МИНОБРНАУКИ РОССИИ
ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕЪХНОЛОГИЙ
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ОТЧЁТ
ПО УЧЕБНОЙ ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ

Студент
гр. БИН-22-2 _____ А.Д. Петроченко

Руководитель
старший преподаватель каф. ИТС _____ Е.Г. Лаврушина

Владивосток 2024

Содержание

Введение ... 3
1 Анализ технического задания ... 4
1.1 Кривая Штейнера: основные сведения и область применения ... 4
1.2 Математический аппарат для построения графика ... 4
1.3 Проектирование экранной формы программы ... 6
1.4 Построение блок-схемы работы программы ... 6
2 Определение интструментария ... 8
2.1 Языки программирования ... 8
2.2 Дополнительные библиотеки ... 9
2.3 Среды разработки ... 10
3 Разработка программы ... 12
3.1 Диаграмма классов ... 12
3.2 Реализация графического интерфейса программы ... 14
3.3 Реализация построения кривой ... 17
3.4 Реализация движения примитива ... 19
4 Тестирование программы ... 20
Заключение ... 22
Список использованных источников ... 23
Приложениие А ... 25
Приложениие Б ... 26
Приложениие В ... 27
Приложение Г ... 28

Введение

Целями прохождения практики является закрепление и углубление знаний, полученных в ходе теоретического обучения, подготовка обучающихся к изучению последующих дисциплин и приобретение ими практических навыков и компетенций в сфере профессиональной деятельности. Основной целью прохождения практики является закрепление и углубление знаний в области алгоритмизации и программирования, полученных в ходе обучения.

Задачи прохождения практики:

– изучение истории появления и сферу применения кривой Штейнера;

– сбор и анализ информации по теме задания;

– создание блок схемы алгоритма построения кривой Штейнера;

– написать программу в соответствии с составленной блок-схемой на выбранном языке программирования и произвести отладку.

Результаты работы подробно изложены в отчёте.

1 Анализ технического задания

1.1 Кривая Штейнера: основные сведения и область применения

Кривая Штейнера – плоская алгебраическая кривая 4-го порядка, которая описывается точкой окружности радиуса r, катящейся по окружности радиуса R=3r и имеющей с ней внутреннее касание; гипоциклоида с модулем m=3 [1]. Кривая Штейнера в общем виде представлена на рисунке 1.

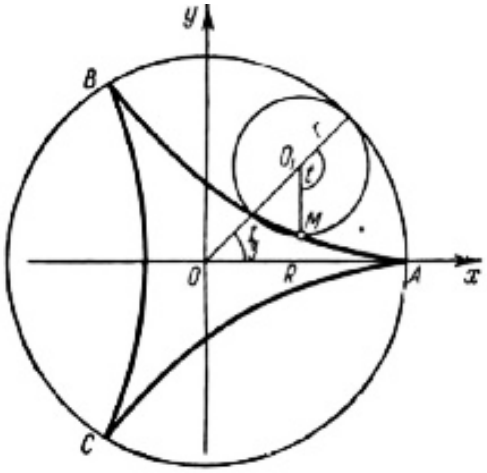

Рисунок 1 – Общий вид кривой Штенера

Кривая Штейнера может быть рассмотрена, как частный случай гипоциклоиды, получаемый в том случае, если радиус катящегося круга в три раза меньше радиуса неподвижного круга [2]. Впервые упоминается Л. Эйлером в 1745 г., а затем была подробно изучена и описана Я. Штейнером в 1856 г.

В математике кривая Штейнера используется для решения задачи об иголке. Множество комплексных собственных значений унистохастических матриц третьего порядка также образует дельтоиду.

1.2 Математический аппарат для построения графика

Параметрические уравнения кривой Штейнера выглядят следующим образом:

$$\begin{cases} x = 2r \cdot cos(\theta) + r \cdot \cos(2\theta), \\ y = 2r \cdot sin(\theta) - r \cdot \sin(2\theta), \end{cases} \quad (1)$$

где r – радиус катящейся окружности;

θ – угол, на который прямая отклоняется от полярной оси, $\theta \in [0\pi; 2\pi]$.

Исключив параметр θ, можно получить уравнение в декартовых прямоугольных координатах:

$$(x^2 + y^2)^2 + 8rx(3y^2 - x^2) + 18r^2(x^2 + y^2) - 27r^4 = 0 \quad (2)$$

Применим формулу (3) для перевода из декартовой системы координат в полярную через параметрические уравнения [3].

$$\begin{cases} \rho = \sqrt{x^2 + y^2} \\ \begin{cases} \varphi = \arctan\left(\frac{y}{x}\right), & x \neq 0, \\ \varphi = \frac{\pi}{2}, & x = 0, \quad y > 0, \\ \varphi = -\frac{\pi}{2}, & x = 0, \quad y < 0 \end{cases} \end{cases} \quad (3)$$

Получим уравнение кривой Штейнера в полярных координатах:

$$\begin{cases} \rho = r\sqrt{5 + 4\cos(3\theta)}, \\ \begin{cases} \varphi = \arctan\left(\frac{2\sin(\theta) - \sin(\theta)}{2\cos(\theta) + \cos(\theta)}\right), & x \neq 0, \\ \varphi = \frac{\pi}{2}, & x = 0, \quad y > 0, \\ \varphi = -\frac{\pi}{2}, & x = 0, \quad y < 0, \end{cases} \end{cases} \quad (4)$$

где r – радиус катящейся окружности;

θ – угол, на который прямая отклоняется от полярной оси, $\theta \in [0\pi; 2\pi]$.

Формулы (1) и (4) понадобятся для решения задачи.

Также для отрисовки примитива, движущегося по кривой, который по заданию должен представлять собой круг, понадобятся уравнение окружности с центром в начале координат в декартовой (5) и полярной (6) системах координат, а также уравнение окружности с центром в некоторой точке в декартовой (7) и полярной (8) системах координат.

$$\begin{cases} x = r * \cos(\theta), \\ y = r * \sin(\theta), \end{cases} \quad (5)$$

$$\begin{cases} \rho = r, \\ \varphi = \theta, \end{cases} \quad (6)$$

$$\begin{cases} x = r * \cos(\theta) + a, \\ y = r * \sin(\theta) + b, \end{cases} \quad (7)$$

$$\begin{cases} \rho = (r * \cos(\theta) + a)^2 + (r * cos(\theta) + b)^2, \\ \varphi = \frac{(r * \sin(\theta) + b)^2}{(r * \cos(\theta) + a)^2} \quad , \end{cases} \quad (8)$$

где r – радиус катящейся окружности;

θ – угол, на который прямая отклоняется от полярной оси, $\theta \in [0\pi; 2\pi]$;

(a; b) – координаты центра круга.

1.3 Проектирование экранной формы программы

Перед тем как перейти к алгоритмизации следует сперва продумать, что именно будет делать программа, и как ею будет управлять пользователь. Исходя из условия задания, взаимодействие пользователя с программой должно происходить при помощи экранной формы.

Программа должна отображать на экране монитора график кривой Штейнера в декартовых и полярных координатах. Пользователю должна быть предоставлены следующие возможности:

– задавать параметр кривой, а именно радиус неподвижной окружности, радиус образующей окружности должен рассчитываться автоматически и выводиться пользователю;

– выбирать систему координат, в которой нужно вывести график;

– задать команду вывести график;

– запустить и остановить движение примитива по кривой;

– масштабировать график и прокручивать изображение по вертикали и горизонтали.

Для осуществления этих действий, на экранной форме необходимы соответствующие управляющие элементы:

– поле ввода данных;

– поле вывода данных;

– подписи полей;

– группа переключателей с возможностью выбора только одного варианта;

– нажимающаяся кнопка;

– чекбокс;

– пространство для отображения графики «холст»;

– полосы вертикальной и горизонтальной прокрутки;

– ползунок.

Макет экранной формы, демонстрирующий планируемое расположение элементов приведен в приложении А.

1.4 Построение блок-схемы работы программы

Следующим этапом выполнения задания является построение блок-схемы алгоритма будущей программы.

После старта программы первым делом происходит инициализация и вывод окна-формой. Параллельно с этим создается в памяти программы создается объект класса, в котором в

дальнейшем будут храниться все необходимые данные о кривой. По умолчанию во время создания должны устанавливаться параметры кривой по умолчанию.

Далее пользователь может взаимодейтсоовать окном:

1. При взаимодействии с полем ввода радиуса автоматически изменяется соответствующие параметры R и r, значение параметра отображается в поле вывода.
2. При взаимодействии с переключателями «холст» переключается в режим для отрисовки в выбранной системе координат.
3. При нажатии на кнопку «Построить» происходит вычисление координат точек в выбранной системе, после чего происходит анимированная отрисовка графика.
4. При установке галочки в чек-боксе создается примитив и запускается его движение по кривой. При снятии галочки движение останавливается.
5. Пользователь также может при необходимости увеличивать или уменьшать график, а также перемещать видимую зону при помощи полос прокрутки и ползунка.

Работа программы завершается только тогда, когда пользователь нажимает на кнопку закрытия окна.

Блок-схема программы представлена в приложении Б.

2 Определение интструментария

2.1 Языки программирования

При выборе языка программирования разработки будущей программы необходимо учитывать множество факторов, но для решения поставленной задачи следует обратить особое внимание на следующие: среда, в которой будет функционировать программа, требовательность языка к ресурсам, является язык компилируемым или интерпретируемым, имеет ли язык уже готовые инструменты или решения, которые могут пригодиться для выполнения поставленных задач, поддержка объектно-ориентированного программирования. Также немаловажную роль играет степень владения языком исполнителем – студентом.

В процессе выбора рассматривались такие языки как C++, Java и Python, так как исполнитель имеет достаточно опыта работы с ними, все эти языки поддерживают объектно-ориентированное программирование и имеют несколько различных библиотек для работы с графикой.

Язык C++ является компилируемым объектно-ориентированным языком, часто используется для написания сложных программ, требующих производительности. Для C++ существует множество графических библиотек, позволяющих создавать графические интерфейсы любой сложности [4].

Язык Java – это кроссплатформенный язык с поддержкой объектно-ориентированного программирования. Работает на большом количестве операционных систем. Надёжный выбор для объектно-ориентированного программирования [5].

Язык Python – один из самых популярных и самых простых для изучения языков программирования. Является интерпретируемым языком, следовательно, может работать на любой платформе. Используется во множестве сфер, включая веб-разработку и игры [6].

После более тщательного анализа рассматриваемых языков были собраны аргументы за и против использования каждого языка, приведенные в таблице 1.

Таблица 1

Язык	Аргументы «за»	Аргументы «против»
Java	Прост в освоении.	Низкая производительность. Требует много памяти.
	Может работать практически на любой системе без проблем и сложных настроек. Не зависит от аппаратного обеспечения	Конструкторы графических интерфейсов ограничены, не подходят для создания сложных интерфейсов.
	Огромное множество образовательных материалов, в том числе и на русском языке.	Громоздкие синтаксические конструкции.

Продолжение таблицы 1

Язык	Аргументы «за»	Аргументы «против»
C++	Высокая производительность.	Установка и настройка дополнительных библиотек требует существенных усилий.
	Возможность контролировать память.	Сложный синтаксис. Может понадобиться много времени для освоения новых библиотек.
	Большой выбор графических библиотек. Возможность создавать сложные динамичные интерфейсы.	Для запуска программы обязательна сборка под каждую платформу отдельно, для чего также может потребоваться использование дополнительных библиотек.
Python	Может быть легко внедрен в веб-приложение.	Не очень хорошо приспособлен для ООП.
	Имеется множество библиотек для упрощения работы со сложными вычислениями: NumPy, SymPy и др.	Низкая скорость из-за того, что код интерпретируется в процессе.
	Имеется специальная готовая библиотека для работы с графиками – Matplotlib	Больше подвержен ошибкам. Требует больше тестов.
	Можно как запускать код, используя интерпретатор, так и преобразовать его в исполняемый файл любой операционной системы.	Невозможно запустить код, если не установлен интерпретатор Python и все необходимые библиотеки.

В конечном итоге был выбран язык Python так как, его сильные стороны значительно упрощают решение поставленной задачи, я выявленные недостатки не будут оказывать большое влияние на результат.

2.2 Дополнительные библиотеки

Помимо основных средств языка программирования Python для работы понадобятся некоторые специальные библиотеки. Большинство из них уже встроены в стандартные дистрибутивы Python, некоторые нужно устанавливать отдельно, но процесс установки занимает совсем немного времени.

Первое, что понадобится для решения задачи – это библиотека, содержащая в себе функции для математических вычислений. Для этого подойдёт библиотека NumPy – фундаментальный пакет для научных вычислений. Данная библиотека значительно упрощает создание и обработку различных массивов данных, включая матрицы и векторы, а также предлагает комплексные математические функции, генераторы чисел, процедуры линейной алгебры и

многое другое [7]. Библиотека распространяется под свободной лицензией и имеет открытый исходный код, а также хорошо взаимодействует с другими библиотеками, включая библиотеку Matplotlib.

Matplotlib – комплексная библиотека для создания статических, анимированных и интерактивных визуализаций на. Она позволяет легко создавать графики и диаграммы, разных видов, настраивать их визуальный стиль. Хорошо работает с многими другими графическими библиотеками, что позволяет интегрировать созданные визуализации в графический интерфейс [8].

Для создания графического интерфейса на существует множество инструментов. Самыми популярными из них являются Tkinter, PyQt и Eel.

Tkinter – бесплатная кроссплатформенная графическая библиотека, позволяющая создавать простые графические компоненты [9]. Преимуществами данной библиотеки является простая установка и интуитивная логика. Из недостатков можно отметить то, что механика расположения элементов может вызвать затруднения у начинающих программистов, особенно при попытке создать адаптивный интерфейс.

PyQt – библиотека, адаптирующая фреймворк Qt для взаимодействия с. Qt представляет кроссплатформенный фреймворк для создания графических приложений, предоставляет разработчикам отличный набор инструментов для проектирования и создания приложений, не беспокоясь о зависимости от платформы [10]. Фреймворк Qt работает с различными языками программирования, включая вышеупомянутый C++. Также при необходимости приложение, созданное при помощи Qt можно превратить в веб-приложение. Большим преимуществом данного фреймворка является наличие приложение-конструктора Qt Designer, который позволяет легко создавать интерфейсы любой сложности и очень прост для использования начинающими.

Именно PyQt был выбран для создания экранной формы будущей программы.

2.3 Среды разработки

Теперь, когда выбран язык и определены все необходимые библиотеки, время перейти к выбору среды разработки.

Среди множества вариантов рассмотрим те, которые уже установлены на рабочую машину исполнителя и были ранее использованы: Visual Studio, PyCharm, Jupyter Notebook.

Visual Studio — полнофункциональная IDE, позволяющая разрабатывать под различные платформы и поставляющаяся с собственным магазином расширений. Имеет бесплатную версию [11].

PyCharm –полнофункциональная среда разработки специально для языка Python. Под-

держивает управление версиями и проектами [12].

Jupyter Notebook – это веб-приложение с открытым исходным кодом, поддерживающее рабочие среды для несколько языков программирования, включая Python. Может запускаться как на компьютере, так и на популярных облачных сервисах [13].

После более тщательного анализа рассматриваемых языков были собраны аргументы за и против использования каждого языка, приведенные в таблице 2.

Таблица 2

Среда разработки	Аргументы «за»	Аргументы «против»
Visual Studio	Множество удобных настроек, облегчающих работу с проектом	Очень ресурсоёмкая программа.
PyCharm	Имеется отладчик и функция автозаполнения кода.	Медленно работает.
	Поддерживает работу с фреймворками	Официально недоступен в России.
Jupyter Notebook	Возможность запускать отдельные части кода выборочно, не перезапуская программу полностью.	Имеет ограничение на память, которую может использовать программа.
	Все выводимые данные остаются в документе перед глазами пользователя.	Документ специального расширения ipynb весит намного больше, чем простой файл с кодом.

В итоге для разработки на ранних этапах была выбрана среда Jupyter Notebook, так как в этой среде намного удобнее отлаживать отдельные функции, не тратя время на перезапуск тех частей кода, которые не изменяются в данным момент. Для работы на поздних этапах, включая этап тестирования больше подойдет среда PyCharm.

3 Разработка программы

3.1 Диаграмма классов

Так как одним из условий задачи является использование объектно-ориентированного программирования, весь функционал программы должен быть представлен в виде классов, взаимодействующих между собой. Прежде чем перейти непосредственно к написанию кода, следует сначала представить внутреннюю структуру программы. Для этого воспользуемся диаграммой классов. Она отражает в себе все классы, которые будут использованы в программе, их атрибуты и методы, а также отношения между классами. Диаграмма классов представлена в приложении В.

Класс Window отвечает за окно экранной формы программы и является наследником класса QWidget из библиотеки PyQt [14]. В качестве атрибутов класса выступают: deltoid – объект класса Deltoida, хранящий в себе кривую, которую будет отображать программа; canvas – объект класса Canvas, хранящий в себе элемент формы область отображения или «холст»; button – объект класса Button, хранящий в себе элемент формы кнопку «Построить»; RLabel – объект класса Label, хранящий в себе элемент формы подпись для поля ввода; RInput – объект класса SpinInput, хранящий в себе элемент формы поле ввода; rLabel – объект класса Label, хранящий в себе элемент формы подпись для поля вывода; rOutput – объект класса Output, хранящий в себе элемент формы поле вывода; system – объект класса RadioButtons, хранящий в себе элемент формы группу переключателей; startAnim – объект класса CheckBox, хранящий в себе элемент формы чекбокс; prim – объект класса Primitive, хранящий в себе движущийся примитив. Класс Window имеет следующие методы: init – непосредственно инициализация объекта, во время которого создается окно со всеми его элементами, а также создается и привязывается кривая; startPrimitive – метод, привязанный к чекбоксу и ответственный за состояние анимации примитива; changeRadValue – метод, привязанный к полю ввода, ответственный за действия, сопровождающие смену значения поля; build (canvas: Canvas) – метод, привязанный к кнопке, инициализирующий за построение кривой.

Класс Canvas – дочерний класс класса FigureCanvas из библиотеки Matplotlib, отвечающий за область отображения, на которой рисуется будущий график. В качестве атрибутов класса выступают: figure – объект класса Figure из библиотеки Matplotlib, представляющий собой контейнер для будущего графика [15]; ax – объект класса Axes из библиотеки Matplotlib [16]; sys – целое число 0 или 1, обозначающее систему координат, на которую сейчас настроен «холст»; hscroll – объект класса QScrollBar из библиотеки PyQt [17], привязанную к холсту горизонтальную полосу прокрутки; vscroll – объект класса QScrollBar из библиотеки PyQt, привязанную к холсту вертикальную полосу прокрутки; zslider – объект класса QSlider из библиотеки PyQt [18], привязанный к холсту ползунок.

Класс Canvas имеет следующие методы: init – инициализация объекта; setXYlimits – устанавливает пределы осей координат; decardCanvas – переводит холст в режим декартовых координат; polarCanvas – переводит холст в режим полярных координат; makeitSquare – устанавливает условным единицам осей одинаковый реальный размер; scrollX – отвечает за горизонтальную прокрутку холста; scrollY – отвечает за вертикальную прокрутку холста; zoom – отвечает за увеличение и уменьшение графика.

Класс Button– дочерний класс класса QPushButton из библиотеки PyQt [19], позволяет создать и настроить обычную нажимающуюся кнопку. Имеет только атрибуты класса-родителя, среди которых в процессе инициализации задается только атрибут text, представляющий собой текст, отображающийся на кнопке, и function – функцию, которая вызывается при нажатии кнопки. Может использовать все методы класса-родителя, среди которых переинициализирован только метод init, который должен включать не только базовую инициализацию, но и устанавливать связанную функцию и фиксированный размер.

Класс Label– дочерний класс класса QLabel из библиотеки PyQt [20], позволяет создать и настроить подпись поля. Имеет только атрибуты класса-родителя, среди которых в процессе инициализации задается только атрибут text, представляющий собой текст подписи. Может использовать все методы класса-родителя, среди которых переинициализирован только метод init, который включает базовую инициализацию и устанавливает фиксированный размер.

Класс SpinInput– дочерний класс класса QDoubleSpinBox из библиотеки PyQt [21], позволяет создать и настроить поле ввода со кнопками. Имеет только атрибуты класса-родителя, среди которых в процессе инициализации задается атрибут value, устанавливающий значение, и function – функция, вызывающаяся при взаимодействии с полем. Может использовать все методы класса-родителя, среди которых переинициализирован только метод init, который включает базовую инициализацию и устанавливает начальное значение, а также связанную функцию и фиксированный размер.

Класс Output– дочерний класс класса QTextBrowser из библиотеки PyQt [22], позволяет создать и настроить поле вывода. Имеет только атрибуты класса-родителя, среди которых в процессе инициализации задается атрибут default, устанавливающий выводимое значение. Может использовать все методы класса-родителя, среди которых переинициализирован только метод init, который включает базовую инициализацию и устанавливает начальное значение и фиксированный размер.

Класс RadioButtons – дочерний класс класса QButtonGroup из библиотеки PyQt [23], позволяет создать и настроить поле вывода. Имеет атрибуты класса-родителя, а также дополнительные атрибуты: массив buttons, хранящий в себе объекты класса QRadioButton [24], и целое число var, хранящее значение, присвоенное группе. Может использовать все методы

класса-родителя, среди которых переинициализирован только метод init, который включает базовую инициализацию группы, а также последовательную инициализацию и добавление в группу нескольких кнопок. Также имеет метод onClicked, выполняющий действие при взаимодействии.

Класс CheckBox – дочерний класс класса QCheckBox из библиотеки PyQt [25], позволяет создать и чекбокс. Имеет только атрибуты класса-родителя, среди которых в процессе инициализации задается только атрибут text, представляющий собой текст, отображающийся на кнопке, и function – функцию, которая вызывается при взаимодействии с чекбоксом. Может использовать все методы класса-родителя, среди которых переинициализирован только метод init, который включает базовую инициализацию и устанавливает фиксированный размер.

Класс Deltoida описывает кривую Штейнера и имеет следующие атрибуты R – целое число, обозначающее радиус неподвижной окружности; r – радиус движущейся окружности, образующей кривую; массивы x и y – массивы дробных чисел, хранящие координаты точек кривой в декартовых координатах; массивы phi и p – массивы дробных чисел, хранящие координаты точек кривой в полярных координатах; ani – объект класса FuncAnimation из библиотека Matplotlib, хранящий анимацию отрисовки графика. Класс имеет следующие методы: init – инициализация объекта; xy_deltoida – метод, осуществляющий расчёт декартовых координат всех точек и заполнение массивов; change_radius – – метод, осуществляющий изменение радиусов кривой переданным значением; draw_decard – метод, осуществляющий анимированную отрисовку графика в декартовых координатах; draw_polar – метод, осуществляющий анимированную отрисовку графика в полярных координатах.

Класс Primitive описывает примитив, движущийся по кривой (как по заданию он должен представлять собой круг). Класс имеет следующие атрибуты: curve – объект класса Deltoida, хранящий в себе кривую, по которой будет двигаться примитив; r – целое число, радиус примитива (круга); массивы x и y – массивы дробных чисел, хранящие координаты точек круга в декартовых координатах; массивы phi и p – массивы дробных чисел, хранящие координаты точек круга в полярных координатах; ani – объект класса FuncAnimation из библиотека Matplotlib, хранящий анимацию движения примитива. Класс имеет следующие методы: init – инициализация объекта; delete – удаление объекта; animate_decard – метод, отвечающий за создание и запуск анимации в декартовой системе координат; animate_polar – метод, отвечающий за создание и запуск анимации в полярной системе координат.

3.2 Реализация графического интерфейса программы

Теперь можно приступить к написанию программы. Для этого был создан новый файл в среде Jupyter Notebook. Первым действием были импортированы в программу все необходи-

мые компоненты из библиотек PyQt5 и Matplotlib. Строки кода, отвечающие за импорт приведены на рисунке Г.1 в приложении Г.

Был создан класс Window и прописан в нем конструктор __init__(), в которой вызывается конструктор родительского класса, после чего при помощи функции resize() устанавливается базовый размер. При помощи функции setWindowTitle() устанавливается заголовок окна – «Кривая Штейнера». Ниже прописывается основной код для запуска программы. Код приведен на рисунке 2.

```
# driver code
if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = Window()
    window.show()
    try:
        sys.exit(app.exec_())
    except SystemExit as e:
        print(f"Program shutdown")
```

Рисунок 2 – Листинг запуска программы

После запуска программы выводится пустое окно.

Далее нужно было создать элементы. Был создан класс Button, прописыван конструктор __init__(), принимающий помимо создаваемого объекта два аргумента: title и function – заголовок кнопки и связанная с ней функция. В конструкторе вызывается конструктор родительского класса, ему передаётся значение title, устанавливается фиксированный размер элемента, и привязывается функция при помощи команды self.clicked.connect(function). В классе Window и вызывается конструктор класса Button, передаётся ему в качестве заголовка «Построить», а в качестве функции временно пишется заглушка, записывается созданный объект в атрибут button. Аналогичным образом создаются остальные классы элементов формы и их конструкторы, с указанием классо-родителей в соответствии с диаграммой классов.

Конструктор класса SpinInput должен принимать аргументы default и function – значение в поле ввода по умолчанию и связанная функция. Привязывается функция при помощи команды self.valueChanged.connect(function), чтобы функция выполнялась только тогда, когда значение в поле изменится.

Конструктор класса Output должен также принимать значение по умолчанию.

Конструктор класса Label должен принимать только текст подписи.

Конструктор класса RadioButtons принимает массив заголовков и значение для группы кнопок по умолчанию, создает некоторое количество переключателей, присваивая им соответственно заголовки, добавляет переключатели в группу. Значение по умолчанию записывается

в атрибут var. Кнопка, соответствующая значению по умолчанию, переводится в активное состояние. В этом же классе сразу прописывается метод onClicked(), который будет изменять значение атрибута var на номер соответствующей кнопки, и добавляется команда привязки в конструктор.

Конструктор класса CheckBox практически идентичен конструктору класса Button, за исключением того, что функция привязывается через команду self.stateChanged.connect(function), что означает, что привязанная функция будет вызываться при изменении состояния чекбокса.

Класс Canvas имеет следующий порядок инициализации: создаётся контейнер для будущей фигуры, внутри него создаётся Axes, устанавливается значение атрибута sys равное значению глобальной переменной, вызовом конструктора класса-родителя создаётся сам холст, после чего также создаются горизонтальная и вертикальная полосы прокрутки и бегунок.

После того, как все классы элементов интерфейса и их конструкторы прописаны, можно приступить к добавлению элементов в форму. По очереди вызываются конструкторы классов, им передаются необходимые дынные и созданные элементы сохраняются в соответствующих атрибутах класса Window: canvas – область отображения; button – кнопка «Построить»; RLabel –подпись поля ввода; RInput – поле ввода; rLabel – подпись поля вывода; rOutput – поле вывода; system – группа переключателей системы координат; startAnim – чекбок «Запустить анимацию». Там, где требуется передать конструктору функцию, вместо неё передаётся конструкция-заглушка, так как необходимых функций пока еще нет.

Полный листинг классов Button, SpinInput,Output, Label, RadioButtons, CheckBox и Canvas представлены на рисунках Г.2 – Г.8 в приложении Г.

После этого следует приступить к их компоновке. Для этого используется такой инструмент, как менеджер геометрии класс QLayout и его дочерние классы QHLayout и QVLayout, которые позволяют размещать элементы друг за другом горизонтально и вертикально соответственно. Исходя из макета (см. приложение А), элементы экранной формы можно разделить на 2 части: в левой части располагаются кнопки, поля ввода и вывода и прочие управляющие элементы, а с в правой располагается область отображения и привязанные к ней полосы прокрутки и бегунок.

Создаются два контейнера QVLayout с именами layoutL и layoutR, обозначающие левую и правую части соответственно. В контейнер layoutL помещаются в следующем порядке элементы: подпись RLabel, поле RInput, подпись rLabel, поле rOutput, по очереди все переключатели из списка system.buttons, кнопка button и чекбокс startAnim. После этого к контейнеру

применяется функция layoutL.addStretch(), которая поможет заполнить оставшееся пространство, тем самым как бы выровняв элементы по верхней границе контейнера. Когда левая часть формы готова, можно приступить к правой части. Следует обратить внимание, что вертикальная полоса прокрутки и область отображения должны располагаться горизонтально относительно друг друга. Для того, чтобы реализовать это в форме создаётся контейнер QHLayout с именем layoutRin и в него помещается полоса прокрутки canvas.vscroll и область canvas. Затем в контейнер layoutR помещается layoutRin, полоса прокрутки canvas.hscroll и бегунок canvas.zslider. Правая часть формы готова.

Последним действием создаётся еще один контейнер QHLayout и добавляется в него сначала левый, потом правый контейнеры. Полученный объект устанавливается в качестве макета окна. После запуска программы получается следующий результат (рисунок 3).

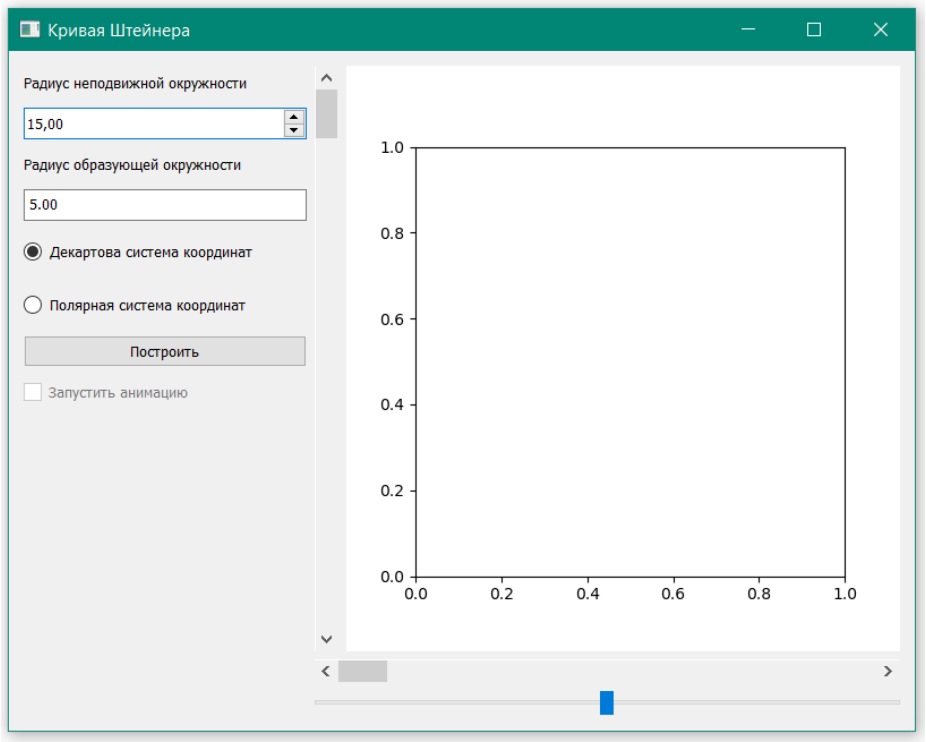

Рисунок 3 – Экранная форма

При открытии окна на весь экран изменяется только размер элементов, расположенных в правой части, размер элементов, расположенных слева остается фиксированным. Полный листинг класса Window представлен на рисунке Г.9 в приложении Г.

3.3 Реализация построения кривой

После создания формы следует перейти к реализации функционала программы. Создаётся класс Deltoida, в нём прописывается конструктор, который устанавливает атрибуту R значение глобальной переменной, рассчитывает и устанавливает атрибуту r значение одной трети от R, создает массивы x и y, phi и p.

Далее в классе прописывается метод xy_deltoida(), рассчитывающий координаты точек кривой. При помощи функции библиотеки NumPy arrange создаётся массив значений от 0 до 2π с шагом 0,01. Прописываются на языке программирования параметрические уравнения кривой в декартовых координатах (см. формулу (1)) и подставляется в них вместо параметра θ созданный массив. Благодаря функциям библиотеки NumPy все значения массива автоматически по очереди подставляться в формулу и выполниться вычисление. Полученные также в виде массива результаты записываются в качестве значения атрибутов объекта x и y.

Теперь нужно сделать так, чтобы график строился при нажатии на кнопку и выводился в области отображения экранной формы. Для этого дописывается вызов конструктора класса Deltoida в инициализацию окна, сохраняя созданный объект в атрибут deltoid. Прописывается в классе Window метод build(), в котором выполняются действия согласно блок-схеме алгоритма с некоторыми дополнениями, а именно:

- вызывается метод deltoid.xy_deltoida();
- проверяется, какая система координат в данный момент установлена;
- область отображения переводится в установленную систему координат;
- вызывается функция для отрисовки графика в нужной системе.

Прописывается функция отрисовки draw_decard в классе Deltoida. В качестве аргумента ей передаётся область отображения canvas, на которую будет выводиться график. Вызывается функция plot() для объекта canvas.ax, ей передаются атрибуты x и y. Вызовом специальных функций добавляются название диаграммы, подписи осей и сетка, устанавливаются пределы осей координат, представляющие собой ближайшее число, кратное 10, которое больше текущего значения R,.положительно и отрицательное. Вызывается метод canvas.makeItSquare() для выравнивания пропорций графика. Для наглядности на также была добавлена окружность радиуса R, выведенная пунктирной линией. Для её отрисовки были выполнены те же действия, что и для отрисовки кривой, только для расчёта координат была использована формула (5).

На этом этапе необходимо вернуться к функции инициализации окна и заменить в вызове конструктора кнопки передаваемую функцию с заглушки на функцию build(). Запустив код, оставим переключатель в позиции декартовой системы и введенные в поля значения по умолчанию и нажмём кнопку «Построить». График появился в области отображения.

Далее была прописана функция для отрисовки графика в полярных координатах draw_polar, в которой также было дано название диаграммы, установлены пределы оси y (пределы оси x в полярной системе должны оставаться неизменными). Для расчёта координат была использована формула (4), записанная на языке программирования.

Следующий этап – добавление анимации отрисовки графика. Для этого используется функция FuncAnimtion, в качестве аргументов которой передаётся canvas.figure, функция об-

новления кадра, а также некоторые другие параметры: blit – параметр, отвечающий за воспроизведение анимации, устанавливается значение True: interval – параметр, отвечающий за интервал между сменой кадров, влияет на скорость анимации, устанавливается значение 3; frames – массив, состоящий из целых чисел, количество которых должно совпадать с количеством кадров, в данном случае должен содержать числа от 0 до 628; repeat – параметр, отвечающий за циклический повтор анимации, устанавливается значение False. Функция смены кадров должна добавлять в график по одной точке за кадр и возвращать каждый раз обновлённую линию графика.

Полный листинг класса Deltoida представлен на рисунке Г.10 в приложении Г.

3.4 Реализация движения примитива

Реализация движения примитива в целом не сильно отличается от реализации отрисовки графика. Первым делом необходимо создать класс Primitive и его конструктор, которому будет передаваться кривая и её радиус. В процессе инициализации устанавливается радиус примитива, рассчитанный как одна девятая от переданного радиуса, устанавливается связь с кривой и создаются пустые массивы для хранения координат.

В классе Window прописывается функция startPrimitive, которая привязывается к чекбоксу. При установке галочки функция создаёт объект класса Primitive и сохраняет его в атрибут Window.prim. Если установлена декартовая система, вызывается функция animate_decard, если полярная, то animate_polar. При снятии галочки вызывается функция prim.delete, которая останавливает анимацию и удаляет объект.

Функция animate_decard прописывается в классе Primitive, и принимает на вход только область отображения canvas. Для посторения круга в декартовой системе координат можно использовать функцию plt.Circle, которой необходимо передать координаты круга, его радиус и цвет. Далее необходимо вызвать функци. Func animation и передать ей canvas.figure, функцию обновления кадра, frames – массив целых числе от 0 до 628, interval –3; repeat – True, blit – True. Функция обновления должна брать точку из массива точек связанной кривой и устанавливать её координаты в качестве координат центра круга и возвращать обновленный круг.

Функция animate_polar имет немного иной алгоритм. Так как функция plt.Circle не предназначена для полярных координат, чтобы изобразить круг пришлось использовать другие инструменты. Функция обновления должна рассчитывать координаты точек по формуле (8) и заполнять пространство внутри окружности при помощи функции plt.fill_between и возвращать это самое заполнение.

Полный листинг класса Primitive представлен на рисунке Г.11 в приложении Г.

4 Тестирование программы

Последним завершающим этапом работы является тестирование программы. В процессе тестирования необходимо проверить, как программа реагирует на различные действия пользователя, которые могут быть корректными или некорректными. Тестирование будет состоять и нескольких кейсов, каждый из которых будет содержать некоторый объем действий. В целом общий набор тестов должен покрывать большую часть всех возможных сценариев поведения пользователя при взаимодействии с программой.

Кейс 1.

Выполняемые действия: ввод корректного значения радиуса (неотрицательного числа), вывод последовательно кривой в декартовой системе, полярной системе, запуск движения примитива, остановка движения примитива.

Результат: Программа корректно выполнила все действия. Ошибок не выявлено.

Кейс 2.

Выполняемые действия: ввод некорректного значения радиуса (отрицательного числа), ввод в качестве значения радиуса 0, вывод последовательно кривой в декартовой системе, полярной системе, запуск движения примитива, остановка движения примитива.

Результат: Программа не позволяет ввести отрицательное число. При указании в качестве радиуса 0, не отображает кривую ни в одной системе координат. Примитив также не запускается. При этом программа продолжает работать корректно. Ошибок не выявлено.

Кейс 3.

Выполняемые действия: попытка запустить анимацию примитива до того, как была построена кривая.

Результат: Программа выдаёт ошибку (рисунок 4).

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[16], line 49, in Window.startPrimitive(self)
     47 if self.startAnim.isChecked():
     48     self.prim = Primitive(self.deltoid.R, self.deltoid)
---> 49     if self.system.var == 0: self.prim.animate_decard(self.canvas)
     50     else: self.prim.animate_polar(self.canvas)
     51     self.canvas.show()

Cell In[11], line 17, in Primitive.animate_decard(self, canvas)
     16 def animate_decard(self, canvas):
---> 17     circle = plt.Circle((self.curve.x[0], self.curve.y[0]), self.r, fc='r')
     18     def update_primitive(i):
     19         circle.center = (5, 5)

IndexError: list index out of range
```

Рисунок 4 – Ошибка при выполнении кейса 2

Решение проблемы: сделать чекбокс недоступным для взаимодействия, пока не будет построена кривая.

Кейс 4.

Выполняемые действия: изменять размеры окна, масштабировать и перемещать график при запущенной анимации примитива.

Результат: при перемещении и масштабировании графика примитив ненадолго замирает, но после продолжает движение. Примитив меняет размеры вместе с остальными элементами интерфейса корректно. Выявлен небольшой недочёт, но не критическая ошибка. Устранения не требует.

Кейс 5.

Выполняемые действия: перестроить график, не отключая анимацию примитива.

Результат: программа выдаёт ошибку (рисунок 5).

```
File D:\Program Files\Anaconda\Lib\site-packages\matplotlib\animation.py:1138, in Animation._draw_next_frame(self, framedata,
lit)
   1134 def _draw_next_frame(self, framedata, blit):
   1135     # Breaks down the drawing of the next frame into steps of pre- and
   1136     # post- draw, as well as the drawing of the frame itself.
   1137     self._pre_draw(framedata, blit)
-> 1138     self._draw_frame(framedata)
   1139     self._post_draw(framedata, blit)

File D:\Program Files\Anaconda\Lib\site-packages\matplotlib\animation.py:1767, in FuncAnimation._draw_frame(self, framedata)
   1763     self._save_seq = self._save_seq[-self._save_count:]
   1765 # Call the func with framedata and args. If blitting is desired,
   1766 # func needs to return a sequence of any artists that were modified.
-> 1767 self._drawn_artists = self._func(framedata, *self._args)
   1769 if self._blit:
   1771     err = RuntimeError('The animation function must return a sequence '
   1772                        'of Artist objects.')

Cell In[11], line 22, in Primitive.animate_decard.<locals>.update_primitive(i)
     20 canvas.ax.add_patch(circle)
     21 x, y = circle.center
---> 22 x = self.curve.x[i]
     23 y = self.curve.y[i]
     24 circle.center = (self.curve.x[i], self.curve.y[i])

IndexError: list index out of range
```

Рисунок 5 – Ошибка при выполнении кейса 5

Решение проблемы: снимать галочку с чекбокса автоматически при запуске построения.

В результате проведенных тестов были выявлены и устранены недочёты. Программа полностью готова к эксплуатации.

Заключение

В ходе прохождения практики была разработана программа с визуальным интерфейсов в виде экранной формы, позволяющая, строить и отображать кривую Штейнера.

В процессе были закреплены навыки разработки на языке программирования Python, углублены знания в области объектно-ориентированной разработки, в частности объектно-ориентированное программирование на языке Python. Были закреплены навыки построения и использования различных диаграмм, таких как блок-схемы и диаграммы нотации UML.

Также был освоен навык разработки графического интерфейса на Python и работы с фреймворком Qt, освоен графический конструктор Qt Designer.

Все поставленные задачи были выполнены, цель достигнута. Приобретенные знания, умения и опыт являются очень полезными и пригодятся как на последующих этапах обучения, так и в будущей профессиональной деятельности.

Список использованных источников

1. Кривая Штейнера – Текст: электронный / Соколов Д. Д. // Большая российская энциклопедия – 2022. – 3 июня 2022. – URL: https://bigenc.ru/c/krivaia-shteinera-b566d0 (дата обращения: 17.06.2024).

2. Плоские кривые: Систематика, свойства, применения. Справочное руководство. / А.А. Савелов – Москва: Либроком, 2020. – 294 с. (дата обращения 17.06.2024)

3. Высшая математика: учебно-практическое пособие. 2-е изд., перераб. и доп. / Н. А. Болотина, Л. П. Харитонова, И. П. Руденок – Волгоград ВолгГАСУ, 2012 (дата обращения 18.06.2024)

4. C++. Введение / Раздел о языке программирования C++ // Metanit.com – URL: https://metanit.com/cpp/tutorial/1.1.php (дата обращения 18.06.2024)

5. Java. Введение / Раздел о языке программирования Java // Metanit.com – URL: https://metanit.com/java/tutorial/1.1.php (дата обращения 18.06.2024)

6. Python. Введение / Раздел о языке программирования Python // Metanit.com – URL: https://metanit.com/python/tutorial/1.1.php (дата обращения 19.06.2024)

7. Главная страница // NumPy.org – URL: https://numpy.org/ (дата обращения 22.06.2024)

8. Matplotlib – Visualization with Python //Matplotlib.org – URL: https://matplotlib.org/ (дата обращения 22.06.2024)

9. Python и Tkinter. Введение. Первая программа/ Руководство по Tkinter /Раздел о языке программирования Python // Metanit.com – URL: https://metanit.com/python/tkinter/1.1.php (дата обращения 22.06.2024)

10. Что такое Qt / Руководство по программированию фреймворка Qt и языка C++ / язык программирования C++ //– URL: https://metanit.com/cpp/qt/1.1.php (дата обращения: 22.06.2024)

11. Интегрированная среда разработки Python для Visual Studio / Возможности Visual Studio для разработки // Visual Studio 2022 IDE – URL: https://visualstudio.microsoft.com/ru/vs/features/python/ (дата обращения 24.06.2024)

12. Features / Pycharm IDE // JetBrains.com – URL: https://www.jetbrains.com/pycharm/features/ (дата обращения 24.06.2024)

13. Записная книжка Jupyter / Документация пользователя // Документация для ноутбука Jupyter– URL: https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

14. Class QWidget / PySide6.QtWidgets / Qt for Python // Qt Documentation – URL: https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/QWidget.html (дата обращения 25.06.2024)

15. Matplotlib.figure.Figure / Matplotlib.figure / API Reference // Matplotlib 3.9.1 documentation – URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.html#matplotlib.figure.Figure (дата обращения 25.06.2024)

16. Matplotlib.figure.Axes / Matplotlib.axes / API Reference // Matplotlib 3.9.1 documentation – URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.html#matplotlib.axes.Axes (дата обращения 25.06.2024)

17. QScrollBar / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QScrollBar.html (дата обращения 26.06.2024)

18. QSlider / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QSlider.html (дата обращения 27.06.2024)

19. QPushButton / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QPushButton.html (дата обращения 27.06.2024)

20. QLabel / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QLabel.html (дата обращения 25.06.2024)

21. QDoubleSpinBox / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QDoubleSpinBox.html (дата обращения 28.06.2024)

22. QTextBrowser / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QTextBrowser.html (дата обращения 28.06.2024)

23. QButtonGroup / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QButtonGroup.html (дата обращения 28.06.2024)

24. QRadioButton / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QRadioButton.html (дата обращения 29.06.2024)

25. QCheckBox / PySide2.QtWidgets / Qt Modules / Qt for Python Documentation / Qt for Python 5.15.11 // Qt Documentation – URL: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QCheckBox.html (дата обращения 29.06.2024)

Приложениие А

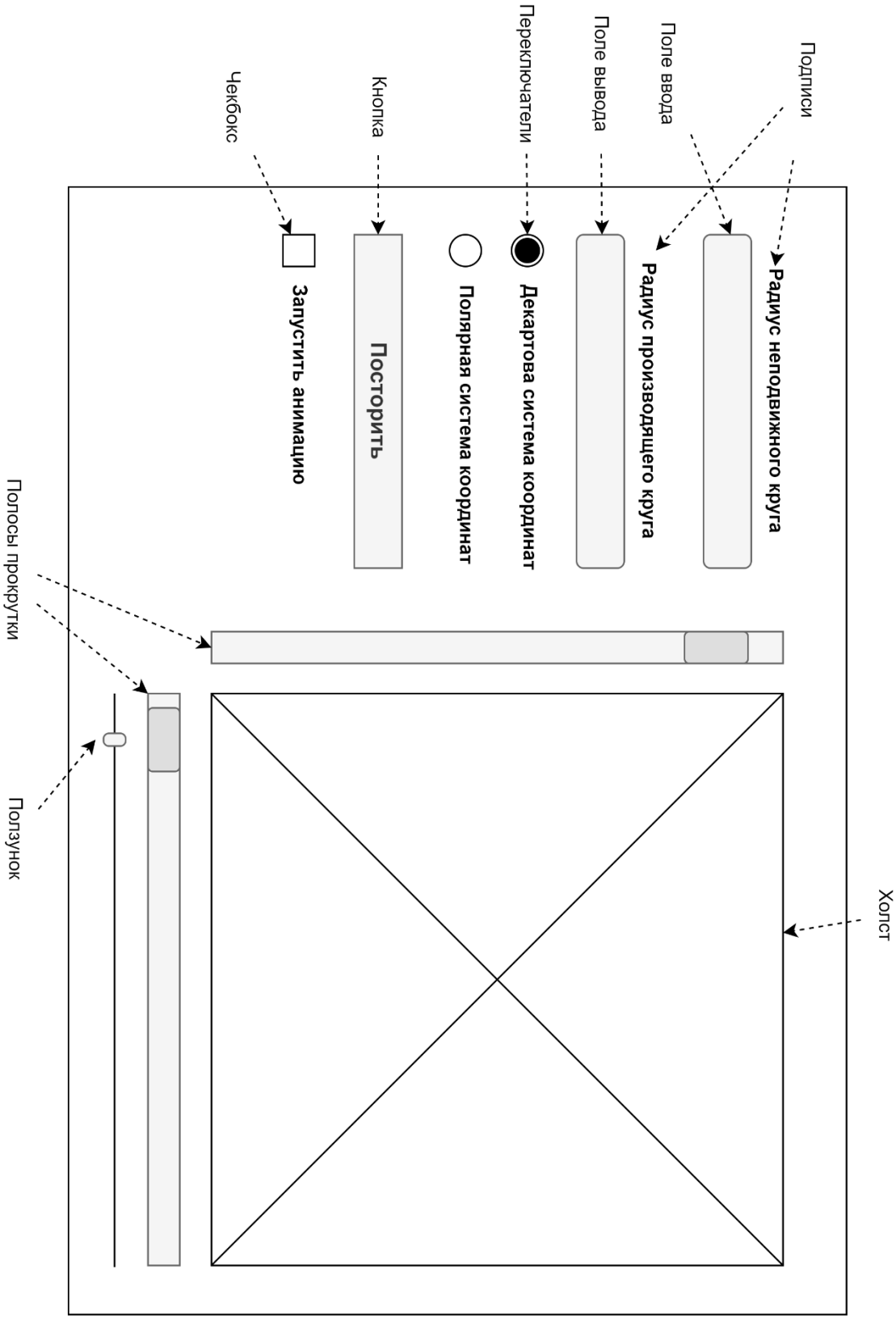

Рисунок А.1 – Макет экранной формы

Приложениие Б

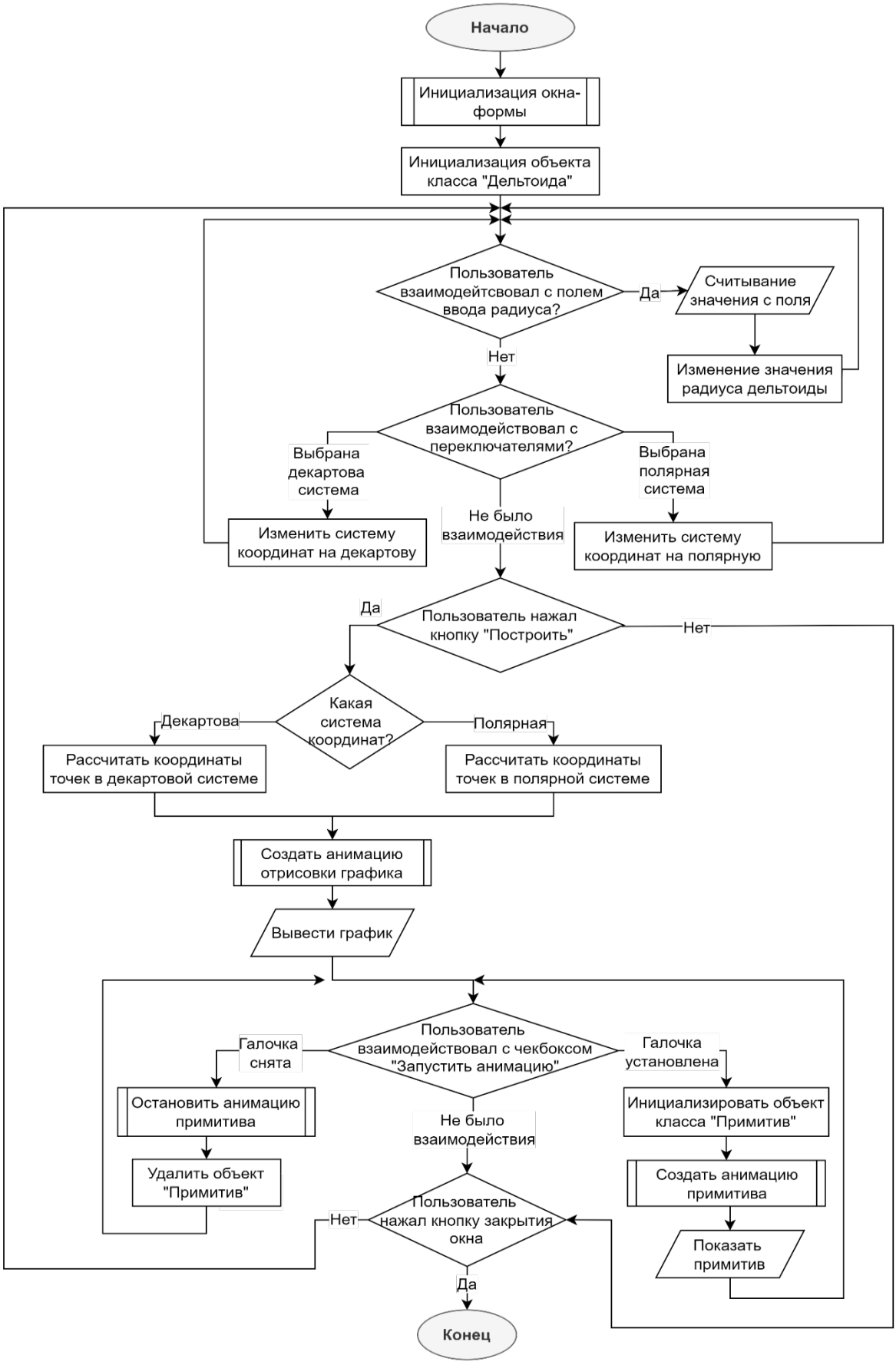

Рисунок Б.1 – Блок-схема алгоритма

Приложениие В

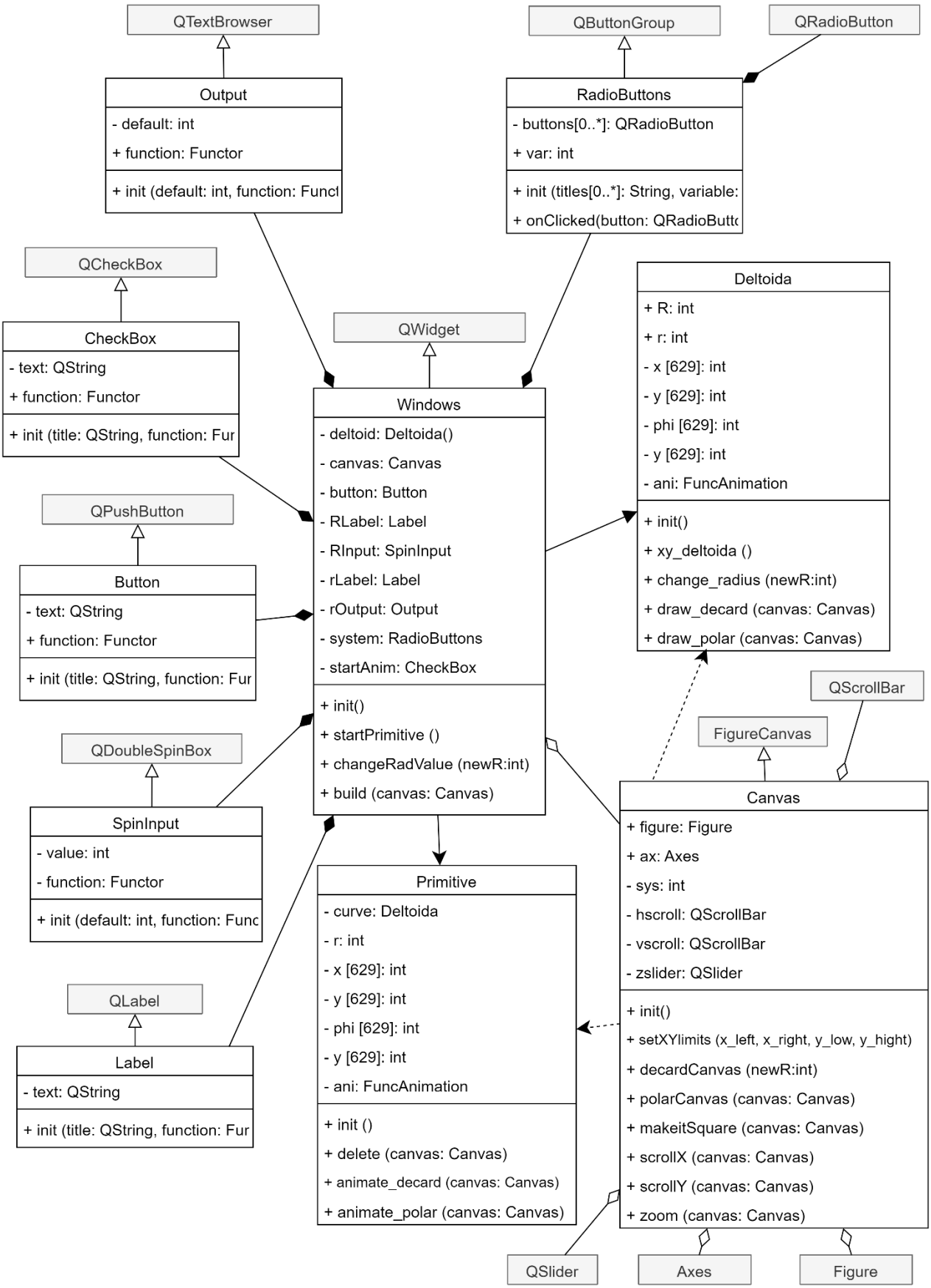

Рисунок В.1 – Диаграмма классов

Приложение Г

```
import sys
from PyQt5.QtWidgets import (QDialog, QApplication, QPushButton, QVBoxLayout, QHBoxLayout, QSizePolicy,
QWidget, QDoubleSpinBox, QTextBrowser, QLabel, QButtonGroup, QRadioButton, QCheckBox, QScrollBar, QSlider)
from PyQt5.QtCore import Qt

from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.animation import FuncAnimation
from matplotlib.patches import Circle
import matplotlib.pyplot as plt

import numpy as np
```

Рисунок Г.1 – Команды импорта компонентов

```
class Button(QPushButton):
    def __init__(self, title, function):
        super().__init__(title)
        self.clicked.connect(function)
        self.setFixedSize(250, 28)
```

Рисунок Г.2 – Листинг класса Button

```
class SpinInput(QDoubleSpinBox):
    def __init__(self, default, function):
        super().__init__()
        self.setValue(default)
        self.valueChanged.connect(function)
        self.setFixedSize(250, 28)
```

Рисунок Г.3 – Листинг класса SpinInput

```
class Output(QTextBrowser):
    def __init__(self, default):
        super().__init__()
        self.setText(default)
        self.setFixedSize(250, 28)
```

Рисунок Г.4 – Листинг класса Output

```
class Label(QLabel):
    def __init__(self, text):
        super().__init__(text)
        self.setFixedSize(250, 30)
```

Рисунок Г.5 – Листинг класса Label

```
class RadioButtons(QButtonGroup):
    def __init__(self, titles, variable):
        super().__init__()
        self.buttons = []
        self.var = variable
        for t in titles:
            b = QRadioButton(t)
            b.setFixedSize(250, 40)
            self.buttons.append(b)
            self.addButton(b)
        self.buttonClicked.connect(self.onClicked)
        self.buttons[self.var].setChecked(True)

    def onClicked(self, button):
        self.var = self.buttons.index(button)
```

Рисунок Г.6 – Листинг класса RadioButtons

```
class CheckBox(QCheckBox):
    def __init__(self, text, function):
        super().__init__(text=text)
        self.setFixedSize(250, 30)
        self.stateChanged.connect(function)
```

Рисунок Г.7 – Листинг класса CheckBox

```
class Canvas(FigureCanvas):
    def __init__(self):
        self.xlimits = (0, 1)
        self.ylimits = (0, 1)
        self.deltaX = self.xlimits[1]-self.xlimits[0]
        self.deltaY = self.ylimits[1]-self.ylimits[0]
        self.figure = plt.figure()
        self.ax = self.figure.add_subplot(111)
        self.sys = 0
        super().__init__(self.figure)
        self.makeItSquare()

        self.hscroll =  QScrollBar()
        self.hscroll.setOrientation(Qt.Horizontal)
        self.hscroll.valueChanged.connect(self.scrollX)

        self.vscroll =  QScrollBar()
        self.vscroll.setOrientation(Qt.Vertical)
        self.vscroll.valueChanged.connect(self.scrollY)

        self.zslider = QSlider(Qt.Orientation.Horizontal, self)
        self.zslider.setRange(-99, 99)
        self.zslider.setPageStep(1)
        self.zslider.valueChanged.connect(self.zoom)
        self.zslider.setValue(0)

    def setXYlimits(self, x_left, x_right, y_low, y_high):
        self.xlimits = (x_left, x_right)
        self.ylimits = (y_low, y_high)
        self.deltaX = self.xlimits[1]-self.xlimits[0]
        self.deltaY = self.ylimits[1]-self.ylimits[0]
        self.ax.set_xlim(x_left, x_right)
        self.ax.set_ylim(y_low, y_high)
        self.hscroll.setMaximum(int(x_right))
        self.hscroll.setMinimum(int(x_left))
        self.vscroll.setMaximum(int(y_high))
        self.vscroll.setMinimum(int(y_low))

    def decardCanvas(self):
        self.sys = 0
        self.ax = self.figure.add_subplot(111)

    def polarCanvas(self):
        self.sys = 1
        self.ax = self.figure.add_subplot(111, polar=True)

    def makeItSquare(self):
        ratio = 1.0
        x_left, x_right = self.ax.get_xlim()
        y_low, y_high = self.ax.get_ylim()
        self.ax.set_aspect( abs ((x_right-x_left)/(y_low-y_high))*ratio)

    def scrollX(self, value):
        self.ax.set_xlim(self.xlimits[0]+value, self.xlimits[1]+value)
        self.figure.canvas.draw()

    def scrollY(self, value):
        self.ax.set_ylim(self.ylimits[0]+value, self.ylimits[1]+value)
        self.figure.canvas.draw()

    def zoom(self, value):
        x_left, x_right = (-self.deltaX*((100-value)/100)/2, +self.deltaX*((100-value)/100)/2)
        y_low, y_high = (-self.deltaY*((100-value)/100)/2, +self.deltaY*((100-value)/100)/2)
        if self.sys == 0:
            self.ax.set_xlim(x_left, x_right)
            self.ax.set_ylim(y_low, y_high)
            self.xlimits = (x_left, x_right)
            self.ylimits = (y_low, y_high)
        else: self.ax.set_ylim(0, y_high-y_low)
        self.figure.canvas.draw()
```

Рисунок Г.8 – Листинг класса Canvas

```
class Window(QWidget):
    global system
    global R
    global r

    def __init__(self):
        super().__init__()
        self.resize(800, 600)
        self.setWindowTitle("Кривая Штейнера")
        self.deltoid = Deltoida()

        self.canvas = Canvas()
        self.button = Button("Построить", self.build)
        self.RLabel = Label("Радиус неподвижной окружности")
        self.RInput = SpinInput(self.deltoid.R, self.changeRadValue)
        self.rLabel = Label("Радиус образующей окружности")
        self.rOutput = Output("%.2f"%self.deltoid.r)
        self.system = RadioButtons(["Декартова система координат", "Полярная система координат"], system)
        self.startAnim = CheckBox("Запустить анимацию", self.startPrimitive)
        self.startAnim.setEnabled(False)

        layoutL = QVBoxLayout()

        layoutL.addWidget(self.RLabel)
        layoutL.addWidget(self.RInput)
        layoutL.addWidget(self.rLabel)
        layoutL.addWidget(self.rOutput)
        for b in self.system.buttons: layoutL.addWidget(b)
        layoutL.addWidget(self.button)
        layoutL.addWidget(self.startAnim)
        layoutL.addStretch()

        layoutR = QVBoxLayout()
        layoutRin = QHBoxLayout()
        layoutRin.addWidget(self.canvas.vscroll)
        layoutRin.addWidget(self.canvas)
        layoutR.addLayout(layoutRin)
        layoutR.addWidget(self.canvas.hscroll)
        layoutR.addWidget(self.canvas.zslider)

        layout = QHBoxLayout()
        layout.addLayout(layoutL)
        layout.addLayout(layoutR)
        self.setLayout(layout)

    def startPrimitive(self):
        if self.startAnim.isChecked():
            self.prim = Primitive(self.deltoid.R, self.deltoid)
            if self.system.var == 0: self.prim.animate_decard(self.canvas)
            else: self.prim.animate_polar(self.canvas)
            self.canvas.show()
        else:
            self.prim = self.prim.delete(self.canvas)

    def changeRadValue(self):
        self.deltoid.change_radius(self.RInput.value())
        self.rOutput.setText("%.2f"%self.deltoid.r)

    def build(self):
        self.startAnim.setCheckState(False)
        self.deltoid.xy_deltoida()
        global system
        self.canvas.figure.clear()
        if self.system.var == 0:
            self.canvas.decardCanvas()
            self.deltoid.draw_decard(self.canvas)
        elif self.system.var == 1:
            self.canvas.polarCanvas()
            self.deltoid.draw_polar(self.canvas)
        self.canvas.draw()
        self.startAnim.setEnabled(True)
```

Рисунок Г.9 – Листинг класса Window

```
class Deltoida:
    def __init__(self):
        global R
        self.R = R
        self.r = self.R/3
        self.x, self.y = [], []
        self.phi, self.p = [], []

    def xy_deltoida(self):
        x = []
        y = []
        t = np.arange(0 * np.pi, 2 * np.pi, 0.01)
        x.append(2*self.r * np.cos(t) + self.r*np.cos((2*t)))
        y.append(2*self.r * np.sin(t) - self.r*np.sin((2*t)))
        self.x = x[0]
        self.y = y[0]

    def change_radius(self, newR):
        self.R = newR
        self.r = self.R/3
        self.x, self.y = [], []

    def draw_decard(self, canvas):
        canvas.ax.grid()
        plt.xlabel("x")
        plt.ylabel("y")
        th = np.linspace( 0 , 2 * np.pi , 150 )
        a = self.R * np.cos( th)
        b = self.R * np.sin(th)
        canvas.ax.plot(a, b, linestyle = '--')
        plt.title("кривая Штенера")
        canvas.setXYlimits(-10-10*int(self.R/10), 10+10*int(self.R/10), -10-10*int(self.R/10), 10+10*int(self.R/10))
        canvas.makeItSquare()
        x, y = [], []
        line, = canvas.ax.plot(self.x, self.y, animated=True, lw=2)
        def update_line(i):
            y.append(self.y[i])
            x.append(self.x[i])
            line.set_ydata(y)
            line.set_xdata(x)
            return [line]
        self.ani = FuncAnimation(canvas.figure, update_line,
                                 blit=True, interval=3, frames = np.arange(0, len(self.x), 1), repeat=False)

    def draw_polar(self, canvas):
        plt.title("кривая Штенера")
        canvas.setXYlimits(0*np.pi, 2*np.pi, 0, 10+10*int(self.R/10))
        t = np.arange(0 * np.pi, 2 * np.pi, 0.01)
        b = np.full((len(t), 1), self.R)
        canvas.ax.plot(t, b, linestyle = '--')
        self.p = self.r*np.sqrt( 5+4*np.cos(3*t) )
        self.phi=np.arctan2(self.y, self.x)
        x, y = [], []
        line, = canvas.ax.plot(self.phi, self.p, animated=True, lw=2)
        def update_line(i):
            y.append(self.p[i])
            x.append(self.phi[i])
            line.set_ydata(y)
            line.set_xdata(x)
            return [line]
        self.ani = FuncAnimation(canvas.figure, update_line,
                                 blit=True, interval=3, frames = np.arange(0, len(self.x), 1), repeat=False)
```

Рисунок Г.10 – Листинг класса Deltoida

```
class Primitive:
    def __init__(self, R, curve):
        self.curve = curve
        self.r = R/9
        self.x, self.y = [], []
        self.phi, self.p = [], []

    def __del__(self):
        pass

    def delete(self, canvas):
        self.ani._stop()
        self.__del__()

    def animate_decard(self, canvas):
        circle = plt.Circle((self.curve.x[0], self.curve.y[0]), self.r, fc='r')
        def update_primitive(i):
            circle.center = (5, 5)
            canvas.ax.add_patch(circle)
            x, y = circle.center
            x = self.curve.x[i]
            y = self.curve.y[i]
            circle.center = (self.curve.x[i], self.curve.y[i])
            return circle,
        self.ani = FuncAnimation(canvas.figure, update_primitive,
                                 frames = np.arange(0, len(self.curve.x), 1), interval=3, repeat=True, blit=True)

    def animate_polar(self, canvas):
        t = np.arange(0 * np.pi, 2 * np.pi, 0.01)
        x0 = self.r*np.cos(t)
        y0 = self.r*np.sin(t)
        p = np.sqrt(x0**2+y0**2)
        phi = np.arctan2(y0, x0)
        def update_primitive(i):
            x = x0 + self.curve.x[i]
            y = y0 + self.curve.y[i]
            p = np.sqrt(x**2+y**2)
            phi = np.arctan2(y, x)
            nfill = plt.fill_between(phi, p, facecolor='red')
            return nfill,
        self.ani = FuncAnimation(canvas.figure, update_primitive,
                                 frames = np.arange(0, len(self.curve.phi), 1), interval=3, repeat=True, blit=True)
```

Рисунок Г.11 – Листинг класса Primitive