МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра прикладной математики

Допустить к защите
И.о. заведующего кафедрой
канд. физ.-мат. наук, доц
_____А.В. Письменский
_____2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СИСТЕМЫ ИЗМЕРЕНИЙ И ОБРАБОТКИ СИГНАЛОВ С ИСПОЛЬЗОВАНИЕМ ЦИФРОВЫХ USB-ОСЦИЛЛОГРАФОВ

Работу выполнил(а) _____ И.Д. Кузнецов

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность (профиль) Математическое и информационное обеспечение экономической деятельности

Научный руководитель
д-р. физ.-мат. наук, проф. _____ Е.В. Глушков

Нормоконтролер
преподаватель _____ Е.С. Троценко

Краснодар
2023

РЕФЕРАТ

Выпускная квалификационная работа 40 с., 3 ч., 16 рис., 7 прил., 14 источн.

СЧИТЫВАНИЕ СИГНАЛА, ОБРАБОТКА СИГНАЛА, ЦИФРОВОЙ ОСЦИЛЛОГРАФ, СРЕДНЕЕ ПО АНСАМБЛЮ СИГНАЛОВ, ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ, БИБЛИОТЕКА ДИНАМИЧЕСКОЙ КОМПОНОВКИ

Целью работы является реализация библиотеки динамической компоновки для считывания данных с цифрового запоминающего осциллографа и проведения их обработки для шумоподавления, а также описание процесса создания этой библиотеки для потенциальной передачи опыта желающим создать аналогичный проект для своих устройств.

Поставленные цели были достигнуты: полученная библиотека была успешно применена при помощи вызова её функций в языке программирования C#, а рабочий код был выложен в свободный доступ в веб-сервис для хостинга IT-проектов GitHub.

Работа представляет собой авторский проект, сочетающий в себе считывание и обработку сигнала, который может быть применён как по прямому назначению в лабораторных условиях, так и как гибкий математический пакет для вычисления вейвлет-преобразований, что может быть полезным широкому кругу лиц, даже незнакомых с теорией обработки волновых сигналов. Работа также может быть полезной с точки зрения обучения обработке сигналов, так как она содержит не только теоретические знания для решения данной задачи, но и конкретные примеры её реализации с описанием неочевидных моментов.

В дальнейшем планируется расширение функционала разработанной программной системы и повышение гибкости использования реализованного вейвлет-преобразования.

СОДЕРЖАНИЕ

Введение 3
1 Постановка задачи 5
2 Особенности использования цифрового осциллографа 6
2.1 Подготовка к взаимодействию с осциллографом 6
2.2 Параметры настройки осциллографа 11
3 Обработка сигнала 14
3.1 Аналоговая обработка сигнала 14
3.2 Нахождение среднего по ансамблю сигналов 15
3.3 Вейвлет-преобразование 18
3.3.1 Подготовительные теоретические знания 18
3.3.2 Вывод формул для программной реализации 22
3.3.3 Процесс выбора вейвлета 24
3.3.4 Особенности программной реализации вейвлет-преобразования 28
Заключение 31
Список использованных источников 32
Приложение А Хранение и обработка допустимых значений вертикального масштабирования 34
Приложение Б Нахождение среднего по ансамблю сигналов 35
Приложение В Преобразование значений переменных времени с учётом пропуска семплов 36
Приложение Г Обработка файла со считыванием данных с учётом пропуска семплов 37
Приложение Д Построение вейвлет-функции 38
Приложение Е Пример использования многопоточности 39
Приложение Ж Реализация формулы Котеса 40

ВВЕДЕНИЕ

Анализ цифровых сигналов является важным этапом в решении многих практических задач, например, в контексте обнаружения дефектов в тонкостенных инженерных конструкциях с использованием ультразвуковых систем мониторинга их состояния [1] или для цифровой обработки речевых сигналов [2]. Однако, прежде чем анализировать сигнал, его необходимо корректно извлечь из измеряющего устройства, например, цифрового запоминающего осциллографа, и избавиться от посторонних шумов, не соответствующих колебаниям исследуемых объектов.

Таким образом, целью текущей работы являются создание библиотеки динамической компоновки (DLL) на языке программирования C++, при помощи которой можно считать и обработать цифровой сигнал, и описание процесса создания этой библиотеки с целью потенциальной передачи опыта желающим создать аналогичный проект для своих устройств. Описание кода при этом приводится в предположении, что читатель уже ознакомлен с базовыми особенностями языка C++.

Разработка данной библиотеки была вызвана реальной научной потребностью в сборе данных с распределительной сети пьезосенсеров и проведением измерений бесконтактным пьезоакустическим преобразованием с использованием координатного стола. В подобных условиях шум может значительно исказить сигнал, а потому одной из важнейших особенностей разрабатываемой библиотеки является высокая эффективность подавления шумов при приемлемо малой продолжительности обработки сигнала.

Таким образом, разрабатываемый продукт должен работать максимально эффективно (с этой целью и был выбран язык программирования C++), быть легко переносимым и иметь возможность дальнейшего развития под свои нужды.

В работе сигнал считывается с цифрового осциллографа OWON VDS6102A ввиду его доступности, приемлемой стоимости и хорошей точности (14 бит для вертикального разрешения и 125 млн значений в секунду с минимальным шагом в 5 нс для горизонтального разрешения).

1 Постановка задачи

Программное взаимодействие с цифровым осциллографом OWON VDS6102A осуществляется при помощи драйверов от компании NI-VISA [3]. В общем случае алгоритм «общения» с устройством можно описать следующим образом: нахождение нужного устройства, подключение к нему, отправка запроса, чтение данных, закрытие соединения. Причём отправка запроса и чтение данных в действительности могут быть произведены неоднократно за одну сессию. Важно отметить, что данный алгоритм не является уникальным ни для используемой модели осциллографа, ни для используемых драйверов, что говорит о некоторой универсальности и полезности дальнейшего описания процесса считывания данных для разработчиков аналогичного программного обеспечения.

Данные, считанные с осциллографа, могут быть зашумлены посторонними сигналами вплоть до нечитаемости полезного сигнала. По этой причине, в разрабатываемой библиотеке динамической компоновки должны присутствовать реализации методов обработки сигналов, способные тем или иным способом произвести шумоподавление.

Наконец, уже разработанная библиотека должна быть успешно проверена в обработке реального сигнала.

2 Особенности использования цифрового осциллографа

2.1 Подготовка к взаимодействию с осциллографом

Как было замечено выше, для программного взаимодействия с цифровым осциллографом OWON VDS6102A используются драйвера от NI-VISA, представляющие из себя интерфейс ввода-вывода для управлениям приборами с персонального компьютера в форме «запрос-ответ». Однако данные драйвера не являются предустановленными в OS Microsoft Windows, поэтому скачивать и устанавливать их требуется отдельно. А после установки драйверов, прежде чем воспользоваться ими в коде, необходимо подключить необходимые заголовочные файлы и библиотеки к проекту. Осуществляется это при помощи директив #include и #pragma аналогично коду на рисунке 1. Путь к драйверам VISA может отличаться в зависимости от места, указанного во время установки драйверов.

```
1   #include "C:\Program Files\IVI Foundation\VISA\Win64\Include\visa.h"
2   #pragma comment(lib, "C:/Program Files/IVI Foundation/VISA/Win64/Lib_x64/msc/visa64.lib")
```

Рисунок 1 – Подключение заголовочных файлов и библиотек драйвера NI-VISA

Далее, прежде чем начать работу с устройством, необходимо к нему подключиться. Для этого требуется открыть сессию VISA, найти нужное устройство и, наконец, открыть соединение с ним. Пример подключения к устройству по USB проиллюстрирован на рисунке 2.

```
1   static ViSession defaultRM;
2   static ViSession instr;
3   static ViUInt32 numInstrs;
4   static ViFindList findList;
5   static ViStatus status;
6   static ViStatus status1;
7   static ViStatus status2;
8   static ViChar instrResourceString[VI_FIND_BUFLEN];
9
10  static char* buffer;
11  static char stringinput[512];
12
13  status1 = viOpenDefaultRM(&defaultRM);
14  if (status1 < VI_SUCCESS)
15      return status1;
16  ViConstString str = "USB?*INSTR";
17  status2 = viFindRsrc(defaultRM, str, &findList, &numInstrs, instrResourceString);
18  if (status2 < VI_SUCCESS) {
19      fflush(stdin);
20      viClose(defaultRM);
21      return status2;
22  }
23  status = viOpen(defaultRM, instrResourceString, VI_EXCLUSIVE_LOCK, VI_NULL, &instr);
24  if (status < VI_SUCCESS)
25      return status;
```

Рисунок 2 – Пример кода для подключения к цифровому осциллографу по USB

Важно отметить, что в случае внештатного поведения при подключении к устройству, например при создании сессии (13 строка на рисунке 2), при поиске устройства (17-я) или при собственно подключении (23-я), встроенные функции VISA вернут статус ошибки, который всегда меньше по значению статусов успеха (тип ViStatus является расширением типа int).

Далее, в случае успешного подключения к осциллографу, требуется отправить на него запрос на считывание данных. Однако, чтобы получить нужные данные, сперва осциллограф требуется настроить. Подробнее параметры настройки осциллографа будут разобраны в следующей главе, однако её пример, используемой в работе библиотеки, проиллюстрирован на рисунке 3. На данный момент важно отметить строки с 20 по 23: именно благодаря ним определяется начало считывания (BEG), определяются первый семпл и количество считываемых семплов в целом (RANG), производится процесс считывания (FETC?) и, наконец, завершается считывания (END) данных.

```
1  string m = (
2      string(":*RST;\n") +
3      string(":ACQ:PREC 14;\n") +
4      string(":ACQ:MODE PEAK;\n") +
5      string(":MEAS:TIM 0.002;\n") +
6      string(":CH1:SCAL ") + vertScale_to_string(vertScale) + string(";\n") +
7      string(":CH1:BAND 20M;\n") +
8      string(":CH1:COUP AC;\n") +
9      string(":CH1:OFFS 0;\n") +
10     string(":CH2:COUP AC;\n") +
11     string(":CH2:OFFS 0;\n") +
12     string(":HORI:SCAL ") + horiScale_to_string(horiScale) + string(";\n") +
13     string(":ACQ:DEPMEM ") + acqDepmem(acquiredDepmem) + string(";\n") +
14     string(":TRIG:SING:MODE EDGE;\n") +
15     string(":TRIG:SING:EDGE:SOUR CH2;\n") +
16     string(":TRIG:SING:EDGE:COUP AC;\n") +
17     string(":TRIG:SING:EDGE:SLOP RISE;\n") +
18     string(":TRIG:SING:EDGE:LEV ") + to_string(trigger_level) + string(";\n") +
19     string(":HORI:OFFS 10;\n") +
20     string(":WAV:BEG CH1;\n") +
21     string(":WAV:RANG 0,") + to_string(sample_size) + string(";\n") +
22     string(":WAV:FETC?;\n") +
23     string(":WAV:END;\n")
24 );
25
26 const char* st = m.c_str();
27
28 strcpy_s(stringinput, RSIZE_MAX, st);
29
30 m = m + m;
```

Рисунок 3 – Пример запроса для настройки работы осциллографа

В данном примере запрос собирается путём конкатенации множества строк, каждая из которых отвечает за свой параметр. После же конкатенации требуется перевести данные в специальный тип char*, т.к. этого в дальнейшем работа может производиться только с этим типом данных. При этом на строке 30 использована небольшая хитрость, не позволяющая сборщику мусора удалить строку m до выполнения работы строки 26.

В коде можно обратить внимание на использование дополнительных функций. Они были прописаны самостоятельно, и их цель – преобразовать входные параметры, удобные для ввода пользователем, в требуемый строковый вид, воспринимаемый считывающим устройством. Причём так как устройство имеет лишь ограниченный набор значений для некоторых параметров, то данные параметры заранее прописаны в коде в виде перечисления (тип enum). Пример данного перечисления для допустимых

значений вертикального разрешения и функция перевода этих значений в строковый вид можно рассмотреть в приложении А.

Хотя смысл каждого из параметров описан в документации устройства [4], очень важно обратить внимание на то, что каждая из команд обязана заканчиваться на символ конца строки ‘\n’. В ином случае команды будут распознаны неверно.

На рисунке 4 приведен пример записи на устройство описываемого на рисунке 3 запроса. Из примечательного стоит выделить то, что после открытия соединения недостаточно просто закончить выполнение программы при возникновении ошибки. Требуется сперва закрыть существующее соединение, иначе во время дальнейшей работы устройство может работать некорректно, например не принимать запрос на подключение вплоть до его перезагрузки.

```
1  static ViUInt32 writeCount;
2  status = viWrite(instr, (ViBuf)stringinput, (ViUInt32)strlen(stringinput), &writeCount);
3  if (status < VI_SUCCESS) {
4      status = viClose(instr);
5      if (status < VI_SUCCESS)
6          return status;
7      status = viClose(defaultRM);
8      return status;
9  }
```

Рисунок 4 – Пример записи запроса на цифровой осциллограф

Особенно интересным представляется заключительный этап считывания данных с осциллографа, проиллюстрированный на рисунке 5. Так как данные в текстовом виде передаются гораздо дольше и занимают гораздо больше памяти, то по-умолчанию программа считывает все данные в виде массива битов.

```
lastNum = 0;
static char* buffer = new char[sample_size * 2 + 16];
do {
    status = viRead(instr, (ViPBuf)buffer, 2 * sample_size + 16, &retCount);
    if (status < VI_SUCCESS)
        return status;
    for (ii = 0; ii < retCount / 2; ii++)
        result_arr[lastNum + ii] +=
            (float(int(buffer[ii * 2])) + int(char(buffer[ii * 2 + 1]) >> 2) * 0.015625) *
            voltage_max * 0.008; //0.008=1/25 / 5
    lastNum += retCount / 2;
} while (status == VI_SUCCESS_MAX_CNT);
```

Рисунок 5 – Процесс считывания битовых данных с осциллографа

Здесь переменная sample_size отвечает за количество считываемых единиц информации (семплов), сохраняемой в массиве result_arr соответствующего размера. Так как при работе с устройством может быть неизвестно точное количество семплов, то программа считывает их при помощи буфера постепенно до тех пор, пока размер считанных в текущей итерации данных не окажется меньше размера буфера.

Известно, что вертикальное разрешение устройства составляет 14 бит. Однако оно не способно передавать 14 бит информации, т.к. она передаётся в байтах. Поэтому в действительности каждый семпл имеет размер 16 бит, где 2 последних бита – незначимые, а остальные соответствуют типичному представлению числа с плавающей запятой в ЭВМ: первые 8 бит выделены на мантиссу, то есть на целую часть числа, а ещё 6 – на порядок, то есть на дробную её часть [5]. Однако так как C++ не способен самостоятельно перевести 16 бит в 14-битное число с плавающей запятой, требуется сделать это вручную, разбив число на байты. Таким образом, буфер имеет размер в два раза больший, нежели количество семплов, плюс небольшое количество для отладочных данных, отправляемых осциллографом автоматически.

Сама обработка двух байтов происходит по следующему алгоритму: сперва к 1-му байту прибавляется 2-й байт, у которого благодаря побитовому сдвигу вправо незначимыми стали первые 2 бита, домноженному на 2^-(14-8), а далее полученная сумма нормируется, используя значения текущего вертикального разрешения и нормирующей константы.

2.2 Параметры настройки осциллографа

В действительности параметров настройки осциллографа существует большое множества, о чём можно подробнее почитать в соответствующей документации [4]. Однако для поставленных задач оказалось достаточно использовать лишь некоторые из них, а параметризировать – всего 4-5. В данной главе будет рассмотрено назначение каждого из них.

Параметр *RST (строка 2 на рисунке 3) используется для сброса предыдущих настроек осциллографа. Это может оказаться полезно в том случае, если помимо используемого кода существуют другие методы взаимодействия с осциллографом.

Параметр ACQ:PREC (строка 3) определяет вертикальное разрешение считываемых значений. Таким образом, под каждое число будет выделено 14 бит: это значение максимально и неизменяемо, так как точность вычислений является важной характеристикой в поставленной задаче.

Параметр ACQ:MODE (строка 4) определяет, каким образом записывается считанный сигнал: по семплам (SAMP) или по пикам (PEAK). Второй вариант в общем случае является немного более точным, особенно при достаточно большом расстоянии между шагами, поэтому именно он и был выбран.

Параметр MEAS:TIM (строка 5) определяет временные интервалы между считываниями сигнала в секундах. По причине той же точности было выбрано минимальное из них, то есть 20 мс.

Параметры на строках с 6 по 9 и с 10 по 11 отвечают за настройку соответствующих каналов. Первый из них используется настроен для считывания самого сигнала, а второй – для считывания триггера, позволяющего запустить считывание сигнала.

При помощи параметра :SCAL (строка 6) определяются границы вертикального разрешения. Чем они шире, тем больший диапазон сигнала

способен считать осциллограф, но тем меньше будет точность считываемого сигнала.

При помощи параметра :BAND (строка 7) можно активировать использование аналогового фильтра при считывании сигнала. Данный фильтр способен отсекать любой сигнал с частотой выше 20 МГц. Подробнее о работе данного фильтра будет рассказано далее.

При помощи параметра :COUP (строки 8 и 10) определяется вид тока, считываемый осциллографом: AC – только переменный, DC – переменный и постоянный, GND – «земля». Так как в постановленной задаче интерес представляет только переменный ток, то параметр всегда равен AC.

При помощи параметра :OFFS (строки 9 и 11) определяется горизонтальный сдвиг (по времени) считываемых значений. В поставленной задаче эта возможность не представляет особого интереса.

Параметр HORI:SCAL (строка 12) определяет горизонтальное разрешение, аналогично вертикальному разрешению.

Параметр ACQ:DEPMEM (строка 14) определяет количество семплов, сохраняющихся при считывании данных с одного триггер-сигнала. Это значение обязано быть не меньше общего количества считываемых за раз семплов.

Параметры с приставкой TRIG:SING: (строки 14-18) отвечают за настройку триггер-сигнала.

Параметр MODE (строка 14) определяет условие реакции на триггер-сигнал. Среди всех значений этого параметра наиболее удобным является режим EDGE: реакция на триггер-сигнал производится в тот момент, когда его энергия переходит через определённое значения. Дальнейшая настройка триггера (строки 15-18) производится именно внутри этого режима.

Параметр SOUR (строка 15) определяется канал-источник триггер-сигнала. Как уже было сказано ранее, таким источником был выбран второй канал.

Параметр SLOP (строка 17) определяет направление изменения значения триггер-сигнала. Режим RISE устанавливает возрастающее изменение, режим FALL – убывающее.

Параметр LEV (строка 18) определяет значение амплитуды триггер-сигнала, относительно которого применяется параметр SLOP.

Параметр HORI:OFFS (строка 19) определяет горизонтальный сдвиг считываемого сигнала. Сам осциллограф разделяет по горизонтали все считываемые значения на 20 делений: 10 до возбуждения триггера и 10 после него. Первая половина этих значений не содержит полезного сигнала. Именно по этой причине и производится сдвиг на 10 делений (вправо).

Оставшиеся параметры (строки 20-23) уже не настраивают считывание данных осциллографом, а собственно запускают это считывание и были рассмотрены в предыдущем разделе.

3 Обработка сигнала

3.1 Аналоговая обработка сигнала

Самым простым способом обработки данных, легко реализуемом на большинстве современных осциллографах, является встроенный аналоговый фильтр. Активировав его, можно отсечь все частоты, которые выходят за диапазон фильтра, избавившись от высокочастотных шумов. На VDS6102A аналоговый фильтр включается при помощи задания параметру канала BANDwidth значения 20M. Таким образом, при помощи данного аналогового фильтра можно отсечь все частоты выше 20 МГц. Результат применения данного фильтра к сигналу проиллюстрирован на рисунке 6.

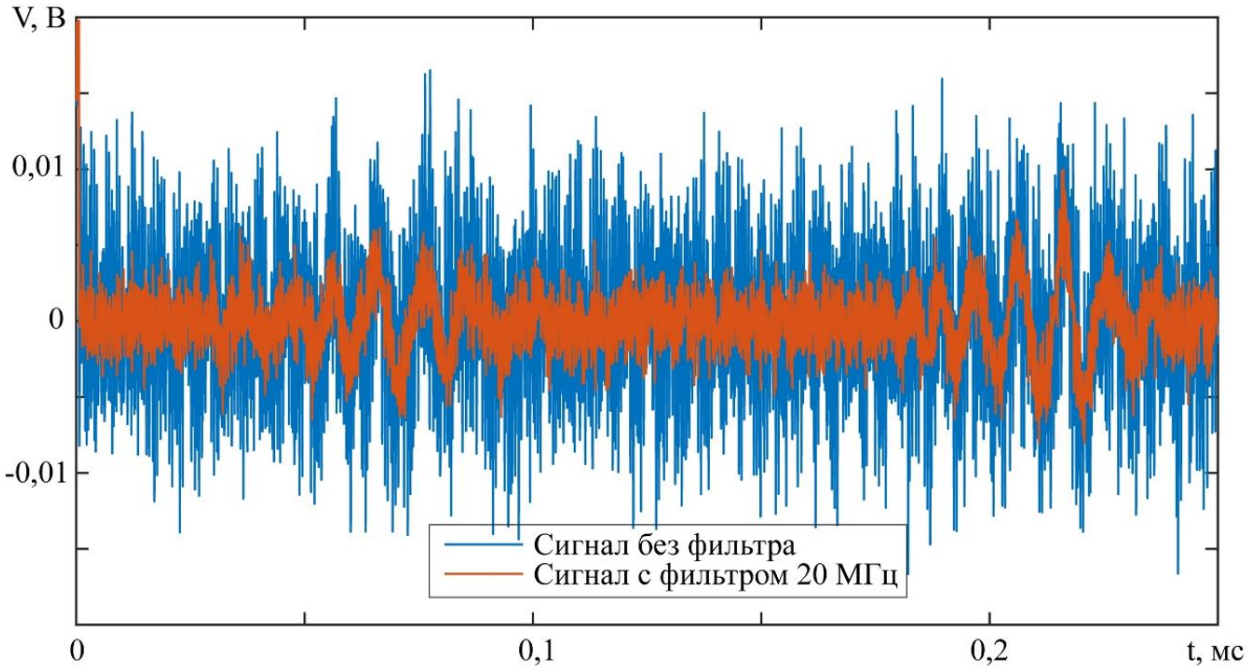

Рисунок 6 – Влияние применения аналогового фильтра на зашумлённость сигнала

В качестве посылаемого сигнала здесь и далее рассматривается нестационарный отклик пьезосенсора на упругие колебания возбуждаемые пьезопреобразователем, расположенным на расстоянии 125 мм от сенсора. В

качестве образца рассматривалась пластина из алюминия размерами 500×600×3 мм^3; управляющая электрическая посылка, подававшаяся на пьезоактуатор – три цикла синуса с частотой 100 кГц.

Главным плюсом данного метода обработки сигнала является то, что его работа не занимает никакого времени, так как считываемый сигнал очищается от высоких частот сразу же при поступлении. Однако, к сожалению, полоса пропускания аналогового фильтра ограничена аппаратно и не может быть изменена. Это значит, что с его помощью нет возможности избавиться от шумов с более низкой частотой.

3.2 Нахождение среднего по ансамблю сигналов

Под ансамблем сигналов подразумевается двумерное множество сигналов, где одному измерению соответствуют значения сигнала, различающиеся по времени, а второму – разные сигналы. Таким образом, если сигнал имеет длину в n считываемых значений, а всего считанных сигналов m, то получится система из $m \times n$ значений.

Если полезный сигнал имеет детерминированный по времени характер, то есть для разных считанных сигналов полезный сигнал имеет одинаковые значения в один момент времени, то существует возможность уменьшить амплитуду не детерминированных по времени шумов относительно полезного сигнала. Очевидно, что при m, стремящемся к бесконечности, значения энергии недетерминированных шумов в текущий момент времени будут стремиться к нормальному распределению с центром в нуле. Тогда и среднее значение их энергии будет стремиться к нулю. Таким образом, сгладить недетерминированные шумы можно при помощи следующей формулы, где $f^{(j)}(i)$ – это j-й считанный сигнал, а $f(i)$ – полезный сигнал в i-й момент времени [6]:

$$f(i) = \frac{1}{m}\sum_{j=1}^{m} f^{(j)}(i), i = \overline{1,n}, m \to \infty \quad (1)$$

Полученная формула называется нахождением среднего по ансамблю сигналов.

Конечно, на практике в силу ограниченности времени и погрешности вычислений целесообразно ограничить значение *m*.

Из преимуществ данного метода можно выделить простоту его реализации. Однако у него есть и серьёзные минусы.

Во-первых, может оказаться, что для нужной точности вычислений требуется лишком большое значение *m*, что может повлечь за собой слишком долгое время вычислений. Это связано с достаточно медленной сходимостью метода, выражаемой по формуле (2) [6]:

$$SNR_m = \sqrt{m} \times SNR_1, \quad (2)$$

где

SNR_m – соотношение сигнал/шум после усреднения m сигналов;

SNR_1 – фактически сигнал без усреднения.

Во-вторых, метод не способен избавиться от детерминированных по времени шумов.

Недостатки метода можно увидеть на рисунке 7. На нём изображено применение метода к уже аналогово обработанному сигналу.

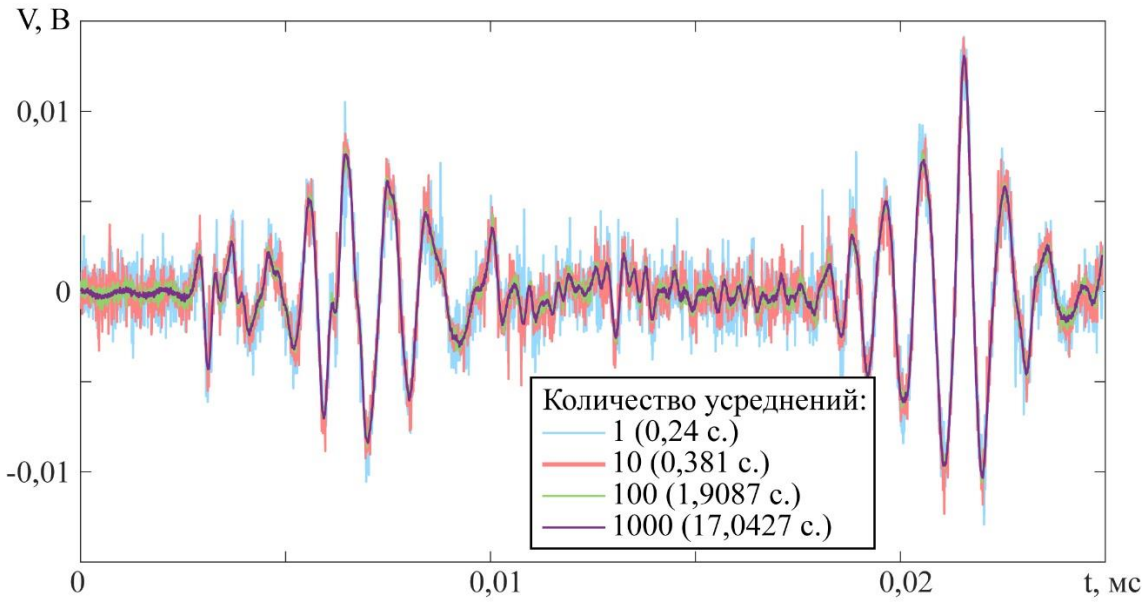

Рисунок 7 – Влияние количества усреднений на зашумленность сигнала

Методу понадобилось 1,9 секунд для достижения приемлемых гладкости и точности на участках с высокой амплитудой сигнала, в то время как для остальных участков понадобилось около 17 секунд, и результат можно улучшить. При этом графики содержат множество шумов, явно не соответствующих отправленной синусоиде. Однако в случае, если задача и не требует высоких точности и скорости выполнения, даже такой результат может оказаться вполне приемлемым.

Для реализации данного метода требуется немного модифицировать предыдущий код. Фрагмент данной реализации проиллюстрирован в приложении Б.

Из неочевидных особенностей реализации стоит отметить, что после первой настройки осциллографа последующие уже не требуются. Поэтому чтобы осциллограф не тратил большое количество времени на перенастройку на те же параметры, желательно оставить в запросе только те команды, которые инициируют считывание данных.

3.3 Вейвлет-преобразование

3.3.1 Подготовительные теоретические знания

В случае, если требуется избавиться от шумов в определённых частотных диапазонах, можно воспользоваться фильтрами с конечной импульсной характеристикой, реализуемых через преобразование Фурье. Однако более аккуратно с этой задачей справляется вейвлет-преобразование. В отличие от преобразования Фурье, оно осуществляет частотно-временной анализ [7, 8] и даёт возможность избавляться даже от шумов с меняющейся частотой, возможно даже пересекающейся частотный диапазон полезного сигнала, но в другие моменты времени [9].

Непрерывное прямое и обратное вейвлет-преобразование W от времени (сдвига) u и масштаба s функции $f \in L^2(R)$ с материнской вейвлет-функцией $\psi(t)$ в случае ограниченного частотно-временного диапазона имеет вид (3-4) [7, 8]:

$$Wf(s,\tau) = \frac{1}{\sqrt{s}} \int_{-t_0}^{+t_1} f(t)\psi(\frac{t-\tau}{s})dt \,, \tag{3}$$

$$f(t) = \frac{1}{C_\psi} \int_{t_0}^{t_1} \int_{s_0}^{s_1} Wf(s,\tau) \frac{1}{s^2\sqrt{s}} \psi(\frac{t-\tau}{s}) ds d\tau \tag{4}$$

Материнская вейвлет-функция $\psi(t)$ обязана иметь нулевое среднее значение (5):

$$\int_{-\infty}^{+\infty} \psi(t)dt = 0 \tag{5}$$

Нормирующая константа C_ψ зависит от взятого материнского вейвлета $\psi(t)$ и может быть вычислена по формуле (6):

$$C_\psi = 2\pi \int_{-\infty}^{+\infty} \frac{|\Psi(\zeta)|}{|\zeta|} d\zeta < +\infty, \qquad (6)$$

где

$\Psi(\zeta)$ – преобразование Фурье от вейвлета $\psi(t)$, причём $\Psi(0)=0$.

Очевидно, что для каждого из материнских вейвлетов $\psi(t)$ нормирующую константу C_ψ достаточно вычислить лишь единожды.

Вейвлет-преобразование имеет множество преимуществ на фоне метода нахождения среднего по ансамблю сигналов.

Во-первых, являясь развитием преобразования Фурье, вейвлет-преобразования сохраняют все его достоинства.

Во-вторых, вейвлет-преобразование может иметь хорошую локализацию по нужным частотам и времени, нивелировав при этом влияние шумов с иными частотами и временем.

В-третьих, вейвлет-преобразование позволяет не только подавлять нужные шумы, но и проводить частотно-временной анализ сигнала, что может быть особенно полезно, например, в анализе речи.

Далее в качестве сигнала рассматриваются два периода колебаний синусоиды с частотой 100 КГц, причём синусоида содержит множество разночастотных шумов. Данный сигнал был получен напрямую из генератора сигналов.

На рисунках 8 и 9 приведены прямые вейвлет-преобразования с разным захватом частот, способные предоставить полезную информацию о поведении функции в частотно-временной области, а на рисунках 10 и 11 – соответствующие обратные вейвлет-преобразования (синий) в сравнении с изначальным сигналом (чёрный), иллюстрирующие гибкость вейвлетов в

шумоподавлении.

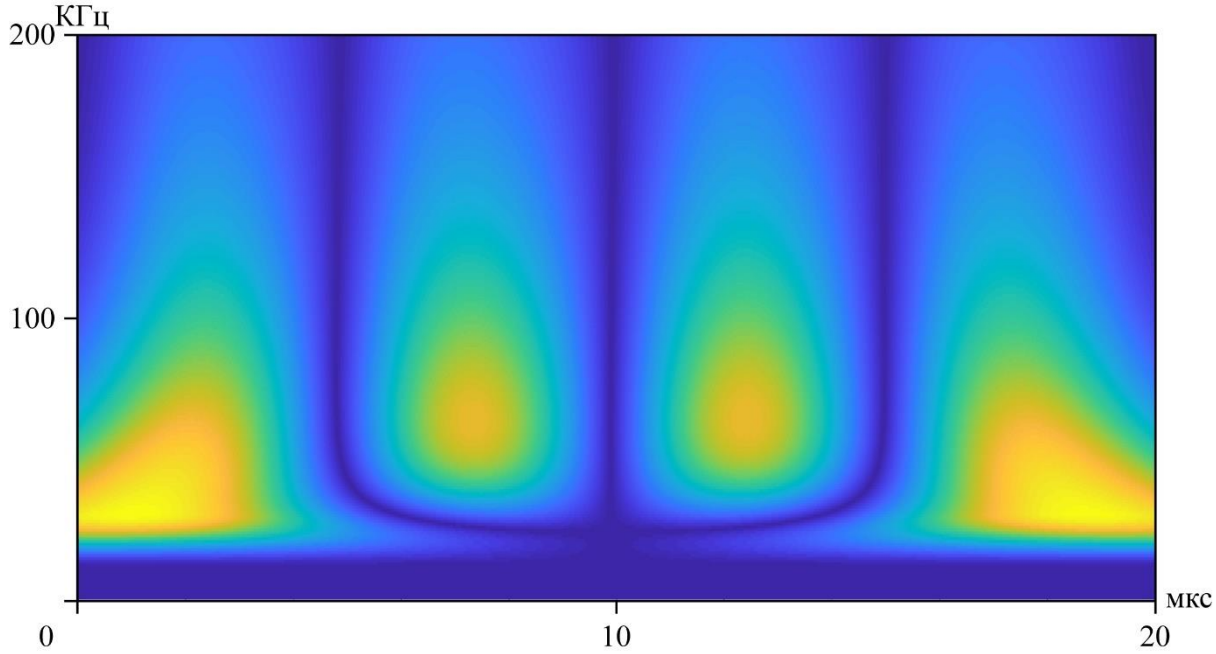

Рисунок 8 – Прямое вейвлет-преобразование до 200 КГц

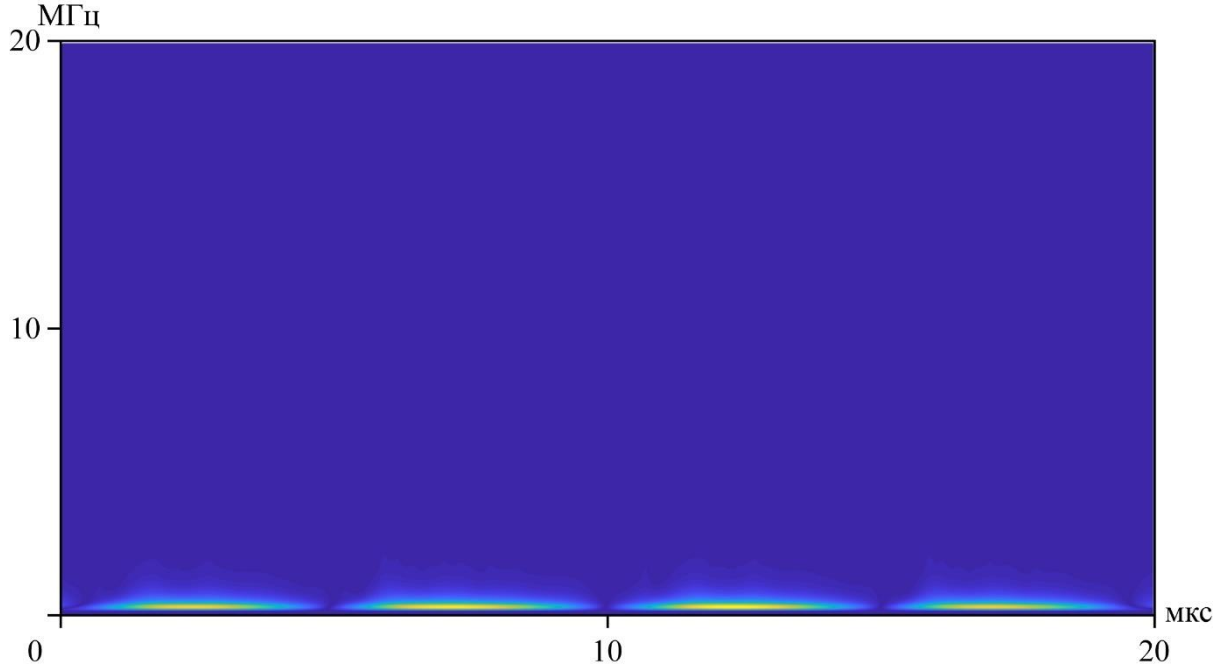

Рисунок 9 – Прямое вейвлет-преобразование до 20 МГц

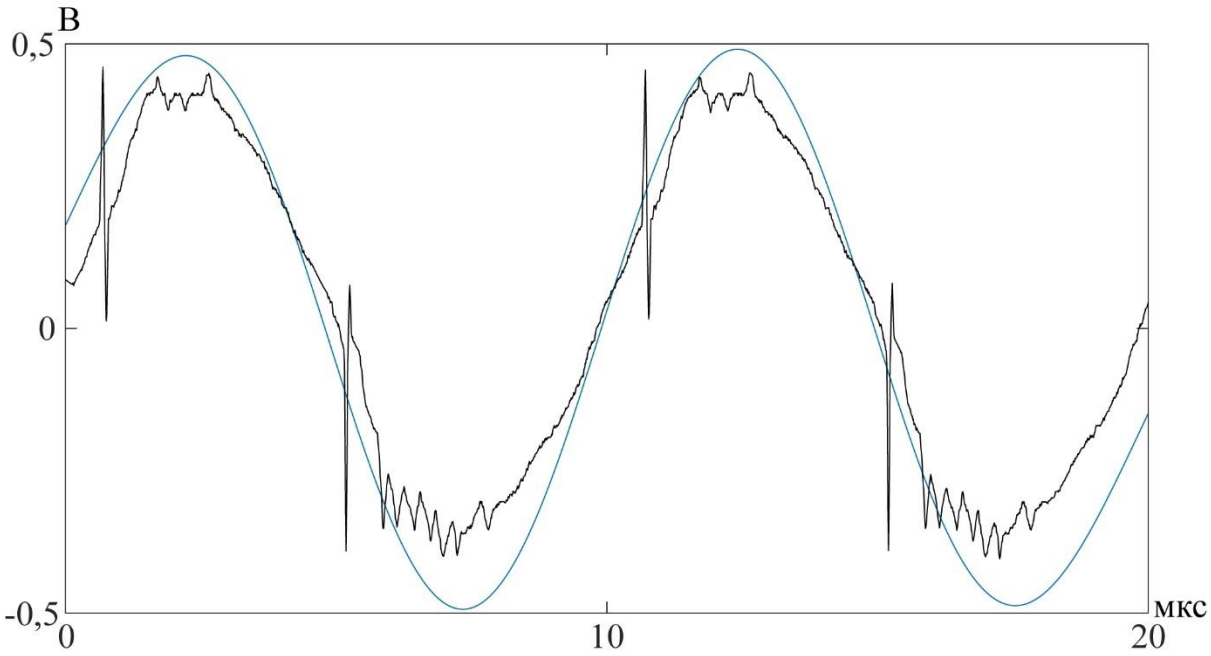

Рисунок 10 – Обратное вейвлет-преобразование до 200 КГц

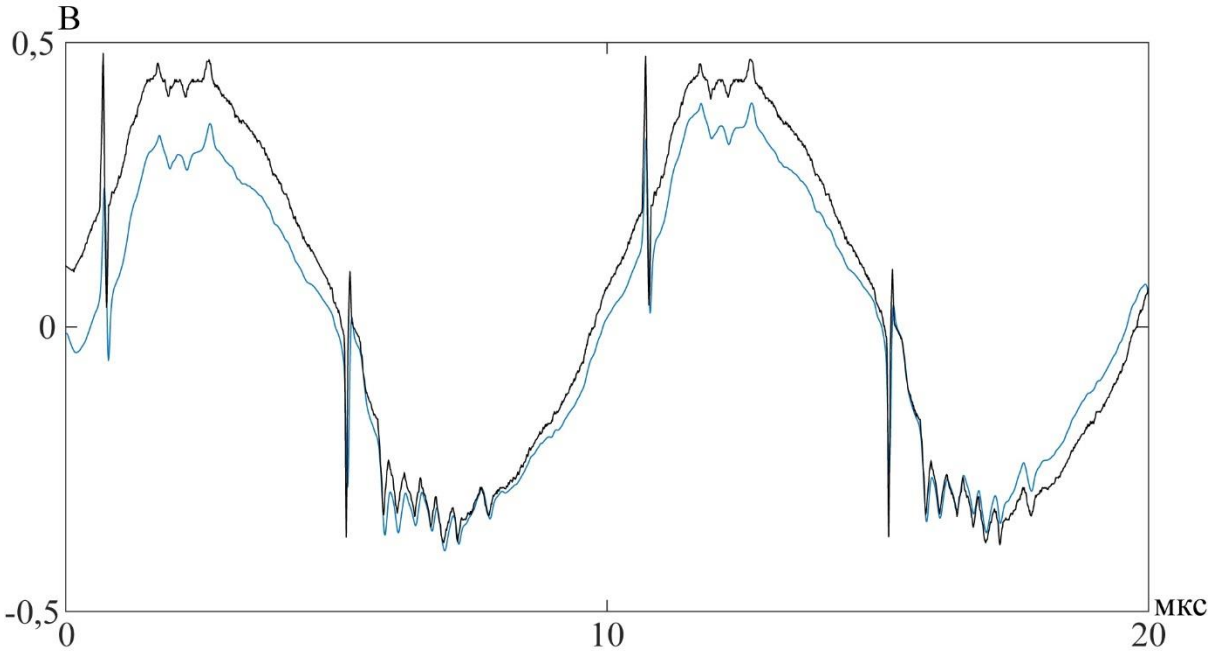

Рисунок 11 – Обратное вейвлет-преобразование до 20 МГц

Из серьёзных минусов вейвлет-преобразования можно выделить относительно высокую сложность его реализации.

На рисунке 12 проиллюстрирована обработка вейвлет-преобразованием того же сигнала, что подвергся обработке методом нахождения среднего по

ансамблю сигналов.

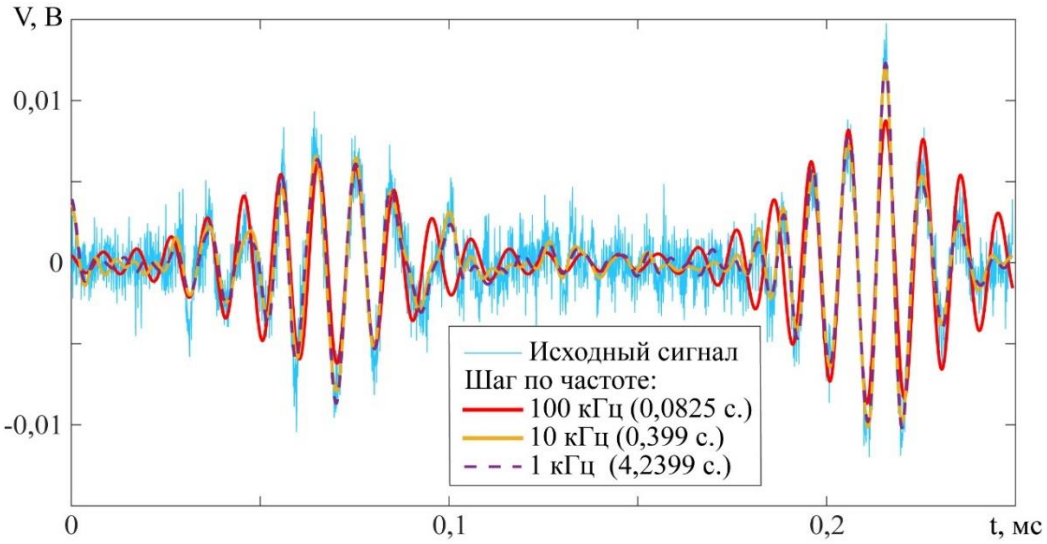

Рисунок 12 – Результаты применения прямого и обратного вейвлет-преобразования

Как можно заметить, скорость сходимости результата даже выросла (0.4 секунда против 1.9), а сам сигнал полностью избавился от высокочастотных шумов и представляет из себя ожидаемую синусоиду. Наличие сразу двух возмущений, к тому же имеющих больше 3-х циклов, объясняется интерференцией волновых сигналов, отражённых от боковых стенок экспериментального образа.

3.3.2 Вывод формул для программной реализации

Так как вейвлет-преобразование используется для обработки волнового сигнала, целесообразно рассматривать вейвлет-преобразование не в контексте искусственной масштабно-временной области, а в контексте привычной для волн частотно-временной области. Таким образом, в приведённых в предыдущем подразделе формулах желательно произвести замену переменной масштаба s на частотную переменную ν по формуле (7).

$$\nu = \frac{1}{s} \tag{7}$$

Таким образом, формулы прямого и обратного вейвлет-преобразований после замены переменных принимают вид (8-9).

$$Wf(\nu,\tau) = \int_{-t_0}^{+t_1} \sqrt{\nu}\, f(t)\psi(\nu(t-\tau))dt\,, \tag{8}$$

$$f(t) = \frac{1}{C_\psi} \int_{t_0}^{t_1} \int_{\nu_0}^{\nu_1} Wf(\nu,\tau)\sqrt{\nu}\psi(\nu(t-\tau))\nu^2 d\nu d\tau \tag{9}$$

Также в формулах (8-9) наблюдается один и тот же паттерн, который можно упростить, введу замену (10).

$$\psi_{\nu,\tau}(t) = \sqrt{\nu}\psi(\nu(t-\tau)) \tag{10}$$

Тогда формулы (8-9) принимают вид (11-12).

$$Wf(\nu,\tau) = \int_{-t_0}^{+t_1} f(t)\psi_{\nu,\tau}(t)dt\,, \tag{11}$$

$$f(t) = \frac{1}{C_\psi} \int_{t_0}^{t_1} \int_{\nu_0}^{\nu_1} Wf(\nu,\tau)\psi_{\nu,\tau}(t)\nu^2 d\nu d\tau \tag{12}$$

Так как нормирующую константу C_ψ достаточно вычислить лишь единожды для разных вейвлетов, то в случае, если изначальная функция уже была известна, константу можно легко вычислить, найдя соотношение максимума получившейся функции и максимума изначальной функции. В действительности изначальная функция $f(t)$ может быть задана искусственно,

без реального считывания значений, например как sin(*t*). Подобный метод вычисления нормирующей константы может оказаться гораздо проще, нежели вычисление интеграла от преобразования Фурье по формуле (6).

3.3.3 Процесс выбора вейвлета

Частотно-временное разрешение вейвлет-преобразования определяется частотно-временной протяжённостью функции $\psi_{\nu,\tau}(t)$, определённой по формуле (10) [7, 8].

В [7, 8] показано, что частотно-временное разрешение $\psi_{\nu,\tau}(t)$ представляется в частотно-временной плоскости (ν,t) прямоугольником Гейзенберга с центром в точке $(\nu\nu_0, \tau)$, ширина которого вдоль оси частоты $\sigma_\nu\nu$ и вдоль оси времени σ_τ/ν:

$$\nu_0 = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \nu \left|\Psi_{\nu,\tau}(\nu)\right|^2 \partial\nu\,, \tag{13}$$

$$\tau_0 = \int_{-\infty}^{+\infty} \tau \left|\psi_{\nu,\tau}(\tau)\right|^2 \partial\tau\,, \tag{14}$$

$$\sigma_\nu^2 = \frac{1}{2\pi}\int_{-\infty}^{+\infty} (\nu-\nu_0)^2 \left|\Psi_{\nu,\tau}(\nu)\right|^2 \partial\nu\,, \tag{15}$$

$$\sigma_\tau^2 = \int_{-\infty}^{+\infty} (\tau-\tau_0)^2 \left|\psi_{\nu,\tau}(\tau)\right|^2 \partial\tau \tag{16}$$

В формулах (13,15) $\Psi_{\nu,\tau}(t)$ – Фурье-символ вейвлет-атома $\psi_{\nu,\tau}(t)$.

Таким образом, на высоких частотах вейвлет-преобразование имеет хорошую локализацию по времени, но плохую по частоте, при этом на низких частотах ситуация изменяется – преобразование лучше локализовано в частотной области и плохо во временной [10]. Здесь важно заметить, что площадь прямоугольника Гейзенберга $\sigma_\nu\sigma_t$ не зависит от частоты ν и сдвига τ

и, по теореме неопределённости Гейзенберга. То есть при увеличении частотного разрешения вейвлета неминуемо уменьшается его временное разрешение, и наоборот.

Конечно, для разных задач существуют разные требования относительно частотно-временного разрешения. Например, в случае стационарной системы сигналов с близкими значениями частот частотное разрешение представляет собой гораздо больший интерес, нежели временной, а в случае единичного прерывистого сигнала гораздо более важным может оказаться временное разрешение.

Данную проблему решает наличие множества видов материнских вейвлетов. Так, например, пакет прикладных программ для решения задач технических вычислений MATLAB [11] предлагает на выбор 3 разных вейвлета, которые представлены вместе со своими частотно-временными разрешениями на рисунках 13 (“Bump”-вейвлет), 14 (Морле) и 15 (Морзе).

“Bump”-вейвлет имеет хорошее разрешение по времени и несколько хуже – по частоте.

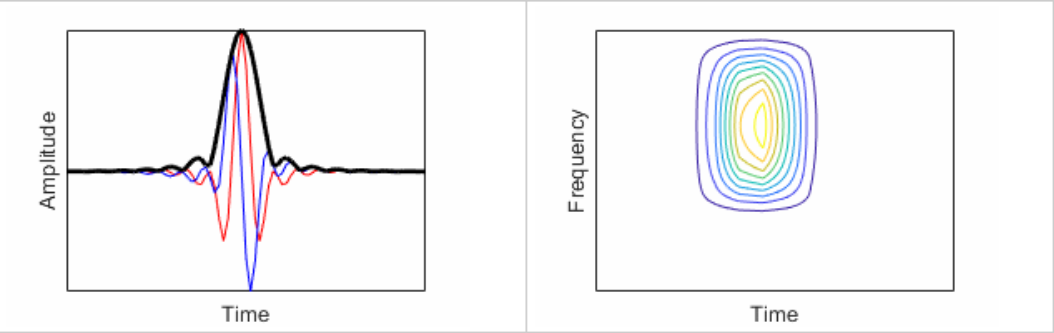

Рисунок 13 – “Bump”-вейвлет и его частотно-временное разрешение

Вейвлет Морле имеет одинаковое разрешение как по времени, так и по частоте.

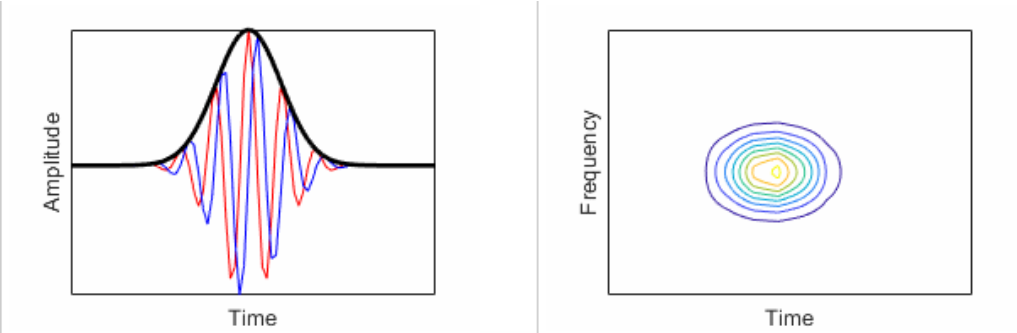

Рисунок 14 – Вейвлет Морле и его частотно-временное разрешение

В то же время, вейвлет Морзе и вовсе является универсальным и позволяет варьировать параметры, отвечающие за разрешение по времени и по частоте.

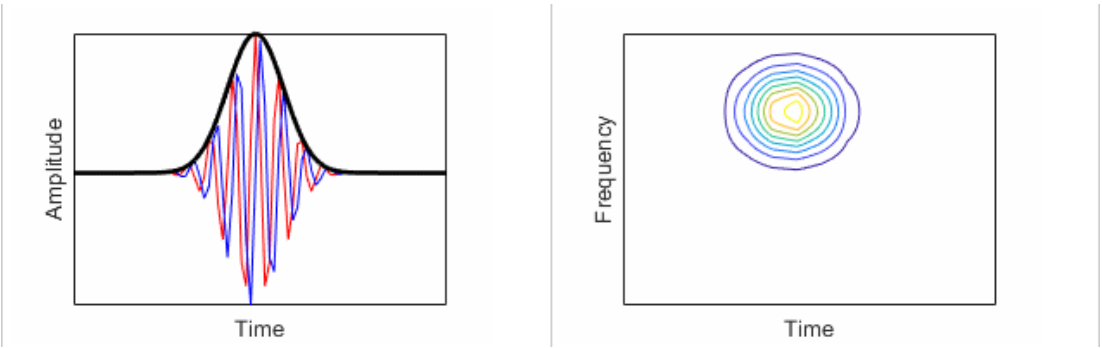

Рисунок 15 – Вейвлет Морзе и его частотно-временное разрешение

В рабочей программе были реализованы иные вейвлет-функции, а именно 4 первых представителя семейства Эрмитовых вейвлетов, вейвлет Пуассона и модифицированный вейлвет Морле, представленные соответственно в формулах (17-22).

$$\psi_{h_1}(t) = t\exp(-\frac{t^2}{2}), \tag{17}$$

$$\psi_{h_2}(t) = (1-t^2)\exp(-\frac{t^2}{2}), \tag{18}$$

$$\psi_{h_3}(t) = (t^3-3t)\exp(-\frac{t^2}{2}), \tag{19}$$

$$\psi_{h_4}(t) = (t^4 - 6t^2 + 3)\exp(-\frac{t^2}{2}), \quad (20)$$

$$\psi_p(t) = \frac{1-t^2}{\left(1+t^2\right)^2}, \quad (21)$$

$$\psi_{mm}(t) = \frac{\cos(2\pi t)}{\cosh(t)}, \quad (22)$$

С целью демонстрации важности выбора вейвлет-функции на рисунке 16 приведен результат прямого и обратного вейвлет-преобразования с каждой из вейвлет-функций. В качестве сигнала применена синусоида с частотой колебания 100 КГц с шумом неопределённого характера, а само преобразование вычислялось вплоть до частоты 200 КГц с шагом по времени 10 нс.

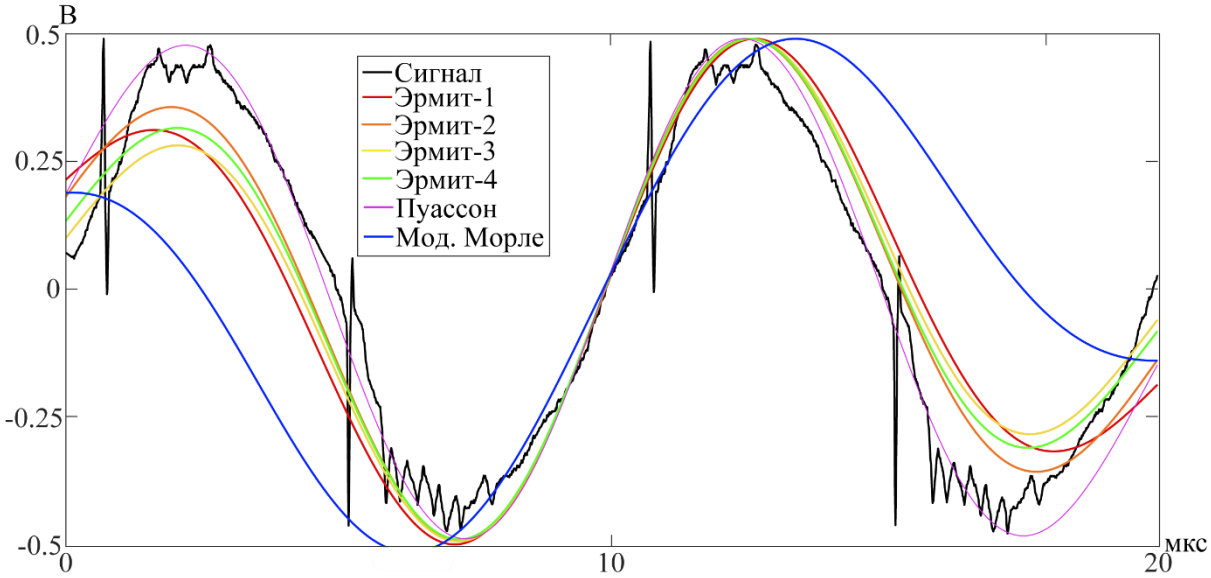

Рисунок 16 – сравнение качества обработки одного сигнала разными вейвлетами

Даже при визуальном анализе приведённого графика можно обнаружить более высокий уровень точности при обработке данного сигнала вейвлетом Пуассона, более низкий – модифицированным вейвлетом Морле и примерно похожий – вейвлетами Эрмита. И более того, данная тенденция сохраняется

при сравнении среднего значения абсолютной ошибки как при сравнении обратных вейвлетов с сигналом, так и при сравнении обратных вейвлетов с синусоидой с частотой 100 КГц. Так, например, для Эрмитов средняя абсолютная ошибка к синусоиде равна соответственно 0,12, 0,09, 0,11, и 0,1 вольт, для Пуассона – всего 0,06 вольт, а для Морле – 0,26 вольт.

Таким образом, при выделении полезного сигнала в районе частоты 100 КГц, наиболее результативным оказался вейвлет Пуассона. Впрочем, для других частот и иного сигнала ситуация также может оказаться иной.

3.3.4 Особенности программной реализации вейвлет-преобразования

Вейвлет-преобразование представляет собой достаточно сложные вычисления, которые при решении «в лоб» могут занять гораздо больше времени, нежели при деликатном подходе. Поэтому данный раздел описывает в первую очередь особенности алгоритма, способные значительно улучшить эффективность работы кода.

Во-первых, в случае, если данные сигнала уже даны и нет возможности изменить шаг по времени для семплов, не стоит считывать каждый первый семпл, если не требуется такая точность измерения. Количество считываемых семплов имеет огромное влияние на скорость вычисления вейвлет-преобразования, ведь для вычисления вейвлет-функция $\psi(\nu(t-\tau))$ требуется порядка $t^2\nu$ значений. Поэтому, если есть возможность, лучше пропускать семплы, порождающие излишнюю точность в вычислениях значений по времени. Программная реализация изменения временных переменных с учётом пропуска семплов описана в приложении В, а программная реализация пропуска ненужных семплов при считывании данных из файла – в приложении Г.

Во-вторых, в действительности всё же не обязательно высчитывать $t^2\nu$ значений для вычисления вейвлет-функций. Хотя $\psi(\nu(t-\tau))$ и имеет внутри себя

целых 3 переменных, но переменная τ отражает лишь сдвиг переменной *t*. Данные переменные на программном уровне могут выражаться одним и тем же массивом, ведь они имеют одно начало, один шаг и один размер. То есть набор из значений этих переменных можно объединить в одну переменную. Так как $t \in [t_0; t_1]$, $\tau \in [t_0; t_1]$, то $(t-\tau) \in [t_0-t_1; t_1-t_0]$, а размер получившегося массива будет равен $((t_1-t_0)-(t_0-t_1))=2(t_1-t_0)$. То есть вместо вычисления и хранения 3-х мерного массива размера $(t_1-t_0)^2(f_1-f_0)$, для вейвлет-функции достаточно получить 2-мерный массив размера $2(t_1-t_0)(\nu_1-\nu_0)$, что в самом худшем случае улучшает процесс вычисления в $t/2$ раз.

В дальнейшем, при использовании вейвлет-функции, при взятии вычисленного значения достаточно просто брать поправку на сдвиг.

Фрагмент программного кода, определяющий массив вейвлет-функции, описан в приложении Д.

В-третьих, вейвлет-преобразование способно достаточно качественно вычисляться параллельно. Поэтому в работе было использовано средство многопоточности OpenMP [12]. Пример его использования в программном коде при вычислении прямого вейвлет-преобразования изображён в приложении Е.

Стоит заметить, что для работы OpenMP требуется сперва включить его поддержку компилятором (/openmp) и импортировать заголовочный файл <omp.h>.

В-четвёртых, при вычислении интеграла требуется найти компромисс между скоростью и точностью при выборе численного метода решения. В работе использовалась реализация формулы Котеса [13], программная реализация которой приведёа в приложении Ж.

Математически же формула Котеса на отрезке $[a;b]$ описана в формуле (23).

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{\frac{N}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\frac{N}{2}} f(x_{2j-1}) + f(x_N) \right], \qquad (23)$$

где

$h = \frac{b-a}{h}$ – величина шага интегрирования;

$x_j = a + jh$ – узлы, определяющие границы отрезков интегрирования.

Некоторые особенности не были описаны в виду их несущественного влияния на скорость работы. И, конечно же, существует ещё большое количество методов оптимизации кода. Однако, следуя указанным решениям, уже удалось ускорить время работы кода в несколько раз, достигнув приемлемого результата.

ЗАКЛЮЧЕНИЕ

В результате работы был написан код для считывания данных с цифрового осциллографа модели VDS6102A и обработки полученного сигнала, на основе которого при помощи пошагового руководства от Microsoft была создана рабочая библиотека динамической компоновки [14]. Данный результат работы выложен в общий доступ на веб-сервис для хостинга IT-проектов GitHub в виде файлов с кодом и собственно библиотекой: https://github.com/Owouwun/WaveletTransform/tree/final_result. Работа самой библиотеки была успешно проверена на языке программирования C#, а файл с кодом этой проверки хранится в том же репозитории.

Также работа содержит множество авторских алгоритмических решений, не имевшихся ранее в свободном доступе. Некоторые из данных решений способны значительно помочь программистам, желающим реализовать подобный проект самостоятельно.

В дальнейшем текущую работу предполагается развить следующим образом: добавить оконообразные функции-фильтры при вычислении прямого вейвлет-преобразования с целью более точного выделения частотно-временного спектра полезного сигнала; расширить возможности вейвлет-функций, включая реализацию параметрических вейвлетов, способных изменять своё частотно-временное разрешение при помощи изменения соответствующего параметра; реализовать оконное преобразование Фурье и провести его сравнение с вейвлет-преобразованием.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Giurgiutiu, V. Structural health monitoring with piezoelectric wafer active sensors : book / V. Giurgiutiu – Columbia: Academic Press, 2014. – 1102 p. – ISBN 9780124186910.

2. Столбов, М.Б. Основы анализа и обработки речевых сигналов : учебное пособие / М.Б. Столбов. – СПб. : НИУ ИТМО, 2021. – 101 с. – URL: https://books.ifmo.ru/file/pdf/2687.pdf (дата обращения: 10.06.2023).

3. NI-VISA Overview : официальный сайт. – Остин, США. – URL: https://www.ni.com/ru-ru/support/documentation/supplemental/06/ni-visa-overview.html (дата обращения: 31.03.2023).

4. OWON : официальный сайт. – Чжанчжоу, Китай. – URL: https://www.owon.com.hk/download.asp?category=Please%20select%20product%20category&series=&model=&SortTag=&seek=&curpage=16 (дата обращения: 31.03.2023)

5. Шаманов А.П. Системы счисления и представление чисел в ЭВМ : учебное пособие / А.П. Шаманов. — Екатеринбург: Издво Урал. ун-та, 2016. — 52 с. – ISBN 978-5-7996-1719-6

6. Umer, H. Reducing noise by repetition: Introduction to signal averaging / H. Umer, A. Sabieh // European journal of Physics. — 2010. — Vol 31, № 3. – P. 453-465.

7. Малла, С. Вэйвлеты в обработке сигналов : учебное пособие / С. Малла – М.: Мир, 2005. – 338 с., ил. – ISBN 5-03-003691-1

8. Блаттер, К. Вейвлет-анализ. Основы теории. : учебное пособие / К. Блаттер – М.: Техносфера, 2004. – 276 с. – ISBN 5-94836-033-4

9. Yang, Y. Processing seismic ambient noise data with the continuous wavelet transform to obtain reliable empirical Green's functions / Y. Yang, C. Liu, C. Langston // Geophysical Journal International. — 2020. — Vol 222, № 2. — P. 1224–1235.

10. Сравнение частотно-временных преобразований: Фурье-анализ, вейвлеты и банки фильтров на основе фазового преобразования / М.И. Вашкевич, И.С. Азаров // Российское научно-техническое общество радиотехники. – 2020. – №2. – URL: https://libeldoc.bsuir.by/handle/123456789/40476 – Дата публикации: 13.10.2020.

11. MathWorks : официальный сайт. – Натик, США. – URL: https://www.mathworks.com/help/wavelet/gs/choose-a-wavelet.html (дата обращения: 26.04.2023).

12. Microsoft : официальный сайт. – Редмонд, США. – URL: https://learn.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170 (дата обращения: 10.05.2023)

13. Бахвалов, Н.С. Численные методы : учебное пособие / Н.С. Бахвалов, Н.П. Жидков, Г.М. Кобельков. — М.: БИНОМ, Лаборатория знаний, 2006. – 636 с. – ISBN 9785947748154

14. Microsoft : официальный сайт. – Редмонт, США. – URL: https://learn.microsoft.com/ru-ru/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp?view=msvc-170 (дата обращения: 10.05.2023)

ПРИЛОЖЕНИЕ А

Хранение и обработка допустимых значений вертикального масштабирования

```
enum VERTICAL_SCALE {
    VS2mV = 2,
    VS5mV = 5,
    VS10mV = 10,
    VS20mV = 20,
    VS50mV = 50,
    VS100mV = 100,
    VS200mV = 200,
    VS500mV = 500,
    VS1V = 1000,
    VS2V = 2000,
    VS5V = 5000
};

string vertScale_to_string(VERTICAL_SCALE sc) {
    switch (sc) {
        case VS2mV:
            return "2mV";
        case VS5mV:
            return "5mV";
        case VS10mV:
            return "10mV";
        case VS20mV:
            return "20mV";
        case VS50mV:
            return "50mV";
        case VS100mV:
            return "100mV";
        case VS200mV:
            return "200mV";
        case VS500mV:
            return "500mV";
        case VS1V:
            return "1V";
        case VS2V:
            return "2V";
        case VS5V:
            return "5V";
        default:
            return "";
    }
}
```

Рисунок А.1 – Программный код хранения и обработки допустимых значений вертикального масштабирования

ПРИЛОЖЕНИЕ Б

Нахождение среднего по ансамблю сигналов

```
for (int k = 0; k < reading_number; k++) {
    if (k == 0)
        status = viWrite(instr, (ViBuf)stringinput, (ViUInt32)strlen(stringinput), &writeCount);
    else
        status = viWrite(instr, (ViBuf)":WAV:BEG CH1;\n:WAV:FETC?;\n:WAV:END;\n", (ViUInt32)37, &writeCount);
...
}
double div = 1. / reading_number;
for (int j = 0; j < sample_size; j++)
    result_arr[j] *= div;
```

Рисунок Б.1 – Фрагмент программного кода для нахождения среднего по ансамблю сигналов

ПРИЛОЖЕНИЕ В

Преобразование значений переменных времени с учётом пропуска семплов

```
void sizing_toDouble(
    int t_input_startIndex,
    int t_input_endIndex,
    double t_input_step,
    int t_input_size,
    int t_sizing,
    double* t_sizing_start,
    double* t_sizing_end,
    double* t_sizing_step,
    int* t_sizing_size
) {
    *t_sizing_start = t_input_startIndex * t_input_step;
    *t_sizing_end = t_input_endIndex * t_input_step;
    *t_sizing_step = t_input_step * t_sizing;
    *t_sizing_size = (t_input_endIndex - t_input_startIndex) / t_sizing;
    return;
}
```

Рисунок В.1 – Программный код преобразования значений переменных времени с учётом пропуска семплов

ПРИЛОЖЕНИЕ Г

Обработка файла со считыванием данных с учётом пропуска семплов

```
double* X = new double[t_size];
ifstream fin(inputFileName);
double gbg;
for (int i = 0; i < t_start_index; i++)
    fin >> gbg;
int read_size = t_end_index - t_start_index;
double divsizing = 1. / t_sizing;
for (int i = 0; i < read_size; i++)
    if (i % t_sizing == 0)
        fin >> X[int(i * divsizing)];
    else fin >> gbg;
fin.close();
int t_current_start_index = t_start / t_step; //Номер начального семпла после sizing
int t_current_end_index = t_current_start_index + t_size; //Номер последнего семпла после sizing
```

Рисунок Г.1 – Программный код обработки файла со считыванием данных с учётом пропуска семплов

ПРИЛОЖЕНИЕ Д

Построение вейвлет-функции

```
void make_waveletFunction_equalStep(
    double t_start, double t_step, int t_size,
    double f_start, double f_step, int f_size,
    wavelets wavelet,
    double* res) {
        switch (wavelet) {
        case modified_morlet:
            for (int i = 0; i < f_size; i++) {
                double divsqrt = 1 / sqrt(f_start + i * f_step);
                for (int j = 0; j < 2 * t_size; j++) {
                    double t_taus = (j - t_size) * t_step * (f_start + i * f_step);
                    res[i * 2*t_size + j] = modified_morlet_wavelet(t_taus) * divsqrt;
                }
            }
            break;
            ...
        }
    }
```

Рисунок Д.1 – Фрагмент программного кода построения вейвлет-функции

ПРИЛОЖЕНИЕ Е

Пример использования многопоточности

```
int nt = omp_get_max_threads();
omp_set_dynamic(0);
omp_set_num_threads(nt);
#pragma omp parallel num_threads(nt)
{
    int tn = omp_get_thread_num();
    complex<double>* f = new complex<double>[t_size];
    for (int i = tn; i < f_size; i += nt)
        for (int j = t_current_start_index; j < t_current_end_index; j++) {
            int temp = t_size - j;
            for (int k = t_current_start_index; k < t_current_end_index; k++)
                f[k - t_current_start_index] = X[k - t_current_start_index] * psitaus[i * 2 * t_size + temp + k];
            res[i][j - t_current_start_index] = kotes(f, t_step, t_size).real();
        }
    delete[]f;
}
delete[]X;
```

Рисунок Е.1 – Фрагмент программного кода вычисления прямого вейвлет-преобразования с использованием многопоточности

ПРИЛОЖЕНИЕ Ж

Реализация формулы Котеса

```
1  double kotes(double* f, double step, int size) {
2      double res = f[0] + f[size - 1];
3
4      for (int k = 1; k < size - 2; k += 2) {
5          res += 2. * f[k];
6          res += 4. * f[k + 1];
7      }
8
9      if (res == res) // Проверка на nan
10         return res * step * 0.33333333333333333;
11     else
12         return 0;
13 }
```

Рисунок Ж.1 – Программный код реализации формулы Котеса

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

Кубанский Государственный университет

о результатах проверки текстового документа
на наличие заимствований

ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Кузнецов И Д
Самоцитирование рассчитано для: Кузнецов И Д
Название работы: РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СИСТЕМЫ ИЗМЕРЕНИЙ И ОБРАБОТКИ СИГНАЛОВ С ИСПОЛЬЗОВАНИЕМ ЦИФРОВЫХ USB-ОСЦИЛЛОГРАФОВ
Тип работы: Выпускная квалификационная работа
Подразделение: ФКТиПМ, кафедра прикладной математики

РЕЗУЛЬТАТЫ

СОВПАДЕНИЯ	1.24%
ОРИГИНАЛЬНОСТЬ	90.85%
ЦИТИРОВАНИЯ	7.91%
САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 15.06.2023

Структура документа: Проверенные разделы: основная часть с.2, 4-33, библиография с.34-35

Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс*; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по Интернету (EnRu); Переводные заимствования издательства Wiley ; eLIBRARY.RU; СПС ГАРАНТ: аналитика; СПС ГАРАНТ: нормативно-правовая документация; Модуль поиска "КубГУ"; Медицина; Диссертации НББ; Коллекция НБУ; Перефразирования по eLIBRARY.RU; Перефразирования по СПС ГАРАНТ: аналитика*; Перефразирования по Интернету; Перефразирования по Интернету (EN); Перефразирования по коллекции издательства Wiley ; Патенты СССР, РФ, СНГ; СМИ России и СНГ; Шаблонные фразы; Кольцо вузов; Издательство Wiley; Переводные заимствования

Работу проверил: Троценко Екатерина Сергеевна
ФИО проверяющего

Дата подписи: 15.06.23 г

Подпись проверяющего

Чтобы убедиться в подлинности справки, используйте QR-код, который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.