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(ка):

Кучменко Н.И. \_\_\_\_\_\_\_\_

*(Ф.И.О., № группы) (подпись)*

Дата выполнения:

«25» Апрель 2025г

Проверил:

Хазиев Н.Н. \_\_\_\_\_\_\_\_

*(Ф.И.О.) (подпись)*

Санкт-Петербург

2025

Оглавление

[Задание к курсовой работе 3](#_Toc197972054)

[Аннотация 4](#_Toc197972055)

[Техническая реализация: 4](#_Toc197972056)

[Процесс взаимодействия: 4](#_Toc197972057)

[Входные данные: 4](#_Toc197972058)

[Выходные данные: 4](#_Toc197972059)

[Области применения: 5](#_Toc197972060)

[Контрольный расчёт 6](#_Toc197972061)

[Таблица идентификаторов 7](#_Toc197972062)

[Описание функций 8](#_Toc197972063)

[Описание логической структуры 10](#_Toc197972064)

[Код программы 14](#_Toc197972065)

[Си: 14](#_Toc197972066)

[Bash: 19](#_Toc197972067)

[Графики (обработка полученных результатов, выводы) 32](#_Toc197972068)

[Анализ графиков зависимостей Uвх и Uвых от времени 32](#_Toc197972069)

[Заключение 34](#_Toc197972070)

[Список используемой литературы 35](#_Toc197972071)

# 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

Данная работа посвящена разработке программного обеспечения для численного анализа параметров функций с использованием языка программирования C. Программа выполняет как точные, так и приближенные вычисления значений функции с заданной пользователем точностью (погрешностью не более 1%).

**Основные функциональные возможности:**

1. **Два режима работы:**
   * Прямой расчет параметров функции по заданным коэффициентам и временному диапазону
   * Приближенный расчет с учетом указанной пользователем погрешности
2. **Реализованные функции:**
   * Формирование массивов времени и значений функции
   * Автоматический вывод и сохранение результатов
   * Табличное представление данных

# 1. Общие сведения

## 1.1. Обозначение и наименование программы

**Полное название:** «Анализ сигнала на выходе электрической цепи»  
**Сокращённое название:** АСВЭЦ, программа  
**Язык интерфейса и документации:** русский

## 1.2. Требуемое программное обеспечение

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:

sudo apt install xterm

sudo apt install eog

## 1.3. Языки разработки

Программа написана на языке программирования **Си** — на нём реализована основная функциональность.  
Меню реализовано с использованием **Bash-скриптов**, которые также запускают скрипт для **wxMaxima**, оформленный в виде текстового файла с расширением .mac.

# 2. Функциональное назначение

## 2.1. 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.

### **Численный анализ:**

* Определение следующих параметров сигнала с заданной точностью:
  + длительность импульса;
  + длительность спада импульса;
  + время достижения входным напряжением значения 80 В;
  + момент времени, когда входное напряжение достигает максимума;
* Автоматическая корректировка количества расчётных точек **N** для обеспечения требуемой точности **eps**.

### **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).

## 2.2. Назначение

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.

## 2.3. 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.

# 3. Контрольный расчёт

|  |  |
| --- | --- |
| Контрольный расчет для n точек | Параметры |
|  | При кол-во контрольных точек n=25 |

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

# 4. Используемые технические средства

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

Таблица 4.1. 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 | - |

# 5. Таблица идентификаторов

|  |  |
| --- | --- |
| a |  |
| b |  |
| tn |  |
| t1 |  |
| tk |  |
| U1 |  |
| U2 |  |
| Uvx |  |
| Uvix |  |
| n | Количество элементов в массиве |
| eps | Погрешность |
| p | Параметр |
| t | Массив времени |
| Uvx | Массив входного напряжение |
| Uvix | Массив выходного напряжение |

# 6. Описание 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. |

# 7. Описание логической структуры

Программа состоит из двух основных частей:

1. **С++ приложение (src)**, которое выполняет математические расчеты и выводит таблицы значений.
2. **Bash-скрипты (scripts)**, которые управляют запуском приложения, меню и записью результатов.

**1. Приложение на C (src)**

**Основные модули:**

* main.c — Точка входа. В зависимости от аргументов командной строки вызывает одну из двух функций:
  + control\_calc() — Прямой расчет.
  + approx\_value() — Приближенный расчет с заданной точностью.
* globals.h — Заголовочный файл с глобальной структурой AppParams, содержащей параметры работы программы.
* app.h — Заголовочный файл с прототипом функции run\_app().
* funct.h — Заголовочный файл с прототипами функций, выполняющих математические операции:
  + form\_time() — Формирование массива времени.
  + form\_Uvx() — Расчет промежуточных значений.
  + form\_Uvix() — Расчет конечных значений по заданному закону.
  + form\_tabl1() — Вывод таблицы результатов.
  + parametr() — Расчет среднего значения.

**Логика работы приложения:**

1. **Обработка аргументов командной строки:**
   * Первый аргумент (arg[1]) — режим работы (1 — прямой расчет, 2 — приближенный).
   * Второй аргумент (arg[2]) — количество точек (n).
   * Третий аргумент (arg[3]) — точность (eps) (только для приближенного метода).
2. **Инициализация структуры параметров (AppParams), которая содержит:**
   * Параметры уравнения (a0, a1, a2).
   * Начальное (tn) и конечное (tk) время.
   * Пороговое значение напряжения (Uvx1).
   * Количество точек (n) и точность (eps).
3. **В зависимости от режима работы вызывается одна из функций:**
   * control\_calc():
     + Вычисляет значения времени, промежуточных (Uvx) и конечных (Uvix) значений.
     + Выводит результаты в виде таблицы.
   * approx\_value():
     + Вычисляет параметр (parametr) до достижения заданной точности (eps).
     + Выводит таблицу с шагами расчета и точностью.

**2. Bash-скрипты (scripts)**

**Основные компоненты:**

* start.sh — Главный скрипт запуска программы.
* functions.sh — Функции для форматирования текста и проверки ввода.
* variables.sh — Глобальные переменные (цвета текста, меню).
* file\_output.sh — Запись результатов в файлы.
* output\_data.sh — Построение графиков с помощью Maxima.
* p1.sh и p2.sh — Обработчики для двух режимов работы программы.

**Логика работы скриптов:**

1. **Старт программы (start()):**
   * Выводит меню с вариантами действий:
     + 1) Контрольный расчет для n точек.
     + 2) Расчет параметра с заданной точностью.
     + 3) Запись данных в файлы.
     + p) Пояснение параметров.
     + q) Выход из программы.
2. **Выбор действия пользователем:**
   * Ввод количества точек (n) и точности (eps).
   * Вызов соответствующей функции:
     + pg1() — Запуск приложения с режимом 1 (прямой расчет).
     + pg2() — Запуск приложения с режимом 2 (приближенный расчет).
3. **Обработка вывода приложения:**
   * В режиме 1 (pg1) программа считывает три строки:
     + Временные точки (t).
     + Промежуточные значения (Uvx).
     + Конечные значения (Uvix).
   * В режиме 2 (pg2) программа считывает строки с шагами расчета и погрешностями.
4. **Запись результатов в файлы (out\_file()):**
   * Три файла данных:
     + massiv\_t.txt — Временные точки.
     + massiv\_Uvx.txt — Промежуточные значения.
     + massiv\_Uvix.txt — Конечные значения.
   * Запуск Maxima для построения графиков.

**3. Общая схема работы**

1. Пользователь запускает программу (start()).
2. Программа выводит меню.
3. Пользователь выбирает один из двух режимов:
   * Прямой расчет (pg1()).
   * Приближенный расчет (pg2()).
4. Введенные параметры передаются приложению на C.
5. Приложение выполняет расчеты и возвращает результаты:
   * Временные точки (t).
   * Промежуточные значения (Uvx).
   * Конечные значения (Uvix).
6. Результаты сохраняются в текстовые файлы.
7. Пользователь может построить графики или выйти из программы.

# Вызов и загрузка

## 8.1 Способ вызова программы с соответствующего носителя данных

Программа запускается вручную через терминал Linux следующим образом:

make run

Или двойным кликом по файлу menu.sh.

Также возможен прямой вызов программы без меню, в случае его неработоспособности:

Pg – это выбор контрольного расчёта/расчёт параметра

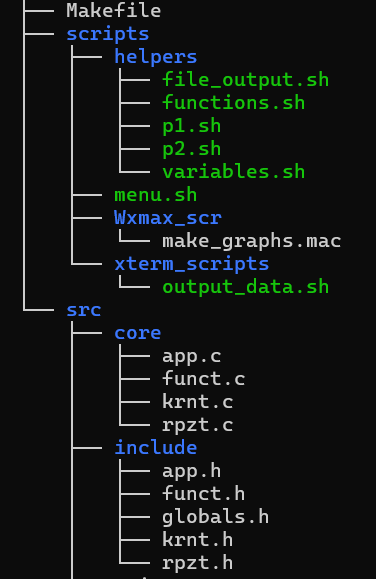
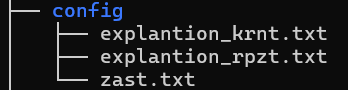
n – кол-во элементов в массиве

eps – предел точности погрешности

./myapp pg n eps

## 8.2 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.

# Код программы

## Си:

#include "app.h"

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

run\_app(count, arg);

return 0;

}

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

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

#include "rpzt.h" // Заголовочный файл, предположительно с функцией control\_calc()

#include "krnt.h" // Заголовочный файл, предположительно с функцией approx\_value()

// Главная управляющая функция приложения

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

// Инициализация структуры параметров приложения

struct AppParams ap\_pr = {

.a = 12, // Начальное значение a

.b = 12, // Конечное значение b

.tn = 10, // Начальное время tn

.tk = 35, // Конечное время tk

.t1 = 22.5, // Временная точка t1

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

.Uvx2 = 25, // Значение напряжения во второй точке

.U1 = 20, // Значение напряжения U1

.U2 = 150, // Значение напряжения U2

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

};

// 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 = atof(arg[3]) / 100; // Перевод из процентов в дробное значение

approx\_value(ap\_pr);

break;

}

}

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

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

t[i] = ap\_pr.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 (t[i] < ap\_pr.t1) {

// До t1 — линейный рост

Uvx[i] = ap\_pr.a \* (t[i] - ap\_pr.tn);

} else {

// После t1 — линейное убывание

Uvx[i] = ap\_pr.a \* (ap\_pr.t1 - ap\_pr.tn) - ap\_pr.b \* (t[i] - ap\_pr.t1);

}

}

}

// 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 dt, float \*U) {

float Umin = U[0], Umax = U[0];

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

if (U[i] < Umin) Umin = U[i];

if (U[i] > Umax) Umax = U[i];

}

float threshold = Umin + 0.5 \* (Umax - Umin); // Порог = середина диапазона

float duration = 0;

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

if (U[i] >= threshold) {

duration += dt; // Считаем длительность превышения порога

}

}

return duration;

}

// 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("%.3g ", t[i]);

else printf("%.3g\n", t[i]);

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

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

else printf("%.3g\n", Uvx[i - n]);

} else {

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

else printf("%.3g\n", Uvix[i - n \* 2]);

}

}

}

#include "globals.h" // Заголовочный файл с определением структуры AppParams и констант

#include "krnt.h" // Заголовок с прототипом control\_calc

#include "funct.h" // Прототипы функций для расчёта и формирования таблиц

// Основная управляющая функция для варианта 1 — расчёт и вывод таблицы

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

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

}

#include "stdio.h" // Для printf

#include "globals.h" // Структура AppParams и глобальные константы

#include "rpzt.h" // Прототип approx\_value

#include "funct.h" // Прототипы функций расчёта

#include "math.h" // Для fabs

// Функция приближённого расчёта значения параметра с заданной точностью

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

float t[N], Uvx[N], Uvix[N]; // Массивы времени и напряжений

float p = 1; // Начальное значение погрешности

float par = 1e10; // Предыдущее значение параметра (большое значение для старта)

float par1 = 0; // Текущее значение параметра

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

printf("n parametr pogrechnost\n"); // Заголовок таблицы

// Цикл итеративного уточнения параметра до достижения заданной погрешности

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

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

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

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

par1 = parametr(ap\_pr.n, dt, Uvix); // Вычисление параметра

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

if (p > 1) p = 1; // Защита от взрыва значения p

printf("%d %g %g\n", ap\_pr.n, par1, p); // Вывод строки результата итерации

par = par1; // Обновление предыдущего значения параметра

ap\_pr.n = 2 \* ap\_pr.n; // Увеличение количества точек в 2 раза

dt = (ap\_pr.tk - ap\_pr.tn) / (ap\_pr.n - 1); // Пересчёт шага

}

}

#include "app.h"

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

run\_app(count, arg);

return 0;

}

ghost@DigmaEVEP4851:~/unvr/pr\_xz/kursach/program$ cat src/core/\*

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

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

#include "rpzt.h" // Заголовочный файл, предположительно с функцией control\_calc()

#include "krnt.h" // Заголовочный файл, предположительно с функцией approx\_value()

// Главная управляющая функция приложения

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

// Инициализация структуры параметров приложения

struct AppParams ap\_pr = {

.a = 12, // Начальное значение a

.b = 12, // Конечное значение b

.tn = 10, // Начальное время tn

.tk = 35, // Конечное время tk

.t1 = 22.5, // Временная точка t1

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

.Uvx2 = 25, // Значение напряжения во второй точке

.U1 = 20, // Значение напряжения U1

.U2 = 150, // Значение напряжения U2

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

};

// 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 = atof(arg[3]) / 100; // Перевод из процентов в дробное значение

approx\_value(ap\_pr);

break;

}

}

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

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

t[i] = ap\_pr.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 (t[i] < ap\_pr.t1) {

// До t1 — линейный рост

Uvx[i] = ap\_pr.a \* (t[i] - ap\_pr.tn);

} else {

// После t1 — линейное убывание

Uvx[i] = ap\_pr.a \* (ap\_pr.t1 - ap\_pr.tn) - ap\_pr.b \* (t[i] - ap\_pr.t1);

}

}

}

// 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 dt, float \*U) {

float Umin = U[0], Umax = U[0];

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

if (U[i] < Umin) Umin = U[i];

if (U[i] > Umax) Umax = U[i];

}

float threshold = Umin + 0.5 \* (Umax - Umin); // Порог = середина диапазона

float duration = 0;

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

if (U[i] >= threshold) {

duration += dt; // Считаем длительность превышения порога

}

}

return duration;

}

// 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("%.3g ", t[i]);

else printf("%.3g\n", t[i]);

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

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

else printf("%.3g\n", Uvx[i - n]);

} else {

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

else printf("%.3g\n", Uvix[i - n \* 2]);

}

}

}

#include "globals.h" // Заголовочный файл с определением структуры AppParams и констант

#include "krnt.h" // Заголовок с прототипом control\_calc

#include "funct.h" // Прототипы функций для расчёта и формирования таблиц

// Основная управляющая функция для варианта 1 — расчёт и вывод таблицы

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

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

}

#include "stdio.h" // Для printf

#include "globals.h" // Структура AppParams и глобальные константы

#include "rpzt.h" // Прототип approx\_value

#include "funct.h" // Прототипы функций расчёта

#include "math.h" // Для fabs

// Функция приближённого расчёта значения параметра с заданной точностью

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

float t[N], Uvx[N], Uvix[N]; // Массивы времени и напряжений

float p = 1; // Начальное значение погрешности

float par = 1e10; // Предыдущее значение параметра (большое значение для старта)

float par1 = 0; // Текущее значение параметра

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

printf("n parametr pogrechnost\n"); // Заголовок таблицы

// Цикл итеративного уточнения параметра до достижения заданной погрешности

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

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

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

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

par1 = parametr(ap\_pr.n, dt, Uvix); // Вычисление параметра

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

if (p > 1) p = 1; // Защита от взрыва значения p

printf("%d %g %g\n", ap\_pr.n, par1, p); // Вывод строки результата итерации

par = par1; // Обновление предыдущего значения параметра

ap\_pr.n = 2 \* ap\_pr.n; // Увеличение количества точек в 2 раза

dt = (ap\_pr.tk - ap\_pr.tn) / (ap\_pr.n - 1); // Пересчёт шага

}

}

ghost@DigmaEVEP4851:~/unvr/pr\_xz/kursach/program$ cat src/include/\*

#ifndef APP\_H

#define APP\_H

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

#endif

#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);

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

float parametr(int n, float dt, float \*U);

#endif

#ifndef GLOBALS\_H

#define GLOBALS\_H

// Максимальное количество точек (для массивов)

#define N 15000

// Структура параметров приложения

struct AppParams {

int n; // Количество точек разбиения

float eps; // Допустимая погрешность (для приближённого метода)

float a, b; // Коэффициенты линейных участков функции Uvx

float tn, tk, t1; // Начальное, конечное время и момент излома

float Uvx1, Uvx2; // Пороговые значения для функции Uvix

float U1, U2; // Соответствующие значения выходного напряжения

};

#endif

#ifndef KRNT\_H

#define KRNT\_H

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

#endif

#ifndef RPZT\_H

#define RPZT\_H

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

#endif

## Bash:

#!/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

#!/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

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

#!/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

}

#!/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"

}

#!/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

}

#!/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

}

#!/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

)

#!/bin/bash

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

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

out\_file\_name=(

"./config/explantion\_krnt.txt"

"./config/explantion\_rpzt.txt"

)

clear

for file\_name in "${out\_file\_name[@]}";do

while IFS= read -r line;do

style "$line" $yellow

done < "$file\_name"

if [ "$file\_name" != "${out\_file\_name[-1]}" ];then

style "Нажмите enter чтоб перелестнуть страницу!" $blue n

else

style "Нажмите enter чтоб закончить просмотр!" $blue n

fi

read

clear

done

## Wxmaxima:

/\* 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]

)$

# Входные данные

### **6.1. Характер и организация входных данных**

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

#### **1. 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.

#### **2. 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.

### **6.2. Формат и кодирование входных данных**

* Все входные значения обрабатываются как **числа с плавающей точкой** типа float (одинарной точности).
* Ввод осуществляется через **консоль**, где данные проверяются на корректность (целочисленность, диапазоны).

#### **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).

# ****11. Выходные данные****

### **7.1. Характер и организация выходных данных**

Программа формирует два основных вида выходных данных:

#### **1. 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).

#### **2. 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.

### **7.2. 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

5.400000

...

#### **Пример консольного вывода таблицы:**

№ t Uvx Uvix

1 10.0 0.0 20.0

2 10.9 1.8 20.0

...

# Графики (обработка полученных результатов, выводы)

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

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

## Анализ графиков зависимостей Uвх и Uвых от времени

**1. График входного сигнала (Uvix):**

* Наблюдается **нелинейный рост** напряжения от ~20V до 140V
* Основное изменение происходит в диапазоне 15-25 сек
* После 25 сек сигнал выходит на **плато** (стабилизируется на уровне ~140V)

**2. График выходного сигнала (Uvx):**

* Демонстрирует **резонансную характеристику** с пиком ~120V при 20 сек
* Наблюдается **фазовый сдвиг** относительно входного сигнала
* После 25 сек происходит **быстрое затухание** до нулевого уровня

**Ключевые выводы:**

1. Цепь обладает **инерционными свойствами** (запаздывание реакции)
2. На частоте ~20 сек проявляется **резонансный эффект**
3. Система **не пропускает постоянную составляющую** (выход → 0 при установившемся входе)
4. Максимальное **усиление** наблюдается в переходном режиме (~120V при 140V вход)

**Техническая интерпретация:**

* Такая характеристика типична для:
  + RLC-цепей с малым затуханием
  + Полосовых фильтров
  + Резонансных трансформаторов

# Заключение

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

**Основные этапы работы:**

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

**Приобретенные навыки:**  
✔ Глубокое 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 можно адаптировать для решения более сложных задач в области электротехники и схемотехники. Особенно ценным оказался опыт создания комплексного решения, объединяющего математические расчеты, программирование и визуализацию данных.

Полученные знания и навыки имеют практическую ценность и могут быть применены как в учебной деятельности, так и в профессиональной работе, связанной с проектированием и анализом электронных схем.

# 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/