TP2: Annotation / Peak detection / RT and IRI measurement

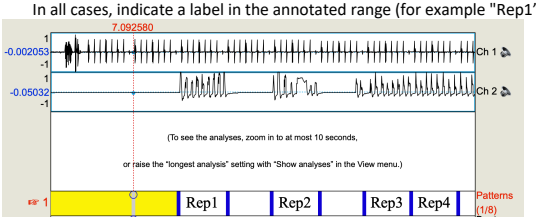
1- Annotations with Praat

- Download Praat from the website https://www.fon.hum.uva.nl/praat/

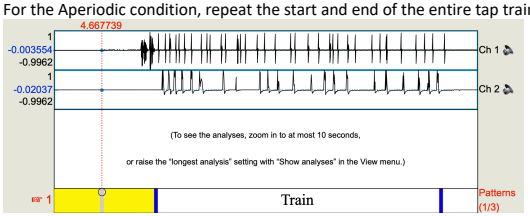
- Open a file « *-BaT.wav »
  The first line corresponds to the auditory stimuli (bips)
  The second line corresponds to the participant's taps
  To visualize the whole file: Ctrl+a
  To zoom/unzoom : Ctr+i / Ctrl+o
  To zoom of a portion of signal : select and Ctrl+n

- Create an annotation file (.Textgrid) with the same name, including an annotation line (Tier) labelled as « Cycles ».
  In the window « Praat object », select the audio file
  Annotate→ to Textgrid
  Field « all tier names » → « Cycles»
  Apply

  For the condition Periodic, identify the begining and end of each cycle of taps (and not of metronome bips), like on the example below.
  With Windows: Select the interval + (Ctrl+1)
  With Mac : place the cursor on the start time of the interval + click in the small circle at the top of the provisional border, in the annotation line (see example below). The same applies to the end time of the interval
  In all cases, indicate a label in the annotated range (for example "Rep1")



For the Aperiodic condition, repeat the start and end of the entire tap train, as in the example below:



- Save your annotated textgrid (File → Save Textgrid as Textfile) !!!

- Process all the "*-BaT.wav" files provided to your group in this way

2- Automatic peak detection with Python

- Like in TP1, read the audio file "*-BaT.wav" (using the function scipy.io.wavfile.read) that contains the metronome signal in the first column (Vector[:,0]) and the tapping signal in the second column (Vector[:,1])
- Read the annotation file (Textgrid) and retrieve the start and end time of each annotated cycle (or start and end of train for the Aperiodic condition)
  Packages "praatio" "parselmouth", "tgre" (!! pip install praatio)
  Then "from praatio import textgrid"

```
tg = textgrid.openTextgrid(tg_filename, False)
xmin, xmax, label = extractInfosFromTier(tg.getTier(tg.tierNames[0]))
```

avec :

```
def extractInfosFromTier (Tier):
        xmi=[]
        xma=[]
        lab=[]
        for start, end, label in Tier.entries:
                xmi.append(start)
                xma.append(end)
                lab.append(label)
 return xmi, xma, lab
```

- On each cycle, automatically detect the instants of the peaks corresponding to the metronome beeps, and the peaks corresponding to the taps of the participant.
  Fonction find_peaks from package scipy.signal
  tpeaks = find_peaks(AudioSignal[int(start*Fs): int(stop*Fs)],
  height=0.2,
  distance=0.5*Fs)[0]
  The "distance" argument corresponds to the approximate interval sought between 2 peaks. In our case, the tempo is supposed to be 120 BPM (every 500ms), so this distance between 2 peaks should be close to 0.5*fs samples (0.5s, fs=20000 Hz).
  The participant being not a robot, he will type slightly faster or slower than this tempo. So we can look for peaks spaced every 300ms, for example.
  The "height" argument corresponds to the amplitude of the peaks you are looking for, so that you ignore any other peaks below a certain noise level. I recommend you to normalize your signal (i.e. to do " Signal=Signal/max(Signal) ") before the peak detection step, and to look for peaks above 5% of the max amplitude of your signal (so specify " Height=0.05 ").

- Save the instants of the detected peaks in a data table of the type :

Sujet	Groupe	Condition	File	Train	BeatNb	BeatInstant	TapInstant
1	0	1	1	1	1	10.28781972	10.64731972
1	0	1	1	1	2	11.39961972	11.23766972
1	0	1	1	1	3	12.28781972	11.88041972
1	0	1	1	1	4	12.84156972	12.49921972
1	0	1	1	1	5	13.48006972	13.44131972
1	0	1	1	1	6	14.46369972	14.25801972
1	0	1	1	1	7	14.98581972	16.55971972
1	0	1	1	1	8	16.83541972	17.04981972
1	0	1	1	1	9	18.29666972	20.17096972
1	0	1	1	1	10	19.93836972	20.95411972
1	0	1	1	1	11	20.63541972	21.93781972
1	0	1	1	1	12	21.68096972	23.45071972
1	0	1	1	1	13	23.60671972	25.1851972
1	0	1	1	1	14	24.91586972	26.53536972
1	0	1	1	1	15	25.94356972	27.36011972
1	0	1	1	1	16	27.04271972	28.64746972
1	0	1	1	1	17	27.82016972	31.36251972
1	0	1	1	1	18	28.79866972	32.50791972
1	0	1	1	1	19	29.50016972	34.24776972
1	0	1	1	1	20	31.15136972	35.67201972

In a simple way, we can create a text file: dataFile = open(ResultsFilename, 'w')
Then write a first header line:
dataFile.write('Sujet\tGroupe\tCondition\tFile\tTrain\tBeatNb\tBeatInstant\tTapInstant\n')

Then write each line corresponding to the kth metronome bip and tap:

```
for k in range(0, min(len(tpeaks _bips), len(tpeaks_taps))) :
dataFile.write('%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n' %(sujet, label_group, label_condition, filename,
train_nber, k, tpeaks_bip[k], tpeaks_tap[k])
```

Finally close the file once all lines have been written : dataFile.close()

If you master the Pandas package, it is also possible (and more elegant) to store the results as a "dataframe" (import pandas as pd)

Create an 'empty' dataframe, with only the labels of each column:

```
Resultats=pd.DataFrame(data=None, columns=['Sujet', 'Groupe', 'Condition', 'File', 'Train', 'BeatNb',
'BeatInstant', 'TapInstant'])
```

Create a dataframe with a first row of data:

```
Data=[('S01', 'PQB', 'Periodic', 'S01_0034', 1, 1, 10.287919729678771, 10.647319729678772)]
Resultats=pd.DataFrame(Data, columns=['Sujet', 'Groupe', 'Condition', 'File', 'Train', 'BeatNb',
'BeatInstant', 'TapInstant'])
```

Add a kth row of data to an existing dataframe:

```
Resultats.loc[k-1]= ['S01', 'PQB', 'Periodic', 'S01_0034', 1, k, 11.399619729678772,
11.237669729678771]
Or
Resultats.append({'Sujet': 'S01',
                  'Groupe' : 'PQB',
                  'Condition': 'Periodic',
                  'File':'S01_0034',
                   'Train' : 1,
                   'BeatNb' : k,
                   'BeatInstant' : 11.399619729678772,
                   'TapInstant' : 11.237669729678771}, ignore_index=True)
```

Visualize the dataframe if needed: print(Resultats)

Save the dataframe as a .csv, .xls or .txt file

```
Resultats.to_csv("Filename.csv")
Resultats.to_excel("Filename.xls")
Resultats.to_csv(r'Filename.txt', header=None, index=None, sep=' ', mode='a')
```

- Save (again) the (same) detected peak times, this time by creating two new lines in an annotation file (Textgrid)

Create a new annotation file, from the one you created manually:

```
tg = textgrid.openTextgrid(tg_filename, False)
newTG = tg
```

Add to this new textgrid a new Point Tier annotation line, different from an IntervalTier annotation line:

```
NewTier = textgrid.PointTier('Label_of_New_Tier', , [], 0, 1.0)
newTG.addTier(NewTier)
print(newTG.tierNames)
```

Add <u>one</u> instant annotation in this PointTier

```
NewTier.insertEntry((t_instant, label), collisionMode='replace', collisionReportingMode='silence')
```

To add <u>several</u> instants (e.g., all the instants of the metronome beeps) in a PointTier, it is necessary to make a loop on the number of instants to add (for k in range (0, tpeaks_bips-1)).

Save this new automatically completed annotation file:

```
newTG.save(NewTextgridFilename, format= "long_textgrid", includeBlankSpaces= True)
```

Check that the script is working properly by re-opening the new textgrid with Praat

- Extend this peak detection, and save the detected instants, to all cycles of the same file

Loop on the number of cycles annotated in the first line of the textgrid

(NB : We could have detected the peaks on the whole file, without cutting into cycles. But in this case, we would detect a lot of metronome beeps that we are not interested in)

```
Cycles = tg.getTier[tg.tierNames[0]]
for start, stop, label in Cycles.entryList :
    … (previous routine on a cycle) …
```

- Extend this peak detection, and save the detected moments, to all files « *-BaT.wav » provided to your group

→ Manual processing, one after the other

→ or Automation by a loop on the different files contained in a directory

Package glob (import glob)

```
d=glob.glob(pathTextgrids + '*.TextGrid')
for n in range(len(d)):
    … (previous routine on a whole file) …
```

- Open the new textgrids created in Praat. Check the automatic annotation of the peaks and correct them manually if necessary (and remember to save your changes)

3- <u>Bonus : Creation of a data table for statistical analysis (in Python)</u>

- Read the textgrids (checked and corrected manually) and retrieve the instants of the metronome beeps and the participant taps.

- Save the instants of the detected peaks in a data table of the same type as before.

Sujet	Groupe	Condition	File	Train	BeatNb	BeatInstant	TapInstant
1	0	1	1	1	1	10.28791972	10.64731972
1	0	1	1	1	2	11.19961972	11.23766972
1	0	1	1	1	3	12.28791972	11.88041972
1	0	1	1	1	4	12.84156972	12.49921972
1	0	1	1	1	5	13.48006972	13.44131972
1	0	1	1	1	6	14.46306972	14.25801972
1	0	1	1	1	7	14.98581972	16.55971972
1	0	1	1	1	8	16.83561972	17.84981972
1	0	1	1	1	9	18.2966972	20.17096972
1	0	1	1	1	10	19.93836972	20.95411972
1	0	1	1	1	11	20.63541972	21.95781972
1	0	1	1	1	12	21.68096972	23.45071972
1	0	1	1	1	13	23.60671972	25.18351972
1	0	1	1	1	14	24.91586972	26.53536972
1	0	1	1	1	15	25.94156972	27.56011972
1	0	1	1	1	16	27.04271972	28.64746972
1	0	1	1	1	17	27.82016972	31.36251972
1	0	1	1	1	18	28.79866972	32.50791972
1	0	1	1	1	19	29.50016972	34.24776972
1	0	1	1	1	20	31.13336972	35.67201972

by adding two more columns of data:

- one for the dependent variable RT, corresponding to the difference between the two columns " TapInstant " and " BeatInstant ".
- one for the variable IRI, corresponding to the difference between BeatInstant(k)-BeatInstant(k-1) (Careful, IRI is not defined for all 1st taps (BeatNb=1) → Value " NaN " in the data table)

- Save the data table in a .csv, .xls or .txt file