МИНОБРНАУКИ РОССИИ
федеральное государственное автономное образовательное
учреждение высшего образования
«Омский государственный университет им. Ф.М. Достоевского»
Кафедра компьютерных технологий и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Богаченко Н. Ф.
«___» _____ 20__ г.

ОДНОПОТОЧНАЯ И МНОГОПОТОЧНАЯ РЕАЛИЗАЦИИ НЕКОТОРЫХ МЕТОДОВ ОПТИМИЗАЦИИ НАД DOUBLE, BIGDECIMAL И РАЦИОНАЛЬНЫМИ ЧИСЛАМИ

Выпускная квалификационная работа
по направлению 09.03.01 – Информатика и вычислительная техника

Научный руководитель:
преподаватель
_____Тюменцев Е. А.
«___» _____ 20__ г.

Выполнила:
студентка группы СИБ-801-О
_____ Смекаль Д. С.
«____» _____ 20__ г.

Омск
20__

Содержание

Введение

Оптимизация — процесс максимизации выгодных характеристик, соотношений (например, оптимизация производственных процессов и производства), и минимизации расходов [1].

В настоящее время во многих сферах экономики и программирования необходима оптимизация различных процессов. Для этого используются программы с библиотеками методов оптимизации.

Библиотека методов оптимизации может подойти для решения задач широкого спектра. Например, для оптимизации бизнеса, решения уравнений, машинного обучения, задач балансировки, составления автоматизированного расписания.

Многие доступные на данный момент библиотеки оптимизации иностранного производства, требуют установки, платные или имеют мало функций в бесплатной версии. Так же они не предоставляют отрытый исходный код, что затрудняет свободное использование методов оптимизации.

Одними из самых распространенных методов оптимизации являются методы решения задач линейного и нелинейного программирований. Многие задачи можно решить этими методами. Среди популярных методов линейного и нелинейного программирований можно выделить симплексные и градиентные методы.

Соответственно, целью дипломной работы стала реализация библиотеки методов оптимизации для решения задач линейного и нелинейного программирований.

Для достижения данной цели поставлен ряд задач:

1. Реализовать некоторые методы для решения задач линейного программирования, а конкретно симплексные методы: симплексный метод, метод искусственного базиса, двойственный симплекс-метод, метод Гамори.

2. Реализовать некоторые методы для решения задач нелинейного программирования, а именно градиентные методы: стохастический градиентный спуск, momentum, rmsprop, adam.
3. Сделать реализацию для однопоточного и многопоточного выполнений.
4. Сделать реализацию для рациональных чисел, BigDecimal и чисел с плавающей точкой.
5. Измерить время выполнения для разных реализаций и проанализировать полученные результаты.

Глава 1. Методы оптимизации

1.1. Характеристика методов решения задач оптимизации

Для решения конкретной задачи оптимизации прежде всего нужно выбрать математический метод, который приводил бы к конечным результатам с наименьшими затратами на вычисления или же давал возможность получить наибольший объем информации об искомом решении. Выбор того или иного метода в значительной степени определяется постановкой оптимальной задачи, а также используемой математической моделью объекта оптимизации.

В настоящее время для решения оптимальных задач применяют в основном следующие методы:

1. методы исследования функций классического анализа.
2. методы основанные на использовании неопределенных множителей Лангранжа.
3. вариационное исчисление.
4. динамическое программирование.
5. принцип максимума.
6. линейное программирование.
7. геометрическое программирование.
8. нелинейное программирование.

Как правило, нельзя рекомендовать какой-либо один метод, который можно использовать для решения всех без исключения задач, возникающих на практике. Одни методы в этом отношении являются более общими, другие - менее общими. Наконец, целую группу методов на определенных этапах решения оптимальной задачи можно применять в сочетании с другими методами.

Отметим также, что некоторые методы специально разработаны или наилучшим образом подходят для решения оптимальных задач с математическими моделями определенного вида.

Пожалуй, наилучшим путем при выборе метода оптимизации, наиболее пригодного для решения соответствующей задачи, следует признать исследование возможностей и опыта применения различных методов оптимизации [2].

1.2. Линейное программирование

Под линейным программированием понимается раздел теории экстремальных задач, в котором изучаются задачи минимизации (или максимизации) линейных функций на множествах, задаваемых системами линейных равенств и неравенств. Такие задачи обычно встречаются при решении вопросов оптимального планирования производства с ограниченным количеством ресурсов, при определении оптимального плана перевозок (транспортные задачи) и т. д.

Для решения большого круга задач линейного программирования имеется практически универсальный алгоритм - симплексный метод, позволяющий за конечное число итераций находить оптимальное решение подавляющего большинства задач. Под линейным программированием понимается раздел теории экстремальных задач, в котором изучаются задачи минимизации (или максимизации) линейных функций на множествах, задаваемых системами линейных равенств и неравенств.

Каждое из условий-неравенств определяет полупространство, ограниченное гиперплоскостью. Пересечение полупространств образует выпуклый n-мерный многогранник Q. Условия равенства выделяют из n-мерного пространства (n-l)-мерную плоскость, пересечение которой с областью Q дает выпуклый (n-l)-мерный многогранник G. Экстремальное

значение линейной формы (если оно существует) достигается в некоторой вершине многогранника. При вырождении оно может достигаться во всех точках ребра или грани многогранника. В силу изложенного для решения задачу линейного программирования теоретически достаточно вычислить значения функции в вершинах многогранника и найти среди этих значений наибольшее или наименьшее. Однако в практических задачах количество вершин области G настолько велико, что просмотр их даже с использованием ЭВМ невозможен.

Идея симплексного метода состоит в следующем. Отыскиваются некоторая вершина многогранника G и все ребра, выходящие из этой вершины. Далее перемещаются вдоль того из ребер, по которому функция убывает (при поиске минимума), и попадают в следующую вершину. Находят выходящие из нее ребра и повторяют процесс. Когда приходят в такую вершину, в которой вдоль всех выходящих из нее ребер функция возрастает, то минимум найден. Отметим, что, выбирая одно ребро, исключают из рассмотрения вершины, лежащие на остальных траекториях. В результате количество рассматриваемых вершин резко сокращается и оказывается посильным для ЭВМ. Симплекс-метод весьма эффективен и широко применяется для решения задач линейного программирования [2].

1.3. Нелинейное программирование

В большинстве инженерных задач построение математической модели не удается свести к задаче линейного программирования. Математические модели в задачах проектирования реальных объектов или технологических процессов должны отражать реальные протекающие в них физические и, как правило, нелинейные процессы. В результате, большинство задач математического программирования, которые

встречаются в научно-исследовательских проектах и в задачах проектирования – это задачи нелинейного программирования.

Одними из распространенных методов нелинейного программирования является градиентный спуск. Основываясь на выпуклых функциях, градиентный спуск итеративно корректирует некоторые из своих параметров, чтобы свести к минимуму конкретную функцию.

Специалисты по данным используют градиентный спуск для нахождения значений параметров функции, которые максимально снижают стоимость функций. Они начинают с определения значений начального параметра. В градиентном спуске используется вычисление для итеративной корректировки значений, чтобы минимизировать конкретную функцию затрат. Для полного понимания спуска градиента необходимо знать, что такое градиент.

Градиентом дифференцируемой функции f(x) в точке x[0] (1) называется n-мерный вектор f(x[0]), компоненты которого являются частными производными функции f(x), вычисленными в точке x[0].

$$df'(x[0]) = (\frac{df(x[0])}{dx_1}, \dots, \frac{df(x[0])}{dx_n}) \quad (1)$$

Вектор-градиент направлен в сторону наискорейшего возрастания функции в данной точке. Вектор, противоположный градиенту, называется антиградиентом и направлен в сторону наискорейшего убывания функции. В точке минимума градиент функции равен нулю. Использование этих методов в общем случае позволяет определить точку локального минимума функции [4].

Глава 2. Реализация библиотеки методов оптимизации

2.1 Типы данных

Для реализации алгоритмов мы решили рассмотреть три типа данных: double, BigDecimal и Fraction, и сравнить их работу. Рассмотрим подробнее эти типы данных.

Числа с плавающей точкой (float, double) хранят свои значения в памяти в экспоненциальном виде: знак-порядок-мантисса (см. рисунок 1). Этот факт приводит к неочевидным с точки зрения десятичной системы округлениям при математических операциях. Число двойной точности — компьютерный формат представления чисел, занимающий в памяти два машинных слова (в случае 32-битного компьютера — 64 бита или 8 байт). Числа двойной точности с плавающей запятой эквивалентны по точности числу с 15-17 значащими десятичными цифрами (в среднем 16,3) [5].

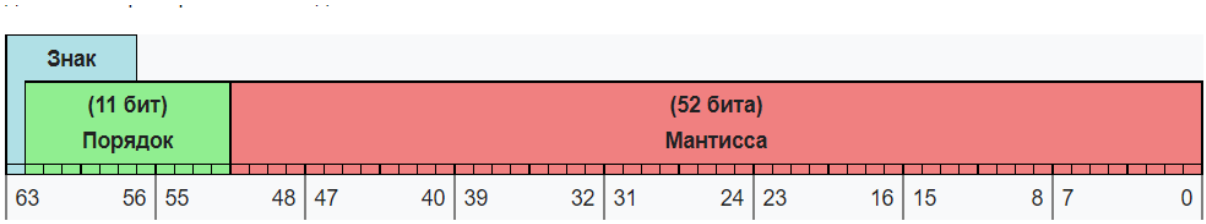

Рис. 1. Представление double

BigDecimal хранит неизменяемые десятичные числа произвольной точности со знаком. BigDecimal состоит из целочисленного немасштабированного значения произвольной точности (unscaled value) и 32-битного целочисленного масштаба (scale). Таким образом, значение числа, представленного BigDecimal может быть выражено по 2 формуле [3].

$$unscaledValue \times 10^{-scale} \quad (2)$$

Экземпляр BigDecimal можно создавать из числа с плавающей точкой и из строки. В первом случае преобразование из

экспоненциального в десятичный вид может привести к неожиданной точности, так что рекомендуется использовать строку. По умолчанию точность приблизительно соответствует точности 128-битных чисел с плавающей запятой IEEE (34 десятичных знака, HALF_EVEN округления) [7]. BigDecimal является хорошим вариантом там, где требуются точные расчеты.

Так же был реализован класс Fraction (см. рисунок 2), который может давать стопроцентную точность. Например, одна третья не может точно быть представлена в виде десятичного числа, так как это будет бесконечная десятичная дробь. Но при очень маленьких или огромных числах может возникнуть переполнение, так же функции класса выполняются медленно, поэтому не всегда целесообразно использовать дроби, но для симплексных методов они показали хорошие результаты.

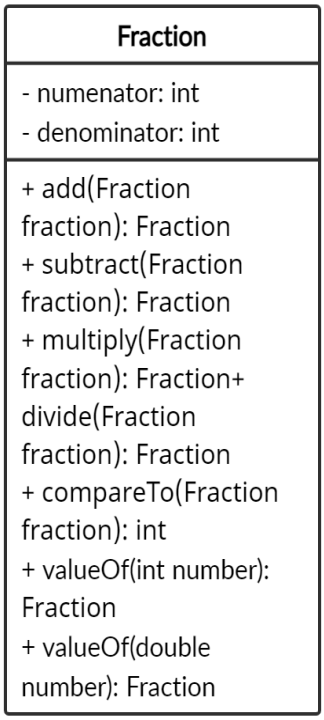

Рис. 2. Дробь

2.2 Реализация для линейного программирования

В качестве методов линейного программирования были выбраны симплексные методы: симплексный метод, метод искусственного базиса, двойственный симплекс метод, метод Гамори. Диаграмма классов симплексных методов изображена на 1 рисунке.

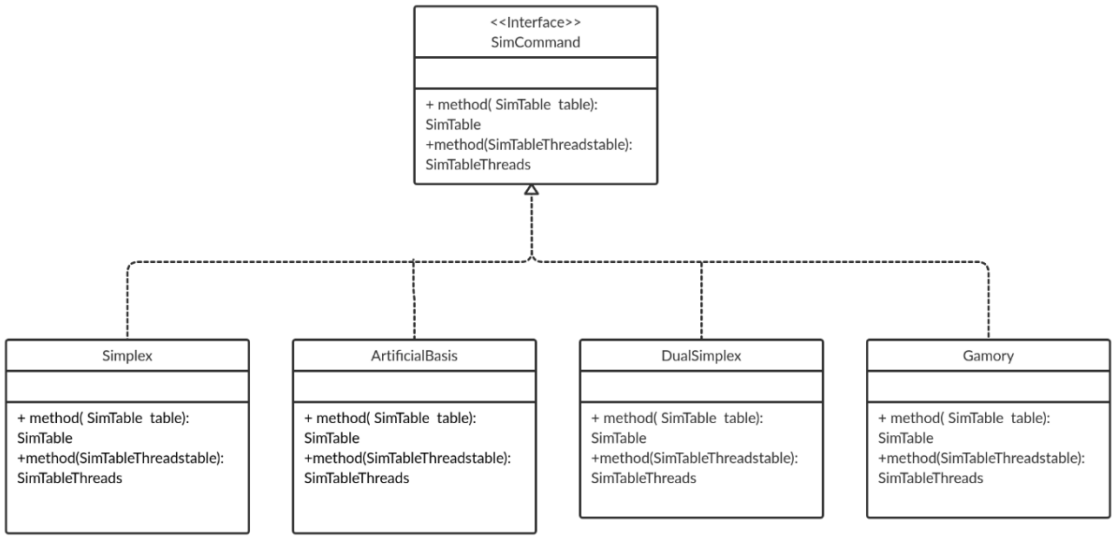

Рис. 3. Симплексные методы

SimCommand – общий интерфейс для симплексных алгоритмов, он состоит из двух функций. Функция method(SimTable table):SimTable принимает на вход симплексную таблицу, которая содержит нужные данные, и возращает на выходе оптимальный план в виде симплексной таблицы. Функция method(SimTableThreads table): SimTableThreads делает тоже самое, но работает на нескольких потоках. Все представленные на диаграмме методы, которые наследуются от интерфейса Command переопределяю эти две функции для работы своего алгоритма. Общая реализация симплексной таблицы приведена на 4 рисунке.

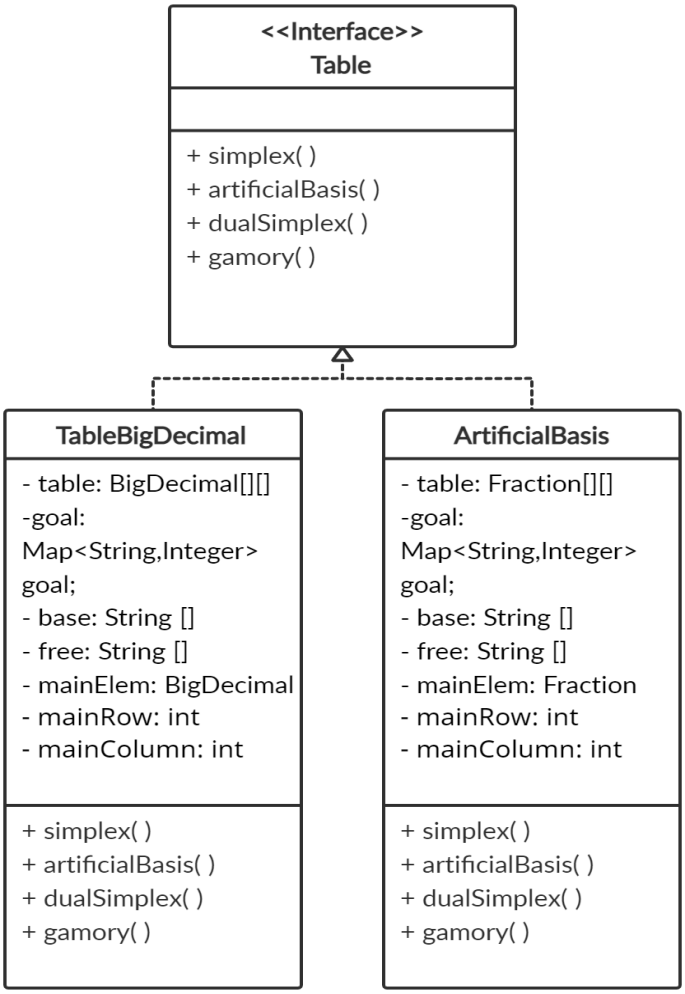

Рис. 4. Общий вид симплексной таблицы

Общая реализация метода на примере симплекс-метода представлена на 5 и 6 рисунках.

```
    @Override
     public SimTable method(SimTable simTable)  {
        simTable.simplex();
        return simTable;
    }
    @Override
     public SimTableThreads method(SimTableThreads table) throws InterruptedException {
        table.simplex();
        return table;
    }
}
```

Рис. 5. Методы для однопоточного и многопоточного выполнений

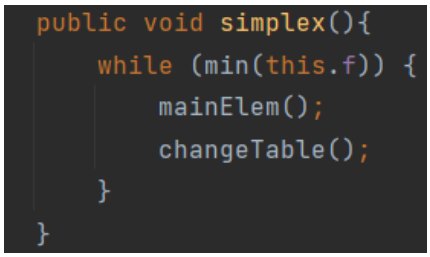

Рис. 6. Реализация симплекс-метода

Описание алгоритма симплекс-метода:

1. Функция min(f) выполняет проверку таблицы на оптимальность. Если в последней строке естьотрицательные числа (кроме свободного члена), то решение не оптимальное, и необходимо продолжить преобразования, а если нет отрицательных чисел, то решение оптимальное и можно закончить выполнение алгоритма.
2. Функция mainElem() осуществляет поиск разрешающего элемента.
3. Функция changeTable() производит преобразование таблицы.

2.3 Реализация для нелинейного программирования

Для реализации методов нелинейного программирования были выбраны методы градиентного спуска: стохастический градиентный спуск, momentum, rsmprop и adam (см. рисунок 7).

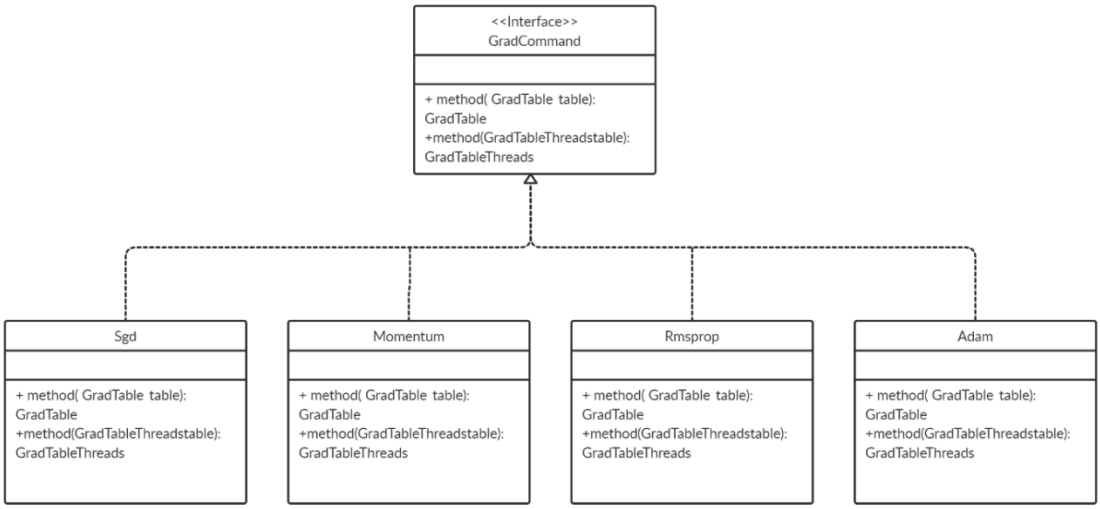

Рис. 7. Градиентные методы

Аналогично с линейными методами GradCommand – общий интерфейс для методов, он состоит из двух функций, которые так же реализуют в наследниках свои алгоритмы: method(GradTable table):GradTable и method(GradTableThreads table). Пример реализации методов на 8, 9 и 10 рисунках.

```
    @Override
    public GradTableBigDecimal method(GradTableBigDecimal table) {
        table.sgd();
        return table;
    }
}
```

Рис. 8. Метод для однопоточного выполнения

```
@Override
public GradTableBigDecimalThreads method(GradTableBigDecimalThreads table) throws InterruptedException{
    table.sgd();
    return table;
}
```

Рис. 9. Метод для многопоточного выполнения

```
public void sgd() {

    int iter = params.get("iter").intValue();
    boolean check = false;
    int length = x.length;
    BigDecimal lr = params.get("lr");
    BigDecimal stop = params.get("stop") != null ? params.get("stop") : null;

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


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

            nextX[j] = nextX[j].subtract(lr.multiply(dx[j].df(x)));
        }
        if (stop != null) {
            for (int k = 0; i < length; i++) {
                check = nextX[k].subtract(x[k]).abs().compareTo(stop) == -1;
            }
        }

        if (check) break;

        for (int l = 0; l < length; l++) {
            x[l] = new BigDecimal(String.valueOf(nextX[l]));
        }

    }
}
```

Рис. 10. Реализация стохастического градиентного спуска

Глава 3. Анализ полученных результатов

3.1 Инструменты для анализа

Было выполнено тестирование методов линейного и нелинейного программирований для сравнения скорости разных реализаций. Для замера времени использовалась встроенная функция System.nanoTime(), которая представляет время в наносекундах (см. рисунок 11). Для многопоточной реализации были использованы библиотеки: Executors, ExecutorService и Callable.

```
long start = System.nanoTime();
simplex.method(table);
long end = System.nanoTime();
long res = end - start;
```

Рис. 11. Измерение времени выполнения

Для каждого алгоритма проводилось 5 испытаний. В каждом испытании производилось по 150 замеров. Для адекватной обработки данных были исключены выбросы – результаты измерений, выделяющиеся из выборки. Выбросы отбирались с помощью критерия Шенона (3).

$$erfc\left(\frac{|p_i-\bar{p}|}{S_p}\right) < \frac{1}{2n} \quad (3)$$

Затем строили распределение. Для примера продемонстрируем распределение на стохастическом градиентном спуске для размера градиента n равным 5 (см. рисунок 12). Сначала проверим однопоточное выполнение над типом данных BigDecimal. На графике по оси абсцисс отложены значения времени для 150 замеров, по оси ординат количество повторений в наборе (сколько раз из 150 встречается значение t).

Из неравенства Чебышева следует, что вероятность того, что значения случайной величины отстоят от математического ожидания этой

случайной величины более чем на k стандартных отклонений составляет менее, чем в выражении на 4 формуле. В специальных случаях оценка может быть усилена. Так, например, как минимум в 95% случаях значения случайной величины, имеющей нормальное распределение, удалены от её среднего не более чем на два стандартных отклонения, а в примерно 99,7% - не более чем на три [6].

$$\frac{1}{k^2} \tag{4}$$

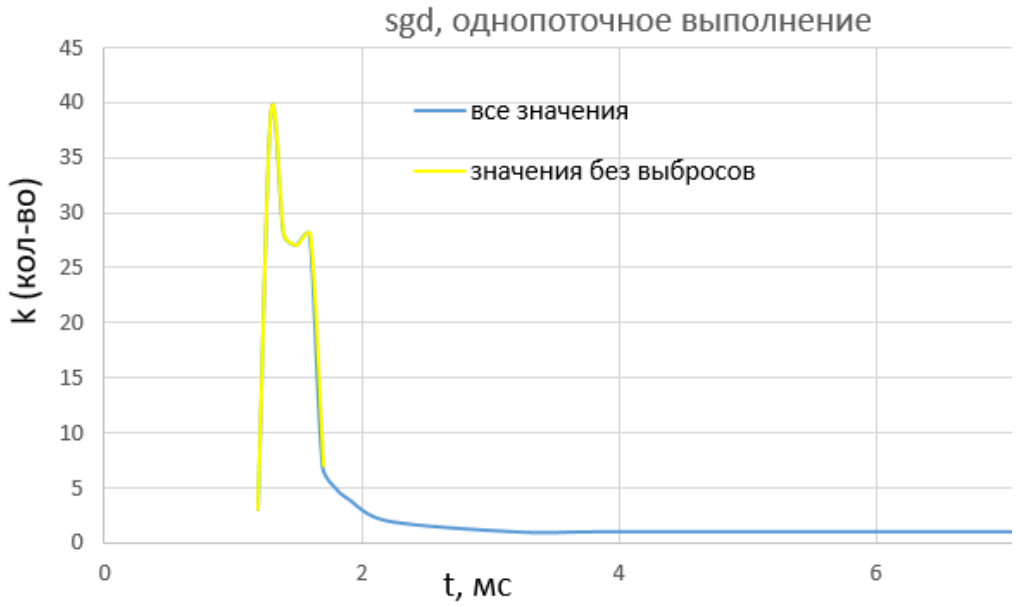

Рис. 12. График распределения для однопоточного выполнения

Пользуясь этим способом проверки, оценим наше распределение. После исключения выбросов остались значения, попадающие в диапазон от 1,2 до 1,7 миллисекунд. Дисперсия и стандартное отклонение данного набора соответственно равны 0,172 и 0,131. Математическое ожидание – 1,445 мс. Отложим допустимый диапазон прибавив и отняв от математического ожидания удвоенное стандартное отклонение. В результате мы получим диапазон в миллисекундах от 1,18 до 1,7. Наши значения вписываются в данный диапазон.

Так же мы можем заметить, что визуально график похож на нормальное распределение и дисперсия оказалась небольшой. Соответственно, можно считать такое распределение нормальным и брать математическое ожидание в качестве среднего значения для сравнения времени выполнения.

Аналогичным способом оценим распределение для многопоточного выполнения (см. рисунок 13), остальные характеристики (объем данных, алгоритм и.т.д) оставим прежними.

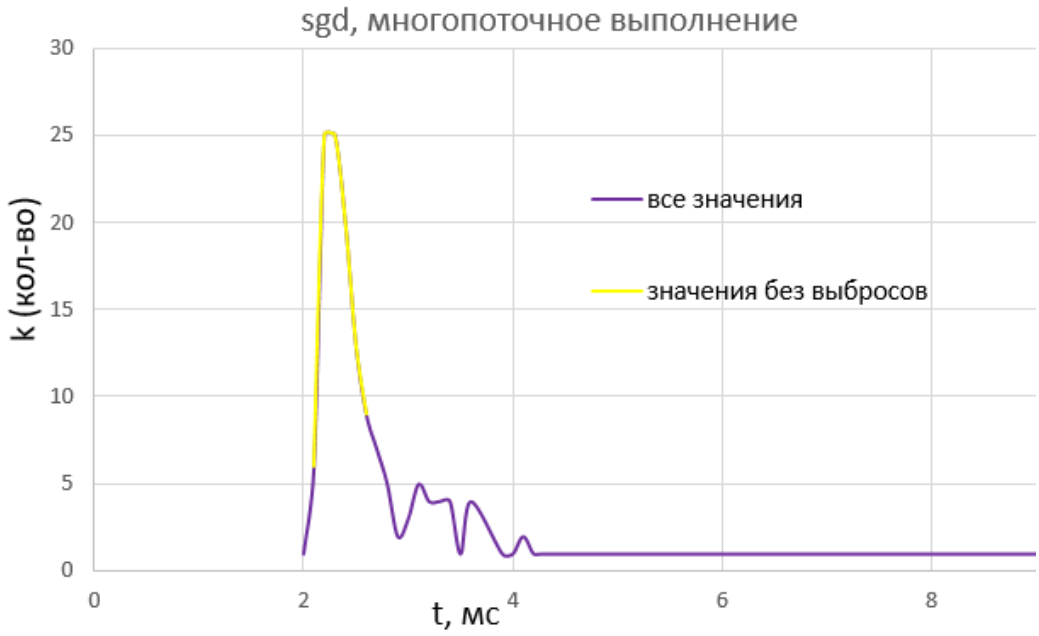

Рис. 13. График распределения для многопоточного выполнения

После исключения выбросов остались значения, попадающие в диапазон от 2,1 до 2,6 миллисекунд. Дисперсия и стандартное отклонение соответственно равны 1,93 и 1,38. Математическое ожидание – 2,337 мс. Отложим допустимый диапазон прибавив и отняв от математического ожидания удвоенное стандартное отклонение. В результате мы получим диапазон в миллисекундах от 2,06 до 2,61. Значения вписались в данный диапазон. Так же небольшая дисперсия и визуально график совпадает с нормальным распределением. Будем считать такое распределение нормальным.

Далее проделаем тоже самое для симплекс-методов для однопоточного и многопоточного выполнений над типом BigDecimal при размерности симплексной таблицы 16 на 16.

Сначала проанализируем однопоточную реализацию (см. рисунок 14). После исключения выбросов остались значения, попадающие в диапазон от 50 до 80 микросекунд. Дисперсия и стандартное отклонение соответственно равны 8,5 и 9,2. Математическое ожидание – 64,4 мкс. Прибавим удвоенное стандартное отклонение и в результате мы получим диапазон в микросекундах от 46 до 82,8. Значения вписались в данный диапазон. Небольшая дисперсия и визуально график похож на нормальное распределение.

Теперь рассмотрим многопоточную реализацию (см. рисунок 15). После исключения выбросов остались значения, попадающие в диапазон от 0,175 до 0,24 миллисекунд. Дисперсия и стандартное отклонение соответственно равны 0,06 и 0,025. Математическое ожидание – 0,198 мс. Прибавим удвоенное стандартное отклонение и получим диапазон в миллисекундах от 0,173 до 0,248. Значения вписались в данный диапазон. Приемлемая дисперсия и визуально график в целом похож на нормальное распределение.

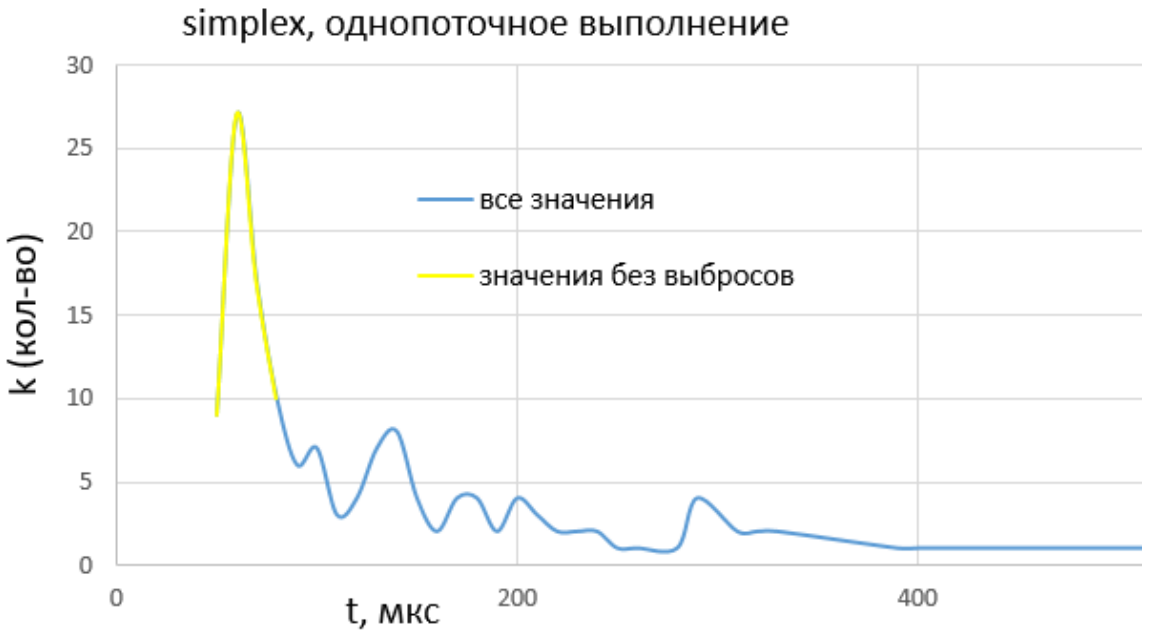

Рис. 14. График распределения для однопоточного выполнения

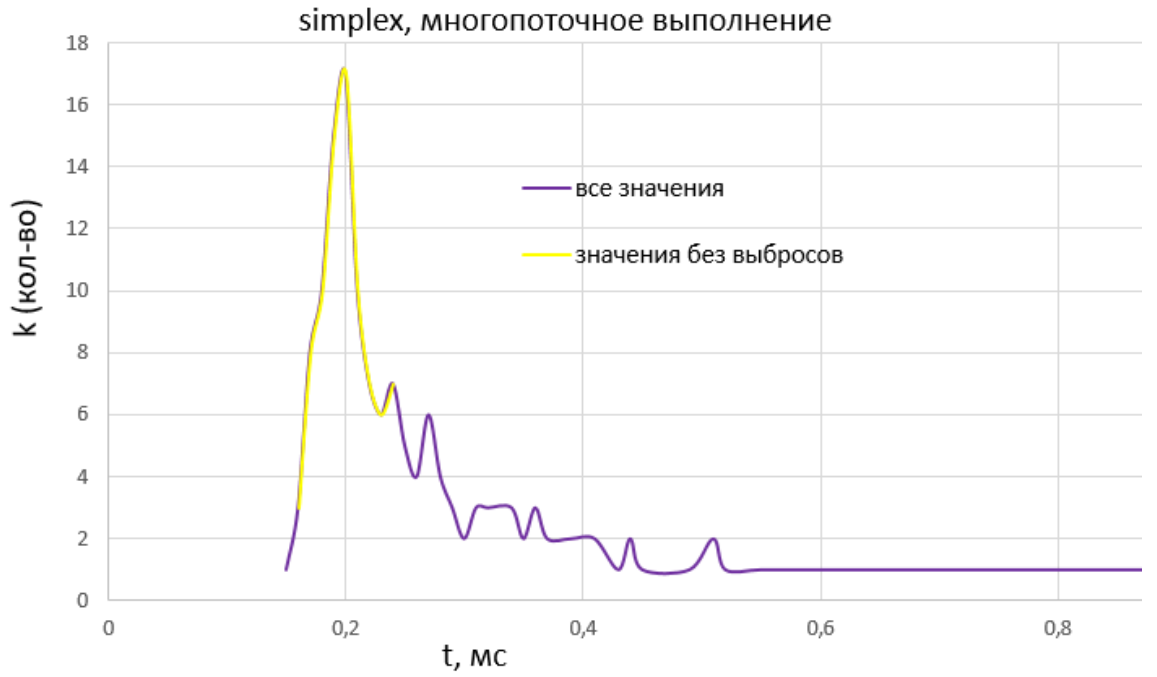

Рис. 14. График распределения для однопоточного выполнения

3.2. Анализ результатов методов линейного программирования

По результатам тестирования однопоточной и многопоточной реализаций (см. рисунок 16) для методов решения задач линейного программирования над типом данных BigDecimal было обнаружено, что выгода от использования нескольких потоков наступает не ранее размерности таблицы 23 на 23. Поэтому многопоточное выполнение имеет смысл только для очень больших данных. Но так как симплексные методы являются np-трудными имеет смысл все же иметь многопоточную реализацию. Однако при использовании дробей многопоточная реализация раньше становится целесообразной (см. рисунок 17), примерно начиная с размерностей таблицы 16 на 16.

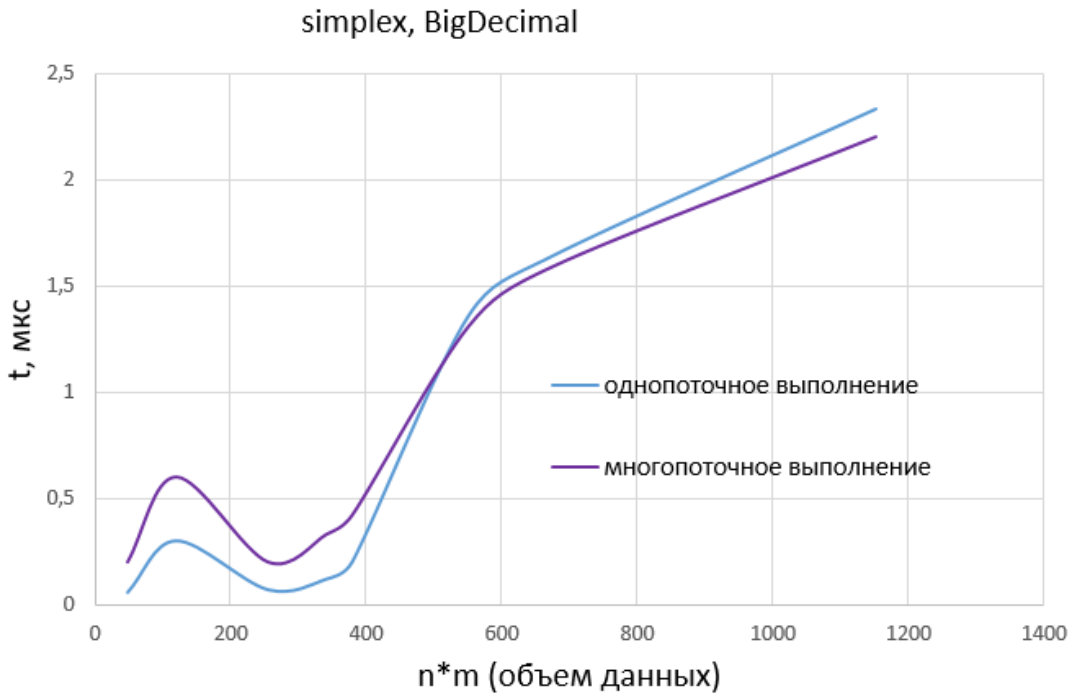

Рис. 16. График для многопоточной и однопоточной реализаций

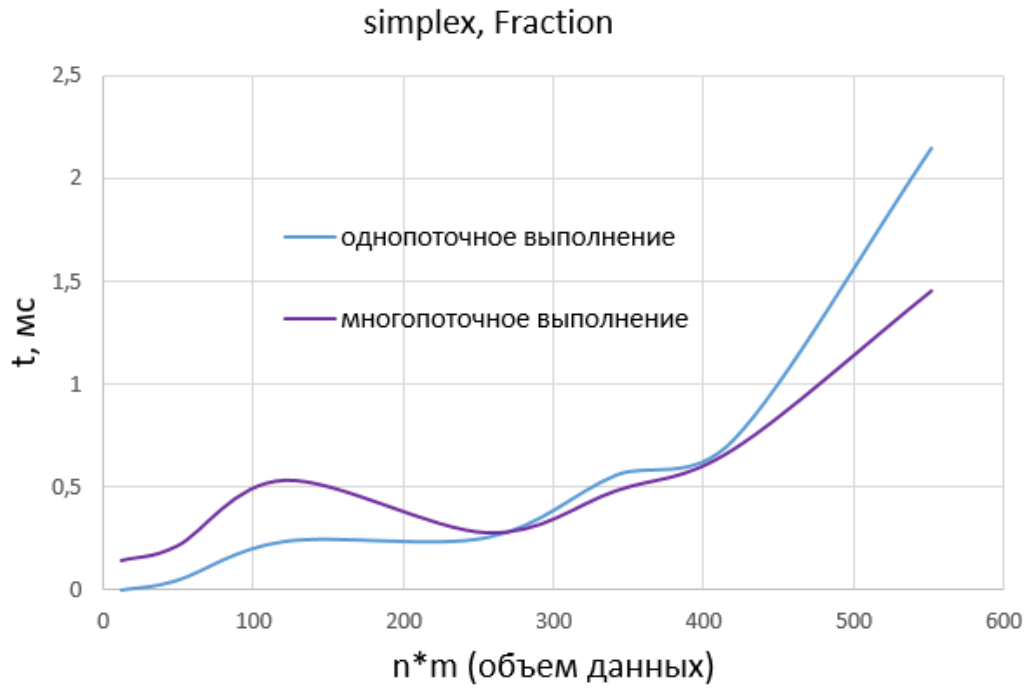

Рис. 17. График для многопоточной и однопоточной реализаций

3.3. Анализ результатов методов нелинейного программирования

Было проведено тестирование методов решения задач нелинейного программирования для сравнения эффективности однопоточного и многопоточного выполнений. Для замера времени использовался тот же способ, что и для методов решения задач линейного программирования.

В результате тестирования многопоточной и однопоточной реализации стохастического градиентного спуска (см. рисунок 18) над типом данных BigDecimal было обнаружено, что многопоточная реализация с определенного момента становится эффективней. Выигрыш в скорости выполнения начинается с девяти переменных. Для более сложных алгоритмов градиентного спуска, эффективность многопоточного выполнения заметна гораздо раньше. Для примера приведем график алгоритма momentum (см. рисунок 19).

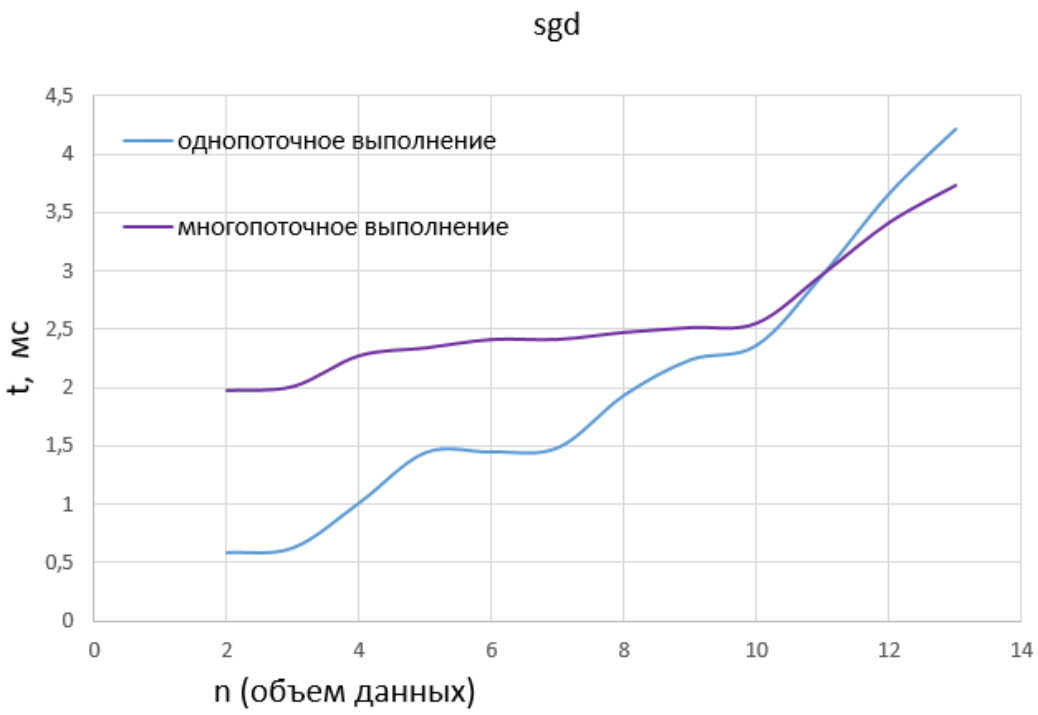

Рис. 18. График для многопоточной и однопоточной реализаций

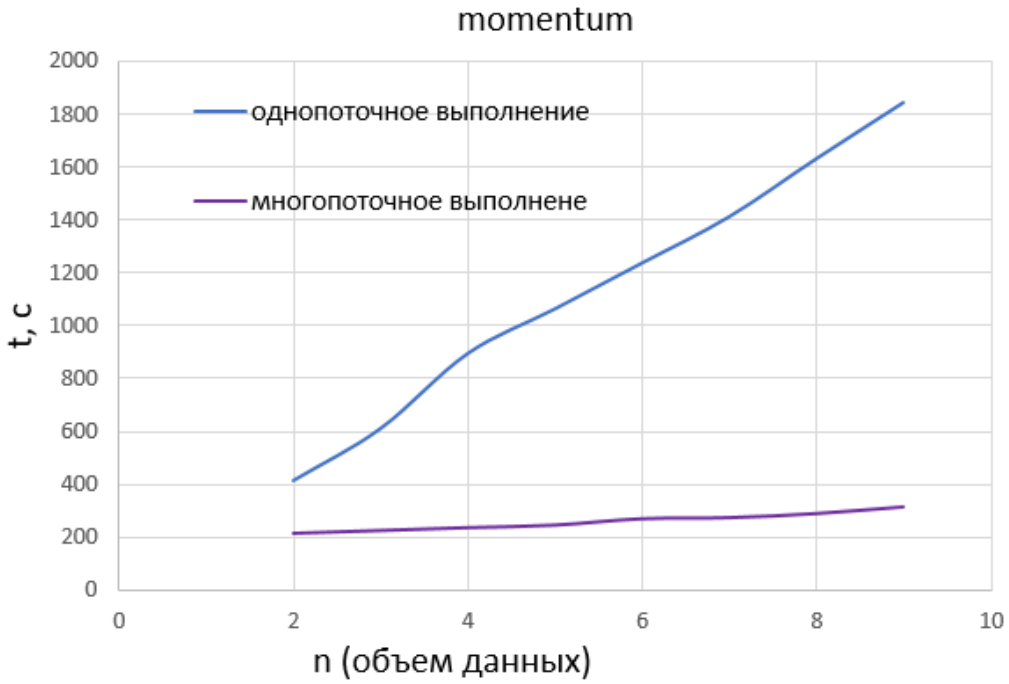

Рис. 19. График для многопоточной и однопоточной реализаций

Заключение

В ходе выпускной квалификационной работы была выполнена однопоточная и многопоточная реализации некоторых методов оптимизации над double, BigDecimal и рациональными числами.

Основные результаты:

1. Были реализованы симплексные методы для решения задач линейного программирования.
2. Были реализованы градиентные методы для решения задач нелинейного программирования.
3. Сделаны реализации для однопоточного и многопоточного выполнений.
4. Выполнены реализации для BigDecimal, рациональных чисел и чисел с плавающей точкой.
5. Измерено время выполнения для разных реализаций и проанализированы полученные результаты.

В результате все поставленные задачи были выполнены.

Список использованной литературы:

1. Оптимизация [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Оптимизация (дата обращения: 03.06.2022).
2. Трифонов А.Г. Постановка задачи оптимизации и численные методы ее решения [Электронный ресурс]. URL: https://hub.exponenta.ru/post/postanovka-zadachi-optimizatsii-i-chislennye-metody-ee-resheniya356 (дата обращения: 03.06.2022).
3. Класс BigDecimal [Электронный ресурс]. URL: https://docs.oracle.com/javase/1.5.0/docs/api/java/math/BigDecimal.html (дата обращения: 20.05.2022).
4. Градиентные методы решения задач нелинейного программирования [Электронный ресурс]. URL: https://studfile.net/preview/3397141/ (дата обращения: 04.06.2022).
5. Число двойной точности [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki /Число_двойной_точности (дата обращения: 13.06.2022).
6. Дисперсия случайной величины [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Дисперсия_случайной_величины (дата обращения: 09.06.2022).
7. Классы BigInteger и BigDecimal [Электронный ресурс]. URL: https://itsobes.ru/JavaSobes/klassy-biginteger-i-bigdecimal/ (дата обращения: 09.06.2022).