**Министерство науки и высшего образования Российской Федерации**

**ФГАОУ ВО «УрФУ имени первого Президента России Б. Н. Ельцина»**

**Кафедра прикладной математики и механики**

**Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Руководитель от УрФУ:**

**Кувшинов Д.Р.**

**Тема задания на практику**

**Построение алгоритма поиска выпуклой оболочки**

**ОТЧЕТ**

**Вид практики: учебная практика**

**Тип практики: учебная практика, практика по получению первичных**

**профессиональных умений и навыков.**

**Руководитель практики: к.ф.-м.н. Кувшинов Д.Р.**

**Студент: Васильев Вячеслав Сергеевич**

**Специальность (направление подготовки): 01.03.03 «Механика и**

**математическое моделирование»**

**Группа: МЕН-300705**

**Екатеринбург 2022**

**Оглавление**

Введение………………………………………………………………………...…2

Глава 1 Теоретические исследования выпуклой оболочки……………….……5

* 1. Понятия и описание выпуклой оболочки……………………………...…5
  2. Применение выпуклой оболочки……………………………………...…..5

Глава 2. Реализация метода построения выпуклой оболочки………..……...…6

2.1) Описание выбранного метода решения задачи и его построение……....................................................................................................…

2.2) Псевдокод…………………………………………………………………….8

2.2) Реализация алгоритма на С++……………………………………….........11

Список цитируемой и используемой литературы………..……………………………………………....…..……….....15

**Введение**

Задача построения выпуклых оболочек является одной из центральных для вычислительной геометрии. Она позволяет разрешить целый ряд других, иногда с первого взгляда не связанных с ней вопросов: построение диаграмм Вороного, построение триангуляций и т.д. Построение выпуклой оболочки конечного множества точек на плоскости довольно широко исследовано и имеет множество приложений распознавании образов, обработке изображений, задаче раскроя и компоновки материалов. Очень широко алгоритмы построения выпуклой оболочки используются геоинформатике и геоинформационных системах. В настоящее время известно достаточно большое число алгоритмов построения выпуклой оболочки.

Выпуклая оболочка - набора точек в n-мерном пространстве представляет собой наименьшее выпуклое множество, которое содержит все эти точки. Математически, это можно представить как выпуклую область, ограничивающую точки набора и не содержащую никаких внутренних углов или выпуклых вогнутостей.

Важность выпуклой оболочки в контексте анализа геометрических данных в n-мерном пространстве заключается в следующем:

1. Упрощение и сжатие данных.
2. Анализ формы и структуры данных.
3. Решение задач оптимизации.

Все это делает понятие выпуклой оболочки важным инструментом в анализе геометрических данных в n-мерном пространстве. Оно позволяет сжимать, представлять и анализировать данные, а также применять их в различных задачах оптимизации и обнаружения аномалий.

**Актуальность** работы связана с необходимостью изучения многих аспектов данной оболочки, а также с большим количеством вариантов применения.

**Объект исследования** — Надмножество вершин выпуклой оболочки.

**Цель** **работы** — Изучение особенностей, нахождение алгоритма поиска Надмножество вершин выпуклой оболочки, а также сравнение его уже имеющимися алгоритмами.

Для достижения поставленной цели необходимо решить ряд задач:

1. Изучить научную литературу по теме работы.
2. Разработать программу для работы с объектами.
3. Сравнить алгоритмы.

В качестве **методов** для решения поставленных задач использованы

1. Анализ информации из различных источников.
2. Алгоритмизация поиска решения

В качестве **материала** использованыопределения терминов из

**Структура работы**: работа состоит из введения, двух глав – теоретической, практической– заключения, списка использованной литературы и двух приложений.

**Глава 1. Теоретические исследования выпуклой оболочки**

**Понятия и описание выпуклой оболочки**

Выпуклая оболочка — это математическое понятие, которое описывает наименьшее выпуклое множество, содержащее заданный набор точек в n-мерном пространстве.

Выпуклое множество: Множество точек в n-мерном пространстве называется выпуклым, если для любых двух точек из этого множества отрезок, соединяющий эти точки, также лежит внутри этого множества. Другими словами, если для любых точек A и B из множества выпуклого множества и любого числа t в интервале [0, 1], точка tA + (1-t)B также принадлежит этому множеству.

Геометрически, выпуклая оболочка представляет собой выпуклый многогранник в трехмерном пространстве или выпуклый полигон в двумерном пространстве. Она обладает свойством того, что все точки внутри оболочки и на ее границе можно достичь, соединяя пары точек на границе оболочки прямыми линиями.

**Применение выпуклой оболочки**

Выпуклая оболочка имеет широкий спектр применений в различных областях. Например:

А) Геометрический анализ, где выпуклая оболочка предоставляет информацию о форме и структуре данных, что позволяет проводить геометрический анализ объектов или точек в пространстве. Это может быть полезно для изучения свойств объектов, их взаимодействий и визуализации.

Б) Обработка и сжатие данных: Построение минимальной выпуклой оболочки позволяет упростить сложные наборы данных, сократить их размерность и представить в более компактной форме. Это может быть полезно для эффективной обработки, хранения и передачи данных.

В) Компьютерная графика и визуализация: В области компьютерной графики и визуализации минимальная выпуклая оболочка используется для отображения сложных сцен, моделей и объектов. Она позволяет представить объекты в упрощенной форме, с сохранением их формы и основных характеристик.

Г) Оптимизация задач: Минимальная выпуклая оболочка может быть применена в задачах оптимизации, таких как оптимальное покрытие области или распределение ресурсов. Она позволяет найти оптимальное расположение точек или объектов с учетом заданных условий и ограничений.

Таким образом, мы видим, что выпуклая оболочка имеет большую сферу применений от геометрии и статистики до компьютерной визуализации.

**Глава 2.** **Реализация метода построения выпуклой оболочки**

**Описание выбранного метода решения задачи**

Для нахождения выпуклой оболочки был разработан алгоритм, позволяющий выделить надмножество вершин выпуклой оболочки.

Алгоритм рандомизации для построения надмножество вершин выпуклой оболочки имеет несколько преимуществ, вот некоторые из них:

А) Практическая эффективность: В большинстве случаев алгоритм рандомизации показывает хорошую производительность и работает достаточно быстро даже на больших наборах данных. Благодаря своей простоте и эффективности, он может быть использован в реальных приложениях с ограниченными вычислительными ресурсами.

Б) Масштабируемость: Алгоритм рандомизации хорошо масштабируется для работы с большими наборами данных. Его производительность не сильно зависит от размера входных данных, что позволяет использовать его для обработки как небольших, так и больших объемов информации.

Также у алгоритма есть и минусы такие как:

А) Зависимость от случайности: Алгоритм использует случайные числа при выборе направлений для построения пространственного симплекса. В некоторых случаях это может приводить к различным результатам при повторном запуске алгоритма. Более того, в некоторых особых ситуациях случайность может приводить к неудачным выборам направлений и значительно ухудшать результаты.

Б) Отсутствие информации о внутренних свойствах: Алгоритм рандомизации фокусируется только на внешних границах и форме выпуклой оболочки, не учитывая внутренние свойства данных. Если требуется получить информацию о взаимном расположении точек внутри оболочки или о других внутренних свойствах, этот алгоритм может быть недостаточным.

Алгоритм рандомизации (Случайного выбора) является эффективным и масштабируемым методом для построения надмножество вершин выпуклой оболочки. Он обладает преимуществами в виде хорошей производительности на больших наборах данных и возможностью использования в приложениях с ограниченными вычислительными ресурсами.

**Реализация алгоритма**

**Псевдокод**

Импортирование необходимых библиотек

Объявление глобальных переменных

Функция dot\_product(a, b):

Инициализация переменной res = 0

Цикл i от 0 до d:

res += a[i] \* b[i]

Возврат res

Функция det(m):

Если размер m равен 1:

Возврат m[0][0]

Создание матрицы matrix размером rows x cols

Заполнение матрицы matrix значениями из m

Возврат определителя matrix

Функция solve\_linear\_system(A, b):

Создание матрицы m размером d x (d + 1)

Заполнение матрицы m значениями из A и b

Цикл col от 0 до d:

Инициализация переменной row\_with\_max\_value = col

Цикл row от (col + 1) до d:

Если abs(m[row][col]) > abs(m[row\_with\_max\_value][col]):

row\_with\_max\_value = row

Обменять строки m[col] и m[row\_with\_max\_value]

Цикл row от (col + 1) до d:

Вычисление коэффициента c = m[row][col] / m[col][col]

Цикл j от col до (d + 1):

m[row][j] -= c \* m[col][j]

Создание списка x размером d

Цикл row от (d - 1) до 0 с шагом -1:

x[row] = m[row][d]

Цикл col от (row + 1) до d:

x[row] -= x[col] \* m[row][col]

x[row] /= m[row][row]

Возврат x

Функция is\_inside\_simplex(simplex, point):

Создание матрицы A размером d x d и списка b размером d

Заполнение матрицы A и списка b значениями из simplex и point

Вычисление коэффициентов coefs с помощью solve\_linear\_system(A, b)

Вычисление суммы коэффициентов sum\_coefs

Если все элементы coefs больше или равны -eps и sum\_coefs <= 1 + eps:

Возврат True

Возврат False

Функция find\_convex\_hull():

Создание списка res

Пока размер p больше или равен (d + 1):

Создание генератора случайных чисел gen

Создание равномерного распределения dis в диапазоне (-1, 1)

Создание списка simplex размером (d + 1)

Инициализация переменной found = False

Пока found равно False:

Создание списка dirs размером (d + 1) x d

Заполнение списка dirs случайными значениями

Для каждого i от 0 до (d + 1):

Инициализация max\_point\_index = -1

Инициализация max\_dot\_product = -INF

Для каждого j от 0 до размера p:

Вычисление dot\_product(dirs[i], p[j])

Если dp > max\_dot\_product:

Обновление max\_dot\_product = dp

Обновление max\_point\_index = j

simplex[i] = p[max\_point\_index]

Создание матрицы m размером d x d

Заполнение матрицы m значениями из simplex

Если abs(det(m)) > eps:

Обновление found = True

Добавление всех точек из simplex в res

Удаление точек из p, для которых is\_inside\_simplex(simplex, point) равно True

Возврат res

Функция generatePointsOnSphere(numPoints):

Создание списка points

Цикл i от 0 до numPoints:

Вычисление значения y

Вычисление значения radius

Вычисление значения theta

Вычисление значения x

Вычисление значения z

Добавление точки [x, y, z] в points

Цикл i от 0 до numPoints:

Вычисление значения y

Вычисление значения radius

Вычисление значения theta

Вычисление значения x

Вычисление значения z

Добавление точки [x, y, z] в points

Возврат points

Основная функция main():

Инициализация переменной d = 3

Инициализация переменной n = 100

Генерация точек на сфере с помощью generatePointsOnSphere(n)

Инициализация переменной p размером n x d и заполнение ее значениями из points

Нахождение выпуклой оболочки с помощью find\_convex\_hull()

Вывод размера convex\_hull

Для каждой точки point в convex\_hull:

Вывод координат точки

**Реализация на С++**

Для работы данной программы генерируются точки на сфере радиусом 1. Для этого использована библиотека <random>, которая псевдослучайно генерирует точки на данной сфере.

(Псевдослучайная последовательность чисел, это последовательность которая была вычислена по некоторому определённому арифметическому правилу, но имеет все свойства случайной последовательности чисел в рамках решаемой задачи.) Для данной задачи используется функция **generatePointsOnSphere,** которой на вход подается количество точек, и возвращает вектор, содержащий сгенерированные точки.

std::vector<std::vector<double>> generatePointsOnSphere(int numPoints) {

std::vector<std::vector<double>> points;

// Генерация точек на верхней полусфере

for (int i = 0; i < numPoints; ++i) {

double y = 1 - (i / static\_cast<double>(numPoints - 1)) \* 2;

double radius = sqrt(1 - y \* y);

double theta = 0.1 \* i;

double x = cos(theta) \* radius;

double z = sin(theta) \* radius;

points.push\_back({ x, y, z });

}

// Генерация точек на нижней полусфере

for (int i = 0; i < numPoints; ++i) {

double y = 1 - (i / static\_cast<double>(numPoints - 1)) \* 2;

double radius = sqrt(1 - y \* y) \* 0.2;

double theta = 0.1 \* i;

double x = cos(theta) \* radius;

double z = sin(theta) \* radius;

points.push\_back({ x, y, z });

}

return points;

}

Функция **find\_convex\_hull** находит выпуклую оболочку для заданного набора точек p. Она использует алгоритм рандомизации для построения выпуклой оболочки. Алгоритм работает следующим образом:

Пока вектор точек p содержит хотя бы d + 1 точек (для построения d-мерной выпуклой оболочки), выполняются следующие шаги:

Создается генератор случайных чисел gen и распределение dis для генерации случайных координат в интервале [-1, 1].

Создается вектор simplex размерности d + 1, который будет содержать вершины выпуклой оболочки.

Пока не будет найдено подходящее множество вершин выпуклой оболочки, выполняются следующие шаги:

Создается матрица dirs, содержащая случайно сгенерированные направления.

Для каждого направления из dirs находится точка в p с наибольшим скалярным произведением. Эти точки составляют вершины simplex.

Создается матрица m, которая содержит разности координат вершин simplex по каждой из осей.

Если определитель m по модулю больше eps, то найдено подходящее множество вершин и цикл прерывается.

Все вершины из simplex добавляются в результат res.

Из вектора p удаляются точки, которые находятся внутри построенного simplex.

vector<vector<double>> find\_convex\_hull() {

vector<vector<double>> res;

while (p.size() >= d + 1) {

mt19937 gen(random\_device{}());

uniform\_real\_distribution<> dis(-1, 1);

vector<vector<double>> simplex(d + 1);

bool found = false;

while (!found) {

vector<vector<double>> dirs(d + 1, vector<double>(d));

for (int i = 0; i < d + 1; ++i) {

for (int j = 0; j < d; ++j) {

dirs[i][j] = dis(gen);

}

}

for (int i = 0; i < d + 1; ++i) {

int max\_point\_index = -1;

double max\_dot\_product = -INFINITY;

for (int j = 0; j < p.size(); ++j) {

double dp = dot\_product(dirs[i], p[j]);

if (dp > max\_dot\_product) {

max\_dot\_product = dp;

max\_point\_index = j;

}

}

simplex[i] = p[max\_point\_index];

}

vector<vector<double>> m(d, vector<double>(d));

for (int i = 0; i < d; ++i) {

for (int j = 0; j < d; ++j) {

m[i][j] = simplex[j + 1][i] - simplex[0][i];

}

}

if (abs(det(m)) > eps) {

found = true;

}

}

for (const auto& point : simplex) {

res.push\_back(point);

}

p.erase(remove\_if(p.begin(), p.end(), [&](const vector<double>& point) {

return is\_inside\_simplex(simplex, point);

}), p.end());

}

return res;

}

Функции dot\_product, det и solve\_linear\_system являются вспомогательными функциями, используемыми в алгоритме рандомизации для нахождения выпуклой оболочки. Вот их краткое описание:

dot\_product(const vector<double>& a, const vector<double>& b): Эта функция вычисляет скалярное произведение двух векторов a и b. Она используется в алгоритме рандомизации для нахождения наиболее удаленных точек.

det(const vector<vector<double>>& m): Эта функция вычисляет определитель квадратной матрицы m. Она используется для проверки, является ли набор вершин simplex подходящим для построения выпуклой оболочки.

solve\_linear\_system(const vector<vector<double>>& A, const vector<double>& b): Эта функция решает линейную систему уравнений Ax = b, где A - матрица коэффициентов, b - вектор правых частей, x - вектор неизвестных. Она используется для определения коэффициентов, задающих положение точек внутри выпуклой оболочки.

В алгоритме рандомизации эти функции применяются для выполнения различных вычислений и проверок, необходимых для построения выпуклой оболочки. Например, вычисление скалярного произведения помогает определить наиболее удаленные точки, вычисление определителя проверяет, является ли множество вершин подходящим для построения выпуклой оболочки, а решение линейной системы позволяет определить положение точек внутри оболочки.

**Вывод**

В ходе учебной практики были изучены особенности построения выпуклой оболочки в n-мерном пространстве и реализован алгоритм рандомизации для данной задачи. Был проведен анализ его работы и сравнение с другими алгоритмами. Важными компонентами алгоритма являются использование случайных векторов. Алгоритм демонстрирует достаточно хорошую эффективность и применимость для нахождения выпуклой оболочки в n-мерном пространстве.

Список цитируемой и используемой литературы:

1. Петров, Н. Н. «Введение в выпуклый анализ». 2009 г.