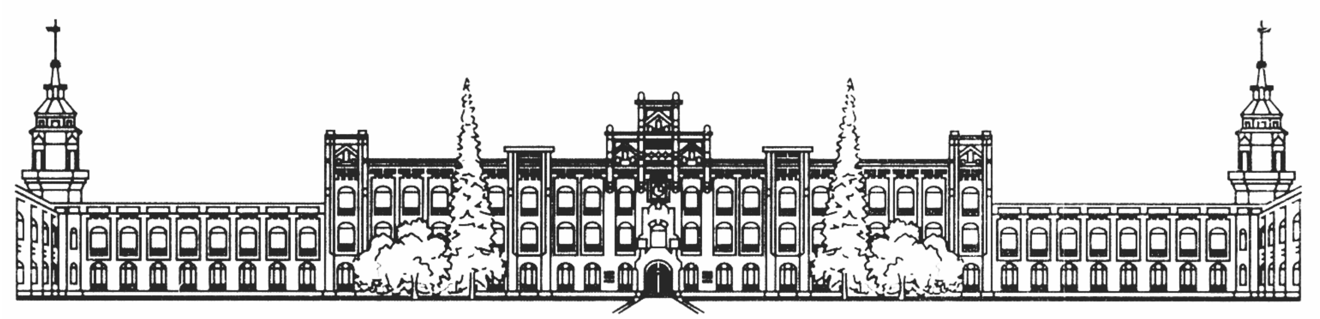


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»
Інститут прикладного системного аналізу

Лабораторна робота
з курсу «Методи оптимізації»
з теми «Чисельні методи нелінійної умовної оптимізації.
Метод проекції градієнта»

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

Перевірили
Спекторський Ігор Якович
Яковлева Алла Петрівна

Київ 2021

Варіанти 1 та 9

Завдання. Скласти програму для умовної мінімізації цільової функції f на множині X одним з методів проекції градієнта. Конкретний тип методу обрати самостійно, ураховуючи особливості цільової функції.

Цільові функції:

Варіант 1. $f(x,y,z) = x^2 + y^2 + z^2$, $X = \{(x,y,z) \in \mathbb{R}^3 : x + y + z = 1\}$.

Варіант 9. $f(x,y,z) = x + 4y + z$, $X = \{(x,y,z) \in \mathbb{R}^3 : x^2 + 3y^2 + 2z^2 \le 1\}$.

Результати роботи.

Варіант 1. Для дослідження цільової функції розглянемо її матрицю Гессе $f''(x,y,z) = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$. Очевидно, що функція є строго опуклою, а її матриця Гессе не є погано обумовленою.

Множина X є гіперплощиною в $\mathbb{R}^3$, а отже є замкненою та опуклою. Явна формула проекції на гіперплощину $H_{p\beta} = \{x \in \mathbb{R}^n : (p,x) = \beta\}$: $\pi_X a = a + (\beta - (p,a)) \cdot \frac{p}{\|p\|^2}$. В нашому випадку $\beta = 1$, а $p = (1,1,1)^T$.

Розв'яжемо задачу аналітично:

$$\mathcal{L}(x,y,z,\lambda) = x^2 + y^2 + z^2 + \lambda(x + y + z - 1)$$

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial x} = 2x + \lambda = 0 \\ \frac{\partial \mathcal{L}}{\partial x} = 2y + \lambda = 0 \\ \frac{\partial \mathcal{L}}{\partial x} = 2z + \lambda = 0 \\ x + y + z = 1 \end{cases} \Rightarrow \begin{cases} x^* = 1/3 \\ y^* = 1/3 \\ z^* = 1/3 \\ \lambda = -2/3 \end{cases}$$

Варіант 9. В цьому випадку мінімізується неперервна функція на компакті, тому розв'язок у задачі існує. Знайти явну функцію для обчислення проекції на еліпсоїд складно або навіть неможливо. Зауважимо, що цільова функція є гармонічною у заданій множині (тобто, $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = 0$), тому досягати екстремуму може лише на границі множини $\partial X = \{(x,y,z) \in \mathbb{R}^3 : x^2 + 3y^2 + 2z^2 = 1\}$. Отже, будемо шукати проекції чисельно, застосувавши *метод штрафних функцій*, який полягає у тому, що задача умовної оптимізації

$$\begin{cases} f(x) \to \min \\ g_1(x) = 0, ..., g_k(x) = 0 \end{cases}$$

замінюється задачею безумовної оптимізації

$$\widehat{f}(x) = f(x) + \alpha \cdot \left(g_1^2(x) + ... + g_k^2(x)\right) \to \min,\ \alpha \gg 0$$

Суть цього методу у тому, що другий доданок при великих значеннях α наближує індикатор множини, що задається обмеженнями умовної задачі. У випадку задачі проектування точки a на множину X, що задається обмеженнями $g_1 = 0, ..., g_k = 0$, цільова функція f має вигляд $f(x) = \|x - a\|^2$.

Розв'яжемо задачу аналітично:

$$\mathcal{L}(x, y, z, \lambda) = x + 4y + z + \lambda(x^2 + 3y^2 + 2z^2 - 1)$$

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial x} = 1 + 2\lambda x = 0 \\ \frac{\partial \mathcal{L}}{\partial x} = 4 + 6\lambda y = 0 \\ \frac{\partial \mathcal{L}}{\partial x} = 1 + 4\lambda z = 0 \\ x^2 + 3y^2 + 2z^2 = 1 \end{cases} \Rightarrow \begin{cases} x = -1/2\lambda \\ y = -2/3\lambda \\ z = -1/4\lambda \\ x^2 + 3y^2 + 2z^2 = 1 \end{cases} \Rightarrow$$

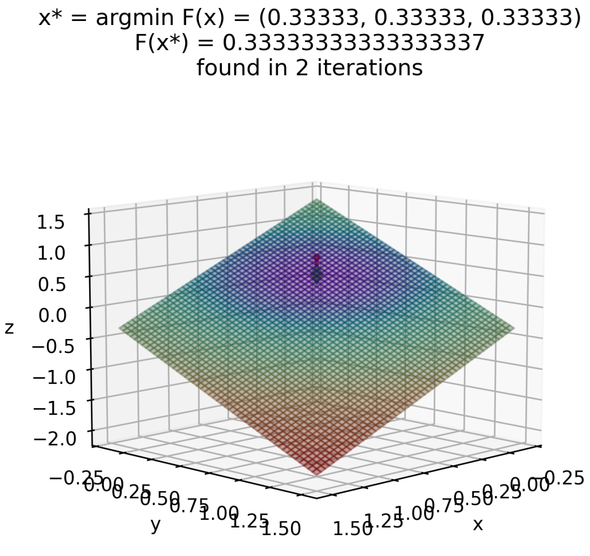
$$\Rightarrow \begin{cases} x = -1/2\lambda \\ y = -2/3\lambda \\ z = -1/4\lambda \\ \lambda = \pm\sqrt{41/24} \end{cases} \Rightarrow \begin{cases} x = \mp\sqrt{6/41} \\ y = \mp 4\sqrt{2/123} \\ z = \mp\sqrt{3/82} \\ \lambda = \pm\sqrt{41/24} \end{cases}$$

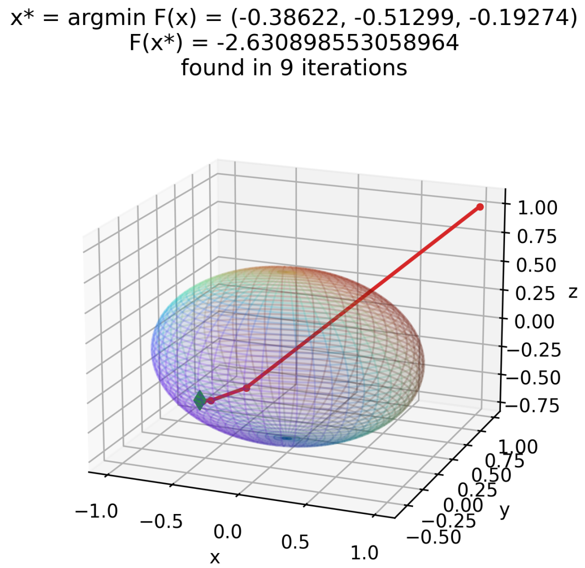
Отже, розв'язок задачі — $\left(-\sqrt{\frac{6}{41}}, -4\sqrt{\frac{2}{123}}, -\sqrt{\frac{3}{82}}\right) \approx (-0.383, -0.51, -0.191)$.

Для пришвидшення збіжності методу проекції, для реалізації було обрано метод дроблення кроку. Критерій зупинки — $|f(x_{k+1}) - f(x_k)| < \varepsilon = 10^{-5}$.

	кількість ітерацій	
x_0	варіант 1	варіант 9
argmin	2	2
$(0, 0, 0)$	2	8
$(1, 1, 1)$	2	9
$(10, 10, 10)$	2	10
$(100, 100, 100)$	2	10
$(1000, 1000, 1000)$	2	10

Приклади траєкторій спуску методу:





Лістинг. Текст програми було розділено на `Optimizer.py` з реалізацією власне методу проекції градієнту і методу Ньютона з попередньої лабораторної роботи, який використовується для розв'язання задачі проекції, та `lab3.py`, де викликаються необхідні функції та зберігаються результати.

`Optimizer.py`

```
import numpy as np
import itertools

def HyperplaneProj(point, p, beta):
    """
    Function for projecting argument point on X: (p, x) = beta
    Parameters:
    -----------
    point    : point, that will be projected on X
    p, beta : hyperplane parameters

    Returns: projX(point)
    """
    return point + (beta - np.dot(p, point))/(np.linalg.norm(p)**2) * p

def EllipsoidUniform(a, b, c, seed=42):
    """
    Select point uniformly on ellipsoid a*x**2 + b*y**2 + c*z**2 = 1
    """
    np.random.seed(seed)
    theta = 2*np.pi*np.random.rand(1)
    phi = np.arccos(2*np.random.rand(1) - 1)
    x0 = np.array([1/np.sqrt(a) * np.cos(theta) * np.sin(phi),
                   1/np.sqrt(b) * np.sin(theta) * np.sin(phi),
                   1/np.sqrt(c) * np.cos(phi)]) # select initial point
   uniformly on Ellipsoid
    np.random.seed(None)
    return x0.flatten().tolist()

def EllipsoidProj(point, a, b, c, opt, alpha=100, seed=42):
    """
    Function for projecting argument point on X: a*x**2 + b*y**2 + c*z**2 <= 1
    using penalty function method
    Parameters:
    -----------
    point        : point, that will be projected on X
    a, b, c      : ellipsoid parameters
    opt          : optimizer, callable

    Returns: projX(point)
    """
    if a*point[0]**2 + b*point[1]**2 + c*point[2]**2 <= 1:
        return point
    else:
        x0 = EllipsoidUniform(a, b, c, seed)
        def penalizer(vect):
            x, y, z = vect[0], vect[1], vect[2]
            return (x-point[0])**2 + (y-point[1])**2 + (z-point[2])**2 + alpha
   *(a*x**2 + b*y**2 + c*z**2 - 1)**2
        proj = opt.minimize(penalizer, x0)[-1, :-1]
        return proj

class GDProjOptimizer:
    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),
```

```python
                  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, proj_func, x0, h=0.005):
        """
        Parameteres
        -----------
        target_func : callable
                      function to minimize
        proj_func   : callable
                      function to calculate projection
        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).astype(float)
        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(proj_func(x - alpha*grad)) >= target_func(x) and
    alpha > 1e-7:
                alpha = alpha*self.lmb
            x = proj_func(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)

class NewtonOptimizer:
    def __init__(self, beta=1, lmb=None, tol=1e-5, max_iter=100):
        """
        Parameteres
        -----------
        beta      : float
                    initial step (default 1)
```

```
        lmb        : float or None
                     step decay parameter
                     if None - classic Newton method is used instead
        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.tol = tol
        self.max_iter = int(max_iter)
        self.lmb = lmb

    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

        def compute_hessian(target_func, x, h):
            n = len(x)
            hessian = np.empty((n, n))
            for k in range(n):
                for m in range(k+1):
                    dx = np.zeros_like(x)
                    dx[k] = h/2
                    dy = np.zeros_like(x)
                    dy[m] = h/2
                    hessian[k, m] = sum([i*j * target_func(x + i*dx + j*dy)
                                        for i, j in itertools.product([1, -1],
                                                                      repeat=2)
                                        ]) / h**2
                    hessian[m, k] = hessian[k, m]
            return hessian

        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)
            hessian = compute_hessian(target_func, x, h)
```

```
            step = np.linalg.pinv(hessian) @ grad

            alpha = self.beta
            if self.lmb is not None:
                while target_func(x - alpha*step) > target_func(x):  # > !!!
                    alpha = alpha * self.lmb

            x = x - alpha * step
            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)
```

lab3.py

```
import numpy as np
from Optimizer import GDProjOptimizer, NewtonOptimizer
from Optimizer import HyperplaneProj, EllipsoidProj, EllipsoidUniform
from Plotter import Ellipsoid, Hyperplane, make_plane, make_ellipsoid


def f1(x_vec):
    x, y, z = x_vec[0], x_vec[1], x_vec[2]
    return x**2 + y**2 + z**2

def f9(x_vec):
    x, y, z = x_vec[0], x_vec[1], x_vec[2]
    return x + 4*y + z

# set optimizer parameteres and create it
opt_params = {
    'beta': 1,
    'lmb': 0.5,
    'tol': 1e-5,
    'max_iter': 100
}
opt = GDProjOptimizer(**opt_params)

hist_var1 = opt.minimize(f1,
                    lambda x: HyperplaneProj(x, np.array([1, 1, 1]), 1),
                    x0=1/3*np.array([1, 1, 1]))
x_min, x_max = np.min(hist_var1[:, 0])-0.5, np.max(hist_var1[:, 0])+0.5
y_min, y_max = np.min(hist_var1[:, 1])-0.5, np.max(hist_var1[:, 1])+0.5

make_plane((x_min, x_max), (y_min, y_max),
           target_func=lambda x, y, z: Ellipsoid(1, 1, 1, x, y, z),
           p=np.array([1, 1, 1]), beta=1, points=hist_var1,
           fname='./lab3/latex/pics/res_var1_(argmin).png')

proj_opt = NewtonOptimizer(**opt_params)
hist_var9 = opt.minimize(f9,
                    lambda x: EllipsoidProj(x, 1, 3, 2, proj_opt),
                    x0=1*np.array([1, 1, 1]))
x_min, x_max = np.min(hist_var9[:, 0])-0.5, np.max(hist_var9[:, 0])+0.5
y_min, y_max = np.min(hist_var9[:, 1])-0.5, np.max(hist_var9[:, 1])+0.5

make_ellipsoid(target_func=lambda x, y, z: Hyperplane([1, 4, 1], 0, x, y, z),
              a=1, b=3, c=2, points=hist_var9,
              fname='./lab3/latex/pics/res_var9_(1).png')
```

Висновки. При виконанні даної роботи ми дослідили застосування методу проекції градієнта до мінімізації квадратичної функції з обмеженням на гіперплощині та лінійної функції з обмеженням на еліпсоїді. 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. Як ми побачили, метод збігається навіть тоді, коли початкова точка не належить множині X, оскільки в такому випадку наступна точка все одно гарантовано належить X. Для розв'язання задачі проектування на еліпсоїд застосували метод штрафних функцій, цільову функцію якого мінімізували за допомогою методу Ньютона. Початкова точка в цьому випадку обирається випадковим чином на поверхні еліпсоїда, щоб забезпечити збіжність методу, оскільки розв'язок задачі теж гарантовано знаходиться на його поверхні.