**Минобрнауки России**

**Юго−Западный государственный университет**

Кафедра биомедицинской инженерии

**КУРСОВАЯ РАБОТА (ПРОЕКТ)**

по дисциплине « Теория и технология программирования для биотехнических систем»

(наименование дисциплины)

на тему «Объектно-ориентированный подход в разработке программных средств фильтрации биологических сигналов»

Направление подготовки (специальность) 12.03.04 Биотехнические системы

(код, наименование)

и технологии

Автор работы (проекта) И.Е. Штукин (инициалы, фамилия) (подпись, дата)

Группа БМ-91б

Руководитель работы (проекта) Д.Е. Скопин (инициалы, фамилия) (подпись, дата)

Работа (проект) защищена

(дата)

Оценка

Члены комиссии

(подпись, дата) (инициалы, фамилия)

(подпись, дата) (инициалы, фамилия)

(подпись, дата) (инициалы, фамилия)

Курск 2020 г.**Минобрнауки России**

**«Юго−Западный государственный университет»**

Кафедра биомедицинской инженерии

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ (ПРОЕКТ)

Студент Штукин И.Е. шифр 19-06-0165 группа БМ-91Б

(фамилия, инициалы)

1. Тема Объектно-ориентированный подход в разработке программных средств фильтрации биологических сигналов

2. Срок представления работы (проекта) к защите « 25 » декабря 2020 г.

3. Исходные данные (*для проектирования, для научного исследования*):

Разработать программу фильтрации биологических сигналов со следующими параметрами: частота дискретизации сигнала 600 Гц, коэффициенты фильтра: b1=1, b2=3, b3=1

4. Содержание пояснительной записки курсовой работы (проекта):

4.1. Основы разработки цифровых фильтров

4.2. Разработка программы фильтрации биологических сигналов

4.4. Результат использования программы

4.5. Изучение разработанного фильтра

4.6. Заключение

4.7. Список использованных источников

5. Перечень графического материала:

Не предусмотрен

Руководитель работы (проекта) Д.Е. Скопин (подпись, дата) (инициалы, фамилия)

Задание принял к исполнению И.Е. Штукин

(подпись, дата) (инициалы, фамилия)

РЕФЕРАТ

Курсовая работа изложена на 2222 страницах, включает в себя 15125125 разделов, введение, заключение, 112351251 рисунков. При создании работы были использованы 82348958923895 различных информационных источников.

Ключевые слова: цифровые фильтры, цифровая фильтрация, Java,

Целью работы является

Для достижения поставленной цели необходимо решить следующие задачи:

1. Исследование предметной области;
2. Анализ существующих алгоритмических решений;
3. Выбор необходимых модулей;
4. Разработка программного обеспечения

Область применения: данное программное обеспечение реализует создание анимации с помощью графических примитивов. Навыки, полученные в процессе разработки курсового проекта можно использовать для реализации пользовательского интерфейса в любых сферах деятельности, например медицине. В основном медицинское программное обеспечение использует подобные средства вывода информации для быстрого и, главное, наглядного оценивания нужных данных пользователем. С помощью показанных способов реализации представления графической информации, показанных в курсовом проекте, можно будет создавать программное обеспечение, выводящее на экран графики, анимацию, данные из базы в форме, удобной пользователю.

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc43552380)

РЕФЕРАТ

ABSTRACT

1. ОСНОВЫ РАЗРАБОТКИ ЦИФРОВЫХ ФИЛЬТРОВ

1.1. Основы разработки программного обеспечения на языке JAVA

1.2. Основы цифровой фильтрации

2. РАЗРАБОТКА ПРОГРАММЫ ФИЛЬТРАЦИИ БИОЛОГИЧЕСКИХ СИГНАЛОВ

2.1. Разработка класса генерации сигнала

2.2. Разработка класса фильтрации сигнала

2.3. Разработка метода сохранения данных

Разработка исследовательского режима работы

2.5. Разработка главной программы

3. РЕЗУЛЬТАТ ИСПОЛЬЗОВАНИЯ ПРОГРАММЫ

4. ИЗУЧЕНИЕ РАЗРАБОТАННОГО ФИЛЬТРА

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ВВЕДЕНИЕ

XXI век назвали веком информационных технологий не просто так – в последнее время человечество всё больше и больше стремится автоматизировать многие рутинные процессы, на которые раньше приходилось тратить огромное количество времени и сил. Благодаря алгоритмизации и информатизации люди в огромном темпе совершенствуют такие необходимые для человечества отрасли, как промышленную, экономическую, энергетическую и многие другие. В любом технологически-развитом производстве всегда будет важно использование цифровых фильтров. К примеру, большинство датчиков (качества воздуха, температуры, ИК и др.) используют аналоговые величины для отображения нужных характеристик. Здесь поможет цифровой фильтр, который будет интерпретировать и преобразовывать данные в вид, понятный компьютеру, а главное – пользователю.

В настоящее время очень часто можно встретить девайсы, включающие в свой состав цифровые фильтры. Самый распространённый пример цифрового фильтра – это эквалайзер в программах, для прослушивания звуковых файлов.

1 ОСНОВЫ РАЗРАБОТКИ ЦИФРОВЫХ ФИЛЬТРОВ

Цифровой фильтр — это математический алгоритм, реализованный на аппаратном и/или программном уровне, который с заданной целью действует на входной и генерирует выходной цифровой сигнал. Термином «цифровой фильтр» обозначают определенную аппаратную или программную процедуру, реализацию алгоритма фильтрации. В цифровых фильтрах часто используются оцифрованные аналоговые сигналы или просто хранящиеся в памяти компьютера числа, представляющие некоторые переменные.

По сути, фильтр — это система или сеть, избирательно меняющая форму сигнала (амплитудно-частотную или фазово-частотную характеристику). Основными целями фильтрации являются улучшение качества сигнала (например, устранение или снижение помех), извлечение из сигналов информации или разделение нескольких сигналов, объединенных ранее для, например, эффективного использования доступного канала связи.

Узкополосный аналоговый сигнал периодически выбирается и конвертируется в набор цифровых выборок, цифровой процессор производит фильтрацию, отображая входную последовательность в выходную согласно вычислительному алгоритму фильтра (рисунок 1). Цифро-аналоговый преобразователь конвертирует отфильтрованный цифровым образом выход в аналоговые значения, которые затем проходят аналоговую фильтрацию для сглаживания и устранения нежелательных высокочастотных компонентов.



Рис. 1 - Упрощенная схема цифрового фильтра реального времени с аналоговым входом и выходом

Цифровые фильтры играют важную роль в цифровой обработке сигналов. По сравнению с аналоговыми фильтрами они предпочтительны во множестве областей (например, сжатие данных, биомедицинская обработка сигналов, обработка речи, обработка изображений, передача данных, цифровое аудио, телефонное эхоподавление), так как обладают рядом преимуществ, часть из которых описана ниже.

* цифровые фильтры могут иметь характеристики, получить которые на аналоговых фильтрах практически невозможно, например, линейную фазовую характеристику;
* в отличие от аналоговых, производительность цифровых фильтров не зависит от изменений среды, например, от колебаний температуры. Таким образом, цифровые фильтры не требуют периодической калибровки;
* если фильтр построен с использованием программируемого процессора, его частотная характеристика может настраиваться автоматически (поэтому такие процессоры широко применяются в адаптивных фильтрах);
* обрабатываемые данные представлены в цифровом виде, поэтому их просто сохранить для последующего использования;
* можно легко использовать достижения из области технологий СБИС и получать небольшие цифровые фильтры с пониженной потребляемой мощностью и более низкой ценой;
* на практике точность, которой можно добиться при использовании аналоговых фильтров, ограничена; например, затухание в полосе подавления нельзя поднять выше 60-70 дБ (если использовать стандартные аналоговые компоненты). Точность цифровых фильтров ограничена только используемой длиной слова;
* производительность цифровых фильтров одинакова для всех устройств серии;
* цифровые фильтры могут использоваться при очень низких частотах, характерных, например, для многих биомедицинских приложений, где применять аналоговые фильтры невозможно. Кроме того, цифровые фильтры могут использоваться в большом диапазоне частот, для чего достаточно просто менять частоту дискретизации.

Не смотря на достоинства, по сравнению с аналоговыми, цифровые фильтры имеют и ряд недостатков:

* ограничение скорости. Максимальная ширина полосы сигналов, которые в реальном времени способны обработать цифровые фильтры, значительно уже, чем у аналоговых фильтров. Так как ограничивается быстродействием АЦП и ЦАП. Так же, скорость работы цифрового фильтра зависит от скорости работы цифрового процессора и числа арифметических операций, которые необходимо выполнить в алгоритме фильтрации;
* влияние конечной разрядности. Цифровые фильтры подвержены шуму (погрешности) АЦП, вызванного конечным числом уровней квантования (разрядность АЦП), и шуму округления, вызванного конечной разрядностью слова (обычно 8 или 16 бит). При использовании рекурсивных фильтров высоких порядков накопление шума округления может привести к неустойчивости фильтра;
* значительное время разработки и внедрения. Разработка и внедрение цифровых фильтров, особенно внедрение аппаратного обеспечения, могут выполняться гораздо дольше, чем подобные процедуры для аналоговых фильтров. В то же время, однажды разработанное аппаратное и/или программное обеспечение может использоваться в других задачах цифровой обработки сигналов с незначительной модификацией или вообще без изменений.

1.1. Основы разработки программного обеспечения на языке JAVA

Java - строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process; язык и основные реализующие его технологии распространяются по лицензии GPL. Права на торговую марку принадлежат корпорации Oracle.

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины. Дата официального выпуска — 23 мая 1995 года. На 2019 год Java — один из самых популярных языков программирования.

Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на Java:

* применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде,
* обширное использование платформенно-ориентированного кода (native-код) в стандартных библиотеках,
* аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами архитектуры ARM).

Время выполнения на Java составляет в среднем в полтора-два раза больше, чем для C/C++, в некоторых случаях Java быстрее, а в отдельных случаях в 7 раз медленнее. С другой стороны, для большинства из них потребление памяти Java-машиной было в 10—30 раз больше, чем программой на C/C++. Также примечательно исследование, проведённое компанией Google, согласно которому отмечается существенно более низкая производительность и бо́льшее потребление памяти в тестовых примерах на Java в сравнении с аналогичными программами на C++.

Идеи, заложенные в концепцию и различные реализации среды виртуальной машины Java, вдохновили множество энтузиастов на расширение перечня языков, которые могли бы быть использованы для создания программ, исполняемых на виртуальной машине. Эти идеи нашли также выражение в спецификации общеязыковой инфраструктуры CLI, заложенной в основу платформы .NET компанией Microsoft.

Основные возможности языка Java:

* Автоматическое управление памятью.
* Расширенные возможности обработки исключительных ситуаций.
* Богатый набор средств фильтрации ввода-вывода.
* Набор стандартных коллекций: массив, список, стек и т. п.
* Наличие простых средств создания сетевых приложений (в том числе с использованием протокола RMI).
* Наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы.
* Встроенные в язык средства создания многопоточных приложений, которые потом были портированы на многие языки (например Python).
* Унифицированный доступ к базам данных: на уровне отдельных SQL-запросов — на основе JDBC, SQLJ; на уровне концепции объектов, обладающих способностью к хранению в базе данных — на основе Java Data Objects (англ.) и Java Persistence API.
* Поддержка обобщений (начиная с версии 1.5).
* Поддержка лямбд, замыканий, встроенные возможности функционального программирования (с 1.8).

1.2 Основы цифровой фильтрации

Цифровой фильтр (ЦФ) есть система, преобразующая один цифровой сигнал в другой. Цифровой фильтр реализуется как программа на компьютере или аппаратным способом в виде цифровой схемы, содержащей регистры, сумматоры, умножители и другие вспомогательные элементы.

ЦФ называется линейным, если выходная последовательность y(n) при нулевых начальных условиях при воздействии вида

x(n) = a1x1(n) + a2x2(n)

описывается как

y(n) = a1y1(n) + a2y2(n),

где y1(n) и y2(n) соответственно отклики ЦФ на x1(n) и x2(n).

То есть при преобразовании входных последовательностей в линейном фильтре выполняется принцип суперпозиции.

Линейный ЦФ называется стационарным или инвариантным во времени (ЛИВ), если откликом на воздействие

x(n) = x1(n – n0)

является

y(n) = y1(n – n0).

Иными словами, сдвиг входной последовательности приводит к такому же сдвигу выходной последовательности без изменения ее формы.

Связь между входной x(n) и выходной y(n) последовательностями в стационарном ЦФ описывается линейным разностным уравнением с постоянными коэффициентами вида



Откуда, полагая а0 = 1 (нормируя относительно а0), получаем алгоритм работы цифрового фильтра во временной области:



2. РАЗРАБОТКА ПРОГРАММЫ ФИЛЬТРАЦИИ БИОЛОГИЧЕСКИХ СИГНАЛОВ

В нашей программе, реализующей полноценный цифровой фильтр, будет возможность использовать два режима работы.

Первый режим работы заключается в том, что будет создаваться сигнал по определённой формуле с постоянной частотой, а также будет происходить генерация выходного фильтрованного (через специфичный алгоритм) сигнала.

Второй режим работы – это так называемый «исследовательский» режим, в котором будет возможность смоделировать автоматизацию изменения частоты для исследования сигнала на его передаточную функцию.

Для организации удобства при написании программы, мы создадим класс «Formulas», в котором будут описаны все константы и основные переменные, которые мы будем использовать в своей программе. В него мы запишем следующее:

public class Formulas {

static double Fs = 600; //частота дискретизации

static int dlina\_frequency = (int) Fs / 2; //длина массива сигнала для исследовательской части

static double Ampl\_in = 220; //константа максимальной амплитуды на входе

static double Ampl\_out = 2; //амплитуда на выходе

static double F = 5; //тестовая частота

static double KotF; //коэффициент передачи от частоты

static double[] KotF\_now = new double[dlina\_frequency]; // массив коэффициента передачи для исследовательского режима

static double in\_max, in\_min, out\_max, out\_min; //переменные для записи максимальным и минимальных значений сигналов

static double[] inp\_max = new double[dlina\_frequency]; //входное максимальное значение для исследования

static double[] inp\_min = new double[dlina\_frequency]; //входное минимальное значение для исследования

static double[] outp\_max = new double[dlina\_frequency]; //выходное максимальное значение для исследования

static double[] outp\_min = new double[dlina\_frequency]; //выходное минимальное значение для исследования

static double[] ampl\_in\_now = new double[dlina\_frequency]; //входная амплитуда реального времени для исследования

static double[] ampl\_out\_now = new double[dlina\_frequency]; выходная амплитуда реального времени для исследования

2.1. Разработка класса генерации сигнала

Для создания цифрового фильтра нам потребуется разработать класс, в котором будет описан алгоритм генерации требуемого сигнала в определённый массив. Перейдём к описанию первого режима работы. Создадим класс под названием Signal.

Введём некоторые переменные внутри класса Signal.

static int max\_length\_freq; //Переменная для хранения значения максимальной частоты исследовательской части программы

static int step; //хранит значение шага цикла в исследовательской части

static int dlina; //сюда записываем длину сигнала для второго режима работы программы

static int usual\_length = 300; //записываем константу длины обычного сигнала

В данном случае значение этой переменной будет равно 300 – это означает, что наша программа создаст 300 значений сигнала. Нам достаточно трёхсот значений для разбора сигнала по одной частоте.

Теперь создадим массивы сигнала. В них будут записываться все значения нашей функции. Также будет создан массив frequency, в котором будет записываться значение текущей частоты.

static double[] input\_signal = new double[Formulas.dlina\_frequency];

static double[] input\_res\_signal = new double[Formulas.dlina\_frequency];

static double[] frequency = new double[Formulas.dlina\_frequency];

}

Теперь создаём метод «Create\_signal», при вызове которого будет происходить генерация сигнала, и массив входных значений будет заполняться.

Для создания нашего входного сигнала мы будем использовать следующую функцию:

где i – итератор, A – амплитуда, f – частота, Fs – частота дискретизации.

void Create\_signal()

{

for (int i = 0; i < dlina; i++) //создаём основной цикл

{

input\_signal[i] = Formulas.Ampl\_in \* Math.cos(2 \* Math.PI \* Formulas.F \* i \* (1 / Formulas.Fs));

}

Formulas.in\_max = Arrays.stream(input\_signal).max().getAsDouble(); //вычисляем максимум от всего массива входного сигнала

Formulas.in\_min = Arrays.stream(input\_signal).min().getAsDouble(); //вычисляем минимум от всего массива входного сигнала

System.out.println("Входной сигнал и амплитуда созданы");

}

2.2. Разработка класса фильтрации сигнала

Продолжим разработку созданием фильтрующего класса Filtration. В этом классе будет происходить преобразование входного сигнала в форму, указанную в задании к проекту.

Выходной сигнал будет генерироваться по следующей формуле:

Вводим необходимые для нас массивы:

double[] output\_signal = new double[Signal.dlina]; //массив выходного сигнала

double[] output\_res\_signal = new double[Formulas.dlina\_frequency]; //массив исследовательского выходного сигнала

Создаём метод Digital\_filter:

void Digital\_filter()

{

double[] beyond = new double[3]; //описываем промежуточный массив для вычисления выходного сигнала

double[] koef = new double[3]; //вводим массив с коэффициентами, данными нам в задании к проекту

koef[0] = 1; //заполняем массив коэффициента заданными в ТЗ

koef[1] = 3;

koef[2] = 1;

for (int j = 2; j < Signal.dlina; j++)

{

for (int g = 0; g < 3; g++) //создаём вложенный цикл для суммирования

{

beyond[g] = koef[g] \* Signal.input\_signal[j-g];

}

output\_signal[j] = beyond[0] + beyond[1] + beyond[2];

}

System.out.println("Цикл фильтра успешно прошёл");

}

Перейдём к написанию исследовательского класса фильтрации. Он будет отличаться от обычного фильтра лишь тем, что внутри его цикла каждый шаг будет считаться амплитуда и коэффициент передачи нашей функции от разной частоты.

Коэффициент передачи мы будем находить по следующей формуле:

2.3. Разработка метода сохранения данных

Теперь нам необходимо создать метод, который будет выводить все нужные нам значения переменных и массивов в доступный и удобный для просмотра вид. В данном случае мы будем записывать данные в .csv файл в виде таблицы, что в дальнейшем позволит нам тщательно исследовать сигналы с помощью графиков, создаваемых в программе Microsoft Excel. Также в конце метода мы будем вычислять некоторые исследовательские данные, такие как: минимальное и максимальное значение выходного сигнала, его амплитуду и общий коэффициент передачи.

Создадим метод Output\_Research:

void Output()

{

try //используем конструкцию try-catch для выявления ошибки, при записи файла

{

FileOutputStream fos = new FileOutputStream("E:\\1java\_prod\\Digital Filtration\\src\\output\_filter\\out.csv"); //создаём объект класса FileOutputStream и указываем в нём директорию расположения файла для записи значений в него

String fields\_csv = "Number;Input;Output;\n"; //записываем в переменную fields\_csv названия столбцов, используемых в таблице

byte[] b = fields\_csv.getBytes(); //декодируем наши названия столбцов в набор байтов

fos.write(b); //выводим набор байтов в файл (сохраняем названия столбцов)

for (int k = 0; k < Signal.dlina; k++)

{

int num = k+1;

fields\_csv = num + ";" + Signal.input\_signal[k] + ";" + output\_signal[k] +";\n"; //подставляем к каждому столбцу конкретное значение нужных именно там переменных и массивов

fields\_csv = fields\_csv.replace('.',','); //замена точек на запятые для более удобного просмотра

b = fields\_csv.getBytes();

fos.write(b); //запись в файл

}

String amplituda\_csv = "Input Max;Input Min;Output Max;Output Min;Inp Amplitude;Out Amplitude;KotF;\n"; //ниже основной таблицы мы выведем несколько переменных для исследования сигналов

byte[] c = amplituda\_csv.getBytes();

fos.write(c);

Formulas.out\_max = Arrays.stream(output\_signal).max().getAsDouble(); //находим максимальное значение выходного сигнала

Formulas.out\_min = Arrays.stream(output\_signal).min().getAsDouble(); //находим минимальное значение выходного сигнала

Formulas.Ampl\_out = (Formulas.out\_max - Formulas.out\_min)/2; //получаем выходную амплитуду

Formulas.KotF = Formulas.Ampl\_out / Formulas.Ampl\_in; //высчитываем коэффициент передачи

System.out.println(Formulas.out\_max); //выведем в консоль эти данные для отладки

System.out.println(Formulas.out\_min);

amplituda\_csv = Formulas.in\_max + ";" + Formulas.in\_min + ";"

+ Formulas.out\_max + ";" + Formulas.out\_min + ";"

+ Formulas.Ampl\_in + ";" + Formulas.Ampl\_out + ";" + Formulas.KotF + ";\n"; //точно так же, как и раньше, присваиваем переменной новые данные

c = amplituda\_csv.getBytes();

fos.write(c); //записываем исследовательские данные ниже основной таблицы

fos.close(); //закрываем файловый поток

System.out.println("Сигнал отфильтрован, амплитуда выходного сигнала создана");

}

catch (IOException e) {

e.printStackTrace(); //если произойдёт какая-то ошибка при записи файла, то её формулировка будет отображена в консоли

}

}

2.4 Разработка исследовательского режима работы

Теперь займёмся разработкой исследовательского метода создания сигнала. В начале выполнения программы пользователю требуется указать вручную требуемые: частоту, шаг цикла и количество значений.

Частота «max\_length\_freq» – мы можем выбрать любое число от единицы (нельзя выбрать отрицательную частоту, а также нельзя выбрать частоту 0, так как программа отработает некорректно) и до 300 (потому что частота дискретизации равна 600, а как мы знаем, можно использовать только частоты до Fs/2).

Шаг цикла «step» – его нам требуется указывать для того, чтобы рассмотреть определённое количество частот. Причём, например, выбрав шаг цикла 99, мы увидим только 3 частоты в нашей программе – 99, 198 и 297. Выбрав шаг частоты 1, мы просмотрим все частоты от 1 и до 300.

Количество значений «dlina» - мы самостоятельно можем выбрать, сколько значений сигнала мы выделим для каждой частоты. Нельзя выбрать больше 300 значений – файл вывода будет очень большим.

Создадим класс Choice\_of\_setup для вопросов к пользователю, а в нём метод Asking:

public void Asking()

{

Scanner scanner = new Scanner(System.in);

// С помощью бесконечных циклов и проверяющих условий мы ограничим пользователя от ввода неправильных значений для переменных.

while (true)

{

System.out.println("До какой частоты будем разгонять сигнал? Максимум = " + Formulas.dlina\_frequency);

Signal.max\_length\_freq = scanner.nextInt();

if (Signal.max\_length\_freq <= Formulas.dlina\_frequency && Signal.max\_length\_freq >= 1)

{

break;

}

else System.out.println("Частота не может быть больше 300, т.к. в ТЗ указана частота дискретизации 600. Нельзя выбрать отрицательную и нулевую частоту. Выберите другую");

}

// Если введённое пользователем значение подходит к нашему случаю, то мы оканчиваем бесконечный цикл командой break.

while (true)

{

System.out.println("Какой шаг вы хотите использовать для исследования? Рекомендуется не ниже " + (Signal.max\_length\_freq/3) + " . Максимум - " + (Signal.max\_length\_freq-1));

Signal.step = scanner.nextInt();

if (Signal.step >= 1 && Signal.step < Signal.max\_length\_freq)

{

break;

}

else System.out.println("Нельзя выбирать шаг меньше единицы. Нельзя выбирать шаг больше " + (Signal.max\_length\_freq-1) + ". Выберите шаг ещё раз");

}

while (true)

{

System.out.println("Сколько значений сигнала вы хотите получить? (для наглядности не более 50)");

Signal.dlina = scanner.nextInt();

if (Signal.dlina >= 3 && Signal.dlina <= 300)

{

Signal.dlina++;

break;

}

else System.out.println("Меньше 3 выбирать нельзя, а также нельзя сгенерировать значений больше, чем 300. Выберите ещё раз");

}

}

Исследовательские методы будут несколько похожи на прошлые, но теперь мы сделаем в одном методе всё сразу – и генерацию, и фильтрацию, и вывод. Это сделано для удобства, так как сразу будет происходить запись в файл.

Метод исследовательского вывода будет также выводить столбцы со значениями частоты (frequency) и коэффициента передачи реального времени (KotF\_now), а ещё он будет выводить значения в другой файл – в out\_cycle.csv.

После расспроса пользователя о данных генерации начнёт своё действие метод Create\_Research\_signal:

void Create\_Research\_signal() {

double[] beyond = new double[3];

double[] koef = new double[3];

koef[0] = 1;

koef[1] = 3; //ввод коэффициентов фильтра по ТЗ

koef[2] = 4;

try {

FileOutputStream fos = new FileOutputStream("E:\\1java\_prod\\Digital Filtration\\src\\output\_filter\\out\_cycle.csv"); //открываем файловый поток для нового файла вывода

for (int i = 0; i < max\_length\_freq; i = i + step) {

frequency[i] = i; //здесь происходит инкрементирование частоты. Это то, что мы указывали в переменной max\_length\_freq в начале выполнения программы

// Здесь мы создаём вложенный цикл, генерирующий значения входного сигнала

for (int z = 2; z <= dlina; z++) {

input\_res\_signal[z] = Formulas.Ampl\_in \* Math.cos(2 \* Math.PI \* frequency[i] \* z \* (1 / Formulas.Fs));

//Фильтрация начинается отсюда. Это ещё один вложенный цикл

for (int g = 0; g < 3; g++) {

beyond[g] = koef[g] \* Signal.input\_res\_signal[z - g];

}

Filtration.output\_res\_signal[z] = beyond[0] + beyond[1] + beyond[2];

// Более того, операция нахождения максимума и минимума всего массива будет происходить каждую итерацию. Это значит, что для каждого нового значения функции в массиве, мы сможем высчитать коэффициент передачи. Такой способ подсчёта позволяет получить идеальную точность измерений, так как не нужно будет вручную считать коэффициент передачи на отдельных частотах с каким-то шагом. Он будет высчитываться автоматически.

Formulas.inp\_max[z] = Arrays.stream(input\_res\_signal).max().getAsDouble(); //мы также ищем максимум входной функции, для вычисления амплитуды

Formulas.inp\_min[z] = Arrays.stream(input\_res\_signal).min().getAsDouble(); //вычисление минимума входного сигнала в реальном времени

Formulas.ampl\_in\_now[z] = (Formulas.inp\_max[z] - Formulas.inp\_min[z]) / 2; //нахождение входной амплитуды каждую итерацию

Formulas.outp\_max[z] = Arrays.stream(Filtration.output\_res\_signal).max().getAsDouble(); //ищем максимальное выходное значение

Formulas.outp\_min[z] = Arrays.stream(Filtration.output\_res\_signal).min().getAsDouble(); //ищем минимальное выходное значение

Formulas.ampl\_out\_now[z] = (Formulas.outp\_max[z] - Formulas.outp\_min[z]) / 2; //ищем амплитуду выходного сигнала по ранее найденным значениям

Formulas.KotF\_now[z] = Formulas.ampl\_out\_now[z] / Formulas.ampl\_in\_now[z]; //и, соответственно, коэффициент передачи при итерированной частоте.

// Теперь сделаем проверку на бесконечность. При определённых обстоятельствах бывает такое, что амплитуда сигнала может стать нулевой, тогда коэффициент будет вычисляться более корректно

if (Double.isInfinite(Formulas.KotF\_now[z]))

{

Formulas.KotF\_now[z] = 0;

}

// Запись в файл начинается здесь. Наши столбцы будут в таком порядке: Частота, Входной сигнал, Выходной сигнал, Коэффициент передачи.

String stolbec = frequency[i] + ";" + input\_res\_signal[z] + ";" + Filtration.output\_res\_signal[z] + ";" + Formulas.KotF\_now[z] + ";" + "\n";

stolbec = stolbec.replace('.', ',');

byte[] v = stolbec.getBytes();

fos.write(v); //записываем всё в файл каждую итерацию цикла

}

// Вывод данных в консоль для отладочных целей

System.out.println("вх ампл " + Formulas.ampl\_in\_now[i] + " на частоте " + i);

}

// Теперь найдём средние максимальные и минимальные значения для отладки

Formulas.in\_max = Arrays.stream(input\_res\_signal).max().getAsDouble();

Formulas.in\_min = Arrays.stream(input\_res\_signal).min().getAsDouble();

System.out.println("Входной исследовательский сигнал и амплитуда созданы");

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

2.5. Разработка главной программы

Главная программа в нашем проекте будет состоять только из вызова описанных ранее методов.

Создаём главный класс Filter\_main:

public class Filter\_main {

public static void main(String[] args) {

Signal signal = new Signal(); //создаём объект класса Signal

Filtration filtration = new Filtration(); //создаём объект класса Filtration

Choice\_of\_setup choice = new Choice\_of\_setup(); //создаём объект класса Choice\_of\_setup

signal.Create\_signal(); //вызываем метод создания сигнала

filtration.Digital\_filter(); //вызов фильтрации сигнала

filtration.Output(); //вызов вывода данных в таблицу

System.out.println("Тестовый метод отработал");

choice.Asking(); //вызываем метод для опроса пользователя

signal.Create\_Research\_signal(); //после опроса начинает свою работу исследовательский режим

System.out.println("Исследовательский метод отработал");

}

}

3. РЕЗУЛЬТАТ ИСПОЛЬЗОВАНИЯ ПРОГРАММЫ

В процессе запуска программы нам необходимо ввести необходимые программе данные (рисунок 2).

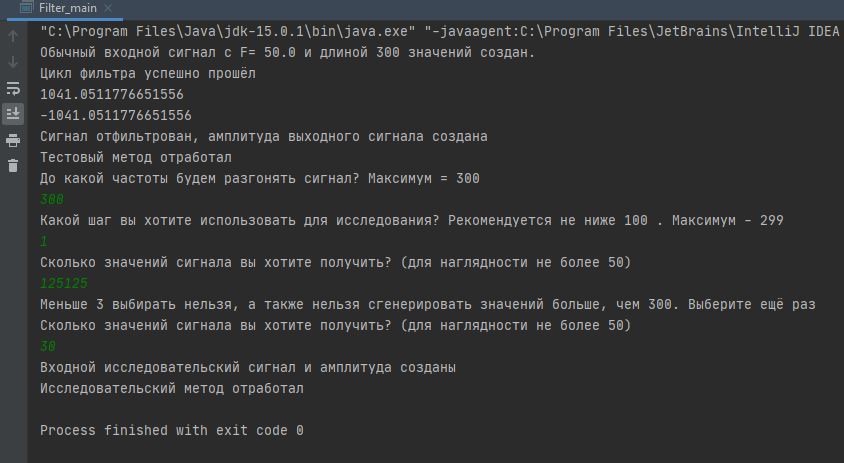


Рис. 2 – Ввод данных в программу

На выходе мы получаем два файла: out.csv и out\_cycle.csv.

При открытии файла out.csv мы увидим готовые значения сигналов, созданные с помощью первого режима программы (рисунок 3).

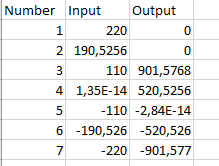


Рис. 3 – Результат работы стандартной фильтрации сигнала и вывода значений в таблицу

Также снизу таблицы запишутся отдельные значения наших сигналов для проверки выполнения алгоритма (рисунок 4).

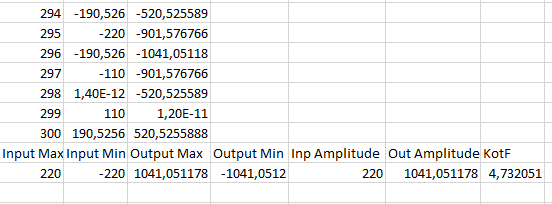


Рис. 4 – Обобщённые данные по всем сигналам внизу таблицы

Теперь проверим файл out\_cycle.csv. В нём мы увидим разделённые значения частоты, входного и выходного сигнала, а также коэффициента передачи.

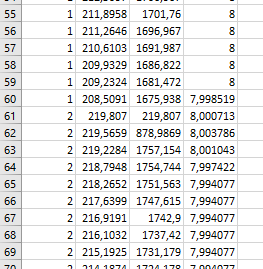


Рис. 5 – Данные, выведенные алгоритмом в файл out\_cycle.csv

Если попытаться запустить программу, когда один из наших файлов вывода уже открыт, то программа не запустится и кинет эксепшн с названием ошибки (рисунок 6).

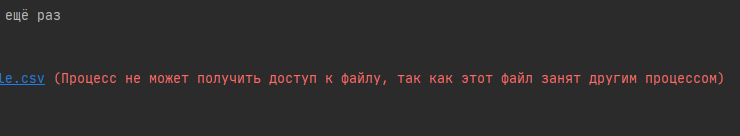


Рис. 6 – Результат запуска программы при использовании наших таблиц другими процессами

4. ИЗУЧЕНИЕ РАЗРАБОТАННОГО ФИЛЬТРА

Для анализа данного сигнала составим графики входного и выходного сигнала. Для начала используем первый файл out.csv, созданный первым режимом работы программы.

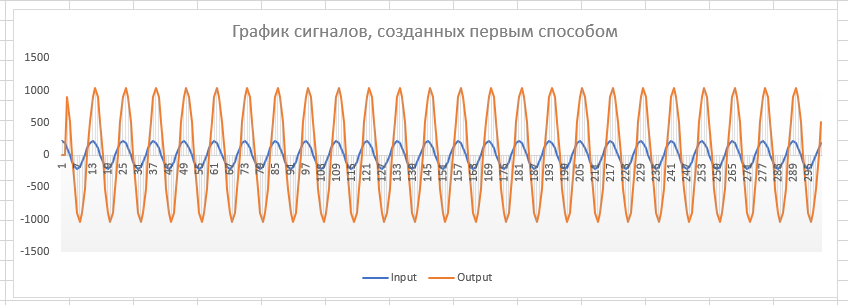


Рис. 7 – Графики входного и выходного сигналов при F=50, Aвх=220 и длине сигнала в 300 значений

Как мы видим, наш фильтр увеличивает амплитуду сигнала. Причём в целом на этой частоте выходной сигнал увеличивается почти в 8 раз. Обратите внимание, что происходит небольшой фазовый сдвиг выходного сигнала от входного.

Но благодаря тому, что мы создали метод для отображения коэффициента передачи реального времени, мы можем гораздо глубже исследовать наш фильтр, нежели бы вручную искать коэффициенты отдельных частот (рисунок 8-9).

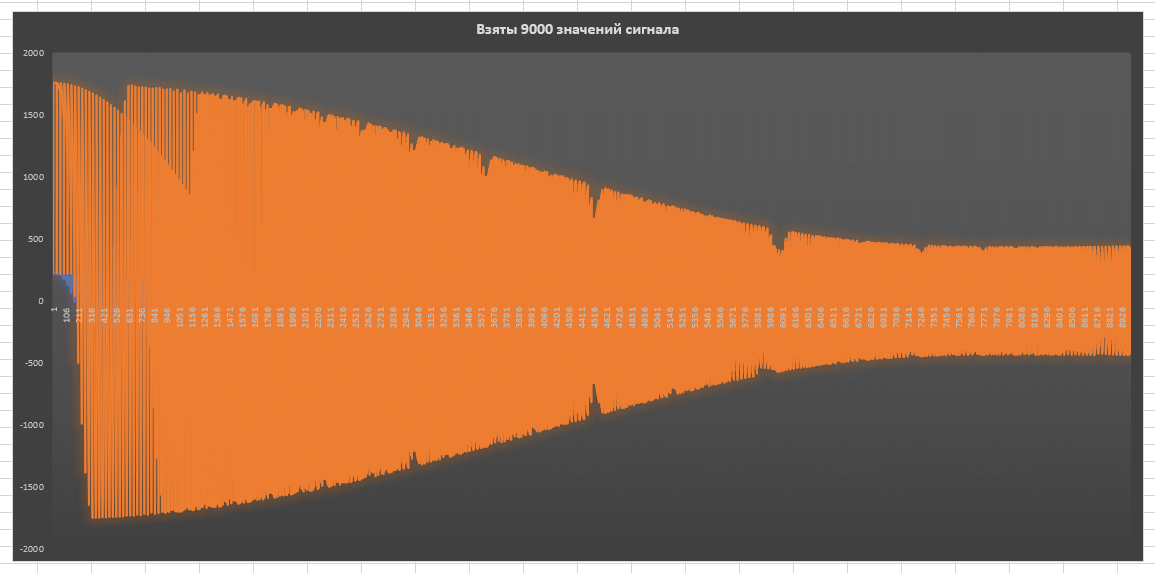


Рис. 8 – График входного и выходного сигнала на разных частотах с шагом 1. Здесь оранжевый сигнал – это выходной сигнал. А синий – входной

Анализируя график, можно заметить, что лучше всего фильтр усиливает частоты от 1 до 148 Гц, а потом, после точки среза, сигнал несколько стабилизируется, относительно прошлых значений.



Рис. 9 – График коэффициента передачи к разным частотам. На оси абсцисс расположен номер значения, а по оси ординат – коэффициент передачи сигнала.

Следует обратить внимание на то, что коэффициент передачи снижается с повышением частоты, причём можно заметить точки среза на некоторых частотах. Если конкретно, то самая сильная точка среза находится на частотах 150-151 Гц. Здесь коэффициент передачи снижается до 3 (рисунок 10).

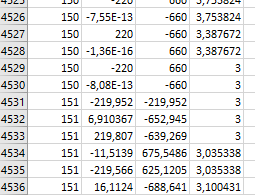


Рис. 10 – Точка среза на частотах 150-151 Гц

ЗАКЛЮЧЕНИЕ

Благодаря глобальной цифровизации и автоматизации вычислительных средств, мы можем навсегда забыть про аналоговые фильтры, собранные из отдельных радиоэлементов, так как они не способны так точно и качественно фильтровать входной сигнал.

С помощью разработанной в данном проекте программы-фильтра можно исследовать любые типы сигналов, в том числе и биологических. Можно сказать, что фильтрация, требуемая по техническому заданию, отлично увеличивает амплитуду входного сигнала. Это значит, что, если на вход будет поступать маленький и плохой сигнал с небольшой частотой, наш цифровой фильтр хорошо усилит его, что позволит более плотно работать над дальнейшими исследованиями этого сигнала. Также достаточно интересно, что усиление сигнала происходит слабее на более высоких частотах.

В дальнейшем эту программу можно доработать – создать для неё более понятный пользовательский интерфейс и добавить туда возможность быстрой замены формул, по которым осуществляется фильтрация или создание сигнала. Можно сказать, что после некоторых доработок для этой программы может получится цифровой осциллограф.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Система программирования PascalABC.NET [Электронный ресурс] // pascalabc.net – Copyright, 2020. – URL: http://pascalabc.net/ (дата обращения 02.06.2020).
2. Абрамов, В. Г. Введение в язык паскаль / В.Г. Абрамов, Н.П. Трифонов, Г.Н. Трифонова. – М.: Наука, 2012. – 320 c.
3. Андреева, Т.А. Программирование на языке Pascal / Т.А. Андреева. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2013. – 139 c.
4. Циклы в Паскале [Электронный ресурс] // pas1.ru – Copyright, 2020. – URL: https://pas1.ru/cycles (дата обращения 02.06.2020).
5. Pascal: Занятие №2. Часть 1: Условный оператор в Паскале (If) [Электронный ресурс] // labs-org.ru – Copyright, 2019. – URL: https://labs-org.ru/pascal-2/ (дата обращения 02.06.2020).
6. Стандартные модули: базовый уровень [Электронный ресурс] // pascalabc.net – Copyright, 2020. – URL: http://pascalabc.net/wiki/index.php?title=Стандартные\_модули:\_базовый\_уровень (дата обращения 02.06.2020).
7. System.Media Пространство имен [Электронный ресурс] // docs.microsoft.com – Copyright, 2020. – URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.media?view=dotnet-plat-ext-3.1 (дата обращения 03.06.2020).
8. Что такое тестирование программного обеспечения [Электронный ресурс] // QAlight – Copyright, 2020. – URL: https://qalight.com.ua/baza-znaniy/chto-takoe-testirovanie-programmnogo-obespecheniya/ (дата обращения 03.06.2020).
9. ГОСТ 2.105 – 95 Единая система конструкторской документации. Общие требования к текстовым документам. – М.: Стандартинформ, 2011 – 41 с.
10. ГОСТ 7.1 – 2003 Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: Стандартинформ, 2010 – 61 с.
11. СТУ 04.02.030-2017 [Электронный ресурс] // Стандарты университета. – Курск.: Юго-Западный государственный университет, 2020. – URL: https://swsu.ru/omk/normative\_documents\_cm/СТУ%2004.02.030-2017\_4.5\_.pdf (дата обращения 17.06.2020).