Using Praat for Linguistic Research

Dr. Will Styler

Document Version: 1.9.1

Last Update: March 19, 2022

Important!

This document will be continually updated and improved. Download the latest version at:

`http://savethevowels.org/praat`

You can also contribute changes, fix typoes, or make additions by making pull requests using github:

`https://github.com/stylerw/usingpraat`

Contents

1 Version History

- 1.9.1 – March 19th, 2022 - Added a few notes to Source Filter Resynthesis section.
- 1.9 - March 4th, 2022 - Moved to open contribution model on GitHub, squished some bugs, fixed and separated the bibliography, paid down some technical debt.
- 1.8.3 - March 10th, 2021 - Added a brief discussion of one bit requantization under formula modification.
- 1.8.2 - October 5th, 2020 - Added a 'Playing Sounds' section, with mention of the 'Add Silence Before' option which specifically helps devices that disable speakers until signal is detected and recent Macs.

- 1.8.1 - December 25th, 2017 - Fixed a few minor typoes in the text, added a note about stereo and sound localization, added details about sampling rate and acoustic nasality measurement to the relevant sections, and added reference to Styler 2017 to the Nasality Section, which more explicitly describes the foibles of nasality measurement.
- 1.8 - December 18th, 2017 - Now in Stereo! Added discussion of recording stereo signals, manipulating mono vs. stereo, and a few other notes (See Sections 5.1, 8.2). Also added some minor details to LongSound, recording, and squished a few typoes.
- 1.7.2 - October 31st, 2017 - Added another example of time in a formula thanks to some clarification from Paul Boersma (see Section 8.13).
- 1.7.1 - October 25th, 2017 - Added an example demonstrating how to use time as a component of a formula manpulation to 8.13. Niche, but useful! Clarified further on October 31st.
- 1.7 - January 2017 - Revisions for the LSA "Praat Beyond the Basics" course, including major revisions to amplitude scaling/matching in Section 8.10, adding Section 8.11 discussing concatenating sounds, some serious and minor formatting and typo fixes, and a few other tweaks and warnings (particularly about Praat becoming unresponsive) within the scripting section.
- 1.6.4 - October 5th, 2016 - A few small clarifications based on feedback from Marcin Włodarczak. Also, updated list of free software for accomplishing batch tasks by removing the (now moribund) Miro Video Converter, adding Sox and OpenSesame, and mentioning Forced Alignment.
- 1.6.3 - May 26th, 2016 - Added words of caution about A1-P1 nasality measurement to Section 7.12. Also fixed a typo in the "Create Sound from Formula" section (Thanks to Penelope Howe for the catch!)
- 1.6.2 - January 14, 2016 - Discussed how to exit a for loop in Section 12.4.1 and added a Section (and stern warning) about `goto` in Section 12.4.5.
- 1.6.1 - October 29th, 2015 - Added some dire-sounding warnings about the need to update Praat regularly, in part reflecting the fact that we're now up to Version 6. Also added Section 12.2.3, which discusses some common ways of fixing things when a Praat script you've downloaded from the internet simply won't run.
- 1.6 - July 5th, 2015 - Added Section 13: "Advanced Techniques", including (initially) Amplitude Envelopes, AM Demodulation, and using Praat Sounds for non-acoustic data. Also added a needlessly florid conclusion, to appease my sense of symmetry.
- 1.5.1 - April 30th, 2015 - Added a quick tip about closing Editor Windows generated by script to Section 12.4.8.
- 1.5 - February 2015 - Added "Script tips!" to various parts of the document to aid in automating these tasks, and added sample code for modifying duration. Completely re-typeset for easier maintenance. Modified some notes about A1-P0, added some scripting detail, and

made lots of small revisions for easier understanding.

- 1.4.9 - January 22nd, 2015 - Added a brief description of modifying Spectral Tilt in Praat
- 1.4.8 - November 28th, 2014 - Added a section discussing tasks best done outside of Praat and the best free alternative tools (Section 12.1.1), gave stern warnings regarding the use of H1-H2 for creak/breathiness (Section 7.11), the use of LongSound files (Section 6.1.1). Added a shameless plug for the author's dissertation research to Section 7.12, and revised the Code Cannibalism section to include a link to the author's Github page containing many tasty scripts.
- 1.4.5 - March 6, 2014 - Made a number of clarifications, improvements, fixes, and updates to the scripting section (12) based on (wonderful) feedback from José Joaquín Atria. Thanks, José! Also, added brief discussion and URLs for various forced-alignment tools in the intro to Section 12.
- 1.4.2 - January 16, 2014 - Added Section 12.4.7, discussing nowarn, noprogress, and other ways to speed up your scripting on large datasets.
- 1.4.1 - January 10, 2014 - Fixed a few typoes and updates based on Gareth Walker's feedback. Thanks Gareth!
- 1.4 - December 9, 2013 - Added discussion of the shift in command syntax in newer versions of Praat (Section 12.3.3). Added discussion of units of intensity, addressing the Pascal vs. dB issues some of my students were having (Section 7.7.1). Added further warning, admonition, doom and gloom to Section 7.12 about A1-P0 as a measure of nasality, which, sadly, is complicated. Updated the "Reasonably Recent" Praat version number to encourage people to upgrade. Added quick code example for finding odd and even numbers in a Praat script to Section 12.4.8. Added a brief discussion of "while" loops (Section 12.4.3).
- 1.3.6 - October 2, 2013 - Added discussion of the smoothing of the intensity line and its relation to the pitch track in Praat.
- 1.3.5 - March 22, 2013 - Updated License for the Manual. A few typoes squashed.
- 1.3 - September 28th, 2012 - Added Section 7.7, Manipulating Duration with explanations of how to slow and speed sound files
- 1.2.5 - August 10th, 2012 - Added discussion of Praat Picture for more complex displays
- 1.2 - May 19th, 2012 - A few small tweaks, added a section on measuring Voice Onset Time
- 1.1.1 - January 2nd, 2012 - A few other small corrections based on Paul Boersma's feedback.
- 1.1 - January 1st, 2012 - Updated with Paul Boersma's valuable feedback and a variety of small corrections.
- 1.0.2 - July 14th, 2011 - Removed some typoes and fixed other small issues.
- 1.0.1 - July 10th, 2011 - Updated to include instructions on removing scripts from Buttons.
- 1.0 - July 10th, 2011 - Version created for LSA Institute Workshop on Praat.

2 Contributors

- **Will Styler** - The author (and editor) of the document, and sole maintainer from 2011-2022.
- **Your name here!** - Go to `https://github.com/stylerw/usingpraat` to contribute text or other documents.

3 Introduction

Praat is a wonderful software package written and maintained by Paul Boersma and David Weenink of the University of Amsterdam. Available for free, with open source code, there is simply no better package for linguists to use in analyzing speech.

Unfortunately, much of the existing documentation for the software is just that, software documentation, and is not designed to help linguists (who may not necessarily consider themselves to be "phoneticians" or have a strong phonetics background) get the measurements and make the changes that they need and desire for their research.

As such, rather than introducing each menu item and function as such, I've instead chosen to describe how to do some of the tasks that linguists want to do without assuming a strong phonetics or programming background. Then, eventually, we'll discuss some of the more complicated measures and tricks one can perform with Praat.

Of course, no one workshop can discuss the myriad features present in Praat, nor cover all of the quirks of the package, but this workshop will hopefully leave you feeling more at home in Praat, and give you an opportunity to go forth and explore further on your own.

3.1 Versions

This guide is kept up to date, so that all code and commands will work in the latest version of Praat (6.0, at the time of writing). Download the latest version right now before getting started with this guide, and in the future, you should start any given project by downloading the latest version from `http://praat.org`. Paul Boersma is *exceptionally* good about not breaking old functionality, and **not using the latest version of Praat is the most common cause of issues when trying to run scripts or following online tutorials, so please do stay up to date!**

All screenshots here are from Praat 5.3.60 running on Mac OS X, but your copy on your platform should not differ significantly. Unless otherwise specified, workflows for making measurements and manipulations do not differ significantly across platforms and versions, and any version-specific issues are mentioned there.

3.2 Other Resources

Although this guide aims to be painfully comprehensive, there are many other resources available for helping with Praat. The first step for dealing with any issue is Praat's built in help guide, accessible from the upper right corner of most windows in the program. You'll be best served by starting with "Intro" and moving from there.

There are also a variety of tutorials for Praat available online, and the Yahoo! Groups "Praat-users" group, whose archives can be searched at the below link:

`http://uk.groups.yahoo.com/group/praat-users/`

You will want to search the archives before posting, as there are likely a great many people who have had your question before in the history of the software.

3.3 Getting and Installing Praat

Praat can be downloaded from `http://www.praat.org`, and its installation will vary depending on your platform.

- Mac OS X - Just drag Praat into your Applications folder.
- Windows - Download the installer and run it, and a link to the program will be placed on your desktop.
- Linux - If you're running Linux, you'll be able to figure out the install on your own. Many distributions have Praat as an installable package in their repositories, but check the version numbers, as you won't want anything older than 5.2.x, and many scripts will want versions newer than 5.4.

3.4 Using this guide

This guide is meant to be useful both to people just starting with the Praat program, and to experienced users who are getting deeper into Praat scripting.

If this is your first exposure to Praat, you can and should ignore all of the "Scripting" sections and "Script tips!" bubbles. That's meant for people who want to automate what they're doing over large datasets. You may want to go there someday, but you'll get more from understanding the interface now that you will from getting behind the scenes with scripting.

4 About Praat

4.1 Praat Windows

Once you've opened Praat, a variety of windows will open automatically, and there are many other windows which will pop up when using the software. It's best to discuss these now so we can refer to them by name later when discussing the path to certain commands.

The **Praat Objects window** (Figure 1) is where you'll start most workflows, using this menu to open, create and save files, as well as to open the various editors and queries which you'll need to work with sound files.

The **Editor window** (Figure 2) is where you'll spend most of your time, and can be accessed by selecting a sound and choosing "View & Edit". When examining a sound file, the editor window will show the sound's waveform on the top and a spectrogram on the bottom, and the cursor will allow you to take selections and measurements. The menus along the top will allow you to show and hide different bits of information (formants, pitch, intensity), as well as to make more detailed queries. When working with other types of Praat objects (e.g. spectra), the editor window will allow you to query those objects as well.

When you make a query, either in the editor window or from the objects window, the **Info Window** will pop up with your results. You can also print to this window when scripting in Praat (see

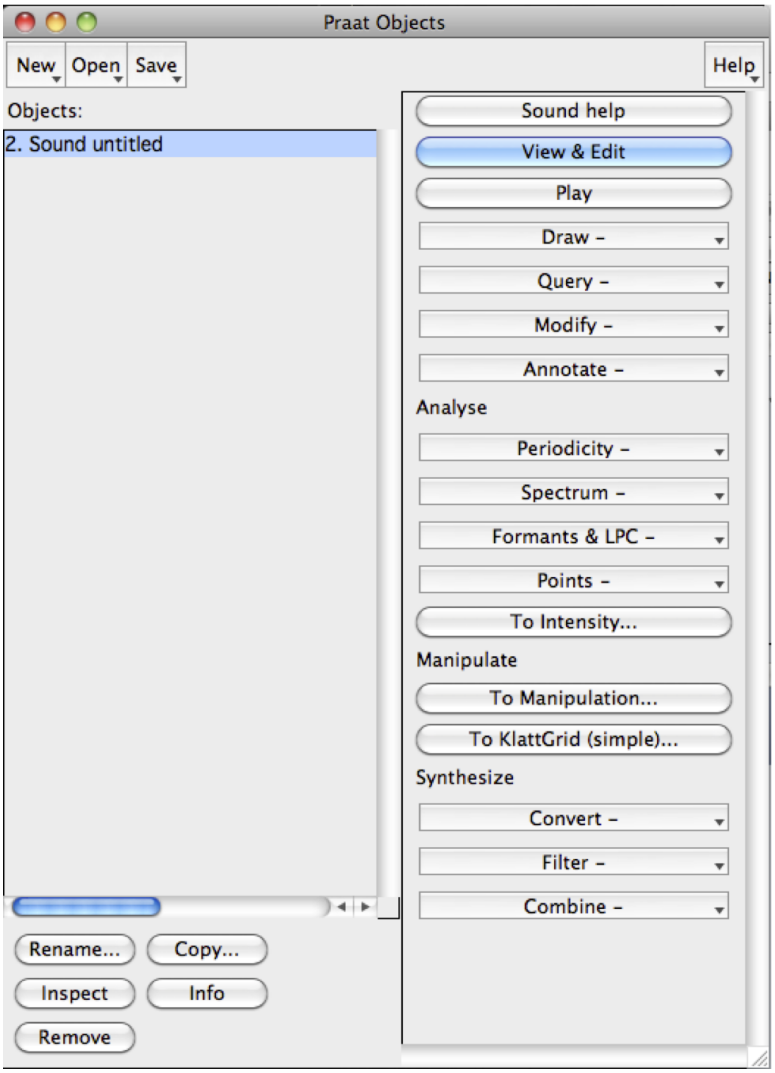

Figure 1: The Praat Objects Window

Section 12). Note that information printed here will not necessarily be saved, and running a new query will overwrite it by default.

The **Praat Picture window** (shown towards the end of the document in Figure 10) is used to create and display publication-quality images, and is open by default when you start the program. For detailed information about using the Pictures window and why it exists at all, see Section 9.

Knowing the names of all these commands allow us to more easily describe the commands to use when working with Praat. For instance, if this guide says that you get the duration of a sound by using:

> Objects → Query → Query Time Domain → Get Total Duration

It means, roughly, "Go to the Objects window, Choose "Query", then from that submenu choose "Query Time Domain", then "Get Total Duration".

5 Recording Sounds

To record sound using Praat, you'll want to plug in your microphone, sound card, or external ADC (Analog-Digital Conversion) box to your computer *before starting Praat*, and then…

> *Objects → New → Record Mono*

This will pull up a recording menu which allows you to choose a sampling frequency (the default, 44100 Hz, is fine for most purposes), as well as the microphone or other sound source. Press *Record* to record, and *Stop* to stop, being careful that the sound level bar stays within the "green" range to avoid clipping. Once you've made a recording, name it and choose *Save to list*, and it will now show up in the Praat objects window where it's ready for editing.

If you don't see a green bar (indicating that Praat hears you) while you are recording, try changing the *Input source* on the left side of the SoundRecorder window. If this doesn't help, go to the computer's sound control panel to ensure the proper microphone is selected, and that the input volume is not turned way down.

Praat is limited in its recording length by a pre-set recording buffer. To record longer sounds, you can either change the buffer size in *Praat → Preferences → Sound Recording Preferences* or you can use Audacity (free, available from `http://audacity.sourceforge.net/`) to record the session and then import the sounds into Praat (see Section 6) afterwards for analysis and manipulation. I tend to recommend Audacity, as it's a bit more robust for this purpose, and the quality is identical.

5.1 Mono vs. Stereo Recording

You do have the option of recording a "Stereo" signal (as opposed to 'Mono' or Monaural), which records not one but two signals, using two 'channels', by using:

> *Objects → New → Record Stereo*

Although stereo sound allows us the ability to localize sounds in side-to-side space, in general, phonetic research works on mono signals, as most microphones only capture one channel, and the human vocal tract for a single speaker contains only one sound source. The exceptions are if you require the sound localization (e.g. being able to distinguish a talker on the right from one on the left) that stereo signals provide, or if you need to record two data sources in a synchronized way, for instance, recording two speakers (each wearing a different headset microphone), or recording one speaker's audio from a microphone with an electroglottograph (EGG) signal capturing vocal fold movement on the second channel.

In these cases, where you're specifically capturing two sources, you'll likely want to set up your recording so that each data source is on a single 'channel' ('L' and 'R'). This has the advantage of automatically synchronizing the two signals, such that the start and end of recording for both is identical, and keeping them together in a single file. The tracks can be split and combined later (see Section 8.2).

But unless you have two distinct sources, or , even if your microphone/computer/setup is capable of stereo recording, *you should likely record as mono*. If you don't need two independent signals, by recording as stereo, you're just doubling your sound files' size with no particular benefit.

6 Opening and Saving Files}

6.1 Opening Files

If you already have a sound file recorded that you'd like to open (recorded in .aiff, .wav or .flac format), there are two ways to open it in Praat. If you're using Praat on OS X, you can drag supported files onto the Praat icon in the dock. However, if that doesn't work, or if you're on a different platform:

Objects → *Open* → *Read from File…*

Then use the next dialog to find the files you're interested in on your hard disk. Once you've loaded the files, they'll appear in your objects window for further use. Note that other files created by Praat can be opened in the same way.

Tip: Praat can't open .wma, .mp3 or .m4a audio files. To convert these easily to .wav files *en masse*, download iTunes, set it to "import" files into the .wav format in Preferences, and use *iTunes* → *Advanced* → *Create .wav version*.

6.1.1 Working with longer sound files

Praat has historically had trouble working on sound files more than 20 minutes or so long, and if you're using a 32-bit version or have little available memory, you may experience frequent out-of-memory errors working with large files unless you use the *Objects* → *Open* → *Open long sound file…* option.

However, *if you can possibly avoid using LongSound objects, avoid using LongSound objects*. Certain analyses, menu commands, and other miscellaneous functions cannot be applied to LongSound

objects, and as computers evolve, they're less necessary. There are still times, particularly when you experience slowdowns when trying to seek to or play a specific area of a long file, where using LongSound is advantageous or necessary, but in general, you'll have a nicer experience importing as conventional sound files until and unless you experience problems.

To avoid these issues, it's recommended to cut your files into chunks shorter than an hour, either using Audacity or by editing the Long Sound object as described in Section 8.3.

For Mac users, this is a mostly a non-issue if you have a 64-bit Mac and download a recent 64-bit build for OS X. You may still see slowdowns using the editor working with Sound files which are very large, but you'll be able to do what you need.

6.2 Playing Files

Playing back files is generally straightforward, using either…

Objects → Play

… or the gray bars at the bottom of the editor window as described below. Note, however, that if you play from the Objects window, the sound will continue to play its entire duration. To stop the sound playing, you'll want to open the sound in the Editor Window, and then immediately close it.

Danger!

On recent Apple Hardware, as well as on systems where the speakers are 'turned off' until a signal is sent, you may find the first part of the sound is cut off. To fix this, use Praat -> Preferences -> Sound Playing Preferences and adjust the value of 'Silence before'.

\end{tabular}

6.3 Saving Files

Praat does not save *anything* by default, and until you save files explicitly, opened or edited versions of the files will exist fleetingly in the objects window. For emphasis, **unless you save files explicitly, they will disappear completely and unrecoverably when Praat is closed**.

To save a file, select the file in the Objects window, then…

Objects → Save → Save as ____ file

For sound files, you'll likely choose *Save as WAV file*, but for the other types of files (TextGrids, formant objects, pitch objects, etc), you'll save them as text files.

As this can be very tedious, you might consider downloading and installing a Praat script (see Section 12) which saves all the objects in the objects window at once.

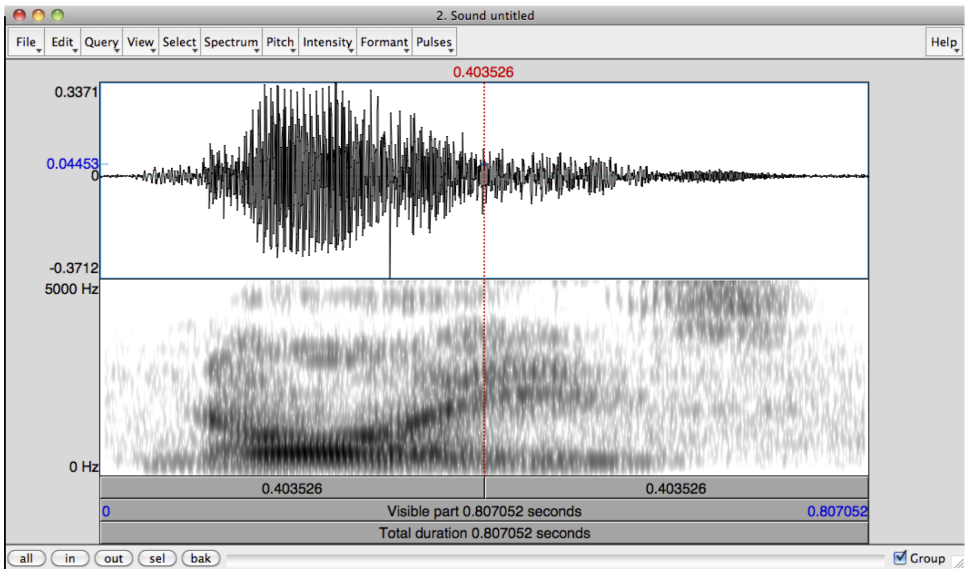

Figure 2: The Praat Editor Window

7 Phonetic Measurement and Analysis in Praat

7.1 Working with Praat Waveforms and Spectrograms

Once a sound has been recorded or opened, you'll spend much of your time interacting with the sound by means of the Editor window. To open a sound in the editor window, select the sound and then…

Objects → *View & Edit*

You're immediately presented with an editor window (like that in Figure 2), showing the waveform of the sound, and if the sound is sufficiently short, a broadband spectrogram showing the spectral energy of the sound over time. In addition, you might also be presented with a series of red dots (representing formants), blue lines (representing the speaker's pitch), and a yellow line (representing intensity). These can be enabled and disabled in the *Editor* → *View* → *Show Analyses* menu.

If your window is showing two waveforms, then you've opened a stereo sound file. The spectrogram displayed will show you the two channels combined. Everything below will remain true, and there's no fear to be had, but particularly if the two channels show two types of vastly different data, the resulting data could be a bit odd, and you might see Sections 5.1 and 8.2 for instructions on separating the channels.

Clicking within this window will place the cursor on the waveform and spectrogram. If you click within the editor window, the cursor will spawn two dotted lines. A vertical bar shows the time within the sound where you clicked (labeled at the top in seconds) and, if you clicked within the spectrogram, a horizontal bar shows the frequency at the cursor (labeled on the left in red). If the pitch or intensity tracks are displayed where the cursor is placed, values at the time the cursor

represents are given on the left side of the editor window.

In addition, you can click and drag (or use the *Select* menu) to select portions of the sound. The time of the start and finish of the selection will be displayed in red, and the duration of the selection (in seconds) will be displayed in the top of the bar.

To play a sound in the editor window, use the three gray bars at the bottom of the editor window. The bottom-most bar (*Total Duration*) will play the entire sound. The middle bar (*Visible Part*) will play only the visible portion of the sound. The different sections of the top bar (split by the cursor or selection), when clicked, will play the corresponding pieces of the visible portions of the sound file. Hitting also plays the visible portion of the file.

Obviously, to view some analyses and to get a closer look at your data, you'll need to use the five buttons in the bottom left corner of the window. As you can imagine, ***all*** shows the entire file, ***in*** and ***out*** zoom in and out, ***sel*** zooms to make the current selection fill the window, and ***bak*** zooms back to the previous zoom level. For longer sound files, in order to view analyses like the spectrogram and formants, you'll need to zoom in to show only a pre-defined amount of time.[1]

The ***Group*** setting in the bottom right corner of the window will ensure that if two sounds are open in two Editor windows at once, they'll share the same zoom characteristics. This is best used to compare two versions of the same file, say, an original versus one with an acoustic modification made.

All of the measures discussed in this section will use the Editor window, and you will spend much of your time working with Praat here, so any time spent gaining familiarity will be repaid tenfold.

7.1.1 Pulling out a smaller section of the file for analysis

Although zooming in and out will get you most of the way there, it's often useful to isolate a section of a sound (usually a single word or vowel) into a different Sound object. To do this, select a portion of a sound, say, a vowel, and then:

Editor → *File* → *Extract Selected Sound (time from 0)*

This will create a new sound in the Objects window, containing just the selected part of the original sound. The *(preserve times)* option (in the same *Editor* → *File* menu) just keeps the timecode on the extracted sound the same as in context (so, if the vowel starts at 0.245 s, the extracted sound file will start at 0.245 s).

This can also be done from the objects window using *Objects* → *Convert* → *Extract part…*, if you know the start and end time of the portion you'd like to extract.

[1] This amount of time can be changed in *Editor -> View -> Show Analyses -> Longest analysis*. 20 seconds is a sane value for most modern computers, much higher will cause your system to lag when viewing files.

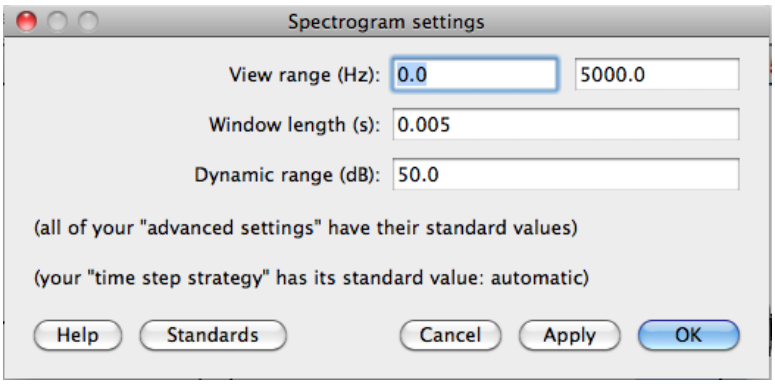

Figure 3: Praat's Spectrogram Settings

7.2 Adjusting the Spectrogram settings

Although the basic 0-5000 Hz broadband spectrogram will suffice for many uses, it's useful to be able to change those settings. To make changes to the spectrogram settings…

Editor → *Spectrum* → *Spectrogram Settings*

This will pull up the Spectrogram settings window (like that in Figure 3)

The most important settings here are the **window length** and **view range**.

View range controls how much of the spectrum is visible. For speech, you'll likely be interested in the range from 0 to 5000 or 6000 Hz, but if you're examining fricatives, you might want to look as high as 15,000 Hz. If you're looking at music, you may focus on the area from 100 to 2000 Hz. Either way, this is how you set which part of the spectrum you care about.

If your sound files have a relatively small or large **dynamic range** (the difference in volume between the loudest and quietest frequencies or times), or if your spectrograms seems too light or too dark, you may want to adjust the dynamic range setting, but 50 dB is usually fine for most purposes[2].

Window length (given in seconds) controls how large of a chunk of the sound Praat will examine when trying to find the frequencies present at a given moment in the signal.

This reveals a fundamental tradeoff in signal processing: You can either have precise information about frequency, or precise information about time. If you examine large chunks of the signal, you'll be able to see patterns repeating over and over, and will have a *very* firm understanding of their frequency. However, if you're looking at a large chunk, sudden changes (say, the end of a stop consonant, or a vowel-to-nasal transition) will seem fuzzy. On the other hand, if you only examine a very small chunk of data for each moment in the spectrogram, you'll have a very good sense of time, able to make precise claims about starts and stops, but you'll have very little accuracy in discussing frequencies[3].

[2]Like so many things, the key to understanding the usefulness of these settings is sitting down and playing around with them a bit. "Fiddling around with settings" is one of the best things a novice Praat user can do with 20 minutes

[3]Another analogy is a camera. If you open the shutter for a longer period of time, moving objects are blurry, so

This choice, of "wide window, good frequency, bad timing" vs. "small window, bad frequency, good timing" is the difference between Broadband and Narrowband spectrograms, and switching between them is a useful technique for analyzing a novel linguistic signal.

7.2.1 Narrowband vs. Broadband Spectrograms

Praat defaults to showing a Broadband spectrogram, which is excellent for viewing the temporal structure of the sound and for seeing vowel formants, but sometimes, you'll want to look at harmonics and F_0 instead. To do this, you'll ask Praat to provide you with a narrowband spectrogram. To do this:

1. *Editor* → *Spectrum* → *Spectrogram Settings*
2. Set the *Window Length* to 0.025 (or the narrowband window length of your choosing)
3. Click OK

Now, harmonics should clearly be visible in the spectrogram. Note that this is often the most reliable way to quickly visualize pitch variation, as it's immune to pitch-tracking issues.

To return to a broadband spectrogram:

1. *Editor* → *Spectrum* → *Spectrogram Settings*
2. Set the *Window Length* to 0.005 (or the broadband window length of your choosing)
3. Click OK

7.3 Measuring Duration

As you might expect, measuring duration is quite easy. Once the sound file is open in the Editor window:

1. Select the portion of the file you'd like to measure (e.g. the vowel) with the cursor
2. Read the duration of the selection (in seconds) from the duration bar along the bottom of the Editor window OR
3. *Editor* → *Query* → *Get selection length* and read your selection in the info window

If you'd like the duration *of an entire file,* just select the file in the Objects window and:

Objects → *Query* → *Query Time Domain* → *Get Total Duration*

7.3.1 Measuring Voice Onset Time (VOT)

"Voice Onset Time" (VOT) is the time between when the stop is released and when the voicing of the following vowel begins. Measuring this time, which can be positive (say, for the English

you don't know what any of them were doing at the moment you hit the shutter. If you only open the shutter for a moment, even fast motion is stopped, and you know precisely what was happening when the shutter was tripped.

voiceless aspirated stop FIXMEta), around zero (for the English "voiced" stop /d/, or, more commonly, the voiceless unaspirated [ta] around the the world), or negative (for fully voiced stops, where voicing starts before the stop is released, as found in most non-English languages). Many languages classify their stops largely based on Voice Onset Time, and it's often an excellent, more gradient empirical measure of the "voiced/voiceless" phonological distinction.

Measuring Voice Onset Time (VOT) is very easy to do in Praat, as it's just a duration measurement between two set points, the release of the stop and the start of voicing.

1. Find the stop release
2. Find the start of voicing
3. Select the span between these two points
4. Read the duration of the selection (in seconds) from the duration bar along the bottom of the Editor window OR
5. *Editor* → *Query* → *Get selection length* and read your selection in the info window
6. If the start of voicing came before the stop release, the VOT is negative. Otherwise, the VOT is positive.

In general, voiced sounds (in languages other than English) will have a VOT which is negative, voiceless unaspirated sounds will have a VOT which is around 0, and aspirated sounds will have a positive VOT.

7.4 Examining and measuring F_0/Pitch

F_0 and Pitch can be measured in a number of ways in Praat, more and less reliably.

7.4.1 Measuring F_0 from a single cycle

The surest way to get an accurate F_0 for a single cycle is to open the file in the Editor window, then:

1. Zoom in to the point where you can see individual cycles in the sound file
2. Select one complete cycle, as accurately as possible, thus, giving Praat the period in seconds (t)
3. Praat will calculate the frequency of the sound in Hertz in the top bar, giving it in the format (__/ s). Use the zoom *sel* button to zoom in if you can't see the frequency readout.[4]

7.4.2 Viewing Pitch via a narrowband spectrogram

The most reliable way of getting a sense of the pitch through the course of the word in Praat is by examining a narrowband spectrogram with a reduced visible range (0 - 400 Hz for speech).

[4]This is going to be an accurate number (so long as you gave an accurate period), but you're welcome to calculate it yourself to make sure. f = $\frac{1}{t}$, where *t* is the period in seconds

This can be done by editing the spectrogram settings as described in Section 7.2. The contours of the harmonics will accurately represent the pitch contours of the voice during the word, and doing this will give you a sense of the contour before using the Praat pitch tracker for more precise measurement.

7.4.3 Using Praat's Pitch Tracking

Praat does have the ability to provide a pitch track in the editor window. To enable the pitch track in the Editor window:

Editor → *Pitch* → *Show Pitch*

At this point, a blue line will be placed on top of the spectrogram representing the pitch, where Praat can find it. Once the pitch track is placed, you can use the cursor to check the pitch at any given point in the word. Just place the cursor and look for the middle blue number on the right side of the window. You can also place your cursor at a given point in the file and *Editor* → *Pitch* → *Get Pitch*. Running *Editor* → *Pitch* → *Get Pitch* when a chunk of the sound is collected will return the average pitch during that selection.

7.4.4 Improving Pitch tracking by changing the Pitch Settings

It's worth noting, though, that Praat's pitch tracking can be quite finicky. You will often see it jump up and down, doubling and halving the actual F_0, and in many cases, especially where the speaker is at all creaky, the pitch track will drop out altogether. This does not represent any specific failing of the software, but instead, comes from the variability and noise inherent in actual phonetic data. Part of the strength of Praat's approach is that you as a user can help Praat improve its pitch tracking for a given file or speaker by changing some of the Pitch settings.

So, in order to do any serious research using the pitch track, and to avoid some of the problems discussed above, you may need to adjust some of the pitch settings, to help Praat's pitch tracker better reflect the speaker's voice. To do so:

Editor → *Pitch* → *Pitch Settings…*

Then adjust the settings as follows:

- *Pitch Range (Hz)*
  - Set the pitch range to a reasonable range (50 - 400 for general usage, going much higher for song or children's voices). If you have a good idea what the speaker's actual range is (taken from a narrowband spectrogram, for instance), set the minimum to just under the speaker's lowest F_0 and the maximum to just over their highest pitch excursion. Changing the minimum pitch will also change the smoothing of the intensity line, when displayed.
- *Unit*
  - Here you can choose your unit of choice for the display of the speaker's F_0, ranging from Hertz to Semitones to auditory-scaled measures like logHz or mel.

- *Method*
  - You'll usually keep this parameter set to "autocorrelation", but you can switch to "cross-correlation" to see if it improves your pitch track.

You may want to tweak the advanced settings as well. To do so:

Editor → *Pitch* → *Advanced Pitch Settings…*

Then adjust the settings as follows:

- *Silence threshold*
  - This is the amplitude threshold for what Praat considers to be speech, relative to the peak amplitude in a file. If Praat can't find ANY pitch in a quiet file, adjust this setting.
- *Voicing threshold*
  - Praat uses this value in its algorithm to help it decide whether voicing is present. If Praat is finding voiceless portions of the word as voiced, raise this number. If Praat isn't detecting voicing that's there in the signal, lower it. This is mostly important when working with data with either very large or very small amounts of background noise, and can often be left alone.
- *Octave Jump cost*
  - Changing this value affects the algorithm's decision about whether a jump in F_0 is reasonable. Larger values discourage abrupt changes in F_0. Increase this number if you are getting pitch-doubling, decrease it if you are failing to track actual rapid changes in F_0.
- *Voiced / unvoiced cost*
  - Increasing this number will make Praat more reluctant to claim a transition between voicing and voicelessness. Turn this up if your pitch track is cutting in and out more than is reasonable.

Danger!

Praat's pitch tracking is a good way to get a rough idea of what's going on with the speaker's F_0, but relying on it to give you sane measures is not wise, especially in scripts. Make sure you sanity-check any measures which seem unreasonable against single-cycle F_0 measurements or against harmonic frequencies, and that you throw out anything completely ridiculous.

\end{tabular}

7.4.5 Scripting: Creating a Pitch Object

Again, if you're just learning Praat for the first time, skip right past these "Scripting" sections. This is down-the-road stuff for automating these measurements, not where you want to focus today!

When scripting, you may want to create a Pitch object (select the sound, then *Objects* → *Analyze Periodicity* → *To Pitch*, specifying the proper range) so that you don't need to open the editor to measure pitch. Once a pitch object is created, you can instead select the Pitch object and run *Objects* → *Query* → *Get value at time…* to find the pitch at whatever time you'd like. Pitch in a pitch object is calculated in the same way as in the Editor window, so the same disclaimers apply.

Script Tip!

Build in sanity-checks for pitch! Specify the highest and lowest reasonable pitches at the start of the script, use them in the creation of pitch objects and elsewhere, and specify that any values out of this range are wrong, and should be measured again. 10 or 600 Hz F_0 measurements won't be a problem anymore!

\end{tabular}

7.4.6 Getting Maximum, Minimum, and Average pitch for a section of speech

This is easy.

1. Select the portion of the sound for which you'd like the Maximum, Minimum or Average Pitch
2. Select the proper command for your task from the *Editor* → *Pitch* menu.

Note that *Editor* → *Pulses* → *Voice Report* will give this information as well.

7.5 Measuring Pulses, Jitter, Shimmer, and Harmonics-to-noise ratio

As a part of its pitch-handling system, Praat includes the ability to find individual glottal pulses in a signal and to analyze the pulses as part of more complex analyses. To view these pulses, *Editor* → *Pulses* → *Show Pulses*, and they'll then display on top of the waveform in your file. Although the pulses themselves are mostly only useful in scripting, the *Editor* → *Pulses* menu contains one of the more useful commands in the program:

Editor → *Pulses* → *Voice Report*

To use this command, simply select a voiced section of the sound, then *Editor* → *Pulses* → *Voice Report*. An information window will then pop up providing you with a variety of useful measures. In addition to maximum and minimum pitch (with additional statistics), you will also be given the **jitter**, **shimmer**, **harmonics-to-noise ratio (HNR)**, and the **noise-to-harmonics ratios** for the selected portion of the sound.

Jitter is a measure of the periodic deviation in the voice signal, or the **pitch perturbation** of the signal. Put differently, each cycle of speech with a given F_0 should, in a perfect world, have the same period. The jitter in a person's voice is how much one period differs from the next in the

speech signal. This is a useful measure in speech pathology, as pathological voices will often have a higher jitter than healthy voices[5].

Shimmer (**amplitude perturbation**) is similar to jitter, but instead of looking at periodicity, it measures the difference in amplitude from cycle to cycle. Once again, this is a useful measure in speech pathology, as pathological voices will often have a higher shimmer than healthy voices (although again, both healthy and unhealthy voices will have *some* shimmer).

Harmonics-to-noise ratio (HNR) and Noise-to-harmonics ratio are both measures of the amount of periodic noise compared to the amount of irregular, aperiodic noise in the voicing signal. Because the aperiodic noise often represents frication in the vocal tract, the HNR will go down significantly with hoarse or breathy speech, and other laryngeal pathologies will lower the HNR further still.

7.6 Measuring Formants

Praat has several methods of built in formant measurement. Of course, the easiest way to examine formant heights is by simply looking at a broad-band spectrogram and using the cursor to find, roughly their frequencies. However, "eyeballing it" won't pass scientific muster, and is more time consuming than using Praat's built-in Linear Predictive Coding (LPC) algorithms as a tool to help to find them.

7.6.1 Using the Formant tools in the Editor window

When you open a sound file in the Editor window, you can choose to have Praat calculate and display where it thinks that the vowel formants are (*Editor* → *Formants* → *Show Formants*). This will overlay a series of red dots onto the image which represent peaks in the series of LPCs which Praat has run at each moment in the word.

This formant track can be queried at any time in a variety of ways, all accessed through the *Editor* → *Formants* menu. If you're interested in a single formant's height, you can place the cursor where you want a measurement and choose *Editor* → *Formants* → *Get formant…*, but it's often more efficient to use *Editor* → *Formants* → *Formant Listing,* which will give you heights for F1, F2, F3, F4, along with the timepoint at which the measures were taken.

In addition, if you're interested in formant bandwidth, bandwidth for the first four formants can be taken using the *Editor* → *Formants* → *Get __ Bandwidth* commands.

For hand measurement, using *Editor* → *Formants* → *Formant Listing* and sanity-checking by visually inspecting the formants on the Spectrogram will usually produce reasonable results, but there are ways to improve Praat's formant-picking performance for a given speaker.

[5] It's worth noting that *all* voices will have some jitter, as the vocal folds are imperfect, and it will be far more in older speakers, smokers, etc. The presence of *any* jitter isn't a pathology, just a massive degree, and low-level jitter doesn't particularly affect speech perception.

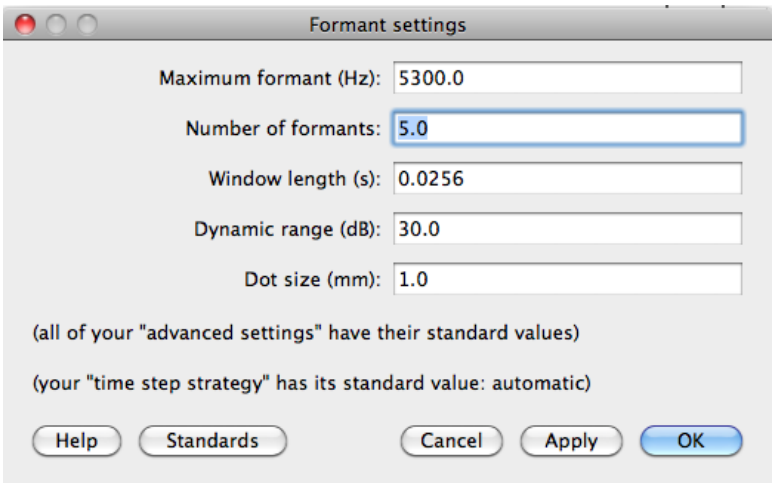

Figure 4: Praat's Formant Settings Window

7.6.2 Improving Formant Finding results

For most speakers, the default settings will suffice, but if you find Praat to be struggling with "missing" or the addition of extra formants, you'll likely find that that particular speaker's formants are more effectively measured if you make some tweaks to Praat's Formant Settings, helping the computer with its task.

To apply any of these changes, you'll want to open the Formant Settings window (see Figure 4):

Editor → *Formants* → *Formant Settings…*

Finding formants is a tricky thing. When you set out looking for areas of the spectrum where there's a bit of extra energy, you *will* find them, so the problem isn't finding peaks, but finding *the correct peaks*. To help in this task, Praat has settings dictating how many formants it will find, and how spread out those formants will be.

We typically will assume that speakers will have one formant per 1000 Hz, and thus, that there will be 5 formants in the 5000 Hz we usually worry about for speech research. Thus, "5" is the default setting for *Number of Formants*, and the highest we'll look for formants (the *Maximum Formant*) is 5000 Hz by default.

Usually, you'll only need to adjust the *Number of Formants*. Although 5 formants is a good baseline, if Praat is finding formants where there are none (latching onto a small bump between two actual formants, usually), you should lower this value down to 4 or 3. If Praat is finding too few formants (missing F2 and labeling F3 as F2, for instance), you'll want to raise this number up to 6.

If you're working with a child, gnome, hobbit, or a creature of otherwise unusually small vocal tract length, you may find that the Praat is finding non-existent formants between the speaker's F1 and F2, and missing the speaker's higher formants (F3 and F4) altogether. In this situation, you'd want to increase the *Maximum Formant (Hz)* value to tell Praat to search a bit higher up in the spectrum for formants, and perhaps lower the number of formants it's searching for. Again,

this is a somewhat speaker-specific process, and the first set of measurements for a new speaker may require some "dialing in" to get accurate measures.

Realize, though, that Praat can always find more peaks, and there are often small peaks not perceptible to humans which may still have an acoustical relevance. When there are "too many formants", Praat is not necessarily finding formants which "aren't there", but is finding additional peaks which, although present, aren't the F1, F2 and F3 peaks which we as linguists are chiefly interested in. When there are "too few", Praat is just giving you only the most prominent peaks that you've asked for. The results of Praat's formant tracker, in reality, are largely determined by what you're asking it to find, and this settings adjustment be done with a mind to what you're actually interested in.

Dot size (mm) simply controls how large the red dots in the formant display are. Although changing this can be useful if the track obscures the spectrogram, this will have no effect on your measurements.

These settings persist even once you've closed Praat, so if you make adjustments here, you'll want to return these settings to the defaults when you've finished with your odd speaker.

Danger!

No matter your settings, Praat will happily find you formants even in fricative noise or silence, and because it doesn't know how many formants it should be searching for where, it commonly merges F1 and F2 for high back vowels (where they're close together). In addition, Praat will often have issues finding a single timepoint, so if you're getting an unusual measurement, a timepoint shortly before or after may be more accurate. Always sanity-check your measurements, make sure you know what you're measuring, and during automated measurement, always run the results by a trained human first!

\end{tabular}

7.6.3 Scripting Only: Formant Objects

When scripting, you may want to create a Formant object (select the sound, then *Objects* → *Formants & LPC* → *To Formant (burg)*…, specifying the proper settings) so that you don't need to open the editor to measure formants. Once a formant object is created, you can instead select the Formant object and run any of the commands in the *Objects* → *Query* menu to get information.

Script Tip!

All of the parameters discussed above are manipulable when scripting, so build in some sanity checks to capture the common failure modes. "If F2 > 3000Hz, increase the number of formants and try again". This little step can save a lot of pain down the road.

\end{tabular}

Oddly, formant measures taken at the same timepoint and with the same settings from a formant object and from the editor window do not always agree, and in fact, can differ significantly. If measuring formants automatically, you may consider using both methods (the formant object and the editor window's formant track). It often happens that if one of the measurement tactics misses or adds a formant, the other will not, so a measure where both are in agreement is often more trustworthy than one where they disagree significantly. The author cannot explain the discrepancy, but is quite happy to leverage it extensively in his scripting.

7.7 Measuring Intensity/Amplitude

Measuring intensity in Praat is relatively straightforward, albeit with a major disclaimer.

To get the overall intensity of a sound, select the desired sound and run *Objects* → *Query* → *Get Intensity (dB)*. To get the intensity at a specific point in the sound, open it in an editor window, *Editor* → *Intensity* → *Show Intensity*, and then use the various commands available in the *Editor* → *Intensity* menu to get whatever information you desire.

By default, Praat's display of the intensity of a word is smoothed to avoid showing individual pulses in the amplitude lines, both in the editor window and in amplitude objects (when drawn or viewed). This smoothing is based on the minimum F_0 of the sound.

If you want to see something closer the amplitude envelope of the sound in Praat (where pulses show up individually as amplitude peaks), or if you want the amplitude curve to be smoother than it normally would be, you must simply adjust the minimum pitch expected by Praat. This can be done in *Editor* → *Pitch* → *Pitch Settings…*, as described more fully in Section 7.4.3. Similar smoothing/desmoothing can be accomplished when creating Intensity objects by altering the minimum pitch value in the *Objects* → *To Intensity…* dialog box.

This decrease in amplitude smoothing is particularly useful for measuring or counting quick, amplitude-based phenomena like taps and flaps.

That said, **in most recordings made for phonetic research, absolute intensity measures as given by Praat are largely meaningless**. To accurately measure the absolute intensity of a speaker's voice, a sound-attenuated booth with a calibrated sound level meter or calibrated microphone with specialized software should be used.

Relative intensity (say, between two segments or words) can be measured with an uncalibrated microphone, but is only accurate if the recording is made in a consistently quiet area, and the speaker stayed in the same general position relative to the microphone throughout the recording(s) (and wouldn't have changed much during the time between the two points of comparison). This issue is discussed in depth in Praat's user manual.

7.7.1 Units of Intensity (dB vs. Pascal)

Praat uses two measures of intensity: Pascal and dB. Pascal tend to be very small numbers (like "0.00033082594541105064") whereas dB measurements are far larger yielding numbers like "59.23328336655995". Often, when scripting or making measurements of intensity of a section

through the interface, we want information in dB, but selecting *Objects* → *Query* → *Get mean…* to get Mean intensity gives us the information in Pascal.

In order to get minimum, maximum, or mean intensity in dB, we must first convert the sound to an intensity object:

Objects → *To Intensity*

Then **select the intensity object** and run *Objects* → *Query* → *Get mean…*. This will return values in dB, as desired.

Similarly, if scripting this process:

```
select Sound soundname$
min = do ("Get minimum...", 0, 0, "Sinc70")
max = do ("Get maximum...", 0, 0, "Sinc70")
mean = do ("Get mean...", 0, 0, 0)
```

… will yield min, max, and mean intensity measurements in Pascal, whereas …

```
select Sound soundname$
do ("To Intensity...", 100, 0, "yes")
min = do ("Get minimum...", 0, 0, "Parabolic")
max = do ("Get maximum...", 0, 0, "Parabolic")
mean = do ("Get mean...", 0, 0, "energy")
```

… will yield measurements in dB.

7.8 Working with Spectra

Sometimes, you need specific details about the frequencies and individual harmonics in a sound at a given moment in time, and examining a narrowband spectrogram alone does not provide sufficient information. In these cases, you'll need to take a spectral slice for analysis. Spectral slices (also referred to as FFTs or spectra) are the result of a fast fourier transform done on a very small portion of the sound, providing you with very specific information about the frequencies present in the sound and their relative amplitudes.

Spectral slices are useful for a variety of measures of F_0, nasality, creak, breathiness, and spectral tilt, and are a crucial part of many measurement workflows.

7.9 Taking a spectral slice

To take a spectral slice, you'll need to do the following:

1. *Editor* → *Spectrum* → *Spectrogram Settings*
2. Set *Window Length* to "0.025" (effectively producing a narrow-band spectrogram)
3. *Editor* → *Spectrum* → *Advanced Spectrogram Settings*
4. Set *Window Shape* to "hamming"

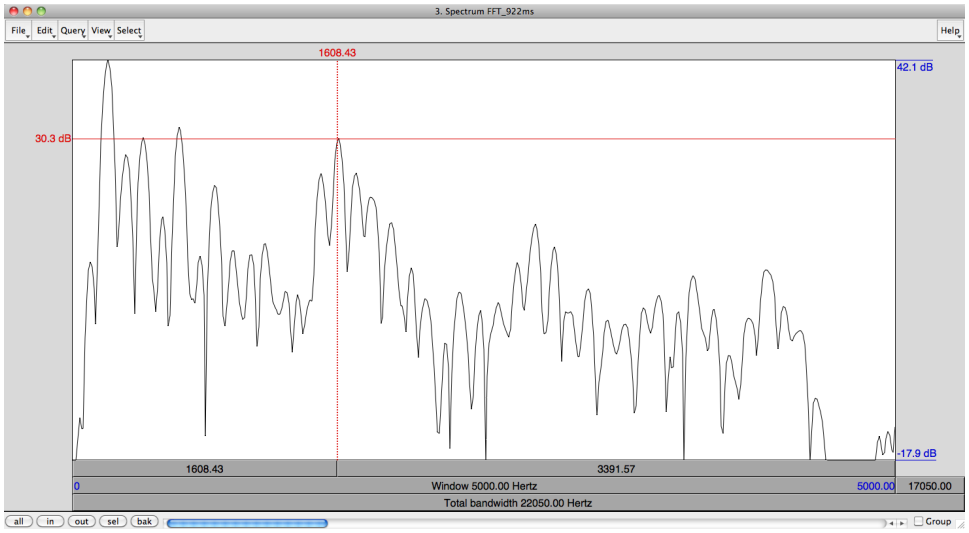

Figure 5: A spectral slice Editor window

5. Select the point at which you'd like to see the slice taken
6. *Editor* → *Spectrum* → *View Spectral Slice*

This will create a new Spectrum object, and pull up a window like that in figure 5, showing amplitude on the Y axis, and frequency (from 0 up to the Nyquist frequency) on the X axis. You can zoom in and out using the buttons in the bottom left corner, as you wish.

If you've selected a portion of the sound (rather than a single timepoint) when you use *Editor* → *Spectrum* → *View Spectral Slice*, Praat will create a spectrum representing the average characteristics across the entire selection, which isn't useful for most of the spectral measures discussed below.

7.10 Measuring Harmonic Amplitude, Frequency

Getting harmonic frequency and amplitude in a Spectrum Editor window is fairly straightforward, as clicking anywhere within the spectrum editor window will give you the frequency and amplitude measurements at the cursor. So, to find the amplitude and frequency of a given point in the spectrum, click that point and read off the amplitude (on the left) and the frequency (at the top), as shown in figure 5.

This can be done even more easily and efficiently by script, allowing you to find the highest point on a given harmonic without clicking guesswork.

7.11 Measuring Creakiness and Breathiness using Spectral Tilt

Spectral tilt is often used in phonetic research as a measure of creak. As discussed in Gordon and Ladefoged 2001 Gordon and Ladefoged (2001):

> One of the major acoustic parameters that reliably differentiates phonation types in

many languages is spectral tilt, i.e., the degree to which intensity drops off as frequency increases. Spectral tilt can be quantified by comparing the amplitude of the fundamental to that of higher frequency harmonics, e.g., the second harmonic, the harmonic closest to the first formant, or the harmonic closest to the second formant. Spectral tilt is characteristically most steeply positive for creaky vowels and most steeply negative for breathy vowels.

Spectral tilt is easily measured by finding H1 and H2, measuring their amplitudes as described above, and comparing the two numbers.

However, H1-H2 is subject to very strong interference from nasality, as described in A.P. Simpson's sternly named paper *The first and second harmonics should not be used to measure breathiness in male and female voices.* Simpson (2012), and investigators of these phenomena would do well to read that paper, and focus on other measures like Harmonics-to-Noise ratio (see Section 7.5).

7.12 Measuring Nasality using A1-P0

A1-P0 is an acoustical measure of nasality first described by Marilyn Chen in *Acoustic correlates of English and French nasalized vowels* (Chen (1997)), and later discussed in Styler 2017 (Styler (2017)). Like spectral tilt, it's a ratio measure of the amplitudes of two harmonics: A1, which is the highest harmonic peak near the first formant, and P0, which is a low frequency harmonic (usually H1 or H2) which corresponds to a low resonance in the nasal passages. See Figure 6 for an illustration of these peaks in spectra. To compute A1-P0, you need to take three main steps[6]

1. Find A1 and measure its amplitude
2. A1 is the highest harmonic near the frequency of the first formant.
    - F1 can be located as described in Section 7.6
    - The frequency of F1 will vary from vowel to vowel, tending to be lower for high vowels and higher for low vowels. The amplitude of A1, though, will not vary by vowel quality
3. Find P0 and measure its amplitude
    - P0 is a specific harmonic peak which is reinforced by resonances within the nasal passages.
    - The frequency of P0 will be specific to each speaker, and won't change from word to word (as the speaker's nasal passages are unlikely to change resonant characteristics) or from vowel to vowel.
    - The amplitude of P0 will change from word to word, depending on the degree of openness of the velopharyngeal port.
    - Although each talker is different, P0 is likely to correspond to either H1 or H2 for a given speaker

[6] Many thanks to Dr. Rebecca Scarborough, on whose handout "Measuring Nasality (using A1-P0)" this section is loosely based

- In general, for speakers with lower pitched voices, P0 will be H2, and for speakers with higher pitched voices, P0 will likely be H1
- The best way to identify this peak is by examining a known nasal vowel and a known oral vowel to determine which peak is reinforced by the nasality. You may need to examine several words before a clear winner emerges.

4. Subtract P0 from A1 to get the measurement

Because the nasal peak (P0) will be reinforced by the resonances in the nose during nasal vowel production, nasal vowels will tend to have *lower* A1-P0 values than non-nasal vowels. Across many tokens, A1-P0 can be a very good predictor of vowel nasality, however, there are several important things to keep in mind when using A1-P0 to measure vowel nasality:

1. A1-P0 only works in situations where F1 is higher in frequency than H1 and H2. Most of the time, this is true, but **for high vowels, A1 and P0 often occur at the same place in the spectrum, leaving the measurement unreliable**.
    - In these situations, A1-P1 (a second nasal resonance at ~950 Hz, described in Chen (1997)) is an oft-used measurement, but identifying P1 can be very difficult for some speakers, and even when found, the A1-P1 is rather unreliable (c.f. Styler 2017). A1-P0 is probably just as relable here.
2. A low A1-P0 relative to known oral tokens is a good predictor of nasality *across a large number of tokens*, but individual nasal/oral vowel pairs may or may not demonstrate a strong effect. It should not be used to measure the nasality of one individual token compared to another.
3. A1-P0 should only be examined in comparison with other tokens from the same speaker. An A1-P0 of -2 may be normal for an oral vowel for some speakers, but indicate extreme nasality for others. The absolute value of A1-P0 is not interpretable across speakers.
    - Different speakers have not only different baseline values, but different amounts of A1-P0 change from oral to nasal vowels. Comparison of raw or Z-Scored A1-P0 values across speakers is not wise, and will lead to untrustworthy measurements.
4. Remember, here you're relying on Praat to give an accurate measurement of the first formant, as well as F_0. Feel free to tweak the formant settings, and don't blindly trust Praat's formant tracker to find what linguists call the first formant and not some other spectral prominence.

This is a complex measure, and I (personally) have spent a great deal of time working with it and its measurements[7].

Although it is among the best acoustical approaches to nasality presently available, it is also remarkably noisy, capricious, and complex. A1-P0 nasality should be one element of your successful analysis, not the sole element, and as I have discovered, no nasality experiment is simple. Proceed with caution.

[7]For more on nasality, please see the description of my dissertation work, **?** and Styler (2017), posted at `http://savethevowels.org/will/publications.html`. I did not write 84 pages worth of Praat manual only to shy away from a completely self-serving and shameless plug in the sole earthly context in which nasality research is actually relevant.

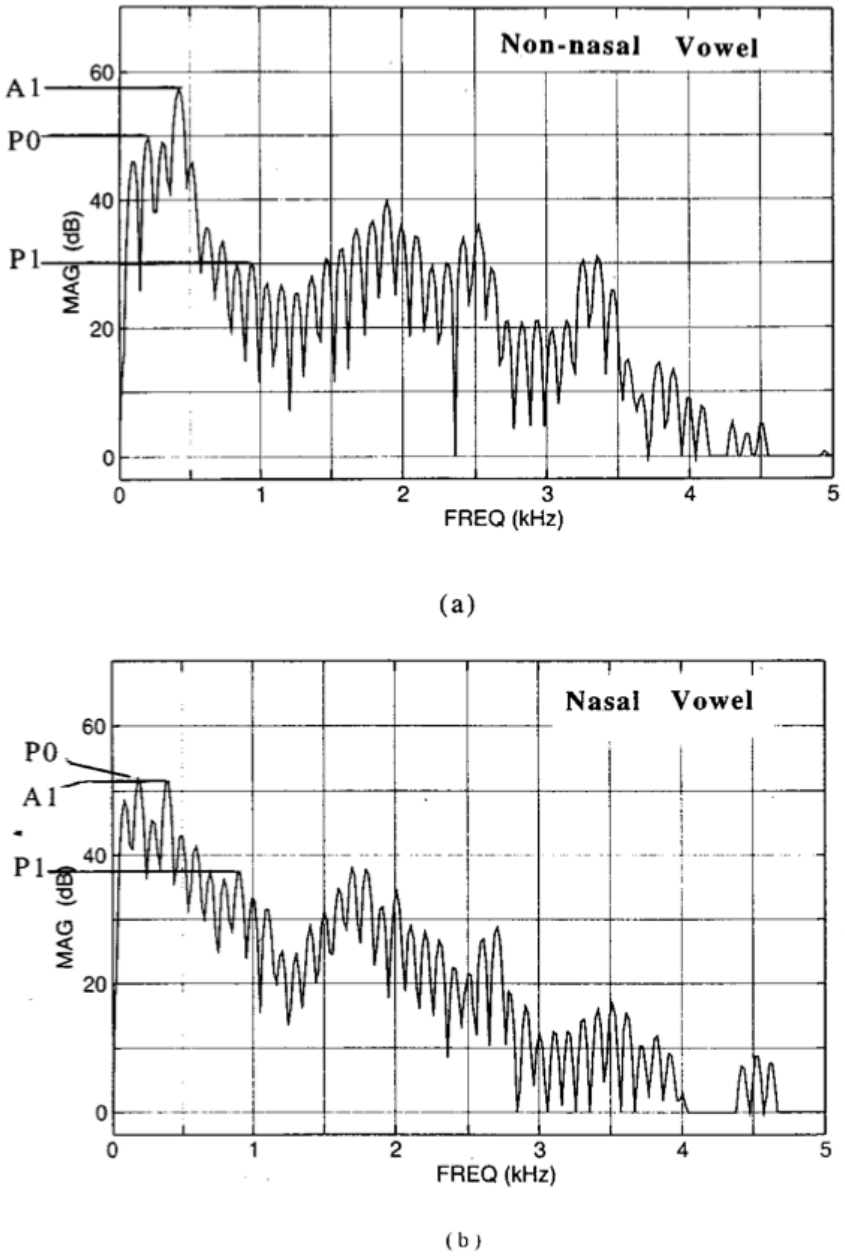

FIG. 2. (a) The spectrum of an oral vowel is compared with (b) the spectrum of a nasalized vowel from the same speaker. The spectral peaks that determine $A1$, $P1$, and $P0$ are labeled. The amplitudes are measured by using the harmonic amplitudes closest to the expected peak according to theory. $A1$-$P1$ and $A1$-$P0$ are greater for the oral vowel than the nasal vowel.

Figure 6: Figure 2 from Chen 1997 (Chen (1997)), showing A1 and P0 in oral vs. nasal vowels

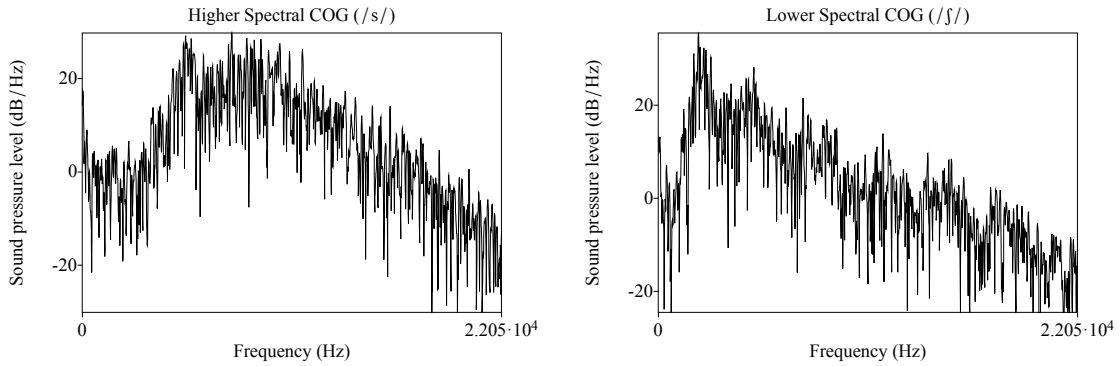

Figure 7: Two spectra showing the higher spectral COG for a token of /s/ (here, 8202 Hz) compared with the lower spectral COG for a token of [ʃ] (here, 3243 Hz)

7.13 Measuring Spectral Center of Gravity

Spectral center of gravity (Spectral COG) is useful for measuring the frequency characteristics of aperiodic sounds in speech (release bursts and fricatives, usually). It is most often used to describe the production of fricatives (with the understanding that sounds with a higher spectral COG are often produced more towards the front of the mouth), and simply measures the overall weighting of the noise in the spectrum by reporting its "center of gravity". See Figure 7 for a more visual demonstration. This measure can be useful in sociophonetic work, as well as in fieldwork, for determining the articulatory positions of different fricatives.

To measure spectral COG in Praat:

1. Open the sound in an Editor window
2. Complete the steps in Section 7.9 to create a spectral slice in Praat at the point you'd like to measure
3. Select the spectral slice in the Objects window, then *Objects* → *Query* → *Get Centre of Gravity*…

An info window will then pop up, presenting you with the spectral COG for the point represented by the spectrum.

For further information about Spectral Center of Gravity and an example of its use in fricative description and cross-linguistic comparison (showing the relationship between articulation and spectral COG), refer to Gordon et al 2002 *A cross-linguistic acoustic study of voiceless fricatives*. Gordon et al. (2002).[8].

[8]Paul Boersma has stated in correspondence that this paper incorrectly implements Spectral COG, leading to inconsistent results, and thus, should not be emulated. For more information, see Boersma & Hamann 2008 (from http://www.fon.hum.uva.nl/paul/papers/BoersmaHamannPhonology2008.pdf), Footnote 7

8 Creating and manipulating sound Files in Praat

8.1 Creating sounds from Formula

To create a sound from formula (a pure or complex tone) , you'll want to use:

Objects → *New* → *Sound* → *Create sound from formula…*

There, you can plug in a formula which will generate the sound you want. For instance:

To create a 1000 Hz puretone at 0.5 dB:

1. *Objects* → *New* → *Sound* → *Create sound from formula…*
2. Under formula, enter `1/2 * sin(2*pi*1000*x)`
3. Specify the time, sampling frequency, number of channels and such as you desire

To create a 250 Hz puretone at 1 dB:

1. *Objects* → *New* → *Sound* → *Create sound from formula…*
2. Under formula, enter `1 * sin(2*pi*250*x)`
3. Specify the time, sampling frequency, number of channels and such as you desire

To create a complex sound with 1 dB components at 250 and 1000 Hz:

1. *Objects* → *New* → *Sound* → *Create sound from formula…*
2. Under formula, enter `(1* sin(2*pi*250*x)) + (1 *sin(2*pi*1000*x))`
3. Specify the time, sampling frequency, number of channels and such as you desire

For more information, see the Praat Help guide's excellent Formulas Tutorial. To create a sound with harmonics, use *Objects* → *New* → *Sound* → *Create sound from tone complex…*

Note as well that you can use 'if' statements in formulae, as well as 'else if' statements (but not 'elif' or 'elsif'), so long as you include the relevant 'endif'. See Section 13.2 for more information.

8.2 Working with Stereo Files (Converting, Combining, and Extracting channels)

Recording as stereo files can be excellent for synchronizing and combining two independent data-streams (two speakers with two microphones, oral vs. nasal airflow, etc), but there are a few specialized commands needed when working with stereo files, each with specific use cases.

8.2.1 Converting a single stereo file into a single mono file

Sometimes, you have a stereo file which doesn't need to be Stereo. Perhaps it was recorded with a stereo microphone in a non-stereo setting, or the two tracks are (nearly) identical. In this case, because you want to preserve both channels' information, you'll want to use…

Objects → *Convert* → *Convert to mono*

This command will take the Stereo track and then *mix the two channels together to create a single mono track*. This incorporates the data from both tracks into the output mono file, so relatively little is lost. But this is *not* the best choice if you only need/want one of the two stereo channels. In that case, you'll want to…

8.2.2 Extracting a stereo file's two channels into separate mono files

If you have a stereo file where both channels need to be preserved and analyzed individually, you'll need to use…

Objects → *Convert* → *Extract all channels*

This command will take the Stereo track and then extract each channel into two separate files (_ch1 and _ch2). These are then just boring mono files, with the same start and end times, but each containing only one channel's information. If you only need one channel (e.g. a mono source was mistakenly recorded as stereo), you can identify the channel in use (1 or 2), and then use…

Objects → *Convert* → *Extract one channel…*

This will allow you to specify a channel and will output only that channel which you specified. But it is functionally identical to using 'Extract all channels' and then ignoring the other channel's output.

8.2.3 Combining two mono sounds into one stereo sound

Sometimes, you might want to create (or re-create) a stereo sound from two component mono sounds. Perhaps the goal is to add noise to one channel (but not the other) for binaural perception experiments, or to modify only one channel of a stereo sound (which you've previously extracted). To combine two mono sounds to stereo…

1. Load the two mono sounds you wish to combine into the objects window of Praat
    - Make sure they have the same sampling frequency (see Section 8.4) and length
2. Select both sounds
3. *Objects* → *Combine* → *Combine to Stereo*
4. The resulting stereo sound will be placed into your objects window with the name 'combined'.

Note that Praat will allow you to combine two signals of unequal length to stereo. In this case, the files will be aligned at the start time. But this means that a subtle change in duration (an extra few ms of pasted pause early in the file) could result in a very unpleasant desynchronization, which will happen without warning or notification.

Chaining this "Combine to Stereo" process with a "Convert to mono" step is one (somewhat roundabout) method of combining two sound files into one signal. See Section for a much more graceful approach to doing the same thing.

8.3 Cropping, Copying, Splicing and Pasting

It's not unusual to need to move, remove, or copy sound within and across different sound files, and luckily, Praat makes that relatively easy. Most file editing is done with a combination of selection, copying and pasting.

However, producing quality splices is not as straightforward as copy and pasting within a word document. Because sound waveforms are continuous, you need to make sure that all the cuts you're making occur at the same point in the cycle, namely, at the zero crossing. Failure to do so will result in loud pops or clicks in the resulting file. To ensure that you're being zero-crossing friendly, following the below steps (substituting "Ctrl" (the "control" key) for "Cmd" (the "command" key) if you're using a Windows or Linux machine):

To **copy/paste** a portion of the soundfile:

1. Select the portion of your soundfile that you'd like to copy
2. *Editor* → *Select* → *Move start of selection to nearest zero crossing* or Cmd + ,
3. *Editor* → *Select* → *Move end of selection to nearest zero crossing* or Cmd + .
4. *Editor* → *Edit* → *Copy* or Cmd + c
5. Put the cursor where you'd like the portion to go
6. *Editor* → *Select* → *Move cursor to nearest zero crossing* or Cmd + 0
7. Paste using *Editor* → *Edit* → *Paste* or Cmd + p

Following these steps will cleanly insert the snippet into the word. Given time, you'll develop muscle memory and find yourself quickly typing "Cmd + , Cmd + . Cmd + c" to copy and "Cmd + 0 Cmd + p" to paste. Copy-pasting can be done either within the same file, or between two different files in Praat.

This can also be done from the objects window using *Objects* → *Convert* → *Extract Part*.

To delete a portion of a file or to remove silence, you'll select, again attending to zero crossings, and use the "Cut" command (without pasting anywhere else):

To **delete** a portion of the soundfile:

1. Select the portion of your soundfile that you'd like to disappear
2. *Editor* → *Select* → *Move start of selection to nearest zero crossing* Cmd + ,
3. *Editor* → *Select* → *Move end of selection to nearest zero crossing* or Cmd + .
4. *Editor* → *Edit* → *Cut* or Cmd + X

Unfortunately, Praat doesn't include an easy way to trim, splice or cut portions of the file from the objects window, meaning that any scripts will have to use GUI scripting (the computer controlling the mouse/selection tool) on the Editor window, telling the Editor what to select, then to cut, etc. This works well, but is slightly less efficient than is desirable. Some progress can be made from the Objects window alone by using a combination of *Objects* → *Convert* → *Extract Part* and *Objects*

→ *Combine* → *Concatenate* to, effectively, create new sounds which resemble a cut or trimmed sound, but this can be quite counterintuitive.

If Praat won't let you copy and paste a chunk between two files, the files may need to be **resampled** to match.

8.4 Sampling rates and Resampling

Praat is very picky about the sampling rates of the files it works with, that is, the number of times per second the audio signal's power is captured in the file. When combining sounds or copy/pasting from one file to another, both sounds will need to have the same sampling rate.

To get the sampling rate of an existing file:

1. Load the sound into the objects window of Praat
2. *Objects* → *Query* → *Query Time Sampling* → *Get Sampling Frequency*
3. An info window will pop up displaying the sampling rate.

To **resample** a soundfile (e.g. change the sampling rate from 44,100 Hz to 22,050 Hz):

1. Load the sound into the objects window of Praat
2. *Objects* → *Convert* → *Resample…*
   - *New Sampling Frequency* = 22050 (or whatever sampling rate you'd like)
   - *Precision (Samples)* = 50 (don't change this number)
3. The filtered sound will be placed into your Objects window as "Sound soundname_(new sampling rate)"

8.5 Filtering Sounds

Not all changes you'll want to make involve parts of files. Sometimes (for perception experiments and otherwise), you'll find it necessary to filter your sound files.

Often, sounds files will have extraneous background noise, or in the case of particularly low-frequency-sensitive microphones, room noise not blocked by sound attenuation. In those cases, you'll want to filter the sound file to remove it:

8.5.1 Low-pass filtering

Low-pass filters are useful for simulating high-frequency hearing loss and cellular phone speech, among other things. They remove all signal above a given frequency.

To **low-pass filter** a soundfile (e.g. removing all sound above 2000 Hz):

1. Load the sound into the objects window of Praat
2. *Objects* → *Filter* → *Filter (Pass Hann Band)*

- *From Frequency* = 0
- *To Frequency* = 2000 (or whatever you'd like your highest frequency sound to be)
- *Smoothing* = 20 Hz (20 Hz is a good baseline. This controls how "soft" the cutoff is. 5 Hz is as low as you'll ever want to go for this setting.)

3. The filtered sound will be placed into your Objects window as "Sound soundname_band"

8.5.2 High-pass filtering

High-pass filters are useful for removing low frequency noise from recordings (which might seep through a sound-booth). They remove all signal below a given frequency.

To **high-pass filter** a soundfile (e.g. removing all sound below 2000 Hz):

1. Load the sound into the objects window of Praat
2. *Objects* → *Filter* → *Filter (Stop Hann Band)*
   - *From Frequency* = 0
   - *To Frequency* = 2000 (or whatever you'd like your highest frequency sound to be)
   - *Smoothing* = 20 Hz (20 Hz is a good baseline. This controls how "soft" the cutoff is. 5 Hz is as low as you'll ever want to go for this setting.)
3. The filtered sound will be placed into your Objects window as "Sound soundname_band"

8.5.3 Band-pass (notch) filtering

Band-pass filters (also called "**Notch filters**") are useful for removing sound in a very specific band of the spectrum. Notch filters are best suited for removing particular background noises with specific frequencies (e.g. computer fans, mains hum, or chair squeaking). Your bands will be wider or narrower depending on the signal you're working to cut out.

To **band-pass filter** a soundfile (e.g. removing all sound between 1500 and 3500 Hz):

1. Load the sound into the objects window of Praat
2. *Objects* → *Filter* → *Filter (Stop Hann Band)*
   - *From Frequency* = 1500
   - *To Frequency* = 3500 (or whatever you'd like your highest frequency sound to be)
   - *Smoothing* = 20 Hz (20 Hz is a good baseline. This controls how "soft" the cutoff is. 5 Hz is as low as you'll ever want to go for this setting.)
3. The filtered sound will be placed into your Objects window as "Sound soundname_band"

Finally, note the *Objects* → *Filter* → *Filter (Formula)* option, which lets you filter sounds in a much more specific way than the two options above provide.

8.6 Manipulating Spectral Tilt

Although you could band-pass out many frequency bands, alter their amplitudes, and then recombine, the most efficient way is to use:

Objects → *Filter* → *Filter (Formula)*

Then, you can use a formula like:

```
self / (1 + x/100) ^ 0.2
```

This formula[9] reduces the amplitude of the signal increasingly as the frequency increases, and can be modified to further increase the amount of spectral tilt by increasing the exponent, or decrease it by decreasing the exponent.

8.7 Pitch Manipulation (To Manipulation...)

Praat does allow you to manipulate the speaker's pitch in already-recorded sound files.

To create a manipulation object (which allows you to change a sound's pitch and duration):

1. Load the sound into the objects window of Praat
2. *Objects* → *To Manipulation...*
    - Leave *Time Step* unchanged
    - Set the pitch range to 75-600 Hz
3. Select the newly created "Manipulation (Soundname)", then *Objects* → *View & Edit*

This will open a manipulation window (like the one shown in Figure 8). It shows you the pitch track (in the center), as well as the waveform. The blue lines on top of the waveform represent pulses. This window allows you to modify the duration and pitch of the sound by creating and moving pitch points. A good first step is stylizing the pitch contour which will change the detailed contour into something more manageable.

Manipulation → *Pitch* → *Stylize Pitch*

If you still have too many points, select a few (by selecting a part of the sound) and go to *Manipulation* → *Pitch* → *Remove pitch point(s)*. If you need a different pitch point, place your cursor where you want a point and *Manipulation* → *Pitch* → *Add pitch point at cursor*.

You can now drag the individual green pitch points around to raise and lower the speaker's pitch at different points throughout the word, to great phonetic (and comedic) effect.

To save the result of your manipulations, use *Manipulation* → *File* → *Publish Resynthesis*, and a pitch-modified copy of the sound will be placed in the Objects window to be saved as usual.

Note as well that Praat can use either LPC resynthesis (discussed later) or "overlap-add" resynthesis to create the file. "Overlap-add" is actually **PSOLA (Pitch Synchronous Overlap Add) resynthesis**, and will almost always produce more natural results.

[9] Thanks to Paul Boersma and Holger Mitterer on the Praat-Users mailing list for posting and revising the formula.

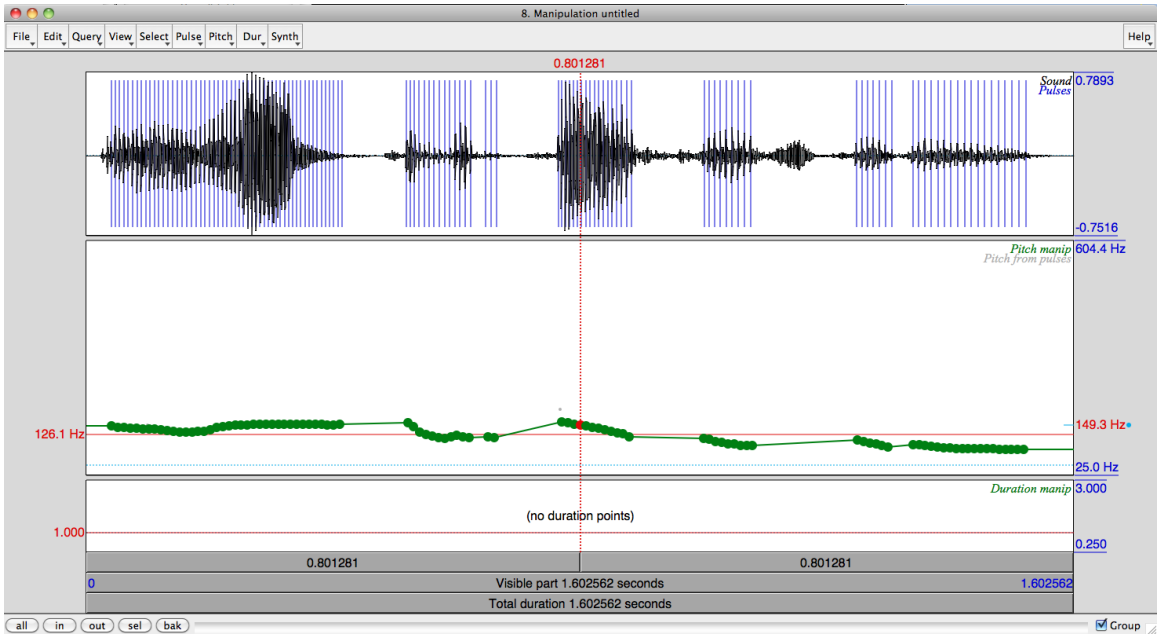

Figure 8: The Praat Manipulation Window

8.8 Matching the pitch tracks of two sounds

When combining sounds, or creating perception experiment stimuli, it can be useful to match the pitch contours of two sounds. Although one can attempt this in the Manipulation window (as described above), it's far easier to use the below procedure to match the pitch track automatically.

To give Sound A the same pitch pattern as Sound B:

1. Load both Sound A and Sound B into the objects window of Praat
2. Trim either Sound A or Sound B so that they have **exactly the same duration**
    - Praat will not allow you to swap the pitch tiers of sound files which are not the same length.
    - You could also use PSOLA to manipulate the duration, as described below.
3. Select Sound B, *Objects* → *To Manipulation…*
    - Leave *Time Step* unchanged
    - Set the pitch range to whatever is reasonable for that speaker. Usually 75-300 Hz works fine.
4. Select the newly created "Manipulation B", then *Objects* → *View & Edit*
5. Select Sound B, *Objects* → *Extract Pitch Tier*
    - This will give you "PitchTier B"

6. Now select Sound A, *Objects → To Manipulation…*
    - Leave *Time Step* unchanged
    - Set the pitch range to whatever is reasonable for that speaker. Usually 75-300 Hz works fine.
7. Select the newly created "Manipulation A" **as well as** "PitchTier B"
8. *Objects → Replace Pitch Tier*
9. Select Manipulation A (which you just combined with PitchTier B), then *Objects → View & Edit*
10. In the Manipulation Window, *Manipulation → File → Publish Resynthesis*
11. This will export a copy of the finished version into the Objects window. Save it from there.

Danger!

Praat's pitch matching feature is only as effective as its pitch tracking feature, which means that both require careful manual review of the results. Although the two sounds' pitch tracks will be significantly closer following this step, they will not be identical, and there may be artifacts and odd jumps left over. If you require the sounds to be exactly matched, match them both to a completely flat pitch generated by formula. (see Section 8.1)

\end{tabular}

8.9 Manipulating Duration (Slowing Down and Speeding Up Sounds)

Similar to modifying pitch in existing sound files, Praat allows you to modify durations, resulting in sped up or slowed down sections of existing files. First, again, you'll need to create a manipulation object:

To create a manipulation object (which allows you to change a sound's pitch and duration):

1. Load the sound into the objects window of Praat
2. *Objects → To Manipulation…* * Leave *Time Step* unchanged * Set the pitch range to whatever is reasonable for that speaker. Usually 75-300 Hz works fine.
3. Select the newly created "Manipulation (Soundname)", then *Objects → View & Edit*

This will open a manipulation window (like the one shown in Figure 8) You'll notice that underneath the pitch manipulation area, there's a small bar labeled "Duration Manip" which starts off saying "(no duration points)". To speed up or slow down a sound file:

1. *Dur → Add Duration Point at Cursor*
    - Although it says "at cursor", adding just one point allows you to manipulate duration for the whole file

2. Now drag that single duration point up or down in the "Duration Manip" area to change the speed of the file
    - Dragging the point up increases the duration, slowing the sound down, dragging it down decreases duration and speeds the sound up.

You can add multiple points and selectively speed or slow certain parts of the recording (to change the length of a vowel, for instance). In addition, as before, Praat can export the file to a wav file, which is again best done using "overlap-add" (*Objects Window → Get Resynthesis (overlap-add)*).

8.9.1 Modifying Duration by Script

To modify duration in a large number of files, you do **not** want to use Praat's manipulation interface. Instead, just TextGrid your files, and use code like the below, which would add 35 ms to a marked vowel interval's duration:

```
# Get vowel_end and vowel_start from a textgrid first!
vowel_durationms = (vowel_end - vowel_start) * 1000
duration_change = 35
finallength = duration_change + vowel_durationms
durfactor = finallength/vowel_durationms
select Sound 'soundname$'
Lengthen (overlap-add): 60, 300, 'durfactor'
```

You could also set `durfactor` to 1.5 if you wanted to make the vowel's new duration 1.5 times larger. Also know that 'Lengthen' will modify the sound in-situ, so you'll want to make a copy of the file.

Script Tip!

Don't be afraid to modify Pitch and Duration. Because Praat uses PSOLA, modifying duration and pitch is exceptionally clean, and won't warp the spectral properties of the sound, so long as Praat can find and keep a good pitch track!

\end{tabular}

8.10 Scaling and Matching Intensity

The 'Scale Intensity' command scales the entire file's amplitude to a certain average level. This can be helpful if you have a file which is too quiet to play back or when preparing stimuli for a perception experiment. This is not absolutely precise, but for most purposes, this will produce a good amplitude match, provided you attend carefully to the input files.

To scale Sound A to 75 dB average intensity:

1. Load Sound A into the objects window of Praat
2. Select Sound A, *Objects* → *Modify* → *Scale intensity…*
    - Fill in the desired value ('75') for *New Average Intensity (dB)*
3. Sound A will be modified in place, overwriting the prior version, and can then be saved.

Note that this is scaling the sound's *average intensity*, over the entire file. This is crucial, as it means that the actual intensity of, say, the vowel within a word will vary depending on the surrounding context within the file. If a file includes several seconds of silence to either side of the word, when set to 75dB average amplitude, the word itself will be higher in amplitude relative to the same word in a file trimmed to the edges of the word. Put differently, if a large proportion of the word has an amplitude near zero, the non-zero portions will need to be relatively louder to offset this silence. This can also present serious problems when scaling very different words to the same setting. Even if the silences around the words are identical, the vowel in "faith" will be relatively louder than the vowel in "rail" because of the need to scale higher to offset the surrounding, quiet fricatives in 'faith' (where there is no such need for the liquids in 'rail')[10]. So, ensure that you're conscious of context when scaling amplitude, and realize that if perfect matches within a given segment are what you're after, you'll need a more detailed approach.

It can also be useful, especially when splicing or sound combination is occurring, to be able to ensure that two sounds are of the same overall intensity.

To scale Sound A to the same average intensity as Sound B:

1. Load both Sound A and Sound B into the objects window of Praat
2. Ensure that both are closely cropped to the boundaries of the word (or that there is an equal silence surrounding both)
3. Select Sound B, *Objects* → *Query* → *Get intensity (dB)*
    - Make note of this number, it's the average intensity of B
4. Select Sound A, *Objects* → *Modify* → *Scale intensity…*
    - Fill in the value obtained in step 3 for *New Average Intensity (dB)*
5. Sound A will be modified in place, overwriting the prior (unmatched) version, and can then be saved.

Danger!

Again, you must ensure that both stimuli are surrounded by similar amounts of silence to ensure that the resulting words are actually roughly matched in amplitude. This has forced me to re-make stimuli at great personal cost. Learn from my pain.

\end{tabular}

[10] To overcome this and ensure that the vowel is exactly 75dB, consider separating the vowel, scaling it to 75dB, and then re-combining with the surrounding word

Finally, remember that scaling file amplitude to a given value does not ensure that it plays back at that same amplitude during experiments. To claim an exact playback amplitude, you'll need to use a calibrated headset and precise, non-listener-controlled amplitude control.

8.11 Concatenating Sounds

Although you can paste the contents of one file at the beginning or end of another within the editor window, this is terribly inefficient and error-prone for more than one or two tokens. To directly concatenate two or more sounds sounds, that is, to place them within a single sound file such that one plays directly after the other, Praat offers the *concatenate* command (*Objects* → *Combine* → *Concatenate*), whose practical use can be rather confusing.

This command is tricky because *Concatenate* combines all currently selected sounds **in the order which they appear in the in the objects window**. No matter the order in which you select the sounds, whether there are intervening non-selected sounds, the sounds' order of creation, or their filenames, *Concatenate* will simply look at the selected Sounds, find their ordering in the objects window, and stitch them together in that order.

To concatenate Sound A and Sound B into one file, with Sound A first:

1. Load both Sound A and Sound B into the objects window of Praat **such that Sound A is imported first**
    - If the sounds are already loaded such that B is first, use *Copy…* to make a new Sound B lower in the objects window
2. Ensure that the files contain the desired amount of empty space to either side of each word (as the entire files, silence and all, will be stitched together).
3. Select Sound A
4. Select Sound B
5. *Objects* → *Combine* → *Concatenate*
6. A new sound, named "Sound chain", will be created, containing Sound A directly followed by Sound B.

Script Tip!

This strict 'ordering within objects window' limitation applies to scripting as well. If, for instance, you've split the onset and coda away from the vowel, and want to re-combine them after vowel manipulation, you'll want to use 'Copy' to create a new version of the coda (which will then necessarily be the newest and last item in Objects), then Select Sound onset (which is earliest in the objects window), Plus Sound modified_vowel, Plus Sound coda (which you've just created), then Concatenate. This 'copy to move to bottom of objects window' hack is one of the ugliest, most just-hold-your-nose-and-code scripting tricks I regularly use.

\end{tabular}

Note that you also have the option to use *Objects* → *Combine* → *Concatenate recoverably*, which works identically to *Concatenate*, but also creates a 'TextGrid chain' annotation showing the extent of each file within the chain file, which can be useful to later split the files back up. *Objects* → *Combine* → *Concatenate with overlap…* performs the concatenation, specifying that the last N seconds of the first file should overlap the first N seconds of the second (which can be useful if, for instance, both words have 50ms of silence, but you want a 25ms inter-stimulus interval). But if you're attempting to combine the sounds, you should instead refer to…

8.12 Combining Sounds

There are two ways to combine two sounds using Praat. The first will sometimes work well, and works best when both sounds are to be added equally and have the exact same file length:

To combine (overlay) Sound A and Sound B:

1. Load both Sound A and Sound B into the objects window of Praat
2. Select Sound A and Sound B together
3. *Objects* → *Combine* → *Combine to Stereo*
    - This will create a new sound which has Sound A in one track, and Sound B in the other
4. Select the sound created by the last step
5. *Objects* → *Convert* → *Convert to Mono*
    - This will then collapse both tracks back into a single mono track

This will often work, and if it does, that's wonderful. If you end up with odd or undesirable results from this, move onto the next section and combine the two sounds using a formula.

8.13 Formula Modification: Waveform addition, subtraction and so much more

Praat is quite capable of doing sample-by-sample mathematical operations on both individual files and on pairs or groups of files. This is done using one of the most obtuse yet most powerful functions available in Praat, *Objects* → *Modify* → *Formula* function.

Formula modification is best visualized by thinking about digitized sound. Remember that when sound is digitized, the waveform itself isn't saved, but instead, the waveform is sampled (at the sampling rate), leaving you with a series of times and the amplitude of the wave at that moment.

So, to add two digitized waveforms, you simply move through the file sample-by-sample and compare each sample. During waveform addition, if, at the 31st sample (e.g.), the amplitude of Waveform A is at 28 and Waveform B is at -5, you add those two timepoints together (28 + (-5)), and in your resulting sound file, the 31st sample will have an amplitude of 23. Because this moves sample-by-sample, both sounds will need to have the same sampling rate, and need to be of the same length.

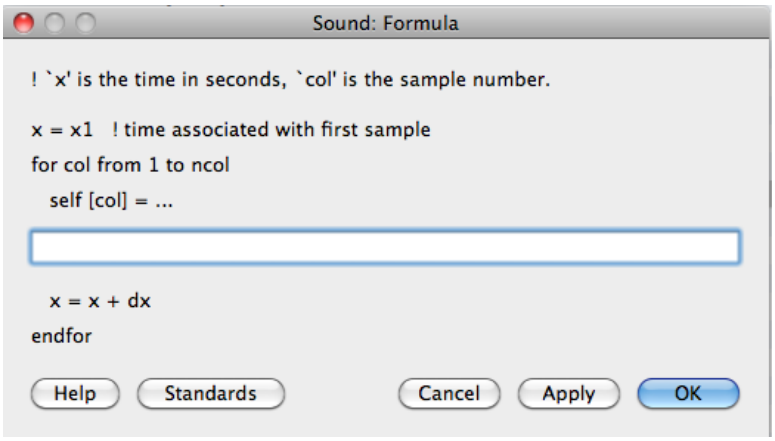

Figure 9: The Praat Formula Modification Window

Unfortunately, formula combination of sounds is among the least polished of Praat's features, but don't let that scare you off. To do any formula modification:

1. Load the sound(s) you'd like to modify into the objects window of Praat
2. Select the sound you'd like to modify
   - Remember, this will modify the existing sound, not create a modified copy, so make sure to make a copy of the sound to work on.
3. *Objects* → *Modify* → *Formula*

This will then pull up a very technical looking input box, pictured in Figure 9. To actually make the combination, you'll need to put into that box the formula which will be **applied to the amplitude value** of each individual sample.

To make this process a bit more clear, let's imagine that we've got a file called sounda and a file called soundb in our Objects window. You've gone through, made a copy of sounda to work on, selected that copy, and then *Objects* → *Modify* → *Formula*. Here are several formulas you could put into that window, and the effect that each of them would have:

- `0`
  - If you just put a "0" into the formula box, Praat would read this as "the amplitude of the sample in question in the selected sound equals 0" (self [col] = 0). Every sample would then be set to an amplitude of zero, and the resulting file would be completely silent.
- `4`
  - If you just put a "4" into the formula box, Praat would read this as "the amplitude of the sample in question in the selected sound equals 4" (self [col] = 4). Every sample would then be set to an amplitude of 4, and the resulting file would be constantly at four, *and would likely stress or damage your headphones or speakers if played loudly*.

- `self [col]`
  - Praat would read this as "the amplitude of the sample in question in the selected sound equals the amplitude of the sample in question in the selected sound" (self[col] = self [col]). As such, it would not modify the sound in any way, and would be a rather complete waste of your time.
    * *In formulas, you'll refer to (sound) [col] frequently, which just means "each individual sample in the sound". "self" is the sound selected when you opened the formula window}.
- `self [col]*2`
  - Praat would read this as "the amplitude of the sample in question in the selected sound equals two times the amplitude of the sample in question in the selected sound" (self[col] = self [col]*2). This formula would double the amplitude of every sample, making the sound file twice as loud.
- `Sound_soundb[col]`
  - Praat would read this as "the amplitude of the sample in question in the selected sound equals the amplitude of the sample in question in the sound file called "soundb". This formula would turn sounda into a sample-by-sample copy of soundb.
    * *Note that in formulas, other sounds in the object window need to be referred to as Sound_soundname, not just soundname.*
- `self [col] + Sound_soundb [col]`
  - Praat would read this as "the amplitude of the sample in question in the selected sound equals the amplitude of the sample in question in the selected sound **plus** the amplitude of the sample in question in the sound file called "soundb"". This formula would **add sounda and soundb using waveform addition**.
  - *When adding, subtracting, or multiplying sounds, you'll usually want to match the resulting sound's amplitude to that of the original sound afterwards (see section 8.10).*
- `self [col] - Sound_soundb [col]`
  - Praat would read this as "the amplitude of the sample in question in the selected sound equals the amplitude of the sample in question in the selected sound **minus** the amplitude of the sample in question in the sound file called "soundb"". This formula would **subtract soundb from sounda using waveform subtraction**.
- `self [col] + (2* Sound_soundb [col])`
  - This formula would **add twice the amplitude of soundb to sounda using waveform addition**. Such a formula would be useful for combining two sounds when you want the acoustical features of one sound to slightly overwhelm those of the other.

Also note that 'x' can be used to represent the time (in seconds) within the formula, allowing formulas like:

- `self [col] + ((x)*Sound_soundb [col])`
  - This formula will add sounda and soundb **such that soundb's amplitude is directly linked to the time in seconds**. This is very niche, but would allow you to mix sounds in such a way that one 'fades in' or 'fades out' over time.
- `self [col] + (Sound_soundb [col] * (x - xmin) / (xmax - xmin))`
  - This formula will directly link Sound B's amplitude to time, such that it is absent at the start of the sounds, and at amplitude 1 at the end.

Using the above examples as a template, you can create a formula to do nearly anything you'd like to your sound waveform, both through the user interface or using Praat scripting.

8.13.1 One-Bit Requantization

Some people use one-bit Requantization to modify sounds such that temporal information is preserved, but no frequency or amplitude information is (resulting in something which is a bit like a very low low pass filtered signal, but without permitting the remaining signal to vary in amplitude). The basic idea is that any portion of the sound with an amplitude greater than zero is set equal to zero, and any portion less than 0 is set equal to negative one. Thus, the signal has a bit depth of one (zero or one, on or off) and there are only two possible amplitude states. This can be done in Praat using the formula:

```
if self>0 then 0 else -1 fi
```

This can be used for stimulus preparation (although the resulting stimuli are acoustically horrifying), but also provides a nice demonstration of the usage of the usage of if/then statements in formulas. You'll want to scale amplitudes afterwards, ideally even before listening, but this is an effective way of producing an awful-sounding yet interesting token.

8.14 Synthesizing Sounds from scratch

Praat offers the ability to synthesize sounds using a Klatt Synthesizer, an articulatory synthesizer, the Vowel Editor and more. The use of these synthesizers is outside the scope of this class, but is well explained in the Praat Documentation.

8.15 Source-Filter Vowel Resynthesis

When creating stimuli for vowel perception experiments (as well as many other times), it can be useful to generate vowels of a controlled quality. Although Praat includes both articulatory and cascade (Klatt) synthesis, as well as the vowel editor, sometimes, it's important to maintain the vocal characteristics of the recorded speaker. In these situations, you'll want to use Source-Filter vowel resynthesis to alter the vowel's formant qualities without altering or replacing other

significant aspects of the signal. Source-filter resynthesis is most efficiently done by script, but can be done step-by-step by hand if needed.

To understand both the process and the workings of source-filter resynthesis (SFR), it's important to understand the source-filter theory of vowel production. This understanding of vowel production holds that a given vowel is composed of two element: the source (the voicing coming from the larynx), and the filter (the articulations and anatomy of the vocal tract above the larynx). When the source signal passes through the vocal tract, the resonances in the mouth heighten some frequencies and damp others. In this way, a relatively unremarkable voicing spectrum, passed through a vocal tract with an /i/ tongue shape, ends up with formants at 250 Hz, 2500 Hz, and 3000 Hz (your resonances may vary).

Source filter resynthesis takes advantage of this idea to modify vowel qualities by taking the following steps, in the abstract:

1. Take a recorded vowel and locate the overall peaks and valleys in the spectrum (the formants) by using an LPC (linear predictive coding) algorithm
    - These peaks and valleys, at least theoretically, should represent the resonances in the mouth caused by a given tongue shape
2. Use this information to reconstruct the voicing signal (the source) without those peaks and valleys
    - This is accomplished by inverse-filtering the signal with the LPC, raising the parts of the spectrum which the LPC says are low, and lowering the parts which the LPC says are high. The end result, ideally, will be the source signal as if the person had no vocal tract at all.
3. Alter the LPC to change the positions or bandwidths of the formants to your desired characteristics
    - By doing this, you modify the "filter", effectively changing the tongue-shape and associated resonances used to initially produce the vowel
4. Filter the reconstructed source (created in Step 2) using the altered LPC (from Step 3)

Performing **Source-Filter Resynthesis of a vowel** in Praat:

1. Isolate a vowel in a single sound file (we'll call it "vowela")
    - Any vowel will do, but you'll get better results if you downsample the vowel first.
2. Select that vowel, then *Objects* → *Formants & LPC* → *To LPC …*
    - Choose whatever algorithm you prefer. Burg is my personal favorite.
3. Select both the vowel and the LPC object that's been created (*Sound vowela* and *LPC vowela*), then *Objects* → *Filter (inverse)*
    - This will create the "neutral" source voicing from that vowel
4. Select the LPC, then *Objects* → *To Formant*

- Praat won't let you edit an LPC object, but you can easily edit a formant object

5. To change formant frequencies, select the Formant object, then *Objects → Modify → Formula (frequencies)*…

- You'll edit this by formula (see Section 8.13), but it will always have the form `if row = [formant number] then self [modification] else self fi`. So, `if row = 1 then self + 100 else self fi` would raise the first formant by 100 Hz.
  - *Changes less than 20 Hz are tough to spot and measure, and may not be particularly precise.*
- To change formant bandwidths, select the Formant object and the "source" generated in step 3 and *Objects → Filter*
  - It's *very* easy to over-thin bandwidths, especially when doing this by script on tokens with variable bandwidths. When this happens, you'll see no formant, and just get a sort of chirpy-sounding prominence. Be cautious that you're not accidentally calling for a 5 Hz wide formant!

This will output your new vowel, ideally, with the new formants. That said, this is a very finicky process with lots of room for error, and it will require some work to get working cleanly. This is also a situation where using a good Praat script can speed things up significantly, and allow the rapid repetition of small changes to improve output.

Given some work, though, and some time, this can be an excellent way to modify vowel qualities. For examples of this in use for stimulus preparation, contact the author.

Script Tip!

Resynthesis works best with downsampled sounds, and introduces some artifacts. You will almost certainly want to modify only the bottom 3500 Hz of the vowel, and then re-combine it with the unmodified higher frequencies, so that you change what you need to, but keep the rest pristine.

\end{tabular}

8.15.1 Tips for Source-Filter Vowel Resynthesis

1. **The best SFVR is none at all!** If you can find or make tokens in other ways, they're often better, and recording a variety of tokens and cherry picking ones with the formant values expected may be more productive for many designs. This is, in many ways, a technique of last resort.
2. **Use many small steps.** You'll generally get better results from 5 50Hz steps than one 250Hz step. It's counterintuitive as you'd think processing errors would stack, but for whatever reason, it's been the case that you want to move slowly, at least last time I spent much time with this.

3. **Downsampling is key.** Use the instructions above for downsampling and your life will be better.

4. **If the F_0 track is bad, the token is too**. To deconvolve source and filter, you'll need a strong F_0 track. Strange results might be because Praat's struggling to isolate F_0, or worse still, struggling to accurately track it. You might be able to adjust F_0 settings some to get closer, but you'll want nice modal tokens with clean-ish pitch tracks as input.

9 Exporting images for use and publication

To be completely honest, the fastest (and usually sufficient) means of getting images from Praat for documents or presentation is arranging the windows to display what you'd like, and then taking a screenshot, which can later be cropped.

However, with a bit more work, Praat can be used to create and annotate publication quality graphs. The **Praat Picture window** (Figure 10) is used to create and display images.

Using the Praat picture window can be thought of as a five step process: Create an object, choose your size, draw your object into the picture window, garnish, and export. To give a brief example, let's say you'd like to export a spectrogram of sounda:

1. Select sounda, *Objects* → *Spectrum* → *To Spectrogram…*
    - Set the Spectrogram settings using the same settings you would in the Editor window (see Section 7.2)
2. Click and drag in the picture window to set the size you'd like.
    - Realize that these images scale nicely, so the default usually suffices. Also, you can create more than one graphic per picture window, just change the selection area.
3. Select the newly created Spectrogram object, then *Objects* → *Draw* → *Paint…*
    - Set these parameters as you'd like.
4. Use the *Picture* → *Margins* menu to add labels, text, and other garnishes
5. Use *Picture* → *File* → *Save as PDF file…* to export your picture

- Be warned, the resulting PDFs will have very large file sizes as they capture a great deal of detail. As such, they can be expanded and shrunken quite gracefully, but will make for some massive slideshow files.

Note that there's no need to include a spectrogram here, you can just as readily create a plot featuring only a pitch contour or formant trace. But often, a spectrogram is useful for delimiting . There is obviously much more complexity possible, but repeating (and adapting) the steps will yield wonderful graphs for any publication. This entire process is quite easily scriptable, as well.

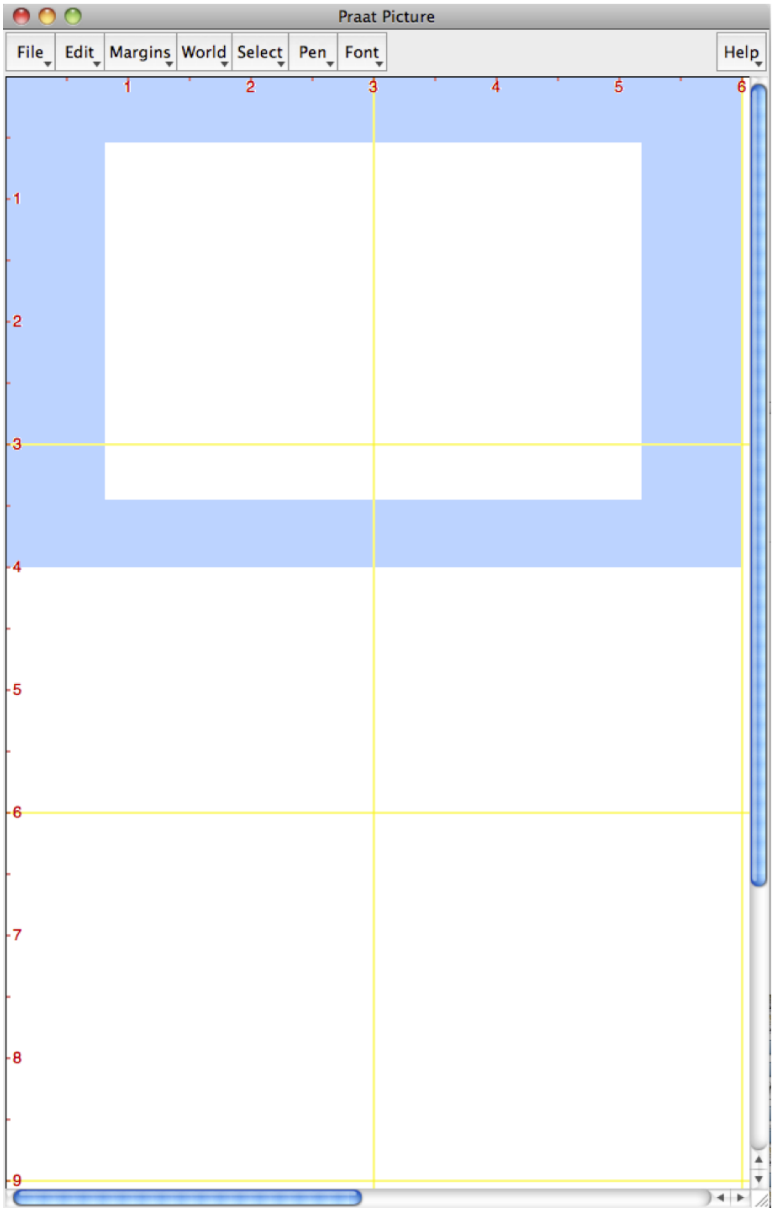

Figure 10: The Praat Picture Window

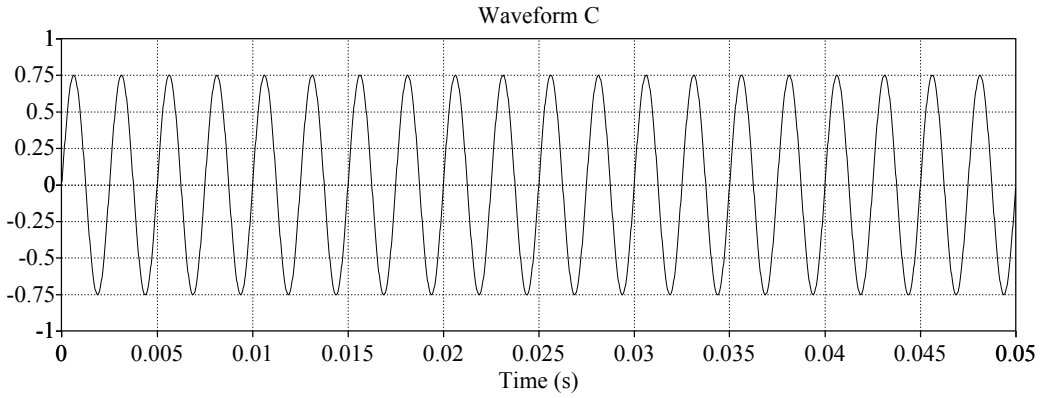

Figure 11: An example of an exported, garnished waveform plot of a 400 Hz pure tone

For additional information about using Praat pictures, consult Jennifer Smith's wonderful *Printing and copying/pasting Praat Images* handout, available at `http://www.unc.edu/~jlsmith/ling120/pdf/5images.pdf`.

Bonus!

By creating sounds from formula (see Section 8.1) and combining them together using waveform addition and subtraction (see Section 8.13), it's a breeze to create the sorts of lightweight-yet-accurate waveform graphs needed when teaching and testing students on the fundamentals of acoustics and waveform addition.

\end{tabular}

9.1 Creating Complex Displays

Using the Praat picture window as a canvas, you can add more detail to a graph than is available from a single command by overlaying plots and combining multiple plots into a larger layout.

9.1.1 Overlaying Plots

You can use the Praat picture window to generate plots with multiple data types on them at once. For instance, if you wanted to overlay a pitch trace onto a spectrogram, you would simply draw a spectrogram (as above), and then, without changing your selection in the window, create a pitch object and draw it, with the same "Time" boundaries as for the spectrogram. You can also overlay a TextGrid's content to label your data.

Although it may not always make sense to do so, you use a series of draw commands to overlay **any** graphics. To do this, complete the same steps above and then repeat the "draw" process for each element, changing the color and width of lines using *Picture* → *Pen* before each draw to help multiple plots stay distinguishable.

Doing this, you can represent as many different types of data on a single plot as you'd like, up to and well beyond the possibility of any readability.

9.1.2 Multiple Plots in the Picture Window

Similarly, if you wanted to draw a waveform above a spectrogram in the picture window (similar to the editor display), you would:

1. Select the portion of the picture window which will have your waveform
2. Select a sound, then *Objects* → *Draw* → *Draw…*
    - Edit the parameters to "zoom" to the portion you need
3. Use the *Picture* → *Margins* menu to add labels, text, and other garnishes
4. Select the still-blank portion of the picture window which you want to contain the spectrogram
5. Select the sound, *Objects* → *Spectrum* → *To Spectrogram…*
    - Set the Spectrogram settings using the same settings you would in the Editor window (see Section 7.2)
6. Select the newly created Spectrogram object, then *Objects* → *Draw* → *Paint…*
    - Set these parameters as you'd like.
7. Use the *Picture* → *Margins* menu to add labels, text, and other garnishes
8. Use the Picture window selection to surround *everything* in the window for export
9. Use *Picture* → *File* → *Save as PDF file…* to export your picture

By doing this, you can create, entirely within Praat, a whole matrix of plots which show all the various facets of a given sound, all without resorting to external photo editors.

10 Annotating Sound Files (Praat TextGrids)

Praat can't reliably tell where one word starts and where another ends. Nor can it find the specific segment you're looking for, nor identify the vowel in the word. As such, you'll often need to segment sound files with that information when using any sort of automated measurements. In Praat, this is done by creating **TextGrid** annotations in a TextGrid file, which is saved separately from the sound itself.

TextGrid annotations are composed of different tiers which mark either intervals or specific points within the sound file. Interval tiers are designed to mark elements of a file with a distinct span, like a vowel, word, or segment. Point tiers mark single points, like release bursts, turn changes, or glottal openings. Both intervals and points can (and should) be labeled. The names and relative position of these tiers are specified when the TextGrid is created.

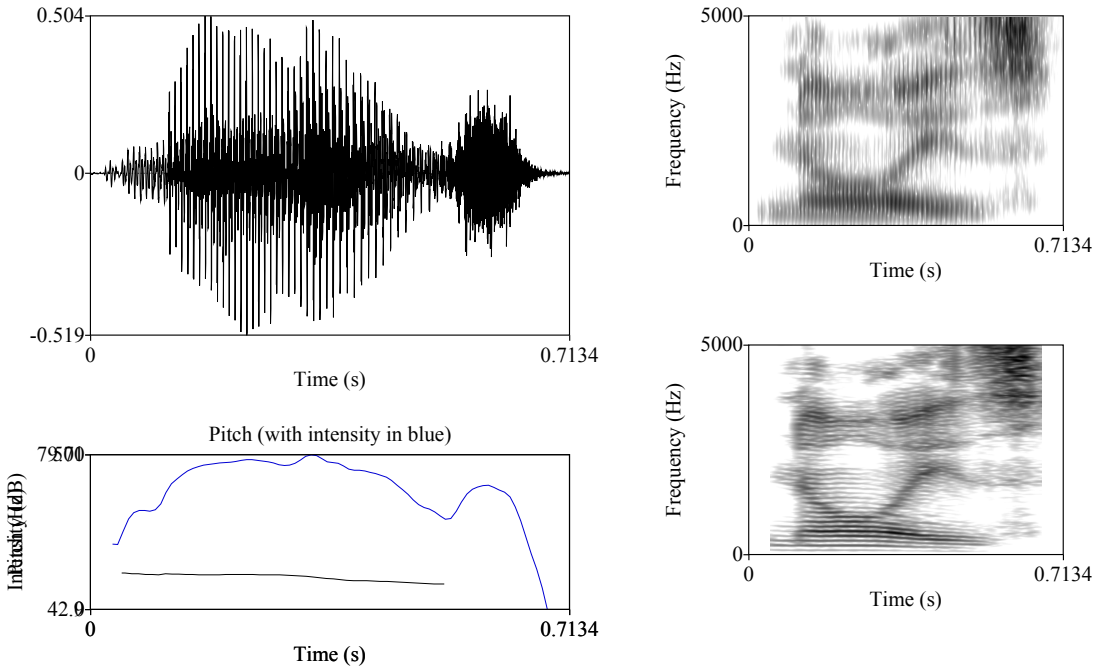

Figure 12: You can now combine, overlay, and juxtapose plots to excess, as in the above single Praat picture

It's worth noting that Praat counts EVERY interval in the file, not just human-created or labeled ones, so if you mark and label a single vowel interval in a file, Praat will consider the file to have three intervals: the section before the start of the vowel, the marked vowel, and the section after the vowel. The marked vowel, in this situation, would be considered interval number 2.

To create a TextGrid for a sound file…

1. Open and then select the sound that you want to TextGrid
2. *Objects* → *Annotate* → *To TextGrid…*
3. Name your tiers, specifying on the following line which tiers, if any, are point tiers. Tiers will be ordered in the order named.
4. Select both the sound you'd like to annotate and the associated TextGrid, and click *Objects* → *View & Edit*

Once your TextGrid is created, you'll be presented with the **TextGrid Editor Window**[11] (Figure 13). To annotate the file…

5. Click on the tier you'd like to make an interval on
6. Using your mouse, select the part of the word you'd like the interval to contain
7. Hit the key
8. Click on the interval you've just created and name it
    - You can use IPA characters in TextGrid labels (provided you have the proper Unicode IPA fonts installed), but when exporting the labels by script or elsewhere, all programs used for analysis must be Unicode aware. Most modern programs are, but many command-line programs (such as SPSS or Python 2.X) are not natively or straightforwardly happy in Unicode, so if you're planning to use any non-Unicode aware programs, you're best served using SAMPA or some other means of transliterating IPA characters into ASCII text.
9. Repeat for other intervals on the same tier, as well as any other tiers
    - Points are created by clicking on a point tier, placing the cursor where you'd like the point, and hitting .

Once your file has been TextGridded as above (looking something like Figure 14), you'll want to save the TextGrid file (either from the Objects window (*Objects* → *Save* → *Save as Text File…*) or within the TextGrid editor (*TextGrid Editor* → *File* → *Save TextGrid as Text File…*).

TextGridded files can then be read in by Praat scripts which measure only certain parts of the word, can be split and labeled according to a given tier by script (see the `file_segmenter.praat` script), or can simply be examined with the benefit of the labels. Once all your files have been

[11] Although the window which is opened when a sound and TextGrid are both selected and *View & Edit*ed looks a lot like the Editor window, Praat considers it to be a very different sort of window. Any scripts added to the Editor window (see Section 12) will have to be added separately to the TextGrid editor window, and some menu options present in the Editor window are not present in the TextGrid editor window.

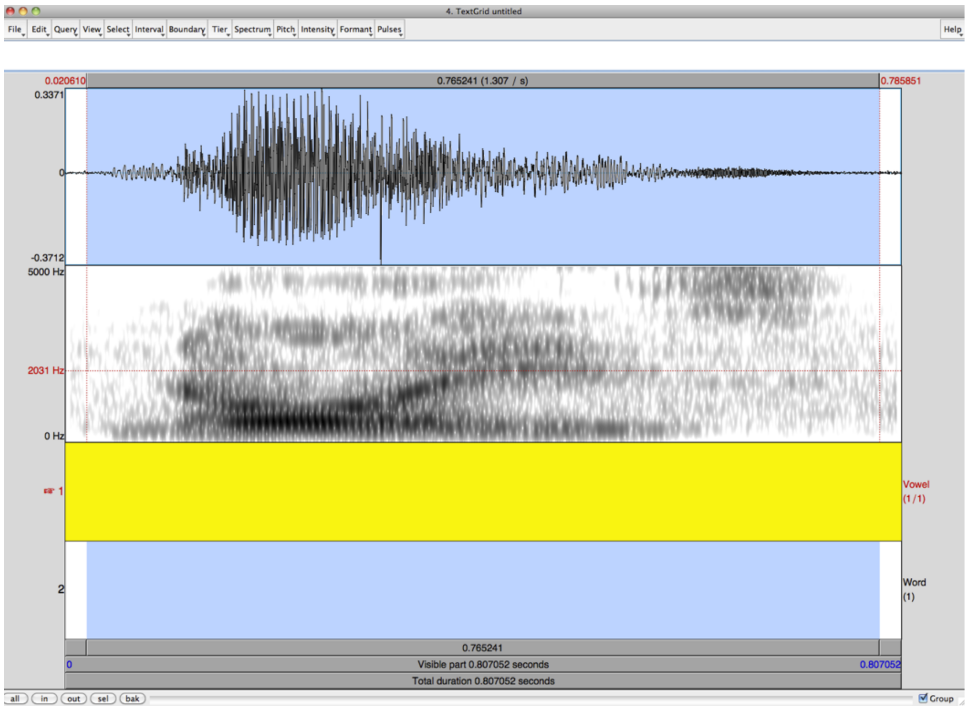

Figure 13: The Praat TextGrid Editor Window

TextGridded, you'll be in a much better position to start automatically measuring data and manipulating your sound files.

11 Using Log Files

So far, all discussion of measurement has assumed you would be taking individual hand measurements and writing them down on your own. These last two sections describe two ways to more efficiently capture data from your measurements.

Praat has the ability to easily capture the measurements you take as you move through your data using log files, which are then easily exported into Excel (or the statistics program of your choosing). This is useful when doing a series of repetitive measures which still require human intervention, but in many cases, the use of Praat scripting can speed this process up further and in some cases, eliminate the need for human intervention altogether.

To create and use a log file:[12]

1. Open a sound in the Editor window
2. *Editor* → *Query* → *Log Settings…*
3. Choose a location and file name for *Log file 1* or *Log file 2*

[12]Large parts of this section are adapted from a handout by Dr. Rebecca Scarborough for a previous LSA institute. All credit belongs to her.

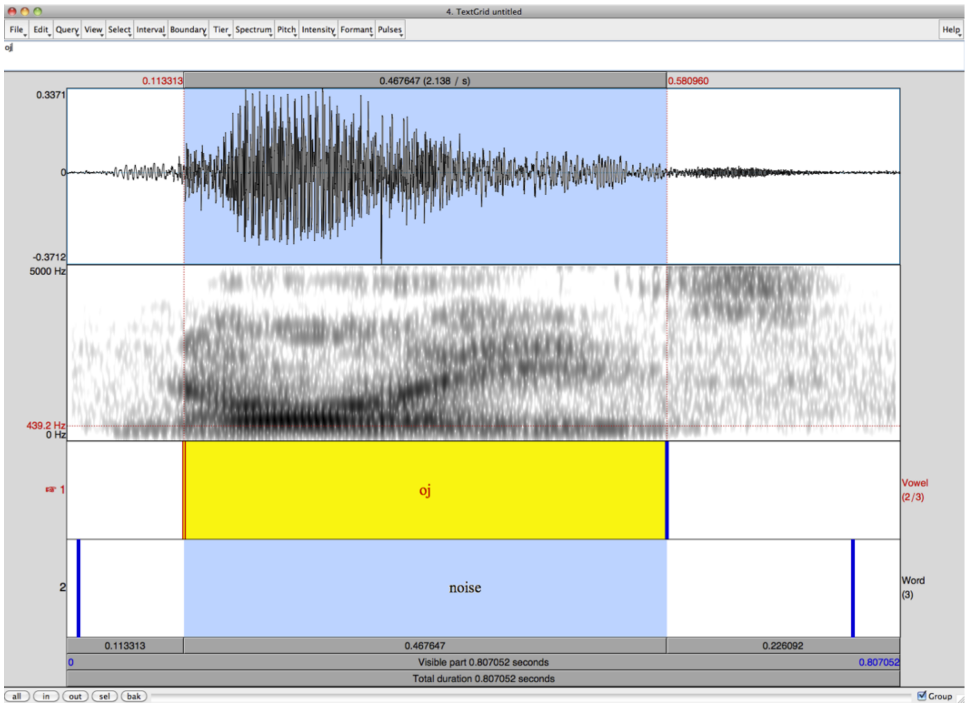

Figure 14: A completed TextGrid annotation showing “Vowel” and “Word” tiers

- On a Windows machine, this will look like C:\Documents and Settings\All Users\Desktop\Log.txt
- On a Mac, this will look like /Users/yourname/Desktop/log.txt

4. Fill in *Log 1 Format* and/or *Log 2 Format* according to what you’d like to measure:
   - **For duration** (include the single quotes):

     ```
     ‘t1:4’ ‘tab$’ ‘t2:4’ ‘tab$’ ‘dur:6’
     ```

     – This will give you the start time (t1) and end time (t2) of a selection and its duration (dur). (The numbers after the colons specify the number of decimal places.)
   - **For formants** (include the single quotes):

     ```
     ‘t1:4’ ‘tab$’ ‘f1:0’ ‘tab$’ ‘f2:0’’tab$’ ‘f3:0’
     ```

     – This will give you the start time (t1) and first three formant frequencies (F1, F2, F3) at the cursor point (like a formant listing for 3 formants).
   - **For pitch** (include the single quotes):

     ```
     ‘time:4’ ‘tab$’ ‘f0:2’
     ```

     – This will give you the time (time) and fundamental frequency (F_0) at the cursor point (like a pitch listing).

5. Close the Log Settings window
6. You can now go back to the editor window and record duration, formant frequencies, or F_0 by simply selecting the relevant point or selection in the waveform (or spectrogram) and hitting F12 (for log 1) or Shift-F12 (for log 2).
7. By default, each measurement will display in an Info window as well as write to the file you set up. If you want, you can log to the log file or the Info window only by changing the settings for *Write log 1 to:* or *Write log 2 to:* in the *Editor* → *Query* → *Log Settings…* window.
8. You can modify logs to include any compatible bits of information using the following variables: * `time` : Time at cursor or at midpoint of the current selection * `t1` and `t2` : Time at beginning and end of selection * `dur` : Duration of selection * `f0` : F_0 at cursor or average F_0 of selection * `f1-f5` : Formant at cursor or average for selection * `b1-b5` : Formant bandwidth at cursor or average for selection * `intensity` : Intensity at cursor or average for selection * `tab$` : tab

12 Scripting in Praat

12.1 What is Praat scripting?

Much like a movie script tells a set of actors exactly what to say and what actions to take, a Praat script is a text file which tells Praat exactly what to do, and walks the program through a series of steps. Anything that you can do through the GUI of the program (GUI is "graphical user interface", the menus and buttons you use to interact with Praat), you can command Praat to do in a script.

Praat scripting is a wonderful tool, then, for situations where you find yourself repetitively doing the same tasks over and over again, and with increased sophistication, you can have Praat make simple decisions based on its measurements and changes. Simply put, Praat scripting is a way to use the computer as a co-pilot, handling the boring, repetitive tasks independently and allowing you to do your job more efficiently.

Praat scripts can be as simple as a single line which tells Praat to adjust the Spectrogram settings to display a narrowband spectrogram, or as complicated as 500+ lines of code which allow Praat to go through a folder of files, measuring 33 acoustical correlates of nasality for each file and presenting any suspect measurements for human confirmation.

Here's a very basic Praat script, which shows a series of commands to Praat, with commented lines (prefaced with #) to explain what it's doing:

```
# This is a comment, Praat ignores lines that start with #

select Sound untitled
# Plays the sound
Play
# Gets the duration
```

```
Get total duration
# Gets the amplitude
Get intensity (dB)
# Renames it
Rename... My_awesome_sound
# Prints a message into the Praat information window
print "Script Finished"
```

Note, though, that Praat scripting isn't the same as writing a new computer program. Praat scripting uses the same commands as you use in the GUI, and only works within Praat, whereas coding (say, in C) is much more opaque, but programs in C can be compiled to run on many different devices and don't need specific programs.

That said, there are a few things that Praat scripting cannot do:

- Praat scripts can't label your data, mark linguistic units, or recognize speech
  - Praat doesn't know what's being said, so you need to TextGrid annotate your files such that Praat knows where to look for the vowel(s) or consonant(s) or syllable(s) that you're interested in.
- Praat scripts only run in Praat (although the *system* command can control the OS and some other programs)
  - Praat scripting only affects Praat, so you can't incorporate a step which, say, opens another program to run it.
- Praat scripts can't generate measurements as consistent as hand measurements
  - Individual words can be odd. Humans can make small adjustments (taking the pitch measurement just to the right of that bit of creak), but Praat doesn't know enough to do that.
- Praat scripts can't sanity-check their data
  - You know fully well that F1 for /i/ is unlikely to be at 8,000 Hz. Praat doesn't. You can build in safeguards, but Praat will allow all sorts of craziness if left to its own devices.
- Praat scripts can't do anything that you couldn't eventually do through the user interface of Praat.
  - Praat scripting depends on Praat's built-in commands, so you can't do something that the program was not designed to do. That said, you can string many commands together to create new commands and processes, and you can certainly make things easier for yourself.

To do anything vaguely scripty in Praat, you'll use the Praat menu (in the menubar on a Mac, at the top of the objects window on a PC), and choose *New Praat Script* to open the scripting editor.

12.1.1 Praat Scripting Alternatives for common tasks

Praat script is a wonderful thing, and combines the speed of scripting with the turned accuracy of measurement that the Praat GUI can get you. However, for some batch tasks, you should consider alternatives to Praat scripting which may be more powerful, easier, or more practical.

Based on the author's own work and experience, some tasks are better conducted in other (free) programs. So, if you're planning to…

- Batch convert files to and from multiple file formats
  - iTunes (Cost-free, but not free-as-in-freedom) can accomplish many conversion tasks, using *iTunes* → *Preferencces* → *General Preferences* → *Import Settings* and then using *File* → *Convert.*
    * Remember, friends don't let friends save phonetic data in lossy formats (e.g. .mp3, .AAC, .wma)!
  - SoX (`http://sox.sourceforge.net`) is a free command-line utility which can convert and modify a variety of file formats, including many not readable by Praat, particularly once combined with ffmpeg.
- Run statistics within the script itself
  - Use PraatR (`http://www.aaronalbin.com/praatr/index.htm`), an implementation of Praat scripting commands within the R Statistics environment, that allows you to write R scripts which call commands within Praat. It's a wonderful thing, and for many uses, can replace Praat scripting altogether.
    * With this kind of power, you can run real-time regressions on formant data, and sanity check measures that way, as the author of PraatR does. It's brilliant.
- Bulk find-and-replace or delete within Textgrid files (e.g. "Unlabel all intervals with the label "V"" or "Change every instance of "spot" to "spat"")
  - Use a plaintext editor, and open the Textgrid files directly, editing using find-and-replace. Some bulk editors like Textmate allow you to modify all files in a folder in the same way. Textgrids are just text files with lots of numbers, and can be edited easily.
- Create vowel charts based on vowel formant data
  - Use the `vowels` package in the R Statistics Environment. It makes pretty vowel charts easily (once the data is properly formatted), and takes care of all the little things (reversing axes, etc). It also incorporates various optional algorithmic vowel normalization methods, if you choose to implement them.
    * Friends don't let friends normalize vowels algorithmically without a firm understanding of what such methods can and cannot do. With great power comes great responsibility.
- Manipulate files on your computer

  - Praat *can* create, delete, and modify files on your drive. But by and large, your best option is to use some sort of machine-native scripting language (like shell script) or Python to conduct these tasks. Praat is more than capable of working with files, but you'll experience pain reading in files with multiple "." characters per name, spaces in names, etc if you do this in Praat.

- Run a perception or psychology experiment
  - Praat has a workable system for running experiments that can be adapted to many workflows. However, it's simply not as robust as other options like PsychoPy (`http://www.psychopy.org`) or OpenSesame (`http://osdoc.cogsci.nl`), especially when working with reaction times or external hardware (button boxes, voice-keys, etc). Learn to use one of these tools, and you'll have more flexibility and power when the time comes (and you'll be able to avoid expensive, non-free experiment packages like ePrime or SuperLab)

- Forced Alignment and automatic speech labeling/text-grid generation
  - Forced Alignment programs like FAVE-align (`http://fave.ling.upenn.edu/usingFAAValign.html`), Penn Phonetics Lab Forced Aligner (P2FA, `http://www.ling.upenn.edu/phonetics/p2fa/`), or Easy-Align (`http://latlcui.unige.ch/phonetique/easyalign.php`) are all reasonable choices for alignment.
    * Always remember that Forced Align speeds up manual annotation (by providing reasonable boundaries to correct). It does *not* replace it!
    * Don't forget that you can edit the resulting Praat Textgrids in your favorite text editor, and automatically un-label every word in your carrier sentence, or remove all non-target phonemes. This can save huge amounts of time.
  - As of Praat 6.0.2x, Paul Boersma appears to be working on including a forced-align component. Keep an eye on that space!

12.1.2 Praat's scripting tutorials

Perhaps the most useful part of Praat's documentation is the Scripting tutorial, accessible through *Objects* → *Help* → *Praat Intro* → *Scripting* or through the Script Editor window. Because these tutorials are so excellent at explaining the peculiarities and structures of Praat's scripting language, this document will skip many of the topics discussed there, focusing instead on overall usage.

12.1.3 Praat scripts vs. Editor scripts

One small but worthwhile note is that Praat actually draws a division between scripts and "editor scripts". They use the same syntax, the same commands, and are stored in the same way, but some scripts are meant to run within the Editor window (because they use the functions in that window), and some are meant to run from the Objects window.

Simply put, if there's a script whose function is only useful once you've already opened the sound

you care about in the editor window, it's going to be run as an Editor script, but if a script needs access to more than one sound, or to the functions in the Objects window, it'll run from the main windows. There is some flexibility here, though, as you can always use the `Edit` and `Editor` commands to work within the editor in a Praat script, and using *endeditor* can bring you back to the Objects window from an editor script.

The primary way in which this distinction matters is that you'll open Editor scripts using the *Editor → File → Open Editor Script…* option, whereas you'll open regular Praat scripts with *Praat Menu → Open Praat Script…*.

Danger!

Sometimes you'll download a script from the internet or from a friend and it just won't run, or it'll fail in odd ways. In those situations, try running it as an Editor script (or as a Praat script, if you're trying it in the Editor). This is a simple step that, if it works, can save you hours of frustration and troubleshooting, as there's often no immediate indication how a given chunk of code was meant to be run.

\end{tabular}

12.2 Working with Scripts

Because a Praat script is literally just a text file with a series of commands to Praat, they can be found all over the internet, and can be saved as a plaintext file (it's best to give it the .praat suffix, if nothing else, for your own reference). Praat scripts can be run in two distinct ways.

12.2.1 Opening and running a Praat script

To open a Praat script in Praat and then run it:

1. *Praat Menu → Open Praat Script…*
    - This will open the script in a Script Editor window, where it can be viewed or edited.

To open an Editor script in Praat and then run it:

1. Open the sound you're interested in in the Editor window.
2. *Editor → File → Open Editor Script…*
    - This will open the script in a Script Editor window, where it can be viewed or edited.

To run a script (either Editor or Praat scripts): 1. *Script Editor → Run → Run* (or Cmd + r, Ctrl + r on Windows)

Note that if you just want to run a small part of the script (useful when cannibalizing scripts for certain functions), you can select the part in question and use *Script Editor → Run → Run Selection*.

12.2.2 Making (and removing) Menu Shortcuts for scripts

Sometimes, you'll want frequent access to a Script without having to find it, open it, then click "Run". To do this, you can add the script to a menu. There are two options on how to do this[13]:

First, you can add the script as a button on the right-hand side of the Object window (called a dynamic menu). This will only work for Non-Editor scripts. To do this:

1. Open the Script, such that it's available in a Script Editor window.
2. *Script Editor* → *File* → *Add to dynamic menu…*
3. Fill in the settings as below in the *Add to dynamic menu"* dialog box
    - *Class* and *Number* refer to which object types and how many must be selected for the new script button to appear. (For example, if you had a script that required a sound and a text grid, you would put *Class 1*: Sound, *Number 1*: 1, *Class 2*: TextGrid, *Number 2*: 1. You would leave *Class 3* blank.)
    - *Command* specifies the text that you want to appear on the button.
    - *After command* and *Depth* pertain to the placement of the button. If you leave them blank, the button will appear at the bottom of the dynamic menu.
    - *Script file* should be filled in automatically, if you are working from a saved Praat script.

The other option is to add your script to a pull-down menu in the Object window. (You can follow the same directions to add an Editor script to an Editor window menu.) To do this:

1. Open the Script, such that it's available in a Script Editor window.
2. *Script Editor* → *File* → *Add to fixed menu…*
3. Fill in the settings as below in the *"Add to fixed menu"* dialog box
    - *Window* specifies whether the command will appear in an Object window menu or a Picture window menu.
    - *Menu* specifies which pull-down menu the command will appear in (from the top of the window).
    - *Command* specifies the text that will appear in the menu.
    - *After command* and *Depth* pertain to the location of the new command in the menu. Leaving the defaults will place the new command at the bottom of the menu.
    - *Script file* should be filled in automatically, if you are working from a saved Praat script.

To remove a script from a menu, take the below steps[14]

1. *Praat Menu* → *Preferences* → *Buttons…*

[13] Thanks, once again, to Dr. Rebecca Scarborough, from whose writing parts of this section are adapted.
[14] Thanks to Kathleen Currie Hall for pointing this out during the LSA Praat workshop!

2. Choose the proper category ("Objects" for scripts in the objects window menus, "Editors" for scripts in the Editors window menus, etc)
3. Find the script you'd like to remove
4. Click on *ADDED* such that it turns to *REMOVED*
5. Close that window, and if needed, restart Praat

Editor scripts can also be associated with one of the Script Logs (Log 3 or 4) in the "Query" menu of the Editor window. To do this:

1. Open a sound in the Editor window
2. *Editor* → *Query* → *Log Settings…*
3. Type the path to the script in the Log Script 3 (or 4) box.

Once this is completed, you can use a keyboard shortcut to run your script.

Bonus!

Adding scripts to the Log Script slots is particularly useful for scripts which you might run extraordinarily often, as you can then, with some additional software, set a button on a Multi-button mouse to one of your log script keystrokes. For instance, if you have a script that will take the current selection, create two TextGrid boundaries, label it "vowel", then resize the view to the next interval on another tier, you could bind it to a log script, then just make a selection, click a mouse button, rinse and repeat, saving countless hours.

\end{tabular}

12.2.3 "The Praat script I downloaded won't run!''

Unfortunately, there are a number of issues which mean that even for an experienced user, downloading a script from the internet doesn't always work immediately. Although it's hard to say exactly why your script didn't run, here are some of the most common issues people run into which cause scripts to fail:

1. **You need to download the latest version of Praat**
   - This is the #1 reason for script failure, right here. There are new improvements to the scripting language every few months. If the script was written by somebody running version 6 and you're running 5.3, the script may include commands which the old version doesn't know how to run. So, go download the latest version from `http://praat.org`.
   - Praat is wonderful in that old scripts are universally supported. Newer versions of Praat always run older scripts, so don't worry about "breaking" other scripts by upgrading.
2. **The file paths are formatted incorrectly**

- If you use a Mac and the script was written on a Windows machine (or vice versa), you'll need to change the file paths. See Section 12.3.2 for more details.
- Also check to make sure they've not hard-coded paths (say, reading the data from the specific folder "/Users/will/schwalapalooza/inputdata"), and change those paths to point to your data where needed.

3. **You're using a different sound file type than the script wants**
    - Make sure the script isn't wanting .aiff files when you're using .wav (or vice versa). This is easy to change by changing the file-names hard-coded in.
    - Because there's no practical difference between the two anymore, I recommend using .wav for your files, as it's more common and you'll face this issue less.
    - Using .mp3 files breaks most scripts, and friends don't let friends save phonetic data in lossy formats (e.g. .mp3, AAC, .wmv).
4. **It's an editor script**
    - You need to make sure you know whether your script is an editor script or not, and run it appropriately. See 12.1.3 for more information.
5. **Your data isn't in the format it wants**
    - More complex scripts need very particular formats. Some require textgrid annotated individual words, split apart and in a folder. Some require a single long wav file, with one single textgrid labeling all of the words. Or maybe the script needs the files to named in a certain way. Or maybe you need to load all of the sounds into the objects window.
    - Good scripts should have a read-me at the top of the file (or nearby) including this information. For bad scripts, you can usually figure this out by looking at the code. But complex scripts should have good documentation.
6. **The Textgrids aren't what it expects**
    - Often, scripts use Textgrid files to let you specify where to measure, but there's no particular standard as to how those are made. Maybe the script is looking at Tier 1 for the vowel, but your Tier 1 has words labeled, and Tier 2 is the vowel. Or maybe it wants point tiers, but you've used intervals. Or maybe it needs a specific label ("V" for vowels) to intervals to measure.
    - Script makers will often allow you to specify tiers or labels for the relevant measures. But failing that, you can always go through and hard-code your tier number, changing the tier number in any commands that read from the textgrid.
    - Remember too that Praat TextGrid files are just text files. If the script wants every measured vowel to be labeled "V", and you want to measure everything labeled "Vwla", "Vwlu", or "Vwli", you can always open a copy of the grid file in a text editor and find-and-replace the labels to relabel them.

7. **You're doomed**

    - Sometimes, you'll never get a script to run. Maybe it never ran, or there's a bug that the author didn't catch. Or maybe the author was doing something very unusual, or was relying on a second script to feed it data. But regardless, it may not be your fault. Try and find another script which does the same thing, or write your own.
    - Despite working with Praat for 10 years, just last week, I downloaded a script for automatically generating .TextGrid files for a folder full of .wav files that I just couldn't make run. After around 20 minutes of battling, I just gave up, and wrote my own.

Finally, a note: When you've found a script online in somebody's repository, it's very tempting to contact the author of the script and ask for help in running it. If it's a short script you can find elsewhere, don't bother, but this might be an option in cases where the script is designed just for what you're doing and would save you years of work. But **contacting the author should be your last resort**, as they're likely just as busy as you are, if not more so, and many don't have time to help people debug work that they've posted online.

So, before you even consider contacting the author, I'd recommend you put 5-10 hours into trying to understand the code and debug it yourself. You'll learn a great deal from this process, and there's a pretty good chance you'll fix the problem yourself. You might also bug somebody else in your life who works with Praat, if such a person exists, because troubleshooting is always easier when you've got the machine and code in front of you and can just "try something."

But if it's still not working, despite putting in a lot of work to fix it, and there's no other script that does what you need, send the author an email. As somebody who maintains a large repository of scripts and gets a *lot* of emails from people, here are a few things you can do to give yourself the best chance of a real response:

- Download the latest version of Praat first.
    - This is my default response to people contacting me, and around 70% of the time, when people upgrade from the antique version of Praat they've had on their machine since grad school, the errors abruptly disappear.
- Include lots of information ("I'm running on Windows 10, Praat Version 6.0.23, using a folder full of .wav files and textgrids, and using the version of the script found at [URL]. I've attached a screenshot of the error message I'm getting when I run the script, and here's a Dropbox link to the folder full of data I'm trying to run on.")
- Check all of the above issues, and mention what you've already tried in your initial email. I'm more likely to respond if I know that you're not just facing Windows filename glitches or an outdated version of Praat.
- Mention what you're working on! It's silly, but if you're doing interesting things, or using the script for your dissertation, you become more human, and the time spent responding seems more reasonable.

But most importantly, be kind. I know that I feel terrible ignoring emails from people asking help,

but some days, I'm just barely able to finish my own work, let alone help with others. So, if you don't get a response within a few weeks, try again, but don't take it personally if it still doesn't work.

And, if all else fails, and you just can't get somebody else's script running, maybe it's time to think about…

12.3 Creating a new script

At its core, a Praat script is just a plaintext file with commands meant to be read by Praat. As such, creating a script is as easy as saving a text file, then editing the contents. The easiest way to do this is using Praat's built-in Script Editor.

To create a new Praat script:

1. *Praat Menu* → *New Praat Script…*
2. *Script Editor* → *File* → *Save*
    - Save it wherever you'd like on your machine, and use .praat as an extension (not because it's required, but because it can't hurt, will help you remember what the file is, and allows Praat to open the file as a script if dragged onto the Praat icon)
3. Begin writing, saving often

12.3.1 Using other text editors

Although Praat has an editor built in which is quite capable, praat scripts can be created from within any plaintext editor. Using an external editor allows you features like syntax highlighting (Praat script is similar to Perl in enough ways to make Perl syntax highlights vaguely useful), always-on line numbers (because Praat gives you line numbers where your script crashed and it's less of a pain to just scroll than to use *Script Editor* → *Search* → *Go to line…*), and the sorts of advanced find/replace tools present in more robust editors.

Some good editor choices include:

For Mac OS X:

- TextMate 2 - (Version 2 is free and open source) - `http://macromates.com/`
- MacVim - `https://code.google.com/p/macvim/`
- SublimeText - $$$ -`http://www.sublimetext.com`

For Windows:

- Notepad + + - `http://notepad-plus-plus.org/` - (also Scott Sadowsky's Notepad + + Syntax Highlighting package for Praat at `http://sadowsky.cl/praat.html#syntax`)
- TextPad - $$$ - `http://www.textpad.com/`
- SublimeText - $$$ -`http://www.sublimetext.com`

For Linux:

- Kate - `http://kate-editor.org` - (See José Joaquín Atria's Kate syntax highlighting package for Praat at `https://github.com/jjatria/praatKateSyntax`)
- SublimeText - $$$ -`http://www.sublimetext.com`

12.3.2 Filenames

Praat has a few odd quirks involving filenames which you'll need to work around in your scripting. First, Praat chokes on filenames with decimals in them. Be careful.

Also, absolute file paths in Praat scripts will need to be referred to differently on Windows machines vs. on Macs:

- directory$ = "c:\Documents\test data\" (Windows)
- directory$ = "/Users/will/Documents/test data/" (Mac)

On Unix-like systems,~ can be used to refer to the home directory (/Users/will/).

Relative file path names (relative to where the script is) do not need to change across platforms. For instance, so if you included the below directory assignments in your script:

- directory$ = "test data/"

(or)

- directory$ = "./test data/"

Praat would always look for data in a folder called "test data", in the same folder as the script itself. This script would function well on any operating system, provided the data is put in the proper folder.[15]

Danger!

Incorrectly formatted absolute path names are the most common issue which prevents you from running scripts you download from the internet. If your script won't run, check the file paths throughout the script and update them to match your operating system.

\end{tabular}

12.3.3 A Note on Praat Script Commands

In the Summer and Fall of 2013, Paul Boersma has updated the Praat script language, changing the syntax of some of the commands to be more familiar to people working with other programming languages. Although the old commands (as shown in the manual) do still work, new commands generated by the the process will look like:

[15] Thanks to Paul Boersma for pointing out this particular tip

```
do ("To Manipulation...", 0.01, 75, 600).
```

… versus the old command format …

```
To Manipulation... 0.01 75 600
```

Don't be alarmed if you see this new format. It's fairly straightforward to convert between the two formats mentally, and old commands do still work. I have not had the time to update all of the code in the manual for the new format, but anything you see here still works if dropped into a script.

That said, *new-style commands will cause scripts to crash in older versions of Praat*. This is yet another reason to download the newest version of Praat every time you start a new project.

12.3.4 How to magically write a Praat script (using the Praat "history" function)

Praat scripting relies on Commands, which are, effectively, verbs which describe the actions Praat should take. These commands can take a variety of forms, but they usually start with an uppercase letter (or, in newer versions, "Do ()"). Here are a bunch of Praat commands, selected randomly from several scripts:

```
Rename...
Extract part...
Get starting point...
select
Get high index...
To Manipulation... 0.01 75 600
Get first formant
Get frequency of maximum...
Erase all
(...and many, many more)
```

The most beautiful part of Praat scripting comes with the realization that if you can do something through the GUI, Praat will magically give you the exact commands to do it by script. Let's examine this wonderful reality by unintentionally writing a Praat script, using Praat's history function.

First, let's do a little bit of work in Praat, to simulate doing phonetic research:

1. Open a sound, and extract the vowel
2. *Praat Menu* → *New Praat Script…*
3. *Script Editor* → *Edit* → *Clear history*

Now, let's get the duration of the sound

4. Select the sound
5. *Objects* → *Query* → *Query Time Domain* → *Get Total Duration*

Now, let's get the average height of F1 and F2 of the sound

6. Select the sound
7. *Objects* → *Formants & LPC* → *To Formant (burg)…*
8. This creates a formant object from which we can get the mean formant values
9. Select the new formant object
10. *Objects* → *Query* → *To mean…*
    - This measures the mean for a given formant. Repeat this for Formants 1 and 2

Now, let's reap what we've sown:

11. Go back to your Script Editor window
12. *Script Editor* → *Edit* → *Paste history*

If all went to plan, your Script Editor window will have something like the code pictured below:

Code generated from the above process:

```
Edit
Extract selected sound (time from 0)
Close
Get total duration
To Formant (burg)... 0 5 5500 0.025 50
Get mean... 1 0 0 Hertz
Get mean... 2 0 0 Hertz
```

Each of those lines are the Praat script commands to do each of the tasks. `To Formant (burg)... 0 5 5500 0.025 50` turns a sound into a Formant object, with the settings specified in the trailing numbers. `Get mean... 1 0 0 Hertz`, intuitively enough, gets the mean of the formant specified in the first number, in the time range specified by the next two (0 = "all"), and gives it in Hertz (as opposed to Bark). Note, though, that because you used the mouse to do the selection (instead of using the commands in the *Editor* → *Select* menu), the exact definition of the selection doesn't show up.

In short, this is the kernel of a Praat script that does everything you just did, and this is one of the most effective ways of figuring out what commands to use in your Praat scripting. The process is always the same:

1. Open the Script Editor
2. *Script Editor* → *Edit* → *Clear history*
3. Do, using the interface, whatever actions you'd like to get the script commands for
4. *Script Editor* → *Edit* → *Paste history*

By doing this, you'll be able to see how to command Praat to do anything that you can do in the GUI.

12.3.5 Writing your first single-command script

Praat scripts can run the gamut between massive, 800+ line programs and single-function, single line editor scripts. Although the massive programs are often flashiest, in many cases, the single-item scripts are the ones you can't live without.

First, let's "write" a script which will turn the Spectrogram displayed in the editor window into a Narrowband spectrogram (as described in Section 7.2.1), using the history function. To do this:

1. Open a sound in the Editor window
2. Open the Script Editor and create a new script
3. *Script Editor* → *Edit* → *Clear history*
4. *Editor* → *Spectrum* → *Spectrogram Settings*
5. Set the *Window Length* to 0.025 (or the narrowband window length of your choosing)
6. Go back to the Script Editor window, then *Script Editor* → *Edit* → *Paste history*
7. Save the script wherever you'd like using *Script Editor* → *File* → *Save*

The resulting "script" will look like:

```
Spectrogram settings... 0 5000 0.025 50
```

That's it.

Because this code works entirely with the Editor window, you'll want to open it as an Editor script. You can use the *Editor* → *File* → *Open Editor Script...* command to open it and run it from there, or you can add it to a menu (likely *Spectrum*) using the procedure in Section 12.2.2.

Once you've done that, if you want to change the spectrogram type, you're only a click away. You'll likely end up with a few of these scripts in that menu, to change the spectrogram into broadband, narrowband, 0-10000 Hz broadband, and so forth. But sometimes, you need to get a bit more complicated.

12.3.6 Scripts with Variables

Variables are constructs in programming which are just labels which store pieces of information, either numeric or string (text). In Praat, they have a few characteristics worth noting[16]:

- Variable names have to start with a lower case letter (so as not to be interpreted as commands to the program) and must not contain spaces or start with a punctuation mark.
- String variables (containing text) must end in the character $.

[16]Thanks, yet again, to Dr. Rebecca Scarborough for portions of this text

- The "=" sign is used to assign values to variables (e.g., `f0 = 120` stores 120 in the variable `f0`).
- Values for string variables must be placed in double quotes. *
  - So, to specify a file's name: `soundname$ = "recording_1_suzanne"`

For a more concrete demonstration, let's take another task. In *Acoustic and Auditory Phonetics* (**?**), Keith Johnson provides a formula to calculate the length of person's vocal tract based on any formant of a neutral vowel. He gives this formula as:

`L =` $(2n-1)c/4F_n$

L = Length in Meters , n = Formant Num., c = the speed of sound in air (343 m/s)

Let's say you want to calculate this more often, and grow tired of manually calculating it. Instead, you'd like to be able to record a neutral vowel, then calculate the length based on F3 at the cursor's location.

First, you'll need to capture the F3 value at the cursor. By using the history command, you'll find that the command for getting F3 at the cursor is *Get third formant*.

However, this normally just pops the output of the command (the F3 height) into an info window. Because we need that information to be accessible to the script, we need to capture it as a variable. Although Praat's tutorial on Scripting has a wonderful section on Variables, and you should really read through that to get a full idea, in Praat, capturing the output of a command as a variable is easy.

To assign a variable to the output of a command, you just name the variable, then put an equals sign, then the command whose output you want to capture, like so:

```
variable = commandtogetoutput
```

Or, in our case:

```
f3height = Get third formant
```

The above line will then capture the output of "Get third formant" and store it. So, if F3 = 2600 Hz at the cursor, once this command is run, the variable `f3height` will be equal to `2600`.

Then, you'll use that number in the formula above to calculate a new variable, the length of the vocal tract. Remember, having run the first command, f3height will now be equal to the height of F3. Praat is perfectly capable of doing basic math, so we can hard-code the other parts of the equation into the script:

```
length = ((2*3 -1) * 343/(4 * f3height))
```

This will assign the output of `((2*3 -1) * 343/(4 * f3height))` to the variable `length`.

Finally, you'll want to convert the output (`length`), which is currently in meters, to a more useful unit, namely, centimeters:

```
lengthcm = length * 100
```

Then, you'll need to display this information in an information window. To do that, use the `print` command.

```
print Your vocal tract length is ‘lengthcm:1’ cm
```

This command will print the sentence "Your vocal tract length is", followed by the `lengthcm` variable's contents, followed by "cm". The `:1` attached to `lengthcm` rounds the output down to one decimal point.

Put together, this is your entire script:

```
f3height = Get third formant
length = ((2*3 -1) * 343/(4 * f3height))
lengthcm = length * 100
print Your vocal tract length is  ‘lengthcm:1’  cm
```

You would then save the script and either open it as an Editor script (because it relies on the mouse cursor within the editor window when getting the third formant), or add to a menu in the editor window. So, here, three steps of measurement and calculation are condensed into one easy menu selection.

12.4 About the Praat Scripting Language

We've already talked about variables, but there are several other programming constructs used in Praat scripting which you'll need to know about to successfully script.

12.4.1 'for' loops

In Praat, sometimes, you'll want to do the same thing over and over again, for, e.g. each item on a list, or for each vowel in a TextGrid. To do this, you'll use a `for` loop. For loops iterate through large amounts of data, doing the same thing many times over and over again.

`for` loops have the form:

```
for [variable] from 1 to [another variable]
    Take this action for each of them
endfor
```

In Praat, `for` loops usually have the format `for [var] from 1 to [other var]`, followed by an indented block, ended with an `endfor` (to tell Praat when to stop).

Here's a sample script:

```
select TextGrid ‘sounda’
number_intervals = Get number of intervals... 2
for k from 1 to number_intervals
    Set interval text... 2 k Vowel
endfor
```

This script selects a TextGrid (named 'sounda'), gets the number of intervals in the TextGrid, then goes through and changes the text for each interval in tier two of that TextGrid to "Vowel")

Really, though, the best way to understand `for` loops is to see them in action and to look through other scripts.

It's worth noting that, if you always plan to start from the first iteration, you can leave off the "from 1'' from the statement. So, the above could be written as:

```
select TextGrid 'sounda'
number_intervals = Get number of intervals... 2
for k to number_intervals
    Set interval text... 2 k Vowel
endfor
```

and it would function identically.

If you need to exit or restart a `for` loop, say, based on a conditional, the best approach is to use a `goto` and `label` (see Section 12.4.5):

```
select TextGrid 'sounda'
number_intervals = Get number of intervals... 2
for k to number_intervals
    Set interval text... 2 k Vowel
    if k > 10
        print "10 intervals already? I'm just going to call this done."
        goto end
    endif
endfor
label end
```

12.4.2 'if' statements

`if` statements tell Praat to take a certain set of actions **only if** the requested conditions are met. This condition is usually checking whether a variable is equal to (or greater, lesser than) a certain value, but it can get more complicated than that.

`if` statements have the form:

```
if variable = something
    Take this action
endif
```

Here are a few example if statements that you might see in a script:

```
vowel_label$ = Get label of interval... 1 2
if vowel_label = "i"
    Set interval text... 2 2 HighFrontVowel
endif
```

The above code will change the TextGrid's label to "HighFrontVowel" only if the vowel's label is "i".

```
vowel_label$ = Get label of interval... 1 2
if vowel_label != "i"
    Set interval text... 2 2 AnotherVowel
endif
```

The above code will change the TextGrid's label to `AnotherVowel'' only if the vowel's label is **not**i''`. As you can see, `if` statements are incredibly useful, and will show up constantly to control the flow of your scripts. You'll learn to love them very quickly.

You can also use `else` and `elsif` to allow yourself multiple choices:

```
vowel_label$ = Get label of interval... 1 2
if vowel_label = "i"
    Set interval text... 2 2 HighFrontVowel
elsif vowel_label = "u"
    Set interval text... 2 2 HighBackVowel
else
    Set interval text... 2 2 SomeOtherVowel
endif
```

Danger!

For much of its life, Praat script used <> to mean "not equal to" (rather than the usual != or /= or =/=). Although recent versions (mercifully) allow the far more common != to mean the same thing, be aware that older scripts will still include this odd <> notation.

\end{tabular}

12.4.3 'while' loops

`while` loops are meant to tell Praat to continue doing something until a certain condition is met. Let's say you wanted to add a second sound to a first sound over and over again until it was 3 seconds long:

```
select Sound 'soundname$'\_original
while chunk_duration < 3
    plus Sound 'soundname$'\_addition
    Concatenate
    chunk\_duration = Get total duration
endwhile
```

Here, a while loop is ideal, as it just keeps going until the criterion is reached with no further code, muss, or fuss.

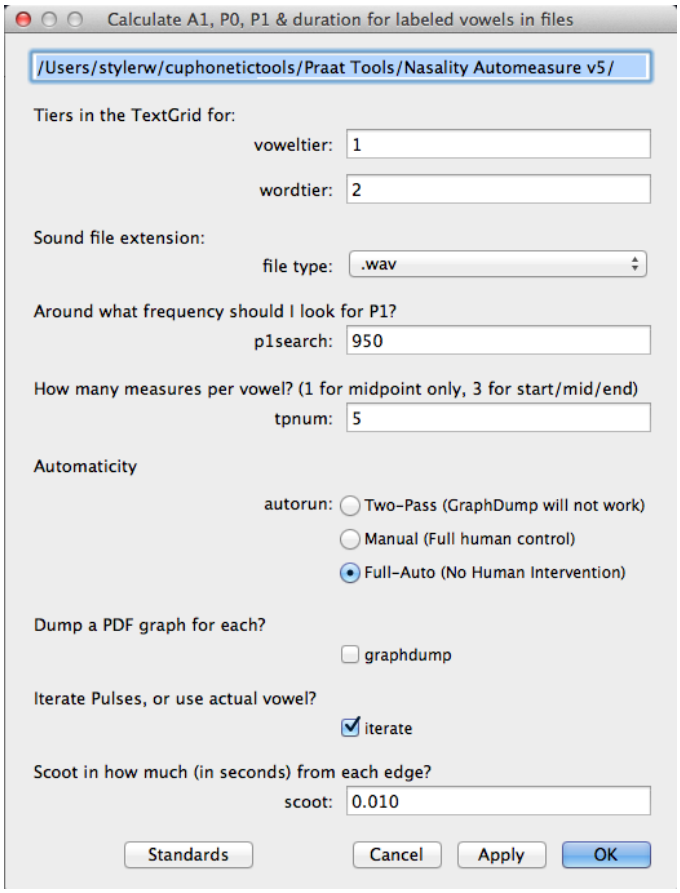

Figure 15: An example of a startup form from the CU Nasality Measurement Script

Danger!

While loops are great, but Praat (especially on OS X) acts funny if it has to do more than a certain number of iterations, as they can quickly fill up your memory. Especially if your while loop deals with something complex which might not always happen, if Praat starts crashing on certain tokens, your while loops are a good place to start looking.

\end{tabular}

12.4.4 Forms

Forms can run either at the start of the script, or during the script itself, and are your way of getting information from the user. Forms can be used to tell the script how to run, specify options, or have the users input starting information (say, pitch ranges or the number of measures per vowel). These let more complex scripts get user feedback before and while they do what they do best. For an example starting form, please see Figure 15.

Forms are created with blocks of code like the below:

```
form [Label for the form]
```

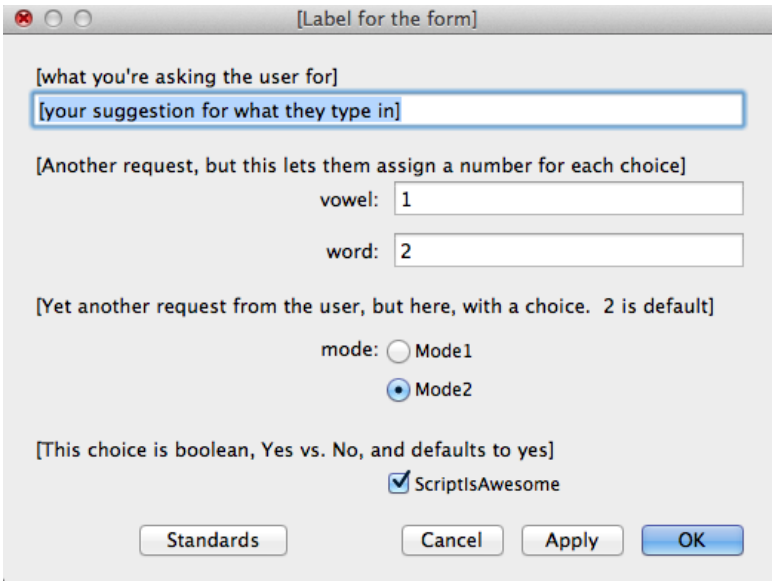

Figure 16: A form generated with the demo form code above

```
    comment [what you're asking the user for]
        text directory [your suggestion for what they type in]
    comment [Another request, but this lets them assign a number for each choice]
        integer vowel 1
        integer word 2
    comment [Yet another request from the user, but here, with a choice.  2 is default]
        choice mode 2
        button Mode1
        button Mode2
    comment [This choice is boolean, Yes vs. No, and defaults to yes]
        boolean ScriptIsAwesome Yes
endform
```

This code would create a form like below, in Figure 16.

You could then use that form's choices to make decisions later:

```
if mode = 1
    print "Your mode is set to 1"
elsif mode = 2
    print "Your mode is set to 2"
endif
if ScriptIsAwesome
    print "You're awesome!"
else
    print "Someday, you'll be awesome"
endif
```

12.4.5 goto and label

The `goto` command allows you to jump around within your script, and is generally used to exit loops, to re-try a procedure, or otherwise cause mischief. To use a `goto`, you must first specify a point in the script using the `label` command, and then `goto` that label:

```
select Sound ‘soundname$’\_original
while chunk_duration < 3
    plus Sound ‘soundname$’\_addition
    Concatenate
    chunk\_duration = Get total duration
    if chunk_duration > 10
        print "You overshot that bigtime.  Check your chunks."
        goto endscript
    endif
endwhile

label endscript
```

Danger!

For the most part, you shouldn’t use gotos except to exit loops. For most other uses, an if statement or while loop will accomplish the same thing, but in a more easy-to-debug way. Much like sticks of dynamite, gotos are very, very seldom the right tool for the job, and they’re just as likely to blow up in your face as they are to fix your problem. Think long and hard before you light that fuse.

\end{tabular}

12.4.6 Commented lines (#)

Not every part of a Praat script is designed to be read by a computer. Lines which start with a # symbol are called **Commented Lines**, and will be ignored by Praat. They’re usually written in by human programmers to explain exactly what the code is doing, or in some cases, as a header. Do yourself a favor and comment the code that you write, as sometimes, it’s the only way you’ll understand your code when you look at it days/months/years later.

Here’s an example of some commented code. Remember, Praat will only “see’’ the lines which don’t start with #:

```
# Now, this next chunk starts a for loop that just goes through
# the above directory and keeps reading in files each time it iterates
for j from 1 to number_files
    select Strings list
    filename$ = Get string... ‘j’
    Read from file... ‘directory$’’filename$’
```

```
# At this point, we now have the variable "soundname"
# which refers to the file being worked on
soundname$ = selected$ ("Sound")

...
```

12.4.7 Scaling scripts up: nowarn, noprogress, and avoiding the editor window

Lengthy praat scripts, meant to run on a great many sounds in a highly automated way, benefit greatly in terms of speed and efficiency by reducing the number of dialogs which pop up.

If you're doing something which modifies a sound in a way that occasionally produces a tiny bit of clipping (yielding an error message like "Advice: 2 of 34583 samples are clipped.'' upon saving), you can use the `nowarn` directive to save without warning. For instance:

```
select Sound ‘soundname$’_mod
nowarn Write to WAV file... ‘directory$’output/‘soundname$’_modified.wav
select TextGrid ‘soundname$’
Write to text file... ‘directory$’output/‘soundname$’_modified.TextGrid
```

This will save the file as a .wav file, and completely ignore any errors resulting from clipping. Mind you, this means that there will be some clipping in your file, which is a Bad Thing. One or two samples may be fine, but in a production script, you'll want to Scale Amplitude or Scale Peaks to make that stop.

If you're doing the same process over and over again in a highly automated fashion on a fast machine, you'll notice that Praat spends more time rendering the "Progress: To Pitch…'' type of dialogs than it does making the pitch file. To suppress the generation of these dialogs (letting Praat run more silently in the background), use the `noprogress` directive:

```
select Sound ‘soundname$’_mod
noprogress To Formant (burg)... 0 formnum formrange 0.0256 50
noprogress To Pitch... 0 60 ‘crazyhighh1’
```

This will generate the formant and pitch objects without displaying a dialog, allowing the process to run much more quickly than it otherwise would have. This will have the side effect of effectively preventing the user from stopping the script mid-run (without force-quitting Praat), but the gain in speed may offset this inconvenience.

Finally, it's important to realize that as you're scripting, anything that actually displays on the screen will be more resource- and time-intensive than a background task, especially if a spectrogram is being generated. As such, if you have the choice of doing a given process using the editor window (Open Sound in Editor, Move cursor to point X, Get First Formant, Get Second formant, close editor window) or creating and querying an object (To Formant…, Get Value at time 1 timepoint, Get Value at time 2 timepoint), it will certainly be faster to use an object and leave the editor closed.

Editor scripting is great, especially as you're just starting off, but if you're planning to run on a large number of items, removing Editor Scripting from the mix will almost certainly result in a speed increase over huge datasets.

Danger!

When you have purged all user-interface-generating elements from your script, at least on macOS, once you start your script running, Praat will appear to "crash", with the user interface becoming unresponsive, even to the extent that the file chooser dialog can't be moved. This is simiply a consequence of your script's efficiency coupled with Praat's coding. You'll see that data and files are saving happily to the designated directories, and checking Activity Monitor will reveal heavy CPU usage, but Praat will appear dead-to-the-world. When the script finishes (or errors out), Praat will return to life.

\end{tabular}

12.4.8 Useful tips

These are a few tips and tricks that I learned well after they would've saved me hours upon hours of time. I'm passing them along in hopes that they'll save you the time more quickly.

- To refer to an object in the Objects window, it's best to give both its type (Sound, TextGrid, etc) and its name. So, instead of coding `select sounda`, you'll want to use `select Sound sounda`.
- If your code is failing, sometimes the line numbers in the crash messages won't tell you exactly how far the script is getting before it crashes. To test this, insert the word "fail'' (or anything else that isn't a real command) into your script at a certain point. The moment it hits that word, the script will crash, and you'll know you've gotten that far. Then repeat until you crash for some other reason. At that point, you'll know exactly where your script goes off the rails.
- Scripts will sometimes fail talking about a certain variable being `--undefined--`. This usually means that you're looking for a pitch or pulse in a place where Praat's pitch tracking can't find voicing. To help prevent these failures (which will crash the entire run of a script), you may considering adding a small block of code like the following, which will only take a certain action if the answer is defined:

```
if pulse_begin_time <> undefined
    Get pitch...
endif
```

- Once again, comment your code. If I had a nickel for every time I've opened up a script I wrote, only to find myself unable to understand what I had written, I'd have a lot more nickels.
- A great many issues in Praat scripting can be resolved by looking at a timepoint just 5-

10 ms to either direction. Adding in a loop which, if an unreasonable measure (or an `--undefined --`) is encountered, tries again at a nearby second timepoint, can really increase the robustness of your scripting.

- If your script is automatically creating lots of objects, you'll want to clean house periodically. Although it's ideal to hard-code the deletion of every item manually (select Sound sounda, Remove), sometimes you wind up generating a large number of no-longer-needed files. A little chunk of code along the lines of the below will do the trick nicely:

```
select all
minus Sound 'soundname$'
minus TextGrid 'soundname$'
minus (whatever other files you care about)
Remove
```

- To check if a number is even or odd in a Praat script, use the "mod" command:

```
oddness = number mod 2
if oddness = 0
  print "Number is even"
elsif oddness = 1
  print "Number is odd"
endif
```

- If you're using editor scripting, Praat will not naturally close the editor window, and will fail after 5 copies of the same window are open. Make sure to include a 'Close' command. So, for instance, in my formant script, after pausing for human confirmation, I have the following, which grabs control of the open window and then closes it:

```
  Edit
editor Sound 'soundname$'_word
         Close
  endeditor
```

12.4.9 Everything Else

Praat scripting takes a long time to really get your mind around, and there are all sorts of other constructs which can be useful in working. You would be well served, once you've got your mind wrapped around the basics, to examine the Praat Scripting tutorial pages (*Objects* → *Help* → *Praat Intro* → *Scripting*) for some of the following concepts, which will improve your scripting efficiency even more.

- Scripting 5.3. Jumps
  - This section talks about using `elsif` and `else` to improve your script's flow

- Scripting 5.5. Procedures
  - This section talks about using Procedures to streamline your code and minimize repetition
- Scripting 6.6. Controlling the User
  - This section talks about using Pause forms to ask the user questions and to confirm measurements.
- Scripting 7.7. Scripting the Editor
  - This section will teach you how to work within the editor window in Praat scripts.

12.5 In defense of Code Cannibalism

There is very seldom any need to reinvent the wheel, and there is no shortage whatsoever of Praat scripts available online and floating around departments. Although it's unlikely that you'll happen to find a script which does *exactly* what you need, chances are very good that what you need can be cobbled together from several scripts. So long as you give attribution to the original author(s) of the script(s) you borrow code from, there's no shame in reusing parts of other scripts as you work on your own.

Also, it's worth noting that one of the best ways to learn how to code is to find a script, see how it works (in terms of what it does) by running it, and then looking at the script's code to find out how it does what it does. Once you've started collecting scripts, you'll find yourself reaching for certain chunks of code every time you, say, want to open and analyze every file in a given folder, or need to generate a plot showing nasality in a given word.

To that end, I've published a collection of my most favorite scripts at `https://github.com/stylerw/styler_praat_scripts`. Among them is demo_formant_script.praat, a very basic formant measurement script that takes in TextGridded sound files and spits out the formant readings for those files. This script was written specifically to be used for learning and cannibalism, and is commented heavily. There's much to learn from these scripts, although many have been written using an older version of the Praat scripting language.

In short, the more time you spend nosing around scripts, the better.

12.6 Closing Remarks on Praat scripting

Scripting can very quickly start to save you some time, and as such, the payoff for even a small amount of work can be great.

However, to get really incredible results, and to trim hours off of your measurement tasks, you'll need to put more time in. You'll need to code, then code some more, then go steal somebody else's code, then re-code it because they did it wrong. Then you'll need to spend lots of time with Praat's tutorials, then spend lots of time on the Praat users list. You'll add and remove quotes until you never want to see one again, and add loops until you're blue in the face.

But then finally, that one script that you've been working on *forever* will work. You'll start it, fill in the form, and it will run like a cheetah after a hyperactive gazelle. And it'll run the next time. And the time after that. And each time you start up your script, it'll keep running, and you'll see the time savings mounting.

Then, one day, you'll do the math and realize that a task that would have taken you 1900 hours of click-by-click hand measurement only takes twelve minutes to run by script, and you've just saved yourself 79 days of pain[17]. At this point, you'll realize that all that time, that agony, totally paid off, and rather than spending the afternoon taking measurements, you can just start a script running to do it for you, then go actually enjoy your life.

That, my friends, is why we Praat script.

13 Advanced Techniques

Sometimes, you will sit down and realize you need to do something ambitious, weird, or crazy in Praat (or elsewhere) to accomplish what you need to. This section serves to detail, with no particular structure, some of the methods I've used for accomplishing various crazy tasks.

Danger!

The techniques described here are fairly esoteric and require some deeper understanding of the nuances of Praat. They're here not so much because you'll likely need them in your phonetic career, but so that they're available, findable, and can save some work for those few people who actually do. Read on if you're curious, but if you're just getting started, feel free to stop here!

\end{tabular}

13.1 Getting Amplitude Envelopes and AM Demodulation in Praat

AM ("Amplitude Modulation") is the process of embedding a lower frequency signal inside a higher frequency signal, and is the process underlying AM radio. AM Demodulation, then, is the process of undoing this, which can be useful when dealing with machine generated signals, and the process of AM demodulation is identical to the process of obtaining an amplitude envelope, which can be occasionally useful for speech analysis.

Although the human voice doesn't use AM modulation meaningfully, it can sometimes be used for transmitting low-frequency signals generated by specialized hardware, such as airflow measurement systems or articulator movement trackers. This is because most consumer or pro-audio level analog-to-digital converters filter out (or capture poorly) information under 20 Hz or so. As such, some systems which don't want to use higher-level capture boxes modulate information below 20 Hz into the amplitude envelope of a higher frequency sound, and then recover the information in software using AM Demodulation.

[17] As an aside, "doing the math" is an excellent way to justify the principled and cautious use of scripting and automation for data collection to reviewers or a dissertation committee.

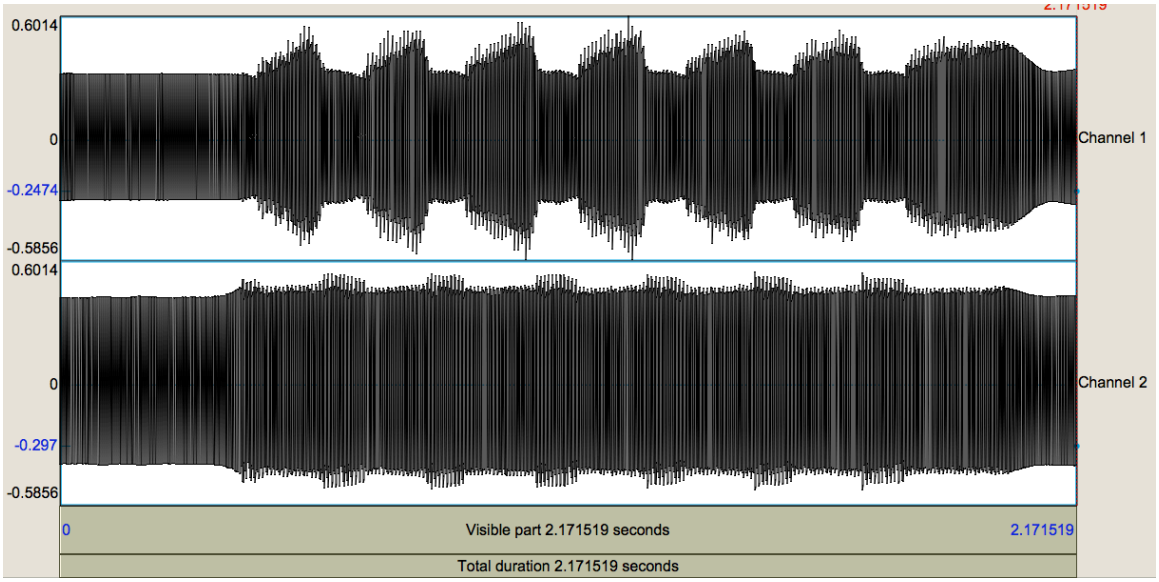

Figure 17: Here's a sample modulated Oral and Nasal Airflow signal showing /mamamamamamama/ with oral flow on channel 1. Note that the amplitude envelope seems to show meaningful information beyond the steady signal.

Once you've confirmed that AM modulation is happening (look for a high frequency "carrier" sound which doesn't vary in frequency, but varies sharply in amplitude), it's very easy to demodulate. As with demodulating any AM signal, there are two steps.

First, we'll need to rectify the signal, allowing only positive signals to pass through. In Praat, we can accomplish this by doing either of the following:

1. Open the Sound in Praat, and Select it
2. *Objects* → *Modify* → *Formula…*
3. Enter 'abs(self)' as the formula, and press "OK"[18]

This simply tells Praat that every point should be interpreted as positive. Then, we filter the sound to smooth the curve:

1. Select the rectified sound
2. *Objects* → *Filter* → *Filter (Pass Hann Band)…*
3. *From:* 0, *To:* 100, *Smoothing:* 20
   - These parameters will vary depending on your data.

This will leave you with a positive trace representing either the unmodulated signal or the amplitude envelope of the selected sound.

[18]You can also use 'if self < 0 then 0 else self endif' as the formula here for a slightly more classical rectifier feel, but in my experience, it doesn't make a meaningful difference.

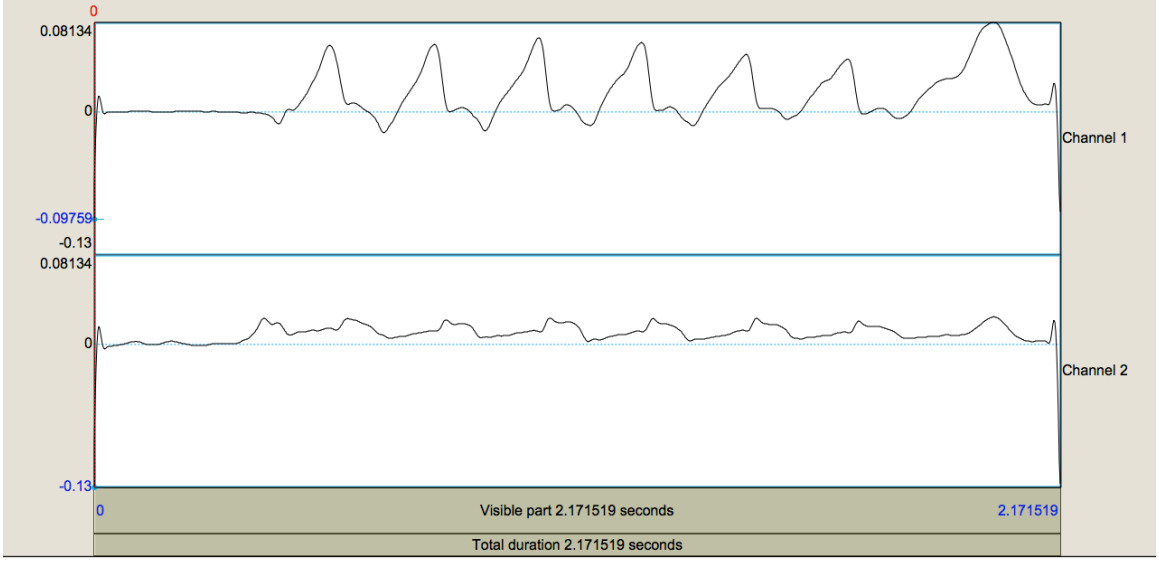

Figure 18: Here's the same sample modulated Oral and Nasal Airflow signal showing /mama-mamamamama/ with oral flow on channel 1, after demodulation. Here, the airflow patterns are plainly apparent, and measurable without the noise.

Note, though, that the output is not a "Sound" in the classical sense, and is inaudible if listened to. These "Sounds" merit some discussion.

13.2 Breaking the "Sound" barrier: Working with analytical "sounds" in Praat

Although we think about "Sounds" as being audible signals, a "Sound" object in Praat (and indeed, any digitized sound) is really just a series of "Amplitude at Time" pairs. Although this certainly can be listenable "Sound", as we saw above in 13.1, we can also import, generate, and analyze inaudible signals like airflow or articulatory traces.

Understanding this allows the "Create Sound From Formula" feature (and its friend, *Objects* → *Modify* → *Formula...*) to be an incredibly useful tool for more complicated signal analysis and graphing.

Although this is applicable to any signals, since we just discussed nasal and oral airflow, let's discuss a common (albeit problematic) measure of nasality in airflow: %Nasalance. %Nasalance is designed to capture the degree of nasality at any point in the vowel or word, by capturing how much of the total flow at any given moment is coming from the nose.

One common way to calculate %Nasalance is $\%Nasalance = \frac{nasal flow}{(oral flow + nasal flow)}$. Although you could export the data as a series of timepoints and easily generate this in R or a spreadsheet program, you might want %Nasalance over time to act like a "Sound" in Praat. If it does, you can easily graph the outout within Praat, or output all the relevant information in a single script, or pull %Nasalance at only a specific time.

To do this analysis and turn it into a "Sound" to be graphed or queried, we can use

Objects → *New* → *Sound* → *Create sound from formula…*

We'll assume that you've got two "flow" signals, similar to those shown in Figure 18 above, called 'oral_flow' and 'nasal_flow'. We're going to generate a new "sound", where each point is mathematically determined by the same points in these two other sounds. This is best done by script (so that the duration and sampling rates are close at hand), but can be done manually, as below:

1. Extract the flow into two sounds, called 'oral_flow' and 'nasal_flow'.
2. *Objects* → *New* → *Sound* → *Create sound from formula…*
   - Name is whatever you'd like
   - One channel
   - The start time is '0', the end time is the duration of the oral and nasal flow files (their durations should be identical)
   - Sampling frequency must match the flow files

The most basic formula we could use would be:

```
Sound_nasal_flow [col] /(Sound_nasal_flow [col] + Sound_oral_flow [col])
```

This implements the formula exactly, and says that the amplitude value for the new file at any given point is equal to $\frac{nasalvalue}{(oralvalue+nasalvalue)}$.

However, we might want to be a bit more robust. Nasalance doesn't make sense when either the oral and nasal flow is almost zero (a miniscule bit of nasal flow during an inhalation is not really the same as "100% nasal flow"). A more advanced formula, which also demonstrates the use of 'else if' in a formula in Praat, would set nasalance to zero when either oral or nasal flow is below a certain level:

```
if Sound_nasal_flow [col] < 0.02 then 0 else if Sound_oral_flow [col] < 0.02
then 0 else Sound_nasal_flow [col] /(Sound_nasal_flow [col] + Sound_oral_flow
[col]) endif endif
```

Put into prose, this is saying:

> If nasal or oral flow is below a certain threshold (0.02 here), nasalance is 0. Otherwise, it's $\frac{nasalvalue}{(oralvalue+nasalvalue)}$

This results in a new "sound" which is *nothing* like a sound (see Figure 19), as it contains only values from 0 to 1 over time, but which can be queried, drawn (see Figure 20 , filtered, or extracted using Praat scripts.

Of course, this same technique of generating, analyzing, and graphing 'sounds' which aren't really sounds can be used for any sort of data. The sole requirement is that the data can be expressed as a single numerical value which changes over time. It's a powerful technique, and can save a great deal of Praat-external analysis, but it also requires considerable understanding of the mathematics of digital signals, and of exactly what you need to do.

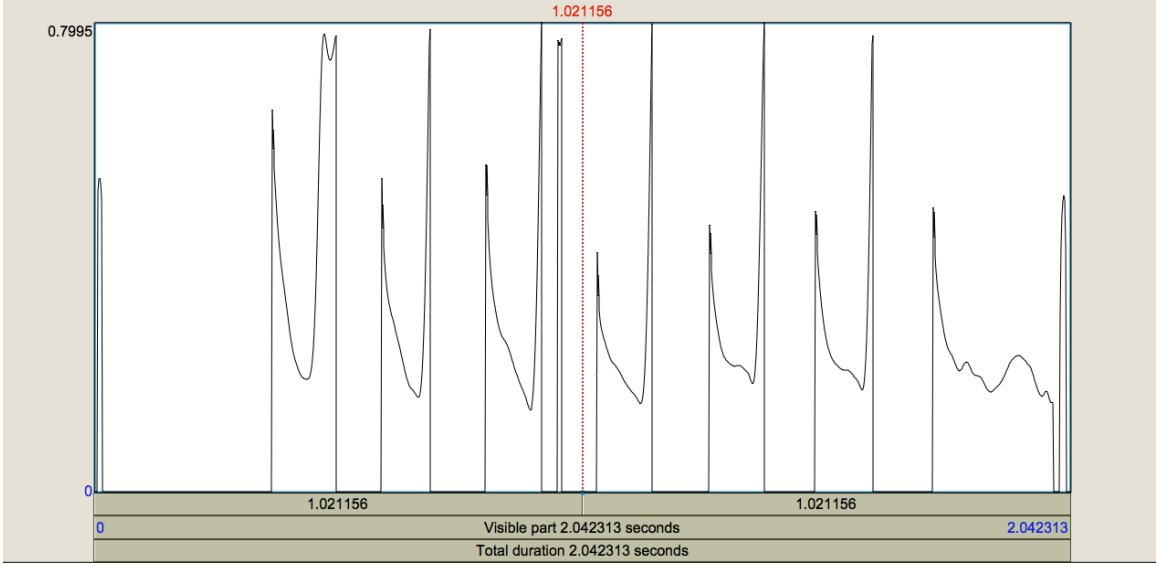

Figure 19: Here's the resulting %Nasalance file resulting from the /mamamamamamama/ flow signals in Figure 18. Below is the same sound, graphed more attractively using the Picture window.

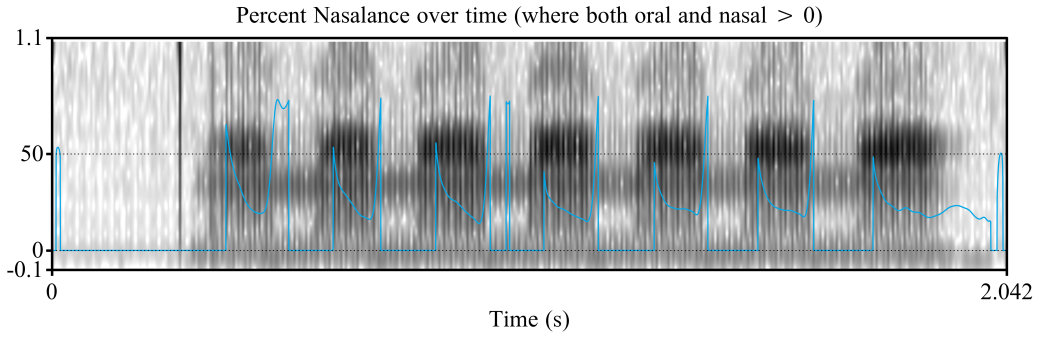

Figure 20: This is the same %Nasalance file resulting from /mamamamamamama/ in Figure 19, graphed more attractively using the 'draw' command and the Picture window.

Danger!

DC signals like these (which don't move back and forth around 0) can be dangerous to some kinds of speakers and headphones (as a constant level of current is pushed through the wiring without any break or reversal for the wiring to cool down). Although a single playback of a short DC file is unlikely to hurt anything (although it won't be audible), hitting "Play" on an hour-long, high-amplitude DC track could damage your speakers or headphones in some scenarios. With great (DC) power comes great responsibility.

\end{tabular}

14 Conclusion

Praat is unquestionably powerful software. Although there are other packages and tools which may offer some improvements in some specific domains, there is no other program which can do even half of what Praat can do without resorting to scripting. And, on top of that, unlike its only serious contender (Matlab with other signal processing toolkits), Praat is freely available and open source.

I hope this guide has helped you learn some of the basics of Praat in a linguistics and speech-science context, and has shown you some of its potential for more complex (and automated) analysis. But, you're far from "done", because sooner or later, as you ask more and more intricate questions, you'll find an analysis you need to do or a number you need to get that Praat simply doesn't have a command for.

At this point, remember that because Praat is also a toolkit and a scripting language, the possibilities for conducting research are frighteningly close to endless, so you can probably do what you need to do within Praat.

Even after 15 years, I'm regularly running into new things which I need to do in Praat, but don't quite know how to. If you're struggling with a new problem, you can do the same things I do. Read the Praat documentation, experiment with other approaches, look for other people's scripts, or even post to the Praat Users Mailing List (See Section 3.2), where Paul Boersma himself answers questions, even about deeply esoteric issues. But each time I run into a wall or suspect something isn't doable, after I think about it a bit, mess around with it for a while, or break down and ask somebody else, I usually find a way.

In closing, I'd wish you easy analyses, but those are usually boring. I'd wish you smooth sailing, but this is speech, so that's never going to happen. I'd wish you good data, but speakers won't provide it, and I'd wish you pleasant work, but sooner or later, the abstract will be due in 1.5 hours.

So, instead of all that idealistic tripe, I leave you with one simple wish, for your use of Praat and your life in Linguistics overall:

May you find a way.

References

Chen, M. Y. (1997). Acoustic correlates of English and French nasalized vowels. *The Journal of the Acoustical Society of America*, 102(4):2360–2370.

Gordon, M., Barthmaier, P., and Sands, K. (2002). A cross-linguistic acoustic study of voiceless fricatives. *Journal of the International Phonetic Association*, 32(2):141–174.

Gordon, M. and Ladefoged, P. (2001). Phonation types: a cross-linguistic overview. *Journal of Phonetics*, 29(4):383–406.

Simpson, A. P. (2012). The first and second harmonics should not be used to measure breathiness in male and female voices. *Journal of Phonetics*, 40(3):477–490.

Styler, W. (2017). On the acoustical features of vowel nasality in English and French. *The Journal of the Acoustical Society of America*, 142(4):2469–2482.