Abstract. The article is devoted to the study of the application of Lagrange interpolation polynomials for interpolating data related to the placement of information on web nodes and activities on web portals. Statistical indicators of website use for presenting information and interacting with users are analyzed. The Lagrange polynomial is used to process and approximate values, which allows the construction of an interpolation function based on available data. The results demonstrate the possibility of using interpolation methods to analyze and evaluate changes in indicators in the field of web resource use. The study shows the feasibility of using mathematical methods in data analysis tasks related to the functioning of web portals.

Keywords: calculation methods, Lagrange polynomial, interpolation, web nodes, web portals, statistical data.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Метод інтерполяції для прогнозування метрик використання хмарних обчислень в статистичному навчанні / Н. А. Потапова, Л. О. Волонтир, І. П. Частоколенко, М. С. Григоренко. *Наука і техніка сьогодні.* 2024. № 4(32). С. 1192–1205. URL: http://perspectives.pp.ua/index.php/nts/article/view/10943

2. Потапова Н. А., Комар О. О. Ефективність застосування інтерполяційного поліному Лагранжа для побудови моделей прогнозу капітальних інвестицій. *Вісник студентського наукового товариства ДонНУ імені Василя Стуса.* 2023. Т. 2, № 15. URL: https://jvestnik-sss.donnu.edu.ua/article/view/14715

3. Чисельні методи: навчальний посібник / Л. О. Волонтир, О. В. Зелінська, Н. А. Потапова, І. А. Чіков. Вінниця: ВНАУ, 2020. 322 с. URL: https://r.donnu.edu.ua/handle/123456789/1805

4. Burden R. L., Faires J. D. Numerical analysis. 9th ed. Boston: Cengage Learning, 2011. 888 p. URL: https://faculty.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf

5. Офіційний сайт Державної служби статистики України. URL: https://www.ukrstat.gov.ua/operativ/operativ2018/zv/ikt/arh_ikt_u.html

УДК 004:005:51

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ОДНОВИМІРНОЇ ОПТИМІЗАЦІЇ

С. О. Головня, О. А. Павлюк

Анотація. У статті проведено порівняльний аналіз методів одновимірної оптимізації, зокрема методу бісекції (дихотомії) та методу золотого перетину. Дослідження базується на зіставленні швидкості збіжності алгоритмів, кількості необхідних обчислень та їх загальної ефективності для різних типів цільових функцій. Практична реалізація та програмне тестування розглянутих математичних методів виконані за допомогою мови C#. На основі отриманих результатів сформовано зведену таблицю порівнянь, яка наочно демонструє переваги, недоліки та оптимальні умови застосування кожного з підходів.

Ключові слова: одновимірна оптимізація, метод бісекції, метод золотого перетину, швидкість збіжності, C#.

Вступ. Сучасний етап розвитку інформаційних технологій характеризується постійним ускладненням обчислювальних задач. Однією з фундаментальних проблем у сфері комп'ютерних наук та прикладної математики є задача оптимізації. Знаходження мінімуму або максимуму цільової функції виступає базовим етапом у процесах машинного навчання, аналізу даних та інженерного моделювання. Особливе місце серед таких задач займає одновимірна оптимізація, яка часто є допоміжним, але критично важливим кроком у багатокрокових алгоритмах.

Ефективність вирішення глобальних задач безпосередньо залежить від швидкості та точності роботи одновимірних методів пошуку. Серед класичних та найбільш поширених підходів виділяють методи виключення інтервалів. До цієї категорії належать метод бісекції (дихотомії) та метод золотого перетину. Основна ідея цих алгоритмів полягає у послідовному звуженні інтервалу, на якому локалізовано екстремум функції. Незважаючи на спільний базовий принцип, ці два методи суттєво відрізняються підходами до вибору точок ділення інтервалу.

Аналіз наукової літератури показує, що вибір оптимального методу залишається актуальною практичною проблемою для розробників. Метод бісекції забезпечує надійне ділення відрізка навпіл, проте часто потребує обчислення додаткових значень функції на кожному кроці. Метод золотого перетину використовує специфічну пропорцію, що дає змогу перевикористовувати попередні результати та зменшити кількість нових обчислень.

Для розробників важливо розуміти не лише теоретичну оцінку швидкості збіжності, але й те, як ці алгоритми поводяться на практиці під час реалізації конкретними мовами програмування, зокрема C#. Відмінності у типах цільових функцій (поліноміальні, тригонометричні

тощо) можуть суттєво впливати на реальну кількість ітерацій та загальний час виконання програми.

Метою статті є проведення порівняльного аналізу методів одновимірної оптимізації – методу бісекції та методу золотого перетину. У межах дослідження було здійснено програмну реалізацію обох алгоритмів мовою програмування C#. Основну увагу приділено зіставленню швидкості їх збіжності, кількості необхідних обчислень та оцінці загальної обчислювальної ефективності, що дасть змогу сформувати практичні рекомендації щодо доцільності використання кожного з методів.

Основна частина. Для проведення порівняльного аналізу методів одновимірної оптимізації необхідно розглянути їх математичні моделі. Нехай задано цільову функцію $f(x)$, яка є унімодальною на початковому відрізку $[a, b]$. Задача полягає у знаходженні точки $x^* \in [a, b]$, в якій функція досягає свого мінімуму з заданою точністю ε. Алгоритм методу дихотомії передбачає ділення поточного інтервалу навпіл. На кожній ітерації k обчислюються дві точки x_1 та x_2, симетрично розташовані відносно середини відрізка на малій відстані δ(де $\delta < \frac{\varepsilon}{2}$):

$$x_1 = \frac{a_k + b_k}{2} - \delta; \quad (1)$$

$$x_2 = \frac{a_k + b_k}{2} + \delta, \quad (2)$$

де k – номер поточної ітерації;

a_k та b_k – межі інтервалу на k-му кроці;

x_1 та x_2 – внутрішні точки для обчислення функції;

δ – константа розрізнення;

ε – задана точність пошуку мінімуму (допустима похибка).

Після обчислення значень $f(x_1)$ та $f(x_2)$ інтервал звужується: якщо $f(x_1) < f(x_2)$, то новим інтервалом стає $[a_k; x_2]$; інакше – $[x_1; b_k]$. Особливістю методу є те, що на кожній новій ітерації доводиться заново обчислювати функцію у двох нових точках.

Метод золотого перетину оптимізує кількість обчислень завдяки властивості пропорції золотого перетину. Точки ділення інтервалу розраховуються за формулами:

$$x_1 = a_k + \frac{3 - \sqrt{5}}{2}(b_k - a_k); \quad (3)$$

$$x_2 = a_k + \frac{\sqrt{5} - 1}{2}(b_k - a_k). \quad (4)$$

Головна перевага цього підходу полягає в тому, що на наступному кроці одна з точок (x_1 або x_2) збігається з точкою з попередньої ітерації. Тому функцію доводиться обчислювати лише один раз. Це суттєво знижує загальне обчислювальне навантаження, особливо для складних цільових функцій.

Для проведення порівняльного аналізу розроблено консольний додаток мовою програмування C#. Базова структура програми передбачає використання делегатів Func<double, double>, що дає змогу передавати різні цільові функції в алгоритми оптимізації як параметри. Для дослідження обрано три тестові функції різної математичної природи:

1. Поліноміальна – $f_1(x) = (x - 2)^2 + 3$ на відрізку [0,5].

2. Трансцендентна – $f_2(x) = e^x - 4x$ на відрізку [0,3].

3. Комбінована – $f_3(x) = x^2 + 2\,cos(x)$ на відрізку $[-2,2]$.

Алгоритми реалізовано у вигляді окремих статичних методів, які відстежують не лише знайдене значення мінімуму, але й три ключові метрики ефективності: кількість ітерацій циклу, загальну кількість обчислень цільової функції, а також час виконання алгоритму в мілісекундах. Фрагмент коду методу дихотомії наведено на рис. 1.

Аналізуючи наведений програмний код методу дихотомії, варто звернути увагу на умову зупинки ітераційного процесу. Головний цикл while продовжує роботу доти, поки довжина поточного інтервалу Math.Abs(b - a) перевищує задану похибку epsilon. Додаткова умова iterations < 1000 відіграє критично важливу роль апаратного запобіжника.

Оскільки комп'ютерна арифметика дійсних чисел з плаваючою комою (тип даних double) має обмежену точність представлення дробових значень, алгоритм може потрапити в стан нескінченного зациклення. Обмеження максимальної кількості ітерацій гарантує стабільне завершення роботи програми за будь-яких умов.

Метод золотого перетину дає змогу зекономити обчислювальні ресурси завдяки перевикористанню раніше знайдених точок.

```
    static double BisectionMethod(double a, double b, double epsilon)
{
   int iterations = 0;
   int functionCalls = 0;
   double delta = epsilon / 4.0;
   while (Math.Abs(b - a) > epsilon && iterations < 1000)
   {
      iterations++;
      double x1 = (a + b) / 2.0 - delta;
      double x2 = (a + b) / 2.0 + delta;
      double f1 = func(x1);
      double f2 = func(x2);
      functionCalls += 2;
      if (f1 < f2)
         b = x2;
      else
         a = x1;
   }
    return (a + b) / 2.0;
}
```

Рис. 1. Фрагмент коду методу дихотомії

Програмна реалізація методу золотого перетину наведена на рис. 2.

```
    static double GoldenSectionMethod(double a, double b, double epsilon)
{
   int iterations = 0;
   int functionCalls = 0;
   double phi = (1.0 + Math.Sqrt(5.0)) / 2.0;
   double resphi = 2.0 - phi;
   double x1 = a + resphi * (b - a);
   double x2 = b - resphi * (b - a);
   double f1 = func(x1);
   double f2 = func(x2);
   functionCalls += 2;
   while (Math.Abs(b - a) > epsilon && iterations < 1000)
   {
      iterations++;
      if (f1 < f2)
      {
         b = x2;
         x2 = x1;
         f2 = f1;
```

```
            x1 = a + resphi * (b - a);
            f1 = func(x1);
        }
        else
        {
            a = x1;
            x1 = x2;
            f1 = f2;
            x2 = b - resphi * (b - a);
            f2 = func(x2);
        }
        functionCalls++;
    }
    return (a + b) / 2.0;
}
```

Рис. 2. Програмна реалізація методу золотого перетину

Програмна реалізація методу золотого перетину містить низку архітектурних оптимізацій. Умова виходу з ітераційного циклу є ідентичною до методу дихотомії, що забезпечує чистоту експерименту та дає змогу коректно порівнювати ці алгоритми за однакової точності ε. Проте ключова відмінність криється в ініціалізації та роботі з пам'яттю. Константи phi та resphi розраховуються один раз і виносяться за межі циклу. Це дає змогу уникнути багаторазового виклику ресурсоємної математичної функції Math.Sqrt (5.0) на кожній ітерації. Логіка умовних операторів if-else всередині циклу наочно демонструє теоретичну перевагу методу: завдяки властивостям пропорції алгоритм просто переприсвоює одне з раніше знайдених значень (наприклад, $f2 = f1$), звертаючись до цільової функції func лише один раз на кожному новому кроці.

Отримані внаслідок програмного тестування експериментальні дані щодо кількості ітерацій, викликів цільової функції та часу апаратного виконання зведено у табл. 1.

Таблиця 1

Зведена таблиця порівняння методів дихотомії та золотого перетину

Цільова функція	Початковий інтервал	Метод пошуку	Кількість ітерацій	Кількість викликів $f(x)$	Час виконання, мс
$f(x) = (x-2)^2 + 3$	[0,5]	Дихотомія	20	40	0,042700
		Золотий перетин	28	30	0,064500
$f(x) = e^x - 4x$	[0,3]	Дихотомія	20	40	0,075100
		Золотий перетин	27	29	0,048500
$f(x) = x^2 + 2\cos(x)$	[–2,2]	Дихотомія	20	40	0,040100
		Золотий перетин	27	29	0,037900

Аналіз експериментальних даних виявляє важливу закономірність щодо часу виконання алгоритмів: для простих поліноміальних функцій метод дихотомії виконується швидше, оскільки операція ділення інтервалу навпіл є менш ресурсоємною для процесора, ніж множення на пропорцію золотого перетину. З погляду архітектури електронно-обчислювальної машини (ЕОМ), обчислення поліномів зводиться до базових операцій додавання та множення. Вони виконуються арифметико-логічним пристроєм процесора за мінімальну кількість мікротактів. У таких умовах «вартість» обчислення самої цільової функції є настільки мізерною, що на перший план виходить ефективність розрахунку нових меж інтервалу, де дихотомія має перевагу.

Однак ситуація кардинально змінюється під час роботи з трансцендентними та комбінованими функціями, де обчислення самого значення функції потребує набагато більше часу. Тригонометричні функції або експонента не обчислюються процесором за одну дію. Для їх

знаходження математичний співпроцесор використовує складні мікропрограми (наприклад, розкладання в ряди Тейлора або алгоритми CORDIC), що потребує в десятки разів більше тактів процесора на кожен виклик.

Саме тут розкривається головна архітектурна перевага методу золотого перетину. Економія обчислювальних ресурсів завдяки зменшенню кількості викликів «важкої» цільової функції (наприклад, 29 викликів проти 40 у дихотомії) повністю компенсує дещо повільніше математичне звуження інтервалу. Отже, науково обґрунтовано, що для складних алгоритмічних задач та інженерного моделювання метод золотого перетину є критично необхідним інструментом оптимізації.

Висновки. Проведений порівняльний аналіз методів одновимірної оптимізації підтверджує, що вибір оптимального алгоритму безпосередньо залежить від математичної природи цільової функції та апаратної вартості її обчислення. Експериментальне тестування, реалізоване мовою C#, дало змогу оцінити ефективність методів не лише за теоретичною швидкістю звуження інтервалу, але й за реальними показниками навантаження на процесор.

Встановлено, що метод бісекції (дихотомії) є доцільним для застосування у випадках оптимізації математично простих виразів (зокрема поліноміальних). У таких задачах алгоритм забезпечує швидке математичне ділення відрізка, а висока швидкість проходження ітераційного циклу повністю нівелює апаратні витрати на багаторазове обчислення самої функції.

Натомість метод золотого перетину довів свою беззаперечну обчислювальну ефективність під час роботи з трансцендентними та комбінованими функціями. Завдяки використанню специфічної пропорції цей алгоритм дає змогу перевикористовувати раніше обчислені значення, зводячи до мінімуму кількість звернень до «важкої» цільової функції. Для складних алгоритмічних задач така економія ресурсів є критично важливою, оскільки вона суттєво зменшує загальний час виконання програми.

Abstract. This article presents a comparative analysis of one-dimensional optimization methods, specifically the bisection (dichotomy) method and the golden section method. The study is based on a comparison of the convergence rates of the algorithms, the number of required computations, and their overall efficiency for various types of objective functions. The practical implementation and software testing of the mathematical methods under consideration were performed using the C# language. Based on the results obtained, a summary comparison table was created, which clearly demonstrates the advantages, disadvantages, and optimal conditions for applying each of the approaches.

Keywords: one-dimensional optimization, bisection method, golden section method, convergence rate, search algorithms, C#.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Чисельні методи: навчальний посібник / Л. О. Волонтир, О. В. Зелінська, Н. А. Потапова, І. А. Чіков. Вінниця: ВНАУ, 2020. 322 с. URL: https://r.donnu.edu.ua/handle/123456789/1805

2. Smit A. Program for Bisection Method. *geeksforgeeks.org.* 2022. URL: https://www.geeksforgeeks.org/program-for-bisection-method/ (дата звернення: 09.03.2026).

3. Proposal of the dichotomous STATIS DUAL method: software and application for the analysis of dichotomous data, applied to the test of learning styles in university students / V. I. Ballesteros-Espinoza та ін. *Mathematics*. 2021. Т. 9. № 21. С. 2797. URL: https://doi.org/10.3390/math9212797 (дата звернення: 09.03.2026).

4. Афанасьєва Д. С. Порівняння ефективності методів дихотомії та хорд у розв'язанні нелінійних рівнянь. Прикладні інформаційні технології. 2025. URL: https://jait.donnu.edu.ua/article/view/16986

5. Синчук Д. Метод золотого перетину для розв'язування завдань одновимірної оптимізації. *Науковий простір: актуальні питання, досягнення та інновації:* Міжнар. науково-практ. конф. (м. Вінниця, 23–24 листоп. 2021 р.). Вінниця, 2021. С. 40–41.