

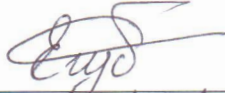
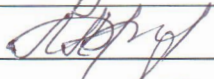


ИНСТИТУТ ИТКН

КАФЕДРА ИНЖЕНЕРНОЙ КИБЕРНЕТИКИ

НАПРАВЛЕНИЕ 01.03.04 ПРИКЛАДНАЯ МАТЕМАТИКА

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему: Трехкомпонентные методы сопряженного градиента для решения задач
оптимизации большой размерности

Студент		П.А. Егубова
Руководитель работы		Л.А. Артемьева
Нормоконтроль проведен		А.С. Островская
Проверка на заимствования проведена		Г.В. Круглова

Работа рассмотрена кафедрой и допущена к защите в ГЭК

И.о. заведующего кафедрой		А.Р. Ефимов
Директор института		С.В. Солодов

Москва июнь 2021

Институт ИТКН

Кафедра Инженерной кибернетики

Направление 01.03.04 Прикладная математика

УТВЕРЖДАЮ 

Зав. Кафедрой Ефимов А.Р.

«22» декабря 2020г.

**ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ
КВАЛИФИКАЦИОННОЙ РАБОТЫ БАКАЛАВРА**

Студенту группы БПМ-17-1 Егубовой Полине Андреевне

1. Тема работы: «Трехкомпонентные методы сопряженного градиента для решения задач оптимизации большой размерности».

2. Цель работы: «Разработать алгоритмическое и программное обеспечение для решения задач оптимизации большой размерности с помощью трехкомпонентного метода сопряженного градиента. Провести сравнительный анализ полученных результатов с результатами работы других методов на том же множестве исходных задач».

3. Исходные данные: множество функций большой размерности различного типа из библиотеки CUTE (The Constrained and Unconstrained Environment).

4. Основная литература, в том числе:

4.1 Jorge Nocedal, Stephen J. Wright. Numerical Optimization. New York, Springer Science + Business Media, LLC, 2006;

4.2 Optimization Methods and Software. Great Britain, Taylor & Francis, 1992 - 2020;

4.3 International Journal of Mathematical Modelling and Numerical Optimisation. Switzerland, Inderscience Publishers, 2009 – 2020.

5. Перечень основных этапов исследования и форма промежуточной отчетности по каждому этапу:

5.1 Аналитический обзор литературы;

5.2 Формулировка содержательной и математической постановки задачи;

- 5.3 Разработка функциональной схемы системы;
 - 5.4 Программная реализация;
 - 5.6 Отладка и тестирование;
 - 5.7 Анализ полученных результатов;
 - 5.8 Подготовка текста дипломной работы.
6. Аппаратура и методики, которые должны быть использованы в работе:
- 6.1 Методы оптимизации;
 - 6.2 Программирование.
7. Использование ЭВМ:
- 7.1 Установка библиотеки CUTE на виртуальной машине с операционной системой Linux Ubuntu, развернутой при помощи программного обеспечения VMWare Workstation;
 - 7.2 Разработка программного обеспечения на языке программирования Python в среде разработки PyCharm.
8. Перечень (примерный) основных вопросов, которые должны быть рассмотрены и проанализированы в литературном обзоре:
- 8.1 Существующие методы оптимизации для решения задач большой размерности и их модификации;
 - 8.2 Алгоритм линейного поиска для вычисления длины шага метода;
 - 8.3 Основные характеристики работы методов оптимизации;
 - 8.4 Способы представления результатов работы методов.
9. Перечень (примерный) графического и иллюстрированного материала:
- 9.1 Схема разработанного алгоритма;
 - 9.2 Примеры исходных данных;
 - 9.3 Схема разработанного программного обеспечения;
 - 9.4 Результаты и их анализ.

Руководитель работы к.ф.-м.н., доцент, Артемьева Людмила Анатольевна

(подпись)

Дата выдачи задания: «22» декабря 2020 г.

Задание принял к исполнению студент

(подпись)

АННОТАЦИЯ

Выпускная квалификационная работа изложена на 75 страницах, содержит 13 рисунков, 1 таблицу, список использованных источников из 27 наименований, 3 приложений.

Ключевые слова: ТРЕХКОМПОНЕНТНЫЕ МЕТОДЫ ОПТИМИЗАЦИИ, МЕТОДЫ СОПРЯЖЕННОГО ГРАДИЕНТА, ТОЧКА МИНИМУМА, БОЛЬШАЯ РАЗМЕРНОСТЬ, ЧИСЛЕННЫЕ ЭКСПЕРИМЕНТЫ, ПРОФИЛЬ ПРОИЗВОДИТЕЛЬНОСТИ, ЛИНЕЙНЫЙ ПОИСК, УСЛОВИЯ АРМИХО, УСИЛЕННЫЕ УСЛОВИЯ ВОЛЬФЕ, PYTHON, CUTEST

Выпускная квалификационная работа посвящена исследованию трехкомпонентных методов сопряженного градиента для решения задач оптимизации.

Целью работы является разработка нового трехкомпонентного метода сопряженного градиента для решения задач оптимизации.

Результатом проведенного исследования является новый метод оптимизации, а также оценка качества его работы, полученная вследствие сравнения результатов серии численных экспериментов данного метода на множестве тестовых задач с характеристиками работы других алгоритмов.

Проведенные численные эксперименты доказали, что разработанный метод является крайне конкурентноспособным среди других методов сопряженного градиента. Данный факт позволяет рекомендовать полученный метод к использованию для решения задач оптимизации в любой области.

Результаты, полученные в рамках настоящей выпускной квалификационной работы, были опубликованы в тезисах студенческой конференции «76-е Дни Науки Студентов НИТУ «МИСиС» [1].

ANNOTATION

The graduation work is presented on 75 pages, contains 13 figures, 1 table, a list of used sources from 27 titles, 3 appendices.

Keywords: THREE-COMPONENT OPTIMIZATION METHODS, CONJUGATE GRADIENT METHODS, MINIMUM POINT, LARGE DIMENSION, NUMERICAL EXPERIMENTS, PERFORMANCE PROFILE, LINEAR SEARCH, ARMIJO CONDITIONS, STRONG WOLF CONDITIONS, PYTHON, CUTEEST

The graduation work is devoted to the study of three-component methods of conjugate gradient for solving optimization problems.

The aim of this work is to develop a new three-component conjugate gradient method for solving optimization problems.

The result of the research is a new optimization method, as well as an assessment of the quality of its work, obtained by comparing the results of a series of numerical experiments of this method on a set of test problems with the performance characteristics of other algorithms.

The conducted numerical experiments proved that the developed method is extremely competitive among other methods of the conjugate gradient. This fact allows recommend the resulting method for use in solving optimization problems in any field.

The results obtained in the framework of this final qualification work were published in the abstracts of the student conference "76th Days of Science of NUST MISIS Students" [1].

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	8
1 Аналитический обзор литературы.....	10
1.1 Методы оптимизации	10
1.2 Длина шага метода.....	11
1.2.1 Разновидности условий останковки линейного поиска	12
1.2.2 Линейный поиск. Условия Армихо.....	15
1.2.3 Линейный поиск. Усиленные условия Вольфе	17
1.3 Разновидности методов сопряженного градиента.....	19
1.3.1. Метод Флетчера–Ривза.....	20
1.3.2 Метод Поляка–Рибьери	21
1.3.3 Метод Гестенса–Штифеля	22
1.3.4 Улучшенный метод Поляка–Рибьери	23
1.3.5 Гибридный метод сопряженного градиента.....	23
1.3.6 Трехкомпонентные методы.....	24
1.4 Тестирование метода	25
1.5 Анализ работы метода	25
1.6 Выводы.....	29
2 Содержательная постановка задачи	31
3 Математическая постановка задачи	32
4 Подготовка и анализ исходных данных.....	33
5 Описание разработанного алгоритма.....	35
6 Методы и инструменты информационных технологий	39
7 Разработанное программное обеспечение	41
8 Анализ полученных результатов	44
ЗАКЛЮЧЕНИЕ	49
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	50
ПРИЛОЖЕНИЕ А	52

ПРИЛОЖЕНИЕ Б.....	56
ПРИЛОЖЕНИЕ В	62

ВВЕДЕНИЕ

Понятия оптимальности и процесса оптимизации являются важнейшими аспектами в большом количестве различных наук от экономики и инженерного дела до менеджмента и бизнеса. В связи с этим область разработки методов, позволяющих находить оптимальное значение функции и аргумент, при котором это значение достигается, всегда интересовала математиков.

В последние годы методы оптимизации приобрели еще большую популярность в связи с использованием в машинном обучении. Интеллектуальные программы, возникшие на его основе – поисковые системы, рекомендательные платформы, программы распознавания речи и изображений – стали неотъемлемой частью современного общества. Благодаря этим разработкам повышается безопасность пластиковых карт банков за счет распознавания мошеннических операций и анализа финансовых транзакций в режиме реального времени; проводятся маркетинговые исследования, позволяющие более точно прогнозировать желания клиентов и спрос на новые продукты; настраиваются алгоритмы рекламы, прогноза пробок и фильтрации спама. Более того, использование современных методов оптимизации в задачах машинного обучения позволяет уменьшать погрешность моделей и увеличивать скорость сходимости при обучении [2].

Непрекращающийся рост размерности обрабатываемых моделями машинного обучения наборов данных вдохновляет исследователей на создание новых методов оптимизации, позволяющих все менее затратно с точки зрения вычислительной мощности и занимаемой памяти и в то же время все более быстро решать задачи поиска оптимума при наличии большого числа переменных.

На данный момент разработано большое количество различных методов оптимизации. В контексте решения задач оптимизации большой размерности популярностью пользуется квазиньютоновский метод в связи с его сверхлинейной скоростью сходимости, однако он требует большого количества ресурсов памяти ЭВМ. Вследствие этого для решения крупноразмерных задач была предложена модификация квазиньютоновского метода с ограниченной памятью, которая позволяет более экономно расходовать этот ресурс, но это привело к снижению скорости сходимости. Благодаря данному факту в некоторых случаях конкуренцию квазиньютоновскому методу оптимизации может составить метод сопряженного градиента за счет экономичного расходования памяти. У этого метода есть особое свойство: при генерации набора сопряженных векторов он вычисляет новый вектор направлений, используя только предыдущий вектор, в связи с чем необходимость в хранении матрицы всех предыдущих направлений отсутствует – последний вектор автоматически

сопряжен со остальными. Это свойство метода сопряженных градиентов позволяет ему быть экономичным и при этом иметь линейную скорость сходимости на некотором классе задач. Благодаря своей простой структуре и низким требованиям к памяти методы сопряженных градиентов широко используются для решения крупномасштабных задач оптимизации [3].

Существуют различные варианты методов сопряженного градиента, отличающиеся друг от друга различными способами выбора параметров. В связи с ростом размерности задач оптимизации усовершенствование имеющихся разновидностей и создание новых модификаций данного метода становится все более актуальным и необходимым.

В данной работе проводится поиск и анализ существующих методов сопряженного градиента, вследствие чего разрабатывается новый трехкомпонентный метод сопряженного градиента, для оценки качества работы которого впоследствии реализуется серия численных экспериментов на множестве тестовых задач оптимизации.

1 Аналитический обзор литературы

1.1 Методы оптимизации

Классическая задача оптимизации без ограничений заключается в минимизации целевой функции, зависящей от некоторого числа переменных, на значения которых не наложено никаких ограничений. Формула (1.1) представляет из себя ее математическую формулировку.

$$f(x) \rightarrow \min, x \in \mathbb{R}^n, \quad (1.1)$$

где $f(x): \mathbb{R}^n \rightarrow \mathbb{R}$ – гладкая функция и $x \in \mathbb{R}^n$ – вектор переменных с $n > 1$ компонентами. Решением такой задачи является $f^* = \inf f(x), x^* = \arg \min f(x), x \in \mathbb{R}^n$.

Численным методам оптимизации, способным вычислить приближение к точке минимума функции с некоторой точностью, для решения такой задачи достаточно иметь информацию о значении функции и ее градиента только в нескольких точках.

Классические методы оптимизации являются итерационными алгоритмами, шаг которых описывается формулой (1.2).

$$x_{k+1} = x_k + \alpha_k p_k \quad (1.2)$$

где x_k и x_{k+1} – приближения к точке минимума функции, полученные на k -ом и $(k + 1)$ -ом шагах соответственно, α_k – длина шага, а p_k – направление движения. Стоит отметить, что разновидности методов оптимизации отличаются друг от друга способами задания α_k и p_k .

В теории, спустя некоторое, часто бесконечно большое, количество итераций, метод сходится к точке минимума, в которой градиент функции равен нулю. На практике поиск прекращается, когда градиент равен нулю с заданной точностью, обычно равной 10^{-6} .

На данный момент разработано большое количество методов оптимизации различных типов. Безусловно, не существует метода, который решал бы все задачи одинаково успешно по сравнению с другими алгоритмами, в связи с чем изучение и постоянный поиск все новых усовершенствований существующих методов решения задач оптимизации пользуется большим интересом у исследователей.

В процессе анализа методов оптимизации будет обсуждаться их глобальная сходимость. Глобальная сходимость гарантирует, что при любом выборе начальной точки x_0 рассчитываемая методом последовательность приближений x_1, x_2, \dots сойдется к точке минимума функции.

1.2 Длина шага метода

В некоторых случаях бывает разумным выбрать в качестве длины шага α_k некоторое постоянное число, однако на практике это часто может привести к снижению скорости сходимости или даже к ее отсутствию. Поэтому большой популярностью в контексте выбора длины шага метода пользуются алгоритмы линейного поиска.

Говоря о длине шага метода α_k важно помнить о следующих ограничениях. С одной стороны, имеет смысл выбрать α_k такое, чтобы получить значительное уменьшение целевой функции, однако в то же время важно не тратить на этот выбор большое количество времени и вычислительной мощности. Наилучшим вариантом выбора альфа является *глобальный* минимизатор одномерной функции $\varphi(\alpha)$, заданной формулой (1.3).

$$\varphi(\alpha) = f(x_k + \alpha p_k), \alpha > 0 \quad (1.3)$$

Однако для вычисления этой величины может потребоваться большое количество времени. Более того, для вычисления даже *локального* минимизатора функции $\varphi(\alpha)$, как правило, требуется слишком много вычислений целевой функции f и ее градиента [4]. Этот факт хорошо видно из рисунка 1.

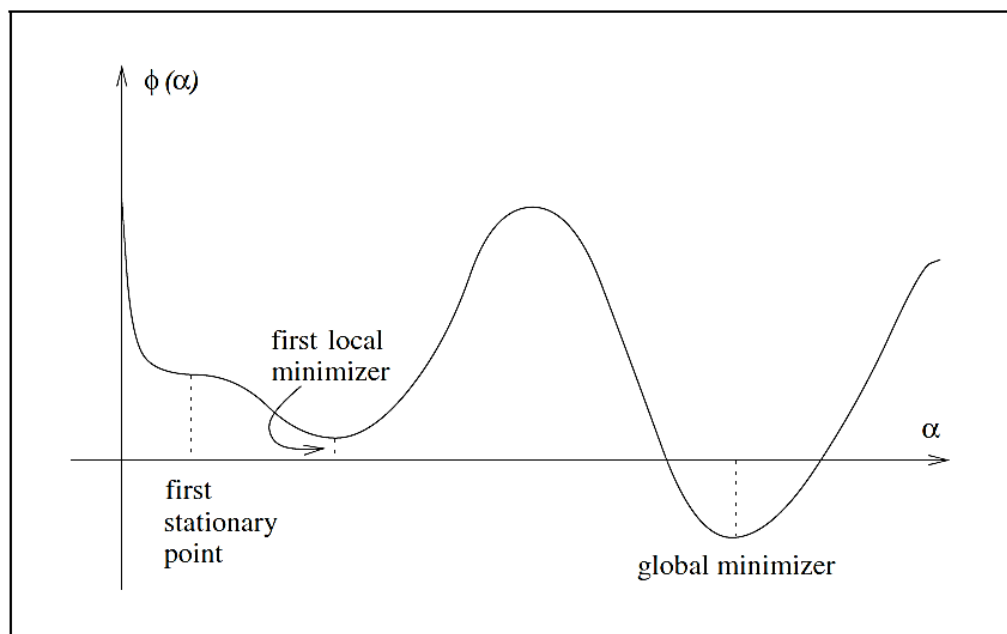


Рисунок 1 – Функция, для которой вычисление глобального минимума крайне затратно [4]

На практике наилучшим вариантом является использование линейного поиска для определения длины шага, который обеспечит достаточное уменьшение функции при минимальных затратах на выполнение этого поиска.

Классические алгоритмы линейного поиска генерируют последовательность пробных значений α тем или иным образом, пока не выполнится заданное условие остановки.

1.2.1 Разновидности условий остановки линейного поиска

В качестве условий остановки для линейного поиска часто выбирается условие Армихо, другими словами, условие достаточного уменьшения, а также классические или усиленные условия Вольфе [4].

Условие Армихо гласит, что вычисленная длина шага α должна прежде всего давать достаточное уменьшение целевой функции f , то есть

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k g_k^T p_k, \quad (1.4)$$

где g_k – градиент функции f в точке x_k , p_k – направление спуска, $c_1 \in (0; 1)$. Иными словами, уменьшение f должно быть пропорционально как длине шага α , так и производной по направлению $g_k^T p_k$. Смысл этого условия проиллюстрирован на рисунке 2.

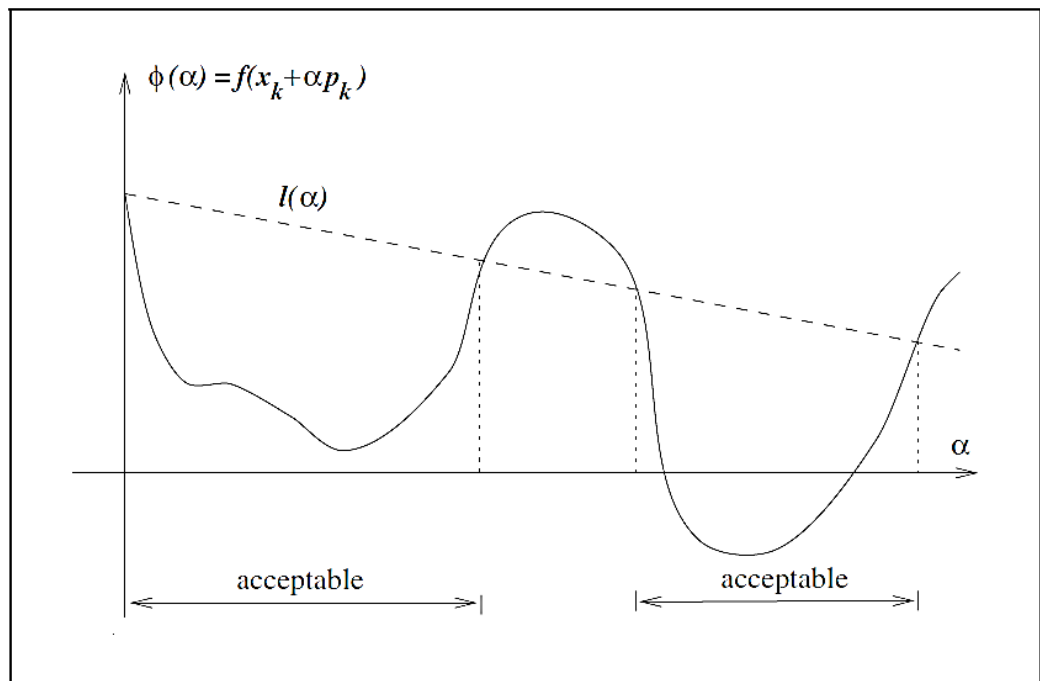


Рисунок 2 – Условие достаточного уменьшения [4]

На практике условия достаточного уменьшения недостаточно для обеспечения разумного прогресса алгоритма, поскольку оно выполняется для всех достаточно малых значений α [4]. Чтобы исключить слишком короткие шаги, часто вводится дополнительное требование, называемое условием кривизны (1.5).

$$g(x_k + \alpha_k p_k)^T p_k \geq c_2 g_k^T p_k, \quad (1.5)$$

для некоторой постоянной $c_2 \in (c_1; 1)$, где c_1 – постоянная из формулы (1.4). Очевидно, что левая часть неравенства (1.5) – это производная функции φ , заданной в (1.3), поэтому условие кривизны гарантирует, что наклон касательной к функции φ в точке α превышает больше чем в c_2 раза наклон касательной к φ в начальной точке. Это важно, поскольку если наклон касательной к φ сильно отрицательный, то при движении в выбранном направлении функция f значительно уменьшается. Напротив, если производная φ в точке α несильно отрицательна или даже положительна, это признак того, что большое уменьшение функции в заданном направлении не ожидается, поэтому стоит прекратить линейный поиск. Это свойство продемонстрировано на рисунке 3.

На практике значение c_1 выбирается довольно маленьким, чаще всего $c_1 = 10^{-4}$. При этом выбор c_2 отличается в зависимости от используемого метода оптимизации. В частности, для метода сопряженного градиента в качестве c_2 обычно принимают значение $c_2 = 0.1$ [4].

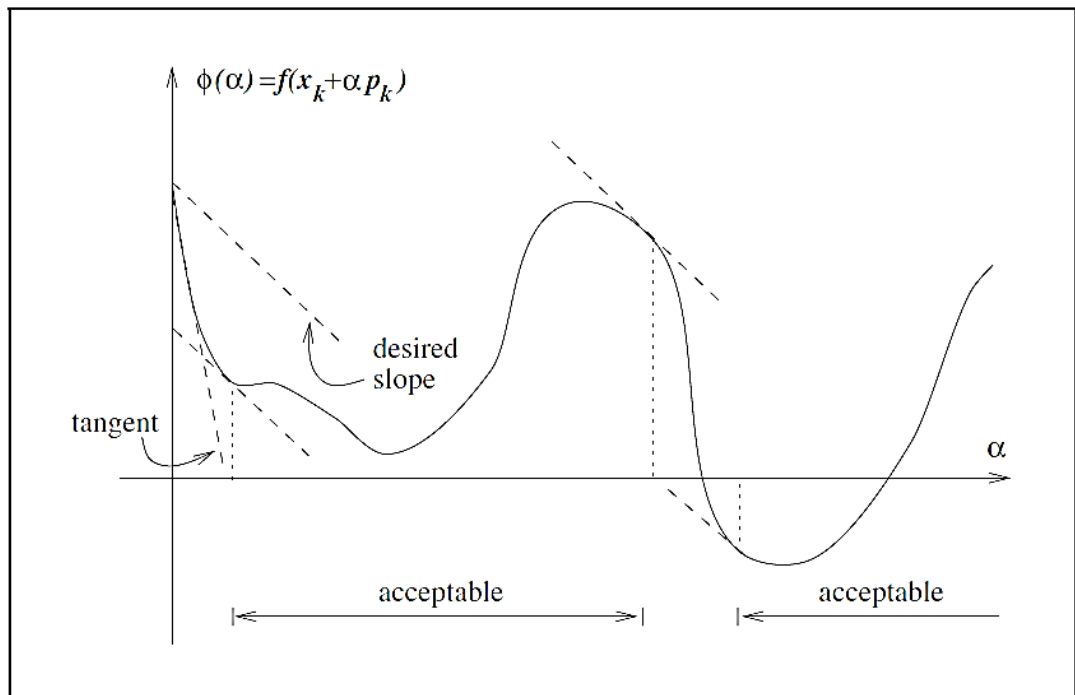


Рисунок 3 – Условие кривизны [4]

В совокупности условия достаточного уменьшения и кривизны известны как классические условия Вольфе:

$$\begin{aligned} f(x_k + \alpha_k p_k) &\leq f(x_k) + c_1 \alpha_k g_k^T p_k, \\ g(x_k + \alpha_k p_k)^T p_k &\geq c_2 g_k^T p_k, \end{aligned} \quad (1.6)$$

где $0 < c_1 < c_2 < 1$.

Важное свойство заключается в том, что длина шага α может удовлетворять условиям Вольфе, не будучи особенно близкой к минимизатору φ , как показано на рисунке 4. В связи с этим имеет смысл изменить условие кривизны таким образом, чтобы α принадлежала по крайней мере широкой окрестности локального минимизатора или стационарной точки φ . Полученные в таком случае условия называют *усиленными условиями Вольфе*:

$$\begin{aligned} f(x_k + \alpha_k p_k) &\leq f(x_k) + c_1 \alpha_k g_k^T p_k, \\ |g(x_k + \alpha_k p_k)^T p_k| &\leq c_2 |g_k^T p_k|, \end{aligned} \quad (1.7)$$

где $0 < c_1 < c_2 < 1$.

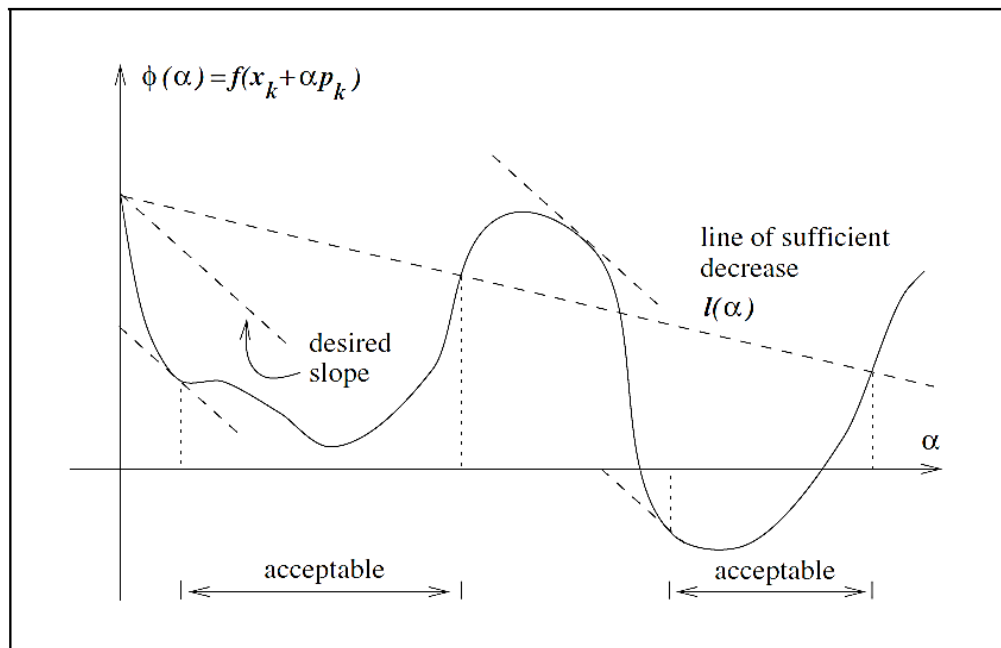


Рисунок 4 – Недостаток условий Вольфе [4]

Единственное отличие от классических условий Вольфе состоит в том, что производная функции φ теперь не может принимать большие по модулю положительные значения, исключая точки, далекие от стационарных точек φ .

Доказано, что длины шагов, удовлетворяющие условиям Вольфе, существуют для любой гладкой и ограниченной снизу функции f [4].

Разработаны различные методы поиска длины шага α , удовлетворяющей одному из описанных выше условий. Кроме того, считается, что p_k – направление уменьшения целевой функции, так что поиск может быть ограничен положительными значениями α . Все процедуры линейного поиска начинаются с оценки α_0 и после генерируют последовательность $\{\alpha_i\}$, которая либо сходится к длине шага, удовлетворяющей заданным условиям, либо определяет, что такой длины шага не существует.

В контексте методов сопряженного градиента, для формирования первоначального предположения о длине шага α_0 необходимо использовать текущую информацию о функции и алгоритме. Существует популярная стратегия, основанная на предположении о том, что изменение функции на итерации x_k будет таким же, как и на предыдущем шаге, то есть:

$$\alpha_0 g_k^T p_k = \alpha_{k-1} g_{k-1}^T p_{k-1}, \quad (1.8)$$

откуда

$$\alpha_0 = \alpha_{k-1} \frac{g_{k-1}^T p_{k-1}}{g_k^T p_k}. \quad (1.9)$$

В другом случае полезно использовать квадратичную интерполяцию, основанную на информации о значении функции и ее производной в предыдущей и текущей точках и определяющую α_0 как минимизатор полученной квадратичной функции, то есть:

$$\alpha_0 = \frac{2(f_k - f_{k-1})}{\varphi'(0)}. \quad (1.10)$$

1.2.2 Линейный поиск. Условия Армихо

Процедура линейного поиска, основанная на выполнении условия Армихо, использует интерполяцию известных значений функции и ее производной в точке. Цель алгоритма

состоит в вычислении значения длины шага α_k , которое при этом не будет «слишком маленьким». Для этого генерируется убывающая последовательность значений $\{\alpha_i\}$ таким образом, чтобы каждое следующее значение α_i было ненамного меньше, чем его предшественник α_{i-1} [4].

Пусть задано начальное значение длины шага α_0 . Если выполняется условие достаточного уменьшения (1.11), то поиск прекращается.

$$\varphi(\alpha_0) \leq \varphi(0) + c_1 \alpha_0 \varphi'(0) \quad (1.11)$$

В противном случае формируется *квадратичное* приближение $\varphi_q(\alpha)$ к φ путем интерполяции значений $\varphi(0)$, $\varphi'(0)$ и $\varphi(\alpha_0)$:

$$\varphi_q(\alpha) = \left(\frac{\varphi(\alpha_0) - \varphi(0) - \alpha_0 \varphi'(0)}{\alpha_0^2} \right) \alpha^2 + \varphi'(0) \alpha + \varphi(0). \quad (1.12)$$

Тогда новое пробное значение α_1 лежит в интервале $(0; \alpha_0)$ и равняется (1.13).

$$\alpha_1 = - \frac{\varphi'(0) \alpha_0^2}{2[\varphi(\alpha_0) - \varphi(0) - \varphi'(0) \alpha_0]} \quad (1.13)$$

После этого вновь проверяется выполнение условия достаточного уменьшения (1.11). Если оно выполняется, то поиск останавливается. В противном случае строится *кубическая* функция приближения $\varphi_c(\alpha)$ к φ путем интерполяции значений $\varphi(0)$, $\varphi'(0)$, $\varphi(\alpha_0)$ и $\varphi(\alpha_1)$. Эта кубическая функция всегда существует и уникальна [5]. Функция формируется следующим образом:

$$\varphi_c(\alpha) = a\alpha^3 + b\alpha^2 + \alpha\varphi'(0) + \varphi(0), \quad (1.14)$$

где

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\alpha_0^2 \alpha_1^2 (\alpha_1 - \alpha_0)} \begin{bmatrix} \alpha_0^2 & -\alpha_1^2 \\ \alpha_0^3 & \alpha_1^3 \end{bmatrix} \begin{bmatrix} \varphi(\alpha_1) - \varphi(0) - \varphi'(0)\alpha_1 \\ \varphi(\alpha_0) - \varphi(0) - \varphi'(0)\alpha_0 \end{bmatrix}. \quad (1.15)$$

Тогда следующее пробное значение α_2 лежит в интервале $(0; \alpha_1)$ и равняется:

$$\alpha_2 = \frac{-b + \sqrt{b^2 - 3a\varphi'(0)}}{3a}. \quad (1.16)$$

При необходимости кубическую интерполяцию повторяют, используя $\varphi(0), \varphi'(0), \varphi(\alpha_{i-1})$ и $\varphi(\alpha_i)$, где α_{i-1} и α_i – два последние вычисленные пробные значения длины шага, до тех пор, пока не будет найден α , удовлетворяющий условию Армихо.

Если на каком-либо шаге генерации пробных значений полученный α_i слишком близок к α_{i-1} , либо слишком сильно меньше его, то устанавливается

$$\alpha_{i-1} = \frac{\alpha_{i-1}}{2}. \quad (1.17)$$

Эта защитная процедура гарантирует, что достигается разумный прогресс на каждой итерации, и при этом конечный α не слишком мал.

1.2.3 Линейный поиск. Усиленные условия Вольфе

Классические и усиленные условия Вольфе относятся к числу наиболее широко применимых и полезных остановочных условий для алгоритма линейного поиска [4].

Алгоритм поиска длины шага α , удовлетворяющий усиленным условиям Вольфе, состоит из двух этапов. Первый этап начинается с пробной оценки α_1 , которое впоследствии увеличивается до тех пор, пока полученное значение не будет удовлетворять условиям остановки, либо пока не найдется интервал, который включает в себе желаемое значение α . Последний случай описывает второй этап, в котором используется так называемая функция *zoom*, последовательно уменьшающая размер интервала до тех пор, пока не будет определена приемлемая длина шага. Общее описание алгоритма представляет из себя следующее:

- устанавливается $\alpha_0 = 0$, $i = 1$, выбирается $\alpha_{\max} > 0$, где α_{\max} – максимально допустимая длина шага, и $\alpha_1 \in (0; \alpha_{\max})$;
- вычисляется значение функции φ в текущей точке α_i ;
- если $\varphi(\alpha_i) > \varphi(0) + c_1 \alpha_i \varphi'(0)$ или $[\varphi(\alpha_i) \geq \varphi(\alpha_{i-1}) \text{ и } i > 1]$, то искомое α^* вычисляется с помощью формулы (1.18) и поиск прекращается;

$$\alpha^* = zoom(\alpha_{i-1}; \alpha_i) \quad (1.18)$$

- в противном случае вычисляется значение производной функции φ в текущей точке α_i ;

- если $|\varphi'(\alpha_i)| \leq -c_2\varphi'(0)$, то искомое α^* вычисляется с помощью формулы (1.19) и поиск прекращается;

$$\alpha^* = \alpha_i \quad (1.19)$$

- если $\varphi'(\alpha_i) \geq 0$, то искомое α^* вычисляется с помощью формулы (1.20) и поиск прекращается;

$$\alpha^* = zoom(\alpha_i; \alpha_{i-1}) \quad (1.20)$$

- если никакое условие не выполнилось, то выбирается новое $\alpha_{i+1} \in (\alpha_i; \alpha_{\max})$, $i = i + 1$ и процесс повторяется вновь с шага 2.

На седьмом шаге для выбора $\alpha_{i+1} \in (\alpha_i; \alpha_{\max})$ можно использовать методы интерполяции, описанные выше, или же установить α_{i+1} в некоторое постоянное кратное α_i . При выборе стратегии важно, чтобы последовательные шаги увеличивались достаточно быстро, чтобы достичь верхнего предела α_{\max} за конечное число итераций.

Функция *zoom* в качестве аргументов принимает два значения – α_{lo} и α_{hi} , где:

- интервал $(\alpha_{lo}; \alpha_{hi})$ содержит значения длин шагов, удовлетворяющие усиленным условиям Вольфе;

- при α_{lo} значение функции φ является наименьшим среди всех длин шагов, сгенерированных до сих пор и удовлетворяющих достаточному условию уменьшения;

- α_{hi} выбирается так, что $\varphi'(\alpha_{lo})(\alpha_{hi} - \alpha_{lo}) < 0$.

Алгоритм функции *zoom* представляет из себя следующий процесс:

- с помощью интерполяции вычисляется пробная длина шага α_j из интервала $(\alpha_{lo}; \alpha_{hi})$, после чего считается $\varphi(\alpha_j)$;

- если $\varphi(\alpha_j) > \varphi(0) + c_1\alpha_j\varphi'(0)$ или $\varphi(\alpha_j) \geq \varphi(\alpha_{lo})$, то задается $\alpha_{hi} = \alpha_j$;

- иначе вычисляется $\varphi'(\alpha_j)$ и если $|\varphi'(\alpha_j)| \leq -c_2\varphi'(0)$, то искомое $\alpha^* = \alpha_j$ и поиск прекращается;

- если $\varphi'(\alpha_j)(\alpha_{hi} - \alpha_{lo}) \geq 0$, то $\alpha_{hi} = \alpha_{lo}$;

- если ни одно из двух условий не выполнилось, то $\alpha_{lo} = \alpha_j$;

- цикл повторяется с новыми значениями α_j , α_{lo} и α_{hi} .

Другими словами, если значение α_j удовлетворяет усиленным условиям Вольфе, то функция *zoom* верно выбрала длину шага и $\alpha^* = \alpha_j$. Иначе, если α_j удовлетворяет достаточному условию уменьшения и имеет меньшее значение функции, чем $\varphi(\alpha_{lo})$, то для поддержания второго условия для функции *zoom* устанавливаем $\alpha_{lo} = \alpha_j$. Если эта настройка приводит к нарушению третьего условия, то ситуация исправляется установкой $\alpha_{hi} = \alpha_{lo}$.

При использовании бесконечного цикла может возникнуть проблема, которая заключается в том, что по мере приближения алгоритма оптимизации к решению два последовательных значения функций $f(x_k)$ и $f(x_{k-1})$ могут быть неразличимы в арифметике конечной точности. В связи с этим в линейный поиск часто включают остановочный тест, если алгоритм не может получить более низкое значение функции после определенного числа итераций, обычно равного 10 [4]. Некоторые реализации алгоритма линейного поиска используют в качестве условия остановки факт близости относительного изменения x к машинной точности или к некоторому заданному порогу.

Практические реализации алгоритма линейного поиска помимо прочего используют свойства интерполирующих полиномов, чтобы сделать предположения о том, где должна лежать длина следующего шага [6].

На данный момент существует большое количество хороших реализаций алгоритмов линейного поиска. Например, в работе Денниса и Шнабеля [7], Лемарешаля [8], Флетчера [9], а также в работах [10, 11].

Стоит отметить, что для линейного поиска, опирающегося на классические условия Вольфе, потребуется такой же объем вычислений, как и для усиленных условий, однако последние имеют преимущество, поскольку, уменьшая c_2 , можно непосредственно контролировать качество поиска, заставляя принятое значение α лежать ближе к локальному минимизатору функции φ . Эта особенность важна для нелинейных методов сопряженного градиента, и поэтому процедура выбора шага, которая обеспечивает выполнение сильных условий Вольфе, имеет широкую применимость [4].

1.3 Разновидности методов сопряженного градиента

Методы сопряженного градиента названы таким образом из-за одного своего замечательного свойства, а именно способности крайне экономично с точки зрения занимаемой памяти генерировать наборы сопряженных векторов.

Набор ненулевых векторов $\{p_0, p_1, p_2, \dots, p_l\}$ сопряжен относительно симметричной положительно определенной матрицы A , если $\langle Ap_i, p_j \rangle = 0$ для всех $i \neq j$. Стоит отметить, что

любой набор сопряженных векторов является также линейно независимым. Важность свойства сопряженности заключается в том, что можно минимизировать функцию за n шагов последовательно минимизируя ее вдоль отдельных направлений в сопряженном множестве.

Ключевой особенностью, благодаря которой этот метод является эффективным для задач большой размерности по сравнению с другими разновидностями методов оптимизации, является тот факт, что метод вычисляет новый вектор направления p_k , используя информацию только лишь о векторе p_{k-1} . В связи с этим нет необходимости хранить информацию о векторах $p_0, p_1, p_2, \dots, p_{k-2}$ – вектор p_{k-1} уже с ними сопряжен. Следовательно, этот метод является экономичным по используемой памяти и производимым вычислениям благодаря отсутствию матричного хранения и операций.

В классическом варианте метода сопряженного градиента направление спуска вычисляется по формуле (1.21).

$$p_k = -g_k + \beta_k p_{k-1} \quad (1.21)$$

Метод сопряженного градиента впервые был предложен Гестенсом и Штифелем [12] как итерационный метод для решения *линейных* систем уравнений с положительно определенной матрицей коэффициентов. Эффективность линейного метода сопряженных градиентов определяется распределением собственных значений матрицы коэффициентов. Преобразовав особым образом матрицу коэффициентов, можно сделать это распределение более благоприятным в контексте использования линейного метода сопряженных градиентов.

Впервые *нелинейный* метод сопряженного градиента был введен Флетчером и Ривзом в 60-х годах XX-го века [13]. Это один из самых ранних известных методов решения задач нелинейной оптимизации большой размерности. Ключевые особенности этого алгоритма заключаются в том, что он не требует матричного хранения и на практике сходится быстрее, чем метод скорейшего спуска.

1.3.1. Метод Флетчера–Ривза

Флетчер и Ривз [13] первыми доказали существование способа расширить область применения метода сопряженных градиентов на нелинейные функции, предложив следующий алгоритм:

- вычисляется значение функции f и ее градиент g в начальной точке x_0 ;
- начальное направление спуска устанавливается равным градиенту в начальной точке, взятым с противоположным знаком;

- вычисляется длина шага α_k с помощью алгоритма линейного поиска;
- новое приближение к точке минимума считается согласно классической итерационной формуле (1.2);
- вычисляется значение градиента функции в полученной точке;
- если значение градиента функции в полученной точке равно нулю с заданной точностью, то точка минимума найдена и поиск прекращается;
- в противном случае вычисляется параметр β_k согласно формуле (1.22) и задается новое направление спуска (1.23), после чего алгоритм возвращается на этап вычисления длины шага α_k .

$$\beta_k^{FR} = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \quad (1.22)$$

$$p_k = -g_k + \beta_k^{FR} p_{k-1} \quad (1.23)$$

Данный алгоритм подходит для крупноразмерных задач нелинейной оптимизации, поскольку для выполнения итерации необходимо вычислить только значение целевой функции и ее градиента на текущем шаге. Для вычисления шага не требуется выполнение каких-либо матричных операций, а с точки зрения организации памяти необходимо хранить лишь несколько векторов.

Однако в работах Пауэлла [14] присутствует доказательство неэффективности метода Флетчера–Ривза, иллюстрирующее, как в некоторых случаях он может работать медленнее метода скорейшего спуска. В связи с этим появились другие модификации метода, призванные исправить описанный недостаток, в том числе метод Поляка–Рибьери, представленный ниже, который ведет себя иначе в подобных условиях, предотвращая наличие слишком коротких шагов.

1.3.2 Метод Поляка–Рибьери

Поляк и Рибьер предложили определять параметр β_k согласно формуле (1.24) [15]:

$$\beta_k^{PR} = \frac{g_k^T (g_k - g_{k-1})}{\|g_{k-1}\|^2} \quad (1.24)$$

В то время как для *линейных* функций оба описанных алгоритма вырождаются в идентичные друг другу, то применительно к общим *нелинейным* функциям поведение этих двух алгоритмов заметно отличается. Численные эксперименты показывают [4], что метод Поляка–Рибьери является более надежным и эффективным алгоритмом, чем метод Флетчера–Ривза. Однако существует пример, приведенный Пауэллом, доказывающий, что метод Поляка–Рибьери может не сходиться на невыпуклых задачах [16].

Следует отметить, что использование в качестве условий останковки линейного поиска усиленных условий Вольфе, не гарантирует, что p_k всегда является направлением спуска. Однако если задать $\beta_k^+ = \max\{\beta_k^{PR}; 0\}$, то это свойство сохраняется.

В статье у Гилберта и Носедаля [17] описывается реализация линейного поиска, которая всегда генерирует направление спуска, а также доказывается глобальная сходимость такого метода.

Не так давно в работах [18] и [19] соответственно были предложены и другие варианты метода сопряженного градиента. В формулах (1.25) и (1.26) приведены два варианта вычисления параметра β_k , которые обладают некоторыми полезными теоретическими и вычислительными свойствами:

$$\beta_k = \frac{\|g_k\|^2}{(g_k - g_{k-1})^T p_{k-1}}, \quad (1.25)$$

$$\beta_k = \left(y_{k-1} - 2p_{k-1} \frac{\|y_{k-1}\|^2}{y_{k-1}^T p_{k-1}} \right)^T \frac{g_k}{y_{k-1}^T p_{k-1}}, \quad (1.26)$$

где $y_{k-1} = g_k - g_{k-1}$. Полученные разновидности метода сопряженного градиента гарантируют, что p_k — это направление спуска, если длина шага α_k удовлетворяет условиям Вольфе.

1.3.3 Метод Гестенса–Штифеля

Гестенсом и Штифелем была предложена формула (1.27), определяющая новый вариант вычисления параметра β_k .

$$\beta_k^{HS} = \frac{g_k^T (g_k - g_{k-1})}{(g_k - g_{k-1})^T p_{k-1}} \quad (1.27)$$

Она задает алгоритм, который аналогичен алгоритму Поляка–Рибьери как с точки зрения его теоретических свойств сходимости, так и с точки зрения его практической эффективности [4].

1.3.4 Улучшенный метод Поляка–Рибьери

В 2009 году в работе [20] была предложена новая усовершенствованная формула вычисления параметра β_k , основанная на формуле Поляка–Рибьери:

$$\beta_k^{NPR} = \frac{\|g_k\|^2 - \frac{\|g_k\|}{\|g_{k-1}\|} |g_k^T g_{k-1}|}{\|g_{k-1}\|^2} \quad (1.28)$$

Нетрудно видеть, что предложенный метод сводится к методу Поляка–Рибьери, если целевая функция строго выпуклая, а линейный поиск точен. Кроме того, в работе [20] показано, что данный способ вычисления параметра β_k наследует все преимущества метода Флетчера–Ривза. Наконец, доказано, что данный метод удовлетворяет достаточному условию спуска и сходится глобально, если для выбора длины шага α используется линейный поиск, удовлетворяющий усиленным условиям Вольфе. Также в работе показана эффективность предложенного метода по сравнению с классическим методом Поляка–Рибьери.

1.3.5 Гибридный метод сопряженного градиента

Десятью годами позже в работе [21] предлагается новый гибридный метод сопряженного градиента для решения задач оптимизации без ограничений, формула параметра β_k для которого представлена в (1.29). Предлагаемый метод объединяет достоинства методов Флетчера–Ривза, Поляка–Рибьери и его улучшенной версии из исследования [20]. Помимо прочего, новый алгоритм удовлетворяет достаточному условию спуска независимо от результатов линейного поиска, а его глобальная сходимость гарантируется при использовании алгоритма линейного поиска, основанного на усиленных условиях Вольфе. Проведенные численные эксперименты показывают, что новый алгоритм является очень конкурентоспособным.

$$\beta_k^{EPF} = \begin{cases} \beta_k^{PR}, & \text{если } \|g_k\|^2 > |g_k^T g_{k-1}|, \\ (1 - \theta_k) \beta_k^{NPR} + \theta_k \beta_k^{FR}, & \text{иначе} \end{cases}, \quad (1.29)$$

$$\text{где } \beta_k^{PR} = \frac{g_k^T y_{k-1}}{\|g_{k-1}\|^2};$$

$$\begin{aligned}
g_k & \text{ -- градиент функции } f(x_k); \\
y_{k-1} & = g_k - g_{k-1}; \\
\|\cdot\| & \text{ -- евклидова норма;} \\
\theta_k & = \frac{\lambda - \Gamma \beta_k^{NPR}}{\Gamma \rho}; \\
\lambda & = y_{k-1}^T g_k; \\
\Gamma & = y_{k-1}^T p_{k-1} - \lambda \frac{p_{k-1}^T g_k}{\|g_k\|^2}; \\
\beta_k^{NPR} & = \frac{\|g_k\|^2 - \frac{\|g_k\|}{\|g_{k-1}\|} |g_k^T g_{k-1}|}{\|g_{k-1}\|^2}; \\
\rho & = \frac{\frac{\|g_k\|}{\|g_{k-1}\|} |g_k^T g_{k-1}|}{\|g_{k-1}\|^2}; \\
\beta_k^{FR} & = \frac{\|g_k\|^2}{\|g_{k-1}\|^2}.
\end{aligned}$$

Кроме того, если $\theta_k > 1$, то $\theta_k = 1$, а если $\theta_k < 0$, то $\theta_k = 0$.

1.3.6 Трехкомпонентные методы

Большое влияние на работу методов оптимизации оказывает способ выбора направления спуска. Базовые алгоритмы Поляка–Рибьера, Флетчера–Ривза и Гестенса–Штифеля являются двухкомпонентными методами сопряженного градиента, получившими свое название от количества слагаемых в формуле направления спуска (1.21). Однако в работе [22] предлагается новый *трехкомпонентный* метод сопряженного градиента. Эта формула (1.30) является наиболее распространенной формой на данный момент.

$$p_k = \begin{cases} -g_k, & \text{если } k = 0 \text{ или } g_k^T p_{k-1} = 0, \\ -g_k + \beta_k p_{k-1} - \beta_k \frac{g_k^T p_{k-1}}{g_k^T g_k} g_k & \text{иначе} \end{cases} \quad (1.30)$$

Основное достоинство предложенного способа выбора направления заключается в том, что условие достаточного уменьшения для такого метода выполняется всегда вне зависимости от выбора параметра β_k и результата линейного поиска длины шага α . В работе [22] также приводится доказательство глобальной сходимости предложенного метода и некоторые численные результаты исследования, демонстрирующие его конкурентоспособность.

1.4 Тестирование метода

Для оценки качества работы и сравнения между собой методов оптимизации необходимо провести серию численных экспериментов на разного рода задачах поиска минимума функции. Для стандартизации данного процесса и получения единообразных результатов исследователями были созданы различные тестовые функции. Важнейшим источником данных функций является библиотека CUTE, созданная и опубликованная Бонгарцем и другими [23]. Впоследствии исследователями были опубликованы расширенные версии данной библиотеки – CUTEr и CUTEst. Наличие таких библиотек позволяет сравнивать между собой различные методы оптимизации, поэтому использование CUTE является международной практикой в сфере разработки оптимизационного программного обеспечения уже не одно десятилетие.

Версия CUTEst библиотеки CUTE на данный момент является наиболее полной и содержит порядка 1500 различных задач оптимизации с ограничениями и без. Все функции разбиты на различные классы, что позволяет легко находить интересующие исследователя задачи. Признаками, по которым классифицируются задачи оптимизации, являются, например, тип целевой функции, наличие у нее ограничений, гладкость, количество существующих производных и так далее. Кроме того, для многих функций существует возможность вручную устанавливать различную размерность – в среднем от 2 до 20 000 переменных. Это позволяет оценивать изменения в результатах работы алгоритма в связи с ростом размерности задачи на примере одной функции. Наконец, каждая тестовая задача оптимизации из коллекции CUTEst сопровождается соответствующей ей начальной точкой x_0 , с которой начинается поиск точки минимума, что является очень важным фактором в контексте тестирования и сравнения различных методов.

1.5 Анализ работы метода

Для определения характеристик работы методов оптимизации необходимо провести серию численных экспериментов на различных тестовых задачах, после чего графически представить полученные результаты для наглядного сравнения различных алгоритмов.

Результаты тестирования методов оптимизации принято изображать с помощью профилей производительности. Такой способ иллюстрирования работы методов был предложен в работе [24]. Профиль производительности – это функция распределения характеристики, являющаяся инструментом для сопоставительного анализа показателей работы различных методов и сравнения оптимизационного программного обеспечения. В

работе [24] также показано, что такие графики сочетают в себе лучшие характеристики других инструментов оценки производительности.

Пусть имеется набор методов S и задач оптимизации P . В первую очередь вычисляется интересующая нас характеристика, например время работы метода, для каждой пары компонент из S и P , то есть, каждый метод решает каждую задачу оптимизации и в качестве результата получает параметр $t_{p,s}$ – вычислительное время, необходимое методу s для решения задачи оптимизации p . Затем для каждого значения $t_{p,s}$ рассчитывается коэффициент производительности по формуле (1.31), то есть, отношение полученного времени работы к наилучшему времени решения *данной задачи* среди рассматриваемых методов.

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s}: s \in S\}} \quad (1.31)$$

Для вычисления общей производительности метода определяется кумулятивная функция распределения коэффициента производительности, заданная формулой (1.32).

$$\rho_s(\tau) = \frac{1}{n_p} \text{size}\{p \in P: r_{p,s} \leq \tau\}. \quad (1.32)$$

Здесь $\rho_s(\tau)$ – вероятность, что коэффициент производительности метода s для задачи p находится в пределах τ от наилучшего коэффициента, то есть, отношение количества задач, для которых полученный $r_{p,s}$ меньше, чем τ , к общему количеству задач. Если набор задач P достаточно велик и является репрезентативным, то качество работы метода тем лучше, чем больше его значение $\rho_s(\tau)$. Аналогично рассчитываются коэффициенты производительности и функции их распределения для любых других характеристик работы метода.

При исследовании работы различных методов оптимизации принято сравнивать производительность алгоритмов по количеству вычислений значений функции и ее градиента в точке, числу выполненных итераций, а также по времени решения задачи оптимизации.

Например, в работе [25] в заключительной части исследования представлены графики, изображенные на рисунках 5 и 6. Они основаны на времени решения задачи, количестве итераций и подсчетах значения функции и ее градиента в точке.

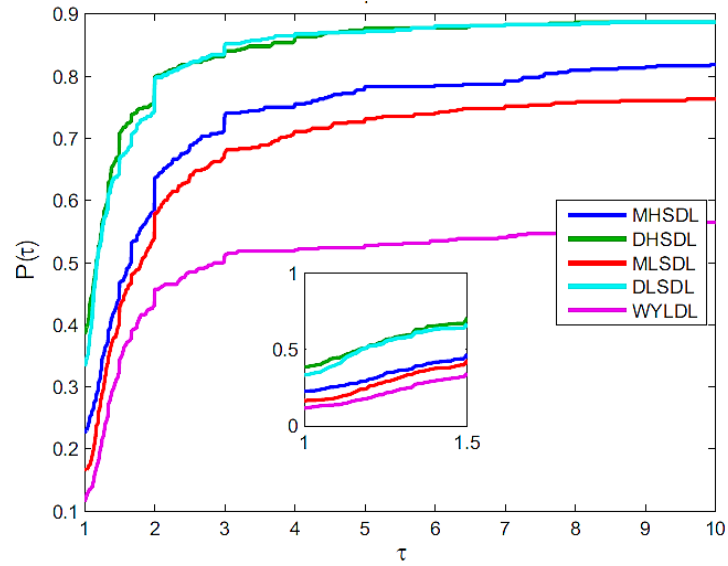


Рисунок 5 – Профили производительности по времени работы методов

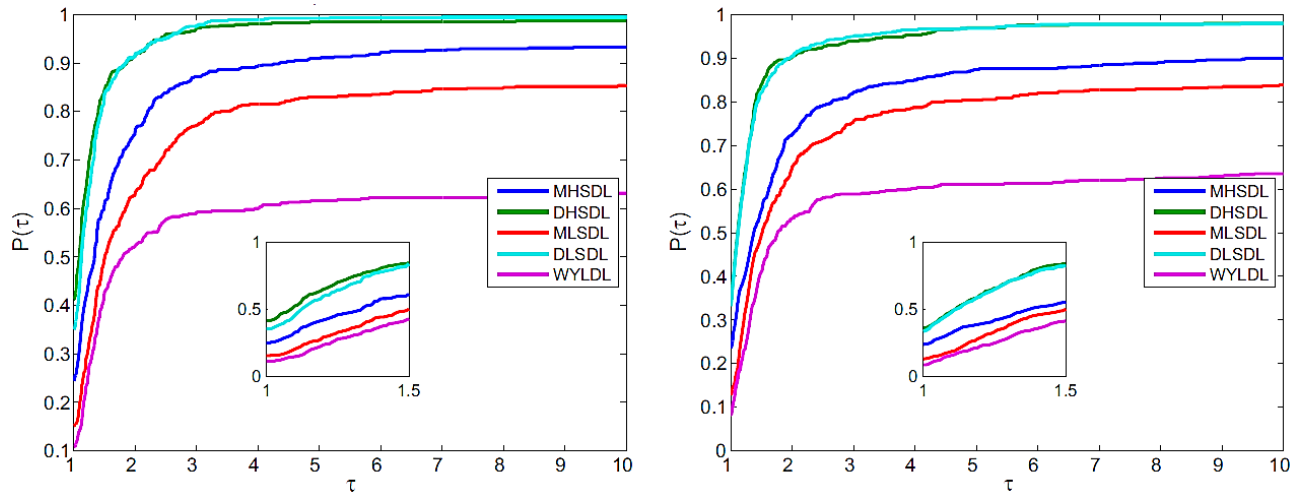


Рисунок 6 – Профили производительности по числу итераций и подсчетов значений функции и ее градиента в точке

Профиль производительности $\rho_s: \mathbb{R} \mapsto [0; 1]$ для метода оптимизации представляет собой неубывающую кусочно–постоянную функцию, непрерывную справа в каждой точке останова. Значение $\rho_s(1)$ – это вероятность того, что данный метод окажется *лучшим* среди методов с точки зрения рассматриваемой характеристики. Таким образом, если необходимо выбрать лучший метод с точки зрения количества раз, когда значение изучаемой метрики у него было лучше, чем у остальных алгоритмов, то необходимо сравнить значения ρ_s методов в точке $\tau = 1$ и выбрать тот, чье значение окажется больше.

Предполагается, что $r_{p,s} \in [1; r_{\max}]$ и $r_{p,s} = r_{\max}$ только в том случае, если метод s не смог решить задачу p . Следовательно, $\rho_s(r_{\max}) = 1$ и $p_s^* = \lim_{\tau \rightarrow r_{\max}} \rho_s(\tau)$ – это вероятность того,

что метод *справится* с решением задачи. Таким образом, если необходимо выбрать лучший метод с точки зрения высокой вероятности успеха решения, то нам необходимо сравнить значения ρ_s для всех методов при достаточно больших значениях τ , и выбрать тот алгоритм, у которого данное значение окажется больше.

Кроме того, важными свойствами графиков профилей производительности является то, что они не только не чувствительны к *сильным* изменениям результатов по *небольшому* числу задач, но и то, что они в значительной степени не подвержены влиянию *небольших* изменений результатов по *многим* задачам оптимизации.

На основании вышесказанного очевидно, что для полноценного анализа работы методов оптимизации необходимо выводить как минимум два графика профилей производительности – на малых значениях τ , близких к единице, и на больших значениях τ , близких к r_{\max} . Однако, иногда даже большие значения τ не позволяют в полной мере оценить производительность некоторых методов. В таком случае в работе [24] предлагается использовать логарифмическую шкалу для профилей производительности. Тогда кумулятивная функция распределения примет вид (1.33).

$$\rho_s(\tau) = \frac{1}{n_p} \text{size}\{p \in P: \log_2 r_{p,s} < \tau\}. \quad (1.33)$$

Такая формула позволяет оценить всю активность метода для $\tau < r_{\max}$. Поскольку нас интересует значения $\rho_s(1)$, то в качестве основания логарифма выбирается двойка. Такой график профиля производительности объединяет в себе все достоинства и особенности предыдущих двух графиков и, таким образом, позволяет составить полное представление о качестве работы того или иного метода оптимизации. Единственным недостатком такого типа графика является отсутствие интуитивной интерпретации результатов, поскольку вместо обычной шкалы вероятности в данном случае используется логарифмическая.

Помимо профилей производительности в рамках тестирования методов оптимизации принято предоставлять технический отчет по проведенным численным экспериментам. Данный отчет представляет из себя сводную таблицу о полученных значениях интересующих характеристик для каждого метода и задачи. Пример такого технического отчета изображен на рисунке 7. В нем приводятся метрики работы двух методов – VPRP и NPRP – на некотором множестве задач оптимизации, отраженных в первом столбце слева. Для каждой задачи также указана ее размерность во втором столбце. Для каждого метода и задачи, которую он решает, выводится 4 характеристики – количество выполненных итераций, число подсчетов значений функции и ее градиента в точке, а также затраченное на решение время.

Problem	n	VPRP				NPRP			
		iter	fn	gn	time	iter	fn	gn	time
DIXMAANA	6000	9	24	24	0.15	6	19	19	0.14
DIXMAANB	6000	6	21	21	0.16	6	21	21	0.16
DIXMAANC	6000	7	23	23	0.15	7	24	24	0.16
DIXMAAND	6000	8	25	25	0.15	8	25	25	0.14
FLETGBV2	5000	4142	8285	8285	27.81	436	885	885	3.02
FLETGBV2	10,000	7205	14,411	14,411	97.46	179	365	365	2.74
MOREBV	10,000	100	201	201	1.02	53	107	107	0.69
SPARSQR	10,000	38	131	131	1.12	75	290	290	2.25
LIARWHD	5000	29	83	83	0.26	23	78	78	0.25
LIARWHD	10,000	34	107	107	1.08	23	68	68	0.74
BRYBND	10,000	46	109	109	1.11	28	67	67	0.93
HILBERTA	200	14	38	38	0.42	14	38	38	0.45
HILBERTB	200	5	13	13	0.24	5	13	13	0.24
PENALTY1	5000	22	98	98	0.18	22	99	99	0.18
PENALTY1	10,000	19	89	89	0.36	19	89	89	0.36
QUARTC	5000	33	113	113	0.13	33	119	119	0.14
QUARTC	10,000	40	133	133	0.30	41	154	154	0.38
NONDIA	5000	10	29	29	0.16	10	38	38	0.18
NONDIA	10,000	7	24	24	1.47	9	36	36	1.54
SROSENB	10,000	11	30	30	0.18	10	28	28	0.18
BROWNAL	400	4	23	23	0.24	4	23	23	0.23
ARGLINA	300	1	5	5	0.29	1	5	5	0.29
COSINE	10,000	9	30	30	0.28	9	30	30	0.28
FMINSRF2	15,625	497	999	999	11.36	472	949	949	9.92
FMINSRF2	10,000	434	874	874	6.23	389	784	784	5.00
Average		509.2	1036.7	1036.7	6.09	75.3	174.2	174.2	1.22

Рисунок 7 – Технический отчет

Такого рода технические отчеты позволяют численно оценить разницу в работе различных методов и выделить интересные особенности, которые не позволяет сделать график, например классы задач, на которых определенный алгоритм справляется значительно хуже или лучше остальных.

1.6 Выводы

В данной главе были рассмотрены существующие разновидности метода сопряженного градиента, их отличия и преимущества относительно друг друга. Методы отличаются друг от друга главным образом способом выбора направления спуска, а также параметром β_k .

Кроме того, было проведено исследование на тему выбора длины шага метода. Наиболее популярной методикой для этого на данный момент является алгоритм линейного поиска, вычисляющий параметр α , который удовлетворяет усиленным условиям Вольфе. Это позволяет методу достигнуть хорошей скорости сходимости, будучи при этом достаточно экономным с точки зрения производимых вычислений. Метод скорейшего спуска для вычисления α , очевидно, давал бы методу еще большую скорость сходимости, но при этом он гораздо менее экономичный, а выбор $\alpha_k = const$, безусловно, является самым экономичным, однако при этом обеспечивает методу наихудшую сходимость.

Помимо прочего, были изучены варианты выбора тестовых задач оптимизации для проведения серии численных экспериментов. Выявлено, что в своих работах различные

исследователи методов оптимизации во всем мире используют коллекцию функций CUTEst, которая является наиболее полной и удобной библиотекой для тестирования оптимизационного программного обеспечения.

Наконец, были найдены графические способы отображения результатов проведенных экспериментов, широко используемые при исследовании методов оптимизации в различных работах. Характеристики работы алгоритма наиболее часто представляются в виде трех типов графиков профилей производительности, а также технического отчета в формате сводной таблицы, содержащей информацию о метриках работы алгоритма.

2 Содержательная постановка задачи

Цель работы – разработать новый трехкомпонентный метод сопряженного градиента, вычисляющий приближение к точке минимума целевой функции.

Для достижения поставленной цели необходимо:

- провести анализ существующих разновидностей методов сопряженного градиента;
- разработать новый трехкомпонентный метод сопряженного градиента;
- выбрать существующие методы оптимизации для проведения сравнительного анализа с разработанным алгоритмом;
- реализовать с помощью программных средств разработки данные алгоритмы;
- сформировать набор тестовых задач оптимизации из библиотеки CUTEst;
- провести серию численных экспериментов на множестве тестовых функций;
- составить технический отчет по полученным результатам;
- изобразить профили производительности рассматриваемых методов оптимизации;
- провести сравнительный анализ разработанного метода с другими алгоритмами оптимизации;
- сделать выводы о применимости предлагаемого метода.

Входные данные:

- набор тестовых функций f с размерностью n и начальной точкой x_0 .

Выходные данные:

- x^* – вычисленное приближение к точке минимума функции;
- f и $\|x\|$ – значение функции и нормы аргумента в точке x^* ;
- $fcalls$ и $gcalls$ – количество вычислений значения функции и ее градиента в точке;
- $iters$ – количество выполненных итераций;
- $time$ – время решения задачи оптимизации.

Выполнить работу в соответствии с ГОСТ 2.111–68, оформить текст согласно ГОСТ 7.32–2001, страницы текста – ГОСТ 9327–60, а список источников – ГОСТ 7.1–2003.

3 Математическая постановка задачи

Разработать трехкомпонентный метод сопряженного градиента для поиска точки минимума функции без ограничений (3.1).

$$f(x) \rightarrow \min, x \in \mathbb{R}^n, \quad (3.1)$$

где f – выпуклая функция общего вида в пространстве \mathbb{R}^n .

Шаг метода задать с помощью итерационной формулы (3.2):

$$x_{k+1} = x_k + \alpha_k p_k, \quad (3.2)$$

где $x_k \in \mathbb{R}^n$ – это вектор k -го приближения решения $x^* \in \mathbb{R}^n$, α_k – положительный скаляр, задающий длину шага метода, $p_k \in \mathbb{R}^n$ – вектор направления поиска.

Вычислять с помощью алгоритма линейного поиска длину шага α_k , удовлетворяющую усиленным условиям Вольфе, сформулированным в (3.3).

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k g_k^T p_k, \quad (3.3)$$

$$|g(x_k + \alpha_k p_k)^T p_k| \leq c_2 |g_k^T p_k|,$$

где $0 < c_1 < c_2 < 1$.

В случае, если алгоритм линейного поиска при заданных параметрах не сходится, использовать в качестве критерия остановки условие Армихо, сформулированное в (3.4).

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k g_k^T p_k, \quad (3.4)$$

Для задания направления спуска определить новую модификацию трехкомпонентной формулы, обеспечивающую разрабатываемому методу оптимизации высокую скорость сходимости и экономичность с точки зрения количества вычислений значений функции и ее градиента в точке в сравнении с другими методами.

4 Подготовка и анализ исходных данных

Для тестирования и сравнения новых методов оптимизации с существующими алгоритмами исследователями были разработаны различные тестовые функции. Важнейшим источником данных функций является библиотека *CUTE*, созданная и опубликованная Бонгарцем и другими [23]. Впоследствии исследователями были опубликованы расширенные версии данной библиотеки – *CUTEr* и *CUTEst*. В данной работе используется последняя и наиболее полная версия из всех.

На основании классификации тестовых функций данной библиотеки и общепринятой мировой практики для тестирования разрабатываемого метода был сформирован набор гладких задач оптимизации без ограничений с размерностью от 2 до 10 000 и явно заданной целевой функцией. Список исходных задач с описанием и наивысшей размерностью приведен в приложении А.

Помимо задач из коллекции *CUTEst* для тестирования и сравнения разрабатываемого метода с другими предлагается использовать собственные тестовые задачи, сгенерированные в большом количестве. В качестве задачи для данных тестов выбрана задача минимизации квадратичной функции с L_2 -регуляризатором, приведенная в (4.1).

$$f(x) = \|Ax - b\|^2 + \lambda \|x\|^2 \rightarrow \min, x \in \mathbb{R}^M \quad (4.1)$$

где A – прямоугольная матрица размера $N \times M$, $b \in \mathbb{R}^N$ – заданный вектор, λ – заданное число, а $\|Ax - b\|$ и $\|x\|$ – евклидовы нормы. Размерности таких задач в проведенном исследовании составляют от 10 до 1000. Для тестирования генерируется 1000 задач каждой размерности, то есть всего 10 000 задач.

При составлении тестовой задачи такого вида с заранее известным единственным точным решением x_* предлагается воспользоваться критерием оптимальности для x_* – условием Ферма, сформулированным в выражении (4.2).

$$2A^T(Ax_* - b) + 2\lambda x_* = 0 \quad (4.2)$$

При $\lambda > 0$ решение x_* удовлетворяет условию (4.3).

$$x_* = A^T y_*, y_* \in \mathbb{R}^N \quad (4.3)$$

После подстановки (4.3) в (4.2) становится очевидна следующая схема составления тестовой задачи:

- генерируется матрица A размера $N \times M$, число $\lambda > 0$ и вектор $y_* \in \mathbb{R}^N$;
- вычисляется вектор $b = (AA^T + \lambda I)y_*$.

Задача оптимизации сформирована. Ее точное решение вычисляется по формуле (4.3).

Стоит отметить, что такую задачу решает одна из линейных моделей машинного обучения – регрессия Риджа, поэтому оценка качества работы разработанного метода на данном классе задач оптимизации является для данной работы еще более актуальной.

Таким образом, в данной работе оценка приводимых методов оптимизации производится двумя способами – на тестовых задачах из библиотеки CUTEst и на собственных квадратичных задачах машинного обучения. Такой подход позволяет с разных сторон оценить качество разработанного метода и сделать более объективный сравнительный анализ.

5 Описание разработанного алгоритма

В рамках выпускной квалификационной работы был разработан новый трехкомпонентный метод сопряженного градиента. Блок–схема данного алгоритма приведена на рисунке 8. Далее приведено подробное описание каждого этапа работы метода.

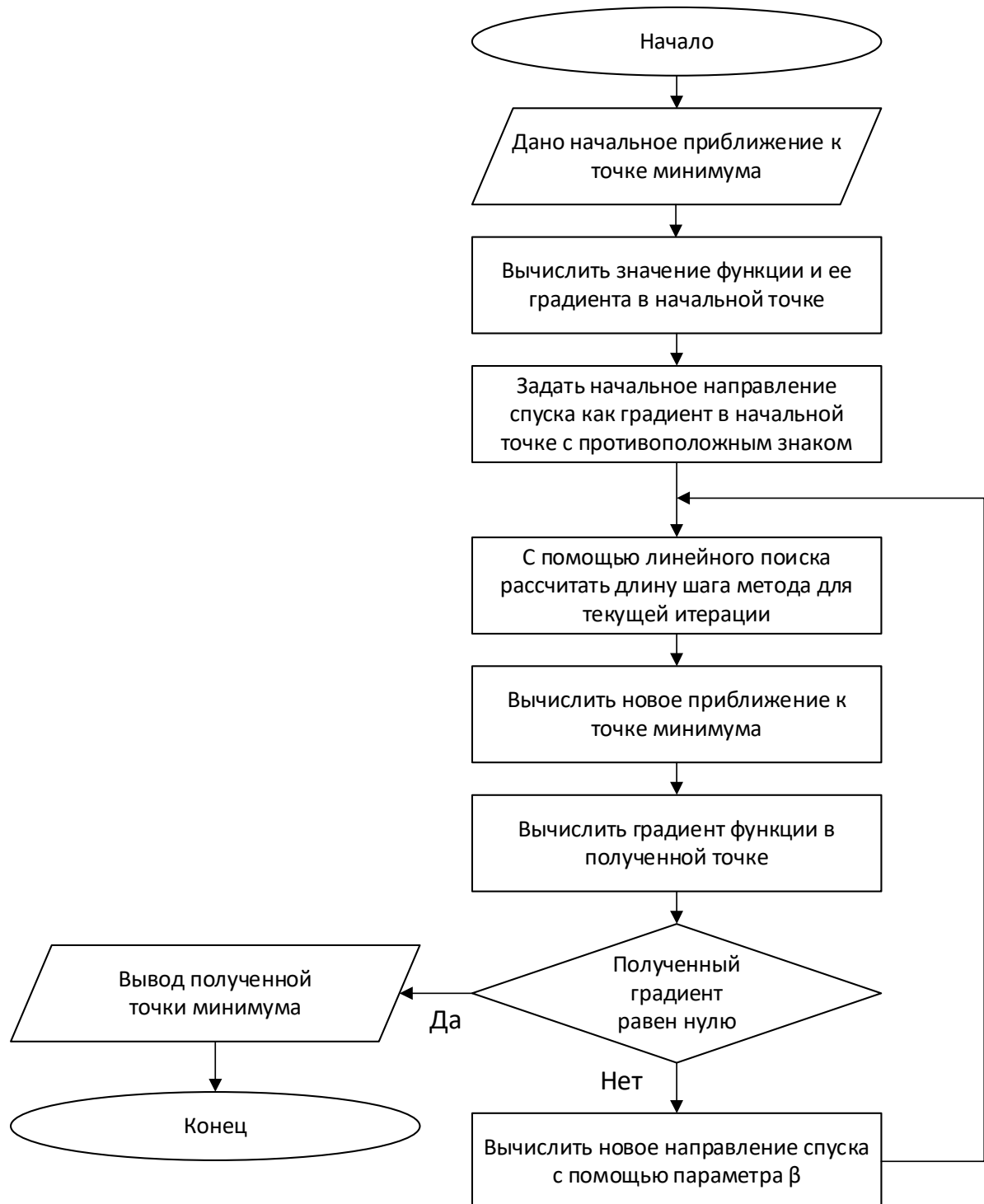


Рисунок 8 – Блок–схема разработанного метода оптимизации

Для целевой функции f дано начальное приближение x_0 к точке минимума. Прежде всего вычисляется значение функции f в точке x_0 : $f_0 = f(x_0)$, а также градиент функции f в точке x_0 : $g_0 = g(x_0)$. На нулевой итерации ($k = 0$) начальное направление спуска p_0 задается градиентом в начальной точке x_0 , взятым с противоположным знаком, то есть $p_0 = -g_0$.

После этого с помощью алгоритма линейного поиска, основанного на усиленных условиях Вольфе, вычисляется длина шага α_k на основании текущих значений x_k и p_k . Если алгоритм не сошелся, то α_k высчитывается согласно условиям Армихо. Если α_k не удалось получить и в данном случае, метод прекращает работу, а задача оптимизации считается нерешенной.

Важной особенностью является тот факт, что при тестировании на квадратичных задачах с L_2 -регуляризатором длина шага α_k вычисляется по формуле (5.1). Известно, что такой выбор α_k является наилучшим из всех возможных, поэтому в использовании алгоритма поиска длины шага нет необходимости.

$$\alpha_k = -\frac{\langle Ax_k - b | Ap_k \rangle + \lambda \langle x_k | p_k \rangle}{\|Ap_k\|^2 + \lambda \|p_k\|^2} \quad (5.1)$$

Далее новое приближение к точке минимума функции рассчитывается по классической итерационной формуле (5.2).

$$x_{k+1} = x_k + \alpha_k p_k \quad (5.2)$$

Затем вычисляется градиент функции в полученной точке x_{k+1} , то есть $g_{k+1} = g(x_{k+1})$. Если полученный $g_{k+1} = 0$, то поиск прекращается, поскольку точка минимума найдена. В противном случае высчитывается параметр β_k с помощью гибридной формулы (5.3), предложенной в работе [26].

$$\beta_k = \begin{cases} \beta_k^{PR}, & \text{если } \|g_k\|^2 > |g_k^T g_{k-1}| \\ (1 - \theta_k) \beta_k^{NPR} + \theta_k \beta_k^{FR} \end{cases} \quad (5.3)$$

$$\text{где } \beta_k^{PR} = \frac{g_k^T y_{k-1}}{\|g_{k-1}\|^2};$$

g_k – градиент функции $f(x_k)$;

$$y_{k-1} = g_k - g_{k-1};$$

$\|\cdot\|$ – евклидова норма;

$$\begin{aligned}
\theta_k &= \frac{\lambda - \Gamma \beta_k^{NPR}}{\Gamma \rho}; \\
\lambda &= y_{k-1}^T g_k; \\
\Gamma &= y_{k-1}^T p_{k-1} - \lambda \frac{p_{k-1}^T g_k}{\|g_k\|^2}; \\
\beta_k^{NPR} &= \frac{\|g_k\|^2 - \frac{\|g_k\|}{\|g_{k-1}\|} |g_k^T g_{k-1}|}{\|g_{k-1}\|^2}; \\
\rho &= \frac{\frac{\|g_k\|}{\|g_{k-1}\|} |g_k^T g_{k-1}|}{\|g_{k-1}\|^2}; \\
\beta_k^{FR} &= \frac{\|g_k\|^2}{\|g_{k-1}\|^2}.
\end{aligned}$$

Кроме того, если $\theta_k > 1$, то $\theta_k = 1$, а если $\theta_k < 0$, то $\theta_k = 0$.

Направление спуска предлагается задавать новой трехкомпонентной формулой (5.4), разработанной в рамках данного исследования. Данная формула является модификацией классической трехкомпонентной формулы (5.5) с добавлением к первому и третьему слагаемому множителя ω (5.6). Впервые формула коэффициента ω была получена в работе [27]. Тогда исследователи предложили использовать данный множитель в двухкомпонентной формуле, и такая модификация метода на серии численных экспериментов продемонстрировала более высокую производительность по сравнению с методом, направление спуска которого задано классической двухкомпонентной формулой. В данной работе предлагается расширить данные идеи на класс трехкомпонентных методов.

$$p_{k+1} = -\omega_k g_k + \beta_k p_k - \omega_k \beta_k \frac{g_k^T p_k}{\|g_k\|^2} g_k \quad (5.4)$$

$$p_{k+1} = -g_k + \beta_k p_k - \beta_k \frac{g_k^T p_k}{\|g_k\|^2} g_k \quad (5.5)$$

$$\omega_k = \frac{p_k^T y_k}{\|g_{k-1}\|^2} \quad (5.6)$$

Условием остановки поиска точки минимума функции является равенство градиента функции в полученной точке нулю с заданной точностью, обычно 10^{-6} . Кроме того, для предотвращения появления бесконечного цикла в случае, если соответствующий метод не

сходится к точке минимума тестовой функции, наложено дополнительное условие на количество итераций. В случае, когда метод оптимизации не сходится к решению задачи за 100 000 итераций, считается, что он работает плохо и для поиска минимума в подобного рода задачах не рекомендуется. Помимо описанных выше условий работа метода может прекратиться в случае отсутствия успеха при выборе длины шага α_k на какой-либо итерации, другими словами, в случае отсутствия сходимости у алгоритма линейного поиска, основанного на условиях Вольфе и Армихо. Тогда также полагается, что данный метод не справился с решением задачи оптимизации.

6 Методы и инструменты информационных технологий

Для работы с функциями из коллекции задач оптимизации *CUTEst* используется *PyCUTEst* – интерфейс *Python* для *CUTEst*, пакета *Fortran* для тестирования оптимизационного программного обеспечения. Он основан на интерфейсе, первоначально разработанном для *CUTEr* профессором Арпадом Бюрменом. Этот интерфейс позволяет получать данные из файла функции библиотеки формата *.sif* по ее названию, а также задавать различные параметры для вызова, в том числе размерность задачи. Это во многом ускоряет и упрощает процесс проведения тестирования работы исследуемых методов, особенно учитывая тот факт, что множество исходных задач является достаточно большим.

Для работы с интерфейсом *PyCUTEst* программное обеспечение реализуется на высокоуровневом языке программирования *Python*. Взаимодействие с данным интерфейсом возможно только на платформе *UNIX*-подобных операционных систем, поэтому в данном исследовании разработка программного кода осуществляется в системе *Linux*. Для упрощения процесса разработки и отладки используется современная среда разработки *PyCharm*, оснащенная широким функционалом и удобным пользовательским интерфейсом.

Помимо описанного выше интерфейса *PyCUTEst* при разработке программного обеспечения используются такие библиотеки, как *NumPy*, *Time*, *Random*, *PrettyTable*, *SciPy* и *Matplotlib*, подробное описание которых приведено ниже.

- *NumPy* – фундаментальный пакет для научных вычислений на *Python*. Это библиотека *Python*, которая предоставляет такой объект как многомерный массив и различные производные объекты, а также набор процедур для быстрых операций с массивами, включая математические, логические, манипуляции с формой, сортировку, выбор, различные преобразования и многое другое. В разработанном программном обеспечении используются реализации *NumPy* для вычисления скалярного произведения, подсчета евклидовой нормы и абсолютного значения числа;

- *Time* – модуль *Python*, предоставляющий различные функции для работы со временем. В контексте данной работы этот модуль используется для подсчета времени, потребовавшегося методу для решения задачи оптимизации;

- *Random* – модуль *Python*, реализующий генераторы псевдослучайных чисел для различных распределений. Используется в данной работе для генерации тестовых квадратичных задач с L_2 -регуляризатором;

- *PrettyTable* – библиотека *Python*, используемая для удобного отображения табличных данных в наглядном формате таблицы ASCII. В работе применяется для вывода технического отчета по серии численных экспериментов;

- *SciPy* – библиотека с открытым исходным кодом на *Python* для математики, естественных наук и инженерии. В контексте данного исследования используется реализация алгоритма линейного поиска, вычисляющего длину шага α_k , удовлетворяющее условиям Армихо;

- *Matplotlib* – комплексная библиотека *Python* для создания статических, анимированных и интерактивных визуализаций. В разработанном программном обеспечении используется для построения графиков профилей производительности методов оптимизации.

Разработка, отладка и численные эксперименты проводятся на персональном компьютере с процессором и оперативной памятью 8.00Гб. Данный компьютер не обладает никакими специфическими характеристиками, поскольку вычислительная мощность ЭВМ не влияет на производительность методов с точки зрения количества итераций и расчетов функций и их градиентов, а именно они являются основными характеристиками работы методов оптимизации.

7 Разработанное программное обеспечение

Для достижения поставленных в работе целей реализовано программное обеспечение на высокоуровневом языке программирования *Python* в среде разработки *PyCharm 2020.1*. Проект состоит из двух отдельных модулей, один из которых посвящен тестированию на задачах *CUTEst*, а другой – на квадратичных задачах с L_2 –регуляризатором. Схема разработанного программного обеспечения приведена на рисунке 9.

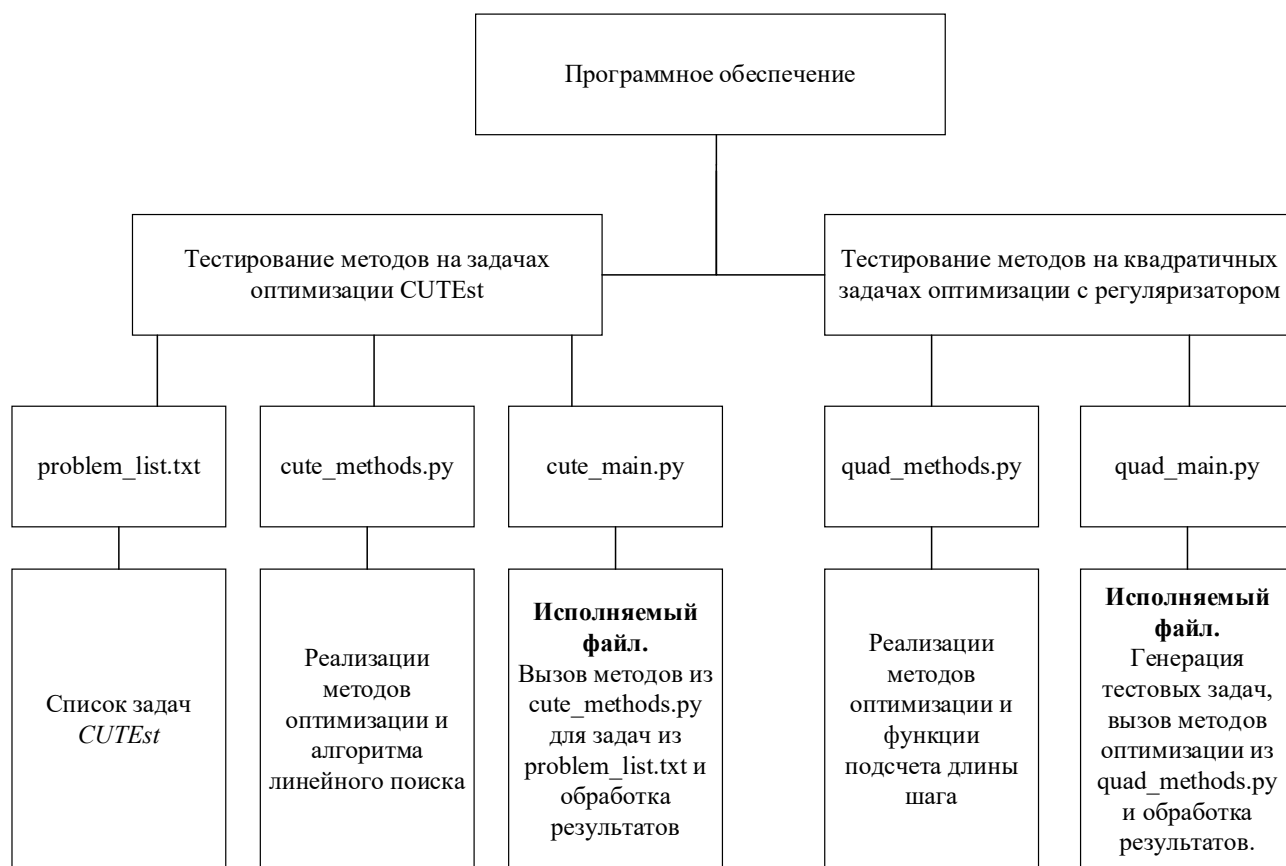


Рисунок 9 – Схема разработанного программного обеспечения

Первый модуль разработанного программного продукта представляет собой два файла формата *.py* и один файл формата *.txt*.

- *problem_list.txt* – текстовый файл, содержащий список наименований тестовых задач из коллекции *CUTEst* и их размерностей. Для тех задач, размерность которых фиксирована, после имени функции параметр не указывается;

- *cute_methods.py* – файл программного кода на языке *Python*, содержащий в себе реализацию разработанного алгоритма и всех методов оптимизации, с которым производился сравнительный анализ, а также функцию вызова задачи оптимизации из библиотеки *PyCUTEst*. Всего помимо предлагаемого в данной работе метода было реализовано 10

алгоритмов поиска минимума функции, 4 из которых являются базовыми двухкомпонентными методами оптимизации, еще 4 – их трехкомпонентными версиями. Остальные два метода являются алгоритмами, предложенными в работах [21, 25]. Для вычисления длины шага α_k в данном программном файле реализован линейный поиск на основе усиленных условий Вольфе. В случае, когда получить искомое значение не представляется возможным, совершается попытка вычисления параметра α_k , удовлетворяющего условиям Армихо, с помощью вызова функции *line_search_armijo* из библиотеки *SciPy*. Если такое α_k также не может быть найдено, работа метода оптимизации над решением данной задачи прекращается и считается неуспешной;

- *cute_main.py* является исполняемым файлом первого модуля проекта. В рамках данного файла выполняется считывание списка имен и параметров тестовых задач из тестового файла *problem_list.txt* в двумерный массив, после чего вызывает все разработанные методы из файла *cute_methods.py* для каждой из задач оптимизации и обрабатывает полученные результаты, а именно формирует технический отчет в формате таблицы с указанием характеристик работы каждого метода на множестве тестовых функций и изображает графики профилей производительности методов.

Второй модуль разработанного программного обеспечения представляет из себя два файла с программным кодом для тестирования на квадратичных задачах с L_2 -регуляризатором.

- *quad_methods.py* содержит реализации вычисления значения функции и градиента тестовой квадратичной задачи, а также функцию для подсчета длины шага α_k , согласно формуле (5.1). Более того, в данном файле реализованы все те же методы оптимизации, что и в файле *cute_methods.py*. Реализации методов в двух представленных модулях программного обеспечения во многом отличаются в силу специфики работы с библиотекой *PyCUTEst*, а также в связи с использованием или отсутствием линейного поиска для подсчета длины шага алгоритма;

- *quad_main.py* является исполняемым файлом второго модуля программного обеспечения. В процессе работы данного файла происходит генерация тестовых задач в количестве 1000 штук для каждой из десяти различных размерностей. Таким образом, в сумме генерируется 10 000 тестовых задач. После этого происходит поочередный вызов методов оптимизации из файла *quad_methods.py*. Результаты, полученные в ходе работы каждого из методов, записываются в технический отчет и используются для построения графиков профилей производительности методов.

В процессе работы над реализацией программного обеспечения для проведения численных экспериментов были сформулированы постановки задач, а также определены инструменты и средства разработки.

Результатом, демонстрирующим достижение поставленных в работе целей, является разработанный программный продукт, выдающий в качестве выходного результата технический отчет о характеристиках решения методами задач оптимизации и графики профилей производительности, позволяющие провести сравнительный анализ методов оптимизации.

Фрагменты программного кода, а также текстовый файл со списком тестовых задач *CUTEst* приведены в приложении В.

8 Анализ полученных результатов

Основным результатом данной дипломной работы является разработанный трехкомпонентный метод для решения задач оптимизации. Для оценки качества его работы была проведена серия численных экспериментов на задачах оптимизации из коллекции *CUTEst* и на квадратичных задачах с L_2 -регуляризатором. Кроме того, были отобраны еще 10 методов оптимизации для проведения сравнительного анализа разработанного алгоритма с другими. Список всех реализованных в программном обеспечении методов оптимизации, включая разработанный в данном исследовании, приведен в таблице 1. Длина шага α_k для всех методов выбирается при помощи алгоритма линейного поиска, поэтому отличие методов заключается в способе выбора направления p_k и параметра β_k . Стоит отметить, что все методы относятся к группе методов сопряженного градиента.

Таблица 1 – Список рассматриваемых методов оптимизации

Название 1	Параметр β_k 2	Направление p_k 3
1. Базовый двухкомпонентный метод Флетчера–Ривза	$\beta_k^{FR} = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$	$p_k = -g_k + \beta_k p_{k-1}$
2. Трёхкомпонентная форма базового метода Флетчера–Ривза	$\beta_k^{FR} = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$	$p_k = -g_k + \beta_k p_{k-1} - \beta_k \frac{g_k^T p_{k-1}}{g_k^T g_k} g_k$
3. Базовый двухкомпонентный метод Поляка–Рибьера	$\beta_k^{PR} = \frac{g_k^T (g_k - g_{k-1})}{\ g_{k-1}\ ^2}$	$p_k = -g_k + \beta_k p_{k-1}$
4. Трёхкомпонентная форма базового метода Поляка–Рибьера	$\beta_k^{PR} = \frac{g_k^T (g_k - g_{k-1})}{\ g_{k-1}\ ^2}$	$p_k = -g_k + \beta_k p_{k-1} - \beta_k \frac{g_k^T p_{k-1}}{g_k^T g_k} g_k$
5. Базовый двухкомпонентный метод Гестенса–Штифеля	$\beta_k = \frac{g_k^T (g_k - g_{k-1})}{(g_k - g_{k-1})^T p_{k-1}}$	$p_k = -g_k + \beta_k p_{k-1}$
6. Трёхкомпонентная форма базового метода Гестенса–Штифеля	$\beta_k = \frac{g_k^T (g_k - g_{k-1})}{(g_k - g_{k-1})^T p_{k-1}}$	$p_k = -g_k + \beta_k p_{k-1} - \beta_k \frac{g_k^T p_{k-1}}{g_k^T g_k} g_k$
7. Базовый двухкомпонентный метод Дай–Юань	$\beta_k = \frac{\ g_k\ ^2}{p_{k-1}^T (g_k - g_{k-1})}$	$p_k = -g_k + \beta_k p_{k-1}$
8. Трёхкомпонентная форма базового метода Дай–Юань	$\beta_k = \frac{\ g_k\ ^2}{p_{k-1}^T (g_k - g_{k-1})}$	$p_k = -g_k + \beta_k p_{k-1} - \beta_k \frac{g_k^T p_{k-1}}{g_k^T g_k} g_k$
9. Трёхкомпонентный метод из работы [25], основанный на базовом методе Дай–Ляо	$\beta_k = \frac{\ g_k\ ^2 - \frac{\ g_k\ }{\ g_{k-1}\ } g_k^T g_{k-1} }{\mu g_k^T p_{k-1} - p_{k-1}^T g_{k-1}} - t \frac{g_k^T s_{k-1}}{p_{k-1}^T y_{k-1}}$	$p_k = -g_k + \beta_k p_{k-1} - \beta_k \frac{g_k^T p_{k-1}}{g_k^T g_k} g_k$

Продолжение таблицы 1

1	2	3
10. Трёхкомпонентный гибридный метод из работы [21], основанный на базовых методах Флетчера–Ривза и Поляка–Рибьери	$\beta_k = \begin{cases} \beta_k^{PR}, & \text{если } \ g_k\ ^2 > g_k^T g_{k-1} \\ (1 - \theta_k)\beta_k^{NPR} + \theta_k\beta_k^{FR} \end{cases}$	$p_k = -g_k + \beta_k p_{k-1} - \beta_k \frac{g_k^T p_{k-1}}{g_k^T g_k} g_k$
11. Новый разработанный трёхкомпонентный метод оптимизации	$\beta_k = \begin{cases} \beta_k^{PR}, & \text{если } \ g_k\ ^2 > g_k^T g_{k-1} \\ (1 - \theta_k)\beta_k^{NPR} + \theta_k\beta_k^{FR} \end{cases}$	$p_k = -\omega_k g_k + \beta_k p_{k-1} - \omega_k \beta_k \frac{g_k^T p_{k-1}}{g_k^T g_k} g_k$

Вследствие проведенных численных экспериментов из 11 описанных выше методов были отобраны 5 алгоритмов, включая предлагаемый в данной работе, наиболее успешно справляющихся с исходными задачами оптимизации. Этими методами являются двухкомпонентные базовые алгоритмы Флетчера–Ривза, Поляка–Рибьери, Гестенса–Штифеля, трехкомпонентный гибридный метод из работы [21] и метод, разработанный в данном исследовании.

На рисунках 10 и 11 приведены основные результаты работы первого модуля программы – графики профилей производительности методов на малых и больших значениях t соответственно.

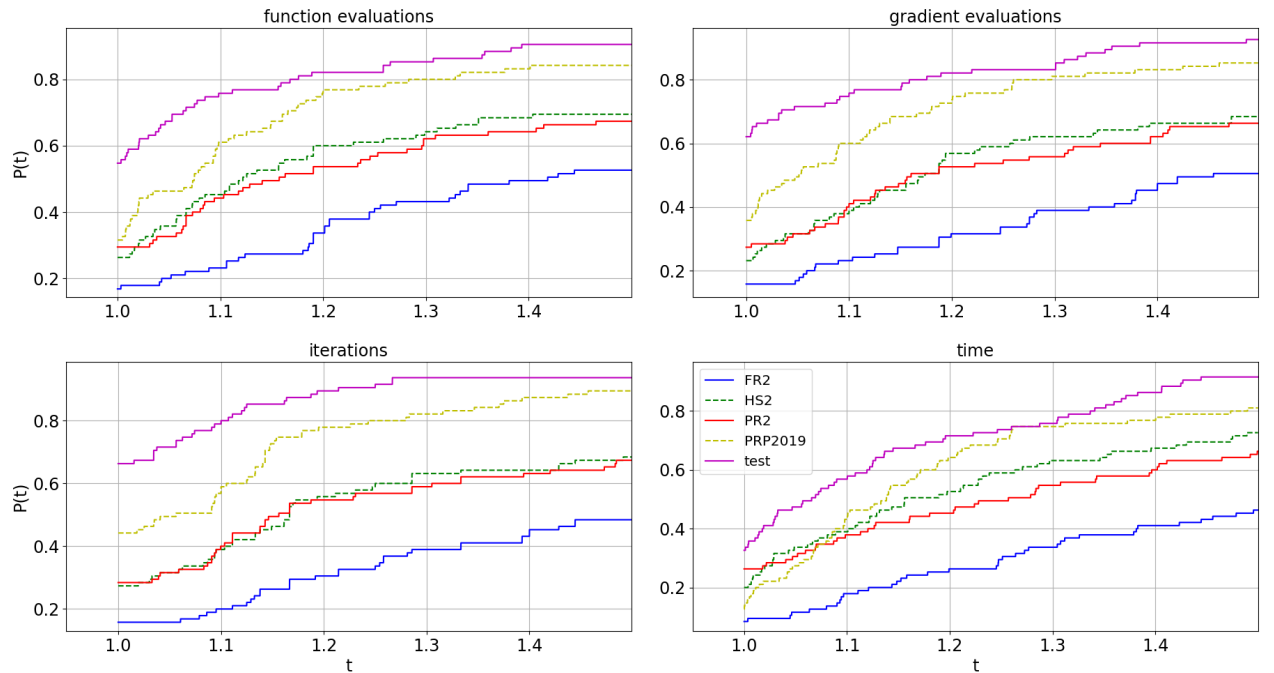


Рисунок 10 – Профили производительности методов при решении задач CUTEst на малых значениях t

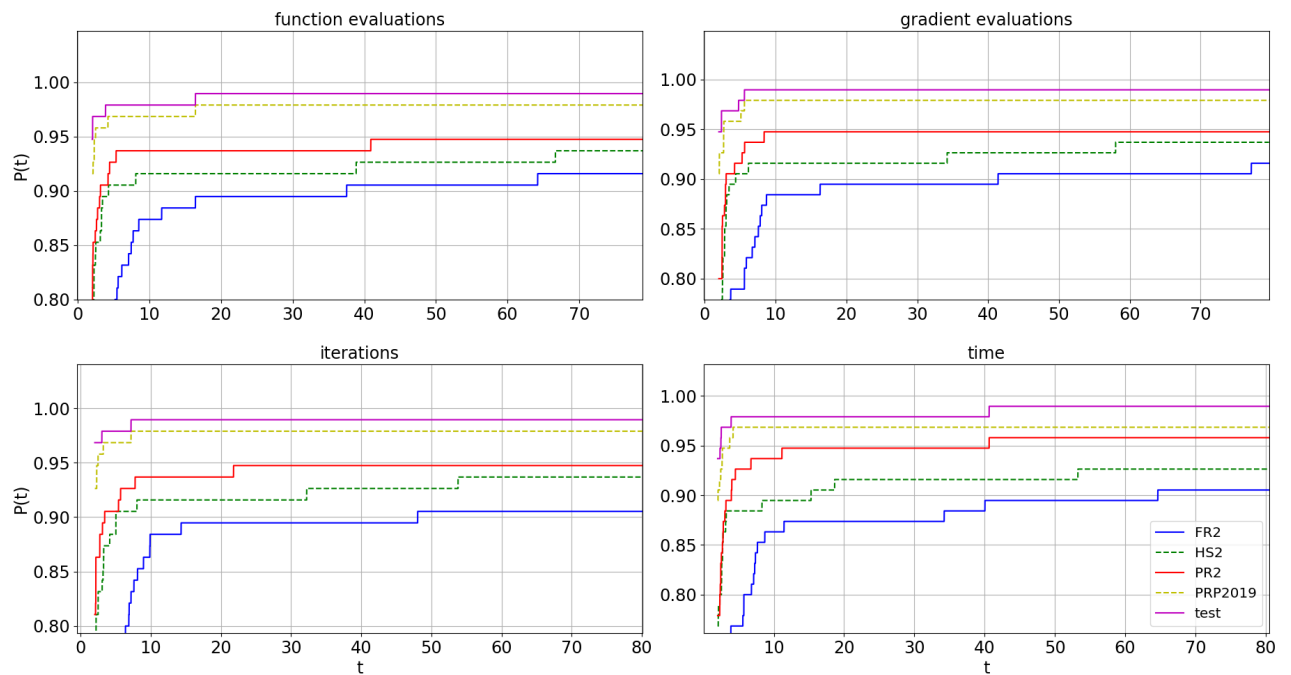


Рисунок 11 – Профили производительности методов при решении задач CUTEst на больших значениях t

Как уже было описано ранее, основной интерес на графиках профилей производительности представляют значения $p_s(1)$ и $p_s(r_{\max})$.

Значение $p_s(1)$ – это вероятность того, что данный метод окажется лучше других в данной метрике большее количество раз. На рисунке 10 видно, что разработанный метод демонстрирует лучшую производительность по всем четырем оцениваемым характеристикам – количеству вычислений значений функции и ее градиента в точке, числу выполненных итераций и затраченному времени. Другими словами, с точки зрения количества выполненных итераций разработанный метод с вероятностью около 70% является лучшим. Вероятность, что данный алгоритм выполнит меньше всех подсчетов значений функции и ее градиента в точке около 60%. Наконец, с точки зрения времени, затраченного на решения задачи оптимизации, разработанный метод также превосходит остальные алгоритмы.

Для оценки работы методов при $\tau \rightarrow r_{\max}$ приведен рисунок 11. На нем видно, что с точки зрения высокой вероятности успеха решения разработанный метод также является лучшим. Ближайшим к нему по производительности является гибридный метод из работы [21], взятый за основу интересующего нас метода. Другими словами, эти два метода решают успешно большее количество задач оптимизации нежели все остальные методы. Об этом свидетельствует и таблица ошибок, приведенная в приложении Б, согласно которой разработанный метод и гибридный метод из работы [21] из 95 тестовых задач не справились с

минимизацией *двух* и *одной* функций соответственно, тогда как остальные методы не нашли решение в среднем в *шести* задачах.

На основании вышесказанного можно утверждать, что разработанный метод работает успешнее всех остальных рассмотренных в данной работе алгоритмов оптимизации из класса сопряженных градиентов по всем четырем характеристикам. Он является как самым быстрым и экономичным, так и самым надежным методом оптимизации среди всех изученных.

На рисунках 12 и 13 изображены профили производительности рассматриваемых пяти методов для числа итераций и затраченного времени на множестве квадратичных задач с L_2 -регуляризатором. Сравнение количества выполненных вычислений значения функции и ее градиента в точке не имеет смысла, поскольку оно равно числу итераций в силу отсутствия алгоритма линейного поиска для выбора длины шага. Значение функции и градиента рассчитывается лишь единожды при получении нового значения x_k .

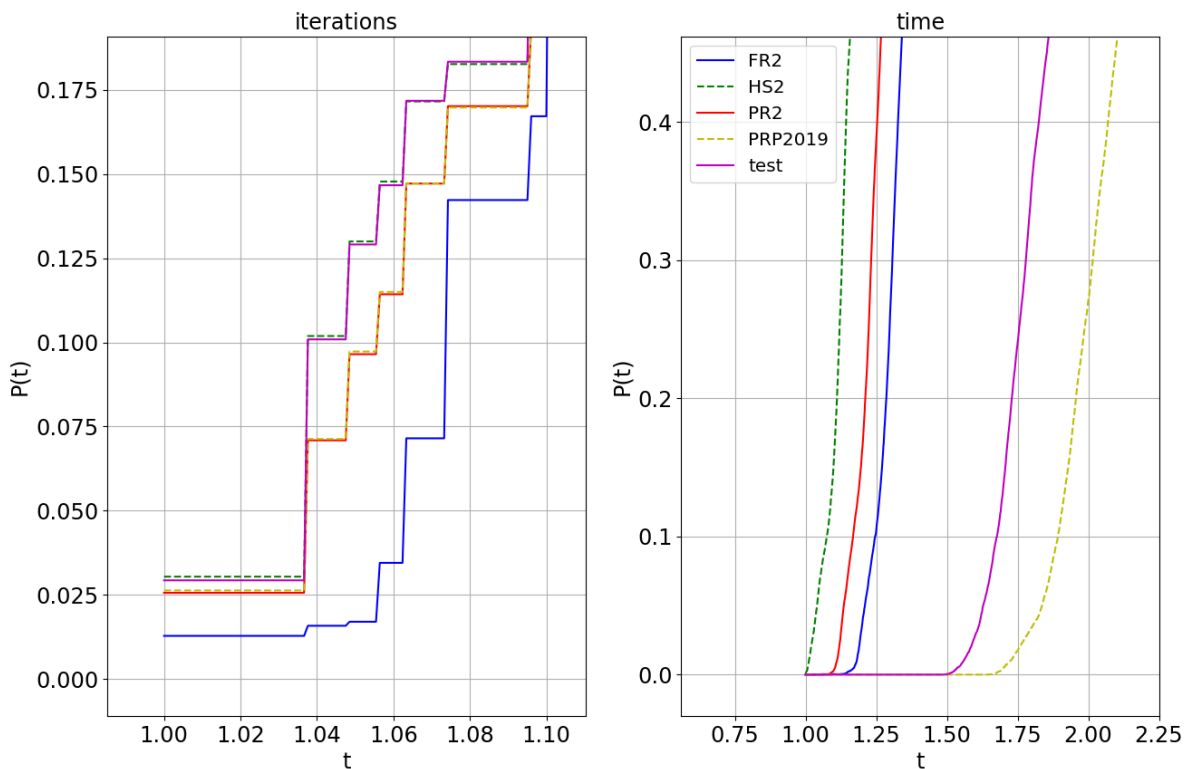


Рисунок 12 – Профили производительности методов при решении квадратичных задач на малых значениях t

На графике рисунке 12 при малых значениях t видно, что с точки зрения числа итераций наиболее быстрыми являются разработанный в данном исследовании метод и двухкомпонентный алгоритм Гестенса–Штиффеля. Больше всех шагов для решения квадратичной задачи требуется совершить двухкомпонентный алгоритм Флетчера–Ривза.

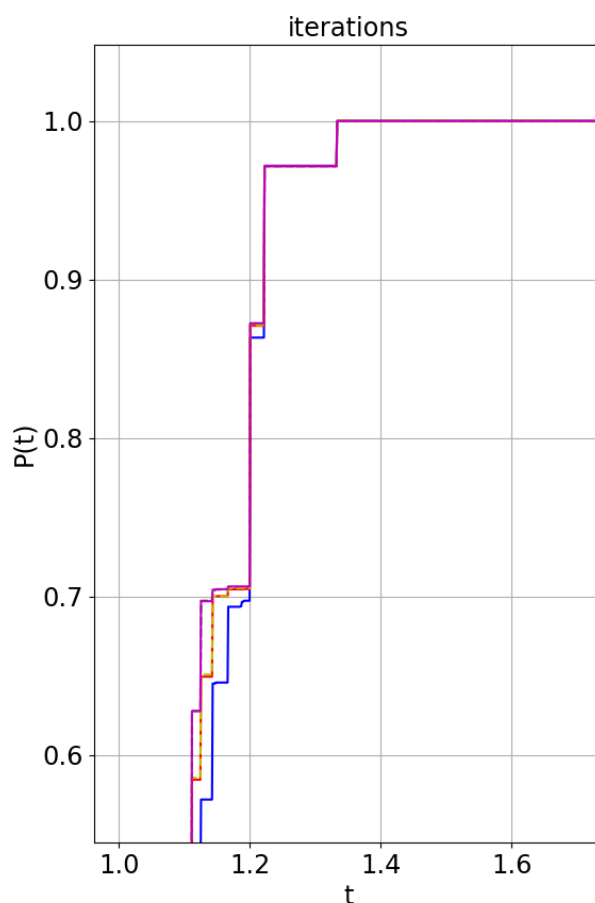


Рисунок 13 – Профили производительности методов при решении квадратичных задач на больших значениях t

Однако на рисунке 13 видно, что начиная с $t = 1.2$ все методы за исключением алгоритма Флетчера–Ривза начинают показывать одинаковую производительность. То есть, с точки зрения количества успешно решенных задач разработанный метод работает также успешно, как и остальные алгоритмы.

С точки зрения затраченного на решение задачи времени разработанный метод оптимизации работает медленнее большинства алгоритмов, обойдя лишь гибридный метод из работы [21].

Таким образом, если приоритетным является скорость получения результата, то для квадратичных задач с L_2 -регуляризатором рекомендуется использовать более простые базовые методы оптимизации. Однако если целью является сокращение числа итераций, то разработанный метод, безусловно, является более предпочтительным.

Технический отчет, полученный в ходе проведения численных экспериментов, приведен в приложении Б.

ЗАКЛЮЧЕНИЕ

В ходе данной работы был проведен аналитический обзор доступных источников информации в рамках исследуемой предметной области, сформулированы содержательная и математическая постановки задач, выбраны средства информационно–коммуникационных технологий для достижения поставленных целей и сформировано множество тестовых задач оптимизации. Вследствие проведенной работы был разработан новый трехкомпонентный метод сопряженного градиента, а также программное обеспечение, нацеленное на проведение тестирования данного алгоритма и обработку полученных результатов. В ходе серии численных экспериментов на множестве тестовых задач разработанный метод оптимизации продемонстрировал высокую производительность, что доказывает его конкурентоспособность и применимость в рамках задач оптимизации без ограничений.

В рамках исследования литературы были выделены наиболее популярные и успешные разновидности методов сопряженных градиентов, продемонстрированы их различия и преимущества относительно друг друга. Кроме того, был проведен анализ способов выбора множества задач для тестирования работы метода на основании популярных исследований в области методов оптимизации. Наконец, были рассмотрены различные способы представления полученных в ходе исследования результатов работы метода. Следующими этапами работы были разработка нового трехкомпонентного метода сопряженного градиента, его программная реализация и проведение серии численных экспериментов на множестве тестовых задач.

Результаты, полученные в ходе настоящего исследования, доказали высокую конкурентоспособность разработанного метода, что говорит о достижении всех поставленных в работе целей и выполнении всех сформулированных задач в полном объеме и качестве. Предложенный метод оптимизации может быть использован в любых сферах, решающих задачи поиска оптимального значения целевой функции без ограничений.

В рамках выполнения выпускной квалификационной работы были приобретены как профессиональные, так и системные и инструментальные компетенции, а именно были развиты способности поиска, изучения и анализа англоязычной научной литературы, усовершенствованы навыки структурированного и концептуального изложения изученной информации и получены новые компетенции в сфере разработки программного обеспечения и интерпретации полученных результатов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 П.А. Егубова. Трехкомпонентные методы сопряженного градиента для решения задач оптимизации большой размерности. // 76-е Дни Науки Студентов НИТУ «МИСиС». –2021.
- 2 Z. Jijun, L. Haibo, C. Zhiwei. The technology of intelligent recognition for drilling formation based on neural network with conjugate gradient optimization and remote wireless transmission. // Computer Communications. –2020. –V.156. –P.35–45.
- 3 Y. Shengwei, F. Qinliang, L. Lue, X. Jieqiong. A class of globally convergent three-term Dai–Liao conjugate gradient methods. // Applied Numerical Mathematics. –2020. – V.151. –P.354–366.
- 4 J. Nocedal, S. J. Wright. Numerical Optimization. New York: Springer Science+Business Media, 2006.
- 5 R. Bulirsch, J. Stoer. Introduction to Numerical Analysis. New York: Springer–Verlag, 1980.
- 6 R. Brent. Algorithms for minimization without derivatives. NJ: Prentice Hall, Englewood Cliffs, 1973.
- 7 J. E. Dennis, R. B. Schnabel. Numerical Methods for Unconstrained Optimization and Nonlinear Equations. NJ: Prentice–Hall, Englewood Cliffs, 1983.
- 8 C. Lemarechal. A view of line searches. // Lecture Notes in Control and Information Science. –1981. – V.30. –P.59–78.
- 9 R. Fletcher. Practical Methods of Optimization. New York: JohnWiley & Sons, 1987.
- 10 J. More, D. Thuente. Line search algorithms with guaranteed sufficient decrease. // ACM Transactions on Mathematical Software. –1994. – V.20. –P.286–307.
- 11 W. Hager, H. Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. // SIAM Journal on Optimization. –2005. – V.16. –P.170–192.
- 12 M. Hestenes, E. Stiefel. Methods of conjugate gradients for solving linear systems. // Journal of Research of the National Bureau of Standards. –1952. – V.49. –P.409–436.
- 13 R. Fletcher, C. Reeves. Function minimization by conjugate gradients. // Computer Journal. –1964. – V.7. –P.149–154.
- 14 M. Powell. Some convergence properties of the conjugate gradient method. // Mathematical Programming. –1976. – V.11. –P.42–49.
- 15 E. Polak, G. Ribiere. Note sur la convergence de methodes de directions conjuguées. // Revue Française d’Informatique et de Recherche Opérationnelle. –1969. – V.16. –P.35–43.
- 16 M. Powell. Nonconvex minimization calculations and the conjugate gradient method. // Lecture Notes in Mathematics. –1984. –V.1066. –P.122–141.

- 17 J. Gilbert, J. Nocedal. Global convergence properties of conjugate gradient methods for optimization. // SIAM Journal on Optimization. –1992. – V.2. –P.21–42.
- 18 Y. Dai, Y. Yuan. A nonlinear conjugate gradient method with a strong global convergence property. // SIAM Journal on Optimization. –1999. – V.10. –P.177–182.
- 19 W. Hager, H. Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. // SIAM Journal on Optimization. –2005. – V.16. –P.170–192.
- 20 L. Zhang. An improved Wei–Yao–Liu nonlinear conjugate gradient method for optimization computation. // Applied Mathematics and Computation. –2009. – V.215. –P.2269–2274.
- 21 P. Mtagulwa, P. Kaelo. An efficient modified PRP–FR hybrid conjugate gradient method for solving unconstrained optimization problems. // Applied Numerical Mathematics. –2019. – V.145. –P.111–120.
- 22 Y. Narushima, H. Yabe, J. A. Ford. A Three–Term Conjugate Gradient Method with Sufficient Descent Property For Unconstrained Optimization. // SIAM Journal on Optimization. –2011. – V.21. –P.212–230.
- 23 I. Bongartz, A. Conn, N. Gould, P. Toint. CUTE: constrained and unconstrained testing environment. // ACM Transactions on Mathematical Software. –1995. – V.21. –P.123–160.
- 24 E. Dolan, J. More. Benchmarking optimization software with performance profiles. // Mathematical Programming. –2002. – V.91. –P.201–213.
- 25 Y. Zheng, B. Zheng. Two New Dai–Liao–Type Conjugate Gradient Methods for Unconstrained Optimization Problems. // Journal of Optimization Theory and Applications. –2017. – V.175. –P.502–509.
- 26 P. Mtagulwa, K. P. An efficient modified PRP–FR hybrid conjugate gradient method for solving unconstrained optimization problems. // Applied Numerical Mathematics. –2019. – V.145. –P.111–120.
- 27 L. Zhang, W. Zhou, D. Li. Global convergence of a modified Fletcher–Reeves conjugate gradient method with Armijo–type line search. // Numerische Mathematik. –2006. – V.104. –P.561–572.

ПРИЛОЖЕНИЕ А

Таблица А.1 – Задачи оптимизации для тестирования разрабатываемого метода

Задача	Описание	Размерность
1	2	3
ARGTRIGLS	Тригонометрическая задача оптимизации. Представляет из себя сумму N групп наименьших квадратов, каждая из которых имеет $N+1$ нелинейных элементов.	200
ARWHEAD	Задача оптимизации четвертой степени, матрица вторых производных которой представляет собой стрелку, направленную вниз, с диагональной центральной частью и шириной границы 1. $f(x) = \sum_{i=0}^{n-2} ((x_i^2 + x_{n-1})^2 - 4x_i + 3)$	5000
BA-L1LS	Задача настройки пучка из реконструктивной геометрии, в которой коллекция фотографий используется для определения положения набора наблюдаемых точек. Каждая наблюдаемая точка видима через ее двумерные проекции на подмножество фотографий. Представляет из себя нелинейную задачу наименьших квадратов.	57
BDQRTIC	Задача оптимизации четвертой степени. $f(x) = \sum_{i=0}^{n-4} ((3 - 4x_i)^2 + (x_i^2 + 2x_{i+1}^2 + 3x_{i+2}^2 + 4x_{i+3}^2 + 5x_{n-1}^2)^2)$	1000
BOXPOWER	Задача оптимизации, матрица вторых производных которой является вырожденной, то есть, квадратной с определителем равным нулю.	100
BROYDN3DLS	Трехдиагональная система нелинейных уравнений Бroyдена в форме наименьших квадратов. $f(x) = \sum_{i=1}^n ((3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 1)$	10000
BROWNAL	Почти линейная задача наименьших квадратов. Представляет из себя сумму N групп наименьших квадратов, последняя из которых имеет нелинейный элемент. $f(x) = \sum_{i=1}^n \left(x_i + \sum_{j=1}^n x_j - (n+1) \right) + \prod_{j=1}^n x_j - 1$	10
BROYDN7D	Семидиагональный вариант трехдиагональной системы Бroyдена с полосой, удаленной от диагонали. $f(x) = (1 - 2x_1 + (3 - 2x_0)x_0)^{\frac{7}{3}} + (1 - x_{n-2} + (3 - 2x_{n-1})x_{n-1})^{\frac{7}{3}} + \sum_{i=0}^{0,5n-1} (x_i + x_{i+0,5n})^{\frac{7}{3}} + \sum_{i=1}^{n-2} (1 - x_{i-1} - 2x_{i+1} + (3 - 2x_i)x_i)^{\frac{7}{3}}$	2500

Продолжение таблицы А.1

1	2	3
BROYDNBDLS	<p>Система нелинейных уравнений Бroyдена, рассматриваемая в форме наименьших квадратов.</p> $f(x) = \sum_{i=1}^n \left(x_i(2 + 5x_i^2) + 1 - \sum_{j \in J} (x_j(1 + x_j)) \right)$ $J = \{j: j \neq i, \max(1, i - 5) \leq j \leq \min(n, i + 1)\}$	10000
BRYBND	<p>Система нелинейных уравнений Бroyдена, рассматриваемая в форме наименьших квадратов.</p> $f(x) = \sum_{i=1}^n \left(x_i(2 + 5x_i^2) + 1 - \sum_{j \in J} x_j(1 + x_j) \right)^2$ $J = \{j: j \neq i, \max(1, i - 5) \leq j \leq \min(n, i + 1)\}$	10000
CHNROSNB	<p>Цепная функция Розенброка.</p> $f(x) = \sum_{i=2}^{25} (4(x_{i-1} - x_i^2)^2 + (1 - x_i)^2)$	50
CHNRSNB	<p>Модифицированная цепная функция Розенброка.</p> $f(x) = \sum_{i=2}^n (16(x_{i-1} - x_i^2)^2(1.5 + \sin(i))^2 + (x_i - 1)^2)$	10
COSINE	$f(x) = \sum_{i=0}^{n-2} \cos(x_i^2 - 0.5x_{i+1})$	10000
DIXON3DQ	<p>Трёхдиагональная квадратичная система Диксона.</p> $f(x) = (x_0 - 1)^2 + (x_{n-1} - 1)^2 + \sum_{i=1}^{n-2} (x_i - x_{i+1})^2$	10000
ECKERLE4LS	$f(x) = \frac{\beta_1}{\beta_2} e^{-\frac{(x-\beta_3)^2}{2\beta_2^2}}$	3
EGGCRAVE	<p>Пример задачи оптимизации из библиотеки SciPy.</p> $f(x) = x_1^2 + x_2^2 + 25(\sin^2(x_1) + \sin^2(x_2))$	4
ELATVIDU	<p>Пример задачи оптимизации из библиотеки SciPy.</p> $f(x) = (x_1^2 + x_2 - 10)^2 + (x_1 + x_2^2 - 7)^2 + (x_1^2 + x_2^3 - 1)^2$	2
EXP2	<p>Пример задачи оптимизации из библиотеки SciPy.</p> $f(x) = \sum_{i=0}^9 (e^{-ix_1/10} - 5e^{-ix_2/10} - e^{-i/10} + 5e^{-i})^2$	2
EXTROSNB	<p>Расширенная функция Розенброка.</p> $f(x) = (x_0 - 1)^2 + 100 \sum_{i=1}^{n-1} (x_i - x_{i-1}^2)^2$	1000

Продолжение таблицы А.1

1	2	3
FLETGBV2	Красевая задача. Представляет из себя симметричную систему уравнений. $f(x) = \frac{1}{2} \left(x_1^2 + \sum_{i=1}^{n-1} (x_i - x_{i+1} + 1)^2 + x_n^2 \right) - h^2 \sum_{i=1}^n (2x_i + \cos x_i) - x_n$	10000
FLETCHCR	Цепная функция Розенброка, заданная Флетчером. $f(x) = 100 \sum_{i=1}^{n-1} (x_{i+1} - x_i + 1 - x_i^2)^2$	1000
FREUROTH	Тестовая задача Фрейдентштейна и Рота. $f(x) = \sum_{i=0}^{n-2} (x_i - 13 + ((5 - x_{i+1})x_{i+1} - 2x_{i+1})^2 + (x_i - 29 + ((1 + x_{i+1})x_{i+1} - 14)x_{i+1})^2$	1000
GULF	Тестовая задача оптимизации размерности 3. $f(x) = \sum_{i=0}^n e^{-\frac{(y_i x_2)^{x_3}}{x_1}} - \frac{i}{100}, \text{ где } y_i = 25 + \left(-50 \ln \left(\frac{i}{100} \right) \right)^{2/3}$	3
HATFLDD	Экспоненциальная задача тестирования методов оптимизации из руководства пользователя OPTIMA.	3
HILBERTA	Квадратичная задача Гильберта.	10
HIMMELBG	Задача оптимизации Химмельблау размерности 2.	2
INTEQNELS	Дискретная интегральная задача без фиксированных переменных в форме наименьших квадратов. $f(x) = \sum_{i=1}^n 1 x_i + 0.5h \left((1 - t_i) \sum_{j=1}^i t_j (x_j + t_j + 1)^3 + t_i \sum_{j=i+1}^n (1 - t_j)(x_j + t_j + 1)^3 \right)$	500
LANCZOS1LS	Задача нелинейной регрессии. $y = \beta_1 e^{-\beta_2 x} + \beta_3 e^{-\beta_4 x} + \beta_5 e^{-\beta_6 x}$	6
LANCZOS2LS	Задача нелинейной регрессии. $y = \beta_1 e^{-\beta_2 x} + \beta_3 e^{-\beta_4 x} + \beta_5 e^{-\beta_6 x}$	6
NONDQUAR	Недиагональная квадратичная тестовая задача. Матрица вторых производных сингулярна в решении. $f(x) = (x_0 - x_1)^2 + (x_{n-2} + x_{n-1})^2 + \sum_{i=0}^{n-3} (x_i + x_{i+1} + x_{n-1})^4$	1000
OSBORNEB	$f(x) = y - (x_1 e^{-tx_5} + x_2 e^{-(t-x_9)^2 x_6} + x_3 e^{-(t-x_{10})^2 x_7} + x_4 e^{-(t-x_{11})^2 x_8})$	11
OSCIPATH	Проблема "колебательного пути" Юрия Нестерова.	500

Продолжение таблицы А.1

1	2	3
PENALTY1	$f(x) = 10^{-5} \sum_{i=0}^{n-1} (x_i - 1)^2 + \left(\sum_{i=0}^{n-1} x_i^2 - 0.25 \right)^2$	500
QING	<p>Пример задачи оптимизации из библиотеки SciPy.</p> $f(x) = \sum_{i=1}^n (x_i^2 - i)^2$	1000
S308	$f(x) = (x_1^2 + x_2^2 + x_1 x_2)^2 + (\sin x_1)^2 + (\sin x_2)^2$	2
SISSER	$f(x) = (x_1 + 1)^2 + (x_2 - 4)^2$	2
SNAIL	<p>Двухмерная задача оптимизации, график которой представляет из себя изображение спиральной долины. Посвящается городу Намюр, эмблемой которого является улитка (snail).</p>	2
STRCHDV	<p>Пример задачи оптимизации из библиотеки SciPy.</p> $f(x) = \sum_{i=1}^{n-1} x^{1/4} (\sin(50x^{0.1}) + 1)^2$	1000
TRIGON1	<p>Пример задачи оптимизации из библиотеки SciPy.</p> $f(x) = \sum_{i=1}^n \left(n - \sum_{j=1}^n \cos x_j + i(1 - \cos x_i - \sin x_i) \right)^2$	100
WATSON	<p>Задача Ватсона.</p> $f(x) = \sum_{j=2}^n (j-1)x_j t^{j-2} - \left(\sum_{j=1}^n x_j t^{j-1} \right)^2 - 1$	31

ПРИЛОЖЕНИЕ Б

Результаты проведенных численных экспериментов представлены на рисунке Б.1.

method	function	dimension	feva	geva	iterations	norm(gk)	time
FR2	ARGTRIGLS	10	281	249	67	7.6e-06	0.0180456
HS2	ARGTRIGLS	10	194	161	45	8.7e-06	0.0077427
PR2	ARGTRIGLS	10	183	150	43	7.6e-06	0.0089839
PRP2019	ARGTRIGLS	10	174	144	40	7e-07	0.0113862
test	ARGTRIGLS	10	148	120	35	8.4e-06	0.0072975
FR2	ARGTRIGLS	50	869	844	217	7.2e-06	0.6528127
HS2	ARGTRIGLS	50	1040	1016	258	9.2e-06	0.7536392
PR2	ARGTRIGLS	50	1199	1182	304	9.8e-06	0.8174611
PRP2019	ARGTRIGLS	50	888	860	222	8.5e-06	0.5786595
test	ARGTRIGLS	50	692	663	173	8.7e-06	0.5123811
FR2	ARGTRIGLS	100	1942	1928	517	9.9e-06	9.964655
HS2	ARGTRIGLS	100	2051	2037	547	9.9e-06	10.2097464
PR2	ARGTRIGLS	100	2564	2550	676	9.5e-06	12.8684909
PRP2019	ARGTRIGLS	100	1844	1827	488	8.3e-06	9.1913504
test	ARGTRIGLS	100	1822	1806	476	9.4e-06	9.0607327
FR2	ARGTRIGLS	200	3448	3432	983	9e-06	162.4313692
HS2	ARGTRIGLS	200	4340	4324	1226	9.9e-06	204.5515587
PR2	ARGTRIGLS	200	5603	5587	1496	8e-06	264.0706191
PRP2019	ARGTRIGLS	200	4346	4324	1222	9.5e-06	203.9569446
test	ARGTRIGLS	200	4055	4035	1101	9.5e-06	190.250825
FR2	ARWHEAD	100	64	28	9	2.8e-06	0.0031858
HS2	ARWHEAD	100	53	29	9	0.0	0.003675
PR2	ARWHEAD	100	52	23	8	4e-07	0.0044329
PRP2019	ARWHEAD	100	48	21	7	1e-07	0.0026342
test	ARWHEAD	100	49	21	7	1e-07	0.002742
FR2	ARWHEAD	500	72	32	9	6.5e-06	0.1388467
HS2	ARWHEAD	500	75	37	10	0.0	0.1284459
PR2	ARWHEAD	500	61	29	8	1e-07	0.1256874
PRP2019	ARWHEAD	500	67	30	8	0.0	0.097487
test	ARWHEAD	500	67	30	8	1e-07	0.0955707
FR2	ARWHEAD	1000	75	32	8	8.2e-06	0.435556
HS2	ARWHEAD	1000	67	27	8	4e-07	0.3477883
PR2	ARWHEAD	1000	64	29	8	2e-07	0.5125465
PRP2019	ARWHEAD	1000	59	22	6	4.3e-06	0.284668
test	ARWHEAD	1000	59	22	6	4.2e-06	0.2825202
FR2	ARWHEAD	5000	93	42	10	4e-07	31.2376177
HS2	ARWHEAD	5000	70	30	8	7.9e-06	18.3769464
PR2	ARWHEAD	5000	84	45	10	0.2979291	34.9268709
PRP2019	ARWHEAD	5000	93	48	11	0.0	30.4432729
test	ARWHEAD	5000	185	28	7	1.8284304	18.4285349
FR2	BA-L1LS	57	267	222	61	8.9e-06	0.3492834
HS2	BA-L1LS	57	135	96	29	8.6e-06	0.0128743
PR2	BA-L1LS	57	154	111	33	4.8e-06	0.0139537
PRP2019	BA-L1LS	57	139	98	30	7.2e-06	0.0161027
test	BA-L1LS	57	140	98	30	5.1e-06	0.014926
FR2	BDQRTIC	100	597	499	122	7.3e-06	0.0511111
HS2	BDQRTIC	100	450	356	86	4.9e-06	0.0366799
PR2	BDQRTIC	100	487	402	99	8.8e-06	0.0387815
PRP2019	BDQRTIC	100	383	299	71	8.2e-06	0.0324194
test	BDQRTIC	100	380	302	75	9.1e-06	0.0334181
FR2	BDQRTIC	500	1596	754	211	3.49e-05	1.1026353
HS2	BDQRTIC	500	1397	942	227	8.13e-05	1.1584489
PR2	BDQRTIC	500	1244	791	200	0.0001204	1.0353535
PRP2019	BDQRTIC	500	1099456	100333	100000	5.35e-05	403.2482912
test	BDQRTIC	500	425	313	70	6.9e-06	0.3835096
FR2	BDQRTIC	1000	1080	649	143	0.0005306	9.3078216
HS2	BDQRTIC	1000	999	596	135	0.0002041	8.1767302
PR2	BDQRTIC	1000	24543	2782	1829	7.92e-05	62.2459288
PRP2019	BDQRTIC	1000	710	322	84	1.07e-05	5.707044
test	BDQRTIC	1000	600	439	93	7.1e-06	6.4015704
FR2	BOXPOWER	10	1323	1166	277	8.3e-06	0.0382277
HS2	BOXPOWER	10	178	149	37	3.2e-06	0.0063905
PR2	BOXPOWER	10	337	272	56	3e-07	0.0088979
PRP2019	BOXPOWER	10	192	157	31	9.9e-06	0.0084217
test	BOXPOWER	10	248	202	37	1.5e-06	0.0072475
FR2	BOXPOWER	100	18881	16434	3569	9.7e-06	1.1265951
HS2	BOXPOWER	100	294	227	45	5e-06	0.0172299
PR2	BOXPOWER	100	400	323	70	0.0	0.0221262
PRP2019	BOXPOWER	100	412	311	51	4e-07	0.0247724
test	BOXPOWER	100	295	213	35	3.3e-06	0.0150571
FR2	BROYDN3DLS	10	101	73	23	6.6e-06	0.0030085
HS2	BROYDN3DLS	10	100	74	23	9.7e-06	0.0029264
PR2	BROYDN3DLS	10	99	73	23	9.5e-06	0.0028343
PRP2019	BROYDN3DLS	10	98	73	23	7.6e-06	0.003336
test	BROYDN3DLS	10	96	67	21	3.3e-06	0.0047424
FR2	BROYDN3DLS	100	145	108	35	8.6e-06	0.0075974
HS2	BROYDN3DLS	100	129	96	31	7.3e-06	0.0092323
PR2	BROYDN3DLS	100	129	96	31	7.3e-06	0.0063221
PRP2019	BROYDN3DLS	100	129	96	31	7.3e-06	0.0077931
test	BROYDN3DLS	100	129	96	31	7.3e-06	0.0087403

	FR2	BROYDN3DLS	500	193	167	47	7.4e-06	0.1867906
	HS2	BROYDN3DLS	500	133	99	32	9.9e-06	0.0987857
	PR2	BROYDN3DLS	500	133	99	32	9.9e-06	0.1022958
	PRP2019	BROYDN3DLS	500	120	89	29	8.1e-06	0.0924116
	test	BROYDN3DLS	500	119	88	29	5.8e-06	0.1145654
	FR2	BROYDN3DLS	1000	190	158	46	9e-06	2.3117752
	HS2	BROYDN3DLS	1000	138	102	33	6.1e-06	1.2982567
	PR2	BROYDN3DLS	1000	138	102	33	6.3e-06	1.2998046
	PRP2019	BROYDN3DLS	1000	123	91	29	6.7e-06	1.1763992
	test	BROYDN3DLS	1000	123	91	29	6.3e-06	1.1593198
	FR2	BROYDN3DLS	5000	551	537	141	1e-05	339.7182151
	HS2	BROYDN3DLS	5000	184	152	44	6e-06	99.042306
	PR2	BROYDN3DLS	5000	203	170	48	8e-06	120.5371421
	PRP2019	BROYDN3DLS	5000	199	170	48	7.3e-06	119.8170005
	test	BROYDN3DLS	5000	184	151	44	8.9e-06	95.0455483
	FR2	BROYDN3DLS	10000	200	169	48	9.4e-06	595.9941888
	HS2	BROYDN3DLS	10000	144	109	34	7.7e-06	428.7077469
	PR2	BROYDN3DLS	10000	164	123	37	8.8e-06	476.2805699
	PRP2019	BROYDN3DLS	10000	141	106	32	5.7e-06	372.4666087
	test	BROYDN3DLS	10000	191	143	38	9.7e-06	456.3855182
	FR2	BROWNAL	10	110	59	13	0.0	0.0035746
	HS2	BROWNAL	10	93	54	13	6e-07	0.002964
	PR2	BROWNAL	10	168	7	3	46.065287	0.0039783
	PRP2019	BROWNAL	10	88	46	9	3.7e-06	0.0032841
	test	BROWNAL	10	88	46	9	3.1e-06	0.0031591
	FR2	BROYDN7D	25	387	374	98	8.9e-06	0.0196092
	HS2	BROYDN7D	25	206	178	50	6.6e-06	0.0131085
	PR2	BROYDN7D	25	206	180	50	9.4e-06	0.0092607
	PRP2019	BROYDN7D	25	192	166	46	9.5e-06	0.0123871
	test	BROYDN7D	25	192	164	46	8.8e-06	0.0121883
	FR2	BROYDN7D	50	396	384	99	8.6e-06	0.0323827
	HS2	BROYDN7D	50	213	193	52	7.3e-06	0.0146751
	PR2	BROYDN7D	50	218	196	53	8.6e-06	0.017499
	PRP2019	BROYDN7D	50	210	182	51	7.1e-06	0.0177988
	test	BROYDN7D	50	210	183	51	7e-06	0.0207145
	FR2	BROYDN7D	250	400	382	100	8.7e-06	0.4625932
	HS2	BROYDN7D	250	220	209	54	9.2e-06	0.22334
	PR2	BROYDN7D	250	221	210	54	8.8e-06	0.220568
	PRP2019	BROYDN7D	250	209	186	51	8.6e-06	0.2054644
	test	BROYDN7D	250	208	185	51	7.8e-06	0.2066283
	FR2	BROYDN7D	500	634	339	97	1.31e-05	4.9989056
	HS2	BROYDN7D	500	221	203	55	9.1e-06	2.9499944
	PR2	BROYDN7D	500	221	201	55	8.6e-06	2.8984085
	PRP2019	BROYDN7D	500	237	203	56	8.5e-06	3.0131506
	test	BROYDN7D	500	223	200	55	8.8e-06	2.9358539
	FR2	BROYDN7D	2500	503	251	70	7.33e-05	172.4385496
	HS2	BROYDN7D	2500	431	178	51	7.8e-05	121.0789861
	PR2	BROYDN7D	2500	424	182	51	3.78e-05	125.7003365
	PRP2019	BROYDN7D	2500	455	168	51	5.41e-05	116.6282828
	test	BROYDN7D	2500	441	167	49	6.78e-05	111.8246053
	FR2	BROYDNBDLS	100	161	125	39	7.1e-06	0.0143485
	HS2	BROYDNBDLS	100	155	117	36	6.7e-06	0.0132866
	PR2	BROYDNBDLS	100	134	98	31	6.8e-06	0.0097808
	PRP2019	BROYDNBDLS	100	304	261	70	9e-06	0.0287548
	test	BROYDNBDLS	100	274	231	61	4.8e-06	0.0249239
	FR2	BROYDNBDLS	500	173	132	39	8.1e-06	0.1929932
	HS2	BROYDNBDLS	500	289	260	69	9.7e-06	0.3334264
	PR2	BROYDNBDLS	500	262	228	60	9.1e-06	0.2732731
	PRP2019	BROYDNBDLS	500	141	106	32	8.4e-06	0.1289933
	test	BROYDNBDLS	500	129	93	28	5.4e-06	0.1109617
	FR2	BROYDNBDLS	1000	146	109	33	6.8e-06	1.3942537
	HS2	BROYDNBDLS	1000	274	243	63	7.6e-06	3.1229617
	PR2	BROYDNBDLS	1000	270	231	62	9.4e-06	2.9680628
	PRP2019	BROYDNBDLS	1000	132	95	29	6.9e-06	1.211515
	test	BROYDNBDLS	1000	139	98	30	8e-06	1.2545215
	FR2	BROYDNBDLS	5000	158	111	34	5.8e-06	69.3760987
	HS2	BROYDNBDLS	5000	177	131	40	7.2e-06	84.7996404
	PR2	BROYDNBDLS	5000	170	124	36	7.1e-06	78.3423427
	PRP2019	BROYDNBDLS	5000	152	112	33	1e-05	70.3274498
	test	BROYDNBDLS	5000	131	89	28	8.8e-06	57.5629643
	FR2	BROYDNBDLS	10000	183	142	42	6.6e-06	590.9903887
	HS2	BROYDNBDLS	10000	163	119	34	7e-06	508.6875392
	PR2	BROYDNBDLS	10000	147	103	30	7.5e-06	419.3603256
	PRP2019	BROYDNBDLS	10000	161	112	34	8.5e-06	484.3495733
	test	BROYDNBDLS	10000	185	134	38	8.8e-06	548.1467013
	FR2	BRYBND	100	161	125	39	7.1e-06	0.017522
	HS2	BRYBND	100	155	117	36	6.7e-06	0.016426
	PR2	BRYBND	100	134	98	31	6.8e-06	0.0149648
	PRP2019	BRYBND	100	304	261	70	9e-06	0.0367574
	test	BRYBND	100	274	231	61	4.8e-06	0.0504136
	FR2	BRYBND	500	173	132	39	8.1e-06	0.2745714
	HS2	BRYBND	500	289	260	69	9.7e-06	0.5081159
	PR2	BRYBND	500	262	228	60	9.1e-06	0.4630536
	PRP2019	BRYBND	500	141	106	32	8.4e-06	0.209235
	test	BRYBND	500	129	93	28	5.4e-06	0.2020721
	FR2	BRYBND	1000	146	109	33	6.8e-06	2.3482515
	HS2	BRYBND	1000	274	243	63	7.6e-06	4.8048873
	PR2	BRYBND	1000	270	231	62	9.4e-06	4.5570975
	PRP2019	BRYBND	1000	132	95	29	6.9e-06	1.7203835
	test	BRYBND	1000	139	98	30	8e-06	1.90032
	FR2	BRYBND	5000	158	111	34	5.8e-06	94.3649865

	HS2	BRYBND	5000	177	131	40	7.2e-06	109.6815423
	PR2	BRYBND	5000	170	124	36	7.1e-06	101.5593875
	PRP2019	BRYBND	5000	152	112	33	1e-05	88.1372954
	test	BRYBND	5000	131	89	28	8.8e-06	65.2054397
	FR2	BRYBND	10000	183	142	42	6.6e-06	437.1640474
	HS2	BRYBND	10000	163	119	34	7e-06	358.6684868
	PR2	BRYBND	10000	147	103	30	7.5e-06	337.3934034
	PRP2019	BRYBND	10000	161	112	34	8.5e-06	357.0158985
	test	BRYBND	10000	185	134	38	8.8e-06	408.0264157
	FR2	CHNROSNB	50	337052	337035	100000	6.0499532	18.388893
	HS2	CHNROSNB	50	1101	1059	266	9.7e-06	0.0604426
	PR2	CHNROSNB	50	1714	1683	422	9.9e-06	0.0961479
	PRP2019	CHNROSNB	50	1468	1408	350	8.9e-06	0.120618
	test	CHNROSNB	50	1276	1227	309	9.9e-06	0.0803837
	FR2	CHNRSNBM	10	163230	163214	46962	9.9e-06	5.7632654
	HS2	CHNRSNBM	10	927	837	211	5.6e-06	0.0331006
	PR2	CHNRSNBM	10	936	865	219	6.9e-06	0.0561543
	PRP2019	CHNRSNBM	10	694	607	148	1.7e-06	0.0285039
	test	CHNRSNBM	10	591	515	129	9.7e-06	0.0270422
	FR2	COSINE	10	573	570	143	1e-05	0.0346869
	HS2	COSINE	10	50	35	10	4.5e-06	0.0048329
	PR2	COSINE	10	49	35	10	5.3e-06	0.0032696
	PRP2019	COSINE	10	50	35	10	1e-06	0.0031453
	test	COSINE	10	50	35	10	1.1e-06	0.0018625
	FR2	COSINE	100	135	102	33	7.5e-06	0.0075526
	HS2	COSINE	100	47	32	10	4.1e-06	0.0025252
	PR2	COSINE	100	47	32	10	3.7e-06	0.005471
	PRP2019	COSINE	100	48	32	10	2.1e-06	0.0028496
	test	COSINE	100	48	32	10	2.1e-06	0.0033862
	FR2	COSINE	1000	65	42	13	5.3e-06	0.5803714
	HS2	COSINE	1000	48	33	10	3.5e-06	0.4393029
	PR2	COSINE	1000	48	33	10	3.5e-06	0.443484
	PRP2019	COSINE	1000	45	30	9	4.2e-06	0.3877333
	test	COSINE	1000	45	30	9	4.2e-06	0.386758
	FR2	COSINE	10000	49	32	10	3.8e-06	125.5108045
	HS2	COSINE	10000	48	32	10	5.3e-06	113.020059
	PR2	COSINE	10000	48	32	10	6.8e-06	115.0400675
	PRP2019	COSINE	10000	45	30	9	5.9e-06	110.8026593
	test	COSINE	10000	45	30	9	5e-06	86.410481
	FR2	DIXON3DQ	100	1629	1625	438	8e-06	0.1061087
	HS2	DIXON3DQ	100	1816	1811	475	8.5e-06	0.146298
	PR2	DIXON3DQ	100	2586	2581	677	9.1e-06	0.1852748
	PRP2019	DIXON3DQ	100	1739	1733	460	1e-05	0.1525378
	test	DIXON3DQ	100	1562	1551	406	9.6e-06	0.1292834
	FR2	DIXON3DQ	1000	17561	17557	5197	9.9e-06	255.6383178
	HS2	DIXON3DQ	1000	30806	30802	9063	9.9e-06	477.8142563
	PR2	DIXON3DQ	1000	52978	52972	14054	9.7e-06	686.8872822
	PRP2019	DIXON3DQ	1000	29560	29533	8538	8.9e-06	368.8620379
	test	DIXON3DQ	1000	24284	24277	5490	9.8e-06	370.8341038
	FR2	ECKERLE4LS	3	19	17	3	2e-07	0.0115205
	HS2	ECKERLE4LS	3	65	51	15	5e-07	0.0621671
	PR2	ECKERLE4LS	3	32	29	4	2e-07	0.0083507
	PRP2019	ECKERLE4LS	3	19	17	3	2e-07	0.0008789
	test	ECKERLE4LS	3	19	17	3	2e-07	0.0007171
	FR2	EGGCRATE	4	90	66	22	9.6e-06	0.0075321
	HS2	EGGCRATE	4	37	22	7	1e-07	0.0013592
	PR2	EGGCRATE	4	37	22	7	2e-07	0.0060612
	PRP2019	EGGCRATE	4	35	18	6	2.2e-06	0.0243707
	test	EGGCRATE	4	35	18	6	2.2e-06	0.0152842
	FR2	ELATVIDU	2	159	116	35	3.3e-06	0.0709687
	HS2	ELATVIDU	2	105	49	12	0.0	0.053454
	PR2	ELATVIDU	2	99	55	13	2e-07	0.1150556
	PRP2019	ELATVIDU	2	80	41	12	1.6e-06	0.0312381
	test	ELATVIDU	2	79	39	12	4e-07	0.0321513
	FR2	EXP2	2	47	44	10	5.3e-06	0.0289659
	HS2	EXP2	2	248	26	7	0.0004132	0.0342822
	PR2	EXP2	2	50	46	10	5e-07	0.0084656
	PRP2019	EXP2	2	51	49	11	1.5e-06	0.0330286
	test	EXP2	2	51	48	11	0.0	0.0182155
	FR2	EXTROSNB	100	153	108	35	7.2e-06	0.0158846
	HS2	EXTROSNB	100	151	104	34	7.8e-06	0.0079636
	PR2	EXTROSNB	100	147	102	33	6.9e-06	0.0114243
	PRP2019	EXTROSNB	100	158	111	36	6.2e-06	0.0131184
	test	EXTROSNB	100	158	111	36	5.9e-06	0.0083604
	FR2	EXTROSNB	1000	179	128	41	9.6e-06	1.7964449
	HS2	EXTROSNB	1000	163	111	35	9.9e-06	1.5025308
	PR2	EXTROSNB	1000	151	100	32	8.5e-06	1.3634783
	PRP2019	EXTROSNB	1000	154	107	35	9.8e-06	1.4639923
	test	EXTROSNB	1000	158	110	36	9.1e-06	1.4825925
	FR2	FLETGBV2	100	610	607	171	9.4e-06	0.0486854
	HS2	FLETGBV2	100	630	629	177	8.9e-06	0.0457744
	PR2	FLETGBV2	100	796	795	224	8.6e-06	0.0661843
	PRP2019	FLETGBV2	100	608	607	170	9.8e-06	0.0495795
	test	FLETGBV2	100	703	699	151	9.1e-06	0.0644631
	FR2	FLETGBV2	5000	1	1	1	4.4e-06	4.32e-05
	HS2	FLETGBV2	5000	1	1	1	4.4e-06	2.92e-05
	PR2	FLETGBV2	5000	1	1	1	4.4e-06	3.07e-05
	PRP2019	FLETGBV2	5000	1	1	1	4.4e-06	3.04e-05
	test	FLETGBV2	5000	1	1	1	4.4e-06	2.58e-05
	FR2	FLETGBV2	10000	1	1	1	1.6e-06	0.0001427
	HS2	FLETGBV2	10000	1	1	1	1.6e-06	3.98e-05

PR2	FLETGBV2	10000	1	1	1	1.6e-06	5.27e-05
PRP2019	FLETGBV2	10000	1	1	1	1.6e-06	4.89e-05
test	FLETGBV2	10000	1	1	1	1.6e-06	4.3e-05
FR2	FLETCHCR	100	337575	337567	100000	359.4142261	25.9980912
HS2	FLETCHCR	100	2931	2903	739	8.5e-06	0.2323269
PR2	FLETCHCR	100	3392	3352	847	8.6e-06	0.2827112
PRP2019	FLETCHCR	100	4501	4136	995	1e-05	0.3855745
test	FLETCHCR	100	3978	3540	860	7e-06	0.3433066
FR2	FLETCHCR	1000	2859	2853	814	9.7e-06	44.2190662
HS2	FLETCHCR	1000	1047	1026	248	9e-06	14.970519
PR2	FLETCHCR	1000	985	961	234	9.7e-06	13.8689358
PRP2019	FLETCHCR	1000	1021	996	239	9.8e-06	14.8607491
test	FLETCHCR	1000	553	486	128	9.1e-06	6.8850839
FR2	FREUROTH	50	1108	764	202	1.35e-05	0.0844581
HS2	FREUROTH	50	595	304	76	1.81e-05	0.0307041
PR2	FREUROTH	50	382	300	77	7.4e-06	0.025122
PRP2019	FREUROTH	50	211	131	32	3.1e-06	0.0156586
test	FREUROTH	50	181	108	25	9.9e-06	0.0172434
FR2	FREUROTH	100	1002	750	186	8.09e-05	0.0848981
HS2	FREUROTH	100	447	347	88	nan	0.0424612
PR2	FREUROTH	100	577	275	73	7.33e-05	0.0592458
PRP2019	FREUROTH	100	210	133	31	5.5e-06	0.0179329
test	FREUROTH	100	184	112	27	5.2e-06	0.0157583
FR2	FREUROTH	500	533	241	64	0.0004734	0.3737282
HS2	FREUROTH	500	257	180	48	0.0003503	0.2661659
PR2	FREUROTH	500	491	193	55	1.71e-05	0.3595757
PRP2019	FREUROTH	500	1099865	100086	100000	1.61e-05	498.2796728
test	FREUROTH	500	398	116	32	2.93e-05	0.2421796
FR2	FREUROTH	1000	463	186	53	0.0019434	3.6043183
HS2	FREUROTH	1000	195	126	32	5.1e-06	2.0337133
PR2	FREUROTH	1000	407	124	32	7.09e-05	2.2132016
PRP2019	FREUROTH	1000	411	109	30	5.72e-05	1.8286567
test	FREUROTH	1000	389	108	26	1.04e-05	1.7795061
FR2	GULF	3	399	313	65	7.8e-06	0.0642611
HS2	GULF	3	60	45	11	0.8040159	0.0153381
PR2	GULF	3	1575	1468	325	5.3e-06	0.1724588
PRP2019	GULF	3	876	702	103	6.2e-06	0.0812858
test	GULF	3	357	261	42	3.4e-06	0.0361538
FR2	HATFLDD	3	72910	72857	19310	1e-05	2.9282257
HS2	HATFLDD	3	328	270	58	8.7e-06	0.0148878
PR2	HATFLDD	3	147	118	27	8.3e-06	0.0044572
PRP2019	HATFLDD	3	242	202	37	1.8e-06	0.0143463
test	HATFLDD	3	186	157	30	9e-06	0.0061886
FR2	HILBERTA	2	23	23	5	7e-06	0.0006012
HS2	HILBERTA	2	1534	1334	269	9.7e-06	0.1752955
PR2	HILBERTA	2	38	36	9	6.7e-06	0.0029647
PRP2019	HILBERTA	2	43	42	8	0.0	0.0012564
test	HILBERTA	2	43	42	8	0.0	0.0012682
FR2	HILBERTA	4	24	22	4	6.5e-06	0.0008242
HS2	HILBERTA	4	24	22	4	6.5e-06	0.0005988
PR2	HILBERTA	4	24	22	4	6.5e-06	0.0005562
PRP2019	HILBERTA	4	24	22	4	6.5e-06	0.00066
test	HILBERTA	4	24	22	4	6.5e-06	0.0006256
FR2	HILBERTA	5	147	132	27	9.8e-06	0.0078269
HS2	HILBERTA	5	371	353	55	8.5e-06	0.0096309
PR2	HILBERTA	5	94	86	12	3.7e-06	0.0019758
PRP2019	HILBERTA	5	90	84	11	8.3e-06	0.0022259
test	HILBERTA	5	87	81	11	8.3e-06	0.0023655
FR2	HILBERTA	6	167	158	30	9.5e-06	0.0044668
HS2	HILBERTA	6	226	216	38	8.4e-06	0.011933
PR2	HILBERTA	6	382	366	68	6.7e-06	0.0092303
PRP2019	HILBERTA	6	91	87	12	2.9e-06	0.0024925
test	HILBERTA	6	91	87	12	2.9e-06	0.0024551
FR2	HILBERTA	10	100	91	17	8.2e-06	0.0074028
HS2	HILBERTA	10	565	413	72	9.9e-06	0.0381922
PR2	HILBERTA	10	79	74	10	3.1e-06	0.0020437
PRP2019	HILBERTA	10	71	67	9	2.9e-06	0.0032594
test	HILBERTA	10	70	67	9	3.1e-06	0.00361
FR2	HIMMELBG	2	212	201	53	7.7e-06	0.0063615
HS2	HIMMELBG	2	30	25	7	3.1e-06	0.001891
PR2	HIMMELBG	2	32	26	8	4.2e-06	0.0029405
PRP2019	HIMMELBG	2	36	28	8	1e-06	0.0033759
test	HIMMELBG	2	32	25	7	4.7e-06	0.0013764
FR2	INTEQNELS	10	25	19	7	2.4e-06	0.0424522
HS2	INTEQNELS	10	21	16	6	8e-06	0.0009086
PR2	INTEQNELS	10	21	16	6	4.6e-06	0.0008419
PRP2019	INTEQNELS	10	21	16	6	6e-07	0.0013229
test	INTEQNELS	10	21	16	6	6e-07	0.0010576
FR2	INTEQNELS	50	25	19	7	4.8e-06	0.0114184
HS2	INTEQNELS	50	25	19	7	7e-07	0.0160972
PR2	INTEQNELS	50	21	16	6	9.3e-06	0.0125837
PRP2019	INTEQNELS	50	21	16	6	1.1e-06	0.0098357
test	INTEQNELS	50	21	16	6	1.1e-06	0.0081104
FR2	INTEQNELS	100	25	19	7	6.7e-06	0.1132757
HS2	INTEQNELS	100	25	19	7	1e-06	0.1017193
PR2	INTEQNELS	100	25	19	7	7e-07	0.1009748
PRP2019	INTEQNELS	100	21	16	6	1.6e-06	0.0833454
test	INTEQNELS	100	21	16	6	1.6e-06	0.0842683
FR2	INTEQNELS	500	29	22	8	9e-07	25.0205302
HS2	INTEQNELS	500	25	19	7	2.2e-06	19.9083217
PR2	INTEQNELS	500	25	19	7	1.5e-06	19.8794903

PRP2019	INTEQNELS	500	21	16	6	3.5e-06	16.8246655
test	INTEQNELS	500	21	16	6	3.5e-06	17.282033
FR2	LANCZOS1LS	6	2331	2085	506	9.7e-06	0.415359
HS2	LANCZOS1LS	6	1783	1608	360	6.8e-06	0.1946948
PR2	LANCZOS1LS	6	1270	1155	254	8.8e-06	0.1304937
PRP2019	LANCZOS1LS	6	1407	1259	231	8.3e-06	0.4240616
test	LANCZOS1LS	6	1483	1329	231	9.9e-06	0.15976
FR2	LANCZOS2LS	6	1704	1525	365	9.6e-06	0.2550228
HS2	LANCZOS2LS	6	1597	1439	313	7.5e-06	0.1849296
PR2	LANCZOS2LS	6	1019	915	202	4.6e-06	0.0915214
PRP2019	LANCZOS2LS	6	1403	1261	221	9.8e-06	0.0981155
test	LANCZOS2LS	6	1332	1198	217	8.4e-06	0.0869791
FR2	NONDQUAR	500	38555	34845	8363	9.7e-06	71.4877337
HS2	NONDQUAR	500	6454	6161	1371	9.9e-06	10.0845158
PR2	NONDQUAR	500	7795	7255	1511	9.2e-06	10.9377823
PRP2019	NONDQUAR	500	5750	5369	976	9.9e-06	8.2867197
test	NONDQUAR	500	5002	4606	847	9.3e-06	6.7147219
FR2	NONDQUAR	1000	30600	28185	6439	9.8e-06	440.7896907
HS2	NONDQUAR	1000	10035	9501	2039	7.1e-06	116.7928732
PR2	NONDQUAR	1000	7072	6602	1397	8.5e-06	84.5213839
PRP2019	NONDQUAR	1000	5563	5167	974	9.2e-06	69.227517
test	NONDQUAR	1000	5457	5028	941	1e-05	72.3799375
FR2	OSBORNEB	11	40648	40620	11337	9.9e-06	4.1430372
HS2	OSBORNEB	11	1083	981	236	8.4e-06	0.1033202
PR2	OSBORNEB	11	1349	1251	308	9.4e-06	0.129346
PRP2019	OSBORNEB	11	1292	1158	282	5.7e-06	0.1288056
test	OSBORNEB	11	1204	1084	265	9.6e-06	0.1199022
FR2	OSCIPATH	5	337616	337594	100000	12.2812888	13.0803348
HS2	OSCIPATH	5	71902	65188	15126	8.7e-06	2.1542553
PR2	OSCIPATH	5	67036	60311	13973	9e-06	4.6767639
PRP2019	OSCIPATH	5	14673	11861	2432	5.1528661	0.8768092
test	OSCIPATH	5	12080	9772	2020	8.307645	0.5230362
FR2	OSCIPATH	10	56	28	10	3.8e-06	0.0020176
HS2	OSCIPATH	10	56	28	10	3.8e-06	0.002217
PR2	OSCIPATH	10	56	28	10	3.8e-06	0.0016967
PRP2019	OSCIPATH	10	56	28	10	3.8e-06	0.0021732
test	OSCIPATH	10	56	28	10	3.8e-06	0.0018996
FR2	OSCIPATH	25	56	28	10	3.8e-06	0.0020222
HS2	OSCIPATH	25	56	28	10	3.8e-06	0.0024264
PR2	OSCIPATH	25	56	28	10	3.8e-06	0.0021241
PRP2019	OSCIPATH	25	56	28	10	3.8e-06	0.0095553
test	OSCIPATH	25	56	28	10	3.8e-06	0.0028585
FR2	OSCIPATH	100	56	28	10	3.8e-06	0.0195274
HS2	OSCIPATH	100	56	28	10	3.8e-06	0.0148827
PR2	OSCIPATH	100	56	28	10	3.8e-06	0.0026842
PRP2019	OSCIPATH	100	56	28	10	3.8e-06	0.0030365
test	OSCIPATH	100	56	28	10	3.8e-06	0.003634
FR2	OSCIPATH	500	56	28	10	3.8e-06	1.120169
HS2	OSCIPATH	500	56	28	10	3.8e-06	0.0319359
PR2	OSCIPATH	500	56	28	10	3.8e-06	0.0290701
PRP2019	OSCIPATH	500	56	28	10	3.8e-06	0.0271606
test	OSCIPATH	500	56	28	10	3.8e-06	0.027882
FR2	PENALTY1	4	164	149	37	6e-07	0.006101
HS2	PENALTY1	4	4819	3695	612	6e-06	0.3547093
PR2	PENALTY1	4	660	569	102	6e-06	0.0273378
PRP2019	PENALTY1	4	132	114	21	8.7e-06	0.0039519
test	PENALTY1	4	124	108	19	9.6e-06	0.0037319
FR2	PENALTY1	50	149	112	28	1.2e-06	0.0081662
HS2	PENALTY1	50	51048	35310	5134	4.2e-06	3.817031
PR2	PENALTY1	50	272	35	8	59.085833	0.008534
PRP2019	PENALTY1	50	278	231	39	7e-06	0.0146744
test	PENALTY1	50	234	188	34	4.9e-06	0.0135703
FR2	PENALTY1	100	171	124	30	3.6e-06	0.0253833
HS2	PENALTY1	100	25152	17657	2766	1.1e-06	4.4145921
PR2	PENALTY1	100	442	363	63	4.2e-06	0.0564815
PRP2019	PENALTY1	100	716	633	96	5.2e-06	0.0710424
test	PENALTY1	100	658	592	90	9.6e-06	0.0700741
FR2	PENALTY1	500	299998	100000	100000	1080676791431.8745	803.0422619
HS2	PENALTY1	500	426	359	70	9.4e-06	1.2232117
PR2	PENALTY1	500	328	227	64	1e-07	0.9673063
PRP2019	PENALTY1	500	405	284	72	5.3e-06	1.027515
test	PENALTY1	500	390	270	70	1e-06	1.0224124
FR2	QING	100	642	633	159	9.3e-06	0.0648765
HS2	QING	100	263	234	64	9.7e-06	0.0218191
PR2	QING	100	263	232	64	9.5e-06	0.0150535
PRP2019	QING	100	265	235	64	7.6e-06	0.0163861
test	QING	100	265	233	64	7.4e-06	0.0168313
FR2	QING	1000	7383	7368	2096	9.9e-06	103.4594353
HS2	QING	1000	899	878	220	9.6e-06	11.8775307
PR2	QING	1000	899	878	220	9.6e-06	11.7637162
PRP2019	QING	1000	869	846	213	9.6e-06	12.07964
test	QING	1000	878	851	221	9.8e-06	11.403753
FR2	RECIPELS	3	228	211	44	6.2e-06	0.0065963
HS2	RECIPELS	3	82	74	16	7.7e-06	0.0023793
PR2	RECIPELS	3	116	104	19	4e-07	0.003075
PRP2019	RECIPELS	3	145	130	23	7.5e-06	0.0050008
test	RECIPELS	3	125	110	20	2.9e-06	0.0045357
FR2	S308	2	132	104	30	8.5e-06	0.0040507
HS2	S308	2	51	35	10	3e-07	0.0019189
PR2	S308	2	47	35	10	4.3e-06	0.0014138
PRP2019	S308	2	54	38	11	0.0	0.0019746

test	S308	2	50	35	10	5.1e-06	0.0019508
FR2	SISSER	2	22	19	4	6.3e-06	0.0008837
HS2	SISSER	2	48	45	7	9.1e-06	0.0011451
PR2	SISSER	2	53	45	8	2e-07	0.0036611
PRP2019	SISSER	2	22	18	4	1e-06	0.0008611
test	SISSER	2	22	18	4	1e-06	0.0008724
FR2	SNAIL	2	36	27	9	5.4e-06	0.0011067
HS2	SNAIL	2	32	24	8	0.0	0.0009502
PR2	SNAIL	2	32	24	8	0.0	0.0010026
PRP2019	SNAIL	2	20	15	5	3.8e-06	0.0007473
test	SNAIL	2	20	15	5	3.8e-06	0.0008501
FR2	STRTCHDV	100	1214	1167	110	9.9e-06	0.1280772
HS2	STRTCHDV	100	669	640	64	9.6e-06	0.0700662
PR2	STRTCHDV	100	732	701	70	6.3e-06	0.0709596
PRP2019	STRTCHDV	100	753	723	73	6.9e-06	0.074916
test	STRTCHDV	100	697	670	68	5.9e-06	0.0693776
FR2	STRTCHDV	1000	1258	1210	115	9.9e-06	17.4961621
HS2	STRTCHDV	1000	690	656	65	9.5e-06	8.6515558
PR2	STRTCHDV	1000	582	553	58	0.0001099	8.4166515
PRP2019	STRTCHDV	1000	748	721	71	9.6e-06	10.5016674
test	STRTCHDV	1000	682	649	66	9.8e-06	9.1874532
FR2	TRIGON1	100	4295	4277	1141	8.7e-06	20.7484231
HS2	TRIGON1	100	2170	2153	571	9.7e-06	10.3470248
PR2	TRIGON1	100	3180	3164	841	9.9e-06	15.5912666
PRP2019	TRIGON1	100	2602	2585	700	9.6e-06	12.9070931
test	TRIGON1	100	2338	2318	612	8.7e-06	13.0226672
FR2	WATSON	31	11471	9699	2076	6.9e-06	2.0236314
HS2	WATSON	31	35721	31586	7171	9.3e-06	7.2062064
PR2	WATSON	31	31392	27111	5921	8.5e-06	4.8445817
PRP2019	WATSON	31	12326	9660	1840	9.5e-06	2.0962976
test	WATSON	31	11815	9224	1742	8.9e-06	2.3495793

Рисунок Б.1 – Технический отчет

Таблица Б.2 – Отчет о количестве нерешенных задач оптимизации

Название метода оптимизации	Количество нерешенных задач оптимизации
1	2
Двухкомпонентный метод Флетчера-Ривза	7
Двухкомпонентный метод Гестенса-Штифеля	4
Двухкомпонентный метод Поляка-Рибьери	5
Трехкомпонентный гибридный метод из работы [21]	1
Разработанный трёхкомпонентный метод оптимизации	2

ПРИЛОЖЕНИЕ В

Фрагменты кода разработанного программного обеспечения

Файл problem_list.txt

```
ARGTRIGLS;10
ARGTRIGLS;50
ARGTRIGLS;100
ARGTRIGLS;200
ARWHEAD;100
ARWHEAD;500
ARWHEAD;1000
ARWHEAD;5000
BA-L1LS;
BDQRTIC;100
BDQRTIC;500
BDQRTIC;1000
BOXPOWER;10
BOXPOWER;100
BROYDN3DLS;10
BROYDN3DLS;100
BROYDN3DLS;500
BROYDN3DLS;1000
BROYDN3DLS;5000
BROYDN3DLS;10000
BROWNAL;10
BROYDN7D;25
BROYDN7D;50
BROYDN7D;250
BROYDN7D;500
BROYDN7D;2500
BROYDNBDLS;100
BROYDNBDLS;500
BROYDNBDLS;1000
BROYDNBDLS;5000
BROYDNBDLS;10000
BRYBND;100
BRYBND;500
BRYBND;1000
BRYBND;5000
BRYBND;10000
CHNROSNB;50
CHNRSNBM;10
COSINE;10
COSINE;100
COSINE;1000
COSINE;10000
DIXON3DQ;100
DIXON3DQ;1000
ECKERLE4LS;
EGGCRATE;
ELATVIDU;
EXP2;
EXTROSNB;100
EXTROSNB;1000
FLETGBV2;100
FLETGBV2;5000
FLETGBV2;10000
FLETCHCR;100
FLETCHCR;1000
```

```

FREUROTH;50
FREUROTH;100
FREUROTH;500
FREUROTH;1000
GULF;
HATFLDD;
HILBERTA;2
HILBERTA;4
HILBERTA;5
HILBERTA;6
HILBERTA;10
HIMMELBG;
INTEQNELS;10
INTEQNELS;50
INTEQNELS;100
INTEQNELS;500
LANCZOS1LS;
LANCZOS2LS;
NONDQUAR;500
NONDQUAR;1000
OSBORNEB;
OSCIPATH;5
OSCIPATH;10
OSCIPATH;25
OSCIPATH;100
OSCIPATH;500
PENALTY1;4
PENALTY1;50
PENALTY1;100
PENALTY1;500
QING;100
QING;1000
RECIPELS;
S308;
SISSER;
SNAIL;
STRTCHDV;100
STRTCHDV;1000
TRIGON1;100
WATSON;31

```

Файл cute_methods.py

```

maxiter = 100000
prec = 1e-5

```

```

def problem(name, params):
    """
    Import test functions from the CUTEst library.

    :param name: CUTEst problem name
    :param params: SIF file parameters to use

    :return: a reference to the Python interface class for this problem (class
    pycutest.CUTEstProblem)
    """
    if ('ROSENBR' in name) or ('MODBEALE' in name) or ('BROYDN7D' in name):
        return cute.import_problem(name, sifParams={'N/2': int(params)})
    elif (name == 'FMINSURF') or (name == 'LMINSURF') or (name == 'NLMSURF') or
    (name == 'MSQRTA'):
        return cute.import_problem(name, sifParams={'P': int(params)})

```

```

elif name == 'WOODS':
    return cute.import_problem(name, sifParams={'NS': int(params)})
elif params:
    return cute.import_problem(name, sifParams={'N': int(params)})
else:
    return cute.import_problem(name)

def FR2(function, params):
    p = problem(function, params)
    x0 = p.x0
    f0, g0 = p.obj(x0, gradient=True)
    feva, geva, it = 1, 1, 1
    gk, pk, xk_old, xk = g0, -g0, x0, x0
    start_time = process_time()

    while it < maxiter and norm(gk) > prec:
        # alpha
        fk_old = p.obj(xk_old)
        feva += 1
        alpha, fc, gc = line_search(p, xk, pk, fk_old)
        if alpha is None:
            result = linesearch.line_search_armijo(p.obj, xk, pk, gk, fk_old)
            alpha = result[0]
            feva += result[1]
        else:
            feva += fc
            geva += gc
            if alpha is None:
                break

        # step
        xk_old = xk
        xk = xk + alpha * pk
        gk_old = gk
        gk = p.gradhess(xk)[0]
        geva += 1

        # beta
        beta = norm(gk) ** 2 / norm(gk_old) ** 2

        # direction
        pk = -gk + beta * pk
        it = it + 1

    if params:
        return function, params, feva, geva, it, round(norm(gk), 7),
        round(process_time() - start_time, 7)
    else:
        return function, cute.problem_properties(function)['n'], feva, geva, it,
        round(norm(gk), 7), \
        round(process_time() - start_time, 7)

def PRP2019(function, params):
    p = problem(function, params)
    x0 = p.x0
    f0, g0 = p.obj(x0, gradient=True)
    feva, geva, it = 1, 1, 1
    gk, pk, xk_old, xk = g0, -g0, x0, x0
    start_time = process_time()

    while it < maxiter and norm(gk) > prec:
        # alpha

```

```

fk_old = p.obj(xk_old)
feva += 1
alpha, fc, gc = line_search(p, xk, pk, fk_old)
if alpha is None:
    result = linesearch.line_search_armijo(p.obj, xk, pk, gk, fk_old)
    alpha = result[0]
    feva += result[1]
else:
    feva += fc
    geva += gc
if alpha is None:
    break

# step
xk_old = xk
xk = xk + alpha * pk
gk_old = gk
gk = p.gradhess(xk)[0]
geva += 1

# beta
yk = gk - gk_old
if (norm(gk)) ** 2 > abs(dot(gk.T, gk_old)):
    beta = ((norm(gk)) ** 2 - dot(gk.T, gk_old)) / (norm(gk_old)) ** 2
else:
    numNPRP = (norm(gk)) ** 2 - (norm(gk) / norm(gk_old)) *
abs(dot(gk.T, gk_old))
    betaNPRP = numNPRP / (norm(gk_old)) ** 2
    betaFR = (norm(gk)) ** 2 / (norm(gk_old)) ** 2
    ro = ((norm(gk) / norm(gk_old)) * abs(dot(gk.T, gk_old))) /
(norm(gk_old)) ** 2
    G = dot(yk.T, pk) - dot(yk.T, gk) * (dot(pk.T, gk) / (norm(gk)) **
2)
    tetta = (dot(yk.T, gk) - betaNPRP * G) / dot(G, ro)
    if tetta > 1:
        tetta = 1
    elif tetta < 0:
        tetta = 0
    beta = (1 - tetta) * betaNPRP + tetta * betaFR

# direction
if abs(dot(gk.T, gk_old)) >= 0.2 * (norm(gk))**2:
    pk = -gk
else:
    ratio = dot(pk.T, gk) / (norm(gk))**2
    pk = -gk + beta * pk - beta * ratio * gk
it += 1

if params:
    return function, params, feva, geva, it, round(norm(gk), 7),
round(process_time() - start_time, 7)
else:
    return function, cute.problem_properties(function)['n'], feva, geva, it,
round(norm(gk), 7), \
round(process_time() - start_time, 7)

def test(function, params):
    p = problem(function, params)
    x0 = p.x0
    f0, g0 = p.obj(x0, gradient=True)
    feva, geva, it = 1, 1, 1
    gk, pk, xk_old, xk = g0, -g0, x0, x0

```

```

start_time = process_time()

while it < maxiter and norm(gk) > prec:
    # alpha
    fk_old = p.obj(xk_old)
    feva += 1
    alpha, fc, gc = line_search(p, xk, pk, fk_old)
    if alpha is None:
        result = linesearch.line_search_armijo(p.obj, xk, pk, gk, fk_old)
        alpha = result[0]
        feva += result[1]
    else:
        feva += fc
        geva += gc
    if alpha is None:
        break

    # step
    xk_old = xk
    xk = xk + alpha * pk
    gk_old = gk
    gk = p.gradhess(xk)[0]
    geva += 1

    # beta
    yk = gk - gk_old
    if (norm(gk)) ** 2 > abs(dot(gk.T, gk_old)):
        beta = dot(gk.T, yk) / norm(gk_old)**2
    else:
        numNPRP = (norm(gk)) ** 2 - (norm(gk) / norm(gk_old)) *
abs(dot(gk.T, gk_old))
        betaNPRP = numNPRP / (norm(gk_old)) ** 2
        betaFR = (norm(gk)) ** 2 / (norm(gk_old)) ** 2
        ro = ((norm(gk) / norm(gk_old)) * abs(dot(gk.T, gk_old))) /
(norm(gk_old)) ** 2
        G = dot(yk.T, pk) - dot(yk.T, gk) * (dot(pk.T, gk) / (norm(gk)) **
2)
        tetta = (dot(yk.T, gk) - betaNPRP * G) / dot(G, ro)
        if tetta > 1:
            tetta = 1
        elif tetta < 0:
            tetta = 0
        beta = (1 - tetta) * betaNPRP + tetta * betaFR

    if beta < 0:
        beta = 0

    # direction
    omega = dot(pk.T, yk) / norm(gk_old)**2

    if abs(dot(gk.T, gk_old)) >= 0.2 * (norm(gk))**2:
        pk = -gk
    else:
        ratio = dot(pk.T, gk) / norm(gk)**2
        pk = -omega * gk + beta * pk - omega * beta * ratio * gk

    it = it + 1

if params:
    return function, params, feva, geva, it, round(norm(gk), 7),
round(process_time() - start_time, 7)
else:

```

```

        return function, cute.problem_properties(function)['n'], feva, geva, it,
        round(norm(gk), 7), \
            round(process_time() - start_time, 7)

def line_search(p, xk, pk, prev_f, extra_condition=None):
    extra_condition2 = None
    fc, gc = [0], [0]
    gfk, gfk_alpha = [None], [None]

    def derivative_evaluation(alpha):
        gfk[0] = p.gradhess(xk + alpha * pk)[0]
        gc[0] += 1
        gfk_alpha[0] = alpha
        return np.dot(gfk[0], pk)

    def f_evaluation(alpha):
        fc[0] += 1
        return p.obj(xk + alpha * pk)

    alpha_star = scalar_search(f_evaluation, derivative_evaluation, prev_f,
        extra_condition2)
    return alpha_star, fc[0], gc[0]

def scalar_search(f_evaluation, derivative_evaluation, prev_f,
    extra_condition=None):
    c1, c2 = 1e-4, 0.1
    f_0 = f_evaluation(0.)
    derivative0 = derivative_evaluation(0.)

    alpha0 = 0
    alpha1 = min(1, 1.01 * 2 * (f_0 - prev_f) / derivative0)
    if alpha1 == 0:
        alpha1 = 1.0
    f_a1 = f_evaluation(alpha1)
    f_a0 = f_0
    derivative_a0 = derivative0
    extra_condition = lambda alpha, f: True

    for i in range(10):
        if (f_a1 > f_0 + c1 * alpha1 * derivative0) or \
            ((f_a1 >= f_a0) and (i > 1)):
            alpha_star = \
                _zoom(alpha0, alpha1, f_a0,
                    f_a1, derivative_a0, f_evaluation, derivative_evaluation,
                    f_0, derivative0, c1, c2, extra_condition)
            break
        derivative_a1 = derivative_evaluation(alpha1)

        if abs(derivative_a1) <= -c2 * derivative0:
            alpha_star = alpha1
            break

        if derivative_a1 >= 0:
            alpha_star = \
                _zoom(alpha1, alpha0, f_a1,
                    f_a0, derivative_a1, f_evaluation, derivative_evaluation,
                    f_0, derivative0, c1, c2, extra_condition)
            break

    alpha2 = 2 * alpha1
    alpha0 = alpha1

```

```

        alpha1 = alpha2
        f_a0 = f_a1
        f_a1 = f_evaluation(alpha1)
        derivative_a0 = derivative_a1

    else:
        alpha_star = alpha1
        # warn('The line search algorithm did not converge', LineSearchWarning)

    return alpha_star

def cubicmin(a, fa, fpa, b, fb, c, fc):
    with np.errstate(divide='raise', over='raise', invalid='raise'):
        try:
            C = fpa
            db = b - a
            dc = c - a
            denom = (db * dc) ** 2 * (db - dc)
            d1 = np.empty((2, 2))
            d1[0, 0] = dc ** 2
            d1[0, 1] = -db ** 2
            d1[1, 0] = -dc ** 3
            d1[1, 1] = db ** 3
            [A, B] = np.dot(d1, np.asarray([fb - fa - C * db,
                                            fc - fa - C * dc]).flatten())

            A /= denom
            B /= denom
            radical = B * B - 3 * A * C
            xmin = a + (-B + np.sqrt(radical)) / (3 * A)
        except ArithmeticError:
            return None
    if not np.isfinite(xmin):
        return None
    return xmin

def quadmin(a, fa, fpa, b, fb):
    with np.errstate(divide='raise', over='raise', invalid='raise'):
        try:
            d = fa
            c = fpa
            db = b - a
            b = (fb - d - c * db) / (db * db)
            x_min = a - c / (2.0 * b)
        except ArithmeticError:
            return None
    if not np.isfinite(x_min):
        return None
    return x_min

def _zoom(a_lo, a_hi, f_lo, f_hi, derivative_lo,
          f_evaluation, derivative_evaluation, f_0, derivative0, c1, c2,
          extra_condition):
    i = 0
    delta1 = 0.2 # cubic
    delta2 = 0.1 # quadratic
    f_rec = f_0
    a_rec = 0
    while True:
        dalpha = a_hi - a_lo
        if dalpha < 0:

```



```

        a, b = a_hi, a_lo
    else:
        a, b = a_lo, a_hi

    cchk = delta1 * dalpha
    a_j = cubicmin(a_lo, f_lo, derivative_lo, a_hi, f_hi, a_rec, f_rec)
    if (i == 0) or (a_j is None) or (a_j > b - cchk) or (a_j < a + cchk):
        qchk = delta2 * dalpha
        a_j = quadmin(a_lo, f_lo, derivative_lo, a_hi, f_hi)
        if (a_j is None) or (a_j > b - qchk) or (a_j < a + qchk):
            a_j = a_lo + 0.5 * dalpha

    f_aj = f_evaluation(a_j)
    if (f_aj > f_0 + c1 * a_j * derivative0) or (f_aj >= f_lo):
        f_rec = f_hi
        a_rec = a_hi
        a_hi = a_j
        f_hi = f_aj
    else:
        derivative_aj = derivative_evaluation(a_j)
        if abs(derivative_aj) <= -c2 * derivative0 and extra_condition(a_j,
f_aj):
            a_star = a_j
            break
        if derivative_aj * (a_hi - a_lo) >= 0:
            f_rec = f_hi
            a_rec = a_hi
            a_hi = a_lo
            f_hi = f_lo
        else:
            f_rec = f_lo
            a_rec = a_lo
            a_lo = a_j
            f_lo = f_aj
            derivative_lo = derivative_aj
    i += 1
    if i > 20:
        a_star = None
        break
    return a_star

```

Файл cute_main.py

```

problem_list = []
with open('problem_list.txt') as file:
    for line in file:
        problem_list.append(line.rstrip().split(';'))
problems = len(problem_list)

solvers = {'FR2': cute_methods.FR2, 'HS2': cute_methods.HS2, 'PR2':
cute_methods.PR2, 'PRP2019': cute_methods.PR2019,
'test': cute_methods.test}
methods = len(solvers)
method_names = np.array(list(solvers))

feva = np.zeros((problems, methods))
geva = np.zeros((problems, methods))
iterations = np.zeros((problems, methods))
solution_time = np.zeros((problems, methods))
error = np.zeros((problems, methods))

```

```

report = PrettyTable()
report.field_names = ["method", "function", "dimension", "feva", "geva",
"iterations", "norm(gk)", "time"]

def save_results(method, problem, function, dim, f, g, it, gk, time):
    report.add_row([method_names[method], function, dim, f, g, it, gk, time])
    if gk > 1e-4 or gk is None:
        error[problem][method] = 1
    feva[problem][method] = f
    geva[problem][method] = g
    iterations[problem][method] = it
    solution_time[problem][method] = time

for problem in range(problems):
    function = problem_list[problem][0]
    params = problem_list[problem][1]

    for method in range(methods):
        function, dim, f, g, it, gk, time =
solvers[method_names[method]](function, params)
        save_results(method, problem, function, dim, f, g, it, gk, time)
    print(problem+1)

for problem in range(problems):
    for method in range(methods):
        if error[problem][method]:
            feva[problem][method] = np.max(feva)
            geva[problem][method] = np.max(geva)
            iterations[problem][method] = np.max(iterations)
            solution_time[problem][method] = np.max(solution_time)

print(report)

errors = PrettyTable()
errors.field_names = method_names
errors.add_row([sum(error[:, 0]), sum(error[:, 1]), sum(error[:, 2]),
sum(error[:, 3]), sum(error[:, 4])])
print(errors)

for problem in range(problems):
    fmin = np.min(feva[problem, :])
    gmin = np.min(geva[problem, :])
    itmin = np.min(iterations[problem, :])
    timemin = np.min(solution_time[problem, :])
    for method in range(methods):
        feva[problem][method] /= fmin
        geva[problem][method] /= gmin
        iterations[problem][method] /= itmin
        solution_time[problem][method] /= timemin

def perfomance(t, method, metric):
    ps = np.zeros((len(t)))
    k = 0
    for tau in t:
        count = 0
        for r in metric[:, method]:
            if r <= tau:
                count += 1
        ps[k] = count / problems
        k += 1

```

```

    return ps

def performance_log2(t, method, metric):
    ps = np.zeros((len(t)))
    k = 0
    for tau in t:
        count = 0
        for r in metric[:, method]:
            if np.log2(r) <= tau:
                count += 1
        ps[k] = count / problems
        k += 1
    return ps

#####
##### [0; 2] #####
#####
plt.figure('close to 1')
t = np.linspace(0, 2, 100000)
line_styles = ['-','--','-','--','-','--']
line_colors = ['b','g','r','y','m','k']

plt.subplot(2, 2, 1)
for method in range(methods):
    plt.plot(t, performance(t, method, feva), linestyle=line_styles[method],
             color=line_colors[method],
             label=method_names[method])
plt.title("function evaluations")
plt.xlabel("t")
plt.ylabel("P(t)")
plt.grid(True)

plt.subplot(2, 2, 2)
for method in range(methods):
    plt.plot(t, performance(t, method, geva), linestyle=line_styles[method],
             color=line_colors[method],
             label=method_names[method])
plt.title("gradient evaluations")
plt.xlabel("t")
plt.ylabel("P(t)")
plt.grid(True)

plt.subplot(2, 2, 3)
for method in range(methods):
    plt.plot(t, performance(t, method, iterations),
             linestyle=line_styles[method], color=line_colors[method],
             label=method_names[method])
plt.title("iterations")
plt.xlabel("t")
plt.ylabel("P(t)")
plt.grid(True)

plt.subplot(2, 2, 4)
for method in range(methods):
    plt.plot(t, performance(t, method, solution_time),
             linestyle=line_styles[method], color=line_colors[method],
             label=method_names[method])
plt.title("time")
plt.xlabel("t")
plt.ylabel("P(t)")
plt.grid(True)
plt.legend()

```

```
plt.show()
```

Файл quad_methods.py

```
maxiter = 100000  
prec = 1e-5
```

```
def feva(x, A, b, l):  
    return (norm(dot(A, x) - b)) ** 2 + l * ((norm(x)) ** 2)
```

```
def geva(x, A, b, l):  
    return 2 * dot(A.T, (dot(A, x) - b)) + 2 * l * x
```

```
def aeva(xk, pk, A, b, l):  
    axb = dot(A, xk) - b  
    ap = dot(A, pk)  
    num = -dot(axb.T, ap) - dot(l, dot(xk.T, pk))  
    den = (norm(ap)) ** 2 + l * (norm(pk)) ** 2  
    return num / den
```

```
def FR2(A, b, l, n):  
    x0 = np.zeros((n, 1))  
    it = 1  
    gk = geva(x0, A, b, l)  
    pk, xk = -gk, x0  
    start_time = process_time()  
  
    while it < maxiter and norm(gk) > prec:  
        alpha = aeva(xk, pk, A, b, l)  
        xk = xk + alpha * pk  
        gk_old = gk  
        gk = geva(xk, A, b, l)  
        beta = norm(gk) ** 2 / norm(gk_old) ** 2  
        pk = -gk + beta * pk  
        it = it + 1  
  
    return xk, it, norm(gk), round(process_time() - start_time, 7)
```

```
def PRP2019(A, b, l, n):  
    x0 = np.zeros((n, 1))  
    it = 1  
    gk = geva(x0, A, b, l)  
    pk, xk_old, xk = -gk, x0, x0  
    start_time = process_time()  
  
    while it < maxiter and norm(gk) > prec:  
        alpha = aeva(xk, pk, A, b, l)  
        xk = xk + alpha * pk  
        gk_old = gk  
        gk = geva(xk, A, b, l)  
        yk = gk - gk_old  
  
        if (norm(gk)) ** 2 > abs(dot(gk.T, gk_old)):  
            beta = dot(gk.T, yk) / norm(gk_old)**2  
        else:
```

```

        numNPRP = (norm(gk)) ** 2 - (norm(gk) / norm(gk_old)) *
abs(dot(gk.T, gk_old))
        betaNPRP = numNPRP / (norm(gk_old)) ** 2
        betaFR = (norm(gk)) ** 2 / (norm(gk_old)) ** 2

        lam = dot(yk.T, gk)
        ro = ((norm(gk) / norm(gk_old)) * abs(dot(gk.T, gk_old))) /
(norm(gk_old)) ** 2
        G = dot(yk.T, pk) - lam * (dot(pk.T, gk) / (norm(gk)) ** 2)
        tetta = (lam - betaNPRP * G) / dot(G, ro)
        if tetta > 1:
            tetta = 1
        elif tetta < 0:
            tetta = 0
        beta = (1 - tetta) * betaNPRP + tetta * betaFR

        if abs(dot(gk.T, gk_old)) >= 0.2 * (norm(gk)) ** 2:
            pk = -gk
        else:
            numerator = dot(pk.T, gk)
            denominator = (norm(gk)) ** 2
            ratio = numerator / denominator
            pk = -gk + beta * pk - beta * ratio * gk

        it = it + 1

    return xk, it, norm(gk), round(process_time() - start_time, 7)

def test(A, b, l, n):
    x0 = np.zeros((n, 1))
    it = 1
    gk = geva(x0, A, b, l)
    pk, xk_old, xk = -gk, x0, x0
    start_time = process_time()

    while it < maxiter and norm(gk) > prec:
        alpha = aevea(xk, pk, A, b, l)
        xk = xk + alpha * pk
        gk_old = gk
        gk = geva(xk, A, b, l)
        yk = gk - gk_old

        if (norm(gk)) ** 2 > abs(dot(gk.T, gk_old)):
            beta = dot(gk.T, yk) / norm(gk_old)**2
        else:
            numNPRP = (norm(gk)) ** 2 - (norm(gk) / norm(gk_old)) *
abs(dot(gk.T, gk_old))
            betaNPRP = numNPRP / (norm(gk_old)) ** 2
            betaFR = (norm(gk)) ** 2 / (norm(gk_old)) ** 2
            ro = ((norm(gk) / norm(gk_old)) * abs(dot(gk.T, gk_old))) /
(norm(gk_old)) ** 2
            G = dot(yk.T, pk) - dot(yk.T, gk) * (dot(pk.T, gk) / (norm(gk)) **
2)
            tetta = (dot(yk.T, gk) - betaNPRP * G) / dot(G, ro)
            if tetta > 1:
                tetta = 1
            elif tetta < 0:
                tetta = 0
            beta = (1 - tetta) * betaNPRP + tetta * betaFR

        if beta < 0:
            beta = 0

```

```

# direction
omega = dot(pk.T, yk) / norm(gk_old) ** 2
if abs(dot(gk.T, gk_old)) >= 0.2 * (norm(gk)) ** 2:
    pk = -gk
else:
    ratio = dot(pk.T, gk) / norm(gk) ** 2
    pk = -omega * gk + beta * pk - omega * beta * ratio * gk
it += 1

return xk, it, norm(gk), round(process_time() - start_time, 7)

```

Файл quad_main.py

```

solvers = {'FR2': quad_methods.FR2, 'HS2': quad_methods.HS2, 'PR2':
quad_methods.PR2, 'PRP2019': quad_methods.PR2019,
'test': quad_methods.test}
methods = len(solvers)
method_names = np.array(list(solvers))
M = np.array([10, 10, 10, 10, 50, 50, 50, 100, 100, 100])
N = np.array([10, 100, 500, 1000, 10, 500, 1000, 10, 500, 1000])
dimensions = len(M)
circles = 1000
tasks = dimensions * circles
iterations = np.zeros((dimensions, circles, methods))
solution_time = np.zeros((dimensions, circles, methods))

report = PrettyTable()
report.field_names = ["method", "M", "N", "iteration", "norm(gk)", "deviation",
"time"]

def save_results(dim, circle, method, m, n, it, gk, xmin, xk, time):
    report.add_row([method_names[method], m, n, it, gk, norm(abs(xmin - xk)),
time])
    iterations[dim][circle][method] = it
    solution_time[dim][circle][method] = time

for dim in range(dimensions):
    m = M[dim]
    n = N[dim]
    for circle in range(circles):
        A = np.zeros((m, n))
        for i in range(m):
            for j in range(n):
                A[i][j] = random.uniform(0, 1)
        l = random.uniform(0, 1)
        y = np.zeros((m, 1))
        for i in range(m):
            y[i][0] = random.uniform(0, 1)
        I = np.eye(m)
        b = dot((dot(A, A.T) + l * I), y)
        xmin = dot(A.T, y)

        for method in range(methods):
            xk, it, gk, time = solvers[method_names[method]](A, b, l, n)
            save_results(dim, circle, method, m, n, it, gk, xmin, xk, time)

print(report)

```

```

for dim in range(dimensions):
    itmin = np.min(iterations[dim, :, :])
    timemin = np.min(solution_time[dim, :, :])
    for method in range(methods):
        for circle in range(circles):
            iterations[dim][circle][method] /= itmin
            solution_time[dim][circle][method] /= timemin

def perfomance(t, method, metric):
    ps = np.zeros((len(t)))
    k = 0
    for tau in t:
        count = 0
        for dim in range(dimensions):
            for circle in range(circles):
                if metric[dim][circle][method] <= tau:
                    count += 1
        ps[k] = count / tasks
        k += 1
    return ps

plt.figure()
t = np.linspace(0, 2, 100000)
line_styles = ['-', '--', '-', '--', '-', '--']
line_colors = ['b', 'g', 'r', 'y', 'm', 'k']

plt.subplot(1, 2, 1)
for method in range(methods):
    plt.plot(t, perfomance(t, method, iterations),
             linestyle=line_styles[method], color=line_colors[method],
             label=method_names[method])
plt.title("iterations")
plt.xlabel("t")
plt.ylabel("P(t)")
plt.grid(True)

plt.subplot(1, 2, 2)
for method in range(methods):
    plt.plot(t, perfomance(t, method, solution_time),
             linestyle=line_styles[method], color=line_colors[method],
             label=method_names[method])
plt.title("time")
plt.xlabel("t")
plt.ylabel("P(t)")
plt.grid(True)
plt.legend()
plt.show()

```