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

Кваліфікаційна робота

на здобуття ступеня вищої освіти «магістр»

«Дослідження збіжності двокрокових методів»

«Convergence study of two-term methods»

Виконав: здобувач денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»

Мацалишенко Павло Олександрович

Керівник
кандидат фіз-мат наук, доцент Яровий А.Т.
Рецензент
кандидат фіз-мат наук, доцент Страхов Є.М.

Рекомендовано до захисту:
Протокол засідання кафедри
№ ___ від _____2024 р.

Завідувач кафедри

_____- _____
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК №_____
протокол № ____ від _____ 2024 р.
Оцінка_____/___ ___/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____– _____–
(підпис) (прізвище, ініціали)

Одеса – 2024

ВСТУП

У дипломній роботі розглядається метод спряжених градієнтів та дві його модифікації. Основна мета дослідження — порівняти їх ефективність за ключовими критеріями, такими як збіжність, швидкість збіжності, час виконання та чутливість до початкових умов.

Робота складатиметься з трьох частин: теоретичної, практичної та висновків. У теоретичному розділі буде подано огляд усіх алгоритмів, включно з їх описом та теоретичним порівнянням. Також наведено загальні відомості про алгоритми та критерії зупинки, які використовуватимуться у дослідженні.

У практичній частині будуть представлені результати роботи програми для мінімізації функцій різних розмірностей. Для наочності будуть додані таблиці із значеннями та графіки. Для кожної функції розглядаються розрахунки з різними початковими даними, щоб зробити дослідження об’єктивнішим.

У висновках буде здійснено порівняння отриманих у практичному розділі результатів. Визначатиметься найкращий алгоритм за кожним із критеріїв, а також буде надано загальну оцінку кожного з методів.

Наприкінці роботи буде додано додатки з використаною літературою та реалізацією програми у вигляді програмного коду.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

1.1 Постановка задачі.

Дослідити метод спряжених градієнтів та його дві модифіковані версії. Реалізувати ці алгоритми у вигляді програмного забезпечення, що дозволить здійснювати мінімізацію заданих функцій та отримувати показники, необхідні для аналізу ефективності. Виконати порівняння методів, виявити їх переваги та недоліки, а також сформулювати висновки щодо кожного з них.

1.2 Загальні теоретичні відомості.

Метод спряжених градієнтів є технікою для визначення локального екстремуму функції, що базується на аналізі її значення та градієнта. Цей метод належить до категорії багатокрокових підходів. На відміну від стандартного градієнтного методу, який орієнтується лише на поточну інформацію без врахування даних попередніх ітерацій, метод спряжених градієнтів прагне використати "історію" процесу для підвищення швидкості збіжності. Разом з цим будемо використовувати наближення Гессіана.

У двох модифікаціях наближення Гессиана постійно оновлюється, що дозволяє наблизитися до квадратичної форми функції. У методі спряжених градієнтів такого явного оновлення Гессиана немає, і він використовує лише градієнт для побудови спряжених напрямків.

1.3 Реалізація методу спряжених градієнтів.

Алгоритм має вигляд:

$$x^{k+1} = x^k + \gamma_k \rho^k,$$

$$\gamma_k = arg\,min_\gamma f(x^k + \gamma\rho^k),$$

$$\rho^k = -r^k + \beta_k \rho^{k-1},\ \beta_k = \left\|r^k\right\|^2 / \left\|r^{k-1}\right\|^2,$$

$$r^k = \frac{\partial \varphi(x_k)}{\partial x},\ \beta_0 = 0.$$

1.4 Процедура відновлення.

Для неквадратичних задач метод спряжених градієнтів може бути адаптований в іншу форму. В такому випадку вводиться процедура відновлення, де крок не обчислюється за стандартною формулою, а визначається на основі антиградієнта, подібно до початкової точки. Такий підхід відновлення зазвичай здійснюється на кількість ітерацій, яка відповідає розміру простору.

$$x^{k+1} = x^k + \gamma_k \rho^k,$$

$$\gamma_k = arg\, min_\gamma f(x^k + \gamma \rho^k),$$

$$\rho^k = -r^k + \beta_k \rho^{k-1}, r^k = \frac{\partial \varphi(x_k)}{\partial x},$$

$$\beta_k: \ 0,\ k = 0,\ n,\ 2n,\ \ldots$$

$$\left\|r^k\right\| / \left\|r^{k-1}\right\|, k \neq 0,\ n,\ 2n,\ \ldots$$

Метод спряжених градієнтів з відновленням володіє властивістю глобальної збіжності, а в околі мінімуму він демонструє квадратичну швидкість збіжності.

Серед методів першого порядку метод спряжених градієнтів є найшвидшим з точки зору швидкості збіжності, особливо коли мова йде про задачі великої розмірності з квадратичними функціями.

1.5 Реалізація першої модифікації методу спряжених градієнтів.

Алгоритм має вигляд:

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

$$\alpha_k: \ min\, \varphi(x^k + \alpha s^k)$$

$$s^k = -\frac{\partial \varphi(x^k)}{\partial x} + \overline{\beta}_k H_{k-1}(x^k - x^{k-1}),\ s^0 = -\frac{\partial \varphi(x^0)}{\partial x}$$

$$\overline{\beta_k} = 0,\ k = 0,\ n,\ 2n,\ 3n,\ \ldots$$

$$\overline{\beta_k} = \beta_k,\ k \neq 0,\ n,\ 2n,\ 3n,\ \ldots$$

$$H_{k+1} = H_k + \frac{(\Delta x^k - H_k \Delta y^k)^T (\Delta x^k - H_k \Delta y^k)}{(\Delta x^k - H_k \Delta y^k, \Delta y^k)},\ H_0 = I$$

$$\Delta x^k = x^{k+1} - x^k$$

$$\Delta y^k = \varphi'(x^{k+1}) - \varphi'(x^k), \quad \text{де } \varphi'(x) = \frac{\partial \varphi(x)}{\partial x}$$

1.6 Реалізація другої модифікації методу спряжених градієнтів.

Алгоритм має вигляд:

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

$$\alpha_k:\ min\ \varphi(x^k + \alpha s^k)$$

$$s^k = -H_k \frac{\partial \varphi(x^k)}{\partial x} + \overline{\beta}_k (x^k - x^{k-1}),\ s^0 = -\frac{\partial \varphi(x^0)}{\partial x}$$

$$\overline{\beta_k} = 0,\ k = 0,\ n,\ 2n,\ 3n,\ \ldots$$

$$\overline{\beta_k} = \beta_k,\ k \neq 0,\ n,\ 2n,\ 3n,\ \ldots$$

$$H_{k+1} = H_k + \frac{(\Delta x^k - H_k \Delta y^k)^T (\Delta x^k - H_k \Delta y^k)}{(\Delta x^k - H_k \Delta y^k, \Delta y^k)},\ H_0 = I$$

$$\Delta x^k = x^{k+1} - x^k$$

$$\Delta y^k = \varphi'(x^{k+1}) - \varphi'(x^k), \quad \text{де } \varphi'(x) = \frac{\partial \varphi(x)}{\partial x}$$

1.7 Відмінності модифікацій від методу спряжених градієнтів.

Модифікації використовують оновлення напрямку руху s^k із врахуванням додаткових параметрів, таких як матриця Гессе або її наближена оцінка H_k.

Вони можуть бути ефективнішими за класичний спряжений градієнт при нелінійних задачах, але складніші в реалізації через необхідність обчислення та зберігання H_k чи її наближень.

Головна відмінність полягає в способі побудови та оновлення напрямків руху: метод спряжених градієнтів працює в просторі спряжених векторів, тоді як модифікації базуються на матричних модифікаціях Гессе.

1.8 Критерії зупинки.

Під час роботи алгоритму необхідно визначити момент завершення процесу мінімізації. Для цього використовуються критерії зупинки, основна функція яких полягає в обмеженні обчислювальних витрат і ресурсів, що витрачаються на додаткові ітерації, коли досягнуто достатньо точного наближення до оптимального рішення.

Будемо використовувати критерій зупинки:

1) $\varphi(x^{k-1}) - \varphi(x^k) < \tau(1 + |\varphi(x^k)|)$,

2) $||x^{k-1} - x^k|| < \sqrt{\tau}(1 + ||x^k||)$,

3) $\left|\left|\frac{\partial\varphi(x^k)}{\partial x}\right|\right| \leq \sqrt[3]{\tau}(1 + |\varphi(x^k)|)$.

де τ - задана точність.

Вважатимемо, що досягнуто заданої точності наближення до точки мінімуму, якщо виконуються всі три умови критерію.

РОЗДІЛ 2. ПРАКТИЧНА ЧАСТИНА.

2.1 Передмова

У практичній частині ми подивимося на результати роботи реалізованих алгоритмів на мові Python (Додаток А). Буде приведено таблиці з такими параметрами та результатами обчислень: початкова точка, знайдена точка мінімуму, кількість ітерацій та час виконання програми. Також для наглядності буде розглянуто графік, на якому буде візуалізоване відношення точність знайденого мінімуму до кількості ітерацій. В якості точності використаємо таку метрику:

$$ln\,|f(x_i) - f(x^*)|$$

Також для наглядності буде розглянута таблиця, де будуть зазначені знайдені точності для кожної початкової точки та для кожного методу.
На основі них зможемо зручно визначити який метод є найбільш точним.

2.2 Практичні результати

Функція №1 (Функція Розенброка). $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,\ n = 2$

Таблиця 2.2.1. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(-1.2, 1)	(1, 1)	1.00007648 1.00015328	17	0.708
2			1.00168167 1.00337303	231	9.004
3			0.99792372 0.99583786	74	2.549

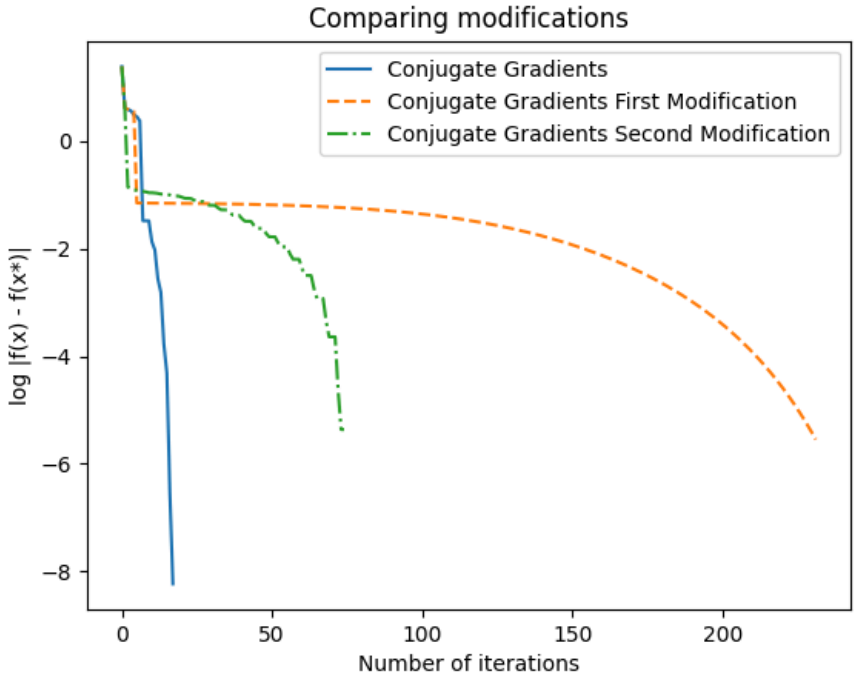

Рис. 2.2.1 ***Графік збіжності методів***

Таблиця 2.2.2. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(0, 0)	(1, 1)	0.99999963 0.99999926	21	0.700
2			1.00257756 1.00517179	157	6.013
3			1.01065163 1.02145197	371	13.620

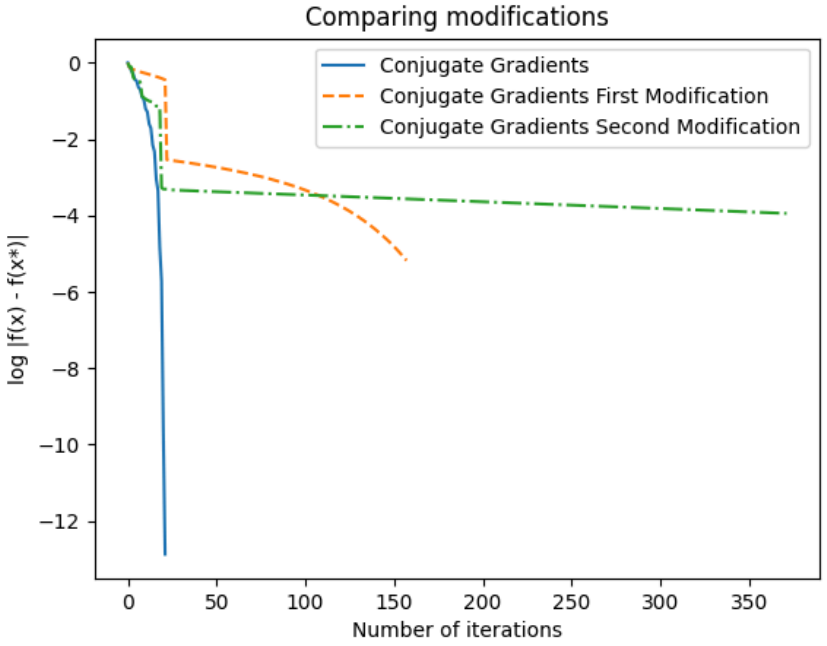

Рис. 2.2.2 ***Графік збіжності методів***

Функція №2 (Розширена функція Уайта та Хольста). $f(x) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2,\ n = 2$

Таблиця 2.2.3. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-1.2, 1)	(1, 1)	0.99994347 0.99983026	17	0.732c
2			1.00187834 1.00565121	75	2.994c
3			0.98917591 0.96783719	24	0.906c

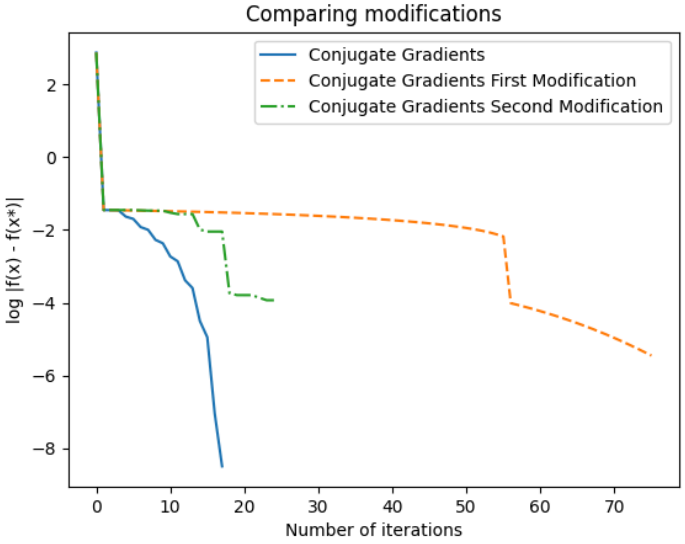

Рис. 2.2.3 ***Графік збіжності методів***

Таблиця 2.2.4. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(0, 0)	(1, 1)	0.99991620 0.99974837	27	0.986
2			0.99772132 0.99317274	833	32.335
3			0.98870229 0.96644283	745	28.824

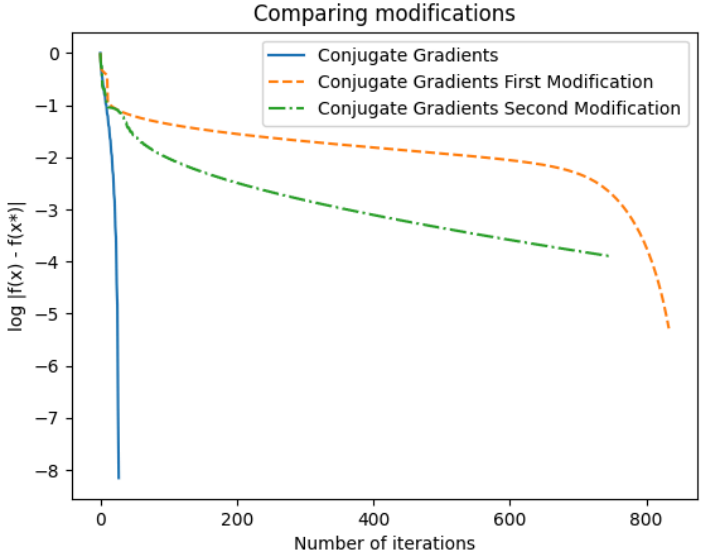

Рис. 2.2.4 ***Графік збіжності методів***

Таблиця 2.2.5. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-5, 5)	(1, 1)	1.00013840 1.00041566	39	1.77c
2			1.691 4.839	3000(max)	145c
3			1.505 3.415	3000(max)	127c

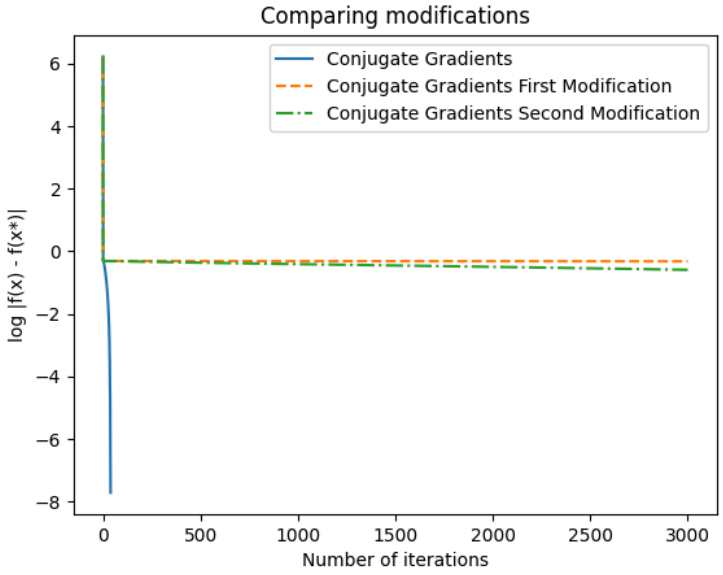

Рис. 2.2.5 ***Графік збіжності методів***

Функція №3 (Функція Хіммельблау). $f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$, $n = 2$

Таблиця 2.2.6. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(0,0)	(3, 2)	2.99999999 2.00000006	9	0.44c
2			2.99998069 2.00011145	14	0.67c
3			2.99999296 1.99994702	12	0.55c

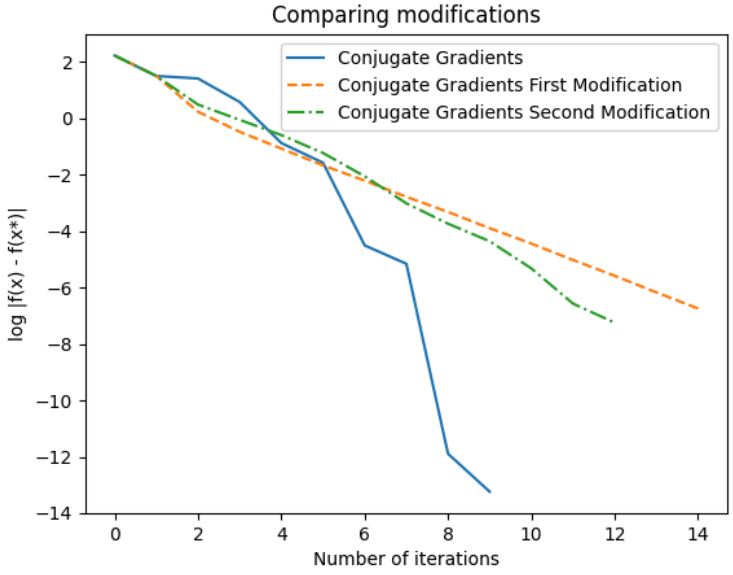

Рис. 2.2.6 ***Графік збіжності методів***

Таблиця 2.2.7. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(1, 1)	(3, 2)	2.9999995 2.0000001	7	0.31
2			2.99998181 2.00003365	9	0.39
3			2.9999996 2.000011	10	0.41

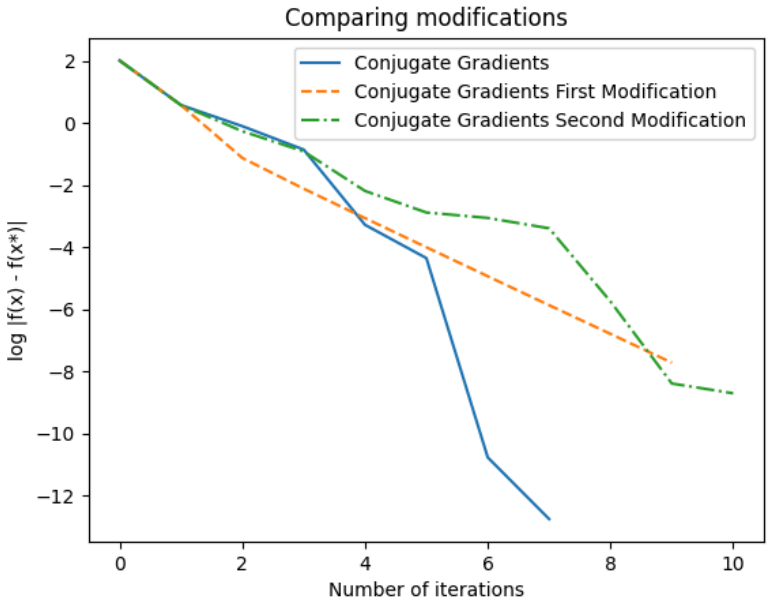

Рис. 2.2.7 ***Графік збіжності методів***

Таблиця 2.2.8. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(15, 15)	(-3.77, -3.28)	-3.779309 -3.283187	5	0.21
2			-3.779303 -3.283177	6	0.25
3			-3.779301 -3.283174	6	0.24

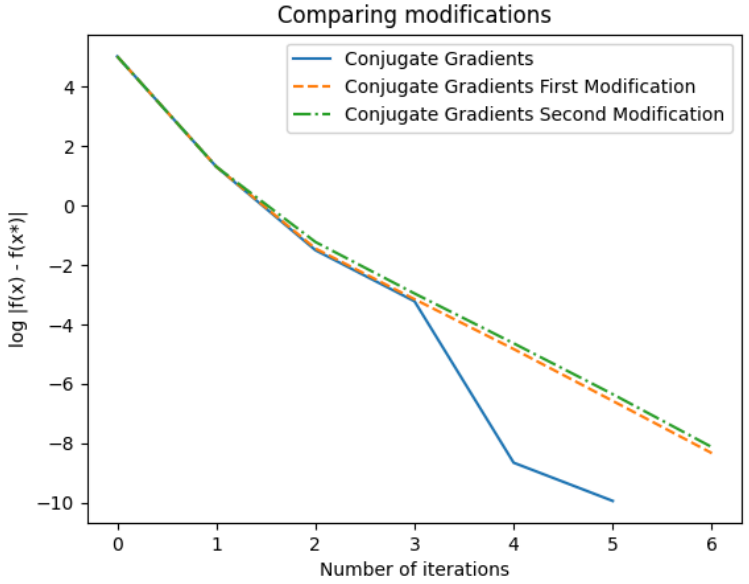

Рис. 2.2.8 ***Графік збіжності методів***

Таблиця 2.2.9. ***Основні параметри***

№	x_0	x^*	Наближення ($e = 10^{-10}$)	Ітерацій	t, c
1	(15, 15)	(-3.77, -3.28)	-3.779310 -3.283185	7	0.25
2			-3.779310 -3.283185	8	0.32
3			-3.779310 -3.283185	9	0.34

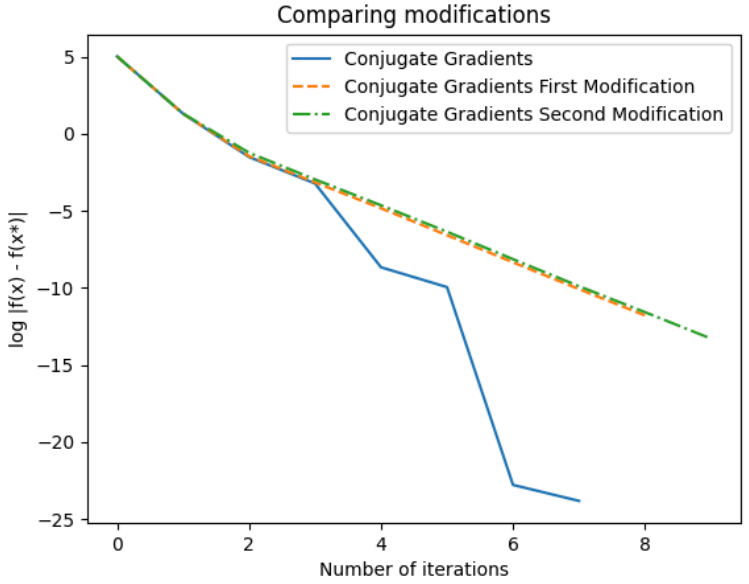

Рис. 2.2.9 ***Графік збіжності методів***

Функція №4. $f(x) = (1.5 - x_1(1 - x_2))^2 + (2.25 - x_1(1 - x^2_2))^2 + (2.625 - x_1(1 - x^3_2))^2,\ n = 2$

Таблиця 2.2.10. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(2, 0.2)	(3, 0.5)	3.00000003 0.49999988	8	0.847c
2			2.99903910 0.49975972	65	6.208c
3			3.02318468 0.50571225	22	2.164c

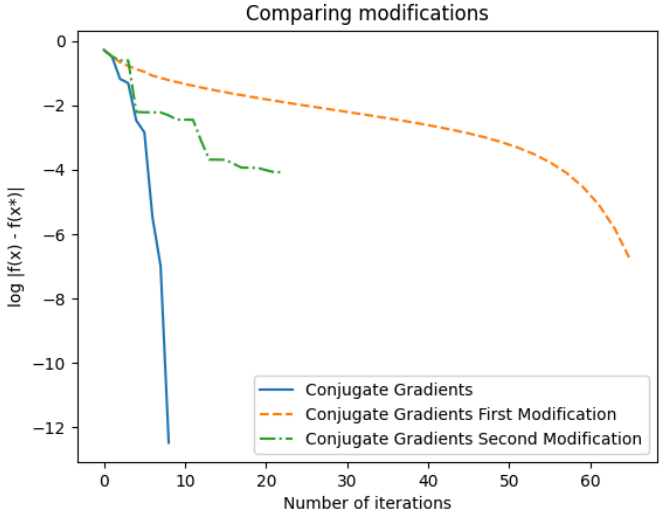

Рис. 2.2.10 ***Графік збіжності методів***

Таблиця 2.2.11. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(0, 0)	(3, 0.5)	2.99981835 0.499954646	9	0.768
2			2.98857411 0.497098856	63	5.821
3			2.99318332 0.49831762	50	4.701

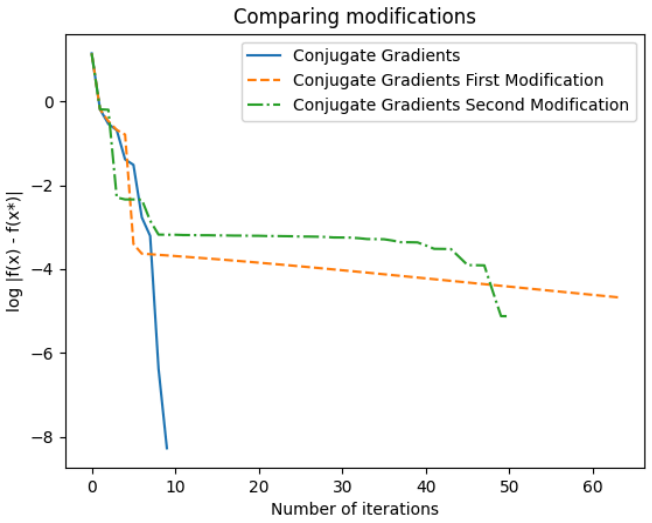

Рис. 2.2.11 ***Графік збіжності методів***

Функція №5. $f(x) = x_1^2 + 20x_2 + 125(x_3 - 1)^2$, $n = 3$

Таблиця 2.2.12. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-1, -1, 2)	(0, 0, 1)	-0.00000001 0.00000002 1.00000000	4	0.148c
2			-0. 0040581 0.0001075 1.00001526	264	5.86c
3			-0. 00392378 0.00000076 0.99989713	400	9.39c

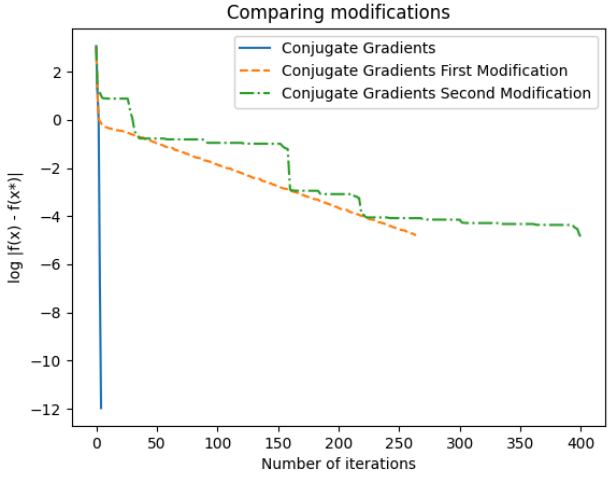

Рис. 2.2.12 ***Графік збіжності методів***

Таблиця 2.2.13. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(1, 1, -1)	(0, 0, 1)	$1.10e^{-15}$ $1.12e^{-14}$ 1. 0000000..	4	0.088
2			0. 00395036 -0. 00010340 0. 99998546	242	5.13
3			0.00448912 0. 00006253 0. 99998587	878	20.67

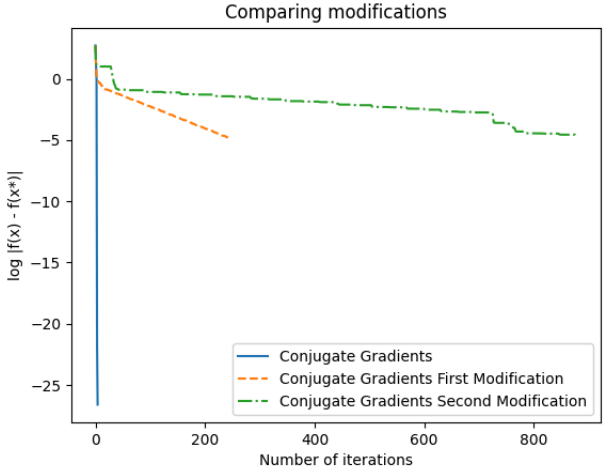

Рис. 2.2.13 ***Графік збіжності методів***

Таблиця 2.2.14. ***Основні параметри***

№	x_0	x^*	Наближення ($e = 10^{-10}$)	Ітерацій	t, c
1	(-1, -1, -2)	(0, 0, 1)	$-1.55e^{-7}$ $-2.76e^{-10}$ 1.00000..	5	0.105c
2			-0.00018410 0.00000485 1.00000069	414	9.24c
3			-0.00017611 0.00000493 1.00000038	1309	31.15c

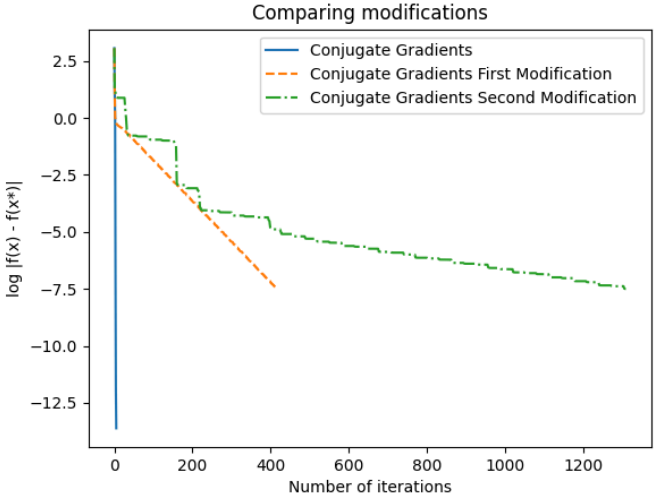

Рис. 2.2.14 ***Графік збіжності методів***

Таблиця 2.2.15. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-10, -10, -10)	(0, 0, 1)	$-1.55e^{-7}$ $3.69e^{-11}$ 1.000000	5	0.11
2			0.00397741 0.00010427 1.00001462	282	8.65
3			-0.0037919 0.0001091 1.00001146	1391	33.34

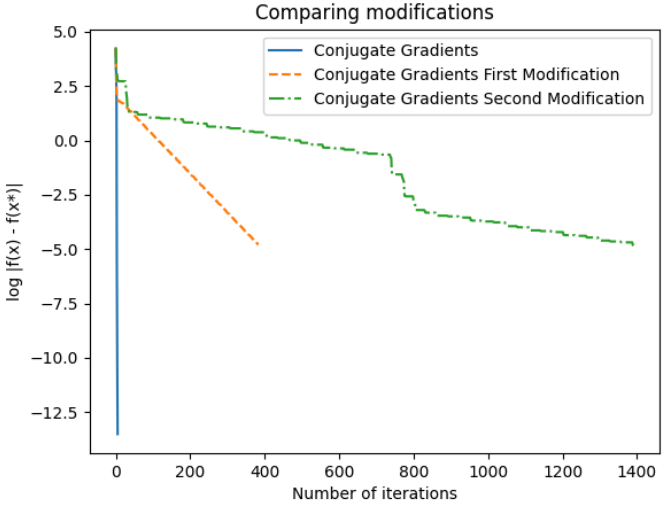

Рис. 2.2.15 ***Графік збіжності методів***

Функція №6. $f(x) = 100(x_3 - (\frac{x_1+x_2}{2})^2)^2 + (1-x_1)^2 + (1-x_2)^2,\ n = 3$

Таблиця 2.2.16. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-1.2, 2, 0)	(1, 1, 1)	0.99773480 0.99413095 0.99184485	19	0.90c
2			0.99999449 0.99999708 0.99999148	260	12.35c
3			0.99807257 1.0017994 0.9998749	532	24.89c

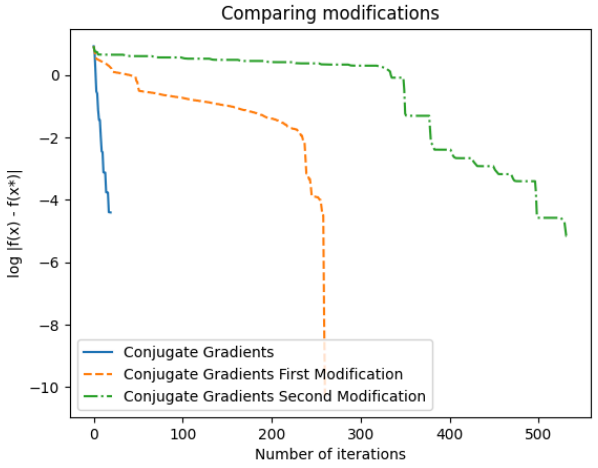

Рис. 2.2.16 ***Графік збіжності методів***

Таблиця 2.2.17. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-10, -10, -10)	(1, 1, 1)	0.99997613 0.99997618 0.99938160	18	0.717
2			0.9942746 0.99427246 0.98854901	514	21.30
3			0.99401674 0.99401674 0.98803086	354	15.17

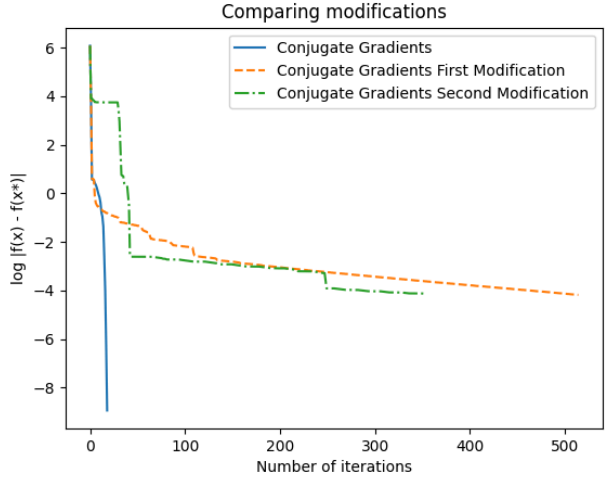

Рис. 2.2.17 ***Графік збіжності методів***

Таблиця 2.2.18. ***Основні параметри***

№	x_0	x^*	Наближення ($e = 10^{-10}$)	Ітерацій	t, c
1	(-1.2, 2, 0)	(1, 1, 1)	0.99999989 0.99999998 0.99999998	34	1.57c
2			0.99999994 0.99999782 0.99999149	261	12.25c
3			0.99981135 1.00005835 0.99986929	681	31.73c

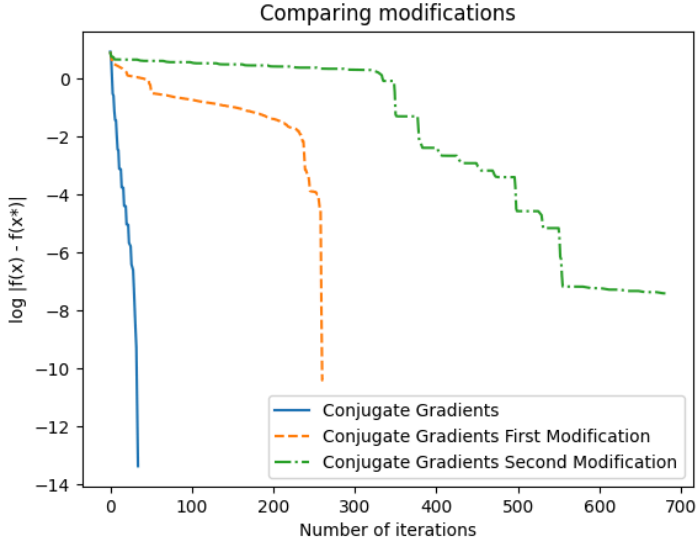

Рис. 2.2.18 ***Графік збіжності методів***

Таблиця 2.2.19. ***Основні параметри***

№	x_0	x^*	Наближення ($e = 10^{-10}$)	Ітерацій	t, c
1	(-10, -10, -10)	(1, 1, 1)	0.99997632 0.99996732 0.99995244	19	0.80
2			0.99973724 0.99973734 0.99947285	860	36.11
3			0.9998995 0.99989504 0.99978829	1642	68.88

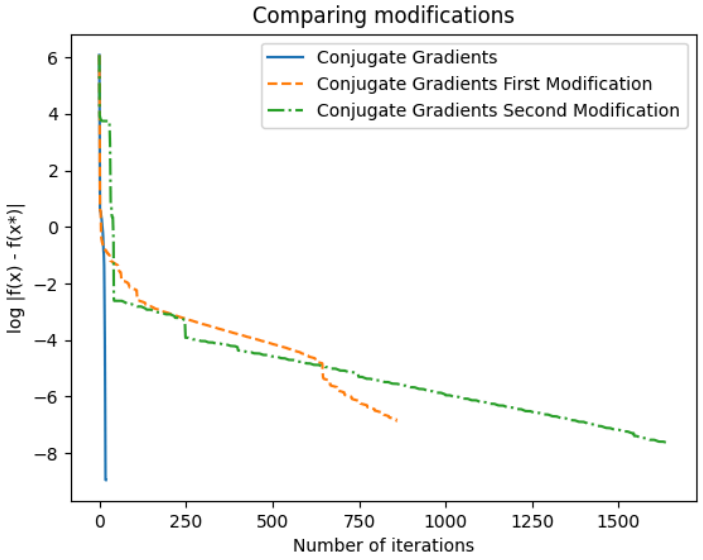

Рис. 2.2.19 ***Графік збіжності методів***

Функція №7. $f(x) = (5x_1 + 100x_2^2 - x_3 - 9)^2 + (x_2 - x_3^3 + 1)^2 + (x_1 + 2x_2 - 2)^2,\ n = 3$

Таблиця 2.2.19. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(0, 0, 0)	(2, 0, 1)	2.00000921 -0.00017734 0.99996587	32	2.12c
2			1.99990431 0.00048157 1.00014364	126	7.56c
3			1.99987731 0.00021929 0.99997936	1926	109.99c

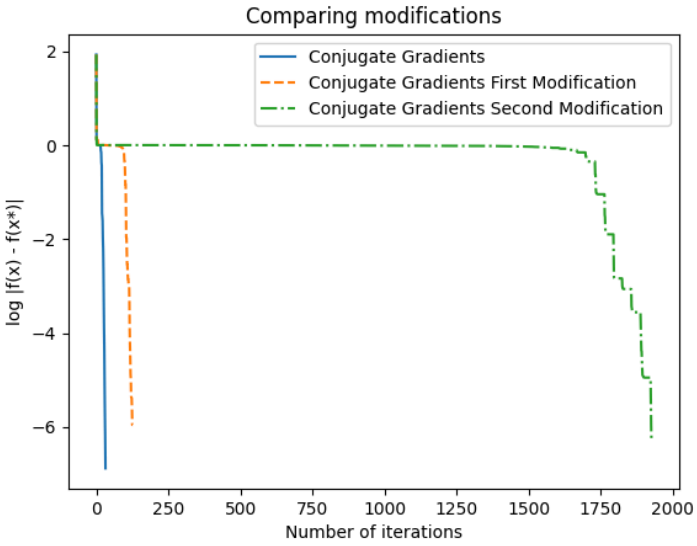

Рис. 2.2.19 ***Графік збіжності методів***

Таблиця 2.2.20. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-1, -1, -1)	(2, 0, 1) (1.793, 0.103, 1.033)	1.79277497 0.10342846 1.03353028	74	4.44
2			1.99997146 0.00077142 1.00003219	2177	134.05
3			1.766270 0.109815 1.035803	5000 (max)	295

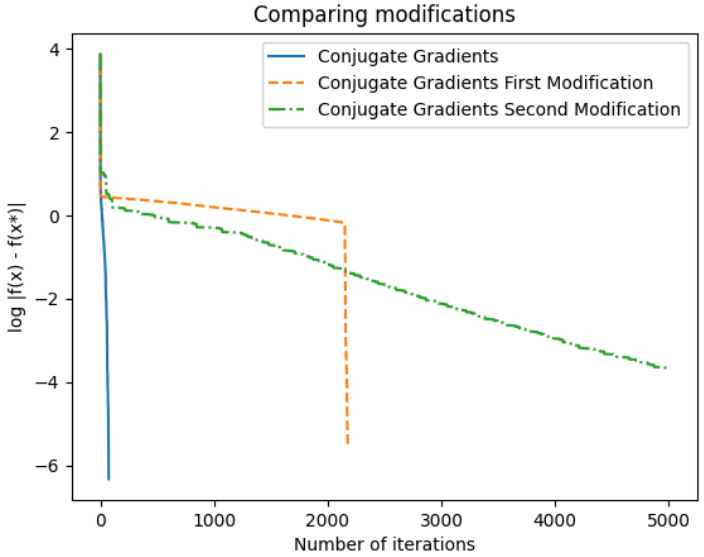

Рис. 2.2.20 ***Графік збіжності методів***

Таблиця 2.2.21. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-5, -5, -5)	(2, 0, 1) (1.793, 0.103, 1.033)	1.99999813 -0.0000730 0.99995962	17	1.01c
2			1.62679281 0.13857358 1.04543045	5000 (max)	295.68c
3			1.74780382 0.11404728 1.03728954	5000 (max)	300.36c

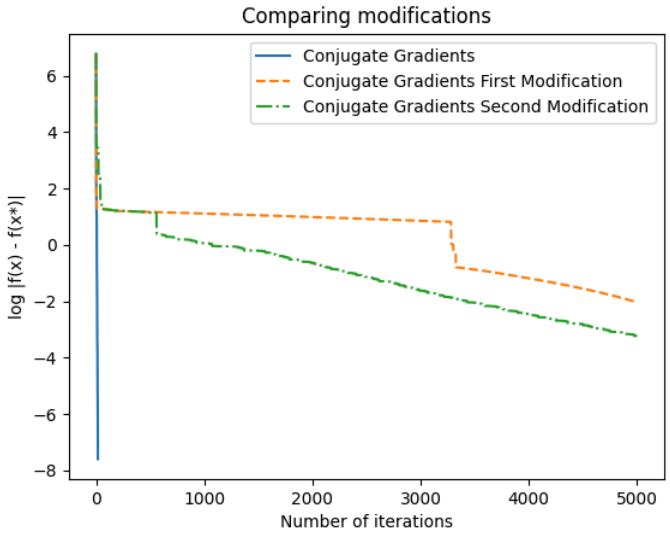

Рис. 2.2.21 ***Графік збіжності методів***

Таблиця 2.2.22. ***Основні параметри***

№	x_0	x^*	Наближення ($e = 10^{-10}$)	Ітерацій	t, c
1	(0, 0, 0)	(2, 0, 1)	2.00000003 -0.00000001 1.00000012	34	1.92
2			1.99999654 0.00002045 0.99999878	138	8.31
3			1.99999505 0.00002457 1.00001324	1997	117.83

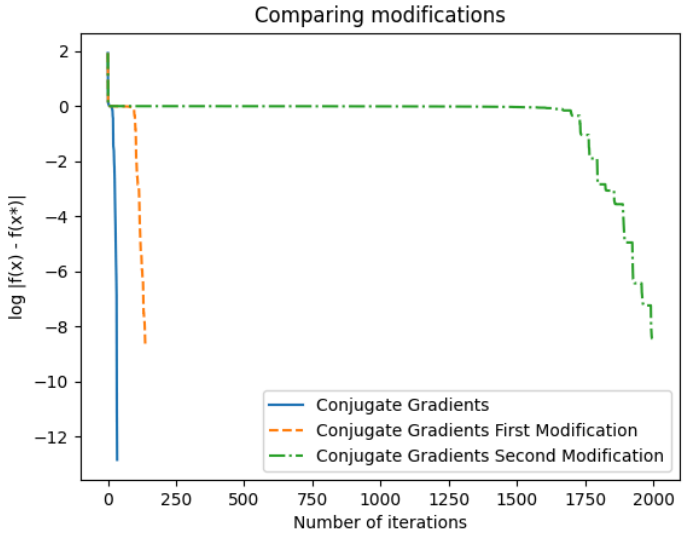

Рис. 2.2.22 ***Графік збіжності методів***

Функція №8 (Функція Пауела). $f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 3x_3)^2 + 10(x_1 - x_4)^2$, $n = 4$, $e = 10^{-6}$

Таблиця 2.2.23. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(1, -2, 3, 0.5)	(0, 0, 0, 0)	0.000000006 -0.00000006 0.00000002 0.00000001	5	0.26
2			0.00112125 -0.00011549 -0.00002906 -0.00082611	146	5.35
3			0.0015424 -0.00015967 0.00018677 0.0011934	450	17.02

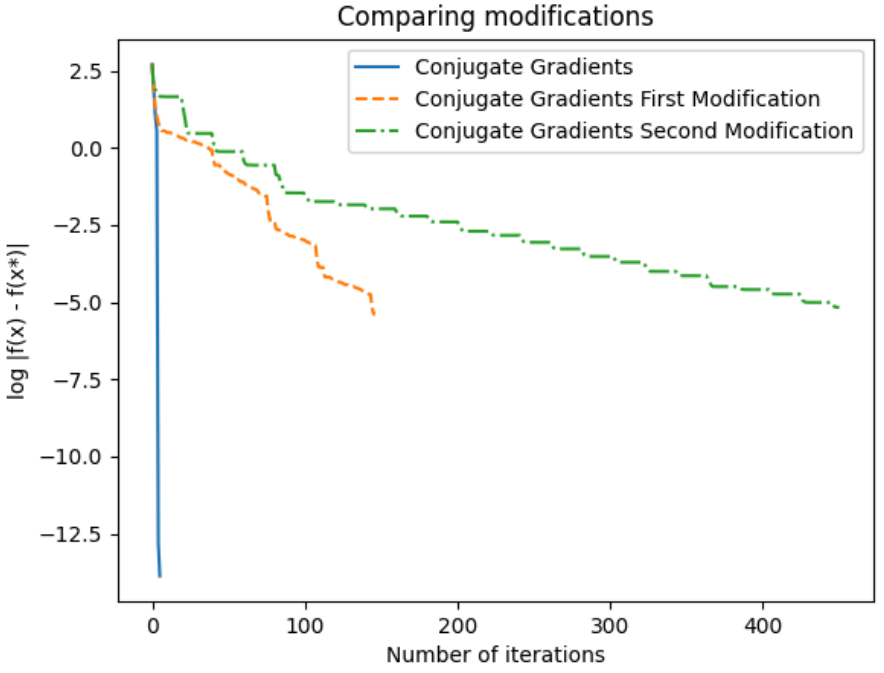

Рис. 2.2.23 ***Графік збіжності методів***

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 3x_3)^2 + 10(x_1 - x_4)^2,\ n = 4,\ e = 10^{-10}$$

Таблиця 2.2.24. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(1, -2, 3, 0.5)	(0, 0, 0, 0)	0.00000006 -0.00000006 0.000000002 0.000000001	5	0.18
2			-0.00006794 0.0000007 -0.0000207 0.0000579	224	8.35
3			0.0000579 -0.00000059 0.0000007 0.00004507	715	26.49

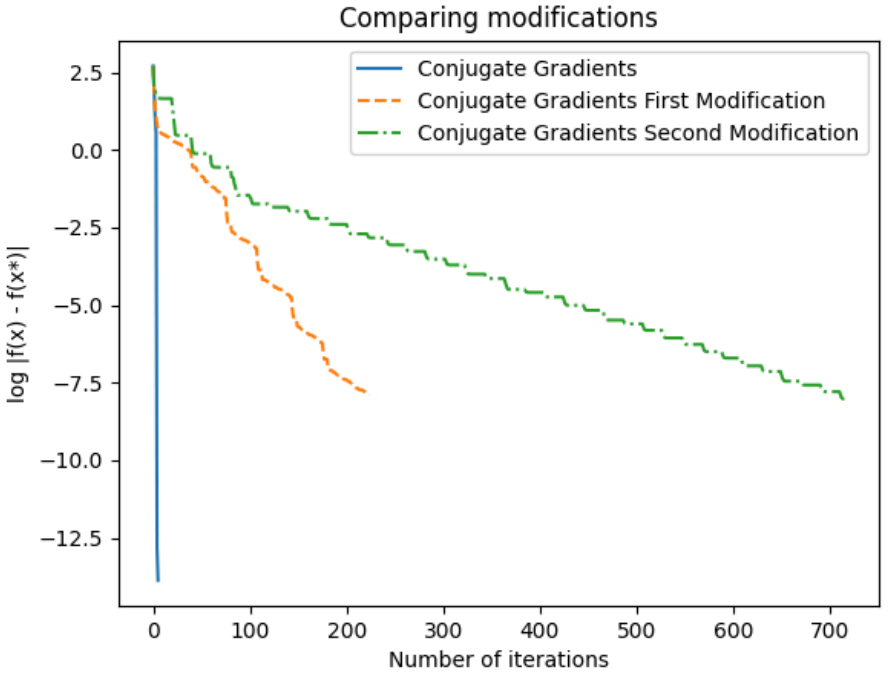

Рис. 2.2.24 ***Графік збіжності методів***

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 3x_3)^2 + 10(x_1 - x_4)^2,\ n = 4,\ e = 10^{-15}$$

Таблиця 2.2.25. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(1, -2, 3, 0.5)	(0, 0, 0, 0)	0. 00000001 -0. 00000001 -0. 00000001 0. 00000001	8	0.33
2			-0.000001 0. 00000017 0. 0000003 0. 00000012	363	13.32
3			0. 000001 -0. 00000001 -0. 00000001 0. 000000012	1025	38.09

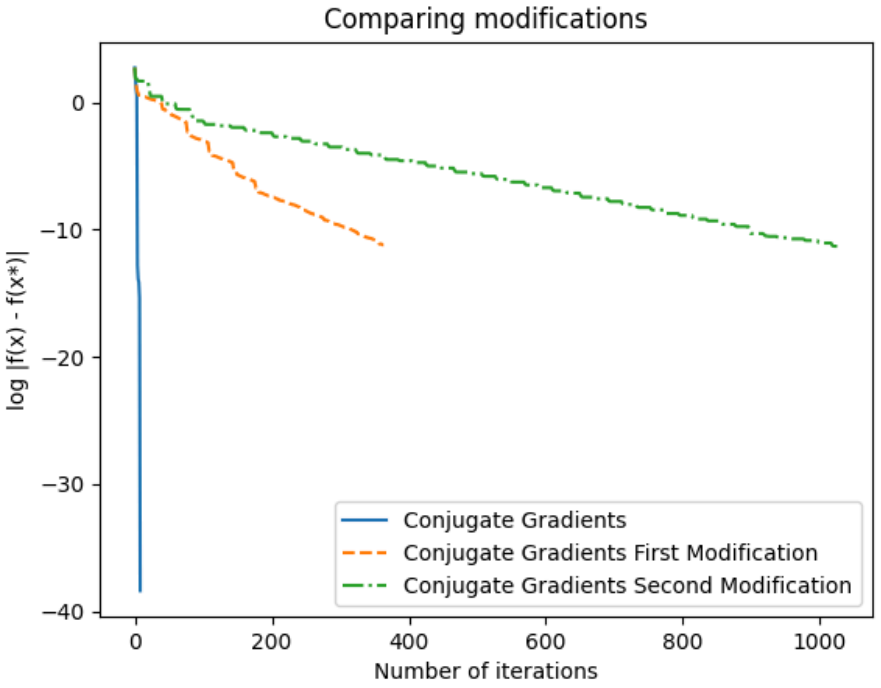

Рис. 2.2.25 ***Графік збіжності методів***

Функція №9. $f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4,\ n = 4,\ e = 10^{-6}$

Таблиця 2.2.26. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(3, -1, 0, 1)	(0, 0, 0, 0)	0.0489795 -0.0048812 0.01134547 0.01112173	41	1.97
2			0.10838716 -0.01083935 0.05137127 0.05232175	732	33.11
3			0.09199504 -0.00918570 0.03146080 0.03205371	984	44.03

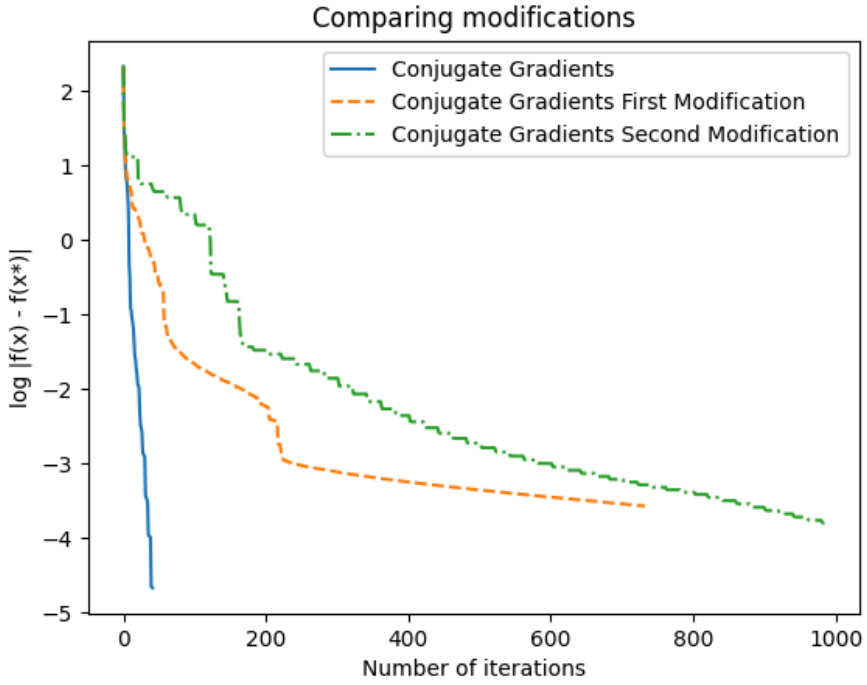

Рис. 2.2.26 ***Графік збіжності методів***

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4,\ n = 4,\ e = 10^{-6}$$

Таблиця 2.2.27. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(10, -10, 10, -10)	(0, 0, 0, 0)	0.01129605 -0.0011225 0.00457334 0.00446107	16	0.69
2			-0.01877962 0.00188691 -0.03703655 -0.03683436	154	6.85
3			0.09049806 0.00905704 0.03038745 0.03107999	867	38.40

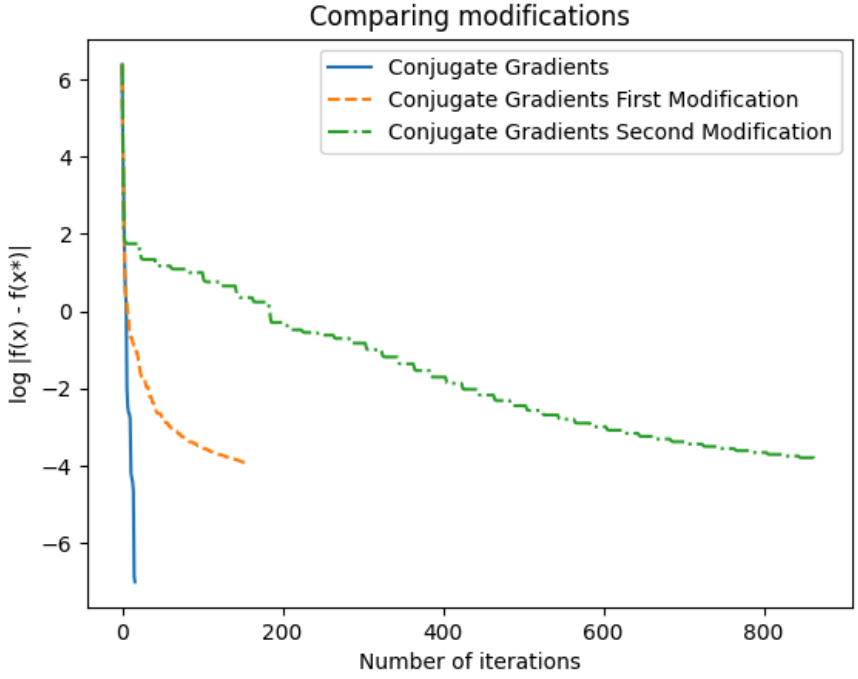

Рис. 2.2.27 ***Графік збіжності методів***

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4,\ n = 4,\ e = 10^{-10}$$

Таблиця 2.2.28. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(3, -1, 0, 1)	(0, 0, 0, 0)	0. 0005607 -0. 0000560 -0. 0009170 0. 00091735	53	2.23
2			0. 04160256 -0. 00416169 0. 01940195 0. 0194555	5000	229.06
3			0. 0084802 -0. 00084889 0. 0176077 0. 01762501	2492	110.40

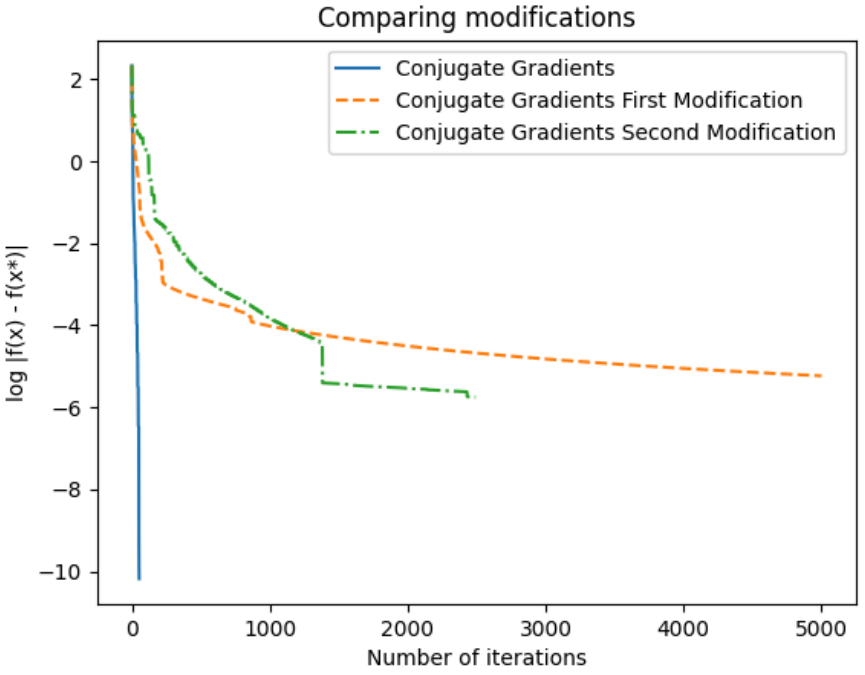

Рис. 2.2.28 ***Графік збіжності методів***

Функція №10. $f(x) = x_1^{\,2} + 3x_2^{\,2} + 8x_3^{\,4} + 4x_4^{\,4} - 2x_1 - 6x_2 - 32x_3 - 4x_4,\ n = 4,\ e = 10^{-8}$

Таблиця 2.2.29. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(0, 0, 0, 0)	(1, 1, 1, 1)	1.00014398 1.00001022 1.00000144 1.000000002	12	0.52
2			0.97140923 1.0024890 1.00033899 1.0008293	51	1.93
3			0.98123314 1.0011009 1.0004546 0.9992633	136	5.09

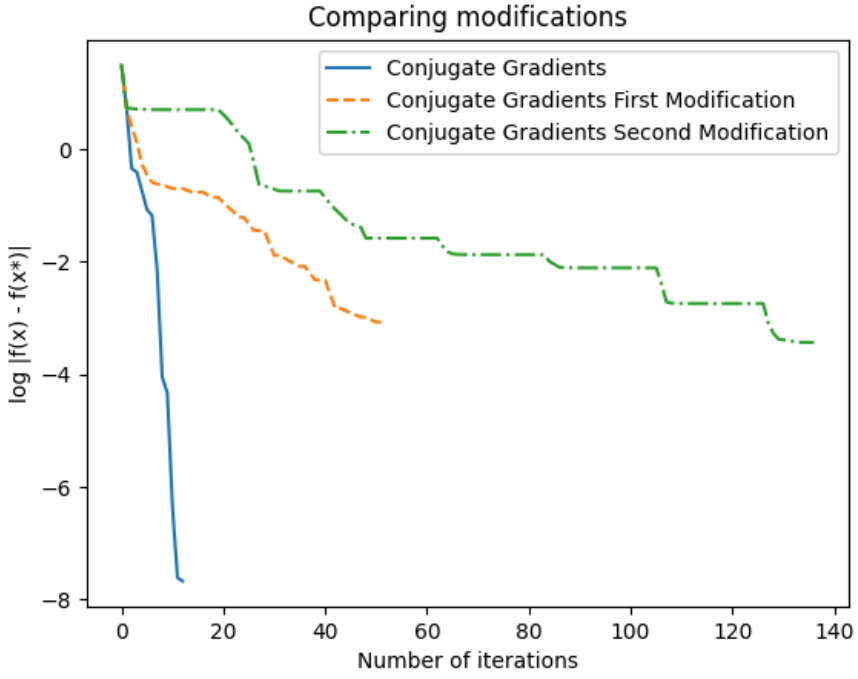

Рис. 2.2.29 ***Графік збіжності методів***

$$f(x) = x_1^{\,2} + 3x_2^{\,2} + 8x_3^{\,4} + 4x_4^{\,4} - 2x_1 - 6x_2 - 32x_3 - 4x_4,\ n = 4,\ e = 10^{-8}$$

Таблиця 2.2.30. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-2, -2, -2, -2)	(1, 1, 1, 1)	0.99999990 0.99999993 1.00 1.0000000031	17	0.68
2			0.98070762 1.00192821 1.000517 0.99810972	55	2.14
3			0.98701742 1.00198903 1.00017392 1.00017007	206	7.87

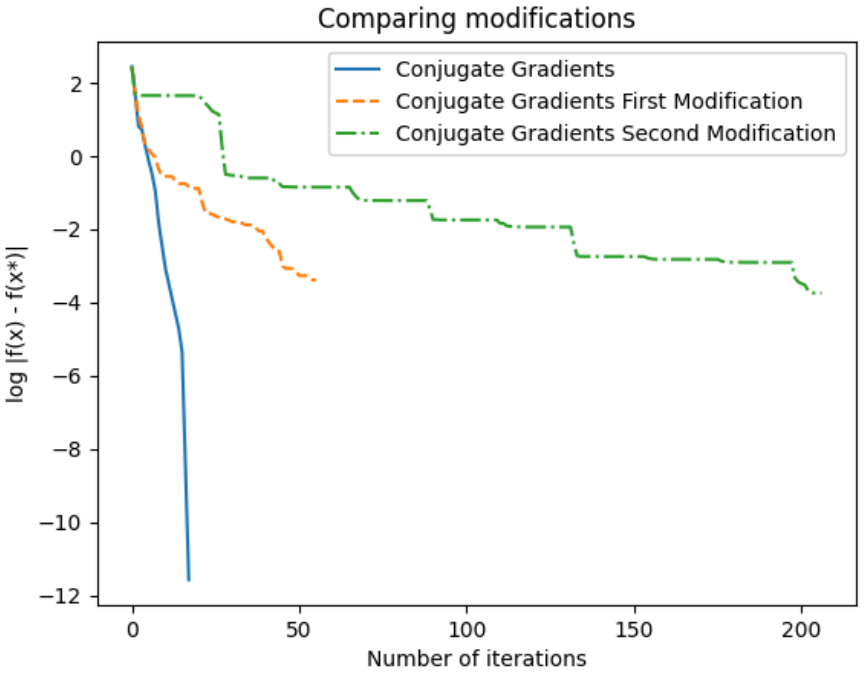

Рис. 2.2.30 ***Графік збіжності методів***

$$f(x) = x_1^{\,2} + 3x_2^{\,2} + 8x_3^{\,4} + 4x_4^{\,4} - 2x_1 - 6x_2 - 32x_3 - 4x_4,\ n = 4,\ e = 10^{-12}$$

Таблиця 2.2.31. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(0, 0, 0, 0)	(1, 1, 1, 1)	0.99999995 1.00000001 0.99999993 1.00000006	16	0.66
2			0.99876636 1.00014482 1.00000954 1.00005685	91	3.66
3			0.9991171 1.0002144 0.9999924 1.0001017	285	11.31

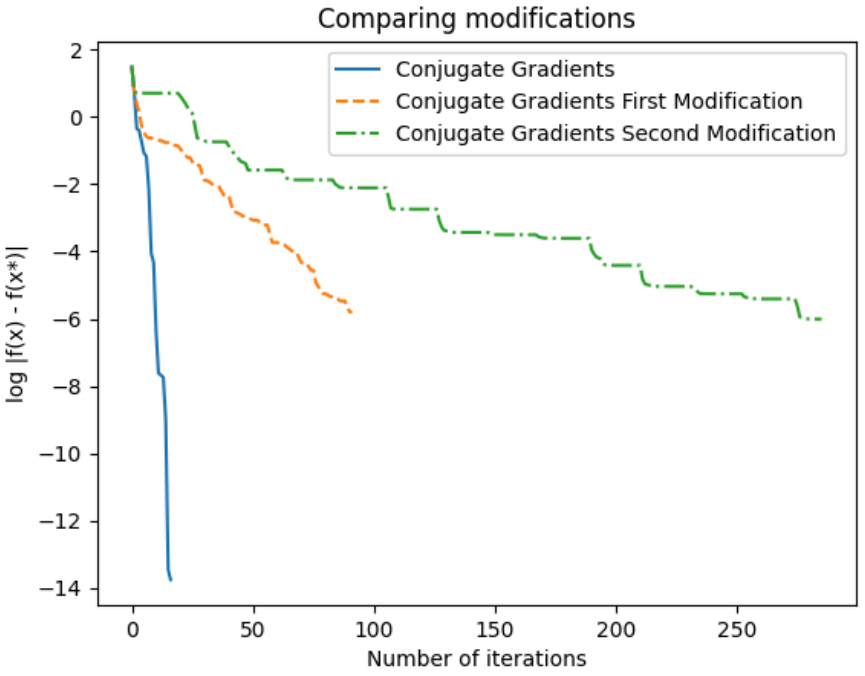

Рис. 2.2.31 ***Графік збіжності методів***

Функція №11. $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 100(x_4 - x_3^2)^2 + (1 - x_3)^2 + 100(x_6 - x_5^2)^2 + (1 - x_5)^2, n = 6$

Таблиця 2.2.32. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-1.2, 1, -1.2, 1, -1.2, 1)	(1, 1, 1, 1, 1, 1)	1.00162076 1.00325236 1.00162076 1.00325236 1.00162076 1.00325236	23	1.15
2			0.99373583 0.98748799 0.99373583 0.98748799 0.99373583 0.98748799	930	48.10
3			0.99374922 0.98751494 0.99374922 0.98751494 0.99374922 0.98751494	1843	122.63

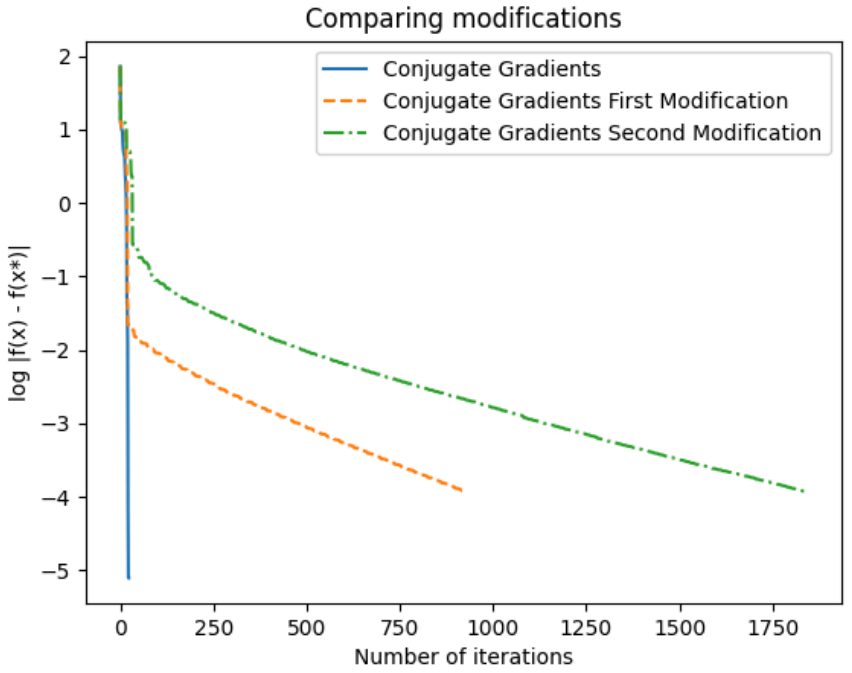

Рис. 2.2.32 ***Графік збіжності методів***

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 100(x_4 - x_3^2)^2 + (1 - x_3)^2 + 100(x_6 - x_5^2)^2 + (1 - x_5)^2,\ n = 6$$

Таблиця 2.2.33. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(-2, 0, -2, 0, -2, 0)	(1, 1, 1, 1, 1, 1)	1.00099485 1.00199428 1.00099485 1.00199428 1.00099485 1.00199428	6	0.36
2			0.99717659 0.99435082 0.99717659 0.99435082 0.99717659 0.99435082	1284	64.95
3			0.99719875 0.99439499 0.99719875 0.99439499 0.99719875 0.99439499	923	46.08

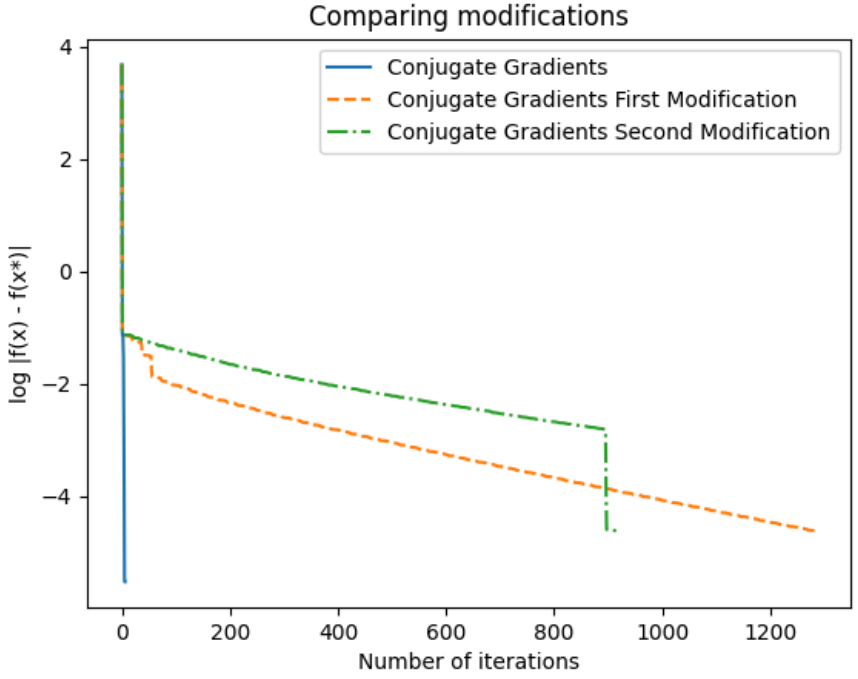

Рис. 2.2.33 ***Графік збіжності методів***

Функція №12. $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 100(x_3 - x_2^2)^2 + (1 - x_2)^2 + 100(x_4 - x_3^2)^2 + (1 - x_3)^2 + 100(x_5 - x_4^2)^2 + (1 - x_4)^2 + 100(x_6 - x_5^2)^2 + (1 - x_5)^2, n = 6, e = 10^{-6}$

Таблиця 2.2.34. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-1.2, 1, -1.2, 1, -1.2, 1)	(1, 1, 1, 1, 1, 1)	0.99995793 0.99992001 0.99983483 0.99965191 0.99928897 0.99857625	68	8.76
2			0.99948988 0.99898020 0.99896065 0.99591756 0.99183540 0.98369854	1680	246.61
3			0.99981453 0.99962462 0.99924705 0.99848890 0.99697152 0.99393479	228	30.07

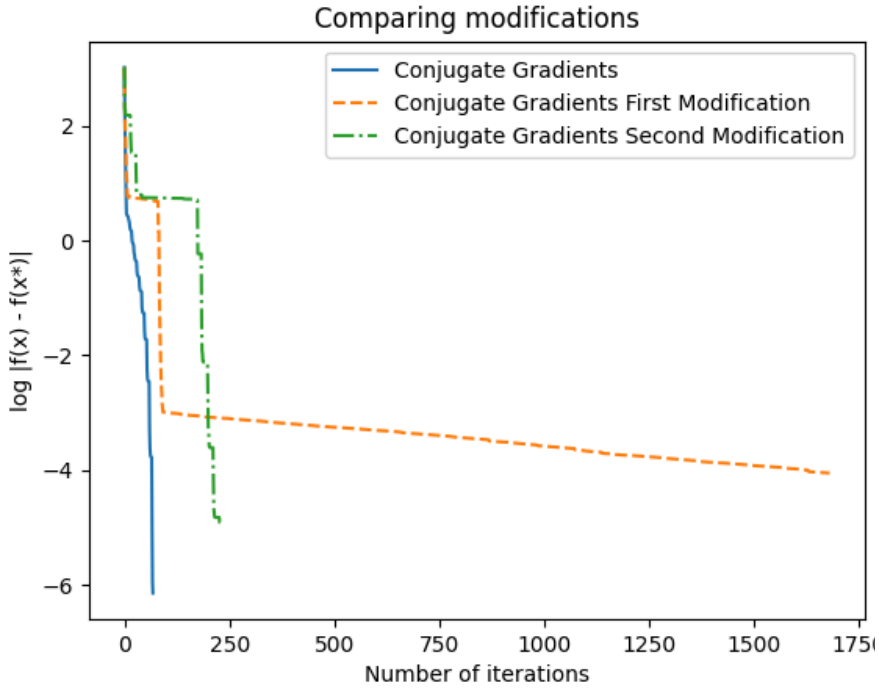

Рис. 2.2.34 ***Графік збіжності методів***

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 100(x_3 - x_2^2)^2 + (1 - x_2)^2 + 100(x_4 - x_3^2)^2 + (1 - x_3)^2 + 100(x_5 - x_4^2)^2 + (1 - x_4)^2 + 100(x_6 - x_5^2)^2 + (1 - x_5)^2,\ n = 6,\ e = 10^{-8}$$

Таблиця 2.2.35. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(-1.2, 1, -1.2, 1, -1.2, 1)	(1, 1, 1, 1, 1, 1)	0.99995759 0.99991187 0.99982401 0.99964747 0.99929174 0.99858164	70	9.48
2			0.99989438 0.99978907 0.99957769 0.99915434 0.99830570 0.99660648	3946	596.83
3			0.99988992 0.99977945 0.99955675 0.99911040 0.99821599 0.99642625	2828	393.33

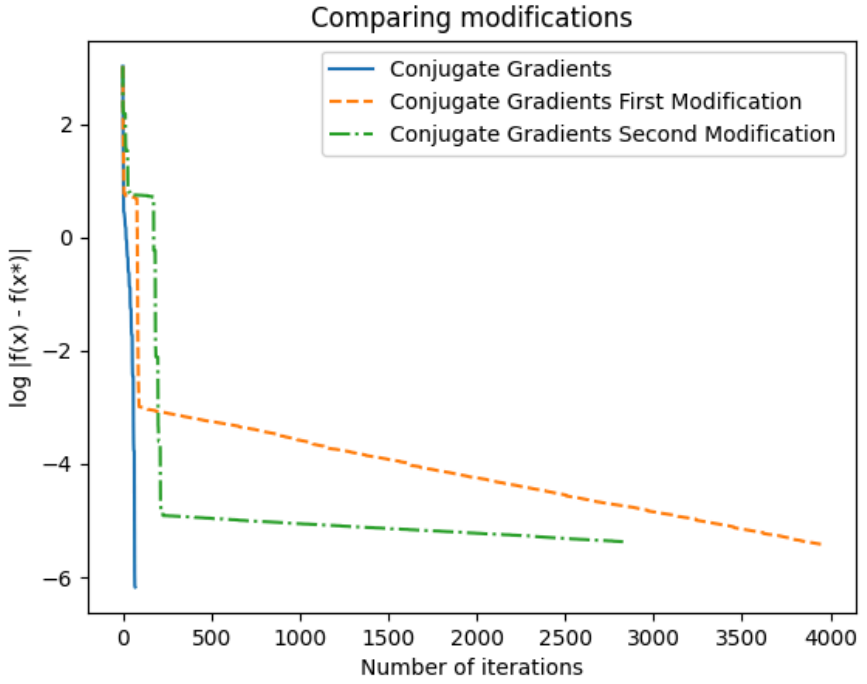

Рис. 2.2.35 ***Графік збіжності методів***

Функція №13. $f(x) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2 + 100(x_4 - x_3^3)^2 + (1 - x_3)^2 + 100(x_6 - x_5^3)^2 + (1 - x_5)^2$ $100(x_8 - x_7^3)^2 + (1 - x_7)^2$, $n = 8$, $e = 10^{-7}$

Таблиця 2.2.36. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(-1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1)	(1, 1, 1, 1, 1, 1, 1, 1)	1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000,	11	0.92
2			0.9892, 0.9681, 0.9892, 0.9681, 0.9892, 0.9681, 0.9892, 0.9681,	10001 (max)	756.34
3			1.0097, 1.0295, 1.0097, 1.0295, 1.0097, 1.0295, 1.0097, 1.0295,	10001 (max)	715.94

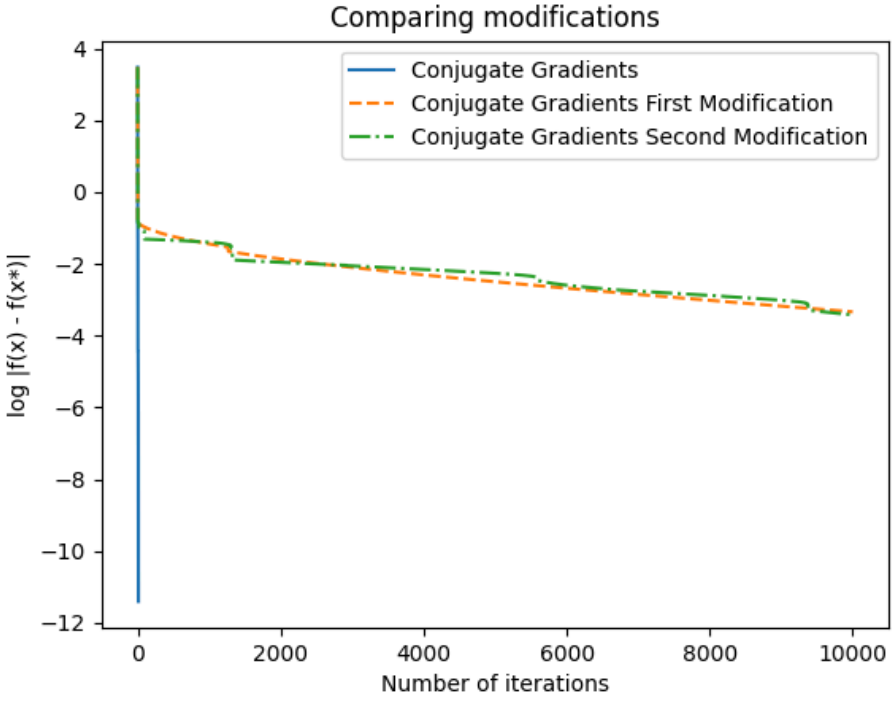

Рис. 2.2.36 ***Графік збіжності методів***

$$f(x) = 100(x_2 - x_1{}^3)^2 + (1 - x_1)^2 + 100(x_4 - x_3{}^3)^2 + (1 - x_3)^2 + 100(x_6 - x_5{}^3)^2 + (1 - x_5)^2$$
$$100(x_8 - x_7{}^3)^2 + (1 - x_7)^2,\ n = 8,\ e = 10^{-6}$$

Таблиця 2.2.37. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(1, 0, 1, 0, 1, 0, 1, 0)	(1, 1, 1, 1, 1, 1, 1, 1)	0.998, 0.9942, 0.998, 0.9942, 0.998, 0.9942, 0.998, 0.9942,	14	0.91
2			0.9921, 0.9765, 0.9921, 0.9765, 0.9921, 0.9765, 0.9921, 0.9765,	10001 (max)	755.42
3			0.9927, 0.9784, 0.9927, 0.9784, 0.9927, 0.9784, 0.9927, 0.9784,	2989	280.29

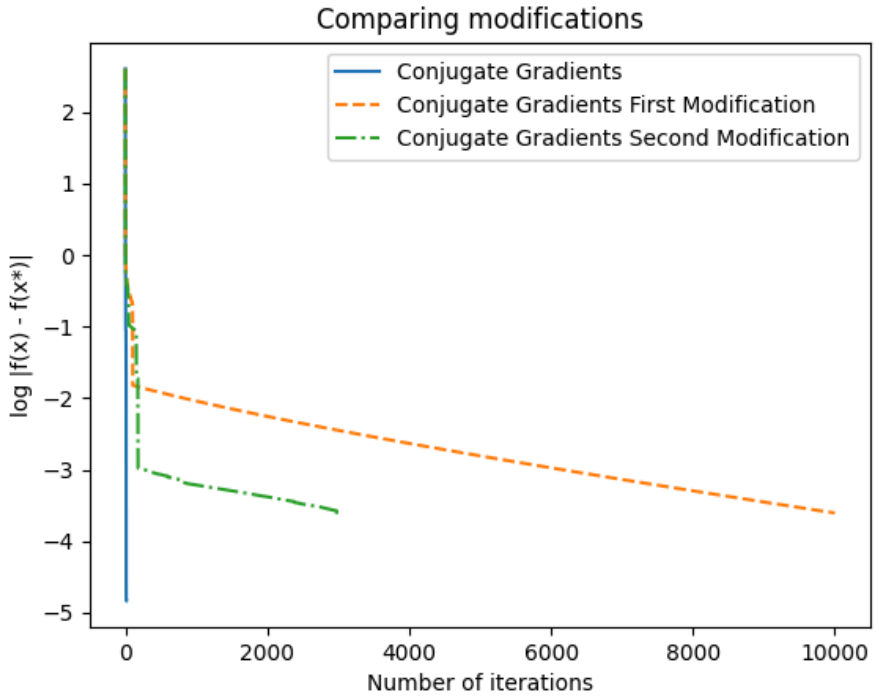

Рис. 2.2.37 ***Графік збіжності методів***

Функція №14. $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 100(x_4 - x_3^2)^2 + (1 - x_3)^2 + 100(x_6 - x_5^2)^2 + (1 - x_5)^2 +$
$+ 100(x_8 - x_7^2)^2 + (1 - x_7)^2 + 100(x_{10} - x_9^2)^2 + (1 - x_9)^2,\ n = 10,\ e = 10^{-6}$

Таблиця 2.2.38. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, с
1	(-1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1)	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	0.9997, 1.0000, 0.9994, 0.9997, 0.9994, 0.9997, 0.9994, 1.0000	25	2.18
2			0.9997, 1.0201, 0.9946, 0.9978, 0.9956, 0.9978, 0.9956, 0.9978, 0.9956, 1.0100	6496	989.70
3			0.9708, 1.8429, 0.9425, 0.9033, 0.8156, 0.9033, 0.8156, 0.9033, 0.8156, 1.3572	10001 (max)	1538.41

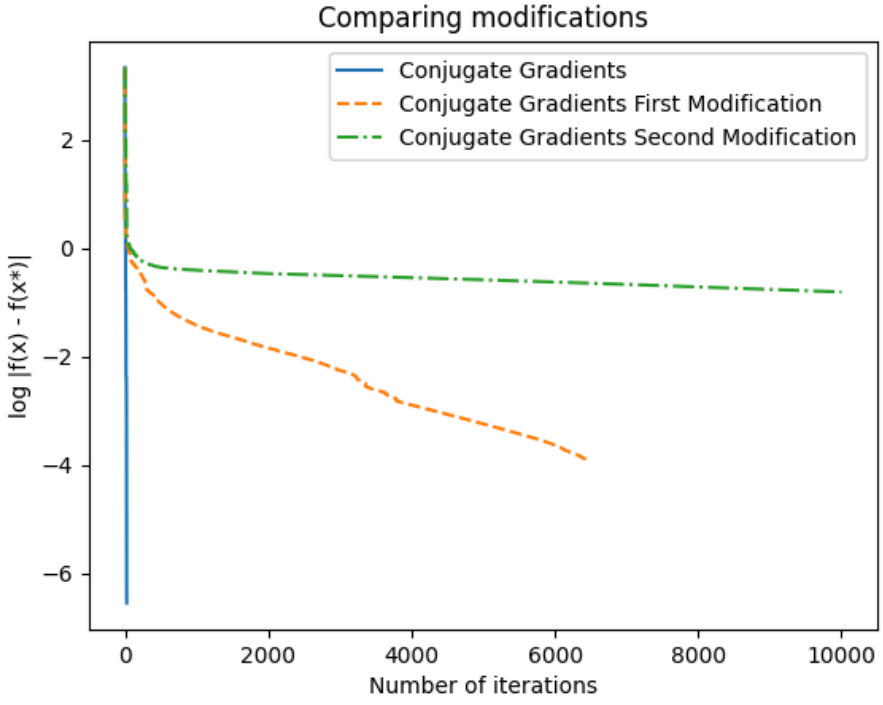

Рис. 2.2.38 ***Графік збіжності методів***

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 100(x_4 - x_3^2)^2 + (1 - x_3)^2 + 100(x_6 - x_5^2)^2 + (1 - x_5)^2 +$$

$$+ 100(x_8 - x_7^2)^2 + (1 - x_7)^2 + 100(x_{10} - x_9^2)^2 + (1 - x_9)^2,\ n = 10,\ e = 10^{-4}$$

Таблиця 2.2.39. ***Основні параметри***

№	x_0	x^*	Наближення	Ітерацій	t, c
1	(0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)	0.9941, 0.9883, 0.9883, 0.9941, 0.9883, 0.9941, 0.9883, 0.9941, 0.9883, 0.9941,	7	0.7
2			0.9775, 0.9559, 0.9559, 0.9775, 0.9559, 0.9775, 0.9559, 0.9775, 0.9559, 0.9775	326	35.62
3			0.9776, 0.9557, 0.9557, 0.9776, 0.9557, 0.9776, 0.9557, 0.9776, 0.9557, 0.9776,	1104	130.37

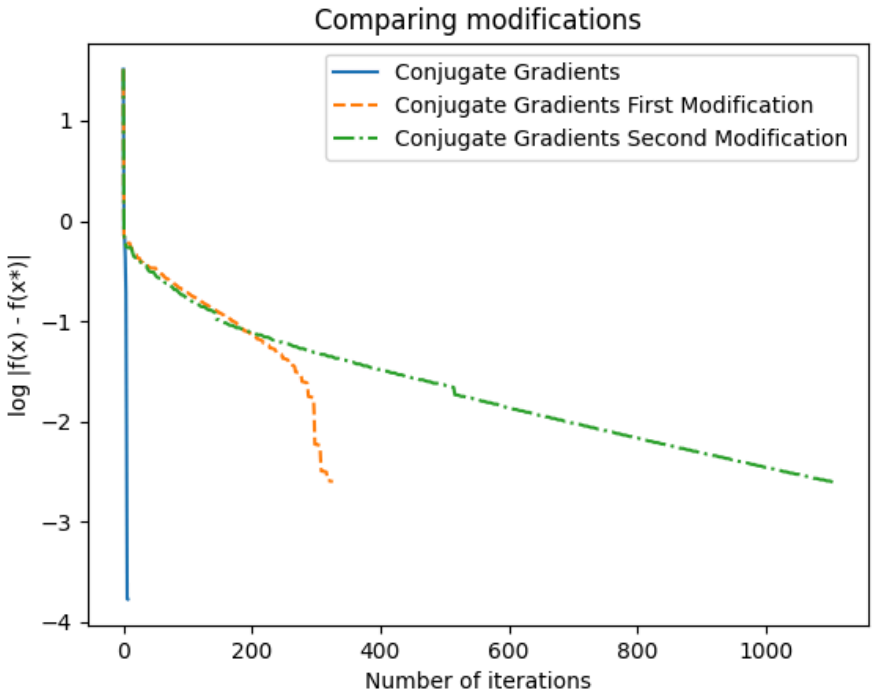

Рис. 2.2.39 ***Графік збіжності методів***

ВИСНОВКИ

На основі отриманих практичних результатів можемо порівняти методи між собою, визначити слабкі та сильні сторони кожного з них. Для простішого орієнтування серед методів введемо такі позначення: СГ - метод спряжених градієнтів(перший метод), М1 - перша модифікація(другий метод), М2 - друга модифікація(третій метод).

Отже, я перевірив роботу кожного методу на 14 тестових функціях.

Таблиця 3.1 ***Кількість тестів***

Кіл-ть вимірів	***n=2***	***n=3***	***n=4***	***n=6***	***n=8***	***n=10***	Всього
Кіл-ть функцій	4	3	3	2	1	1	14
Кіл-ть тестів	11	11	9	4	2	2	39

Для кожної функції я робив декілька досліджень:

- змінював початкову точку - віддаляв від шуканого мінімуму, або навпаки наближував. За допомогою цього досліджував методи на чутливість до місця старту роботи.
- змінював шукану точність наближення. Таким чином, досліджував методи на чутливість до заданої шуканої точності точки мінімуму.

Для кожного розрахунку отримали такі параметри, як точність знайдених точок до шуканого мінімуму (розв’язку), кількість ітерацій, витрачений час на роботу програми. Також хотів зауважити, що для різних функцій я використовував константну максимальну кількість ітерацій, тобто програма закінчувала роботу, якщо кількість ітерацій у пошуку мінімуму перевищувала задану кількість. Проаналізуємо результати відносно кожного з параметрів.

- За точністю наближень найкращим є метод СГ, на всіх тестових функціях його результати були кращими ніж у обох модифікацій. Модифікації, в свою чергу, показали приблизно однакові результати на дистанції. Але важливо зазначити, що зі збільшенням виміру, модифікації

працюють набагато гірше. Для найскладніших тестових функцій, вони навіть не досягли точки мінімуму з потрібною точністю. Тож можна зробити висновок, що починаючи з кількості вимірів більше п`яти, їх використовувати не доцільно та дуже неефективно.

- За кількістю ітерацій схожа ситуація. Метод СГ для знаходження розв`язку потребує значно менше ітерацій, ніж його модифікації. Важливо зазначити, що зі збільшенням кількості вимірів, метод СГ не потребує в десятки разів більше ітерацій, на відміну від його модифікацій. Якщо дивитися на модифікації, то можемо зробити такий висновок: навіть на легких та маловимірних функціях, модифікації інколи потребують сотні ітерацій. В цьому плані вони дуже поступаються методу СГ. Якщо ж порівнювати їх між собою, то перша модифікація потребує в більшості випадків менше ітерацій, ніж друга. Але чим більше вимір функцій, тим менше розрив, а інколи й навпаки - друга модифікація потребує менше ітерацій, ніж перша. Також важливо зазначити, що були випадки, коли програма зупиняла свою роботу через те, що модифікації перебільшували визначений максимум ітерацій, що означає, що вони не можуть ефективно знайти мінімум з потрібною точністю.
- За часом виконання метод СГ є найкращим, причому з великим відривом. На деяких функціях час його виконання був в десятки, або навіть в сотні разів меншим ніж час виконання модифікацій. Для маловимірних функцій в багатьох випадках час його виконання був меншим за секунду. Важливо зазначити, що зі збільшенням виміру функції, метод СГ потребував ненабагато більше часу. Тобто графік збільшення часу виконання приблизно дорівнює графіку збільшення кількості вимірів. Якщо порівнювати модифікації, то перевагу надам першій модифікації. В більшості випадків вона виконувалася швидше, ніж друга.
- Порівняємо методи з огляду на місце старту розрахунків. Для звичайного методу СГ місце початку роботи програми є не чутливим. Незалежно від

місця старту початку роботи програми, метод СГ потребує пропорційно однакову кількість ітерацій та часу виконання. Для модифікацій цей параметр є чутливим - чим далі від точки мінімуму програма починає роботу, тим більше ітерацій та часу потребує на виконання.

- Дивлячись на чутливість до заданої точності результатів, метод СГ знов-таки є набагато кращим та ефективнішим. Зі збільшенням точності, час роботи методу СГ не сильно збільшується, що не можна сказати про модифікації. Зі збільшенням заданої точності, методи модифікацій потребують набагато більше часу на знайдення точки мінімуму, а інколи взагалі не знаходили шукану точність, через перевищення ліміту на ітерації.

Отже, підсумуємо:

метод спряжених градієнтів (СГ) за всіма параметрами, дослідженнями та тестами є набагато кращим та ефективнішим за його модифікації. Для великовимірних функцій, модифікації взагалі не доцільно використовувати за рахунок їх неефективного та дуже повільного виконання.

СПИСОК ЛІТЕРАТУРИ

1) А.Т. Яровий Методи оптимізації та варіаційне числення: Навчально-методичний посібник для студентів спеціальності “Математика” / А.Т.Яровий, Є.М.Страхов, 2017р. - С. 84-86
2) А.Т. Яровий Методи оптимізації: Навчально-методичний посібник для студентів спеціальностей “Математика” та “Прикладна математика”/ А.Т.Яровий, Є.М.Страхов, 2020р. - С. 68-71
3) Д.М. Хіммельблау Прикладне нелінійне програмування, 1975р. - С. 98-107

ДОДАТКИ

Додаток А

Код програми на мові програмування Python, у якій реалізовані алгоритми

methods/abc_minimization_method.py

```
import matplotlib.pyplot as plt
from itertools import cycle, islice
from typing import List, Dict


def get_styles(methods_number: int) -> list:
    available_styles = ["-", "--", "-.", ":"]
    return list(islice(cycle(available_styles), 0, methods_number + 1))


def make_plot(list_of_results: List[Dict]) -> None:
    fig, ax = plt.subplots()

    list_of_styles = get_styles(len(list_of_results))

    for index, method_dict in enumerate(list_of_results):
        x, y = method_dict["x"], method_dict["y"]
        ax.plot(x, y, linestyle=list_of_styles[index], label=method_dict["name"])

    ax.set_title("Comparing modifications")
    ax.set_xlabel("Number of iterations")
    ax.set_ylabel("log |f(x) - f(x*)|")
    ax.legend()

    plt.show()
```

methods/conjugate_gradients.py

```
import numpy as np

from datetime import datetime
from typing import List, Callable, Tuple

from methods.abc_minimization_method import ABCMinimisationMethod
from mathematics.general import get_value_at_k, map_symbol_value, get_anti_dt
from mathematics.modification import get_beta_k, get_x_next
from stopping_criteria.conjugate_stopping_criteria.conjugate_stopping_criteria import check_all_criteria


class ConjugateGradients(ABCMinimisationMethod):
```

```
def __init__(self,
             function: str,
             x_0: List[int | float],
             min_point: List[int | float],
             accuracy: int,
             iteration_threshold: int,
             alpha_k_calculating_method: Callable):
    super().__init__(function, x_0, min_point, accuracy, iteration_threshold, alpha_k_calculating_method)

def _get_s_k_base_modification(self,
                               x_current: np.ndarray[float | int],
                               beta_current: int | float,
                               s_previous: np.ndarray[float | int],
                               iteration: int) -> np.ndarray[float | int]:
    symbol_value_mapping = map_symbol_value(self.free_symbols, x_current, self.dimension)

    if iteration == 0:
        s_0 = []
        for i in range(self.dimension):
            s_0.append(get_anti_dt(self.function, self.free_symbols[i]).subs(symbol_value_mapping))
        return np.array(s_0)

    else:
        x_current_anti_gradient = []
        for i in range(self.dimension):
            x_current_anti_gradient.append(get_anti_dt(self.function,
                                                       self.free_symbols[i]).subs(symbol_value_mapping))

        x_current_anti_gradient_np = np.array(x_current_anti_gradient)

        return x_current_anti_gradient_np + beta_current * s_previous

def run_method(self) -> Tuple[np.ndarray[float | int], int | float]:
    start_time = datetime.now()
    function_value_at_known_min_point = get_value_at_k(function=self.function,
                                                       free_symbols=self.free_symbols,
                                                       x_current=self.min_point,
                                                       dimension=self.dimension)

    iteration_counter = 0
    list_of_function_value_at_k_point = [get_value_at_k(function=self.function,
                                                        free_symbols=self.free_symbols,
                                                        x_current=self.x_0,
                                                        dimension=self.dimension)]

    x_current = self.x_0
    x_previous = np.array([])
    s_previous = np.array([])

    while True:
        if iteration_counter > self.iteration_threshold:
            self.print_result_info(min_point=x_current,
                                   iteration_number=iteration_counter,
                                   time=datetime.now() - start_time,
                                   reached_min_func_value=list_of_function_value_at_k_point[-1],
                                   started_func_value=list_of_function_value_at_k_point[0],
                                   known_min_function_value=function_value_at_known_min_point,
```

```
                    additional_message="!!!!!!!!! THRESHOLD !!!!!!!!!\n")
            break

        beta_current = get_beta_k(function=self.function,
                                  free_symbols=self.free_symbols,
                                  x_current=x_current,
                                  x_previous=x_previous,
                                  iteration_number=iteration_counter,
                                  dimension=self.dimension)

        s_current = self._get_s_k_base_modification(x_current=x_current,
                                                    beta_current=beta_current,
                                                    s_previous=s_previous,
                                                    iteration=iteration_counter)

        alpha_current = self.alpha_k_calculating_method(function=self.function,
                                                        free_symbols=self.free_symbols,
                                                        x_current=x_current,
                                                        s_current=s_current,
                                                        dimension=self.dimension)

        x_next = get_x_next(x_current=x_current,
                            alpha_current=alpha_current,
                            s_current=s_current)

        list_of_function_value_at_k_point.append(get_value_at_k(function=self.function,
                                                                 free_symbols=self.free_symbols,
                                                                 x_current=x_next,
                                                                 dimension=self.dimension))
        iteration_counter += 1

        if check_all_criteria(function=self.function,
                              free_symbols=self.free_symbols,
                              x_current=x_next,
                              x_previous=x_current,
                              dimension=self.dimension,
                              accuracy=self.accuracy):

            self.print_result_info(min_point=x_next,
                                   iteration_number=iteration_counter,
                                   time=datetime.now() - start_time,
                                   reached_min_func_value=list_of_function_value_at_k_point[-1],
                                   started_func_value=list_of_function_value_at_k_point[0],
                                   known_min_function_value=function_value_at_known_min_point)
            break

        else:
            x_previous = x_current
            x_current = x_next
            s_previous = s_current

    return np.array(list_of_function_value_at_k_point), function_value_at_known_min_point
```

methods/conjugate_gradients_1st_modification.py

```
import numpy as np

from datetime import datetime
from typing import List, Callable, Tuple

from methods.abc_minimization_method import ABCMinimisationMethod
from mathematics.general import get_value_at_k, map_symbol_value, get_anti_dt
from mathematics.modification import get_beta_k, get_x_next, get_h_k
from stopping_criteria.conjugate_stopping_criteria.conjugate_stopping_criteria import check_all_criteria


class ConjugateGradientsFirstModification(ABCMinimisationMethod):
    def __init__(self,
                 function: str,
                 x_0: List[int | float],
                 min_point: List[int | float],
                 accuracy: int,
                 iteration_threshold: int,
                 alpha_k_calculating_method: Callable):
        super().__init__(function, x_0, min_point, accuracy, iteration_threshold, alpha_k_calculating_method)

    def _get_s_k_first_modification(self,
                                    x_current: np.ndarray[float | int],
                                    x_previous: np.ndarray[float | int],
                                    beta_current: float | int,
                                    h_previous: np.ndarray[np.ndarray[float | int]],
                                    iteration: int) -> np.ndarray[float | int]:
        symbol_value_mapping = map_symbol_value(self.free_symbols, x_current, self.dimension)

        if iteration == 0:
            s_0 = []
            for i in range(self.dimension):
                s_0.append(get_anti_dt(self.function, self.free_symbols[i]).subs(symbol_value_mapping))
            return np.array(s_0)

        else:
            x_current_anti_gradient = []
            for i in range(self.dimension):
                x_current_anti_gradient.append(get_anti_dt(self.function,
                                                           self.free_symbols[i]).subs(symbol_value_mapping))

            x_current_anti_gradient_np = np.array(x_current_anti_gradient)
            delta_x = np.subtract(x_current, x_previous)

            return x_current_anti_gradient_np + beta_current * np.dot(h_previous, delta_x)

    def run_method(self) -> Tuple[np.ndarray[float | int], int | float]:
        start_time = datetime.now()
        function_value_at_known_min_point = get_value_at_k(function=self.function,
                                                           free_symbols=self.free_symbols,
                                                           x_current=self.min_point,
                                                           dimension=self.dimension)

        iteration_counter = 0
        list_of_function_value_at_k_point = [get_value_at_k(function=self.function,
```

```
                                    free_symbols=self.free_symbols,
                                    x_current=self.x_0,
                                    dimension=self.dimension)]

x_current = self.x_0
x_previous = np.array([])
h_previous = np.eye(self.dimension)

while True:
    if iteration_counter > self.iteration_threshold:
        self.print_result_info(min_point=x_current,
                               iteration_number=iteration_counter,
                               time=datetime.now() - start_time,
                               reached_min_func_value=list_of_function_value_at_k_point[-1],
                               started_func_value=list_of_function_value_at_k_point[0],
                               known_min_function_value=function_value_at_known_min_point,
                               additional_message="!!!!!!!!! THRESHOLD !!!!!!!!!\n")
        break

    beta_current = get_beta_k(function=self.function,
                              free_symbols=self.free_symbols,
                              x_current=x_current,
                              x_previous=x_previous,
                              iteration_number=iteration_counter,
                              dimension=self.dimension)

    s_current = self._get_s_k_first_modification(x_current=x_current,
                                                 x_previous=x_previous,
                                                 beta_current=beta_current,
                                                 h_previous=h_previous,
                                                 iteration=iteration_counter)

    alpha_current = self.alpha_k_calculating_method(function=self.function,
                                                    free_symbols=self.free_symbols,
                                                    x_current=x_current,
                                                    s_current=s_current,
                                                    dimension=self.dimension)

    x_next = get_x_next(x_current=x_current,
                        alpha_current=alpha_current,
                        s_current=s_current)

    list_of_function_value_at_k_point.append(get_value_at_k(function=self.function,
                                                            free_symbols=self.free_symbols,
                                                            x_current=x_next,
                                                            dimension=self.dimension))
    iteration_counter += 1

    try:
        if check_all_criteria(function=self.function,
                              free_symbols=self.free_symbols,
                              x_current=x_next,
                              x_previous=x_current,
                              dimension=self.dimension,
                              accuracy=self.accuracy):

            self.print_result_info(min_point=x_next,
```

```
                        iteration_number=iteration_counter,
                        time=datetime.now() - start_time,
                        reached_min_func_value=list_of_function_value_at_k_point[-1],
                        started_func_value=list_of_function_value_at_k_point[0],
                        known_min_function_value=function_value_at_known_min_point)
            break

        else:
            x_previous_previous = x_previous
            x_previous = x_current
            x_current = x_next
            if iteration_counter == 1:
                h_previous = np.eye(self.dimension)
            else:
                h_previous = get_h_k(function=self.function,
                                     free_symbols=self.free_symbols,
                                     x_next=x_previous,
                                     x_current=x_previous_previous,
                                     dimension=self.dimension,
                                     h_previous=h_previous)
    except TypeError:
        self.print_result_info(min_point=x_current,
                               iteration_number=iteration_counter,
                               time=datetime.now() - start_time,
                               reached_min_func_value=list_of_function_value_at_k_point[-1],
                               started_func_value=list_of_function_value_at_k_point[0],
                               known_min_function_value=function_value_at_known_min_point,
                               additional_message="!!!!!!!!! ENTRAPMENT !!!!!!!!!\n")
        break

    return np.array(list_of_function_value_at_k_point), function_value_at_known_min_point
```

methods/conjugate_gradients_2st_modification.py

```
import numpy as np

from datetime import datetime
from typing import List, Callable, Tuple

from methods.abc_minimization_method import ABCMinimisationMethod
from mathematics.general import get_value_at_k, map_symbol_value, get_anti_dt
from mathematics.modification import get_beta_k, get_x_next, get_h_k
from stopping_criteria.conjugate_stopping_criteria.conjugate_stopping_criteria import check_all_criteria


class ConjugateGradientsSecondModification(ABCMinimisationMethod):
    def __init__(self,
                 function: str,
                 x_0: List[int | float],
                 min_point: List[int | float],
                 accuracy: int,
                 iteration_threshold: int,
                 alpha_k_calculating_method: Callable):
        super().__init__(function, x_0, min_point, accuracy, iteration_threshold, alpha_k_calculating_method)

    def _get_s_k_second_modification(self,
```

```
                        x_current: np.ndarray[float | int],
                        x_previous: np.ndarray[float | int],
                        beta_current: float | int,
                        h_current: np.ndarray[np.ndarray[float | int]],
                        iteration: int) -> np.ndarray[float | int]:

        symbol_value_mapping = map_symbol_value(self.free_symbols, x_current, self.dimension)

        if iteration == 0:
            s_0 = []
            for i in range(self.dimension):
                s_0.append(get_anti_dt(self.function, self.free_symbols[i]).subs(symbol_value_mapping))
            return np.array(s_0)

        else:
            x_current_anti_gradient = []
            for i in range(self.dimension):
                x_current_anti_gradient.append(get_anti_dt(self.function,
                                                self.free_symbols[i]).subs(symbol_value_mapping))

            x_current_anti_gradient_np = np.array(x_current_anti_gradient)
            delta_x = np.subtract(x_current, x_previous)

            return np.dot(h_current, x_current_anti_gradient_np) + np.dot(beta_current, delta_x)

    def run_method(self) -> Tuple[np.ndarray[float | int], int | float]:
        start_time = datetime.now()
        function_value_at_known_min_point = get_value_at_k(function=self.function,
                                                free_symbols=self.free_symbols,
                                                x_current=self.min_point,
                                                dimension=self.dimension)

        iteration_counter = 0
        list_of_function_value_at_k_point = [get_value_at_k(function=self.function,
                                                free_symbols=self.free_symbols,
                                                x_current=self.x_0,
                                                dimension=self.dimension)]

        x_current = self.x_0
        x_previous = np.array([])
        s_previous = np.array([])
        h_current = np.eye(self.dimension)

        while True:
            if iteration_counter > self.iteration_threshold:
                self.print_result_info(min_point=x_current,
                                       iteration_number=iteration_counter,
                                       time=datetime.now() - start_time,
                                       reached_min_func_value=list_of_function_value_at_k_point[-1],
                                       started_func_value=list_of_function_value_at_k_point[0],
                                       known_min_function_value=function_value_at_known_min_point,
                                       additional_message="!!!!!!!!! THRESHOLD !!!!!!!!!\n")
                break

            beta_current = get_beta_k(function=self.function,
                                      free_symbols=self.free_symbols,
                                      x_current=x_current,
```

```
                    x_previous=x_previous,
                    iteration_number=iteration_counter,
                    dimension=self.dimension)

s_current = self._get_s_k_second_modification(x_current=x_current,
                                  x_previous=x_previous,
                                  beta_current=beta_current,
                                  h_current=h_current,
                                  iteration=iteration_counter)

alpha_current = self.alpha_k_calculating_method(function=self.function,
                                   free_symbols=self.free_symbols,
                                   x_current=x_current,
                                   s_current=s_current,
                                   dimension=self.dimension)

x_next = get_x_next(x_current=x_current,
              alpha_current=alpha_current,
              s_current=s_current)

list_of_function_value_at_k_point.append(get_value_at_k(function=self.function,
                                       free_symbols=self.free_symbols,
                                       x_current=x_next,
                                       dimension=self.dimension))
iteration_counter += 1

try:
   if check_all_criteria(function=self.function,
                    free_symbols=self.free_symbols,
                    x_current=x_next,
                    x_previous=x_current,
                    dimension=self.dimension,
                    accuracy=self.accuracy):

      self.print_result_info(min_point=x_next,
                        iteration_number=iteration_counter,
                        time=datetime.now() - start_time,
                        reached_min_func_value=list_of_function_value_at_k_point[-1],
                        started_func_value=list_of_function_value_at_k_point[0],
                        known_min_function_value=function_value_at_known_min_point)
      break

   else:
      x_previous = x_current
      x_current = x_next
      s_previous = s_current
      h_current = get_h_k(function=self.function,
                     free_symbols=self.free_symbols,
                     x_next=x_current,
                     x_current=x_previous,
                     dimension=self.dimension,
                     h_previous=h_current)
except TypeError:
   self.print_result_info(min_point=x_current,
                     iteration_number=iteration_counter,
                     time=datetime.now() - start_time,
                     reached_min_func_value=list_of_function_value_at_k_point[-1],
```

```
                started_func_value=list_of_function_value_at_k_point[0],
                known_min_function_value=function_value_at_known_min_point,
                additional_message="!!!!!!!!! ENTRAPMENT !!!!!!!!!\n")
            break

    return np.array(list_of_function_value_at_k_point), function_value_at_known_min_point
```

stopping_criteria/conjugate_stopping_criteria/conjugate_stopping_criteria.py

```
import numpy as np
import sympy

from typing import List

from mathematics.general import map_symbol_value, get_dt, get_vector_norm


def check_first_stopping_criteria(function: sympy.core.add.Add,
                                  free_symbols: List[sympy.Symbol],
                                  x_current: np.ndarray[float | int],
                                  x_previous: np.ndarray[float | int],
                                  dimension: int,
                                  accuracy: float) -> bool:
    previous_symbol_value_mapping = map_symbol_value(free_symbols, x_previous, dimension)
    current_symbol_value_mapping = map_symbol_value(free_symbols, x_current, dimension)

    previous_function_solution = function.subs(previous_symbol_value_mapping)
    current_function_solution = function.subs(current_symbol_value_mapping)

    return previous_function_solution - current_function_solution < accuracy * (1 +
abs(current_function_solution))


def check_second_stopping_criteria(x_current: np.ndarray[float | int],
                                   x_previous: np.ndarray[float | int],
                                   accuracy: float) -> bool:
    return get_vector_norm(x_previous - x_current) < accuracy ** 0.5 * (1 + get_vector_norm(x_current))


def check_third_stopping_criteria(function: sympy.core.add.Add,
                                  free_symbols: List[sympy.Symbol],
                                  x_current: np.ndarray[float | int],
                                  dimension: int,
                                  accuracy: float) -> bool:
    symbol_value_mapping = map_symbol_value(free_symbols, x_current, dimension)

    function_derivative_with_substitution = []
    for i in range(dimension):
        function_derivative_with_substitution.append(get_dt(function, free_symbols[i]).
                                                     subs(symbol_value_mapping))

    current_function_solution = function.subs(symbol_value_mapping)

    return (get_vector_norm(np.array(function_derivative_with_substitution))
```

```
        <= accuracy ** (1 / 3) * (1 + abs(current_function_solution)))


def check_all_criteria(function: sympy.core.add.Add,
                       free_symbols: List[sympy.Symbol],
                       x_current: np.ndarray[float | int],
                       x_previous: np.ndarray[float | int],
                       dimension: int,
                       accuracy: float) -> bool:
    return all([check_first_stopping_criteria(function=function,
                                              free_symbols=free_symbols,
                                              x_current=x_current,
                                              x_previous=x_previous,
                                              dimension=dimension,
                                              accuracy=accuracy),

                check_second_stopping_criteria(x_current=x_current,
                                               x_previous=x_previous,
                                               accuracy=accuracy),

                check_third_stopping_criteria(function=function,
                                              free_symbols=free_symbols,
                                              x_current=x_current,
                                              dimension=dimension,
                                              accuracy=accuracy)])
```

controller.py

```
from typing import Callable

from methods.conjugate_gradients import ConjugateGradients
from methods.conjugate_gradients_1st_modification import ConjugateGradientsFirstModification
from methods.conjugate_gradients_2nd_modification import ConjugateGradientsSecondModification
from mathematics.modification import get_alpha_k_single_factor_minimization,
get_alpha_k_doubling_method
from mathematics.general import convert_y_values_to_plot
from drawing.plotter import make_plot


class Controller:
    def __init__(self, settings: dict):
        self._settings = settings

    @staticmethod
    def _get_minimization_method(method_name):
        methods_dict = {
            "Conjugate Gradients": ConjugateGradients,
            "Conjugate Gradients 1st Modification": ConjugateGradientsFirstModification,
            "Conjugate Gradients 2nd Modification": ConjugateGradientsSecondModification,
        }

        try:
            return methods_dict[method_name]
        except KeyError:
            raise KeyError(f"Wrong Method Name: {method_name}")
```

```
    def _get_alpha_k_calculating_method(self) -> Callable:
        methods_dict = {
            "Single-Factor Minimization": get_alpha_k_single_factor_minimization,
            "Doubling Method": get_alpha_k_doubling_method
        }

        selected_methods = [k for k, v in self._settings["Alpha k Selection"].items() if v]
        if len(selected_methods) != 1:
            raise Exception(f"Only one alpha calculating method could be selected. It has been selected "
                            f"{len(selected_methods)} methods")
        else:
            method_name = selected_methods[0]

        try:
            return methods_dict[method_name]
        except KeyError:
            raise KeyError(f"Wrong Method Name: {method_name}")

    def run(self) -> None:
        alpha_k_method = self._get_alpha_k_calculating_method()
        selected_methods = [k for k, v in self._settings["Methods Selection"].items() if v]
        if len(selected_methods) == 0:
            raise Exception("Please, select any minimization method")

        methods_result_list = []

        for selected_method_name in selected_methods:
            # initialize calculator class
            method = self._get_minimization_method(method_name=selected_method_name)
            method_instance = method(function=self._settings["Function Settings"]["Function"],
                                     x_0=self._settings["Function Settings"]["Starting Coordinates"],
                                     min_point=self._settings["Function Settings"]["Specified Minimum Coordinates"],
                                     accuracy=self._settings["Function Settings"]["Accuracy"],
                                     iteration_threshold=self._settings["Function Settings"]["Iteration Threshold"],
                                     alpha_k_calculating_method=alpha_k_method)

            result_list, function_value_at_known_point = method_instance.run_method()

            methods_result_list.append({"name": method_instance,
                                        "x": [i for i in range(len(result_list))],
                                        "y": convert_y_values_to_plot(result_list, function_value_at_known_point)})

        if self._settings["Plotter Settings"]["Plot"]:
            make_plot(methods_result_list)
```

main.py

```
from collections import defaultdict

from controller import Controller
```

```
settings = {
    "Function Settings": {
        "Function": "(x_1**2 + x_2 - 11)**2 + (x_1 + x_2**2 - 7)**2",
        "Starting Coordinates": [1, 1],
        "Accuracy": -5,
        "Specified Minimum Coordinates": [3, 2],
        "Iteration Threshold": 1000000
    },
    "Methods Selection": {
        "Conjugate Gradients": True,
        "Conjugate Gradients 1st Modification": True,
        "Conjugate Gradients 2nd Modification": True,
    },
    "Alpha k Selection": {
        "Single-Factor Minimization": True,
        "Doubling Method": False
    },
    "Plotter Settings": {
        "Plot": True
    }
}

# input_lst = [
#     "100*(x_2-x_1**2)**2 + (1-x_1)**2 + 100*(x_4-x_3**2)**2 + (1-x_3)**2 + 100*(x_6-x_5**2)**2 +
(1-x_5)**2 + 100*(x_8-x_7**2)**2 + (1-x_7)**2", [-1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1], -7, [1,1,1,1, 1,1,1,1]
# ]
# new_settings = settings["Function Settings"]
# new_settings['Function'] = input_lst[0]
# new_settings['Starting Coordinates'] = input_lst[1]
# new_settings['Accuracy'] = input_lst[2]
# new_settings['Specified Minimum Coordinates'] = input_lst[3]
# new_settings['Iteration Threshold'] = 1000000
# print(settings)
#
# controller = Controller(settings)
# controller.run()


# 3 n=2 +++
# 2 n=3 +++
# 2 n=4 +++
# 2 n=6-, 8+
# 1 n=10-

input = [
    # 1) n = 2:

    # # 1.1.1)
    # [[-1.2, 1], '100*(x_2 - x_1**2) ** 2 + (1 - x_1)**2', -6, [1,1]],
    # # 1.1.2) change start point
    # [[0, 0], '100*(x_2 - x_1**2) ** 2 + (1 - x_1)**2', -6, [1,1]],

    # # 1.2.1)
    # [[-1.2, 1], '100*(x_2 - x_1**3)**2 + (1 - x_1)**2', -6, [1, 1]],
    #
    # # 1.2.2) change start point
```

```
# [[0, 0], '100*(x_2 - x_1**3)**2 + (1 - x_1)**2', -6, [1, 1]],

# # 1.2.3) change start point
# [[-5, 5], '100*(x_2 - x_1**3)**2 + (1 - x_1)**2', -6, [1, 1]],

# # 1.3.1)
# [[0,0], '(x_1**2 + x_2 -11)**2 + (x_1+x_2**2-7)**2', -6, [3, 2]],
#
# # 1.3.2) change start point
# [[1, 1], '(x_1**2 + x_2 -11)**2 + (x_1+x_2**2-7)**2', -6, [3, 2]],
#
# # 1.3.3) change start point
# [[15, 15], '(x_1**2 + x_2 -11)**2 + (x_1+x_2**2-7)**2', -6, [3, 2]],
#
# # 1.3.4) change accuracy
# [[15, 15], '(x_1**2 + x_2 -11)**2 + (x_1+x_2**2-7)**2', -10, [3, 2]],

# # 1.4,1
# [[2, 0.2], '(1.5-x_1*(1-x_2))**2+(2.25-x_1*(1-x_2**2))**2+(2.625-x_1*(1-x_2**3))**2', -6, [3, 0.5]],
#
# # 1.4,2 change start point
# [[0, 0], '(1.5-x_1*(1-x_2))**2+(2.25-x_1*(1-x_2**2))**2+(2.625-x_1*(1-x_2**3))**2', -6, [3, 0.5]],

# 2) n = 3:

# # 2.1.1
# [[-1,-1,-2], 'x_1**2+20*x_2**2+125*(x_3-1)**2', -6, [0,0,1]],
#
# # 2.1.2 change start point
# [[1,1,-1], 'x_1**2+20*x_2**2+125*(x_3-1)**2', -6, [0,0,1]],
#
# # 2.1.3 change accuracy
# [[-1,-1,-2], 'x_1**2+20*x_2**2+125*(x_3-1)**2', -10, [0, 0, 1]],
#
# # 2.1.4 change start point
# [[-10,-10,-10], 'x_1**2+20*x_2**2+125*(x_3-1)**2', -6, [0, 0, 1]],

# # 2.2.1
# [[-1.2,2,0], '100*(x_3 - ((x_1 + x_2) / 2)**2)**2 + (1-x_1)**2 + (1-x_2)**2', -6, [1,1,1]],
#
# # 2.2.2 change start point
# [[-10, -10, -10], '100*(x_3 - ((x_1 + x_2) / 2)**2)**2 + (1-x_1)**2 + (1-x_2)**2', -6, [1, 1, 1]],
#
# # 2.2.3 change accuracy
# [[-1.2, 2, 0], '100*(x_3 - ((x_1 + x_2) / 2)**2)**2 + (1-x_1)**2 + (1-x_2)**2', -10, [1, 1, 1]],
#
# # 2.2.4 change start point and accuracy
# [[-10, -10, -10], '100*(x_3 - ((x_1 + x_2) / 2)**2)**2 + (1-x_1)**2 + (1-x_2)**2', -10, [1, 1, 1]],

# # # 2.3.1
# [[0,0,0], '(5*x_1+100*x_2**2-x_3-9)**2 + (x_2-x_3**3+1)**2+(x_1+2*x_2-2)**2', -6, [2,0,1]],
#
# # 2.3.2 change start point
# [[-1, -1, -1], '(5*x_1+100*x_2**2-x_3-9)**2 + (x_2-x_3**3+1)**2+(x_1+2*x_2-2)**2', -6, [2, 0, 1]],
#
# # 2.3.3 change start point
# [[-5, 5, -5], '(5*x_1+100*x_2**2-x_3-9)**2 + (x_2-x_3**3+1)**2+(x_1+2*x_2-2)**2', -6, [2, 0, 1]],
```

```
    #
    # # 2.3.4 change accuracy
    # [[0, 0, 0], '(5*x_1+100*x_2**2-x_3-9)**2 + (x_2-x_3**3+1)**2+(x_1+2*x_2-2)**2', -10, [2, 0, 1]],

    # n = 4:

    # 3.1.1
    # [[1, -2, 3, 0.5], '(x_1 + 10*x_2)**2 + 5*(x_3-x_4)**2 + (x_2-3*x_3)**2+10*(x_1-x_4)**2', -6,
[0,0,0,0]],
    #
    # # 3.1.2
    # [[1, -2, 3, 0.5], '(x_1 + 10*x_2)**2 + 5*(x_3-x_4)**2 + (x_2-3*x_3)**2+10*(x_1-x_4)**2', -10, [0, 0,
0, 0]],
    #
    # # 3.1.3
    # [[1, -2, 3, 0.5], '(x_1 + 10*x_2)**2 + 5*(x_3-x_4)**2 + (x_2-3*x_3)**2+10*(x_1-x_4)**2', -15, [0, 0,
0, 0]],

    # # 3.2.1
    # [[3,-1,0,1], '(x_1+10*x_2)**2+5*(x_3-x_4)**2+(x_2-2*x_3)**4+10*(x_1-x_4)**4', -6, [0,0,0,0]],
    #
    # # 3.2.2
    # [[10,-10,10,-10], '(x_1+10*x_2)**2+5*(x_3-x_4)**2+(x_2-2*x_3)**4+10*(x_1-x_4)**4', -6, [0,0,0,0]],
    #
    # # 3.2.3
    # [[3,-1,0,1], '(x_1+10*x_2)**2+5*(x_3-x_4)**2+(x_2-2*x_3)**4+10*(x_1-x_4)**4', -10, [0,0,0,0]],

    # # 3.3.1
    # [[0,0,0,0], 'x_1**2+3*x_2**2+8*x_3**4+x_4**4-2*x_1-6*x_2-32*x_3-4*x_4', -8, [1,1,1,1]],
    # # 3.3.2
    # [[-2,-2,-2,-2], 'x_1**2+3*x_2**2+8*x_3**4+x_4**4-2*x_1-6*x_2-32*x_3-4*x_4', -8, [1,1,1,1]],
    # # 3.3.3
    # [[0,0,0,0], 'x_1**2+3*x_2**2+8*x_3**4+x_4**4-2*x_1-6*x_2-32*x_3-4*x_4', -12, [1,1,1,1]],

    # n = 6:

    # # 4.1.1
    # [[-1.2, 1, -1.2, 1, -1.2, 1], "100*(x_2-x_1**2)**2 + (1-x_1)**2 + 100*(x_4-x_3**2)**2 + (1-x_3)**2 +
100*(x_6-x_5**2)**2 + (1-x_5)**2", -6, [1,1,1,1,1,1]],

    # # 4.1.2
    # [[-2, 0, -2, 0, -2, 0], "100*(x_2-x_1**2)**2 + (1-x_1)**2 + 100*(x_4-x_3**2)**2 + (1-x_3)**2 +
100*(x_6-x_5**2)**2 + (1-x_5)**2", -7, [1,1,1,1,1,1]]

    # 4.2.1
    #[[-1.2, 1, -1.2, 1, -1.2, 1], "100*(x_2-x_1**2)**2 + (1-x_1)**2 + 100*(x_3-x_2**2)**2 + (1-x_2)**2 +
100*(x_4-x_3**2)**2 + (1-x_3)**2 + 100*(x_5-x_4**2)**2 + (1-x_4)**2+ 100*(x_6-x_5**2)**2 +
(1-x_5)**2", -6, [1,1,1,1,1,1]],

    # # 4.2.2
    # [[-1.2, 1, -1.2, 1, -1.2, 1], "100*(x_2-x_1**2)**2 + (1-x_1)**2 + 100*(x_3-x_2**2)**2 + (1-x_2)**2 +
100*(x_4-x_3**2)**2 + (1-x_3)**2 + 100*(x_5-x_4**2)**2 + (1-x_4)**2+ 100*(x_6-x_5**2)**2 +
(1-x_5)**2", -8, [1, 1, 1, 1, 1, 1]],

    # n = 8:

    # 5.1.1
```

```
    # [[-1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1], "100*(x_2 - x_1 ** 3) ** 2 + (1 - x_1) ** 2 + 100*(x_4 - x_3 ** 3) ** 2 + (1 - x_3) ** 2 + 100*(x_6 - x_5 ** 3) ** 2 + (1 - x_5) ** 2 + 100*(x_8 - x_7 ** 3) ** 2 + (1 - x_7) ** 2", -7, [1,1,1,1,1,1,1,1]] ,    # [[-1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1], "100*(x_2 - x_1 ** 3) ** 2 + (1 - x_1) ** 2 + 100*(x_4 - x_3 ** 3) ** 2 + (1 - x_3) ** 2 + 100*(x_6 - x_5 ** 3) ** 2 + (1 - x_5) ** 2 + 100*(x_8 - x_7 ** 3) ** 2 + (1 - x_7) ** 2", -7, [1,1,1,1,1,1,1,1]] ,

    # 5.1.2 change start point
    # [[1, 0, 1, 0, 1, 0, 1, 0,], "100*(x_2 - x_1 ** 3) ** 2 + (1 - x_1) ** 2 + 100*(x_4 - x_3 ** 3) ** 2 + (1 - x_3) ** 2 + 100*(x_6 - x_5 ** 3) ** 2 + (1 - x_5) ** 2 + 100*(x_8 - x_7 ** 3) ** 2 + (1 - x_7) ** 2", -6, [1,1,1,1,1,1,1,1]] ,    # [[-1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1], "100*(x_2 - x_1 ** 3) ** 2 + (1 - x_1) ** 2 + 100*(x_4 - x_3 ** 3) ** 2 + (1 - x_3) ** 2 + 100*(x_6 - x_5 ** 3) ** 2 + (1 - x_5) ** 2 + 100*(x_8 - x_7 ** 3) ** 2 + (1 - x_7) ** 2", -7, [1,1,1,1,1,1,1,1]] ,

    # n = 10:

    # 6.1.1
    # [[-1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1],
    #  "100*(x_2-x_1**2)**2 + (1-x_1)**2 + 100*(x_4-x_3**2)**2 + (1-x_3)**2 + 100*(x_6-x_5**2)**2 + (1-x_5)**2 + 100*(x_8-x_7**2)**2 + (1-x_7)**2 + 100*(x_10-x_9**2)**2 + (1-x_9)**2",
    #  -6, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]],

    # 6.1.2
    # [[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
    #  "100*(x_2-x_1**2)**2 + (1-x_1)**2 + 100*(x_4-x_3**2)**2 + (1-x_3)**2 + 100*(x_6-x_5**2)**2 + (1-x_5)**2 + 100*(x_8-x_7**2)**2 + (1-x_7)**2 + 100*(x_10-x_9**2)**2 + (1-x_9)**2",
    #  -4, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]],
]


for input_lst in input:
    new_settings = settings["Function Settings"]
    new_settings['Function'] = input_lst[1]
    new_settings['Starting Coordinates'] = input_lst[0]
    new_settings['Accuracy'] = input_lst[2]
    new_settings['Specified Minimum Coordinates'] = input_lst[3]
    new_settings['Iteration Threshold'] = 10000

    controller = Controller(settings)
    controller.run()
```

mathematics/general.py

```
import sympy
import numpy as np

from typing import List, Dict


def get_dt(function, symbol):
    return sympy.diff(function, symbol)


def get_anti_dt(function, symbol):
```

```
    return -sympy.diff(function, symbol)


def get_vector_norm(vector):
    return sum([abs(element) ** 2 for element in vector]) ** 0.5


def map_symbol_value(symbols_array, values_array, dimension):
    symbol_value_mapping = {}
    for i in range(dimension):
        symbol_value_mapping[symbols_array[i]] = values_array[i]

    return symbol_value_mapping


def get_value_at_k(function, free_symbols, x_current, dimension):
    symbol_value_mapping = map_symbol_value(free_symbols, x_current, dimension)
    return function.subs(symbol_value_mapping)


def convert_y_values_to_plot(function_values_at_all_points,
                             function_value_at_known_min_point):
    y = []
    for i in function_values_at_all_points.astype(np.float64):
        y.append(np.log10(abs(i - np.array(function_value_at_known_min_point).astype(np.float64))))

    return np.array(y)
```

mathematics/modification.py

```
import numpy as np
import sympy
from sympy import symbols, simplify, lambdify
from scipy.optimize import minimize_scalar

from typing import List

from mathematics.general import (map_symbol_value, get_dt, get_vector_norm,
                                 get_value_at_k)


def get_alpha_k_single_factor_minimization(function: sympy.core.add.Add,
                                           free_symbols: List[sympy.Symbol],
                                           x_current: np.ndarray[float | int],
                                           s_current: np.ndarray[float | int],
                                           dimension: int) -> float | int:
    alpha = symbols("alpha")

    alpha_vector = {}
    for i in range(dimension):
        alpha_vector[free_symbols[i]] = x_current[i] + alpha * s_current[i]

    equation_with_alpa_substitution = simplify(function.subs(alpha_vector))
```

```
    lam_f = lambdify(alpha, equation_with_alpa_substitution)
    alpha_k = minimize_scalar(lam_f)
    return alpha_k.x


def get_alpha_k_doubling_method(function: sympy.core.add.Add,
                                free_symbols: List[sympy.Symbol],
                                x_current: np.ndarray[float | int],
                                s_current: np.ndarray[float | int],
                                dimension: int) -> float | int:
    alpha = 1
    x_next = get_x_next(x_current, alpha, s_current)

    while (get_value_at_k(function, free_symbols, x_next, dimension) >
           get_value_at_k(function, free_symbols, x_current, dimension)):
        alpha /= 2
        x_next = get_x_next(x_current, alpha, s_current)

    return alpha


def get_x_next(x_current: np.ndarray[float | int],
               alpha_current: float | int,
               s_current: np.ndarray[float | int]) -> np.ndarray[float | int]:
    return x_current + s_current * alpha_current


def get_beta_k(function: sympy.core.add.Add,
               free_symbols: List[sympy.Symbol],
               x_current: np.ndarray[float | int],
               x_previous: np.ndarray[float | int],
               iteration_number: int,
               dimension: int) -> int | float:
    if iteration_number % dimension == 0:
        return 0

    else:
        x_current_symbol_value_mapping = map_symbol_value(free_symbols, x_current, dimension)
        x_previous_symbol_value_mapping = map_symbol_value(free_symbols, x_previous, dimension)

        current_function_derivative_with_substitution = []
        previous_function_derivative_with_substitution = []
        for i in range(dimension):
            current_function_derivative_with_substitution.append(get_dt(function, free_symbols[i]).
                                                 subs(x_current_symbol_value_mapping))

            previous_function_derivative_with_substitution.append(get_dt(function, free_symbols[i])
                                                  .subs(x_previous_symbol_value_mapping))

        return (get_vector_norm(np.array(current_function_derivative_with_substitution)) ** 2
                / get_vector_norm(np.array(previous_function_derivative_with_substitution)) ** 2)


def get_delta_x(x_next: np.ndarray[float | int],
                x_current: np.ndarray[float | int]) -> np.ndarray[float | int]:
    return np.subtract(x_next, x_current)
```

```
def get_delta_y(function: sympy.core.add.Add,
                free_symbols: List[sympy.Symbol],
                x_next: np.ndarray[float | int],
                x_current: np.ndarray[float | int],
                dimension: int) -> np.ndarray[float | int]:
    next_symbol_value_mapping = map_symbol_value(free_symbols, x_next, dimension)
    current_symbol_value_mapping = map_symbol_value(free_symbols, x_current, dimension)

    derivative_list = []
    for i in range(dimension):
        derivative_list.append(get_dt(function, free_symbols[i]))

    next_derivative_w_substitution = []
    current_derivative_w_substitution = []
    for derivative in derivative_list:
        next_derivative_w_substitution.append(derivative.subs(next_symbol_value_mapping))
        current_derivative_w_substitution.append(derivative.subs(current_symbol_value_mapping))

    return (np.array(next_derivative_w_substitution).astype(np.float64)
            - np.array(current_derivative_w_substitution).astype(np.float64))


def get_h_k(function: sympy.core.add.Add,
            free_symbols: List[sympy.Symbol],
            x_next: np.ndarray[float | int],
            x_current: np.ndarray[float | int],
            dimension: int,
            h_previous: np.ndarray[np.ndarray[float | int]]): #-> np.ndarray[np.ndarray[float | int]]:

    delta_x = get_delta_x(x_next=x_next,
                          x_current=x_current)

    delta_y = get_delta_y(function=function,
                          free_symbols=free_symbols,
                          x_next=x_next,
                          x_current=x_current,
                          dimension=dimension)

    bracketed_expression = np.subtract(delta_x, np.dot(h_previous, delta_y))
    denominator = np.dot(bracketed_expression, delta_y)
    if denominator == 0:
        return np.eye(dimension)
    else:
        expression_right_side = np.divide(np.dot(bracketed_expression.T, bracketed_expression), denominator)

        return np.add(h_previous, expression_right_side)
```

drawing/plotter.py

```
import matplotlib.pyplot as plt
from itertools import cycle, islice
from typing import List, Dict
```

```
def get_styles(methods_number: int) -> list:
    available_styles = ["-", "--", "-.", ":"]
    return list(islice(cycle(available_styles), 0, methods_number + 1))


def make_plot(list_of_results: List[Dict]) -> None:
    fig, ax = plt.subplots()

    list_of_styles = get_styles(len(list_of_results))

    for index, method_dict in enumerate(list_of_results):
        x, y = method_dict["x"], method_dict["y"]
        ax.plot(x, y, linestyle=list_of_styles[index], label=method_dict["name"])

    ax.set_title("Comparing modifications")
    ax.set_xlabel("Number of iterations")
    ax.set_ylabel("log |f(x) - f(x*)|")
    ax.legend()

    plt.show()
```