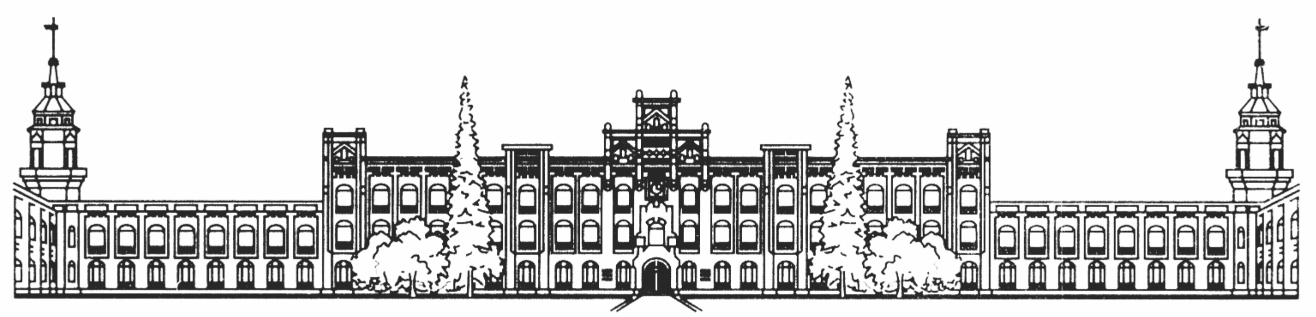


Міністерство освіти і науки України

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

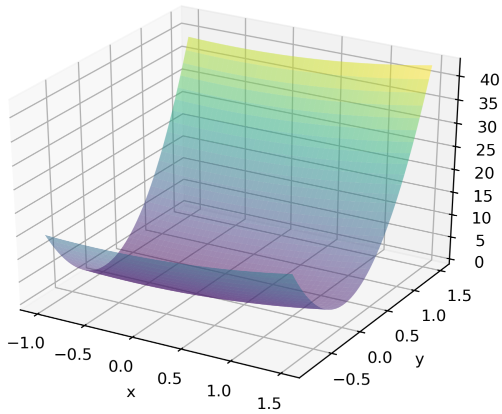
Київ 2021

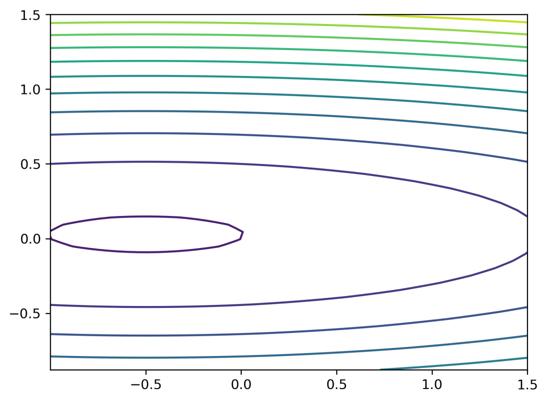
Варіант 1

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

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

Результати роботи. Для визначення опуклості 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 цільова функція є строго опуклою. До того ж, відношення власних чисел рівне 1/18, тому лінії рівня функції будуть видовженими еліпсами. Також це видно з графіку цільової функції та її ліній рівня.





Знайдемо точку мінімуму аналітично:

$$\begin{cases} f'_x(x, y) = 2x + 0.01y + 1 = 0 \\ f'_y(x, y) = 36y + 0.01x - 1 = 0 \end{cases} \Leftrightarrow \begin{pmatrix} 2 & 0.01 \\ 0.01 & 36 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 2 & 0.01 \\ 0.01 & 36 \end{pmatrix}, \det A = 72 - 10^{-4} = 71.9999, A^{-1} = \frac{1}{\det A} \begin{pmatrix} 36 & -0.01 \\ -0.01 & 2 \end{pmatrix}$$

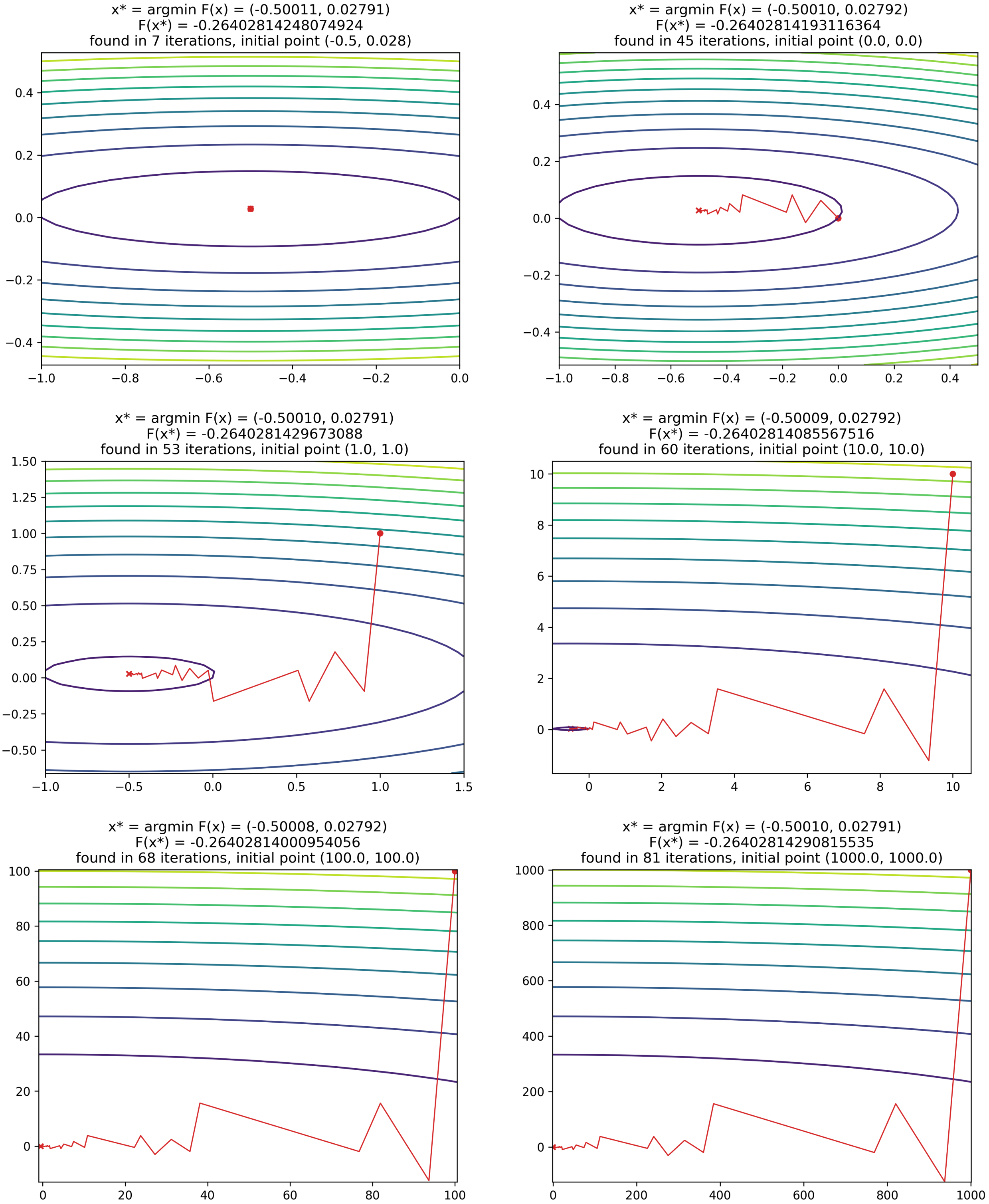
$$\begin{pmatrix} x \\ y \end{pmatrix} = A^{-1} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \frac{1}{71.9999} \begin{pmatrix} -36.01 \\ 2.01 \end{pmatrix} \approx \begin{pmatrix} -0.50014 \\ 0.0279167 \end{pmatrix}$$

Для чисельного розв'язання задачі реалізовано метод дроблення кроку та метод найшвидшого спуску для квадратичної функції.

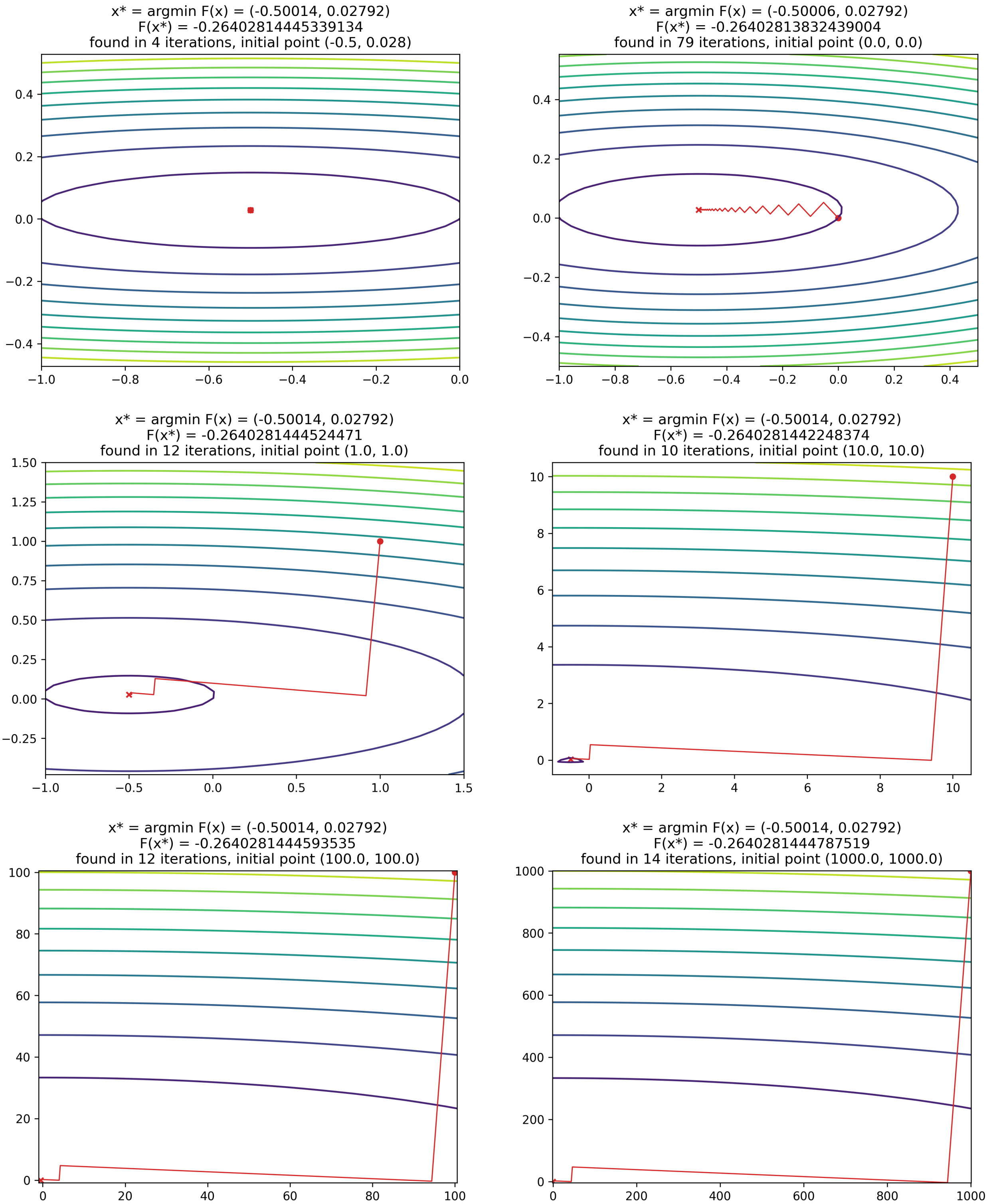
	кількість ітерацій	
x_0	метод дроблення кроку	метод найшвидшого спуску
$(-0.5, 0.0028)$	7	2
$(0, 0)$	45	79
$(1, 1)$	53	12
$(10, 10)$	60	10
$(100, 100)$	68	12
$(1000, 1000)$	81	14

Відповідні графіки з траєкторіями руху методів до точки мінімуму:

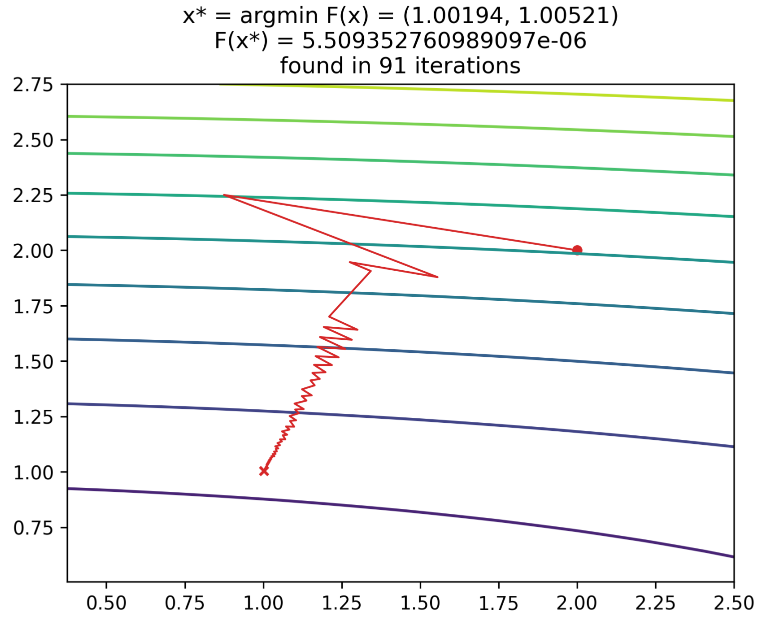
Метод дроблення кроку

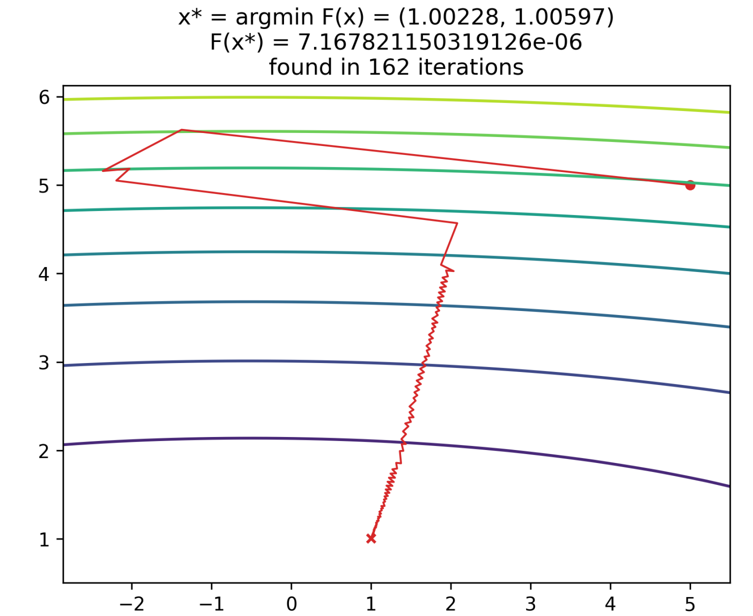


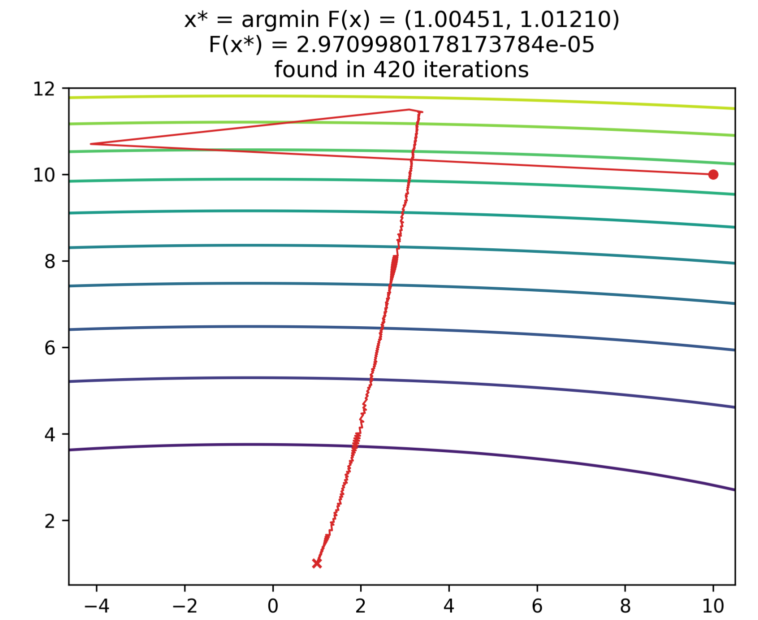
Метод найшвидшого спуску



Перевіримо роботу градієнтного методу з дробленням кроку на тестовій функції $f(x,y) = (y - x^2)^2 + (1 - x)^2$, що має мінімум в точці $(1, 1)$.







Лістинг. Текст програми було розділено на `Optimizer.py` з реалізацією власне градієнтного методу та `lab1.py`, де задаються цільові функції.
`Optimizer.py`

```
import numpy as np

class GDOptimizer:
    def __init__(self, tol = 1e-5, max_iter = 100):
        """
        Parameters
        -----------
        max_iter  : int
                    maximum number of iterations,
                    default is 100
        tol       : tolerance for algorithm stopping
                    |f(x_prev) - f(x_new)| < tol
        """
        self.tol = tol
        self.max_iter = int(max_iter)

    def minimize(self, target_func, x0, beta = 0.1, lmb = 0.5, grad_check =
    False, h = 0.005):
        """
        Minimize arbitrary function with given initial point
        Parameters
        -----------
        target_func : callable
                      function to minimize
        x0          : np.array of shape (n, )
                      initial point
        beta, lmb   : float
                      parameters of step decay,
                      beta - initial step (default 0.1),
                      lmb - decay rate (default 0.5)
        grad_check  : bool
                      whether to check ||grad||^2 < tol or not
        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).astype(float)
            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).astype(float)
        history.append([*x, target_func(x)])
        for k in range(self.max_iter):
            grad = compute_grad(target_func, x, h)
            if grad_check and np.linalg.norm(grad)**2 < self.tol:
                break
            alpha = beta
```

```
            while target_func(x - alpha*grad) >= target_func(x):
                alpha = alpha*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)

    def minimizeQuad(self, A, b, x0):
        """
        Minimize quadratic function (Ax, x) + (b, x)
        with given initial point
        Parameters
        -----------
        A, b     : np.array
                   parameters of target function
        x0       : np.array of shape (n, )
                   initial point
        -----------
        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
        """
        target_func = lambda x: np.dot(A @ x, x) + np.dot(b, x)
        compute_grad = lambda x: 2*(A @ x) + b
        history = []
        x = np.copy(x0).astype(float)
        history.append([*x, target_func(x)])
        for k in range(self.max_iter):
            grad = compute_grad(x)
            alpha = np.dot(2*A @ x + b, grad)/(2*np.dot(A @ grad, grad))
            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

# define 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

def Rosenbrok(x_vect):
    x, y = x_vect[0], x_vect[1]
    return (y-x**2)**2 + (1-x)**2


A = np.array([[1, 0.005], [0.005, 18]])
b = np.array([1, -1])

# set optimizer parameteres and create it
opt_params = {
    'tol': 1e-5,
    'max_iter': 1000
}
opt = GDOptimizer(**opt_params)
```

```
# initial point
x0 = np.array([1, 1])
#hist = opt.minimize(Rosenbrok, x0, beta = 1, lmb = 0.5)
hist = opt.minimize(f, x0, beta = 1, lmb = 0.5)
#hist = opt.minimizeQuad(A, b, x0)

# pictures
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=f'./latex/pics/
    result_{x0[0], x0[1]}_frac.png')
```

Висновки. При виконанні даної роботи ми дослідили застосування градієнтного методу до мінімізації заданої квадратичної функції. Для пришвидшення збіжності було реалізовано метод дроблення кроку та метод найшвидшого спуску. Також було порівняно кількість ітерацій для збіжності методу з різних початкових точок. Метод найшвидшого спуску, цілком очікувано, збігався за меншу кількість ітерацій, ніж метод дроблення кроку.