R0ANNE

Zeste de savoir

Comment fonctionne vraiment Auto-Tune ?

30 décembre 2020

Table des matières

Le célèbre plug-in qui modifie la voix est partout. Souvent confondu avec le vocoder, qui mélange la voix avec un instrument, l'autotune est un procédé qui se contente d'aligner les sons de la voix sur les notes les plus proches. Mais savez-vous vraiment comment il fonctionne?

> *i* Ce billet se concentre sur le plug-in Auto-Tune édité par la société Antares (qui est à l'origine de cette technique en particulier), afin de ne pas trop avoir à s'étaler sur d'autres procédés.

Les deux premières parties de ce billet vous rappellent les bases de la perception sonore, dans un langage fait pour être accessible, car je pense que c'est nécessaire pour bien comprendre les choix de conception associés. Vous pouvez les sauter si vous êtes vraiment à l'aise avec ça.

1. Anatomie d'un phonème

En linguistique, le phonème est la plus petite unité audible que l'on puisse distinguer d'un son parlé. Un mot est composé de divers phonèmes qui se suivent et qui s'enchaînent.

Un son, c'est un ensemble de vibrations dans l'air simultanées qui sont perçues comme une information unique par notre organisme.

Une seule vibration, dispose de deux grands attributs:

— L'**amplitude**: plus l'amplitude est élevée, plus le son est fort.
— La **fréquence**: plus la fréquence est élevée, plus le son est aigu.

L'amplitude et la fréquence sont deux grandeurs physiques mesurables. L'amplitude, c'est la force de la vibration alors que la fréquence, c'est la vitesse à laquelle les ondulations se succèdent.

Un son n'est pas fait d'une seule vibration, mais de plusieurs vibrations sur des fréquences liées entre elles.

Pour être plus précis, notre organisme dispose de méthodes très précises pour décider si plusieurs vibrations constituent un même son:

— Si deux vibrations sont émises sur des fréquences très proches, alors elles seront considérées comme un même son: c'est le phénomène du masquage fréquentiel , ou masquage simultané. À chaque fréquence est associée une «bande passante critique », c'est à dire l'espace entre celle-ci et une autre pour qu'elles puissent être considérées entre des sons distincts. Cet espace a été mesuré et a servi à la création d'outils mathématiques (comme l'échelle de Bark).

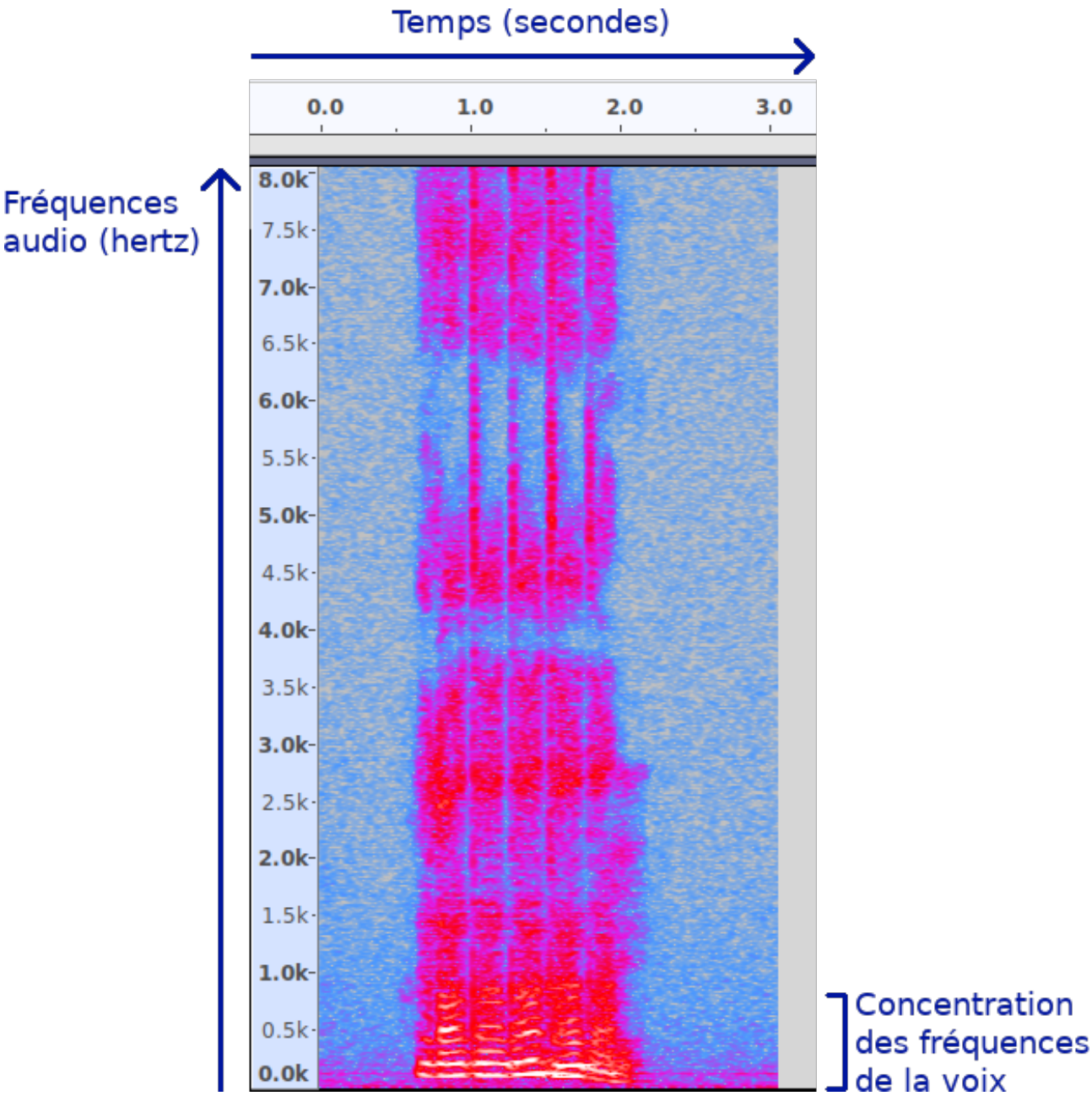
— Plus important encore, si **une fréquence est une multiple d'une autre fréquence**, alors **seule la note de la fréquence la plus basse («la fondamentale») sera entendue**. La **répartition de l'amplitude entre tous les multiples («les harmoniques») de cette note**, quant à elle, contribuera elle à former **la texture («l'enveloppe»)** du son que vous entendez.

? Un multiple ?! Mais, pourquoi est-ce que notre organisme cherche à faire des maths comme ça?

Il s'agit d'analyser un phénomène naturel qui s'appelle la résonance . Lorsque vous faîtes, par exemple, vibrer une corde, il va automatiquement y avoir une vibration associée à une fréquence plus basse (une grande ondulation), qui se décomposera en des fréquences plus hautes (des ondulations plus petites) suivant cette règle. Même chose lorsque l'air passe dans vos cordes vocales (vous n'avez pas besoin d'en savoir plus, après on entre dans la physique).

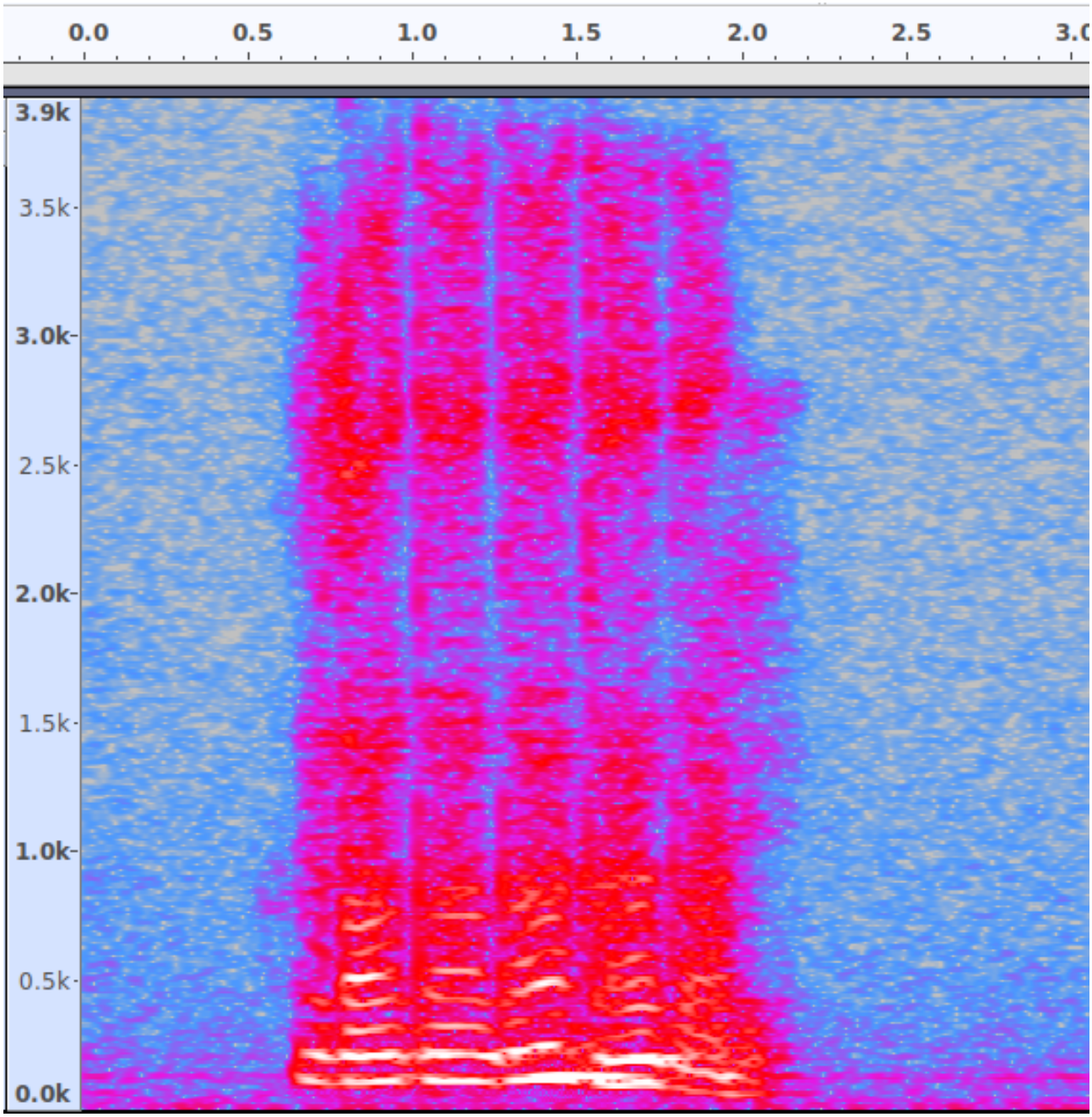
Vous savez maintenant de quoi est fait un son. Nous allons aller très vite et commencer à observer une représentation graphique et visuelle d'un morceau de voix. Voici un extrait:

Voici maintenant une représentation de cet extrait produite avec le logiciel Audacity:



Vous pouvez observer que nous avons sur l'axe horizontal les secondes, et sur l'axe vertical la fréquence (exprimée en hertz). Aux intersections, se trouve l'amplitude (la puissance du son, l'échelle de couleurs utilisée est gris-bleu-rouge-jaune). Cette représentation s'appelle un **spectrogramme** .

Ici, je montre les fréquences allant jusqu'à 8 KHz (8000 Hz, donc 8000 vibrations par secondes), mais les composantes audibles de la voix se trouvent essentiellement sous 2 KHz. Zoomons un peu.



Nous remarquons clairement plusieurs choses.

— Comme nous l'avons énoncé ci-dessus, ma voix est bien faîte de plusieurs vibrations séparées (individuellement il s'agit de **notes pures**). Ces vibrations se situent à des **fréquences multiples entre elles**, et donc deviennent **des harmoniques**.

— Toutes ces harmoniques ensembles forment **un son**, et **la manière dont l'amplitude des harmoniques varie au film du temps** forme **la texture du son, ou «l'enveloppe»**. Ici, il y a cinq enveloppes correspondant aux cinq notes que je prononce successivement (on le voit bien en regardant en haut du spectre).

— La première harmonique que vous entendez (tout en bas, autour de 100 Hz) s'appelle **la fondamentale**. C'est elle qui définit la note que vous entendrez (Do, Ré, Mi, Fa, Sol…). Les autres harmoniques ne servent qu'à former l'enveloppe.

— Parmi les harmoniques les plus basses, il y en a deux ou trois (en général la fondamentale plus deux autres) qui sont plus fortes que les autres. Ces harmoniques sur lesquelles l'accent sera mis vont déterminer la **voyelle** que vous allez entendre: on les appellent les **formants** . Elles sont définies par deux grands paramètres, le degré d'ouverture de la gorge et la position de la langue (en utilisant ces deux paramètres, on a formé un outil visuel appelé triangle vocalique qui permet de deviner quelle sera la voyelle produite, je vous le retranscris ci-dessous).

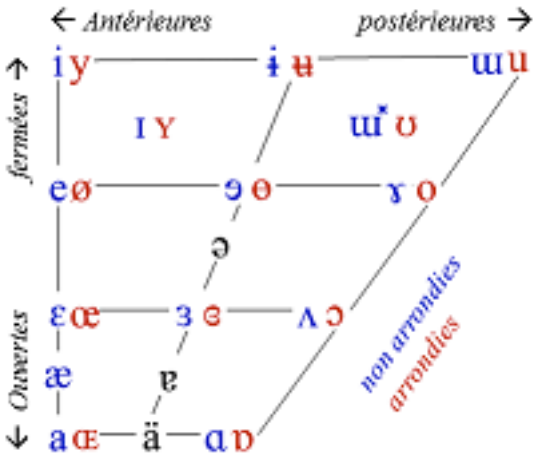


FIGURE 1.1. – Les classes de voyelles issues de l'article Wikipédia mis en lien (licence CC-BY-SA, auteur «UCAAS»). Ces symboles sont issus de l'alphabet international que vous avez peut-être appris à l'école primaire si votre instituteur était sérieux.

— La consonne, quant à elle, est définie par l'évolution de l'amplitude des harmoniques à travers le temps (l'enveloppe dont je vous ai parlé plus haut). Si l'amplitude croît tout de suite à partir de rien, c'est que l'attaque est forte (donc la consonne sera bien marquée), si elle croît lentement ce sera le contraire.

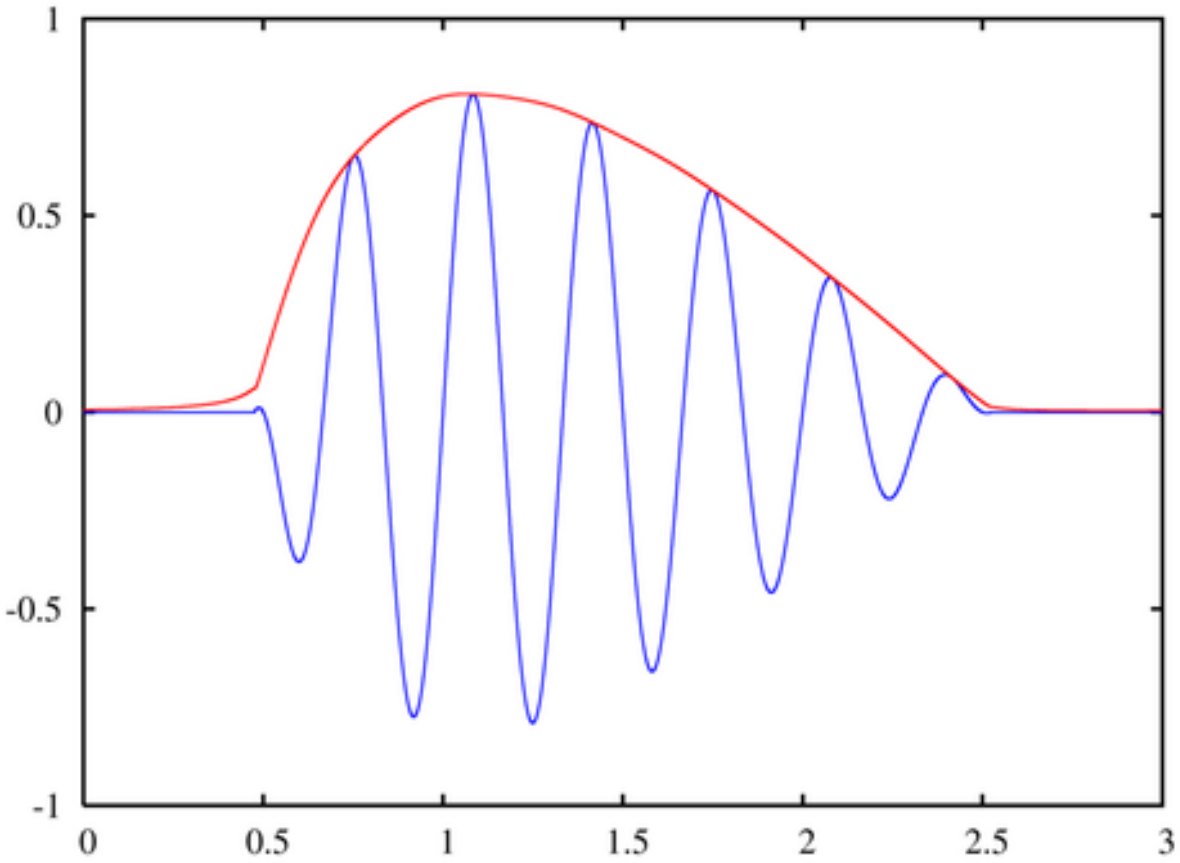
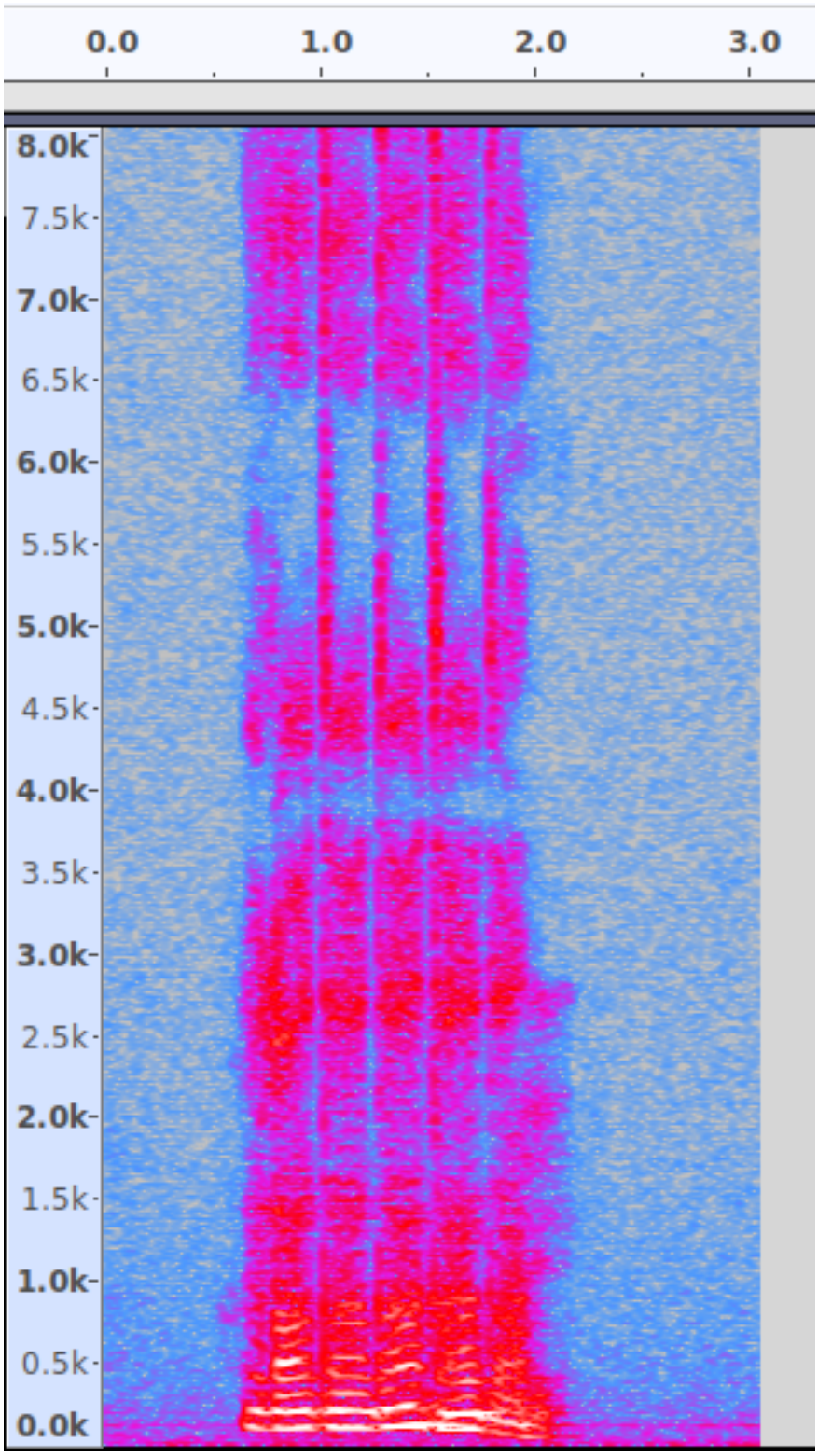


FIGURE 1.2. – L'enveloppe d'une note (licence CC-BY-SA Wikipédia, auteur «Omegatron»). Attention, jusqu'ici je vous montrais plusieurs vibrations en parallèle, mais ici je vous montre une seule note pure sous forme d'ondulation, avec l'amplitude sur l'axe vertical et le temps sur l'axe horizontal. Nous verrons les différences plus bas.

L'enveloppe combinée avec la gamme de fréquences que je peux reproduire constitue le timbre de ma voix.

2. Domaine fréquentiel et domaine temporel

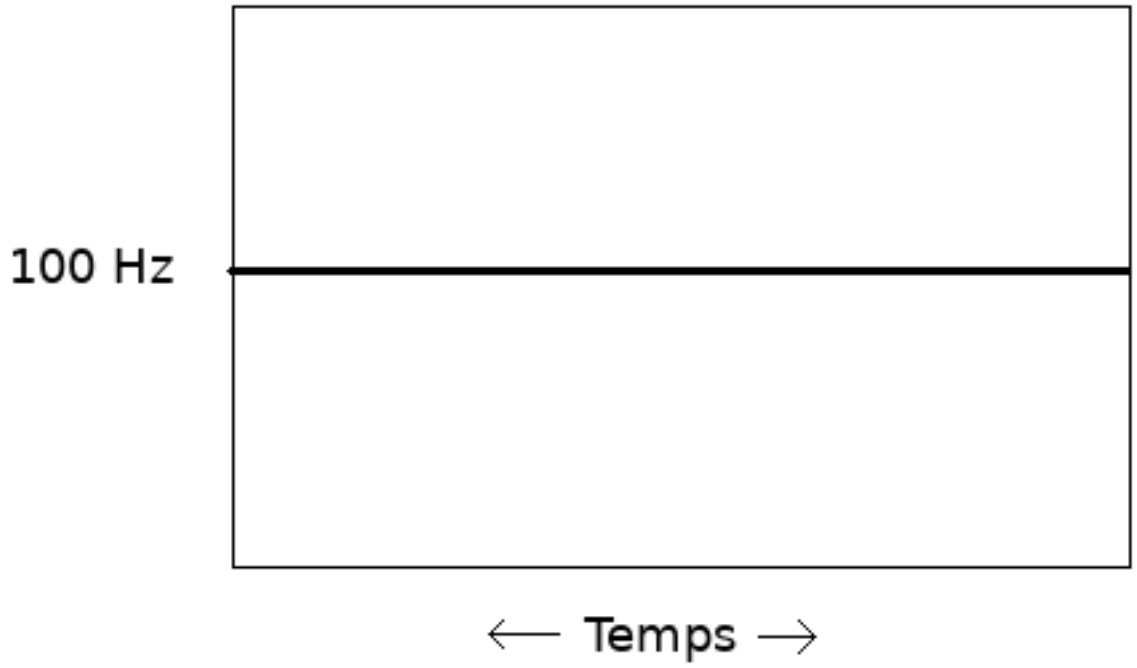
Reprenons l'image que j'ai montré ci-dessus:



Pour rappel, nous avons:

— La **fréquence (sur une longueur d'onde) sur l'axe vertical**
— Le **temps (sur le signal entier) sur l'axe horizontal**
— L'**amplitude aux intersections**.

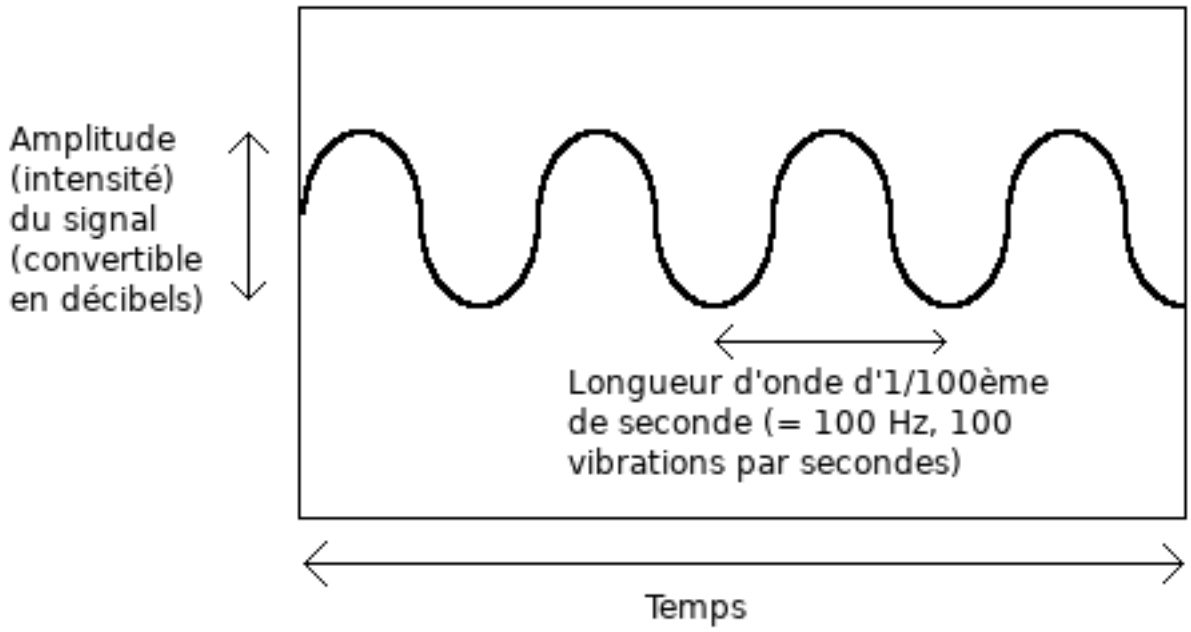
Si j'ai une seule vibration à fréquence constante, elle sera représentée par une ligne droite, puisqu'un de mes axes constitue la fréquence.



Mais une onde est d'abord une ondulation. Je devrais pouvoir la représenter sous forme de vague.

Voici la même vibration, mais représentée avec seulement:

— L'**amplitude sur l'axe vertical**, et
— Le **temps (sur le signal entier) sur l'axe horizontal**:



On dit que la première image se trouve dans le **domaine fréquentiel**, alors que la seconde se trouve dans le **domaine temporel** (les fréquences ne sont pas représentées). Un signal peut être modulé du domaine fréquentiel au domaine temporel, ou démodulé du domaine temporel au domaine fréquentiel.

Un signal sonore composé de plusieurs fréquences est fait de **plusieurs vagues de différentes fréquences** (donc qui forment des vibrations plus ou moins longues), à l'origine mélangées, que nous pouvons extraire.

Démonstration: voici un signal composé de plusieurs harmoniques, où chaque harmonique est un multiple d'une fréquence de base:

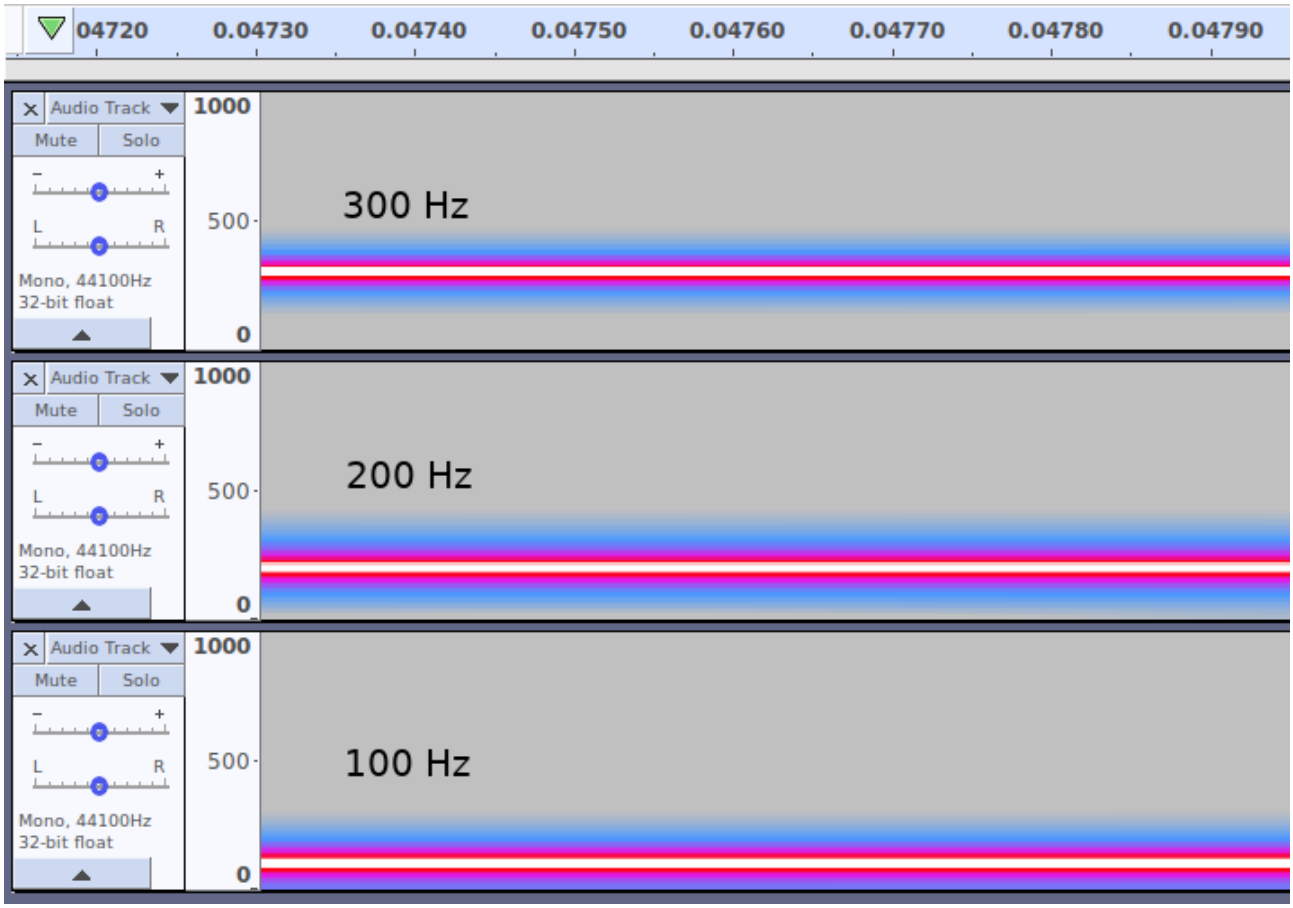


FIGURE 2.3. – Je génère trois notes pures (vibrations) à 100 Hz, 200 Hz, 300 Hz.

Voici ce qui se passe quand nos trois harmoniques sont modulées en forme d'ondes:

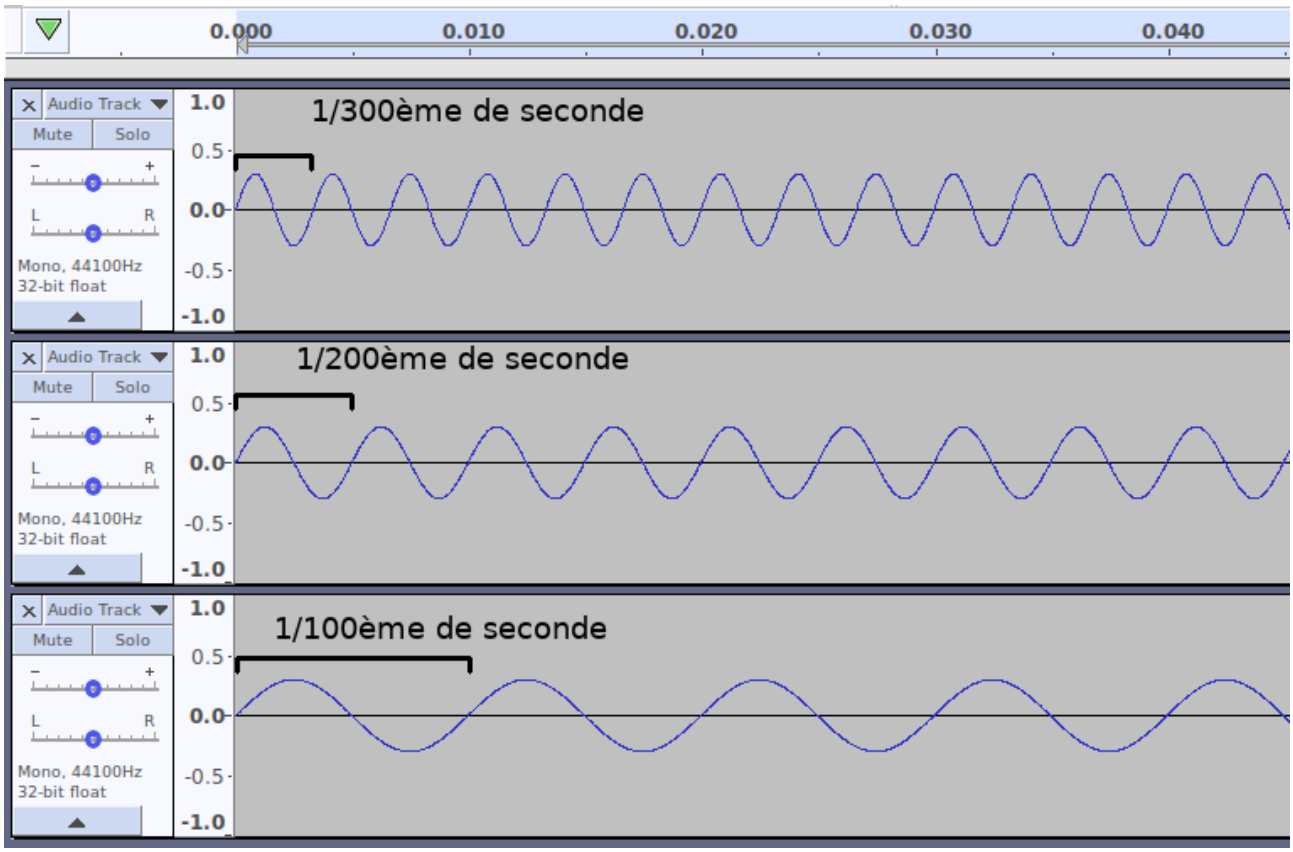


FIGURE 2.4. – La même chose, mais dans le domaine temporel.

Voici ce qui se passe quand on fusionne nos trois ondes pour former un seuil signal (cela se fait en… additionnant des valeurs):

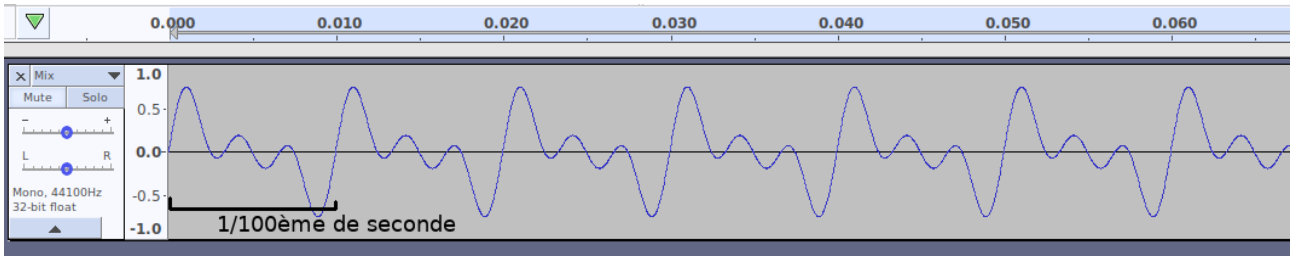


FIGURE 2.5. – Encore la même chose, mais en une seule fois.

Comme on peut le voir, notre signal présente un motif récurrent. On appelle ce motif récurrent la **forme d'onde**.

Attention. Ce motif pourrait prendre d'autres formes. Par exemple, si je décale un peu le début de la deuxième vague:

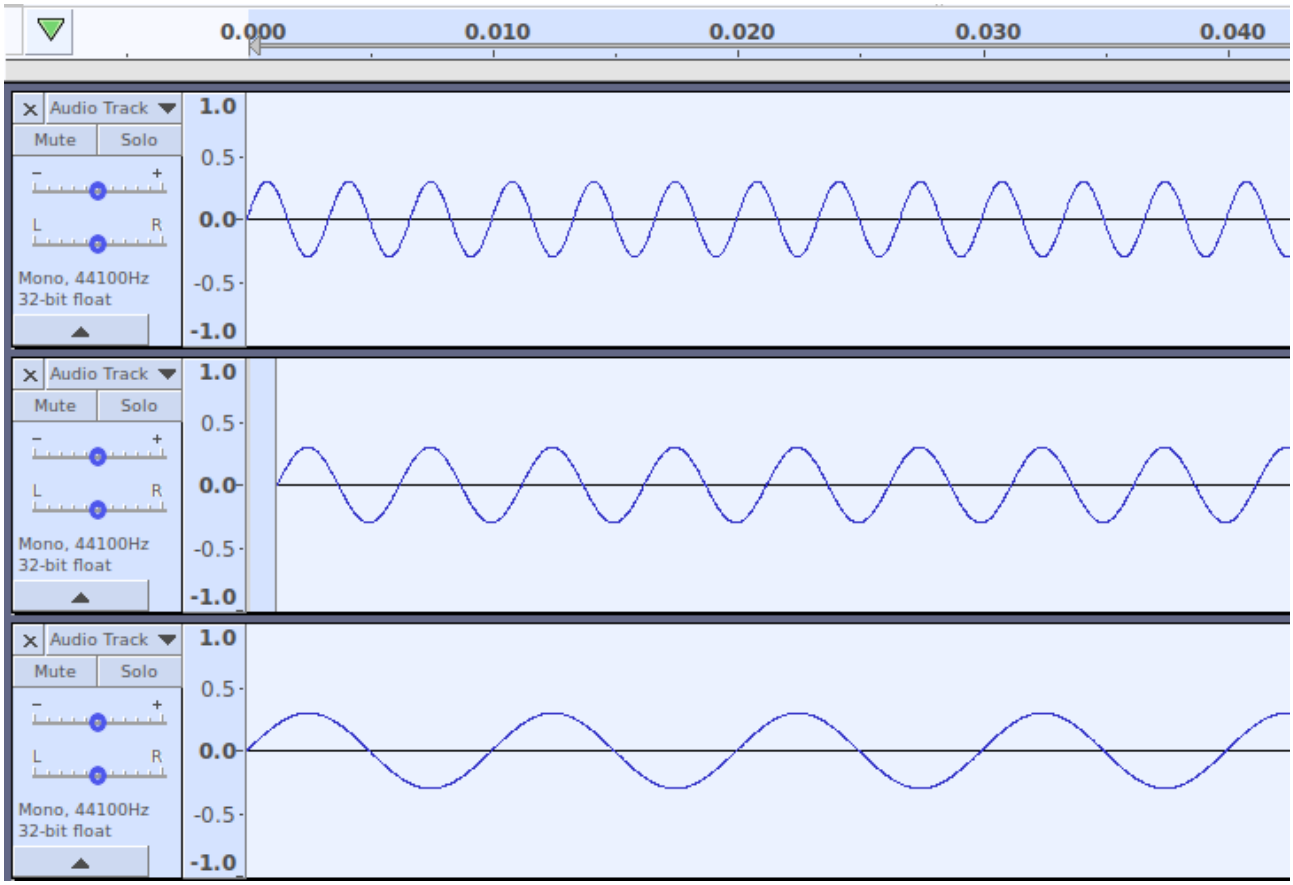


FIGURE 2.6. – Presque la même chose...

Eh bien cela donne un résultat complètement différent:

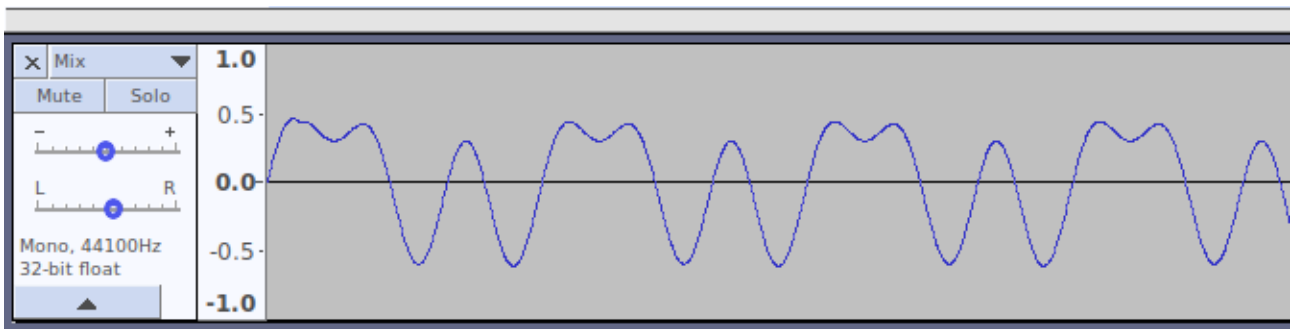


FIGURE 2.7. – ...mais le motif produit n'est plus le même

On appelle ce décalage d'une des vibrations composant le signal **la phase**. Ce signal est différent sur le papier, par contre, vous entendrez exactement la même chose.

Attention. Ce motif n'est récurrent que parce que les différentes fréquences impliquées sont multiples d'une même fréquence de base. La taille de la périodicité revient à la longueur d'onde de la fréquence de base. On dit que ce signal est **harmonique**.

Imaginons que je mélange (additionne) le son d'une voix (ce signal est harmonique) avec le son d'une chute d'eau dans une rivière (ce signal n'est pas du tout harmonique, plein de fréquences sans rapport qui se mélangent, c'est pour ça que le son d'une chute d'eau n'a pas de note

distincte, votre cerveau hésite sans cesse ente des notes différentes). Le signal ne sera pas du tout harmonique, il sera **inharmonique**.

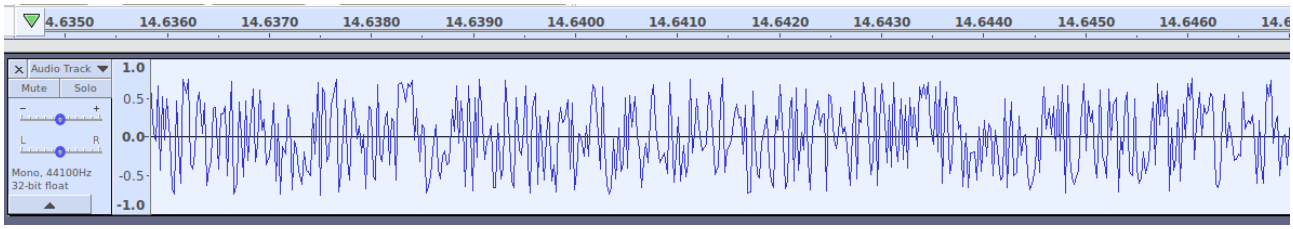


FIGURE 2.8. – Un signal inharmonique, du bruit blanc : pas de motif récurrent observable.

? Mais, comment est-ce que notre organisme fait la conversion entre domaine temporel et fréquentiel?

À un instant T, chacune de votre oreille reçoit un seul flux d'air. (La différence entre ces deux flux d'air vous permet de localiser un son dans l'espace.) Si nous enregistrions juste une l'intensité de ce flux d'air à travers le temps, alors nous obtiendrons une représentation en domaine temporel (c'est une peu ce que fait votre microphone).

Mais votre oreille interne est composée quelques milliers de petites cellules, associées à des structures organiques, appelées les cils de la cochlée (*hair cells*), de différentes tailles, qui auront pour fonction de vibrer en fonction des sons émis.

La taille de chacune de ces structures est légèrement différente. Ces différences de tailles feront, pour des raisons physiques, que les vibrations seront reçues différemment. De là, votre organisme va coder l'information induite par les vibrations reçues sur ce qu'on appelle le nerf auditif, et les envoyer au cerveau. Les fréquences y seront retranscrites individuellement.

Ces cellules sont en nombre limité, ce qui fait notamment que la plus petite différence perceptible entre deux fréquences est d'environ 1/230ème d'octave (une octave, c'est la différence entre la même note de deux gammes successives: **Do**, ré, mi, fa, sol, la, si, **Do**) dans les fréquences de la parole.

Le nerf auditif envoie ensuite les informations vers une région du cerveau précise, le cortex auditif. Certaines régions du cerveau humain fonctionnent encore un peu comme une boîte noire pour nous, mais on a des outils qui nous permettent de faire le constat que chaque fréquence entendue active un emplacement précis dans le cortex auditif. On appelle ce mécanisme tonotopie .

? Ok. Maintenant, tu vas nous parler d'Auto-Tune, qui est un programme informatique. Comment est-ce qu'un ordinateur fait la conversion entre le domaine fréquentiel et le domaine temporel?

Eh bien, pour ça, on a tendance le plus souvent à utiliser une fonction mathématique qui s'appelle **la transformation de Fourier**.

La transformation de Fourier d'origine s'applique sur des grandeurs physiques, qui ont potentiellement une résolution infinie (ou plutôt, nous n'en percevons pas les limites, sauf avec des outils

très avancés). En informatique, un son est forcément codé avec une quantité limitée de 0 et de 1, et donc une quantité finie de valeurs. C'est pour ça que la transformation de Fourier utilisée en informatique est la **transformation de Fourier discrète**, ou **DFT** (discrète = elle agit sur un nombre limité de valeurs).

Il existe une version optimisée de la DFT, qui utilise quelques astuces de programmation pour aller plus vite (réutiliser des valeurs, …), qui s'appelle la **transformation de Fourier rapide** (ou **FFT**). C'est la plus utilisée en informatique, elle est plus rapide mais elle produit la même chose que la DFT.

? Ok. C'est ça qu'Auto-Tune utilise du coup?

Non. Pour des raisons d'efficacité, **Auto-Tune fonctionne uniquement dans le domaine temporel**. Il traite des formes d'ondes (des vagues superposées entre elles), parce que c'est comme ça que votre ordinateur obtient du son depuis votre microphone et le fait sortir vers votre carte son (il se représente un flux d'air avec différentes intensités à travers le temps, du coup).

La transformation de Fourier, qui permet de produire un spectrogramme, comme je l'ai montré plus haut, est utilisée par de nombreuses applications. Par exemple:

— Shazam prend un spectrogramme, cherche les pics d'amplitude au sein de ce spectrogramme, et compare la distance temps/fréquence/amplitude entre ces pics, avec des distances entre pics connues pour permettre d'identifier des chansons, qu'il va aller chercher dans une base.
— Certains concurrents d'Auto-Tune utilisent la transformation de Fourier.

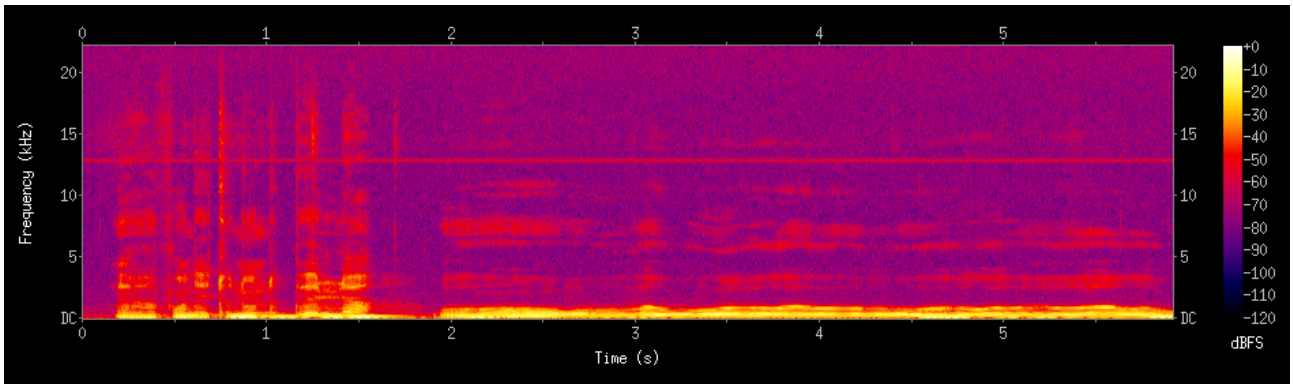
Néanmoins, Auto-Tune s'en passe.

Bien entendu, cela pose des contraintes. Pour fonctionner, Auto-Tune commence par essayer de deviner la fréquence de la fondamentale de la voix (la taille de la récurrence du signal). Si la voix est mélangée à quelque chose d'aussi ou de plus fort, ça ne marchera pas du tout, il n'y aura pas de récurrence à observer. C'est pour ça qu'Auto-Tune ne traite que la voix (ou assimilé) avec ses caractéristiques de signal harmonique associées, pas un morceau fini.

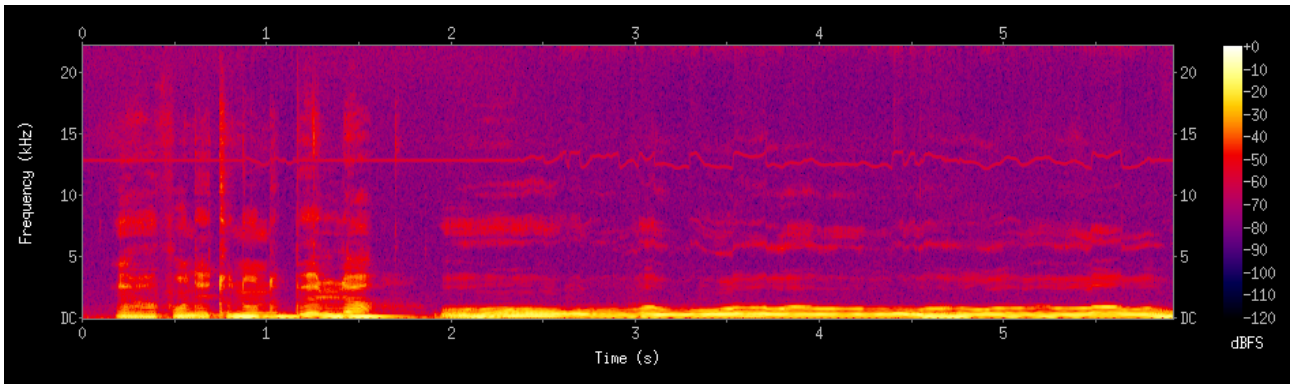
3. Observation du fonctionnement d'Auto-Tune

Passons à la partie la plus intéressante.

Je vais chanter dans un micro en faisant exprès de changer très vite d'une note à l'autre. Voici ce que ça donne:



Je vais maintenant y appliquer le plug-in Auto-Tune avec les réglages maximaux.



C'est étonnamment mélodieux. Pourquoi?

Ok, on parlait de son, maintenant on va parler de musique. Comme vous l'avez vu, notre oreille sait percevoir des fréquences distinctes avec une résolution allant jusqu'à environ 1/230ème d'octave dans les fréquences de la voix. Ce sont toutes les variations de la voix que vous entendez dans le 1er extrait (elle semble monter et descendre de manière fluide).

Dans la musique occidentale, une octave compte 12 demi-tons (Do, do#, ré, ré#, mi, fa, fa#, sol, sol#, la, la#, si). Les notes avec # sont les noires sur les touches du piano, les notes sans sont les blanches.

Il y a quelque chose qu'on n'a pas encore vu. On a dit que la fréquence de la fondamentale (l'harmonique la plus basse, dont les autres harmoniques sont des multiples) définissait la note que vous entendez. Mais comment exactement?

Eh bien **à chaque octave, la fréquence double**. La note Do1 (une note très grave) se situe environ à 65,4 Hz. Le Do2, le do d'au-dessus, c'est le double, 130,8 Hz. Le Do3, le do d'au dessus, c'est *encore* le double, 261,6 Hz (donc 4x la Do1). On dit que les octaves telles que vous les percevez évoluent de manière **logarithmique**.

i

Et du coup, comment je fais pour calculer Do1# qui vient juste après Do1? (la touche noire à droite.)

Eh bien, pour monter d'une octave, c'est très simple: il faut multiplier par 2, donc, pour monter d'un demi-ton, qui est 1/12ème d'octave, il faut multiplier par $2^{\frac{1}{12}}$ (soit environ 1.05946). Ça aussi ça augmente logarithmiquement. Fastoche, non?

Du coup:

— Do1 = 65.4 Hz
— Do1# = $65.4 \times (2^{\frac{1}{12}}) = 69.3$ Hz
— Ré1 = $65.4 \times (2^{\frac{1}{12}}) \times (2^{\frac{1}{12}}) = 73.4$ Hz
— Ré1# = $65.4 \times (2^{\frac{1}{12}}) \times (2^{\frac{1}{12}}) \times (2^{\frac{1}{12}}) = 77.8$ Hz
— Mi1 = $65.4 \times (2^{\frac{1}{12}}) \times (2^{\frac{1}{12}}) \times (2^{\frac{1}{12}}) \times (2^{\frac{1}{12}}) = 82.4$ Hz
— Vous avez remarqué, au lieu d'aligner la même multiplication comme ça, je pourrais faire des puissances.
— Bon, on s'ennuie un peu à passer d'un demi-ton à un autre, et si je passais directement à une autre octave?
— Mi2 = $82.4 \times 2 = 164.8$ Hz
— Mi3 = $164.8 \times 2 = 329.6$ Hz

i

Gardez cet article sous la main, c'est un mémo très utile pour faire correspondre fréquences fondamentales et notes: https://fr.wikipedia.org/wiki/Fréquences_des_touches_du_piano

Tout ça, ça veut dire que l'audition humaine mélange un semblant d'arithmétique *linéaire* (chaque harmonique est *un multiple* de la fondamentale) et *logarithmique* (chaque octave est *le double* de la précédente). Quel gros bazar. Y'a que l'évolution pour faire émerger un truc désordonné comme ça!

Votre perception de la puissance sonore aussi est logarithmique. Ou presque. Les courbes ne sont pas tout à fait droites. Et certaines fréquences sont entendues plus fortes que d'autres , un peu (les cris sont entends plus forts et vous préviennent du danger, pas mal non ?). C'est pour ça que le décibel (dB) est une échelle logarithmique, mais on en a créé des variantes qui fonctionnent en mesurant des bandes de fréquences distinctes et en pondérant la pression reçue par vos oreilles en fonction de la hauteur du son dans la bande concernée, comme le dB (A) qu'on utilise pour mesurer les dégâts du bruit environnemental (sur un chantier, etc.). Mais je m'égare.

Du coup, dans la musique occidentale, on a décidé de diviser une octave en 12, parce que deux-cent trente, c'est beaucoup trop proche pour discerner une note de l'autre.

Et dans ces 12 demi-tons, on a décidé de ne pas en utiliser *plus de 7 à la fois*, parce que si on se limite aux bons demi-tons, ben certaines notes sembleront être proches entre elles, parce qu'elles partageront presque des harmoniques. Une sélection de demi-tons qu'il faut utiliser, c'est une **gamme** (ou une **échelle**), et par exemple, gamme de Do majeur, c'est: Do, ré, mi, fa, sol, la, si*(, do)*. Les autres gammes peuvent utiliser des notes avec des dièses, etc.

Les savants occidentaux d'il y a quelques siècles ont employé plein d'énergie à classer les gammes selon leurs critères esthétiques perçus. Il y a les plus tristes, les plus joyeuses, les mineures et les majeures, etc. Mais dans d'autres cultures par exemple, ils s'en foutent, ils font de la musique avec tous les 12 demi-tons, ça s'appelle la musique chromatique .

Donc voilà, pour que ma voix ressemble à de la musique, j'ai décidé de garder le réglage par défaut d'Auto-Tune, qui est d'aligner ma voix sur la gamme de Do majeur, c'est à dire, Do, ré, mi, fa, sol, la, si, do (sur toutes les octaves qu'elle peut couvrir), afin que ça ait l'air d'être la musique.

FIGURE 3.9. – Comme vous le voyez, on peut régler la gamme, en haut à gauche. «*Major C*» c'est juste «Do majeur» (et si vous vous demandez pourquoi on dit Do alors que les anglais utilisent des lettres, eh bien ça vient d'un vieux chant latin qui commençait par «Ut, ré, mi, fa, sol, la, si», où «Ut» est devenu «Do». Et même à l'origine ça ne veut rien dire). Pas mal non?

Venons-en au fait. Auto-Tune est composé de trois parties:

— Un **détecteur de hauteur** (*pitch detector*) qui va essayer de deviner la fréquence de la fondamentale.

— Un module de **suivi de la hauteur** (*pitch tracking*) qui va continuer à faire de la détection de fondamentale, mais en essayant de prendre moins de ressources que le détecteur de hauteurs tant qu'on reste sur une note proche.

— Un **correcteur de hauteur** (*pitch corrector*), qui va, grosso modo, **corriger la hauteur de la note pour l'aligner sur la note de la gamme choisie la plus proche**.

Le détecteur de hauteur fonctionne en **estimant la largeur de la fondamentale** en utilisant **un algorithme d'autocorrélation** (on verra ce que c'est après), alors que le correcteur de hauteur va **plus ou moins redimensionner chaque forme d'onde pour ajuster la fondamentale sur la fréquence de la note la plus proche** (les autres harmoniques s'ajusteront en conséquence), en ajoutant ou supprimant les morceaux de notes manquants ou en trop pour que le son reste alors aussi long.

En plus de ces deux méthodes, Auto-Tune va proposer des solutions (activables ou désactivables) pour permettre, *si l'utilisateur le souhaite*, de robotiser davantage ou moins la voix (à l'origine, Auto-Tune avait pour objet de permettre de chanter juste et non de robotiser la voix):

— Ajouter un **vibrato**, c'est-à-dire une vibration minime (en-dessous de la plus basse fréquence audible) sur le timbre vocal, qui deviendra alors une variation perceptible et rendra la note de la voix moins droite (très légèrement).

— Rendre le changement de note engendré par le correcteur de hauteur **plus ou moins rapide**, afin que la transition soit moins abrupte.

Pourquoi est-ce que je vous précise les traductions anglaises? Eh bien parce que ce sont celles qui sont utilisées dans le brevet original d'Auto-Tune , qui est la plus grosse source utilisée pour la rédaction de ce qui va suivre.

FIGURE 3.10. – Illustration détaillée des options et autres indicateurs présents sur l'interface d'Auto-Tune. À noter qu'il existe d'autres versions du plug-in qui proposent davantage d'options et d'effets.

4. Anatomie d'un brevet

Le brevet d'Auto-Tune (je vous conseille la version PDF que je trouve plus lisible) est rempli le 15 octobre 1998 et a expiré le 14 octobre 2018. Auto-Tune a été conçu à l'origine par un ingénieur de l'industrie pétrolière qui a trouvé amusante l'idée de créer un algorithme qui permettrait de chanter juste.

Ce brevet est très riche en détails utiles puisqu'il fournir un algorithme complet pour l'implémentation d'Auto-Tune, sur une machine de l'époque, tout en flowchart et avec une explication en anglais très détaillé mais sans trop de jargon.

L'algorithme a sans doute un peu changé par rapport aux implémentations actuelles mais j'ai pu constater que certaines conditions logiques subsistaient (par exemple, celle d'effectuer le suivi de la hauteur tous les 5 nouveaux points d'amplitude à traiter) dans les versions les plus récentes.

Dans le brevet, Auto-Tune est décrit comme un appareil électronique avec un microprocesseur. S'il est aujourd'hui majoritairement utilisé comme un plug-in pour DAW (*Digital Audio Workstation* – les logiciels qui permettent de faire de la musique), il en existe aussi des versions matérielles qui se branchent sur de l'appareillage électronique classique, par exemple pour les performances de concert (il en existe même des versions qui s'insèrent dans des étagères à matériel – «rack» – de dimensions standards).

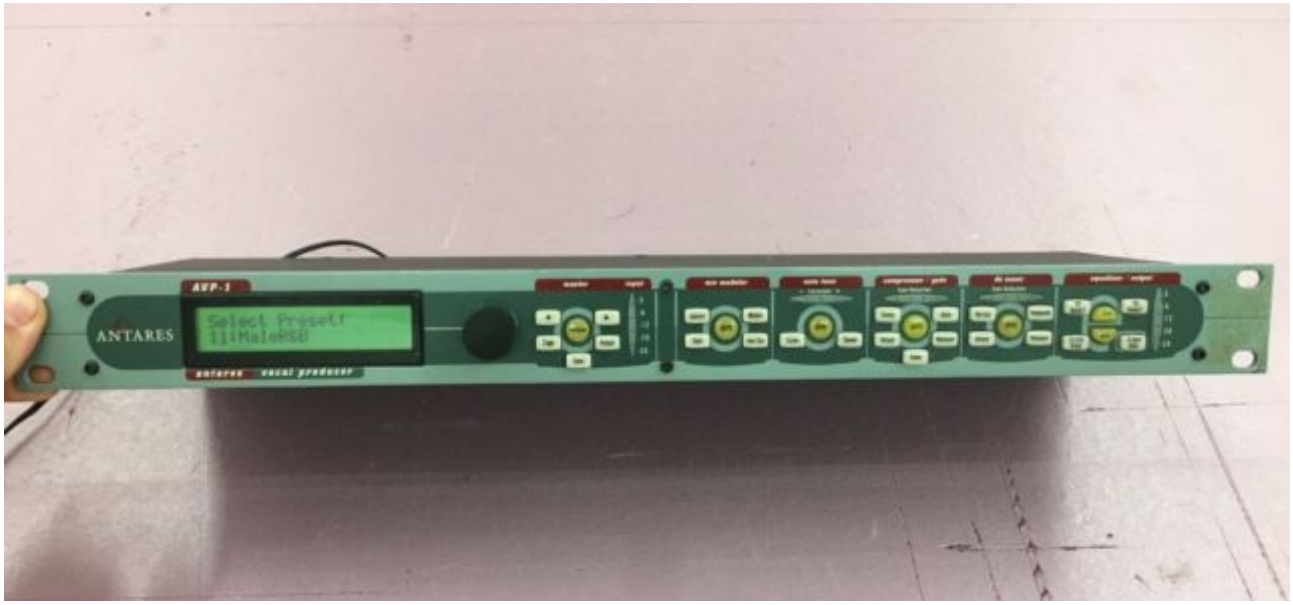


FIGURE 4.11. – Un Auto-Tune matériel ! Si vous faîtes de l'audiovisuel, vous pourrez le ranger à côté de votre encodeur vidéo et de vos onduleurs. (photo produit)

4.1. La détection de la hauteur

Du coup, on y apprend que la grande méthode d'Auto-Tune pour trouver la longueur de la fondamentale dans le signal du domaine temporel, c'est **l'autocorrélation**.

L'autocorrélation est une méthode mathématique très simple qui consiste à multiplier chaque valeur d'un signal (nos points d'amplitudes) avec des version décalées d'elle-même, en testant tous les décalages possibles.

Ensuite, on fait la somme de ces valeurs multipliées. Lorsqu'on a trouvé le décalage qui s'avère être la longueur d'onde de la fondamentale, eh bien **la somme finale sera très élevée, car on aura multiplié deux signaux quasi identiques entre eux**. Du coup, on garde ce décalage et on considère que c'est probablement notre longueur d'onde.

> C'est une méthode qui peut être appliquée en audio comme ici, mais pas seulement: en statistiques, elle peut servir à trouver les motifs récurrents dans un histogramme, par exemple.

Pour utiliser l'autocorrélation, ou tout autre méthode s'appliquant à de l'audio numérique, il nous faut d'abord notre signal du domaine temporel, qui sera donc composé d'une quantité finie d'amplitudes (qui se suivent dans le temps). Chacune de ces amplitudes est un nombre.

La norme, c'est de coder dans un signal **44100 amplitudes par seconde**, on dit alors qu'il a une fréquence d'échantillonnage (*sampling frequency*) de 44100 Hz. C'est la configuration la plus courante, on l'appelle aussi «qualité CD». Il faut au moins deux amplitudes pour faire une onde complète (sinon, vous ne pouvez pas voir la vague formée par l'onde monter et descendre): notre signal échantillonné à 44100 Hz pourra donc retranscrire des fréquences jusqu'à 22050 Hz (à deux points d'amplitude par seconde).

> Pourquoi est-ce que les CD codent 44100 points d'amplitudes par seconde? D'où vient ce nombre?

Votre oreille peut entendre des fréquences situées environ entre 20 Hz et 20000 Hz (les limites exactes dépendent notamment de votre âge, la limite sera plus autour de 16000 Hz pour un adulte). Le choix de ce nombre exact de 44100 est lié à d'obscures histoires de taille du signal lorsqu'il est émis conjointement avec des différents formats vidéo (bref, ne vous prenez pas la tête avec).

> Ok, du coup, j'ai récupéré mon signal avec 44100 amplitudes par seconde. Comment est-ce que je calcule mon autocorrélation?

Eh bien, dans le brevet d'origine, la première étape, pour détecter une première fois la hauteur de la fondamentale, ce n'est pas de travailler directement sur un signal à 44100 Hz. C'est de travailler sur un signal *8 fois plus dense*, donc à $44100 \div 8 = 5512.5$ Hz, ce qui fait que la plus haute fréquence qui pourra être retranscrite sera de $5512.5 \div 2 = 2756.25$ Hz. On ne garde qu'un point d'amplitude (*«sample»*) sur 8 dans l'entrée: on a volontairement perdu en qualité, on dit qu'on a *downsamplé* le signal.

Pourquoi on fait ça? Eh bien, **pour aller plus vite, pour avoir moins de données à traiter**. Et parce que les fondamentales de la voix humaine sont largement situées sous 2,7 KHz (celle d'un homme se situe en moyenne entre 85 et 180 Hz et celle d'une femme entre 165 et 255 Hz – le brevet mentionne que celle d'une voix soprano peut atteindre 1046 Hz). Bien entendu, on garde la résolution nécessaire pour distinguer des notes individuelles, histoire que notre signal ressemble à quelque chose à ce stade. Nous travaillerons sur le signal en qualité d'origine plus tard.

Du coup, le brevet (pages 12 et 13 du fichier PDF) nous explique que l'algorithme commence par calculer deux suites de valeurs (il nomme L les décalages en samples à essayer, et x les samples du signal d'origine, à évaluer en partant de la fin du signal):

— Pour chaque décalage à essayer, une somme des carrés des valeurs (sur deux fois la taille de ce décalage en sample): $E(L) = \sum_{j=0}^{2L} x_j x_j$

— Pour chaque décalage à essayer, une somme de la multiplication de chaque valeur avec sa valeur dans la série décalée: $H(L) = \sum_{j=0}^{L} x_j x_{j-L}$

En essayant tous les décalages en samples de 2 à 110. Ça lui donnera deux tableaux, la première donnée du tableau permettra de tester l'autocorrélation à 2756,25 Hz (puisque $(44100 \div 8) \div 2 = 2756.25$ Hz) jusqu'à la dernière qui lui permettra de tester l'autocorrélation à 50,1 Hz (puisque $(44100 \div 8) \div 110 = 50.1$ Hz).

Pour un décalage donné, le brevet part du principe que $E(L)$ sera toujours supérieur ou égal à $2H(L)$. **Mais que si le signal décalé est quasiment le même, alors l'écart entre $E(L)$ et $2H(L)$ sera *très* faible.**

Du coup, **pour décider si un décalage correspond bien à un motif périodique, Auto-Tune va prendre une très petite valeur** (que le brevet nomme *eps*) **et vérifier si l'écart se situe bien en-dessous de** $E(L) - 2H(L) \leq eps \times E(L)$. Si c'est le cas, on a trouvé notre fondamentale – ou presque.

? Et si je n'ai pas encore commencé à chanter, s'il n'y a pas encore de motif périodique à détecter ou de son à corriger, et donc qu'il n'y a pas de fondamentale à trouver? Est-ce que le logiciel va refaire le calcul à chaque fois?

Non. Pour rappel, nous travaillons sur les derniers samples du signal reçu (à ce stade, les 2×2 à 2×110 derniers samples du signal selon la valeur de L). À chaque fois qu'un nouveau sample (downsamplé) arrive, on dit qu'il y en a un qui *sort de la fenêtre* et un autre qui *arrive dans la fenêtre* en même temps.

Au prochain sample reçu, Auto-Tune va se rappeler de chacune des 109 valeurs calculées pour E et pour H, et simplement y soustraire le produit associé au premier sample de la fenêtre (celui qui ne nous intéresse plus) et ajouter celui associé au nouveau (qui nous intéresse). Cela nous permet de limiter le nombre d'opérations.

? Et est-ce que ce calcul est vraiment fiable? Il ne peut pas se tromper et prendre la première harmonique au-dessus de la fondamentale pour la fondamentale, par exemple?

Si. C'est dû à un mécanisme dit de la fondamentale manquante où votre cerveau peut partiellement recréer la fondamentale en présence de plusieurs harmoniques, lorsqu'elle n'est pas présente, et deviner sa hauteur d'origine (c'est utile pour vous permette d'entre la basse dans un morceau électronique lorsque les hauts-parleurs de votre ordinateur ne savent pas la reproduire, ou la fondamentale d'une voix grave lorsque votre téléphone ne sait pas la reproduire non plus, par exemple. Le brevet dit aussi que c'est utile pour certaines voyelles où la fondamentale a peu d'énergie).

Du coup, Auto-Tune cherche à pallier ce problème également. En essayant des décalages de la fréquence la plus haute à la plus basse, il ne va pas se contenter de calculer $E(L) - 2H(L) \leq$

$eps \times E(L)$ pour la fréquence la plus haute qui satisfait ce critère, mais aussi en remplaçant L par $2L$ (multiplier la longueur d'onde par 2, ce qui revient à diviser la fréquence par 2). Si la différence entre $E(L) - 2H(L)$ est *plus faible pour l'harmonique du dessous*, alors on considère que c'est la fondamentale.

i

Remarquez que chacune des étapes que je vous décris est illustrée, non seulement en texte, mais aussi en diagrammes dans le brevet. Le brevet comprend cinq schémas qui sont exposés au début du document:

- Le schéma n° 1 décrit très schématiquement les composants d'un Auto-Tune matériel. Si on étudie la partie logicielle de l'implémentation, on ne retiendra évidemment que la partie centrale de ce dessin, le «microprocesseur».
- Le schéma n° 2 décrit toujours l'implémentation matérielle, mais cette fois la boucle principale d'instructions exécutée au sein du microprocesseur (charger les paramètres, afficher l'interface, traiter les signaux qui arrivent, etc.). Avec une implémentation logicielle sous forme de plug-in, cette partie est gérée par l'interface VST, comme nous le verrons plus bas.
- Le schéma n° 3 décrit le mécanisme de correction de la hauteur que l'on vient de voir. Il est exécuté au début du processus de traitement de la voix, au moment où la hauteur de la voix n'a pas été clairement identifiée et jusqu'à ce qu'elle le soit, ainsi qu'au moment où elle n'est plus reconnue par le mécanisme de suivi de la hauteur que nous allons voir ensuite.
- Le schéma n° 5 décrit le mécanisme de suivi de la hauteur qui permet de détecter les variations minimes de la hauteur par la suite tout en faisant moins de traitements.
- Le schéma n° 4, enfin, décrit le mécanisme de correction de la hauteur à proprement parler.

Chaque étape d'un de ces schémas est reliée au texte par un numéro d'étape qui est référencé par des clauses.

4.2. Le suivi de la hauteur

?

Maintenant qu'on a trouvé la note de la voix, est-ce que le calcul sera fait à nouveau à chaque fois que du son arrive?

Encore une fois, non. On va continuer à faire des autocorrélations, mais d'une part sur un signal qui ne sera pas downsamplé (pour détecter des variations plus précises), et d'autre part **sur une plage restreinte de fréquences autour de la fondamentale qui a été précédemment trouvée**. Si on ne la trouve plus, et que la nouvelle fondamentale s'est trop éloignée de la précédente, alors on va revenir au mécanisme initial (voir la section précédente).

Le brevet suggère, sur le signal non-downsamplé à 44100 Hz, de ne tolérer que 4 samples de plus ou de moins de changement d'une cycle de périodicité à un autre (c'est la marge pour passer d'un décalage d'un sample sur le signal downsamplé à un autre – plus vous montez haut dans les fréquences, plus cette différence est importante, dans les fréquences de la voix ça va faire

quelques hertz). Autant dire que ce mécanisme n'est utilisé que pour les changements à court terme. En plus, il n'est exécuté que tous les cinq nouveaux samples non-downsamplés.

C'est ici que l'on commence à calculer le changement de hauteur que l'on va introduire pour réaliser l'étape finale du traitement, **la correction de la hauteur**.

? Comment fait-on ça, d'ailleurs?

4.3. La correction de la hauteur

Changer la hauteur de la voix revient plus ou moins à rétrécir ou agrandir la taille de chaque cycle ayant la taille de la fondamentale. Sur le papier, c'est un mécanisme très proche du *downsampling* (pour rendre la hauteur plus importante) ou de l'*upsampling* (pour rendre la hauteur plus basse) que nous pouvons effectuer pour changer la fréquence d'échantillonnage du signal.

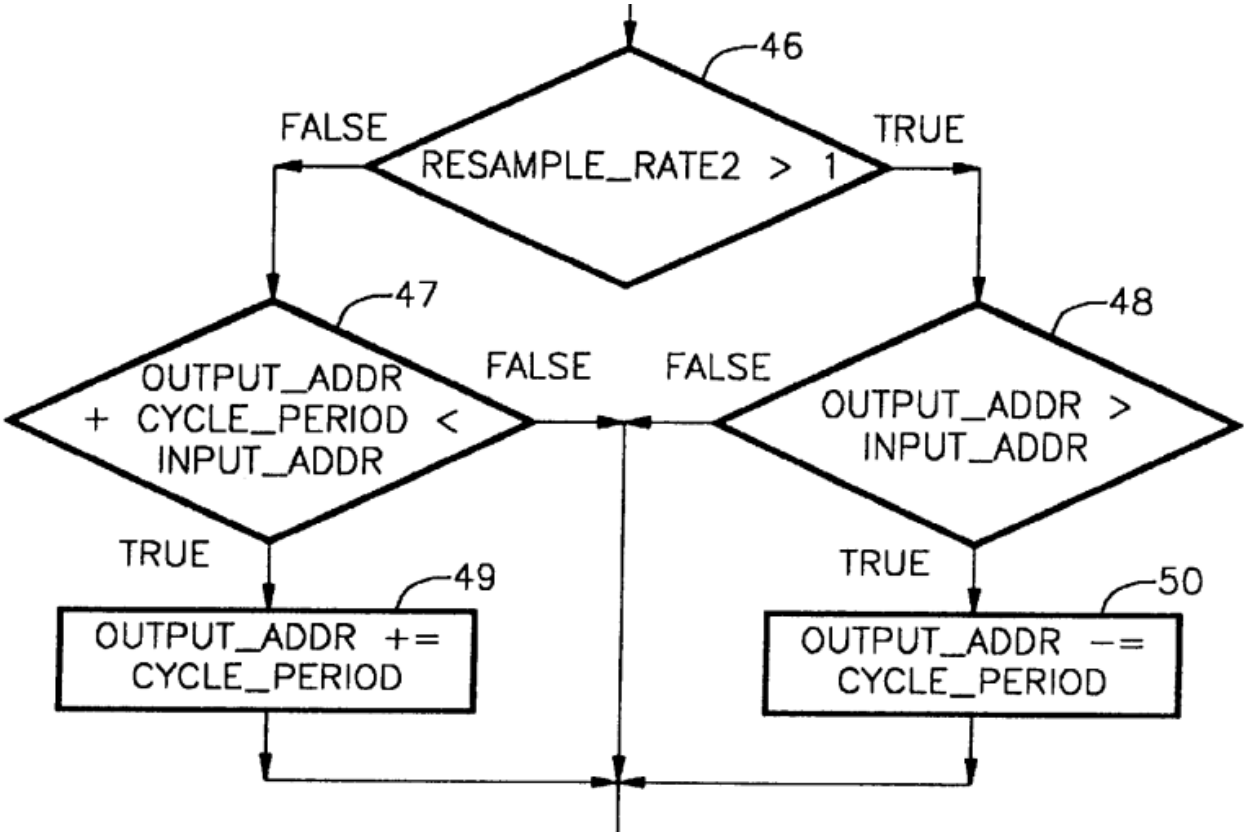


FIGURE 4.12. – Parfois, la réalité n'est pas surprenante… Changer la hauteur, c'est essentiellement **ajouter** ou **enlever de l'espace entre les samples**. Si la hauteur augmente, donc que nos cycles rétrécissent, cela fera un morceau du cycle précédent à reproduire en moins. Si elle diminue, donc que nos cycles s'élargissent, cela fera un morceau de cycle à reproduire en plus.

On sait que l'on veut passer de la toute dernière hauteur détectée de la voix, à celle alignée sur la note de la gamme que nous avons choisie la plus proche. Ces changements de hauteur se feront donc en interpolant (calculer une moyenne transitoire entre plusieurs valeurs) les derniers changements de hauteur détectés sur la voix, en prenant en compte les deux paramètres d'ajustement progressifs qui peuvent être passés par l'utilisateur au programme, dont nous avons parlé plus haut:

— Créer des changements de hauteur plus progressifs entre les variations au niveau de la fondamentale d'origine, et le réalignement sur la hauteur désirée: cela afin de donner un aspect plus humain à la voix. (Paramètre *Decay* dans les schémas)
— Introduire éventuellement en plus une vibration artificielle sur la fondamentale, afin de renforcer l'aspect plus humain de la voix, encore une fois si souhaité par l'utilisateur. (Paramètre *Vibrato_modulation* dans les schémas)

5. Anatomie d'un plug-in VST

Auto-Tune n'est généralement pas distribué et utilisé comme un bout de métal métal à ranger sur une étagère, mais comme un **plug-in logiciel** à installer dans votre éditeur audio ou votre séquenceur (plus généralement DAW - *Digital Audio Workstation*, station de travail audio numérique).

Il y a aujourd'hui et depuis la fin des années 90 une grande interface standard pour créer des plug-ins qui savent s'interfacer avec à peu près tous ces logiciels, **le format VST**. Il a connu plusieurs versions, la dernière est la 3, nous nous intéresserons ici à la version 2.4 (apparue au cours des années 2000) qui est toujours la plus utilisée et distribuée.

? VST, c'est un format, du coup?

Pas exactement. **Un plug-in VST est**, sous Windows, **un fichier DLL**. Autrement dit c'est juste un bout de logiciel présenté sous forme de *bibliothèque* (c'est-à-dire un morceau de code détaché du reste si vous êtes non-programmeur), qui exposera des *fonctions* dont certaines recevront ou retourneront des portions d'information standards (le son à traiter et le son traité, des informations sur le plug-in, ses paramètres qu'il s'agisse de les changer, de les charger ou de les sauvegarder, des notes sur un clavier MIDI…).

Le logiciel qui charge le plug-in (en général un DAW) s'appelle, quant à lui, *l'hôte VST*.

! Attention, à partir de là, le billet rentre dans les détails techniques et s'adresse davantage aux informaticiens aguerris. Vous pouvez sauter à la conclusion si l'informatique n'est pas votre domaine.

Il existe aussi des VST natifs pour d'autres plateformes que Windows, mais on utilise plus communément ce qu'on appelle des ponts VST, des bouts de logiciels qui vont d'un côté se présenter comme un hôte pour le plug-in (si vous êtes sous Linux et que vous cherchez à charger un plug-in Windows, cette partie utilisera *Wine*), et d'un autre côté comme un plug-in qui se

chargera dans le vrai hôte (par exemple, un .SO à charger dans LMMS ou Ardour si vous faîtes de la musique sous Linux).

Un certain nombre de ces ponts existent sous Linux. Ils s'appellent LinVst , AirWave , vst-bridge , YaBridge , ou encore Carla . Certains logiciels, comme LMMS, en incluent un de base, mais leur compatibilité peut être limitée selon les versions.

? Et, comment est-ce que le VST fait pour afficher *sa propre fenêtre à l'intérieur de celle du DAW*?

Eh bien, **c'est en fait l'hôte qui envoie un objet fenêtre** (créé avec l'API du système: un objet de type `HWND` sous Windows, `Window` (Xlib) sous Linux ou `WindowRef` sous macOS) à travers un appel de bibliothèque. Le plug-in dessine ensuite dedans, le plus souvent en passant par OpenGL.

VST est à l'origine développé par *Steinberg*, la société allemande qui développe le DAW Cubase, qui l'a défini initialement en 1996. Le téléchargement du SDK (kit de développement) pour VST 3 est gratuit, en théorie soumis à la signature d'une charte à envoyer par e-mail. Mais de très nombreuses versions se trouvent sur Github, y compris du SDK VST 2 .

Il existe aussi des frameworks C++ entiers, fonctionnant par-dessus le SDK, permettant de créer votre propre interface graphique et plus globalement votre propre code de traitement pour votre VST. Le plus connu et populaire s'appelle JUCE . Il est écrit par les auteurs du DAW Tracktion.

Outre le SDK officiel, qui est écrit en C++, des implémentations du format VST existent aussi dans d'autres langages, par exemple dernièrement Rust (j'ai testé, ça fonctionne).

? Ok, et du coup, au niveau logiciel, le plug-in s'interface comment avec l'hôte exactement?

Allons voir ça de plus près.

Dans notre fichier .DLL, il y a une fonction externe (un *symbole*) et une seule qui est exposée par le plug-in VST 2, elle s'appelle *VSTPluginMain*.

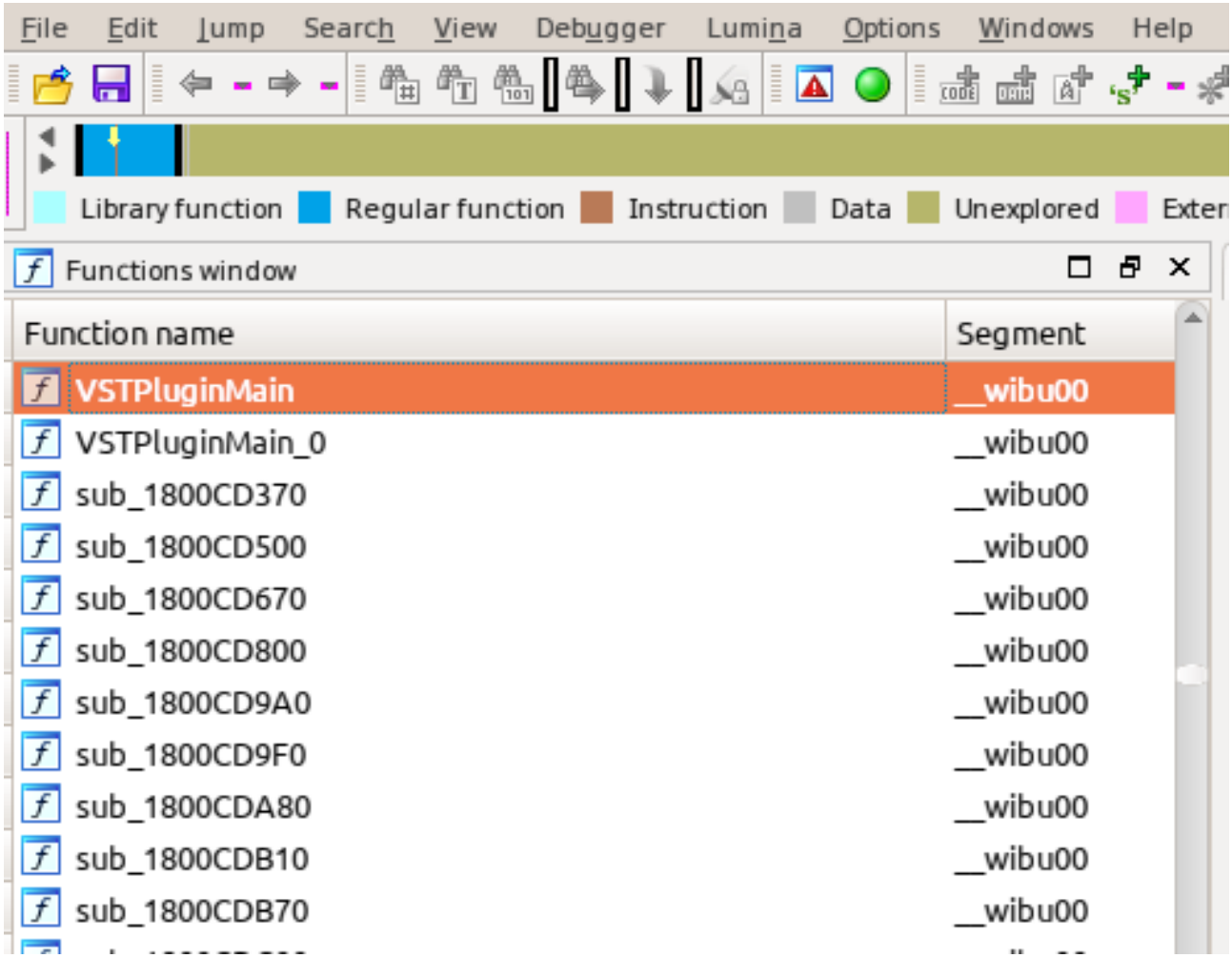


FIGURE 5.13. – La fonction *VSTPluginMain* est la seule à porter un nom au sein du plug-in compilé, si on omet la fonction *DllEntryPoint* qui est présente nativement dans les DLL Windows.

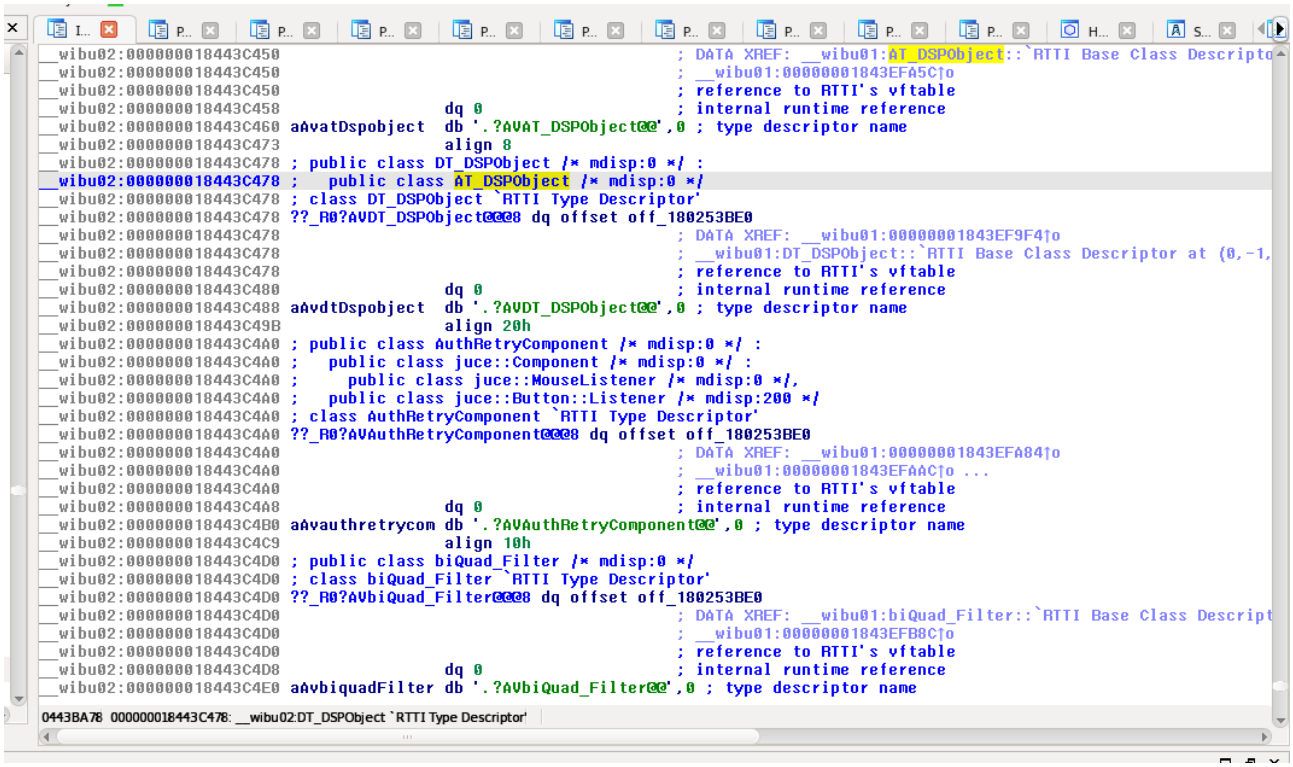


FIGURE 5.14. – Néanmoins, le compilateur (ici Visual C++) a laissé des métadonnées (appelées métadonnées RTTI) qui permettent de connaître le nom ainsi que la chaîne d'héritage de certaines classes, et ainsi de mettre un nom dessus. Ces objets de métadonnées sont eux-mêmes référencées indirectement lors de l'initialisation des structures binaires représentant les objets instanciés. Au sein du code, on repère ainsi facilement les classes ajoutées par le développeur du plug-in, dont certaines sont notamment clairement liées au traitement de signal (*DSP*, *Digital Signal Processing*) et d'autres sont des extensions du code de JUCE, dont on reconnaît clairement certaines fonctions (par exemple, ce callback pour `processReplacing`) données à titre d'exemple dans le code.

La fonction *VSTPluginMain* est donc appelée par l'hôte au lancement du plug-in, elle prendra en entrée un pointeur vers une fonction permettant d'envoyer des messages à l'hôte, et elle retournera une structure nommée `AEffect*` (laquelle est définie ici) qui contiendra diverses métadonnées sur le plug-in (version, identifiant unique du plug-in), mais aussi **plusieurs callbacks** (fonctions à rappeler par l'hôte) importants:

— `dispatcher` qui permet de recevoir toutes sortes de commandes, désignées par des codes numériques appelées *opcodes* (définis là et ici pour les plus avancés)
— `setParameter` et `getParameter` qui permettent de régler des paramètres textes (nous verrons ce que c'est plus bas)
— `processReplacing` et `processDoubleReplacing`, qui permettent d'effectuer des traitements sur le son ou de retourner le son produit, respectement en utilisant des nombres flottans de 32 ou 64 bits

À noter qu'il existe de nombreuses commandes qui peuvent être exécutes du plug-in vers l'hôte comme de l'hôte vers le plug-in. Le plug-in peut demander son nom et sa version à l'hôte et adapter son comportement selon (à la manière des User-Agents pour les navigateurs), etc. De même que chaque VST dans la structure `AEffect*` précise dans le champ `flags` les

fonctionnalités qu'il supporte (est-ce un synthétiseur ou un effet, est-ce qu'il supporte telles extensions, etc.).

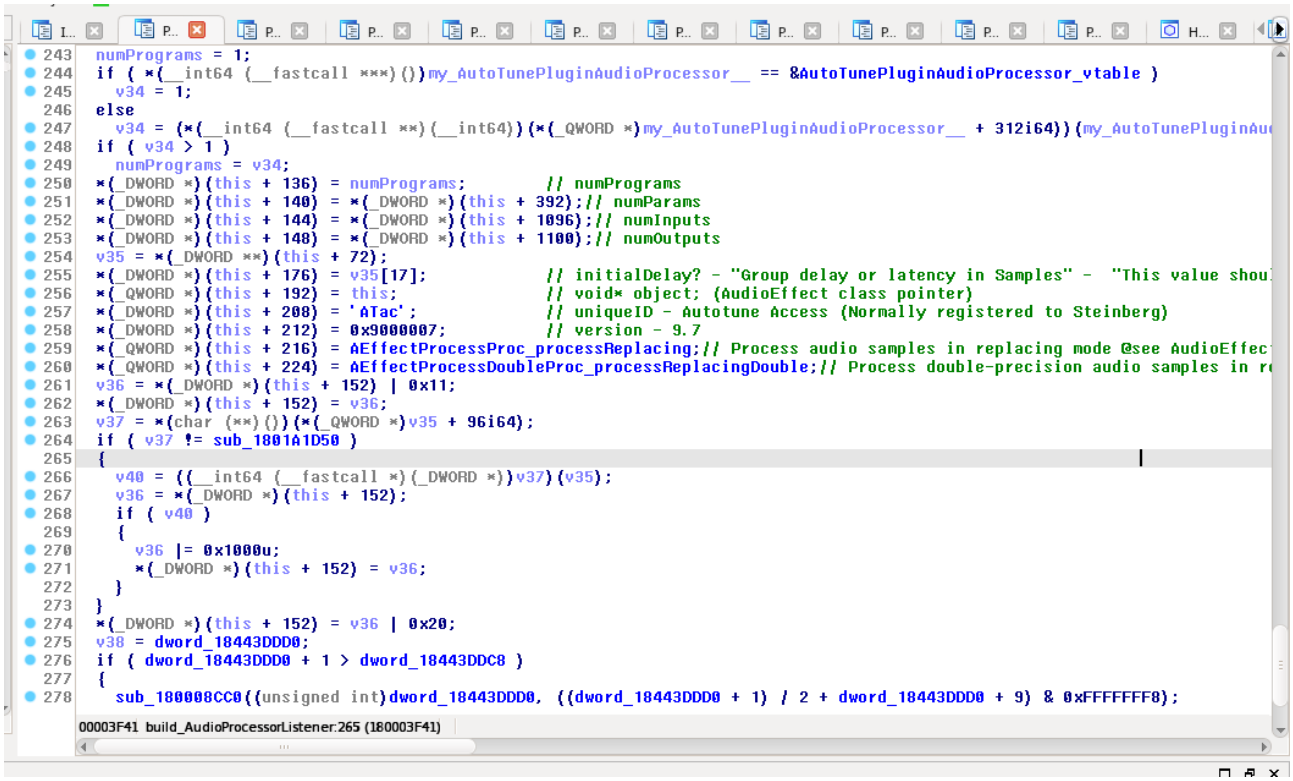


FIGURE 5.15. – Le remplissage de la structure `AEffect*`, très reconnaissable, annoté par mes soins

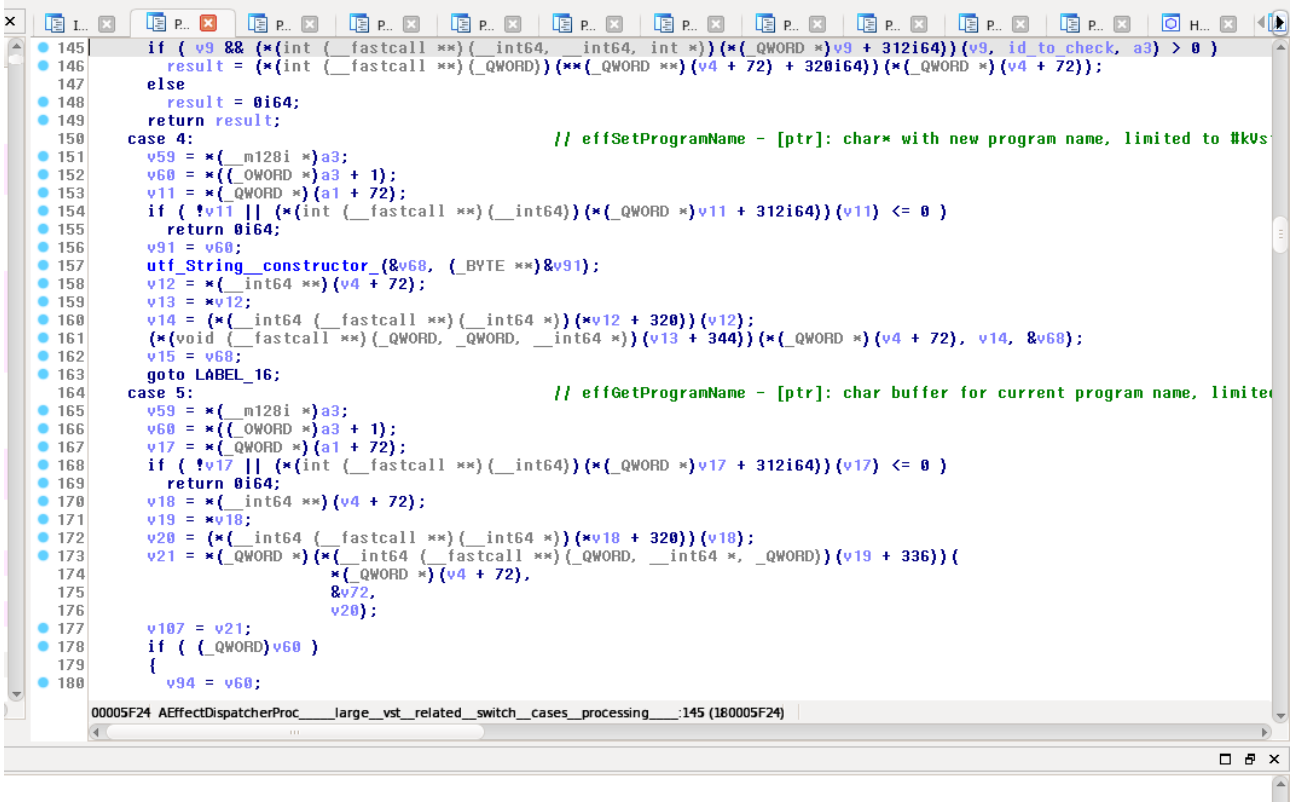


FIGURE 5.16. – Un morceau du callback `dispatcher`, (très partiellement) annoté par mes soins également.

? Une dernière question, est-ce qu'un VST s'utilise *uniquement* en mode graphique?

Non. Le manuel du développeur fourni par Steinberg recommande même de travailler au départ en mode texte lorsque vous concevez un VST, afin de faciliter le début du développement. Pour ne pas utiliser un VST en mode graphique, il suffit que l'hôte n'envoie pas d'objet fenêtre à initialiser.

Un VST expose des paramètres qui disposent chacun d'un nom et de valeurs (des nombres flottants) qui peuvent être connus et modifiés par l'hôte. Soit le plug-in expose dès son lancement tous ces paramètres (c'est le cas d'Auto-Tune qui peut techniquement être utilisé sans interface graphique), soit soit une option de votre DAW vous permet de faire en sorte que vous cliquiez sur un des réglages présents dans l'interface du plug-in, et qu'il retourne au DAW un nouveau paramètre texte à gérer.

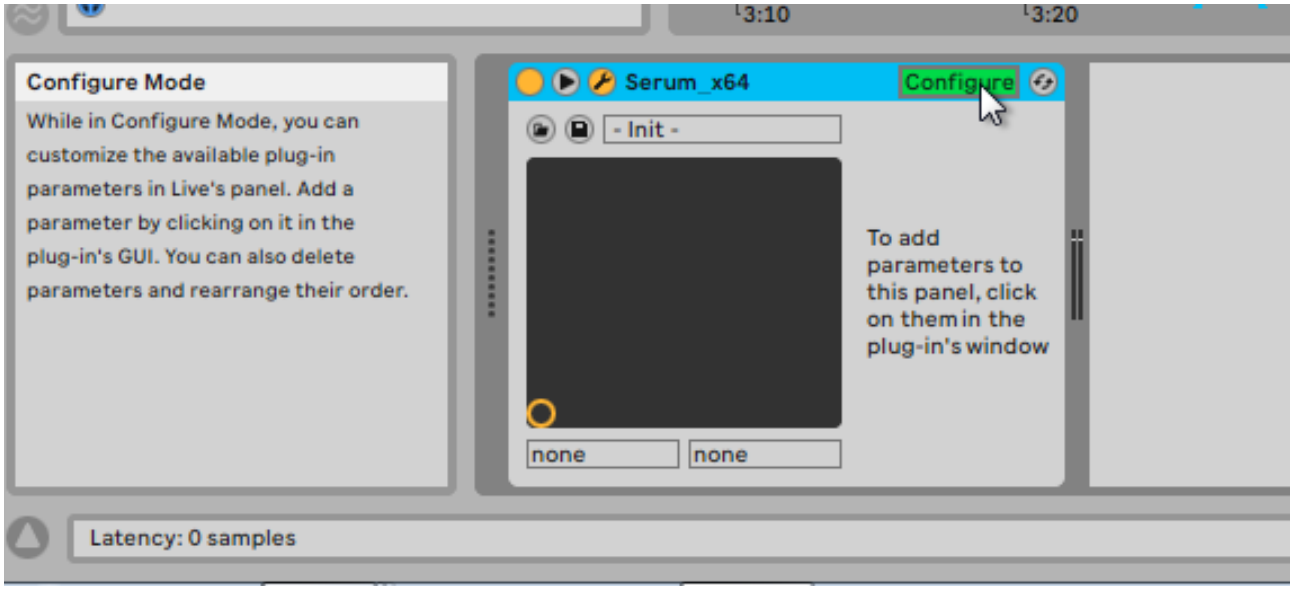


FIGURE 5.17. – Illustration de l'accès à la fonction « Configure » sous le logiciel Ableton.

Ensuite, le DAW peut décider de changer lui-même, progressivement les paramètres du plug-in au fur et à mesure qu'il joue le morceau, par exemple en faisant des transitions: c'est ce que le logiciel Ableton, par exemple, propose sous le nom d'*automations*. Vous pouvez dessiner des *automations* à la souris sur chaque piste composant votre morceau, cela vous permettra d'introduire des variations progressives sur n'importe quel réglage exposé par le plug-in VST (ou par le DAW lui-même, comme le réglage du volume…), c'est donc appelé ainsi parce que c'est «comme si ça tournait les boutons automatiquement» (c'est ce que ça fait).

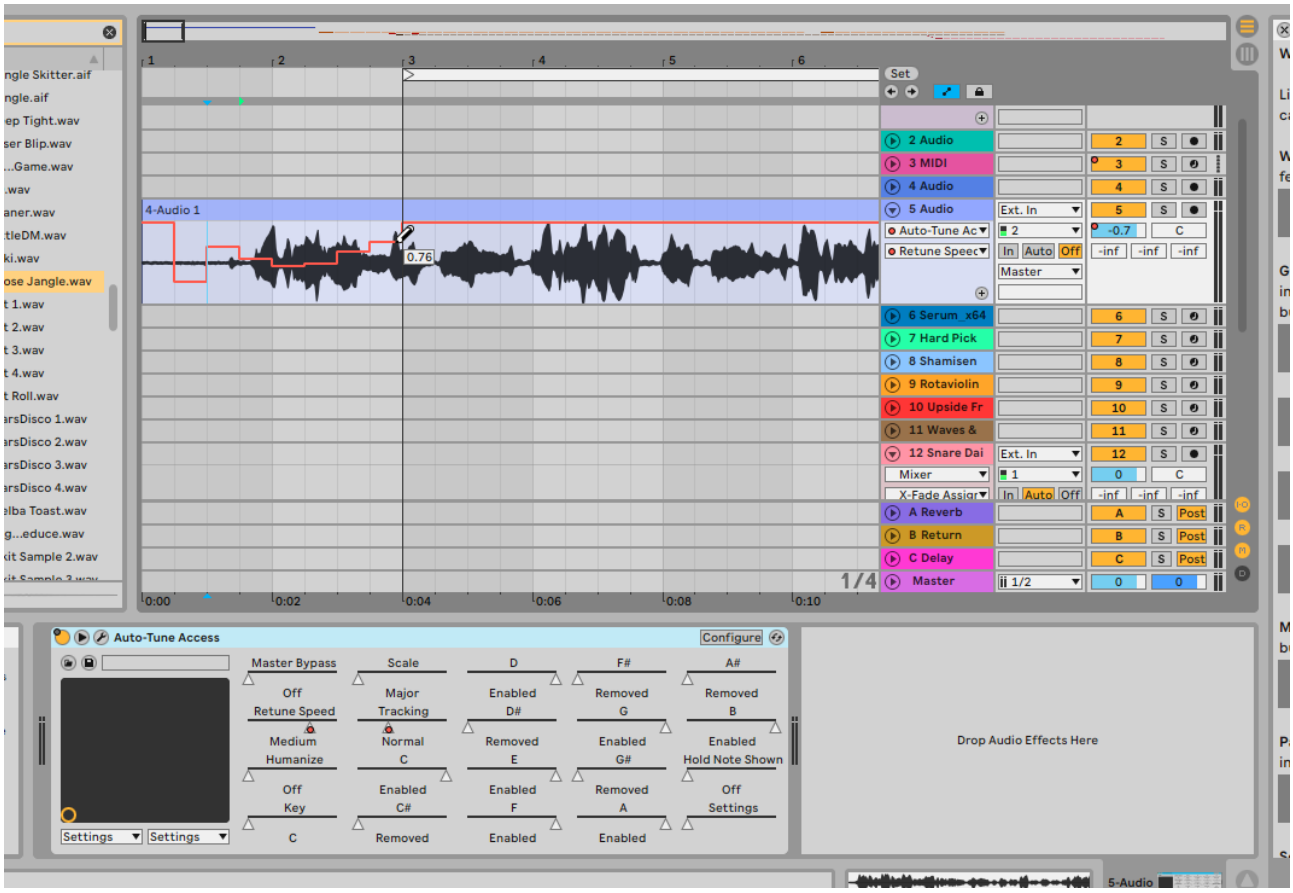


FIGURE 5.18. – Réglage d'une automation à la main sur un des paramètres d'Auto-Tune, dans l'interface d'Ableton.

? Mais, ça veut dire que je pourrais faire de l'Auto-Tune en ligne de commande par exemple?

Carrément. Le moyen le plus simple pour ça est d'utiliser un logiciel libre appelé MrsWatson qui vous permet simplement d'utiliser un VST en mode non-graphique, quand celui-ci le permet. Pour utiliser Auto-Tune sous Linux sans interface, en réglant le plug-in sur la plus faible latence de correction (ce qui produit la voix la plus robotique), je peux utiliser cette commande (ici sous forme de fonction Bash):

```
1 autotune() {
2     wine ~/win32/MrsWatson-0.9.7/Windows/mrswatson64.exe
          --plugin-root "Z:\home\marin\win32\Antares" --plugin
          "Auto-Tune Access" --input "$@" --output
          /tmp/output_autotune.wav --display-info --parameter 1,10000
3     mplayer /tmp/output_autotune.wav
4 }
```

Le paramètre `--display-info` utilisé seul vous listera les paramètres texte acceptés par le plug-in dès son lancement. Illustration de sortie complète:

Contenu masqué n°1

Bon, je crois qu'on a tout dit ou presque sur l'Auto-Tune?

Allez, une petite citation de Daft Punk pour clore ce billet.

> On nous demande toujours quel vocodeur on utilise. Mais la vérité c'est qu'on en utilise un différent à chaque morceau. On en a un de Roland [SVC-350], *Auto-Tune*, un Digitech Vocalist. On utilise la même approche pour les décaleur de phase. Chaque morceau de l'album utilise un décaleur de phase différent. Nous avons un phaser Mu-Tron, une pédale Moogerfooger, un vieux phaser AMS, et un Ensoniq DP-4. Avec les versions plus anciennes, on peut avoir plusieurs versions du même modèle mais le son sera différent à chaque fois.
> On utilise aussi les vocodeurs d'une façon dont la plupart des gens ne les utilisent pas. *Auto-Tune* c'est bien pour chanter juste, mais nous utilisons *Auto-Tune* d'une manière pour laquelle il n'a pas été conçu. Beaucoup de gens se plaignent des musiciens qui utilisent *Auto-Tune*. Ça me rappelle les années 70 quand des musiciens ont tenté d'interdire le synthétiseur en France. Ils disaient que les musiciens perdraient leur travail. Ils ne considéraient pas qu'ils pouvaient utiliser ces outils de manière novatrice au lieu de remplacer simplement les instruments qui venaient avant. Les gens ont souvent peur de ce qui a l'air nouveau.

Un vocodeur est un effet qui extrait les caractéristiques de la voix pour en faire une resynthèse, souvent pour la mélanger avec un autre instrument. Auto-Tune n'en est pas un mais s'en rapproche. Un décaleur de phase va rejouer un même son par-dessus un premier, mais en ajoutant un décalage temporel variable afin de recouvrir certes fréquences de manière à créer un effet de «balayage» qui va générer des trous sur les fréquences de manière périodique (quand certaines fréquences sont privilégiées, il s'agit d'un phaser , sinon il s'agit d'un flanger - deux effets très en vogue dans les années 70).

One more time, un des premiers tubes internationaux utilisant les capacités de déformation de voix de l'Auto-Tune:

Élément externe (Video) —
Consultez cet élément à l'adresse `https://www.youtube.com/embed/FGBhQbmPwH8?feature=oembed`.

Contenu masqué

Contenu masqué n°1

```
$ wine ~/win32/MrsWatson-0.9.7/Windows/mrswatson64.exe
    --plugin-root "Z:\home\marin\win32\Antares" --plugin "Auto-Tune
    Access" --input ~/junk/Vladadadadam.wav --output
    /tmp/output_autotune.wav --display-info --parameter 1,10000
ERROR: ld.so: object 'libgtk3-nocsd.so.0' from LD_PRELOAD cannot be
    preloaded (cannot open shared object file): ignored.
ERROR: ld.so: object 'libgtk3-nocsd.so.0' from LD_PRELOAD cannot be
    preloaded (cannot open shared object file): ignored.
- 00000000 000002 MrsWatson version 0.9.7 initialized, build
    20140125
W 00000000 000002 Running in 64-bit mode, this is experimental.
    Hold on to your hats!
- 00000000 000003 Setting 2 channels
- 00000000 000003 Setting sample rate to 44100Hz
- 00000000 000003 Setting 2 channels
- 00000000 000003 Setting sample rate to 44100Hz
- 00000000 000005 Plugin 'Auto-Tune Access' is of type VST2.x
- 00000000 000007 Opening VST2.x plugin 'Auto-Tune Access'
<<<about to register WIBU client>>>
AutoKey: Registering instance with address at 00000000000D0360
AutoKey: Creating new thread singleton
AutoKey: Adding instance to map
Now there are 1 clients for this thread
- 00000000 002306 Information for VST2.x plugin 'Auto-Tune Access'
- 00000000 002306 Vendor: Antares
- 00000000 002306 Version: 70
- 00000000 002307 Unique ID: ATac
- 00000000 002307 Plugin type: effect, category 1
- 00000000 002308 Version: 150994951
- 00000000 002308 I/O: 2/2
- 00000000 002308 Parameters (20 total):
- 00000000 002308   0: 'Master Bypass' (0.000000)
- 00000000 002309   1: 'Retune Speed' (1.000000)
- 00000000 002309   2: 'Humanize' (0.000000)
- 00000000 002309   3: 'Key' (0.000000)
- 00000000 002309   4: 'Scale' (0.000000)
- 00000000 002310   5: 'Tracking' (1.000000)
- 00000000 002310   6: 'C' (1.000000)
- 00000000 002310   7: 'C#' (0.000000)
- 00000000 002310   8: 'D' (1.000000)
- 00000000 002310   9: 'D#' (0.000000)
- 00000000 002311   10: 'E' (1.000000)
- 00000000 002311   11: 'F' (1.000000)
- 00000000 002311   12: 'F#' (0.000000)
```

Contenu masqué

```
- 00000000 002311   13: 'G' (1.000000)
- 00000000 002312   14: 'G#' (0.000000)
- 00000000 002312   15: 'A' (1.000000)
- 00000000 002313   16: 'A#' (0.000000)
- 00000000 002313   17: 'B' (1.000000)
- 00000000 002313   18: 'Hold Note Shown' (0.000000)
- 00000000 002314   19: 'Settings' (0.000000)
- 00000000 002314 Programs (1 total):
- 00000000 002314   0: ''
- 00000000 002314 Current program: ''
- 00000000 002315 Common canDo's:
- 00000000 002315   sendVstEvents: No
- 00000000 002316   sendVstMidiEvent: No
- 00000000 002316   receiveVstEvents: No
- 00000000 002316   receiveVstMidiEvent: No
- 00000000 002316   receiveVstTimeInfo: Yes
- 00000000 002321   offline: Don't know
- 00000000 002321   midiProgramNames: Don't know
- 00000000 002321   bypass: Yes
- 00000000 002321 Set parameter 1 on plugin 'Auto-Tune Access' to
    10000.000000 (Fast)
- 00000000 002322 Starting processing input source
W 00135168 002396 Possible dropout! Plugin 'Auto-Tune Access' spent
    0ms processing time (0ms max)
- 00135168 002396 Finished processing input source
- 00135680 002397 Total processing time 2sec, approximate
    breakdown:
- 00135680 002398   MrsWatson Initialization: 2sec (96.9%)
- 00135680 002398   MrsWatson Input Source: 2ms (0.1%)
- 00135680 002398   MrsWatson Output Source: 1ms (0.1%)
- 00135680 002398   Auto-Tune Access Audio Processing: 68ms (2.9%)
- 00135680 002399   Auto-Tune Access MIDI Processing: 0ms (0.0%)
- 00135680 002399 Read 135168 frames from
    /home/marin/junk/Vladadadadam.wav
- 00135680 002399 Wrote 135168 frames to /tmp/output_autotune.wav
- 00135680 002399 Shutting down
- 00135680 002400 Closing plugin 'Auto-Tune Access'
AutoKey: Unregistering instance with address at 00000000000D0360
AutoKey: Removing instance from map
Now there are 0 clients for this thread
AutoKey: Deleting thread singleton
- 00135680 002404 Goodbye!
```

Retourner au texte.

Liste des abréviations

DFT Discrete Fourier Transform. 13

FFT Fast Fourier Transform. 13