Rapport du Projet Vocodeur

Eliott Vigier

Paul Zerial

Table des matières

[Introduction 2](#_Toc151384706)

[I - Modification de la vitesse et du pitch 2](#_Toc151384707)

[1. Modification de la vitesse 2](#_Toc151384708)

[2. Modification du pitch 5](#_Toc151384709)

[II - Robotisation de la voix 9](#_Toc151384710)

[III - Saturation, Réverbération et Effet Rétro de la voix 10](#_Toc151384711)

[1. Saturation de la voix 10](#_Toc151384712)

[2. Réverbération de la voix 12](#_Toc151384713)

[3. Effet Rétro de la voix 13](#_Toc151384714)

[IV - Interface graphique du vocodeur 15](#_Toc151384715)

[V - Fichiers 16](#_Toc151384716)

[Conclusion 17](#_Toc151384717)

# **Introduction**

Dans ce rapport, nous allons expliquer en profondeur les choix et étapes que nous avons faits pour réaliser ce vocodeur de phase. Tout d’abord, le vocodeur de phase est un outil de traitement numérique utilisé pour analyser et synthétiser des signaux audios, il intervient dans les domaines fréquentiel et temporel en utilisant des informations de phase extraites d'une transformation fréquentielle pour interpoler les informations présentes dans les signaux audio. Ici, nous l’avons construit pas à pas pour y ajouter des effets intéressants ; les courbes expliquées dans la suite de ce rapport reprennent les éléments vus en cours pour mieux lier la théorie et la pratique. Outre la robotisation de la voix, nous avons rajouté un effet de réverbération créant ainsi, une sorte d’écho comme si le son résonnait dans une grande pièce, un effet de saturation, ce qui entraîne une distorsion plus ou moins marquée dans le signal audio et un effet rétro au signal, ce qui réduit la qualité sonore du signal audio, lui donnant un caractère granuleux.

# **I - Modification de la vitesse et du pitch**

## **Modification de la vitesse**

1. **Fonctionnement du code**

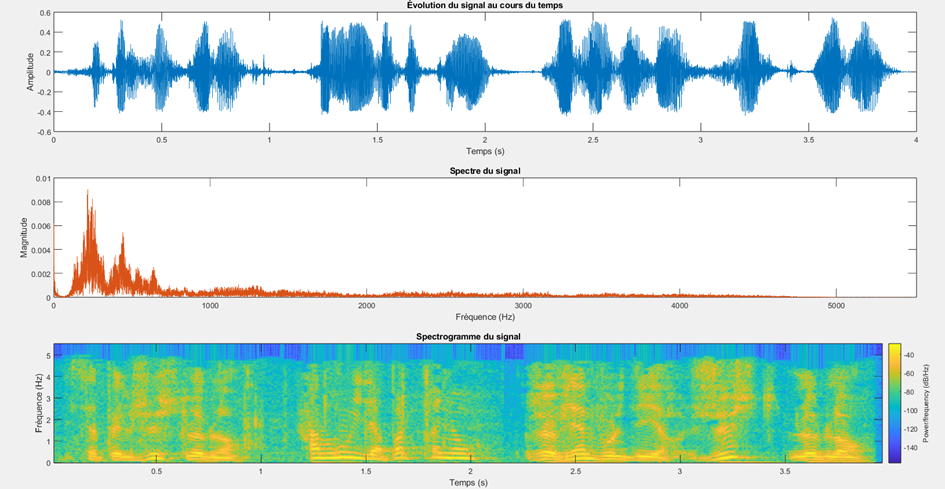
Pour réaliser la modification de la vitesse, sans modification du pitch, du signal audio, nous avons utilisé le programme *Vocodeur.m*. En effet, lors de l’exécution du code, le programme *PVoc.m* est d’abord appelé pour modifier le signal audio à l’aide d’un rapport de vitesse. Ce rapport, dans un premier temps, permet de “ralentir” le signal par rapport au signal d’origine. S’en suit les définitions des différents vecteurs pour le signal ralenti puis l’affichage du signal ralenti dans le domaine temporel, fréquentiel ainsi que le spectrogramme de celui-ci. Dans un second temps, nous avons modifier le rapport de vitesse pour accélérer cette fois-ci le signal audio.

Dans les deux cas, comme dit précédemment, on utilise le programme *PVoc.m* qui permet de modifier le rapport de vitesse. Néanmoins, le programme *PVoc.m*, utilise une fonction *TFCT\_Interp.m* que nous avons dû coder nous-même. Cette dernière est un élément important dans le processus de modification de la vitesse du signal audio sans changer le pitch. Elle permet l’implémentation de la transformée de Fourier à court terme (TFCT) et de recalculer les valeurs de la TFCT à des instants temporels différents, ce qui est nécessaire lorsqu'on change la vitesse de lecture d'un signal audio.

Ainsi, cela permettra dans un premier temps, de ralentir le signal en étirant les composantes fréquentielles dans le temps tout en conservant leur hauteur. Inversement, dans un second temps, lors de l'accélération du signal, la fonction compresse ces composantes dans le temps sans affecter la hauteur. De cette façon, ce code illustre les effets de la modification de vitesse sur un signal audio en affichant le signal original, le signal ralenti et le signal accéléré. Enfin, les graphiques sont affichés en utilisant la fonction *subplot* pour créer une présentation en colonnes.

1. Observations

**Signal d’origine :**

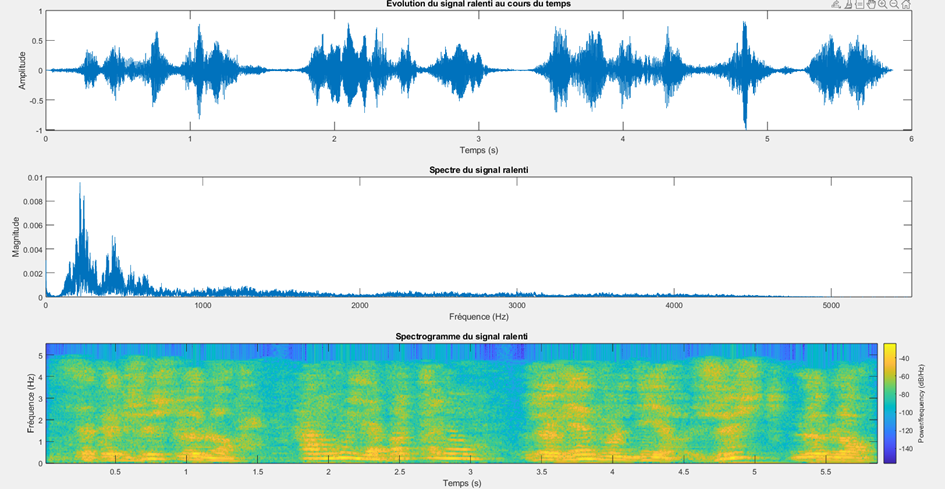
****

**Évolution temporelle :** Le signal montre des variations d'amplitude caractéristiques d'un enregistrement audio naturel, avec des pics correspondant à des événements sonores plus forts.

**Spectre de fréquences :** Le spectre indique que la majorité de l'énergie est concentrée dans les basses fréquences, avec une décroissance rapide à mesure que la fréquence augmente. Cela est typique de nombreux signaux audio naturels qui ont plus d'énergie dans les basses fréquences.

**Spectrogramme :** Le spectrogramme montre la distribution des fréquences dans le temps. Les couleurs chaudes (jaunes/rouges) indiquent des zones de plus grande intensité, à des moments où la parole ou la musique sont plus forts ou plus présents.

**Signal ralenti :**

****

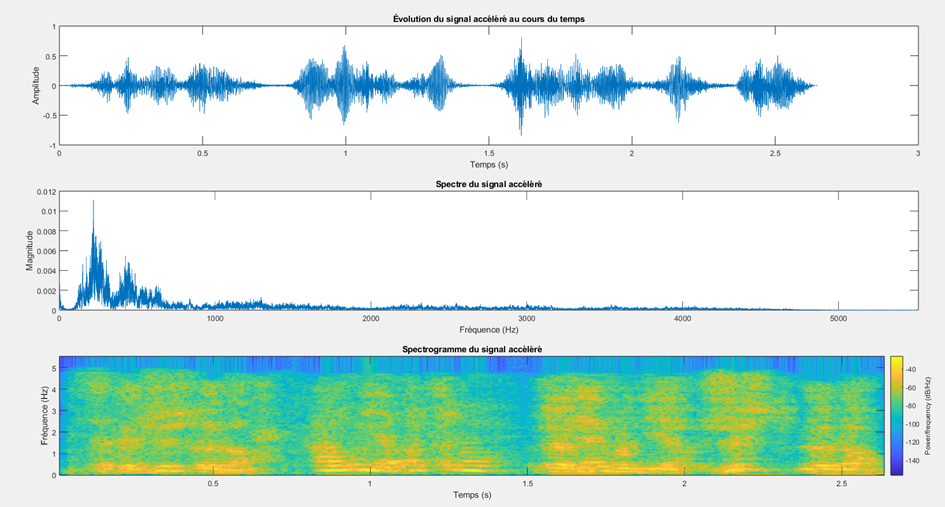
**Évolution temporelle :** Les formes d'onde sont étirées dans le temps, indiquant un ralentissement du signal. Cela se manifeste par des périodes plus longues entre les pics et les creux du signal.

**Spectre de fréquences :** Le spectre montre une énergie étalée sur une gamme de fréquences plus large et des magnitudes réduites, ce qui est cohérent avec le ralentissement du signal. Lorsqu'un signal est ralenti sans changer la hauteur (pitch), les composants de haute fréquence s'étendent sur une plage plus large.

**Spectrogramme :** Le ralentissement du signal est également visible ici, avec des événements sonores qui s'étendent sur de plus longues périodes. Les fréquences sont également étalées, ce qui reflète le changement de la structure temporelle du signal.

A l’inverse :

**Signal Accéléré :**

****

**Évolution temporelle :** On observe que les évènements du signal se produisent plus rapidement, ce qui est le résultat de l'accélération. Les formes d'onde semblent plus serrées les unes contre les autres.

**Spectre de fréquences :** Le spectre semble similaire au signal original mais avec une augmentation des fréquences. Cela est dû au fait que l'accélération du signal a tendance à augmenter la fréquence des composants sonores.

**Spectrogramme :** On constate que les événements qui étaient étalés dans le temps sont maintenant plus concentrés. Cela est cohérent avec l'accélération du signal et la réduction de sa durée.

## **2. Modification du pitch**

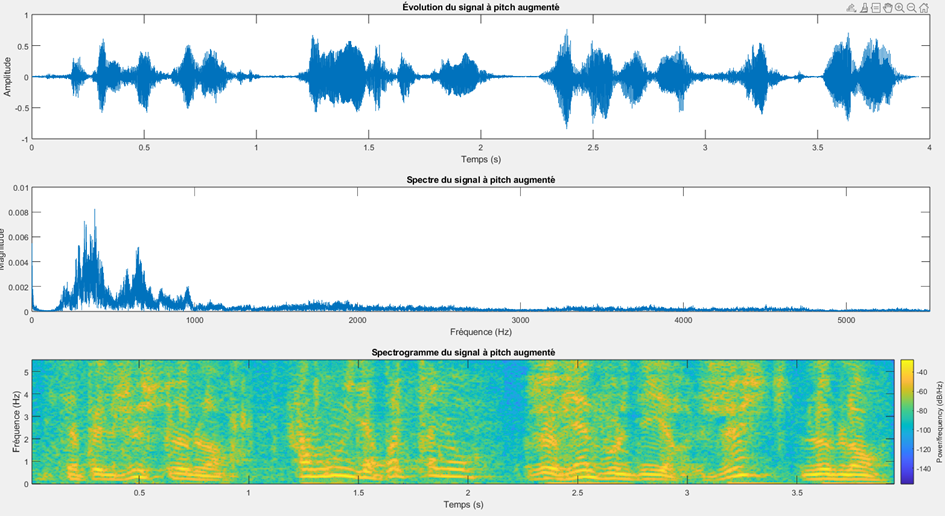
**a. Fonctionnement du code**

De manière similaire et dans le but de réaliser la modification du pitch, sans modification de la vitesse, du signal audio, nous avons utilisé aussi un rapport de pitch. Le pitch est donc d’abord augmenté en utilisant *PVoc.m*. Néanmoins la vitesse est alors changée et il faut donc rééchantillonner le signal résultant afin de maintenir la même vitesse grâce à la fonction *resample* de MATLAB. Puis, notre code contient les différentes définitions des vecteurs et des courbes qui seront affichées l’une après l’autre. Après cela, nous réitérons ces principes pour diminuer le rapport de pitch et observer les résultats comme nous le verrons plus tard.

On finit ce code par faire une somme du signal original et du signal à pitch augmenté en prenant le même nombre d’échantillons.

1. **Observations**

**Signal à pitch augmenté :**

****

**Évolution temporelle :**

Le graphique du haut montre le signal audio dans le domaine temporel. L'augmentation du pitch ne change pas la durée des événements sonores, mais augmente la fréquence des ondes sonores, les rendant plus aiguës. On peut observer que la forme d'onde générale reste la même, mais elle pourrait paraître légèrement plus "serrée" si la vitesse a également été légèrement augmentée lors de la modification du pitch.

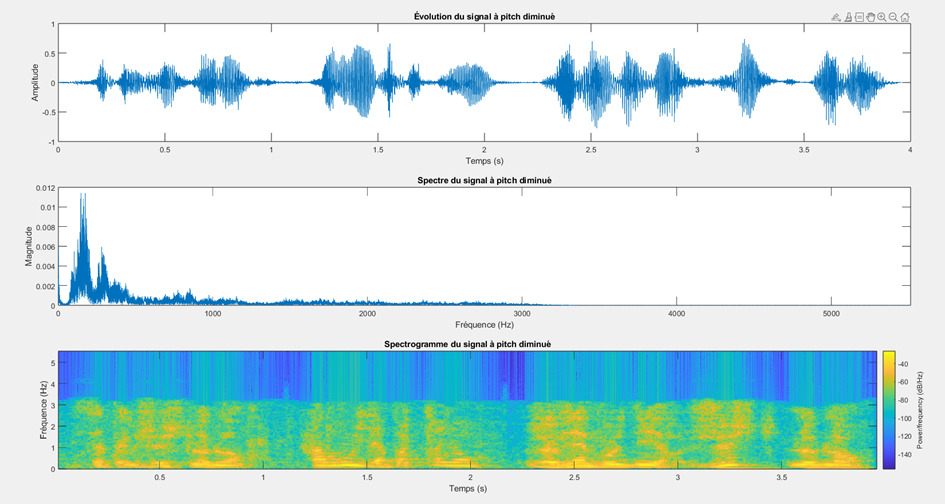
**Spectre de fréquences :**

Le spectre fréquentiel montre la répartition de l'énergie du signal sur différentes fréquences. Après l'augmentation du pitch, les pics dans le spectre apparaissent à des fréquences plus élevées par rapport au signal original, ce qui est cohérent avec des sons plus aigus. Cela est visible par le déplacement des pics du spectre vers la droite, indiquant des fréquences plus élevées.

**Spectrogramme :**

Le spectrogramme montre comment les fréquences présentes dans le signal varient avec le temps. Lorsque le pitch est augmenté, on observe les fréquences élevées plus prononcées, ce qui se traduit par une présence accrue de couleurs plus chaudes (jaune/rouge) dans les régions supérieures du spectrogramme.

**Signal à pitch diminué :**

****

**Évolution temporelle :**

Le graphique en haut montre l'amplitude du signal audio dans le temps. Avec la diminution du pitch, les ondes sonores ont une fréquence plus basse, ce qui les rend plus graves. Bien que l'allure générale de la forme d'onde soit préservée, les ondes peuvent sembler "plus larges" en raison de la diminution de la fréquence.

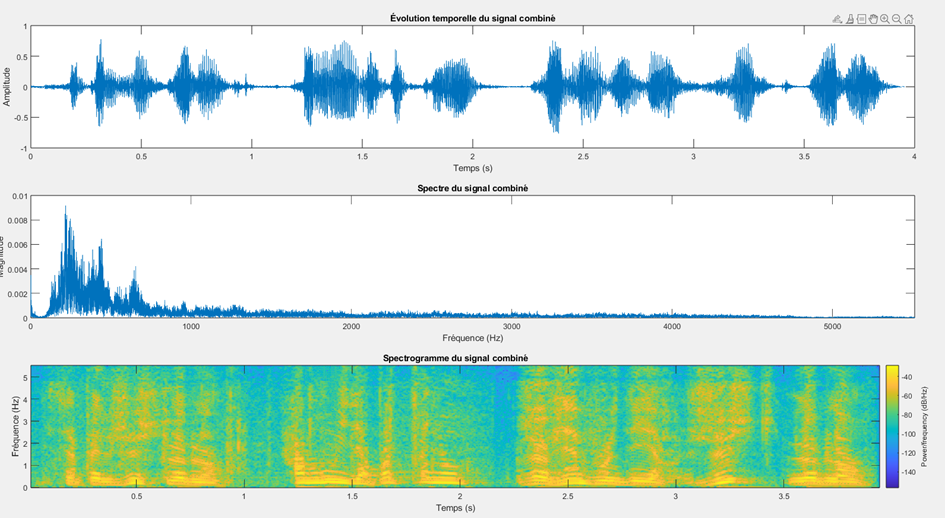
**Spectre de fréquences :**

Le graphique du milieu montre le spectre de fréquences du signal. On voit les pics du spectre se déplacer vers les fréquences plus basses, ce qui indique que le contenu en fréquences est devenu plus grave. La hauteur globale des pics est légèrement affectée, ce qui peut changer l'équilibre énergétique entre les différentes fréquences.

**Spectrogramme :**

Le graphique du bas est un spectrogramme, qui illustre les fréquences présentes dans le signal audio au fil du temps. Dans le cas d'une diminution du pitch, les zones plus chaudes (jaune/rouge) apparaissent plus basses dans le graphique, ce qui indique une prédominance de fréquences plus basses. Cela est dû au fait que les composants sonores de fréquence plus basse sont maintenant plus présents.

**Signal combiné :**

****

**Évolution temporelle :**

Le graphique en haut montre la forme d'onde du signal combiné. On observe une amplitude qui varie avec le temps, similaire à celle de l'original, mais avec des variations plus rapides ou aigües dues à la superposition de la version à pitch augmenté.

**Spectre de** **fréquences :**

Le spectre fréquentiel révèle la distribution de l'énergie du signal sur une plage de fréquences. La combinaison de l'original et de la version à pitch augmenté montre des pics de fréquences à la fois dans les gammes basses (du signal original) et hautes (du signal à pitch augmenté). Cela entraîne un spectre plus "rempli" en raison de la superposition des deux signaux.

**Spectrogramme :**

Le spectrogramme montre comment les fréquences varient dans le temps pour le signal combiné. Les fréquences aigües de la version à pitch augmenté ajoutent des composantes supplémentaires en haut du spectrogramme, tandis que les basses fréquences de l'original restent visibles en dessous. Cela donne un spectrogramme plus dense, avec des éléments des deux signaux visibles à travers le temps.

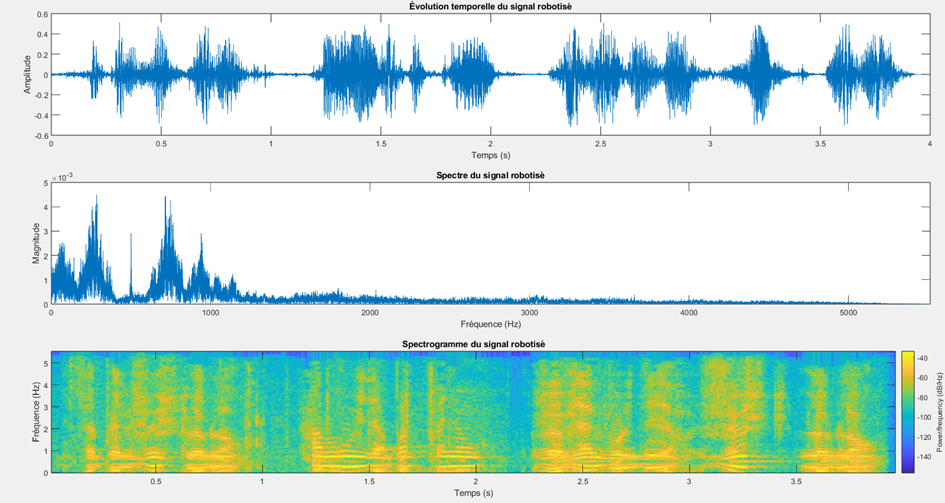
# **II - Robotisation de la voix**

1. **Fonctionnement du code**

Dans la partie *Robotisation de la voix* dans le fichier MATLAB, on commence par définir la fréquence porteuse, c’est-à-dire la fréquence fondamentale du signal robotisé. Lors de l’appel de la fonction *Rob.m*, celle-ci permet d’effectuer la modulation d’amplitude du signal original avec la fréquence porteuse. On affiche ensuite les courbes que l’on veut étudier grâce aux vecteurs définis pour représenter le signal robotisé dans le domaine temporel et fréquentiel. On affiche donc, ensuite, l’amplitude du signal au cours du temps, le spectre de fréquences puis le spectre et spectrogramme du signal.

1. **Observations**

**Signal Robotisé :**



**Évolution temporelle :**

Le graphique en haut montre la forme d'onde du signal. On peut remarquer que l'amplitude varie avec le temps comme un signal normal, mais il y a des caractéristiques distinctes comme des modulations ou des répétitions qui sont typiques de l’effet robotique.

**Spectre de fréquences :**

Le graphique du milieu montre le spectre de fréquences. L’effet robotique crée des pics distincts ou des formes (entre 0 et 1 kHz par exemple) dans le spectre dues à l'ajout de composantes fréquentielles synthétiques ou à la suppression de certaines fréquences naturelles.

**Spectrogramme :**

Le spectrogramme en bas montre la distribution des fréquences dans le temps. L'effet robotique se manifeste par des bandes fréquentielles plus définies ou des motifs répétitifs, reflétant le traitement du signal qui crée cet effet. Les changements abrupts de couleur dans le spectrogramme représentent des transitions ou des modulations qui ne sont pas présentes dans les signaux acoustiques naturels.

# **III - Saturation, Réverbération et Effet Rétro de la voix**

## **Saturation de la voix**

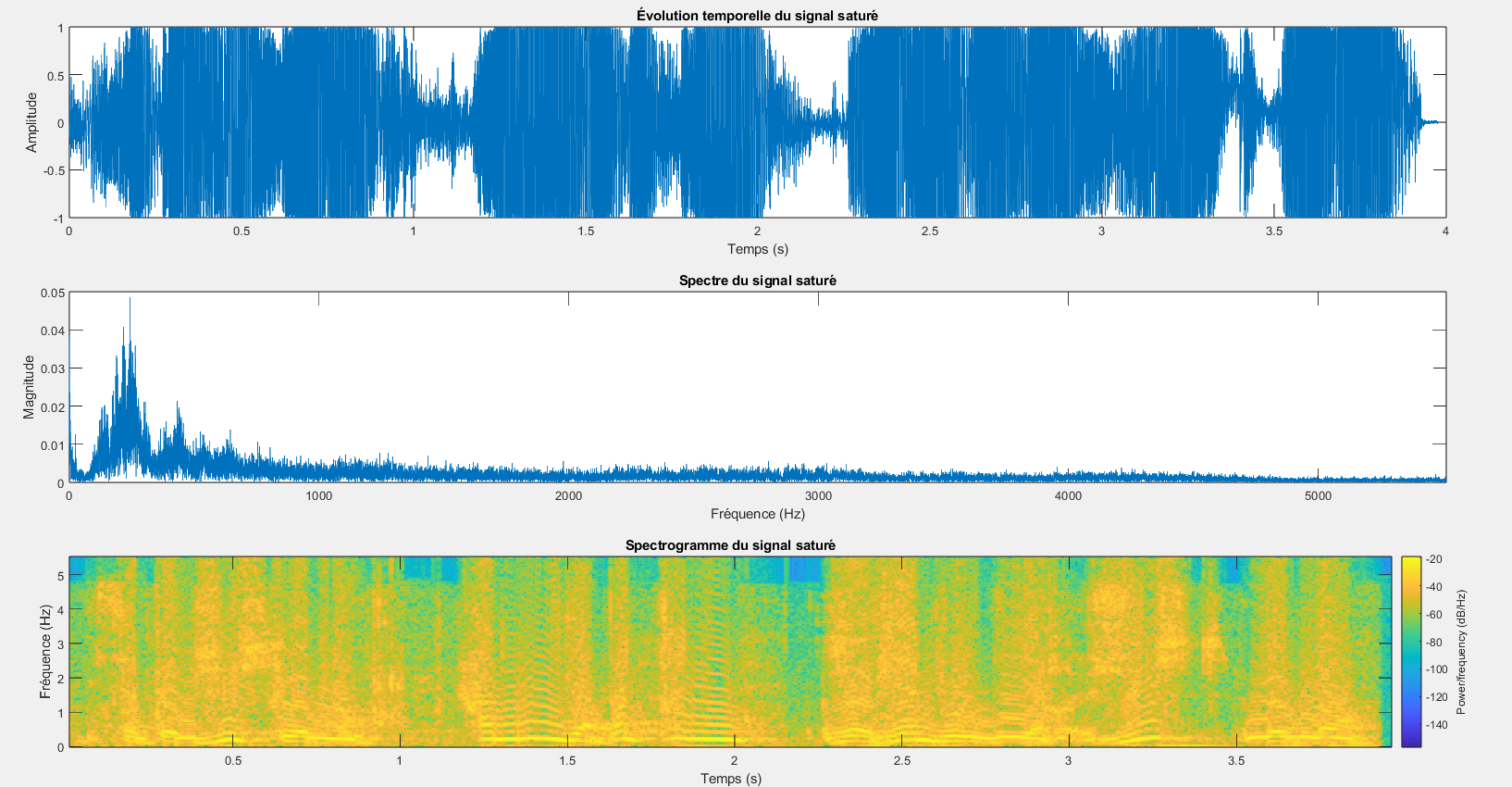
1. **Fonctionnement du code**

La fonction *Saturation.m* MATLAB permet d’appliquer un effet de saturation à un signal audio. Elle prend deux paramètres : y, le signal audio d'entrée, et “niveauSaturation”, l'intensité de la saturation à appliquer.

L'effet de saturation est réalisé en utilisant la fonction tangente hyperbolique (tanh). Premièrement, on multiplie le signal d'entrée y par le “niveauSaturation” pour modifier les amplitudes du signal. Ensuite on applique la fonction tanh. Ainsi, lorsque “niveauSaturation” est augmenté, l'effet de saturation devient plus prononcé, entraînant une distorsion plus marquée dans le signal audio. Cette dernière est caractérisée par une compression des amplitudes du signal aux extrêmes, donnant au signal un caractère plus fort et vibrant. Inversement, lorsque “niveauSaturation” est diminué, l'effet de saturation est moins fort, causant moins de distorsion dans le signal audio.

1. **Observations**

**Signal Saturé :**



**Évolution temporelle :**

Le graphique en haut montre la forme d'onde du signal audio avec une amplitude qui varie dans le temps. La saturation se caractérise par un aplatissement des crêtes de la forme d'onde là où le signal dépasse le niveau de saturation maximal, ce qui est visible ici comme des sections plates aux sommets et aux creux des ondes.

**Spectre de fréquences :**

Le spectre fréquentiel, au milieu, montre une augmentation de l'énergie sur une large gamme de fréquences, en particulier dans les moyennes et hautes fréquences. Cela est dû à la distorsion harmonique introduite par la saturation, qui ajoute de multiples fréquences supplémentaires au signal original.

**Spectrogramme :**

Le spectrogramme en bas montre la distribution des fréquences dans le temps. La saturation rend le spectrogramme plus "dense" ou "encombré" avec des informations supplémentaires sur une plus large gamme de fréquences en raison des harmoniques et de la distorsion ajoutées au signal. On voit ici que le spectre est quasiment jaune-orangé sur tout le temps. Ceci indique également que le son est beaucoup plus fort.

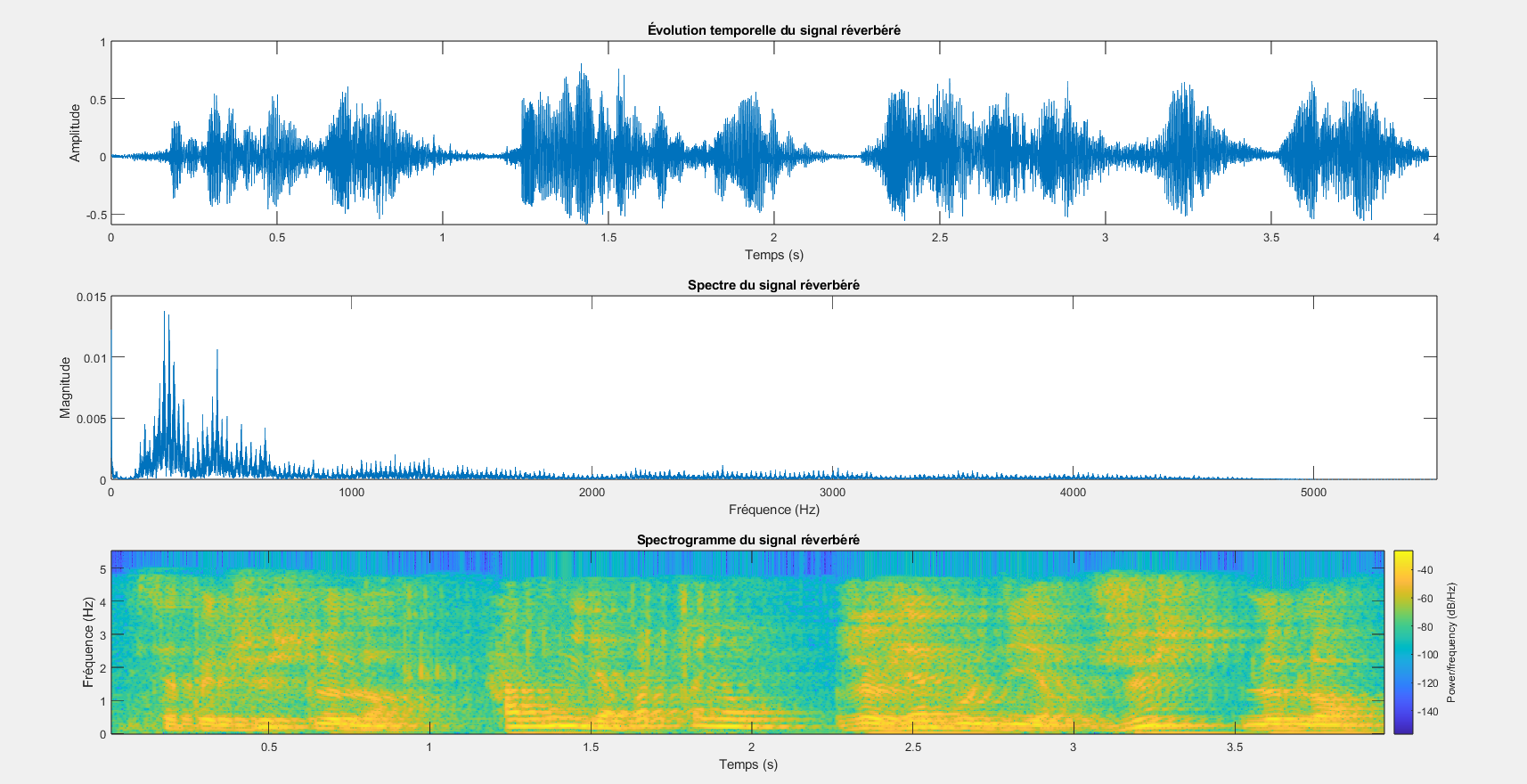
## **Réverbération de la voix**

1. **Fonctionnement du code**

Pour ajouter un effet de réverbération, on utilise le fichier MATLAB *Reverb.m*. Il utilise le signal d'entrée y, le gain de l'écho (qui doit être entre 0 et 1 pour contrôler l'intensité de l'écho), le délai de l'écho delay\_ms en millisecondes, et la fréquence d'échantillonnage Fs. Premièrement, elle convertit le délai de l'écho de millisecondes en échantillons en fonction de Fs. Puis, elle crée un nouveau signal audio où chaque échantillon du signal original est suivi d'une copie de lui-même, décalée dans le temps par le nombre d'échantillons calculé et atténuée par le gain. Ce processus crée un effet d'écho, donnant l'impression que le son résonne, comme s'il était dans un espace large ou une salle. En modifiant le gain et le delay\_ms, on peut ajuster l'intensité et la durée de la réverbération pour obtenir différents effets sonores.

1. **Observations**

**Signal réverbéré :**



**Évolution temporelle :** Le graphique supérieur montre la forme d'onde du signal audio avec une amplitude variant avec le temps. La réverbération est caractérisée par un prolongement des sons après qu'ils sont produits, visible ici comme une série d'échos ou de répétitions qui diminuent progressivement après chaque pic d'amplitude, donnant un aspect "flou" ou "étiré" à la forme d'onde.

**Spectre de fréquences :** Le spectre fréquentiel, au milieu, révèle une présence accrue de composantes fréquentielles à travers une vaste gamme, notamment dans les basses fréquences. Cela est dû à la nature de la réverbération, qui peut enrichir le signal avec des fréquences résonantes et prolonger la présence des fréquences basses et moyennes.

**Spectrogramme :** Le spectrogramme en bas montre la distribution des fréquences dans le temps avec des couleurs indiquant l'intensité. La réverbération crée une sorte de "sillage" derrière chaque note ou son, résultant en des bandes verticales qui s'étendent au-delà de l'impulsion initiale du son. Cela peut être interprété comme le temps de réverbération où les sons continuent à être présents même après qu'ils ont été joués.

## **Effet Rétro de la voix**

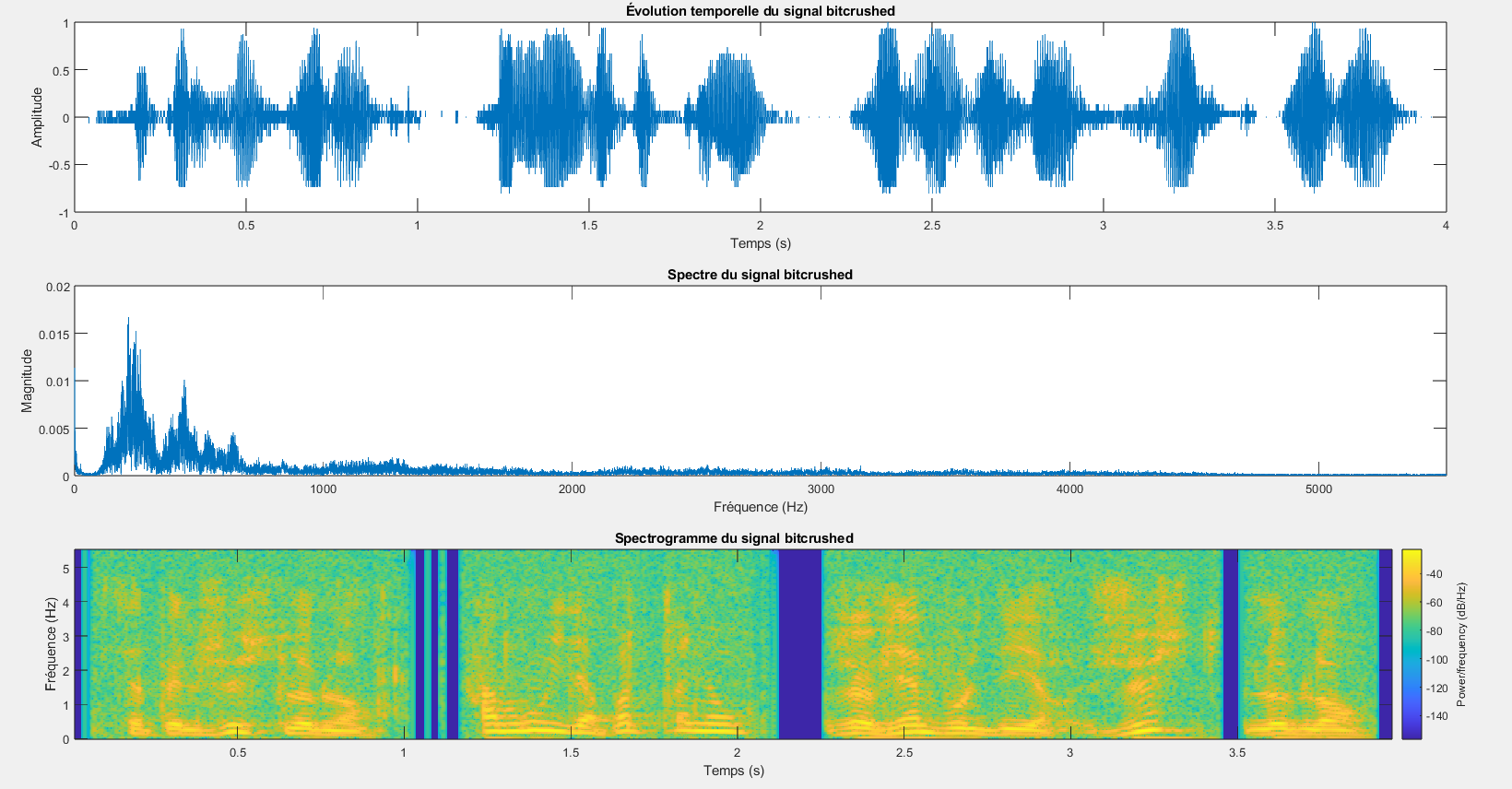
1. **Fonctionnement du code**

L'effet de bitcrushing ou rétro transforme un signal audio en réduisant sa qualité sonore, lui donnant un caractère distinctement numérique et granuleux. Il est appliqué dans MATLAB grâce à la fonction *Bitcrusher.m* qui prend deux arguments : y, le signal d’entré et bitDepth la profondeur de bit désirée pour l'effet de bitcrushing.

Dans un premier temps, la fonction normalise le signal entrant pour le maintenir entre -1 à 1, afin d'éviter toute distorsion indésirable. Ensuite, on détermine l'intensité de l'effet de bitcrushing en calculant le facteur de réduction basé sur la profondeur de bit souhaitée, puis on l’applique. Enfin, le signal est réajusté pour rester similaire au signal d’origine. Ce processus transforme ainsi le signal audio original en une version « bitcrushée » ou rétro.

1. **Observations**

**Signal bitcrushed :**

****

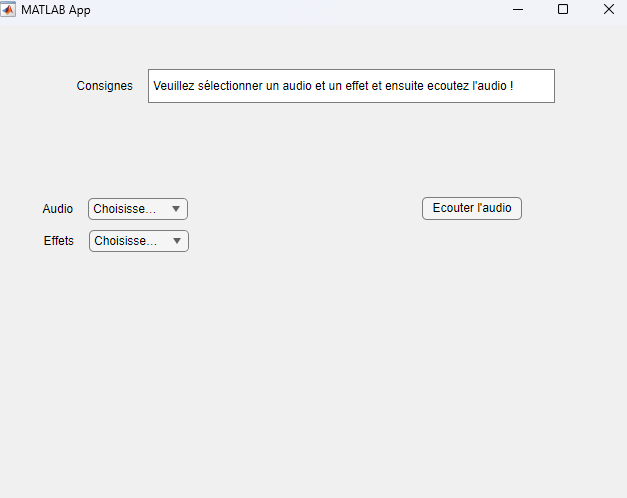
**Évolution temporelle :** Ce graphique montre l'amplitude du signal audio dans le temps. L'effet de Bitcrusher se manifeste par une apparence « en escalier » de l'onde sonore due à la quantification réduite. En augmentant « la profondeur de bit », l'effet serait de plus en plus extrême et ces « escaliers » serait de plus en plus visible.

**Spectre de fréquences :** Le graphique du milieu affiche le spectre de fréquences du signal, qui montre la magnitude des différentes fréquences présentes dans le signal. Après le bitcrushing, on voit une légère augmentation de la magnitude dans les fréquences élevées (5 kHz), due à la réduction de bit qui est légère dans notre cas.

**Spectrogramme :** Le graphique du bas est un spectrogramme, qui représente les fréquences présentes dans le signal audio au fil du temps. Les couleurs indiquent l'intensité des fréquences à différents moments. On peut observer des lignes verticales bleues foncées qui indiquent des silences ou des bruits très forts (dus à la réduction de bit). Le bitcrushing introduit un aspect visuel hachuré, fragmenté dans le spectrogramme.

# **IV - Interface graphique du vocodeur**

Nous avons développé une interface graphique pour notre vocodeur car il intègre une multitude d'effets sonores. Cette interface facilite grandement la sélection et l'application de ces effets, ainsi que le choix des fichiers audio à traiter.



**Structure de l'Interface :**

L’interface graphique (GUI) est constituée de deux menus déroulants pour la sélection de fichiers audio et d'effets, d'un bouton d'écoute pour jouer le signal modifié (ou non), et d'un champ de texte pour les consignes d'utilisation.

**Fonctionnalités :**

**Sélection de l'audio :** Un menu déroulant (fichier) permet à l'utilisateur de choisir parmi les fichiers audios prédéfinis. La sélection déclenche la fonction *fichierValueChanged* qui charge le fichier audio et met à jour les données audios courantes.

**Application des effets** : Un autre menu déroulant (effets) offre le choix entre différents effets sonores. Le changement de valeur appelle *effetsValueChanged* qui applique l'effet sélectionné au signal audio en utilisant notamment la fonction *PVoc.m*.

**Lecture du signal :** Le bouton écouter active la lecture du signal audio modifié. La fonction *ecouterButtonPushed* est liée à ce bouton pour gérer la lecture audio.

**Instructions :** Un champ d'édition (*ConsignesEditField*) affiche des instructions textuelles pour guider l'utilisateur dans l'utilisation de l'application.

**Gestion des Erreurs :**

Des alertes sont générées via *uialert* en cas de sélection d'un fichier inexistant ou tentative de lecture sans fichier audio chargé.

L'interface développée permet une interaction intuitive du vocodeur, offrant une expérience utilisateur simplifiée pour l'application d'effets audio en temps réel.

# **V - Fichiers**

Le dossier *Vocodeur\_version\_finale* contient un sous-dossier nommé "audio" qui regroupe quatre fichiers audios. Trois de ces fichiers sont les audios de base, tandis que le quatrième a été ajouté par notre groupe. En dehors de ce sous-dossier, on trouve le fichier principal *Vocodeur.m*. Lorsque ce script est exécuté, il produit l'évolution temporelle, le spectre de fréquences, et le spectrogramme du signal audio original ainsi que des différents effets appliqués. Le fichier est configuré pour jouer par défaut l'audio *extrait\_cours.wav*, mais il offre la possibilité de commenter ou décommenter des sections du code pour écouter chaque son et effet individuellement.

Pour faciliter la sélection des fichiers audio et leur écoute, il y a également une interface graphique, nommée *Interface\_graphique\_du\_vocodeur.mlapp*. Cette interface simplifie l'interaction avec le script *Vocodeur.m*, rendant l'expérience utilisateur plus intuitive et directe.

Ces deux fichiers, *Vocodeur.m* et *Interface\_graphique\_du\_vocodeur.mlapp*, utilisent un ensemble de fonctions supplémentaires pour le traitement du signal audio. Ces fonctions incluent *PVoc.m* pour le traitement de pitch via *Phase Vocoder*, *TFCT.m* et *TFCTInv.m* pour la transformation de Fourier à court terme et son inverse, *TFCT\_Interp.m* pour l’interpolation des résultats de la TFCT, ainsi que *Reverb.m*, *Saturation.m*, *Rob.m*, et *Bitcrusher.m* pour appliquer divers effets audios comme la réverbération, la saturation, l'effet robotique et le bitcrushing. Ensemble, ces outils forment un ensemble complet pour l'analyse et la manipulation audio dans le cadre du projet sur le vocodeur.

# **Conclusion**

Pour conclure, ce projet nous a permis de comprendre et d’analyser de manière plus fine les effets des transformées étudiées en cours et ainsi de construire les différents codes demandés : *Rob.m*, *TFCT\_Interp.m* et *Vocodeur.m*. Nous avons pu aussi explorer d’autres applications à ce vocodeur de phase comme les fonctions *Reverb.m*, *Saturation.m*, *Rob.m*, et *Bitcrusher.m*. Nous avons ainsi successivement étudié la modification de la vitesse et du pitch, la robotisation de la voix, ainsi que l'application d'effets tels que la saturation, la réverbération, et l'effet rétro. Enfin, l'interface graphique développée a permis de simplifier l'interaction avec le vocodeur, offrant une expérience utilisateur plus simplifiée.