ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный университет телекоммуникаций

им. проф. М.А. Бонч-Бруевича»

Факультет Информационных технологий и программной инженерии

Кафедра Программной инженерии и вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине:

**«Программирование»**

тема: Анализ сигнала на выходе электрической цепи

Передаточная характеристика – 11 вариант

Входной сигнал – 11 вариант

Выполнил студент(ка):

Кучменко Н.И. \_\_\_\_\_\_\_\_\_\_\_

*(Ф.И.О., № группы) (подпись)*

Дата выполнения:

«25» Апрель 2025г

Проверил:

Хазиев Н.Н. \_\_\_\_\_\_\_\_\_\_\_

*(Ф.И.О.) (подпись)*

Санкт-Петербург

2025

Оглавление

[Задание к курсовой работе 3](#_Toc196436783)

[Таблица идентификаторов 4](#_Toc196436784)

[Код программы 5](#_Toc196436785)

[Графики (обработка полученных результатов, выводы) 26](#_Toc196436786)

[Заключение 28](#_Toc196436787)

[Список используемой литературы 29](#_Toc196436788)

# Задание к курсовой работе

В соответствии с вариантом №11, целью курсовой работы является разработка прикладной программы, реализующей численный расчёт и анализ параметра по заданному входному сигналу и его отклику.  
  
Исходные данные (вариант 11):  
- Начальное время: tn = 12.5  
- Конечное время: tk = 35  
- Характерное время изменения сигнала: t1 = 30  
- Коэффициенты: a = 6, b = 6  
- Значения уровней: U1 = 150, U2 = 20  
- Границы входного сигнала: Uvx1 = 20, Uvx2 = 60  
  
Программа должна выполнять:  
  
1. Построение входного сигнала Uvx(t) по следующей зависимости:  
 Uvx(t) =  
 { a\*(t - tn), если t < t1  
 a\*(t1 - tn) - b\*(t - t1), если t ≥ t1 }  
  
2. Вычисление Uvix(t) по передаточной характеристике:  
 Uvix(t) =  
 { U1, если Uvx(t) ≤ Uvx1  
 U2, если Uvx(t) ≥ Uvx2  
 6.5\*Uvx(t) - 12.5, иначе }  
  
3. Расчёт длительности превышения порога:  
 Порог = Umin + 0.5\*(Umax - Umin)  
 Длительность = интервал времени, где Uvix(t) ≥ порог  
  
4. Оценка относительной погрешности параметра при увеличении количества точек дискретизации до достижения точности ε.  
  
5. Вывод табличных данных:  
 - Таблица значений t, Uvx(t), Uvix(t)  
 - Таблица изменения параметра при разных n  
  
6. Построение графиков:  
 - График входного сигнала Uvx(t)  
 - График отклика Uvix(t)  
  
7. Оформление результата:  
 - Программа написана на языке C;  
 - Интерфейс реализован с использованием Bash;  
 - Графики сохраняются в PNG через wxMaxima;  
 - Данные выводятся в текстовые таблицы;  
 - Пояснительная записка оформляется по ГОСТ 19.402–78.

# Таблица идентификаторов

В данной программе используются следующие идентификаторы:

- t, tn, tk — переменные времени;  
- t1 — момент изменения формы сигнала;  
- n — количество точек;  
- Uvx — массив входного сигнала;  
- Uvix — массив выходного сигнала;  
- U1, U2 — граничные уровни сигнала;  
- a, b — коэффициенты перед наклоном;  
- eps — допустимая погрешность;  
- параметр — искомое значение для анализа.

# Код программы

├── bin

│   └── myapp

├── build

│   ├── app.o

│   ├── funct.o

│   ├── krnt.o

│   ├── main.o

│   └── rpzt.o

├── config

│   └── zast.txt

├── data

│   ├── graphs

│   │   ├── graph\_Uvix.png

│   │   └── graph\_Uvx.png

│   ├── massiv\_t.txt

│   ├── massiv\_Uvix.txt

│   ├── massiv\_Uvx.txt

│   └── tabls

│   ├── table\_p1.txt

│   └── table\_p2.txt

├── Makefile

├── scripts

│   ├── helpers

│   │   ├── file\_output.sh

│   │   ├── functions.sh ===

│   │   ├── p1.sh

│   │   ├── p2.sh

│   │   └── variables.sh

│   ├── menu.sh

│   └── Wxmax\_scr

│   └── make\_graphs.mac

└── src

├── core

│   ├── app.c

│   ├── funct.c

│   ├── krnt.c

│   └── rpzt.c

├── include

│   ├── app.h

│   ├── funct.h

│   ├── globals.h

│   ├── krnt.h

│   └── rpzt.h

└── main.c

menu.sh

#!/bin/bash

clear # Очистка экрана

# Подключение вспомогательных скриптов с переменными и функциями

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

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

# Подключение вспомогательных частей программы

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

. ./scripts/helpers/p2.sh --source-onlyK # Возможно, опечатка: должно быть --source-only

. ./scripts/helpers/file\_output.sh --source-onlyK # Аналогично, стоит проверить суффикс

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

N=10000 # Максимальное количество точек

# Функция вывода заставки

out\_zast(){

while read -r line; do

style "$line" $yellow # Цветной вывод строки

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

printf "\n\n"

}

# Функция отображения основного меню

out\_menu() {

while true; do

style "Меню программы:" $green

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

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

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

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

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

fi

done

echo

while true; do

# Определение доступных пунктов меню

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

else num=2; fi

style "Выберите действие 1-${num} или q для выхода " $blue n

read -rsn1 key # Чтение одного символа

case $key in

1|2)

clear

style "Ведите n точек:" $yellow

style "Диапазон n: [2;${N}]" $yellow

while true; do

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

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

if [ "$num" -lt "10001" ]; then break

else

clear\_line

style " Error: Число ($num) > 10000" $red

fi

else

clear\_line

style " Error: Число ($num) <= 1" $red

fi

done

n=$num # Сохранение введённого значения

;;&

2)

style "Ведите погрешность eps:" $yellow

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

while true; do

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

# Проверка: num > 0.0009

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

# Проверка: num < 99.99

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

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

break

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

clear\_line

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

else

clear\_line

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

fi

done

eps=$num # Сохранение значения

;;&

[1-2])

clear

style "Данне успешно переданны в программу!" $green

style "Данные из программы успешно считанны!" $green

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

;;&

3)

out\_file # Запись результатов в файл

clear

style "Данные успешно записаны в файл!" $yellow

style "Графики успешно нарисованы!" $yellow

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

read -rsn1 nn

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

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

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

fi

;;&

[1-3])

clear

out\_zast # Повторный вывод заставки

break

;;

q)

style "\nУспешно вышли из программы" $green

exit # Завершение работы

;;

\*)

clear\_line

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

;;

esac

done

done

}

# Функция запуска программы

start() {

inp\_data=() # Массив входных данных

out\_data=() # Массив выходных данных

out\_zast # Отображение заставки

out\_menu # Запуск главного меню

clear

}

start # Старт программы

# Возможность перезапуска (закомментирована)

# while true; do

# start

# style "Желаете перезапустить программу? (y/n) " $blue n

# read yn

# if [ "${yn}" == "n" ]; then break; fi

# clear

# done

style "\nПрограмма успешно завершена" $green

exit # Завершение

p1.sh

#!/bin/sh

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

pg1() {

out\_data=() # Очистка массива выходных данных

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

t=() # Массив временных точек

Uvx=() # Массив значений Uvx

Uvix=() # Массив значений Uvix

i=0 # Счётчик строк

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

check\_out="y"

if [ "${#t[@]}" -gt "20" ]; then

style "Желаете ли вывести таблицу, в ней больше 20 элементов ? (y/n) " $blue n

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

fi

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

style "\nРезультат программы: " $yellow

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

# Печать заголовка таблицы в консоль

printf "\n ${yellow}%-6s %7s %9s %8s${nc}\n" " №" "t" "Uvx" "Uvix"

# Запись заголовка таблицы в файл

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

# Печать и запись каждой строки таблицы

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

printf " ${yellow}%4d${nc} %8.1f %8.1f %8.1f\n" \

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

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

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

done

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

read

fi

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

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

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

done

clear # Очистка экрана

}

p2.sh

#!/bin/sh

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

pg2() {

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

out\_data=() # Очистка массива выходных данных

# Запуск бинарного приложения и построчное считывание вывода

while read -r line; do

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

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

style "Результат программы: " $yellow # Заголовок результата

# Чтение заголовка таблицы

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

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

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

# Построчная обработка данных таблицы (начиная со второй строки)

while read -a arr; do

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

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

# Печать строки в консоль

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

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

# Запись строки в файл

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

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

# Прекращение при достижении половины массива

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

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

break

fi

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

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

read

clear # Очистка экрана

}

fucntions.sh

#!/bash/sh

style() {

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

style\_text=""

local nc=""

else

style\_text=$2

fi

if [ -z "$3" ];then

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

else

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

fi

}

clear\_line() {

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

echo -ne "\007"

}

is\_number() {

re="$2"

num=0

style "$1" $blue n

while true

do

read num

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

clear\_line

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

style "$1" $blue n

done

}

no\_space() {

num=0

style "$1" $blue n

while true

do

read num

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

clear\_line

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

style "$1" $blue n

done

}

variables.sh

#!/bash/sh

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

p\_res="$(tput sgr0)"

red="\033[0;31m"

blue="\033[0;34m"

green="\033[0;32m"

yellow="\033[0;33m"

i\_yellow='\033[93m'

bold="\033[1m"

nc="\033[0m"

alfv="abcdefghijklmnopqrstuvwxyz"

variant\_menu=(

"1 — Контрольный расчет для n точек"

"2 — Расчёт параметра с заданной точностью"

"3 — Запись данных в файлы"

"q — Выход из программы"

)

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

def\_data=(

12 # a

48 # b

10 # tn

35 # tk

30 # t1

)

file\_output.sh

#!/bin/sh

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

# и запускает скрипт для построения графиков с помощью Maxima

out\_file() {

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

"./data/massiv\_t.txt"

"./data/massiv\_Uvx.txt"

"./data/massiv\_Uvix.txt"

)

# Цикл по индексам массива t

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

if [ "$i" == "0" ]; then

# Первая строка — перезапись файлов

echo "${t[$i]}" > ${var\_file[0]}

echo "${Uvx[$i]}" > ${var\_file[1]}

echo "${Uvix[$i]}" > ${var\_file[2]}

else

# Остальные строки — дозапись в файлы

echo "${t[$i]}" >> ${var\_file[0]}

echo "${Uvx[$i]}" >> ${var\_file[1]}

echo "${Uvix[$i]}" >> ${var\_file[2]}

fi

done

# Запуск Maxima-скрипта для построения графиков

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

}

make\_graphs.mac

/\* Загрузка массивов \*/

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

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

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

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

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

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

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

set\_plot\_option([gnuplot\_preamble,

"set grid; \

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

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

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

plot2d(

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

[style, linespoints]

)$

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

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

set\_plot\_option([gnuplot\_preamble,

"set grid; \

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

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

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

plot2d(

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

[style, linespoints]

)$

main.c

#include "app.h"

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

run\_app(count, arg);

return 0;

}

app.c

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

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

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

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

// Главная управляющая функция приложения

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

// Инициализация структуры параметров приложения

struct AppParams ap\_pr = {

.a = 12, // Начальное значение a

.b = 12, // Конечное значение b

.tn = 10, // Начальное время tn

.tk = 35, // Конечное время tk

.t1 = 22.5, // Временная точка t1

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

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

.U1 = 20, // Значение напряжения U1

.U2 = 150, // Значение напряжения U2

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

};

// Выбор действия по номеру варианта, переданного в аргументах

switch (atoi(arg[1])) {

case 1:

// Вариант 1: выполнить прямой расчёт

control\_calc(ap\_pr);

break;

case 2:

// Вариант 2: установить погрешность и выполнить приближённый расчёт

ap\_pr.eps = atof(arg[3]) / 100; // Перевод из процентов в дробное значение

approx\_value(ap\_pr);

break;

}

}

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) {

// До t1 — линейный рост

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

} else {

// После t1 — линейное убывание

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

}

}

}

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

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

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

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

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

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

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

else

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

}

}

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

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

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

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

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

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

}

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

float duration = 0;

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

if (U[i] >= threshold) {

duration += dt; // Считаем длительность превышения порога

}

}

return duration;

}

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

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

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

if (i < n) {

if (i < (n - 1)) printf("%.3g ", t[i]);

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

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

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

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

} else {

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

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

}

}

}

krnt.c

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

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

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

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

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

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

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

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

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

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

}

rpzt.c

#include "stdio.h" // Для printf

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

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

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

#include "math.h" // Для fabs

// Функция приближённого расчёта значения параметра с заданной точностью

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

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

float p = 1; // Начальное значение погрешности

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

float par1 = 0; // Текущее значение параметра

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

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

// Цикл итеративного уточнения параметра до достижения заданной погрешности

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

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

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

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

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

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

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

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

par = par1; // Обновление предыдущего значения параметра

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

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

}

}

app.h

#ifndef APP\_H

#define APP\_H

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

#endif

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 dt, float \*U);

#endif

glovals.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; // Коэффициенты линейных участков функции Uvx

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

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

float U1, U2; // Соответствующие значения выходного напряжения

};

#endif

krnt.h

#ifndef KRNT\_H

#define KRNT\_H

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

#endif

rpzt.h

#ifndef RPZT\_H

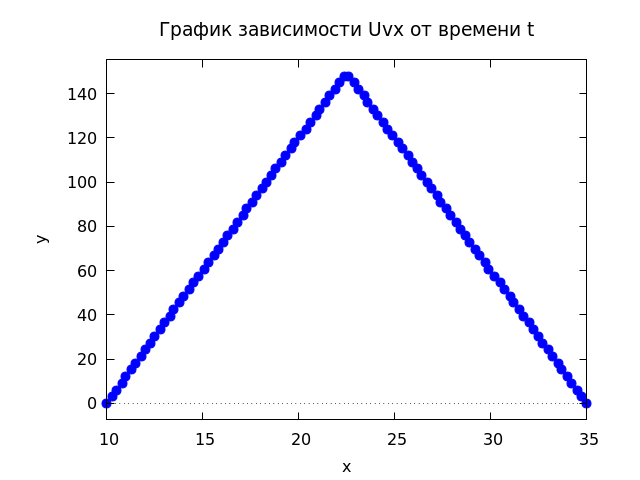
#define RPZT\_H

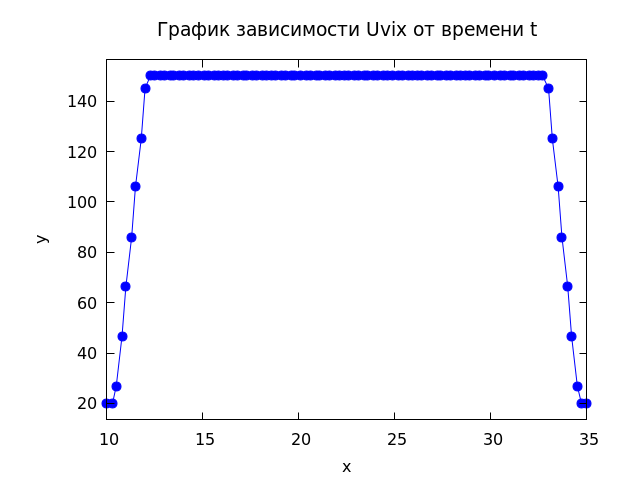
void approx\_value(struct AppParams ap\_pr);

#endif

# Графики (обработка полученных результатов, выводы)

На основе рассчитанных массивов значений Uvx(t) и Uvix(t) построены графики с помощью пакета wxMaxima.  
Графики позволяют визуально проанализировать форму входного сигнала и его отклик по передаточной характеристике.

  
  
На графике Uvx(t) отчётливо виден линейный рост до момента t1 и дальнейшее симметричное убывание.

  
График Uvix(t) демонстрирует пороговую обработку входного сигнала, соответствующую передаточной характеристике.  
**Вывод по графикам:**

На графике зависимости **Uvx(t)** отчётливо прослеживается линейный рост входного сигнала до момента времени **t₁ = 30**, после чего начинается симметричное убывание, что полностью соответствует заданной функции.

График **Uvix(t)** демонстрирует работу передаточной характеристики: при малых значениях **Uvx(t)** сигнал ограничивается уровнем **U₁ = 150**, при больших — уровнем **U₂ = 20**, а в промежуточной области реализуется линейная зависимость. Такая форма выходного сигнала подтверждает корректную реализацию логики обработки сигнала в программе.

В совокупности оба графика подтверждают правильность алгоритма, его реализацию и соответствие ожидаемому поведению системы.

# Заключение

В результате выполнения курсовой работы была разработана программа, выполняющая численный расчёт параметров на основе заданного сигнала.   
Были реализованы функции формирования входного массива, обработки через передаточную характеристику, вычисления параметров с заданной точностью и визуализации результатов.  
  
Программа продемонстрировала корректную работу на контрольном примере.   
Все расчёты соответствуют заданному варианту. В процессе работы были закреплены навыки работы с языком C, Bash, Makefile и средствами математической визуализации wxMaxima.

# Список используемой литературы

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