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

ОТЧЁТ
ПО УЧЕБНОЙ ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ

Студент
гр. БИН-22-02 _____ В. В. Савченко

Руководитель
старший преподаватель _____ Е. Г. Лаврушина

Владивосток 2024

МИНОБРНАУКИ РОССИИ
ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АНАДИЗА ДАННЫХ
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ЗАДАНИЕ

на учебную ознакомительную практику

Студенту гр. БИН-22-02 Савченко Владимиру Вячеславовичу

1 Тема работы: Разработка программы

2 Срок сдачи работы: 13 июля 2024 г.

3 Содержание задания

3.1 Решение практических задач

Задача 1.

Используя произвольные язык программирования и среду разработки создайте программу, которая отображает на экране монитора график кривой или поверхности (в соответствии с вариантом задания № 13, Кривые Безье третьего порядка) в декартовой и полярной системах координат с центром в центре экрана монитора (окна или иной прямоугольной области экрана). При изменении размеров окна, график и все его атрибуты (координатная сетка, метки на шкале, подписи и т.д.) должны автоматически масштабироваться.

Параметры уравнения кривой или поверхности должны вводиться в специально отведённые ячейки экранной формы программы.

Задача 2.

Используя результаты предыдущего задания создайте анимацию примитива, движущегося по траектории построенной кривой:

- для чётных вариантов в качестве примитива используется закрашенная окружность красного цвета радиуса $r > 2$;

- для нечётных вариантов в качестве примитива используется закрашенный квадрат синего цвета со стороной $a > 2$.

3.2 Оформление и защита отчета

Отчет по учебной исследовательской практике должен содержать: титульный лист; индивидуальное задание; содержание; цель и задачи; описание выполненных заданий (в соответствии

с методическим руководством и требованиями преподавателя); выводы и предложения; список использованных источников; графический материал (схемы, графики, технологические карты).

Календарный план-график работ прилагается к отчету отдельным листом.

Требования к оформлению отчетов.

Отчёт предоставляется в печатном виде с выполнением требований нормоконтроля и состоит из следующих разделов:

Введение. Во введении обосновывается цель и задачи прохождения практики.

В разделе 1 выполняется краткий обзор литературы по теме индивидуального задания, подбирается необходимый математический инструментарий (аппарат), производится его анализ и разрабатывается алгоритм решения задачи.

В разделе 2 осуществляется обоснование выбора среды реализации разработанного алгоритма.

В разделе 3 описывается процесс кодирования в выбранной среде и языке программирования и основные элементы управления создаваемым приложением.

В разделе 4 описывается процесс тестированию и отладки программного кода.

Раздел 5 (рекомендуемый) должен содержать краткое иллюстрированное руководство пользователя.

Заключение. В заключении обобщается изложенный в отчёте материал, делаются выводы.

Объем отчёта составляет 20-25 страниц.

Отчёт по практике оформляется в соответствии с «Требованиями к оформлению текстовой части выпускных квалификационных работ, курсовых работ (проектов), рефератов, контрольных работ, отчётов по практикам, лабораторным работам (СК-СТО-ТР04-1.005-2020)». Для оформления графического материала блок-схем, алгоритмов использовать ГОСТ 19.701-90 который распространяется на условные обозначения (символы) в схемах алгоритмов, программ, данных и систем и устанавливает правила выполнения схем, применяемых для отображения различных видов задач обработки данных и средств их решения.

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

Задание получил: _____ В. В. Савченко

Содержание

Введение

Учебная ознакомительная практика является важным этапом в процессе обучения студентов, предоставляя им возможность применить теоретические знания на практике и приобрести навыки разработки программного обеспечения. В рамках данной практики студентам предлагается создать программу, которая отображает на экране монитора график кривой или поверхности в декартовой и полярной системах координат с центром в центре экрана.

Целью данной работы является разработка программы, которая позволяет визуализировать графики в различных системах координат, обеспечивая при этом автоматическое масштабирование графика и всех его атрибутов при изменении размеров окна. Это включает в себя координатную сетку, метки на шкале, подписи и другие элементы, которые должны корректно отображаться независимо от размеров окна. Данная задача требует от студентов не только знаний в области программирования, но и понимания математических основ, необходимых для построения графиков.

Для реализации данной задачи используется произвольный язык программирования и среда разработки, что позволяет студентам выбрать наиболее удобные и знакомые инструменты для выполнения работы. В процессе разработки программы студенты сталкиваются с различными аспектами программирования, такими как обработка событий, создание графического интерфейса, математические вычисления и анимация. Это способствует углублению знаний студентов в области программирования и разработки программного обеспечения, а также развивает их навыки решения практических задач и работы с различными инструментами и технологиями.

1 Анализ технического задания

1.1 Кривая Безье третьего порядка: основные сведения и область применения

Кривая Безье третьего порядка, также известная как кубическая кривая Безье, является одной из наиболее часто используемых кривых в компьютерной графике и других областях, связанных с обработкой графической информации. Эти кривые создаются на основе четырёх контрольных точек, которые определяют форму и направление кривой. Кубическая кривая Безье обладает высокой степенью гибкости и позволяет моделировать сложные формы с плавными изгибами и переходами.

Контрольные точки кривой Безье играют ключевую роль в её построении. Первая и последняя точки определяют начало и конец кривой, тогда как две промежуточные точки оказывают влияние на её форму, создавая изгибы и отклонения от прямой линии. Несмотря на то, что кривая сама по себе не проходит через промежуточные контрольные точки, они задают направление и кривизну, что делает кривую Безье мощным инструментом для моделирования различных визуальных эффектов. Кривая Безье третьего порядка с опорными точками, соединёнными отрезками (рисунок 1).

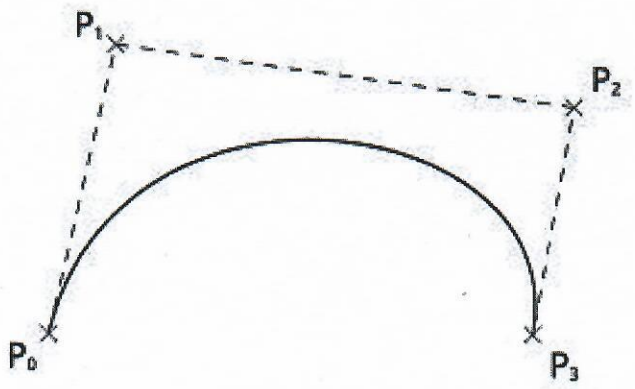


Рисунок 1 – Общий вид кривой Безье третьего порядка

История кривых Безье началась в 1962 году, когда французский инженер Пьер Безье, работая в компании Renault, искал способ более точно и гибко моделировать сложные формы автомобильных кузовов. До этого времени геометрические фигуры чаще всего описывались с помощью линейных участков и круговых дуг, что не всегда позволяло получить гладкие и естественные переходы.

Безье, используя свой опыт в геометрии и математике, разработал новый метод описания кривых, который позже стал известен как "кривые Безье". Его метод позволял создавать гладкие и контролируемые кривые с помощью нескольких контрольных точек, которые определяли форму кривой.

Открытие Безье было революционным в своей простоте и гибкости. Его метод оказался применимым не только к моделированию автомобильных кузовов, но и к многим другим областям, где требуется гладкое и контролируемое изменение формы объектов.

Сегодня кривые Безье являются одним из важнейших инструментов в компьютерной графике, графическом дизайне, анимации и других областях, где важно создавать гладкие и естественные формы. Их применяют для создания шрифтов, логотипов, иллюстраций, анимационных эффектов, моделирования трехмерных объектов и многого другого. История кривых Безье - яркий пример того, как практические задачи могут породить мощные математические инструменты, применяемые в самых разнообразных областях.

1.2 Математический аппарат для построения графика

В декартовой системе координат [3] Уравнение кривой Безье третьего порядка задаётся следующим способом (1):

$$B(t) = (1-t)^3 * P_0 + 3 * t(1-t)^2 * P_1 + 3 * t^2 * (1-t) * P_2 + t^3 * P_3, \quad (1)$$

где P_n – опорные точки, $t \in [0, 1]$ – параметр.

Для перевода координат из декартовой в полярную используются формулы (2) и (3):

$$\rho = \sqrt{(x^2 + y^2)}. \quad (2)$$

$$\theta = \arctan\left(\frac{x}{y}\right). \quad (3)$$

1.3 Блок-схема работы программы

Была реализована блок-схема работы программы (Приложение А).

2 Выбор языка программирования и среды разработки

2.1 Языки программирования

Для решения задачи требуется язык программирования, обладающий определёнными свойствами:

— язык должен предоставлять библиотеки для построения графиков, управления цветом, стилем линий и другими атрибутами графика;

— язык должен легко работать с математическими функциями, векторами и матрицами для вычисления координат точек на кривой и поверхности;

— язык должен позволять обрабатывать события, такие как изменение размеров окна для динамического масштабирования графика и ввод координат для обновления графика;

— язык должен быть достаточно простым для быстрого изучения и реализации проекта.

С учётом этих требований были рассмотрены следующие языки программирования:

а) Python:

Преимущества:

1) простота в изучении и использовании;
2) обширная экосистема библиотек и фреймворков, подходящих для различных задач, в том числе для работы с данными, машинного обучения, веб-разработки и визуализации;
3) высокая читаемость кода.

. Недостатки:

1) более медленная скорость работа по сравнению с некоторыми другими языками;
2) не подходит для задач, требующих высокой производительности и работы с низкоуровневыми ресурсами.

б) JavaScript:

Преимущества:

1) используется для создания интерактивных веб-страниц и веб-приложений;
2) большое количество фреймворков для веб-разработки.

. Недостатки:

1) сложнее использовать для задач, не связанных с веб разработкой.

в) C++

Преимущества:

1) высокая производительность и контроль за ресурсами;

. Недостатки:

1) сложный язык, требующий глубокого понимания;
2) требует больше времени на разработку.

В конечном счете, выбор Python был обусловлен:

— универсальностью, делающую его подходящим для анализа данных и разработки GUI интерфейсов;

— большим количеством ресурсов, документации и готовых решений;

— простотой использования.

2.2 Среды разработки

Для разработки приложения рассматривались следующие среды разработки:

а) PyCharm:

Преимущества:

1) предоставляет широкий набор функций, необходимых для разработки сложных приложений, включая интеллектуальное автозаполнение кода, отладку, рефакторинг, интеграцию с системами контроля версий;
2) предоставляет специальные плагины для работы с библиотеками для графической визуализации, таких как matplotlib, что делает разработку более удобной.

б) VS Code:

Преимущества:

1) является легкой и быстрой ide с широкими возможностями настройки т поддержкой разных языков программирования;
2) имеет большое количество расширений для работы с разными библиотеками и языками, в том числе Python, и для графической визуализации.

. Недостатки:

1) меньше встроенных функций по сравнению с аналогами.

д) Qt Creator:

Преимущества:

1) мощная IDE для разработки кроссплатформенных приложений с использованием PyQt.

Недостатки:

1) Могут потребоваться дополнительные настройки для работы с библиотеками визуализации, такими как matplotlib.

Выбор среды разработки зависел от конкретных потребностей проекта. PyCharm идеально подходит для сложных проектов, требующих широкого набора функций. VS Code является отличным выбором для более лёгких проектов. Qt Creator идеален для разработки GUI-приложений с использованием PyQt.

В итоге, для данного проекта был выбран PyCharm в связи с его удобством и наличием необходимых функций.

3 Разработка программы

3.1 Реализация графического интерфейса

Разработка графического интерфейса приложения была основана на использовании библиотеки tkinter для создания основного окна и matplotlib для отображения графика кривой Безье.

Приложение состоит из окна, созданного с использованием tkinter, в котором отображается график кривой Безье, реализованный с помощью matplotlib. Для отображения графика в окне используется класс FigureCanvasTkAgg.

Интерфейс предоставляет пользователю различные инструменты для взаимодействия с графиком кривой Безье. Пользователь может вводить координаты точек кривой в текстовые поля и нажимать кнопку "Обновить", чтобы пересчитать и отобразить новую кривую. Это позволяет гибко изменять форму кривой и наблюдать за изменениями в реальном времени.

Также интерфейс включает в себя кнопки для переключения между декартовой (рисунок 2) и полярной системами координат (рисунок 3). При переключении обновляются аннотации и координаты точек кривой, что позволяет пользователю лучше понимать геометрию кривой в разных системах координат.

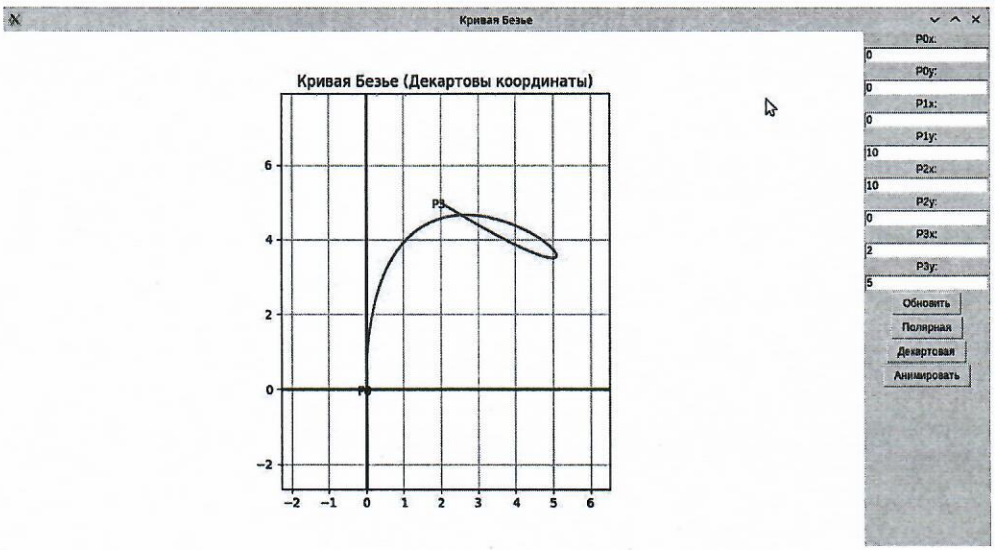


Рисунок 2 – Внешний вид интерфейса приложения с изображением декартовой системы координат

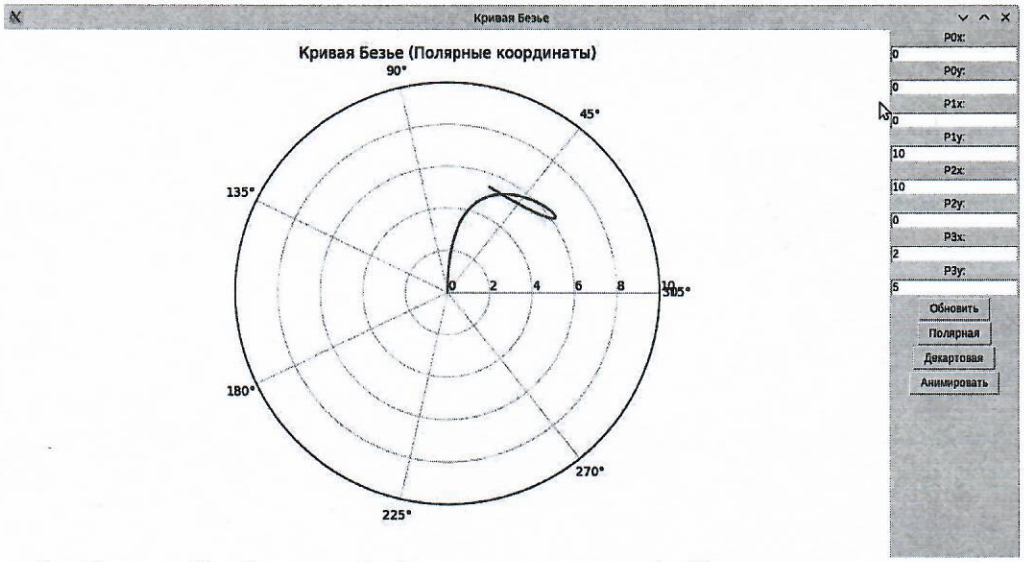


Рисунок 3 – Внешний вид интерфейса приложения с изображением полярной системы координат

В интерфейсе также предусмотрена кнопка для запуска анимации. Нажатие на эту кнопку запускает анимацию движения квадрата вдоль кривой Безье. Анимация позволяет визуализировать движение объекта по кривой, что может быть полезно для различных приложений, таких как моделирование движения или анимация. При повторном нажатии на кнопку анимация останавливается, что позволяет пользователю контролировать процесс анимации.

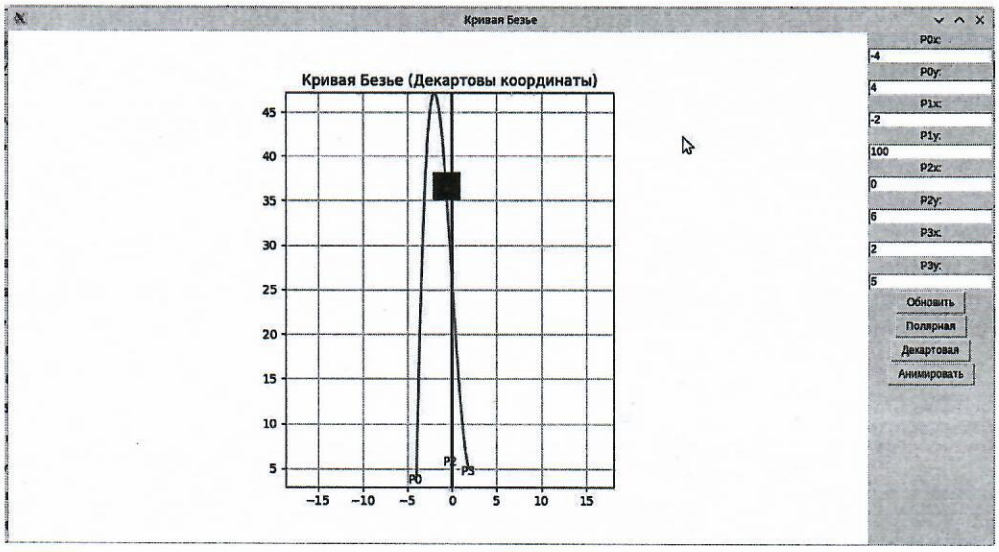


Рисунок 4 – Визуализация анимации в приложении

Разработанный графический интерфейс обеспечивает удобство использования программы, позволяя пользователю гибко настраивать параметры кривой Безье, выбирать систему координат и наблюдать за ее движением в режиме анимации. Все элементы управления расположены интуитивно понятно, что делает программу легкой в использовании даже для неподготовленного пользователя.

3.2 Реализация графического интерфейса

Программный код приложения, созданного для визуализации и анимации кривой Безье, представляет собой сложную систему, состоящую из нескольких ключевых частей, гармонично работающих вместе для достижения поставленной цели.

В самом начале программы происходит импорт необходимых библиотек: matplotlib, tkinter, numpy и math. Эти библиотеки являются фундаментом для всей работы приложения. Matplotlib предоставляет инструменты для создания графиков и визуализации данных. Tkinter используется для создания графического интерфейса приложения, обеспечивая взаимодействие пользователя с программой. Numpy - мощная библиотека для работы с многомерными массивами и матрицами, необходимая для выполнения математических операций. Наконец, math - стандартная библиотека Python, содержащая широкий набор математических функций и констант, необходимых для различных расчетов.

Далее код реализует функции для обработки различных событий, происходящих при взаимодействии пользователя с приложением. Эти функции делают приложение интерактивным и отзывчивым. Например, функция on_press запоминает координаты точки, на которую пользователь нажал мышкой, что позволяет в дальнейшем использовать эти координаты для перемещения графика или добавления аннотации. Функция on_motion следит за движением мыши и перемещает график в соответствии с ее траекторией, обеспечивая гладкое и удобное управление. Другие функции отвечают за масштабирование графика при прокрутке колеса мыши и за обработку нажатий клавиш, что позволяет пользователю изменять вид графика и управлять различными параметрами приложения.

Особую роль играют функции, отвечающие за работу с кривой Безье. Эти функции обеспечивают точные расчеты координат точек кривой, позволяя построить ее на графике с максимальной точностью. Функция bezier_curve вычисляет координаты точек кривой Безье для заданных значений параметра t, который меняется от 0 до 1 и определяет позицию точки на кривой. Функция cartesian_to_polar преобразует декартовы координаты в полярные, что позволяет отображать кривую в полярной системе координат.

Создание графического интерфейса приложения осуществляется с помощью библиотеки tkinter. Создается основное окно приложения, в котором размещаются элементы управления, такие как текстовые поля и кнопки, а также сам график, построенный с помощью matplotlib. График кривой Безье может отображаться как в декартовой, так и в полярной системе координат, что позволяет пользователю выбирать наиболее удобный способ визуализации.

Одним из важных элементов приложения является анимация. С помощью функции start_animation, которая использует класс FuncAnimation из библиотеки matplotlib.animation,

реализуется движение квадрата вдоль кривой Безье. Пользователь может запускать и останавливать анимацию в любой момент, нажимая на соответствующую кнопку, что делает процесс наблюдения за движением объекта по кривой интерактивным и увлекательным.

В конце программы запускается основной цикл tkinter, который обеспечивает работу графического интерфейса и обработку всех событий, происходящих при взаимодействии пользователя с приложением. Этот цикл позволяет приложению отслеживать все изменения в интерфейсе, перемещать график, изменять его масштаб, а также запускать и останавливать анимацию в соответствии с действиями пользователя.

4 Тестирование программы

Программа, визуализирующая кривую Безье в декартовой и полярной системах координат, подверглась тщательному и всестороннему тестированию. Эта проверка была проведена с целью убедиться в её корректной работе как при вводе корректных, так и некорректных данных, а также для проверки стабильности и надёжности функционирования при различных пользовательских взаимодействиях.

Тестирование включало в себя множество различных сценариев использования, начиная от стандартных операций ввода данных и заканчивая проверкой поведения программы при экстремальных условиях и аномальных вводных. Основное внимание уделялось тому, как программа обрабатывает информацию и какие механизмы она использует для предотвращения и исправления возможных ошибок.

Важным аспектом тестирования было также изучение отзывчивости интерфейса и его интуитивной понятности для конечного пользователя. Таким образом, тестирование позволило выявить сильные и слабые стороны программы, обеспечив всесторонний анализ её возможностей и потенциальных улучшений.

В результате выполненного тестирования были получены данные, которые подтвердили, что программа качественно справляется с поставленными задачами и обеспечивает высокий уровень пользовательского опыта.

4.1 Проверка работы программы при вводе корректных данных

Программа успешно демонстрирует свою работоспособность при вводе корректных данных. Ввод правильных значений в текстовые поля для координат точек кривой Безье подтверждает точность и надежность программы. Графическое представление кривой на экране строится плавно и точно в обеих системах координат - декартовой и полярной. Это свидетельствует о высоком уровне интеграции математических алгоритмов и принципов в программное обеспечение.

Своевременное и точное обновление координат точек кривой при вводе данных пользователя подчеркивает эффективность обработки информации программой. Мгновенный пересчет и визуализация кривой на графике при вводе новых координат свидетельствуют о высокой производительности и отзывчивости программы. Это особенно важно при работе с графическими приложениями, где требуется оперативная обработка и отображение данных.

Программа успешно визуализирует кривую в декартовой системе координат, обеспечивая точное отображение аннотаций для каждой точки кривой. Это позволяет пользователю легко интерпретировать данные и анализировать результаты. Полярный график также корректно отображает кривую, эффективно преобразуя координаты из декартовой

системы в полярную. Это говорит о глубокой интеграции математических принципов и алгоритмов в программу и подчеркивает её универсальность и многофункциональность.

Процесс масштабирования и перемещения графика также был успешно протестирован. Это подтверждает удобство и плавность работы с программой. Используя простые и интуитивно понятные средства управления - мышь и клавиши-стрелки - пользователь может легко изменять масштаб и положение графика на экране. Плавное и гибкое изменение масштаба обеспечивает высокую степень контроля, а корректное обновление границ осей при перемещении графика гарантирует стабильность и надежность отображения.

В целом, успешное тестирование с корректными входными данными демонстрирует надежность и стабильность программы. Приложение успешно обрабатывает вводимые данные, обеспечивает точное и корректное отображение графика в различных системах координат и максимизирует удобство использования благодаря плавной и эффективной функциональности масштабирования и перемещения. Это подтверждает высокую степень готовности программы к использованию и её способность обеспечить качественный пользовательский опыт.

4.2 Проверка работы программы при вводе некорректных данных

В процессе тестирования, при вводе текстовых символов вместо ожидаемых числовых значений, программа демонстрировала свою устойчивость. Она генерировала сообщения об ошибке и предотвращала неверную интерпретацию вводных данных. Это свидетельствует о внедренной системе проверки данных на валидность, которая служит гарантом корректности обработки информации и защиты пользовательского опыта от нежелательных ошибок.

Программа также успешно обрабатывает ситуации, когда данные о координатах точек кривой отсутствуют или недоступны. Это обеспечивает надёжность работы приложения, предотвращает нежелательное поведение и обеспечивает стабильность работы, исключая возможность аварийных ситуаций и сбоев. Таким образом, программа демонстрирует высокую степень устойчивости и надежности при работе с некорректными данными.

Динамические особенности программы также были тщательно проверены. Функция анимации отработала безупречно: программа плавно перемещала квадрат по кривой, демонстрируя правильное обновление координат и корректное отображение анимации. Это подчеркивает способность программы успешно работать с динамическими изменениями данных, гарантируя целостность визуализации и представления данных в любых условиях.

В общем и целом, результаты тестирования подчёркивают успешное выполнение программы с некорректными данными. Программа обеспечивает корректное отображение ошибок и предотвращение недопустимых действий. Эффективность работы программы при

взаимодействии с графиком подтверждается корректной обработкой масштабирования, перемещения и анимации. Программа обеспечивает удобный и надёжный пользовательский опыт, поддерживая стабильность работы в широком диапазоне вводных данных и условий работы. Это демонстрирует высокую надёжность и устойчивость программы, делая её привлекательной для использования в различных сферах, где требуется работа с графиками и кривыми.

Заключение

В ходе прохождения учебной ознакомительной практики было разработано программное обеспечение, которое отображает график кривой Безье в декартовой и полярной системах координат. Программа написана на языке Python с использованием библиотек tkinter и matplotlib. Основной целью было создание приложения, которое корректно отображает кривую в двух системах координат и автоматически масштабируется при изменении размеров окна.

В процессе разработки я освоил работу с библиотеками для создания графического интерфейса и визуализации данных, а также углубил знания в области программирования на Python. Программа позволяет вводить координаты контрольных точек кривой Безье и отображает результат в виде графика. Реализована возможность масштабирования и перемещения графика, а также добавлена функция анимации для визуализации движения по кривой. Программа поддерживает отображение кривой в полярной системе координат и предоставляет удобный графический интерфейс.

Тщательное тестирование программы позволило выявить и устранить ошибки, улучшив устойчивость и надежность приложения. Программа демонстрирует высокую степень интеграции математических алгоритмов и принципов, обеспечивая точное и корректное отображение графика в различных системах координат. Это подтверждает её готовность к использованию и способность обеспечить качественный пользовательский опыт.

Разработка данного программного обеспечения позволила мне углубить знания и получить ценный опыт, который будет полезен в дальнейшей профессиональной деятельности. Я освоил работу с различными библиотеками Python, научился создавать графические интерфейсы и визуализировать данные, а также улучшил навыки программирования и отладки кода. Этот опыт будет полезен в будущем при решении задач, связанных с визуализацией данных и созданием пользовательских интерфейсов.

Таким образом, поставленные задачи были успешно выполнены, и разработанное программное обеспечение соответствует требованиям задания. Программа демонстрирует возможности языка Python и его библиотек для решения задач визуализации и взаимодействия с пользователем. Она обеспечивает удобный и интуитивно понятный интерфейс, точное отображение графиков и стабильную работу при различных условиях. Это делает её полезным инструментом для различных приложений, связанных с графиками и кривыми.

В целом, разработка данного программного обеспечения стала важным этапом в моем профессиональном развитии, позволившим мне приобрести новые знания и навыки, которые будут полезны в дальнейшей карьере. Программа демонстрирует возможности современных

технологий и их применение для решения практических задач, что делает её ценным вкладом в моё образование и профессиональное становление.

Кроме того, разработка данного программного обеспечения позволила мне углубить понимание принципов работы с графическими интерфейсами и алгоритмами визуализации данных. Я научился эффективно использовать возможности библиотек tkinter и matplotlib для создания интерактивных и наглядных приложений. Это знание будет полезно при разработке других проектов, требующих визуализации сложных данных и взаимодействия с пользователем.

Также, в процессе работы над проектом я освоил методы оптимизации производительности приложения, что позволило добиться высокой скорости работы и стабильности программы. Это особенно важно для приложений, работающих с большими объемами данных и требующих быстрой обработки информации.

Кроме того, я научился применять различные техники тестирования и отладки кода, что позволило значительно улучшить качество и надежность программы. Этот опыт будет полезен в будущем при разработке сложных программных систем, требующих высокой степени надежности и устойчивости к ошибкам.

Таким образом, разработка данного программного обеспечения не только позволила мне выполнить поставленные задачи, но и значительно расширила мои профессиональные навыки и знания. Я уверен, что полученный опыт будет полезен в дальнейшей карьере и поможет мне успешно решать сложные задачи в области программирования и разработки программного обеспечения.

Список использованных источников

1 WikipediA [Электронный ресурс]: свободная энциклопедия – // Bézier curve – URL: https://dic.academic.ru/dic.nsf/ruwiki/1188667

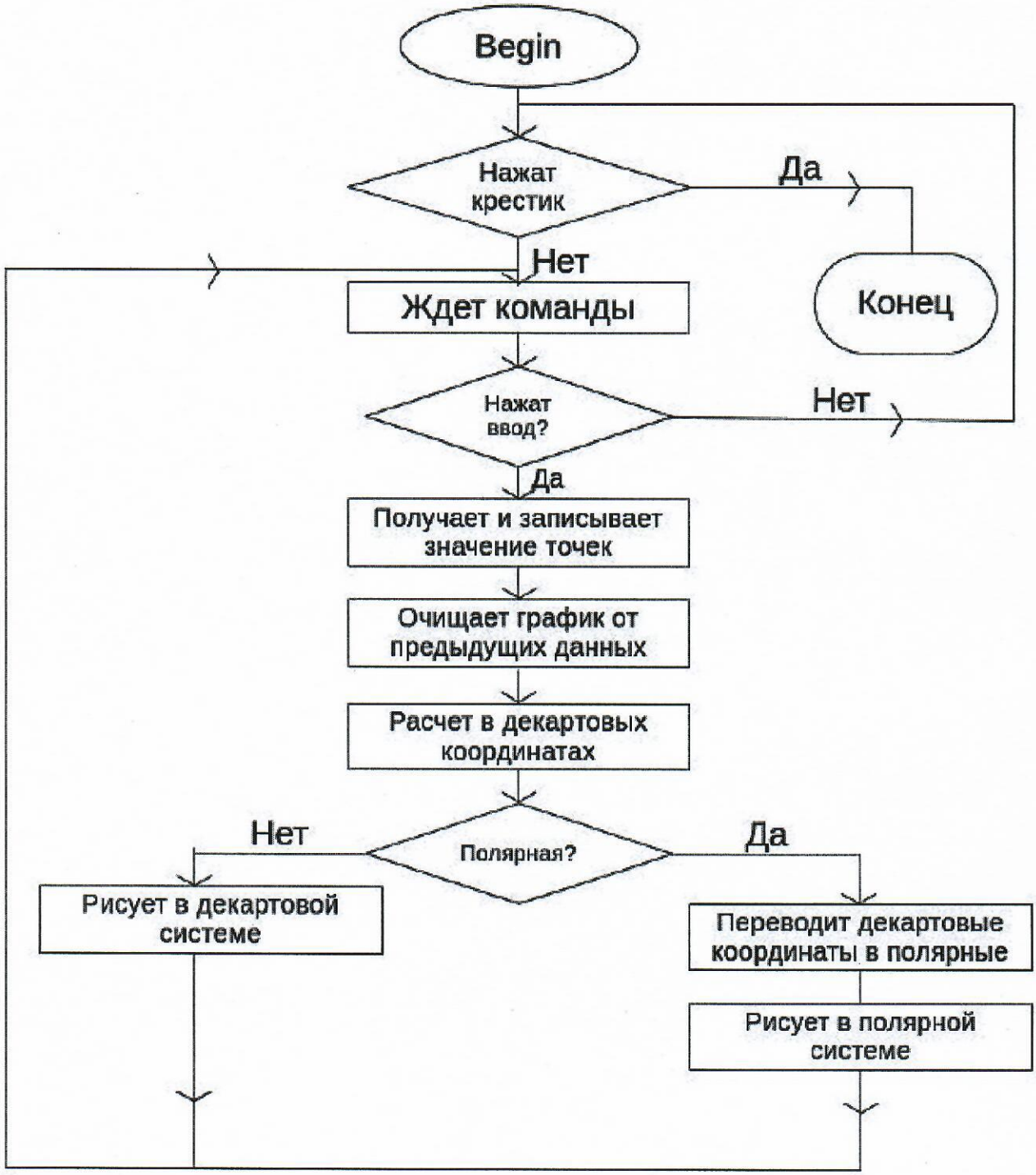
2 Утемисова А.А., Романов П.Ю. НЕКОТОРЫЕ МЕТОДЫ ПОСТРОЕНИЯ КРИВЫХ БЕЗЬЕ. [Электронный ресурс] // М.: «Мир», 1993. – URL: https://ssi.magtu.ru/doc/bezier.pdf

3 Роджерс Д., Адамс Дж. Математические основы машинной графики. — М.: Мир, 2001.

4 СК-СТО-ТР-04-1.005-2015. Требования к оформлению текстовой части выпускаемых квалификационных работ, курсовых работ (проектов), рефератов, контрольных работ, отчетов по практикам, лабораторным работам [Электронный ресурс] // ВГУЭС – URL: https://www.vvsu.ru/files/ED115C05-F320-42DE-BFB4-8A6348317716.pdf

Приложение А

Блок-схема



Приложение Б

Исходный код программы.

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
import numpy
from matplotlib.animation import FuncAnimation
# Функция для масштабирования графика
def on_press(event):
    if event.button == 1:  # Левая кнопка мыши
        ax._press_event = event, ax.get_xlim(), ax.get_ylim()
annotations = []
annotations_polar = []
def on_release(event):
    ax._press_event = None
def annotate_bezier_points(points, ax):
    global annotations
    # Удаляем все существующие аннотации
    for ann in annotations:
        ann.remove()
    annotations.clear()
    # Добавляем новые аннотации
    for i, (x, y) in enumerate(points):
        ann = ax.annotate(f'P{i}', (x, y), textcoords="offset points", xytext=(5, -5), ha='right')
        annotations.append(ann)  # Добавляем аннотацию в список
    ax.figure.canvas.draw_idle()
def annotate_bezier_points_polar(points, ax_polar):
    global annotations_polar
    # Удаляем все существующие аннотации
    for ann in annotations_polar:
        ann.remove()
    annotations_polar.clear()
    # Преобразование точек в полярные координаты
    r, theta = cartesian_to_polar(numpy.array([p[0] for p in points]), numpy.array([p[1] for p in points]))
```

```
    # Добавляем аннотации в полярную систему координат
    for i, (r_i, theta_i) in enumerate(zip(r, theta)):
        # Преобразуем угол в градусы
        theta_deg = numpy.degrees(theta_i)
        # Добавляем аннотацию
        ann = ax_polar.annotate(f'P{i}', (theta_deg, r_i),
                                textcoords="offset points", xytext=(5, -5), ha='right')
        annotations_polar.append(ann)
    ax_polar.figure.canvas.draw_idle()
def on_motion(event):
    if not ax._press_event:
        return
    press_event, x_lim, y_lim = ax._press_event
    if event.xdata is not None and press_event.xdata is not None:
        dx = event.xdata - press_event.xdata
        dy = event.ydata - press_event.ydata
    ax.set_xlim(x_lim[0] - dx, x_lim[1] - dx)
    ax.set_ylim(y_lim[0] - dy, y_lim[1] - dy)
    canvas.draw_idle()
def zoom(event):
    base_scale = 1.1
    # Получаем текущие пределы осей
    cur_xlim = ax.get_xlim()
    cur_ylim = ax.get_ylim()
    cur_xrange = (cur_xlim[1] - cur_xlim[0]) * 0.5
    cur_yrange = (cur_ylim[1] - cur_ylim[0]) * 0.5
    xdata = event.xdata  # Получаем координаты точки в месте события
    ydata = event.ydata
    if event.button == 'up':
        # Приближаем
        scale_factor = 1 / base_scale
    elif event.button == 'down':
        # Отдаляем
        scale_factor = base_scale
    else:
```

```
        # Ничего не делаем, если это не прокрутка колёсиком
        scale_factor = 1
    # Обновляем пределы осей
    ax.set_xlim([xdata - cur_xrange * scale_factor,
                 xdata + cur_xrange * scale_factor])
    ax.set_ylim([ydata - cur_yrange * scale_factor,
                 ydata + cur_yrange * scale_factor])
    canvas.draw_idle()  # Перерисовываем график
def bezier_curve(points, t_values):
    n = len(points) - 1
    curve_x = []
    curve_y = []
    for t in t_values:
        x = sum([binom(n, i) * (t ** i) * ((1 - t) ** (n - i)) * point[0] for i, point in enumerate(points)])
        y = sum([binom(n, i) * (t ** i) * ((1 - t) ** (n - i)) * point[1] for i, point in enumerate(points)])
        curve_x.append(x)
        curve_y.append(y)
    return curve_x, curve_y
import math
def binom(n, k):
    return math.factorial(n) // (math.factorial(k) * math.factorial(n - k))
# Функция для перемещения графика с помощью клавиш-стрелок
def on_key(event):
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    if event.key == 'up':
        ax.set_ylim(ylim[0] + 1, ylim[1] + 1)
    elif event.key == 'down':
        ax.set_ylim(ylim[0] - 1, ylim[1] - 1)
    elif event.key == 'left':
        ax.set_xlim(xlim[0] - 1, xlim[1] - 1)
    elif event.key == 'right':
        ax.set_xlim(xlim[0] + 1, xlim[1] + 1)
    canvas.draw_idle()
def update_bezier_points():
```

```
    global bezier_curve_x, bezier_curve_y
    # Обновляем координаты точек Безье согласно введенным данным
    new_points = []
    for i in range(4):  # У кривой Безье четыре вершины
        try:
            x = float(text_boxes[i * 2].get())  # Исправлено здесь
            y = float(text_boxes[i * 2 + 1].get())  # Исправлено здесь
            new_points.append((x, y))
        except ValueError:
            print("Неверный формат ввода!")
            return
    bezier_points[:] = new_points
    # Пересчитываем кривую Безье
    bezier_curve_x, bezier_curve_y = bezier_curve(bezier_points, t_values)
    line_cartesian.set_data(bezier_curve_x, bezier_curve_y)
    # Обновляем полярный график
    r, theta = cartesian_to_polar(numpy.array(bezier_curve_x), numpy.array(bezier_curve_y))
    line_polar.set_data(theta, r)
    annotate_bezier_points(bezier_points, ax)
    annotate_bezier_points_polar(bezier_points, ax)
    plt.draw()
# Создаем окно tkinter
root = tk.Tk()
root.title("Кривая Безье")
fig, ax = plt.subplots(figsize=(10, 6), dpi=100)
ax.set_aspect('equal')
# Добавление оси X
ax.axhline(y=0, color='black', linewidth=2)
# Добавление оси Y
ax.axvline(x=0, color='black', linewidth=2)
fig.canvas.mpl_connect('scroll_event', zoom)
fig.canvas.mpl_connect('button_press_event', on_press)
fig.canvas.mpl_connect('button_release_event', on_release)
fig.canvas.mpl_connect('motion_notify_event', on_motion)
fig.canvas.mpl_connect('key_press_event', on_key)
```

```
# Создаем холст для фигуры
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.draw()
canvas_widget = canvas.get_tk_widget()
canvas_widget.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
# Создаем фрейм для элементов управления
control_frame = tk.Frame(root)
control_frame.pack(side=tk.RIGHT, fill=tk.Y)
ax.grid(True)
ax.set_title('Кривая Безье (Декартовы координаты)')
# Координаты для кривой Безье третьего порядка
bezier_points = [(-4, 4), (-2, 10), (0, 6), (2, 5)]
annotate_bezier_points(bezier_points, ax)
t_values = numpy.linspace(0, 1, 100)
bezier_curve_x, bezier_curve_y = bezier_curve(bezier_points, t_values)
line_cartesian, = ax.plot(bezier_curve_x, bezier_curve_y, color='red', label='Кривая Безье')
# Отображаем круговую разметку полярной системы
ax_polar = fig.add_axes([0.1, 0.1, 0.8, 0.8], polar=True)
ax_polar.set_rticks(numpy.arange(0, 120, 20))  #  установите нужное количество радиальных
линий
ax_polar.set_rlabel_position(0)
ax_polar.grid(True, which='both', linestyle='-', linewidth=0.5)
def cartesian_to_polar(x, y):
    global r, theta
    r = numpy.sqrt(x**2 + y**2)
    theta = numpy.arctan2(y, x)
    return r, theta
r, theta = cartesian_to_polar(numpy.array(bezier_curve_x), numpy.array(bezier_curve_y))
line_polar, = ax_polar.plot(theta, r, color='red', label='Кривая Безье')
ax_polar.grid(True)
ax_polar.set_title('Кривая Безье (Полярные координаты)')
ax_polar.set_xticks(numpy.linspace(0, 2 * numpy.pi, 8))
ax_polar.set_xticklabels(['0°', '45°', '90°', '135°', '180°', '225°', '270°', '315°'])
ax_polar.set_yticks(numpy.arange(0, 11, 2))
ax_polar.set_visible(False)
```

```
text_boxes = []
for i, point in enumerate(bezier_points):
    # Создаем текстовые поля для X и Y координат
    label_x = tk.Label(control_frame, text=f"P{i}x:")
    label_x.pack()
    text_box_x = tk.Entry(control_frame)
    text_box_x.insert(0, f"{point[0]}")
    text_box_x.pack()
    text_boxes.append(text_box_x)

    label_y = tk.Label(control_frame, text=f"P{i}y:")
    label_y.pack()
    text_box_y = tk.Entry(control_frame)
    text_box_y.insert(0, f"{point[1]}")
    text_box_y.pack()
    text_boxes.append(text_box_y)
# Кнопка "Обновить"
update_button = tk.Button(control_frame, text="Обновить", command=update_bezier_points)
update_button.pack()
# Кнопки переключения
button_polar = tk.Button(control_frame, text='Полярная', command=lambda:
switch_to_polar(None))
button_polar.pack()
button_cartesian = tk.Button(control_frame, text='Декартовая', command=lambda:
switch_to_cartesian(None))
button_cartesian.pack()
# Глобальная переменная для отслеживания анимации
animation = None
# Функция для запуска анимации
def start_animation():
    global animation  # Используем глобальную переменную

    # Проверяем, не запущена ли уже анимация
    if animation is not None:
        animation.event_source.stop()  # Останавливаем анимацию, если она уже запущена
```

```
        animation = None
        for patch in ax.patches:
            patch.remove()
        canvas.draw()
        return
    # Создаем квадрат
    square = plt.Rectangle((0, 0), 1, 0, color='blue', animated=True)  # Сторона 2
    ax.add_patch(square)
    # Функция для обновления анимации
    def animate(i):
        nonlocal square
        # Получаем координаты точки на кривой
        side = 3
        if ax.get_visible():
            if i < len(bezier_curve_x):
                x = bezier_curve_x[i]
                y = bezier_curve_y[i]
                square.set_bounds(x - side / 2, y - side / 2, side, side)  # Устанавливаем позицию квадрата
        else:
            if i < len(r):
                x = r[i]
                y = theta[i]
                square.set_bounds(x - side / 2, y - side / 2, side, side)
        return square,
    # Запускаем анимацию
    animation = FuncAnimation(fig, animate, frames=len(bezier_curve_x), interval=50, blit=True)
    canvas.draw()
# Кнопка для анимации
animate_button = tk.Button(control_frame, text="Анимировать", command=start_animation)
animate_button.pack()
def switch_to_polar(event):
    ax.set_visible(False)
    ax_polar.set_visible(True)
    fig.canvas.draw()
def switch_to_cartesian(event):
```

```
        ax.set_visible(True)
        ax_polar.set_visible(False)
        fig.canvas.draw()
    root.mainloop()
```