Praat scripting II: perceptual experiments

Dr. Fernanda Barrientos

February 28, 2019

1 Intro

Now we will focus on some of the intricacies of perceptual experiments. I have spent a significant amount of time trying to figure what the best way to create perceptual stimuli is, but to date I have no definitive answer to that. It will largely depend on your purposes, and what type of things you're willing to sacrifice in order to get a response to your question.

1.1 What kind of perceptual experiment should I perform?

This is not a trivial matter: perception experiments can look into different kinds of phenomena. Here is a couple of things that you might want to take into account:

1. **Who are my subjects?** Are the subjects native speakers of the language that you're looking into, or are they nonnative? If you are thinking about performing a forced-choice task, make sure that the choices will be understandable to your subjects (unless you're looking for confusion).

2. **Am I looking for phonetic or phonological phenomena?** There is a significant body of literature about the link between a given perceptual task and the type of knowledge being elicited (e.g. Colantoni et al 2016). The most important aspects to take into account are:

   - Identification vs. discrimination: Identification (labeling) tasks are oriented to make the subject recover a very robust internal representation (e.g. a phoneme). Subjects listen to a certain sound and choose a label from a range of possibilities. On the other hand, a discrimination task may tap either phonetic or phonological information, depending on the ISI and the experimental design itself. Do we want our subject to tell whether two sounds are the same or different? Are the stimuli embedded? Are they natural? Do we want the subject to perform an odd-one-out task? etc.
   - Inter-stimuli interval (ISI): in a discrimination task, the ISI will determine how much of your short-term memory you'd like your subjects to use; generally speaking, it has been assumed that the longer the ISI, the more subjects will need to resort to long-term memory representations (i.e. phonological knowledge). Shorter ISI tasks are associated to retrieving phonetic phenomena.

3. **What am I willing to sacrifice?** There is always a trade-off between natural-sounding stimuli and control of the phonetic detail present in the stimuli. For instance, if we want to test perception along a continuum between two vowels, we will have to use synthetic stimuli, as asking a human speaker to elicit a 10-vowel continuum with exactly 0.1 Bark of distance between each token is impossible. However, synthetic stimuli are not particularly easy to create and thus they usually range between very robot-like and somewhat convincingly human. But again, this will highly depend on your research question.

2 Understanding what we are doing: source-filter theory

Praat makes use of the source-filter theory in order to create perceptual stimuli. In this model, we assume that human speech is given by a source (i.e. the vocal folds vibrating at a given frequency) and a filter that provides resonance to the source (i.e. the rest of the vocal tract), thus changing the quality of the sound. If you were able to listen to a set of vocal folds vibrating (that is, without being attached to a vocal tract), you would probably hear something like a raspberry noise.

Hence, and in order to create perceptual stimuli, we will then need to have a very clear question in mind. Let's say that we want to see where the perceptual boundary between Spanish /a/ and /e/ is; we will then ask native speakers of Spanish to (a) identify the sounds along the continuum; and (b) discriminate between acoustically adjacent stimuli. Thus, we will need a source (that is, a glottal pulse; we can either declare it in a script or take it from an existing sound) and a filter, i.e. a given formant frequency, along with certain specification regarding the range of the formant values, etc.

3 Resynthesis: Using existing sounds

First we are going to use an existing sound and play around with it. You can download my Spanish /a/ here (https://pandabar.github.io/vowela5.wav) and we will manipulate the formant frequency so that we end up with a 10-step /a-e/ continuum.

3.1 Preliminary tasks

Since we are working with a Spanish /a/ sound as the starting point and we want to create a continuum, we will have to tell Praat our desired formant values for the other endpoint. Here I simply recorded myself producing a cardinal /a/ and then a cardinal /e/, after which I just measured the formant values manually. Here you can try different things: for instance, you can ask a native speaker to produce vowels and then ask other native speakers to rate them; or you can use formant values reported elsewhere.

Vowel	F0	F1	F2	F3	F4
/a/	208	700	1450	2800	4200
/e/	215	450	2300	3000	4440

IMPORTANT: Bandwidths are not relevant when it comes to vowel recognition, but they make a difference regarding the quality of the sound. They are supposed to increase with formant frequency (i.e. F1 has a narrow bandwidth, and F5 the widest). Here I vaguely followed some values reported elsewhere, but they are by no means accurate.

I will then take either one of the vowels and manipulate it 10 times so I can get 10 different vowel sounds, each one of them equally spaced in Hz. I had recordings of both /a/ and /e/, but I picked /a/ simply because the quality of the recording was clearer. We will then start our script by defining my output folder and stating the number of Hz between each one of the /a/-/e/ continuum. For the latter, we can tell Praat to calculate this for us (see the last four lines):

```
#Define target folder (where my created sounds will land)
out_directory$ ="/continuum"
#Define continuum endopoints in Hz
#Giving /e/ values
f1_e = 450
f2_e = 2300
f3_e = 3000
f4_e = 4440
#Giving /a/ values
f1_a = 700
f2_a = 1450
f3_a = 2800
f4_a = 4200
#Now we calculate the distances between endpoints in Hz
dif_f1= f1_a-f1_e
dif_f2= f2_e-f2_a
dif_f3= f3_e-f3_a
dif_f4= f4_e-f4_a
```

In order to perform the resynthesis, we will have to carry out a few tasks in order to prepare the sound file. We need to resample it to 11,000 Hz (10,000 Hz for male voices) and scale the peak, in order to have a uniform intensity throughout the entire sound. So let's start with this small script:

```
#open file
vowel = Read from file: "vowela5.wav"
#Resampling
vowel_res = Resample... 11000 50
Scale peak... 0.99
```

3.2 Obtaining the source

Now that our sound is ready to be manipulated, we will extract the source from the sound. So, we will get the formant frequencies and then we will do an inverse filtering. That will leave us with the source. We will add the following lines to our script:

```
#Here we take the formant frequencies from the vowel
vowel_lpc = To LPC (autocorrelation): 16, 0.025, 0.005, 50
#and now we reverse-filter the sound
selectObject: vowel_res
plus vowel_lpc
Filter (inverse)
#And we rename it in order to make things clearer.
Rename... vowel_source
```

Now you can look at the Objects window. We will have the following objects: (a) the original vowel; (b) the resampled/rescaled vowel; (c) the formant values, obtained via LPC; and (d) the source.

3.3 Refiltering

The following for loop takes the source, then the formant values of the original vowel, and then these formant values will take the values for /a/, and will incrementally add or subtract the corresponding amount of Hertz until it reaches the other endpoint (/e/). In other words, the first line of `Formula (frequencies)` says: *for each number from 1 to 10, take the F1 of my original vowel, then add the difference between both F1s divided by 9, and multiply it by whatever number you are thinking of now (1 to 10) minus one. Leave the other formants alone.* Then the following line does the same for F2, then the next for F3, and then the last one for F4. The Bandwidth lines work in the same manner.

```
#refilter loop
for i from 1 to 10
#get the formants of my original vowel
selectObject: vowel_res
formant_id = To Formant (burg): 0, 5, 5500, 0.015, 50
selectObject: formant_id
#change the values of the original vowel to those for the corresponding step
Formula (frequencies): "if row=1 then self - ((dif_f1/9)*(i-1)) else self fi"
Formula (frequencies): "if row=2 then self + ((dif_f2/9)*(i-1)) else self fi"
Formula (frequencies): "if row=3 then self + ((dif_f3/9)*(i-1)) else self fi"
Formula (frequencies): "if row=4 then self + ((dif_f4/9)*(i-1)) else self fi"
#take the source and filter it through these new formant values
selectObject: "Sound vowel_source"
plus formant_id
Filter
selectObject: "Sound source_filt"
Rename... token'i'
#save as .wav file
Save as WAV file... 'out_directory$'token'i'.wav
select formant_id
Remove
endfor
```

After this, you should have 10 sound files in the folder that you previously defined as your target, which will be named: `token1.wav, token2.wav`, etc. You may want to play them in order to double check the end result. As you might see, it is not perfect, but this is why scripting pays off: you could also make all these changes manually, but it would take a lot of time. Since all the filtering messes with the real sound, the results are unlikely to be great. But here is a few things that seem to have worked for me:

- Bandwidths: give your formants at least some bandwidth.
- Use a schwa-like sound for the source, or at least something close to that. Avoid using rounded and/or high vowels.
- Make sure that the sound where you'll get the source from is excellent quality: do take the time to get into a recording booth and/or use a good quality microphone. You may even try removing some noise with Praat (this is achieved by going to *Filter → remove noise*) or using a program such as Audacity (which gives the best results, in my experience).
- Avoid creaky voices! They do not synthesize well.
- Mind the crossing: you will get better results when your vowels are spliced at a zero crossing (that is, when the oscillogram shows you a crossing of the sound wave with the zero dB line).

3.4 What if I want to create a Bark or Mel continuum?

Unfortunately, Praat would not let us do that very easily. You will need to convert the endpoint formant values into Bark or Mel[1] and then calculate the corresponding Bark/Mel steps. After that you will need to convert everything back to Hertz and feed your script with a Table of values.

How do we do that? Here (https://pandabar.github.io/hertztomelandhertzagain.praat) is a script that calculates everything for you within Praat (You can also use R packages that do this; for instance `seewave`). Due to time constraints we will not have time to break it down into detail, but feel free to ask me after the session. We will have to conform ourselves to just running the script and obtaining two .txt files, one with the Mel values[2] and another one with the corresponding formant values in Hz (we will use the latter for the resynthesis).

Here is a similar script that does that; you will notice that the first bit is exactly the same and that it's the for loop what changes.

```
#open file
vowel = Read from file: "vowela5.wav"
#Resampling
vowel_res = Resample... 11000 50
Scale peak... 0.99
#Here we take the formant frequencies from the vowel
vowel_lpc = To LPC (autocorrelation): 16, 0.025, 0.005, 50
#and now we reverse-filter the sound
selectObject: vowel_res
plus vowel_lpc
Filter (inverse)
#And we rename it in order to make things clearer.
Rename... vowel_source
```

So far, so good: no changes. But now we will need to tell Praat to open the .txt file that we just created with formant values in Hz that have already been converted from Mel.

[1]Mel or Bark? To be honest, I don't think there is too much of a difference. Both are perceptual scales and as long as you use either one of them instead of Hz when working with vowels you should be fine.

[2]Praat has a set of built-in functions for this: melToHertz(), hertzToMel(), hertzToBark(), and barkToHertz(). You just put the number inside the brackets and that's all -no formula fuss needed! Look into the script for more details.

```
#Reading table with values
myTable=Read Table from tab-separated file: "meltohz.txt"
nrow=Get number of rows
out_directory$ = "/media/fernanda/Desktop/Essex/continuum/"
#Main loop - Create resynthesized sounds from the Table values
for irow from 1 to nrow
f1= Get value... irow F1MeltoHz
f1bw = Get value... irow B1
f2= Get value... irow F2MeltoHz
f2bw= Get value... irow B2
f3= Get value... irow F3MeltoHz
f3bw= Get value... irow B3
f4= Get value... irow F4MeltoHz
f4bw= Get value... irow B4
selectObject: vowel_res
formants_'irow'= To Formant (burg): 0.005, 5, 5000, 0.025, 50
Formula (frequencies): "if row = 1 then f1 else self fi"
Formula (bandwidths): "if row = 1 then f1bw else self fi"
Formula (frequencies): "if row = 2 then f2 else self fi"
Formula (bandwidths): "if row = 2 then f2bw else self fi"
Formula (frequencies): "if row = 3 then f3 else self fi"
Formula (bandwidths): "if row = 3 then f3bw else self fi"
Formula (frequencies): "if row = 4 then f4 else self fi"
Formula (bandwidths): "if row = 4 then f4bw else self fi"
selectObject: "Sound vowel_source"
plusObject: formants_'irow'
Filter
Rename... token_'irow'
Save as WAV file... 'out_directory$'tabletoken_'irow'.wav
selectObject: myTable
endfor
```

4 Synthesis from scratch

You can also create a synthetic sound from scratch, that is, without the vowel sound from which we extracted the source. Again, there are several ways to achieve this. One of them (perhaps the simplest one) is to use the Praat Vowel Editor tool. The second option is to use the Praat Klatt grid. This is a bit more problematic as you will have to have a good idea of all the possible parameters that making a voice sound needs; (flutter, antiformants, amplitude, pitch contour, etc.) however, you can also make very minimal Praat Klatt grids that will give you an idea of a vowel sound. I have not mastered this myself, but here is a very minimal example just in case.

What did I do here? I simply used the same Table with values as in the section above, and fed it to the Klatt grid. As you can see, this (still minimal) example needs some further specifications. For instance, we need to specify how long we want this sound to be. The second digit on line 7 (1) refers to the number of seconds. So this is a 1 second long vowel.

Then, I have also added a pitch point: one single value on 0.5. Then I added more voicing amplitude points in order to make it sound louder in the middle and lower at the beginning and end. Then I added the formant frequencies and bandwidths. I just specified one point across time, but if we wanted to create, say, a diphthong, then we would need to create more formant frequency and bandwidth points. Finally, the `To Sound` command creates the actual sound.

```
#Open table object
myTable=Read Table from tab-separated file: "meltohz.txt"
nrow=Get number of rows
out_directory$ = "/klattcont"
#Main loop - Create Klattgrids from a list of values on a Table object
for irow from 1 to nrow
klatt=Create KlattGrid... kg_'irow' 0 1 5 1 1 6 1 1 1
selectObject: myTable
pitch= Get value... irow F0
f1= Get value... irow F1_Hz
f1bw = Get value... irow F1_Bw
f2= Get value... irow F2_Hz
f2bw= Get value... irow F2_Bw
f3= Get value... irow F3_Hz
f3bw= Get value... irow F3_Bw
selectObject: klatt
Add pitch point... 0.05 pitch
Add voicing amplitude point... 0.0 10
Add voicing amplitude point... 0.01 80
Add voicing amplitude point... 0.98 80
Add voicing amplitude point... 0.99 10
Add oral formant frequency point... 1 0.01 f1
Add oral formant bandwidth point... 1 0.01 f1bw
Add oral formant frequency point... 2 0.01 f2
Add oral formant bandwidth point... 2 0.01 f2bw
Add oral formant frequency point... 3 0.01 f3
Add oral formant bandwidth point... 3 0.01 f3bw
To Sound
Save as WAV file... 'out_directory$'/'irow'.wav
endfor
```

5 Creating stimuli with ISI

So far we have only isolated vowels, which is what we need if wanted to run an identification experiment. But what if we wanted to run a discrimination one, with two or three stimuli separated by a given ISI? We have several options, depending on the experimental setup (will you use an independent programme, such as PsychoPy, E-prime, an online platform such as LimeSurvey, or Praat itself?). The blanket option is to simply create a WAV file with the entire stimulus ready to go. Praat will let you create such stimuli by using the Concatenate function.

Let's say that we want to create stimuli that are immediately adjacent to each other in the acoustic continuum: an example of this would be one with `token1.wav` and `token2.wav` with a, say, 1-second ISI in between. The script is below.

Warning: this script will REMOVE ALL THE OBJECTS from the Object window!!! If there is anything that you would like to save (Sounds, TextGrids, Tables), do it now (but note that a script is NOT an Object).

```
#Warning: this script will REMOVE ALL THE OBJECTS from the Object window!!!
#PROCEED WITH CAUTION!!
select all
Remove

# Directories: the sound directory has only the vowels.
# Out directory is where concatenated vowels go.
sound_directory$ = "continuum/"
out_directory$ = "concatenated/"

# String list
Create Strings as file list... list 'sound_directory$'/*.wav
noFiles = Get number of strings
```

For the following for loop I had to define some string variables (those ending with the dollar sign). They are esentially telling Praat to (a) read the numbers on the file names (that is, we get rid of the .wav extension and tell it to take into account the number before it), and (b) to read these digits as actual numbers and not characters, so that then we can tell Praat to open the file that has that same number + 1.

```
# Main loop
for ifile to noFiles
select Strings list
filename$ = Get string... ifile
token$= replace$(filename$, ".wav", "", 0)
step$= right$(token$, 1)
first= number(step$)
following= first +1
```

I also added here an if jump: I am telling Praat to ignore `token10.wav` as there is no token11.wav to concatenate it to (however, `token10.wav` will be attached to `token9.wav`). Since I told Praat to look at the last digit of the file name, it will only read the 0 in `token10.wav`. So, I just asked to ignore any number if it's lower than 1. After doing this, I asked to create a 1 second silence and then read the following file.

```
if first >=1
appendInfoLine: "Opening token ", first,"..."
Read from file... 'sound_directory$'token'first'.wav
appendInfoLine: "Creating 1 s ISI..."
Create Sound from formula... silence 1 0 1 11000 0
appendInfoLine: "Opening token ", following,"..."
Read from file... 'sound_directory$'token'following'.wav
```

And now I just asked Praat to concatenate the first sound, the silence, and the following one, in that order. To do so, I told Praat to select everything on the Objects window except the Strings list and concatenate (this is why this script has to start with a blank Objects window).

```
#Concatenating sound A + silence + sound B
select all
minus Strings list
Concatenate
appendInfoLine: "Concatenating token ", first, " and token ", following,"..."
# Write
Save as WAV file... 'out_directory$'token'first'_token'following'.wav
appendInfoLine: "Saving token", first, "_token", following,".wav..."
# Remove before next iteration of for loop
select all
minus Strings list
Remove
endif
endfor
appendInfoLine: "all files concatenated, woohoo!"
```

Note as well that this script writes information to the Praat Info window. This can be done by using the `appendInfoLine` command. While this is not strictly necessary, it is a nice way to see what Praat is doing, and when: here I wanted to make sure that Praat was concatenating the right sounds. If you look at the `appendInfoLine` commands above, you will notice that strings (i.e. pieces of natural language) go in quotation marks; the defined variables do not. Furthermore, if you want to have both together, you need to separate them with a comma.

6 The experiment interface

6.1 The script

Now that we have our stimuli, we need to present them to the subject. Praat provides the Experiment option, where you create an Experiment object. This has a lot of advantages: since Praat can be run from a USB stick as a standalone and you run this experiment on Praat, you don't need to use the same computer nor an internet connection.

```
"ooTextFile"
"ExperimentMFC 6"
blankWhilePlaying? <no>
stimuliAreSounds? <yes>
stimulusFileNameHead = "concatenated/"
stimulusFileNameTail = ".wav"
stimulusCarrierBefore = ""
stimulusCarrierAfter = ""
stimulusInitialSilenceDuration = 0.5 seconds
stimulusMedialSilenceDuration = 0
stimulusFinalSilenceDuration = 0.5 seconds
numberOfDifferentStimuli = 9
"token1_token2"        ""
"token2_token3"        ""
"token3_token4"        ""
"token4_token5"        ""
"token5_token6"        ""
"token6_token7"        ""
"token7_token8"        ""
"token8_token9"        ""
"token9_token10"        ""
```

The commands more or less straightforward: the files that we will use are in the ”concatenated/” folder, and they all the .wav extension. Since we have no carriers (e.g. a .wav file with a voice saying ”this is the vowel...”) we left those blank. We told Praat to leave 0.5 seconds of silence before and after playing a stimulus, and that the total number of different stimuli is 9 (and then proceeded to list them without the .wav extension).

```
numberOfReplicationsPerStimulus = 10
breakAfterEvery = 30
randomize = <PermuteBalancedNoDoublets>
startText = "This is a perception experiment.
Listen carefully to the stimulus. You will hear two vowel-like sounds. Are
they the same or different? Click ’same’ if you think that these are two
identical sounds, or ’different’ whenever you think there is a small
difference. You may replay the sound only twice.
Click to begin the experiment."
runText = "Same or different?"
pauseText = "You may take a little break. Click when you’re ready to continue."
endText = "You have completed the experiment. Thank you!"
maximumNumberOfReplays = 2
replayButton = 0.8 0.9 0.8 0.9 "Replay" ""
okButton = 0 0 0 0 "" ""
oopsButton = 0 0 0 0 "" ""
responsesAreSounds? <no> "" "" "" "" 0 0 0
numberOfDifferentResponses = 2
0.25 0.45 0.5 0.6 "Same" 40 "" "same"
0.55 0.75 0.5 0.6 "Different" 40 "" "diff"
numberOfGoodnessCategories = 0
```

This second bit defines the number of times that we want to play the same stimulus: we have told Praat to play them 10 times each (therefore, every subject will give 90 discrimination judgements). There will be a break after 30 trials, which are all randomized.

Then we added the instructions text with the `startText` command. If you want a text permanently showing throughout each trial, use the `runText` command. If you would like to tell the subject that he/she has a break, then write something after the `pauseText` command; the text will pop up after each trial block. And finally, the `endText` command will allow you to enter a small goodbye text.

After that, we have given Praat a few more instructions. In this experiment, we are allowing the subject to replay the sounds. If we wanted them not to do so, then we simply change the digit after the `maximumNumberOfReplays` command to 0, and then we assign zeros to every coordinate after the `replayButton` command. The screen is divided as a grid with values between 0 and 1; therefore, if we wanted our Replay button to show up on the bottom right, we can enter something like: `replayButton = 0.8 0.9 0.8 0.9 "Replay" ""`. We operate in the same way if we wanted to have an OK and/or an "oops" button (this one will allow the subject to select a different answer during a given trial).

Finally, we are telling Praat that the responses are not sounds, so there is no need to add a "Record" button. Instead, we are stating that there are two possible responses, and then we give the coordinates for the "Same" and "Different" buttons. And since we are not asking them to rate the sounds, we have just filled in the `numberofGoodnessCategories` command with a zero.

However, if you have a different experimental design in mind, Praat gives you more options, which can be checked in the online manual.

6.2 How to run it

Now that the experiment is ready, we must run it. To do so, use the *Open>Read from file* option of the Praat menu: If you named your script `experiment.praat`, then you will see an object called `ExperimentMFC experiment`. Click *Run* and the experiment will begin.

After the subject is finished with the entire experiment, simply close the pop up window. You will be back to the Objects window. Click on *Extract results*; that will create a **ResultsMFC experiment** object. Here you must rename it with your subject's ID number. After that, click on *Collect to Table* and you will get a Table object with all the responses, reaction times included. Save as tab-separated or comma-separated file.

7 Some notes on experiment design

- Repeat the same stimulus to your subjects. This will allow you to include a measure of within-subject deviance.
- Whenever possible, use a Bark or Mel scale.
- Give your subjects as many breaks as possible. They shouldn't listen to more than 30 stimuli per block. It does not matter too much whether they will actually take the breaks or not, but the opportunity must be given.
- Don't feel forced to follow an established protocol; just make sure that the results of your experiment will actually shed light upon the question. This is perhaps the most difficult part, but thinking in terms of the kind of response that you want will make things easier.

```
appendInfoLine: "Happy scripting!"
```