|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**Лабораторная работа №2**

**«Графический метод решения задачи математического программирования»**

**ДИСЦИПЛИНА: «Моделирование»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-72Б | |  |  | ( | Сафронов Н.С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Никитенко У.В. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

Калуга, 2023

**Цель работы:** изучение математического аппарата математического программирования на примере задач небольшой размерности, допускающих графическое решение.

**Постановка задачи**

**Вариант 14**

Решить задачу нелинейного программирования графическим методом:

при ограничениях

**Листинг программы**

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.colors

from matplotlib.animation import FuncAnimation

import matplotlib.lines

from typing import Union

def first\_condition(

xs: Union[np.array, float], ys: Union[np.array, float]

) -> Union[np.array, float]:

return (xs - 2) \* (ys + 1)

def second\_condition(

xs: Union[np.array, float], ys: Union[np.array, float]

) -> Union[np.array, float]:

return xs + ys

def first\_condition\_inequality(

xs: Union[np.array, float], ys: Union[np.array, float]

) -> Union[np.array, float]:

return first\_condition(xs, ys) >= 4

def second\_condition\_inequality(

xs: Union[np.array, float], ys: Union[np.array, float]

) -> Union[np.array, float]:

return second\_condition(xs, ys) <= 6

class Animation:

def \_\_init\_\_(self, xs, ys, line, points, min\_label, max\_label, root\_label):

self.\_xs = xs

self.\_ys = ys

self.\_line = line

self.\_points = points

self.\_min\_label = min\_label

self.\_max\_label = max\_label

self.\_root\_label = root\_label

self.\_previous\_points = []

self.\_pause = False

self.\_previous\_root = 5

def to\_tuple(self) -> tuple:

return (

self.\_line,

self.\_points,

self.\_min\_label,

self.\_max\_label,

self.\_root\_label,

)

def get\_frames(self):

MAX\_ROOT = 10

STEP = 0.01

while self.\_previous\_root < MAX\_ROOT:

if not self.\_pause:

self.\_previous\_root += STEP

yield self.\_previous\_root

def on\_click(self, event):

self.\_pause ^= True

def get\_animation\_frame(self, z: float) -> tuple:

y = (z - self.\_xs) / 2

self.\_line.set\_data(self.\_xs, y)

self.\_root\_label.set\_text(f"z = {z:.2f}")

intersections = {}

y\_1 = ((z / 2) + np.sqrt(z\*\*2 / 4 - 8)) / 2 - 1

x\_1 = z - 2 \* y\_1

if (

abs(first\_condition(x\_1, y\_1) - 4) <= 1e-6

and second\_condition(x\_1, y\_1) <= 6

):

intersections[x\_1] = y\_1

y\_2 = z - 6

x\_2 = z - 2 \* y\_2

if (

abs(second\_condition(x\_2, y\_2) - 6) <= 1e-6

and first\_condition(x\_2, y\_2) >= 4

):

intersections[x\_2] = y\_2

if len(self.\_previous\_points) == 0 and len(intersections) > 0:

xs, ys = ([\*intersections.keys()], [\*intersections.values()])

min\_label.set\_text(f"min $z = $f({xs[0]:.2f}, {ys[0]:.2f}) = {z:.2f}")

self.\_pause = True

elif len(self.\_previous\_points) > 0 and len(intersections) == 0:

xs, ys = self.\_previous\_points

z = xs[0] + 2 \* ys[0]

max\_label.set\_text(f"max $z = $f({xs[0]:.2f}, {ys[0]:.2f}) = {z:.2f}")

self.\_pause = True

self.\_points.set\_data([\*intersections.keys()], [\*intersections.values()])

if len(intersections) > 0:

self.\_previous\_points = ([\*intersections.keys()], [\*intersections.values()])

else:

self.\_previous\_points = []

return self.to\_tuple()

if \_\_name\_\_ == "\_\_main\_\_":

x\_range = np.linspace(0, 7, 250)

y\_range = np.linspace(0, 4, 250)

xs, ys = np.meshgrid(x\_range, y\_range)

figure, axis = plt.subplots()

matplotlib.pyplot.contour(xs, ys, (first\_condition(xs, ys) - 4), [0], colors="k")

matplotlib.pyplot.contour(xs, ys, (second\_condition(xs, ys) - 6), [0], colors="k")

first\_region = first\_condition\_inequality(xs, ys)

second\_region = second\_condition\_inequality(xs, ys)

extent = (xs.min(), xs.max(), ys.min(), ys.max())

plt.imshow(

first\_region.astype(int), extent=extent, origin="lower", cmap="Blues", alpha=0.5

)

plt.imshow(

second\_region.astype(int),

extent=extent,

origin="lower",

cmap="Greens",

alpha=0.5,

)

line = axis.plot([], [], "r")[0]

points = axis.plot([], [], "ro")[0]

min\_label = axis.text(0.1, 0.3, "", fontsize=10)

max\_label = axis.text(0.1, 0.1, "", fontsize=10)

root\_label = axis.text(0.1, 0.5, "", fontsize=10)

animation = Animation(xs, ys, line, points, min\_label, max\_label, root\_label)

figure.canvas.mpl\_connect('button\_press\_event', animation.on\_click)

ani = FuncAnimation(

figure,

animation.get\_animation\_frame,

frames=animation.get\_frames,

init\_func=animation.to\_tuple,

blit=True,

interval=10,

)

green\_legend\_handle = matplotlib.lines.Line2D(

[], [], color="green", marker="s", ls="",

label="$x\_1$ + $x\_2$ <= 6"

)

blue\_legend\_handle = matplotlib.lines.Line2D(

[], [], color="blue", marker="s", ls="",

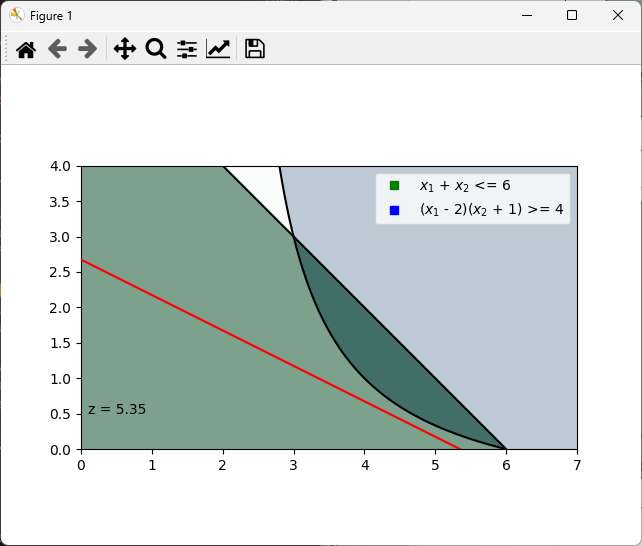
label="($x\_1$ - 2)($x\_2$ + 1) >= 4"

)

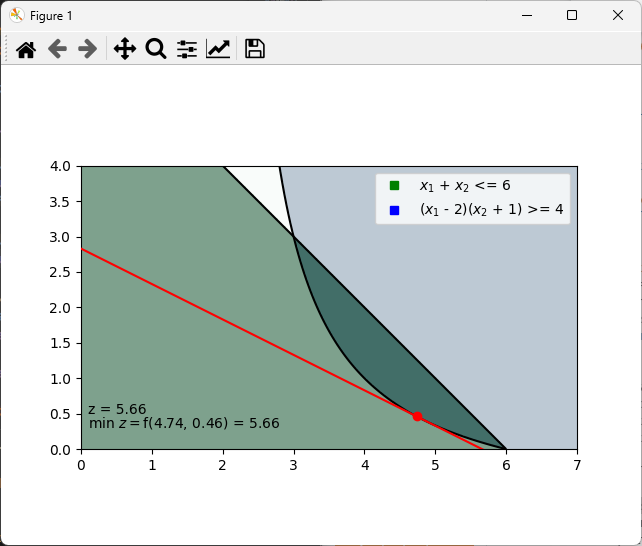
plt.legend(handles=[green\_legend\_handle, blue\_legend\_handle])

plt.show()

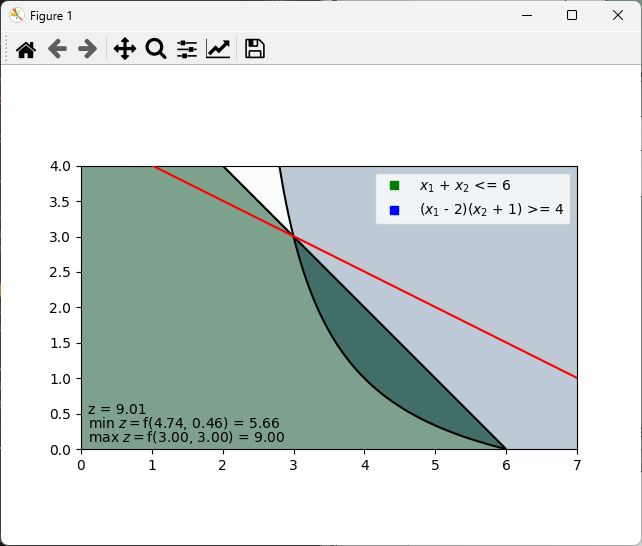
**Результаты выполнения программы**



**Рисунок 1 –** Анимация нахождения решения графическим методом в начальный момент



**Рисунок 2 –** Найденная точка минимума



**Рисунок 3 –** Найденная точка максимума

**Вывод:** в ходе выполнения лабораторной работы был изучен математический аппарат математического программирования на примере задач небольшой размерности, допускающих графическое решение.