МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»

(ННГУ)

Институт информационных технологий, математики и механики

Передовая инженерная школа

Направление подготовки: «Прикладная информатика»

Отчет по лабораторной работе

на тему:

**«Метод Нелдера – Мида»**

**Выполнили:**

студенты группы 3821Б1ПИмэ

Волков Д.А.

Бузникова М.О

Гурьянов И.А

И студенты группы 3821Б1ПИсм

Стаценко А.П

Трошанин Ф.И

**Преподаватель:**

Сморякова Валентина Михайловна

Нижний Новгород

2024

Оглавление

[Введение 3](#_Toc164873033)

[Цель работы: 4](#_Toc164873034)

[Теоретический материал 5](#_Toc164873035)

[Алгоритм 6](#_Toc164873036)

[Реализация метода на Python 7](#_Toc164873037)

[Архитектура программы 7](#_Toc164873038)

[Описание основной логики программы 7](#_Toc164873039)

[Источники: 14](#_Toc164873040)

# Введение

Метод Нелдера — Мида, также известный как метод деформируемого многогранника и симплекс-метод, — метод безусловной оптимизации функции от нескольких переменных, не использующий производной градиентов функции.

Метод Нелдера – Мида был разработан в 1965 году Джоном Нелдером и Роджером Мидом. Они искали метод оптимизации, который мог бы работать с функциями, не имеющими производных или не имеющими аналитических формул для производных. Они также хотели разработать метод, который был бы прост в реализации и эффективен для использования на вычислительных машинах того времени. Исследования привели их к идее использования симплекса - многогранника в пространстве параметров функции.

Симплекс представляет собой набор точек, образующих многогранник, где каждая точка – это набор значений параметров оптимизируемой функции. Идея заключается в том, чтобы изменять и перемещать симплекс в пространстве параметров, чтобы найти оптимальное значение функции.

# Цель работы:

В результате лабораторной работы, мы должны не только изучить метод Нелдера – Мида, но и научиться применять его на практике, а также запрограммировать его, на выбранном нами языке программирования.

Еще перед нами стоит задача выявить преимущества и недостатки данного метода.

# Теоретический материал

Метод Нелдера – Мида, или метод деформируемого многогранника, применяется для нахождения решения задачи оптимизации вещественных функций многих переменных

причем функция f(x) не является гладкой.

Другой особенностью метода является то, что на каждой итерации вычисляется значение функции f(x) не более чем в трех точках. Это особенно важно в случае сложно вычислимой функции f(x). Метод Нелдера – Мида прост в реализации и полезен на практике, но, с другой стороны, для него не существует теории сходимости — алгоритм может расходиться даже на гладких функциях.

Симплексом (n-симплексом) называется выпуклая оболочка множества независимых (n + 1) точек (вершин симплекса).

# Алгоритм

Алгоритм метода Нелдера – Мида включает следующие шаги:

1. Инициализация: строится симплекс - многогранник в n-мерном пространстве, где n - количество переменных функции оптимизации;
2. Оценка значений функции: вычисляются значения функции в вершинах симплекса;
3. Сортировка: вершины сортируются в порядке убывания значений функции;
4. Центр тяжести: вычисляется центр тяжести симплекса без наихудшей вершины;
5. Отражение: проводится отражение на основе центра тяжести и лучшей вершины симплекса;
6. Оценка нового значения функции в отраженной точке;
7. Расширение: если новая точка лучше, чем лучшая вершина симплекса, происходит расширение в этом направлении;
8. Сжатие: если новая точка хуже, чем наихудшая вершина, происходит сжатие симплекса в сторону лучшей вершины;
9. Метод средней точки: если ни одно из вышеупомянутых действий не улучшает значение функции, то сжимаем симплекс в сторону центра тяжести без наихудшей вершины;
10. Условие останова: алгоритм завершается, если выполнено условие сходимости, например, при достижении заданной точности или после определенного числа итераций.

Алгоритм метода Нелдера – Мида повторяется до достижения оптимального значения функции или условия останова.

# Реализация метода на Python

## Архитектура программы

Наша реализация состоит из трех файлов:

1. input.txt;
2. func\_parser.py;
3. nelder\_mead.py.

Файл input.txt содержит входные данные – функция, которую необходимо оптимизировать.

Файл func\_parser.py содержит метод, который парсит строку `s`, преобразуя символы в математическое выражение с использованием символа "x" как переменной и чисел как показателя степени. Возвращает также количество уникальных чисел в строке и преобразованную строку

И файл nelder\_mead.py. Помимо реализации метода Нелдера – Мида, в нем содержатся функции, которые работают с массивами точек и их значениями, выполняющие операции по нахождению симметричных точек, средних значений и поиску минимальных и максимальных значений в массивах.

## Описание основной логики программы

Начнем по порядку. Файл входных данных содержит в себе следующую функцию:

Функция представляет собой квадратичную функцию с минимумом в точке (0; 6,588525). Такие функции обычно хорошо подходят для метода Нелдера – Мида, так как этот метод является недифференцируемым и применим для минимизации негладких функций без необходимости вычисления производных.

Теперь более подробно о файле, в котором происходит преобразование строки символов в математическое выражение. Пройдемся по коду:

class FuncParser:

@staticmethod

def func\_pars(s):

answ = []

arg = False

num\_of\_x = ""

nums = set()

for c in s:

if c == "x":

answ.append("x[")

arg = True

elif c.isdigit():

num\_of\_x += c

elif not c.isdigit() and arg:

answ.append(str(int(num\_of\_x) - 1) + "]")

nums.add(int(num\_of\_x))

arg = False

num\_of\_x = ""

if c == "^":

answ.append("\*\*")

else:

answ.append(c)

elif not c.isdigit():

if len(num\_of\_x) > 0:

answ.append(num\_of\_x + c)

num\_of\_x = ""

elif c == "^":

answ.append("\*\*")

else:

answ.append(c)

if arg:

answ.append(str(int(num\_of\_x) - 1) + "]")

nums.add(int(num\_of\_x))

if len(num\_of\_x) > 0:

answ.append(num\_of\_x)

num\_of\_x = ""

return len(nums), "".join(answ)

Этот код представляет собой статический метод «func\_pars», определенный в классе «FuncParser». Метод принимает строку «s» в качестве аргумента и возвращает кортеж из двух значений: количество уникальных чисел, содержащихся в строке, и преобразованную строку.

Алгоритм работы метода следующий:

1. Проходим по символам строки «s»;
2. Если символ равен «x», добавляем «x[« в результат и устанавливаем флаг «arg» в True;
3. Если символ является цифрой, добавляем его в «num\_of\_x»;
4. Если символ не является цифрой и «arg» установлен в True, добавляем значение «num\_of\_x» минус 1 с символом "]" в результат. Добавляем значение «num\_of\_x» в множество «nums». Сбрасываем флаг «arg» и очищаем «num\_of\_x»;
5. Если символ равен «^», добавляем «\*\*» в результат;
6. В других случаях, добавляем символ в результат;
7. Если «num\_of\_x» не пуст, добавляем его с текущим символом в результат;
8. Если «arg» установлен в True, добавляем значение «num\_of\_x» минус 1 с символом «]» в результат. Добавляем значение «num\_of\_x» в «nums»;
9. После завершения цикла, если «num\_of\_x» не пуст, добавляем его в результат;
10. Возвращаем кортеж из количества уникальных чисел в «nums» и объединенной строки «answ».

Таким образом, метод парсит строку «s», преобразуя символы в математическое выражение с использованием символа «x» как переменной и чисел как показателя степени. Возвращает также количество уникальных чисел в строке и преобразованную строку.

А теперь основной файл с реализацией метода Нелдера – Мида.

Для реализации нам понадобились вспомогательные функции:

1. symmetrical\_point;
2. average\_except\_one\_point;
3. index\_second\_max;
4. index\_min\_max;
5. shrink\_points;

Функция, возвращающая координату симметричной точки – symmetrical\_point.

def symmetrical\_point(x1, x2, a):

return x1 + a \* (x2 - x1)

Функция принимает три аргумента: x1, x2 и a. Для нахождения точки используется формула:

,

где x1 и x2 - координаты точек, а «a» - расстояние до симметричной точки.

Следующая функция позволяет вычислить среднее значение массива, исключая определенный элемент по его индексу.

def average\_except\_one\_point(array, exception\_index):

average\_point = (sum(array) - array[exception\_index]) / (len(array) - 1)

return average\_point

Average\_except\_one\_point принимает два аргумента: array (список чисел) и exception\_index (индекс элемента для исключения из вычисления среднего).

Еще нам понадобилась функция по нахождению второго максимального индекса в массиве – index\_second\_max.

def index\_second\_max(array):

second\_max = min(array)

index\_max = array.index(max(array))

len\_array = len(array)

for i in range(len\_array):

if i != index\_max and second\_max < array[i]:

second\_max = array[i]

return array.index(second\_max)

Далее мы реализовали функцию по поиску индексов максимального и минимального значения в списке – index\_min\_max.

def index\_min\_max(array):

return array.index(min(array)), array.index(max(array))

И функция shrink\_points. Функция принимает два аргумента: array\_points (список точек) и array\_values (список значений). Она находит индекс точки с минимальным значением в array\_values, затем создает симметричный симплекс относительно этой точки с коэффициентом уменьшения 0.5 и возвращает его.

def shrink\_points(array\_points, array\_values):

index\_min = array\_values.index(min(array\_values))

symmetrical\_simplex = symmetrical\_point(

array\_points, array\_points[index\_min], 0.5

)

return symmetrical\_simplex

Наконец можем перейти коду с реализацией метода Нелдера – Мида:

def calculate\_neldermead(function, dimension, alpha=1.5, beta=1.5, gamma=1.5):

if alpha <= 0 or beta <= 1 or gamma <= 1 or dimension <= 0:

raise RuntimeError

simplex = np.eye(dimension + 1, dimension)

while not np.isclose(simplex, simplex[0]).all():

function\_value = [function(x0) for x0 in simplex]

index\_min, index\_max = index\_min\_max(function\_value)

average\_x = average\_except\_one\_point(simplex, index\_max)

symmetrical\_x = symmetrical\_point(

average\_x, simplex[index\_max], -alpha

)

if function\_value[index\_min] > function(symmetrical\_x):

x1 = symmetrical\_point(average\_x, symmetrical\_x, gamma)

if function(symmetrical\_x) > function(x1):

simplex[index\_max] = x1

else:

simplex[index\_max] = symmetrical\_x

else:

if function\_value[index\_second\_max(function\_value)] >= function(

symmetrical\_x

):

simplex[index\_max] = symmetrical\_x

if function(symmetrical\_x) < function\_value[index\_max]:

function\_x2 = function(symmetrical\_x)

x2 = symmetrical\_x

else:

function\_x2 = function\_value[index\_max]

x2 = simplex[index\_max]

x3 = symmetrical\_point(average\_x, x2, beta)

if function(x3) > function\_x2:

simplex = shrink\_points(simplex, function\_value)

else:

simplex[index\_max] = x3

return np.round(sum(simplex) / (dimension + 1), 5)

Функция calculate\_neldermead выполняет метод Нелдера-Мида для минимизации многомерных функций. Параметры: function (целевая функция, которая принимает вектор в качестве входных данных и возвращает скалярное значение), dimension (размерность целевой функции), alpha, beta, gamma (параметры метода Нелдера-Мида).

Функция проверяет, действительны ли входные данные:

alpha, beta, gamma должны быть положительными и dimension должна быть положительной.

Инициализация:

Создается единичный симплекс, матрица из dimension + 1 строк и dimension столбцов. Каждая строка симплекса представляет один из углов симплекса.

Основной цикл:

1. Действия продолжаются, пока симплекс не перестанет существенно меняться.
2. Для каждого угла симплекса вычисляется соответствующее значение целевой функции.
3. Находятся индексы угла с минимальным и максимальным значениями целевой функции (index\_min и index\_max).

В зависимости от относительных значений целевой функции и параметров alpha, beta, gamma симплекс обновляется следующим образом:

1. Если function\_value[index\_min] (минимальное значение целевой функции) больше, чем значение целевой функции для симметричной точки (symmetrical\_x): вычисляется новая точка x1 как симметричная точка между центром симплекса (кроме угла с максимальным значением) и симметричной точкой. Если значение целевой функции для x1 меньше, чем для symmetrical\_x, то index\_max заменяется на x1. В противном случае index\_max заменяется на symmetrical\_x.
2. Если значение целевой функции для второго максимального угла больше, чем значение целевой функции для symmetrical\_x: index\_max заменяется на symmetrical\_x, если значение целевой функции для symmetrical\_x меньше, чем для index\_max.
3. Иначе вычисляются новые точки x2 и x3 и обновляется симплекс в зависимости от значений целевой функции для этих точек.

После того как симплекс перестал существенно меняться, функция возвращает среднюю точку всех углов симплекса с округлением до 5 знаков после запятой.

# Источники:

1. <https://ru.wikipedia.org/wiki/Метод_Нелдера_—_Мида>
2. <https://www.mql5.com/ru/articles/13805>
3. <https://mathprofi.com/uploads/files/1156_f_41_metody-optimizacii-negladkih-funkcii.pdf?key=a94b32268f6014b85c9c48ef842cc5db/>