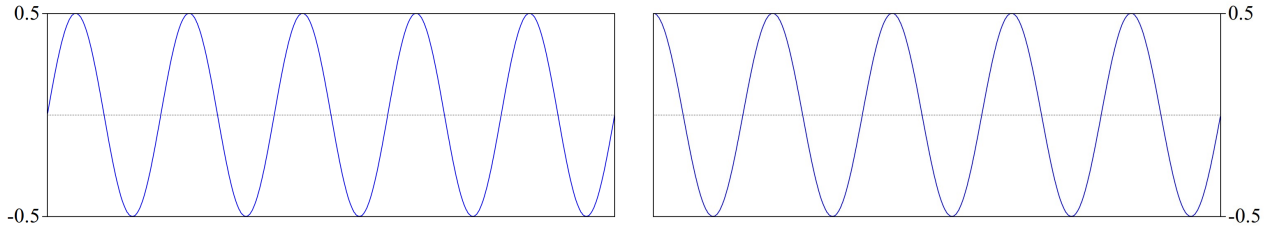
Session 5 (09:00 – 13:00, Fri.): The perception domain

- Creating stimuli: cross-splicing (zero-crossings and overlap); RMS intensity normalization.
- Creating continua: some pointers and tips; working example for vowel continuum from reference values.
- Running experiments: Experiment MFC, Demo window; Open Sesame.
- Scripting challenges: preparing, programming and extracting results from a perception experiment in Praat.

5.1. Creating stimuli

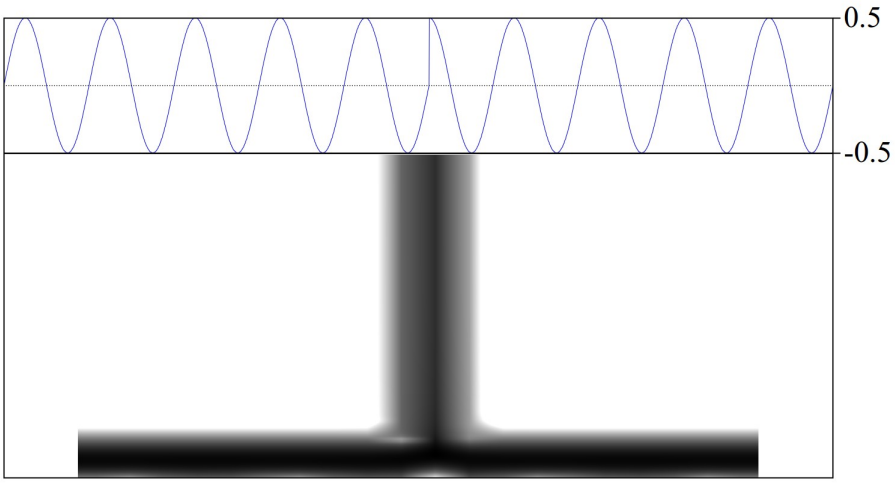
5.1.1. Cross-splicing

- **Cross-splicing** is a fairly common task when preparing stimuli for perception experiments. However, moving one section of speech and just concatenating it next to another (i.e., **splicing**), can be a source of trouble and should be avoided in most cases.
  - Consider for example the following two sine waves. Both have exactly the same fundamental frequency, intensity range and sampling frequency, and thus sound like very good candidates for splicing.
  - However, the end of the first section is ascending and exactly at the zero-crossing point, whereas the start of the next section is at the maximum of the sine movement[1].

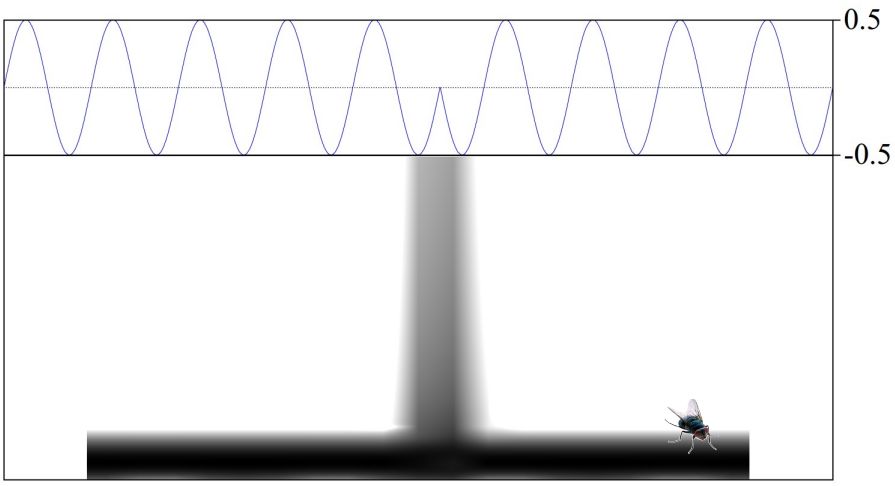


  - If we were to try to concatenate these two sines at the point where the first one ends and the second starts, we will cause a sudden change in the direction of the curve's trajectory, which will be perceived by listeners as some sort of noise.
  - Indeed, when we splice together these two sections we end up with a click-like sound in the intersection, as illustrated below with the waveform (which doesn't really look so bad) and a spectrogram (which doesn't look good at all: there is a click right in the middle).

[1] The scripts to create all these images are located in your folder “companion_folder_session_5”, in a file called “_creating_sine_images.praat”.



- Notice that splicing (= just concatenating) at **zero-crossings**, which is often suggested as a solution to these problems, does not guarantee that the waveforms will be moving in the same trajectory:



- Moreover, most sounds aren't pure sine waves, and thus it is just (practically) impossible to try to adjust the place of concatenation until the trajectories match well enough to prevent these clicks from happening.

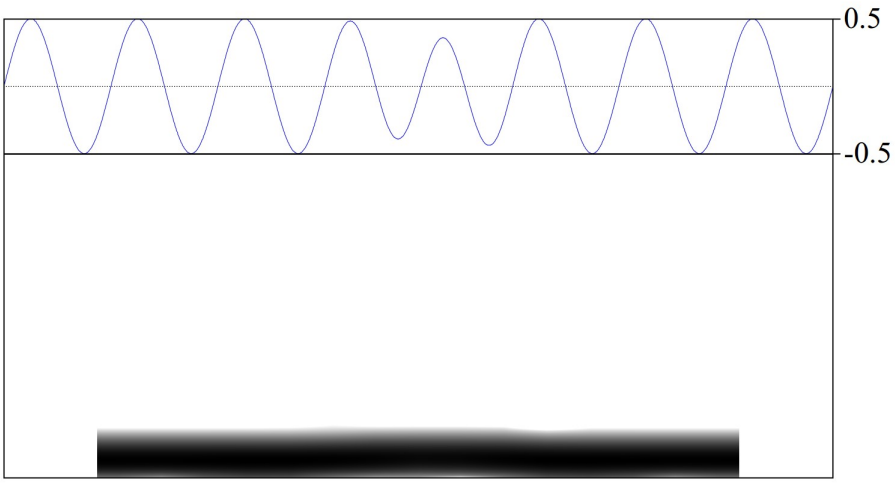
- A solution to this problem is **cross-splicing**. When two signals are cross-spliced, an amount of overlap is defined, and the portions of the signals that overlap (the ending of the first part and the beginning of the second) transition into each other smoothly.
- So, **how to conduct cross-splicing** in Praat?
  - When two or more Sound objects are selected in Praat, a series of commands become available in the dynamic menu, under `Combine -`. In particular, you can `Concatenate`, `Concatenate recoverably` or `Concatenate with overlap`.
  - The first two options splice the two sounds together without intervening the signals. In the case of the second command, “recoverably” means creating a TextGrid that is associated to the newly concatenated sound, and that shows the place where the two sounds were spliced, as here:

```
id_1 = Create Sound from formula: "sine200", 1, 0, 0.5, 44100,
  ... "1/2 * sin(2*pi*300*x)"
id_2 = Create Sound from formula: "sine400", 1, 0, 0.5, 44100,
  ... "1/2 * sin(2*pi*400*x)"

selectObject: id_1, id_2
Concatenate recoverably
View & Edit
```

(5.1)

- The command **concatenating with overlap** allows you cross-splice two signals. The only argument that it requires is the amount of overlap in seconds. This function works surprisingly well, and not only for sounds that were similar to begin with (as in the examples above and below), but for any sort of sounds.
  - When we concatenate the sections from above, both trajectories meet gracefully, and the spectrogram shows no clicking sound:



  - Do keep into account that the sum of both sounds after concatenation is shorter than the the combination of the two original sounds (shorter by the amount of overlap, of course).

- When creating stimuli, try **several overlap values** until obtaining **satisfactory results**. Do keep track of the overlap values that you end up using, given that you may have to report them when describing your stimuli.
- At least in my work, most of the cross-splicing has been done **after synthesis** processes and I need to concatenate synthetic sections of stimuli to non-synthetic sections.

- **Time to have some fun 5.1**:
  - Create a very short sound from formula (e.g., 50 ms = 0.05 s). You can find this command in `New > Sound > Create Sound from formula...`.
  - Split the sound in half and then remove different tiny bits from the beginning of your second half.
    - You can use the editor window: select the portion you want to remove from the sinewave and then use `Edit > Cut` (“Ctrl. + X”).
  - Concatenate the edited two halves recoverably and see the effect of combining non-matching sines by looking at spectrograms of the signal.

- Concatenate the edited halves with overlap, trying different combinations, and see the effects of doing this by looking at spectrograms.

5.1.2. RMS intensity normalization

- When you have a bunch of sound files for perception experiments you need to make sure that their intensity has been homogenized so that they are effectively comparable.
- Praat provides two options for intensity scaling: `Scale peak...` and `Scale intensity...`.
  - **Scale peak** takes a single argument, which is the new absolute peak for the signal that is being affected. Notice that if you have a sound that has low relative amplitude almost everywhere and a **sudden intensity peak**, this relationship between the low amplitude and high amplitude peaks will remain unmodified.

  - You can also use **scale intensity**. This function multiplies the amplitude of each sound so that, on average (root-mean-square = RMS), the intensity of the whole sound becomes the one selected by the user, and not only that of the peak.
    - In most cases that I can think of (unless your work is about peaks, I suppose), **RMS** normalization **should be preferred**, because it provides more homogeneous sounds with respect to intensity.
    - This Praat command, however, has a weakness, which is that some samples might get **clipped**, which is then perceived as distorted sound by listeners.
      - You can read the documentation to understand exactly why this happens, but in essence Praat is ignorant of potential big fluctuations in your signal, and thus the average intensity might very well require sections of the signal going beyond allowed values.

    - An alternative version for the RMS normalization procedure can be found in CPrAN's plugin sndutils. This version of the RMS normalization **prevents clipping** from happening by preliminarily inspecting the batch of Sound objects that will be normalized and then setting the average intensity as the closest one to the one selected by the user in which no clipping occurs in any of the normalized objects.
      - Follow the links that I've provided above to install this plugin for Praat. CPrAN is currently in development, so do contact the developer if you have issues installing the plugin or me if you're having trouble making it work.

5.2. Creating continua

5.2.1. Some pointers and tips

- **Bad models = bad synthesis**. Your synthesis is only going to be as good as your acoustic model of the object you're trying to synthesize.
  - By a model I'm referring here to the acoustic characteristics of the unit you're trying to synthesize. So, for vowels, a model would normally include oral formant values, oral formant bandwidth values, intensity values for those formants, duration, etc.

        - A model is usually built by measuring good individual examples of your target unit (e.g., good and representative vowels) or by using reference values, when they are available.
    - If a bad or poor model comes in, a bad synthesis comes out.
    - Including more parameters to your model and your synthesis is normally beneficial.
        - Be careful, though, not to loose control over the acoustic variables you want to talk about.
        - In cue weighting experiments, for example, you need to be certain that your variable of interest are represented faithfully.

- **Still a bit of a black art**: Creating truly natural sounding stimuli requires a lot of fiddling and trial and error.
    - I've heard respectable researchers refer to Klatt synthesis as a bit of a “black art”.
    - Be prepared to spend a considerable amount of time writing a script to create stimuli, and failing several times.
    - Be aware of the dangers of using other people's scripts as well. If you don't understand them, you shouldn't be using them for research that's going to get published.

- **Keep a log of settings**:
    - As mentioned above, expect to spend some time trying different **settings** until your stimuli are good enough.
        - By “settings” I mean, for example, boundaries in a TextGrid that is used to define boundaries between segments, amount of overlap, number of steps for a continuum, duration, etc.
    - Once you finish a script to synthesize sounds, and you're trying different settings to obtain the best results (e.g., overlap for cross-splicing), report the settings you used at each iteration of your building process (e.g., “attempt 1”, “attempt 2”) by sending them into an external log, that you can read afterwards if you need to replicate the process.

- **Keep a log of the outcome**:
    - Whether you'll use your script for your dissertation or for publication purposes, you'll need to report the actual characteristics of your stimuli (of the synthesized stimuli, not of the intended model).
    - It is easier to obtain this information in the same script that generates the stimuli, so don't forget to add a section that includes this.

5.2.2. Working example for vowel continuum from reference values

- We'll take a careful look at an example of a relatively simple script that creates a vowel continuum from reference values.
- The script receives a series of acoustic parameters from two vowels to calculate intermediate steps and populate KlattGrids, which afterwards are synthesized to sounds using Klatt synthesis.
    - D.H. Klatt & L.C. Klatt (1990): "Analysis, synthesis and perception of voice quality variations among male and female talkers." *Journal of the Acoustical*

Society of America 87: 820–856.

- We'll be looking at the code contained in the file “01_vowel_continuum.Praat” in Sublime Text, which you can find in your “companion_folder_session_5” folder, online (the code is too long to be copied here, and is difficult to read without syntax highlighting).

- **Time to have some fun 5.2**: Save a new version of the script “01_vowel_continuum.Praat” and name it “01_vowel_continuum_MODIFIED.Praat”. Adapt that script so that it is able to query the relevant acoustic parameters of two vowels from a Sound object, which have to be annotated in a TextGrid, and then build a continuum between these two vowels.
  - Basically, you need to query the acoustic parameters instead of using reference values.
  - Use the 2 vowels contained in the file “01_vowels_e_a.wav”.
  - In order to complete this challenge, you'll need to do the following:
    - Inspect the sound in Praat and generate a TextGrid (outside your script). Annotate that TextGrid manually to define the boundaries of the vowels /e/ and /a/ and the label of each one.
    - You script should be able to load both the sound file and the annotated TextGrid.
    - It should be able to query the relevant intervals from your TextGrid to know where in the time domain are the vowels located.
    - It should generate Praat objects so that you extract acoustic parameters from them (at least, generate a Formant object)
    - It should be able to query the generated objects to extract acoustic parameters. You can keep duration, pitch and intensity stable (equal for all stimuli), but do query oral formant values and oral formant bandwidth values at least.
    - It should be able to pass all these parameters to the current engine that creates the stimuli.

5.3. running experiments: Experiment MFC, Demo window and Open Sesame.

5.3.1. Experiment MFC

- You can run **Multiple Forced Choice** experiments in Praat (full documentation here). Among other tasks, you can run identification and discrimination experiments using this interface. Also, as stated in the documentation, given that Praat runs in all major platforms, and that Praat is free, it is a good alternative for small experiments, pilots and such.
  - MFC experiments are actually easy to code in Praat. We'll take a look at an example, which is a modified version of the code provided in the documentation (you can find this script in your folder “companion_folder_session_5”, in a file named “02_MFC_experiment_example.praat”).
  - In order to use MFC experiments in Praat, you need to adapt an experimental script that will look very similar to this (**let's read this carefully**):

```
"ooTextFile"
"ExperimentMFC 6"
blankWhilePlaying? <no>
stimuliAreSounds? <yes>
stimulusFileNameHead = "mfc_files/"
stimulusFileNameTail = ".wav"
stimulusCarrierBefore = ""
stimulusCarrierAfter = ""
stimulusInitialSilenceDuration = 0.5 seconds
stimulusMedialSilenceDuration = 0
stimulusFinalSilenceDuration = 0.5 seconds
numberOfDifferentStimuli = 8
    "01_mesón" ""
    "02_batería" ""
    "03_diagonal" ""
    "04_pulpo" ""
    "05_mugre" ""
    "06_corcho" ""
    "07_ceja" ""
    "08_suponer" ""
numberOfReplicationsPerStimulus = 1
breakAfterEvery = 4
randomize = <PermuteBalancedNoDoublets>
startText = "Instructions here...

Click to start."
runText = "Local instructions..."
pauseText = "You can have a short break if you like. Click to proceed."
endText = "The experiment has finished."
maximumNumberOfReplays = 0
replayButton = 0 0 0 0 "" ""
okButton = 0 0 0 0 "" ""
oopsButton = 0 0 0 0 "" ""
responsesAreSounds? <no> "" "" "" "" 0 0 0
numberOfDifferentResponses = 2
    0.3 0.4 0.5 0.6 "Yes" 40 "" "1"
    0.5 0.6 0.5 0.6 "No" 40 "" "0"
numberOfGoodnessCategories = 5
    0.25 0.35 0.10 0.20 "1 (poor)"
    0.35 0.45 0.10 0.20 "2"
    0.45 0.55 0.10 0.20 "3"
    0.55 0.65 0.10 0.20 "4"
    0.65 0.75 0.10 0.20 "5 (good)"
```

(5.2)

- Remember to provide a legal **path** to the experimental files!
- To run the experiment, read the experimental file in Praat and select `Run`.
- Once you've ran the experiment, click on `Extract results > Collect to Table` to, well, extract your results (**let's take a look** at some results).
- Save your results as a CSV file by using `Save as comma-separated file`.

5.3.2. Demo window

- Many of the limitations of Experiment MFC can be solved by programming a graphical user interface (GUI) in Praat's Demo window (documentation here, but check also here).
  - You can draw stuff into the Demo window or include images and you can ask for input from the user as well.
  - As stated in the documentation, this window can only be accessed by scripts.

- A first minimal example could be the following code, in which we draw the waveform of a Sound object into the screen:

```
wav_id = Read from file: "mfc_files/04_pulpo.wav"
demo Draw: 0, 3, -1, 1, "yes", "curve"
```
(5.3)

- You can think of the Demo window as another canvas, where you'd draw things.
- Just as in Praat's Picture window, "*you can use `Select outer viewport...` and `Select inner viewport...`, if you know that the size of the Demo window is "100" horizontally and "100" vertically (rather than 12×12, as the Picture window)*" (taken from Praat's documentation).
- Getting **input from the user** is not particularly easy, but you can programme quite complex scripts, including adaptive tasks which display stimuli depending on the listener's previous responses to previous stimuli.
  - An example is provided below. Let's take a careful look and notice the usage of the function `demoWaitForInput` (it always returns 1, forever and ever).
  - This example has been taken from the documentation and it has been modified only by adding one comment:

```
label FIRST_SCREEN
demo Erase all
demo Black
demo Times
demo 24
demo Select inner viewport: 0, 100, 0, 100
demo Axes: 0, 100, 0, 100
demo Paint rectangle: "purple", 0, 100, 0, 100
demo Pink
demo Text: 50, "centre", 50, "half", "This is the first page"
while demoWaitForInput ( )
    if demoClicked ( )
       goto SECOND_SCREEN
    elsif demoKeyPressed ( )
       if demoKey$ ( ) = "→" or demoKey$ ( ) = " "
          # You could do something here to gather data!
          goto SECOND_SCREEN
       endif
    endif
endwhile
label SECOND_SCREEN
demo Erase all
demo Paint rectangle: "purple", 0, 100, 0, 100
demo Text: 50, "centre", 50, "half", "This is the second page"
while demoWaitForInput ( )
    if demoClicked ( )
       goto END
    elsif demoKeyPressed ( )
       if demoKey$ ( ) = "←"
          goto FIRST_SCREEN
       elsif demoKey$ ( ) = "→" or demoKey$ ( ) = " "
          goto END
       endif
    endif
endwhile
label END
```
(5.4)

  - Given that this is a script, you can modify any part of it to control the flow of

the script.
    - For example, you can send the responses of your user to a Table that you created beforehand (**let's do it!**).
    - Or you can play stimuli by opening stimuli and using `Play` (**let's do it!**).
    - If you store your participant's responses at each iteration of your experiment (in a Table, for example), you can analyse the responses each time and programme an adaptive procedure.
  - You can also get **click locations** (see documentation).
  - Or `Play` sounds while the script moves forward, by using `asynchronous`:

```
wav_id = Read from file: "04_asynchronous.wav"
Play
pauseScript: "Notice that the object is still there."
asynchronous Play
Remove
pauseScript: "Now we're playing the sound, but the object is gone."
```

(5.5)

  - Finally, you can combine the usage of the Demo window with that of the pause window (not the same than `pauseScript`), which can actually receive information from the user just as forms do (documentation here).

5.3.3. Open Sesame

- An excellent free (open source) alternative to Eprime exists, called **Open Sesame** (you can download this software from: http://osdoc.cogsci.nl/).
  - In essence, it is a graphical user interface that manages Python psycholinguistic packages.
  - You don't need to know programming to program very simple experiments, but you'll need to know some Python (not much!) to access all the capabilities of Open Sesame.
  - **Let's see an example** of an experiment. You'll find this example in your folder "companion_folder_session_5", in the file called "05_OpenSesame_experiment.osexp".

5.4. Scripting challenges

- **Time to have some fun 5.3**: Your final task for this section will be to use the vowel continuum that you programmed before as stimuli for an identification task, using Praat's Experiment MFC.
  - Prepare the experimental script and make sure it works well, that stimuli have been randomized and that the instructions to the participant are clear.
  - Make it so that each stimuli is repeated 3 times in total, and that there is a pause in the middle of the experiment.
  - Run the experiment and take it yourself, and then extract the results and save them into a CSV file.
  - Once you've done all that, show me your experimental file and your results.