МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

**по дисциплине Методы поисковой оптимизации**

Работу выполнила\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ромашкина А.А.

Факультет Компьютерных технологий и прикладной математики

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии курс 4

Краснодар

2022

СОДЕРЖАНИЕ

[Задание 3](#_Toc116326542)

[Алгоритм 3](#_Toc116326543)

[Результат работы программы 4](#_Toc116326544)

[Листинг 5](#_Toc116326545)

Тема работы: Метод градиентного спуска с постоянным шагом.

# Задание

Необходимо разработать программу для решения задачи квадратичного программирования для функции

с ограничениями .

# Алгоритм

1. Составить функцию Лагранжа:

где

1. Определить условия дополняющей нежесткости с помощью условий Куна-Таккера:
2. Преобразовать неравенства в равенства, введя дополнительные переменные:

Исходная задача эквивалентна задаче нахождения допустимого базисного решения системы линейных уравнений (1) и (4), удовлетворяющего условиям дополняющей нежесткости (3) и (6).

1. Далее применяем метод искусственных переменных для определения начального базиса. Для этого в уравнения системы (1) и (4), в которых знаки дополнительных переменных или совпадают со знаками свободных членов, вводятся неотрицательные искусственные переменные , знаки которых не совпадают со знаками соответствующих свободных членов.
2. Затем с помощью симплекс-метода решается вспомогательная задача ЛП: при ограничениях (1) – (6) с введенными неотрицательными искусственными переменными.

Если в результате решения , то оптимальное допустимое базисное решение вспомогательной задачи определяет решение задачи КП.

Если , то задача КП не имеет решения.

# Результат работы программы

Вся работа программы представлена на рисунках 1-2 ниже.

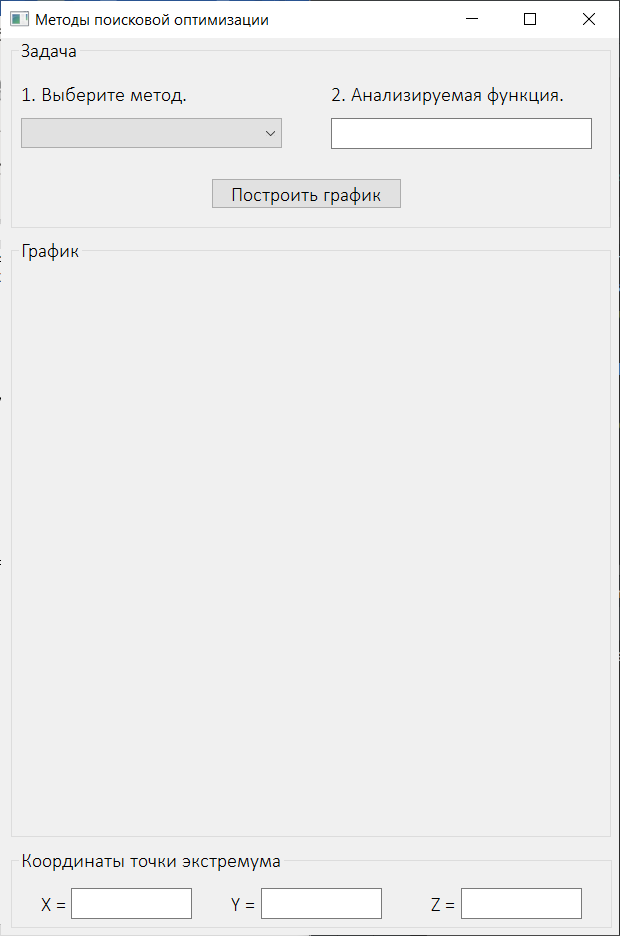


Рисунок 1 – Начальный вид программы при запуске

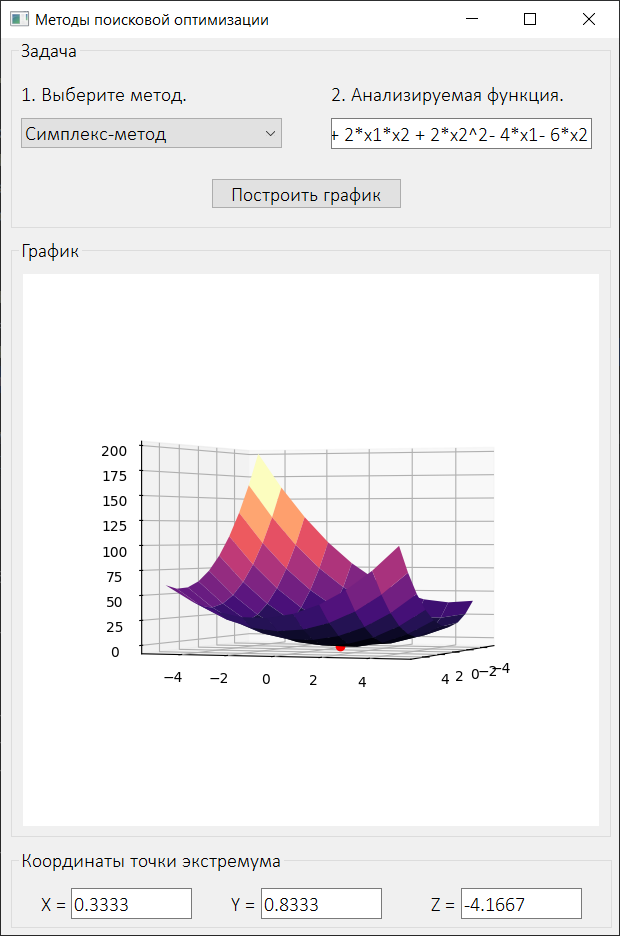


Рисунок 2 – Вид программы при выборе метода градиентного спуска с постоянным шагом

# Листинг

import math

import numpy as np

class Function:

@staticmethod

def function(x\_k):

# x\_k[0] + 2 \* x\_k[1]

# 2 \* (x\_k[0] \*\* 2) + 4 \* x\_k[0] \* x\_k[1] + 3 \* (x\_k[1] \*\* 2) - 6 \* x\_k[0] - 3 \* x\_k[1]

# 2 \* (x\_k[0] \*\* 2) + 2 \* x\_k[0] \* x\_k[1] + 2 \* (x\_k[1] \*\* 2) - 4 \* x\_k[0] - 6 \* x\_k[1]

return 2 \* (x\_k[0] \*\* 2) + 2 \* x\_k[0] \* x\_k[1] + 2 \* (x\_k[1] \*\* 2) - 4 \* x\_k[0] - 6 \* x\_k[1]

def div\_to\_def\_lead\_row(i, j):

if np.sign(i) != np.sign(j):

return math.inf

else:

return i / j

def find\_max\_exclude(list\_items, list\_exclude\_ind):

ind\_max = None

max\_item = None

n = len(list\_items)

for i in range(n):

if i not in list\_exclude\_ind:

ind\_max = i

max\_item = list\_items[i]

break

for i in range(n):

if i not in list\_exclude\_ind:

if list\_items[i] > max\_item:

ind\_max = i

max\_item = list\_items[i]

return ind\_max

def to\_fixed(numObj, digits=0):

return f"{numObj:.{digits}f}"

def print\_number(number):

if np.sign(number) >= 0:

return " " + to\_fixed(number, 2) + " "

else:

return " " + to\_fixed(number, 2) + " "

class SimplexTable:

def \_\_init\_\_(self, basic\_variables, free\_values, free\_variables):

self.basic\_variables = basic\_variables

self.free\_values = free\_values

self.free\_variables = free\_variables

self.coefficients = None

self.leading\_column = None

self.leading\_row = None

def define\_leading\_column\_and\_row(self, check\_for\_identical\_list\_variables, flag):

potential\_columns = []

exclusion = []

n = len(self.free\_variables)

list\_variables = [0 for i in range(n)]

for i in range(len(self.free\_variables)):

potential\_columns.append(abs(self.free\_variables[i][1][-1]))

while True:

# flag1 = 1

self.leading\_column = find\_max\_exclude(potential\_columns, exclusion)

self.define\_leading\_row()

print("leading\_column: " + str(self.leading\_column))

print("leading\_row: " + str(self.leading\_row))

# j = self.free\_variables[self.leading\_column][1]

# for i in range(len(self.free\_variables[self.leading\_column][1])):

# if j[i] == 0.00 or j[i] == -0.00:

# exclusion.append(self.leading\_column)

# flag1 = 0

# if flag1:

if flag != 1:

for i in range(n):

list\_variables[i] = self.free\_variables[i][0][0]

list\_variables[self.leading\_column] = self.basic\_variables[self.leading\_row]

if check\_for\_identical\_list\_variables(set(list\_variables)) or self.free\_variables[self.leading\_column][1][self.leading\_row] == 0:

exclusion.append(self.leading\_column)

else:

break

else:

break

def define\_leading\_row(self):

t = []

for i in range(len(self.free\_values) - 1):

t.append(div\_to\_def\_lead\_row(self.free\_values[i], self.free\_variables[self.leading\_column][1][i]))

self.leading\_row = t.index(min(t))

def update\_basic\_variables(self, prev\_table):

self.basic\_variables.append(prev\_table.free\_variables[prev\_table.leading\_column][0][0])

for i in range(len(prev\_table.basic\_variables)):

if i != prev\_table.leading\_row:

self.basic\_variables.append(prev\_table.basic\_variables[i])

def update\_free\_variables(self, prev\_table):

for i in range(len(prev\_table.free\_variables)):

if i != prev\_table.leading\_column:

self.free\_variables.append([[prev\_table.free\_variables[i][0][0]], []])

else:

self.free\_variables.append([[prev\_table.basic\_variables[prev\_table.leading\_row]], []])

def calc\_coefficients(self, prev\_table):

self.coefficients = []

for i in range(len(prev\_table.free\_variables[prev\_table.leading\_column][1])):

if i != prev\_table.leading\_row:

self.coefficients.append(-1 \* prev\_table.free\_variables[prev\_table.leading\_column][1][i])

def calc\_first\_row\_table(self, prev\_table):

self.free\_values.append(prev\_table.free\_values[prev\_table.leading\_row] / prev\_table.free\_variables[prev\_table.leading\_column][1][prev\_table.leading\_row])

self.free\_variables[prev\_table.leading\_column][1].append(1 / prev\_table.free\_variables[prev\_table.leading\_column][1][prev\_table.leading\_row])

for item in range(len(prev\_table.free\_variables)):

if item != prev\_table.leading\_column:

self.free\_variables[item][1].append(prev\_table.free\_variables[item][1][prev\_table.leading\_row] / prev\_table.free\_variables[prev\_table.leading\_column][1][prev\_table.leading\_row])

def calc\_table\_rows(self, prev\_table):

for i in range(1, len(prev\_table.basic\_variables)):

self.free\_values.append(self.free\_values[0] \* self.coefficients[i - 1] + prev\_table.free\_values[prev\_table.basic\_variables.index(self.basic\_variables[i])])

for j in range(len(prev\_table.free\_variables)):

tmp = self.free\_variables[j][1][0] \* self.coefficients[i - 1]

if self.free\_variables[j][0][0] != prev\_table.basic\_variables[prev\_table.leading\_row]:

tmp += prev\_table.free\_variables[j][1][prev\_table.basic\_variables.index(self.basic\_variables[i])]

self.free\_variables[j][1].append(tmp)

def print\_table(self):

# " x1 ", " x2 ", " x3 ", " x4 ", " x5 ", " x6 ", " F "

# " x1 ", " x2 ", " x3 ", " x4 ", " x5 ", " x6 ", " F "

# " x1 ", " x2 ", " l ", " v1 ", " v2 ", " z1 ", " z2 ", " w ", " F "

# " x1 ", " x2 ", " l ", " v1 ", " v2 ", " z1 ", " z2 ", " w ", " F "

# " x1 ", " x2 ", " l1 ", " l2 ", " v1 ", " v2 ", " z1 ", " z2 ", " w1 ", " w2 ", " F "

# " x1 ", " x2 ", " l1 ", " l2 ", " v1 ", " v2 ", " z1 ", " z2 ", " w1 ", " w2 ", " F "

var = [" x1 ", " x2 ", " l ", " v1 ", " v2 ", " z1 ", " z2 ", " w ", " F "]

var\_ = [" x1 ", " x2 ", " l ", " v1 ", " v2 ", " z1 ", " z2 ", " w ", " F "]

variables = ""

coeff = None

if self.coefficients is None:

coeff = " "

for x in self.free\_variables:

variables += var\_[x[0][0]] + "|"

# -------------------------------------------------

# "basic | free | free\_variables| coefficients"

# "-------------------------------------------------------------------------"

# "basic | free | free\_variables | coefficients"

# "---------------------------------------------------------------------------------"

# "basic | free | free\_variables | coefficients"

print("-------------------------------------------------------------------------")

print("basic | free | free\_variables | coefficients")

print("variables | values |" + variables)

print("-------------------------------------------------------------------------")

fv = ""

for j in self.free\_variables:

fv += print\_number(j[1][0]) + "|"

print(var[self.basic\_variables[0]] + "|" + print\_number(self.free\_values[0]) + " |" + fv + " ")

for i in range(1, len(self.basic\_variables)):

fv = ""

for j in self.free\_variables:

fv += print\_number(j[1][i]) + "|"

if self.coefficients is not None:

coeff = print\_number(self.coefficients[i - 1])

print(var[self.basic\_variables[i]] + "|" + print\_number(self.free\_values[i]) + " |" + fv + coeff)

print("-------------------------------------------------------------------------")

class SimplexMethod:

def \_\_init\_\_(self, list\_variables\_of\_free\_variables, f):

self.list\_variables\_of\_free\_variables = list\_variables\_of\_free\_variables

self.cur\_table = None

self.prev\_table = None

self.f = f

def check\_for\_identical\_list\_variables(self, list\_variables):

for i in self.list\_variables\_of\_free\_variables:

if set(i) == list\_variables:

return True

return False

def update\_list\_variables\_of\_free\_variables(self, table):

tmp = []

for i in range(len(self.list\_variables\_of\_free\_variables[-1])):

tmp.append(self.list\_variables\_of\_free\_variables[-1][i])

tmp[table.leading\_column] = table.basic\_variables[table.leading\_row]

self.list\_variables\_of\_free\_variables.append(tmp)

print("list\_variables\_of\_free\_variables: " + str(self.list\_variables\_of\_free\_variables))

def is\_end(self):

# for i in self.cur\_table.free\_variables:

# if i[1][-1] < 0:

# return False

for i in self.cur\_table.free\_variables:

if i[0][0] != 5 and i[0][0] != 6 and i[1][-1] < 0:

return False

# if i[0][0] != 6 and i[0][0] != 7 and i[1][-1] < 0:

if self.cur\_table.free\_values[-1] >= 0:

return True

return False

def simplex\_method(self, basic\_variables, free\_values, free\_variables):

self.prev\_table = SimplexTable(basic\_variables, free\_values, free\_variables)

self.prev\_table.print\_table()

self.prev\_table.define\_leading\_column\_and\_row(self.check\_for\_identical\_list\_variables, 1)

print("list\_variables\_of\_free\_variables: " + str(self.list\_variables\_of\_free\_variables))

while True:

self.cur\_table = SimplexTable([], [], [])

self.cur\_table.update\_basic\_variables(self.prev\_table)

self.cur\_table.update\_free\_variables(self.prev\_table)

self.cur\_table.calc\_coefficients(self.prev\_table)

self.cur\_table.calc\_first\_row\_table(self.prev\_table)

self.cur\_table.calc\_table\_rows(self.prev\_table)

self.cur\_table.print\_table()

print("is\_end = " + str(self.is\_end()))

if self.is\_end():

break

self.cur\_table.define\_leading\_column\_and\_row(self.check\_for\_identical\_list\_variables, 0)

self.update\_list\_variables\_of\_free\_variables(self.prev\_table)

self.prev\_table.basic\_variables = self.cur\_table.basic\_variables

self.prev\_table.free\_values = self.cur\_table.free\_values

self.prev\_table.free\_variables = self.cur\_table.free\_variables

self.prev\_table.coefficients = self.cur\_table.coefficients

self.prev\_table.leading\_column = self.cur\_table.leading\_column

self.prev\_table.leading\_row = self.cur\_table.leading\_row

input("Press key >>> ")

res = [self.cur\_table.free\_values[self.cur\_table.basic\_variables.index(0)],

self.cur\_table.free\_values[self.cur\_table.basic\_variables.index(1)]]

return res