

# Projet Synthèse de la Parole

Damien BIGUET

## Arborescence

`data/enregistrement.wav`: Enregistrement final des logatomes

`data/enregistrement.TextGrid`: Avec son TextGrid Praat

`synthese.py`: Implémentation en Python

`cli.py`: Interface en ligne de commande du script du projet

`output/result.wav`: Résultat final du script

`output/raw_concat.wav`: Résultat intermédiaire (concaténation brute, sans manip PSOLA).

`output/espeak-synth.{wav,TextGrid}`: Résultat intermédiaire annexe (synthèse eSpeak)

Le dossier `output` contient aussi des fichiers préfixés `s1_`, `s2_`, `s3_`, `s4_` qui correspondent aux résultats obtenus (avec les options par défaut) sur ma machine pour chacune des phrases disponibles.

## Pré-requis

Le script a été testé dans un environnement virtuel sous Python 3.12, il faut donc commencer par le mettre en place:

```
python3.12 -m venv .venv
source .venv/bin/activate
pip install -U -r requirements.txt
```

Les tests ont été effectués sous macOS, mais normalement la chose devrait être portable, même sous Windows.

## Utilisation

Plusieurs options sont disponibles pour configurer divers aspects de la synthèse, affectant soit la synthèse eSpeak, soit le fonctionnement du script en lui-même.

Ces options sont détaillées dans l'aide de l'interface ligne de commande:

```
./cli.py --help
```

Je reviendrais sur quelques une d'entre elles plus bas.

## Choix des phrases

Les phrases sont visibles au bas du message d'aide:

```
List of available sentences:
```

- 1: Ah bah maintenant, elle va marcher beaucoup moins bien forcément !
- 2: Je ne vous jette pas la pierre, Pierre, mais j'étais à deux doigts de m'agacer.
- 3: Barrez-vous, cons de mimes !
- 4: C'est une voiture de collection de prestige. Il y en a plus que trois qui roulent dans le monde et moi... J'ai la numéro 4.

Elles sont toutes tirées de répliques cultes de comédies françaises ;).

L'idée étant d'obtenir des phrases ayant un sens, et qui sont connues telles que prononcées/jouées d'une manière bien particulière, ce qui va rendre le contraste avec la prosodie de notre synthèse d'autant plus vive!

# Enregistrement des log atomes

Il me semble que l'on n'avait pas encore abordé eSpeak à l'époque, donc j'ai procédé par une transcription manuelle en API, puis la séparation en log atomes, le tout listé dans un fichier texte, que j'avais sous les yeux pendant l'enregistrement.

Par exemple:

‘Ah bah maintenant, elle va marcher beaucoup moins bien  
forcément !’

/abam̩tn̩əɛlvamaʁʃebokumw̩ɛbjɛfɔʁsemã/ IPA

\_a pa (juste pa?)  
ap ab ap /!\ à bien voiser le b  
pa ba pa  
ap am ap  
pa m̩ pa  
ap ēt ap  
pa tn ap  
pa n̩ pa  
p ðε pa  
ap εl ap  
pa lv ap  
pa va pa  
ap am ap (doublon)  
pa ma pa  
ap aʁ ap  
pa ʁʃ ap  
pa ſe pa  
ap eb ap  
pa bo pa  
ap ok ap  
pa ku pa  
ap um ap  
pa mw ap  
ap w̩ pa  
ap ēb ap  
pa bj ap  
pa j̩ pa  
ap ēf ap  
pa fɔ pa < o vs. o :(  
ap ɔʁ ap < ditto  
pa ʁs ap  
pa se pa  
ap em ap < missing ;'(  
pa m̩ pa  
apa (juste ap?) ã\_

### Logatomes

Déjà aux premiers essais d'enregistrement, je m'étais rendu compte qu'il allait falloir bien marquer quelques oppositions spécifiques (e.g., b/d, e/ɛ, o/ɔ), mais une fois confronté à eSpeak, il y a aussi eu quelques désaccords de transcription, souvent liées à des liaisons entre deux mots, qu'eSpeak ne fait pas puisqu'il marque une pause entre chaque mot (ce qui va en particulier mener à des désaccords sur le schwa, e.g., ø vs. θ).

Par exemple, j'avais quelques transcriptions avec des schwa que je n'ai quasiment pas marqués à la réalisation.

Bref, j'ai ré-enregistré quelques rustines en fin de fichier, et laissé quelques logatomes non-annotés quand ils étaient inutilisables (ou ré-annotés certains quand la réalisation trahissait la transcription).

J'ai aussi pu rencontrer quelques choix de transcription plutôt discutables d'eSpeak, par exemple, pour le mot "trois" dans la dernière phrase, qu'eSpeak transcrit /tʁa/ (j'ai envie de dire à l'anglaise?...) .

Bref, il aurait sûrement été plus simple de vérifier le rendu d'eSpeak *avant* l'enregistrement (que ce soit dans Praat, ou via `espeak-ng -v 'fr' -x` en ligne de commande).

L'enregistrement initial a été effectué avec un micro-casque sur un PC fixe, et les rustines via un autre micro-casque sur un PC portable (et avec un rhume, ce qui va exacerber les différences...).

Pour l'annotation, j'ai dégrossi la tâche grâce à la fonction `Annotate > To TextGrid (silences)` de Praat, qui, comme son nom l'indique, va générer des intervalles pour délimiter les silences. Cela m'a évité quelques clics (d'où les labels \*\*\* dans mon TextGrid).

## Fonctionnement du script

Par habitude, le code est commenté en anglais. Je vais ici revenir sur le déroulé général et expliciter certains des choix effectués.

Afin de valider les résultats pendant le travail sur le script (et aussi par habitude, je dois l'avouer), le script sort *beaucoup* d'infos sur le terminal!

Il sera donc peut-être moins déroutant de commencer par la phrase 3, qui est la plus courte (e.g., `./cli.py -s3`).

Toute la logique se trouve dans le fichier `synthese.py`:

La fonction `extract_diphone` s'occupe d'extraire un diphone spécifique de l'enregistrement, grâce aux annotations du TextGrid. Comme expliqué en cours, on

passe par des `call` manuels pour découper plus proprement sur des frontières de périodes (au croisements avec zéro quand l'intensité monte).

En plus de l'object sonore, on va aussi préparer et retourner des métadonnées à propos de ce diphone, et plus précisément sur ses deux phonèmes constituants, car on va travailler phonème par phonème pour les manipulations PSOLA, et non pas diphone par diphone, on va donc avoir besoin de pouvoir faire le lien entre ces deux représentations.

La fonction `espeak_sentence` va utiliser eSpeak pour faire la synthèse de la phrase demandée, selon nos réglages.

Tout d'abord, les silences en fin de phrase sont supprimés (pour simplifier la logique de la boucle).

Comme précédemment, on va collecter (grâce au TextGrid de la synthèse eSpeak) tout un tas de métadonnées à propos de ces *phonèmes*, afin de calculer leur durée et leur fréquence fondamentale.

En ce qui concerne la f0, on utilise par défaut la moyenne, phonème par phonème, mais l'option `--pitch-points=trio` permet de tester le résultat en prenant trois points par phonème, au début, milieu, et fin de celui-ci. Mais, même sur un phonème voisé, nous n'avons aucune garantie que la "ligne" de f0 soit continue, on peut donc tomber sur une valeur inexistante si l'on se contente d'une approche naïve. Cette option utilise donc la fonction `find_pitch_point` pour naviguer à l'intérieur du phonème pour ces trois points, afin d'obtenir réellement une valeur de f0 pour chacun de ces points.

La fonction `synthesize_sentence` va utiliser les deux fonctions précédentes pour réaliser la concaténation brutes des diphones nécessaires pour synthétiser notre phrase, et mettre à jour les métadonnées en conséquence.

En particulier, on va calculer la position de chaque *phonème* dans le flux concaténé, grâce aux métadonnées collectées. Cela implique un petit peu de gymnastique mentale et d'arithmétique, car *deux* diphones consécutifs contribuent à un seul phonème réalisé...

(C'est principalement pour valider cette étape que les nombreuses sorties ont été utilisées).

Avec l'option `--skip-word-gaps=false`, la fonction `insert_word_gaps` est utilisée pour honorer les silences en fin de mot, à la manière d'eSpeak. On opère phonème par phonème sur le flux concaténé (encore grâce aux métadonnées collectés, on avait accumulé les marqueurs de silence pour calculer une durée à appliquer *avant* le prochain phonème).

Cela implique une autre séance de gymnastique pour les métadonnées des phonèmes, vu que l'on doit décaler les *timestamps* de tous les phonèmes qui suivent le silence inséré (de la durée de ce même silence), afin de ne pas tout décaler. (Cette fonctionnalité est désactivée par défaut).

La fonction `manipulate_sound` va manipuler le flux concaténé, en opérant phonème par phonème, pour appliquer à chacun de nos phonèmes les propriétés de durée et de fréquence fondamentale des phonèmes réalisés par eSpeak (i.e., on calque la prosodie d'eSpeak sur nos phonèmes) via l'algorithme PSOLA.

Au même titre que l'option `--pitch-points=trio` mentionnée plus haut, j'ai aussi voulu expérimenter avec différentes manières d'appliquer la durée, via l'option `--duration-points [mid|edges|bracketed]`. Par défaut, comme vu en cours, un seul point de durée est inséré au milieu du phonème, et on laisse Praat interpoler entre les points; en mode `edges`, on isole le phonème entier entre deux points à ses extrémités; et en mode `bracketed`, on rajoute en plus du mode `edges` un cadre neutre pour limiter l'interpolation entre phonèmes.

La fonction `synthesize` enchaîne simplement chacune de ces opérations, dans l'ordre, en capturant au passage les résultats intermédiaires dans des fichiers.

## Validation des résultats

On va montrer ici que les calculs nous permettent bien de repérer précisément les *phonèmes* dans le flux concaténé, étape critique pour ne pas appliquer des manipulations PSOLA n'importe comment.

En prenant pour exemple `./cli.py -s1`, le *e* de “forcément”, qu'il est très facile d'isoler visuellement dans le spectrogramme:

Les informations affichées par le script:

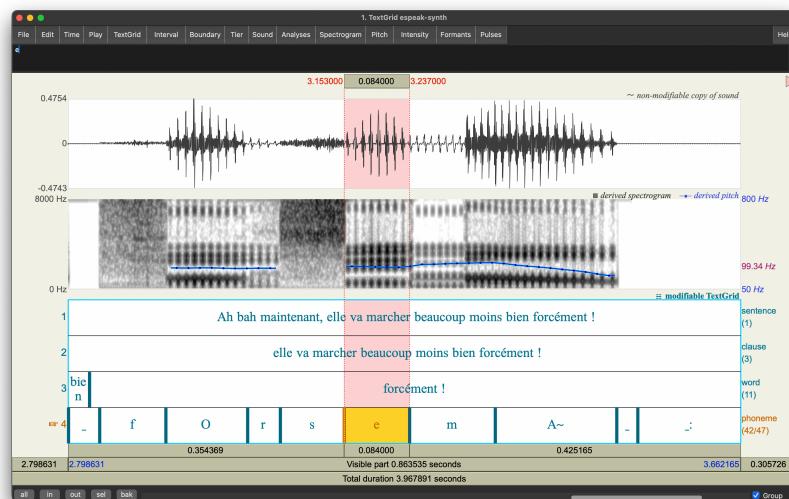
```

phoneme: r
concat duration: 0.12112139714058401 (3.287617928110164 -> 3.408739325250748)
target duration: 0.04200000000000026 (3.029 -> 3.071)
Adding a mean pitch point of 95.19056378626361 Hz @ midpoint 3.348178626680456
scaling midpoint @ 3.348178626680456 by 0.3467595403581038
phoneme: s
concat duration: 0.12451162408190442 (3.4087475458403875 -> 3.533259169922292)
target duration: 0.0819999999999985 (3.071 -> 3.153)
No pitch data
scaling midpoint @ 3.4710033578813397 by 0.6585730497424065
phoneme: e
concat duration: 0.06355974598418612 (3.5332523944101615 -> 3.5968121403943476)
target duration: 0.08400000000000007 (3.153 -> 3.237)
Adding a mean pitch point of 99.21413279648064 Hz @ midpoint 3.5650322674022545
scaling midpoint @ 3.5650322674022545 by 1.3215911847869808
phoneme: m
concat duration: 0.11376636439113952 (3.5968249641716024 -> 3.710591328562742)
target duration: 0.1099999999999988 (3.237 -> 3.347)
Adding a mean pitch point of 109.79836105969602 Hz @ midpoint 3.653708146367172
scaling midpoint @ 3.653708146367172 by 0.9668938669940218
phoneme: A-
concat duration: 0.22574932889668275 (3.7105823838629886 -> 3.9363317127596713)
target duration: 0.1560000000000014 (3.347 -> 3.503)
Adding a mean pitch point of 93.86697148415871 Hz @ midpoint 3.82345704831133
scaling midpoint @ 3.82345704831133 by 0.6910319546127893
phoneme: _

```

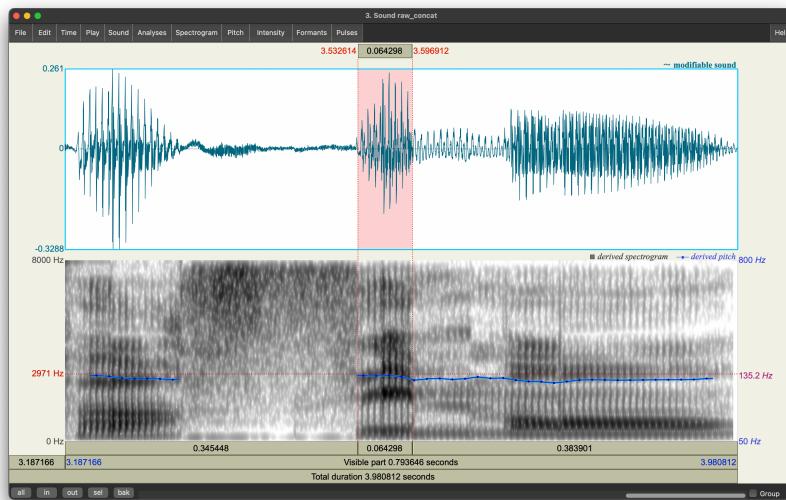
Sortie du script

Correspondent bien à la position de ce phonème dans eSpeak (target dans la sortie):



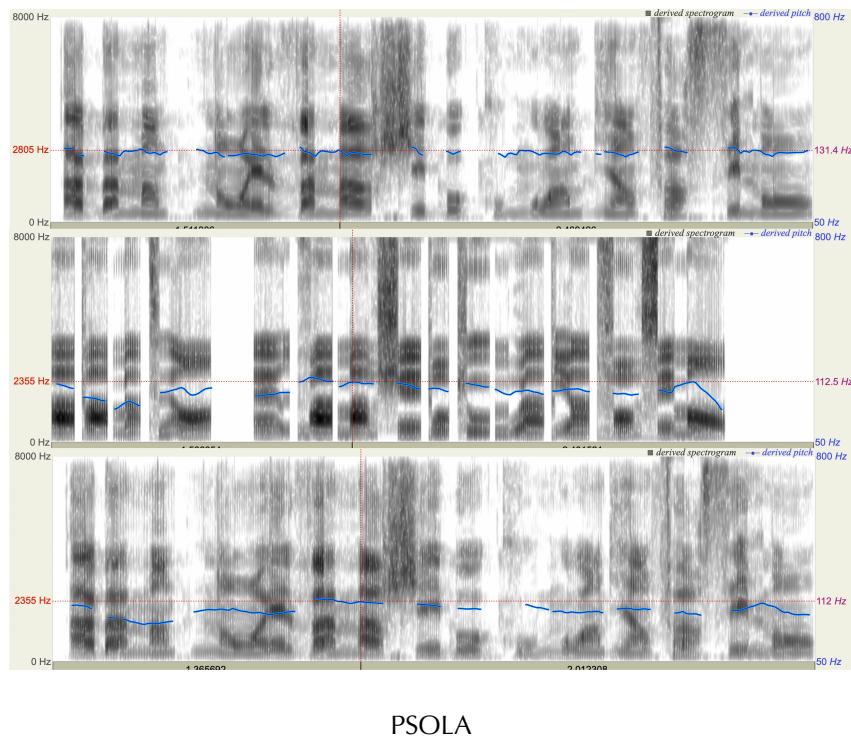
eSpeak

Et dans le flux concaténé avant PSOLA (concat dans la sortie):



Concat

De manière générale, si l'on aligne, de haut en bas: le flux concaténé brut, le flux eSpeak, et le flux final, on voit bien que la fréquence fondamentale a bien été calquée correctement:



## Bilan & Remarques

Grâce aux options implémentés, qui nous permettent de tester des réglages différents très rapidement, on peut remarquer que l'impact d'eSpeak sur le résultat final est plutôt massif (en particulier au niveau du choix de la voix, `-v`, `--voice`, ou de la vitesse de parole, `-w`, `--wpm`).

(En parlant de tests, l'option `-A`, `--autoplay` est très utile pour ces phases d'expérimentations).

Mais il ne faut tout de même pas oublier la fidélité de l'enregistrement des logatomes et de son annotation (je me rends compte que j'ai eu du mal à neutraliser mon timbre, du coup je pense que le rendu est un peu plus haut/fébrile que sur de la parole spontanée).

À titre personnel, les petits accrocs dûs à la transcription en Kirshenbaum m'ont fait très vite regretter de ne pas être resté en IPA du début à la fin...

J'ai aussi passé pas mal de temps à essayer de gérer les silences entre les mots (autrement que de les ignorer), d'où l'option `-G`, `--skip-word-gaps` (les premières itérations plus ou moins fructueuses tentaient de gérer ça pendant la concaténation, ce

qui rendait le suivi de la position des phonèmes plutôt sportif. C'est devenu beaucoup plus clair une fois géré sur une passe dédiée, phonème par phonème, sur le résultat concaténé).

Une idée (laissée en TODO dans le code) n'a pas été implémenté: au vu de certains doublons dans les logatomes enregistrés, je m'étais dis qu'il aurait pu être intéressant de constituer une banque de phonèmes, en parcourant l'intégralité de l'enregistrement, au lieu de faire les recherches à la demande (la méthodologie actuelle, qui implique que l'on choisit le premier logatome qui convient dans l'enregistrement). L'idée étant de pouvoir intégrer un degré de variabilité en choisissant une réalisation des doublons au hasard, et/ou de les choisir au plus proche des logatomes précédemment sélectionnés...

Un autre plan sur la comète aurait été d'utiliser autre chose qu'eSpeak, mais il fallait arriver à avoir un résultat *annoté*, ce qui est apparemment plus facile à dire qu'à faire (à moins de sauvegarder des synthèses de nos phrases et de les annoter manuellement)... Le projet [MAUS](#) propose un outil qui semble y arriver (au format TextGrid, en plus!), mais le modèle Français n'est pas open-source, ce qui rendait son utilisation en local impossible (un accès via un service web est cependant ouvert, mais je n'ai pas été plus loin à ce niveau, juste testé deux ou trois enregistrements pour voir). J'ai aussi vaguement jeté un œil à l'outil [ELAN](#), qui semble aussi exposer cette fonctionnalité (au vu des restrictions, via l'API REST j'imagine). Bref, j'ai préféré me concentrer sur eSpeak pour ne pas perdre trop de temps.

En ce qui concerne le style, je suis un peu rouillé en programmation objet en Python, du coup j'ai fait outre sur ce point, mais ça ne serait pas trop compliqué à adapter, et rendrait quelques signatures de fonctions plus propres (limitant en passant le recours à des variables globales).