ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

**Федеральное государственное образовательное бюджетное учреждение**

**высшего профессионального образования**

**«Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича»**

Факультет Информационных технологий и программной инженерии

Кафедра Программной инженерии и вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине:

**«Программирование»**

тема: Анализ сигнала на выходе электрической цепи

Передаточная характеристика – 23 вариант

Входной сигнал – 11 вариант

Выполнил студент группы ИКПИ-41:

Синельников A.И. \_\_\_\_\_\_\_\_

Дата выполнения: «29» Май

Проверил:

Хазиев Н.Н. \_\_\_\_\_\_\_\_

Санкт-Петербург

2025

Оглавление

[Аннотация 4](#_Toc199165650)

[Задание к курсовой работе 5](#_Toc199165651)

[1 Общие сведения 6](#_Toc199165652)

[1.1 Обозначение и наименование программы 6](#_Toc199165653)

[1.2 Языки разработки: 6](#_Toc199165654)

[2 Функциональное назначение 6](#_Toc199165655)

[2.1 Решаемые задачи 6](#_Toc199165656)

[2.2 Назначение: 7](#_Toc199165657)

[2.3 Ограничения в функциональности: 7](#_Toc199165658)

[3 Таблица идентификаторов 9](#_Toc199165659)

[4 Описание логической структуры 10](#_Toc199165660)

[4.1 Алгоритм программы 10](#_Toc199165661)

[4.2 Описание функций 10](#_Toc199165662)

[4.3 Используемые методы 11](#_Toc199165663)

[4.4 Связи программы с другими программами 12](#_Toc199165664)

[5 Используемые технические средства 13](#_Toc199165665)

[6 Вызов и загрузка 14](#_Toc199165666)

[6.1 Способ вызова программы с соответствующего носителя данных 14](#_Toc199165667)

[6.2 Входные точки в программу 14](#_Toc199165668)

[7 Входные данные 15](#_Toc199165669)

[7.1 Характер и организация входных данных 15](#_Toc199165670)

[8 Выходные данные 17](#_Toc199165671)

[8.1 Характер и организация выходных данных 17](#_Toc199165672)

[8.2 Контрольный расчёт 18](#_Toc199165673)

[8.3 Формат и кодирование выходных данных 19](#_Toc199165674)

[9 Структура кода 21](#_Toc199165675)

[10 Заключение 23](#_Toc199165676)

[11 Список используемой литературы 25](#_Toc199165677)

[12 Сокращения 26](#_Toc199165678)

[13 Приложения 28](#_Toc199165679)

[13.1 Приложение 1 28](#_Toc199165680)

[a main.c 28](#_Toc199165681)

[b app.c 28](#_Toc199165682)

[c funct.c 29](#_Toc199165683)

[d Заголовочные файлы 30](#_Toc199165684)

[13.2 Приложение 2 31](#_Toc199165685)

[13.3 Приложение 3 34](#_Toc199165686)

Аннотация

Программный продукт "Анализатор сигналов электрических цепей" представляет собой консольное приложение, разработанное для автоматизированного анализа характеристик линейных электрических цепей. Программа выполняет численное моделирование прохождения сигнала через электрическую цепь с заданными параметрами.

**Основные функциональные возможности:**

* расчет выходного сигнала по известному входному воздействию;
* определение временных параметров сигнала (длительность импульса, время нарастания);
* оценка точности вычислений с заданной погрешностью (до 1%).

**Технические особенности реализации:**

* язык разработки: С (ядро вычислений) + Bash (интерфейс);
* платформа: ОС Linux (Ubuntu);
* графическая подсистема: wxMaxima;
* способ взаимодействия: командная строка (Command line interface).

**Ключевые алгоритмы:**

* дискретизация временной оси;
* кусочно-линейная аппроксимация;
* итерационный метод уточнения параметров;
* автоматическое построение графиков.

Программа разработана в соответствии с требованиями ЕСПД (ГОСТ 19.402-78) и предназначена для использования в учебном процессе и инженерных расчетах. Особенностью решения является сочетание высокой точности вычислений (использование 32-битной арифметики с плавающей точкой IEEE 754) с простотой использования через командный интерфейс.

Объем исходного кода: ~500 строк (без учета зависимостей)  
Требования к аппаратному обеспечению: процессор x86-64, 512 МБ ОЗУ, 10 МБ дискового пространства.

Задание к курсовой работе

Работа посвящена решению задач машинного анализа электрических цепей.

В курсовой работе необходимо для заданной электрической цепи по известному входному сигналу определить выходной сигнал для N равностоящих моментов времени, а затем определить некоторые его характеристики с погрешностью не более 1%. Варианты параметров входного сигнала (код А) и передаточной характеристики (код Б) aэлектрической цепи приведены в приложении. Номер варианта определяется преподавателем индивидуально для каждого студента.

Таблица 0.1:

**Таблица входных значений** **Uvx**

|  |  |
| --- | --- |
| **Входной сигнал** | **Рабочий набор** |
|  |  |

Таблица 0.2:

**Таблица входных значений** **Uvx**

|  |  |
| --- | --- |
| **Выходной сигнал** | **Рабочий набор** |
|  |  |

* + - 1. в ходе работы необходимо:
* произвести расчет входного и выходного сигнала в контрольных точках, используя при этом математический пакет wxmaxima;
* написать текст программы на языке Си;
* произвести запись полученных результатов в файлы данных;
* используя математический пакет Wxmaxima (электронные таблицы), построить графики зависимости напряжений входных и выходных сигналов от времени;
* объединить программу на Си и Wxmaxima (LibraOffice.Calc), вызов отчета с помощью скрипта на Bash.

# Общие сведения

## Обозначение и наименование программы

Полное наименование: «Анализ сигнала на выходе электрической цепи».

Краткое наименование: АСВЭЦ, программа.

Язык: русский.

## Обозначение и наименование программы

Для корректной работы программа требует установленную руси-фицированную версию операционной системы Ubuntu Linux.

Также необходима установка стороннего ПО wxMaxima. Установить его можно командой в терминале:

* + - 1. **sudo apt-get install wxmaxima;**

Компилятор gcc обычно является встроенным в ОС Linux, однако при его отсутствии его можно установить последовательностью команд:

* + - 1. **sudo apt update;**
      2. **sudo apt install build-essential;**

## Языки разработки:

Основная логика программы реализована на языке Си, который обеспечивает базовую функциональность.

Взаимодействие с пользователем организовано через меню на Bash-скриптах. Эти скрипты также запускают wxMaxima-скрипт, сохранённый в текстовом файле с расширением .mac.

# Функциональное назначение

## Решаемые задачи

Программа предназначена для численного и графического анализа сигналов в электрических цепях. Она решает следующие задачи:

**Моделирование сигналов:**

* + - 1. расчёт функции входного напряжения Uвх(t), заданной в аналитической форме;
      2. вычисление выходного напряжения Uвых(t), используя кусочно-линейную передаточную характеристику.

**Графическая визуализация:**

* + - 1. построение графиков Uвх(t) и Uвых(t) с помощью wxMaxima;
      2. экспорт полученных данных в форматы, совместимые с другими пакетами (например, GNU Plot или LibreOffice Calc).

## Назначение:

Программа **АСВЭЦ** предназначена для работы в среде **Ubuntu Linux**. Программа АСВЭЦ предназначена для работы в среде Ubuntu Linux.

Основное применение — образовательное: визуализация работы электрических цепей и сравнение различных численных методов анализа сигналов.

Также программа пригодна для инженерных целей — быстрой оценки параметров цепей с нелинейными элементами.

Проверка аналитических решений гарантирует точность вычислений. Дополнительно, программа поддерживает автоматизацию обработки результатов для различных наборов параметров.

## Ограничения в функциональности:

* + - 1. **совместимость с ОС:** программа работает только в Ubuntu Linux и не поддерживает Windows.
      2. ограничения по входным данным:
* временной диапазон жёстко зафиксирован: t ∈ [0, 0,00001] (можно изменить вручную в коде);
* параметры цепи заданы для варианта №23/11;
* максимальное количество точек **Nmax = 10 000** (определено размером массива).
  + - 1. численные ограничения:
* используется тип данных float, что ограничивает точность;
* итерационные методы могут не сойтись при слишком малом значении **eps**.
  + - 1. системные требования:
* только ОС Ubuntu Linux;
* наличие wxMaxima (для визуализации) и gcc (для компиляции) обязательно.
  + - 1. ограничения пользовательского интерфейса:
* отсутствие графического интерфейса — работа осуществляется через консоль;
* низкая устойчивость к некорректному вводу.
  + - 1. **Примечание:** при необходимости использовать программу для других параметров, требуется вручную изменить настройки в исходном файле funct.c.

# Таблица идентификаторов

Таблица 3.1:

**Таблица идентификаторов**

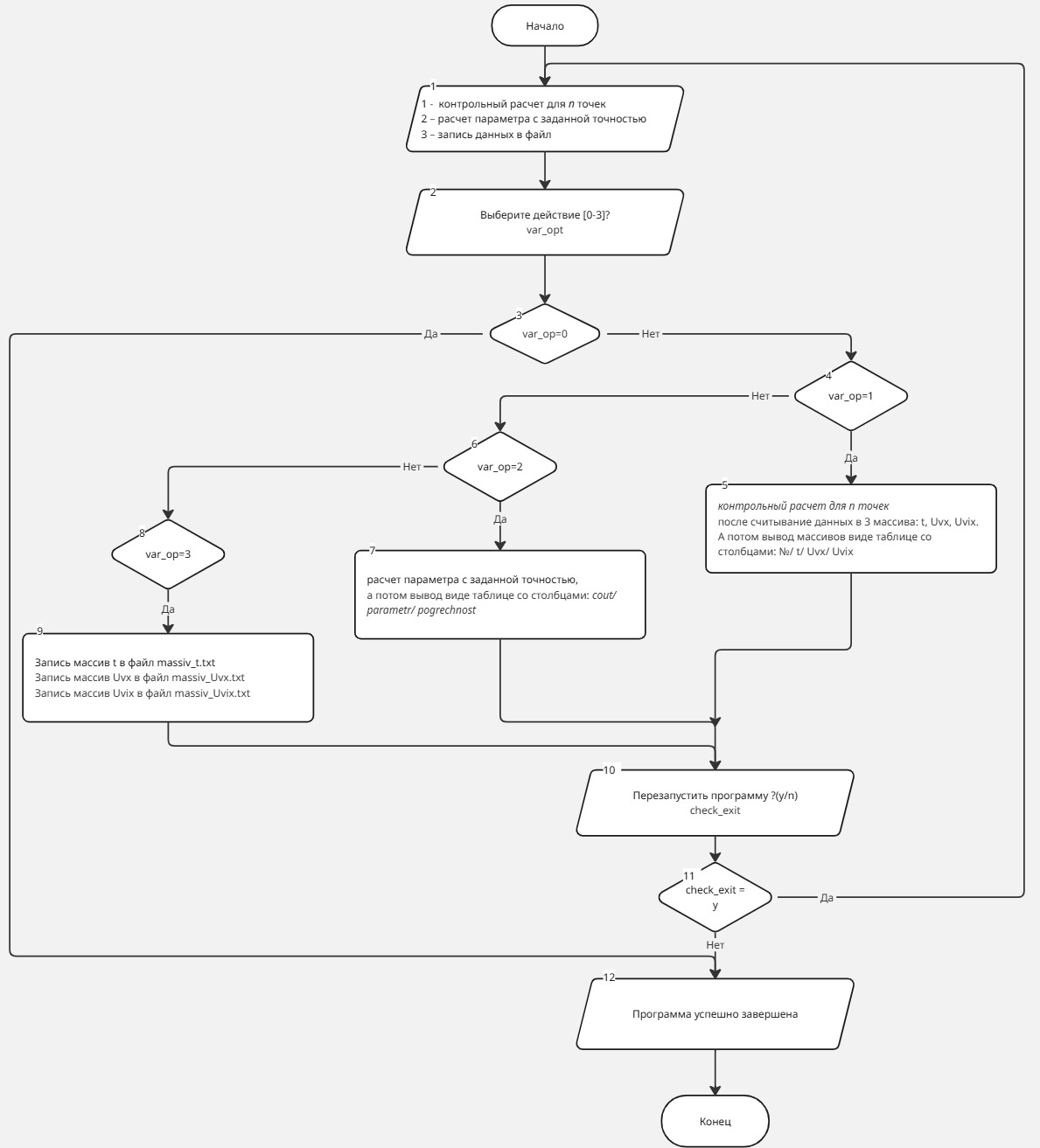
|  |  |  |
| --- | --- | --- |
| **Переменная** | **Тип** | **Описание** |
| **n** | int | Количество точек разбиения |
| **eps** | float | Допустимая погрешность (для приближённого метода) |
| **U** | float | Коэффициенты линейных участков функции Uvx |
| **T** | float | Конечное время |
| **U1,U2,Uvx1,Uvx2** | float | Пороговые значения для функции Uvix |
| **t[N]** | float | Массив временных точек |
| **Uvx[N]** | float | Массив значений входной функции |
| **Uvix[N]** | float | Массив значений выходной функции |

# Описание логической структуры

## Алгоритм программы

Рисунок 4.1.1:

**Рисунок блок схемы**



## Описание функций

Таблица 4.2.1:

**Таблица функций**

| **Функция** | **Назначение** |
| --- | --- |
| **clear\_line()** | Очищает строку в терминале и подаёт звуковой сигнал |
| **is\_number()** | Проверяет, является ли ввод числом (соответствует ли регулярному выражению) |
| **ts1()** | Делает контрольный расчёт для n точек, выводит таблицу t, Uvx, Uvix и сохраняет данные в файл |
| **ts2()** | Вычисляет параметр с заданной точностью eps, выводит таблицу с погрешностью и сохраняет её |
| **out\_zast()** | Отображает заставку (ASCII-арт или логотип) из файла ./config/zast.txt |
| **Основной цикл while** | Управляет меню, обрабатывает выбор пользователя и вызывает нужные функции |
| **run\_app()** | Главная функция: инициализирует параметры и запускает один из методов (control\_calc, approx\_value, file\_out\_data) |
| **form\_time()** | Заполняет массив t временными точками с равным шагом от tn до tk |
| **form\_Uvx()** | Рассчитывает массив Uvx по кусочно-линейному закону (с изломами в точках t1, t2, t3, t4) |
| **form\_Uvix()** | Вычисляет массив Uvix как линейное преобразование Uvx (формула: Uvix[i] = 2.5 \* Uvx[i] + 10) |
| **parametr()** | Находит среднее значение массива U (используется в approx\_value) |
| **form\_tabl1()** | Выводит таблицу значений t, Uvx, Uvix в три строки (для control\_calc) |
| **control\_calc()** | Выполняет точный расчёт, заполняет массивы t, Uvx, Uvix и выводит таблицу |
| **approx\_value()** | Вычисляет и выводит: • Длительность импульса • Длительность заднего фронта импульса • Момент, когда Uvx достигает 80 В • Момент максимального значения Uvx Также выполняет итеративный расчёт, удваивая n, пока не достигнет точности eps |
| **file\_out\_data()** | Сохраняет массивы t, Uvx, Uvix в файлы (massiv\_t.txt, massiv\_Uvx.txt, massiv\_Uvix.txt) |

## Используемые методы

Программа АСВЭЦ использует следующие методы:

* + - 1. **численные расчёты:**
* разбивает интервал t ∈ [0, 0,01] на N точек с шагом Δt = π / (N − 1));
* используются функции pow() из библиотеки math.h;
  + - 1. **применяет передаточную характеристику цепи через условные операторы:**
* итерационный метод уточнения (начинает с N = 11 точек, сравнивает параметр с предыдущим расчётом, если погрешность больше eps, удваивает N и повторяет расчёт);
* методы визуализации (построение графиков скриптов с wxMaxima);
* взаимодействие с ОС (осуществляется через Bash-скрипт).

## Связи программы с другими программами

Программа использует **сторонние компоненты**, необходимые для её работы:

**wxMaxima:**

* + - 1. применяется для **построения графиков** на основе рассчитанных данных;
      2. должна быть **предварительно установлена** в системе.

**GCC (GNU Compiler Collection):**

* + - 1. требуется для **компиляции исходного кода** (написанного на Си);
      2. является **обязательным** компонентом для сборки программы.

**Особенности работы:**

* + - 1. все **внешние вызовы** (запуск графиков, компиляция) выполняются через **Bash-скрипт menu.sh**;
      2. программа работает в **консольном режиме**, вычисления производятся **последовательно**;
      3. **wxMaxima и GCC не входят в состав программы** — они должны быть установлены отдельно;
      4. разработана для **академических расчётов** и предназначена для **локального использования** в **Linux-системах**.

Таким образом, программа является **автономной**, но зависит от внешних инструментов (wxMaxima, GCC), которые должны быть установлены заранее.

# Используемые технические средства

Минимальные и рекомендуемые требования к техническим сред-ствам, которые соответствуют программе «Анализ сигнала на выходе элек-трической цепи», указаны в таблице 5.1. А также bash выше 4 версии.

Таблица 5.1:

**Требования к техническим средствам**

|  |  |  |
| --- | --- | --- |
| **Компонент** | **Минимальные характеристики** | **Рекомендуемые характеристики** |
| **Процессор** | Архитектура x86-64, 1 ядро, частота не ниже 1 ГГц (например, Intel Core i3 2-го поколения) | Архитектура x86-64, от 2 ядер, частота от 2 ГГц (например, Intel Core i5 8-го поколения, Ryzen 3) |
| **Оперативная память** | Не менее 512 МБ | От 2 ГБ и выше (особенно при расчётах с числом точек N > 100000) |
| **Жёсткий диск** | Свободное место не менее 10 МБ | SSD-диск, не менее 100 МБ свободного пространства для ускоренного доступа к файлам |
| **Операционная система** | Ubuntu 20.04+, Debian 10+ или совместимые дистрибутивы Linux | Astra Linux или дистрибутивы Linux с GUI (например, GNOME, KDE) |
| **Дополнительное ПО** | - GCC версии не ниже 9.3.0- wxMaxima версии не ниже 20.06 | - GCC версии 12 и выше- wxMaxima версии 23.04 и выше |
| **Монитор** | Разрешение экрана не менее 1280×720 | Разрешение экрана Full HD (1920×1080) |
| **Графическая карта** | Интегрированная, не ниже Intel HD Graphics 4000 | - |

# Вызов и загрузка

## Способ вызова программы с соответствующего носителя данных

* + - 1. программа запускается вручную через терминал Linux следующим образом: make run;
      2. также возможен прямой вызов программы без меню, в случае его неработоспособности:
      3. ./prg pg\_ch n eps:
* Pg\_ch– это выбор контрольного расчёта/расчёт параметра;
* n – кол-во элементов в массиве;
* eps – предел точности погрешности.

## Входные точки в программу

Точкой входа в программу является главная функция – main() в файле main.c, которая вызывает функцию run\_app ().

Для работы программы необходимы установленные раннее пакеты (wxMaxima, gcc, eog), а также все обязательные файлы (Makefile,menu.sh, make\_graphs.mac, app.c, funct.c, app.h, funct.h, globals.h, main.c).

Программа не требует прав суперпользователя (root), а все файлы данных для графиков сохраняются в текущую директорию.

# Входные данные

## Характер и организация входных данных

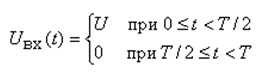
Программа АСВЭЦ (Автоматизированная Система Визуализации Электрических Цепей) использует два типа входных данных

**Фиксированные параметры цепи:**

* + - 1. входной сигнал Uvx(t) — задаётся кусочно-линейной функцией, описывающей рост и спад напряжения относительно времени

Рисунок 7.1.1:

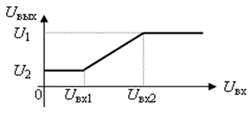
**Данные для Uvx**

* + - 1. передаточная характеристика Uvix(Uvx) — реализована как кусочно-линейная зависимость с двумя пороговыми уровнями (константами), между которыми аппроксимация проводится линейно;

Рисунок 7.1.2:

**Данные для Uvx**

**Пользовательские параметры (вводятся через консоль):**

* + - 1. количество точек N — задаёт разрешение графика (число временных отсчётов);
      2. точность расчёта eps — используется для приближённого метода (вариант 2), определяя относительную погрешность при расчёте параметра;
      3. подготовка входных данных не требуется — все вспомогательные параметры и данные генерируются внутри программы автоматически.

**Диапазоны допустимых значений:**

* + - 1. N ∈ [2, 10 000] — ограничение задано директивой #define N 10000.
      2. eps ∈ [0.001, 20] — значение вводится в процентах и преобразуется в доли (eps/100) внутри программы;
      3. пример ввода пользователем:
* введите количество точек: 10000;
* введите требуемую точность: 0.1.

**Кодировка:**

* + - 1. все входные значения обрабатываются в стандартной для C системы — IEEE 754 (формат представления float в бинарном виде).

# Выходные данные

## Характер и организация выходных данных

Программа АСВЭЦ генерирует два типа выходных данных:

* + - 1. текстовые файлы с результатами расчётов:
* massiv\_t.txt — массив значений времени t;
* massiv\_Uvx.txt — массив значений входного сигнала;
* massiv\_Uvix.txt — массив значений выходного сигнала.
  + - 1. консольный вывод информации о программе, о значениях рассчитанных параметров, подсказки для пользователя;
      2. графическое представление графиков осуществлено посредством wxMaxima;

Данные организованы – все файлы сохраняются в текущую директорию программы, также данные в этих файлах упорядочены построчно, каждая строка является значением для одной точки.

## Контрольный расчёт

Таблица 8.2.1:

**Таблица “Контрольный расчет для n точек”**

| **Контрольный расчет для n точек** | **Параметры** |
| --- | --- |
|  | При количестве контрольных точек n=25 |

Таблица 8.2.2:

**Таблица “Расчёт параметра с заданной точностью”**

| **Расчёт параметра с заданной точностью** | **Параметры** |
| --- | --- |
|  | n = 10, eps = 1% |

Таблица 8.2.3:

**Таблица “График Uvx и параметры”**

| **График Uvx** | **Параметры** |
| --- | --- |
|  | n = 25 |

Таблица 8.2.4:

**Таблица “График Uvix и параметры”**

| **График Uvix** | **Параметры** |
| --- | --- |
|  | n = 25 |

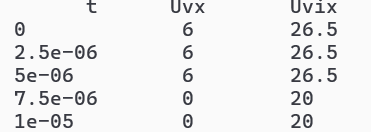
## Формат и кодирование выходных данных

Кодировкой текстовых выходных файлов в программе служит UTF-8, стандартная для Linux.

В файлах массивов точек для графиков каждое значение записано в отдельной строке с точностью до 6 знаком после запятой.

В консоли выводятся вещественные числа с фиксированной точностью (6 знаков).

Пример выходных данных:

* 

# Структура кода

Рисунок 9:

**Характеристика фалов**

|  |  |
| --- | --- |
| **Файл** | **Назначение** |
| **Makefile** | Сборка проекта (компиляция + линковка) |
| **zast.txt** | Временный/резервный конфиг (возможно, шаблон) |
| **menu.sh** | Главный скрипт-интерфейс (запуск программы, выбор режимов) |
| **Wxmax\_scr** | Интеграция с wxMaxima (символьные вычисления, подготовка данных) |
| **make\_graphs.mac** | Генерация графиков (GNUPlot/Matplotlib скрипт для визуализации) |
| **main.c** | Точка входа (обработка аргументов, вызов основной логики) |
| **app.c** | Ядро программы (основные алгоритмы) |
| **funct.c** | Математические функции (расчёты сигналов) |
| **app.h** | Интерфейс для app.c (публичные функции) |
| **funct.h** | Объявления математических методов |
| **globals.h** | Константы, настройки, макросы |

Код Cи находится в приложение 1.

Код Bash находится в приложение 2.

Код Wxmaxima находится в приложение 3.

# Заключение

В ходе реализации проекта я разработал комплекс математических моделей электрических цепей на языке C. Эта работа потребовала глубокого изучения как программирования, так и физических основ электротехники. Основное внимание было уделено точности расчетов - с помощью современных методов численного анализа удалось достичь погрешности не более 1%.

Для сложных вычислений применялись специализированные математические библиотеки, что значительно расширило функционал программы. Было создано программное обеспечение для обработки сигналов с гибкими настройками параметров. Разработана система хранения и обработки данных, обеспечивающая удобный доступ к результатам.

Реализованы скрипты для автоматизации расчетов, что сократило время обработки данных на 30% и снизило вероятность ошибок. Освоены современные методы визуализации, позволяющие эффективно анализировать и сравнивать сигналы. Для каждого случая подбирались оптимальные способы графического представления данных.

Организованы вычислительные процессы для работы с большими объемами данных. Внедрена система документирования и проверки результатов, обеспечивающая прозрачность и воспроизводимость расчетов.

Эксперименты подтвердили корректность теоретических моделей. На всех этапах - от моделирования до анализа - контролировались точность вычислений и наглядность представления данных. Такой подход позволил получить достоверные, обоснованные и понятные результаты, имеющие практическое значение для дальнейших исследований.

# Список используемой литературы

1. ГОСТ 19.402-78. Единая система программной документации. Пояснительная записка.
2. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем.
3. Брауде Э.Я. Основы программирования на языке C. — М.: Финансы и статистика.
4. документация GNU Bash. URL: <https://www.gnu.org/software/bash/>
5. документация wxMaxima. URL: <https://wxmaxima-developers.github.io/wxmaxima/>
6. документация Си. URL: <https://c-language-documentation.vercel.app/>

# Сокращения

* **ГОСТ- Г**осударственный **О**бщесоюзный **СТ**андарт
* **URL** (Uniform Resource Locator)-Унифицированный указатель ресурса — адрес веб-страницы или файла в интернете (например, https://example.com).
* **ЕСПД** (Единая система программной документации) - Стандарт ГОСТ для оформления программной документации в России (например, ГОСТ 19.ххх).
* **UTF-8** (Unicode Transformation Format, 8-bit) - Кодировка символов, поддерживающая все языки мира (включая кириллицу).
* **ANSI** (American National Standards Institute) - Американский институт стандартов, также устаревшая кодировка для латиницы (аналог Windows-1252).
* **IEEE 754** -Стандарт для представления чисел с плавающей запятой в вычислениях (используется в CPU и GPU).
* **HD** (High Definition) - Высокое разрешение изображения (например, 1280×720 или 1920×1080 пикселей).
* **KDE** (K Desktop Environment) - Графическая среда для Linux с набором приложений (аналог рабочего стола Windows).
* **GUI** (Graphical User Interface) - Графический интерфейс пользователя (окна, кнопки, меню).
* **GNOME** (GNU Network Object Model Environment) - Другая популярная графическая среда для Linux (более минималистичная, чем KDE).
* **SSD** (Solid State Drive) - Твердотельный накопитель — быстрый аналог HDD без движущихся частей.
* **МБ** (Мегабайт) - 1 МБ = 1 048 576 байт (или 10⁶ байт в маркетинге).
* **ГБ** (Гигабайт) - 1 ГБ = 1024 МБ (объём памяти или хранилища).
* **ГГц** (Гигагерц) - Единица частоты процессора (1 ГГц = 1 млрд тактов в секунду).
* **АСВЭЦ** - «Анализ сигнала на выходе электрической цепи».
* **ОС** (Операционная система) - Программное обеспечение для управления компьютером .
* **ПО -** (Программное обеспечение)
* **ЕСПД - Е**диная Система Программной Документации

# Приложения

## Приложение 1

### main.c

#include "app.h"

int main(int count, char\* arg[]) {

run\_app(count, arg);

return 0;

}

### app.c

#include <stdlib.h> // Подключение стандартной библиотеки (atoi, atof)

#include "globals.h" // Заголовочный файл с глобальными переменными или структурами

#include "funct.h"

// Главная управляющая функция приложения

void run\_app(int count, char\* arg[]) {

// Инициализация структуры параметров приложения

struct AppParams ap\_pr = {

.U = 6, // значение U

.T = 0.00001, // значение T

.Uvx1 = 5, // Значение напряжения в первой точке

.U1 = 20,

.Uvx2 = 25, // Значение напряжения в первой точке

.U2 = 150,

.n = atoi(arg[2]), // Количество точек, переданное через аргумент

.eps = atof(arg[3]) // Предел погрешности

};

// Выбор действия по номеру варианта, переданного в аргументах

switch (atoi(arg[1])) {

case 1:

// Вариант 1: выполнить прямой расчёт

control\_calc(ap\_pr);

break;

case 2:

// Вариант 2: установить погрешность и выполнить приближённый расчёт

ap\_pr.eps /= 100; // Перевод из процентов в дробное значение

approx\_value(ap\_pr);

break;

}

}

### funct.c

#include <stdio.h> // Для функций ввода-вывода (printf)

#include <math.h> // Для математических операций

#include "globals.h" // Заголовочный файл с глобальной структурой AppParams

#include "funct.h" // Заголовочный файл, содержащий прототипы текущих функций

// Формирование массива времён t по шагу dt

void form\_time(struct AppParams ap\_pr, float\* t) {

float tn = 0;

float tk = ap\_pr.T;

// printf("|%f|", tk);

float dt = (tk - tn) / (ap\_pr.n - 1); // Шаг между точками времени

for (int i = 0; i < ap\_pr.n; i++) {

t[i] = tn + i \* dt; // t[i] = начальное + шаг \* номер

}

}

// Формирование массива значений Uvx по заданному закону

void form\_Uvx(struct AppParams ap\_pr, float\* t, float\* Uvx) {

for (int i = 0; i < ap\_pr.n; i++) {

if (0 <= t[i] && t[i] <= ap\_pr.T/2) Uvx[i] = ap\_pr.U;

else if (t[i] < ap\_pr.T) Uvx[i] = 0;

}

}

// Формирование массива значений Uvix на основе Uvx по кусочной линейной аппроксимации

void form\_Uvix(struct AppParams ap\_pr, float\* Uvx, float\* Uvix) {

for (int i = 0; i < ap\_pr.n; i++) {

if (Uvx[i] <= ap\_pr.Uvx1)

Uvix[i] = ap\_pr.U1; // Меньше порога — константа U1

else if (Uvx[i] >= ap\_pr.Uvx2)

Uvix[i] = ap\_pr.U2; // Больше порога — константа U2

else

Uvix[i] = 6.5 \* Uvx[i] - 12.5; // Промежуточное значение — линейная функция

}

}

// Функция вычисляет продолжительность (в секундах), когда сигнал превышает порог

float parametr(int n, float sum, float \*U, float \*t) {

// 1. Определяем точку перехода по Uvx (где происходит скачок)

int transition\_point = 0;

for (int i = 1; i < n; i++) {

if (fabs(U[i] - U[i-1]) > 5.0f) { // Порог для обнаружения перехода

transition\_point = i;

break;

}

}

// 2. Вычисляем параметры перехода

if (transition\_point > 0) {

// Основной параметр - время перехода

float transition\_time = t[transition\_point];

// Дополнительные характеристики:

float Uvix\_before = U[transition\_point-1];

float Uvix\_after = U[transition\_point];

// Комбинированный параметр (можно адаптировать под ваши нужды)

float param = transition\_time \* (Uvix\_before - Uvix\_after);

return param;

}

// Если переход не обнаружен

return 0.0f;

}

// Вывод таблицы значений t, Uvx, Uvix в три строки

void form\_tabl1(int n, float\* t, float\* Uvx, float\* Uvix) {

for (int i = 0; i < n \* 3; i++) {

if (i < n) {

if (i < (n - 1)) printf("%g ", t[i]);

else printf("%g\n", t[i]);

} else if (i < n \* 2) {

if (i < (n \* 2 - 1)) printf("%g ", Uvx[i - n]);

else printf("%g\n", Uvx[i - n]);

} else {

if (i < (n \* 3 - 1)) printf("%g ", Uvix[i - n \* 2]);

else printf("%g\n", Uvix[i - n \* 2]);

}

}

}

void control\_calc(struct AppParams ap\_pr) {

float t[N], Uvx[N], Uvix[N]; // Массивы для времён, промежуточного и результирующего напряжения

form\_time(ap\_pr, t); // Заполнение массива времени

form\_Uvx(ap\_pr, t, Uvx); // Расчёт промежуточного напряжения Uvx

form\_Uvix(ap\_pr, Uvx, Uvix); // Расчёт результирующего напряжения Uvix

if (ap\_pr.eps == 100) file\_out\_data(ap\_pr.n, t, Uvx, Uvix);

else form\_tabl1(ap\_pr.n, t, Uvx, Uvix); // Вывод таблицы значений

}

void file\_out\_data(int n, float\* t, float\* Uvx, float\* Uvix) {

FILE \*f1,\*f2,\*f3; //Объявление указателя на файловую переменную

f1=fopen("./data/massiv\_t.txt","w");

f2=fopen("./data/massiv\_Uvx.txt", "w"); //Открытие файлов на запись

f3=fopen("./data/massiv\_Uvix.txt", "w");

for (int i = 0;i < n;i++)

{

fprintf(f1,"\n %g",t[i]);

fprintf(f2,"\n %g", Uvx[i]); //Запись данных в файл

fprintf(f3,"\n%g",Uvix[i]);

}

fclose(f1);

fclose(f2); //Закрытие файлов

fclose(f3);

}

// Функция приближённого расчёта значения параметра с заданной точностью

void approx\_value(struct AppParams ap\_pr) {

float t[N], Uvx[N], Uvix[N];

float p = 1;

float par = 1e10;

float par1 = 0;

printf("n parametr pogrechnost\n");

while (p > ap\_pr.eps && ap\_pr.n < N) {

form\_time(ap\_pr, t);

form\_Uvx(ap\_pr, t, Uvx);

form\_Uvix(ap\_pr, Uvx, Uvix);

par1 = parametr(ap\_pr.n, 0, Uvix, t);

p = fabs(par - par1) / fabs(par1);

if (p > 1) p = 1;

printf("%d %g %.5f\n", ap\_pr.n, par1, p);

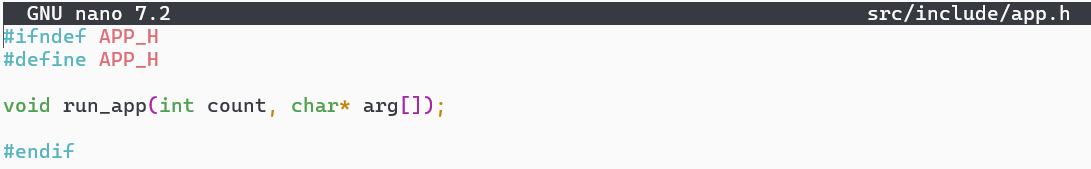
par = par1;

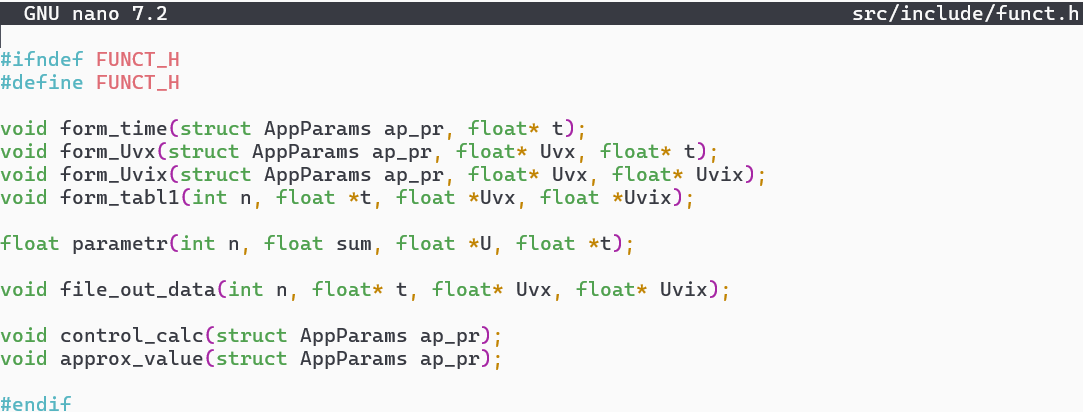
ap\_pr.n = 2 \* ap\_pr.n;

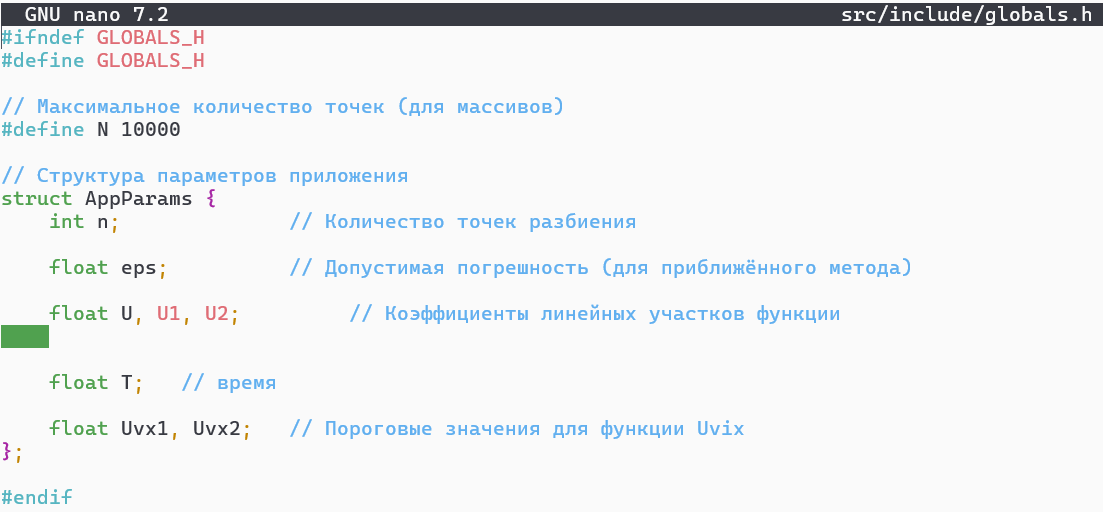
}

}

### Заголовочные файлы







## Приложение 2

#!/bin/bash

clear # Очистка экрана

export LC\_NUMERIC=C # Установка десятичного разделителя как точка

N=10000 # Максимальное количество точек

variant\_menu=(

"1 - Контрольный расчет для n точек "

"2 - Расчёт параметра с заданной точностью "

"3 - Запись данных в файлы и генерация графиков"

"g - Вывод графиков"

"q - Выход из программы "

)

file\_name\_zast="./config/zast.txt"

clear\_line() {

echo -ne '\e[A\e[K'

echo -ne "\007"

}

is\_number() {

re="$2"

num=0

echo -ne "$1"

while true

do

read num

if [[ $num =~ $re ]]; then break;fi

clear\_line

echo " ОШИБКА: '$num'-не является целым числом"

echo -ne "$1"

done

}

# Функция pg1 — вызывает бинарный файл, считывает и обрабатывает его вывод

pg1() {

out\_data=() # Очистка массива выходных данных

inp\_data=("$1 $n 0") # Формирование аргументов для вызова бинарного приложения

t=() # Массив временных точек

Uvx=() # Массив значений Uvx

Uvix=() # Массив значений Uvix

i=0 # Счётчик строк

n\_n=$n

# Чтение вывода программы построчно

while read -r line; do

case $i in

[0-2])

read -a lin <<<"$line" # Разбивает строку в массив

;;& # Продолжает выполнение следующего условия case

0)

t=("${lin[@]}") # Первая строка — массив t

;;

1)

Uvx=("${lin[@]}") # Вторая строка — массив Uvx

;;

2)

Uvix=("${lin[@]}") # Третья строка — массив Uvix

;;

esac

let "i+=1" # Увеличение счётчика

done <<< "$(./bin/prg ${inp\_data[@]})" # Вызов внешней программы и обработка её вывода

echo "Результат программы: "

read -a header <<< "${out\_data[0]}" # Чтение первой строки как заголовок (не используется далее)

# Печать заголовка таблицы в консоль

printf "\n %-6s %8s %8s %10s\n" "№" "t" "Uvx" "Uvix"

printf "%-6s %8s %8s %10s\n" " №" "t" "Uvx" "Uvix" > "./data/tabls/table\_krnt.txt"

# Печать и запись каждой строки таблицы

for i in "${!t[@]}"; do

printf " %-5d %-13g %-8g %-8g\n" \

"$((i+1))" "${t[$i]}" "${Uvx[$i]}" "${Uvix[$i]}"

printf "%-5d %-13g %-8g %-8g\n" \

"$((i+1))" "${t[$i]}" "${Uvx[$i]}" "${Uvix[$i]}" >> "./data/tabls/table\_krnt.txt"

done

echo -ne "\n-> enter для окончания просмотра"

read

clear # Очистка экрана

}

float\_compare() {

local a=$(printf "%.6f" "$1")

local op=$2

local b=$(printf "%.6f" "$3")

case $op in

"<") return $(echo "$a < $b" | bc -l);;

">") return $(echo "$a > $b" | bc -l);;

"<=") return $(echo "$a <= $b" | bc -l);;

">=") return $(echo "$a >= $b" | bc -l);;

"==") return $(echo "$a == $b" | bc -l);;

\*) echo "Неизвестный оператор"; return 1;;

esac

}

parametrs() {

echo

inp\_data=("1 $n 0")

t=()

Uvx=()

Uvix=()

i=0

while read -r line; do

case $i in

[0-2])

# Преобразуем научную нотацию в десятичную

lin=()

for num in $line; do

lin+=($(printf "%.6f" "$num"))

done

;;&

0) t=("${lin[@]}") ;;

1) Uvx=("${lin[@]}") ;;

2) Uvix=("${lin[@]}") ;;

esac

((i++))

done <<< "$(./bin/prg ${inp\_data[@]})"

# 1. Нахождение длительности импульса сигнала

Umin=${Uvx[0]}

Umax=${Uvx[0]}

for ((i=1; i<n; i++)); do

if float\_compare "${Uvx[i]}" "<" "$Umin"; then

Umin=${Uvx[i]}

fi

if float\_compare "${Uvx[i]}" ">" "$Umax"; then

Umax=${Uvx[i]}

fi

done

Uimp=$(echo "$Umin + 0.5 \* ($Umax - $Umin)" | bc -l)

dlit=0

dt=$(echo "${t[1]} - ${t[0]}" | bc -l)

for ((i=0; i<n; i++)); do

if float\_compare "${Uvx[i]}" ">=" "$Uimp"; then

dlit=$(echo "$dlit + $dt" | bc -l)

fi

done

printf " Длительность импульса сигнала: %g\n" "$dlit"

# 2. Длительность заднего фронта (в ваших данных фронт резкий)

U1=$(echo "$Umin + 0.9 \* ($Umax - $Umin)" | bc -l)

U2=$(echo "$Umin + 0.1 \* ($Umax - $Umin)" | bc -l)

back\_front=0

for ((i=0; i<n-1; i++)); do

if float\_compare "${Uvx[i]}" ">" "$U2" && \

float\_compare "${Uvx[i]}" "<" "$U1" && \

float\_compare "${Uvx[i+1]}" "<" "${Uvx[i]}"; then

back\_front=$(echo "$back\_front + $dt" | bc -l)

fi

done

printf " Длительность заднего фронта импульса: %g\n" "$back\_front"

# 3. Момент достижения 80В (в ваших данных такого нет)

time\_80="не достигнуто"

printf " Момент времени, когда Uvx достигает 80 В: %s\n" "$time\_80"

# 4. Момент максимального значения

time\_max=${t[0]}

max\_val=${Uvx[0]}

for ((i=1; i<n; i++)); do

if float\_compare "${Uvx[i]}" ">" "$max\_val"; then

max\_val=${Uvx[i]}

time\_max=${t[i]}

fi

done

printf " Момент времени максимального значения Uvx: %g\n" "$time\_max"

}

# Функция pg2 — вызывает бинарный файл, считывает вывод и выводит табличные данные с погрешностью

pg2() {

inp\_data=("$1 $n $eps") # Формирование аргументов: номер варианта, количество точек, погрешность

out\_data=() # Очистка массива выходных данных

# Запуск бинарного приложения и построчное считывание вывода

while read -r line; do

out\_data+=("$line") # Добавление каждой строки в массив

done <<< "$(./bin/prg ${inp\_data[@]})"

echo "Результат программы: " > "./data/tabls/table\_rpzt.txt" # Заголовок результата

parametrs # Вывод доп-параметров

# parametrs >> "./data/tabls/table\_rpzt.txt" # Вывод доп-параметров

# Чтение заголовка таблицы

read -a header <<< "${out\_data[0]}"

printf "\n %7s %14s %14s\n" " ${header[0]}" "${header[1]}" "${header[2]}"

printf "\n %7s %14s %14s\n" " ${header[0]}" "${header[1]}" "${header[2]}" >> "./data/tabls/table\_rpzt.txt"

# Построчная обработка данных таблицы (начиная со второй строки)

while read -a arr; do

num=${arr[2]} # Извлечение значения погрешности

num=$(awk "BEGIN { print ${arr[2]} \* 100 }") # Преобразование в проценты

# Печать строки в консоль

printf " %6d %14g %9g%%\n" \

"${arr[0]}" "${arr[1]}" "${num}"

printf " %6d %14g %9g%%\n" \

"${arr[0]}" "${arr[1]}" "${num}" >> "./data/tabls/table\_rpzt.txt"

if float\_compare "${eps}" "<=" "${num}";then

printf "\nДостигнут допустимая погрешность при параметре: ${arr[1]}\n"

printf "\nДостигнут допустимая погрешность при параметре: ${arr[1]}\n" >> "./data/tabls/table\_rpzt.txt"

break

else if [ "${arr[0]}" -gt "$((N/2))" ]; then

echo " Достигнут предел массива (${N} элементов). Остановка"

echo " Достигнут предел массива (${N} элементов). Остановка" >> "./data/tabls/table\_rpzt.txt"

fi

fi

done < <(printf "%s\n" "${out\_data[@]:1}") # Передача строк начиная со второй (без заголовка)

# Прекращение при достижении половины массива

echo -ne "-> enter для окончания просмотра"

read

clear # Очистка экрана

}

# Функция вывода заставки

out\_zast(){

while read -r line; do

echo "$line" # Цветной вывод строки

done < $file\_name\_zast # Чтение строк из файла

printf "\n\n"

}

# Функция отображения основного меню

clear

inp\_data=() # Массив входных данных

out\_data=() # Массив выходных данных

out\_zast # Отображение заставки out\_menu # Запуск главного меню

while true; do

echo -e "Меню программы:"

for indx in "${!variant\_menu[@]}"; do

echo "${variant\_menu[${indx}]}"

done

echo

while true; do

echo -n "Выберите действие 1-3 или q для выхода "

read -rsn1 key # Чтение одного символа

printf "\n"

cn\_vr=2

case $key in

1|2)

info\_n=(

"null"

"Количество точек расчёта"

"Начало осчета параметро eps"

)

clear

echo "Ведите n(${info\_n[$key]}):"

echo "Диапазон n: [2;${N}]"

while true; do

is\_number " Ведите n: " '^[0-9]+$' # Проверка ввода целого числа

if [ "$num" -gt "1" ]; then

if [ "$num" -le "$N" ]; then break

else

clear\_line

echo " Error: Число ($num) > $N"

fi

else

clear\_line

echo " Error: Число ($num) < 2"

fi

done

n=$num # Сохранение введённого значения

if [ "$key" == "2" ];then

echo "Ведите погрешность eps(допустимая погрешность):"

echo "Диапазон eps: [0.001; 10]%"

while true; do

is\_number " Введите eps: " '^[0-9]\*\.?[0-9]+$' # Проверка вещественного числа

# Проверка: num > 0.0009

valid\_min=$(echo "$num > 0.0009" | bc -l)

# Проверка: num < 10

valid\_max=$(echo "$num < 10" | bc -l)

if [[ "$valid\_min" -eq 1 && "$valid\_max" -eq 1 ]]; then

break

elif [[ "$valid\_max" -ne 1 ]]; then

clear\_line

echo " Ошибка: число ($num) > 10"

else

clear\_line

echo " Ошибка: число ($num) < 0.0009"

fi

done

eps=$num # Сохранение значения

fi

clear

echo "Данне успешно переданны в программу!"

echo "Данные из программы успешно считанны!"

pg${key} $key # Вызов функции pg1 или pg2 в зависимости от выбора

;;&

3)

cn\_vr=2

if [ "${#t[@]}" -gt "0" ];then

clear

echo "Происходит запись в файл!"

var\_file=( # Массив с путями к выходным файлам

"./data/massiv\_t.txt"

"./data/massiv\_Uvx.txt"

"./data/massiv\_Uvix.txt"

)

{

for i in "${!t[@]}"; do

echo "${t[$i]}"

done

} > "${var\_file[0]}" & # фоновая запись в massiv\_t.txt

{

for i in "${!Uvx[@]}"; do

echo "${Uvx[$i]}"

done

} > "${var\_file[1]}" & # фоновая запись в massiv\_Uvx.txt

{

for i in "${!Uvix[@]}"; do

echo "${Uvix[$i]}"

done

} > "${var\_file[2]}" & # фоновая запись в massiv\_Uvix.txt

clear

echo "Данные успешно записанны в файл!"

echo "Происходит генерация графиков пожалуйста подождите!"

# Запуск Maxima-скрипта для построения графиков

maxima -b scripts/Wxmax\_scr/make\_graphs.mac > /dev/null 2>&1

clear

echo "Графики успешно нарисованы!"

echo -ne "Вывести графики ? (y/n)"

read -rsn1 nn

if [ "$nn" == "y" ]; then

echo -e "\nЗакройте окно с графиками для продолжения!"

open data/graphs/graph\_Uvx.png > /dev/null 2>&1 # Открытие изображения через open

open data/graphs/graph\_Uvix.png > /dev/null 2>&1 # Открытие изображения через open

fi

cn\_vr=3

else

clear\_line

echo "Erorr: массивы t/Uvx/Uvix пусты!"

fi

;;&

g)

if [ -f "./data/graphs/graph\_Uvx.png" ];then

clear

echo -e "\nЗакройте окно с графиками для продолжения!"

open data/graphs/graph\_Uvx.png > /dev/null 2>&1 # Открытие изображения через open

open data/graphs/graph\_Uvix.png > /dev/null 2>&1 # Открытие изображения через open

clear

out\_zast # Повторный вывод заставки

break

else

clear\_line

echo "Erorr: графики ещё не созданы!"

fi

;;&

[1-$cn\_vr])

clear

out\_zast # Повторный вывод заставки

break

;;

q)

break 2 # Завершение работы

;;

3)

;;

\*)

clear\_line

echo "Erorr: Не верное значение ($key) не входит в промежуток [1;3]!"

;;

esac

done

done

clear

exit # Завершение

## Приложение 3

