МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра прикладної математики

Курсовий проект на тему:

«Метод найшвидшого спуску. Партан-метод найшвидшого спуску»

із дисципліни

«Дослідження операцій»

Виконали:

Студенти групи КМ-72

Борознюк Д.О., Городецький Д.С.

Керівник:

Ладогубець Т.С.

Київ 2020

ЗМІСТ

[ПОСТАНОВКА ЗАДАЧІ 3](#_Toc42091556)

[1 ТЕОРЕТИЧНІ ВІДОМОСТІ 4](#_Toc42091557)

[1.1 Методи мінімізації, що використовують похідні. Градієнтні методи. 4](#_Toc42091558)

[1.2 Метод найшвидшого спуску (Коші) 4](#_Toc42091559)

[1.3 Партан-методи. Партан-метод найшвидшого спуску. 6](#_Toc42091560)

[2 БЕЗУМОВНА ОПТИМІЗАЦІЯ 8](#_Toc42091561)

[2.1 Функція Розенброка 8](#_Toc42091562)

[2.1.1 Схема обчислення похідних 8](#_Toc42091563)

[2.1.2 Величина кроку h при обчисленні похідних. 10](#_Toc42091564)

[2.1.3 Спосіб обчислення кроку 11](#_Toc42091565)

[2.1.4 Вид методу одновимірного пошуку 12](#_Toc42091566)

[2.1.5 Точність методу одновимірного пошуку 13](#_Toc42091567)

[2.1.6 Значення параметру в алгоритмі Свенна 14](#_Toc42091568)

[2.1.7 Вигляд критерія закінчення 15](#_Toc42091569)

[2.2 Коренева функція 16](#_Toc42091570)

[2.2.1 Схема обчислення похідних 16](#_Toc42091571)

[2.2.2 Величина кроку h при обчисленні похідних. 18](#_Toc42091572)

[2.2.3 Спосіб обчислення кроку 19](#_Toc42091573)

[2.2.4 Вид методу одновимірного пошуку 20](#_Toc42091574)

[2.2.5 Точність методу одновимірного пошуку 22](#_Toc42091575)

[2.2.6 Значення параметру в алгоритмі Свенна 23](#_Toc42091576)

[2.2.7 Вигляд критерія закінчення 24](#_Toc42091577)

[3 УМОВНА ОПТИМІЗАЦІЯ 25](#_Toc42091578)

[3.1 Теоретичні відомості 25](#_Toc42091579)

[3.2 Функція Розенброка 26](#_Toc42091580)

[ВИСНОВОК 31](#_Toc42091581)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 34](#_Toc42091582)

[ДОДАТОК А (лістинг програми безумовної оптимізації) 35](#_Toc42091583)

[ДОДАТОК Б (лістинг програми умовної оптимізації) 40](#_Toc42091584)

# ПОСТАНОВКА ЗАДАЧІ

Дослідити збіжність методу найшвидшого спуску, партан-методу найшвидшого спуску при мінімізації кореневої функції та 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.

.

Використати метод штрафних функцій (метод зовнішньої точки) для умовної оптимізації найкращого з цих методів при розташуванні локального мінімума поза допустимої області(випукла/невипукла).

# 1 ТЕОРЕТИЧНІ ВІДОМОСТІ

## Методи мінімізації, що використовують похідні. Градієнтні методи.

При використанні прямих методів пошуку потрібно тільки обчислення значень функції, але кількість цих обчислень може бути досить великою. Методи першого порядку використовують інформацію про значеннях похідних функцій, тобто дають можливість знаходження стаціонарних точок, що задовольняють необхідній умові першого порядку.

Методи, які використовують похідні, носять ітеративний характер, так як компоненти градієнта є нелінійними функціями. Градієнт вказує напрямок зростання цільової функції. Оскільки методи використовують значення похідних, передбачається, що - існують і неперервні.

Градієнтні методи використовують значення перших похідних цільової функції. На k-му етапі перехід з точки в точку визначається наступною ітераційною процедурою:

, де

- вектор переходу з точки  в точку ;

- одиничний вектор в напрямку ;

- вектор в напрямку ;

 довжина кроку (скаляр).

Визначення  пов'язано з особливостями застосовуваного методу.

## Метод найшвидшого спуску (Коші)

Градієнт цільової функції вказує на найбільше локальне збільшення функції, отже, в задачах мінімізації функції потрібно рухатися в напрямку, протилежному градієнту, тобто в напрямку антиградієнту (напрямку найшвидшого спуску). Антиградієнт в точці спрямований в бік зменшення цільової функції f (x) і ортогональний до ліній рівня в точці.

– напрям протилежний градієнту функції в точці (напрямок найшвидшого спуску).

- напрям, протилежний нормованому(одиничному) градієнту функції в точці.

Застосовується два методи вибору довжини кроку:

1. величина вибирається фіксованою, або змінюється від кроку до кроку;
2. при переході з точки в точку цільова функція мінімізується по λ за допомогою методів одновимірного пошуку.

Для опуклої цільової функції, що має похідні до 3-ого порядку, цей метод сходиться при *k*→∞. Це лінійна швидкість збіжності. Швидкість збіжності може бути занадто повільною внаслідок малого значення ∇*f(x)* в околі мінімуму. При цьому не можна прискорити рух до точки мінімуму. Цей метод використовується, як початкова процедура в задачах мінімізації.

Основною складністю при використанні методу найшвидшого спуску є його залежність від вибору масштабу оптимізуючих змінних. Якщо гіперпростір дуже витягнутий, так, що утворює «хребет», або «яр» (погано обумовлена ​​матриця Гессе), процедура найшвидшого спуску сходиться дуже повільно.

На рисунку 1 показана типова звивиста траєкторія оптимізації при використанні даної процедури у вузькій впадині. Напрямок найшвидшого спуску виявляється майже ортогональним найкращому напрямку досягнення мінімуму. У таких випадках необхідно використовувати інформацію про другі похідні.

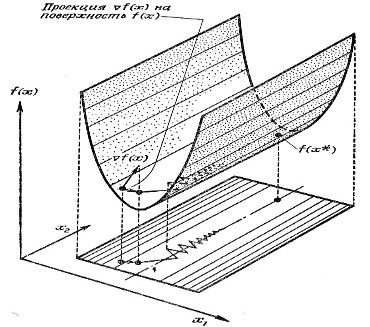


Рис 1.2.1. Траєкторія пошуку у вузькій впадині

## Партан-методи. Партан-метод найшвидшого спуску.

Партан – це скорочення від терміна parallel tangents (паралельні дотичні). Партан-методи використовують спряжені напрямки та спряжені градієнти.

Розглянемо загальний партан-алгоритм для випадку квадратичної функції 2-х змінних. Р1 і Р2 - будь-які дві точки площини. Спочатку відбувається рух з Р2 паралельно дотичній до лінії рівня в точці Р1 до тих пір, поки не буде досягнутий мінімум функції в деякій точці Р3. Дотичні в Р1 і Р3 паралельні, а мінімум f(x) знаходиться на лінії, що проходить через точки Р1 і Р3. Напрямки, отримані з допомогою загального Партан - алгоритму, є спряженими.



Рис. 1.3.1 Загальний партан-метод

В партан-методі найшвидшого спуску напрямки обираються у напрямку, протилежному градієнту. Даний метод має скінченну збіжність.

Алгоритм ітераційного Партан - методу.

Початкові дані:

* початкова точка ;
* параметр закінчення.

1. Визначається напрямок антиградієнта в точці
2. Функція мінімізується уздовж цього напрямку, виходячи з , отримуємо точку .
3. Визначається напрямок антиградієнта в точці .
4. Функція мінімізується уздовж цього напрямку, виходячи з, отримуємо точку .
5. Точки і з'єднуються прямою, вздовж якої проводиться мінімізація функції f (x), отримуємо точку .
6. Перевіряється виконання критерію закінчення пошуку, наприклад

Якщо нерівність не виконується, процедура повторюється з початковою точкою . Якщо завдання містить n змінних, слід зробити n градієнтних кроків, а потім з'єднати  з .

# БЕЗУМОВНА ОПТИМІЗАЦІЯ

## Функція Розенброка

Було проведено дослідження методів Коші та партан-методу в залежності від наступних параметрів.

### Схема обчислення похідних

Кожен з методів було досліджено на лівій, правій та центральній схемах. Було обрано наступні параметри:

* Точність методу пошуку: [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001]
* Величина кроку h при обчисленні похідних: 0.00001;
* Алгоритм одновимірного пошуку: Золотий перетин;
* Точність одновимірного пошуку: 0.0001;
* Значення параметру в алгоритмі Свенна: 0.001;

Результати дослідження:

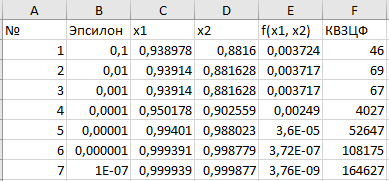


Рис 2.1.1.1 – Центральна схема для методу Коші

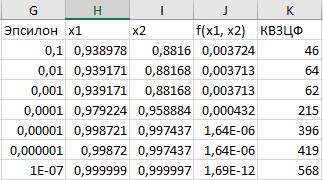


Рис 2.1.1.2 – Центральна схема для партан-методу

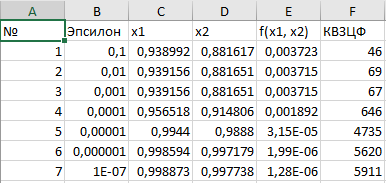


Рис 2.1.1.3 – Ліва схема для методу Коші

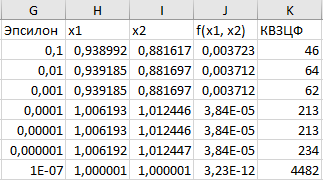


Рис 2.1.1.4 – Ліва схема для партан-методу

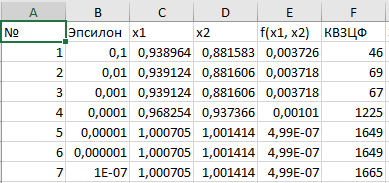


Рис 2.1.1.5 – Права схема для метода Коші

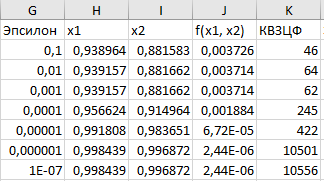


Рис 2.1.1.6 – Права схема для партан-методу

Отже, в подальшому дослідженні буде використано праву схему для методу Коші та центральну схему для партан-методу. Також за результатами дослідження було обрано точність методу ε = 0.00001 для обох методів.

### Величина кроку h при обчисленні похідних.

Початкові параметри для дослідження кроку:

* Алгоритм одновимірного пошуку: Золотий перетин;
* Точність одновимірного пошуку: 0.0001;
* Значення параметру в алгоритмі Свенна: 0.001;

та оптимальне значення точності методу, знайдене у попередньому розділі.

Метод Коші було досліджено на наступній множині величин кроку:

h = [0.0005, 0.0001, 0.00005, 0.00001, 0.000005, 0.000001] і отримані наступні результати:

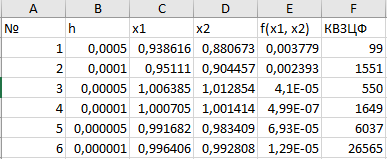


Рис 2.1.2.1 – Дослідження кроку h для метода Коші

За результатами дослідження було обрано крок h = 0.00005

Партан-метод було досліджено на наступній множині величин кроку: h = [0.1, 0.01, 0.001, 0.0001] і отримані наступні результати:

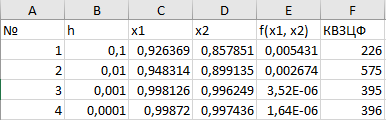


Рис 2.1.2.2 – Дослідження кроку h для партан-методу

За результатами дослідження було обрано крок h = 1-E3.

### Спосіб обчислення кроку

Для дослідження впливу способу обчислення кроку на збіжність методу Коші було обрано наступні параметри:

* Для постійного кроку:
  + Величина кроку h при обчисленні похідних: 0.00001;
  + Точність методу пошуку: 0.01
  + Схема обчислення похідних: Права
* Для оптимального кроку
  + Алгоритм одновимірного пошуку: Золотий перетин;
  + Точність одновимірного пошуку: 0.0001;
  + Значення параметру в алгоритмі Свенна: 0.001;

Та оптимальні параметри, знайдені у попередніх розділах.

Результати:

* Для постійного кроку:

Точка: [0.99493726 0.99000112], кількість обчислень значення цільової функції: 129728

* Для оптимального кроку:

Точка: [1.006385 1.012854], кількість обчислень значення цільової функції: 550

Для дослідження впливу кроку на збіжність партан-методу було обрано наступні параметри:

* Для постійного кроку:
  + Величина кроку h при обчисленні похідних: 0.001;
  + Точність методу пошуку: 0.001
  + Схема обчислення похідних: Центральна
* Для оптимального кроку
  + Алгоритм одновимірного пошуку: Золотий перетин;
  + Точність одновимірного пошуку: 0.0001;
  + Значення параметру в алгоритмі Свенна: 0.001;

Та оптимальні параметри, знайдені у попередніх розділах.

Результати:

* Для постійного кроку:

Точка: [0.99429727 0.99431204], кількість обчислень значення цільової функції: 5260

* Для оптимального кроку:

Точка: [0.998126 0.996249], кількість обчислень значення цільової функції: 395

Як і очікувалось, кількість обчислень значення цільової функції при використанні постійного кроку значно перевищує кількість обчислень при використанні оптимального кроку. Тобто використання постійного кроку не є обґрунтованим та в подальшому дослідженні буде використано саме оптимальний крок.

### Вид методу одновимірного пошуку

Для дослідження впливу методу одновимірного пошуку на збіжність методу Коші було обрано наступні параметри:

* + Точність одновимірного пошуку: 0.0001;
  + Значення параметру в алгоритмі Свенна: 0.001;

Та оптимальні параметри, знайдені у попередніх розділах

Для золотого перетину:

Результати:

Точка: [1.00558311 1.0132957], кількість обчислень значення цільової функції: 495

Для ДСК-Пауелла:

Результати:

Точка: [0.99110834 0.98225091], кількість обчислень значення цільової функції: 7835

Для дослідження впливу методу одновимірного пошуку на збіжність партан-методу було обрано наступні параметри:

* + Точність одновимірного пошуку: 0.0001;
  + Значення параметру в алгоритмі Свенна: 0.001;

Та оптимальні параметри, знайдені у попередніх розділах

Для золотого перетину:

Результати:

Точка: [0.99830025 0.99615877], кількість обчислень значення цільової функції: 351

Для ДСК-Пауелла:

Результати:

Точка: [1.00162995 1.00260312], кількість обчислень значення цільової функції: 1356

За результатами дослідження для подальшого дослідження було обрано метод золотого перетину.

### Точність методу одновимірного пошуку

Для дослідження було обрано наступні параметри:

* Значення параметру в алгоритмі Свенна: 0.001;

Та оптимальні параметри, знайдені у попередніх розділах

Метод Коші було досліджено на наступній точності методу одновимірного пошуку:

odnom = [0.05,0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00001, 0.000001, 0.0000001] і отримані наступні результати:

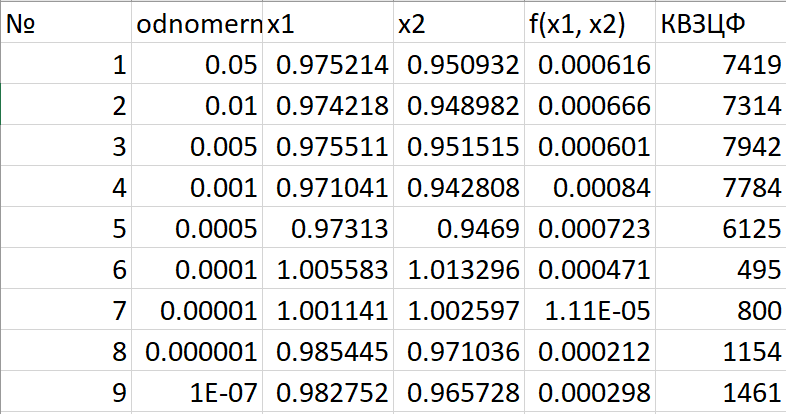


Рис 2.1.5.1 – Точність одновимірного пошуку для методу Коші

За результатами дослідження було обрано точність одновимірного пошуку odnom = 0.0001.

Партан-метод було досліджено на наступній точності методу одновимірного пошуку: odnom = [0.5, 0.1, 0.05,0.01, 0.005, 0.001, 0.0005,0.0001, 0.00001, 0.000001, 0.0000001] і отримані наступні результати:

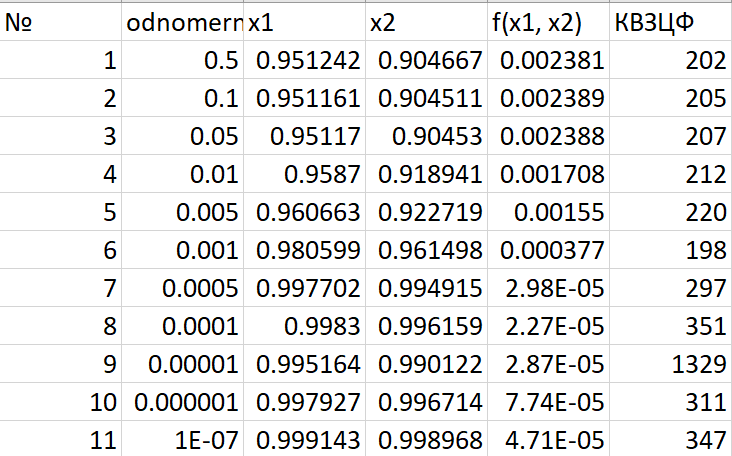


Рис 2.1.5.2 – Точність одновимірного пошуку для партан-методу

За результатами дослідження було обрано точність одновимірного пошуку odnom = 0.0005

### Значення параметру в алгоритмі Свенна

Для дослідження впливу значення параметру в алгоритмі Свенна на збіжність методу Коші було обрано оптимальні параметри, знайдені в попередніх розділах:

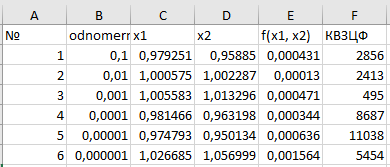


Рис 2.1.6.1 – Значення параметру в алгоритмі Свенна для методу Коші

За результатами дослідження було обрано параметр dx = 0.001

Для дослідження впливу методу одновимірного пошуку на збіжність партан-методу було оптимальні параметри, знайдені в попередніх розділах:

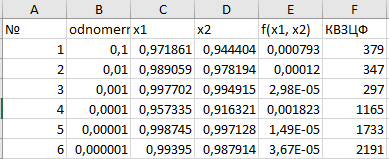


Рис 2.1.6.2 – Значення параметру в алгоритмі Свенна для партан-методу

За результатами дослідження було обрано параметр dx = 0.001

### Вигляд критерія закінчення

Для дослідження впливу методу одновимірного пошуку на збіжність методу Коші було оптимальні параметри, знайдені в попередніх розділах.

Для критерію :

Точка: [1.005583 1.013296], кількість обчислень значення цільової функції: 495

* Для критерію :

Використання даного критерію для методу Коші виявилось неефективним – оскільки норма градієнта мало змінюється при прямуванні до мінімуму (точки (1; 1)), необхідно брати більші значення точності методу пошуку. Це, в свою чергу, приводить до значних похибок округлення.

Для дослідження впливу критерія закінчення на збіжність партан-методу було обрано оптимальні параметри, знайдені в попередніх розділах.

Для критерію :

Точка: [0.997702 0.994915], кількість обчислень значення цільової функції: 297

* Для критерію :

Точка: [0.999842 0.999682], кількість обчислень значення цільової функції: 508

Отже, критерій виявися неефективним у знаходженні мінімуму цільової функції. Це пов’язано з тим, що величина норми градієнту мало змінюється, що в свою чергу призводить до збільшення кількості обчислень значення цільової функції та похибки методу Коші та партан-методу.

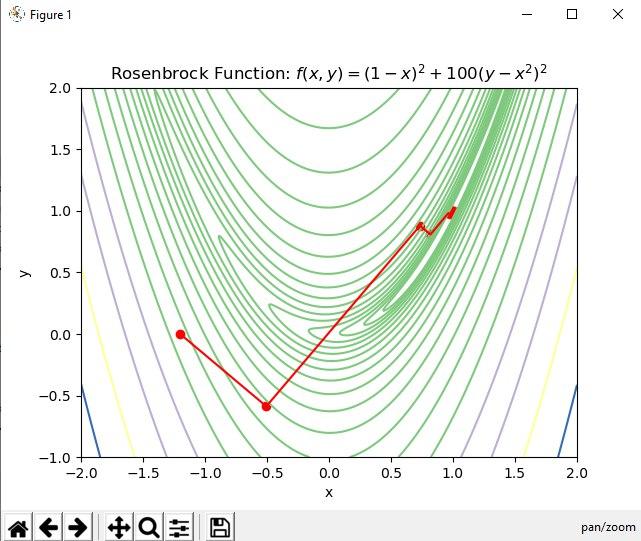


Рис 2.1.7.1 – Траєкторія руху метода Коші при найкращих параметрах

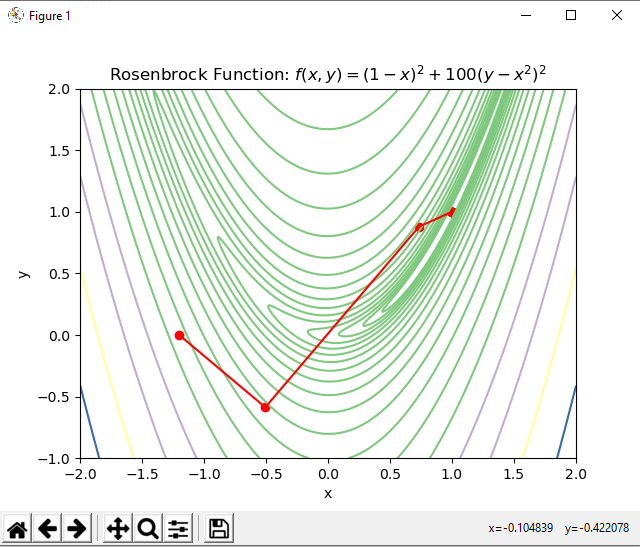


Рис 2.1.7.2 – Траєкторія руху партан-метода при найкращих параметрах

## Коренева функція

Було проведено дослідження методів Коші та партан-методу в залежності від наступних параметрів.

### Схема обчислення похідних

Кожен з методів було досліджено на лівій, правій та центральній схемах. Було обрано наступні параметри:

* Величина кроку h при обчисленні похідних: 0.001;
* Алгоритм одновимірного пошуку: Золотий перетин;
* Точність одновимірного пошуку: 0.0001;
* Значення параметру в алгоритмі Свенна: 0.01;
* Точність методу пошуку: [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001]

Результати дослідження:

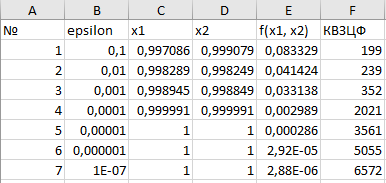


Рис 2.2.1.1 – Центральна схема для методу Коші

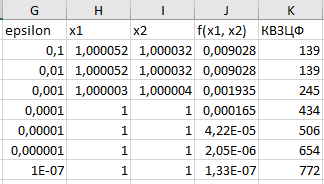


Рис 2.2.1.2 – Центральна схема для партан-методу

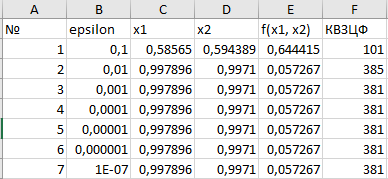


Рис 2.2.1.3 – Ліва схема для методу Коші

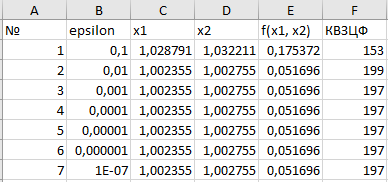


Рис 2.2.1.4 – Ліва схема для партан-методу

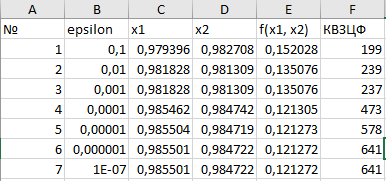


Рис 2.2.1.5 – Права схема для метода Коші

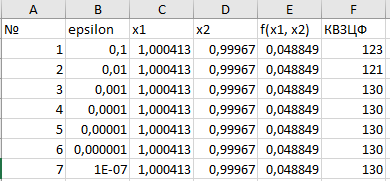


Рис 2.1.1.6 – Права схема для партан-методу

Отже, в подальшому дослідженні буде використано центральну схему для методу Коші та партан-методу. Також за результатами дослідження було обрано точність методу ε = 0.001 для метода Коші та ε = 0.0001 для партан-методу.

### Величина кроку h при обчисленні похідних.

Було обрано наступні параметри:

* Алгоритм одновимірного пошуку: Золотий перетин;
* Точність одновимірного пошуку: 0.0001;
* Значення параметру в алгоритмі Свенна: 0.01;

та оптимальне значення точності методу, знайдене у попередньому розділі.

Метод Коші було досліджено на наступній множині величин кроку:

h = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001] і отримані наступні результати:

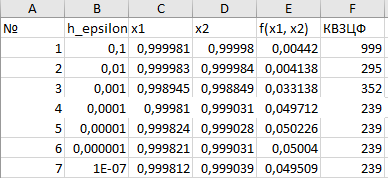


Рис 2.2.2.1 – Дослідження кроку h для метода Коші

За результатами дослідження було обрано крок h = 0.0001

Партан-метод було досліджено на наступній множині величин кроку: h = [0.1, 0.01, 0.001, 0.0001] і отримані наступні результати:

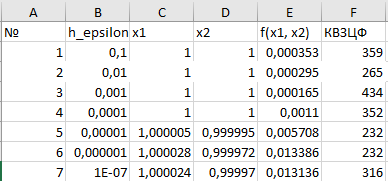


Рис 2.2.2.2 – Дослідження кроку h для партан-методу

За результатами дослідження було обрано крок h = 0.01.

### Спосіб обчислення кроку

Для дослідження впливу способу обчислення кроку на збіжність методу Коші було обрано наступні параметри:

* Для постійного кроку:
  + Величина кроку h при обчисленні похідних: 0.0001;
  + Точність методу пошуку: 0.001
  + Схема обчислення похідних: Центральна
* Для оптимального кроку
  + Величина кроку h при обчисленні похідних: 0.0001;
  + Алгоритм одновимірного пошуку: Золотий перетин;
  + Точність одновимірного пошуку: 0.001;
  + Значення параметру в алгоритмі Свенна: 0.001;
  + Точність методу пошуку: 0.001;

та оптимальні параметри, знайдені у попередніх розділах

Результати:

* Для постійного кроку:

Точка: [0.99726778 0.99728525], кількість обчислень значення цільової функції: 1192

* Для оптимального кроку:

Точка: [0.99981 0.999031], кількість обчислень значення цільової функції: 239

Для дослідження впливу кроку на збіжність партан-методу було обрано наступні параметри:

* Для постійного кроку:
  + Величина кроку h при обчисленні похідних: 0.01;
  + Точність методу пошуку: 0.0001
  + Схема обчислення похідних: Центральна
* Для оптимального кроку
  + Алгоритм одновимірного пошуку: Золотий перетин;
  + Точність одновимірного пошуку: 0.0001;
  + Значення параметру в алгоритмі Свенна: 0.001;

та оптимальні параметри, знайдені у попередніх розділах

Результати:

* Для постійного кроку:

Точка: [0.9999841 0.99175746], кількість обчислень значення цільової функції: 1212

* Для оптимального кроку:

Точка: [1 1], кількість обчислень значення цільової функції: 265

Як і очікувалось, кількість обчислень значення цільової функції при використанні постійного кроку значно перевищує кількість обчислень при використанні оптимального кроку. Тобто використання постійного кроку не є обґрунтованим та в подальшому дослідженні буде використано саме оптимальний крок.

### Вид методу одновимірного пошуку

Для дослідження впливу методу одновимірного пошуку на збіжність методу Коші було обрано наступні параметри:

Для золотого перетину:

* + Величина кроку h при обчисленні похідних: 0.0001;
  + Точність одновимірного пошуку: 0.0001;
  + Значення параметру в алгоритмі Свенна: 0.001;
  + Точність методу пошуку: 0.01
  + Схема обчислення похідних: Центральна

Результати:

Точка: [0.99980985 0.99903067], кількість обчислень значення цільової функції: 239

Для ДСК-Пауелла:

* + Величина кроку h при обчисленні похідних: 0.0001;
  + Точність одновимірного пошуку: 0.01;
  + Значення параметру в алгоритмі Свенна: 0.001;
  + Точність методу пошуку: 0.01
  + Схема обчислення похідних: Центральна

Результати:

Точка: [1.00414888 1.00454431], кількість обчислень значення цільової функції: 953

Для дослідження впливу методу одновимірного пошуку на збіжність партан-методу було обрано наступні параметри:

* + Величина кроку h при обчисленні похідних: 0.01;
  + Точність одновимірного пошуку: 0.0001;
  + Значення параметру в алгоритмі Свенна: 0.01;
  + Точність методу пошуку: 0.0001
  + Схема обчислення похідних: Центральна

Для золотого перетину:

Результати:

Точка: [1.00000009 1.00000008], кількість обчислень значення цільової функції: 265

Для ДСК-Пауелла:

Результати:

Точка: [1.00075212 1.00094738], кількість обчислень значення цільової функції: 1632

Метод золотого перетину виявився більш ефективним та буде застосовуватися у подальшому дослідженні.

### Точність методу одновимірного пошуку

Метод Коші було досліджено на наступній наступних параметрах:

* + Значення параметру в алгоритмі Свенна: 0.01;
  + оптимальні параметри, знайдені у попередніх розділах
  + та на наступній точності методу одновимірного пошуку:

odnom = [0.5, 0.1, 0.05, 0.005, 0.001, 0.0005, 0.0001, 0.00001, 0.000001]

Були отримані наступні результати:

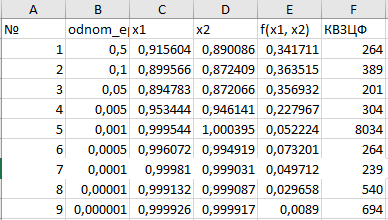


Рис 2.2.5.1 – Точність одновимірного пошуку для метода Коші

За результатами дослідження було обрано odnom = 0.0001

Партан-метод було досліджено на наступній наступних параметрах:

* + Значення параметру в алгоритмі Свенна: 0.01;
  + оптимальні параметри, знайдені у попередніх розділах
  + та на наступній точності методу одновимірного пошуку:

odnom = [0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005,0.0001, 0.00001, 0.000001, 0.0000001]

Були отримані наступні результати:

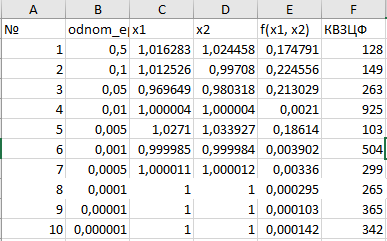


Рис 2.2.5.2 – Точність одновимірного пошуку для партан-метода

За результатами дослідження було обрано odnom = 0.0001

### Значення параметру в алгоритмі Свенна

Для дослідження впливу значення параметру в алгоритмі Свенна на збіжність методу Коші були обрані оптимальні параметри, знайдені у попередніх розділах.

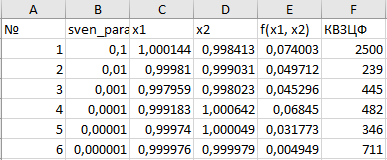


Рис 2.2.6.1 – Значення параметру в алгоритмі Свенна для методу Коші

За результатами дослідження було обрано параметр dx = 0.01

Для дослідження впливу значення параметру в алгоритмі Свенна на збіжність партан-методу були обрані оптимальні параметри, знайдені у попередніх розділах.

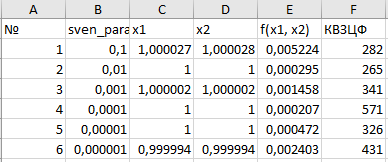


Рис 2.2.6.2 – Значення параметру в алгоритмі Свенна для партан-методу

За результатами дослідження було обрано параметр dx = 0.01

### Вигляд критерія закінчення

Для дослідження впливу критерія закінчення на збіжність методу Коші були обрані оптимальні параметри, знайдені у попередніх розділах.

Для критерію :

Точка: [0.99980985 0.99903067], кількість обчислень значення цільової функції: 239

* Для критерію :

Точка: [1 1], кількість обчислень значення цільової функції: 2251

Для дослідження впливу критерія закінчення на збіжність партан-методу були обрані оптимальні параметри, знайдені у попередніх розділах.

Для критерію :

Точка: [1.00000009 1.00000008], кількість обчислень значення цільової функції: 265

* Для критерію :

Точка: [1.00000009 1.00000008], кількість обчислень значення цільової функції: 255

Отже, критерій виявися неефективним у знаходженні мінімуму цільової функції для методу Коші. Це пов’язано з тим, що величина норми градієнту мало змінюється, що в свою чергу призводить до збільшення кількості обчислень значення цільової функції та похибки методу Коші. Для партан-методу даний критерій дав кращий результат в силу вдало підібраних параметрів.

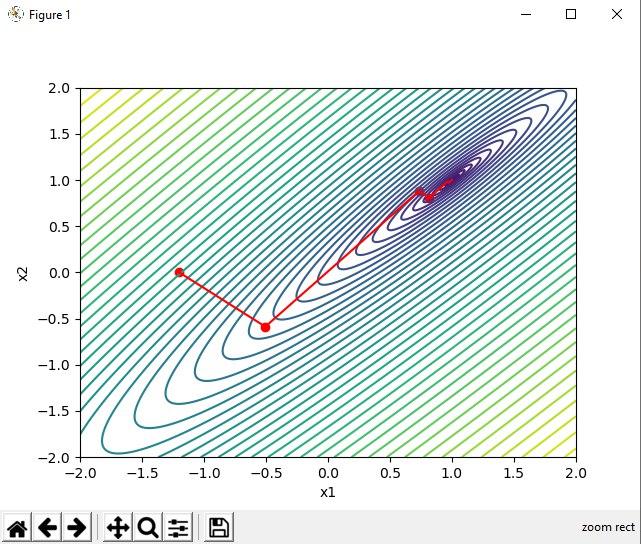


Рис 2.2.7.1 – Траєкторія руху метода Коші при найкращих параметрах

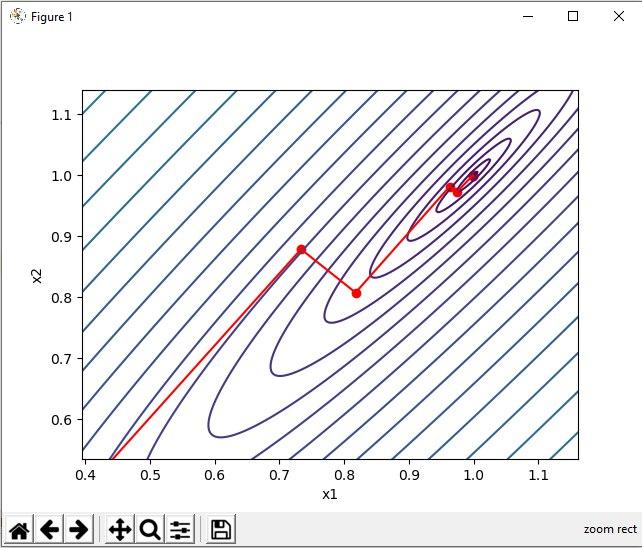


Рис 2.2.7.2 – Траєкторія руху метода Коші в околі точки мінімума

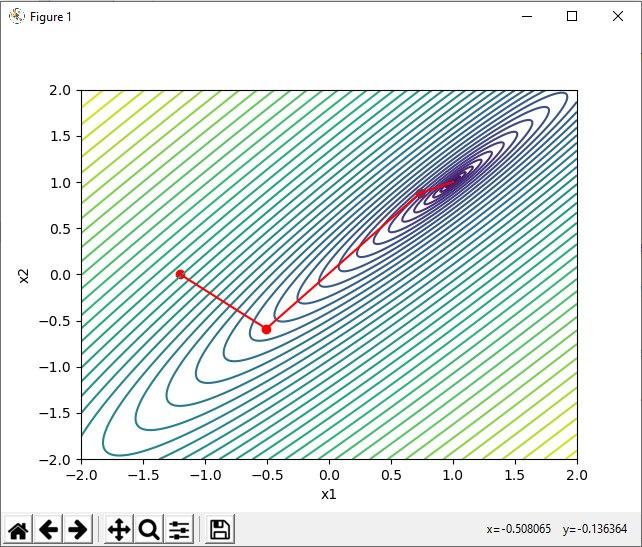


Рис 2.2.7.3 – Траєкторія руху партан-метода при найкращих параметрах

# УМОВНА ОПТИМІЗАЦІЯ

## Теоретичні відомості

Методи штрафних функцій використовуються для вирішення задач нелінійного програмування.

Загальна задача нелінійного програмування:

Припускається, що початкова точка – відома і будується послідовність - стаціонарних точок штрафних функцій.

Штрафна функція – це цільова функція задачі безумовної оптимізації, з її допомогою задача умовної оптимізації зводиться до послідовності задач безумовної оптимізації.

Методи штрафних функцій можна розділити на 2 класи:

* Параметричні методи – тут штрафний параметр грає важливу роль. Ці методи діляться в свою чергу на 2 підкласи:
* Методи внутрішньої точки – тут точки постійно знаходяться всередині допустимої області, для чого будується спеціальний «бар’єр», вздовж границі допустимої області.
* Методи зовнішньої точки – які і розглядаються у даній курсовій роботі. Тут точки не обов’язково повинні знаходитися всередині допустимої області, але в границі, усе одно, метод вийде на межу допустимої області. Штраф будується так, щоб надати точкам з допустимої області перевагу над точками з-поза допустимої області.
* Непараметричні методи – в них штрафний параметр виступає в ролі вагового коефіцієнта.

Існує декілька видів штрафних функцій. Їх вигляд та правила за якими перераховуються штрафні параметри визначають конкретний метод.

В загальному випадку, штрафна функція визначається наступним виразом:

де – штраф, функція від штрафного параметра R і обмежень.

В даній курсовій роботі було використано штрафну функція зі штрафом вигляду квадрату зрізки:

*,*

В цьому методі R на кожному кроці зростає і прямує до нескінченності.

Алгоритм методу штрафних функцій

Задано та похибка

1. Побудувати штрафну функцію
2. Знайти мінімізацією функції
3. Перевірити виконання критерію закінчення. Якщо виконується, то пошук закінчено. Інакше - до пункту 4
4. Обчислити . Перейти до пункту 2.

## Функція Розенброка

Для дослідження збіжності партан-методу на випуклій/невипуклій області були обрані наступні параметри:

* + Величина кроку h при обчисленні похідних: 0.01;
  + Метод одновимірного пошуку: золотий перетин;
  + Точність одновимірного пошуку: 0.01;
  + Точність методу пошуку: 0.0001
  + Значення параметру в алгоритмі Свенна: 0.001
  + Схема обчислення похідних: Центральна
  + Критерій закінчення: ;
  + Початкове значення штрафного параметру R=1, збільшення параметру R в 10 разів для кожної наступної ЗБО.

Дані параметри були взяті з дослідження партан-методу для безумовної задачі оптимізації. Деякі з них були змінені внаслідок дослідження партан-методу для задачі умовної оптимізації

В якості випуклої області було обрано коло радіусу 1 та з центром в точці (0, 0).

Початкова точка [-1.8, 0]. Результати:

* Ітерація №1

Кінцева точка ЗБО: [0.78553071 0.61772345]

Кількість обчислень функції= 215

* Ітерація №2

Кінцева точка ЗБО: [0.78553071 0.61772345]

Кількість обчислень функції= 226

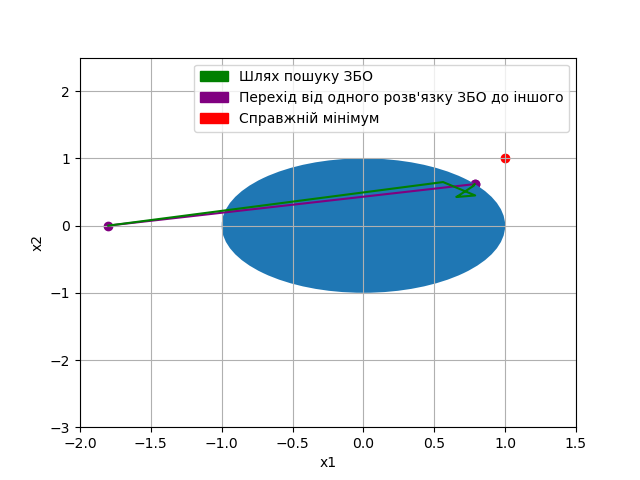


Рис 3.2.1 – Умовна оптимізація. Випукла область. Точка [-1.8, 0]

Початкова точка [1.2, 0]. Результати:

* Ітерація №1

Кінцева точка ЗБО: [0.78732995 0.61523631]

Кількість обчислень функції= 209

* Ітерація №2

Кінцева точка ЗБО: [0.78732995 0.61523631]

Кількість обчислень функції= 220

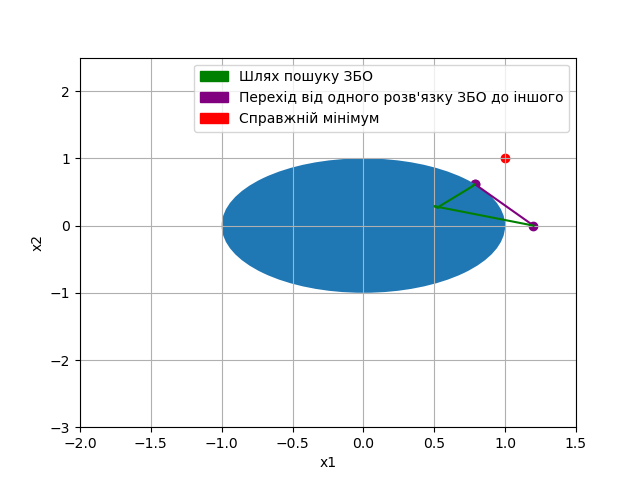


Рис 3.2.2 – Умовна оптимізація. Випукла область. Точка [1.2, 0]

В якості невипуклої області було обрано область, утворену перетином двох наступних нерівностей:

Початкова точка [1.2, 0]. Результати:

* Ітерація №1

Кінцева точка ЗБО: [0.78803448 0.61541199]

Кількість обчислень функції= 123

* Ітерація №2

Кінцева точка ЗБО: [0.78803448 0.61541199]

Кількість обчислень функції= 134

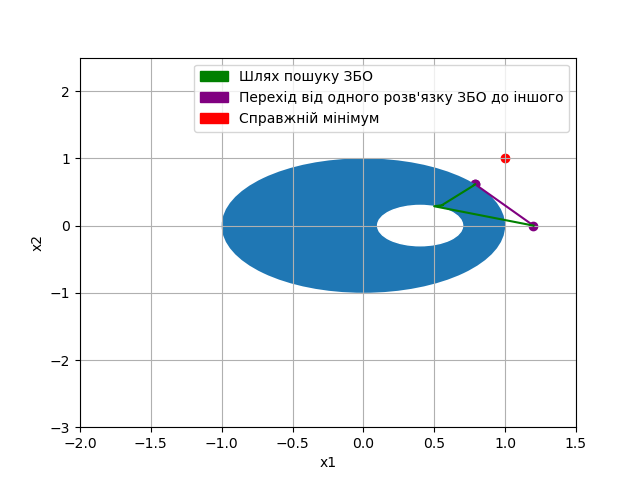


Рис 3.2.3 – Умовна оптимізація. Невипукла область. Точка [1.2, 0]

Початкова точка [-1.8, 0]. Результати:

* Ітерація №1

Кінцева точка ЗБО: [0.78910264 0.6142453 ]

Кількість обчислень функції= 251

* Ітерація №2

Кінцева точка ЗБО: [0.78910264 0.6142453 ]

Кількість обчислень функції= 262

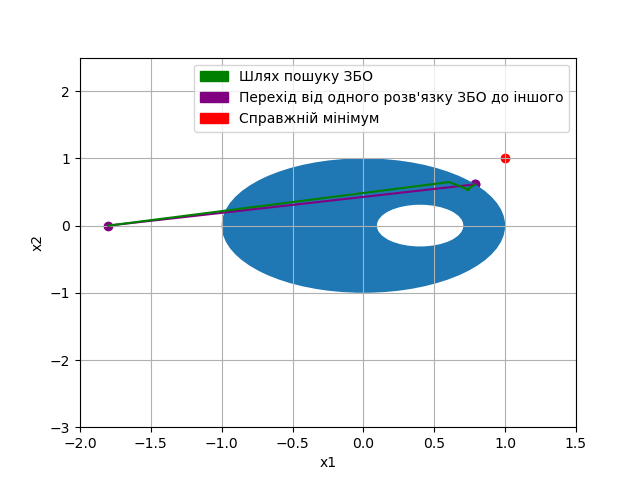


Рис 3.2.4 – Умовна оптимізація. Невипукла область. Точка [-1.8, 0]

# ВИСНОВОК

В ході виконання курсової роботи було досліджено метод Коші та партан-метод найшвидшого спуску для функції Розенброка та кореневої функції. Найменшу кількість обчислень значення цільової функції та найкращу точність було отримано на наступних параметрах:

* Функція Розенброка

|  |  |  |
| --- | --- | --- |
| Метод: | Коші | Партан |
| Точність методу: | 0.00001 | 0.00001 |
| Схема обчислення похідних: | права | центральна |
| Величина кроку h: | 0.00005 | 0.001 |
| Спосіб обчислення кроку: | оптимальний | оптимальний |
| Вид методу одновимірного пошуку: | золотий перетин | золотий перетин |
| Точність методу одновимірного пошуку | 0.0001 | 0.0005 |
| Значення параметру Свенна: | 0.001 | 0.001 |
| Вигляд критерію закінчення: |  |  |
| Точка: | [1.005583, 1.013296] | [0.997702, 0.994915] |
| КОЗЦФ | 495 | 297 |

* Коренева функція

|  |  |  |
| --- | --- | --- |
| Метод: | Коші | Партан |
| Точність методу: | 0.001 | 0.0001 |
| Схема обчислення похідних: | центральна | центральна |
| Величина кроку h: | 0.0001 | 0.01 |
| Спосіб обчислення кроку: | оптимальний | оптимальний |
| Вид методу одновимірного пошуку: | золотий перетин | золотий перетин |
| Точність методу одновимірного пошуку | 0.0001 | 0.0001 |
| Значення параметру Свенна: | 0.01 | 0.01 |
| Вигляд критерію закінчення: |  |  |
| Точка: | [0.99980985,0.99903067] | [1.00000009,1.00000008] |
| КОЗЦФ | 239 | 255 |

Для функції Розенброка партан-метод виявився кращим через те, що йому легше було проходити «яр» даної функції за рахунок кроку у напрямку xk-xk-2, в той час як метод Коші йшов по стінкам «яру» функції Розенброка.

Для кореневої функції метод Коші виявився більш ефективним. Даний результат можна пояснити тим, що було вдало вибрано параметри для методу Коші, що дозволило даному методу пройти вздовж ліній рівня кореневої функції (рис 2.2.7.1, рис 2.2.7.2). Хоча й варто зазначити, що партан-метод дав набагато кращу точність, причому обчисливши лише на 16 значення цільової функції більше, що є також непоганим результатом.

Для умовної оптимізації було обрано функцію Розенброка та партан-метод. Застосовувались найкращі параметри, що були знайдені для задачі безумовної оптимізації, а саме за використання параметрів:

* + Величина кроку h при обчисленні похідних: 0.01;
  + Метод одновимірного пошуку: золотий перетин;
  + Точність одновимірного пошуку: 0.01;
  + Точність методу пошуку: 0.0001
  + Значення параметру в алгоритмі Свенна: 0.001
  + Схема обчислення похідних: Центральна
  + Критерій закінчення: ;
  + Початкове значення штрафного параметру R=1, збільшення параметру R в 10 разів для кожної наступної ЗБО.
  + Початкова точка: [1.2, 0],

було отримано наступний результат:

Точка: [0.78803448 0.61541199]

Кількість обчислень функції= 134

Даний результат можна пояснити тим, що найбільша кількість обчислень цільової функції спостерігалась саме в «ярі» функції Розенброка – околі точки (1, 1) та дослідженням уже найкращих для задачі безумовної оптимізації параметрів.

Загалом обидва методи виявились досить ефективними для пошуку мінімуму функції Розенброка та кореневої функції, навіть без застосування модифікацій. Найбільш важливими параметрами для підбору виявились точність одновимірного пошуку та параметр dx в алгоритмі Свенна – саме вони дозволяли зекономити більше всього обчислень функції.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Himmelblau D.M. “APPLIED NONLINER PROGRAMMIG”, mCgRAW-Hill Book Company, 1972;

# ДОДАТОК А (лістинг програми безумовної оптимізації)

import numpy as np

import matplotlib.pyplot as plt

counter1 = 0

def evaluate\_f(x1, x2):

global counter1

counter1 += 1

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

return (10 \* (x1 - x2) \*\* 2 + (x1 - 1) \*\* 2) \*\* (1 / 4)

def grad\_center(x, h\_epsilon):

h = h\_epsilon \* np.linalg.norm(x)

dx = (evaluate\_f(x[0] + h, x[1]) - evaluate\_f(x[0] - h, x[1])) / (2 \* h)

dy = (evaluate\_f(x[0], x[1] + h) - evaluate\_f(x[0], x[1] - h)) / (2 \* h)

return np.array([dx, dy])

def grad\_right(x, h\_epsilon):

h = h\_epsilon \* np.linalg.norm(x)

dx = (evaluate\_f(x[0] + h, x[1]) - evaluate\_f(x[0], x[1])) / h

dy = (evaluate\_f(x[0], x[1] + h) - evaluate\_f(x[0], x[1])) / h

return np.array([dx, dy])

def grad\_left(x, h\_epsilon):

h = h\_epsilon \* np.linalg.norm(x)

dx = (evaluate\_f(x[0], x[1]) - evaluate\_f(x[0] - h, x[1])) / h

dy = (evaluate\_f(x[0], x[1]) - evaluate\_f(x[0], x[1] - h)) / h

return np.array([dx, dy])

def gold(start, svenn, direction, epsilon\_la):

x0 = start[0]

y0 = start[1]

x = direction[0]

y = direction[1]

current\_interval = svenn

L = current\_interval[1] - current\_interval[0]

la\_1 = current\_interval[0] + 0.382 \* L

la\_2 = current\_interval[0] + 0.618 \* L

f\_la1 = evaluate\_f(x0 + (la\_1 \* x), y0 + (la\_1 \* y))

f\_la2 = evaluate\_f(x0 + (la\_2 \* x), y0 + (la\_2 \* y))

while (L > epsilon\_la):

if (f\_la1 < f\_la2):

current\_interval = [current\_interval[0], la\_2]

f\_la2 = f\_la1

L = current\_interval[1] - current\_interval[0]

la\_1 = current\_interval[0] + 0.382 \* L

la\_2 = current\_interval[0] + 0.618 \* L

f\_la1 = evaluate\_f(x0 + (la\_1 \* x), y0 + (la\_1 \* y))

elif (f\_la1 > f\_la2):

current\_interval = [la\_1, current\_interval[1]]

f\_la1 = f\_la2

L = current\_interval[1] - current\_interval[0]

la\_1 = current\_interval[0] + 0.382 \* L

la\_2 = current\_interval[0] + 0.618 \* L

f\_la2 = evaluate\_f(x0 + (la\_2 \* x), y0 + (la\_2 \* y))

return (current\_interval[0] + current\_interval[1]) / 2

def svenn\_la2(direction, start, dx, case, la0=0):

x0 = start[0]

y0 = start[1]

x = direction[0]

y = direction[1]

nX = np.linalg.norm(start)

f0 = evaluate\_f(x0 + ((la0) \* x), y0 + ((la0) \* y))

values\_list = [f0]

la\_list = [la0]

f1\_l = evaluate\_f(x0 + ((la0 - dx) \* x), y0 + ((la0 - dx) \* y))

f1\_r = evaluate\_f(x0 + ((la0 + dx) \* x), y0 + ((la0 + dx) \* y))

if f1\_l > f0 and f0 > f1\_r:

determinator = 1

values\_list.append(f1\_r)

la\_list.append(la0 + dx)

elif f1\_l < f0 and f0 < f1\_r:

determinator = -1

values\_list.append(f1\_l)

la\_list.append(la0 - dx)

elif f1\_l > f0 and f0 < f1\_r:

if case == 1:

return [la0 - dx, la0 + dx]

else:

return [la0 - dx, la0, la0 + dx]

i = 1

while (values\_list[i] < values\_list[i - 1]):

la\_i = la\_list[i] + determinator \* (2 \*\* i) \* dx

la\_list.append(la\_i)

values\_list.append(evaluate\_f(x0 + ((la\_i) \* x), y0 + ((la\_i) \* y)))

i += 1

last4 = [la\_list[i], (la\_list[i] + la\_list[i - 1]) / 2, la\_list[i - 1], la\_list[i - 2]]

last4\_evaluated = []

for la in last4:

last4\_evaluated.append(evaluate\_f(x0 + ((la) \* x), y0 + ((la) \* y)))

ind = last4\_evaluated.index(min(last4\_evaluated))

if case == 1:

return sorted([last4[ind + 1], last4[ind - 1]])

else:

return sorted([last4[ind + 1], last4[ind], last4[ind - 1]])

def dsk\_powell(x, la, s, accuracy):

la = sorted(la)

s\_norm = 1

x1 = la[0]

x2 = la[1]

x3 = la[2]

dx = abs(x2 - x1)

f1 = evaluate\_f(x[0] + x1 \* s[0] / s\_norm, x[1] + x1 \* s[1] / s\_norm)

f2 = evaluate\_f(x[0] + x2 \* s[0] / s\_norm, x[1] + x2 \* s[1] / s\_norm)

f3 = evaluate\_f(x[0] + x3 \* s[0] / s\_norm, x[1] + x3 \* s[1] / s\_norm)

x\_dsk = x2 + (dx \* (f1 - f3)) / (2 \* (f1 - 2 \* f2 + f3))

f\_dsk = evaluate\_f(x[0] + x\_dsk \* s[0] / s\_norm, x[1] + x\_dsk \* s[1] / s\_norm)

if ((x2 - x\_dsk) < accuracy) and ((f2 - f\_dsk) < accuracy):

return x\_dsk

x\_top = x\_dsk

end = False

while not end:

lis = sorted([x1, x2, x3, x\_top])

ind = lis.index(x\_top)

if ind == 0:

lis = [lis[ind], lis[ind + 1], lis[ind + 2]]

elif ind == 3:

lis = [lis[ind - 2], lis[ind - 1], lis[ind]]

else:

lis = [lis[ind - 1], lis[ind], lis[ind + 1]]

x1 = lis[0]

x2 = lis[1]

x3 = lis[2]

f1 = evaluate\_f(x[0] + x1 \* s[0] / s\_norm, x[1] + x1 \* s[1] / s\_norm)

f2 = evaluate\_f(x[0] + x2 \* s[0] / s\_norm, x[1] + x2 \* s[1] / s\_norm)

f3 = evaluate\_f(x[0] + x3 \* s[0] / s\_norm, x[1] + x3 \* s[1] / s\_norm)

a1 = (f2 - f1) / (x2 - x1)

a2 = (1 / (x3 - x2)) \* ((f3 - f1) / (x3 - x1) - (f2 - f1) / (x2 - x1))

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

f\_x\_top = evaluate\_f(x[0] + x\_top \* s[0] / s\_norm, x[1] + x\_top \* s[1] / s\_norm)

f\_list = [f1, f2, f3, f\_x\_top]

list\_x = [x1, x2, x3, x\_top]

f\_min = min(f\_list)

inx\_min = f\_list.index(f\_min)

x\_min = list\_x[inx\_min]

if (((f\_min - f\_x\_top) < accuracy) and (x\_min - x\_top < accuracy)):

end = True

return x\_top

def plot\_rozenbrock(way):

rosenbrockfunction = lambda x, y: (1 - x) \*\* 2 + 100 \* (y - x \*\* 2) \*\* 2

X, Y = np.meshgrid(np.linspace(-2, 2., 1000), np.linspace(-1, 2, 1000))

Z = rosenbrockfunction(X, Y)

x\_coords = []

for i in range(len(way)):

x\_coords.append(way[i][0])

y\_coords = []

for i in range(len(way)):

y\_coords.append(way[i][1])

plt.contour(X, Y, Z, np.logspace(-0.5, 3.5, 20, base=10), cmap='Accent')

plt.scatter(x\_coords, y\_coords, color='red')

plt.plot(x\_coords, y\_coords, color='red')

plt.title('Rosenbrock Function: $f(x,y) = (1-x)^2+100(y-x^2)^2$')

plt.xlabel('x')

plt.ylabel('y')

plt.show()

def plot\_kornevaya(x):

start, stop, n\_values = -2, 2, 2000

x\_vals = np.linspace(start, stop, n\_values)

y\_vals = np.linspace(start, stop, n\_values)

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

Z = (10 \* (X - Y) \*\* 2 + (X - 1) \*\* 2) \*\* (1 / 4)

x\_list = []

y\_list = []

for lis in x:

x\_list.append(lis[0])

y\_list.append(lis[1])

plt.contour(X, Y, Z, levels=50)

plt.plot(x\_list, y\_list, color="red")

plt.xlabel('x1')

plt.ylabel('x2')

plt.scatter(x\_list, y\_list, color="red")

plt.show()

def Koshi1(x0, epsilon, h\_epsilon, odnom\_epsilon, dx, derivative, case=1):

x = [np.array(x0)]

end = False

k = 0

while not end:

grad = derivative(x[k], h\_epsilon)

S = -grad

svenn = svenn\_la2(S, x[k], dx, case=case)

if case == 1:

odnom\_pousk = gold(x[k], svenn, S, odnom\_epsilon)

else:

odnom\_pousk = dsk\_powell(x[k], svenn, S, odnom\_epsilon)

la\_opt = odnom\_pousk

x\_new = x[k] + la\_opt \* (S)

x.append(x\_new)

print('Koshu:', x\_new)

if (np.linalg.norm(x[k] - x[k + 1]) / np.linalg.norm(x[k]) < epsilon) \

and ((evaluate\_f(x[k][0], x[k][1]) - evaluate\_f(x[k + 1][0], x[k + 1][1])) < epsilon):

break

k += 1

return x

def Partan(x0, epsilon, h\_epsilon, odnom\_epsilon, dx, derivative, case=1):

x = [np.array(x0)]

end = False

k = 1

while not end:

grad = derivative(x[k - 1], h\_epsilon)

if k % 3 == 0:

S = x[k - 1] - x[k - 3]

else:

S = -grad

svenn = svenn\_la2(S, x[k - 1], dx, case=case)

if case == 1:

odnom\_pousk = gold(x[k - 1], svenn, S, odnom\_epsilon)

else:

odnom\_pousk = dsk\_powell(x[k - 1], svenn, S, odnom\_epsilon)

la\_opt = odnom\_pousk

x\_new = x[k - 1] + la\_opt \* (S)

x.append(x\_new)

print('Partan', x\_new)

if (np.linalg.norm(grad) < epsilon): # (np.linalg.norm(x[k-1]-x[k])/np.linalg.norm(x[k-1])<epsilon) \

# and ((evaluate\_f(x[k-1][0],x[k-1][1])-evaluate\_f(x[k][0],x[k][1]))<epsilon):

break

k += 1

return x

def analysis(epsilon, h\_epsilon, odnom\_epsoilonnn, dx, deriv):

x\_Koshu = []

x\_Partan = []

f\_Koshu = []

f\_Partan = []

counter\_Koshu = []

counter\_Partan = []

deriv\_type = deriv.\_\_name\_\_

for dxx in dx:

global counter1

counter1 = 0

x1 = Koshi1([-1.2, 0], epsilon, h\_epsilon, odnom\_epsoilonnn , dxx, grad\_center,case=1)

counter\_Koshu.append(counter1)

counter1 = 0

x2 = Partan([-1.2, 0], epsilon, h\_epsilon, odnom\_epsoilonnn, dxx, grad\_center, case=1)

counter\_Partan.append(counter1)

counter1 = 0

x\_Koshu.append(x1)

x\_Partan.append(x2)

f\_Koshu.append(evaluate\_f(x1[0], x1[1]))

f\_Partan.append(evaluate\_f(x2[0], x2[1]))

return x\_Koshu, x\_Partan, f\_Koshu, f\_Partan, counter\_Koshu, counter\_Partan, deriv\_type

# ДОДАТОК Б (лістинг програми умовної оптимізації)

import numpy as np

import matplotlib.pyplot as plt

from time import sleep

import matplotlib.patches as mpatches

counter1 = 0

global\_points = []

def evaluate\_f(x1, x2, R):

global counter1

counter1 += 1

if (x1)\*\*2 + x2\*\*2 <= 1 and (x1-0.4)\*\*2 + x2\*\*2 >= 0.09:

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

else:

return (100\*(x1\*\*2 - x2)\*\*2) + (x1 - 1)\*\*2 + R\*(((x1-0.4)\*\*2 + x2\*\*2)\*\*2)+ R\*(((x1)\*\*2 + x2\*\*2)\*\*2)

## if (x1)\*\*2 + x2\*\*2 <= 1:

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

## else:

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

def grad\_center(x, epsilon, R):

h = epsilon\*np.linalg.norm(x)

dx = (evaluate\_f(x[0]+h,x[1], R)-evaluate\_f(x[0]-h,x[1], R))/(2\*h)

dy = (evaluate\_f(x[0],x[1]+h, R)-evaluate\_f(x[0],x[1]-h, R))/(2\*h)

return np.array([dx,dy])

def grad\_right(f, x, h, R):

dx = (f(x[0]+h, x[1], R) - f(x[0], x[1], R))/h

dy = (f(x[0], x[1]+h, R) - f(x[0], x[1], R))/h

return np.array([dx, dy])

def grad\_left(f, x, h, R):

dx = (f(x[0], x[1], R) - f(x[0]-h, x[1], R))/h

dy = (f(x[0], x[1], R) - f(x[0], x[1]-h, R))/h

return np.array([dx, dy])

def gold(start, svenn, direction,epsilon\_la, R):

x0 = start[0]

y0 = start[1]

x = direction[0]

y = direction[1]

current\_interval = svenn

L = current\_interval[1]-current\_interval[0]

la\_1 = current\_interval[0]+0.382\*L

la\_2 = current\_interval[0]+0.618\*L

f\_la1 =evaluate\_f(x0+(la\_1\*x),y0+(la\_1\*y), R)

f\_la2 =evaluate\_f(x0+(la\_2\*x),y0+(la\_2\*y), R)

while(L>epsilon\_la):

if(f\_la1<f\_la2):

current\_interval = [current\_interval[0],la\_2]

f\_la2 = f\_la1

L = current\_interval[1]-current\_interval[0]

la\_1 = current\_interval[0]+0.382\*L

la\_2 = current\_interval[0]+0.618\*L

f\_la1 =evaluate\_f(x0+(la\_1\*x),y0+(la\_1\*y), R)

elif(f\_la1>f\_la2):

current\_interval = [la\_1,current\_interval[1]]

f\_la1 = f\_la2

L = current\_interval[1]-current\_interval[0]

la\_1 = current\_interval[0]+0.382\*L

la\_2 = current\_interval[0]+0.618\*L

f\_la2 =evaluate\_f(x0+(la\_2\*x),y0+(la\_2\*y), R)

return (current\_interval[0]+current\_interval[1])/2

def svenn\_la2(direction, start, dx, case, la0, R):

x0 = start[0]

y0 = start[1]

x = direction[0]

y = direction[1]

nX = np.linalg.norm(start)

f0 = evaluate\_f(x0 + ((la0) \* x), y0 + (

(la0) \* y), R)

fl = evaluate\_f(x0 + ((la0 - dx) \* x), y0 + ((la0 - dx) \* y), R)

fr = evaluate\_f(x0 + ((la0 + dx) \* x), y0 + ((la0 + dx) \* y), R)

values\_list = [f0]

la\_list = [la0]

if fl > f0 and f0 > fr:

determinator = 1

values\_list.append(fr)

la\_list.append(la0 + dx)

elif fl < f0 and f0 < fr:

determinator = -1

values\_list.append(fl)

la\_list.append(la0 - dx)

elif fl > f0 and f0 < fr:

if case == 1:

return [la0 - dx, la0 + dx]

else:

return [la0 - dx, la0, la0 + dx]

i = 1

while (values\_list[i] < values\_list[i - 1]):

la\_i = la\_list[i] + determinator \* (2 \*\* i) \* dx

la\_list.append(la\_i)

values\_list.append(evaluate\_f(x0 + ((la\_i) \* x), y0 + ((la\_i) \* y), R))

i += 1

last4 = [la\_list[i], (la\_list[i] + la\_list[i - 1]) / 2, la\_list[i - 1], la\_list[i - 2]]

last4\_evaluated = []

for la in last4:

last4\_evaluated.append(evaluate\_f(x0 + ((la) \* x), y0 + ((la) \* y), R))

ind = last4\_evaluated.index(min(last4\_evaluated))

if case == 1:

return sorted([last4[ind - 1], last4[ind + 1]])

else:

if ind == 1:

last3 = [last4[0], last4[1], last4[2]]

if ind == 2:

last3 = [last4[1], last4[2], last4[3]]

return last3

def dsk\_powell(x, la, s, accuracy, R):

la = sorted(la)

x1 = la[0]

x2 = la[1]

x3 = la[2]

dx = abs(x2 - x1)

f1 = evaluate\_f(x[0] + x1 \* s[0], x[1] + x1 \* s[1], R)

f2 = evaluate\_f(x[0] + x2 \* s[0], x[1] + x2 \* s[1], R)

f3 = evaluate\_f(x[0] + x3 \* s[0], x[1] + x3 \* s[1], R)

x\_dsk = x2 + (dx \* (f1 - f3)) / (2 \* (f1 - 2 \* f2 + f3))

f\_dsk = evaluate\_f(x[0] + x\_dsk \* s[0], x[1] + x\_dsk \* s[1], R)

if ((x2 - x\_dsk) < accuracy) and ((f2 - f\_dsk) < accuracy):

return x\_dsk

x\_top = x\_dsk

end = False

while not end:

lis = sorted([x1, x2, x3, x\_top])

ind = lis.index(x\_top)

if ind == 0:

lis = [lis[ind], lis[ind + 1], lis[ind + 2]]

elif ind == 3:

lis = [lis[ind - 2], lis[ind - 1], lis[ind]]

else:

lis = [lis[ind - 1], lis[ind], lis[ind + 1]]

x1 = lis[0]

x2 = lis[1]

x3 = lis[2]

f1 = evaluate\_f(x[0] + x1 \* s[0], x[1] + x1 \* s[1], R)

f2 = evaluate\_f(x[0] + x2 \* s[0], x[1] + x2 \* s[1], R)

f3 = evaluate\_f(x[0] + x3 \* s[0], x[1] + x3 \* s[1], R)

a1 = (f2 - f1) / (x2 - x1)

a2 = (1 / (x3 - x2)) \* ((f3 - f1) / (x3 - x1) - (f2 - f1) / (x2 - x1))

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

f\_x\_top = evaluate\_f(x[0] + x\_top \* s[0], x[1] + x\_top \* s[1], R)

f\_list = [f1, f2, f3, f\_x\_top]

list\_x = [x1, x2, x3, x\_top]

f\_min = min(f\_list)

inx\_min = f\_list.index(f\_min)

x\_min = list\_x[inx\_min]

if (1 / (x3 - x2) <= 0.0001):

end = True

if ((f\_min - f\_x\_top) < accuracy) and (x\_min - x\_top < accuracy):

end = True

return x\_top

def Koshi1(x0, epsilon, h\_epsilon, odnom\_epsilon, case, R):

x = [np.array(x0)]

end = False

k=0

while not end:

grad = grad\_center(x[k], h\_epsilon, R)

S = -grad

svenn = svenn\_la2(S, x[k], 0.001, case, 0, R)

if case == 1:

odnom\_pousk = gold(x[k], svenn, S, odnom\_epsilon, R)

else:

odnom\_pousk = dsk\_powell(x[k], svenn, S, odnom\_epsilon, R)

la\_opt = odnom\_pousk

x\_new = x[k] + la\_opt\*(S)

x.append(x\_new)

if(np.linalg.norm(x[k]-x[k+1])/np.linalg.norm(x[k])<epsilon) \

and ((evaluate\_f(x[k][0],x[k][1], R)-evaluate\_f(x[k+1][0],x[k+1][1], R))<epsilon):

break

k +=1

return x

def Partan(x0, epsilon, h\_epsilon, odnom\_epsilon, case, R=0):

x = [np.array(x0)]

end = False

k=1

while not end:

grad = grad\_center(x[k-1], h\_epsilon, R)

if k%3 == 0:

S = x[k-1] - x[k-3]

else:

S = -grad

svenn = svenn\_la2(S, x[k-1], 0.001, case, 0, R)

if case == 1:

odnom\_pousk = gold(x[k-1], svenn, S, odnom\_epsilon, R)

else:

odnom\_pousk = dsk\_powell(x[k-1], svenn, S, odnom\_epsilon, R)

la\_opt = odnom\_pousk

x\_new = x[k-1] + la\_opt\*(S)

x.append(x\_new)

if(np.linalg.norm(x[k-1]-x[k])/np.linalg.norm(x[k-1])<epsilon) \

and ((evaluate\_f(x[k-1][0],x[k-1][1], R)-evaluate\_f(x[k][0],x[k][1], R))<epsilon):

break

k +=1

return x

end = False

x\_k\_list = [[-1.8, 0]]

R = 1

k = 1

while end != True:

x\_k = x\_k\_list[-1]

x = Partan(x\_k, 0.0001, 0.01 , 0.01, 1, R)

## x = Partan(x\_k, 0.001, 0.01 , 0.0005, 1, R)

print("Ітерація №{}".format(k))

print("Кінцева точка ЗБО: ", x[-1])

print("Кількість обчислень функції=",counter1)

x\_k\_1 = x[-1]

x\_k\_list.append(x\_k\_1)

for dot in x:

global\_points.append(dot)

if ((np.linalg.norm(np.array(x\_k\_1) - np.array(x\_k)))/np.linalg.norm(x\_k) < 0.001):

end = True

break

R \*= 10

k +=1

x\_list = []

y\_list = []

for lis in x\_k\_list:

x\_list.append(lis[0])

y\_list.append(lis[1])

dots\_x = []

dots\_y = []

for lis in global\_points:

dots\_x.append(lis[0])

dots\_y.append(lis[1])

circle1 = plt.Circle((0,0),1,fill=True)

circle2 = plt.Circle((0.4,0),0.3,fill=True,color='white')

fig, ax = plt.subplots()

ax.add\_artist(circle1)

ax.add\_artist(circle2)

plt.plot(x\_list,y\_list, color = "purple")

plt.plot(dots\_x,dots\_y, color = "green")

plt.xlabel("x1")

plt.ylabel("x2")

plt.scatter(x\_list,y\_list, color = "purple")

plt.scatter([1],[1], color = "red")

green\_patch = mpatches.Patch(color='green', label='Шлях пошуку ЗБО')

purple\_patch = mpatches.Patch(color='purple', label='Перехід від одного розв\'язку ЗБО до іншого')

red\_patch = mpatches.Patch(color='red', label='Справжній мінімум')

plt.legend(handles=[green\_patch,purple\_patch,red\_patch])

plt.xlim(-2, 1.5)

plt.ylim(-3, 2.5)

plt.grid(True)

plt.show()