МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.И. ЛЕНИНА»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к выпускной квалификационной работе

(форма ВКР)

Разработка последовательного и параллельного алгоритмов сжатия ЭМГ-данных

(тема ВКР)

Выполнил Д.М. Васильев
Руководил к.т.н. С.Г. Сидоров
Консультировали:

Заведующий кафедрой _____ Сидоров С.Г.
подпись Ф.И.О.

Иваново 2018 г.

Реферат

Количество страниц – 45, количество рисунков – 39, количество таблиц – 1, количество источников – 10.

Электромиограф (ЭМГ) — это прибор для исследования биоэлектрической активности мышц и нервных структур для обнаружения в них очагов заболевания у пациента. Объектом исследования являются данные, получаемые с такого прибора. Предполагается, что после снятия показаний, ЭМГ-устройство будет передавать их на компьютер по беспроводной связи для дальнейшей обработки. Целью данной работы является сжатие этих данных для экономии заряда аккумулятора в устройстве при их передаче. Задание было поставлено во время прохождения практики в компании «Нейрософт».

Основная часть данной работы разделена на несколько глав, где:

1. **Теория о миографии** рассказывает об основах анализа ЭМГ-кривых и на что нужно обращать внимание при сжатии.
2. Глава о **существующих алгоритмах сжатия** рассказывает об анализе теории по сжатию данных и обосновывает выбор одного из метода
3. В **разработке собственного алгоритма** показаны применяемые методы, которые помогают обойти некоторые недостатки дельта-кодирования, учитывая особенности ЭМГ-кривой
4. **Результаты** – иллюстрация работы конечного продукта собственного алгоритма сжатия
5. **Перспективы развития** – будущие доработки алгоритма
6. **Заключение** – анализ проделанной работы

Определения, обозначения, сокращения

- Электромиография (ЭМГ, ЭНМГ, миография, электронейромиография) — (мио - мышцы и графо - пишу), метод исследования биоэлектрических потенциалов, возникающих в скелетных мышцах человека и животных при возбуждении мышечных волокон; регистрация электрической активности мышц.
- ЭМГ-прибор – прибор, генерирующий ЭМГ-кривую
- Кодирование длин серий (англ. run-length encoding, RLE) или кодирование повторов — алгоритм сжатия данных, заменяющий повторяющиеся символы (серии) на один символ и число его повторов.
- Алгори́тм Ле́мпеля — Зи́ва — Ве́лча (Lempel-Ziv-Welch, LZW) — это универсальный алгоритм сжатия данных без потерь, созданный Авраамом Лемпелем, Яковом Зивом и Терри Велчем.
- Дельта-кодирование (англ. Delta encoding) — способ представления данных в виде разницы (дельты) между последовательными данными вместо самих данных.
- Зона видимости оси – разность между осью и точками, которая не превышает установленного значения.

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«Ивановский государственный энергетический университет имени В.И. Ленина»

Факультет ИВТ Кафедра ВВС

Направление подготовки (специальность) 09.03.01 «Информатика и вычислительная техника», профиль «Высокопроизводительные вычислительные системы на базе больших ЭВМ»

код, направление (специальность), профиль подготовки

УТВЕРЖДАЮ
Зав. кафедрой
«12» мая 2018 г.

З А Д А Н И Е

на выпускную квалификационную работу студента

Васильева Дмитрия Максимовича

(фамилия, имя, отчество)

1. Форма и тема ВКР пояснительная записка к выпускной квалификационной работе «Разработка последовательного и параллельного алгоритма сжатия ЭМГ-данных»

   утверждены приказом по университету от «19» апреля 2018 г. № 349-3
2. Срок сдачи ВКР 20 июня 2018г.
3. Содержание пояснительной записки (перечень подлежащих разработке вопросов)
   1. Рассмотреть существующие алгоритмы сжатия данных и области их применения
   2. Выбрать в качестве опорного алгоритма сжатия наиболее подходящий алгоритм с учетом особенностей его реализации в мобильном устройстве
   3. Адаптировать опорный алгоритм сжатия с учетом особенностей ЭМГ данных к решаемой задаче
   4. Разработать программу, реализующую различные варианты собственного алгоритма сжатия
   5. Провести сравнительный анализ эффективности разработанного алгоритма и универсальных распространенных алгоритмов

Дата выдачи задания «12» мая 2018 г.

Руководитель

Задание принял к исполнению

Оглавление

Введение

Электромиография (ЭМГ, ЭНМГ, миография, электронейромиография) — (мио - мышцы и графо - пишу), метод исследования биоэлектрических потенциалов, возникающих в скелетных мышцах человека и животных при возбуждении мышечных волокон; регистрация электрической активности мышц [1].

Электромиография выполняется с помощью специального прибора – электромиографа (рис. 1).

Рис. 1. Исследование биоэлектрических потенциалов с помощью электромиографа

В ряде случаев использование стационарного электромиографа невозможно (например, при измерении ЭМГ в движении). Вместо стационарного электромиографа предпочтительнее оказывается использование его беспроводной (мобильной) версии.

Для мобильных электромиографов актуальной является задача продления времени работы от аккумулятора, что напрямую связано с объемами передаваемых данных по радиоканалу.

При большом количестве одновременно работающих электромиографов с большим числом одновременно обрабатываемых датчиков, актуальной также является задача обработки больших массивов данных в режиме реального времени.

Исходя из вышеперечисленного поиск оптимальных алгоритмов упаковки ЭМГ данных является актуальной задачей.

Цель данной работы – разработка энергоэффективного алгоритма, сжимающего сигналы, поступающие с ЭМГ-устройств, для ускорения передачи данных и экономии заряда аккумулятора, что позволит в дальнейшем запустить этот прибор в производство.

Для достижения цели необходимо выполнить следующие задачи:

- Сделать обзор существующих алгоритмов сжатия;
- Разработать алгоритм сжатия данных, получаемых с ЭМГ, для компании «Нейрософт»;
- Сравнить эффективность разработанного алгоритма с существующими алгоритмами. Новый алгоритм должен иметь большую эффективность (коэффициент сжатия).

На данный момент существует множество алгоритмом сжатия данных, которые делятся на два вида: «с потерями информации» и «без потерь информации». Большей эффективности можно добиться при использовании алгоритмов с потерями за счет удаления избыточных данных, которые затем можно восстановить, но с определенной погрешностью. Также применяют комбинации из этих методов. Разработка собственного алгоритма сжатия будет базироваться на «дельта-сжатии», т.к. для рассматриваемых сигналов он представляется наиболее оптимальным (менее затратным по вычислительной части).

Необходимость разработки собственного алгоритма сжатия заключается в том, что существующие универсальные алгоритмы не учитывают всех особенностей ЭМГ-сигнала, вследствие чего не могут обеспечить максимальную эффективность сжатия.

Постановка задачи

Цель работы: разработать энергоэффективный алгоритм сжатия ЭМГ данных.

Для достижения данной цели необходимо решить следующие задачи:

1. Анализ существующих алгоритмов сжатия
2. Разработка оптимизированного алгоритма сжатия данных
3. Разработка программы по написанному алгоритму
4. Сравнение эффективности разработанного алгоритма с существующими алгоритмами

Электромиография

Так как алгоритмы с потерями информации сжимают данные эффективнее, чем алгоритмы без потерь, было принято решение использовать алгоритмы сжатия с потерями. Для повышения эффективности сжатия необходимо учитывать информативность различных областей графиков ЭМГ.

При анализе ЭМГ учитывается**:**

- частота биопотенциалов
- величина их амплитуды
- общая структура осциллограмм - монотонность осцилляций или их расчлененность на залпы, частота и длительность этих залпов и пр. [8]

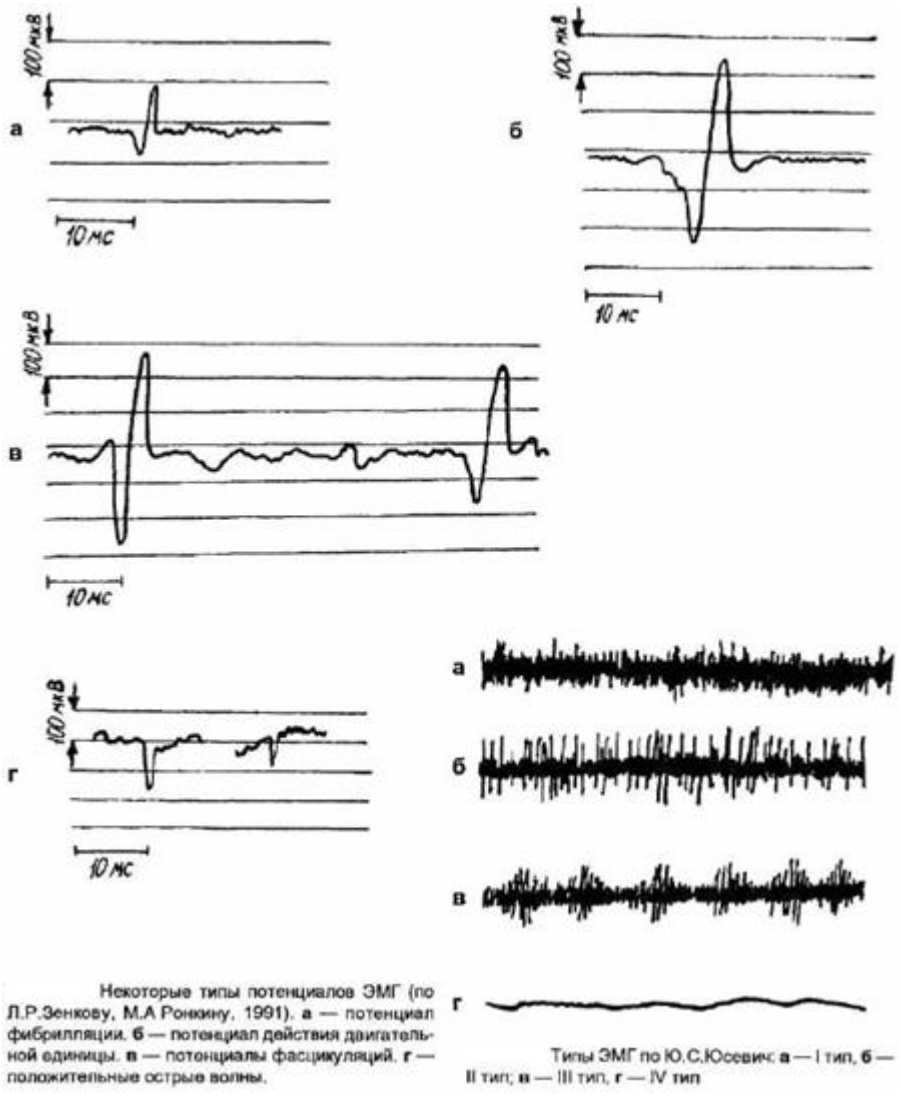


Рис. 2. Типы миограмм

Оценка ЭМГ-кривой основывается на общей характеристике кривой, максимальной амплитуде колебаний определению, суммарной электрической активности мышц и отнесению ЭМГ к тому или иному классу. В отечественной медицине наиболее широко используют методические приемы и способы анализа, выработанные Ю.С. Юсевичем.

Автор выделяет четыре типа ЭМГ:

I тип — интерференционная кривая, представляющая собой высокочастотную (до 50—100 кол/с) полиморфную активность, возникающую при произвольном сокращении мышцы. Снижение амплитуды интерференционной ЭМГ наблюдается при первично мышечных и при аксональных поражениях.

II тип — редкая (до 20—40 кол/с) ритмическая активность, характерная для поражения передних рогов спинного мозга.

III тип — усиление частых колебаний в покое, группировка их в ритмические разряды, появление вспышек колебаний на фоне ЭМГ произвольного мышечного сокращения; этот тип характерен для различного рода супраспинальных расстройств.

Тип IV ЭМГ — полное биоэлектрическое молчание в покое, при тоническом напряжении или попытке к произвольному сокращению; характеризует полный паралич мышцы.

По миограмме можно определить заболевание у пациента. Когда мышца только начинает сокращаться, величина амплитуды этих колебаний составляет порядка 100-150 мкВ, а в состоянии максимального сокращения – 1000-3000 мкВ. Эти показатели напрямую зависят от того, какого возраста человека и какова его физического развития. Исказить показания прибора может толстый слой подкожного жира в области исследования и заболевания свертывающей системы крови.

- При первичном мышечном заболевании – миозитах, прогрессирующих мышечных дистрофиях регистрируется снижение амплитуды осцилляций. Снижение амплитуды зависит от соответственно то того, как поражены мышци - в тяжелых случаях до 20—150 мкВ при максимальном возбуждении, а при медленно прогрессирующем течении заболевания и в начальных стадиях - до 500мкВ. На локальной ЭМГ в это же время число потенциалов находится в пределах нормы, однако их амплитуда и длительность снижены. Это связано с уменьшением количества нормальных мышечных волокон, способных к сокращению.
- При поражениях периферических нервных стволов, а именно: наследственных, метаболических (в том числе диабетических), токсических (в том

числе и алкогольных) полинейропатий глобальная ЭМГ демонстрирует урежение осцилляций, а также неравномерные по амплитуде и частоте одиночные потенциалы. Общий фон ЭМГ характеризуется низкоамплитудной активностью. На локальной ЭМГ регистрируются полифазные потенциалы действия с практически нормальными характеристиками. При гибели большинства нервных волокон постепенно биоэлектрическая активность мышц снижается до полного биоэлектрического молчания – потенциалы отсутствуют.

- Спинальные амиотрофии – наследственных заболеваниях мотонейронов спинного мозга с мышечной слабостью, характеризуются подергиванием мышц на локальной ЭМГ спонтанной активностью в виде потенциалов фибрилляций, острых волн, увеличение амплитуды. Глобальная же ЭМГ может обнаружить в покое спонтанную биоэлектрическую активность (фасцикуляции 100 – 400 мкВ), а при максимальном напряжении высокоамплитудный ритмичный потенциал – «ритм частокола».
- При миотонических синдромах – группе наследственных заболеваний с замедленным расслаблением мышц (дистрофическая миотония, миотония Томсена, Беккера, Эйленбурга…) мышцы легко возбудимы и на ЭМГ регистрируется миотоническое последействие: низкоамплитудная, высокочастотная электрическая активность в течение длительного времени после прекращения произвольного мышечного сокращения с медленным постепенным угасанием. На локальной ЭМГ при миотонии возникает повышенная возбудимость мышечных волокон - серия потенциалов действия эквивалентной амплитуды в ответ на введение игольчатого электрода.
- Миастенические синдромы – нарушении нервно – мышечной синаптической передачи вызывают на ЭМГ нарастающее снижение амплитуды вызванного мышечного потенциала при повторной ритмической стимуляции.
- Эссенциальный тремор и болезнь Паркинсона выглядят на поверхностной ЭМГ, как серия ритмичных «залпов» повышения амплитуды колебаний и последующего ее снижения. Продолжительность и частота таких залпов напрямую зависит от того, где локализован патологический процесс.

Для более точной диагностики используется совместное исследование – электромиографии с электронейрографией – электронейрография [2][3].

Существующие алгоритмы сжатия

В данной главе рассмотрены такие алгоритмы, как RLE-кодирование, LZW-кодирование и Дельта-кодирование [3].

RLE-кодирование.

Кодирование длин серий (англ. run-length encoding, RLE) или кодирование повторов — алгоритм сжатия данных, заменяющий повторяющиеся символы (серии) на один символ и число его повторов. Серией называется последовательность, состоящая из нескольких одинаковых символов. При кодировании (упаковке, сжатии) строка одинаковых символов, составляющих серию, заменяется строкой, содержащей сам повторяющийся символ и количество его повторов [4]. Например, изображение ночного неба может содержать длинные серии символов, представляющих темный фон, а цифровая музыка может иметь длинную серию одинаковых бит в песнях. На рисунке 3 проиллюстрирован принцип этого кодирования для последовательности данных с частым повторением серии нулей. Всякий раз, когда нуль встречается во входных данных, в выходной файл записываются два значения: нуль, указывающий на начало кодирования, и число нулей в серии. Если среднее значение длины серии больше двух, происходит сжатие. С другой стороны, множество одиночных нулей в данных может привести к тому, что кодированный файл окажется больше исходного.

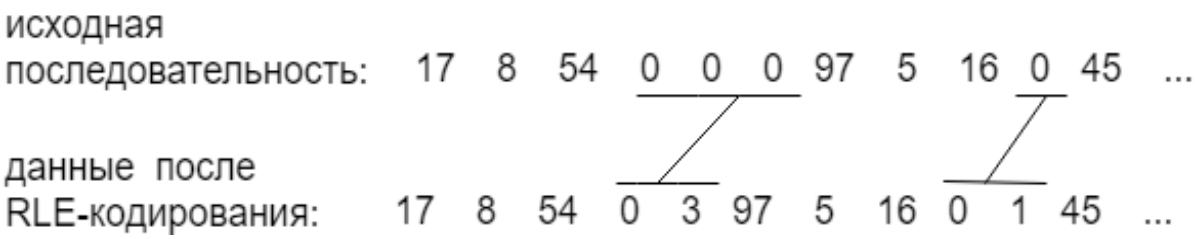


Рис. 3. Демонстрация RLE-кодирования

Очевидно, что этот алгоритм возможно эффективно использовать только там, где одинаковая информация повторяется часто.

Ниже, на рисунке 4, представлены блок-схемы кодирования декодирования по алгоритму RLE. Слева представлен алгоритм сжатия, а справа распаковки. Оба алгоритма довольны просты в своем исполнении.

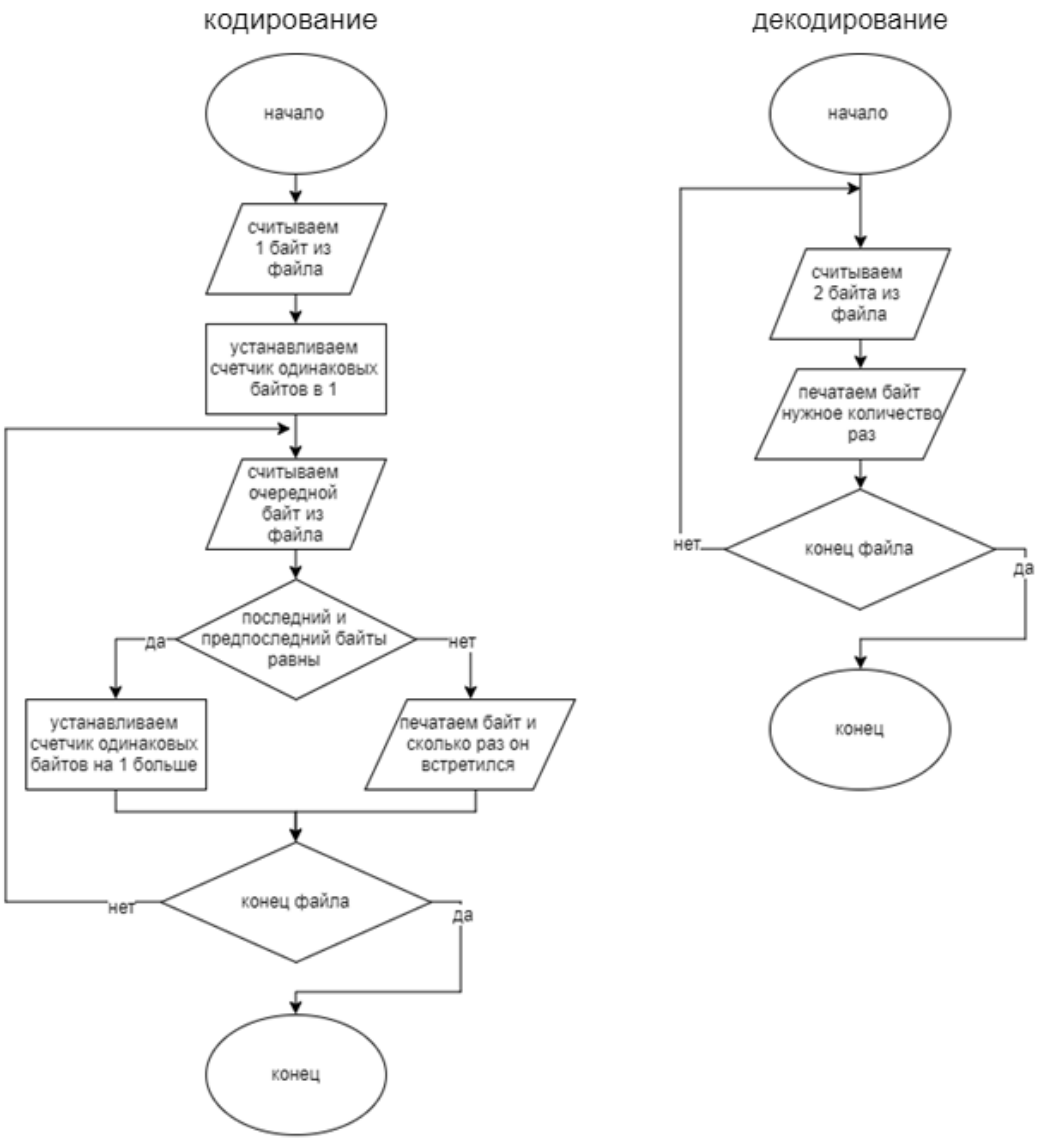


Рис. 4. Блок-схемы алгоритма RLE

Алгоритм LZW

Алгори́тм Ле́мпеля — Зи́ва — Ве́лча (Lempel-Ziv-Welch, LZW) — это универсальный алгоритм сжатия данных без потерь, созданный Авраамом Лемпелем, Яаковом Зивом и Терри Велчем. Он был опубликован Велчем в 1984 году в качестве улучшенной реализации алгоритма LZ78, опубликованного Лемпелем и Зивом в 1978 году. Алгоритм разработан так, чтобы его можно было быстро реализовать, но он не обязательно оптимален, поскольку он не проводит никакого анализа входных данных [5]. Процесс сжатия выглядит следующим образом: последовательно считываются символы входного потока и происходит проверка, существует ли в созданной таблице строк такая строка. Если такая строка существует, считывается следующий символ, а если строка не существует, в поток заносится код для предыдущей найденной строки, строка заносится в таблицу, а поиск начинается снова.

Например, если сжимают байтовые данные (текст), то строк в таблице окажется 256 (от 0 до 255). Новые строки формируют таблицу последовательно, т. е. можно считать индекс строки ее кодом.

Для декодирования на вход подается только закодированный текст, поскольку алгоритм LZW может воссоздать соответствующую таблицу преобразования непосредственно по закодированному тексту. Алгоритм генерирует однозначно декодируемый код за счет того, что каждый раз, когда генерируется новый код, новая строка добавляется в таблицу строк. LZW постоянно проверяет, является ли строка уже известной, и, если так, выводит существующий код без генерации нового. Таким образом, каждая строка будет храниться в единственном экземпляре и иметь свой уникальный номер. Следовательно, при декодировании во время получения нового кода генерируется новая строка, а при получении уже известного, строка извлекается из словаря.

Кодирование

- Начало.
- Шаг 1. Все возможные символы заносятся в словарь. Во входную фразу X заносится первый символ сообщения.
- Шаг 2. Считать очередной символ Y из сообщения.
- Шаг 3. Если Y — это символ конца сообщения, то выдать код для X, иначе:

    - Если фраза XY уже имеется в словаре, то присвоить входной фразе значение XY и перейти к Шагу 2
    - Иначе выдать код для входной фразы X, добавить XY в словарь и присвоить входной фразе значение Y. Перейти к Шагу 2.
- Конец.

Декодирование

- Начало.
- Шаг 1. Все возможные символы заносятся в словарь. Во входную фразу X заносится первый код декодируемого сообщения.
- Шаг 2. Считать очередной код Y из сообщения.
- Шаг 3. Если Y — это конец сообщения, то выдать символ, соответствующий коду X, иначе:
    - Если фразы под кодом XY нет в словаре, вывести фразу, соответствующую коду X, а фразу с кодом XY занести в словарь.
    - Иначе присвоить входной фразе код XY и перейти к Шагу 2
- Конец.

Преимущества:

- Алгоритм является однопроходным.
- Для декомпрессии не надо сохранять таблицу строк в файл для распаковки. Алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов.

Недостатки:

- Алгоритм не проводит анализ входных данных.

Дельта-кодирование

Дельта-кодирование (англ. Delta encoding) — способ представления данных в виде разницы (дельты) между последовательными данными вместо самих данных [6]. На рисунке 4 приводится пример работы этого механизма. Первое значение в кодируемом файле является совпадает с исходным. Все последующие значения в кодируемом файле равны разности между соответствующим и предыдущим значениями входного файла.

исходная последовательность:	17 19 24 24 24 21 15 10 89 95 96 96 95 ...
данные после Дельта-кодирования:	17 2 5 0 0 -3 -6 -5 79 6 1 0 0 -1 ...

Рис. 5. Демонстрация Дельта-кодирования

Дельта-кодирование используется для сжатия данных, если значения исходного файла изменяются плавно, т.е. разность между следующими друг за другом величинами невелика. Это условие не выполняется для текста ASCII и исполняемого кода, но хорошо подходит в тех случаях, когда информация поступает в виде сигнала. Например, на рисунке 6 показан фрагмент аудиосигнала, где все выборки принимают значения в диапазоне от 127 до -127. На рисунке 7 представлен кодированный вариант этого сигнала, основное отличие которого от исходного сигнала заключается в меньшей амплитуде. Если исходный сигнал не меняется или меняется линейно, в результате дельта-кодирования появятся серии выборок с одинаковыми значениями, с которыми может работать RLE-алгоритм [7].

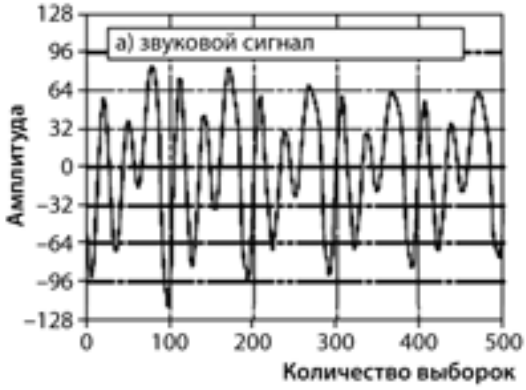


Рис. 6. Исходный сигнал

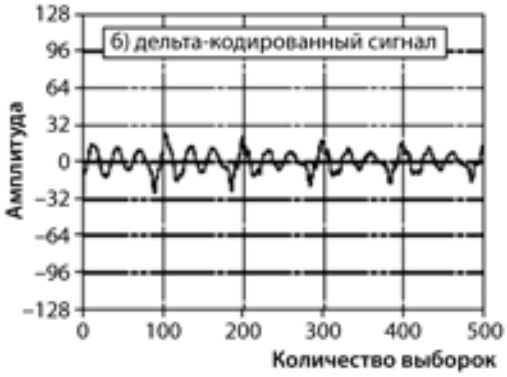


Рис. 7. Кодированный сигнал

Так выглядит принципиальные блок-схемы для кодирования и декодирования по этому методу (рис. 8):

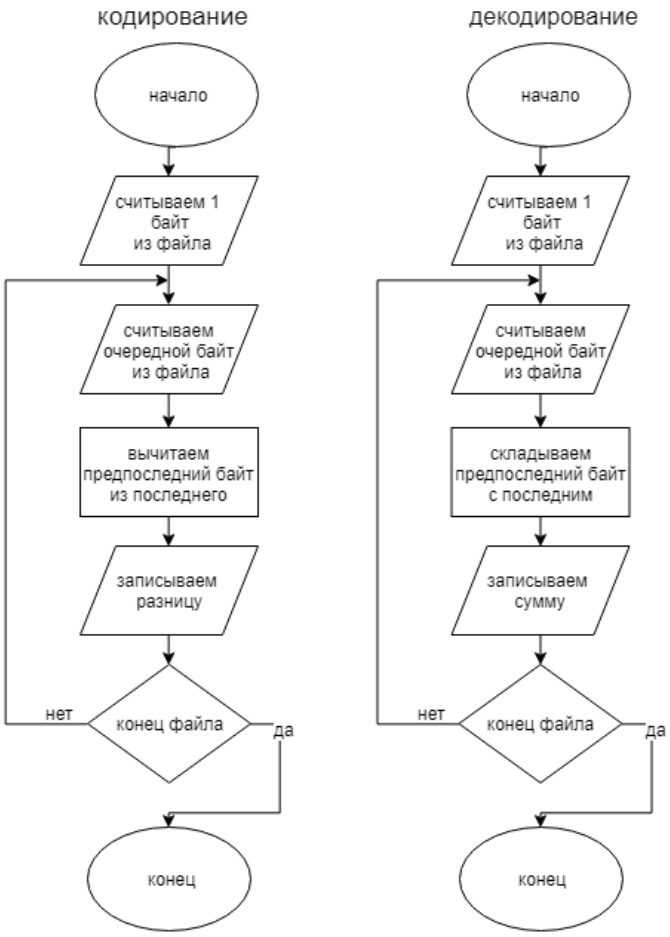


Рис. 8. Блок-схема Дельта кодирования

Исходя из вышеперечисленных данных, можно сделать вывод, что для сжатия ЭМГ-сигнала лучше всего подходит этот метод.

Разработка собственного алгоритма сжатия

В рассматриваемом случае главным недостатком дельта-кодирования является то, что сигнал, поступающий с ЭМГ-устройства, может изменяться как плавно, так и резко (Рис. 9).

Рис. 9. Форма ЭМГ сигнала

Из-за этого недостатка график после применения дельта сжатия выглядит так (Рис. 10):

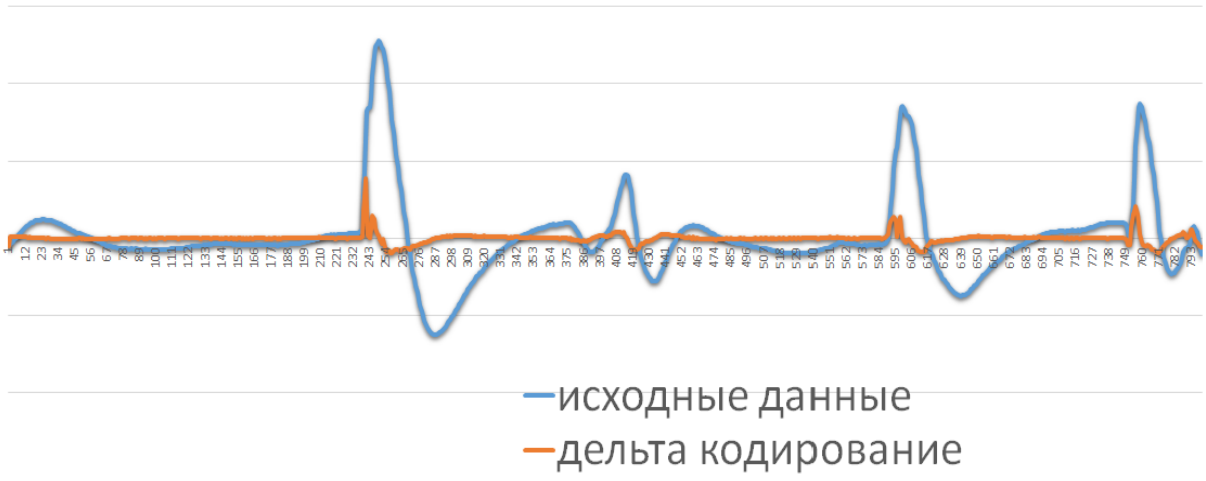


Рис. 10. Кодирование ЭМГ сигнала

Как видно на рисунке, там, где график изменяется плавно, сжатый сигнал колеблется около нуля, но, когда график меняется резко, сжатый сигнал начинает колебаться сильнее и принимает большие значения по модулю. За счет этого при дельта кодировании приходится хранить в памяти большую разрядную сетку, а значит и кодировать значения сигнала большим количеством бит. Чтобы обойти это, можно применить 2 способа: либо смещать ось абсцисс вверх или вниз, так чтобы входило

как можно больше элементов, идущих подряд, либо поворачивать ось в сторону возрастания графика.

Первый метод реализуется довольно простым способом путем добавления положительного или отрицательного смещения один раз, когда второй способ требует прибавлять смещения на каждом шаге и занимает больше машинного времени для расчета поворота оси. Однако, при правильных расчетных формулах, он показывает большую эффективность упаковки данных и тем самым может сэкономить заряд аккумулятора у ЭМГ-устройства.

Все эти операции служат для того, чтобы можно было записывать разницу между соседними значениями, используя меньшее количество бит, т.е. используя меньшую разрядную сетку.

В обоих случаях приходиться использовать служебные байты для обозначения поворота или смещения оси. Это говорит о том, что если поворотов или смещений будет много, то служебные байты будут расти в количестве и, соответственно, эффективность будет падать.

Ниже представлено подробное описание двух способов.

Смещение оси

Для понимания этой идем будет легче ориентироваться на рисунок:

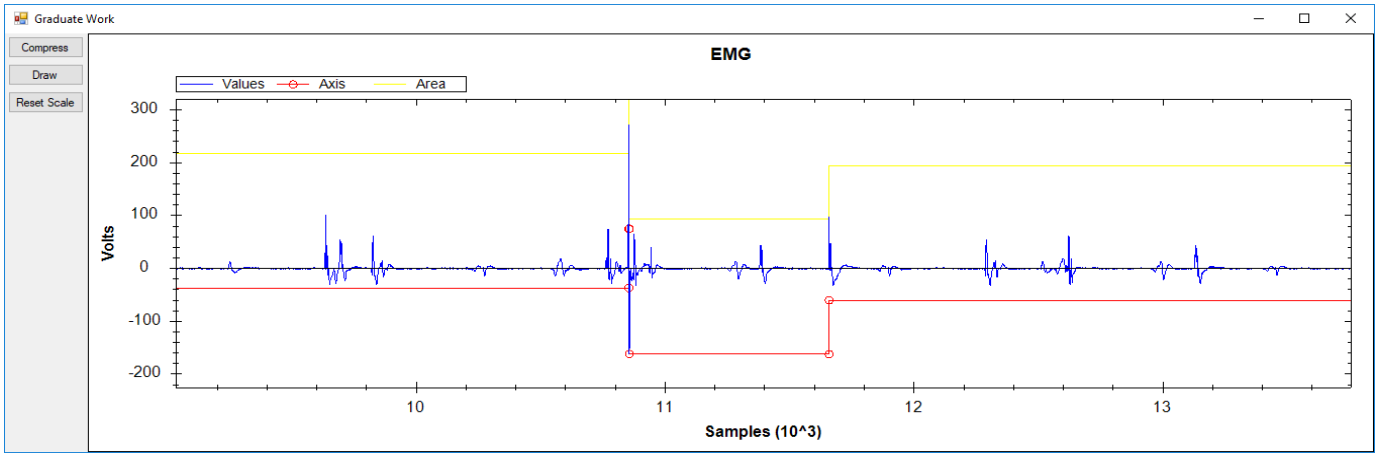


Рис. 11 Иллюстрация смещения оси абсцисс

Красной линией обозначена ось абсцисс, а желтой – область видимости, в которой мы можем расположить значения, используя определенное количество бит (в данном случае 8).

Изначально, ось находится в своем обычном положении, т.е. смещение равно нулю. Как только значения сигнала выходят за «область видимости» оси, она начинает смещается по направлению возрастания графика и уже разница считается от нового положения. За счет этого простого действия теперь не приходится кодировать все значения большим количеством бит, т.к. расстояние между осью и графиком будет сокращено, и остается только указывать смещения оси.

Чтобы выбрать направление смещения, нужно поступать в соответствии с данной блок-схемой (Рис.12) (*в условии говорится о количестве бит, которые требуются для записи самого большого по модулю числа).

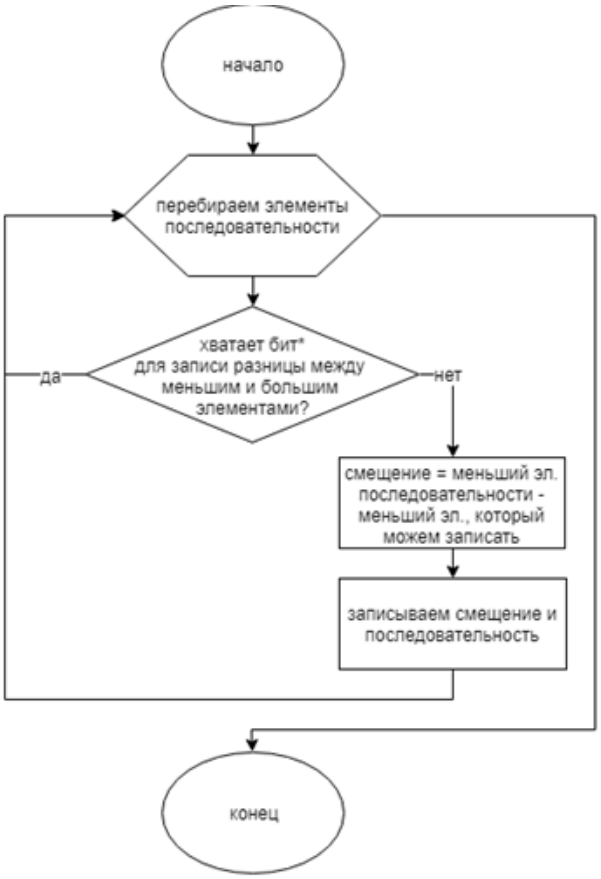


Рис. 12. Блок-схема выбора смещение оси

Ниже представлен пример того, как можно реализовать данный алгоритм на языке программирования C#:

```
            const sbyte serviceByte = sbyte.MinValue;
            const sbyte maxDel = sbyte.MaxValue;
            const sbyte minDel = sbyte.MinValue + 1;
            const short maxNumbers = maxDel - minDel;

            short[] sequence = new short[1024];
            short last = binRdr.ReadInt16();

            do //beginning of the main cycle
            {
                sequence[0] = last; //sequence of bytes
                short realMin = last, realMax = last; //min and max items in the
sequence
                int N = 1;
                do //check if we can write the difference between max and min
                {
                    short cur = binRdr.ReadInt16();
                    sequence[N] = cur;
                    N++;
                    if (cur < realMin)
                        if (realMax - cur < maxNumbers) //maxNumbers - the
largest difference
```

```
                realMin = cur;
            else
            {
                last = cur;
                N--;
                break;
            }
        if (cur > realMax)
            if (cur - realMin < maxNumbers)
                realMax = cur;
            else
            {
                last = cur;
                N--;
                break;
            }
    } while (binRdr.PeekChar() > -1);
    short offset = (short)(realMin - minDel);
   //data writing
    binWrtr.Write(serviceByte);
    binWrtr.Write(offset);
    for (int i = 0; i < N; i++)
    {
        sbyte fakeDel = (sbyte)(sequence[i] - offset);
        binWrtr.Write(fakeDel);
    }
} while (binRdr.PeekChar() > -1);
```

Однако, у этого метода есть недостаток: в том месте, где график смещается в одну сторону на протяжении длинного промежутка времени, ось приходится смещать часто, что ведет за собой потерю эффективности, т.к. используется большое количество служебных байт. Рисунок демонстрирует подобный случай:

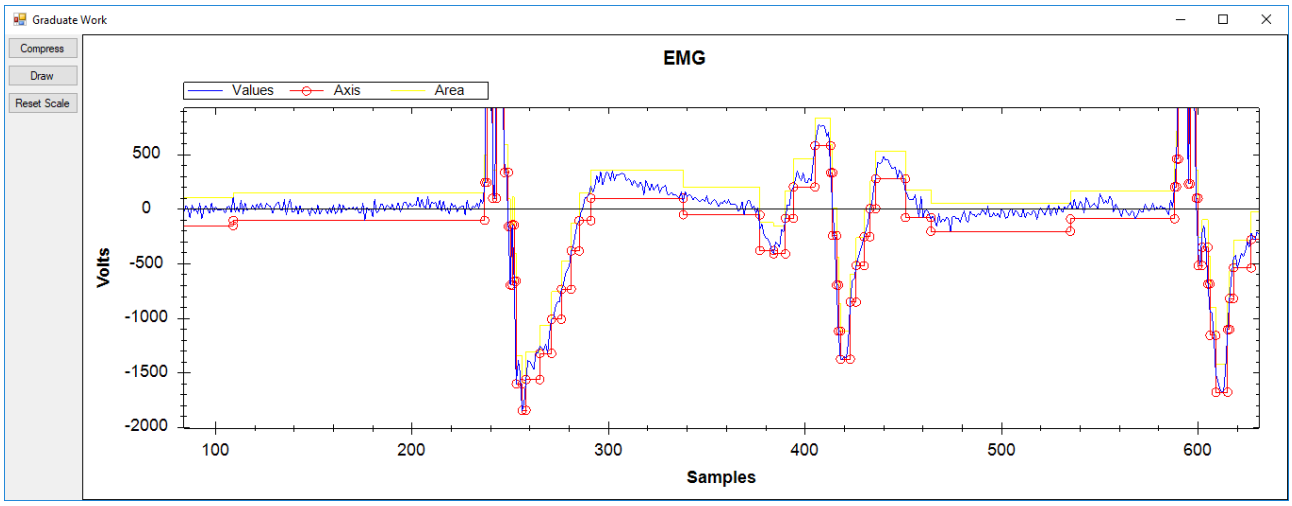


Рис. 13. Демонстрация недостатка смещение оси

Поворот оси

Второй способ – это поворот оси по направлению графика. Идея точно такая-же только вместо того, чтобы смещать всю ось целиком, здесь меняется только ее угол поворота.

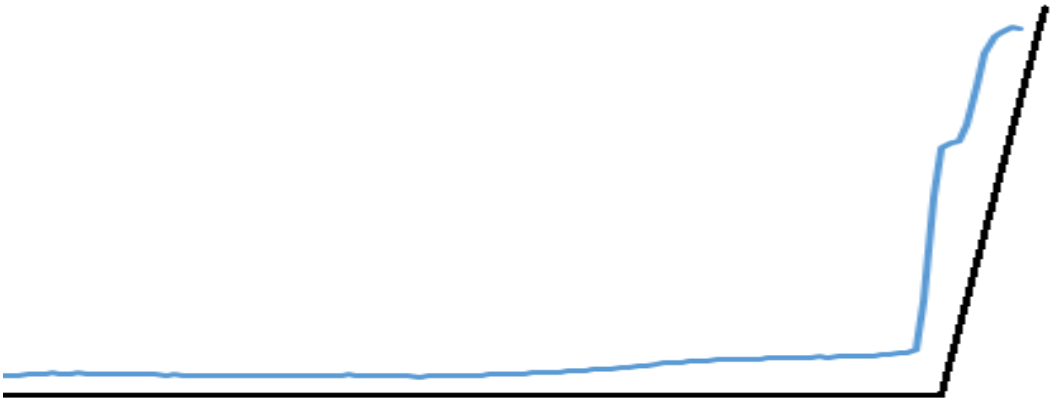


Рис. 14. Демонстрация поворота оси

Данный способ лишен недостатка, который есть у простого смещения за счет того, что это смещение прибавляется на каждом шаге и, соответственно, «ступенек», как в первом варианте, быть не может

Алгоритм для поиска оптимального угла поворота:

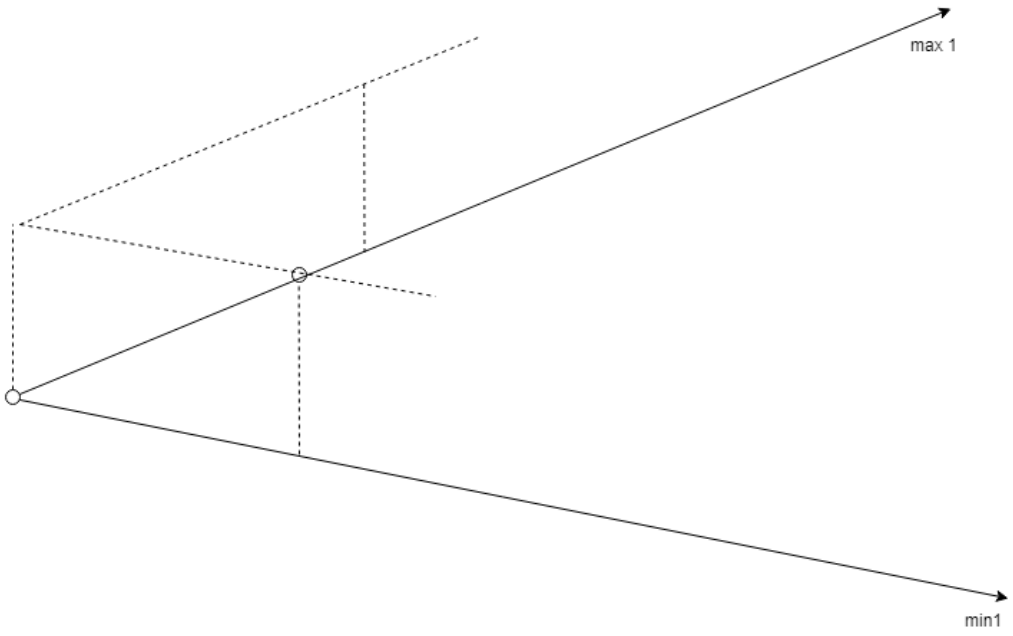


Рис. 15. Изначальный выбор max и min углов поворота оси

Чтобы выбрать оптимальный угол поворота оси, изначально выбирается максимальный и минимальный углы, которые основываются на двух точках. Максимальный угол – это тот угол, при котором ось проходит через эти 2 точки, а минимальный – при котором 2-ая точка лежит на границе зоны видимости оси (пунктирная линия). Эти углы будут опорными значениями. За счет этого сразу видно максимальный диапазон, на который можно поворачивать оси. Иначе, если максимальную поднять выше или минимальную опустить ниже, то вторая точка не будет входить в зону видимости.

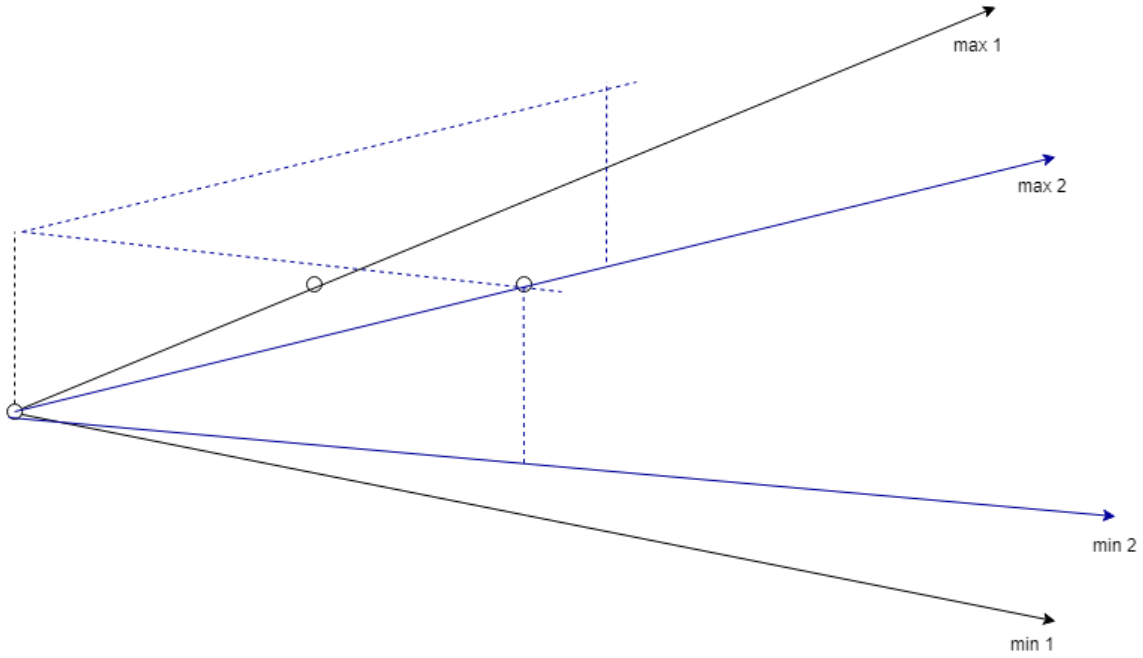


Рис. 16. Третья точка

Далее берется третья точка и углы порота корректируются, относительно нее. Т.к. в данном случае она оказывается между минимальным и максимальным углами поворота, корректируется как min1, так и max1. Т.е. максимальный угол поворота уменьшается, а минимальный увеличивается.

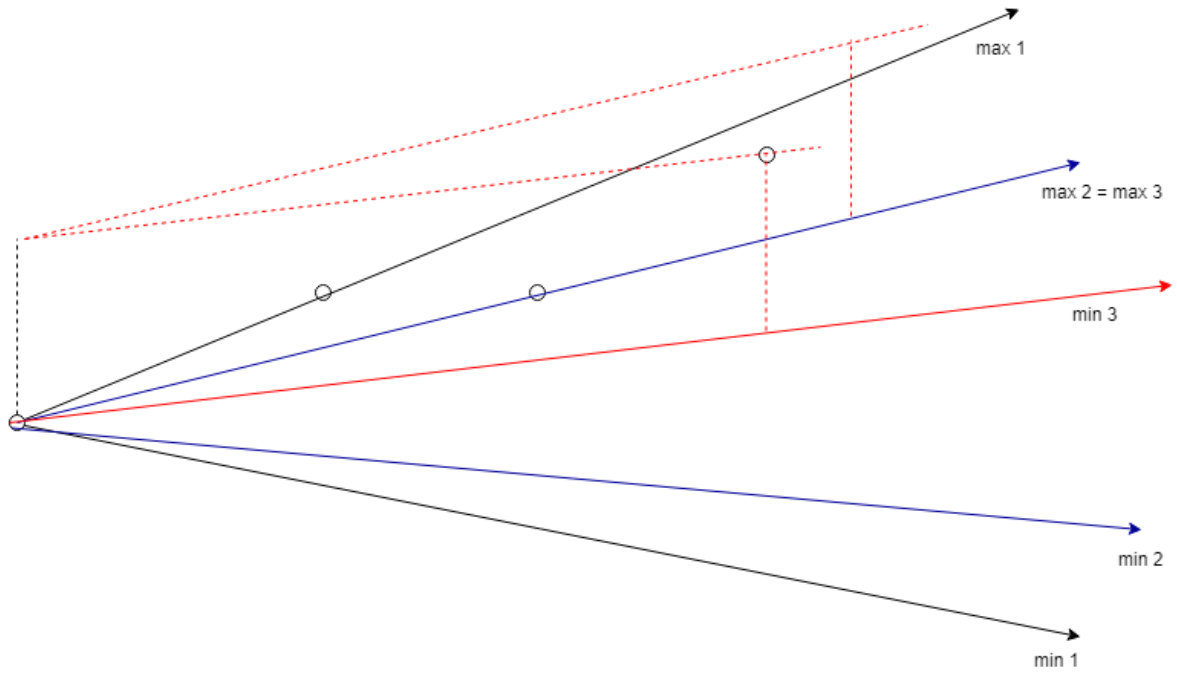


Рис. 17 Четвертая точка

Следующая точка находится выше оси с максимальным углом поворота. В следствии этого мы можем изменять наклон только у минимальной оси. Если же изменять и у максимальной, то предыдущие точки могут пропасть из зоны видимости, поэтому этого делать нельзя.

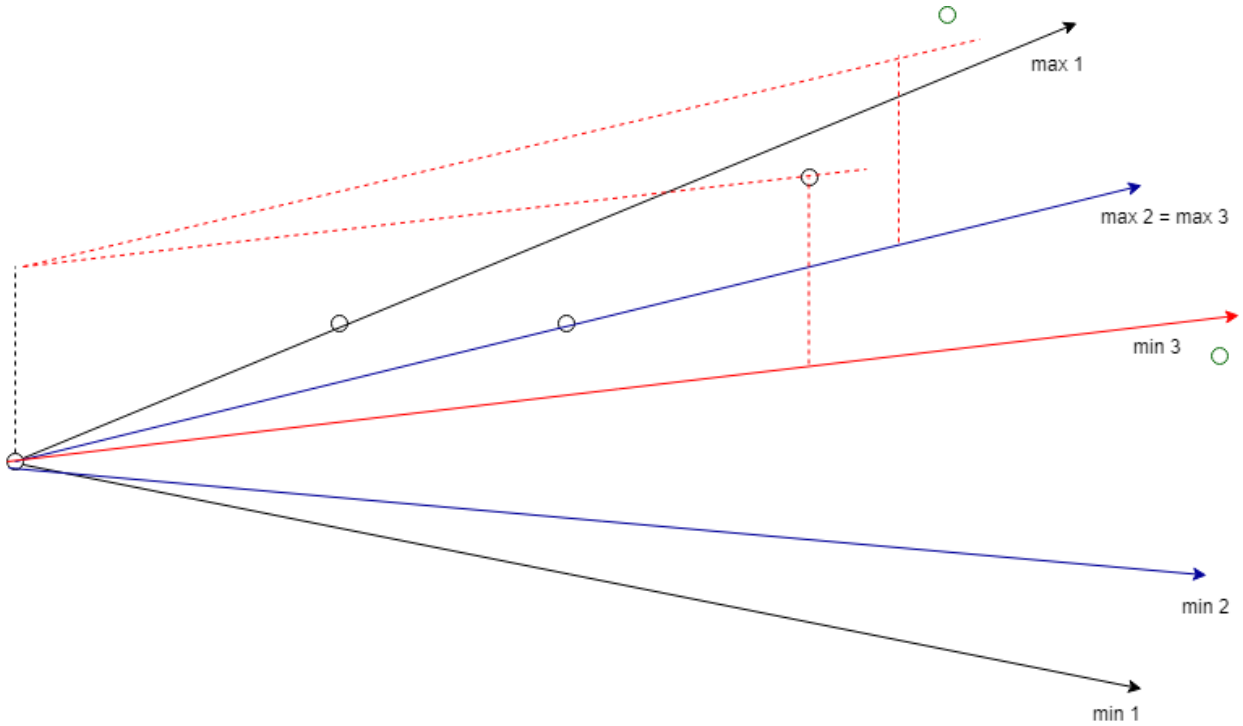


Рис. 18. Точки, не входящие в область видимости

Остальные точки уже не входят в область видимости т.к. они находятся либо выше максимальной оси, либо ниже минимальной. Это говорит о том, что на этом моменте нужно выбирать максимальный или минимальный угол поворота и потом поворачивать ось снова.

В итоге можно сформировать правило по выбору угла:

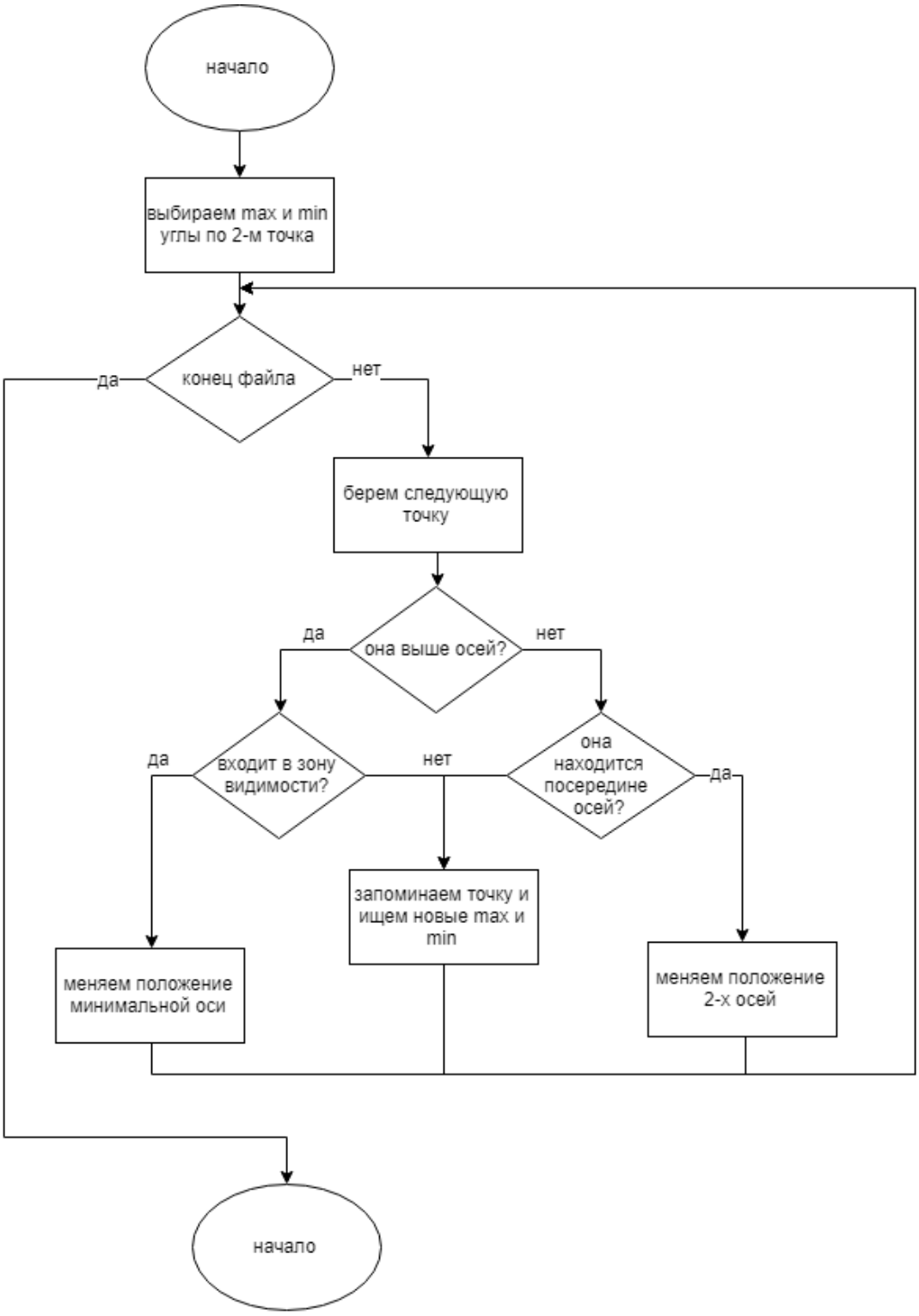


Рис. 19. Блок-схема по выбору угла поворота

Пример реализации такого алгоритма на языке C#:

```
            int totalOffset = last; //overall offset
            short offset = 0; //current offset
            bool frst = true;
            bool isEndOfFile = true;

            do
            {
                sequence.Clear(); //sequence of points
                short first;
                if (frst) //if it's the first passing
                    first = binRdr.ReadInt16();
                else
                    first = last;
                sequence.Add(first);

                //calculating offsets
                short offsetMin = Convert.ToInt16(first - maxNumbers -
totalOffset);
                short offsetMax = Convert.ToInt16(first - totalOffset);

                isEndOfFile = true;

                while (binRdr.PeekChar() > -1)
                {
                    short cur = binRdr.ReadInt16(); //current point
                    sequence.Add(cur);
                    //above all axes
                    if (cur >= totalOffset + offsetMax * sequence.Count() &&
                        cur <= totalOffset + offsetMax * sequence.Count() +
maxNumbers)
                    {
                        double tmp = Convert.ToDouble(cur - maxNumbers -
totalOffset) / sequence.Count();
                        short newMin = Convert.ToInt16(Math.Ceiling(tmp));
                        if (newMin > offsetMin)
                            offsetMin = newMin;
                    }
                    //between axes
                    if (cur <= totalOffset + offsetMax * sequence.Count() &&
                        cur >= totalOffset + offsetMin * sequence.Count())
                    {
                        double tmp = Convert.ToDouble(cur - totalOffset) /
sequence.Count();
                        short newMax = Convert.ToInt16(Math.Floor(tmp));
                        if (newMax < offsetMax)
                            offsetMax = newMax;
                        tmp = Convert.ToDouble(cur - maxNumbers - totalOffset) /
sequence.Count();
                        short newMin = Convert.ToInt16(Math.Ceiling(tmp));
                        if (newMin > offsetMin)
                            offsetMin = newMin;
                    }
                    //out of range
                    if (cur > totalOffset + offsetMax * sequence.Count() +
maxNumbers ||
                        cur < totalOffset + offsetMin * sequence.Count() ||
offsetMax < offsetMin)
```

```
            {
                last = cur;
                frst = false;
                isEndOfFile = false;
                break;
            }
        }
        offset = offsetMax; //choosing an efficien offset
```

После выполнение данного алгоритма получается такой график:

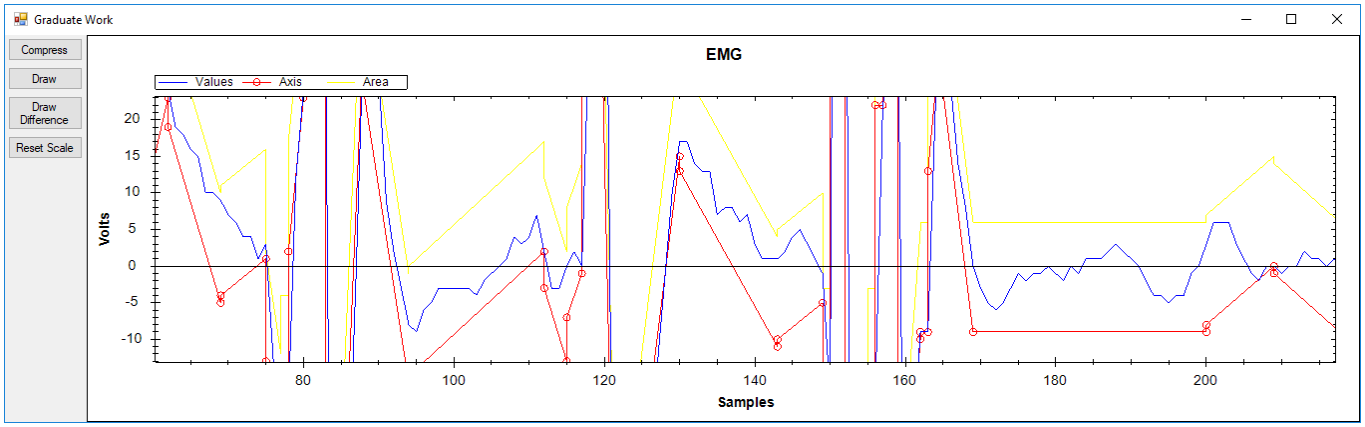


Рис. 20. График с поворотом оси

На рисунке видно, что при таком подходе поворотов становится меньше, вследствие чего, экономится память, т.к. служебных байтов становится меньше.

Перевод значений в другой диапазон

Помимо всего, ЭМГ-устройство снимает показания с пациента с большей точность, чем нужно для специалиста, анализирующего данные [6] (на выходе получаются значения типа double, причем все значения находятся около нуля). Следовательно, все значения можно записывать с меньшей точность и распределить по большему диапазону (Рис. 21):

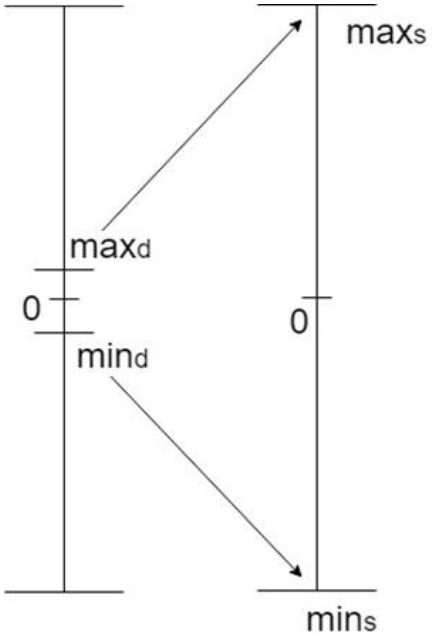


Рис. 21. Масштабирование данных

Слева изображен диапазон double, где maxd и mind – это максимальное и минимальное значение, выданные ЭМГ-устройством. Справа изображен диапазон чисел типа short, который занимает в 4 раза меньше бит. Получается, что значения на графике «растягиваются» на другую область, в следствии чего происходит сжатие.

Формулы для перевода из одного диапазона в другой:

$$\frac{x\ - min_d}{max_d - min_d} = \frac{y\ - min_s}{max_s - min_s}$$

$$y = min_{s+}\frac{(x\ - min_d)(max_s - min_s)}{max_d - min_d}$$

где, y – значение в правом столбце, а x – значение в левом столбце.

Исходя из того, что большинство используемых экранов – это Full HD с разрежением 1024 пикселя в высоту, было принято решение переводить числа именно в такой диапазон (от 0 до 1023), т.к. большая разрядная сетка будет избыточной при анализе ЭМГ-кривой на мониторе.

Для того, чтобы представить динамическое изменение оси абсцисс в файле, можно поступить следующим образом (Рис.21):

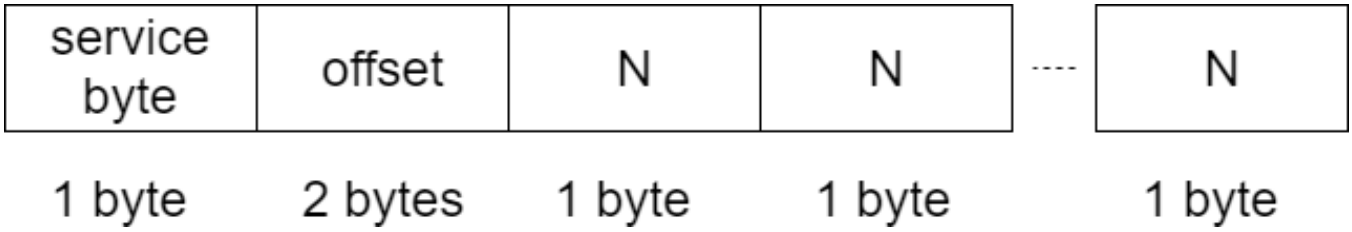


Рис. 22. Кодирование ЭМГ данных

Service byte – служебный байт, показывающий, что в этом месте ось изменяет свое направление. Далее идет offset, занимающий 2 байта и показывающий, на сколько идет смещение. После этого идут сами значения, занимающие 1 байт.

Упаковка значений в 1 байт

За счет того, что максимальное число различных значений составляет 1024, которые можно закодировать всего 10 битами, становится логичным кодировать разницу между значениями еще меньшим количеством бит. После экспериментов было выяснено, что оптимальным числом является 4 бита. Это значит, что в один байт будет помещаться сразу 2 значения.

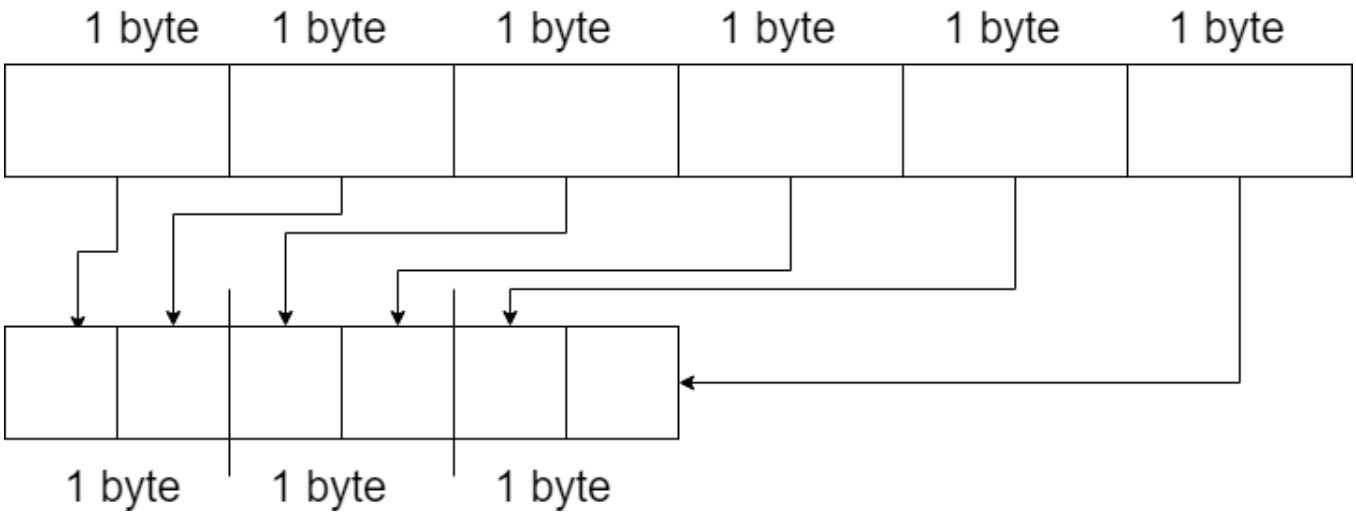


Рис. 23. Упаковка значений в один байт

Сложность заключается в том, что компьютер не может записывать в файл значения, которые меньше байта, поэтому приходиться использовать побитовые операции, а именно: побитовый сдвиг и логическое сложение. Как показано на рисунке, каждый четный байт становится первой половиной в результативных байтах, а каждый нечетный – второй.

Пример реализации данной идеи:

```
List<byte> cipher = new List<byte>();
byte package = 0;
for (int i = 0; i < plain.Count(); i++)
{
    if (i % 2 == 0)
    {
        package = Convert.ToByte(plain[i] << 4);
        if (i == plain.Count() - 1)
            cipher.Add(package);
    }
    if (i % 2 == 1)
    {
        package = Convert.ToByte(package | plain[i]);
        cipher.Add(package);
    }
```

```
}
```

Где cipher – зашифрованный кусок последовательности, plain – исходный кусок последовательности.

Если в последовательности нечетное число чисел, то последний элемент также становится первой половиной «пакета», но в конце добавляются нули. Чтобы при расшифровки не получилось больше элементов, чем в исходной последовательности, добавляется служебный байт, который показывает, четное ли число элементов в последовательности или нет. В итоге, структура файла выглядит так:

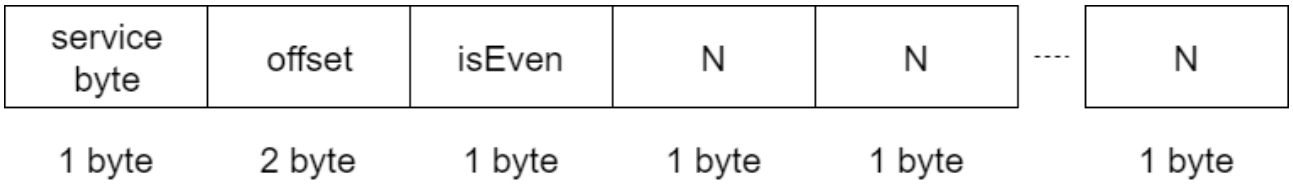


Рис. 24. Кодирование ЭМГ с учетом честности

Выборка и восстановление данных

Чтобы еще больше уменьшить объем передаваемых данных, можно делать выборку, т.е. передавать не все значения, а с пропусками. Это означает, что даже если пропускать хотя-бы одно значение, то это уже уменьшит объем передаваемых данных в 2 раза. Чтобы восстановить эти данные, будем использовать интерполяцию. Интерполяция – это способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений [9].

Кусочно-линейная интерполяция

Простейшим и часто используемым видом локальной интерполяции является линейная (или кусочно-линейная) интерполяция. Она заключается в том, что узловые точки соединяются отрезками прямых, то есть через каждые две точки (x_i, y_i) и (x_{i+1}, y_{i+1}) проводится прямая, то есть составляется полином первой степени: $F(x) = a_0 + a_1 x$, при $x_{i-1} \leq x \leq x_i$

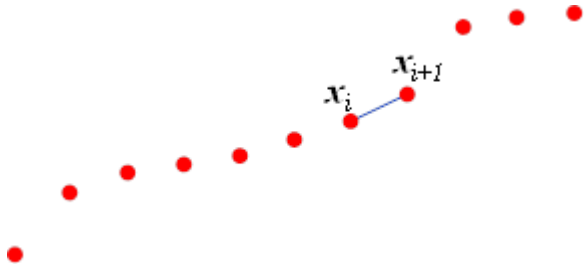


Рис. 25. Кусочно-линейная интерполяция

Коэффициенты a_0 и a_1 разные на каждом интервале $[x_i; x_{i+1}]$, и находятся из выполнения условий интерполяции на концах отрезка:

$$\begin{cases} f_{i-1} = a_0 + a_1 x_{i-1} \\ f_i = a_0 + a_1 x_1 \end{cases}$$

Из системы уравнений (3.6) можно найти коэффициенты:

$$a_0 = f(x_{i-1}) - a_1 x_{i-1}$$

$$a_1 = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

При использовании кусочно-линейной интерполяции сначала нужно определить интервал, в который попадает значение x, а затем подставить его в выражение, используя коэффициенты для данного интервала.

Кусочно-квадратичная интерполяция

В случае квадратичной интерполяции, для каждых трех узловых точек (x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1}), строится уравнение параболы: $F(x) = a_0 + a_1 x + a_1 x^2$, при $x_{i-1} \leq x \leq x_{i+1}$

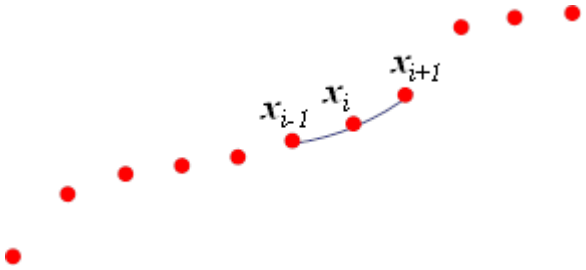


Рис. 26. Кусочно-квадратичная интерполяция

Здесь коэффициенты a_0, a_1 и a_2 разные на каждом интервале $[x_{i-1}; x_{i+1}]$, и определяются решением системы уравнений для условия прохождения параболы через три точки:

$$\begin{cases} f_{i-1} = a_0 + a_1 x_{i-1} + a_1 x_{i-1}^2 \\ f_i = a_0 + a_1 x_i + a_1 x_i^2 \\ f_{i+1} = a_0 + a_1 x_{i+1} + a_1 x_{i+1}^2 \end{cases}$$

Из системы уравнений можно найти коэффициенты:

$$a_0 = f(x_{i-1}) - a_1 x_{i-1} - a_2 x_{i-1}^2$$

$$a_1 = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} - a_2 (x_i + x_{i-1})$$

$$a_2 = \frac{f(x_{i+1}) - f(x_{i-1})}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)} - \frac{f(x_i) - f(x_{i-1})}{(x_i - x_{i-1})(x_{i+1} - x_i)}$$

Т.к. кусочно-квадратичная интерполяция находит значения точнее, будем использовать именно ее [10].

Выборка и восстановление точек может происходить несколькими способами:

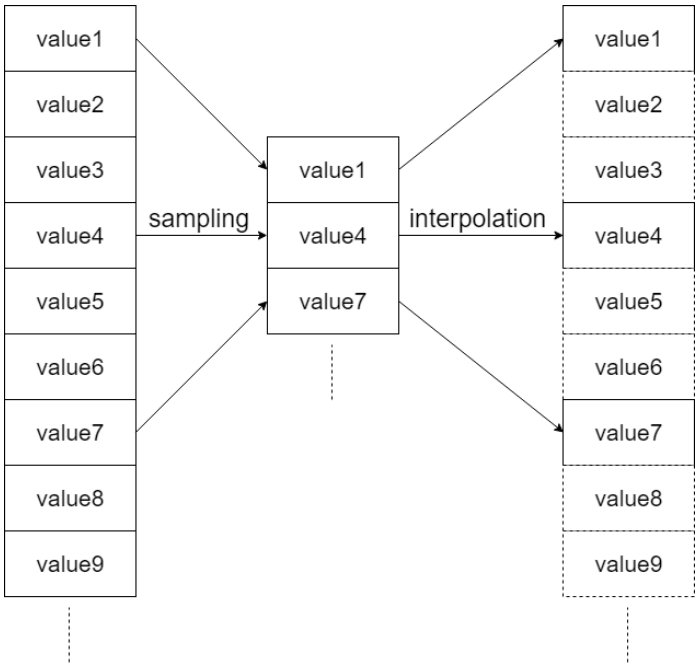


Рис. 27. Выборка и восстановление: вариант 1

В первом варианте выборка начитается с первого элемента и пропускается 2 значения.

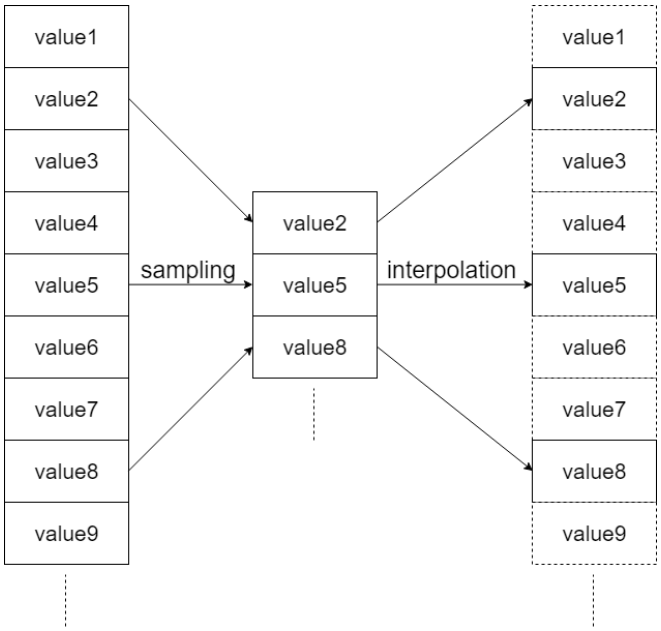


Рис. 28. Выборка и восстановление: вариант 2

Во втором варианте выборка также идет через 2 значения, но при этом идет со второго элемента. Такая вариативность дает разные результаты при погрешности, что будет учтено в следующей главе.

Зависимость размера файла от пропускаемых значений

Проведя исследования, можно изобразить график, показывающий зависимость размера сжатого файла (в килобайтах) от количества пропускаемых значений:

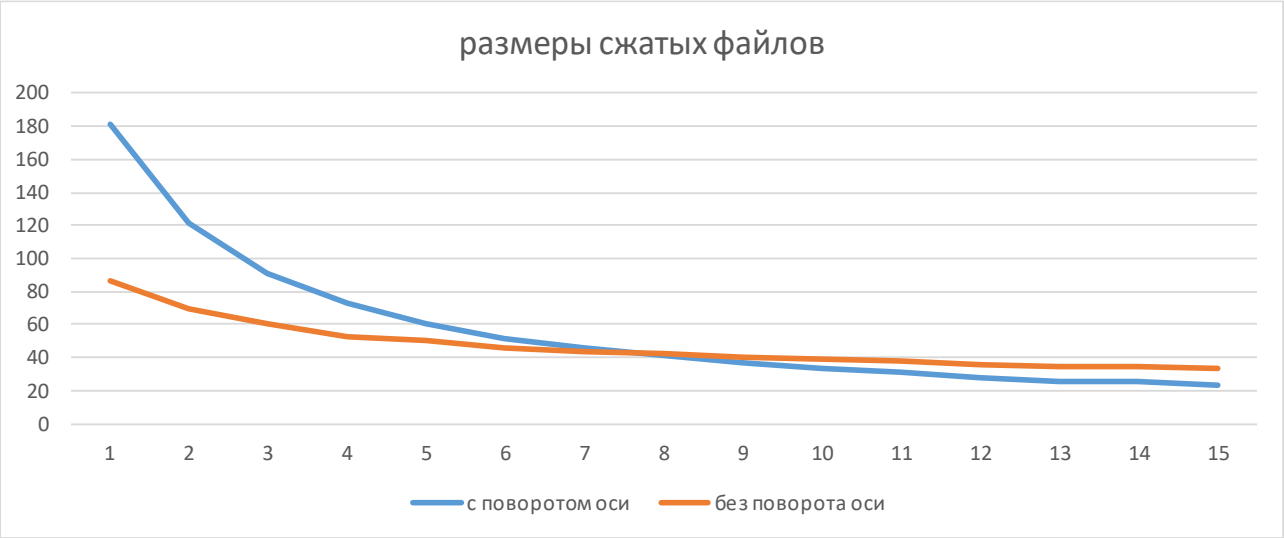


Рис. 29. Размеры сжимаемых файлов

График показывает, что при пропуске 8 значений поворот оси уже теряет свою эффективность из-за служебных байт. Следовательно, разумней будет не использовать поворот, если идет выборка более, чем 7 значений.

Выбор количества пропускаемы элементов при выборке

Чтобы выбрать количество элементов, которые нужно отбрасывать при выборке, надо посчитать общее отклонение восстанавливаемой кривой от исходной.

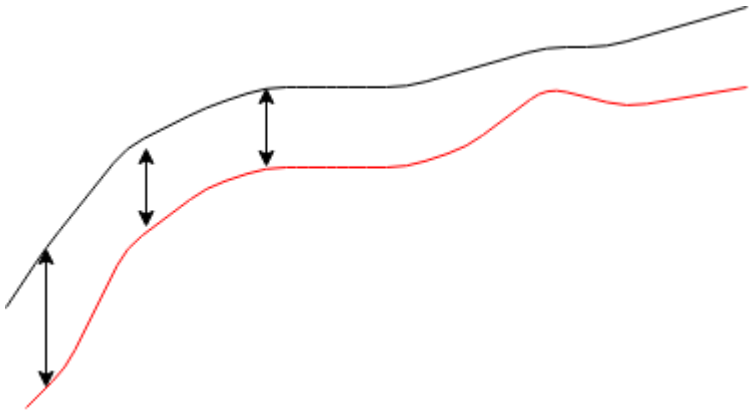


Рис. 30. Демонстрация отклонения

Кривая, расположенная сверху – это исходная кривая, а снизу – восстановленная. Между ними есть расхождение – неточность. Чтобы посчитать отклонение, надо на каждом шаге посчитать разности и потом их сложить.

Рис. 31. Погрешность без смещения

На данном изображении представлено изменение погрешности при пропуске от 1 точки в последовательности до 500. Как видно на графике, погрешность сильно колеблется. Это происходит потому, что здесь пропуск точек шел только после каждого первого элемента, т.е. как было описано в первом варианте предыдущей главы.

Чтобы устранить это колебание, надо просчитывать все варианты, когда смещение происходит от разных точек. Очевидно, что чем больше выбрасывается

значений, тем больше вариантов, откуда начинать выборку. Точнее говоря, вариантов всегда на 1 больше, чем количество отбрасываемых точек.

Рис. 32. Средняя погрешность

Из проведенного исследования видно, как изменяется средняя погрешность. Тенденция стала ясней, но до сих пор не понятно, на сколько велики значения, откладываемые слева, поэтому возникает необходимость перейти в процентное соотношение. Однако, при подсчете среднего процентного отклонения, может возникнуть ситуация, когда значительная погрешность в одной точке может быть компенсировано множеством хороших значений в других точках. В связи с этим оценка погрешности производиться на основании максимальных отклонений от исходной кривой.

Рис. 33. Максимальная погрешность в процентах

Основываясь на этом графике, были проведены эксперименты, которые показали, что при оптимальном соотношением числа пропускаемых точек,

коэффициента сжатия и искажение ЭМГ-кривой, лучше всего остановиться на 4 отбрасываемых точках.

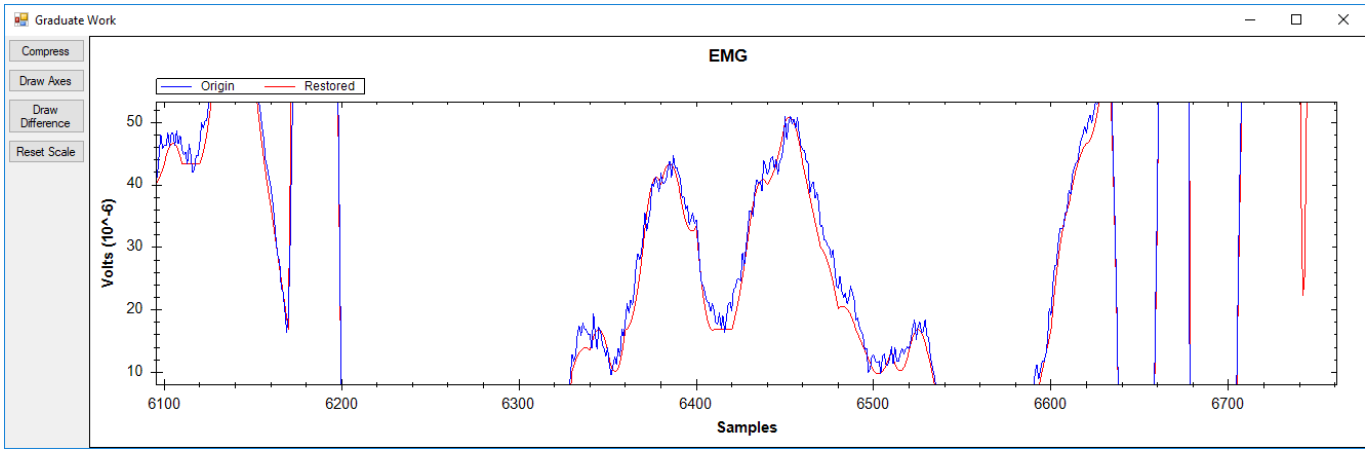


Рис. 34. Разница между исходной и восстановленной кривыми

Так выглядит различие восстановленной кривой и исходной (синий – исходная, красная - восстановленная)

Свой алгоритм сжатия

Блок-схема показывает, в каком порядке выполняется алгоритм. Слева представлена упаковка данных, а справа – распаковка.

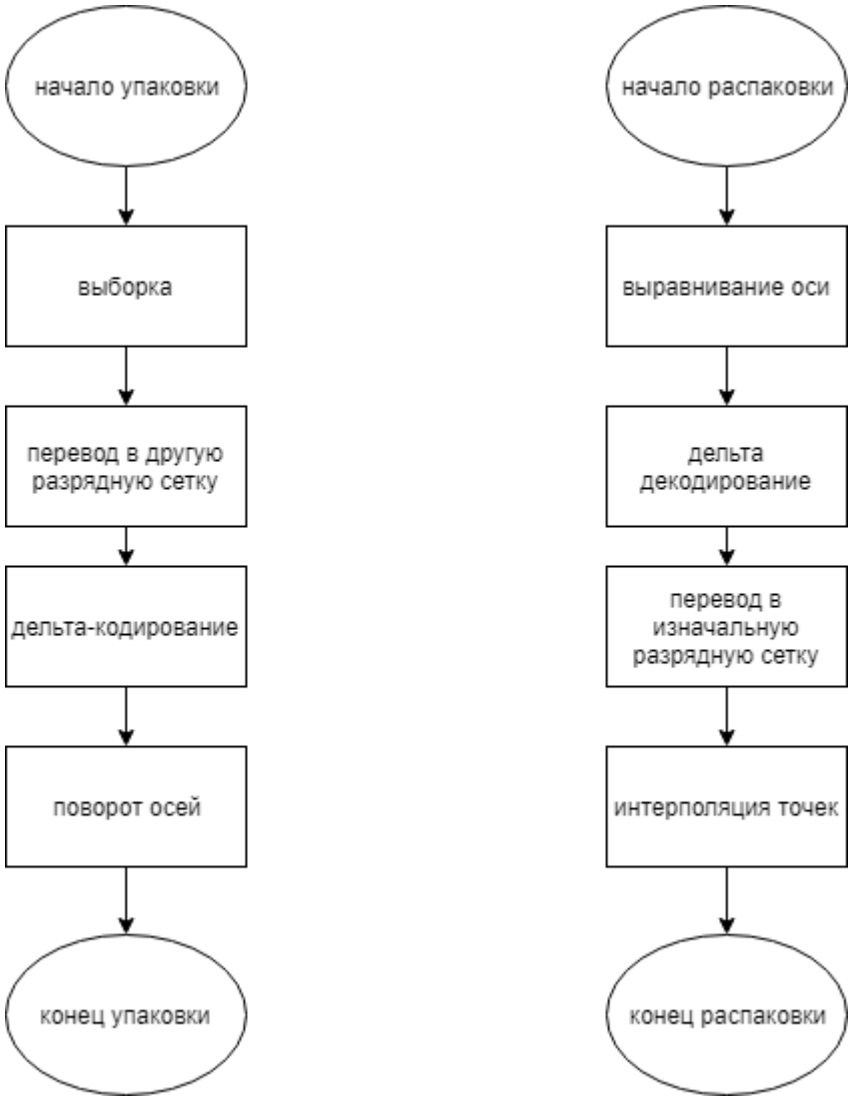


Рис. 35. Блок-схемы архивации и распаковки данных

Окончательная структура сжатого файла

В итоге, структура закодированного файла выглядит так:

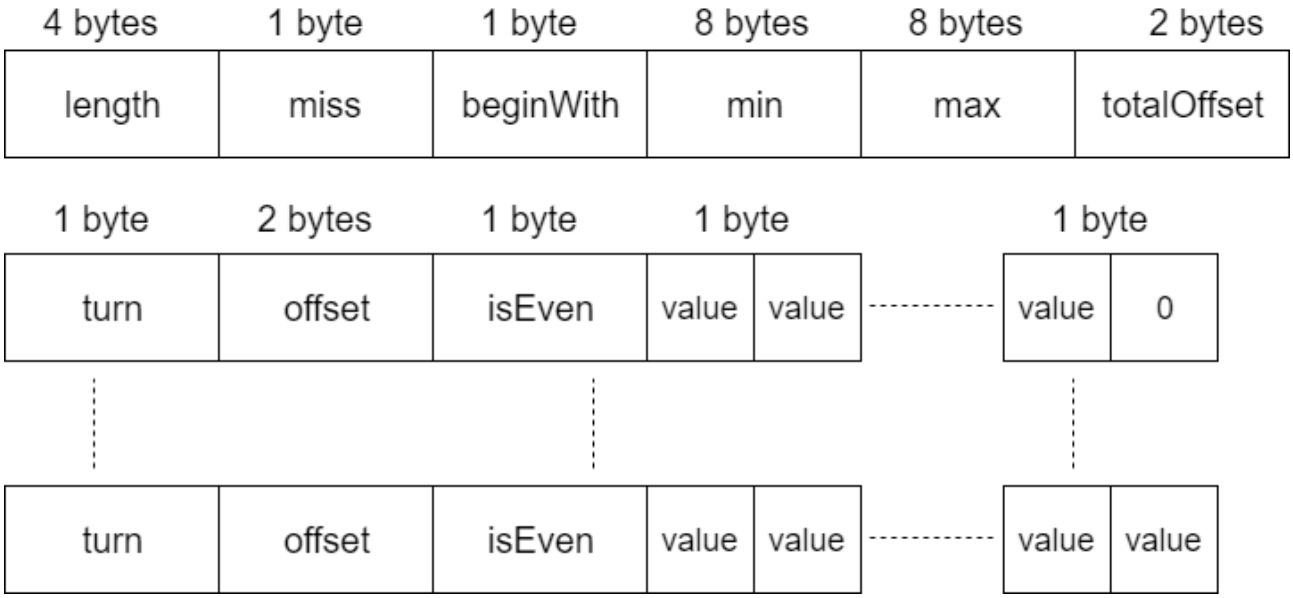


Рис. 36. Структура сжатого файла

В самом начале пишется количество значений исходного файла. Это число может быть довольно большим, поэтому под него выделяется 4 байта. Длина файла указывается для того, чтобы восстановить точно такое же количество элементов. Далее указывается количество пропускаемых значений и с какого элемента эти значения пропускать. Т.к. было выяснено, что эти числа небольшие, поэтому им хватит и одного байта. Минимальное и максимальное значение в исходной кривой нужно для того, чтобы восстанавливать значения при переходе из хранимой разрядной сетки в исходную. В конце «шапки» пишется начальное значение последовательности.

Turn – служебный байт, который свидетельствует о повороте оси, затем идет смещение, на которое ось поворачивается, и байт, указывающий четное ли количество элементов в последующем ряду чисел. Это нужно для того, чтобы понять является ли последняя часть байта значащим нулем или нет.

Результаты

После учета всех вышеперечисленных доработок можно увидеть результат программы, основанной на собственном алгоритме:

Табл. 1. Размеры закодированных файлов

	Исходный размер	WinRAR	ZIP	Разработанный алгоритм
Файл 1	1479648	1232070 (1,2)	1435051 (1,03)	54073 (27)
Файл 2	2390992	2016990 (1,18)	2313822 (1,03)	119572 (19)

На таблице 1 представлены размеры исходных файлов и их упакованных версий. Из табл.1 видно, что разработанный алгоритм эффективнее универсальных алгоритмов, реализованных в архиваторах WinRAR и ZIP от 19 до 27 раз.

Заключение

Для создания собственного алгоритма сжатия были проведены исследования существующих алгоритмов, вследствие чего было выяснено, что дельта-кодирование подходит больше всего, но у него есть несколько недостатков. Чтобы их избежать, был разработан собственный алгоритм, учитывающий резкие изменения формы сигнала с помощью динамически изменяемой оси абсцисс. Также была уменьшена разрядность сжимаемых данных и снижен объем выборки за счет прореживания количества отсчетов до приемлемого уровня. Восстановление удаленных данных производится с помощью операций интерполирования и экстраполирования.

Анализ эффективности упаковки данных разработанным алгоритмом показал, что по сравнению с универсальными алгоритмами архиваторов WinRar и ZIP, собственный метод оказался эффективней от 19 до 27 раз.

Таким образом, цель достигнута, все поставленные задачи решены.

Перспективы развития

Дальнейшее повышение эффективности алгоритма может быть связано со следующим:

1. Первое улучшение алгоритма: планируется в дальнейшем разбивать кривую на определенные участки, в которых будет использовать свое прореживание, т.е. будет выглядеть примерно так:

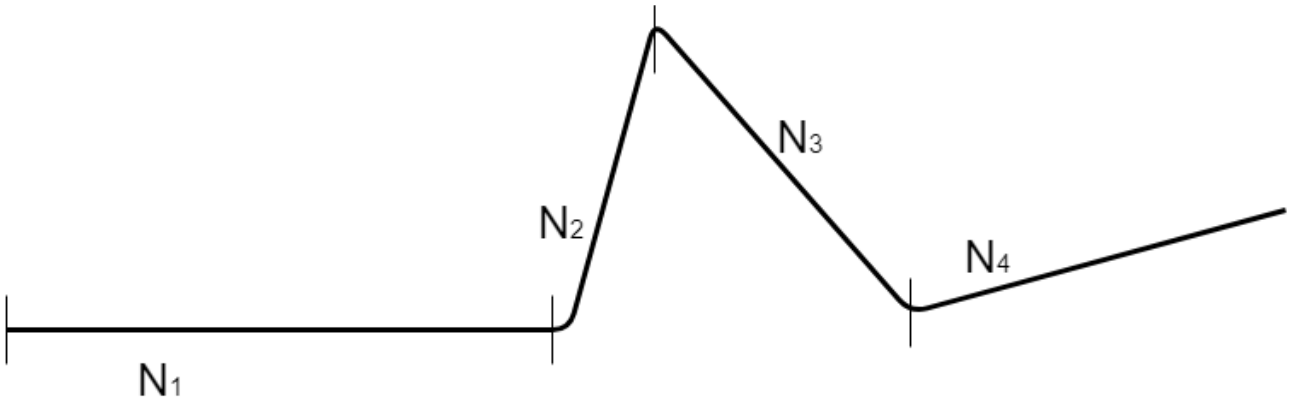


Рис. 37. Различное число выбираемых точек

Где N – число выбираемых точек.

Такой подход обеспечит меньшую потерю данных, т.к. кривая на каждом участке может вести себя непредсказуемым образом и число прореживаемых точек должно быть динамическим.

2. Второй доработкой будет являться иной способ расчета разницы между двумя соседними точками. На данный момент разница считается, согласно такой схеме:

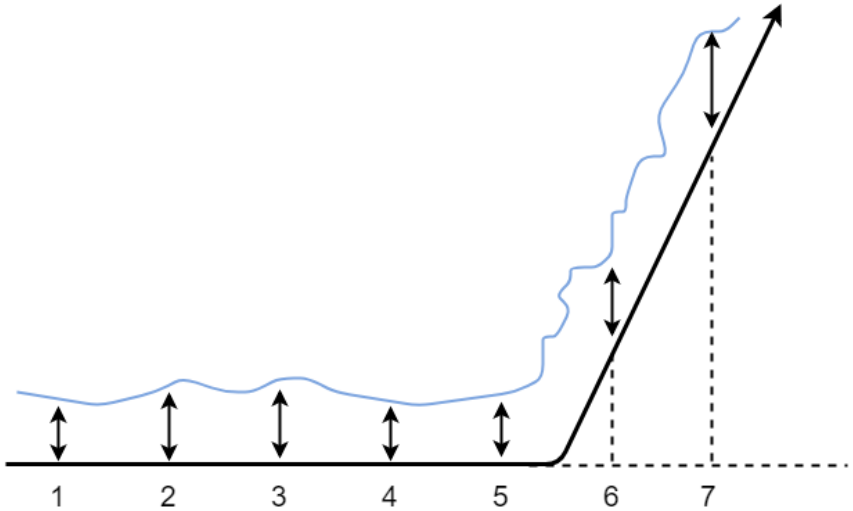


Рис. 38. Поворот оси на данный момент

Однако, даже при повороте оси дельта-кодирование может выдавать большие значения, поэтому планируется использовать такую схему:

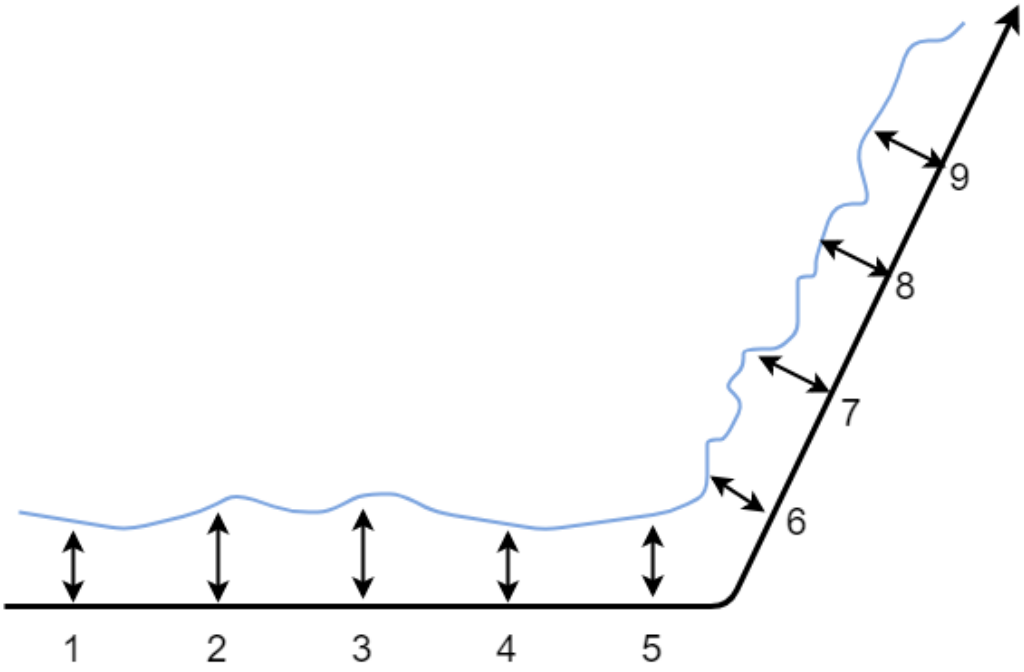


Рис. 39. Поворот оси в перспективе

С другой стороны, данный подход может сильно осложнить алгоритм, т.к. придется просчитывать синусы и косинусы, что затратит много машинного времени и, соответственно, заряд аккумулятора в ЭМГ-устройстве. Для этого потребуются дальнейшие анализы.

3. Третей перспективой является нахождение оптимального способа анализа погрешности, который выдают точный результат и не требует много операций.

Используемая литература

1. Электромиография. Свободная энциклопедия [Электронный ресурс]: общедоступная многоязычная универсальная интернет-энциклопедия со свободным контентом. - Электрон. дан. - Режим доступа: https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BC%D0%B8%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F

2. С.Л. Миронов. Расшифровка ЭКГ, 2016

3. Роберт Седжвик, Фундаментальные алгоритмы.

4. С.Г. Николаев. Практикум по клинической электромиографии, издание второе, переработанное дополнение, 2003.

5. Кодирование длин серий. Свободная энциклопедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B4%D0%BB%D0%B8%D0%BD_%D1%81%D0%B5%D1%80%D0%B8%D0%B9

6. Алгоритм Лемпеля — Зива — Велча. Свободная энциклопедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0

7. Дельта-кодирование. Свободная энциклопедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D1%8C%D1%82%D0%B0-%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

8. Методы сжатия данных. «Время электроники»: общедоступный новостной сайт. [Электронный ресурс]. Режим доступа: http://www.russianelectronics.ru/leader-r/review/8602/doc/46598/

9. Интерполяция. Свободная энциклопедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F

10. Университет ИТМО, кафедра прикладной и компьютерной оптики. Интерполяция. Режим доступа: http://aco.ifmo.ru/el_books/numerical_methods/lectures/glava3.html

Приложение 1

Класс Compression, сжимающий кривую.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.IO;

namespace Compress
{
    class Compression
    {
        static public void TxtToBinDouble(string pathIn, string pathOut)
        {
            StreamReader stmRdr = new StreamReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
            BinaryWriter binWtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);

            string line;
            while ((line = stmRdr.ReadLine()) != null)
            {
                double number = Convert.ToDouble(line);
                binWtr.Write(number);
            }

            stmRdr.Close();
            binWtr.Close();
        }

        static public void DoubleToShort(string pathIn, string pathOut,
            short newMin = short.MinValue, short newMax = short.MaxValue)
        {
            BinaryReader binRdr = new BinaryReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
            BinaryWriter binWrtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);
            //pathOut = pathOut.Replace(".bin", ".txt");
            //StreamWriter strmWrtr = new StreamWriter(pathOut);

            int length = binRdr.ReadInt32();
            byte miss = binRdr.ReadByte();
            byte beginWith = binRdr.ReadByte();
            binWrtr.Write(length);
            binWrtr.Write(miss);
            binWrtr.Write(beginWith);
            //strmWrtr.Write(Convert.ToString(length) + "\r\n");
            //strmWrtr.Write(Convert.ToString(miss) + "\r\n");
            //strmWrtr.Write(Convert.ToString(beginWith) + "\r\n");

            double maxValue = Double.MinValue, minValue = Double.MaxValue;
            //search for max and min
            do
            {
                double value = binRdr.ReadDouble();
                if (value > maxValue)
                    maxValue = value;
                if (value < minValue)
                    minValue = value;
            } while (binRdr.PeekChar() > -1);
```

```
            binWrtr.Write(minValue);
            binWrtr.Write(maxValue);
            //strmWrtr.Write(Convert.ToString(minValue) + "\r\n");
            //strmWrtr.Write(Convert.ToString(maxValue) + "\r\n");

            //into int16
            binRdr.BaseStream.Position = 0;
            binRdr.ReadInt32();
            binRdr.ReadByte();
            binRdr.ReadByte();
            do
            {
                double value = binRdr.ReadDouble();
                short x = (short)(newMin + (value - minValue) * (newMax -
newMin) / (maxValue - minValue));
                binWrtr.Write(x);
                //strmWrtr.Write(Convert.ToString(x) + "\r\n");
            } while (binRdr.PeekChar() > -1);

            binRdr.Close();
            binWrtr.Close();
            //strmWrtr.Close();
        }

        static public void ShortToDouble(string pathIn, string pathOut)
        {
            BinaryReader binRdr = new BinaryReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
            BinaryWriter binWrtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);
            //pathOut = pathOut.Replace(".bin", ".txt");
            //StreamWriter strmWrtr = new StreamWriter(pathOut);

            double doubleMin = double.MaxValue, doubleMax = double.MinValue;

            int length = binRdr.ReadInt32();
            byte miss = binRdr.ReadByte();
            byte beginWith = binRdr.ReadByte();

            doubleMin = binRdr.ReadDouble();
            doubleMax = binRdr.ReadDouble();

            binWrtr.Write(length);
            binWrtr.Write(miss);
            binWrtr.Write(beginWith);
            //strmWrtr.Write(Convert.ToString(length) + "\r\n");
            //strmWrtr.Write(Convert.ToString(miss) + "\r\n");
            //strmWrtr.Write(Convert.ToString(beginWith) + "\r\n");

            short shortMin = short.MaxValue, shortMax = short.MinValue;
            do
            {
                short cur = binRdr.ReadInt16();
                if (cur < shortMin)
                    shortMin = cur;
                if (cur > shortMax)
                    shortMax = cur;
            } while(binRdr.PeekChar() > -1);
            binRdr.BaseStream.Position = 0;
            binRdr.ReadInt32();
            binRdr.ReadByte();
            binRdr.ReadByte();
            binRdr.ReadDouble();
```

```
            binRdr.ReadDouble();

            do
            {
                short cur = binRdr.ReadInt16();
                double x = doubleMin + (cur - shortMin) * (doubleMax -
doubleMin)
                    / (shortMax - shortMin);
                binWrtr.Write(x);
                //strmWrtr.Write(Convert.ToString(x) + "\r\n");
            } while(binRdr.PeekChar() > -1);

            binRdr.Close();
            binWrtr.Close();
            //strmWrtr.Close();
        }

        static public void DelShort(string pathIn, string pathOut)
        {
            BinaryReader binRdr = new BinaryReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
            BinaryWriter binWrtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);
            //pathOut = pathOut.Replace(".bin", ".txt");
            //StreamWriter strmWrtr = new StreamWriter(pathOut);

            int length = binRdr.ReadInt32(); //reand length
            byte miss = binRdr.ReadByte();
            byte beginWith = binRdr.ReadByte();
            double min = binRdr.ReadDouble(); //read min
            double max = binRdr.ReadDouble(); //read max
            binWrtr.Write(length);
            binWrtr.Write(miss);
            binWrtr.Write(beginWith);
            binWrtr.Write(min);
            binWrtr.Write(max);
            //strmWrtr.Write(Convert.ToString(length) + "\r\n");
            //strmWrtr.Write(Convert.ToString(miss) + "\r\n");
            //strmWrtr.Write(Convert.ToString(beginWith) + "\r\n");
            //strmWrtr.Write(Convert.ToString(min) + "\r\n");
            //strmWrtr.Write(Convert.ToString(max) + "\r\n");

            short last;
            last = binRdr.ReadInt16();
            binWrtr.Write(last);
            //strmWrtr.Write(Convert.ToString(last) + "\r\n");
            do
            {
                short cur = binRdr.ReadInt16();
                short del = (short)(cur - last);
                binWrtr.Write(del);
                //strmWrtr.Write(Convert.ToString(del) + "\r\n");
                last = cur;
            } while (binRdr.PeekChar() > -1);

            binRdr.Close();
            binWrtr.Close();
            //strmWrtr.Close();
        }

        static public void DelSmartOffset(string pathIn, string pathOut, byte
bits = 4)
        {
```

```
            BinaryReader binRdr = new BinaryReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
            BinaryWriter binWrtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);
            //pathOut = pathOut.Replace(".bin", ".txt");
            //StreamWriter strmWrtr = new StreamWriter(pathOut);

            int length = binRdr.ReadInt32();
            byte miss = binRdr.ReadByte();
            byte beginWith = binRdr.ReadByte();
            double min = binRdr.ReadDouble();
            double max = binRdr.ReadDouble();
            binWrtr.Write(length);
            binWrtr.Write(miss);
            binWrtr.Write(beginWith);
            binWrtr.Write(min);
            binWrtr.Write(max);
            //strmWrtr.Write(Convert.ToString(length) + "\r\n");
            //strmWrtr.Write(Convert.ToString(miss) + "\r\n");
            //strmWrtr.Write(Convert.ToString(beginWith) + "\r\n");
            //strmWrtr.Write(Convert.ToString(min) + "\r\n");
            //strmWrtr.Write(Convert.ToString(max) + "\r\n");

            byte serviceByte = (byte)(Math.Pow(2, bits) - 1);
            byte minDel = Byte.MinValue;
            byte maxDel = (byte)(serviceByte - 1);
            short maxNumbers = (short)(maxDel - minDel);

            List<short> sequence = new List<short>();
            short last = binRdr.ReadInt16();
            binWrtr.Write(last);
            //strmWrtr.Write(Convert.ToString(last) + "\r\n");
            int turns = 0;
            int totalOffset = last; //overall offset
            short offset = 0; //current offset
            bool frst = true;
            bool isEndOfFile = true;

            do
            {
                sequence.Clear(); //sequence of points
                short first;
                if (frst) //if it's the first passing
                    first = binRdr.ReadInt16();
                else
                    first = last;
                sequence.Add(first);

                //calculating offsets
                short offsetMin = Convert.ToInt16(first - maxNumbers -
totalOffset);
                short offsetMax = Convert.ToInt16(first - totalOffset);

                isEndOfFile = true;

                while (binRdr.PeekChar() > -1)
                {
                    short cur = binRdr.ReadInt16(); //current point
                    sequence.Add(cur);
                    //above all axes
                    if (cur >= totalOffset + offsetMax * sequence.Count() &&
                        cur <= totalOffset + offsetMax * sequence.Count() +
maxNumbers)
```

```
                    {
                        double tmp = Convert.ToDouble(cur - maxNumbers -
totalOffset) / sequence.Count();
                        short newMin = Convert.ToInt16(Math.Ceiling(tmp));
                        if (newMin > offsetMin)
                            offsetMin = newMin;
                    }
                    //between axes
                    if (cur <= totalOffset + offsetMax * sequence.Count() &&
                        cur >= totalOffset + offsetMin * sequence.Count())
                    {
                        double tmp = Convert.ToDouble(cur - totalOffset) /
sequence.Count();
                        short newMax = Convert.ToInt16(Math.Floor(tmp));
                        if (newMax < offsetMax)
                            offsetMax = newMax;
                        tmp = Convert.ToDouble(cur - maxNumbers - totalOffset) /
sequence.Count();
                        short newMin = Convert.ToInt16(Math.Ceiling(tmp));
                        if (newMin > offsetMin)
                            offsetMin = newMin;
                    }
                    //out of range
                    if (cur > totalOffset + offsetMax * sequence.Count() +
maxNumbers ||
                        cur < totalOffset + offsetMin * sequence.Count() ||
offsetMax < offsetMin)
                    {
                        last = cur;
                        frst = false;
                        isEndOfFile = false;
                        break;
                    }
                }
                offset = offsetMax; //choosing an efficien offset
                binWrtr.Write(serviceByte);
                binWrtr.Write(offset);
                turns++;
                //strmWrtr.Write(Convert.ToString(serviceByte) + " " +
Convert.ToString(offset));

                List<byte> plain = new List<byte>();
                int n;
                if (isEndOfFile)
                    n = sequence.Count();
                else
                    n = sequence.Count() - 1;
                for (int i = 0; i < n; i++)
                {
                    totalOffset += offset;
                    plain.Add(Convert.ToByte(sequence[i] - totalOffset));
                }

                binWrtr.Write(Convert.ToByte(plain.Count % 2));
                //strmWrtr.Write(" " + Convert.ToString(plain.Count % 2) +
"\r\n");

                //bits shift
                List<byte> cipher = new List<byte>();
                byte package = 0;
                for (int i = 0; i < plain.Count(); i++)
                {
                    if (i % 2 == 0)
```

```
                {
                    package = Convert.ToByte(plain[i] << 4);
                    if (i == plain.Count() - 1)
                        cipher.Add(package);
                }
                if (i % 2 == 1)
                {
                    package = Convert.ToByte(package | plain[i]);
                    cipher.Add(package);
                }
            }

            for (int i = 0; i < cipher.Count(); i++)
            {
                binWrtr.Write(cipher[i]);
                //strmWrtr.Write(Convert.ToString(cipher[i]) + "\r\n");
            }
        } while (binRdr.PeekChar() > -1);
        //strmWrtr.Write("turns: " + Convert.ToString(turns) + "\r\n");

        binRdr.Close();
        binWrtr.Close();
        //strmWrtr.Close();
    }

    static public void DecDelSmartOffset(string pathIn, string pathOut, byte
bits = 4)
    {
        BinaryReader binRdr = new BinaryReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
        BinaryWriter binWrtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);
        //pathOut = pathOut.Replace(".bin", ".txt");
        //StreamWriter strmWrtr = new StreamWriter(pathOut);

        int length = binRdr.ReadInt32();
        byte miss = binRdr.ReadByte();
        byte beginWith = binRdr.ReadByte();
        double min = binRdr.ReadDouble();
        double max = binRdr.ReadDouble();
        binWrtr.Write(length);
        binWrtr.Write(miss);
        binWrtr.Write(beginWith);
        binWrtr.Write(min);
        binWrtr.Write(max);
        //strmWrtr.Write(Convert.ToString(length) + "\r\n");
        //strmWrtr.Write(Convert.ToString(miss) + "\r\n");
        //strmWrtr.Write(Convert.ToString(beginWith) + "\r\n");
        //strmWrtr.Write(Convert.ToString(min) + "\r\n");
        //strmWrtr.Write(Convert.ToString(max) + "\r\n");

        byte serviceByte = (byte)(Math.Pow(2, bits) - 1);
        short offset = 0;
        byte isEven = 0;
        int totalOffset = binRdr.ReadInt16();
        binWrtr.Write((short)totalOffset);
        //strmWrtr.Write(Convert.ToString(totalOffset) + "\r\n");

        List<byte> sequence = new List<byte>();
        binRdr.ReadByte();
        do
        {
            offset = binRdr.ReadInt16();
```

```
                isEven = binRdr.ReadByte();
                while(binRdr.PeekChar() > -1)
                {
                    byte cur = binRdr.ReadByte();
                    if (cur == serviceByte)
                        break;
                    sequence.Add(Convert.ToByte(cur >> 4));
                    sequence.Add(Convert.ToByte(cur & 15));
                }
                int n;
                if (isEven == 0)
                    n = sequence.Count();
                else
                    n = sequence.Count() - 1;
                for (int i = 0; i < n; i++)
                {
                    totalOffset += offset;
                    short res = Convert.ToInt16(totalOffset + sequence[i]);
                    binWrtr.Write(res);
                    //strmWrtr.Write(Convert.ToString(res + "\r\n"));
                }
                sequence.Clear();
            } while(binRdr.PeekChar() > -1);

            binRdr.Close();
            binWrtr.Close();
            //strmWrtr.Close();
        }

        static public void DecDelShort(string pathIn, string pathOut)
        {
            BinaryReader binRdr = new BinaryReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
            BinaryWriter binWrtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);
            //pathOut = pathOut.Replace(".bin", ".txt");
            //StreamWriter strmWrtr = new StreamWriter(pathOut);

            int length = binRdr.ReadInt32();
            byte miss = binRdr.ReadByte();
            byte beginWith = binRdr.ReadByte();
            double min = binRdr.ReadDouble();
            double max = binRdr.ReadDouble();
            binWrtr.Write(length);
            binWrtr.Write(miss);
            binWrtr.Write(beginWith);
            binWrtr.Write(min);
            binWrtr.Write(max);
            //strmWrtr.Write(Convert.ToString(length) + "\r\n");
            //strmWrtr.Write(Convert.ToString(miss) + "\r\n");
            //strmWrtr.Write(Convert.ToString(beginWith) + "\r\n");
            //strmWrtr.Write(Convert.ToString(min) + "\r\n");
            //strmWrtr.Write(Convert.ToString(max) + "\r\n");

            short last = binRdr.ReadInt16();
            binWrtr.Write(last);
            //strmWrtr.Write(Convert.ToString(last) + "\r\n");
            do
            {
                short cur = binRdr.ReadInt16();
                short res = (short)(last + cur);
                binWrtr.Write(res);
                //strmWrtr.Write(Convert.ToString(res) + "\r\n");
```

```
                last = res;
            } while (binRdr.PeekChar() > -1);
            binRdr.Close();
            binWrtr.Close();
            //strmWrtr.Close();
        }

        static public void Sampling(string pathIn, string pathOut, byte miss,
byte beginWith = 0)
        {
            BinaryReader binRdr = new BinaryReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
            BinaryWriter binWrtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);
            //pathOut = pathOut.Replace(".bin", ".txt");
            //StreamWriter strmWrtr = new StreamWriter(pathOut);

            int length = 0;
            do
            {
                binRdr.ReadDouble();
                length++;
            } while (binRdr.PeekChar() > -1);
            binWrtr.Write(length);
            binWrtr.Write(miss);
            binWrtr.Write(beginWith);
            //strmWrtr.Write(Convert.ToString(length) + "\r\n");
            //strmWrtr.Write(Convert.ToString(miss) + "\r\n");
            //strmWrtr.Write(Convert.ToString(beginWith) + "\r\n");
            binRdr.BaseStream.Position = 0;

            for (int i = 0; i < beginWith; i++)
                binRdr.ReadDouble();

            do
            {
                double number = binRdr.ReadDouble();
                binWrtr.Write(number);
                //strmWrtr.Write(Convert.ToString(number) + "\r\n");
                for (int i = 0; i < miss; i++)
                {
                    if (binRdr.PeekChar() > -1)
                        number = binRdr.ReadDouble();
                    else
                        break;
                }
            } while (binRdr.PeekChar() > -1);

            binRdr.Close();
            binWrtr.Close();
            //strmWrtr.Close();
        }

        static public void Interpolation(string pathIn, string pathOut)
        {
            BinaryReader binRdr = new BinaryReader(File.Open(pathIn,
FileMode.Open), Encoding.ASCII);
            BinaryWriter binWrtr = new BinaryWriter(File.Open(pathOut,
FileMode.Create), Encoding.ASCII);
            pathOut = pathOut.Replace(".bin", ".txt");
            StreamWriter strmWrtr = new StreamWriter(pathOut);

            int originLength = binRdr.ReadInt32();
```

```
            int restoredLength = 0;
            byte missed = binRdr.ReadByte();
            byte beginWith = binRdr.ReadByte();
            int firstX = beginWith, middleX = 0, lastX = 0;
            double firstY = 0, middleY = 0, lastY = 0;
            firstY = binRdr.ReadDouble();
            bool firstPass = true;
            double a1, a2, a0;
            do
            {
                int beginning;
                double testMiddle = binRdr.ReadDouble();
                if (binRdr.PeekChar() > -1)
                {
                    middleY = testMiddle;
                    middleX = firstX + missed + 1;
                    lastY = binRdr.ReadDouble();
                    lastX = middleX + missed + 1;
                    beginning = firstX + 1;
                }
                else
                {
                    firstX = middleX;
                    firstY = middleY;
                    middleX = lastX;
                    middleY = lastY;
                    lastY = testMiddle;
                    lastX = middleX + missed + 1;
                    beginning = middleX + 1;
                }
                a2 = (lastY - firstY) / ((lastX - firstX) * (lastX - middleX)) -
                    (middleY - firstY) / ((middleX - firstX) * (lastX -
middleX));
                a1 = (middleY - firstY) / (middleX - firstX) - a2 * (middleX +
firstX);
                a0 = firstY - a1 * firstX - a2 * firstX * firstX;
                if (firstPass)
                {
                    for (int i = 0; i <= lastX; i++)
                    {
                        double x = a0 + a1 * i + a2 * i * i;
                        binWrtr.Write(x);
                        restoredLength++;
                        strmWrtr.Write(Convert.ToString(x) + "\r\n");
                    }
                    firstPass = false;
                }
                else
                {
                    for (int i = beginning; i <= lastX; i++)
                    {
                        double x = a0 + a1 * i + a2 * i * i;
                        binWrtr.Write(x);
                        restoredLength++;
                        strmWrtr.Write(Convert.ToString(x) + "\r\n");
                    }
                }
                firstX = lastX;
                firstY = lastY;
            } while(binRdr.PeekChar() > -1);

            for (int i = restoredLength; i < originLength; i++)
            {
```

```
                double x = a0 + a1 * i + a2 * i * i;
                binWrtr.Write(x);
                restoredLength++;
                strmWrtr.Write(Convert.ToString(x) + "\r\n");
            }
            binRdr.Close();
            binWrtr.Close();
            strmWrtr.Close();
        }
        static public double CalcDiff(string pathOrigin, string pathRestored)
        {
            BinaryReader binOrigin = new BinaryReader(File.Open(pathOrigin,
FileMode.Open), Encoding.ASCII);
            BinaryReader binRestored = new BinaryReader(File.Open(pathRestored,
FileMode.Open), Encoding.ASCII);
            double result = 0;
            do
            {
                double curRestored = binRestored.ReadDouble();
                double curOrigin = binOrigin.ReadDouble();
                result += Math.Abs(curRestored - curOrigin);
            } while (binRestored.PeekChar() > -1);
            binOrigin.Close();
            binRestored.Close();
            return result;
        }
        static public double CalcPercentDiff(string pathOrigin, string
pathRestored)
        {
            BinaryReader binOrigin = new BinaryReader(File.Open(pathOrigin,
FileMode.Open), Encoding.ASCII);
            BinaryReader binRestored = new BinaryReader(File.Open(pathRestored,
FileMode.Open), Encoding.ASCII);

            double max, min;
            max = min = binOrigin.ReadDouble();
            do
            {
                double cur = binOrigin.ReadDouble();
                if (cur < min)
                    min = cur;
                if (cur > max)
                    max = cur;
            } while (binOrigin.PeekChar() > -1);

            binOrigin.BaseStream.Position = 0;
            double diff = max - min;
            double maxPercent = 0;
            do
            {
                double curRestored = binRestored.ReadDouble();
                double curOrigin = binOrigin.ReadDouble();
                double percent = Math.Abs(curRestored - curOrigin) / diff;
                if (percent > maxPercent)
                    maxPercent = percent;
            } while (binRestored.PeekChar() > -1);
            binOrigin.Close();
            binRestored.Close();
            return maxPercent;
        }
    }
}
```

Приложение 2

Презентация

Разработка последовательного и параллельного алгоритмов сжатия ЭМГ-данных

Выполнил: Васильев Д.М.

Руководитель: С.Г. Сидоров, к.т.н., доц.

Цель работы

Разработка энергоэффективного алгоритма сжатия ЭМГ данных

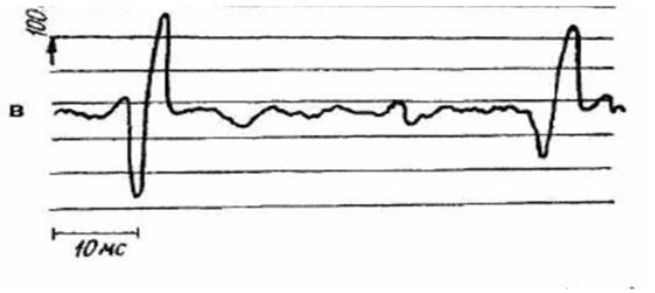
1

Электромиография (ЭМГ)

Электромиография – это метод исследования биоэлектрических потенциалов, возникающих в скелетных мышцах человека и животных при возбуждении мышечных волокон

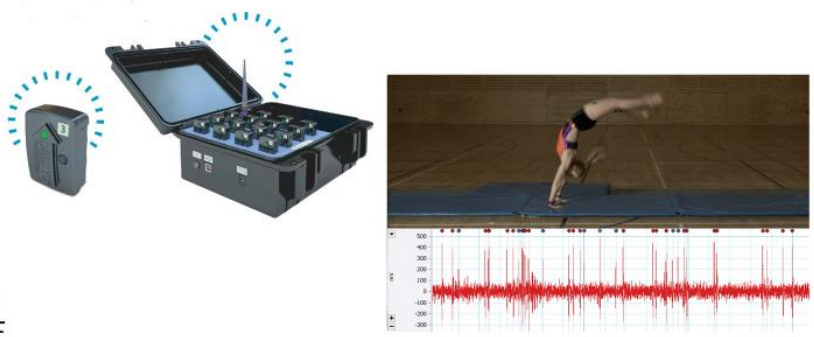
2

Форма ЭМГ-сигнала



3

Мобильная электромиография



4

Постановка задач

- Анализ существующих алгоритмов сжатия
- Разработка оптимизированного алгоритма сжатия данных
- Сравнение эффективности разработанного алгоритма с существующими алгоритмами

5

Выбор опорного алгоритма

Были рассмотрены алгоритмы:

- RLE – рассчитан на повторяющиеся последовательности данных, например, изображения
- LZW – кодирование с помощью словарей, рассчитан на кодирование текста.
- Деревья Шеннона-Фано – кодирование более частых символов меньшим количеством бит
- DCT – алгоритм с потерями данных, который широко распространен в кодировании файлов с расширением jpeg. Требует много машинного времени
- Дельта-кодирование – используется, где данные изменяются плавно

6

Дельта-кодирование

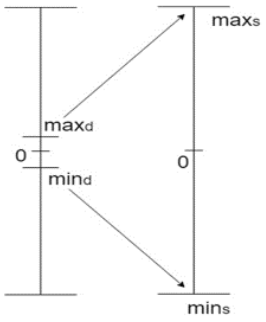
исходная последовательность:	17 19 24 24 24 21 15 10 89 95 96 96 95 ...
данные после Дельта-кодирования:	17 2 5 0 0 -3 -6 -5 79 6 1 0 0 -1 ...

7

ЭМГ-кривая после дельта-кодирования

8

Перевод из double (8 байт) в short (2 байта)



$$\frac{x - min_d}{max_d - min_d} = \frac{y - min_s}{max_s - min_s}$$

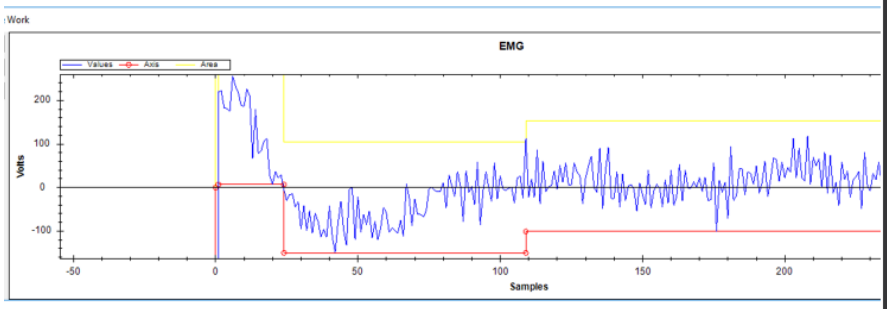
$$y = min_s + \frac{(x - min_d)(max_s - min_s)}{max_d - min_d}$$

9

Плавное и неплавное изменение

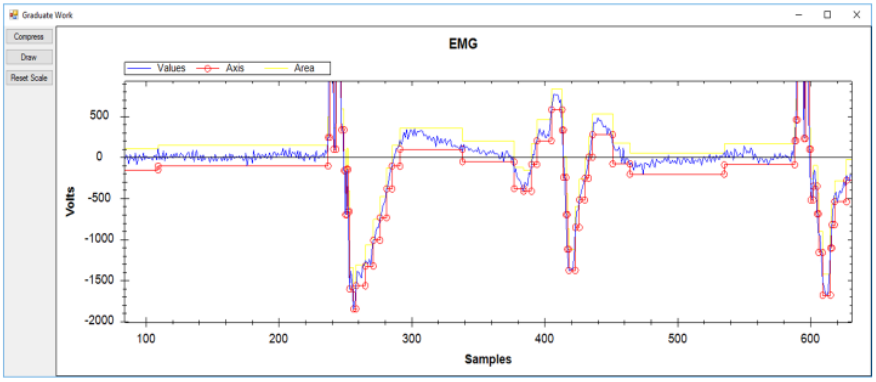
10

смещение оси



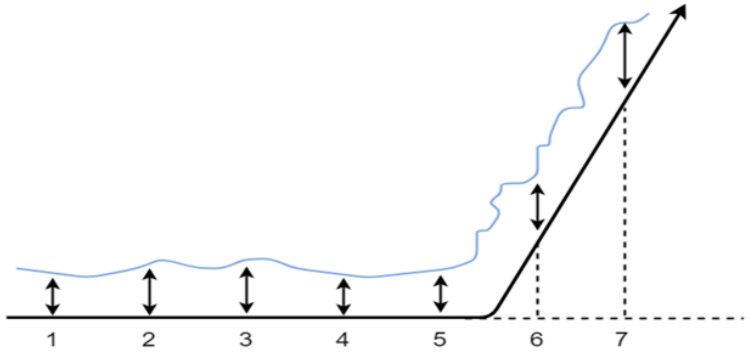
11

недостаток смещение оси



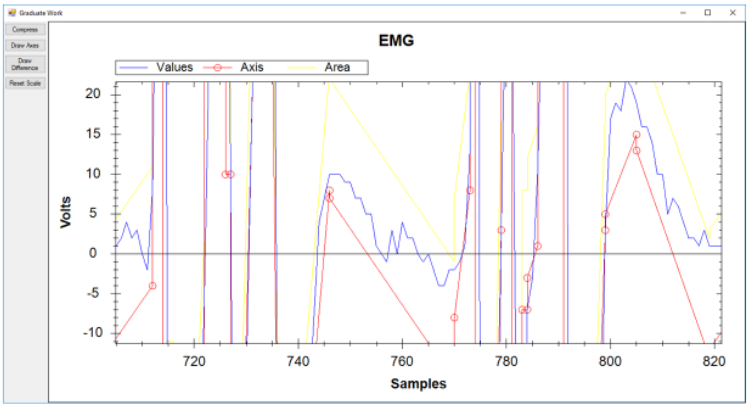
12

Поворот оси абсцисс



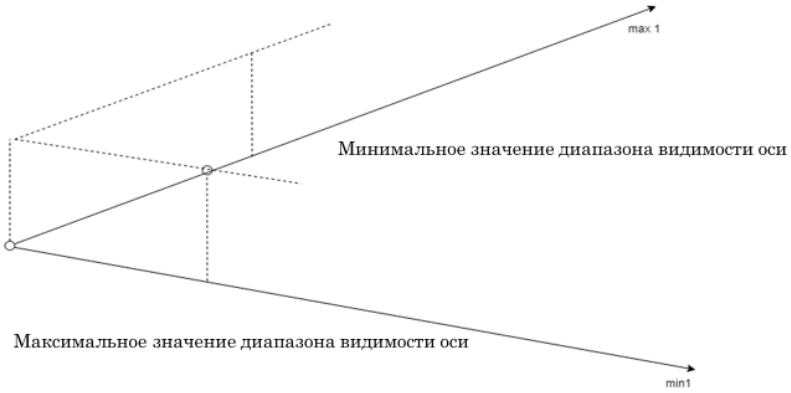
13

Демонстрация поворота оси



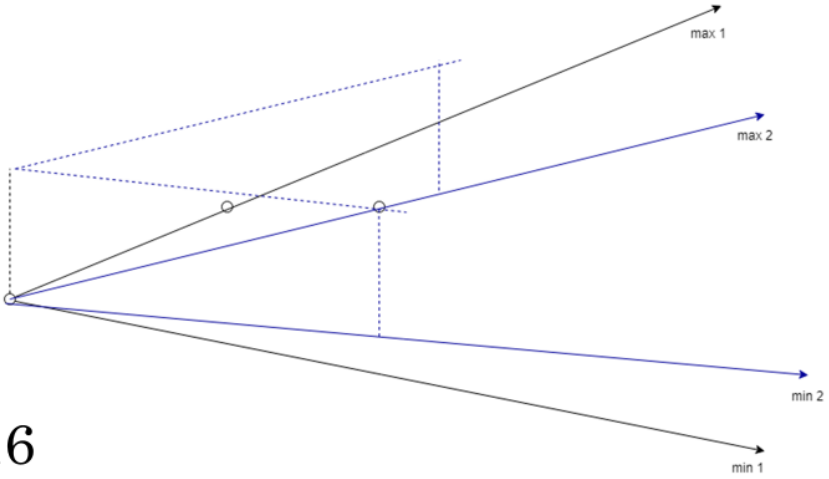
14

Выбор угла поворота оси: 2 точки



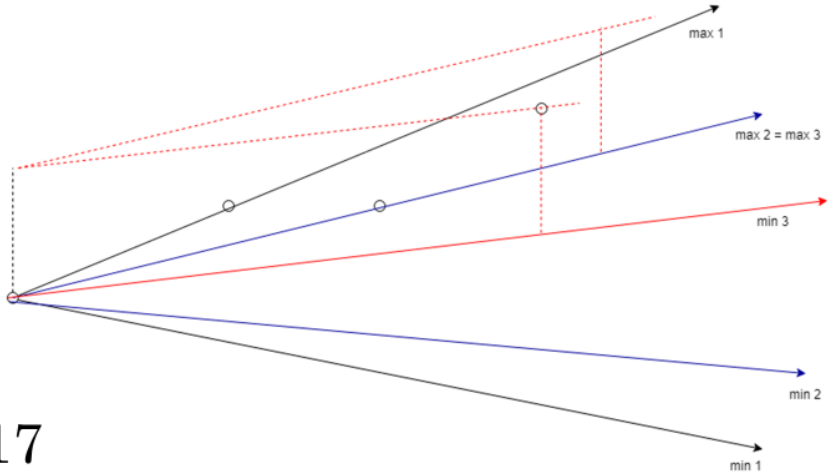
15

Выбор угла поворота оси: 3 точки



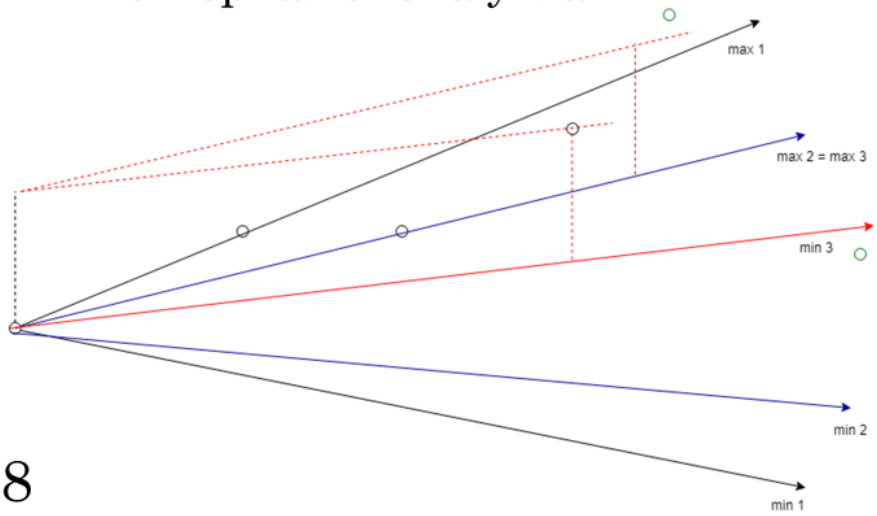
16

Выбор угла поворота оси: 4 точки



17

Повторная смена угла

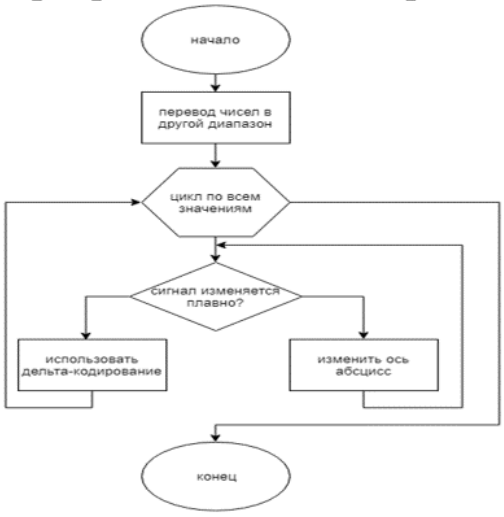


18

Блок-схема выбора угла поворота оси

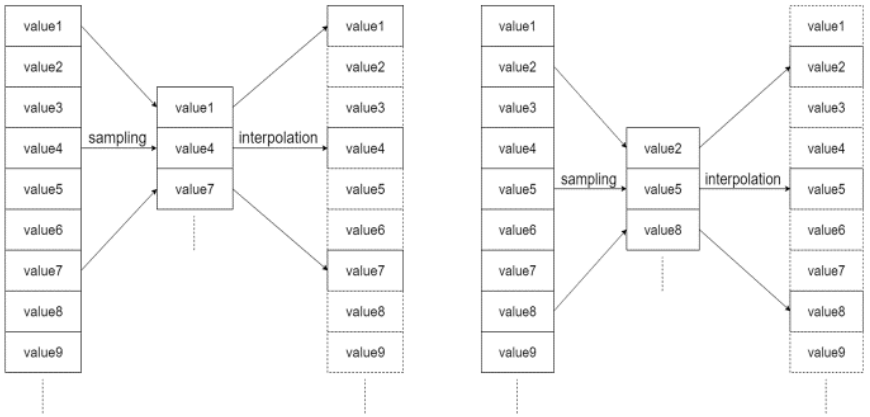
19

Блок-схема разработанного алгоритма сжатия



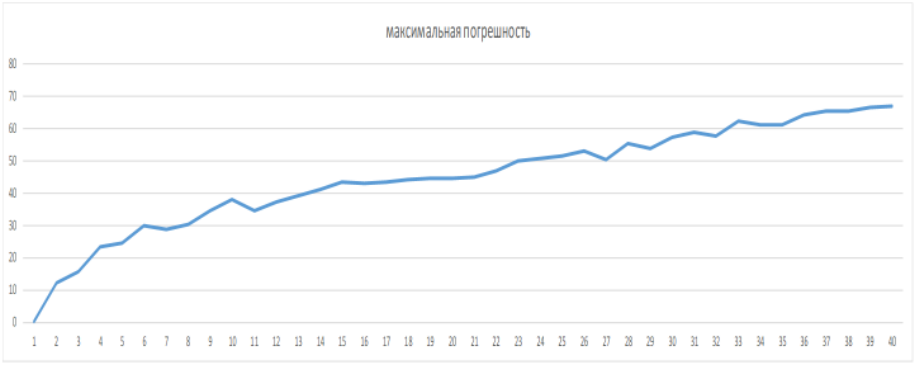
20

Прореживание



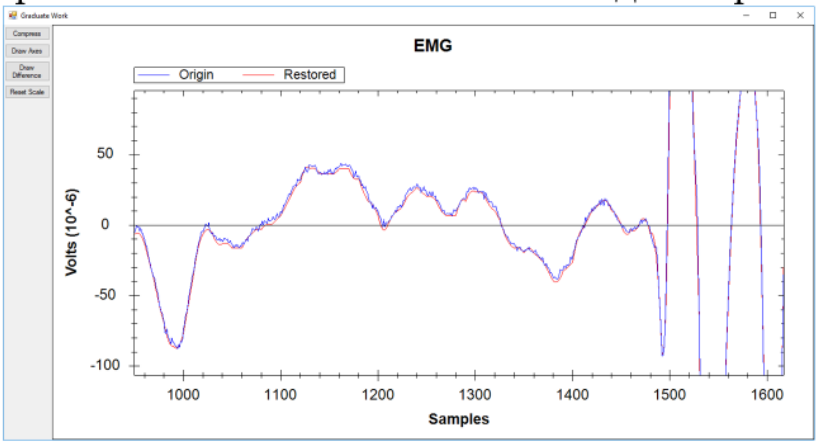
21

Максимальная погрешность в процентах



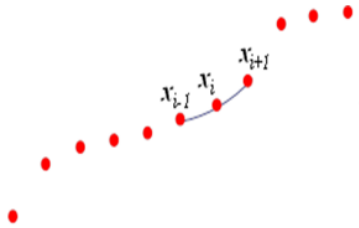
22

Сравнение восстановленной и исходной кривых



23

Интерполяция



$$a_0 = f(x_{i-1}) - a_1 x_{i-1} - a_2 x_{i-1}^2$$

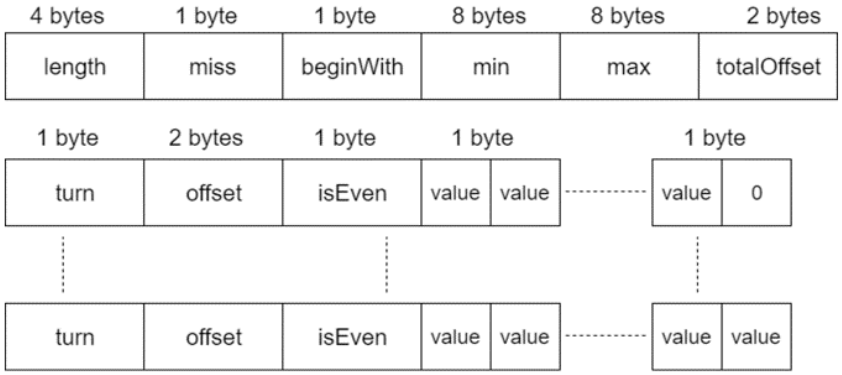
$$a_1 = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} - a_2(x_i + x_{i-1})$$

$$a_2 = \frac{f(x_{i+1}) - f(x_{i-1})}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)} - \frac{f(x_i) - f(x_{i-1})}{(x_i - x_{i-1})(x_{i+1} - x_i)}$$

$$\begin{cases} f_{i-1} = a_0 + a_1 x_{i-1} + a_1 x_{i-1}^2 \\ f_i = a_0 + a_1 x_i + a_1 x_i^2 \\ f_{i+1} = a_0 + a_1 x_{i+1} + a_1 x_{i+1}^2 \end{cases}$$

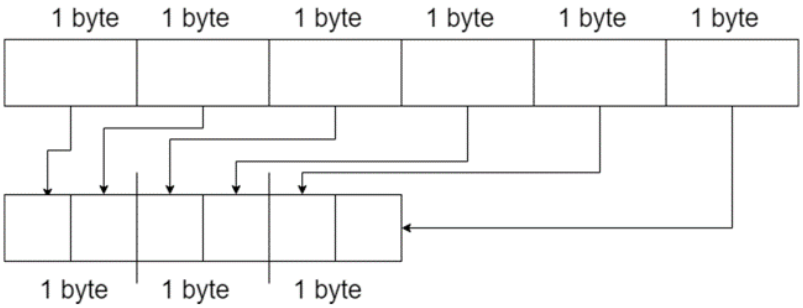
24

Схема представления данных в файле



25

упаковка значений в один байт



26

Результаты

	Исходный размер	WinRAR	ZIP	Разработанный алгоритм
Файл 1	1479648	1232070 (1,2)	1435051 (1,03)	54073 (27)
Файл 2	2390992	2016990 (1,18)	2313822 (1,03)	119572 (19)

27

Выводы

Поставленная цель достигнута, энергоэффективный алгоритм разработан.
Решены следующие задачи:

- Проведен анализ существующих алгоритмов сжатия
- Выбран опорный алгоритм (дельта кодирование)
- Разработан алгоритм оптимизированный для ЭМГ-данных
- Проведено сравнение с существующими универсальными алгоритмами. Показано, что разработанный алгоритм превосходит универсальные алгоритмы по степени сжатия

28

Спасибо за внимание

Приложение 3

Диплом за конкурс работ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.И.ЛЕНИНА

ДИПЛОМ

награждается
участник конкурса студенческих
научных работ
«Энергия-2018»

Васильев Дмитрий Максимович

за III место
среди учащихся старших курсов по направлению
«Математическое моделирование и информационные технологии»

Председатель оргкомитета,
проректор по научной работе В.В. Тютиков

Апрель 2018 г.

Приложение 4

Антиплагиат

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

Автор: Васильев Дима duscha1997@mail.ru / ID: 5837495
Проверяющий: Васильев Дима (duscha1997@mail.ru / ID: 5837495)

Отчет предоставлен сервисом «Антиплагиат»- http://www.antiplagiat.ru

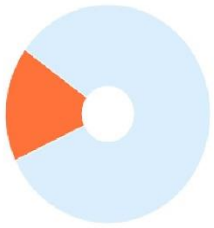
ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 3
Начало загрузки: 19.06.2018 22:29:26
Длительность загрузки: 00:00:07
Имя исходного файла: Васильев 4-46 ВКР
Размер текста: 1230 кБ
Символов в тексте: 57910
Слов в тексте: 7061
Число предложений: 778

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 19.06.2018 22:29:34
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска:

ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
17,19%	0%	82,81%



Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Доля в тексте	Источник	Ссылка	Актуален на	Модуль поиска	Блоков в отчете	Блоков в тексте
[01]	3,63%	3,63%	Электромиография (ЭМГ) - П...	http://medicalj.ru	18 Ноя 2017	Модуль поиска Интернет	17	17
[02]	0,27%	3,02%	Алгоритмы LZW, LZ77 и LZ78...	http://habrahabr.ru	раньше 2011	Модуль поиска Интернет	2	15
[03]	1,66%	2,78%	!!!Sekciya 4. Informatika....doc	http://enu.kz	раньше 2011	Модуль поиска Интернет	6	18

Еще источников: 13
Еще заимствований: 11,64%