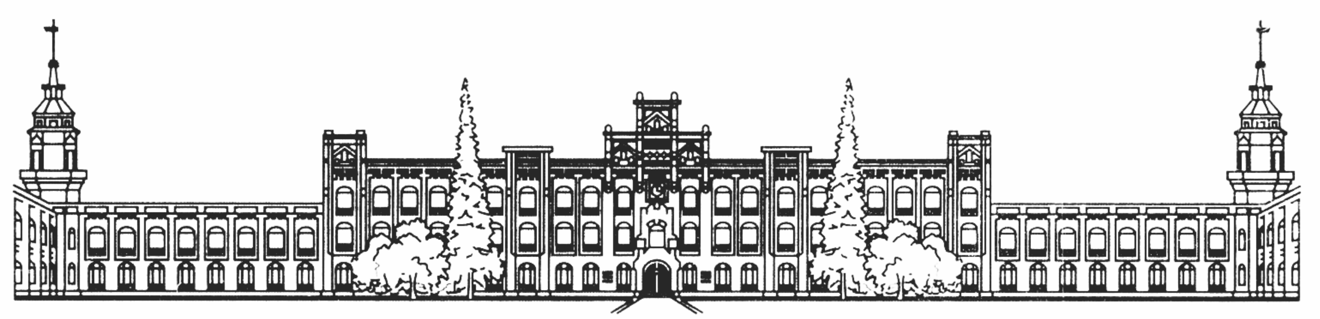

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»
Інститут прикладного системного аналізу

Лабораторна робота
з курсу «Методи оптимізації»
з теми «Метод лінеаризації в нелінійному програмуванні»

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

Перевірили
Спекторський Ігор Якович
Яковлева Алла Петрівна

Київ 2021

Завдання. Реалізувати метод лінеаризації для задачі мінімізації нелінійної функції на множині, що задана набором нелінійних нерівностей:

$$\begin{cases} f_0(x) \to \min \\ f_i(x) \le 0, \ i = 1, ..., m \end{cases} \tag{1}$$

Усі функції $f_0, f_1, ..., f_m \in C^1(\mathbb{R}^n)$ — неперервно диференційовні на $\mathbb{R}^n$.
Опис методу. Покладемо $\Phi_N(x) = f(x) + N \cdot F(x)$ для великого $N > 0$. $F(x) = \max\{0, f_1(x), ..., f_m(x)\} \ge 0$ для всіх $x \in \mathbb{R}^n$, причому рівність досягається тоді і тільки тоді, коли $f_i(x) \le 0, \ i = 1, ..., m$. N є одним з параметрів методу. Нехай $x_0 \in \mathbb{R}^n$ — довільна початкова точка. На k-тому кроці роботи методу спочатку розв'язується задача квадратичного програмування відносно p:

$$\begin{cases} \frac{1}{2}\|p\|^2 + \left\langle f_0'(x^k), p \right\rangle \to \min\limits_p \\ f_i(x^k) + \left\langle f_i'(x^k), p \right\rangle \le 0, \ i = 1, ..., m \end{cases} \tag{2}$$

Позначимо

$$D = \begin{pmatrix} -f_1'(x^k) \\ -f_2'(x^k) \\ \cdots \\ -f_m'(x^k) \end{pmatrix} = \begin{pmatrix} -(\mathrm{grad} f_1(x^k))^T \\ -(\mathrm{grad} f_2(x^k))^T \\ \cdots \\ -(\mathrm{grad} f_m(x^k))^T \end{pmatrix}, f = \begin{pmatrix} f_1(x^k) \\ f_2(x^k) \\ \cdots \\ f_m(x^k) \end{pmatrix}, c = f_0'(x^k)$$

Тоді задача (2) запишеться як

$$\begin{cases} \frac{1}{2}\|p\|^2 + \langle c, p \rangle \to \min\limits_p \\ Dp \ge f \end{cases} \tag{3}$$

Ця задача є лінійним наближенням вихідної задачі в околі точки x^k. Якщо p^k — розв'язок цієї задачі, то $x_{k+1} = x^k + \alpha^k p^k$, де α^k — перше число в послідовності $1, \frac{1}{2}, \frac{1}{4}, ...$, при якому виконується нерівність

$$\Phi_N(x^k + \alpha^k p^k) \le \Phi_N(x^k) - \varepsilon \alpha^k \|p^k\|^2 \tag{4}$$

Б. М. Пшеничний в книзі «Метод лінеаризації» [1] доводить, що точка $x \in \mathbb{R}^n$ є стаціонарною для вихідної задачі тоді і тільки тоді, коли задача квадратичного програмування

$$\begin{cases} \frac{1}{2}\|p\|^2 + \langle f_0'(x), p \rangle \to \min\limits_p \\ f_i(x) + \langle f_i'(x), p \rangle \le 0, \ i = 1, ..., m \end{cases} \tag{5}$$

має розв'язок $p(x)$ і він рівний нулю. Там само доводиться, що за деяких умов (наприклад, ліпшицевість градієнтів функцій, що задають обмеження), метод є збіжним до стаціонарних точок.

Квадратичну задачу (3), що виникає на кожному кроці роботи методу, можна замінити так званою дуальною (двоїстою) задачею:

$$\begin{cases} -\frac{1}{2}\|u\|^2 + \langle f, v\rangle \to \max\limits_{u,v} \\ D^T v - u = c \\ v \geq 0 \end{cases} \Leftrightarrow \begin{cases} \frac{1}{2}\|u\|^2 - \langle f, v\rangle \to \min\limits_{u,v} \\ D^T v - u = c \\ v \geq 0 \end{cases} \tag{6}$$

Виключивши з задачі $u = D^T v - c$, отримаємо

$$\begin{cases} \frac{1}{2}\left\langle DD^T v, v\right\rangle - \langle Dc + f, v\rangle \to \min\limits_{v} \\ v \geq 0 \end{cases} \tag{7}$$

Якщо v^* — її розв'язок, то $p = u^* = D^T v^* - c$ буде розв'язком вихідної квадратичної задачі, як показано в [1] та [2].

Задачу (7) можна розв'язати *методом мультиплікативних оновлень*, викладеним у [3]: для задачі

$$\begin{cases} \frac{1}{2}\langle Av, v\rangle + \langle b, v\rangle \to \min\limits_{v} \\ v \geq 0 \end{cases}$$

з $A > 0$ покладають $A^+ = \max\{0, A\}$, $A^- = -\min\{0, A\}$ (поелементно). Нехай v^0 — деяке початкове наближення, на кожній ітерації його координати оновлюють за формулою

$$v_i^{k+1} = v_i^k \cdot \left(\frac{-b_i + \sqrt{b_i^2 + 4\left(A^+ v^k\right)_i \left(A^- v^k\right)_i}}{2\left(A^+ v^k\right)_i}\right),\ i = 1, ..., n$$

Ці оновлення можна записати як $v^{k+1} = \varphi\left(v^k\right)$. В [3] доводиться, що таке відображення φ має нерухому точку, яка є розв'язком задачі мінімізації, тому послідовність v^k збігається до розв'язку задачі.

Нарешті, у [1] показано, що замість вибору параметру N у методі лінеаризації можна задавати деяке велике початкове значення N_0, а потім на кожній ітерації оновлювати його за допомогою вектора v^k, отриманого при розв'язанні дуальної задачі (7), за правилом

$$\begin{cases} N_{k+1} = 2\sum\limits_{i=1}^{m} v_i^k, & \text{якщо } \sum\limits_{i=1}^{m} v_i^k > N_k \\ N_{k+1} = N_k, & \text{інакше} \end{cases}$$

Також зауважимо, що метод можна застосувати до обмежень-рівностей виду $f(x) = 0$, оскільки вони еквівалентні виконанню двох обмежень-нерівностей $f(x) \leq 0$ та $f(x) \geq 0$.

Результати роботи. Ми перевірили роботу методу на декількох задачах зі ста випадкових початкових точок:

задача	**середня к-сть ітерацій**
$\begin{cases} x^2+y^2+z^2 \to \min \\ x+y+z=1 \end{cases}$	4.17
$\begin{cases} x+4y+z \to \min \\ x^2+3y^2+2z^2 \leq 1 \end{cases}$	241.5
$\begin{cases} (x-2)^2+(y-1)^2 \to \min \\ y-x^2 \leq 0 \end{cases}$	20.23
$\begin{cases} -xy \to \min \\ x^2+y^2 \leq 1 \\ x \geq 0, y \geq 0 \end{cases}$	20.9
$\begin{cases} x^2+y \to \min \\ x+y-1 \leq 0 \\ x^2+y^2 \leq 9 \end{cases}$	27.7

В цілому, метод лінеаризації досить швидко збігається до розв'язку задачі, але іноді збіжність була довгою через вибір початкової точки.

Лістинг. Текст програми було розділено на `Optimizer.py` з реалізацією методу лінеаризації та `lab5.py`, де викликаються необхідні функції та зберігаються результати.

`Optimizer.py`

```
import numpy as np

def gradient(target_func, x, h=0.005):
    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 NonnegativeQuadraticOptimizer(A, b):
    """
    Solve 1/2 * (Av, v) + (b, v) -> min under v >= 0
    """
    A_plus, A_minus = np.maximum(0, A), -np.minimum(0, A)
    v = np.random.rand(len(b))
    v_temp = v.copy()
    for _ in range(100):
        Ap_v, Am_v = A_plus @ v, A_minus @ v
        for i in range(len(v)):
            if np.abs(Ap_v[i]) >= 1e-8:
                v_temp[i] = v[i] * (-b[i] + np.sqrt(b[i]**2 + 4*Ap_v[i]*Am_v[i
  ]))/(2*Ap_v[i])
            else:
                v = np.random.rand(len(b))
```

```python
                break
        if np.linalg.norm(v - v_temp) < 1e-7:
            v = v_temp.copy()
            break
        v = v_temp.copy()
    return v

def GeneralQuadraticOptimizer(p, A, b):
    """
    Solve 1/2 * ||x||^2 + (p, x) -> min under A * x >= b
    using dual problem:
        1/2 * ||u||^2 - (b, v) -> min under
        A.T * v - u = p
        v >= 0
    solution in u == solution in x
    """
    # dual target: u = A.T @ v - p, so it equals
    # 1/2*(AA.T*v, v) - (A*p + b, v) + ||p||^2 -> min, v >= 0
    v = NonnegativeQuadraticOptimizer(A @ A.T, -A @ p - b)
    u = A.T @ v - p
    return u, v

def LinearizationOptimizer(f0, constraints, x0, eps, tol=1e-5, max_iter=100):
    """
    Solve f0(x) -> min under f_i(x) <= 0 for f_i in constraints
    using linearization method
    0 < eps < 1 is method parameter
    """
    N_k = 100
    def F_N(x, N):
        return f0(x) + N*max([0] + [f(x) for f in constraints])
    history = []
    x = np.copy(x0).astype(float)
    history.append([*x0, f0(x0)])
    for k in range(max_iter):
        df0 = gradient(f0, x)
        d_constr = []
        constr_val = []
        for i in range(len(constraints)):
            d_constr.append(gradient(constraints[i], x))
            constr_val.append(constraints[i](x))
        d_constr = np.array(d_constr)
        constr_val = np.array(constr_val)
        p, v = GeneralQuadraticOptimizer(df0, -d_constr, constr_val)
        alpha = 1
        while F_N(x + alpha*p, N_k) > F_N(x, N_k) - eps*alpha*np.linalg.norm(p)
    **2:
            alpha = alpha * 0.5
        x = x + alpha*p
        if v.sum() > N_k:
            N_k = 2 * v.sum()
        history.append([*x, f0(x)])
        if k > 0 and np.linalg.norm(p) < tol:
                break
    return np.array(history)
```

lab4.py

```python
import numpy as np
from Optimizer import LinearizationOptimizer

"""def f0(x_vect):
    x, y = x_vect[0], x_vect[1]
    return (x-2)**2 + (y-1)**2
```

```
def f1(x_vect):
    x, y = x_vect[0], x_vect[1]
    return -(-x**2 + y)"""

"""def f0(x_vect):
    x, y, z = x_vect[0], x_vect[1], x_vect[2]
    return x + 4*y + z

def f1(x_vect):
    x, y, z = x_vect[0], x_vect[1], x_vect[2]
    return x**2 + 3*y**2 + 2*z**2 - 1"""

"""def f0(x_vect):
    x, y, z = x_vect[0], x_vect[1], x_vect[2]
    return x**2 + y**2 + z**2

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

def f2(x_vect):
    x, y, z = x_vect[0], x_vect[1], x_vect[2]
    return -(x + y + z - 1)"""

"""def f0(x_vect):
    x, y = x_vect[0], x_vect[1]
    return -x*y

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

def f2(x_vect):
    x, y = x_vect[0], x_vect[1]
    return x**2 + y**2 - 1

def f3(x_vect):
    x, y = x_vect[0], x_vect[1]
    return -x

def f4(x_vect):
    x, y = x_vect[0], x_vect[1]
    return -y"""

def f0(x_vect):
    x, y = x_vect[0], x_vect[1]
    return x**2 + y

def f1(x_vect):
    x, y = x_vect[0], x_vect[1]
    return x + y - 1

def f2(x_vect):
    x, y = x_vect[0], x_vect[1]
    return x**2 + y**2 - 9


res = []
for k in range(10):
    x0 = np.random.randn(2)
    hist = LinearizationOptimizer(f0, [f1, f2], x0, eps=0.5, max_iter=1000)
    res.append(len(hist))
    print(f'{k}\targmin = {hist[-1, :-1]}, found in {len(hist)} iterations')
print('average =', sum(res)/len(res))
```

Висновки. Реалізований нами метод лінеаризації виявився досить швидким та універсальним методом розв'язання задач умовної мінімізації нелінійної функції на множині, заданій нелінійними обмеженнями-нерівностями та рівностями. Він є складним у реалізації та з обчислювальної точки зору, оскільки на кожному кроці необхідно розв'язувати задачу квадратичного програмування з лінійними обмеженнями, але ця складність виправдовується універсальністю застосування методу: цільова функція та функції-обмеження мають бути лише неперервно-диференційовними. Але через те, що опуклість функцій-обмежень та строга опуклість цільової функції не вимагається, в загальному випадку метод збігається лише до стаціонарних точок, тому в таких випадках краще запустити мінімізацію з декількох початкових точок та порівняти отримані значення.

Бібліографія

[1] Пшеничный Б.Н. *Метод линеаризации.* — М.: Наука. Главная редакция физико-математической литературы, 1983. — 136 с.

[2] Dorn, W., 1960. Duality in quadratic programming. *Quarterly of Applied Mathematics*, 18(2), pp.155-162.

[3] Sha, F., Lin, Y., Saul, L. and Lee, D., 2007. Multiplicative Updates for Nonnegative Quadratic Programming. Neural Computation, 19(8), pp.2004-2031.