НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Курсова робота

із дисципліни «Методи оптимізації»

на тему: «Метод найшвидшого спуску»

|  |  |
| --- | --- |
| Студента групи КМ-03  Передерея Б. О. | Керівник:  Старший викладач Ладогубець Т. С.  Кількість балів:\_\_\_\_\_\_\_ |
|  | Оцінка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Київ – 2022

**Зміст**

[**Постановка задачі** 2](#_Toc136535217)

[**Теоретична частина** 2](#_Toc136535218)

[**Основна частина** 4](#_Toc136535219)

[**Безумовна оптимізація** 4](#_Toc136535220)

[**Умовна оптимізація** 17](#_Toc136535221)

[**Оптимізація при опуклій допустимій області** 17](#_Toc136535222)

[**Оптимізація при випуклій допустимій області** 18](#_Toc136535223)

[**Висновки** 22](#_Toc136535224)

[**Список використаної літератури** 24](#_Toc136535225)

[**Додаток 1 (Код програми)** 25](#_Toc136535226)

# **Постановка задачі**

Дослідити збіжність методу найшвидшого спуску при мінімізації функції Розенброка в залежності від:

1. Величини кроку h при обчисленні похідних.
2. Схеми обчислення похідних.
3. Способу обчислення кроку: постійний, оптимальний.
4. Виду методу одновимірного пошуку (ДСК-Пауелла або Золотого перетину).
5. Точності методу одновимірного пошуку.
6. Значення параметру в алгоритмі Свена.
7. Вигляду критерію закінчення. .
8. Наявності модифікацій (методи Бута, Люстерніка, важкої кульки).

Використати метод штрафних функцій (метод зовнішньої точки) для умовної оптимізації при розташування локального мінімума поза випуклої допустимої області.

# **Теоретична частина**

Один з найпростіших методів першого порядку оптимізації функцій - метод найшвидшого спуску. Цей метод базується на знаходженні мінімуму функції шляхом здійснення кроків у напрямку, протилежному градієнту функції, тобто у напрямку найшвидшого спуску, осклільки від’ємний градієнт у точці направлений у строну найбільшого зменшення по всім значенням і він є ортогональним лінії рівня у точці .

Алгоритм методу найшвидшого спуску можна описати наступним чином. Спочатку задавши початкову точка x0, проводиться ітераційний процес, на кожному кроці якого виконується наступне:

1. Обчислюється градієнт функції в точці :
2. Знаходиться напрямок спуску, який дорівнює протилежному градієнту з нормуванням:

при сталому кроці або при оптимальному

1. Виконується визначення кроку , який мінімізує функцію (він може бути як сталим, так і оптимальним).
2. Обчислюється нова точка як:
3. Якщо задана точність не досягнута, повторюється ітераційний процес.

Від’єдним градієнт дає лише направлення оптимізації, але не велечину кроку. При цьому можна використовувати різні методи та алгоритми отримання оптимального (різні методи одновимірного пошуку для мінімізації цільової функції від ) та сталого значення кроку (наприклад, алгоритм Adam).

Хоча метод є простим у реалізації, він дуже низьку швидкість збіжності, що обгрунтовується як теоретичною частиною методу, так і практичними дослідами, тому на практиці для мінімазції цільових функцій він використовується дуже рідко. Зачасту метод найшвидого спуску використовують у випадках, коли інші методи використати неможливо, або коли функція має дуже велику кількість параметрів (наприклад як у нейронних мережах), оскільки оновлення самих параметрів функції здійснюється поступово.

Для пришвидшення збіжності методу можуть використовуватися його модифікації, такі як метод Бута, Люстерніка та важкої кульки. Метод Бута використовується у випадку оптимального кроку для стискання сили скачків між ярами функції, за допомогою домноження кожного оптимального кроку на певний коєфіцієнт, менший за 1. Метод Люстерніка використовується у випадку константого кроку і він потрібен для задання певного відскоку функції у випадку заповільнення її ходу ().

Метод найшвидшого спуску часто використовується як частину інших методів оптимізації, наприклад, у методі Флетчера-Рівса або в інших методах в якості першого кроку.

Щодо умовної оптимізації, у цій курсовій роботі буде розглянуто метод штрафних функцій. Цей метод може бути представленим у різних варіаціях, які, однак, мають одну спільну рису – у всіх цих методах задача нелійного програмування перетворюється або на одну (еквівалентну початковій) задачу без обмежень, або в еквіваленту послідовність задач без обмежень. Перевага, яку ми отримуємо за рахунок переходу від задачі мінімізації за наявності обмежень до завдання мінімізації у відсутність обмежень, у тому, що в останньому випадку мінімізація може здійснюватися за допомогою набагато простіших (порівняно з першим випадком) алгоритмів (у цій курсовій роботі – методом найшвидшого спуску). При використанні методів штрафних функцій виходить максимальний оптимізаційний ефект за рахунок постійного компромісу між необхідністю задоволення обмежень та процесом мінімізації функції, який досягається шляхом присвоєння належних вагів функціям, що задають обмеження.

Одним з параметричних методів штрафних функцій є метод зовнішньої точки. Цей метод генерує послідовність точок, які виходять за межі допустимої області, аде дають рішення у допустимій області. Сама штрафна функція не дозволяє вектору x занадто сильно відійти від границі допустимої області, оскільки у цьому методі використовуються квадратичні штрафи з певним коефіцієнтом R у випадку, якщо обмеження не виконується, і прирівнює квадратичний штраф до нуля, якщо воно виконується.

Підсумовючи, основою методів штрафних функцій в області нелінійного програмування покладена ідея перетворення загальної нелінійної задачі у послідовність задач без обмежень шляхом додавання до цільової функції однієї чи декількох функцій, що задають обмеження, з тим, щоб обмеження, як такі, у задачі оптимізації не фігурували.

# **Основна частина**

## **Безумовна оптимізація**

*Вплив величини кроку h при обчисленні похідних*

Початкові умови:

Початкова точка: (-1.2, 0)

Критерій закінчення:

Величина похибки: 0.001

МОП: Золотий переріз

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена: 0.01

Схема похідної: центральна

Дельта лямбда у Свені: 0.01 \*

Результати:

|  |  |  |  |
| --- | --- | --- | --- |
| Величина кроку h | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
| 0.1 | [0.32693 0.10388] | 0.45392 | 843 |
| 0.01 | [0.76475 0.58438] | 0.05536 | 3166 |
| 0.001 | [0.81534 0.66392] | 0.03417 | 2408 |
|  | [0.76806 0.58935] | 0.05383 | 2751 |
|  | [0.83631 0.69889] | 0.02682 | 1640 |
|  | [0.76914 0.59021] | 0.05348 | 2239 |
|  | [0.92755 0.85999] | 0.00526 | 1699 |
|  | [0.78929 0.6225 ] | 0.04442 | 2558 |
|  | [0.77247 0.59617] | 0.0518 | 2219 |
|  | [0.8144 0.66232] | 0.03453 | 2588 |
|  | [0.77646 0.60246] | 0.04999 | 3138 |
|  | [0.8317 0.69126] | 0.02835 | 2587 |
|  | [0.78355 0.6136 ] | 0.04686 | 1730 |
|  | [0.80381 0.64559] | 0.03852 | 849 |
|  | [0.9663 0.93346] | 0.00114 | 1591 |

Величина кроку впливала на результат нелінійно, тобто зменшення величини кроку не гарантували підвищення точності. З наведеної таблиці найкраща себе показали , яке підходить більше для зменшення кількості обчислень функції, але з трохи гіршими результатами, або , яка має на 742 обчислень більше, ніж , але це h дало у 30 раз більшу точність.

*Вплив схем обчислення похідних*

Початкові умови:

Початкова точка: (-1.2, 0)

Критерій закінчення:

Величина похибки: 0.001

МОП: Золотий переріз

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена: 0.01

Величина кроку у похідних: h =

Дельта лямбда у Свені: 0.01 \*

Результати:

|  |  |  |  |
| --- | --- | --- | --- |
| Схема похідних | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
| Центральна | [0.9663 0.93346] | 0.00114 | 1591 |
| Лівостороння | [0.81311 0.66078] | 0.03494 | 1417 |
| Правостороння | [0.90273 0.81438] | 0.00949 | 1513 |

Изображение выглядит как линия, диаграмма, График, оригами

Автоматически созданное описание

*Графік кількості обчислень функції залежно від значення h та схеми похідних*

Изображение выглядит как снимок экрана, линия, График, диаграмма

Автоматически созданное описание

*Графік значення функції залежно від значення h та схеми похідних*

Згідно графіку кількості обчислень функцію найкраще себе показала правостороння похідна при точності з кількістю обчислень функції 507, але з графіку значень отримане мінімальне значення функції є дорівнює 0.4086. Найкращі з отриманих результатів:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Похідна | h | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
| Лівостороння |  | [0.90474 0.81790] | 0.009117 | 1165 |
| Правостороння |  | [0.95204 0.90578] | 0.00234 | 1380 |
| Центральна |  | [0.96630 0.93346] | 0.00114 | 1591 |

На основі наведеної таблиці можна обрати потрібний результат у залежності від того, що нам важливіше – кількість обчислень або точність отриманих результатів. Хоча між центральною та лівосторонньою границею різниця у кількості обчислень функції 426, точність центральної схеми більше у 8 разів.

*Способу обчислення кроку: постійний, оптимальний.*

Початкова точка: (-1.2, 0)

Критерій закінчення:

Величина похибки: 0.0001

МОП: Золотий переріз

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена: 1

Величина кроку у похідних: h =

Схема похідних: Центральна

Дельта лямбда у Свені: 1 \*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Крок | Зменшення лямбда при збільшені функції | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | У 2 рази | [0.99917 0.99834] | 0.0 | 33104 |
|  | У 2 рази | [0.99019 0.98047] | 0.0001 | 15526 |
|  | У 2 рази | [1.00144 1.00282] | 0.0 | 55 |

Найкраще себе показала , але цей результат є дуже унікальним, оскільки результати дуже залежать від інших параметрів методу та від початкової точки. Наприклад, почнемо з точки (-1.1, 0) з тими самими параметрами алгоритму. Результати:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Крок | Зменшення лямбда при збільшені функції | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | У 2 рази | [0.98224 0.96476] | 0.00032 | 14075 |
|  | У 2 рази | [0.99019 0.98046] | 0.0001 | 14546 |
|  | У 2 рази | [1.00063 1.00126] | 0.0 | 25775 |

Тобто підібрані параметри для 55 обчислень при дуже сильно залежать від початкової точки і не зберігають своїх властивостей при найменшій зміні. Щодо роботи лямбди оптимальної:

Початкова точка: (-1.2, 0)

|  |  |  |  |
| --- | --- | --- | --- |
| Крок | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
| МОП (золотий переріз) | [0.96776 0.93647] | 0.00104 | 100 |
| МОП (дск) | [0.95041 0.90334] | 0.00246 | 100 |

Початкова точка: (-1.1, 0)

|  |  |  |  |
| --- | --- | --- | --- |
| Крок | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
| МОП (золотий переріз) | [1.07562 1.15734] | 0.00573 | 7863 |
| МОП (дск) | [0.96295 0.9308 ] | 0.00261 | 84 |

Як виявилося, оптимальний крок є більш стійким до зміни початкових параметрів і робить як мінімум у два рази менше обчислень цільової функції, аніж константний крок. Це можна пояснити тим, що фіксований крок не використовує жодної інформації про саму функцію, окрім факту зменшення функції у новій точці. Можливо, результати були би кращими, якби було використано отримання кроку за допомогою інформацію про значення похідних чи функцій на минулих кроках (наприклад алгоритм Adam, який використовує інформацію про значення функції на минулих кроках і є популярним алгоритмом у нейронних мережах), але у цій роботі інші (більш кращі методи) для отримання константної лямбди досліджуватися не будуть.

*Вплив виду методу одновимірного пошуку та точності методу одновимірного пошуку*

Початкова точка: (-1.2, 0)

Критерій закінчення:

Величина похибки: 0.001

Величина параметру в алгоритмі Свена: 0.01

Величина кроку у похідних: h =

Дельта лямбда у Свені: 0.01 \*

Результати:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод МОП | Точність МОП | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
| Золотий перетин | 1 | [-0.5701 0.33418] | 2.47361623 | 35 |
| Золотий перетин |  | [-0.5701 0.33418] | 2.47361623 | 35 |
| Золотий перетин |  | [ 0.4266 0.18053] | 0.32900201 | 829 |
| Золотий перетин |  | [ 0.77247 0.59617] | 0.05179839 | 2219 |
| Золотий перетин |  | [ 0.97698 0.95444] | 0.00053006 | 7898 |
| Золотий перетин |  | [ 0.99979 0.99958] | 4e-08 | 14138 |
| Золотий перетин |  | [ 0.99939 0.99878] | 3.7e-07 | 270146 |
| Золотий перетин |  | [ 0.99994 0.99988] | 0.0 | 463535 |
| Золотий перетин |  | [ 0.99998 0.99997] | 0.0 | 600053 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод МОП | Точність МОП | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
| ДСК Пауелла | 1 | [-0.51361 0.26943] | 2.29419939 | 40 |
| ДСК Пауелла |  | [-0.51361 0.26943] | 2.29419939 | 40 |
| ДСК Пауелла |  | [ 0.67567 0.45514] | 0.10538051 | 482 |
| ДСК Пауелла |  | [ 0.91256 0.83244] | 0.00765649 | 3304 |
| ДСК Пауелла |  | [ 0.99458 0.98917] | 2.941e-05 | 4248 |
| ДСК Пауелла |  | [ 0.99914 0.99829] | 7.3e-07 | 21956 |
| ДСК Пауелла |  | [ 0.9609 0.92327] | 0.00152913 | 28523 |
| ДСК Пауелла |  | [ 0.99994 0.99988] | 0.0 | 166113 |
| ДСК Пауелла |  | [ 0.99999 0.99999] | 0.0 | 216985 |

Що метод золотого перетину, що метод ДСК Пауелла, хоч і дали майже повністю мінімізували цільову функцію, але виконували велику кількість кроків. При точності від до ДСК Пауелла було у 1.5-2 рази гірше за метод золотого перерізу згідно кількості обчислень функції, але при більшій величині похибки методу найшвидшого спуску (окрім ) ДСК Пауелла виконувало у 1.5-3 рази менше обчислень цільової функції при похибці.

*Значення параметру в алгоритмі Свена*

Початкова точка: (-1.2, 0)

Критерій закінчення:

МОП: Золотий переріз

Величина похибки МОП: 0.001

Величина похибки: 0.001

Величина кроку у похідних: h =

Схема похідної: центральна

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр Свена | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
| 10 | [0.96943 0.93934] | 0.00096 | 135 |
| 1 | [0.9682 0.93747] | 0.00101 | 95 |
| 0.1 | [0.86142 0.74089] | 0.01934 | 1433 |
| 0.01 | [0.9663 0.93346] | 0.00114 | 1591 |
| 0.001 | [0.80349 0.64522] | 0.03863 | 1019 |
|  | [0.81851 0.66870] | 0.0331 | 2585 |
|  | [0.79936 0.63767] | 0.04043 | 3595 |
|  | [0.86143 0.7412 ] | 0.01927 | 3552 |
|  | [0.99189 0.98379] | 7e-05 | 2543 |
|  | [0.78858 0.62151] | 0.04471 | 2386 |
|  | [0.77267 0.59568] | 0.05186 | 2985 |
|  | [0.88787 0.78798] | 0.01258 | 164 |
|  | [0.94176 0.88671] | 0.0034 | 2360 |
|  | [0.82962 0.68779] | 0.02905 | 5008 |
|  | [0.77881 0.60602] | 0.04895 | 4336 |
|  | [0.78053 0.60765] | 0.04841 | 4042 |
|  | [0.76276 0.58028] | 0.05651 | 9157 |

Закономірність між зменшенням параметру в алгоритмі Свена та кількістю обчислень цільової функції немає лінійної залежності. Найкраще себе показали значення 1 та . Що цікаво – при значенню 1 через дуже великий крок у Свені, сам алгоритм дуже часто повертав інтервал [-1; 1], підраховуючи цільову функцію лише два рази (в обох точках функція зростала). Значення у Свені заставляло видавати значення з більшою оптимізацією. Але швидкі тести показали, що збільшення загальної точності МНС дуже швидко нівелює ефект значення параметру Свена. При параметру у Свені функція виконує 8030 обчислень функції. Але значення потрібно дослідити, варіюючи інші гіперпараметри

*Наявності модифікацій (методи Бута, Люстерніка)*

Початкова точка: (-1.2, 0)

Критерій закінчення:

Величина похибки: від до

МОП: Золотий переріз

Величина похибки МОП: 0.0001

Величина параметру в алгоритмі Свена: 1

Величина кроку у похідних: h =

Схема похідних: Центральна

Дельта лямбда у Свені: 1 \*

Без модифікації (оптимальний крок):

|  |  |  |  |
| --- | --- | --- | --- |
| Величина похибки | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | [0.944811 0.892425] | 0.0030517290534505083 | 59 |
|  | [0.944811 0.892425] | 0.0030517290534505083 | 59 |
|  | [0.944811 0.892425] | 0.0030517290534505083 | 59 |
|  | [0.9549 0.911692] | 0.0020360036153700685 | 2992 |
|  | [0.997625 0.995244] | 5.654671001116072e-06 | 11150 |
|  | [1.000018 1.000036] | 3.187467096938542e-10 | 71341 |
|  | [1.000017 1.000035] | 2.9935832718279436e-10 | 71446 |
|  | [0.999996 0.999992] | 1.653632936741718e-11 | 186229 |
|  | [1. 1.] | 4.845172548228179e-15 | 352854 |
|  | [1. 1.] | 3.454693824482231e-17 | 354345 |

Метод Бута

|  |  |  |  |
| --- | --- | --- | --- |
| Величина похибки | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | [0.848868 0.722703] | 0.023292838671245996 | 73 |
|  | [0.850066 0.722198] | 0.022497284225225064 | 104 |
|  | [0.850712 0.722537] | 0.022424769012345636 | 139 |
|  | [0.960754 0.922862] | 0.0015437126434409516 | 2653 |
|  | [0.993329 0.986682] | 4.454622335755933e-05 | 25126 |
|  | [1.000229 1.000458] | 5.238276656513157e-08 | 36056 |
|  | [0.999976 0.999952] | 5.8495937052521e-10 | 127090 |
|  | [0.999995 0.99999 ] | 2.3596883061668478e-11 | 128483 |

Без модифікації (сталий крок)

|  |  |  |  |
| --- | --- | --- | --- |
| Величина похибки | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | [1.000966 1.003067] | 0.000129444104671098 | 36 |
|  | [1.000966 1.003067] | 0.000129444104671098 | 36 |
|  | [1.000966 1.003067] | 0.000129444104671098 | 36 |
|  | [1.001442 1.002824] | 2.4594578419536843e-06 | 55 |
|  | [1.001415 1.002834] | 2.0023628766815063e-06 | 68 |
|  | [1.000002 1.000005] | 5.8375053802681465e-12 | 37763 |
|  | [1.000002 1.000005] | 5.8375053802681465e-12 | 37763 |
|  | [1.000002 1.000005] | 5.8375053802681465e-12 | 37763 |

Метод Люстерніка

Оскільки використання у разі уповільнення функції відкидало дуже далеко задану точку, було прийнято рішення замість нього використовувати константний крок помножений на 2 у разі уповільнення .

|  |  |  |  |
| --- | --- | --- | --- |
| Величина похибки | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | [1.00096636 1.00306727] | 0.000129444104671098 | 36 |
|  | [1.00096636 1.00306727] | 0.000129444104671098 | 36 |
|  | [1.00096636 1.00306727] | 0.000129444104671098 | 36 |
|  | [1.00144187 1.00282414] | 2.4594578419536843e-06 | 55 |
|  | [1.00141482 1.00283418] | 2.0023628766815063e-06 | 68 |
|  | [1.00141024 1.00283288] | 1.999604498792486e-06 | 79 |
|  | [1.00141024 1.00283288] | 1.999604498792486e-06 | 79 |
|  | [1.00018288 1.00036688] | 3.356232248364021e-08 | 25430 |

Метод Бута спрацював гірше на низьких точностях до включно, але на точностях - метод Бута виконав у 3 рази менше обчислень цільової функції, що говорить про доцільність його використання у випадку потреби великої точності.

Щодо методу Люстерніка – він спрацьовував лише у випадках великої точності (тобто коли почало прямувати до нуля) і справді зміг пришвидшити збіжність з 37663 обчислень функції до 79, але на точності кількість обчислень функції була меншою лише у 1.5 рази.

*Вигляд критерію закінчення та фінальний підбір параметрів*

На основі проведеного дослідження були виявлено два найкращих параметри для алгоритму. Перший з них:

Критерій закінчення МНС:

Величина похибки МНС: 0.001

МОП: Золотий переріз

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена: 1

Схема похідної: права

Величина кроку h:

|  |  |  |  |
| --- | --- | --- | --- |
| Величина похибки | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | [0.96677204 0.93700294] | 0.0016585903258537 | 36 |
|  | [0.96766118 0.93649704] | 0.0010474608333755 | 61 |
|  | [0.96766118 0.93649704] | 0.0010474608333755 | 61 |
|  | [0.96775585 0.93647002] | 0.0010403471054342 | 90 |
|  | [0.99888499 0.99776471] | 1.2474843234e-06 | 1109 |
|  | [0.99973041 0.99946033] | 7.27101884e-08 | 5270 |
|  | [0.99998729 0.99997445] | 1.632922e-10 | 107061 |

Але, через велике значення параметру в алгоритмі Свена, цей набір параметрів гарно працює лише на точності до і при збільшені точності перестає працювати взагалі або видає дуже велику кількість підрахунків функції. Тому його буде використано для умовної оптимізації. Також на основі проведеного дослідження було підібрано універсальні значення алгоритму, які використовували більше обчислень функції, але давали точний результат при будь-якій точності. А саме якщо важлива кількість розрахунків цільової функції:

Початкова точка: (-1.2, 0)

Критерій закінчення:

МОП: ДСК Пауелла

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена:

Величина кроку у похідних: h =

Схема похідних: Ліва

|  |  |  |  |
| --- | --- | --- | --- |
| Величина похибки | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | [0.941343128822 0.885875223238] | 0.0034469619600370595 | 97 |
|  | [0.941343128822 0.885875223238] | 0.0034469619600370595 | 97 |
|  | [0.941343128822 0.885875223238] | 0.0034469619600370595 | 97 |
|  | [0.998025684823 0.996047300038] | 3.904268568326625e-06 | 891 |
|  | [0.999937452577 0.999874330687] | 3.945632329986833e-09 | 980 |
|  | [0.999937452577 0.999874330687] | 3.945632329986833e-09 | 980 |
|  | [0.999945583744 0.999890852539] | 2.9712356023272747e-09 | 1186 |
|  | [0.999995118259 0.999990207384] | 2.3916411964537352e-11 | 1888 |
|  | [0.999995119284 0.999990207809] | 2.391614698230838e-11 | 1902 |
|  | [0.99999511952 0.999990207911] | 2.3916136988818482e-11 | 1924 |
|  | [0.999995119528 0.999990207914] | 2.3916136974470714e-11 | 1934 |

Якщо важлива точність:

Початкова точка: (-1.2, 0)

Критерій закінчення:

МОП: ДСК Пауелла

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена:

Величина кроку у похідних: h =

Схема похідних: Ліва

|  |  |  |  |
| --- | --- | --- | --- |
| Величина похибки | Точка мінімуму, до якої прийшов алгоритм | Значення у точці мінімуму | Кількість обчислень функції |
|  | [0.9413391311104482 0.8858673090700308] | 0.003447450493897172 | 97 |
|  | [0.9950807362047298 0.990166122541372 ] | 2.4237372722857764e-05 | 474 |
|  | [0.9999371498735872 0.9998740529303117] | 3.95642679899016e-09 | 958 |
|  | [0.9999891841845501 0.9999783253474618] | 1.1716795790243104e-10 | 1469 |
|  | [0.9999909626474174 0.999981889007576 ] | 8.180601162792882e-11 | 1882 |
|  | [0.9999999980862093 0.9999999961647991] | 3.668400559736167e-18 | 2294 |
|  | [0.9999999599659612 0.9999999197710706] | 1.6053116416662584e-15 | 2615 |
|  | [0.9999999967684055 0.9999999935239561] | 1.0459727980622604e-17 | 2917 |
|  | [0.9999999996936957 0.9999999993861611] | 9.397368212432901e-20 | 3448 |
|  | [0.9999999999401763 0.9999999998801123] | 3.584647381882577e-21 | 1212 |
|  | [1.0000000000000038 1.0000000000000075] | 1.4248800100554526e-29 | 1173 |

## **Умовна оптимізація**

### **Оптимізація при опуклій допустимій області**

Задача умовної оптимізації:

За умов

Результати, отримані за допомогою WolframAlpha:

Изображение выглядит как текст, снимок экрана, фрукт

Автоматически созданное описание

Початкові умови (метод штрафних функцій, зовнішня точка):

Початкова точка: (-1.2, 0)

Точність УО: 0.0001

Критерій закінчення УО:

Критерій закінчення МНС:

Величина похибки МНС: 0.001

МОП: Золотий переріз

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена: 1

Схема похідної: права

Величина кроку h:

Початкова R = 1, зміна у 10 разів з кожною новою задачею БО

|  |  |  |  |
| --- | --- | --- | --- |
| Початкова точка | Точка мінімуму | Значення у точці мінімуму | Кількість обчислень функції |
| [-1.2 0. ] | [0.82081738 0.67363354] | 0.04836977 | 517 |
| [0.82081738 0.67363354] | [0.78828466 0.62066965] | 0.04531438 | 709 |
| [0.78828466 0.62066965] | [0.78655488 0.61818205] | 0.04564934 | 761 |
| [0.78655488 0.61818205] | [0.78642212 0.61771124] | 0.04567226 | 788 |
| [0.78642212 0.61771124] | [0.78639000 0.61760427] | 0.04569403 | 818 |
| [0.78639000 0.61760427] | [0.78641999 0.61766631] | 0.04567885 | 850 |

За 850 обрахунків цільової функції метод штрафних зміг досягти точності заданої точності. Різниця між результатами WolframAlpha та методом штрафних функцій 4.05\*, що говорить про коректність реалізації.

Изображение выглядит как снимок экрана, Красочность, Графика, графический дизайн

Автоматически созданное описание

*Графік роботи методу штрафних функцій для опуклої області*

### **Оптимізація при випуклій допустимій області**

Задача умовної оптимізації:

За умов

Результати, отримані за допомогою WolframAlpha:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Початкова точка: (-1.2, 0)

Точність УО: 0.0001

Критерій закінчення УО:

Критерій закінчення МНС:

Величина похибки МНС: 0.001

МОП: Золотий переріз

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена: 1

Схема похідної: права

Величина кроку h:

Початкова R1/R2 = 1, збільшення у 10 разів з кожною новою задачею БО

|  |  |  |  |
| --- | --- | --- | --- |
| Початкова точка | Точка мінімуму | Значення у точці мінімуму | Кількість обчислень функції |
| [-1.2 0. ] | [0.96897402 0.93870922] | 0.06112935 | 91 |
| [0.96897402 0.93870922] | [0.76043668 0.57714217] | 0.05942101 | 446 |
| [0.76043668 0.57714217] | [0.75350671 0.5665858 ] | 0.06109383 | 590 |
| [0.75350671 0.5665858 ] | [0.75278775 0.56554267] | 0.06126307 | 700 |
| [0.75278775 0.56554267] | [0.75272864 0.56541217] | 0.06128432 | 832 |
| [0.75272864 0.56541217] | [0.75268574 0.56537883] | 0.06129821 | 864 |

*Изображение выглядит как снимок экрана, Красочность, Графика, круг

Автоматически созданное описание*

*Графік роботи методу штрафних функцій для випуклої області*

Через доволі великі штрафи алгоритм пішов у протилежну сторону від мінімуму і дав хибний мінімум задачі УО, який більший 0.021 від справжнього. Зменшення штрафу дало правильні результати.

Початкова R1/R2 = 0.4, збільшення у 4 рази з кожною новою задачею БО.

|  |  |  |  |
| --- | --- | --- | --- |
| Початкова точка | Точка мінімуму | Значення у точці мінімуму | Кількість обчислень функції |
| [-1.2 0. ] | [0.96822006 0.9373234 ] | 0.02503366 | 120 |
| [0.96822006 0.9373234 ] | [1.18578778 1.40675365] | 0.03856629 | 1258 |
| [1.18578778 1.40675365] | [1.19920024 1.43881063] | 0.04175363 | 1343 |
| [1.19920024 1.43881063] | [1.20462296 1.45188296] | 0.04232494 | 1430 |
| [1.20462296 1.45188296] | [1.20585162 1.45472471] | 0.04249078 | 1491 |
| [1.20585162 1.45472471] | [1.20609843 1.45536388] | 0.04253568 | 1553 |
| [1.20609843 1.45536388] | [1.2061053 1.45543542] | 0.04255108 | 1586 |

Изображение выглядит как снимок экрана, Красочность, Графика, графический дизайн

Автоматически созданное описание

*Графік роботи методу штрафних функцій для випуклої області*

Різниця між значеннями функції, яку надав WolframAlpha, та програмою приблизно дорівнює , що говорить про коректність реалізації.

# **Висновки**

У результаті курсової роботи було проведено дослідження збіжності методу найшвидшого спуску при мінімізації функції Розенброка у залежності від параметрів, зазначених у постановці задачі. У результаті підбору параметрів найбільш універсальним набором було визначено наступні умови:

Початкова точка: (-1.2, 0)

Критерій закінчення:

Крок: оптимальний

МОП: ДСК Пауелла

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена:

Величина кроку у похідних: h =

Схема похідних: Ліва

Графіки роботи методу залежності від заданої точності:

Изображение выглядит как снимок экрана, Прямоугольник, линия, дизайн

Автоматически созданное описание

*Графік кількості підрахунків функції відносно точності методу МНС*

*Изображение выглядит как снимок экрана, Прямоугольник, линия, прямоугольный

Автоматически созданное описание*

*Графік значення функції залежно від точності методу*

З числовими результатами роботи методу найшвидшого спуску при заданих параметрах можна ознайомитися у розділі безумовної оптимізації, вигляд критерію закінчення та фінальний підбір параметрів.

Щодо умовної оптимізації, вона була проведена як на опуклій, так і випуклій допустимій області. У якості методу УО було використано метод штрафних функцій для зовнішньої точки. Оскільки під час УО висока точність роботи методу не потребувалася, а потрібно було саме кількість обрахунків функції, то було використано наступні параметри, які давали меншу точність, але низьку кількість підрахунків цільової функції:

Початкова точка: (-1.2, 0)

Критерій закінчення МНС:

Величина похибки МНС: 0.001

МОП: Золотий переріз

Величина похибки МОП: 0.001

Величина параметру в алгоритмі Свена: 1

Схема похідної: права

Величина кроку h:

В якості опуклої допустимої області було використано обмеження . Згідно зазначених результатів в основній частині, різниця між справжнім мінімумом задачі УО та отриманим алгоритмом, реалізованим у цій курсовій роботі, склала 4.05\*, що вказує на коректність реалізації.

Умовна оптимізація випуклої області отримала іще одну умову, а саме , що потребувала зміни росту штрафів у штрафній функції, оскільки штрафи з задачі, де була присутня опукла область, хоч і оптимізовували функцію у випуклій допустимій області та виходили на її межі, але, на жаль, у результаті давали несправжній мінімум. При зменшенні початкового штрафу з 1 до 0.4 і росту цього штрафу від 10 разів до 4 було отримано справжній мінімум задачі УО, а різниця значення функції в отриманій точці відрізнялася на , що також може бути свідчення про правильність програмної реалізації.

# **Список використаної літератури**

1. Himmelblau D. M. Applied nonlinear programming. New York : McGraw-Hill, 1972. 498 p. (page 72-83, 333-337)
2. C M. J. Steepest Descent. OSTI.GOV | U.S. Department of Energy Office of Scientific and Technical Information. URL: https://www.osti.gov/servlets/purl/983240 (date of access: 29.05.2023).
3. The Method of Steepest Ascent (Descent). Department of Mathematical Sciences | Montana State University. URL: https://math.montana.edu/jobo/st578/sec6.pdf (дата звернення: 29.05.2023).

# **Додаток 1 (Код програми)**

Код разом зі звітом можна знайти за посиланням: <https://github.com/B-Perederei/Method-of-steepest-descent>

#!/usr/bin/env python

# coding: utf-8

# In[ ]:

import numpy as np

import math

import matplotlib.pyplot as plt

np.set\_printoptions(precision=128)

# In[ ]:

def fderright(x, fx, h):

    gradFx1 = (f(x + h\*np.array([1, 0])) - fx) / h

    gradFx2 = (f(x + h\*np.array([0, 1])) - fx) / h

    return np.array([gradFx1, gradFx2])

def fderleft(x, fx, h):

    gradFx1 = (fx - f(x - h\*np.array([1, 0]))) / h

    gradFx2 = (fx - f(x - h\*np.array([0, 1]))) / h

    return np.array([gradFx1, gradFx2])

def fdercenter(x, h):

    gradFx1 = (f(x + h\*np.array([1, 0])) - f(x - h\*np.array([1, 0]))) / (2\*h)

    gradFx2 = (f(x + h\*np.array([0, 1])) - f(x - h\*np.array([0, 1]))) / (2\*h)

    return np.array([gradFx1, gradFx2])

def fder(x, fx, h, type):

    if type == 'right':

        return fderright(x, fx, h)

    if type == 'left':

        return fderleft(x, fx, h)

    if type == 'center':

        return fdercenter(x, h)

    else:

        raise Exception('Not valid type')

# In[ ]:

def norm(x):

    x1 = x[0]

    x2 = x[1]

    return math.sqrt(x1\*\*2 + x2\*\*2)

def stopCriteria1(xk, fxk, xk\_1, fxk\_1, e):

    if ((norm(xk\_1 - xk) / norm(xk)) <= e) and (math.fabs(fxk\_1 - fxk) <= e):

        return True

    return False

def stopCriteria2(gradFxk, e):

    if (norm(gradFxk) <= e):

        return True

    return False

# In[ ]:

# Code for finding optimal Lambda

functionCalculatedForLambda = 0

svenCoefLambda = 0.01 # 0.00001 # 0.000098

l0 = 0

eGoldenCut = 0.001 # 0.1 # 0.11

eDSKPauela = 0.001

'''

def f(x):

    x1 = x[0]

    x2 = x[1]

    return 3\*(x1-15)\*\*  2 -x1\*x2 +4\*x2\*\*2

'''

def FLambda(xk, l, Sk):

    global functionCalculatedForLambda

    functionCalculatedForLambda += 1

    return f(xk + l\*Sk)

# trustInterval = sven(np.array([-22, -22]), np.array([0, 1]))

# print(goldenCut(trustInterval, np.array([-22, -22]), np.array([0, 1])))

# print(functionCalculatedForLambda)

# In[ ]:

def sven(xk, Sk):

    deltaL = svenCoefLambda \* (norm(xk) / norm(Sk))

    FLambda0 = FLambda(xk, l0, Sk)

    FLambdaPlus = FLambda(xk, l0+deltaL, Sk)

    FLambdaMinus = FLambda(xk, l0-deltaL, Sk)

    coef = 0

    currentFValue = 0

    if FLambda0 < FLambdaPlus and FLambda0 < FLambdaMinus:

        # print("Sven: both Lambdas are increasing funcion")

        return np.array([-deltaL, deltaL])

    elif FLambda0 > FLambdaPlus and FLambda0 > FLambdaMinus:

        # raise Exception("Sven: both Lambdas are deacreasing function")

        print("Sven: both Lambdas are deacreasing function")

        if FLambdaPlus <= FLambdaMinus:

            coef = 1

            currentFValue = FLambdaPlus

        else:

            coef = -1

            currentFValue = FLambdaMinus

    elif FLambda0 > FLambdaPlus:

        coef = 1

        currentFValue = FLambdaPlus

    elif FLambda0 > FLambdaMinus:

        coef = -1

        currentFValue = FLambdaMinus

    else:

        # raise Exception(f"Can't choose Lambda, Left: {FLambdaMinus}, Center: {FLambda0} Right: {FLambdaPlus}")

        return np.array([FLambdaMinus, FLambdaPlus])

    l1 = l0 + coef \* deltaL

    currentlValue = l1

    lambdaValues = [l0, l1]

    FValues = [FLambda0, currentFValue]

    FNewLambda = 0

    n = 1

    while True:

        newLambda = currentlValue + 2\*n\*coef \* deltaL

        lambdaValues.append(newLambda)

        FNewLambda = FLambda(xk, newLambda, Sk)

        FValues.append(FNewLambda)

        if (currentFValue < FNewLambda):

            break

        currentlValue = newLambda

        currentFValue = FNewLambda

        n = 2\*n

    # print(f"SVEN - deltaL: {deltaL}, Lambdas: {lambdaValues}, functionCalculatedForLambda: {functionCalculatedForLambda}")

    Fcenter = FLambda(xk, (lambdaValues[-2] + lambdaValues[-1]) / 2, Sk)

    if Fcenter < FValues[-2]:

        if coef == 1:

            return np.array([lambdaValues[-2], lambdaValues[-1]])

        elif coef == -1:

            return np.array([lambdaValues[-1], lambdaValues[-2]])

    else:

        if coef == 1:

            return np.array([lambdaValues[-3], (lambdaValues[-2] + lambdaValues[-1]) / 2])

        elif coef == -1:

            return np.array([(lambdaValues[-2] + lambdaValues[-1]) / 2, lambdaValues[-3]])

# In[ ]:

def goldenCut(lTrustInterval, xk, Sk):

    a = lTrustInterval[0]

    b = lTrustInterval[1]

    biggerCut = (math.sqrt(5) - 1) / 2

    smallerCut = 1 - biggerCut

    L = b - a

    x1 = a + smallerCut \* L

    x2 = a + biggerCut \* L

    Fx1 = FLambda(xk, x1, Sk)

    Fx2 = FLambda(xk, x2, Sk)

    while (L > eGoldenCut):

        if Fx1 < Fx2:

            b = x2

            x2 = x1

            L = b - a

            x1 = a + smallerCut \* L

            Fx2 = Fx1

            Fx1 = FLambda(xk, x1, Sk)

        else:

            a = x1

            x1 = x2

            L = b - a

            x2 = a + biggerCut \* L

            Fx1 = Fx2

            Fx2 = FLambda(xk, x2, Sk)

    # print(f"GOLDENCUT - INTERVAL: {np.array([x1, x2])}, functionCalculatedForLambda: {functionCalculatedForLambda}")

    return np.array([x1, x2])

# In[ ]:

def dskStopCriteria(x2, Fx2, x\_new, Fx\_new):

    if (math.fabs(Fx2 - Fx\_new) <= eDSKPauela) and (math.fabs(x2 - x\_new) <= eDSKPauela):

        return True

    return False

# NOTE: OPTIMIZE DSKPAUELA

def DSKPauela(lTrustInterval, xk, Sk):

    # First iteration

    x1 = lTrustInterval[0]

    x2 = (lTrustInterval[0] + lTrustInterval[1]) / 2

    x3 = lTrustInterval[1]

    deltaX = x2 - x1

    Fx1 = FLambda(xk, x1, Sk)

    Fx2 = FLambda(xk, x2, Sk)

    Fx3 = FLambda(xk, x3, Sk)

    x\_new = x2 + (deltaX \* (Fx1 - Fx3)) / (2\*(Fx1 - 2\*Fx2 + Fx3))

    # print(f"x1: {x1}, x2: {x2}, x3: {x3}, Fx1: {Fx1}, Fx2: {Fx2}, Fx3: {Fx3}")

    i = 0

    while True:

        i += 1

        Fx\_new = FLambda(xk, x\_new, Sk)

        if dskStopCriteria(x2, Fx2, x\_new, Fx\_new):

            # print(f"DSKPAUELA - Lambda: {x\_new}, functionCalculatedForLambda: {functionCalculatedForLambda}")

            return x\_new

        if Fx\_new < Fx2:

            if x1 < x\_new and x\_new < x2:

                x1 = x1

                x3 = x2

                x2 = x\_new

                Fx1 = Fx1

                Fx2 = Fx\_new

                Fx3 = Fx2

            elif x2 < x\_new and x\_new < x3:

                x1 = x2

                x3 = x3

                x2 = x\_new

                Fx1 = Fx2

                Fx2 = Fx\_new

                Fx3 = Fx3

            else:

                raise Exception(f"DSKPauela: can't choose points: x1: {x1} x2: {x2}, x3: {x3}, x\_new: {x\_new}, iteration: {i}")

        elif Fx\_new > Fx2:

            if x\_new > x2:

                x1 = x1

                x2 = x2

                x3 = x\_new

                Fx1 = Fx1

                Fx2 = Fx2

                Fx3 = Fx\_new

            elif x\_new < x2:

                x1 = x\_new

                x2 = x2

                x3 = x3

                Fx1 = Fx\_new

                Fx2 = Fx2

                Fx3 = Fx3

            else:

                raise Exception(f"Equal points")

        else:

            raise Exception(f"Equal points")

        a1 = (Fx2 - Fx1) / (x2 - x1)

        a2 = (1 / (x3 - x2)) \* ((Fx3 - Fx1) / (x3 - x1) - (Fx2 - Fx1) / (x2 - x1))

        x\_new = (x1 + x2) / 2 - a1 / (2\*a2)

# In[ ]:

# NOT IMPLEMENTED PROPERLY

def adam(m, v, l, Sk, i, alpha=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8):

    m = beta1 \* m + (1 - beta1) \* Sk

    v = beta2 \* v + (1 - beta2) \* (Sk \*\* 2)

    m\_hat = m / (1 - beta1 \*\* i)

    v\_hat = v / (1 - beta2 \*\* i)

    l = l - alpha \* m\_hat / (np.sqrt(v\_hat) + epsilon)

    return m, v, l

# In[ ]:

import time

pathOfMethod = []

functionCalculated = 0

def f(x):

    x1 = x[0]

    x2 = x[1]

    global functionCalculated

    functionCalculated += 1

    return 100\*(x1\*\*2-x2)\*\*2 + (x1-1)\*\*2

def MSD(x0, h, e, stopCriteria = 'fchange', derType='center', l\_type='opt', mop\_type='gc', modMethod = 'None', iprint=1):

    global pathOfMethod

    pathOfMethod = []

    pathOfMethod.append(x0)

    i = 0

    oldxk = x0

    xk = x0

    Fxk = f(xk)

    gradFxk = fder(xk, Fxk, h, type=derType)

    oldgrad = gradFxk

    l = 2.5

    lusternickKick = False

    while True:

        i += 1

        if i % iprint == 0:

            print(f"\nITERATION NUMBER {i}")

        Sk = - gradFxk

        if l\_type == 'opt':

            if mop\_type == 'gc':

                l = goldenCut(sven(xk, Sk), xk, Sk)

            elif mop\_type == 'dsk':

                l = DSKPauela(sven(xk, Sk), xk, Sk)

            else:

                raise Exception('Not possible mop\_type')

            if modMethod == 'booth':

                sigma = 0.9

                xk\_new = xk + sigma \* l \* Sk

                Fxk\_new = f(xk\_new)

            else:

                xk\_new = xk + l \* Sk

                Fxk\_new = f(xk\_new)

        elif l\_type == 'const':

            if modMethod == 'lustr' and not np.array\_equal(oldgrad, gradFxk) and norm(oldgrad) != 0:

                Bk = norm(gradFxk) / norm(oldgrad)

                if Bk > 0.95:

                    lusternickKick = True

                    xk\_new = xk + 2 \* l \* Sk

                    Fxk\_new = f(xk\_new)

                else:

                    while True:

                        xk\_new = xk + l \* Sk / norm(Sk)

                        Fxk\_new = f(xk\_new)

                        if Fxk\_new > Fxk:

                            l = l / 2

                        else:

                            break

            else:

                while True:

                    xk\_new = xk + l \* Sk / norm(Sk)

                    Fxk\_new = f(xk\_new)

                    if Fxk\_new > Fxk:

                        l = l / 2

                    else:

                        break

        else:

            raise Exception('Not possible l\_type')

        gradFxk\_new = fder(xk\_new, Fxk\_new, h, type=derType)

        if i % iprint == 0:

            print(f'Xk: {xk}, Xk+1: {xk\_new}, Lambda: {l}, Fxk: {Fxk}, Fxk+1: {Fxk\_new}, GradFxk: {gradFxk}, GradFxk+1: {gradFxk\_new}')

        if stopCriteria == 'fchange':

            if stopCriteria1(xk, Fxk, xk\_new, Fxk\_new, e):

                return xk\_new

        elif stopCriteria == 'grad':

            if stopCriteria2(gradFxk\_new, e):

                return xk\_new

        else:

            raise Exception('Not possible stop criteria')

        oldxk = xk

        oldgrad = gradFxk

        pathOfMethod.append(xk\_new)

        xk = xk\_new

        Fxk = Fxk\_new

        gradFxk = gradFxk\_new

        if lusternickKick:

            oldgrad = gradFxk\_new

            lusternickKick = False

        if functionCalculated > 600000:

            print('EXITED: TOO MUCH OPEARTIONS')

            time.sleep(3)

            return xk

x0 = np.array([-1.2, 0])

h = 0.0000001 # 0.0000001

e = 0.00000001

eDSKPauela = 0.001

eGoldenCut = 0.001

svenCoefLambda = 10\*\*(-10)

functionCalculated = 0

functionCalculatedForLambda = 0

res = MSD(x0, h, e, derType='left', l\_type='opt', mop\_type='dsk', stopCriteria='fchange', modMethod='lustr1', iprint=1000)

print(functionCalculated)

print(np.round(res, 64))

print(round(f(res), 8))

# In[ ]:

def printArray(array):

    for a in array:

        print(a)

e = 0.001

eDSKPauela = 0.001

eGoldenCut = 0.001

svenCoefLambda = 0.01

functionCalculated = 0

functionCalculatedForLambda = 0

h\_values = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001,

                     0.0000000001, 0.00000000001, 0.000000000001, 0.0000000000001, 0.00000000000001,

                     0.000000000000001])

resultsMinimum = []

resultsFcalc = []

resultsF = []

for h in h\_values:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='opt', mop\_type='dsk')

    resultsMinimum.append(res)

    resultsFcalc.append(functionCalculated)

    resultsF.append(f(res))

printArray(np.round(resultsMinimum, 5))

printArray(np.round(resultsF, 5))

printArray(np.round(resultsFcalc, 5))

# In[ ]:

start = 0.01

stop = 0.000000000000001

step = 0.1

array = []

while start >= stop:

    array.append(start)

    start \*= step

h\_values = np.array(array)

h\_values = np.array([0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001,

                     0.0000000001, 0.00000000001, 0.000000000001, 0.0000000000001, 0.00000000000001,

                     0.000000000000001])

e = 0.001

rightMinimum = []

leftMinimum = []

centerMinimum = []

rightFcalc = []

leftFcalc = []

centerFcalc = []

print(h\_values)

global functionCalculated

for h in h\_values:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center')

    centerMinimum.append(res)

    centerFcalc.append(functionCalculated)

for h in h\_values:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='right')

    rightMinimum.append(res)

    rightFcalc.append(functionCalculated)

for h in h\_values:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='left')

    leftMinimum.append(res)

    leftFcalc.append(functionCalculated)

# In[ ]:

# Build plots

plt.figure(figsize=(16, 8))

plt.semilogx(h\_values, rightFcalc, label='Правостороння')

plt.semilogx(h\_values, leftFcalc, label='Лівостороння')

plt.semilogx(h\_values, centerFcalc, label='Центральна')

plt.xticks(h\_values)

# Adding a legend and labels

plt.legend()

plt.xlabel('h')

plt.ylabel('Кількість обчислень функції')

plt.gca().invert\_xaxis()

# Displaying the plot

plt.show()

# In[ ]:

resultsRigthMiminum = list(map(lambda x: f(x), rightMinimum))

resultsLeftMiminum = list(map(lambda x: f(x), leftMinimum))

resultsCenterMiminum = list(map(lambda x: f(x), centerMinimum))

plt.figure(figsize=(16, 8))

plt.semilogx(h\_values, resultsRigthMiminum, label='Правостороння')

plt.semilogx(h\_values, resultsLeftMiminum, label='Лівостороння')

plt.semilogx(h\_values, resultsCenterMiminum, label='Центральна')

plt.xticks(h\_values)

plt.yticks([0.001, 0.01, 0.01, 0.025, 0.05, 0.075, 0.1, 0.3])

# Adding a legend and labels

plt.legend()

plt.xlabel('h')

plt.ylabel('Значення функції')

plt.gca().invert\_xaxis()

# Displaying the plot

plt.show()

# In[ ]:

print(min(resultsLeftMiminum))

print(np.round(leftMinimum[resultsLeftMiminum.index(min(resultsLeftMiminum))], 5))

print(leftFcalc[resultsLeftMiminum.index(min(resultsLeftMiminum))])

print()

print(np.round(rightMinimum[resultsRigthMiminum.index(min(resultsRigthMiminum))], 5))

print(np.round(min(resultsRigthMiminum), 5))

print(rightFcalc[resultsRigthMiminum.index(min(resultsRigthMiminum))])

print()

# In[ ]:

eMOPvalues = np.array([1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001])

global functionCalculated

gcMinimum = []

dskMinimum = []

gcFCalc = []

dskFCalc = []

h = 0.0000001

e = 0.001

for e in eMOPvalues:

    eGoldenCut = e

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='opt', mop\_type='gc')

    gcMinimum.append(res)

    gcFCalc.append(functionCalculated)

for e in eMOPvalues:

    eDSKPauela = e

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='opt', mop\_type='dsk')

    dskMinimum.append(res)

    dskFCalc.append(functionCalculated)

# In[ ]:

resultsGCMiminum = list(map(lambda x: f(x), gcMinimum))

resultsDSKMiminum = list(map(lambda x: f(x), dskMinimum))

plt.figure(figsize=(16, 8))

plt.semilogx(eMOPvalues[5:], resultsGCMiminum[5:], label='Золотий переріз')

plt.semilogx(eMOPvalues[5:], resultsDSKMiminum[5:], label='ДСК Пауелла')

plt.xticks(eMOPvalues[5:])

# plt.yticks([0.001, 0.01, 0.01, 0.025, 0.05, 0.075, 0.1, 0.3])

# Adding a legend and labels

plt.legend()

plt.xlabel('e для МОП')

plt.ylabel('Значення функції')

plt.gca().invert\_xaxis()

# Displaying the plot

plt.show()

# In[ ]:

# Build plots

plt.figure(figsize=(16, 8))

plt.semilogx(eMOPvalues, gcFCalc, label='Золотий переріз')

plt.semilogx(eMOPvalues, dskFCalc, label='ДСК Пауелла')

plt.xticks(eMOPvalues)

# Adding a legend and labels

plt.legend()

plt.xlabel('h')

plt.ylabel('Кількість обчислень функції')

plt.gca().invert\_xaxis()

# Displaying the plot

plt.show()

# In[ ]:

print(eMOPvalues)

print(np.around(gcMinimum, 5))

printArray(np.around(resultsGCMiminum, 8))

printArray(gcFCalc)

# In[ ]:

print(np.around(dskMinimum, 5))

printArray(np.around(resultsDSKMiminum, 8))

printArray(dskFCalc)

# In[ ]:

# Зміна параметру Свена

svenCoefs = np.array([10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001,

                      0.0000000001, 0.00000000001, 0.000000000001, 0.0000000000001, 0.00000000000001,

                      0.000000000000001])

x0 = np.array([-1.2, 0])

h = 10\*\*(-15) #0.0000001

e = 0.001

eGoldenCut = 0.001

svenMinimum = []

svenFcalc = []

svenF = []

for s in svenCoefs:

    functionCalculated = 0

    svenCoefLambda = s

    res = MSD(x0, h, e, derType='center', l\_type='opt', mop\_type='gc')

    svenMinimum.append(res)

    svenFcalc.append(functionCalculated)

    svenF.append(f(res))

printArray(svenCoefs)

printArray(np.round(svenMinimum, 5))

printArray(np.round(svenF, 5))

printArray(np.round(svenFcalc, 5))

# In[ ]:

# Testing end criteria

x0 = np.array([-1.2, 0])

h = 0.000000000000001 # 0.0000001

eArray = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001]) # 0.0000001, 0.00000001]) # 0.000000001,

                      # 0.0000000001, 0.00000000001, 0.000000000001, 0.0000000000001, 0.00000000000001,

                      # 0.000000000000001])

eGoldenCut = 0.001

svenCoefLambda = 1

functionCalculated = 0

functionCalculatedForLambda = 0

fchangeMinimum = []

fchangeFcalc = []

fchangeF = []

# eArray = np.array([0.001])

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='opt', mop\_type='gc', stopCriteria='fchange')

    fchangeMinimum.append(res)

    fchangeFcalc.append(functionCalculated)

    fchangeF.append(f(res))

'''

gradMinimum = []

gradFcalc = []

gradF = []

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='opt', mop\_type='gc', stopCriteria='grad')

    gradMinimum.append(res)

    gradFcalc.append(functionCalculated)

    gradF.append(f(res))

'''

# In[ ]:

print(fchangeFcalc)

print(fchangeF)

# In[ ]:

from scipy.optimize import minimize

def rosenbrok(x):

    return 100\*(x[0]\*\*2-x[1])\*\*2 + (x[0]-1)\*\*2

result = minimize(rosenbrok, x0, method='CG')

print(result)

# In[ ]:

# Дослідження методу Бута

x0 = np.array([-1.2, 0])

h = 0.0000001 # 0.0000001

eDSKPauela = 0.001

eGoldenCut = 0.0001

svenCoefLambda = 1

functionCalculated = 0

functionCalculatedForLambda = 0

eArray = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001])

withoutModMinimum = []

withoutModFcalc = []

withoutModF = []

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='opt', mop\_type='gc', stopCriteria='fchange')

    withoutModMinimum.append(res)

    withoutModFcalc.append(functionCalculated)

    withoutModF.append(f(res))

boothModMinimum = []

boothModFcalc = []

boothModF = []

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='opt', mop\_type='gc', stopCriteria='fchange', modMethod='booth')

    boothModMinimum.append(res)

    boothModFcalc.append(functionCalculated)

    boothModF.append(f(res))

# In[ ]:

printArray(np.round(withoutModMinimum, 6))

printArray(withoutModFcalc)

printArray(withoutModF)

printArray(np.round(boothModMinimum, 6))

printArray(boothModFcalc)

printArray(boothModF)

# In[ ]:

# Дослідження методу Бута та Люстерніка

x0 = np.array([-1.2, 0])

h = 0.0000001 # 0.0000001

eDSKPauela = 0.001

eGoldenCut = 0.0001

svenCoefLambda = 1

functionCalculated = 0

functionCalculatedForLambda = 0

eArray = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001])

withoutModMinimum = []

withoutModFcalc = []

withoutModF = []

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='const', mop\_type='gc', stopCriteria='fchange', iprint=1000)

    withoutModMinimum.append(res)

    withoutModFcalc.append(functionCalculated)

    withoutModF.append(f(res))

lustrModMinimum = []

lustrModFcalc = []

lustrModF = []

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='center', l\_type='const', mop\_type='gc', stopCriteria='fchange', modMethod='lustr', iprint=1000)

    lustrModMinimum.append(res)

    lustrModFcalc.append(functionCalculated)

    lustrModF.append(f(res))

# In[ ]:

# printArray(np.round(withoutModMinimum, 6))

# printArray(withoutModFcalc)

# printArray(withoutModF)

printArray(np.round(lustrModMinimum, 8))

printArray(lustrModFcalc)

printArray(lustrModF)

# In[ ]:

# Дослідження методу Люстерніка

x0 = np.array([-1.2, 0])

h = 0.0000001 # 0.0000001

eArray = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001])

eDSKPauela = 0.001

eGoldenCut = 0.001

svenCoefLambda = 1

functionCalculated = 0

functionCalculatedForLambda = 0

res = MSD(x0, h, e, derType='center', l\_type='const', mop\_type='gc', stopCriteria='fchange', modMethod='lustr', iprint=50)

print(functionCalculated)

print(np.round(res, 8))

print(round(f(res), 8))

# In[ ]:

# Дослідження критерію зупинки

x0 = np.array([-1.2, 0])

h = 0.0000001 # 0.0000001

eDSKPauela = 0.001

eGoldenCut = 0.001

svenCoefLambda = 10\*\*(-10)

functionCalculated = 0

functionCalculatedForLambda = 0

eArray = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001, 0.0000000001, 0.00000000001])

fchangeMinimum = []

fchangeFcalc = []

fchangeF = []

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='left', l\_type='opt', mop\_type='dsk', stopCriteria='fchange', iprint=1000)

    fchangeMinimum.append(res)

    fchangeFcalc.append(functionCalculated)

    fchangeF.append(f(res))

print('ok')

gradMinimum = []

gradFcalc = []

gradF = []

for e in eArray:

    functionCalculated = 0

    h = e \* 10\*\*(-4)

    res = MSD(x0, h, e, derType='left', l\_type='opt', mop\_type='dsk', stopCriteria='grad', iprint=1000)

    gradMinimum.append(res)

    gradFcalc.append(functionCalculated)

    gradF.append(f(res))

# printArray(np.round(fchangeMinimum, 12))

# printArray(gradFcalc)

# printArray(gradMinimum)

# In[ ]:

x0 = np.array([-1.2, 0])

h = 0.0000001 # 0.0000001

eDSKPauela = 0.001

eGoldenCut = 0.001

svenCoefLambda = 1

functionCalculated = 0

functionCalculatedForLambda = 0

eArray = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001])

fchangeMinimum = []

fchangeFcalc = []

fchangeF = []

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='right', l\_type='opt', mop\_type='gc', stopCriteria='fchange', iprint=10000)

    fchangeMinimum.append(res)

    fchangeFcalc.append(functionCalculated)

    fchangeF.append(f(res))

printArray(np.round(fchangeMinimum, 8))

# In[ ]:

x0 = np.array([-1.2, 0])

h = 0.0000001 # 0.0000001

eDSKPauela = 0.001

eGoldenCut = 0.001

svenCoefLambda = 10\*\*(-10)

functionCalculated = 0

functionCalculatedForLambda = 0

eArray = np.array([0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001, 0.0000000001,

                   0.000000000001, 0.0000000000001, 0.00000000000001, 0.000000000000001, 0.0000000000000001])

fchangeMinimum = []

fchangeFcalc = []

fchangeF = []

for e in eArray:

    functionCalculated = 0

    res = MSD(x0, h, e, derType='left', l\_type='opt', mop\_type='dsk', stopCriteria='fchange', iprint=10000)

    fchangeMinimum.append(res)

    fchangeFcalc.append(functionCalculated)

    fchangeF.append(f(res))

printArray(np.round(fchangeF, 28))

# In[ ]:

plt.figure(figsize=(14, 8))

plt.bar(range(len(eArray)), fchangeFcalc, align='edge')

plt.xlabel('Точність', fontsize=12)

plt.ylabel('Кількість обчислень цільової функції', fontsize=12)

plt.title('Графік кількості підрахунків функції відносно точності методу МНС', fontsize=14)

# Set custom x-tick labels

x\_ticks = [i + 0.5 for i in range(len(eArray))]

plt.xticks(x\_ticks, eArray)

plt.show()

plt.show()

# In[ ]:

plt.figure(figsize=(14, 8))

plt.plot(range(len(eArray)), fchangeF)

plt.xticks(range(len(eArray)), eArray)

plt.xlabel('Точність')

plt.ylabel('Значення функції')

plt.title('Значення функції залежно від точності методу')

plt.show()

# In[ ]:

pathOfUOoptimizer = []

BASEDR = 1

def f(x):

    x1 = x[0]

    x2 = x[1]

    global functionCalculated

    functionCalculated += 1

    # HERE

    if (x1)\*\*2+(x2)\*\*2 <= 1:

        R = 0

    else:

        global BASEDR

        R = BASEDR

        print(R)

    return 100\*(x1\*\*2-x2)\*\*2 + (x1-1)\*\*2 + R\*(1-(x1)\*\*2-(x2)\*\*2)\*\*2

x0 = np.array([-1.2, 0]) # -3 -3 10000000 e = 0.001

h = 10\*\*(-7) # 0.0000001

e = 0.0001

eDSKPauela = 0.001

eGoldenCut = 0.001

svenCoefLambda = 1

functionCalculated = 0

functionCalculatedForLambda = 0

eUO = 0.0001

R = 0.1

x = x0

while True:

    R = R\*10

    BASEDR = R

    res = MSD(x, h, e, derType='right', l\_type='opt', mop\_type='gc', stopCriteria='fchange', iprint=1000)

    pathOfUOoptimizer.extend(pathOfMethod)

    if stopCriteria1(x, f(x), res, f(res), eUO):

        break

    x = res

    # print(np.round(f(res), 8))

    # print(functionCalculated)

# print(functionCalculated)

print(np.round(f(res), 8))

# In[ ]:

def freal(x):

    x1 = x[0]

    x2 = x[1]

    return 100\*(x1\*\*2-x2)\*\*2 + (x1-1)\*\*2

def area(x):

    x1 = x[0]

    x2 = x[1]

    return (x1)\*\*2 + (x2)\*\*2 - 1

# Generate data points for plotting

x = np.linspace(-3, 3, 1000)

y = np.linspace(-3, 3, 1000)

X, Y = np.meshgrid(x, y)

Z\_freal = freal([X, Y])

Z\_area = area([X, Y])

# Create a 2D plot for freal

fig, ax = plt.subplots(figsize=(16, 12))

cmap = ax.contourf(X, Y, Z\_freal, levels=30, cmap='viridis')

# Add contour lines for area

contour = ax.contour(X, Y, Z\_area, levels=[0], colors='red')

# Set labels and title

ax.set\_xlabel('x1')

ax.set\_ylabel('x2')

ax.set\_title('Умовна оптимізація функції Розенброка')

# Add a colorbar

cbar = fig.colorbar(cmap)

ax.plot(1, 1, 'ro', label='БО Мінімум', color = 'Green')

# Plot the pathOfMethod

path\_x = [point[0] for point in pathOfUOoptimizer]

path\_y = [point[1] for point in pathOfUOoptimizer]

ax.plot(path\_x, path\_y, 'bo-', label='Шлях методу')

ax.plot(-1.2, 0, 'ro', label='Початкова точка', color = 'Yellow')

# Show the legend

ax.legend()

# Show the plot

plt.show()

# In[ ]:

pathOfUOoptimizer = []

BASEDR = 1

def f(x):

    x1 = x[0]

    x2 = x[1]

    global functionCalculated

    global BASEDR

    functionCalculated += 1

    # HERE

    if (x1-1)\*\*2+(x2-1)\*\*2 <= 1:

        R1 = 0

    else:

        R1 = BASEDR

    if (x1-1)\*\*2+(x2-1)\*\*2 >= 1/4:

        R2 = 0

    else:

        R2 = BASEDR

    return 100\*(x1\*\*2-x2)\*\*2 + (x1-1)\*\*2 + R1\*(1-(x1-1)\*\*2-(x2-1)\*\*2)\*\*2 + R2\*(1/4-(x1-1)\*\*2-(x2-1)\*\*2)\*\*2

x0 = np.array([-1.2, 0]) # -3 -3 10000000 e = 0.001

h = 10\*\*(-7) # 0.0000001

e = 0.0001

eDSKPauela = 0.001

eGoldenCut = 0.001

svenCoefLambda = 1

functionCalculated = 0

functionCalculatedForLambda = 0

eUO = 0.0001

R = 0.1

x = x0

while True:

    R = R\*4

    BASEDR = R

    res = MSD(x, h, e, derType='right', l\_type='opt', mop\_type='gc', stopCriteria='fchange', iprint=1000)

    pathOfUOoptimizer.extend(pathOfMethod)

    # print(np.round(res, 8))

    # print(np.round(f(res), 8))

    print(functionCalculated)

    if stopCriteria1(x, f(x), res, f(res), eUO):

        break

    x = res

# In[ ]:

def freal(x):

    x1 = x[0]

    x2 = x[1]

    return 100\*(x1\*\*2-x2)\*\*2 + (x1-1)\*\*2

def areaOutter(x):

    x1 = x[0]

    x2 = x[1]

    return (x1-1)\*\*2 + (x2-1)\*\*2 - 1

def areaInner(x):

    x1 = x[0]

    x2 = x[1]

    return (x1-1)\*\*2 + (x2-1)\*\*2 - 1/4

# Generate data points for plotting

x = np.linspace(-3.5, 3.5, 1000)

y = np.linspace(-3.5, 3.5, 1000)

X, Y = np.meshgrid(x, y)

Z\_freal = freal([X, Y])

Z\_areaOutter = areaOutter([X, Y])

Z\_areaInner = areaInner([X, Y])

# Create a 2D plot for freal

fig, ax = plt.subplots(figsize=(16, 12))

cmap = ax.contourf(X, Y, Z\_freal, levels=30, cmap='viridis')

# Add contour lines for area

contour1 = ax.contour(X, Y, Z\_areaOutter, levels=[0], colors='red')

contour2 = ax.contour(X, Y, Z\_areaInner, levels=[0], colors='red')

# Set labels and title

ax.set\_xlabel('x1')

ax.set\_ylabel('x2')

ax.set\_title('Умовна оптимізація функції Розенброка')

# Add a colorbar

cbar = fig.colorbar(cmap)

ax.plot(1, 1, 'ro', label='БО Мінімум', color = 'Green')

# Plot the pathOfMethod

path\_x = [point[0] for point in pathOfUOoptimizer]

path\_y = [point[1] for point in pathOfUOoptimizer]

ax.plot(path\_x, path\_y, 'bo-', label='Шлях методу')

ax.plot(-1.2, 0, 'ro', label='Початкова точка', color = 'Yellow')

# Show the legend

ax.legend()

# Show the plot

plt.show()