МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ОТЧЕТ

по лабораторной работе

по дисциплине

Методы и средства обработки сигналов

РУКОВОДИТЕЛЬ:

_____ Авербух М. Л.
(подпись) (фамилия, и.,о.)

СТУДЕНТ:

_____ Халеев А.А.
(подпись) (фамилия, и.,о.)

21-ВМз
(шифр группы)

Работа защищена «___»_____

С оценкой _____

Нижний Новгород 2023

Задание

Имеем функцию $y(x) = a_1 \cdot \sin(b_1 \cdot x) + a_2 \cdot \sin(b_2 \cdot x) + a_3 \cdot \sin(b_3 \cdot x)$

1) Пользователем задаются:
   a_1, b_1, a_2, b_2, a_3, b_3, x_0 (начальное значение), $x_к$ (конечное значение), Δx (шаг).
2) Расчет y(x) по заданным значениям a1, b1, a2, b2, a3, b3, x0 (начальное значение), xк (конечное значение), Δx (шаг).
3) Отображение векторов x и y (в виде таблицы) с возможностью редактирования.
4) Построение графика y(x) по указанным векторам.
5) Наложение случайного шума на сигнал (отображение на графике)
6) Восстановление искаженного сигнала (отображение на графике)

Проектное решение

Задание выполнено на языке Python, с реализацией графического пользовательского интерфейса для наглядности результатов, а также для упрощения анализа результатов работы.

Функциональные возможности разрабатываемого приложения:

1) Построение графиков функции, искаженного и восстановленного отображения функции
2) Пользовательская регулировка уровня шума
3) Выбор фильтра для восстановления функции
4) Изменение параметров функции, предельных значений построения, шага дискретизации
5) Управление масштабом графика по оси «Y» в случае выхода графика за границы видимой области
6) Сохранение графика в виде изображения
7) Увеличение части графика, перемещение по графику
8) Настройка отображения осей

Зависимости / используемые библиотеки / инструменты приложения:

1. **NumPy (numpy):** Библиотека для работы с массивами и математических операций. Используется для генерации данных и операций с массивами.

2. **Pandas (pandas):** Библиотека для анализа и манипуляции данными. Применяется в классах **Exponential** и **MovingAverage** для работы со скользящим средним и экспоненциальным сглаживанием.

3. **Matplotlib (matplotlib):** Библиотека для создания статических, интерактивных и анимированных визуализаций в Python. В коде используется для построения графиков.

4. **SciPy (scipy):** Библиотека, используемая для научных и технических вычислений. В коде импортируются функции **medfilt** для медианного сглаживания и **gaussian_filter** для гауссовского сглаживания.

5. **PySide6 (PySide6):** Библиотека Python, предоставляющая доступ к Qt библиотекам для создания графического интерфейса пользователя.

6. **Matplotlib Qt backend (matplotlib.backends.backend_qt5agg):** Используется для интеграции визуализаций Matplotlib с графическим интерфейсом пользователя Qt.

Эти библиотеки и инструменты вместе образуют основу для разработки приложения, которое может обрабатывать и визуализировать данные, предоставлять пользовательский интерфейс и применять различные методы сглаживания данных.

Ход выполнения

Построение графиков

Класс **PlotLayout** наследуется от **QVBoxLayout** и **QWidget**, и организует построение графиков в несколько этапов:

1. **Инициализация:**
    - Конструктор принимает параметры сигнала **a1**, **a2**, **a3**, **b1**, **b2**, **b3**, начальное и конечное значения **x0**, **xk** и шаг **dx**.
    - Создается словарь **self.params**, содержащий эти параметры для дальнейшего использования при построении графиков.
    - Создается холст **MplCanvas** для рисования графиков с использованием **matplotlib**.
    - Устанавливается начальный уровень шума **self.noise_lvl** и стратегия сглаживания **self.sm_strategy**.
2. **Сигналы:**
    - Определены сигналы **noise_lvl_changed**, **strategy_changed**, и **params_changed**, которые связаны с методом **draw_plot**. Эти сигналы эмитируются, когда соответствующие параметры изменяются.
3. **Построение графика (draw_plot):**
    - Метод **draw_plot** вызывается для инициализации графика и в ответ на изменения параметров.
    - Очищает текущий график перед новой отрисовкой.
    - Рассчитывает значения **x** и **y** для оригинального сигнала на основе текущих параметров и добавляет шум к **y**, создавая зашумленный сигнал **y_noisy**.
    - Применяет выбранную стратегию сглаживания к зашумленному сигналу, получая **y_smooth**.
    - Рисует на графике оригинальный сигнал, зашумленный сигнал и восстановленный сигнал.
    - Добавляет легенду, заголовок и фиксирует ось Y с учетом масштаба **self.y_scale**.
    - Перерисовывает график.
4. **Изменение уровня шума (change_noise_lvl):**
    - Метод **change_noise_lvl** изменяет **self.noise_lvl** и эмитирует сигнал **noise_lvl_changed**.

5. **Изменение стратегии сглаживания (change_strategy):**
    - Метод **change_strategy** изменяет объект **self.sm_strategy** в зависимости от переданного названия стратегии и эмитирует сигнал **strategy_changed**.
6. **Изменение параметров (change_params):**
    - Метод **change_params** обновляет словарь **self.params** новыми значениями и эмитирует сигнал **params_changed**.
7. **Изменение масштаба по оси Y (y_scale_inc и y_scale_dec):**
    - Методы **y_scale_inc** и **y_scale_dec** используются для увеличения и уменьшения значения **self.y_scale**, которое влияет на масштаб оси Y графика.
    - При увеличении масштаба (**y_scale_inc**), **self.y_scale** увеличивается на 1, и эмитируется сигнал **params_changed**, что вызывает перерисовку графика.
    - При уменьшении масштаба (**y_scale_dec**), **self.y_scale** уменьшается на 1, но не меньше нуля, чтобы избежать отрицательного масштаба.
      Если **self.y_scale** уже равен 0, уменьшение не происходит. Также, после уменьшения масштаба, эмитируется сигнал **params_changed** для обновления графика.

Класс **PlotLayout** представляет собой комплексную систему для построения и обновления графиков в зависимости от пользовательских вводов и взаимодействий. Он интегрирует **matplotlib** для рисования графиков и использует сигнально-слотовый механизм Qt для обновления графиков в ответ на изменения параметров.

Наложение шума

Наложение шума в классе **PlotLayout** реализовано в методе **draw_plot**. Вот как это происходит:

1. **Генерация X и У:**
    - Сначала создается массив **x** с использованием функции **np.arange**, которая генерирует последовательность чисел от **self.params['x0']** до **self.params['xk']** с шагом **self.params['dx']**.
    - Затем вычисляется соответствующий массив **y** путем подставления значений **x** в формулу оригинального сигнала.
2. **Генерация шума:**
    - Шум генерируется с помощью функции **np.random.normal**, которая создает массив случайных значений, распределенных по нормальному (гауссову) распределению с математическим ожиданием **0** и стандартным отклонением **self.noise_lvl**, причем размер массива равен длине массива **y**.
3. **Наложение шума на сигнал:**
    - Сгенерированный шумовой массив **noise** складывается с массивом **y**, создавая зашумленный сигнал **y_noisy**.

Этот зашумленный сигнал **y_noisy** затем отображается на графике в методе **draw_plot** вместе с оригинальным сигналом и сигналом после сглаживания, демонстрируя эффект шума на данные.

Используемые фильтры

В коде реализованы четыре стратегии сглаживания шумных данных:

1. **Экспоненциальное сглаживание (Exponential):** Экспоненциальное сглаживание придает более новым данным больший вес, что позволяет отслеживать тренды более чутко к недавним изменениям. В этом классе используется метод **ewm** (Exponential Weighted functions) из библиотеки pandas с параметром **span**, который определяет скорость убывания весов. Чем меньше значение **span**, тем больший вес имеют последние значения. Возвращаемое значение - сглаженный набор данных.

2. **Гауссово сглаживание (Gaussian):** Гауссово сглаживание использует гауссовский фильтр, применяя свертку шумных данных с гауссовой функцией. Параметр **sigma** контролирует степень сглаживания; большие значения **sigma** приводят к более сильному сглаживанию. Гауссово сглаживание эффективно сглаживает шум и сохраняет края.

3. **Медианное сглаживание (Median):** Медианное сглаживание - это нелинейный фильтр, который заменяет каждый элемент массива медианой элементов в окне заданного размера (**kernel_size**). Этот метод хорошо справляется с сигналами, которые содержат выбросы или импульсный шум, поскольку медиана устойчива к таким выбросам.

4. **Скользящее среднее (Moving Average):** Скользящее среднее сглаживает временные ряды путем замены каждого значения средним значением в окне фиксированного размера (**window_size**). Это простой и часто используемый метод сглаживания, который уменьшает временную вариабельность и выделяет долгосрочные тенденции.

Каждый из этих фильтров реализуется в виде класса, который наследует от абстрактного базового класса **SmoothStrategy**, и каждый класс предоставляет реализацию метода **get_y**, который принимает зашумленный сигнал **y_noisy** и возвращает сглаженный сигнал.

Приложение 1

Программный код

main.py:

```
from sys import argv

from PySide6.QtGui import QIcon
from PySide6.QtWidgets import QApplication

from source.app import MainWindow


def main():
    app = QApplication(argv)
    window = MainWindow()
    window.setWindowIcon(QIcon("designed_ui/icons/logo.png"))
    window.show()
    app.exec()


if __name__ == '__main__':
    main()
```

app.py:

```
import os
from enum import IntEnum
from fnmatch import fnmatch

from PySide6.QtCore import Signal
from PySide6.QtWebEngineWidgets import QWebEngineView
from PySide6.QtWidgets import QInputDialog, QMainWindow, QStackedLayout, QTextEdit

from functools import partial
from designed_ui.designed_interface import Ui_MainWindow
from source.plot import PlotLayout

# from designed_interface import Ui_MainWindow


class MainWindow(QMainWindow, Ui_MainWindow):
    temp_saved = Signal()
    temp_loaded = Signal()
    temp_generated = Signal()

    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.current_dir = os.path.dirname(os.path.abspath(__file__))

        self.plot_lay = PlotLayout(self.a1_spin.value(), self.a2_spin.value(),
                                   self.a3_spin.value(), self.b1_spin.value(),
                                   self.b2_spin.value(), self.b3_spin.value(),
                                   self.x0_spin.value(), self.xk_spin.value(),
                                   self.dx_spin.value())

        self.plot_area.setLayout(self.plot_lay)
        self.average_checked()

        # ---------------------------- connections----------------------------------------
        self.noise_slider.valueChanged.connect(self.show_noise)
        self.noise_slider.valueChanged.connect(self.plot_lay.change_noise_lvl)
        self.average_radio.toggled.connect(self.average_checked)
        self.exp_radio.toggled.connect(self.exp_checked)
        self.gausse_radio.toggled.connect(self.gausse_checked)
        self.median_radio.toggled.connect(self.median_checked)

        self.a1_spin.valueChanged.connect(self.change_params)
        self.a2_spin.valueChanged.connect(self.change_params)
        self.a3_spin.valueChanged.connect(self.change_params)
        self.b1_spin.valueChanged.connect(self.change_params)
        self.b2_spin.valueChanged.connect(self.change_params)
        self.b3_spin.valueChanged.connect(self.change_params)
        self.x0_spin.valueChanged.connect(self.change_params)
        self.xk_spin.valueChanged.connect(self.change_params)
        self.dx_spin.valueChanged.connect(self.change_params)
        self.scale_inc_button.clicked.connect(self.plot_lay.y_scale_inc)
        self.scale_dec_button.clicked.connect(self.plot_lay.y_scale_dec)

        # -------------------------------------------------------------------------------

    def show_noise(self, value: int):
        self.noise_line.setText(f'{value / 100} ')

    def average_checked(self):
        self.plot_lay.change_strategy("Moving Average")

    def exp_checked(self):
        self.plot_lay.change_strategy("Exponential")

    def gausse_checked(self):
        self.plot_lay.change_strategy("Gaussian")

    def median_checked(self):
        self.plot_lay.change_strategy("Median")

    def change_params(self):
        self.plot_lay.change_params({'a1': self.a1_spin.value(), 'a2': self.a2_spin.value(),
                                     'a3': self.a3_spin.value(), 'b1': self.b1_spin.value(),
                                     'b2': self.b2_spin.value(), 'b3': self.b3_spin.value(),
                                     'x0': self.x0_spin.value(), 'xk': self.xk_spin.value(),
                                     'dx': self.dx_spin.value()})
```

utils.py:

```python
from abc import ABC, abstractmethod
from scipy.signal import medfilt
from scipy.ndimage import gaussian_filter
import pandas as pd


class SmoothStrategy(ABC):
    @abstractmethod
    def get_y(self, y_noisy):
        pass


class Exponential(SmoothStrategy):
    def get_y(self, y_noisy):
        return pd.Series(y_noisy).ewm(span=5).mean().values


class Gaussian(SmoothStrategy):

    def get_y(self, y_noisy):
        return gaussian_filter(y_noisy, sigma=2)


class Median(SmoothStrategy):

    def get_y(self, y_noisy):
        return medfilt(y_noisy, kernel_size=5)


class MovingAverage(SmoothStrategy):
    def get_y(self, y_noisy):
        window_size = 5
        return pd.Series(y_noisy).rolling(window=window_size).mean().values
```

plot.py:

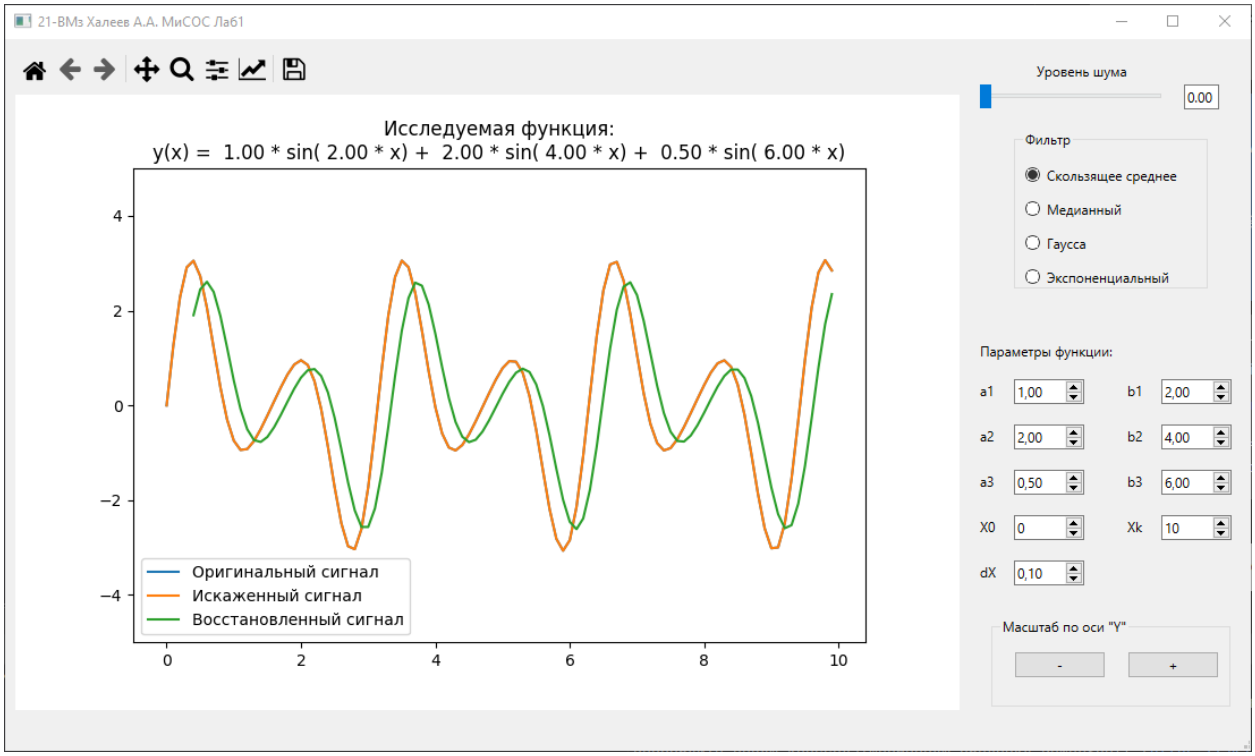
```python
import numpy as np

import matplotlib
from PySide6.QtCore import Signal
from PySide6.QtWidgets import QVBoxLayout, QWidget
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg, NavigationToolbar2QT as NavigationToolbar
from matplotlib.figure import Figure

from source.utils import SmoothStrategy, Exponential, Gaussian, Median, MovingAverage

matplotlib.use('Qt5Agg')


class MplCanvas(FigureCanvasQTAgg):

    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        super(MplCanvas, self).__init__(fig)


class PlotLayout(QVBoxLayout, QWidget):
    noise_lvl_changed = Signal()
    strategy_changed = Signal()
    params_changed = Signal()

    def __init__(self, /, a1, a2, a3, b1, b2, b3, x0, xk, dx, *args, **kwargs, ):
        super(PlotLayout, self).__init__(*args, **kwargs)
        # ---------------------------- connections-----------------------------------------
        self.noise_lvl_changed.connect(self.draw_plot)
        self.strategy_changed.connect(self.draw_plot)
        self.params_changed.connect(self.draw_plot)
        # ---------------------------------------------------------------------------------

        self.y_scale = 0
        self.sm_strategy = MovingAverage()
        self.noise_lvl = 0

        # Передаем параметры в словарь
        self.params = {'a1': a1, 'a2': a2, 'a3': a3, 'b1': b1, 'b2': b2, 'b3': b3, 'x0': x0, 'xk': xk, 'dx': dx}

        # Создание объекта FigureCanvas для matplotlib
        self.sc = MplCanvas(self, width=5, height=4, dpi=100)

        # Создание панели инструментов
        toolbar = NavigationToolbar(self.sc, self)

        self.addWidget(toolbar)
        self.addWidget(self.sc)
        self.draw_plot()

    def change_noise_lvl(self, lvl):
        self.noise_lvl = lvl / 100
        self.noise_lvl_changed.emit()

    def draw_plot(self):
        # Сначала очистим предыдущий график
        self.sc.axes.clear()

        x = np.arange(self.params['x0'], self.params['xk'], self.params['dx'])
        y = self.params['a1'] * np.sin(self.params['b1'] * x) + self.params['a2'] * np.sin(
            self.params['b2'] * x) + self.params['a3'] * np.sin(self.params['b3'] * x)

        # Генерация шума и добавление его к y
        noise = np.random.normal(0, self.noise_lvl, len(y))
        y_noisy = y + noise
        y_smooth = self.sm_strategy.get_y(y_noisy)

        self.sc.axes.plot(x, y, label='Оригинальный сигнал')
        self.sc.axes.plot(x, y_noisy, label='Искаженный сигнал')
        self.sc.axes.plot(x, y_smooth, label='Восстановленный сигнал')

        # Восстановление легенды и других элементов графика, если они есть
        self.sc.axes.legend(loc='lower left')
        # Зафиксировать ось Y
        self.sc.axes.set_ylim([-5 - self.y_scale, 5 + self.y_scale])
        # Задать название графика
        self.sc.axes.set_title('Исследуемая функция:\n'
                               f"y(x) = {self.params['a1']: .2f} * sin({self.params['b1']: .2f} * x) + "
                               f"{self.params['a2']: .2f} * sin({self.params['b2']: .2f} * x) + "
                               f"{self.params['a3']: .2f} * sin({self.params['b3']: .2f} * x)")
        # Перерисовка графика
        self.sc.draw()
```

```
    def change_strategy(self, checkbox: str) -> None:
        match checkbox:
            case "Exponential":
                self.sm_strategy = Exponential()
            case "Gaussian":
                self.sm_strategy = Gaussian()
            case "Median":
                self.sm_strategy = Median()
            case "Moving Average":
                self.sm_strategy = MovingAverage()
        self.strategy_changed.emit()

    def change_params(self, new_params: dict) -> None:
        self.params = new_params
        self.params_changed.emit()

    def y_scale_inc(self):
        self.y_scale += 1
        self.params_changed.emit()

    def y_scale_dec(self):
        if self.y_scale == 0:
            return
        self.y_scale -= 1
        self.params_changed.emit()
```

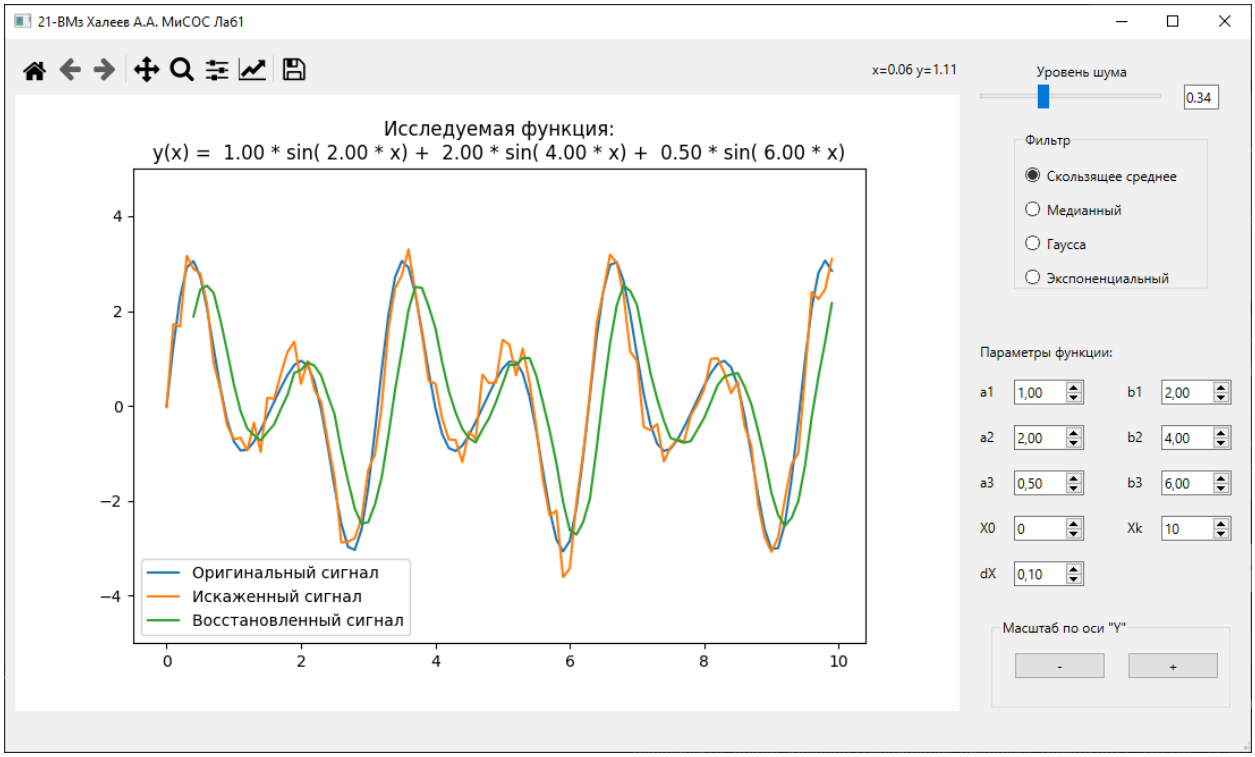
Приложение 2

Результаты тестирования

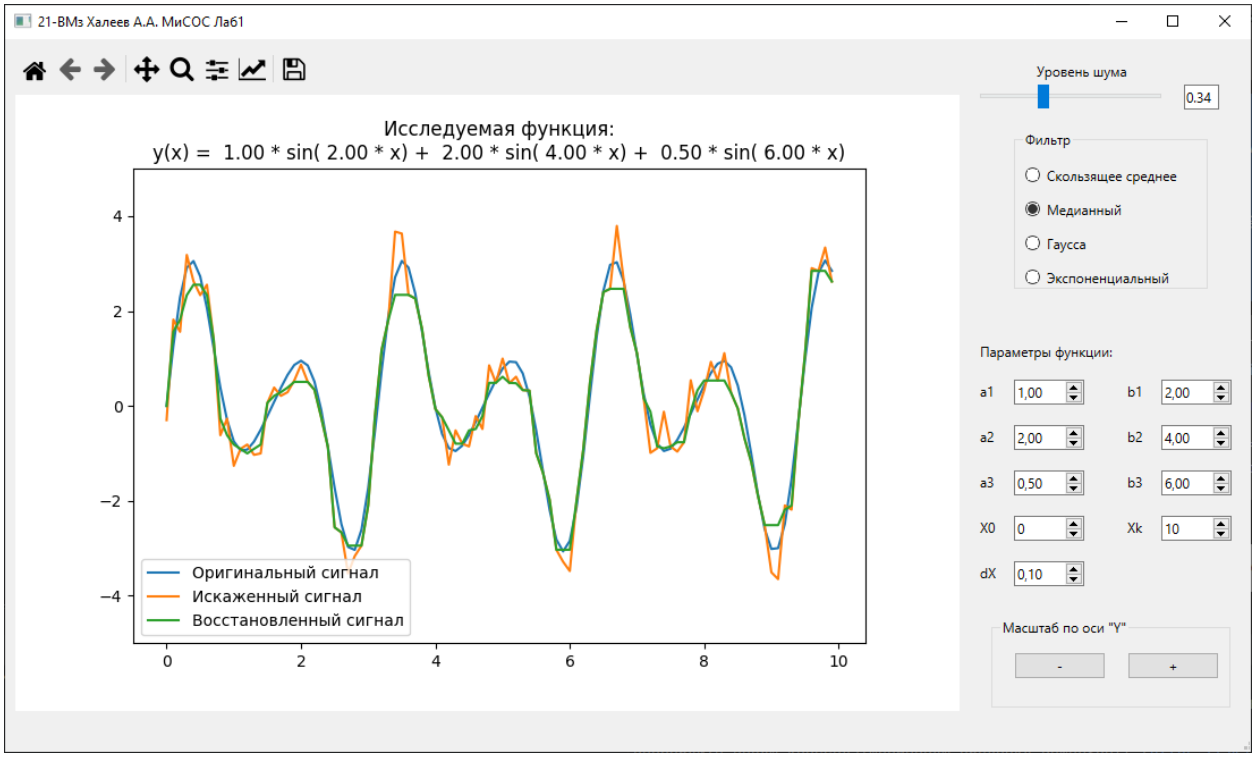
Окно программы после запуска:



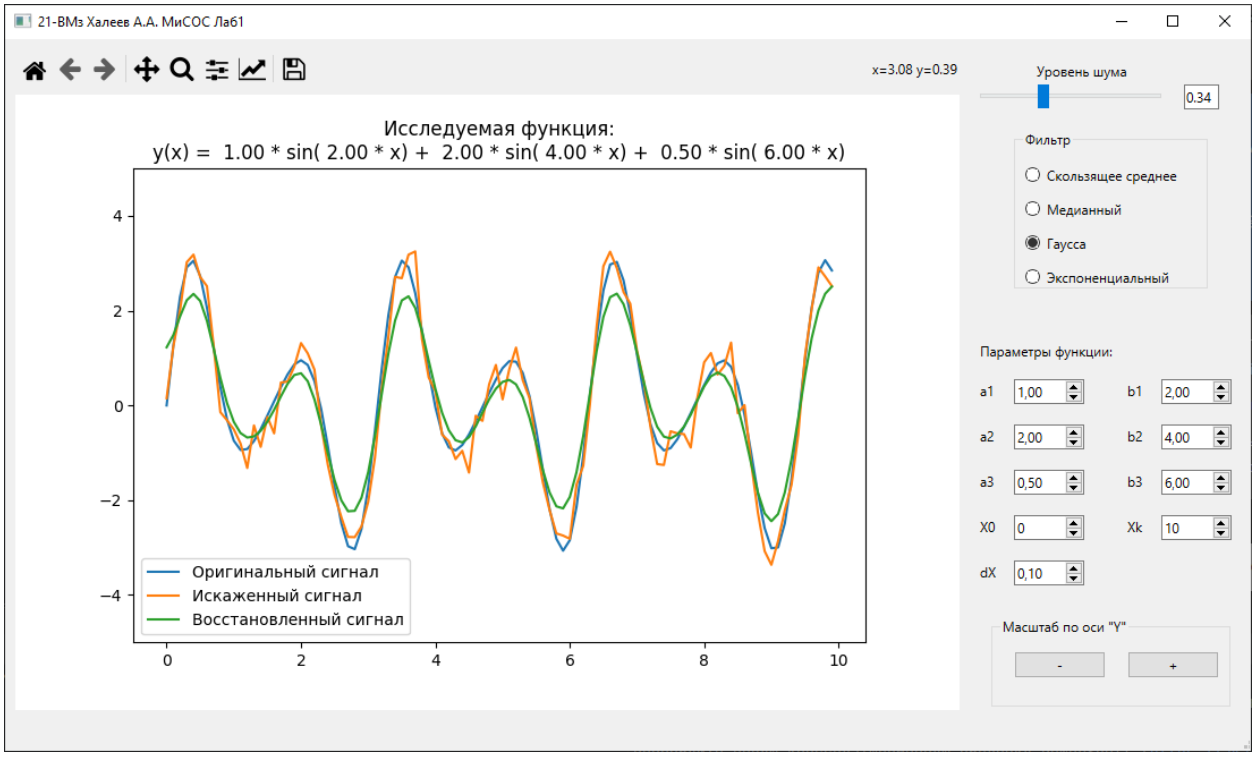
С добавлением шума:



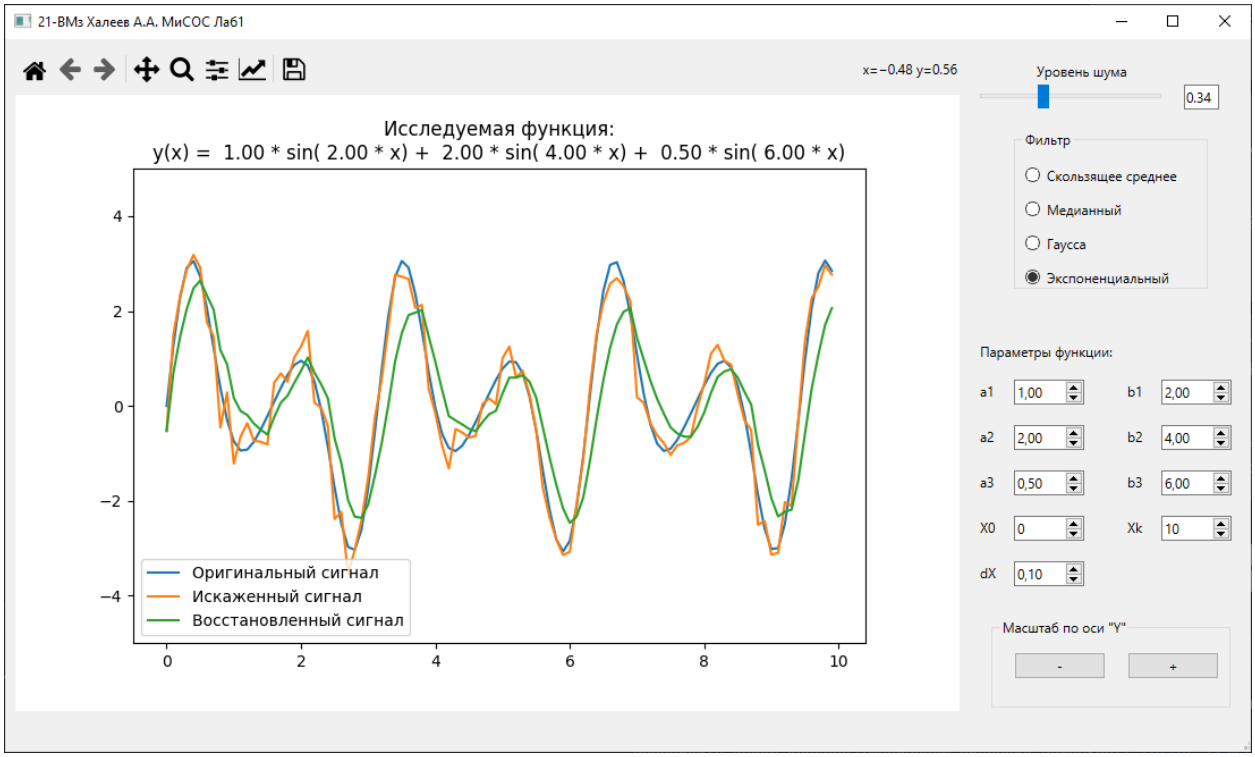
Медианное сглаживание:



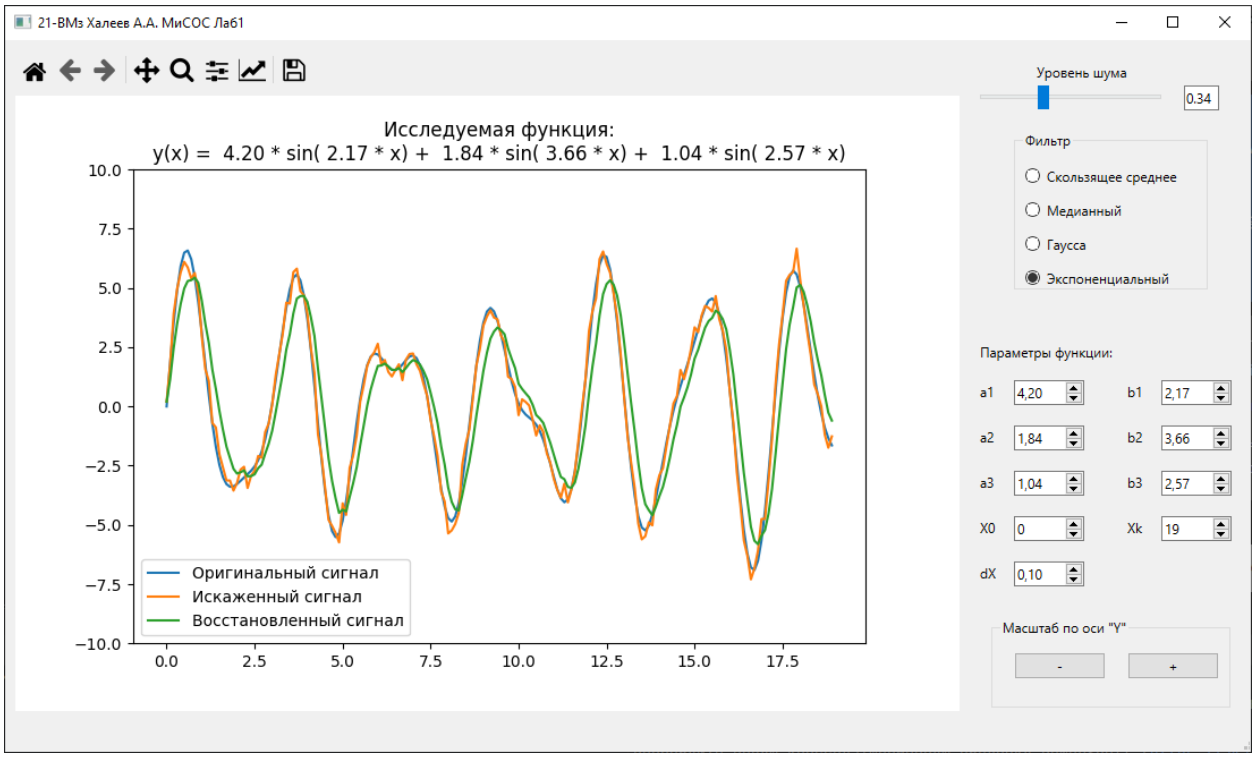
Гауссово сглаживание:



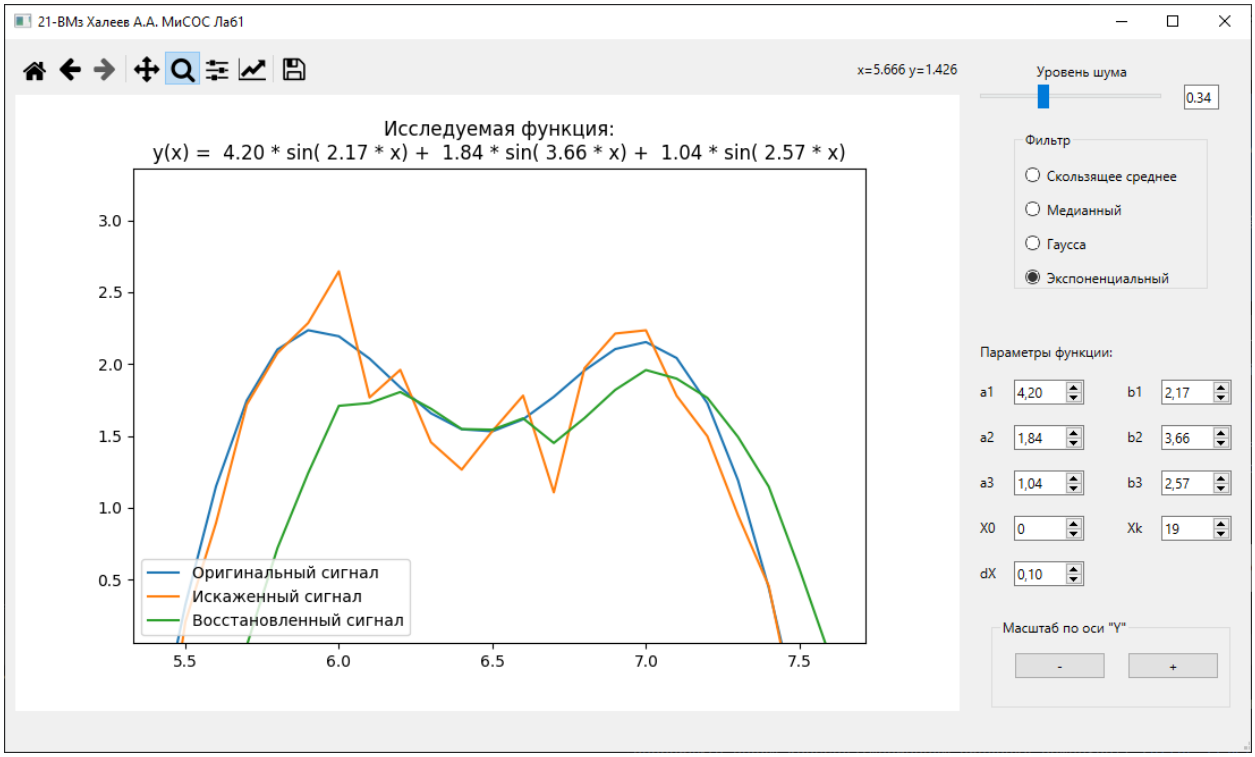
Экспоненциальное сглаживание:



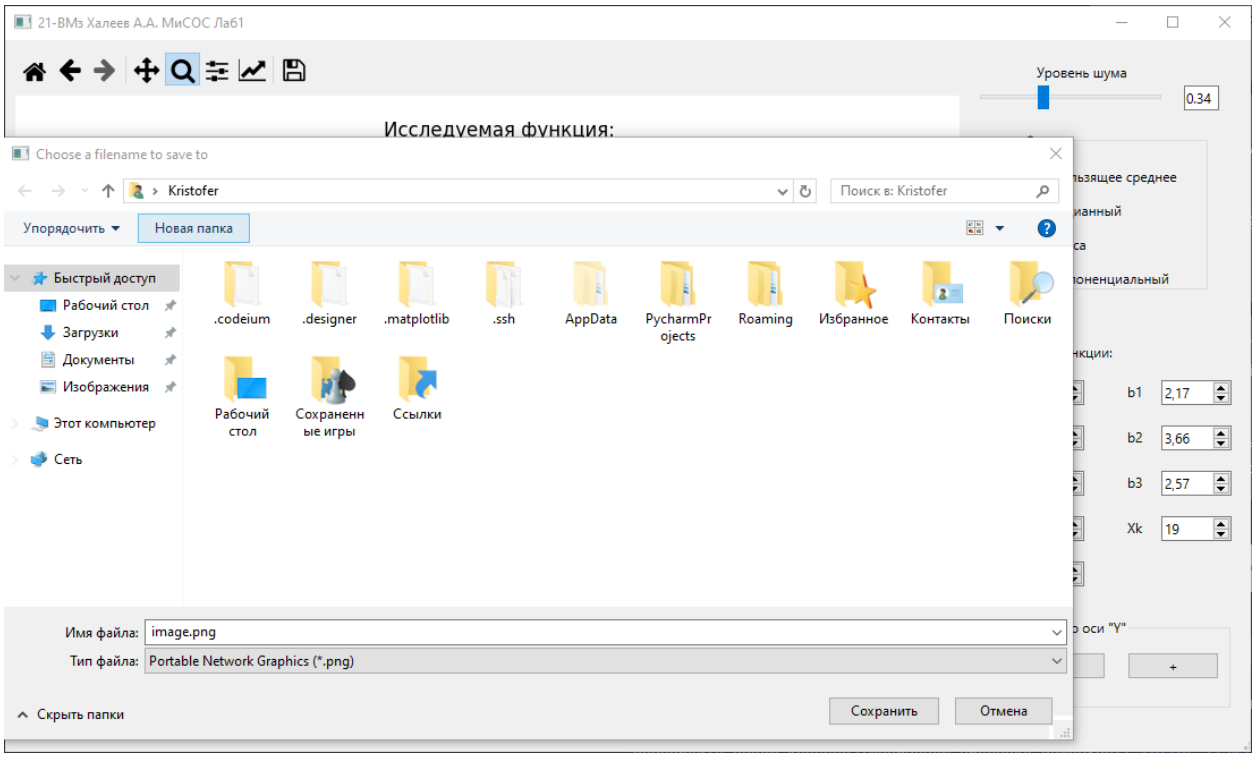
Изменение параметров функции:



Фокус на область графика с помощью панели инструментов:



Окно сохранения графика:



Вывод:

Разработанное приложение работает без ошибок, полностью выполняет задуманную функциональность. С помощью программы можно визуализировать график заданной функции с произвольными параметрами, а также имитировать искаженный сигнал и анализировать эффективность применения сглаживания разных типов.

Учитывая, что сигнал задан комбинацией синусоидальных функций, эффективность методов сглаживания, представленных в программе, будет следующей:

1. **Экспоненциальное сглаживание:**
    - Может быть менее эффективно для такого периодического сигнала, так как он предполагает больший вес для последних данных, что может привести к уменьшению амплитуды колебаний.
    - Хорошо подходит, если требуется быстро реагировать на изменения, но может быть менее предпочтительным для анализа чисто гармонических колебаний.
2. **Гауссово сглаживание:**
    - Будет сглаживать пики и впадины сигнала, что может привести к потере информации о точных амплитудах и периодах синусоид.
    - Может быть полезным для уменьшения случайного шума, сохраняя при этом общую форму синусоидальных компонентов сигнала.
3. **Медианное сглаживание:**
    - Подходит для удаления выбросов или редких шумов, которые не соответствуют общему периодическому характеру сигнала.
    - Может использоваться для сохранения резких переходов в сигнале, что важно, если эти переходы являются частью сигнала.
4. **Скользящее среднее:**
    - Подходит для сглаживания случайного шума, но при этом может сгладить и полезные детали синусоидального сигнала.
    - Размер окна скользящего среднего должен быть тщательно подобран, чтобы не утратить информацию о периодах и амплитудах синусоидальных компонентов.

Для сигнала, состоящего из нескольких синусоидальных функций, целью сглаживания обычно является уменьшение случайного шума, при этом сохраняя амплитуду и частотные характеристики основного сигнала. В этом случае, наиболее подходящим методом сглаживания будет тот, который минимизирует искажение основных гармонических компонентов сигнала. Таким фильтром в данном случае можно считать Гауссово сглаживание.