Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування та економічної кібернетики

КУРСОВА РОБОТА

бакалавра

на тему: **Дослідження збіжності методів при наявності відновлення**

Виконав: здобувач III курсу
денної форми навчання
спеціальності 113 Прикладна математика
Лазор Володимир Вікторович

Керівник: кандидат фізико-математичних
наук, доцент кафедри оптимального
керування і економічної кібернетики
Яровий А. Т.

Одеса — 2024 р.

ЗМІСТ

ВСТУП

Метод спряжених градієнтів (далі - МСГ) є одним з найбільш відомих ітеративних алгоритмів мінімізації нелінійних функцій, який широко використовується в різних галузях знань. Даний метод є ефективним для розв'язування систем лінійних алгебраїчних рівнянь (далі - СЛАР) вигляду $Ax = b$, де A - відома симетрична невід'ємно-визначена матриця, b - відомий вектор, x - невідомий вектор. Зі збільшенням кількості рівнянь в системі (особливо коли матриця A - розріджена) класичні методи розв'язування СЛАР стають неефективними в сенсі використання пам'яті комп'ютера, що негативно впливає на їх швидкодію. Для таких задач МСГ є помітно ефективнішим, оскільки на обчислення витрачається значно менше пам'яті, і, відповідно, метод буде працювати швидше [1].

Однак розв'язування деяких СЛАР зводиться до мінімізації погано обумовлених функцій, що може призвести до зменшення точності отриманих розв'язків і, відповідно, до більшої кількості ітерацій методу. Це зумовлено похибками при округленні чисел, що накопичуються із кожною новою ітерацією [5]. Щоб позбутись проблеми із накопиченням похибок і, таким чином, прискорити процес мінімізації методом спряжених градієнтів, було запропоновано модифікований варіант МСГ із процедурою відновлення параметру β. Робота в цьому напрямку триває і зараз, що підтверджує наукова стаття 2023-го року дослідників із Китайського університету Мінзу [7], які запропонували свою процедуру відновлення. Класичною вважається процедура, коли параметр β прирівнюється до 0. Ця процедура згадується у статтях [6], [4], а також у навчально-методичному посібнику "Методи оптимізації" [10], тому саме її буде розглянуто в цій роботі.

Нашою метою буде дослідити, як саме зазначена вище процедура відновлення вплине на швидкодію і точність роботи методу спряжених градієнтів на погано обумовлених функціях. Критеріями оцінки будуть кількість ітерацій та отримані значення функції на останній ітерації.

Ідею з відновленням можна перенести і на інші методи оптимізації, наприклад, методи другого порядку та методи змінної метрики (квазіньютонівські).

РОЗДІЛ 1

ПОСТАНОВКА ЗАДАЧІ

Дослідження методів буде проводитись експериментальним шляхом.
Хід роботи розділимо на 4 етапи:

1) формулювання умов проведення експерименту, наведення теоретичних відомостей про використовувані поняття;
2) наведення теоретичних відомостей про досліджувані методи;
3) дослідження роботи алгоритмів на тестових функціях за різних початкових умов;
4) порівняння методів на основі результатів експерименту, формулювання висновків

У ході роботи будуть досліджуватись 2 методи:

- метод спряжених градієнтів (без відновлення параметру β);
- метод спряжених градієнтів з відновленням параметру β через кожні n ітерацій (про те, як визначається число n, буде згадано у наступному розділі)

У ході експерименту ми дамо відповідь на такі запитання:

- наскільки точно кожен з методів знаходить точку мінімума заданої функції?
- який з методів знайде точку мінімума за меншу кількість ітерацій?
- як поводяться методи залежно від функції?

Код програми, що реалізує обидва алгоритми МСГ, можна переглянути в **додатку А**. Детальніше про обраний підхід до написання програми та обґрунтування такого підходу можна дізнатись у цьому ж додатку.

1.1 Умови проведення експерименту

1.1.1 Вимоги до тестових функцій

Щоб функцію можна було мінімізувати методом спряжених градієнтів, мають виконуватись:

- **необхідна умова мінімума першого порядку**;
- **достатня умова мінімума другого порядку**

Раніше в цій роботі згадувалось поняття тестової функції. Тестовими прийнято називати функції, на основі яких створюються тестові задачі. Виділимо деякі з вимог до тестових задач [11]:

- тести повинні бути різноманітними, наприклад, різного виміру, різної обумовленості (це дозволить виявити слабкі місця алгоритма, якщо вони є);
- розв'язок тестової задачі має бути відомим (без цього було б неможливим оцінити близкість отриманої точки до реальної точки мінімума)

1.1.2 Критерії оцінки якості роботи алгоритмів

Перед тим як порівнювати вищезазначені методи, необхідно визначитись із критеріями, за якими ми будемо оцінювати їх якість.
Перелічимо ці критерії:

- метод має давати стабільні результати для будь-якої функції з одного класу (за умови, що функція відповідає переліченим вимогам);
- для обох методів використовується той самий **критерій зупинки**;
- якщо виконуються перші 2 пункти, то порівнюється точність, з якою методи знайшли наближені розв'язки задачі;
- якщо точність обох методів задовільна, то порівнюється кількість ітерацій, за яку було знайдено наближений розв'язок

1.1.3 Критерій зупинки для градієнтних методів

Як визначити, коли методу слід припинити обчислення і зафіксувати наявний результат? Для цього існують критерії зупинки. Критерій зупинки - це набір умов, кожна з яких описує ситуацію, коли подальші обчислення не призведуть до помітно кращого результату.

У випадку безумовної мінімізації гладких функцій критерій зупинки A складається з трьох умов [9]:

$$A = \begin{cases} A_1: \ f(x^{k-1}) - f(x^k) < \tau(1 + |f(x^k)|) \\ A_2: \ \|x^{k-1} - x^k\| < \sqrt{\tau}(1 + \|x^k\|) \\ A_3: \ \|\frac{\partial f(x^k)}{\partial x}\| \leqslant \sqrt[3]{\tau}(1 + |f(x^k)|) \end{cases}$$

A_1 - досягнута потрібна точність розв'язку;
A_2 - метод дуже повільно рухається до точки мінімума;
A_3 - метод зациклився, або почав розбігатись.

РОЗДІЛ 2

МЕТОД СПРЯЖЕНИХ ГРАДІЄНТІВ

В цьому розділі ми наведемо теоретичні відомості про метод спряжених градієнтів та процедуру відновлення параметра β. Але спершу необхідно розібратись із методом, який покладений в основу усіх інших градієнтних методів - методом найшвидшого спуску.

2.1 Метод найшвидшого спуску

Розглянемо задачу мінімізації деякої гладкої функції $f(x) \to min$ без обмежень. Постає логічне запитання - а в якому напрямку слід рухатись, щоб якнайшвидше досягти точки мінімуму x^*? Відповісти на це запитання допоможе поняття градієнта функції:

$$\frac{\partial f\left(x\right)}{\partial x} = \begin{bmatrix} \frac{\partial f(x_1)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x_n)}{\partial x_n} \end{bmatrix}$$

Як відомо з математичного аналізу, градієнт - це вектор, у напрямку якого функція в деякому околі заданої точки x зростає найшвидше. Звідси природньо зробити висновок: якщо ми хочемо наблизитись до точки мінімуму x^*, то найшвидше буде рухатись у напрямку, протилежному до напрямку градієнта:

$$-\frac{\partial f(x)}{\partial x}$$

Означення 2.1. Нехай функція $f(x)$ — неперервно-дифереційовна.
Метод найшвидшого спуску має вигляд:

$$x^{k+1} = x^k - \alpha_k \frac{\partial f(x^k)}{\partial x}$$

2.1.1 Вибір параметру α_k

Способів обрати параметр α_k існує декілька. В цій роботі було обрано спосіб, в якому потрібно скористатись одновимірною мінімізацією функції $f(x^k + \alpha s^k)$ за параметром α, де $s^k = \frac{\partial f(x^k)}{\partial x}$ - напрямок спуску:

$$\alpha_k : \min_{\alpha \geqslant 0} f\left(x^k + \alpha_k s^k\right)$$

Цей спосіб було обрано не дарма: він дозволяє визначити таке значення α_k, при якому рух на даній ітерації у напрямку найшвидшого спуску приведе до точки, яка розташована відносно точки мінімуму найближче.

Щоб підтвердити цю тезу, розглянемо зображення, які було взято зі статті [2]:

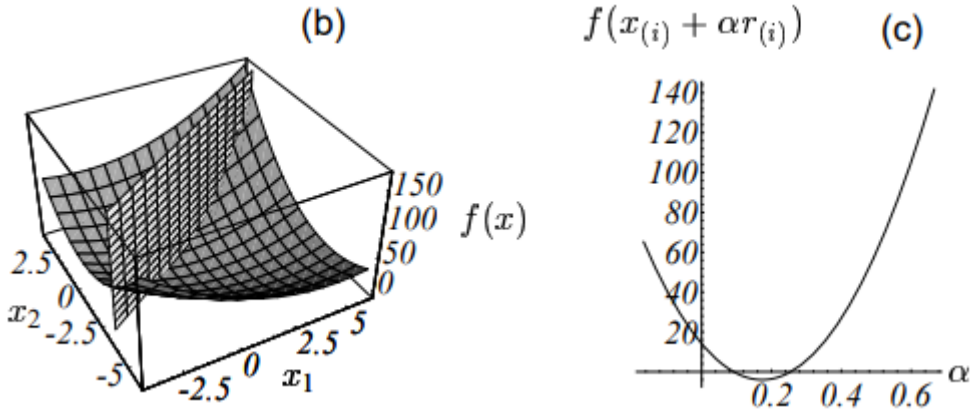

Рис. 2.1. На рисунку (b) зображена функція двох змінних $f(x)$, яку перетинає площина, така, що її перетин із $f(x)$ дасть функцію $f(x^k+\alpha s^k)$, що зображена на рисунку (c).

2.2 Метод спряжених градієнтів

Із розвитком градієнтних методів виявилось, що метод найшвидшого спуску, насправді, не обов'язково буде найшвидшим. Це призвело до появи модифікації цього методу - двокрокового методу спряжених градієнтів. Метод називається двокроковим, оскільки для знаходження (k+1)-го наближення використовуються два попередніх: (k-1)-е і k-е.

Означення 2.2. Нехай функція $f(x)$ — неперервно-диференційовна.
Метод спряжених градієнтів має вигляд:

$$x^{k+1} = x^k - \alpha_k \frac{\partial f(x^k)}{\partial x} + \beta_k \left(x^k - x^{k-1}\right)$$
$$\{\alpha_k; \beta_k\} : \min_{\alpha,\beta} f\left(x^k - \alpha \frac{\partial f(x^k)}{\partial x} + \beta \left(x^k - x^{k-1}\right)\right)$$

МСГ можна переписати в дещо іншому вигляді:

$$x^{k+1} = x^k - \alpha_k s^k,$$
$$\alpha_k : \min_{\alpha} f\left(x^k + \alpha_k s^k\right),$$
$$s^k = -\frac{\partial f(x^k)}{\partial x} + \beta_k s^{k-1}, \quad s^0 = -\frac{\partial f(x^0)}{\partial x}, \quad \beta_0 = 0.$$

Метод отримав свою назву через такі властивості:

- градієнти $\frac{\partial f(x^i)}{\partial x}, \quad i = \overline{0, n-1}$ — взаємно ортогональні;
- напрямки $s^0, ..., s^n$ — взаємно спряжені, тобто $\left(As^i, s^j\right) = 0$

Коли розв'язується СЛАР вигляду $Ax = b$, напрямки $s^0, ..., s^n$ також називають A-ортогональними [3]. Детальніше про спряжені напрямки можна дізнатись в цій ж статті.

2.2.1 Вибір параметру β_k

Як і у випадку параметра α_k, способів обрати параметр β_k існує декілька [4], [10]. В цій роботі було обрано коефіцієнт Флетчера-Рівза:

$$\beta_k^{FR} = \frac{\|\frac{\partial f(x^k)}{\partial x}\|}{\|\frac{\partial f(x^{k-1})}{\partial x}\|}$$

2.2.2 Процедура відновлення параметру β_k

Процедура відновлення параметру β_k дозволяє кожні n кроків "забувати" попередні напрямки спуску і зробити крок у напрямку градієнта (найшвидшого спуску).
В такому разі метод спряжених градієнтів набуває вигляду:

$$x^{k+1} = x^k - \alpha_k s^k,$$

$$\alpha_k : \min_\alpha f\left(x^k + \alpha_k s^k\right),$$

$$s^k = -\frac{\partial f\left(x^k\right)}{\partial x} + \beta_k s^{k-1}, \quad s^0 = -\frac{\partial f\left(x^0\right)}{\partial x},$$

$$\beta_k = \begin{cases} 0, \ k = 0, n, 2n, \dots \\ \frac{\left\|\frac{\partial f\left(x^k\right)}{\partial x}\right\|^2}{\left\|\frac{\partial f\left(x^{k-1}\right)}{\partial x}\right\|^2} \end{cases}$$

Залишається відкритим питання: яка кількість кроків буде оптимальною для проведення відновлення? Природньо зробити припущення, що відновлення варто робити кожні $n = dim\ E^m = m$ кроків, тобто прирівняти число n до вимірності простору (інакше кажучи, до кількості змінних у функції). Таке припущення пояснюється тим, що МСГ не може згенерувати більшу за вимірність простору кількість взаємно спряжених векторів [4].

РОЗДІЛ 3

ДОСЛІДЖЕННЯ РОБОТИ АЛГОРИТМІВ

3.1 Тестові задачі

Функція Розенброка, $n = 2$

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2,$$
$$x^0 = (-1.2; 1),\ x^* = (1; 1),\ f^* = 0.$$

Функція Пауелла

$$f(x) = (x_1 + 10x_2)^2 + 5\,(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10\,(x_1 - x_4)^4,$$
$$x^0 = (3; -1; 0; 1),\ x^* = (0; 0; 0; 0),\ f^* = 0.$$

Тестова функція 3

$$f(x) = 100\left(x_2 - x_1^3\right)^2 + (1 - x_1)^2,$$
$$x^0 = (-1.2; 1),\ x^* = (1; 1),\ f^* = 0.$$

Тестова функція 4

$$f(x) = 100\left[x_3 - \left(\frac{x_1 + x_2}{2}\right)^2\right]^2 + (1 - x_1)^2 + (1 - x_2)^2,$$
$$x^0 = (-1.2; 2; 0),\ x^* = (1; 1; 1),\ f^* = 0.$$

3.2 Порівняння методів

В цій секції представлені результати експерименту, а саме: таблиці знайдених наближень мінімумів тестових функцій і таблиці з кількістю ітерацій кожного методу. Усі наближення були розраховані при різній заданій точності τ, відповідно, різною буде і кількість ітерацій методів залежно від точності.

3.2.1 МСГ без відновлення

Функція	$\tau = 0.01$	$\tau = 0.001$	$\tau = 0.0001$	$\tau = 0.00001$
Розенброка	0.0412	0.0005	0.00010	0.000025
Пауелла	0.0035	0.0009	0.00030	0.000100
Тестова 3	0.0049	0.0049	0.00050	0.000120
Тестова 4	0.0012	0.0003	0.00009	0.000036

Табл. 3.1. Результати мінімізації МСГ **без відновлення**
(числа в комірках - f_{min} відповідних функцій)

Функція	$\tau = 0.01$	$\tau = 0.001$	$\tau = 0.0001$	$\tau = 0.00001$
Розенброка	49	70	79	89
Пауелла	22	31	40	53
Тестова 3	38	39	53	63
Тестова 4	30	37	43	51

Табл. 3.2. Кількість ітерацій МСГ **без відновлення**
(числа в комірках - кількість ітерацій методу
для відповідних функцій із заданою точністю)

3.2.2 МСГ з відновленням через $n = dim\ E^m$ кроків

Функція	$\tau = 0.01$	$\tau = 0.001$	$\tau = 0.0001$	$\tau = 0.00001$
Розенброка	0.0433	0.0010	0.00009	0.000019
Пауелла	0.0097	0.0011	0.00055	0.000208
Тестова 3	0.0524	0.0058	0.0001	0.000014
Тестова 4	0.0143	0.0029	0.0004	0.000242

Табл. 3.3. Результати мінімізації МСГ **з відновленням**
(числа в комірках - f_{min} відповідних функцій)

Функція	$\tau = 0.01$	$\tau = 0.001$	$\tau = 0.0001$	$\tau = 0.00001$
Розенброка	37	43	45	47
Пауелла	16	22	24	28
Тестова 3	25	33	39	41
Тестова 4	10	13	18	22

Табл. 3.4. Кількість ітерацій МСГ **з відновленням**
(числа в комірках - кількість ітерацій методу для відповідних функцій із заданою точністю)

В наступній секції будуть представлені графіки функції Розенброка та тестової функції №3, на яких позначено шлях спуску, побудований методом спряжених градієнтів з відновленням і без відновлення. Синім кольором позначені ті частини графіків, в яких значення функцій наближаються до нуля, відповідно мінімуми цих функцій також будуть знаходитись в позначених зонах.

3.2.3 Візуалізація роботи МСГ

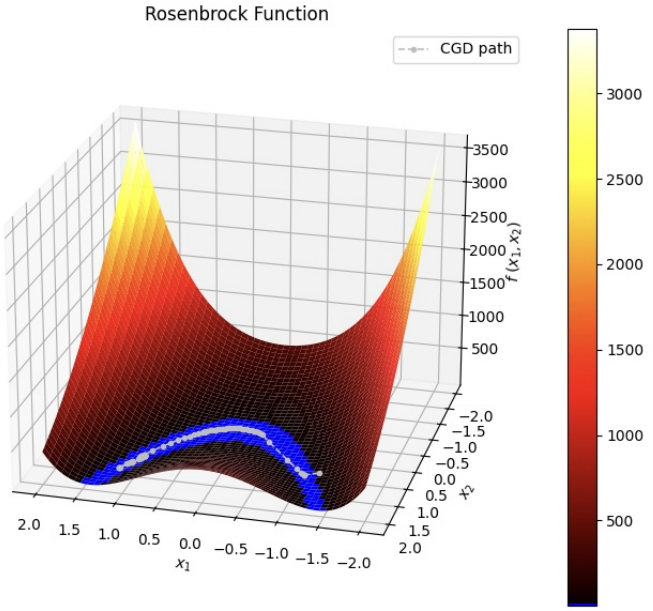

Рис. 3.1. Процес мінімізації функції Розенброка МСГ без відновлення

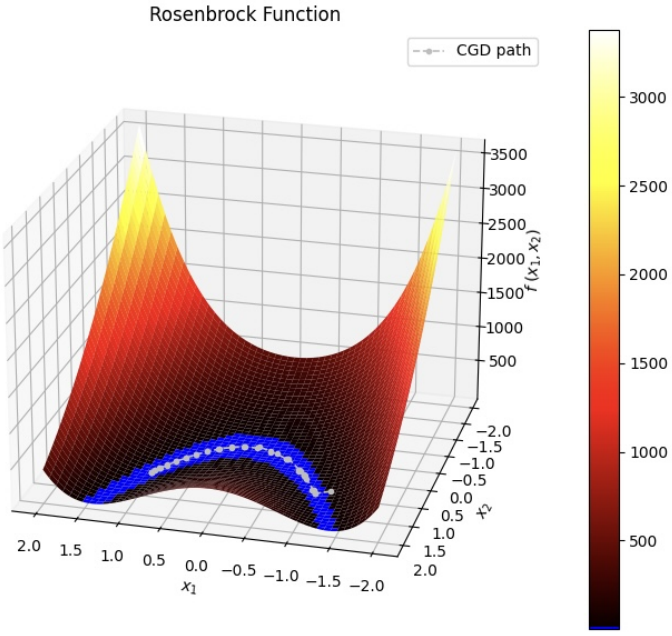

Рис. 3.2. Процес мінімізації функції Розенброка МСГ з відновленням

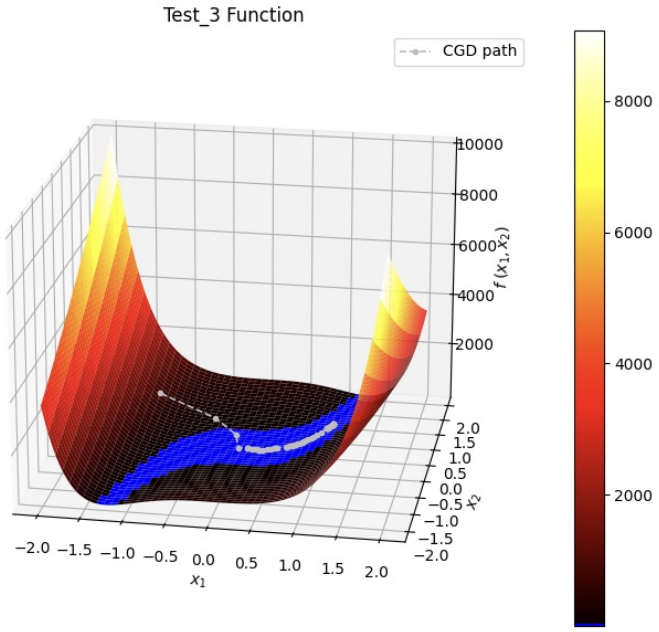

Рис. 3.3. Процес мінімізації тестової функції №3 МСГ без відновлення

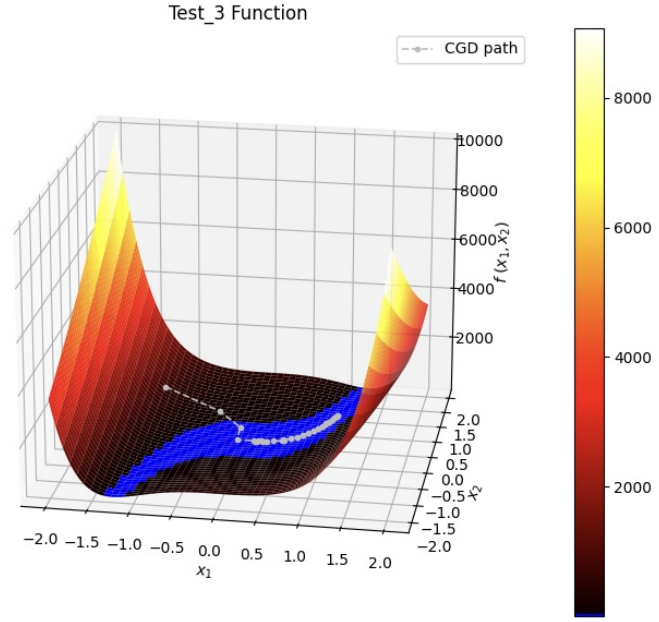

Рис. 3.4. Процес мінімізації тестової функції №3 МСГ з відновленням

ВИСНОВКИ

За даними складених таблиць можна зробити такі висновки:

- МСГ з відновленням при заданій точності $\tau < 0.01$ мінімізував функцію Розенброка за майже вдвічі меншу кількість ітерацій, незначно поступаючись в точності МСГ без відновлення;
- МСГ з відновленням при будь-якій заданій точності мінімізував тестову функцію №4 за більш ніж в 2 рази меншу кількість ітерацій, ніж МСГ без відновлення, поступаючись у точності;
- Для функції Пауелла результат аналогічний тому, що отримано для функції Розенброка;
- Для тестової функції №3 приріст у швидкості МСГ з відновленням незначний, проте результати мінімізації при заданій точності $\tau < 0.001$ стали краще.

Результати експерименту показали, що метод спряжених градієнтів в середньому працює помітно швидше з відновленням, ніж без відновлення. Точність розрахунків є доволі низькою при $\tau = 0.01$, проте якщо її збільшити, то МСГ з відновленням майже не поступається МСГ без відновлення.

МСГ без відновлення має тенденцію "перестрибувати" через точку мінімума:

- (функція Розенброка при $\tau = 0.01$) $x_{min} = (1.2029; 1.4476)$
- (функція Розенброка при $\tau = 0.001$) $x_{min} = (1.0226; 1.0457)$
- (тестова функція №4 при $\tau = 0.01$) $x_{min} = (1.0021; 0.9649; 0.9666)$

МСГ з відновленням запобігає "перестрибуванню":

- (функція Розенброка при $\tau = 0.01$) $x_{min} = (0.7921; 0.6265)$
- (функція Розенброка при $\tau = 0.001$) $x_{min} = (0.9684; 0.9377)$
- (тестова функція №4 при $\tau = 0.01$) $x_{min} = (0.9495; 0.8916; 0.8468)$

Вищезазначені факти у випадку функції Розенброка і тестової функції №3 також можна **побачити візуально**.

СПИСОК ЛІТЕРАТУРИ

1. J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Pittsburgh, 1994. - 1 с.
2. J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Pittsburgh, 1994. - 7 с.
3. J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Pittsburgh, 1994. - 23 с.
4. J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Pittsburgh, 1994. - 42-44 с.
5. M. R. Hestenes, E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems, 1952. - 419 с.
6. M. R. Hestenes, E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems, 1952. - 421 с.
7. A new conjugate gradient method with a restart direction and its application in image restoration / Yixin Li, Chunguang Li, Wei Yang, Wensheng Zhan. China, Yinchuan: School of Mathematics and Information Science, North Minzu University, 2023. - 3 с.
8. Яровий А. Т. Методи оптимізації: навчально-методичний посібник. Одеса, 2020. - 19-20 с.
9. Яровий А. Т. Методи оптимізації: навчально-методичний посібник. Одеса, 2020. - 63 с.
10. Яровий А. Т. Методи оптимізації: навчально-методичний посібник. Одеса, 2020. - 70 с.
11. Яровий А. Т. Методи оптимізації: навчально-методичний посібник. Одеса, 2020. - 109 с.

ДОДАТОК А

Код програми

Про обраний підхід до написання програми

Для реалізації методу спряжених градієнтів я використовував мову програмування **Python** разом із такими бібліотеками (модулями):

- **numpy** - для спрощення обчислень, зокрема множення матриць;
- **sympy** - для символьних обчислень

Для побудови графіків я використав бібліотеку **matplotlib**.

Доречним буде запитання: навіщо додавати символьні розрахунки у комп'ютерну програму? Такий підхід я обрав з двох причин:

1) це спрощує програмний код;
2) це дає можливість розробити універсальну програму, яка буде працювати на будь-якій функції (що, звісно, відповідає вимогам, зазначеним у цій роботі)

Універсальність програми досягається за рахунок символьного знаходження градієнту, в який потім підставляються конкретні значення змінних функції. Це дозволяє у символьному вигляді знайти вектор, в якому міститься невідома - β, яку потім буде легко знайти, розв'язуючи задачу одновимірної мінімізації.

Нижче наведено повний код програми, що реалізує метод спряжених градієнтів. Програма дозволяє самостійно вказувати, чи буде в методі використовуватись процедура відновлення.

Реалізація МСГ на Python

```
import numpy as np
import sympy as sym
from sympy.solvers import solve

def conjugate_gradient(function, x_0, precision=0.01,
                       show_iterations=False,
                       use_restart=False):
    """
    Conjugate gradient descent method
        :param function: the given function
        :param x_0: starting point (an initial guess)
        :param precision: precision of the CGD method
        :param show_iterations: if set to True, prints
        each iteration of the algorithm
        :param use_restart: if set to True,
        activates restart procedure
        :return: approximated minimum of the function
    """
    variables = list(sym.ordered(function.free_symbols))
    dim = len(variables)
    # Symbolic gradient
    gradient = sym.Matrix([function.diff(x) for x in variables])

    def substitute(func, expression):
        """
        Assisting function to shorten the operation
        of passing an expression to
        the input function and its gradient
            :param func: the given function
            :param expression: function's argument
            :return: function's value at a given point
        """
```

```
    result = sym.N(func.subs(list(zip(variables, expression))))
    return result

def f(expression):
    """
    Shortens the operation of passing an expression
    to an input function
    """
    return substitute(function, expression)

def grad(expression):
    """
    Shortens the operation of passing an expression
    to a gradient of an input function
    """
    return substitute(gradient, expression)

def minimize_1d(func, variable):
    """
    1d minimization of the given function
    (performed by sympy.minimum())
        :param func: given function
        :param variable: variable which is to be minimized by
        :return: point of global minimum
    """
    # returns minimum of a function
    f_min = sym.N(sym.minimum(func, variable))
    # choosing only real parts if roots are complex
    x_min = [sym.re(num) for num in solve(func - f_min,
                                          variable)]
    # converting precise value into float
    x_min = sym.N(min(list(map(abs, list(map(float, x_min))))))
    return x_min
```

```
def norm(vector):
    """
    Calculates norm of the given vector
    :param vector:
    :return:
    """
    return np.linalg.norm(np.array(sym.N(vector)).astype(float))

def print_iterations():
    if show_iterations is True:
        print(f"k={k} ----------------------------------")
        print('|GD Results|:')
        print(f"\t(symbolic) x_{k} = {x}\n")
        print(f"\tx_{k} = {x_k[k]}\n")

# 0-th iteration
a = sym.Symbol('a')
b = 0
x_k = [sym.Matrix(x_0)]  # steps of minimization
s_k = [sym.Matrix(-grad(x_0))]  # steps of anti-gradient

total_iterations = 0

k = 1
while True:
    # Calculating x_k and f(x_k) for current (k) iteration
    x = x_k[k-1] + a * s_k[k-1]
    f_a = f(x)
    alpha = minimize_1d(f_a, a)
    x_k.append(sym.N(x.subs(a, alpha)))

    print_iterations()  # prints only if show_iterations=True

    # Calculating beta and s for the next (k+1) iteration
```

```
        numerator = norm(grad(x_k[k]))**2
        denominator = norm(grad(x_k[k-1]))**2

        # restart procedure
        if use_restart:
            b = numerator / denominator if k % dim != 0 else 0
        else:
            b = numerator / denominator

        s = -grad(x_k[k]) + b * s_k[k-1]
        s_k.append(s)

        # Stop criterion (with conditions A1-A3)
        A1 = abs(sym.N(f(x_k[k-1]) - f(x_k[k]))) < precision *
        (1 + abs(sym.N(f(x_k[k]))))
        A2 = norm(x_k[k-1] - x_k[k]) < np.sqrt(precision) *
        (1 + norm(x_k[k]))
        A3 = norm(grad(x_k[k])) <= precision**(1/3) *
        (1 + abs(sym.N(f(x_k[k]))))
        if A1 and A2 and A3:
            total_iterations = k
            break

        k += 1

    x_k = np.array(x_k).astype(float)  # converting back to list

    if show_iterations is True:
        print("\n---------------------------------------")
        print(f"Iterations: {k}")

    return x_k, total_iterations
```

ДОДАТОК Б

Допоміжні теореми та означення

Теорема 3.1. *[8] Нехай функція $f(x)$ диференційовна у точці $x^* \in E^n$, де E^n - евклідовий простір. Тоді якщо x^* — локальний розв'язок задачі $f(x) \to min$, то*

$$\frac{\partial f(x^*)}{\partial x} = 0$$

Теорема 3.2. *[8] Нехай функція $f(x)$ двічі диференційовна у точці $x^* \in E^n$ Якщо x^* — точка локального мінімуму в задачі $f(x) \to min$, то матриця $\frac{\partial^2 f(x^*)}{\partial^2 x}$ невід'ємно-визначена, тобто*

$$\left(\frac{\partial f(x^*)}{\partial x} x, x\right) \geqslant 0 \quad \text{для всіх} \quad x \in E^n$$