

2022/03/05 版

NNSVS・Vocal2lab
音声合成チュートリアル

著 148nasuka

第1章 VOCAL2LAB	2
1.1 VOCAL2LABについて	2
1.2 VOCAL2LABのセットアップ	4
1.3 VOCAL2LABの使い方	7
1.4 自動生成ラベルの編集方法	9
第2章 音声合成エンジン NNSVS	14
2.1 NNSVSについて	14
2.2 環境構築：WSL2（UBUNTU）のセットアップ	15
2.3 NNSVSのセットアップ	25
2.4 NNSVSの動作確認	30
2.5 NNSVSのカスタマイズ	33
第3章 音声&楽譜データの作成方法	43
3.1 音声データ作成方法	43
3.2 楽譜データ作成方法	54

第1章 VOCAL2LAB

1. 1 VOCAL2LABについて

NNSVS 向けの教師データ作成を支援する自動ラベリングツール。任意の楽曲を NNSVS に歌わせる為のツール Music2lab も同梱。

動作には以下の環境が必要

- Windows
- Python3
- Perl

GitHub で公開中 : <https://github.com/148nasuka/Vocal2lab>

ディレクトリ構造 /Vocal2lab

📁 Data_in	自動ラベリング入力
📁 Data_out	自動ラベリング出力
📁 Julius	音声認識エンジン
📁 Music2lab	楽譜ラベル生成ツール(任意の曲を歌わせる時に使用)
📁 Setup	初回セットアップツール
📁 temp	一時フォルダ
🐍 help.py	
🐍 MultiExecution.py	
🐍 Vocal2lab.py	

※セットアップ完了後のディレクトリ構造

ディレクトリ構造 /Vocal2lab/Julius

bin	エンジン本体(コンパイル済み)
log	デバッグ用ログ
models	学習済みモデル
wav	入力フォルダ
License.md	
README.md	
segment_julius.pl	

※セットアップ完了後のディレクトリ構造

ディレクトリ構造 /Vocal2lab/Music2lab

lab_out	楽譜ラベル出力
xml_in	楽譜データ入力
music2lab.py	

※セットアップ完了後のディレクトリ構造

1.2 VOCAL2LAB のセットアップ

Strawberry Perl のインストール方法と Vocal2lab のセットアップ方法

手順1：「Strawberry Perl のダウンロード」

(1/5)

The Perl for MS Windows, free of charge!



'When I'm on Windows, I use Strawberry Perl!'
-- Larry Wall

Perl is a programming language suitable for writing simple scripts as well as complex applications – see <https://www.perl.org>.

Strawberry Perl is a perl environment for MS Windows containing all you need to run and develop perl applications. It is designed to be as close as possible to perl environment on UNIX systems.

It includes perl binaries, compiler (gcc) + related tools, all the external libraries (crypto, math, graphics, xml...), all the bundled database clients and all you expect from Strawberry Perl.

Recommended version:

[strawberry-perl-5.32.1.1-64bit.msi](#)
[strawberry-perl-5.32.1.1-32bit.msi](#)

More downloads (all releases):

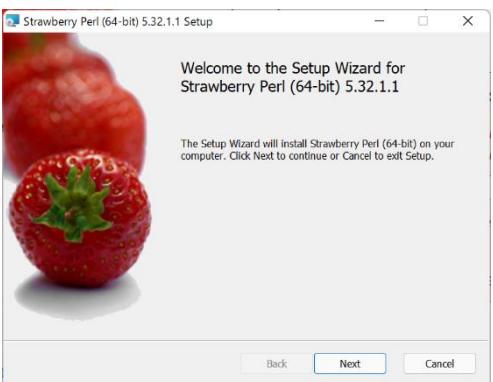
[ZIP, Portable, special editions](#)
You can find here release notes and other details.

<https://strawberryperl.com/>

上記リンクからダウンロードサイトに移動し、
Windows64bit 版をダウンロードする

手順2：「Strawberry Perl のインストール」

(2/5)



The Setup Wizard will install Strawberry Perl (64-bit) on your computer. Click Next to continue or Cancel to exit Setup.

Back Next Cancel

ダウンロードしたインストーラに従ってインストールを完了する

手順3：「Perl環境の確認」

(3/5)

Windows PowerShellで実行

```
perl -v
```

上記コマンドでPerlのバージョンが表示されれば
導入完了

手順4：「Vocal2labのダウンロード」

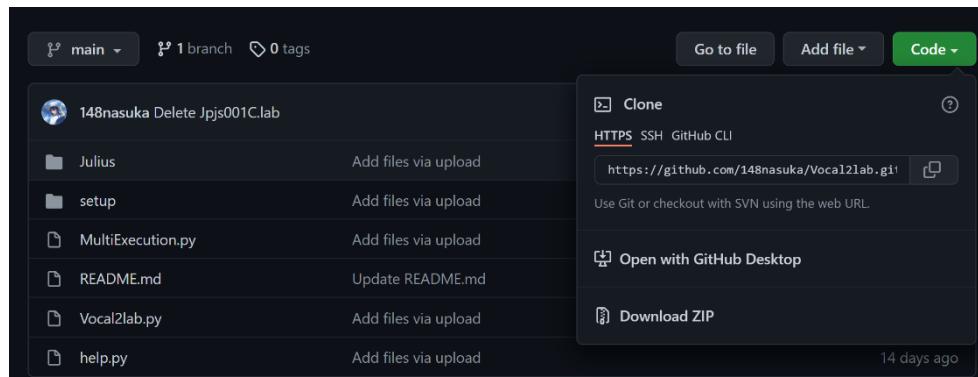
(4/5)

Windows PowerShellで実行

```
cd C:\Users\[ユーザー名]\Downloads
```

```
git clone https://github.com/148nasuka/Vocal2lab.git
```

上記コマンドでWindowsのダウンロードフォルダへ移動し、
Vocal2labをダウンロードする



※GitHubからダウンロードしてくることもできる
<https://github.com/148nasuka/Vocal2lab>

手順5：「Vocal2labのセットアップ」

(5/5)

Windows PowerShellで実行

```
cd ./Vocal2lab/Setup  
python ./setup.py
```

上記コマンドでセットアップフォルダへ移動し、

管理者権限でスクリプトを実行する

(必須Pythonライブラリのインストールと音素辞書の確認を行う為)

※venv仮想環境の場合、自動セットアップができない為
手動で以下のライブラリをインストールする必要がある

- numpy==1.20.0
- pysinsy
- librosa
- SoundFile

また、デフォルトのpysinsyに音素辞書が無いため
/Vocal2lab/Setup/Sinsy_dicの辞書ファイルを

Pysinsyインストール先のディレクトリに_dicディレクトリを作成してコピーする

```
*****  
pysinsy 音素辞書ファイルの確認をします  
*****  
  
pysinsy インストール先 : C:\Users\148na\AppData\Roaming\Python\Python38\site-packages\pysinsy\_dic  
japanese.euc_jp.conf : check  
japanese.euc_jp.table : check  
japanese.shift_jis.conf : check  
japanese.shift_jis.table : check  
japanese.utf_8.conf : check  
japanese.utf_8.table : check  
  
*****  
インストール完了  
*****  
  
PS C:\Users\148na\Documents\学校\8\研究\Vocal2lab\Setup> |
```

自動セットアップ完了時は上記のような出力になる

1.3 VOCAL2LAB の使い方

手順 1：「Data_in フォルダに楽譜と音声ファイルを入れる」 (1/3)

名前	更新日時	種類	サイズ
pjs001.musicxml	2021/09/30 7:29	MUSICXML ファイル	29 KB
pjs001.wav	2021/09/30 7:29	WAV ファイル	2,254 KB
pjs002.musicxml	2021/09/30 7:29	MUSICXML ファイル	37 KB
pjs002.wav	2021/09/30 7:29	WAV ファイル	2,817 KB
pjs003.musicxml	2021/09/30 7:29	MUSICXML ファイル	25 KB
pjs003.wav	2021/09/30 7:29	WAV ファイル	1,845 KB
pjs004.musicxml	2021/09/30 7:29	MUSICXML ファイル	23 KB
pjs004.wav	2021/09/30 7:29	WAV ファイル	2,029 KB
pjs005.musicxml	2021/09/30 7:29	MUSICXML ファイル	36 KB
pjs005.wav	2021/09/30 7:29	WAV ファイル	2,784 KB

対になる楽譜と音声ファイルは同じ名前にしておく

- ※音声ファイルは
・モノラル 48kHz/16bit
・モノラル 96kHz/16bit
のいずれかに統一する事

手順 2：「Vocal2lab を実行」 (2/3)

Windows PowerShell で実行

```
python ./Vocal2lab.py [入力ファイル名] [出力ファイル名]  
Data_in フォルダ内から一つだけラベリングをする場合
```

Windows PowerShell で実行

```
python ./Vocal2lab.py --multi  
Data_in フォルダ内すべてに対してラベリングをする場合
```

Windows PowerShell で実行

```
python ./Vocal2lab.py --multi ds  
Data_in フォルダ内すべてに対してラベリングかつ、  
出力音声を 48kHz にそろえる場合
```

Windows PowerShell で実行

```
python ./Vocal2lab.py --help  
※コマンドリストを確認することも可能
```

手順3：「出力データの確認」

(3/3)

📁 2022-02-07_21-31-36	2022/02/07 21:31	ファイル フォルダー
📁 2022-02-07_21-36-02	2022/02/07 21:36	ファイル フォルダー
📁 2022-02-07_21-39-43	2022/02/07 21:39	ファイル フォルダー
📁 error	2022/01/08 0:21	ファイル フォルダー
📄 pjs005.lab	2022/02/21 20:29	LAB ファイル

一つだけラベリングした場合 Data_out フォルダにラベルだけ出力される

📁 pjs001	2022/02/07 21:39	ファイル フォルダー
📁 pjs002	2022/02/07 21:39	ファイル フォルダー
📁 pjs003	2022/02/07 21:39	ファイル フォルダー
📁 pjs004	2022/02/07 21:39	ファイル フォルダー
📁 pjs005	2022/02/07 21:39	ファイル フォルダー

まとめてラベリングした場合、日付のフォルダが Data_out に生成され、その中には楽曲ごとに各データが分けられる

※日付フォルダを丸ごと NNSVS の教師データとして与えることが可能

1.4 自動生成ラベルの編集方法

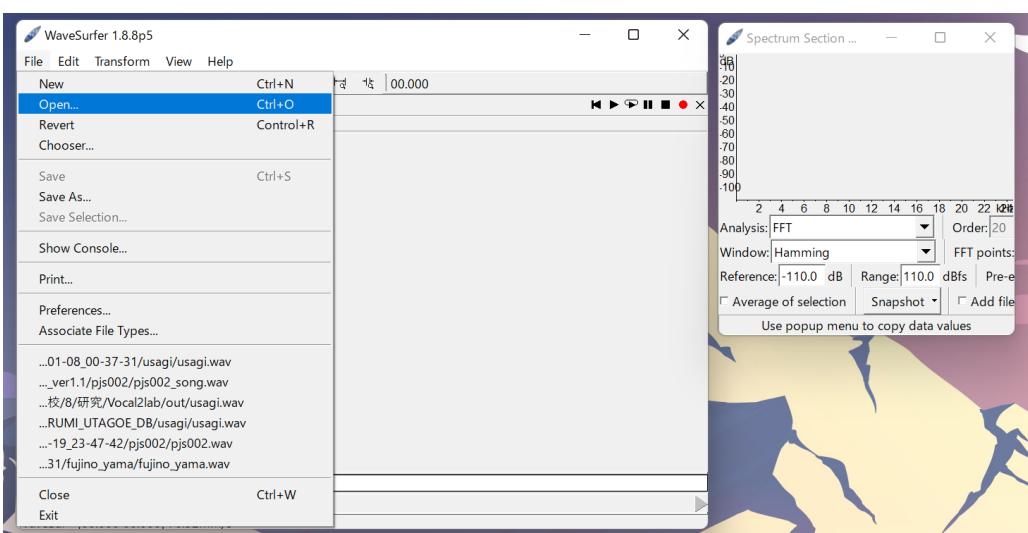
自動生成ラベルは音声波形表示ソフト Wavesurfer で編集することが可能

ソフトウェアは下記リンクからダウンロードして圧縮ファイルを展開すると実行可能

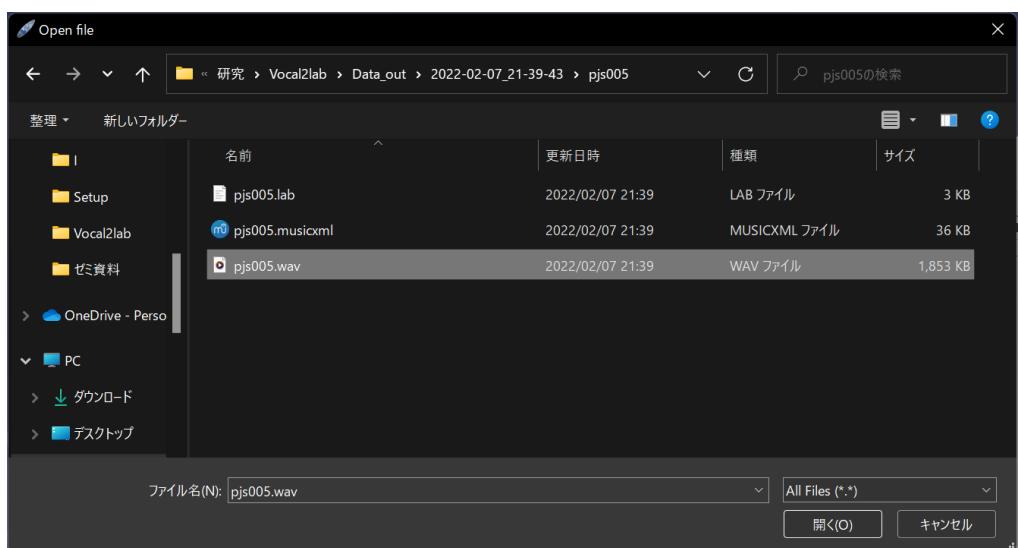
Wavesurfer : <https://sourceforge.net/projects/wavesurfer/>

ラベル編集手順 1：「音声を読み込む」

(1/6)



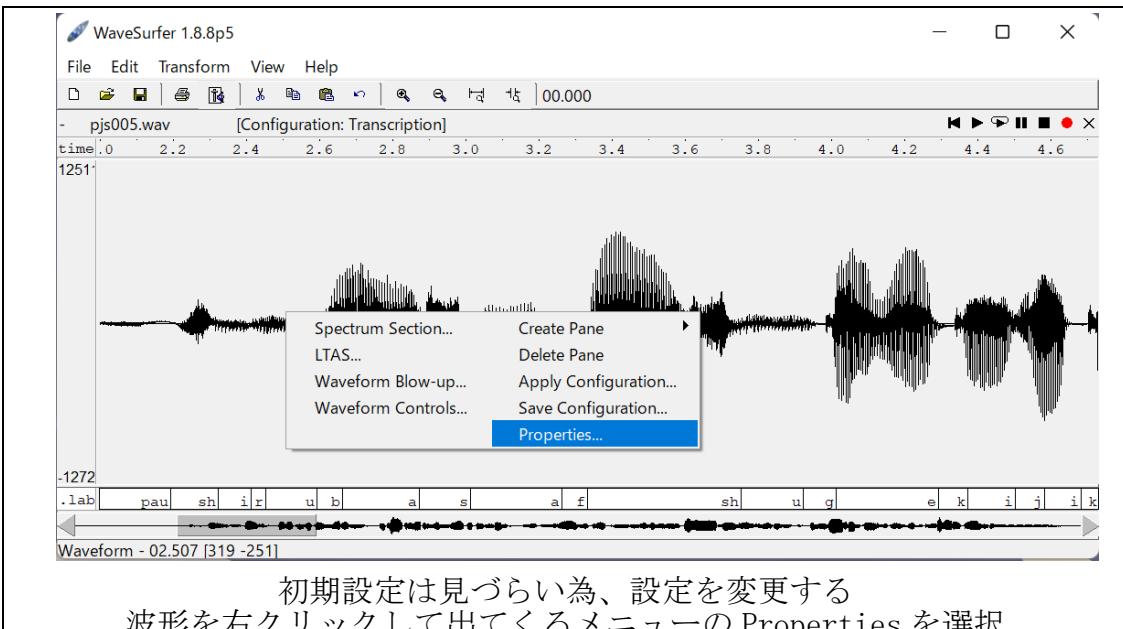
左上の File メニューの open を選択



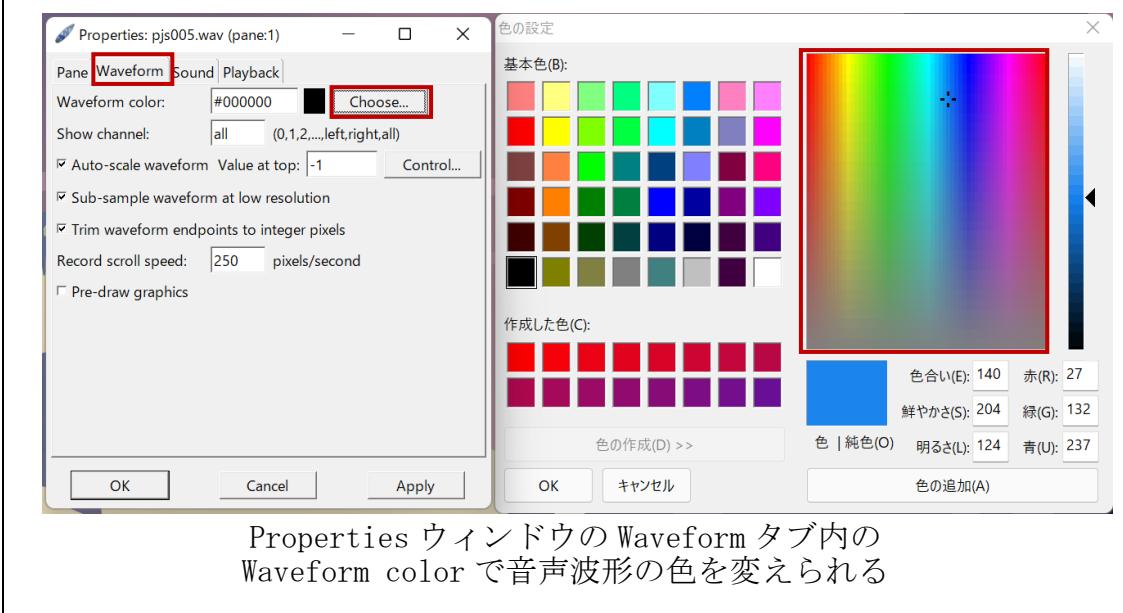
Vocal2lab の Data_out 内の日付フォルダ内から
任意の音源を選択する

ラベル編集手順 2：「音声波形を見やすくする」

(2/6)



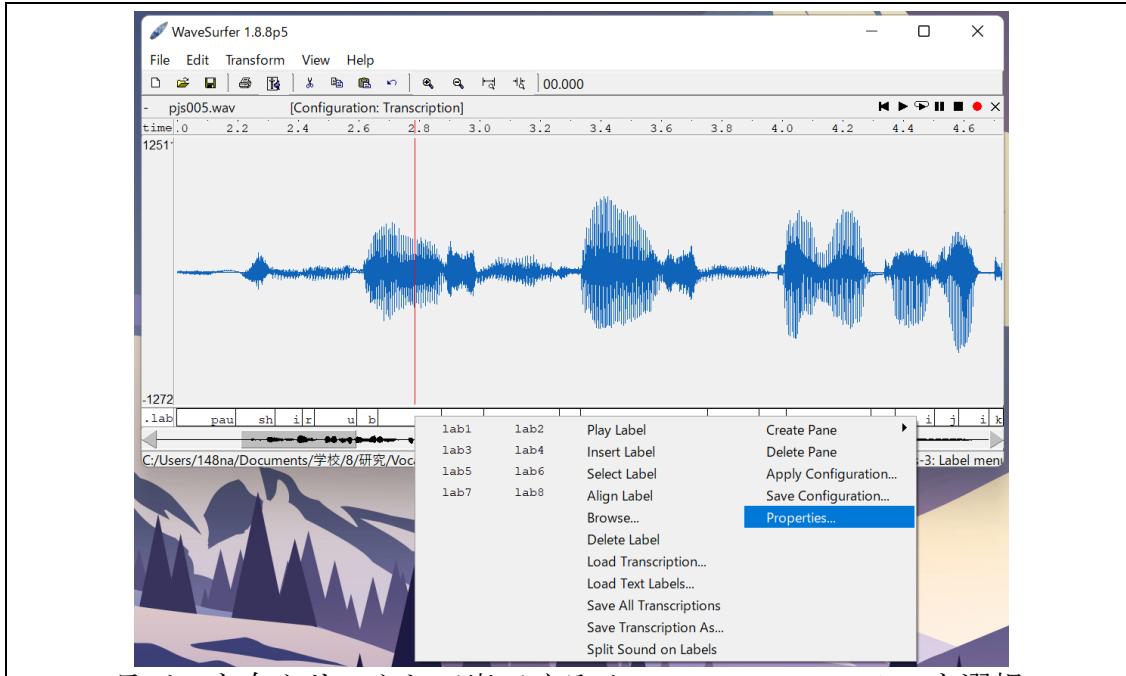
初期設定は見づらい為、設定を変更する
波形を右クリックして出てくるメニューの Properties を選択



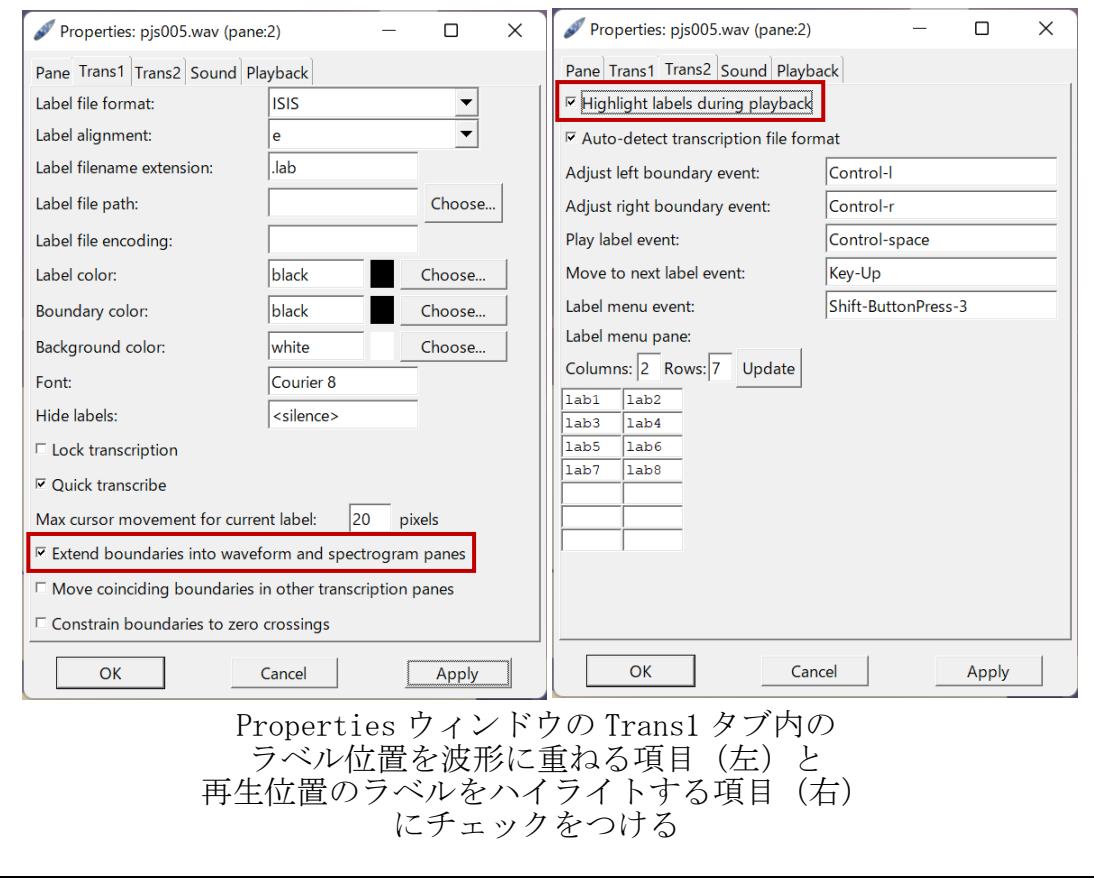
Properties ウィンドウの Waveform タブ内の
Waveform color で音声波形の色を変えられる

ラベル編集手順 3：「ラベル表示を見やすくする」

(3/6)



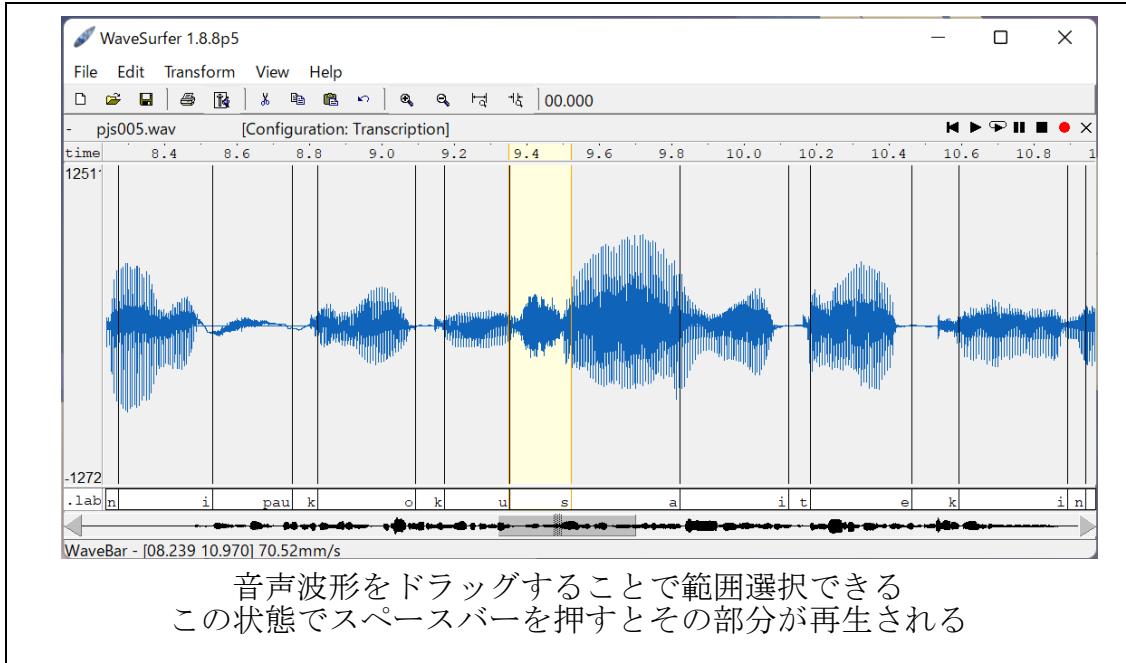
ラベルを右クリックして出てくるメニューの Properties を選択



Properties ウィンドウの Trans1 タブ内の
ラベル位置を波形に重ねる項目（左）と
再生位置のラベルをハイライトする項目（右）
にチェックをつける

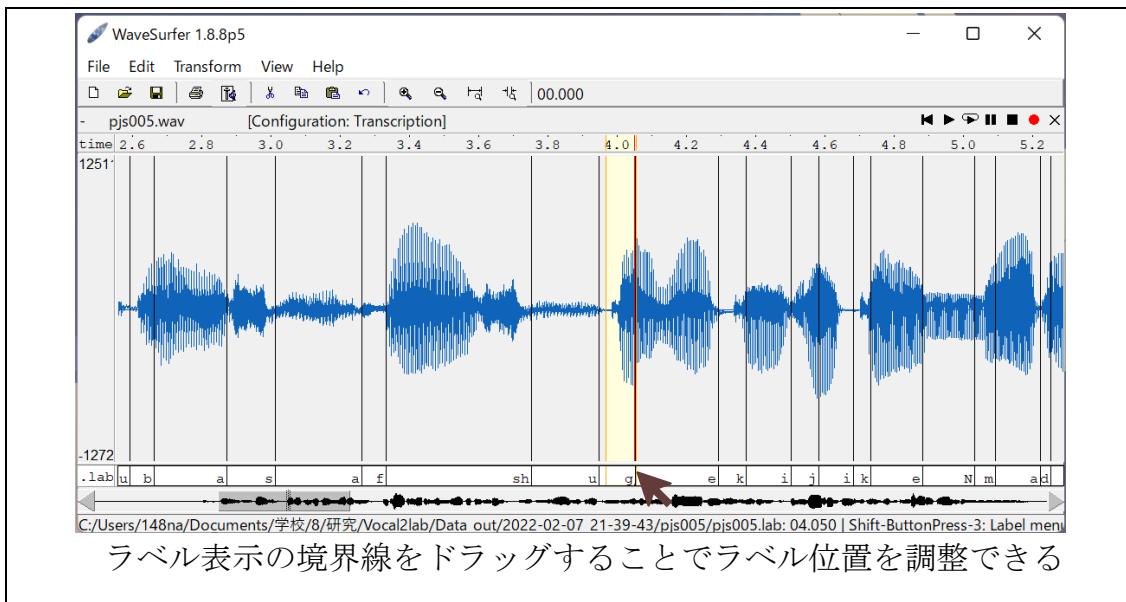
ラベル編集手順 4：「ラベル位置と音声を確認する」

(4/6)



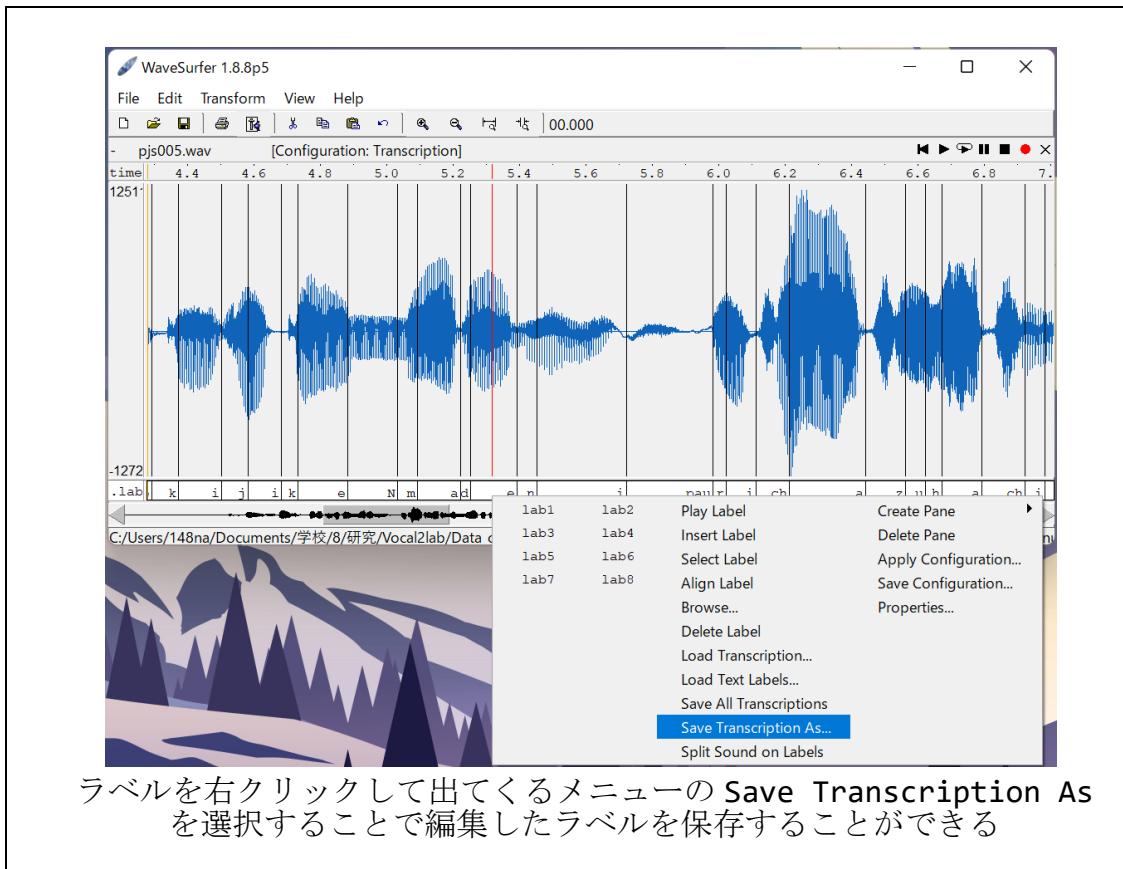
ラベル編集手順 5：「ラベル位置を変える」

(5/6)



ラベル編集手順 6：「ラベルの保存」

(6/6)



第2章 音声合成エンジン NNSVS

2.1 NNSVSについて

NNSVS (Neural network-based singing voice synthesis) はオープンソースの歌声合成エンジン。音素形式は Sinsy のものと共通となる。

参考

- NNSVS 開発者ブログ : <https://r9y9.github.io/blog/2020/05/10/nnsvs/>
- NNSVS (GitHub リンク) : <https://github.com/r9y9/nnsvs>
- Sinsy : <https://www.sinsy.jp/>
- Sinsy 音素リスト : <https://sinsy.sp.nitech.ac.jp/reference.pdf>

このエンジンを動作させるには Linux 環境が必要であり、Python ライブラリの手動インストールも必要になるので、その手順を以下の節にてステップバイステップで説明する。尚、このチュートリアルでは以下の環境で行う。

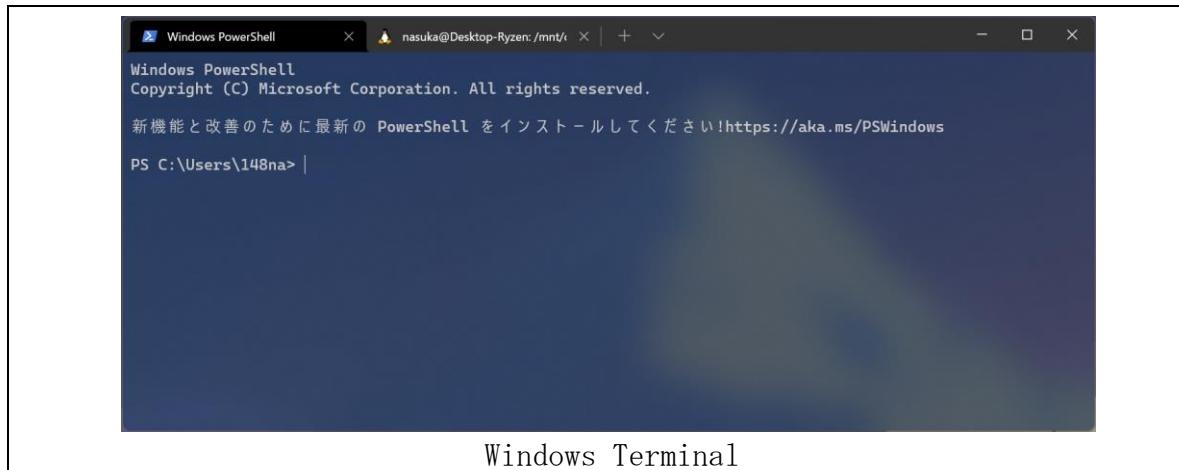
OS	Windows11
CPU	AMD Ryzen7 3700X
GPU	Nvidia RTX2060 6GB
RAM	32GB

2.2 環境構築：WSL2 (UBUNTU) のセットアップ

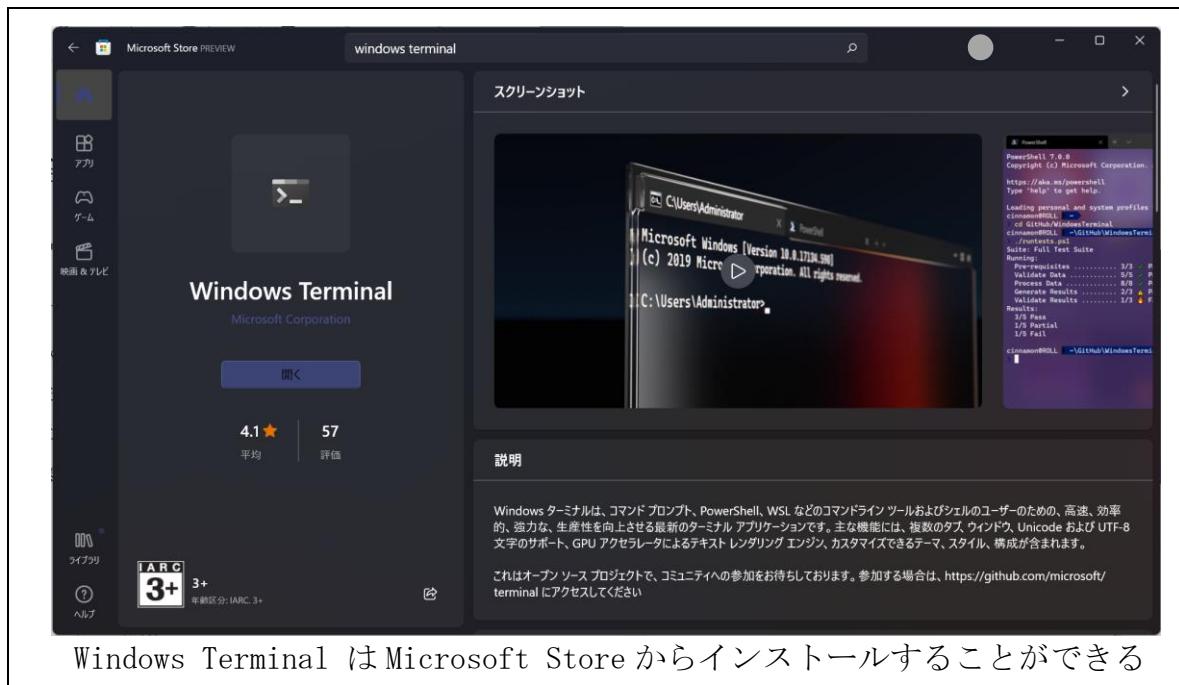
① Windows Terminal の導入

利便性の為、Ubuntu ターミナルと Windows のシェルをまとめて扱える

「Windows Terminal」の利用をお勧めする。



Windows Terminal



Windows Terminal は Microsoft Store からインストールすることができる

詳細な利用方法は以下の記事を参考にすると良い

4thsight.xyz 記事：<https://4thsight.xyz/9087>

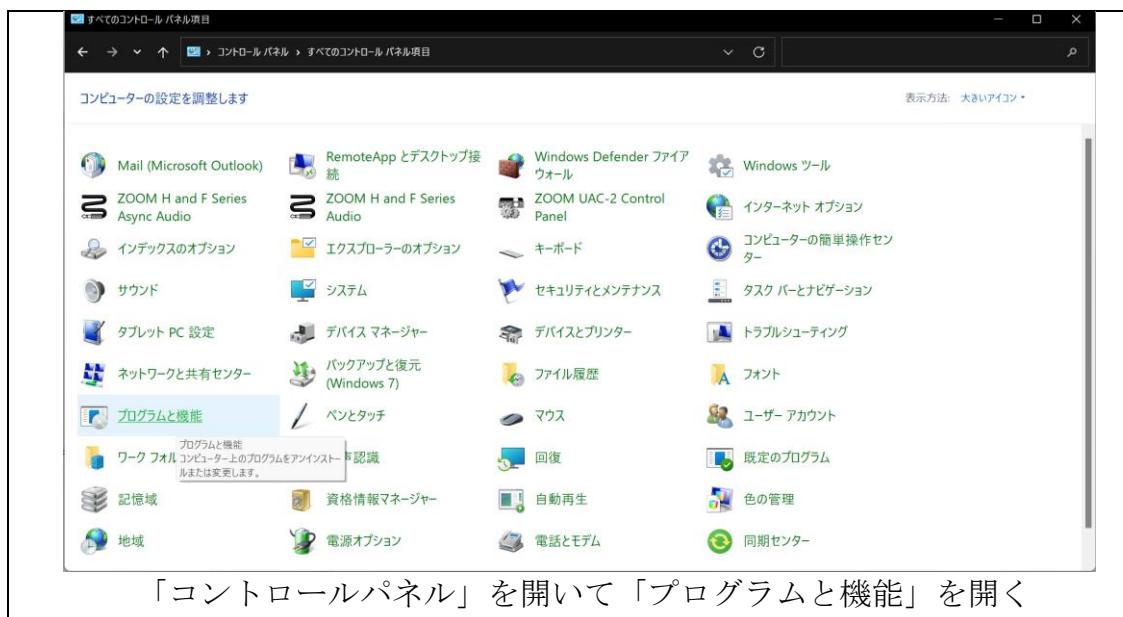
② WSL2 の有効化

Windows 環境で CUDA が利用できる Ubuntu を動かすには高パフォーマンスな仮想環境 WSL2 が必須である。②では WSL の導入の流れを説明する。

※Windows10 ビルド 2004 以降の場合は手順②-1 から手順②-4 をスキップしても問題ない。

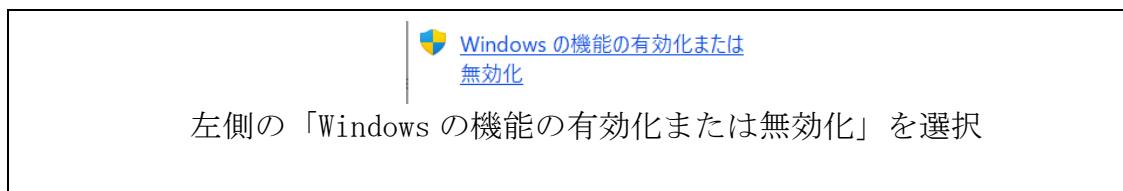
手順②-1：「Windows の設定変更」

(1/6)



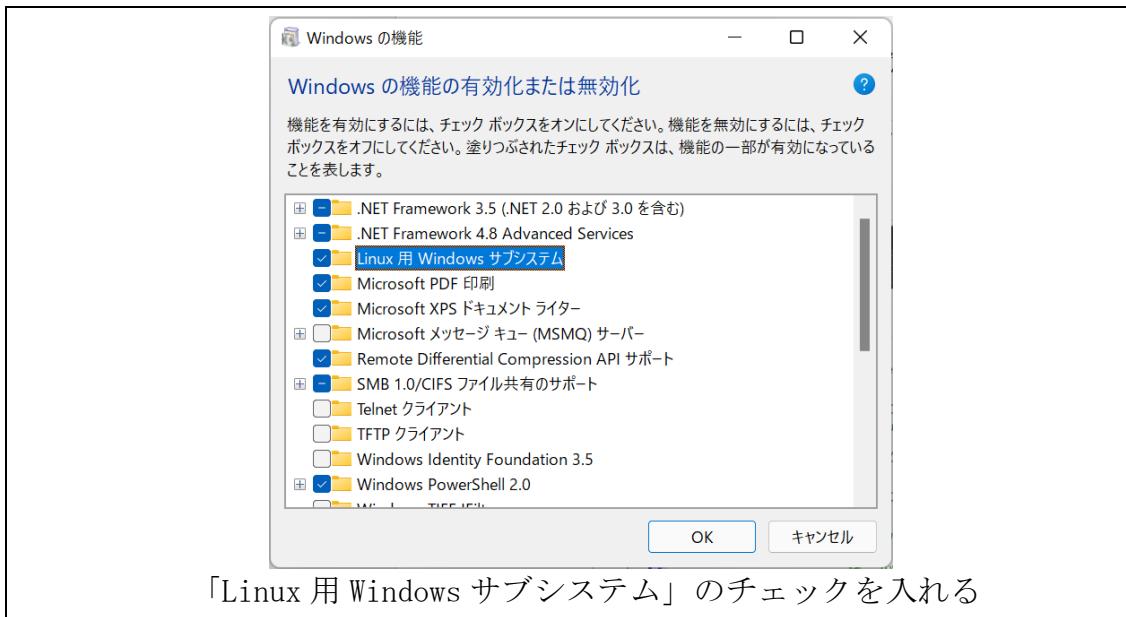
手順②-2：「Windows の設定変更」

(2/6)



手順②-3：「Windows の設定変更」

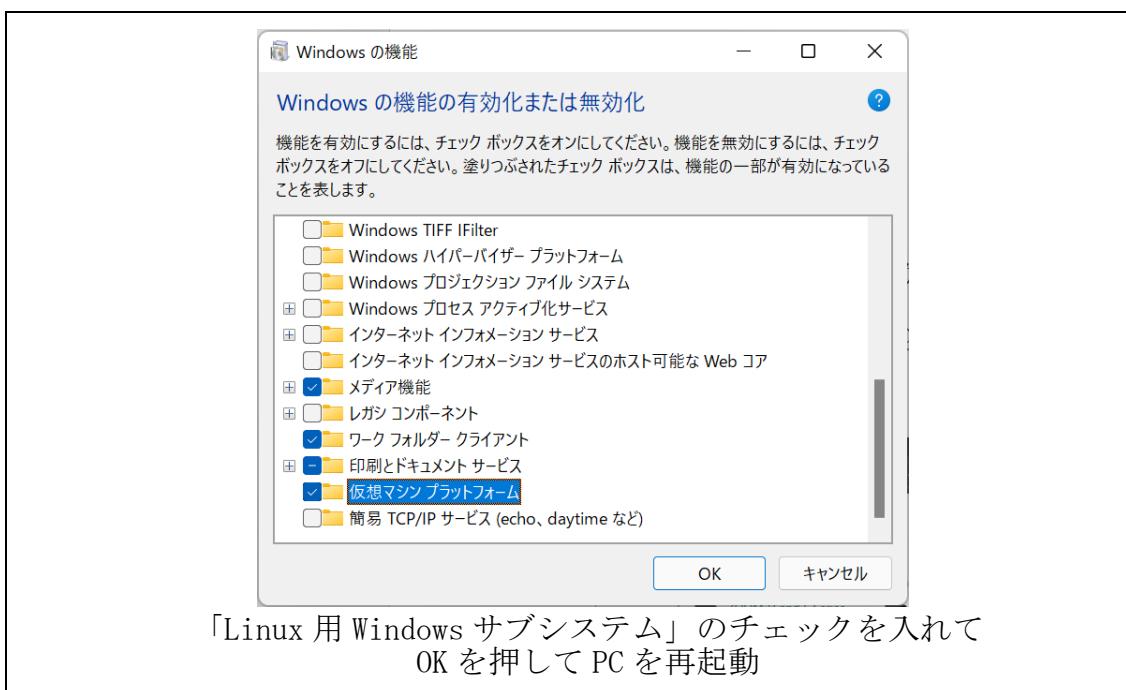
(3/6)



「Linux用Windowsサブシステム」のチェックを入れる

手順②-4：「Windows の設定変更」

(4/6)



「Linux用Windowsサブシステム」のチェックを入れて
OKを押してPCを再起動

手順②-5：「WSL のインストール」

(5/6)

Windows PowerShell で実行

```
wsl -install
```

Windows Terminal を管理者権限で開いて上記のコマンドで
WSL をインストールする

手順②-6：「WSL のインストール」

(6/6)

Windows PowerShell で実行

```
wsl --set-default-version 2
```

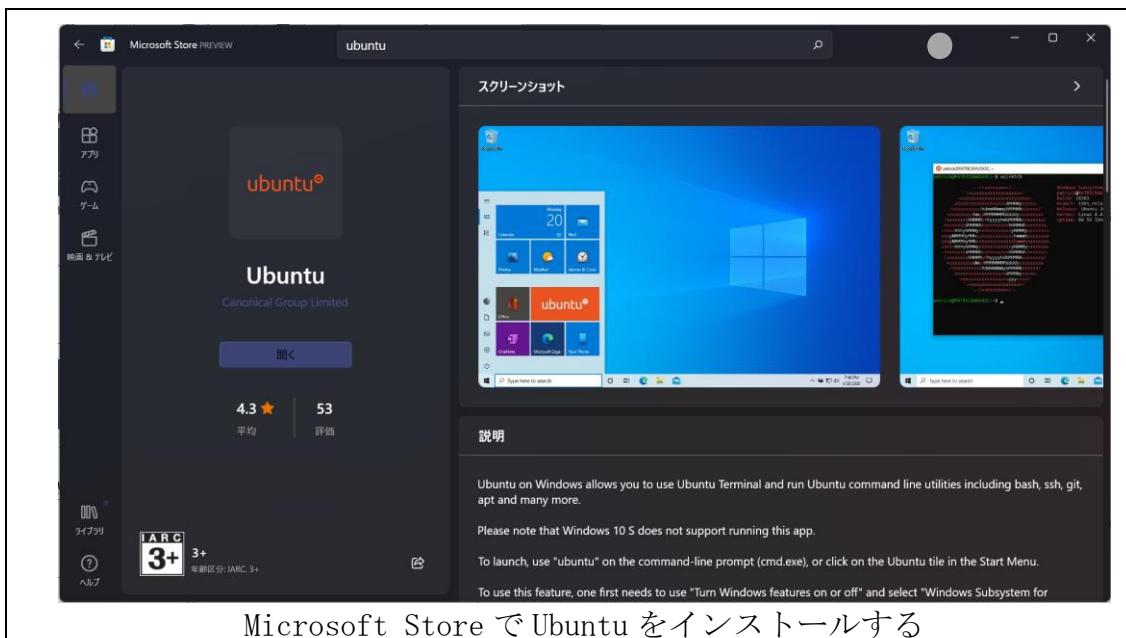
WSL のバージョンを上記のコマンドで
2 に固定する

③ Ubuntu のインストール&セットアップ

有効化した WSL 仮想環境に Ubuntu をインストールしていく。

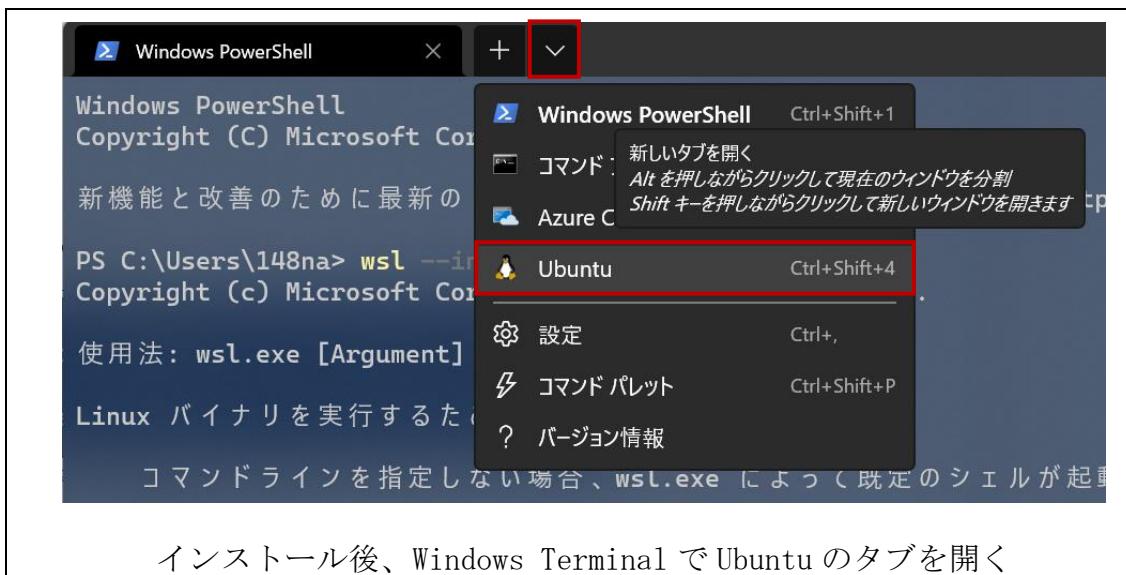
手順③-1：「Ubuntu のダウンロード」

(1/7)



手順③-2：「Ubuntu のインストール」

(2/7)



手順③-3：「Ubuntu のインストール」

(3/7)

```
Installing, this may take a few minutes...
Ubuntu のインストールが始まるので待つ
```

手順③-4：「Ubuntu の初期設定」

(4/7)

```
Please create a default UNIX user account. The username does not need
to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: [ユーザー名]
Enter new UNIX password: [パスワード]
Retype new UNIX password:[パスワード（もう一回）]
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo".
See "man sudo_root" for details.
```

Ubuntu のユーザー名とパスワードを設定する
※忘れないようにメモしておくと良い

手順③-5：「Ubuntu の更新」

(5/7)

```
Ubuntu Terminal で実行
sudo apt-get update
```

インストールが終わったら、上記のコマンドを入力して
Ubuntu の更新の確認をする

(補足) 更新の確認に失敗する場合

```
Ubuntu Terminal で実行
sudo rm /etc/resolv.conf
sudo sh -c "echo 'nameserver 8.8.8.8' > /etc/resolv.conf"
```

上記のコマンドを入力して
Ubuntu のネットワーク設定を修正する

手順③-6：「Ubuntu の更新」

(6/7)

Ubuntu Terminal で実行

```
sudo apt-get upgrade
```

インストールが終わったら、上記のコマンドを入力して
Ubuntu の更新を実行する

```
...
```

```
3 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

```
17 standard security updates
```

```
Need to get 17.2 MB of archives.
```

```
After this operation, 167 kB of additional disk space will be used.
```

```
Do you want to continue? [Y/n] Y
```

アップデートの確認が出たら「Y」を入力

手順③-7：「Ubuntu のシャットダウン」

(7/7)

Windows PowerShell で実行

```
wsl --shutdown
```

アップデートが終わったら、上記のコマンドを入力して
Ubuntu を終了し、Ubuntu タブも閉じる

④ WSL-Ubuntu 用の CUDA 環境構築

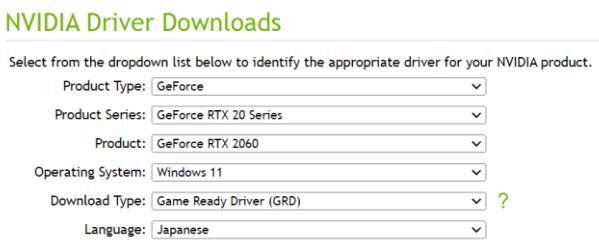
WSL で GPU 处理をするために専用のドライバーキットをインストールする。

ドライバのインストール時は WSL-Ubuntu はシャットダウンしておく。

※WSL で GPU を使うには Windows10 バージョン 21H2 以降である必要がある

(NNSVS を CPU だけで動作させる場合はこの手順を省略してもよい。)

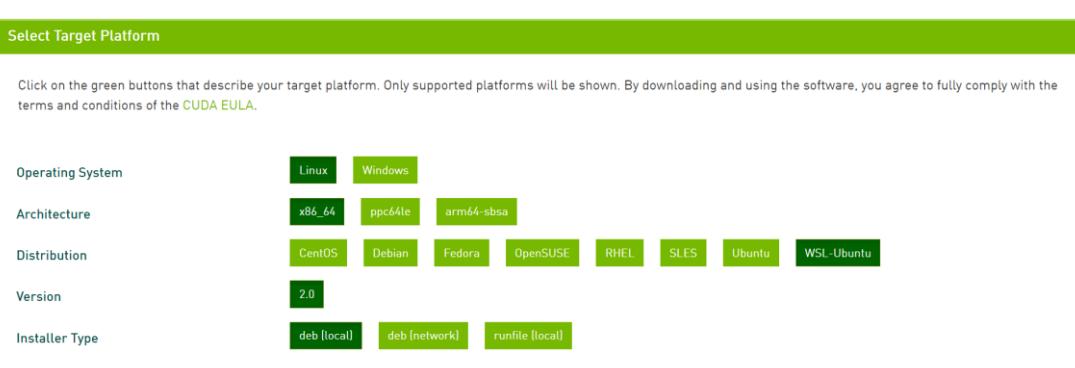
手順④-1：「CUDA ドライバのインストール」 (1/4)



自分の環境にあった方をダウンロードして
インストール後、PC の再起動をする
<https://www.nvidia.com/download/index.aspx>

※Windows 用のみをインストールする事

手順④-2：「WSL 用 CUDA Toolkit のインストール」 (2/4)



下記リンクから
Linux, x86_64, WSL-Ubuntu, 2.0, deb(local)
になっていることを確認する。

https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=WSL-Ubuntu&target_version=2.0&target_type=deb_local

手順④-3：「WSL 用 CUDA Toolkit のインストール」

(3/4)

The screenshot shows the 'Base Installer' section of the CUDA Toolkit download page. It contains a box with terminal commands for installing CUDA:

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/cuda-wsl-ubuntu.pin  
$ sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin-600  
$ wget https://developer.download.nvidia.com/compute/cuda/11.6.0/local_installers/cuda-repo-wsl-ubuntu-11.6-local_11.6.0-1_amd64.deb  
$ sudo dpkg -i cuda-repo-wsl-ubuntu-11.6-local_11.6.0-1_amd64.deb  
$ sudo apt-key add /var/cuda-repo-wsl-ubuntu-11.6-local/7fa2af80.pub  
$ sudo apt-get update  
$ sudo apt-get -y install cuda
```

Below the commands, there is a note about the CUDA Toolkit containing open-source software and links for source code, checksums, and installation guides.

確認後、サイト下部に表示されるコマンド列をコピーして Ubuntu タブを開き直して実行

手順④-4：「CUDA の確認」

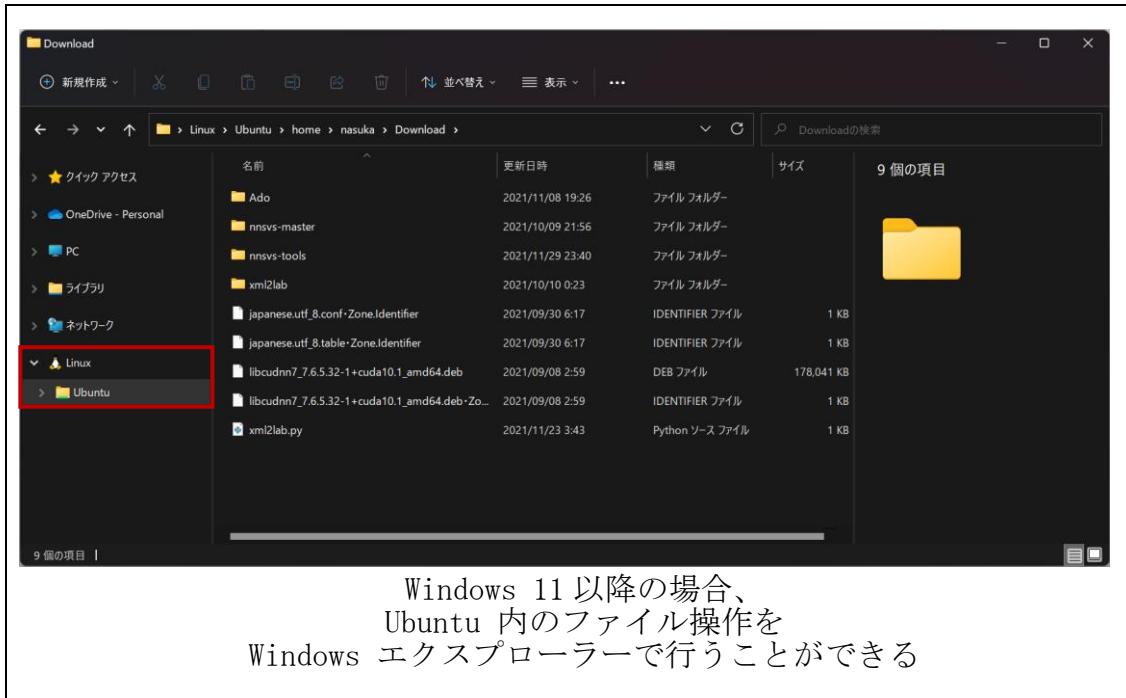
(4/4)

The screenshot shows an Ubuntu terminal window with the title 'Ubuntu Terminal' and the command 'nvidia-smi' entered. The output of the command is displayed, showing GPU information and process details.

```
nasuka@Desktop-Ryzen:/mnt/c/Users/148na$ nvidia-smi  
Thu Feb  3 03:12:50 2022  
+-----+  
| NVIDIA-SMI 510.00      Driver Version: 510.06      CUDA Version: 11.6 |  
+-----+  
| GPU  Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC | | | | | |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |  
|          |          |          |          |          |          |          | MIG M. |  
+-----+-----+-----+-----+-----+-----+-----+-----+  
| 0  NVIDIA GeForce ...  On  | 00000000:07:00.0  On  |                  N/A | | | | | |
| 24%   50C    P0    39W / 160W | 4371MiB /  6144MiB |  N/A       Default |  
|          |          |          |          |          |          |          | N/A  |  
+-----+-----+-----+-----+-----+-----+-----+-----+  
  
+-----+  
| Processes:  
|  GPU  GI  CI      PID  Type  Process name          GPU Memory  
|        ID  ID                    Usage          |  
+-----+  
| No running processes found  
+-----+  
nasuka@Desktop-Ryzen:/mnt/c/Users/148na$
```

導入に成功していれば上記のように GPU 情報が表示される

便利情報



参考

- CUDA on WSL2 の速度比較と環境構築 :

<https://qiita.com/SwitchBlade/items/cac2da388906b6486d69>

- 【Windows10】WSL2 の有効化と Ubuntu のインストール方法 :

<https://algorithm.joho.info/unix/windows-subsystem-for-linux-install/>

- WSL2 の Ubuntu に CUDA をインストールして PyTorch を動かす :

<https://takake-blog.com/wsl-nvidia-cuda/>

2.3 NNSVS のセットアップ

① Python 環境の下準備

C/C++コンパイル環境と Python、C/C++を相互に使用するためのライブラリ「Cython」をインストールする。

※NNSVS は venv 仮想環境下で動かないで直接インストールする事

Ubuntu Terminal で実行

```
sudo apt-get install libssl-dev libffi-dev
```

上記のコマンドで追加のパッケージをインストール

Ubuntu Terminal で実行

```
sudo apt-get install python3-dev python3-pip
```

上記コマンドで追加の Python 環境をインストール

Ubuntu Terminal で実行

```
python3 -V
```

上記コマンドで python3 のバージョンを確認

Ubuntu Terminal で実行

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.X 0
```

上記コマンドでデフォルトの Python を設定する

※/usr/bin/python3.X の部分はインストール済みの Python に合わせる

Ubuntu Terminal で実行

```
python -V
```

上記コマンドで Python3 が呼び出されれば設定完了

Ubuntu Terminal で実行

```
pip install cython
```

```
pip install numpy==1.20.0
```

上記コマンドで Cython, numpy をインストール

② Pytorch のインストール

※Pytorch の前提ライブラリ numpy 等のインストール説明は省略する。

手順②-1：「Pytorch のインストール」

(1/2)

The screenshot shows the PyTorch download page with the following configuration:

- PyTorch Build: Stable (1.10.2) is selected.
- Your OS: Linux is selected.
- Package: Pip is selected.
- Language: Python is selected.
- Compute Platform: CUDA 10.2 is selected.
- Run this Command: pip3 install torch==1.10.2+cu113 torchvision==0.11.3+cu113 torchaudio==0.2+cu113 -f https://download.pytorch.org/whl/cu113/torch_stable.html

以下のようにコマンドを実行する

以下のリンクから
Stable, Linux, Pip, Python, CUDA 11.X
と、なっていることを確認してコマンドをコピーして
Ubuntu ターミナルで実行する

<https://pytorch.org/>

手順②-2：「Pytorch の確認」

(2/2)

Ubuntu Terminal で実行

```
python -c "import torch; print(torch.cuda.is_available())"
```

上記のコマンドを実行することで
GPU が利用可能か確認できる

```
nasuka@Desktop-Ryzen:/mnt/c/Users/148na$ python3 -c "import torch; print(torch.cuda.is_available())"
True
nasuka@Desktop-Ryzen:/mnt/c/Users/148na$ |
```

GPU の認識に成功している場合
「True」 が返される

③ nnmnkwi のインストール

nnmnkwi は NNSVS での音声合成に必要な計算やパラメータの橋渡し用のライブラリである。Git からの手動インストールが必要。

手順③-1：「nnmnkwi のインストール」 (1/2)

Ubuntu Terminal で実行

```
pip install git+https://github.com/r9y9/nnmnkwi.git --system
```

上記コマンドで nnnkwi をインストール

手順③-1：「nnmnkwi の確認」 (2/2)

Ubuntu Terminal で実行

```
python -c "import nnnkwi"
```

上記コマンドでエラーが出なければ成功

参考

- nnnkwi (Github) : <https://github.com/r9y9/nnmnkwi>

④ Pysinsy のインストール

音声合成エンジン Sinsy の Python 移植版。NNSVS に使用する場合は Git からの手動インストールが必要。

手順④-1：「pysinsy のインストール」

(1/2)

Ubuntu Terminal で実行

```
pip install git+https://github.com/r9y9/pysinsy.git --system
```

上記のコマンドで pysinsy をインストール

手順④-2：「pysinsy のインストール先の確認」

(2/3)

Ubuntu Terminal で実行

```
python -c "import pysinsy; print(pysinsy.__file__)"
```

pysinsy のインストール先を確認

出力例

```
/mnt/c/Users/148na/pysinsy/pysinsy/__init__.py
```

手順④-3：「Sinsy 音素辞書のインストール」

(3/3)

Ubuntu Terminal で実行

```
cd /mnt/c/Users/148na/pysinsy/pysinsy/
cp -r /Vocal2lab/Setup/Sinsy_dic/ ._dic/
```

Vocal2lab/Setup/Sinsy_dic/ の音素辞書フォルダを
pysinsy インストール先へ _dic という名前でコピー

参考

- pysinsy (GitHub) : <https://github.com/r9y9/pysinsy>

⑤ NNSVS のインストール

ニューラルネットワークを利用した音声合成エンジン。内部の編集をしながら扱う Python ライブライの為、これも Git から手動インストールをする。

手順⑤-1：「NNSVS のインストール」

(1/2)

Ubuntu Terminal で実行

```
cd ~  
git clone https://github.com/r9y9/nnsvs.git  
cd nnsvs  
sudo python setup.py develop
```

上記コマンドで NNSVS をインストール
※インストールに時間がかかる

手順⑤-2：「NNSVS の確認」

(2/2)

Ubuntu Terminal で実行

```
python -c "import nnsvs"
```

上記のコマンドでエラーが出なければ成功

参考

- NNSVS (GitHub) : <https://github.com/r9y9/nnsvs>

2.4 NNSVS の動作確認

この節では動作確認のために PJS を利用した音声合成をする。

手順 1：「NNSVS インストールディレクトリを開く」

(1/9)

Ubuntu Terminal で実行

```
cd ~/nnsvs/egs/pjs/svs-world-conv/
```

ターミナルで NNSVS 内の pjs ディレクトリに移動する
(前節 1.3-⑤)の例に従った場合は上記のディレクトリになる)

手順 2：「pjs のダウンロード」

(2/9)

Ubuntu Terminal で実行

```
sudo apt-get install unzip  
sudo bash run.sh --stage -1 --stop-stage -1
```

上記のコマンドで zip 解凍パッケージのインストールと
pjs のダウンロードを行う

手順 3：「学習の前処理」

(3/9)

Ubuntu Terminal で実行

```
sudo bash run.sh --stage 0 --stop-stage 0
```

上記コマンドで
pjs 内から訓練用データ（学習に使う部分）と
検証データ（学習の調整に使う部分）と
テストデータ（学習の評価に使う部分）に分割

※詳しくは 1.5 節で説明

手順 4：「特徴量の抽出」

(4/9)

Ubuntu Terminal で実行

```
sudo bash run.sh --stage 1 --stop-stage 1
```

上記コマンドで特徴量の抽出を行う
(負荷が高い)

手順5：「タイムラグ（発音タイミング）の学習」

(5/9)

Ubuntu Terminal で実行

```
sudo bash run.sh --stage 2 --stop-stage 2
```

上記コマンドでタイムラグ（発音タイミング）の学習を行う

手順6：「音素継続長（子音と母音のタイミング）の学習」

(6/9)

Ubuntu Terminal で実行

```
sudo bash run.sh --stage 3 --stop-stage 3
```

上記コマンドで音素継続長（子音と母音のタイミング）の学習を行う

/nnsvs/nnsvs/bin/train.py 25,26 行目

```
logger = None
```

```
use_cuda = False
```

※途中で止まってしまう場合

上記コードで CPU 处理にすることで改善できる

手順7：「音響特徴量の学習」

(7/9)

Ubuntu Terminal で実行

```
sudo bash run.sh --stage 4 --stop-stage 4
```

上記コマンドで音響特徴量の学習を行う
(負荷が高い)

/nnsvs/nnsvs/bin/train.py 26 行目

```
use_cuda = torch.cuda.is_available()
```

※手順6で CPU 处理に切り替えた場合は
GPU 处理に戻すことで処理を高速化できる

手順8：「歌声の生成」

(8/9)

Ubuntu Terminal で実行

```
sudo bash run.sh --stage 5 --stop-stage 6
```

上記コマンドで歌声の生成ができる

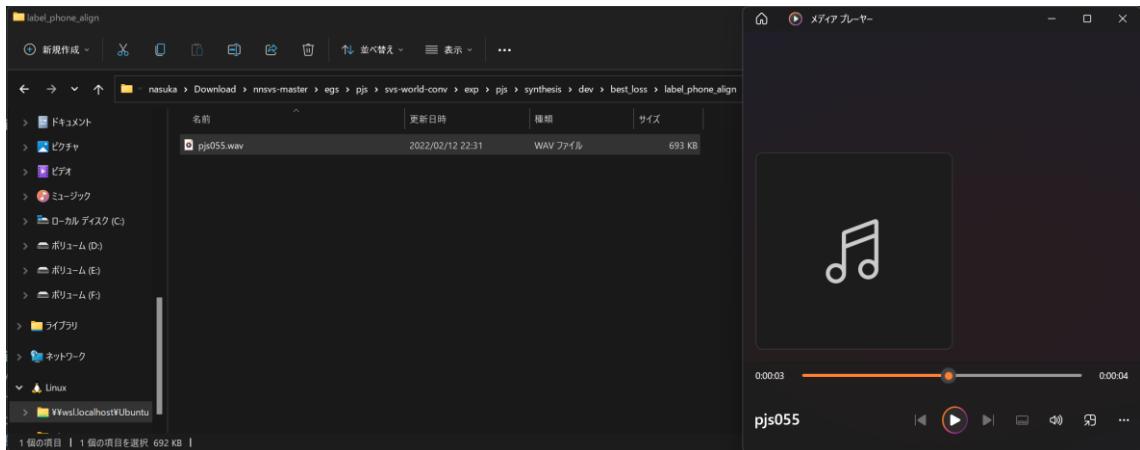
手順9：「生成音声の確認」

(9/9)

生成音声保存先

```
./pjs/svs-world-conv/exp/pjs/synthesis/dev/best_loss/label_phone_align/
```

デフォルトで生成音声は上記のディレクトリに保存されている
※生成音声の元楽譜は手順3「学習の前処理」で指定した検証データとなる



Windows 11以降の場合、エクスプローラーで簡単に音声を再生できる

参考

- Windows で可搬性のある NNSVS 環境で遊ぶ：

<https://qiita.com/taroushirani/items/26658cb691b8e5fcd698>

- Google Collaboratory で NNSVS で遊ぶ：

<https://qiita.com/taroushirani/items/ecl6cb9a6b3b691f5e74>

2.5 NNSVS のカスタマイズ

この節では以下のことについて説明する。

- ① 自作歌声データベースを使った学習方法
- ② 任意の曲を歌わせる方法
- ③ ハイパーパラメータの編集方法

① 自作歌声データベースを使った学習方法

NNSVS では /nnsvs/egs/ 下のディレクトリを「レシピ」と呼んでいる。

「レシピ」はデータベース毎に AI シンガーを新規作成できる仕組み。

NNSVS ではデフォルトで 7 個のレシピが作成されている。

jsut-song	Just コーパス
kiritan_singing	東北きりたん
natsume_singing	夏目悠李
nit-song070	nit-song070
ofuton_p_utagoe_db	おふとん P
oniku_kurumi_utagoe_db	御丹宮くるみ
pjs	PJS

▲デフォルトのレシピとデータベース名

詳細：<https://github.com/r9y9/nnsvs/tree/master/egs>

今回は pjs レシピを基に新規レシピを作成する方法を紹介する。

尚、pjs レシピを基にする場合は教師データの音声が 48kHz/16bit 統一 であることに注意する。

※例では Vocal2lab を使用して作成したデータセットを使うとする。

手順①-1：「NNSVS のダウンロード」

(1/8)

Ubuntu Terminal で実行

```
cd ~/Download  
git clone https://github.com/r9y9/nnsvs.git
```

自作データベース用に再び NNSVS をダウンロード

手順①-2：「pjs を NNSVS へコピー」

(2/8)

Ubuntu Terminal で実行

```
sudo cp -r ~/Download/nnsvs/egs ~/nnsvs/egs/[任意の名前]
```

ダウンロードした pjs を NNSVS へ任意の名前でコピー

手順①-3：「Python スクリプトの編集」

(3/8)

```
./egs/[任意の名前]/local/data_prep.py 229 行目  
# wav_path = join(args.pjs_root, name, f"{name}_song.wav") 変更前  
wav_path = join(args.pjs_root, name, f"{name}.wav")
```

上記のコードに変更
(「学習の前処理」のファイル探索規則を変更)

手順①-4：「ディレクトリ作成」

(4/8)

Ubuntu Terminal で実行

```
cd ~/nnsvs/egs/[任意の名前]/svs-world-conv/  
sudo mkdir -p ./downloads
```

データベース用のディレクトリを作成

手順①-5：「データベースを配置」

(5/8)

コマンドの場合

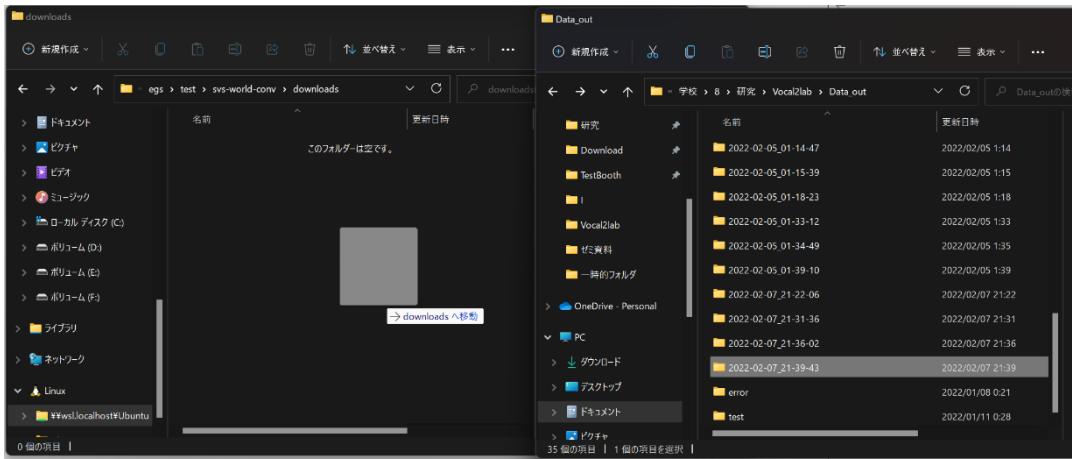
Ubuntu Terminal で実行

```
cd /mnt/c/Users/[ユーザー名]/Download/Vocal2lab/Data_out  
自作データベースのディレクトリ (Windows) へ移動
```

Ubuntu Terminal で実行

```
sudo cp -r ./[生成日時] ~/nnsvs/egs/test/svs-world-conv/downloads/  
レシピへ自作データベースをコピー
```

ファイルエクスプローラーの場合



手順①-6：「シェルスクリプトの編集」

(6/8)

```
~/nnsvs/egs/svs-world-conv/run.sh 59行目  
# python local/data_prep.py downloads/PJS_corpus_ver1.1 data --gain-  
normalize 変更前  
python local/data_prep.py downloads/[任意の名前] data --gain-normalize
```

上記のコードに変更
(自作データベースを前処理に指定)

手順①-7：「訓練用、評価用、検証用データの指定」

(7/8)

```
./egs/[任意の名前]/run.sh 63~67 行目
#find data/acoustic/ -type f -name "*.wav" -exec basename {} .wav \$; \$
#      | grep -v 030 | sort > data/list/utt_list.txt
#grep 056 data/list/utt_list.txt > data/list/$eval_set.list
#grep 055 data/list/utt_list.txt > data/list/$dev_set.list
#grep -v 056 data/list/utt_list.txt | grep -v 056 >
data/list/$train_set.list 変更前

find data/acoustic/ -type f -name "*.wav" -exec basename {} .wav \$; \$
| sort > data/list/utt_list.txt
grep 評価用データ名 data/list/utt_list.txt > data/list/$eval_set.list
grep 検証用データ名 data/list/utt_list.txt > data/list/$dev_set.list
grep -v 評価用データ名 data/list/utt_list.txt | grep -v 検証用データ名 >
data/list/$train_set.list
```

上記のコードに変更

※それぞれのデータ名は歌声データベース内のフォルダ名で指定

名前	更新日時	種類
arupusu_ichimanjaku	2022/01/08 0:53	ファイル フォルダー
fujino_yama	2022/01/08 0:53	ファイル フォルダー
hato	2022/01/08 0:53	ファイル フォルダー
hiraita_hiraita	2022/01/08 0:53	ファイル フォルダー
hoshinoyo	2022/01/08 0:53	ファイル フォルダー
jugoya_otsukisan	2022/01/08 0:53	ファイル フォルダー

例：データベースが上記のような構成の場合
評価用、検証用のデータ名をこのフォルダ名の中から選ぶ

```
find data/acoustic/ -type f -name "*.wav" -exec basename {} .wav \$; \$
| sort > data/list/utt_list.txt
grep hato data/list/utt_list.txt > data/list/$eval_set.list
grep hoshinoyo data/list/utt_list.txt > data/list/$dev_set.list
grep -v hato data/list/utt_list.txt | grep -v hoshinoyo >
data/list/$train_set.list
```

例で示したデータベースの場合、コードはこのようになる

手順①-8：「学習」

(8/8)

Ubuntu Terminal で実行

```
sudo bash run.sh --stage 0 --stop-stage 6
```

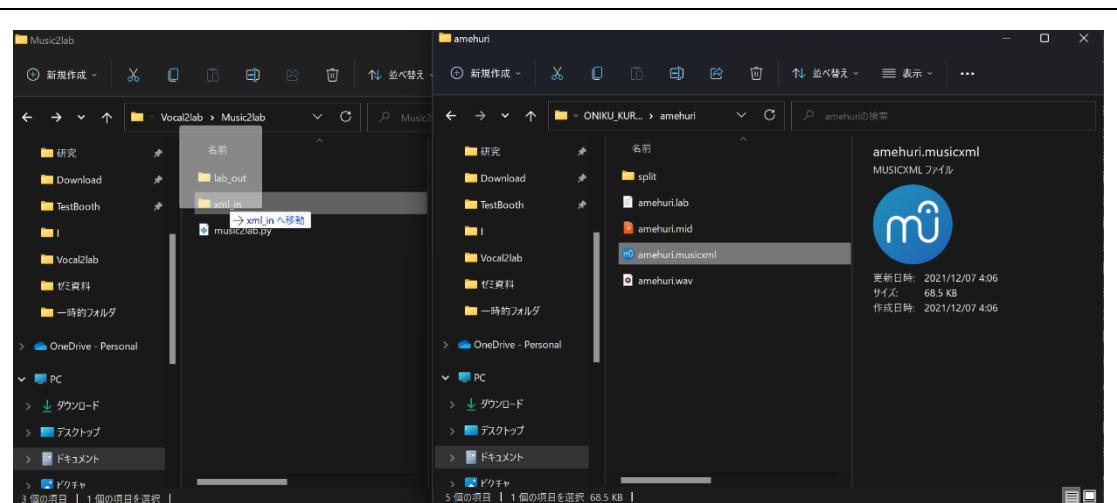
1. 4 の動作確認同様に上記コマンドで学習をする

② 任意の曲を歌わせる方法

NNSVS は楽譜ラベルを入力に任意の曲を歌わせることができる。この節では Vocal2lab を使用した楽譜ラベル生成を行うので事前に Vocal2lab のセットアップを完了しておく必要がある。

手順②-1：「Vocal2lab で楽譜ラベルの生成」

(1/6)



Musescore で作成した楽譜ファイルを /Vocal2lab/Music2lab/xml_in/ へ入れる

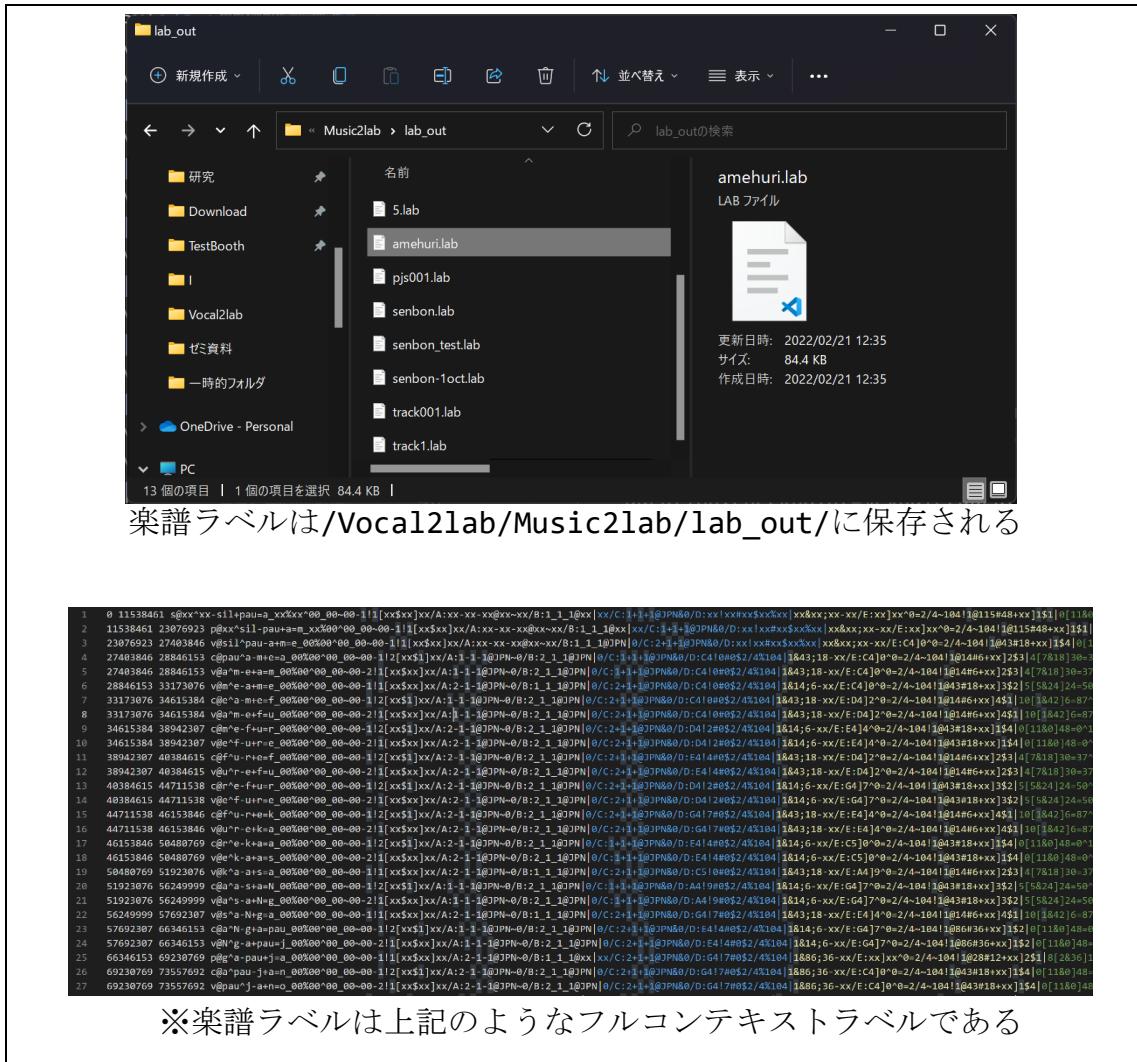
Windows PowerShell で実行

```
cd /Vocal2lab/Music2lab  
python ./music2lab.py amehuri.musicxml amehuri.lab generate
```

Vocal2lab 内の Music2lab を実行

手順②-2：「楽譜ラベルの確認」

(2/6)



手順②-3：「楽譜ラベルの配置」

(3/6)

Ubuntu Terminal で実行

```
cd /mnt/c/Users/[ユーザー名]/Download/Vocal2lab/Music2lab/xml_out/  
楽譜ラベルディレクトリ (Windows) へ移動
```

Ubuntu Terminal で実行

```
sudo cp -r ./amehuri.lab ~/Download/  
        楽譜ディレクトリを任意の Linux ディレクトリへコピー  
        (例ではホームディレクトリの Download ディレクトリへ配置)
```

手順②-4：「シェルスクリプトの編集」

(4/6)

Ubuntu Terminal で実行

```
cd ~/nnsvs/egs/_common/spsvs/  
sudo cp ./synthesis.sh ./synthesisYourSong.sh  
          編集用にスクリプトを複製する
```

```
~/nnsvs/egs/_common/spsvs/synthesisYourSong.sh 44～47 行目  
#utt_list=../data/list/$s.list ¥  
#in_dir=data/acoustic/$input/ ¥  
#out_dir=$expdir/synthesis/$s/${acoustic_eval_checkpoint/.pth/}/$input ¥  
#ground_truth_duration=$ground_truth_duration 変更前  
  
ground_truth_duration=false ¥  
label_path=~/Download/amehuri.lab ¥ #楽譜ラベルのパス  
out_wav_path=~/Download/amehuri.wav #保存先のパス  
          複製した synthesisYourSong.sh の  
          44～47 行目を上記のように置き換える
```

手順②-5：「シェルスクリプトの編集」

(5/6)

```
~/nnsvs/egs/[任意のレシピ]/svs-world-conv/run.sh 95～98 行目  
#if [ ${stage} -le 6 ] && [ ${stop_stage} -ge 6 ]; then  
    #echo "stage 6: Synthesis waveforms"  
    . $NNSVS_COMMON_ROOT/synthesis.sh  
#fi  
  
if [ ${stage} -le 6 ] && [ ${stop_stage} -ge 6 ]; then  
    echo "stage 6: Synthesis waveforms"  
    . $NNSVS_COMMON_ROOT/synthesisYourSong.sh  
fi  
          任意レシピの run.sh の stage6 で呼び出すスクリプトを  
          synthesisYourSong.sh へ  
          置き換える
```

手順②-6：「歌声の生成」

(6/6)

Ubuntu Terminal で実行

```
sudo bash run.sh --stage 6 --stop-stage 6
```

学習済みのレシピで stage6 を実行すれば
指定したディレクトリに音声データが保存される

③ ハイパーパラメータの編集方法

NNSVS は 3 つのニューラルネットワークを用いて学習している。それぞれのネットワークのハイパーパラメータ（層数、各層の次元数、最適化関数、エポック数など）をレシピ毎に変更することが可能である。

この節では簡易的に説明する。（部分的な理解しかできていない為）

・タイムラグ（発音タイミング）モデル 構造設定

```
/[任意のレシピ]/svs-world-conv/conf/train/timelag/model/  
timelag_ffn.yaml  
stream_sizes: [1]  
has_dynamic_features: [false]  
stream_weights: [1]  
  
netG:  
    _target_: nnsvs.model.FeedForwardNet #フィードフォワードネット  
    in_dim: 420          #入力層の次元数  
    out_dim: 1           #出力層の次元数  
    hidden_dim: 128      #隠れ層の次元数  
    num_layers: 2         #層数  
    dropout: 0.5         #ドロップアウト係数
```

上記ディレクトリの `timelag_ffn.yaml` にネットワーク構造が定義される

- ・音素継続長（子音と母音のタイミング）モデル 構造設定

```
/[任意のレシピ]/svs-world-conv/conf/train/duration/model/
duration_lstm.yaml
stream_sizes: [1]
has_dynamic_features: [false]
stream_weights: [1]

netG:
  _target_: nnsvs.model.LSTM_RNN #リカレントニューラルネットワーク
  in_dim: 420                  #入力層の次元数
  out_dim: 1                   #出力層の次元数
  hidden_dim: 64                #隠れ層の次元数
  num_layers: 2                 #層数
  bidirectional: true          #双方向 LSTM
  dropout: 0.5                 #ドロップアウト係数
上記ディレクトリの timelag_ffn.yaml にネットワーク構造が定義される
```

- ・音響特徴量モデル 構造設定

```
/[任意のレシピ]/svs-world-conv/conf/train/acoustic/model/
acoustic_conv.yaml
# (mgc, lf0, vuv, bap)
stream_sizes: [180, 3, 1, 15]
has_dynamic_features: [true, true, false, true]
num_windows: 3
# If None, automatically set based on stream sizes
stream_weights:

netG:
  _target_: nnsvs.model.Conv1dResnet #一次元 Resnet
  in_dim: 424                      #入力層の次元数
  out_dim: 199                      #出力層の次元数
  hidden_dim: 128                   #隠れ層の次元数
  num_layers: 6                     #層数
  dropout: 0.1                      #ドロップアウト係数
上記ディレクトリの acoustic_conv.yaml にネットワーク構造が定義され
る
```

- ・学習設定

```
myconfig.yaml
out_dir: exp
nepochs: 50          #エポック数
checkpoint_epoch_interval: 20 #学習パラメータ保存周期

stream_wise_loss: false
use_detect_anomaly: true

optim:
    optimizer:           #最適化関数
        name: Adam
        params:
            lr: 0.001
            betas: [0.9, 0.999]
            weight_decay: 0.0
    lr_scheduler:         #スケジューラ
        name: StepLR
        params:
            step_size: 20
            gamma: 0.5

resume:
    checkpoint:
        load_optimizer: false

cudnn:
    benchmark: false
    deterministic: true
```

学習実行時の振る舞いは
/[任意のレシピ]/svs-world-conv/conf/train/[モデル名]
/train/myconfig.yaml
に定義されている

第3章 音声&楽譜データの作成方法

3.1 音声データ作成方法

音声を録音する際は以下の点に気を付ける

- ・雑音が入らないようにする
- ・音割れしないようにする
- ・収録音声の音量レベルを揃える
- ・歌声の音程とテンポが正確になるようにする

この説明では以下のソフトを利用した手順となる

- ① 音声収録、音量レベル調整（音声編集ソフト Audacity）
- ② ノイズ除去（音声解析&修復ソフト iZotopeRX9）
- ③ 楽譜用にデータの長さを調節（音楽制作ソフト MPCBeats）
- ④ 歌声のピッチ&タイミング補正（ボーカル補正ソフト Melodyne5）

Audacity : <https://www.audacityteam.org/>

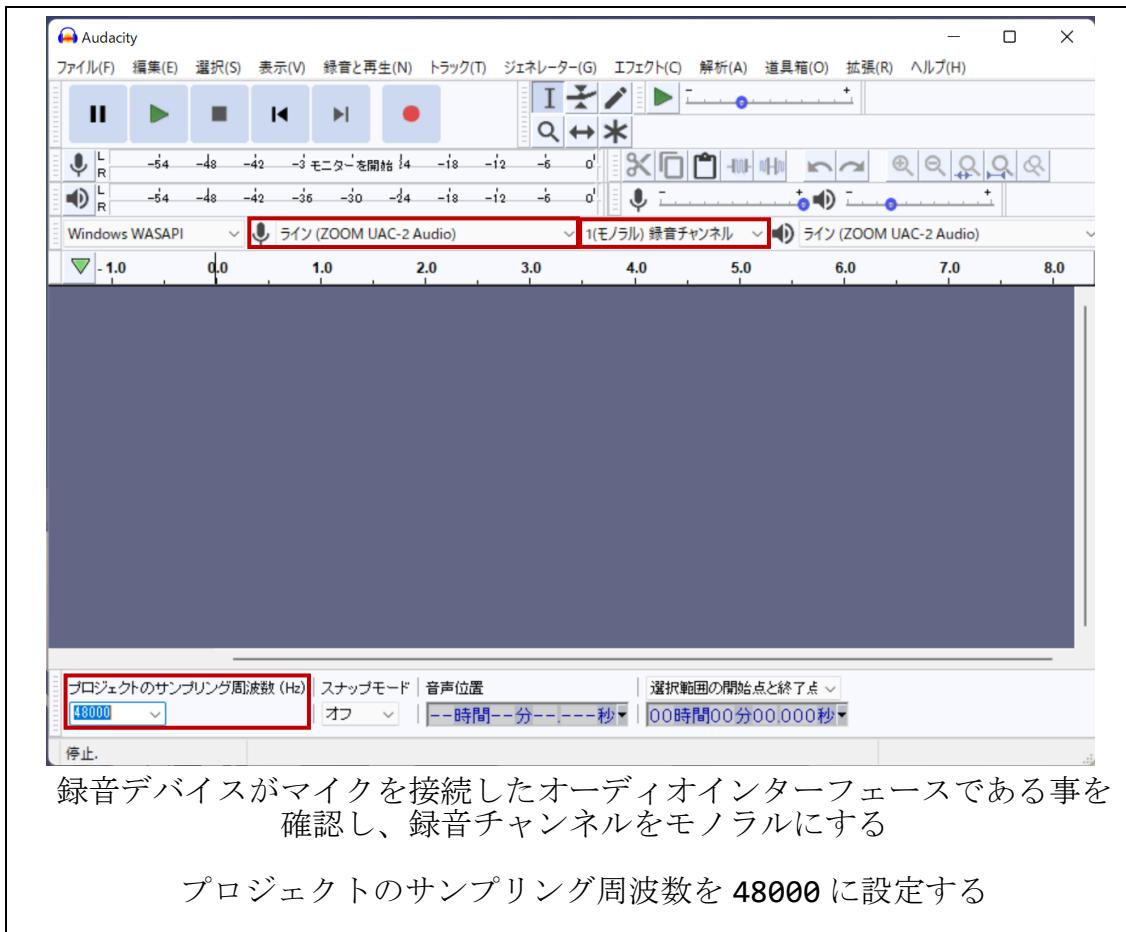
iZotopeRX9: <https://www.izotope.com/en/products/rx.html>

MPCBeats: <https://www.akapro.com/mpc-beats>

Melodyne5: <https://www.celemony.com/en/melodyne/what-is-melodyne>

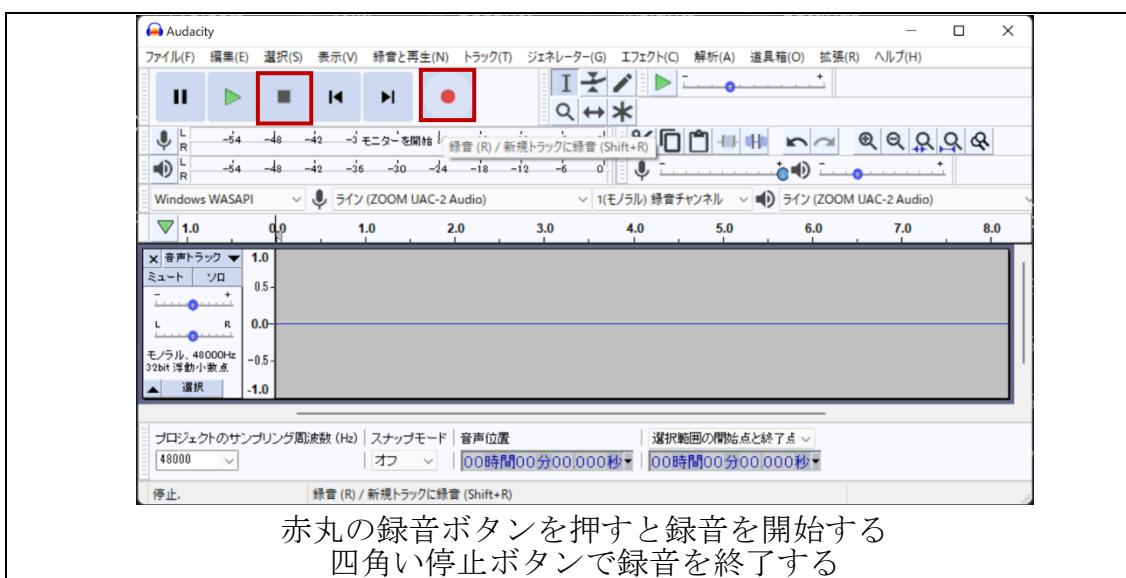
手順①-1：「音声の録音設定」（Audacity）

(1/5)



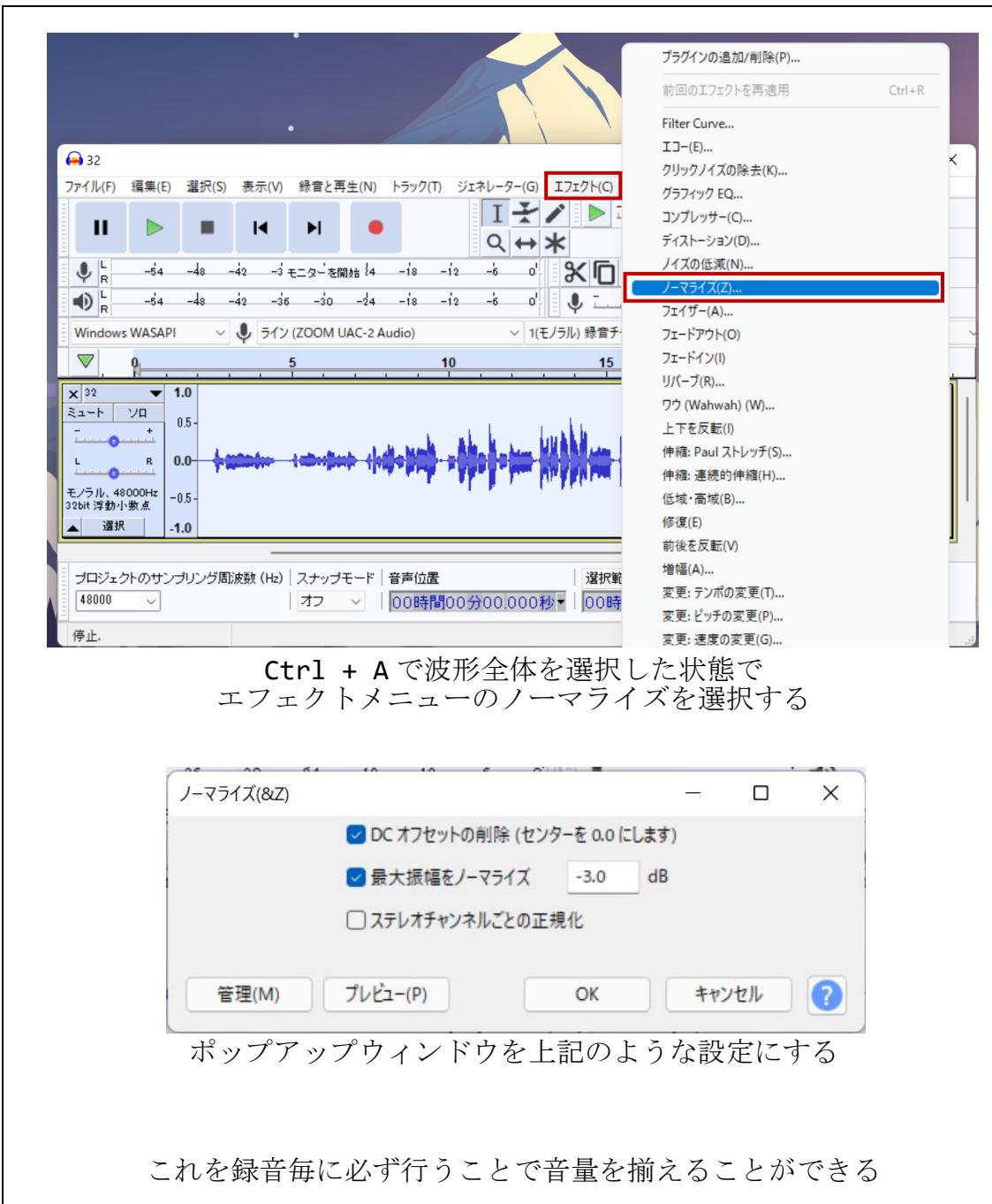
手順①-2：「音声の録音」

(2/5)



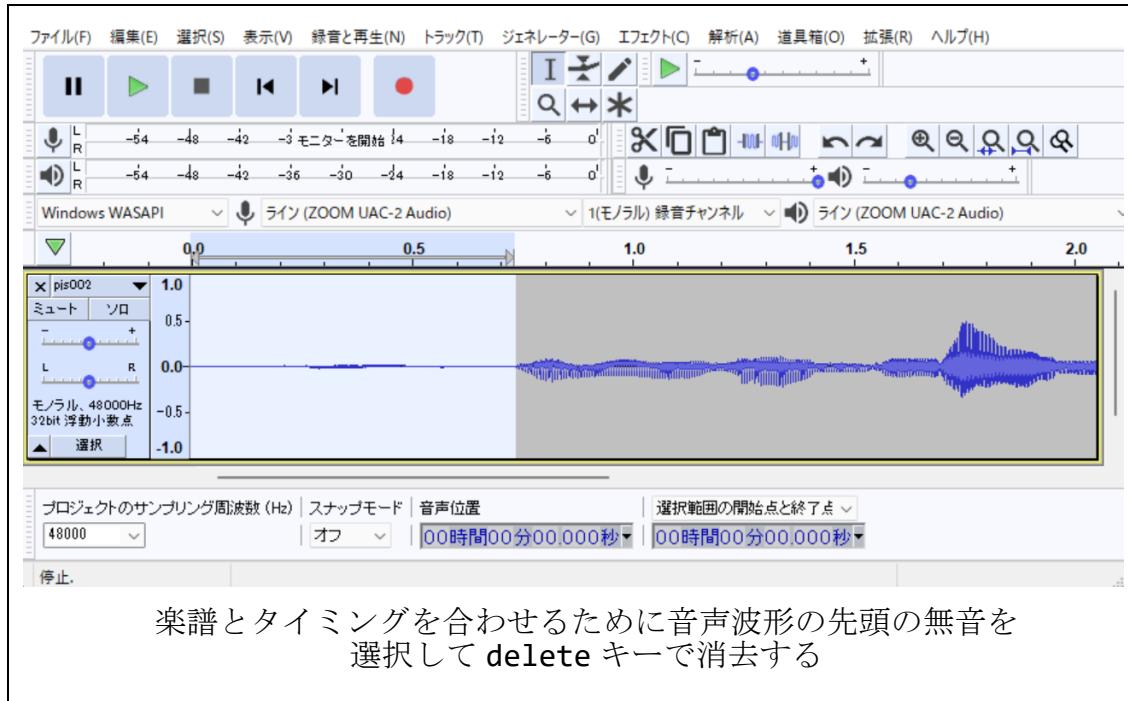
手順①-3：「音量レベル調整」

(3/5)



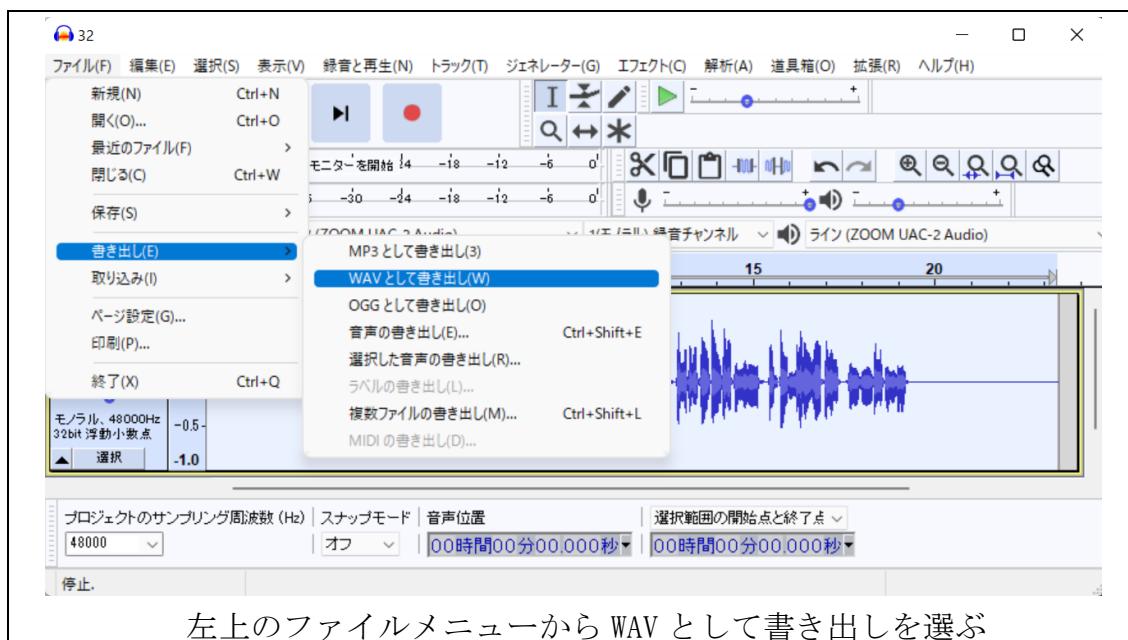
手順①-4：「先頭の無音を消去」

(4/5)



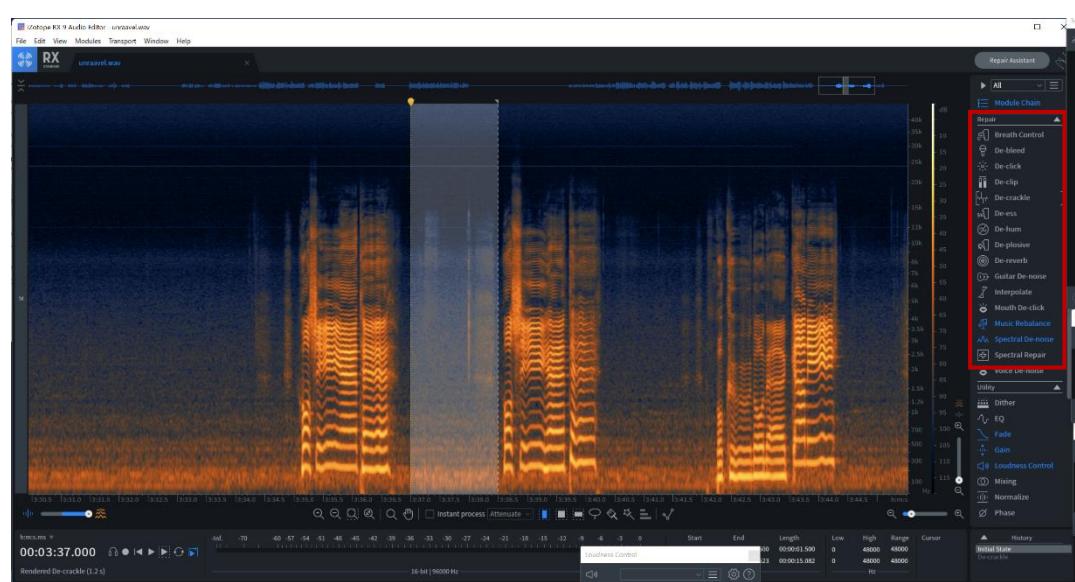
手順①-5：「音声の書き出し」

(5/5)



手順②：「ノイズ除去」（iZotopeRX9）

(1/1)



中央の音声波形を範囲選択して
右側のノイズフィルタを使い分けることで処理する

以下にノイズフィルタの種類を簡易的に挙げる

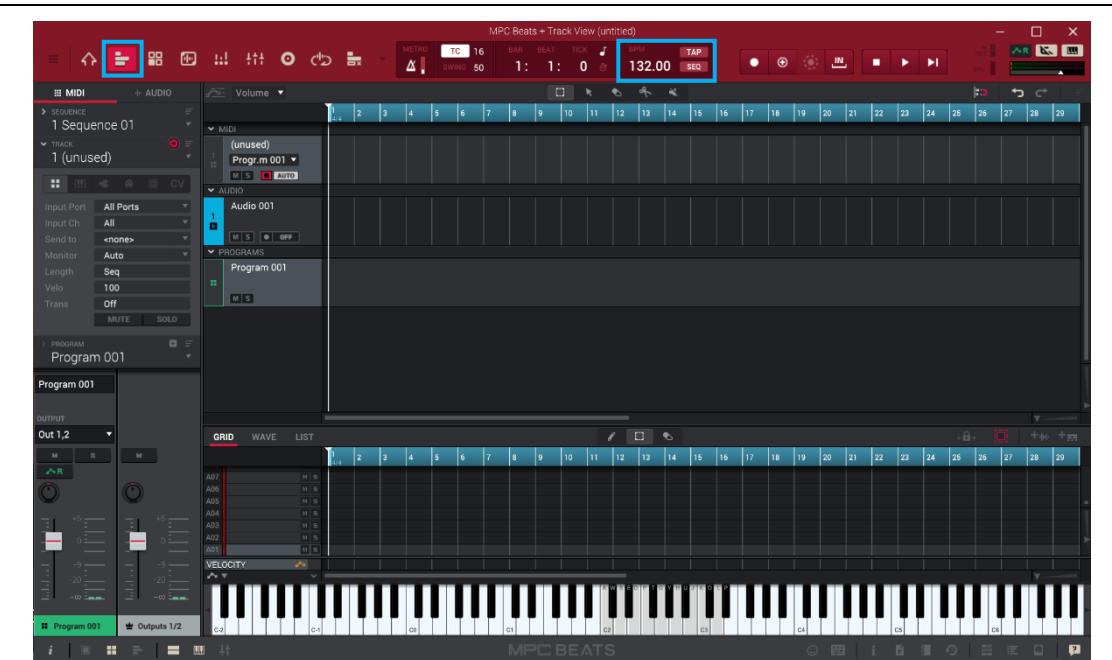
Breath Control	息継ぎの音を調整
De-bleed	録音時に混ざったオケ音源を低減させる
De-click	メトロノーム等の定期的なノイズを低減
De-clip	音割れを補正
De-crackle	マイクが拾うホワイトノイズ成分を低減
De-ess	歯擦音の過度な高音域を低減
De-hum	機材の干渉によるノイズ成分を低減
De-plosive	マイクに息がかかった時のノイズを低減
De-reverb	リバーブを低減
Guitar De-noise	ギターのノイズを低減
Interpolate	音声波形の補間
Mouth De-click	開口音の低減

詳細：RX9 公式ドキュメント

<https://s3.amazonaws.com/izotopedownloads/docs/rx9/ja/index.html>

手順③-1：「音声のインポート」（MPCBeats）

(1/4)



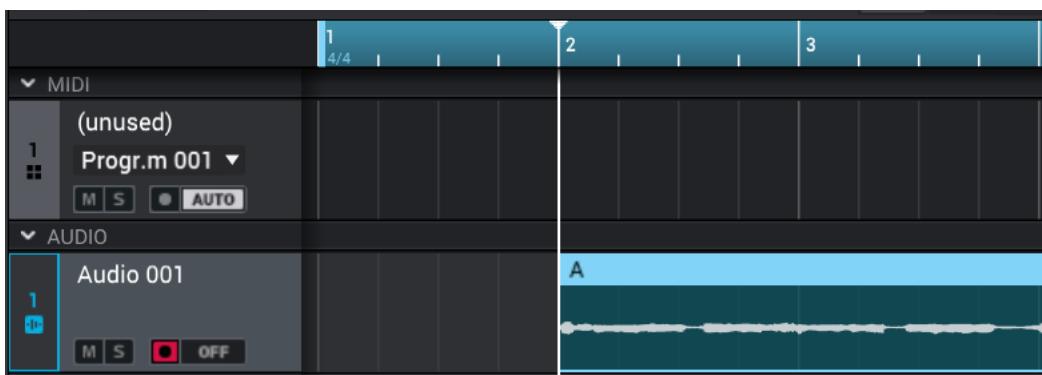
新規プロジェクトを作成し、Track View に切り替えた後
歌った曲のテンポを上部の BPM に入力する



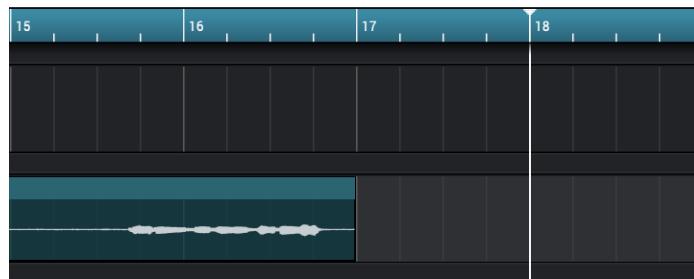
オーディオトラックに音声をドラッグ&ドロップで追加

手順③-2：「音声位置の調整」（MPCBeats）

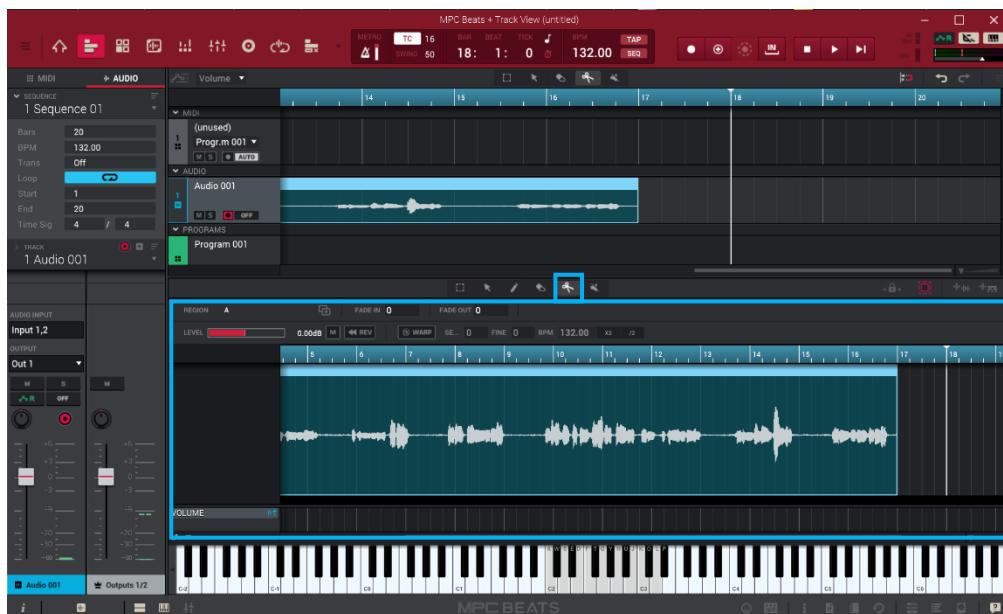
(2/4)



必ず1小節以上開けてから音声を配置する
※4/4拍子かつ最初の音が1拍目にくる曲の場合、
上記画像のようになる



終了位置も1小節以上開ける



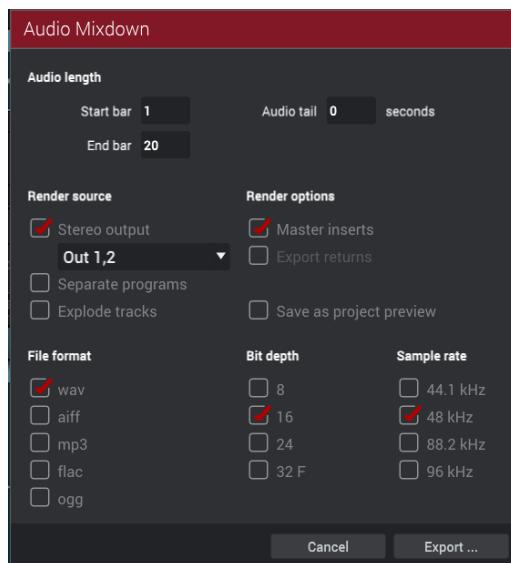
音声を分割したい場合は音声クリップを選択したときに表示される
エディットウィンドウの分割モードで編集できる

手順③-3：「音声の書き出し」（MPCBeats）

(3/4)



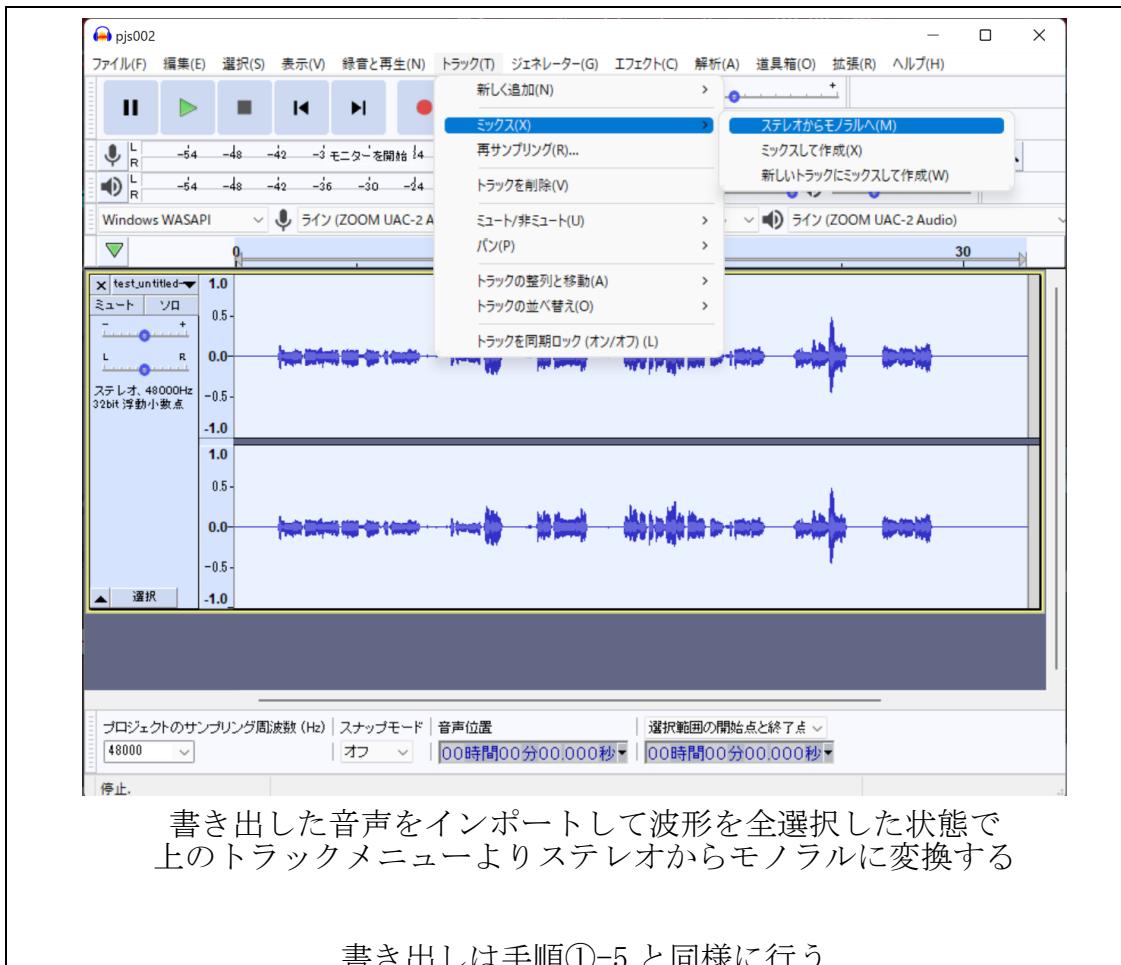
左上のメニューから
Export > As Audio Mixdown
を選択



Mixdown 設定では上記のような設定にして書き出しをする
※Start bar と End bar は音声に応じて変更する

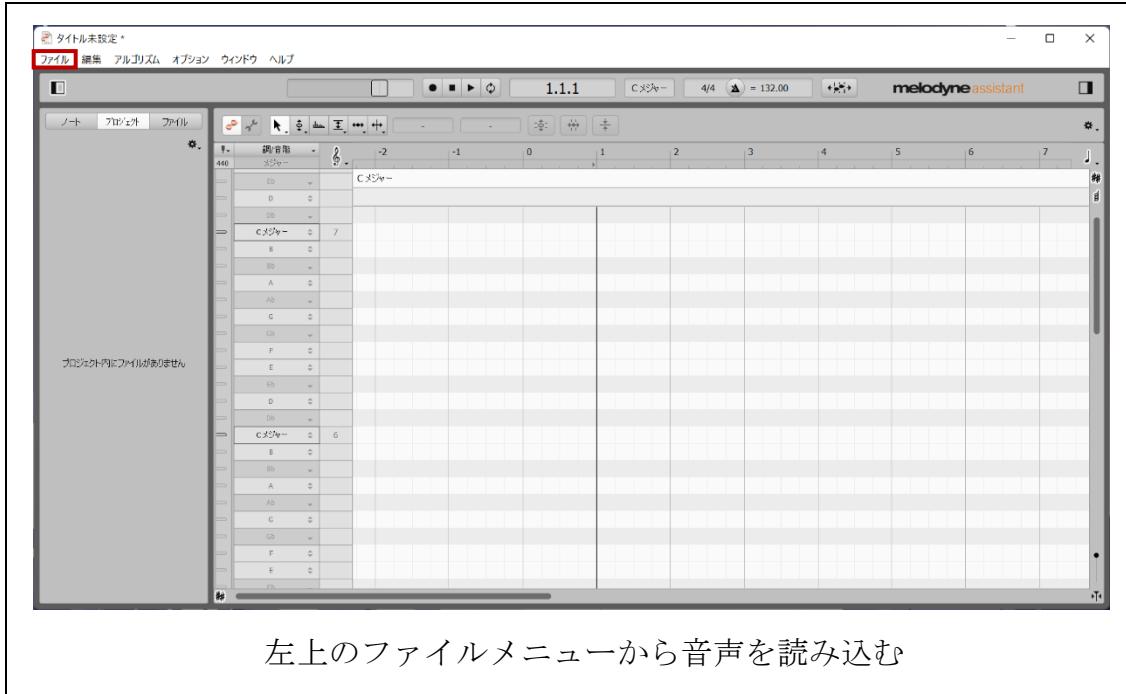
手順③-4：「音声をモノラルに戻す」（Audacity）

(4/4)



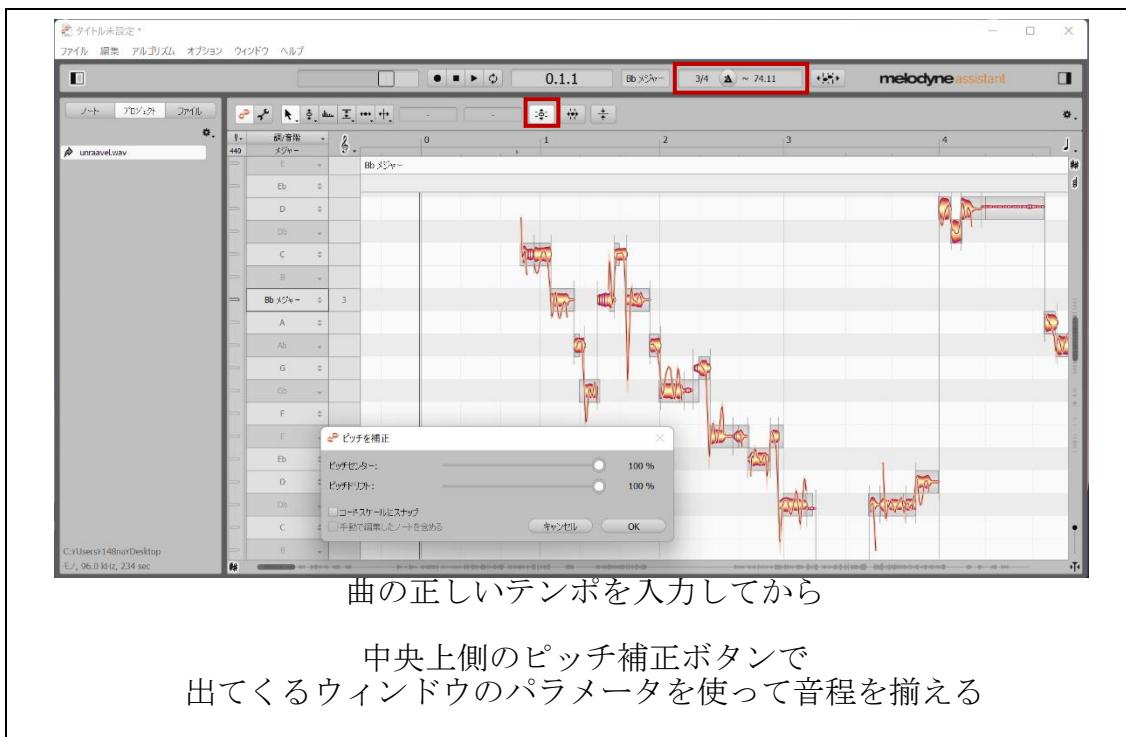
手順④-1：「音声を読み込む」（Melodye5）

(1/5)



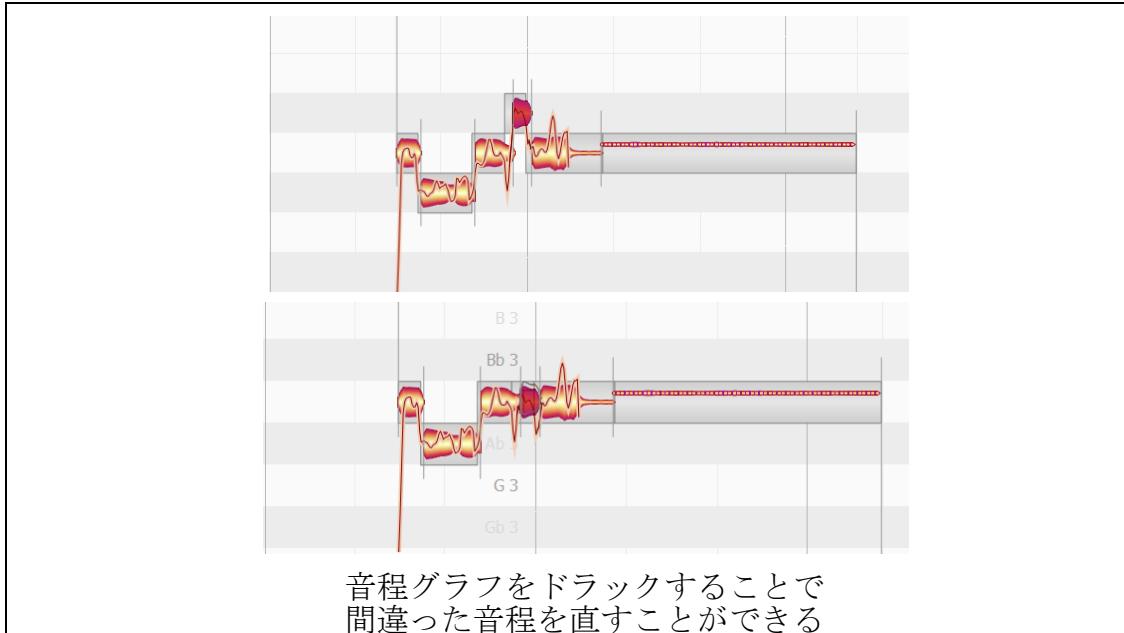
手順④-2：「ピッチ補正をする」（Melodye5）

(2/5)



手順④-3：「ピッチ補正をする」（Melodye5）

(3/5)



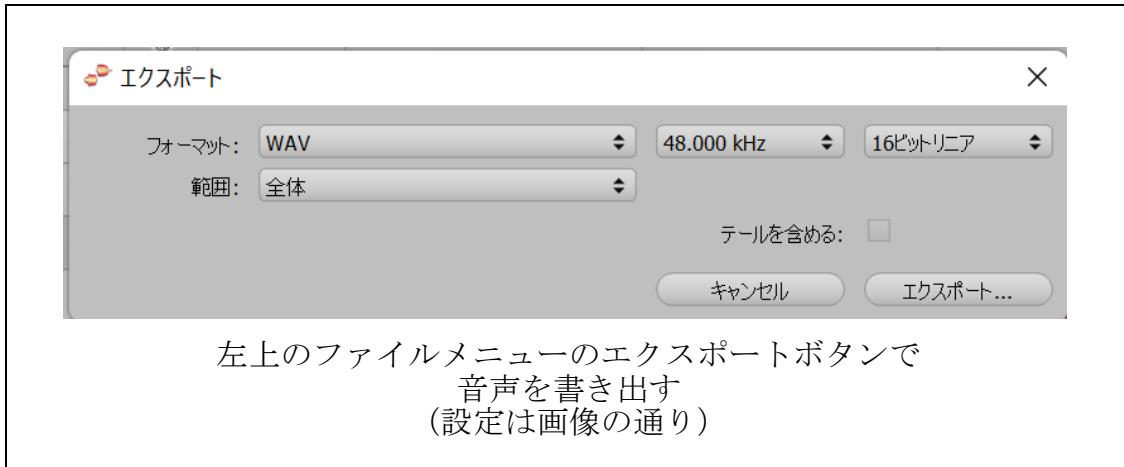
手順④-4：「タイミング補正をする」（Melodye5）

(4/5)



手順④-5：「書き出し」 (Melodye5)

(5/5)



3.2 楽譜データ作成方法

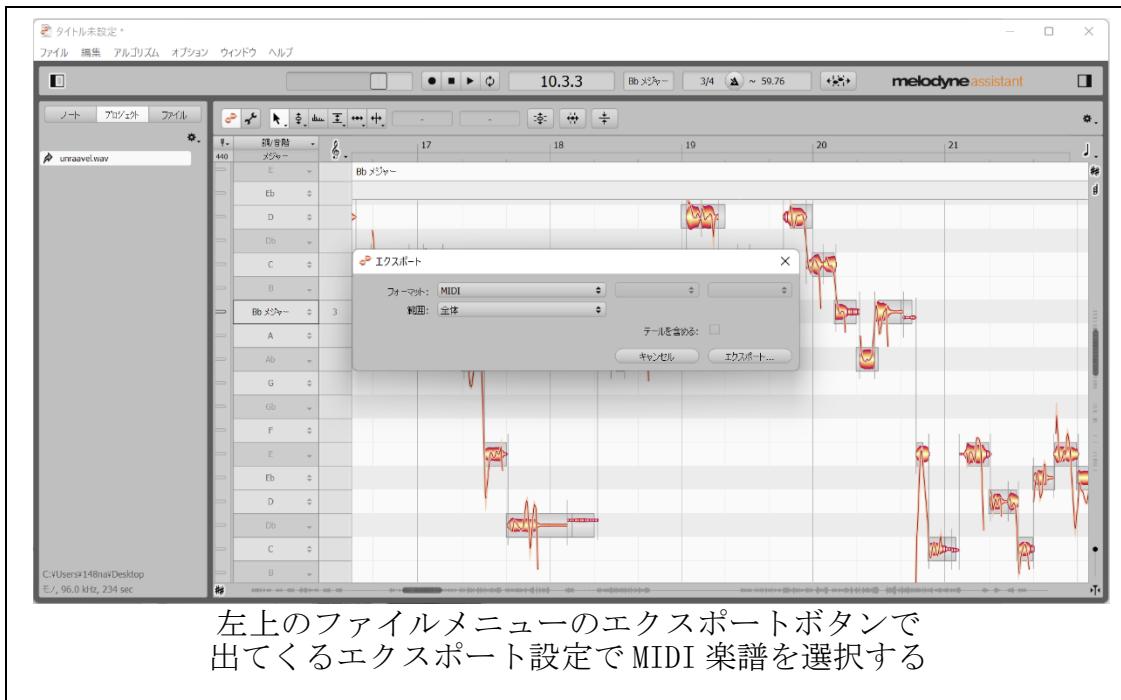
この説明では以下のソフトを利用した手順となる

- ① 声から MIDI 楽譜を生成(ボーカル補正ソフト Melodyne5)
- ② MIDI 楽譜を修正(音楽制作ソフト MPCBeats)
- ③ MIDI 楽譜から XML 楽譜にする(楽譜制作ソフト Musescore3)

Musescore3 : <https://musescore.com/>

手順①：「声から MIDI 楽譜を生成」 (Melodye5)

(1/1)



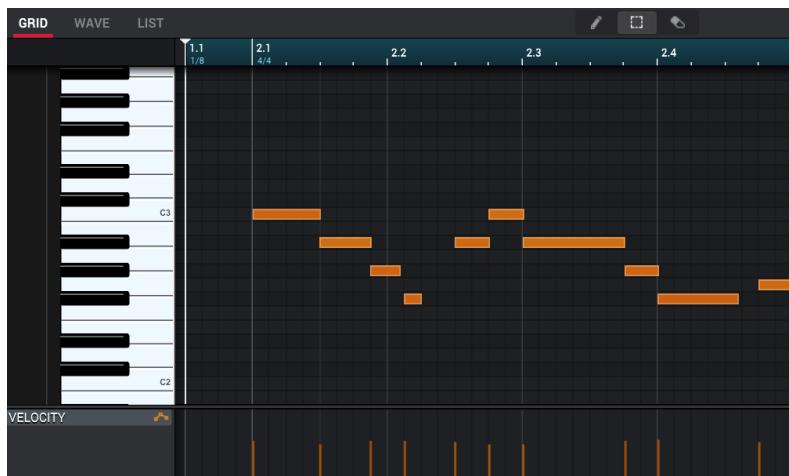
手順②-1：「MIDI 楽譜の読み込み」 (MPCBeats)

(1/3)

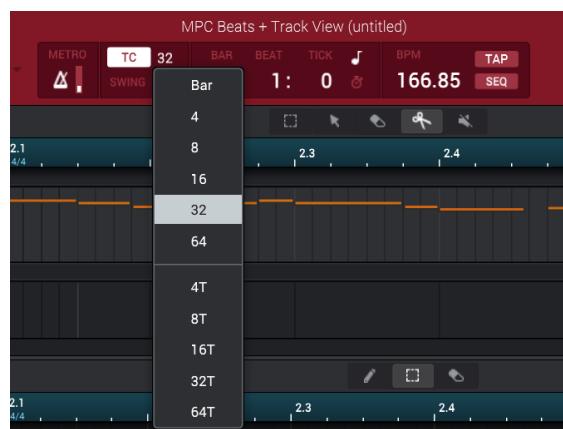


手順②-2：「MIDI 楽譜の編集」(MPCBeats)

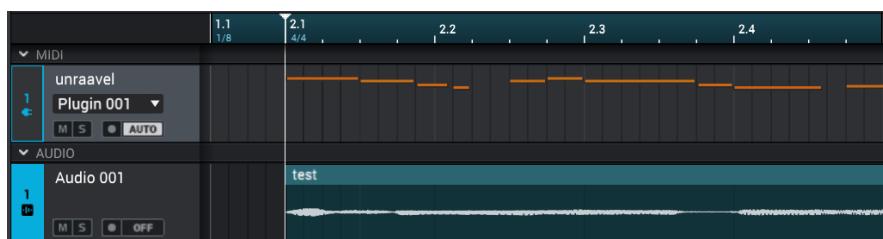
(2/3)



下側の編集ウィンドウで表示されるノーツを調整する



調整の際、スナップ設定を32分音符基準にしておくとやりやすい



また、音声を読み込んで重ねながら調整するとやりやすい

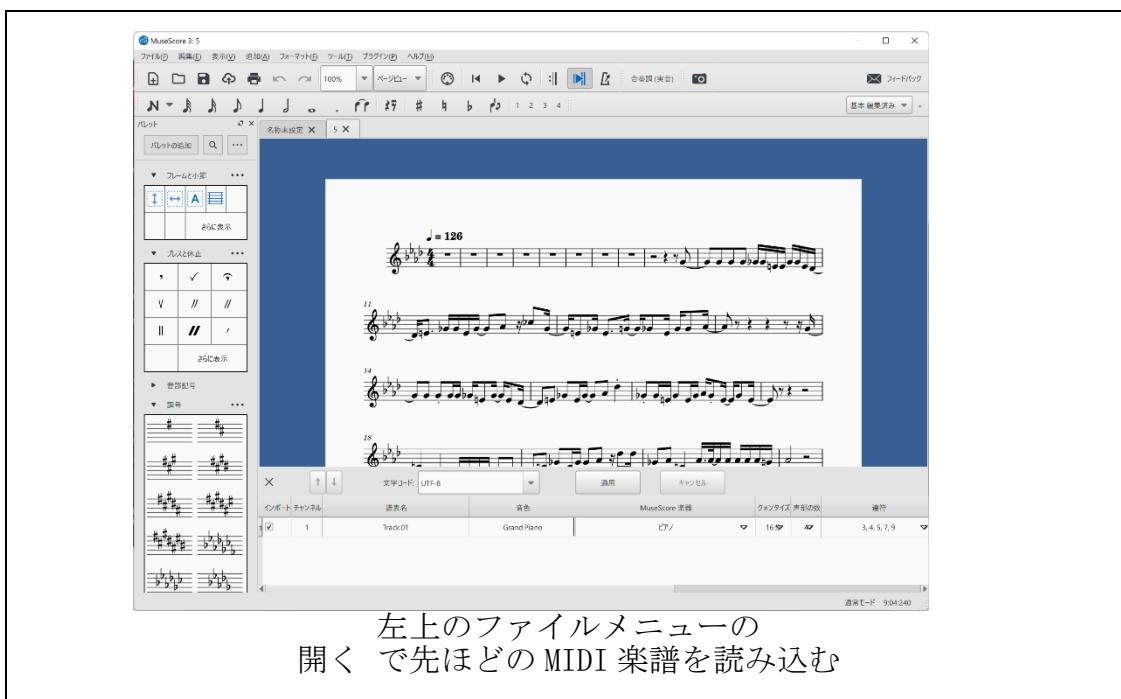
手順②-3：「MIDI 楽譜の書き出し」(MPCBeats)

(3/3)



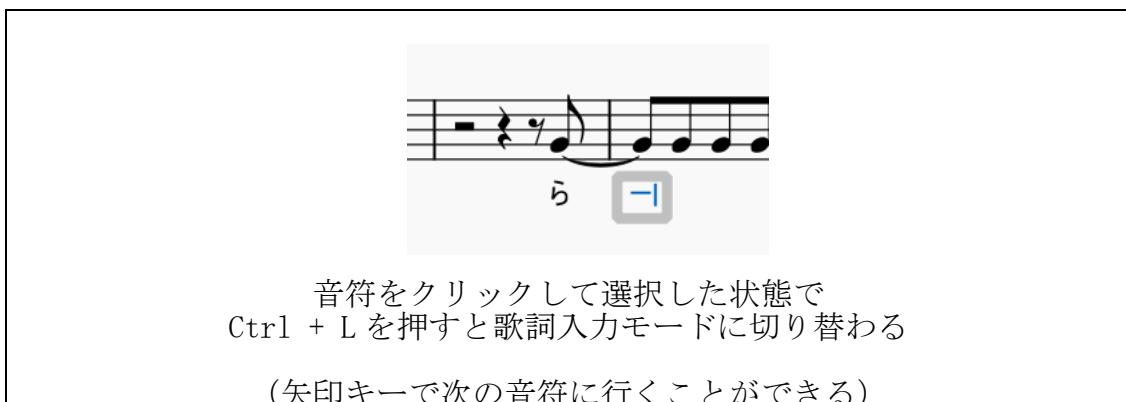
手順③-1：「MIDI 楽譜の読み込み」(Musescore3)

(1/3)



手順③-2：「歌詞を入れる」(Musescore3)

(2/3)



手順③-3：「楽譜を書き出す」(Musescore3)

(3/3)

