МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«Южный федеральный университет»

Институт математики, механики
и компьютерных наук им. И. И. Воровича

Кафедра информатики и вычислительного эксперимента

Кобзарь Дмитрий Валерьевич

ГРАФИЧЕСКАЯ БИБЛИОТЕКА ДЛЯ JUPYTER NOTEBOOK PASCALABC.NET

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению подготовки
02.03.02 – Фундаментальная информатика и информационные технологии

Научный руководитель –
доц., к. ф.-м. н. Михалкович Станислав Станиславович

Допущено к защите:
заведующий кафедрой _____ Михалкович С. С.

Ростов-на-Дону – 2022

Оглавление

1. Введение

Визуализация данных (далее ВД) – термин, обозначающий представление данных в виде, который обеспечивает наиболее эффективную работу человека по их изучению. Особенно часто ВД используется в областях, связанных с большими объёмами данных. Таким образом, ВД находит широкое применение как в научных и статистических исследованиях, так и в бизнес-анализе, новостных сводках и аналитических разборах [1].

В сфере IT особенно важна наглядность данных, с которыми работает специалист. Например, программисты, работающие в сфере Big Data и Data Science, в основном используют в своей работе язык Python в связке с библиотекой Matplotlib (далее mpl) для изучения полученных в вычислениях данных, так как это наиболее удобный способ визуализации среди решений с открытым кодом. Для ещё более наглядного отображения результатов работы конкретной программы используется фреймворк Jupyter Notebook, который позволяет совмещать код программы с её выводом в удобном интерфейсе “ноутбука”. Из примера выше становится ясно, что даже самые передовые сферы IT зависимы от инструментов ВД.

Если же говорить про обучение школьников и студентов младших курсов в сфере IT, то ими зачастую используется язык PascalABC.NET, как наиболее удобный язык для начинающих программистов. За ВД в данном случае отвечает модуль GraphWPF, на основе которого можно реализовать и более сложные модули, как например модуль Plotter, разрабатываемый мною в качестве курсовой работы на 3 курсе обучения[2].

Естественно, при разработке ядра PascalABC.NET для платформы Jupyter Notebook возникла потребность в адаптации графических модулей для работы в веб-приложении, а не в графическом окне Windows. Данной проблеме и посвящена дипломная работа.

2. Постановка задачи

Главная цель работы – реализовать графические модули ядра PascalABC.NET для Jupyter Notebook.

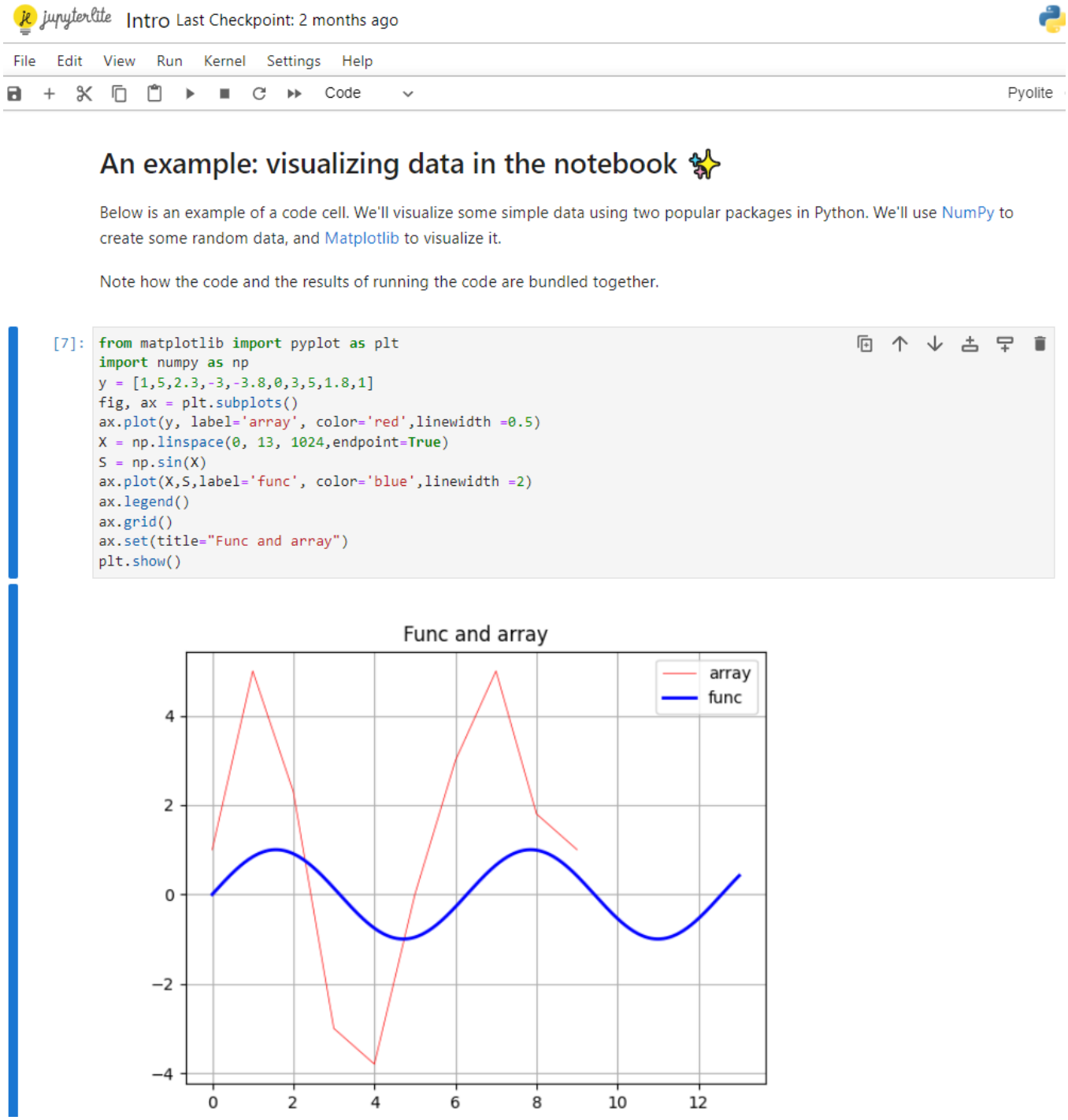
Для этого требуется решить следующие задачи:

- Разработать способ вывода графики из программы на PascalABC.NET в Jupyter Notebook
- Разработать аналог модулей GraphWPF и Plotter для вывода графики в ядре PascalABC.NET Jupyter Notebook
- Адаптировать графические модули ядра для работы под Mono
- Развернуть ядро PascalABC.NET Jupyter Notebook на сервере Jupyter Hub

3. Визуализация данных в Jupyter Notebook

Jupyter Notebook — это среда разработки, где сразу можно видеть результат выполнения кода и его отдельных фрагментов. Так называемые “ноутбуки” – текстовые файлы, содержащие как текстовую информацию (например в формате Markdown), так и блоки с кодом, который можно запустить комбинацией клавиш и получить блок вывода данного кода.

Рис. 1. ВД в Jupyter Notebook с помощью Matplotlib



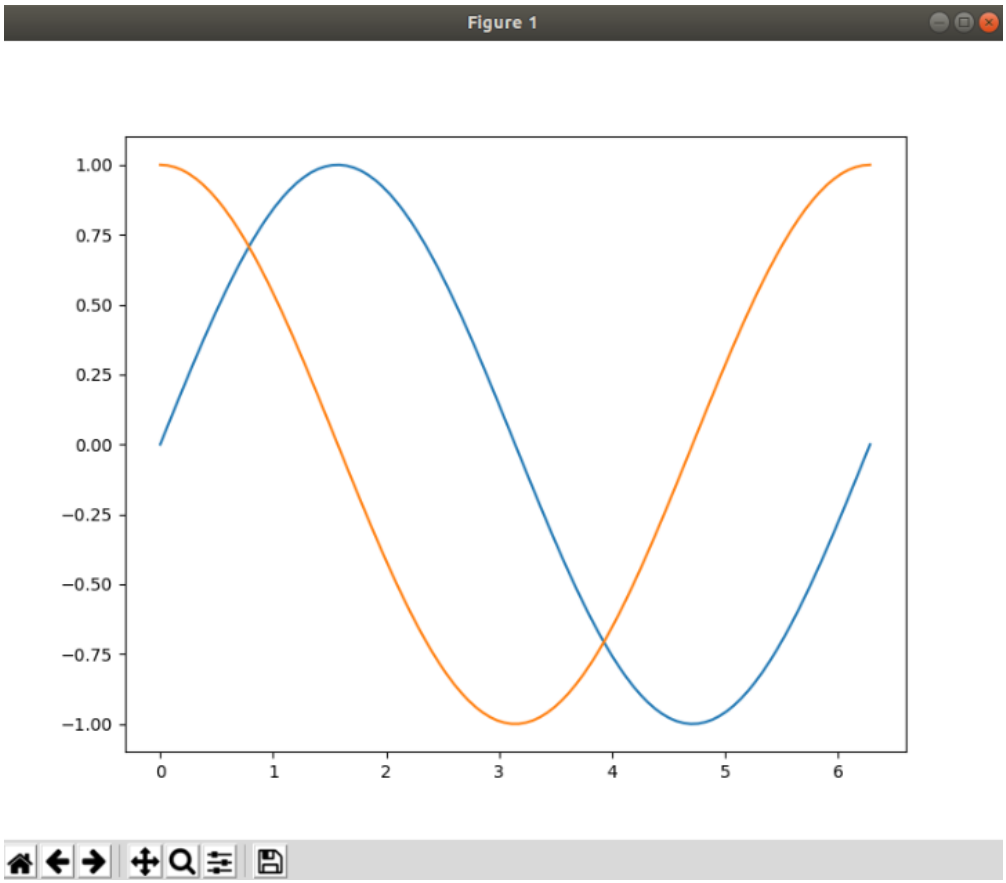
На рисунке 1 наглядно видна структура ноутбука: сверху расположен приветствующий текст в формате Markdown, далее идёт блок с произвольным кодом на языке Python, после него находится блок вывода. В данном случае вывод состоит из закодированного в base64 изображения с результатом работы библиотеки mpl (листинг 1).

Листинг 1. Пример вывода графических данных библиотекой mpl.

```
<div class="lm-Widget" data-mime-type="image/png">
    <img src="…mCC">
</div>
```

Данный способ вывода графики является достаточно простым в реализации, но имеет один существенный недостаток – теряется возможность организовать интерактивный вывод. Например, на рисунке 2 изображён вывод графика с помощью mpl в графическое окно.

Рис. 2. ВД с помощью Matplotlib в графическом окне Ubuntu



Как можно заметить, библиотека предоставляет возможность взаимодействия с графиком: масштабирования, смещения области видимости графика и т.д. Естественно, такие возможности отсутствуют при выводе графических данных статическим изображением.

4. Документация Jupyter Notebook

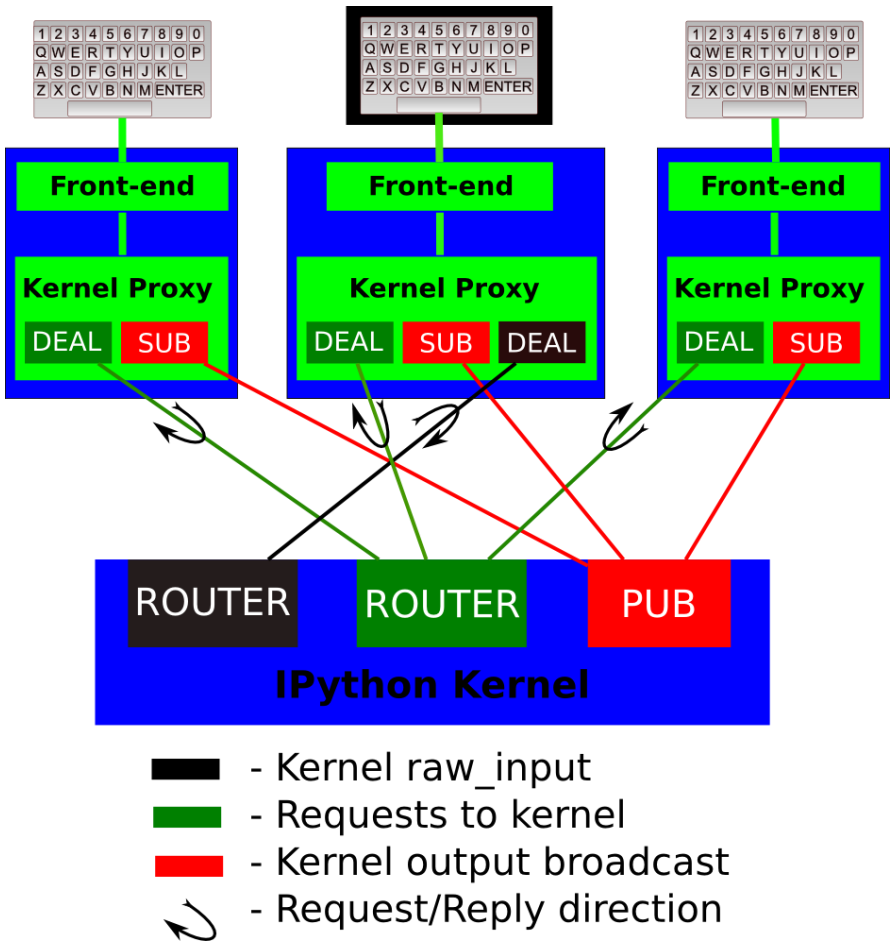
Jupyter Notebook предоставляет развёрнутую документацию по своему программному продукту [3].

4.1 Взаимодействие с ядрами в Jupyter Notebook

Ядро – это отдельная программа, которая запускается серверным приложением Jupyter Notebook. Во время работы с конкретным ноутбуком пользователь выбирает из списка установленных ядер то, которое будет отвечать за компиляцию ячеек с кодом в данном ноутбуке.

Для взаимодействия ядра с серверным приложением используется библиотека ZeroMQ (сокращённо ZMQ) [4]. С помощью так называемых ZMQ сокетов, работающих по протоколу TCP, реализуется возможность обмена сообщениями между различными приложениями.

Рис. 3. Схема коммуникации Jupyter Notebook с ядром IPython.



В общей сложности общение с ядром идёт по 5 сокетам:

1. Shell – основной сокет, по которому устанавливается начальное соединение с ядром и передача основных запросов ядру, таких как запрос на выполнение конкретного блока кода.

2. IOPub – сокет, отвечающий за передачу данных выполняемого кода (stdout, stderr) в блок вывода.

3. stdin – сокет, отвечающий за передачу данных во время ввода в работающую программу (если в коде присутствуют методы типа Read()).

4. Control – сокет, отвечающий за передачу запросов на прерывание выполнение кода, остановку или перезапуск ядра.

5. Heartbeat – сокет, регулярно проверяющий работоспособность ядра.

4.2 Типы сообщений в Jupyter Notebook

Отдельный раздел документации посвящён протоколу обмена сообщениями в Jupyter Notebook [5]. Далее перечислены основные типы сообщений для различных сокетов:

1. Сокет Shell

    - kernel_info_request– сообщение-запрос на присоединение к ядру.

    - kernel_info_reply – сообщение-ответ на запрос соединения. Содержит данные о версии протокола дальнейшей коммуникации.

    - execute_request – сообщение, запрашивающее выполнения кода у ядра. Сообщение содержит код для выполнения.

    - execute_reply – сообщение, сообщающее о результатах выполнения кода (“ok” или “error” или “aborted”).

2. Сокет Control

    - shutdown_request – сообщение-запрос на выключение или перезапуск ядра.
    - shutdown_reply – сообщение-ответ с результатом завершения или перезапуска работы ядра.
    - interrupt_request – сообщение-запрос на прерывание выполняемого ядром кода.
    - interrupt_reply – сообщение-ответ с результатом прерывания выполнения кода.

3. Сокет IOPub

    - display_data – сообщение, содержащее данные для блока вывода.
    - update_display_data – сообщение, содержащее дополнительные данные для блока вывода.
    - execute_result – сообщение, содержащее результаты выполнения блока кода.
    - clear_output – сообщение-запрос на очистку блока вывода.
    - status – сообщение, содержащее текущее состояние ядра (busy, idle, starting).

4. Сокет stdin

    - input_request – сообщение-запрос ядра на ввод данных пользователем.
    - input_reply – сообщение-ответ, содержащее введённые пользователем данные во время выполнения кода.

Heartbeat сокет не имеет никаких типов сообщений и лишь отправляет обратно каждый полученный на вход набор байт, сигнализируя о том что ядро всё ещё работает.

4.3 Выводимые типы в Jupyter Notebook

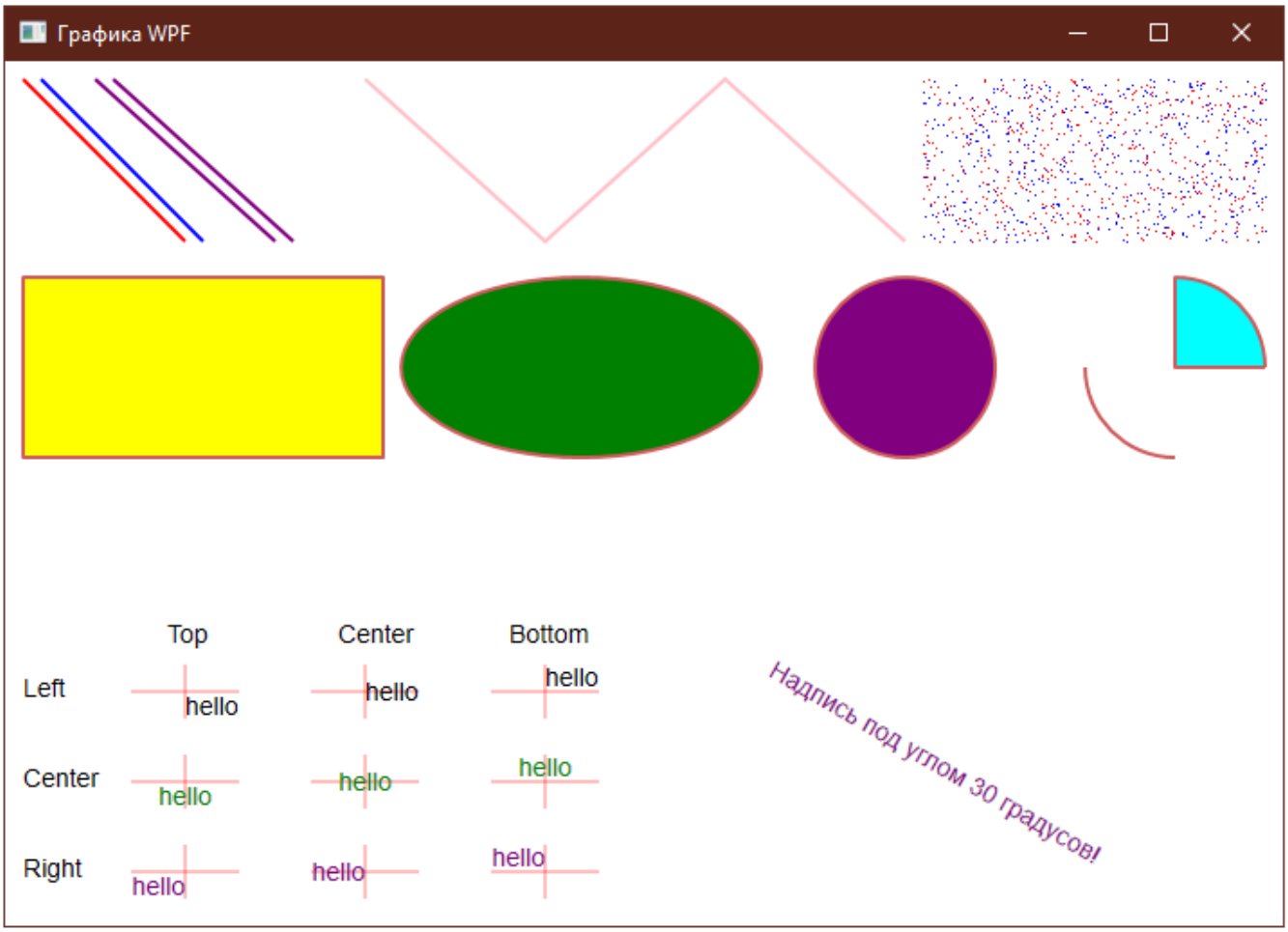
В ходе изучения документации было выяснено, что ядро может возвращать в веб-приложение данные в виде любого из MIME – типов [6]. В частности, интересна возможность вывода данных в виде html текста, в который можно при желании встроить JS код. Таким образом вывод Jupyter Notebook ограничен лишь возможностями JavaScript.

5. Визуализация данных в PascalABC.NET

5.1 GraphWPF

Как уже упоминалось во введении, за ВД в PascalABC.NET ответственен модуль GraphWPF, которые реализует отрисовку графических примитивов в окне с использованием методов отрисовки платформы. NetFramework.

Рис. 4. Вывод примитивов в GraphWPF (код в приложении, листинг 4)



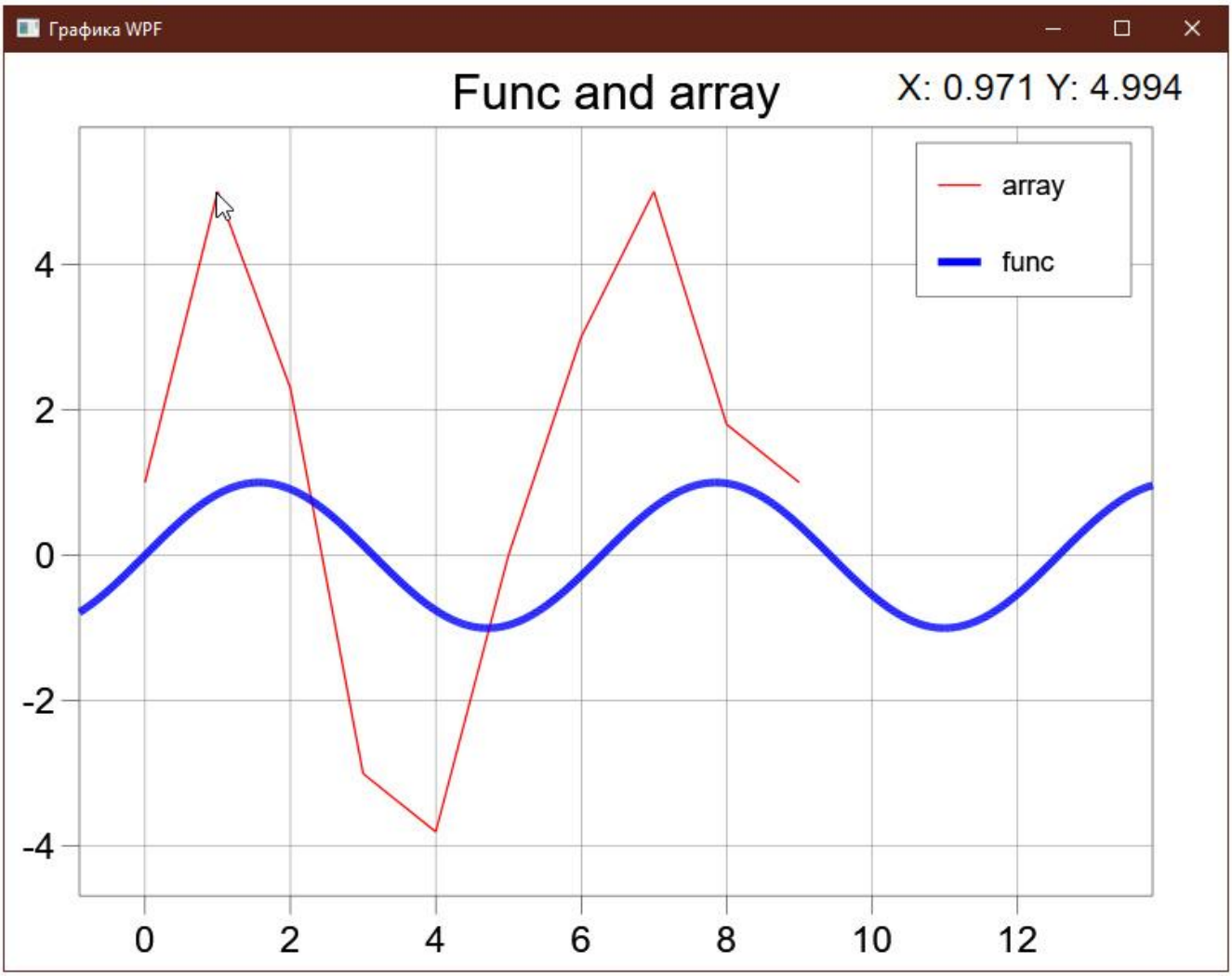
5.2 Plotter

На основе данного модуля в качестве курсовой работы был разработан модуль Plotter [2], служащий для построения разнообразных двумерных графиков. Модуль во многом повторяет функционал библиотеки mpl и программы MATLAB, что позволяет удобно строить графики с использованием языка PascalABC.NET.

Листинг 2. Код вывод графика с использованием Plotter.

```
uses Plotter;
begin
  var fig := new Figure();
  var ax := fig.AddSubplot();
  ax.Plot(new real[10](1,5,2.3,-3,-3.8,0,3,5,1.8,1)).
     linewidth := 0.5;
  ax.Plot((x:real) -> sin(x),Colors.Blue).
      linewidth := 2;
  ax.grid := true;
  ax.EqualProportion := true;
  ax.SetLegend(Arr('array','func'));
  ax.Title := 'Func and array';
  Show(fig);
end.
```

Рис. 5. Вывод программы с использованием Plotter (листинг 2)



6. Реализация графики ядра PascalABC.NET

Как было описано ранее, ядро может в качестве вывода возвращать JS код, с помощью которого было решено реализовывать графику.

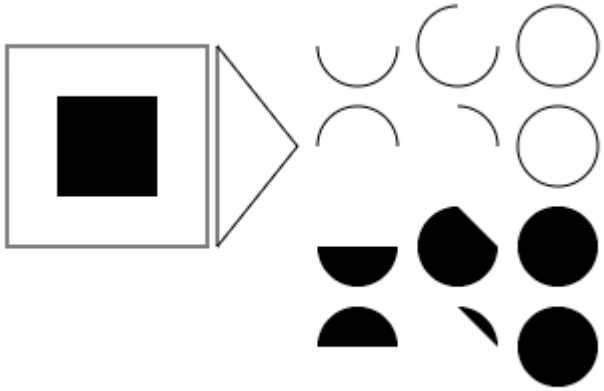
6.1 Графические примитивы на JavaScript canvas

Canvas – это HTML элемент, использующийся для рисования графики средствами языка JS [7]. Он может, к примеру, использоваться для рисования графов, создания коллажей или простой анимации.

Листинг 3. Код вывода графических примитивов на canvas JS.

```
<canvas id="canvas" width="500" height="300"></canvas>
<script>
     var ctx = document.getElementById('canvas')
                      .getContext('2d');
     ctx.fillRect(50, 50, 50, 50);
     ctx.strokeRect(25, 25, 100, 100);
     ctx.beginPath();
     ctx.moveTo(130, 25);
     ctx.lineTo(170, 75);
     ctx.lineTo(130, 125);
     ctx.lineTo(130, 25);
     ctx.stroke();
     for (var i = 0; i < 4; i++) {
      for (var j = 0; j < 3; j++) {
        ctx.beginPath();
        var x = 200 + j * 50;
        var y = 25 + i * 50;
        var endAngle = Math.PI + (Math.PI * j) / 2;
        ctx.arc(x, y, 20, 0, endAngle, i % 2 !== 0);
        if (i > 1) ctx.fill(); else ctx.stroke();
        }
      }
</script>
```

Рис. 6. Вывод программы с использованием canvas JS (листинг 3)



6.2 Генерация JavaScript кода

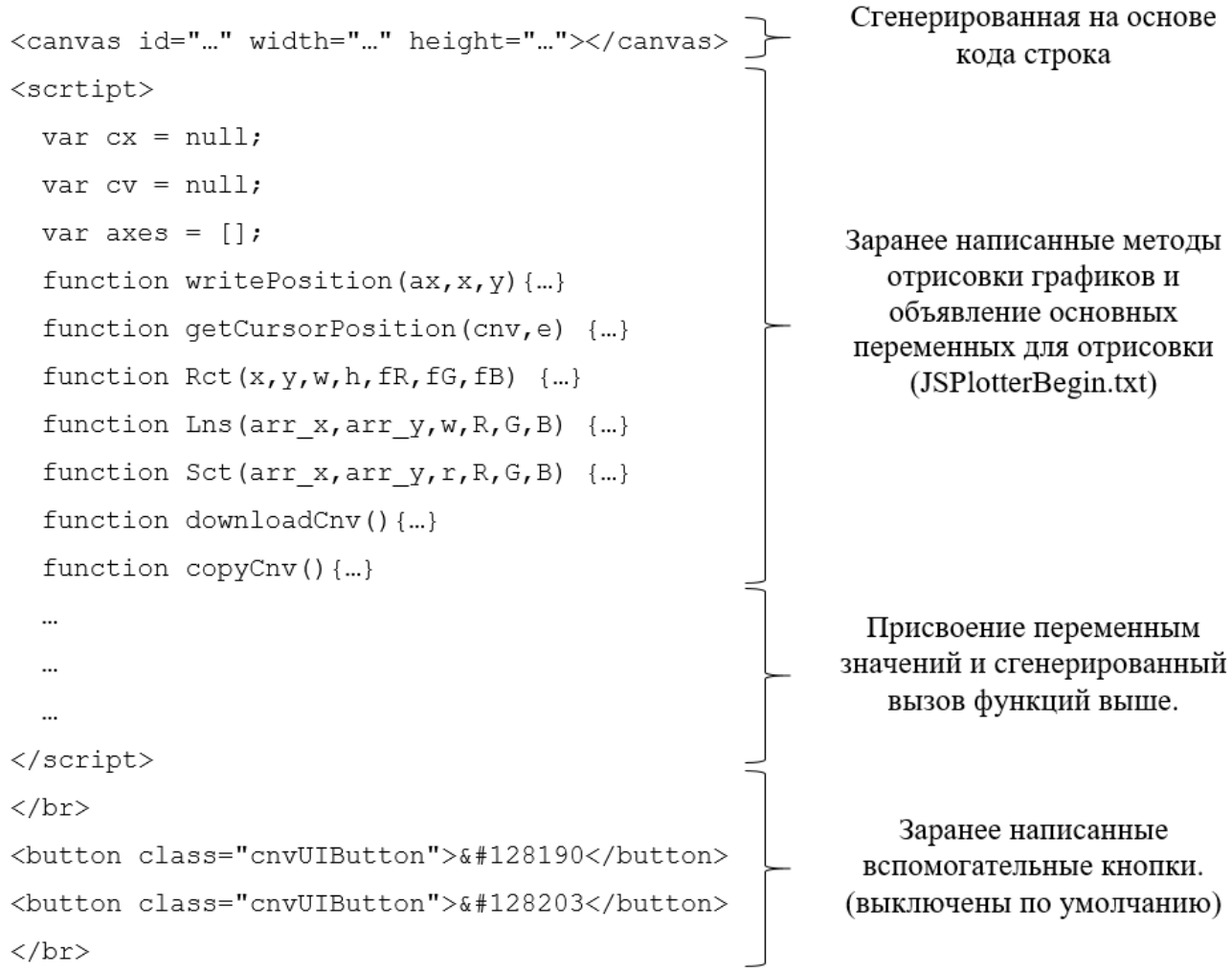
Из примера выше видно, что JS позволяет выводить все основные примитивы, которые выводит модуль GraphWPF. Из этого вытекает логичное решение: переписать модули GraphWPF и Plotter таким образом, чтобы вместо методов отрисовки соответствующего примитива в окне происходила генерация строки с кодом на JS, выводящим аналогичный примитив на canvas.

Но возникла следующая проблема: при использовании GraphWPF и отрисовке небольшого количества примитивов всё работает штатно, но при использовании модуля Plotter происходит отрисовка огромного количества примитивов и соответственно генерация огромного количества кода на JS, который долго передаётся стандартным потоком вывода программы. Сам модуль Plotter для ускорения вывода изображения использует механизм FastDraw, который позволяет запрашивать отрисовку не каждого отдельного примитива, а сложных элементов графика.

6.3 Оптимизация сгенерированного JavaScript кода

Решение проблемы с генерацией большого количества JS кода модулем Plotter было следующим – перенести часть функционала по отрисовке сложной геометрии графика из PascalABC на JS. Другими словами, потребовалось написать методы отрисовки графиков на JS, задача модуля же свелась к генерации кода с вызовом этих методов и передаче в них сырых данных для отрисовки. Более того, такой подход позволял реализовать ту самую интерактивность, которой так не хватает выводу mpl в Jupyter Notebook. В процессе разработки вывод графических модулей принял следующий вид:

Рис. 7. Шаблон вывода модуля Plotter.



Для однообразности и оптимизации, для модуля GraphWPF были также написаны соответствующие JS методы, уменьшающие количество генерируемого модулем кода.

6.4 Сравнение производительности

Для проверки быстродействия был рассмотрен случай вывода большого количества прямых линий – основных составляющих любого графика.

1. Первый вариант работы предусматривает, что на PascalACB.NET будет сгенерирован лишь вызов заранее написанной JS функции, рисующей по заданным массивам координаты линии с указанным цветом и шириной.
2. Второй вариант работы предусматривает генерацию JS кода для каждого отдельной линии с указанием её ширины и цвета.

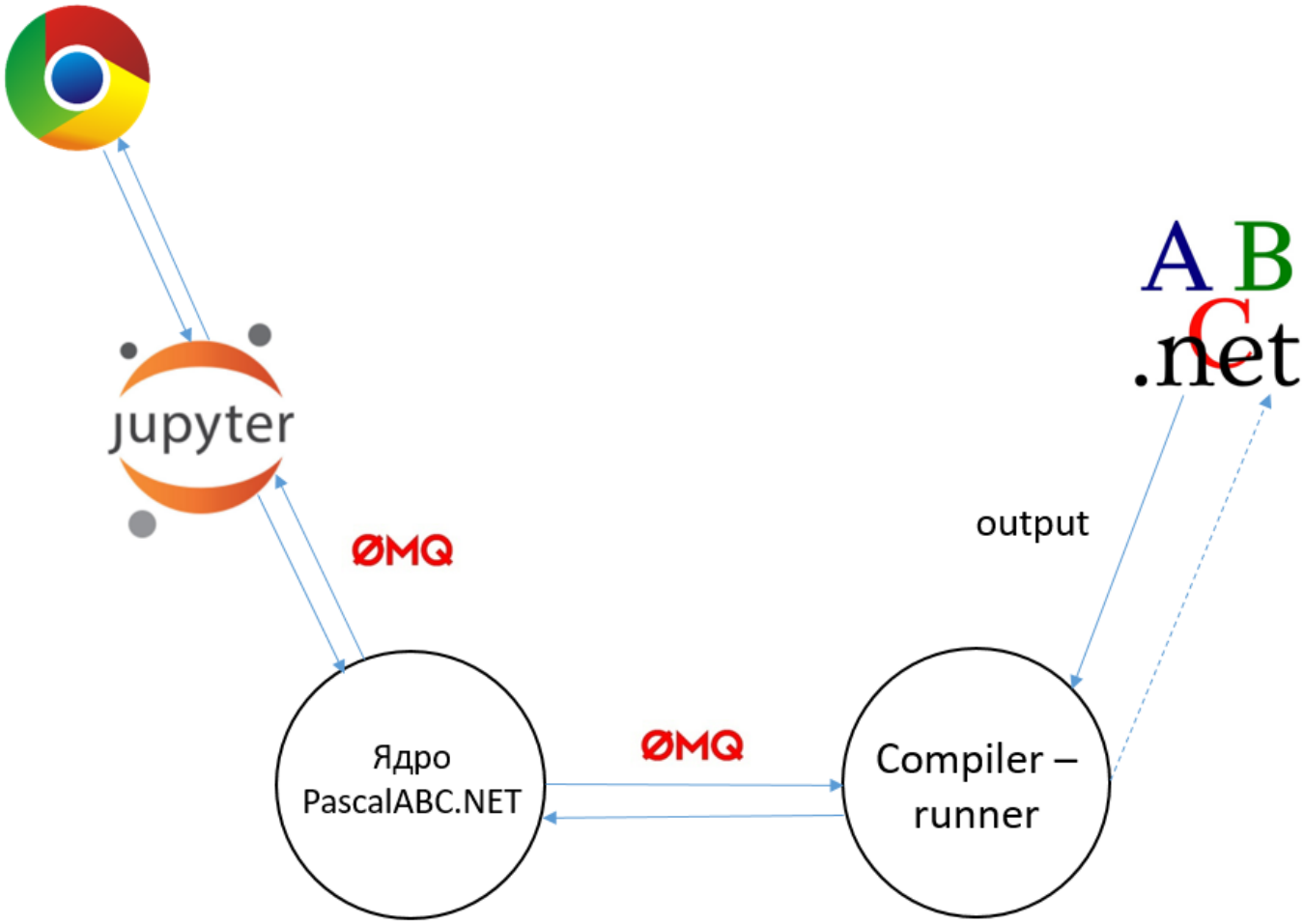
Таблица 1. Сравнение производительности при генерации кода отрисовки 100000 линий

	Размер кода отрисовки, байт	**Время передачи в поток вывода, мс**	**Время отрисовки примитивов, мс**	**Общее время, мс**
1	1 535 477	982	540	1032
2	14 435 693	5852	1024	6876

6.5 Структура вывода графических данных

В результате работы над ядром PascalABC.NET для Jupyter Notebook была получена следующая структура:

Рис. 8. Принцип работы ядра PascalABC.NET.



Compile-runner - программа, которая компилирует по запросу ядра код в .exe файл запускает его и перехватывает его поток вывода. Далее с помощью ZMQ сокетов текст вывода попадает сначала в программу самого ядра, после чего на серверное приложение Jupyter Notebook. Соответственно пользователь в браузере видит уже результат интерпретированного JS кода.

Данный подход также позволяет адекватно выводить текстовые данные из скомпилированной PascalABC программы. Другими словами, что про-

грамма ядра, что программа компилятора абсолютно одинаково обрабатывают что текстовый, что графический вывод скомпилированной программы. Интерпретация полученных данных происходит уже на frontend части веб-приложения.

6.6 Структура разработанных модулей

Итогом разработки стали модули GraphJupyter и PlotterJupyter, аналоги GraphWPF и Plotter соответственно.

1. GraphJupyter – модуль реализует методы отрисовки из GraphWPF [8]. Из модуля были удалены методы работы с событиями и класс графического окна (рис. 9).

   Были добавлены следующие методы:

   - WindowSize(w,h :integer); - задаёт размер области отрисовки
   - NeedButtons(f : boolean); - включение/выключение вывода вспомогательных кнопок сохранения и копирования полученного изображения (рис. 10)

2. PlotterJupyter – модуль реализует методы отрисовки из Plotter [2]. Интерактивный метод вывода текущей позиции курсора относительно координатной сетки графика также был реализован благодаря переносу части функционала на JS (рис. 11).

   Также был добавлен метод NeedButtons(f : boolean) с аналогичной функцией что и в GraphJupyter (рис. 12).

Больше примеров работы графических модулей можно увидеть в приложении.

6.7 Пример графического вывода ядра

Рис. 9. Вывод кода (листинг 4) с помощью GraphJupyter.

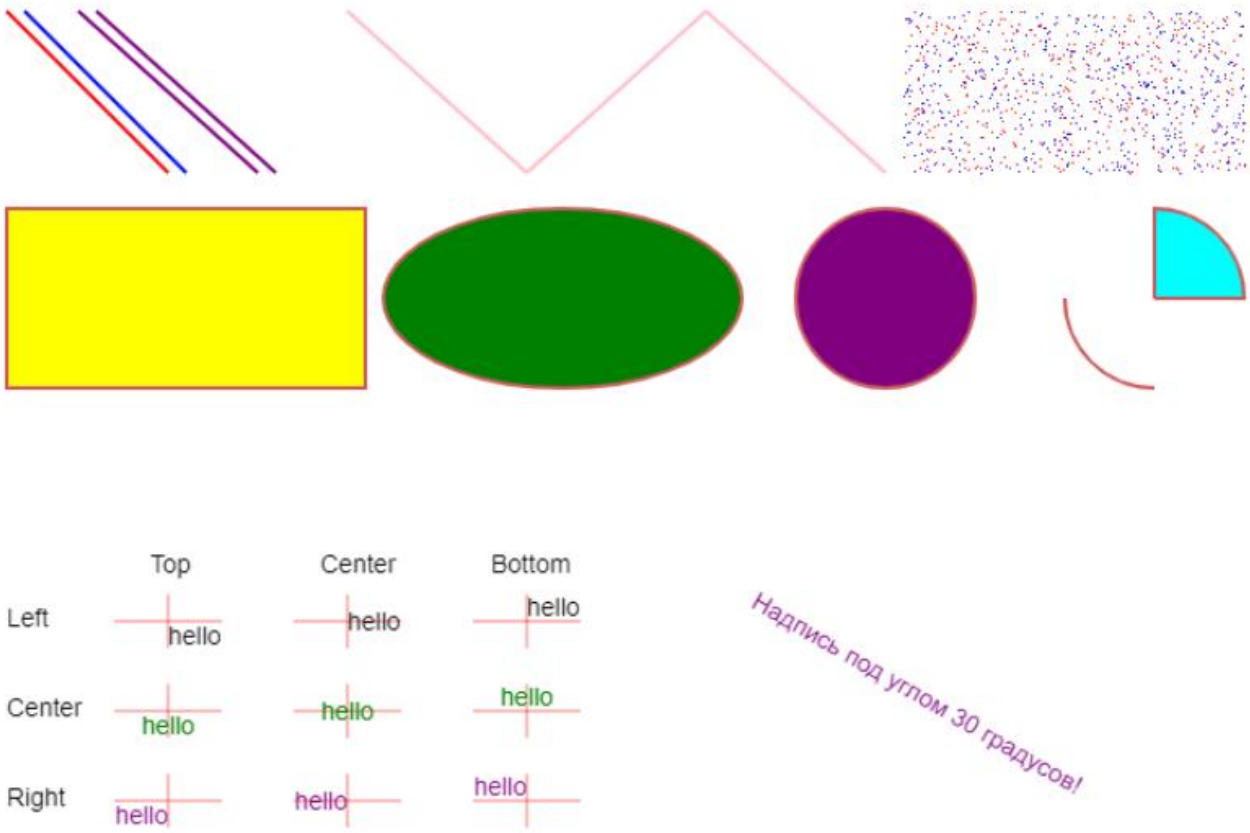


Рис. 10. Пример вывода GraphJupyter в интерфейсе Jupyter Notebook.

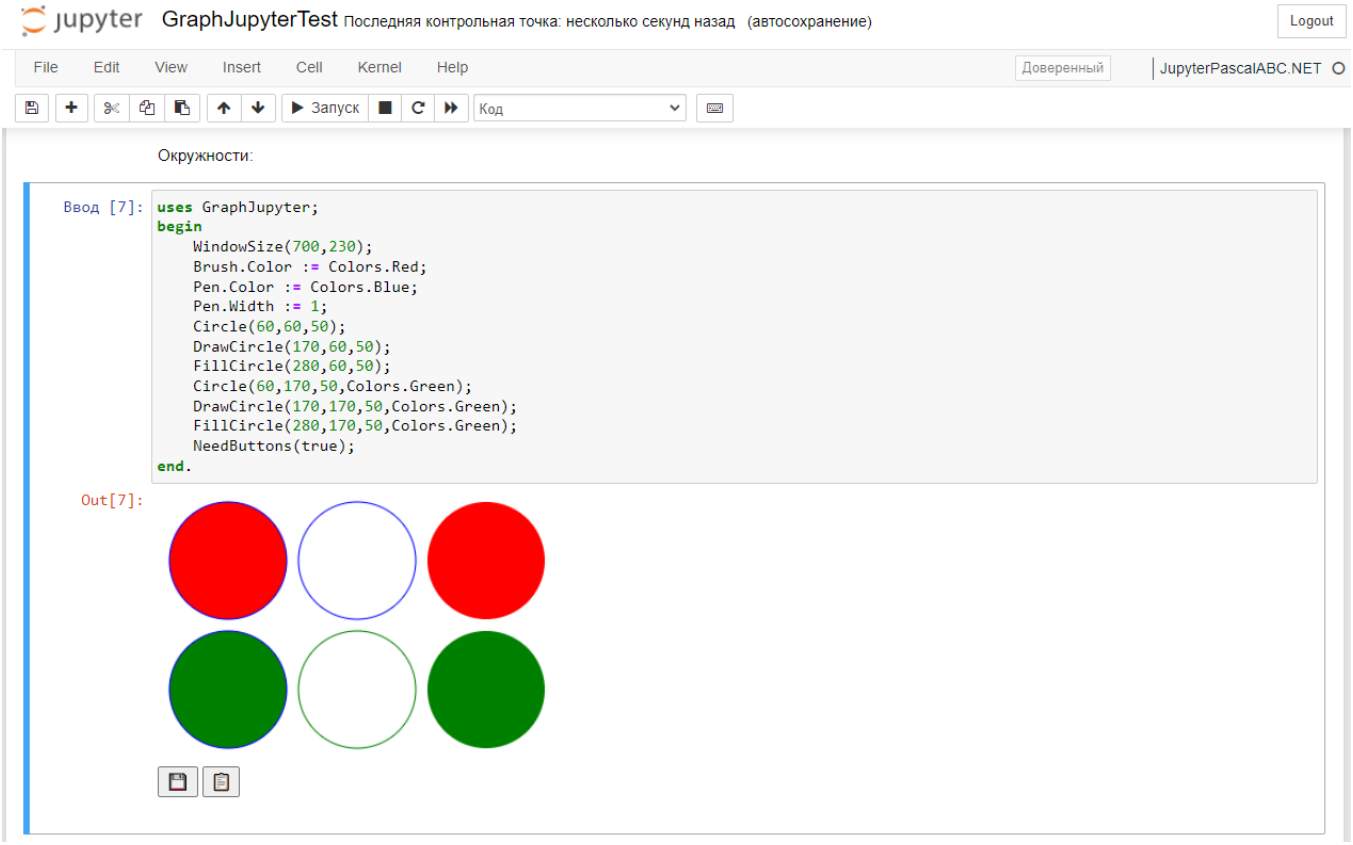


Рис. 11. Вывод программы с использованием PlotterJupyter (листинг 2)

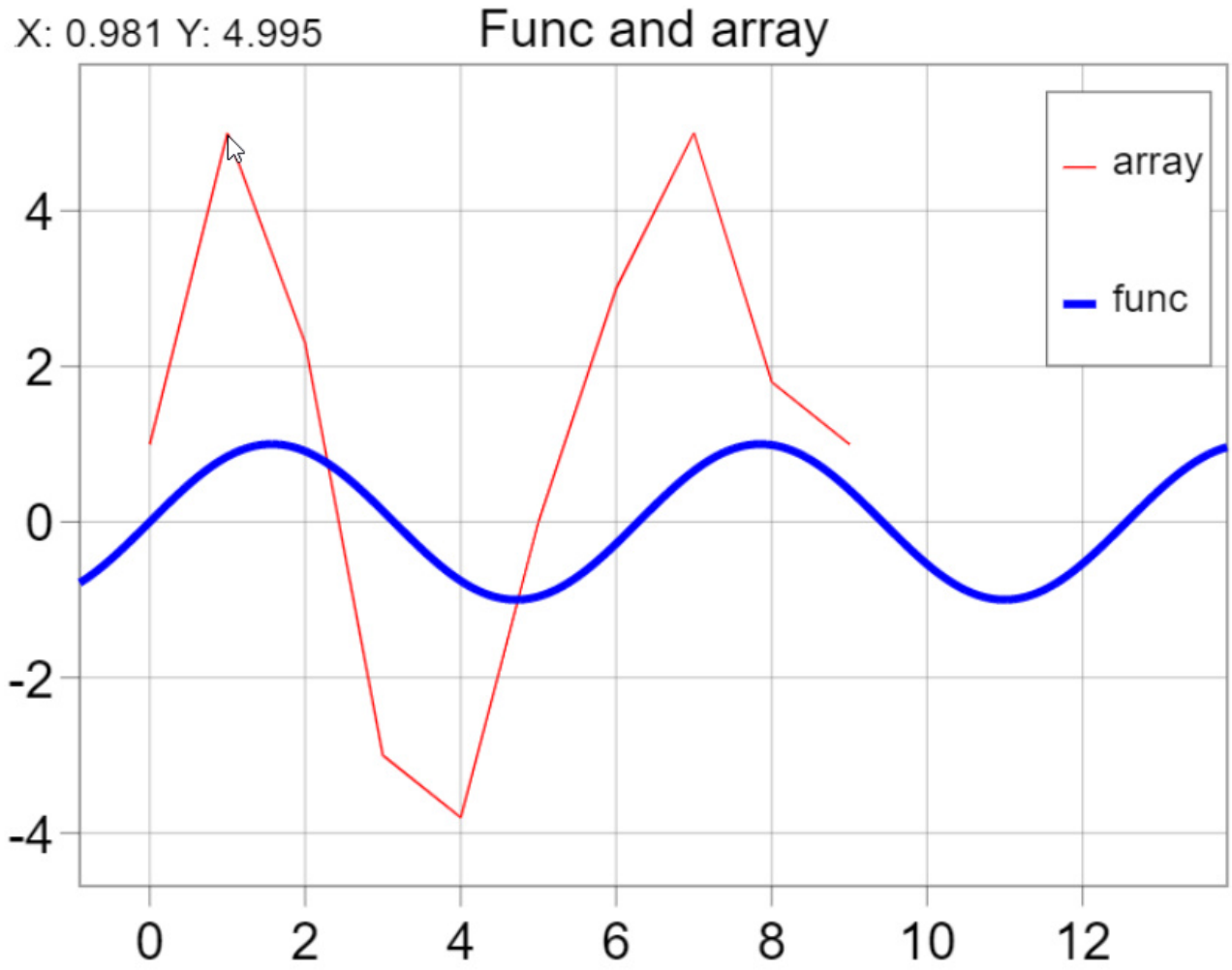
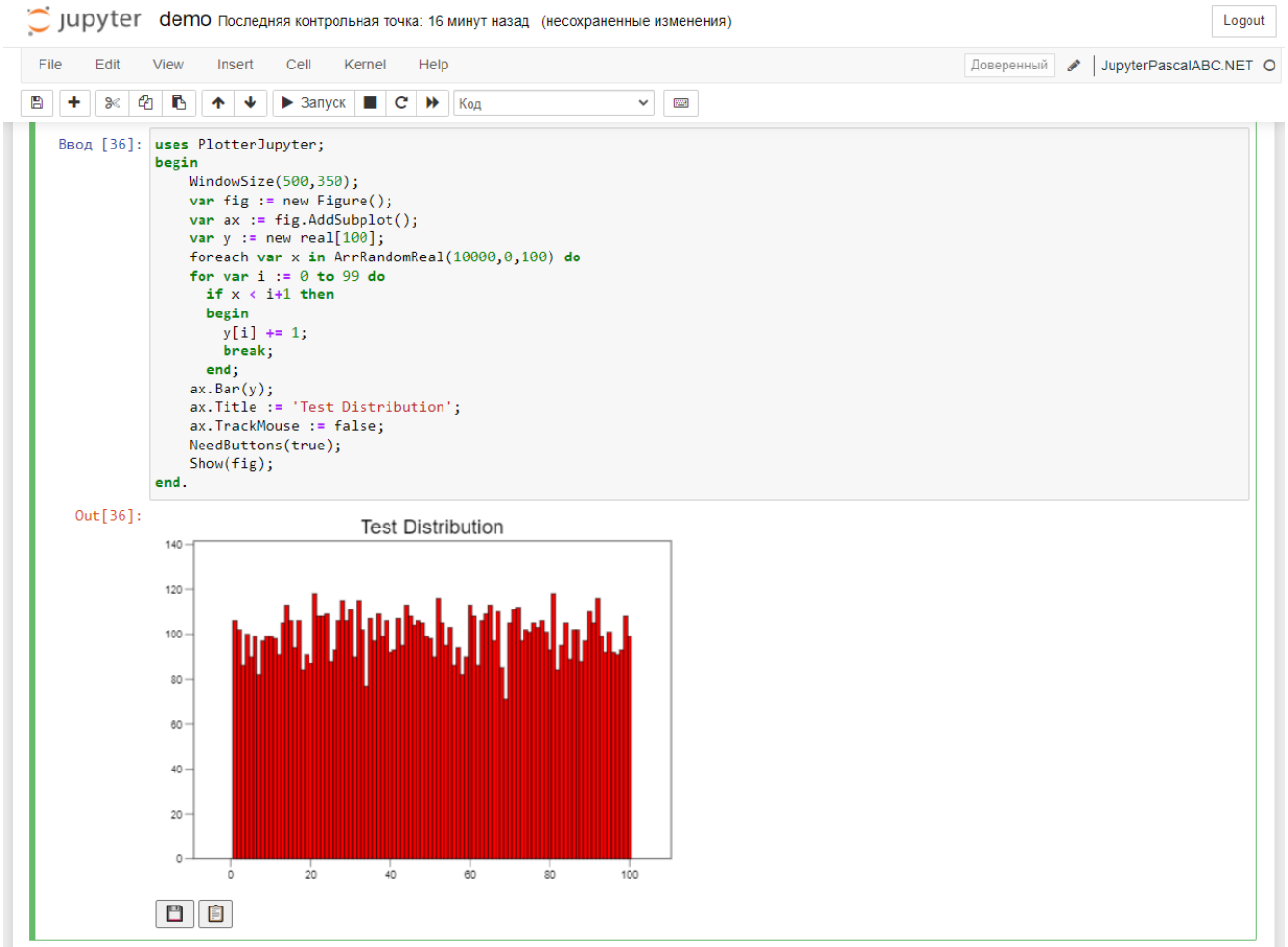


Рис. 12. Пример вывода PlotterJupyter в интерфейсе Jupyter Notebook.



7. Установка ядра в Jupyter Hub

Jupyter Hub [8] – многопользовательская версия Jupyter Notebook, предоставляет доступ к вычислительным средствам и ресурсам платформы Jupyter без установки программного обеспечения. Требуемые для работы ядра устанавливаются на сервер Jupyter Hub, после чего пользователи по ссылке могут взаимодействовать с ними.

7.1 Запуск программ ядра на Linux

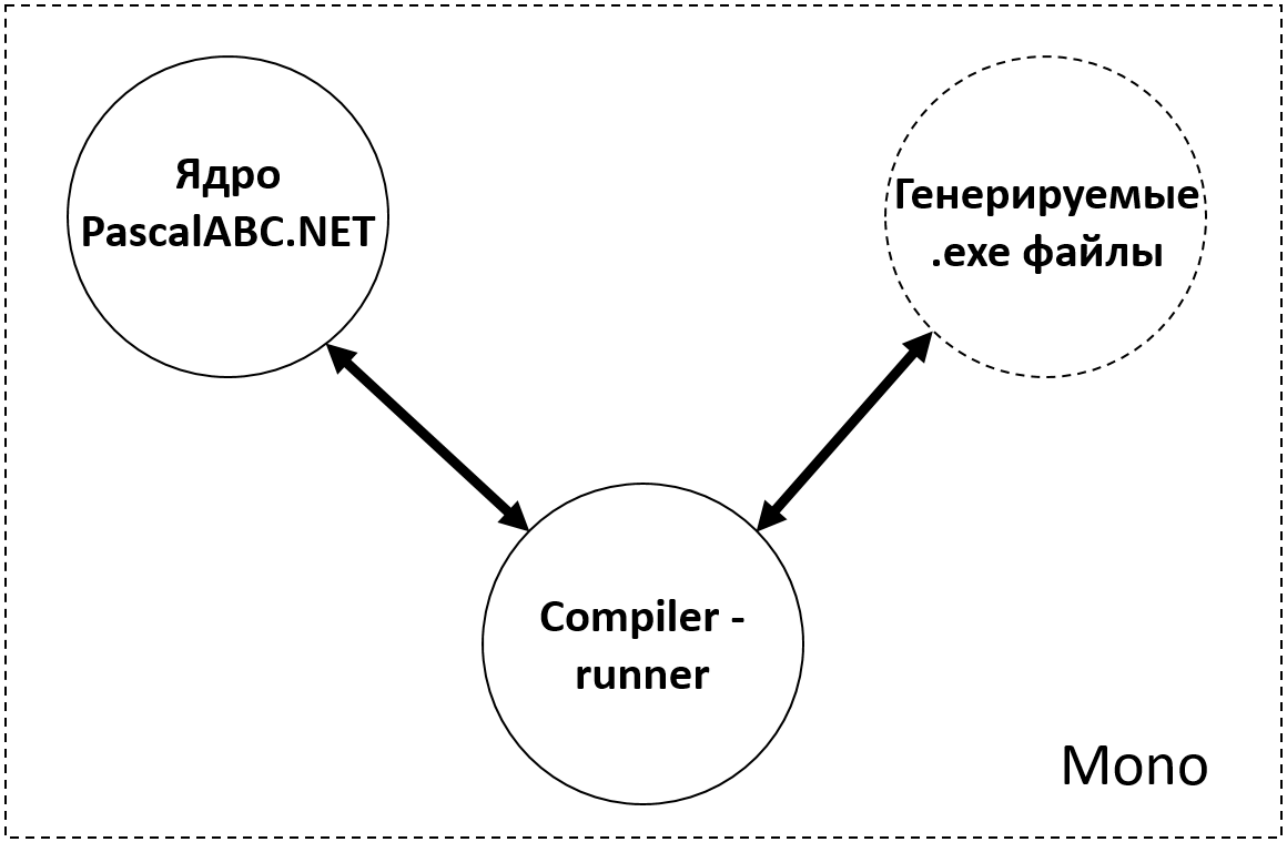
Основной проблемой при установке ядра в Jupyter Hub стала операционная система Linux, на которой работает сервер. До этого ядро и программа-компилятор разрабатывались на NET Framework [9] и тестировались только под операционной системой Windows, соответственно появилась задача по запуску этих программ под Linux. Имелось 2 варианта:

- Переписать программы с минимальными изменениями на более современную платформу NET [10], позволяющую собирать и запускать программы не только на Windows, но и на Linux и macOS.
- Использовать платформу Mono [11], реализующую NET Framework на Unix-подобных операционных системах (в т.ч. Linux).

Первый вариант был достаточно быстро отброшен по причине того, что .dll библиотеки компилятора PascalABC.NET не смогли бы должным образом работать в программе-компиляторе ядра, написанном на NET.

Использование Mono выглядело более целесообразным, особенно учитывая то, что на сервере Jupyter Hub уже были приложения использующие Mono, соответственно список зависимостей при установке ядра PascalABC.NET не расширялся. Запуск скомпилированных программ также возможен только при помощи Mono. Таким образом все элементы ядра PascalABC.NET запускаются с использованием данной платформы (рис. 13).

Рис. 13. Схема работы программ ядра под Linux.



7.2 Зависимость WPF

Очередной проблемой стало использование в графических модулях ядра библиотек PresentationCore.dll и PresentationFramework.dll, которые используются в технологии визуализации данных WPF [12]. Данные библиотеки содержат необходимые классы для работы с цветами, шрифтами, кистями и т.п. К сожалению, оказалось, что платформа Mono не поддерживает технологию WPF, разработки в этой сфере были остановлены более 10 лет назад.

В итоге было принято решение – убрать зависимость от данных библиотек, путём написания собственных классов-контейнеров для работы с графическими данными. Были разработаны необходимые классы Color, Colors и FontOptions. Если с классами Color и Colors, представляющими контейнер для цвета и список цветов соответственно, никаких проблем не возникло, то с FontOptions пришлось менять логику работы графических модулей в плане вывода текста.

Ранее модуль PlotterJupyter располагал методами, позволяющими определять размеры конкретной строки с заданным шрифтом, что позволяло грамотно позиционировать строки на canvas-е. Данный подход не являлся наилучшим, поскольку система WPF определяла размеры текста, а отрисовка производилась уже методами языка JS. В таком случае могли возникать проблемы с наличием разных шрифтов на клиентской части приложения и на сервере. Когда стало понятно, что использовать WPF классы не получится, было решено передать функционал позиционирования и определения размера текста на клиентскую часть. Таким образом получилось решить сразу две проблемы: избавить модуль от зависимости WPF и решить проблемы с шрифтами. Теперь модуль отвечает только за хранение названия шрифта и его размеров, в свою очередь с помощью JS на клиентской части уже определяются необходимые параметры текста для его позиционирования.

7.3 Установка ядра в Jupyter Hub

Для установки ядра на сервер требуется совершить несколько консольных команд, которые переместят в нужное место файлы ядра и пропишут в систему Jupyter путь до программы ядра, параметры его запуска, название и прочие конфигурационные данные.

С целью автоматизации этого процесса была использована программа Docker [13], с помощью которой удобно устанавливать нужные ядра в Jupyter Hub, а также обновлять и модифицировать их в будущем (листинг 4).

Листинг 4. Примерный код, отвечающий за установку ядра.

```
RUN mkdir -p /JupyterPABC_kernel
WORKDIR /JupyterPABC_kernel
RUN wget -nv https://template.ru/JPABC.zip && \
    unzip JPABC.zip
RUN cd /JupyterPABC_kernel &&
     sudo jupyter kernelspec install
          kernel-spec --name=pabc_release
```

8. Заключение

В рамках дипломной работы была изучена платформа Jupyter Notebook и способы ВД с помощью ядра IPython и библиотеки Matplotlib. Полученные знания были использованы при разработке ядра PascalABC.NET и адаптации существующих графических модулей под работу в веб-приложении [14].

Главная цель работы достигнута, реализованы графические модули ядра PascalABC.NET для Jupyter Notebook.

Решены все поставленные задачи:

- Разработан способ вывода графики из программы на PascalABC.NET в Jupyter Notebook
- Разработаны аналоги модулей GraphWPF и Plotter для вывода графики в ядре PascalABC.NET Jupyter Notebook
- Графические модули ядра адаптированы для работы под Mono
- Ядро PascalABC.NET Jupyter Notebook развёрнуто на сервере Jupyter Hub.

В процессе разработки была решена проблема интерактивности вывода модуля Plotter путём отрисовки графических данных на frontend-е с помощью элемента canvas и языка JS. Также избыточное количество сгенерированного JS кода было на порядки уменьшено путём переноса сложных методов отрисовки из графических модулей на язык JS. Полученная структура вывода графических данных полностью унифицирована с выводом текстовых данных и предоставляет перспективы модернизации уже имеющихся графических модулей.

Результатом работы являются модули GraphJupyter и PlotterJupyter, интегрированные в ядро PascalABC.NET для Jupyter Notebook. Модули максимально близко повторяют функционал GraphWPF и Plotter соответственно и позволяют в большей степени использовать функционал PascalABC.NET в веб-приложении Jupyter без предварительной установки IDE.

Ознакомиться с исходным кодом можно по следующим ссылкам:

- Ядро PascalABC: https://github.com/Looken15/JupyterPascalABC.NET
- Приложение-компилятор: https://github.com/barakuda211/ZMQServerPas

9. Список литературы

1. Wikipedia: Data visualization – URL: https://en.wikipedia.org/wiki/Data_visualization (дата обр. 20.05.2022)
2. Графическая библиотека визуализации данных для PascalABC.NET – URL: https://hub.sfedu.ru/repository/material/801288947/ (дата обр. 12.02.2022)
3. ZMQ – URL: https://zeromq.org/get-started/ (дата обр. 20.12.2021)
4. Документация Jupyter – URL: https://jupyter-client.readthedocs.io/en/latest/messaging.html# (дата обр. 20.12.2021)
5. MIME типы данных – URL: https://en.wikipedia.org/wiki/MIME (дата обр. 10.02.2022)
6. Руководство по Canvas – URL: https://developer.mozilla.org/ru/docs/Web/API/Canvas_API/Tutorial (дата обр. 15.02.2022)
7. Документация по GraphWPF – URL: http://pascalabc.net/downloads/pabcnethelp/index.htm?page=PABCUnits/GraphWPF/index.html (дата обр. 15.02.2022)
8. Jupyter Hub – URL: https://jupyter.org/hub (дата обр. 25.05.2022)
9. NET Framework – URL: https://ru.wikipedia.org/wiki/.NET_Framework (дата обр. 25.05.2022)
10. NET – URL: https://en.wikipedia.org/wiki/.NET (дата обр. 25.05.2022)
11. Mono – URL: https://ru.wikipedia.org/wiki/Mono (дата обр. 25.05.2022)
12. WPF – URL: https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation (дата обр. 27.05.2022)
13. Docker – URL: https://ru.wikipedia.org/wiki/Docker (дата обр. 01.06.2022)

14. Jupyter-ноутбуки для PASCALABC.NET и их использование в учебном процессе / Студенческая конференция (12 мая 2022 г.). Материалы конференции. Ростов н/Д; Изд-во ЮФУ.

10. Приложение

Листинг 5. Пример вывода графических примитивов GraphWPF.

```
uses GraphWPF;
begin
    Window.SetSize(710,480);
    Pen.Color := Colors.Red;
    Pen.Width := 2;
    Line(10,10,100,100);                    //линия
    Line(20,10,110,100,Colors.Blue);

    var arr1:= Arr((new Point(50,10),new Point(150,100)),
                (new Point(60,10),new Point(160,100)));
    Lines(arr1,Colors.Purple);               //массив линий

    var arr:= Arr(new Point(200,10),new Point(300,100),
                 new Point(400,10),new Point(500,100));
    Pen.Color := Colors.IndianRed;
    PolyLine(arr,Colors.Pink);              //кривая

    for var i := 10 to 500 do               //окрашивание отдельных пикселей
    begin
        SetPixel(Random(510,700),Random(10,100),Colors.Red);
        SetPixel(Random(510,700),Random(10,100),Colors.Blue);
    end;

    Rectangle(10,120,200,100,Colors.Yellow); //прямоугольник
    Ellipse(320,170,100,50,Colors.Green);    //эллипс
    Circle(500,170,50,Colors.Purple);        //окружность

    Sector(650,170,50,0,90,Colors.Aqua);     //сектор
    Arc(650,170,50,180,270);                 //дуга

    Brush.Color := Colors.Black;
    Pen.Width := 0.5;
    Pen.Color := Colors.Red;

    for var i := 1 to 3 do              //вывод текста с разным выравниванием
        for var j := 1 to 3 do
        begin
            Line(i*100-30,300+j*50,i*100+30,300+j*50);
            Line(i*100,300+j*50-15,i*100,300+j*50+15);
        end;
    TextOut(90,310,'Top');
    TextOut(185,310,'Center');
    TextOut(280,310,'Bottom');
    TextOut(10,340,'Left');
    TextOut(10,390,'Center');
    TextOut(10,440,'Right');


    Font.Color := Colors.Black;
    TextOut(100,350,'hello',Alignment.LeftTop);
    TextOut(200,350,'hello',Alignment.LeftCenter);
    TextOut(300,350,'hello',Alignment.LeftBottom);

    Font.Color := Colors.Green;
    TextOut(100,400,'hello',Alignment.CenterTop);
    TextOut(200,400,'hello',Alignment.Center);
    TextOut(300,400,'hello',Alignment.CenterBottom);
```

```
    Font.Color := Colors.Purple;
    TextOut(100,450,'hello',Alignment.RightTop);
    TextOut(200,450,'hello',Alignment.RightCenter);
    TextOut(300,450,'hello',Alignment.RightBottom);

    TextOut(430,330,'Надпись под углом 30 градусов!',Alignment.LeftTop,30);
end.
```

Листинг 6. Пример вывода текста с помощью GraphJupyter.

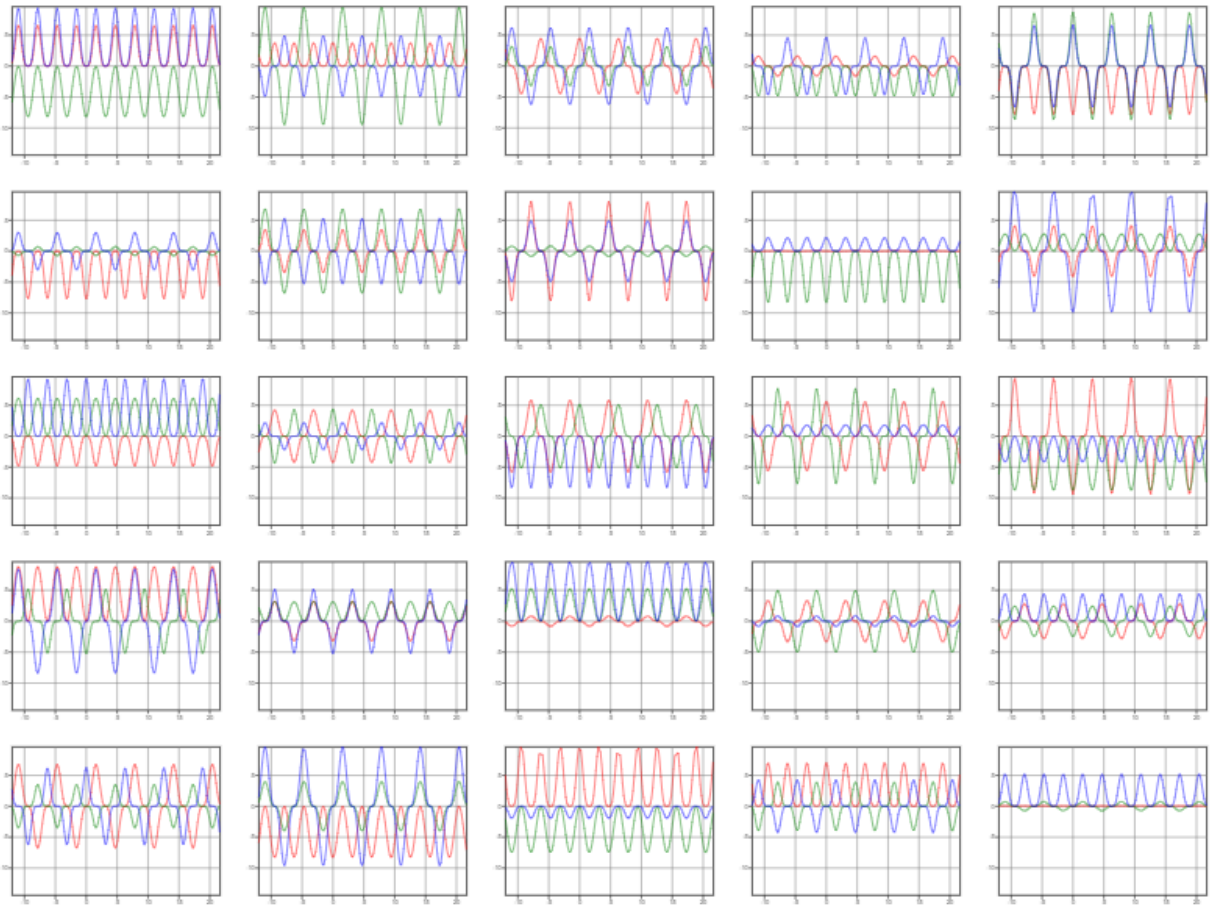
```
uses GraphJupyter;
begin
    WindowSize(700,300);
    Brush.Color := Colors.Black;
    Pen.Width := 0.5;
    Pen.Color := Colors.Red;

    for var i := 1 to 3 do
        for var j := 1 to 3 do
        begin
            Line(i*100-30,j*50,i*100+30,j*50);
            Line(i*100,j*50-15,i*100,j*50+15);
        end;
    TextOut(90,10,'Top');
    TextOut(185,10,'Center');
    TextOut(280,10,'Bottom');
    TextOut(10,40,'Left');
    TextOut(10,90,'Center');
    TextOut(10,140,'Right');
    Font.Color := Colors.Black;
    TextOut(100,50,'hello',Alignment.LeftTop);
    TextOut(200,50,'hello',Alignment.LeftCenter);
    TextOut(300,50,'hello',Alignment.LeftBottom);
    Font.Color := Colors.Green;
    TextOut(100,100,'hello',Alignment.CenterTop);
    TextOut(200,100,'hello',Alignment.Center);
    TextOut(300,100,'hello',Alignment.CenterBottom);
    Font.Color := Colors.Purple;
    TextOut(100,150,'hello',Alignment.RightTop);
    TextOut(200,150,'hello',Alignment.RightCenter);
    TextOut(300,150,'hello',Alignment.RightBottom);
    TextOut(400,30,'Текст под углом 30 градусов!',Alignment.LeftTop,30);
end.
```

Рис. 13. Вывод листинга 5.

Листинг 7. Пример вывода большого количества графиков PlotterJupyter.

```
uses PlotterJupyter;
begin
    var rows := 5; var cols := 5;
    var fig := new Figure();
    var cls := new Color[3](Colors.Red,Colors.Green,Colors.Blue);
    for var i := 0 to rows*cols-1 do
    begin
    var ax := fig.AddSubplot(rows,cols,i);
    for var c := 0 to 2 do
    begin
      var cos_sin := Random(2);
      var len := Random(2,5);
      var koef := Random(-10.0,10.0);
      var crv : Curve;
      if cos_sin = 1 then
        crv := ax.Plot((x: real)->(koef*power(cos(x),len)))
      else
        crv := ax.Plot((x: real)->(koef*power(sin(x),len)));
      crv.SetFacecolor(cls[c]);
    end;
    ax.Grid := true;
    ax.Setylim(-12,12);
    ax.EqualProportion := true;
    ax.TrackMouse := false;
    end;
    Show(fig);
end.
```

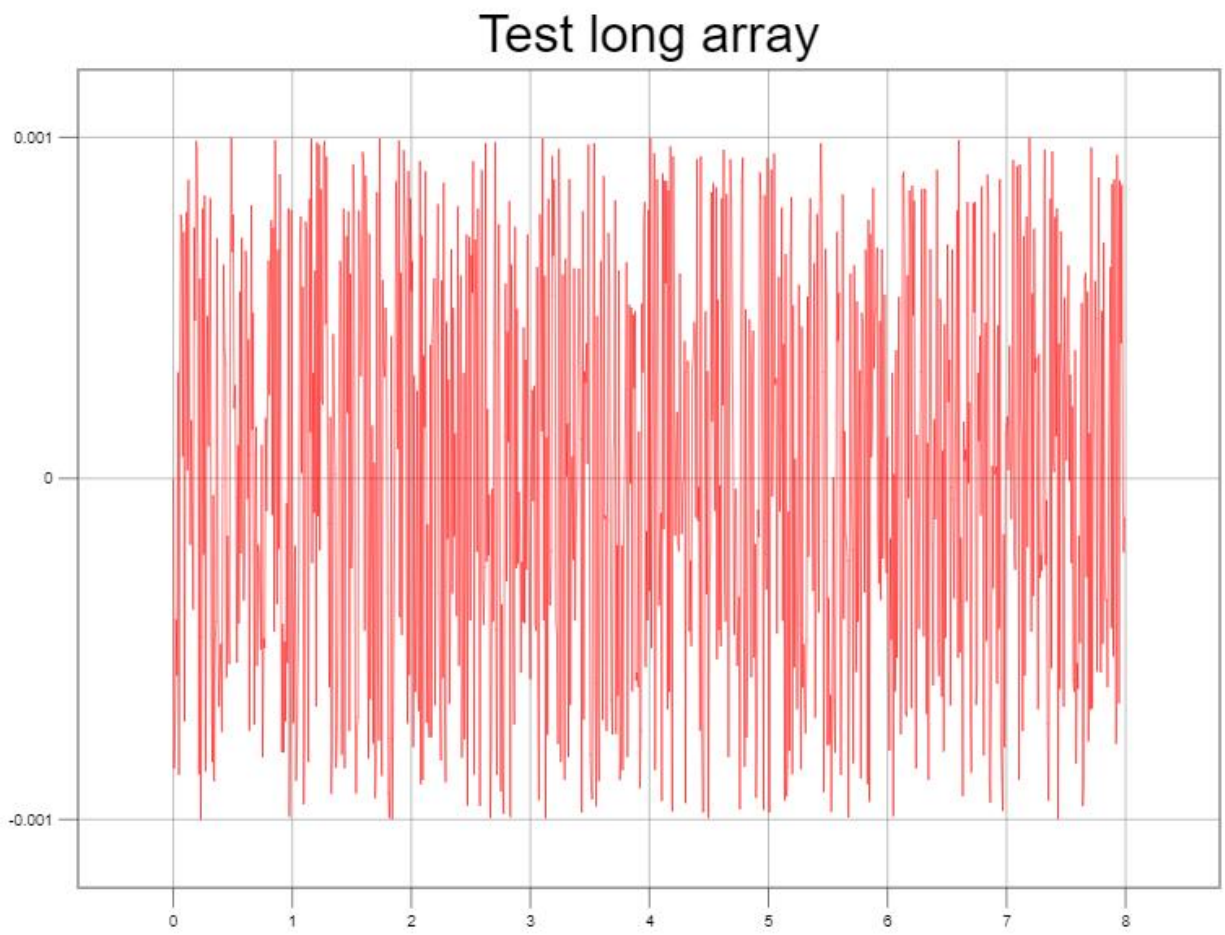
Рис. 14. Вывод листинга 6.



Листинг 8. Пример вывода большого массива данных PlotterJupyter.

```
uses PlotterJupyter;
begin
    var fig := new Figure();
    var ax := fig.AddSubplot();
    var x := new real[1000];
    var y := new real[1000];
    x[0] := 0; y[0] := 0;
    for var i:= 1 to 999 do
    begin
    y[i] := Random(0.001,-0.001);
    x[i] := x[i-1]+0.008;
    end;
    ax.Plot(x,y).linewidth := 0.2;
    ax.grid := true;
    ax.Title := 'Test long array';
    ax.TrackMouse := false;
    Show(fig);
end.
```

Рис. 15. Вывод листинга 7.



Листинг 9. Пример графиков разных размеров PlotterJupyter.

```
uses PlotterJupyter;
begin
    var fig := new Figure();
    var ax := fig.AddSubplot();
    var x := new real[1000];
    var y := new real[1000];
    x[0] := 0; y[0] := 0;
    for var i:= 1 to 999 do
    begin
    y[i] := Random(0.001,-0.001);
    x[i] := x[i-1]+0.008;
    end;
    ax.Plot(x,y).linewidth := 0.2;
    ax.grid := true;
    ax.Title := 'Test long array';
    ax.TrackMouse := false;
    Show(fig);
end.
```

Рис. 16. Вывод листинга 8.