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 «бакалавр»

«Дослідження збіжності методу еліпсоїдів»

«Research on the Convergence of the Ellipsoid Method»

Виконала: здобувачка денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»
Завальнюк Яна Павлівна

Керівник: канд. фіз.-мат. наук, доц. Яровий А. Т. _____
Рецензент: канд. фіз.-мат. наук, доц. Страхов Є. М.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _____ 2024 р.

Завідувач кафедри

_____ _____

Захищено на засіданні ЕК № _____

Протокол № ___ від _____ 2024 р.

Оцінка _____ / _____ / _____

Голова ЕК

_____ _____

Одеса — 2024 р.

ЗМІСТ

ВСТУП

Метод еліпсоїдів має велике теоретичне і практичне значення.
Його розглядають як особливий випадок методу УГСРП у напрямку субградієнта з незмінним коефіцієнтом розтягу простору і множником кроку, що змінюється формулою геометричної прогресії; або як метод послідовних відтинань, де на кожному кроці оптимізації за допомогою еліпсоїда апроксимується область локалізації розв'язку.

Цей метод оптимізації використовується для задач опуклого програмування, задач пошуку сідлових точок опукло-увігнутих функцій, особливі випадки задач варіаційних нерівностей, спеціальні класи задач лінійної і нелінійної додатковості, тощо. Але здебільшого і з моменту свого створення він використовується для мінімізації опуклих функцій.

Ще один відомий клас таких методів - методи змінної метрики(квазіньютонівські), які є ефективним засобом розв'язання задач безумової оптимізації.
У сучасній науковій літературі існує значна кількість досліджень, присвячених порівнянню методу еліпсоїдів і квазіньютонівських методів. Ці дослідження включають аналітичні та експериментальні підходи для оцінки ефективності цих методів на різних наборах функцій та у різних умовах.

Коло питань:

- **Швидкість збіжності:** Порівняння швидкості збіжності методу еліпсоїдів і квазіньютонівських методів для різних класів функцій.
- **Вплив розмірності:** Вивчення впливу розмірності простору на ефективність обох методів і визначення області їхньої застосовності в залежності від розмірності проблеми.
- **Порівняння складності:** Оцінка обчислювальної складності, включаючи аналіз кількості обчислень та вимог до пам'яті.

Для методу еліпсоїдів розглянемо ітеративний алгоритм і алгоритм побудови еліпсоїдів. Доведемо теореми збіжності методу, як субградієнтного і як градієнтного.

Згадаємо постановку задачі та алгоритм методу змінної метрики (квазіньютонівські).

Реалізуємо мінімізацію функції обома методами і порівняємо їх.

Зробимо відповідні висновки на основі досліджень.

РОЗДІЛ 1

ТЕОРЕТИЧНА ЧАСТИНА

1.1 Метод еліпсоїдів

1.1.1 Постановка задачі

Нехай на E^n задано векторне поле $g(x)$. Необов'язково, щоб воно було неперервним. [5]

$$g(x) \in E^n, (x \in E^n) \tag{1.1.1}$$

Розглянемо задачу: необхідно знайти таку точку x^*, що $(g(x), x - x^*) \geq 0$ для усіх $x \in E^n$. Припустимо, що ця задача має рішення. До того ж, нам відомо, що $x \in S(x_0, R)$, де $S(x_0, R)$ - замкнутий шар із центром у точці x_0 і радіусом R. Точка $x*$ належить також і півкулі

$$\bar{S}_0(x_0, R) = S_0(x_0, R) \cap \{x : (g(x_0), x - x_0) \leq 0\}. \tag{1.1.2}$$

Побудуємо еліпсоїд з мінімальним об'ємом, який містить $\bar{S}_0(x_0, R)$, та за допомогою лінійного перетворення будемо розтягувати простір таким чином, щоб заданий еліпсоїд перетворився на кулю. Радіус цієї нової кулі буде помітно зменшено, відносно $R = R_0$

$$R_1 = R_0 \sqrt[3]{1 - \frac{n}{2}}. \tag{1.1.3}$$

Тому, при повторені даного процесу, буде локалізовуватися $x*$ у кулях з послідовно зменшуваним радіусом.

Варто відзначити, що об'єми тих еліпсоїдів, що утворюються під час процедури, наближуються до нуля зі швидкістю геометричної прогресії, знаменник якою розраховується за формулою

$$q = \sqrt{\frac{n-1}{n+1}} \left(\frac{n}{\sqrt{n^2 - 1}} \right)^n, \tag{1.1.4}$$

що, як можна легко побачити, залежить виключно від розміру простору(кількості змінних). Детальний опис виразу цього знаменника у формулах(1.1.9) - (1.1.11).

Приклад двох послідовних кроків у початковому просторі показанний на рисунку 1.1.[8]

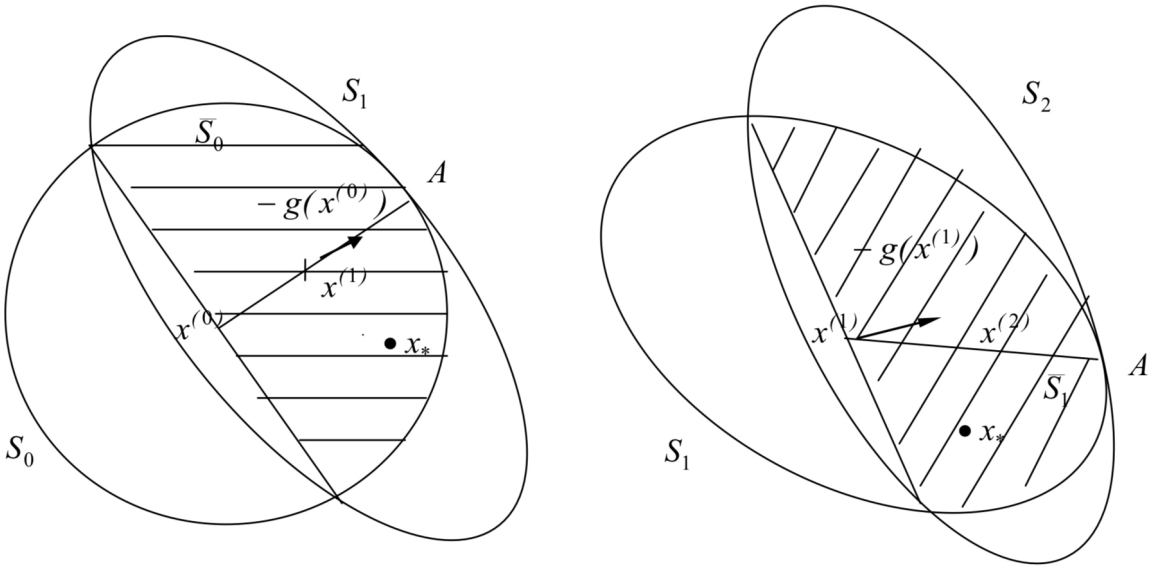


Рис. 1.1

На даному етапі можна припустити, що на швидкість збіжності методу еліпсоїдів впливає відносно невелика кількість даних. Далі ми спробуємо це довести чи спростити під час практичних досліджень.

Також важливим є зазначити, що мінімізація методом еліпсоїдів не повинна відображатися на значеннях функції, як прийнято вважати. Зменшення(тобто мінімізації) відбувається у відстані h від точки x_k до x_{k+1}.

1.1.2 Алгоритм методу

Розглянемо ітеративний алгоритм розв'язку заданої задачі за $n > 1$.[8] Припустимо, що $g(x) \neq 0$ при $x \neq x^*$.

Перед початком розрахунків маємо:

- $x_0 \in E^n$
- $B_0 = I_n$ - одинична матриця
- $h_0 = \frac{R}{n+1}$

I на k-му кроку отримали:

- $x_k \in E^n$
- B_k - матриця $n \times n$, B_k - спряжена матриця.
- $h_k > 0$.

На кроці $(k+1)$ розраховуємо:

1) значення вектору $g(x_k)$; якщо $g(x_k) = 0$, то x_k - шукана точка;
2) вектор, що задає напрям перетворення простору

$$\xi_k = \frac{B_k^* g(x_k)}{\|B_k^* g(x_k)\|}; \tag{1.1.5}$$

3) значення нової точки

$$x_{k+1} = x_k - h_k B_k \xi_k; \tag{1.1.6}$$

4) матрицю перетворення простору

$$B_{k+1} = B_k R_\beta(\xi_k), \beta = \sqrt{(n-1)/(n+1)}; \tag{1.1.7}$$

де $R_\beta(\xi_k)$ - оператор розтягування простору у напрямку ξ_k з коефіцієнтом β;

5) значення крокового множника

$$h_{k+1} = h_k r, r = n/\sqrt{n^2-1}. \tag{1.1.8}$$

Теорема 1.1: Послідовність $\{x\}_{k=0}^{\infty}$, що генерується алгоритмом, зазначеним у формулах (1.5) - (1.8), задовольняє наступну нерівність

$$\|A_k(x_k - x^*)\| \geq h_k(n+1), k = 0, 1, 2, \ldots \tag{1.1.9}$$

де $A_k = B_k^{-1}$. Множина точок x, що задовольняють нерівність

$$\|A_k(x_k - x)\| \geq (n+1)h_k = R(n/\sqrt{n^2-1})^k, \tag{1.1.10}$$

являє собою еліпсоїд Φ_k, об'єм якого

$$\nu(\Phi_k) = \frac{\nu_0 R^n (n/\sqrt{n^2-1})^{nk}}{detA_k}$$

де ν_0 - об'єм одиничного n-мірного шару. Отримаємо оцінку

$$\frac{\nu(\Phi_{k+1})}{\nu(\Phi_k)} = \frac{(n/\sqrt{n^2-1})^n detA_k}{detA_{k+1}} = \frac{(n/\sqrt{n^2-1})^n detA_k}{detR_\alpha(\xi_k)detA_k} =$$

$$= \frac{1}{\alpha}\left(\frac{n}{\sqrt{n^2-1}}\right)^n = \sqrt{\frac{n-1}{n+1}}\left(\frac{n}{\sqrt{n^2-1}}\right)^n. \tag{1.1.11}$$

1.1.3 Алгоритм побудови еліпсоїдів

Метод еліпсоїдів заснований на використанні в R^n еліпсоїду мінімального об'єму, який містить півкулю, отриманий в результаті перетину n-мірної кулі і півпростору, що проходить через його центр. Цей еліпсоїд має сплюснуту форму у напрямку нормалі до гіперплощини, що проходить через центр кулі з радіусом r. На рисунку 1.2 показані параметри еліпсоїду мінімального об'єму, що містить півкулю в R^n.[6]

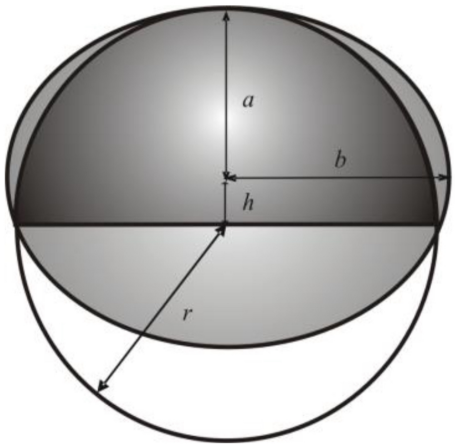


Рис. 1.2

Параметри цього еліпсоїду наступні:

1) a - довжина меншої півосі у напрямку нормалі, що визначає півкулю;
2) b - довжина більшої півосі(кількість таких напівосей дорівнює $n-1$);
3) h - відстань від центру кулі до центру еліпсоїду у напрямку меншшої з його півосей.

$$a = r\frac{n}{n+1}, \tag{1.1.12}$$

$$b = r\frac{n}{\sqrt{n^2-1}}, \tag{1.1.13}$$

$$h = r\frac{n}{n+1}, \tag{1.1.14}$$

Ітерація методу еліпсоїдів уявляє собою перехід від наяного еліпсоїду до

наступного із постійним коефіцієнтом зменшення їх об'ємів. Цей коефіцієнт визначається відношенням об'єму еліпсоїду з довжинами напівосей a і b до об'єму кулі радіусу r в R^n та може бути записан у вигляді

$$q_n = \left(\frac{a}{r}\right)\left(\frac{b}{r}\right)^{n-1} = \frac{n}{n+1}\left(\frac{n}{\sqrt{n^2-1}}\right)^{n-1} < 1. \qquad (1.1.15)$$

1.1.4 Теорема збіжності

Розглянемо теорему збіжності методу еліпсоїдів, як субградієнтного методу.

Умови:

1) Функція f опукла і обмежена знизу.
2) Послідовність кроків $\{\alpha_k\}$ задовольняє умовам:
   - $\sum_{k=0}^{\infty} \alpha_k = \infty$
   - $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$

Теорема: Якщо f опукла і обмежена знизу, і послідовність кроків $\{\alpha_k\}$ задовольняє заданим умовам, то субградієнтний метод, що генерує послідовність $\{x_k\}$ за правилом

$$x_{k+1} = x_k - \alpha_k g_k, \tag{1.1.16}$$

де $g_k \in \partial f(x_k)$, збігається до оптимального значення функції f

$$\lim_{k\to\infty} f(x_k) \longrightarrow f(x^*). \tag{1.1.17}$$

Але для доведення абсолютної збіжності методу еліпсоїдів необхідно, щоб збіжність досягалася як при субградієнті, так і при градієнті.
Саме тому ми розглянемо теорему про збіжність, але замінивши субградієнт на градієнт.

Умови:

1) Функція f опукла, диференційована і обмежена знизу.
2) Послідовність кроків $\{\alpha_k\}$ задовольняє умовам:
   - $\sum_{k=0}^{\infty} \alpha_k = \infty$
   - $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$

Теорема: Якщо f опукла і обмежена знизу, і послідовність кроків $\{\alpha_k\}$ задовольняє заданим умовам, то субградієнтний метод, що генерує

послідовність $\{x_k\}$ за правилом

$$x_{k+1} = x_k - \alpha_k \bigtriangledown f(x_k), \tag{1.1.18}$$

збігається до оптимального значення функції f

$$\lim_{k\to\infty} f(x_k) \longrightarrow f(x^*). \tag{1.1.17}$$

Доведемо це, аналогічно доведенню збіжності субградієнтного методу, використовуючи той факт,що дл дифеференційованих функцій градієнт грає роль субградієнту. Позначимо x^* точку мінімума функції f. Тоді

$$\|x_{k+1} - x^*\|^2 = \|x_k - \alpha_k \bigtriangledown f(x_k) - x^*\|^2 \tag{1.1.19}$$

Використаємо $\|a+b\|^2 = \|a\|^2 + 2a^Tb + \|b\|^2$ щоб разгорнути квадрат

$$\|x_{k+1} - x^*\|^2 = \|x_k - x^*\|^2 - 2\alpha_k \bigtriangledown f(x_k)^T(x_k - x^*)+$$

$$+\alpha_k^2\| \bigtriangledown f(x_k)\|^2 \tag{1.1.20}$$

Для опуклої фунції f та її градієнту виконується

$$f(x_k) - f(x^*) \leq \bigtriangledown f(x_k)^T(x_k - x^*)$$

Підставляємо це в рівняння

$$\|x_{k+1} - x^*\|^2 \leq \|x_k - x^*\|^2 - 2\alpha_k(f(x_k) - f(x^*))+$$

$$+\alpha_k^2\| \bigtriangledown f(x_k)\|^2 \tag{1.1.21}$$

Складаємо цю нерівність по k от 0 до $N-1$

$$\|x_N - x^*\|^2 + 2\sum_{k=0}^{N-1} \alpha_k(f(x_k) - f(x^*)) \leq \|x_0 - x^*\|^2+$$

$$+\sum_{k=0}^{N-1} \alpha_k^2\| \bigtriangledown f(x_k)\|^2 \tag{1.1.22}$$

Оскільки $\|x_N - x^*\|^2 \geq 0$, отримаємо

$$2\sum_{k=0}^{N-1} \alpha_k (f(x_k) - f(x^*)) \leq \|x_0 - x^*\|^2 + \sum_{k=0}^{N-1} \alpha_k^2 \| \bigtriangledown f(x_k)\|^2 \qquad (1.1.23)$$

Враховуючи те, що $\sum_{k=0}^{\infty} \alpha_k = \infty$ і $\sum_{k=0}^{\infty} \alpha_k < \infty$, права частина залишається обмеженною, а ліва частина зростає нескінчено.
Це означає, що $f(x_k) \longrightarrow f(x^*)$ при $k \longrightarrow \infty$.

Теорему доведено.

1.1.5 Загальні висновки на основі теоретичних даних

Методом еліпсоїдів можна мінімізувати опуклу функцію; для цього необхідно знайти точку x^* , яка задовольняє умові $(g(x), x - x^*) \geq 0$, належить замкнутій кулі $S(x_0, R)$ і знаходиться в півкулі $\bar{S}_0(x_0, R)$.

Алгоритм включає ітеративні кроки з визначенням нової точки x_{k+1}, напрямку перетворення простору ξ_k, матриці перетворення простору B_{k+1}, і значення крокового множника h_{k+1}.

Була доведена теорема про збіжність методу еліпсоїдів.

Зазначено, що на швидкість збіжності методу еліпсоїдів впливає тільки розмірність опуклої функції, що буде доведено чи спростовано у практичній частині досліджень.

1.2 Метод змінної метрики(квазіньютонівський)

1.2.1 Постановка задачі

Нехай функція $f(x)$ є двічі неперервно-диференційованою.[5] Розглянемо метод

$$x_{k+1} = x_k + \alpha_k p_k, \tag{1.2.1}$$

де $p_k = -H_k f'(x_k)$.
Квазіньютонівська умова вимагає, щоб матриця H_{k+1}, яка наближує $[f''(x_k)]^{-1}$, виконувала умову

$$H_{k+1}\Delta y_k = \Delta x_k$$

де

$$\Delta y_k = f'(x_{k+1}) - f'(x_k); \Delta x_k = x_{k+1} - x_k.$$

У методі Бройдена-Флетчера-Гольдфарба-Шанно(БФГШ), що ми використовуватимемо для порівняння, для знаходження матриці H_{k+1} використовується формула

$$H_{k+1} = H_k + \left(1 - \frac{[\Delta y_k]^T H_k \Delta y_k}{[\Delta x_k]^T \Delta y_k}\right) \frac{\Delta x_k [\Delta x_k]^T}{[\Delta x_k]\Delta y_k} -$$

$$- \frac{\Delta x_k [\Delta y_k]^T H_k + H_k \Delta y_k [\Delta x_k]^T}{[\Delta x_k]^T \Delta y_k} \tag{1.2.2}$$

Квазіньютонівські методи також відомі як методи змінної метрики. Ця назва пояснюється тим, що будь-яка симетричка, додатно визначена матриця H_k визначає скалярний добуток $(u, v)_k = (H_k, u, v)$ і пов'язану з ним метрику.

1.2.2 Алгоритм методу

Крок 1

Взяти

$$p_j = -H_j f'(y_j)$$

і за α_j обрати оптимальний розв'язок задачі мінімізації $f(y_j + \alpha p_j)$ при $\alpha_j \geq 0$. Покласти

$$y_{j+1} = y_j + \alpha_j p_j.$$

Якщо $j > n$, то перейти до кроку 2.
Якщо $j = n$, то покласти $y_1 = x_{k+1} = y_{n+1}$, замінити k на $k + 1$, покласти $j = 1$ та перейти до кроку 1.

Якщо $\|f'(y_j)\| < \epsilon$, то зупинитися. Де $\epsilon > 0$ - обране число для зупинки алгоритму.

Крок 2

Побудувати матрицю H_{j+1} за формулою, зазначеною в (1.2.2). Замінити j на $j + 1$ та перейти до кроку 1.

РОЗДІЛ 2

ПРАКТИЧНА ЧАСТИНА

2.1 Реалізація методу еліпсоїдів

2.1.1 Мінімізація функції

Приклад мінімізації функції методом еліпсоїдів був реалізований створенням відповідної програми на мові Python(ДОДАТОК А).

Головна задача - порівняння з методом змінної метрики - буде виконана програмним методом; але для більш повного розуміння методу еліпсоїдів, його алгоритму і принципів побудови необхідних еліпсоїдів розглянемо наступну задачу.

Нехай f опукла і обмежена знизу. Необхідно знайти таку точку x^*, що $(g(x), x - x^*) \geq 0$ для усіх $x \in E^n$. Припустимо, що ця задача має рішення. До того ж, нам відомо, що $x \in S(x_0, R)$, де $S(x_0, R)$ - замкнутий шар із центром у точці x_0 і радіусом R.

Точка $x*$ належить також і півкулі $\bar{S}_0(x_0, R)$. Розмірність мінімізуємої опуклої функції $n = 2$.

На початковому етапі візьмемо коло з $R = 1$ і центром $x_0 = (1; 2)$. Також маємо $B_0 = I_n$ - одинична матриця.

$$\varphi(x) = (x_2 - x_1^2)^2 + (1 - x_1)^2$$

$$\xi_k = [1; 1]$$

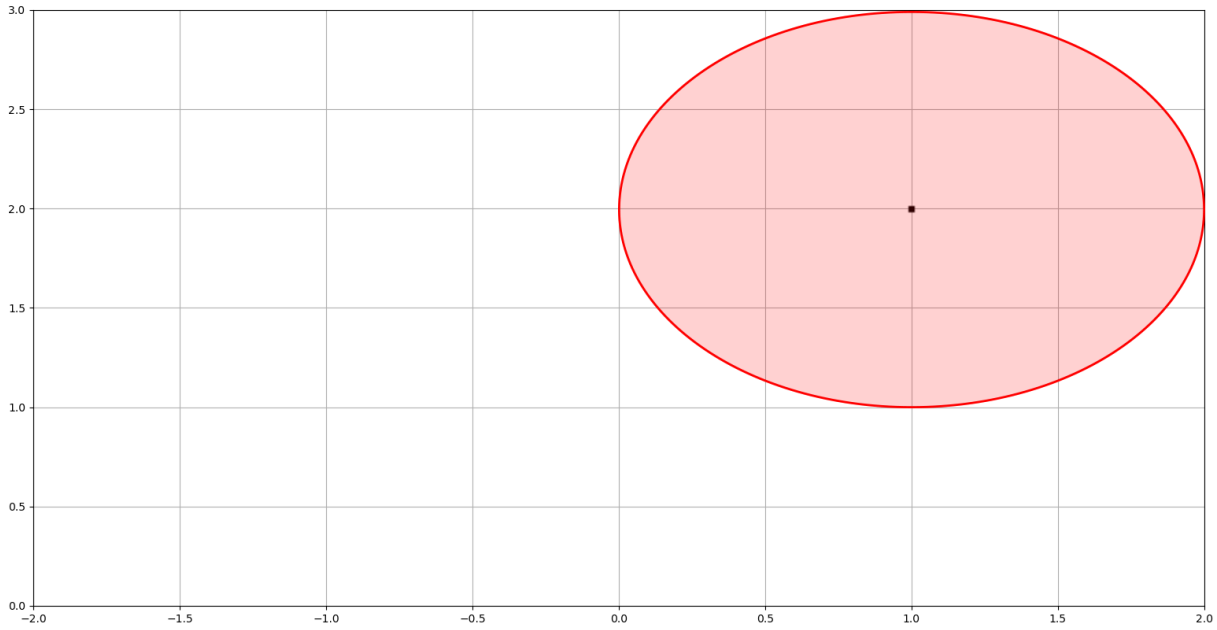


Рис. 2.1

У теоретичній частині були наведені формули, необхідні для побудови еліпсоїду для кожного кроку ітерації. Ми розглянемо перехід від початкового етапу до першої завершенної ітерації.

Маючи радіус початкового кола, розмірність простору мінімізуємої функції і користуючись формулами (1.1.12)-(1.1.14), отримаємо

$$a = 1 \times \frac{2}{2+1} = 0{,}66667$$

$$b = 1 \times \frac{2}{\sqrt{2^2+1}} = 0{,}8944$$

$$h = 1 \times \frac{2}{2+1} = 0{,}66667$$

Розрахуємо значення наступної точки x_1 за формулою

Знайдемо вираз $B_0\xi_k$:

$$B_0\xi_k = I \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Далі обчислимо вираз $h_0 B_0 \xi_k$:

$$h_0 B_0 \xi_k = h_0 \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0{,}66667 \\ 0{,}66667 \end{pmatrix}.$$

Тепер обчислимо $x_0 - h_0 B_0 \xi_k$:

$$x_1 = x_0 - h_0 B_0 \xi_k = \begin{pmatrix} 1 \\ 2 \end{pmatrix} - \begin{pmatrix} 0{,}66667 \\ 0{,}66667 \end{pmatrix} = \begin{pmatrix} 0{,}33333 \\ 1{,}33333 \end{pmatrix}.$$

Що дозволяє побудувати наступний еліпсоїд.

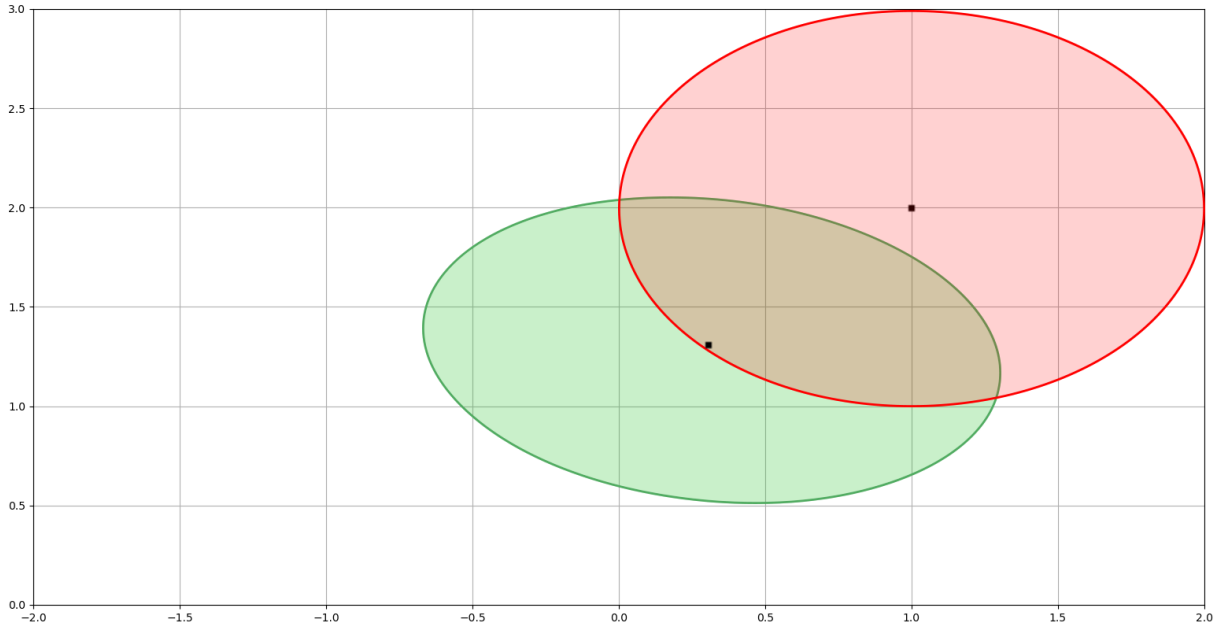


Рис. 2.2

Для наступних ітерацій необхідно роозрахувати значення матриці перетворення B_1 за формулою

$$B_{k+1} = B_k R_\beta(\xi_k), \beta = \sqrt{(n-1)/(n+1)}$$

Розрахуємо значення

$$\beta = \sqrt{(2-1)/(2+1)} = \frac{1}{\sqrt{3}}$$

Оператор розтягування простору $R_\beta(\xi_k)$ визначається через проекцію на напрямок ξ_k і ортогональне доповнення до нього. Для $\xi_k = (1,1)$ маємо:

$$R_\beta(\xi_k) = I + (\beta - 1)\frac{\xi_k \xi_k^T}{\|\xi_k\|^2}.$$

Обчислимо норму ξ_k:

$$\|\xi_k\|^2 = 1^2 + 1^2 = 2.$$

Обчислимо $\xi_k \xi_k^T$:

$$\xi_k \xi_k^T = \begin{pmatrix} 1 \\ \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \\ 1 & 1 \end{pmatrix}.$$

Тепер обчислимо $(\beta - 1)\frac{\xi_k \xi_k^T}{\|\xi_k\|^2}$:

$$(\beta - 1)\frac{\xi_k \xi_k^T}{\|\xi_k\|^2} = \left(\frac{1}{\sqrt{3}} - 1\right)\frac{1}{2}\begin{pmatrix} 1 & 1 \\ \\ 1 & 1 \end{pmatrix}.$$

Тепер обчислимо $(\beta - 1)\frac{\xi_k \xi_k^T}{2}$:

$$(\beta - 1)\frac{\xi_k \xi_k^T}{2} = \frac{1 - \sqrt{3}}{2\sqrt{3}}\begin{pmatrix} 1 & 1 \\ \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}} \\ \\ \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}} \end{pmatrix}.$$

Тепер знайдемо $R_\beta(\xi_k)$:

$$R_\beta(\xi_k) = I + \begin{pmatrix} \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}} \\ \\ \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}} \\ \\ \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}} \end{pmatrix} = \begin{pmatrix} 1 + \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}} \\ \\ \frac{1-\sqrt{3}}{2\sqrt{3}} & 1 + \frac{1-\sqrt{3}}{2\sqrt{3}} \end{pmatrix}.$$

Тепер обчислимо кінцеві значення:

$$1+\frac{1-\sqrt{3}}{2\sqrt{3}}=1+\frac{1}{2\sqrt{3}}-\frac{\sqrt{3}}{2\sqrt{3}}=1+\frac{1}{2\sqrt{3}}-\frac{1}{2}=\frac{2\sqrt{3}+1-\sqrt{3}}{2\sqrt{3}}=\frac{\sqrt{3}+1}{2\sqrt{3}}.$$

Таким чином,

$$R_\beta(\xi_k)=\begin{pmatrix}\frac{\sqrt{3}+1}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}}\\ \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{\sqrt{3}+1}{2\sqrt{3}}\end{pmatrix}.$$

Так як B_0 - одинична матриця, то

$$B_1=B_0R_\beta(\xi_k)=I\cdot R_\beta(\xi_k)=R_\beta(\xi_k).$$

$$B_1=\begin{pmatrix}\frac{\sqrt{3}+1}{2\sqrt{3}} & \frac{1-\sqrt{3}}{2\sqrt{3}}\\ \frac{1-\sqrt{3}}{2\sqrt{3}} & \frac{\sqrt{3}+1}{2\sqrt{3}}\end{pmatrix}.$$

Критерій зупинки на цьому етапі не виконується, що свідчить про необхідність наступних ітерацій.

Це був аналітичний розрахунок значень, необхідних на кожному кроці. Звичайно, що мінімізувати цю функцію цілком таким чином буде вкрай довго і складно. Саме тому кінцевий розв'язок ми отримуємо завдяки програмному рішенню. Також вже додамо порівняння з квазіньютонівським методом БФГШ.

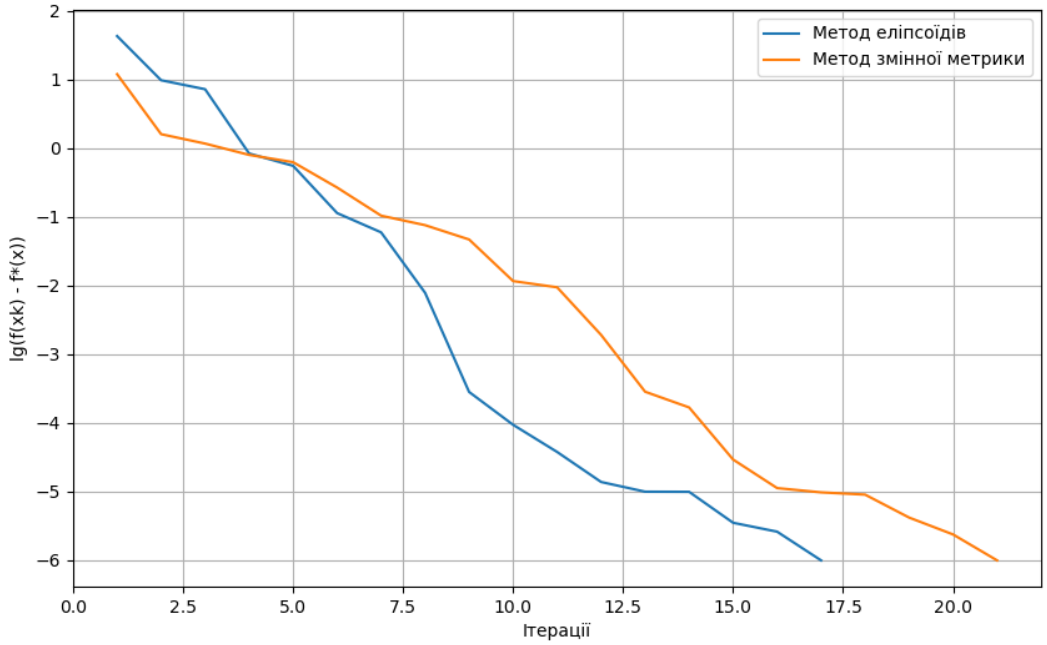


Рис. 2.3

Метод еліпосоїдів: Увесь витраченний час: 0.007633352640979039
Використання пам'яті: 5.72052 КБ
Пікове використання пам'яті: 6.28451 КБ
Метод змінної метрики: Увесь витраченний час: 0.007072524821686281
Використання пам'яті: 5.19221 КБ
Пікове використання пам'яті: 5.6735 КБ

Методи	x_0	$f(x_0)$	x^*	$f(x^*)$	n	time	memory usage
ME	(1;2)	1	(1.325; 0.792)	0	17	0.0076	6.3 КБ
BFGS			(1.325; 0.792)	0	21	0.0071	5.7 КБ

2.1.2 Мінімізація функції з великою кількістю змінних

Важливим для кращого дослідження збіжності є перевірка методом оптимізації функції зі значно більшою кількістю змінних. Обчислювальна складність буде збільшуватися прямо пропорційно до збільшення кількості змінних.
Спробуємо дослідити ці зміни шляхом мінімізації наступної функції.

$$\phi(x) = \sum_{i=1}^{60}(x_i - i)^2; \xi_0 = (0; 1)$$

$B_0 = I_n$ - одинична матриця.

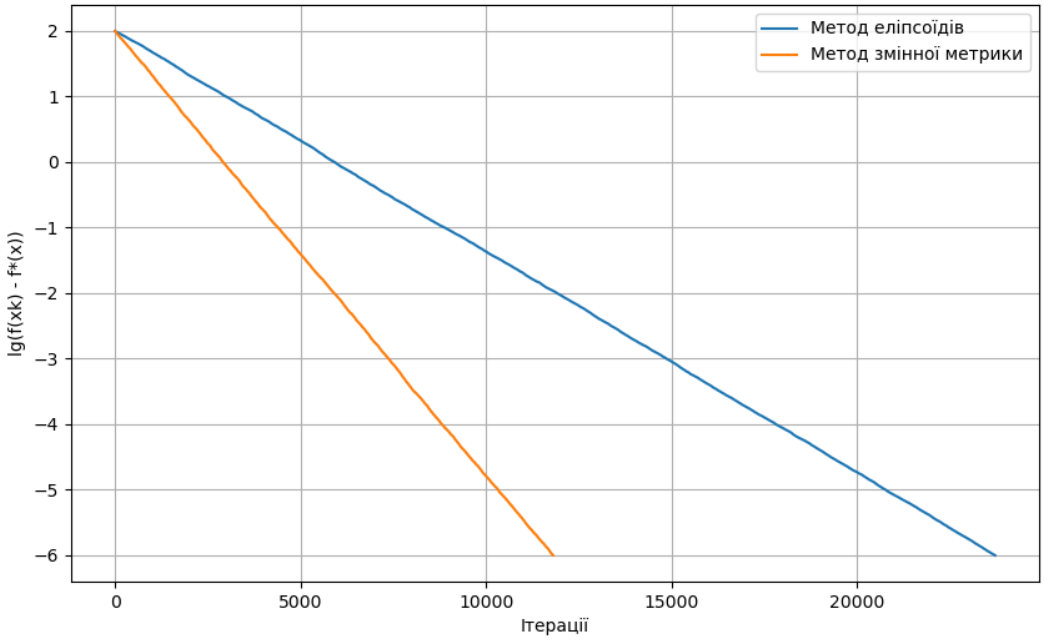


Рис. 2.4

Метод еліпосоїдів: Кількість ітерацій: 23719
Увесь витраченний час: 1.8923481
Використання пам'яті: 361.17362 КБ
Пікове використання пам'яті: 417.993214 КБ

Метод змінної метрики: Кількість ітерацій: 11806
Увесь витраченний час: 1.000000539
Використання пам'яті: 195.54207 КБ
Пікове використання пам'яті: 205.622315 КБ

Як можна побачити, велика кількість змінних погіршує загальну ефективність методу і значно зменшує швидкість збіжності, порівняно з методом змінної метрики. Кількість ітерацій, необхідних для мінімізації методом еліпсоїдів, майже в 2 рази більша. Те саме можна сказати і про використаний обсяг пам'яті. Часу знадобилося на 0.89 секунди більше.

Збільшення кількості змінних призводить до зростання розмірності простору, в якому проводиться оптимізація. Це ускладнює пошук глобального мінімуму. BFGS використовує апроксимацію гессіана, що дозволяє зменшити обсяг необхідної пам'яті.

Метод змінної метрики виявився більш ефективним для задач з великою кількістю змінних. Він забезпечує швидшу збіжність та вимагає менше обчислювальних ресурсів, таких як пам'ять та час. Метод еліпсоїдів, хоча і є корисним для певних типів задач, виявляється менш ефективним у випадках з великою кількістю змінних через велику кількість ітерацій та збільшені обчислювальні витрати.

2.2 Порівняння двох методів мінімізації

№	Методи	x_0	$f(x_0)$	x^*	$f(x^*)$	n	time	mem.us.
1	ME	$[-1.2; 1]^T$	24.2	(0.93; 0.99)	2.4×10^{-6}	24	0.0128	5.5 КБ
	BFGS			(0.95; 0.99)	2.2×10^{-6}	28	0.0079	5.2 КБ
2	ME	$[3; -1; 0; 1]^T$	335	(0.1; 0; 0; 0)	1.7×10^{-6}	92	0.0455	22.98 КБ
	BFGS			(0.1; 0; 0; 0)	2.1×10^{-6}	96	0.0351	22.4 КБ
3	ME	$[-1.2; 1]^T$	1538.2	(∞ ; 0.28)	7.29×10^{-7}	24	0.0121	5.7 КБ
	BFGS			(∞ ; 0.48)	7.33×10^{-7}	27	0.0076	5.4 КБ
4	ME	$[1; 1]^T$	106	(2.722; 1.993)	2.41×10^{-6}	25	0.0116	5.7 КБ
	BFGS			(2.801; 1.987)	2.42×10^{-6}	29	0.0081	5.5 КБ
5	ME	$[1; 1]^T$	79520120	(-22.76; -36.221)	0.9×10^{-6}	24	0.0132	5.2 КБ
	BFGS			(-20.59; -37.824)	0.87×10^{-6}	31	0.0108	4.9 КБ
6	ME	(0;0)	50	(0.1; 0.3)	22.1×10^{-8}	24	0.0115	6.1 КБ
	BFGS			(0.1; 0.4)	21.8×10^{-8}	24	0.0117	5.3 КБ
7	ME	$[-1.2; 2; 0]^T$	8.4	(1.03; 0.93; 0.95)	2.6×10^{-6}	22	0.0102	6.2 КБ
	BFGS			(1.1; 0.9; 0.97)	2.2×10^{-6}	24	0.009	5.6 КБ
8	ME	$[2.7; 90; 1500; 10]^T$	1.5×10^{10}	(2.712; 140.1; 1701; 30.98)	0.41×10^{-6}	102	0.0541	22.9 КБ
	BFGS	$1500; 10]^T$		(2.71; 139.9; 1705; 31.4)	0.38×10^{-6}	113	0.0398	21.6 КБ
9	ME	$[2.5; 2.5]^T$	-0.21	(0; -0.98)	1.73×10^{-6}	21	0.0119	5 КБ
	BFGS			(0; -0.96)	1.72×10^{-6}	24	0.0067	4.4 КБ

Табл. 2.1

№	Методи	x_0	$f(x_0)$	x^*	$f(x^*)$	n	time	mem.us.
10	ME	$[2; 0.2]^T$	-0.703	(2.92; 0.491)	3.29×10^{-6}	21	0.0092	5.9 КБ
	BFGS			(2.9; 0.5)	3.36×10^{-6}	23	0.0087	5.6 КБ
11	ME	$[-1.2; 1]^T$	4.96	(1; 0.98)	12.18×10^{-8}	19	0.0092	5.2 КБ
	BFGS			(0.94; 0.99)	12×10^{-8}	21	0.0066	4.7 КБ
12	ME	$[-1.2; 1]^T$	484.12	(0.97; 1)	2.73×10^{-6}	16	0.0077	4.4 КБ
	BFGS			(1; 0.99)	2.9×10^{-6}	18	0.0046	3.7 КБ
13	ME	$[-1.2; 1]^T$	497.3	(0.9; 0.9)	104.1×10^{-9}	21	0.0086	6.1 КБ
	BFGS			(0.9; 0.98)	104.9×10^{-9}	21	0.0071	5.8 КБ
14	ME	$[2; 0.2]$	0.517	(2.92; 0.49)	4.81×10^{-6}	28	0.0079	5.7 КБ
	BFGS			(3; 0.48)	4.69×10^{-6}	32	0.0082	5.6 КБ
15	ME	$[-3; -1; -3; -1]$	12283	(1; 0.9; 0.98; 1)	1.86×10^{-6}	108	0.0578	24 КБ
	BFGS			(1; 1; 0.91; 1)	2.04×10^{-6}	114	0.0441	22.2 КБ
16	ME	$[-3; -1; -0; 1]$	2665	(-0; 0; 0; 0.01)	2.51×10^{-6}	98	0.0552	25.4 КБ
	BFGS			(0; -0; 0.03; 0)	2.59×10^{-6}	104	0.0379	24 КБ
17	ME	$[0.1; 0.1]^T$	-0.027	(1.892; 0.03)	4.19×10^{-6}	12	0.007	3.5 КБ
	BFGS			(1.898; 0.001)	4.24×10^{-6}	19	0.0053	2 КБ

Табл. 2.2

ВИСНОВКИ

Порівнявши обидва методи мінімізації отримали, що метод еліпсоїдів для стандартних функцій Розенброка являє швидшу збіжність. Кількість ітерацій для функцій з двома змінними орієнтовно дорівнює 15-25; з трьома змінними - 45-60; з чотирма - 90-110. У той самий час методу змінної метрики потребується більше ітерацій: для двох змінних - 20-35; для трьох - 55-70; для чотирьох - 100-120.

Але обсяг використаної пам'яті для методу еліпсоїдів помітно більший. Те саме можна сказати і про витрачений час. Це говорить про те, що метод еліпсоїдів має більшу обчислилювальну складність.

Під час порівняння методів для мінімізації функції з великою кількістю змінних було видно, що метод еліпсоїдів для таких задач виявився менш ефективним. Кількість ітерацій, пам'яті і часу, необхідних для розрахунку мінімуму, виявився майже удвічі більший за конкурентний метод. Що ще більш явно свідчить про велику обчислювальну складність методу.

Під час дослідів було помітно, що обсяг пам'яті, кількість ітерацій і час змінювалися в залежності від кількості змінних, що підтверджує основне твердження про залежність швидкості збіжності тільки від кількості змінних.

Головним висновком досліджень є наступне:

Метод еліпсоїдів демонструє швидшу збіжність для опуклих функцій з невеликою кількістю змінних, витрачаючи більше часу і пам'яті. Проте для великих і складних задач він точно не буде найкращим з можливих методів. Збіжність методом еліпсоїдів досягається як при використанні субградієнту, так і градієнту.

В роботі було досліждено збіжність методу еліпсоїдів; детально розглянуто ітеративний алгоритм і алгоритм побудови необхідних для рішення

еліпсоїдів. Порівняли з методом змінної метрики (квазіньютонівським) програмним шляхом і зробили висновки на основі отриманих результатів.

СПИСОК ЛІТЕРАТУРИ

1. Поляк Б. Т. Введення в оптимізацію / Б. Т. Поляк. — М.: Наука, 1983. — 384 с.
2. Шено Я. Д. Про методи спряженого градієнта. Журнал обчислювальної математики і математичної фізики / Я. Д. Шено. — М.: Наука, 1964. — 224 с.
3. Литвин А. О. Процедури розтягування простору: методи пошуку екстремуму без обмежень. / А. О. Литвин. — М.: Наукові вісті НТУУ «КПІ», 2010. — 368 с.
4. Бойко В. В. Метод розтягування простору у задачах неімовірнісної оптимізації. / В. В. Бойко. — М.: Вісник Дніпропетровського університету, 2012. — 416 с.
5. Кисельова О. М. Шевельова А. Є. Чисельні методи оптимізації. / О. М. Кисельова А. Є. Шевельова— М.: Видавництво дніпропетровського національного університету, 2008. — 210 с.
6. Юдін Д. Б. Лінійне програмування. Теорія і кінцеві методи. / Д. Б. Юдін. — М.: Наука, 1969. — 312 с.
7. ШОР Н. З. Методи недифференційованної оптимізації і складні екстримальні задачі: Еврика, 2008. - 272 с.
8. ШОР Н. З. Методи мінімізації негладких функцій і матричні задачі оптимізації: Еврика, 2009. - 240 с.
9. ШОР Н. З. Алгоритми послідовностей і негладкої оптимізації: Еврика, 2012. - 270 с.

РОЗДІЛ 3

ДОДАТОК А

Для створення коду для мінімізації функції методом еліпсоїдів Шено скористаємось алгоритмом, який був описаний у "Введені до оптимізації"Б. Т. Поляка[1].

Код був написан на мові програмування **Python**. Нам знадобляться бібліотеки **numpy** та **matplotlib.pyplot**, які ми оголосимо як **np** та **plt** відповідно. Також додамо бібліотеку **tracemalloc** для відстежування використовуємої пам'яті.

Для опису створення прикладу програми для мінімізації функції методом еліпсоїдів Шено візьмемо наступну тестову функцію:

$$\varphi(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2, x^0 = [-1.2; 1]^T, x^* = [1; 1], \varphi(x^*) = 0$$

3.0.1 Створення основних робочих функцій

```
tracemalloc.start()

def linear_transform(vector, matrix):
    assert len(vector) == matrix.shape[1], "Неправильна
     ↪  розмірність вектора і матриці перетворення."
    transformed_vector = np.dot(matrix, vector)
    return transformed_vector

def func(x1, x2):
    return 100*(x2 - x1**2)**2 + (1 - x1)**2

def subgradient(x1, x2):
    subgrad = []
    subgrad.append(400 * x1**3 - 400 * x1 * x2 + 2 * x1 - 2)
    subgrad.append(200 * (x2 - x1**2))
    return subgrad
```

- **linear-transform(vector, matrix)** - описує лінійне перетворення між вектором та матрицею перетворення. Ця функція будет неодноразово використовуватися для впровадження коректних формул для дослідження мінімуму.
- **func(x1, x2)** - має виводити значення заданої в умові функції для подальших розрахунків.
- **subgradient(x1, x2)** - має виводити значення субградієнту функції для подальших розрахунків.
- **tracemalloc.start** - починає відстежування витрат пам'яті.

3.0.2 Задання головних параметрів та створення матриць для зберігання даних

```
input_vector = np.array([-1.2, 1])
transformation_matrix = np.array([[1, 1], [1, 1]])
matrix_of_x = []
matrix_of_x.append(input_vector)
matrix_of_H = []
matrix_of_H.append(transformation_matrix)
matrix_of_f = []

f_star = func(1, 1)
ak = 1.5
MQN = []
MESH = []
```

- **input-vector** - вектор початкових значень змінних, які далі будуть змінюватись.
- **transformation-matrix** - початкова матриця перетворення, яка далі буде змінювати значення.
- **matrix-of-x** - поки що пуста матриця для зберігання значень змінних.
- **matrix-of-H** - поки що пуста матриця для зберігання значень матриці перетворень.
- **matrix-of-f** - поки що пуста матриця для зберігання значень функції, що буде використовуватись при побудові графіків для порівняння.
- **f-star** - мінімальне значення функції, задане в умові.
- **ak** - початкове значення коефіцієнта розтягування простору, яке далі буде змінюватись.
- **MQN, MESH** - поки що пусті матриці для зберігання часу ітерацій.

3.0.3 Головна частина коду. Пошук глобального мінімуму

Цей етап розберемо детальніше. По-перше, необхідно створити цикл, який буде запускати кожну ітерацію, допоки критерій зупинки не буде досягнут. Також у цій частині коду ми "дістанемо"значення x_1 та x_2 з початкового вектора значень.

```
for _ in range(40):
    x1 = input_vector[0]
    x2 = input_vector[1]
```

Далі починається розрахунок значень функції. Враховуючи те, що нам необхідно зберігати значення $\lg(\varphi(xk) - \varphi(x^*))$, то створимо змінну, яка розраховує це значення, та будемо зберігати його в **matrix-of-f**. Також зберігаємо перше значення функції.

```
f = func(x1,x2)
f1 = func(x1,x2)
a = np.log10(f - f_star)
matrix_of_f.append(a)
```

Наступна частина коду є створенням наступних формул для розрахунку значень, які визначають довжину кроку γ_k, напрямок розтягування простору s^k, наступні змінні x^{k+1} та матрицю перетворень H^{k+1}:

$$\gamma_k = \frac{2(f(xk) - f(x^*))}{(H_k \frac{\partial f(x^k)}{\partial x}, \frac{\partial f(x^k)}{\partial x})}$$

$$s^k = \frac{\partial f(x^k)}{\partial x}$$

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

$$H_{k+1} = H_k - (1 - \frac{1}{\alpha_k^2}) \frac{H_k s^k (s^k)^T H_k}{(H_k s^k, s^k)}, H_0 = I$$

Також ми не забуваємо оновлювати значення коефіцієнта розтягування простору за формолою:

$$\alpha_k = \frac{\|(H_k, x^k)\|}{\|x^k\|}$$

```
vector_grad_values = subgradient(x1, x2)

linear_H_and_s = linear_transform(vector_grad_values,
 ↪  transformation_matrix)
scalar_mult_Hs_and_s = np.dot(linear_H_and_s,
 ↪  vector_grad_values)
numerator_of_gamma = f - f_star
transposed_vector_grad_values =
 ↪  np.transpose(vector_grad_values)

length_of_step_gamma =
 ↪  2*(numerator_of_gamma)/(scalar_mult_Hs_and_s)

numerator_H_p1 =
 ↪  linear_transform(transposed_vector_grad_values,
 ↪  transformation_matrix)

input_vector = input_vector -
 ↪  length_of_step_gamma*linear_H_and_s
matrix_of_x.append(input_vector)

output_vector = linear_transform(input_vector,
 ↪  transformation_matrix)
n1 = np.linalg.norm(input_vector)
n2 = np.linalg.norm(output_vector)
ak = n2/n1

transformation_matrix = transformation_matrix - (1 -
 ↪  1/ak**2)*(np.dot(linear_H_and_s,
 ↪  numerator_H_p1))/(scalar_mult_Hs_and_s)
matrix_of_H.append(transformation_matrix)
```

Додамо зберігання часу, необхідного для проведення однієї ітерації.

```
current_time = time.time()
elapsed_time = current_time - start_time
time.sleep(0.001)
MESH.append(elapsed_time:.5f)
```

Додаємо також критерій зупинки, а саме:

$$\varphi(x) \geq \varphi(x_0) + (\lg(x) - x_0)$$

```
if f >= (f1 + (np.log10(input_vector) - matrix_of_x[0])):
    break

current1, peak1 = tracemalloc.get_traced_memory()
tracemalloc.stop()
```

3.0.4 Створення программи для мінімізації функції квазіньютонівським методом

Для цього можна скористуватися існуючої функцією в бібліотеці **scipy.optimize** і додамо функцію зворотнього виклику для збереження значень функції.

```
tracemalloc.start()
def callback(x):
    func_values.append(np.log10(func(x) - f_star))

result = minimize(func, input_vector, method='BFGS',
 ↪  jac=gradient, options={'disp': True, 'callback':
 ↪  callback})
```

Зберігання часу, необхідного для однієї ітерації.

```
current_time1 = time1.time()
elapsed_time1 = current_time1 - start_time1
time.sleep(0.001)
MQN.append(elapsed_time1:.5f)
current2, peak2 = tracemalloc.get_traced_memory()
tracemalloc.stop()
```

3.0.5 Побудова графіку для порівняння методів

Для цього скористаємося бібліотекою **matplotlib.pyplot**.

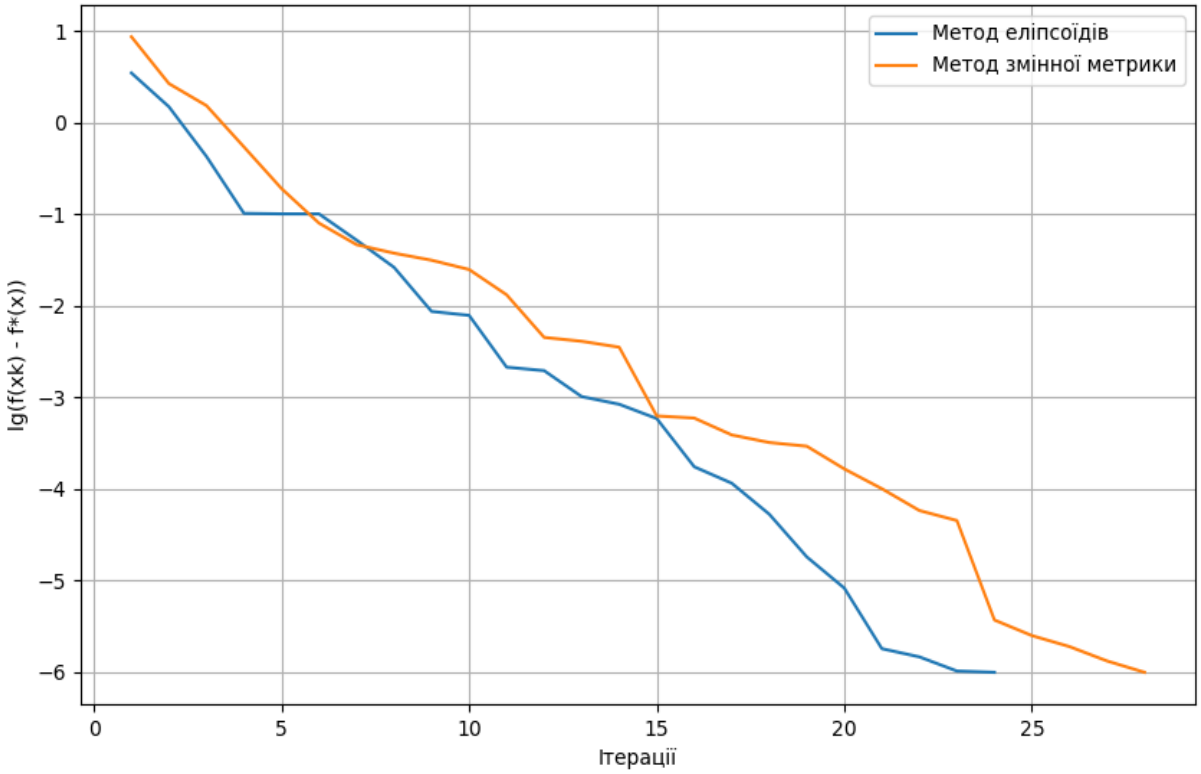
```
plt.figure()
plt.plot(matrix_of_f, label='Метод еліпсоїдів')
plt.plot(func_values, label='Квазіньютонівський метод')

plt.xlabel('Ітерації')
plt.ylabel('lg(f(xk) - f*(x))')
plt.title('Графік')
plt.legend()

plt.show()

print("Метод еліпсоїдів Шено: ", MESH)
print("Всього часу витрачено: ", sum(MESH))
print(f"Використання пам'яті: {current1 / 1024} КБ")
print(f"Пікове використання пам'яті: {peak1 / 1024} КБ")
print("Квазіньютонівський метод: ", MQN)
print("Всього часу витрачено: ", sum(MQN))
print(f"Використання пам'яті: {current2 / 1024} КБ")
print(f"Пікове використання пам'яті: {peak2 / 1024} КБ")
```

3.0.6 Результат роботи коду



Метод еліпосоїдів: [0.0006218408940321583, 0.0002021800394260549, 0.0005432657266138759, 0.0008661259703966495, 0.0002930815662679544 5, 0.0006266276675614527, 0.0001535718374800095 7, 0.0007175810112904287, 0.000688262885356373, 0.0005767946324904881, 0.0002459192592306577, 0.0009382865245480597, 0.0007074596947302702, 0.0004907316217106227, 5.948722138823015e-05, 0.0006218094693912916, 0.0008312994089994593, 0.0003553221078876573, 0.0006370331486390872, 0.000893004452497705, 0.0007379107706093077, 0.0001798066261705255 8, 0.0007657938795013499]

Увесь витраченний час: 0.01275319641621967

Використання пам'яті: 5.390625 КБ

Пікове використання пам'яті: 5.5078125 КБ

Метод змінної метрики: [0.0004190082692568949, 0.0002327326704515441, 0.0002683129339440198, 0.0003689515651838653, 5.739272788899469e-05,

0.00028570189586749783, 0.0005219851680503911, 2.2401676009770962e-05, 5.473033012114652e-05, 0.0002690145637119302, 0.0004812220018308761, 0.00022156693049239274, 0.0002017006795714675, 0.00015207235077144233, 0.0003697218741884493, 0.0002837310013869482, 0.00014942251408107596, 0.0005569056871141471, 0.0004740282601423805, 7.615932589831013e-05, 0.0005677117736377613, 0.0005864734697946769, 0.0002815004840715017, 0.0003500551795951137, 0.00043178836645972184, 7.519121794962553e-05, 0.00015616980422079592]

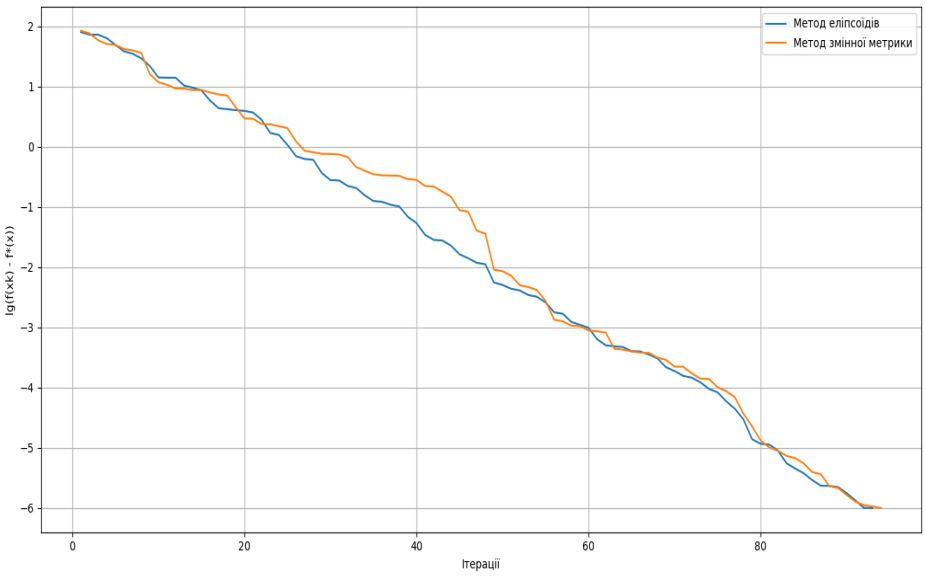
Увесь витраченний час: 0.00791565272169274

Використання пам'яті: 5.078125 КБ

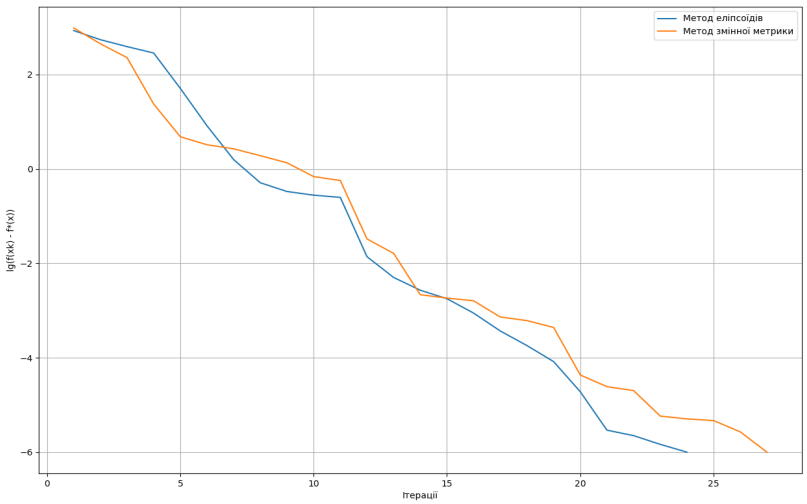
Пікове використання пам'яті: 5.1953125 КБ

ДОДАТОК Б

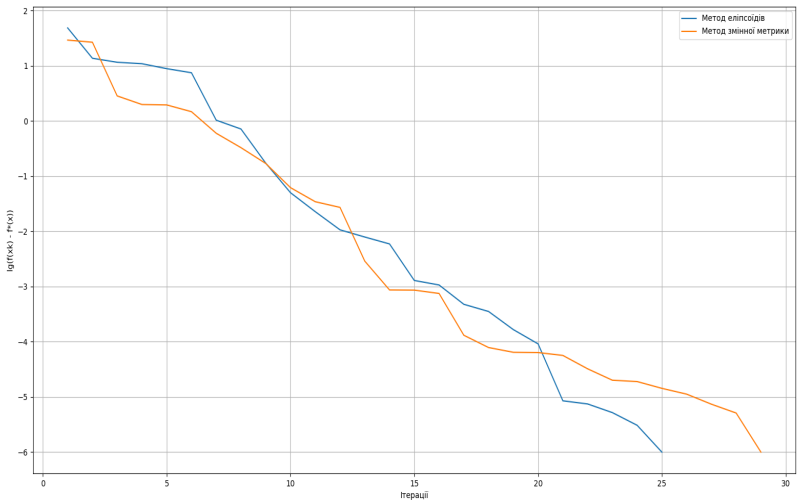
Дослідження тестових функцій



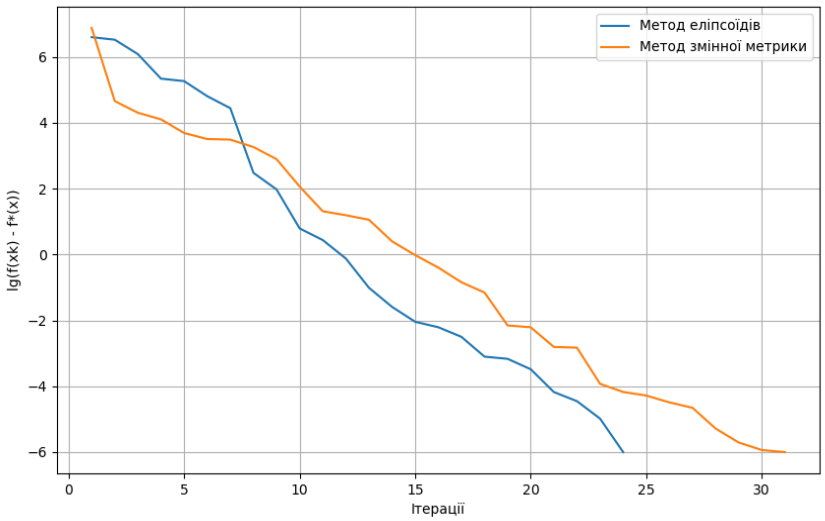
2. $\varphi(x) = (x_1 - 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4, x^0 = [3; -1; 0; 1]^T, x^0 = [1; 1; 1; 1], x^* = (0; 0; 0; 0;), \varphi(x^*) = 0$



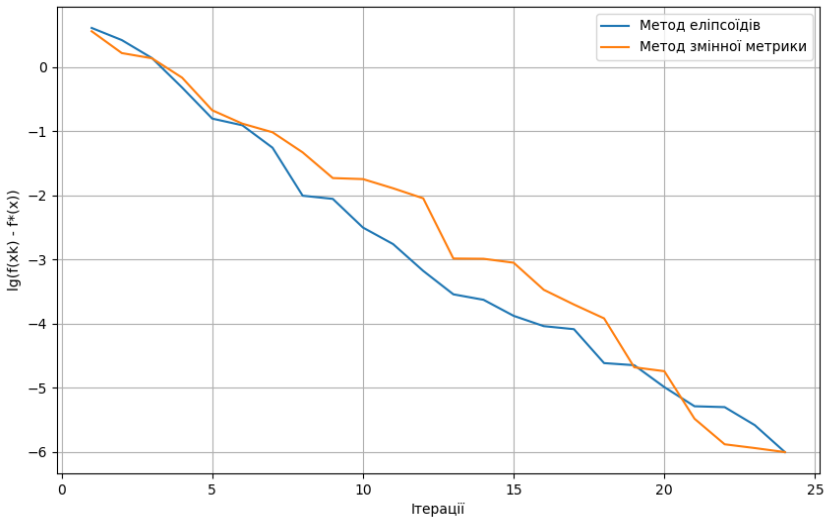
3. $\varphi(x) = (x_1 x_2)^2(1 - x_1^2)[1 - x_1 - x_2(1 - x_1)^5]^2, x^0 = [-1.2; 1]^T, x^0 = [1; \infty]^T, x^0 = [0; \infty]^T, x^* = [\infty; 0]^T, \varphi(x^*) = 0$



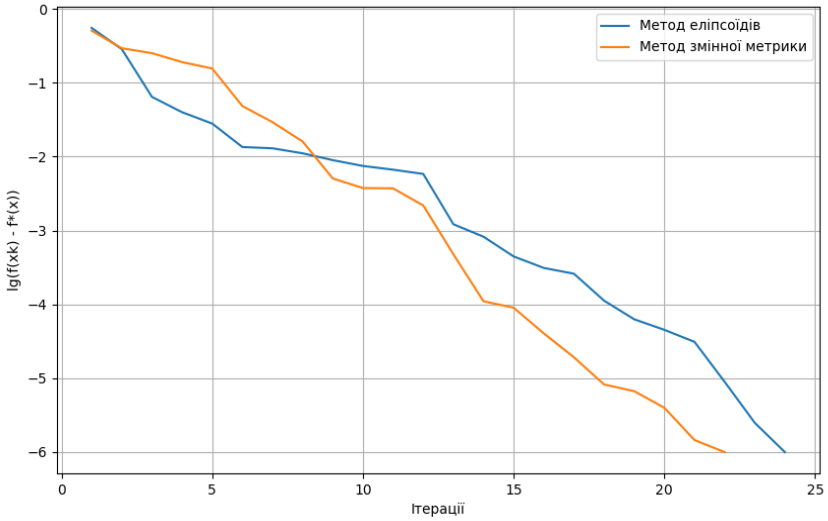
4. $\varphi(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^-7)^2, x^0 = [1; 1]^T, x^* = [3.38443; -1.84812]^T, x^* = [3; 2]^T, \varphi(x^*) = 0$



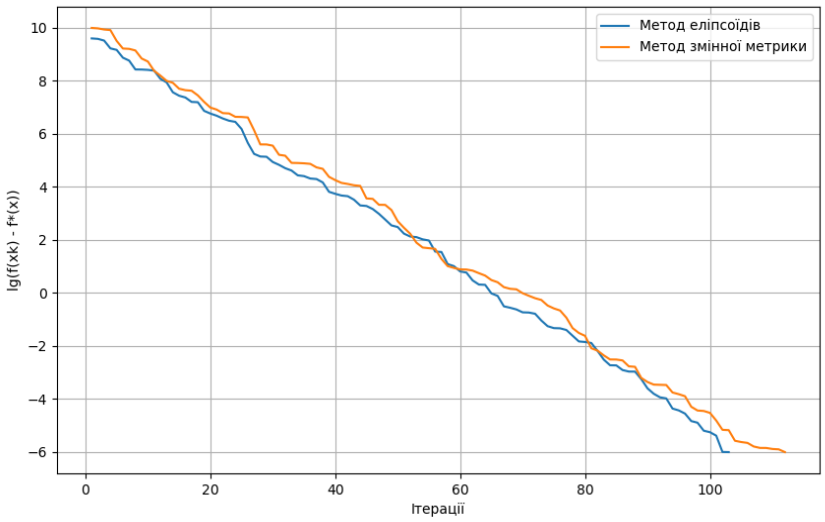
5. $\varphi(x) = (x_1^2 + 12x_2 - 1)^2 + (48x_1^2 + 49x_2^2 + 84x_1 + 2324x_2 - 681)^2, x^0 = [1; 1]^T, x^* = [0.28581; 0.27936]^T, \varphi(x^*) = 5.9225, x^* = [-21.026653; -36.7660090]^T, \varphi(x^*) = 0$



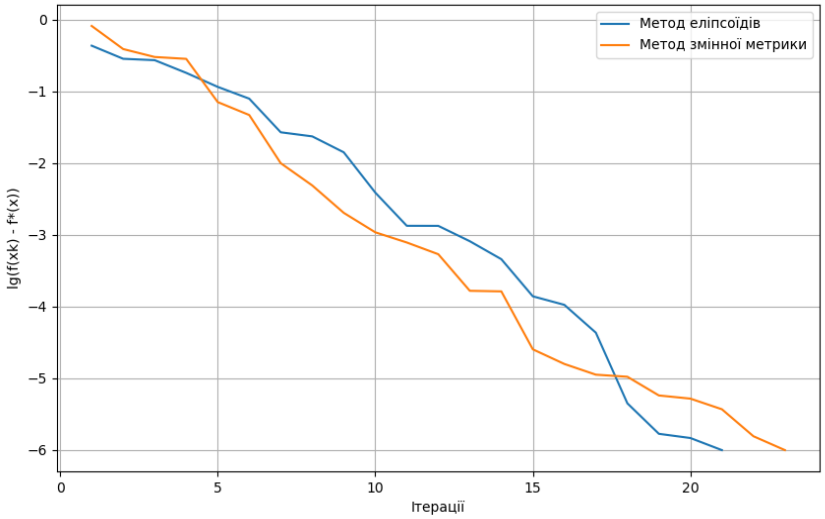
6. $\varphi(x) = (x_1^2 + 12x_2 - 1)^2 + (x_1 + x - 2^2 - 7)^2$



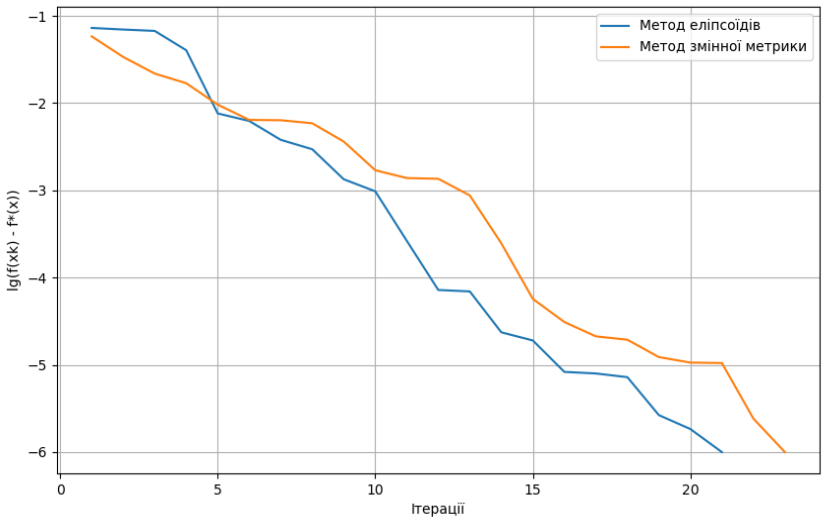
7. $\varphi(x) = 100[x_3 - (\frac{x_1+x_2}{2})^2]^2 + (1 - x_1)^2 + (1 - x_2)^2, x^0 = [-1.2; 2; 0]^T, x^* = [1; 1; 1], \varphi(x^*) = 0$



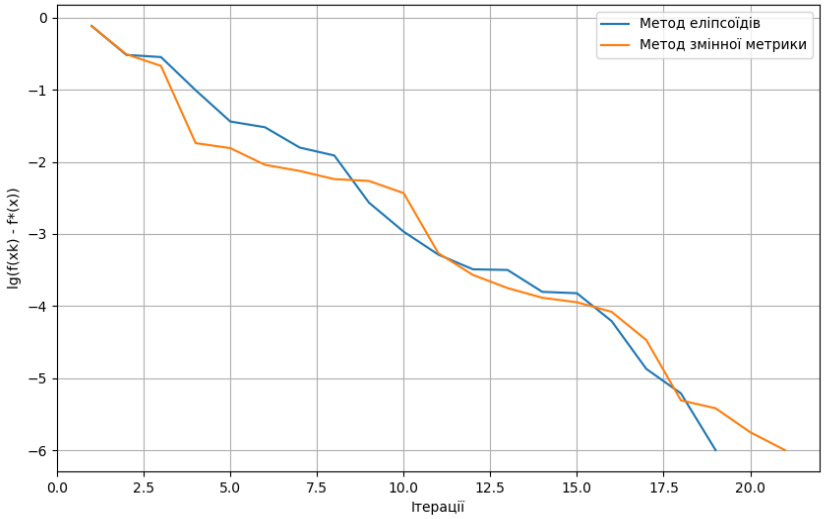
8. $\varphi(x) = 10^4 \sum_{i=1}^{7} (\frac{\frac{x_1^2 + x_2^2 a_i + x_3^2 a_i^2}{1 + x_4^2 a_i} - b_i}{b_i}), x^0 = [2.7; 90; 1500; 10]^T, x^* = [2.714; 140.4; 1707; 31.51]^T, \varphi(x^*) = 318.572$



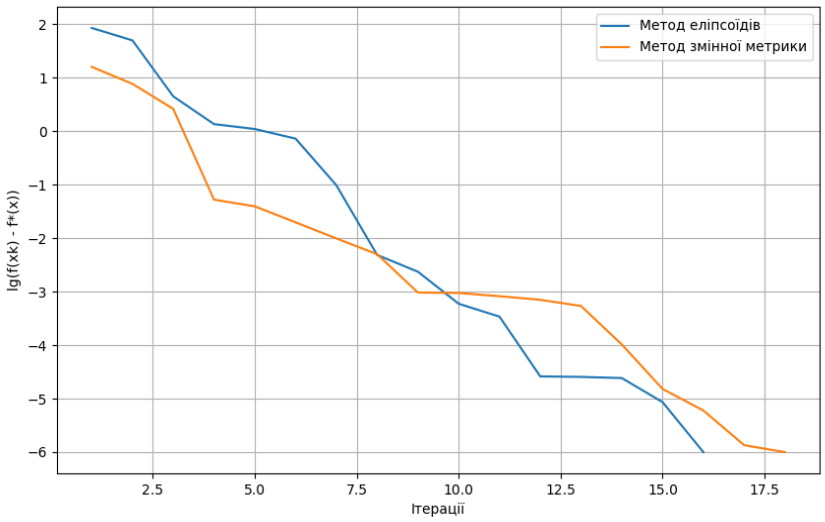
9. $\varphi(x) = -e^{-x_1 - x_2}(2x_1^2 + 3x_2^2), x^0 = [2.5; 2.5]^T, x^* = [0; 1]^T, x^* = [0; -1]^T, \varphi(x^*) = 1.1036$



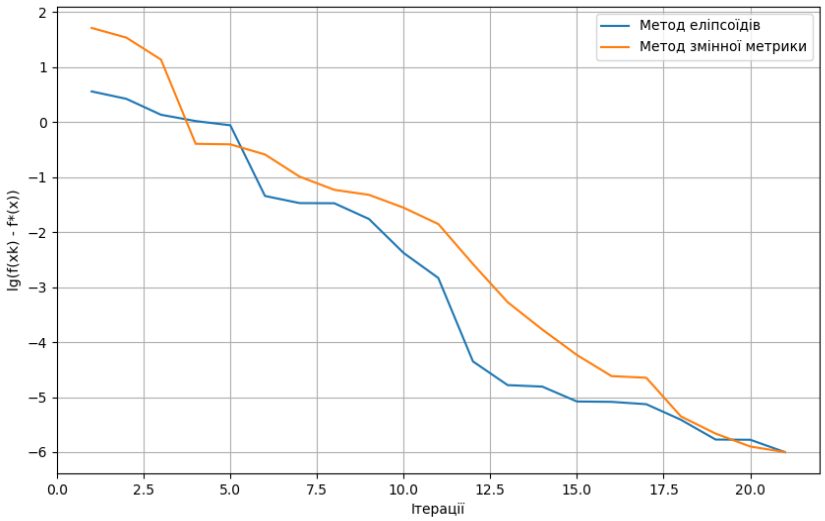
10. $\varphi(x) = u_1^2 + u_2^2 + u_3^2, u_i = c_i - x_1(1 - x_1), c_1 = 1.5, c_2 = 2.25, c_3 = 2.625, x^0 = [2; 0.2]^T, x^* = [3; 0.5]^T, \varphi(x^*) = 0$



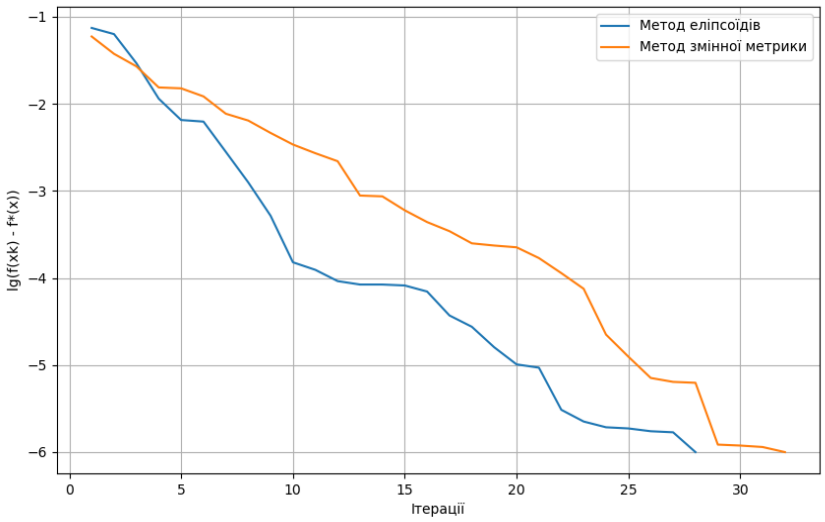
11. $\varphi(x) = (x_2 - x_1^2)^2 + (1 - x_1)^2, x^0 = [-1.2; 1]^T, x^* = [1; 1], \varphi(x^*) = 0$



12. $\varphi(x) = (x_2 - x_1^2)^2 + 100(1 - x_1)^2, x^0 = [-1.2; 1]^T, x^* = [1; 1]^T, \varphi(x^*) = 0$

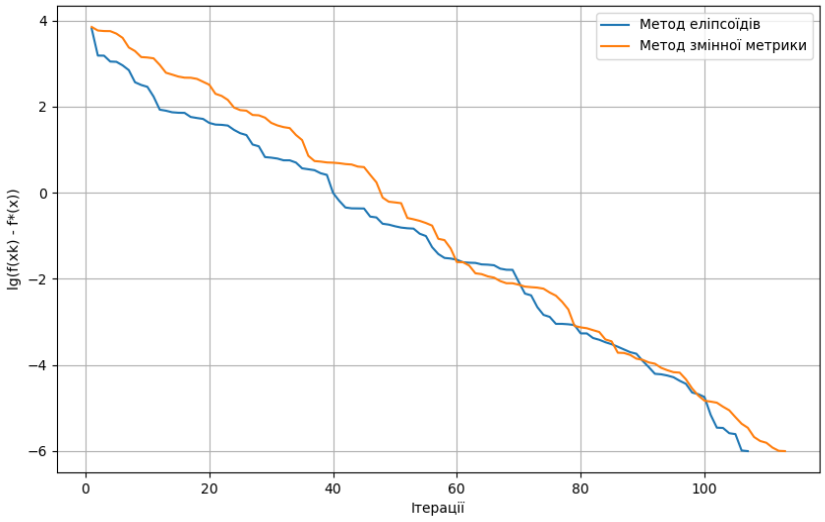


13. $\varphi(x) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2, x^0 = [-1.2; 1]^T, x^* = [1; 1]^T, \varphi(x^*) = 0$



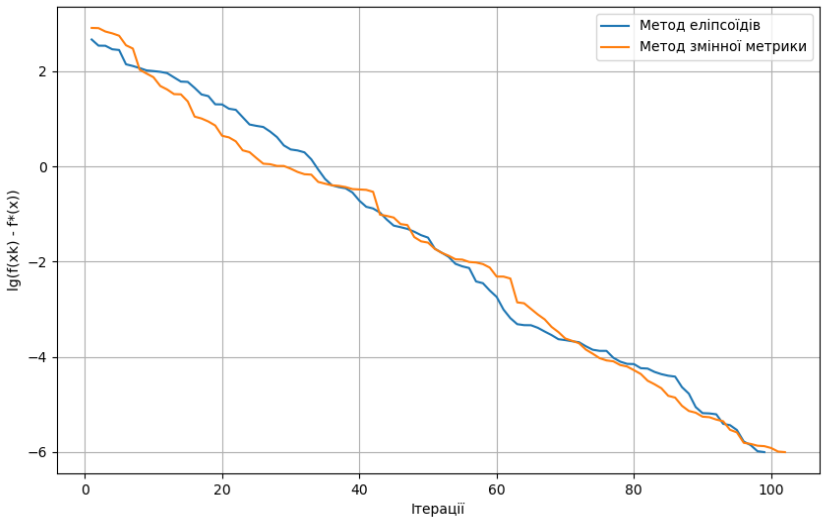
14.

$\varphi(x) = [1.5 - x_1(1 - x_2)]^2 + [2.23 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2, x^0 = [2; 0.2], x^* = [3; 0.5]^T, \varphi(x^*) = 0$

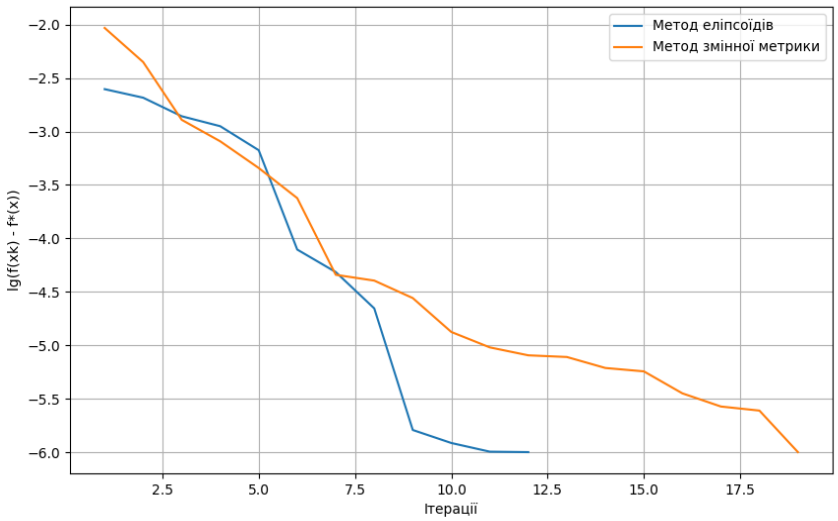


15.

$\varphi(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2) + (1 - x_3)^3 + 10.1[(x - 2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1), x^0 = [-3; -1; -3; -1], x^* = [1; 1; 1; 1], \varphi(x^*) = 0$



16. $\varphi(x) = (x_1 + 40x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4, x^0 = [-3; -1; -0; 1], x^* = [0; 0; 0; 0], \varphi(x^*) = 0$



17.

$\varphi(x) = -x_1^2 e[1 - x_1^2 - 20.25(x_1 - x_2)^2], x^0 = [0.1; 0.1]^T, x^* = [1.9; 0], \varphi(x^*) = 0$

ДОДАТОК В

Перевірка часу і пам'яті, необхідного для мінімізації функції до точності 10^{-6}

2. Метод еліпосоїдів: Увесь витраченний час: 0.0455014420417647
Використання пам'яті: 22.865625 КБ
Пікове використання пам'яті: 22.9828125 КБ
Метод змінної метрики: Увесь витраченний час: 0.0351311657376778
Використання пам'яті: 22.125 КБ
Пікове використання пам'яті: 22.2421875 КБ

3. Метод еліпосоїдів: Увесь витраченний час: 0.012069326175242416
Використання пам'яті: 5.5060546875 КБ
Пікове використання пам'яті: 5.6962890625 КБ
Метод змінної метрики: Увесь витраченний час: 0.007608522736236984
Використання пам'яті: 5.265625 КБ
Пікове використання пам'яті: 5.3828125 КБ

4. Метод еліпосоїдів: Увесь витраченний час: 0.0116695730504845
Використання пам'яті: 5.5859375 КБ
Пікове використання пам'яті: 5.703125 КБ
Метод змінної метрики: Увесь витраченний час: 0.008174175017382705
Використання пам'яті: 5.390625 КБ
Пікове використання пам'яті: 5.5078125 КБ

5. Метод еліпосоїдів: Увесь витраченний час: 0.01322109638865986
Використання пам'яті: 5.078125 КБ
Пікове використання пам'яті: 5.1953125 КБ
Метод змінної метрики: Увесь витраченний час: 0.010893250714606244
Використання пам'яті: 4.9158203125 КБ
Пікове використання пам'яті: 4.9990078125 КБ

6. Метод еліпосоїдів: Увесь витраченний час: 0.011511227098186945
Використання пам'яті: 5.6474609375 КБ
Пікове використання пам'яті: 6.185546875 КБ
Метод змінної метрики: Увесь витраченний час: 0.01179589936572223
Використання пам'яті: 5.1875 КБ
Пікове використання пам'яті: 5.3046875 КБ

7. Метод еліпосоїдів: Увесь витраченний час: 0.010217937513989254
Використання пам'яті: 5.9853515625 КБ
Пікове використання пам'яті: 6.154296875 КБ
Метод змінної метрики: Увесь витраченний час: 0.009007939959506212
Використання пам'яті: 5.24544554 КБ
Пікове використання пам'яті: 5.6355214 КБ

8. Метод еліпосоїдів: Увесь витраченний час: 0.054052826875867384
Використання пам'яті: 21.96378173 КБ
Пікове використання пам'яті: 22.8691103 КБ
Метод змінної метрики: Увесь витраченний час: 0.03984873644203846
Використання пам'яті: 21.034018 КБ
Пікове використання пам'яті: 21.62118390 КБ

9. Метод еліпосоїдів: Увесь витраченний час: 0.011946808067191622
Використання пам'яті: 4.571183 КБ
Пікове використання пам'яті: 5.00167 КБ
Метод змінної метрики: Увесь витраченний час: 0.006743999628978488
Використання пам'яті: 4.03391 КБ
Пікове використання пам'яті: 4.4419 КБ

10. Метод еліпосоїдів: Увесь витраченний час: 0.0091701241097240 63
Використання пам'яті: 5.59026 КБ
Пікове використання пам'яті: 5.993214 КБ
Метод змінної метрики: Увесь витраченний час: 0.0086584539035324 43
Використання пам'яті: 5.12397 КБ
Пікове використання пам'яті: 5.622315 КБ

11. Метод еліпосоїдів: Увесь витраченний час: 0.0091861667539846 03
Використання пам'яті: 4.94264 КБ
Пікове використання пам'яті: 5.24778 КБ
Метод змінної метрики: Увесь витраченний час: 0.0067218291474179 88
Використання пам'яті: 4.230819 КБ
Пікове використання пам'яті: 4.719390 КБ

12. Метод еліпосоїдів: Увесь витраченний час: 0.0076693259218154 975
Використання пам'яті: 3.937 КБ
Пікове використання пам'яті: 4.449523 КБ
Метод змінної метрики: Увесь витраченний час: 0.0045532934611942 34
Використання пам'яті: 3.02918 КБ
Пікове використання пам'яті: 3.72673 КБ

13. Метод еліпосоїдів: Увесь витраченний час: 0.0085842906470663 06
Використання пам'яті: 5.772351 КБ
Пікове використання пам'яті: 6.147892 КБ
Метод змінної метрики: Увесь витраченний час: 0.0070453192664945 655
Використання пам'яті: 5.296485 КБ
Пікове використання пам'яті: 5.81052 КБ

14. Метод еліпосоїдів: Увесь витраченний час: 0.007928484288995606
Використання пам'яті: 5.4874 КБ
Пікове використання пам'яті: 5.73098 КБ
Метод змінної метрики: Увесь витраченний час: 0.008213190111663148
Використання пам'яті: 5.0459 КБ
Пікове використання пам'яті: 5.633109 КБ

15. Метод еліпосоїдів: Увесь витраченний час: 0.05799392444815631
Використання пам'яті: 22.93401 КБ
Пікове використання пам'яті: 24.000512 КБ
Метод змінної метрики: Увесь витраченний час: 0.04404647451987011
Використання пам'яті: 20.7245801 КБ
Пікове використання пам'яті: 22.14529 КБ

16. Метод еліпосоїдів: Увесь витраченний час: 0.05521765645779585
Використання пам'яті: 24.65275 КБ
Пікове використання пам'яті: 25.38192 КБ
Метод змінної метрики: Увесь витраченний час: 0.03793291282209131
Використання пам'яті: 23.986168 КБ
Пікове використання пам'яті: 24.01172 КБ

17. Метод еліпосоїдів: Увесь витраченний час: 0.007020503906268837
Використання пам'яті: 2.67143 КБ
Пікове використання пам'яті: 3.5297312 КБ
Метод змінної метрики: Увесь витраченний час: 0.005318449795202718
Використання пам'яті: 1.99563 КБ
Пікове використання пам'яті: 2.03481 КБ