**Федеральное государственное образовательное**

**бюджетное учреждение**

**высшего образования**

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ**

**ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ»**

**(Финансовый университет)**

**Факультет**

**информационных технологий и анализа больших данных**

**Направление «Прикладная математика и информатика»**

**Домашнее задание № 3**

«Методы многомерной оптимизации»

Студенты группы ПМ19-3:

Караваев Артем Евгеньевич

Лазаренко Владлена Владимировна

Минаков Артем Дмитриевич

Пластун Екатерина Сергеевна

Голомысов Даниил Владиславович

Руководитель:

Аксенов Дмитрий Андреевич

**Москва 2022**

Оглавление.

1. Математическая модель (постановка задачи)
2. Алгоритмы
   1. Алгоритм 1
      1. Описание входных данных
      2. Описание алгоритма решения
      3. Описание выходных данных
   2. Алгоритм 2
      1. Описание входных данных
      2. Описание алгоритма решения
      3. Описание выходных данных
   3. Алгоритм 3
      1. Описание входных данных
      2. Описание алгоритма решения
      3. Описание выходных данных
3. Варианты использования системы
   1. ВИ 1
   2. ВИ 2
4. Архитектура решения
   1. Функции считывания информации
   2. Функции обработки информации
   3. Функции вывода информации
5. Тестирование
6. Выводы и заключение
7. **Постановка задачи (физическая модель)**

К нам пришел заказчик с новым проектом по нахождению оптимального способа реализации автомобилией. Фирма реализует автомобили двумя способами: через оптовую и розничную торговлю. При реализации х автомобилей в розницу расходы на реализацию составляют у. е., а при продаже у автомобилей оптом ‒ у. е. Найти оптимальный способ реализации автомобилей, минимизирующий суммарные расходы, если общее число предназначенных для продажи автомобилей составляет 200 шт.

1. **Математическая модель и алгоритмы.**
   1. **Составим математическую модель задачи.**

Функция **L(x,y) =** ‒ суммарные расходы при реализации. По условию требуется найти минимум функции L.

Так как для продажи предназначено 200 автомобилей, то х и у связаны между собой условием связи: х + у = 200, х ≥ 0, у ≥ 0. Таким образом, получили задачу на условный экстремум с «простым» условием связи. Для решения такого типа задач, как известно из курса математического анализа, нужно из условия связи выразить одну переменную через другую, например, у через х: у = 200 − х и подставить полученное выражение в функцию L(х, у).

Тогда последняя превратится в функцию одной переменной.

**L(x) =**

* 1. **Алгоритм 1**

**Поиск экстремума функции одной и многих переменных методом градиентного спуска с постоянным шагом.**

Градиентный\_спуск — [метод](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B) нахождения локального минимума с помощью движения вдоль [градиента](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82). Наиболее простой в реализации из всех методов локальной оптимизации. Имеет довольно слабые условия сходимости, но при этом скорость сходимости достаточно мала (линейна).

* + 1. **Описание входных данных**

**Обязательные параметры:**

а) Минимизируемая функция в аналитическом виде;

б) Функция градиента в аналитическом виде;

в) Константа шага (λ);

**Необязательные параметры:**

г) Максимальное число итераций (по умолчанию: 500);

д) Точность оптимизации по аргументу для критерия Останова (по умолчанию: 10^ (-5));

* + 1. **Описание алгоритма решения**

1. Задаем начальную точку х0. Выбираем рандомно (должна лежать в пределах eps), можно просто задать х0 = (0;0). Если в функции больше двух переменных, например 3, то точка будет состоять из трех координат х0 =(0;0;0).
2. Находим градиент функции. Он состоит из частных производных функции по каждой переменной. . Пример: F(x1, x2) = 2x1^2 – 3x2^2. .
3. Далее будет вычисляться значение градиента функции в каждой рассматриваемой точке. Формула: . Пример: , в точке (0,0). = 0
4. Вычисляем значение градиента для начальной точки.
5. Вычисляем значение функции в начальной точке.
6. Пока > eps или j (колво итераций) < maxk, то:

6.1. Вычисляем новые координаты точки: x(j) = x(j-1) – h\*. – это считается в точке x(j-1).

6.2. Вычисляем значение градиента в точке (x(j)).

6.3. Вычисляем значение функции в точке x(j).

**Ответ:** x(j), F(x(j)).

* + 1. **Описание выходных данных**

Формат выходных данных:

а) Найденное значение координаты точки экстремума;

б) Значение функции в точке экстремума;

г) Отчет о работе алгоритма (например флаг: 0 – найдено значение с заданной точностью; 1 – достигнуто максимальное количество итераций; 2 – выполнено с ошибкой).

* 1. **Алгоритм 2**

**Поиск экстремума функции одной и многих переменных методом градиентного спуска с дроблением шага.**

Градиентный\_спуск — [метод](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B) нахождения локального минимума с помощью движения вдоль [градиента](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82). Наиболее простой в реализации из всех методов локальной оптимизации. Имеет довольно слабые условия сходимости, но при этом скорость сходимости достаточно мала (линейна). Отличается от алгоритма 1 тем, что на каждом шаге мы уменьшаем шаг.

* + 1. **Описание входных данных**

**Обязательные параметры:**

а) Минимизируемая функция в аналитическом виде;

б) Функция градиента в аналитическом виде;

в) Начальный шаг .

г) Параметр дробления sigm: может принимать значения (0;1).

**Необязательные параметры:**

г) Максимальное число итераций (по умолчанию: 500);

д) Точность оптимизации по аргументу для критерия Останова (по умолчанию: 10^ (-5));

* + 1. **Описание алгоритма решения**

1. Задаем начальную точку х0. Выбираем рандомно (должна лежать в пределах eps), можно просто задать х0 = (0;0). Если в функции больше двух переменных, например 3, то точка будет состоять из трех координат х0 =(0;0;0).
2. Находим градиент функции. Он состоит из частных производных функции по каждой переменной. . Пример: F(x1, x2) = 2x1^2 – 3x2^2. .
3. Далее будет вычисляться значение градиента функции в каждой рассматриваемой точке. Формула: . Пример: , в точке (0,0). = 0
4. Вычисляем значение градиента для начальной точки.
5. Вычисляем значение функции в начальной точке.
6. Пока (значение градиента) > eps или j (кол-во итераций) < maxk, то:

6.1. Вычисляем новые координаты точки: x(j) = x(j-1) – h\*. – это считается в точке x(j-1).

6.2. Вычисляем значение функции в точке x(j).

6.3. Если F(x(j)) > F(x(j-1)), то h = h/sigm. Если F(j) < F(j-1), то h остается прежним.

6.4. Вычисляем значение градиента в точке (x(j)).

**Ответ:** x(j), F(x(j)).

* + 1. **Описание выходных данных**

**Формат выходных данных:**

а) Найденное значение координаты точки экстремума;

б) Значение функции в точке экстремума;

г) Отчет о работе алгоритма (например флаг: 0 – найдено значение с заданной точностью; 1 – достигнуто максимальное количество итераций; 2 – выполнено с ошибкой).

* 1. **Алгоритм 3**

**Поиск экстремума функции многих наискорейшего градиентного спуска.**

Градиентный\_спуск — [метод](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B) нахождения локального минимума с помощью движения вдоль [градиента](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82).

Этот вариант градиентного метода основывается на выборе нового шага в каждой последующей итерации. Метод наискорейшего спуска требует решения на каждом шаге задачи одномерной оптимизации. Практика показывает, что этот метод часто требует меньшего числа операций, чем градиентный метод с постоянным шагом.

* + 1. **Описание входных данных**

**Обязательные параметры:**

а) Минимизируемая функция в аналитическом виде;

б) Функция градиента в аналитическом виде;

**Необязательные параметры:**

в) Максимальное число итераций (по умолчанию: 500);

г) Точность оптимизации по аргументу для критерия Останова (по умолчанию: 10^ (-5));

д) Флаг «вывод промежуточных результатов»

е) Флаг «запись промежуточных результатов в датасет»

* + 1. **Описание алгоритма решения**

1. Находим градиент функции. Он состоит из частных производных функции по каждой переменной. . Пример: F(x1, x2) = 2x1^2 – 3x2^2. .
2. Далее будет вычисляться значение градиента функции в каждой рассматриваемой точке. Формула: . Пример: , в точке (0,0). = 0
3. Вычисляем значение градиента для начальной точки (например, Х(0;0) для 2 переменных).
4. Вычисляем значение функции в начальной точке.
5. Находим новый шаг(λ) и новый Х
6. Проверяем условие остановки:

Пока (значение градиента) > eps или k (колво итераций) < maxk, то:

1. Находим следующую точку по ф-ле:
   * 1. **Описание выходных данных**

**Формат выходных данных:**

а) Найденное значение координаты точки экстремума;

б) Значение функции в точке экстремума;

г) Отчет о работе алгоритма (например флаг: 0 – найдено значение с заданной точностью; 1 – достигнуто максимальное количество итераций; 2 – выполнено с ошибкой).

* 1. **Алгоритм 4**

**Поиск экстремума функции многих переменной при помощи алгоритма Ньютона-сопряженного градиента.**

Метод сопряжённых градиентов — итерационный метод для безусловной оптимизации в многомерном пространстве.

* + 1. **Описание входных данных**

**Обязательные входные параметры:**

а) На вход подается функция многих переменных в аналитическом виде.

* + 1. **Описание алгоритма решения**

1. Вычисляется градиента функции .
2. Далее будет вычисляться значение градиента функции в каждой рассматриваемой точке. Формула: . Вычисляем значение градиента для начальной точки (например, Х(0;0) для 2 переменных).

**3)** 0 итерация полностью как в методе наискорейшего спуска:

* + - * Проверяем условие Останова
      * Вычисляем значение ф-ции в начальной точке
      * Находим следующую точку по ф-ле:

*Далее выполняется итерационный процесс.*

**4)** Вычисляем значение градиента в новой точке и т.д (как и раннее)

**НО** меняется нахождение следующей точки с 1 итерации:

Изображение выглядит как текст

Автоматически созданное описание

**5)** Подставляем новый X в начальную ф-цию, находим новый шаг t0.

**6)** Определяем новые значения аргументов функции после выполнения k-го шага расчета

**7)** проверяем критерии останова итерационного процесса. Вычислительный процесс заканчивается, **когда будет достигнута точка, в которой оценка градиента будет равна нулю:**

В противном случае возвращаемся к шагу 4 и продолжаем итерационный расчет.

* + 1. **Описание выходных данных**

а) Найденное значение координаты точки экстремума;

б) Значение функции в точке экстремума;

1. **Варианты использования системы**

Предполагается, что при нажатии кнопки «старт» Пользователю поочередно будут высвечиваться поля. В самом первом поле он введет используемые переменные. Во втором – функцию в аналитическом виде. Во всех остальных полях по требованию программы пользователю будут предлагаться вводиться необязательные параметры.

В результате запуска программы Пользователь на выходе получит точку – оптимальный план выпуска/продажи, а также значение функции в этой точке – значение прибыли/затрат.

* 1. **ВИ 1 и ВИ 2** 
     1. **Ввод названия переменных и функции в аналитическом виде.**
        1. Для ввода данных для решения Вам нужно выполнить следующее:



* В параметре variables после равно в квадратных скобках через запятую (как показано на рисунке) написать названия переменных, используемых для вычисления.
* присвоить f функцию в кавычках в аналитическом виде.

\*В качестве знака умножения используем \*, деления /, возведения в степень \*\*.

На выводе получим функцию, которая нам понадобится для следующих шагов:



* + - 1. По аналогии заполняем данные для следующей функции gradFunc(), которая находит градиент функции:

****

Отличие от пункта 3.1.1.1. состоит в том, что переменные вводятся по раздельности после ключевого слова «vari», i = 1,2,3… и соответствует количеству переменных в функции.

На выводе получим список, состоящий из градиента функции:



* + - 1. Далее в функциях f(x) и g(x) вводим результаты, полученные в пункте 3.1.1.2. В f(x) вводим результат, полученный в результате работы функции func (копируем и вставляем без кавычек). В g(x) внутри круглых скобок пишем результат, полученный в результате работы функции gradFunc (копируем и вставляем без кавычек).

**Изображение выглядит как текст

Автоматически созданное описание**

* + - 1. **Далее просто вызываем нужную функцию.**

**gradient\_const\_step() -** в случае применения метода градиентного спуска с постоянным шагом.

**gradient\_changing\_step() -** в случае применения метода градиентного спуска с дроблением шага.

* + 1. **Вывод полученных результатов.**

**Изображение выглядит как текст

Автоматически созданное описание**

**Первая строка** выводит время выполнения работы алгоритма в секундах.

**Вторая** строка в круглых скобках выводит три значения. Первое значение соответствует найденное точке минимума. Второе – значению функции в найденной точке. Третье – результат работы алгоритма (0 – найдено значение с заданной точностью; 1 – достигнуто максимальное количество итераций; 2 – выполнено с ошибкой).

* 1. **ВИ 3**
     1. **Ввод названия переменных и функции в аналитическом виде.**

Для ввода данных для решения Вам нужно выполнить следующее:

* Ввести любые названия переменных на английском языке через пробел, затем нажмите клавишу Enter.

Пример:



* Ввести функцию в аналитическом виде, используя обозначения переменных выше.

В качестве знака умножения используем \*, деления /, возведения в степень \*\*.

Пример:



* + 1. **Вывод полученных результатов.**
       1. **Первая строка** выводит найденное значение координаты экстремума.
       2. **Вторая** **строка** выводит значение функции в найденной точке экстремума.
       3. **Третья строка** выводит время выполнения работы алгоритма в секундах.
  1. **ВИ 4**
     1. **Ввод названия переменных и функции в аналитическом виде.**

Для ввода данных для решения Вам нужно выполнить следующее:

* Ввести любые названия переменных на английском языке через пробел, затем нажмите клавишу Enter.

Пример:



* Ввести функцию в аналитическом виде, используя обозначения переменных выше, затем нажмите клавишу Enter.

В качестве знака умножения используем \*, деления /, возведения в степень \*\*.

Пример:



* + 1. **Вывод полученных результатов.**

**Первая строка** выводит найденное значение координаты экстремума.

**Вторая** **строка** выводит значение функции в найденной точке экстремума.

**Третья строка** выводит время выполнения работы алгоритма в секундах.

**Изображение выглядит как текст

Автоматически созданное описание**

1. **Архитектура решения**
   1. **Метод градиентного спуска с постоянным шагом и с дроблением шага.**
      1. **Функции считывания информации**

**def func()**

"""

Функции для перевода заданной функции из аналитического вида в нужный для дальнейших вычислений.

Parameters

-----------

variables: list

Используемые переменные.

f: str

Функция в аналитическом виде

Returns

-----------

Заданная функция для дальнейших вычислений.

"""

**def gradFunc()**

"""

Функция для нахождения градиента заданной функции.

Parameters

-----------

var1, var2: str

Используемые переменные.

f: str

Функция в аналитическом виде

Returns

-----------

Список строк из значений производной по каждой переменной.

"""

**def f(x)**

"""

Функция, возвращающая введенную функцию для дальнейших вычислений.

"""

**def g(x)**

"""

Функция, возвращающая массив найденного градиента функции.

"""

* + 1. **Функции обработки и вывода информации**

**def gradient\_const\_step()**

"""

Функция для нахождения координаты точки минимума и значения функции в этой точке методом градиентного спуска с постоянным шагом.

Parameters

-----------

func: str

Функция в нужном виде.

grad: np.array

Градиент функции.

nvar: int

Количество используемых переменных.

h: float

Шаг для градиентного спуска.

maxk: int

Максимальное количество итераций для нахождения ответа.

eps : float

Заданная точность алгоритма.

xk: array

Массив из нулей, где количество нулей равняется количеству введенных переменных.

k: int

Количество итераций в цикле.

xk: float

Текущее значение найденного х.

fk: int

Значение функции в текущей точке.

Returns

-----------

xk: float

Список координат точки минимума.

fk: float

Найденное значение функции в точке минимума.

0/1/2

Отчет о работе алгоритма

"""

**def gradient\_changing\_step()**

"""

Функция для нахождения координаты точки минимума и значения функции в этой точке методом градиентного спуска с дроблением шагом.

Parameters

-----------

func: str

Функция в нужном виде.

grad: np.array

Градиент функции.

nvar: int

Количество используемых переменных.

sigm: str

Параметр дробления шага.

h: float

Шаг для градиентного спуска.

maxk: int

Максимальное количество итераций для нахождения ответа.

eps: float

Заданная точность алгоритма.

xk: array

Массив из нулей, где количество нулей равняется количеству введенных переменных.

xj: array

Массив со значением переменной на предыдущей итерации.

k: int

Количество итераций в цикле.

xk: float

Текущее значение найденного х.

fk: int

Значение функции в текущей точке.

Returns

-----------

xk: float

Список координат точки минимума.

fk: float

Найденное значение функции в точке минимума.

0/1/2

Отчет о работе алгоритма

"""

* 1. **Метод наискорейшего градиентного спуска.**
     1. **Функции считывания информации**

"""

Функция для ввода данных, с помощью которого будет осуществляться алгоритм Ньютон-сопряженного градиента

Parameters

----------

params: str

Строка на вход с переменными, которые могуи быть заданы любой русской или английской буквой

F: str

Функция в аналитическом виде в формате строки

Returns

-----------

Final: dictionary

Словарь со всеми переменными и функциями, которые ранее были введены

"""

* + 1. **Функции обработки и вывода информации**

"""

Функция, которая осуществляет алгоритм алгоритм Ньютон-сопряженного градиента

Parameters

----------

dictionary: dict

Словарь, со всеми переменными, которые были заданы в функции ввода

eps: float

критерий остановки

func: str

Функция в аналитическом виде в формате строки

Returns

---------

X: list

Найденная точка минимума

fun: float

Значение функции в точке экстремума

(time.time() - start\_time): float

Время выполнения в секундах

"""

* 1. **Алгоритм Ньютона-сопряженного градиента.**
     1. **Функции считывания информации**

**def inputSopr():**

"""

Функция для ввода данных, с помощью которого будет осуществляться алгоритм Ньютон-сопряженного градиента

Parameters

----------

params: str

Строка на вход с переменными, которые могуи быть заданы любой русской или английской буквой

F: str

Функция в аналитическом виде в формате строки

Returns

-----------

Final: dictionary

Словарь со всеми переменными и функциями, которые ранее были введены

"""

* + 1. **Функции обработки и вывода информации**

**def sopr(dictionary):**

"""

Функция, которая осуществляет алгоритм алгоритм Ньютон-сопряженного градиента

Parameters

----------

dictionary: dict

Словарь, со всеми переменными, которые были заданы в функции ввода

eps: float

критерий остановки

func: str

Функция в аналитическом виде в формате строки

Returns

---------

X: list

Найденная точка минимума

fun: float

Значение функции в точке экстремума

(time.time() - start\_time): float

Время выполнения в секундах

"""

1. **Тестирование**

Таблица 1. Результаты тестирования программы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Входные данные** | **С постоянным шагом** | **С дроблением шага** | **Наискорейшего спуска** | **Ньютона-сопряженного градиента** | **Стороннее решение** |
|  | х = (0;0)  f(x) = -6 | х = (0;0)  f(x) = -6 | х = (0;0)  f(x) = -6 | х = (0;0)  f(x) = -6 | х = (0;0)  f(x) = -6 |
|  | х = 2,499  f(x) = 0,5 | х = 2,499  f(x) = 0,5 | х = 2,5  f(x) = 0,5 | х = 2,5  f(x) = 0,5 | х = 2,5  f(x) = 0,5 |
|  | х = -3873  f(x) = -15494 | х = -3873  f(x) = -15494 | х = -pi - asin(4)  f(x) = -4\*pi - sqrt(15)\*I - 4\*asin(4) | х = -pi - asin(4)  f(x) = -4\*pi - sqrt(15)\*I - 4\*asin(4) | х = -0,3  f(x) = -0,244 |
|  | х = -1,65  f(x) = 1 | х = -157,9  f(x) = 10,12 | х = 1  f(x) = 0 | х = 1  f(x) = 0 | х = 1  f(x) = 0 |
|  | х = (-0,31;-0,156)  f(x) = 0,77 | х = (-0,31;-0,156)  f(x) = 0,77 | х = (-0,263; -0,263).  f(x) = 0,5 | x=( -LambertW(2/3)/2, -LambertW(2/3)/2  )  f(x) = 3\*LambertW(2/3)\*\*2/4 + exp(-LambertW(2/3)) | х = (-0,263; -0,263).  f(x) = 0,5 |

Таблица 2. Время выполнения в секундах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Функция | **С постоянным шагом** | **С дроблением шага** | **Наискорейшего спуска** | **Ньютона-сопряженного градиента** |
|  | 0.0 | 0.0 | 0.010969877243041992 | 0.0019953250885009766 |
|  | 0.028 | 0.033 | 0.015982866287231445 | 0.0036635398864746094 |
|  | 6.45 | 7.84 | 0.053792476654052734 | 0.0660698413848877 |
|  | 0.324 | 0.365 | 0.003917217254638672 | - |
|  | 0.047 | 0.056 | 0.0067272556554638672 | 0.05316901206970215 |

**По итогам тестирования необходимо сделать выводы о предпочтительном алгоритме использования и его точности/погрешности.**

1. **Заключение**

**Функция заказчика: 4\*x + x\*\*2 + (200 - x)\*\*2**

**Решение методом Ньютона-сопряженного градиента:**

x = 99, f(x) = 20398. Время работы алгоритма - 0.0254569 сек.

**Решение методом градиентного спуска с постоянным шагом:**

x = 98.99, f(x) = 20398. Время работы алгоритма - 0.032903 сек.

**Решение методом градиентного спуска с дроблением шага:**

x = 98.99, f(x) = 20398. Время работы алгоритма - 0.0418810 сек.

**Решение методом наискорейшего спуска:**

**Решение стороннего источника:**

x = 99, f(x) = 20398.

**Таблица 3. Сравнение алгоритмов.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Критерий** | **С постоянным шагом** | **С дроблением шага** | **Наискорейшего спуска** | **Ньютона-сопряженного градиента** |
| Количество возможных параметров | Не ограничено | Не ограничено | Не ограничено | Не ограничено |
| Требования к памяти | Нет | Нет | Нет | Нет |
| Время выполнения | 0.032903 | 0.0418810 | 0.003917217254638672 | 0.0254569 |
| Удобство использования | **-** | **-** | **+** | **+** |
| Правильность работы алгоритма | Ответ сошелся | Ответ сошелся | Ответ сошелся | Ответ сошелся |

**По результатам таблицы 3 необходимо выбрать оптимальный для заказчика, по Вашему мнению, алгоритм.**

**Для выбранного алгоритма следует описать возможные перспективы развития данного алгоритма.**

Наиболее перспективным оказался третий алгоритм – алгоритм наискорейшего спуска, потому что он удобен в использовании и быстрее остальных в использовании. Этот метод может быть использован в разных прикладных задачах , в прогнозировании . Заказчиком может быть практически любая компания , которая распологает необходимым объемом данных.