**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 №2**

по дисциплине: **Программное и аппаратное обеспечение**

**информационных систем**

**Выполнил студент** 1 **курса, группы** ТИСа-о24 Якимов П.В.

Фамилия, имя, отчество

**Принял доцент, кандидат технических наук** Рыбалкин А.Д.

Фамилия, имя, отчество

«\_\_\_\_\_» **\_\_**\_\_\_\_\_\_\_\_\_\_\_\_2024 г. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Подпись

Новочеркасск, 2024 г

**Лабораторная работа №2**

**«Разработка программной части ИС и АС»**

**Цель работы:** Изучить методы сопровождения исходного кода, создания и использования репозиториев методы получения сигналов и данных, методы предварительной обработки информации перед применением бизнес-логики приложения.

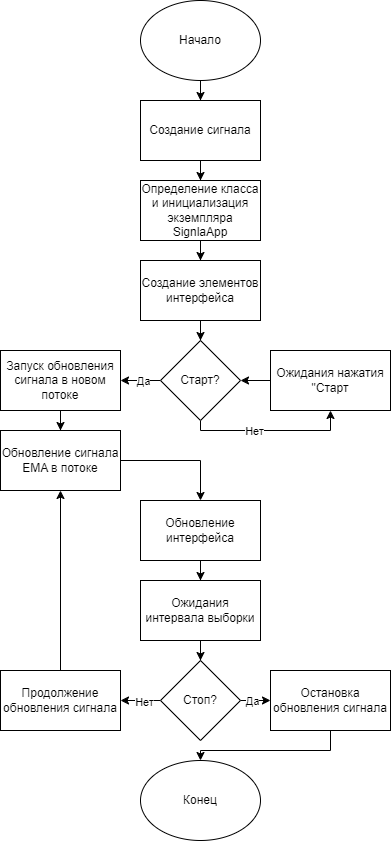
**Теоретический материал:** Современные методы разработки программного обеспечения включают нативные и кроссплатформенные инструменты. Нативные инструменты, такие как C++ и Java, предназначены для создания приложений на определённых платформах. Кроссплатформенные решения, например, JavaScript и Python, позволяют разрабатывать программы, работающие на разных операционных системах.

Системы контроля версий (VCS), такие как Git, помогают управлять изменениями в коде и совместной работой в команде. Основные команды Git, такие как git init, git add, git commit, git push, позволяют организовать процесс разработки и отслеживать историю изменений. Платформы вроде GitHub и GitLab предоставляют инструменты для управления проектами и документации. Предварительная обработка данных включает методы фильтрации и сглаживания для повышения качества данных. Алгоритмы, такие как скользящие средние и экспоненциальное сглаживание (EMA), помогают устранять шумы и улучшать анализ сигналов.

Использование аппаратных средств, таких как Arduino и Raspberry Pi, позволяет получать данные от датчиков и тестировать алгоритмы обработки сигналов. При отсутствии физического оборудования возможно использование математических моделей, файлов или внешних API.

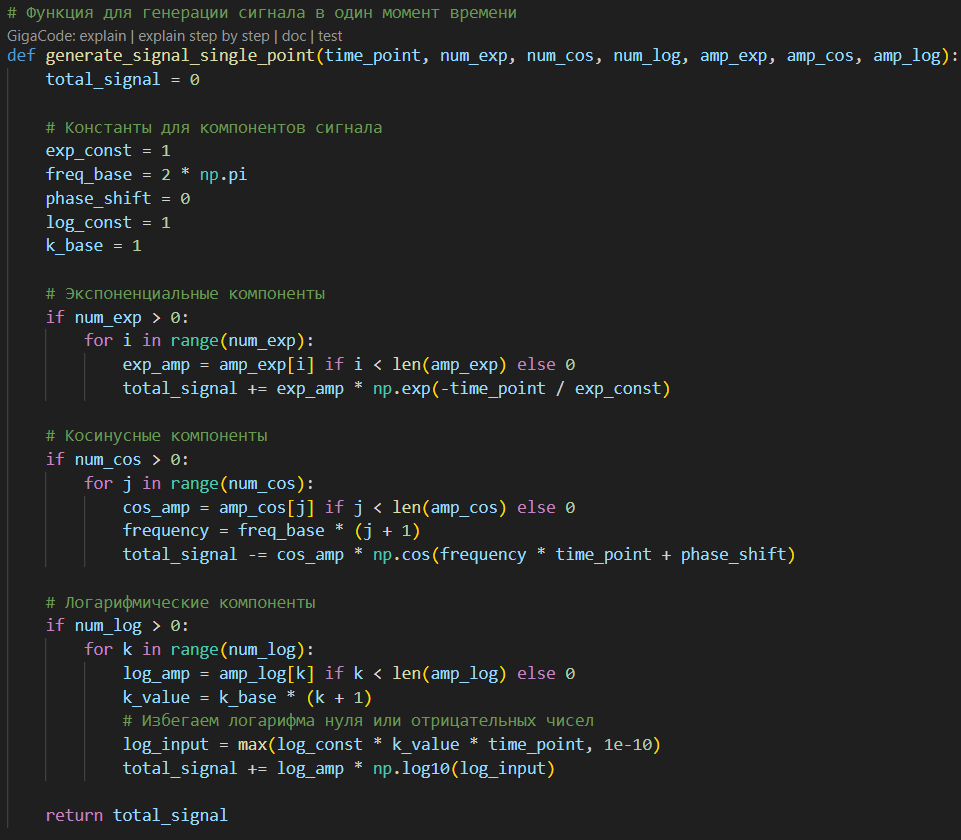
UML и моделирование служат для визуального представления программных систем. UML позволяет создавать диаграммы классов, последовательностей и потоков данных, что помогает разработчикам лучше понимать структуру и поведение системы.

**Ход работы:**

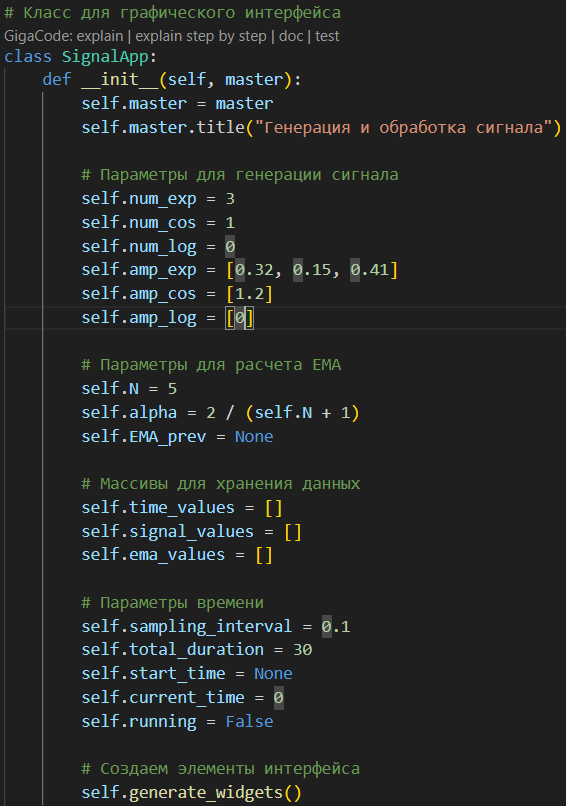
1) Разработана блок-схема, демонстрирующая последовательность работы программы. Схема включает ключевые этапы: определение функций и класса SignalApp, а также обработку нажатий кнопок "Старт" и "Стоп", что иллюстрирует общую логику приложения, как показано на рисунке 1.

*Рисунок 1 – Блок-схема программы*

2) В первой части кода происходит импорт необходимых библиотек, таких как NumPy, для числовых операций, модули для работы с временем и потоками, а также библиотеки для создания графического интерфейса и построения графиков. Основной функцией является *generate\_signal\_single\_point*, которая принимает параметры, включая временную точку и количество членов для различных компонентов сигнала. Внутри функции инициализируются константы для экспоненциального затухания, базовой частоты косинусных членов и логарифмических компонентов, а затем происходит суммирование сигналов, что иллюстрируется на рисунке 2.

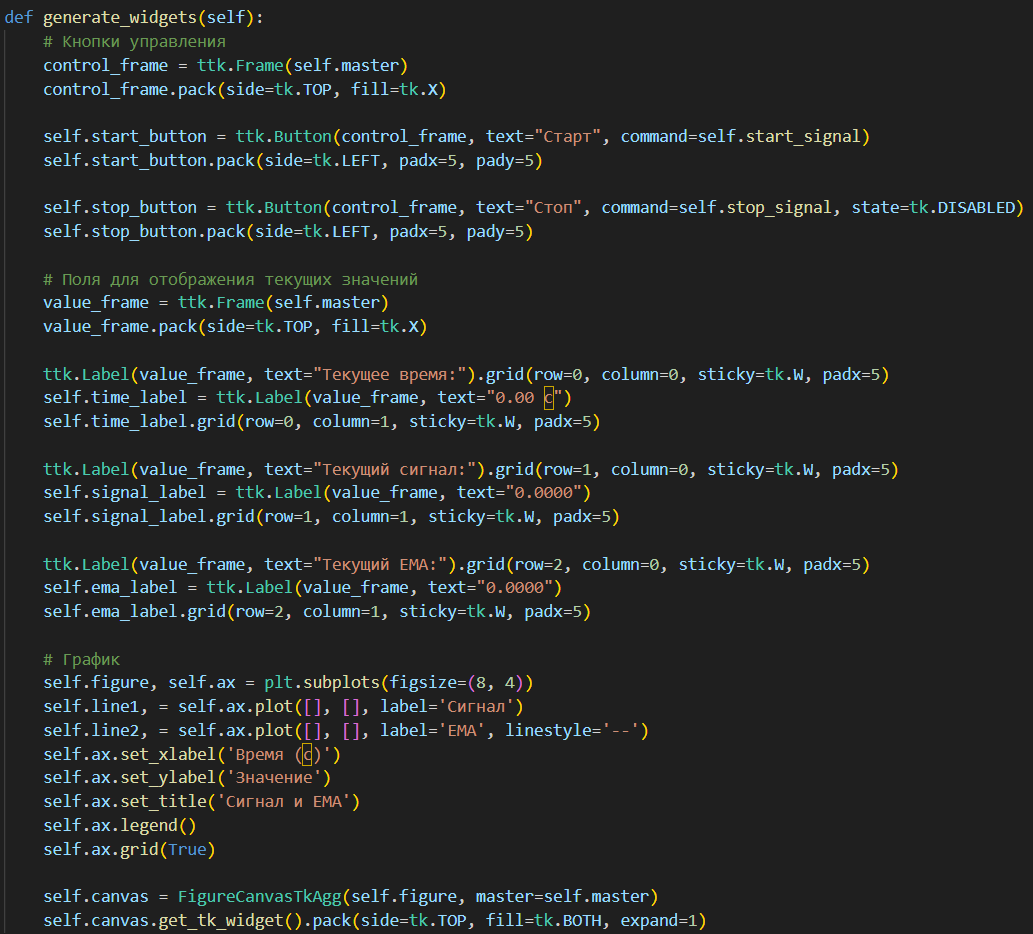


*Рисунок 2 – Создание функции для генерации сигнала*

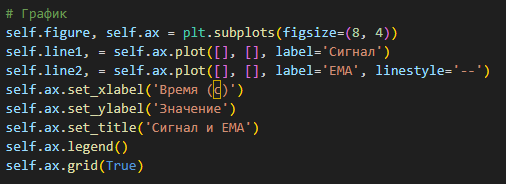
3) Далее пределяется класс SignalApp, который отвечает за создание графического интерфейса для генерации и обработки сигнала. Внутри конструктора инициализируются параметры для генерации сигнала, а также массивы амплитуд для каждого типа сигнала. Задаются параметры для расчета экспоненциального скользящего среднего, включая сглаживающий фактор и массивы для хранения данных. В конце инициализации вызывается метод generate\_widgets, задающий элементы интерфейса, что показано на рисунке 3.

*Рисунок 3 – Определение класса SignalApp*

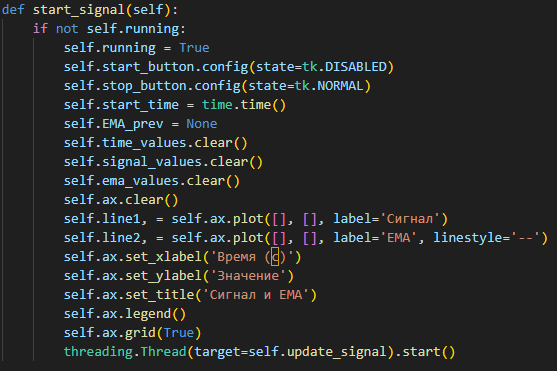
4) Метод generate\_widgets отвечает за создание и размещение элементов управления в графическом интерфейсе. В нем создаются кнопки для начала и остановки генерации сигнала, а также поля для отображения текущих значений времени, сигнала и EMA. Каждое поле подписано, чтобы можно было легко следить за изменениями значений. Эти элементы расположены в соответствующих фреймах, обеспечивая структурированный и удобный интерфейс. Всё это проиллюстрировано на рисунке 4.



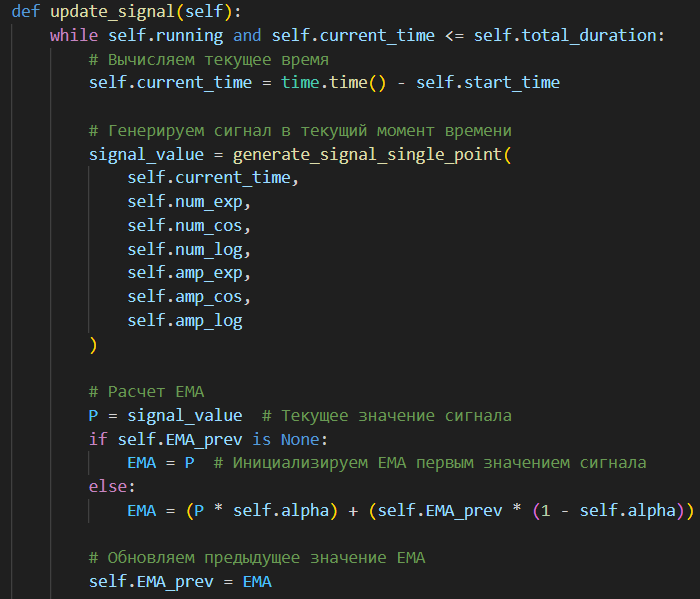
*Рисунок 4 – Создание элементов интерфейса*

5) В данной части кода создается график для визуализации сигнала и EMA. Используя Matplotlib, создается фигура и оси, на которых будут отображаться данные. Устанавливаются метки для осей и заголовок графика, а также создаются линии для сигнала и EMA. Затем график упаковывается в интерфейс, что позволяет видеть визуализацию сигналов в реальном времени, что демонстрирует рисунок 5.

*Рисунок 5 – Настройка графика для визуализации данных*

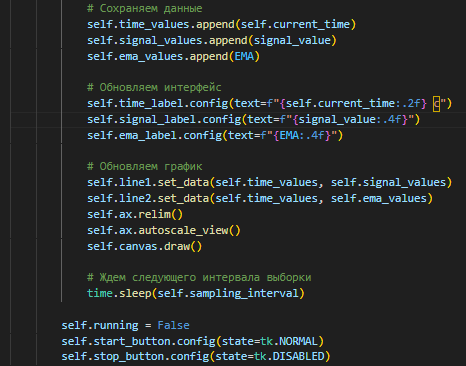
6) Методы start\_signal и stop\_signal управляют процессом генерации сигнала. При запуске метода start\_signal проверяется, не выполняется ли уже генерация. Если нет, происходит инициализация необходимых параметров, таких как время и массивы для хранения данных. Затем запускается отдельный поток, который будет обновлять данные сигнала. Метод stop\_signal останавливает генерацию сигнала. Все действия показаны на рисунке 6.

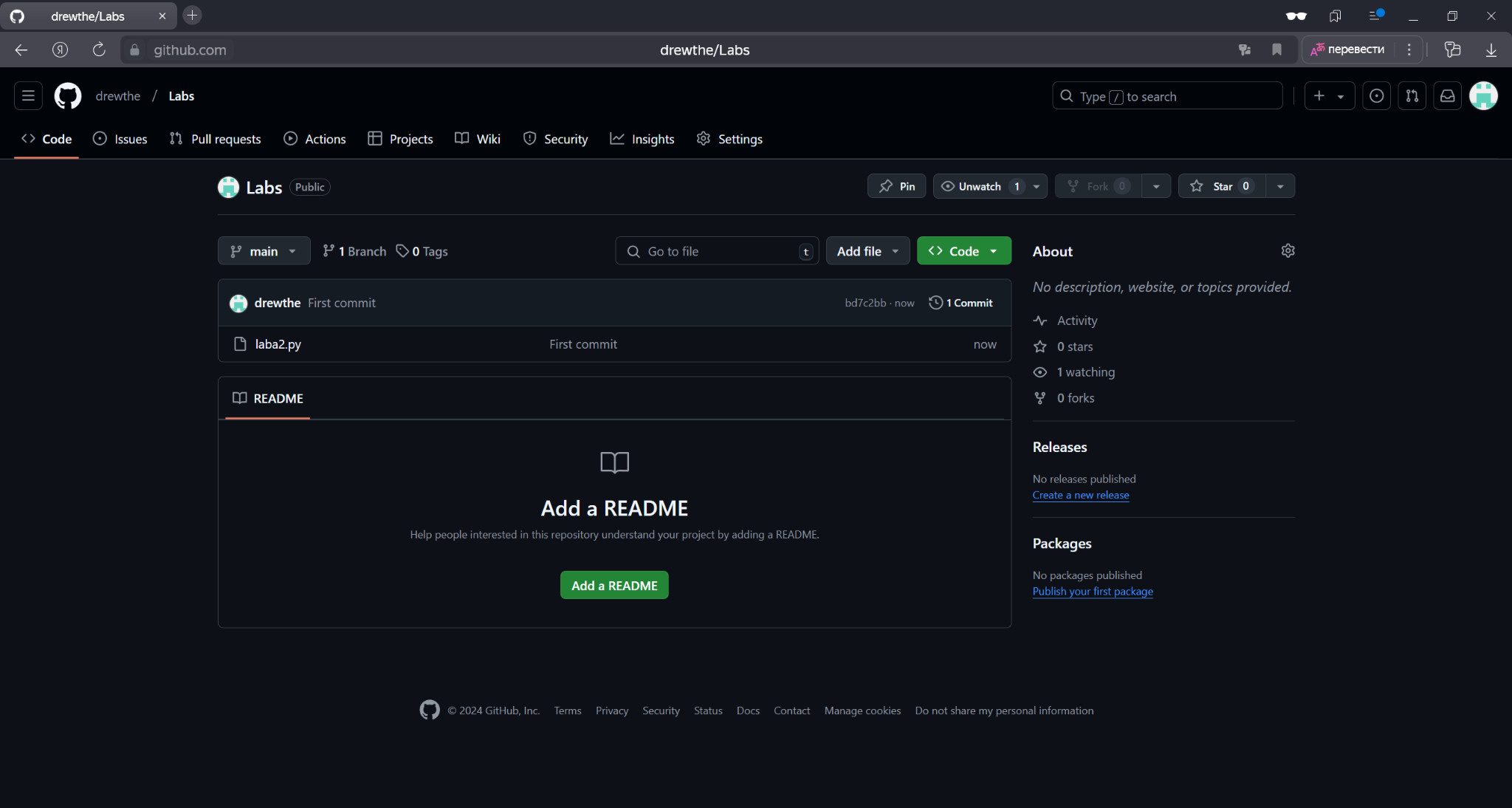
*Рисунок 6 – Управление процессом генерации сигнала*

7) В методе update\_signal осуществляется непрерывное обновление сигнала, пока работает генерация. В этом методе вычисляется текущее время и вызывается функция generate\_signal\_single\_point для генерации сигнала в данный момент. Также рассчитывается EMA на основе текущего значения сигнала и предыдущего значения EMA, если оно доступно. Это позволяет отслеживать изменения сигнала в реальном времени и показано на рисунке 7.

*Рисунок 7 – Обновление сигнала в реальном времени*

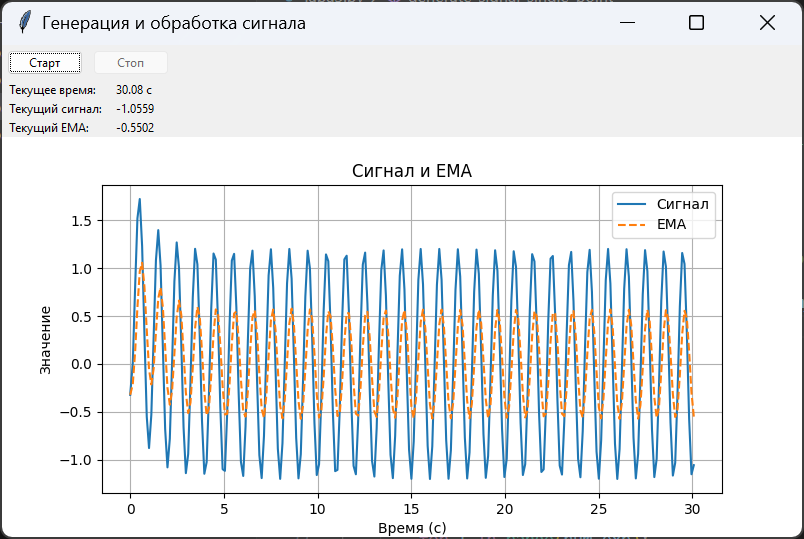
8) В последней части кода сохранение данных, таких как текущее время, значение сигнала и EMA, в соответствующие массивы. После этого обновляются метки в интерфейсе с текущими значениями, а также обновляется график, чтобы отобразить новые данные. График пересчитывается и перерисовывается, чтобы отражать изменения в сигнале и EMA. Наконец, происходит ожидание следующего интервала выборки, что обеспечивает плавную работу приложения. Происходит запуск приложения, где создается корневое окно с помощью Tkinter и инициализируется класс SignalApp. После этого запускается главный цикл обработки событий интерфейса, что позволяет взаимодействовать с графическим интерфейсом и наблюдать за изменениями сигналов в реальном времени.

*Рисунок 8 – Сохранение данных, обновление интерфейса и запуск*

9) На этом этапе проект загружается на платформу для размещения репозиториев, GitHub. Сначала инициализируется репозиторий с помощью команды *git init*, затем добавляются все файлы проекта с помощью команды *git add .*. После этого выполняется коммит изменений с описанием, например, *git commit -m "First commit"*. Для подключения к удаленному репозиторию используется команда *git remote add origin*, после чего проект загружается на удалённый сервер командой *git push -u origin master*. В последствие, проект переносится на GitHub. Весь процесс и результат показаны на рисунке 9.

*Рисунок 8 – Перенос проекта в GitHub*

**Результаты работы:**

После нажатия кнопки "Старт" происходит генерация сигнала с использованием функции generate\_signal\_single\_point, который состоит из экспоненциальных, косинусных и логарифмических компонентов. График отображает зависимость сигнала от времени: по оси X — время, по оси Y — значение сигнала. Основной сигнал представлен сплошной линией, а скользящее среднее (EMA) — пунктирной. Заголовок графика и подписи осей задаются соответствующими методами Matplotlib. В реальном времени обновляются текущие значения времени, сигнала и EMA, позволяя наблюдать за динамикой сигнала на протяжении всего периода генерации (рисунок 10).

*Рисунок 9 – Визуализированный сигнал*

**Вывод:** Разработана программа для генерации и визуализации сложных сигналов. Она демонстрирует методы обработки данных в реальном времени и расчет скользящего среднего (EMA). Графический интерфейс обеспечивает удобное взаимодействие с пользователем, а результаты подтверждают практическую применимость приложения для анализа сигналов.