

# 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 – 1 вариант

Входной сигнал – 1 вариант

Выполнил студент группы ИКПИ-41:

Акулаева А.А. \_\_\_\_\_

Дата выполнения: «29» Май

Проверил:

Хазиев Н.Н. \_\_\_\_\_

Санкт-Петербург

2025

## Оглавление

<b>Аннотация</b>	<b>4</b>
<b>Задание к курсовой работе</b>	<b>5</b>
<b>1 Общие сведения</b>	<b>6</b>
1.1 Обозначение и наименование программы	6
1.2 Языки разработки:	6
<b>2 Функциональное назначение</b>	<b>6</b>
2.1 Решаемые задачи	6
2.2 Назначение:	7
2.3 Ограничения в функциональности:	7
<b>3 Таблица идентификаторов</b>	<b>9</b>
<b>4 Описание логической структуры</b>	<b>10</b>
4.1 Алгоритм программы	10
4.2 Описание функций	11
4.3 Используемые методы	12
4.4 Связи программы с другими программами	12
<b>5 Используемые технические средства</b>	<b>14</b>
<b>6 Вызов и загрузка</b>	<b>15</b>
6.1 Способ вызова программы с соответствующего носителя данных	15
6.2 Входные точки в программу	15
<b>7 Входные данные</b>	<b>16</b>
7.1 Характер и организация входных данных	16
7.2 Формат и кодировка входных данных	17
<b>8 Выходные данные</b>	<b>18</b>
8.1 Характер и организация выходных данных	18
8.2 Контрольный расчёт	19
8.3 Формат и кодирование выходных данных	20

<b>9</b>	<b>Структура кода</b>	<b>22</b>
<b>10</b>	<b>Заключение</b>	<b>23</b>
<b>11</b>	<b>Список используемой литературы</b>	<b>25</b>
<b>12</b>	<b>Сокращения</b>	<b>26</b>
<b>13</b>	<b>Приложения</b>	<b>28</b>
13.1	<i>Приложение 1</i>	28
a	main.c	28
b	app.c	28
c	funct.c	29
d	Заголовочные файлы	33
13.2	<i>Приложение 2</i>	34
13.3	<i>Приложение 3</i>	47

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

#### **Ключевые алгоритмы:**

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

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

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

**Задание к курсовой работе**

Работа посвящена решению задач машинного анализа электрических цепей.

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

**Входной сигнал**

$$U_{\text{ВХ}}(t) = U_0 - U \cdot \sin t$$

**Рабочий набор**

$$U_0 = 2\text{В}; U = 3\text{В};$$

$$\underline{t_{\text{нач}} = \pi\text{с}; t_{\text{кон}} = 2\pi\text{с};}$$

**Выходной сигнал**

$$\underline{U_{\text{ВЫХ}} = a U_{\text{ВХ}}}$$

**Рабочий набор**

$$\underline{a = 3,5}$$

— 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.

## **1 Общие сведения**

### **1.1 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;**

### **1.2 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  $U_{вх}(t)$ , заданной в аналитической форме;
- вычисление выходного напряжения  $U_{вых}(t)$ , используя кусочно-линейную передаточную характеристику.

#### **Графическая визуализация:**

- построение графиков  $U_{вх}(t)$  и  $U_{вых}(t)$  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**.  
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:  $t \in [\pi, 2\pi]$  (можно изменить вручную в коде);
  - параметры цепи заданы для варианта №1;
  - максимальное количество точек **Nmax = 10 000** (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.

Акулаева Алёна ИКПИ-41

– **Примечание:** при необходимости использовать программу для других параметров, требуется вручную изменить настройки в исходном файле `funct.c`.



**3 Таблица идентификаторов**

Таблица 3.1:

**Таблица идентификаторов**

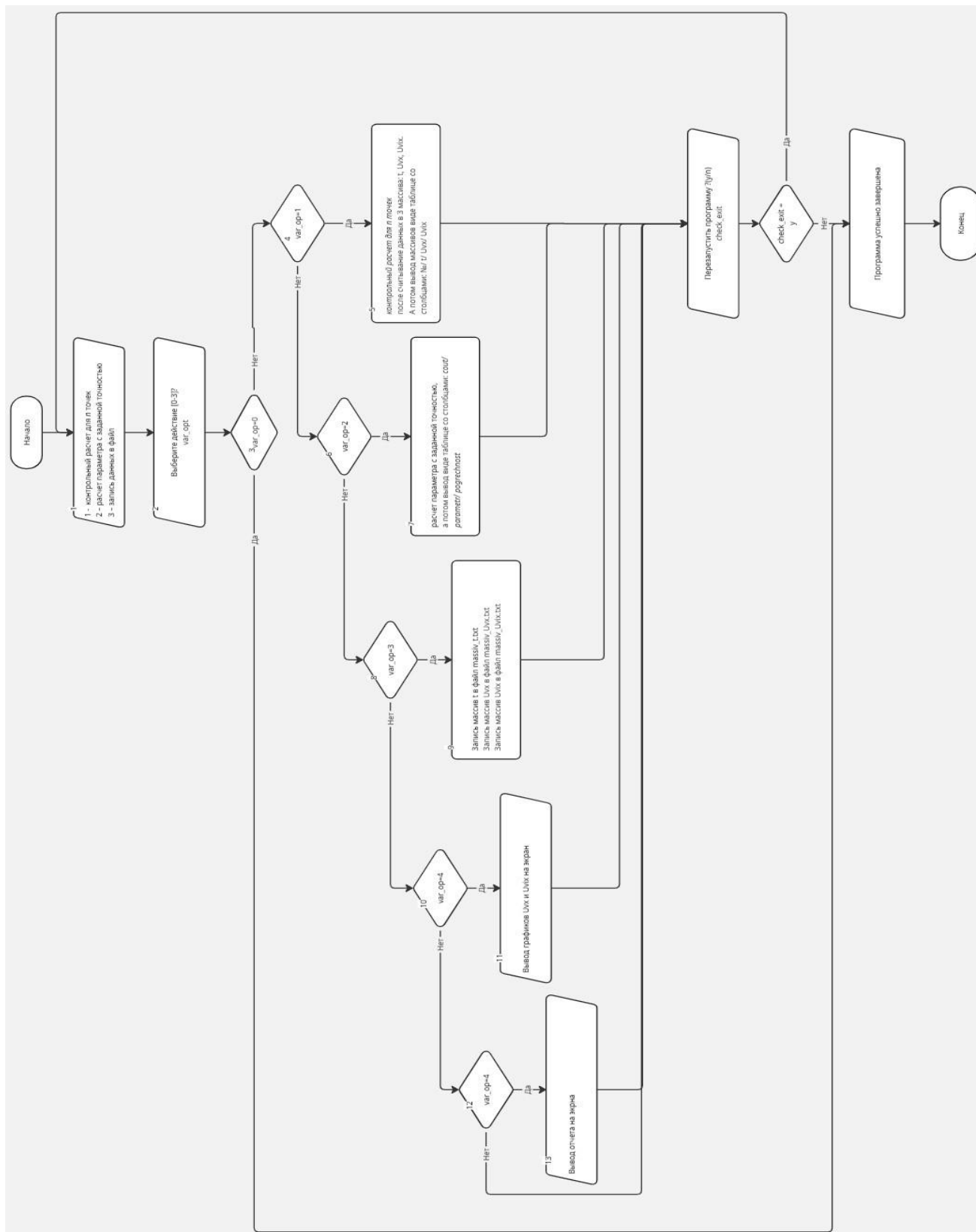
Переменная	Тип	Описание
<b>n</b>	int	Количество точек разбиения
<b>eps</b>	float	Допустимая погрешность (для приближённого метода)
<b>U0, U</b>	float	Коэффициенты линейных участков функции $U_{vx}$
<b>tn</b>	float	Начальное время
<b>tk</b>	float	Конечное время
<b>a</b>	float	Пороговые значения для функции $U_{vix}$
<b>t[N]</b>	float	Массив временных точек
<b>Uvx[N]</b>	float	Массив значений входной функции
<b>Uvix[N]</b>	float	Массив значений выходной функции

## 4 Описание логической структуры

### 4.1 Алгоритм программы

Рисунок 4.1.1:

Блок схемы



## 4.2 Описание функций

Таблица 4.2.1:

Таблица функций

Функция	Описание
<b>clear_line()</b>	Очищает текущую строку в терминале и воспроизводит звуковой сигнал
<b>is_number()</b>	Проверяет, является ли введённое значение числом, соответствующим регулярному выражению
<b>ts1()</b>	Выполняет контрольный расчёт для n точек, выводит таблицу значений t, Uvx, Uvix и сохраняет их в файл
<b>ts2()</b>	Выполняет расчёт параметра с заданной погрешностью eps, выводит таблицу с погрешностью и сохраняет её в файл
<b>out_zast()</b>	Выводит заставку (ASCII-арт или логотип) из файла ./config/zast.txt
<b>Основной цикл while</b>	Управляет меню программы, обрабатывает выбор пользователя и вызывает соответствующие функции
<b>run_app()</b>	Главная функция приложения, инициализирует параметры и вызывает нужный метод (control_calc, approx_value, file_out_data)
<b>form_time()</b>	Заполняет массив t временными точками с равномерным шагом от tn до tk
<b>form_Uvx()</b>	Вычисляет массив Uvx по кусочно-линейному закону с изломами в точках t1, t2, t3, t4
<b>form_Uvix()</b>	Вычисляет массив Uvix как линейное преобразование Uvx ( $Uvix[i] = 2.5 * Uvx[i] + 10$ )
<b>parametr()</b>	Вычисляет среднее значение массива U (используется в approx_value)
<b>paramtrs()</b>	Выводит длительности импульса сигнала, длительности заднего фронта импульса сигнала, Длительность заднего фронта импульса, Момент времени, когда Uvx достигает 80 В
<b>form_tabl1()</b>	Выводит таблицу значений t, Uvx, Uvix в три строки (для control_calc)
<b>control_calc()</b>	Выполняет точный расчёт, заполняет массивы t, Uvx, Uvix и выводит таблицу
<b>approx_value()</b>	Выполняет итеративный расчёт с удвоением n до достижения заданной погрешности eps

Функция	Описание
<code>file_out_data()</code>	Сохраняет массивы $t$ , $U_{vx}$ , $U_{vix}$ в отдельные файлы ( <code>massiv_t.txt</code> , <code>massiv_Uvx.txt</code> , <code>massiv_Uvix.txt</code> )

### 4.3 Используемые методы

Программа АСВЭЦ использует следующие методы:

– **численные расчёты:**

- разбивает интервал  $t \in [\pi, 2\pi]$  на  $N$  точек с шагом  $\Delta t = \pi / (N - 1)$ ;
- вычисляет значение сигнала по формуле  $U_{vx}(t) = U_0 - U \cdot \sin(t)$  для каждой точки  $t[i]$ ;
- используются функции `pow()` из библиотеки `math.h`;

– **применяет передаточную характеристику цепи через условные операторы:**

- итерационный метод уточнения (начинает с  $N = 20$  точек, сравнивает параметр с предыдущим расчётом, если погрешность больше `eps`, удваивает  $N$  и повторяет расчёт);
- методы визуализации (построение графиков скриптов с `wxMaxima`);
- взаимодействие с ОС (осуществляется через Bash-скрипт).

### 4.4 Связи программы с другими программами

Программа АСВЭЦ в ходе своей работы запускает следующие программы с помощью `bash`:

– `wxMaxima`:

- используется для построения графиков;
- должна быть установлена в системе;

– `open`:

- используется для вывода графиков и отчета.

– `gcc`:

- обязателен для компиляции кода.

– `GNU Make`:

- используется для компиляции си кода через `gcc`;
- используется для вызова `bash` скрипта командой:  
`bash /scripts/menu.sh`

Программа работает в консоли, все вычисления выполняются последовательно, `wxMaxima` и `gcc` не являются частью программы.

Акулаева Алёна ИКПИ-41

Программа также ориентирована на академические расчёты и локальное использование на Linux-системах.

## 5 Используемые технические средства

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

Таблица 5.1:

**Требования к техническим средствам**

<b>Компонент</b>	<b>Минимальные характеристики</b>	<b>Рекомендуемые характеристики</b>
<b>Процессор</b>	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)
<b>Оперативная память</b>	Не менее 512 МБ	От 2 ГБ и выше (особенно при расчётах с числом точек $N > 100000$ )
<b>Жёсткий диск</b>	Свободное место не менее 10 МБ	SSD-диск, не менее 100 МБ свободного пространства для ускоренного доступа к файлам
<b>Операционная система</b>	Ubuntu 20.04+, Debian 10+ или совместимые дистрибутивы Linux	Astra Linux или дистрибутивы Linux с GUI (например, GNOME, KDE)
<b>Дополнительное ПО</b>	- GCC версии не ниже 9.3.0- wxMaxima версии не ниже 20.06	- GCC версии 12 и выше- wxMaxima версии 23.04 и выше
<b>Монитор</b>	Разрешение экрана не менее 1280×720	Разрешение экрана Full HD (1920×1080)
<b>Графическая карта</b>	Интегрированная, не ниже Intel HD Graphics 4000	-

## **6 Вызов и загрузка**

### **6.1 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:
  - \_ `./prg ts n eps`:
    - `ts` – это выбор контрольного расчёта/расчёт параметра;
    - `n` – кол-во элементов в массиве;
    - `eps` – предел точности погрешности.

### **6.2 Входные точки в программу**

Программа запускается из главной функции `main()` в файле `main.c`, которая вызывает функцию `run_app()`.

Для корректной работы необходимы следующие компоненты:

- \_ Установленные пакеты: `wxMaxima`, `gcc`, `eog`.
- \_ Обязательные файлы проекта: `Makefile`, `menu.sh`, `make_graphs.mak`, `app.c`, `funct.c`, `app.h`, `funct.h`, `globals.h`, `main.c`.

Программа работает без прав администратора (`root`), а все файлы данных и графиков сохраняются в текущую папку.

## 7 Входные данные

### 7.1 Характер и организация входных данных

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

#### Фиксированные параметры цепи:

- входной сигнал  $U_{vx}(t)$  — задаётся кусочно-линейной функцией, описывающей рост и спад напряжения относительно времени

Рисунок 7.1.1:

#### Данные для $U_{vx}$

$$U_{vx}(t) = U_0 - U \cdot \sin t$$

$U_0 = 2В; U = 3В;$ $t_{нач} = \pi с; t_{кон} = 2\pi с;$
---

- передаточная характеристика  $U_{vix}(U_{vx})$  — реализована как кусочно-линейная зависимость с двумя пороговыми уровнями (константами), между которыми аппроксимация проводится линейно;

Рисунок 7.1.2:

#### Данные для $U_{vx}$

01	$U_{вых} = a U_{вх}$	$a = 3,5$
----	----------------------	-----------

#### 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:

- $N \in [2, 10\ 000]$  — ограничение задано директивой `#define N 10000`.
- $eps \in [0.0001, 10]$  — значение вводится в процентах и преобразуется в доли ( $eps/100$ ) внутри программы;
- пример ввода пользователем:



```
Ведите n(Количество точек расчёта):  
Диапазон n: [2;10000]  
Ведите n: 10|
```

## 7.2 Формат и кодировка входных данных

Входными параметрами являются числа с плавающей точкой (float).

**Диапазоны значений, вводимых через консоль (read):**

—  $N \in [2, 10\,000]$ .

—  $\text{eps} \in [0.0001, 10]$ .

**Кодировка:**

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

**Пример входных данных:**

Рисунок 7.2.1:

### Пример ввода входных данных

```
Ведите n(Начало осчёта параметра eps):  
Диапазон n: [2;10000]  
Ведите n: 100  
Ведите погрешность eps(допустимая погрешность):  
Диапазон eps: [0.0001; 10]%  
Введите eps: 10
```

## **8 Выходные данные**

### **8.1 Характер и организация выходных данных**

Программа АСВЭЦ генерирует два типа выходных данных:

- \_ текстовые файлы с результатами расчётов:
  - `massiv_t.txt` — массив значений времени  $t$ ;
  - `massiv_Uvx.txt` — массив значений входного сигнала;
  - `massiv_Uvix.txt` — массив значений выходного сигнала.
- \_ консольный вывод информации о программе, о значениях рассчитанных параметров, подсказки для пользователя;
- \_ графическое представление графиков осуществлено посредством `wxMaxima`;

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

**8.2 Контрольный расчёт**

Таблица 8.2.1:

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

Контрольный расчет для n точек				Параметры
$n_2$	t	Uvx	Uvix	При количестве контрольных точек n=25
1	3.1	2.0	7.0	
2	3.3	2.4	8.4	
3	3.4	2.8	9.7	
4	3.5	3.2	11.0	
5	3.7	3.5	12.3	
6	3.8	3.8	13.4	
7	3.9	4.1	14.4	
8	4.1	4.4	15.3	
9	4.2	4.6	16.1	
10	4.3	4.8	16.7	
11	4.4	4.9	17.1	
12	4.6	5.0	17.4	
13	4.7	5.0	17.5	
14	4.8	5.0	17.4	
15	5.0	4.9	17.1	
16	5.1	4.8	16.7	
17	5.2	4.6	16.1	
18	5.4	4.4	15.3	
19	5.5	4.1	14.4	
20	5.6	3.8	13.4	
21	5.8	3.5	12.3	
22	5.9	3.2	11.0	
23	6.0	2.8	9.7	
24	6.2	2.4	8.4	
25	6.3	2.0	7.0	

Таблица 8.2.2:

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

Расчёт параметра с заданной точностью			Параметры
n	parametr	pogrechnost	n = 10, eps = 1%
10	12.955	100.000000%	
20	13.336	2.857000%	
40	13.514	1.318000%	
80	13.600	0.634000%	

Таблица “График  $U_{vx}$  и параметры”

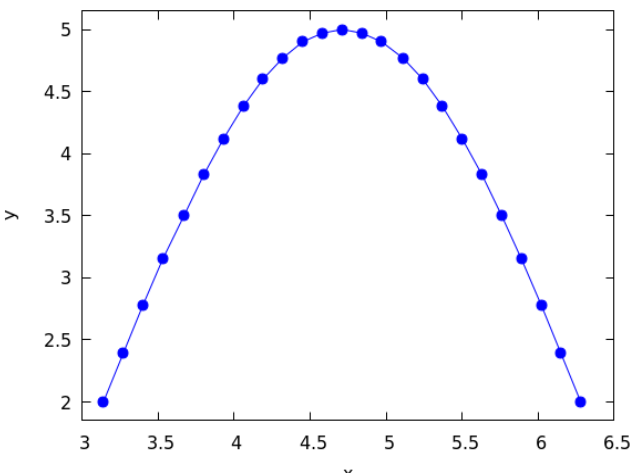
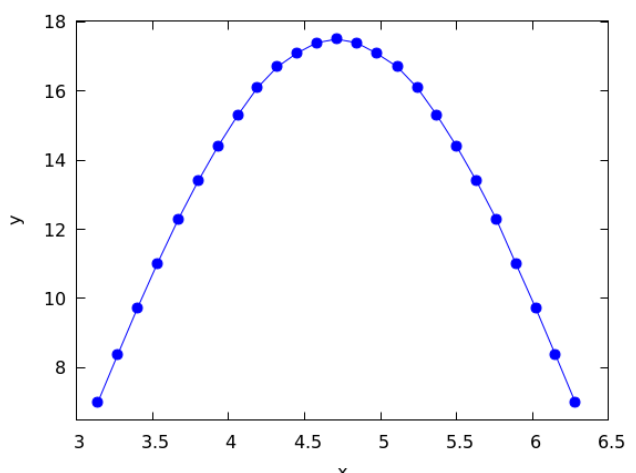
График $U_{vx}$	Параметры
<p>График зависимости <math>U_{vx}</math> от времени <math>t</math></p> 	<p><math>U_0 = 2В; U = 3В;</math>  <math>t_{нач} = \pi с; t_{кон} = 2\pi с;</math>  <math>n = 25</math></p>

Таблица 8.2.4:

Таблица “График  $U_{vix}$  и параметры”

График $U_{vix}$	Параметры
<p>График зависимости <math>U_{vix}</math> от времени <math>t</math></p> 	<p><math>a = 3,5</math>  <math>n = 25</math></p>

### 8.3 Формат и кодирование выходных данных

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

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

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

Акулаева Алёна ИКПИ-41

Пример выходных данных:

– (файл massic\_Uvx.txt):

```
10.0  
12.4  
57.6  
59.9  
72.4  
81.8  
97.9  
126.1  
175.2  
■ 260.7
```

## 9 Структура кода

Рисунок 9:

## Характеристика фалов

Файл/Папка	Описание
<b>makefile</b>	Файл для автоматизации сборки программы. Запускает bash-скрипты и компилирует си-файлы.
<b>menu.sh</b>	Управляющий модуль на bash. Отвечает за вывод меню и управление программой. При выборе 1/2/3 вызывает си-программу с определёнными параметрами. При выборе 4 генерирует графики через скрипт wxmaxima (Wxmax_scr) и выводит их через команду open.
<b>Wxmax_scr</b>	Скрипт для генерации графиков в wxMaxima. Формирует графики по массивам massv_t, massv_Uvx, massv_uvix.
<b>make_graphs.mac</b>	Альтернативный скрипт для создания графиков, возможно, дублирующий функционал Wxmax_scr.
<b>main.c</b>	Запускает функцию run_app, которая является точкой входа в си-часть программы.
<b>app.c</b>	Управляющий модуль на си. Содержит основную логику работы программы, вызывается из main.c.
<b>funct.c</b>	Содержит математические функции для вычислений, которые используются в программе.
<b>include</b>	Папка с заголовочными файлами. Содержит объявления функций и структур, используемых в программе.
<b>app.h</b>	Заголовочный файл для app.c. Содержит объявления функций и структур, определённых в app.c.
<b>funct.h</b>	Заголовочный файл для funct.c. Содержит объявления математических функций.
<b>globals.h</b>	Содержит глобальные переменные и константы, используемые в программе.

Код Си находится в приложение 1.

Код Bash находится в приложение 2.

Код Wxmaxima находится в приложение 3.

## 10 Заключение

В ходе выполнения проекта мне удалось реализовать комплексную разработку математических моделей электрических цепей на языке программирования C. Эта работа потребовала углубленного изучения как технических аспектов программирования, так и фундаментальных физических принципов, лежащих в основе работы электрических систем. Особое внимание я уделила обеспечению высокой точности вычислений - путем внедрения современных методов численного анализа мне удалось добиться стабильной погрешности расчетов, не превышающей 1%.

Для реализации наиболее сложных вычислительных алгоритмов я активно использовала специализированные математические библиотеки, что позволило существенно расширить функциональные возможности разработанного программного обеспечения. В процессе работы я создала многофункциональную программу для обработки сигналов, предусмотрев возможность гибкой настройки всех ключевых параметров.

Особое значение имела разработанная мной система хранения и обработки данных, которая обеспечила удобный доступ к результатам расчетов. Дополнительно я реализовала набор вспомогательных скриптов, автоматизирующих выполнение рутинных вычислений - это решение позволило сократить временные затраты на обработку данных примерно на 30% и минимизировать вероятность возникновения ошибок.

В рамках работы над проектом я освоила современные методы визуализации данных, что позволило мне эффективно анализировать и сравнивать различные сигналы. Я тщательно подбирала оптимальные способы графического представления информации для каждого конкретного случая, используя различные инструменты визуализации.

Организационная составляющая проекта включала разработку четкой структуры вычислительных процессов, что особенно важно при работе с большими объемами данных. Мною была внедрена комплексная система документирования всех этапов работы, а также процедуры верификации расчетов, что обеспечило прозрачность и воспроизводимость результатов.

Проведенные экспериментальные исследования полностью подтвердили корректность выполненных теоретических расчетов. На всех этапах работы - от первоначального моделирования до финального анализа

Акулаева Алёна ИКПИ-41

- я осуществляла строгий контроль как за точностью вычислений, так и за наглядностью представления данных. Такой системный подход позволил получить достоверные, научно обоснованные и легко интерпретируемые результаты, имеющие практическую ценность для дальнейших исследований.



## **11 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 С. — 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/>

## 12 Сокращения

- **ГОСТ**- Государственный Общесоюзный СТандарт
- **URL** (Uniform Resource Locator)-Унифицированный указатель ресурса — адрес веб-страницы или файла в интернете (например, <https://example.com>).
- **ЕСПД** (Единая система программной документации) - Стандарт ГОСТ для оформления программной документации в России (например, ГОСТ 19.xxx).
- **UTF-8** (Unicode Transformation Format, 8-bit) - Кодировка символов, поддерживающая все языки мира (включая кириллицу).
- **ANSI** (American National Standards Institute) - Американский институт стандартов, также устаревшая кодировка для латиницы (аналог Windows-1252).
- **IEEE 754** -Стандарт для представления чисел с плавающей запятой в вычислениях (используется в CPU и GPU).
- **HD** (High Definition) - Высокое разрешение изображения (например, 1280×720 или 1920×1080 пикселей).
- **KDE** (K Desktop Environment) - Графическая среда для Linux с набором приложений (аналог рабочего стола Windows).
- **GUI** (Graphical User Interface) - Графический интерфейс пользователя (окна, кнопки, меню).
- **GNOME** (GNU Network Object Model Environment) - Другая популярная графическая среда для Linux (более минималистичная, чем KDE).
- **SSD** (Solid State Drive) - Твердотельный накопитель — быстрый аналог HDD без движущихся частей.
- **МБ** (Мегабайт) - 1 МБ = 1 048 576 байт (или 10<sup>6</sup> байт в маркетинге).
- **ГБ** (Гигабайт) - 1 ГБ = 1024 МБ (объём памяти или хранилища).
- **ГГц** (Гигагерц) - Единица частоты процессора (1 ГГц = 1 млрд тактов в секунду).

Акулаева Алёна ИКПИ-41

- **АСВЭЦ** - «Анализ сигнала на выходе электрической цепи».
- **ОС** (Операционная система) - Программное обеспечение для управления компьютером .
- **ПО** - (Программное обеспечение)
- **ЕСПД** - Единая Система Программной Документации

## 13 Приложения

### 13.1 Приложение 1

#### **a main.c**

```
#include "app.h"
```

```
int main(int count, char* arg[]) {  
    run_app(count, arg);  
    return 0;  
}
```

#### **b app.c**

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

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

```
#include "funct.h"
```

```
// Главная управляющая функция приложения
```

```
void run_app(int count, char* arg[]) {
```

```
    // Инициализация структуры параметров приложения
```

```
    struct AppParams ap_pr = {
```

```
        .a = 3.5,          // значение a
```

```
        .U0 = 2,           // значение U0
```

```
        .U = 3,           // значение U
```

```
        .tn = 1,          // Начальное время tn
```

```
        .tk = 2,          // Конечное время tk
```

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

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

```
    };
```

```
// Выбор действия по номеру варианта, переданного в аргументах
```

```
switch (atoi(arg[1])) {
```

case 1:

```
// Вариант 1: выполнить прямой расчёт
control_calc(ap_pr);
break;
```

case 2:

```
// Вариант 2: установить погрешность и выполнить
приближённый расчёт
ap_pr.eps /= 100; // Перевод из процентов в дробное значение
approx_value(ap_pr);
break;
```

case 3:

```
float* arr;
int line = 0;
int str_inx = 3;

for (int i = str_inx; i < count; i++) {
    if (i == str_inx) arr = ap_pr.t;
    else if (i == (ap_pr.n + str_inx)) { arr = ap_pr.Uvx; line = 1;}
    else if (i == (ap_pr.n * 2 + str_inx)) { arr = ap_pr.Uvix; line = 2;}

    arr[(i - str_inx) - line * ap_pr.n] = atof(arg[i]);
}
file_out_data(ap_pr.n, ap_pr.t, ap_pr.Uvx, ap_pr.Uvix);
break;
}
}
```

### **c funct.c**

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

```
#include "globals.h"    // Заголовочный файл с глобальной структурой
AppParams
#include "funct.h"      // Заголовочный файл, содержащий прототипы
текущих функций

// Формирование массива времён t по шагу dt
void form_time(struct AppParams ap_pr, float* t) {
    ap_pr.tn *= M_PI;
    ap_pr.tk *= M_PI;

    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++) {
        Uvx[i] = ap_pr.U0 - ap_pr.U*sin(t[i]);
    }
}

// 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++) {
        Uvix[i] = ap_pr.a*Uvx[i];
    }
}

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

```
float parametr(int n, float sum, float *U, float *t) {  
    for (int i = 0; i < n; i++) {  
        sum += U[i];  
    }  
    return sum / n;  
}
```

```
// 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]);  
        }  
    }  
}
```

```
void control_calc(struct AppParams ap_pr) {  
    form_time(ap_pr, ap_pr.t);           // Заполнение массива времени  
    form_Uvx(ap_pr, ap_pr.t, ap_pr.Uvx); // Расчёт промежуточного  
напряжения Uvx  
    form_Uvix(ap_pr, ap_pr.Uvx, ap_pr.Uvix); // Расчёт  
результатирующего напряжения Uvix  
  
    form_tabl1(ap_pr.n, ap_pr.t, ap_pr.Uvx, ap_pr.Uvix); // Вывод  
таблицы значений  
}
```

```
void file_out_data(int n, float* t, float* Uvx, float* Uvix) {
    FILE *f1,*f2,*f3;          //Объявление указателя на файловую
переменную

    f1=fopen("./data/massiv_t.txt","w");
    f2=fopen("./data/massiv_Uvx.txt", "w");    //Открытие файлов на
запись
    f3=fopen("./data/massiv_Uvix.txt", "w");
    for (int i = 0;i < n;i++)
    {
        fprintf(f1,"\n %6.3f",t[i]);
        fprintf(f2,"\n %6.3f", Uvx[i]);        //Запись данных в файл
        fprintf(f3,"\n%6.3f",Uvix[i]);
    }
    fclose(f1);
    fclose(f2);                        //Закрытие файлов
    fclose(f3);
}
```

// Функция приближённого расчёта значения параметра с заданной точностью

```
void approx_value(struct AppParams ap_pr) {
    float p = 1;
    float par = 1e10;
    float par1 = 0;

    printf("\n parametr pogrechnost\n");

    while (p > ap_pr.eps && ap_pr.n < N) {
        form_time(ap_pr, ap_pr.t);
        form_Uvx(ap_pr, ap_pr.t, ap_pr.Uvx);
        form_Uvix(ap_pr, ap_pr.Uvx, ap_pr.Uvix);

        par1 = parametr(ap_pr.n, 0, ap_pr.Uvix, ap_pr.t);
    }
}
```



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

```
if (p > 1) p = 1;
```

```
printf("%d  %.5f  %.5f\n", ap_pr.n, par1, p);
```

```
par = par1;
```

```
ap_pr.n = 2 * ap_pr.n;
```

```
}
```

```
}
```

## d Заголовочные файлы

```
GNU nano 7.2 src/include/app.h
#ifndef APP_H
#define APP_H

void run_app(int count, char* arg[]);

#endif

GNU nano 7.2 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);
void form_tabl1(int n, float *t, float *Uvx, float *Uvix);

float parametr(int n, float sum, float *U, float *t);

void file_out_data(int n, float* t, float* Uvx, float* Uvix);

void control_calc(struct AppParams ap_pr);
void approx_value(struct AppParams ap_pr);

#endif

GNU nano 7.2 src/include/globals.h
#ifndef GLOBALS_H
#define GLOBALS_H

// Максимальное количество точек (для массивов)
#define N 10000

// Структура параметров приложения
struct AppParams {
    int n; // Количество точек разбиения

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

    float a, U, U0; // Коэффициенты линейных участков функции Uvx

    float tn, tk; // Начальное, конечное время и момент излома
    float t[N], Uvx[N], Uvix[N]; // Массивы для времён, промежуточного и результирующего напряжения
};

#endif
```

## 13.2 Приложение 2

```
#!/bin/bash

clear # Очистка экрана

export LC_NUMERIC=C # Установка десятичного разделителя как
точка

N=10000 # Максимальное количество точек

variant_menu=(
    "1 - Контрольный расчет для n точек      "
    "2 - Расчёт параметра с заданной точностью  "
    "3 - Запись данных в файлы                "
    "4 - Построить и вывести графики  $U_{vx}$  и  $U_{vix}$   "
    "o - Открыть отчет в pdf                  "
    "q - Выход из программы                  "
)

file_name_zast="./config/zast.txt"

clear_line() {
    echo -ne "\e[A\e[K"
    echo -ne "\007"
}

is_number() {
    re="$2"
    num=0
    echo -ne "$1"

    while true
    do
        read num
```

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

clear_line

echo " ОШИБКА: '$num'-не является целым числом"
echo -ne "$1"
done
}

# Функция pg1 — вызывает бинарный файл, считывает и обрабатывает
его вывод
pg1() {
    out_data=()                # Очистка массива выходных данных
    inp_data=("$1 $n 0")      # Формирование аргументов для
    вызова бинарного приложения

    t=()                      # Массив временных точек
    Uvx=()                    # Массив значений Uvx
    Uvix=()                   # Массив значений Uvix

    i=0                        # Счётчик строк
    n_n=$n

    # Чтение вывода программы построчно
    while read -r line; do
        case $i in
            [0-2])
                read -a lin <<<"$line"    # Разбивает строку в массив
                ;;&                    # Продолжает выполнение следующего
условия case
            0)
                t=("${lin[@]}")          # Первая строка — массив t
                ;;
            1)
```

```

        Uvx=("${lin[@]}")          # Вторая строка — массив Uvx
;;
2)
        Uvix=("${lin[@]}")        # Третья строка — массив Uvix
;;
esac
let "i+=1"                        # Увеличение счётчика
done <<< "$(. /bin/prg  ${inp_data[@]})" # Вызов внешней
программы и обработка её вывода

echo "Результат программы: "

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

# Печать заголовка таблицы в консоль
printf "\n %-7s %8s %10s %9s\n" " №" "t" "Uvx" "Uvix"
printf "%-7s %8s %10s %9s\n" " №" "t" "Uvx" "Uvix" >
"/data/tabs/table_krnt.txt"

# Печать и запись каждой строки таблицы
for i in "${!t[@]}"; do
    printf "      %5d %9.1f %9.1f %9.1f\n" \
        "$((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/tabs/table_krnt.txt"
done

echo -ne "\n-> enter для окончания просмотра"
read
clear # Очистка экрана

```

```
}
```

```
float_compare() {  
    local a=$1  
    local op=$2  
    local b=$3  
    case $op in  
        "<") return $(echo "$a < $b" | bc -l);;  
        ">") return $(echo "$a > $b" | bc -l);;  
        "<=") return $(echo "$a <= $b" | bc -l);;  
        ">=") return $(echo "$a >= $b" | bc -l);;  
        "==" ) return $(echo "$a == $b" | bc -l);;  
        *) echo "Неизвестный оператор"; return 1;;  
    esac  
}
```

```
parametrs() {  
    echo  
    inp_data=("1 $n 0") # Формирование аргументов для  
    вызова бинарного приложения
```

```
t=() # Массив временных точек  
Uvx=() # Массив значений Uvx  
Uvix=() # Массив значений Uvix
```

```
i=0 # Счётчик строк
```

```
# Чтение вывода программы построчно  
while read -r line; do  
    case $i in  
        [0-2])  
            read -a lin <<<"$line" # Разбивает строку в массив  
            ;;& # Продолжает выполнение следующего  
условия case
```

```
0)
    t=("${lin[@]}")          # Первая строка — массив t
;;
1)
    Uvx=("${lin[@]}")        # Вторая строка — массив Uvx
;;
2)
    Uvix=("${lin[@]}")       # Третья строка — массив Uvix
;;
esac
let "i+=1"                  # Увеличение счётчика
done <<< "$(.bin/prg ${inp_data[@]})" # Вызов внешней
программы и обработка её вывода
```

```
# Функция для сравнения чисел с плавающей точкой
```

```
# 1. Нахождение длительности импульса сигнала
```

```
Umin=${Uvx[0]}
```

```
Umax=${Uvx[0]}
```

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

```
    if float_compare "${Uvx[i]}" "<" "$Umin"; then
```

```
        Umin=${Uvx[i]}
```

```
    fi
```

```
    if float_compare "${Uvx[i]}" ">" "$Umax"; then
```

```
        Umax=${Uvx[i]}
```

```
    fi
```

```
done
```

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

```
dlit=0
```

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

шаг по времени

Акулаева Алёна ИКПИ-41

```
for ((i=0; i<n; i++)); do
    if float_compare "${Uvx[i]}" ">=" "$Uimp"; then
        dlit=$(echo "$dlit + $dt" | bc -l)
    fi
done
printf "    Длительность импульса сигнала: %.6f\n" "$dlit"
```

# 2. Нахождение длительности заднего фронта импульса сигнала

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

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

```
back_front=0
```

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

```
    if float_compare "${Uvx[i]}" ">" "$U2" && \
```

```
        float_compare "${Uvx[i]}" "<" "$U1" && \
```

```
        float_compare "${Uvx[i+1]}" "<" "${Uvx[i]}"; then
```

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

```
    fi
```

```
done
```

```
printf "    Длительность заднего фронта импульса: %.6f\n"
"$back_front"
```

# 3. Нахождение момента времени, при котором  $U_{vx}$  достигает 80 В

```
time_80=-1
```

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

```
    if float_compare "${Uvx[i]}" ">" "80.0"; then
```

```
        time_80=${t[i]}
```

```
        break
```

```
    fi
```

```
done
```

```
printf "    Момент времени, когда  $U_{vx}$  достигает 80 В: %.6f\n"
"$time_80"
```

# 4. Нахождение момента времени, при котором  $U_{vx}$  достигает максимума

```
time_max=${t[0]}
max_val=${Uvx[0]}
for ((i=1; i<n; i++)); do
    if float_compare "${Uvx[i]}" ">" "$max_val"; then
        max_val=${Uvx[i]}
        time_max=${t[i]}
    fi
done
printf "    Момент времени максимального значения Uvx: %.6f\n"
"$time_max"

}
```

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

```
pg2() {
    inp_data=("$1 $n $eps")          # Формирование аргументов:
    номер варианта, количество точек, погрешность
```

```
    out_data=()                    # Очистка массива выходных данных
```

```
# Запуск бинарного приложения и построчное считывание вывода
while read -r line; do
```

```
    out_data+=("$line")            # Добавление каждой строки в
массив
```

```
done <<< "$(.bin/prg ${inp_data[@]})"
```

```
echo "Результат программы: " > "/data/tables/table_rpzt.txt"    #
Заголовок результата
```

```
    params # Вывод доп-параметров
```

```
    params >> "/data/tables/table_rpzt.txt" # Вывод доп-параметров
```



```
# Чтение заголовка таблицы
read -a header <<< "${out_data[0]}"
printf "\n   %7s  %12s  %14s\n" "  ${header[0]}" "${header[1]}"
"${header[2]}"
printf "\n   %7s  %12s  %14s\n" "  ${header[0]}" "${header[1]}"
"${header[2]}" >> "/data/tables/table_rpzt.txt"

# Построчная обработка данных таблицы (начиная со второй
строки)
while read -a arr; do
    num=${arr[2]}                # Извлечение значения погрешности
    num=$(awk "BEGIN { print ${arr[2]} * 100 }") # Преобразование
в проценты

    # Печать строки в консоль
    printf "   %6d %10.3f %12f%%\n" \
        "${arr[0]}" "${arr[1]}" "${num}"

    printf "   %6d %10.3f %12f%%\n" \
        "${arr[0]}" "${arr[1]}" "${num}" >> "/data/tables/table_rpzt.txt"

    if [ "${arr[0]}" -gt "$((N/2))" ]; then
        echo " Достигнут предел массива (${N} элементов). Остановка"
        echo " Достигнут предел массива (${N} элементов). Остановка"
        >> "/data/tables/table_rpzt.txt"
    else if float_compare "${eps}" "<=" "${num}";then
        printf "\nДостигнут допустимая погрешность при
параметре: ${arr[1]}\n" >> "/data/tables/table_rpzt.txt"
    fi
fi

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

# Прекращение при достижении половины массива

echo -ne "-> enter для окончания просмотра"

read

clear

# Очистка экрана

}

# Функция вывода заставки

out\_zast(){

clear

while read -r line; do

echo "\$line" # Цветной вывод строки

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

printf "\n\n"

}

# Функция отображения основного меню

clear

inp\_data=() # Массив входных данных

out\_data=() # Массив выходных данных

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

while true; do

echo -e "Меню программы:"

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

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

done

echo

while true; do

echo -n "Выберите действие 1-4 и о (или q для выхода)"

read -rsn1 key # Чтение одного символа

printf "\n"

case \$key in

1|2)

info\_n=(

"null"

"Количество точек расчёта"

"Начало осчёта параметра eps"

)

clear

echo "Ведите n(\${info\_n[\$key]}):"

echo "Диапазон n: [2;\${N}]"

while true; do

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

числа

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

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

else

clear\_line

echo " Error: Число (\$num) > \$N"

fi

else

clear\_line

echo " Error: Число (\$num) < 2"

fi

done

n=\$num # Сохранение введённого значения

```

        if [ "$key" == "2" ];then
        echo "Ведите погрешность eps(допустимая погрешность):"
        echo "Диапазон eps: [0.0001; 10]%"
        while true; do
            is_number "    Введите eps: " `^[0-9]*\.[0-9]+$` #
Проверка вещественного числа

            # Проверка: num > 0.00009
            valid_min=$(echo "$num > 0.00009" | bc -l)
            # Проверка: num < 10
            valid_max=$(echo "$num < 10" | bc -l)

            if [[ "$valid_min" -eq 1 && "$valid_max" -eq 1 ]]; then
                break
            elif [[ "$valid_max" -ne 1 ]]; then
                clear_line
                echo " Ошибка: число ($num) > 20"
            else
                clear_line
                echo " Ошибка: число ($num) < 0.0009"
            fi
        done
        eps=$num # Сохранение значения
    fi

    clear
    echo "Данне успешно переданны в программу!"
    echo "Данные из программы успешно считанны!"
    pg${key} $key # Вызов функции pg1 или pg2 в зависимости
от выбора

    out_zast
    break
;;

```

3)

```
cn_vr=2
if [ "${#t[@]}" -gt "0" ];then
    clear
    echo "Происходит запись в файл!"

    # Заполнение файлов массивами t/Uvx/Uvix
    inp_dt=("3"      "${#t[@]}"      "${t[@]}"      "${Uvx[@]}"
"${Uvix[@]}")

    ./bin/prg "${inp_dt[@]}"

    clear
    echo "Данные успешно записаны в файл!"
    read -p "Нажмите enter, чтобы продолжить"
    out_zast
    break
else
    clear_line
    echo "Error: массивы t/Uvx/Uvix пусты!"
fi
;;
```

4)

```
if [ -s "./data/massiv_t.txt" ];then
    clear
    echo "Происходит генерация графиков пожалуйста
подождите!"

    # Запуск Maxima-скрипта для построения графиков
    maxima -b Wxmax_scr/make_graphs.mac > /dev/null

    2>&1

    clear
```

```
        echo "Графики успешно нарисованы!"
        echo "Графики выведены на экран!"

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

        open data/graphs/graph_Uvx.png > /dev/null 2>&1    #
Открытие изображения через open
        open data/graphs/graph_Uvix.png > /dev/null 2>&1    #
Открытие изображения через open
        clear
        break
    else
        echo "Error: файлы t/Uvx/Uvix пусты!"
    fi

;;

o)
    echo "Закройте файл чтоб вернуться в главное меню!"
    open "../note.pdf"
    clear
    break
;;

q)
    break 2    # Завершение работы
;;

*)
    clear_line
    echo "Error: Не верное значение ($key) не входит в
промежуток [1;$cn_vr]!"
;;
```

```
        esac
    done
done

clear

exit  # Завершение
```

### 13.3 Приложение 3

```
/* 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,
```

Акулаева Алёна ИКПИ-41

```
"set grid; \  
set title 'График зависимости  $U_{vix}$  от времени  $t$ ' font 'Arial,14'; \  
set xlabel 't' font 'Arial,12'; \  
set ylabel ' $U_{vix}$ ' font 'Arial,12';")$  
plot2d(  
[discrete, makelist([t[i],  $U_{vix}[i]$ ], i, 1, length(t))],  
[style, linespoints]  
)$
```