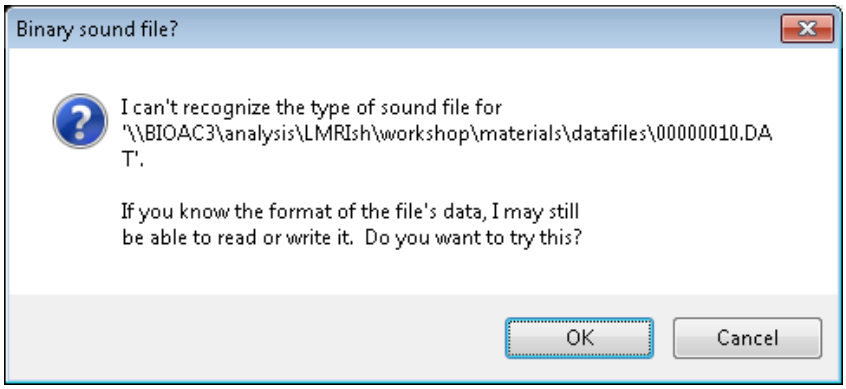


Note the difference between the channels that Ishmael *reads in* and the channels that you *view*. The channels that are read in are given to all of Ishmael's analysis tools, like detection, localization, and recording, even if you don't happen to be viewing all of them. You can view (in the spectrogram and signal waveform) some or all of the channels that are read in. Do this via the **View** menu.

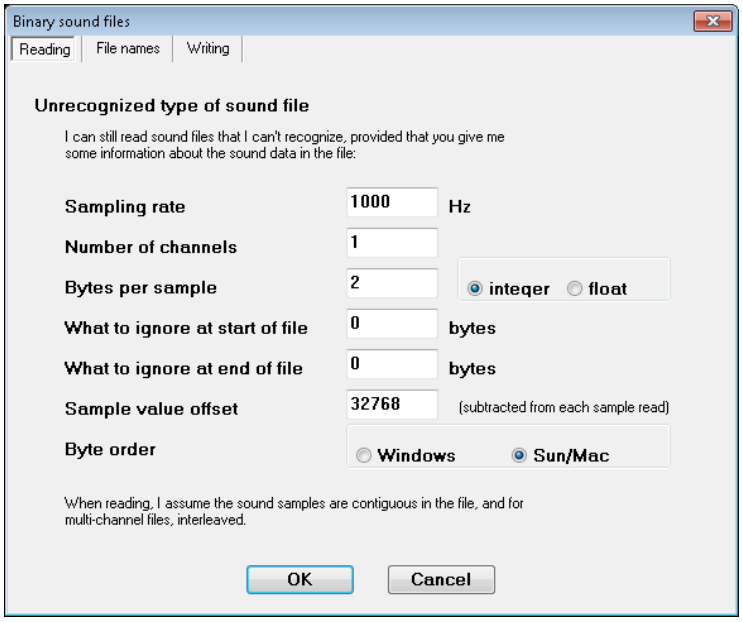
Opening unknown file formats

Ishmael is also able to open binary files that are not the typical .wav, .aif, or .au formats.

Try opening the file **00016945.DAT**:



This is a file format used in our PMEL/OSU hydrophones. Ishmael can open this file if you give it information about the sound file. Typically you must get this information from the person who created the files (or the instrument that recorded the files).



- **Sampling rate** is how fast the sound signal was sampled.

- **Number of channels** can be anything up to 64 channels. For most audio files it's either 1 (mono) or 2 (stereo), but a lot of marine data files, such as ones from towed arrays, have more channels.

- **Bytes per sample** says how large each data sample is. Most commonly this is 2 (16-bit samples), though Ishmael also supports 1 (8-bit samples), 3 (24-bit samples), and 4 (32-bit samples). And most audio data files have integer sample values, but a few use floating-point samples.

- **What to ignore at start of file** allows Ishmael to ignore a header in your sound file. You need to know how large, in bytes, the header is.

- **What to ignore at end of file** allows Ishmael to ignore a trailer. This is somewhat rare to have in audio data files, but a few formats have it. Again, find the length in bytes.

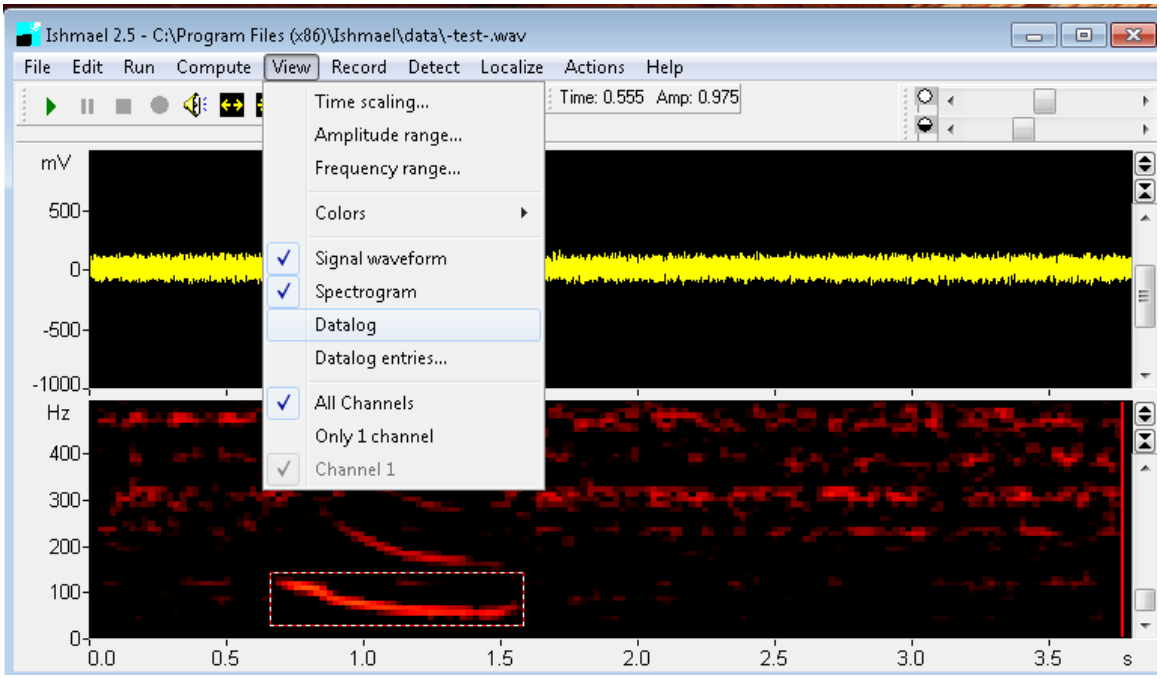
- **Sample value offset** is subtracted from each sample as Ishmael reads it in. For most audio files this will be 0, but some files use only positive numbers and require a subtraction to get a zero-mean signal. The file in the example above, **00016945.DAT**, uses a sample offset of 2^{15}, or 32768.

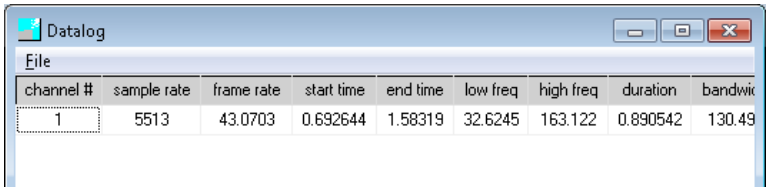
- **Byte order** refers to how samples are encoded. Unfortunately, Windows and much of the Unix world encode numbers differently, and you need to know which one you have. (The Windows encoding is also called "little-endian" and the Unix one "big-endian".) If your data has 1 byte per sample, you don't have to worry about byte order.

Acoustic measurement in Ishmael

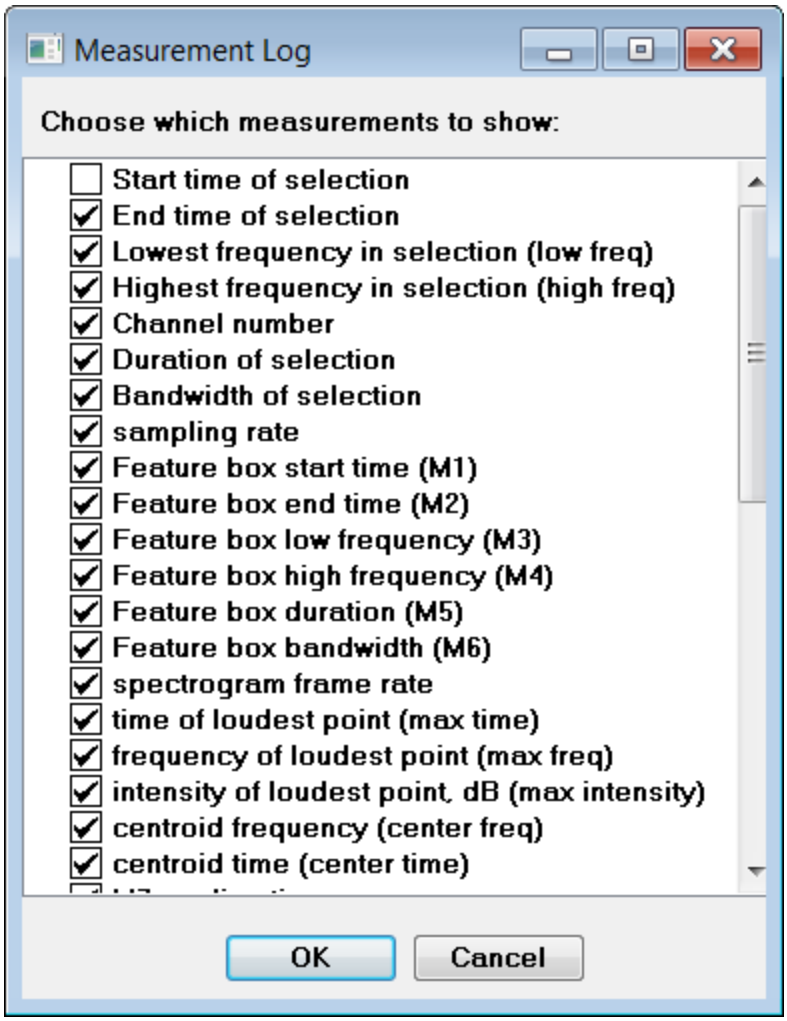
Making and storing measurements

Ishmael can be used to make measurements of calls. Hold down the left mouse button, drag the cursor to make a box around the signal of interest, then **right click** and the measurement log will appear. Each successive measurement will be added to the log. If the log has gotten closed, you can re-open it with **View→Datalog**.





You can change which measurements are made via the **View→Datalog entries** menus. The basic measurements – the start and end time of your box, as well as its lower and upper frequencies – are the first four measurements listed. These are followed by a long list of acoustic measurements of various sorts.



The parameters beginning with an **M**, like **M1**, **M2**, **M3**, etc., are part of a set of special *noise-resistant measurements*. These measurements focus on the loudest parts of calls – the parts that are likely to be present even when noise levels are relatively high – and thus offer some stability in varying noise levels. The system is described in detail in the **Mellinger and Bradbury 2007** article (see **References** at end of this tutorial), which is in the 'articles' folder.

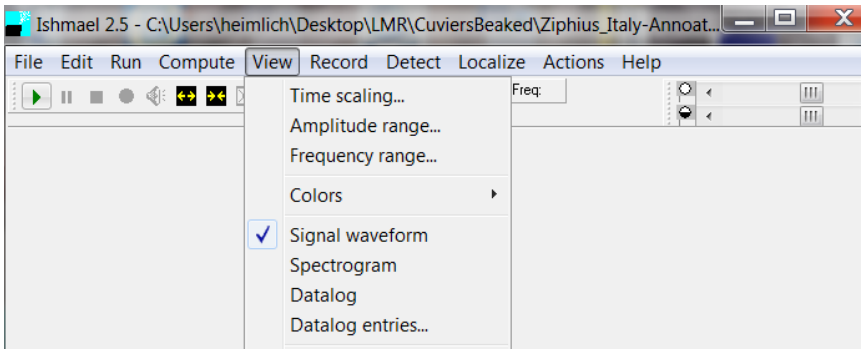
Converting sound file formats

In order to convert sound files to a new format, we will basically be opening these files and "re-recording" with Ishmael using settings for the desired format.

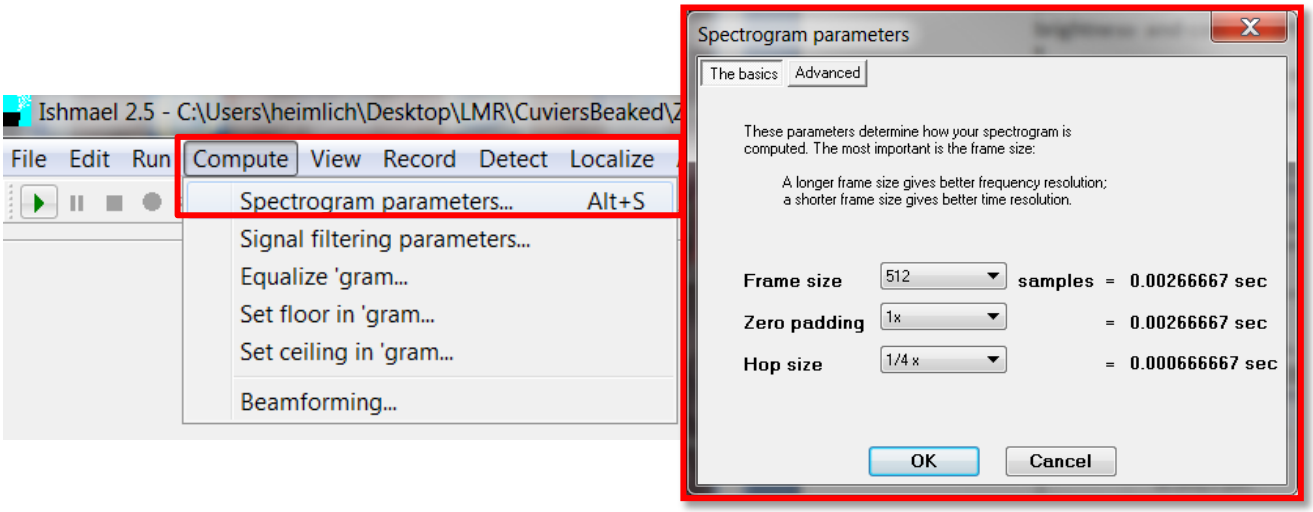
1) Load the settings file **Ishmael Factory Settings.ipf** to reset your Ishmael settings.

2) **Spectrogram settings:** Open the sound file **BlueWh-NEP-etp-ne-00095-1500-short.wav** with **File→Open file.**

On the **View** menu, choose **Signal waveform**. Click **Run** on the menu bar or the **Run** button (the green arrow under the menu bar), then adjust the brightness and contrast scrollbars in the upper right corner of the window until you can see data in the spectrogram.

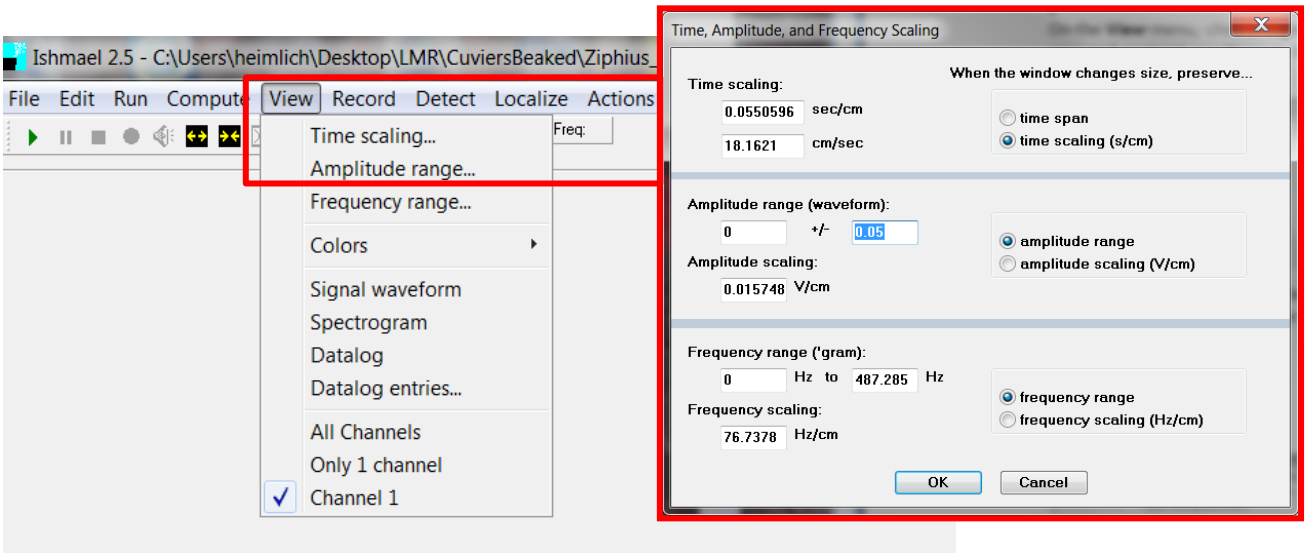


If you want to see the sound clearly as it's being re-recorded, adjust the spectrogram parameters to clarify the image using **Compute→Spectrogram parameters** and tweak the brightness and contrast.

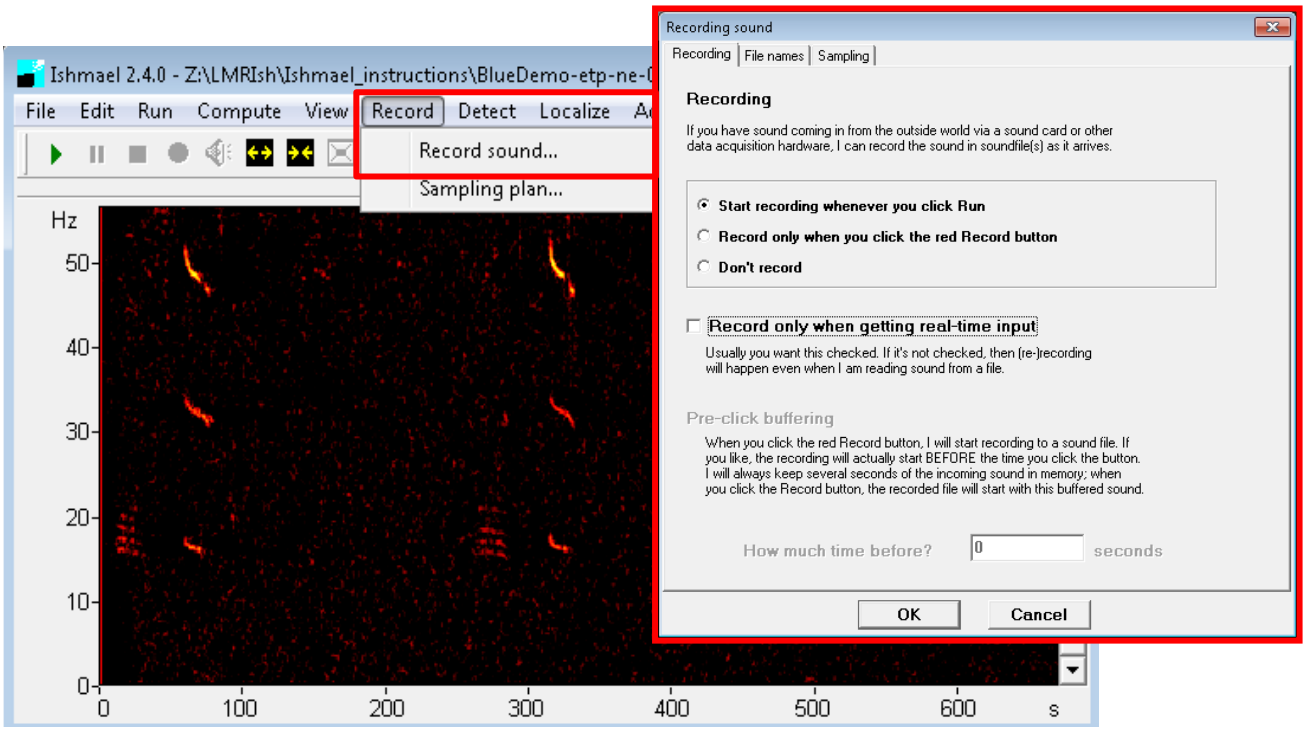


Note: at this time, Ishmael can convert to .wav, .aif, and .au files, which are the most popular lossless sound file formats on Windows, Mac, and Unix respectively; MP3 format is not supported. Make sure you use the correct extension for your desired file type!.

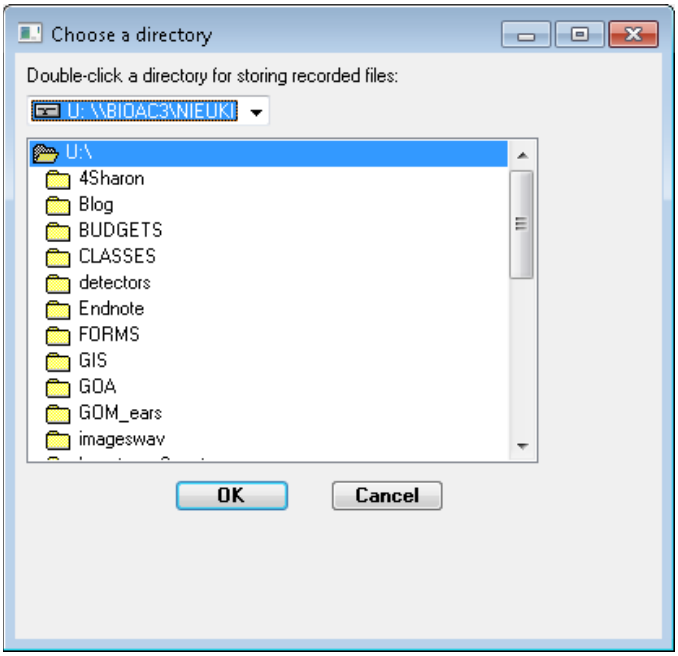
Then adjust **View→Amplitude range** until you can see the waveform clearly. If you want to save these settings for future use, do **File→Save settings as default.**



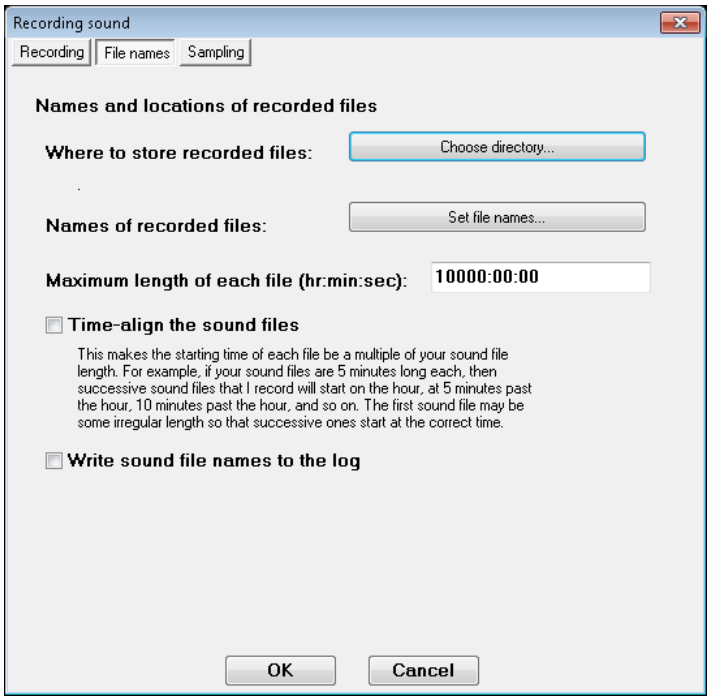
3) Choose **Record→Record sound** from the menu. Check "**Start recording whenever you click "Run**" and *uncheck* "**Record only when getting real-time input**" on this page, since you're now using input from a file. Click on the **File names** tab at the top of the menu, set the maximum length of each file to a time longer than any of your sound files – say, 1000:00:00 – and un-check **Time-align the sound files.**

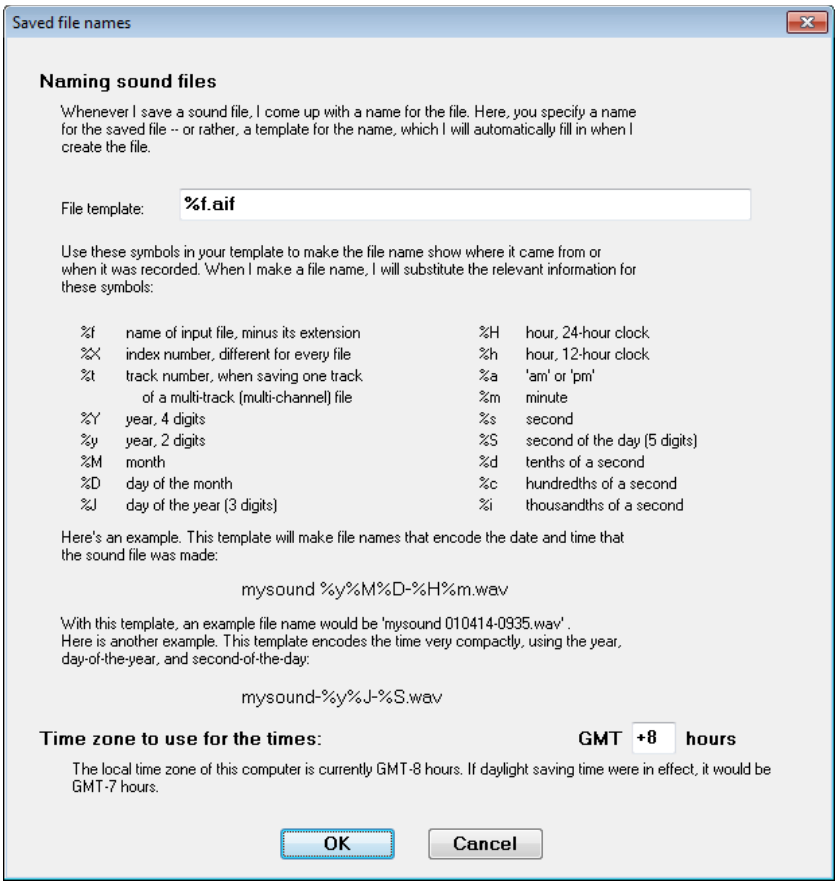


4) Choose the directory (folder) in which you want to store the converted files



5) Click on the "**Set file names**" button. For the file template enter **%f.*exten,*** where ***exten*** is the extension of the file type you are converting to. For instance, if you are converting WAVE files (with extension .wav) to AIFF files (with extension .aif), enter **%f.aif** for the file template. The time zone choice here doesn't matter. Click **OK.**





6) Click **Run** on the menu bar or the **Run** button (the green arrow under the menu bar). Your converted sound file should appear in the folder (directory) you chose for recorded sounds.

7) Save your settings for the file conversion. Choose **File→Save settings as default** OR save your settings as an ipf file and give it a name, like **Convert_to_ AIF.ipf**, that reflects the file conversion parameters that you have chosen.

8) It's usually a good idea to test a converted file to make sure things worked correctly and that your naming convention was interpreted properly.

9) You can also sub-sample your data to convert only part of it, but that's beyond the scope of this tutorial. (**Record→Record sound→Sampling** or **Record→Sampling plan)**

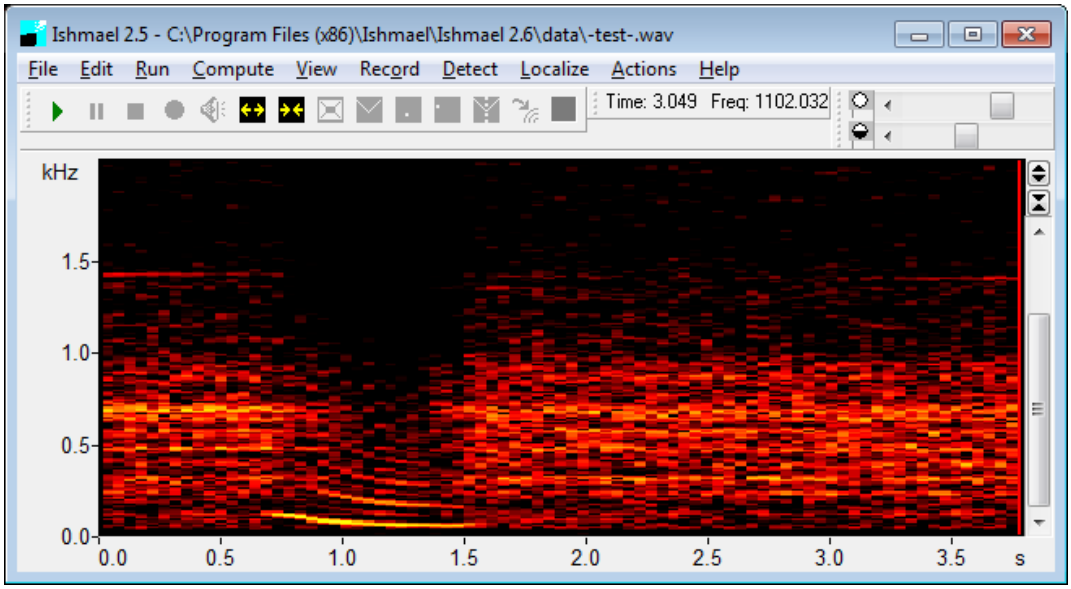
10) **Batch Processing:** To convert a number of files en masse, do **File→Open file** and simply choose all the files at once using the Shift and Control keys when you click with the mouse, or type **Ctrl+A** to select all of the files in a folder. Then instead of clicking the **Run** button, choose **Run→Batch run** from the menu. Un-check the box labeled **Pause after each file** and start the batch run.

Filtering

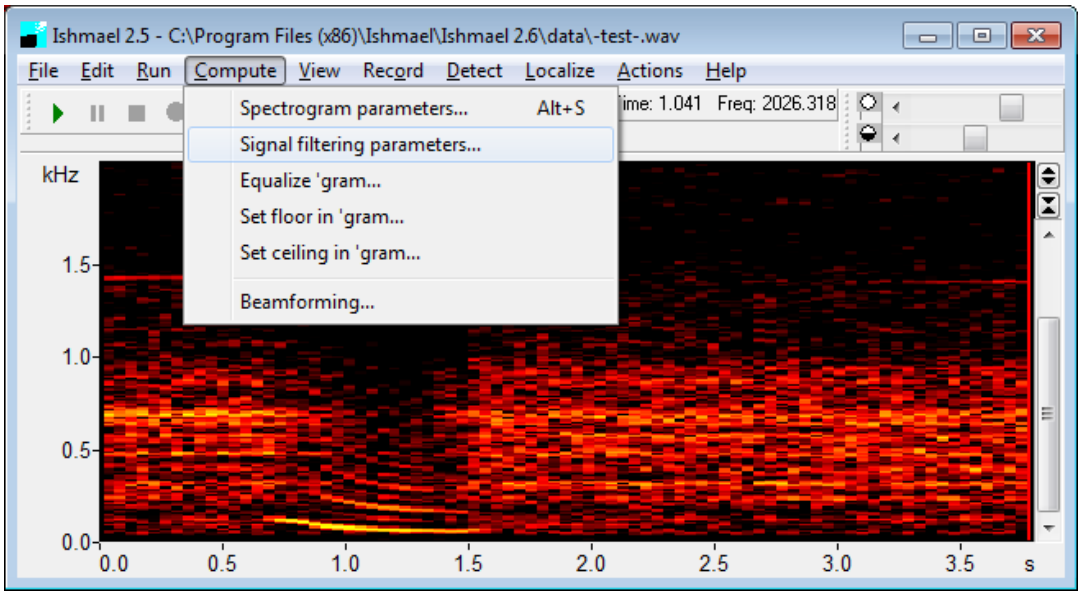
There are a few issues in signal analysis that can be best handled by filtering: removing noise that occurs in a certain frequency band, and filtering a signal in preparation for changing its sample rate.

1) Load **Ishmael Factory Settings.ipf** to reset your Ishmael settings.

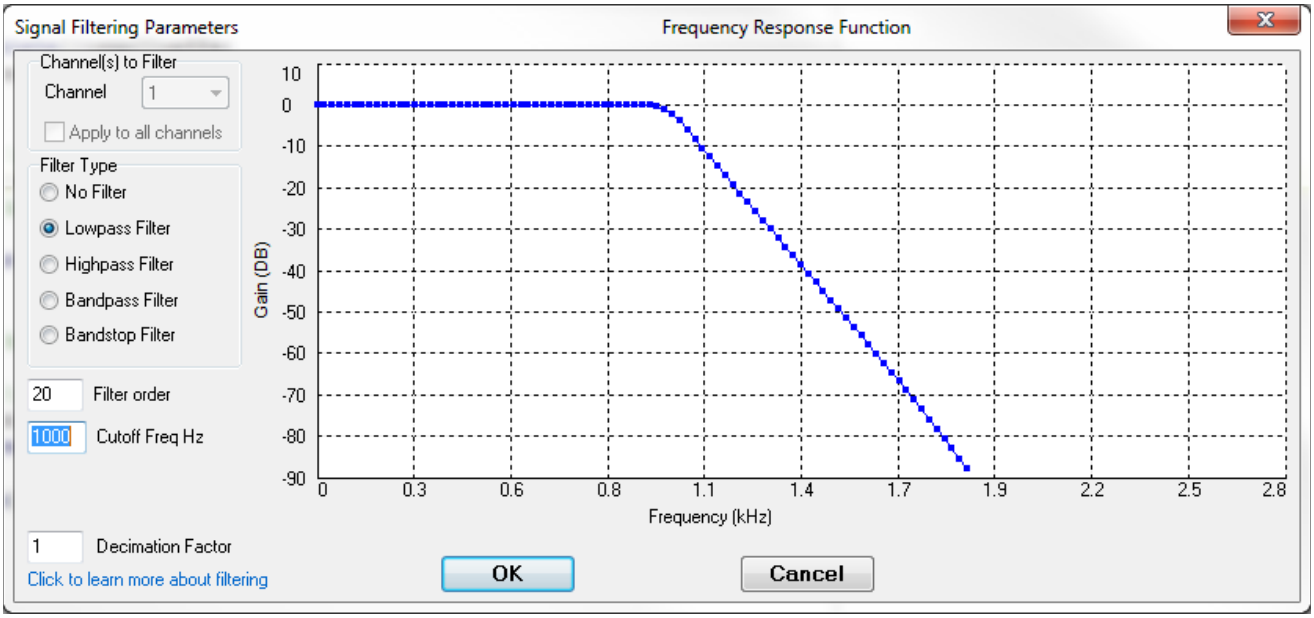
2) To remove noise that occurs in a certain frequency band, we want to filter out that band, or alternatively speaking, pass any sound in the remaining frequencies. For example, open **test.wav** again and make a spectrogram with a frame size of 512 samples. Adjust its brightness so you can see the downsweeping sound around 1 second as well as the bands of noise:



3) Since the interesting sound here occurs below 500 Hz (0.5 kHz), we can remove the sound above that frequency – or alternatively, "pass" the sound below that frequency -- using a *lowpass filter*. To do this, open the filtering dialog box (**Compute → Signal filtering parameters**…).

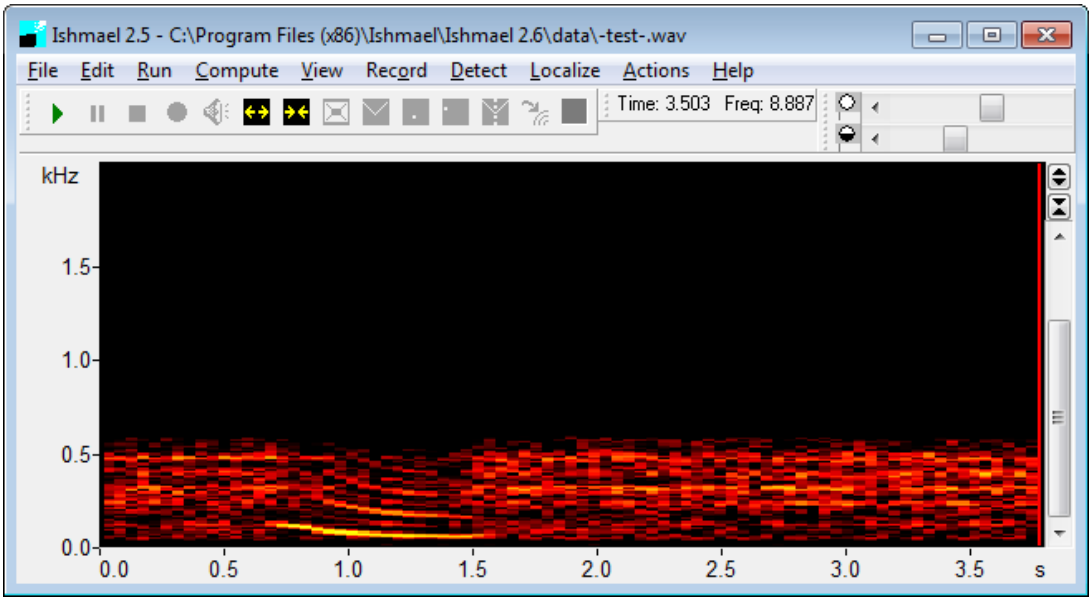


4) In the dialog box that opens, choose **Lowpass Filter**, and enter a **Filter order** of 20 and a **Cutoff frequency** of 1000 Hz:



5) The Frequency Response Function is shown on the right. For each frequency on the X-axis (horizontal axis), the *gain* of the filter – the amount of amplification or reduction – is shown. For frequencies up to about 500 Hz (0.5 kHz), this filter has a gain of 0 dB, which means there is no gain or loss of amplitude. But at frequencies above 500 Hz, the signal will be reduced in amplitude more and more, as represented by the gain values becoming more and more negative. At a gain of -60 dB, sounds are pretty much eliminated, and even at -30 or -40 dB, they are very much reduced**.**

6) Click OK and then Click **Run** on the menu bar or the **Run** button (the green arrow under the menu bar):



You can see that the sound above about 500 Hz is nearly absent. The lowpass filter has allowed the frequencies below 500 Hz to pass through it nearly unaffected, while frequencies above that are removed.

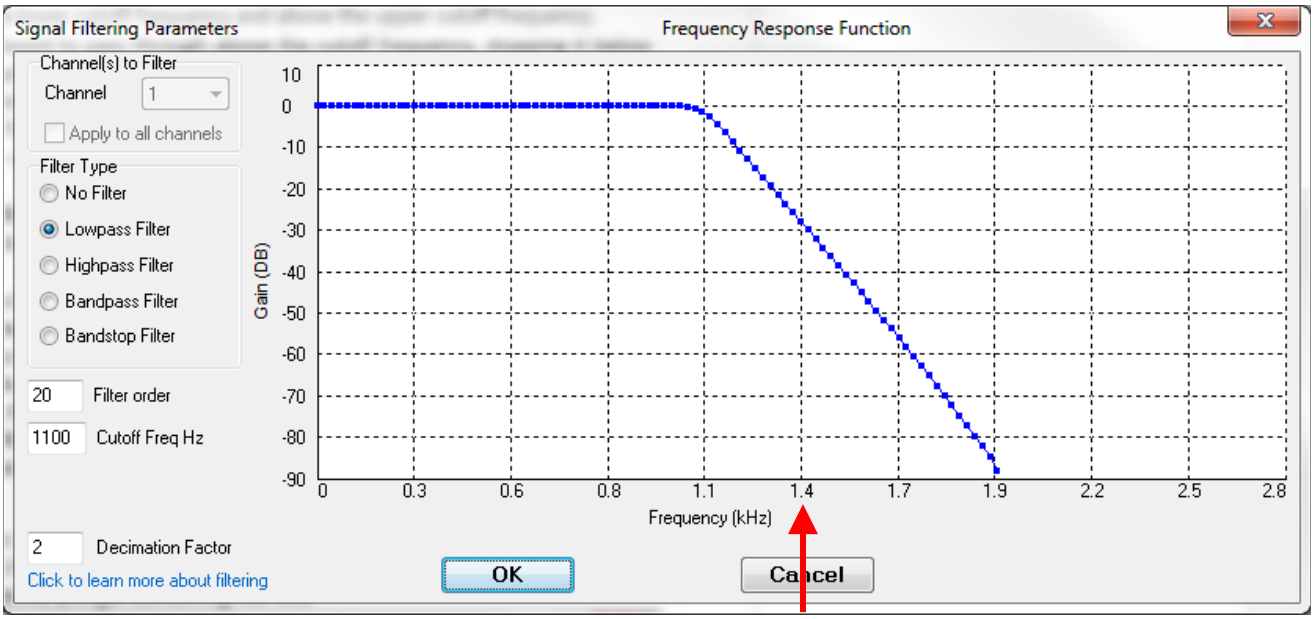
7) What do the various signal filtering parameters mean?

- ***Channel*** says what channel you're applying this filter to, which is applicable only for multi-channel files. You can have a different filter for each channel. Or by checking the "Apply to all channels" box, you can use the same filter across all the channels.

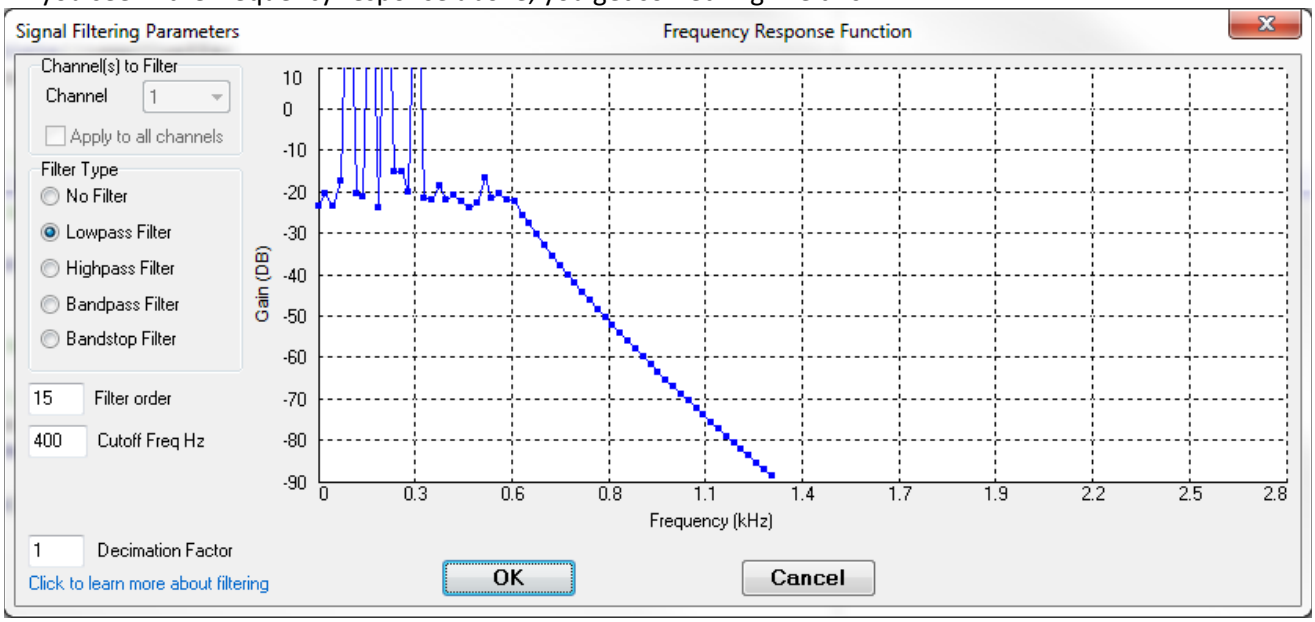
- ***Filter type*** indicates how you want the filter to work:
  1. A ***bandpass*** filter allows sound to pass through between two frequencies you specify, stopping sound below the lower cutoff frequency and above the upper cutoff frequency.
  2. A ***bandstop*** filter is the reverse: It stops sound within a given frequency range, allowing sound to pass through below the lower cutoff frequency and above the upper cutoff frequency.
  3. A ***highpass*** filter allows sound to pass through above the cutoff frequency, stopping it below that. This is useful when you have loud low-frequency sound that you want to eliminate.
  4. A ***lowpass*** filter, like the one above, allows sound to pass through below the cutoff frequency, stopping it above that. This is useful for eliminating high-frequency sounds (which can be annoying to listen to) as well as in preparation for changing the sampling rate of a sound by decimation.

- ***Filter order*** says how many filter coefficients there are, which in turn affects how sharp the cutoff is – i.e., how quickly sound is stopped below or above the cutoff frequency – as well as filter response stability.

- ***Decimation factor*** allows you to re-sample a sound signal to a lower sample rate. A lower sample rate saves on computation time and, if you then re-save the sound using **Record → Record sound**, storage space on disk. For this to work correctly, you need to calculate the upper frequency limit of the re-sampled sound: do this by dividing the current upper frequency limit by your decimation factor. In the example shown below, the decimation factor is 2, so the new upper frequency limit is 2.8/2, or 1.4 kHz. Then design your filter so that is at least -20 to -30 dB down at that frequency; at the frequency of 1.4 kHz, the filter shown below is about -25 or -30 dB down, so it would work well for decimation:



8) ***Filter response stability*:** Sometimes when you pick a filter type and specify the filter order and cutoff frequency or frequencies, the filter's frequency response goes haywire. Instead of the nice smooth curve you see in the frequency response above, you get something like this:



This means that the filter design algorithm wasn't successful in creating a useful filter. To solve this problem, try different filter order and cutoff frequency values until you get a reasonable filter. Generally speaking, you get a better response curve by (1) using a lower filter order, and (2) moving the cutoff frequency (or frequencies) away from the minimum and maximum possible frequencies – from the left and right sides of the response function graph. Play with these parameters until the filter design algorithm gives you a reasonably smooth graph.

Detection

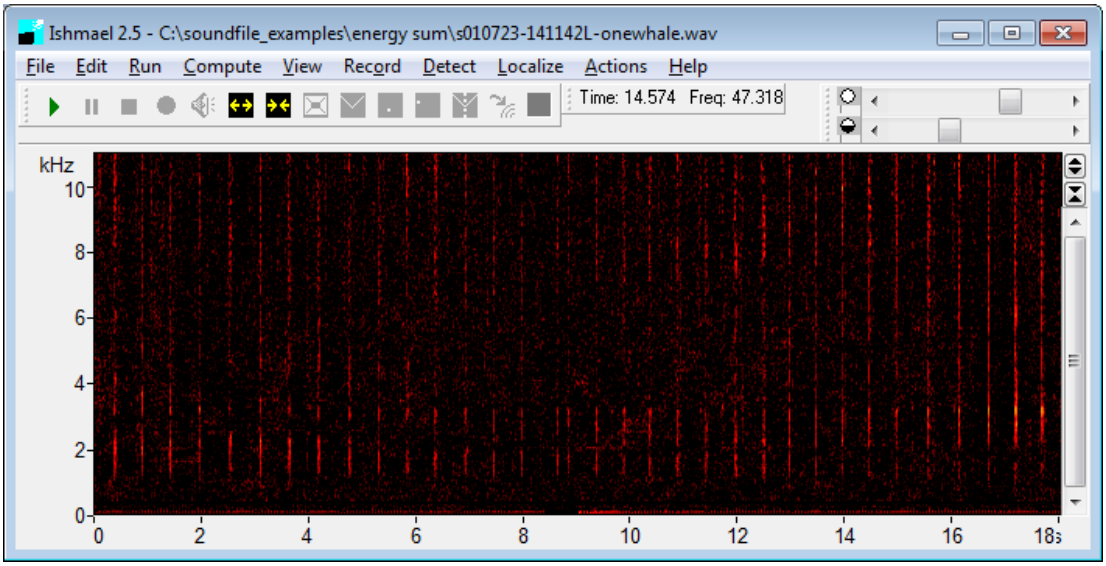
Ishmael has a variety of ways for detecting calls in a sound signal. The simplest of them is the energy sum detector, which simply measures the amount of sound present in some frequency band of the spectrogram.

Energy sum detection

This example is for detecting clicks sounds from sperm whales.

1) Load **Ishmael Factory Settings.ipf** to reset your Ishmael settings.

2) In Ishmael, open **s010723-141142L-onewhale.wav** from the energy sum folder and display the spectrogram (by clicking the **Run** button), adjusting brightness, contrast, time scaling, and frequency scaling as needed to see the spectrogram.

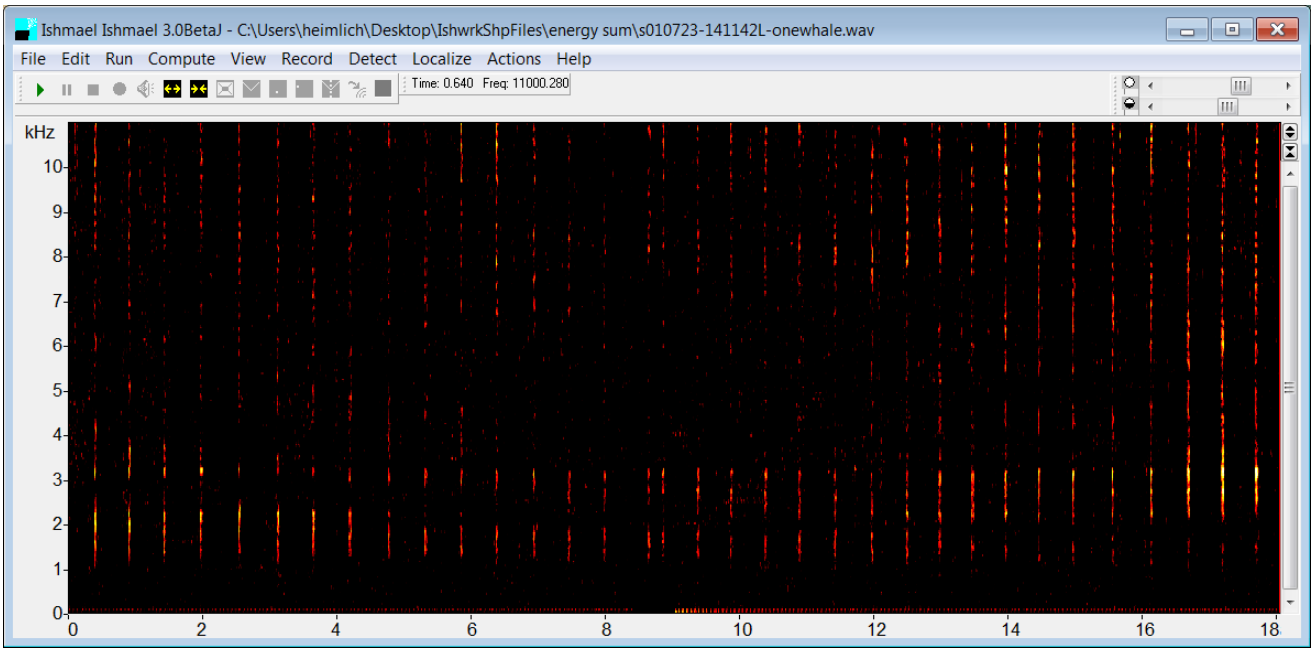


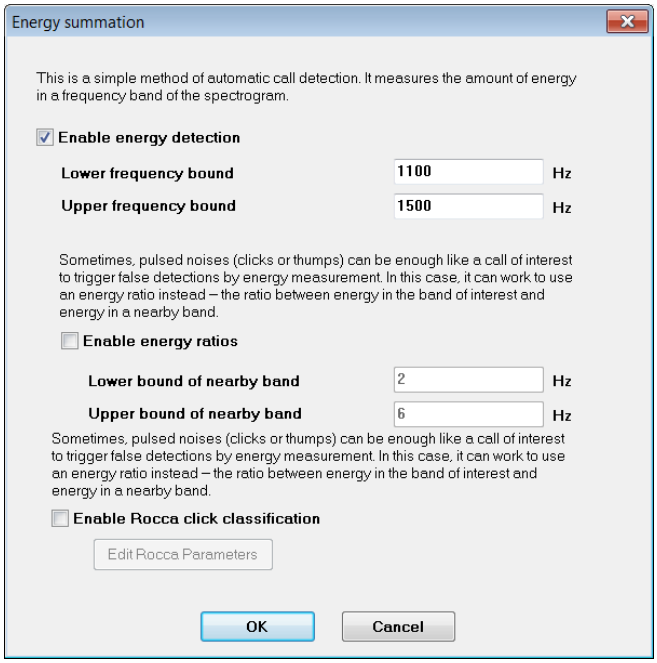
3) Adjust spectrogram parameters (**Compute→Spectrogram parameters**), particularly the frame size, until you can clearly see the clicks that are present – they should be well separated from each other in the spectrogram. A frame size of 512 works pretty well.

4) Turn on equalization (**Compute→Equalize 'gram**) using a time constant of 1 second. Also in these controls turn on **Use spectrogram floor value**, which will prevent hidden data in the black part of the spectrogram from affecting the detection, and make sure it's on "Automatic". Re-adjust the brightness and contrast**.**

Note: Turning on equalization and changing the spectrogram settings improves the detection function as it affects the data directly (not just the display). The data are NOT changed

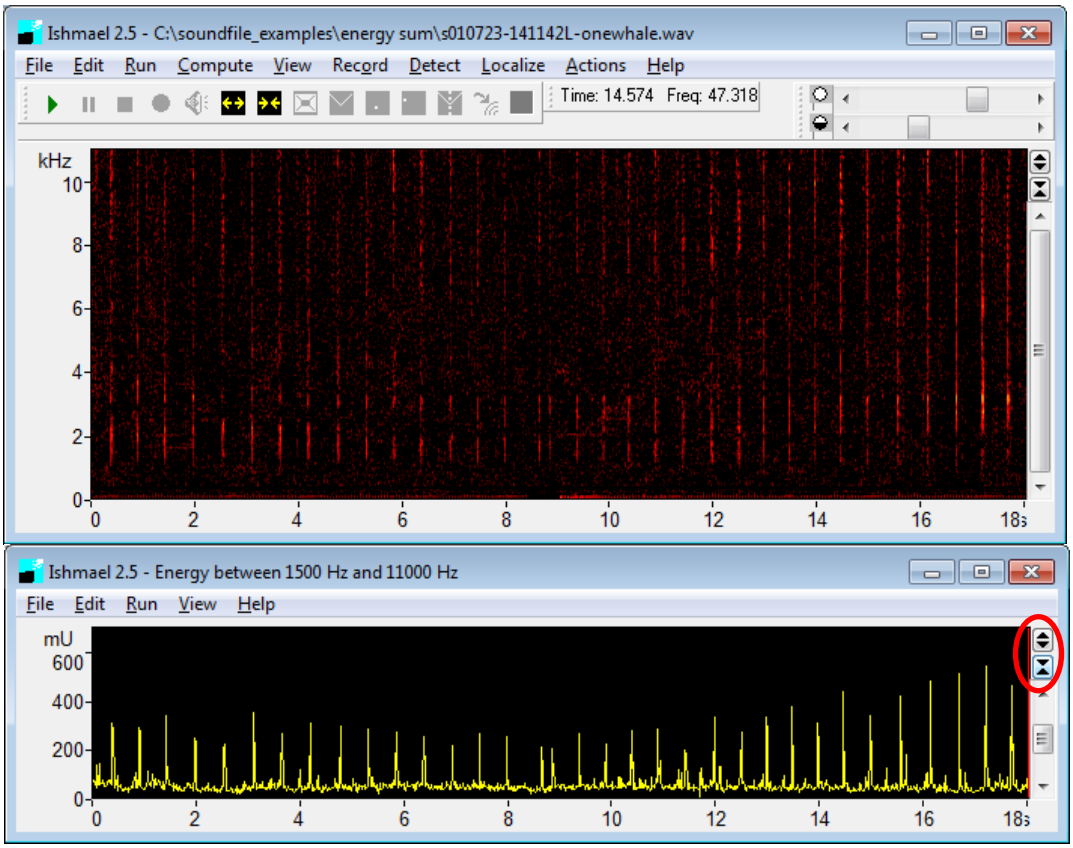
5) Set up the energy-sum detector: Do "**Detect→Energy sum**", enable detection, and choose frequency bounds. The bounds should encompass the frequency range where the clicks occur but not where interfering sounds are. For this sound, there aren't a lot of interfering sounds left after equalization, so use the whole band where the clicks occur: 1500 Hz for the lower limit and 11000 Hz for the upper limit. Keep "Enable energy ratios" unchecked for now.





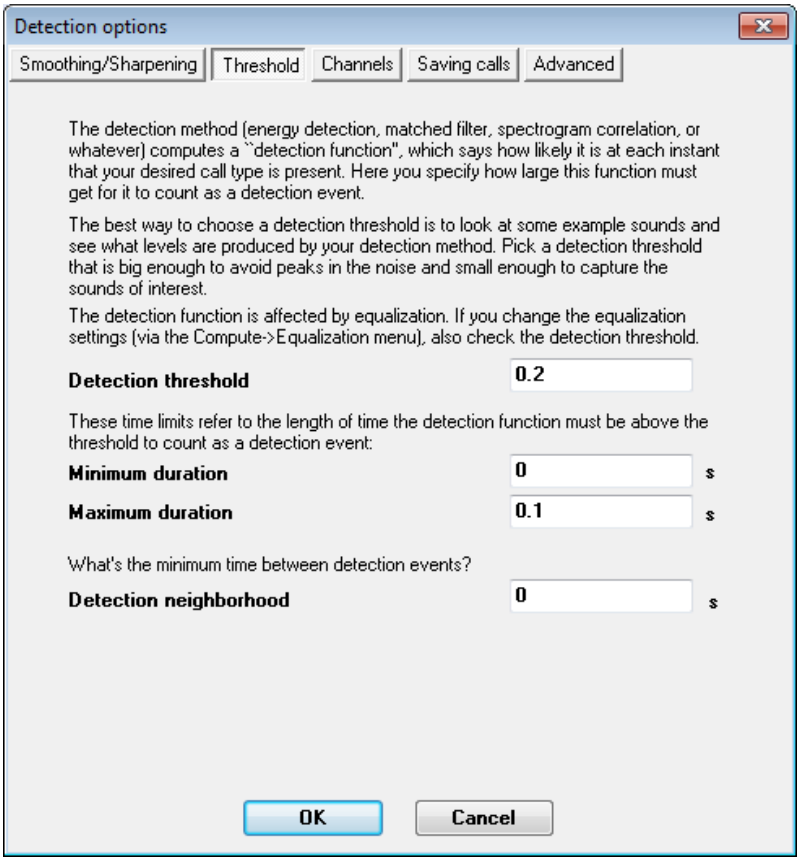
6) Click "**Run**" on the menu bar or the **Run** button (the green arrow under the menu bar). You should see a new window down below that has the *detection function*, which represents the likelihood that a call is present at each point in time.

Sometimes the detection function window is all black; if so, try compressing the display (using the button at the right edge of the window) several times until the yellow line appears.) Use the stretch/shrink buttons and scroll bar at the right edge of the window to change the scale until it looks something like this (your time and frequency scaling may be different:

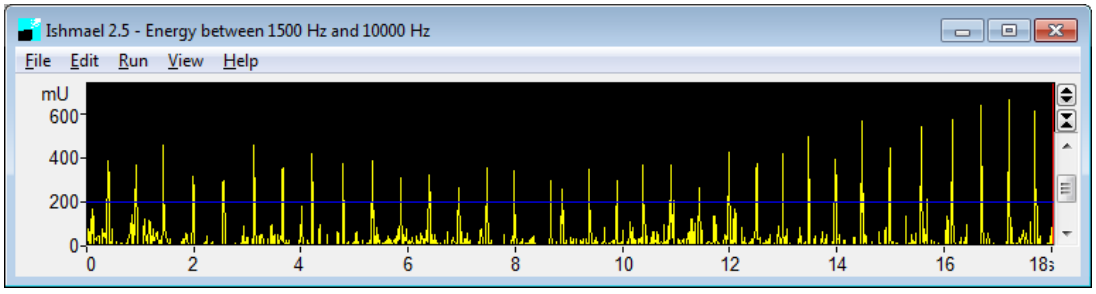


7) Look at the units for the detection function on the Y-axis (in the upper left corner) – "mU". Detection function values don't represent any physical quantity, so Ishmael uses the generic term "Unit", or U, for them. In this case the values on the Y-axis are 200, 400, and 600 "milli-Units". The "m" is a metric prefix meaning 1/1000, so 200 mU equals 0.2 U, 400 mU equals 0.4 U, and so on**.**

8) Set detection threshold, which will appear as a blue line in the detection window, with **Detect→Detection options->Threshold (tab)**. You can do this by looking at the Y-axis and choosing a value that's below the height of the peaks – the sperm whale clicks – but above the height of the "grass", the detection function value for background noise. Here, a value of 0.2 U (which equals 200 mU) or 0.15 U will work. Also enter a **Minimum duration** of 0 and a **Maximum duration** of 0.1 s:

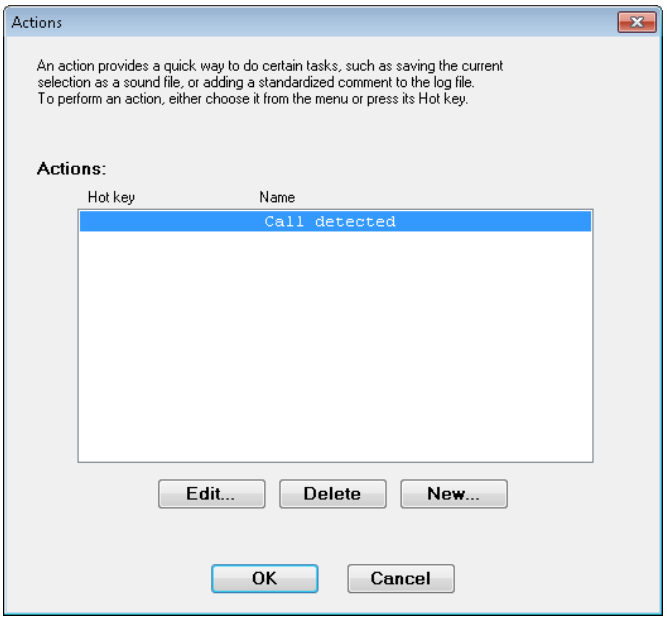


9) Click OK, then click **Run** to see the threshold, which Ishmael displays as a blue line. An example:

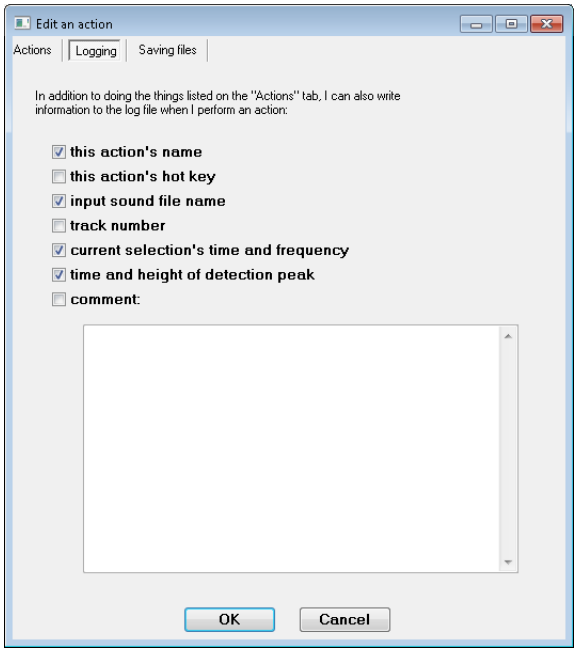


Ishmael uses the detection function and the threshold to decide what counts as a detection. Any time the function goes over the threshold, Ishmael will register a detection.

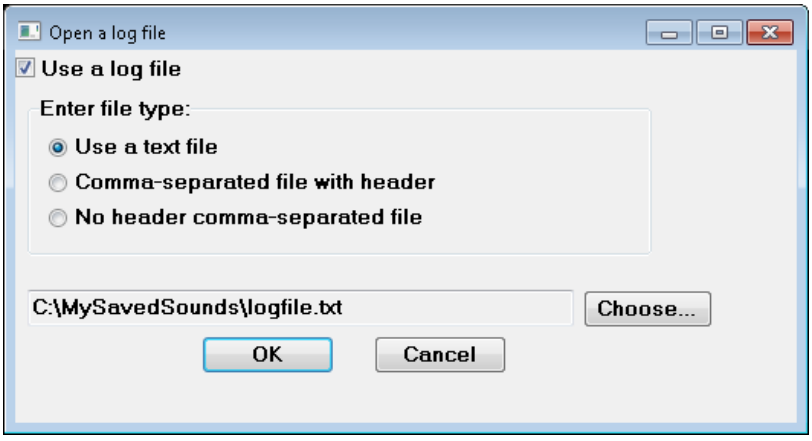
10) You've now set up Ishmael to detect the sperm whale clicks. What should it do when it registers a detection? The most useful things you can do are (a) write information about the detection to a log file, and (b) save the call as a small, separate sound file for examination later. To do these things, choose "**Actions→Edit actions**", pick the "**Call detected**" action, and click **Edit**. Then the "**Logging**" tab allows you to make log file entries, and the "**Saving files**" tab will set up automatic saving of detected calls**.**



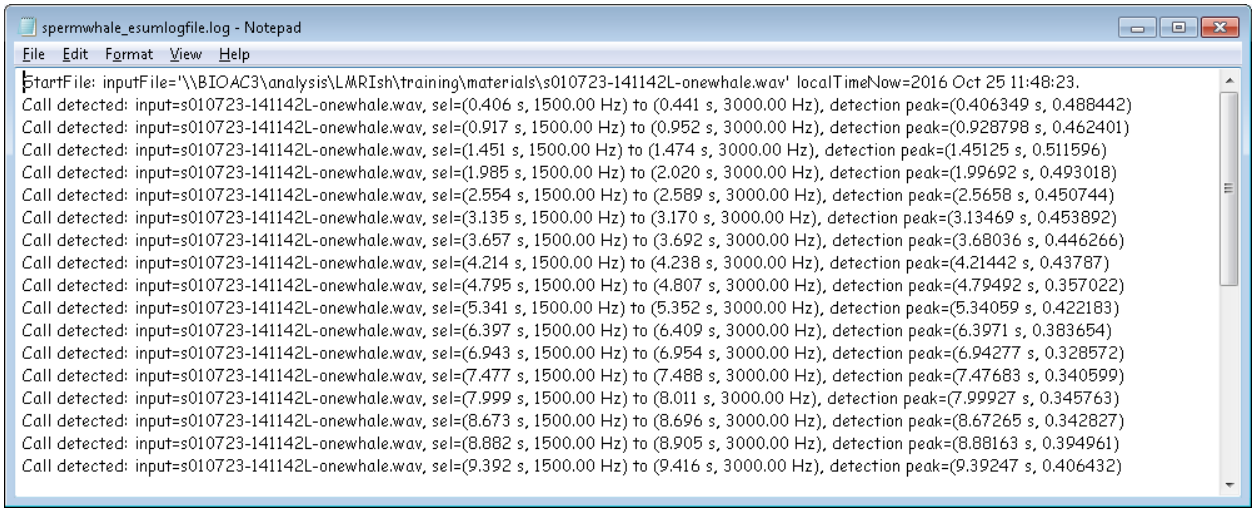
Set up logging of detection information to a log file. You'll need to specify what information to write: the action's name, which in this case is "Call detected"; the input file name; the time and frequency of the "current selection", which is the detection; the time and height of the detection function; what track (channel) the detection occurred in; etc.



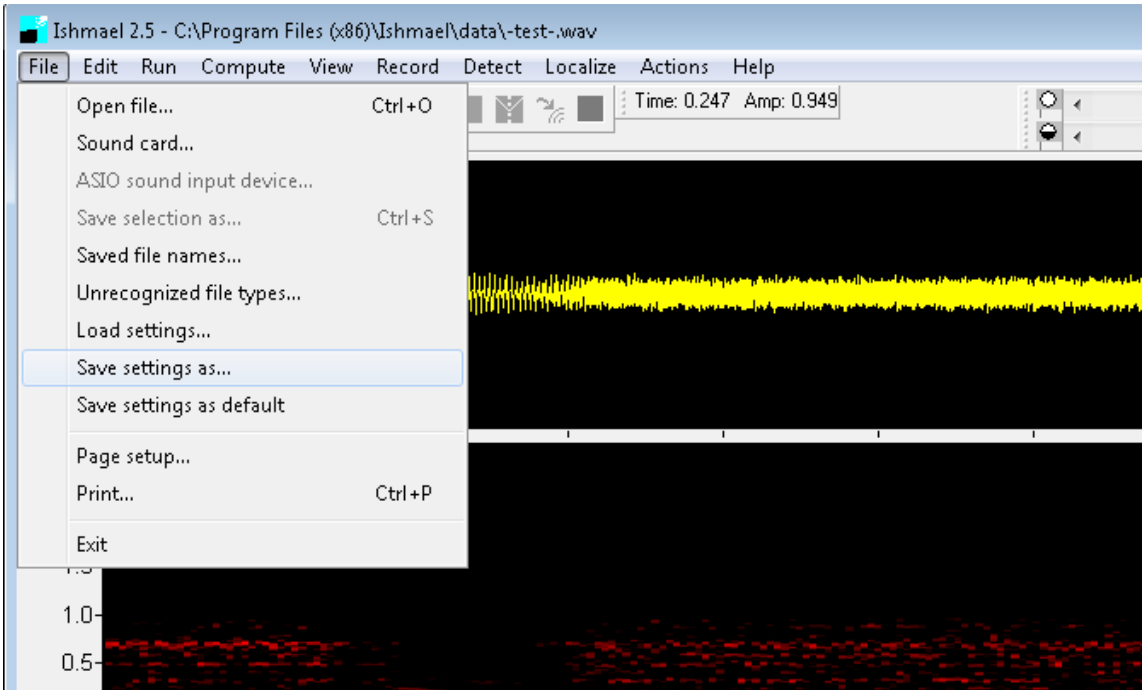
You'll also need to specify a folder and file name for Ishmael to write to, which you do via "**Actions→Open log file**" in the main Ishmael window. Check the box for **Use a log file** and choose a log file format: Text log files are relatively easy for a person to read but harder to use as input for programs, while comma-separated value (.csv) files are easy to read in Excel, MATLAB, etc., but are harder for a person to read. The .csv files have the option of having a header line describing each field in the file. Then choose a folder and file name for your log file.



Click "**Run**" and then examine your log file using WordPad, Notepad, MS Word, etc. for text files, or Excel or Google Sheets for .csv files. Here's what a text log file looks like:



11) Once you have your detector working correctly, make sure you save your detector settings by choosing **File→Save settings as**. Name your detector something that makes sense, and include some of the parameters that you've chosen in the name of the detector**.**



Note: We recommend that you always save (as an .ipf file) a copy of the detector you've used with each log file containing the results of that detector. This is especially important when running detectors on large data sets, since changing detection parameters will change what gets detected. It can also be useful to name your detector log file with the naming convention you've used for your detector. This ensures that when you revisit the results from a detection run in, say, a year, you are clear what parameters resulted in the detections that you logged, you or someone else can analyze other data using the exact same detection parameters, and that you can report these parameters to others..

12) Now you try it. Load **Ishmael Factory Settings.ipf** to reset your Ishmael settings. Load the file **Humpback-2002March12-JJacobsen@390s.wav**, which is a recording of humpback whale song units, and try setting up the detection parameters: First make a spectrogram so you can clearly see the humpback's vocalizations, adjusting the frame length, brightness and contrast, time scaling, and frequency scaling. (Hint: This is a bit longer file than most of what we've seen yet, so the view is compressed horizontally. You'll need to adjust the time scaling with the "stretch horizontally"  button. And you'll also want to adjust the frequency scaling -- click the "stretch vertically"  button – so you see what's happening in the range from 100 Hz to a few kHz.) Turn on equalization (**Compute->Equalize 'gram**) and **Use a spectrogram floor value**; the calls seem to be 1-2 seconds long, so try an equalization time constant that's about 4 times as long as that. Then turn on the **Energy sum** detector using frequency bounds that cover either the whole frequency band of these sounds, or perhaps just the lower portion of them. Adjust the vertical scaling of detection function display. Set a threshold for the detection function so that the peaks in the detection function go over it, and set your maximum call duration to something longer than your longest call, like 4 s. Open a log file to save the detection results, and enable saving of the start/end time and low/high frequency. When you're done, save your settings (use **Save all preferences**) so you can use this detector later.

13) Finally, load **Fin-93-001-1217.ch04.wav** and try building a detector for these fin whale calls on your own.

Improving the detection function

There are a number of things you can do to improve the detection function.

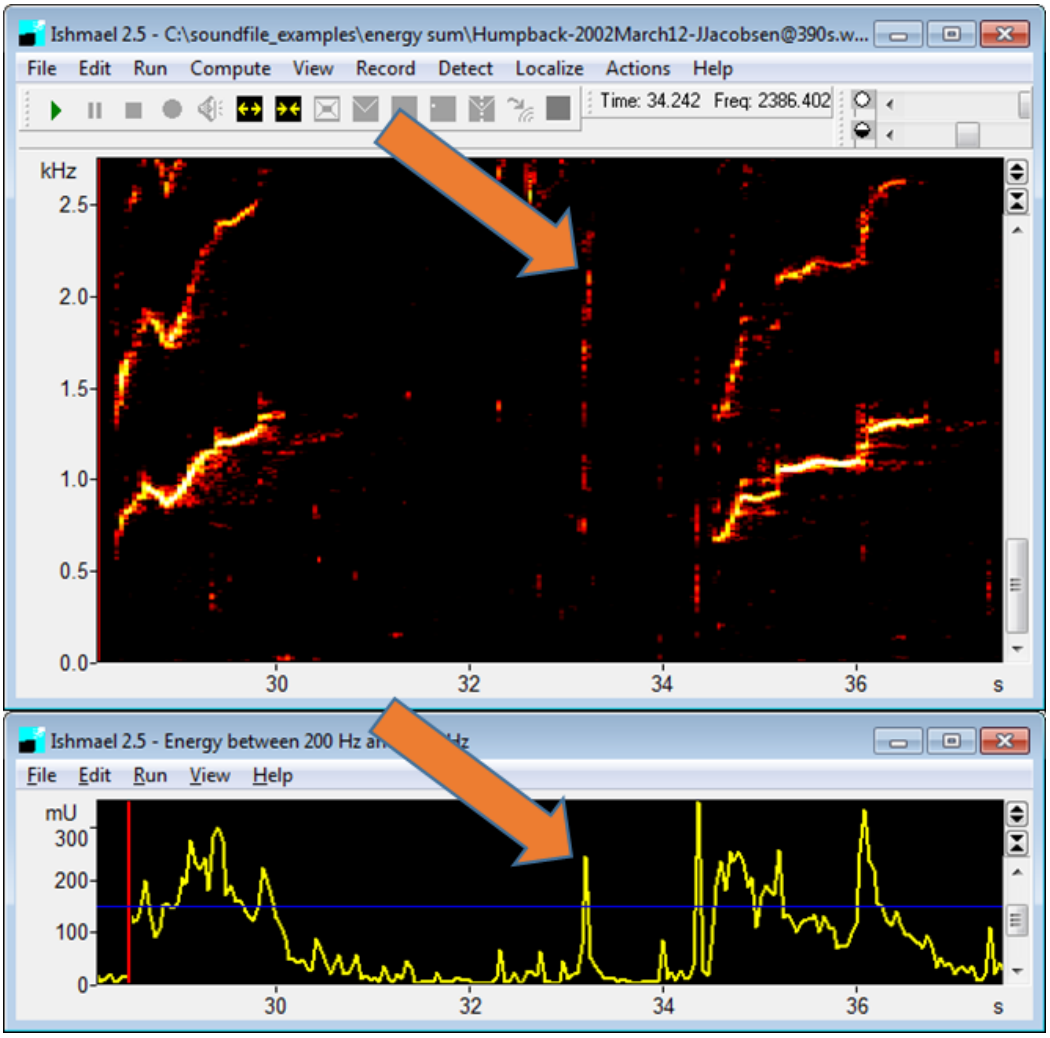
Detection neighborhood

Reload the settings you just saved for the humpback whale detector (in step 12) above). On the **Detect->Detection options->Threshold (tab)**, set a **Threshold** of 0.1 (i.e., 100 mU) and re-run it. You may have noticed in your log file that each vocalization you see in the spectrogram could be associated with several entries in the log file. This happens because the detection function wiggles around a lot, crossing above and below the detection threshold; every time it goes above, it results in a new detection and a new line in the log file**.**

You can stop this, and get only one log entry per call, using a **Detection neighborhood**, which is on the **Detect->Detection options->Threshold (tab)**. With a detection neighborhood of, say, 1.5 second, Ishmael will wait 1.5 s beyond the end of one detection before it starts another detection. (For Ishmael, the start and end of a detection are defined as the times the detection function goes above and below the threshold.) Enter this number for **Detection neighborhood** and look at the resulting log. The set of detections corresponds much better to the units you can see in the spectrogram.

Minimum/maximum duration

Another issue happens at 33 s, where an impulsive sound – the vertical streak in the spectrogram – triggers a detection**.**



To prevent a detection like this, you can set a **Minimum duration**, which is also on **Detect->Detection options->Threshold (tab)**. In this case, the vertical streak makes the detection function go over threshold for a very short time. If we set a **Minimum duration** longer than this, that streak won't register as a detection.

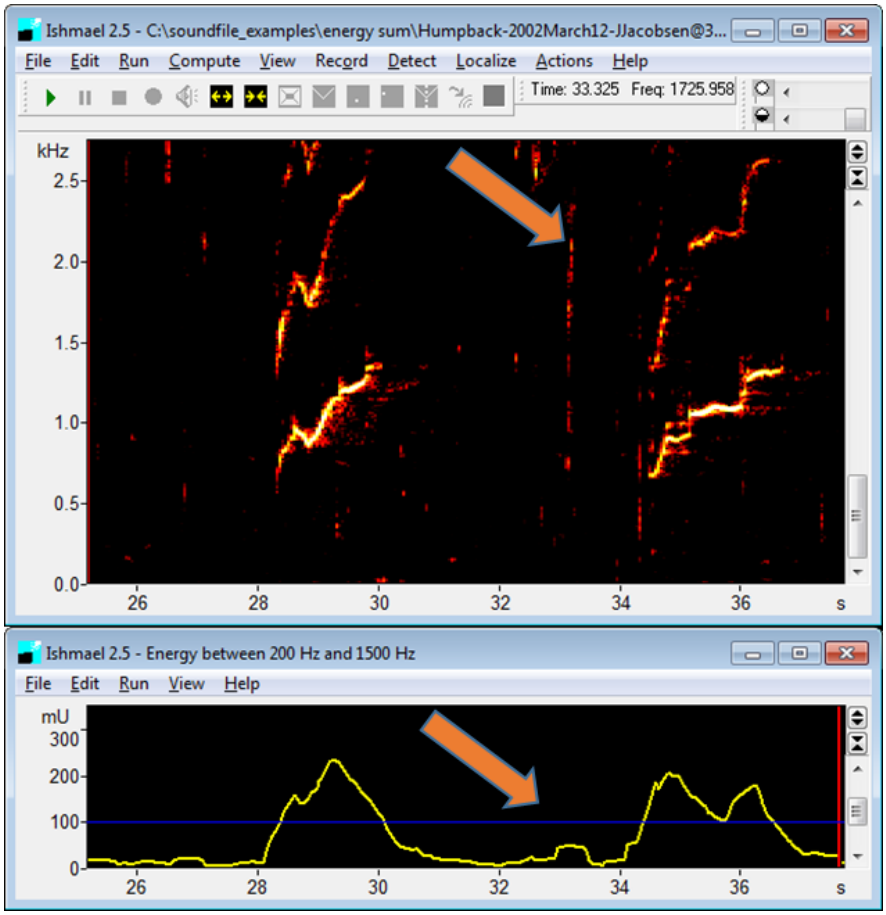
Try a **Minimum duration** of 0.1 s. This is long enough that it will eliminate short impulses like the one at 33 s in this sound file, but short enough that the detection function is over threshold that much for the actual humpback sounds.

There is a similar parameter **Maximum duration**. It's useful mainly when there are noises in your recording that Ishmael is detecting. If these noises last noticeably longer than your calls, you can often eliminate their detection by setting a maximum duration that's just longer than your call. Again, Ishmael uses the time that your detection function is over threshold as its "duration", so look carefully at the detection function when choosing a **Maximum duration** value to see how long it's above threshold for your calls as well as for the noise sounds.

Smoothing

Another way to handle spikes in the detection function is using **Smoothing**, which is on the **Detect->Detection options->Smoothing/Sharpening (tab)**. When smoothing is turned on, Ishmael takes a running average of the detection function.

Click **Enable smoothing** and try a **smoothing time constant** of 0.5 s, a number chosen to be longer than the rapid wiggles in the detection function but shorter than the calls themselves. Smoothing eliminates sharp spikes in the detection function, making it easier to detect calls. One aspect of this is the "grass" that you see at the bottom of the detection function, the short small spikes in the detection function; smoothing works like a lawn mower, reducing the height of the grass. It also has the benefit of making the detection function not jump around as much during a call, so that the function often goes over the threshold only once during a vocalization.



Sharpening

For odontocete echolocation clicks and other short-duration impulsive sounds, it's sometimes helpful to sharpen the detection function using a mathematical method called the Teager-Kaiser operator. Load **Ishmael Factory Settings.ipf** to reset your Ishmael settings, then open **s010723-141142L-onewhale.wav**. Make a spectrogram with a short frame size, like 256 or 512 samples, since the sounds to detect are short, impulsive clicks. Adjust brightness and contrast to get a good spectrogram with a fairly dark background. Set up an energy detector covering the frequency range of the clicks and run it to see the resulting detection

function. Now try opening the **Detect->Detection options->Smoothing/Sharpening (tab)** and clicking **Enable Teager-Kaiser operator**. Notice how the spikes in the detection function become taller, standing out better from the "grass" at the bottom.

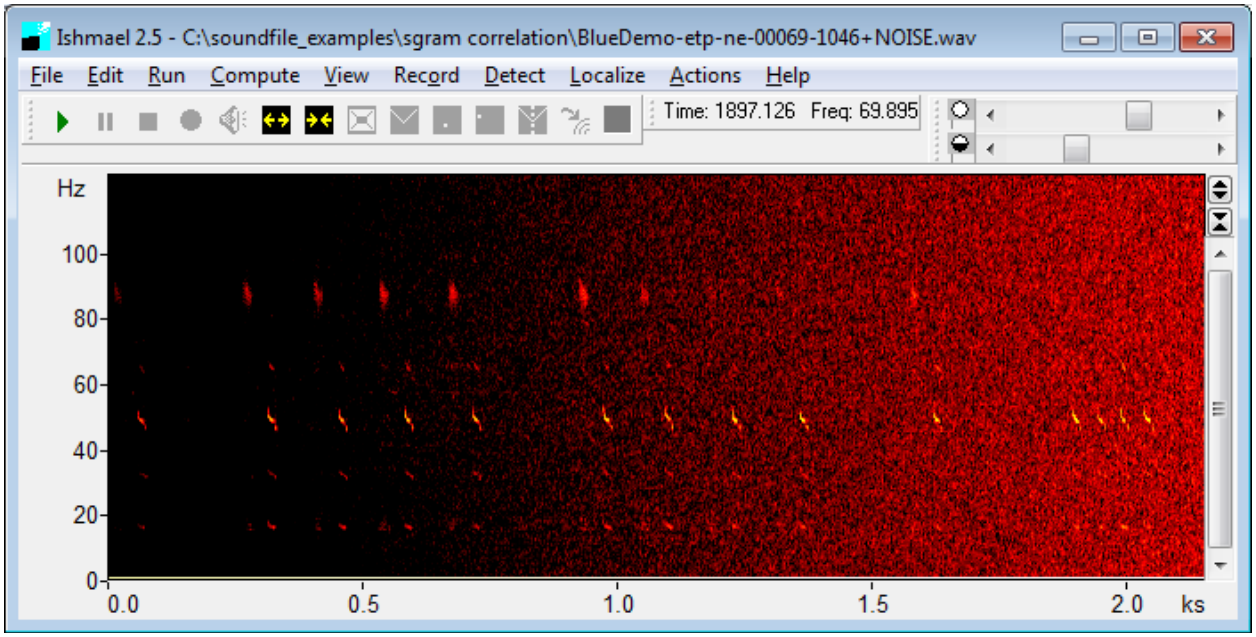
Generally for odontocete clicks the T-K operator improves a detection function, but for other call types, it's hard to know whether will make things better. The best thing is to try it and see if it improves the height of your detection peaks above the noise.

Spectrogram correlation

Spectrogram correlation is a detection method that's useful for relatively stereotyped vocalizations made up of frequency contours. The example here is with blue whale "B" calls, but you should be able to adapt it for other frequency-contour calls as well.

1) Load **Ishmael Factory Settings.ipf** to reset your Ishmael settings, then open the file **BlueDemo-etp-ne-00069-1046+NOISE.wav**. While the "Open a sound file" dialog box is open, click the "Time stamps" tab, click the "Is this date and time" button, and then enter 0:00:00 in the Time field. The Date field doesn't matter for now**.**

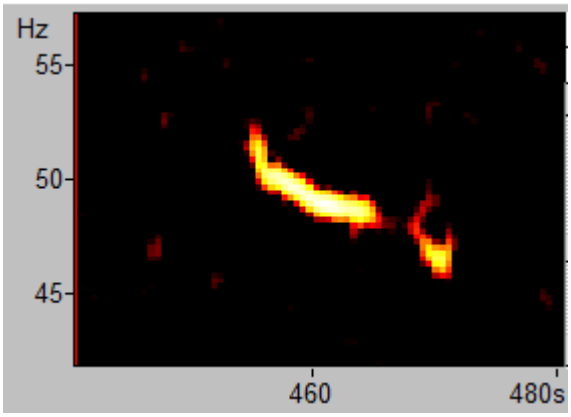
2) Make a spectrogram. Fiddle with spectrogram parameters using **Compute→Spectrogram parameters** (try 512 samples, no padding, hop size 1/4x) until you can see the first calls clearly:



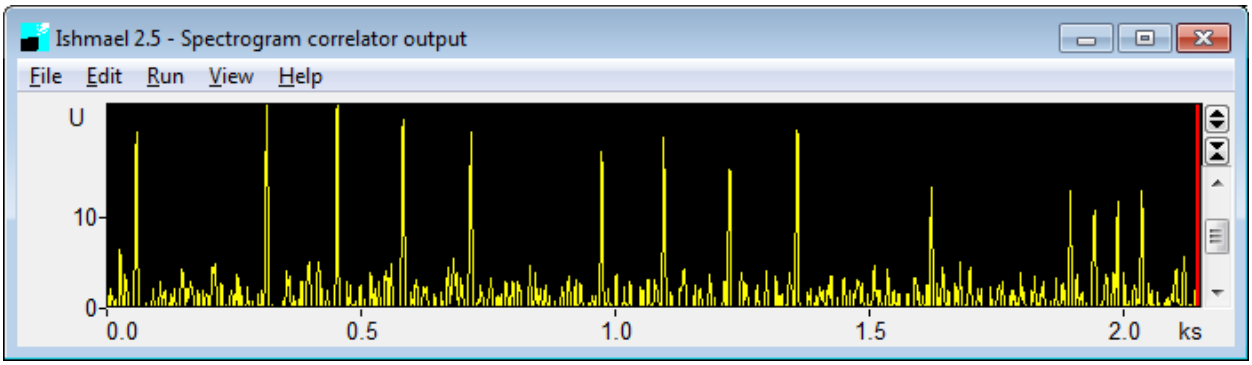
3) Enable spectrogram correlation, enter a **Contour width** of 2 Hz, and then on the **Define contour** tab, enter these numbers in the box:

Start time	End time	Start freq	End freq
0	9	51	48.5

This means that the detector will look for frequency contours of the swept parts of these blue whale AB calls that are 9 seconds long (i.e., starting at time = 0 and ending at time = 9) and extend from 51 to 48.5 Hz. Where did these parameters come from? We looked at, and measured, a zoomed-in spectrogram of a call. If you have MATLAB, Osprey (see the Filtering section above) is a useful tool for making these measurements; you can also do it in Ishmael by making a spectrogram like the one below and then making measurement at the start and end of the contour and examining the time/frequency values.

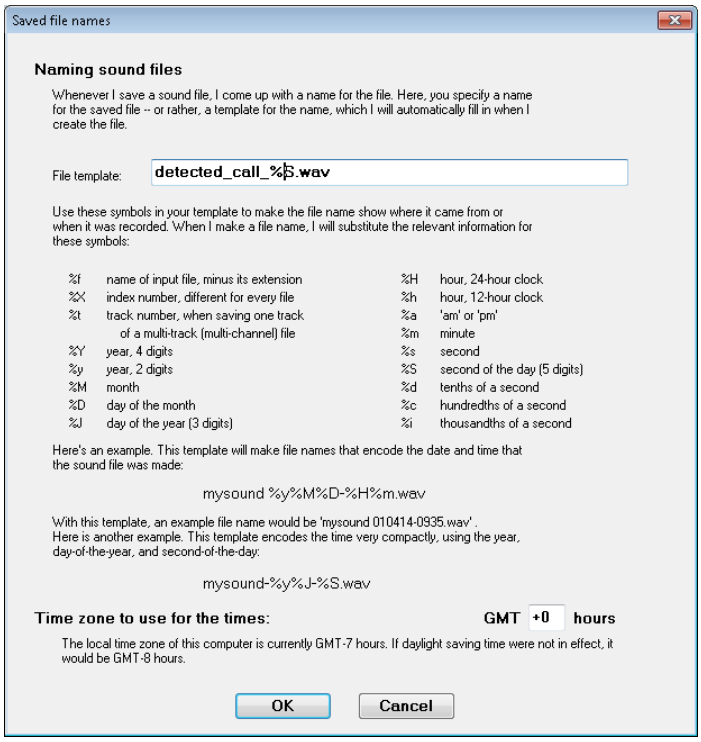


4) Click **Run** on the menu bar or the **Run** button (the green arrow under the menu bar) to start the detector. The vertical scaling in the detection window will probably be wrong; fiddle with it using the scrollbar and the and buttons at right until you get something like this**:**

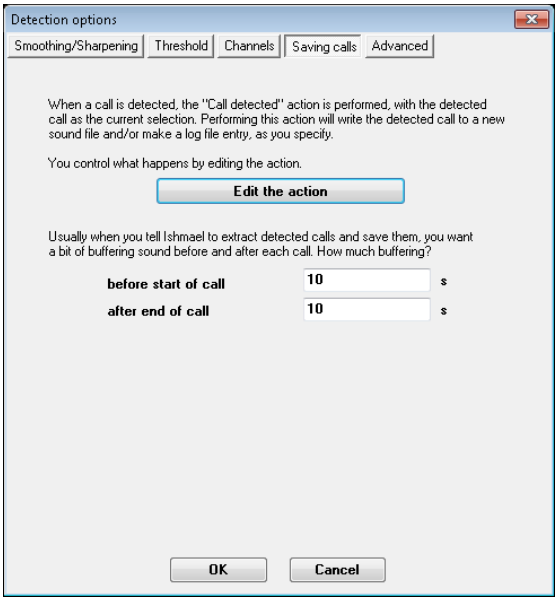


5) Set a threshold – choose a good value by examining the Y-axis of the detection function and choosing a value that's above the "grass" and below the peaks.

6) Set up saving detections as separate sound files. This time start at **Detect→Detection options→Saving calls (tab)→Edit the action**, then check the box labeled **Save the selection as a new sound file**. Then on the **Saving files** tab, first click the **Choose directory** button and decide what folder to save files in. Next, click the **Set file names** button and enter the template **detected_call_%S.wav**. This will make the times at which the calls occur, in seconds (that's what %S does), part of the saved file names. Click OK.



7) Back on the **Saving calls** tab, specify the amount of time before and after a detection to save in sound files. Try 10 s for each**.**



8) Click **Run** on the menu bar or the **Run** button (the green arrow under the menu bar). Ishmael will detect calls and write a separate sound file for each into the folder you specified.

9) Examine the new sound files! Start up a new copy of Ishmael, do **File→Open**, navigate to the folder where you saved detected calls, and select ALL of the saved calls. They should have names like **detected_call_00047.wav** and **detected_call_00302.wav**. This time, instead of clicking the **Run** button, choose **Run→Batch run** from the menu – this will make Ishmael open each file in turn. Check the box labeled **Pause after each file** or it will flash through all of them too fast to see.

10) You'll notice that the detected calls aren't centered in the sound file. This is because with spectrogram correlation, the detection function has a peak near the start of the call, so it's the *start* of the call that is centered in the sound file. You can fix this in the copy of Ishmael running the detector by changing the buffer times before the start of the call and after the end of the call in **Detect→Detection options→Saving calls (tab)**. Try 5 s before the call and 20 s after. Fix this and re-run Ishmael, then check your new detected sound files.

11) You can also record these detections to a log file. Choose **Action→Open a log file**, turn on logging, and specify a log file name. Then do **Actions->Edit actions**, select the **Call detected** action and click **Edit**, select the **Logging** tab, and specify what you want to write to the log file. At a minimum, you want the time of the detection – either the **current selection's time and frequency** (the selection in this case is the detected call) or **time and height of detection peak.**

12) See Appendix 2 for information on grouping successive detections together. This is useful when you want to consider something like a pulse train, for which Ishmael might make long series of detections, as a single unit in your analysis.

13) Now you try it – open the file **upcalls.wav** and make a spectrogram correlation detector for the right whale calls in it. We'll work through this example together.

14) Finally, load **BlueDemo-etp-ne-00069-1046.wav** and try building a detector for these blue whale calls on your own.

Matched filter detector

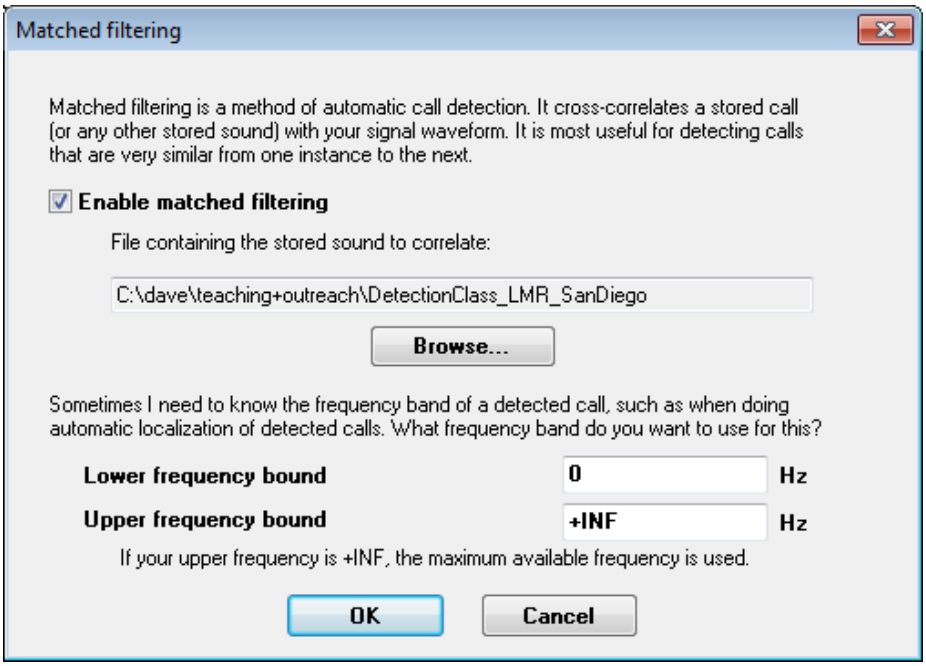
A matched filter works by cross-correlating Ishmael's input signal with another signal that you specify, called the *kernel.* Cross-correlation is the optimum method for detecting a known signal in white Gaussian noise (Van Trees, 1968). It works best when the signal you wish to detect is quite constant from one instance to the next and from one animal to the next. Unlike the other detectors in Ishmael, it uses the signal waveform (time series), not the spectrogram, so your spectrogram settings don't matter at all. The flip side of this is that the kernel sample rate must match the sample rate of the input file (or real-time sound).

1) Load the settings file **Ishmael Factory Settings.ipf** to clear any previous settings.

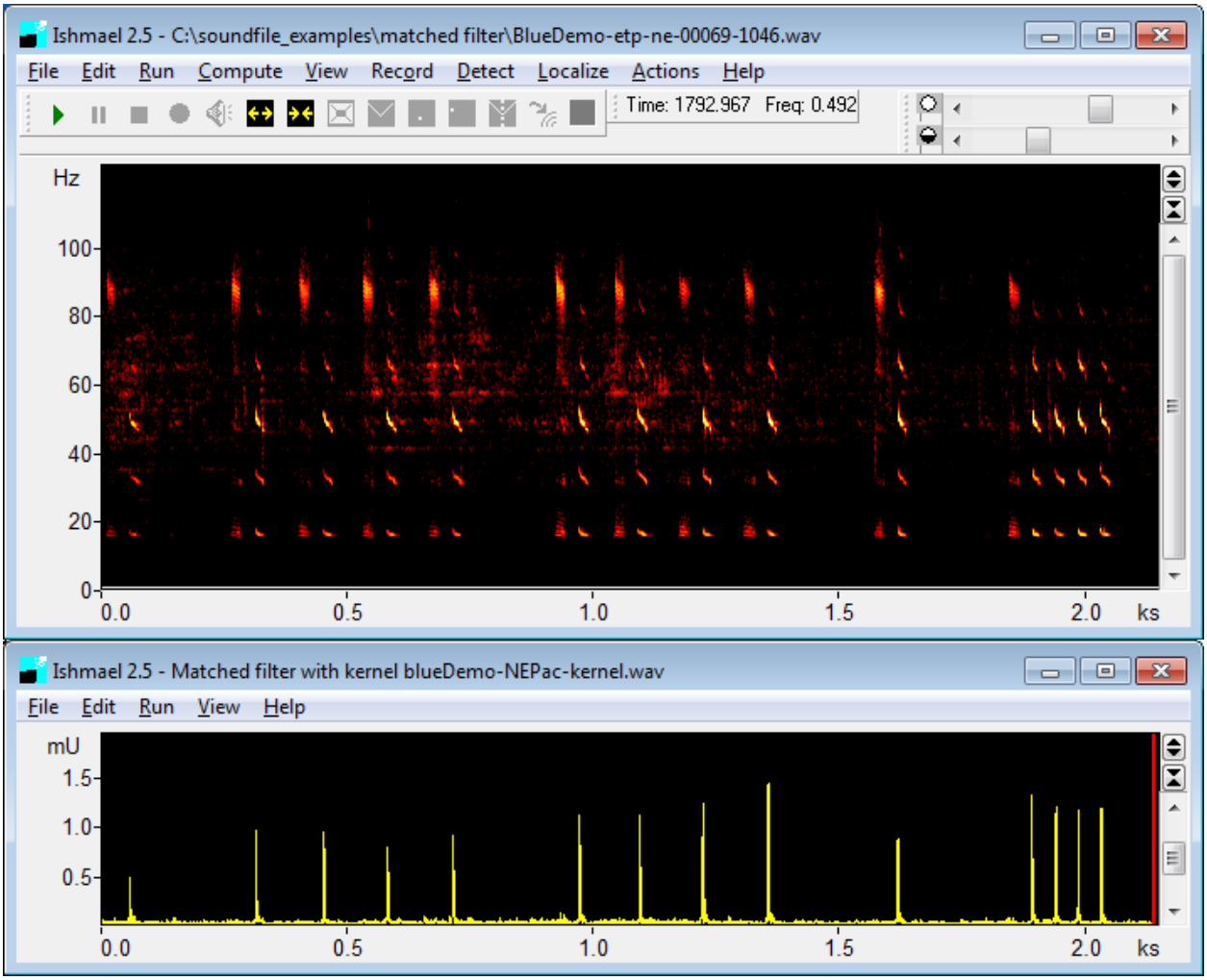
2) Open the file **matched filter/BlueWhale-etp-ne-00069-1046.wav**. Make a spectrogram so you can see what's going on; this spectrogram isn't used by the detector, but it's nice to see the processing happen as you run the matched filter detector

3) Choose **Detect→Matched filter** from the Detection menu. Check the **Enable matched filtering** box, at which point Ishmael will ask you for a sound file to use as the kernel. Select **blueWhale-NEPac-kernel.wav**. The lower and upper frequency bound aren't used at this point, so you don't need to bother with them

Check the **Enable matched filtering** box, at which point Ishmael will ask you for a sound file to use as the kernel. Select **blueWhale-NEPac-kernel.wav**. The lower and upper frequency bound aren't used at this point, so you don't need to bother with them.



4) Click OK and then click **Run** (or the green **Run** button). You'll see a detection function, which may look like a flat line; click the expand button on its right ( ) a lot of times until the detection function starts to show some detail. By expanding the graph and maybe using the vertical scroll bar, you should get something like this:



The result is nice tall peaks in the detection function where the calls are that extend far above the rest of the detection function. This makes it easy for a detection threshold to separate peaks (calls) from non-peaks (noise). To set a detection threshold, open **Detect->Detection options->Threshold (tab)** and enter a detection threshold of 0.0003. (You need those three zeroes in front of the 3 because the detection function has units of "mU", meaning they're scaled by one-thousandth.) The detection peaks exceed this threshold and thus will be counted as detections.

5) **About matched filter kernels:** Matched filtering is typically done with either a natural or a synthetic kernel. To use a natural signal, find the clearest, most typical example you can of the call you wish to search for. The less noise is with it, and the louder the call is above background noise, the better the matched filter will work. You can find a call using Ishmael's spectrogram display, or any other method you like. Select the call, making the selection just long enough to contain the call and nothing more. Choose **File->Save selection as** and save the call as a sound file. It's helpful to label this file with the word "kernel" in the name so later you know what it is.

The other method way to do it is using a synthetic kernel. The advantage of this is that a synthetic sound can have no noise whatsoever, but the disadvantage is that the sound might not be exactly the same as the call made by your target species.

6) Now you try it. Load the settings file **Ishmael Factory Settings.ipf** to clear any previous settings. Open the file **Zc06_204a22410-24210@500s.wav**, which has clicks from Cuvier's beaked whales (*Zc*) in it, set up the spectrogram so you can see the clicks, and run a matched filter on it using the kernel **CuviersDetector-MatchedFiltKernel-v3.wav**. Scale your detection function so you can easily see its peaks.

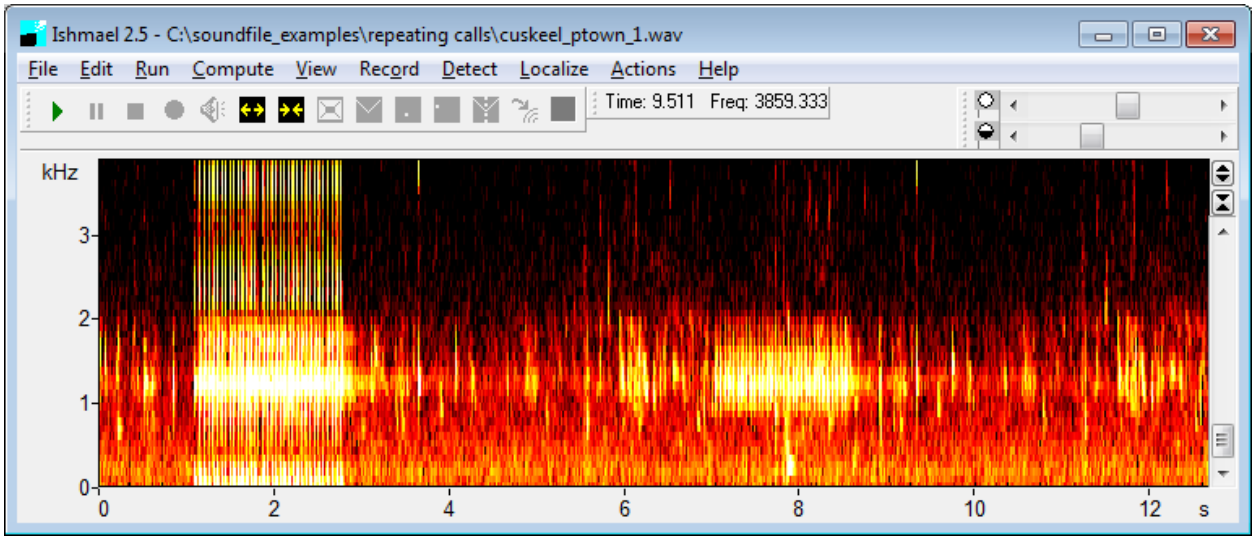
Note: Constructing synthetic kernels of animal calls (e.g., Dooling *et al.,* 1982; Buck *et al.,* 2000) is beyond the scope of this discussion. If you decide to do it, construct your synthetic kernel at the **same sample rate** as the sound signals you wish to analyze, then save it as a sound file.

Repetitive call example: detecting cusk eel pulse trains

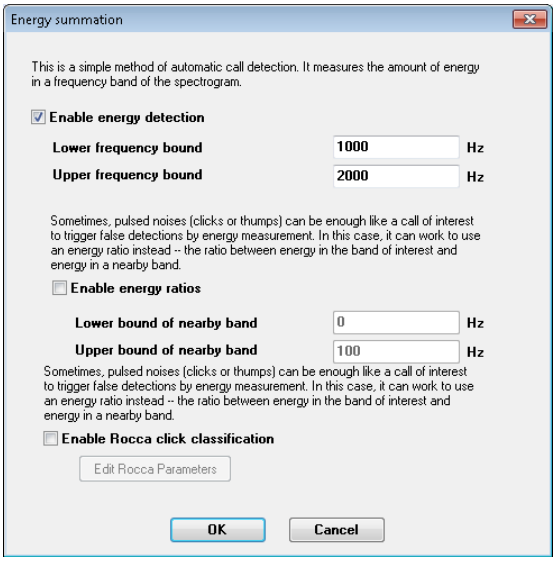
This example shows a method for detecting calls that occur with a high degree of regularity. Many animals, from whales to fish to birds to insects, produce calls at very regular intervals, and this regularity is a useful feature for detecting such calls. The method shown here *operates on the detection function* produced by another detection method – in this case, energy summation – to produce a secondary detection function to which a threshold is then applied.

1) Load the settings file **Ishmael Factory Settings.ipf** to clear any previous settings, then open the file **cuskeel_ptown_1.wav** via **File→Open file**.

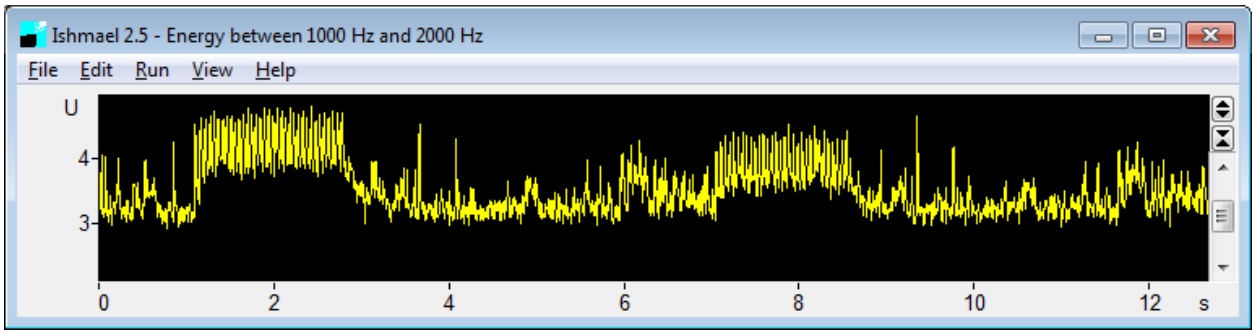
2) Make a spectrogram of the sound, using a short enough frame size (say 512 samples) that you can see individual pulses. Zoom in vertically, since the main part of the calls is below 3 kHz:



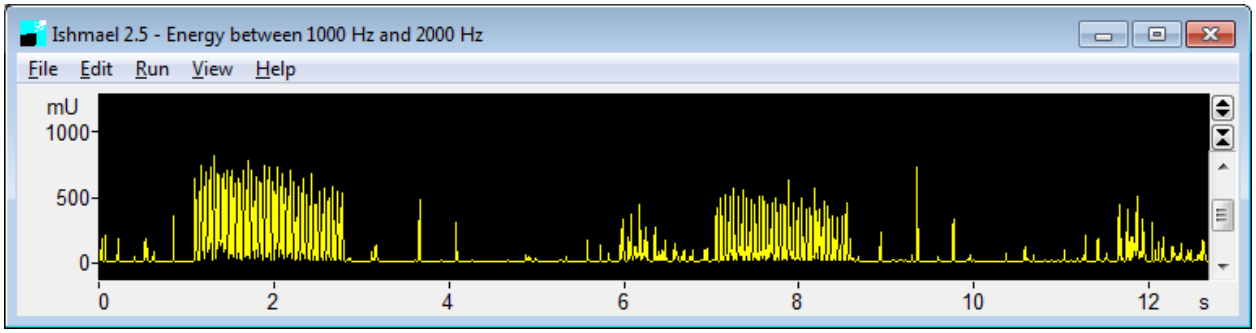
3) Turn on energy detection and pick a suitable frequency range for detecting the pulses. Notice that most of the energy in these pulses is between 1000 Hz and 1700 Hz, so try using these settings for the lower and upper frequency bounds:



After adjusting the vertical span of the detection function, you should get something like this:

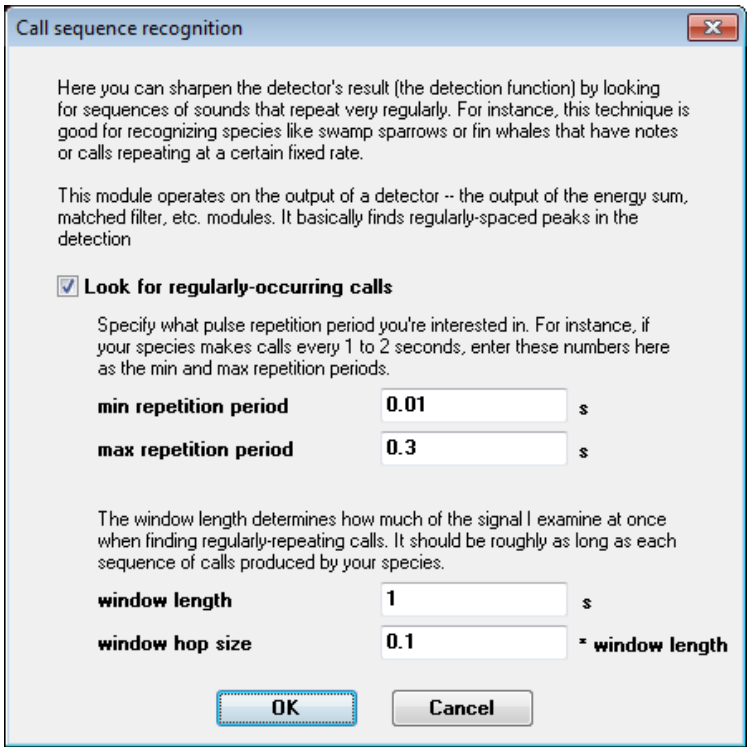


4) Note that the time with no detections does not go down to zero. To adjust this, turn equalization on with a time constant of 3 s, check **Use a spectrogram floor value** (with the **automatic** option), and re-run. If you get a black box, adjust the spectrogram brightness to make it lighter. The result should look something like this:

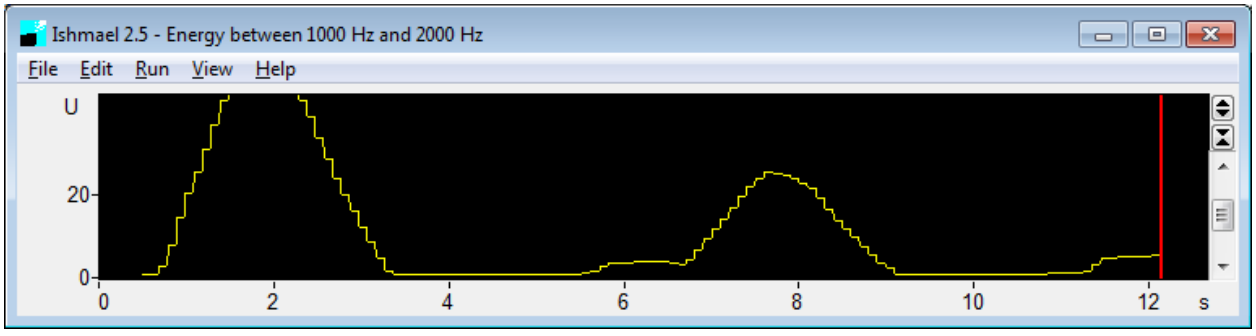


The two sets of pulses at 1-3 s and 7-9 s are clearly visible in the detection function, but the detection function also includes false detections in several places.

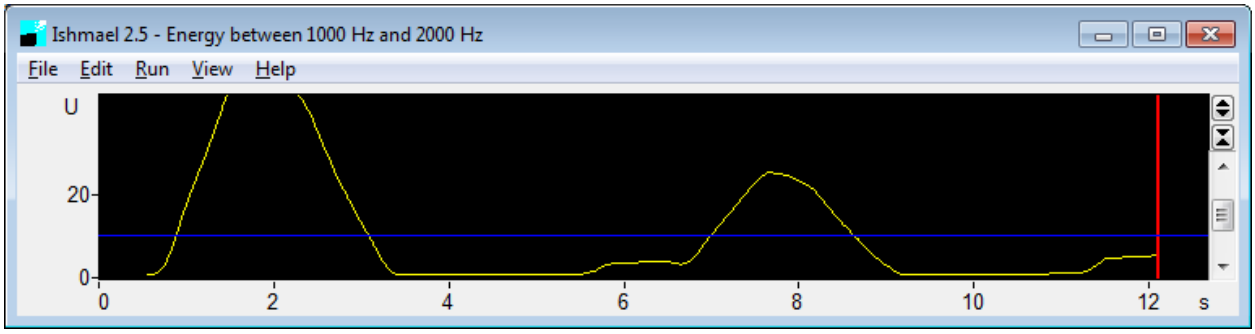
5) To fix this problem, try regular sequence detection. Choose **Detect→Regular sequences**, check **Enable**, and enter min/max repetition periods of 0.01 to 0.30 s. These limits are meant to cover the range of inter-pulse intervals (IPIs) possible for this fish's pulse trains. The pulse trains here have an IPI of about 0.05 s. Set the **window length** to 1 s – about the length of a pulse sequence – and leave the **window hop size** at 0.1.



6) Click **Run** on the menu bar or the **Run** button (the green arrow under the menu bar). Adjust the detection function window so you can see much of the resulting detection function – something like this:



The two clear pulse trains have detection function peaks far above the rest of the detection function, so it's now easy to set a threshold that detected but doesn't detect the background noises. You can also **turn smoothing on** (0.1s) to smooth the detection function.

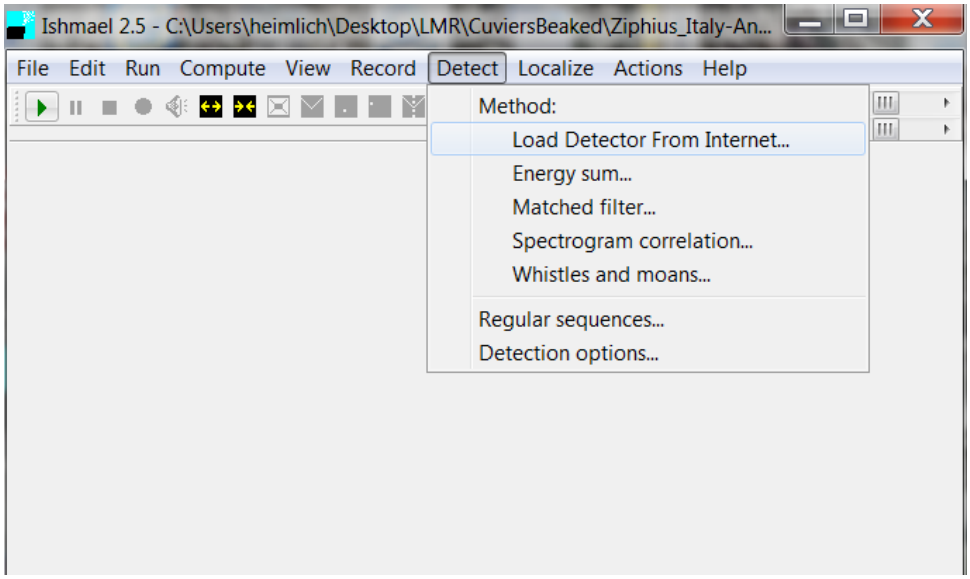


7) Now you try it. Load **Minke-93-001-2321.ch13.wav** and try constructing a detector for the pulse-train sounds present in this file. First turn off regular sequence detection if it's still on from the previous exercise. Adjust the spectrogram settings so that you can see individual pulses, which will require using a relatively short frame size so they don't smear together. Next set up a detector – an energy detector will work well for these pulses – that detects individual pulses. You should see the pulses appear as a series of peaks in the detection function; note their repetition rate. Then turn on regular sequence detection, using values for the minimum and maximum repetition period that bracket the repetition rate.

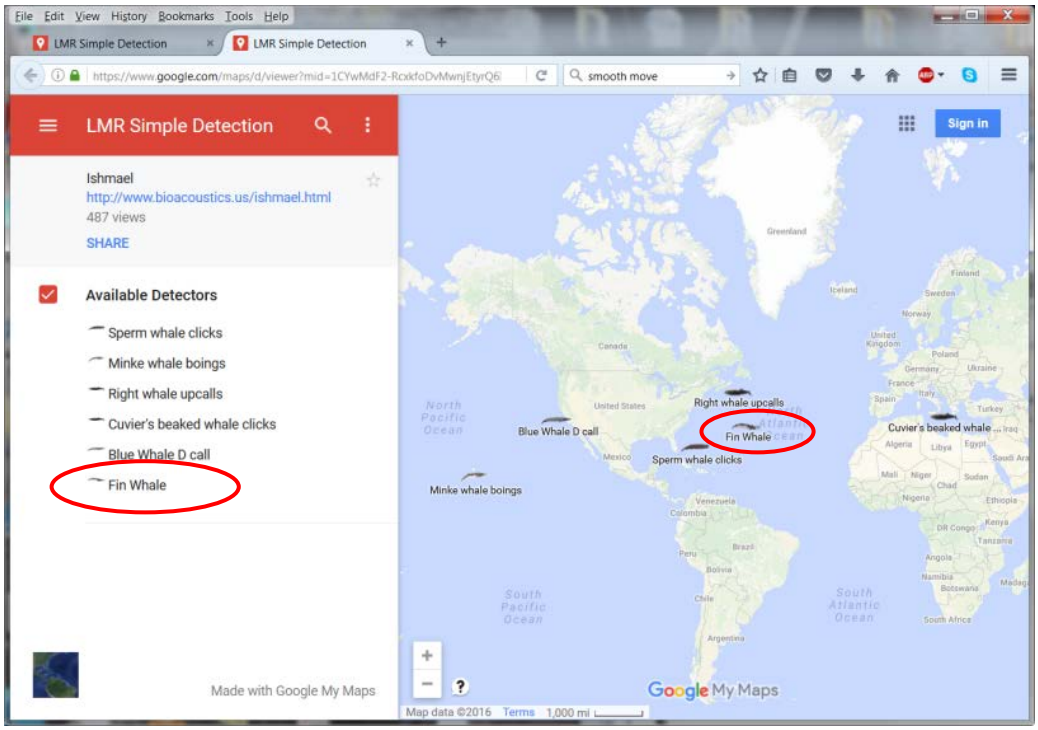
8) Finally, open **MAR-Feb99-CE-disk1-airgun-datafile.510@9999s.wav** and try building a detector for these airgun sounds on your own. The airgun sounds start around 1500 s into the file (note that Ishmael's X-axis is probably in "ks", or kiloseconds – thousands of seconds).

Downloading detectors from the internet

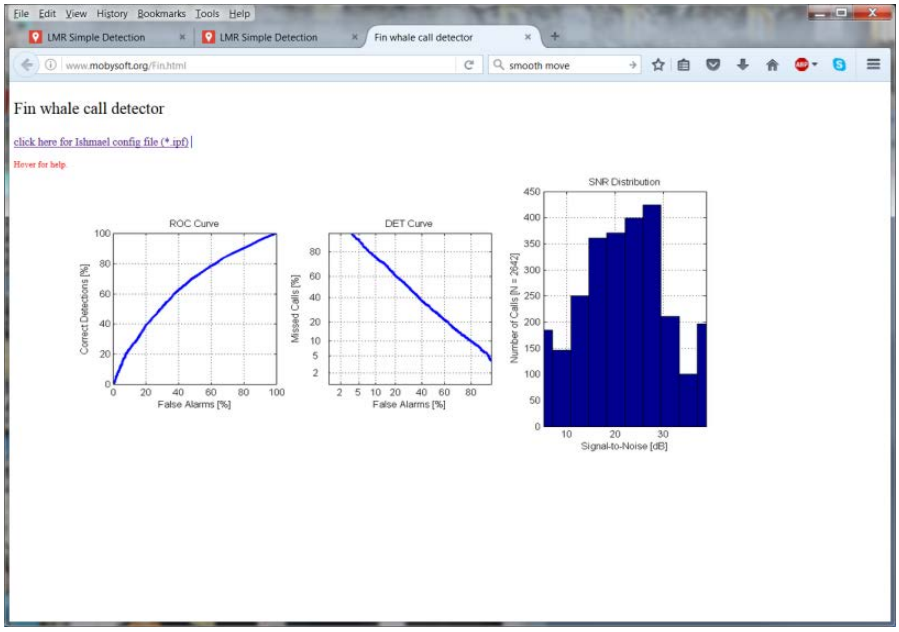
1) Select **Detect->Load Detector from Internet**:



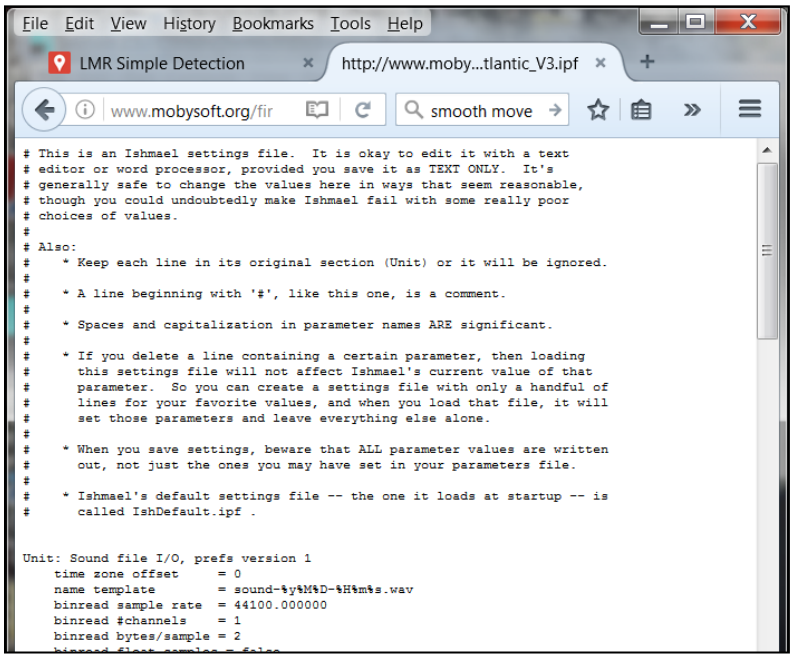
This opens a new browser window with a Google world map which identifies the locations and species of available detectors to download; the same detectors are also presented in list form. Detectors can be downloaded by selecting from the map or from the list.



2) Try choosing the Fin whale detector. Selecting from either option will result in a new browser window that shows just the detector of interest, with a link to the performance evaluation for that detector.



3) To see the code content of the detector, click once on the link "click here for Ishmael config.file (*ipf)". This will refresh the browser page to show the full code.



4) To download the detector, right click on this new page and save as a text file with an ipf extension. The *.ipf should be in your Downloads folder. From there, you can place it wherever is the most practical for your purposes.

5) Once you have the *.ipf file on your computer, you have two options. The first is recommended. Drag and drop it into a running copy of Ishmael. This will keep most settings (file name, channel details, etc.) and only add settings associated with the detector. You can also double click on the *.ipf file which opens a completely new version of Ishmael, but settings may not have the values you want.

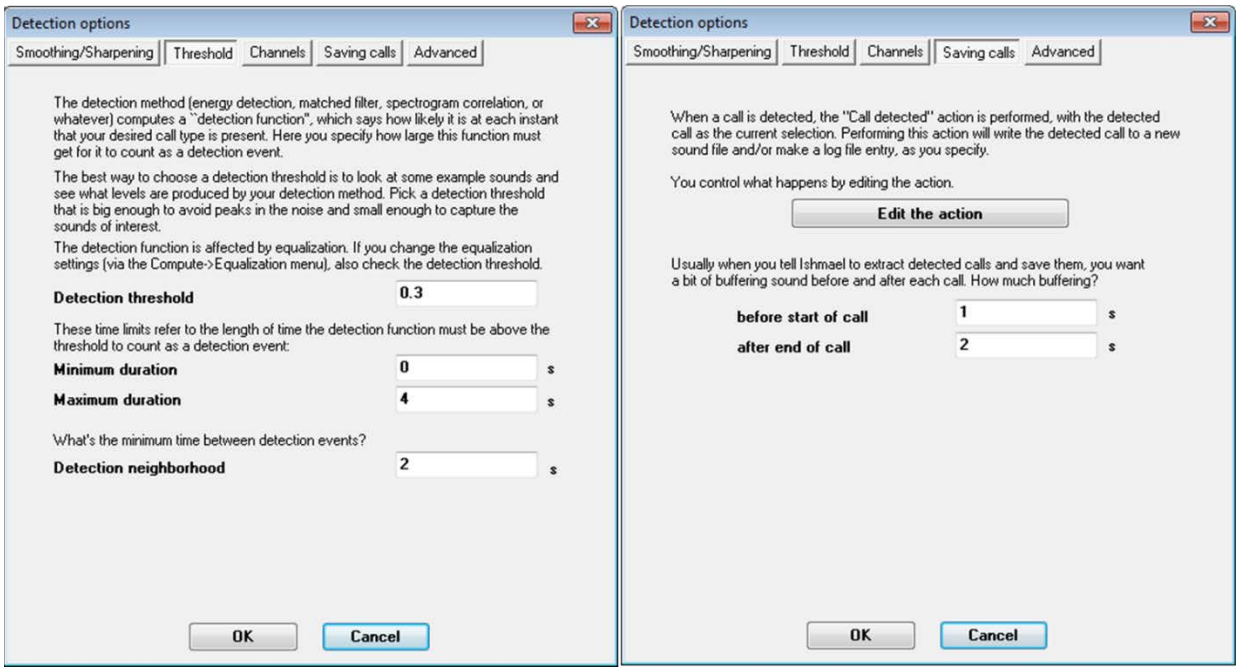
6) Now you try it. Download 10 or so minke whale boing wav files from the folder **soundfile_examples\minke_DCL5_locT123**. Next, download the minke whale boing detector from the LMR Simple Detection web page, and try setting up the detector to run on your boing wave files.

Manual verification of detections

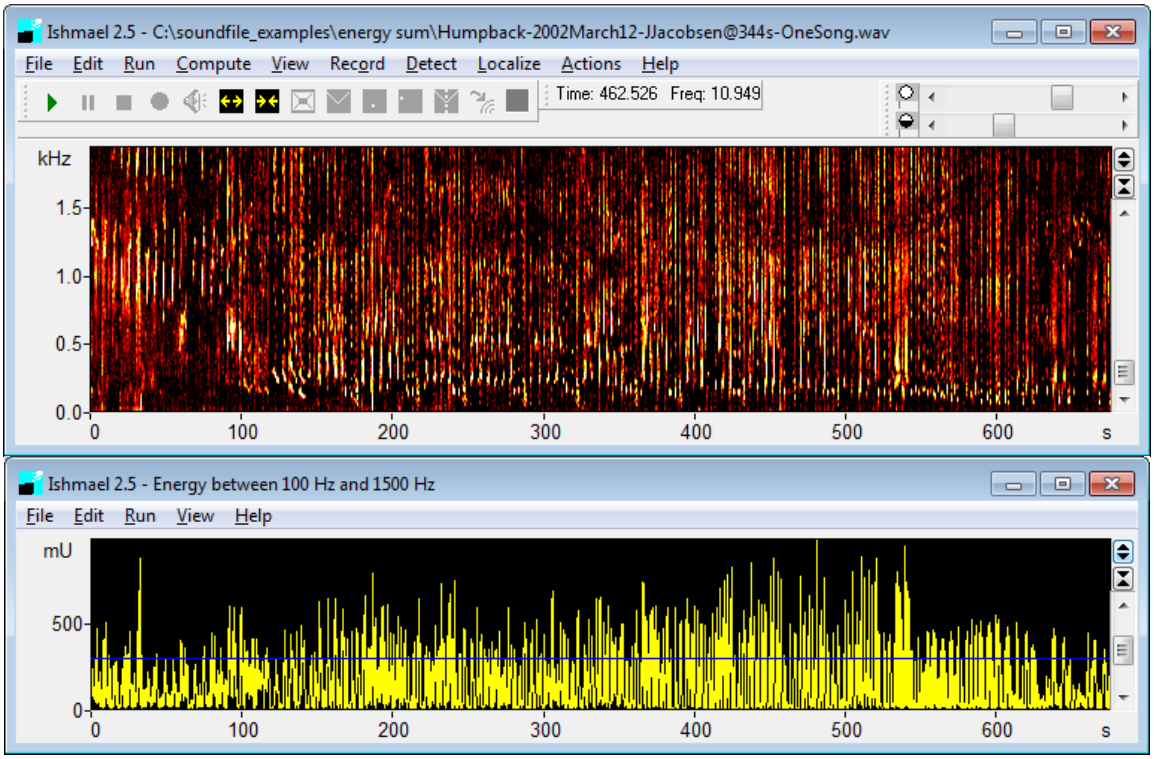
Once Ishmael has detected a bunch of sounds, you can manually check them to see whether they are correct or not. As an example, we'll look at the humpback sounds we detected above, using a detector that makes some mistakes.

1) Load the settings file **Ishmael Factory Settings.ipf** to clear any previous settings, then open the file **Humpback-2002March12-JJacobsen@344s-OneSong.wav** via **File→Open file**.

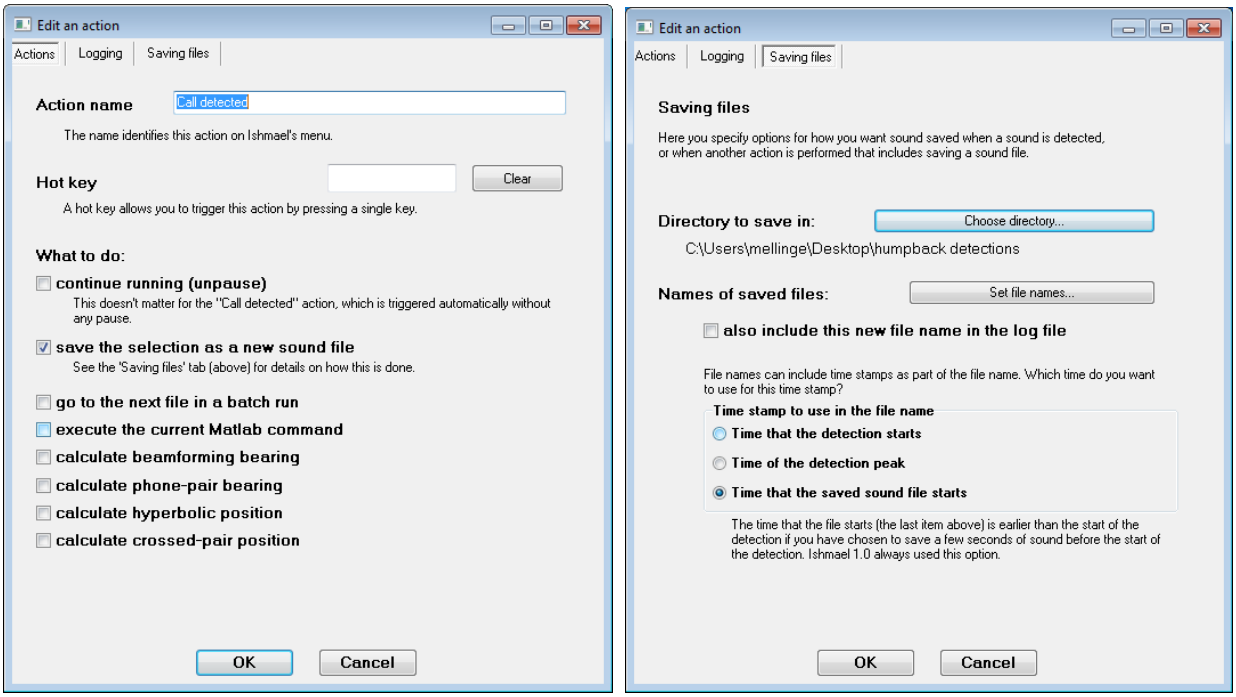
2) As before, make a spectrogram so you can clearly see the humpback's vocalizations, adjusting the frame length, brightness and contrast, time scaling, and frequency scaling. Turn on equalization (**Compute->Equalize 'gram**) and the **Use a spectrogram floor value**; the calls seem to be 1-2 seconds long, so try an equalization time constant that's about 4 times as long as that. Then turn on the **Energy sum** detector using frequency bounds that cover either the whole frequency band of these sounds, or perhaps just the lower portion of them. Adjust the vertical scaling of detection function display. Set a threshold for the detection function so that the peaks in the detection function go over it, and set your maximum call duration to something longer than the longest call, like 4 s. In **Detect->Detection options->Saving calls (tab)**, set the **time before start of the call** to 1 s and the **time after end of call** to 2 s:



You should have something roughly like this:



3) Now set it up to save each detection in a separate file. On **Detect->Detection options->Saving calls (tab)**, click the button labeled **Edit the action**. In the ensuing dialog box, on the **Actions** tab, check the box for **Save the selection as a new sound file**. Over on the **Saving files** tab, choose a directory (folder) to deposit the new files in; this can be any folder of your choosing.

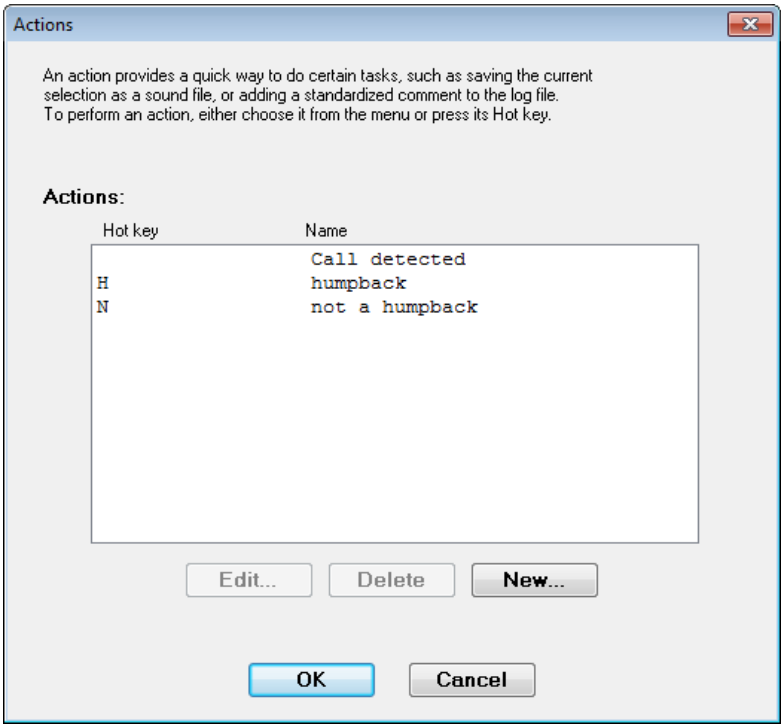


4) Click **Set file names** and use the name "humpback detection %S.wav" as the **File template**. The "%S" indicates that each sound file will have the number of seconds since the start o fthe sound file that the detection occurred. See Appendix 1 for information about getting time stamps from your input file into Ishmael.

5) Click OK and then click **Run** (or the green **Run** button) to detect the humpback sounds. If you look in the folder where you told Ishmael to put detected sounds, you should see a large number of small files.

6) Now set up a way to manually check these files to see whether or not they're actually humpback sounds. You do this by examining the files in a **Batch run** and manually indicating whether or not each one is a humpback.

First, start a new copy of Ishmael and load **Ishmael Factory Settings.ipf** to clear any previous settings. Create a new Action via **Actions->Edit actions->New**. In the **Action name** box, enter "humpback", and in the **Hot key** box, type an H. Check the box for **go to next file in a batch run**. On the **Logging** tab, check the boxes for **this action's name** and **input sound file name**. Click OK; your new action should show up underneath **Call detected** in the list of actions. Click **New** again, and name this action "not a humpback", give it N for its **Hot key**, and check **go to next file in a batch run**. Again, on the **Logging** tab, check the boxes for **this action's name** and **input sound file name** and click OK.



These two new Actions will allow you to look at a sound file with a detection and decide whether it's a humpback or not, then press one key – either H or N – to indicate which it is. Ishmael will record your answer in the log file and go on to the next sound file.

7) To process the sound files, do **File->Open**, navigate to the folder where the detected sounds are, select *all* of them (an easy way to do this is to click on the first file, scroll down to the last file, and hold down the Shift key while clicking on the last file) and click OK. Uncheck **Fit file to window … in frequency**; doing so will allow you to adjust the frequency scaling to your liking without having it reset with each file you look at. In the Open a log file using **Actions->Open log file,** making sure **Use a log file** at the top is checked. Now run the files, but instead of clicking **Run**, choose **Run->Batch run** from the menu. Ensure that **Pause after each file** is checked and click OK.

8) You should see the first sound file that was detected. Adjust its spectrogram parameters to your liking (you can probably use the settings you used for the detection above in step 2), including stretching the spectrogram vertically so you can see the low-frequency portion of the sound better.

9) The sound that was detected should start at around 1 second. This is because you specified that the saved sound files should have 1 s of time before the start of the call (above, in step 2)

10) If you think it's a humpback, press the H key (lowercase – don't hold down Shift). If you don't think it's a humpback, press the N key. Ishmael will write a log file entry with your choice and display the next file. By pressing H or N for each one, you can fairly quickly skip through the entire set of detections.

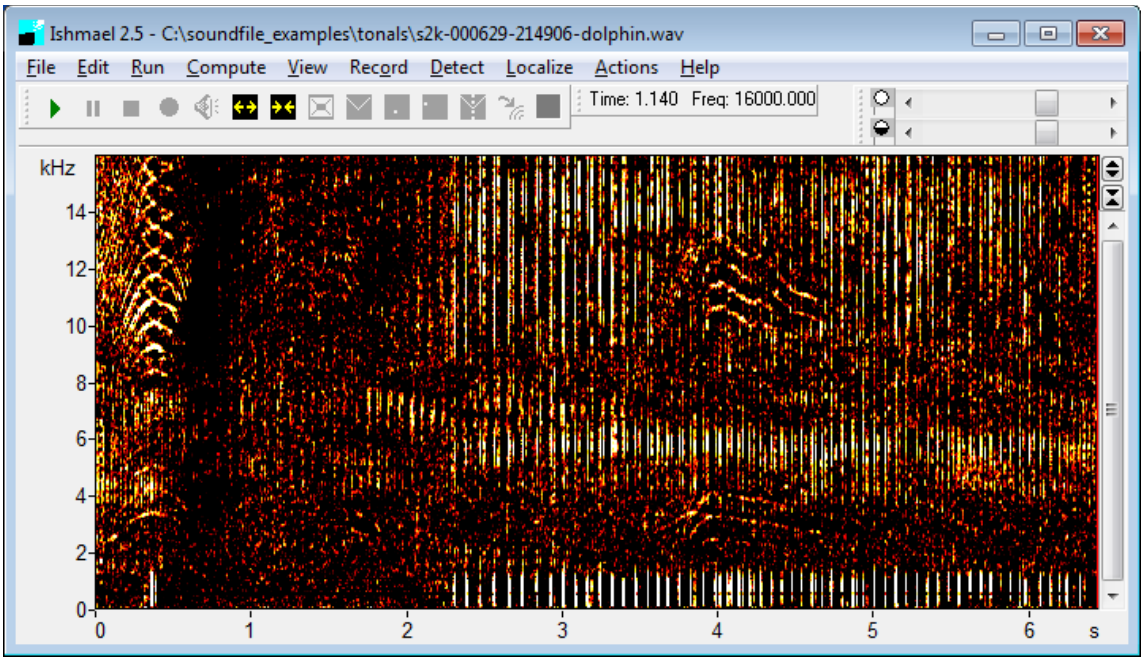
11) Open your log file in (for a text log) Wordpad, Notepad, Word, or (for a .csv log) Excel. Files are labeled as to whether they contained humpback sounds or not. In Excel, you can sort the results to get the set of humpback sounds from the set of non-humpback sounds if you like.

Whistle and moan detection: tonal sounds

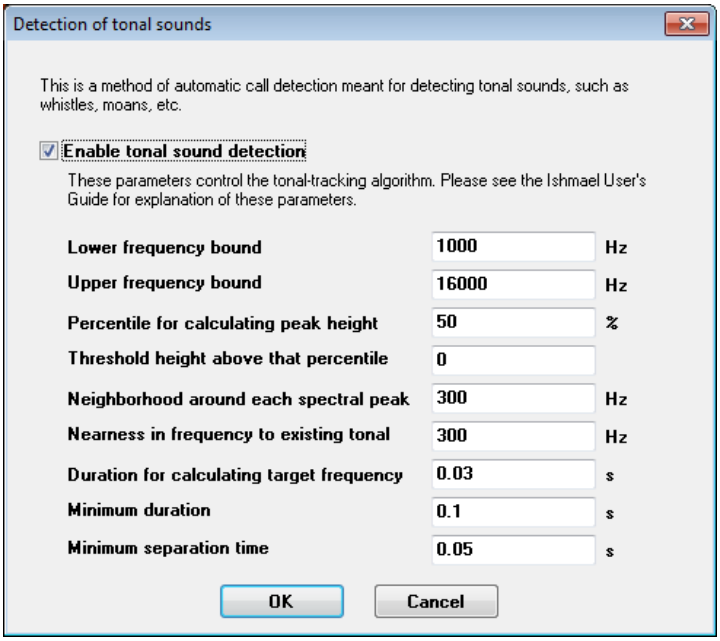
Many marine mammals make sounds that appear as tone-like ("tonal") frequency contours in a spectrogram. Examples include whistles of many species of dolphins and whales, some humpback song units that sound like moans, and portions of minke whale boings. Ishmael can detect these using its **Whistle and moan** detector. This is far and away the most complex detector in Ishmael, so it will take some explaining here!

1) Load the settings file **Ishmael Factory Settings.ipf** to clear any previous settings, then open the file **s2k-000629-214906-dolphin.wav** via **File→Open file**.

2) Make a spectrogram of the sound. I found that a frame size of 512 samples, 1x zero-padding, and 1/4x hop size worked well. Turn on equalization (**Compute->Equalize 'gram**) with a time constant of 0.2 s, and turn on **Use a spectrogram floor value** (with **automatic** checked). Adjust brightness and contrast so you can see the two burst pulses, at about 0.5 s and 4 s, in this recording. There are also a lot of echolocation clicks, which are the vertical stripes:

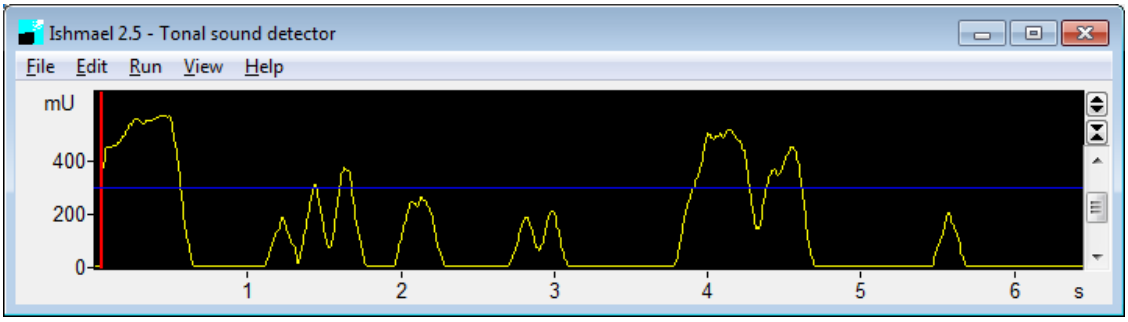


3) The whistle and moan detector is a fairly complex process, requiring a lot of parameters. Check **Enable tonal sound detection** and enter these values into the dialog box:



Also do **Detect->Detection options->Smoothing/Sharpening (tab)** and turn on **Smoothing**, with a time constant of 0.1 s.

4) You should get something like this:



Notice this has two high peaks where the burst pulse sounds are (~0.5 s and ~4 s). You may also have peaks just over threshold, as this graph does, at 1.5 and 1.7 s. If you look closely at the spectrogram, you can just barely see that there are indeed whistle-like (or burst-pulse-like) sounds at these times, in the range of 11-12 kHz. I didn't notice these sounds until Ishmael pointed them out. If you don't want to detect such faint whistle-like sounds, raise the threshold a bit.

5) How should all those parameters be set?

- **Lower** and **Upper frequency bound** determine where in the spectrogram Ishmael will look for whistles and moans.
- **Percentile for calculating peak height** controls how much of a given spectrogram time-slice gets considered when Ishmael is searching for a whistle or moan. You don't often have to change this number, but if Ishmael isn't finding your whistles or moans, you might need to *lower* it.
- **Threshold above that percentile** controls how loud a whistle or moan has to be relative to background noise for Ishmael to consider it. You don't often have to change this number, but if Ishmael isn't finding some of your whistles or moans, you might need to *lower* it.
- **Neighborhood around each spectral peak** says how far away a peak must be from another peak in a spectrogram time-slice that occurs at the same time. Set this to roughly the width, in Hz, of the whistle or moan you want to detect. For instance, the dolphin whistles look to me like they're a little fatter than about 1/10 the distance between two frequency tick marks at the left; since these tick marks are 2 kHz apart, 1/10 of that would be 200 Hz, and a little more than that is 300 Hz.
  This number needs adjusting depending on the frequency of your whistle or moan: if you have Ishmael set up to detect baleen whale moans and then switch to dolphin whistles, you'll need to increase the neighborhood substantially. Sometimes you need to *decrease* this number if Ishmael isn't finding your whistles and moans.
- **Nearness in frequency to existing tonal** controls how steeply a whistle or moan can sweep up or down in frequency. It represents how far the center frequency of the whistle/moan can jump from one spectrogram frame to the next. This number also needs adjusting…
  - depending on the frequency of your whistle or moan, as higher frequency sounds require a *larger* value for **nearness**;
  - depending on how quickly your sounds sweep up and down, as faster-sweeping sounds require a *larger* value for **nearness**; and
  - possibly depending on the frame size and hop size of your spectrogram, as larger frame sizes and larger hop sizes may also require a *larger* **nearness** value.
- When Ishmael discovers a whistle or moan, it starts tracking it, and quickly starts estimating how quickly it is moving up or down in frequency. **Duration for calculating target frequency** controls how much time in the recent past Ishmael looks at when estimate where the whistle or moan will go next. Set this to a duration over which your whistle or moan doesn't bend very far. The dolphin whistle detection parameters above used 0.03 s, which is about 1/30 of a second, because dolphin whistles tend to change direction quickly.
- Any whistle or moan that Ishmael registers as a detection must last at least **Minimum duration** seconds. If this number is too long, short whistles and moans won't be detected.
- The **Minimum separation time** controls what happens when two whistles overlap. The have to separate from each other for at least this many seconds before they're considered independent.

6) Now you try it. Download the **Beluga.wav** file from the tonals folder, and try setting up the detector to identify the beluga whale calls in the file.

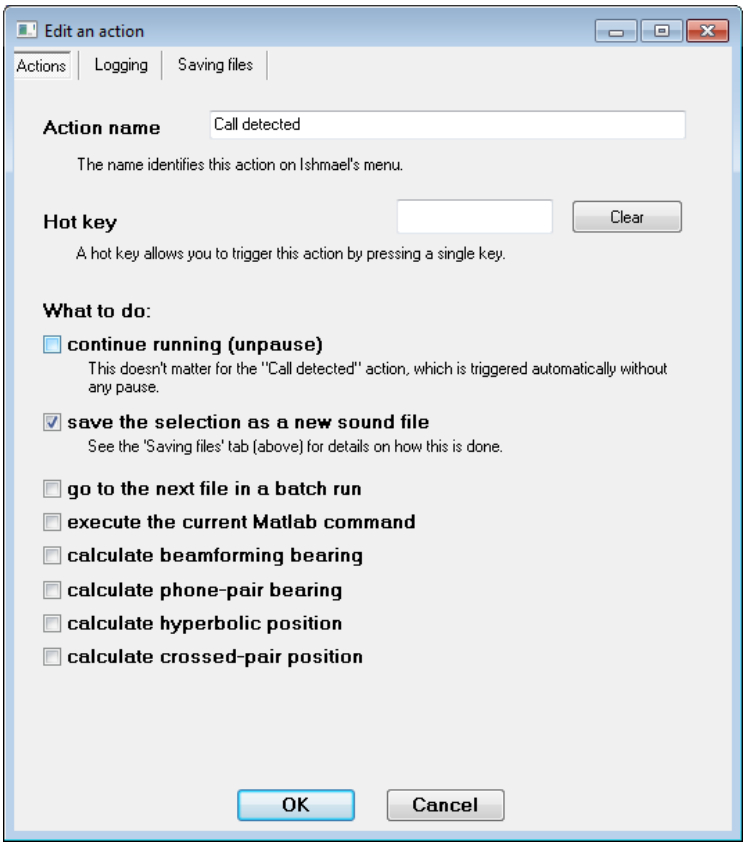
Note: Getting all the parameters right for the whistle and moan detector is admittedly an art. The best way to proceed is to take an existing detector for a sound that resembles your target sound and see if it works. If it's not finding the target sounds, relax the parameters (as specified in the italic words in the above list) and see if you can get Ishmael to start finding them!.

Grouping detections

Sometimes it makes sense to group detections together. For instance, when an animal produces pulse-train vocalizations, like the cusk eel examples above or Atlantic minke whales or screech owls, it makes sense to lump together all the pulses in a train. To do this in Ishmael, do **Detect->Detection options->Saving calls (tab)**. The numbers here for **before start of call** and **after end of call** determine how Ishmael will lump successive detections together. Also, on the **Advanced** tab, you should check the box for **Save successive calls together (allow re-triggering)**. This will make Ishmael try to lump successive closely-spaced detections together.

Lumping detections together can affect Ishmael's output in two ways:

- If you're saving copies of your calls, which you do with **Edit the action** and then checking the box labeled **save the selection as a new sound file**, any sound files that would overlap in time (because of the extra buffer time you specify for **before start of call** and **after end of call**), then these sound files get lumped together into a single sound file.



  There can be more than two detections that get lumped; Ishmael keeps lumping them together as long as the successive saved files would overlap.

- If successive calls would overlap and get lumped as saved sound files, then they are lumped together in Ishmael's log file too. This is true *even if you're not saving sound files*; if the saved sound files would have overlapped, the detections are lumped together in the log.

Performance evaluation for detectors- MATLAB

In this section we show how to make performance evaluation graphs called *Receiver Operating Characteristic* (ROC) curves and *Detection Error Tradeoff* (DET) curves for your detector. "Evaluating a detector's performance" means seeing whether it found the calls in your sound files, and also whether it incorrectly detected things that weren't calls. We'll do this in MATLAB.

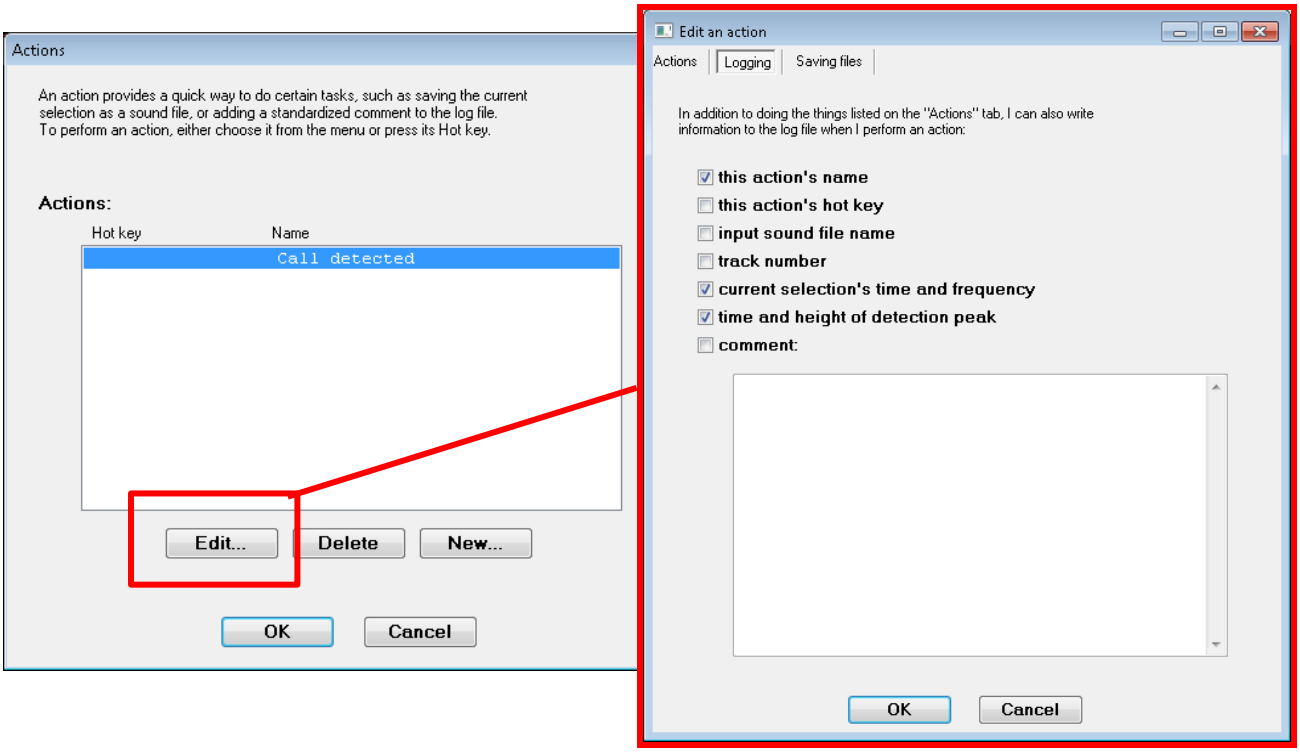
To do this, we have to know the *ground truth*, that is, what really IS a call. This is normally done manually, by having a person scroll through some sound file(s) and pick out the calls in them. In this example, this has already been done for a data set of minke whale calls recorded off Hawaii.

1) In the Windows file browser, find the folder **minke_DCL5_locT123** and look at its contents. There are a bunch of sound files (.wav files), and for each one, a corresponding .txt file. The .txt files have the manually-done *annotations*, also called *labels*, indicating where a person found calls are in each file. Open one and take a look at it just to see what's in it, then close it.

2) There's also a detector for Ishmael, **minkeBoingDetector09-tonal-higher.ipf**, that we're going to evaluate the performance of. Set up that detector on the sound files: Drag **minkeBoingDetector09-tonal-higher.ipf** from the file browser onto Ishmael, which will load it into Ishmael. This tells Ishmael how to run the detection process, but doesn't tell it what to do after it makes a detection. For that, do **Actions->Edit actions**, pick **`Call detected`**, and click **Edit**. On the **Logging** tab, check these items:

- **This action's name**
- **Current selection's time and frequency**
- **Time and height of detection peak**

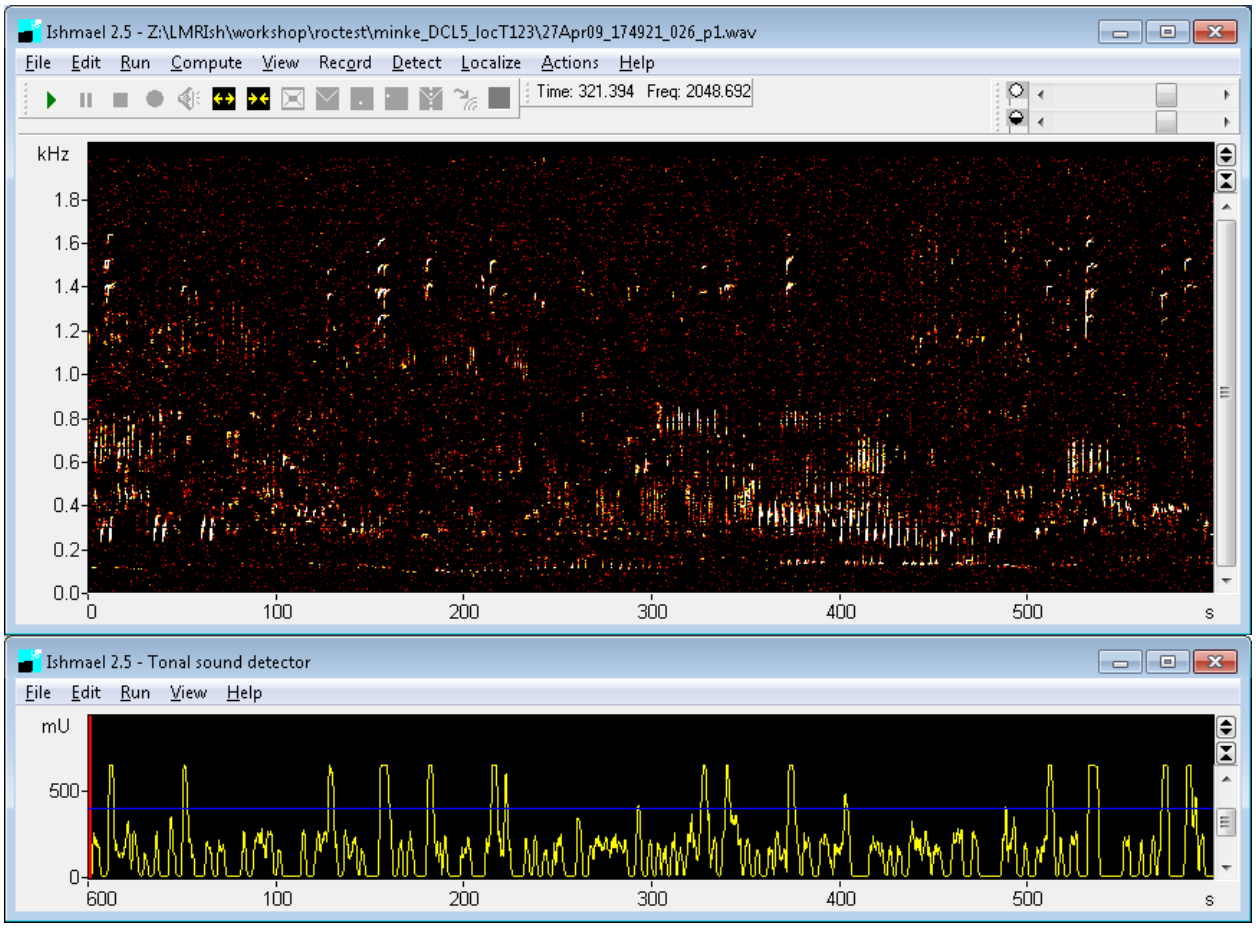
Click **OK**, and then **OK** again.



3) Load the files you want to run the detector on: **File->Open file**, navigate to the **minke_DCL5_locT123** directory, and either highlight all 21 wav files or hit Ctrl-A to do the same, then click the **Open** button.

4) Next, open a log file for holding the detection results: **Action->Open log file**, then check **Use a text file**. Click on **Choose**, navigate to the **minke_DCL5_locT123** folder where the sound files are, and enter the file name **MinkeBoingDet09.log**

5) Run the detector: Do **Run->Batch run**, uncheck **Pause after each file** if it's checked, and click OK. It should take less than a minute to process these sound files (which have 3.5 hours of recorded sound). Open **MinkeBoingDet09.log** by double-clicking on it. In the log, you can see the file names Ishmael processed as well as the calls it found. Close it.

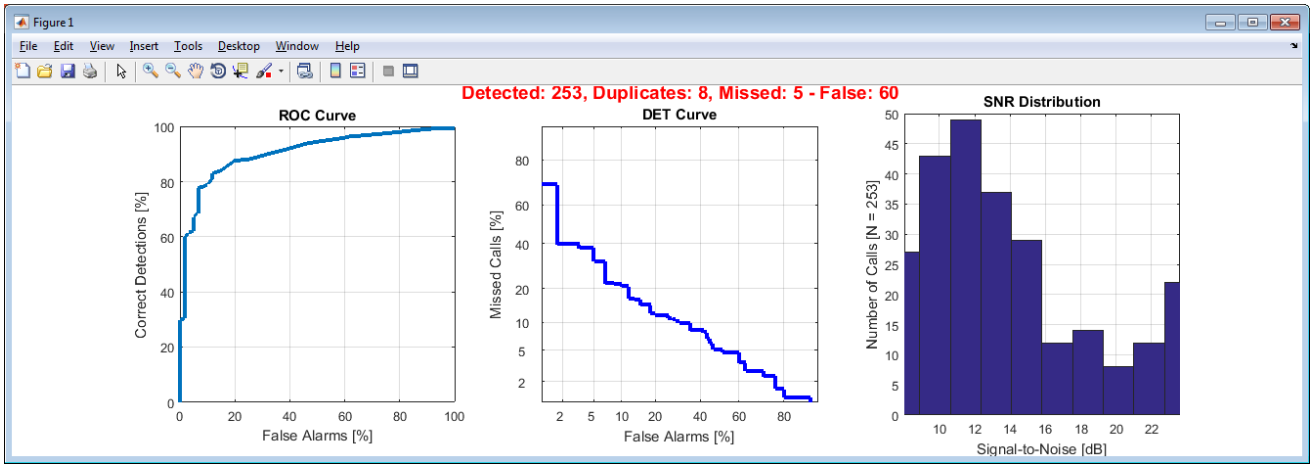


6) Launch MATLAB. Prepare it for our task: In MATLAB's command window, click the  button (it's just above the area with the >> prompt for typing in to MATLAB), navigate to the folder named **ROC_DET_Matlab_code,** and click **Select folder**. The click the **Set Path** button, click **Add Folder**, navigate to the **ROC_DET_Matlab_code** folder again, click on the **utils** folder, and click **Select Folder**. Click **Close** in the **Set Path** window.

7) In MATLAB's command window, type in **edit detector_eval** and press enter. MATLAB should open that file (**detector_eval.m**) in its Editor window. Configure it to read our manual annotation files (the .txt files) as well as the Ishmael detection log. Scroll down past line 55, where it says **`Minke whale boings off Hawaii`**, to where these variables get set:

- **`dirr`**: make this the full path name, starting with **`C:\`**, where the Ishmael log file and annotation files are. Put single quotes around it, like this:
  `dirr = 'C:\soundfile_examples\minke_DCL5_locT123\';`
- **`annExt`**: make this **`'.txt'`** (including the single-quote marks), the extension of the manual annotation files.
- **`detExt`**: make this **`'MinkeBoingDet09.log'`** (including the single-quote marks), the name of the Ishmael log file Ishmael produced.
- **`startCol`**: make this **`1`**, which specifies which column in the annotation files have the time the detections start at.
- **`stopCol`**: make this **`2`**, which specifies which column in the annotation files have the time the detections stop (end) at.
- **`snrCol`**: make this **`6`**, which specifies which column in the annotation files have the signal-to-noise ratio (SNR).
- **`dt`**: make this **`3.0`**, which says that the time of Ishmael's detection has to be within 3.0 seconds of the time of the manual annotation to count as the "same detection".
- **`newLogFileExt`**: set this to **`''`** (that's two single-quote marks)
- **`useFilenameTime`**: set this to **`false`**.

8) Run it: Click the green **Run** button at the top of MATLAB's Editor window, or type detector_eval in MATLAB's Command window. In the Command window, you should see it working through the manual annotation files one by one, then making a figure somewhat like this:



The left figure shows a ROC curve, the middle a DET curve, and the right a histogram of signal-to-noise ratios (SNRs).

9) About performance curves: A ROC curve is made by varying the detection threshold over a range of values and counting the fraction of false alarms (incorrect detections) and correct detections for each threshold. By plotting one point for each threshold and connecting the dots, MATLAB makes a curve. *Good performance is when the ROC curve comes close to the upper left corner*.

The DET curve in the middle is made similarly, but the Y-axis has missed calls instead of correct detections, the distribution takes the probabilities of the detecting and missing calls into account, and the curve is shown using log-log scaling so the detail comes out better. *Good performance is when the DET curve comes close to the lower-left corner*.

The right panel shows the signal-to-noise ratio for this set of recordings. The SNR says how much a call stands out above background noise; it's important for assessing ROC and DET curves because clearer (higher-SNR) calls are easier to detect. For baleen whale sounds, a SNR of 10 is okay, 15 is good, and 20 is very good. Most of the data here are in the okay-to-good range, so this is a reasonable representation of this detector's performance. If the SNRs had mostly been up around 15-20, we wouldn't be learning much about this detector because the detection problem presented to it – detecting some really clear calls – would have been too easy.

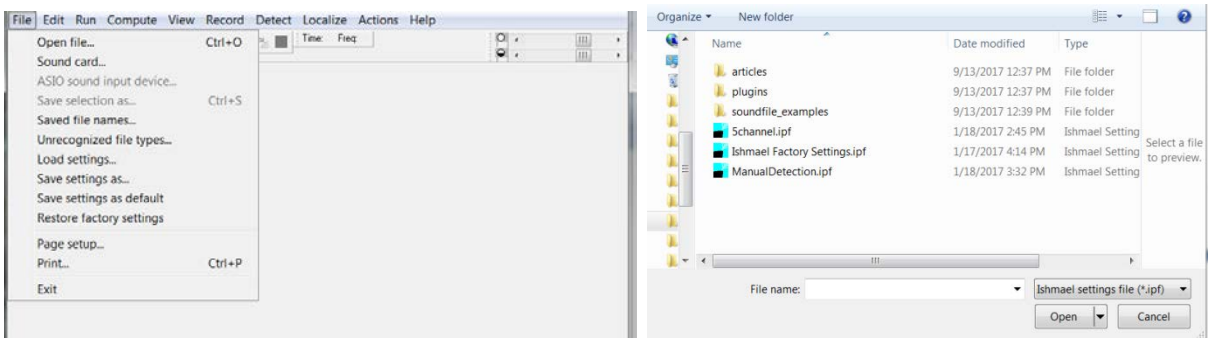
A more detailed write-up about performance assessment is in the *Signal Processing* chapter in your **articles** folder.

Advanced detection using the MATLAB interface

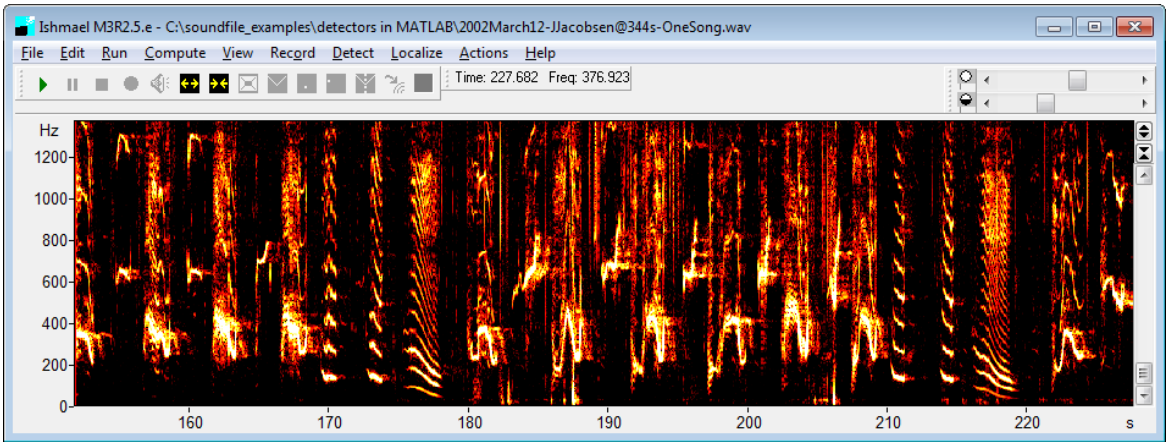
Ishmael now has an interface to MATLAB so that detectors written in MATLAB can be run in Ishmael. If you're MATLAB-savvy and have a new type of detector you want to try, you can write your detector in MATLAB and run it from Ishmael. Although there are a few pointers below on how to do this, it's mostly beyond the scope of this workshop; however, Dave M can work with you later to do this.

This is an example of using a detector, the Generalized Power Law (GPL) detector, to detect humpback whale sounds. GPL was developed and written by Tyler Helble; this detector uses his code. Ishmael's MATLAB interface is still pretty clunky to use, but it will become significantly smoother with the next release of Ishmael.

1) Load the settings file **Ishmael Factory Settings.ipf** to clear any previous settings.



2) Open the file **2002March12-JJacobsen@344s-OneSong.wav**. Make a spectrogram using a frame size of 4096 samples. You can use a larger or smaller hop size, but be sure to use *no zero-padding* for this detector. Adjust brightness and contrast too, and stretch the screen horizontally and vertically until you can see some of the humpback song units. It looks like this partway into the file:



3) You can use equalization if you like, though it's not really necessary, but *do not use a spectrogram floor value* since it will cause MATLAB to crash. (It puts a lot of zeros in the spectrogram, causing problems.)

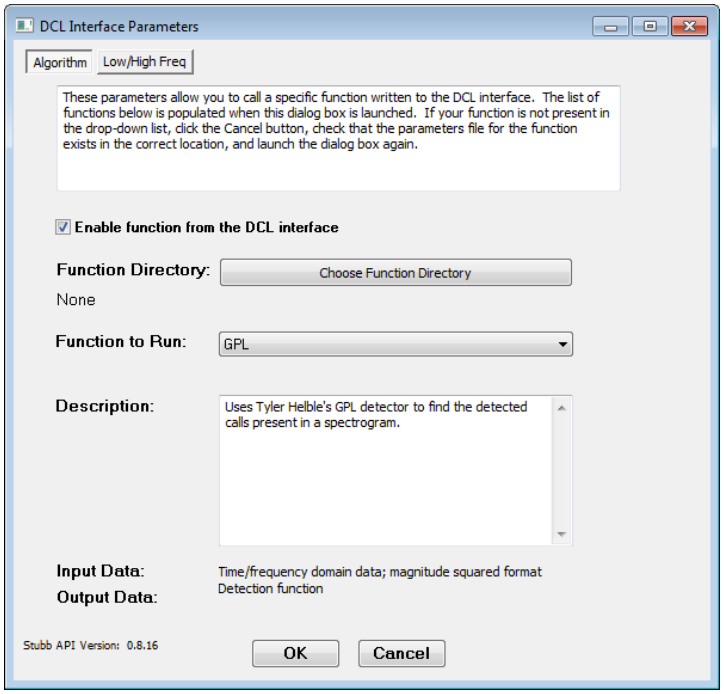
4) Launch MATLAB from the Windows Start button.

5) Find your 'stubb\plugins\GPL' directory. It might be in one of these places:
C:\Program Files (x86)\Ishmael\Ishmael 2.6\stubb\plugins\GPL
C:\ C:\Users\<*your user name*>\AppData\Local\Ishmael\stubb\plugins\GPL
C:\Ishmael\Ishmael 2.6\stubb\plugins\GPL

6) In MATLAB's command window, execute this code (you can copy and paste):

```
cd('<your stubb\plugins\GPL folder>');      % substitute the appropriate folder
addpath('GPL_v2_5')
startup
```

7) In Ishmael, do **Detect->Use stubb plugin->Algorithm (tab)**. Check the box for **Enable function from the DCL interface**, then click on the **Function to Run** drop-down and choose **GPL**.



8) Click OK.

9) Click **Run** on the menu bar or the **Run** button (the green arrow under the menu bar). Nothing happens until….

10) In MATLAB, type in **`GPLDetector`** and press Enter. At this point you should see Ishmael start to run. The detection function is all 1, but in the MATLAB window you will see a list of humpback detection times. Each one is a pair, the start and stop-time of a humpback song unit. This is currently the only output from the GPL detector; we thought we'd have its output coming into Ishmael, but it's not quite working yet.

11) Each time you run Ishmael, you'll need to also run GPLDetector in MATLAB. It doesn't matter what order you do them in – each one waits for the other if it's not running yet.

12) Sometimes Ishmael and/or MATLAB gets hung up. You have to kill both of them, possibly using the Windows Task Manager: do Ctrl-Alt-Delete and click **Start Task Manager**. On the **Applications** tab, right-click on Ishmael (or MATLAB) and choose **End Task**.

Advanced detection using ROCCA

Ishmael has an interface to ROCCA (the Real-time Odontocete Call Classification Algorithm), a system for classifying clicks and whistles developed by Dr. Julie Oswald and implemented by Michael Oswald of Bio-Waves, Inc. The ROCCA module uses parameters measured from delphinid whistles or clicks, along with user-selected random forest classification models, to classify the whistles and/or clicks to species. Whistles are detected using Ishmael's Whistle and Moan Detector, and clicks detected with Ishmael's Energy Sum Detector.

Individual whistle/click detections and classifications are also grouped together into user-defined ***Encounters***, a set of whistles and/or clicks that can all be attributed to one school of odontocetes. Variables describing the acoustic behavior of the encounter are calculated using the outputs from the whistle and click detectors, and an overall **Encounter Classification** is generated based on those variables in combination with the output of the whistle and click classifiers.

ROCCA functionality can be included in the Ishmael application, but requries Java to be installed on your computer.

About Java

Specifically, in order for Java to be able to allocate enough memory for the large classifier files, the 64-bit version of Java must be installed. If you are not sure whether or not you have Java installed, click on the Windows Start button (lower left corner on the screen) and type 'cmd' in the *Search Programs and Files* text box. This will open up a white-on-black console window. Type 'java –version' at the prompt and hit enter.

- If you get an error message, Java is not installed. Go to java.com/en/download/manual.jsp and find the link that specifically says '64-bit' (usually it is called Windows Offline 64-bit). Download and install Java.
- If you do not see the words '64-bit' somewhere in the output, you have the 32-bit version installed. Here's an example of a 32-bit version:

```
C:\Users\mo55>java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) Client VM (build 25.144-b01, mixed mode)
```

  Don't worry, you can install both 32-bit and 64-bit on the same machine. Go to java.com/en/download/manual.jsp and find the link that specifically says '64-bit' (usually it is called Windows Offline 64-bit). Download and install java. This version will now become your default version of Java. If you need to go back to the 32-bit version (perhaps to run some older programs), the easiest way is to return to the Java webpage and download/install the 32-bit version (the one that does *not* have the words 64-bit in it). Note, however, that Ishmael/ROCCA will likely not be able to load the classifier files if it is trying to use 32-bit Java.
- If the words '64-bit' are included in the output, you have the correct version installed. See the last line here:

```
C:\Users\mo55>java -version
java version "1.8.0_141"
Java(TM) SE Runtime Environment (build 1.8.0_141-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.141-b15, mixed mode)
```

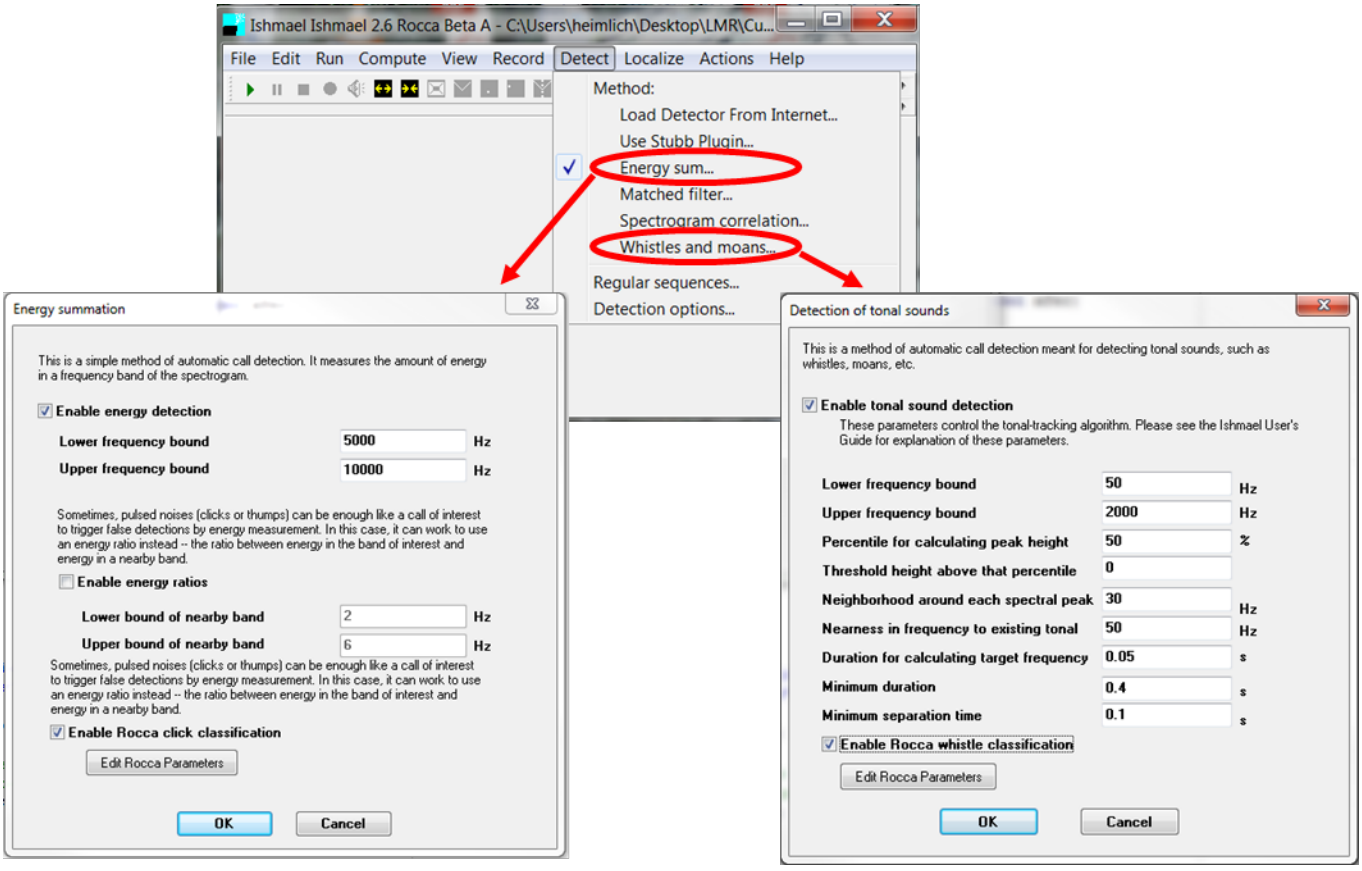
ROCCA classifier files

You will also need ROCCA classifier file(s) for the region you are interested in. These files contain the information necessary for ROCCA to evaluate the detections that Ishmael sends to it and classify them to species. Classifier files for three different regions (Northwest Atlantic, Temperate Pacific and Hawaiian Islands) can be downloaded from the Downloads section of the Pamguard website (www.pamguard.org/downloads.php?cat_id=5). Each download is a zipped package containing three different classifier files (with the prefix .model) and a PDF containing a description of the classifier. There is a whistle classifier, a click classifier, and an encounter classifier for each region. All three classifiers are random forest classifiers. The encounter classifier uses the output from the whistle and click classifiers to identify acoustic encounters (or schools) to species. The description for each classifier includes information about the species recognized by the classifiers, as well as information about the datasets used to train them. More detailed information about the classifiers can be found in Oswald and Yack (2017).

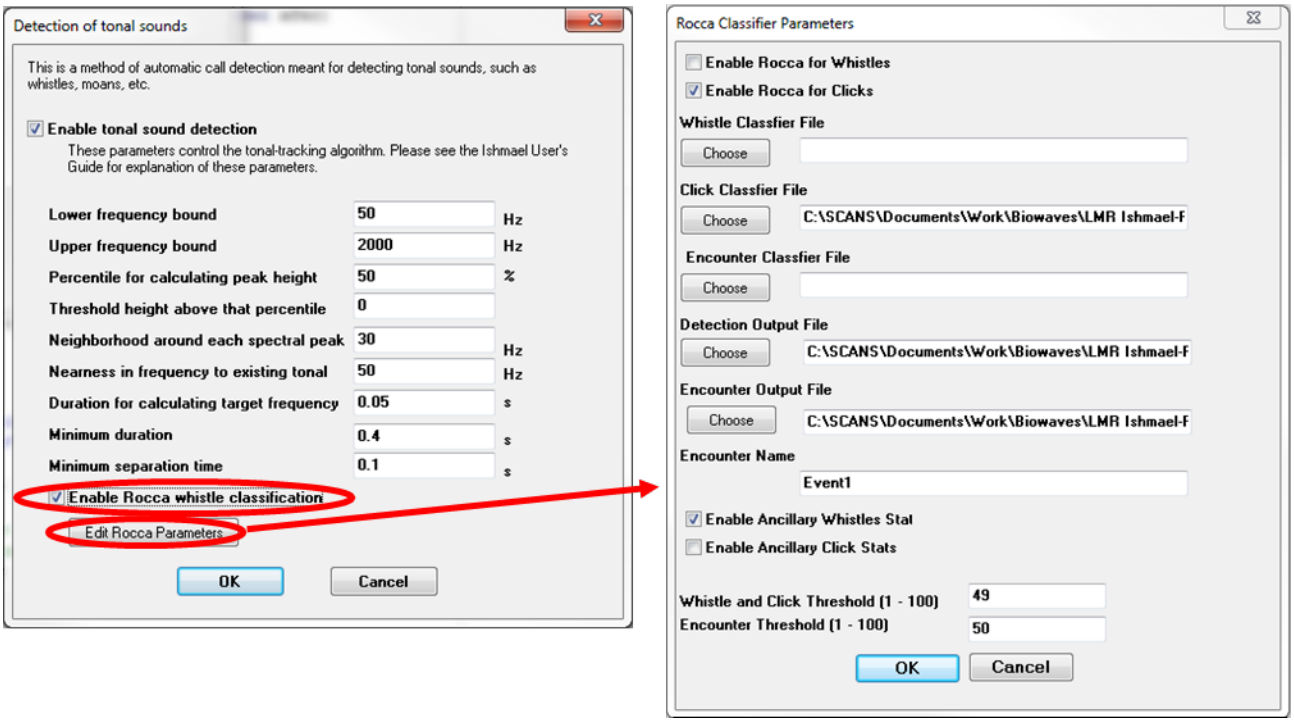
Once you have downloaded the zip file containing the classifiers, right-click on it and select 'Extract all'. You are then given the option to specify where to save the files to. Store your classifier files anywhere on your computer; you can use the Desktop if you like.

Using ROCCA

1) In Ishmael, open either the click detector using **Detect->Energy sum** or the whistle detector using **Detect->Whistles and moans**. Configure it as described in the **Dectection** section (page32) so that it detects the desired type sounds of your target species



2) To enable the corresponding ROCCA classification, check the "**Enable Rocca….**" box at the bottom of the window that pops up ("**Enable Rocca click classification**" for Energy Sum or "**Enable Rocca whistle classification**" for tonal sounds). Then click the button labeled **Edit Rocca Parameters**.



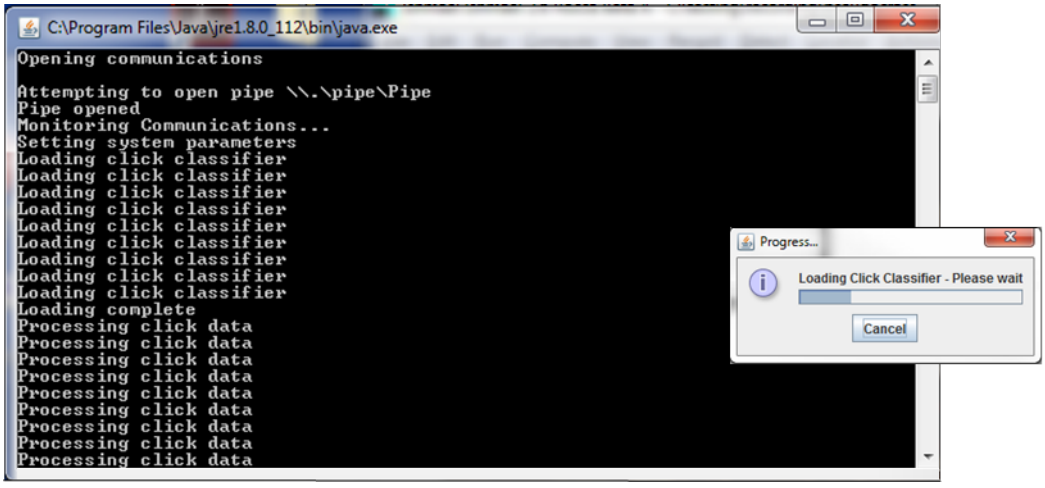
A new window will pop up to allow you to enter parameters specific to the ROCCA classification module:

- The **Whistle Classifier File** and **Click Classifier File** boxes tell ROCCA which classifiers to use. Hit the **Choose** button and select the appropriate classifier file previously downloaded, or type the path and filename directly into the text box. Either a Whistle Classifier or a Click Classifier must be specified, but you can specify both if you want. Note that if both are specified, both will be loaded into memory. For memory-limited machines, this may cause out-of-memory errors.
- **Encounter Classifier File** is optional, and will only run if both a Whistle and a Click Classifier have been selected and the ancillary variables check boxes have been checked. If an Encounter Classifier is chosen, the encounter (school) will be classified based on the output of the whistle and click classifiers and the ancillary variables. If an Encounter Classifier is not specified, encounters will be classified as the species receiving the highest cumulative number of random forest tree votes in each Encounter.
- The **Detection Output File** box tells ROCCA where to save information about each whistle or click detection passed to it by Ishmael. This should be a .csv file. Information for each detection is added as a new row. See Tables 1 and 2 for the variables that are measured for each whistle or click detection and saved to the file. The species classification for each detection is also saved, at the end of the row after all measured variables.
- The **Encounter Output File** box tells ROCCA where to save information about each Encounter . This information is saved as a .csv file. Table 3 gives the ancillary variables measured using the whistles and clicks in each encounter. These variables, along with the random forest votes for each species and the overall Encounter Classification, are all saved to the Encounter Output File.

- **Encounter Name** is a label that lets the user name the current acoustic encounter to group whistle and click detections under. Changing the name tells ROCCA to begin a new Encounter and to clear the whistle and click information calculated for the previous Encounter. Each row of data in the Encounter Output File is a separate Encounter, and includes the Encounter Name.
- **Enable Ancillary Whistle Stats** gives the user the option of calculating ancillary variables for the whistle detections in the Encounter (see Table 3). The calculations take time, and if there are many whistles in the Encounter the calculations may slow down the overall processing. If you want to use an Encounter Classifier file, ancillary whistle stats ***must*** be run. If you are not using an Encounter Classifier file, the ancillary whistle stats are optional.
- **Enable Ancillary Click Stats** gives the user the option of calculating ancillary variables for the click detections in the Encounter (see Table 3). The calculations take time, and if there are many clicks in the Encounter the calculations may slow down the overall processing. If you want to use an Encounter Classifier file, ancillary click stats ***must*** be run. If you are not using an Encounter Classifier file, the ancillary click stats are optional.
- **Enable GPS data** gives the user the option of sending GPS data to ROCCA. GPS data is used by the Encounter Classifier to help with the Encounter classification. If Ishmael has not been configured to accept GPS data, do not check this box.
- **Whistle and Click Threshold** is the threshold value (in percent) to use when classifying individual whistles or clicks. In the random forest classifier, if the percentage of trees voting for the predicted species falls below this threshold, the whistle or click is considered non-classifiable and is labelled as ambiguous (*AMBIG*).
- **Encounter Threshold** is the threshold value (in percent) to use when classifying Encounters. If the percentage of trees voting for the predicted species in the random forest falls below this threshold, the Encounter is considered non-classifiable and is labelled as ambiguous (*AMBIG*).

Enter file names either by typing or pasting in the file name, or by clicking the Choose button, navigating to the folder you want, and entering the classifier file name you want to use.

Once the parameters have been set, you can simply click Ishmael's green **Run** button to start the analysis. The first time you do this, you will see Java launching in a white-on-black command window, then ROCCA loading the classifier file(s), which typically takes several seconds.



It's fine to minimize this command window so it's in your toolbar at the bottom instead of the middle of the screen.

ROCCA is now running and connected to Ishmael. Whenever Ishmael detects a click or whistle, it will send it to ROCCA, which will classify each detection to one of its known species (as documented in the PDF file included in the classifier file download), as *AMBIG* if the votes fall below the Whistle/Click Threshold, as *NA* if the detection has been pruned (see next page), or as *ERR* if there was an error during the classification process. Measured parameters and classifications are written to ROCCA's Detection Output File. When Ishmael analysis has ended – either by reaching the end of the acoustic file, or the user manually stopping Ishmael – an Encounter classification is generated and saved to the Encounter Output File.

Whistle, click and Encounter results can be viewed by opening the .csv output files in Excel or another spreadsheet program. Note, however, that new results cannot be added to the files by Ishmael if they are open in another application, so always remember to close the file before running Ishmael again.

When you close Ishmael, the ROCCA process should stop and the white-on-black command window should disappear. (If it doesn't, close it manually by clicking the red "X" in the upper right corner.)

Note: on *Pruning*: in order to minimize false detections, certain measured whistle and click variables are tested against predefined ranges to validate the detection. The variables and ranges are specific to each classifier region, and are listed in Tables 6 and 7. Detections with values that fall outside of the range are *pruned*; that is, they are not classified. The detection information is still saved to the Detection Output File, but *NA* is given as the classification.

ROCCA Tables

Table1. ROCCA: Variables measured from clicks

Click variable	Description
Duration	Click length [s]. *Calculated by identifying the peak amplitude of the click and searching in both directions until the amplitude has dropped by 12 dB to find the beginning and end times of the click*
Freqpeak	Peak frequency [Hz]
BW3DB	-3dB bandwidth [Hz]
BW3DBLow	-3dB bandwidth lower limit [Hz]
BW3DBHigh	-3dB bandwidth upper limit [Hz]
BW10DB	-10dB bandwidth [Hz]
BW10DBLow	-10dB bandwidth lower limit [Hz]
BW10DBHigh	-10dB bandwidth upper limit [Hz]
RMSSignal	RMS of the click [dB]
RMSNoise	RMS of the noise [dB]
SNR (Signal-to-noise ratio)	RMSSIGNAL- RMSNOISE [dB]
NCrossings	Number of zero crossings
Sweeprate	Sweep rate of the zero crossings [kHz/ ms]

Note: Tables 3,4 and 5 are on the following pages.

Table 2. ROCCA: Variables measured from whistles

Variable	Explanation
Begsweep	slope of the beginning sweep (1 = positive, -1 = negative, 0 = zero)
Begup	binary variable: 1 = beginning slope is positive, 0 = beginning slope is negative
Begdwn	binary variable: 1 = beginning slope is negative, 0 = beginning slope is positive
Endsweep	slope of the end sweep (1 = positive, -1 = negative, = 0 zero)
Endup	binary variable: 1 = ending slope is positive, 0 = ending slope is negative
Enddwn	binary variable: 1 = ending slope is negative, 0 = ending slope is positive
Beg	beginning frequency (Hertz [Hz])
End	ending frequency (Hz)
Min	minimum frequency (Hz)
Dur	duration (seconds)
Range	maximum frequency - minimum frequency (Hz)
Max	maximum frequency (Hz)
mean freq	mean frequency (Hz)
median freq	median frequency (Hz)
std freq	standard deviation of the frequency (Hz)
Spread	difference between the 75th and the 25th percentiles of the frequency
quart freq	frequency at one-quarter of the duration (Hz)
half freq	frequency at one-half of the duration (Hz)
Threequart	frequency at three-quarters of the duration (Hz)
Centerfreq	(minimum frequency + (maximum frequency-minimum frequency))/2
rel bw	relative bandwidth: (maximum frequency - minimum frequency)/center frequency
Maxmin	maximum frequency/minimum frequency
Begend	beginning frequency/end frequency
Cofm	coefficient of frequency modulation (COFM): take 20 frequency measurements equally spaced in time, then subtract each frequency value from the one before it. COFM is the sum of the absolute values of these differences, all divided by 10,000
tot step	number of steps (10 percent or greater increase or decrease in frequency over two contour points)
tot inflect	number of inflection points (changes from positive to negative or negative to positive slope)
max delta	maximum time between inflection points
min delta	minimum time between inflection points
maxmin delta	maximum delta/minimum delta
mean delta	mean time between inflection points
std delta	standard deviation of the time between inflection points
median delta	median of the time between inflection points
mean slope	overall mean slope (Hz/s)
mean pos slope	mean positive slope (Hz/s)
mean neg slope	mean negative slope (Hz/s)
mean abs slope	mean absolute value of the slope (Hz/s)
Posneg	mean positive slope/mean negative slope
perc up	percent of the whistle that has a positive slope
perc dwn	percent of the whistle that has a negative slope
perc flt	percent of the whistle that has zero slope

Variable	Explanation
up dwn	number of inflection points that go from positive slope to negative slope
dwn up	number of inflection points that go from negative slope to positive slope
up flt	number of times the slope changes from positive to zero
dwn flt	number of times the slope changes from negative to zero
flt dwn	number of times the slope changes from zero to negative
flt up	number of times the slope changes from zero to positive
step up	number of steps that have increasing frequency
step dwn	number of steps that have decreasing frequency
step.dur	number of steps/duration
inflect.dur	number of inflection points/duration

(cont. Table 2. ROCCA: Variables measured from whistles)

Table 3. ROCCA: Ancillary variables measured for each Encounter.

Ancillary Variable	**Description**
Encounter Duration	Time from start of first whistle or click to end of last whistle or click (in seconds)
Number of Whistles	Total number of whistles
Number of Clicks	Total number of clicks
Whistling Duration	Time from beginning of first whistle to end of last whistle (in seconds)
Clicking Duration	Time from start of first click to end of last click (in seconds)
Minimum Time Between Whistles	Minimum time between the beginning of one whistle and the beginning of the next whistle (in seconds)
Minimum Time Between Clicks	Minimum time between the beginning of one click and the beginning of the next click (in seconds)
Maximum Time Between Whistles	Maximum time between the beginning of one whistle and the beginning of the next whistle (in seconds)
Maximum Time Between Clicks	Maximum time between the beginning of one click and the beginning of the next click (in seconds)
Average Time Between Whistles	Average time between the beginning of one whistle and the beginning of the next whistle (in seconds)
Average Time Between Clicks	Average time between the beginning of one click and the beginning of the next click (in seconds)
Average Whistles per Second	Number of whistles per second (Number of Whistles / Whistling Duration)
Average Clicks per Second	Number of clicks per second (Number of Clicks / Clicking Duration)
Whistle Density	Sum of Individual Whistle Durations / Whistling Duration
Click Density	Sum of Individual Click Durations / Clicking Duration
Average Whistle Overlap	Total amount of time whistles overlap other whistles (in seconds)
Average Click Overlap	Total amount of time clicks overlap other clicks (in seconds)
Whistle : Click Ratio	Number of Whistles / Number of Clicks
Latitude	Latitude (decimal degrees)
Longitude	Longitude (decimal degrees)

Table 4. ROCCA: Whistle variables used to remove false detections and inaccurate contour extractions by geographic region. Whistles with values below and/or above the listed values will not be classified.

Dataset	Variable	Remove values below	Remove values above
Northwest Atlantic	Maximum frequency	10.6 kHz	27 kHz
	Mean absolute slope	9.1Hz/s	82 kHz/s
	Duration	0.15 s	2.5 s
Temperate Pacific	Maximum frequency	5 kHz	25 kHz
	Minimum frequency	3 kHz	18 kHz
	Duration	0.1 s	1.5 s
	Mean absolute slope	2 kHz/s	28 kHz/s
Hawaii	Duration	0.1 s	1 s
	Maximum frequency	5 kHz	25 kHz
	Minimum frequency	4 kHz	15 kHz
	Mean absolute slope	4 kHz/s	60 kHz/s
	Frequency range	0.8 kHz	14 kHz

Table 5. ROCCA: Click variables used to remove false detections by geographic region. Clicks with values below and/or above the listed values will not be classified.

Dataset	Variable	Prune values below	Prune values above
Northwest Atlantic	SNR (dB)	N/A	35
	Duration (s)	0.005	0.6
Temperate Pacific	SNR (dB)	N/A	35
	Duration (s)	0.005	0.6
Hawaii	SNR (dB)	N/A	40
	Duration (s)	0.01	0.6

Final exercise

Detect calls in your own acoustic dataset!

References

Dooling, R.J., C.W. Clark, R. Miller, and T. Bunnell. (1982). Program package for the analysis and synthesis of animal vocalizations. Beh. Res. Methods & Instrumentation 14(5):487.

Buck, J.R., H.B. Morgenbesser, and P.L. Tyack. (2000). Synthesis and modification of the whistles of the bottlenose dolphin, *Tursiops truncatus*. J. Acoust. Soc. Am. 108:407-416.

Mellinger, D.K., and J. W. Bradbury. (2007) Acoustic measurement of marine mammal sounds in noisy environments. Proc. International Conference on Underwater Acoustic Measurements: Technologies and Results, Heraklion, Greece, pp. 273-280.

Mellinger, D.K., M.A. Roch, E.-M. Nosal, and H. Klinck. (2016) Signal processing. Chapter 15 in W. Au and M. Lammers, eds., *Listening in the Ocean*, pp. 359-409. Springer-Verlag: New York.

Oswald, J.N. and T.M. Yack. (2017) Development of automated whistle and click classifiers for odontocete species in the western Atlantic Ocean, Temperate Pacific and the waters surrounding the Hawaiian Islands. Final Report, Living Marine Resources Contract N39430-14-C-1431. Avail. from julie.oswald@bio-waves.net.

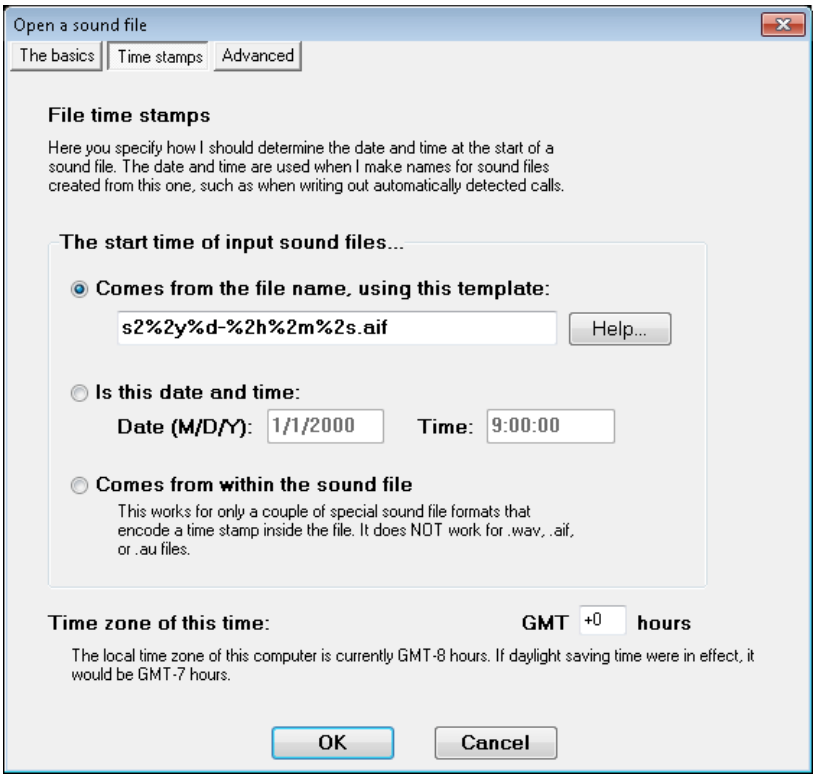
Van Trees, H.L. (1968). *Detection, Estimation, and Modulation Theory*, Vol. I. Wiley: New York.

APPENDIX 1: Time stamps in Ishmael

Ishmael is able to interpret the time in a file in a couple of ways.

Choose **File→Open file**, click on the basics tab, choose your basic preferences, then click on the **Time Stamps** tab of the **Open a sound file** drop down menu. This is where you can make sure that Ishmael is interpreting time in a file correctly.

If you are opening a *.wav, *.au or *.aif file and that file has the date and time encoded in the name of each file, you can set up a template that Ishmael parses the name correctly.

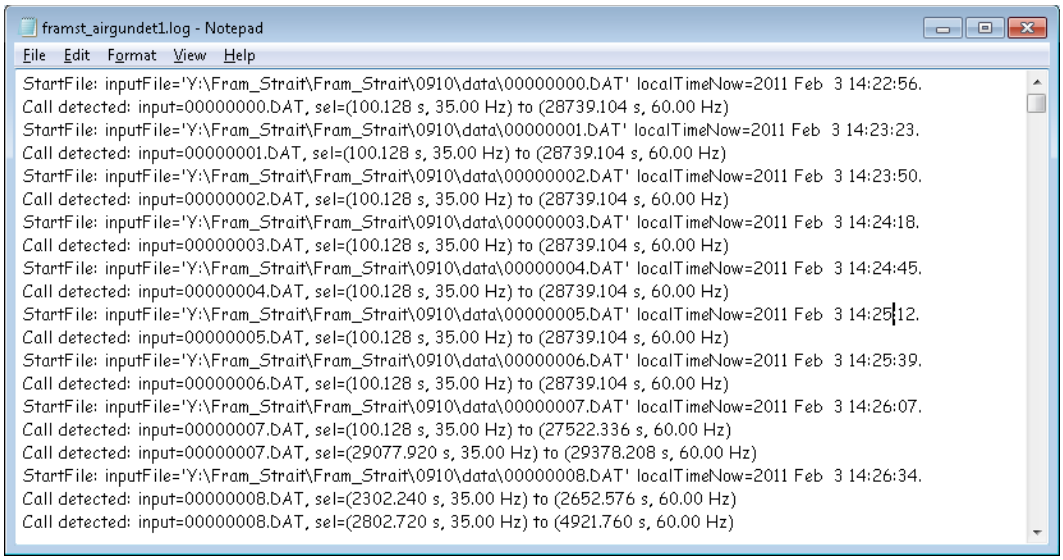


For example….

mysound-4-14-99-1052.wav would be **mysound-%M-%D-%Y-%2h%2m.wav**

Optionally, you can choose the second option, **Is this date and time** and enter this information in the spaces provided. Finally, a few file formats have time encoded the header of the file (e.g. our PMEL/OSU hydrophone files) and you denote that with the third option.

Telling Ishmael how to interpret time is very important in many instances. For example, when running a detector, the time of the detection is written to the log file. If you don't provide a means for Ishmael to interpret date and time, you'll get a log that looks like the one below:

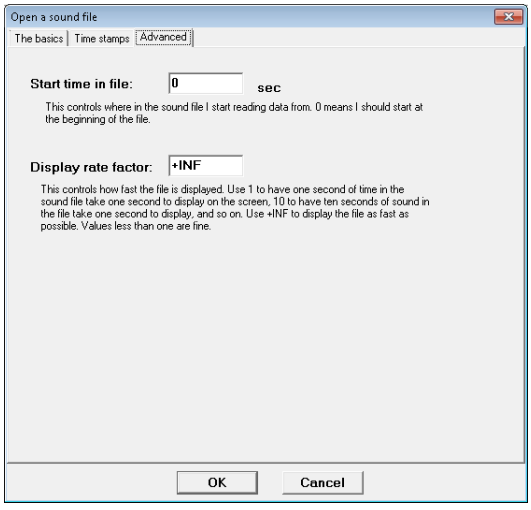


This is also a great place to convert local time to GMT if necessary. If your files were stamped with local time, and you'd like to convert them to GMT, enter the offset hours in the space provided.

Finally, click on the "**Advanced**" tab. The "**start time in file**" option allows you to start some number of seconds into a file. This is convenient when you have long files and are testing a detector on a sound that begins well into your file and don't want to wait for the first half of the data to scroll by. NOTE: if you use this option, make sure you reset the "**start time in file**" to zero, or your process (detector, etc.) will not be run on data occurring before the start time you've indicated. This goes for all files you've chosen in a directory!

The "**Display rate factor**" is useful when you want to slow down the scrolling display of a file. This is often needed when testing a detector, examining unknown calls, etc.

Click OK when all of your file settings have been made.



APPENDIX 2: Localization

When set up correctly, Ishmael can do localization—determining the location of a sound source— and tracking (combining successive locations to estimate movement). Both of these….

- can be done in 1, 2, or 3 dimensions
- are usually done using the differences in the times that a call arrives at several hydrophones
    - for example, with two closely-spaced hydrophones, a given time delay corresponds to a sound source at a certain bearing angle

Here we are going to concentrate on localization, and go over two methods: **Phone pair** bearing calculation and **Hyperbolic X-Y** position calculation. Having Ishmael set up correctly for all localization methods **requires these things:**

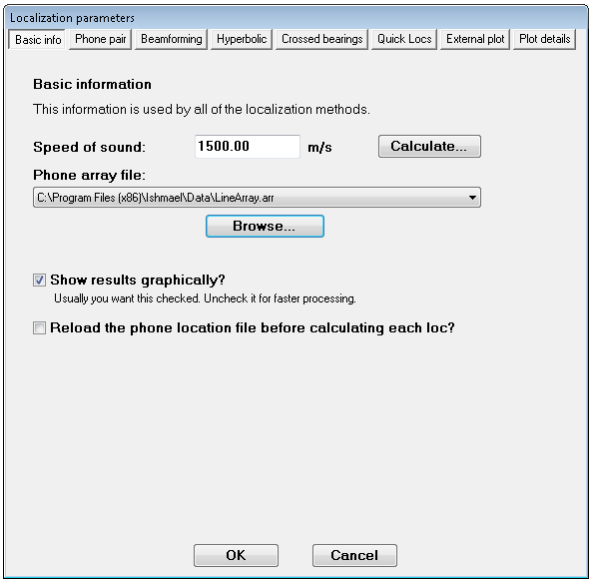
- Sounds with **two or more channels**, either from a sound file or from real-time input. Ishmael's methods for determining bearings require at least two channels of sound (two simultaneously operating microphones or hydrophones), and the X-Y position method requires at least three channels.
- Information about the **positions of the microphones** or hydrophones.
- Information about the **speed of sound**.

General Configuration

To set up Ishmael for doing localization, first create a text file with the positions of your microphones/hydrophones. This file, the ***phone array file***, should have one line for each phone, specifying two values per phone: the X- and Y-positions of the phone, in meters, with spaces or tabs between the two numbers in the file. The positions should be relative to some origin or (0,0) point. This origin can be anywhere you like, typically at one of the phones or (for towed hydrophone arrays) at the ship. An example phone array file for a four-phone array looks like this:

-100 20 -20 150 100 30 75 -30

Save your *.arr file, being sure to save it as "text only" in your word processor. In Ishmael, choose **Localize**->**Loc options...** from the menu, choose the **Basic info** tab, and enter the name of your phone array file. Also enter the speed of sound here—in the **ocean**, it's typically around **1500 m/s**; in **air**, typically around **340 m/s**. What you do next depends on the type of localizations you want to do. In the dialog box, pick the appropriate tab:

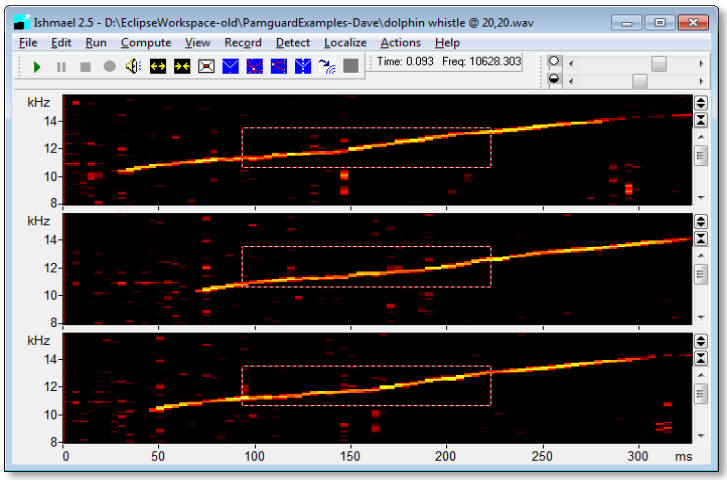


Localizing sounds – Hyperbolic X-Y calculation

Here we'll try to localize a whistling dolphin via a **Hyperbolic X-Y calculation**. These locations are X-Y positions calculated by using differences in the times that sounds arrive at multiple phones. Conceptually, the time difference for each pair of phones determines a hyperbola on which the calling animal lies. The intersection of these hyperbolae determines the animal's X-Y position. This localization method is useful when the sound source is relatively close to the phone array—say, closer than several times the separation of the most distant phones. Positions are plotted on a map; you deter-mine how big the map is using the four values specified in this dialog box. Choose values large enough to encompass both your phone array and some space around it.

1) Open Ishmael and clear all the default settings by loading **Ishmael Factory Settings.ipf** via **File→Open** file. Now open the file **dolphin whistle @ 20,20.wav** .

2) Make a nice spectrogram of this wav file so that the whistle is visible in the three channels. Select part of this whistle (the box will automatically be drawn in all three spectrograms).

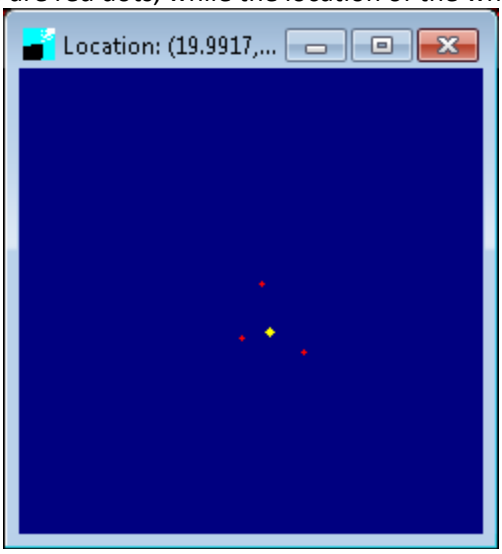


3) Next, load the array file **dolphin whistle @ 20,20.arr** via the **Localize→Loc options** menu and the **Basic info** tab. Also, change the speed of sound to 1500 m/s. Click **Ok**. Take a quick look at the *.arr file with Notepad or another text editor. It should look like this:

```
-1.0000000e+002  0.0000000e+000
-2.0000000e+001  2.0000000e+002
 1.5000000e+002 -5.0000000e+001
```

4) Click the "hyperbolic location" button  .

5) Hydrophone locations are red dots, while the location of the whistling dolphin is the white dot.



6) What happens after you locate a vocalizing animal? You can save the information in a log file send the information to:

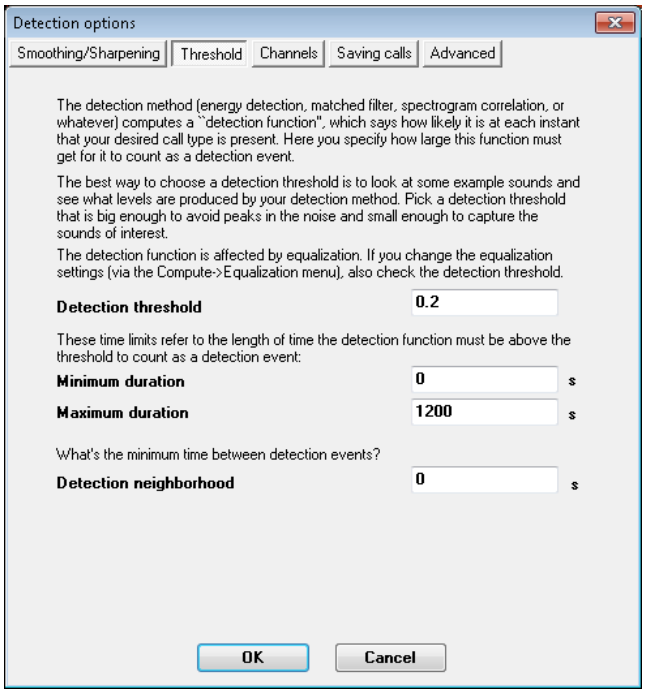
- WhalTrak (Jay Barlow)
- WhaleTrack II (Glenn Gailey)
- WILD (Whale Identification, Logging, and Display – Chris Kyburg)
- Save it to a text file……

Localizing sounds - Phone pair bearing calculation

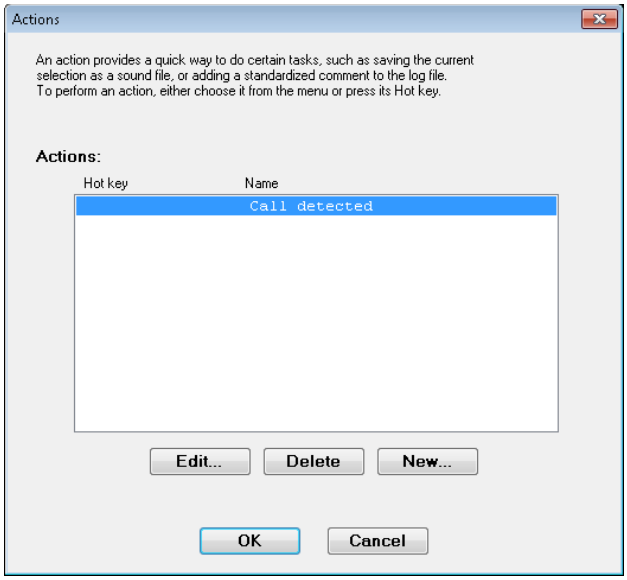
A pair of phones determines a hyperbola on which the sound source lies. When the phones are close together, this hyperbola is very nearly a pair of bearing angles, one on the left side and one on the right side of the line joining the two phones. When you localize a sound using this method, bearing angles are plotted on a chart that shows only one of the two bearing angles, the one between 0° and 180°.. Choose which two phones you wish to use for the bearing calculation.

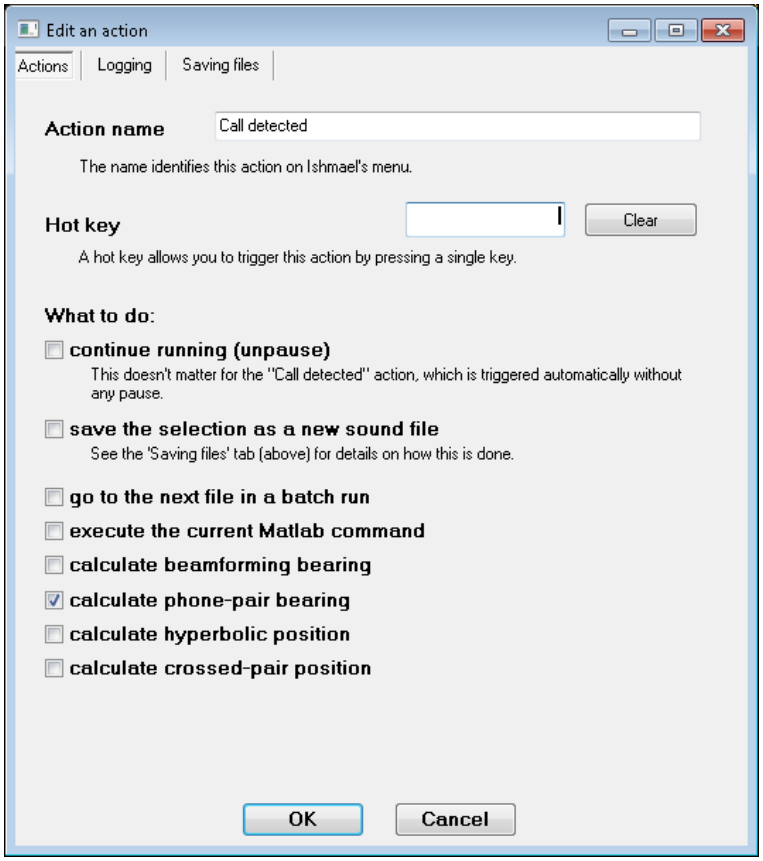
Here we will set up a detector to detect sperm whale clicks in acoustic data and localize the source of these clicks with the **Phone pair bearing** localization method in Ishmael.

- Open Ishmael and clear all the default settings by loading **Ishmael Factory Settings.ipf** via **File→Open** file. Now open the file **BTplotsperm2001-010723-153000.wav**
- Make a spectrogram (frame size = 128, hop size 1/2x), turn on equalization (time constant 0.1s), set a floor (0.4) and ceiling (0.728405), and click **OK**.
- Set up an **Energy Sum** detector
- **Detect→Energy sum**, chose Enable energy detection and set the lower and upper frequency bounds (2000-5000 Hz). Click **Ok**.
- **Detect→Detection options→Threshold** (tab), set the threshold to 0.2, and 0 and 1200 for the minimum and maximum duration
- **Detect→Detection options→Channels** and chose just one channel; Detections will be made in this channel and then correlated with other clicks in the other channel(s)



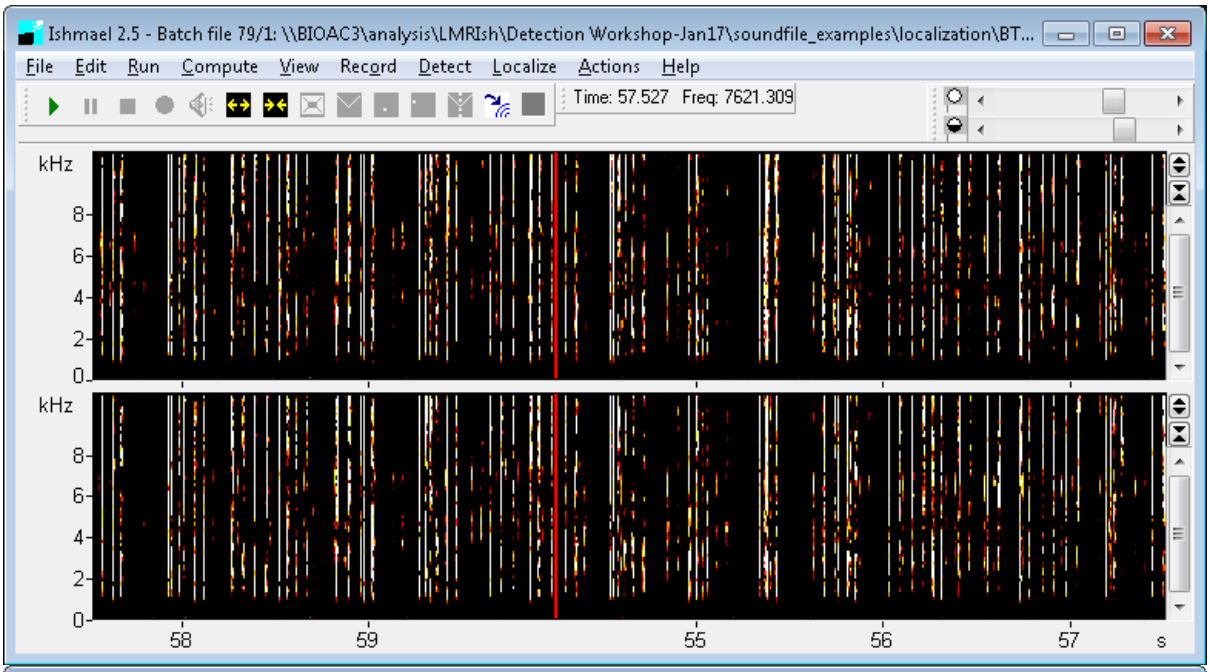
- **Detect→Detection options→Saving calls** (tab)
    - set "before start of call" and "after end of call" to 0.005
- **Actions→Edit actions**, click on **Call detected**, **Edit** and on the Actions page check **calculate phone-pair bearing**, then click on the **OK** button.

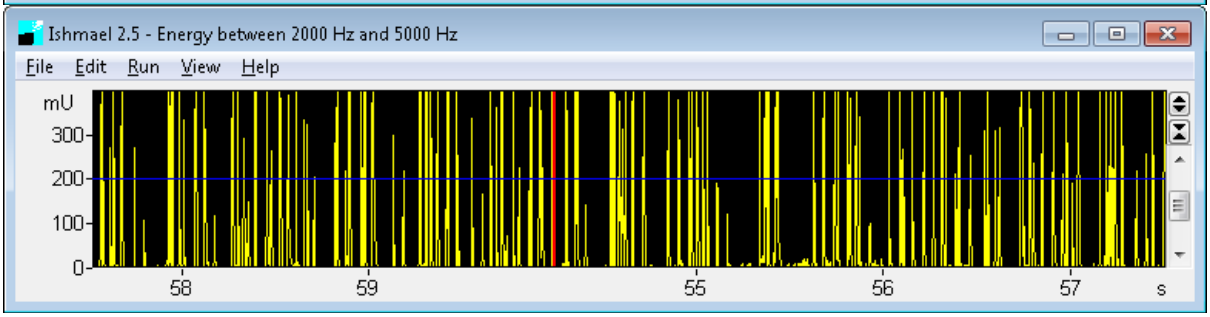


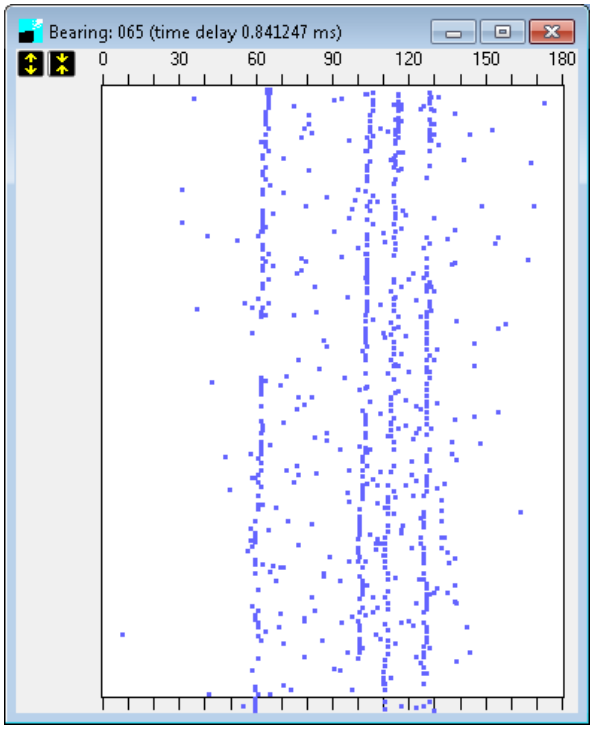


- Choose **Localize→Loc options→Basic info**, enter the speed of sound (1500m/s) and browse to the location of the phone array file **BTplot-JGordonArray.arr** and click open to load it.
    - Also check **Show results graphically**, then …..
- Click on the **Localize→Loc options→Phone pair** tab, uncheck **write intermediate results to log file**, and choose **No** under **show intermediate results on screen**. Click **Ok**.
- Now click the green **Run** button (or **Run** from the menu bar). You should see the detector window appearing with many detections exceeding the threshold, and plot window appear, drawing a blue dot for each localization.

See next page for the result!


Ishmael 2.5 - Batch file 79/1: \\BIOAC3\analysis\LMRIsh\Detection Workshop-Jan17\soundfile_examples\localization\BT...
File Edit Run Compute View Record Detect Localize Actions Help
Time: 57.527 Freq: 7621.309
kHz
8
6
4
2
0
kHz
8
6
4
2
0
58
59
55
56
57
s


Ishmael 2.5 - Energy between 2000 Hz and 5000 Hz
File Edit Run View Help
mU
300
200
100
0
58
59
55
56
57
s


Bearing: 065 (time delay 0.841247 ms)
0
30
60
90
120
150
180

NOTES