СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 1](#_Toc131115183)

[ВВЕДЕНИЕ 2](#_Toc131115184)

[1. МЕТОДЫ НУЛЕВОГО ПОРЯДКА 3](#_Toc131115185)

[1.1 Симплексный метод 3](#_Toc131115186)

[1.1.1 Описание метода 3](#_Toc131115187)

[1.1.2 Ручной расчет 4](#_Toc131115188)

[1.1.3 Программная реализация 4](#_Toc131115189)

[1.2 Метод Нелдера-Мида 5](#_Toc131115190)

[1.2.1 Описание метода 5](#_Toc131115191)

[1.2.2 Ручной расчет 6](#_Toc131115192)

[1.2.3 Программная реализация 6](#_Toc131115193)

[2 МЕТОДЫ ПЕРВОГО ПОРЯДКА 7](#_Toc131115194)

[2.1 Метод наискорейшего градиентного спуска 7](#_Toc131115195)

[2.1.1 Описание метода 7](#_Toc131115196)

[2.1.2 Ручной расчет 8](#_Toc131115197)

[2.1.3 Программная реализация 8](#_Toc131115198)

[2.2 Метод Флетчера-Ривса 8](#_Toc131115199)

[2.2.1 Описание метода 8](#_Toc131115200)

[2.2.2 Ручной расчет 8](#_Toc131115201)

[2.2.3 Программная реализация 8](#_Toc131115202)

[3 МЕТОДЫ ВТОРОГО ПОРЯДКА 9](#_Toc131115203)

[3.1 Метод Ньютона-Рафсона 9](#_Toc131115204)

[3.1.1 Описание метода 9](#_Toc131115205)

[3.1.2 Ручной расчет 9](#_Toc131115206)

[3.1.3 Программная реализация 9](#_Toc131115207)

[ЗАКЛЮЧЕНИЕ 10](#_Toc131115208)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 11](#_Toc131115209)

[ПРИЛОЖЕНИЯ 12](#_Toc131115210)

ВВЕДЕНИЕ

…

В качестве исследуемой функции, выбрана функция Бута (Рисунок 1). Глобальный минимум функции , где .

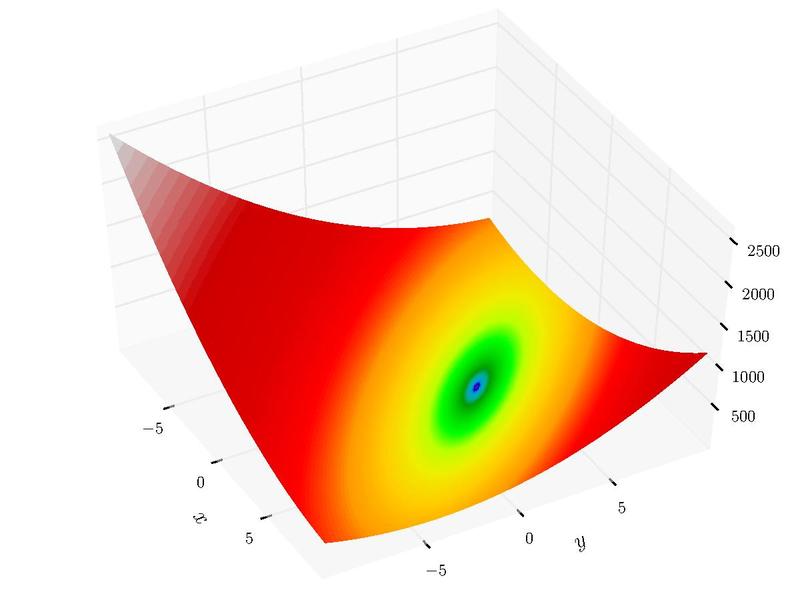


Рисунок 1 – Функция Бута

Для корректного сравнения результатов работы написанных программ, использованы одинаковые входные данные (начальная координата, точность поиска).

1. МЕТОДЫ НУЛЕВОГО ПОРЯДКА

Методы нулевого порядка используют информацию только о значениях целевой функции 𝑓(𝑥). Направление минимизации в данном случае полностью определяется последовательными вычислениями значений целевой функции. Ниже рассматриваются методы, довольно часто применяемые на практике: метод Хука−Дживса, симплексный метод, метод Нелдера−Мида (деформируемого многогранника). Основное достоинство этих методов состоит в том, что они не требуют непрерывности целевой функции и существования производных.

* 1. Симплексный метод
     1. Описание метода

Регулярным симплексом в n-мерном пространстве называется правильный многогранник с вершиной. При симплексом является правильный треугольник, при – тетраэдр и т.д. Отрезок, соединяющий 2 вершины симплекса, называется ребром симплекса.

Поиск симплексным методом ведется по следующей схеме. Устанавливаются координаты вершин симплексов. Определяется вершина с наибольшим значением целевой функции. Вместо нее сроиться новая вершина отражением старой через центр тяжести остальных вершин симплекса (Рисунок 1.1).

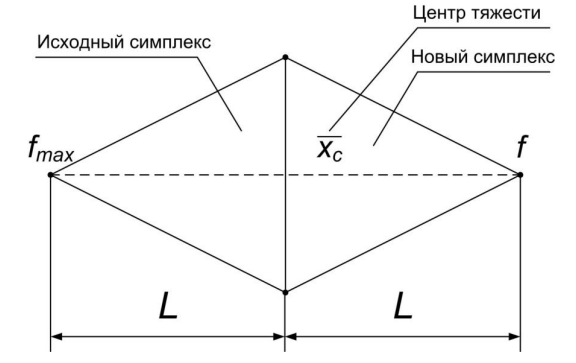


Рисунок 1.1 - Построение нового симплекса

Если попытка отражения не приводит к уменьшению целевой функции, то выполняется операция редукции, в результате которой формируется новый симплекс с уменьшенными вдвое сторонами. При операции редукции в качестве базовой точки выбирается вершина старого симплекса, в которой функция принимает минимальное значение.

В результате исключения вершин симплексов с наибольшим значением целевой функции процесс поиска сходится к минимальному значению.

Поиск завершается, когда разности между значениями функции в центре тяжести симплекса и вершинах становится достаточно малым.[1]

* + 1. Ручной расчет

…

* + 1. Программная реализация

Код программы написан на языке программирования Python. Для программной реализации алгоритма были задействованы библиотеки: numpy (для упрощения работы с базисом), math (для реализации математических операций), pandas (для заполнения и корректного отображения таблицы векторов). Полный код программы представлен в приложении (Приложение А). Результат работы программы представлен на рисунке (Рисунок 1.2).

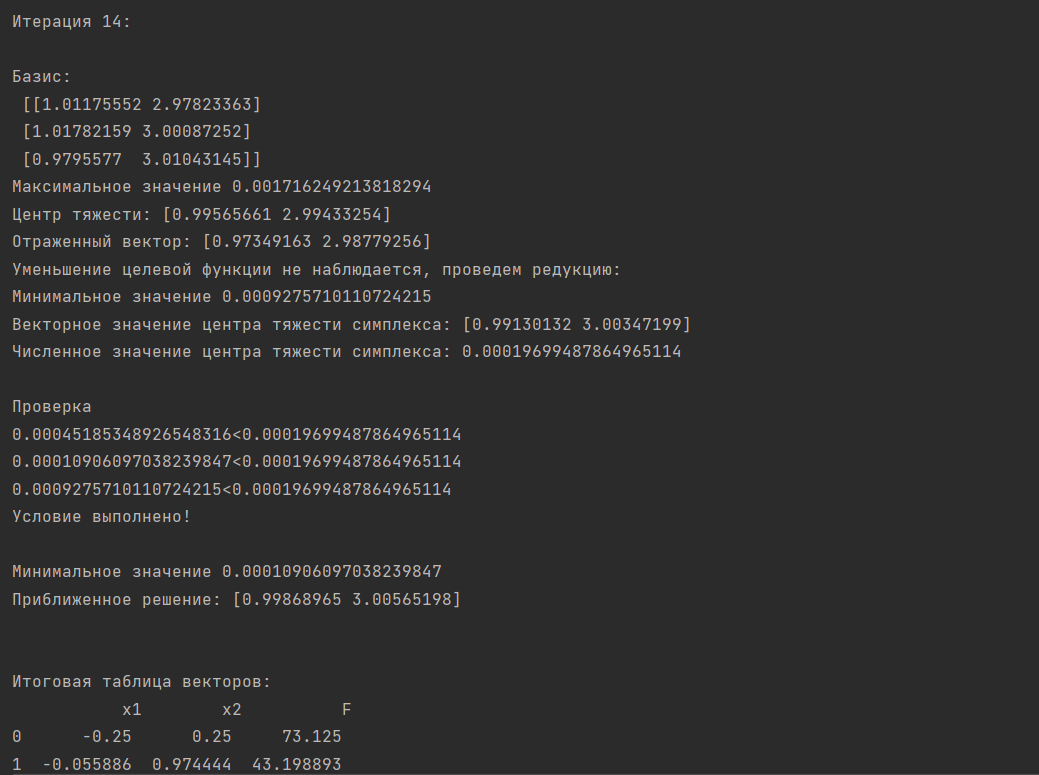


Рисунок 1.2 – Результат работы программы

* 1. Метод Нелдера-Мида
     1. Описание метода

…

* + 1. Ручной расчет

…

* + 1. Программная реализация

…

1. МЕТОДЫ ПЕРВОГО ПОРЯДКА

Методы первого порядка используют информацию о значениях целевой функции и её первых производных. Предполагается, что функция и её первые производные существуют и непрерывны. Направление смещения от точки к точке описывается представленной на первой лекции интеграционной процедурой (1)

, где

k – номер итерации ;

– текущее приближение;

– величина шага;

– вектор, определяющий направление убывания функции и совпадает с направлением вектора антиградиента целевой функции.

Все итерационные процессы, в которых направление движения на каждом шаге совпадает с антиградиентом функции, называются градиентными методами. Существует несколько модификаций градиентных методов, различающихся правилом выбора длины шага в направлении антиградиента функции.

* 1. Метод наискорейшего градиентного спуска
     1. Описание метода

…

* + 1. Ручной расчет

…

* + 1. Программная реализация

…

* 1. Метод Флетчера-Ривса
     1. Описание метода

…

* + 1. Ручной расчет

…

* + 1. Программная реализация

…

1. МЕТОДЫ ВТОРОГО ПОРЯДКА

В методах второго порядка при поиске минимума функции многих переменных используют информацию о частных производных целевой функции первого и второго порядка. К этой группе относят метод Ньютона и его модификации.

* 1. Метод Ньютона-Рафсона
     1. Описание метода

…

* + 1. Ручной расчет

…

* + 1. Программная реализация

…

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Безусловная оптимизация [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк . — М.: РТУ МИРЭА, 2020 . — Электрон. опт. диск (ISO)

ПРИЛОЖЕНИЯ

Приложение А – Программная реализация симплексного метода

Приложение Б – Программная реализация метода Нелдера-Мида

Приложение В – Программная реализация метода наискорейшего градиентного спуска

Приложение Г – Программная реализация метода Флетчера-Ривса

Приложение Д – Программная реализация метода Ньютона-Рафсона

Приложение А

Программная реализация симплексного метода

Листинг А.1 – Код программы

|  |
| --- |
| from typing import Union import pandas as pd import numpy as np from math import sqrt   simplex\_table = pd.DataFrame(columns=['x1', 'x2', 'F']) # Таблица   class ConstantsNamespace:  DIMENSION: int = 2 # Размерность задачи  EDGE\_LENGTH: float = 0.75 # Длина ребра симплекса  ACCURACY: float = 0.001 # Точность  START\_POINT: tuple = (-0.25, 0.25) # Начальные координаты   def target\_function(x: tuple) -> float:  """Целевая функция   Args:  x (tuple): \_description\_   Returns:  float: \_description\_  """  return (x[0] + 2 \* x[1] - 7) \*\* 2 + (2 \* x[0] + x[1] - 5) \*\* 2 # Функция Бута  def find\_center(index: int, basis: list, n: int) -> tuple:  """Функция для нахождения центра тяжести   Args:  index (int): \_description\_  basis (Union[list, tuple]): \_description\_  n (int): \_description\_  Returns:  tuple: \_description\_  """  center = (0, 0)  for i in range(len(basis)):  if i != index:  center += basis[i]  center /= n  print(f'Центр тяжести: {center}')  return center |

Продолжение листинга А.1

|  |
| --- |
| def find\_center\_simplex(basis: list, n: int) -> tuple:  """Функция нахождения векторного значения центра тяжести симплекса   Args:  basis (Union[list, tuple]): \_description\_  n (int): \_description\_  Returns:  tuple: \_description\_  """  center = sum(basis) / (n + 1)  print(f'Векторное значение центра тяжести симплекса: {center}')  return center   def find\_increments(n: int, m: Union[float, int]) -> tuple:  """Функция для нахождения инкрементов   Args:  n (int): \_description\_  m (float): \_description\_  Returns:  tuple: \_description\_  """  b1 = ((sqrt(n + 1) - 1) / (n \* sqrt(2))) \* m  b2 = ((sqrt(n + 1) + n - 1) / (n \* sqrt(2))) \* m  print(f'Приращения: delta1 = {b1}, delta2 = {b2}.')  return b1, b2   # Функция для нахождения ндекса макисмального значения функции в базизсном векторе def find\_max(basis: list) -> int:  max\_f = [target\_function(basis[0]), 0]  for i in range(0, len(basis)): # 0 писать не обязательно, итерация и так с нуля  if target\_function(basis[i]) > max\_f[0]:  max\_f = [target\_function(basis[i]), i]  print('Максимальное значение {0}'.format(max\_f[0]))  return max\_f[1]   # Функция для нахождения индекса минимального значения функции в базизсном векторе def find\_min(basis: list) -> int:  min\_f = [target\_function(basis[0]), 0]  for i in range(0, len(basis)):  if target\_function(basis[i]) < min\_f[0]:  min\_f = [target\_function(basis[i]), i]  print('Минимальное значение {0}'.format(min\_f[0]))  return min\_f[1] |

Продолжение листинга А.1

|  |
| --- |
| # Функция для нахождения отраженного вектора def reflected\_vector(center: tuple, basis: list, index: int) -> tuple:  new\_vector = 2 \* np.array(center) - np.array(basis[index])  print(f'Отраженный вектор: {new\_vector}')  return new\_vector   def reduction(basis: list, index: int) -> list:   """Редукция   Args:  basis (): \_description\_  index (int): \_description\_   Returns:  \_type\_: \_description\_  """  for i in range(len(basis)):  if i != index:  basis[i] = basis[index] + 0.5 \* (basis[i] - basis[index])  simplex\_table.loc[len(simplex\_table)] = np.append(basis[i], target\_function(basis[i]))  return basis   def check\_end(center: float, basis: list, e: float) -> bool:  """Функция проверки завершения алгоритма   Args:  center (float): \_description\_  basis (tuple): \_description\_  e (float): \_description\_   Returns:  flag (bool): \_description\_  """  print('\nПроверка')  flag: bool = True  for vector in basis:  f\_vector: float = target\_function(vector)  if f\_vector - center > e:  print(f'{f\_vector}>{center}')  flag = False  else:  print(f'{f\_vector}<{center}')  return flag |

Продолжение листинга А.1

|  |
| --- |
| class SimplexMethod:  """  Класс для расчета симплекса  """   def \_\_init\_\_(self, n: int, m: Union[float, int], e: float, x: tuple):  """Конструктор класса SimplexMethod   Args:  n (int): \_description\_  m (int): \_description\_  e (float): \_description\_  x (\_type\_): \_description\_  """   self.n: int = n  self.m: float = m  self.e: float = e  self.basis: list = np.zeros((n + 1, n)) # = [[0 for \_ in range(n)]] \* (n + 1)  self.increments: tuple = find\_increments(n, m)  self.\_\_start\_basis(x)   def \_\_start\_basis(self, x: tuple):  """Функция для инициализации стартового базиса   Args:  x (list): Стартовая точка  """  self.basis[0] = x  self.basis[1][0], self.basis[1][1] = self.basis[0] + self.increments  self.basis[2][1], self.basis[2][0] = self.basis[0] + self.increments  simplex\_table.loc[0] = np.append(self.basis[0], target\_function(self.basis[0]))  simplex\_table.loc[1] = np.append(self.basis[1], target\_function(self.basis[1]))  simplex\_table.loc[2] = np.append(self.basis[2], target\_function(self.basis[2]))   def simplex\_run(self, i: int) -> bool:  """  This is a function that's evaluate simplex table?.  Args:  i (integer): Текущая итерация   Returns:  bool: Окончание метода  """  print(f'\nИтерация {i}:\n')  print('Базис:\n', self.basis)  max\_f: int = find\_max(self.basis)  center: tuple = find\_center(max\_f, self.basis.copy(), self.n)  new: tuple = reflected\_vector(center, self.basis.copy(), max\_f)  if target\_function(new) >= target\_function(self.basis[max\_f]):  print('Уменьшение целевой функции не наблюдается, проведем редукцию:')  simplex\_table.loc[len(simplex\_table)] = ['-', '-', '-'] |

Продолжение листинга А.1

|  |
| --- |
| min\_f: int = find\_min(self.basis)  self.basis = reduction(self.basis, min\_f)  else:  self.basis[max\_f] = new  simplex\_table.loc[len(simplex\_table)] = np.append(self.basis[max\_f], target\_function(self.basis[max\_f]))  f\_center = target\_function(find\_center\_simplex(self.basis, self.n))  print(f'Численное значение центра тяжести симплекса: {f\_center}')   if not check\_end(f\_center, self.basis.copy(), self.e):  print('Условие не выполнено\n')  print(simplex\_table)  return True  print('Условие выполнено!\n')  print(f'Приближенное решение: {self.basis[find\_min(self.basis)]}')  return False   def simplex\_method(self):  i = 0  while self.simplex\_run(i):  i += 1  print('\n\nИтоговая таблица векторов: \n', simplex\_table)   if \_\_name\_\_ == "\_\_main\_\_":  constants = ConstantsNamespace()  s1 = SimplexMethod(constants.DIMENSION, constants.EDGE\_LENGTH, constants.ACCURACY, constants.START\_POINT)  s1.simplex\_method() |