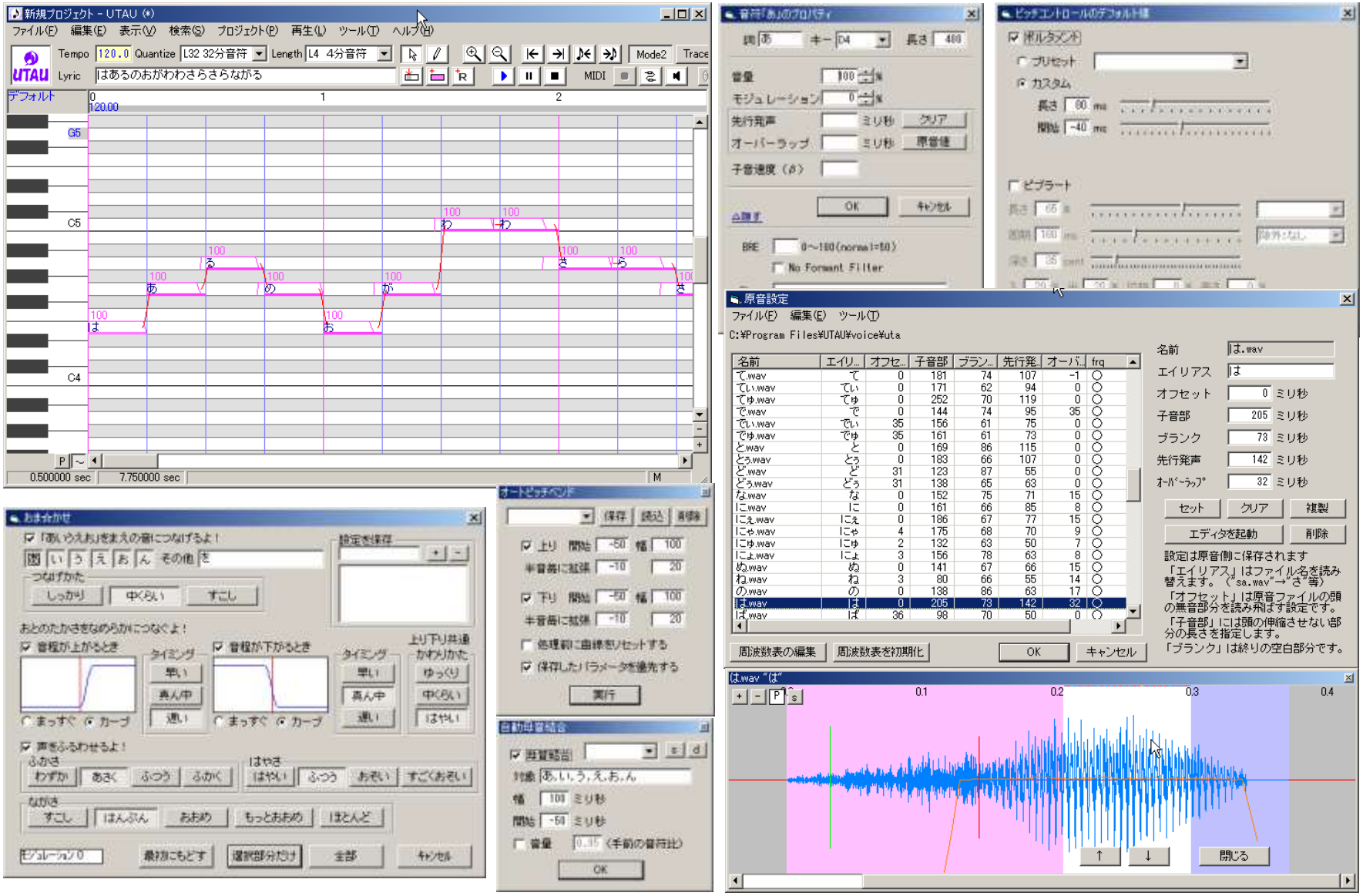
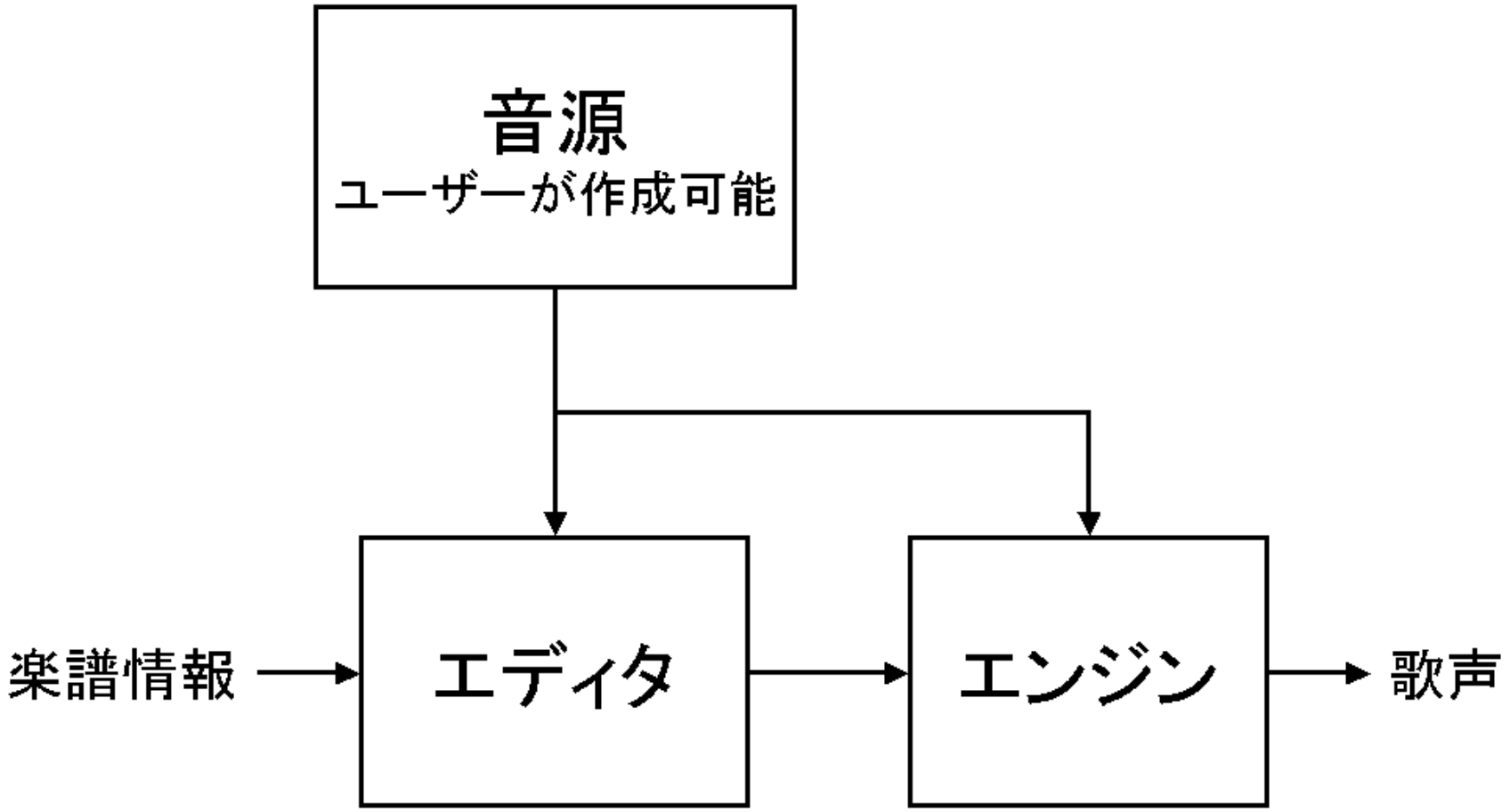
# UTAUの基本的アルゴリズムと開発経緯

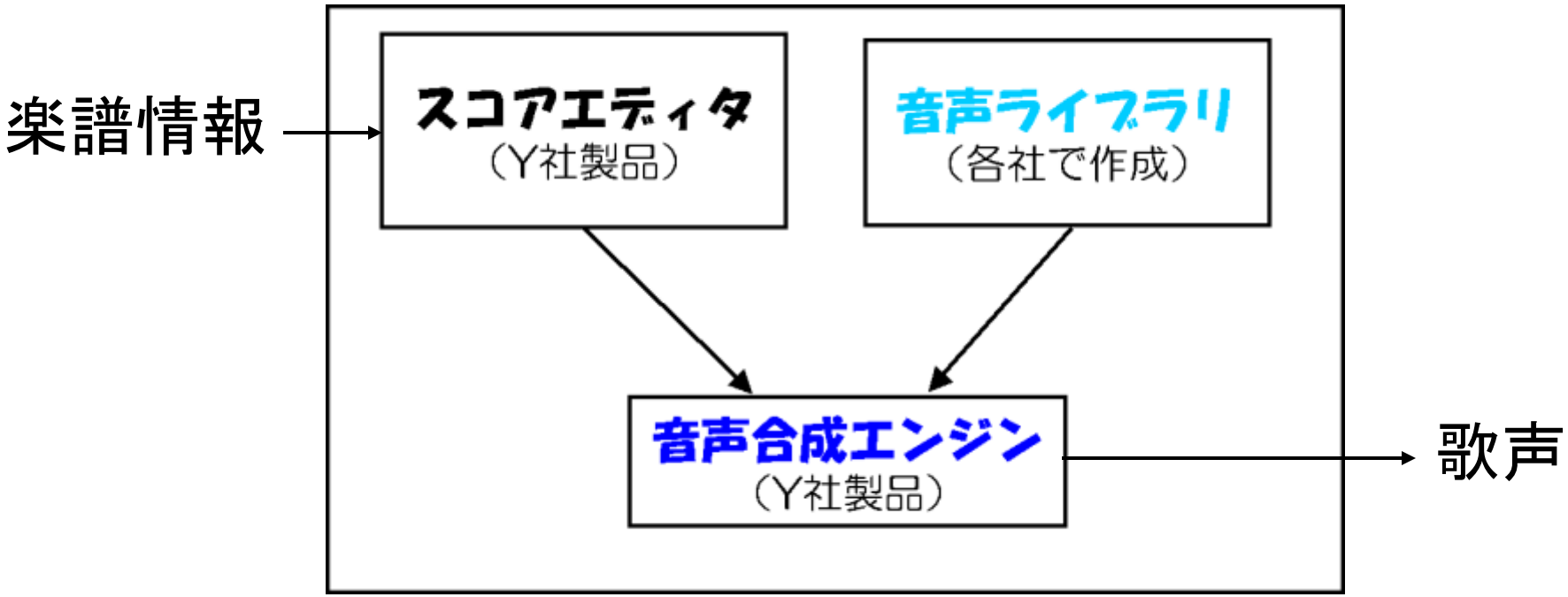
# 歌声ソフトウェアUTAU



# ソフトウェア『UTAU』の概略

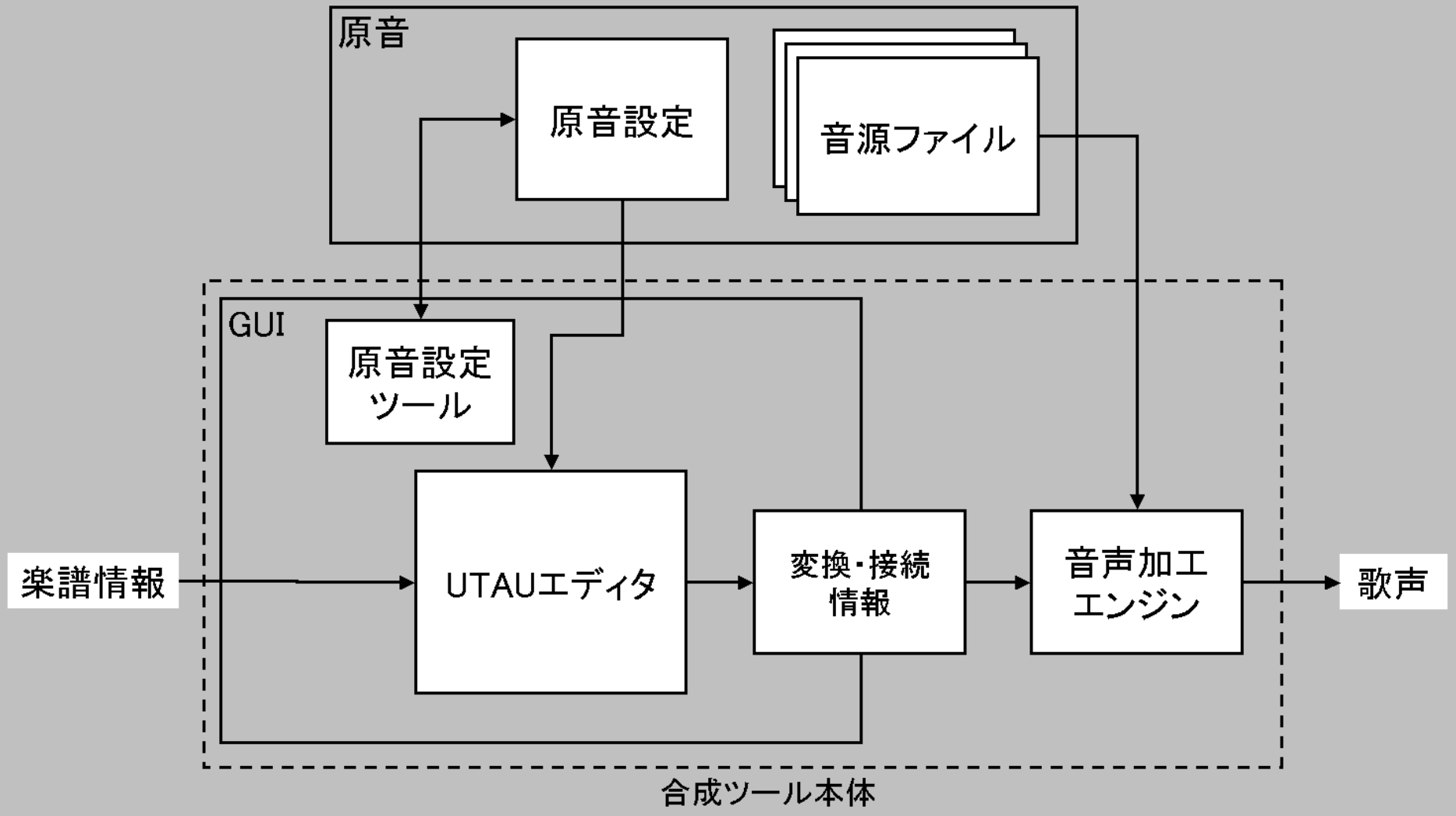


# 参考：VOCALOIDの構成

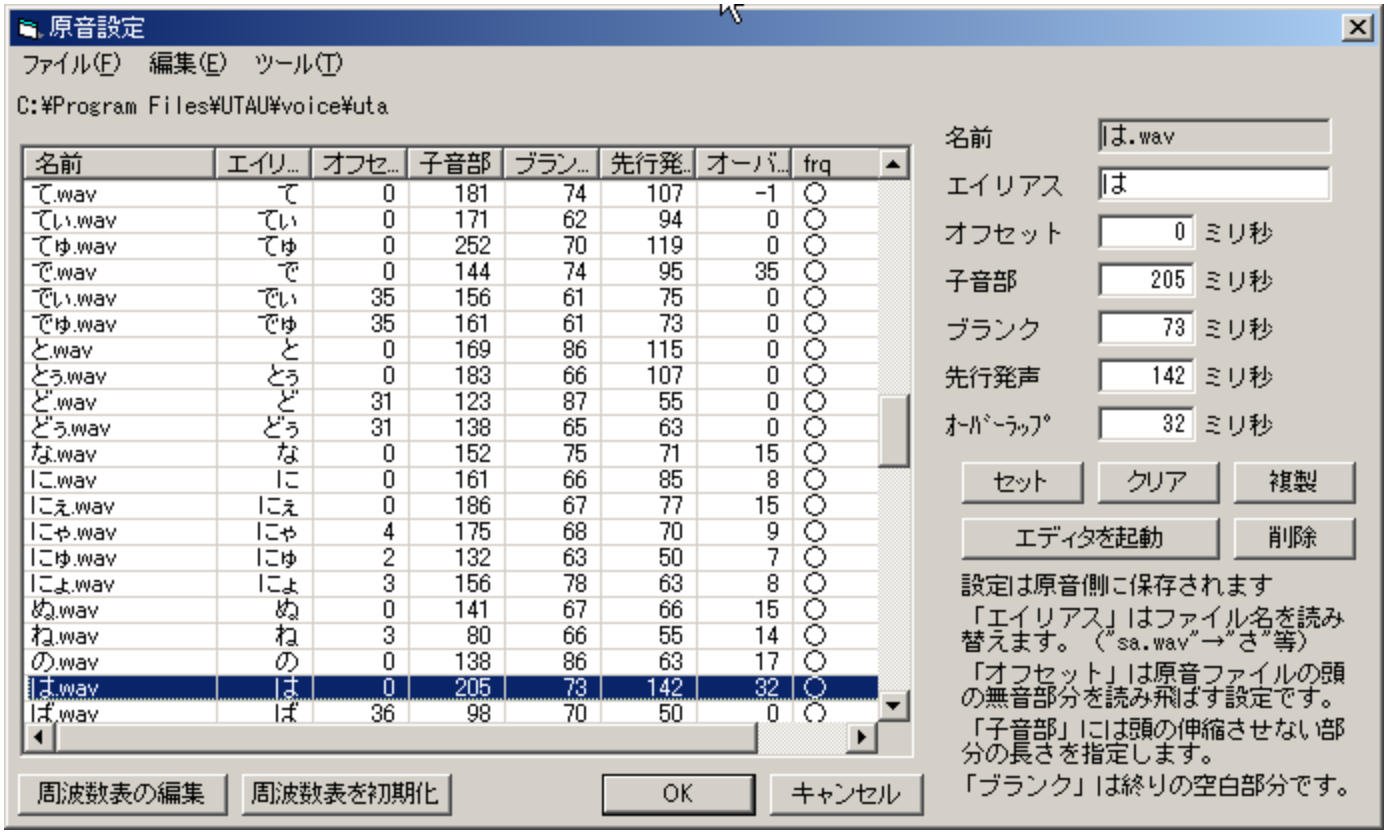



※この枠内全部で一つの製品

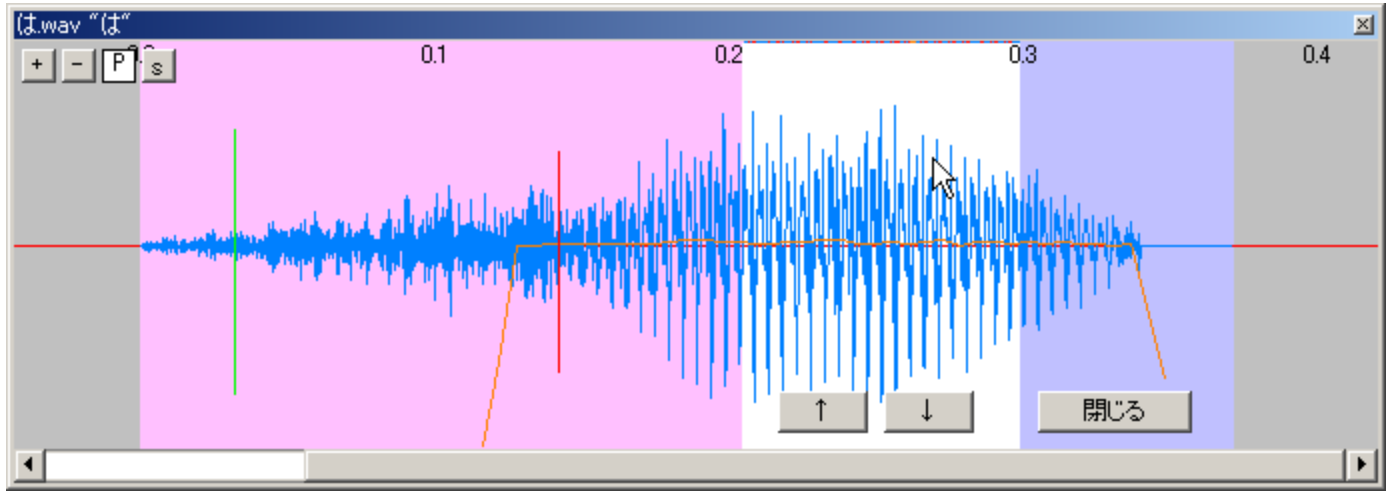
# UTAUの構成


原音
原音設定
音源ファイル
GUI
原音設定
ツール
楽譜情報
UTAUエディタ
変換・接続
情報
音声加工
エンジン
歌声
合成ツール本体

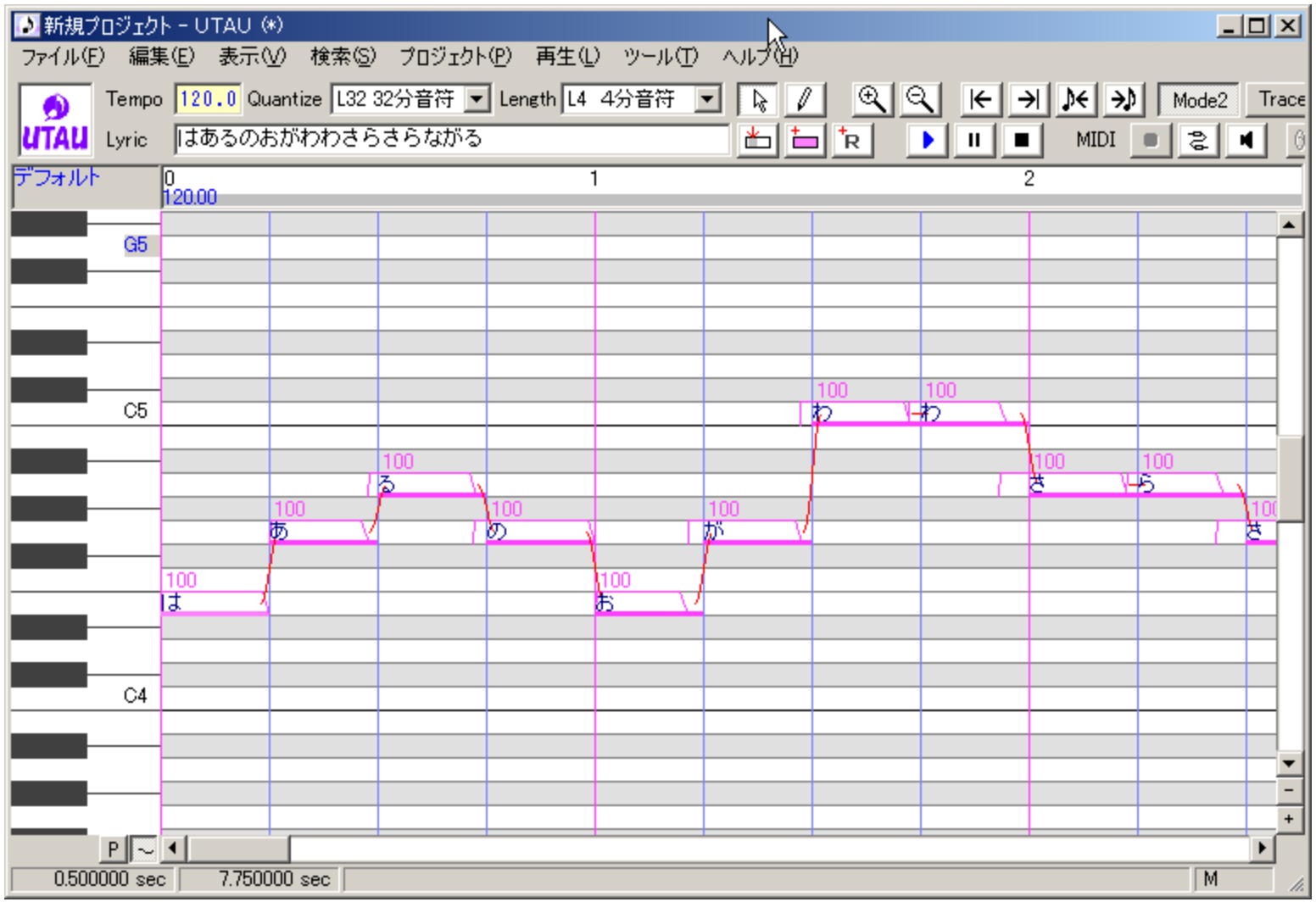
# 原音設定ツール1


原音設定
ファイル(F)　編集(E)　ツール(T)
C:¥Program Files¥UTAU¥voice¥uta
名前　エイリ…　オフセ…　子音部　ブラン…　先行発…　オーバ…　frq
て.wav　て　0　181　74　107　-1
てい.wav　てい　0　171　62　94　0
てゅ.wav　てゅ　0　252　70　119　0
で.wav　で　0　144　74　95　35
でい.wav　でい　35　156　61　75　0
でゅ.wav　でゅ　35　161　61　73　0
と.wav　と　0　169　86　115　0
とぅ.wav　とぅ　0　183　66　107　0
ど.wav　ど　31　123　87　55　0
どぅ.wav　どぅ　31　138　65　63　0
な.wav　な　0　152　75　71　15
に.wav　に　0　161　66　85　8
にぇ.wav　にぇ　0　186　67　77　15
にゃ.wav　にゃ　4　175　68　70　9
にゅ.wav　にゅ　2　132　63　50　7
にょ.wav　にょ　3　156　78　63　8
ぬ.wav　ぬ　0　141　67　66　15
ね.wav　ね　3　80　66　55　14
の.wav　の　0　138　86　63　17
は.wav　は　0　205　73　142　32
ば.wav　ば　36　98　70　50　0
周波数表の編集
周波数表を初期化
OK
キャンセル
名前　は.wav
エイリアス　は
オフセット　0　ミリ秒
子音部　205　ミリ秒
ブランク　73　ミリ秒
先行発声　142　ミリ秒
ｵｰﾊﾞｰﾗｯﾌﾟ　32　ミリ秒
セット
クリア
複製
エディタを起動
削除
設定は原音側に保存されます
「エイリアス」はファイル名を読み替えます。〈"sa.wav"→"さ"等〉
「オフセット」は原音ファイルの頭の無音部分を読み飛ばす設定です。
「子音部」には頭の伸縮させない部分の長さを指定します。
「ブランク」は終りの空白部分です。

# 原音設定ツール2



# エディタ画面


新規プロジェクト - UTAU (*)
ファイル(F) 編集(E) 表示(V) 検索(S) プロジェクト(P) 再生(L) ツール(T) ヘルプ(H)
Tempo 120.0 Quantize L32 32分音符 Length L4 4分音符
Mode2 Trace
Lyric はあるのおがわわさらさらながる
MIDI
デフォルト
0
1
2
120.00
G5
C5
C4
100 は
100 あ
100 る
100 の
100 お
100 が
100 わ
100 わ
100 さ
100 ら
P
0.500000 sec
7.750000 sec
M

# 開発経緯と『歌わせる』為の基本的技法

1. 人力ボーカロイドとは
2. UTAU開発に至った経緯

# 人力ボーカロイドとは

人力VOCALOIDとは、人物やキャラクターのアカペラ素材を人力で切り貼りし、まるでVOCALOIDのように歌わせる技術のことである。

各人物・キャラクターにおける固有の名称については「人力VOCALOIDの一覧」を参照のこと。

## 概要

初音ミクに代表されるVOCALOIDは、音階と歌詞を入力することで機械処理で自動的に歌ってくれるツールであるが、
中には「自分の好きなキャラクターに歌ってもらいたい！」「発売まで待ちきれない！」等の衝動に駆られるときがある。

人力VOCALOIDとは、そんな強い意志を抱いた賢者が、"本来歌うことを想定せずに収録した声"もしくは
"既に完成した歌"などから、歌詞および音階を切り出して、切り張り編集などをして歌わせる技術である。

このジャンルにおいては、ゆっきPが「鏡音リン・レン」発売前に双海亜美・真美のボーカル音源を
Audacityで切り貼り編集した「【人力Vocaloid】鏡音リンが「恋のミクル伝説」を歌ってくれた」で一世を風靡した。

近年、人力VOCALOIDの技術は向上しており、より自然に滑らかに歌わせている作品が増えてきている。

だが、何故か権利申し立てにより削除されてしまっている物も存在する。

※初心者にも作りやすい人力VOCALOIDの支援ツールが存在する。→UTAU

※類義語として「暴歌ロイド」タグが存在する。自然さよりもネタに特化した作品に付けられることが多い。

※極論を言えば、人力VOCALOIDは個人ユーザーによる切り貼りの作業の果てに作られた物の為、普通は権利削除はされないはずなのだが、使っている曲(？)等によっては、削除対象になるようである。→例:ドリ音P作【人力VOCALOID】ハートキャッチ☆パラダイス！【真】が2011年9月5日に権利申し立てにより削除されている。

## 関連動画

▼人力Vocaloidのはじまり

ニコニコ動画 →WATCH

再生：52,988 コメント：1,367 マイリスト：392

07/09/14 14:15 投稿

VOCALOID持ってないのでHARUCALOIDに歌わせてみた(未完成版)

1:39

21世紀の生んだ文明の利器VOCALOIDに、無謀にも手作業で挑んだ暇人の軌跡。元と

ニコニコ動画 →WATCH

再生：59,076 コメント：1,009 マイリスト：998

07/11/10 18:10 投稿

アイドルマスター　ハルカロイドとミキロイドを一緒に歌わせてみた

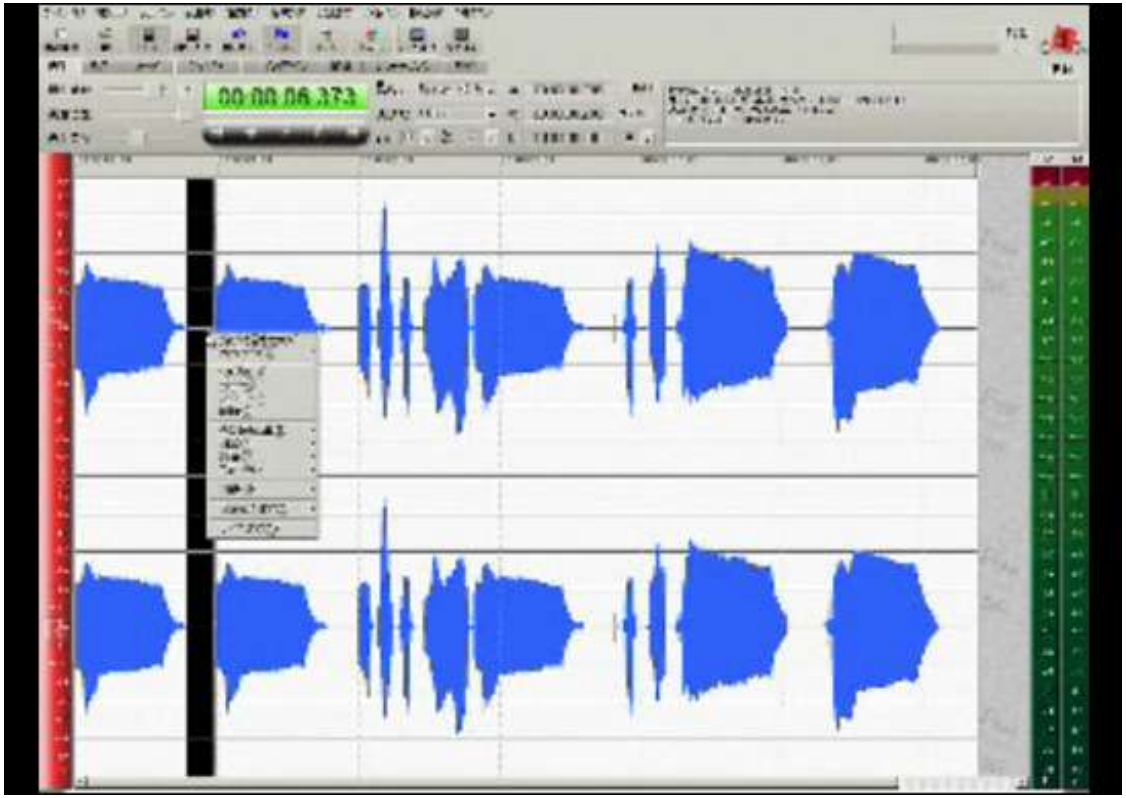
1:37

春香＋美希で200万パワー！ハモリを加えて400万パワー！そして前作の3倍の分量を作れば400万×3の……1200…

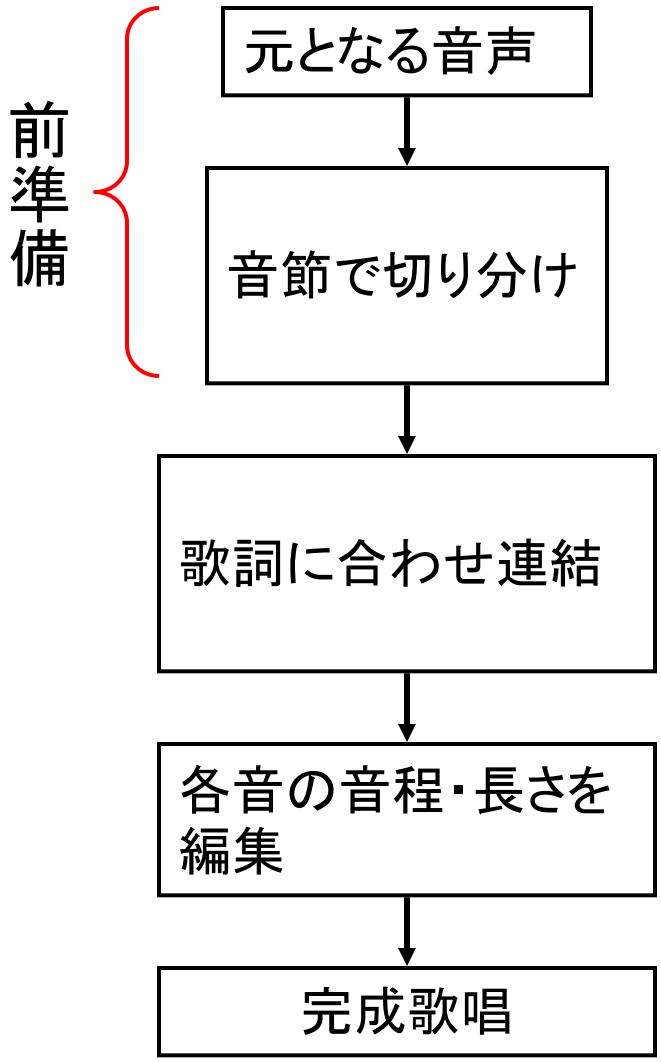
# 発端となった動画

『初音ミクを無料ソフトだけで作ってみた』

http://www.nicovideo.jp/watch/sm5165688



# 人力ボーカロイド概要




アナウンス・朗読、
ドラマやアニメの音声
既存曲からボーカル抽出等

あ.wav、い.wav、う.wav等の
ように歌詞を組み立てる為の
素片に分解する

サウンド編集ソフトを使って

まず歌詞を組み立てる

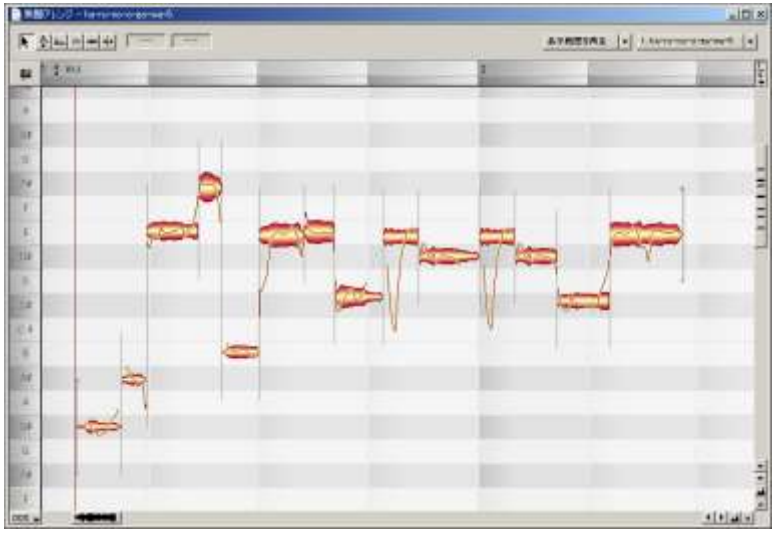
ボーカル補正ソフトを使って音
程と音の長さを曲に合わせる

これを伴奏等とMIXして曲を
完成させる

# 使用したツール

## Audacity

フリーのサウンド編集ソフト
シンプルだが使い勝手は良い

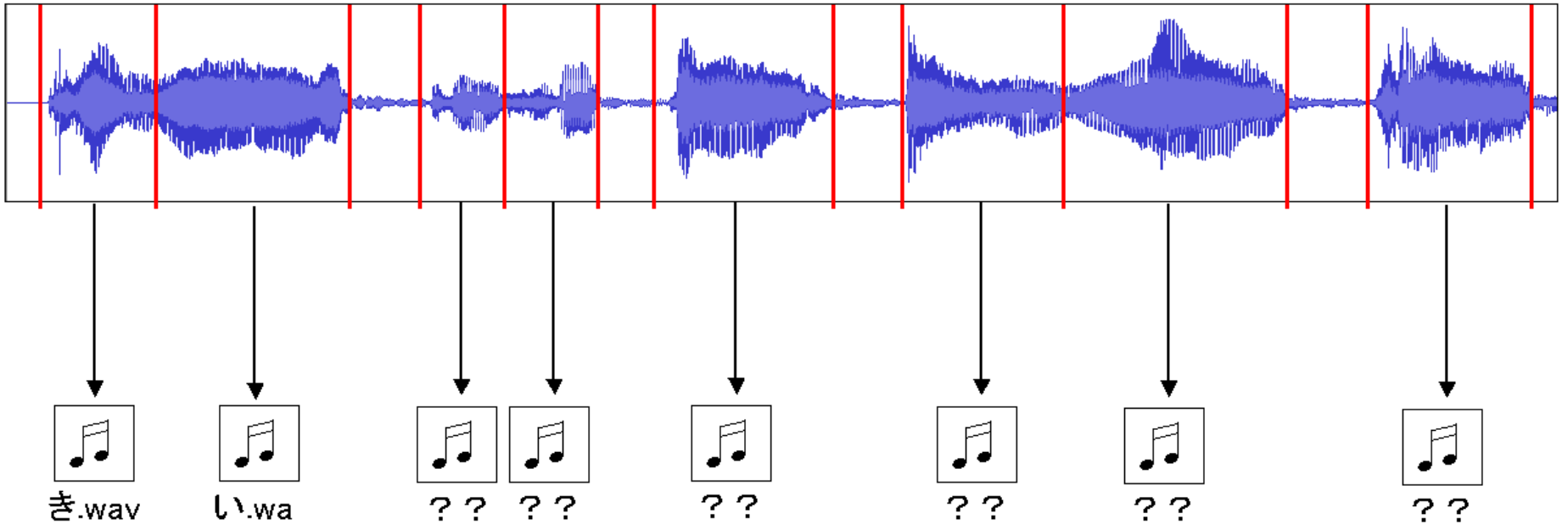


## Melodyne demo版

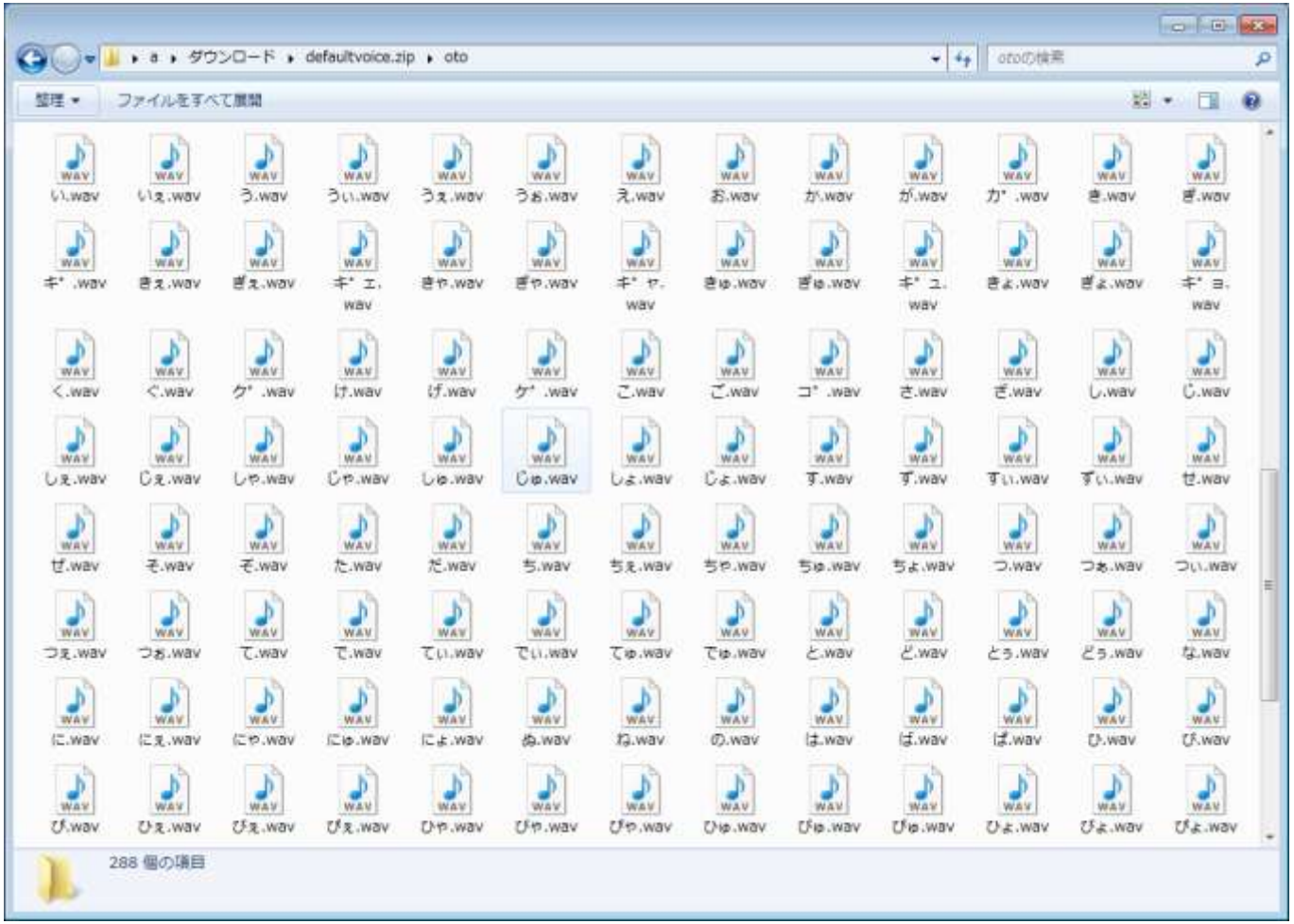
ピッチ修正やテンポ変更・タイミング調整等が行えるサウンド編集ソフト

**※Melodyneといえば和音分解機能が有名だが、当時のは単音用のソフトでそのような機能は搭載されていなかった。**

# 手順1:元音声を音節で切り分ける



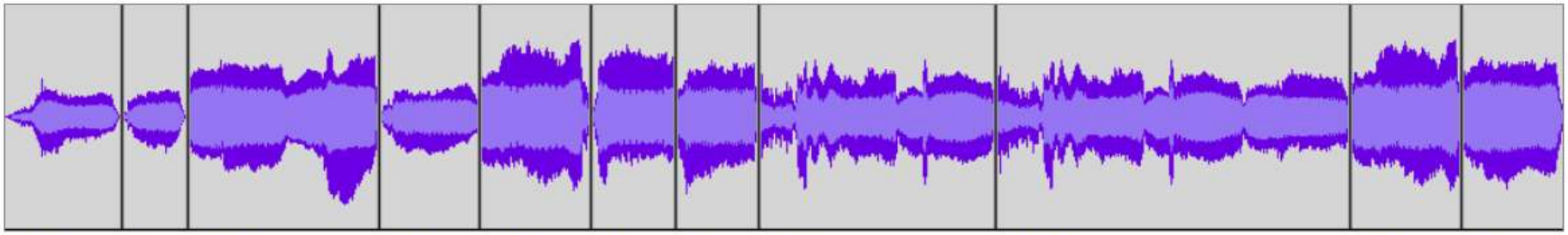
# 切り分けた音源ファイルの例



# 日本語で汎用的に歌わせる為に必要な音節

| あ | か | さ | た | な | は | ま | や | ら | わ | ん | が | ざ | だ | ば | ぱ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| い | き | し | ち | に | ひ | み |  | り |  |  | ぎ | じ |  | び | ぴ |
| う | く | す | つ | ぬ | ふ | む | ゆ | る |  |  | ぐ | ず |  | ぶ | ぷ |
| え | け | せ | て | ね | へ | め |  | れ |  |  | げ | ぜ | で | べ | ぺ |
| お | こ | そ | と | の | ほ | も | よ | ろ | を |  | ご | ぞ | ど | ぼ | ぽ |
|  | きゃ | しゃ | ちゃ | にゃ | ひゃ | みゃ |  | りゃ |  |  | ぎゃ | じゃ |  | びゃ | ぴゃ |
|  | きゅ | しゅ | ちゅ | にゅ | ひゅ | みゅ |  | りゅ |  |  | ぎゅ | じゅ |  | びゅ | ぴゅ |
|  | きょ | しょ | ちょ | にょ | ひょ | みょ |  | りょ |  |  | ぎょ | じょ |  | びょ | ぴょ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | きぇ | しぇ | ちぇ | にぇ | ひぇ | みぇ |  | りぇ |  |  | ぎぇ | じぇ |  | びぇ | ぴぇ |
| ふぁ | ふぃ | ふぇ | ふぉ |  |  |  |  |  |  |  |  |  |  |  |  |
| いぇ | うぃ | うぇ | うぉ |  |  |  |  |  |  |  |  |  |  |  |  |
| つぁ | つぃ | つぇ | つぉ |  |  |  |  |  |  |  |  |  |  |  |  |
| すぃ | てぃ | てゅ | とぅ |  |  |  |  |  |  |  |  |  |  |  |  |
| ずぃ | でぃ | でゅ | どぅ |  |  |  |  |  |  |  |  |  |  |  |  |
| ヴぁ | ヴぃ | ヴ | ヴぇ | ヴぉ |  |  |  |  |  |  |  |  |  |  |  |

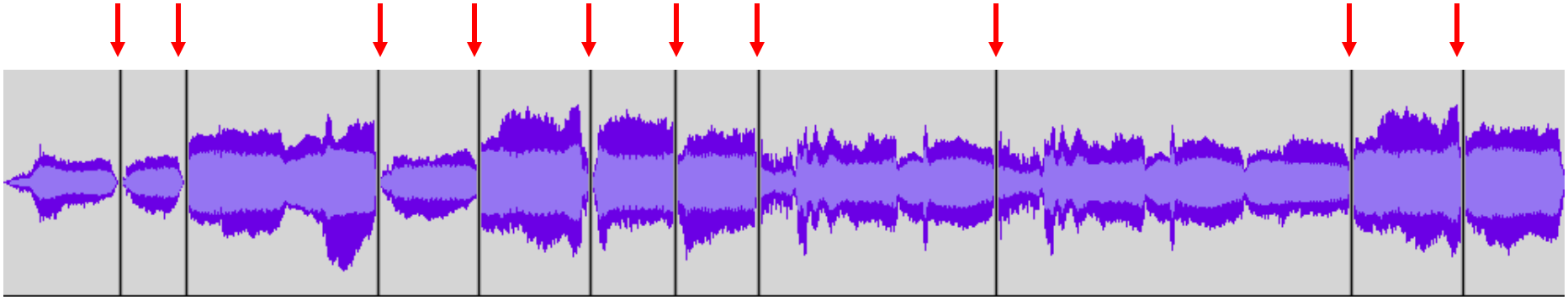
# 手順2：歌詞に合わせて連結



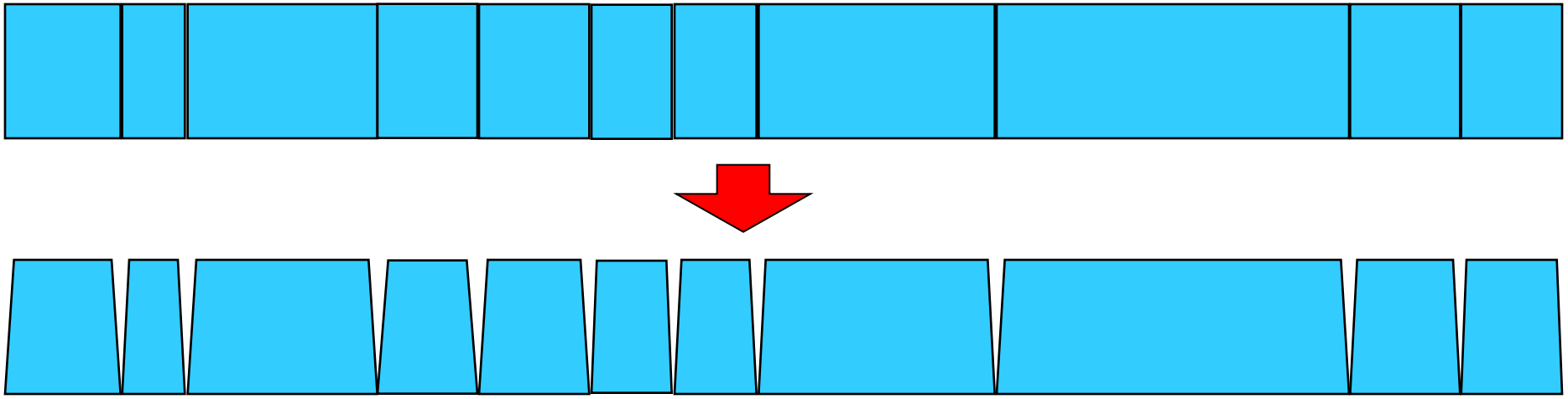

※元歌詞：
「行けるの」

※元歌詞：「まっさらな」

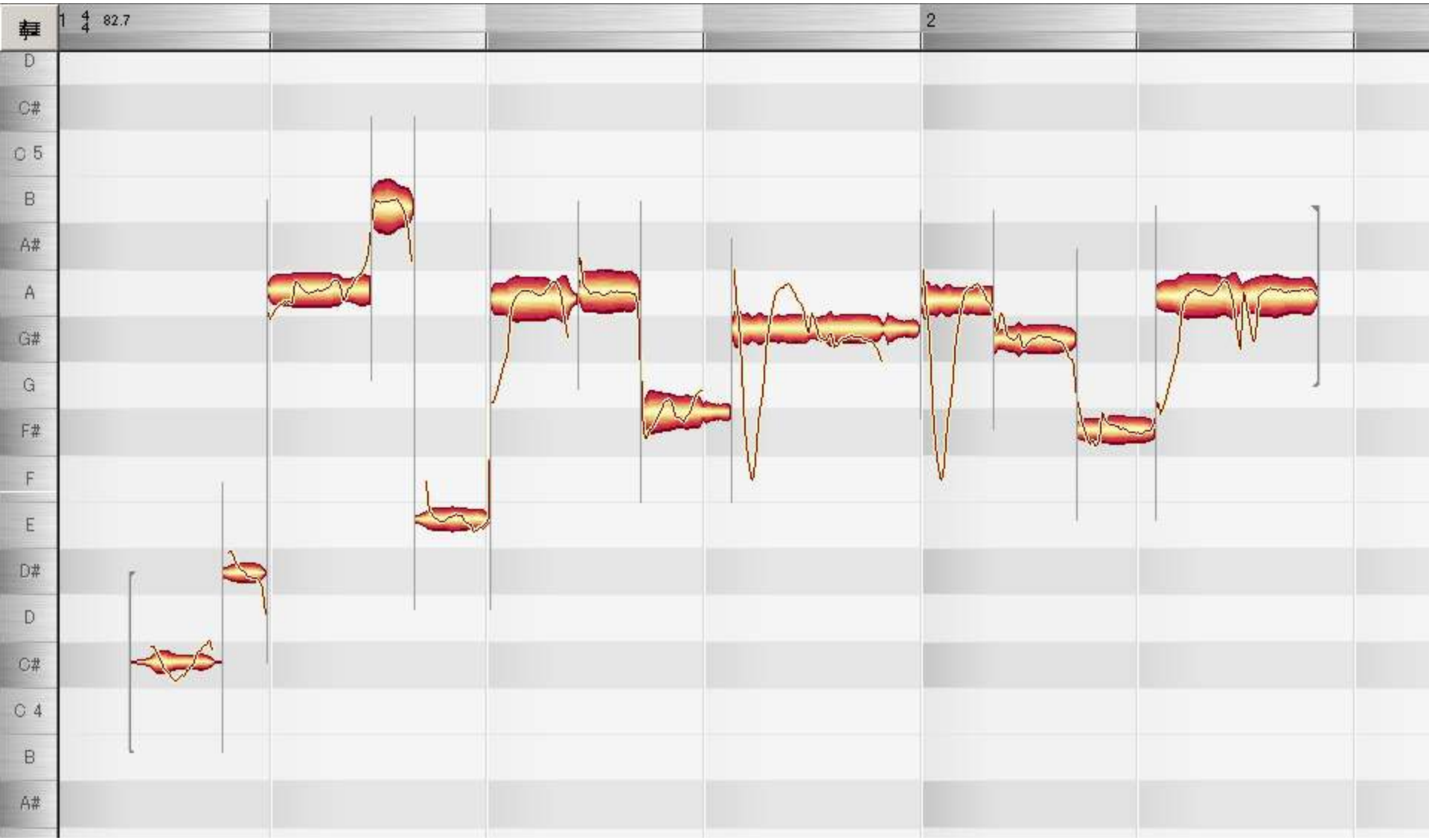
つなぎ目にノイズが入るので



素片毎に細かくフェードイン・フェードアウト



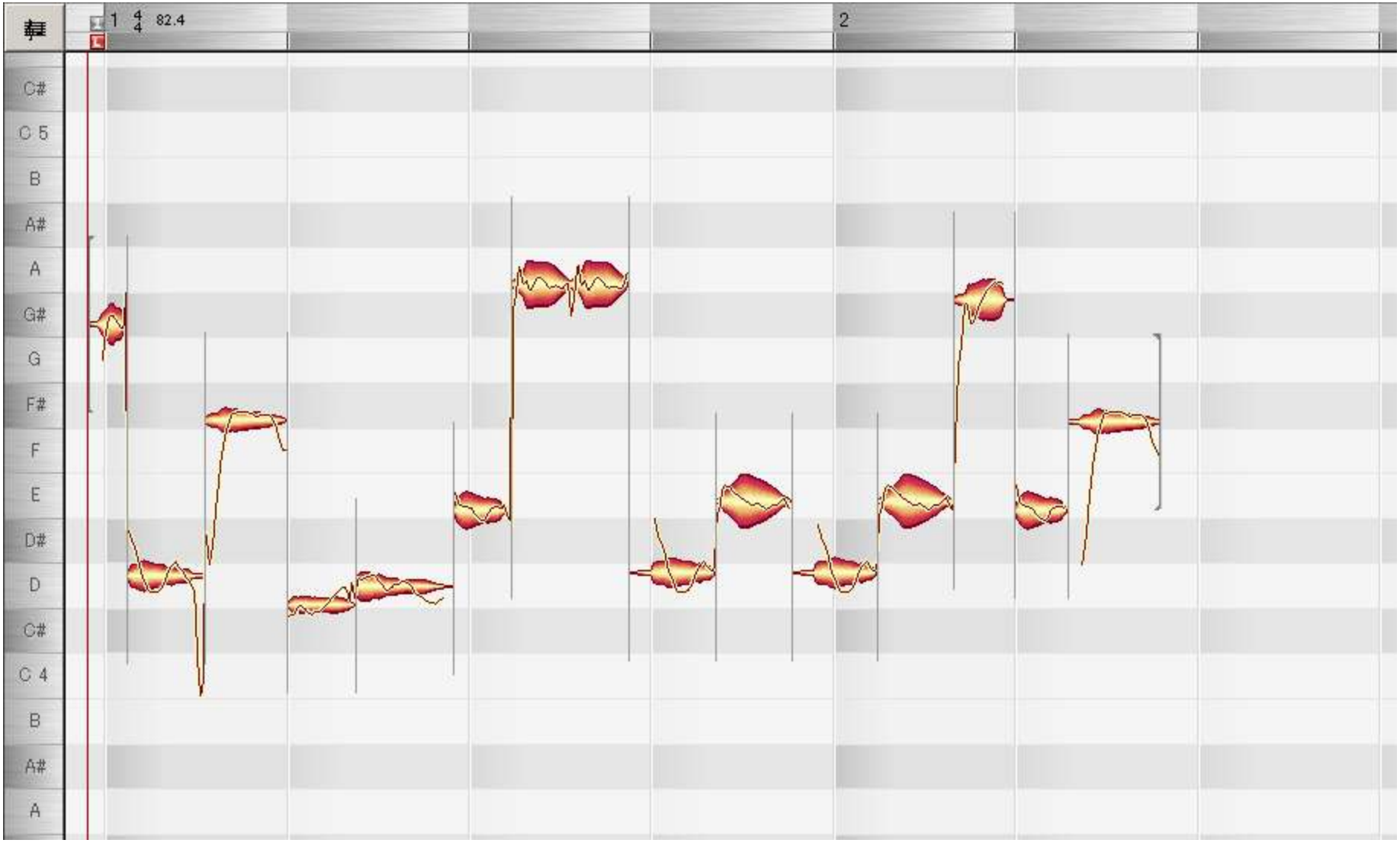
# 手順３：ボーカル修正ソフトで編集



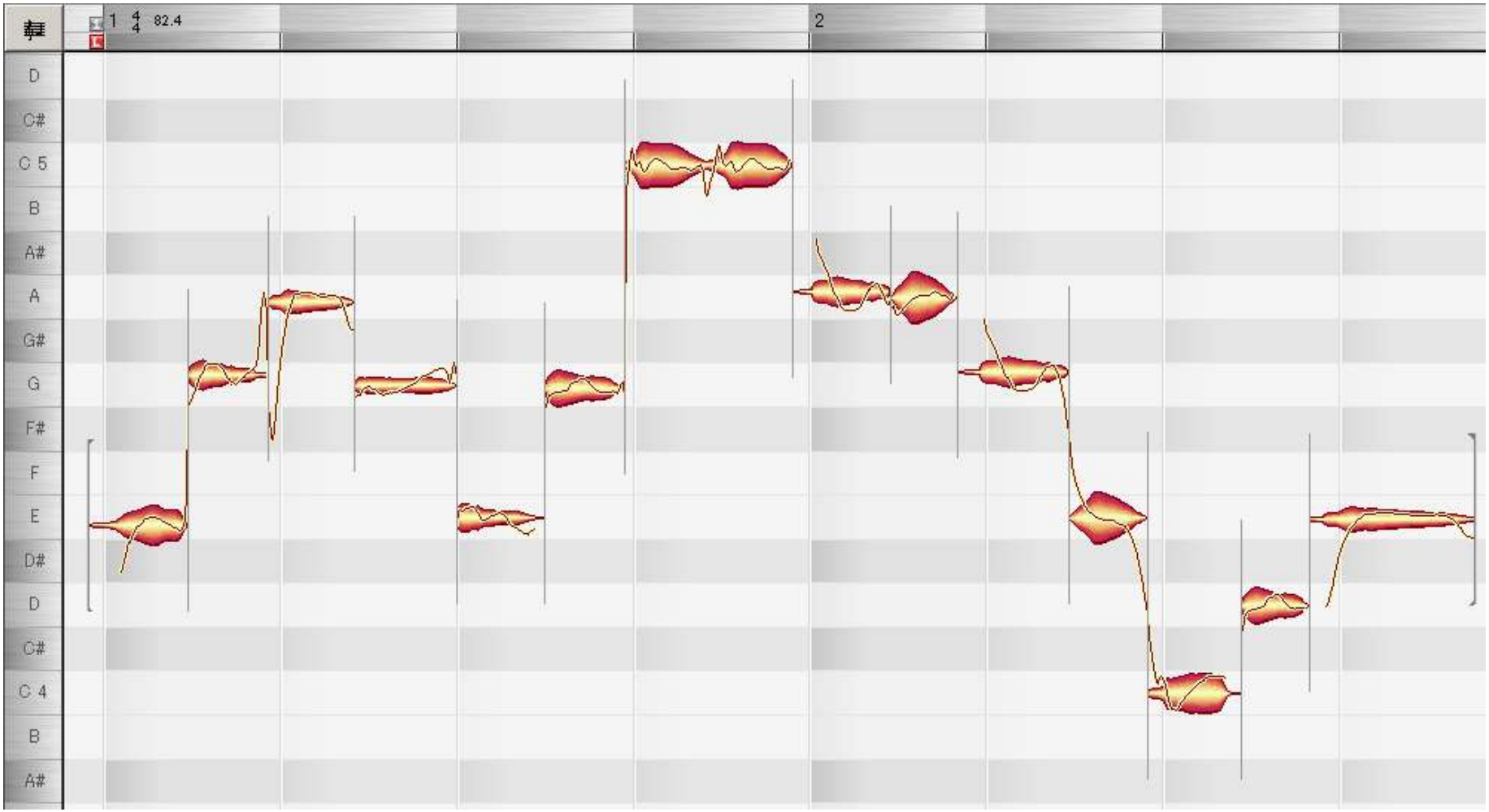

※違う曲から切り出しているの音程も長さも全く合ってない

# 音程と長さを編集後


# 例2:編集前



# 例2:編集後

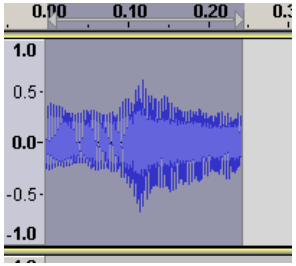


# UTAU開発に至った経緯

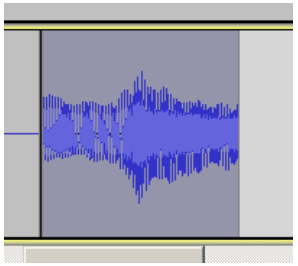
# 人力ボーカロイドは作業効率が悪い

- 編集単位が細かく似た操作の繰り返し
- 殆どがマウス操作で効率化しづらい
- 一部差し替え等の修正が手軽にできない
- 音節ファイルの管理・検索が面倒

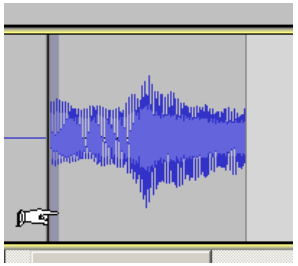
# Audacityでの操作



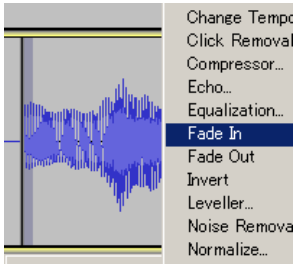

1．元の波形を
読み込んでコピー



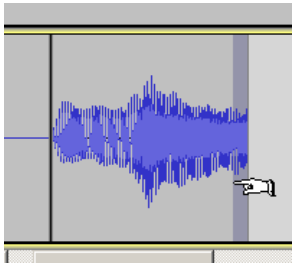
2．作業ファイルに
貼り付ける



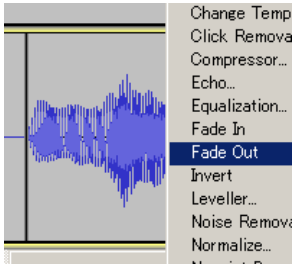
3．マウスで
範囲選択



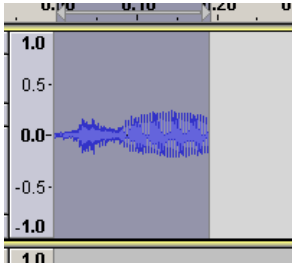

4．Effectメニュー
→「Fade In」



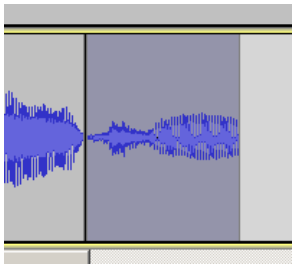
5．マウスで
範囲選択




6．Effectメニュー
→「Fade Out」




7．次の波形を
読み込んでコピー



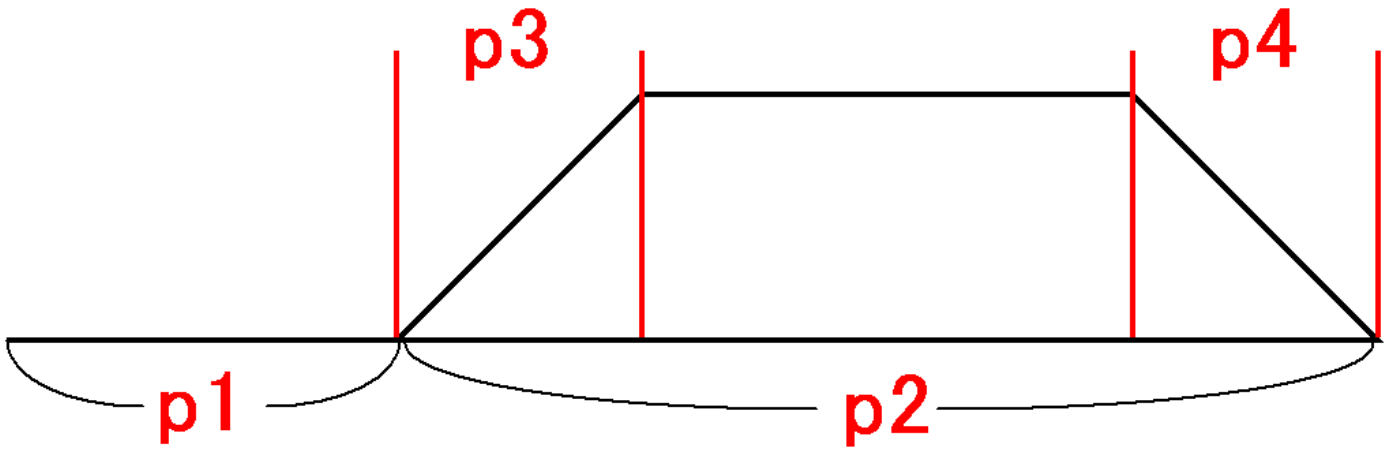
8．次を貼り
付ける

※このような作業を音節の数だけ繰り返す必要がある

# 音節の切り張りを自動化

- 元音声ファイルから位置と長さを指定して切り出し、フェードインとフェードアウト加工を施して出力ファイルに追加するプログラムを作成した。

使い方：
wavtool 入力ファイル 追加先ファイル p1 p2 p3 p4

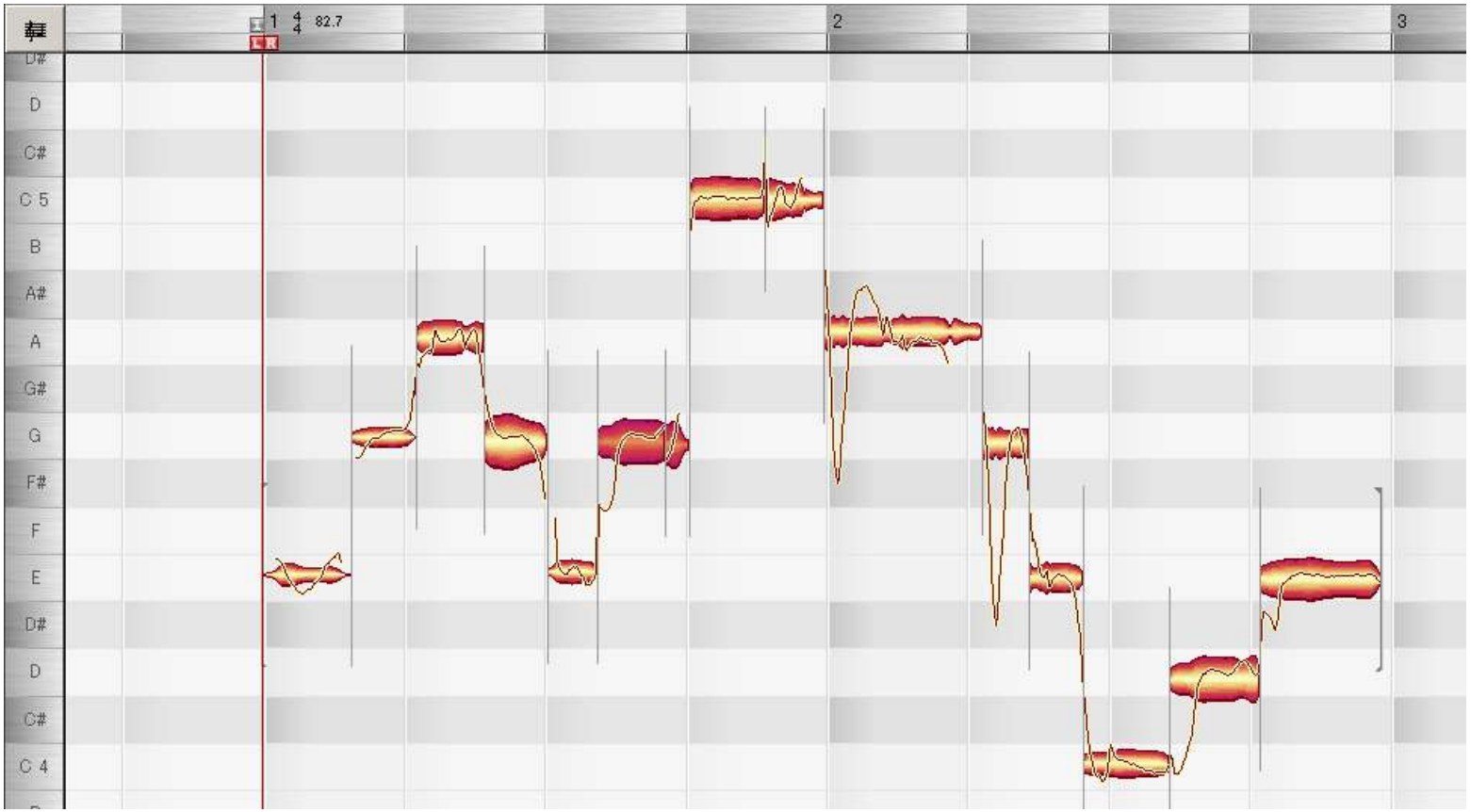


# バッチファイルを作成して一気に処理する

```
wavtool は.wav output.wav 0 250 12 24
wavtool あ.wav output.wav 0 250 12 24
wavtool る.wav output.wav 0 250 12 24
wavtool の.wav output.wav 0 250 12 24
wavtool お.wav output.wav 0 250 12 24
wavtool が.wav output.wav 0 250 12 24
wavtool わ.wav output.wav 0 250 12 24
wavtool わ.wav output.wav 0 250 12 24
wavtool さ.wav output.wav 0 250 12 24
wavtool ら.wav output.wav 0 250 12 24
wavtool さ.wav output.wav 0 250 12 24
wavtool ら.wav output.wav 0 250 12 24
wavtool な.wav output.wav 0 250 12 24
wavtool が.wav output.wav 0 250 12 24
wavtool る.wav output.wav 0 375 12 24
```

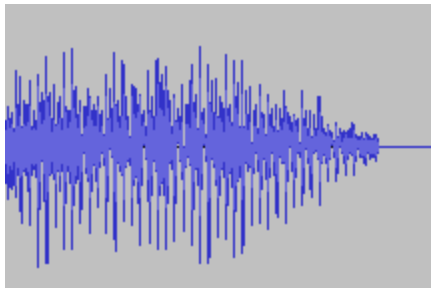
テキストでコピーペースト等が使えるのでAudacityのマウス操作と比べて作業時間が劇的に短縮できた

# ボーカル補正ソフトで行った処理も
# コマンド化できればもっと効率化できる

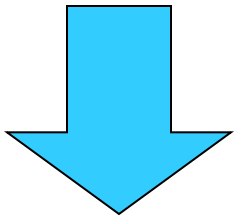



のではないか？

# 音程変更と長さの変更

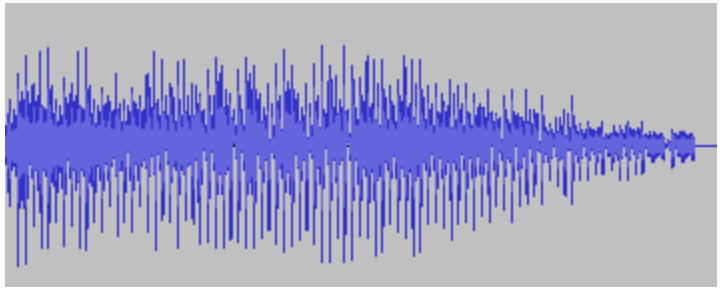


こういう元音声があったとき、



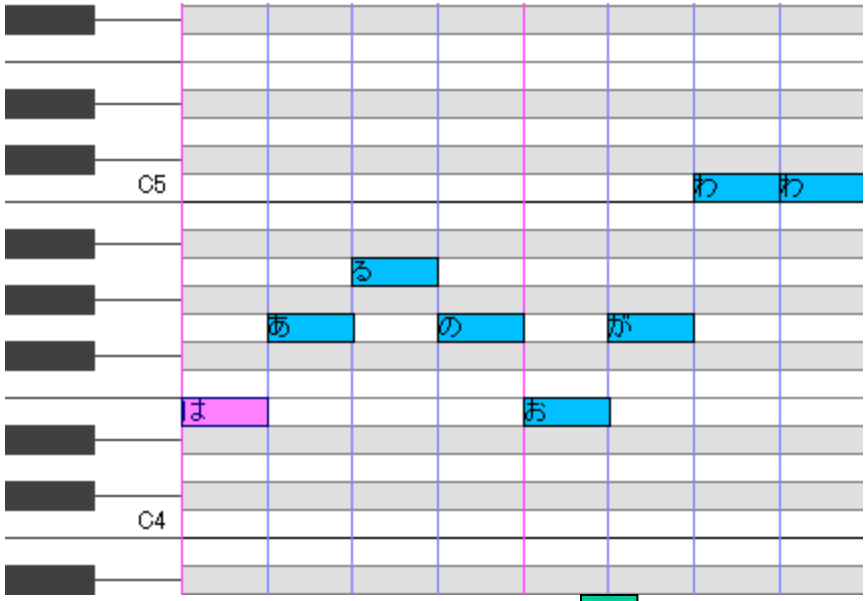
```
>resamp 元ﾌｧｲﾙ 出力ﾌｧｲﾙ 音程 長さ
```

このようにパラメータを指定して



任意の長さ・任意の音程に変更できるコマンドを作ればよい

# UTAUの原型




バッチファイルが複雑になったので、

```
resamp は.wav tmp.wav E4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp あ.wav tmp.wav G4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp る.wav tmp.wav A4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp の.wav tmp.wav G4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp お.wav tmp.wav E4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp が.wav tmp.wav G4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp わ.wav tmp.wav C5 500
wavtool tmp.wav output.wav 0 500 12 24
resamp わ.wav tmp.wav C5 500
wavtool tmp.wav output.wav 0 500 12 24
```

バッチファイルを生成する
GUIフロントエンドを作成した

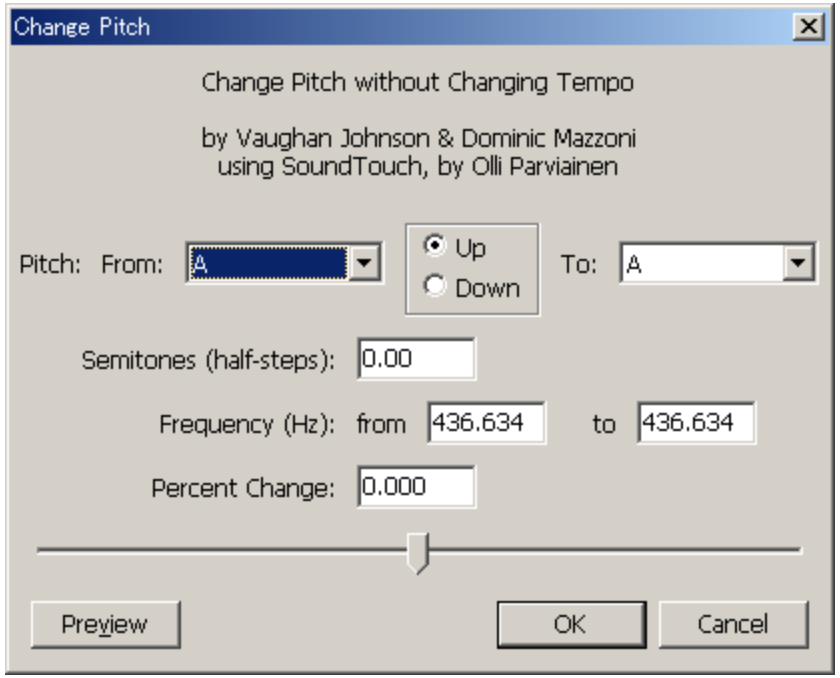
# 音声伸縮･音程変更の着想

- 音声合成の予備知識が無い作者が音声伸縮･音程変更プログラムを作成するにあたって辿った思考プロセスなど

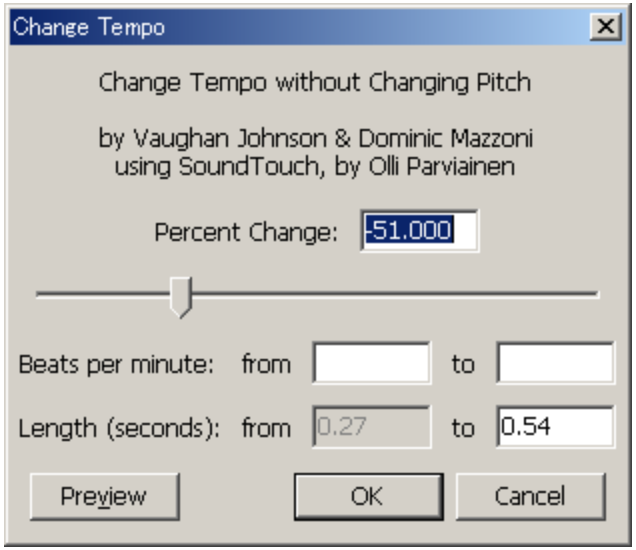
# 元の音声を好きな音程･長さに変更するには

- 音程を変えないで長さを変えられること
- 長さを変えないで音程を変えられること

# Audacityのピッチ変更／テンポ変更



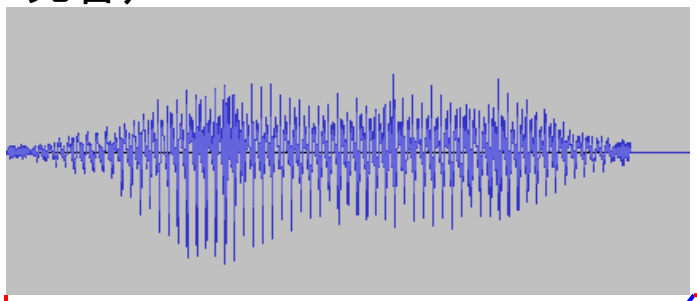

ピッチ変更




テンポ（長さ）変更

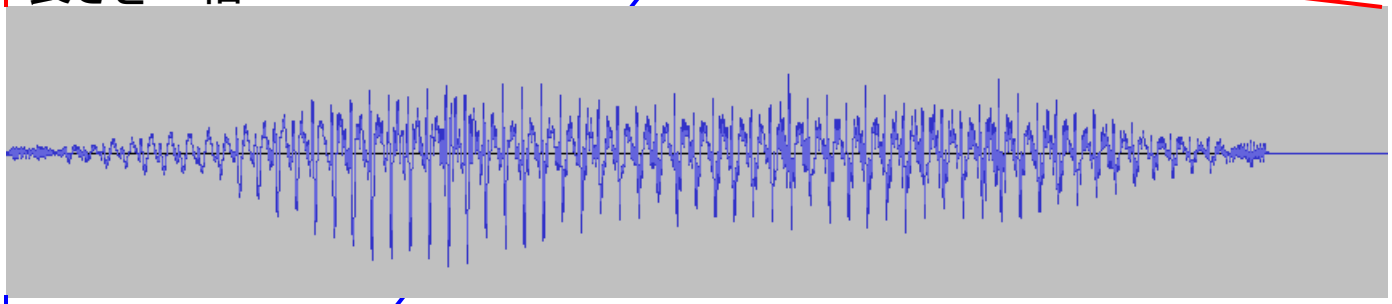
# 音声データの時間方向への伸縮について

元音声

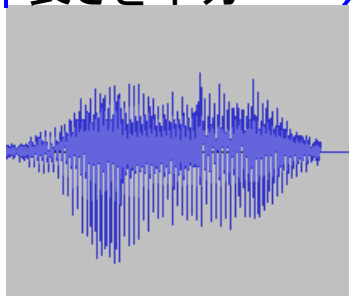


単純に引き伸ばすと声が低くなり、

長さを二倍

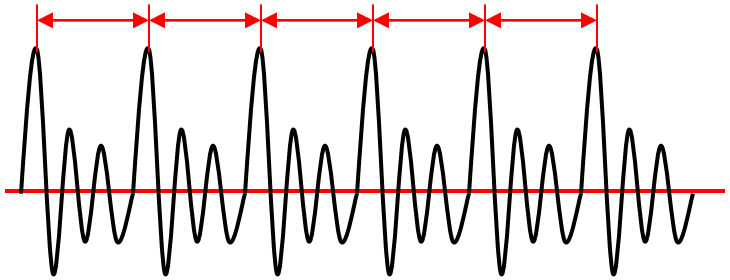


長さを半分



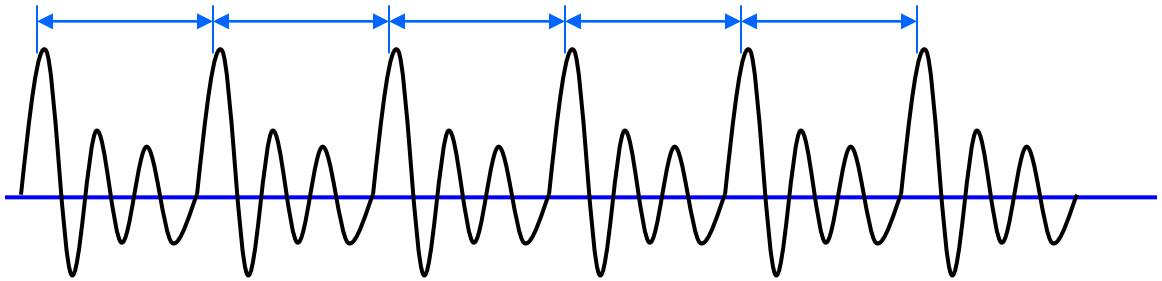
縮めると高くなる

# 単純に引き伸ばすと音程が低くなる

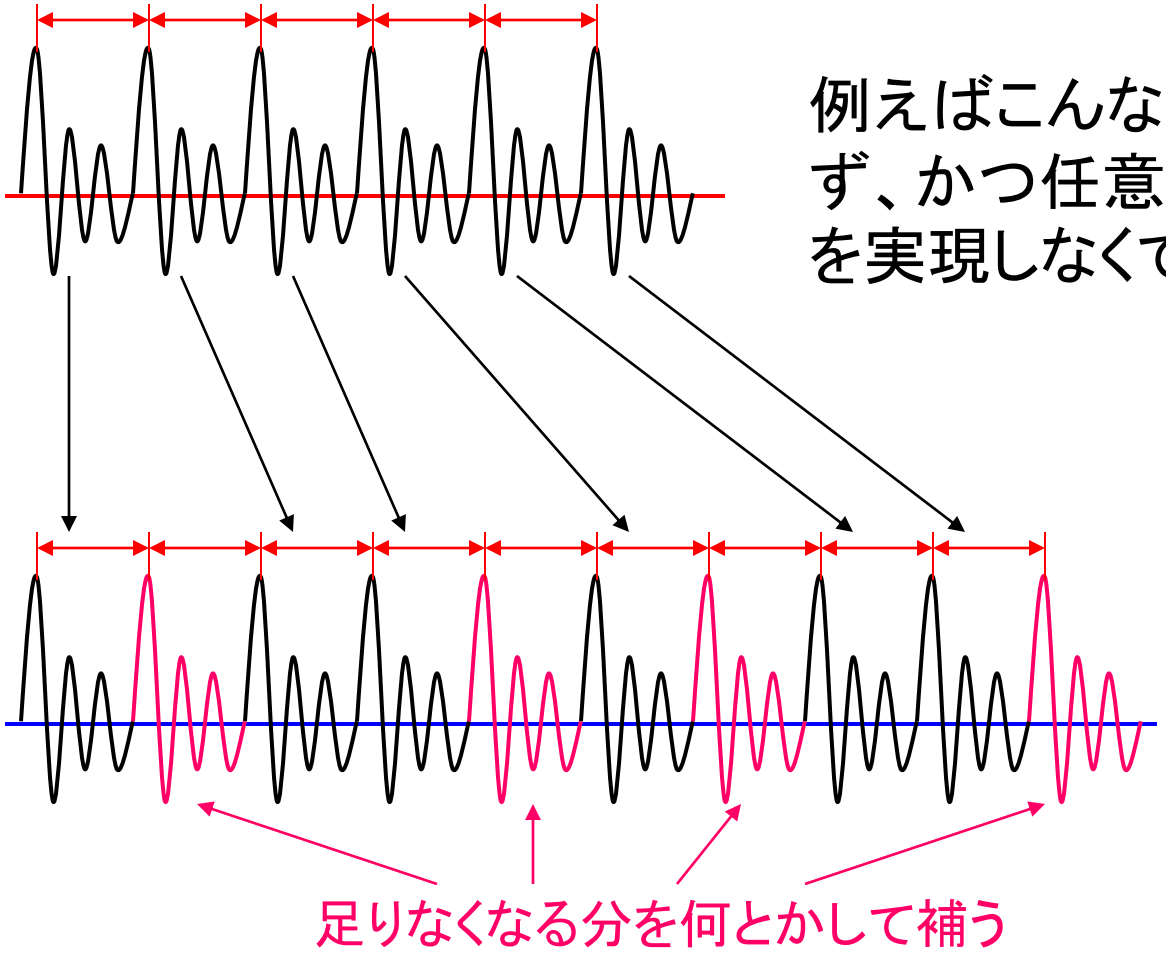


細かく見ると波の周期が音の高さを表している

単純な伸縮では周期の長さが変わってしまう

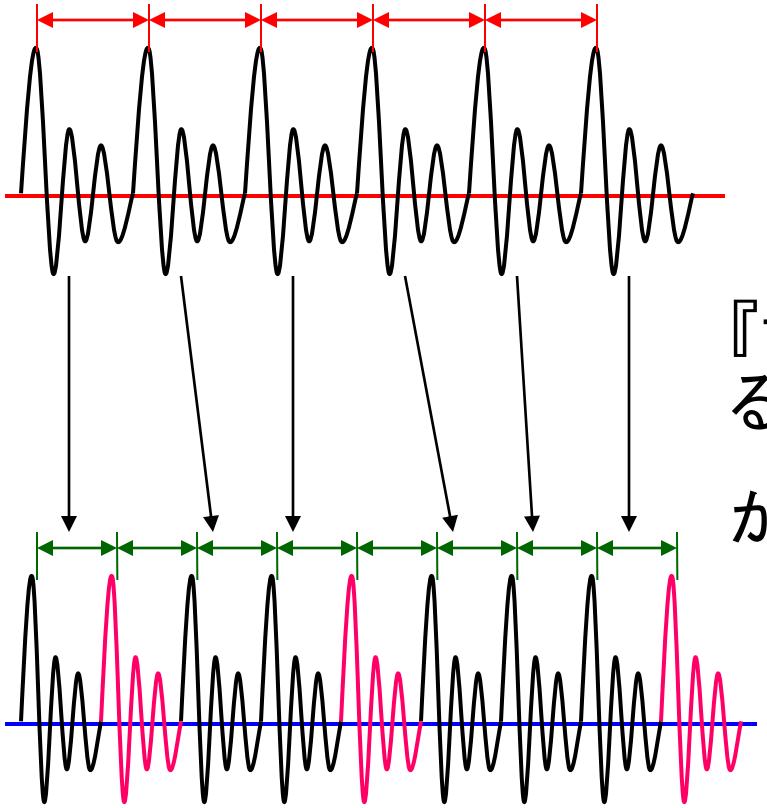


# 音程を変えないで長さを変えるには




例えばこんな風に周期を変えず、かつ任意の長さへの伸縮を実現しなくてはならない

# 長さを変えないで音程を変えるには

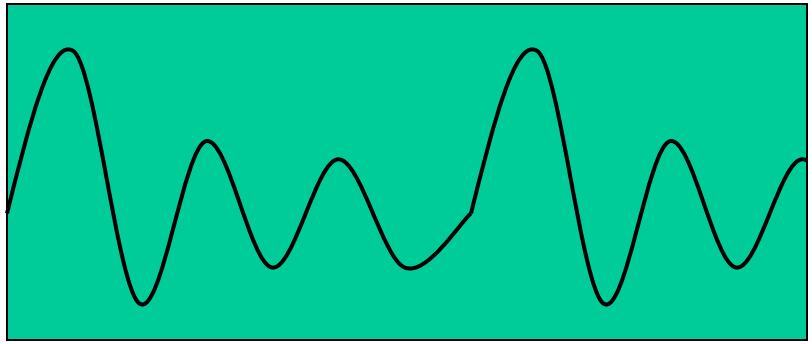


『音程を変えないで長さを変える』

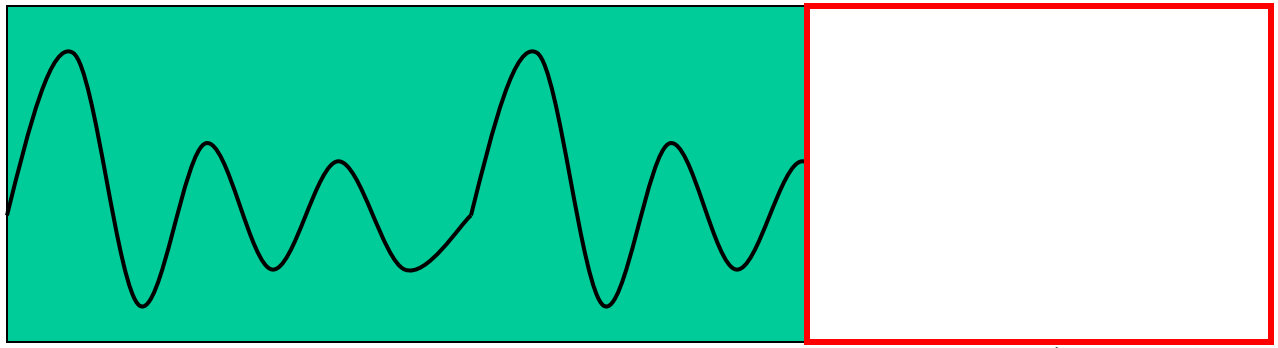
が出来ていればこちらも出来る

# そこで考えたのは、

例えばこういう波形があって



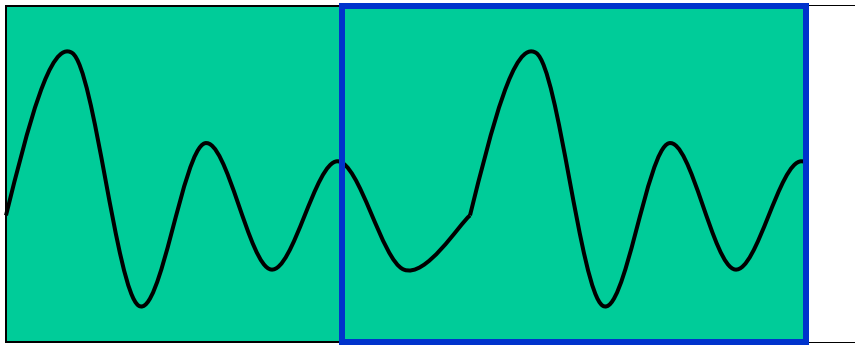
こうして引き伸ばしたい場合



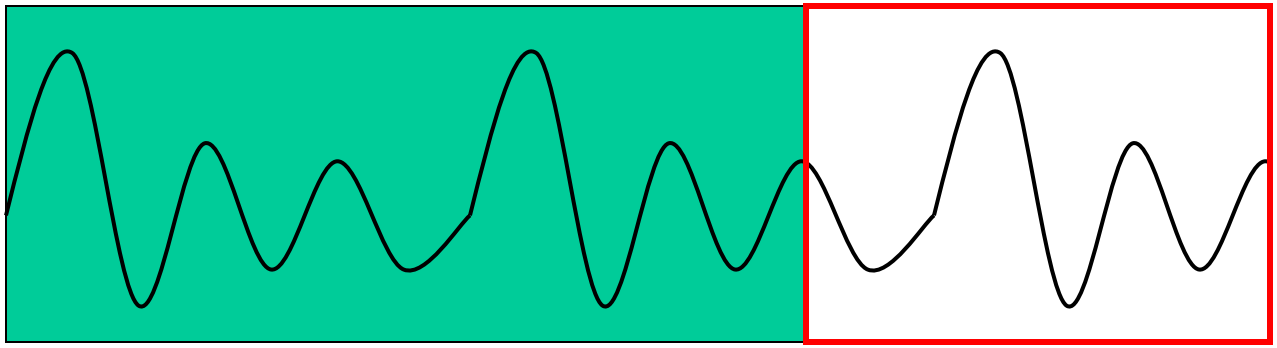
この足りない分は

# そこで考えたのは、

一周期前のこのあたりから

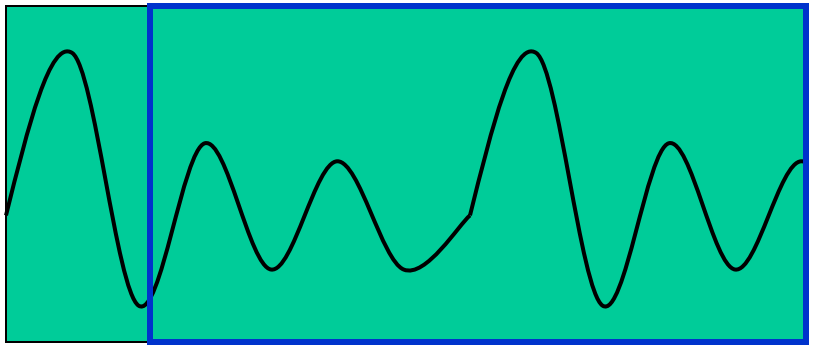


持ってくるしかないよね

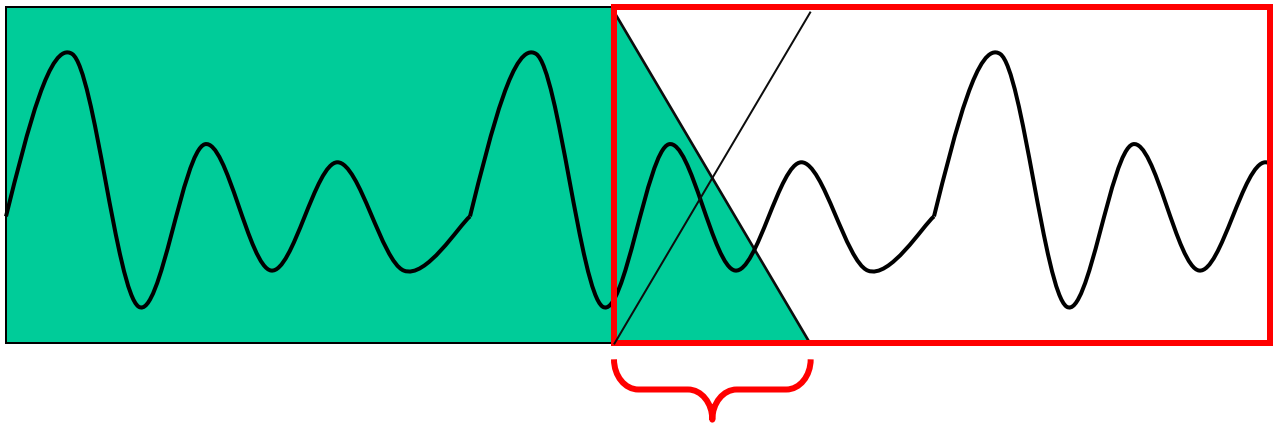


# そこで考えたのは、

でも実際はつなぎ目が綺麗にならないだろうから

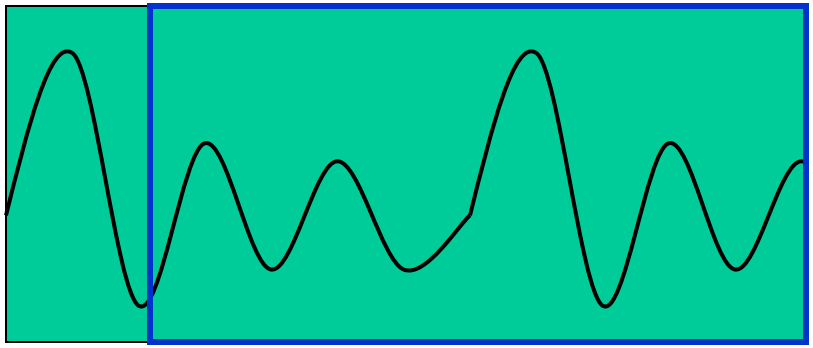


ある程度重ねてクロスフェードした方がいいよね

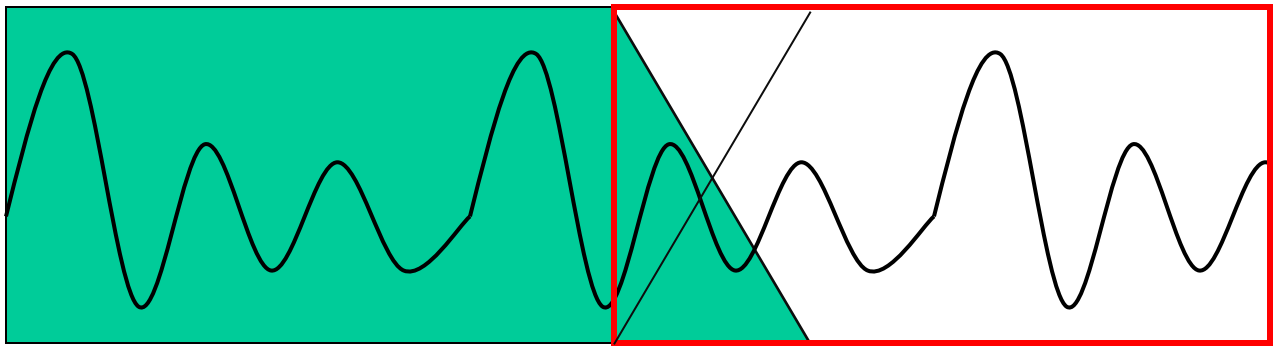


# そこで考えたのは、

でも実際はつなぎ目が綺麗にならないだろうから

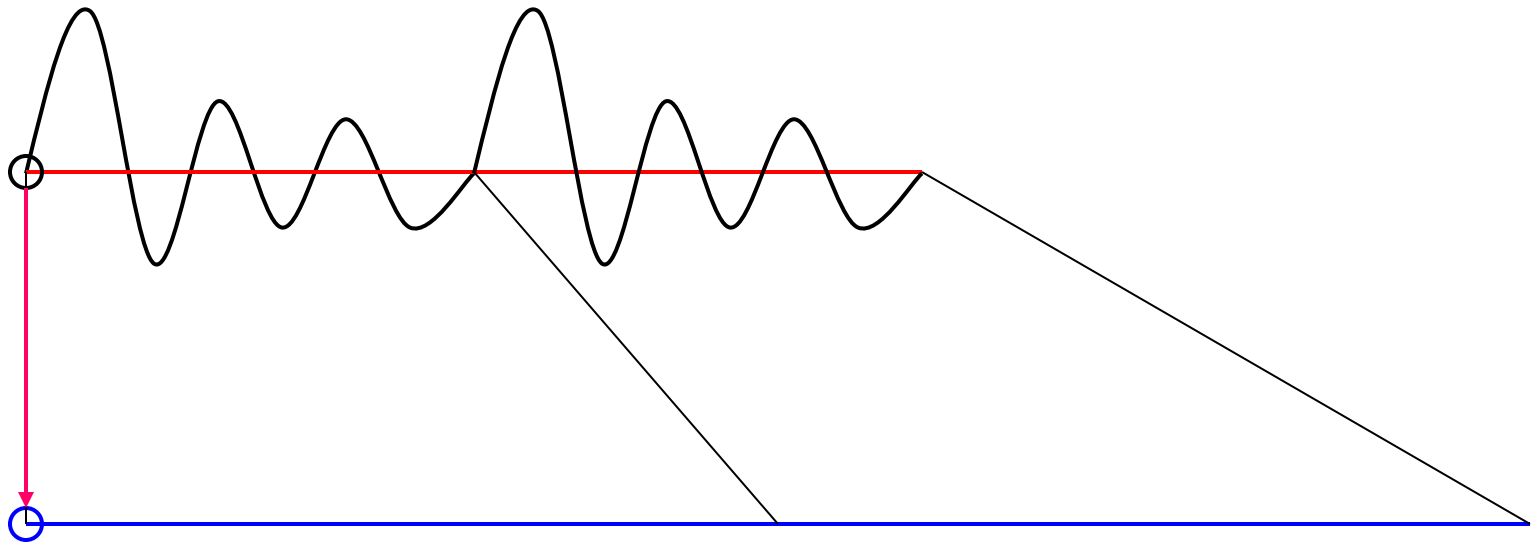


ある程度重ねてクロスフェードした方がいいよね



という発想でした

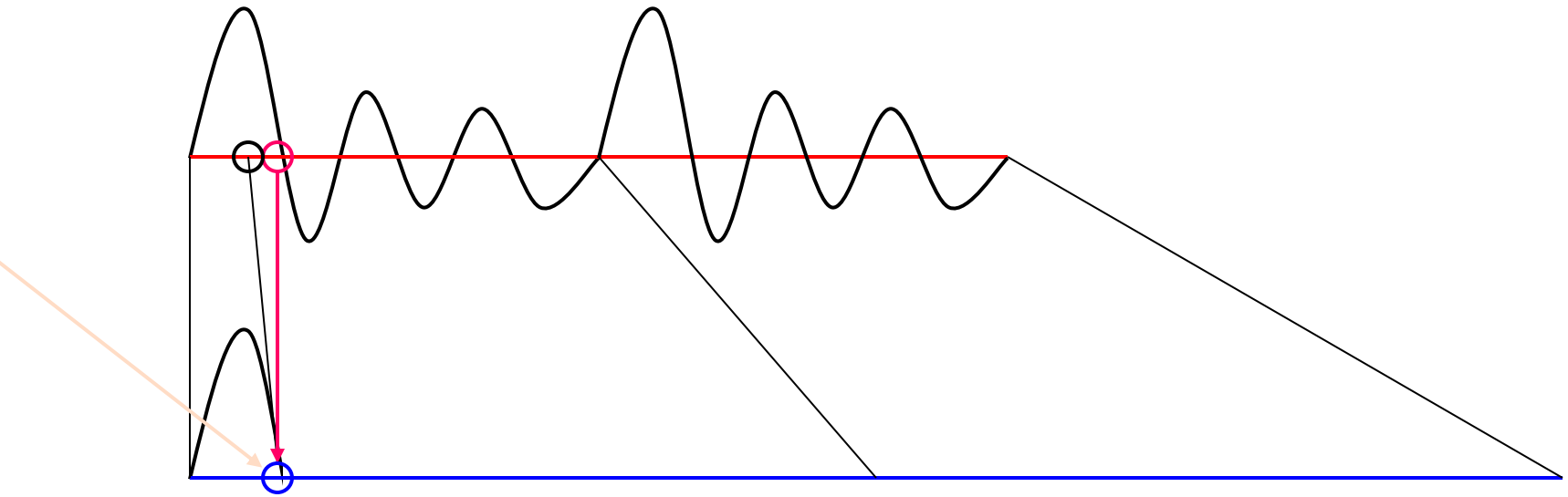
# 音声伸縮の一つの方法(01/17)



○　単純な伸縮によるコピー元サンプル位置

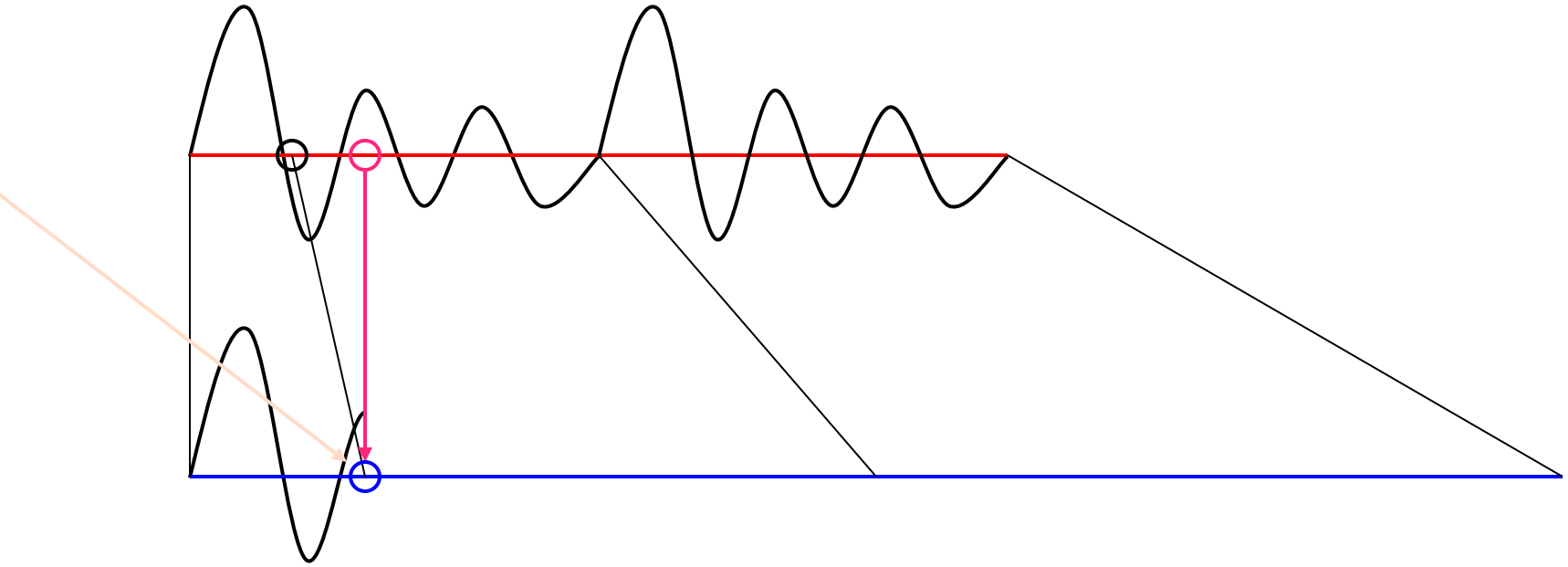
○ ○　実際にコピーするサンプル位置

# 音声伸縮の一つの方法(02/17)



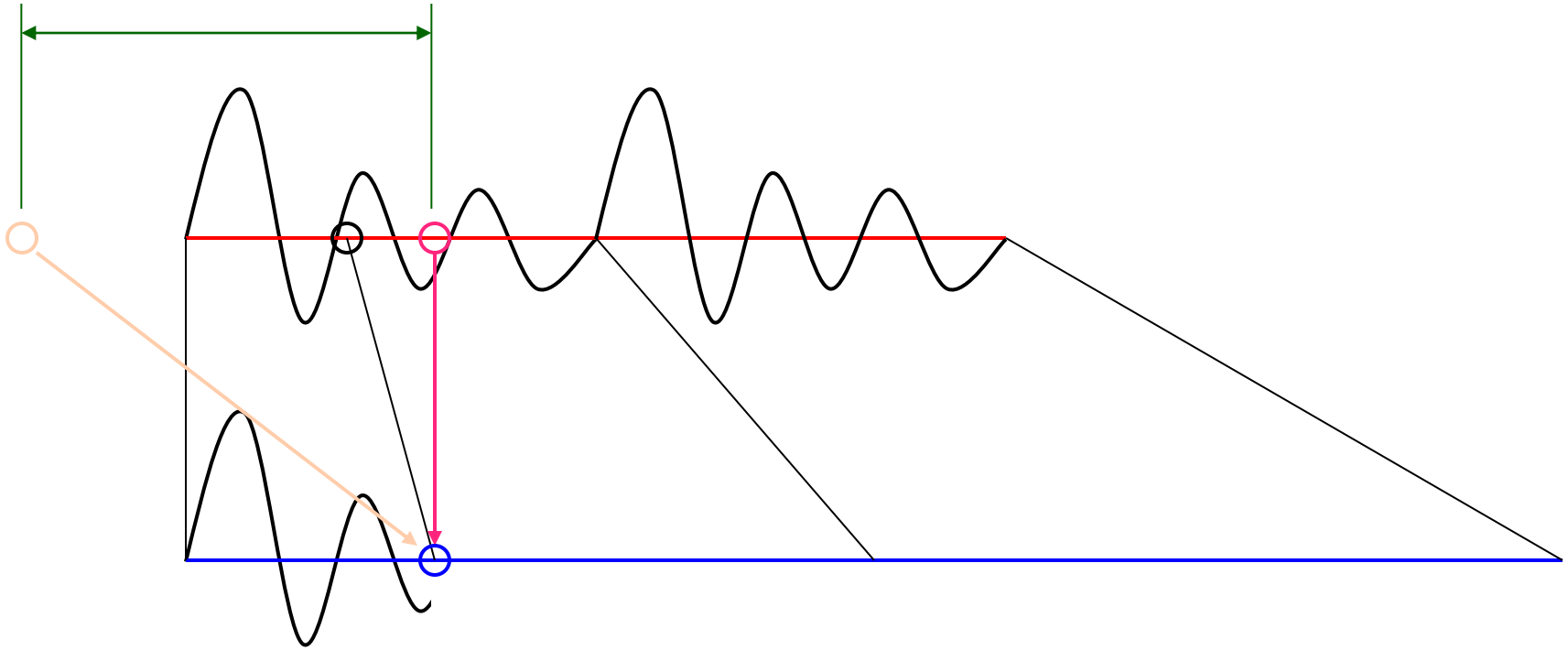
- ○ 単純な伸縮によるコピー元サンプル位置
- ○ ○ 実際にコピーするサンプル位置

# 音声伸縮の一つの方法(03/17)



- ○ 単純な伸縮によるコピー元サンプル位置
- ○ ○ 実際にコピーするサンプル位置

# 音声伸縮の一つの方法(04/17)

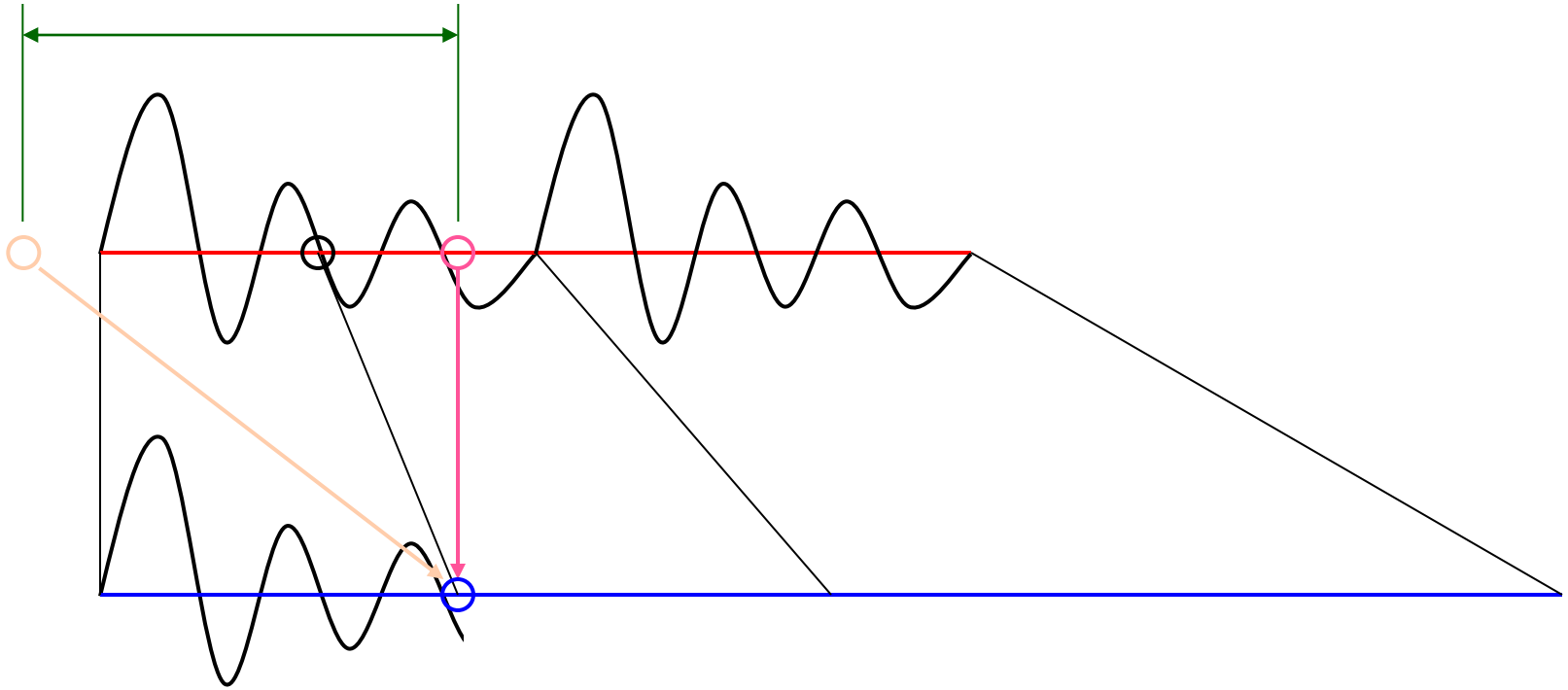


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(05/17)

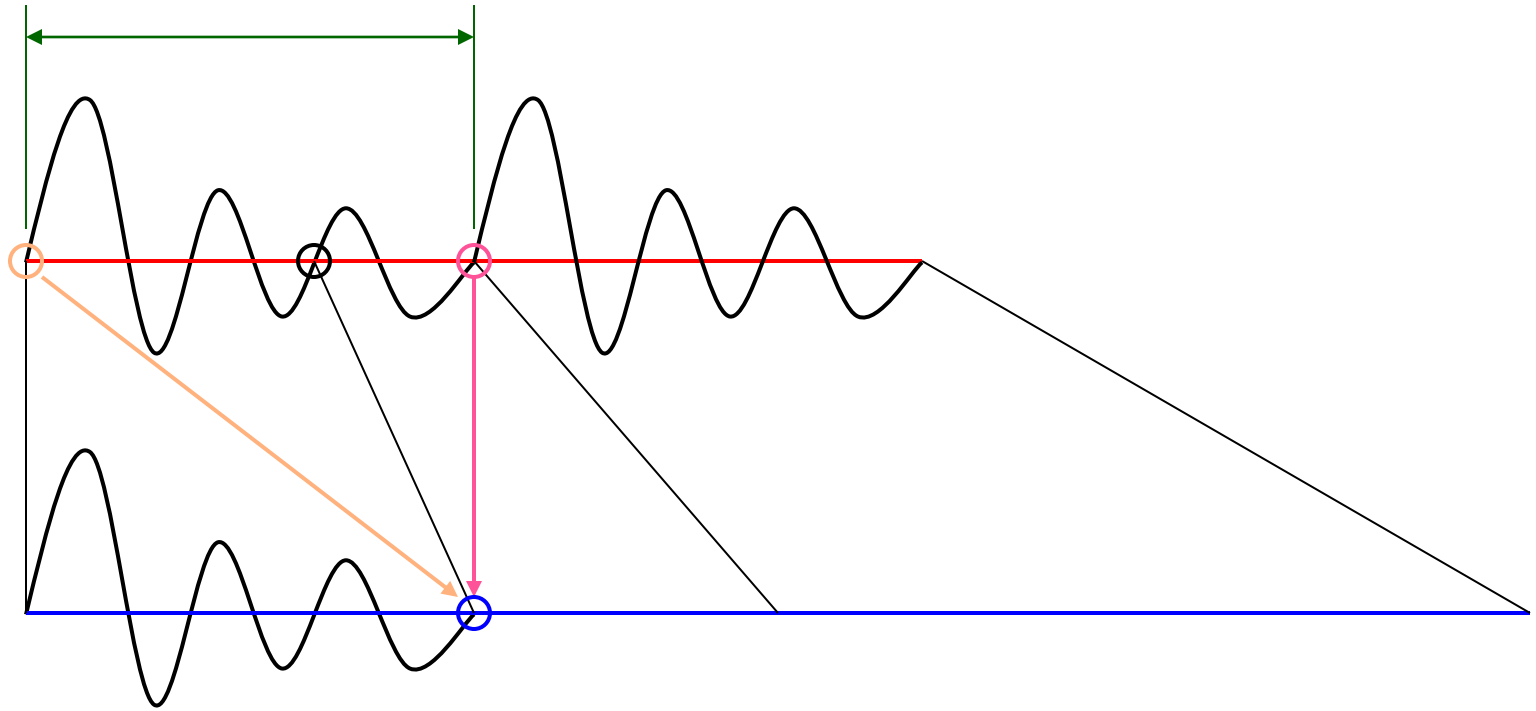


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(06/17)

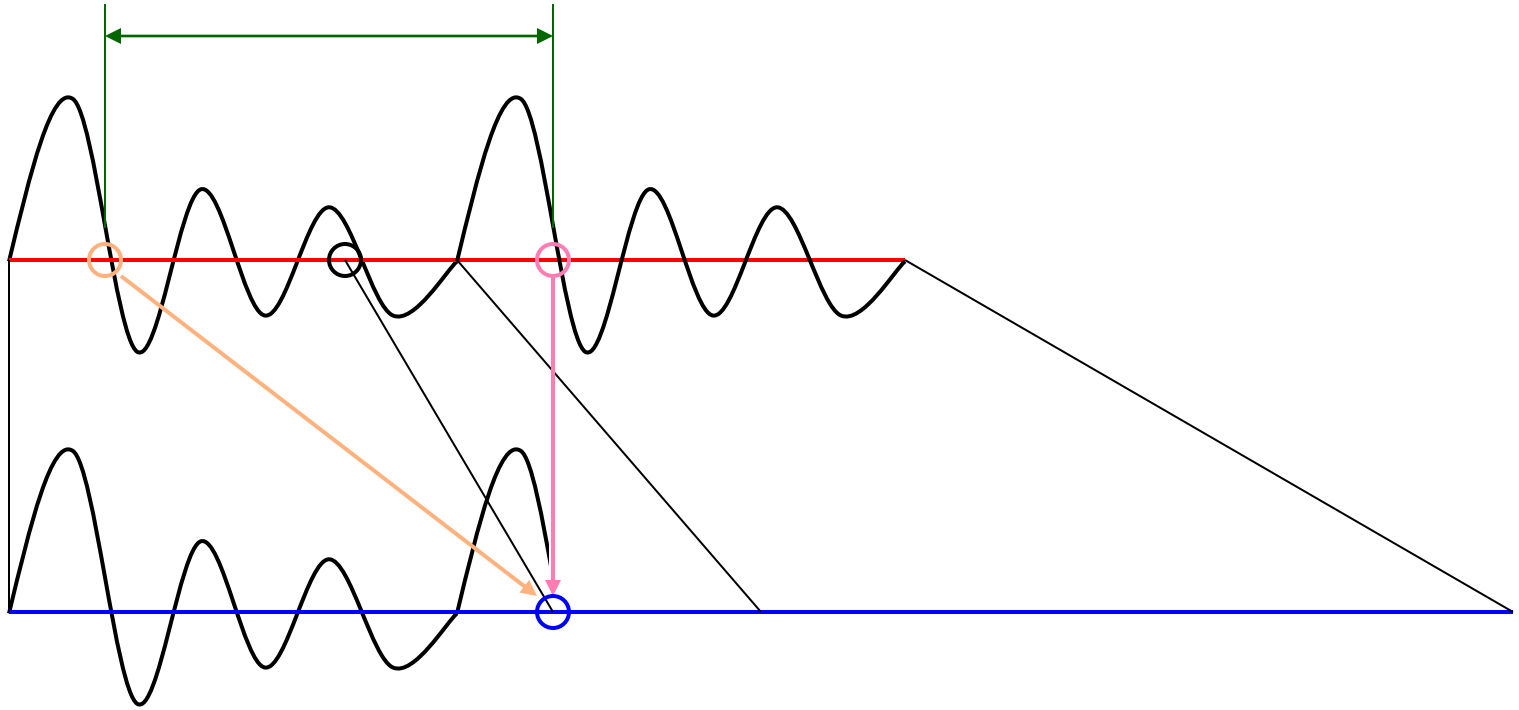


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(07/17)

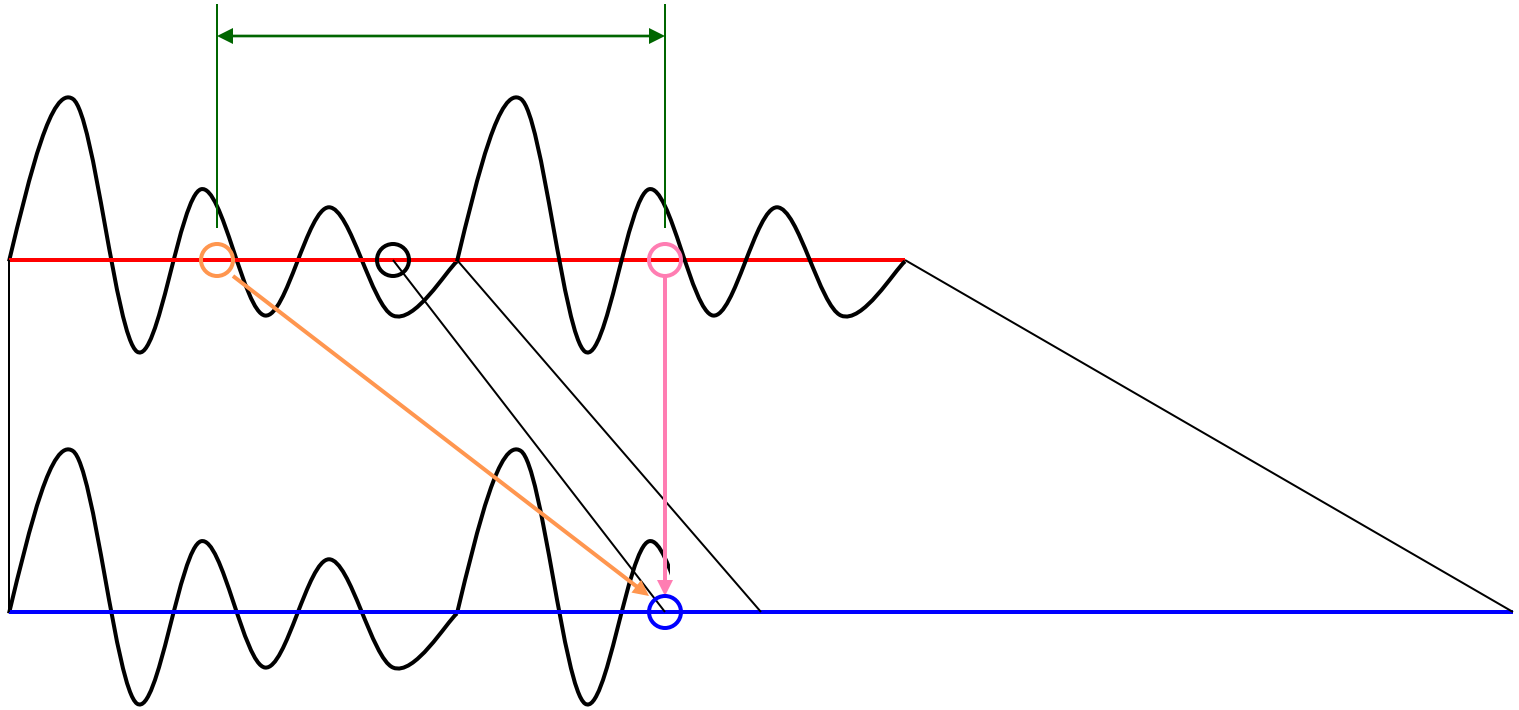


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(08/17)

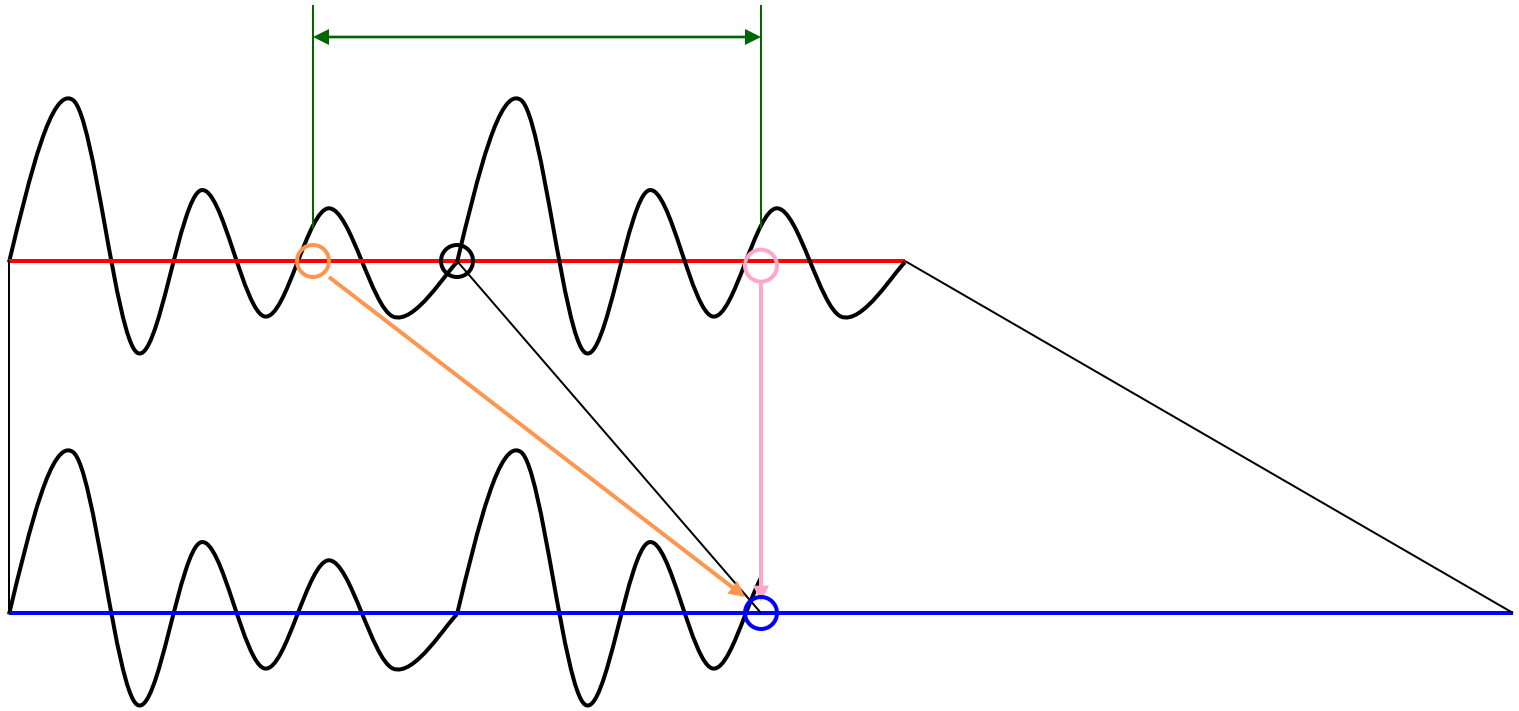


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(09/17)

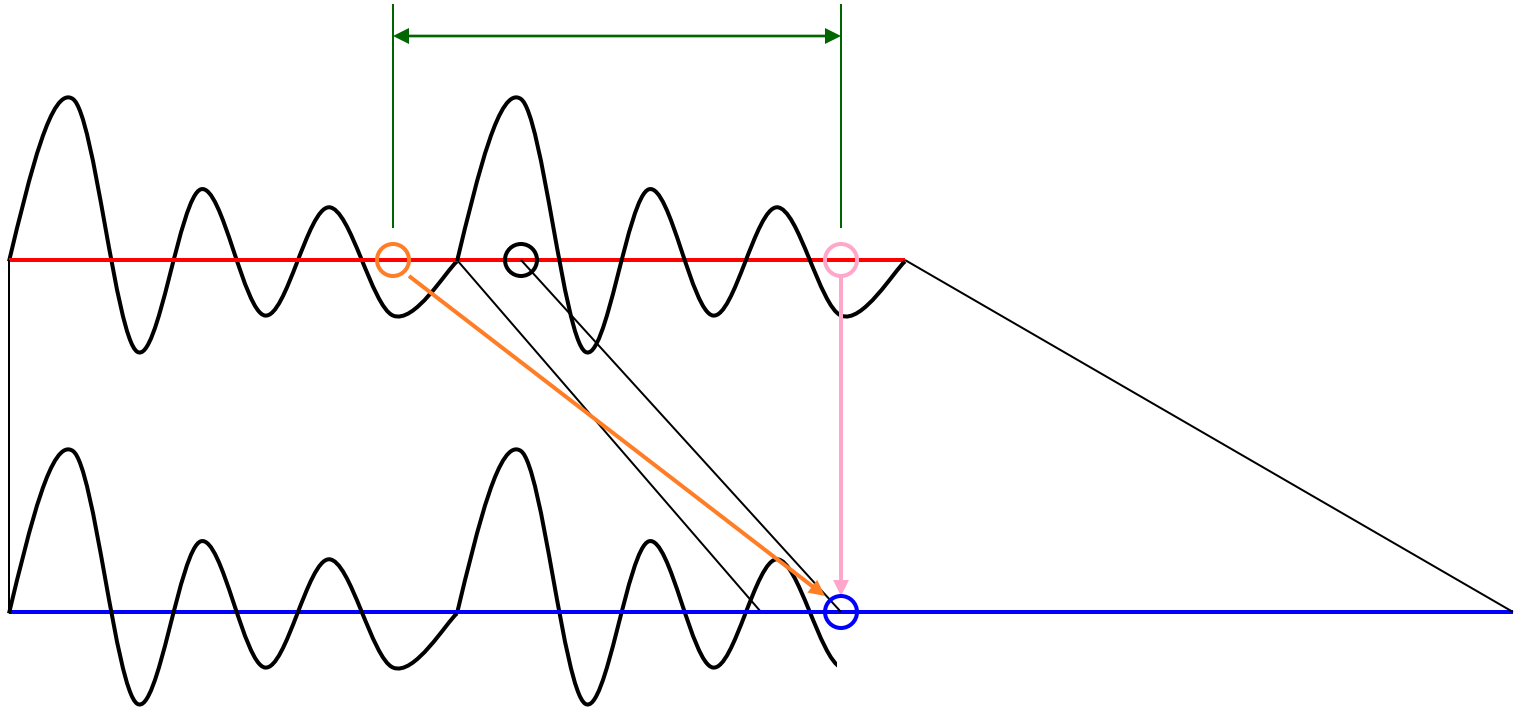


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(10/17)

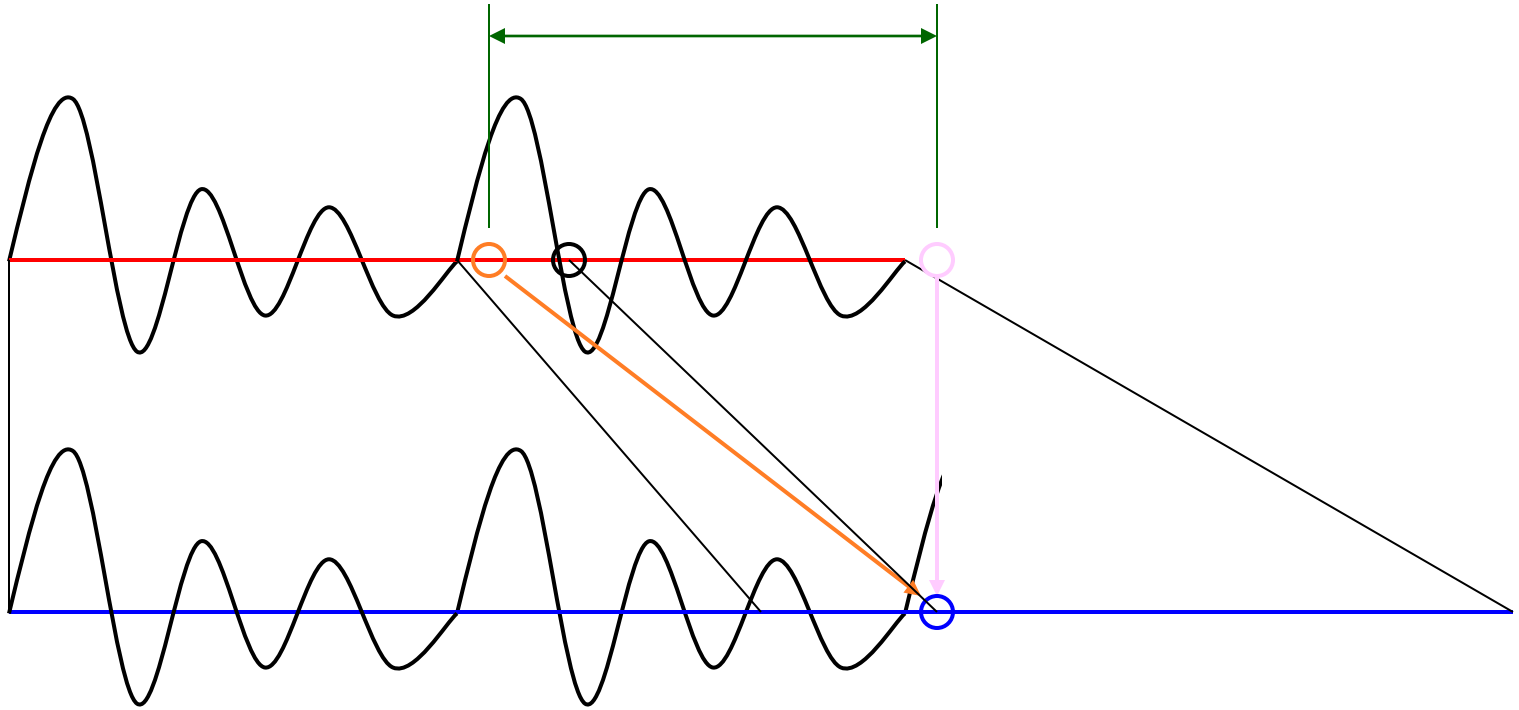


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(11/17)

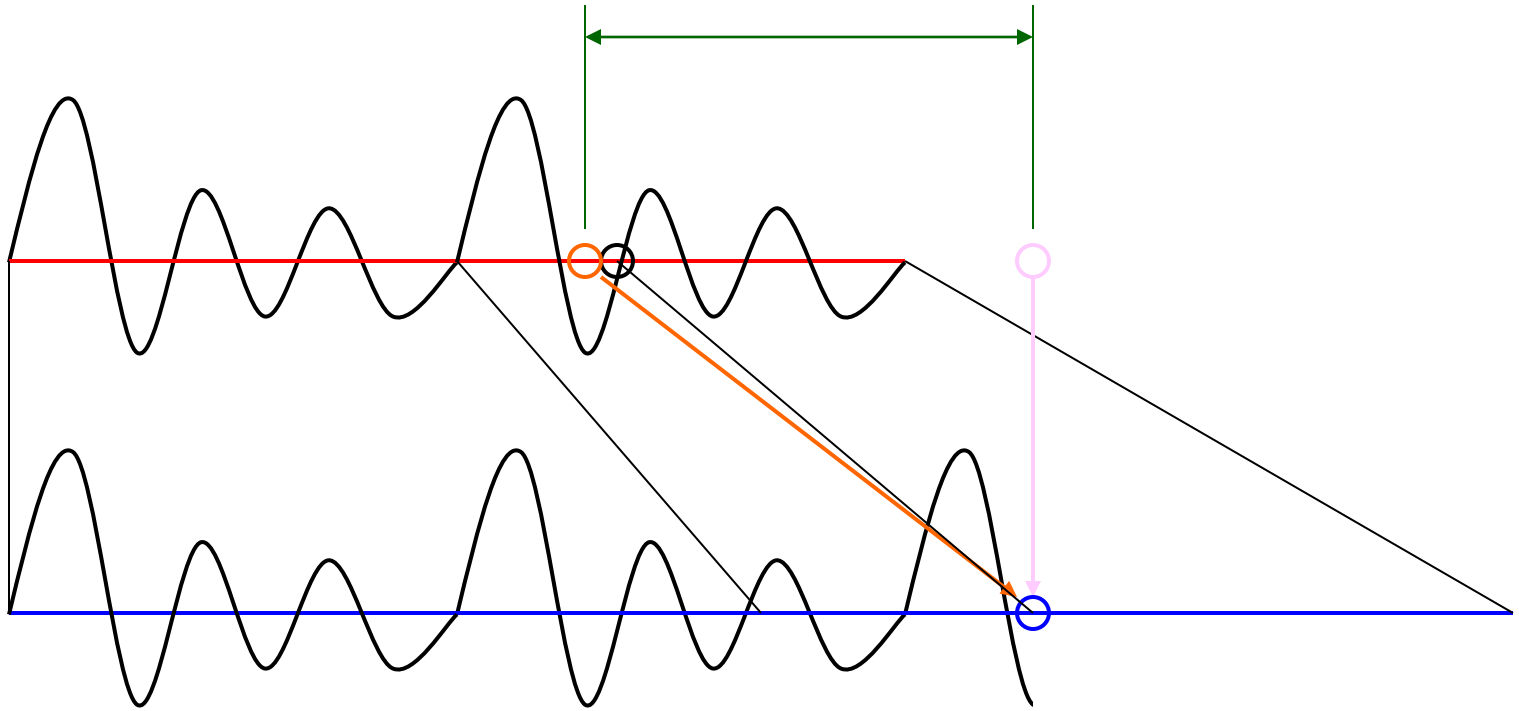


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(12/17)

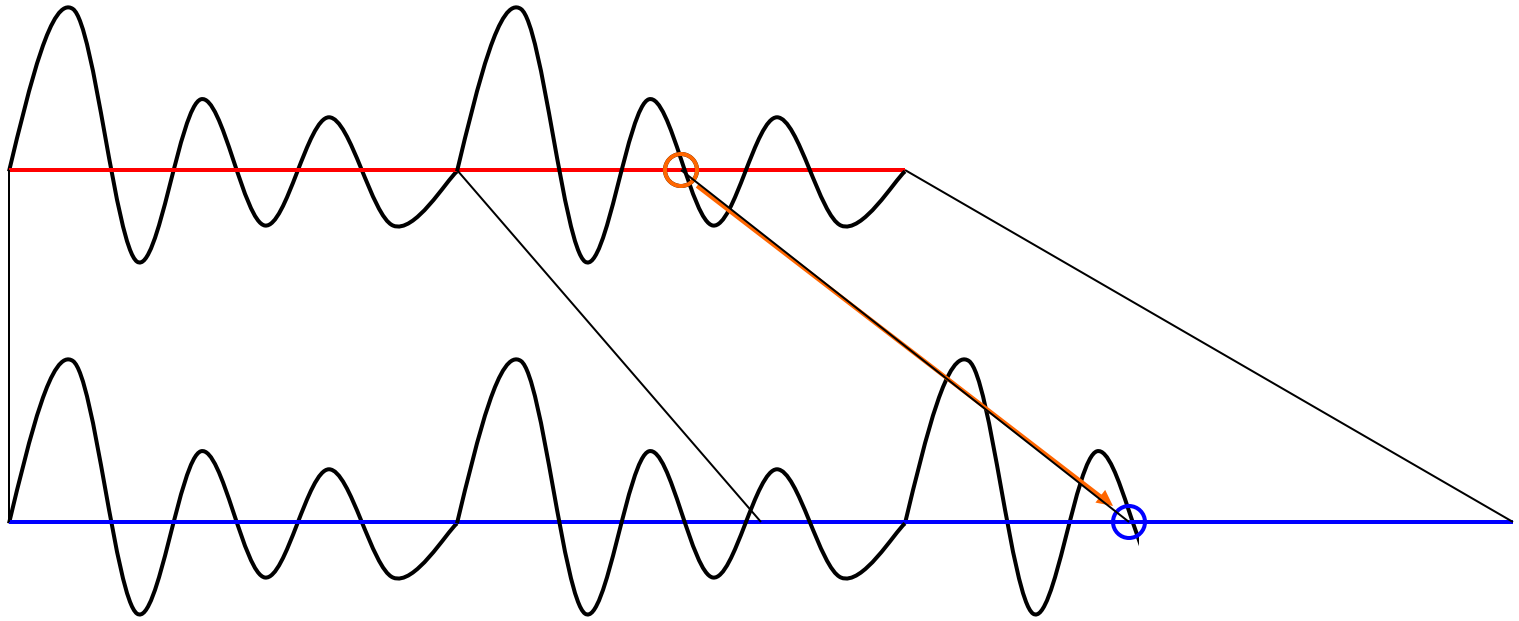


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(13/17)

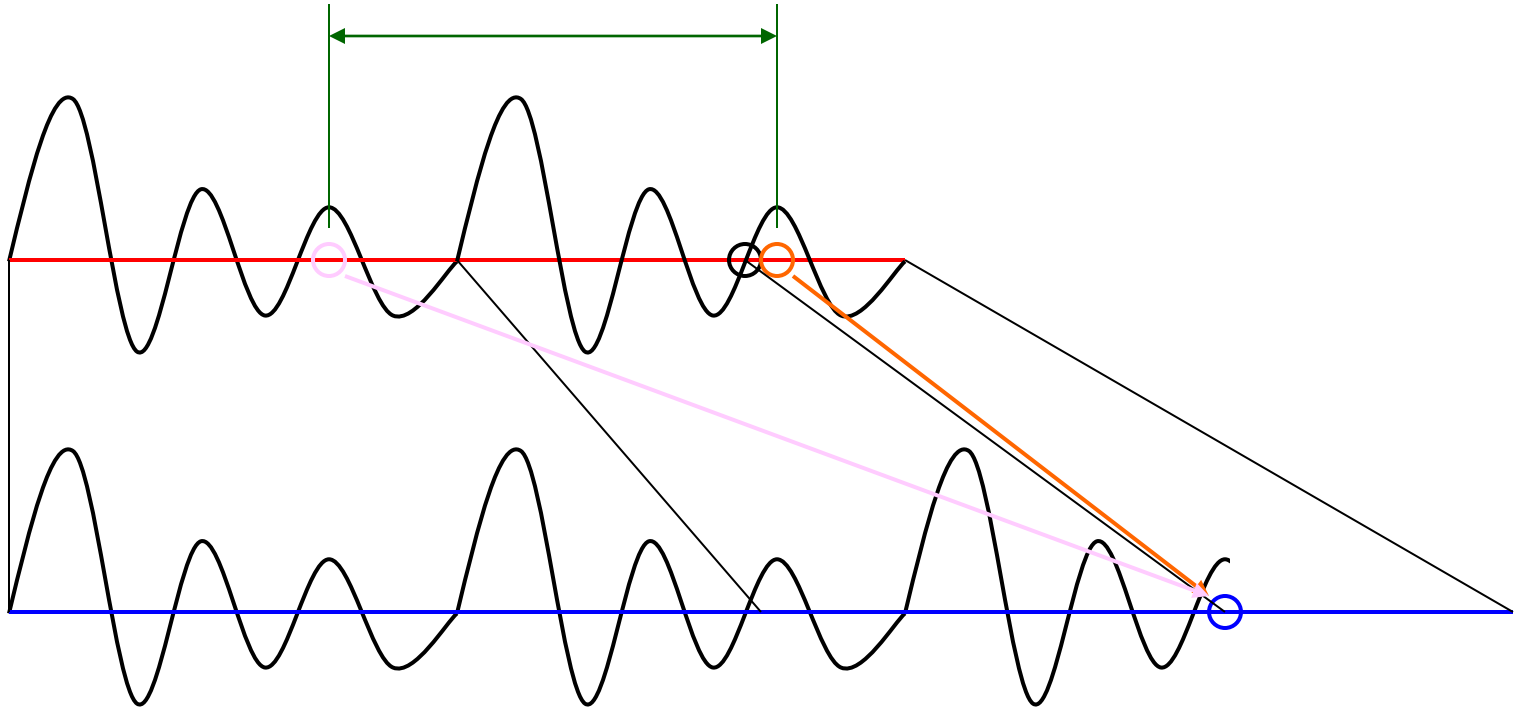


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(14/17)

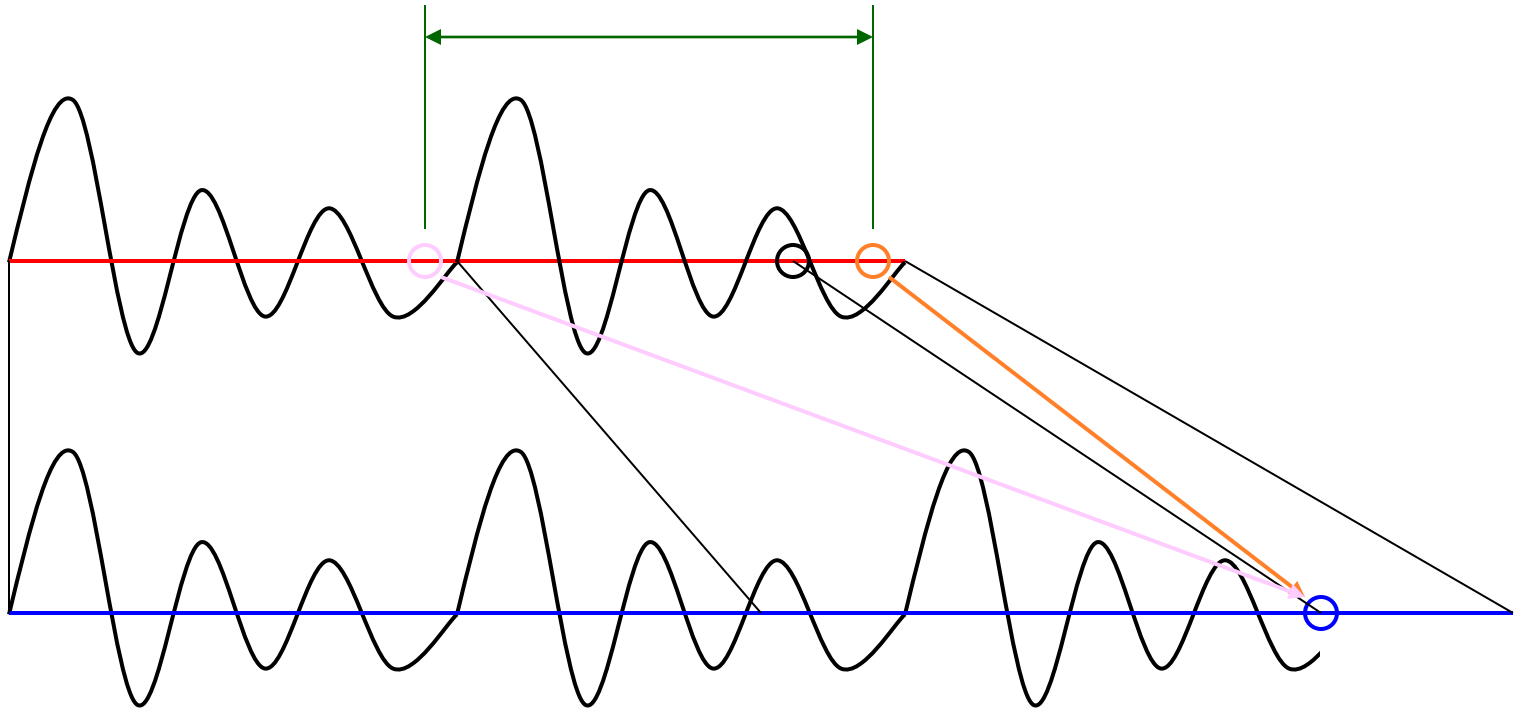


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(15/17)

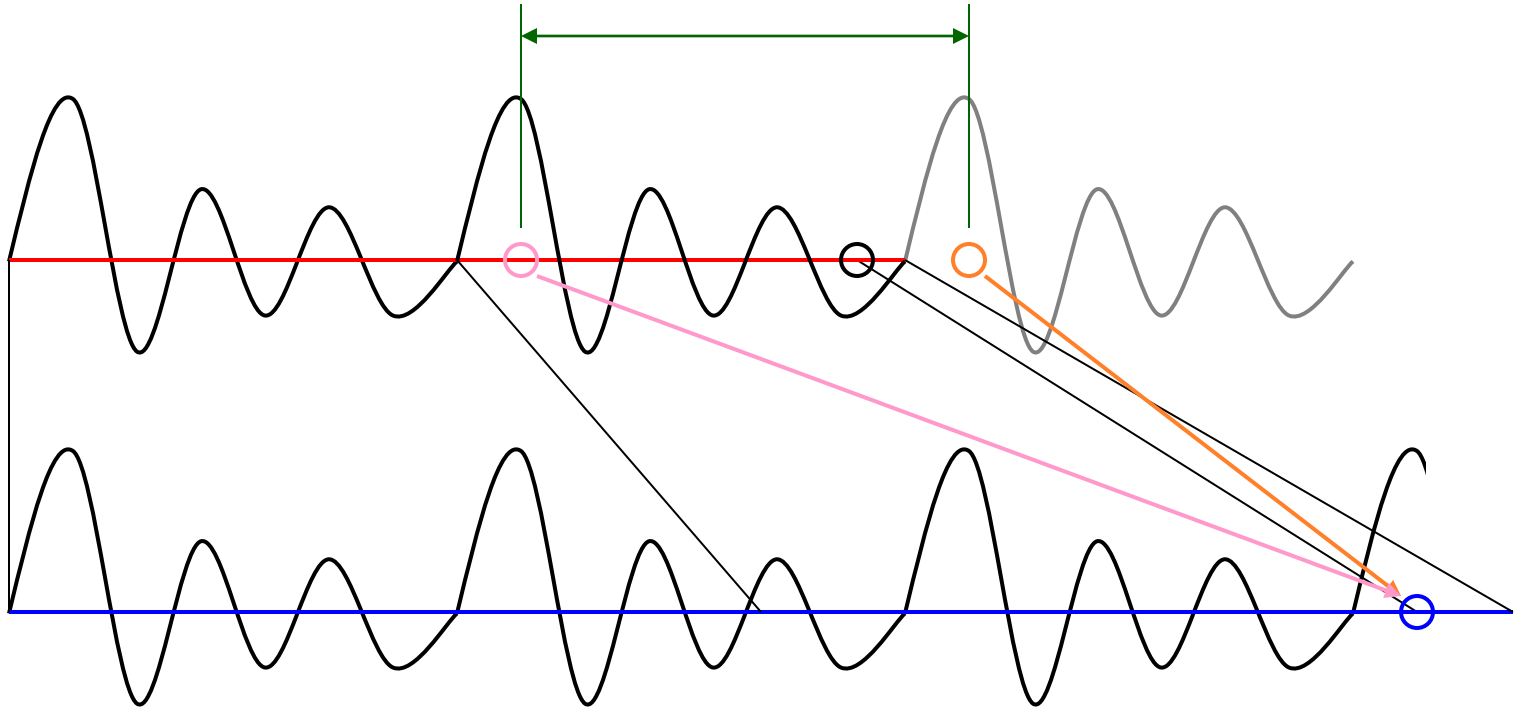


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(16/17)

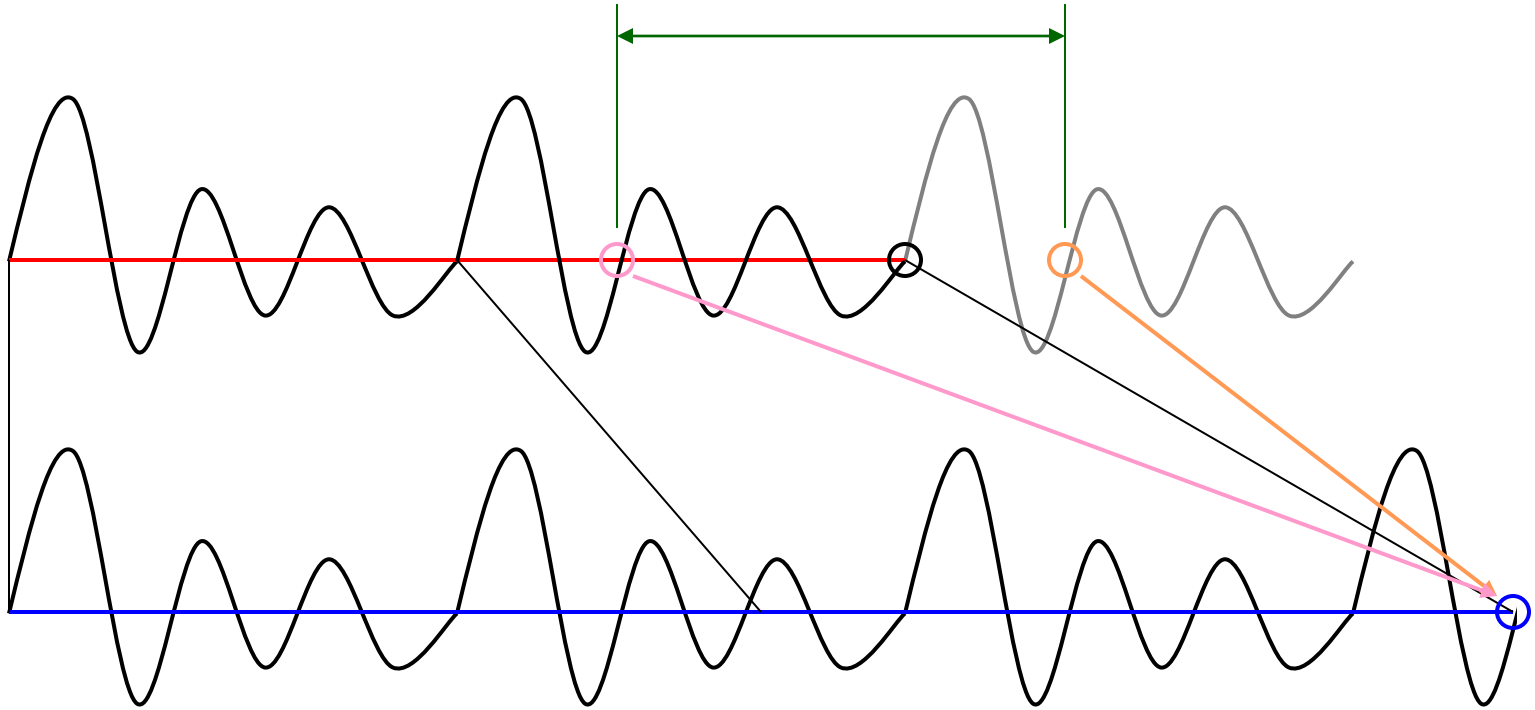


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮の一つの方法(17/17)

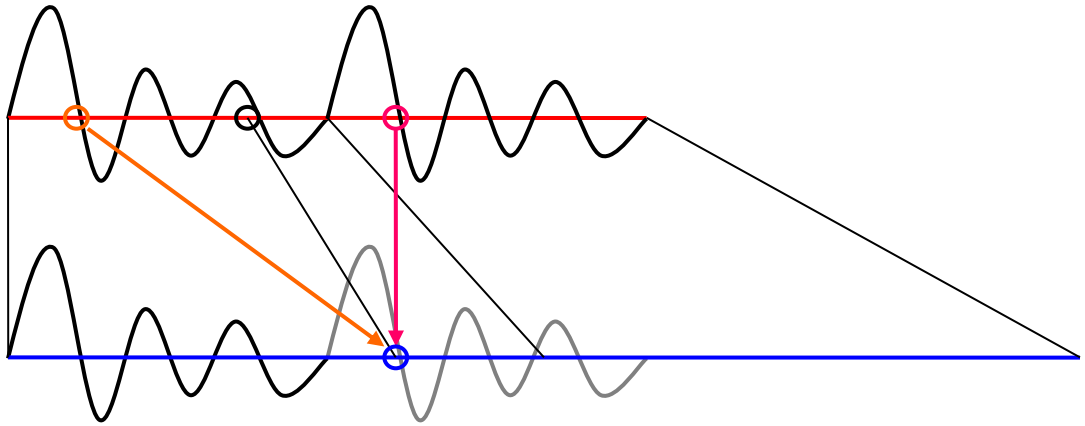


○　単純な伸縮によるコピー元サンプル位置

○ ○　実際にコピーするサンプル位置

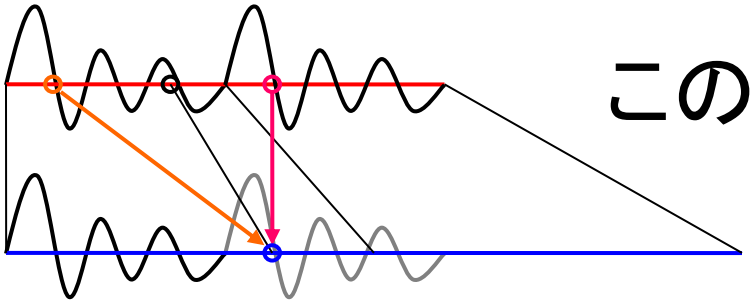
※○と○の間隔は常に周期長・○からの距離が遠い程比率を小さくする

# 音声伸縮と音程変更を同時に行う



先ほどの説明では入力〇〇の移動速度が出力〇側と等速だったが、入力〇〇側を出力〇に対して適切な速度にすることで音程も変更できる。

この方法の利点は長さの伸縮比率が黒丸〇の速度で、音程の変更は赤丸〇〇の速度で決まり、音程と伸縮のパラメータが独立しているところにある



# この方法を実装するのに必要だった技術

1．元音声データの各点での正確な周期を取得

各点での周期の正確さがこの方法のキモになります。

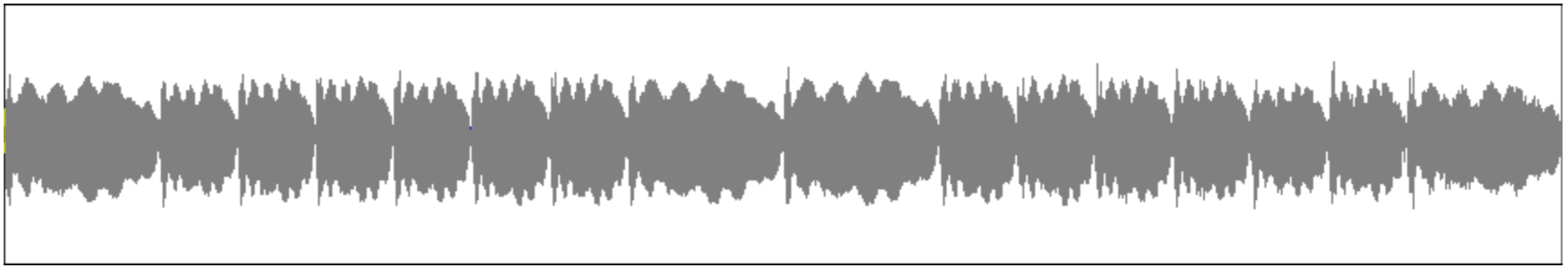
2．『0．8サンプル』のような半端な位置での値を取得

これは適当な補完関数を使って任意の位置の値を計算した。

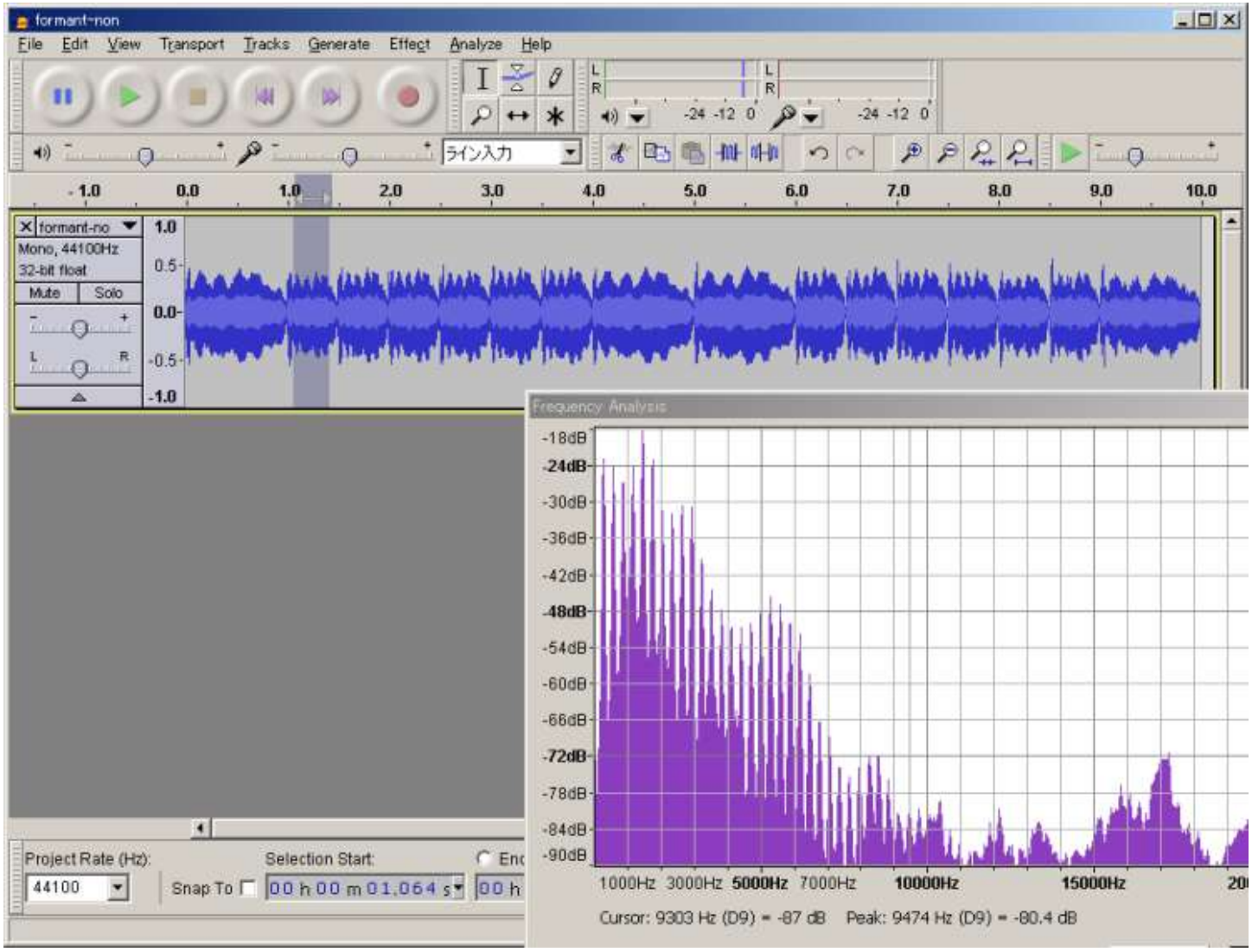
初期版に用いたのは三次スプライン曲線。

# 出力サンプル

元音声

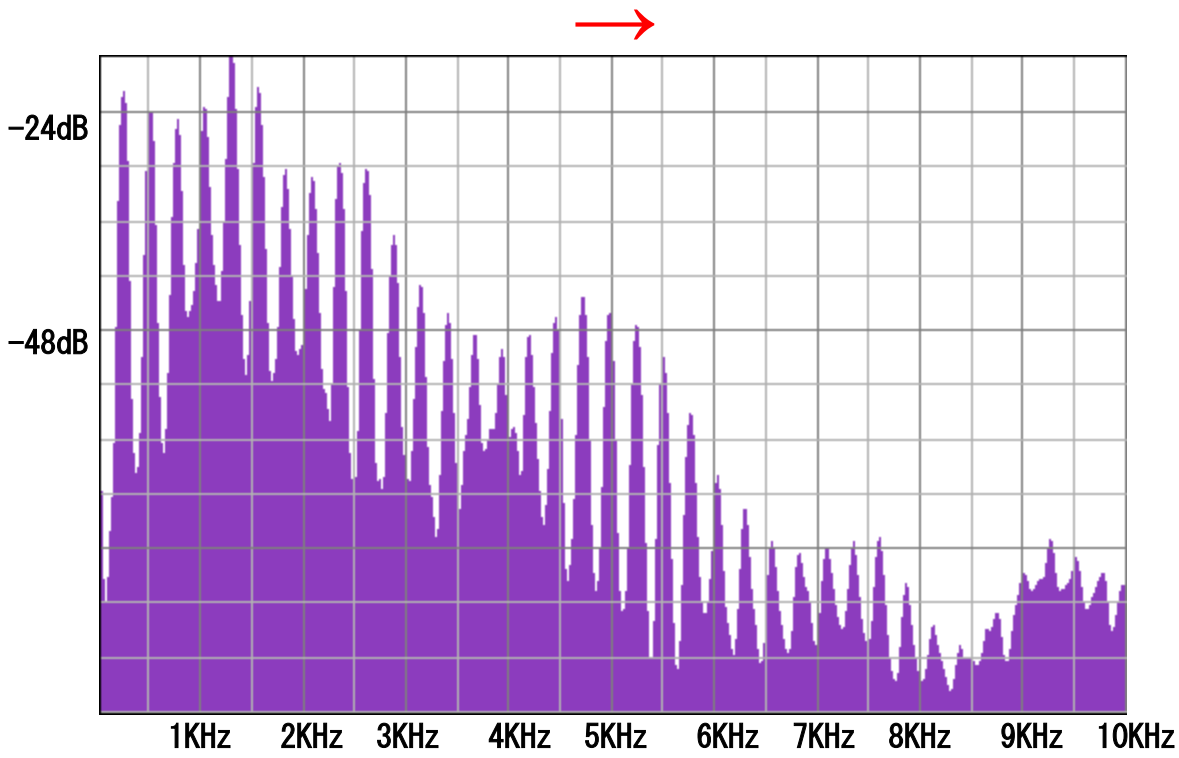


# スペクトラム解析を表示させる



# これだけでは足りない理由01

スペクトラムの山が移動

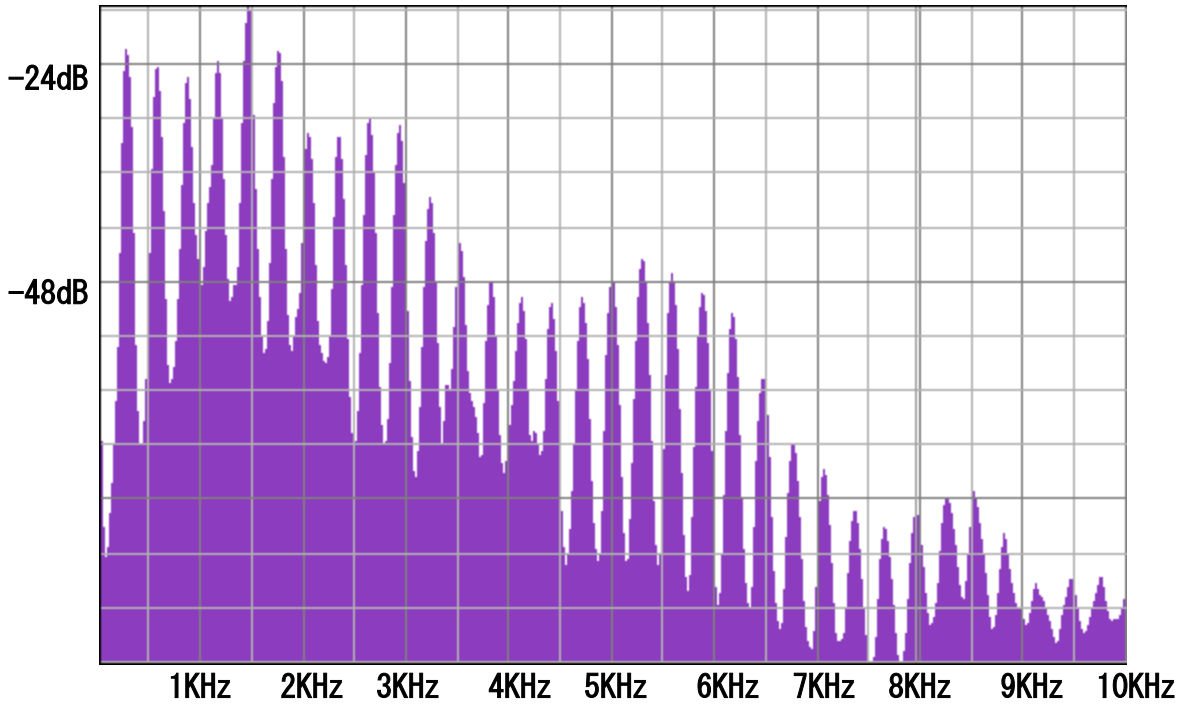



「あ」G#3(206.4Hz)→C4(261.6Hz)

# これだけでは足りない理由02

スペクトラムの山が移動

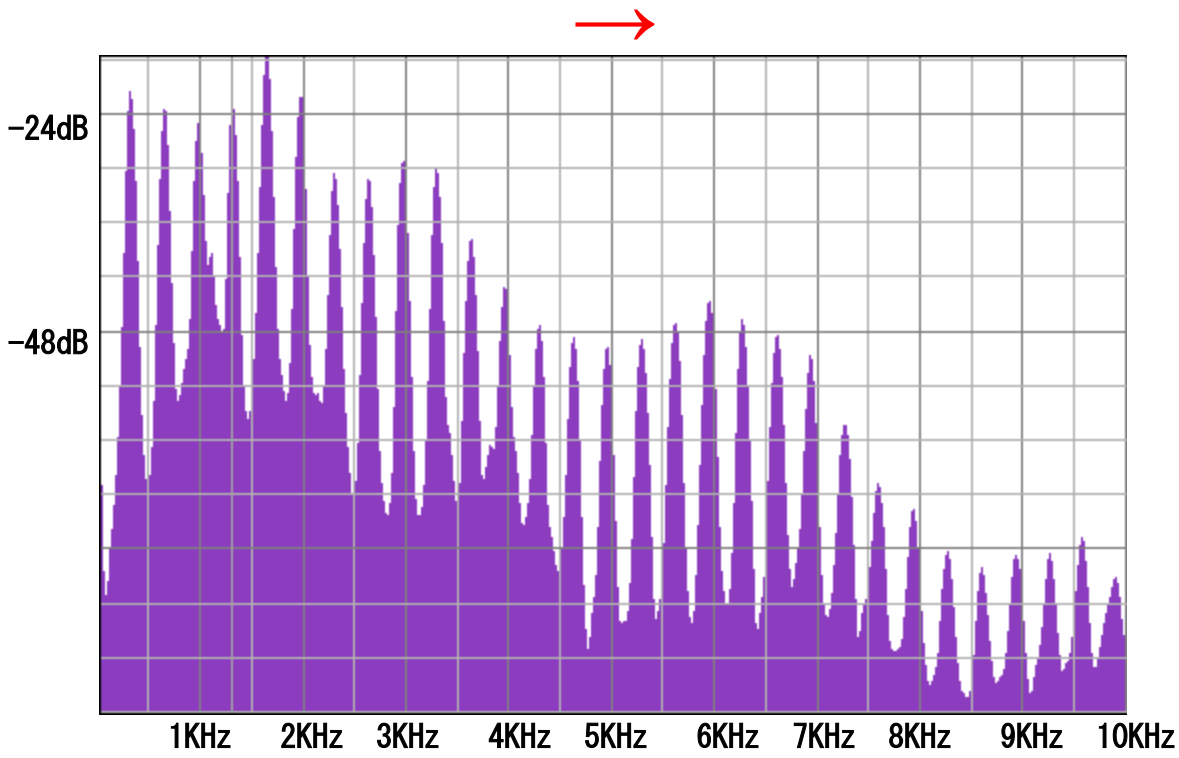
→




「あ」G#3(206.4Hz)→D4(293.7Hz)

# これだけでは足りない理由03

スペクトラムの山が移動

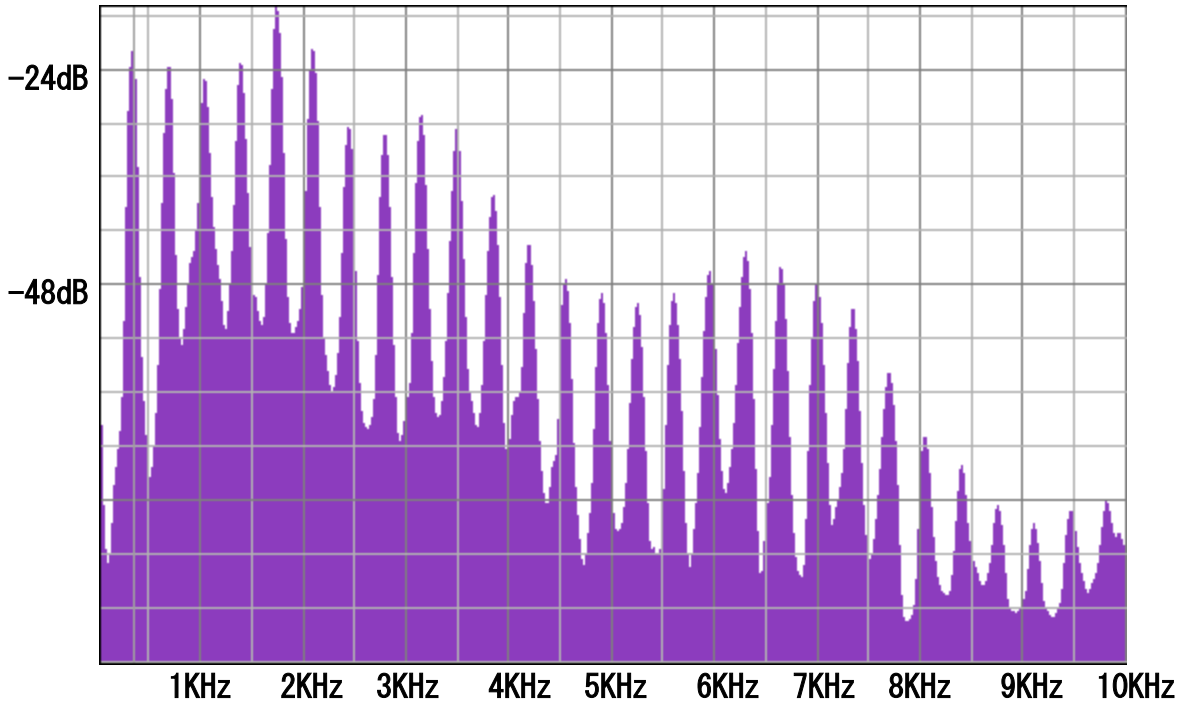



「あ」G#3(206.4Hz)→E4(329.6Hz)

# これだけでは足りない理由04

スペクトラムの山が移動

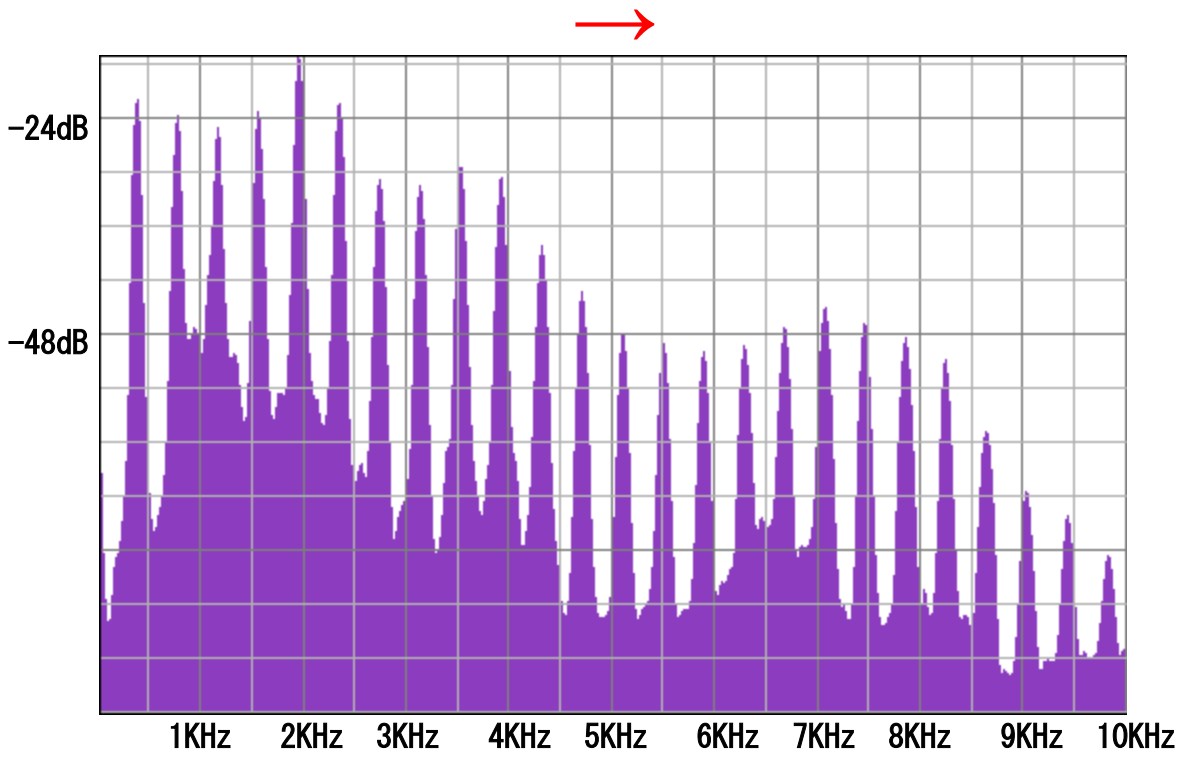
→




「あ」G#3(206.4Hz)→F4(349.2Hz)

# これだけでは足りない理由05

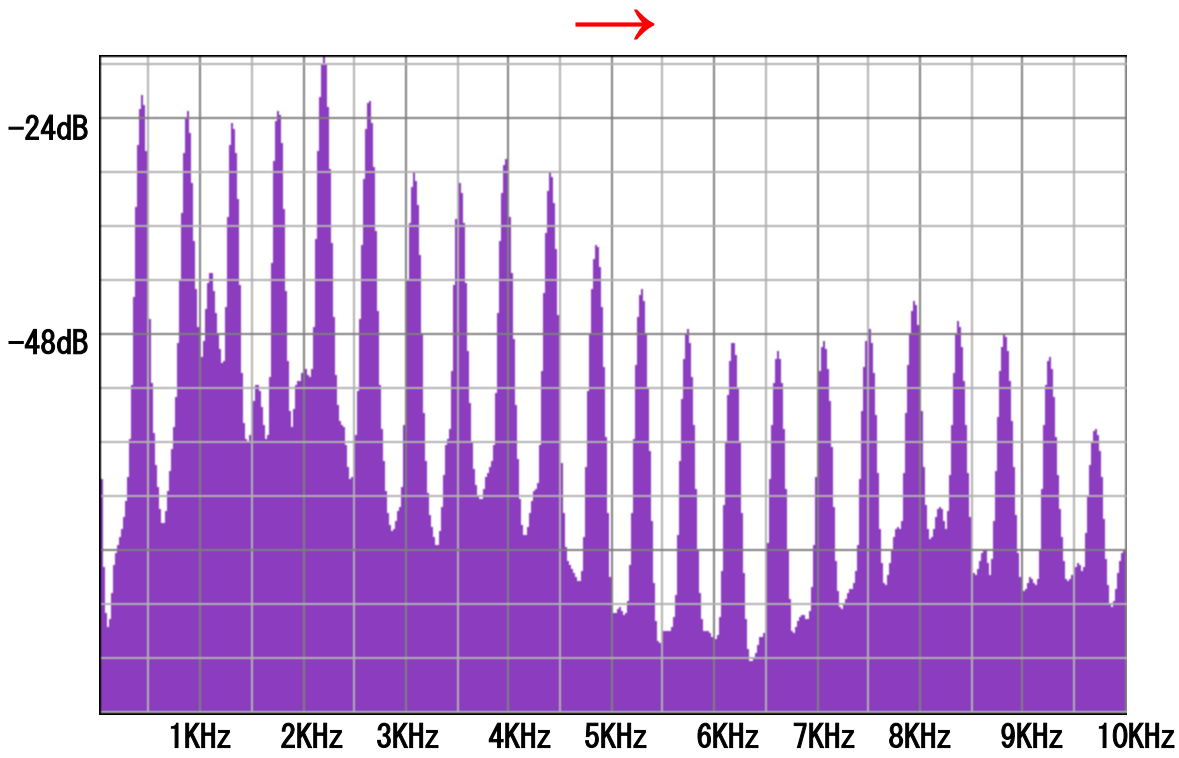
スペクトラムの山が移動




「あ」G#3(206.4Hz)→G4(392.0Hz)

# これだけでは足りない理由06

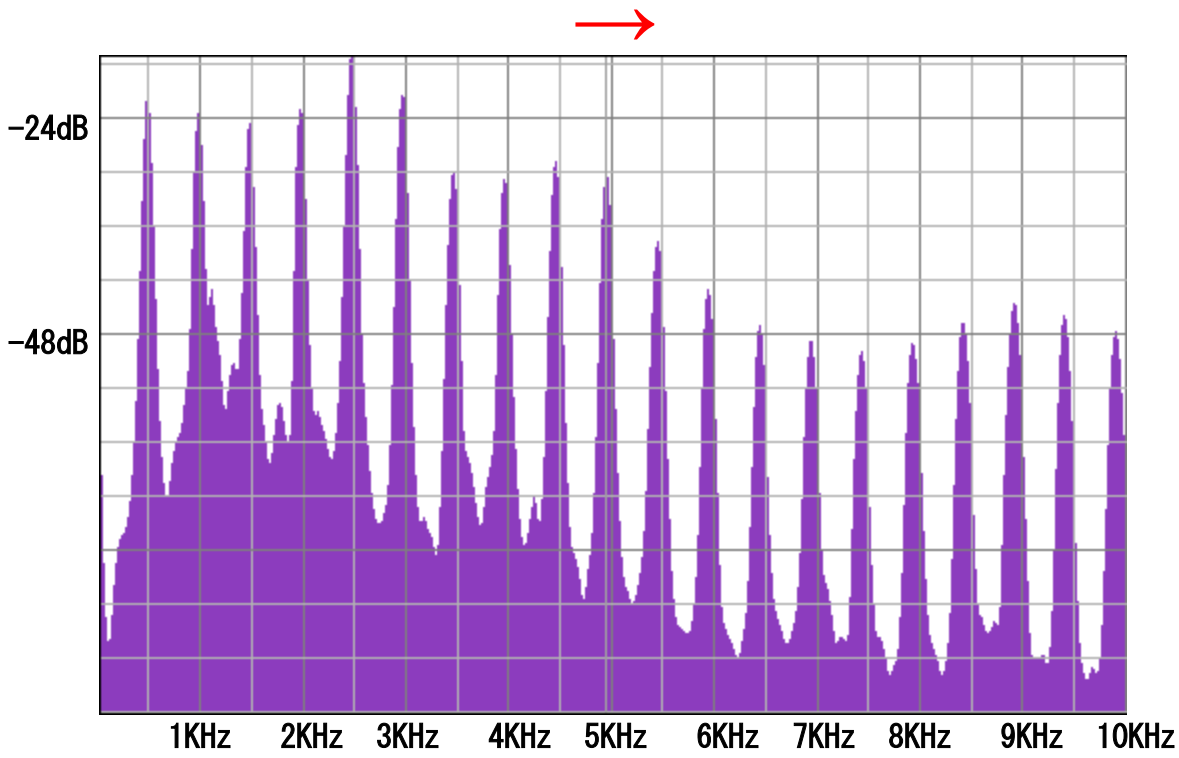
スペクトラムの山が移動




「あ」G#3(206.4Hz)→A4(440.0Hz)

# これだけでは足りない理由07

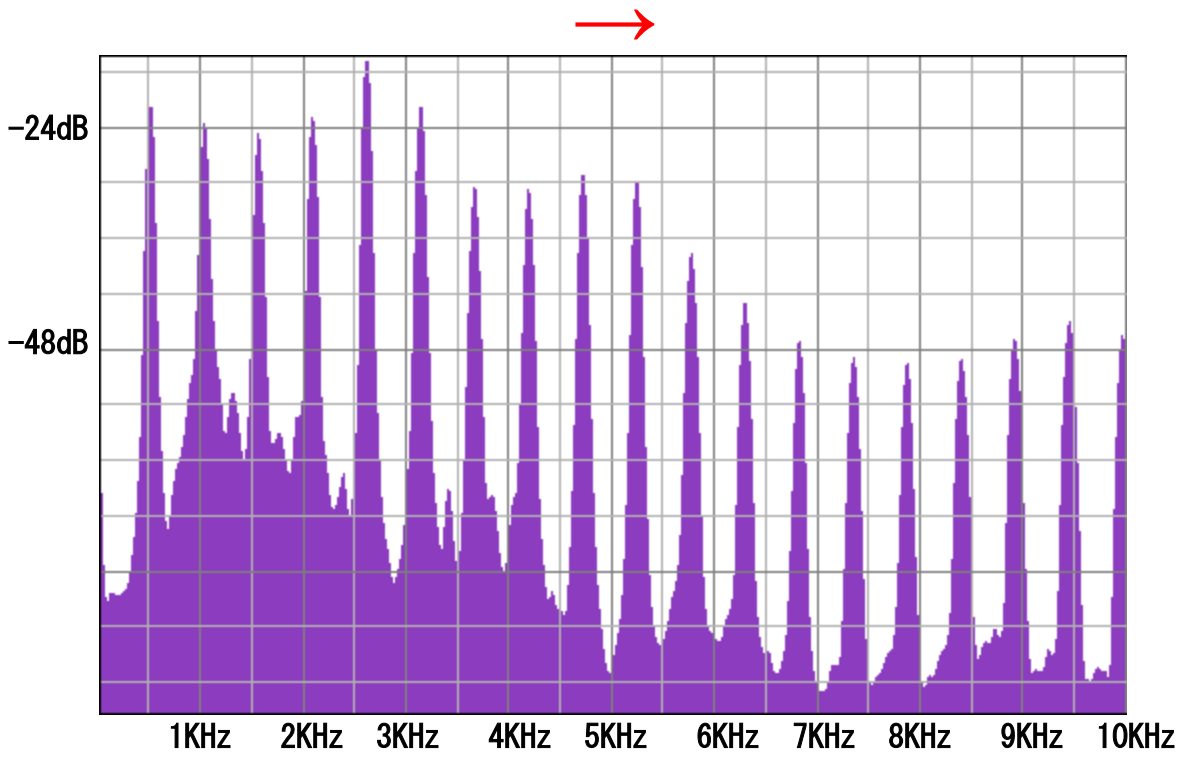
スペクトラムの山が移動




「あ」G#3(206.4Hz)→B4(493.9Hz)

# これだけでは足りない理由08

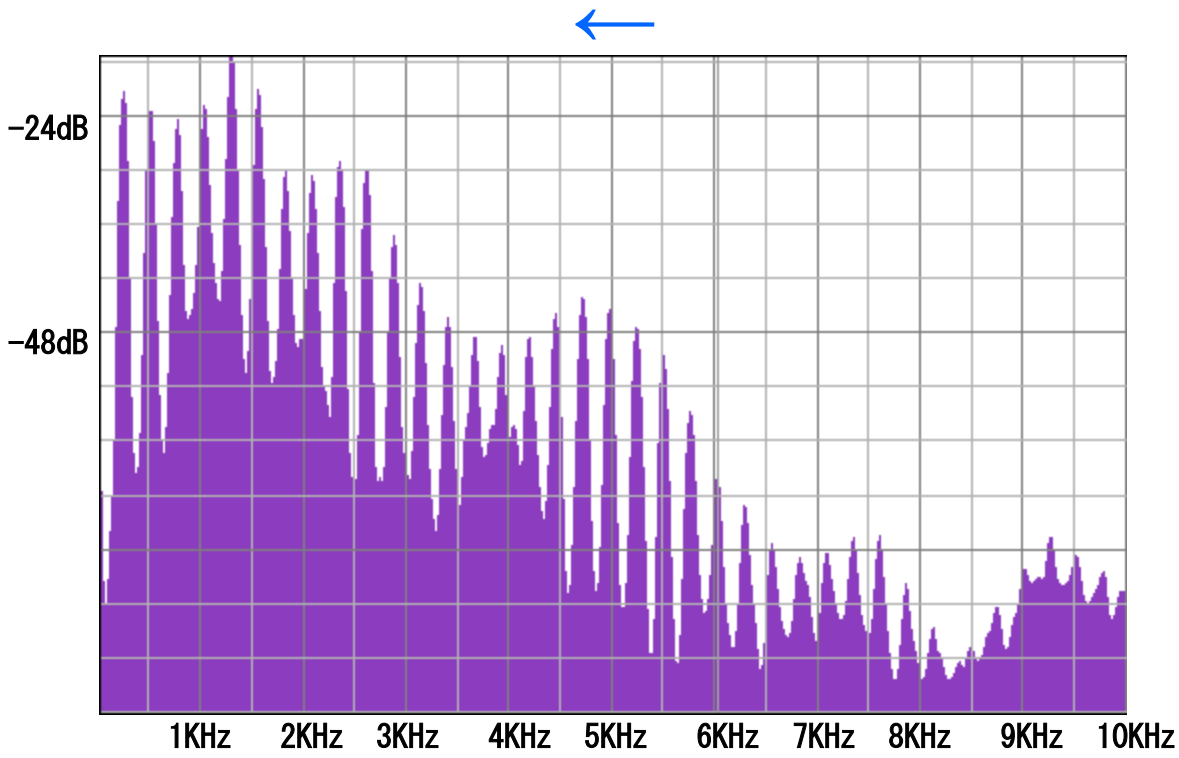
スペクトラムの山が移動




「あ」G#3(206.4Hz)→C5(523.3Hz)

# これだけでは足りない理由09

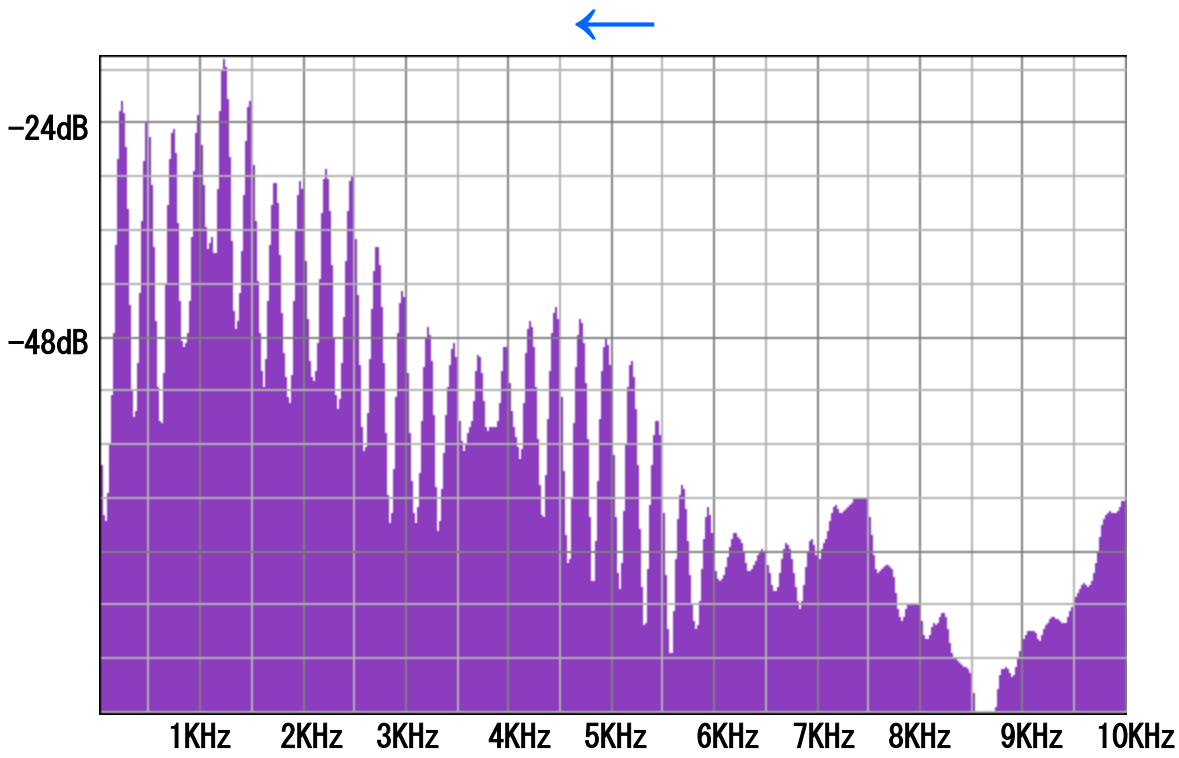
## スペクトラムの山が移動




「あ」G#3(206.4Hz)→C4(261.6Hz)

# これだけでは足りない理由10

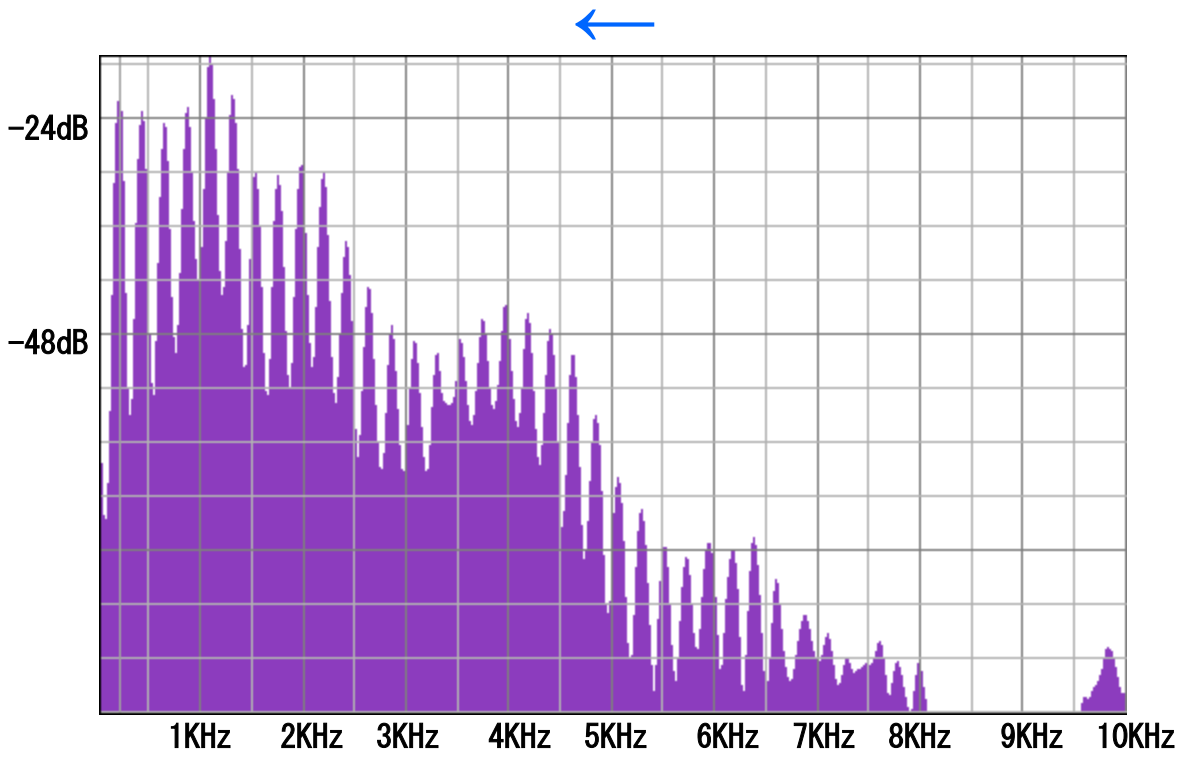
スペクトラムの山が移動




「あ」G#3(206.4Hz)→B3(246.9Hz)

# これだけでは足りない理由11

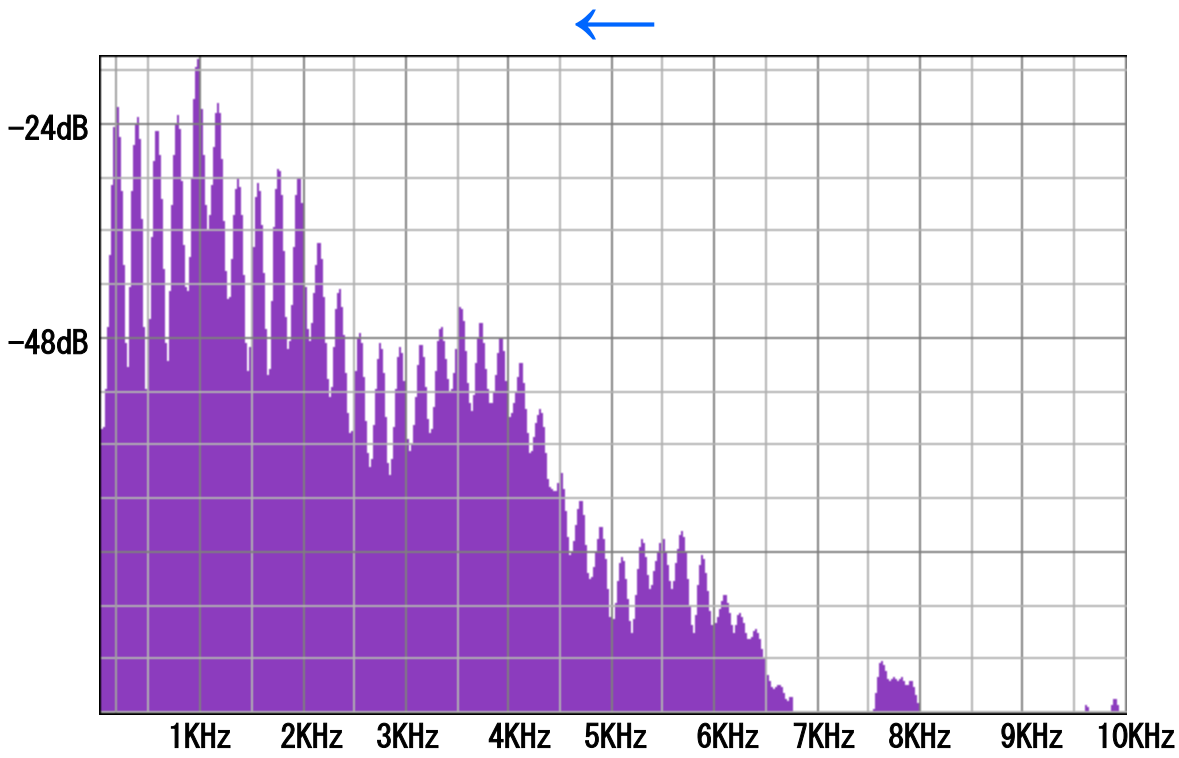
スペクトラムの山が移動




「あ」G#3(206.4Hz)→A3(220.0Hz)

# これだけでは足りない理由12

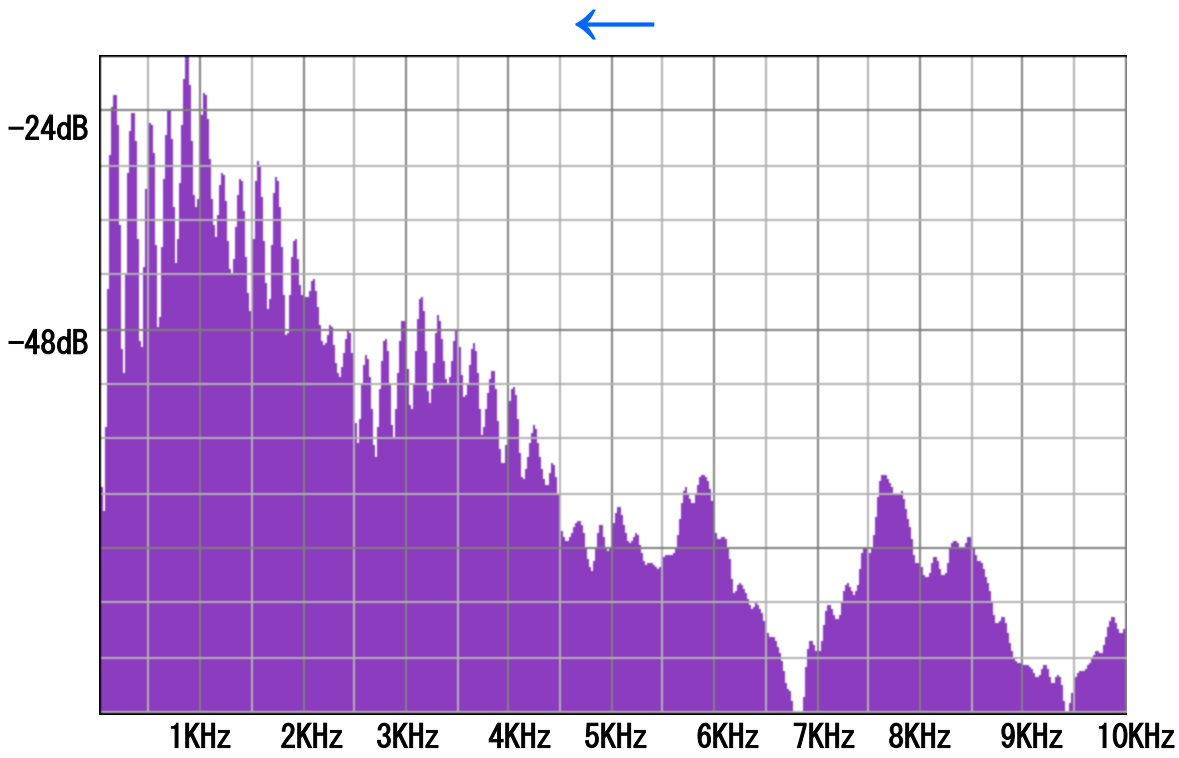
スペクトラムの山が移動




「あ」G#3(206.4Hz)→G3(261.6Hz)

# これだけでは足りない理由13

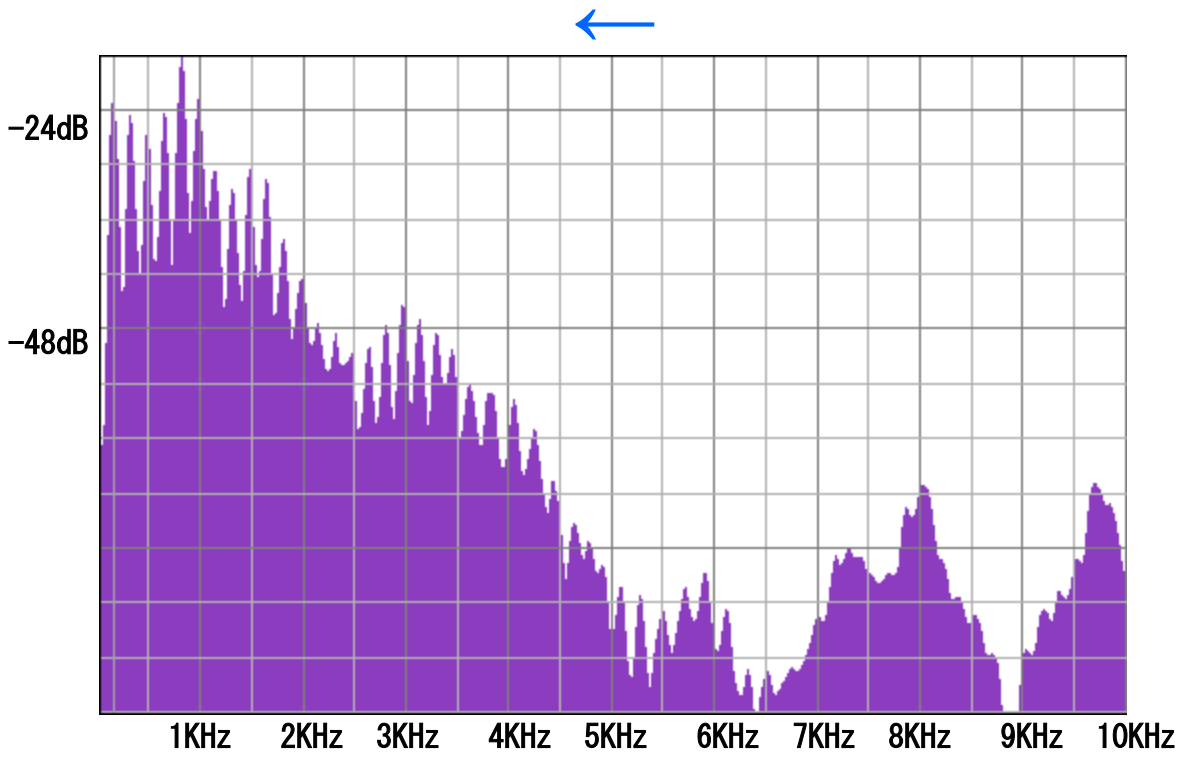
スペクトラムの山が移動




「あ」G#3(206.4Hz)→F3(174.6Hz)

# これだけでは足りない理由14

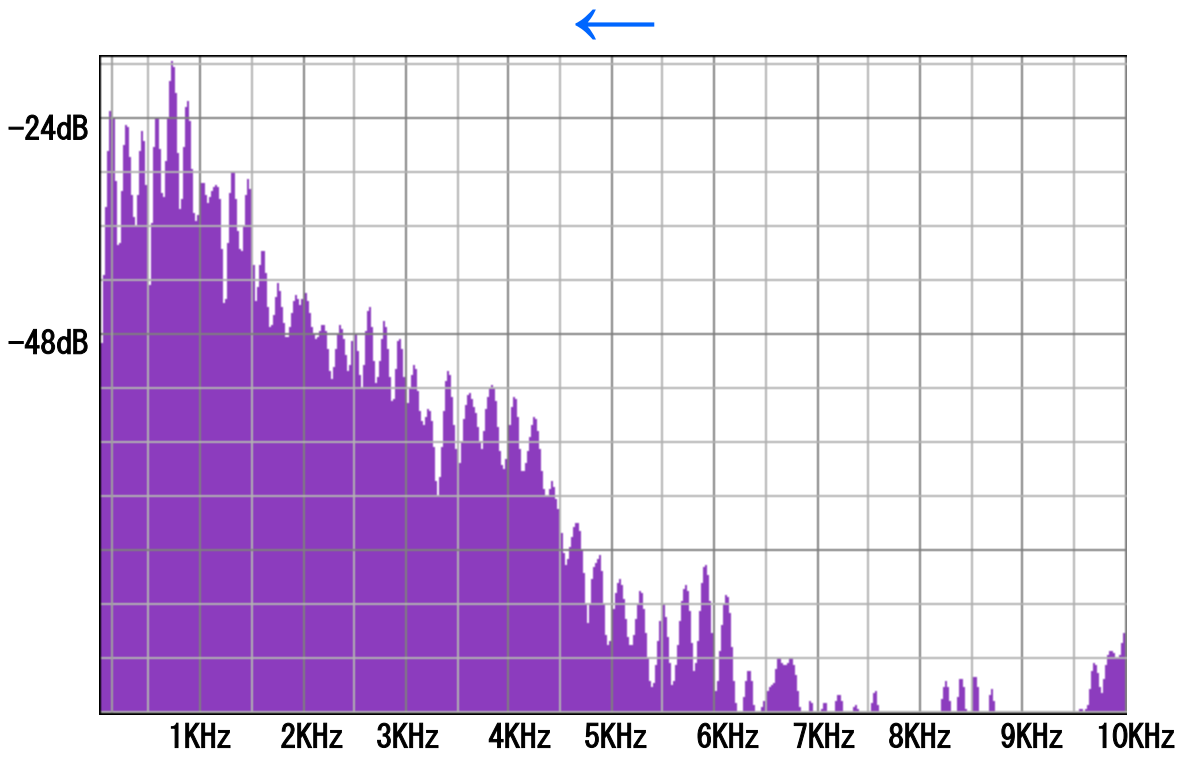
スペクトラムの山が移動




「あ」G#3(206.4Hz)→E3(164.8Hz)

# これだけでは足りない理由15

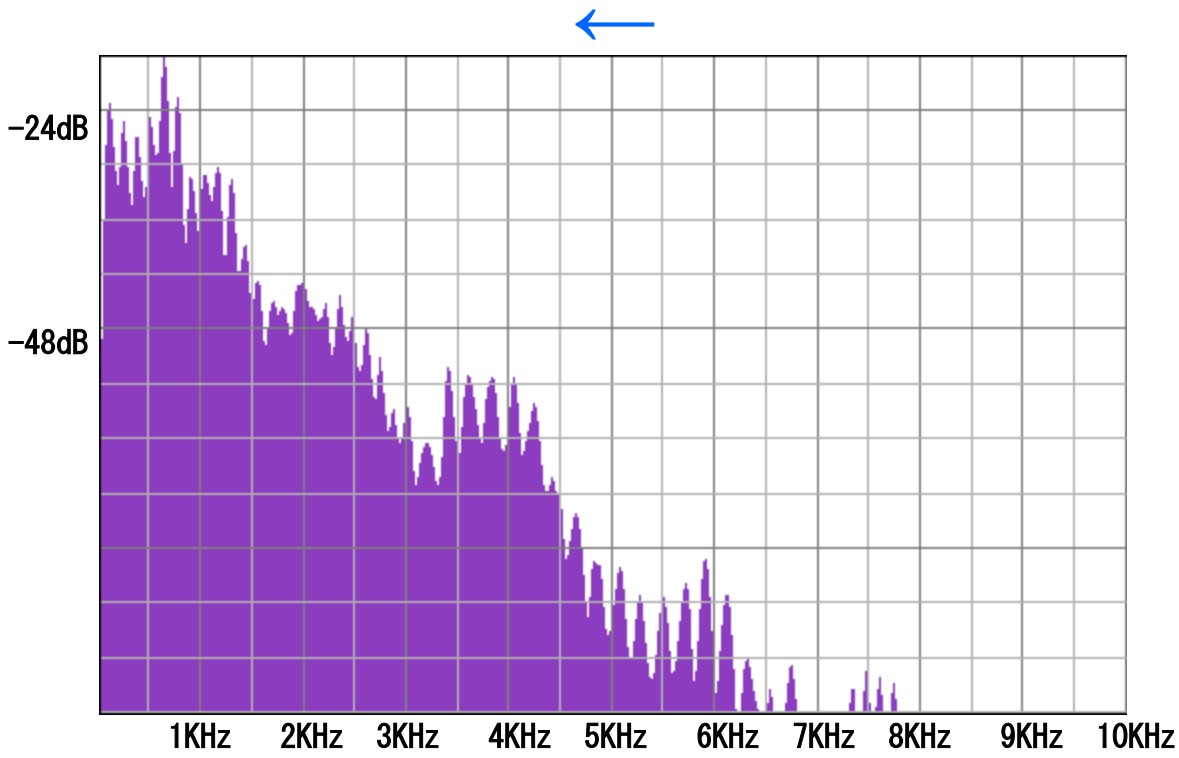
スペクトラムの山が移動




「あ」G#3(206.4Hz)→D3(146.8Hz)

# これだけでは足りない理由16

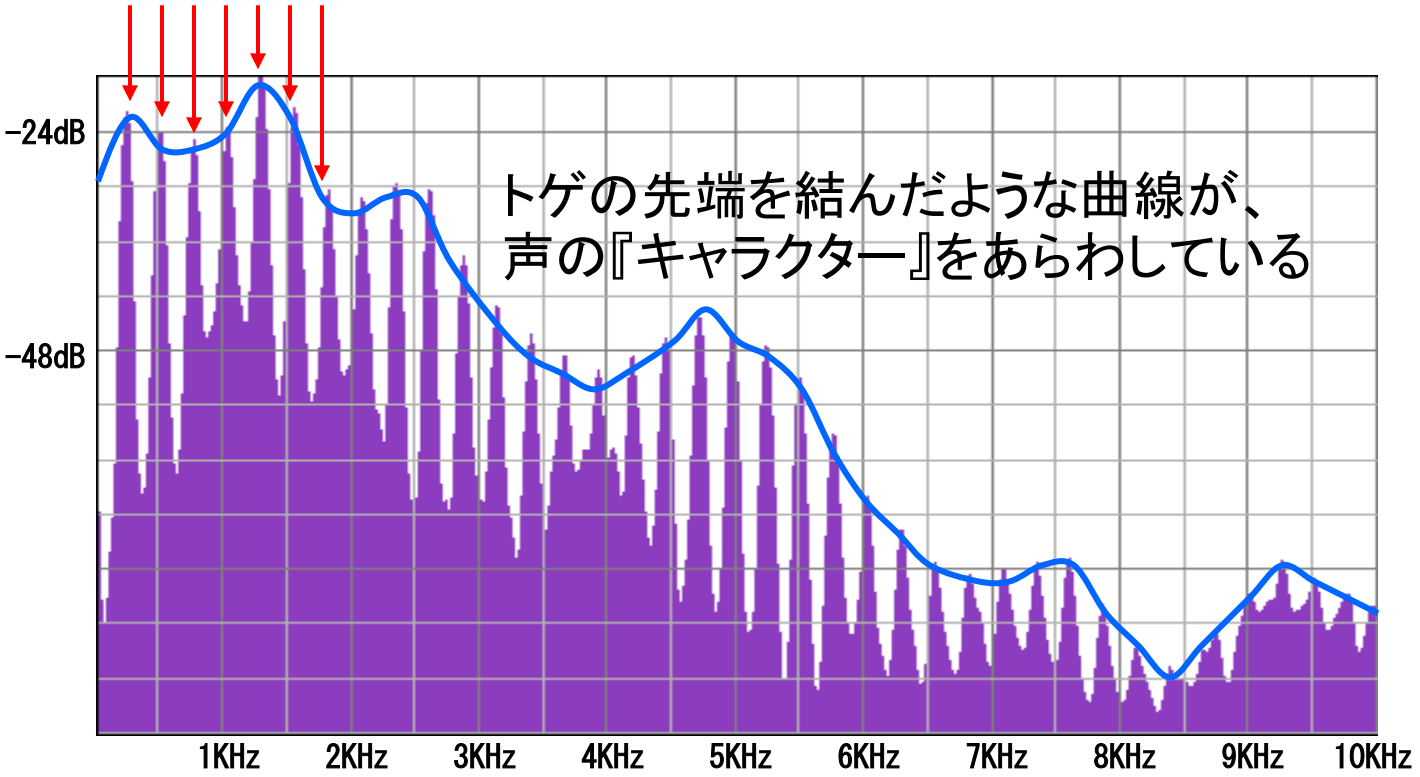
スペクトラムの山が移動




「あ」G#3(206.4Hz)→C3(130.8Hz)

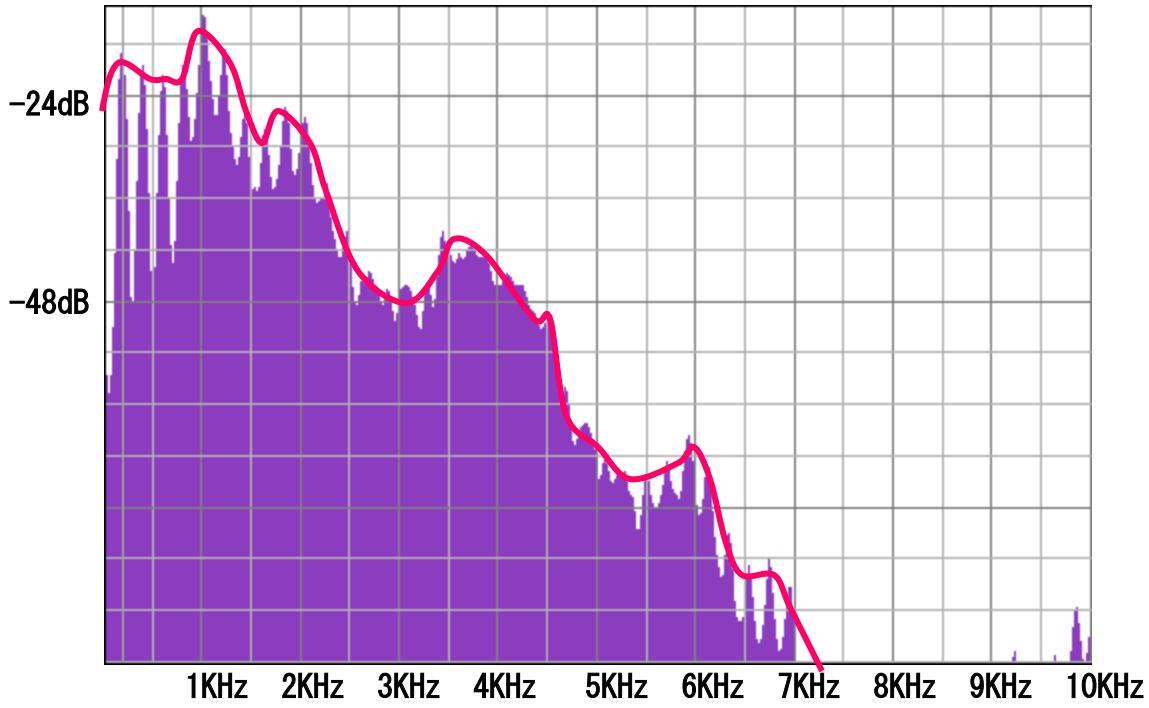
# スペクトラムの解読

この等間隔のトゲトゲが音の高さを表す。間隔が広いほど高い音。



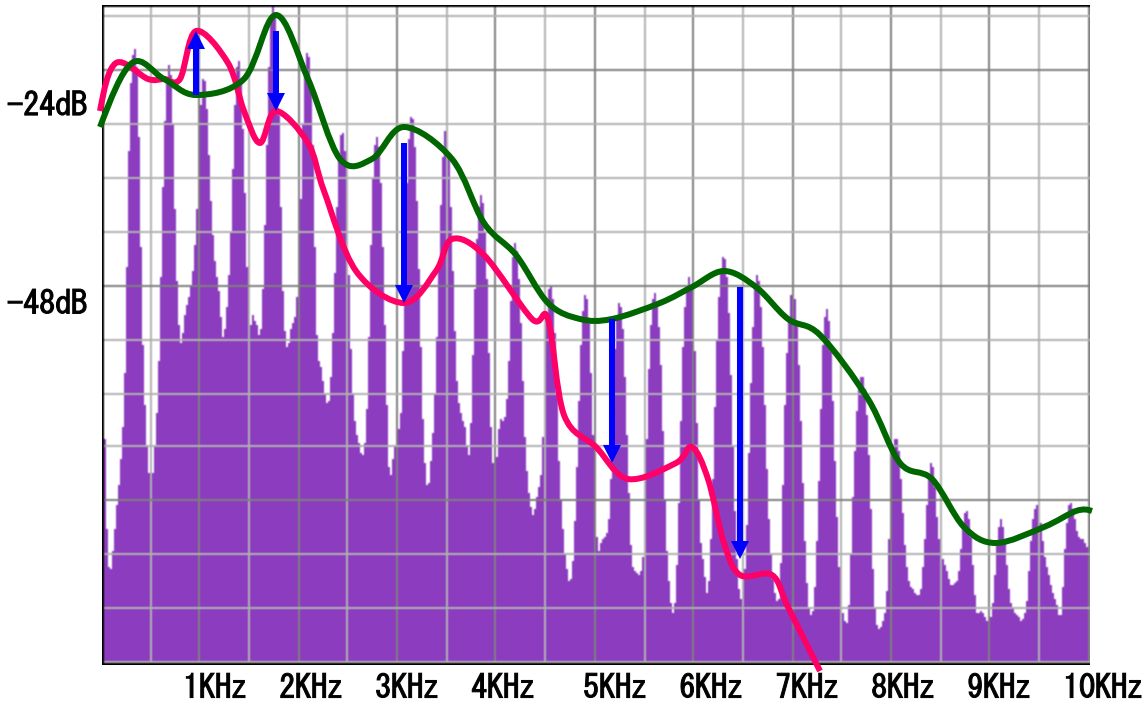

※『調音成分』とか『声道フィルタ特性』等の用語はUTAU開発後、かなり後になって知りました。

# ゆえに、元音声のこの曲線を計算して




元音声「あ.wav」　G#3(206.4Hz)

# 出力音声のこの曲線を元音声の曲線に合わせれば良い

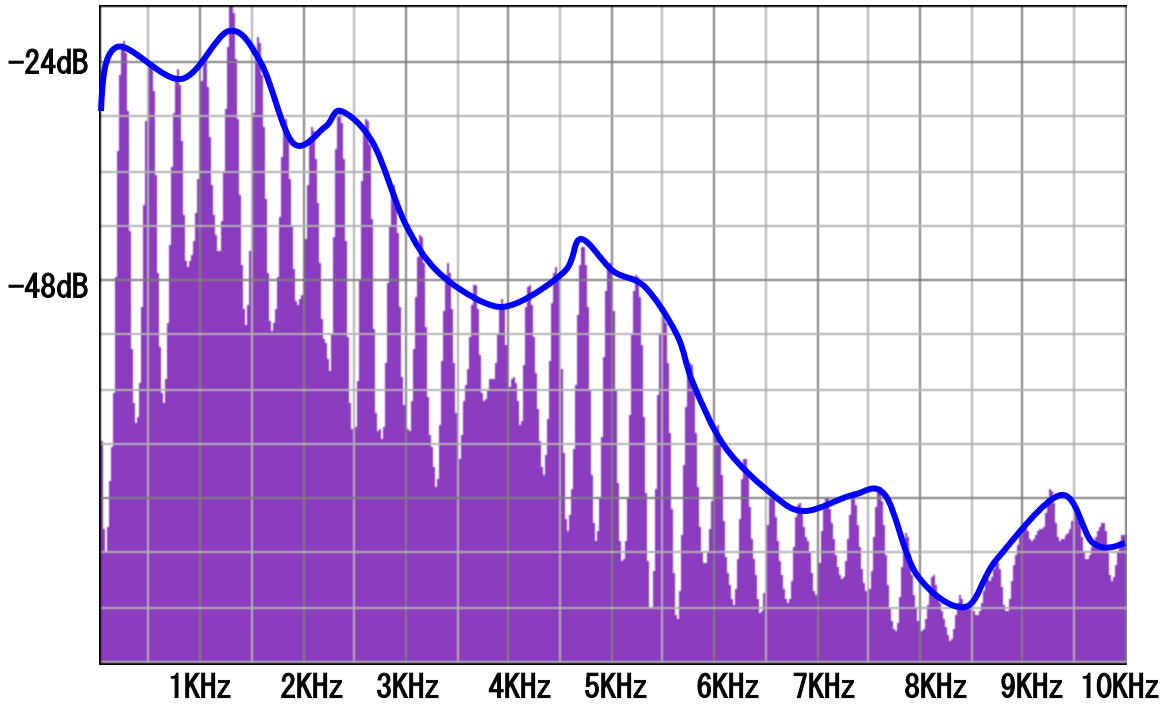



「あ.wav」元音声　G#3(206.4Hz)

# 問題は二つ

- あの曲線を求めるには？
- 求めた曲線を反映させるには？

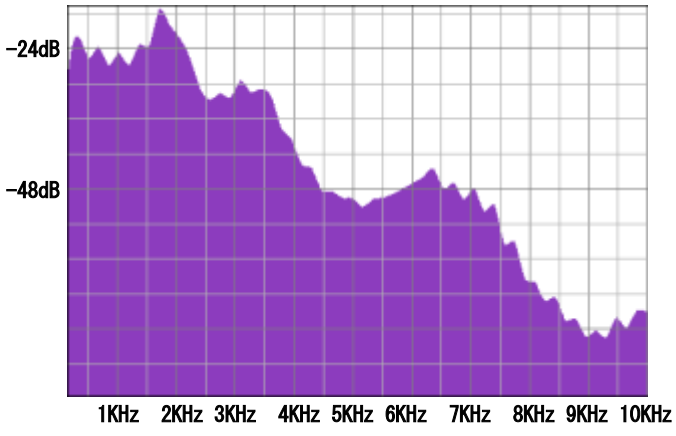
# 1．この曲線の計算法は？



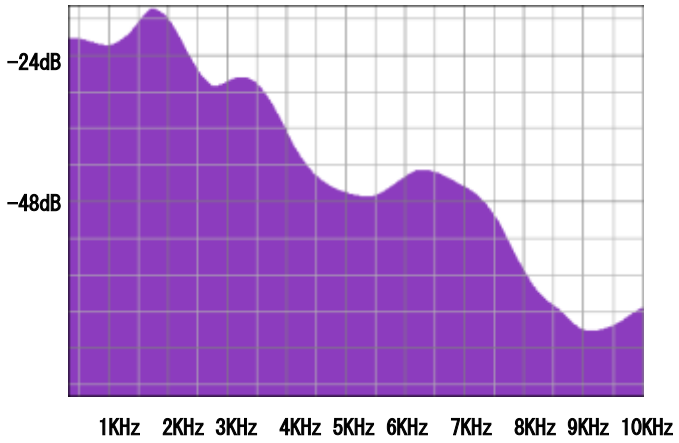
# 『あの曲線』を求める方法1

## FFT（高速フーリエ変換）のポイント数を下げる

N=256

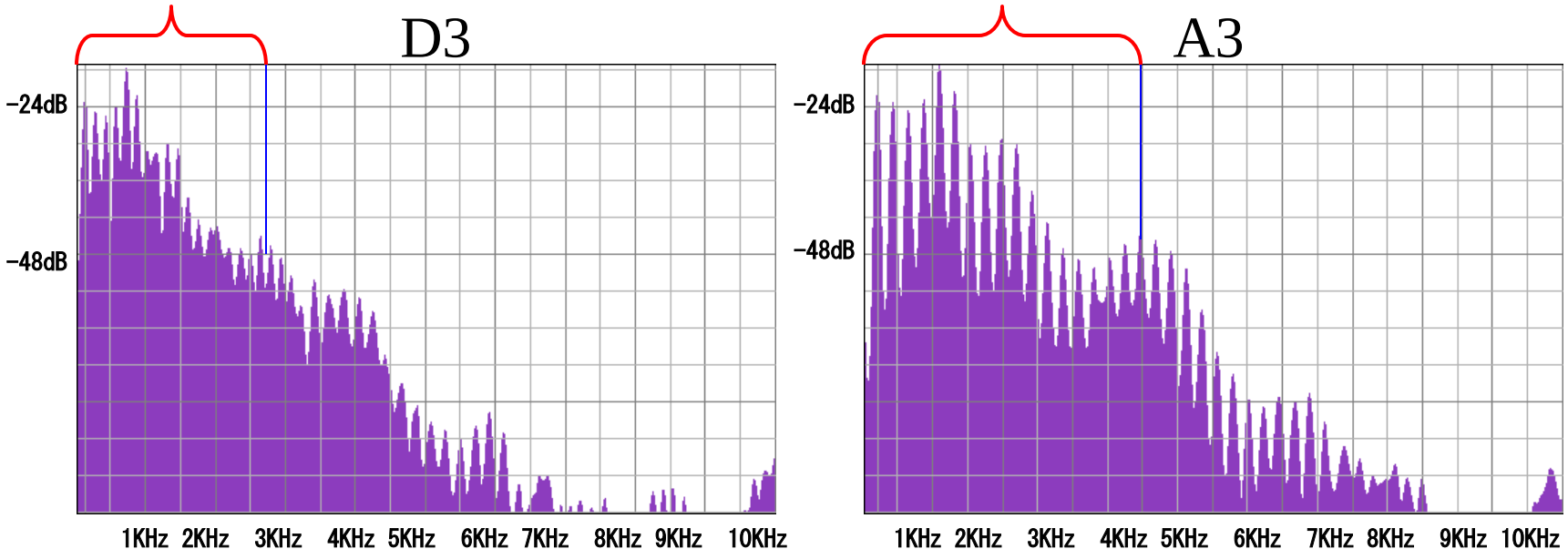



N=128



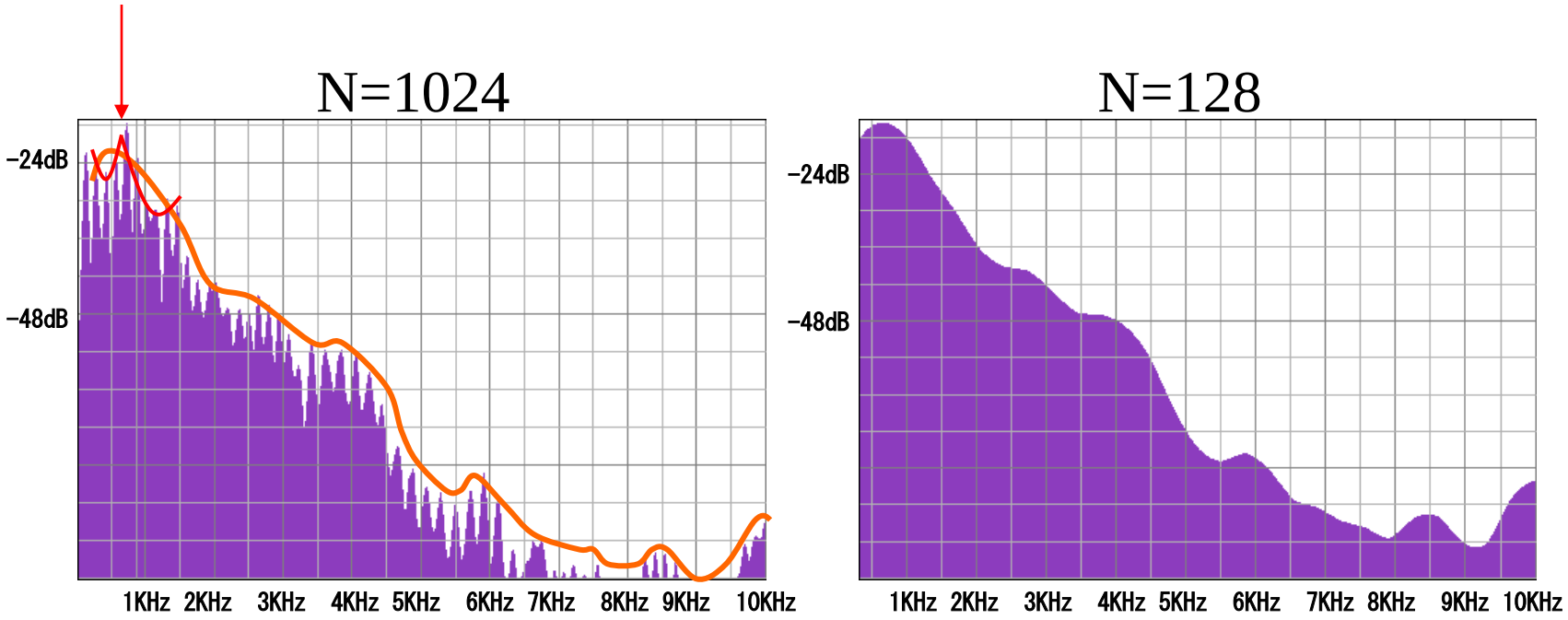
# 方法1の問題点

出力側の音程が低いほど、スペクトラムの構造が低域に詰め込まれる



# 方法1の問題点

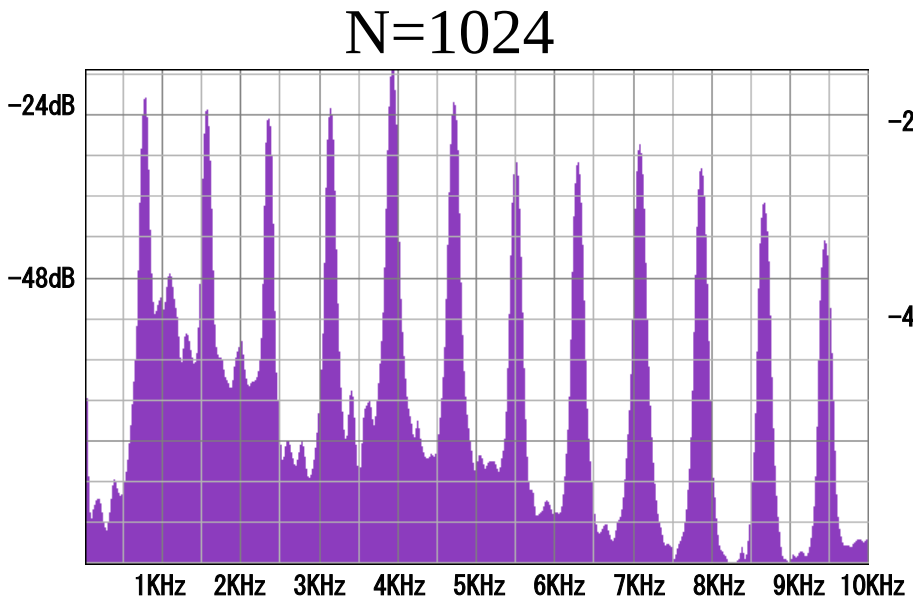
N=128では低域に詰め込まれた構造がつぶれてしまうため、
この曲線を基準にすると、ここの構造が残ってしまう

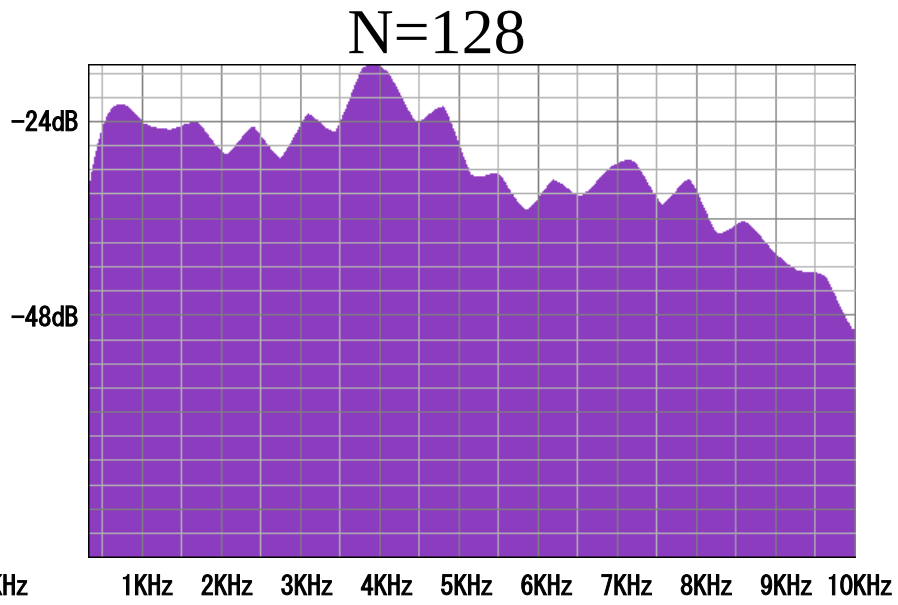



人間の耳はこの付近の周波数に敏感なため、この構造は聴感に大きく影響する。
（これがなかなか判らず苦労しました）

# 方法1の問題点

また、出力が極端に高音の場合ギザギザが残ってしまう。
この場合も不要な構造がノイズの増加等の問題を起こす

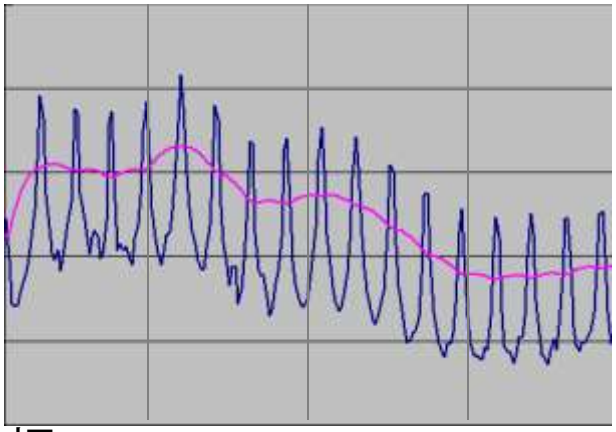


# 『あの曲線』を求める方法2

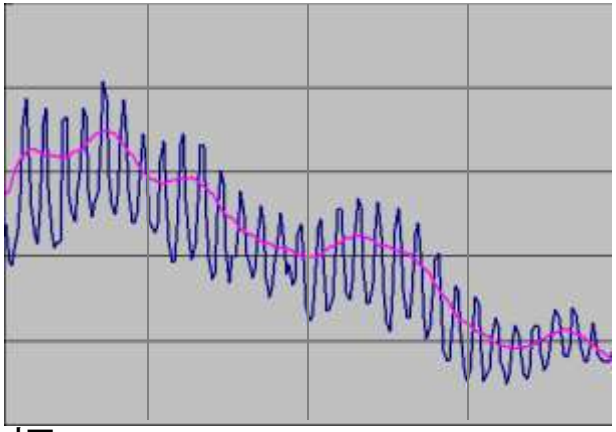
ポイント数は減らさず、スペクトラムを画像の『ぼかし』の
要領で周波数方向にスムージングする

N=1024　C5=523.3Hz



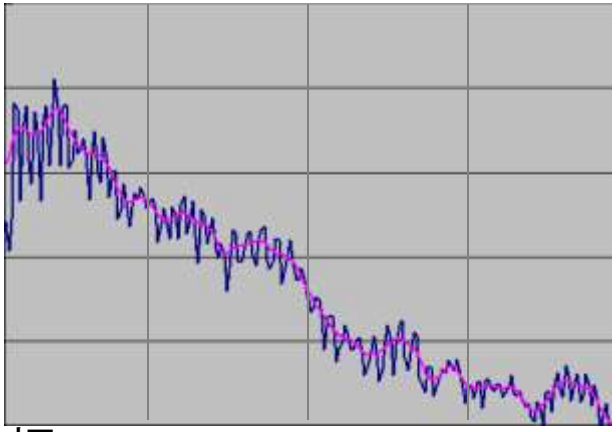
幅=14

N=1024　E4=329.6Hz



幅=9

N=1024　D3=146.8Hz

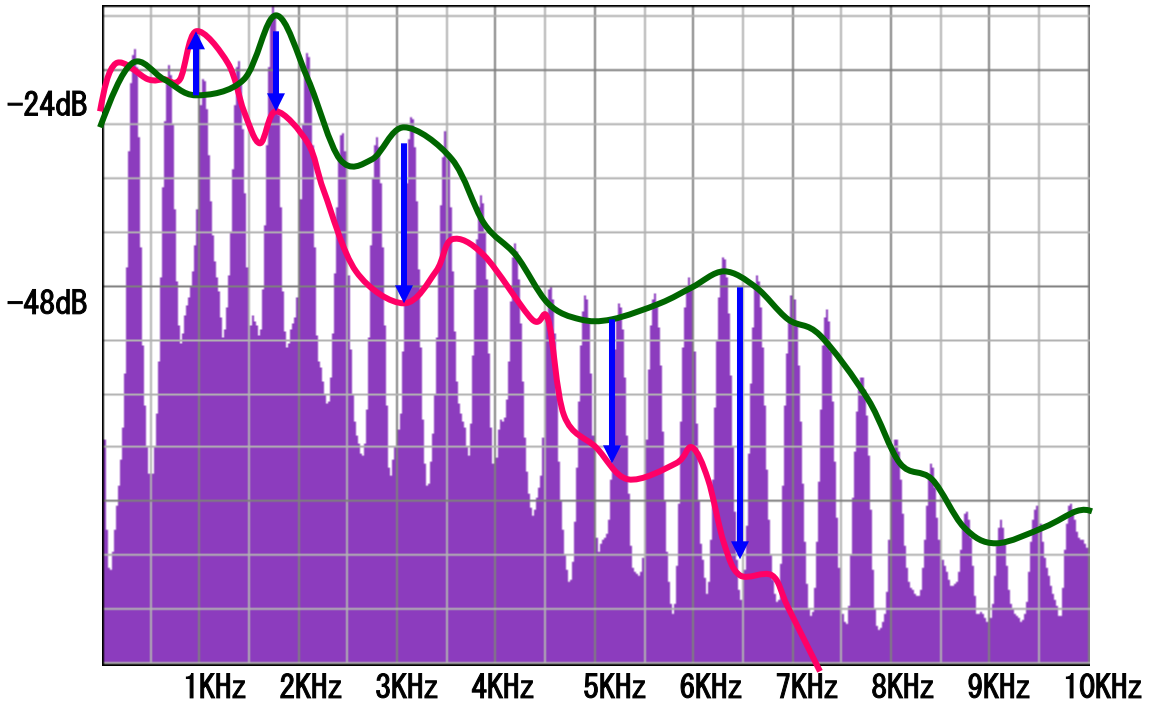


幅=5

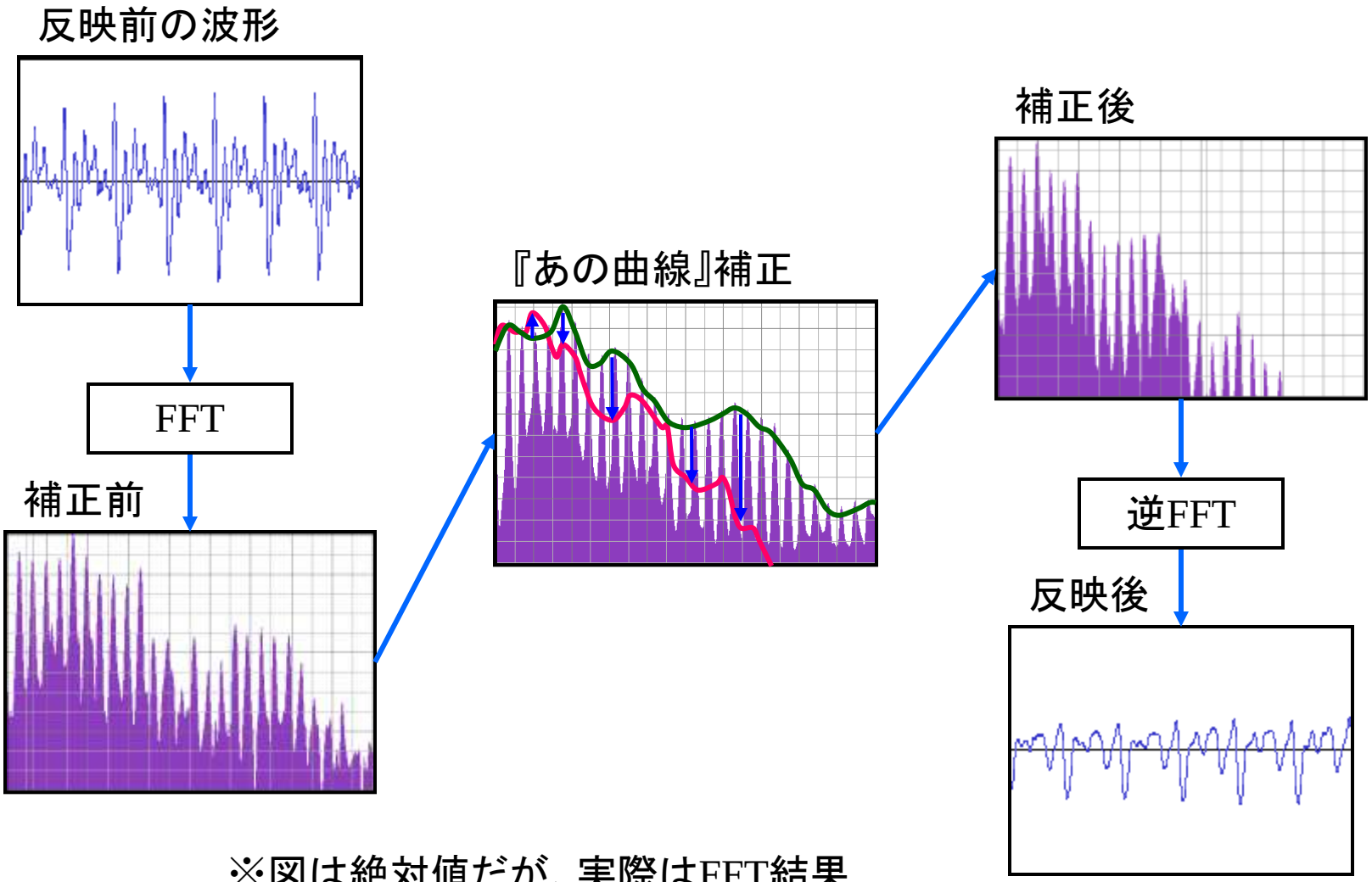
このとき、音程に合わせてスムージング関数の幅
を調節して方法1の問題を回避する。

# ２．求めた曲線を反映させるには？

まず、この図から、『あの曲線』の“入力側／出力側”を
元のスペクトラムに掛け算すれば良いらしいと判る。



# 2. 求めた曲線を反映させるには？
# つまり、



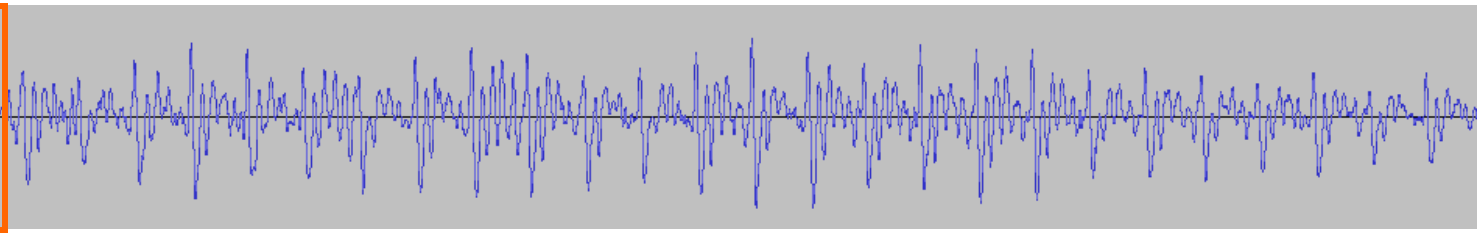

※図は絶対値だが、実際はFFT結果を複素数のまま計算

# あの曲線は時間で変化するので

短い単位で曲線も計算しつつ順に反映していく

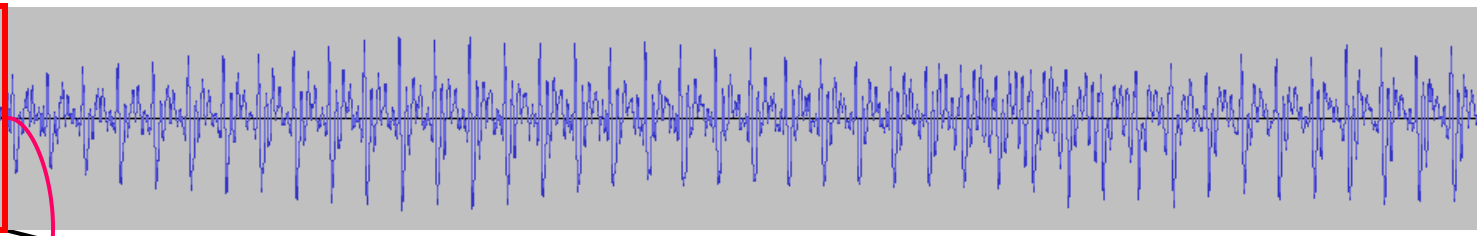
元音声

伸縮前の該当個所



曲線反映前

曲線反映前

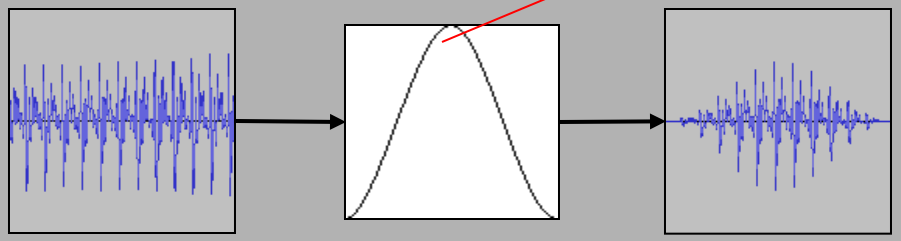


あの曲線

出力波形

最終出力

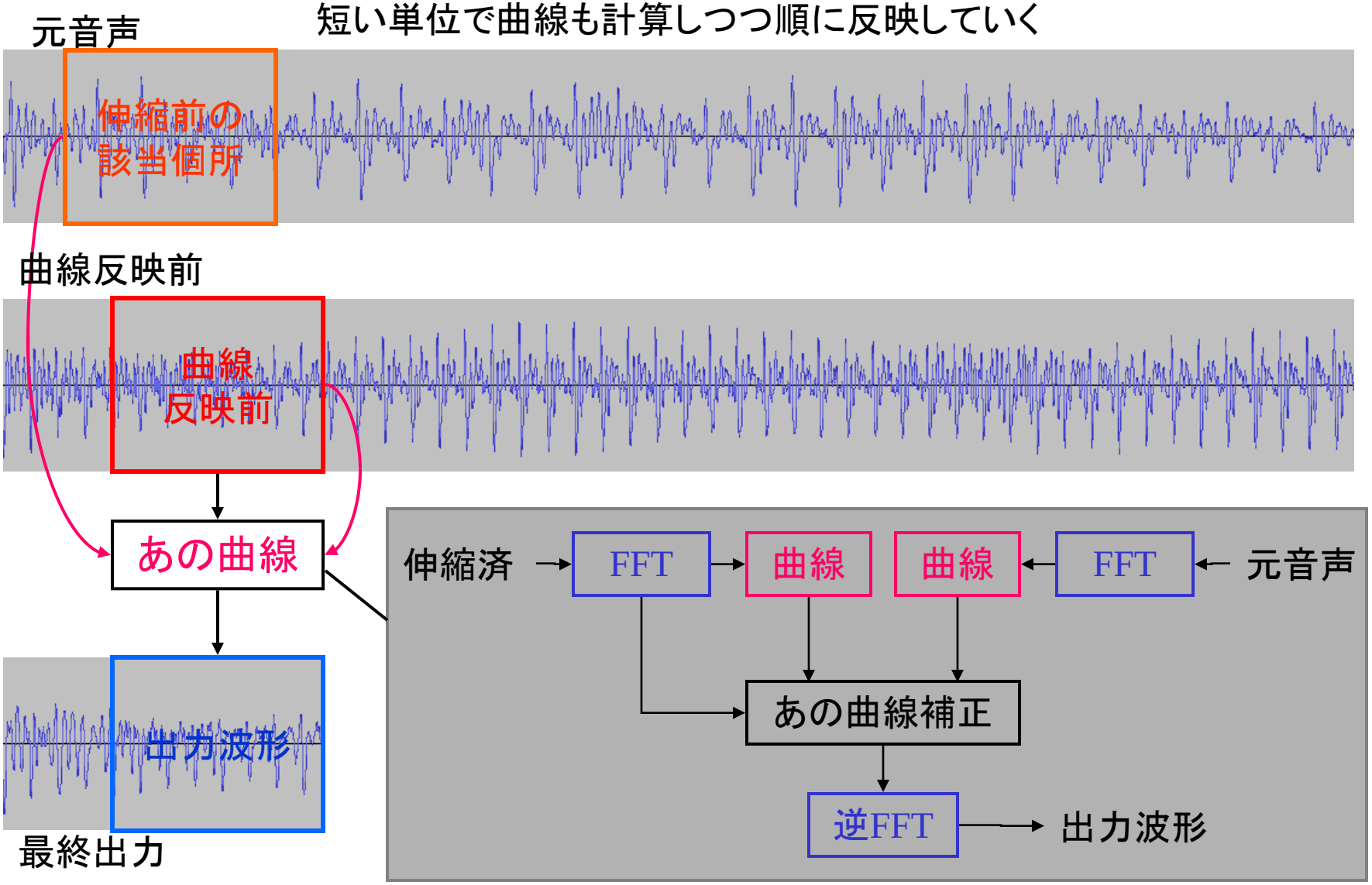
処理する波形は切り出すときこんな窓を掛けて、



曲線反映後は波形が半分重なるように出力していく

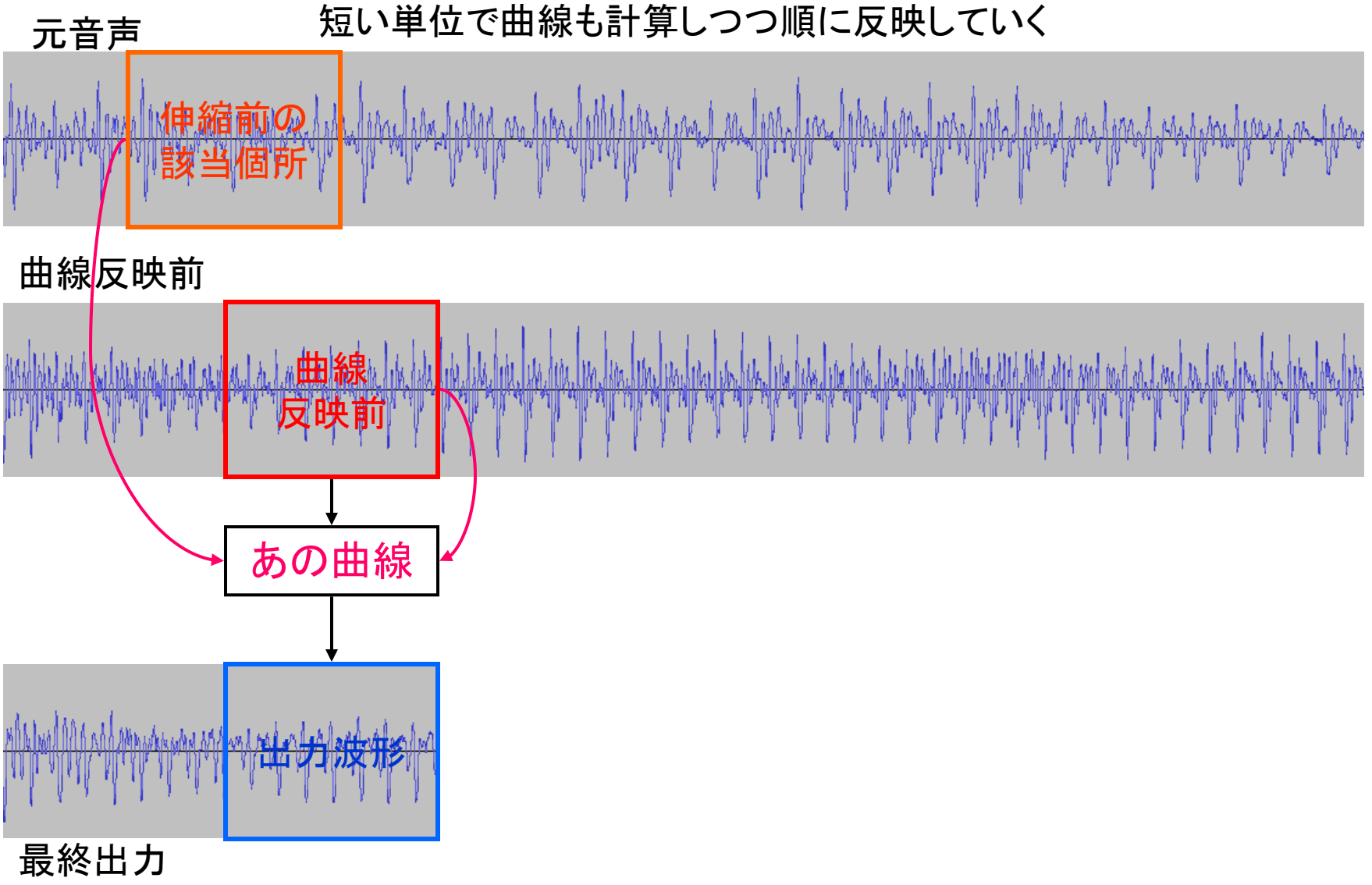
# あの曲線は時間で変化するので

短い単位で曲線も計算しつつ順に反映していく



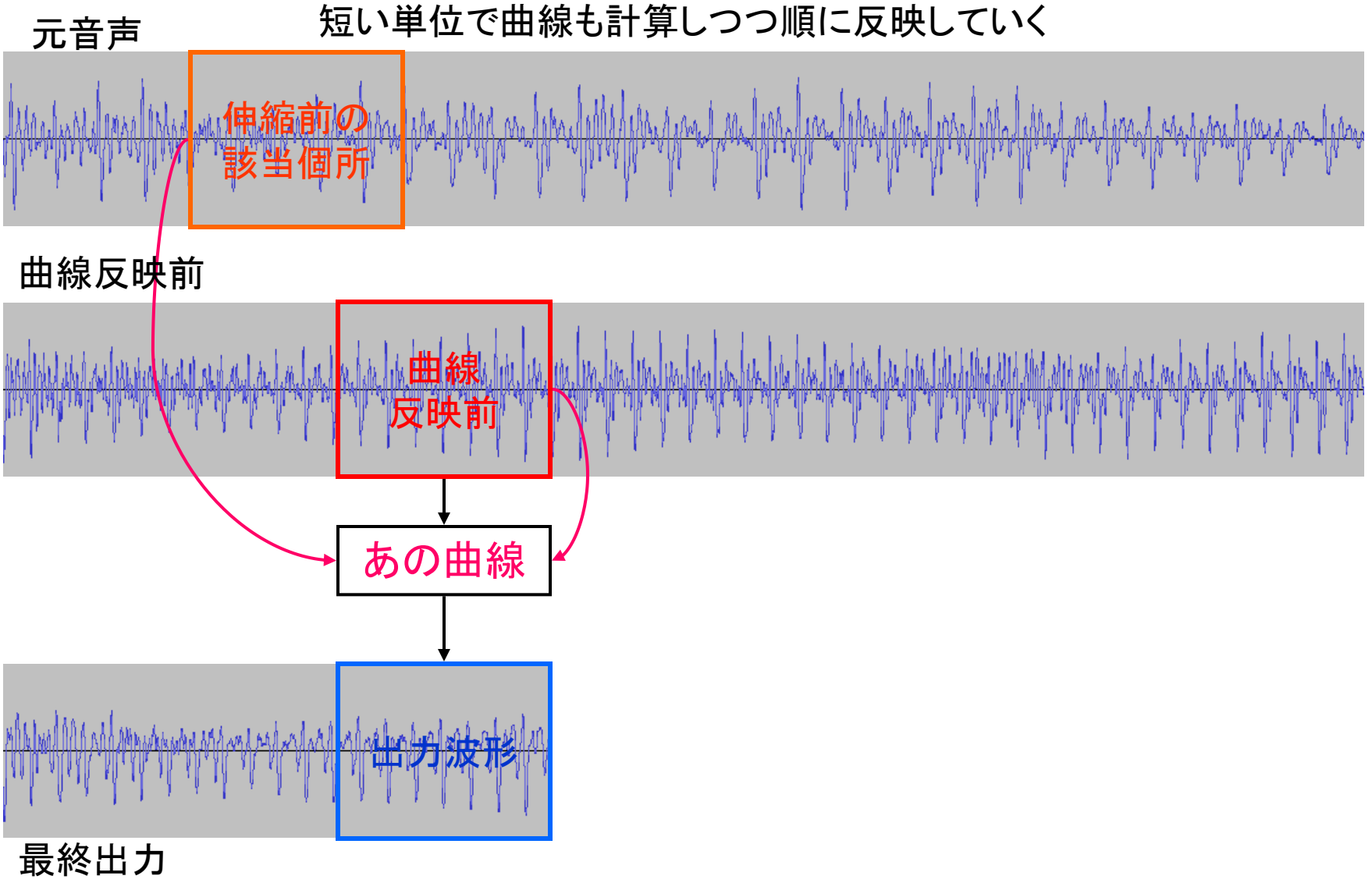
# あの曲線は時間で変化するので

短い単位で曲線も計算しつつ順に反映していく



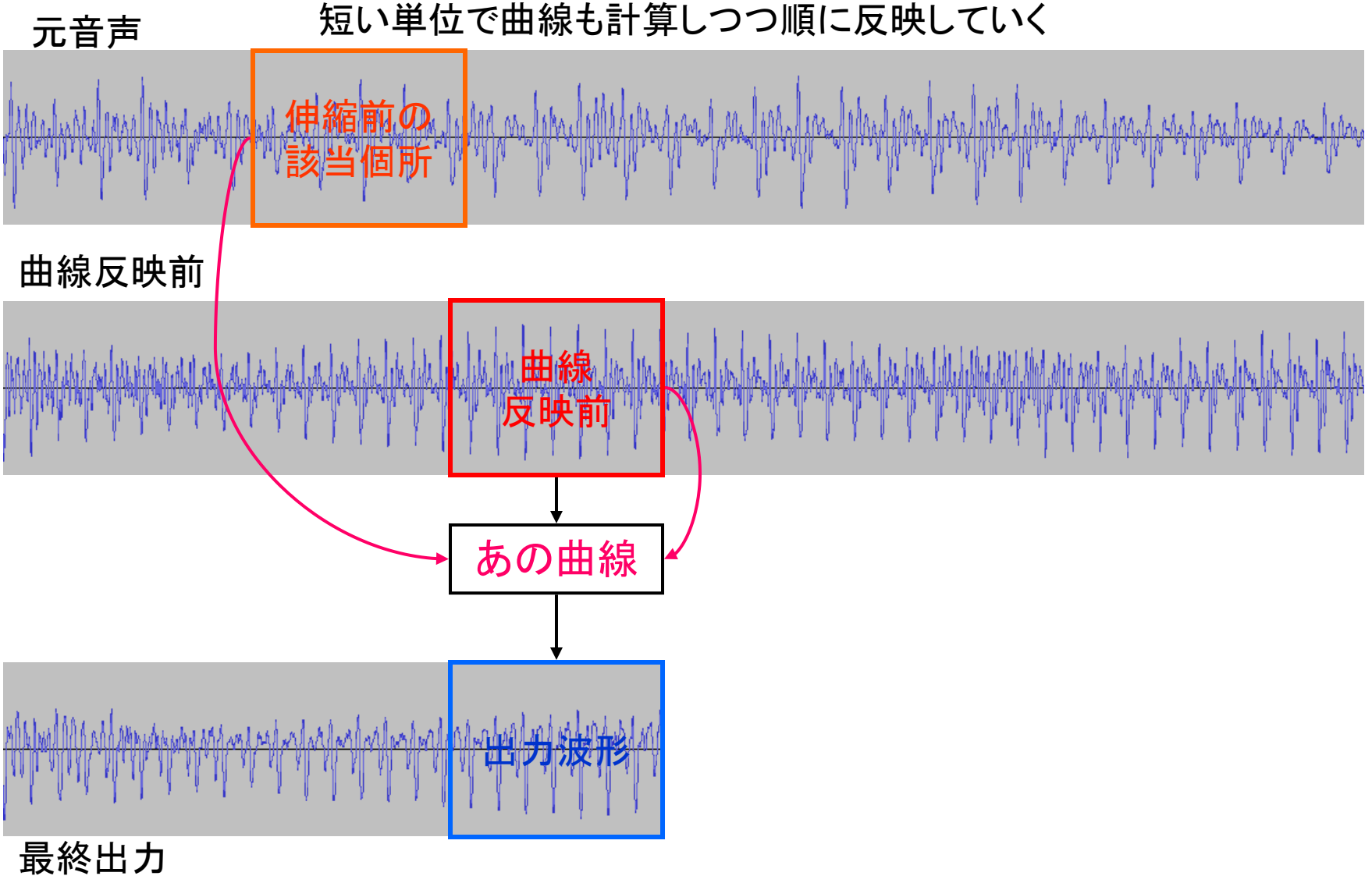
# あの曲線は時間で変化するので

短い単位で曲線も計算しつつ順に反映していく

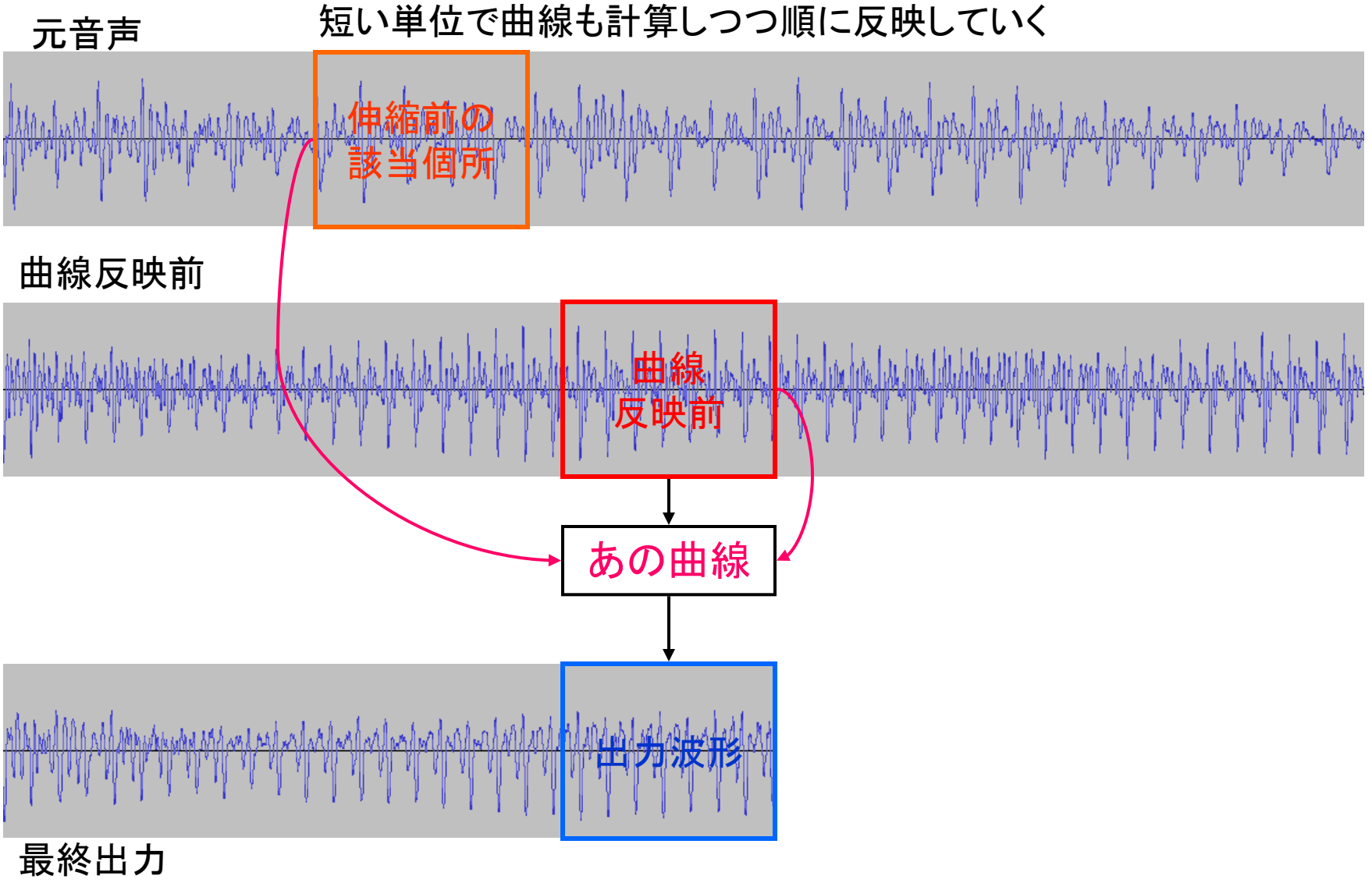


# あの曲線は時間で変化するので

短い単位で曲線も計算しつつ順に反映していく

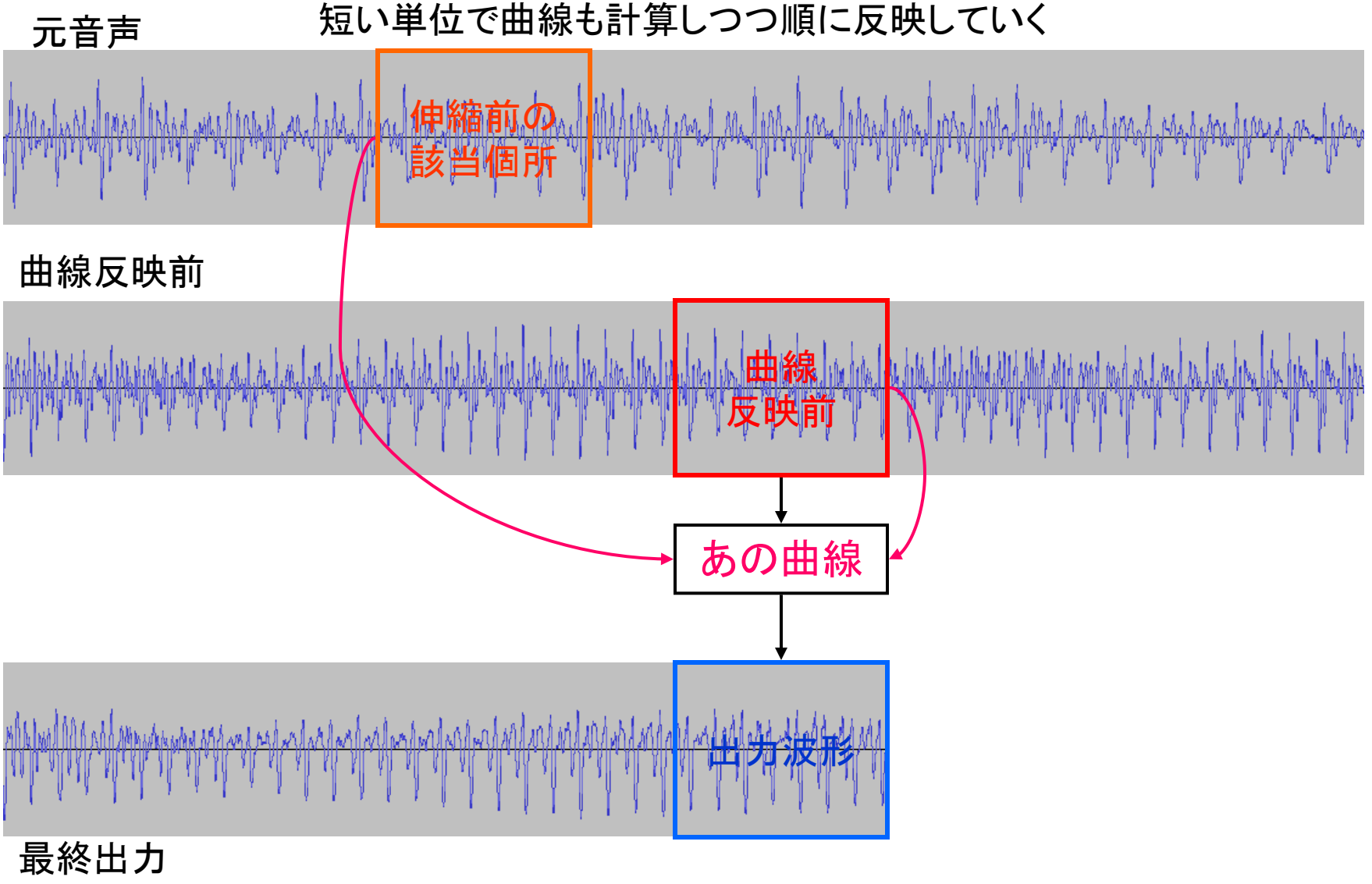


# あの曲線は時間で変化するので



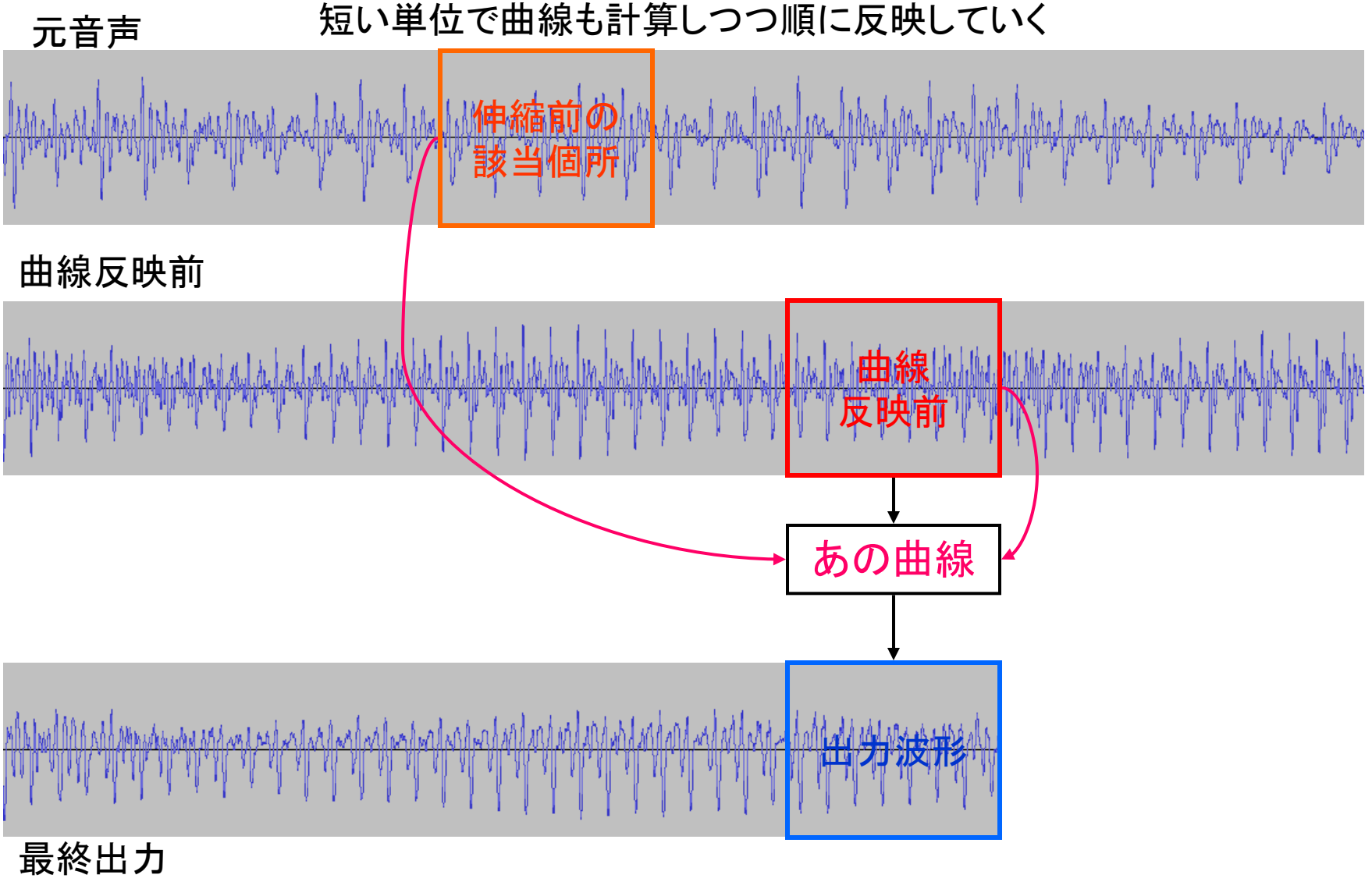
# あの曲線は時間で変化するので

短い単位で曲線も計算しつつ順に反映していく



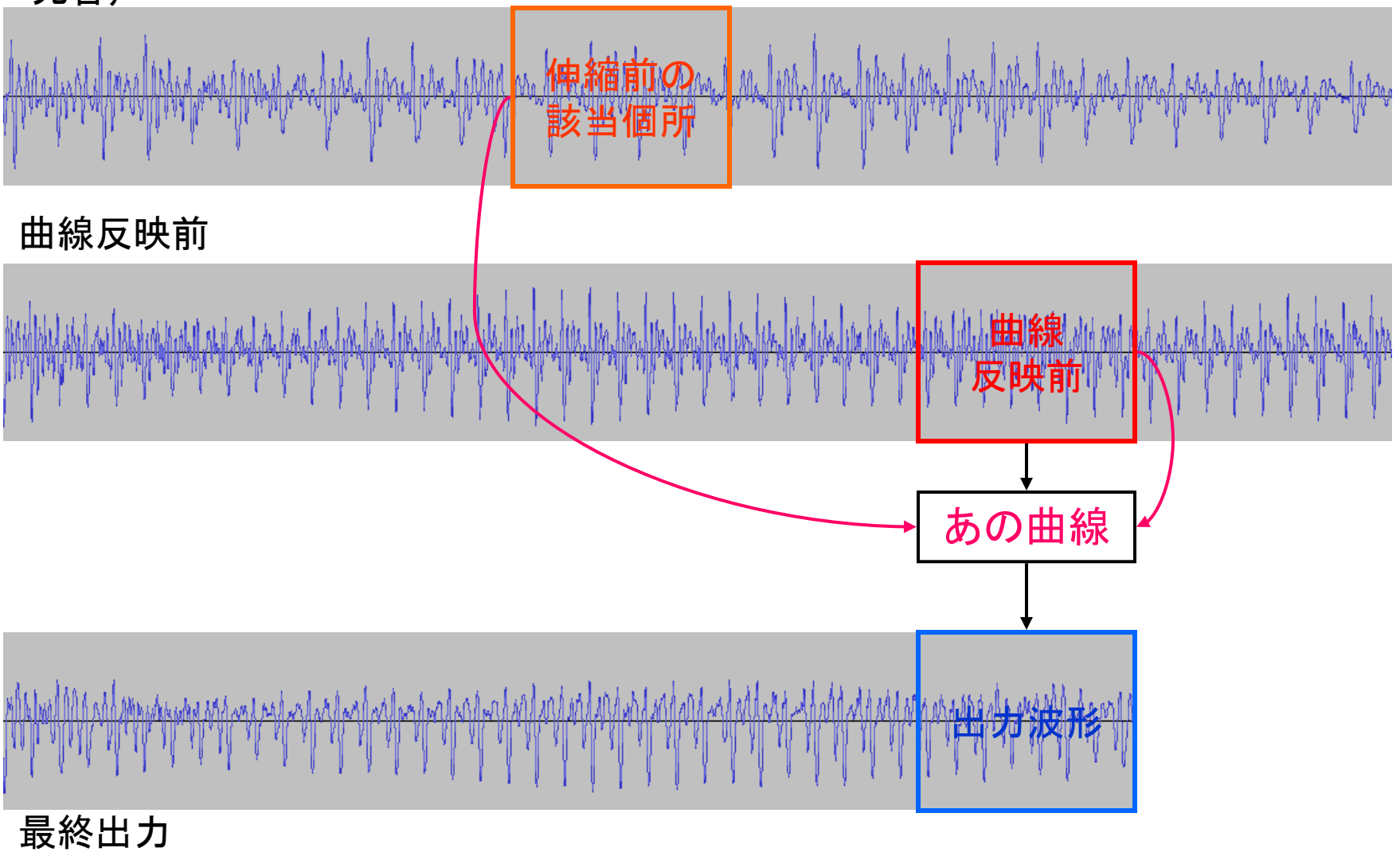
# あの曲線は時間で変化するので

短い単位で曲線も計算しつつ順に反映していく

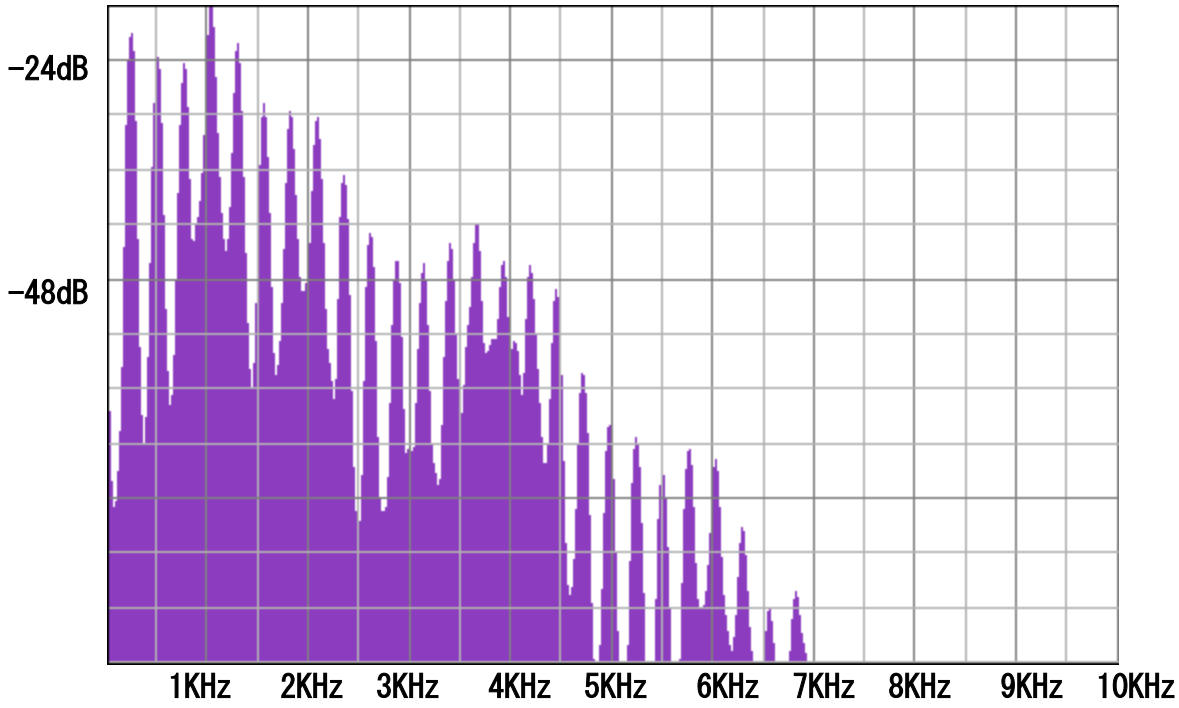


# あの曲線は時間で変化するので

短い単位で曲線も計算しつつ順に反映していく

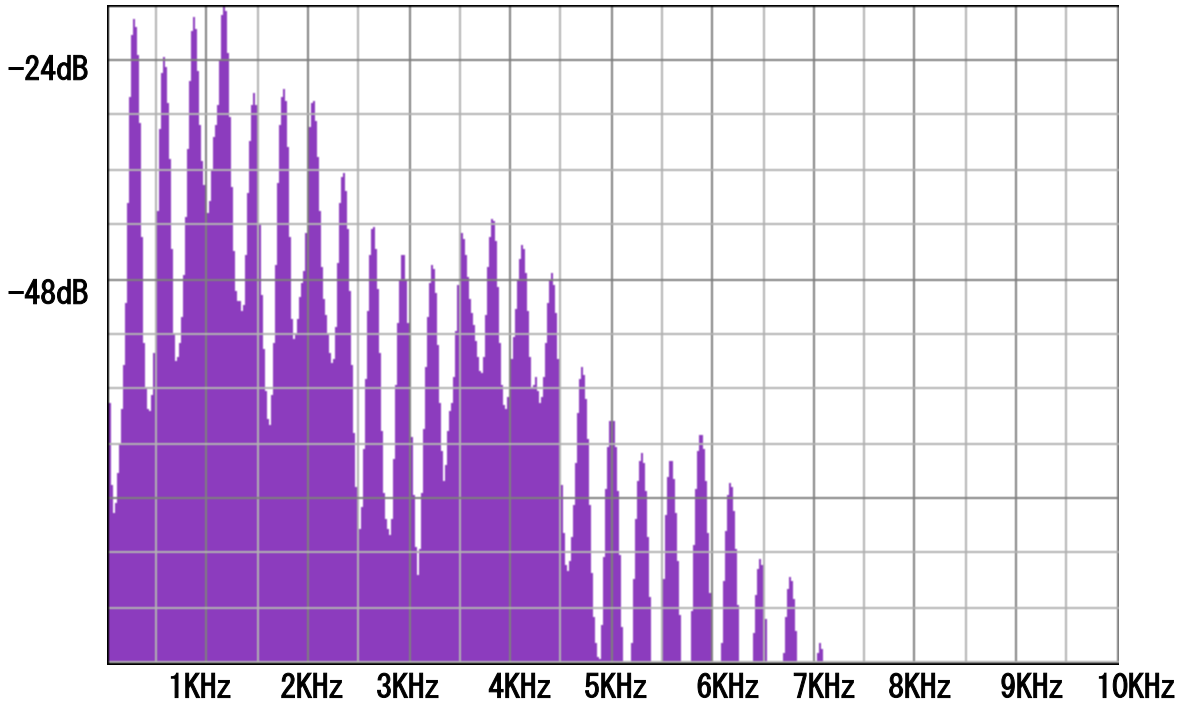


# あの曲線を元に戻した結果



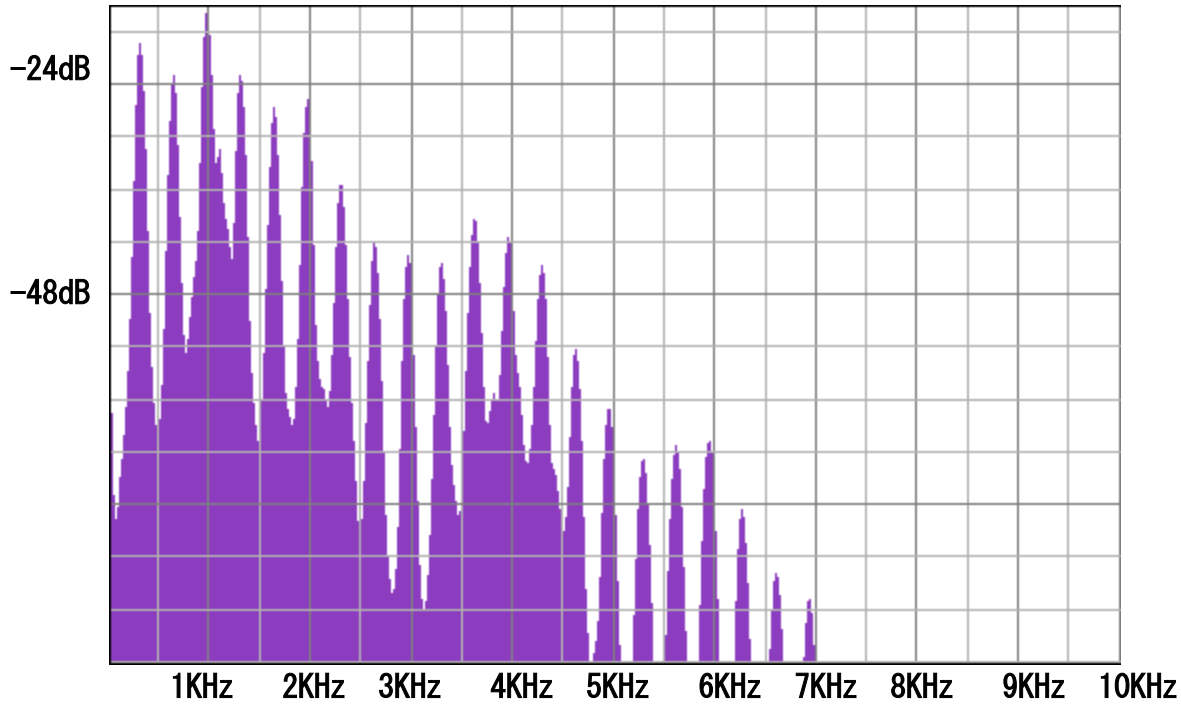

「あ」G#3(206.4Hz)→C4(216.6Hz)

# あの曲線を元に戻した結果



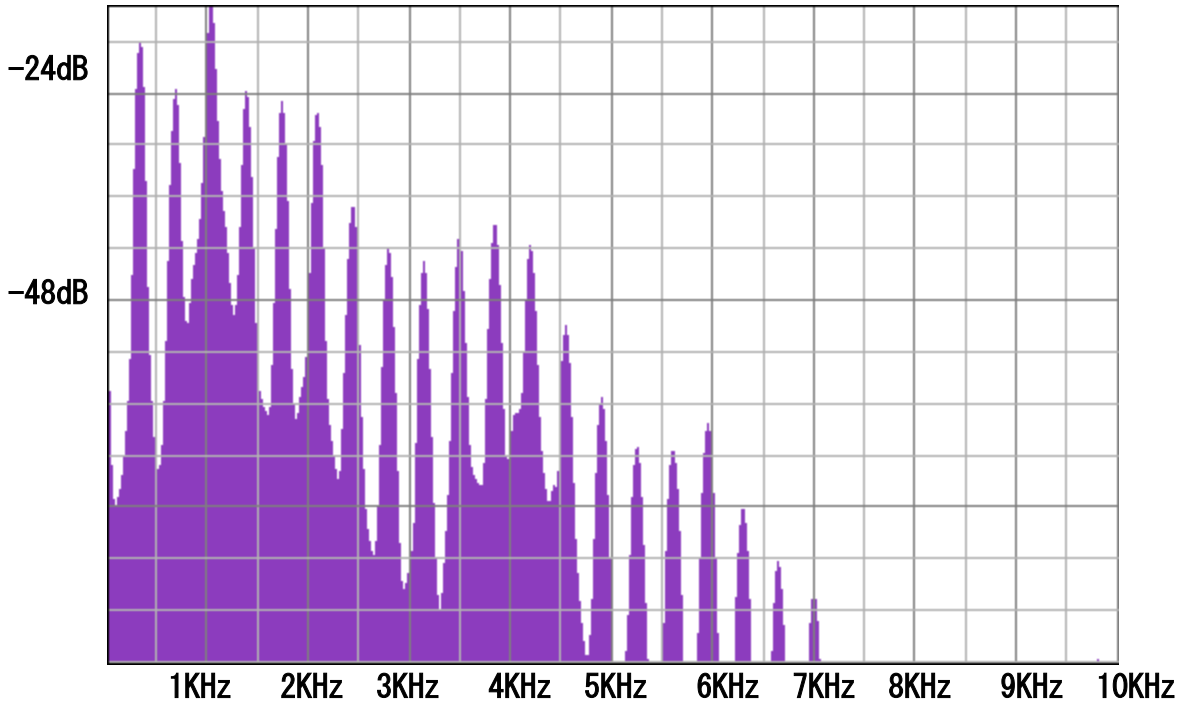

「あ」G#3(206.4Hz)→D4(293.7Hz)

# あの曲線を元に戻した結果



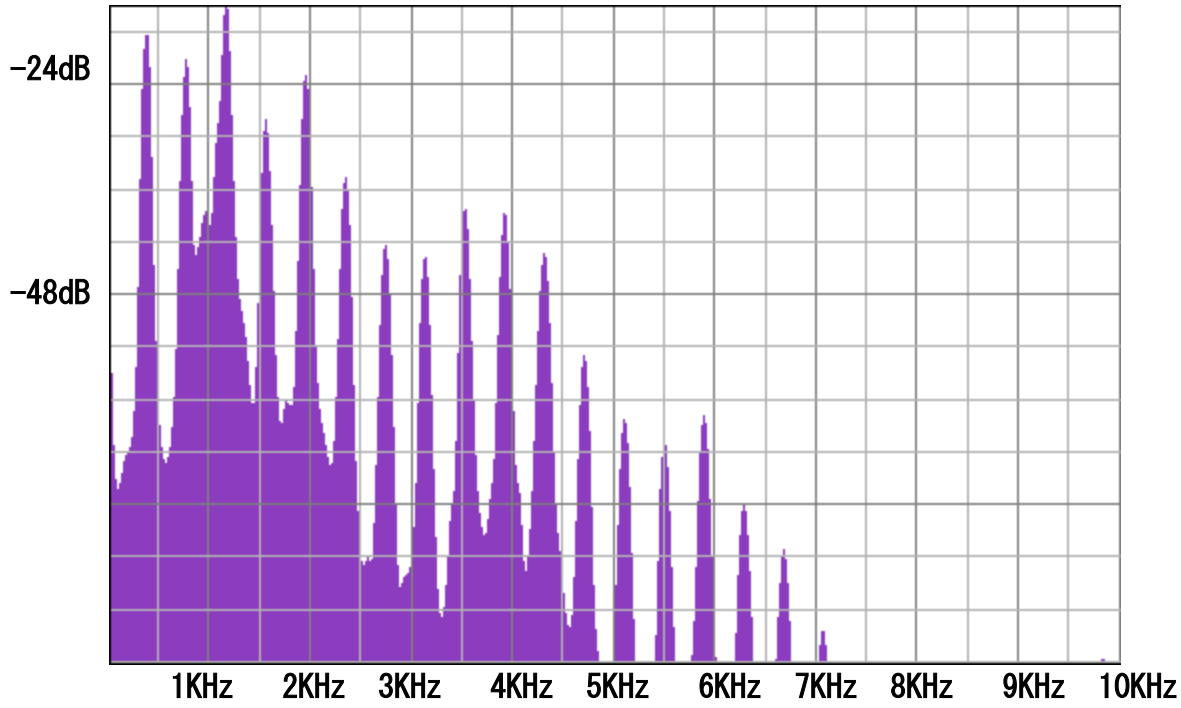

「あ」G#3(206.4Hz)→E4(329.6Hz)

# あの曲線を元に戻した結果



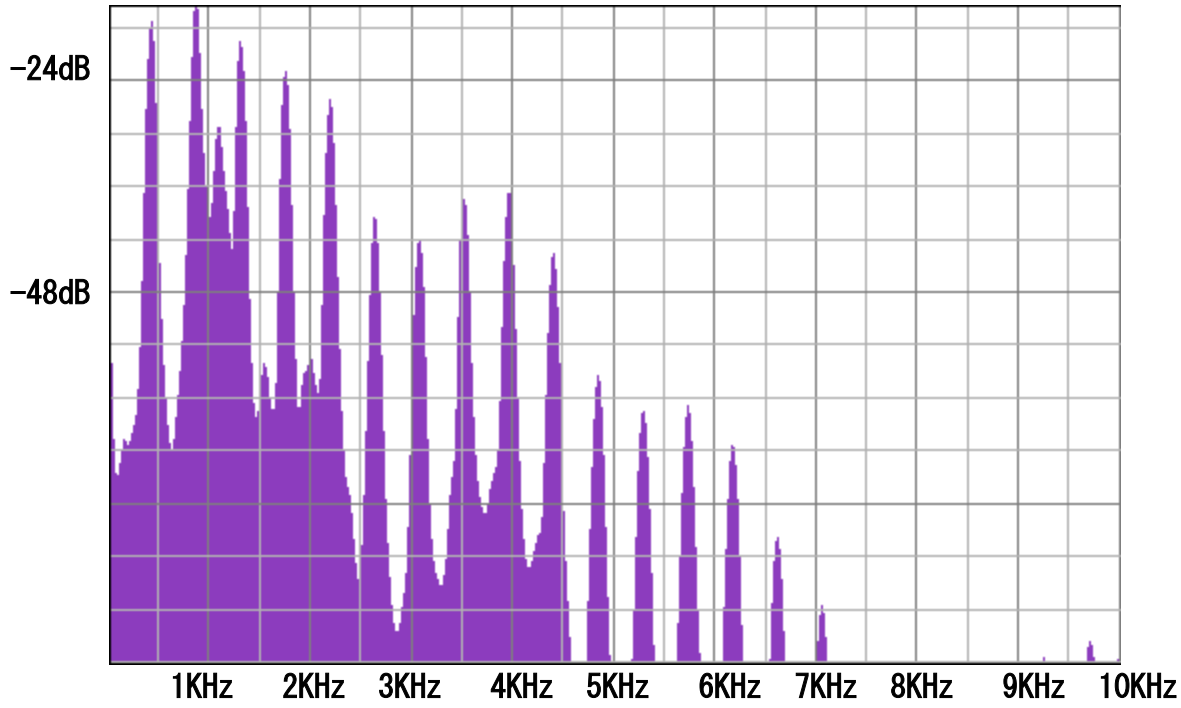

「あ」G#3(206.4Hz)→F4(349.2Hz)

# あの曲線を元に戻した結果



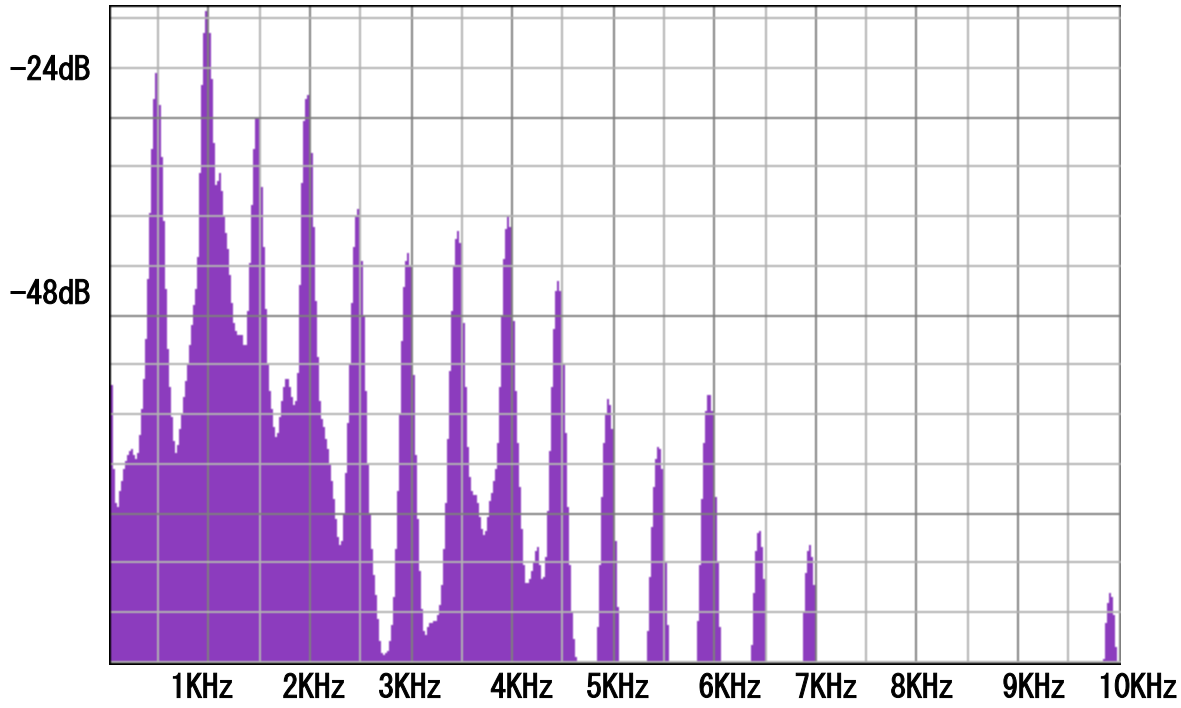

「あ」G#3(206.4Hz)→G4(392.0Hz)

# あの曲線を元に戻した結果



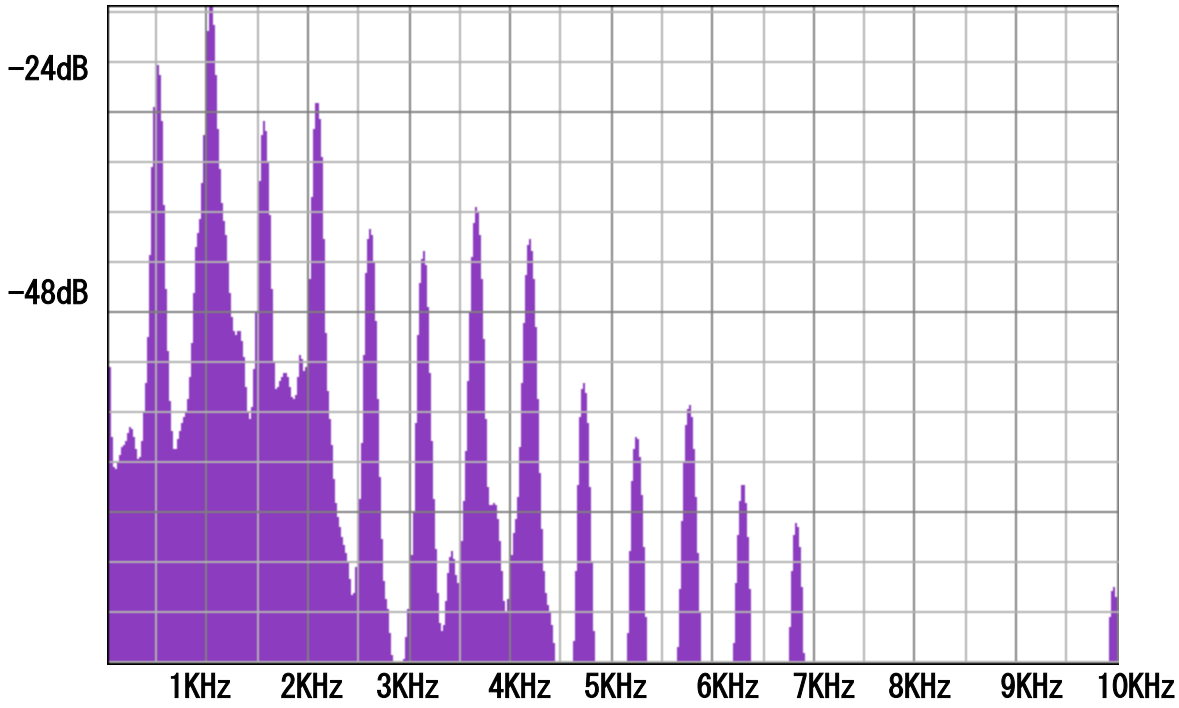

「あ」G#3(206.4Hz)→A4(440.0Hz)

# あの曲線を元に戻した結果



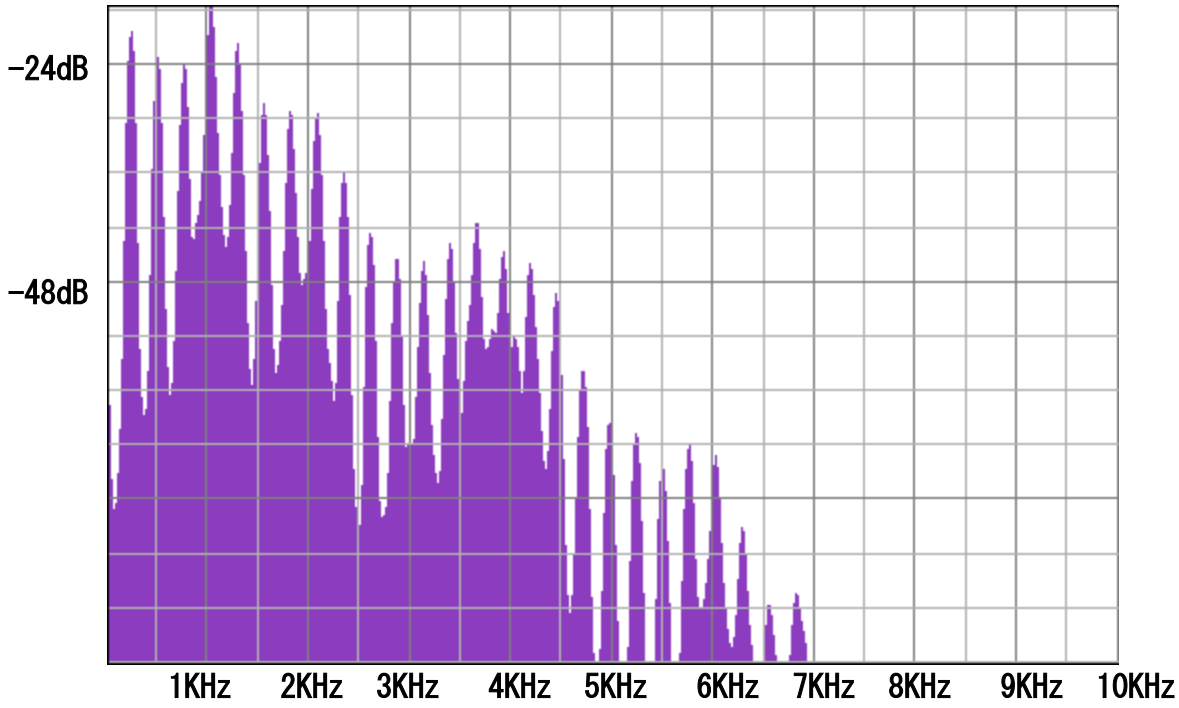

「あ」G#3(206.4Hz)→B4(493.9Hz)

# あの曲線を元に戻した結果



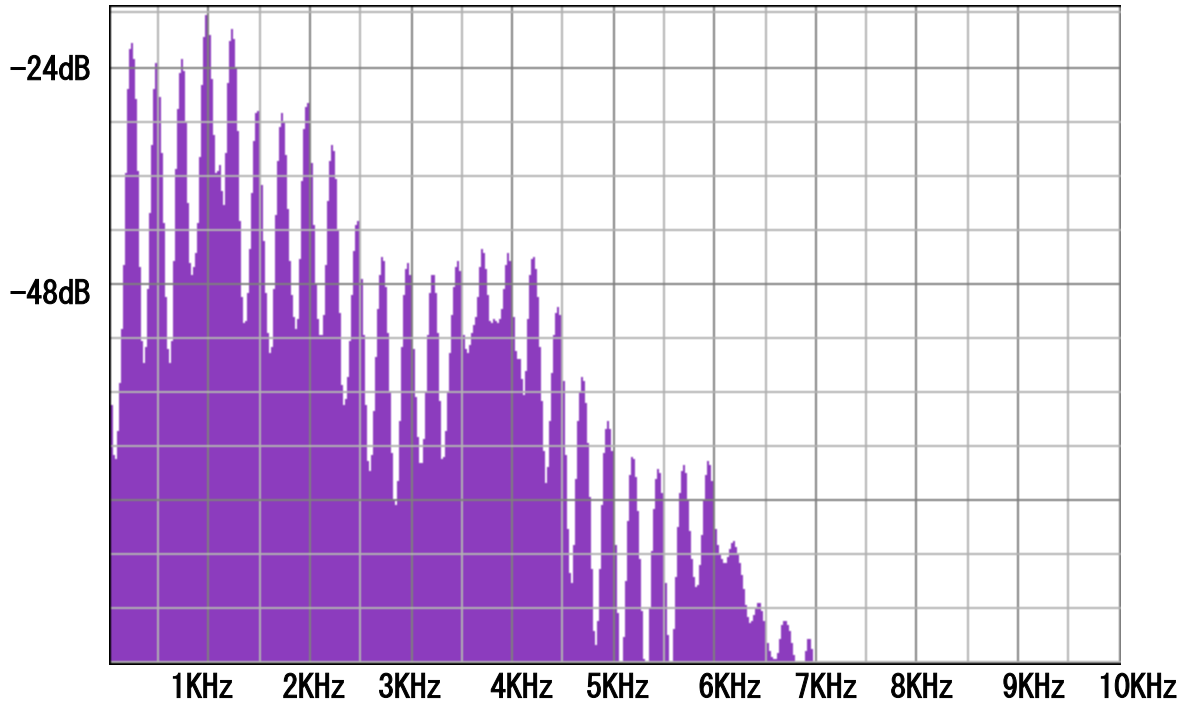

「あ」G#3(206.4Hz)→C5(523.3Hz)

# あの曲線を元に戻した結果



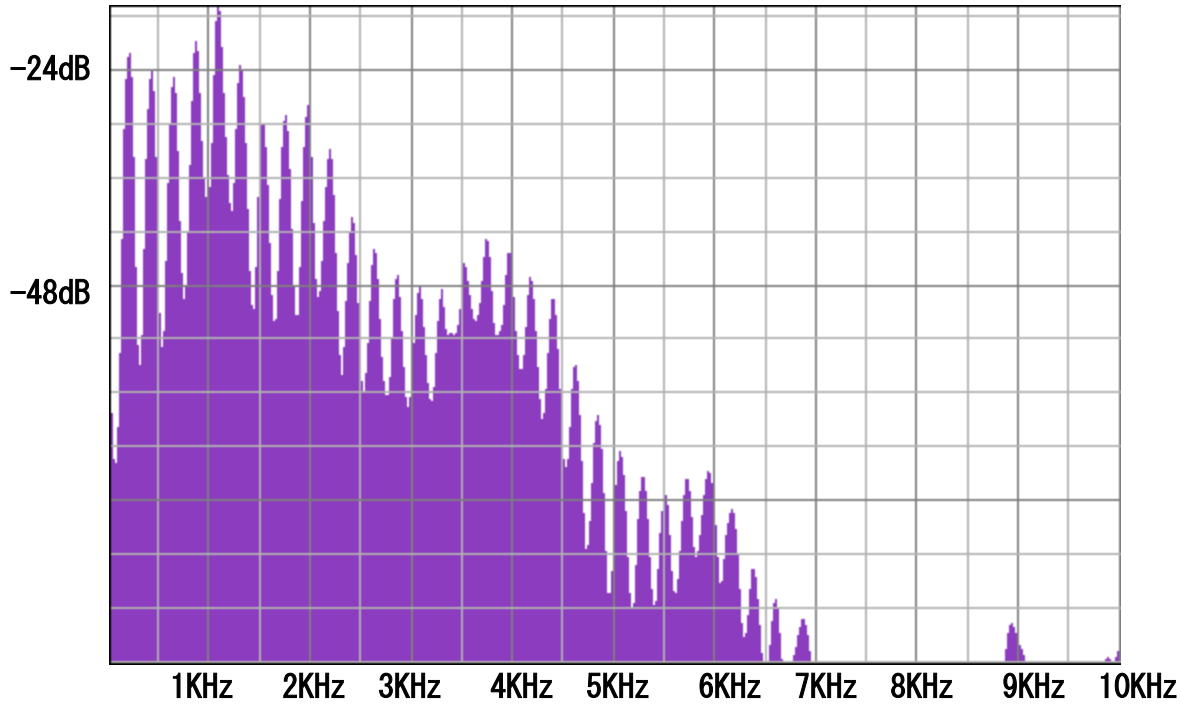

「あ」G#3(206.4Hz)→C4(216.6Hz)

# あの曲線を元に戻した結果



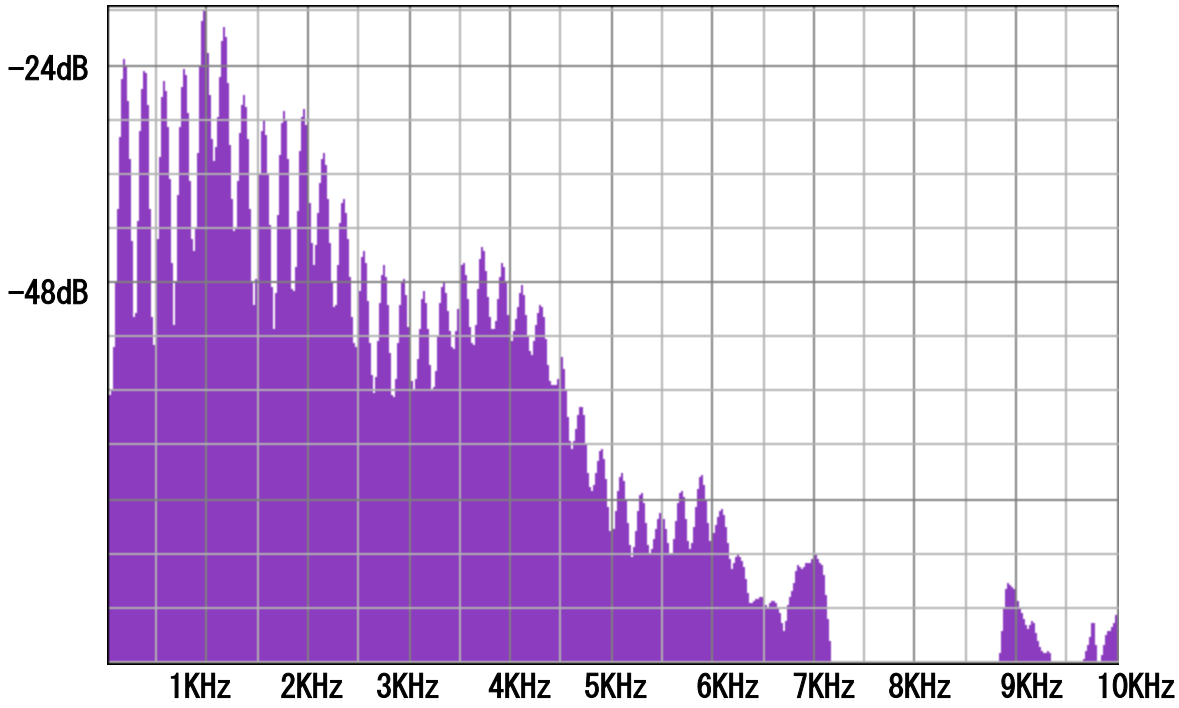

「あ」G#3(206.4Hz)→C4(216.6Hz)

# あの曲線を元に戻した結果



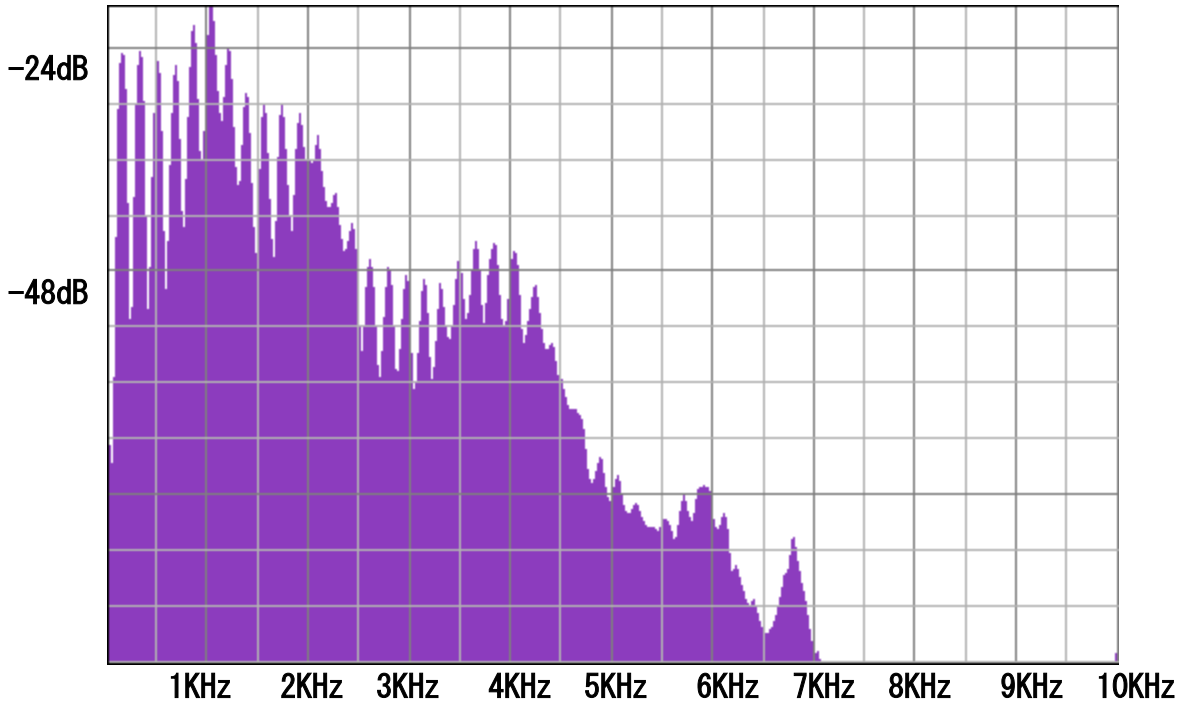

「あ」G#3(206.4Hz)→C4(216.6Hz)

# あの曲線を元に戻した結果



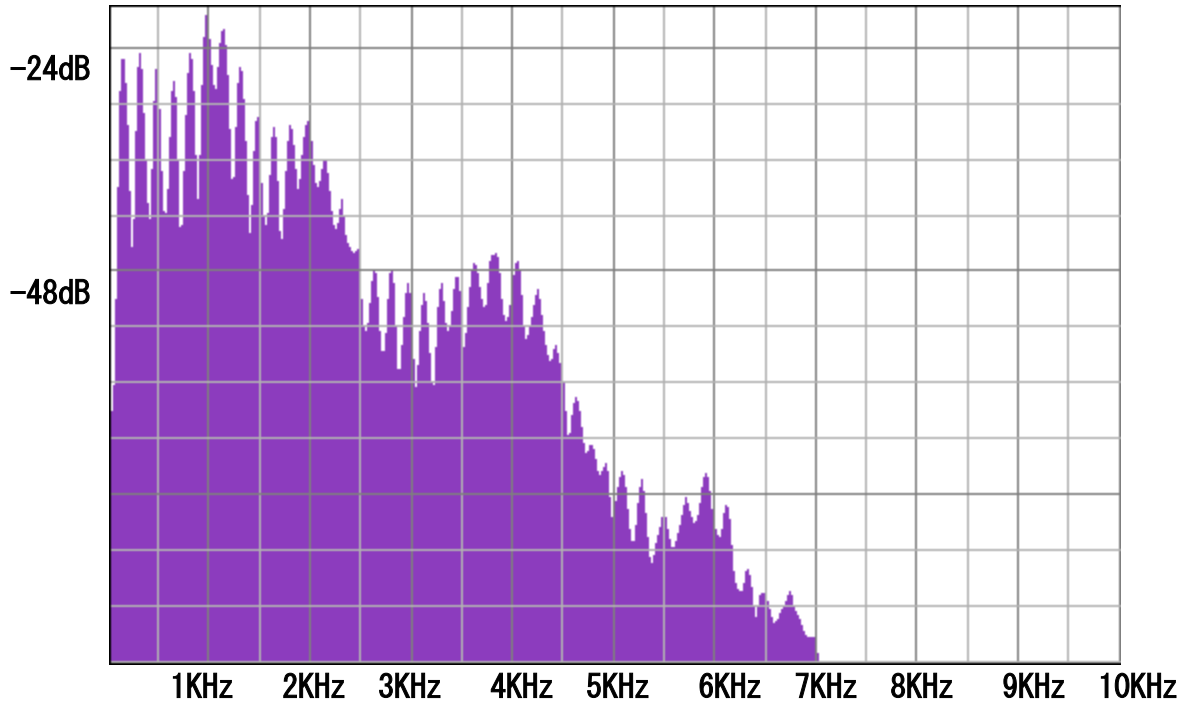

「あ」G#3(206.4Hz)→C4(216.6Hz)

# あの曲線を元に戻した結果



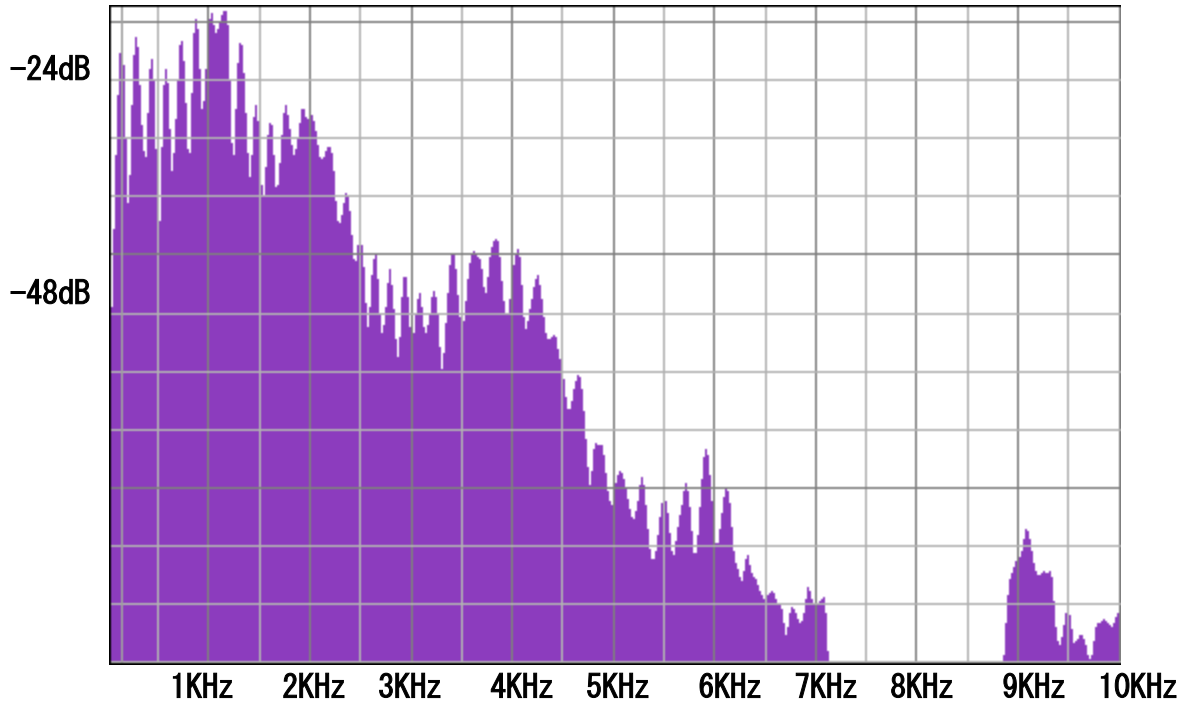

「あ」G#3(206.4Hz)→C4(216.6Hz)

# あの曲線を元に戻した結果



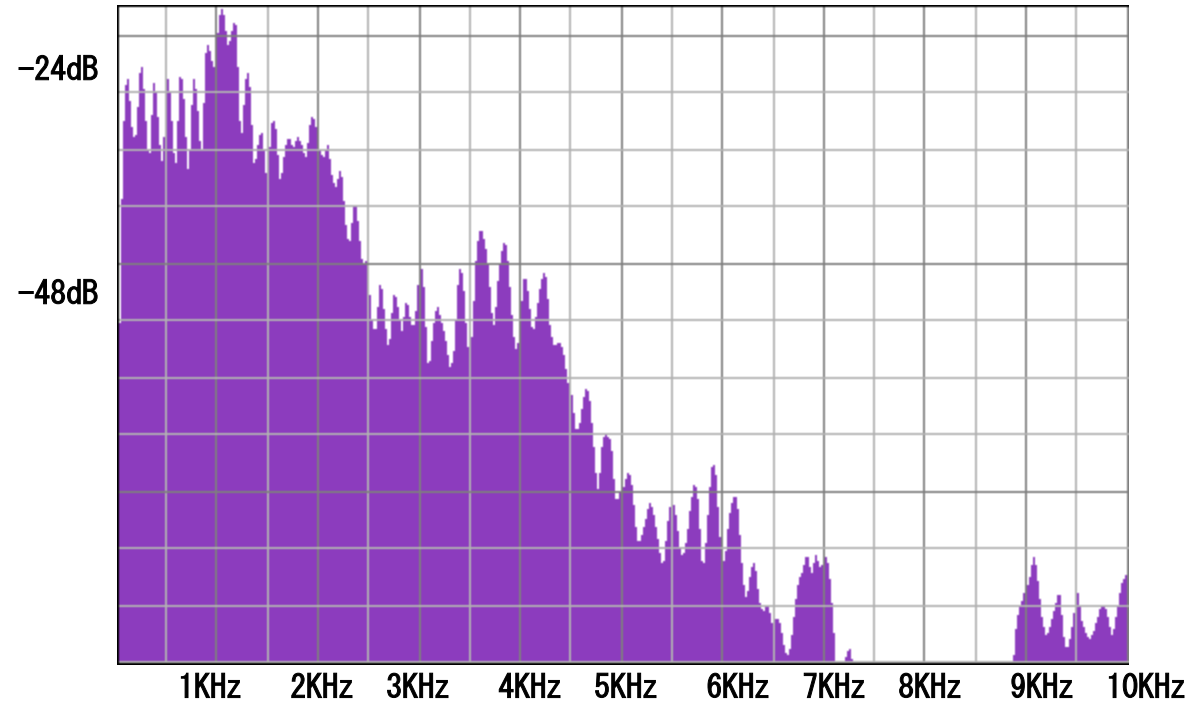

「あ」G#3(206.4Hz)→C4(216.6Hz)

# あの曲線を元に戻した結果



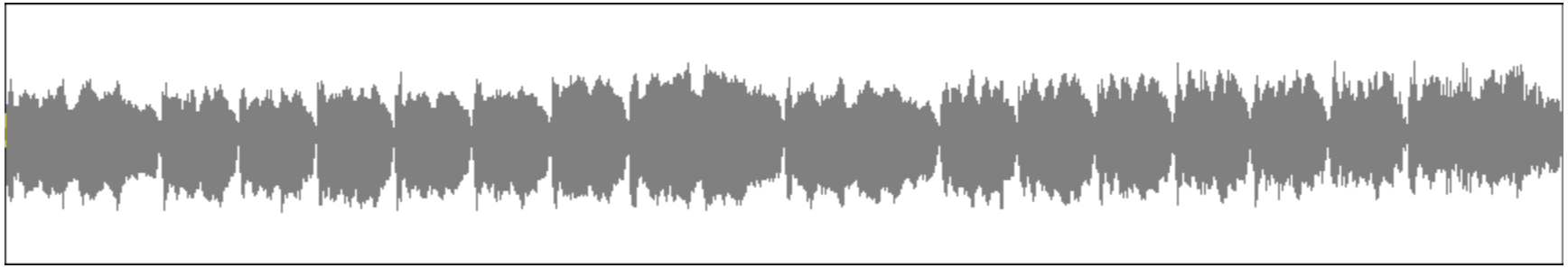

「あ」G#3(206.4Hz)→C4(216.6Hz)

# あの曲線を元に戻した結果



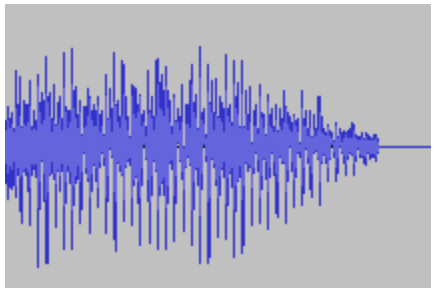

「あ」G#3(206.4Hz)→C4(216.6Hz)

# 出力サンプル



参考：処理前

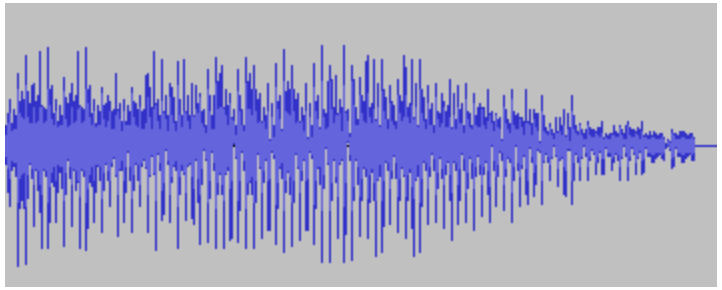
# 音程変更と長さの変更できた



こういう元音声があったとき、

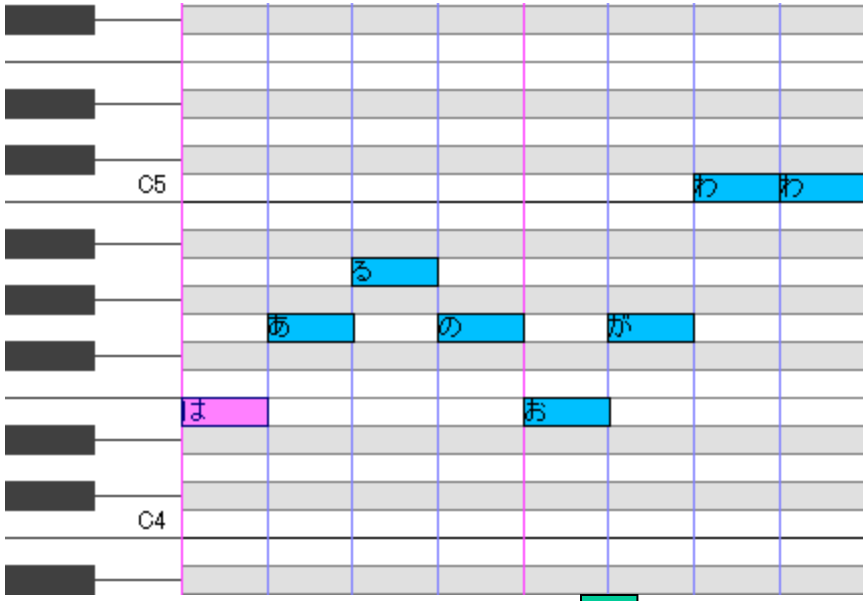
>resamp 元ﾌｧｲﾙ 出力ﾌｧｲﾙ 音程 長さ

このようにパラメータを指定して



任意の長さ・任意の音程に変更できるコマンドができた！

# 完成！

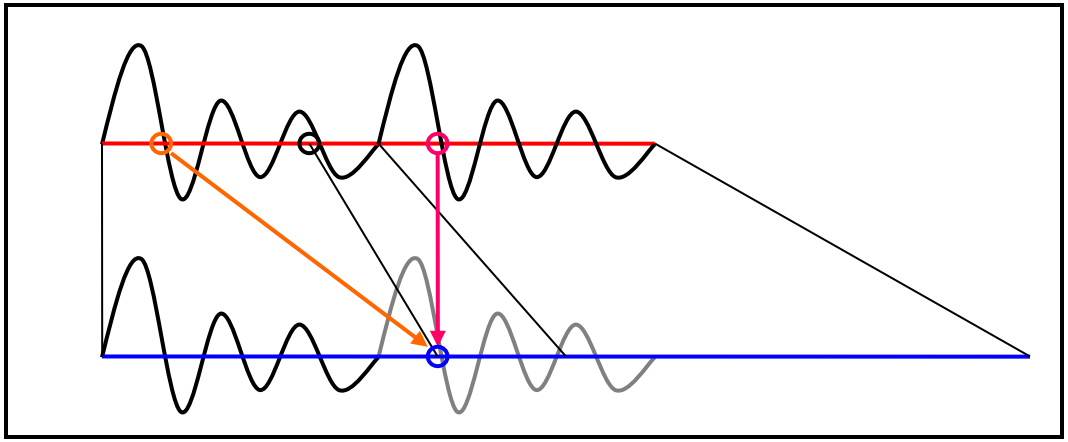



バッチファイルが複雑化して、テキスト編集では効率が悪くなった

```
resamp は.wav tmp.wav E4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp あ.wav tmp.wav G4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp る.wav tmp.wav A4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp の.wav tmp.wav G4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp お.wav tmp.wav E4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp が.wav tmp.wav G4 500
wavtool tmp.wav output.wav 0 500 12 24
resamp わ.wav tmp.wav C5 500
wavtool tmp.wav output.wav 0 500 12 24
resamp わ.wav tmp.wav C5 500
wavtool tmp.wav output.wav 0 500 12 24
```

バッチファイルを生成するGUIフロントエンド

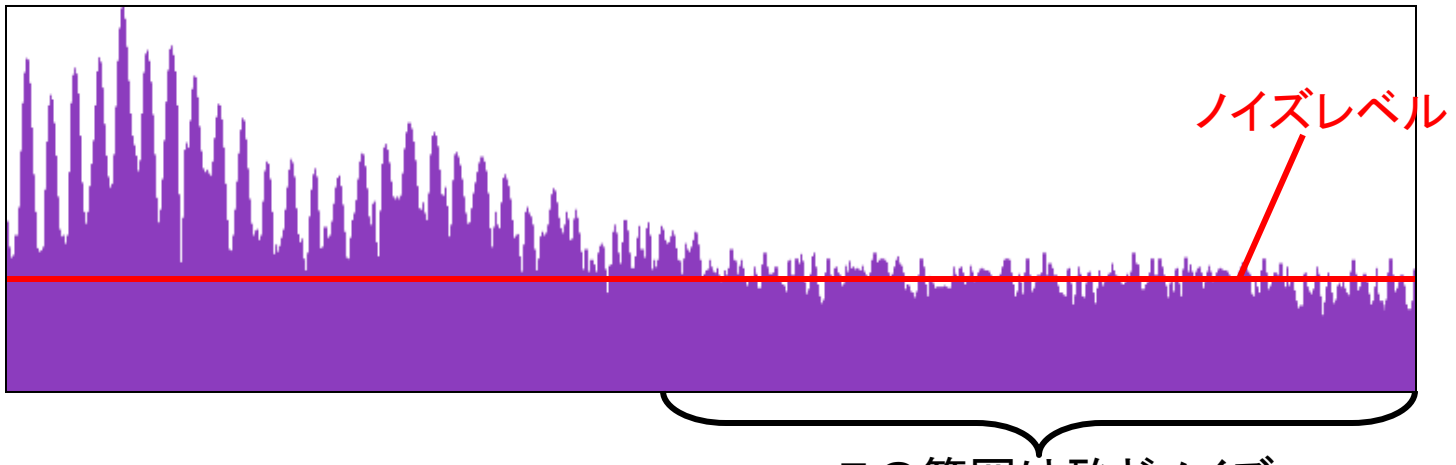
# 伸縮方法に残る問題点

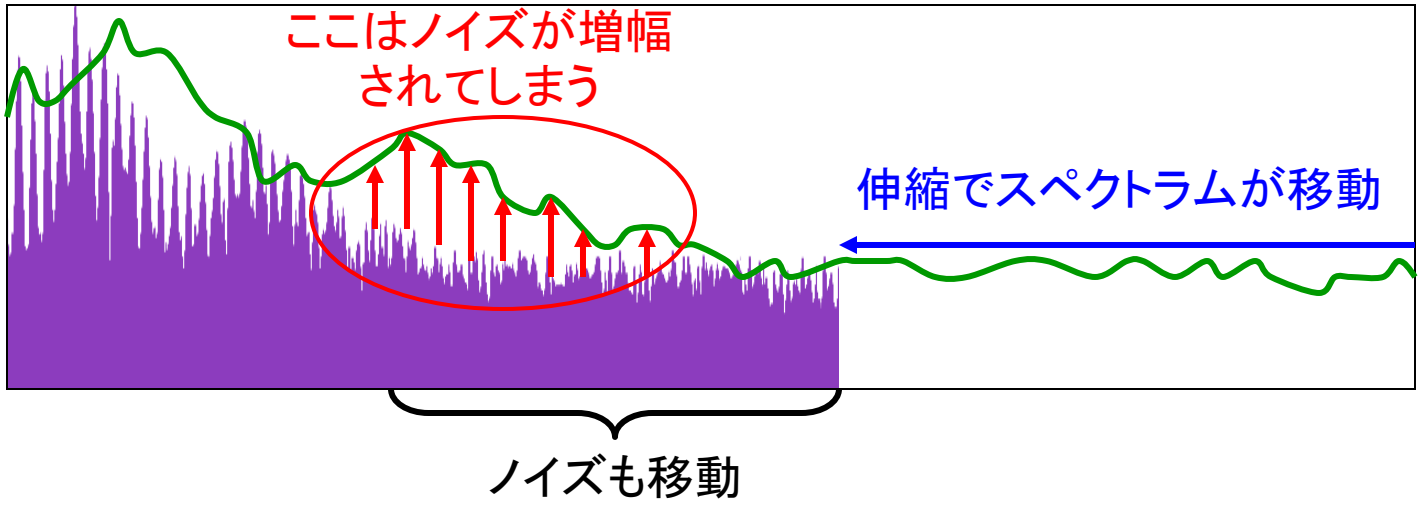


- 常に一波長ずれた波を重ねる構造な為、基本周波数とその高調波（スペクトラムの『トゲトゲ』）が強調されてしまう
- 二つのサンプル点のブレンド比率が変化する周期と、声の基本周波数が合成されたノイズが発生することがある

# この方法（全体）の問題点

ノイズが乗ってる音源の場合



# 用語

- 音声伸縮→タイムストレッチ
- 音程変更→ピッチシフト
- 声のキャラクタを保つ→フォルマントシフト
- あの曲線→スペクトラム包絡

# その他のUTAUに関わる技術トピック

- 元音声にあるピッチの揺らぎをどうするか？→モジュレーション
- 子音の発声タイミングを前にずらさないと『リズム音痴』になる→先行発声
- 母音を先行する音節の最後とクロスフェードするとより滑らかに聞こえる→母音結合
- 音節をクロスフェードする際にピッチを揃える→オートピッチコントロール
- など

# 大きく分けて二つの仕組み

1．今まで述べたような、ピッチ変更・音声伸縮といった基礎的な技術

2．上記を踏まえて楽譜データからどのように歌唱を組み立てていくかという、よりマクロな技術

# UTAUのマルチプラットフォーム展開

- Windows版
- Mac OSX版：utau-synth
- Android版（開発中）
- 組み込み版（研究中）