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 – 11 вариант

Входной сигнал – 11 вариант

Выполнил студент группы ИКПИ-41:

Кучменко Н.И. \_\_\_\_\_\_\_\_

Дата выполнения: «13» Май

Проверил:

Хазиев Н.Н. \_\_\_\_\_\_\_\_

Санкт-Петербург

2025

# Оглавление

[Оглавление 2](#_Toc198082713)

[Задание к курсовой работе 4](#_Toc198082714)

[Аннотация 5](#_Toc198082715)

[1) Общие сведения 7](#_Toc198082716)

[1.1) Обозначение и наименование программы 7](#_Toc198082717)

[1.2) Языки разработки: 7](#_Toc198082718)

[2) Функциональное назначение 7](#_Toc198082719)

[2.1) Решаемые задачи 7](#_Toc198082720)

[2.2) Назначение: 8](#_Toc198082721)

[2.3) Ограничения в функциональности: 8](#_Toc198082722)

[3) Контрольный расчёт 10](#_Toc198082723)

[4) Используемые технические средства 12](#_Toc198082724)

[5) Таблица идентификаторов 13](#_Toc198082725)

[6) Описание функций 14](#_Toc198082726)

[7) Алгоритм работы программы 16](#_Toc198082727)

[7.1) Блок схема 16](#_Toc198082728)

[7.2) Описание логической структуры программы 16](#_Toc198082729)

[a) Инициализация и запуск 16](#_Toc198082730)

[b) Обработка пользовательского ввода 17](#_Toc198082731)

[c) Выполнение расчетов в Си-программе 17](#_Toc198082732)

[d) Режимы работы: 18](#_Toc198082733)

[e) Обработка результатов в Bash 18](#_Toc198082734)

[8) Вызов и загрузка 20](#_Toc198082735)

[8.1) Способ вызова программы с соответствующего носителя данных 20](#_Toc198082736)

[8.2) Входные точки в программу 20](#_Toc198082737)

[9) Входные данные 22](#_Toc198082738)

[9.1) Характер и организация входных данных 22](#_Toc198082739)

[10) Выходные данные 24](#_Toc198082740)

[10.1) Характер и организация выходных данных 24](#_Toc198082741)

[10.2) Формат и кодирование выходных данных 25](#_Toc198082742)

[10.3) Пример файла massiv\_Uvx.txt: 25](#_Toc198082743)

[10.4) Пример консольного вывода таблицы: 25](#_Toc198082744)

[11) Структура кода 26](#_Toc198082745)

[12) Заключение 28](#_Toc198082746)

[13) Список используемой литературы 30](#_Toc198082747)

[14) Сокращения 31](#_Toc198082748)

[15) Приложения 33](#_Toc198082749)

[15.1) Приложение 1 33](#_Toc198082750)

[a) src/main.c 33](#_Toc198082751)

[b) src/core/app.c 33](#_Toc198082752)

[c) src/core/funct.c 33](#_Toc198082753)

[d) src/core/krnt.c 34](#_Toc198082754)

[e) src/core/rpzt.c 34](#_Toc198082755)

[f) src/include/app.h 35](#_Toc198082756)

[g) src/include/funct.h 35](#_Toc198082757)

[h) src/include/krnt.h 35](#_Toc198082758)

[i) src/include/rpzt.h 35](#_Toc198082759)

[j) src/include/globals.h 35](#_Toc198082760)

[15.2) Приложение 2 36](#_Toc198082761)

[a) scripts/menu.sh 36](#_Toc198082762)

[b) scripts/helpers/file\_output.sh 42](#_Toc198082763)

[c) scripts/helpers/functions.sh 43](#_Toc198082764)

[d) scripts/helpers/variables.sh 46](#_Toc198082765)

[e) scripts/helpers/p1.sh 47](#_Toc198082766)

[f) scripts/helpers/p2.sh 52](#_Toc198082767)

[15.3) Приложение 3 53](#_Toc198082768)

[a) scripts/Wxmax\_scr/make\_graphs.mac 53](#_Toc198082769)

# 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) электрической цепи приведены в приложении. Номер варианта определяется преподавателем индивидуально для каждого студента.

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

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

* 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.

# 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.ххх) 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.

# Общие сведения

## Обозначение и наименование программы

Для корректной работы программа требует установленную русифицированную версию операционной системы Ubuntu Linux.  
Также необходима установка стороннего ПО **wxMaxima**. Установить его можно командой в терминале:

**sudo apt-get install wxmaxima**

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

**sudo apt update**

**sudo apt install build-essential**

Ещё необходимые ПО eog и xterm:

**sudo apt install xterm**

**sudo apt install eog**

## Языки разработки:

Программа написана на языке программирования Си — на нём реализо-вана основная функциональность.

Меню реализовано с использованием Bash-скриптов, которые также запускают скрипт для wxMaxima, оформленный в виде текстового файла с расширением .mac.

# Функциональное назначение

## Решаемые задачи

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

* **Моделирование сигналов:**
* Расчёт функции входного напряжения Uвх(t), заданной в аналитической форме;
* Вычисление выходного напряжения Uвых(t), используя кусочно-линейную передаточную характеристику.
* Графическая визуализация:
  + Построение графиков Uвх(t) и Uвых(t) с помощью **wxMaxima**;
  + 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:

* **Совместимость с ОС:** программа работает только в Ubuntu Linux и не поддерживает Windows.
* **Ограничения по входным данным:**
  + - Временной диапазон жёстко зафиксирован: t ∈ [π, 2π] (можно изменить вручную в коде);
    - Параметры цепи заданы для варианта №11;
    - Максимальное количество точек **Nmax = 15 000** (определено размером массива).
* **Численные ограничения:**
  + - Используется тип данных float, что ограничивает точность;
    - Итерационные методы могут не сойтись при слишком малом значении **eps**.
* **Системные требования:**
  + Только ОС **Ubuntu Linux**;
  + Наличие **wxMaxima** (для визуализации) и **gcc** (для компиляции) обязательно.
* **Ограничения пользовательского интерфейса:**
  + Отсутствие графического интерфейса — работа осуществляется через консоль;
  + Низкая устойчивость к некорректному вводу.
* **Примечание:** при необходимости использовать программу для других параметров, требуется вручную изменить настройки в исходном файле funct.c.

# Контрольный расчёт

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

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

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

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

# Используемые технические средства

Минимальные и рекомендуемые требования к техническим средствам, которые соответствуют программе «Анализ сигнала на выходе электрической цепи», указаны в таблице 4.1. 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 | - |

# Таблица идентификаторов

Таблица 6.1:

**Таблица идентификаторов**

|  |  |
| --- | --- |
| **Идентификаторы** | **Пояснения** |
| **a** | коэффициент |
| **b** | второй коэффициент |
| **tn** | начальное время |
| **t1** | временной параметр |
| **tk** | Конечное время |
| **U1** | напряжение первого уровня |
| **U2** | напряжение второго уровня |
| **n** | Количество элементов в массиве |
| **eps** | Погрешность |
| **p** | Параметр |
| **t** | Массив времени |
| **Uvx** | Массив входного напряжение |
| **Uvix** | Массив выходного напряжение |

# Описание функций

Таблица 7.1:

**Таблица функций**

| **Функция** | **Описание** |
| --- | --- |
| **main** | Основная точка входа в программу на языке C. Запускает функцию run\_app. |
| **run\_app** | Управляющая функция приложения на C. Выбирает действие (прямой или приближённый расчёт). |
| **form\_time** | Формирует массив времён t по шагу времени. |
| **form\_Uvx** | Вычисляет массив напряжений Uvx на основе временных точек. |
| **form\_Uvix** | Вычисляет массив напряжений Uvix по кусочной линейной аппроксимации. |
| **parametr** | Вычисляет среднее значение сигнала U за промежуток времени. |
| **form\_tabl1** | Выводит таблицу значений времён и напряжений (t, Uvx, Uvix). |
| **control\_calc** | Основная функция расчёта (вариант 1). Формирует массивы и выводит таблицу. |
| **approx\_value** | Основная функция приближённого расчёта (вариант 2) с заданной точностью. |
| **menu.sh** | Скрипт bash. Отображает меню программы и управляет пользователем. |
| **out\_zast** | Выводит заставку программы. |
| **out\_menu** | Основное меню программы. Обрабатывает пользовательский ввод. |
| **start** | Запускает программу, очищает массивы данных. |
| **out\_file** | Записывает массивы t, Uvx, Uvix в текстовые файлы и запускает построение графиков. |
| **style** | Выводит текст с заданным стилем (цветом). |
| **clear\_line** | Очищает текущую строку терминала. |
| **is\_number** | Проверяет, является ли введённое значение числом. |
| **out\_info\_pr** | Выводит текстовые файлы с пояснениями к параметрам программы. |
| **pg1** | Запускает бинарное приложение (вариант 1) и выводит результаты в таблице. |
| **pg2** | Запускает бинарное приложение (вариант 2) и выводит таблицу с погрешностями. |

# Алгоритм работы программы

## Блок схема

## Описание логической структуры программы

### Инициализация и запуск

* Программа запускается через команду make run, которая вызывает главный скрипт menu.sh.
* menu.sh предоставляет пользователю интерактивное меню с вариантами:

1 — Контрольный расчет для n точек

2 — Расчет параметра с заданной точностью eps

p — Вывод пояснений к параметрам

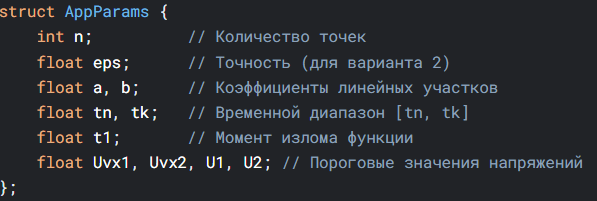
q — Выход из программы

### Обработка пользовательского ввода

* **При выборе варианта 1**:
* Пользователь вводит n (количество точек, диапазон [2; 15000]).
* Bash-скрипт передает аргументы в Си-программу: ./myapp 1 n.0
* **При выборе варианта 2:**
* Пользователь вводит n и eps (точность, диапазон [0.001; 99]).
* Bash-скрипт передает аргументы: ./myapp 2 n eps.

### Выполнение расчетов в Си-программе

* + Входные параметры передаются через структуру AppParams:

****

* + Основные функции:
* **form\_time():** Генерирует массив времени t с равномерным шагом.
  + **form\_Uvx():** Рассчитывает входное напряжение Uvx(t) как кусочно-линейную функцию.
  + **form\_Uvix():** Преобразует Uvx в выходное напряжение Uvix через пороговую аппроксимацию.
  + **parametr():** Вычисляет длительность превышения сигналом порогового значения.
  + **form\_tabl1():** Форматирует результаты в таблицу.

### Режимы работы:

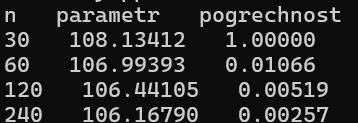
#### Вариант 1 (control\_calc):

* Вычисляет массивы **t, Uvx, Uvix** для заданного n.
* Возвращает таблицу значений в формате:



#### Вариант 2 (approx\_value):

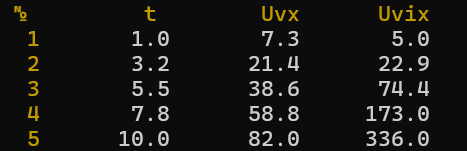
* Итерационно увеличивает n (в 2 раза на каждом шаге) до достижения точности eps.
* Для каждого n вычисляет параметр duration (длительность превышения порога).
* Возвращает таблицу в формате:



### Обработка результатов в Bash

#### ****Для варианта 1****:

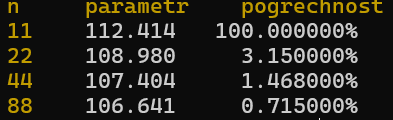
* Данные (t, Uvx, Uvix) записываются в файлы:
  + massiv\_t.txt
  + massiv\_Uvx.txt
  + massiv\_Uvix.txt
* Масивы выводятся виде таблицы:



* Строятся графики через wxMaxima (скрипт make\_graphs.mac):
  + - graph\_Uvx.png — зависимость Uvx(t).
    - graph\_Uvix.png — зависимость Uvix(t).
* Пользователю предлагается просмотреть графики через eog.

#### Для варианта 2

* Итерационно увеличивает n (в 2 раза на каждом шаге) до достижения точности eps.
* Для каждого n вычисляет параметр duration (длительность превышения порога).
* Возвращает таблицу в формате:



* **Возврат в главное меню**

После выполнения любого варианта программа возвращает пользователя в меню menu.sh для новых расчетов или выхода.

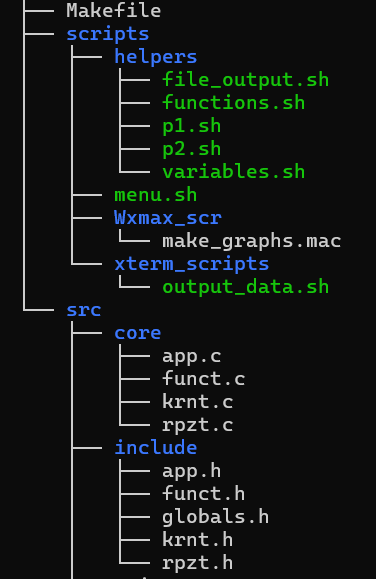
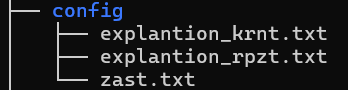
# Вызов и загрузка

## Способ вызова программы с соответствующего носителя данных

* Программа запускается вручную через терминал Linux следующим образом: make run
* Также возможен прямой вызов программы без меню, в случае его неработоспособности:
* Pg – это выбор контрольного расчёта/расчёт параметра
* n – кол-во элементов в массиве
* eps – предел точности погрешности
* ./myapp pg n eps

## 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 (, ).

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

# Входные данные

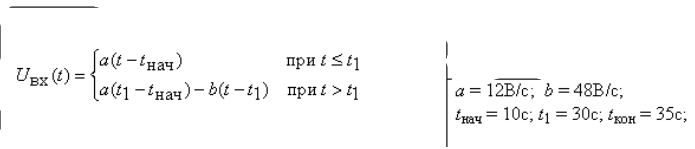
## Характер и организация входных данных

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

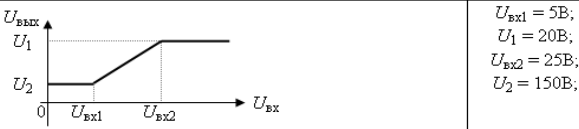
#### Фиксированные параметры цепи

Эти значения жёстко заданы в программе и не требуют ручного ввода:

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



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



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

* **Количество точек** N — задаёт разрешение графика (число временных отсчётов).
* **Точность расчёта** eps — используется для приближённого метода (вариант 2), определяя относительную погрешность при расчёте параметра.
* Подготовка входных данных не требуется — все вспомогательные параметры и данные генерируются внутри программы автоматически.

#### Диапазоны допустимых значений:

* N ∈ [2, 15 000] — ограничение задано директивой #define N 15000.
* eps ∈ [0.001, 99.99] — значение вводится в процентах и преобразуется в доли (eps/100) внутри программы.

#### Пример ввода пользователем:

* Введите количество точек: 1000
* Введите требуемую точность: 0.01

#### ****Кодировка:****

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 формирует два основных вида выходных данных:

#### Текстовые файлы с результатами расчётов

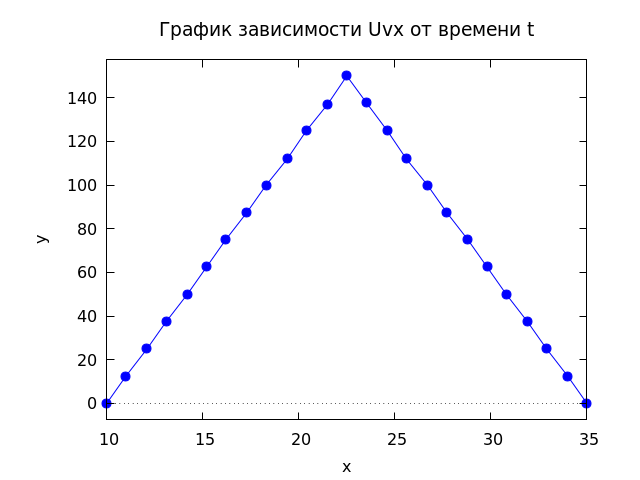
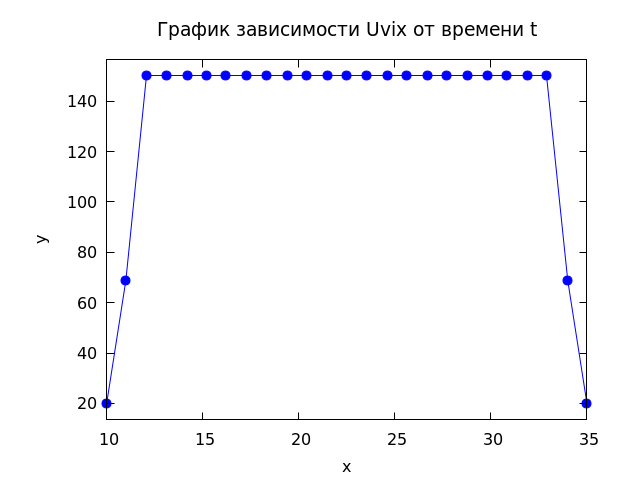
* massiv\_t.txt — значения временной сетки t.
* massiv\_Uvx.txt — значения сигнала Uvx.
* massiv\_Uvix.txt — значения выходного сигнала Uvix.

Файлы автоматически сохраняются в папке ./data/, по одному значению в строке. Также отдельно сохраняется таблица в table\_p1.txt или table\_p2.txt (в зависимости от варианта).

#### Консольный вывод

* Интерфейс реализован с цветной разметкой (поддержка tput, ANSI escape codes).
* Отображаются подсказки, результаты расчётов и таблицы.
* Возможность вызова вспомогательного окна xterm для вывода пояснений.
* Поддержка запуска графиков в графическом интерфейсе (eog).

#### Графики

* Генерация графиков происходит через **Maxima** (скрипт make\_graphs.mac).
* Графики сохраняются в ./data/graphs/ и могут быть открыты через eog.

## Формат и кодирование выходных данных

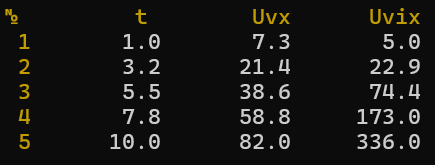
* **Формат текстовых файлов** — обычный текст UTF-8 (одна строка = одно значение).
* **Формат чисел** — вывод значений осуществляется с точностью до 6 знаков после запятой.
* **Кодировка** — UTF-8, стандартная для Linux-терминала.

## **Пример файла** massiv\_Uvx.txt**:**

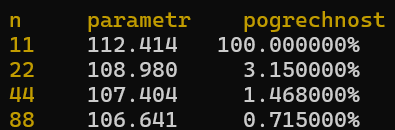
* 0.000000
* 1.800000
* 3.600000

## Пример консольного вывода таблицы:

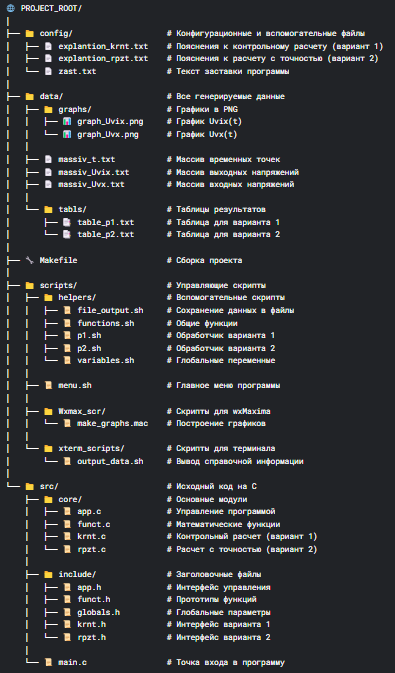
#### Контрольный расчет для n точек



#### Расчёт параметра с заданной точностью



# Структура кода



Код Cи находится в приложение 1.

Код Bash находится в приложение 2.  
Код Wxmaxima находится в приложение 3.

# Заключение

В ходе выполнения проекта я освоил комплексный подход к анализу электрических цепей, научившись преобразовывать входной сигнал в выходной с точностью до 1%. Работа сочетала теоретические расчеты и практическую реализацию.

**Основные этапы работы:**

1. **Математическое моделирование**
   1. Исследовал поведение входного сигнала в цепи
   2. Провел точные расчеты выходных параметров с заданной точностью
2. **Программная реализация**
   1. Разработал программу на языке C с использованием математических функций
   2. Организовал сохранение результатов для последующего анализа
3. **Визуализация данных**
   1. Создал наглядные графики изменения сигналов
   2. Провел сравнительный анализ преобразования сигнала в цепи
4. **Автоматизация процессов**
   1. Реализовал скрипт для автоматического выполнения всех этапов анализа
   2. Оптимизировал процесс обработки данных

**Приобретенные навыки:**

✔ Получено понимание принципов моделирования электрических цепей  
✔ Практический опыт программирования на C с использованием математических библиотек  
✔ Навыки автоматизации расчетов и обработки данных  
✔ Умение представлять результаты в графической форме

**Выводы:**

Выполненная работа позволила не только изучить методы анализа электрических цепей, но и получить ценные практические навыки в области программирования и автоматизации вычислений. Разработанный инструментарий

# 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

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

# Приложения

## Приложение 1

### src/main.c

#include "app.h"  
  
int main(int count, char\* arg[]) {  
 run\_app(count, arg);  
 return 0;  
}

### src/core/app.c

#include <stdlib.h>  
#include "globals.h"  
#include "rpzt.h"  
#include "krnt.h"  
  
void run\_app(int count, char\* arg[]) {  
 struct AppParams ap\_pr = {  
 .a = 12, .b = 12, .tn = 10, .tk = 35,  
 .t1 = 22.5, .Uvx1 = 5, .Uvx2 = 25,  
 .U1 = 20, .U2 = 150, .n = atoi(arg[2])  
 };  
 switch (atoi(arg[1])) {  
 case 1: control\_calc(ap\_pr); break;  
 case 2: ap\_pr.eps = atof(arg[3]) / 100; approx\_value(ap\_pr); break;  
 }  
}

### src/core/funct.c

#include <stdio.h>  
#include <math.h>  
#include "globals.h"  
#include "funct.h"  
  
void form\_time(struct AppParams ap\_pr, float\* t) {  
 float dt = (ap\_pr.tk - ap\_pr.tn) / (ap\_pr.n - 1);  
 for (int i = 0; i < ap\_pr.n; i++) t[i] = ap\_pr.tn + i \* dt;  
}  
  
void form\_Uvx(struct AppParams ap\_pr, float\* t, float\* Uvx) {  
 for (int i = 0; i < ap\_pr.n; i++)  
 Uvx[i] = (t[i] < ap\_pr.t1) ? ap\_pr.a \* (t[i] - ap\_pr.tn)  
 : ap\_pr.a \* (ap\_pr.t1 - ap\_pr.tn) - ap\_pr.b \* (t[i] - ap\_pr.t1);  
}  
  
void form\_Uvix(struct AppParams ap\_pr, float\* Uvx, float\* Uvix) {  
 for (int i = 0; i < ap\_pr.n; i++)  
 Uvix[i] = (Uvx[i] <= ap\_pr.Uvx1) ? ap\_pr.U1 : (Uvx[i] >= ap\_pr.Uvx2) ? ap\_pr.U2 : 6.5 \* Uvx[i] - 12.5;  
}

### src/core/krnt.c

#include "globals.h"  
#include "krnt.h"  
#include "funct.h"  
  
void control\_calc(struct AppParams ap\_pr) {  
 float t[N], Uvx[N], Uvix[N];  
 form\_time(ap\_pr, t);  
 form\_Uvx(ap\_pr, t, Uvx);  
 form\_Uvix(ap\_pr, Uvx, Uvix);  
}

### src/core/rpzt.c

#include <stdio.h>  
#include <math.h>  
#include "globals.h"  
#include "rpzt.h"  
#include "funct.h"  
  
void approx\_value(struct AppParams ap\_pr) {  
 float t[N], Uvx[N], Uvix[N];  
 float p = 1, par = 1e10, par1 = 0;  
 float dt = (ap\_pr.tk - ap\_pr.tn) / (ap\_pr.n - 1);  
  
 while (p > ap\_pr.eps && N > ap\_pr.n) {  
 form\_time(ap\_pr, t);  
 form\_Uvx(ap\_pr, t, Uvx);  
 form\_Uvix(ap\_pr, Uvx, Uvix);  
 par1 = parametr(ap\_pr.n, dt, Uvix);  
 p = fabs(par - par1) / fabs(par1);  
 par = par1;  
 ap\_pr.n \*= 2;  
 dt = (ap\_pr.tk - ap\_pr.tn) / (ap\_pr.n - 1);  
 }  
}

### src/include/app.h

#ifndef APP\_H  
#define APP\_H  
  
void run\_app(int count, char\* arg[]);  
  
#endif

### src/include/funct.h

#ifndef FUNCT\_H  
#define FUNCT\_H  
  
void form\_time(struct AppParams ap\_pr, float\* t);  
void form\_Uvx(struct AppParams ap\_pr, float\* Uvx, float\* t);  
void form\_Uvix(struct AppParams ap\_pr, float\* Uvx, float\* Uvix);  
  
#endif

### src/include/krnt.h

#ifndef KRNT\_H  
#define KRNT\_H  
  
void control\_calc(struct AppParams ap\_pr);  
  
#endif

### src/include/rpzt.h

#ifndef RPZT\_H  
#define RPZT\_H  
  
void approx\_value(struct AppParams ap\_pr);  
  
#endif

### src/include/globals.h

#ifndef GLOBALS\_H  
#define GLOBALS\_H  
  
#define N 15000  
  
struct AppParams {  
 int n;  
 float eps, a, b, tn, tk, t1;  
 float Uvx1, Uvx2, U1, U2;  
};  
  
#endif

## Приложение 2

### scripts/menu.sh

#!/bin/bash

clear # Очистка экрана

# Подключение вспомогательных скриптов с переменными и функциями

. ./scripts/helpers/variables.sh --source-only

. ./scripts/helpers/functions.sh --source-only

# Подключение вспомогательных частей программы

. ./scripts/helpers/p1.sh --source-only

. ./scripts/helpers/p2.sh --source-only

. ./scripts/helpers/file\_output.sh --source-only

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

N=15000 # Максимальное количество точек

#sed -i "5s/.\*/#define N $N/" src/include/globals.h

#make >/dev/null

# Функция вывода заставки

out\_zast(){

while read -r line; do

style "$line" $yellow # Цветной вывод строки

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

printf "\n\n"

}

# Функция отображения основного меню

out\_menu() {

while true; do

style "Меню программы:" $green

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

if [ "$indx" != "2" ]; then

style "${variant\_menu[${indx}]}" $yellow

elif [[ "$indx" == "2" && "${#t[@]}" -gt "0" ]]; then

style "${variant\_menu[${indx}]}" $i\_yellow

fi

done

echo

while true; do

# Определение доступных пунктов меню

if [ "${#t[@]}" -gt "0" ]; then con\_vr=3

else con\_vr=2; fi

style "Выберите действие 1-${con\_vr} и p или q для выхода " $blue n

read -rsn1 key # Чтение одного символа

printf "\n"

case $key in

1|2)

clear

style "Ведите n точек:" $yellow

style "Диапазон n: [2;${N}]" $yellow

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

style " Error: Число ($num) > $N" $red

fi

else

clear\_line

style " Error: Число ($num) < 2" $red

fi

done

n=$num # Сохранение введённого значения

;;&

2)

style "Ведите погрешность eps:" $yellow

style "Диапазон eps: [0.001; 99.99]%" $yellow

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 < 99.99

valid\_max=$(echo "$num < 99.99" | bc -l)

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

break

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

clear\_line

style " Ошибка: число ($num) > 99.99" $red

else

clear\_line

style " Ошибка: число ($num) < 0.0009" $red

fi

done

eps=$num # Сохранение значения

;;&

[1-2])

clear

style "Данне успешно переданны в программу!" $green

style "Данные из программы успешно считанны!" $green

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

;;&

3)

if [ "$con\_vr" == "3" ];then

out\_file # Запись результатов в файл

else

clear\_line

style "Erorr: Не верное значение ($key) не входит в промежуток [1;$con\_vr]!" $red

break

fi

;;&

p)

clear

style "Закройте окно для возврата в меню!" $yellow

xterm \

-geometry 80x31-20+5 \

-bg black \

-bd red \

-fg green \

-fa 'Ubuntu Mono' \

-fs 20 \

-e 'tput civis; ./scripts/xterm\_scripts/output\_data.sh; tput cnorm'

;;&

[1-$con\_vr]|p)

clear

out\_zast # Повторный вывод заставки

break

;;

q)

return # Завершение работы

;;

\*)

clear\_line

style "Erorr: Не верное значение ($key) не входит в промежуток [1;$con\_vr] и p!" $red

;;

esac

done

done

}

# Функция запуска программы

start() {

clear

inp\_data=() # Массив входных данных

out\_data=() # Массив выходных данных

out\_zast # Отображение заставки

out\_menu # Запуск главного меню

}

start # Старт программы

style "\nПрограмма успешно завершена" $green

i=3

while [ "$i" -gt "0" ];do

style "Консоль очистится через: $i" $yellow

let "i-=1"

sleep 0.3

done

clear

exit # Завершение

### scripts/helpers/file\_output.sh

#!/bin/sh

# Функция out\_file — записывает массивы t, Uvx и Uvix в отдельные текстовые файлы

# и запускает скрипт для построения графиков с помощью Maxima

out\_file() {

clear

style "Происходит запись в файл!" $green

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]}" &

{

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

echo "${Uvx[$i]}"

done

} > "${var\_file[1]}" &

{

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

echo "${Uvix[$i]}"

done

} > "${var\_file[2]}" &

prgs\_t $! "Запись в файл"

clear

style "Данные успешно записанны в файл!" $green

# Запуск Maxima-скрипта для построения графиков

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

prgs\_t $! "Генерация графиков"

clear

style "Графики успешно нарисованы!" $yellow

style "Вывести графики ? (y/n)" $blue n

read -rsn1 nn

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

style "\nЗакройте окно с графиками для продолжения!" $yellow

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

fi

}

### scripts/helpers/functions.sh

#!/bash/sh

style() {

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

style\_text=""

local nc=""

else

style\_text=$2

fi

if [ -z "$3" ];then

echo -e "${style\_text}${1}${nc}"

else

echo -ne "${style\_text}${1}${nc}"

fi

}

clear\_line() {

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

echo -ne "\007"

}

is\_number() {

re="$2"

num=0

style "$1" $blue n

while true

do

read num

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

clear\_line

style " ОШИБКА: '$num'-не является целым числом" $red

style "$1" $blue n

done

}

no\_space() {

num=0

style "$1" $blue n

while true

do

read num

if [ "$num" != " " ] && [ "$num" != "" ]; then break;fi

clear\_line

style "ОШИБКА: в поле либо пусто, либо в нем пробел!" $red

style "$1" $blue n

done

}

prgs\_bar() {

res=$(awk "BEGIN {print ($1 / $2) \* 100}")

int\_res=$(echo "scale=0; $res / 4" | bc)

sym\_beg=$4

sym\_end=$5

if [ "$int\_res" == "0" ];then echo -ne "\e[?25l";fi

if [ "$sym\_beg" == "" ];then sym\_beg="#";fi

if [ $int\_res -gt "0" ]

then str="$(printf '%.0s'"$sym\_beg" $(seq 1 ${int\_res}))$(printf '%.0s'"$sym\_end" $( seq 1 $((25-$int\_res)) ) )"

else str="$(printf '%.0s'"$sym\_end" $( seq 1 $((25-$int\_res)) ) )";fi

printf "\r${yellow}%s%-25s%s %.2f%%${nc}" "$3 [" "$str" "]" "$res"

if [ "$int\_res" == "25" ];then echo -ne "\e[?25h";fi

}

prgs\_t() {

i=0

echo -ne "\e[?25l"

while kill -0 ${1} 2>/dev/null;do

printf "\r${yellow}%20s${nc}" "${2}$(printf '%.0s.' $(seq 1 $i)) "

sleep 0.2

let "i+=1"

if [ "$i" == "4" ];then i=1;fi

done

echo -ne "\e[?25h"

}

### scripts/helpers/variables.sh

#!/bash/sh

p\_blue="$(tput setaf 6)"

p\_res="$(tput sgr0)"

red="\033[0;31m"

blue="\033[0;34m"

green="\033[0;32m"

yellow="\033[0;33m"

i\_yellow='\033[93m'

bold="\033[1m"

nc="\033[0m"

alfv="abcdefghijklmnopqrstuvwxyz"

variant\_menu=(

"1 — Контрольный расчет для n точек"

"2 — Расчёт параметра с заданной точностью"

"3 — Запись данных в файлы"

"p - Вывод пояснений к параметрам"

"q — Выход из программы"

)

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

def\_data=(

12 # a

48 # b

10 # tn

35 # tk

30 # t1

)

### scripts/helpers/p1.sh

#!/bin/sh

# 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/myapp ${inp\_data[@]})"

style "Результат программы: " $yellow # Заголовок результата

# Чтение заголовка таблицы

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

printf "\n ${yellow}%7s %12s %14s${nc}\n" " ${header[0]}" "${header[1]}" "${header[2]}"

printf "%7s %12s %14s\n" "${header[0]}" "${header[1]}" "${header[2]}" > "data/tabls/table\_p2.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 }")

# num=$(echo "${arr[2]} \* 100" | bc -l) # Преобразование в проценты

# Печать строки в консоль

printf " ${yellow}%6d${nc} %10.3f %12f%%\n" \

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

# Запись строки в файл

printf "%7d %10.3f %12f%%\n" \

"${arr[0]}" "${arr[1]}" "${num}" >> "data/tabls/table\_p2.txt"

# Прекращение при достижении половины массива

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

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

break

fi

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

style "-> enter для окончания просмотра" $yellow n

read

clear # Очистка экрана

}

ghost@DigmaEVEP4851:~/unvr/pr\_xz/kursach/program$ cat scripts/helpers/p1.sh

#!/bin/sh

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

pg1() {

out\_data=() # Очистка массива выходных данных

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

sleep 1000 &

pig=$!

echo

prgs\_t $pig "Работа программы" &

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/myapp ${inp\_data[@]})" # Вызов внешней программы и обработка её вывода

kill $pig > /dev/null

wait

check\_out="y"

style "\rЖелаете ли вывести таблицу ? (y/n) " $blue n

read -rsn1 check\_out # Чтение ответа пользователя без вывода на экран

if [ "$check\_out" == "y" ]; then

style "\nРезультат программы: " $yellow

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

# Печать заголовка таблицы в консоль

printf "\n ${yellow}%-7s %8s %10s %9s${nc}\n" " №" "t" "Uvx" "Uvix"

# Запись заголовка таблицы в файл

printf "%-4s %7s %9s %8s\n" "№" "t" "Uvx" "Uvix" > "data/tabls/table\_p1.txt"

# Печать и запись каждой строки таблицы

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

printf " ${yellow}%5d${nc} %9.1f %9.1f %9.1f\n" \

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

printf "%5d %9.1f %9.1f %9.1f\n" \

"$((i+1))" "${t[$i]}" "${Uvx[$i]}" "${Uvix[$i]}" >> "data/tabls/table\_p1.txt"

done

style "\n-> enter для окончания просмотра" $yellow n

read

fi

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

printf "%4d %8.1f %8.1f %8.1f\n" \

"$((i+1))" "${t[$i]}" "${Uvx[$i]}" "${Uvix[$i]}" >> "data/tabls/table\_p1.txt"

done

clear # Очистка экрана

}

### scripts/helpers/p2.sh

#!/bin/sh

# 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/myapp ${inp\_data[@]})"

style "Результат программы: " $yellow # Заголовок результата

# Чтение заголовка таблицы

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

printf "\n ${yellow}%7s %12s %14s${nc}\n" " ${header[0]}" "${header[1]}" "${header[2]}"

printf "%7s %12s %14s\n" "${header[0]}" "${header[1]}" "${header[2]}" > "data/tabls/table\_p2.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 }")

# num=$(echo "${arr[2]} \* 100" | bc -l) # Преобразование в проценты

# Печать строки в консоль

printf " ${yellow}%6d${nc} %10.3f %12f%%\n" \

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

# Запись строки в файл

printf "%7d %10.3f %12f%%\n" \

"${arr[0]}" "${arr[1]}" "${num}" >> "data/tabls/table\_p2.txt"

# Прекращение при достижении половины массива

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

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

break

fi

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

style "-> enter для окончания просмотра" $yellow n

read

clear # Очистка экрана

}

## Приложение 3

### scripts/Wxmax\_scr/make\_graphs.mac

/\* Загрузка массивов \*/

t : read\_list("data/massiv\_t.txt")$

Uvix : read\_list("data/massiv\_Uvix.txt")$

Uvx : read\_list("data/massiv\_Uvx.txt")$

/\* Общая настройка вывода PNG через cairo + шрифт \*/

set\_plot\_option([gnuplot\_term, pngcairo])$

/\* ---------- График Uvx(t) ---------- \*/

set\_plot\_option([gnuplot\_out\_file, "data/graphs/graph\_Uvx.png"])$

set\_plot\_option([gnuplot\_preamble,

"set grid; \

set title 'График зависимости Uvx от времени t' font 'Arial,14'; \

set xlabel 't' font 'Arial,12'; \

set ylabel 'Uvx' font 'Arial,12';"])$

plot2d(

[discrete, makelist([t[i], Uvx[i]], i, 1, length(t))],

[style, linespoints]

)$

/\* ---------- График Uvix(t) ---------- \*/

set\_plot\_option([gnuplot\_out\_file, "data/graphs/graph\_Uvix.png"])$

set\_plot\_option([gnuplot\_preamble,

"set grid; \

set title 'График зависимости Uvix от времени t' font 'Arial,14'; \

set xlabel 't' font 'Arial,12'; \

set ylabel 'Uvix' font 'Arial,12';"])$

plot2d(

[discrete, makelist([t[i], Uvix[i]], i, 1, length(t))],

[style, linespoints]

)$