МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС
"ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ"
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

ФАКУЛЬТЕТ СИСТЕМНИХ ДОСЛІДЖЕНЬ

КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ

З КУРСУ "ЧИСЕЛЬНІ МЕТОДИ ОПТИМІЗАЦІЇ"

ЛАБОРАТОРНА РОБОТА №1

Тема: **«Чисельні методи безумовної оптимізації першого порядку. Градієнтний метод та його варіації»**

Виконала: студентка 3-го курсу
групи КА-41
Лочман Я.В.

Київ – 2017

ЗАВДАННЯ (ВАРІАНТ 6)

Мінімізувати функцію одним з градієнтних методів.

$$f(x, y) = x^2 + 4y^2 + 0.001xy - y$$

КОД ПРОГРАМИ

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm

### FUNCTION INITIALISATION
def f(x):
    return 100.0*(x[1]-x[0]**3)**2 + 100.0 * (1 - x[0])**2
def f2(x):
    return x[0]**2 + 4.0 * x[1]**2 + 0.001 * x[0] * x[1] - x[1]
f_type = 'nonquadratic'
if (f_type == 'quadratic'):
    A = np.array([[2, 0.001], [0.001, 8]], dtype = float)
    b = np.array([0, -1], dtype = float)

### SETTINGS
eps = 10**(-5)
h = np.array([0.01,0.01], dtype = float)
x0 = np.array([-2,-2], dtype = float)

### APPROXIMATE GRADIENT
def df1(x,h1):
    return (f([x[0] + h1, x[1]]) - f([x[0] - h1, x[1]])) / (2 * h1)
def df2(x,h2):
    return (f([x[0], x[1] + h2]) - f([x[0], x[1] - h2])) / (2 * h2)
def df(x,h):
    return np.array([df1(x,h[0]), df2(x,h[1])])

### TRUE GRADIENT (ONLY FOR QUADRATIC FUNCTIONS)
def truedf(x):
    return A.dot(x)+b

### METHODS OF CHOOSING ALPHA
def choose_alpha_method(x,h,f_type='none'):
    if (f_type == 'quadratic'):
        return alpha_quadr(x,h)
    else:
        return alpha_split(x,h)

def alpha_quadr(x,h):
    return -np.dot((A.dot(x)+b),h)/np.dot(A.dot(h),h)

def alpha_split(x,h,b=1,l=0.5):
    alpha = b
    q = 0.1
    while (f(x+alpha*h)-f(x)>=0):
    #while (f(x+alpha*h)-f(x)>q*alpha*df(x).dot(h)):
        alpha *= l
    return alpha

### STOP CONDITIONS
```

```
def stop1(x1,x2,k):
    plt.ylabel('|| x_new - x_old || ')
    d = norm(x2-x1, ord=2)
    plt.scatter(k, d)
    return d<=eps

def stop2(x1,x2,k):
    plt.ylabel('| f(x_new) - f(x_old) | ')
    d = abs(f(x2)-f(x1))
    plt.scatter(k, d)
    return d<=eps

def stop3(x,h,k):
    plt.ylabel('|| f\'(x_new) || ')
    d = norm(df(x,h), ord=2)
    plt.scatter(k, d)
    return d<=eps

### ACTUALLY THE GRADIENT METHOD
def gradient_method(x0,h):
    fout = open('output.txt', 'w')
    fout.write('The initial point is ({x}, {y})\n\n'.format(x=x0[0],
y=x0[1]))
    x_new = x0
    k = 0
    plt.xlabel('iteration')
    while True:
        x_old = x_new
        step = - df(x_old,h)
        alpha = choose_alpha_method(x_old,step,f_type)
        x_new = x_old + alpha * step
        k += 1
        fout.write('{iter:>3}. alpha = {al:<17.15f},   x_{iter:<3} =
({x:>18.15f}, {y:>18.15f})\n'.format(iter=k, x=x_new[0], y=x_new[1],
al=alpha))
        if (stop1(x_old,x_new,k)):
            break
    print('Gradient method found approximate solution in {}
iterations'.format(k))
    fout.write('\nThe approximate solution of the problem is ({x:>10.7f},
{y:>10.7f})\n'.format(x=x_new[0], y=x_new[1]))
    fout.write('The value of function in this point is
{v:>10.7f}\n'.format(v=f(x_new)))
    fout.close()
    return x_new

### PROGRAM
minim = gradient_method(x0,h)
print('The initial point is ({x}, {y})'.format(x=x0[0], y=x0[1]))
print('The approximate solution of the problem is ({x:>10.7f},
{y:>10.7f})'.format(x=minim[0], y=minim[1]))
print('The value of function in this point is
{v:>10.7f}'.format(v=f(minim)))
plt.show()
```

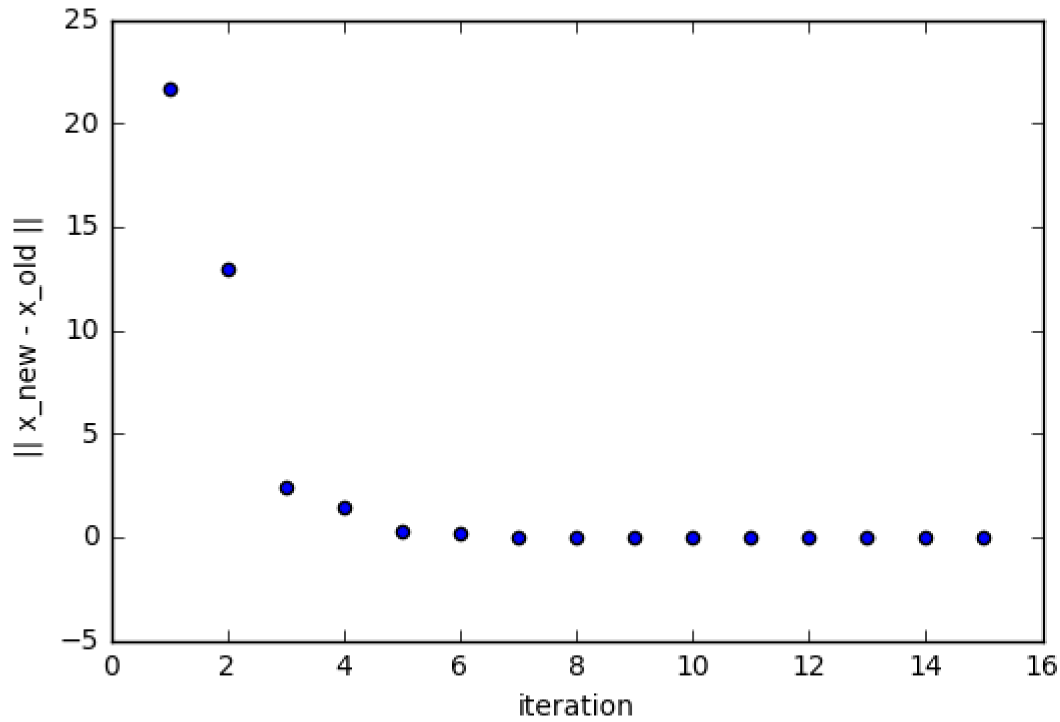
РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ

```
// реалізовано для h = (0.01, 0.01); eps = 10**(-5); x0 = (-20, -20)

output.txt:
The initial point is (-20.0, -20.0)

  1. alpha = 0.130695092502509,   x_1   = (-14.769582398047573,
1.044523794757971)
  2. alpha = 0.425784976838707,   x_2   = (-2.192694542047668,
-2.081342880172009)
  3. alpha = 0.130695092502540,   x_3   = (-1.619273688741934,
0.225809188775126)
  4. alpha = 0.425784976838686,   x_4   = (-0.240445014789193,
-0.116885653676767)
  5. alpha = 0.130695092502536,   x_5   = (-0.177579771508579,
0.136051914365254)
  6. alpha = 0.425784976838685,   x_6   = (-0.026416102572187,
0.098481612347890)
  7. alpha = 0.130695092502536,   x_7   = (-0.019524063697166,
0.126211649820544)
  8. alpha = 0.425784976838690,   x_8   = (-0.002951696703372,
0.122092744547337)
  9. alpha = 0.130695092502533,   x_9   = (-0.002196109078541,
0.125132842482125)
 10. alpha = 0.425784976838610,   x_10  = (-0.000379248256696,
0.124681278886980)
 11. alpha = 0.130695092502546,   x_11  = (-0.000296411715993,
0.125014570735655)
 12. alpha = 0.425784976839482,   x_12  = (-0.000097225730840,
0.124965064940563)
 13. alpha = 0.130695092502221,   x_13  = (-0.000088144199791,
0.125001604374086)
 14. alpha = 0.425784976832830,   x_14  = (-0.000066307052882,
0.124996176957496)
 15. alpha = 0.130695092501653,   x_15  = (-0.000065311426972,
0.125000182846652)

The approximate solution of the problem is (-0.0000653,  0.1250002)
The value of function in this point is -0.0625000
```



ВИСНОВКИ

Під час виконання лабораторної роботи, я реалізувала градієнтний метод розв’язку оптимізаційної задачі. А точніше дві його варіації: метод найскорішого спуску в разі, якщо цільова функція є квадратичною (матриця А — додатньо-визначена); і разом з методом подрібнення кроку для вибору альфа для будь-яких функцій. Перший метод досить швидко знаходить точку мінімуму (3 — 15 ітерацій для даної функції, залежно від початкової точки). Останній, на жаль, обов’язково знаходить лише стаціонарну точку, що не завжди може бути розв’язком опитимізаційної задачі. Також дослід показав, що для так званих ярних функцій цей метод є дуже повільним (більше тисячі ітерацій, залежно від початкової точки)

Також побудувала графік, який наглядно показує, як швидко змінюється норма різниці x^k та x^{k+1} на перших ітераціях, і як повільно — на останніх. Також є можливість вибрати іншу з трьох умов зупинки.