МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

«Нижегородский государственный университет им. Н.И. Лобачевского»

Национальный исследовательский университет

**ОТЧЕТ**

**о проделанной работе по курсу Методы оптимизации на тему**

**«Нахождение локальных минимумов выпуклой функции»**

Подготовили учащиеся гр. 381906-3 Олюнин А.В. Михеев И.А.

Проверил преподаватель ИИТММ

Шульпин С. М.

Содержание:

[1. Общая постановка задачи 3](#_Toc117629901)

[2. Формулировка задачи и ее интерпретация 4](#_Toc117629902)

[3. Геометрическая преставление задачи 5](#_Toc117629903)

[4. Геометрическое представление условий стационарности 7](#_Toc117629904)

[5. Применение условий стационарности 9](#_Toc117629905)

[5.1. Определение кандидатов локального минимума 9](#_Toc117629906)

[5.2. Проверка выделенных точек на минимум 9](#_Toc117629907)

[6. Вывод 11](#_Toc117629908)

[7. Список литературы и источников информации 12](#_Toc117629909)

[8. Приложение. Код программы и рисунки. 13](#_Toc117629910)

1. Общая постановка задачи

Основные задачи, поставленные в данной лабораторной работе:

1. Рассмотреть задачу нелинейного динамического программирования, поставленную в следующей форме:

где и .

1. Решить задачу с использованием применения теоремы Каруша-Куна-Таккера в дифференциальной форме для выпуклого случая. Проанализировать полученные результаты и сделать выводы.

2. Формулировка задачи и ее интерпретация

Требуется рассмотреть задачу динамического программирования, поставленную следующим образом:

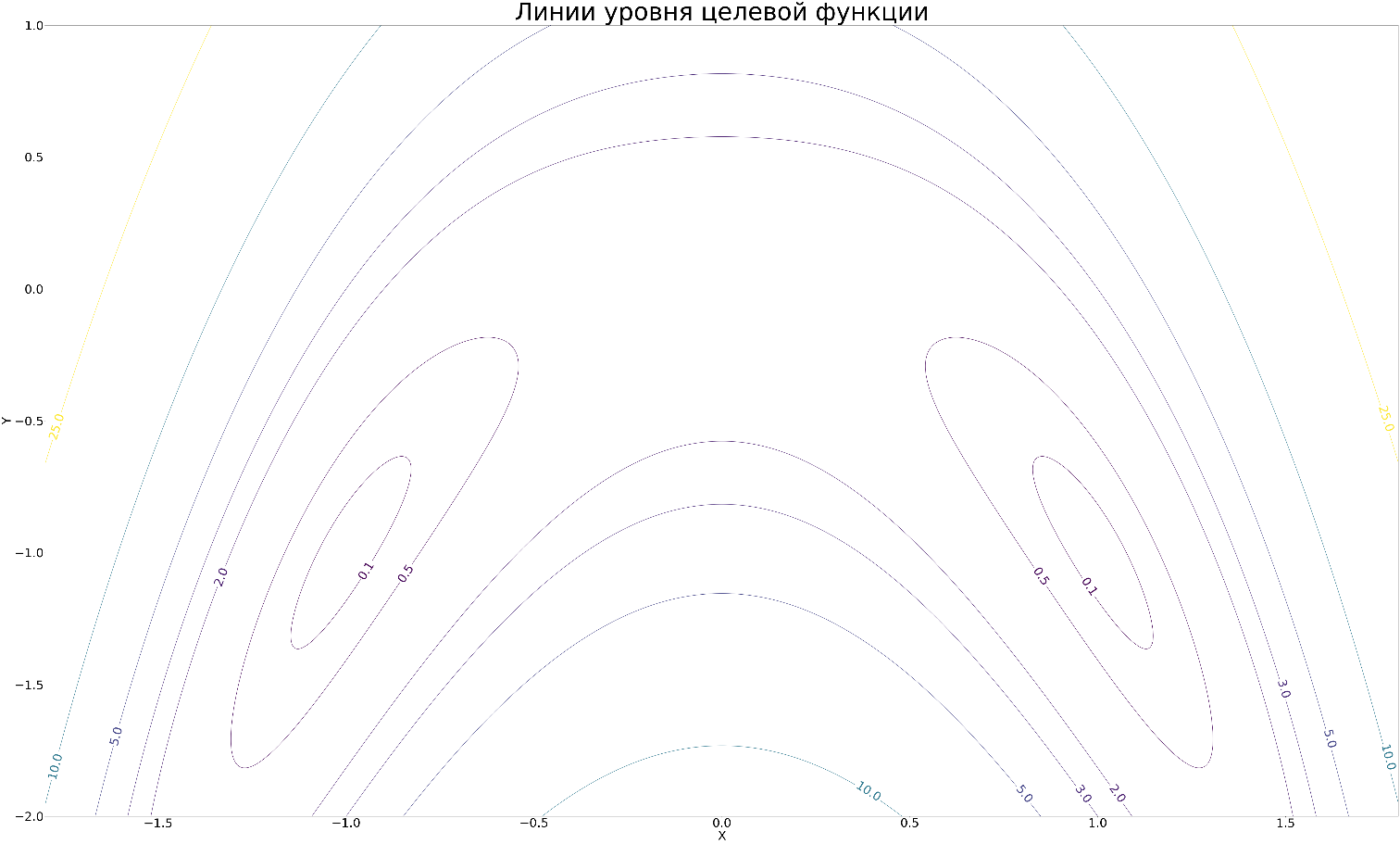
при ограничениях:

где рассмотреть в заданных пределах . Необходимо сделать следующее:

1. Построить достаточно подробные картины линий равного уровня минимизируемой функции, позволяющие понять структуру ее зависимости от параметров; для указанных диапазонов изменения переменных построить вид допустимого множества, понять какие из ограничений отвечают за различные участки границы допустимого множества.
2. Выполнить наложение с прозрачным фоном картинки с границей допустимого множества на картину линий уровня функции. На основе полученной общей картинки понять ориентацию градиентов функции Q, в точках D и ориентацию градиентов активных ограничений в точках границы множества D, выяснить: выпукла ли минимизируемая функция, выпукло ли допустимое множество, регулярно ли допустимое множество.
3. Исходя из геометрического смысла условий оптимальности, отметить на картинке со структурой задачи все точки, где могут выполняться условия Каруша-Куна-Таккера.

3. Геометрическая представление задачи

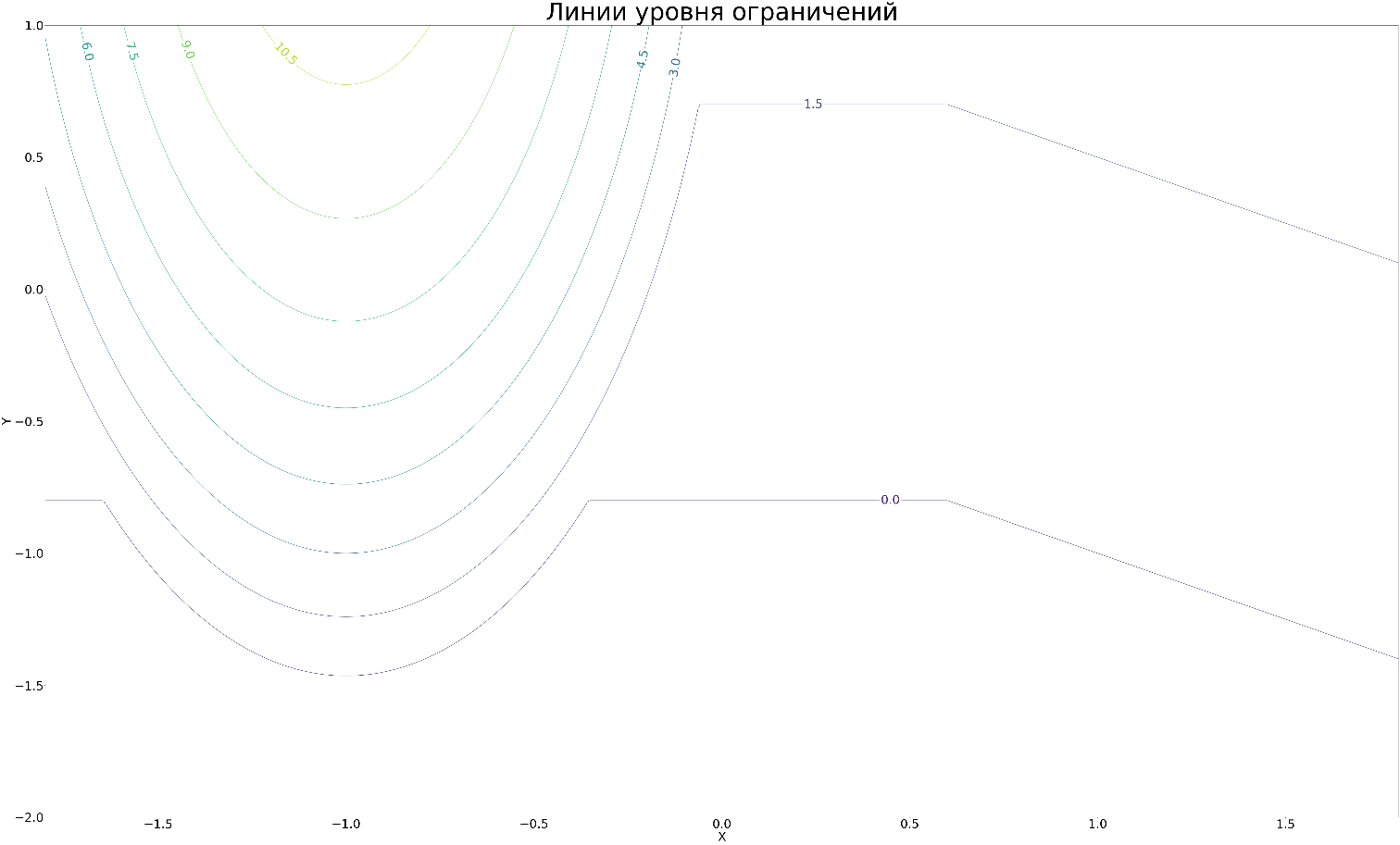
Для того чтобы решить задачу, необходимо построить линии уровня заданной функции и функция ограничений. Воспользуемся библиотекой построения математических графиков matplotlib языка программирования Python. Линии уровня целевой функции , построенные на рис. 3.1. имеют разный цветовой оттенок, где сиреневый означает наименьшие значений, а желтый наибольшие, то есть первые находятся на наименьшей высоте по оси , а вторые на наибольшей по отношению друг другу согласно заданному множеству .



*Рис. 3.1. Линии уровня целевой функции*

В рамках нашей задачи рассмотрение ограничений можно свести к построению некоторой новой функции , взятой как максимум исходных функций . То есть, рассмотрим следующую функцию:

Построим эту функцию аналогичным целевой функции способом. Цвета на графике соответствуют тому же смыслу, изложенному ранее. Но мы будем далее пользоваться более удобным представлением данной функции (см. на рис. 3.3), где области разукрашены в цвета согласно их уровню. Это позволит сразу определить область, в которой стоит искать претендентов на локальный минимум.



*Рис. 3.2. Линии уровня ограничений*



*Рис. 3.3. Области уровней ограничений*

4. Геометрическое представление условий стационарности

В целях решения поставленной задачи в пунктах 1-2 воспользуемся теоремой Каруша-Куна-Таккера условий оптимальности первого порядка для дифференцируемого выпуклого случая.

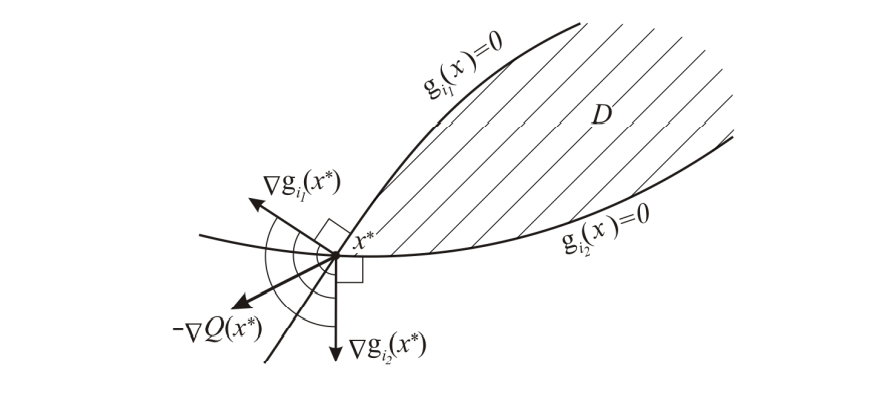
Формулировка теоремы:

*Рассмотрим задачу:*

*где . При этом – выпуклы в и дифференцируемы. Ограничения – аффинные функции, множество – регулярно (например, выполняются условия Слейтера: ). Тогда для глобальной оптимальности точки необходимо и достаточно выполнение следующих условий:*

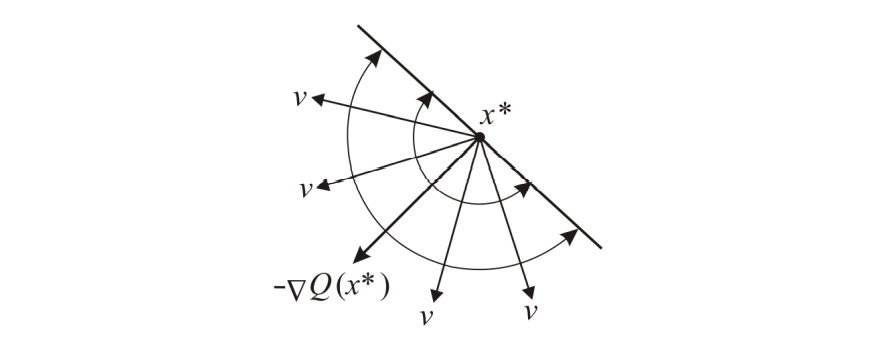
1. *– допустимость*
2. *– существование множителей Лагранжа*
3. *– не отрицательность*
4. *– стационарность*
5. *– условие дополняющей нежесткости.*

Приведем геометрическую интерпретацию выполнения и нарушения условий оптимальности первого порядка. Для простоты возьмем случай, когда ограничения в виде равенств отсутствуют, а число активных ограничений типа неравенства равно двум. Тогда, согласно условию теоремы под пунктом 4, антиградиент функции размещается в конусе, натянутом на векторы градиентов ограничений функций и (см. рис. 4.1.)



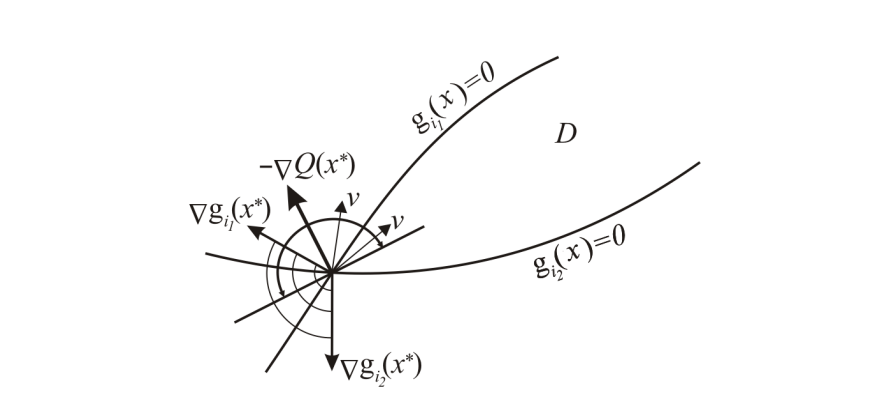
*Рис. 4.1. Ориентация антиградиента при выполнении условий оптимальности*

Действительно, направления, в которых строго локально убывает в точке , образуют открытое полупространство, граница которого ортогональна вектору (см. рис. 4.2.).



*Рис. 4.2. Конус направлений строгого локального убывания функции в точке*

Если же антиградиент выходит, например, за вектор , то правильный знак у нарушится и окажется (см. рис. 4.3).

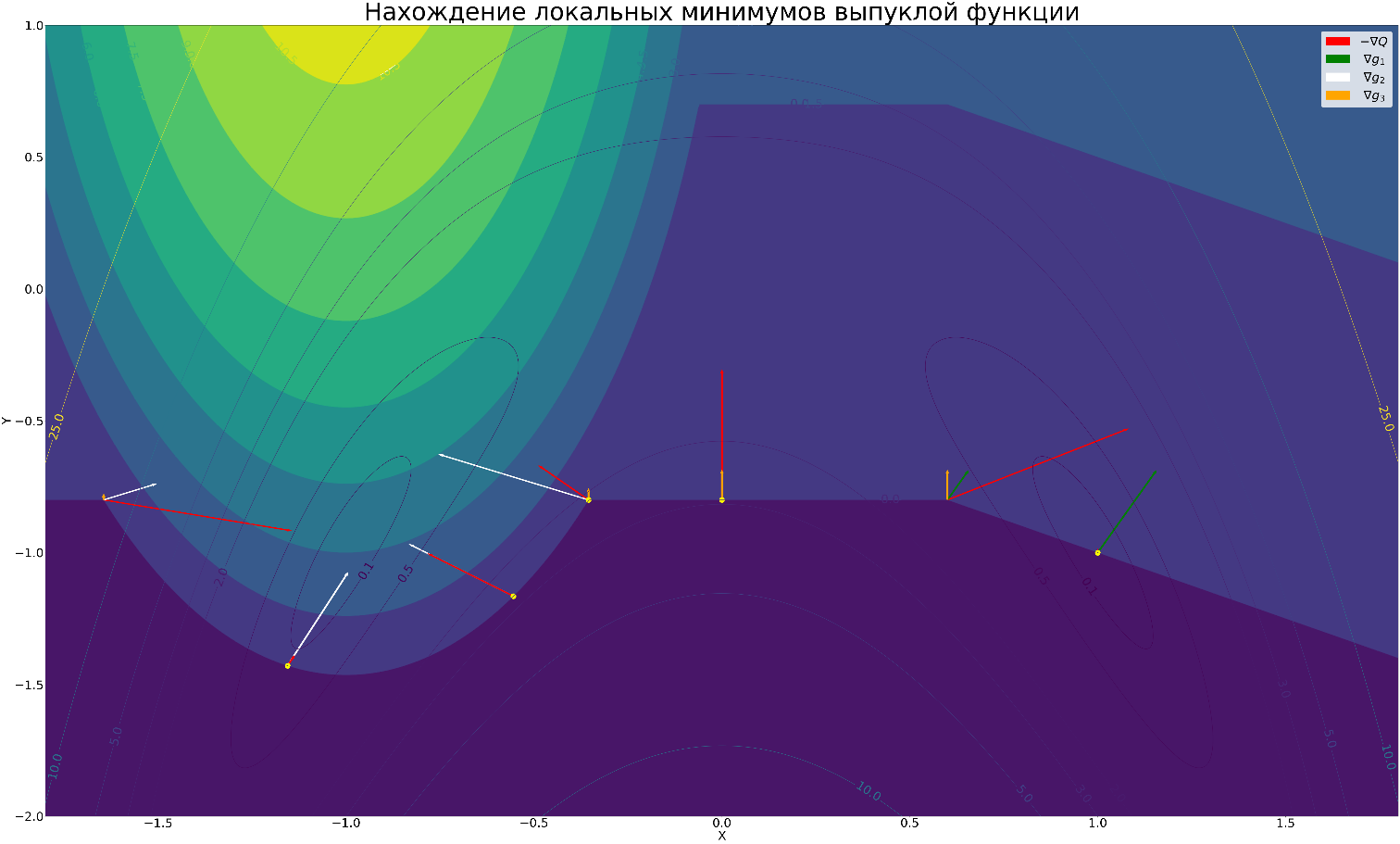


*Рис. 4.3. Нарушение условий оптимальности в точке*

5. Применение условий стационарности

5.1. Определение кандидатов локального минимума

Воспользуемся полученными в пункте 3 графиками и совместим линии уровня целевой функции и ограничения неравенств функции на одном графике (см. рис. 5.1).



Условия первого порядка нарушены

*Рис. 5.1. Нахождение локальных минимумов выпуклой функции*

Рассмотрев рисунок, нетрудно понять, какие точки стоит рассмотреть, как кандидаты на локальный минимум. Это будут те точки, где могут быть выполнены условия первого порядка. Тогда рассмотрим два вида точек:

1. Точки, в которых активны два или более ограничений , и при этом лежит в конусе, образованном с помощью .
2. Точки, в которых активно одно ограничение , и при этом сонаправлен с .

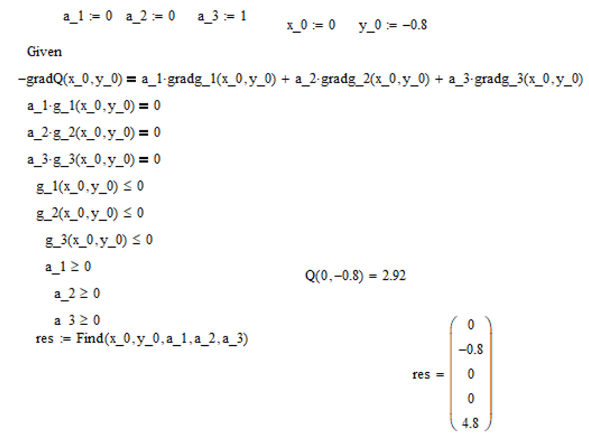
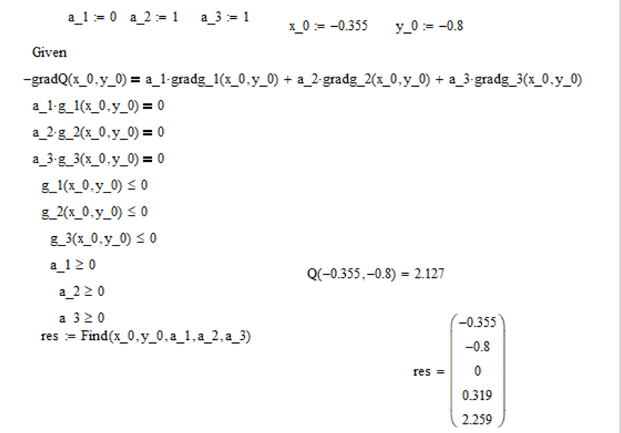
Все такие точки обозначены на рис. 5.1 желтым цветом. Кроме того, ради эксперимента проверим точки, которые не обозначены никаким цветом.

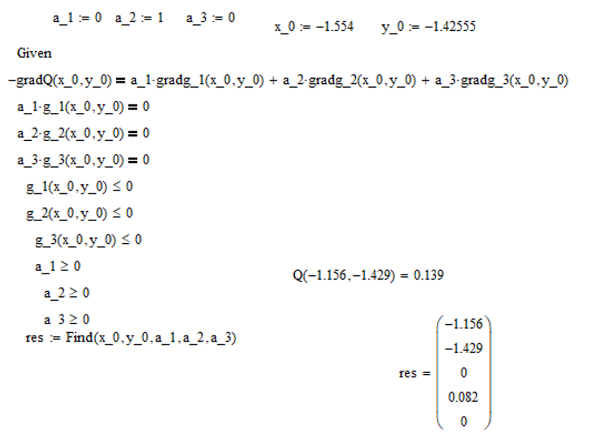
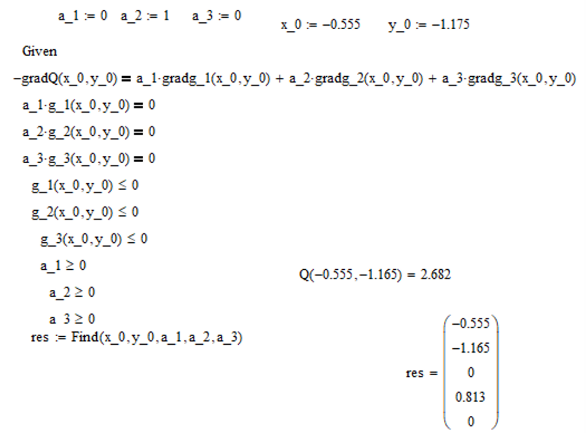
5.2. Проверка выделенных точек на минимум

Таким образом в пункте 5.1 мы выделили следующие группы точек для проверки:

* 1. Желтые
     1. (0.355, 0.8)
     2. (0, 0.8)
     3. (0.555, 1.165)
     4. (1.156, 1.429)
     5. (1, 1)
  2. Эксперимент
     1. (0.6, 0.8)
     2. (1.645, 0.8)

Воспользуемся программными возможностями Mathcad и проверим данные точки на оптимальность. Далее приведены несколько скриншотов, сделанных в программной среде ради демонстрации результата (см. рис. 5.2.1).





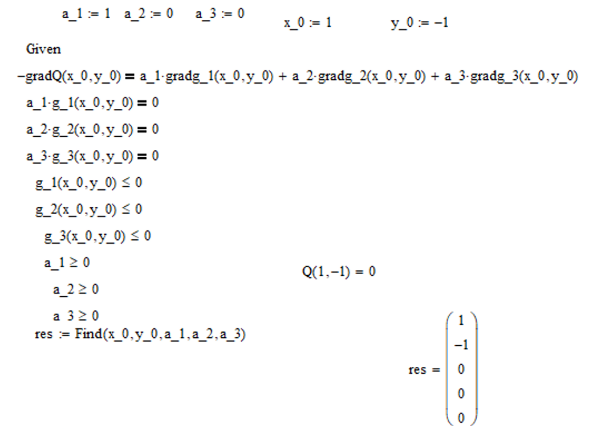
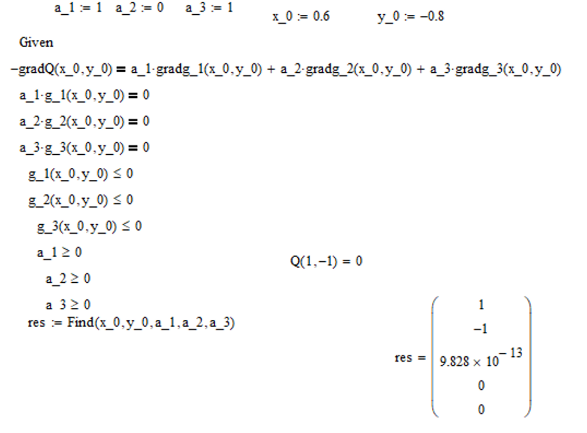


Рис. 5.2.1. Проверка желтых точек (a, b, c, d, e)

Используя решение системы нелинейных уравнений и неравенств в программе Mathcad, можно найти локальный минимум функции. Попытки изменить точки в небольших пределах все равно приводили к верному результату, тогда можно с уверенностью сказать, что все желтые точки являются локальными минимумами. Теперь, проверим точки, выбранные в качестве эксперимента для проверки: точки f, g (см. рис. 5.2.2).



*Рис. 5.2.2. Проверка экспериментальных точек (f)*

К сожалению, скриншот проверки точки *g* не сохранился. Но, даже по примеру проверки точки *f* видно, что Mathcad указывает на другую, ранее выявленную желтую точку локального минимума *e*. Значит, точка *f* точно не является точкой локального минимума.

Теперь можно собрать результаты проверки точек воедино и попытаться найти глобальный минимум. Для этой цели найдем среди желтых точек ту, которая дает наименьшее значение функции . Это точка : . Глобальный ли это минимум? В данной задаче это определенно так, ведь исходя из вида целевой функции: . Значит меньшее значение функция просто не может принимать вовсе. А в силу выпуклости точка – единственный глобальный минимум.

6. Вывод

В данной лабораторной работе была рассмотрена задача динамического программирования теории выпуклых множеств и функций. Построены изображения линий с учетом заданных диапазонов и ограничений. Для определения ориентации векторов выполнено наложение с прозрачным фоном картинки с границей допустимого множества на картину линий уровня функции. Это позволило нам сделать вывод, что заданные функция и множество являются выпуклыми. Для заданного множества условия Слейтера выполняются, следовательно, оно является регулярным. Определили точки локального минимума, проверили их на выполнение условия Каруша-Куна-Таккера для нахождения глобального минимума. Среди точек одна оказалась точкой глобального минимума.

В ходе работы была написана программа на языке программирования Python, приведенная в разделе «приложение». Она реализует построение фазовой диаграммы для различных начальных условий и управляющих параметров. Также для выполнения лабораторной, использовалась программа для решения математических задач Mathcad.

7. Список литературы и источников информации

* Городецкий С. Ю. “Лекции по нелинейному математическому программированию” учебно-методическое пособие

8. Приложение. Код программы и рисунки.

import matplotlib.pyplot as plt

import matplotlib.patches as mpatches

import numpy as np

from turtle import color

from traceback import print\_tb

def Q(x,y):

    return 3\*(y+x\*\*2)\*\*2 + (x\*\*2-1)\*\*2

def dxQ(x, y):

    return 12\*x\*y + 16\*x\*\*3 - 4\*x

def dyQ(x, y):

    return 6\*y + 6\*x\*\*2

def LQ(x, z):

    a = 3

    b = 6\*x\*\*2

    c = 3\*x\*\*4 + (x\*\*2-1)\*\*2-z

    return (-b - np.sqrt(b\*\*2 - 4\*a\*c))/(2\*a)

def RQ(x, z):

    a = 3

    b = 6\*x\*\*2

    c = 3\*x\*\*4 + (x\*\*2-1)\*\*2-z

    return (-b + np.sqrt(b\*\*2 - 4\*a\*c))/(2\*a)

def test(x):

    return (-6\*x\*\*2+np.sqrt(36\*x\*\*4-4\*3\*(3\*x\*\*4+(x\*\*2-1)\*\*2-2)))/(6)

def test1(x):

    return (-6\*x\*\*2-np.sqrt(36\*x\*\*4-4\*3\*(3\*x\*\*4+(x\*\*2-1)\*\*2-2)))/(6)

def g\_1(x, y):

    return y + 0.5\*x + 0.5

def dxg\_1(x, y):

    return 0.5

def dyg\_1(x, y):

    return 1

def g\_2(x, y):

    return -10\*(x+1)\*\*2 - (y-2)\*\*2 + 12

def dxg\_2(x, y):

    return -20\*x-20

def dyg\_2(x, y):

    return -2\*y+4

def g\_3(x, y):

    return y + 0.8

def dxg\_3(x, y):

    return 0

def dyg\_3(x, y):

    return 1

def M(x, y):

    # print(g\_1(x,y))

    a = [max([-0.275, g\_1(x,y)[i], g\_2(x,y)[i], g\_3(x,y)[i]]) for i in range(len(g\_1(x,y)))]

    return a

fig, ax = plt.subplots(1, 1)

rng = 250

feature\_x = np.linspace(-1.8, 1.8, rng)

feature\_y = np.linspace(-2.0, 2.0, rng)

[x, y] = np.meshgrid(feature\_x, feature\_y)

z = g\_3(x, y)

g1 = g\_1(x, y)

g2 = g\_2(x, y)

g3 = g\_3(x, y)

for i in range (rng):

    for j in range (rng):

        z[i][j] = max([-0.175, g1[i][j], g2[i][j], g3[i][j]])

print(z.shape)

# ax.contourf(x, y, z)

con = ax.contour(x, y, z)

ax.clabel(con, fontsize = 10)

ax.set\_title('Filled Contour Plot')

ax.set\_xlabel('feature\_x')

ax.set\_ylabel('feature\_y')

plt.show()

fig, ax = plt.subplots(1, 1)

# x = np.linspace(-2.0, 2, 100000)

# z = [0, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# for i in z:

#     ax.plot(x, LQ(x, i), color="black")

#     ax.plot(x, RQ(x, i), color="black")

feature\_x = np.linspace(-1.8, 1.8, 1000)

feature\_y = np.linspace(-2.0, 1.0, 1000)

[x, y] = np.meshgrid(feature\_x, feature\_y)

z = Q(x, y)

print(x)

print(z)

# ax.contourf(x, y, z)

con = ax.contour(x, y, z)

ax.clabel(con, fontsize = 10)

ax.set\_title('Filled Contour Plot')

ax.set\_xlabel('feature\_x')

ax.set\_ylabel('feature\_y')

plt.show()

fig = plt.figure(figsize=(100, 60))

ax = fig.add\_subplot(1, 1, 1)

rng = 250

feature\_x = np.linspace(-1.8, 1.8, rng)

feature\_y = np.linspace(-2.0, 1.0, rng)

[x, y] = np.meshgrid(feature\_x, feature\_y)

M = g\_3(x, y)

g1 = g\_1(x, y)

g2 = g\_2(x, y)

g3 = g\_3(x, y)

z = Q(x, y)

for i in range(rng):

    for j in range(rng):

        M[i][j] = max([-0.175, g1[i][j], g2[i][j], g3[i][j]])

print(z.shape)

con = ax.contour(x, y, z, levels=[0.1, 0.5, 2, 3, 5, 10, 25])

ax.clabel(con, levels=[0.1, 0.5, 2, 3, 5, 10, 25], fontsize = 50)

con = ax.contourf(x, y, M)

ax.clabel(con, fontsize = 50)

con = ax.contour(x, y, M)

ax.clabel(con,  fontsize = 50)

x = np.array([0.6, -0.355, -1.645])

y = np.array([-0.8, -0.8, -0.8])

dxq = dxQ(x, y)

dyq = dyQ(x, y)

dxg1 = dxg\_1(x, y)

dyg1 = dyg\_1(x, y)

dxg2 = dxg\_2(x, y)

dyg2 = dyg\_2(x, y)

dxg3 = dxg\_3(x, y)

dyg3 = dyg\_3(x, y)

print(dxg1)

alpha = 0.9

ax.plot([x[0], x[0]\*alpha + (1-alpha)\*(x[0] - dxq[0])],

        [y[0], y[0]\*alpha + (1-alpha)\*(y[0] - dyq[0])], color="red")

plt.arrow(x[0], y[0], x[0]\*(alpha-1) + (1-alpha)\*(x[0] - dxq[0]), y[0]\*(alpha-1) +

          (1-alpha)\*(y[0] - dyq[0]), shape='full', lw=0, width = 0.005, head\_width=.01, color="red")

ax.plot([x[0], x[0]\*alpha + (1-alpha)\*(x[0] + dxg1)],

        [y[0], y[0]\*alpha + (1-alpha)\*(y[0] + dyg1)], color="white")

plt.arrow(x[0], y[0], x[0]\*(alpha-1) + (1-alpha)\*(x[0] + dxg1), y[0]\*(alpha-1) +

          (1-alpha)\*(y[0] + dyg1), shape='full', lw=0, width = 0.005, head\_width=.01, color="white")

ax.plot([x[0], x[0]\*alpha + (1-alpha) \* (x[0] + dxg3)],

        [y[0], y[0]\*alpha + (1-alpha)\*(y[0] + dyg3)], color="orange")

plt.arrow(x[0], y[0], x[0]\*(alpha-1) + (1-alpha)\*(x[0] + dxg3), y[0]\*(alpha-1) +

          (1-alpha)\*(y[0] + dyg3), shape='full', lw=0, width = 0.005, head\_width=.01, color="orange")

alpha = 0.97

ax.plot([x[1], x[1]\*alpha + (1-alpha)\*(x[1] - dxq[1])],

        [y[1], y[1]\*alpha + (1-alpha)\*(y[1] - dyq[1])], color="red")

plt.arrow(x[1], y[1], x[1]\*(alpha-1) + (1-alpha)\*(x[1] - dxq[1]), y[1]\*(alpha-1) +

          (1-alpha)\*(y[1] - dyq[1]), shape='full', lw=0, width = 0.005, head\_width=.01, color="red")

ax.plot([x[1], x[1]\*alpha + (1-alpha)\*(x[1] + dxg2[1])],

        [y[1], y[1]\*alpha + (1-alpha)\*(y[1] + dyg2[1])], color="white")

plt.arrow(x[1], y[1], x[1]\*(alpha-1) + (1-alpha)\*(x[1] + dxg2[1]), y[1]\*(alpha-1) +

          (1-alpha)\*(y[1] + dyg2[1]), shape='full', lw=0, width = 0.005, head\_width=.01, color="white")

ax.plot([x[1], x[1]\*alpha + (1-alpha)\*(x[1] + dxg3)],

        [y[1], y[1]\*alpha + (1-alpha)\*(y[1] + dyg3)], color="orange")

plt.arrow(x[1], y[1], x[1]\*(alpha-1) + (1-alpha)\*(x[1] + dxg3), y[1]\*(alpha-1) +

          (1-alpha)\*(y[1] + dyg3), shape='full', lw=0, width = 0.005, head\_width=.01, color="orange")

alpha = 0.99

ax.plot([x[2], x[2]\*alpha + (1-alpha)\*(x[2] - dxq[2])],

        [y[2], y[2]\*alpha + (1-alpha)\*(y[2] - dyq[2])], color="red")

plt.arrow(x[2], y[2], x[2]\*(alpha-1) + (1-alpha)\*(x[2] - dxq[2]), y[2]\*(alpha-1) +

          (1-alpha)\*(y[2] - dyq[2]), shape='full', lw=0, width = 0.005, head\_width=.01, color="red")

ax.plot([x[2], x[2]\*alpha + (1-alpha)\*(x[2] + dxg2[2])],

        [y[2], y[2]\*alpha + (1-alpha)\*(y[2] + dyg2[2])], color="white")

plt.arrow(x[2], y[2], x[2]\*(alpha-1) + (1-alpha)\*(x[2] + dxg2[2]), y[2]\*(alpha-1) +

          (1-alpha)\*(y[2] + dyg2[2]), shape='full', lw=0, width = 0.005, head\_width=.01, color="white")

ax.plot([x[2], x[2]\*alpha + (1-alpha)\*(x[2] + dxg3)],

        [y[2], y[2]\*alpha + (1-alpha)\*(y[2] + dyg3)], color="orange")

plt.arrow(x[2], y[2], x[2]\*(alpha-1) + (1-alpha)\*(x[2] + dxg3), y[2]\*(alpha-1) +

          (1-alpha)\*(y[2] + dyg3), shape='full', lw=0, width = 0.005, head\_width=.01, color="orange")

x, y = -1, -1.5

dxq = dxQ(x, y)

dyq = dyQ(x, y)

dxg2 = dxg\_2(x, y)

dyg2 = dyg\_2(x, y)

alpha = 0.95

ax.plot([x, x\*alpha + (1-alpha)\*(x - dxq)],

        [y, y\*alpha + (1-alpha)\*(y - dyq)], color="red")

plt.arrow(x, y, x\*(alpha-1) + (1-alpha)\*(x - dxq), y\*(alpha-1) +

          (1-alpha)\*(y - dyq), shape='full', lw=0, width = 0.005, head\_width=.01, color="red")

ax.plot([x, x\*alpha + (1-alpha)\*(x + dxg2)],

        [y, y\*alpha + (1-alpha)\*(y + dyg2)], color="white")

plt.arrow(x, y, x\*(alpha-1) + (1-alpha)\*(x + dxg2), y\*(alpha-1) +

          (1-alpha)\*(y + dyg2), shape='full', lw=0, width = 0.005, head\_width=.01, color="white")

ax.set\_title('Нахождение локальных минимумов выпуклой функции', size = 100)

ax.set\_xlabel('X', size = 50)

ax.set\_ylabel('Y', size = 50)

r\_patch = mpatches.Patch(color='red', label=f'$-\\nabla Q$')

g\_patch = mpatches.Patch(color='green', label=f' $\\nabla g\_1$')

w\_patch = mpatches.Patch(color='white', label=f' $\\nabla g\_2$')

o\_patch = mpatches.Patch(color='orange', label=f' $\\nabla g\_3$')

ax.legend(handles=[r\_patch, w\_patch, o\_patch, g\_patch], prop={'size': 50})

plt.tick\_params(axis='both', which='major', labelsize=50)

plt.show()