МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

Работа допущена к защите
_____Руководитель
«___»_____20___г.

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

на тему: «Численные вероятностные алгоритмы»

Студент _____ Бессонов М.П.
Шифр 191009
Институт приборостроения, автоматизации и информационных технологий
Направление подготовки 09.03.04 «Программная инженерия»
Группа 92-ПГ
Руководитель _____ Артемов А.В.
Оценка: «_____» Дата 14.01.21

Орел 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

УТВЕРЖДАЮ:
_____ Зав. кафедрой
«__»_____20__г.

ЗАДАНИЕ
на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Студент Бессонов М.П. Шифр 191009

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92-ПГ

1 Тема курсовой работы

«Численные вероятностные алгоритмы»

2 Срок сдачи студентом законченной работы «12» декабря 2021

3 Исходные данные

Дана полиномиальная знакопостоянная на промежутке a<=x<=b функция f(x), задаваемая пользователем степенью полинома и его коэффициентами. Найти площадь фигуры, ограниченной графиком функции, и прямыми x=a, x=b (a<b), y=0.

4 Содержание курсовой работы

Описание и анализ поставленной задачи

Анализ методов решения задачи

Анализ и выбор структур данных

Реализация и тестирование программного средства

Оценка эффективности

5 Отчетный материал курсовой работы

Пояснительная записка курсовой работы

Руководитель _____ Артемов А.В.

Задание принял к исполнению: «12» октября 2021

Подпись студента _____

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир предъявляет высокие требования к выпускнику технического вуза. Современный специалист в области IT должен уметь работать в разных программах и с разными ресурсами. Кроме того, он должен уметь работать с различными структурами данных, а также реализовывать и исследовать различные алгоритмы.

Существуют несколько классических алгоритмических задач, результат у которых можно найти только приблизительно. И чем дольше работает такой алгоритм, тем точнее будет полученный ответ. Эти задачи решаются с использованием вероятностных алгоритмов. Применение численных вероятностных алгоритмов является актуальным, так как они позволяют получить экономию во времени работы за счёт замены абсолютной достоверности результата достоверностью с некоторой вероятностью.

Объектом нашей работы является вероятностная задача о нахождении площади фигуры, ограниченной полиномиальной функцией. Предметом нашей работы является процесс разработки вероятностного алгоритма для решения задачи о нахождении площади фигуры.

Цель нашей работы: приобрести навыки разработки алгоритмов на основе решения задачи о нахождении площади фигуры.

Для достижения цели нам необходимо решить соответствующие задачи:

1) описание и анализ поставленной задачи;
2) анализ методов решения задачи;
3) анализ и выбор структур данных;
4) реализация и тестирование программного средства;
5) оценка эффективности.

ОПИСАНИЕ И АНАЛИЗ ПОСТАВЛЕННОЙ ЗАДАЧИ

Начало качественной теории вероятностных алгоритмов было положено в 1956 году, когда впервые было установлено, что посредством вероятностных алгоритмов можно вычислить в точности те же функции, что и посредством обычных, детерминированных алгоритмов [1].

Одной из задач, использующей для решения численные вероятностные алгоритмы является задача о нахождении площади фигуры, ограниченной какой-либо функцией и прямыми.

Вероятностные методы основаны на воспроизведении большого числа реализаций случайного процесса, то есть на определенных этапах своей работы они обращаются к генератору случайных чисел с целью получения экономии во времени работы. Суть методов заключается в статистическом моделировании случайных процессов, численном моделировании реализаций случайных процессов и оценивании параметров по реализациям случайных процессов методами математической статистики [2].

Одним из основных вероятностных методов, работающих со случайными величинами, является метод Монте-Карло. Решение задачи о нахождении площади фигуры с помощью этого метода можно описать так: поместим данную нам фигуру в другую фигуру, площадь которой будет легко вычисляться (квадрат или прямоугольник) и будем наугад ставить в новой фигуре точки. Будем исходить из того, что, чем больше площадь фигуры, тем чаще в нее будут попадать точки. Таким образом, при большом числе точек N, наугад выбранных внутри квадрата или прямоугольника, доля точек, содержащихся в данной нам фигуре k, приближенно равна отношению площади этой фигуры и площади квадрата или прямоугольника.

В данной работе нам необходимо реализовать вероятностный метод, решающий задачу нахождении площади фигуры, ограниченной прямыми линиями и полиномиальной функцией. Также нам необходимо реализовать алгоритмы, вычисляющие площадь фигуры методами численного

интегрирования. Полученные ответы нужно сравнить с расчетным (точным) значением интеграла.

Разработаем требования, в соответствии с которыми мы будем реализовывать наши алгоритмы. В нашей работе мы рассмотрим функциональные требования (определяют функциональность (поведение) программной системы) [8].

К функциональным требованиям нашей программы можно отнести следующие:

– отображение найденной площади фигуры вероятностным методом, методами численного интегрирования и прямым вычислением;

– отображение разницы в процентах между ответами, полученными в результате работы вероятностного метода и методов численного интегрирования от расчетного значения;

– отображение количества точек, попавших в область фигуры;

– отображение времени работы каждого алгоритма.

АНАЛИЗ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ

Для решения задачи нам необходимо выбрать алгоритмы, то есть составить ту последовательность действий, после выполнения которой исполнитель решит поставленную задачу.

Задача, которую нам необходимо решить – задача о нахождении площади фигуры, ограниченной полиномиальной функцией и прямыми. Для решения задачи воспользуемся алгоритмом, основанном на методе Монте-Карло. Этот метод используется, как правило, в тех случаях, когда площадь фигуры трудно посчитать аналитически, но можно ее посчитать приблизительно, воспользовавшись техникой бросания точек.

По условию задачи наша площадь ограничена осью y=0 и знакопостоянной на заданном промежутке полиномиальной функцией. Так как такая функция может находиться как ниже оси OX, так и выше, перед началом работы нашего алгоритма необходимо вычислить максимальное и минимальное значение функции на промежутке, ограниченном линиями по оси OX - a и b. Для расчёта значения функции мы будем использовать коэффициенты полинома, которые пользователь будет вводить с клавиатуры. В зависимости от количества коэффициентов (степени полинома), будем умножать на эти коэффициенты «x» в необходимой степени.

После этого наш алгоритм будет генерировать случайные точки с координатами x и y. При попадании в область нашей фигуры алгоритм будет увеличивать соответствующую переменную, а затем рассчитает результат путем перемножения площади квадрата (прямоугольника) и отношения попавших точек к общему числу точек.

Существует похожий метод, реализующий нахождение площади ограниченной фигуры. В своем расчете он также использует случайную величину. Для его реализации нам необходимо случайно генерировать координату по оси X и находить значение функции в этой точке. После необходимо вычислить значение площади фигуры по формуле «площадь =

ширина * средняя высота». Такой метод обеспечивает меньшую точность вычисления, так как среднее значение не учитывает большие перепады в функции. Для их устранения необходимо обеспечить большее число испытаний, чем в рассмотренном нами ранее методе.

Для сравнения результата реализуем два алгоритма численного интегрирования. Первый алгоритм будет основан на методе трапеций. Этот алгоритм будет разбивать нашу площадь на фигуры и находить значение функции в каждой из ключевых точек [7, с. 62]. Второй алгоритм будет основан на методе левых прямоугольников. Этот алгоритм будет также разбивать нашу площадь на фигуры (прямоугольники) и находить сумму их площадей [7, с. 61].

Таким образом, алгоритмы численного интегрирования решают задачу по нахождению площади через расчет интегральной функции, ограниченной тремя прямыми и заданной полиномиальной функцией. При этом следует учитывать, что площадь фигуры не может быть отрицательной, но так как по условию функция может быть только знакопостоянной (то есть находится либо ниже, либо выше оси OX) на заданном промежутке, при расположении площади фигуры ниже оси OX мы будем просто менять знак в ответе нашего интеграла.

Расчетное значение интеграла для получения площади будем находить по формуле, представленной на рисунке 1. Полученный ответ также всегда будет положительным.

$$\int_{X_1}^{X_2} a_i x^i = a_i \frac{X_2^{i+1} - X_1^{i+1}}{i+1}.$$

Рисунок 1 – Формула расчета интеграла полинома

Рассчитаем асимптотическую сложность наших алгоритмов [4].

В каждом из приведенных алгоритмов нам необходимо вычислить

значение полиномиальной функции в какой-то точке. Для этого необходимо пройти по всем заданным коэффициентам полинома. При этом такие вычисления функции нам необходимо проводить столько раз, сколько задано испытаний (интервалов), кроме прямого расчета. При этом стоит учитывать, что для нахождения площади методом Монте-Карло необходимо рассчитать максимальное и минимальное значение функции, чтобы генерировать значения в нужном диапазоне.

Таким образом, можно сказать, что асимптотическая сложность алгоритма, реализующего метод Монте-Карло равна O(N*M + fMinMax), где N – число наших коэффициентов полинома, M – число испытаний (интервалов), а fMinMax сложность алгоритма поиска максимума и минимума.

Для нахождения максимума и минимума мы реализуем функцию, которая просто перебирает все значения от левой границы до правой с минимальным константным шагом, но при этом на каждом шаге рассчитывает значение. Поэтому можно сказать, что асимптотическая сложность алгоритма fMinMax равняется O(C*N), где C – количество шагов, зависящее от левой и правой границы, а N – число наших коэффициентов.

Сложность алгоритма, реализующего метод прямоугольников равна O(N*M). Сложность алгоритма, реализующего метод трапеций O(2N*M), так как на каждой итерации нам необходимо вычислить значение в двух точках. Сложность алгоритма, рассчитывающего точное значение интеграла равна O(N), так как, согласно формуле на рисунке 1, для нахождения значения нам необходимо только пройти по всем коэффициентам полинома. То есть, можно сказать, что сложность всех алгоритмов, которые мы будем реализовывать, линейная, то есть для них будет справедлива оценка O(n).

АНАЛИЗ И ВЫБОР СТРУКТУР ДАННЫХ

Для хранения данных и их дальнейшего отображения на экране выберем конкретные структуры данных для работы с нашими алгоритмами.

На этапе анализа метода решения мы выяснили, что нам необходимо работать с коэффициентами, которые задают полином. Для хранения этого набора коэффициентов создадим структуру данных – «динамический массив» [3] типа float (чисел с плавающей точкой). Каждому элементу нашего массива будет присваиваться порядковый номер - его индекс. Это позволит нам непосредственно обращаться к нашим коэффициентам, а также добавлять элементы в конец нашего массива, затрачивая на это константное время.

Наш динамический массив должен изначально выделять под себя некоторую последовательность ячеек в физической памяти, а при невозможности очередного добавления элемента создать новый массив с расширенным в два раза запасом и скопировать все данные в этот новый массив.

Для хранения наших коэффициентов мы также можем использовать связный список [5] или статический массив [6]. Но так как добавление данных и их последовательное извлечение из нашей структуры данных являются главными операциями, динамический массив будет работать лучше, чем связный список, так как на операцию добавления он будет тратить константное время, а на операцию извлечения всех элементов - линейное время. В связном списке на операцию извлечения также тратится линейное время, но чтение будет происходить медленнее, так как нужно еще тратить ресурсы на чтение адреса следующего элемента списка. На добавление же константное время будет тратиться только если мы будем отдельно хранить хвост связного списка.

Кроме того, динамический массив позволит нам считывать коэффициенты, введенные пользователем, последовательно и сразу

добавлять их в нашу структуру. Это позволит избежать траты лишнего времени на подсчет количества коэффициентов, что было бы необходимо при использовании статического массива. Также использование именно динамического массива является удобным решением для пользователя, так как ему не придется заранее думать о том, сколько коэффициентов в полиноме должно быть. Он может сразу вводить их в поле и не быть ограниченным каким-то заданным ранее статическим размером.

Также так как нам придется работать со случайно сгенерированными величинами, динамический массив позволит без особых трудностей хранить необходимые нам данные. Например, мы можем хранить в них координаты всех точек, которые попали или не попали в область нашей фигуры, чтобы потом использовать их для построения графика.

Реализуемая нами структура данных позволяет избежать неэффективного использования памяти, но при частом изменении размера массива может привести к снижению скорости работы из-за накладных расходов на изменение размера динамического массива.

РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Для реализации нашей задачи воспользуемся языком программирования Python версии 3.7. Этот язык имеет простой и понятный интерфейс, имеет широкие возможности применения, а также он кроссплатформенный. С другой стороны, этот язык не обладает слишком хорошей производительностью, когда речь идет о больших проектах, но для демонстрации решения нашей задачи этот недостаток не будет критичным.

Для решения наших задач создадим структуру данных «динамический массив» типа float.

Для этого создадим класс DynamicArray, в котором будем хранить его первоначальный размер, который будет равен 1, количество фактических элементов и первоначальный массив.

Создадим функции для получения его длины и элемента по конкретному индексу (если заполненность массива не равна нулю).

Для создания функции добавления реализуем две функции увеличения размера массива. Функция «_resize» будет создавать новый массив удвоенной вместимости и переписывать в него все наши элементы. Функция «make_array» будет возвращать массив определенного размера. При невозможности добавления нового элемента в текущий массив эти функции будут вызываться. При добавлении новому элементу будет присваиваться индекс и увеличиваться заполненность массива.

Полный код нашего класса, описывающего структуру данных «динамический массив» приведен в приложении А.

Реализуем описанные нами алгоритмы на языке программирования Python. Для этого создадим следующие функции.

Для вероятностного алгоритма, основанного на методе Монте-Карло создадим функцию MCIntegrate, которая будет принимать на вход левую и правую границу нашей функции, а также число испытаний. Также для

работы нам необходимо максимальное и минимальное значения нашей полиномиальной функции. Для этого создадим функцию functionOptimize, которая на вход будет принимать левую и правую границы и в цикле с небольшим шагом будет искать максимум и минимум нашей полиномиальной функции. Далее в нашей функции MCIntegrate необходимо определить переменные для количества точек, попавших в фигуру (figure_point) и результата вычисления (result_MC). После этого в цикле нам необходимо генерировать случайные координаты x и y, рассчитывать нашу функцию от сгенерированного x и сравнивать полученное значение со сгенерированным y, учитывая расположение нашей фигуры в системе координат. Если сгенерированный y попадает в область нашей функции, увеличиваем переменную figure_point.

По завершению всех испытаний находим результат, который равен отношению попавших в фигуру точек (figure_point) к общему числу испытаний (nTrials), умноженному на площадь нашей фигуры (разница правой и левой границы, помноженной на найденное ранее максимальное или минимальное значение - высоту).

Для вычисления значения функции по коэффициентам создадим отдельный класс Poly, который будет обрабатывать наши коэффициенты из динамического массива. Определим в нем функцию evaluate, которая в цикле будет получать наши коэффициенты и по правилу приводить их к ответу. Кроме того в этом классе будет рассчитываться точное значение интеграла при помощи функции take_integral, которая будет преобразовывать наши коэффициенты полинома, используя заданные пользователем границы площади, для получения точного значения. Также в этом классе реализована функция, возвращающая внешний вид полиномиальной функции (show).

Для сравнения с результатом, полученным с помощью расчетного (точного) интегрирования, реализуем функции численного интегрирования. Для реализации алгоритмов были выбраны метод трапеций и метод левых прямоугольников.

Реализация этих алгоритмом (rect_integral и tr_integral), демонстрирующих работу этих методов, очень похожа. Они оба на вход принимают правую и левую границу, внутри которых считается интеграл, а также число интервалов. Далее в этих функциях рассчитывается шаг, равный разнице между правой и левой границей, поделенной на число интервалов. После этого в цикле рассчитывается значение интеграла согласно математическим формулам численных методов [7, с. 61-62].

Если минимальное значение нашей функции меньше нуля, и при этом максимальное значение равно нулю, алгоритмы численного интегрирования, а также функция точного получения значения интеграла после расчета заменят его знак на противоположный (для получения площади фигуры). Если же минимальное и максимальное значение имеют противоположные знаки программа выдаст предупреждение о том, что полиномиальная функция не является знакопостоянной на заданном промежутке.

Код реализации наших алгоритмов приведен в приложении Б.

Общий вид программы, реализующей все описанные выше алгоритмы приведен на рисунке 2. Число испытаний, заданное пользователем также передается как число интервалов необходимых для работы численных методов.

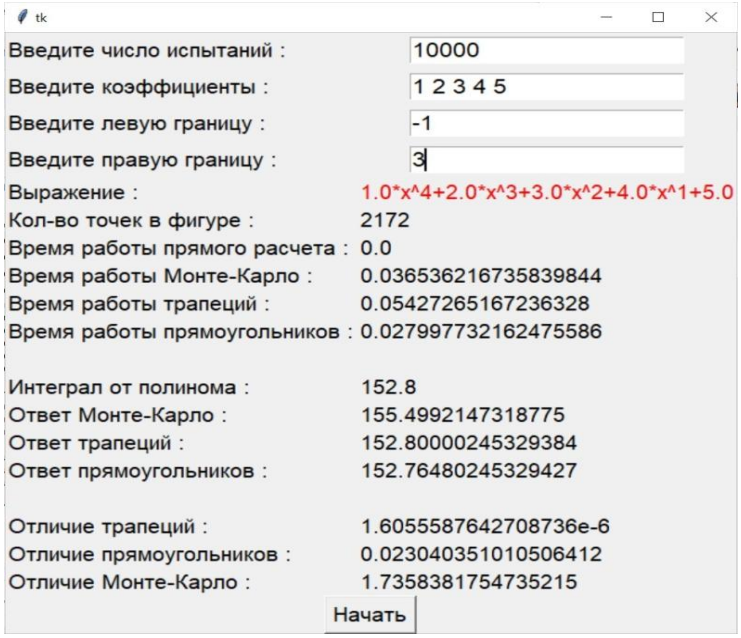

Рисунок 2 – Внешний вид программы

График представленной коэффициентами полиномиальной функции и сгенерированных точек представлен на рисунке 3.

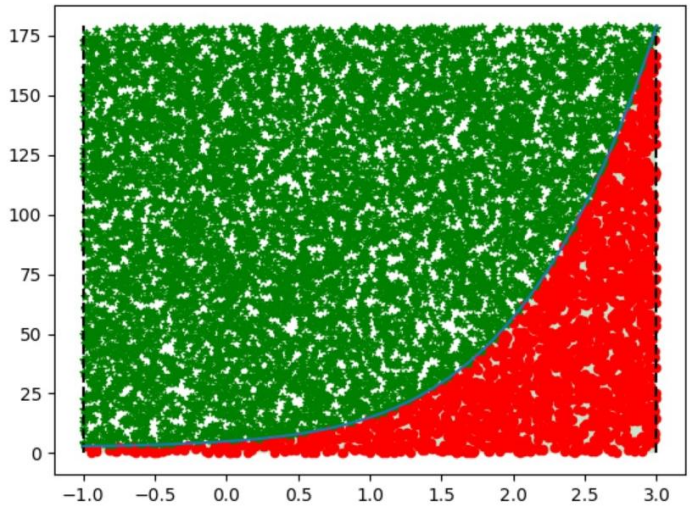

Рисунок 3 – График полиномиальной функции

Как видно из результатов, все три алгоритма дают примерно одинаковый результат. Отличие между методом Монте-Карло и точным интегралом составляет менее 5% при достаточно небольшом количестве испытаний (интервалов), что говорит о достаточной надежности метода.

Следует отметить, что при многократном повторении испытаний полученное вероятностным алгоритмом среднее будет фактически соответствовать реальному значению (в нашем примере около 153). На рисунке 4 приведен пример такой генерации из 10000 испытаний.

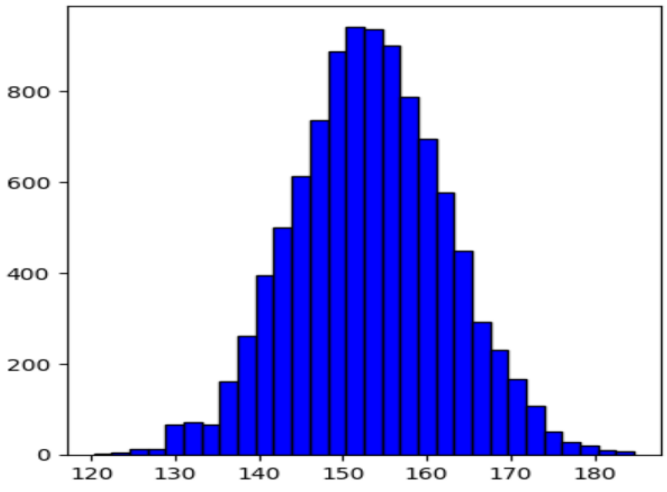

Рисунок 4 – Распределение результатов

ОЦЕНКА ЭФФЕКТИВНОСТИ

Проведем замеры времени работы наших алгоритмов, а также оценим точность наших алгоритмов, по сравнению с точным расчетом интеграла. Алгоритм получения точного ответа зависит только от количества полиномов, поэтому его время не будет подвергаться существенным изменениям и мы будем включать его в наше тестирование.

Так как в остальных наших алгоритмов можно изменять как количество коэффициентов, так и число испытаний (интервалов), проведем два сравнения, изменяя эти параметры по отдельности. Так как число испытаний (интервалов) проще задается в большом диапазоне, начнем изменения с него. Стоит отметить, что чем больше это число, тем точнее методы будут рассчитывать площадь искомой фигуры.

Число коэффициентов возьмем константным и равным 5 (значения: 1, 2, 3, 4, 5). Границы во время тестирования также будут неизменяемыми числами и равняться -5 и 5.

При этом при тестировании алгоритма, реализующего метод Монте-Карло будем учитывать оба варианта: известно или неизвестно нам максимальное и минимальное значение.

Результаты замеров приведены в таблице 1 (время отображается в секундах). Графическое отображение результатов замеров приведено на рисунке 5.

Таблица 1 – Сравнение эффективности по затраченному времени (интервалы)

Число испытаний (интервалов)	Монте-Карло (с поиском)	Монте-Карло (без поиска)	Метод трапеций	Метод прямоугольников
10000	0,326	0,035	0,057	0,028
50000	0,506	0,203	0,333	0,152
100000	0,633	0,373	0,574	0,286

Продолжение таблицы 1

Число испытаний (интервалов)	Монте-Карло (с поиском)	Монте-Карло (без поиска)	Метод трапеций	Метод прямоугольников
250000	1,141	0,859	1,467	0,741
500000	2,031	1,778	2,854	1,444
750000	2,941	2,621	4,434	2,129
1000000	3,891	3,532	5,911	2,883
1500000	5,768	5,502	9,067	4,383
2000000	7,488	7,032	11,802	5,919
2500000	9,281	8,944	14,794	7,314

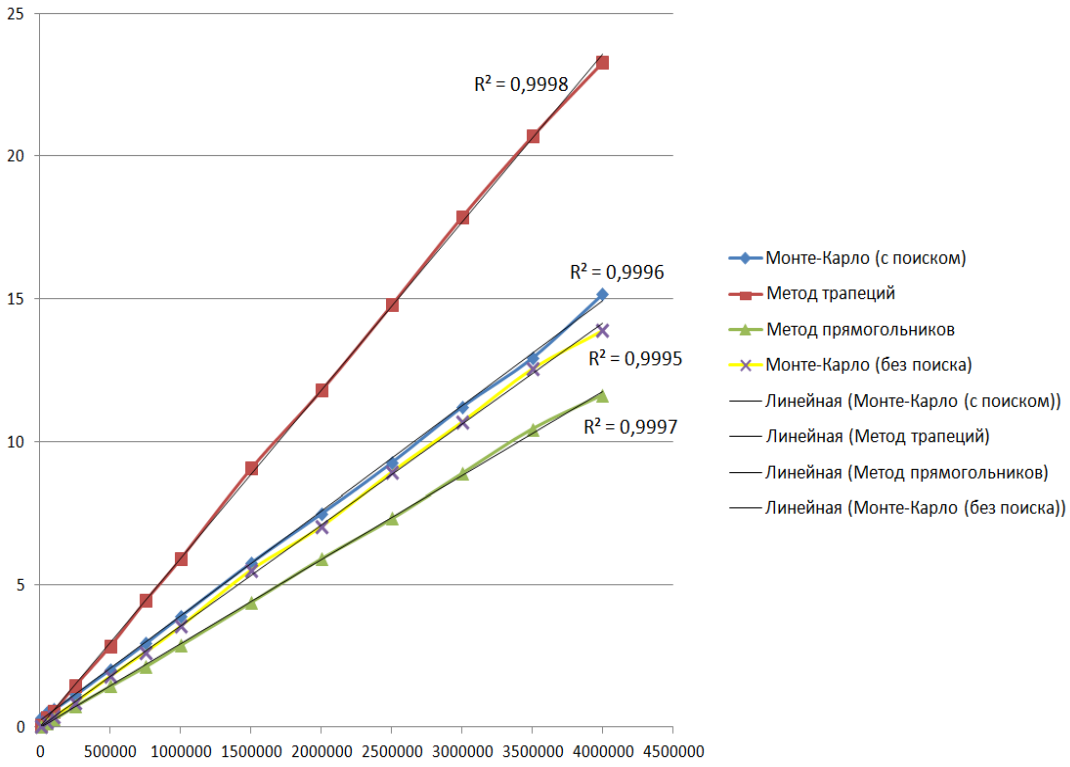

Рисунок 5 – Результаты замеров (испытания/интервалы)

Как видно из тестирования, самой маленькой временной сложностью из приведённых алгоритмов обладает метод прямоугольников. Метод Монте-Карло, рассчитывающий максимум и минимум при каждом тесте, на начальном этапе показывает самое худшее время, так как расчет максимального и минимального значения добавляет сложность. Но так как

мы не меняли границы, а шаг расчета является константой, при большом числе испытаний (интервалов) сложность метода трапеций начинает превышать сложность метода Монте-Карло из-за двойной работы по расчету функции.

По графическому отображению результатов замеров, а также по построенным к ним графикам аппроксимирующих функций можно судить, что сложность алгоритма с увеличением размера входных данных растет линейно. Все три величины достоверности аппроксимации, которые показывают близость значения линии тренда к нашим фактическим замерам, выше 0.999, что указывает на хорошее совпадение расчетной прямой с исходными данными. Следовательно, можно сказать, что время при увеличении размера входных данных будет также увеличиваться с постоянной скоростью.

Если же находить максимум и минимум отдельно от алгоритма и только передавать рассчитанные значения в функцию, реализующую метод Монте-Карло, алгоритм будет стабильно быстрее трапеций и медленнее прямоугольников при любом количестве испытаний (интервалов).

Преимущество в скорости зависит от многих факторов. В общем случае, метод Монте-Карло не является самым лучшим, когда речь заходит об эффективности вычислений. Это связано с тем, что нам приходится выполнять одну и ту же операцию много раз, что требует вычислительных ресурсов.

Далее будем изменять число коэффициентов нашего полинома. При этом число испытаний (интервалов) будет константным и равным 10000. Так как наша функция работает только со знакопостоянными функциями, зададим диапазон от 1 до 5 для более удобной оценки точности функции. При каждом же тесте будем добавлять по одному новому коэффициенту (от 1 до 10). Для более точных результатов будем проводить 1000 тестов на каждый набор коэффициентов, а затем усреднять полученные результаты. Результаты для метода Монте-Карло также будем получать в двух случаях:

известны максимум и минимум функции и неизвестны. Результаты замеров приведены в таблице 2 (время отображается в секундах). Графическое отображение результатов замеров приведено на рисунке 6.

Таблица 2 – Сравнение эффективности по затраченному времени (коэффициенты)

Число коэффициентов	Монте-Карло (с поиском)	Монте-Карло (без поиска)	Метод трапеций	Метод прямоугольников
1	0,051	0,016	0,018	0,009
2	0,075	0,02	0,028	0,014
3	0,102	0,025	0,039	0,02
4	0,131	0,031	0,049	0,025
5	0,157	0,036	0,059	0,03
6	0,183	0,042	0,069	0,035
7	0,209	0,047	0,08	0,041
8	0,235	0,052	0,09	0,046
9	0,261	0,058	0,101	0,051
10	0,286	0,063	0,111	0,057

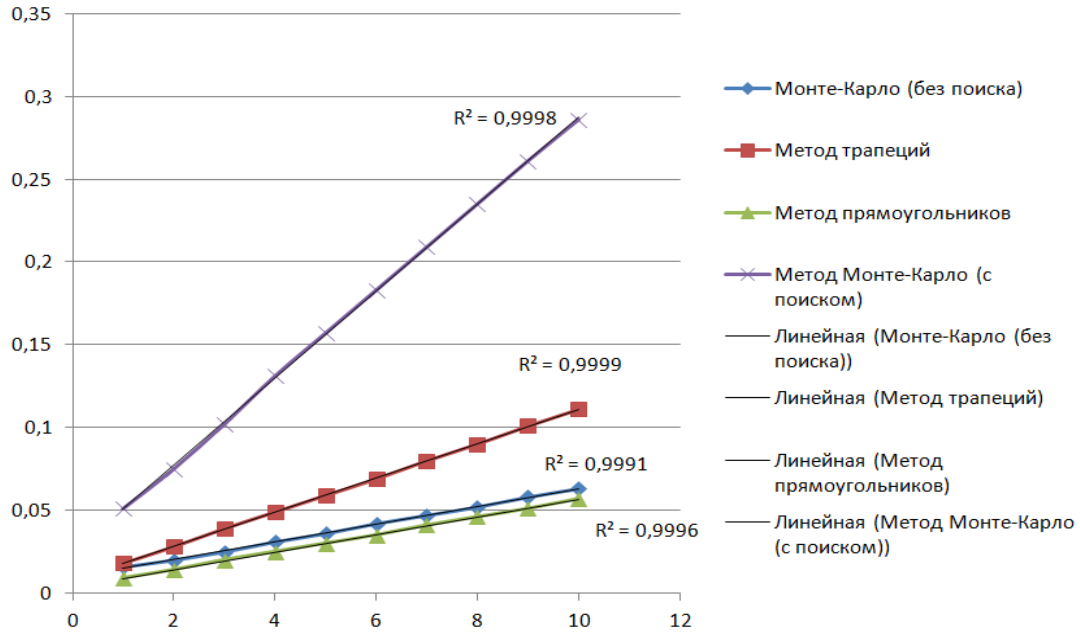

Рисунок 6 – Результаты замеров (количество коэффициентов)

Как видно из тестирования и графического отображения результатов замеров, прирост времени от увеличения количества коэффициентов линеен и достаточно мал при заданном количестве испытаний.

Таким образом, можно сделать вывод, что более влиятельным фактором является число испытаний (интервалов), так как обеспечивает большую временную изменчивость при работе алгоритмов. Число коэффициентов слабо влияет на изменение времени алгоритма при небольшом количестве испытаний (интервалов). Но при большом количестве испытаний (интервалов), увеличение количества коэффициентов в два раза существенно повлияет на время работы алгоритма, которое в этой ситуации также увеличится примерно в два раза.

Оценим точность наших алгоритмов. Так как метод Монте-Карло использует случайную величину, точность вычисления будет меняться в зависимости от теста. Проведем отдельное тестирование. Для этого возьмем количество коэффициентов равным 5 (1, 2, 3, 4, 5). Границы поставим от -1 до 3. Такой интеграл будет равным 152,8. Будем изменять количество испытаний (интервалов) и оценивать точность нашего алгоритма. Результаты замеров приведены в таблице 3 (результаты представлены, как отличие в процентах от точного ответа).

Таблица 3 – Сравнение точности алгоритмов

Число испытаний (интервалов)	Метод трапеций	Метод прямоугольников	Монте-Карло
1000	0,002	0,231	2,119
10000	1,606*10^(-6)	0,0231	1,876
50000	6,434*10^(-8)	0,005	1,363
100000	1,604*10^(-8)	0,002	0,515
250000	2,863*10^(-9)	0,001	0,425
500000	3,026*10^(-10)	0,0005	0,281

Как видно из тестирования, самой большой точностью вычисления (самым маленьким отличием от расчетного интеграла) из представленных методов обладает метод трапеций, так как при вычислении интеграла он выстраивает кривую посредством ломаных линий.

Метод прямоугольников стоит на втором месте по точности, так как он выстраивает кривую с помощью ступенчатой функции, перепрыгивающей с одной величины на другую на каждом верхнем ребре прямоугольника.

Метод Монте-Карло в конкретном случае при количестве испытаний (интервалов) свыше 100000 показывает отличие, меньшее 0,5%, что говорит о достаточной надежности метода. При отдельном тестировании этого метода этот алгоритм может показывать еще меньшее отличие (в лучшем случае около 0,01%).

Таким образом, самым точным из приведенных методов является метод трапеций, но у него и самые большие временные затраты при вычислении. Если же абсолютная точность не нужна, метод прямоугольников будет являться лучшим решением из-за небольших временных затрат и достаточно высокой точности (по сравнению со стандартными пороговыми значениями, к которым относится 5%) при достаточном количестве испытаний. Получение площади фигуры с помощью метода Монте-Карло может быть выгодно при достаточно сложной функции, которую трудно посчитать классическими численными методами.

ЗАКЛЮЧЕНИЕ

В настоящей работе был описан и реализован численный вероятностный алгоритм для решения задачи о нахождении площади фигуры, ограниченной полиномиальной функцией и прямыми. Мы выяснили, что в общем случае, метод Монте-Карло не является самым лучшим, когда речь заходит об эффективности вычислений. Но если интеграл трудно считается другими численными методами, метод Монте-Карло будет показывать наилучшие результаты.

Мы разработали основные требования к программе, спроектировали и рассмотрели особенности ее реализации. Таким образом, можно сказать, что все поставленные перед нами задачи были выполнены и цель курсовой работы была достигнута.

СПИСОК ЛИТЕРАТУРЫ

1. Вероятностный алгоритм [Электронный ресурс]. - Режим доступа: https://ww.google-info.org/1022979/1/veroyatnostnyy-algoritm.html (дата обращения: 24.10.2021).

2. Вычисление интегралов методом Монте-Карло [Электронный ресурс]. - Режим доступа: https://vuzlit.ru/890406/vychislenie_integralov_metodom_monte_karlo (дата обращения: 24.10.2021).

3. Одномерные динамические массивы [Электронный ресурс]. - Режим доступа: https://intuit.ru/studies/courses/648/504/lecture/11451 (дата обращения: 24.10.2021).

4. Оценка сложности алгоритмов [Электронный ресурс]. - Режим доступа: http://wiki.likt590.ru/doku.php/tema:ocenka_slozhnosti_algoritmov (дата обращения: 24.10.2021).

5. Список [Электронный ресурс]. - Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA (дата обращения: 24.10.2021).

6. Статические одномерные массивы [Электронный ресурс]. - Режим доступа: https://spravochnick.ru/informatika/staticheskie_odnomernye_massivy/ (дата обращения: 24.10.2021).

7. Стивенс Р. Алгоритмы. Теория и практическое применение. - Москва: Издательство «Э», 2016. - 544 с. (дата обращения: 24.10.2021).

8. Функциональные и нефункциональные требования (Functional and Non-functional Requirements) [Электронный ресурс]. - Режим доступа: https://studfile.net/preview/2152457/page:4/ (дата обращения: 24.10.2021).

ПРИЛОЖЕНИЕ А

(обязательное)

СТРУКТУРА ДАННЫХ «ДИНАМИЧЕСКИЙ МАССИВ»

```
import ctypes

class DynamicArray(object):

    def __init__(self):
        self.n = 0
        self.capacity = 1
        self.A = self.make_array(self.capacity)

    def __len__(self):
        return self.n

    def __getitem__(self, k):
        if not 0 <= k <self.n:
            return IndexError('K is out of bounds !')
        return self.A[k]

    def get_all(self):
        return [self.A[i] for i in range(self.n)]

    def append(self, ele):
        if self.n == self.capacity:
            self._resize(2 * self.capacity)
        self.A[self.n] = ele
        self.n += 1
```

```
def _resize(self, new_cap):
    B = self.make_array(new_cap)
    for k in range(self.n):
        B[k] = self.A[k]
    self.A = B
    self.capacity = new_cap

def make_array(self, new_cap):
    return (new_cap * ctypes.c_float)()
```

ПРИЛОЖЕНИЕ Б
(обязательное)
РЕАЛИЗАЦИЯ АЛГОРИТМОВ

```
import random
import time
from tkinter import *
from tkinter import messagebox
from Dyn_Array import DynamicArray

message = ""
error = 0

result = DynamicArray()

n = 0
fMax = 0
fMin = 0
figure_point = 0



class Poly():
    def __init__(self, coeff):
        self.coeff = coeff
        self.N = len(coeff)

    def evaluate(self, x):
        res = 0.0
        for i in range(self.N):
            res += self.coeff[i] * (x**(self.N-i-1))
```

```
        return res

    def printPoly(self):
        polinom=""
        for i in range(self.N):
            if i == self.N-1:
                polinom+=str(abs(self.coeff[i]))
            else:
                if self.coeff[i] != 0.0:
                    polinom+=str(abs(self.coeff[i]))
                    polinom+="*x^"
                    polinom+=str(self.N-i-1)
                    if self.coeff[i+1] > 0:
                        polinom+="+"
                    else:
                        polinom+="-"
        return polinom

    def take_integral(self, a, b):
        integral = 0
        for i in range(self.N):
            integral += self.coeff[i]*(b**((self.N)-i) - a**((self.N)-i))/((self.N)-i)
        return abs(integral)

p = Poly([])

def fn():
    global message, result, p, error
    del result
    result = DynamicArray()
```

```
    answer = 0
    if(len(coef_entry.get()) != 0):
        message = coef_entry.get()
        for x in message.split():
            try:
                result.append(float(x))
            except:
                messagebox.showinfo("Ошибка", "Данные должны быть числами и задаваться через пробел")
                error=1
                return
        p = Poly(result)
        answer = p.printPoly()
        labelText.set(answer)

def get_text():
    global p, fMax, fMin, error, figure_point, n

    if(len(n_entry.get()) == 0):
        messagebox.showinfo("Ошибка", "Заполните число испытаний!")
        return
    if(len(coef_entry.get()) == 0):
        messagebox.showinfo("Ошибка", "Заполните коэффициенты!")
        return
    try:
        n=int(n_entry.get())
        a=float(a_entry.get())
        b=float(b_entry.get())
    except:
```

```
        messagebox.showinfo("Ошибка", "Границы и число испытаний должны быть числами!")
        return
    if(a>=b):
        messagebox.showinfo("Ошибка", "Левая граница должна быть меньше правой!")
        return
    if(int(n_entry.get())<=0):
        messagebox.showinfo("Ошибка", "Число испытаний должно быть положительным!")
        return
    fn()
    if(error==1):
        error=0
        return

    fMin,fMax = functionOptimize(a,b)
    if(fMin<0 and fMax>0):
        messagebox.showinfo("Ошибка", "Функция не знакопостоянная!")
        return

    start = time.time()
    integral = p.take_integral(a,b)
    end = time.time()
    time_integral_text.set(end-start)

    start = time.time()
    prymoug = rect_integral(a,b,n)
    end = time.time()
    time_pryam_text.set(end-start)
```

```
    start = time.time()
    trapec = tr_integral(a,b,n)
    end = time.time()
    time_trapec_text.set(end-start)

    start = time.time()
    S1 = MCIntegrate(n,a,b)
    end = time.time()
    PointText.set(figure_point)
    time_Monte_text.set(end-start)

    answer_Integral_text.set(integral)
    answer_Monte_text.set(S1)
    answer_trapec_text.set(trapec)
    answer_pryam_text.set(prymoug)

    procent_trapec_text.set(abs(100 * (integral - trapec) / trapec))
    procent_pryam_text.set(abs(100 * (integral - prymoug) / prymoug))
    procent_monte_text.set(abs(100 * (integral - S1) / S1))

def rect_integral(xmin,xmax,n):
    dx=(xmax-xmin)/(n)
    area=0
    x=xmin
    for i in range(n):
        area+=dx*p.evaluate(x)
        x+=dx
    if area<0:
        area=-area
```

```
    return area

def tr_integral(xmin,xmax,n):
    dx=(xmax-xmin)/(n)
    area=0
    x=xmin
    for i in range(n):
        area+=dx*(p.evaluate(x)+p.evaluate(x+dx))/2
        x+=dx
    if area<0:
        area=-area
    return area

def functionOptimize(t1, t2):
    dt = 0.0001
    fMin = 0.0
    fMax = 0.0
    t = t1
    while (t <= t2):
        val = p.evaluate(t)
        if (val < fMin):
            fMin = val
        if (val > fMax):
            fMax = val
        t += dt
    return fMin, fMax

def MCIntegrate(nTrials, t1, t2):
    global figure_point
    figure_point = 0
```

```
    uBound, lBound = fMax, fMin
    #lBound, uBound = functionOptimize(t1, t2)
    result_MC = 0.0
    for i in range(nTrials):
        k = random.uniform(t1,t2)
        if (lBound<0):
            y = random.uniform(lBound,0)
        else:
            y = random.uniform(0,uBound)
        val = p.evaluate(k)
        if (val >= 0 and y >= 0 and y <= val):
            figure_point+=1
        elif (val < 0 and y >= val and y <= 0):
            figure_point+=1
    result_MC = (figure_point/nTrials)*(t2-t1)*(uBound)+(figure_point/nTrials)*(t2-t1)*(-lBound)
    return result_MC

root = Tk()

n = StringVar()
a = StringVar()
b = StringVar()
coef = StringVar()
labelText = StringVar()
PointText = StringVar()

time_integral_text = StringVar()
time_Monte_text = StringVar()
time_trapec_text = StringVar()
```

```
time_pryam_text = StringVar()
answer_Integral_text = StringVar()
answer_Monte_text = StringVar()
answer_trapec_text = StringVar()
answer_pryam_text = StringVar()
procent_trapec_text = StringVar()
procent_pryam_text = StringVar()
procent_monte_text = StringVar()

n_label = Label(text="Введите число испытаний :", font='bold')
coef_label = Label(text="Введите коэффициенты :", font='bold')
a_label = Label(text="Введите левую границу :", font='bold')
b_label = Label(text="Введите правую границу :", font='bold')
expr_label = Label(text="Выражение :", font='bold')
count_point = Label(text="Кол-во точек в фигуре :", font='bold')

n_label.grid(row=0, column=0, sticky="w")
coef_label.grid(row=1, column=0, sticky="w")
a_label.grid(row=2, column=0, sticky="w")
b_label.grid(row=3, column=0, sticky="w")
expr_label.grid(row=4, column=0, sticky="w")
count_point.grid(row=5, column=0, sticky="w")

time_Monte = Label(text="Время работы прямого расчета :", font='bold')
time_Monte.grid(row=6, column=0, sticky="w")

time_Monte = Label(text="Время работы Монте-Карло :", font='bold')
time_Monte.grid(row=7, column=0, sticky="w")

time_trapec = Label(text="Время работы трапеций :", font='bold')
```

```
time_trapec.grid(row=8, column=0, sticky="w")

time_pryam = Label(text="Время работы прямоугольников :", font='bold')
time_pryam.grid(row=9, column=0, sticky="w")

otst_1 = Label(font='bold')
otst_1.grid(row=10, column=0, sticky="w")

answer_integral = Label(text="Интеграл от полинома :", font='bold')
answer_integral.grid(row=11, column=0, sticky="w")

answer_Monte = Label(text="Ответ Монте-Карло :", font='bold')
answer_Monte.grid(row=12, column=0, sticky="w")

answer_trapec = Label(text="Ответ трапеций :", font='bold')
answer_trapec.grid(row=13, column=0, sticky="w")

answer_pryam = Label(text="Ответ прямоугольников :", font='bold')
answer_pryam.grid(row=14, column=0, sticky="w")

otst_2 = Label(font='bold')
otst_2.grid(row=15, column=0, sticky="w")

procent_trapec = Label(text="Отличие трапеций :", font='bold')
procent_trapec.grid(row=16, column=0, sticky="w")

procent_pryam = Label(text="Отличие прямоугольников :", font='bold')
procent_pryam.grid(row=17, column=0, sticky="w")

procent_Monte = Label(text="Отличие Монте-Карло :", font='bold')
```

```
procent_Monte.grid(row=18, column=0, sticky="w")

n_entry = Entry(textvariable=n,font=("Arial 16"))
coef_entry = Entry(textvariable=coef,font=("Arial 16"))
a_entry = Entry(textvariable=a,font=("Arial 16"))
b_entry = Entry(textvariable=b,font=("Arial 16"))

expr_show = Label(textvariable=labelText, font='bold',fg='#f00')
expr_show.grid(row=4, column=1, sticky="w")

point_show = Label(textvariable=PointText, font='bold')
point_show.grid(row=5, column=1, sticky="w")

time_integral_1 = Label(textvariable=time_integral_text, font='bold')
time_integral_1.grid(row=6, column=1, sticky="w")

time_Monte_1 = Label(textvariable=time_Monte_text, font='bold')
time_Monte_1.grid(row=7, column=1, sticky="w")

time_trapec_1 = Label(textvariable=time_trapec_text, font='bold')
time_trapec_1.grid(row=8, column=1, sticky="w")

time_pryam_1 = Label(textvariable=time_pryam_text, font='bold')
time_pryam_1.grid(row=9, column=1, sticky="w")

answer_Integral = Label(textvariable=answer_Integral_text, font='bold')
answer_Integral.grid(row=11, column=1, sticky="w")

answer_Monte = Label(textvariable=answer_Monte_text, font='bold')
answer_Monte.grid(row=12, column=1, sticky="w")
```

```
answer_trapec_1 = Label(textvariable=answer_trapec_text, font='bold')
answer_trapec_1.grid(row=13, column=1, sticky="w")

answer_pryam_1 = Label(textvariable=answer_pryam_text, font='bold')
answer_pryam_1.grid(row=14, column=1, sticky="w")

procent_trapec_1 = Label(textvariable=procent_trapec_text, font='bold')
procent_trapec_1.grid(row=16, column=1, sticky="w")

procent_pryam_1 = Label(textvariable=procent_pryam_text, font='bold')
procent_pryam_1.grid(row=17, column=1, sticky="w")

procent_Monte_1 = Label(textvariable=procent_monte_text, font='bold')
procent_Monte_1.grid(row=18, column=1, sticky="w")

point_show = Label(textvariable=PointText, font='bold')
point_show.grid(row=5, column=1, sticky="w")
n_entry.grid(row=0,column=1, padx=5, pady=5)
coef_entry.grid(row=1,column=1, padx=5, pady=5)
a_entry.grid(row=2,column=1, padx=5, pady=5)
b_entry.grid(row=3,column=1, padx=5, pady=5)

message_button = Button(text="Начать", command=get_text, font='bold')
message_button.grid(row=19, column=0, columnspan=2)

root.mainloop()
```