МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова»

Факультет **информационных технологий и управления**

Кафедра **«Программное обеспечение вычислительной техники»**

Направление **09.04.01 – Информатика и вычислительная техника**

ОТЧЕТ

по Лабораторной работе №1

по дисциплине: **Программное и аппаратное обеспечение информационных систем**

Выполнил студент 1 **курса, группы** ТИСа-о24 Блохин Э.Е.
Фамилия, имя, отчество

Принял доцент, кандидат технических наук Рыбалкин А.Д.
Фамилия, имя, отчество

«_____» _____2024 г. _____
Подпись

Новочеркасск, 2024 г

Лабораторная работа №1

«Применение CASE-средств для проектирования и разработки программного и аппаратного обеспечения ИС и АС»

Цель работы: Изучить классификацию и применение CASE-средств для разработки программного обеспечения, освоить их практическое использование для проектирования и документирования информационных систем, а также разработать первую часть индивидуального задания.

Теоретический материал: CASE-средства (Computer-Aided Software Engineering) — это программные инструменты, которые поддерживают процесс проектирования, разработки и сопровождения программного обеспечения. Они автоматизируют многие этапы жизненного цикла программного продукта, облегчая задачу разработчикам и повышая эффективность работы.

CASE-средства можно разделить на несколько категорий:

- Средства верхнего уровня — используются для проектирования системы, анализа требований и создания моделей. Примеры: UML-диаграммы, диаграммы потоков данных;
- Средства нижнего уровня — помогают непосредственно в написании кода, тестировании и сопровождении программного обеспечения;
- Интегрированные CASE-средства — объединяют весь процесс разработки, от проектирования до внедрения, поддерживая команды разработчиков на всех этапах.

Кроме того, CASE-средства часто используются для моделирования баз данных, управления проектами и документирования. Ключевой элемент большинства систем — UML, который позволяет визуализировать структуру и поведение программных систем с помощью диаграмм.

Для успешного выбора CASE-инструмента важно учитывать его интеграцию с другими системами, гибкость и возможность масштабирования, а также удобство для команды разработчиков.

Ход работы:

1) Разработана диаграмма, отражающая структуру программы. Схема включает три ключевых блока: экспоненциальный компонент, косинусный компонент и логарифмический компонент, которые складываются в общий сигнал, что продемонстрировано на рисунке 1.

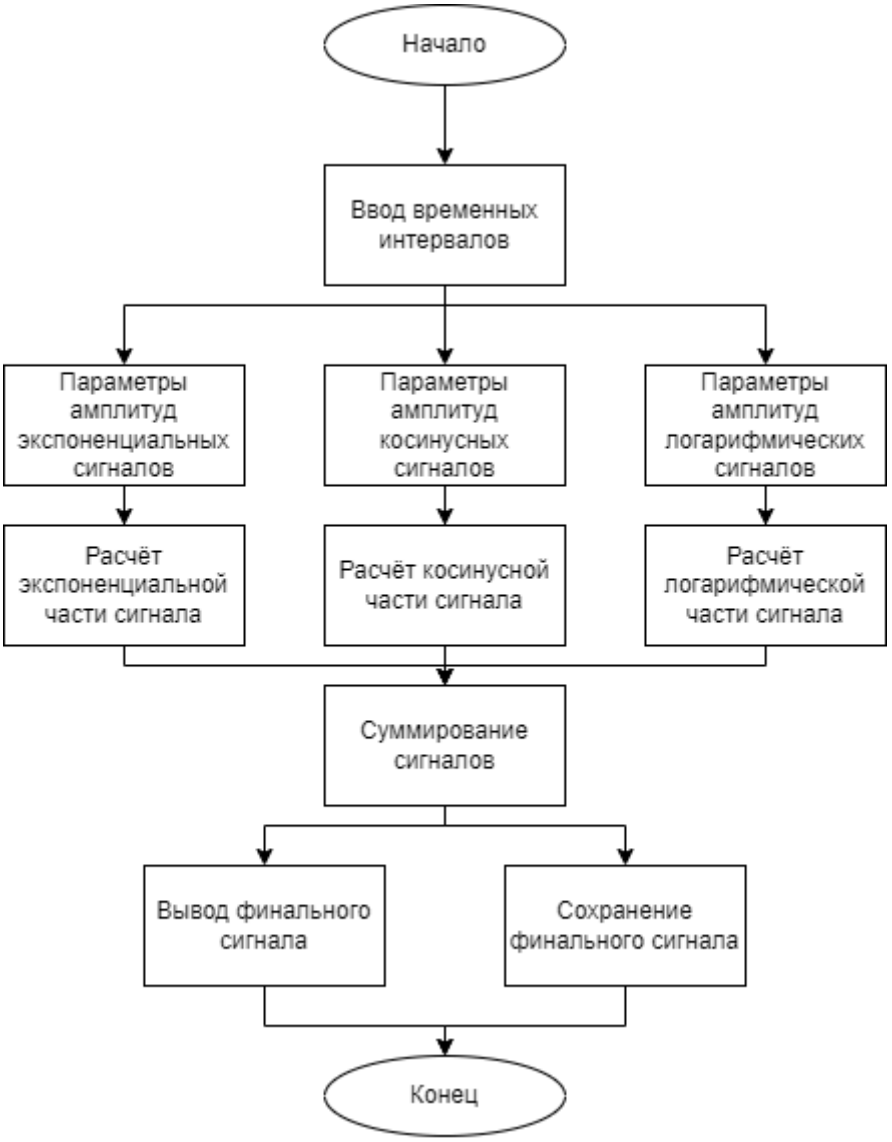

Рисунок 1 – Структурная схема программы

2) В первой части кода создается функция create_signal, которая

принимает несколько параметров для генерации сигнала:

- time: временные точки, для которых будет генерироваться сигнал;
- num_exp, num_cos, num_log: количество экспоненциальных, косинусных и логарифмических членов соответственно;
- amp_exp, amp_cos, amp_log: массивы амплитуд для всех членов.

Инициализируется начальный сигнал с нулевыми значениями, а также задаются константы для экспоненциальной функции, косинусных членов и логарифмических терминов, что показано на рисунке 2.

```
def create_signal(time, num_exp, num_cos, num_log, amp_exp, amp_cos, amp_log)
    total_signal = np.zeros_like(time)

    # Константы
    exp_const = 1  # Константа для экспоненциальной функции
    freq_base = 2 * np.pi  # Базовая частота для косинусных членов
    phase_shift = 0  # Начальный фазовый сдвиг для косинусных членов
    log_const = 1  # Константа для логарифмической функции
    k_base = 1  # Базовое значение k для логарифмов
```

Рисунок 2 – Функция генерации сигналов и константы

3) Добавление экспоненциальных, косинусных и логарифмических членов к сигналу, используя заданные формулы. Экспоненциальные члены добавляются для моделирования спадающего сигнала с различными амплитудами, косинусные — для гармонических колебаний, частота которых увеличивается с каждым добавленным членом, а логарифмические — для моделирования сигналов с нелинейным ростом на основе логарифмов. Каждый из этих типов сигналов добавляется поочередно, и итоговый сигнал формируется как сумма всех компонентов, что показано на рисунке 3.

```python
# Добавляем экспоненциальные члены
if num_exp > 0:
    for i in range(num_exp):
        exp_amp = amp_exp[i] if i < len(amp_exp) else 0
        total_signal += exp_amp * np.exp(-time / exp_const)

# Добавляем косинусные члены
if num_cos > 0:
    for j in range(num_cos):
        cos_amp = amp_cos[j] if j < len(amp_cos) else 0
        frequency = freq_base * (j + 1)
        total_signal -= cos_amp * np.cos(frequency * time + phase_shift)

# Добавляем логарифмические члены
if num_log > 0:
    for k in range(num_log):
        log_amp = amp_log[k] if k < len(amp_log) else 0
        k_value = k_base * (k + 1)
        log_term = np.log10(np.maximum(log_const * k_value * time, 1e-10))
        total_signal += log_amp * log_term

return total_signal
```

Рисунок 3 – Добавление компонентов к сигналу

4) В примере использования функции create_signal задается временной диапазон от 0.1 до 10 секунд с 1000 равномерно распределенными точками, чтобы избежать проблем с логарифмом от нуля. Затем задаются параметры для каждого типа сигнала: экспоненциальных, косинусных и логарифмических членов. В частности, для экспоненциальных членов указывается, что их два, для косинусных — один, а для логарифмических — один. Соответственно, для каждого типа сигнала заданы массивы амплитуд: для экспоненциальных — два значения, для косинусных и логарифмических — по одному значению. После этого вызывается функция create_signal, которая использует эти параметры для генерации итогового сложного сигнала, состоящего из всех трёх компонентов, что продемонстрировано на рисунке 4.

```python
t = np.linspace(0.1, 10, 1000)
n = 3  # Экспоненциальные члены
m = 1  # Косинусные члены
l = 0  # Логарифмические члены
A_vals = [0.35, 0.25, 0.4]
B_vals = [1]
C_vals = [0]

signal = create_signal(t, n, m, l, A_vals, B_vals, C_vals)

plt.plot(t, signal)
plt.title('Сгенерированный сигнал')
plt.xlabel('Время (t)')
plt.ylabel('Сигнал (s)')
plt.show()
```

Рисунок 4 – Генерация сложного сигнала

Результаты работы:

После генерации итогового сигнала с помощью функции create_signal, этот сигнал визуализируется на графике с использованием библиотеки Matplotlib. Вызов функции plt.plot(t, signal) строит график зависимости сигнала от времени, где по оси X откладывается время, а по оси Y — значение сигнала. Заголовок графика задается с помощью plt.title('Сгенерированный сигнал'), а оси подписываются с помощью plt.xlabel('Время (t)') и plt.ylabel('Сигнал').

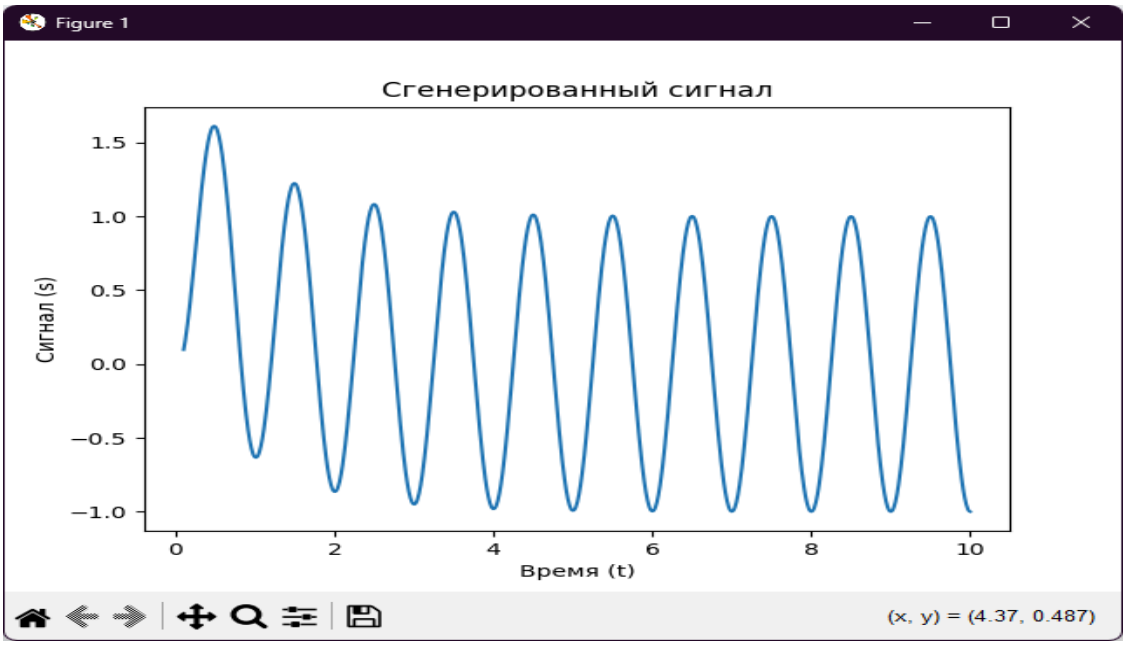

Рисунок 5 – Визуализированный сигнал

Вывод: Разработана программа для генерации сложных сигналов, объединяющая экспоненциальные, косинусные и логарифмические компоненты. Программа успешно создает и визуализирует сигнал, позволяя варьировать параметры и амплитуды каждого компонента. Итоговый график подтверждает корректность работы программы и её применимость для моделирования сигналов.