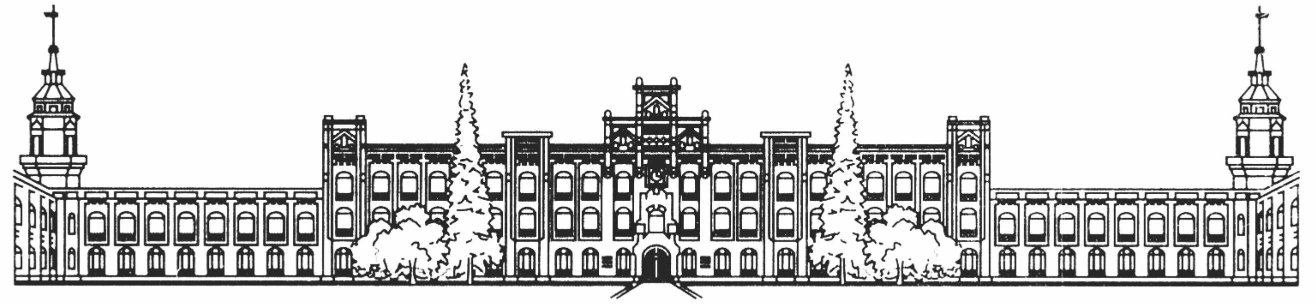


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»
Інститут прикладного системного аналізу

Лабораторна робота
з курсу «Методи оптимізації»
з теми «Методи спряжених градієнтів>

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

Перевірили
Спекторський Ігор Якович
Яковлева Алла Петрівна

Київ 2021

Варіант 1

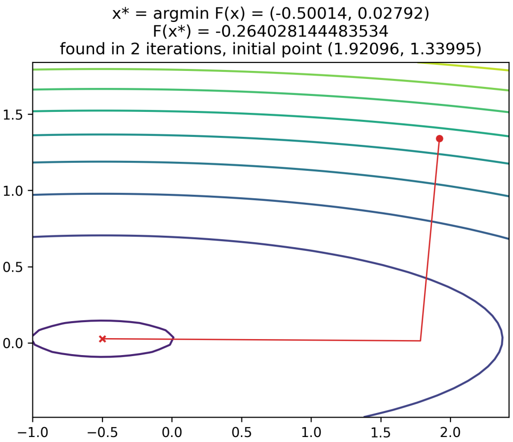
Завдання. Скласти програму для мінімізації цільової функції методом спряжених градієнтів.

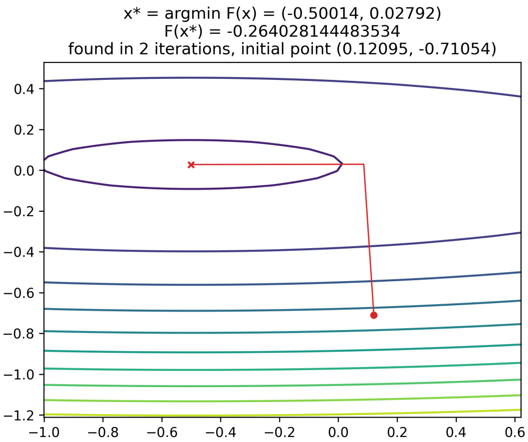
Цільова функція: $f(x,y) = x^2 + 18y^2 + 0.01xy + x - y$

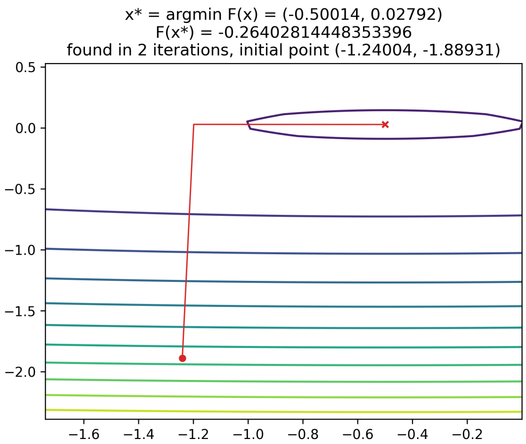
Результати роботи. Цільова функція є квадратичною:

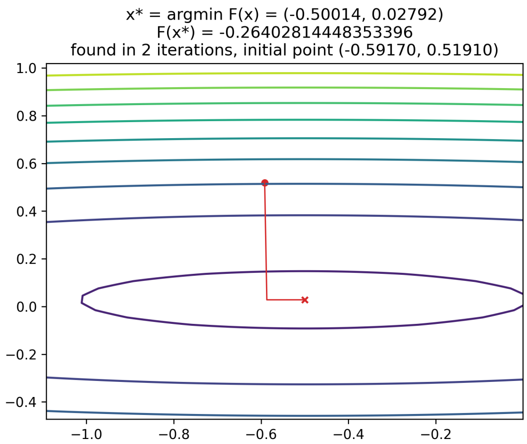
$$f(x,y) = \frac{1}{2}\left\langle \begin{pmatrix} 2 & 0.01 \\ 0.01 & 36 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \right\rangle + \left\langle \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \right\rangle$$

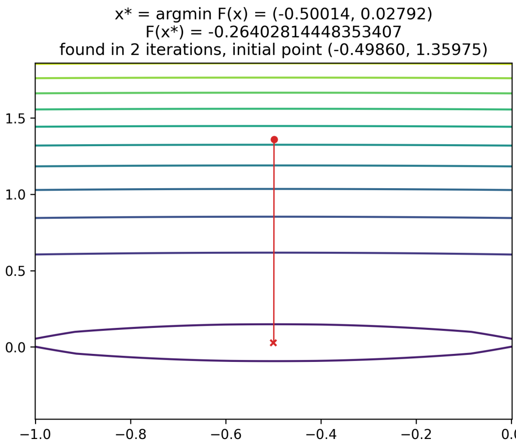
Мінімум — у точці $\begin{pmatrix} -0.50014 \\ 0.0279167 \end{pmatrix}$. Метод спряжених напрямків має мінімізувати її за не більше ніж 2 кроки. Дійсно, при запуску методу з випадкових початкових точок отримали точку мінімуму за два кроки.

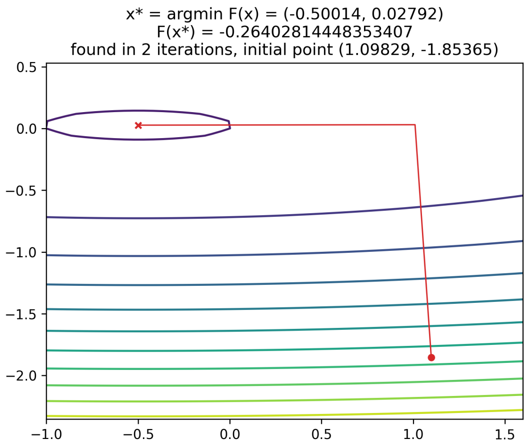




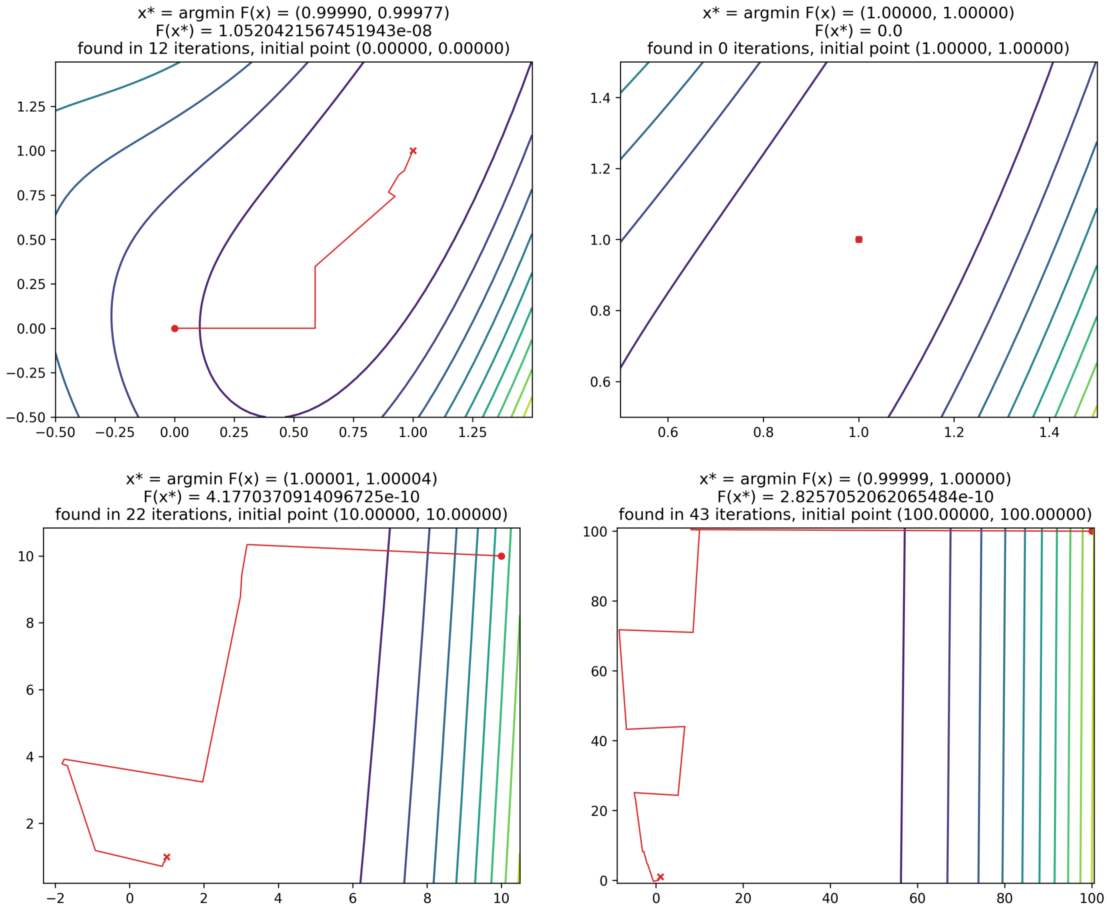




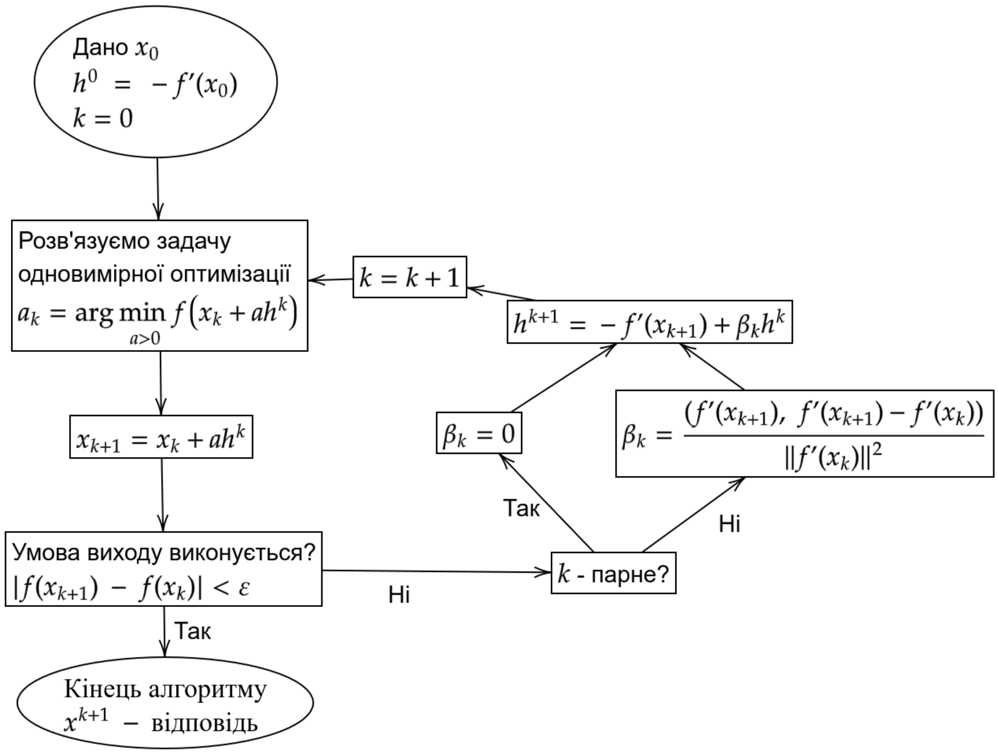




Ми також вирішили перевірити цей метод на функції Розенброка $f(x, y) = (y - x^2)^2 + (1 - x)^2$, яка вже не є квадратичною. Вона має мінімум у точці $(1, 1)$. Перевірку зробили з чотирьох початкових точок.



Блок-схема алгоритму (для неквадратичної функції).



Лістинг. Текст програми було розділено на `Optimizer.py` з реалізацією власне методу спряжених градієнтів для випадку квадратичних та неквадратичних функцій, а також методу дихотомії для розв'язку задачі одновимірної оптимізації та `lab4.py`, де викликаються необхідні функції та зберігаються результати.
`Optimizer.py`

```python
import numpy as np

def dichotomy(f, a, b, delta, tol=1e-5, max_iter=100):
    for _ in range(max_iter):
        d = (b - a)/2 * delta
        x1 = (a + b)/2 - d
        x2 = (a + b)/2 + d
        if f(x1) < f(x2):
            b = x2
        else:
            a = x1
        if abs(b-a) < 2*tol:
            break
    x = (a+b)/2
    return x


def QuadraticCG(A, b):
    """
    Use conjugate gradients method to minimize
    1/2 * (Ax, x) + (b, x)
    A is symmetric non-negative defined n*n matrix,
    b is n-dimensional vector
    """
    def target(x):
        return 1/2 * np.dot(A @ x, x) + np.dot(b, x)
    def grad(x):
        return A @ x + b
    x = np.random.randn(len(b))
    history = []
    history.append([*x, target(x)])
    r, h = -grad(x), -grad(x)
    for _ in range(1, len(b)+1):
        alpha = np.linalg.norm(r)**2/np.dot(A @ h, h)
        x = x + alpha * h
        history.append([*x, target(x)])
        beta = np.linalg.norm(r - alpha*(A @ h))**2/np.linalg.norm(r)**2
        r = r - alpha*(A @ h)
        h = r + beta*h
    return np.array(history)


def ConjugateGradient(target_func, x0, renewal=True, grad_check=False, d=0.005,
    tol=1e-5, max_iter=100):
    """
    Minimize arbitrary function using conjugate gradients method
    Parameters
    -----------
    target_func : callable
                  function to minimize
    x0          : np.array of shape (n, )
                  initial point
    renewal     : bool
                  whether to make parameters renewal or not
                  default is True
    grad_check  : bool
                  whether to check ||grad||^2 < tol or not
```

```
    d            : step for computing gradients,
                   default is 0.005
    max_iter     : int
                   maximum number of iterations,
                   default is 100
    tol          : tolerance for algorithm stopping
                   |f(x_prev) - f(x_new)| < tol
    -----------
    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, d):
            n = len(x)
            grad = np.zeros_like(x)
            for i in range(n):
                x_plus = np.copy(x)
                x_plus[i] += d
                x_minus = np.copy(x)
                x_minus[i] -= d
                grad[i] = (target_func(x_plus) - target_func(x_minus))/(2*d)
            return grad

    history = []
    x = x0.copy().astype('float')
    history.append([*x, target_func(x)])

    grad = compute_grad(target_func, x, d)
    if grad_check and np.linalg.norm(grad)**2 < tol:
        return np.array(history)
    h = -grad
    for k in range(max_iter):
        old_grad = grad

        alpha = dichotomy(lambda a: target_func(x + a*h), 0, 1, 0.001)
        x = x + alpha*h
        history.append([*x, target_func(x)])

        grad = compute_grad(target_func, x, d)
        if renewal and not k % 2:
            beta = 0
        else:
            beta = (grad @ (grad - old_grad)) / np.linalg.norm(old_grad)**2
        h = -grad + beta * h

        if k > 0 and np.abs(history[-1][1] - history[-2][1]) < tol:
            break

    return np.array(history)
```

lab4.py

```
import numpy as np
from Optimizer import QuadraticCG, ConjugateGradient
from Plotter import PlotContour

A = 2*np.array([[1, 0.005], [0.005, 18]])
b = np.array([1, -1])

def f(x_vect):
    x, y = x_vect[0], x_vect[1]
    x = np.array([x, y])
    return (1/2 * x.T @ A @ x + b.T @ x)[0, 0]
```

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

hist = QuadraticCG(A, b)

f = Rosenbrok
x0 = 100*np.array([1, 1])
hist = ConjugateGradient(f, x0, renewal=True, grad_check=True)

# 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, hist,
            # fname='./lab4/latex/pics/rosenbrok_1.png'
            )
```

Висновки. При виконанні даної роботи ми дослідили застосування методу спряжених градієнтів до мінімізації квадратичної функції та функції Розенброка. Для квадратичної функції, як і очікувалось, метод збігався до точки мінімуму не більше ніж 2 кроки. Для функції Розенброка використовувався варіант методу з оновленням кроків, а для розв'язку задачі одновимірної оптимізації — метод дихотомії. В цьому випадку метод спряжених градієнтів виявився суттєво швидшим за градієнтий метод та дещо повільнішим за метод Ньютона. При цьому слід зауважити, що на відміну від метода Ньютона він потребує значно менше обрахунків, а тому є більш бажаним для мінізації подібних функцій.