МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Лабораторная работа №1

(наименование темы проекта или работы)

# ОТЧЕТ

по лабораторной работе №1

Вариант №1

по дисциплине

Методы и средства обработки сигналов

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

_____ Авербух М.Л.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

_____ Куприн А.Д.

(подпись) (фамилия, и.,о.)

Группа: 22-ВМз

(шифр группы)

Работа защищена «___» _____

С оценкой _____

Нижний Новгород, 2024

# Задача

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

Пользователем задаются: $a_1$, $b_1$, $a_2$, $b_2$, $a_3$, $b_3$, $x_0$ (начальное значение), $x_к$ (конечное значение), $\Delta x$ (шаг). Расчет $y(x)$ по заданным значениям $a_1$, $b_1$, $a_2$, $b_2$, $a_3$, $b_3$, $x_0$ (начальное значение), $x_к$ (конечное значение), $\Delta x$ (шаг). Отображение векторов x и y (в виде таблицы). Построение графика $y(x)$ по указанным векторам.

# Код программы

```
import sys

import numpy as np
from loguru import logger
from pydantic import BaseModel
from numpy.typing import NDArray
from PyQt6.QtCore import QObject
from PyQt6.QtWidgets import (
    QApplication,
    QHeaderView,
    QWidget,
    QHBoxLayout,
    QLineEdit,
    QPushButton,
    QFormLayout,
    QVBoxLayout,
    QLabel,
    QTableWidget,
    QTableWidgetItem,
    QMessageBox,
    QFrame
)
from matplotlib.figure import Figure
from matplotlib.backends.backend_qtagg import FigureCanvasQTAgg as FigureCanvas



class FunctionCharacteristics(BaseModel):
    """
        Класс для хранения и передачи между функциями характеристик полученных из
форм инпута
    """
    a1: float
    b1: float
    a2: float
    b2: float
    a3: float
    b3: float
    x0: float
    xk: float
    delta_x: float


class LiveGraph(FigureCanvas):
    def __init__(self, parent: QObject | None = None, width: int = 5, height: int = 4, dpi: int = 100):
        # создаем область где будет помещен наш график
        fig = Figure(figsize=(width, height), dpi=dpi)

        # рисуем оси
        self.axes = fig.add_subplot(111)
        super().__init__(fig)
        self.xdata = np.array([])
        self.ydata = np.array([])
```

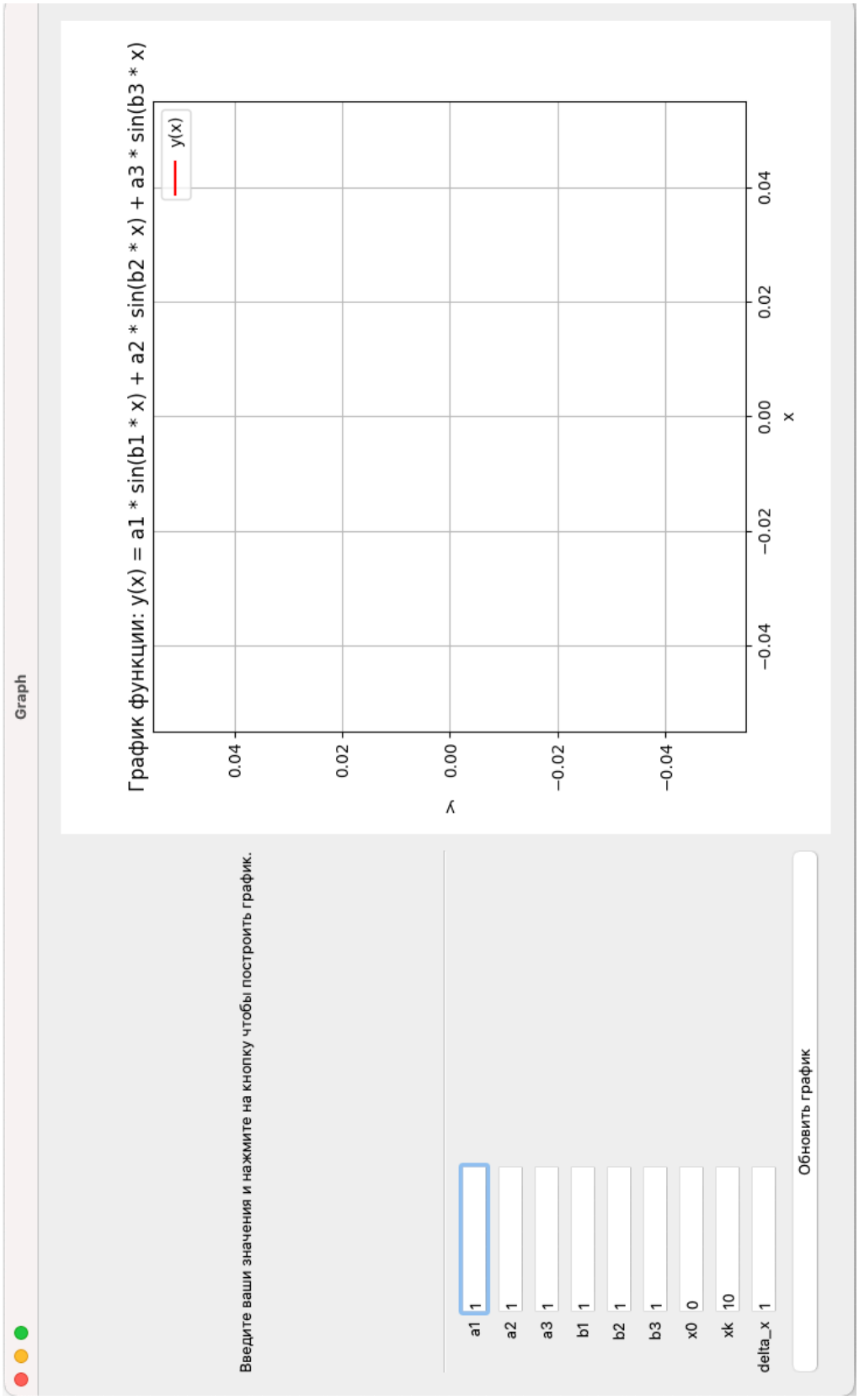
```
        # отображаем начальные данные по осям
        self.line, = self.axes.plot(self.xdata, self.ydata, "r-", label="y(x)")
        self.axes.grid(True)
        self.axes.set_xlabel("x")
        self.axes.set_ylabel("y")
        self.axes.set_title("График функции: y(x) = a1 * sin(b1 * x) + a2 * sin(b2 * x) + a3 * sin(b3 *
x)")
        self.axes.legend()

    def _calculate_y(self, x_vals: NDArray, chr: FunctionCharacteristics):
        return chr.a1 * np.sin(chr.b1 * x_vals) + chr.a2 * np.sin(chr.b2 * x_vals) + chr.a3 *
np.sin(chr.b3 * x_vals)

    # данная функция обновляет график
    def update_graph(self, characteristics: FunctionCharacteristics):
        """
            Функция обновляет график
        """
        self.xdata = np.arange(characteristics.x0, characteristics.xk, characteristics.delta_x)
        self.ydata = self._calculate_y(self.xdata, characteristics)
        self.line.set_data(self.xdata, self.ydata)
        self.axes.relim()
        self.axes.autoscale_view()
        self.draw()
        return (self.xdata, self.ydata)


class CustomGraphApp(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def _create_edit_forms(self):
        # создаем словарь с полями форм инпута и делаем аттрибутом класса
        self.edit_forms =  {
            "a1": QLineEdit("1"),
            "a2": QLineEdit("1"),
            "a3": QLineEdit("1"),
            "b1": QLineEdit("1"),
            "b2": QLineEdit("1"),
            "b3": QLineEdit("1"),
            "x0": QLineEdit("0"),
            "xk": QLineEdit("10"),
            "delta_x": QLineEdit("1"),
        }

    def init_ui(self):
        """
            Функция инициализирует пользовательский интерфейс
        """
        self.setWindowTitle("Graph")

        layout = QHBoxLayout()

        # создаем и настраиваем область для графика
        graph_layout = QHBoxLayout()
        self.graph = LiveGraph(self, width=10, height=10, dpi=100)
        graph_layout.addWidget(self.graph)

        # создаем и настраиваем область для таблицы значений
```

```
        self.info_layout = QVBoxLayout()
        self.info_label = QLabel("Введите ваши значения и нажмите на кнопку чтобы
построить график.")
        self.info_layout.addWidget(self.info_label)

        # разделяем чертой элементы интерфейса
        line = QFrame()
        line.setFrameShape(QFrame.Shape.HLine)
        line.setFrameShadow(QFrame.Shadow.Sunken)
        self.info_layout.addWidget(line)

        # создаем и настраиваем область для форм инпута и кнопки
        control_layout = QFormLayout()
        self._create_edit_forms()
        for key, edit_input in self.edit_forms.items():
            control_layout.addRow(key, edit_input)
        self.update_button = QPushButton("Обновить график")
        self.update_button.clicked.connect(self.update_graph_values)
        control_layout.addRow(self.update_button)

        # добавляем созданные области в главный блок
        self.info_layout.addLayout(control_layout)
        layout.addLayout(self.info_layout)
        layout.addLayout(graph_layout)
        self.setLayout(layout)
        self.resize(1900, 1000)

    def update_graph_values(self):
        """
            Функция обновляет график по новым заданным значениям
        """
        try:
            data = {k: float(v.text()) for k, v in self.edit_forms.items()}
            characteristics = FunctionCharacteristics(**data)
            x_data, y_data = self.graph.update_graph(characteristics)
            self.display_results(x_data, y_data)
        except ValueError as err:
            logger.error(f"Все поля должны быть заполнены числовыми значениями: {err}")
            self.show_error_message(f"Все поля должны быть заполнены числовыми
значениями: {err}")

    def display_results(self, x_vals: NDArray, y_vals: NDArray):
        """
            Функция обновляет данные в таблице рассчетов x, y
        """
        #обнуляем предыдущие результаты
        self.info_label.setText("Рассчитанные значения:")
        if hasattr(self, "current_table") and self.current_table:
            self.info_layout.removeWidget(self.current_table)
            self.current_table.deleteLater()
            self.current_table = None

        #создаем и заполняем таблицу с рассчитаныими значениями
        table = QTableWidget()
        table.setRowCount(len(x_vals))
        table.setColumnCount(2)
        table.setHorizontalHeaderLabels(["x", "y"])
        for i, (x, y) in enumerate(zip(x_vals, y_vals)):
            table.setItem(i, 0, QTableWidgetItem(f"{x:.2f}"))
            table.setItem(i, 1, QTableWidgetItem(f"{y:.2f}"))
```

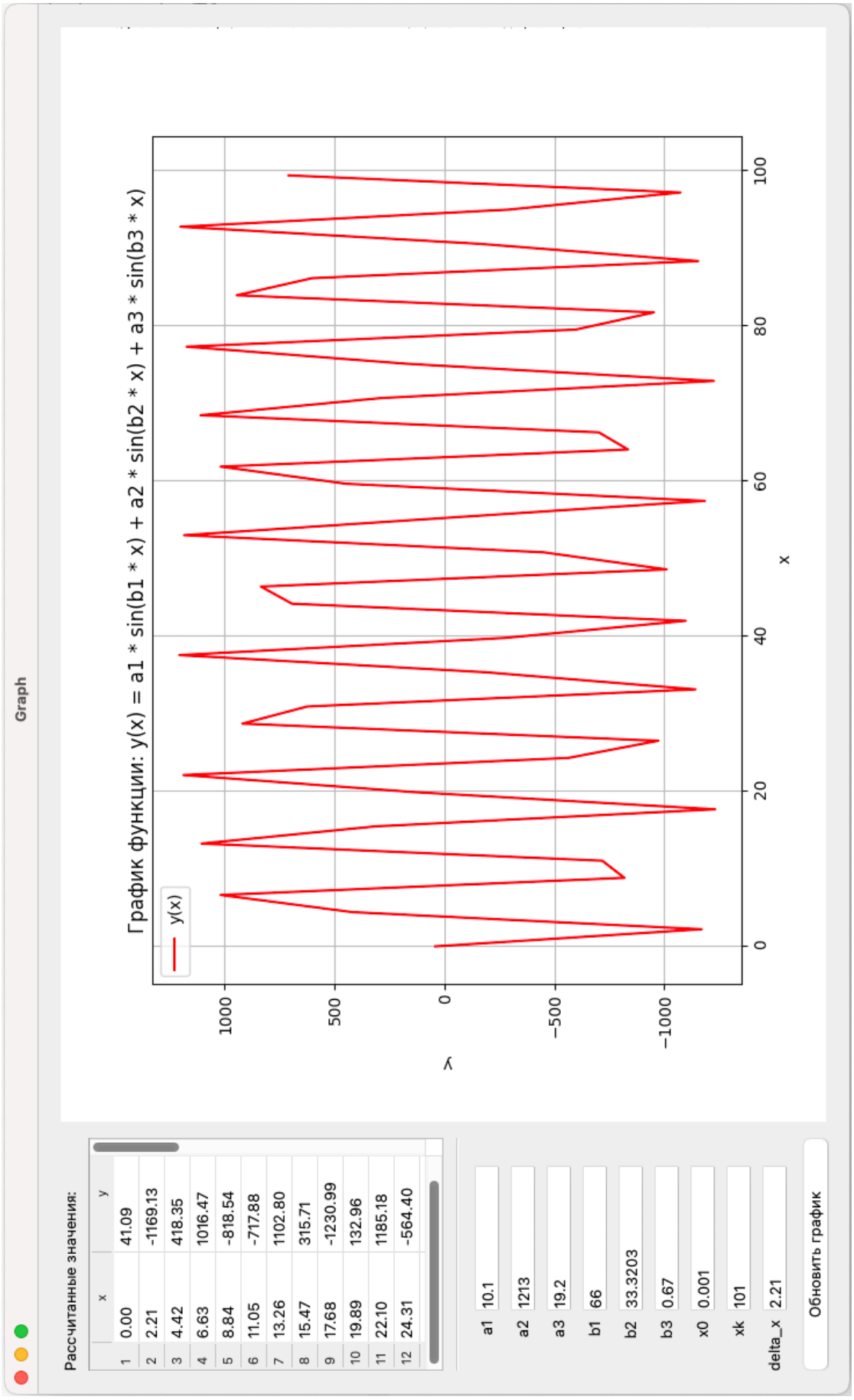
```
        #немного косметики, растягиваем таблицу на весь блок лэйаута
        h_header = table.horizontalHeader()
        if h_header:
            h_header.setStretchLastSection(True)
            h_header.setSectionResizeMode(0, QHeaderView.ResizeMode.Stretch)
            h_header.setSectionResizeMode(1, QHeaderView.ResizeMode.Stretch)
        v_header = table.verticalHeader()
        if v_header:
            v_header.setSectionResizeMode(QHeaderView.ResizeMode.Stretch)

        #обновляем данные в таблице
        self.info_layout.update()
        self.info_layout.insertWidget(1, table)
        self.current_table = table

    def show_error_message(self, message: str):
        """
            Функция выводит в отдельном окне сообщение о случившейся ошибке
        """
        error_dialog = QMessageBox()
        error_dialog.setIcon(QMessageBox.Icon.Critical)
        error_dialog.setWindowTitle("Input Error")
        error_dialog.setText("An error occurred while processing the input.")
        error_dialog.setInformativeText(message)
        error_dialog.setStandardButtons(QMessageBox.StandardButton.Ok)
        error_dialog.exec()


def main():
    logger.info("start app")
    app = QApplication(sys.argv)
    main_window = CustomGraphApp()
    main_window.show()
    sys.exit(app.exec())


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

# Ход работы программы

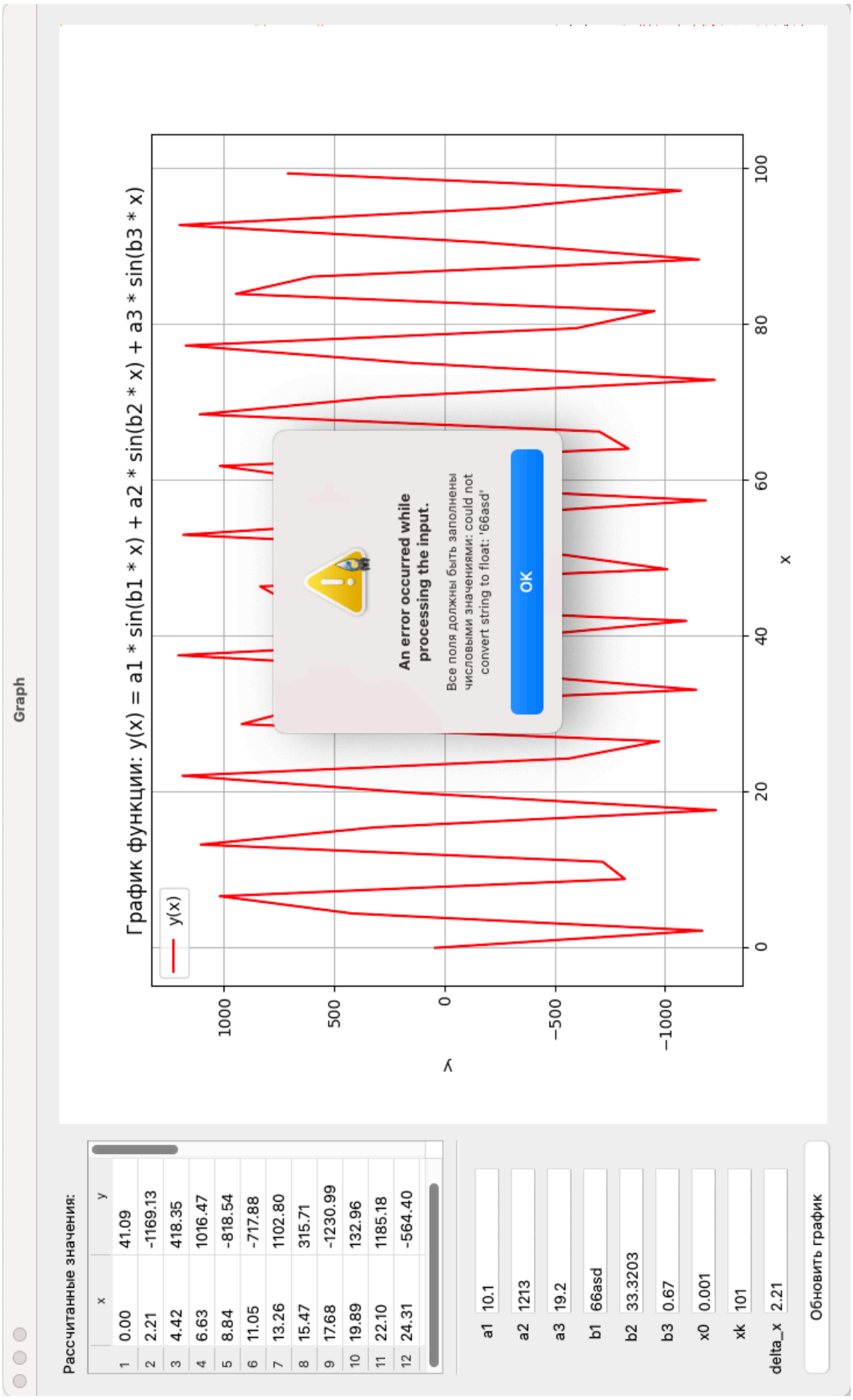
## 1. Старт работы



# 2. После введения значения



# 3. Ошибка при неправильно введенном значении



# Выводы

В ходе выполнения данной задачи я изучил библиотеки numpy, matplotlib и PyQt6, и получил множество новых знаний, которые пригодятся в дальнейших проектах. Одним из интересных открытий стала функция `arange` и класс `Array` из модуля numpy. Изначально я планировал использовать генератор списка для создания массивов с точками x и y, используя в генераторе цикл `for` с генератором `range`. Однако оказалось, что `range` не поддерживает работу с типом `float`, в то время как `arange` работает со всеми числовыми типами, что значительно упрощает задачу.
При проектировании приложения я решил разделить логику на три класса: два класса для интерфейса и один для валидации данных и обмена между функциями. В итоге получилось приложение, в котором формы для ввода атрибутов, таблица рассчитанных значений и график функции располагаются в интерфейсе, что позволяет не закрывая программу рассчитывать различные графики, изменяя атрибуты функции в полях ввода.