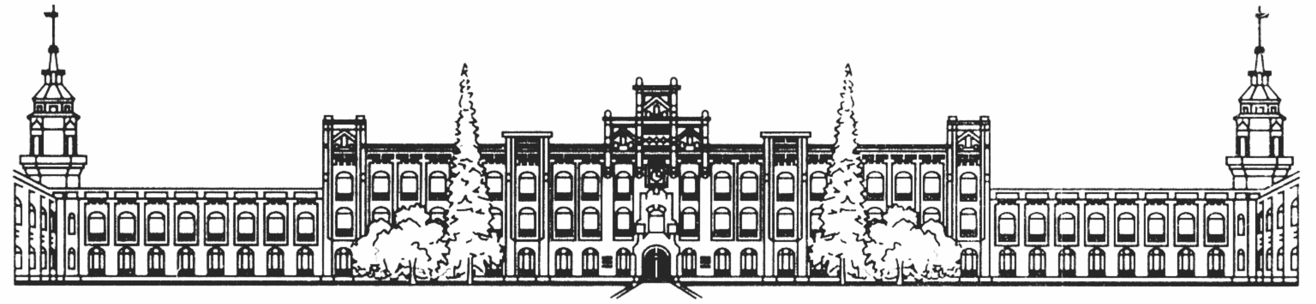


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»
Інститут прикладного системного аналізу

Лабораторна робота
з курсу «Методи оптимізації»
з теми «Чисельні методи безумовної оптимізації першого порядку.
Градієнтний метод та його варіації»

Виконали студенти 3 курсу групи КА-81
Галганов Олексій
Єрко Андрій
Фордуй Нікіта

Перевірили
Спекторський Ігор Якович
Яковлева Алла Петрівна

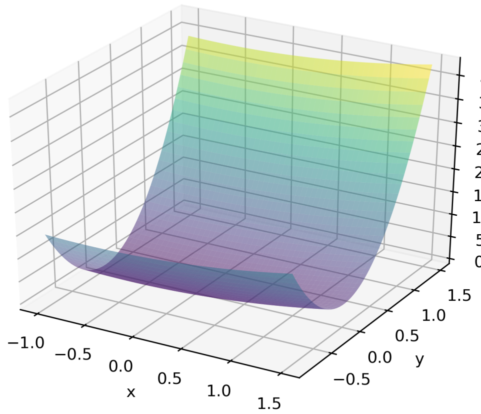
Київ 2021

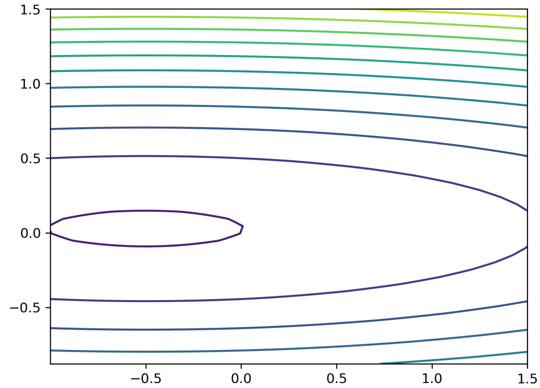
Варіант 1

Завдання. Скласти програму для мінімізації цільової функції одним з градієнтних методів. Конкретний тип градієнтного методу обрати самостійно, враховуючи особливості цільової функції.

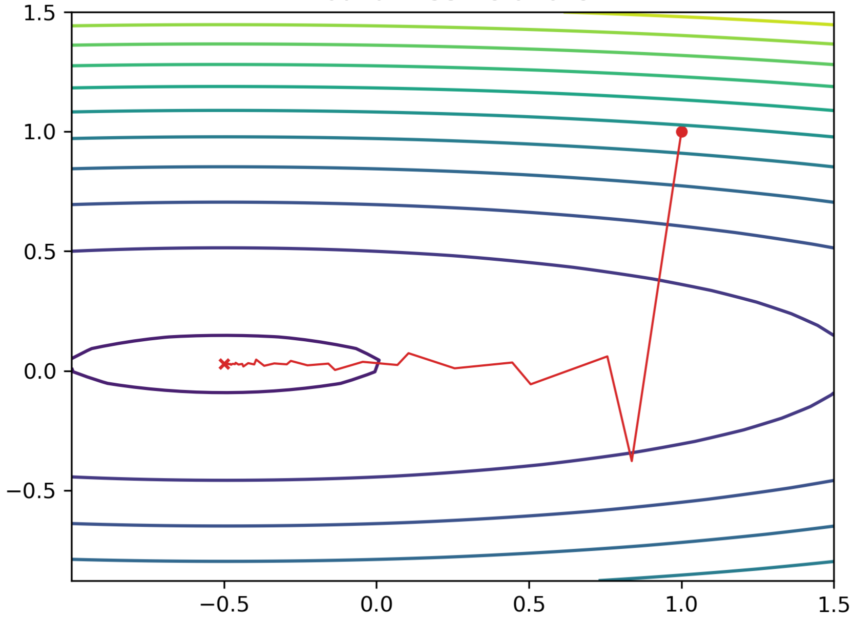
Цільова функція: $f(x, y) = x^2 + 18y^2 + 0.01xy + x - y$

Результати роботи. Для визначення ярності цільової функції розглянемо її матрицю Гессе $f''(x, y) = \begin{pmatrix} 2 & 0.01 \\ 0.01 & 36 \end{pmatrix}$. Її власні числа приблизно рівні 2 та 36, тому матриця є погано обумовленою. Також це видно з графіку цільової функції та її ліній рівня.





Отже, функція має «ярну» структуру, тому для прискорення збіжності необхідно застосувати якусь модифікацію градієнтного методу. Для реалізації було обрано метод дроблення кроку, який має допомогти зменшити зигзагоподібний характер шляху до точки мінімуму. Обраний критерій зупинки — $|f(x_{k+1}) - f(x_k)| < \varepsilon = 10^{-5}$. Було отримано збіжність до точки мінімуму за 55 ітерацій.



Лістинг. Текст програми було розділено на `Optimizer.py` з реалізацією власне градієнтного методу та `lab1.py`, де він викликається та зберігаються результати.

`Optimizer.py`

```
import numpy as np

class GDOptimizer:
    def __init__(self, beta = 0.1, lmb = 0.5, tol = 1e-5, max_iter = 100):
        """
        Parameteres
        -----------
        beta, lmb : float
                    parameteres of step decay,
                    beta - initial step (default 0.1),
                    lmb - decay rate (default 0.5)
        max_iter  : int
                    maximum number of iterations,
                    default is 100
        tol       : tolerance for algorithm stopping
                    |f(x_prev) - f(x_new)| < tol
        """
        self.beta = beta
        self.lmb = lmb
        self.tol = tol
        self.max_iter = int(max_iter)

    def minimize(self, target_func, x0, h = 0.005):
        """
        Parameteres
        -----------
        target_func : callable
                      function to minimize
        x0          : np.array of shape (n, )
                      initial point
        h           : step for computing gradients,
                      default is 0.005
        -----------
        Returns history - np.array with shape (n_iter, n+1), n - number of
    variables;
                history[:, -1] - values of target functions
                history[-1, :-1] - solution
                history[-1, -1] - value of target function at solution point
        """
        def compute_grad(target_func, x, h):
            n = len(x)
            grad = np.zeros_like(x)
            for i in range(n):
                x_plus = np.copy(x)
                x_plus[i] += h
                x_minus = np.copy(x)
                x_minus[i] -= h
                grad[i] = (target_func(x_plus) - target_func(x_minus))/(2*h)
            return grad

        history = []
        x = np.copy(x0)
        history.append([*x0, target_func(x0)])
        for k in range(self.max_iter):
            grad = compute_grad(target_func, x, h)
            alpha = self.beta
            while target_func(x - alpha*grad) >= target_func(x):
                alpha = alpha*self.lmb
```

```
            x = x - alpha*grad
            history.append([*x, target_func(x)])
            if k > 0 and np.abs(history[-1][1] - history[-2][1]) < self.tol:
                break
        return np.array(history)
```

lab1.py

```
import numpy as np
from Optimizer import GDOptimizer
from Plotter import PlotContour, PlotSurface

# deifine target function
def f(x_vect):
    x, y = x_vect[0], x_vect[1]
    return x**2 + 18*y**2 + 0.01*x*y + x - y

# set optimizer parameteres and create it
opt_params = {
    'beta' : 0.1,
    'lmb' : 0.3,
    'tol' : 1e-5,
    'max_iter' : 100
}
opt = GDOptimizer(**opt_params)

# initial point
x0 = np.array([1, 1])
hist = opt.minimize(f, x0)

# save results
x_min, x_max = np.min(hist[:, 0])-0.5, np.max(hist[:, 0])+0.5
y_min, y_max = np.min(hist[:, 1])-0.5, np.max(hist[:, 1])+0.5
PlotContour((x_min, x_max), (y_min, y_max), f, fname = '../latex/pics/
    contour_init.png')
PlotSurface((x_min, x_max), (y_min, y_max), f, fname = '../latex/pics/
    surface_init.png')
PlotContour((x_min, x_max), (y_min, y_max), f, hist, fname = '../latex/pics/
    contour_final.png')
PlotSurface((x_min, x_max), (y_min, y_max), f, hist, fname = '../latex/pics/
    surface_final.png')
```

Висновки. При виконанні даної роботи ми дослідили застосування градієнтного методу до мінімізації заданої квадратичної функції, що має ярну структуру. Для збільшення швидкості збіжності було реалізовано метод дроблення кроку, який на кожній ітерації обирає крок α_k найбільшим серед тих, які забезпечують рух у напрямку мінімуму.