

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 – 19 вариант

Входной сигнал – 19 вариант

Выполнил студент группы ИКПИ-41:

Потапов Е.С. _____

Дата выполнения: «29» Май

Проверил:

Хазиев Н.Н. _____

Санкт-Петербург

2025

Оглавление

Аннотация	4
Задание к курсовой работе	5
1 Общие сведения	6
1.1 Обозначение и наименование программы	6
1.2 Языки разработки:	6
2 Функциональное назначение	6
2.1 Решаемые задачи	6
2.2 Назначение:	7
2.3 Ограничения в функциональности:	7
3 Таблица идентификаторов	9
4 Описание логической структуры	10
4.1 Алгоритм программы	10
4.2 Описание функций	11
4.3 Используемые методы	12
4.4 Связи программы с другими программами	12
5 Используемые технические средства	14
6 Вызов и загрузка	15
6.1 Способ вызова программы с соответствующего носителя данных	15
6.2 Входные точки в программу	15
7 Входные данные	16
7.1 Характер и организация входных данных	16
7.2 Формат и кодировка входных данных	17
8 Выходные данные	18
8.1 Характер и организация выходных данных	18
8.2 Контрольный расчёт	19
8.3 Формат и кодирование выходных данных	21

9	Структура кода	22
10	Заключение	23
11	Список используемой литературы	25
12	Сокращения	26
13	Приложения	28
13.1	Приложение 1	28
a	main.c	28
b	app.c	28
c	funct.c	30
d	Заголовочные файлы	33
13.2	Приложение 2	34
13.3	Приложение 3	50

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.

$U_{\text{вх}}(t) = \begin{cases} a(1 - e^{-b(t-t_{\text{нач}})}) & \text{при } t \leq t_1 \\ a(1 - e^{-b(t_1-t_{\text{нач}})}) - d(t-t_1) & \text{при } t_1 < t \leq t_2 \\ (a(1 - e^{-b(t_1-t_{\text{нач}})}) - d(t_2-t_1)) \cdot e^{-c(t-t_2)} & \text{при } t > t_2 \end{cases}$	$a = 50\text{В/с}; b = 0,07(1/\text{с});$ $c = 0,1(1/\text{с});$ $t_{\text{нач}} = 15\text{с}; t_1 = 30\text{с};$ $t_2 = 50\text{с};$ $t_{\text{кон}} = 100\text{с};$
$U_{\text{вых}} = \begin{cases} a & \text{при } U_{\text{вх}} \leq U_{\text{вх1}} \\ bU_{\text{вх}}^2 & \text{при } U_{\text{вх}} > U_{\text{вх1}} \end{cases}$	$a = 5\text{В};$ $b = 0,05(1/\text{В});$ $U_{\text{вх1}} = 10\text{В}$

— 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:

а Совместимость с ОС

Программа работает **только в Ubuntu Linux** и не поддерживает Windows или другие операционные системы.

б Ограничения входных данных

- _ временной диапазон жестко задан: $t \in [15, 100]$ (можно изменить вручную в коде);
- _ параметры цепи рассчитаны для **конкретного варианта** (требуется правка кода для других значений);
- _ максимальное количество точек ограничено **10 000** (из-за размера массива).

с Численные ограничения

- _ используется тип **float**, что снижает точность вычислений;
- _ итерационные методы могут не сходиться при слишком малой погрешности **eps**.

д Системные требования

- _ требуется **Ubuntu Linux** (другие дистрибутивы не тестировались).
- _ обязательные пакеты:
 - **gcc** — для компиляции программы;
 - **wxMaxima** — для построения графиков.

е Ограничения интерфейса:

- _ программа работает **только через консоль** (графического интерфейса нет);
- _ низкая устойчивость к некорректному вводу (например, текст вместо чисел);
- _ кодировки и локализация;
- _ программа использует **UTF-8** для вывода текста и специальных символов (например, греческих букв).

f Для корректного отображения убедитесь, что:

- _ в системе установлена локаль **UTF-8** (например, LANG=en_US.UTF-8);
- _ терминал поддерживает Unicode-символы.

g Настройка параметров:

- _ для изменения:
 - временного диапазона (tn, tk);
 - параметров цепи (сопротивлений, индуктивностей и др.);
 - максимального числа точек (Nmax);необходимо вручную править код в файле globals.h.

3 Таблица идентификаторов

Таблица 3.1:

Таблица идентификаторов

Переменная	Тип	Описание
n	int	Количество точек разбиения
eps	float	Допустимая погрешность (для приближённого метода)
a, b, c, d	float	Коэффициенты линейных участков функции U_{vx}
tn	float	Начальное время
t1	float	Момент времени t_1
t2	float	Момент времени t_2
tk	float	Конечное время
a_vix, b_vix, U_{vx1}	float	Пороговые значения для функции U_{vix}
t[N]	float	Массив временных точек
$U_{vx}[N]$	float	Массив значений входной функции
$U_{vix}[N]$	float	Массив значений выходной функции

Потапов Егор Сергеевич ИКПИ-41

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
paramtrs()	Выводит длительности импульса сигнала, длительности заднего фронта импульса сигнала, Длительность заднего фронта импульса, Момент времени, когда U_{vx} достигает 80 В
form_tabl1()	Выводит таблицу значений t , U_{vx} , U_{vix} в три строки (для <code>control_calc</code>)
control_calc()	Выполняет точный расчёт, заполняет массивы t , U_{vx} , U_{vix} и выводит таблицу
approx_value()	Выполняет итеративный расчёт с удвоением n до достижения заданной погрешности <code>eps</code>
file_out_data()	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>)
form_time()	Заполняет массив t временными точками с равномерным шагом от t_n до t_k
form_Uvx()	Вычисляет массив U_{vx} по кусочно-линейному закону с изломами в точках t_1 , t_2 , t_3 , t_4
form_Uvix()	Вычисляет массив U_{vix} как линейное преобразование U_{vx} ($U_{vix}[i] = 2.5 * U_{vx}[i] + 10$)
parametr()	Вычисляет среднее значение массива U (используется в <code>approx_value</code>)
clear_line()	Очищает текущую строку в терминале и воспроизводит звуковой сигнал
is_number()	Проверяет, является ли введённое значение числом, соответствующим регулярному выражению
ts1()	Выполняет контрольный расчёт для n точек, выводит таблицу значений t , U_{vx} , U_{vix} и сохраняет их в файл
ts2()	Выполняет расчёт параметра с заданной погрешностью <code>eps</code> , выводит таблицу с погрешностью и сохраняет её в файл
out_zast()	Выводит заставку (ASCII-арт или логотип) из файла <code>./config/zast.txt</code>
Основной цикл while	Управляет меню программы, обрабатывает выбор пользователя и вызывает соответствующие функции
run_app()	Главная функция приложения, инициализирует параметры и вызывает нужный метод (<code>control_calc</code> , <code>approx_value</code> , <code>file_out_data</code>)

4.3 Используемые методы

Программа АСВЭЦ использует следующие методы:

– **численные расчёты:**

- разбивает интервал $t \in [15, 100]$ 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. 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

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	-

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}

$U_{vx}(t) = \begin{cases} a(1 - e^{-b(t-t_{нач})}) & \text{при } t \leq t_1 \\ a(1 - e^{-b(t_1-t_{нач})}) - d(t-t_1) & \text{при } t_1 < t \leq t_2 \\ (a(1 - e^{-b(t_1-t_{нач})}) - d(t_2-t_1)) \cdot e^{-c(t-t_2)} & \text{при } t > t_2 \end{cases}$	$a = 50\text{В/с}; b = 0,07(1/\text{с});$ $c = 0,1(1/\text{с});$ $t_{нач} = 15\text{с}; t_1 = 30\text{с};$ $t_2 = 50\text{с};$ $t_{кон} = 100\text{с};$
--	---

- 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_{vix}

$U_{vix} = \begin{cases} a & \text{при } U_{vx} \leq U_{vx1} \\ bU_{vx}^2 & \text{при } U_{vx} > U_{vx1} \end{cases}$	$a = 5\text{В};$ $b = 0,05(1/\text{В});$ $U_{vx1} = 10\text{В}$
---	---

Пользовательские параметры (вводятся через консоль):

- количество точек N — задаёт разрешение графика (число временных отсчётов);
- точность расчёта ϵ_{ps} — 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`.
- $\text{eps} \in [0.001, 10]$ — значение вводится в процентах и преобразуется в доли ($\text{eps}/100$) 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_0	t	Uvx	Uvix	При количестве контрольных точек n=25
1	15.0	-92.9	5.0	
2	18.5	-61.5	5.0	
3	22.1	-37.0	5.0	
4	25.6	-17.9	5.0	
5	29.2	-3.0	5.0	
6	32.7	29.3	42.8	
7	36.2	25.0	31.3	
8	39.8	20.8	21.5	
9	43.3	16.5	13.6	
10	46.9	12.3	7.5	
11	50.4	9.5	5.0	
12	54.0	16.3	13.4	
13	57.5	21.2	22.4	
14	61.0	24.5	30.1	
15	64.6	26.9	36.2	
16	68.1	28.6	40.9	
17	71.7	29.8	44.3	
18	75.2	30.6	46.7	
19	78.8	31.1	48.5	
20	82.3	31.6	49.8	
21	85.8	31.8	50.7	
22	89.4	32.0	51.3	
23	92.9	32.2	51.8	
24	96.5	32.3	52.1	
25	100.0	32.3	52.3	

Таблица 8.2.2:

Таблица “Расчёт параметра с заданной точностью”

Расчёт параметра с заданной точностью	Параметры															
Длительность импульса сигнала: 9.4 Длительность заднего фронта импульса: 18.8 Момент времени, когда Uvx достигает 80 В: Не достигает Момент времени максимального значения Uvx: 33.9	n = 10, eps = 1%															
<table><tr><td>n</td><td>parametr</td><td>pogrechnost</td></tr><tr><td>10</td><td>30.241</td><td>100.000000%</td></tr><tr><td>20</td><td>29.588</td><td>2.208000%</td></tr><tr><td>40</td><td>30.495</td><td>2.974000%</td></tr><tr><td>80</td><td>30.308</td><td>0.616000%</td></tr></table>	n	parametr	pogrechnost	10	30.241	100.000000%	20	29.588	2.208000%	40	30.495	2.974000%	80	30.308	0.616000%	
n	parametr	pogrechnost														
10	30.241	100.000000%														
20	29.588	2.208000%														
40	30.495	2.974000%														
80	30.308	0.616000%														

Таблица 8.2.3:

Таблица “График U_{vx} и параметры”

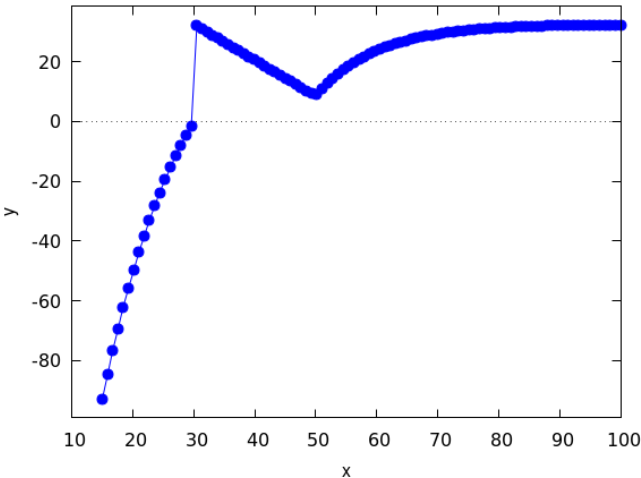
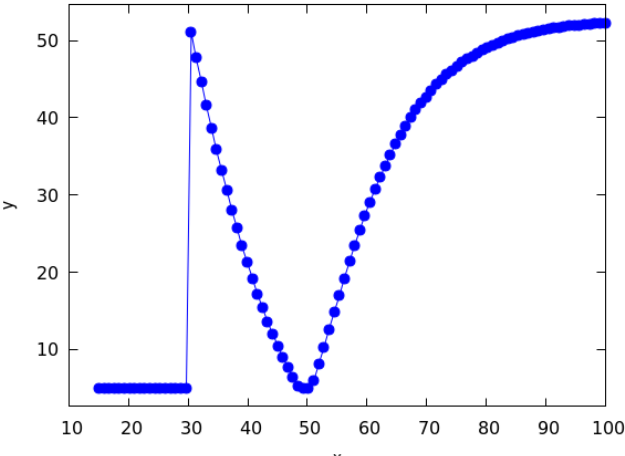
График U_{vx}	Параметры
<p>График зависимости U_{vx} от времени t</p> 	<div> $a = 50 \text{ В/с}; b = 0,07(1/\text{с});$ $c = 0,1(1/\text{с});$ $t_{\text{нач}} = 15 \text{ с}; t_1 = 30 \text{ с};$ $t_2 = 50 \text{ с};$ $t_{\text{кон}} = 100 \text{ с};$ </div> <p>$n = 25$</p>

Таблица 8.2.4:

Таблица “График U_{vix} и параметры”

График U_{vix}	Параметры
<p>График зависимости U_{vix} от времени t</p> 	<div> $a = 5 \text{ В};$ $b = 0,05(1/\text{В});$ $U_{\text{вх1}} = 10 \text{ В}$ </div> <p>$n = 25$</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 знаком после запятой.

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

Пример выходных данных:

t	Uvx	Uvix
15.0	-92.9	5.0
36.2	25.0	31.3
57.5	21.2	22.4
78.8	31.1	48.5
100.0	32.3	52.3

9 Структура кода

Рисунок 9:

Характеристика фалов

Файл/Папка	Описание
makefile	Файл для автоматизации сборки программы. Запускает bash-скрипты и компилирует си-файлы.
functions.sh	Содержит функции для вывода результатов программы си, а также функции для управление меню стрелочками
menu.sh	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.
Wxmax_scr	Скрипт для генерации графиков в wxMaxima. Формирует графики по массивам massv_t, massv_Uvx, massv_uvix.
make_graphs.mac	Альтернативный скрипт для создания графиков, возможно, дублирующий функционал Wxmax_scr.
main.c	Запускает функцию run_app, которая является точкой входа в си-часть программы.
app.c	Управляющий модуль на си. Содержит основную логику работы программы, вызывается из main.c.
funct.c	Содержит математические функции для вычислений, которые используются в программе.
include	Папка с заголовочными файлами. Содержит объявления функций и структур, используемых в программе.
app.h	Заголовочный файл для app.c. Содержит объявления функций и структур, определённых в app.c.
funct.h	Заголовочный файл для funct.c. Содержит объявления математических функций.
globals.h	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 С. Эта работа потребовала углубленного изучения как технических аспектов программирования, так и фундаментальных физических принципов, лежащих в основе работы электрических систем.

Особое внимание я уделил обеспечению высокой точности вычислений. Благодаря внедрению современных методов численного анализа мне удалось добиться стабильной погрешности расчетов, не превышающей 1%. Для реализации наиболее сложных вычислительных алгоритмов я активно использовал специализированные математические библиотеки, что позволило существенно расширить функциональные возможности разработанного программного обеспечения.

Я разработал многофункциональную программу для обработки сигналов, предусмотрев возможность гибкой настройки всех ключевых параметров. Особое значение имела созданная мной система хранения и обработки данных, которая обеспечила удобный доступ к результатам расчетов.

Дополнительно я реализовал 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 проекта я освоил современные методы визуализации данных, что позволило эффективно анализировать и сравнивать различные сигналы. Я тщательно подбирал оптимальные способы графического представления информации для каждого конкретного случая, используя различные инструменты визуализации.

Организационная составляющая проекта включала разработку четкой структуры вычислительных процессов. Я внедрил комплексную систему 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

Потапов Егор Сергеевич ИКПИ-41

- я осуществлял 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 10⁶ байт в маркетинге).
- **ГБ** (Гигабайт) - 1 ГБ = 1024 МБ (объем памяти или хранилища).
- **ГГц** (Гигагерц) - Единица частоты процессора (1 ГГц = 1 млрд тактов в секунду).

Потапов Егор Сергеевич 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 = 50,          // линейный коэффициент  $U_{vx}$ 
```

```
        .b = 0.07,        // линейный коэффициент  $U_{vx}$ 
```

```
        .c = 0.1,         // линейный коэффициент  $U_{vx}$ 
```

```
        .d = 1.2,         // линейный коэффициент  $U_{vx}$ 
```

```
        .tn = 15,         // Начальное время  $t_n$ 
```

```
        .t1 = 30,         // Момент времени  $t_1$ 
```

```
        .t2 = 50,         // Момент времени  $t_2$ 
```

```
        .tk = 100,        // Конечное время  $t_k$ 
```

```
        .a_vix = 5,        // линейный коэффициент  $U_{vix}$ 
```

```
        .b_vix = 0.05,     // линейный коэффициент  $U_{vix}$ 
```

```
        .Uvx1 = 10,        // линейный коэффициент  $U_{vix}$ 
```

```
.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) {  
    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) Uvx[i] = ap_pr.a*(1-exp(-ap_pr.b*(t[i]-  
ap_pr.t1)));  
        else if (t[i] <= ap_pr.t2) Uvx[i] = ap_pr.a*(1-exp(-  
ap_pr.b*(ap_pr.t1-ap_pr.tn)))-ap_pr.d*(t[i]-ap_pr.t1);  
        else Uvx[i] = ap_pr.a*(1-exp(-ap_pr.b*(ap_pr.t1-ap_pr.tn)))-  
ap_pr.d*(ap_pr.t2-ap_pr.t1)*exp(-ap_pr.c*(t[i]-ap_pr.t2));  
    }  
}  
  
// 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.a_vix;
        else Uvix[i] = ap_pr.b_vix*pow(Uvx[i], 2);
    }
}
```

// 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);      //Заккрытие 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, b, c, d; // Коэффициенты линейных участков функции Uvx

    float a_vix, b_vix, Uvx1; // Коэффициенты линейных участков функции Uvix
    float tn, t1, t2, 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

a menu.sh

```
#!/bin/bash
```

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

```
trap cleanup SIGINT
```

```
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 Uvx и Uvix  "
    "o - Открыть отчет в pdf                    "
    "q - Выход из программы                     "
)
```

```
file_name_zast="./config/zast.txt"
```

```
cleanup() {
```

```
    tput cnorm
    clear
    exit
}

# 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 -rsn1
clear # Очистка экрана
}

# 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]}" "${num}" >> "./data/tabs/table_rpzt.txt"  
if float_compare "${eps}" "<=" "${num}";then  
printf "\nДостигнут допустимая погрешность при  
парамetre: ${arr[1]}\n"  
printf "\nДостигнут допустимая погрешность при  
парамetre: ${arr[1]}\n" >> "./data/tabs/table_rpzt.txt"  
break  
else if [ "${arr[0]}" -gt "$((N/2))" ]; then  
echo " Достигнут предел массива (${N} элементов).  
Остановка"  
echo " Достигнут предел массива (${N} элементов).  
Остановка" >> "./data/tabs/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 -rsn1
```

```
clear # Очистка экрана  
}
```

```
# Функция вывода заставки
```

```
out_zast(){  
clear
```

Потапов Егор Сергеевич ИКПИ-41

```
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

out_menu() {
    for indx in "${!variant_menu[@]}"; do
        echo "${variant_menu[$indx]}"
    done
    echo

    echo -n "Выберите действие 1-4 и о (или q для выхода)"
    read -rsn1 key # Чтение одного символа
    printf "\n"
}

while true; do
    ij=8
    echo -e "Управление меню происходит двумя способами:"
    echo "    Первое-символами(1, 2, 3, 4, о, q)"
    echo "    Второе-стрелочками(вниз-верх, и enter/-> чтоб выбрать вариант)"

    echo -e "\nМеню программы:"
```

```
while true; do
    moving_arrows "${variant_menu[@]}"
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
mv_curs "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 scripts/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!"
```

```
eog data/graphs/graph_Uvx.png > /dev/null 2>&1 #
```

Открытие изображения через eog

```
clear
out_zast
break
else
mv_curs "Error: файлы t/Uvx/Uvix пусты!"
fi
```

```
::
```

5)

```
clear
echo "Закройте файл чтоб вернуться в главное меню!"
open "../note.pdf"
```

```
out_zast
```

```
clear
break
```

```
::
```

6)

```
break 2
```

```
::
```

*)

```
mv_curs "Error: Не верное значение ($key) не входит в
промежуток [1;4] и не является(o, q)!"
```

```
::
```

```
esac
```

```
done
```

```
done
```

cleanup

b functions.sh

```
#!/bin/sh
```

```
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  
}
```

```
mv_curs() {  
    if [ "$ij" != "8" ];then  
        tput cud $((ij-8))  
    fi  
    echo "$1"  
    tput cuu $ij  
    let "ij+=1"
```

}

```
moving_arrows() {
```

```
    arr=("$@")
```

```
    yellow="\033[0;33m"
```

```
    nc="\033[0m"
```

```
    curs=-1
```

```
    up_pred=$(( ${#arr[@]} - 1 ))
```

```
    tput civis
```

```
    while true;do
```

```
        for inx in "${!arr[@]};do
```

```
            if [ "$inx" == "$curs" ];then
```

```
                printf "\r  ${yellow}${arr[$inx]}${nc}\n"
```

```
            else
```

```
                printf "\r  ${arr[$inx]}\n"
```

```
            fi
```

```
        done
```

```
        echo
```

```
        read -rsn1 key
```

```
        [[ -z "$key" ]] && break
```

```
        case $key in
```

```
            1|2|3|4)
```

```
                echo "$key"
```

```
                key=$key
```

```
                echo "$key"
```

```
                return
```

```
        ;;
```

```
        o)
```

```
            key=5
```

```
        return
    ;;
```

```
q)
    key=6
    return
;;
```

```

$'\x1b') # Escape-последовательность (стрелки и др.)
    read -rsn2 -t 0.1 rest # Дополнительно читаем 2 символа
    case "$rest" in
        '[A')
            let "curs-=1"
            curs=$(( curs < 0 ? ${up_pred} : curs))
            ;;
        '[B')
            let "curs+=1"
            curs=$(( curs > ${up_pred} ? 0 : curs))
            ;;
        '[C')
            break
            ;;
    esac
;;

*)
    key=$key
    return
;;
esac
up_rows=$(( ${#arr[@]} + 1 ))
tput cuu $up_rows
done
tput cnorm
```

```
        key=$((curs+1))
    }
```

```
float_compare() {
    local a=$(printf "%f" "$1")
    local op=$2
    local b=$(printf "%f" "$3")

    # Нормализация чисел (удаление лишних нулей и точек)
    a=$(echo "$a" | sed 's/^-\?0\+//; s/\.0*$//; s/^\.0./; s/^-\./-0./; /^$/d')
    b=$(echo "$b" | sed 's/^-\?0\+//; s/\.0*$//; s/^\.0./; s/^-\./-0./; /^$/d')
    [ -z "$a" ] && a=0
    [ -z "$b" ] && b=0
```

```
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);;
    "!=") return $(echo "$a != $b" | bc -l);;
    *)   echo "Неизвестный оператор: $op" >&2; return 2;;
esac
}
```

```
params() {
    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
```

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

2. 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: %g\n"
"$back_front"
```

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

```
printf "    Момент времени, когда Uvx достигает 80 В: %s\n" "Не  
достигает"
```

4. Нахождение момента времени, при котором Uvx достигает максимума

```
time_max=${t[2]}  
max_val=0  
for ((i=2; i<n; i++)); do  
    if float_compare "${Uvx[i]}" ">" "$max_val"; then  
        max_val=${Uvx[i]}  
        time_max=${t[i]}  
    fi  
done  
printf "    Момент времени максимального значения Uvx: %g\n"  
"$time_max"  
  
}
```

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,  
  "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]  
)$
```