*Министерство образования и науки*

*Республики Казахстан*

*Петропавловский колледж*

*железнодорожного транспорта*

*СПЕЦИАЛЬНОСТЬ*

*«Вычислительная техника и программное обеспечение»*

*КУРСОВОЙ ПРОЕКТ*

*по предмету: «Моделирование производственных и экономических процессов»*

*на тему: «Создание модели для решения задач статистического моделирования методом Монте - Карло»*

*Проверил: Bыполнил:  
Петрищев К.Р. Хайдаев Я.Е.  
 гр. ПО-21*

*2015*СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ 4
   1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 7
      1. Постановка задачи 7
      2. Математическая реализация 8
   2. ПРАКТИЧЕСКАЯ ЧАСТЬ 15
      1. Описание программы 15
      2. Структура программы на языке С++ и UML-диаграмма 18
2. ЗАКЛЮЧЕНИЕ 24
3. СПИСОК ЛИТЕРАТУРЫ 25
4. ПРИЛОЖЕНИЕ А 26

ВВЕДЕНИЕ

Статистика наука, зародившаяся еще в Шумерском царстве и применяемая во множестве аспектов человеческой жизни, при необходимости рассчитать и сопоставить данные.

В рамках данного курсового проекта будет рассматриваться тема статистических методов путём решения одной из задач подходящей для применения подобных расчётов.

Статистика — отрасль знаний, в которой излагаются общие вопросы сбора, измерения и анализа массовых статистических (количественных или качественных) данных; изучение количественной стороны массовых общественных явлений в числовой форме.

Статистические методы — методы анализа статистических данных. Выделяют методы прикладной статистики, которые могут применяться во всех областях научных исследований и любых отраслях народного хозяйства, и другие статистические методы, применимость которых ограничена той или иной сферой. Имеются в виду такие методы, как статистический приемочный контроль, статистическое регулирование технологических процессов, надежность и испытания, планирование экспериментов.

Статистические методы анализа данных применяются практически во всех областях деятельности человека. Их используют всегда, когда необходимо получить и обосновать какие-либо суждения о группе (объектов или субъектов) с некоторой внутренней неоднородностью.

Целесообразно выделить три вида научной и прикладной деятельности в области статистических методов анализа данных (по степени специфичности методов, сопряженной с погруженностью в конкретные проблемы):

1. Разработка и исследование методов общего назначения, без учета специфики области применения;
2. Разработка и исследование статистических моделей реальных явлений и процессов в соответствии с потребностями той или иной области деятельности;
3. Применение статистических методов и моделей для статистического анализа конкретных данных.

Кратко рассмотрим три только что выделенных вида научной и прикладной деятельности. По мере движения от 1. к 3. сужается широта области применения конкретного статистического метода, но при этом повышается его значение для анализа конкретной ситуации. Если работам вида 1. соответствуют научные результаты, значимость которых оценивается по общенаучным критериям, то для работ вида 3. основное - успешное решение конкретных задач той или иной области применения (техники и технологии, экономики, социологии, медицины и других). Работы вида 2. занимают промежуточное положение, поскольку, с одной стороны, теоретическое изучение свойств статистических методов и моделей, предназначенных для определенной области применения, может быть весьма сложным и математизированным с другой - результаты представляют не всеобщий интерес, а лишь для некоторой группы специалистов. Можно сказать, что работы вида 2. нацелены на решение типовых задач конкретной области применения.

При применении статистических методов в конкретных областях знаний и отраслях народного хозяйства получаем научно-практические дисциплины типа "статистические методы в промышленности", "статистические методы в медицине" и др. С этой точки зрения эконометрика - это "статистические методы в экономике". Эти дисциплины группы 2. обычно опираются на вероятностно-статистические модели, построенные в соответствии с особенностями области применения. Весьма поучительно сопоставить вероятностно-статистические модели, применяемые в различных областях, обнаружить их близость и вместе с тем констатировать некоторые различия. Так, видна близость постановок задач и применяемых для их решения статистических методов в таких областях, как научные медицинские исследования, конкретные социологические исследования и маркетинговые исследования, или, короче, в медицине, социологии и маркетинге. Они часто объединяются вместе под названием «выборочные исследования».

Отличие выборочных исследований от экспертных проявляется, прежде всего, в числе обследованных объектов или субъектов – в выборочных исследованиях речь обычно идет о сотнях, а в экспертных – о десятках. Зато технологии экспертных исследований гораздо изощрённее. Еще более выражена специфика в демографических или логистических моделях, при обработке нарративной (текстовой, летописной) информации или при изучении взаимовлияния факторов.

Одним из примеров таких приемов является Монте-Карло – это группа численных методов, основанных на получении большого числа реализаций стохастического (случайного) процесса, который формируется таким образом, чтобы его вероятностные характеристики совпадали с аналогичными величинами решаемой задачи. Используется для решения задач в различных областях физики, химии, математики, экономики, оптимизации, теории управления и других.

В рамках курсового проекта будет детально рассмотрен метод Монте-Карло, а также разработана модель для автоматизации статистических вычислений.

В качестве примера применения данной модели рассмотрим задачу по нахождению площади многоугольника. Разработка модели и приложения ведется на языке С++ в интегрированной среде разработки Microsoft Visual Studio 2013.

1. АНАЛИТИЧЕСКАЯ ЧАСТЬ.
   1. Постановка задачи

Целью курсового проекта является разработка модели, позволяющей, реализовывать системы решения статистических задач в областях, где применим метод Монте-Карло.

Для начала рассмотрим час метод. Из определения следует что принцип действия заключается в многократном проведении испытания со случайными процессами, такими как генерирование случайных величин. И произведении расчетов над ними. Также одним из принципов данного метода сопоставление некоторых групп с соседними, либо же некого множества с более крупным множества. Исходя из, вышесказанного ясно что для реализации вычислительной модели потребуется генератор (псевдо)случайных чисел. Он как раз и необходим для создания выше названых областей значений в заданных пределах. Затем путем нахождения, внутренних величин, соответствующих определённым критериям и сопоставления числа удовлетворяющих – общему числу можно получить, некоторые аналитические данные.

Генераторы случайных чисел делятся на несколько типов: физические, табличные и алгоритмические. Для данной задачи наилучшим вариантом будет последний. Существуют различные алгоритмы вычисления псевдослучайных чисел, среди них чаще всего применяются метод срединных произведений и метод срединных квадратов. Оба эти метода основываются на произведении, соответствующих математических операций над заранее заданными числами и извлечении, срединных цифр из результата вычислений.

Так как для проведения какого-либо анализа требуются исходные данные, необходимо разработать метод ввода данных, из файла и/или из командной строки в качестве параметров. Аналогичные требования к системе вывода данных.

В качестве примера применения модели вычислений методом Монте-Карло, как уже говорилось ранее, будет реализовано приложение для нахождения площади произвольного, многоугольника на плоскости. Из этого следует что необходима математическая модель работы с фигурами и точками на плоскости.

Систематизируем задачи:

1. Разработать и реализовать генератор случайных чисел;
2. Разработать модель для расчетов, связанных с фигурами на плоскости;
3. Разработать систему ввода данных: ввод из командной строки, чтение из указанного в параметрах файла;
   1. Математическая реализация

Первым делом следует рассмотреть подробнее способы получения псевдослучайных чисел:

Метод срединных произведений состоит в том, что на вход подаются 2 или более чисел, эти числа перемножаются друг на друга, затем из полученного произведения извлекаются средние цифры. Традиционно генераторы случайных чисел генерируют числа от 0 до 1, рассмотрим пример работы:

Имеются некоторые четырехзначные числа R0 и R1. Эти числа умножаются друг на друга и заносится в R2. Далее для следующей операции берется R1 и из R2 берется середина (четыре средних цифры), снова умножаются, и записывается в R3. Затем данная процедура проделывается некоторое количество раз для обеспечения большей распределённости случайных величин при каждом вызове генератора случайных чисел. Следует отметим, что на самом деле в качестве случайного числа необходимо брать не ghij, а 0.ghij — с приписанным слева нулем и десятичной точкой. Вышеописанный процесс проиллюстрирован на рисунке 2.

R0 \* R1 = R2

R1 \* R2\* = R3

R2\* · R3\* = R3

0.R2\*

0.R3\*

Случайные числа

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| e | f | g | h | i | j | k | l |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| e | f | g | h | i | j | k | l |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| e | f | g | h | i | j | k | l |

**Рисунок 1 –** иллюстрация метода срединных произведений

Метод срединных квадратов выглядит следующим образом, на вход подаётся число, это число возводится в квадрат, затем из полученного результата, аналогично предыдущему методу извлекаются средние цифры. Рассмотрим пример работы данного способа:

Имеется некоторое четырехзначное число R0. Это число возводится в квадрат и заносится в R1. Далее из R1 берется середина (четыре средних цифры) — новое случайное число — и записывается в R0. Затем процедура повторяется. Отметим, что на самом деле в качестве случайного числа необходимо брать не ghij, а 0.ghij — с приписанным слева нулем и десятичной точкой.

R0

R1 = R0 \* R0

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c | d |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| e | F | g | h | i | j | k | l |

**Рисунок 2 –** иллюстрация метода срединных квадратов

Для гарантированного равномерного распределения случайных чисел, и не допущения повторных значений в качестве входных чисел будем применять системные миллисекунды, а так же числа от 1111 до 9999, сгенерированные стандартной функцией С++ – rand().

Перейдём к рассмотрению работы с фигурами. Задача предполагает нахождение площади фигуры методом Монте-Карло. На вход будут подаваться координаты фигуры(искомой). Затем её необходимо вписать в большую(контрольную) фигуру с известной площадью. Далее сгенерировать множество случайных точек, которые должны принадлежать контрольной фигуре. На следующем этапе необходимо вычислить количество, случайных точек, попавших в пространство искомой фигуры. Отношение числа точек принадлежащих, искомой фигуре в общему количеству, точек в контрольном многоугольнике, умноженная на площадь, контрольной фигуры будет приближенным значением, искомой площади. Путем проведения многократных испытаний и вычислением среднего значения получится наиболее точная величина площади.

Из вышесказанного следует, что при получении координат, необходимо выполнять проверку не корректных данных, например, координат прямой линии, так как линия не может иметь площадь. Это реализовывается следующим образом: Во-первых, многоугольник имеет как минимум три точки. Во-вторых, ни одна область, состоящая, из вышеназванных вершин, недолжна иметь площадь равную нулю. Значит для данной проверки разобьём многоугольник на области, состоящие из трёх точек и вычислим их площади, если ни одна из площадей не ровна нулю, то введены координаты многоугольника, иначе – прямой.

Необходим алгоритм проверки точки на принадлежность к области многоугольника. Для это из каждой случайной точки опустим лучи один параллельный оси абсцисс, другой – ординат. Далее если луч параллельный абсцисс будет пересекать четное количество сторон фигуры, больше нуля – точка расположена ниже фигуры, если количество пересечений равно нулю – точка выше фигуры, если же количество пересечений не четно, то координата точки по оси абсцисс находится в приделах фигуры. Аналогичные вычисления следует произвести с лучом параллельным оси ординат, если количество пересечений не четно, координата точки по ординатам находится в области многоугольника. В случае, соблюдения всех перечисленных условий точка располагается внутри данной фигуры. Данные операции следует произвести со всеми, сгенерированными, случайными точками. Рассмотрим пример определения принадлежности точки к фигуре:

На плоскости нарисован пятиугольник A(2; 4), B(2; 10), C(4; 14), D(6; 8), E(4; 4), а также точки M(1; 6), N(4; 4), K(5; 12). Проведем из каждой точки по два луча, один – вверх, другой – вправо. Как можно увидеть на рисунке 3 луч «a», выходящий из точки M не пересекает ни одну из сторон многоугольника ABCDE, но луч «b» пересекает две из них, значит точка M находится вне фигуры – левее. Луч «с», исходящий из точки N пересекает одну из сторон пятиугольника в точке C, а луч «d» – пересек сторону ED, следовательно, точка N расположена внутри области фигуры. Далее точка K, оба её луча не имеют ни одного пересечения, ни с одной из сторон ABCDE, так же на примере этой точки можно заметить, что если бы применялся метод сравнения координат случайных точек и координат вершин фигуры, то в отношении точек с подобным расположением не удалось бы получить верный ответ.

**Рисунок 3** – иллюстрация проверки точки на принадлежность к области фигуры

На основе выше описанных алгоритмов реализуется задача по нахождению площади фигуры. Что будет продемонстрировано, в следеющем примере:

Нахождение площади фигуры методом Монте-Карло. Определить методом Монте-Карло площадь пятиугольника с координатами углов (0, 0), (0, 10), (5, 20), (10, 10), (7, 0). Рисуется в двухмерных координатах заданный пятиугольник, вписываясь в прямоугольник, чья площадь, как нетрудно догадаться, составляет (10 – 0) • (20 – 0) = 200, рисунок 4.

**Рисунок 4** – иллюстрация нахождения площади

Используем таблицу случайных чисел для генерации пар чисел R, G, равномерно распределенных в интервале от 0 до 1. Число R будет имитировать координату X (0 ≤ X ≤ 10), следовательно, X = 10 · R. Число G будет имитировать координату Y (0 ≤ Y ≤ 20), следовательно, Y = 20 · G. Сгенерируем по 10 чисел R и G и отобразим 10 точек (X; Y) на рисунке 4 и в таблице 1

Таблица 1 - Решение задачи методом Монте-Карло

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер | R | G | X | Y | Точка (X; Y) попала в KLMN? | Точка (X; Y) попала в ABCDE? |
| 1 | 0,543374 | 0,554705 | 5,433739 | 11,0941 | Да | Да |
| 2 | 0,663769 | 0,32263 | 6,637686 | 6,452594 | Да | Да |
| 3 | 0,132668 | 0,325595 | 1,326676 | 6,511908 | Да | Да |
| 4 | 0,332327 | 0,490945 | 3,323272 | 9,818905 | Да | Да |
| 5 | 0,057314 | 0,112871 | 0,573144 | 2,257425 | Да | Да |
| 6 | 0,685227 | 0,856813 | 6,852274 | 17,13625 | Да | Нет |
| 7 | 0,868189 | 0,056411 | 8,681893 | 1,128222 | Да | Нет |
| 8 | 0,152591 | 0,888838 | 1,525913 | 17,77675 | Да | Нет |
| 9 | 0,635132 | 0,223465 | 6,351315 | 4,46931 | Да | Да |
| 10 | 0,880999 | 0,089966 | 8,809988 | 1,799325 | Да | Нет |
|  |  |  |  |  | 10 | 6 |

Статистическая гипотеза заключается в том, что количество точек, попавших в контур фигуры, пропорционально площади фигуры: 6:10 = S:200. То есть, по формуле метода Монте-Карло, получаем, что площадь S пятиугольника равна: 200 • 6/10 = 120. Проследим, как менялась величина S от опыта к опыту, таблица 2

Таблица 2 - Оценка точности ответа

|  |  |  |
| --- | --- | --- |
| Количество испытаний N | Оценка вероятности попадания случайной точки в испытуемую область | Оценка площади S методом Монте-Карло |
| 1 | 1/1 = 1.00 | 200 |
| 2 | 1/2 = 0.50 | 100 |
| 3 | 2/3 = 0.67 | 133 |
| 4 | 3/4 = 0.75 | 150 |
| 5 | 3/5 = 0.60 | 120 |
| 6 | 4/6 = 0.67 | 133 |
| 7 | 5/7 = 0.71 | 143 |
| 8 | 5/8 = 0.63 | 125 |
| 9 | 6/9 = 0.67 | 133 |
| 10 | 6/10 = 0.60 | 120 |

Точность результата растет с ростом количества испытаний, в этом и состоит метод Моте-Карло – проанализировать некоторое количество данных в диапазоне при некотором количестве испытаний и получить наиболее точные данные.

1. ПРАКТИЧЕСКАЯ ЧАСТЬ
   1. Описание программы

Первый элемент программы – генератор случайных чисел, он будет основан на совокупности нескольких источников псевдослучайных величин таких, как время, стандартный генератор случайных чисел и два вида алгоритмических.

Одним из алгоритмических методов генерирования случайных чисел будет – метод срединных произведений. В качестве первого множителя станет число, полученное путем сложения количества миллисекунд, полученных при двух запросах к системному времени. Второй множитель – число, сгенерированное стандартным генератором в диапазоне от 1111 до 9999. Количество шагов генерирования, также будет зависеть от количества миллисекунд.

Вторым алгоритмом станет метод срединных квадратов, в качестве начального числа, возводимого в квадрат, станет – сгенерированное стандартным ГСЧ, количество шагов генерации также будет зависеть от значения миллисекунд.

Далее все вышеперечисленные способы получения псевдослучайных чисел будет объединены, следующим образом: берется сумма системных миллисекунд, число, сгенерированное стандартным генератором, затем много кратно повторяется, следующие действия:

1. В генератор методом срединных произведений подаются выше полученные числа;
2. Генерируется число и подается в генератор срединных квадратов;
3. Последнее, в качестве второго операнда снова подаётся в генератор срединных произведений.

Последним действием полеченное четырехзначное число разделить на 10000. Таким образом можно получить величины, максимально распределенные в диапазоне от 0 до 1. Для получения случайных чисел в заданном диапазоне, сгенерированное число необходимо умножить на разность, максимального и минимального чисел диапазона, а затем, полученное произведение сложить, с максимальным числом диапазона.

Следующим звеном программы является математическая модель работы с фигурами. Любая фигура состоит из точек или вершин и сторон или ребер. Точки имеют лишь координаты на плоскости, но ребра, так как они являются линиями обладают коэффициентами углового и вертикального смещения. Коэффициенты должны вычисляться на этапе инициализации объекта ребро. Для вычисления данных коэффициентов следует вспомнить простое уравнение прямой

где

x, y – координаты точки;

k – коэффициент углового смещения;

b – коэффициент вертикального смещения

Множитель k для отрезка прямой вычисляется отношением разностей координат конечной и начальной точки, разность y1 и y0 делится на разность x1 и x0.

Вычисление b происходит путем подстановки известных координат любой из точек, коэффициента k и преобразования уравнения прямой, следующим образом:

Таким образом получается сторона фигуры со всеми необходимыми параметрами для вычисления столкновений, её и луча исходящего из случайной точки.

Для выявления столкновения луча и прямой есть два подхода. Первый – вычислить все точки, принадлежащие прямой, а так все точки, принадлежащие лучу на данном отрезке координат проверить каждую точку луча и прямей на совпадение координат. Второй способ – зная координаты случайной точки и коэффициенты прямой, путем подстановки в уравнение прямой можно узнать координаты пересечения, проверяемого отрезка и ребра фигуры. Так как необходимо проверять лучи, направленные вверх и вправо – расположение, проверяемой точки с расположением точки пересечения следует сравнить, следующим образом:

1. Если координата X точки, испускающей луч вправо меньше ординаты места пересечения линий – луч имеет пересечение, иначе пересечения нет.
2. Если координата Y точки, испускающей луч вверх меньше абсциссы места пересечения отрезков – пересечение луча и ребра существует, иначе пересечения с вертикальным лучом нет.

Как можно заметить второй метод более рационален по потребности в ресурсах памяти и времени выполнения, с минимальным количеством вычислений и проверок.

Для решения задачи по нахождению площадей фигур необходимо проверять точки на принадлежность фигуре. Выше описанный метод позволит проверить каждый луч, каждой случайной точки на пересечение, с каждым ребром фигуры. При наличии нечетного количества пересечений у обоих лучей точки – последняя считается, принадлежащей к фигуре.

Важной частью приложения является система ввода вывода. Здесь будет присутствовать как ввод параметров из командной строки, как и файловый ввод свойств многоугольника, а также вывод результата в виде текстового файла или строки в окне команд. Для указания пути к входному файлу будет обрабатываться параметр «-i» после, которого должен быть написан полный путь. Если координаты фигуры должны вводиться не из файла, а из командной строки вводится «-p» после чего пары координат, вершин. Чтобы указать файл для вывода обрабатывается аргумент «-o» вместе с полным путём, если же путь вывода не будет указан, результат вычислений отобразится в окне команд.

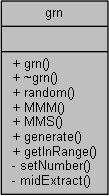
Неотъемлемой частью является модель метода Монте-Карло она будет непосредственно проводить статистический анализ: принимая параметры фигур, генерируя случайные точки при помощи генератора случайных чисел; подсчитывая количество точек, попавших в фигуру контрольную и испытуемую; вычислять соотношение числа точек в интересующем многоугольнике и контролирующем.

Само приложение будет проводить многократные испытания для последующего усреднения показаний, что даст наиболее точный результат.

* 1. Структура программы на языке С++ и UML-диаграмма

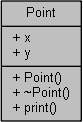
При создании программного решения была выбрана объектно-ориентированная парадигма программирования. Поэтому, все ключевые звенья были выделены в классы, а вычислительные процессы в методы данных классов. Так же применяется раздельное компилирование, то есть абстрактное объявление полей и методов класса находится в файле заголовка, а реализация – в файле исходного кода.

Один из основных элементов модели вычислений методом Монте-Карло является генератор случайных чисел. Данный класс был назван grn – generator of random numbers. Объекты данного класса являются статическими, это означает, что они не имеют полей, в них присутствуют лишь методы приватной и публичной степени защищённости. Схему данного класса можно увидеть на рисунке 5



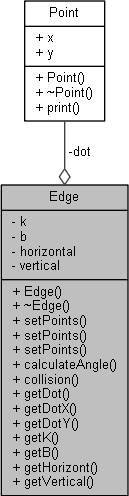
**Рисунок 5 –** схема класса grn

Модель обработки фигур состаит из нескильких классов: Point, Edge, Polyangle. Класс описыввающий точку содержит 2 открытых поля и конструктор класса, без каких-лиюбо других методов. Схема приведена на рисунке 6



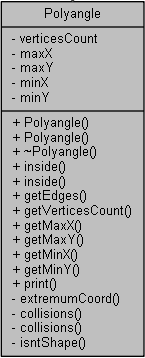
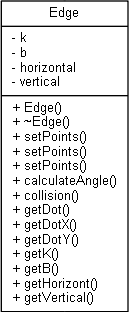
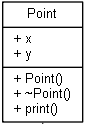
**Рисунок 6 –** схема класса Point

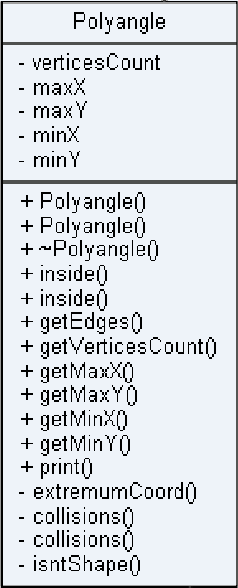
Класс представляющий ребро или сторону фигуры содержит в себе поле из указателя на объекты класса Point, а также поля, содержащие коэффициенты, присущие прямым, и переменные показывающие является ли линия вертикальной или горизонтальной. Для инициализации класса предусмотрены методы по вводу значений в поля – точки прямой, а также – вычисляющие коэффициенты прямой. Обеспечение доступа к полям, тоже происходит посредством публичных методов. Схема класса – рисунок 7.



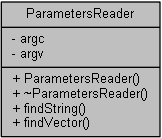
**Рисунок 7 –** схема класса Edge

Наибольшим звеном при работе с многоугольниками – является фигура – класс Polyangle. Он содержит в себе указатель на стороны фигур, которых может быть минимум три и до бесконечности. Среди полей присутствуют: количество вершин многоугольника, координат экстремума фигуры. Координаты фигуры инициализируются в конструкторе класса, который способен принимать указатель на ребра фигуры или указатель на массив вершин. Экстремум вычисляется приватным методом, сразу после инициализации всех рёбер. Существует функция объекта для проверки принадлежности точки к данной фигуре, которая принимает объект точка или координаты точки как целочисленные переменные. Если при инициализации фигуры были введены параметры прямой линии, функция isntShape() выбросит исключение, и программа напечатает сообщение о данной неполадке, для этого внутри данного класса присутствует внутренний класс – isLine. Схема класса приведена ни рисунке 8.



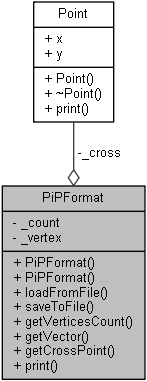
**Рисунок 8 –** схема класса Polyangle

Следующей важной частью – является система ввода команд и файлов и вывода файлов и сообщений. Её представляют классы ParameterReader и PiPFormat. Первый представляет набор методов для поиска среди параметров, нужной команды и возвращения, идущего после команды вектора чисел или строки. Вектор ищется функцией findeVector() и возвращиет std::vector<int>, если вектор значений пуст, функция вернёт исключение. Строка же находится при помощи функции findeString() и возвращает std::string, если строка после, указанного параметра пуста, функция вернёт исключение. Конструктор класса принимает количество параметров и строку, содержащую их.



**Рисунок 9 –** схема класса ParameterReader

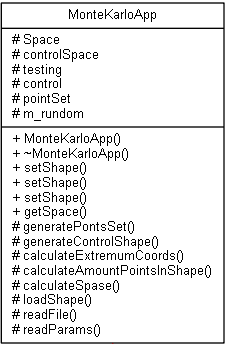
Второй элемент системы ввода вывода предназначен для работы с файлами. Он способен извлекать вектор чисел из файла и записывать в файл сформатированную строку при помощи методов, отведенных на это, в случае не удачного срабатывания функций предусмотрены исключения то момент, когда читаемый файл не найден, или путь вывода не доступен по каким-либо причинам.



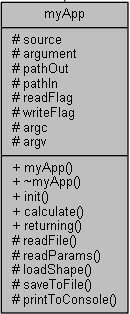
**Рисунок 10 –** схема класса PiPFormat

Главными частями приложения являются классы MonteKarloApp и производный от него – myApp, они выполняют все основные операции, а именно:

1. Инициализируют приложение;
2. Принимают параметры запуска и командной строки;
3. Создают фигуры – контрольную и испытуемую;
4. Генерируют случайные точки для вычислений;
5. И производят эти самые вычисления.

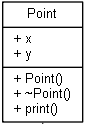
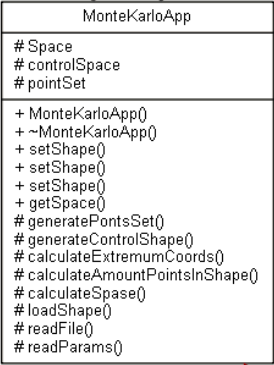


**Рисунок 11 –** схема класса MonteKarloApp



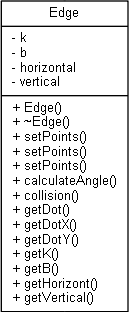
**Рисунок 12 –** схема класса myApp

Таким образом выполнена структура модели вычислений методом Монте-Карло и структура приложения по вычислению площадей фигур, созданной на основе первой. Некоторые ключевые моменты программного кода приведены в приложении А.

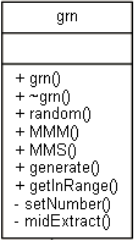


-dot

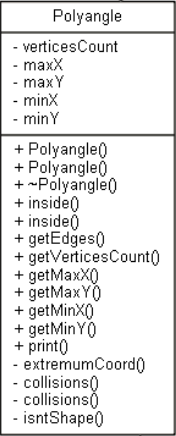
-\_cross



-edge

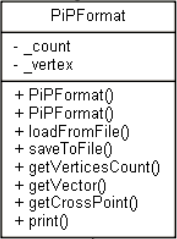


#m\_rundom

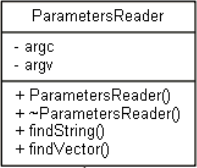


#testing

#coritrol



#source

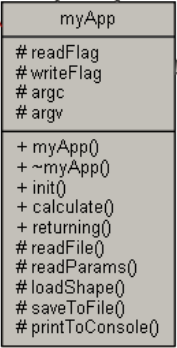


#argument



#pathOut

#pathln



**Рисунок 13 –** UML-диаграмма приложения

ЗАКЛЮЧЕНИЕ

В процессе работы над курсовым проектом был изучен принцип действия статистического метода анализа данных в определённом диапазоне. Разработан алгоритм, компьютерной программы. При разработке алгоритма была построена UML-диаграмма классов, необходимых, для реализации алгоритма компьютерной программы. Для программирования, в качестве инструментальных средств, выбран язык программирования С++. Была проведена работа по отладке программы, выявлению, исправлению ошибок и неточностей. В итоге получена реализация программы, позволяющей строить модель решения статистической задачи с применением метода Монте-Карло.

СПИСОК ЛИТЕРАТУРЫ

1. Н. Ш. Кремер. «Теория вероятностей и математическая статистика.» Москва, «Юнити», 2006;
2. Б. В. Гнеденко. «Теория вероятностей и математическая статистика.», Москва, «Наука», 1970;
3. С.М. Ермаков, Михайлов Г.А. «Статистическое моделирование.», Москва, «Наука», 1983;
4. Г. Крамер. «Математические методы статистики.», Москва, «Мир», 1975;
5. Р. Лафоре. «Объектно-ориентированное программирование в С++», Москва, «Питер», 2004;

ПРИЛОЖЕНИЕ A

grn.cpp

int grn::MMS(int n) // set dddd, sqr(dddd) = ddmmmmdd, mmmm is middle of square

{

int fNumber,//first number dddd

midSquare = 0; //mid of square

fNumber = n == 0 ? setNumber() : n;//dddd

int end = clockInThisSecond();

for (int i = 0; i < end; i++)

{

midSquare = midExtract(fNumber\*fNumber);

fNumber = midSquare == 0 ? midExtract(random()) : midSquare;

status(i, end, "drn::MMS");

}

return midSquare;

}

int grn::MMM(int n) // set dddd, dddd\*cccc = ddmmmmdd, mmmm is middle of multiplication

{

int fNumber,//first number dddd

sNumber,//second number cccc

midMultipl = 0; //mid of multiplication

fNumber = n | setNumber();//dddd

sNumber = random();//cccc

int end = clockInThisSecond();

for (int i = 0; i < end; i++)// for mutcher range

{

midMultipl = midExtract(fNumber \* sNumber);

fNumber = midMultipl;

if (midMultipl == 0)

{

fNumber = setNumber();

i--;

}

status(i, end, "drn::MMM");

}

return midMultipl;

}

float grn::generate()

{

float rNumber = 0.0;

intmidSquare = random(),

midMultipl = MMS();

int end = clockInThisSecond();

for (int i = 0; i < end ; i++)

{

midSquare = MMS(midMultipl);

midMultipl = MMM(midSquare);

status(i, end, "drn::generate");

}

rNumber = midMultipl;

rNumber /= 10000;

return rNumber;

}

edge.cpp

bool Edge::collision(Point p)

{

if (horizontal)

{

if (

p.x <= maxNumber(dot[0].x, dot[1].x) &&

p.x >= minNumber(dot[0].x, dot[1].x)

)

{

int y0 = k \* p.x + b;

return y0 >= p.y ? true : false;

}

else return false;

}

else if (vertical)

{

if (

p.y <= maxNumber(dot[0].y, dot[1].y) &&

p.y >= minNumber(dot[0].y, dot[1].y)

)

{

return dot[0].x >= p.x ? true : false;

}

else return false;

}

else

{

int x0,

y0;

if (

(

p.y <= maxNumber(dot[0].y, dot[1].y) &&

p.y >= minNumber(dot[0].y, dot[1].y)

)

||

(

p.x <= maxNumber(dot[0].x, dot[1].x) &&

p.x >= minNumber(dot[0].x, dot[1].x)

)

)

{

x0 = (p.y - b) / k;

y0 = k \* p.x + b;

return x0 >= p.x || y0 >= p.y ? true : false;

}

else return false;

}

return false;

}

polyangle.cpp

int Polyangle::collisions(Point p)

{

int collisions\_count = 0;

for (int i = 0; i < verticesCount / 2; i++)

{

if (edge[i].collision(p))

{

collisions\_count++;

}

status(i, verticesCount/2, "polyangle::collisions");

}

return collisions\_count;

}

bool Polyangle::inside(Point p)

{

// isntShape();

return collisions(p) % 2 == 0 ? false : true;

}

ParameterReader.cpp

string ParametersReader::findString(string opt)

{

std::string str;

if (argc > 1)

for (int i = 0; i < argc; i++)

{

string arg = string(argv[i]);

if (arg == opt)

{

if (

!(string(argv[i + 1])[0] == '-') &&

i + 1 <= argc

)

{

str = string(argv[i + 1]);

}

else throw Empty();

}

}

else

{

throw Empty();

}

if (str.empty())

throw Empty();

return " ";

}

std::vector<int> ParametersReader::findVector(string opt)

{

std::vector<int> v;

int sizeInt = sizeof(int)\*8,

\_sizeInt = -1 \* sizeInt;

if (argc > 1)

for (int i = 0; i < argc; i++)

{

string arg = string(argv[i]);

if (arg == opt)

{

if (

atoi(argv[i + 1]) >= \_sizeInt &&

atoi(argv[i + 1]) <= sizeInt

)

{

while (

(i + 1 >= argc) ? false :

(

atoi(argv[i + 1]) >= \_sizeInt

&& atoi(argv[i + 1]) <= sizeInt

)

)

{

v.push\_back(atoi(argv[i + 1]));

i++;

}

if (!v.empty())

return v;

else throw Empty();

}

else throw Empty();

}

}

else

{

throw Empty();

}

return v;

}