МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (МАИ)

Факультет №3 **Кафедра** 304

Направление подготовки «Программная инженерия» **Группа** 30-411Б

Квалификация (степень) Бакалавр

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ ДИПЛОМНОМУ ПРОЕКТУ

На тему: Разработка ПО для системы верификаци пользователя по его голосу

Автор квалификационной работы Ерохин Максим Владимирович _____
(Фамилия Имя Отчество) (подпись)

Руководитель к.т.н. доцент Волков Вадим Иосифович _____
(Фамилия Имя Отчество) (подпись)

К о н с у л ь т а н т ы :

а) _____ _____
(Фамилия Имя Отчество) (подпись)

б) _____ _____
(Фамилия Имя Отчество) (подпись)

в) _____ _____
(Фамилия Имя Отчество) (подпись)

К з а щ и т е д о п у с т и т ь

Зав. кафедрой д.т.н. профессор Брехов Олег Михайлович _____
(Фамилия Имя Отчество) (подпись)

“___”_____ 20____г.

Москва 2015 г.

Содержание

Список основных специальных терминов с определениями, список аббревиатур и сокращений с расшифровкой 3
1. Введение 4
2. Специальная часть 7
2.1 Обзор прототипов и похожих разработок 7
2.2 Постановка задачи 8
2.3 Разработка метода ввода речевого сигнала в программу 11
2.4 Разработка метода выделения речевого сигнала. 11
2.4.1 Разбиение сигнала на фреймы 11
2.4.2 Удаление тишины 12
2.5 Разработка метода получения вектора признаков речевого сигнала 13
2.5.1 Оконная функция 13
2.5.2 Быстрое преобразование Фурье 14
2.5.3 Мел-шкала и банк фильтров 19
2.5.4 Дискретное косинусное преобразование 23
2.6 Разработка метода сравнения векторов признаков 24
2.7 Выводы по разделу 29
3. Технологическая часть 30
3.1 Разработка методики тестирования 30
3.1.1 Выбор программного обеспечения для реализации алгоритмов 30
3.1.2 Реализация алгоритмов 30
3.2 Тестирование 32
3.2.1 Автоматизированное тестирование 32
3.2.2 Анализ работы алгоритма на этапе тестирования 35
3.3 Определение погрешности системы 39
3.4 Выводы по разделу 39
4. Заключение 40
5. Список источников и литературы 41
Приложение А. Исходный код программы на языке C++ 42
Приложение Б. Журнал тестирования 66

Список основных специальных терминов с определениями, список аббревиатур и сокращений с расшифровкой

Верификация – это процедура проверки подлинности предъявленного пользователем идентификатора. Таким идентификатором может быть пароль, ключ, голос, изображение сетчатки глаза, отпечаток пальца и многое другое.

Фреймы – небольшие фрагменты сигнала, длительностью несколько десятых секунды. Фрейм является основной единицей обработки оцифрованного сигнала.

Отсчет – численное значение амплитуды сигнала в определенный момент времени.

ОС – операционная система

ПО – программное обеспечение

ДПФ – дискретное преобразование Фурье

БПФ – быстрое преобразование Фурье

1. Введение

В эпоху бурного развития технологий, проблемы информационной защиты встают наиболее остро. С массовым внедрением компьютеров во все сферы деятельности человека, объем информации, хранящейся в электронном виде, вырос в тысячи раз, а с появлением компьютерных сетей даже отсутствие физического доступа к компьютеру перестало быть гарантией сохранности информационных ресурсов. Использование автоматизированных систем обработки информации и управления обострило проблему защиты информации, от несанкционированного доступа. Основные проблемы защиты информации в компьютерных системах возникают из-за того, что информация не является жёстко связанной с носителем. Её можно легко и быстро скопировать и передать по каналам связи.

В современном мире для решения этой проблемы часто применяются системы безопасности с использованием биометрических признаков для аутентификации пользователя. В качестве одного из биометрических признаков можно использовать индивидуальные спектральные характеристики человеческого голоса. В отличие от таких биометрических признаков, как отпечатки пальцев, отпечаток ладони, рисунок сетчатки глаза, анализ характеристик голоса человека не требует специального (и зачастую дорогостоящего) оборудования, и может легко интегрироваться в существующие системы безопасности. Также этот метод может использоваться для верификации личности человека на расстоянии, например, при телефонном разговоре или при анализе записанной речи. Большинство современных систем с задачей идентификации и верификации пользователя по его голосу либо имеют закрытый исходный код, дороги в использовании и в обслуживании, либо представлены на языках высокого уровня, что является проблемой использования их на многих современных платформах.

В данной выпускной квалификационной работе будет реализован и протестирован алгоритм верификации личности пользователя по голосу на основе сравнения мел-кепстральных спектральных коэффициентов.

Результат разработки представлен в виде программы на языке C++.

C++ — компилируемый статически типизированный язык программирования общего назначения. Так как C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков, программа может работать быстрее, чем программы на большинстве других современных языков. Количество платформ которые поддерживает C++ трудно переоценить.

Список платформ, поддерживаемых современными версиями C++

- Windows x86/x64
- Linux x86/x64
- Платформы на процессорах ARM x86/x64
- Arduino
- Многие другие

Если учитывать последние вышедшие стандарты языка C++ — C++11 и C++14, можно сказать, что программировать на нем стало еще удобнее, а среда Microsoft Visual Studio 2013 всячески помогает пользователю в этом.

Также, впоследствии, код на C++ можно легко улучшить, добавить дополнительные функции. Так как C++ сейчас является одним из самых популярных и поддерживаемых языков программирования, можно быть уверенным, что и программу выполненную на этом языке можно будет всегда использовать и поддерживать.

Также код на языке C++ может быть встроен в программу как для мобильных устройств, встраиваемых систем, программиируемых микроконтроллеров, так и использоваться для таких систем, как Linux и Windows.

В данной пояснительной записке для графического отображения значений векторов и матриц будет использоваться Matlab функция imagesc(), отображающая значения элементов матрицы с помощью шкалы цветовых температур. Если параметрами функции не определено иное, то для отображения матрицы находятся

минимальный и максимальный элементы, и значениям на отрезке от минимального до максимального линейно сопоставляются цвета, приведенные на рис. 1.1.

Рис. 1.1 Шкала цветовых температур

Пример значений матрицы и ее отображения с помощью шкалы цветовых температур приведен на рис. 1.2.

Такое отображение позволяет визуально находить в матрицах минимальные и максимальные значения, области близких значений.

0.1622	0.6020	0.4505	0.8258	0.1067
0.7943	0.2630	0.0838	0.5383	0.9619
0.3112	0.6541	0.2290	0.9961	0.0046
0.5285	0.6892	0.9133	0.0782	0.7749
0.1656	0.7482	0.1524	0.4427	0.8173

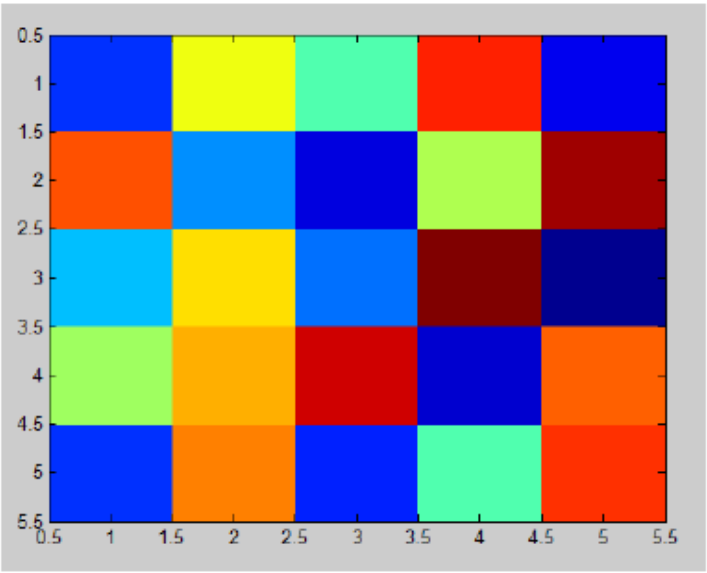

Рис.1.2 Отображение матрицы с помощью шкалы цветовых температур

2. Специальная часть

Цель данного раздела – выбор и описание алгоритмов для реализации верификации личности пользователя по голосу, описание теоретических основ выбранных алгоритмов и определение их оптимальных параметров.

2.1 Обзор прототипов и похожих разработок

1. На основании доклада ООО «Форенэкс» «Судебная экспертиза звукозаписей» можно сделать вывод о существовании следующих программных систем с назначением, схожим с разрабатываемой в данной работе:

- OTExpert – программный комплекс криминалистического исследования фонограмм речи;
- Justiphone – программный комплекс криминалистического исследования фонограмм речи;
- Phonexi – программный комплекс для криминалистической идентификации говорящего по фонограммам устной речи на основе методики «Диалект»;
- «Сапфир» – программный комплекс для криминалистической идентификации говорящего по фонограммам устной речи на основе методики «Диалект»;
- «Этнос» - программный комплекс для идентификации говорящего по фонограммам устной речи на различных языках;
- «ИКАР-Лаб» - программный комплекс для криминалистического исследования фонограмм и идентификации говорящего по фонограммам устной речи на основе спектрально-формантных методов.

Так как перечисленные выше продукты применяются в области криминалистики, то информации по реализованным алгоритмам и их параметрам в публичном доступе нет.

2. HTK (Hidden Markov Model Toolkit) – набор инструментов для работы со скрытыми моделями Маркова, используется как для распознавания речи, так и для

других задач, связанных с обработкой речевых сигналов. Имеет реализацию алгоритма получения мел-кепстральных коэффициентов, что может использоваться для идентификации и верификации личности.

3. HTK MFCC MATLAB – реализация алгоритма получения мел-кепстральных коэффициентов, используемого в HTK, на языке Matlab.

4. Speaker-recognition – проект студентов университета Маастрихта на тему идентификации личности пользователя по голосу. Реализован на языке Matlab.

5. VoiceKey – биометрическая платформа для подтверждения личности по голосу в телефонном канале, WEB-кабинете и мобильном приложении. Так как платформа ориентирована на бизнес-клиентов ее программный код закрыт, а использование платное.

2.2 Постановка задачи

В соответствии с техническим заданием требуется реализовать систему верификации личности пользователя по голосу. Назначение системы – верификация – означает режим распознавания «один к одному», когда определяется степень схожести образцов голоса признакам одного пользователя. Любой речевой сигнал можно представить как вектор в каком-либо параметрическом пространстве. Эти вектора называются векторами признаков, и между собой сравниваются. Так как верификация выполняется при произнесении короткой ключевой фразы, то система является текстозависимой.

Для выполнения заданных функций система должна иметь 2 режима работы: обучение и верификация. Оба режима используют общую функциональную часть для получения вектора признаков речевого сигнала, который в дальнейшем запоминается либо используется для сравнения.

Для решения поставленной задачи могут использоваться следующие методы:

- Спектрально-формантный метод,
- Метод линейного предсказания,

- Метод выделения динамических характеристик,
- Метод получения мел-кепстральных частотных коэффициентов.

Отличительной особенностью метода получения мел-кепстральных частотных коэффициентов является независимость полученного вектора от длины исходного голосового образца, его относительно малый размер и учет в нем разброса индивидуальных характеристик голосового тракта идентифицируемого субъекта. Мел-кепстральные частотные коэффициенты это своеобразное представление энергии спектра сигнала. Плюсы использования этого метода заключаются в следующем:

- Спектр проецируется на специальную мел-шкалу, позволяя выделить наиболее значимые для восприятия человеком частоты.
- Количество вычисляемых коэффициентов может быть ограничено любым значением (например, 29), что позволяет “сжать” фрейм и, как следствие, количество обрабатываемой информации.

В качестве основы данной работы была выбрана реализация получения мел-кепстральных коэффициентов HFCC FB-29, имеющая следующие параметры:

- Количество мел-кепстральных коэффициентов: 29,
- Анализируемый диапазон: [100, 6250] Гц,
- Коэффициент ширины фильтров: 0.75,
- Выравнивание коэффициентов отсутствует.

Схема получения вектора признаков изображена на рис. 2.1.

Минимальная разрядность данных в данном проекте – 32 бита для данных с плавающей точкой.

Минимальная частота дискретизации по теореме Котельникова должна быть больше или равна удвоенной максимальной анализируемой частоте, и для выбранных параметров алгоритма составляет 12500 Гц. Так как увеличение частоты дискретизации влияет на используемые системныересурсы (прежде всего на объем

оперативной памяти, требуемой для работы алгоритма), имеет смысл также ограничить максимальную частоту дискретизации. В качестве оптимального значения частоты дискретизации из стандартных значений для аудиофайлов и звуковых карт стоит выбрать 16 кГц.

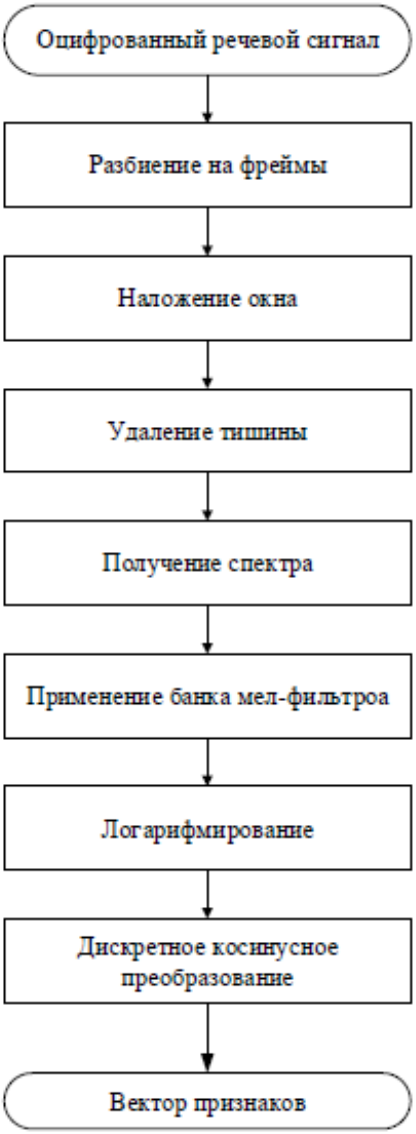

Рис. 2.1 Схема получения вектора признаков

Из недостатков этого метода можно отметить влияние на систему АЧХ микрофона и аналогового тракта звуковой карты, эмоционального и физического состояния пользователя.

2.3 Разработка метода ввода речевого сигнала в программу

Источником аудиоданных в ПК являеется аудиофайл. Данные в файле могут быть в WAV или PCM с плавающей точкой (IEEE Float). Для ввода данных из файла используется стандартный бинарный ввод из wav-файла. Программа разбирает заголовок WAV-файла и в соответствии с ним производит считывание аудиоданных. В результате имеется массив чисел с плавающей точкой длина которого определяется размером аудиофайла.

2.4 Разработка метода выделения речевого сигнала.

Для более удобного последующего преобразования исходных данных, сигнал следует разбить на небольшие фрагменты – фреймы, а для того, чтобы анализировать только сигнал с речью был разработан алгоритм удаления тишины.

2.4.1 Разбиение сигнала на фреймы

Для речевого сигнала имеет смысл анализировать не всю временную область, а небольшие фрагменты, длительностью несколько десятых секунды – фреймы. Для уменьшения потерь информации при разбиении сигнала на фреймы используется сдвиг начала фрейма назад относительно конца предыдущего фрейма, таким образом, что фреймы частично перекрываются. Схема разбиения сигнала на фреймы изображена на рис. 2.2.

Минимальная длина фрейма ограничена минимальной анализируемой частотой в соответствии с теоремой Котельникова, т.е. для получения корректных результатов на этапе выполнения дискретного преобразования Фурье необходимо, чтобы длина анализируемого фрейма была больше или равна периоду сигнала, соответствующего минимальной анализируемой частоте.

При увеличении длины фрейма снижается использование системных ресурсов (прежде всего оперативной памяти), но ухудшается точность системы. В данном дипломном проекте в соответствии со значением минимальной анализируемой частоты выбрана длина фрейма 100 мс и сдвиг 50 мс (50% перекрытие).

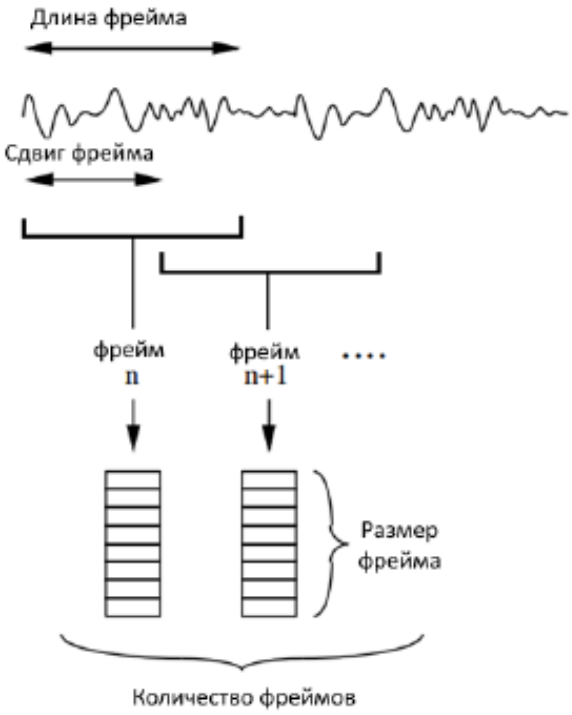

Рис. 2.2 Разбиение сигнала на фреймы

2.4.2 Удаление тишины

Для повышения точности системы, необходимо произвести сглаживание входного сигнала. Задача сглаживания — это, по сути, задача фильтрации сигнала от скачкообразных (ступенчатых) изменений. Считается, что полезный сигнал их не содержит. Ступенчатый сигнал за счёт множества резких, но небольших по амплитуде, перепадов уровня содержит высокочастотные составляющие, которых нет в сглаженном сигнале. Для этого использовался алгоритм скользящего среднего:

$$g_i = \frac{1}{2K+1} \sum_{j=-K}^{K} f_{i+j}$$

g_i – результат сглаживания f_{i+j} – исходный сигнал

От числа K зависит степень сглаживания сигнала. Чем больше K, тем более заметен результат сглаживания, однако при слишком большом K форма сигнала становится менее выразительной, поэтому следует выбрать оптимальное K, например 25. Также следует отметить, что на первых и последних точках i

невозможно вычислить значения сглаживания. Область где это можно сделать определяется следующим образом:

$$i = 1 + K, 2 + K, \dots, N - K.$$

Для уменьшения объема анализируемых данных и повышения точности системы имеет смысл анализировать только те фреймы, которые содержат речевой сигнал. Для этого производится операция удаления тишины. При высоком значении коэффициента сигнал/шум для решения данной задачи можно использовать метод подсчета средней энергии фреймов. Энергия фрейма подсчитывается как сумма квадратов амплитуд, деленная на количество отсчетов

$$E = \frac{\sum_{i=1}^{N}(S_i)^2}{N}$$

Далее считается среднее значение энергий всех фреймов

$$E_{mean} = \frac{\sum_{j=1}^{K} E_j}{K}$$

Далее выбирается граничное значение, например, 10% от E_{mean}, и отбрасываются фреймы с тишиной, то есть средняя энергия которых меньше этого значения.

$$threshould = E_{mean} * 0.1$$

2.5 Разработка метода получения вектора признаков речевого сигнала

В качестве вектора признаков в данном проекте выбрана матрица мел-кепстральных коэффициентов HFCC всех фреймов, содержащих полезный сигнал.

2.5.1 Оконная функция

Спектральный анализ, в теории, предназначен для анализа непрерывных периодических сигналов. При обрезании сигнала, в спектре появляются несуществовавшие в сигнале высокочастотные составляющие. Чтобы бороться с их появлением и прибегают к использованию т.н. оконных функций, изменяющих оригинальный сигнал в каждом анализируемом окне (фрейме). Для этой цели могут использоваться различные оконные функции – окно Хэмминга, окно Ханна, окно

Кайзера и другие. Искажения, вносимые применением окон, определяются размером окна и его формой.

В нашем случае будет использоваться окно Хэмминга. Результатом станет выделение центральной части кадра и плавное затухание амплитуд на его краях. Это необходимо для достижения лучших результатов при прогонке преобразования Фурье, поскольку оно ориентировано на бесконечно повторяющийся сигнал. Окно Хэмминга имеющее формулу:

$$w(n) = 0.55 - 0.46\cos(\frac{2\pi n}{N-1})$$

где n – номер отсчета, N – общее количество отсчетов.

И вид, изображенный на рис.2.3.

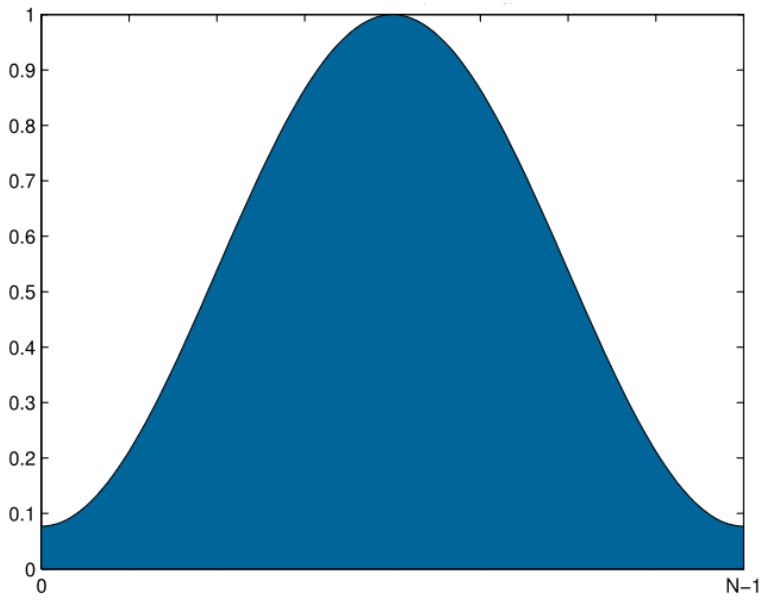

Рис. 2.3 Окно Хэмминга

2.5.2 Быстрое преобразование Фурье

Для получения спектра фрейма, необходимо применить к нему дискретное преобразование Фурье. Если осуществить дискретное преобразование Фурье для большого числа данных, то вычисление займет продолжительное время. Для увеличения скорости преобразования в данной работе был реализован алгоритм быстрого преобразования Фурье.

$$X(k) = \sum_{n=0}^{N-1} x_n e^{-i2\pi k\frac{n}{N}}$$

Где k = 0, ..., N-1, а X(k) – k-ая комплексная амплитуда

X(k) раскладывается на реальные и мнимые части:

$$X(k) = re_k + j * im_k$$

Быстрое преобразование Фурье – это алгоритм вычисления, который успешно использует свойства периодичности тригонометрических функций для того, чтобы избежать ненужных вычислений в дискретном преобразовании Фурье. И в случае БПФ, и в случае ДПФ результат вычислений один и тот же. Отличаются они только методом вычисления. В отличие от простейшего алгоритма ДПФ, который имеет сложность порядка O(N^2), БПФ имеет сложность всего лишь O($\mathrm{Nlog}_2 N$)

Рассмотрим подробнее алгоритм быстрого преобразования Фурье. Пусть число данных равно N, тогда некоторое комплексное число w представим в виде:

$$w = e^{-j(2\pi/N)}$$

В этом случае в дискретном преобразовании Фурье для ряда значений сигнала $\{f_0, f_1, \dots, f_{N-1}\}$ выражаются соотношением:

$$C_k = \frac{1}{N}\sum_{i=0}^{N-1} w^{ki} f_i$$

Для случая N=8 коэффициенты Фурье будут C_k выглядеть так:

$$C_k = \frac{1}{8}(w^0 f_0 + w^k f_1 + w^{2k} f_2 + w^{3k} f_3 + w^{4k} f_4 + w^{5k} f_5 + w^{6k} f_6 + w^{7k} f_7)$$

Записывать каждый коэффициент в таком виде довольно хлопотно, поэтому избавимся от 1/8 в записи коэффициентов. А значит впредь под коэффициентом C_k будем подразумевать NC_k.

Теперь распишем все коэффициенты $\{C_0, C_1, \dots, C_7\}$. Степенной ряд w представим в виде матрицы. (Рис. 2.4)

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \end{bmatrix} = \begin{bmatrix} w^0 & w^0 & w^0 & w^0 & w^0 & w^0 & w^0 & w^0 \\ w^0 & w^1 & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ w^0 & w^2 & w^4 & w^6 & w^8 & w^{10} & w^{12} & w^{14} \\ w^0 & w^3 & w^6 & w^9 & w^{12} & w^{15} & w^{18} & w^{21} \\ w^0 & w^4 & w^8 & w^{12} & w^{16} & w^{20} & w^{24} & w^{28} \\ w^0 & w^5 & w^{10} & w^{15} & w^{20} & w^{25} & w^{30} & w^{35} \\ w^0 & w^6 & w^{12} & w^{18} & w^{24} & w^{30} & w^{36} & w^{42} \\ w^0 & w^7 & w^{14} & w^{21} & w^{28} & w^{35} & w^{42} & w^{49} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix}$$

Рис. 2.4

Итак, понятно, что произведение данной матрицы на вектор, представляющий собой ряд значений сигнала, является соотношением, определяющим коэффициенты Фурье. Для того чтобы вычислить коэффициенты Фурье этим способом, необходимо произвести 8*8 = 64 умножения и 7*8=56 сложений. Обобщая, можно сказать, что в вычислении ДПФ для N значений необходимо умножить N^2 раз и сложить $(N-1) * N$ раз. Если N невелико, то объем вычислений тоже мал, но если, например, N = 1 000, то число операций достигает 1 000 000.

Также в степенном ряду w скрыта некоторая закономерность. Из Рис. 2.5 ясно, что

$$w^8 = w^0$$

$$w^9 = w^1$$

$$w^{10} = w^2$$

$$\dots$$

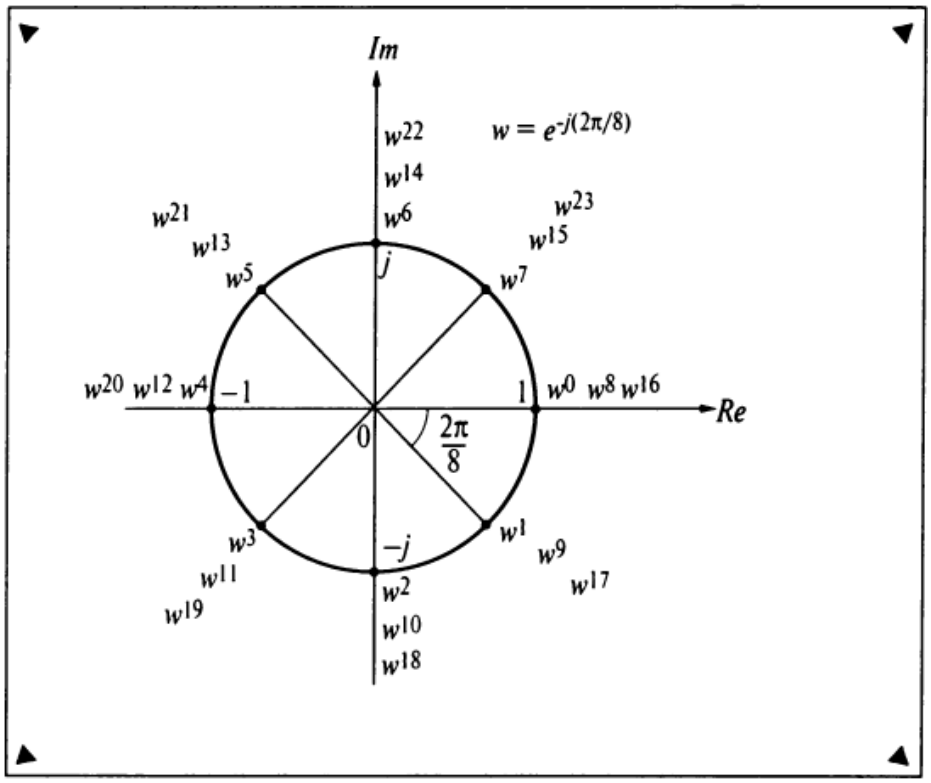

Рис. 2.5

То есть значения w, начиная с w^8, равны соответствующему значению от w^0 до w^7.

Учитывая эту закономерность матрица будет выглядеть так (Рис. 2.6):

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \end{bmatrix} = \begin{bmatrix} w^0 & w^0 & w^0 & w^0 & w^0 & w^0 & w^0 & w^0 \\ w^0 & w^1 & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ w^0 & w^2 & w^4 & w^6 & w^0 & w^2 & w^4 & w^6 \\ w^0 & w^3 & w^6 & w^1 & w^4 & w^7 & w^2 & w^5 \\ w^0 & w^4 & w^0 & w^4 & w^0 & w^4 & w^0 & w^4 \\ w^0 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ w^0 & w^6 & w^4 & w^2 & w^0 & w^6 & w^4 & w^2 \\ w^0 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix}$$

Рис. 2.6

Таким образом, если правильно переставить элементы матрицы, то число операций умножения уменьшится. Именно эти закономерности использует алгоритм БПФ.

Быстрое преобразование Фурье принимает на вход 2^n отсчетов, поэтому если количество отсчетов неравно степени двойки, то необходимо дополнить исходные данные нулями до следующей степени двойки. Так как спектр симметричен относительно середины, то половина его отсчетов отбрасывается. Быстрое преобразование Фурье возвращает и реальную и мнимую часть спектра, для дальнейшей работы необходимо вычислить модуль каждого комплексного числа спектра.

$$M(k) = \sqrt{re_k^2 + im_k^2}$$

k = 0, ..., N-1 – номер отсчета

M(k) – Модуль комплексного числа.

re – реальная часть спектра

im – мнимая часть спектра

Пример получаемого спектра приведен на рис. 2.7.

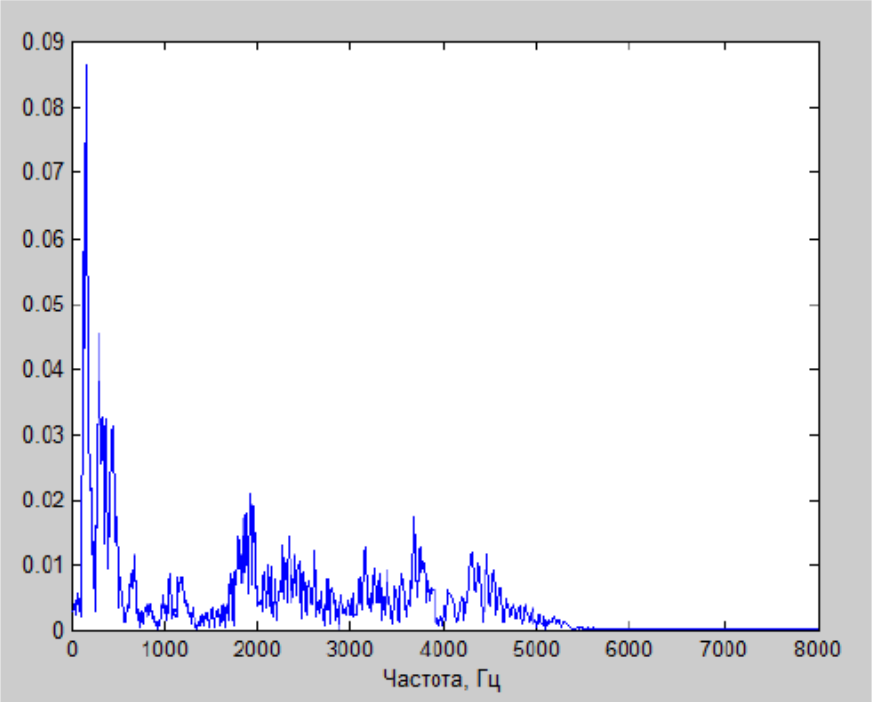

Рис. 2.7 Спектр фрейма

2.5.3 Мел-шкала и банк фильтров

На сегодняшний день наиболее успешными являются системы распознавания голоса, использующие знания об устройстве слухового аппарата. Результаты исследований показывают, что человеческое ухо интерпретирует звуки не линейно, а в логарифмическом масштабе. Использование модели восприятия звука человеком в системах распознавания и обработки речевых сигналов повышает точность работы таких систем.

Результатом исследования субъективного восприятия человеком звуковых частот является мел – психофизическая единица высоты звука, и мел-шкала (Рис. 2.8).

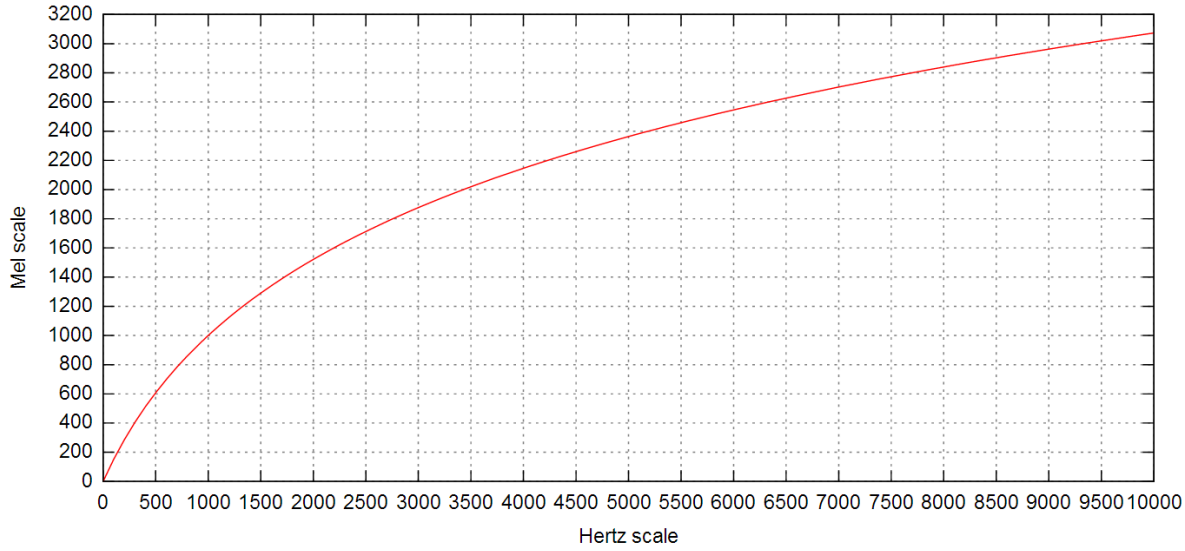

Рис. 2.8 Мел шкала

Далее, для вычичсления мел-кепстральных частотных коэффициентов нужно разложить полученный выше спектр сигнала по мел-шкале. Для этого нам потребуется создать «гребенку» фильтров. По сути, каждый мел-фильтр это треугольная оконная функция, которая позволяет просуммировать количество энергии на определённом диапазоне частот и тем самым получить мел-коэффициент. Зная количество мел-коэффициентов и анализируемый диапазон частот мы можем построить набор таких фильтров (рис. 2.9).

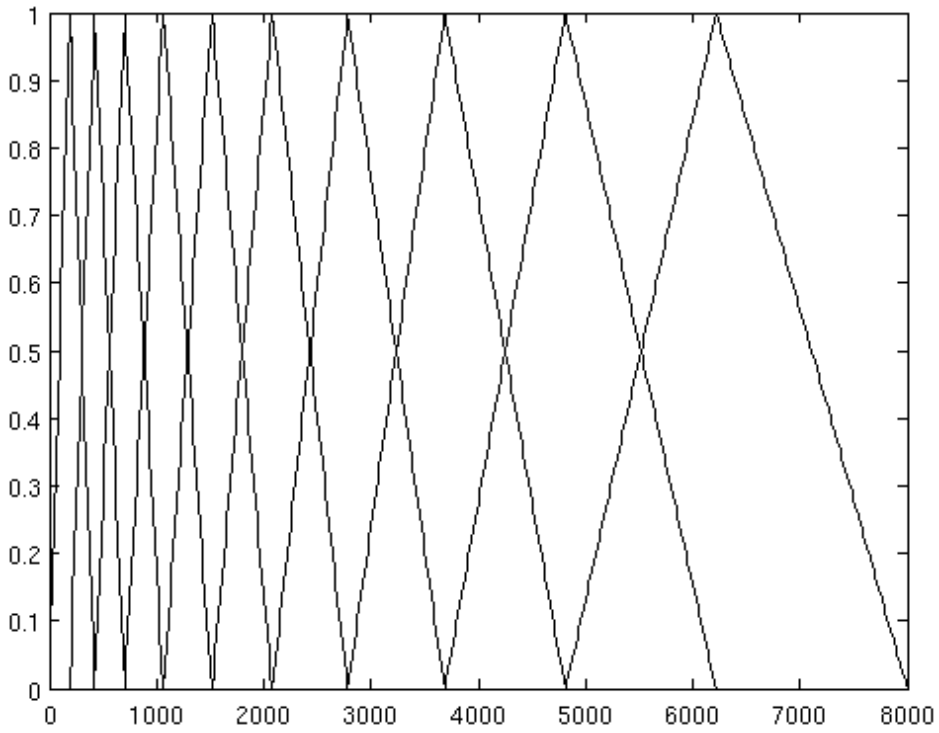

Рис. 2.9. «Гребенка» фильтров

Для перевода частоты в высоту звука и обратно используются формулы

$$m = 1127 \ln(1 + \frac{f}{700})$$

$$f = 700(e^{\frac{m}{1127}} - 1)$$

m – высота звука

f – частота звука

На основе этих преобразований формируется банк фильтров, линейно распределенных на мел-шкале в выбранном для анализа диапазоне.

Алгоритм формирования банка фильтров HFCC-E:

1. Выбираются границы анализируемого диапазона частот f_{low}и f_{high}, количество фильтров M, коэффициент масштабирования ширины фильтров ERB.

2. Вычисляются центральные частоты первого и последнего фильтров

$$f_{c_i} = \frac{1}{2}(-\bar{b} + \sqrt{\bar{b}^2 - 4\bar{c}}),$$

где $b = \frac{b-\hat{b}}{a-\hat{a}}$ и $c = \frac{c-\hat{c}}{a-\hat{a}}$ индекс i = 1 или M соответственно, a, b, c – коэффициенты Глассберга и Мура:

$a = 6.23*10-6 * ERB$, $b = 93.39*10-3 * ERB$, $c = 28.52 * ERB$.

Для первого фильтра:

$$\hat{a} = \frac{1}{2}\frac{1}{700+f_{low}} \quad \hat{b} = \frac{700}{700+f_{low}} \quad \hat{c} = -\frac{f_{low}}{2}\left(1+\frac{700}{700+f_{low}}\right)$$

Для последнего фильтра:

$$\hat{a} = -\frac{1}{2}\frac{1}{700+f_{high}} \quad \hat{b} = -\frac{700}{700+f_{high}} \quad \hat{c} = -\frac{f_{high}}{2}\left(1+\frac{700}{700+f_{high}}\right)$$

3. Так как центры всех фильтров распределены равномерно на мел-шкале, то центральные частоты оставшихся фильтров легко рассчитываются по следующим формулам:

Расстояние между центрами соседних фильтров:

$$\Delta\hat{f} = \frac{\hat{f}_{c_M} - \hat{f}_{c_1}}{M-1}$$

где все частоты - в мелах.

Центральные частоты оставшихся фильтров считаются по формуле

$$\hat{f}_{c_i} = \hat{f}_{c_1} + (i-1)\Delta\hat{f}$$

для i = 2..M

4. Далее для каждой центральной частоты $\hat{f}_{c_i}$ в мелах считается центральная частота f_{c_i} в Гц и вычисляется ширина каждого фильтра

$$ERB_i = 6.23 * 10^{-6} * {f_{c_i}}^2 + 93.39 * 10^{-3} * f_{c_i} + 28.52$$
$$ERB_i = ERB_i * ERB$$

5. Далее считаются граничные частоты f_{low_i} и f_{high_i}

$$f_{low_i} = -(700 + ERB_i) + \sqrt{(700 + ERB_i)^2 + f_{c_i}(f_{c_i} + 1400)}$$

$$f_{high_i} = f_{low_i} + 2 * ERB_i$$

Вид банка фильтров HFCC на отрезке частот от 0 до 8000 Гц для значений ERB 1.0, 0.75 и 1.5 приведен на рис. 2.10.

Применение фильтра заключается в попарном перемножении его значений со значениями спектра. Результатом этой операции является мел-коэффициент. Поскольку фильтров у нас M, коэффициентов будет столько же. После применения фильтра для понижения чувствительности коэффициентов к шумам следует прологарифмировать результат.

$$X_i = \log_{10}(\sum_{i=0}^{N-1}|X(k)| * H_i(k)),$$

i = 1..M

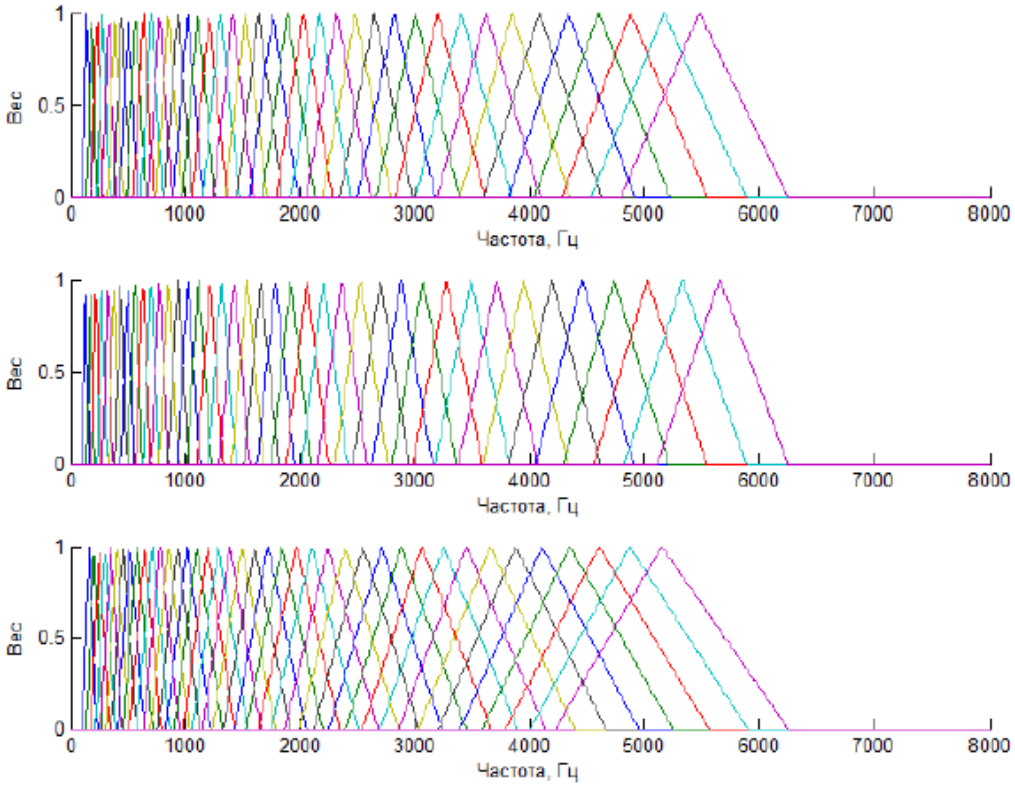

Рис. 2.10 Банки фильтров HFCC

Пример полученных значений показан на рис. 2.11.

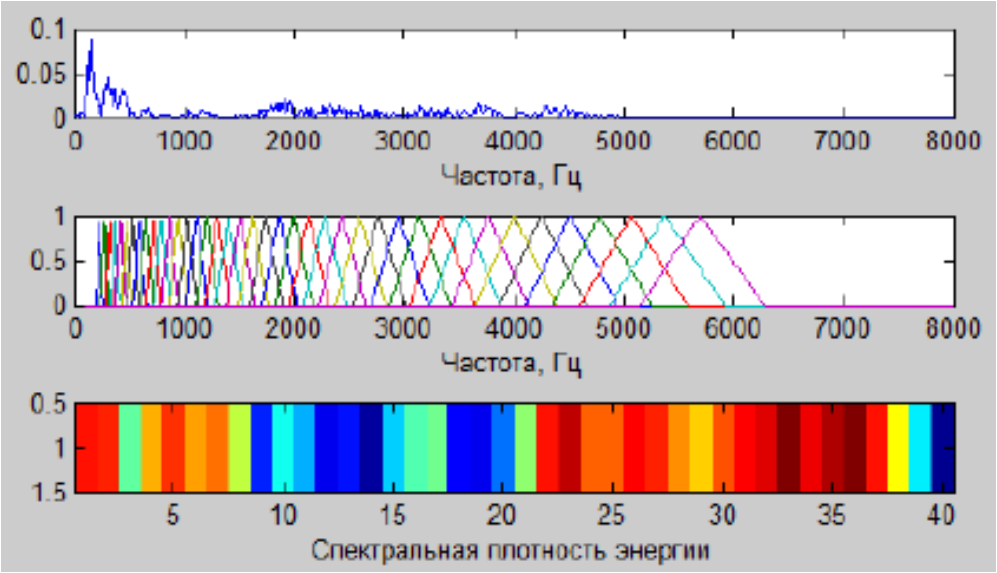

Рис. 2.11. Получение спектральной плотности энергии

2.5.4 Дискретное косинусное преобразование

Для того чтобы «сжать» полученные результаты, повысив значимость первых коэффициентов и уменьшив значимость последних, используется дискретное косинусное преобразование. После такого преобразования получаются мел-кепстральные частотные коэффициенты для каждого фрейма.

Формула дискретного косинусного преобразования:

$$C_j = \sum_{i=1}^{M} X_i * \cos(j * \left(i - \frac{1}{2}\right) * \frac{\pi}{M}),$$

j = 1..J, где M – количество фильтров, J – количество мел-кепстральных коэффициентов; обычно J < M.

Пример полученных мел-кепстральных коэффициентов на основе спектральной плотности энергии показан на рис. 2.12.

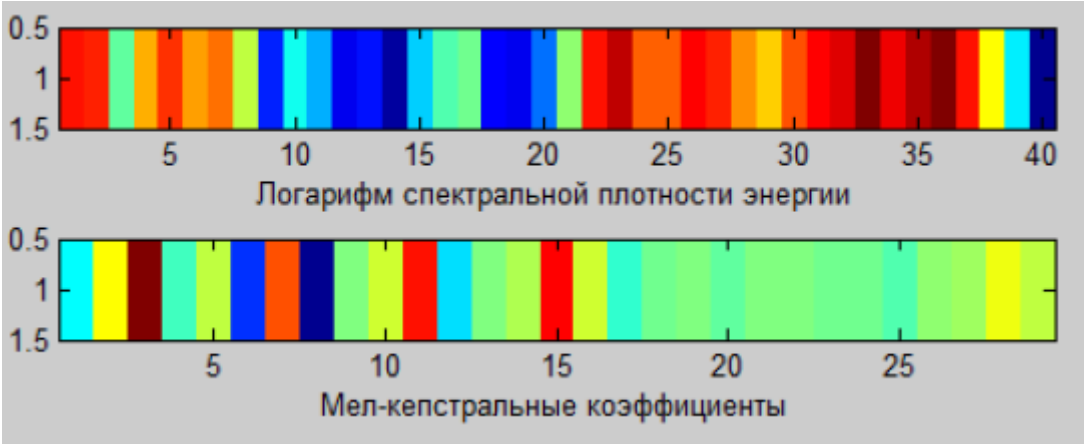

Рис 2.12 Получение мел-кепстральных коэффициентов

2.6 Разработка метода сравнения векторов признаков

При сравнении наборов мел-кепстральных коэффициентов отдельных фреймов необходимо получить численное значение меры различия этих векторов. Так как длина векторов (количество мел-кепстральных коэффициентов) фиксирована, то в качестве меры различия можно выбрать расстояние между векторами. Это может быть как евклидово расстояние, так и расстояние городских кварталов или расстояние Маланобиса. В данном проекте выбрано евклидово расстояние.

Так как образцы голоса, и, как следствие, вектора признаков, могут иметь различия как в общей продолжительности, так и в относительной длительности сегментов (слогов, слов), то требуется использовать способ сравнения, устойчивый к таким изменениям. Выравнивание путем линейного сжатия или растяжения не решает данную задачу, так как речевой сигнал протекает во времени неравномерно – при изменении длительности слова неравномерно меняется длительность звуков. Для сравнения векторов признаков с учетом этих особенностей используется метод динамического выравнивания времени.

Схема сравнения векторов признаков приведена на рис. 2.13.

Рис. 2.13 Схема сравнения векторов признаков.

Для сравнения векторов признаков, представляющих собой матрицы мел-кепстральных коэффициентов A размером K×N (K – количество мел-кепстральных коэффициентов) и B размером K×M, сначала строится матрица расстояний D размером M×N, элементами d_{ij} которой являются евклидовы расстояния столбцов (фреймов) A_i и B_j:

$$D_{ij} = \frac{1}{K}\sqrt{\sum_{k=1}^{K}(a_{ki} - b_{kj})^2}$$

Отображение матрицы D для схожих сигналов с помощью шкалы цветовых температур приведено на рис. 2.14.

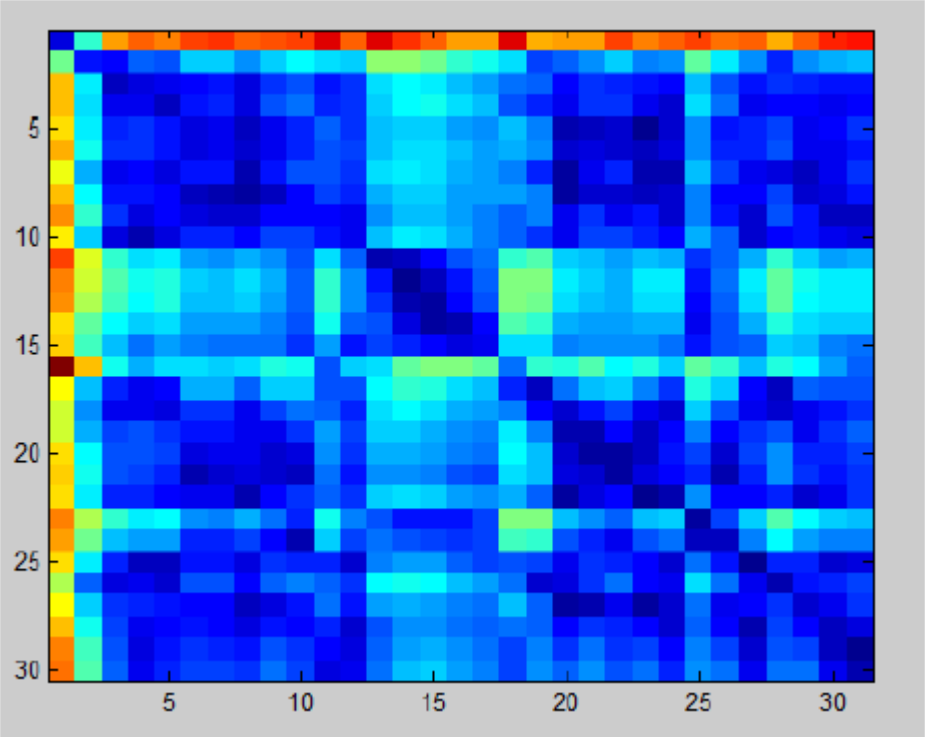

Рис. 2.14 Матрица расстояний

На матрицу расстояний накладывается маска F размером M×N (Рис. 2.15):

$$F_{i,j} = \begin{cases} 1, & d_{i,j} \le 0.125 \\ 1 + 2d_{i,j}, & 0.125 < d_{i,j} \le 0.5 \\ \infty, & \mathrm{d}_{i,j} > 0.5 \end{cases}$$

где

$$d_{i,j} = \left| \frac{i}{M} - \frac{j}{N} \right|$$

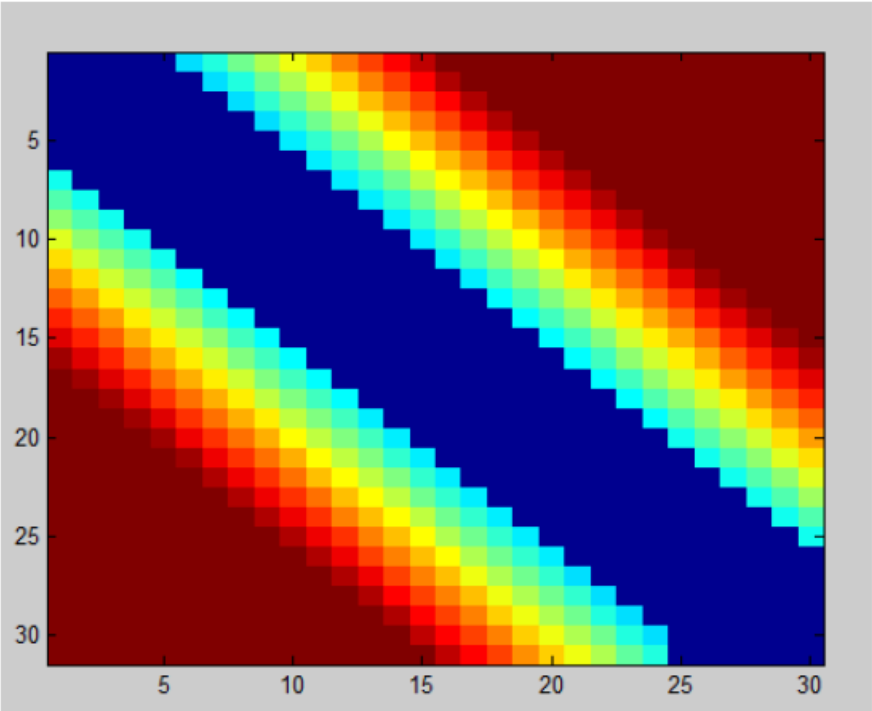

Рис. 2.15 Маска

С помощью этой маски элементы в матрице расстояний корректируются с учетом их расстояния от главной диагонали матрицы, что уменьшает вероятность срабатывания системы при изменении порядка слов в ключевой фразе (или при произнесении неправильной ключевой фразы) и препятствует обходу верификации с помощью технических средств.

Матрица схожести S формируется поэлементным умножением матрицы расстояний D на маску F.

$$S_{ij} = D_{ij} * M_{ij}$$

Отображение получающейся матрицы схожести с помощью шкалы цветовых температур показано на рис. 2.16.

Для получения меры схожести векторов признаков из каждой строки матрицы выбирается минимальный элемент, и для этих элементов считается среднеквадратичное значение.

$$dist = \frac{1}{N}\sum_{j=1}^{N}\sqrt{(\min_i D_{i,j})^2}$$

Критерием подтверждения личности будет выполнение условия

$$dist < d_{\text{п}}$$

где $d_{\text{п}}$ – установленный порог максимального различия векторов признаков.

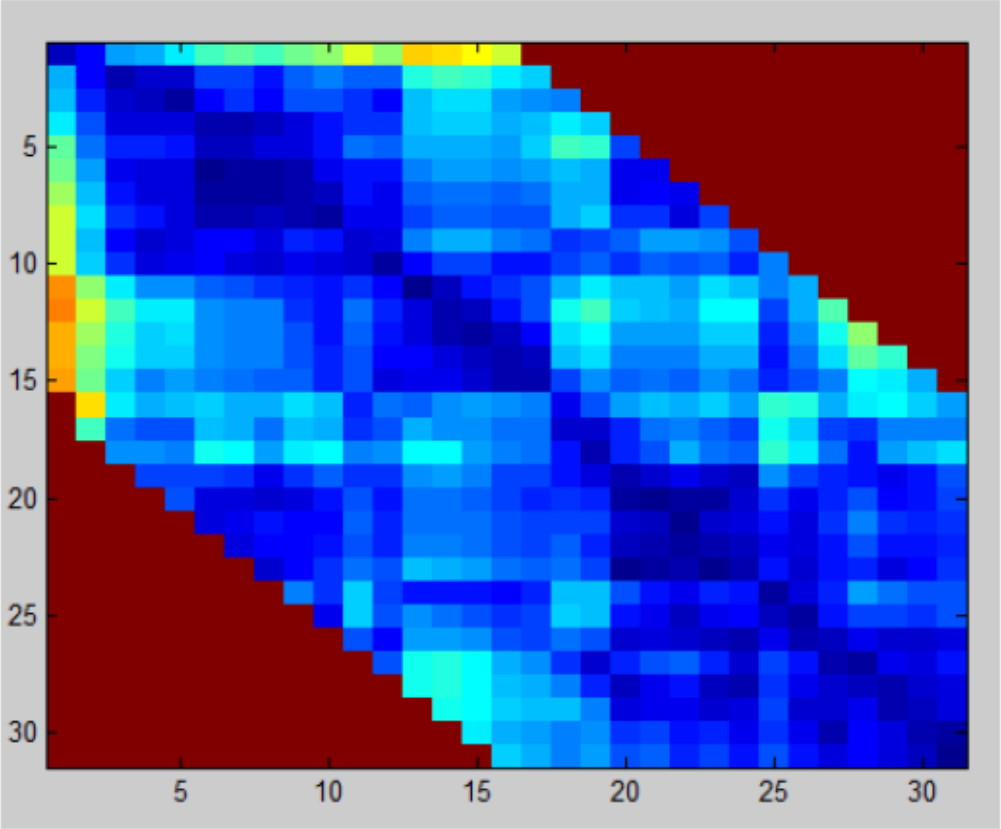

Рис. 2.16 Матрица схожести

Критерием присвоения статуса «Сомнительно» будет невыполнение предыдущего условия при выполнении следующего

$$dist < d_{\text{п}} * \frac{K_{\text{р}}}{100}$$

где $K_{\text{р}}$ – коэффициент различия векторов признаков для статуса «Сомнительно», в %.

2.7 Выводы по разделу

В данном разделе был выбраны алгоритмы получения и сравнения векторов признаков речевых сигналов и оптимальные значения параметров данных алгоритмов. В качестве векторов признаков будут использоваться матрицы мел-кепстральных коэффициентов, в качестве алгоритма сравнения был выбран метод определения евклидовых расстояний между векторами мел-кепстральных коэффициентов с использованием динамического выравнивания времени.

3. Технологическая часть

Цель данного раздела – описание реализации выбранных алгоритмов, тестирование системы и определение ее погрешности.

3.1 Разработка методики тестирования

Для подтверждения корректности выбора алгоритмов и их реализации, а также для выбора оптимальных значений граничного расстояния подтверждения личности и коэффициента статуса «Сомнительно» необходимо провести тестирование и оценку полученных результатов по трем критериям:

1. Проверка верификации образцов пользователя, для которого обучалась система (отсутствие ложно-отрицательных результатов),

2. Проверка непрохождения верификации для образцов других людей (отсутствие ложно-положительных срабатываний),

3. Проверка непрохождения верификации при произнесении пользователем неправильной ключевой фразы,

4. Проверка непрохождения верификации для образцов, не содержащих речевой сигнал.

3.1.1 Выбор программного обеспечения для реализации алгоритмов

Для моделирования и тестирования выбранных алгоритмов в данном дипломном проекте использовалась среда Microsoft Visual Studio 2013 и язык программирования C++

3.1.2 Реализация алгоритмов

Для получения граничного расстояния подтверждения для определенного количества образцов была реализована функция get_threshold(), которая принимает в качестве входного аргумента массив имен файлов для подготовки системы и возвращает граничное расстояние подтверждения. В этой функции программа с помощью встроенной C-функции fopen_s() получает частоту дискретизации образца и вектор амплитуд. Далее вызывается функци получения вектора признаков get_feature_vector(), принимающей на вход оцифрованный цифровой сигнал в виде

вектора амплитуд и частоту дискретизации входного сигнала, и возвращающей вектор признаков – массив мел-кепстральных коэффициентов.

На этом этапе вектор амплитуд представляет массив чисел с плавающей точкой лежащих в интервале от -1 до 1. Далее производится сглаживание сигнала в функции filter() принимающей вектор амплитуд, а также коэффициент К. В результате тестирования было выяснено, что К равное 25 наиболее оптимально (см. 2.4.2).

После сглаживания сигнала идет разбиение сигнала на фреймы, а также применение оконной функции (см. 2.4.1 и 2.5.1) в функции vec2frames(), принимающей на вход цифровой сигнал в виде вектора амплитуд и два целых числа определяющих количество фреймов и перекрытие фреймов, и возвращающая матрицу амплитуд. В результате работы этой функции исходный сигнал представляется в виде матрицы, например 4410x78.

После разбиения на фреймы вызывается функция удаления тишины silence_removal(), принимающая на вход сигнал, разбитый на фреймы, в виде матрицы, и возвращающая тот же тип данных, но содержащий только фреймы с полезным сигналом (см. 2.4.2), например матрица 4410x73. В этом примере в результате выполения алгоритма удаления тишины были удалены 5 столбцов.

Далее, в цикле для каждого столбца матрицы, вызывается для вычисления спектра сигнала вызывается функция алгоритма быстрого преобразования Фурье FFT(), которая принимает массив реальных и мнимых частей сигнала (в нашем случае мнимая часть заполняется нулями) и длину этих массивов (длина должна быть в виде 2^n), и возвращает реальную и мнимую часть спектра (см. 2.5.2). В нашем примере 4410 отсчетов расширяются нулями до 2^{13}= 8192 отсчета. В результате получается матрица спектров 8192x73. Но так как у нас отсчетов изначально было 4410 мы удаляем часть и получаем матрицу 4097x73.

Также необходимо получить банк мел-фильтров в функции trifbank_hfcc(), принимающей на вход минимальную и максимальную частоту, количество

фильтров, частоту дискретизации исходного сигнала, коэффициент ширины фильтров и возвращающей матрицу мел-коэффициентов, которые следует наложить на спектр исходного сигнала. В итоге получилась матрица 40x4097 представляющая «гребенку фильтров» (см. 2.5.3). Далее в той же функции get_feature_vector() производится наложение спектра на «гребенку фильтров», производится логарифмирование и дискретное косинусное преобразование (см. 2.5.4).

Таким образом получилась матрица 29x73 содержащая мел-кепстральные частотные коэффициенты (вектора признаков).

Сравнение векторов признаков реализовано в функции compare_feature_vectors(), принимающей на вход 2 вектора признаков и возвращающая численное значение меры различия этих векторов. Вызывается эта функция из функции get_threshold(). Используя алгоритм сравнения описанный в разделе 2.6, функция возвращает расстояния между всеми образцами используемыми для подготовки системы к распознаванию.

Функция verify_samples() предназначена для верификации образцов и выдачи результата по каждому из образцов. Принимает на вход граничное расстояние статуса «подтверждено» и «сомнительно». Использует функцию get_feature_vector() и compare_feature_vectors(), только не запоминает расстояния между векторами, а сравнивает с граничными расстояниями и возвращает статус проверяемого образца «подтверждено», «сомнительно» или «отвергнуто».

3.2 Тестирование

3.2.1 Автоматизированное тестирование

Автоматизированное тестирование реализации алгоритма выполнялось с помощью программы на языке C++. Для подготовки и тестирования системы было записано по несколько образцов ключевой фразы от 8 разных людей, с разной скоростью произношения. Программа проводит подготовку системы на основе трех образцов пользователя, затем производит проверку для каждого файла, находящегося в директории с тестируемыми файлами (./verify). Для каждого

протестированного файла выводится численное значение меры различия векторов признаков и принятое решение.

Проверяемые файлы делятся на 2 группы:

1. Образцы пользователя, для которого проводилось обучение системы,

2. Образцы других людей,

В первой группе находятся образцы пользователя, для которого проводилось обучение системы. Для тестирования этой группы было записано 9 образцов произношения ключевой фразы.

Во второй группе находятся образцы произношения ключевой фразы от других людей. Для тестирования этой группы было записано 54 образца от 8 разных людей.

По результатам тестирования на основе полученных расстояний между векторами признаков в каждой группе выводятся рекомендуемые значения границы подтверждения и статуса «Сомнительно».

$$d_{\text{п рек.}} = \frac{\max\limits_i dist_i^{1\text{ гр.}} + \min\limits_j dist_j^{2\text{гр.}}}{2}$$

$$\text{К}_{\text{р рек.}} = 1 + \frac{\max\limits_i dist_i^{1\text{ гр.}} - \min\limits_j dist_j^{2\text{гр.}}}{2 * d_{\text{п рек.}}}$$

Схема алгоритма программы тестирования приведена на рис. 3.1.

Журнал тестирования описанным выше методом с помощью программы и исходный код тестирования приведены в приложении.

По итогам тестирования с изначально заданными значениями расстояния подтверждения как среднего расстояния между векторами признаков обучающей группы, увеличенного на 33%

$$d_{\text{п}} = \frac{1}{N}\sum_{i=1}^{N} dist_i^{\text{обуч.}} * 1.33 = 4.3$$

И коэффициента статуса «сомнительно», равного 10%

$$K_{\text{р}} = 1.1$$

были получены результаты, приведенные в таблице 3.1.

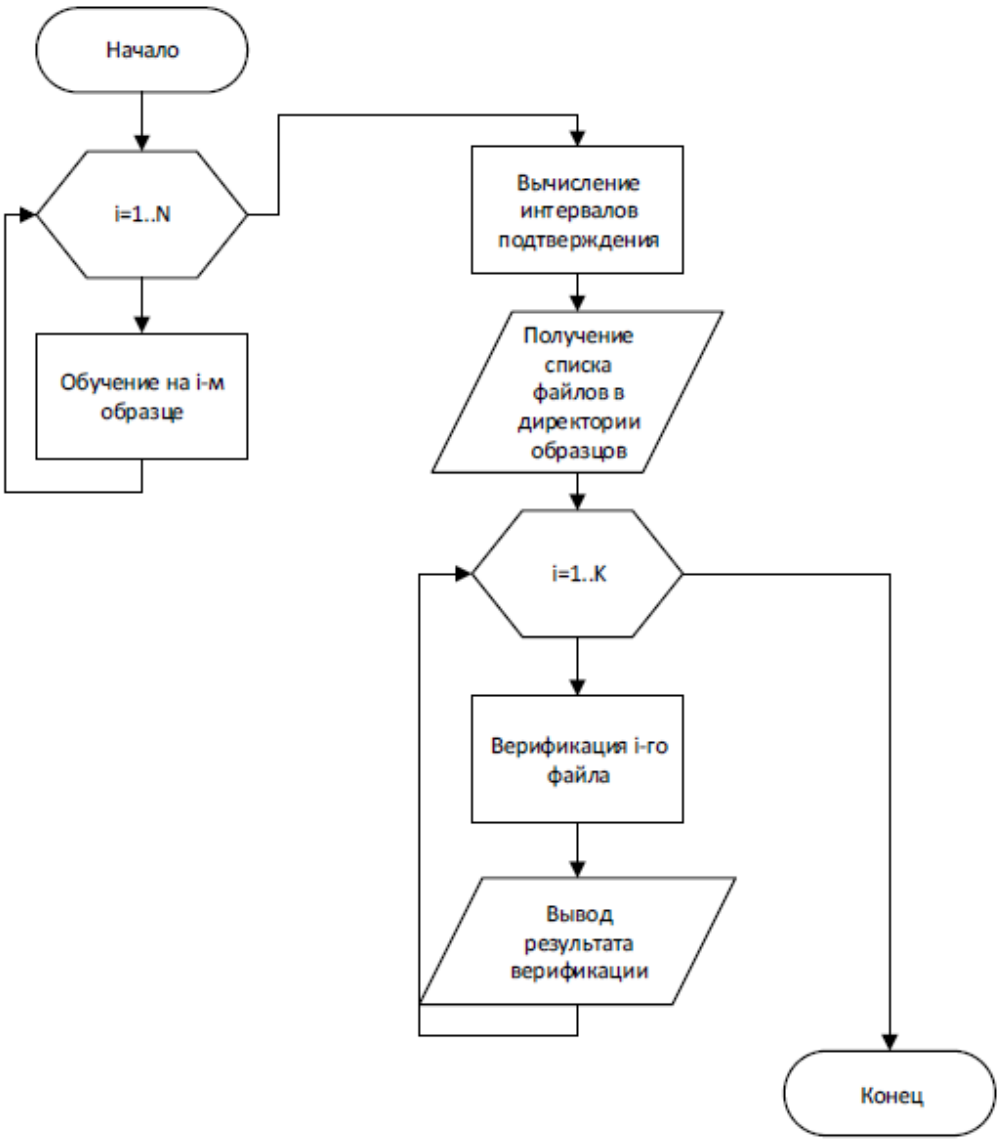

Рис. 3.1 Схема алгоритма тестирования

Таблица 3.1 Результаты тестирования

Группа	Образцов	Подтверждено	Сомнительно	Отвергнуто	Верно
1	8	7	1	0	7
2	54	0	1	53	53
Всего	62	7	2	53	60

Итоговая точность составила 96.7%.

Были получены рекомендуемые значения

$$d_{\text{п рек.}} = 4.38$$

$$K_{\text{р рек.}} = 1.04$$

При повторном тестировании с использованием рекомендованных значений была достигнута точность 100%.

3.2.2 Анализ работы алгоритма на этапе тестирования

1. Сравнение обучающих образцов. Для анализа обучающих образцов произнесения ключевой фразы от одного человека (пользователя) были выведены матрицы мел-кепстральных коэффициентов, матрицы расстояний и схожести (Рис. 3.2 – 3.5).

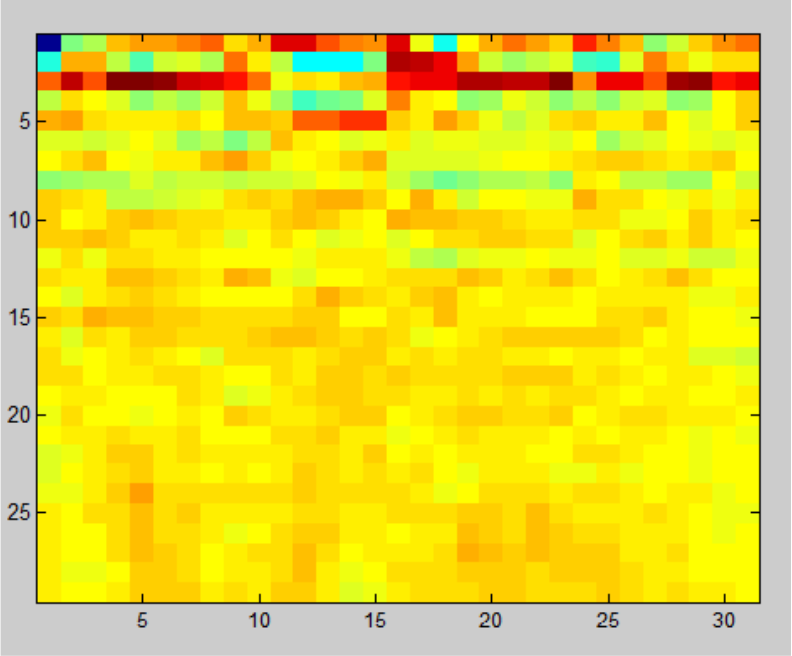

Рис. 3.2 Матрица мел-кепстральных коэффициентов образца 1

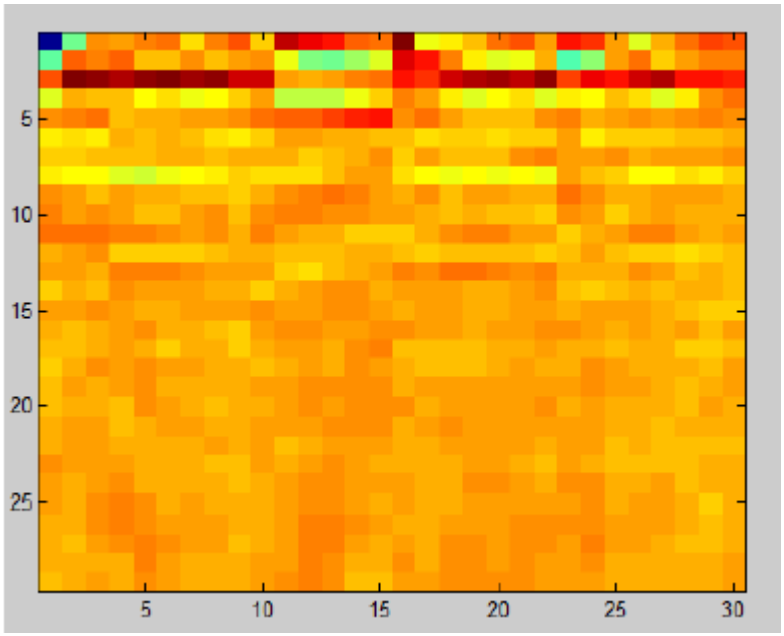

Рис. 3.3 Матрица мел-кепстральных коэффициентов образца 2

Значения коэффициентов в матрице, изображенной на рис. 3.2, находятся в диапазоне [-13.01, 7.82], изображенной на рис. 3.3 – в диапазоне [-15.6, 7.5]. По этим изображениям видно, что матрицы имеют схожую структуру.

В матрице расстояний, изображенной на рис. 3.4, хорошо просматривается область минимальных значений, расположенная вдоль главной диагонали матрицы.

Наложение маски на эту матрицу дополнительно выделяет эти элементы и отсекает области, расположенные в левом нижнем и правом верхнем углах.

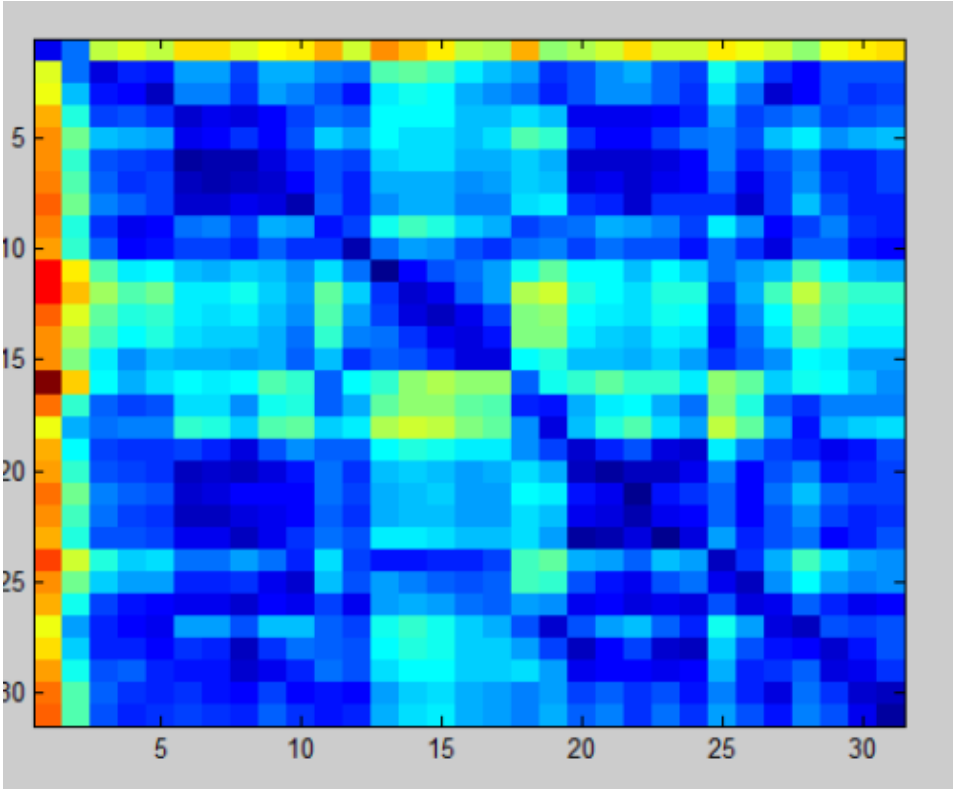

Рис. 3.4 Матрица расстояний обучающих образцов

Итоговое расстояние между обучающими образцами равно 3.1

2. Образец из второй группы, имеющий наибольшее сходство с обучающими образцами.

Матрицы мел-кепстральных коэффициентов и расстояний для этого образца изображены на рис. 3.6, 3.7. Значения матрицы мел-кепстральных коэффициентов находятся в диапазоне [-4.33, 6.2]. На этой матрице заметны области низких и высоких значений, схожие с аналогичными на рис. 3.2, 3.3.

На матрице расстояний (Рис. 3.7) заметны области низких значений как вдоль главной диагонали, так и на расстоянии от нее. Итоговое расстояние для этого образца составляет 4.56.

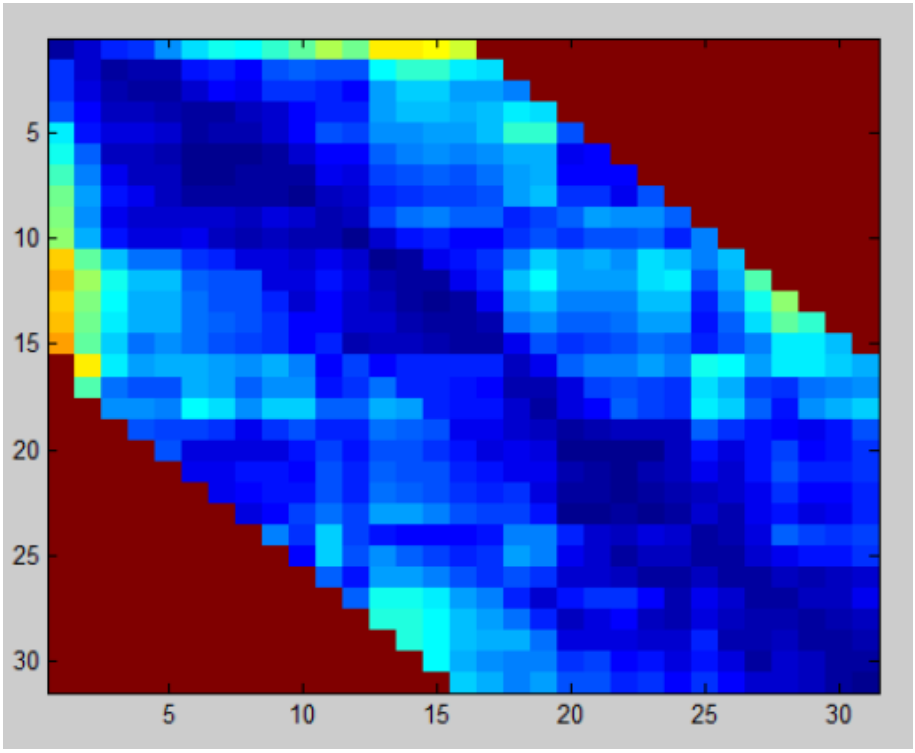

Рис. 3.5 Матрица схожести обучающих образцов

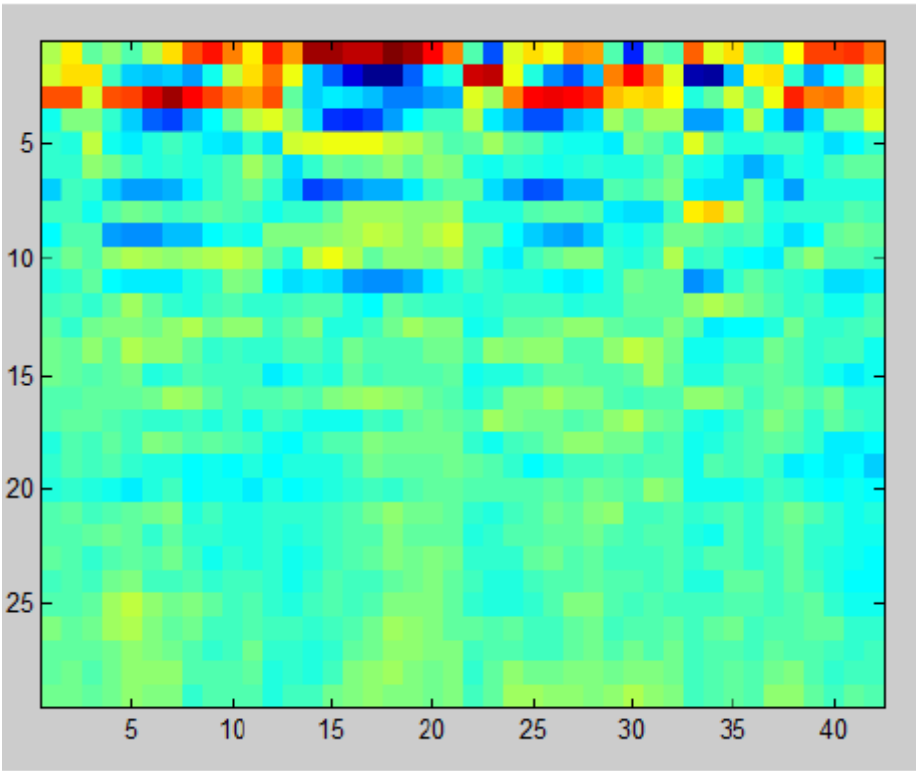

Рис. 3.6 Матрица мел-кепстральных коэффициентов образца из группы 2

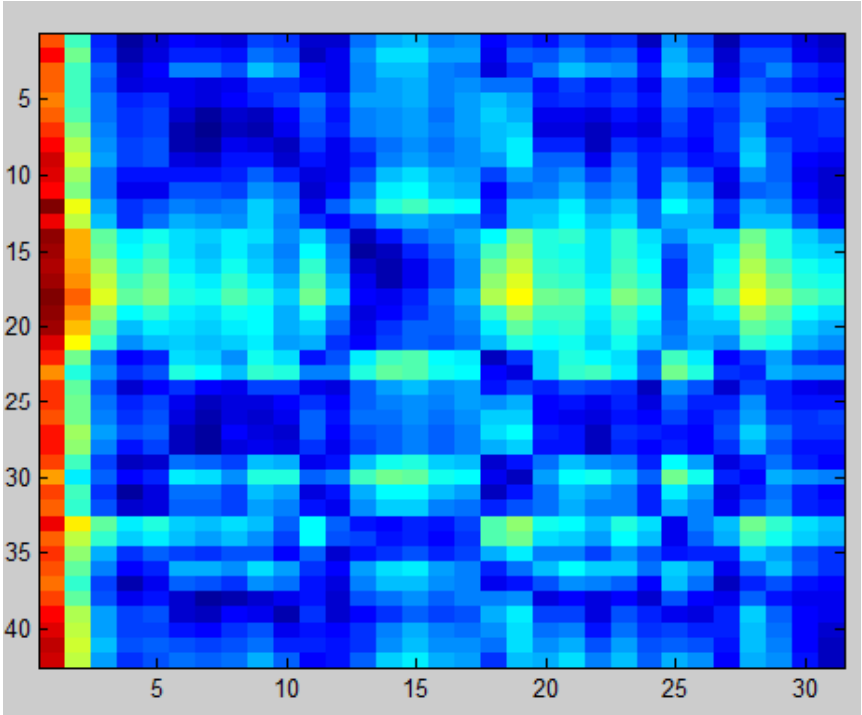

Рис. 3.7 Матрица расстояний для образца из группы 2

3.3 Определение погрешности системы

В результате тестирования системы на 62 образцах голоса при заданных значениях параметров алгоритма были подтверждены 7 из 8 образцов голоса, соответствовавшие пользователю, для которого была обучена система, были отвергнуты 53 из 54 образцов голоса других людей, и 1 из 54 образцов попал в область неопределенного результата. На основе полученных данных можно сделать вывод, что погрешность системы составила 2/62 = 3.3%.

3.4 Выводы по разделу

В данном разделе были реализованы описанные алгоритмы в виде C++ программы. Было выполнено автоматизированное тестирование и на основе его результатов определена погрешность системы, которая составила 3.3%.

4. Заключение

Из проделанной работы следует, что разработанное программное обеспечение полностью удовлетворяет поставленной задаче. В данной работе были рассмотрены теоретические и практические аспекты создания системы верификации личности пользователя по голосу. Выбранные алгоритмы были протестированы на образцах голосов различных людей и показали высокую точность результата.

В результате получено программное обеспечение для верификации пользователя по голосу с погрешностью не более 3,3%. Так как программа написана на языке C++ она легко модифицируема и имеет возможности как для использования во встраиваемых системах, мобильных операционных системах, так и для использования на домашний компютерах.

Дальнейшее развитие алгоритмической составляющей может включать в себя добавление алгоритма подавления шума для повышения устойчивости системы к воздействию шумов. Практической части – распараллеливание вычислений, для увеличения скорости работы алгоритма.

5. Список источников и литературы

1. Л.Р. Рабинер, Р.В. Шафер «Цифровая обработка речевых сигналов»
2. О.С. Агашин, О.Н. Корелин «Методы цифровой обработки речевого сигнала в задаче распознавания изолированных слов с применением сигнальных процессоров»
3. Г. Н. Зубов «Судебная экспертиза звукозаписей. Состояние, проблемы, перспективы».
4. Todor Ganchev, Nikos Fakotakis, George Kokkinakis «Comparative Evaluation of Various MFCC Implementations on the Speaker Verification Task»
5. Xuedong Huang, Alex Acero, Hsiao-Wuen Hon «Spoken Language Processing: A Guide to Theory, Algorithm and System Development»
6. Fathi E. Abd El-Samie. Information Security for Automatic Speake Identification. Springer 2011. – 137 p.
7. T. Herbig, F.Gerl, W. Minker. Self-Learning Speaker Identification.Springer 2011 – 185 p.
8. Задорожный М.С. Система верификации личности пользователя по голосу. – Пояснительная записка к дипломному проекту. 2014. – 87 с.

Приложение А. Исходный код программы на языке C++

Файл «main.h»

```
#ifndef C_DSP_WAV_H
#define C_DSP_WAV_H

#include <stdlib.h>
#include <Windows.h>
#include <string>
#include <iostream>
#include <tchar.h>
#include <conio.h>
#include <math.h>
#include <vector>
#include <float.h>
#include <locale.h>
#include "fft.h"

#endif
```

Файл «main.cpp»

```
#include "main.h";

using namespace std;
#define PI 3.1415926535
#define TRAIN_COUNT 3
vector <vector <float>> MFCCs_train[TRAIN_COUNT];
const double TwoPi = 6.283185307179586;

#define  NUMBER_IS_2_POW_K(x)   ((!((x)&((x)-1)))&&((x)>1))  // x is pow(2, k), k=1,2, ...
#define  FT_DIRECT        -1    // Direct transform.
#define  FT_INVERSE         1    // Inverse transform.
```

```
// Структура, описывающая заголовок WAV файла.
struct WAVHEADER
{
	// WAV-формат начинается с RIFF-заголовка:

	// Содержит символы "RIFF" в ASCII кодировке
	// (0x52494646 в big-endian представлении)
	char chunkId[4];

	// 36 + subchunk2Size, или более точно:
	// 4 + (8 + subchunk1Size) + (8 + subchunk2Size)
	// Это оставшийся размер цепочки, начиная с этой позиции.
	// Иначе говоря, это размер файла - 8, то есть,
	// исключены поля chunkId и chunkSize.
	unsigned long chunkSize;

	// Содержит символы "WAVE"
	// (0x57415645 в big-endian представлении)
	char format[4];

	// Формат "WAVE" состоит из двух подцепочек: "fmt " и "data":
	// Подцепочка "fmt " описывает формат звуковых данных:

	// Содержит символы "fmt "
	// (0x666d7420 в big-endian представлении)
	char subchunk1Id[4];

	// 16 для формата PCM.
	// Это оставшийся размер подцепочки, начиная с этой позиции.
	unsigned long subchunk1Size;
```

```
// Аудио формат, полный список можно получить здесь http://audiocoding.ru/wav_formats.txt
// Для PCM = 1 (то есть, Линейное квантование).
// Значения, отличающиеся от 1, обозначают некоторый формат сжатия.
unsigned short audioFormat;

// Количество каналов. Моно = 1, Стерео = 2 и т.д.
unsigned short numChannels;

// Частота дискретизации. 8000 Гц, 44100 Гц и т.д.
unsigned long sampleRate;

// sampleRate * numChannels * bitsPerSample/8
unsigned long byteRate;

// numChannels * bitsPerSample/8
// Количество байт для одного сэмпла, включая все каналы.
unsigned short blockAlign;

// Так называемая "глубиная" или точность звучания. 8 бит, 16 бит и т.д.
unsigned short bitsPerSample;

// Подцепочка "data" содержит аудио-данные и их размер.

// Содержит символы "data"
// (0x64617461 в big-endian представлении)
char subchunk2Id[4];

// numSamples * numChannels * bitsPerSample/8
// Количество байт в области данных.
unsigned long subchunk2Size;
```

```
        // Далее следуют непосредственно Wav данные.
};

bool  FFT(float *Rdat, float *Idat, int N, int LogN, int Ft_Flag)
{
        // parameters error check:
        if ((Rdat == NULL) || (Idat == NULL))                return false;
        if ((N > 16384) || (N < 1))                          return false;
        if (!NUMBER_IS_2_POW_K(N))                           return false;
        if ((LogN < 2) || (LogN > 14))                       return false;
        if ((Ft_Flag != FT_DIRECT) && (Ft_Flag != FT_INVERSE)) return false;

        register int  i, j, n, k, io, ie, in, nn;
        float         ru, iu, rtp, itp, rtq, itq, rw, iw, sr;

        static const float Rcoef[14] =
        { -1.0000000000000000F, 0.0000000000000000F, 0.7071067811865475F,
        0.9238795325112867F, 0.9807852804032304F, 0.9951847266721969F,
        0.9987954562051724F, 0.9996988186962042F, 0.9999247018391445F,
        0.9999811752826011F, 0.9999952938095761F, 0.9999988234517018F,
        0.9999997058628822F, 0.9999999264657178F
        };
        static const float Icoef[14] =
        { 0.0000000000000000F, -1.0000000000000000F, -0.7071067811865474F,
        -0.3826834323650897F, -0.1950903220161282F, -0.0980171403295606F,
        -0.0490676743274180F, -0.0245412285229122F, -0.0122715382857199F,
        -0.0061358846491544F, -0.0030679567629659F, -0.0015339801862847F,
        -0.0007669903187427F, -0.0003834951875714F
        };
```

```
nn = N >> 1;
ie = N;
for (n = 1; n <= LogN; n++)
{
        rw = Rcoef[LogN - n];
        iw = Icoef[LogN - n];
        if (Ft_Flag == FT_INVERSE) iw = -iw;
        in = ie >> 1;
        ru = 1.0F;
        iu = 0.0F;
        for (j = 0; j<in; j++)
        {
                for (i = j; i<N; i += ie)
                {
                        io = i + in;
                        rtp = Rdat[i] + Rdat[io];
                        itp = Idat[i] + Idat[io];
                        rtq = Rdat[i] - Rdat[io];
                        itq = Idat[i] - Idat[io];
                        Rdat[io] = rtq * ru - itq * iu;
                        Idat[io] = itq * ru + rtq * iu;
                        Rdat[i] = rtp;
                        Idat[i] = itp;
                }

                sr = ru;
                ru = ru * rw - iu * iw;
                iu = iu * rw + sr * iw;
        }

        ie >>= 1;
```

```
}

for (j = i = 1; i<N; i++)
{
        if (i < j)
        {
                io = i - 1;
                in = j - 1;
                rtp = Rdat[in];
                itp = Idat[in];
                Rdat[in] = Rdat[io];
                Idat[in] = Idat[io];
                Rdat[io] = rtp;
                Idat[io] = itp;
        }

        k = nn;

        while (k < j)
        {
                j = j - k;
                k >>= 1;
        }

        j = j + k;
}

if (Ft_Flag == FT_DIRECT) return true;

rw = 1.0F / N;
```

```
	for (i = 0; i<N; i++)
	{
		Rdat[i] *= rw;
		Idat[i] *= rw;
	}

	return true;
}

void EraseColMatrinx(vector <vector <float>> &source, int N)
{
	for (int i = 0; i < source.size(); ++i)
	{
		source[i].erase(source[i].begin() + N);
	}
}

void filter(const float k, vector<float> &speech)
{
	vector<float> resSpeech(speech.size());
	int size = speech.size();
	for (int i = k; i < size - k; ++i)
	{
		float sum = 0;
		for (int j = -k; j < k + 1; ++j)
		{
			sum = speech[i + j];
		}
		resSpeech[i] = (1 / k)*sum;
	}
	for (int i = 0; i < k; ++i)
```

```
            resSpeech[speech.size() - i-1] = speech[i];
        for (int i = 0; i < k; ++i)
            resSpeech[i] = speech[i];
        speech = resSpeech;
}

void MulMatrix(const vector <vector <float> > &A, const vector <vector <float> > &B, vector <vector <float> > &Res)
{
        if (A.front().size() != B.size())
            throw 1;
        int a = A.size();
        int b = B.size();
        int c = B.front().size();

        float **Am = new float*[a]; // строки в массиве
        for (int count = 0; count < a; count++)
            Am[count] = new float[b]; // столбцы
        float **Bm = new float*[b]; // строки в массиве
        for (int count = 0; count < b; count++)
            Bm[count] = new float[c]; // столбцы
        float **R = new float*[a]; // строки в массиве
        for (int count = 0; count < a; count++)
            R[count] = new float[c]; // столбцы
        vector <vector <float> > Rm(a, vector<float>(c, 0));

        for (int i = 0; i < a; ++i)
            for (int j = 0; j < b; ++j)
            {
                Am[i][j] = A[i][j];
            }
        for (int i = 0; i < b; ++i)
```

```
            for (int j = 0; j < c; ++j)
            {
                Bm[i][j] = B[i][j];
            }

        for (int i = 0; i < a; ++i)
            for (int j = 0; j < c; ++j)
            {
                R[i][j] = 0;
                for (int k = 0; k < b; ++k)
                    R[i][j] += Am[i][k] * Bm[k][j];
            }
        for (int i = 0; i < a; ++i)
            for (int j = 0; j < c; ++j)
            {
                Rm[i][j] = R[i][j];
            }
        Res = Rm;

        for (int count = 0; count < a; count++)
            delete[] Am[count];
        for (int count = 0; count < b; count++)
            delete[] Bm[count];
        for (int count = 0; count < a; count++)
            delete[] R[count];
}

void diag(const vector <float> &V, vector <vector <float> > &R)
{
        vector <vector <float> > A(V.size(), vector<float>(V.size(), 0));
        for (int i = 0; i < V.size(); ++i)
```

```
            A[i][i] = V[i];
        R = A;
}

void vec2frames(vector<float> &vec, int Nw, int Ns, vector <vector <float> > &resFrames)
{
        int L = vec.size();

        int M = floor((L - Nw) / Ns + 1);
        int E = (L - ((M - 1) * Ns + Nw));
        if (E > 0)
        {
                int P = Nw - E;
                vec.insert(vec.end(), P, 0);
                M = M + 1;
        }

        vector<int> indf(M);
        vector<int> inds(Nw);
        for (int i = 0; i < M; ++i)
                indf[i] = i * Ns;
        for (int i = 0; i < Nw; ++i)
                inds[i] = i + 1;

        vector <vector <float> > frames(inds.size(), vector<float>(indf.size()));
        for (int i = 0; i < inds.size(); ++i)
                for (int j = 0; j < indf.size(); ++j)
                {
                        frames[i][j] = vec[inds[i] + indf[j] - 1];
                }
```

```
	vector<float> window_s(Nw);
	for (int i = 0; i < window_s.size(); ++i)
		window_s[i] = 0.54 - 0.46 * cos((2.0 * PI * i) / (window_s.size() - 1.0));
	vector<vector<float>> ham_matrix(Nw, vector<float>(Nw, 0));
	diag(window_s, ham_matrix);
	MulMatrix(ham_matrix, frames, resFrames);
}

void silence_removal(vector <vector <float>> &source)
{
	const int Nf = source.front().size();
	vector <float> E(Nf, 0);
	const int Nw = source.size();
	float E_mean = 0;
	for (int i = 0; i < Nf; ++i)
	{
		float sum = 0;
		for (int j = 0; j < Nw; ++j)
			sum += fabs(source[j][i])*fabs(source[j][i]);
		E[i] = sum * (1.0 / Nw);
		E_mean += E[i];
	}
	E_mean /= Nf;
	float T_E = E_mean / 10;
	vector<bool> voicedIndexes(Nf);
	for (int i = 0; i < E.size(); ++i)
		voicedIndexes[i] = ((E[i] > T_E) ? 1 : 0);
	for (int i = Nf - 1; i >= 0; --i)
		if (!voicedIndexes[i])
			EraseColMatrinx(source, i);
```

```
}

void trifbank_hfcc(unsigned Nhfcc, unsigned K, int fmin, int fmax, unsigned fs, float ERBscaleFactor, vector <vector <float>> &resFbHfcc)
{
	unsigned Nfft = K * 2;

	// Коэффициенты Глассберга - Мура
	float a = 6.23e-6;
	float b = 93.39e-3;
	float c = 28.52;
	a *= ERBscaleFactor;
	b *= ERBscaleFactor;
	c *= ERBscaleFactor;

	// Центральная частота первого фильтра
	float ahat = 0.5 * (1.0 / (700.0 + fmin));
	float bhat = 0.5 * (2.0 * 700.0 / (700.0 + fmin));
	float chat = 0.5 * (700.0 * (-1 + 700.0 / (700.0 + fmin)) - fmin);
	float bbar = (b - bhat) / (a - ahat);
	float cbar = (c - chat) / (a - ahat);
	float fcLow = 0.5 * (-bbar + sqrt(pow(bbar, 2) - 4.0 * cbar));

	// Центральная частота второго фильтра
	ahat = -0.5 * (1.0 / (700.0 + fmax));
	bhat = -0.5 * (2.0 * 700.0 / (700.0 + fmax));
	chat = 0.5 * (fmax - 700.0 * (-1 + 700.0 / (700.0 + fmax)));
	bbar = (b - bhat) / (a - ahat);
	cbar = (c - chat) / (a - ahat);
	float fcHigh = 0.5 * (-bbar + sqrt(pow(bbar, 2) - 4.0 * cbar));

	// Мел шкала
```

```
float fcLowmel = 2595.0 * log10(1 + fcLow / 700.0);
float fcHighmel = 2595.0 * log10(1 + fcHigh / 700.0);
float melSpace = (fcHighmel - fcLowmel) / (Nhfcc - 1);
vector <float> fcHfccmel(Nhfcc + 2);
fcHfccmel[0] = 0;
for (int i = 0; i < Nhfcc; ++i)
        fcHfccmel[i + 1] = fcLowmel + melSpace * i;
fcHfccmel[Nhfcc+1] = 2595.0 * log10(1.0 + (fs / 2.0) / 700.0);
vector <float> fcHfcc(fcHfccmel.size());
for (int i = 0; i < fcHfcc.size(); ++i)
        fcHfcc[i] = 700.0 * (-1.0 + pow(10, fcHfccmel[i] / 2595.0));

// Инициализация банка фильтров
vector<vector<float>> fbHfcc(fcHfcc.size()-2, vector<float>(Nfft / 2, 0));
vector<float> fftFreqs(Nfft / 2);
for (int i = 0; i < Nfft / 2; ++i)
        fftFreqs[i] = ((i / (2.0 - 1.0)) / Nfft) * fs;

// Создание банка фильтров
vector <float> centerF(fcHfcc.size()-2);
vector <float> centerFmel(fcHfcc.size() - 2);
for (int i = 2; i < fcHfcc.size(); ++i)
{
        centerF[i-2] = fcHfcc[i - 1];
        centerFmel[i-2] = 2595.0 * log10(1.0 + centerF[i-2] / 700.0);
}

// Определение ширины каждого фильтра и масштабирование ширины
vector <float> ERB(centerF.size());
for (int i = 0; i < ERB.size(); ++i)
{
```

```
        ERB[i] = (6.23 * pow((centerF[i] / 1000.0), 2) + 93.39 * (centerF[i] / 1000.0) + 28.52)*ERBscaleFactor;
    }

    // Границы фильтров
    vector<float> lowerFmel(centerF.size());
    vector<float> upperFmel(centerF.size());
    for (int i = 0; i < lowerFmel.size(); ++i)
    {
        lowerFmel[i] = 2595.0 * log10((-2.0 / 1400.0) * ERB[i] + 0.5 * sqrt(pow(((2.0 / 700.0) * ERB[i]), 2) + 4.0 * pow(10, (2.0 * centerFmel[i] / 2595.0))));
        upperFmel[i] = 2.0 * centerFmel[i] - lowerFmel[i];
    }

    // Перевод границ фильтров в шкалу герц
    vector<float> lowerF(centerF.size());
    vector<float> upperF(centerF.size());
    for (int i = 0; i < lowerFmel.size(); ++i)
    {
        lowerF[i] = 700.0 * (-1.0 + pow(10.0, (lowerFmel[i] / 2595.0)));
        upperF[i] = 700.0 * (-1.0 + pow(10.0, (upperFmel[i] / 2595.0)));
    }

    vector<bool> freq(fbHfcc.front().size());
    vector<bool> freq2(fbHfcc.front().size());
    for (int chan = 0; chan < centerF.size(); ++chan)
    {
        for (int j = 0; j < fbHfcc.front().size(); ++j)
        {
            freq[j] = (fftFreqs[j] > lowerF[chan]) && (fftFreqs[j] <= centerF[chan]);
            freq2[j] = (fftFreqs[j] > centerF[chan]) && (fftFreqs[j] < upperF[chan]);
```

```
                        fbHfcc[chan][j] = freq[j] * (fftFreqs[j] - lowerF[chan]) / (centerF[chan] - lowerF[chan]) \
                                + freq2[j] * (upperF[chan] - fftFreqs[j]) / (upperF[chan] - centerF[chan]);
                }
        }
        resFbHfcc = fbHfcc;
}

void dctm(int N, int M, vector <vector <float>> &DCT)
{
        vector <vector <float>> matrix1(N, vector<float>(M, 0));
        vector <vector <float>> matrix2(N, vector<float>(M, 0));
        for (int i = 0; i < N; ++i)
                for (int j = 0; j < M; ++j)
                {
                        matrix1[i][j] = i;
                        matrix2[i][j] = PI*(j+1.0-0.5) / M;
                        DCT[i][j] = cos(matrix1[i][j] * matrix2[i][j]) * sqrt(2.0 / M);
                }
}

void ceplifter(int N, int L, vector <float> &y)
{
        for (int i = 0; i < N; ++i)
                y[i] = 1 + 0.5 * L * sin(PI * i / L);
}

void get_feature_vector(vector<float> speech, const int fs, vector <vector <float>> &MFCCs_train)
{
        float alpha = 0.97;                     // коэффициент предварительной обработки
        int frame_duration = 100;               // длительность фрейма в мс
        int frame_shift = 50;                   // сдвиг фреймов в мс
```

```
int M = 40;                                    // количество фильтров
int L = 22;                                    // параметр выравнивания
float ERBScaleFactor = 0.75;          // коэффициент ширины фильтров
int fmin = 100;                                // минимальная частота, Гц
int fmax = 6250;                         // максимальная частота, Гц
int Ncc = 29;                                  // количество мел - кепстральных коэффициентов

filter(25, speech);

int Nw = round(0.001 * frame_duration * fs);
int Ns = round(0.001 * frame_shift * fs);

vector <vector <float> > frames;
vec2frames(speech, Nw, Ns, frames);
silence_removal(frames);
const int Nfft = pow(2, (int)ceil(log2(Nw)));
const int K = (Nfft / 2) + 1;
int i;
float *re = new float[Nfft];
float *im = new float[Nfft];

std::vector <std::vector <float>> MAG(Nfft, vector<float>(frames.front().size(), 0));
for (int j = 0; j < frames.front().size(); ++j)
{
        for (i = 0; i < Nfft; ++i)
        {
                re[i] = 0.0;
                im[i] = 0.0;
        }
        for (i = 0; i < 4410; ++i)
        {
```

```
                re[i] = frames[i][j];
        }

        FFT(re, im, Nfft, 13, FT_DIRECT);

        for (i = 0; i < Nfft; ++i)
        {
                MAG[i][j] = sqrt(re[i]*re[i] + im[i] * im[i]);
        }
}
delete[] re;
delete[] im;

vector<vector<float>> H(M, vector<float>(K, 0));
trifbank_hfcc(M, K, fmin, fmax, fs, ERBScaleFactor, H);

vector<vector<float>> FBE(H.size(), vector<float>(MAG.front().size(), 0));

MAG.erase(MAG.begin() + K, MAG.end());
MulMatrix(H, MAG, FBE);

vector<vector<float>> DCT(Ncc + 1, vector<float> (M, 0));
dctm(Ncc + 1, M, DCT);

vector<vector<float>> CC(DCT.size(), vector<float>(FBE.front().size(), 0));
for (int i = 0; i < FBE.size(); ++i)
        for (int j = 0; j < FBE.front().size(); ++j)
                FBE[i][j] = log(FBE[i][j]);
MulMatrix(DCT, FBE, CC);
vector<vector<float>> HFCCs = CC;
HFCCs.erase(HFCCs.begin());
```

```
	MFCCs_train = HFCCs;
}

void disteusq(vector <vector <float>> X, vector <vector <float>> Y, vector <vector <float>> &D)
{
	vector <vector<float>> x(X.front().size(), vector<float>(X.size()));
	vector <vector<float>> y(Y.front().size(), vector<float>(Y.size()));
	for (int i = 0; i < x.size(); ++i)
		for (int j = 0; j < x.front().size(); ++j)
			x[i][j] = X[j][i];
	for (int i = 0; i < y.size(); ++i)
		for (int j = 0; j < y.front().size(); ++j)
			y[i][j] = Y[j][i];
	int nx = x.size();
	int ny = y.size();
	int p = x.front().size();
	vector <vector<float>> d(nx, vector<float>(ny, 0));
	vector <vector <vector<float>>> z(nx, vector<vector<float>>(ny, vector<float>(p)));
	vector <vector <vector<float>>> zy(nx, vector<vector<float>>(ny, vector<float>(p)));
	for (int i = 0; i < nx; ++i)
		for (int j = 0; j < p; ++j)
			for (int k = 0; k < ny; ++k)
			{
				z[i][k][j] = x[i][j];
			}
	for (int i = 0; i < nx; ++i)
		for (int j = 0; j < ny; ++j)
			for (int k = 0; k < p; ++k)
			{
				zy[i][j][k] = y[j][k];
				z[i][j][k] -= zy[i][j][k];
```

```
                    d[i][j] += z[i][j][k] * z[i][j][k];
                }
        D = d;
}

void compare_vectors(vector <vector <float>> MFCCs_train[], int count, vector<float> &res)
{
        vector<int> N(count);
        for (int i = 0; i < count; ++i)
        {
                N[i] = MFCCs_train[i].front().size();
        }
        float threshold = 0;

        for (int i = 0; i < count - 1; ++i)
        {
                for (int j = 1; i < count; ++i)
                {
                        if (i == j)
                                continue;
                        vector <vector <float>> D;
                        disteusq(MFCCs_train[i], MFCCs_train[j], D);
                        float width = 0.5;
                        float d;
                        vector <vector <float>> F(N[i], vector<float>(N[j], FLT_MAX));
                        for (int k = 1; k <= N[i]; ++k)
                        for (int l = 1; l <= N[j]; ++l)
                        {
                                float t1 = (float)k / (float)(N[i]);
                                float t2 = (float)l / (float)(N[j]);
                                d = fabs(t1 - t2);
```

```
                        if (d <= width / 4.0)
                                F[k - 1][l - 1] = 1.0;
                        else
                        if (d <= width)
                                F[k - 1][l - 1] = 1.0 + d * 5.0;
                }
                vector <vector <float>> S(N[i], vector<float>(N[j], 0));
                for (int k = 0; k < N[i]; ++k)
                for (int l = 0; l < N[j]; ++l)
                        S[k][l] = D[k][l] * F[k][l];
                vector<float> minS(N[i], FLT_MAX);
                float sum = 0;
                for (int k = 0; k < N[i]; ++k)
                {
                        for (int l = 0; l < N[j]; ++l)
                        if (S[k][l] < minS[k])
                                minS[k] = S[k][l];
                        sum += minS[k] * minS[k];
                }
                float dist = sqrt(sum / S.size());
                res.push_back(dist);
                cout << "Расстояние между двумя векторами признаков " << i + 1 << " и " << j
+ 1 << " составляет " << dist << endl;
                }
        }
}

float get_threshold(string samples[TRAIN_COUNT])
{
        FILE *file;
        errno_t err;
        long sampleRate[TRAIN_COUNT];
```

```
vector<float> speech[TRAIN_COUNT];

for (int k = 0; k < TRAIN_COUNT; ++k)
{
	err = fopen_s(&file, samples[k].c_str(), "rb");
	if (err)
	{
		printf_s("Failed open file, error %d", err);
		return 0;
	}

	WAVHEADER header;

	fread_s(&header, sizeof(WAVHEADER), sizeof(WAVHEADER), 1, file);
	speech[k].resize(header.subchunk2Size / 4);
	int j = 0; int i = 0;
	for (std::vector<float>::iterator it = speech[k].begin(); it != speech[k].end(); ++it)
	{
		float a, b;
		fread_s(&a, sizeof(a), sizeof(a), 1, file);
		*it = a;
		i++;
		if (i == 4410) j++;
	}
	sampleRate[k] = header.sampleRate;
	fclose(file);
	get_feature_vector(speech[k], sampleRate[k], MFCCs_train[k]);
}
vector<float> dists;
compare_vectors(MFCCs_train, TRAIN_COUNT, dists);
float sum = 0;
```

```
	for (int i = 0; i < dists.size(); ++i)
		sum += dists[i];
	return sum;
}

void verify_samples(float threshold, float threshold2, vector<int> &res)
{
	WIN32_FIND_DATA FindFileData;
	HANDLE hf;
	FILE *file;
	errno_t err;
	hf = FindFirstFile("verify\\*.wav", &FindFileData);
	if (hf != INVALID_HANDLE_VALUE)
	{
		do
		{
			vector<float> speech;
			int sampleRate;
			vector <vector <float>> MFCCs;
			string filename = FindFileData.cFileName;
			err = fopen_s(&file, ("verify\\" + filename).c_str(), "rb");
			if (err)
			{
				printf_s("Failed open file, error %d", err);
				continue;
			}

			WAVHEADER header;

			fread_s(&header, sizeof(WAVHEADER), sizeof(WAVHEADER), 1, file);
			speech.resize(header.subchunk2Size / 4);
```

```
                    for (std::vector<float>::iterator it = speech.begin(); it != speech.end(); ++it)
                    {
                            float a, b;
                            fread_s(&a, sizeof(a), sizeof(a), 1, file);
                            *it = a;
                    }
                    sampleRate = header.sampleRate;
                    fclose(file);
                    get_feature_vector(speech, sampleRate, MFCCs);
                    float distmin = FLT_MAX;
                    for (int i = 0; i < TRAIN_COUNT; ++i)
                    {
                            vector <vector <float>> MFCCsmas[2] = { MFCCs_train[i], MFCCs};
                            vector<float> dists;
                            compare_vectors(MFCCsmas, 2, dists);
                            if (distmin > dists.front())
                                    distmin = dists.front();
                    }
                    if (distmin < threshold)
                    {
                            res.push_back(1);
                            cout << FindFileData.cFileName << " - статус: подтвержден, расстояние:
" << distmin << endl;
                    }
                    else if (distmin < threshold2)
                    {
                            res.push_back(2);
                            cout << FindFileData.cFileName << " - статус: сомнительно, расстояние:
" << distmin << endl;
                    }
                    else
                    {
```

```
                        res.push_back(0);
                        cout << FindFileData.cFileName << " - статус: отвергнут, расстояние: "
<< distmin << endl;
                    }

                } while (FindNextFile(hf, &FindFileData) != 0);
                FindClose(hf);
        }
}

int _tmain(int argc, _TCHAR* argv[])
{
        setlocale(LC_ALL, "");
        string samples[10] = { "train-01.wav", "train-02.wav", "train-03.wav", "train-04.wav","train-
05.wav",
                "train-06.wav", "train-07.wav", "train-08.wav", "train-09.wav" "train-10.wav"};
        float threshold = get_threshold(samples);
        float c1 = 1.1;
        threshold = (threshold / TRAIN_COUNT) * 1.33;
        float threshold2 = threshold * c1;
        cout << "Граничное расстояние подтверждения " << threshold << endl;
        cout << "Граничное расстояние результата \"Сомнительно\" " << threshold2 << endl;
        vector<int> results;
        verify_samples(threshold, threshold2, results);

        _getch();
        return 0;
}
```

Приложение Б. Журнал тестирования

Подготовка системы

Расстояние между двумя векторами признаков 1 и 2 составляет 5.75769

Расстояние между двумя векторами признаков 1 и 3 составляет 7.61817

Расстояние между двумя векторами признаков 2 и 3 составляет 10.8757

Граничное расстояние подтверждения 10.7515

Граничное расстояние результата "Сомнительно" 11.8267

Проверка своих образцов

Verify-01.wav - статус: подтвержден, расстояние: 4.5621

Verify-02.wav - статус: подтвержден, расстояние: 7.22936

Verify-03.wav - статус: подтвержден, расстояние: 5.33278

Verify-04.wav - статус: подтвержден, расстояние: 5.60323

Verify-05.wav - статус: подтвержден, расстояние: 9.80032

Verify-06.wav - статус: сомнительно, расстояние: 11.0213

Verify-07.wav - статус: подтвержден, расстояние: 5.87211

Verify-08.wav - статус: подтвержден, расстояние: 8.74221

Проверка чужих образцов

Всего образцов: 54

Verify-09.wav - статус: отвергнут, расстояние: 16.4679

Verify-10.wav - статус: отвергнут, расстояние: 17.8584

Verify-11.wav - статус: отвергнут, расстояние: 17.2158

Verify-12.wav - статус: отвергнут, расстояние: 17.4461

Verify-13.wav - статус: отвергнут, расстояние: 17.7064

Verify-14.wav - статус: отвергнут, расстояние: 18.8163

Verify-15.wav - статус: отвергнут, расстояние: 20.7464

Verify-16.wav - статус: отвергнут, расстояние: 21.0038

Verify-18.wav - статус: отвергнут, расстояние: 22.0348

Verify-19.wav - статус: отвергнут, расстояние: 18.7383

Verify-20.wav - статус: отвергнут, расстояние: 18.892

Verify-21.wav - статус: отвергнут, расстояние: 19.7334

Verify-22.wav - статус: отвергнут, расстояние: 19.2584

Verify-23.wav - статус: сомнительно, расстояние: 11.0544

Verify-24.wav - статус: отвергнут, расстояние: 22.0147

Verify-25.wav - статус: отвергнут, расстояние: 19.1601

Verify-26.wav - статус: отвергнут, расстояние: 20.542

Verify-27.wav - статус: отвергнут, расстояние: 22.0031

Verify-28.wav - статус: отвергнут, расстояние: 17.2488

Verify-29.wav - статус: отвергнут, расстояние: 16.8763

Verify-30.wav - статус: отвергнут, расстояние: 14.2234

Verify-31.wav - статус: отвергнут, расстояние: 17.4898

Verify-32.wav - статус: отвергнут, расстояние: 18.2047

Verify-33.wav - статус: отвергнут, расстояние: 18.4352

Verify-34.wav - статус: отвергнут, расстояние: 17.7496

Verify-35.wav - статус: отвергнут, расстояние: 18.1023

Verify-36.wav - статус: отвергнут, расстояние: 19.1174

Verify-37.wav - статус: отвергнут, расстояние: 20.87455

Verify-38.wav - статус: отвергнут, расстояние: 18.8163

Verify-39.wav - статус: отвергнут, расстояние: 15.5763

Verify-40.wav - статус: отвергнут, расстояние: 17.4877

Verify-41.wav - статус: отвергнут, расстояние: 20.2149

Verify-42.wav - статус: отвергнут, расстояние: 14.1997

Verify-43.wav - статус: отвергнут, расстояние: 22.5177

Verify-44.wav - статус: отвергнут, расстояние: 18.74418

Verify-45.wav - статус: отвергнут, расстояние: 18.5729

Verify-46.wav - статус: отвергнут, расстояние: 17.7416

Verify-47.wav - статус: отвергнут, расстояние: 15.8721

Verify-48.wav - статус: отвергнут, расстояние: 18.8163

Verify-49.wav - статус: отвергнут, расстояние: 16.7102

Verify-50.wav - статус: отвергнут, расстояние: 21.7772

Verify-51.wav - статус: отвергнут, расстояние: 20.7464

Verify-52.wav - статус: отвергнут, расстояние: 17.7132

Verify-53.wav - статус: отвергнут, расстояние: 17.7064

Verify-54.wav - статус: отвергнут, расстояние: 14.2234