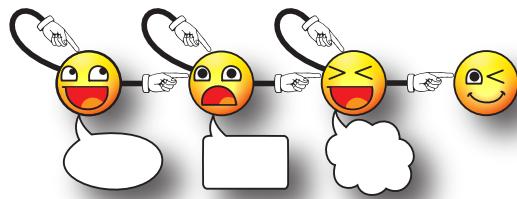


# Spoken Word Recognition

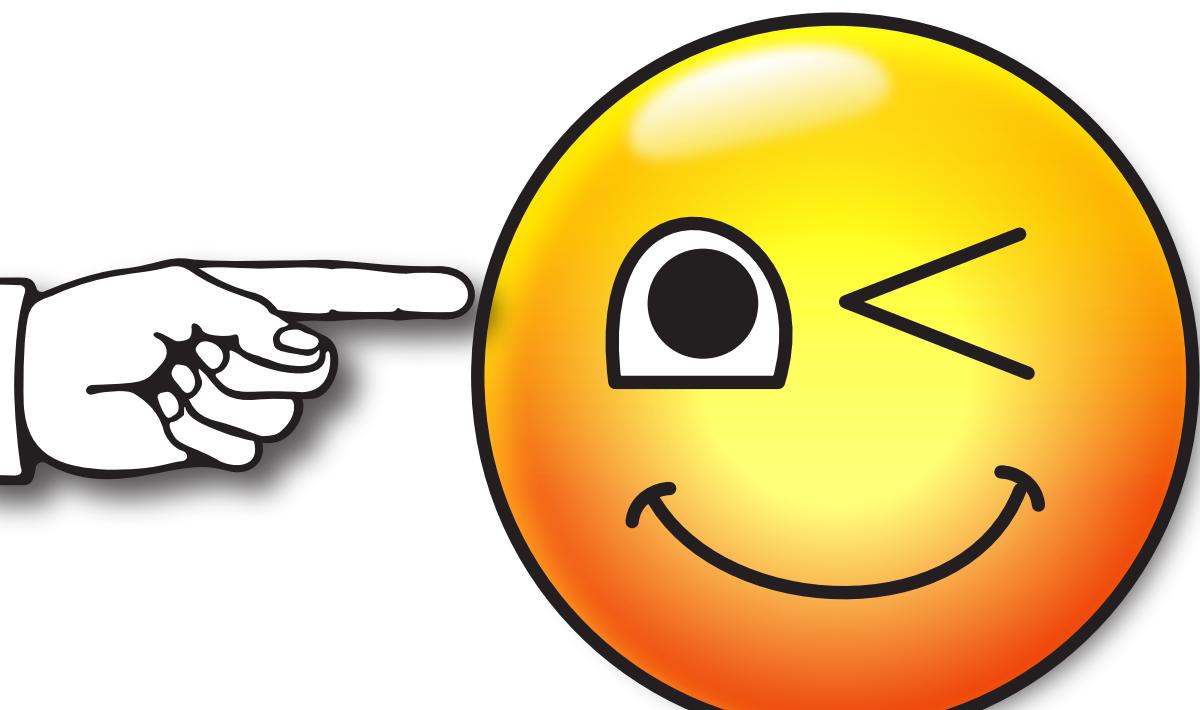


Takagi Kazuyuki

takagi@uec.ac.jp



12 NOVEMBER 2025



メディア情報学実験  
メディア情報検索・認識（音声）  
～単語音声認識～

高木一幸 takagi@uec.ac.jp

2025年／令和07年11月12日版

# 目 次

|                             |            |
|-----------------------------|------------|
| <b>第 1 章 はじめに</b>           | <b>1-1</b> |
| 1.1 自動音声認識とは . . . . .      | 1-1        |
| 1.1.1 音声認識の難しさ . . . . .    | 1-1        |
| 1.1.2 音声認識の歴史 . . . . .     | 1-2        |
| 1.2 単語音声認識とは . . . . .      | 1-3        |
| 1.3 概要 . . . . .            | 1-4        |
| 1.4 必須条件 . . . . .          | 1-5        |
| 1.5 日程 . . . . .            | 1-6        |
| 1.6 レポート . . . . .          | 1-7        |
| 1.6.1 形式・期限 . . . . .       | 1-7        |
| 1.6.2 課題一覧 . . . . .        | 1-7        |
| 1.6.3 穴埋め問題について . . . . .   | 1-7        |
| 1.7 計算機実習の準備 . . . . .      | 1-8        |
| 1.7.1 ファイルのコピー . . . . .    | 1-8        |
| 1.7.2 環境設定の追加 . . . . .     | 1-8        |
| 1.7.3 シェル環境の起動方法 . . . . .  | 1-8        |
| 1.7.4 元の環境に戻す場合 . . . . .   | 1-8        |
| 1.7.5 エディタについての注意 . . . . . | 1-8        |
| <b>第 2 章 音声</b>             | <b>2-1</b> |
| 2.1 音声生成のしくみ . . . . .      | 2-2        |
| 2.1.1 音源—フィルタ理論 . . . . .   | 2-2        |
| 2.1.2 声道モデル・母音の生成 . . . . . | 2-3        |
| 2.1.3 子音の生成 . . . . .       | 2-5        |
| 2.2 日本語の音素 . . . . .        | 2-7        |
| 2.2.1 母音 . . . . .          | 2-7        |
| 2.2.2 子音 . . . . .          | 2-11       |
| 2.2.2.1 破裂音 . . . . .       | 2-11       |
| 2.2.2.2 摩擦音 . . . . .       | 2-12       |
| <b>第 3 章 スペクトル分析</b>        | <b>3-1</b> |
| 3.1 短時間フレーム分析 . . . . .     | 3-1        |
| 3.2 高域強調 . . . . .          | 3-1        |
| 3.3 窓関数 . . . . .           | 3-4        |
| 3.4 離散フーリエ変換 . . . . .      | 3-5        |

---

|                                   |            |
|-----------------------------------|------------|
| 3.5 メル周波数                         | 3-5        |
| 3.6 フィルタバンク分析                     | 3-5        |
| 3.7 メル周波数ケプストラム係数 (MFCC)          | 3-7        |
| 3.8 特徴抽出はデータ圧縮                    | 3-7        |
| 3.9 スペクトル分析例                      | 3-9        |
| 3.10 実習                           | 3-10       |
| 3.10.1 フィルタバンク分析                  | 3-10       |
| 3.10.2 MFCC 分析                    | 3-11       |
| 3.10.3 音声収録の練習                    | 3-13       |
| 3.10.4 レポート (第 1 週)               | 3-25       |
| <b>第 4 章 統計的音声認識</b>              | <b>4-1</b> |
| 4.1 枠組み                           | 4-1        |
| 4.2 基本原理                          | 4-2        |
| <b>第 5 章 隠れマルコフモデル (HMM)</b>      | <b>5-1</b> |
| 5.1 基本 HMM の定式化                   | 5-1        |
| 5.2 HMM の 3 つの問題                  | 5-2        |
| 5.3 確率計算法                         | 5-3        |
| 5.3.1 前向きパス (Forward) アルゴリズム      | 5-4        |
| 5.3.2 後ろ向きパス (Backward) アルゴリズム    | 5-6        |
| 5.3.3 最適状態系列推定 (Viterbi アルゴリズム)   | 5-6        |
| 5.3.4 パラメータ推定 (Baum-Welch アルゴリズム) | 5-8        |
| 5.4 音声認識のための HMM                  | 5-10       |
| 5.4.1 一方向性 HMM                    | 5-10       |
| 5.4.2 連続 HMM                      | 5-11       |
| 5.4.3 スケーリング (scaling)            | 5-14       |
| 5.5 学習から認識までの手順                   | 5-14       |
| 5.6 実習                            | 5-15       |
| 5.6.1 Forward アルゴリズム              | 5-16       |
| 5.6.2 Backward アルゴリズム             | 5-17       |
| 5.6.3 Baum-Welch アルゴリズム           | 5-18       |
| 5.6.3.1 HMM の学習実験 [9]             | 5-18       |
| 5.6.3.2 HMM による認識実験 [9]           | 5-20       |
| 5.6.3.3 実習手順                      | 5-21       |
| 5.6.4 多次元ガウス確率密度関数                | 5-25       |
| 5.6.5 レポート (第 2 週)                | 5-26       |
| <b>第 6 章 単語音声認識</b>               | <b>6-1</b> |
| 6.1 はじめに                          | 6-1        |
| 6.2 単語 HMM                        | 6-2        |
| 6.3 単語音声認識実験の概要                   | 6-3        |

---

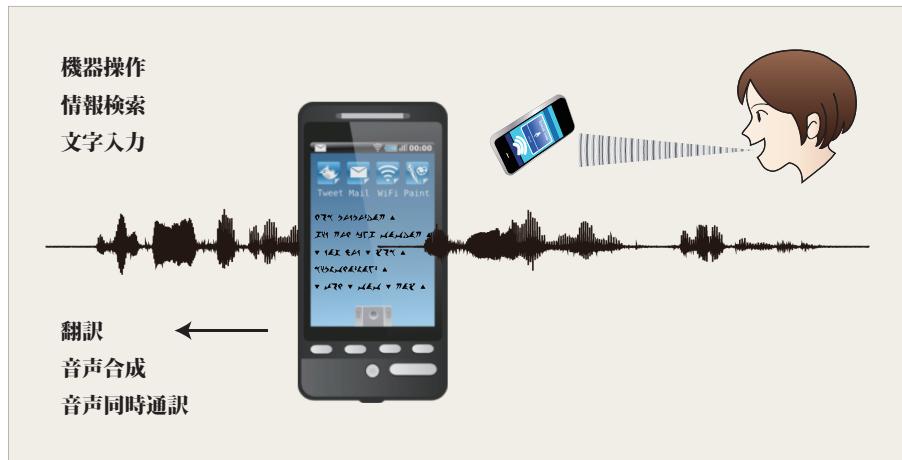
|                        |                             |      |
|------------------------|-----------------------------|------|
| 6.3.1                  | 基本構成 . . . . .              | 6-3  |
| 6.3.2                  | 音声区間検出 . . . . .            | 6-5  |
| 6.4                    | 単語 HMM の学習 . . . . .        | 6-7  |
| 6.5                    | 実習 . . . . .                | 6-8  |
| 6.5.1                  | 認識用設定ファイルの作成 . . . . .      | 6-9  |
| 6.5.2                  | 単語音声データの作成 . . . . .        | 6-10 |
| 6.5.3                  | スペクトル分析 . . . . .           | 6-23 |
| 6.5.4                  | 単語 HMM の学習 . . . . .        | 6-23 |
| 6.5.4.0.1              | 学習の失敗 . . . . .             | 6-26 |
| 6.5.5                  | 学習の検証 . . . . .             | 6-27 |
| 6.5.6                  | On-The-Fly 単語音声認識 . . . . . | 6-28 |
| 6.5.6.1                | 音声区間検出の閾値の決定 . . . . .      | 6-28 |
| 6.5.6.2                | 単語音声認識の実行 . . . . .         | 6-29 |
| 6.5.6.3                | 認識結果の検討 . . . . .           | 6-30 |
| 6.5.7                  | プログラム構成 . . . . .           | 6-32 |
| 6.5.8                  | レポート（第 3 週） . . . . .       | 6-34 |
| 第 7 章 おわりに             |                             | 7-1  |
| 付 錄 A 数字単語音声のスペクトル分析実例 |                             | A-1  |
| 付 錄 B 実習用ファイル          |                             | B-1  |

# 第1章 はじめに

## 1.1 自動音声認識とは

自動音声認識 (automatic speech recognition: ASR) とは、音声波に含まれる言語情報を機械（コンピュータ）で自動的に抽出することです。狭い意味では、音声の内容を自動的に文字に書き取ることをいいます。キーボードを打つ代わりに、コンピュータに話した内容がディスプレイに表示されるタイプライタのようなものといつていいくらいでしょう。広い意味では、誰が話しているかなどの個人性情報といったものの抽出を行う話者認識などの技術を含みます。

音声認識には、あらかじめコンピュータに登録した単語やコマンドを認識する単語認識（音声コマンド認識とも呼ばれます）から、数千以上の単語を扱い単語ごとに区切らず普通に話された音声を認識する大語彙連続音声認識（ディクテーションとも呼ばれます）までさまざまなものがあります。



### 1.1.1 音声認識の難しさ

音声認識を構成する中心的な技術要素はパターン認識 (pattern recognition) です。音声認識はパターン認識の中でも最も難しいクラスの問題と位置付けられています [38]。パターン認識の問題の難しさは、入力及び出力それぞれの複雑さと、その対応関係で定義されます。

パターン認識の問題における最も簡単なクラスは、「1 入力 1 出力 パターン認識問題」で、枠内に書かれた文字を認識するような問題が典型的な例です。これに対して、音声は時系列信号なので、入力は複数（一定の時間間隔で区切られた特徴量の系列）であり、それらを単語の系列として認識することになるので出力も複数です。さらに、入力の系列長と出力の系列長の間に明確な対応関係がありま

せん。早口で話すのとゆっくり話すのとでは、単位時間の単語数が異なります。従って、音声認識はパターン認識の中で最も難しい問題である「系列長の対応関係が無い複数入力複数出力のパターン認識問題」なのです。

### 1.1.2 音声認識の歴史

音声認識の研究は80年以上前から行われていて、さまざまな技術が研究開発されてきました [1–3, 5–8, 11, 12, 14–16, 18–27, 29–38]。これまでの研究開発の世代を簡単に紹介します。以下は文献 [24, 35] で解説されているものです。

1950年から1960年代の第1世代においては、音声信号の物理的特徴に関する知見に基づいて、アナログフィルタバンクと論理回路を用いた経験則的な処理が用いられました。

1960年から1980年代の第2世代では、音声認識に有効な音響特微量と動的パターンマッチング手法に関する研究が進み、あらかじめ蓄えておいたテンプレートとDPマッチングに代表される時間軸整合マッチングが行われました。

1980から1990年代の第3世代では、DPマッチングを拡張した形で隠れマルコフモデル (Hidden Markov Model: HMM) と、HMMの各状態の音響特微量のパターンをモデル化する混合正規分布 (Gaussian Mixture Model: GMM) が導入されました。このような音声モデルをGMM-HMMと呼びます。N-gram言語モデルが導入され、その後の実用的音声認識システムの基盤技術となりました。

1990年から2000年代は、誤り率最小化に基づく識別的モデル、モデル適応、話し言葉音声データベースなどによってこの基盤技術の高度化が進められました(第3.5世代)。

2010年代以降は第4世代と呼ばれ、深層ニューラルネットワーク (Deep Neural Network: DNN) によって大幅な性能向上が達成されています。音響モデルについては、GMMによる確率計算をDNNに置き換えたDNN-HMMが、言語モデルについては、再帰型ニューラルネットワーク (Recurrent Neural Network: RNN) をN-gramと併用するモデルが一般的になっています [35]。近年ではRNNを発展させた長・短期記憶 (Long Short-Term Memory: LSTM) のみで直接音声認識結果を出力するEnd-to-End (端から端まで、つまり入力層に音響特微量を入れると出力層に認識結果が出てくる。) が研究されています。最近では、大規模言語モデル (Large Language Model: LLM) の中核を成すTransformerにより、自然言語だけではなく画像、音楽、音声などのマルチモーダル (multimodal、複数の種類のデータを統合して扱う) 処理が可能になりました。この技術の応用として、例えば自動同時音声通訳の実用化が見通せる段階になっています。ChatGPT (Chat Generative Pre-trained Transformer) の最新版であるChatGPT-4o (omni) では人間同士の会話のような応答速度が高い自然な対話が可能となっています。

この実習では第3世代のGMM-HMMを用いた単語音声認識を行います。受講者が認識させたい単語の音声を自分の声で録音して学習用データを作り、GMM-HMMの音声モデルを学習させて、単語認識ができるような仕組みを作ります。この内容にした理由については第7章「おわりに」をご覧ください。

## 1.2 単語音声認識とは

この実験では、もっとも簡単で基本的な音声認識方式である単語音声認識<sup>1</sup>を課題として行います。単語音声を認識することができれば、たとえば、ウェブ資料の検索のためにキーワードを入力したり、カーナビに目的地の地名を入力することができます。

単語音声認識を実現するためには、まず、認識対象となる単語を認識するための音声パターンをコンピュータに登録します。例えば、カーナビに行き先を入力するための単語認識を行う場合は、「東京駅」「調布」「電通大」などの音声パターン  $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_K\}$  を登録しておきます。そして発話された単語をスペクトル分析して得られた音声パターン  $\mathbf{x}$  と登録パターン  $\mathbf{y}_k$  ( $k = 1, 2, \dots, K$ ) との比較を行います。 $d(\cdot, \cdot)$  を適当なスペクトル距離尺度とします。比較の結果、パターン間距離  $d(\mathbf{x}, \mathbf{y}_k)$  が最も小さいもの、すなわち  $\underset{k}{\operatorname{argmin}} d(\mathbf{x}, \mathbf{y}_k)$  を認識結果とします<sup>2</sup>（図1.1）。

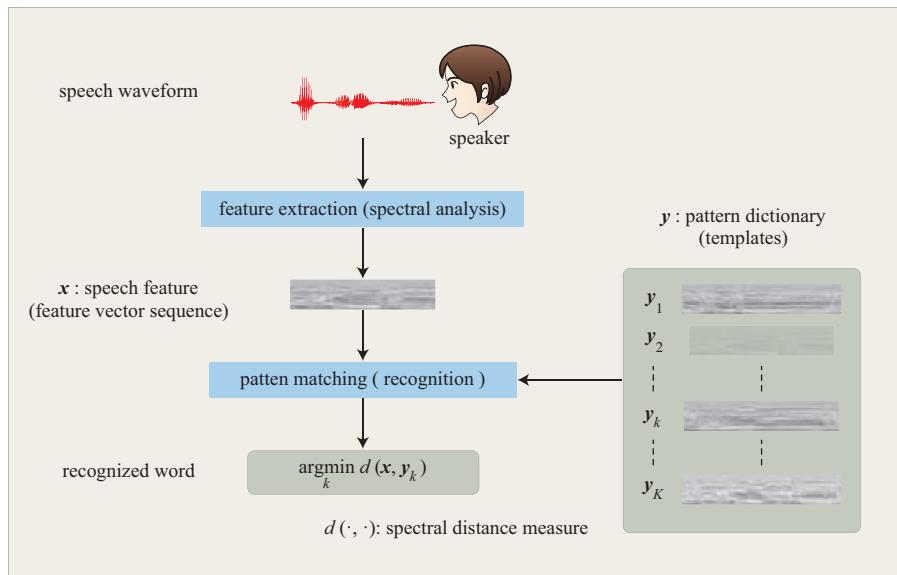


図1.1: 音声のパターン認識（単語音声）。 $\mathbf{x}$ ：音声スペクトル特徴パターン， $\mathbf{y}$ ：パターン辞書， $d(\cdot, \cdot)$ ：スペクトル距離尺度

発話された音声と登録パターンとの距離を計算する場合、単純に比較することはできるでしょうか？異なる話者が発話した「青い」/aoi/という単語の音声波形を比較してみます（図1.2）。話者K（図1.2（a））に比べて、話者Fの発話（図1.2（b））の継続長は1割ほど短いのですが、/o/の継続長は逆に5 ms長くなっています（話者Kは125 ms、話者Fは130ms）。このため、話者Kの発話を単純に時間方向に1割縮めても、話者Fの発話内容と時間的に一致しません。この例のように、一般に2つの発話の長さは異なります。録音した音声を再生するのでない限り、発話の長さが同じになることはありません。さらに、単語に含まれる個々の音の長さは、音の種類によって伸縮のしかたが異なります。このた

<sup>1</sup>音声認識の文脈では単語音声認識を単に単語認識と呼ぶこともあります。

<sup>2</sup>距離ではなく類似度  $s(\cdot, \cdot)$  を用いる場合は、類似度  $s(\mathbf{x}, \mathbf{y}_k)$  が最も大きいもの、すなわち  $\underset{k}{\operatorname{argmax}} s(\mathbf{x}, \mathbf{y}_k)$  を認識結果とします

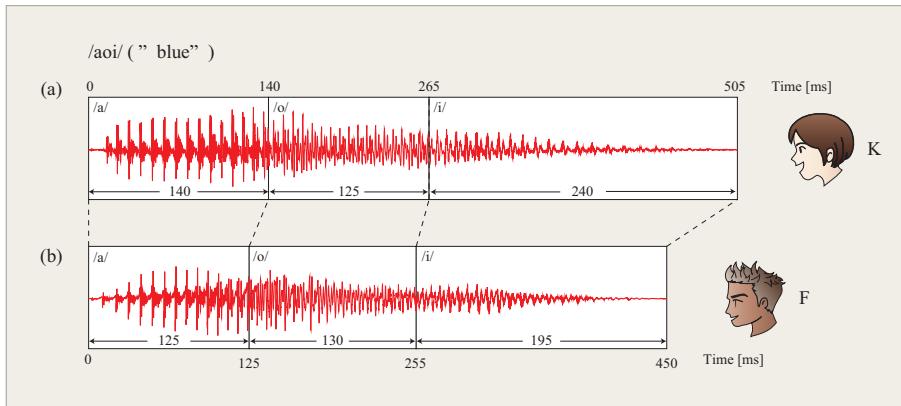


図 1.2: 音声パターンは非線型に伸縮する。「青い」 /aoi/ の話者 K と話者 F の音声波形、音素の位置と継続時間の比較。

め、音声パターンの比較は、時間軸方向に部分的に伸縮させながら行う必要があります。時間軸方向の非線型伸縮を行いながらパターン比較を行う古典的手法には、DP マッチングや隠れマルコフモデルがあります。本実験では隠れマルコフモデル (Hidden Markov Model: HMM) を用います。

### 1.3 概要

音声信号のスペクトル分析法、統計的音声認識の枠組みを理解し、音声認識とはどのような技術であるのかを知り、隠れマルコフモデル (HMM) の原理と算法を身につけ、自分の音声による単語認識を体験することによって、音声認識の実際について学ぶことが本実験テーマの狙いです。

第2章では音声の基本事項、すなわち声帯や舌などの音声を生成する器官、音声生成の仕組み、音声信号のスペクトルの性質、日本語の音素について概説しています。音声のスペクトル分析の結果を見るために必要な基礎知識です。第6章で単語音声データを作成する際の音声区間の始まりと終りを定めるときに役立つ知識です。

第3章では音声情報処理に必要な音声スペクトル分析について学びます。スペクトル分析では、音声に含まれているさまざまな高さの音の成分の強弱を抽出します。音声の振幅波形をフーリエ変換して得たパワースペクトルを人間の聴覚特性を模擬したフィルタバンクに入力し、その出力に対して離散コサイン変換を行い、メル周波数ケプストラム係数 (MFCC) を得ます。この MFCC を音声の特徴量として用いて単語認識を行います。この章では、音声振幅波形から MFCC を得る計算手順を学び、C 言語でプログラミングを行い、音声の分析を行います。

第4章では、この実験の基礎となる統計的音声認識について説明します。これは現在一般に用いられている音声認識技術に共通する基礎原理です。

第5章では、HMM の原理を学びます。HMM をパターン認識器として用いる場合、入力パターンを生成する確率を効率的に計算する算法が必要です。そのための Forward アルゴリズム、Backward アルゴリズムの C 言語のプログラムを作り、コンピュータ上で実行してみます。また、HMM パラメータの学習に用い

られる Baum-Welch アルゴリズムを C 言語でプログラミングし、簡単なパターンの実例を用いて HMM を学習し、その HMM を使ってパターン認識を行ってみます。さらに、単語音声のモデル化に必要な多次元ガウス確率密度関数のプログラミングをします。

第 6 章では、単語認識を行うために必要な単語 HMM と、それを用いて単語認識を行う方法について学びます。受講者が自らの音声を使って単語音声認識を行います。自分の声を使って単語の音声データを収集し、そのスペクトルパターンから単語 HMM を学習します。学習した HMM の認識性能を測定した後、最後にマイク入力の音声を On-The-Fly 認識（入力された音声をその場で直ちに認識）する実験を行います。

## 1.4 必須条件

C 言語でプログラムを書き、コンパイルして実験用のプログラムを製作します。そのプログラムを用いて複数のデータに対する繰り返し計算を行うために、シェルプログラミングの機能を使います。そのため、この実験では、受講者が C 言語プログラミングの基本を理解し、実習した経験があること、Linux の CUI（文字ユーザインタフェース）すなわちキーボードからコマンドを入力して計算機を操作することに慣れています。IED 教室の端末は Ubuntu で起動してください。実験を行うときは、まずデスクトップ上部右端のアイコンをクリックして「端末」（図 1.3）を起動してください。端末は複数起動しておいた方が作業が効率的です。ファイル操作、音声編集ソフトウェアやテキストエディタの起動、音声認識の実施は全てこの端末にコマンド入力することによって行います。

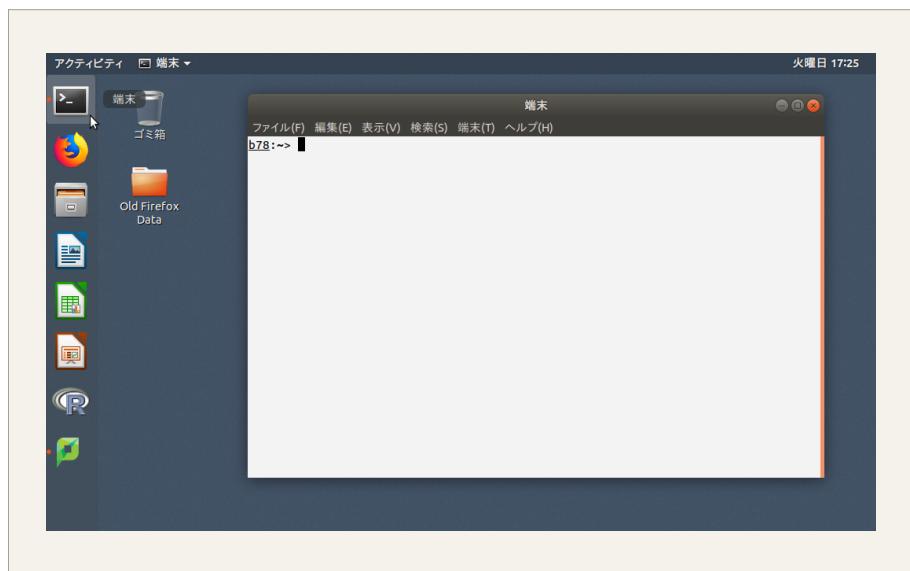


図 1.3: 実験の手順は端末へのコマンド入力によって進める。端末はデスクトップ上のアイコンをクリックして起動する。

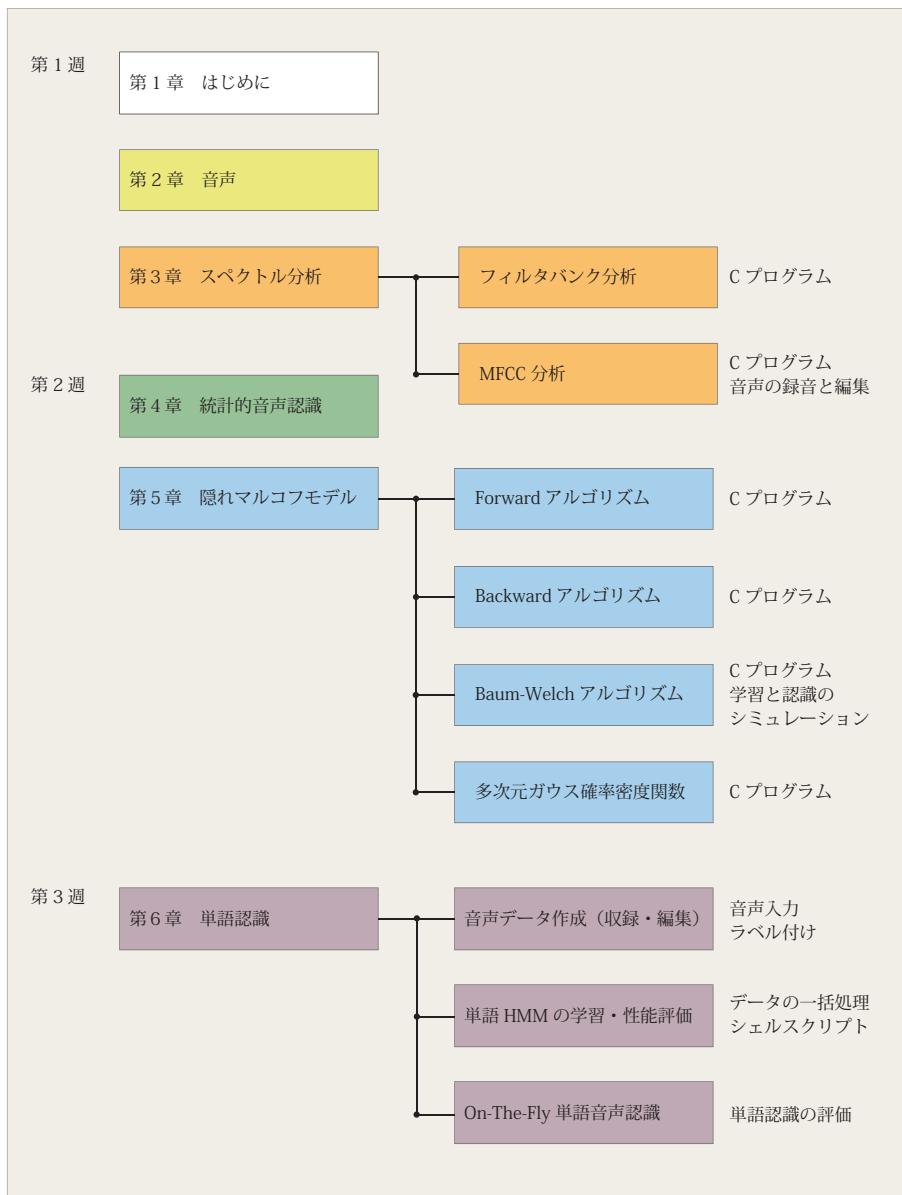


図 1.4: 実験書の構成と日程

## 1.5 日程

第1週では、音声信号のスペクトル分析について学び、C言語でスペクトル分析のプログラムを作成します。そして、音声入力の練習を兼ねて自分の音声を録音し、スペクトル分析をしてみます（第2章、第3章）。

第2週では、HMMの基本算法であるForwardアルゴリズムとBackwardアルゴリズムのプログラムを作ります。また、音声のパターンをコンピュータに学習させるためのHMM算法であるBaum-Welchアルゴリズムの実習、及び多次元ガウス確率密度関数のプログラミングを行います（第4章、第5章）。

第3週では、自分の音声を録音し、その音声を学習データとして用いて単語HMMを学習し、それを用いて音声認識の実験を行います。まず、あらかじめ録音しておいた自分の音声を用いて認識率の評価をし、つぎに、マイクから入力した音声を直接認識することを試します（第6章）。

## 1.6 レポート

### 1.6.1 形式・期限

実験日の翌週の水曜日の 23:59:59 までに, `asr@takagi.inf.uec.ac.jp` に提出すること. 毎回提出してください. レポートの形式は Portable Document Format (PDF) に限ります. PDF ファイルに対し, セキュイティ設定 (ファイルロック, パスワード等) を行わないこと. PDF ファイル名は半角英数字で “学籍番号\_通し番号\_姓” とすること (例: `1234567_1_Takagi.pdf`). メールの Subject は PDF ファイルと同一とすること.

### 1.6.2 課題一覧

#### 第1日 スペクトル分析：第 3.10 節

- フィルタバンク分析：第 3.10.1 節
- メル周波数ケプストラム分析：第 3.10.2 節
- 自分の声の録音, 編集, 分析：第 3.10.3 節

#### 第2日 HMM 算法のプログラミングとシミュレーション：第 5.6 節

- 前向きパスアルゴリズム：第 5.6.1 節
- 後ろ向きパスアルゴリズム：第 5.6.2 節
- Baum-Welch アルゴリズム：第 5.6.3 節
- 多次元ガウス確率密度関数：第 5.6.4 節

#### 第3日 単語音声認識：第 6.5 節

- 単語認識タスクの設計：第 6.5.1 節
- 単語 HMM の学習：第 6.5.4 節
- 学習の検証：第 6.5.5 節
- On-The-Fly 単語認識：第 6.5.6 節

### 1.6.3 穴埋め問題について

課題の中には C 言語のソースコードの一部が削除されていて, その部分を補つて所望の処理を行うプログラムを完成させるものがあります. ソースコード中の穴埋め部分は `/* fill in blank */` で示しています. 穴埋め部分は 1 行で示されていますが, 補うべきコードは, 代入文の右辺である場合や, ループや条件判断などの制御を伴う複数行のものであったりします. 十分ご注意ください.

## 1.7 計算機実習の準備

### 1.7.1 ファイルのコピー

自分のホームディレクトリの直下に `asr` というサブディレクトリを作つて、実験に必要なファイルを用意します。

```
[~] % cd  
[~] % cp -pR ~takagi/asr ./
```

と入力すると、カレントディレクトリ直下に `asr` というサブディレクトリができます。念のため、上記を実行する前に同じ名前のディレクトリが無いことを確認して下さい。コピーされるファイルの内訳については、付録Bをご覧ください。

### 1.7.2 環境設定の追加

音声認識実験に必要なコマンドのパスを`~/.cshrc`に追加したり、音声編集プログラム (WaveSurfer) の設定を行うため、初回のみ以下のコマンドを実行してください。

```
[~] % tcsh -f ~/asr/addpath
```

この操作により、シェル環境設定ファイル`$home/.cshrc`が`$home/.cshrc_backup_asr`に退避保存され、この実験のためのパス設定などが`$home/.cshrc`に追加されます。

### 1.7.3 シェル環境の起動方法

初回設定を終えた後は、ターミナルを開いたときに以下のコマンドを実行するだけで、設定済みのシェル環境 (tcsh) が使えるようになります。

```
[~] % tcsh
```

これにより、追加されたコマンドパスやエイリアスが自動的に有効化されます。本実験ではこのシェル環境での実行を前提としています。

### 1.7.4 元の環境に戻す場合

メディア情報検索・認識実験（音声）のすべての課題が完全に終了するなどし、元のシェル環境の設定に戻したい場合は、退避保存されている以下のファイルを復元してください。

```
[~] % cp ~/.cshrc_backup_asr ~/.cshrc
```

### 1.7.5 エディタについての注意

Ubuntu 付属の「テキストエディタ」は、本実験の単語認識実験に用いる設定ファイルの作成に適していないので、これ以外の開発環境やテキストエディタを使って下さい。

## 第2章 音声

人はその言語能力によって文化を築き、社会、芸術、科学技術を発展させてきました。このことにより、人は他の生物とは一線を画する存在となっています。犬、鳥、鯨なども声を使ってコミュニケーションしますが、意味の区別がある声の種類は5~30程度だといわれています。また、人のように声を組み合わせて新しい意味を作ることはできないといわれています。

人の音声言語は、音素が組み合わされて形態素（語）を構成し、形態素（語）が組み合わされて文が構成されるという二重構造になっています（図2.1）。このことによって、比較的少数の音素<sup>1</sup>を任意の個数並べることにより数千~十数万の語を構成することができ、それらの語を任意の個数並べることによって、事实上無限の文の生成が可能です。これを二重文節といい、他の生物には無い人の言語だけの特徴だといわれています。

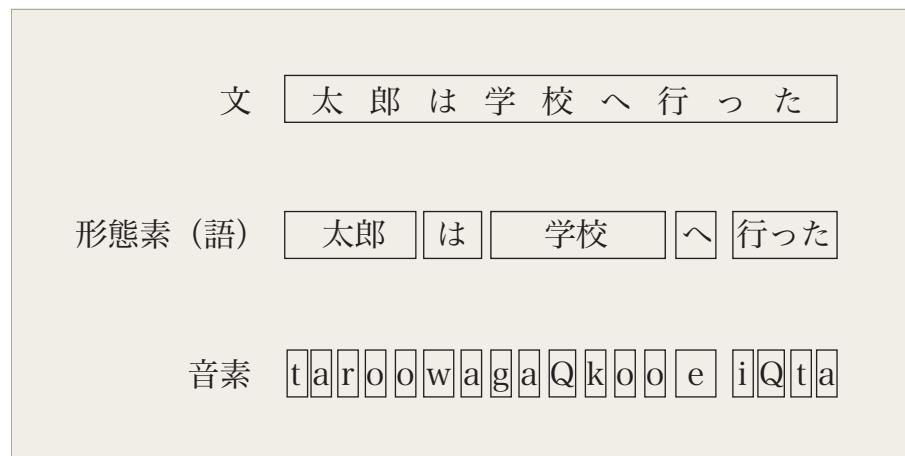


図 2.1: 言語の二重分節。文は形態素（語）に分解され、形態素（語）は音素に分解される。

19世紀以降の科学技術の発展により、言語の実体としての音声の研究[1-3]は急速に進展してきました。それは、電気通信技術の発展と、その影響による音響機器、電子回路、放射線機器、およびコンピュータの発達の恩恵を受け、これらのツールを用いて音声信号を分析したり合成することが可能となったことによります。

音声研究の歴史において、研究対象は上で述べた二重文節の小さな単位から大きな単位へと発展してきました。音素の性質は発声器官の形とその運動によって決まるので、X線撮影で撮影した発声器官の形を測定する基礎研究が古くから行われていました。発声器官の形やその変化の仕方が分かれば、音声の周波数特性やその変化の仕方を計算することができます。この章では、発声器官について説

<sup>1</sup>最も少ない言語で11、最も多い言語で141、多くの言語は20から40種類[10].

明し、音声生成の音響工学的な解釈、日本語の主な母音、子音の性質について解説します。

## 2.1 音声生成のしくみ

我々が話す時には、肺からの空気を制御して少しづつ吐き出し、喉頭と声道（喉、口、歯、口唇、一部の音では鼻を併用）を使って発声します（図2.2）。声道にさまざまな狭めを作ることによって、さまざまな音色の音声が生成されます。最も重要な音源は喉頭（喉仏）であり、ここには声帯があります。適切な張力が

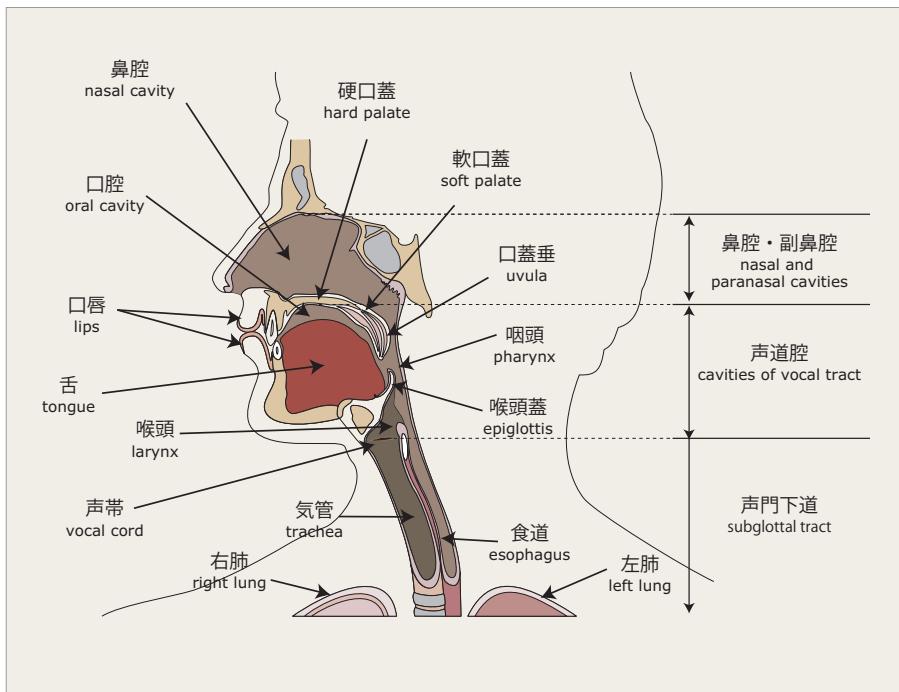


図2.2: 発声器官概略図（頭部正中矢状断面）。音声器官は肺、喉頭、咽頭、口、鼻からなります。声道は咽頭、口からなり、その形は母音の種類によって異なります。声道の形は口唇、顎、舌、喉頭の位置や形によって変化します。

掛けた声帯の間を呼気が通ると、声帯は細かく振動し、断続的な空気流を声道に送り込みます（図2.3）。この声帯音源の響き方は声道の形によって変わります。声道は、ある周波数の振動を強め、一方で別の周波数の振動を弱めるようなフィルタとして作用します。全ての音声の音が声帯音源を含んでいる訳ではありません。声帯音源を含んでいる音声を有声音、そうでない音声を無声音といいます。

### 2.1.1 音源—フィルタ理論

音声の生成において、肺は動力源として、振動する声帯は発振器として、声道は共鳴器として働きます。音源—フィルタ理論によれば、音源は肺からの空気流によって振動する声帯によって発生する声帯音源です。音源の音響スペクトルは周波数が高くなるにしたがって振幅が減少します（S）。フィルタに相当するのは声道であり、音源の周波数特性に変化を与えます。その結果フィルタ関数が得ら

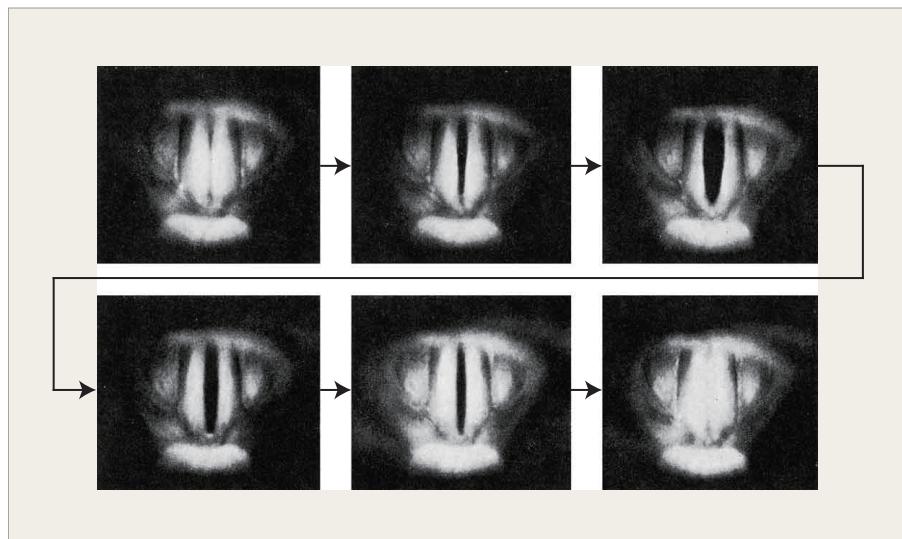


図 2.3: 声帯の振動の様子 (1 周期, 約 8 ms) [3]

れます (F). 音源にフィルタが適用されると、母音の出力スペクトル (O) に共振ピークが生じます (図 2.4).

### 2.1.2 声道モデル・母音の生成

音声を生成しているときの発声器官の形を正確に知ることは、音声の工学的研究が始まった当初から重要な問題でした。発声器官の形状が分かれば、音響理論によって共鳴の特性すなわち声道フィルタの特性を計算することができるからです。X 線を用いて撮影したり、舌や口蓋に電極を貼付けたりして声道の形を測定することが行われてきましたが、最近は磁気共鳴画像診断装置 (Magnetic Resonance Imaging: MRI) が用いられています。図 2.5 は話者が MRI に仰向けの姿勢で入り、日本語の 5 母音を発声した時の正中矢状面の画像です<sup>2</sup>。母音によって、口の開き具合と舌の形 (位置) に特徴があるのがわかります。例えば、/a/ は/i/ に比べて口を大きく開き、舌を奥のやや低い位置に構えています。舌が最も盛り上がっている (したがって声道が最も狭くなっている) 位置が口腔の前側か後側かに注目すると、前側であるのが/i/ と/e/、後側であるのが/a/、/u/、/o/ です。前者は前舌母音 (front vowel)，後者は後舌母音 (back vowel) と呼ばれています。

X 線写真や MRI 画像から得られた発声器官の形状から声道断面積を推定し、それを用いて声道のフィルタ特性を計算することができます。英語の母音、"feet" の /i/、"father" の /a/、"boot" の /u/ の声道の概形が図 2.6 の左側に描かれています。図の中央は、声道を少数の円筒を接続したモデルとして表したモデル、右側は対応する音響スペクトルです。聴覚実験により、人間は音響スペクトルの概形 (特に共振周波数) によって、母音の種類を識別していることが分かっています。

<sup>2</sup>転載厳禁 この MRI データは、ATR 人間情報科学研究所が独立行政法人情報通信研究機構からの研究委託「人間情報コミュニケーションの研究開発」に基づいて収録し、公表した『ATR 母音発話 MRI データ』の一部です。本データの使用および成果の発表は、株式会社 ATR-Promotions との使用許諾契約に基づいております。

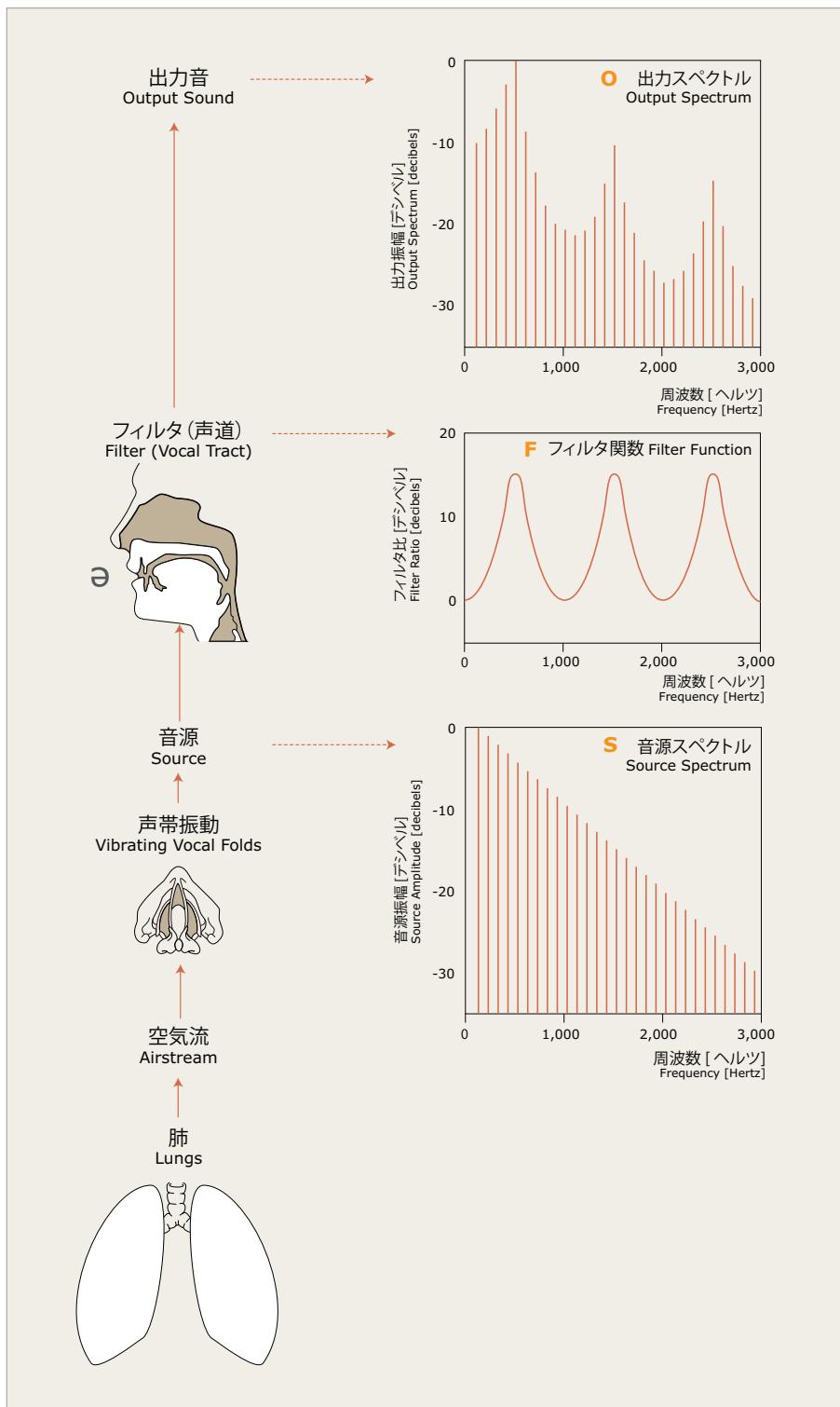


図 2.4: 音源—フィルタモデル. イラストは [4] 掲載のものに基づく.

ます。声帯の共振周波数は母音の認識にとても重要なことで、特にフォルマント (formant) と呼ばれています。

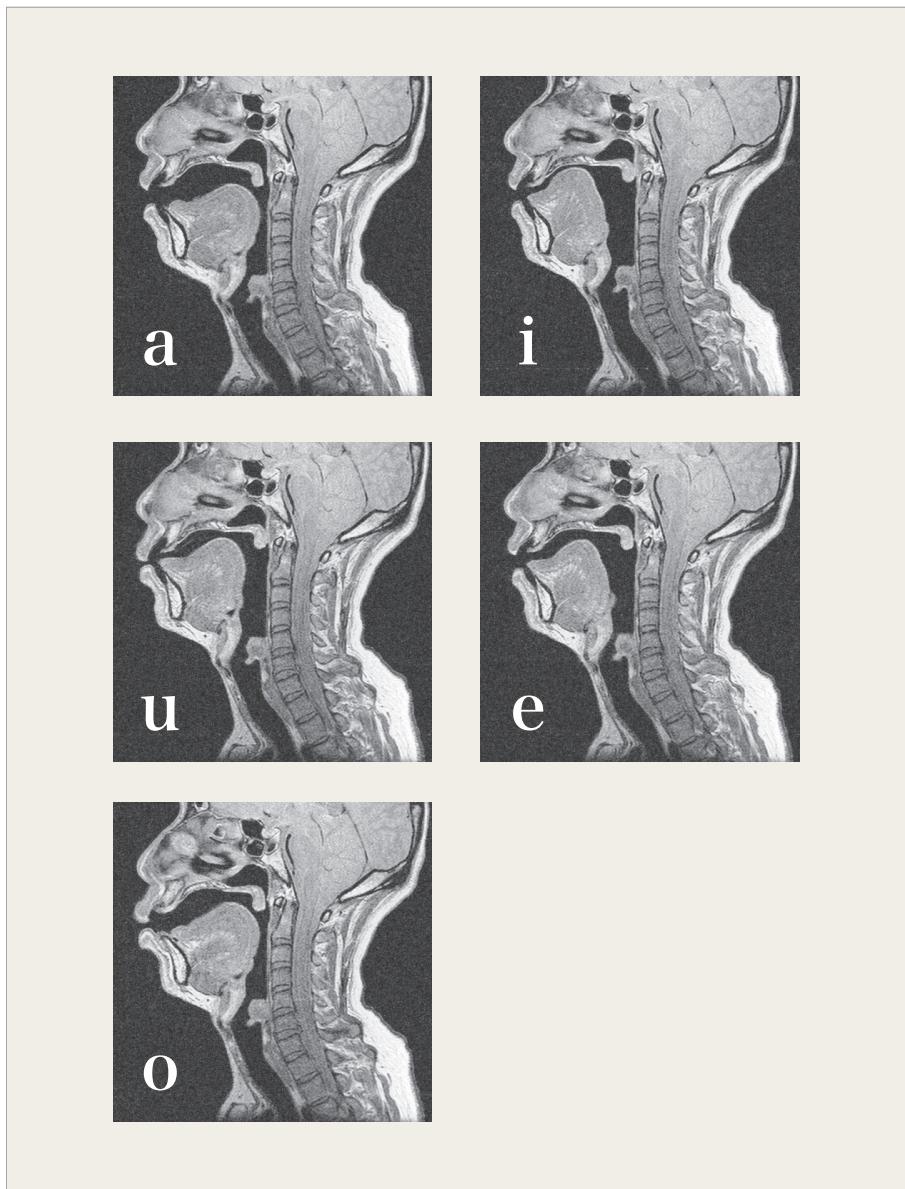


図 2.5: 日本語 5 母音発声時の頭部 MRI 画像 (『ATR 母音発話 MRI データ』) 転  
載厳禁

### 2.1.3 子音の生成

子音は一般的に母音と比べて声道の狭めが強いのが特徴です。声道に強い狭めの位置では呼気による圧力が高まって乱流が生じ、この位置が音源となります。声道に一時的な閉鎖が生じた後の急激な解放によってその部分が音源となる場合もあります。音源の生成のされ方（狭め、閉鎖）や位置（口唇側、声帯側）によって音源の特性はさまざまです。声帯の振動が伴う場合もあります。器官の瞬間的な動きによって生成されるものがあります。

狭めによる乱流が音源となる子音は摩擦子音 (fricative) と呼ばれ、/f/, /ʃ/, /v/などがあります。一時的な声道の閉鎖の後の解放によって生じる子音を破裂子音 (plosive) あるいは閉鎖子音 (stop) といい、代表的なものとして/p/, /t/, /k/があります。図 2.7 には、閉鎖音/p/, /t/, /k/の調音位置（唇、歯、軟口蓋）を示しています。

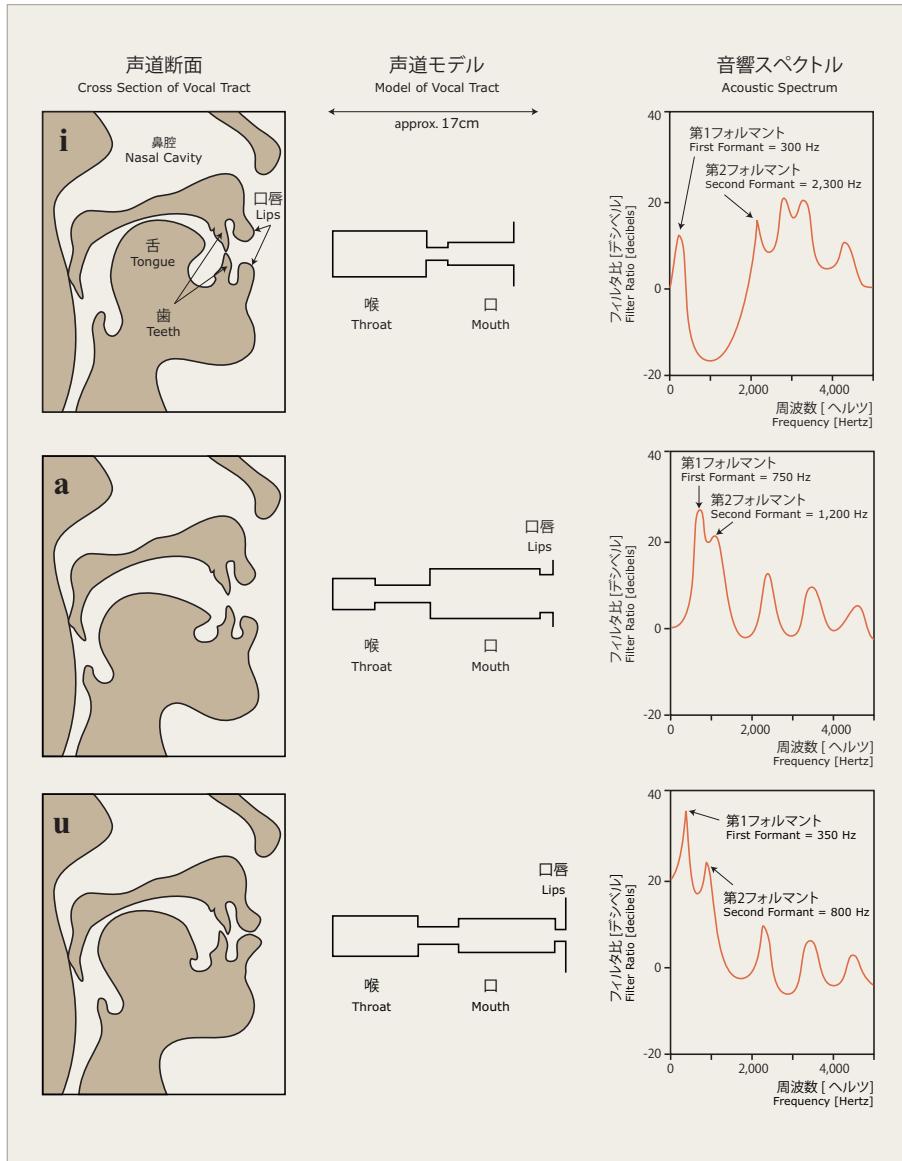


図 2.6: 声道の円筒モデル。イラストは [4] 掲載のものに基づく。

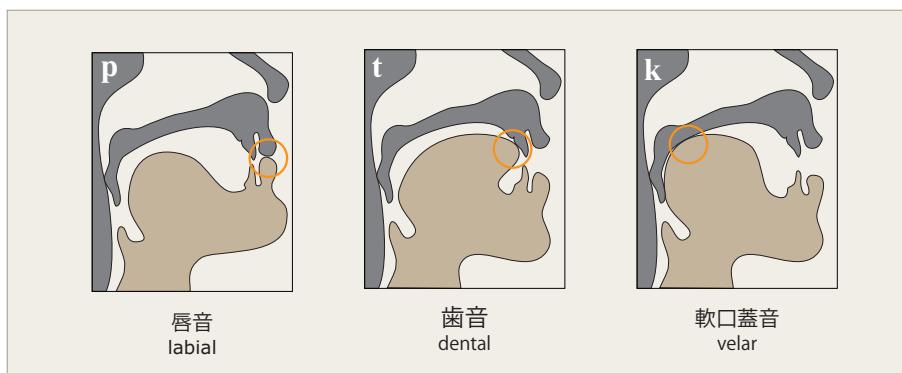


図 2.7: 破裂(閉鎖)子音の調音位置。その調音位置により、/p/は口唇音 (labial), /t/は歯茎 (dental), /k/は軟口蓋音 (velar) と呼ばれます。イラストは [4] 掲載のものに基づく。

## 2.2 日本語の音素

日本語をローマ字で書き表すと、例えば、赤 /aka/、秋 /aki/ のようになります。この 2 つの単語は、ローマ字の第 3 文字の /a/ と /i/ の違いによって区別されます。このように、ある言語において単語の意味の違いに関わる音声の最小単位を音素 (phoneme) といいます。音素は大きく母音 (vowel)、子音 (consonant) に分けることができます。母音は日本語では /a, i, u, e, o/ の 5 種類、子音には /k, s, t, n, h, m, z, d, b, p, N/ などの音素や、半母音 (semivowel) と呼ばれる /j, w/ などがあります。表 2.8 は、日本語の子音を調音位置と調音方法で分類したものです。

音素は有聲音 (voiced sound) と無聲音 (unvoiced sound) に分類されます。有聲音は発声の際に声帯の振動を伴うもので、母音などがこれに該当します。一方、無聲音には口の中の狭めを空気が通るときの音素 (摩擦音/s, sh/など) や、口がいったん閉じて急に開くときの音素 (破裂音または閉鎖音/p, t, k/など) があります。これらの音素に声帯の振動が伴うと、それぞれ/z, dz/, /b, d, g/ となります。有聲音の中には、音が鼻から出る鼻音 (nasal) (/m, n/など) というものもあります。半母音は子音とそうではない母音の間の中間的な狭めを形成して発声されるものです。

| 調音位置 | 両唇音 |    | 歯音 |    | 歯茎音 |    | 口蓋音 |    | 声門音 |
|------|-----|----|----|----|-----|----|-----|----|-----|
| 調音方法 | 無声  | 有声 | 無声 | 有声 | 無声  | 有声 | 無声  | 有声 | 無声  |
| 破裂音  | p   | b  |    |    | t   | d  | k   | g  |     |
| 摩擦音  | f   |    | s  | z  | ʃ   | ʒ  |     |    | h   |
| 破擦音  |     |    | ts | dz | tʃ  | dʒ |     |    |     |
| 鼻音   |     | m  |    |    |     | n  |     | ŋ  |     |
| 流音   |     | w  |    |    |     |    |     | j  |     |
| 弾音   |     |    |    |    |     | r  |     |    |     |

図 2.8: 日本語子音の調音位置と調音方法による分類。声道の最も狭い部分を調音位置といいます。”無声”は声帯の振動を伴わないもの,”有声”は声帯の振動を伴うものをいいます。

### 2.2.1 母音

音波が円筒のような音響管を通過すると、ある周波数成分が強められ、ある周波数成分は弱められるという共鳴現象が生じ、その共鳴周波数は円筒の形に依存します。声帯から唇までの声道は一種の音響管として機能するので、円筒管の場合と同様に共鳴が生じます。声道の形は舌や唇を使ってある程度自由に変えることができます。そうすることによって共鳴のしかたが変わり、異なる音色の母音ができます。

日本語の 5 母音の音声波形（男性）の例を図 2.9 に示しました。波形は母音によって形が大きく異なっていることが分かるでしょう。母音の波形は、同じような波形が繰り返しています。図 2.9 の /a/ の場合は約 9 ms です。この 1 周期を音声の基本周期（fundamental period），基本周期の逆数を基本周波数（fundamental frequency）といいます。基本周波数は声の高さと関係があります。基本周波数が低ければ低い声になり、高ければ高い声になります。基本周波数は、性別によって異なりますし、個人差もあります。基本周波数の値は、男性の場合およそ 80～200 Hz、女性の場合およそ 150 Hz～400 Hz です。基本周波数はアクセントやイントネーションなどによって影響を受けるので、同じ人の音声でも一定ではありません。

音声スペクトルには音素によって特徴的な山や谷が見られ、山の部分をフォルマント（formant）といい、その代表周波数（山のピークの周波数）をフォルマント周波数といいます。周波数の低い方から第 1 フォルマント、第 2 フォルマント、… と呼び、F1, F2, … と表記します。日本語 5 母音（男声）をスペクトログラム表示した図 2.10 に見える濃い横筋がフォルマントに対応しています。スペクトログラム（Spectrogram）はスペクトルを 3 次元のグラフ（時間、周波数、信号成分の強さ）で表わしたもので、一般的には、横軸が時間、縦軸が周波数で、点の明るさや色がその点の時間-周波数位置での信号成分の強さを表しています。F1 と F2 の位置を矢印で示してあります。2 つのホルマントの位置が、母音によってはっきり異なっていることがわかります。

母音による F1, F2 の周波数の違いは、話者によらず一定の傾向があります。図 2.11 は男性と女性各約 30 名の音声を分析して得られた図です [19]。横軸に F1 の周波数、縦軸に F2 の周波数をとり、各母音の測定結果を分布で表わしています。黒丸（●）は男性、白丸（○）は女性の平均値を表わしており、それを中心に標準偏差が描かれています。男女間の平均的違いは、母音間の位置関係を保ったまま、おおよそ周波数を縮尺した関係にあることがわかります。

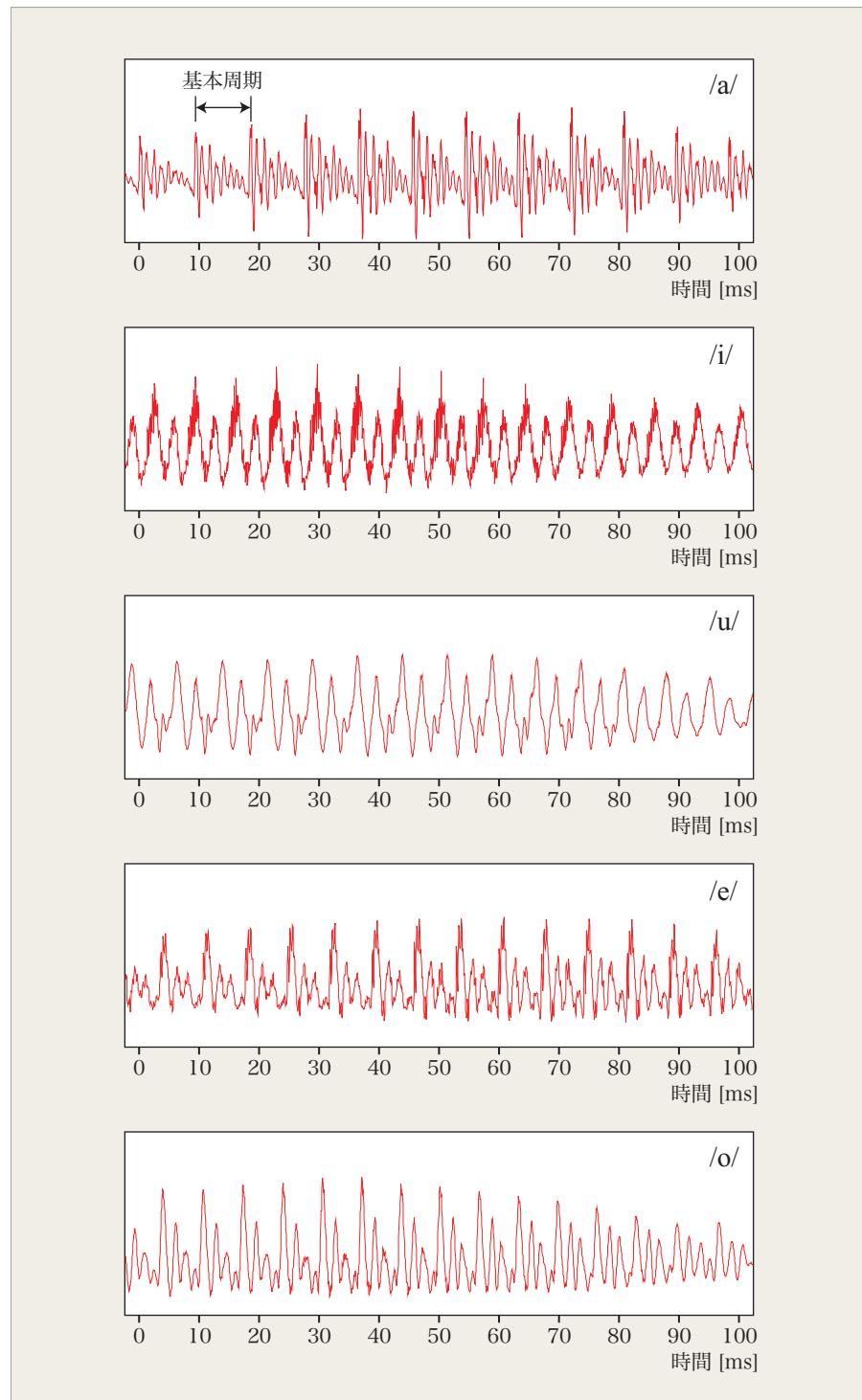


図 2.9: 日本語 5 母音の音声波形（男声）

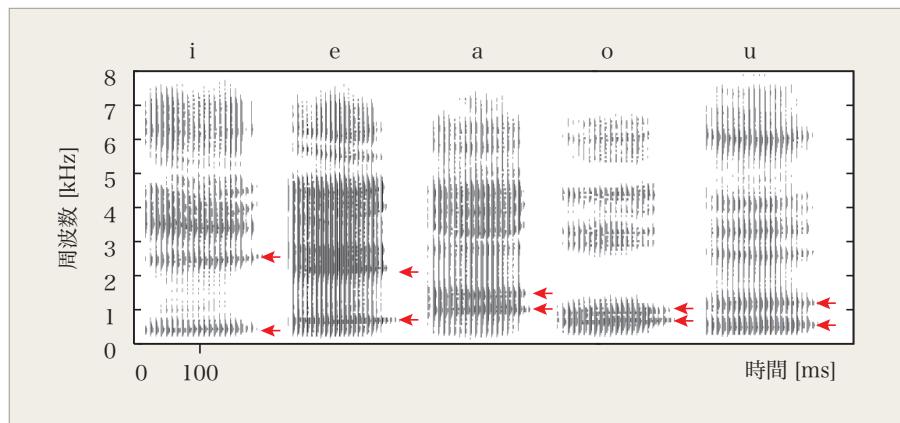


図 2.10: 5 種類の母音（男声）のスペクトログラム表示

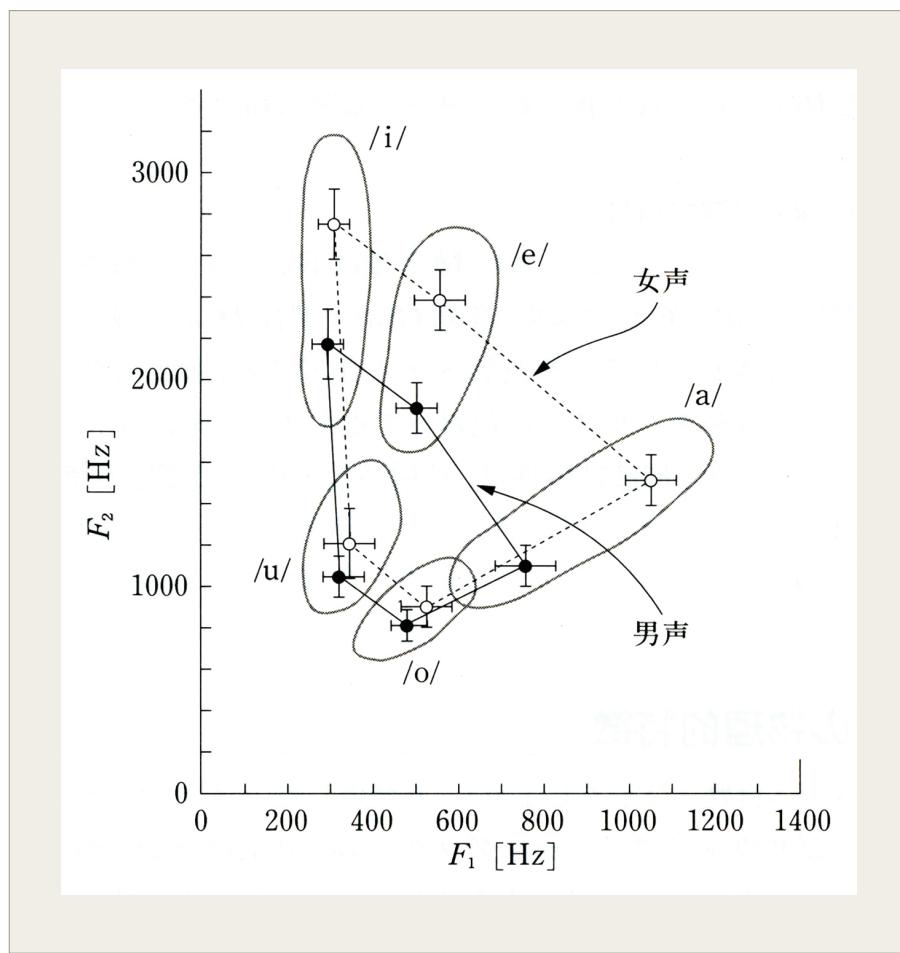


図 2.11: 日本語母音の第 1 第 2 フォルマント周波数の分布 (男性女性各約 30 名) [19]

## 2.2.2 子音

子音は声道に強い狭めあるいは閉鎖を作ることにより生成されます。子音の音源はこの狭めあるいは閉鎖付近に生じます。声道の閉鎖による子音の場合は、閉鎖が解放された時の音が特徴的です。また鼻音のように、声道に鼻腔への分岐が生じる場合があります。このように、子音は、音源の形態や位置、閉鎖の強さ、調音器官の時間的变化の様態、声道の分岐の状態などによって分類されます。

子音は音響理論的扱いが難しく、スペクトルの特徴が複雑です。子音（図 2.8）は母音に比べて種類が多く、非定常な現象によって特徴付けられるものがあります。日本語に限っても子音の性質を網羅的に説明するのは、この実習書の目的の範囲を超えます。

子音の性質についての詳しい説明は関連図書（例えば [3, 11, 20]）に譲ります。この実習で音声認識タスクの実例として説明に用いている数字音声の音響的特徴については付録 A で説明しているので、子音のスペクトルの実例として、課題に取り組む際の参考にしてください。ここでは、他の子音に比べて音響的性質が説明しやすい特徴があり、初心者でも理解しやすい破裂音と摩擦音を取り上げて解説します。

### 2.2.2.1 破裂音

日本語の破裂音には無声破裂音/p/, /t/, /k/と有声破裂音/b/, /d/, /g/があります。無声口蓋破裂音の/k/は、先行母音からの過渡部、閉鎖部、破裂部、気音部、後続母音への過渡部からなります（図 2.12）。波形から分かるように、物理的には閉鎖部は無音ですが、この部分も人間が破裂音を知覚する重要な特徴になっています。

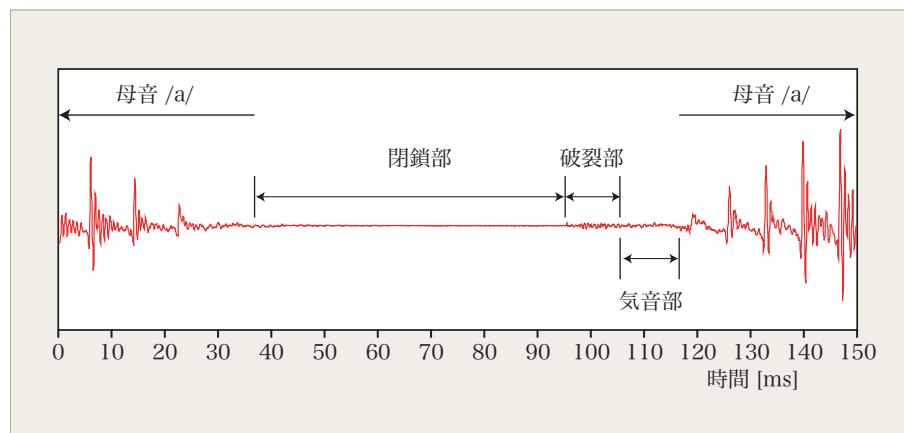


図 2.12: /aka/ の無声破裂音/k/ の部分の音声波形の例

無声破裂音と有声破裂音の違いや破裂の位置（図 2.7）によって、破裂の瞬間から母音に遷移するまでの時間や母音のフォルマントの過渡部の形が異なります。子音の調音位置と母音の種類によって様相が異なるので、母音部も含めた音声波形およびスペクトルの時間的变化の記述が必要です。

### 2.2.2.2 摩擦音

声道の強い狭めの位置に肺からの気流が通過するときに乱流が生じ、それが音源となって摩擦音が生じます。破裂音と異なり、定常的で比較的安定していて、息が続く限り継続することができます。無声歯茎摩擦音 /s/ は周期性のない雑音性の波形を示します（図 2.13（上））。無声子音では声帯が振動していないため、母音のような基本周期に対応した繰り返し波形は現れません。スペクトル特性には主として調音点（狭めの位置）から唇側の共振特性が現れます。3 kHz 以上の周波数帯域にパワーが現れます（図 2.14）。有声歯茎摩擦音である /z/ の場合は、声帯が振動しているため、母音の波形とは違いますが、振幅の小さい周期性の波形が現れています（図 2.13（下））。/s/ と /z/ の違いはスペクトルにも現れています（図 2.14）。/z/ の場合は /s/ に比べて高域成分が少なく、500 Hz 以下の低い周波数成分にパワーが見られます。このような低周波数成分は有声子音の特徴です。

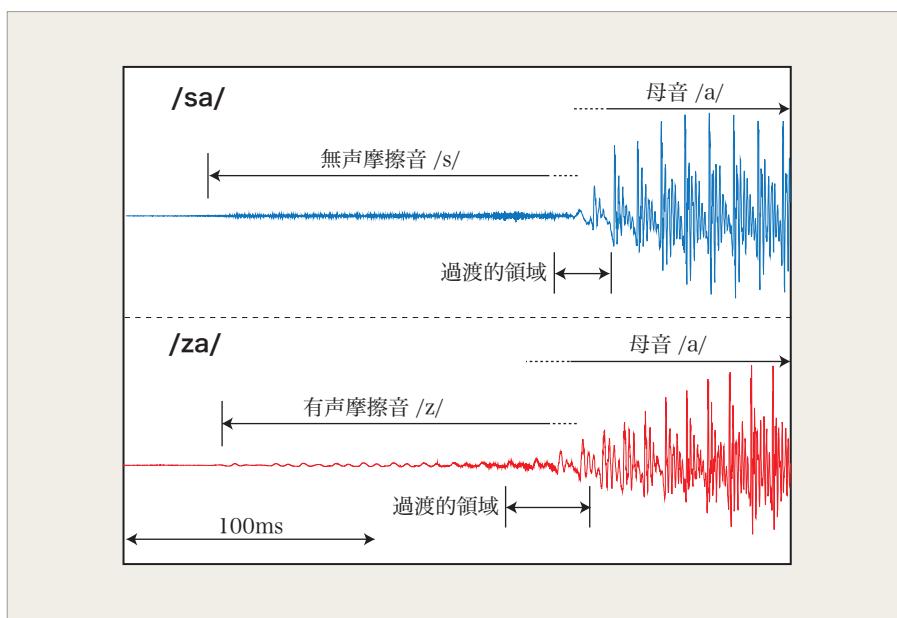


図 2.13: 無声摩擦音 /s/（上）と有声摩擦音 /z/（下）の音声波形の例

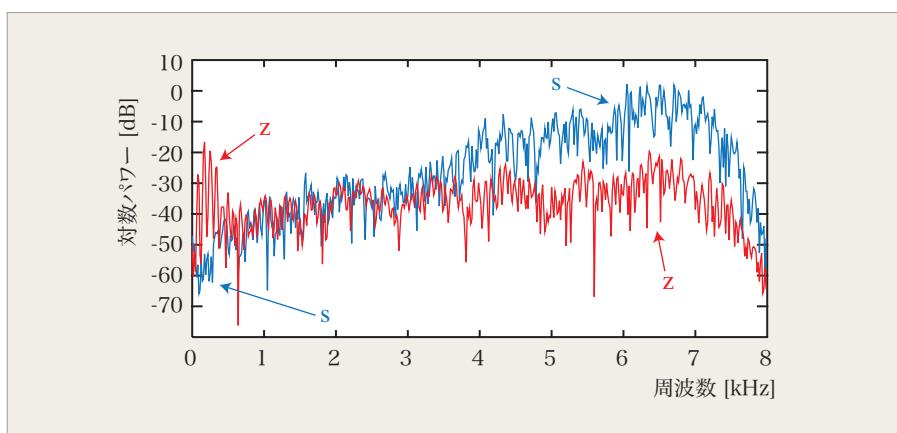


図 2.14: /sa/ と /za/ の対数パワースペクトルの例

## 第3章 スペクトル分析

音声認識のためのスペクトル分析の目的は、音声波形から、たとえば「あ」と「い」をうまく区別するための特徴を抽出することです。「あ」と「い」を発音するとき、我々は舌の位置や口の構えなどを変化させています。「あ」と「い」の区別は、声の高さを変えて話す人が変わっても変わりません。音声認識に必要なのは声の高さや声の個人性といった特徴ではなく、発声器官、とりわけ声道（vocal tract）の形に関する特徴です。

### 3.1 短時間フレーム分析

スペクトル分析では、音声波形から連続する短い時間区間（数十 ms）を切り出し、切り出された信号が定常信号であると仮定して分析を行います。切り出す単位をフレーム（frame）といいます。一定の幅の短時間のフレームを一定の幅で時間軸方向にずらし、フレームの範囲の音声波形データを切り出して分析し、スペクトル特徴量（特徴ベクトル）を求めます（図 3.1)<sup>1</sup>。この処理をフレームが音声波形の終端に達するまで繰り返します。このような分析を短時間フレーム分析（short time frame analysis）といいます。

一度に切り出す音声波形の長さをフレーム幅（frame width）といいます。フレームのずらし幅をフレームシフト（frame shift）といいます。普通、フレームシフトはフレーム幅より小さく設定し、隣り合うフレーム同士が一部重複するようにします。音声認識では、フレーム幅は 20 ms から 40 ms、フレームシフトは 5 ms から 25 ms の範囲のものが使われます。

フレーム分析で得られた一連のスペクトル特徴ベクトルを時間順に並べたものを音声認識の観測系列  $O = o_1 o_2 \cdots o_T$  ( $T$  はフレーム数) として用います。 $D$  を音声波形の継続長、 $W$  をフレーム幅、 $S$  をフレームシフトとすると、この音声を分析するときのフレーム数は  $\lfloor \frac{D-(W-S)}{S} \rfloor$  です<sup>2</sup>。例えば、1 s (=1000 ms) の音声をフレーム幅 32 ms、フレーム間隔 10 ms で分析する場合、フレーム数は  $T = \lfloor \frac{1000-(32-10)}{10} \rfloor = \lfloor 97.8 \rfloor = 97$  となります。

### 3.2 高域強調

人間の音声の周波数成分のパワーは、有聲音の場合  $-6 \text{ db/oct}$ 、つまり周波数が 2 倍になるごとに 6 dB 下がる性質を持っています。これは次のような理由によります。音声を生成する過程は、(1) 音源の生成（有聲音源・無聲音源）、(2) 声

<sup>1</sup>各フレームの分析は、隣接フレームとの関係を考慮せず、個別に行ないます。

<sup>2</sup>  $\lfloor \cdot \rfloor$  : 床関数。 $\lfloor x \rfloor$  は実数  $x$  に対して  $x$  以下の最大の整数。

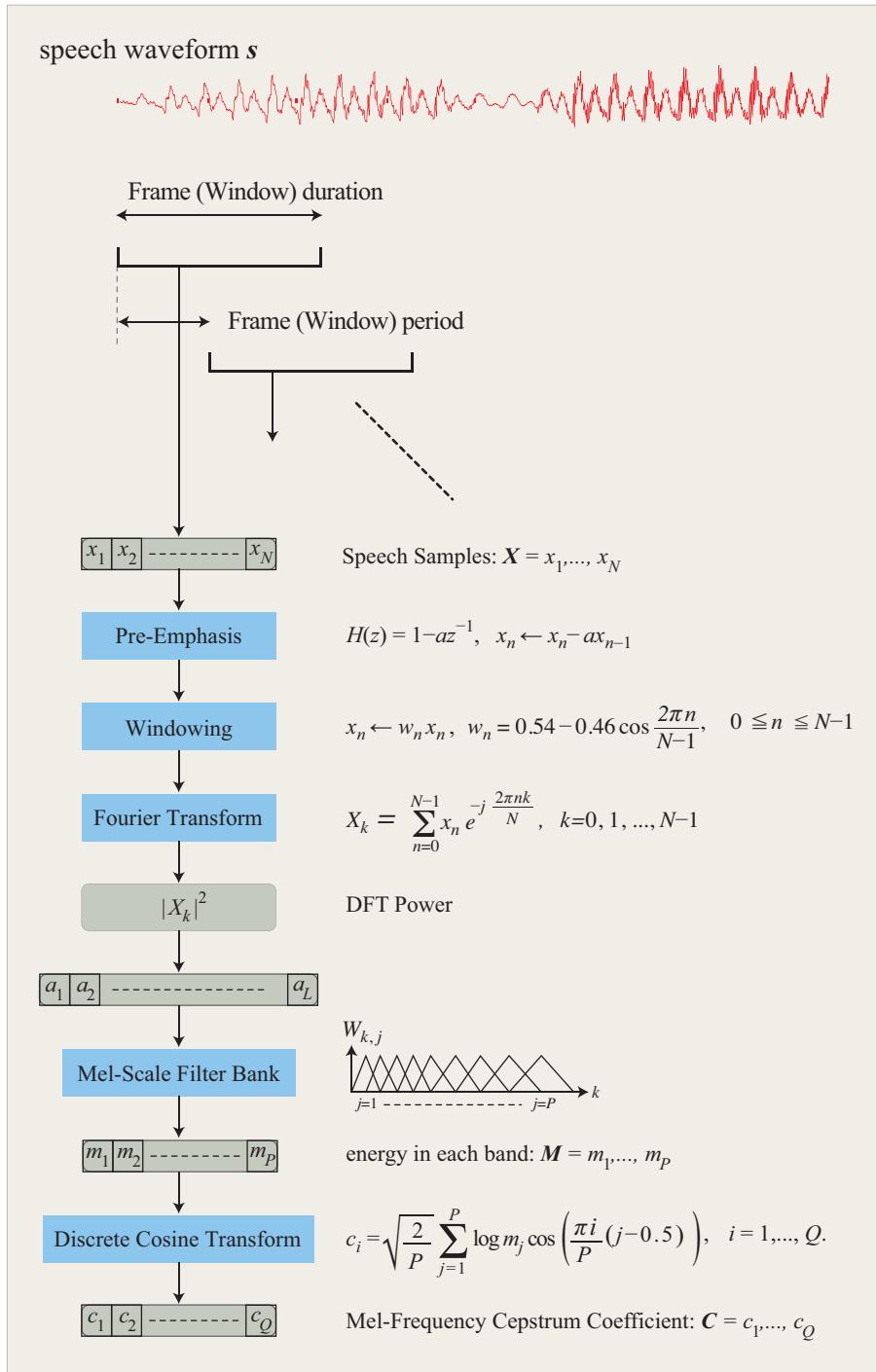


図 3.1: 音声波形からスペクトル特徴量 MFCC を計算する手順

道による調音, (3) 口・鼻からの放射の3つの作用の組み合わせとして, モデル化することができます (2.1.1節). 有声音源はパルス列として近似することができるのですが, これはかなり強い高域減衰特性  $-12 \text{ db/oct}$ を持ちます. 一方, 放射特性は, 逆に  $+6 \text{ db/oct}$  の特性を持つので, 最終的に音声は有声音の場合,  $-6 \text{ db/oct}$  の特性を示すことになるのです (図 3.2).

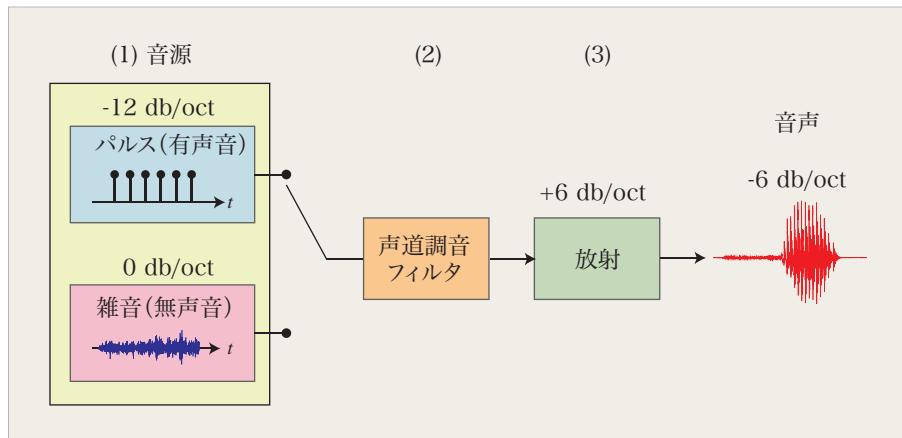


図 3.2: 音声生成の工学的モデル

この実験で我々が音声のスペクトルを計算する目的は、スペクトルに現れる音韻性（「あ」と「い」の性質の違いなど）の抽出です。音韻性は主として舌の形による声道調音フィルタの特性に由来します。したがって、音韻性の情報をより効率的に利用するための最初の処理として、音声スペクトルの  $-6 \text{ db/oct}$  の傾斜を元に戻します。すなわち、音声分析の前に  $+6 \text{ db/oct}$  の補正を行います。これを高域強調 (pre-emphasis) といいます。この処理は、音声振幅波形サンプル  $x_n$  に対する差分演算、

$$x_n \leftarrow x_n - \alpha x_{n-1}, \quad n = 1, \dots, N$$

あるいは1次のディジタルフィルタ

$$H(z) \triangleq 1 - \alpha z^{-1}$$

によって行います。ここで、 $\alpha$  は 1 に近い値（本実験では 0.97）に設定します。図 3.3 は、母音「あ」について高域強調のスペクトルに対する効果を示したもので、この処理により高周波数成分のパワーが持ち上がっているのがわかります。高域強調によってスペクトルが平坦化されるので、信号のダイナミックレンジを圧縮し、実質的に SNR<sup>3</sup>を向上させる効果もあります。

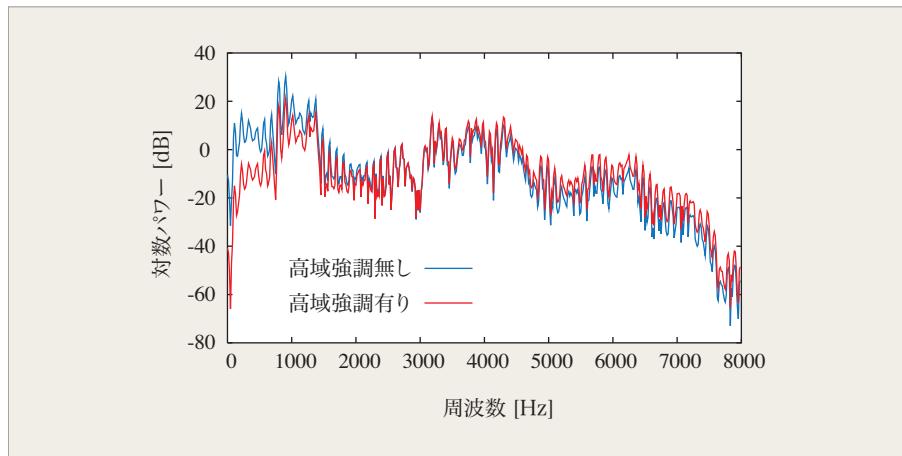


図 3.3: 高域強調の有無による「あ」の対数パワースペクトルの違い。

<sup>3</sup>Singal-to-Noise Ratio : 信号対雑音比。値が大きいほど良い。

### 3.3 窓関数

フレームで切り出した音声波形の端点は不連続になるので、信号の周期性を前提としたフーリエ変換などのスペクトル分析には不都合です。フレームの切り出しによる不連続の影響を低減するために、切り出した区間の両端の振幅を小さくするように重み関数を掛けることが行われます。このような関数を窓関数 (window function) といいます。

音声分析で良く用いられる窓関数の1つにハミング窓 (Hamming window) があります。この窓  $w_n$  は、

$$w_n \triangleq 0.54 - 0.46 \cos \frac{2\pi n}{N-1}, \quad 0 \leq n \leq N-1$$

で定義され、図3.4 (a) のような形をしています。この窓を、音声振幅波形サンプル  $x_n$  に掛け、

$$x_n \leftarrow w_n x_n, \quad n = 1, \dots, N$$

としたものをスペクトル分析の入力として用います。図3.4 (b) は実際の音声波形からフレームで切り取った波形、その波形にハミング窓を掛けた後の波形です。

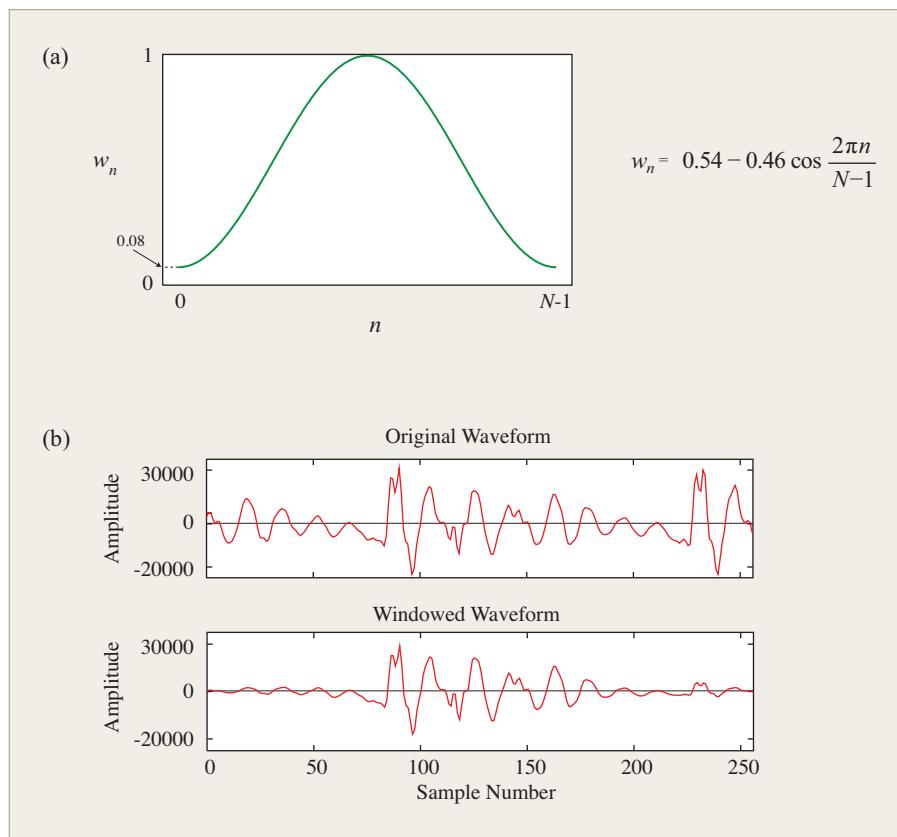


図 3.4: ハミング窓：(a) 窓関数、(b) 音声波形（上段）と窓掛けした音声波形（下段）。

### 3.4 離散フーリエ変換

音声波形サンプル  $x_1, \dots, x_N$  についての離散フーリエ変換 (Discrete Fourier Transform: DFT) 対は,

$$\begin{aligned} X_n &= \sum_{k=0}^{N-1} x_k e^{-j2\pi nk/N} \\ x_k &= \frac{1}{N} \sum_{n=0}^{N-1} X_n e^{j2\pi nk/N} \end{aligned}$$

と定義されます。DFT を効率的に計算する手法として高速フーリエ変換 (Fast Fourier Transform: FFT) があります。実習のプログラムでは FFT を用いています。FFT のアルゴリズムについては、信号処理の教科書およびソースコード ~/asr/wrecog/program/ad2fb.c を参照してください。

### 3.5 メル周波数

聴覚の周波数分解能は、低い周波数では高く、高い周波数では低いというように、周波数に対して非線形の特性を持っています。スペクトル分析をこの非線型特性に基づいて行うと、音声認識で良い結果が得られることが知られています。聴覚の非線型特性を反映した周波数として良く用いられるものとしてメル周波数 (Mel-frequency) があり、周波数  $f[\text{Hz}]$  とは、

$$\text{Mel}(f) \triangleq 2595 \log_{10}\left(1 + \frac{f}{700}\right) [\text{Mel}]$$

という近似関数 (図 3.5) で対応づけられることが知られています。

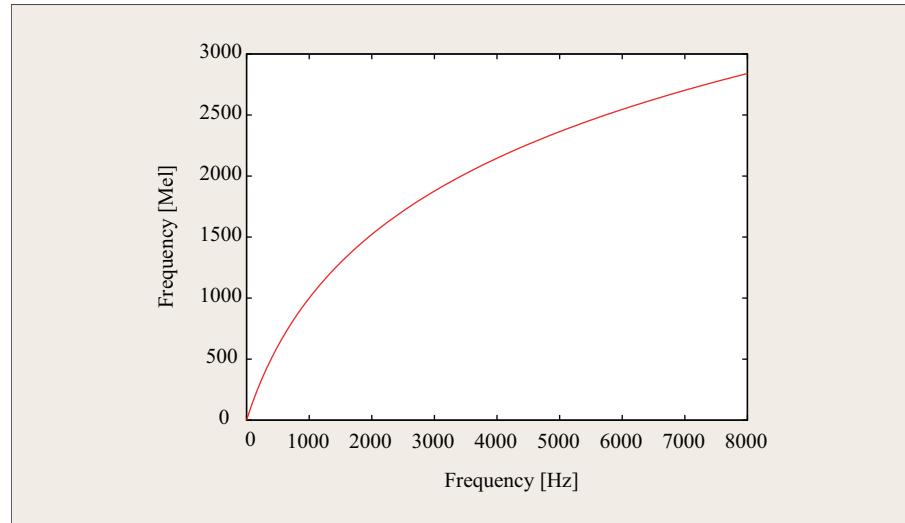


図 3.5: 周波数からメル周波数への変換関数  $\text{Mel}(f)$

### 3.6 フィルタバンク分析

聴覚の周波数に対する対数的特性を反映した分析方法です。まず、フレームで切り出して、高域強調をし、窓関数を掛けた音声波形サンプルをフーリエ変換

し、パワースペクトル  $|X_k|^2$  を求めます。つぎに、パワースペクトルをメル周波数軸上で等間隔、つまり、普通の周波数軸上では周波数が高くなるほど幅が広くなる三角窓が並んだフィルタバンク（filterbank）で処理をします。各三角窓の範囲をチャネル（channel）といいます。三角窓毎に窓の範囲にあるパワースペクトル  $|X_k|^2$  に対して<sup>4</sup>、窓の重み  $W_{k,j}$  を掛けて和をとることにより、そのチャネルの出力

$$m_j \triangleq \frac{1}{A_j} \sum_{k=k_{lo}(j)}^{k_{hi}(j)} W_{k,j} |X_k|^2, \quad j = 1, \dots, P \quad (3.1)$$

を計算します。 $P$  はチャネル数です。フィルタの重み  $W_{k,j}$  は、

$$W_{k,j} \triangleq \begin{cases} \frac{k - k_{lo}(j)}{k_c(j) - k_{lo}(j)}, & k_{lo}(j) \leq k < k_c(j) \\ \frac{k_{hi}(j) - k}{k_{hi}(j) - k_c(j)}, & k_c(j) \leq k \leq k_{hi}(j) \end{cases} \quad (3.2)$$

となります。 $k_{lo}(j), k_c(j), k_{hi}(j)$  はそれぞれ  $j$  番目のフィルタの下限、中心、上限のスペクトルの要素番号であり、隣り合うフィルタ間で  $k_c(j) = k_{hi}(j-1) = k_{lo}(j+1)$  という関係があります。中心周波数  $k_c(j)$  はメル周波数軸上では等間隔に並んでいます（図 3.6）。3.1 式の  $A_j$  はフィルタの面積を正規化する係数で、

$$A_j \triangleq \sum_{k=k_{lo}(j)}^{k_{hi}(j)} W_{k,j} \quad (3.3)$$

で定義されます。

フーリエ変換によって得られたスペクトル（図 3.3）には、細かく変化するギザギザの波形と大きく変化する凸凹があります。前者は有声音の音源（図 3.2）に由来する成分で、音源の基本周波数とその高調波成分です。後者は声道調音フィルタの特性を表わす成分です。各チャネルの帯域は基本周波数に比べてずっと広いので、チャネル内での積和計算（＝平均化）により高調波の影響が相殺されます。結果として、フィルタバンク出力には、声道調音フィルタ特性の概形が得られることになります。

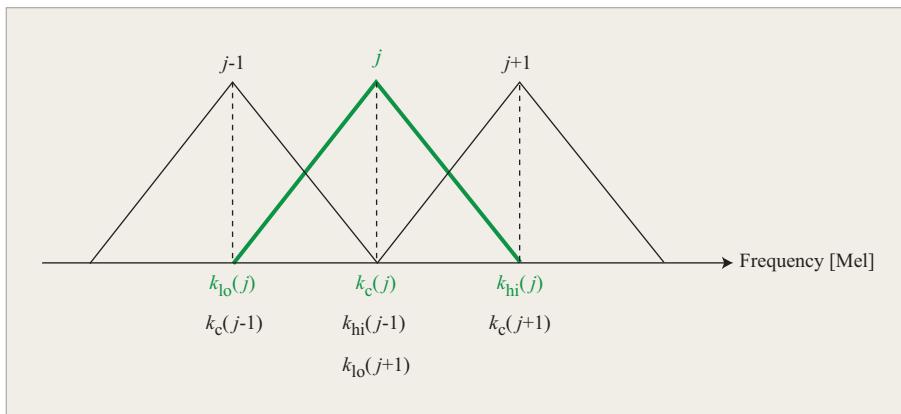


図 3.6: MFCC を計算するためのフィルタバンクの三角窓

<sup>4</sup>振幅スペクトル  $|X_k|$  を用いる計算法もあります。

### 3.7 メル周波数ケプストラム係数 (MFCC)

フィルタバンク分析によって得られた各チャネルの出力  $m_j(j = 1, \dots, P)$  の対数を離散コサイン変換 (Discrete Cosine Transform: DCT)

$$c_i \triangleq \sqrt{\frac{2}{P}} \sum_{j=1}^P \log m_j \cos\left(\frac{\pi i}{P}(j - 0.5)\right), \quad i = 1, \dots, Q \quad (3.4)$$

することにより、メル周波数ケプストラム係数 (Mel-Frequency Cepstrum Coefficient: MFCC) が得られます。 $Q$  はケプストラムの次元数です。

MFCC の値はマイクロフォンの種類、マイクロフォンと口の位置関係によって変動し、音声認識の結果に影響を与えます。その変動の影響を抑えるのに有効な方法としてケプストラム平均分散正規化 (Cepstral Mean and Variance Normalization, CMVN) があります。第  $t$  フレームのケプストラム係数ベクトルを  $\mathbf{c}_t = (c_1^{(t)}, \dots, c_Q^{(t)})$ 、ケプストラム係数ベクトルの時系列を  $\mathbf{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_T\}$  とするとき、正規化されたケプストラム  $\mathbf{c}_{t'}$  は、

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{T} \sum_{t=1}^T \mathbf{c}_t \\ \boldsymbol{\sigma} &= \sqrt{\frac{1}{T} \sum_{t=1}^T (\mathbf{c}_t - \boldsymbol{\mu})^2} \\ \mathbf{c}_{t'} &= \frac{\mathbf{c}_t - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \end{aligned}$$

と計算されます。ここで、 $\boldsymbol{\mu}$  は平均値ベクトル、 $\boldsymbol{\sigma}$  は標準偏差ベクトルです。平均値と標準偏差を計算する時系列  $\mathbf{C}$  の範囲は、音声認識処理の応用目的などによって、単語、文、段落、文章などが取られます。本実験では、単語単位の正規化を行ったものを特徴量として用いています。ちなみに、CMVN の処理は [ad2mfcc.c](#) というソースコードに記述されています。興味があれば確認しておいてください。

### 3.8 特徴抽出はデータ圧縮

音声波形から MFCC を得るまでの各段階でどのような結果が得られるのか、具体例で見てみましょう（図 3.7）。本実験の分析条件の下で順を追って説明します。サンプリング周波数は 16 kHz、フレーム幅は 32 ms なので、1 フレームの音声サンプル数は  $16 \times 32 = 512$  です。フレーム内に/a/の波形が 3 周期入っています。高域強調を行い、窓関数を掛けた音声波形のサンプル数は 512 です。これをフーリエ変換して得た振幅スペクトル、および振幅スペクトルを 2 乗したパワースペクトルの点数は 256 になります。スペクトルで大きな値を示しているのは/a/のフォルマント (formant)（2.2.1 節）に対応する周波数成分です。パワースペクトルでは振幅スペクトルに比べて周波数成分の強弱が強調されています。メルフィルタバンク分析のチャネル数は 28 なので出力は 28 次元です。チャネルの幅は低い周波数ほど広いので、スペクトルに現れているフォルマントピークは高周波数側が圧縮された位置で表示されています。フィルタバンク出力の対数を離散コサイン変換して得られる MFCC は 20 次元です。

スペクトル分析の過程において、音声認識に必要な情報を抽出しつつ、パラメータ数は 512 から 20 まで  $1/25.6$  に減っています。画像など他のメディアのパターン認識においても、このような特徴抽出（feature extraction）によって、不要なデータを捨て、必要な情報のみを残して認識に利用しています。つまり、特徴抽出はデータ圧縮（data compression）とみることができます。

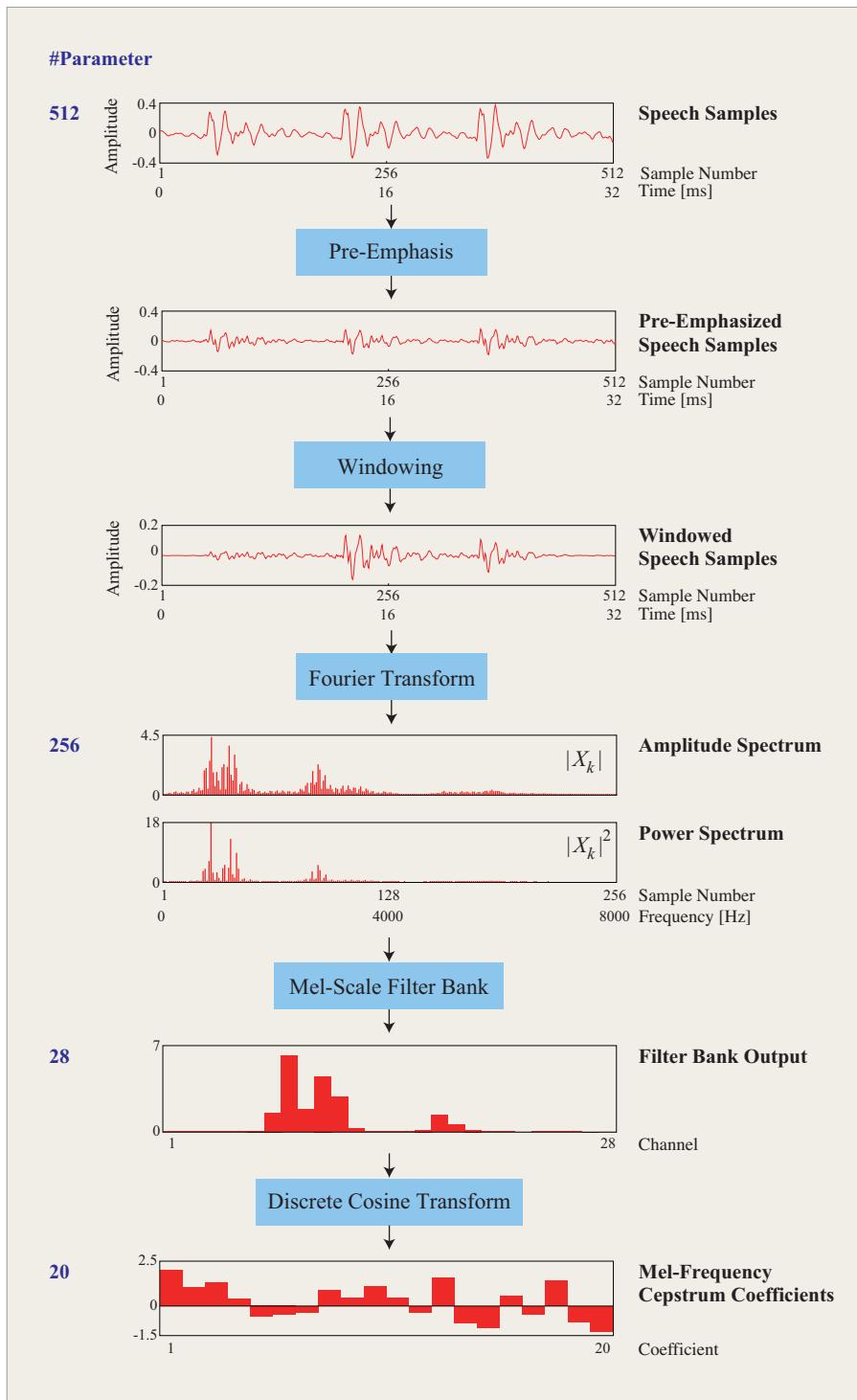


図 3.7: 512 点の/a/の音声波形をスペクトル分析し 20 次元の MFCC を計算。音声認識に必要な情報を抽出しつつ、パラメータ数は 512 から 20 まで  $1/25.6$  に削減することができました。特徴抽出はデータ圧縮の処理でもあるともいえます。

### 3.9 スペクトル分析例

実際の音声のスペクトル分析例で、音の違いがどのようにスペクトルのパターンに現われているかみましょう。図3.8は/kakuritsu/（「確率」）という単語の音声波形、スペクトログラム、音素ラベル、MFCCの値を表示したものです。

音声波形データの最初と最後には345 msの無音区間があります。単語音声は345 msから1000 msに存在し、その部分では音声波形の振幅が大きくなっています（図3.8 (a)）。音声波形に対応して、スペクトログラムの濃淡パターン（図3.8 (b)）は時間とともに絶えず変化していますが、局所的なパターンには他の部分とは明らかに異なった特徴があることがわかります。図3.8 (c)には、この波形の音素（第2.2節）名および音素の開始時間と終了時間を図示しました。この情報に基づいて音声波形とスペクトログラムに音素境界を引いてあります。そのようにしてみると、音によって波形やスペクトログラムのパターンに特徴があることがわかります。例えば、母音は4 kHz以下の低い周波数のエネルギーが強いこと、子音 /ts/ はその逆であること、さらに、同じ音素 /u/ であっても前後の音によってパターンに違いがみられることなどがわかります。MFCC（図3.8 (d)）のパターンも、局所的に特徴のあるパターンがみられ、音素ラベルと対応している様子がわかります。

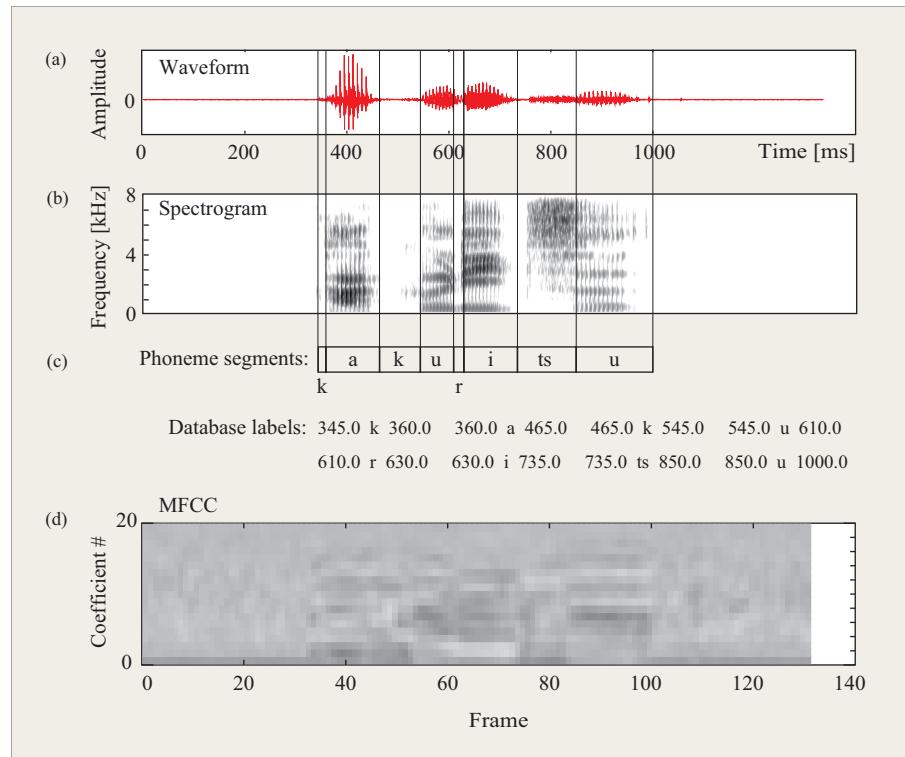


図3.8: 音声のスペクトル分析例 (/kakuritsu/「確率」). (a) 音声振幅波形, (b) スペクトログラム(周波数成分の強さを濃淡で表わしたもの), (c) ラベル情報, (d) メル周波数ケプストラム係数(MFCC).

## 3.10 実習

音声波形から MFCC を求める過程のうち、フィルタバンク分析と MFCC 分析の計算式を C 言語でプログラムし、サンプル音声を用いて計算結果を確認します。以下の説明は、[計算機実験を~/asr/wrecogで行うことを前提としています](#)。端末を開き、事前にこのディレクトリに移動しておいてください。

### 3.10.1 フィルタバンク分析

[program](#) というディレクトリに [ad2fb.c](#) という C の言語のソースファイルがあります。これは、音声波形からフィルタバンク出力を計算する関数です。ただし、肝心のコードの部分が空白になっています。穴埋め指示のある部分を埋めてソースコードを完成させてください。完成したら、以下のコマンドを入力してコンパイルを実行します。

```
[~/asr/wrecog]% make -C program fb
```

ソースコードに C 言語の文法エラーがある場合は、その旨メッセージが出力されるので、エラーメッセージが出なくなるまで修正してください<sup>5</sup>。文法エラーがない場合は、[fb](#) というコマンドができています。これは、音声波形を分析してフィルタバンクを出力するコマンドです。プログラムが正しく出来ているか確認しましょう。端末で、次のように入力して、計算確認用のサンプル音声[~/asr/wrecog/sample/sample.wav](#) を分析し、結果を [./mysample.fb](#) に保存します。

```
[~/asr/wrecog]% fb sample/sample.wav ./mysample.fb
```

このファイルと正解ファイルを比較して同じであれば、フィルタバンク分析のプログラムが正しく書かれていることになります。比較のために、

```
[~/asr/wrecog]% head -28 sample/sample.fb ./mysample.fb
```

と入力し、正解と自分の分析結果の第 1 フレームのフィルタバンク出力値を表示します。1 列目は時間 [ms]、2 列目はチャネル番号、3 列目はフィルタバンク出力値です。

```
[~/asr/wrecog]% head -28 sample/sample.fb ./mysample.fb
==> sample/sample.fb <==
 10.0  1      6.926785e-04
 10.0  2      5.409857e-01
 10.0  3      4.872707e-01
  (中略)
 10.0  26     4.739638e-02
 10.0  27     2.987086e-02
```

<sup>5</sup>コンパイル時にエラーメッセージが出なくなっても、プログラムの内容に間違いがないとは限りません。

```

10.0 28      2.556659e-02

==> ./mysample.fb <==

10.0 1      6.926785e-04
10.0 2      5.409857e-01
10.0 3      4.872707e-01
(中略)
10.0 26     4.739638e-02
10.0 27     2.987086e-02
10.0 28     2.556659e-02

```

この実行例では数値が一致していますが、計算の順序などによっては数値計算の誤差により若干異なる数値になることがあります。その場合でも、 少数点以下5桁まで一致していれば問題ありません。

### 3.10.2 MFCC 分析

`program` というディレクトリに `fb2mfcc.c` という C の言語のソースファイルがあります。これは、フィルタバンク出力から MFCC を計算する関数です。ただし、肝心のコードの部分が空白になっています。穴埋め指示のある部分を埋めてソースコードを完成させてください。完成したら、以下のコマンドを入力してコンパイルを実行します。

```
[~/asr/wrecog]% make -C program mfccf
```

ソースコードに C 言語の文法エラーがある場合は、その旨メッセージが出力されるので、エラーメッセージが出なくなるまで修正してください。文法エラーがない `mfccf` というコマンドができています。これは、音声波形を分析して MFCC を出力するコマンドです。プログラムが正しく出来ているか確認しましょう。端末で、次のように入力して、計算確認用のサンプル音声 `sample/sample.wav` を分析し、結果を `./mysample.mfcc` に保存します。

```
[~/asr/wrecog]% mfccf sample/sample.wav ./mysample.mfcc
```

このファイルと正解ファイルを比較して同じであれば、MFCC 分析のプログラムが正しく書けていることになります。比較はつぎのように行います。`.mfcc` ファイルはバイナリ形式なので、まず、`prtmdcc` コマンドによって、テキスト形式のファイルに変換します。

```
[~/asr/wrecog]% prtmdcc ./mysample.mfcc > ./mysample.mfcc.txt
```

つぎに、

```
[~/asr/wrecog]% head -20 sample/sample.mfcc.txt ./mysample.mfcc.txt
```

と入力し、正解と自分の分析結果の第1フレームのフィルタバンク出力値を表示します。1列目は時間 [ms]、2列目は次元番号、3列目は MFCC の値です。

```
[~/asr/wrecog] % head -20 sample/sample.mfcc.txt ./mysample.mfcc.txt
==> sample/sample.mfcc.txt <==
 10.0  1      0.997497
 10.0  2      -2.034434
 10.0  3      1.693631
(中略)
 10.0  18     2.062192
 10.0  19     -1.878791
 10.0  20     -0.653017

==> ./mysample.mfcc.txt <==
 10.0  1      0.997497
 10.0  2      -2.034434
 10.0  3      1.693631
(中略)
 10.0  18     2.062192
 10.0  19     -1.878791
 10.0  20     -0.653017
```

この実行例では数値が一致していますが、計算の順序などによっては数値計算の誤差により若干異なる数値になることがあります。その場合でも、 少数点以下5桁まで一致していれば問題ありません。

### 3.10.3 音声収録の練習

自分の声を録音してスペクトル分析をしてみましょう。ヘッドセットを用いて音声を録音し、録音した音声を聞いて正しく録音できているか検査します。普通の録音と異なり、実験用データとして音声収録を行うので、その手順には特別な注意が必要です。今回の実習のための手順を説明します。

音声の入出力を伴う実習を行うときは、音声入出力を伴う他のアプリケーション（メディアプレイヤー、YouTube や goo 辞書など）は、終了しておいてください。WaveSurfer など実習に用いる音声入出力アプリケーションの動作に影響を及ぼすことがあります。

#### 1. 録音

- (a) ヘッドセット (ELECOM HS-HP22TBK) のコードを解き、**フレキシブルパイプの根本の固い部分を持って、大きく回転してください**。マイクロフォンが左側に来るようヘッドセットを装着します。マイクロフォンの先端は**口のやや下**にセットします（図 3.9）。マイクロフォンの角度を変えるときは、本体の取り付け部の固い部分を回してください。マイクアームの柔らかいアームの部分は微調整に用います。マイクアームを**急な角度で曲げたり、捩ったりしない**ように扱ってください。これは、できるだけマイクロフォンと口の位置関係を一定に保つことにより、入力特性を安定させるためです。**ヘッドセットを頭に付けずマイクを持って録音しない**ようにしてください。再生音量を調整するボリュームコントローラーがありますので、確認してください。

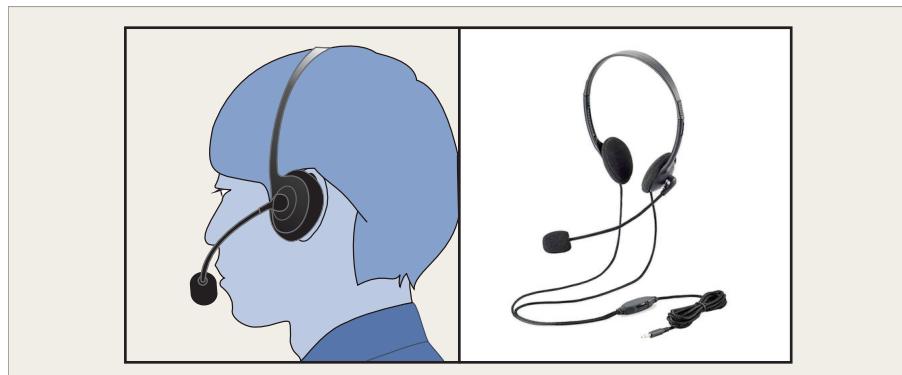


図 3.9: ヘッドセットは頭部にきちんと装着し、マイクは口のやや下にセットしてください。

- (b) PC の本体前面に音声入出力ジャックがあります。ヘッドセットの 4 極ミニプラグをしっかりと奥まで差し込んでください。プラグが適切に差し込まれると「オーディオデバイスを選択」が表示されるので、「ヘッドセット」を選択してください。（図 3.10）。



図 3.10: ヘッドセットの 4 極ミニプラグが適切に差し込まれると、「オーディオデバイスを選択」が表示される。「ヘッドセット」をクリックする。

- (c) 作業用のディレクトリ (`~/asr/wrecog`) に移動します。ディレクトリの移動は端末で `cd` コマンドによって行います。現在のディレクトリを確かめるためには `pwd` コマンドを使います。
- (d) 音量調節をします。ランチャー（デスクトップの左下角）から「設定」を起動してください（図 3.11）。
- (e) メニューから「サウンド」を選択します（図 3.12）

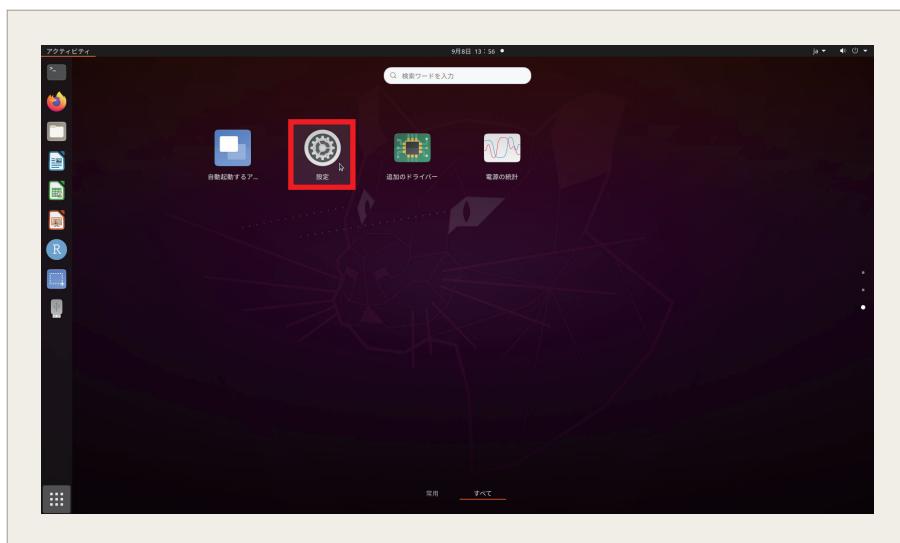


図 3.11: 音声入出力の設定をするために「設定」パネルを起動する。アイコンの配置はユーザによって異なります。

**出力** 「出力デバイス」メニューで「アナログヘッドフォン-内部オーディオ」を選択します。次に「テスト」クリックして、ヘッドフォンの左右の音の再生を検査します。マイクロフォンが取り付けられている方が左側です。「システム音量」と「音量レベル」を調節して、聴きやすい音量にします（図3.12）。

**入力** 「入力デバイス」メニューで「ヘッドセットマイクロфон-内部オーディオ」を選択します。次に、音量を調節します。リラックスした楽な発声で適当な文を読み上げ（例えばこの文）、音量の表示が7割付近で振れる状態になるように、スライドを調節します（図3.13）。このパネルはデスクトップ画面に出したままにしておいてください。「サウンド」設定パネルが開いた状態でないと、音声が入力されないことがあります。入出力音量は実験の途中で適宜調節するとよいでしょう。



図 3.12: ヘッドフォンの左右確認と音量調節



図 3.13: 入力音量の調節

(f) WaveSurfer<sup>6</sup>を起動します。端末のコマンド入力で、

```
% wavesurfer &
```

と入力してください。このとき、端末に「tkdnd not found, no drag and drop support」と表示されることがあります、実習に影響無いので無視して構いません。

(g) 録音条件を設定します。サンプリング周波数を16kHz、1チャンネルに設定してください。まず、「File」メニューの「Preferences...」を選択します（図3.14）。

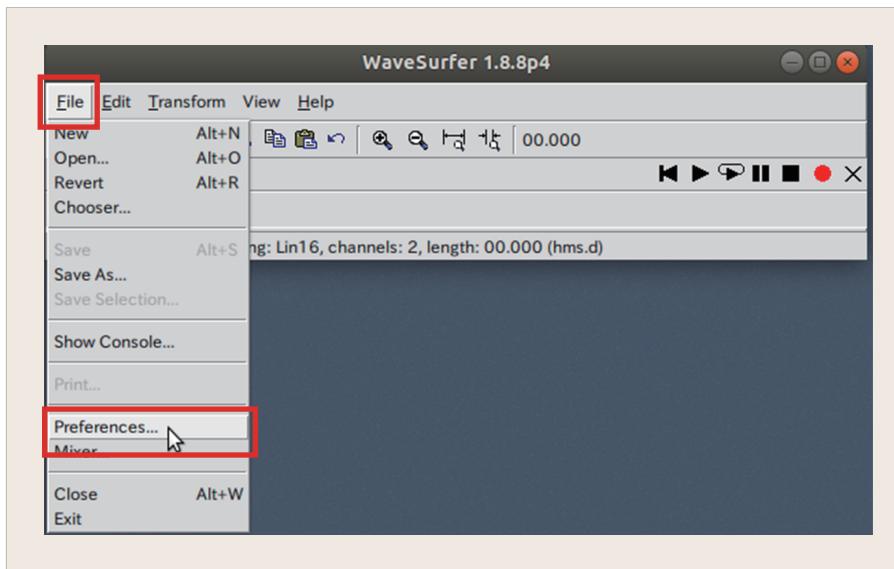


図3.14: WaveSurfer の録音条件の設定パネルの起動

つぎに、「Sound I/O」パネルの「New sound default rate:」に“16000”，「New sound default channels:」に“1”を設定します。「New sound default encoding:」を“Lin16”，「Record time limit:」を“600”に設定し、パネル下に表示されている「OK」ボタンを押してください（図3.15）。

<sup>6</sup>スウェーデンのKTHが開発したオープンソースの音声分析ソフトウェア。Linux, Windows, macOSで動作。<https://sourceforge.net/projects/wavesurfer/>

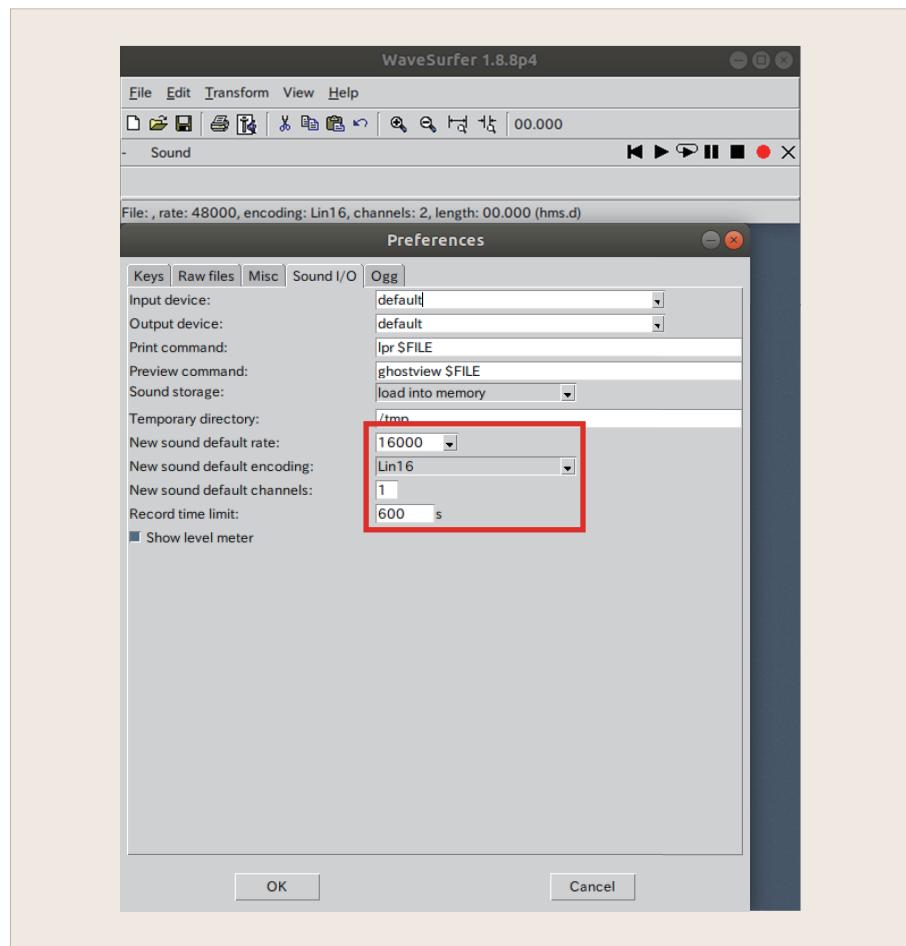


図 3.15: WaveSurfer の録音条件の設定. サンプリング周波数=16000Hz(16kHz), チャネル数=1. 録音時間制限は 600 秒.

- (h) WaveSurfer で新しい窓を生成します. 「File」メニューの「New」を選択すると (図 3.16 (a)), 新しい窓の種類を指定するための「Choose Configuration」というダイアログボックスが表示されます. 「Waveform」を選択してください (図 3.16 (b)).

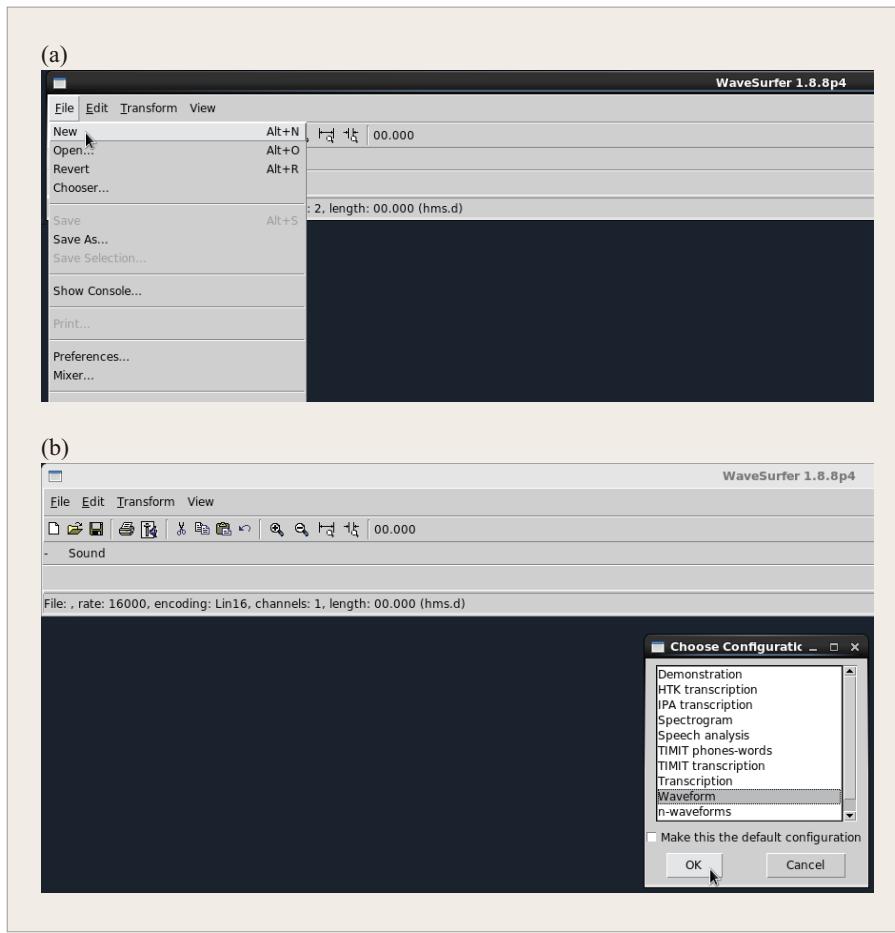


図 3.16: WaveSurfer を起動して新しい窓を作成する。

- (1) WaveSurfer の窓の中央でマウスを右クリックして押したままとし、現われたメニューの「Create Pane」を選択して現われたメニューから「Time Axis」を選択して時間目盛を表示します（図 3.17）。時間目盛は左端に「time」と表示されていますが、初期状態では空白です。

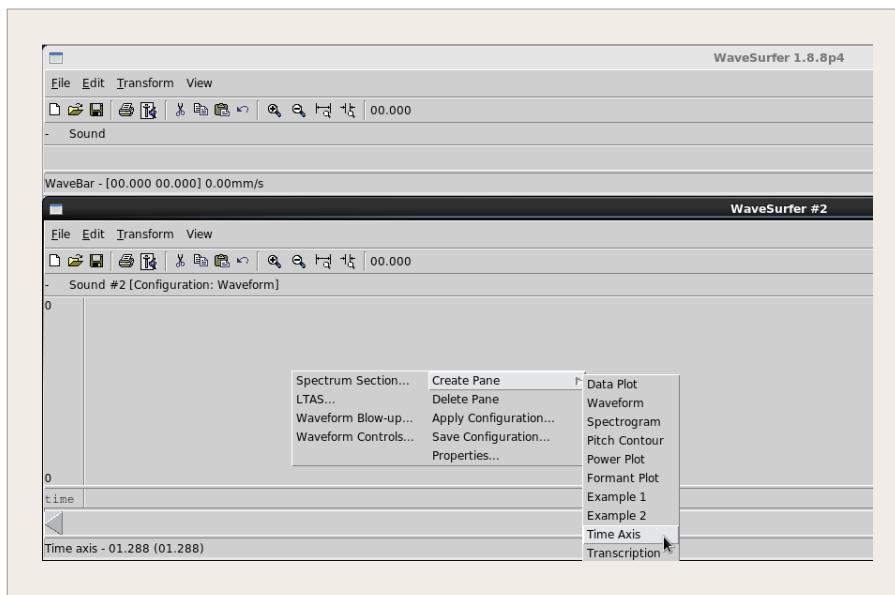


図 3.17: 時間目盛を表示する。

- (j) WaveSurfer の録音ボタンを押すと収録が始まります。画面に音声波形がリアルタイムに表示されます。自分の名前 (姓 (family name) または名 (given name) あるいはそれに相当する呼び名) を 5 回録音してください。普通に会話をするときの自然な感じで発声してください。単語の間は 1 秒程度置くようにしてください。音量の調整はマイクの位置でも調整することができます。波形の振幅が範囲 ( $-32768 \sim +32767$ ) を超えてしまうと (図 3.18)，歪んだ音声になってしまいます。歪んだ音声は音声分析・認識のデータとして用いることができません。今回は練習として 5 回発声しますが、実際に分析に用いるのは 5 つの発話のうちの 1 つです。一般に音声収録において、最初の方の発声は不安定で、中間辺りで発声された単語がサンプルとして適当な場合が多いです。2 番目以降から比較的明瞭発音のものを 1 つ選んでください。

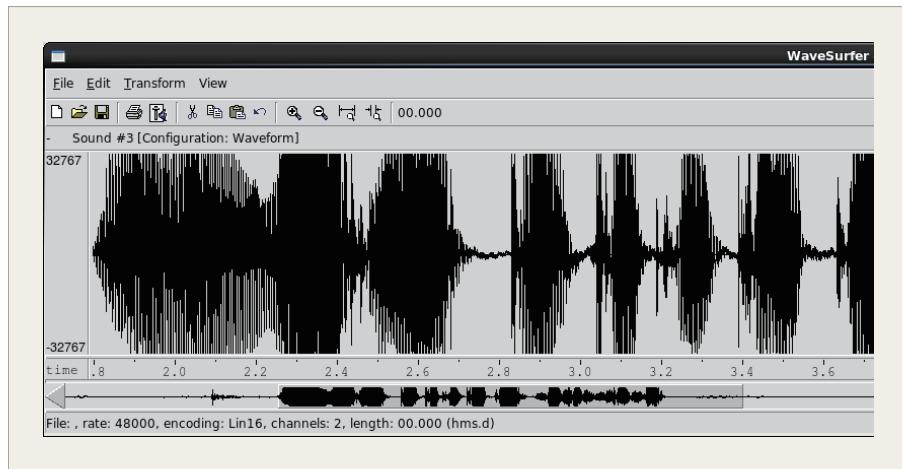


図 3.18: 入力レベルが過大で最大値を超えた部分は音声が歪んでしまう。

- (k) 停止ボタンを押して、録音を終えます。WaveSurfer は録音した音声波形全体をバッファに蓄えています。
- (l) 再生ボタンを押して、一度全体を聞きましょう。どうでしたか？録音された音声が妥当かどうか判断できない場合は、スタッフに聞いてください。
- (m) 録音した音声を編集します。まずは、再生ボタンを押して、一度全体を聴きましょう。5 つの中から良い発声と思うものを 1 つ選んで保存します。音声の一部だけを選んで聴きたい場合は、マウスの左ボタンを押しながら範囲を指定し、再生ボタンを押します (図 3.19)。

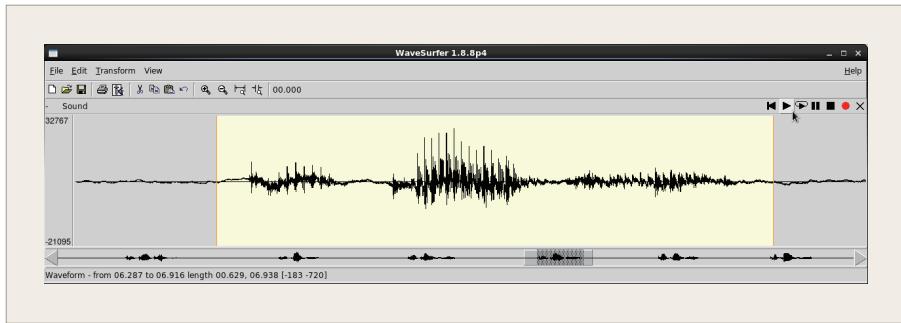


図 3.19: 音声の一部を選択する。

次に、「File」メニューの「New」を選択して新しい窓をつくり、そこに5つの発話がある元のファイルから選択した1つの発話の波形をコピー&ペーストします（図3.20）。音声の前後に100ms（0.1s）の余白を付けてください。横軸の表示倍率によって目盛の時間幅が異なるので、間違えないように十分注意してください。余白が多すぎると、スペクトル表示の結果を見たときに、音の特徴やスペクトルの変化を観察しにくくなります。

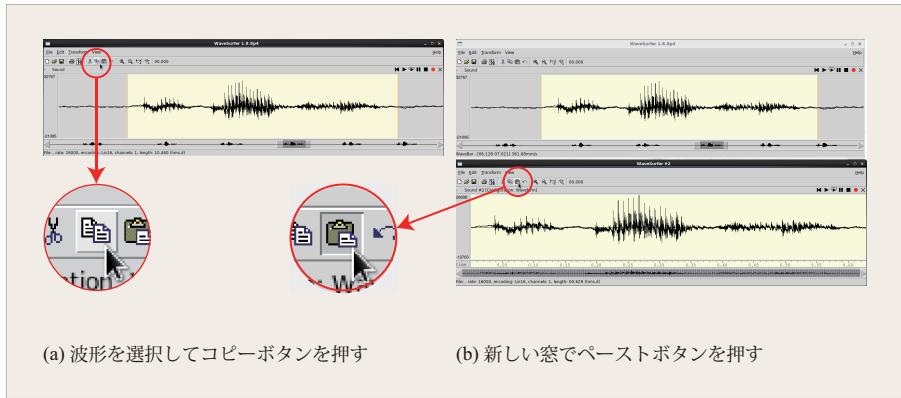


図 3.20: 選択した音声をコピーして、新しい窓（バッファ）に貼り付ける。

- (n) 編集が済んだ録音データをファイルに保存します。メニューから「File」→「Save As」を選択し、ファイル名を入力します。「Files of type:」は「MS Wav Files (\*.wav, \*.WAV)」を指定してください。ファイルを保存するディレクトリは`~/asr/wrecog/wav`とし、ファイル名は「名前のローマ字表記.wav」とします（図3.21）。以下では、録音した音声データファイルを `takagi.wav` として説明します。

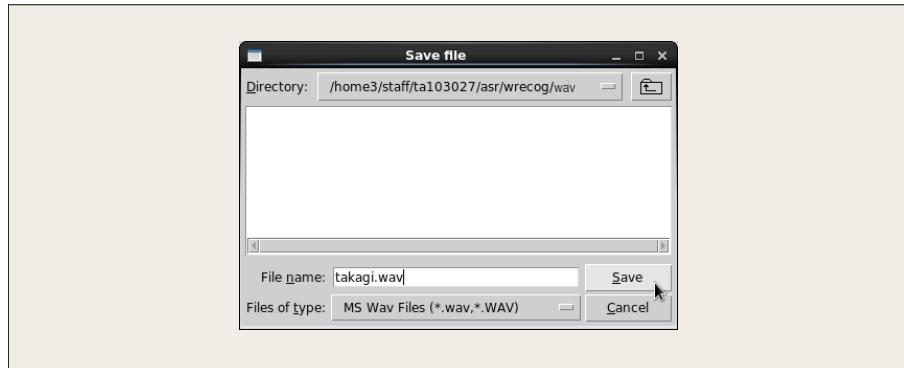


図 3.21: WAV 形式で音声波形を保存する。保存するディレクトリは ~/asr/wrecog/wav。拡張子.wav を付け忘れないように。

- (o) WAV 形式で音声データで保存されていることを確認するため、端末で、`play wav/takagi.wav` と入力し、録音した音声が再生されることを確認してください。`play` コマンドを実行すると、ヘッドセットから音声が聞こえ、

```
[~/asr/wrecog]% play wav/takagi.wav
play WARN alsa: can't encode 0-bit Unknown or not applicable

wav/takagi.wav:

File Size: 28.8k      Bit Rate: 256k
Encoding: Signed PCM
Channels: 1 @ 16-bit
Samplerate: 16000Hz
Replaygain: off
Duration: 00:00:00.90

In:100% 00:00:00.90 [00:00:00.00] Out:14.4k [ -====|===== ]      Clip:0
Done.
[~/asr/wrecog]%
```

というように端末に表示されます。“Encoding:”(波形値記録形式), “Channels:”(チャネル数および量子化ビット数), “Samplerate:”(サンプリング周波数) が上記の例と同じでなければなりません。“Duration”は音声データの長さです。この際に表示される“rec WARN alsa: can't encode 0-bit Unknown or not applicable”的メッセージは無視して結構です。

## 2. スペクトル分析

保存した音声のフィルタバンク分析およびMFCC分析を行い、音声波形と並べて図として表示します。そのため `drawspec` というコマンドを使います。端末で、

```
[~/asr/wrecog]% drawspec wav/takagi.wav
```

と入力すると、`fb` と `mfccf` でこの音声ファイルを分析し、その結果を音声波形とともに `gnuplot` を用いて、図 3.22 のように表示します。

この図を PNG ファイルとして保存するときは、

```
[~/asr/wrecog]% drawspec wav/takagi.wav png takagi.png
```

と入力してください<sup>7</sup>. 図 3.22 のような出力が得られます. ただし, フィルタバンクの出力値は対数を取っています. PNG ファイルのファイル名は, この例のように「発話内容.png」とします.

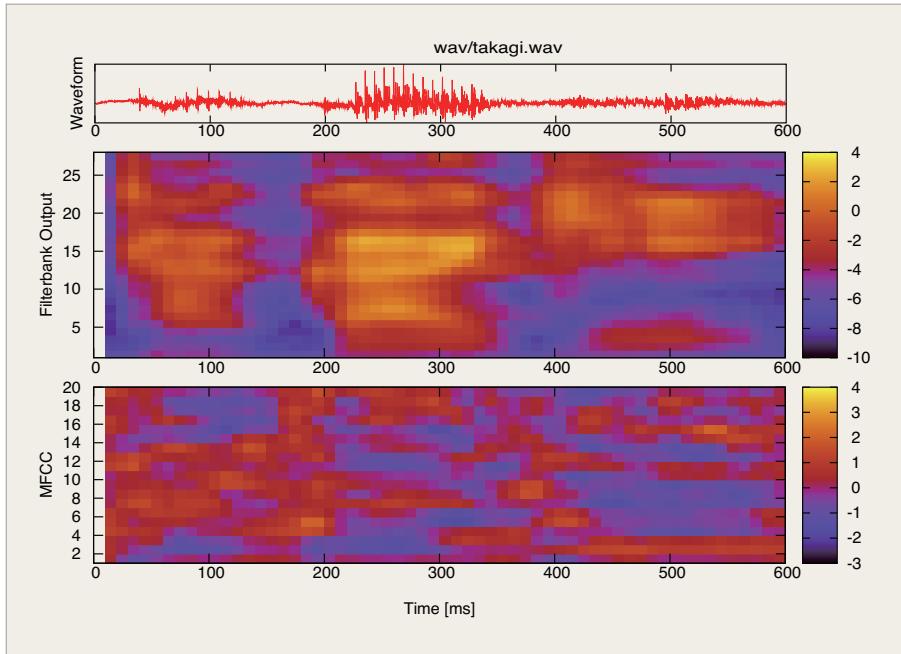


図 3.22: drawspec の出力例. 発話は/takagi/. 上から, 音声波形, フィルタバンク出力, メル周波数ケプストラム係数 (MFCC). 横軸は時間 [ms].

スペクトル分析の結果の図の中段に描かれるフィルタバンク出力に現れる音声のスペクトルパターンを見てみましょう. 縦軸はフィルタバンクチャネル番号です. フィルタバンク出力にはメル周波数軸で分析した声道調音フィルタ特性の概形が得られます (3.6 節, 3.8 節). 母音のフォルマント (2.2.1 節) に対応するチャネルの出力値は大きいので, 明るい色で表示されています.

図 3.22 は/takagi/という音声データの 50ms~130ms 付近と 210ms~330ms 付近の母音/a/の区間では, 第 6, 第 13, および第 17 チャネル付近に明るい領域があります. 母音/a/のフォルマントがここに現れているのです. この音声データの 460ms~580ms 付近は母音/i/です. 明るい領域は第 3, 第 16, および第 22 チャネル付近にあります. 母音/a/とはフォルマントの位置が異なることが分かります.

暗い色で表示されているのはエネルギーが少ない周波数の領域です. 無声子音の区間や無音声の区間にに対応していることが分かります.

自分の音声データに含まれている音素の種類や位置とフィルタバンク出力のパターンの対応を観察してください.

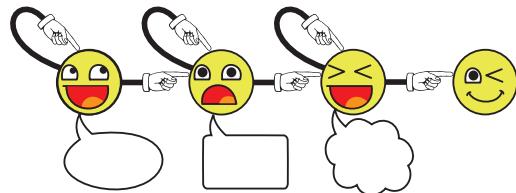
<sup>7</sup>drawspec はシェルスクリプトです. 他の形式で保存したい場合は自分で改造してください.

### 3.10.4 レポート（第1週）

1. 実習課題の目的
2. プログラミングおよびスペクトル分析
  - (a) `ad2fb.c` および `fb2mfcc.c` の穴埋め部分.
  - (b) 自分の名前の分析結果（図3.22の形式の `drawspec` の出力). 第2章を参考にして、メルフィルタバンク出力について上記の例のような所見を、できる範囲で良いので、述べてください。メルフィルタバンクのチャネルとその中心周波数の対応は以下のとおりです。

| channel | center frequency [Hz] | channel | center frequency [Hz] |
|---------|-----------------------|---------|-----------------------|
| 1       | 64                    | 15      | 1877                  |
| 2       | 133                   | 16      | 2111                  |
| 3       | 208                   | 17      | 2367                  |
| 4       | 291                   | 18      | 2645                  |
| 5       | 381                   | 19      | 2949                  |
| 6       | 479                   | 20      | 3280                  |
| 7       | 586                   | 21      | 3641                  |
| 8       | 703                   | 22      | 4035                  |
| 9       | 830                   | 23      | 4465                  |
| 10      | 969                   | 24      | 4934                  |
| 11      | 1120                  | 25      | 5446                  |
| 12      | 1286                  | 26      | 6004                  |
| 13      | 1466                  | 27      | 6612                  |
| 14      | 1663                  | 28      | 7276                  |

3. 考察
4. 一般事項
  - (a) 本日のポイントは何であったか？
  - (b) 良くわかったこと
  - (c) わからなかったこと
  - (d) 要望
  - (e) 感想



# 第4章 統計的音声認識

## 4.1 枠組み

音声認識システムでは、音声による通信を、話者のメッセージ  $M$  が記号列  $W$  に符号化され、それが音声波形  $S$  に形を変えて通信路を経由し、聴き手に  $\tilde{S}$  として伝わり、 $\tilde{S}$  の特徴分析で得られた観測時系列  $O$  から記号列  $\tilde{W}$  を復元し、その記号列から発話者のメッセージ  $\tilde{M}$  を推定する過程と捉えます（図 4.1）。

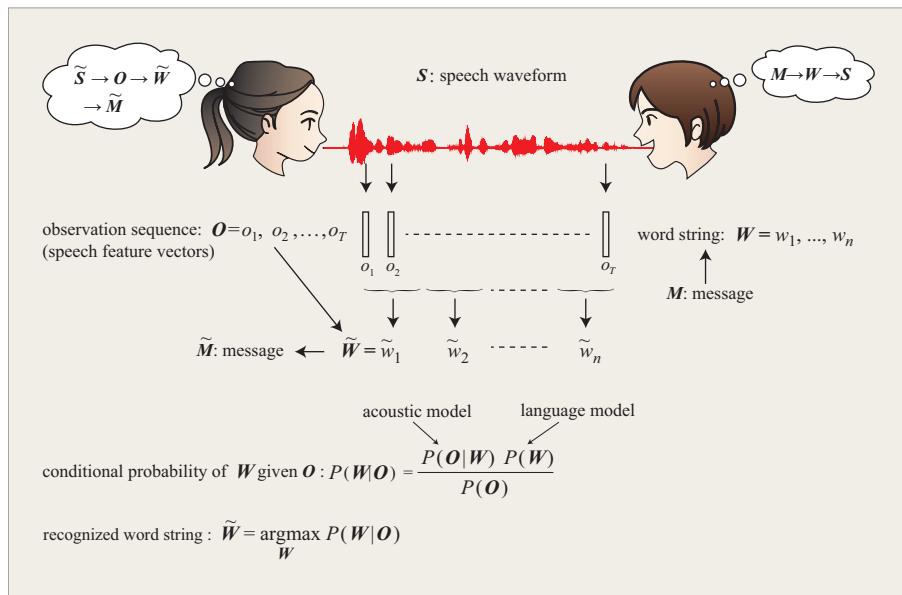


図 4.1: 音声通信の統計的解釈。 $M$ : メッセージ（話者が伝えたい内容）、 $W$ : 単語列（発話）、 $S$ : 音声波形、 $\tilde{S}$ : 聴き手が受け取った音声波形、 $O$ : 観測系列（聞き手の受け取った音声特徴の系列）、 $\tilde{W}$ : 推定単語列、 $\tilde{M}$ : 推定メッセージ

音声認識の究極の目的は、音声波形  $\tilde{S}$  から話者のメッセージ  $M$  の推定値  $\tilde{M}$  を得ることであるともいえます。たとえば、部屋に人間とロボットが居るとして、人間が室温を下げるといいたいと、ロボットに「暑いですね。」と話したとします。その発話を聴いてロボットがエアコンの温度を調節して室温を下げてくれたならば、ロボットは人間のメッセージ（意図）を理解したということができます。ロボットにこのような対応をさせるためには、単に聴いた音声波形を「暑いですね。」という文に変換することができるだけではダメです。発話の字面の背後にある意図を理解することが必要であり、音声理解（speech understanding）と呼ばれています。現在、一般に音声認識と呼ばれている技術は、意図理解までは行わず、話者の発した単語列  $W$  の推定値  $\tilde{W}$  を求める指しています。

## 4.2 基本原理

最大事後確率基準に基づくベイズ (Bayes) の識別規則に基づいた音声認識の原理を説明します。

音響特徴量の系列を  $\mathbf{O}$  とします。 $\mathbf{O}$  は、 $m$  次元の実数値ベクトル  $\mathbf{o}_t \in R^m$  の長さ  $T$  の系列

$$\mathbf{O} \triangleq \mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T \quad (4.1)$$

とします。一般に、 $\mathbf{O}$  は音声波形サンプル  $\mathbf{X} = (x_1, \dots, x_N)$  に対して短時間フレーム分析を行うことにより得ます。発話された単語列を

$$\mathbf{W} \triangleq w_1, w_2, \dots, w_n, \quad w_i \in \mathcal{W} \quad (4.2)$$

とします。 $\mathbf{W}$  は、ある有限の語彙  $\mathcal{W}$  の要素  $w_i$  からなる長さ  $n$  の単語列です。

$P(\mathbf{W}|\mathbf{O})$  は音響特徴量系列  $\mathbf{O}$  が与えられたとき単語列  $\mathbf{W}$  が発話された条件付き確率であり、事後確率 (posterior probability) と呼ばれます。ベイズの定理により、 $P(\mathbf{W}|\mathbf{O})$  は、

$$P(\mathbf{W}|\mathbf{O}) = \frac{P(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{O})} \quad (4.3)$$

と書き直すことができます。

$P(\mathbf{W})$  は単語列  $\mathbf{W}$  の生起確率です。たとえば、単語列  $\mathbf{W}$  が 0.7 の確率で生起し、他の単語列をすべて合わせても 0.3 の確率でしか生起しなければ、どの観測データ  $\mathbf{O}$  に対してもそれが単語列  $\mathbf{W}$  に属すると答えれば、0.7 の確率で正答となります。観測する前からわかっている確率という意味で、これを事前確率 (prior probability) と呼びます。

$P(\mathbf{O}|\mathbf{W})$  は単語列  $\mathbf{W}$  が生起したという条件の下で音声特徴量時系列  $\mathbf{O}$  が観測される確率を表わしていて、クラス条件付き確率 (class conditional probability) です。 $\mathbf{O}$  が  $\mathbf{W}$  に属しているのが尤もらしいと考えられる確率と解釈することができることから尤度 (likelihood) とも呼ばれます<sup>1</sup>。

$P(\mathbf{O})$  は  $\mathbf{O}$  が観測される確率であり、 $\mathbf{O}$  と  $\mathbf{W}$  の同時確率  $P(\mathbf{O}, \mathbf{W})$  から

$$P(\mathbf{O}) = \sum_{\mathbf{W}} P(\mathbf{O}, \mathbf{W}) = \sum_{\mathbf{W}} P(\mathbf{O}|\mathbf{W})P(\mathbf{W}) \quad (4.4)$$

のように可能な単語列全体の関する和をとることで得られます。このような操作を周辺化といい、 $P(\mathbf{O})$  を周辺確率と呼びます。

式 4.3 を眺めてみると、事後確率  $P(\mathbf{W}|\mathbf{O})$  は事前確率  $P(\mathbf{W})$  をクラス条件付き確率と周辺確率の比  $P(\mathbf{O}|\mathbf{W})/P(\mathbf{O})$  で修正したものとみることができることに気が付くでしょう。すなわち、単語列  $\mathbf{W}$  が与えられたときに観測値  $\mathbf{O}$  が得られる尤度  $P(\mathbf{O}|\mathbf{W})$  が、単語列  $\mathbf{W}$  が与えられない場合の確率  $P(\mathbf{O})$  よりも大きければ、事後確率は事前確率よりも大きくなり、そうでなければ事後確率が事前確率よりも小さくなります。

---

<sup>1</sup>ただし、尤度の和は  $\sum_{\mathbf{W}} P(\mathbf{O}|\mathbf{W}) \neq 1$  なので、厳密な意味で確率とはいえません。

単語列  $\mathbf{W}_i$  と  $\mathbf{W}_j$  の識別境界 (discrimination boundary) は、事後確率が等しくなるところ、すなわち、

$$P(\mathbf{W}_i|\mathbf{O}) = \frac{P(\mathbf{O}|\mathbf{W}_i)P(\mathbf{W}_i)}{P(\mathbf{O})} = \frac{P(\mathbf{O}|\mathbf{W}_j)P(\mathbf{W}_j)}{P(\mathbf{O})} = P(\mathbf{W}_j|\mathbf{O}) \quad (4.5)$$

が成立するところです。式 4.5 からわかるように、周辺確率  $P(\mathbf{O})$  は単語列に共通に現れているので、識別規則に含める必要はありません。したがって、音声認識器は  $P(\mathbf{O}|\mathbf{W})P(\mathbf{W})$  を最大化する単語列  $\tilde{\mathbf{W}}$ 、

$$\tilde{\mathbf{W}} \triangleq \underset{\mathbf{W}}{\operatorname{argmax}} P(\mathbf{O}|\mathbf{W})P(\mathbf{W}) \quad (4.6)$$

を求めることになります。 $P(\mathbf{O}|\mathbf{W})$  を音響モデル (acoustic model)、 $P(\mathbf{W})$  を言語モデル (language model) と言います。

# 第5章 隠れマルコフモデル (HMM)

隠れマルコフモデル (Hidden Markov Model: HMM) は、出力シンボルによって一意に状態遷移を決定することができない非決定性確率有限状態オートマトンとして定義されます。出力シンボル系列が与えられても状態遷移系列を一意に決めることができません、言い換れば、観測できるのはシンボル系列だけであることから hidden (隠れ) マルコフモデルと呼ばれています。HMM は音声認識システムの音響モデル  $P(O|W)$  として用いられます。

## 5.1 基本 HMM の定式化

HMM は以下のパラメータで規定されます。

$N$  モデルの状態数。一般に、状態は互いに接続されていて、任意の状態から（それ自身も含めて）他の任意の状態に遷移することができます。状態の集合を  $S \triangleq \{S_1, S_2, \dots, S_N\}$  で表わすことにします。

$M$  出力シンボル種類数。モデル化する信号の離散記号を  $\mathcal{V} \triangleq \{v_1, v_2, \dots, v_M\}$  で表わします。

**A** 状態遷移確率  $\mathbf{A} = \{a_{ij}\}$ 。時刻  $t$  での状態を  $q_t$  とするとき、

$$a_{ij} \triangleq P(q_{t+1} = S_j | q_t = S_i), \quad 1 \leq i, j \leq N, \quad \sum_{j=1}^N a_{ij} = 1.$$

**B** シンボル出力確率。状態  $j$  における出力シンボルの確率分布  $\mathbf{B} = \{b_j(k)\}$ 。ここで、

$$b_j(k) \triangleq P(v_k | q_t = S_j) \quad 1 \leq j \leq N, \quad 1 \leq k \leq M.$$

**$\pi$**  初期状態確率  $\boldsymbol{\pi} = \{\pi_i\}$ 。

$$\pi_i \triangleq P(q_1 = S_j), \quad \sum_{i=1}^N \pi_i = 1, \quad 1 \leq i \leq N.$$

適当な  $N, M, \mathbf{A}, \mathbf{B}$ , および  $\boldsymbol{\pi}$  が与えられれば、以下の手順を用いて、HMM を  $O = o_1, o_2, \dots, o_T, \quad o_t \in \mathcal{V}$  なる観測系列の生成器として用いることができます。

1. 初期状態確率分布  $\boldsymbol{\pi}$  に基づいて、初期状態  $q_1 = S_i$  を選ぶ。
2.  $t = 1$  と設定する。

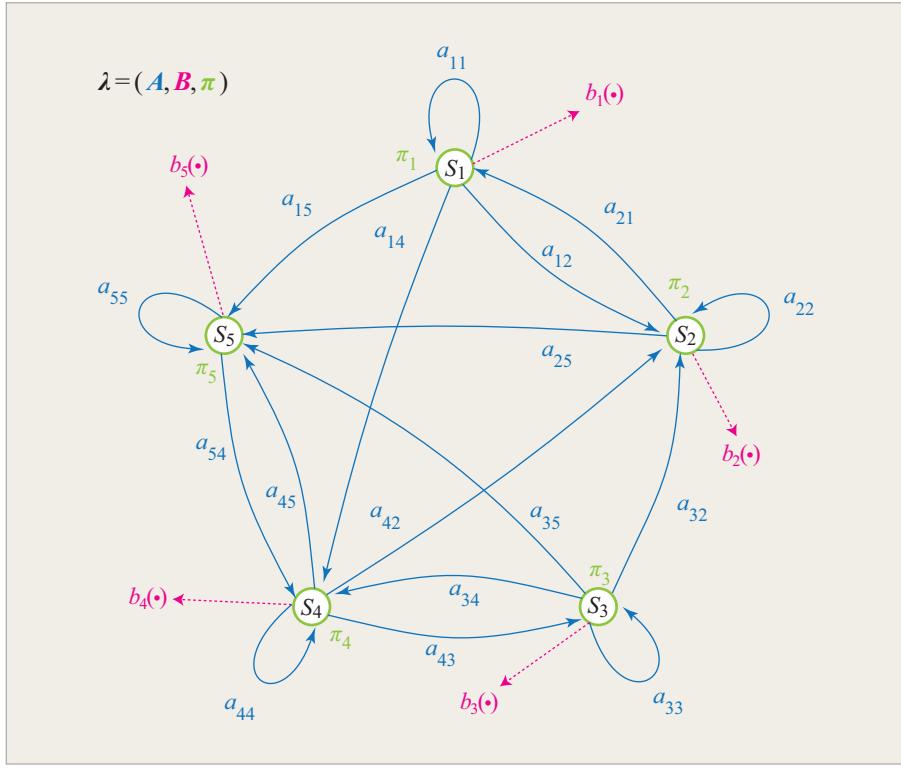


図 5.1: 5 状態の隠れマルコフモデル。確率 0 の状態遷移は描いていない。

3. 状態  $S_i$  のシンボル出力確率分布  $b_i(k)$  にしたがって,  $o_t = v_k$  を選ぶ.
4. 状態  $S_i$  の状態遷移確率  $a_{ij}$  にしたがって, 次の時刻の状態  $q_{t+1} = S_j$  に遷移する.
5. 時刻を 1 進める, すなわち,  $t \leftarrow t + 1$  とする. もし,  $t < T$  ならばステップ 3 に戻り, さもなければ終了する.

以上により, HMM を規定するためには,  $N$ ,  $M$ , 観測系列の仕様, 確率に関するパラメータ  $A$ ,  $B$ ,  $\pi$  が必要ですが. 簡単のため, モデル  $\lambda$  のパラメータを

$$\lambda = (A, B, \pi)$$

と表わすことにします.

## 5.2 HMM の3つの問題

HMM を実際に用いるためには, 次の3つの問題の解が必要です.

**問題 1** 観測系列  $O = o_1, o_2, \dots, o_T$  と, モデル  $\lambda = (A, B, \pi)$  が与えられたとき,  $P(O|\lambda)$  を効率的に計算する.

**問題 2** 観測系列  $O = o_1, o_2, \dots, o_T$  と, モデル  $\lambda$  が与えられたとき, 最適な状態遷移系列  $Q = q_1 q_2 \cdots q_T$  を求める.

**問題 3**  $P(O|\lambda)$  を最大化するモデル  $\lambda$  を求める.

問題 1 は、評価、つまり、モデルと観測系列が与えられたとき、その観測系列がそのモデルから生成される確率を計算する問題です。見方を変えれば、モデルが観測系列にどれだけ適合しているかの点数付けを行うことです。後者の観点で捉えると、複数のモデルの中から与えられた観測系列に最も適合するものを選ぶ（クラス分類する）ことができます。

問題 2 は、モデルの隠された部分（状態）に関することです。理論上、与えられた観測系列に対する“正しい”状態遷移系列というものを得ることはできません。しかし、適当な最適化基準を設定し、その下でもっともらしい状態系列を推定することは可能です。推定ができるれば、モデルの構造を調べたり、音声のセグメンテーション<sup>1</sup>に応用することができます。

問題 3 は、与えられた観測系列が最もよく生成されるようにモデルのパラメータを最適化することです。このとき用いられる観測系列を学習データといい、モデルパラメータを最適化することを HMM を学習するといいます。

以下、5.3 節では、まず、HMM が観測系列を生成する確率の計算の演算量の問題を考えます。5.3.1 節および 5.3.2 節では、問題 1 の解法である Forward アルゴリズムおよび Backward アルゴリズムを説明します。問題 2 の解法であるビタビ (Viterbi) アルゴリズムは 5.3.3 節で解説します。5.3.4 節で、問題 3 の解法である Baum-Welch アルゴリズムを説明します。

### 5.3 確率計算法

モデル  $\lambda$  が観測系列  $\mathbf{O} = \mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T$  を生成する確率を計算します。もっとも直接的な方法は、長さ  $T$  の全ての可能な状態遷移系列を枚挙し、各状態系列に関する確率を累積することです。その中の 1 つの状態系列を  $\mathbf{Q} = q_1 q_2 \cdots q_T$  とします。状態系列  $\mathbf{Q}$  に対する観測系列  $\mathbf{O}$  の確率は、観測シンボルの独立性を仮定すると、

$$P(\mathbf{O}|\mathbf{Q}, \lambda) = \prod_{t=1}^T P(\mathbf{o}_t|q_t, \lambda)$$

と表わすことができます。HMM の定義より、

$$P(\mathbf{O}|\mathbf{Q}, \lambda) = b_{q_1}(\mathbf{o}_1) b_{q_2}(\mathbf{o}_2) \cdots b_{q_T}(\mathbf{o}_T) \quad (5.1)$$

となります。状態系列  $\mathbf{Q}$  の確率は、

$$P(\mathbf{Q}|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T} \quad (5.2)$$

となります。 $\mathbf{O}$  と  $\mathbf{Q}$  の同時確率は、式 5.1 と式 5.2 の積です。すなわち、

$$P(\mathbf{Q}, \mathbf{O}|\lambda) = P(\mathbf{O}|\mathbf{Q}, \lambda) P(\mathbf{Q}|\lambda)$$

---

<sup>1</sup>segmentation. 連続音声を局所的にまとまりのある小単位に分割すること。小単位が属するカテゴリを認定することを含むのが一般的。

です。したがって、全ての可能な状態遷移系列  $\mathbf{Q} = q_1 \cdots q_T$  についてのこの同時確率の和は、

$$\begin{aligned} P(\mathbf{O}|\lambda) &= \sum_{\text{all } \mathbf{Q}} P(\mathbf{O}|\mathbf{Q}, \lambda) P(\mathbf{Q}|\lambda) \\ &= \sum_{\text{all } \mathbf{Q}} \pi_{q_1} b_{q_1}(\mathbf{o}_1) a_{q_1 q_2} b_{q_2}(\mathbf{o}_2) \cdots a_{q_{T-1} q_T} b_{q_T}(\mathbf{o}_T) \end{aligned} \quad (5.3)$$

となります。

$P(\mathbf{O}|\lambda)$  の計算に要する演算量を見積もってみましょう。各時刻  $t$  において到達可能な状態は  $N$  あるため、可能な状態系列は  $N^T$  種類存在します。各状態系列の確率は  $2T - 1$  回の乗算を行い、 $N^T$  種類の確率の和を求めるために  $N^T - 1$  回の加算を要します。したがって、式 5.3 に必要な演算回数は、乗算が  $N^T(2T - 1)$  回、加算が  $N^T - 1$  です。

実際に、演算回数がどれほどのものになるか計算してみます。状態数  $N = 3$ 、観測系列長  $T = 50$  の場合、乗算回数は  $3^{50} \times (2 \times 50 - 1) = 7.17897988 \times 10^{23} \times 99 \approx 7.1 \times 10^{25}$ 、加算は  $3^{50} - 1 = 7.17897988 \times 10^{23} - 1 \approx 7.2 \times 10^{23}$  となります。この演算量は現実離れしています。音声認識で HMM を利用するためには、 $P(\mathbf{O}|\lambda)$  の効率的な算法が必要です。

### 5.3.1 前向きパス (Forward) アルゴリズム

前向き (forward) 変数  $\alpha_t(i)$  を定義します。

$$\alpha_t(i) \triangleq P(\mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_t, q_t = S_i | \lambda). \quad (5.4)$$

$\alpha_t(i)$  はモデル  $\lambda$  が与えられたとき、時刻  $t$  における状態が  $S_i$  である場合の部分観測系列  $\mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_t$  の確率を表わしています。 $\alpha_t(i)$  は帰納的に計算することができます。

#### 1) 初期化

$$\alpha_1(i) = \pi_i b_i(\mathbf{o}_1), \quad 1 \leq i \leq N. \quad (5.5)$$

#### 2) 帰納

$$\alpha_{t+1}(j) = \left[ \sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(\mathbf{o}_{t+1}), \quad 1 \leq t \leq T-1, \quad 1 \leq j \leq N. \quad (5.6)$$

#### 3) 停止

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^N \alpha_T(i). \quad (5.7)$$

まず、前向き確率  $\alpha_1(i)$  を、初期状態確率  $\pi_i$  と最初の観測シンボル  $\mathbf{o}_1$  の同時確率として初期化します（式 5.5）。時刻  $t + 1$  において状態  $S_j$  には、時刻  $t$  における  $N$  個の状態  $S_i$ ,  $1 \leq i \leq N$  から遷移が可能です（図 5.2 (a)）。前向き確率  $\alpha_t(i)$  は  $\mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_t$  が観測され、かつ状態が  $S_i$  にあるという同時事象の確率

なので、積  $\alpha_i(t)a_{ij}$  は  $o_1 o_2 \cdots o_t$  が観測され、かつ時刻  $t$  での状態  $S_i$  から時刻  $t+1$  で状態  $S_j$  に遷移したという同時事象の確率ということになります。この積を時刻  $t$  における可能な状態  $S_i$ ,  $1 \leq i \leq N$  について足し合わせれば、時刻  $t$  までの系列  $o_1 o_2 \cdots o_t$  が観測されて時刻  $t+1$  に状態  $S_j$  にある確率が求まります。この確率に状態  $S_j$  においてシンボル  $o_{t+1}$  を観測する確率  $b_j(o_{t+1})$  を掛けることにより、 $\alpha_{t+1}(j)$  を得ます（式 5.6）。この計算は、時刻  $t$  において全ての状態  $S_j$ ,  $1 \leq j \leq N$  について行い、全ての  $t = 1, 2, \dots, T$  で繰り返します。そして、時刻  $t = T$  での前向き確率の和  $\sum_i^N \alpha_T(i)$  として所望の  $P(O|\lambda)$  が得られます。

$$\begin{aligned}\sum_i^N \alpha_T(i) &= \sum_i^N P(o_1 o_2 \cdots o_T, q_T = S_i | \lambda) \\ &= P(o_1 o_2 \cdots o_T | \lambda) \\ &= P(O | \lambda)\end{aligned}$$

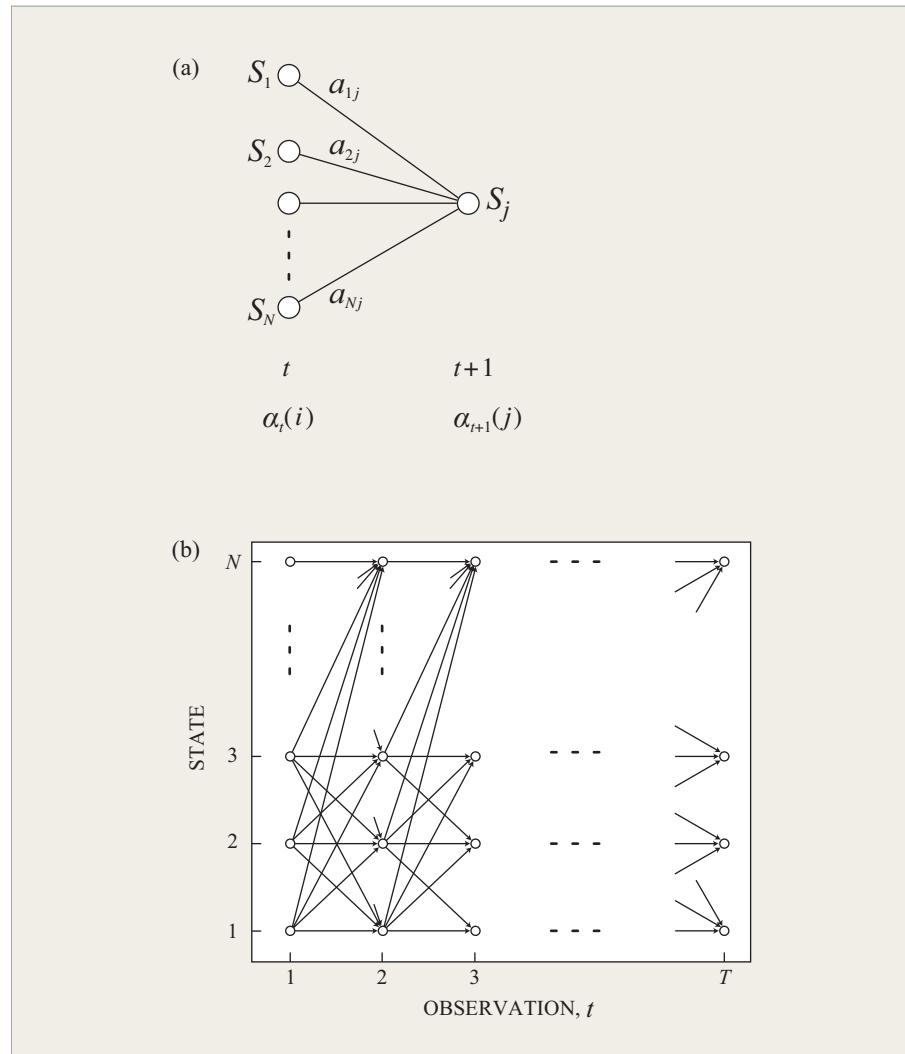


図 5.2: 前向きパス (Forward) アルゴリズム. (a) 帰納計算, (b) トレリス.

計算量の見積りは、乗算が  $N(N+1)(T-1) + N$  回、加算が  $N(N-1)(T-1)$  回となります。5.3 節と同じ例 ( $N = 3$ ,  $T = 50$ ) を用いて演算量を計算して

みると、乗算回数は  $3 \times (3 + 1) \times (50 - 1) + 3 = 3 \times 2 \times 49 = 591$ , 加算は  $3 \times (3 - 1) \times (50 - 1) = 3 \times 2 \times 49 = 294$  となります。直接計算の場合と比べて、総演算量が 23 枠も削減されています！

前向き確率計算は、図 5.2 (b) のようなトレリス (trellis, 格子) 上で実行されていると考えると理解しやすいかもしれません。各時刻  $t$  (横軸上の点) において  $N$  個の状態しかないので、観測系列の長さに関係なく、その時刻までの可能な全ての状態系列が  $N$  個のいずれかの状態 (縦軸上の点) に合流します。時刻  $t = 1$  において計算が必要なのは  $\alpha_1(i)$ ,  $1 \leq i \leq N$ , 時刻  $t = 2, \dots, T$  において計算が必要なのは  $\alpha_t(j)$ ,  $1 \leq j \leq N$  であり、それぞれは直前の  $N$  個の値  $\alpha_{t-1}(i)$ ,  $1 \leq i \leq N$  を用いれば求めることができます。このことが計算の大幅な効率化に役立っています。

### 5.3.2 後ろ向きパス (Backward) アルゴリズム

前向き変数と同様に、後ろ向き (backward) 変数  $\beta_t(i)$  を定義します。

$$\beta_t(i) \triangleq P(\mathbf{o}_{t+1} \mathbf{o}_{t+2} \cdots \mathbf{o}_T | q_t = S_i, \lambda).$$

#### 1) 初期化

$$\beta_T(i) = 1, \quad 1 \leq i \leq N. \quad (5.8)$$

#### 2) 帰納

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j), \quad t = T-1, T-2, \dots, 1, \quad 1 \leq i \leq N. \quad (5.9)$$

#### 3) 停止

$$P(\mathbf{O} | \lambda) = \sum_{i=1}^N \pi_i b_i(\mathbf{o}_1) \beta_1(i). \quad (5.10)$$

初期化では、時刻  $T$  で全ての  $i$  について  $\beta_T(i) = 1$  と設定します。帰納計算においては、時刻  $t$  で状態  $S_i$  にあり、かつ時刻  $t+1$  以降の観測系列に対応する  $\beta_t(i)$  を計算するため、時刻  $t+1$  で可能な全ての状態  $S_j$  を考慮します（図 5.3）。すなわち、 $S_i$  から  $S_j$  への遷移確率  $a_{ij}$ ,  $S_j$  でシンボル  $\mathbf{o}_{t+1}$  を観測する確率  $b_j(\mathbf{o}_{t+1})$ , および状態  $S_j$  以降の後ろ向き変数値  $\beta_{t+1}(j)$  の積の  $1 \leq j \leq N$  についての和を計算します。時刻  $t = 1$  についての計算が済めば、アルゴリズムは終了します。Forward アルゴリズムと同様、Backward アルゴリズムの計算量は  $O(N^2 T)$  であり、トレリス上で計算が実行されます。

### 5.3.3 最適状態系列推定 (Viterbi アルゴリズム)

観測系列  $\mathbf{O} = \mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_T$  に対する、モデル  $\lambda$  の最適状態遷移系列  $\mathbf{Q} = q_1 q_2 \cdots q_T$  を求める問題の解法として、ビタビ (Viterbi) アルゴリズムを説明し

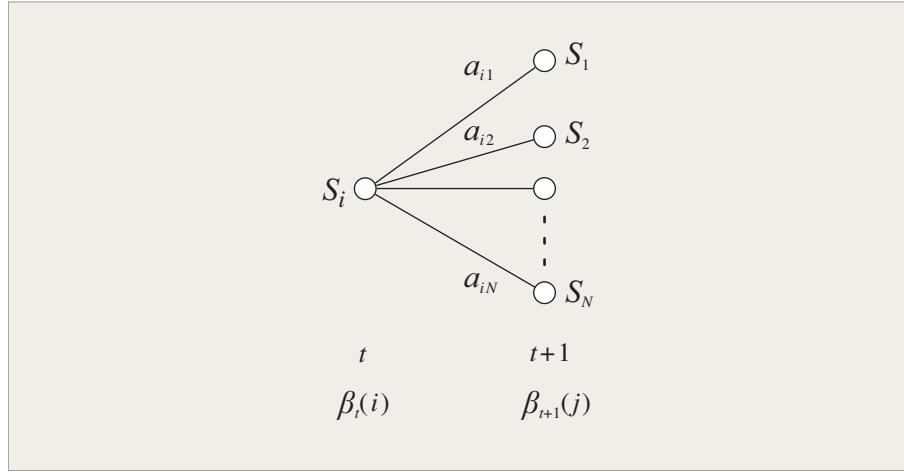


図 5.3: 後ろ向きパス (Backward) アルゴリズム

ます。まず、モデル  $\lambda$  について、観測系列  $\mathbf{O}$  が与えられたとき、時刻  $t$  において状態  $S_i$  にいる確率

$$\gamma_t(i) \triangleq P(q_t = S_i | \mathbf{O}, \lambda) \quad (5.11)$$

を定義します。前向き確率  $\alpha_t(i)$  は時刻  $t$  において状態  $S_i$  にあり部分観測系列  $\mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_t$  を観測した確率、後ろ向き確率  $\beta_t(i)$  は時刻  $t$  において状態  $S_i$  にあり部分観測系列  $\mathbf{o}_{t+1} \mathbf{o}_{t+2} \cdots \mathbf{o}_T$  を観測する確率なので、式 5.11 は、 $\alpha_t(i)$  と  $\beta_t(i)$  を用いて書き表すことができます。すなわち、

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathbf{O}|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^N \alpha_t(i)\beta_t(i)}.$$

分母の  $P(\mathbf{O}|\lambda)$  は変数  $\gamma_t(i)$  を確率尺度にするためのものです。したがって、

$$\sum_{i=1}^N \gamma_t(i) = 1.$$

変数  $\gamma_t(i)$  を用いることにより、時刻  $t$  においてもっともやうる状態  $q_t$  を

$$q_t = \operatorname{argmax}_{1 \leq i \leq N} \gamma_t(i), \quad 1 \leq t \leq T$$

と得ることができます。しかし、この基準で各時刻毎に求めた  $q_t$  を並べても、全体として妥当な状態系列  $\mathbf{Q} = q_1 q_2 \cdots q_T$  になるとは限りません。例えば、 $a_{ij} = 0$  となる  $i$  と  $j$  が存在する場合には、実際にあり得ない状態系列になってしまう可能性があります。

このような可能性を避けるため、動的計画法 (Dynamic Programming: DP) に基づいた Viterbi アルゴリズムが考案されました。与えられた観測系列  $\mathbf{O} = \mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_T$  に対して单一の最良状態系列  $\mathbf{Q} = q_1 q_2 \cdots q_T$  を求めるため、

$$\delta_t(i) \triangleq \max_{q_1 q_2 \cdots q_{t-1}} P(q_1 q_2 \cdots q_t = i, \mathbf{o}_1 \mathbf{o}_2 \cdots \mathbf{o}_t | \lambda)$$

を定義します。変数  $\delta_t(i)$  は  $t = 1, \dots, T$  までのシンボルを観測して状態  $S_i$  に至る状態系列のうち最も良い（最大の確率を与える）状態系列の確率です。 $\delta_t(i)$  を

基にして、1 時刻先の  $\delta_{t+1}$  は

$$\delta_{t+1}(i) = \left( \max_i \delta_t(i) a_{ij} \right) b_i(\mathbf{o}_{t+1}) \quad (5.12)$$

と計算することができます。最適状態系列を得るためにには、各時刻  $t$  の各状態  $j$  における式 5.12 を満たす状態遷移元  $i$  を記録しておくことが必要です。この目的のために配列  $\psi_t(j)$  を用意します。以下に、Viterbi アルゴリズムの手順を示します。

### 1) 初期化

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(\mathbf{o}_1), \quad 1 \leq i \leq N. \\ \psi_1(i) &= 0. \end{aligned}$$

### 2) 帰納

$$\begin{aligned} \delta_t(j) &= \max_{1 \leq i \leq N} (\delta_{t-1}(i) a_{ij}) b_j(\mathbf{o}_t), \quad 2 \leq t \leq T, \quad 1 \leq j \leq N. \\ \psi_t(j) &= \operatorname{argmax}_{1 \leq i \leq N} (\delta_{t-1}(i) a_{ij}), \quad 2 \leq t \leq T, \quad 1 \leq j \leq N. \end{aligned}$$

### 3) 停止

$$\begin{aligned} P^*(\mathbf{O}|\lambda) &= \max_{1 \leq i \leq N} \delta_T(i). \\ q_T^* &= \operatorname{argmax}_{1 \leq i \leq N} \delta_T(i). \end{aligned} \quad (5.13)$$

### 4) 経路のバックトラッキング

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \dots, 1.$$

Viterbi アルゴリズムは、Forward アルゴリズムと類似しています。Forward アルゴリズムと異なるのは、直前の状態に関する和 ( $\sum$ ) が最大値 (max) に置き換わっている点です。Viterbi アルゴリズムも Forward アルゴリズムと同様、トレス上上で効率的に実行することができます。

### 5.3.4 パラメータ推定 (Baum-Welch アルゴリズム)

観測系列  $\mathbf{O}$  を生成する確率が最大になるようなモデル入のパラメータ  $(\mathbf{A}, \mathbf{B}, \pi)$  を求める問題には、解析的な解が知られていません。このような問題に対しては、まず、決定すべきパラメータに適当な初期値を与え、繰り返し計算しながらパラメータの値を少しづつ修正し、最終的に局所最適解を得る手法が用いられます。Baum-Welch アルゴリズムはそのような手法の 1 つです。

まず、変数  $\xi_t(i, j)$  を定義します。これは、モデル  $\lambda$  と観測系列  $\mathbf{O}$  が与えられたとき、時刻  $t$  で状態  $S_i$  にあり、かつ時刻  $t+1$  で状態  $S_j$  にある確率を表わすものとします。すなわち、

$$\xi_t(i, j) \triangleq P(q_t = S_i, q_{t+1} = S_j | \mathbf{O}, \lambda). \quad (5.14)$$

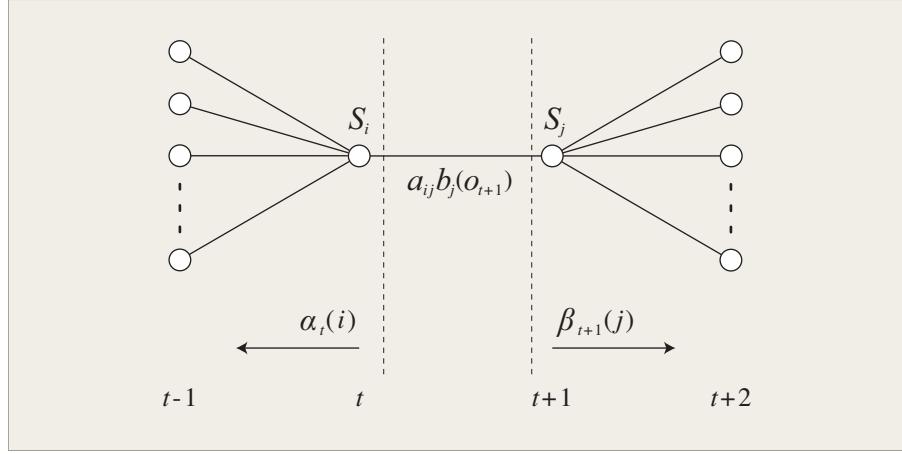


図 5.4: システムが時刻  $t$  で状態  $S_i$  にあり、かつ時刻  $t+1$  で状態  $S_j$  にある同時確率を計算する。

式 5.14 の状況を図示すると、図 5.4 のようになります。前向き確率  $\alpha_t(i)$  と後ろ向き確率  $\beta_t(i)$  の定義より、 $\xi_t(i, j)$  は

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}{P(\mathbf{O}|\lambda)} \quad (5.15)$$

$$= \frac{\alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}. \quad (5.16)$$

と表わすことができます。ここで、分子は  $P(q_t = S_i, q_{t+1} = S_j, \mathbf{O}|\lambda)$  であり、 $P(\mathbf{O}|\lambda)$  で割ることにより、 $P(q_t = S_i, q_{t+1} = S_j | \mathbf{O}, \lambda)$  を得ています。

5.3.3 節で定義した  $\gamma_t(i)$  は、モデル  $\lambda$  について観測系列  $\mathbf{O}$  が与えられたとき、時刻  $t$  において状態  $S_i$  にいる確率なので、 $\xi_t(i, j)$  の  $j$  についての和は  $\gamma_t(i)$  と等しくなります。すなわち、

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j). \quad (5.17)$$

$\gamma_t(i)$  の  $t$  についての和をとれば、状態  $S_i$  を通過する回数の期待値、別の見方をすれば状態  $S_i$  からの状態遷移の回数の期待値が得られます。同様に、 $\xi_t(i, j)$  の  $t$  についての和は、状態  $S_i$  から  $S_j$  への遷移回数の期待値となります。すなわち、

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{状態 } S_i \text{ からの遷移回数の期待値}, \quad (5.18)$$

$$\sum_{t=1}^{T-1} \xi_t(i, j) = \text{状態 } S_i \text{ から } S_j \text{ への遷移回数の期待値}. \quad (5.19)$$

上記の式を基に、事象計数の概念を用いると、HMM のパラメータの再推定公

式を導くことができます.  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\pi$  の再推定公式は,

$$\tilde{\pi}_i = \text{時刻 } t = 1 \text{ で状態 } S_i \text{ にいる頻度の期待値} = \gamma_1(i). \quad (5.20)$$

$$\tilde{a}_{ij} = \frac{\text{状態 } S_i \text{ から } S_j \text{ への遷移回数の期待値}}{\text{状態 } S_i \text{ からの遷移回数の期待値}} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (5.21)$$

$$\begin{aligned} \tilde{b}_j(k) &= \frac{\text{状態 } S_j \text{ でシンボル } v_k \text{ を観測する期待値}}{\text{状態 } S_j \text{ にいる回数の期待値}} \\ &= \frac{\sum_{t=1}^{T-1} \gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j) \text{ s.t. } o_t = v_k} \end{aligned} \quad (5.22)$$

と導出されます.

現在のモデルを  $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$  用いて式 5.20, 式 5.21, 式 5.22 の右辺を計算し, 左辺の値を更新されたモデル  $\tilde{\lambda} = (\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\pi})$  として用います. このようにして求めたモデル  $\tilde{\lambda}$  はモデル  $\lambda$  に比べて良いモデル, すなわち,

$$P(\mathbf{O}|\tilde{\lambda}) > P(\mathbf{O}|\lambda) \quad (5.23)$$

となることが知られています. したがって,  $\lambda$  の代わりに  $\tilde{\lambda}$  を用いて上記の手順を繰り返すことにより, 観測系列  $\mathbf{O}$  が生成される確率は単調増加します. ある限界点に達するまで計算を繰り返すことにより, HMM パラメータの最尤推定値 (の 1 つ) を得ることができます. ただし, 極大値しか得られません. 一般に最適値探索の空間は複雑で多くの極大値が存在するので, 最大値に収束するとは限りません.

## 5.4 音声認識のための HMM

### 5.4.1 一方向性 HMM

取り扱う時系列パターンの性質に応じて, 状態遷移に種々の拘束を設けることがあります. 例えば,

$$a_{ij} = 0, \quad j < i \quad (5.24)$$

という拘束を課した HMM を考えることができます. 前の状態に戻ることが許されていないので, 一方向性モデル (Bakis モデル) と呼ばれています. 一方向性 HMM は, 時間の経過とともに性質が変る音声信号のような信号をモデル化するのに適しています. 状態系列は  $S_1$  から始まり  $S_N$  で終了するので, 式 5.24 に加えて, 初期状態確率は, 通常,

$$\pi_i = \begin{cases} 0 & i \neq 1 \\ 1 & i = 1 \end{cases} \quad (5.25)$$

に設定されます.

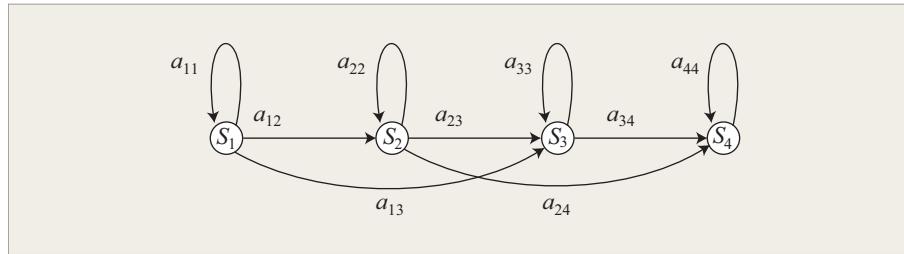


図 5.5: 一方向性 HMM. 状態数は 4. 2つ先までの状態への遷移が可能.

大幅な状態の飛び越し起きないように、状態遷移に拘束条件を加えることがあります。すなわち、

$$a_{ij} = 0, \quad j > i + \Delta \quad (5.26)$$

とします。 $\Delta$  の値を 2 と設定すると図 5.5 のような 2 つより先の状態への遷移が許されない HMM になります。したがって、この HMM の状態遷移行列は、

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix} \quad (5.27)$$

となります。当然、一方向性モデルの最終状態については、

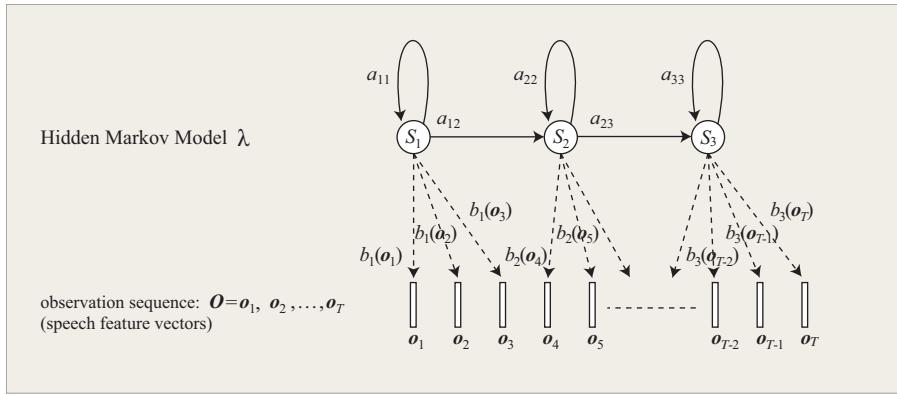
$$\begin{aligned} a_{NN} &= 1 \\ a_{Ni} &= 0, \quad i < N \end{aligned} \quad (5.28)$$

です。

一方向性 HMM を用いるときは、出力記号列  $\mathbf{O} = o_1 o_s \cdots o_T$  が観測されたとき、時刻  $T$  において最終状態に達していることが仮定されています。そうすると、 $S_T = S_N$  という情報を用いることができるので、前向きアルゴリズム、後ろ向きアルゴリズム、Baum-Welch アルゴリズムの形が少し変ります。また、最終状態に達すると、その後の出力記号列は、他の状態における遷移確率や記号の出力確率に関する情報を全く含みません。したがって、パラメータ推定を行うときは、単一の長い観測系列パターンではなく、それぞれ初期状態から出発した HMM の出力をみなせる複数の時系列パターンが必要です。複数の学習パターンを用いてパラメータ推定を行う方法については、関連図書を参照してください。

### 5.4.2 連続 HMM

音声認識で音響モデル  $P(\mathbf{O}|\lambda)$  として HMM を持つ場合、一般に、観測系列  $\mathbf{O}$  は連続値（実数ベクトル）です（図 5.6）。連続値を量子化して処理することも可能ですが、量子化処理によって情報が失われるため、性能がかなり低下する可能性があります。連続値の出力を扱う連続 HMM は、離散出力 HMM に比べて良い性能が期待できます。

図 5.6: 音声スペクトル時系列  $O$  を生成する HMM  $\lambda$ .

連続 HMM の出力を定義する確率密度関数としては、混合ガウス分布がよく用いられます。この場合、シンボル出力確率  $b_j(O)$  は、

$$b_j(\mathbf{o}_t) = \sum_{m=1}^M c_{jm} \mathcal{N}(\mathbf{o}_t, \mu_{jm}, \Sigma_{jm}), \quad 1 \leq j \leq N \quad (5.29)$$

と表わされます。ここで、 $\mathbf{o}_t$  は時刻  $t$  の観測ベクトル、 $c_{jm}$  は状態  $j$  の第  $m$  混合成分の混合係数、 $\mathcal{N}$  はガウス分布、 $\mu_{jm}$  と  $\Sigma_{jm}$  は状態  $j$  の第  $m$  混合成分の平均値ベクトルと共に分散行列です（図 5.4.2）。混合係数  $c_{jm}$  は確率としての制約

$$\sum_{m=1}^M c_{jm} = 1, \quad 1 \leq j \leq N, \quad (5.30)$$

$$c_{jm} \geq 0, \quad 1 \leq j \leq N, 1 \leq m \leq M \quad (5.31)$$

を満たします。したがって、

$$\int_{-\infty}^{\infty} b_j(x) dx = 1, \quad 1 \leq j \leq N. \quad (5.32)$$

上記の混合ガウス分布は 1 次元ですが、実際の音声認識に用いるのは多次元の

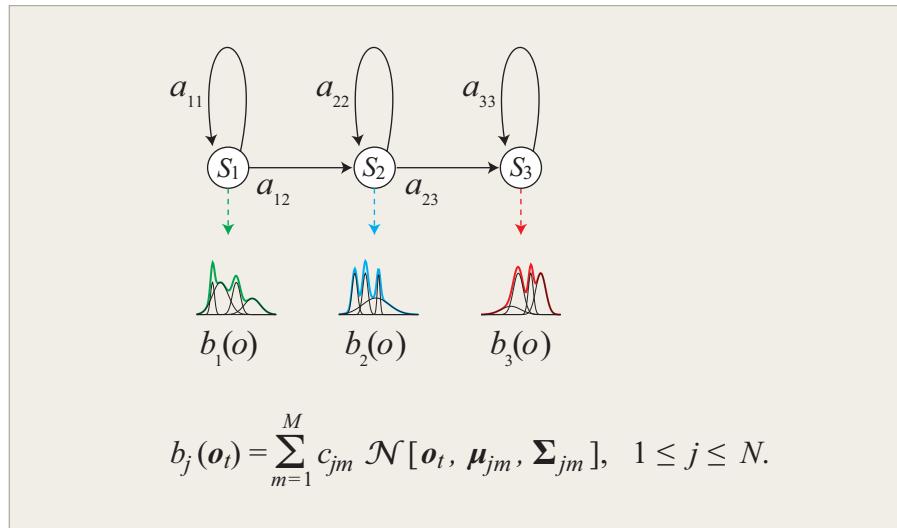


図 5.7: 出力確率密度が混合ガウス分布である連続 HMM

特徴量なので、多次元ガウス分布の確率密度関数を使います。

$$\mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_m|}} \exp \left\{ -\frac{1}{2} (\mathbf{o}_t - \boldsymbol{\mu}_m)' \boldsymbol{\Sigma}_m^{-1} (\mathbf{o}_t - \boldsymbol{\mu}_m) \right\} \quad (5.33)$$

であり、 $D$  は  $\mathbf{o}_t$  の次数、' は転置、 $M$  は混合数、 $\boldsymbol{\Sigma}_m^{-1}$  は  $\boldsymbol{\Sigma}_m$  の逆行列、 $|\boldsymbol{\Sigma}_m|$  は  $\boldsymbol{\Sigma}_m$  の行列式を表わします。多次元データを利用した場合、 $|\boldsymbol{\Sigma}_m|$  は共分散行列となります。この実験では、次元間の相関が無いと仮定した対角共分散行列を使用します。この場合、共分散行列は

$$\boldsymbol{\Sigma}_m = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ 0 & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \sigma_D^2 \end{pmatrix} \quad (5.34)$$

となり、密度関数は、

$$\mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}) = \frac{1}{\sqrt{\prod_{d=1}^D 2\pi\sigma_d^2}} \exp \left\{ -\frac{1}{2} \sum_{d=1}^D \frac{(\mathbf{o}_t^{(d)} - \boldsymbol{\mu}_m^{(d)})^2}{\sigma_d^2} \right\} \quad (5.35)$$

で与えられます。ここで、 $\mathbf{o}_t^{(d)}$  は観測ベクトル  $\mathbf{o}_t$  の第  $d$  次元、 $\boldsymbol{\mu}_m^{(d)}$  は第  $m$  混合の平均値ベクトル  $\boldsymbol{\mu}_m$  の第  $d$  次元です。

連続 HMM のパラメータ推定は、離散記号を出力する HMM に対する方法と同様の考え方で行うことができます。Baum-Welch の再推定公式を連続出力密度の HMM に対して適用することができます。この場合、

$$\tilde{c}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k)}{\sum_{t=1}^T \sum_{k=1}^M \gamma_t(j, k)}, \quad (5.36)$$

$$\tilde{\mu}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) \mathbf{o}_t}{\sum_{t=1}^T \gamma_t(j, k)}, \quad (5.37)$$

$$\tilde{\Sigma}_{jk} = \frac{\sum_{t=1}^T \gamma_t(j, k) (\mathbf{o}_t - \boldsymbol{\mu}_{jk})(\mathbf{o}_t - \boldsymbol{\mu}_{jk})'}{\sum_{t=1}^T \gamma_t(j, k)} \quad (5.38)$$

となります。ここで、 $\gamma_t(j, k)$  は時刻  $t$  において状態  $S_j$  にある場合、 $\mathbf{o}_t$  の第  $k$  番目の混合成分に対応しています。すなわち、

$$\gamma_t(j, k) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \cdot \frac{c_{jk}\mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})}{\sum_{m=1}^M c_{jm}\mathcal{N}(\mathbf{o}_t, \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})} \quad (5.39)$$

式 5.36 の  $c_{jk}$  は、状態  $S_j$  にあって第  $k$  混合成分を用いている回数の期待値と状態  $S_j$  にある回数の期待値の比となっています。同様に、平均値ベクトル  $\boldsymbol{\mu}_{jk}$  の再推定式（式 5.37）では、式 5.36 の分子を観測ベクトルで重み付けしているので、観測ベクトルのうち第  $k$  成分に帰する部分の割合の期待値になっています。共分散行列の再推定式（式 5.38）も同様の解釈で理解することができます。

### 5.4.3 スケーリング (scaling)

前向きパスアルゴリズム (5.3.1 節) や後ろ向きパスアルゴリズム (5.3.2 節) によって  $\alpha_t(i)$  や  $\beta_t(i)$  を求めるとき、パスが進むとともにこれらの値が徐々に小さくなり、コンピュータによる計算においてはアンダーフローが起こることが多いです。これを回避するための何らかの工夫が必要です。一つの方法は、前向きアルゴリズムで  $\alpha_t(i)$  ( $1 \leq j \leq N$ ) が得られるたびに

$$c_t \triangleq \left[ \sum_i \alpha_t(i) \right]^{-1} \quad (5.40)$$

を計算します。そして

$$\alpha_t(i) \triangleq c_t \alpha_t(i) \quad (5.41)$$

というスケーリング (scaling) を行います。 $\beta_t(i)$  についても、後ろ向きアルゴリズムで  $\beta_t(i)$  ( $1 \leq j \leq N$ ) が得られるたびに、上記の係数  $c_t$  を用いて

$$\beta_t(i) \triangleq c_t \beta_t(i) \quad (5.42)$$

とします。このようにスケーリングを施した  $\alpha_t(i)$ ,  $\beta_t(i)$  を用いてパラメータの更新を行う計算法については、文献 [6] を参照してください。 $P(\mathbf{O}|\lambda)$  そのものはアンダーフローのため計算できないことが多いのですが、 $\log P(\mathbf{O}|\lambda)$  は

$$\log P(\mathbf{O}|\lambda) = - \sum_{i=1}^T \log c_t \quad (5.43)$$

で求められることが知られています [9]。

スケーリングはこの実習の単語 HMM の学習プログラム `train` (6.5.4 節) でつかわれています。C 言語のソースファイル `train.c` の `scale` という配列がスケーリング係数  $c_t$  です。

## 5.5 学習から認識までの手順

HMM を使って時系列パターンの認識を行う手順をまとめます。

1. 認識対象を定め、それをいくつのカテゴリに分類するのかを決めます。
2. HMM の出力記号、状態数などを決めます。
3. カテゴリ毎に学習パターンを用意します。
4. HMM パラメータの初期値を設定します。
5. Baum-Welch アルゴリズムを用いて、カテゴリ毎に HMM のパラメータを推定します。
6. Forward アルゴリズム、Backward アルゴリズム、あるいは Viterbi アルゴリズムを用いて、未知パターンの出力確率をカテゴリ毎に計算し、出力確率が最大となるカテゴリを選んで認識結果とします。

## 5.6 実習

HMM の確率計算を計算機プログラムで実行してみましょう。状態数  $N = 4$ , 出力シンボル種類数  $M = 2$ , シンボル  $\mathcal{V} = \{v_1 = 0, v_2 = 1\}$ , 状態遷移確率は

$$\mathbf{A} = \begin{pmatrix} 0.2 & 0.3 & 0.1 & 0.4 \\ 0.1 & 0.5 & 0.1 & 0.3 \\ 0.3 & 0.4 & 0.2 & 0.1 \\ 0.2 & 0.1 & 0.4 & 0.3 \end{pmatrix}, \quad (5.44)$$

シンボル出力確率は

$$\mathbf{B} = \begin{pmatrix} 0.6 & 0.4 \\ 0.7 & 0.3 \\ 0.1 & 0.9 \\ 0.2 & 0.8 \end{pmatrix}, \quad (5.45)$$

初期状態確率が  $\pi = \{0.3, 0.4, 0.1, 0.2\}$  である図 5.8 のような HMM  $\lambda$  を考えます。

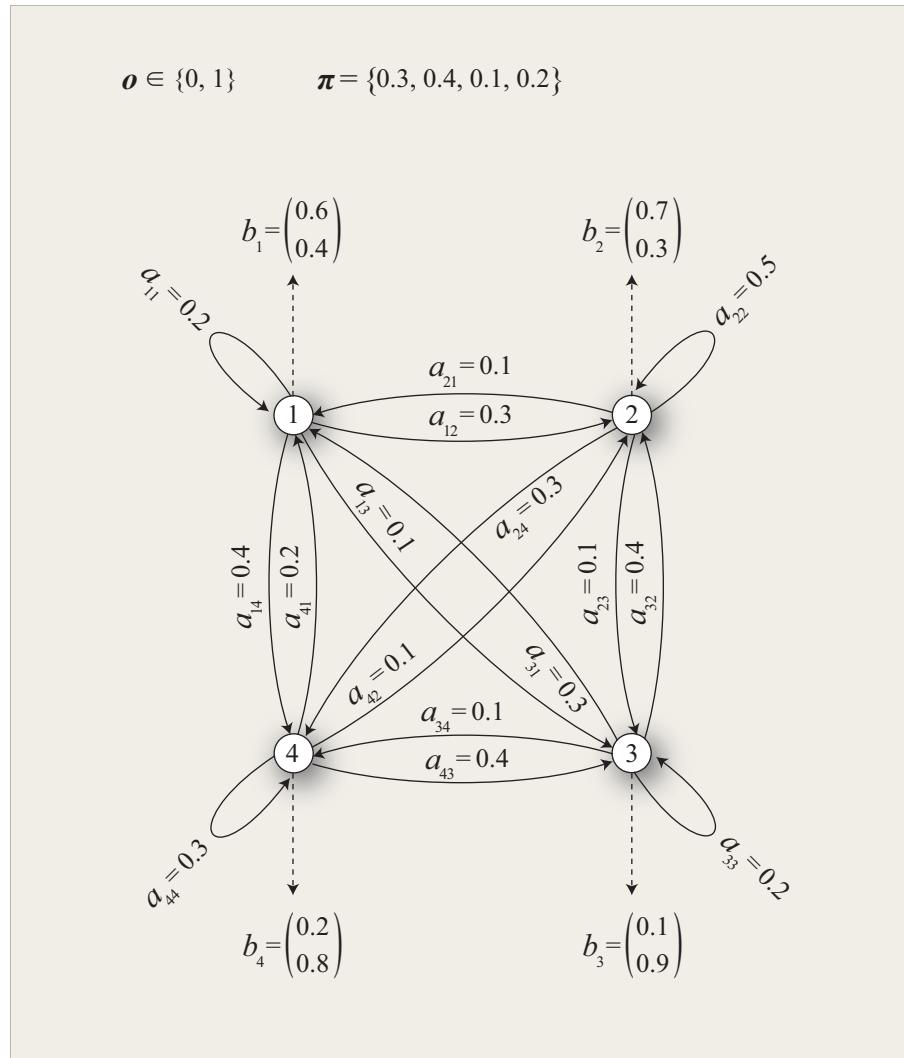


図 5.8: 4 状態離散出力 HMM

以下の説明は、[計算機実験を~/asr/drill](#) で行うことを前提としています。

### 5.6.1 Forward アルゴリズム

観測系列  $O = 0110$  に対して,  $\alpha_t(i)$  および  $P(O|\lambda)$  を Forward アルゴリズムで求めてください。

`forward.c` という C の言語のソースファイルがあります。ただし、肝心のコードの部分が空白になっています。穴埋め指示のある部分を埋めてソースコードを完成させてください。HMM の出力記号は  $\mathcal{V} = \{v_1 = 0, v_2 = 1\}$  ですが、 $v_k$  の値が配列  $O[T]$  の要素の値となっていて、シンボル出力確率の配列の添字に対応しています。C 言語の配列の添字は 0 から始まることに注意してください。完成したら、以下のコマンドを入力してコンパイルを実行します。

```
[~/asr/drill]% make drillF
```

ソースコードに C 言語の文法エラーがある場合は、その旨、メッセージが 출력されるので、エラーメッセージが出なくなるまで修正してください。文法エラーがない場合は、`drillF` というコマンドができているので、これを入力すると、`paramFB.txt` から HMM のパラメータを読み込み、forward アルゴリズムが実行され、 $\alpha_t(i)$  の値と  $P(O|\lambda)$  が表示されます。

```
[~/asr/drill]% drillF
```

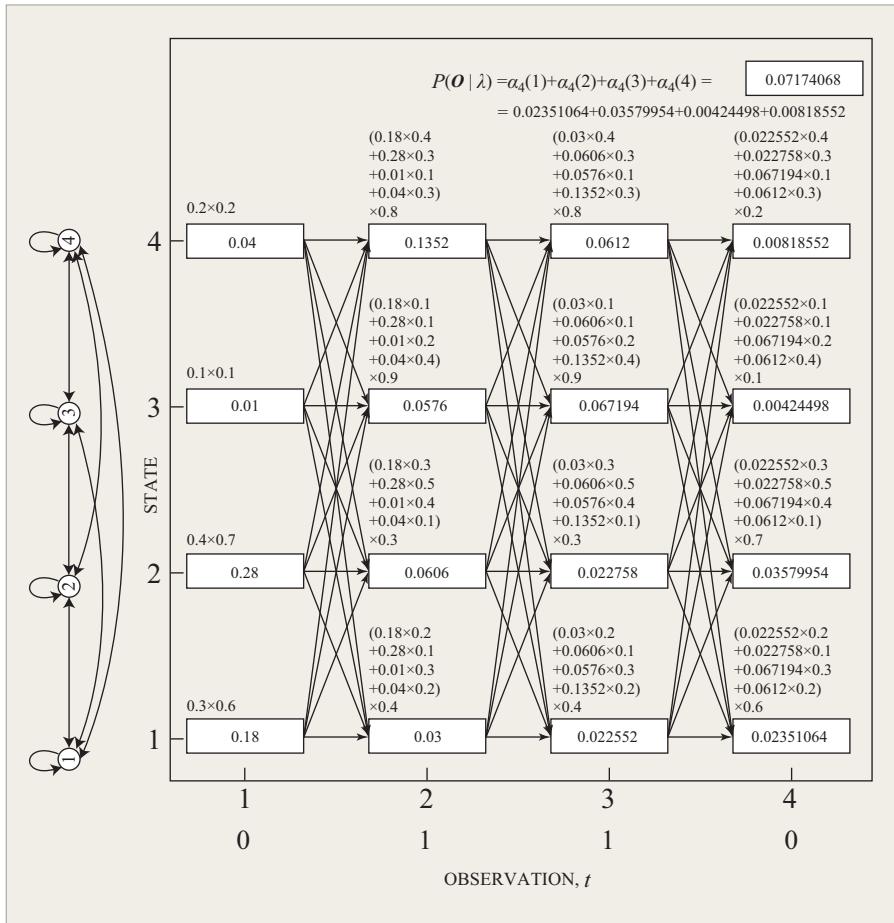


図 5.9: Forward アルゴリズム計算の過程

コマンド実行によって端末に表示される計算結果は図 5.9 と合致しているでしょうか？レポートにはソースコードの穴埋め部分、および `drillF` コマンドの実行結果を載せてください。

### 5.6.2 Backward アルゴリズム

観測系列  $O = 0110$  に対して、 $\beta_t(i)$  および  $P(O|\lambda)$  を backward アルゴリズムで求めてください。

`backward.c` という C 言語のソースファイルの一部が削除されています。穴埋め指示のある部分を埋めてソースコードを完成させてください。以下のコマンドを入力してコンパイルを実行します。

```
[~/asr/drill]% make drillB
```

ソースコードに C 言語の文法エラーがある場合は、その旨、メッセージが出力されるので、エラーメッセージが出なくなるまで修正してください。文法エラーがない場合は `drillB` というコマンドができているので、これを入力すると、`paramFB.txt` から HMM のパラメータを読み込み、Backward アルゴリズムが実行され、 $\beta_t(i)$  の値と  $P(O|\lambda)$  が表示されます。

```
[~/asr/drill]% drillB
```

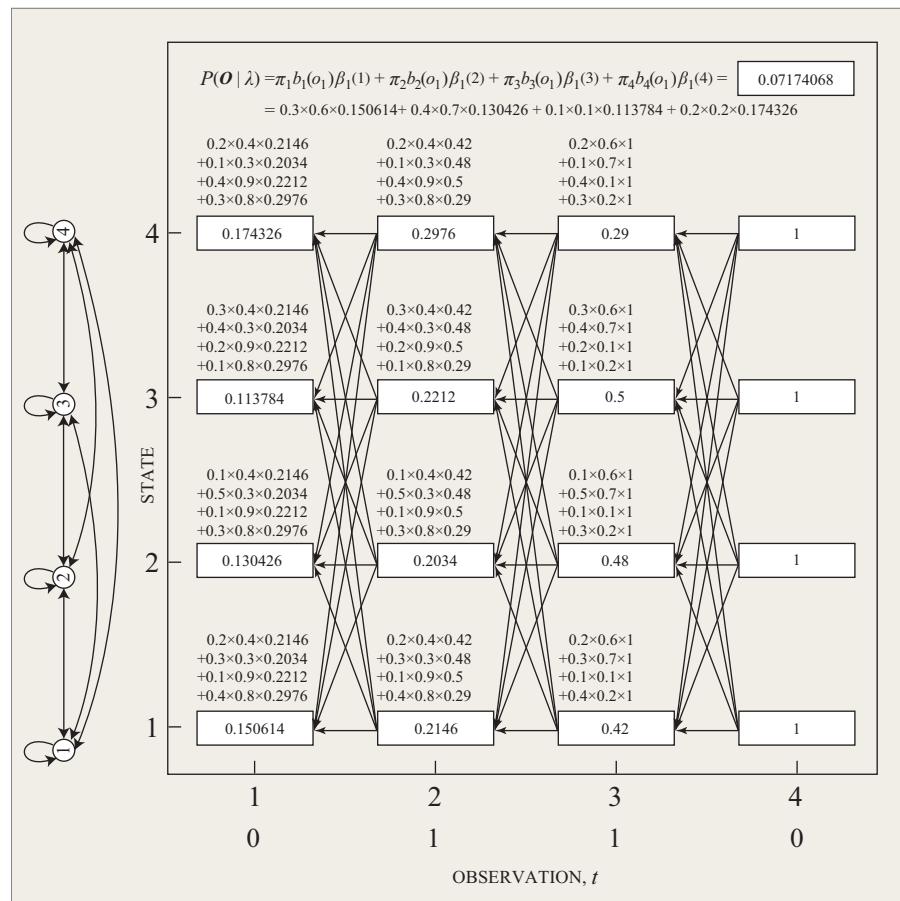


図 5.10: Backward アルゴリズム計算の過程

計算結果が図 5.10 と合致しているでしょうか？ レポートにはソースコードの穴埋め部分、および `drillB` コマンドの実行結果を載せてください。

### 5.6.3 Baum-Welch アルゴリズム

与えられた記号列(離散値)から HMM パラメータの推定値を求める Baum-Welch アルゴリズムをプログラミング言語で記述しやすい形にまとめたのが 5-19 ページの Algorithm 1 です。

#### 5.6.3.1 HMM の学習実験 [9]

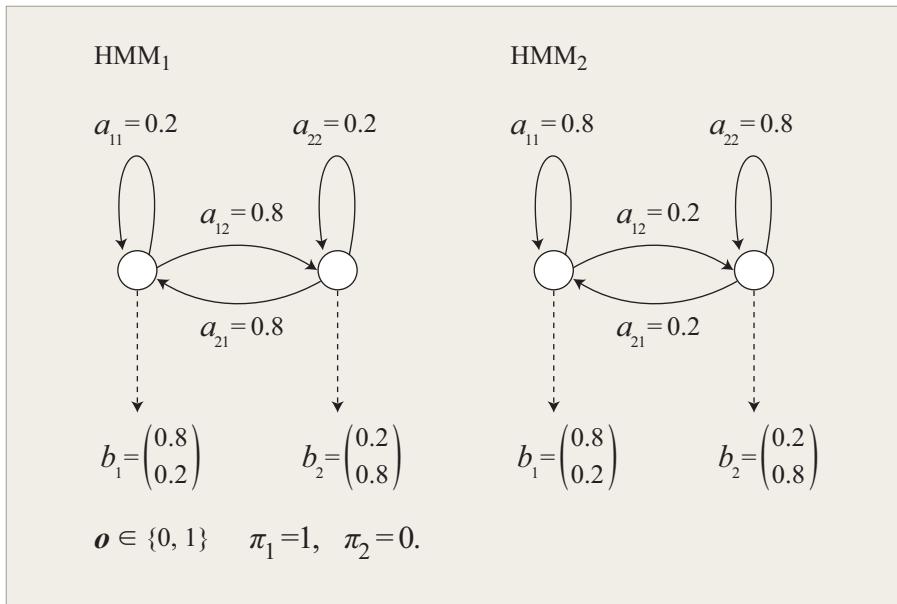


図 5.11: 2 状態 2 出力離散 HMM

状態数が 2, 出力記号が 2 種類={0,1} の異なる HMM を HMM<sub>1</sub>, HMM<sub>2</sub> とします(図 5.11). それぞれのパラメータ  $\lambda_1$  と  $\lambda_2$  を

$$\lambda_1 = (\Pi_1, \mathbf{A}_1, \mathbf{B}_1), \quad \Pi_1 = (1.0, 0.0), \quad \mathbf{A}_1 = \begin{pmatrix} 0.2 & 0.8 \\ 0.8 & 0.2 \end{pmatrix}, \quad \mathbf{B}_1 = \begin{pmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{pmatrix}$$

$$\lambda_2 = (\Pi_2, \mathbf{A}_2, \mathbf{B}_2), \quad \Pi_2 = (1.0, 0.0), \quad \mathbf{A}_2 = \begin{pmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{pmatrix}, \quad \mathbf{B}_2 = \begin{pmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{pmatrix}$$

と設定します。HMM による記号生成のシミュレーションプログラムを用いて、HMM<sub>1</sub>, HMM<sub>2</sub> からそれぞれ長さ 1000 の記号列  $\boldsymbol{O}_1 = o_1(1), o_1(2), \dots, o_1(1000)$ ,  $\boldsymbol{O}_2 = o_2(1), o_2(2), \dots, o_2(1000)$  を生成します。そして、Baum-Welch アルゴリズムを用いて、記号列  $\boldsymbol{O}_1$  からパラメータ  $\lambda_1$ , 記号列  $\boldsymbol{O}_2$  からパラメータ  $\lambda_2$  を推定する実験を行います。パラメータの初期値は、

$$\Pi_i = (1.0, 0.0), \quad \mathbf{A}_i = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}, \quad \mathbf{B}_i = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}, \quad (i = 1, 2)$$

**Algorithm 1** Baum-Welch Algorithm

---

```

1: Baum-Welch Algorithm
2:  $\bar{\lambda} = (\bar{\pi}_1, \dots, \bar{\pi}_N, \bar{a}_{1,1}, \dots, \bar{a}_{N,N}, \bar{b}_{1,1}, \dots, \bar{b}_{N,M})$ : パラメータ初期値 [入力]
3:  $\mathbf{o}$ : 出力記号 [入力],  $o(1), o(2), \dots, o(T)$ : 記号列 [入力]
4:  $\epsilon$ : 収束判定用閾値 [入力]
5:  $\bar{\lambda}$ : パラメータ推定値 [出力]
6:  $P(\lambda^{old}) \leftarrow -\infty$ : 負の十分大きな値
7: START:  $\lambda \leftarrow \bar{\lambda}$ : パラメータの更新
8: Forward アルゴリズムにより  $\lambda$  の値から  $\alpha_t(i)$  ( $1 \leq t \leq T, 1 \leq i \leq N$ ) と  $P(\lambda)$  を
   計算する。
9: if  $\log P(\lambda) - \log P(\lambda^{old}) < \epsilon$  then
10:    $\bar{\lambda} \leftarrow \lambda$ : 推定結果
11:   exit
12: else
13:    $P(\lambda^{old}) \leftarrow P(\lambda)$ 
14: end if
15: Backward アルゴリズムにより  $\lambda$  の値から  $\beta_t(i)$  ( $1 \leq t \leq T, 1 \leq i \leq N$ ) と  $P(\lambda)$  を
   計算する。
16: for  $i = 1$  to  $N$  do
17:    $\bar{\pi}_i = \frac{\alpha_1(i)\beta_1(i)}{P(\lambda)}$ 
18: end for
19: for  $i = 1$  to  $N$  do
20:   for  $j = 1$  to  $N$  do
21:      $\bar{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \alpha_t(i)a_{i,j}b_j(o(t+1))\beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(j)}$ 
22:   end for
23: end for
24: for  $j = 1$  to  $N$  do
25:   for  $k = 1$  to  $M$  do
26:      $\bar{b}_{j,k} = \frac{\sum_{t=1}^T \delta_{v(k),o(t)}\alpha_t(j)\beta_t(j)}{\sum_{t=1}^T \alpha_t(j)\beta_t(j)}$ 
27:   end for
28: end for
29:  $\bar{\lambda} = (\bar{\pi}_1, \dots, \bar{\pi}_N, \bar{a}_{1,1}, \dots, \bar{a}_{N,N}, \bar{b}_{1,1}, \dots, \bar{b}_{N,M})$ 
30: goto START
31: end of Baum-Welch Algorithm

```

---

と設定します。

Baum-Welch アルゴリズムを実行したところ,  $HMM_1$ ,  $HMM_2$  について, 繰り返し回数 100 回ほどでそれぞれのパラメータの推定値,

$$\tilde{\lambda}_1 = (\tilde{\Pi}_1, \tilde{\mathbf{A}}_1, \tilde{\mathbf{B}}_1), \quad \tilde{\Pi}_1 = (1.0, 0.0), \quad \tilde{\mathbf{A}}_1 = \begin{pmatrix} 0.23 & 0.81 \\ 0.79 & 0.23 \end{pmatrix}, \quad \tilde{\mathbf{B}}_1 = \begin{pmatrix} 0.74 & 0.27 \\ 0.27 & 0.74 \end{pmatrix}$$

$$\tilde{\lambda}_2 = (\tilde{\Pi}_2, \tilde{A}_2, \tilde{B}_2), \quad \tilde{\Pi}_2 = (1.0, 0.0), \quad \tilde{A}_2 = \begin{pmatrix} 0.77 & 0.23 \\ 0.18 & 0.82 \end{pmatrix}, \quad \tilde{B}_2 = \begin{pmatrix} 0.82 & 0.18 \\ 0.18 & 0.82 \end{pmatrix}$$

が得られました。

### 5.6.3.2 HMM による認識実験 [9]

HMM<sub>1</sub>, HMM<sub>2</sub> を用いて、長さ 50 の出力記号列をそれぞれ 100 パターンずつ生成し、推定されたパラメータの値  $\tilde{\lambda}_1$ ,  $\tilde{\lambda}_2$  を用いてこれらのパターンを認識する実験を行います。HMM<sub>1</sub> によって生成されるパターンをカテゴリ 1 に属するパターン、HMM<sub>2</sub> によって生成されるパターンをカテゴリ 2 に属するパターンと呼ぶことにします。

認識するためには、Forward アルゴリズムを用い、各パターン  $O$  に対して  $P(O|\tilde{\lambda}_1)$  と  $P(O|\tilde{\lambda}_2)$  を計算します。そして、 $P(O|\tilde{\lambda}_1) \geq P(O|\tilde{\lambda}_2)$  ならばそのパターンはカテゴリ 1 に、 $P(O|\tilde{\lambda}_1) < P(O|\tilde{\lambda}_2)$  ならばそのパターンはカテゴリ 2 に属していると判定します。

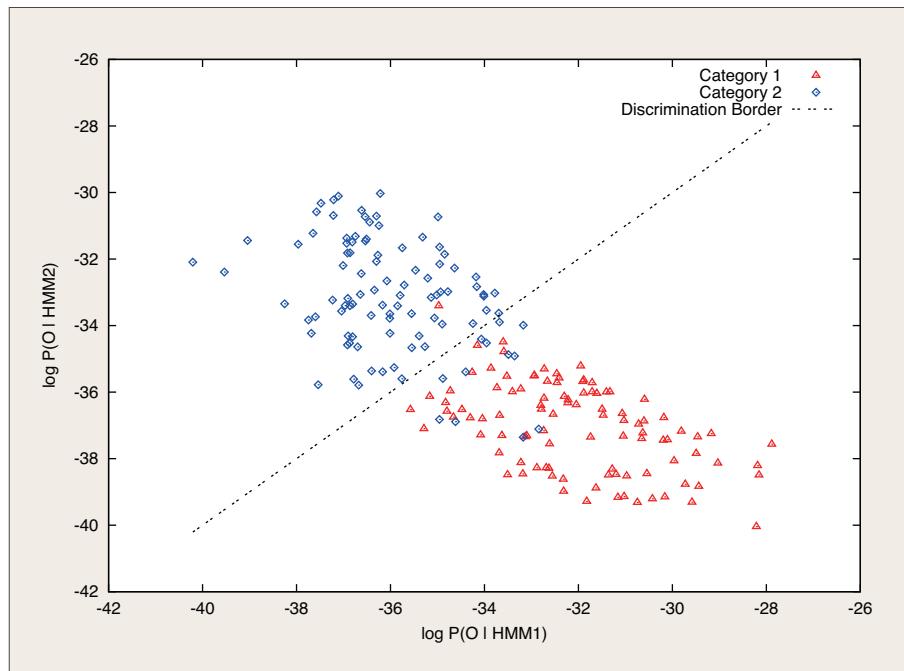


図 5.12: 尤度計算結果。カテゴリ 1 のパターン横軸は  $\log P(O|\tilde{\lambda}_1)$ 、縦軸は  $\log P(O|\tilde{\lambda}_2)$ 。△はカテゴリ 1 のパターン、◊はカテゴリ 2 のパターン、直線(点線)は識別境界、すなわち  $\log P(O|\tilde{\lambda}_1) = \log P(O|\tilde{\lambda}_2)$  を満たす領域です。

図 5.12 はカテゴリ 1 とカテゴリ 2 に属するパターンについての認識プログラムの計算結果を図示したものです。横軸は  $\log P(O|\tilde{\lambda}_1)$ 、縦軸は  $\log P(O|\tilde{\lambda}_2)$  を示しています。点線で示しているのは  $\log P(O|\tilde{\lambda}_1) = \log P(O|\tilde{\lambda}_2)$  なる識別境界です。したがって、カテゴリ 1 のパターン (△) のうち点線より下にあるパターンは正しく認識され、上にあるパターンは誤って認識されることになります。この例の場合、100 パターンのうち正しく認識されたパターンは 99 個です。カテゴリ 2 に属するパターン (◊) については、点線より上にあるパターンは正しく認

識され、下にあるパターンは誤って認識されることになります。この例の場合、100 パターンのうち正しく認識されたパターンは 88 個です。カテゴリ 1 とカテゴリ 2 の平均認識率は 93.5%となりました。

### 5.6.3.3 実習手順

- (1) Baum-Welch アルゴリズムのコーディング `baumwelch.c` という C 言語のソースファイルがあります。Baum-Welch アルゴリズムのソースコード `baumwelch.c` が完成したら、以下のコマンドを入力してコンパイルを実行します。

```
[~/asr/drill]% make drillT
```

ソースコードに C 言語の文法エラーがある場合は、その旨メッセージが出力されるので、エラーメッセージが出なくなるまで修正してください。文法エラーがない場合は、`drillT` というコマンドができます。

- (2) 学習データの生成 記号列生成のために `gen` というプログラムが用意されています。まず、

```
[~/asr/drill]% make gen
```

と入力してコンパイルしてください。HMM<sub>1</sub> から 1000 個の記号を生成するには、

```
[~/asr/drill]% gen 1 1000 > d1.txt
```

と入力します。 $O_1$  のサンプルが 1 行 1 個のテキスト形式で出力されるので、リダイレクションを利用して `d1.txt` に保存しています。このプログラムは実行された時刻が異なると異なる系列を生成します。したがって、このデータを用いた実験結果はこのテキストの例とは若干異なることがあります。同様に、

```
[~/asr/drill]% gen 2 1000 > d2.txt
```

と入力して  $O_2$  のサンプルを 1000 個生成し、`d2.txt` に保存しています。

- (3) HMM パラメータの推定 生成した  $O_1$  と  $O_2$  を用いて、HMM のパラメータを推定します。まず、HMM<sub>1</sub> のパラメータ推定を行いましょう。次のように入力します。

```
[~/asr/drill]% drillT d1.txt paramHMM1.txt > d1.log
```

Baum-Welch アルゴリズムにより、パラメータの推定が繰り返されます。収束条件が満たされると、学習データ `d1.txt` を用いて学習された HMM のパラメータは端末に表示されるとともに、`paramHMM1.txt` に保存されます。ファイル `d1.log` には、繰り返し計算の毎回ごとの暫定の HMM パラメータの値が記録されます。同様に HMM<sub>2</sub> の学習も行います。

```
[~/asr/drill]% drillT d2.txt paramHMM2.txt > d2.log
```

(4) 学習過程のグラフ化 この実習の HMM の初期パラメータは全て 0.5 ですが, Baum-Welch アルゴリズムによって再推定を繰り返すことにより, 学習データの信号の生成源のパラメータに単調に近づいて行きます. その様子をグラフに描いてみましょう. 次の 2 つのコマンドは, [d1.log](#) あるいは [d2.log](#) に記録された  $a_{11}$  と  $a_{12}$  の再推定ごとの値を曲線として描く gnuplot スクリプトです.

```
[~/asr/drill]% drawLC_d1
```

```
[~/asr/drill]% drawLC_d2
```

コマンドを実行すると画面にグラフが表示され, その後に端末でリターンキーを押すと表示が消え, 表示されたグラフが [LC\\_a11-a12\\_HMM1.png](#) と [LC\\_a11-a12\\_HMM2.png](#) というファイルに保存されます. 実際にパラメータ推定を行ってみたら,  $HMM_1$  は 107 回,  $HMM_2$  は 125 回で再推定が収束しました. その様子を上記コマンドでグラフにしたのが図 5.13 です. 再推定回数や収束曲線の形状は学習データによって異なりますが, およそこの事例のように真のパラメータ値に漸近する変化をします.

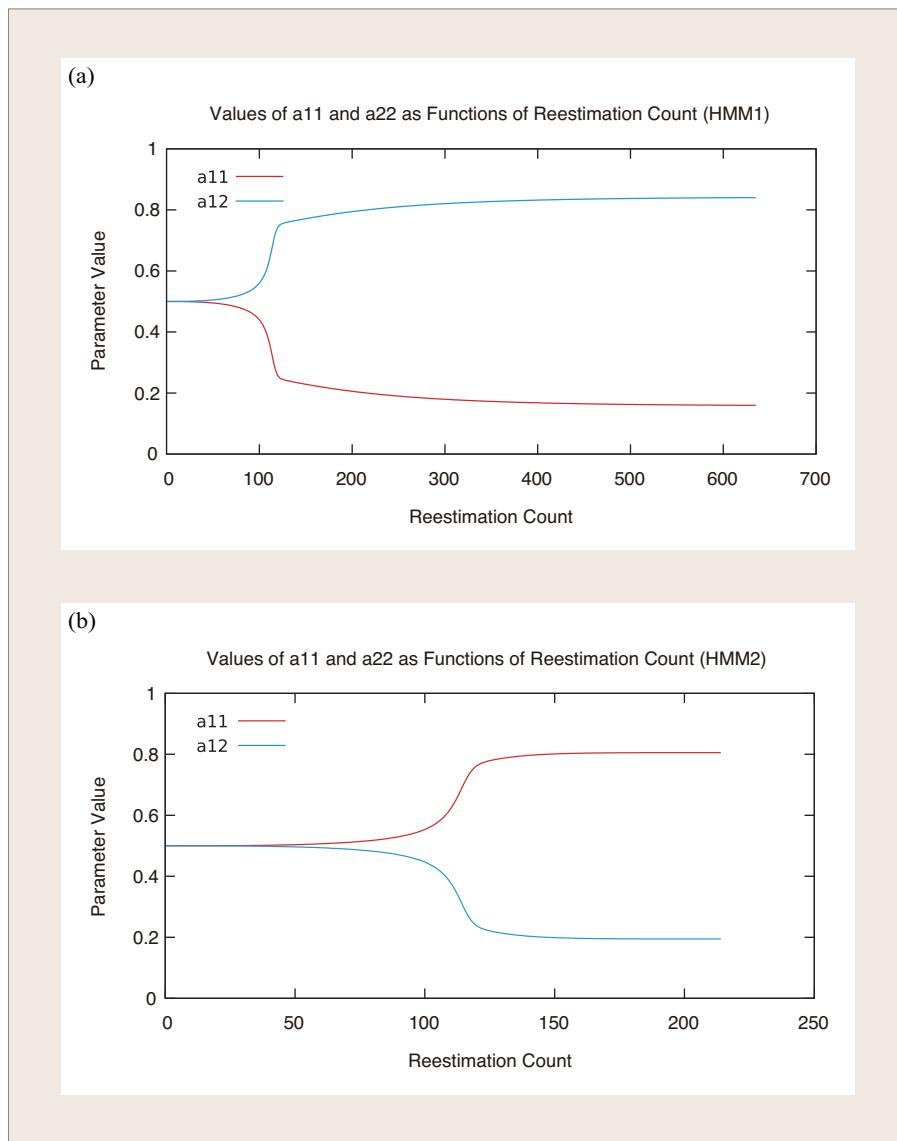


図 5.13: (a)HMM<sub>1</sub> と (b)HMM<sub>2</sub> の状態遷移確率  $a_{11}$  と  $a_{12}$  が再推定の繰り返しにより信号源のパラメータに近づく様子. `drawLC_d1` と `drawLC_d2` の出力を説明用に曲線の色などを調整したもの. 他のパラメータも 0.2 あるいは 0.8 に漸近する同じ軌跡を描く. 再推定回数や収束曲線の形状は必ずしもこれと類似でない場合がある.

## (5) 学習済みモデルの評価

### (5.1) 評価データの生成

HMM<sub>1</sub>, HMM<sub>2</sub> から長さ 50 の出力記号列をそれぞれ 100 パターン生成します. 学習データの生成に用いたプログラム `gen` を再び使います. プログラムのあるディレクトリにある `data1` と `data2` というサブディレクトリに, HMM<sub>1</sub> からのパターンと HMM<sub>2</sub> からのパターンのファイルを保存します. まず,

```
[~/asr/drill]% genData
```

上記のコマンドを実行することにより, 認識用データが生成され, そのファイル名の一覧が `data1.list` と `data2.list` に保存されます.

ファイルの行数や内容を確認してください。テキストファイルの行数を数えるには、例えば、端末に `wc -l data1.list` と入力します。

### (5.2) 認識率測定

認識用データの  $HMM_1$  と  $HMM_2$  に対する尤度を Forward アルゴリズム (穴埋めして完成した forward 関数を使います) によって計算するプログラムを作ります。

```
[~/asr/drill]% make drillR
```

と入力してコンパイルを実行します。`drillR` は、HMM のパラメータファイルと認識を指定して実行します。 $HMM_1$  からのパターンを認識させてみると次のようになりました。テキストの横幅に収めるためにコマンドを途中で改行して書いていますが、実際は1行です。

```
[~/asr/drill]% drillR paramHMM1.txt paramHMM2.txt
data1.list > data1.result
n1= 94
n2= 6
```

端末に表示される数値は  $HMM_1$  が生成したパターンとして認識されたデータの数 (`n1`) と  $HMM_2$  が生成したパターンとして認識されたデータの数 (`n2`) です。この数値はあくまでも一例です。

`data1.result` というファイルには、各パターンについて、1行に  $\log P(O|\tilde{\lambda}_1)$  と  $\log P(O|\tilde{\lambda}_2)$  が並んで記録されます。内容を確かめておいてください。`data2.list` についても、同様に認識を実行します。

```
[~/asr/drill]% drillR paramHMM1.txt paramHMM2.txt
data2.list > data2.result
n1= 17
n2= 83
```

前述のように、`gen` によって生成される記号列は `gen` が実行された時刻に依存するため、推定される HMM のパラメータの推定値は若干異なったものになります。さらに、認識データも生成時刻によって異なるので、必ずしも上記の結果と同じ結果にはなりません。

学習と認識を複数回繰り返すと、学習データと認識データの確率的揺らぎにより、HMM パラメータの推定結果は揺らぎ、認識結果も毎回揺らぎます。複数回の試行を推奨します。

認識結果の表示  $HMM_1$  からのパターンの認識結果 `data1.result` と  $HMM_2$  からのパターンの認識結果 `data2.result` ができたたら、

```
[~/asr/drill]% drawR
```

と入力します。図 5.12 のようなグラフが表示されます（ちなみに、図 5.12 のデータは上記で得られたのとは別の試行の結果です。）。`drawR` を

実行した端末で Return キーを押すと、グラフの表示は消え、`drawR.png` という PNG 形式のファイルにグラフが保存されます。`drawR` は gnuplot スクリプトです。他のファイル名や形式で保存したい場合は編集して使ってください。

#### 5.6.4 多次元ガウス確率密度関数

単語認識のための HMM の学習や認識のために必要な多次元ガウス確率密度関数のプログラミングをします。この作業は `~/asr/wrecog` で行います。C 言語のソースファイル `program/gpdf.c` があります。式 5.35 に従ってこのソースを完成させてください。実例を用いて確率密度を計算するプログラムを以下のようにして作成します。

```
[~/asr/wrecog]% make -C program drillG
```

サンプルの HMM パラメータと時系列データを用いて多次元ガウス確率密度を計算し、尤度を算出するテストプログラム `drillG` が出来ます。

プログラムが正しく出来ているか確認しましょう。端末に次のように入力して、計算確認用の HMM パラメータ `sample/sampleG.hmm` に記述されている多次元ガウス確率密度関数を用いて MFCC 時系列 `sample/sampleG.mfcc` の Viterbi 尤度  $\log P^*(\mathcal{O}|\lambda)$  (式 5.13) を計算して端末に表示します。正しくできていれば次のような計算結果の数値が得られます。

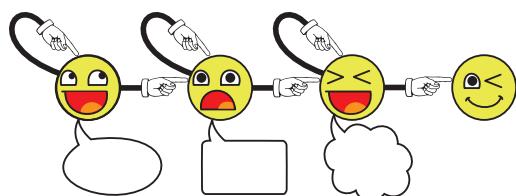
```
[~/asr/wrecog]% drillG sample/sampleG.hmm sample/sampleG.mfcc
log likelihood= -3.818172e+02
```

単語認識実験には多次元ガウス確率密度関数の計算が必要です。上記と同じ結果が出たら、次のコマンドを実行してください。単語認識実験に必要な全てのプログラムのコンパイルが実行されます。

```
[~/asr/wrecog]% make -C program all
```

### 5.6.5 レポート (第 2 週)

1. 実習課題の目的
2. HMM 算法のプログラミングとシミュレーション
  - (a) 前向きパスアルゴリズム
    - i. `forward.c` の穴埋め部分.
    - ii. `drillF` の実行結果.
  - (b) 後ろ向きパスアルゴリズム
    - i. `backward.c` の穴埋め部分.
    - ii. `drillB` の実行結果.
  - (c) Baum-Welch アルゴリズム
    - i. `baumwelch.c` の穴埋め部分.
    - ii. `drillT` による  $HMM_1$  と  $HMM_2$  のパラメータ推定結果.
    - iii. `drawLC_d?` による  $HMM_1$  と  $HMM_2$  の  $a_{11}$  と  $a_{12}$  の学習曲線.
    - iv. `drillR` による認識結果
      - A. 認識率
      - B. 尤度計算結果 (図 5.12 と同形式の `drawR` の出力).
  - (d) 多次元ガウス確率密度関数
    - i. `gpdf.c` の穴埋め部分.
    - ii. `drillG` の実行結果.
3. 考察
4. 一般事項
  - (a) 本日のポイントは何であったか？
  - (b) 良くわかったこと
  - (c) わからなかったこと
  - (d) 要望
  - (e) 感想, その他



# 第6章 単語音声認識

## 6.1 はじめに

単語を発声した音声を認識することを**孤立単語音声認識 (isolated word recognition)**といい、通常、これを**単語音声認識 (spoken word recognition)**あるいは**単語認識 (word recognition)**と呼びます。単語音声認識を応用して機械を声で操作することに使う文脈では**音声コマンド認識 (voice command recognition)**と呼ぶこともあります。

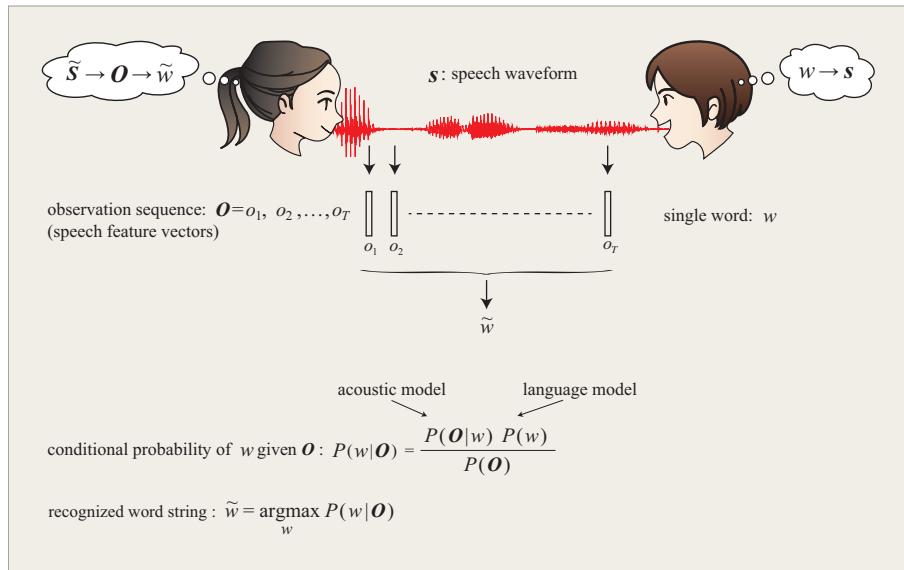


図 6.1: 単語認識の定式化。 $W$ : 単語,  $S$ : 音声波形,  $\tilde{S}$ : 聴き手が受け取った音声波形,  $O$ : 観測系列 (聞き手の受け取った音声特徴の系列),  $\tilde{W}$ : 推定単語

単語音声がスペクトル分析によって得た音声特徴ベクトル時系列 (観測系列) を

$$O = o_1, o_2, \dots, o_T \quad (6.1)$$

とします。ここで、 $o_t (t = 1, \dots, T)$  は時刻  $t$  で観測された音声特徴ベクトルです。単語音声認識は、

$$\operatorname{argmax}_i P(w_i|O), \quad w_i \in \mathcal{W} \quad (6.2)$$

を計算する問題と考えることができます。ここで、 $\mathcal{W}$  は単語の集合 (認識対象単語),  $w_i (i = 1, \dots, |\mathcal{W}|)$  は  $i$  番目の単語を表わします。式 6.2 の確率は直接計算することはできませんが、Bayes の定理を用いて 4.2 節と同様に、

$$P(w_i|O) = \frac{P(O|w_i)P(w_i)}{P(O)} \quad (6.3)$$

と変形し、この式から計算することができます。

事前確率  $P(w_i)$  が与えられた条件のもとで、単語の発話された確率の計算は  $P(\mathbf{O}|w_i)$  のみに依存します。観測系列  $\mathbf{O}$  の次元の大きさを考えると、条件付同時確率  $P(o_1, o_2, \dots, o_T|w_i)$  を単語音声のサンプルデータから直接計算するのは実際的に無理です。しかしながら、隠れマルコフモデル (Hidden Markov Model: HMM) などの単語音声生成モデルを用いれば、条件付き確率  $P(\mathbf{O}|w_i)$  の計算の問題を HMM の出力確率  $P(\mathbf{O}|\lambda)$  の計算の問題に置き換えることができます。

なお、この実習では、全ての単語は等確率で出現すると仮定します。すなわち、

$$P(w_i) \triangleq \frac{1}{|\mathcal{W}|}, \quad i = 1, \dots, |\mathcal{W}|.$$

## 6.2 単語 HMM

隠れマルコフモデル (HMM) は、5 章の演習で取り扱ったような離散記号を出力することができるだけでなく、音声スペクトルのような連続値、連続値ベクトルを出力することもできます。HMM で単語音声をモデル化するためには、出力  $b_j(o_t)$  として確率密度関数を定義します。連続 HMM の詳細は 5.4.2 節を参照してください。

単語音声は音素が決められた順に発声されたものです。たとえば、日本語の数詞の 1 は音素が/i/, /ch/, /i/の順に発声されて単語音声となります。順番を変えて/ch/, /i/, /i/と発声されると違う単語になってしまいます。このように、音の順番が単語の意味の違いに関わるので、単語をモデル化する HMM としては、状態遷移の方向に強い制約を掛けた一方向性 HMM (5.4.1 節) を用います。つまり、各状態の出力は音素（実際には音素よりも小さな音声単位）のスペクトルの確率分布であり、遷移先はその状態があるいは後の状態に制限されています。前の状態に戻らないことにより、発音の時間進行の方向が決まるのです。

実習で用いる数詞の「1」の単語音声を例に説明します（図 6.2）。単語の前後に 100ms の無音区間を含んだ波形データを HMM で学習します。波形データをスペクトル分析した結果の MFCC には、3.8 節で述べたように局所的に特徴のあるパターンが見られ、音素（波形）と対応して変化していることがわかります。ちなみに、子音/ch/は 2 種類の音が連なっている音素です。前半に音声が一旦途切れる無音部（閉鎖部、c1）があり、後半に破裂部および摩擦部（ch）があります。母音（この単語では/i/）はその区間でほぼ一定の性質を示します<sup>1</sup>。

単語 HMM の状態数は、上記のような音素の性質や MFCC のパターンの変化のしかたを考慮して決めます。図 6.2 の例では、単語 1 の HMM の状態数を 7 とっています。単語 HMM の学習では、まず、単語音声の MFCC の時系列  $\mathbf{O}$  を状態数で均等に分割し、時間順に状態に割り当てます。次に、状態に割り当てられた部分時系列について、各次元の係数の値の分布を混合正規分布で表現します（5.4.2 節）。この例では、MFCC の特徴のある各区間がほどよくそれぞれ 1 つの状態に割り当てられています。

<sup>1</sup>より正確には、前の音素からの過渡区間、定常部、後の音素への過渡区間という性質の異なる 3 つの部分からなりますが、/ch/のような子音と比べれば、定常であるといえます。この実習では個別の音素のモデル化は行わないで、このような詳細は気にしないことにします。

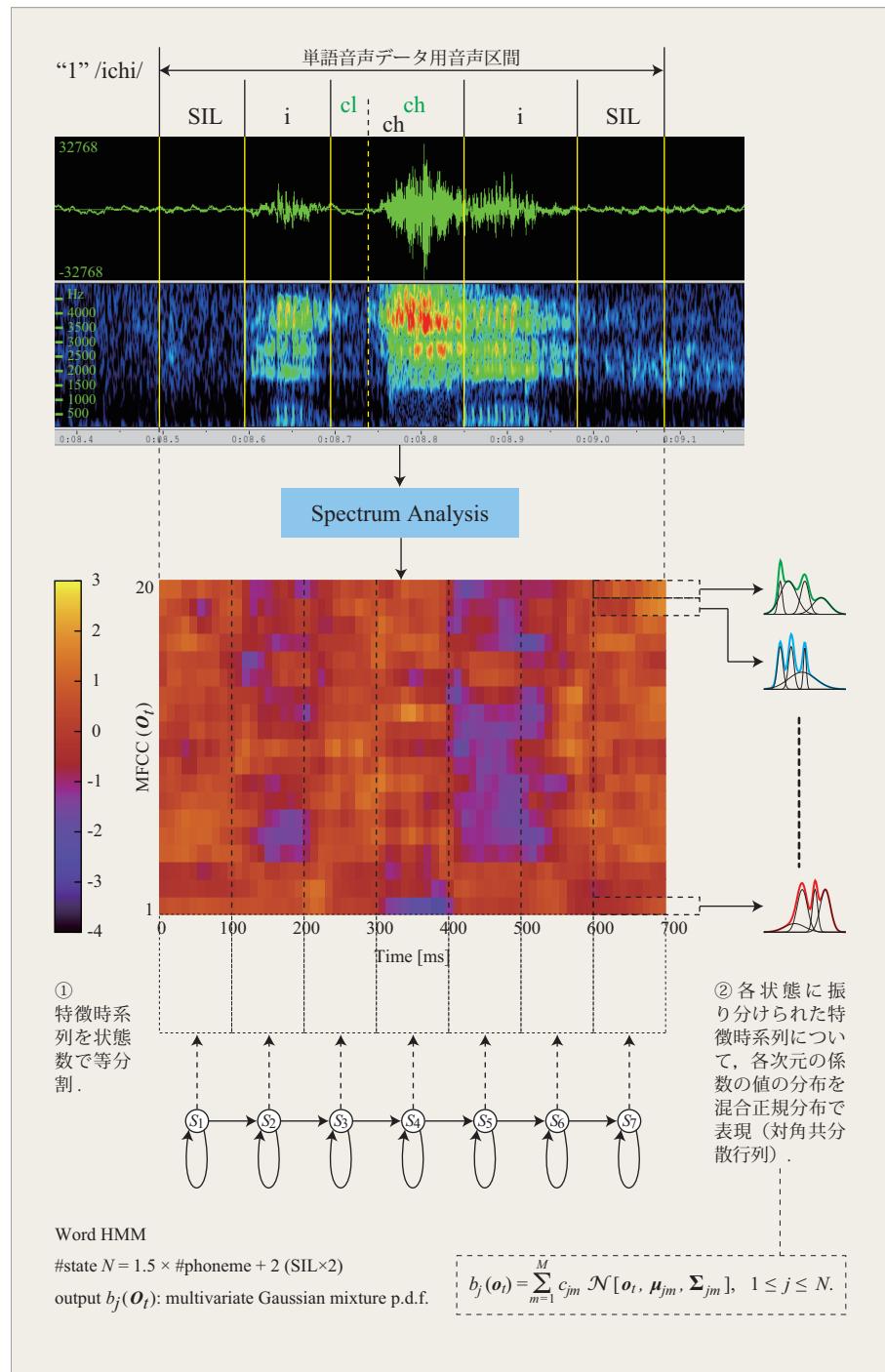


図 6.2: 単語音声スペクトルを生成する HMM. HMM の状態  $S_i$  からの出力は、単語を構成する音の部品に対応します。出力  $b_j(\mathbf{o}_t)$  は多次元実数ベクトルである MFCC の値の分布を表わす多次元混合ガウス分布確率密度関数で定義します。

## 6.3 単語音声認識実験の概要

### 6.3.1 基本構成

発話された単語音声波形をスペクトル分析して得られた特徴時系列  $\mathbf{O}$  について、単語 HMM  $\lambda_i$  の Viterbi 確率  $P^*(\mathbf{O}|\lambda_i)$  (式 5.13) を計算し、もっとも確率の高いモデル  $\lambda_i$  を選びます (図 6.3). Viterbi 確率は  $P(\mathbf{O}|\lambda_i)$  の近似値ですが、

ForwardあるいはBackward確率の計算に比べて計算量が少なく、実用上問題ないことが知られています。このため、実習ではViterbiアルゴリズムによって単語音声の認識を行っています。実用システムでは、誤った発声による誤動作を防

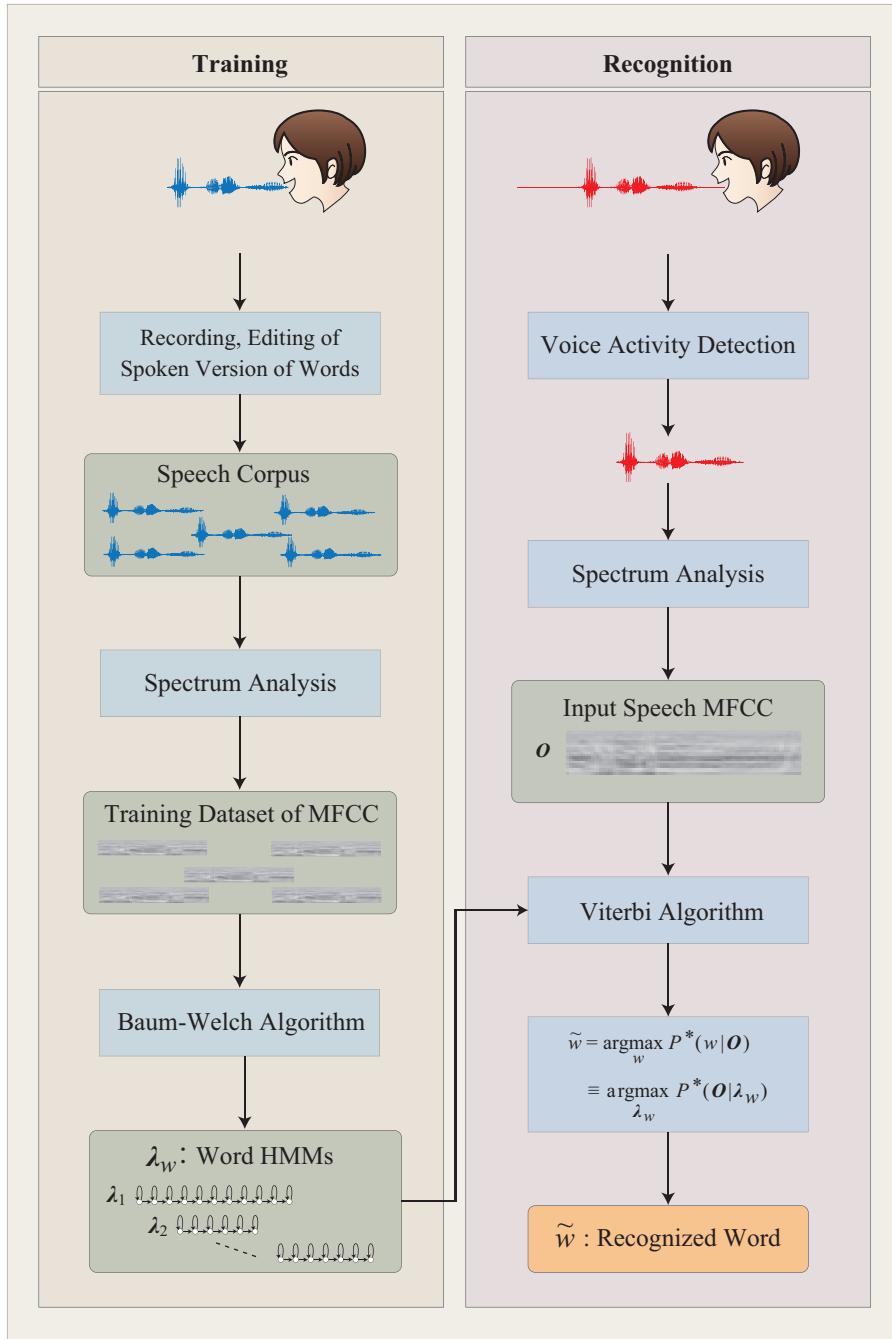


図 6.3: 単語音声認識実験の概要

ぐため、どの単語モデルも十分に高い確率値を出さない場合は、入力音声はどの単語でもないと判断し、入力を棄却するようにすることができます。しかし、本実験では、入力音声はあらかじめ登録した単語のいずれかであるとします。したがって、どんな変な音が入力されても、 $P^*(O|\lambda_i)$ を計算することができれば<sup>2</sup>、認識対象の単語のうち最も確率が高いもの  $\arg\max_i P^*(O|\lambda_i)$  が認識結果として出力されます。

<sup>2</sup>極端に短い音声が入力された場合などは、計算に失敗します。

### 6.3.2 音声区間検出

入力信号には無音の部分や音声でない信号が混ざっていることが多いので、スペクトル分析の前に、音声区間を推定する処理が必要です。この実習の On-The-Fly 単語認識（6-28 ページ）では、入力信号には無音の部分と音声の部分の 2 種類が含まれているという前提で単語認識を行います。無音といっても、マイクから入る背景騒音、マイクや信号処理回路に起因する微かな雑音が含まれているので、信号のパワーは 0 にはなりません。しかし、音声が存在する区間に比べれば、パワーはかなり小さい値を示します。そこで、本実験では信号のパワー（対数パワー）の値に基づいて、入力信号から自動的に音声区間を切り出す処理、すなわち音声区間検出（Voice Activity Detection: VAD）を行っています。

この実習では、音声波形を 20 ms 幅のセグメント（区間）に区切れます。そして、セグメント毎にその対数パワー

$$E = 10 \log_{10} \frac{1}{N} \sum_{i=1}^N x_i^2$$

を計算します。この値は音声が始まる前の無音区間では小さいですが、音声区間の始まりが近づくと急に大きくなります（図 6.4）。そこで、先頭のセグメントから順に対数パワー  $E$  の値を調べて、その値が閾値  $\theta$  を初めて超えたセグメントを A とします。次に、セグメント A の後に  $E$  が  $\theta$  を初めて下回った時の最初のセグメントを B とします。音声区間の途中でも、無声子音などの区間では対数パワーの値が閾値を下回ることがあります（図 6.4 の第 24～27 セグメント）が、通常その区間の長さは比較的短いです。このような区間を無音声と判断しないために、セグメント A の後に  $E$  が  $\theta$  を下回っても、その状態が 10 セグメント（200 ms）<sup>3</sup> 連続しなければ音声区間が終わったとは判断しません

セグメント A とセグメント B を音声区間の開始および終了セグメントとすると、音声の開始および終了部の無声子音等のパワーが低い音声部分が切り取られてしまう可能性があります。したがって、A の前 9 個目のセグメント C、および B の後 9 個目のセグメント D をそれぞれ音声区間の開始セグメント、終了セグメントとしています（図 6.4）。

<sup>3</sup> この閾値を変えたいならば、~/asr/sound/vad.c の MAXLOWSEG を変更してください。

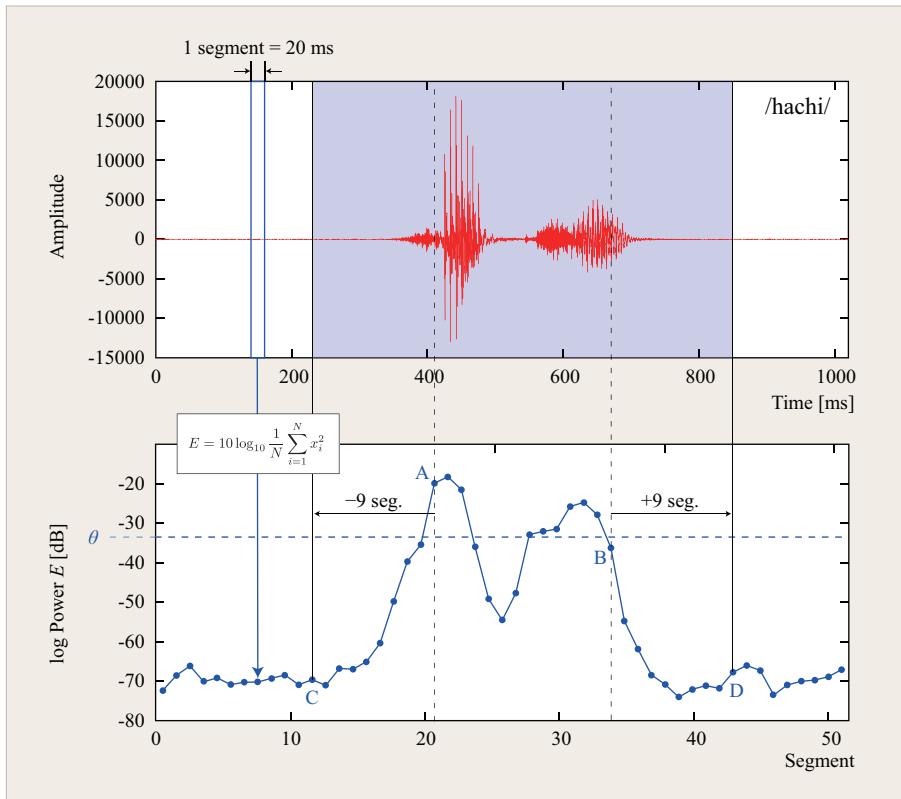


図 6.4: 音声区間検出の方式。上は音声波形 (/hachi/)。下は対数パワーの変化グラフ。音声波形を 20 ms 幅のセグメント（区間）に区切り、セグメントの対数パワー  $E = 10 \log_{10} \frac{1}{N} \sum_{i=1}^N x_i^2$  を計算します。対数パワーの値が閾値  $\theta$  を初めて超えたセグメント A を検出します。A の後に閾値を初めて連続 5 セグメント下回った時の最初のセグメントを B とします。A の前 9 個目のセグメント C、および B の後 9 個目のセグメント D をそれぞれ音声区間の開始セグメント、終了セグメントとします。

この方法は、最も基本的な方法で、計算量が少ないという利点があります。しかし、音声波形の振幅は実行環境（録音レベル、マイクと口の距離など）に大きく影響されるので、実行環境毎に閾値の調整をする必要があります。雑音が多い（SNR が低い）環境では正しく音声区間の検出ができなくなります。

## 6.4 単語 HMM の学習

単語 HMM に用いる一方向性 HMM (5.4.1 節) の場合、初期状態確率は  $\pi_1 = 1, \pi_{i>1} = 0$  に決まっているので、学習するパラメータは状態遷移確率  $A$  とシンボル出力確率  $B$  ということになります。状態数は単語の長さに応じて定めます。すなわち、短い単語に対しては状態数を少なく、長い単語に対しては状態数を多く定めます。

一方向性 HMM の学習を行うためには複数の学習用パターンが必要です。そこで、実習では多数の単語音声を録音し、スペクトル分析して MFCC パターンに変換します。そして、この多数の MFCC パターンを用いて、Baum-Welch アルゴリズム (5.3.4 節) により  $A$  と  $B$  を推定します。

Baum-Welch アルゴリズムにおいて、 $\log P(O|\tilde{\lambda})$  は単調増加します。その様子をグラフに描くと、例えば図 6.5 のようになります。 $\log P(O|\tilde{\lambda})$  の値は、学習の初期段階で急速に増加し、その後再推定を繰り返してもあまり変化しなくなることがわかります。この時点で学習が終わったと判断し学習を終了しています。

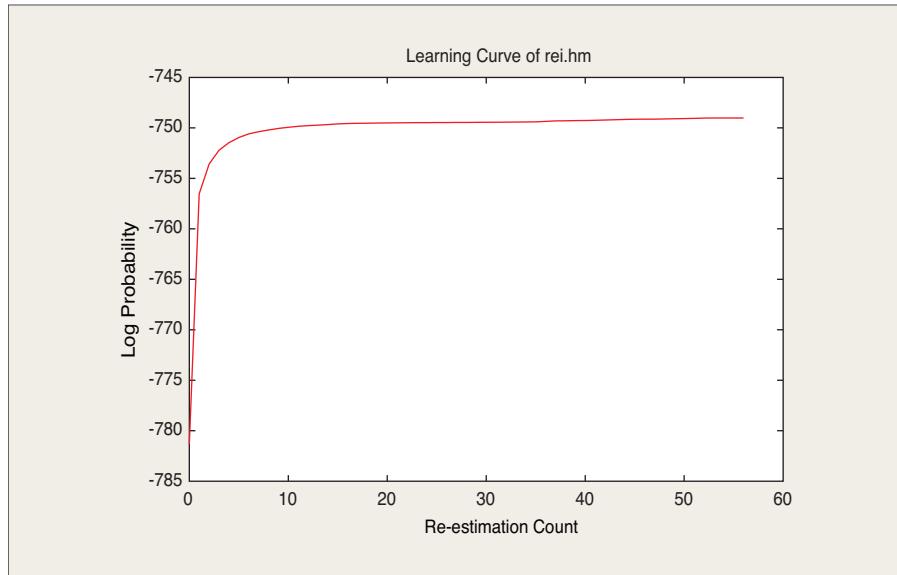


図 6.5: Baum-Welch アルゴリズムによる HMM の学習曲線。数字 0 の 50 個の学習用データ (`~takagi/mfcc/m0/rei_???.mfcc`) を用いて学習した場合。

## 6.5 実習

IED の端末にヘッドセットを接続し、自分の声で発声した単語の音声データを収録し、スペクトル分析を行って、単語 HMM の学習を行います。学習した単語 HMM を用いてヘッドセットから入力した音声を直接認識する On-The-Fly 単語音声認識を行います。

この実習では、受講者本人の音声のみを学習データとして単語 HMM（音響モデル）を学習し、最大 10 種類程度の単語を前後に空白（非音声区間）を置いて発話した音声を認識する実験を行います。音声認識では最も単純で容易なタスクです。このような音声認識タスクを特定話者小語彙孤立単語音声認識（speaker dependent small vocabulary isolated spoken word recognition）といいます。この種のタスクで実用面で有用なのは数字単語認識です。数字は種類が限られている上、各種情報の入力に役立つからです。なお、受講者本人の音声のみで音響モデルを学習するので、他の人の音声入力では認識率が低下します。

0～9 の数字の認識を行うという設定で実習手順を説明します<sup>4</sup>。音声データは`~/asr/wrecog/wav/digits.wav`、ラベルは`~/asr/wrecog/wav/digits.lab`として説明を進めます<sup>5</sup>。単語に複数の読み方がある場合は、その読み方の数だけ単語 HMM を作った方が良い結果が得られるが、ここでは簡単のため表 6.1 の 10 種類の発音とします。

表 6.1: 数字単語一覧の例

| 単語表記 | 単語名   | 発音      | 音素数 | ファイル名   |
|------|-------|---------|-----|---------|
| 0    | rei   | /rei/   | 3   | rei_*   |
| 1    | ichi  | /ichi/  | 3   | ichi_*  |
| 2    | ni    | /ni/    | 2   | ni_*    |
| 3    | saN   | /saN/   | 3   | saN_*   |
| 4    | yoN   | /yoN/   | 3   | yoN_*   |
| 5    | go    | /go/    | 2   | go_*    |
| 6    | roku  | /roku/  | 4   | roku_*  |
| 7    | nana  | /nana/  | 4   | nana_*  |
| 8    | hachi | /hachi/ | 4   | hachi_* |
| 9    | kyuu  | /kyuu/  | 3   | kyuu_*  |

HMM の学習を行うためには、同じ単語の多数の発声が必要です。**1つの単語について 10 個の学習用音声データ**を用意してください。以下の説明は、この実習を`~/asr/wrecog`で行うこと前提としています。WaveSurfer はこのディレクトリに移動してコマンドラインで起動してください。

<sup>4</sup>可能であれば、個性や創意を發揮し、認識対象単語（5 種類以上）はこのテキストの例と異なるものを考えてください。ただし、極端に短い単語や長い単語が混在しないほうが良いです。

<sup>5</sup>テキストの実例では数字 (digits) の音声データなので、このようなファイル名にしました。各自の認識対象に応じた内容を表わすファイル名を付けてください。適切なファイル名を付けるのは重要なことです。

### 6.5.1 認識用設定ファイルの作成

単語認識を行う場合、認識対象の単語の一覧表（表 6.1）に対応した設定ファイルを用意する必要があります。認識タスクに応じて、予め用意されているファイルの内容を変更し、次の 2 種類のテキスト形式のファイルを作成してください。ファイル名は変えないでください。

単語 HMM 一覧 ファイル名=[~/asr/wrecog/lib/HMMList](#)。表示文字列、単語名、HMM ファイル名の対応表です。3 カラムからなります。各カラムの区切りには半角スペースまたはタブを用います。空行があってはなりません。最後の行末には必ず改行を入れてください。

- 第 1 カラム：端末に表示される文字列（64byte まで；半角全角 OK）。
- 第 2 カラム：単語名（半角英数字 [a-zA-Z0-9]）。
- 第 3 カラム：HMM ファイル名（[~/asr/wrecog](#) 基準の相対パス名）。

```
-----  
0 rei    hmm/rei.hmm  
1 ichi   hmm/ichi.hmm  
2 ni     hmm/ni.hmm  
3 saN   hmm/saN.hmm  
4 yoN   hmm/yoN.hmm  
5 go    hmm/go.hmm  
6 roku  hmm/roku.hmm  
7 nana  hmm/nana.hmm  
8 hachi hmm/hachi.hmm  
9 kyuu  hmm/kyuu.hmm  
-----
```

単語名一覧表 ファイル名=[~/asr/wrecog/lib/wordlist](#)。単語名の一覧表

```
-----  
rei  
ichi  
ni  
saN  
yoN  
go  
roku  
nana  
hachi  
kyuu  
-----
```

単語認識プログラム (`recogf`, `countCorr`, `recog`) には必要ありませんが、  
単語 HMM の学習曲線を描くための gnuplot スクリプト `~/asr/wrecog/drawLC`  
(以下) は、単語名を自分の描く学習曲線の単語名に変更してください。

---

```
unset key
set xlabel "Re-estimation Count"
set ylabel "Log Probability"

# 以下 2 行の "rei" を自分の描く学習曲線の単語名に変更する。
set title "Learning Curve of rei.hmm"
plot "log/rei.log" using 3 with lines

pause -1
set terminal png
set out "learningCurve.png"
replot
```

---

### 6.5.2 単語音声データの作成

多くの単語発話を収録する作業は、話者に負担がかかります。一度の発話で良いデータが取れることはないので、練習も兼ねて必要な発話数よりも多く発音して、その中から良いものを選びます。この実習での「良いデータ」とは、必ずしもアナウンサーのようなハッキリしていて綺麗な発音の音声ではなく、音声認識で用いる実際の発話に近いデータのことをいいます。

#### 1. 録音

- (a) 収録の準備をします（3.10.3 節）。
- (b) 「設定」の「サウンド」パネルはデスクトップ画面に出したままにしておいてください。「サウンド」設定パネルが開いた状態でないと、音声が入力されないことがあります。入出力音量は実験の途中で適宜調節するとよいでしょう。
- (c) 単語を 12 回繰り返して発話してください。ただし、学習に用いるのはこのうちの 10 個です。一般に発話が安定しない傾向が強い最初と最後の発話を除外するのが良いと言われています。単語の間は 1 秒程度空けてください。音が最大値を越えて割れないように音量に気をつけてください。一度に全ての単語発声データ（単語種類数 × 12 回）を続けて録音してください。全ての発話の録音が終了したら、適当な名前を付けて（このテキストではファイル名を `digits.wav` として説明します。）`~/asr/wrecog/wav` ディレクトリに音声データを WAV 形式で

保存してください（保存のしかたの例は図 3.21）。保存したら、一旦このバッファを閉じてください（バッファ右端の「×」印をクリック。）

(d) 音声データの先頭から末尾に向かってラベルを入力していきます。

- i. 保存した音声ファイルを開くと「Choose Configuration」パネルが現われる所以、”HTK transcription”を選んで「OK」ボタンを押します（図 6.6）。これで、ラベルが入力できるモードになりました。

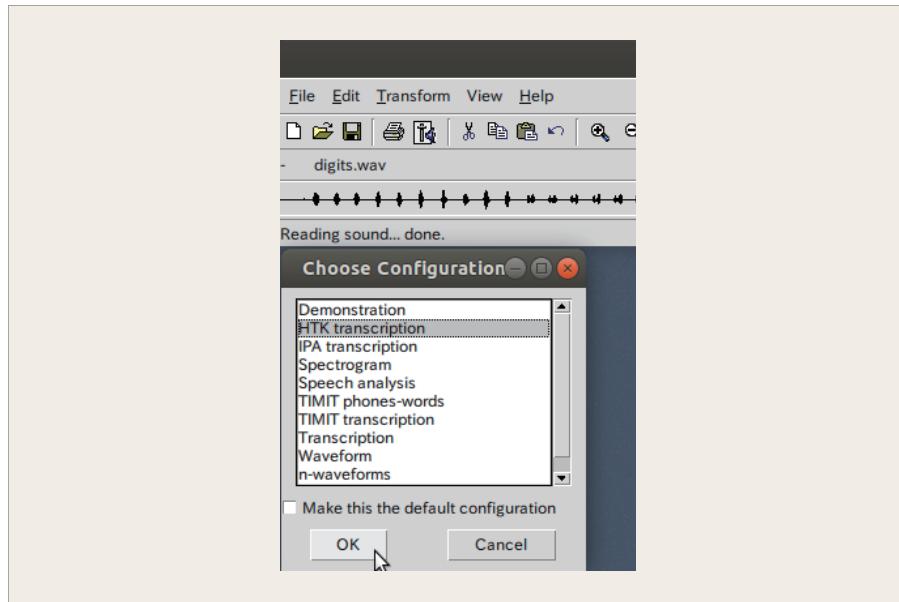


図 6.6: ラベルを入力するために”HTK transcription” モードにする。

- ii. ラベル形式の設定をします。ラベル入力領域（時間目盛「time」の下に「.lab」と表示されている初期状態で空白の欄）で右クリックして現われたメニューから「Properties...」を選択し、「Trans1」の「Label file format:」を”WaveSurfer”に設定して「Apply」ボタンを押してください（図 6.7）。この設定によりラベルファイルに記録される時間が秒単位になります。

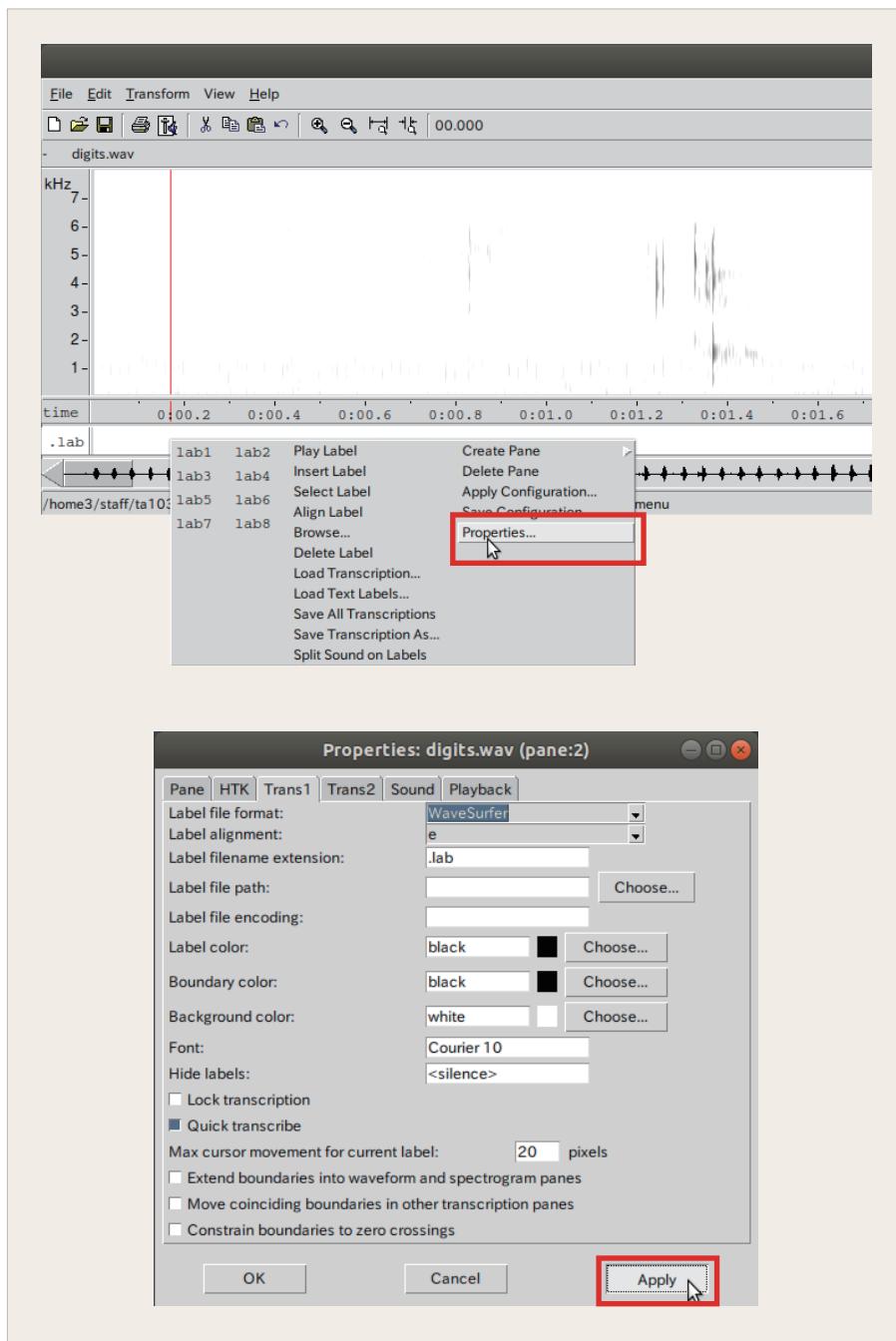


図 6.7: ラベル入力領域で右クリックし、「Properties...」を選択し、「Trans1」の「Label file format:」を”WaveSurfer”に設定して「Apply」ボタンを押す。つぎに「OK」ボタンを押してパネルを閉じる。

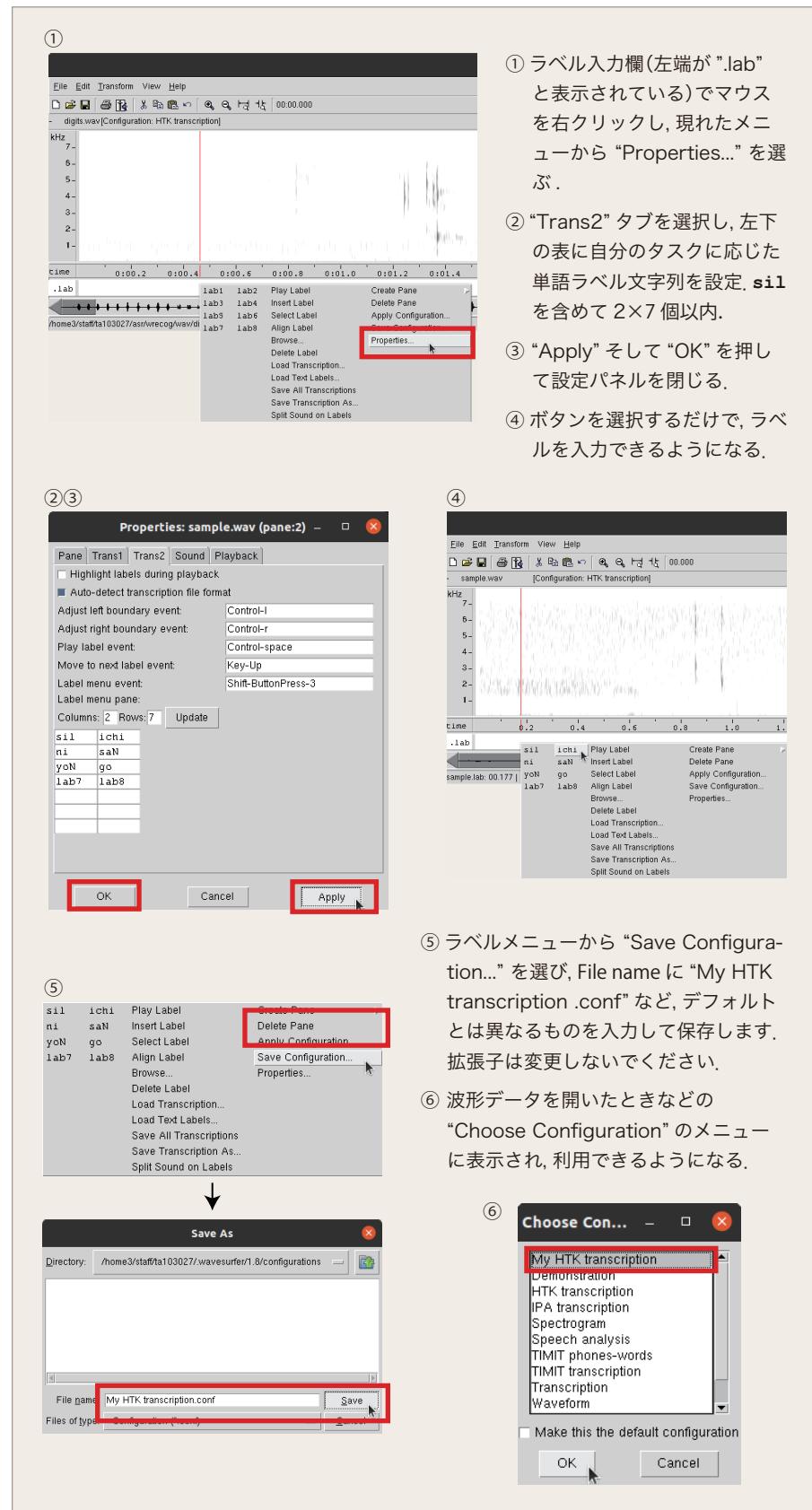


図 6.8: 自分のタスクの単語一覧にあるラベルを設定する。

iii. 単語ラベル文字列を WaveSurfer のラベルに設定します(図 6.8).

まず、ラベル入力欄(左端が”.lab”と表示されている)でマウス

を右クリックし、現れたメニューから”Properties...”を選んでください。次に、”Trans2”タブを選択し、左下の表に自分のタスクに応じた単語ラベル文字列をキーボードから入力してください。**音声（あるいは学習に使わない区間）を表す”sil”というラベルは必ず設定してください。**そして”Apply”と”OK”を押して設定パネルを閉じます。これで、ボタンを選択するだけでラベルを入力できるようになりました。最後に、ラベルメニューから”Save Configuration...”を選び、File name に、たとえば、”My HTK transcription.conf”など、デフォルトとは異なる名前を指定して入力したラベルを保存します。拡張子は変更しないでください。この設定は次に波形データを開いたときなどの”Choose Configuration”のメニューに表示され、利用できるようになります。WaveSurfer のラベルは  $2 \times 7$  が最大になっていて、これを超える数の設定はできません。バグと推測されますが、回避方法が見つかっていません。 ”sil”を含めて 14 個以内にしてください。ちなみに、ラベル入力領域に任意の英数字文字列をキーボードから直接入力することもできます。

- iv. HMM の学習に用いる発話にラベルを付けます。单語の前後に 100ms (0.1s) の余白を付けてください。横軸の表示倍率によって目盛の時間幅が異なるので、間違えないように十分注意してください。
- v. スペクトログラムを見たり、音声の一部を選択して（図 3.19）再生したりして、单語音声の始端と終端の位置を調べます。この実習で音声認識タスクの実例として説明に用いている数字音声の音響的特徴については付録 A で説明しています。ラベル付けの参考にしてください。音声ファイルの時刻 0 秒から最初の单語ラベルの開始時間（单語開始時間の 100ms 前）までの区間には”sil”というラベルを付けます。音が入っていても学習に使わない区間にも”sil”を付けます（図 6.9）。

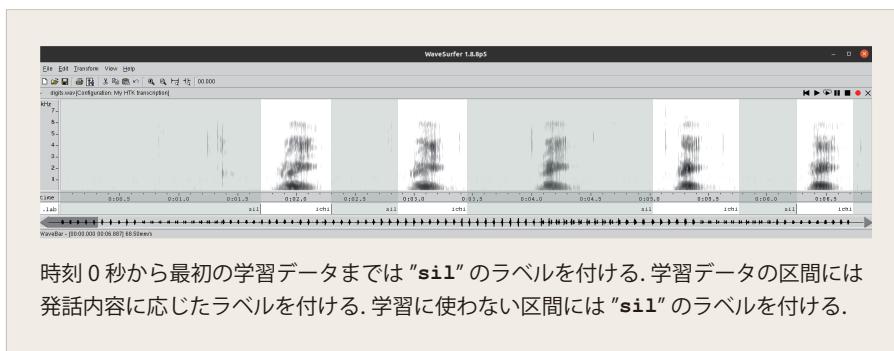


図 6.9: 時刻 0 秒から最初の学習データまでは”sil”のラベルを付ける。学習データの区間には発話内容に応じたラベルを付ける。学習に使わない区間には”sil”のラベルを付ける。

vi. 音割れ（振幅値が  $[-32768, 32767]$  に収まっていない）データは学習から除外します。念を入れて確かめたい場合は、スペクトログラム（グレースケール模様）のパネルでマウスを右クリックして押したままとし、現れたメニューの「Create Pane」を選択して現れたメニューから「Waveform」を選択すると波形が表示され、振幅目盛の値と波形表示により音割れを判断することができます（図 6.10）。



図 6.10: 音割れを念入りに調べたい場合は、Waveform パネルを表示する。音割れデータには「sil」ラベルを付与する。

vii. まず、ラベル入力領域において **単語の始端から 100ms 前の位置**にカーソルを移動して右クリックします。ポップアップメニューから入力したいラベルを選んでください（図 6.11）。ラベル情報には音声区間の開始時刻と終了時刻が必要ですが、最初のラベル挿入操作時はその境界で終了する音声区間の開始時刻が入力されていないので、WaveSurfer からエラー表示されます（図 6.12）。その音声区間の開始時刻が指定されていないという意味です。これは最初のラベルを入力するときだけであり、この後の作業への影響は無いので、「Skip Messages」を押して無視してください。

viii. この要領で、**終端から 100ms 後ろの位置**にカーソルを合わせ、全てのラベルを入力します。学習データに不適当と思われるデータは“sil”の範囲に含めます。

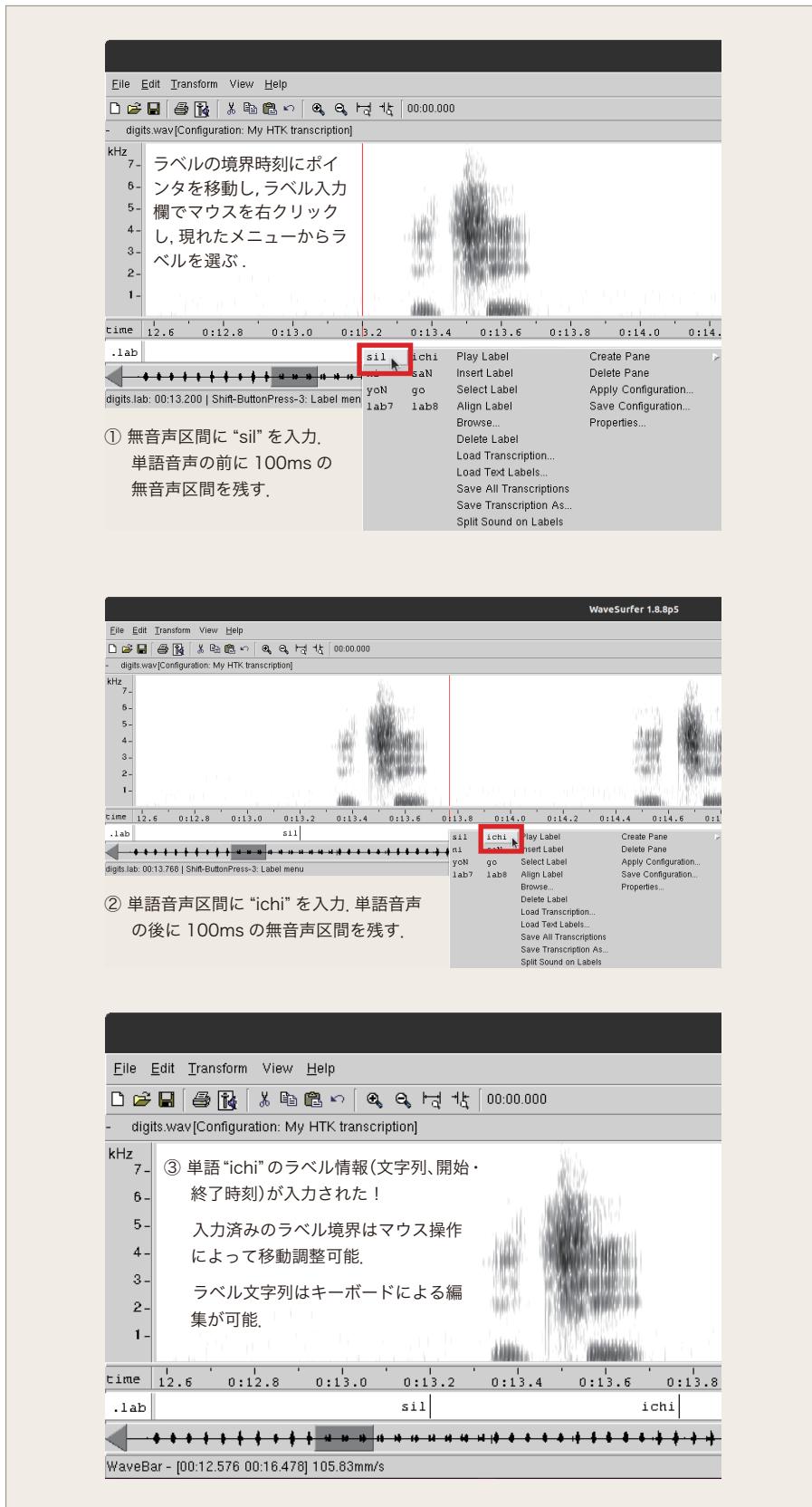


図 6.11: ラベル入力の実施例. 境界時刻位置のラベル入力欄で右クリックし、メニューのラベルを選ぶと、この位置を音声区間の終端とするラベルが挿入される。入力済みラベルの境界や文字列はマウスやキーボードの操作で変更可能。

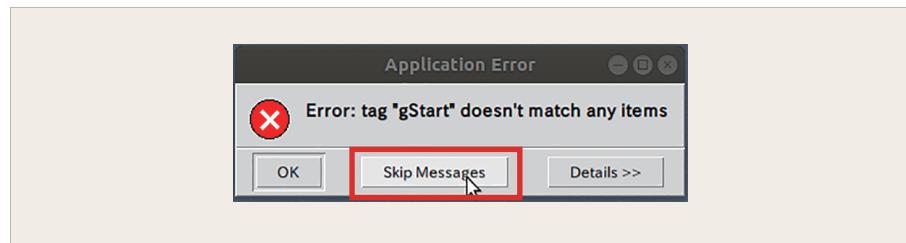


図 6.12: ラベル入力時に WaveSurfer からエラーが表示されることがあるが「Skip Messages」を押して無視し、やり直せばよい。

**区間の変更** ラベルを入力した後でも区間の境界を移動させて調節することができます。

**ラベルの音声再生** ラベル区間内で右クリックしたときに表示されるメニューで「Play Label」を選択すると、この区間の音声が再生されます。この機能を用いて、開始と終端が正しいかどうか確認することを推奨します。

**ラベルの削除** ラベル区間内で右クリックしたときに表示されるメニューで「Delete Label」を選択します。

(e) ラベルの保存は作業途中でも行うことができます(図 6.13)。作業途中 WaveSurfer を終了して中断しても、保存しておけば後で再開して作業を続けることができます。

- i. ラベル情報の最初の保存は、まず、ラベル入力欄で右クリックし、メニューから「Save Transcription As...」を選択します。
- ii. ラベルは音声波形ファイルの保存ディレクトリ~/asr/wrecog/wavに保存します。ファイル名は音声ファイル名に拡張子「lab」をつけたものとして「Save」してください。
- iii. ラベル情報の2回目以降の保存は、ファイル保存アイコンのクリックで行うことができます。既にラベルファイルがある場合は上書きされます。このとき、上書きの警告などは表示されません。

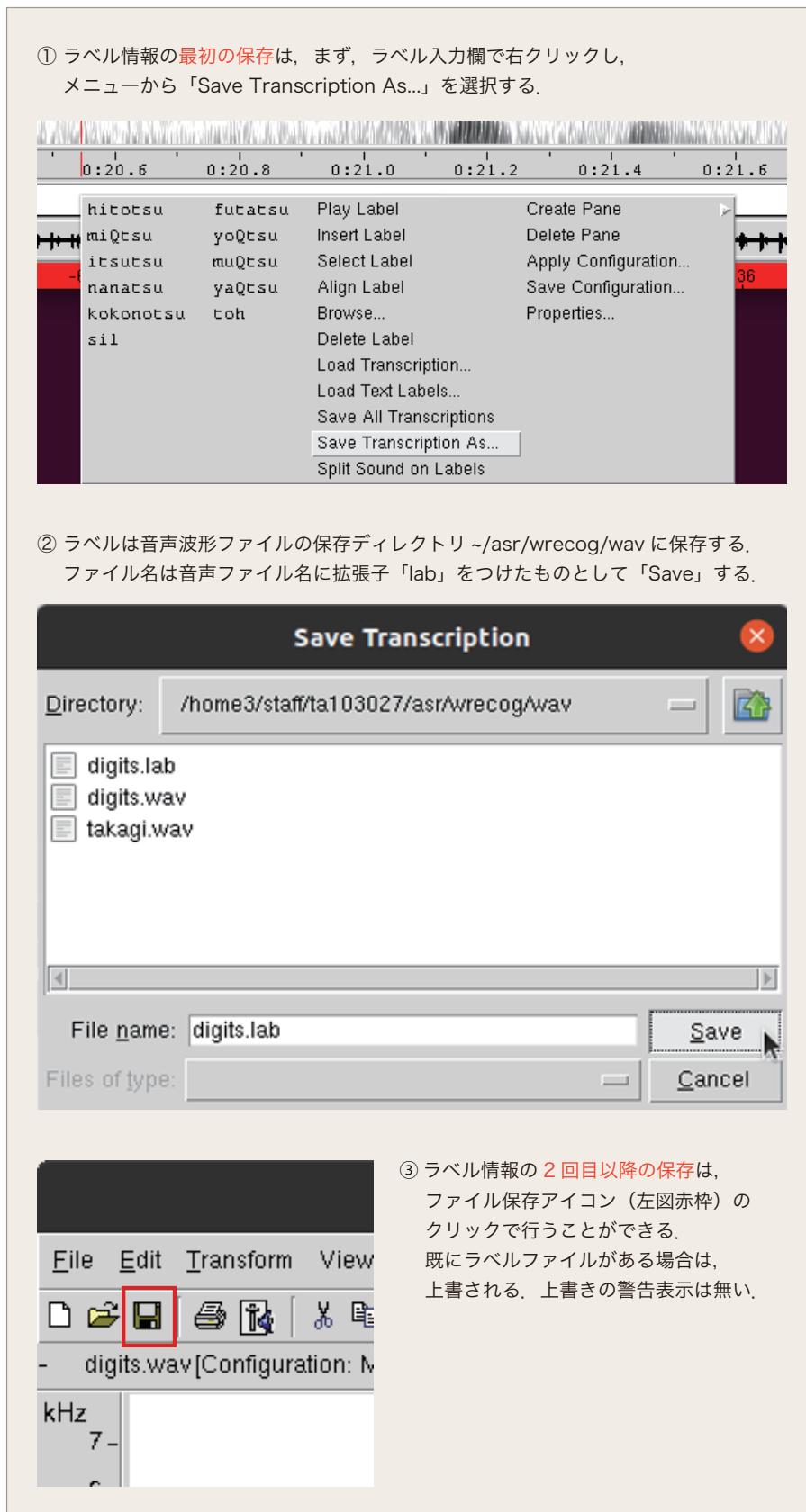
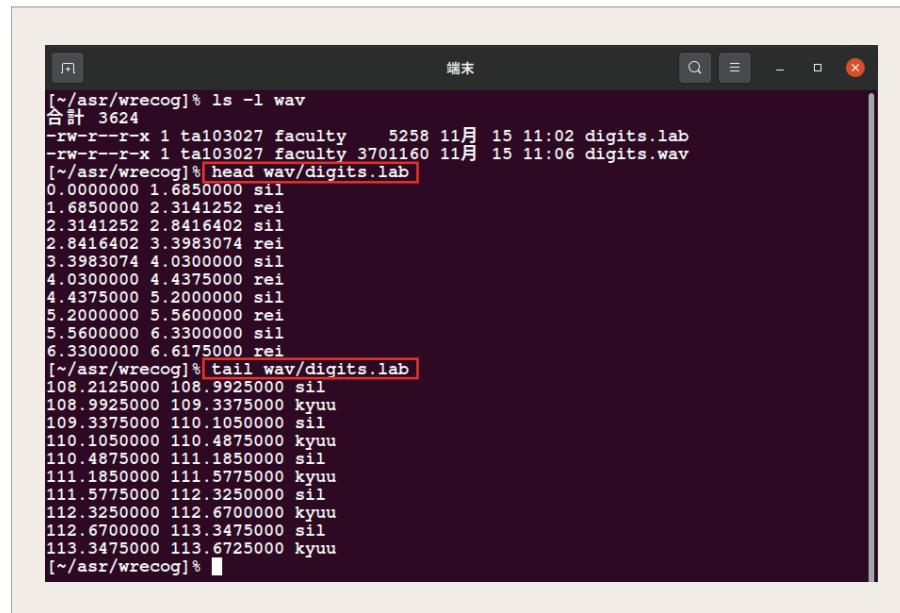


図 6.13: ファイル保存アイコンをクリックして入力済みラベル情報を保存。

## 2. ラベルデータの検査

ラベルデータはテキスト形式で記録されているので、端末に表示したり（図 6.14）、テキストエディタで編集することができます。ラベルが正しく付与されているかどうか検査してください。



The screenshot shows a terminal window titled "端末" (Terminal) with the following command history:

```
[~/asr/wrecog] % ls -l wav
合計 3624
-rw-r--r-x 1 ta103027 faculty 5258 11月 15 11:02 digits.lab
-rw-r--r-x 1 ta103027 faculty 3701160 11月 15 11:06 digits.wav
[~/asr/wrecog] % head wav/digits.lab
0.000000 1.6850000 sil
1.6850000 2.3141252 rei
2.3141252 2.8416402 sil
2.8416402 3.3983074 rei
3.3983074 4.0300000 sil
4.0300000 4.4375000 rei
4.4375000 5.2000000 sil
5.2000000 5.5600000 rei
5.5600000 6.3300000 sil
6.3300000 6.6175000 rei
[~/asr/wrecog] % tail wav/digits.lab
108.2125000 108.9925000 sil
108.9925000 109.3375000 kyuu
109.3375000 110.1050000 sil
110.1050000 110.4875000 kyuu
110.4875000 111.1850000 sil
111.1850000 111.5775000 kyuu
111.5775000 112.3250000 sil
112.3250000 112.6700000 kyuu
112.6700000 113.3475000 sil
113.3475000 113.6725000 kyuu
[~/asr/wrecog] %
```

図 6.14: WaveSurfer で付けた HTK 形式の音声セグメントラベル。ラベルファイルの拡張子は”lab”。時間数値の単位は秒。

次の点に注意します。

時間の数値が秒単位として妥当であるか？ ラベルの入力形式が適切でない場合は、大きな整数値になっていることがあります。WaveSurfer で当該データを読み込み、ラベル形式を設定し直した（前述）後、ラベルファイルを保存し直すことによって時間が秒単位になります。

単語名（“rei”，“ichi”など）が正しく揃っているか？ 繰りが 1 文字でも異なっていると、異なる単語データとして処理されてしまいます。必要があれば、エディタで編集して整えてください。

余分なラベルが記録されていないか？ ラベル入力作業の過程での手違いにより意図しないラベルが入力されてしまっていることがあります。必要があれば、エディタで編集して整えてください。

上記の修正をした場合、WaveSurfer で当該データを読み込んで、音声波形とラベルの対応を確認してください。

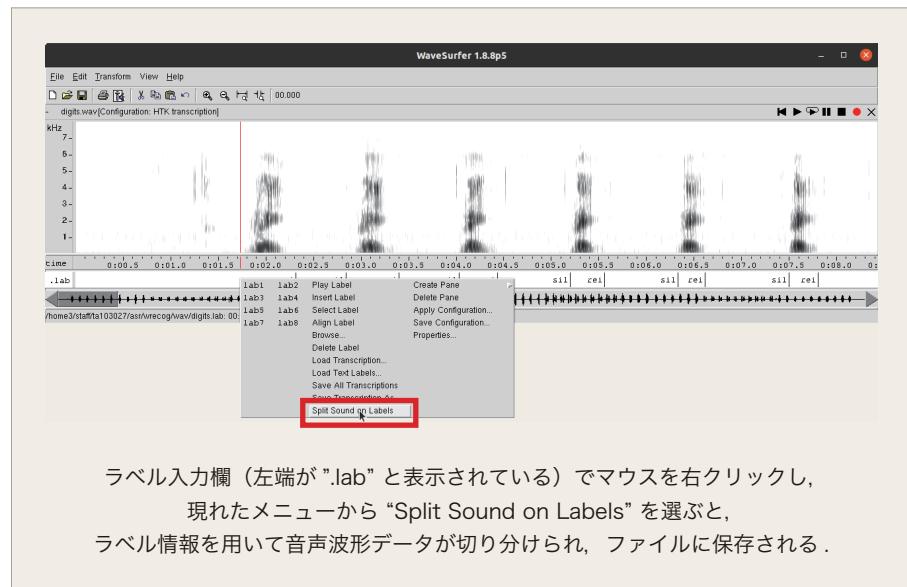


図 6.15: WaveSurfer のラベル入力欄（左端が”.lab”と表示されている）でマウスを右クリックし、現れたメニューから“Split Sound on Labels”を選ぶと、ラベル情報を用いて音声波形データが切り分けられ、ファイルに保存される。

### 3. 単語音声学習データの作成

WaveSurfer で付けたラベルに基づいて、音声波形データから個々の単語の発話を切り取り、別々のファイル保存します。切り取られた音声波形の小区間をセグメント (segment) といいます。WaveSurfer のラベル入力欄（左端が”.lab”と表示されている）でマウスの右クリックし、現れたメニューから“Split Sound on Labels”を選ぶと（図 6.15）、ラベル情報を用いてセグメントデータが生成されてファイルに保存されます。音声波形ファイルが `digits.wav`、そのラベルファイルが `digits.lab` である場合、音声波形ファイルとラベルファイルが格納されているディレクトリの下に `digits.wav.split` という名前のサブディレクトリが作られ、その中にセグメント波形データが保存されます。

どんなセグメント波形ファイルができたのか、確認してみましょう。このテキストの説明のための例の場合、次のような 6 行のセグメント番号、ラベル文字列、拡張子 (.wav) からなる WAV ファイルが生成されました。

```
[~/asr/wrecog]% ls wav/digits.wav.split/ | head
000000rei.wav
000001sil.wav
000002rei.wav
000003sil.wav
000004rei.wav
000005sil.wav
000006rei.wav
000007sil.wav
000008rei.wav
```

```
000009sil.wav
[~/asr/wrecog]% ls wav/digits.wav.split/ | tail
000189sil.wav
000190kyuu.wav
000191sil.wav
000192kyuu.wav
000193sil.wav
000194kyuu.wav
000195sil.wav
000196kyuu.wav
000197sil.wav
000198kyuu.wav
```

`ls` コマンドで保存先として指定したディレクトリの内容を表示し、ファイルを確認してください。学習データの準備はとても重要です。念のため、ファイル名一覧だけではなく、ファイルサイズもチェックしておきましょう。例えば、単語の長さが 0.5 秒の場合、その WAV ファイルのサイズ（バイト数）はいくつですか？ちなみに、この実験の音声データ形式（16kHz サンプリング、1 チャネル、linear PCM）に対する WAV データのヘッダサイズは 44 バイトです。

#### 4. 作成した単語音声の検聴

単語音声データが正しく作成されたかチェックします。この作業は大事です。必ず作成した音声波形ファイルをひとつひとつ検聴してください。例えば "rei" とラベル付けされた音声セグメントを一括検聴する場合は、端末で、

```
[~/asr/wrecog]% foreach f ('ls wav/digits.wav.split/*rei.wav')
foreach? echo $f
foreach? play $f
foreach? end
```

のように入力すると、ワイルドカードに一致する音声ファイルについて、順番にファイル名が表示され音声が再生されます。検聴の結果によって、必要があれば、音声を収録し直したり、ラベルを付け直すなどしてください。

`foreach` 文の括弧の中の ‘ はバッククオート (back quote) であることに注意してください。IED 教室の端末のキーボードでは「@」のキーに割り当てられています（図 6.16）。このテキストの PDF 版からコピー＆ペーストで端末にコマンドを入力すると、表示フォントの影響でバッククオートが正しく入力されません。バッククオートの部分は直接キーボードから入力してください。

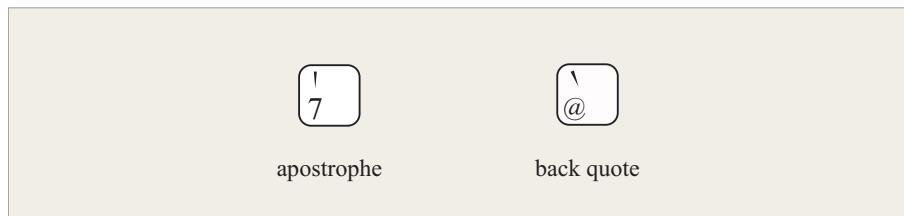


図 6.16: apostrophe と back quote の IED 教室端末のキー位置。

音声データを再生する `play` コマンドを実行した際に表示される「play WARN alsal: can't encode 0-bit Unknown or not applicable」のメッセージは無視して結構です。

### 6.5.3 スペクトル分析

録音した全ての音声波形をスペクトル分析して MFCC に変換し、ファイルに保存します。スペクトル分析には第 3 章で作成した `mfccf` を用います。個々の音声ファイルごとにコマンドを入力するのは大変なので、シェルの機能を利用し、以下のように一括処理を行います。

```
[~/asr/wrecog]% foreach f ( `ls wav/digits.wav.split/*.wav` )
foreach? set g = $f:t:r
foreach? set h = mfcc/${g}.mfcc
foreach? mfccf $f $h
foreach? end
```

ここで、`:t`(tail)はファイル名からパス名を取り除いた部分を取り出す、`:r`(root)はファイル名から拡張子を取り除いた部分を取り出す作用をします。MFCC の値を保存したファイルは、`mfcc` というディレクトリに保存されます。MFCC のファイルの数は必ず確認しておいてください。セグメントの種類毎の数（この例では `rei`）を確認するためには以下のようにします。-1 の 1 は数字の 1 ではなく、エル  $\ell$  です。

```
[~/asr/wrecog]% ls mfcc/*rei.mfcc | wc -l
10
```

### 6.5.4 単語 HMM の学習

1. 単語 HMM の学習用データの一覧を単語毎に作成します。データの一覧はディレクトリ `~/asr/wrecog/lib` に保存してください。たとえば、単語「0」の場合、

```
[~/asr/wrecog]% ls mfcc/*rei.mfcc > lib/rei.list
```

と入力します。実行後に `lib/rei.list` の内容を確認しておいてください。この作業も `foreach` を使えば、一度に行うことができます。単語名の一覧を `lib/wordlist` というテキストファイルに用意しておけば、以下のように一括処理することができます。

```
[~/asr/wrecog] % foreach d ( 'cat lib/wordlist' )
foreach? ls mfcc/??????${d}.mfcc > lib/${d}.list
foreach? end
```

2. Baum-Welch アルゴリズム（5.3.4 節）を用いて、単語 HMM を学習します。

状態数は単語に含まれる音素の数に応じて決めます。この実習の実験条件においては、単語を構成する各音素と前後の無音声区間に 1 状態を割り当てるのが適当と思われます。音素（第 2.2 節）1 つあたり 1 状態を割り当ててください。長音（/aa/, /ii/, /uu/, /ee/, /oo/）や促音（/Q/）は 1 音素とみなします。第 2 章の言語の二重文節の説明図（図 2.1）では、言語的に長音を 2 音素としていますが、音声現象としては長音は短音の継続長が長くなったものと見做して差し支えありません。さらに、単語の開始前と終了後に無音声（sil）のための 1 状態を割り当ててください（図 6.17）。さらに、単語の開始前と終了後に無音声（sil）のための 1 状態を割り当ててください。したがって、例えば、「電気通信大学（/deNkitsuushiNdaigaku/）」という単語の HMM の状態数は 19 とします。このようにして決めた状態数で HMM の学習を試みますが、学習が失敗する場合は状態数を増減して試してください。本実習では学習データが少ないので、状態数を減らす方向に調整した方が学習が成功することが多いと思われます。ファイルには、必ず **単語名（表 6.1）に対応した名前** を付けてください。

| 単語<br>(正書法) | 音素 + 無音声区間         | HMM 状態数<br>(本実験での目安) |
|-------------|--------------------|----------------------|
| 太郎          | sil t a r oo sil   | 6                    |
| は           | sil w a sil        | 4                    |
| 学校          | sil g a Q k oo sil | 7                    |
| へ           | sil e sil          | 3                    |
| 行った         | sil i Q t a sil    | 6                    |

図 6.17: 本実験の条件において単語 HMM に割り当てる状態数の目安. 音素 1 つあたり 1 状態. 長音 (/aa/, /ii/, /uu/, /ee/, /oo/), 促音 (/Q/) は 1 音素とみなす. 単語の開始前と終了後に無音声 (sil) のための 1 状態を割り当てる.

HMM の状態数を 5 に定めて, *rei* (表 6.1 の単語表記「0」の単語) の単語モデルを作成する場合は

```
[~/asr/wrecog]% train lib/rei.list 5 hmm/rei.hmm log/rei.log
```

と入力し, HMM の学習を実行します<sup>6</sup>. 単語の長さが比較的揃っている場合は, 以下のようにして全ての単語 HMM の学習を一度に行うことができます (状態数を 7 とした場合). ただし, 学習に失敗した単語があるかもしれませんので, 全ての単語 HMM の学習記録の検査をしてください.

```
[~/asr/wrecog]% foreach w ( 'cat lib/wordlist' )
foreach? train lib/${w}.list 7 hmm/${w}.hmm log/${w}.log
foreach? end
```

*lib/rei.list* は学習データの一覧です. *hmm/rei.hmm* は学習された HMM のパラメータ値が保存されているテキストファイルです. パラメータの再推定毎の  $\log P(\mathbf{O}|\tilde{\lambda})$  の値が *log/rei.log* に記録されます. どれか 1 つの HMM の  $\log P(\mathbf{O}|\tilde{\lambda})$  の値を用いて, 図 6.5 のような学習曲線のグラフを作ってください. この図は, gnuplot のバッチファイル *drawLC* を使い,

```
[~/asr/wrecog]% gnuplot drawLC
```

と入力して作りました. *drawLC* は gnuplot スクリプトファイルです. **学習曲線を描画する単語の名前を自分用に編集して使ってください.** 上記を実行すると学習曲線が表示されます. 表示内容を確認してリターンキーを押すと, *learningCurve.png* というファイルに図のデータが保存されます.

<sup>6</sup>この *train* という学習プログラムは HMM の混合数を 4 に固定しています.

6.5.4.0.1 学習の失敗 学習プログラム `train` を実行すると、以下のように学習が失敗することがあります。

```
[~/asr/wrecog]% train lib/hachi.list 6 hmm/hachi.hmm log/hachi.log
training_data= lib/hachi.list
tokens= 10
frame= 556
dim= 20
state= 6
mix= 4
hmm_file= hmm/hachi.hmm
log_file= log/hachi.log
(0) logp= -1.011268e+03
(1) logp= -9.675306e+02
(2) logp= -9.605083e+02
(3) logp= -nan
```

この例のように、 $\log P(\mathcal{O}|\tilde{\lambda})$  の値が途中で `nan`（非数）となり、ここで学習プログラムが終了してしまいます。学習に失敗する場合は、次のような原因が考えられます。

- 音声波形データの準備が正しくなされていない。
- スペクトル分析に失敗している。
- 音声が正しく収録されていない（単語の途中で切れている、音が割れている、など）
- 学習データに違う単語のデータが混ざっている
- 状態数が少なすぎる、または多すぎる

単純な学習作業なので、学習データが整っていれば、失敗することはほとんどありません。学習に失敗する場合は、音声収録から学習までの手順を見直してください。作業の過程で作るファイルの数、ファイル名、サイズ、ファイル形式、内容などをチェックするとよいでしょう。上記の失敗例の場合、学習データに問題は無かったので、状態数を 6 から 5 に変えてみたら、以下のように学習が成功しました。

```
[~/asr/wrecog]% train lib/hachi.list 5 hmm/hachi.hmm log/hachi.log
training_data= lib/hachi.list
tokens= 10
frame= 556
dim= 20
state= 5
mix= 4
```

```

hmm_file= hmm/hachi.hmm
log_file= log/hachi.log
(0) logp= -7.223173e+02
(1) logp= -6.907365e+02
(2) logp= -6.867466e+02
(3) logp= -6.852862e+02
...
(46) logp= -6.766227e+02
(47) logp= -6.766215e+02
(48) logp= -6.766205e+02

```

### 6.5.5 学習の検証

単語 HMM が学習できたので認識してみましょう。この実験には `recogf` という単語認識プログラムを用います。HMM の学習データの認識を行い、学習が出来ているかどうかの検証をします。まず、`recogf` というコマンドに、数字単語 HMM 一覧ファイル `lib/HMMList` と認識したい音声の MFCC ファイルを指定すると、ビタビ確率  $\underset{i}{\operatorname{argmax}} P^*(w_i|O)$ ,  $w_i \in \mathcal{W}$  (式 5.13) を計算して表示します。以下に一例を示します。

```

[~/asr/wrecog]% recogf lib/HMMList mfcc/000000rei.mfcc
0 rei
[~/asr/wrecog]% recogf lib/HMMList mfcc/000040ni.mfcc
2 ni
[~/asr/wrecog]% recogf lib/HMMList mfcc/000060saN.mfcc
3 saN
[~/asr/wrecog]% recogf lib/HMMList mfcc/000140nana.mfcc
7 nana

```

認識させたいファイルをいくつか指定してみましたが、ここで認識させたものについては全て正しく認識されました。

次に、学習に用いた単語音声を認識し、正しく認識されたデータの個数を数えてみます。認識対象となるのは `lib/wordlist` に書かれている単語です。

```

[~/asr/wrecog]% countCorr
Evaluation start.  Wait a moment.
rei, 10
ichi, 10
ni, 10
saN, 10
yoN, 10
go, 10

```

```
roku, 10
nana, 10
hachi, 10
kyuu, 10
```

10 個の識別で、かつ学習データを用いて評価しています。認識率は 100%となりました。HMM の性能評価としての意味は無く、HMM の学習の確認と次の On-The-Fly 認識に必要な各種設定ファイルが正しく作られていることの確認の意味があります。認識性能の評価をするためには、On-The-Fly 認識の他、学習に用いない性能評価用の音声データを用います。

### 6.5.6 On-The-Fly 単語音声認識

単語 HMM の学習に成功したら、いよいよマイクから入力した音声を直接認識してみましょう。認識処理の結果は端末に表示されるだけなので、適宜適当なファイルに保存しておいて、後で実行例としてレポートに使ってください。「設定」の「サウンド」パネルはデスクトップ画面に出したままにしておいてください。「サウンド」設定パネルが開いた状態でないと、音声が入力されないことがあります。入出力音量は実験の途中で適宜調節するとよいでしょう。

#### 6.5.6.1 音声区間検出の閾値の決定

音声認識では、マイクから入力される音響信号から音声が存在する区間を自動的に検出します。この実習では音量に基づいて音声区間を検出（6.3.2 節）します。最初に音声区間検出の閾値を決めます。ヘッドセットを装着します。`vu` というコマンドを用いると、マイク入力のパワーレベルの概算値が分かります。なにもしゃべらない時のレベル値より少し大きい値を音声区間検出の閾値の目安とするとよいでしょう。はじめに、端末エミュレータの横幅を少し広げておいてください。一行の文字数が不足すると、表示が”流れで” しまいます。

表示は状況に応じて変化します。下記の例は、ある時点の状況を示したもので、グラフ上の「\*」は最大値の位置を示しています。この状況の場合、-25 dB に定めるのが適當だと思われます。終了するためには、`Ctrl+C` をキー入力します。

```
[~/asr/wrecog] % vu
```

```
Short Time Speech Power
-60   -50   -40   -30   -20   -10    0 dB
+-----+-----+-----+-----+-----+
rec WARN alsa: can't encode 0-bit Unknown or not applicable
.=====*=====
```

この際に表示される “rec WARN alsa: can't encode 0-bit Unknown or not applicable” のメッセージは無視して結構です。

### 6.5.6.2 単語音声認識の実行

閾値を決めたら、次は、単語認識のプログラムを起動します。音声認識を行うためには、例えば以下のように端末で `recog` コマンドを起動します。

```
[~/asr/wrecog]% recog -25 lib/HMMList
```

“-25”は `vu` の表示を見て決めた音声区間検出の閾値 [dB] です。この数字は一例です。大きい値を指定すると、音声パワーの大きい部分だけが処理されるようになります。音声の最初と末尾が切れる可能性が高くなります。値を小さくすると、音声の最初と末尾が切れる可能性は減りますが、音声以外の音も処理される可能性が高まります。認識がうまく行かない場合は、適宜調整してみてください。

「Return キーを押してください」と表示されます。この状態は音声の入力を行っていませんので、話しても処理しません。Return キーを押して、単語を発声してください。認識処理が終わると、端末結果が表示されます。単語の順位、単語表記、単語名、尤度が、尤度 ( $\log P^*(O|\lambda)$ ) の降順に表示されます。実際に入力した単語が第 1 位の単語と一致していれば認識成功ということになります。音声区間検出によって検出された音声波形は `~/asr/wrecog/vadwav` に保存されます。発話毎に異なったファイル名が付けられます<sup>7</sup>。On-The-Fly 単語認識を終了するためには、`Ctrl+C` をキー入力します。この際に表示される “rec WARN alsal: can't encode 0-bit Unknown or not applicable” のメッセージは無視して結構です。

```
[~/asr/wrecog]% recog -25 lib/HMMList
Return キーを押してください
Isolated Word Recognition On-The-Fly
rec WARN alsal: can't encode 0-bit Unknown or not applicable
vad_file= vadwav/76302.wav
-----
rank word (kana)= log-likelihood
-----
1. 8 (hachi)= -952.510926
2. 1 (ichi)= -977.724834
3. 3 (saN)= -1032.404311
4. 0 (rei)= -1087.581653
5. 2 (ni)= -1167.379263
6. 4 (yoN)= -1227.270401
7. 6 (roku)= -1240.591021
8. 7 (nana)= -1261.096937
9. 5 (go)= -1360.261797
10. 9 (kyuu)= -1373.558098
```

---

<sup>7</sup>MFCC 分析を行ったプロセスのプロセス番号で名前を振っています。

ハングアップ

Return キーを押してください

Isolated Word Recognition On-The-Fly

```
rec WARN alsal: can't encode 0-bit Unknown or not applicable
```

```
vad_file= vadwav/76327.wav
```

```
-----
```

```
rank word (kana)= log-likelihood
```

```
-----
```

1. 8 (hachi)= -911.710779
2. 1 (ichi)= -916.062055
3. 0 (rei)= -1009.450527
4. 3 (saN)= -1066.173722
5. 2 (ni)= -1104.335671
6. 4 (yoN)= -1208.086255
7. 6 (roku)= -1217.270178
8. 7 (nana)= -1236.537874
9. 5 (go)= -1378.731672
10. 9 (kyuu)= -1437.539677

ハングアップ

Return キーを押してください

Isolated Word Recognition On-The-Fly

```
rec WARN alsal: can't encode 0-bit Unknown or not applicable
```

```
vad_file= vadwav/76349.wav
```

```
-----
```

```
rank word (kana)= log-likelihood
```

```
-----
```

1. 0 (rei)= -nan
2. 1 (ichi)= -nan
3. 2 (ni)= -nan
4. 3 (saN)= -nan
5. 4 (yoN)= -nan
6. 5 (go)= -nan
7. 6 (roku)= -nan
8. 7 (nana)= -nan
9. 8 (hachi)= -nan
10. 9 (kyuu)= -nan

### 6.5.6.3 認識結果の検討

この実行例では、1回目と2回目の音声入力では、第1位の単語が発話した単語に一致していました。しかし、3回目の音声入力に対しては単語 HMM の尤度

の計算に失敗しました。音声区間検出によって切り出されて尤度計算に用いた音声波形は [vadwav/76349.wav](#)（認識結果の上に表示）に保存されています。認識に成功した第 2 回目の音声波形 [vadwav/76327.wav](#) と比べてみましょう（図 6.18）。

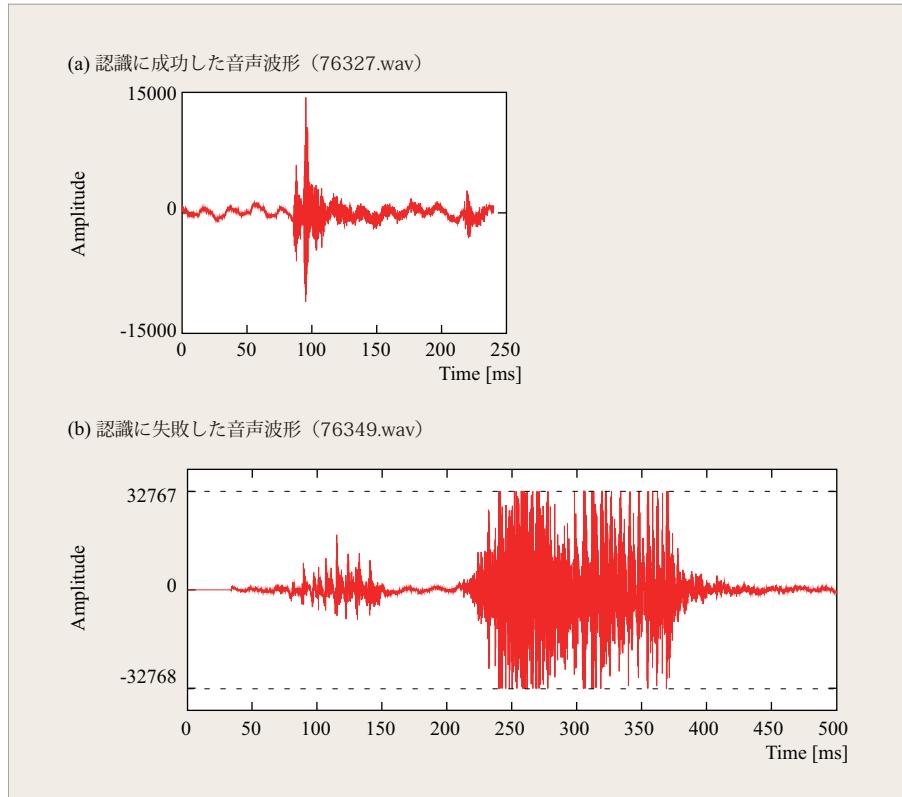


図 6.18: On-The-Fly 単語認識に (a) 成功した音声と (b) 失敗した音声の例。 (b) の場合は入力レベルが大きすぎて符号付 16 ビット整数の値の範囲を超えててしまっている。この図は ad2txt16 の出力を gnuplot で表示したもの説明用に修整したもの。

音声波形の振幅は符号付 16 ビット整数で表現されています。すなわち、値の範囲は  $-32768 \sim +32767$  です。認識に成功した音声波形（図 6.18(a)）は振幅値がこの範囲に収まっていますが、失敗した音声波形（図 6.18(b)）は振幅値が超えてしまっている部分があります。声が大きすぎたためです。

認識結果が間違っていると分かって声を張って発話すると、このような入力となり、かえって失敗することがあります。落ち着いて、単語 HMM の学習用の音声データを発話したときと同じ程度の音量で発話しましょう。

その他に、認識が失敗する事例で良くあるのは、入力された音声が途中で切れてしまう場合です。音声区間自動検出の閾値に比べて単語音声のパワーが小さすぎる場合、単語の途中の破裂子音（/k/, /t/, /g/, /d/, /p/）の閉鎖部（図 2.12）や促音「っ」が長すぎる場合に、単語が終わったと判断され、ここで入力を打ち切られた音声が認識処理されることがあります。

On-The-Fly 単語認識で正解が得られるための条件は以下の通りです。

1. 認識対象の単語が入力される。

2. 学習データと同じ話者が発話する。
3. 学習データと同じ発音で発話する。
4. 学習データと同じ条件（マイク等の種類、マイクの位置、背景雑音の状態など）で入力される。
5. 音声区間検出の閾値が適切である。
6. 適切な音量で入力され、音声区間検出が正しく実行される。
7. 認識対象の単語に類似した発音の単語が無い。

満たされない条件があると、単語認識が失敗する可能性が高まります。認識に失敗した場合は、その原因について考察してください。そして、原因と推定される条件を改善してやり直してみましょう。

WaveSurfer で `vadwav` に保存されている音声波形を表示したり、聞いてみると、認識の成否の原因を知ることができる場合が多いです。是非、調べてみてください。

### 6.5.7 プログラム構成

この実習では、最も単純な音声認識タスクである特定話者孤立単語音声認識 (Speaker-Dependent Isolated Spoken Word Recognition) を行いました。特定話者 (Speaker-Dependent: SD) とは、ある一人の話者の音声のみに対応しているということです。孤立単語音声認識 (Isolated Spoken Word Recognition) とは、単語が 1 つだけ含まれていて、単語の前後に無音区間がある音声を認識する方式のことです。

この章の実習を実現するための信号処理とパターン認識のコマンドは C 言語で作られています。On-The-Fly 音声認識のコマンド (`recog`) は、シェルスクリプトです。PulseAudio の録音コマンド (`rec`)、音声区間検出 (`vad` コマンド、第 6.3.2 節)、スペクトル分析 (`mfcc` コマンド、第 3 章、第 1 日の実習で作成)、および、Viterbi アルゴリズム (第 5.3.3 節) による一方向性連続 HMM (第 5.4 節) に対する確率計算 (`_recog` コマンド) 等を組み合わせて実現しています (図 6.19)。

このシェルスクリプトにおけるコマンドの組み合わせの中核では Unix のパイプ (`pipe`) という仕組みを使っています。パイプは「前のコマンドの標準出力を次のコマンドの標準入力に送ることにより、コマンドの実行結果を次のコマンドに渡すシェルの機能です」[17]。パイプを介して複数のコマンドを組み合わせることによって複雑な機能を実現することが可能になります。

On-The-Fly 音声認識に関わっているスクリプトやソースコードファイルを以下に列挙します。実習作業に余裕があれば、内容を見て処理の仕組みの理解に努めてください。この実習に提供しているソースコード等オリジナルのファイル一覧は付録 B に掲載しています。

`wrecog/program/recog` On-The-Fly 単語認識. 以下の 5 つのコマンドを組み合わせているシェルスクリプト.

`sound/audioIN` 音声入力（録音）コマンド. シェルスクリプト.

`sound/vad.c` 音声区間検出. C 言語.

`wrecog/program/mfcc.c` スペクトル (MFCC) 分析. C 言語.

`wrecog/program/_recog.c` Viterbi アルゴリズム. C 言語.

`sound/hupAudioIN` 音声入力プロセスを終了する. シェルスクリプト.

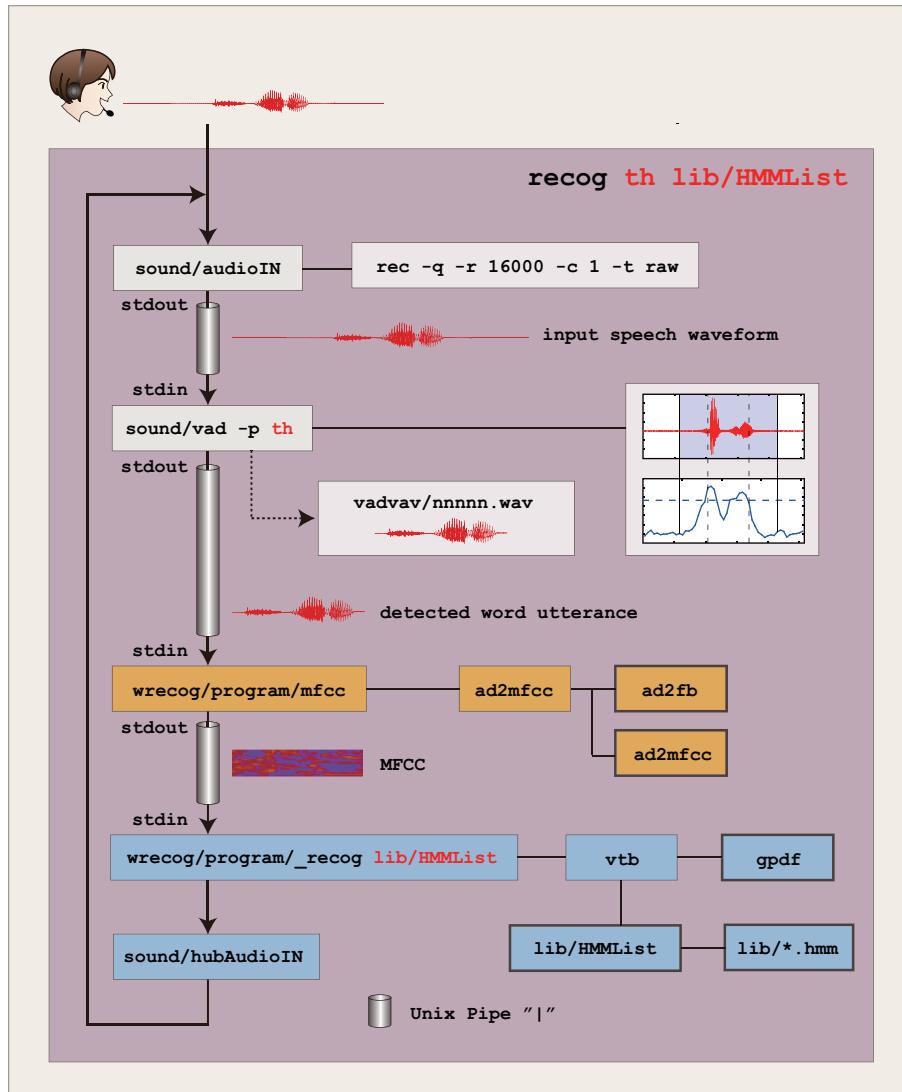


図 6.19: On-The-Fly 単語音声認識のモジュール構成. 音声入力 (`sound/audioIN`) から音声区間検出プログラム (`sound/vad`) に入力波形データを送り, `sound/vad` は音声区間を切り出して送り出し, スペクトル分析 (`wrecog/program/mfcc`) プログラムはそれを分析して MFCC 時系列を認識プログラム (`wrecog/program/_recog` に送る. 認識結果は `recog` を実行した端末に出力される. これを無限ループで繰り返している. 図中の太枠のモジュールは穴埋め問題で完成あるいは認識単語に応じて受講者が作成するもの.

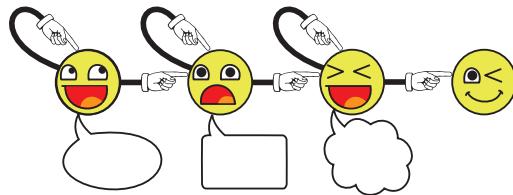
### 6.5.8 レポート（第 3 週）

#### 1. 単語音声認識実験

- (a) 単語認識タスクを設計してください。
- あなたの単語認識タスクの目的を述べてください。仮に使うとしたらどのような用途に用いますか？
  - 5種類以上の単語からなる単語集合を定義し、一覧表（表 6.1 の形式）を作ること。
  - 各単語のモデルに用いた HMM の状態数を上記一覧表に記述すること。
- (b) HMM の学習曲線（図 6.5 に相当するもの）。認識対象の単語のうち任意の 1つに対するものでよい。
- (c) 学習の検証の結果
- (d) On-The-Fly 単語音声認識の結果と考察

#### 2. このテーマについて（第 1 週から第 3 週の総括）

- (a) ポイントは何であったか？
- (b) 良くわかったこと
- (c) わからなかったこと
- (d) 要望
- (e) 感想、その他



## 第7章 おわりに



単語音声認識の実験はいかがでしたか？

私がこの実験を計画するにあたり、いろいろな選択肢がありました。フリーソフトを組み合わせて入力した音声を文字にする「音声タイプライタ」を作る、音声認識ソフトウェアを利用してコンピュータを操作したりロボットを動かす、などは受講者にとって魅力的な実習内容だと思います。実際に行うことも可能です。しかし、2コマ×3回の時間でこのような実習を行おうとすると、既存ソフトウェアをうまく使って組み合わせて応用した”的な内容になってしまいます。

大学では、そのようなことよりも音声認識の基本原理そのものを学ぶべきだと思いました。本実験では、音声認識の原理そのもの、すなわちブラックボックスをホワイトボックスとして理解できるような実習を行うことを狙いとしました。その目的のため、ソースコードは個々のプログラムあたりせいぜい数百行に收めること、全て開示すること、自分でコンパイルできることとしました。

メディア情報学実験のテーマとして音声認識実験をこの制約で組み立てるとなると、音声タイプライタは実施不可能でした。音声タイプライタを実装するためには、単語HMMを用意してもダメで、音素HMMを学習し、3万語程度以上の単語辞書を作成し、単語同士のつながりを規定するための文法規則あるいは統計情報を作成し、これらの情報を用いることによって入力音声から得られる数百の認識候補から答えを効率的に選びだすためのアルゴリズムが必要です。このような機能を備えたソフトウェアを作るとすると、専門の研究者による長年の開発が必要で、ソースコードの量は膨大になります。例えば、フリーの音声認識ソフトウェアであるHTK (Hidden Markov Model ToolKit, <https://htk.eng.cam.ac.uk/>)

や Julius (<https://julius.osdn.jp/>) のソースコードはコメントも含めて C 言語で約 11~12 万行もあります。音声認識に必要なことはほぼ全部できる汎用のツールなので、その機能を実現するために多量の記述が必要なのです。さらに、音素 HMM の学習データを作成するためには、そこそこの長さがある数百種類の文を録音し、音素の種類と位置を示した情報を付与する必要があります。手慣れたエキスパートでも、この作業だけでかなりの日数を要します。

スペクトル分析、HMM などの原理（数理）は教科書的に説明し、音声認識の実行は HTK や Julius などのフリーソフトを利用するような実習のイメージがちっと頭をよぎったことがあります。でも、それでは原理と実装の間の溝は埋まりません。

あれこれ考えてみて、結局、研究室（高木研究室とその母体である電通大尾閥研究室（1989 年～2008 年））で受け継がれている単語認識のプログラム群を利用することになりました。研究用に書かれたプログラムなので、最小限の入出力処理以外は、教科書に書かれた原理（数理）を実行する数値計算のソースコードしかなく、ソースコードのファイルを単独でコンパイルすれば動く、小さくて綺麗なプログラムが揃っています。そこで、このプログラム群からいくつかピックアップし主として入出力処理を改良したプログラム、および IED での音声入力用に新たに書いたプログラムを合わせて、単語認識実験を行うことにしたのです。

HMM 算法に関する実習プログラムは上坂先生尾閥先生の『パターン認識と学習のアルゴリズム』[9] を参考に作成しました。この実習のソースコードは音声入力、音声区間検出など含めたプログラムも全部合わせ、コメントも含めて約 6700 行です。最も大きいソースコードは単語 HMM の学習プログラム `train.c` の 1529 行です。

受講生が「学んだ音声認識アルゴリズムを用いて実際に単語認識を行うソースコードを理解」できるようにする目標をもって教材を作りましたが、残念ながら現状では中途半端になってしまいました。第 3 週の単語認識の実習に関わる内容です。単語 HMM の学習を行うプログラム `train` に用いている Baum-Welch アルゴリズムでは連続 HMM 用の計算式（5.4.2 節）を用いています。本当は、式 5.29～式 5.43 までの計算式を理解してもらって、穴埋め式の実習を行ってもらいたかったのですが、実習の時間が足りないため割愛しています。また、単語認識を行うビタビアルゴリズムのプログラム (`~/asr/wrecog/program/vtb.c`) はテキストにその原理を解説（5.3.3 節）してますが、実習の課題からは除外してあります。これを含めて多少の事項は割愛しても、認識アルゴリズムを C 言語のソースコードとして把握可能な規模に収め、自分の音声を用いて音声認識を行う実習を優先させたためです。ビタビアルゴリズムも単語音声を認識するために多次元次数ベクトルの扱いを行うと、時間内に消化しきれなくなってしまいます。そこで、HMM の学習とそれを用いた認識の基本原理は、演習的設定の条件で実習を行ってもらい、実際の単語 HMM の学習とそれを用いた単語認識は、出力確率の記述は複雑になるけれどもアルゴリズムは同じ、という説明にとどめています。実はコメントを含めてわずか 484 行のプログラム (`~/asr/wrecog/program/_recog.c`) なので、ちょっと頑張れば、どこでどの計算を行っているのかくらいのことは、

分かってもらえるものと思っています。

On-The-Fly 単語認識では、音声区間検出の閾値の設定に苦労したことと思います。本実験で用いた検出法が最も簡単なものだったからかもしれません。音声区間検出は音声認識の性能を左右する極めて重要な技術です。認識したい音声を他の信号から区別するのは、実はとても難しいのです。古くから研究が進められていて、現在も新しい方法が提案されています。最近、音声区間検出手法を評価するためのデータとツールが作られました [22]。この古くて新しい技術に興味がある人は、まずは文献 [26] を読んでみると良いでしょう。

音声認識の研究に興味を持ったのであれば、まず、荒木雅弘『イラストで学ぶ音声認識 改訂第2版』(文献 [38])を手にとってみるとよいでしょう。比較的最近の音声認識技術の全般が分かりやすく解説されています。

音声認識の基礎から最新の手法までを学びたい方は高島遼一『Pythonで学ぶ音声認識(機械学習実践シリーズ)』[33]が良いでしょう。近年、画像像処理、パターン認識、機械学習に広く使われている Python を用いて、音声分析の基本、DP マッチング、GMM-HMM、DNN-HMM、End-to-End モデルによる連続音声認識までを実例を用いて実行し、その仕組みを体験することができます。GitHub にソースコードが載っています。

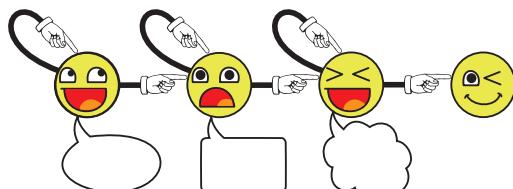
中川 聖一編著『音声言語処理と自然言語処理(増補)』[30]は音声言語(音声を媒介とする意図伝達手段)と文字言語(文字を媒介とする意図伝達手段)の工学的応用(音声認識、音声合成、機械翻訳、検索など)を目的とした基礎技術について、有機的に関連させて解説している良書です。各章に章末問題とその解答が巻末に設けられている他、音声言語処理と自然言語処理に有用な各種フリーソフトウェアが紹介されています。

最近刊行された専門書として、上述の荒木雅弘『イラストで学ぶ音声認識 改訂第2版』(文献 [38])のほか、武田龍・高島遼一『音源分離・音声認識』(文献 [36])、伊藤克亘・宮澤幸希『Pythonで学ぶ音声コミュニケーション情報処理』(文献 [37])もお薦めです。

限られた実習時間で音声認識を理解して(すくなくとも理解したつもりになって)もらうために工夫をしているつもりですが、いかがだったでしょうか? 率直なご意見とご提案をお待ちしています。

高木一幸 2014年11月4日(2025年10月10日改訂)

takagi@uec.ac.jp



## 関連文献(出版年順)

- [1] Tsutomu Chiba, Masato Kajiyama, The Vowel—It's Nature and Structure—, Phonetic Society of Japan, Setagaya, Tokyo, Japan, 1958.
- [2] Gunnar Fant, Acoustic Theory of Speech Production, Mouton & Co., Printers, New York, The Netherlands, 1960.
- [3] J. L. Flanagan, Speech Analysis Synthesis and Perception, Springer-Verlag, W. H Freeman and Company, New York, U.S.A., 1964.
- [4] George A. Miller, Language and Speech, W. H Freeman and Company, New York, U.S.A., 1981.
- [5] 古井貞熙, ディジタル音声処理, 東海大学出版会, 東京, 1985 年.
- [6] 中川聖一, 確率モデルによる音声認識, 社団法人電子情報通信学会, 東京, 1988 年.
- [7] Sadaoki Furui, Digital Speech Processing, Synthesis, and Recognition, Marcel Dekker, Inc., New York, U.S.A., 1989.
- [8] Alex Waibel, Kai-Fu Lee, Readings in Speech Recognition, Morgan Kaufmann Publishers, Inc., Palo Alto, California, U.S.A., 1990.
- [9] 上坂吉則, 尾関和彦, パターン認識と学習のアルゴリズム, 株式会社文一総合出版, 東京, 1990 年.
- [10] デイヴィッドクリスタル著, 風間喜代三／長谷川欣佑監訳, 言語学百科事典, 株式会社大修館書店, 東京, 1992 年.
- [11] Lawrence Rabiner, Biing-Hwang Juang, Fundamentals of Speech Recognition, PTR Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1993.
- [12] Lawrence Rabiner, Biing-Hwang Juang 共著, 古井貞熙監訳, 鹿野清宏, 嵐山茂樹, 松岡達雄, 南泰浩, 松井知子, 高橋敏, 山田智一, 吉岡理, 訳, 音声認識の基礎(上)(下), NTT アドバンステクノロジ株式会社, 東京, 1995 年.
- [13] 鹿野清宏, 中村哲, 伊勢史郎, 音声・音情報のディジタル信号処理, デジタル信号処理シリーズ 5, 株式会社昭晃堂, 東京, 1997 年.

- [14] Frederick Jelinek, Statistical Methods for Speech Recognition, The MIT Press, Cambridge, Massachusetts, U.S.A., 1997.
- [15] 古井貞熙, 音声情報処理, 電子情報通信工学シリーズ, 森北出版株式会社, 東京, 1998年.
- [16] 鹿野清宏, 伊藤克亘, 河原達也, 武田一哉, 山本幹雄, IT Text 音声認識システム, 株式会社オーム社, 東京, 2001年.
- [17] 小野斉大, 田谷文彦, 三澤明, UNIX コマンドブック第2版, ソフトバンククリエイティブ株式会社, 東京, 2003年.
- [18] 安藤彰男, リアルタイム音声認識, 社団法人電子情報通信学会, 東京, 2003年.
- [19] 古井貞熙, 酒井善則, ねっとテクノロジー解体新書(5) 画像・音声処理技術, 株式会社電波新聞社, 東京, 2004年.
- [20] 板橋秀一編著, 音声工学, 森北出版株式会社, 東京, 2005年.
- [21] 古井貞熙, 新音響・音声工学, 株式会社近代科学社, 東京, 2006年.
- [22] 北岡教英, 山田武志, 枝植覚, 宮島千代美, 西浦敬信, 中山雅人, 傳田遊亀, 藤本雅清, 山本一公, 滝口哲也, 黒岩眞吾, 武田一哉, 中村哲, "CENSREC-1-C: 雜音下音声区間検出評価基盤の構築," 情報処理学会研究報告 SLP, 音声言語情報処理 2006(107), pp.1-6, 2006-10-20.
- [23] 荒木雅弘, フリーソフトでつくる音声認識システム, 森北出版株式会社, 東京, 2007年.
- [24] Sadaoki Furui, "Selected Topics from 40 Years of Research on Speech and Speaker Recognition," Proceedings of INTERSPEECH 2009, pp.1-8, Brighton, United Kingdom, September 6-10, 2009.
- [25] 古井貞熙, 人と対話するコンピュータを創っています—音声認識の最前線—, 株式会社角川学芸出版, 東京, 2009年.
- [26] 藤本雅清, "音声区間検出の基礎と最近の研究動向," 電子情報通信学会技術研究報告 SP, 音声 110(81), pp.7-12, 2010-06-10.
- [27] 尾関和彦, メディア情報処理の基礎数理, 共立出版株式会社, 東京, 2011年.
- [28] 平井有三, はじめてのパターン認識, 森北出版株式会社, 東京, 2012年.
- [29] 篠田浩一, 機械学習プロフェッショナルシリーズ「音声認識」, 講談社, 東京, 2017年.
- [30] 中川聖一編著, 小林聰, 峯松信明, 宇津呂武仁, 秋葉友良, 北岡教英, 山本幹雄, 甲斐充彦, 山本一公, 土屋雅稔, 音声言語処理と自然言語処理(増補), 株式会社コロナ社, 2018年.

- [31] 西川仁, 佐藤智和, 市川治著, 清水昌平編, テキスト・画像・音声データ分析, データサイエンス入門シリーズ, 株式会社講談社, 2020 年.
- [32] 久保陽太郎, 機械学習による音声認識, 日本音響学会編音響テクノロジーシリーズ 24, 株式会社コロナ社, 2021 年.
- [33] 高島遼一, Python で学ぶ音声認識, 機械学習実践シリーズ, 株式会社インプレス, 2021 年.
- [34] 滝口哲也編著, 音声 (上), 日本音響学会編音響学講座 6, 株式会社コロナ社, 東京, 2021 年.
- [35] 岩野公司編著, 音声 (下), 日本音響学会編音響学講座 7, 株式会社コロナ社, 東京, 2023 年.
- [36] 武田龍・高島遼一, 音源分離・音声認識, 株式会社コロナ社, 東京, 2024 年.
- [37] 伊藤克亘・宮澤幸希, Python で学ぶ音声コミュニケーション情報処理, 株式会社朝倉書店, 東京, 2025 年.
- [38] 荒木雅弘, イラストで学ぶ音声認識 改訂第 2 版, 株式会社講談社, 東京, 2025 年.

## 付録 A 数字単語音声のスペクトル分析実例

録音した音声データから単語音声の区間を正確に切り取るためには、音声の性質についての若干の知識が必要です。音声波形の表示と聴取だけでは単語の開始および終了時刻を見誤ることが多いので、録音した音声データのスペクトログラムを表示し、そのパターンの違いに基づいて単語音声の区間を定めてください。

実習に用いる数字単語音声の特徴を説明します。説明に用いるのは、高木の声で録音した音声の振幅を正規化し、WaveSurfer の Demonstration モードで表示した図です。声には性別、年齢、個人性による特徴があります。また、全く同じ発声を繰り返すことは不可能なので、同一人物が同じ単語を発音しても、音声波形やスペクトルの形は微妙に異なります。しかし、音素による波形やスペクトルの特徴は一定の性質を示します。

0 /rei/ 日本語のラ行音は音韻環境<sup>1</sup>によって特性が異なりますが、/rei/の/r/は図 A.1 (a) のようなパターンを示します。波形は無音声区間 (SIL) の背景雑音に比べて振幅が大きく、スペクトログラムでは、350 Hz, 2300 Hz 付近にエネルギーの高い場所があります。これは後続の/e/の影響によるものです。/ei/の部分は 2 種類の母音がなめらかに変化していく、500 Hz および 1300 Hz~3000 Hz に強い成分が含まれています。この単語の終端位置は、スペクトログラムでフォルマントの表示が弱まって見えなくなるところに定めます。

1 /ichi/ 子音/ch/が母音/i/に挟まれている単語です。この例では、1 つ目の/i/に比べて 2 つ目の/i/が長く強く発音されていることがわかります(図 A.1 (b))。/ch/は無声破擦子音と呼ばれる子音の 1 つで、(i) 先行母音からの過渡部、(ii) 無音部(閉鎖部)、(iii) 破裂部、(iv) 摩擦部、(v) 後続母音への過渡部からなります。図では (i) および (ii) を合わせて c1, (iii)~(v) を ch としています。音の無い c1 の部分も子音の一部であることに注意してください。母音/i/の第 1 フォルマントは 400 Hz 付近、第 2 フォルマントは 2300 Hz 付近にあります。

2 /ni/ ナ行の鼻子音/n/の部分では 500 Hz 付近に共鳴が見られますが、波形やスペクトログラムだけでは/r/との区別は難しい場合があります。鼻音化の現象は後続の/i/にも影響しています。単語の開始点および終了点は図 A.1 (c) のように、スペクトログラム上で共鳴ピークの表示の端に定

---

<sup>1</sup>前後にどのような音韻(音)があるかということ。

めてください。録音した音声には雑音が含まれているので、音声波形だけでは/n/の開始点や/i/の終了点を決めるのが難しいことがわかります。

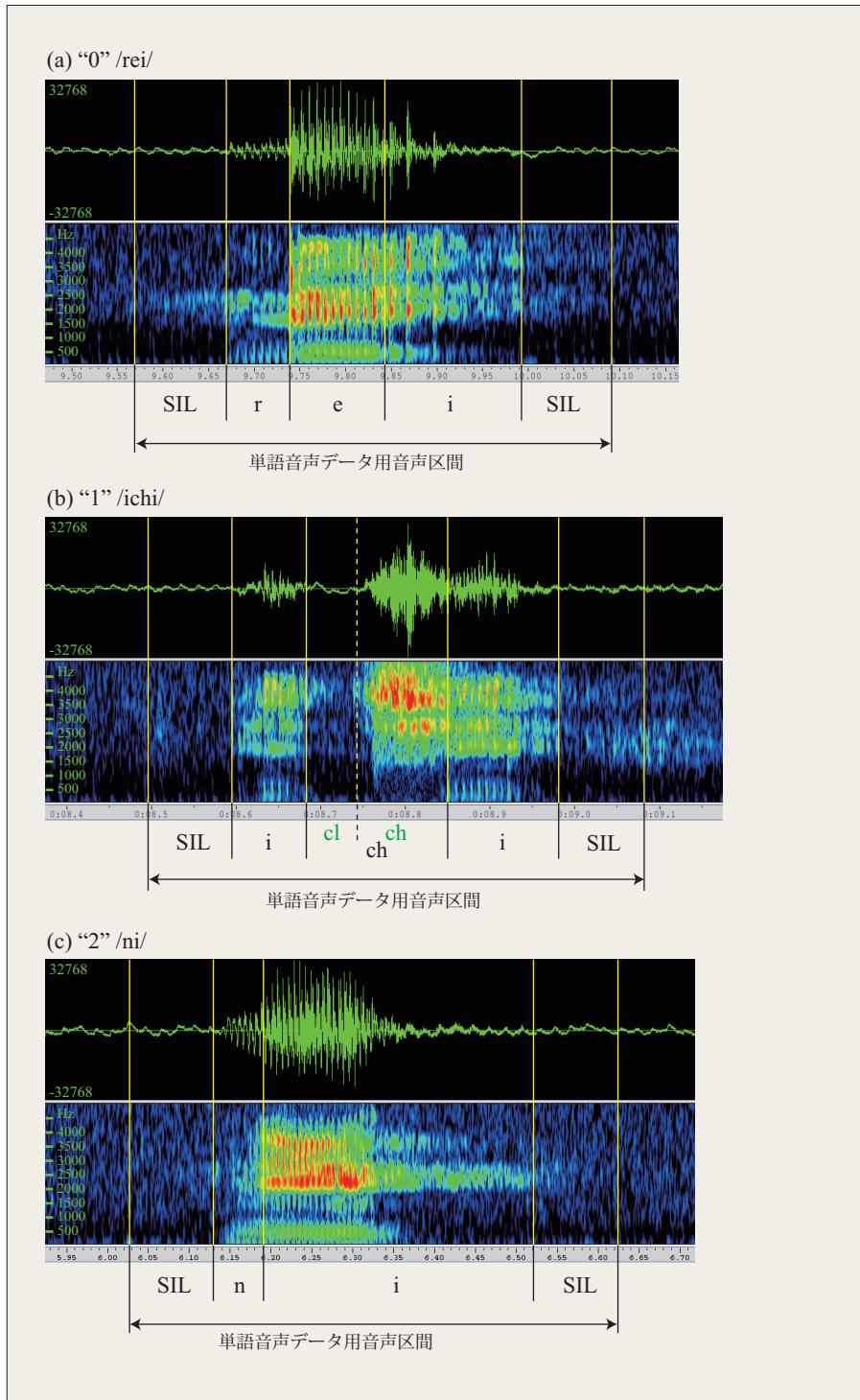


図 A.1: 単語 HMM 学習用音声データの例 (0, 1, 2).

3 /saN/ この単語の最初の音は無声摩擦子音という子音です。/s/はその1つで、日本語の場合「さ」「す」「せ」「そ」で使われます。定常的で比較的安定しています。波形の振動は非常に細かく、スペクトルには3000 Hz以上の高域に強いパワーの成分が現れます。母音/a/の第1フォルマントは650 Hz、第2フォルマントは1400 Hz付近にあります。/i/のフォルマント(図A.1 (b) (c))と見比べてみてください。この単語発声の場合(図A.2 (a)) /N/のスペクトルパターンが長く継続しているように見えますが、波形の観察と音声聴取により無音(SIL)との境界を決めました<sup>2</sup>。

4 /yoN/ 最初の音は半母音と呼ばれる音素/y/です。音声波形もスペクトルも母音と同様の性質を示します。/yo/は/yo/を速く発音した場合の音に似たパターンを示します(図A.2 (b))。それは/y/の前半のフォルマントが/i/に近いことからもわかります。最後の/N/は/saN/の/N/と同じパターンを示しています。

5 /go/ /g/は有声破裂音と呼ばれる子音の1つです。声帯が振動しているので、低い周波数帯域にエネルギーの強い部分があります(図A.2 (c))。日本語の有声破裂音には他に/b/、/d/があり、/g/との違いは、後続母音の種類によって区別することができますが、詳しくは関連図書を見てください。単語開始部はスペクトログラム表示で/g/のエネルギーが現れ始める位置、終了部は/o/のエネルギーが消える位置に設定します。

---

<sup>2</sup>/saN/と発音して口を閉じたあとに、喉が余韻で振動していたようです。

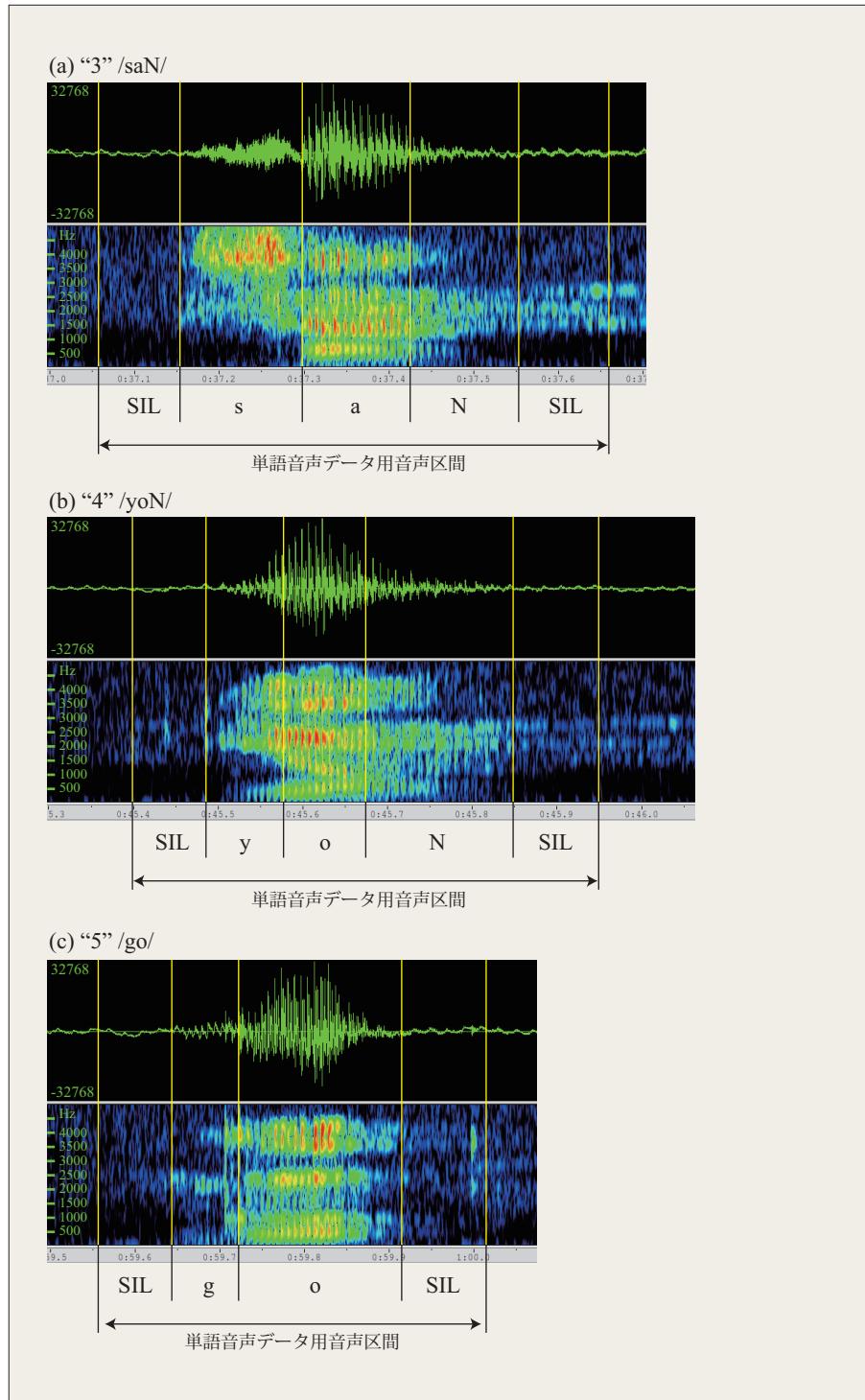


図 A.2: 単語 HMM 学習用音声データの例 (3, 4, 5).

6 /roku/ 最初の音素/r/は/rei/と同じパターンを示しています。この単語の母音は/o/と/u/です。ここまでで日本語の5母音の全てが揃いました。母音は音声波形にも特徴がありますが、スペクトルに現れるフォルマントの位置に特徴があります（2.2.1節）。無声破裂子音である/k/は/ch/（図A.1(a)）と同様に無音部(c1)や破裂部(k)からなります。単語開始点と終了点は/rei/と同様に定めればよいでしょう（図A.3(a)）。

7 /nana/ この単語の開始は/ni/と同じ鼻子音の/n/です。2番目の/n/は母音/a/に挟まれているため、1番目の/n/に比べて母音の共鳴の影響を強く受けています。音声に背景雑音が含まれるときには、音声波形だけで語頭/n/の開始点を決めるのは難しいことがあります。スペクトログラムを見たほうが良くわかります。

8 /hachi/ ハ行音の子音/h/は無声摩擦子音という種類の子音で、声帯が振動していないので低い周波数にエネルギーが無く、したがってスペクトログラムの低周波数帯にはパターンが現れていません。/ichi/で使われている無声破擦子音がこの単語でも使われています。/ichi/とは1音だけ異なっているので、/ichi/のスペクトログラム（図A.1(b)）ととても良く似たパターンになっています。

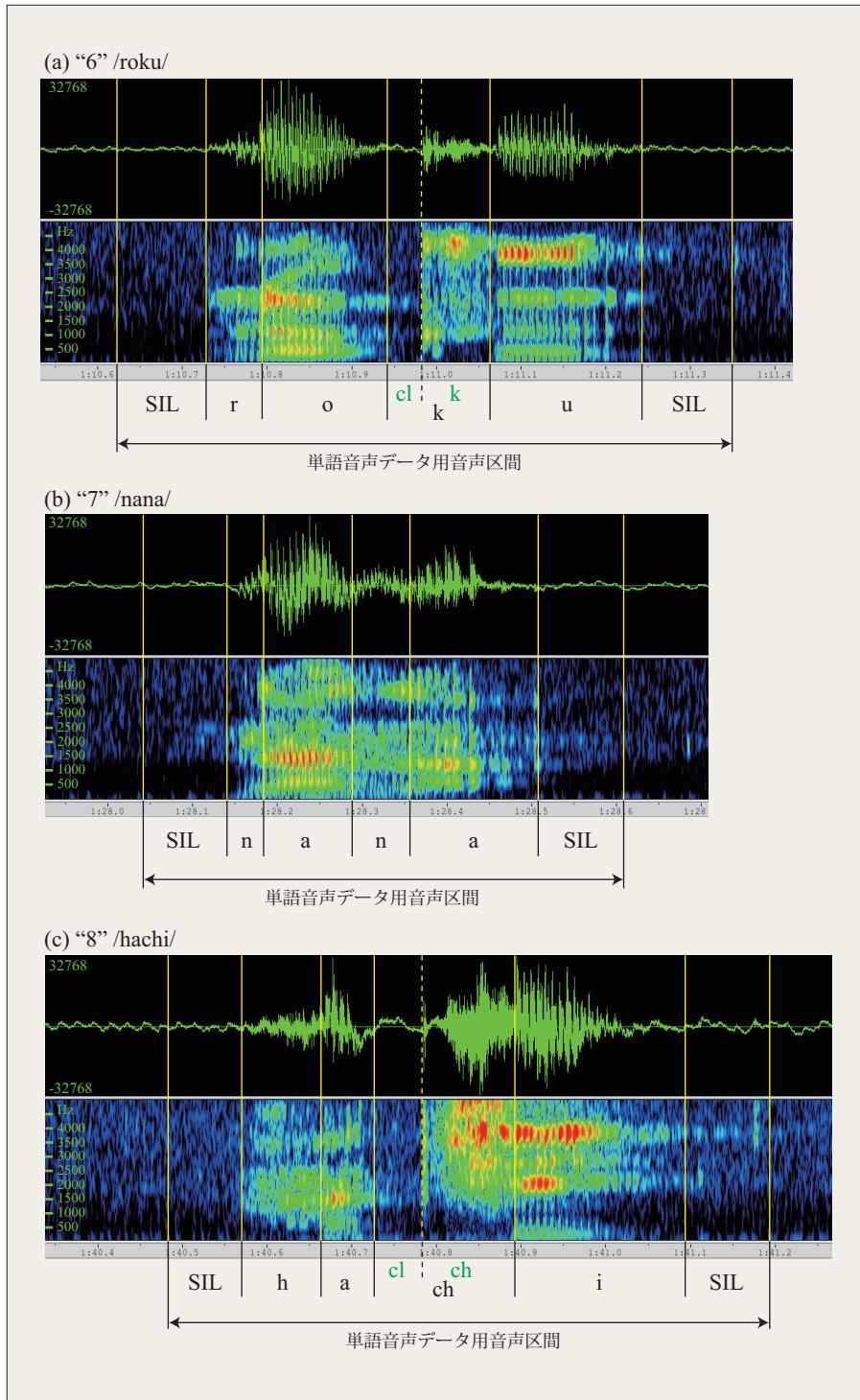


図 A.3: 単語 HMM 学習用音声データの例 (6, 7, 8).

9 /kyuu/ 無声破裂子音/k/で始まっている単語です。/roku/の場合は母音が先行していたので、閉鎖部が判別できましたが、語頭の破裂子音の閉鎖部は音声信号からはわかりません。したがってこのような場合は破裂部の開始を単語の開始とします。スペクトログラムには開始が明確に現れています。/yoN/の場合と同様に、半母音/y/は/i/に類似のスペクトルパターンを示します。/uu/は長音ですが、短音である6の/u/（図A.3(a)）と継続長はほとんど同じです。実は、長音と短音の区別は単に継続長で区別できるものではありません。この区別は、聴き手の母国語に強く影響されます。つまり、日本語のように音の長短が単語の意味に関わることがある言語を母国語とする人は長音と短音の違いが簡単に分かるのですが、英語や中国語のように音の長短で単語の意味を区別しない言語を母国語とする人は日本語の上級者でない限り、この2つの「う」の違いを区別するのは困難です。みなさんの音声データの6の/u/と9の/uu/ではいかがでしょう？継続長を比べてみてください。

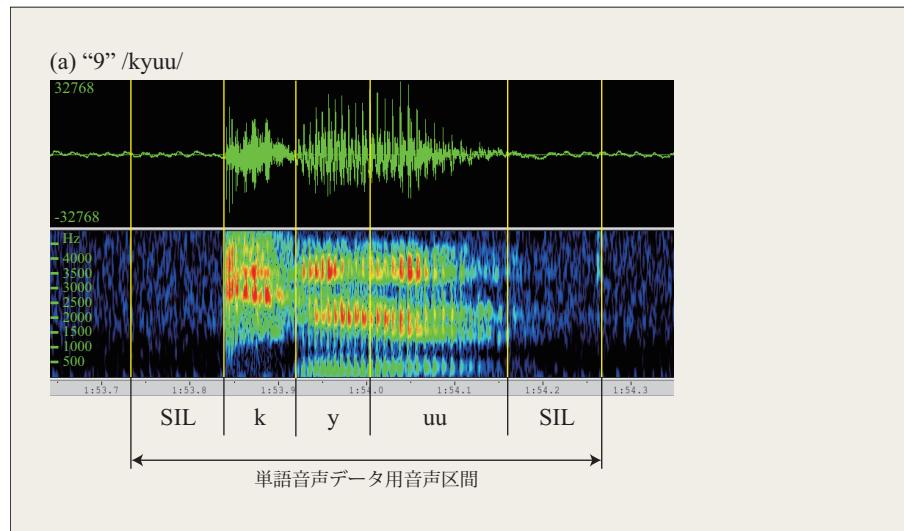


図 A.4: 単語 HMM 学習用音声データの例 (9).

## 付録B 実習用ファイル

`asr` : この実習の最上位ディレクトリ

`.wavesurfer` : WaveSurfer の環境設定ディレクトリ

`addpath` : 実習用コマンドのパス設定スクリプト

`drill` : HMM 算法の実習作業ディレクトリ

`Makefile` : ファイル生成手順、プログラム構成ファイルの関係

`backward.c` : Backward アルゴリズム関数 [穴埋め]

`baumwelch.c` : Baum-Welch アルゴリズム関数 [穴埋め]

`data1` : 信号源 ( $HMM_1$ ) の認識データ格納用ディレクトリ

`data2` : 信号源 ( $HMM_2$ ) の認識データ格納用ディレクトリ

`drawLC_d1` :  $HMM_1$  の学習曲線を描く gnuplot スクリプト

`drawLC_d2` :  $HMM_2$  の学習曲線を描く gnuplot スクリプト

`drawR` : HMM パターン認識結果の図を描く gnuplot スクリプト

`drillB.c` : Backward アルゴリズム実行

`drillF.c` : Forward アルゴリズム実行

`drillR.c` : HMM によるパターン認識シミュレーション

`drillT.c` : HMM による時系列パターン学習

`forward.c` : Forward アルゴリズム関数 [穴埋め]

`gen.c` : 信号源 ( $HMM_1$  または  $HMM_2$ ) から時系列データを生成

`genData` : HMM によるパターン認識シミュレーションデータの生成

`paramFB.txt` : 動作確認用 HMM パラメータ

`sound` : 音声入出力機能

`Makefile` : ファイル生成手順、プログラム構成ファイル同士の関係

`_vu.c` : 入力音声表示

`ad2txt16.c` : 音声波形振幅値表示（波形描画用）

`audioIN` : 音声入力（録音）シェルスクリプト

`hupAudioIN` : 音声入力プロセス終了シェルスクリプト

`vad.c` : 音声区間検出

`vu` : 入力音声レベル表示シェルスクリプト

`studentbook.pdf` : この実験指導書

`wrecog` : 単語音声認識実習作業ディレクトリ

`drawLC`： 単語 HMM 学習曲線描画 gnuplot スクリプト  
`hmm`： 学習済み HMM パラメータ格納ディレクトリ  
`lib`： 単語 HMM 学習・認識用ファイル情報格納ディレクトリ  
`log`： 単語 HMM 学習・認識過程記録格納ディレクトリ  
`mfcc`： MFCC ファイル格納ディレクトリ  
`program`： 単語認識プログラム格納ディレクトリ  
`Makefile`： ファイル生成手順、プログラム構成ファイルの関係  
`_recog.c`： On-The-Fly 単語認識  
`ad2fb.c`： フィルタバンク計算関数 [穴埋め]  
`ad2mfcc.c`： 音声波形から MFCC を計算し CMVN 処理を行う  
`countCorr`： 単語 HMM 学習検証用シェルスクリプト  
`drawingSpec`： 音声スペクトル描画用 Perl スクリプト  
`drillG.c`： 多次元ガウス確率密度関数計算  
`fb.c`： フィルタバンク計算  
`fb2mfcc.c`： MFCC 計算関数 [穴埋め]  
`gpdf.c`： 多次元ガウス確率密度関数 [穴埋め]  
`mfcc.c`： On-The-Fly 単語認識 MFCC 計算  
`mfccf.c`： 音声波形ファイル対象 MFCC 計算  
`prtmfcc.c`： MFCC の値の表示  
`recog`： On-The-Fly 音声認識コマンドシェルスクリプト  
`recogf.c`： 単語認識 (MFCC ファイル対象、バッチ処理用)  
`train.c`： 単語 HMM 学習  
`vtb.c`： Viterbi 計算  
`result`： 認識結果格納ディレクトリ  
`sample`： 動作確認用データ格納ディレクトリ  
`sample.fb`： フィルタバンク計算の正解 (プリント形式)  
`sample.mfcc`： MFCC 計算の正解  
`sample.mfcc.txt`： MFCC 計算の正解 (プリント形式)  
`sample.wav`： 正解計算用音声波形ファイル  
`sampleG.hmm`： 多次元ガウス確率密度関数検算用 HMM  
`sampleG.mfcc`： 多次元ガウス確率密度関数検算用 MFCC  
`vadwav`： On-The-Fly 検出された音声波形ファイル格納ディレクトリ  
`wav`： 音声波形ファイル格納ディレクトリ

