УДК 004

ПОСТРОЕНИЕ ГРАФИКОВ В НАСТОЛЬНОМ ПРИЛОЖЕНИИ: ИНТЕГРАЦИЯ MATPLOTLIB

Сиводедова М.В., студентка гр. ПИм-241, I курс
Черепанов П.В., студент гр. ПИм-241, I курс
Научный руководитель: Ананенко Е.В., старший преподаватель
Кузбасский государственный технический университет
имени Т.Ф. Горбачева,
г. Кемерово

Визуализация данных играет ключевую роль в анализе и интерпретации информации в самых разных областях науки и техники. Например, благодаря наглядной визуализации специалист сможет принимать грамотные управленческие решения, а маркшейдер на горнодобывающем предприятии – оценивать напряженно-деформируемое состояние бортов карьеров и откосов отвалов. Одним из наиболее популярных инструментов для построения графиков является библиотека Matplotlib, широко используемая в среде Python. Интеграция Matplotlib в настольное приложение позволяет пользователям визуализировать данные в удобном формате, взаимодействовать с графиками и анализировать результаты непосредственно в программном интерфейсе.

Для реализации настольного приложения с поддержкой построения графиков можно использовать язык программирования Python и библиотеку Matplotlib. В качестве графического интерфейса часто применяют библиотеку PySide, которая позволяет быстро запрограммировать пользовательский интерфейс любой сложности.

PySide – это официальный набор привязок для работы с Qt в Python.[1][2] Он позволяет создавать настольные приложения с графическим интерфейсом. PySide предоставляет возможности для: разработки сложных пользовательских интерфейсов, интеграции с библиотеками визуализации данных, поддержки событийно-ориентированного программирования.

Matplotlib представляет собой библиотеку для визуализации данных в Python. [3] Она поддерживает различные виды графиков, включая линейные, гистограммы и диаграммы рассеяния. Основными преимуществами Matplotlib являются: гибкость в настройке графиков, поддержка интерактивного режима, совместимость с различными GUI-фреймворками.

В качестве примера интеграции Matplotlib с PySide можно рассмотреть следующий код (Рис. 1), там определяется класс Graph, который наследуется от QGraphicsView и предназначен для отображения графиков в пользовательском интерфейсе, созданном с использованием Matplotlib и PySide. Для встраивания Matplotlib в PySide используется FigureCanvasQTAgg, который преобразует график в виджет Qt.

```
class Graph(QGraphicsView):
    def __init__(self, main_window):
        super().__init__()
        self.main_window = main_window
        self.ui = main_window.ui       self.ui.settingsForGraphBut-
ton.clicked.connect(self.show_settings_window)
        self.figure, self.ax = plt.subplots(figsize=(10, 5), dpi=90)
        self.canvas = FigureCanvasQTAgg(self.figure)
        self.toolbar = NavigationToolbar(self.canvas,self.ui.graph)
        layout = QVBoxLayout()
        layout.addWidget(self.canvas)
        layout.addWidget(self.toolbar)
        self.ui.graph.setLayout(layout)
    def drawMultiple(self, data_dict):
        self.ax.clear()
        self.canvas.flush_events()
        colors = ['b', 'r', 'g', 'y', 'c', 'm', 'k']
        color = 0
        all_dates = []
        for label, (x, y) in data_dict.items():
            x_dates = [datetime.strptime(date, '%d.%m.%Y') for date in [i[1]
for i in x.items()]]
            all_dates.extend(x_dates)
            all_dates = set(all_dates)
            all_dates = sorted(list(all_dates))
            if color == len(colors) - 1:
                color = 0
            self.ax.plot(x_dates, y, marker='.', markersize=7, label=label,
color=colors[color])
            color += 1
        self.ax.set_xticks(all_dates)
        settingsGraph = SettingsGraph()
        coalpit_id = self.ui.coalpitsFilterComboBoxForGraph.currentData()
        self.ax.axhspan(settingsGraph.selectGY1(coalpit_id), set-
tingsGraph.selectGY2(coalpit_id), facecolor='green',
                        alpha=0.3)
        self.ax.axhspan(settingsGraph.selectYY1(coalpit_id), set-
tingsGraph.selectYY2(coalpit_id),
                        facecolor='yellow',
                        alpha=0.3)
        self.ax.axhspan(settingsGraph.selectRY1(coalpit_id), set-
tingsGraph.selectRY2(coalpit_id), facecolor='red',
                        alpha=0.3)
        plt.setp(self.ax.xaxis.get_majorticklabels(), rotation=45,
ha='right')      self.ax.xaxis.set_major_formatter(mdates.DateFormat-
ter("%d.%m.%Y"))
        self.figure.autofmt_xdate()
        self.ax.set_xlabel('Дата')
        self.ax.set_ylabel('Скорость смещения полного вектора (мм/ст)')
        self.ax.set_title('График скорости смещения полного вектора')
        self.ax.grid(True)
        self.ax.legend()
        self.figure.tight_layout()
        self.canvas.draw()
        self.canvas.update()
        self.ui.graph.update()
```

Рисунок – 1 Пример класса для отображения графиков в пользовательском интерфейсе

Конструктор (__init__):

- Инициализирует родительский класс QGraphicsView.
- Сохраняет ссылку на главное окно и его интерфейс (ui).
- Привязывает кнопку настроек графика к методу show_settings_window.
- Создает графическую область (figure, ax) для построения графика с фиксированными размерами.
- Создает объект FigureCanvasQTAgg, который позволяет встроить график Matplotlib в интерфейс PyQt.
- Добавляет панель инструментов NavigationToolbar для взаимодействия с графиком.
- Выполняет настройку компоновки элементов пользовательского интерфейса, используя менеджер компоновки QVBoxLayout. В данной компоновке располагаются два ключевых элемента: холст для отображения графика и панель инструментов, обеспечивающая удобный доступ к функциям управления визуализацией. После добавления этих элементов компоновка применяется к виджету graph, который используется в пользовательском интерфейсе для представления данных в графическом формате.
- Виджет graph представляет собой объект типа QGraphicsView, который необходимо создать заранее, используя инструменты визуального проектирования в среде разработки Qt Designer. Это позволяет значительно упростить процесс формирования интерфейса, обеспечивая точное размещение элементов и их корректное взаимодействие в рамках программы. Благодаря заранее настроенному виджету удаётся добиться стабильной работы приложения и удобства при дальнейшем развитии его функционала.

Метод drawMultiple(data_dict):

- Очищает текущий график и сбрасывает события рендеринга.
- Использует список цветов для отображения различных серий данных.
- Перебирает переданный словарь data_dict, содержащий метки и соответствующие им координаты.
- Преобразует даты из строкового формата в объекты datetime и сортирует их.
- Строит линии графика, используя разные цвета и маркеры.
- Добавляет цветовые зоны (axhspan) для выделения диапазонов значений, полученных из объекта SettingsGraph, который использует идентификатор, полученный из выпадающего списка.

На рисунке 2 приведен фрагмент графика скорости смещения полного вектора, полученный по результатам маркшейдерского мониторинга и созданный с помощью интеграции Matplotlib с PySide что позволяет разрабатывать полезные прикладные настольные приложения с возможностью визуализации данных. Использование Matplotlib облегчает процесс построения графиков, а инструменты PySide помогают создать удобный пользовательский интерфейс.

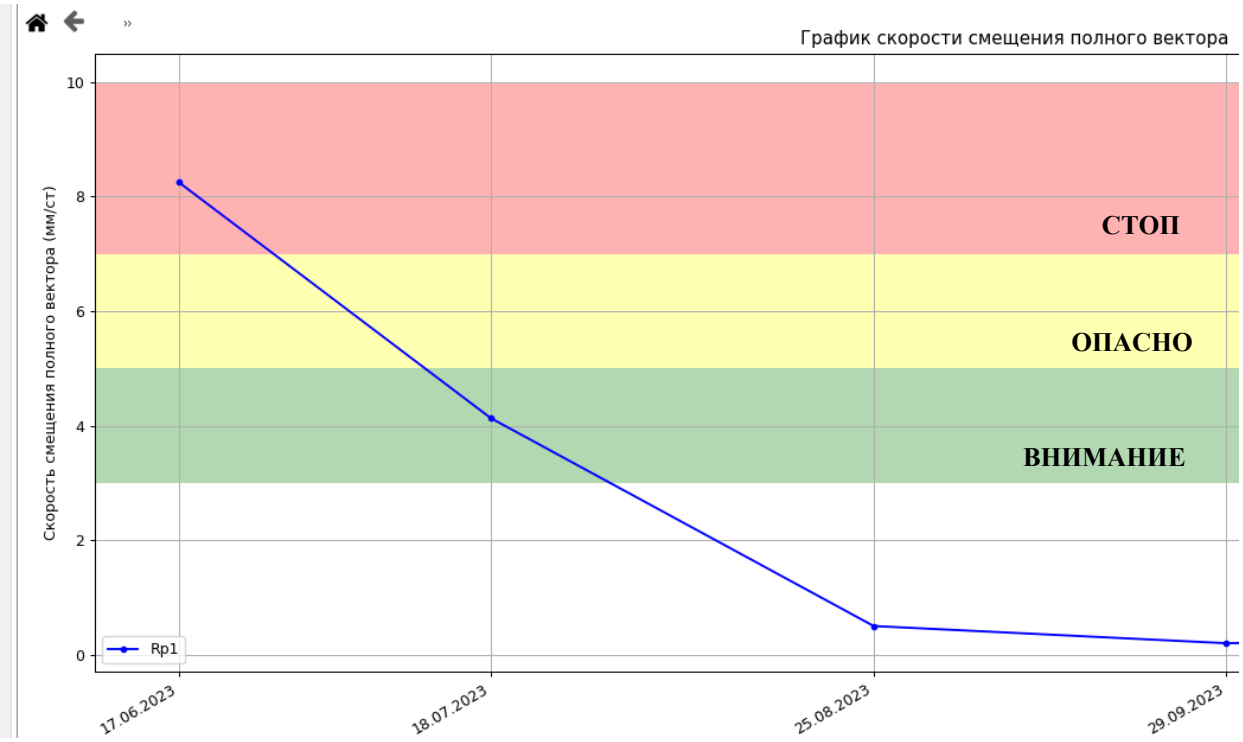

Рисунок – 2 Фрагмент графика скорости смещения полного вектора, построенного по результатам маркшейдерского мониторинга и интегрированного в настольное приложение

Таким образом, комбинация этих технологий делает возможным создание сложных аналитических инструментов в формате настольного приложения и внедрять их для обработки различных данных на горнодобывающих предприятиях.

Список литературы:

1. PySide : сайт. – URL: https://pypi.org/project/PySide6/ (дата обращения: 29.03.2025)
2. Qt : сайт. – URL: https://doc.qt.io/qtforpython-6/examples/index.html (дата обращения: 29.03.2025)
3. Matplotlib: сайт. – URL: https://matplotlib.org/ (дата обращения: 29.03.2025)