## Augmentation de la base de données

Puisqu’en cherchant les données pour la classification des mots singuliers français on n’a rien trouvé, on a essayé de construire notre propre base. Ici car on est limité par le temps, on ne peut pas collecter des milliers d’enregistrements. Donc on a décide d’augmenter la base de données en ajoutant des transformations des audios originales.

On a trouvé un article [Audio Data Augmentation](https://medium.com/@alibugra/audio-data-augmentation-f26d716eee66) par Ali Buğra Kanburoğlu où les trois méthodes suivants sont mentionnés :

1. Ajout du bruit blanc

Cela va pas trop aider puisque les données vont être soit trop corrélées, soit on va perdre la qualité en ajoutant un bruit trop fort. En plus il y a un risque que le modèle va apprendre la présence du bruit comme un facteur pertinent pour la classification. Et encore il y a des problèmes pour les petites de phrases d’une amplitude beaucoup plus petite que celui de la reste (ex : le [t] dans « à droite » est souvent peu fort et risque d’être consomme par le bruit)

1. Décalage de l’audio

Cela n’a aucun intérêt car con coupe la silence au début et à la fin de la phrase.

1. Élongation de l’audio

Voici une idée intéressante, mais elle nécessite des améliorations. Car on normalise la durée d’audio aussi, cela n’est pas trop intéressant en soi. Par contre, on pourrait utiliser cet approche pour redistribuer la vitesse au long de l’audio en gardant la même durée.

Et alors, c’est comment on a décidé de créer un algorithme de redistribution de la vitesse du son.

## Formule de redistribution

On est parti à partir d’une idée simple que la durée de l’audio c’est sa vitesse multipliée par le temps :

Où, si on considère que la vitesse dépend du temps :

Normalement la vitesse est constante, , mais on va introduire une fonction de perturbation pour la modifier :

Alors, pour introduire les perturbations on a besoin de générer des fonctions intégrables pour lesquels on a :

Mais car l’audio n’est pas continu et on ne dispose que d’une fonction qui nous permet de modifier la durée d’un morceau d’audio par un facteur constant, on a besoin de discrétisation (dans la suite on va considérer l’intervalle sans perte de généralité puisque on peut toujours mettre la fonction à l’échelle nécessaire) :

d’où on a les coefficients d’échelle :

Ici on rencontre le premier problème. Il n’est pas évident de chercher des fonctions dont l’intégrale est égal à zéro. Heureusement, cela est facilement généralisable. Considérons une fonction intégrable. Pour avoir une fonction dont l’intégrale est nul il suffit d’enlever la valeur de l’intégral :

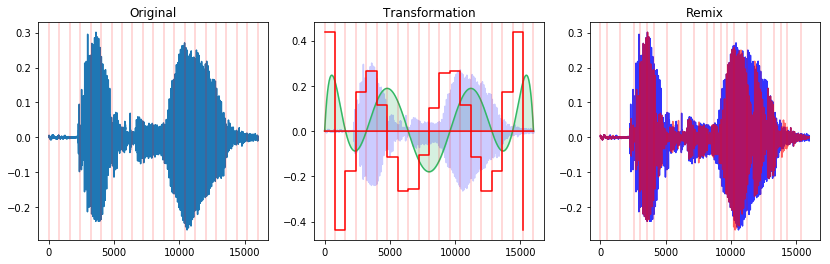
Adaptons l’index pour Python :

Le deuxième soucis est ce qu’on voulait éviter des intégrales dans le code Python. Alors, on va considérer qu’on connaît la primitive de  :

Alors par la formule de Newton-Leibnitz on a :

Donc on se retrouve avec un problème beaucoup moins compliquée : génération des fonctions dérivables quelconques. La solution la plus évidente est la génération des polynômes a partir de leurs racines en utilisant ***numpy***. Les racines des polynômes sont distribuées aléatoirement sur .

Et voici le résultat (la courbe verte est le polynôme et la courbe rouge représente les  :



Remarque : par symétrie quand on génère un polynôme on obtient quatre au prix de l’un puisqu’on peut également utiliser , et .