НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра прикладної математики

**Курсова робота**

із дисципліни «Методи оптимізації»

на тему:

ПАРТАН-МЕТОД НАЙШВИДШОГО СПУСКУ

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студент групи КМ-81 | ст. вик. Норкін Б.В. |
| *Цуркановський С.О.* |  |

Київ — 2021

ЗМІСТ

[ВСТУП 3](#_Toc73626320)

[1.1 Мета роботи 3](#_Toc73626321)

[1.2 Завдання курсової роботи 3](#_Toc73626322)

[ОСНОВНА ЧАСТИНА 5](#_Toc73626323)

[2.1 Постановка задачі 5](#_Toc73626324)

[2.2 Теоретична частина 6](#_Toc73626325)

[2.3 Практична частина 7](#_Toc73626326)

[ВИСНОВКИ 16](#_Toc73626327)

[СПИСОК ЛІТЕРАТУРИ 17](#_Toc73626328)

[ДОДАТОК. Код програми 18](#_Toc73626329)

# ВСТУП

## 1.1 Мета роботи

Метою виконання курсової роботи з кредитного модуля «Дослідження операцій» є поглиблення, узагальнення та закріплення знань, отриманих студентом під час навчання, їх застосування у комплексному вирішенні конкретної проблеми та формування у студентів здібностей:

* аналізувати сучасні чисельні методи оптимізації для розв'язування широкого спектру задач нелінійного програмування;
* аналізувати вимоги до чисельних методів оптимізації для розв'язування конкретної задачі нелінійного програмування;
* виконувати вибір чисельного метода оптимізації для розв'язування конкретної задачі нелінійного програмування;
* реалізовувати алгоритм обраного методу;
* виконувати дослідження отриманих результатів. Метою курсової роботи є дослідження партан-методу найшвидшого спуску при вирішенні конкретних завдань.

1.2 Завдання курсової роботи

Основними завданнями курсової роботи є формування умінь і навичок при проведенні самостійного навчально-наукового дослідження. Згідно з вимогами програми навчальної дисципліни студенти в результаті виконання курсової роботи мають продемонструвати такі результати навчання:

знання:

* ролі методів оптимізації в прикладних науках і розв’язанні практичних задач;
* основних особливостей методів оптимізації для задач нелінійного програмування;
* умов використання методів залежно від особливостей задачі;
* можливостей адаптації методів при вирішенні конкретних практичних задач;

уміння:

* аналізувати поставлену задачу оптимізації;
* обрати найбільш ефективний для її розв’язання метод;

реалізувати обраний метод та одержати практичні результати.

# ОСНОВНА ЧАСТИНА

2.1 Постановка задачі

Задачею даної курсової роботи є дослідження збіжності методу найшвидшого спуску та партан-методу найшвидшого спуску при мінімізації степеневої функції в залежності від:

1. Величини кроку h при обчисленні похідних.
2. Схеми обчислення похідних.
3. Виду методу одновимірного пошуку (ДСК-Пауелла або Золотого перетину).
4. Точності методу одновимірного пошуку.
5. Значення параметру в алгоритмі Свена.
6. Вигляду критерію закінчення.
7. Порівняти з методом найшвидшого спуску.

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

Наша цільова функція має вигляд:

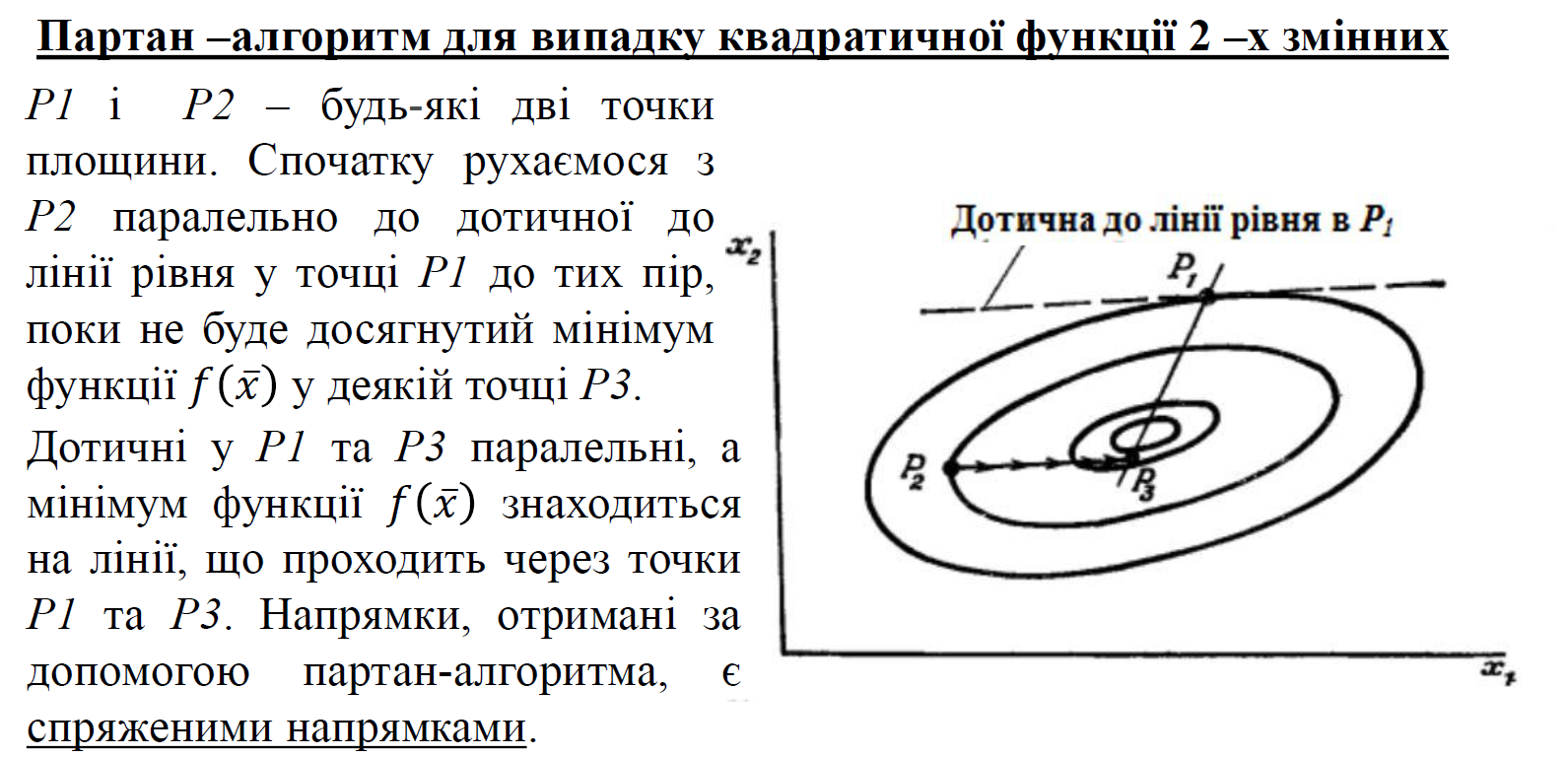
2.2 Теоретична частина

2.2.1 Метод найшвидшого спуску

Метод найшвидшого спуску є методом першого порядку, тобто використовує значення функцій і їх перших похідних. Суть методу полягає у тому, щоб ітераційно наблизитись до точки мінімуму по напряму значення антиградієнту.

2.2.2 Партан-методи

Партан-методи (від англ. parallel tangents) або методи паралельних дотичних полягають у побудові спряженого напрямку, на якому, за теоремою про паралельні дотичні, повинен знаходитися очікуваній результат.

Теорема про паралельні дотичні:

2.2.3 Метод зовнішньої точки

Метод зовнішньої точки полягає у активації обмежень якщо точка не знаходиться у допустимій області. Кожне обмеження, що не виконується, стає активованим. Цільова функція у даному методі має вигляд:

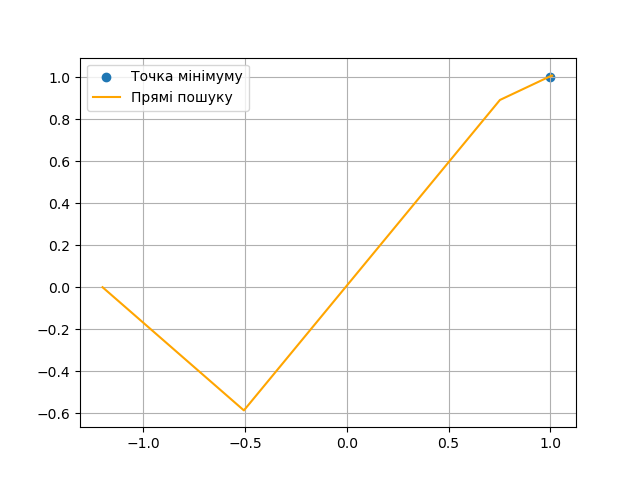
, де початкова цільова функція, а – обмеження.

2.3 Практична частина

2.3.1 Початкові характеристики

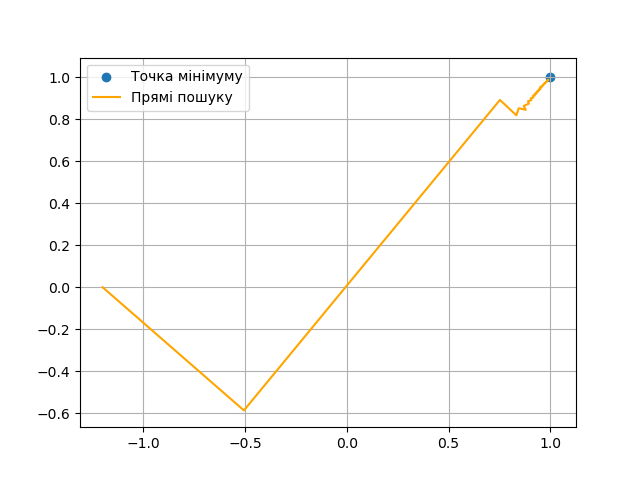
Оберемо початкові характеристики для тестування, на основі яких уже проведено перший тест:

1. Величина кроку h при обчисленні похідних: 0.02 см
2. Схема обчислення похідних: метод Рунге-Ромберга-Річардсона
3. Виду методу одновимірного пошуку: метод Золотого перетину.
4. Точності методу одновимірного пошуку: 0.001
5. Значення параметру в алгоритмі Свена: 0.001
6. Вигляду критерію закінчення:

Отримана точка x = [1.0084, 1.0087]. У процесі пошуку цільова функція була викликана 422 рази. Графік пошуку вказаний на рис. 1.

*Рис. 1.* Графік пошуку партан МНС з початковими характеристиками

По графіку (рис. 1) видно, що було побудовано 3 напрямки, останній з яких є спряжений і вказував приблизно на точку мінімуму. Також велика кількість викликів функції вказує на те, що пошук зробив досить багато невеликих кроків, які майже не мали впливу на отриманий результат.

Для порівняння, чистий метод найшвидшого спуску з тими самими параметрами повертає точку x = [0.9890, 0.9881], що є гіршим результатом.

*Рис. 2.* Графік пошуку простого МНС з початковими характеристиками

Число викликів функції при МНС становить 4326, що більш ніж в десять разів більше при гіршій точності. По графіку на рис. 2 видно, що досягнувши 3 ітерації МНС почав зігзагами наближатися до точки мінімуму, що і є причиною такої великої кількості виклику функції. Дана проблема може бути викликаною тим, що напрям антиградієнту не вказує на точку мінімуму, а саме значення функції в даному околі є дуже малим і збиває методи одновимірного пошуку.

2.3.2 Аналіз ефективності методу в залежності від величини кроку h при обчисленні похідних

Розглянемо точність та кількість виклику функції за таких значеннь кроку h: 0.02, 0.01, 0.001, 0.0001, 0.00001.

|  |  |  |  |
| --- | --- | --- | --- |
| h step | counter | x1 | x2 |
| 0.02 | 422 | 1.0084 | 1.0087 |
| 0.01 | 444 | 1.0121 | 1.0127 |
| 0.001 | 356 | 0.9925 | 0.9910 |
| 0.0001 | 356 | 0.9918 | 0.9904 |
| 1e-05 | 356 | 0.9918 | 0.9904 |

Так як останні 3 значення мають однакову ефективність візьмемо значення кроку 0.001, адже серед них воно має найкращу точність.

2.3.3 Аналіз ефективності методу в залежності від схеми обчислення похідних

Розглянемо такі методи обчислення похідних: метод Рунге-Ромберга-Річардсона, метод центральної різниці, метод передньої різниці, метод задньої різниці.

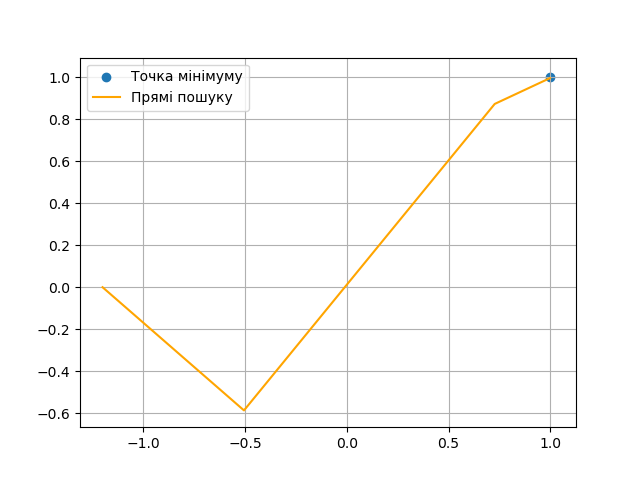
|  |  |  |  |
| --- | --- | --- | --- |
| diff method | counter | x1 | x2 |
| RRR\_diff | 356 | 0.9925 | 0.9910 |
| central\_diff | 337 | 0.9967 | 0.9961 |
| forward\_diff | 345 | 0.9975 | 0.9970 |
| backward\_diff | 345 | 0.9975 | 0.9970 |

Бачимо, що метод центральної різниці має кращу ефективність за некритичного зменшення точності, тож оберемо його.

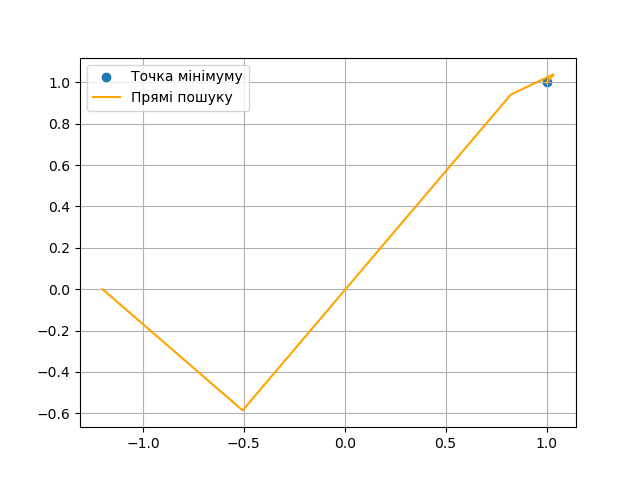
2.3.4 Аналіз ефективності методу в залежності від виду методу одновимірного пошуку

Розглянемо такі методи одновимірного пошуку: ДСК-Пауелла та Золотого перетину.

|  |  |  |  |
| --- | --- | --- | --- |
|  | counter | x1 | x2 |
| Golden Ratio | 337 | 0.9967 | 0.9961 |
| DSC-Powell | 298 | 0.9995 | 0.9990 |



*Рис. 3.* Графік пошуку за методу Золотого перетину

*Рис. 4.* Графік пошуку за методу ДСК-Пауелла

Хоча, методу ДСК-Пауелла було потрібно більше кроків, кількість викликів функції помітно менша, а точність помітно краща за метод Золотого перетину, тож оберемо метод ДСК-Пауелла.

2.3.5 Аналіз ефективності методу в залежності від точності методу одновимірного пошуку

Розглянемо кількість виклику функції за таких значень точності методу одновимірного пошуку: 0.5, 0.1, 0.01, 0.001, 0.0001.

|  |  |  |  |
| --- | --- | --- | --- |
| epsilon | counter | x1 | x2 |
| 0.5 | 628 | 0.9885 | 0.9877 |
| 0.1 | 361 | 0.9606 | 0.9587 |
| 0.01 | 283 | 1.0025 | 1.0027 |
| 0.001 | 298 | 0.9995 | 0.9990 |
| 0.0001 | 224 | 1.0000 | 1.0001 |

Візьмемо значення точності методу одновимірного пошуку 0.0001, так як за цього значення ефективність і точність функції є кращими серед усіх варіантів.

2.3.6 Аналіз ефективності методу в залежності від значення параметру в алгоритмі Свена.

Розглянемо точність та кількість виклику функції за таких значень параметру кроку в алгоритмі Свена: 0.5, 0.1, 0.01, 0.001, 0.0001.

|  |  |  |  |
| --- | --- | --- | --- |
| Sven step | counter | x1 | x2 |
| 0.5 | 149 | 1.0000 | 1.0001 |
| 0.1 | 152 | 1.0014 | 1.0015 |
| 0.01 | 188 | 1.0019 | 1.0020 |
| 0.001 | 224 | 1.0000 | 1.0001 |
| 0.0001 | 251 | 1.0000 | 1.0000 |

Візьмемо значення параметру кроку в алгоритмі Свена 0.5, так як за цього значення ефективність і точність функції є кращими серед усіх варіантів.

2.3.7 Аналіз ефективності методу в залежності від вигляду критерію закінчення.

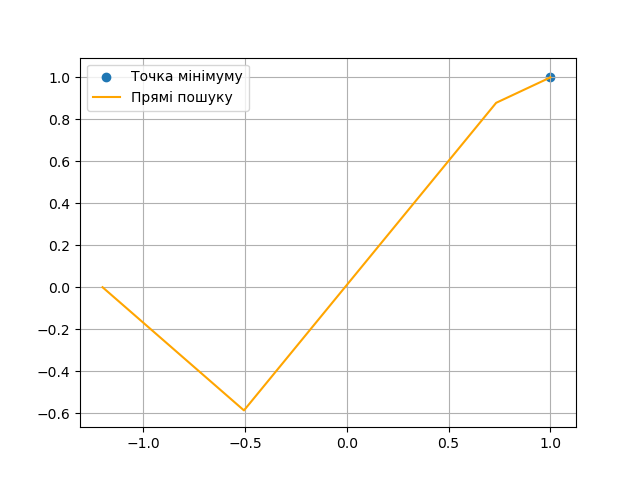
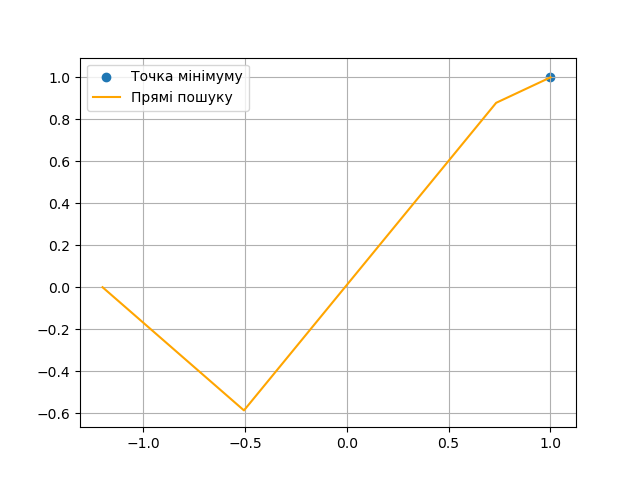
Розглянемо такі критерії закінчення для нашого методу:

1. , наш початковий критерій
2. , градієнтний критерій

|  |  |  |  |
| --- | --- | --- | --- |
|  | counter | x1 | x2 |
| Module | 149 | 1.0000 | 1.0001 |
| Gradient | 124 | 1.0001 | 1.0000 |

По даним з таблиці видно, що за градієнтного критерію закінчення, ефективність пошуку краща за такої самої точності.

Подивимось на графіки:

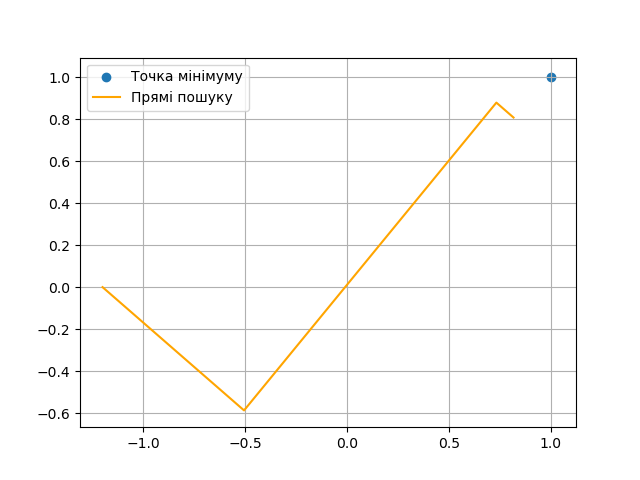
*Рис. 5.* Графік пошуку за початкового критерію закінчення

*Рис. 6.* Графік пошуку за градієнтного критерію закінчення

Обидва графіки (рис. 5 і 6) виглядають ідентично. Це означає, що для досягнення бажаної точності початковому критерію потрібно більше непотрібних кроків для закінчення обчислень. Тож оберемо градієнтний критерій закінчення.

2.3.8 Порівняння з стандартним МНС за нових параметрів.

|  |  |  |  |
| --- | --- | --- | --- |
|  | counter | x1 | x2 |
| Partan | 124 | 1.0001 | 1.0000 |
| No partan | 116 | 0.8174 | 0.8076 |

Бачимо, що не дивлячись на те, що без партану щоб досягти необхідної точності з нашим критерієм закінчення було потрібно менше обчислень, точність отриманого результату виявилась набагато гіршою ніж партан методу та за початкових параметрів.

*Рис. 7.* Графік пошуку Стандартного МНС за нових параметрів

По графіку (рис. 7) бачимо, що за нового критерію закінчення стандартний МНС зупиняє обчислення, коли досягає точки, в якій за початкового критерію він починав робити зігзаги.

2.3.9 Умовна оптимізація.

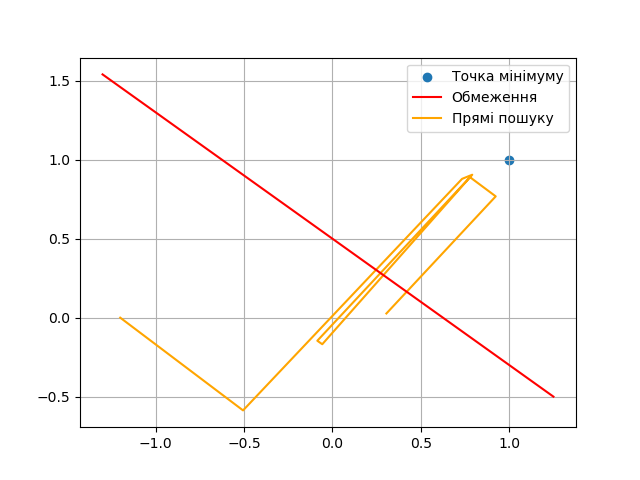
Введемо обмеження для нашої функції

Для умовної оптимізації будемо використовувати метод зовнішньої точки. Тоді наша цільова функція набуде вигляду:

, де R = 0 якщо обмеження виконується.

Розглянемо операцію оптимізації за таких значень R: 1, 10, 100, 1000.

|  |  |  |  |
| --- | --- | --- | --- |
| R | counter | x1 | x2 |
| 1 | 656 | 0.2525 | 0.1708 |
| 10 | 358 | 0.3053 | 0.0266 |
| 100 | 6759 | 0.2343 | 0.1904 |
| 1000 | 214 | -0.0828 | -0.2811 |



*Рис. 8.* Графік пошуку умовної оптимізації за R = 10

По графіку (рис. 8) видно, що коли пошук виходить за допустиму область, то метод зовнішньої точки повертає його назад.

# ВИСНОВКИ

За результатами проведених досліджень було виявлено, що найкращими параметрами для нашої задачі, які дають найточніший результат за найменшу кількість викликів цільової функції є:

1. Величина кроку h при обчисленні похідних: 0.001
2. Схема обчислення похідних: метод центральної різниці
3. Виду методу одновимірного пошуку: метод ДСК-Пауелла.
4. Точності методу одновимірного пошуку: 0.0001
5. Значення параметру в алгоритмі Свена: 0.5
6. Вигляду критерію закінчення:

За даних параметрів кількість викликів функції становить усього 124, а отримана точка x = [1.0001, 1.0000], наближена до точки мінімуму x = [1, 1].

Також було виявлено, що стандартний МНС програє партан методу і по точності і по кількості викликів функції. За початкового критерію закінчення навіть з кількістю викликів функції більш ніж у 10 разів більше за партан методом не біла досягнута точність партан методу. За градієнтного критерію закінчення метод мав меншу кількість викликів функції за партан, але зупинився набагато далі від точки мінімуму ніж партан метод.

Було також проведено тестування методу на задачах умовної оптимізації, використовуючи метод зовнішньої точки. У результаті тесту було виявлено, що комбінація цих методів не здатна з високою точністю знайти точку мінімуму в випуклій області, але здатна залишатися в допустимій області.

# СПИСОК ЛІТЕРАТУРИ

1. Хіммельблау Д.М. “Прикладное нелинейное программирование”: Москва: Видавництво «Мир». 1975
2. Гілл Ф., Мюррей М., Райт М. “Практическая оптимизация”: Москва: Видавництво «Мир». 1985

# ДОДАТОК. Код програми

from math import sqrt

import numpy as np

import numdifftools as nd

import matplotlib.pyplot as plt

f\_call\_counter = 0

def f(x1, x2):

global f\_call\_counter

f\_call\_counter += 1

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

def create\_dir\_func(x, direction = [1, 0], func = f):

def dir\_func(h):

return func(x[0] + direction[0]\*h, x[1] + direction[1]\*h)

return dir\_func

def module(x1, x2):

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

def reset(x\_arr, y\_arr):

global f\_call\_counter

f\_call\_counter = 0

x\_arr, y\_arr = [-1.2], [0]

return x\_arr, y\_arr

def finish\_criteria\_grad(dirs, eps, x\_arr, y\_arr):

return (module(dirs[0], dirs[1]) >= eps)

def finish\_criteria\_delta(dirs, eps, x\_arr, y\_arr):

if len(x\_arr) <= 1:

return True

func\_delta\_crit = ((abs(f(x\_arr[-1], y\_arr[-1]) - f(x\_arr[-2], y\_arr[-2]))) > eps)

x\_lenght\_delta\_crit = ( ((module(x\_arr[-1] - x\_arr[-2], y\_arr[-1] - y\_arr[-2])) / (module(x\_arr[-2], y\_arr[-2]))) > eps)

return (func\_delta\_crit|x\_lenght\_delta\_crit)

def Golden\_ratio(intr, eps, x, dirs):

func = create\_dir\_func(x, dirs)

a = min(intr)

b = max(intr)

x\_arr = []

func\_arr = []

x1 = a + 0.382\*(abs(a - b))

x2 = a + 0.618\*(abs(a - b))

x\_arr.extend([a, x1, x2, b])

func\_arr.extend([func(x) for x in x\_arr])

center\_index = func\_arr.index(min(func\_arr))

while (module(((x[0] + b\*dirs[0]) - (x[0] + a\*dirs[0])),((x[1] + b\*dirs[1])-(x[1] + a\*dirs[1]))))>eps:

x\_arr = []

func\_arr = []

x1 = a + 0.382\*(abs(a - b))

x2 = a + 0.618\*(abs(a - b))

x\_arr.extend([a, x1, x2, b])

func\_arr.extend([func(x) for x in x\_arr])

center\_index = func\_arr.index(min(func\_arr))

try:

a = x\_arr[center\_index - 1]

b = x\_arr[center\_index + 1]

except:

return (a + b)/2

return x\_arr[center\_index]

def dsc\_Powell(intr, eps, x, dirs):

a = min(intr)

b = max(intr)

func = create\_dir\_func(x, dirs)

xmin = (a + b) / 2

f1 = func(a)

f2 = func(xmin)

f3 = func(b)

xApprox = xmin + ((b - xmin) \* (f1 - f3)) / (2 \* (f1 - 2 \* f2 + f3))

while (abs(xmin - xApprox) >= eps or (abs(func(xmin) - func(xApprox)))>= eps):

if xApprox < xmin:

b = xmin

else:

a = xmin

xmin = xApprox

funcRes = [

func(a),

func(xmin),

func(b),

]

a1 = (funcRes[1] - funcRes[0]) / (xmin - a)

a2 = ((funcRes[2] - funcRes[0]) / (b - a) - a1) / (b - xmin)

xApprox = (a + xmin) / 2 - a1 / (2 \* a2)

return xmin

def Sven(x0, delta\_lambd, func):

x\_arr = [x0]

lambd\_arr = [func(x0)]

k = 0

coef = 1

if (func(x0)>func(x0 + delta\_lambd)):

coef = 1

x0 += delta\_lambd

x\_arr.append(x0)

lambd\_arr.append(func(x0))

elif (func(x0)>func((x0 - delta\_lambd))):

coef = -1

x0 -= delta\_lambd

x\_arr.append(x0)

lambd\_arr.append(func(x0))

else:

return [-delta\_lambd, 0, delta\_lambd]

k += 1

while (func((coef\*delta\_lambd\*(2\*\*k)))<(func(coef\*delta\_lambd\*(2\*\*(k-1))))):

x0 += (coef\*delta\_lambd\*(2\*\*k))

k += 1

x\_arr.append(x0)

lambd\_arr.append(func(x0))

x\_arr.pop(-1)

lambd\_arr.pop(-1)

x\_arr.append((x0 + (coef\*delta\_lambd\*(2\*\*(k-1))))/2)

lambd\_arr.append(func((x0 + (coef\*delta\_lambd\*(2\*\*(k-1))))/2))

x\_arr.append((x0 + (coef\*delta\_lambd\*(2\*\*(k-1)))))

lambd\_arr.append(func((x0 + (coef\*delta\_lambd\*(2\*\*(k-1))))))

center\_index = lambd\_arr.index(min(lambd\_arr))

x\_result = x\_arr[center\_index - 1: center\_index + 2]

return x\_result

def RRR\_diff(func, h):

Dh = (func(h) - func(- h))/(2\*h)

D2h = (func(2\*h) - func(- (2\*h)))/(2\*h)

return (4\*Dh - D2h)/3

def central\_diff(func, h):

dx = nd.Derivative(func, step=h, method='central')

return float(dx(0))

def forward\_diff(func, h):

dx = nd.Derivative(func, step=h, method='forward')

return float(dx(0))

def backward\_diff(func, h):

dx = nd.Derivative(func, step=h, method='backward')

return float(dx(0))

def get\_grad(x, h, diff\_func, func = f):

fun = create\_dir\_func(x, [1, 0], func)

dx1 = diff\_func(fun, h)

fun = create\_dir\_func(x, [0, 1], func)

dx2 = diff\_func(fun, h)

return [dx1, dx2]

def Gradient\_descent(x\_arr, y\_arr, eps = 0.001, delta\_lambd = 0.001, h = 0.02, partan = True, diff\_func = RRR\_diff, search\_method = Golden\_ratio, finish\_criteria = finish\_criteria\_delta, restriction = False):

x = [x\_arr[-1], y\_arr[-1]]

dirs = get\_grad(x, h, diff\_func)

while finish\_criteria(dirs, 0.001, x\_arr, y\_arr):

if search\_method == dsc\_Powell:

dirs = [i/module(dirs[0], dirs[1]) for i in dirs]

interval = Sven(0, delta\_lambd, create\_dir\_func(x, dirs))

lambd = search\_method(interval, eps, x, dirs)

x = [x[0] + lambd\*dirs[0], x[1] + lambd\*dirs[1]]

x\_arr.append(x[0])

y\_arr.append(x[1])

if ((len(x\_arr)%3) == 0)&partan:

dirs = [x[0] - x\_arr[-3], x[1] - y\_arr[-3]]

else:

dirs = get\_grad(x, h, diff\_func)

return x\_arr, y\_arr

x\_arr, y\_arr = [-1.2], [0]

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr)

print(f\_call\_counter)

plt.grid(True)

search, = plt.plot(x\_arr, y\_arr, color="orange")

dot = plt.scatter(1, 1)

plt.legend([dot, search], [ "Точка мінімуму", "Прямі пошуку"])

plt.show()

print("{:15s} {:7d} {:3.4f} {:3.4f}".format("Partan", f\_call\_counter, x\_arr[-1], y\_arr[-1]))

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, partan = False)

plt.grid(True)

search, = plt.plot(x\_arr, y\_arr, color="orange")

dot = plt.scatter(1, 1)

plt.legend([dot, search], [ "Точка мінімуму", "Прямі пошуку"])

plt.show()

print("{:15s} {:7d} {:3.4f} {:3.4f}".format("No partan", f\_call\_counter, x\_arr[-1], y\_arr[-1]))

print("{:7s} {:7s} {:6s} {:6s}".format("h step", "counter", "x1", "x2"))

values\_arr = [0.02, 0.01, 0.001, 0.0001, 0.00001]

counter\_arr = []

for val in values\_arr:

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, h = val)

print("{:7s} {:7d} {:3.4f} {:3.4f}".format(str(val), f\_call\_counter, x\_arr[-1], y\_arr[-1]))

counter\_arr.append(f\_call\_counter)

values\_arr = [RRR\_diff, central\_diff, forward\_diff, backward\_diff]

counter\_arr = []

print()

print("{:15s} {:7s} {:6s} {:6s}".format("diff method", "counter", "x1", "x2"))

for val in values\_arr:

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, h = 0.001, diff\_func = val)

print("{:15s} {:7d} {:3.4f} {:3.4f}".format(val.\_\_name\_\_, f\_call\_counter, x\_arr[-1], y\_arr[-1]))

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, h = 0.001, diff\_func = central\_diff)

print("\n{:15s} {:7d} {:3.4f} {:3.4f}".format("Golden Ratio", f\_call\_counter, x\_arr[-1], y\_arr[-1]))

plt.grid(True)

search, = plt.plot(x\_arr, y\_arr, color="orange")

dot = plt.scatter(1, 1)

plt.legend([dot, search], [ "Точка мінімуму", "Прямі пошуку"])

plt.show()

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, h = 0.001, diff\_func = central\_diff, search\_method = dsc\_Powell)

print("{:15s} {:7d} {:3.4f} {:3.4f}".format("DSC-Powell", f\_call\_counter, x\_arr[-1], y\_arr[-1]))

plt.grid(True)

search, = plt.plot(x\_arr, y\_arr, color="orange")

dot = plt.scatter(1, 1)

plt.legend([dot, search], [ "Точка мінімуму", "Прямі пошуку"])

plt.show()

print("\n{:7s} {:7s} {:6s} {:6s}".format("epsilon", "counter", "x1", "x2"))

values\_arr = [0.5, 0.1, 0.01, 0.001, 0.0001]

counter\_arr = []

for val in values\_arr:

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, eps = val, h = 0.001, diff\_func = central\_diff, search\_method = dsc\_Powell)

print("{:7s} {:7d} {:3.4f} {:3.4f}".format(str(val), f\_call\_counter, x\_arr[-1], y\_arr[-1]))

counter\_arr.append(f\_call\_counter)

print("\n{:10s} {:7s} {:6s} {:6s}".format("Sven step", "counter", "x1", "x2"))

values\_arr = [0.5, 0.1, 0.01, 0.001, 0.0001]

counter\_arr = []

for val in values\_arr:

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, eps = 0.0001, delta\_lambd = val, h = 0.001, diff\_func = central\_diff, search\_method = dsc\_Powell)

print("{:10s} {:7d} {:3.4f} {:3.4f}".format(str(val), f\_call\_counter, x\_arr[-1], y\_arr[-1]))

counter\_arr.append(f\_call\_counter)

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, eps = 0.0001, delta\_lambd = 0.5, h = 0.001, diff\_func = central\_diff, search\_method = dsc\_Powell)

print("\n{:15s} {:7d} {:3.4f} {:3.4f}".format("Module", f\_call\_counter, x\_arr[-1], y\_arr[-1]))

plt.grid(True)

search, = plt.plot(x\_arr, y\_arr, color="orange")

dot = plt.scatter(1, 1)

plt.legend([dot, search], [ "Точка мінімуму", "Прямі пошуку"])

plt.show()

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, eps = 0.0001, delta\_lambd = 0.5, h = 0.001, diff\_func = central\_diff, search\_method = dsc\_Powell, finish\_criteria = finish\_criteria\_grad)

print("{:15s} {:7d} {:3.4f} {:3.4f}".format("Gradient", f\_call\_counter, x\_arr[-1], y\_arr[-1]))

plt.grid(True)

search, = plt.plot(x\_arr, y\_arr, color="orange")

dot = plt.scatter(1, 1)

plt.legend([dot, search], [ "Точка мінімуму", "Прямі пошуку"])

plt.show()

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent(x\_arr, y\_arr, eps = 0.0001, delta\_lambd = 0.5, h = 0.001, partan = False, diff\_func = central\_diff, search\_method = dsc\_Powell, finish\_criteria = finish\_criteria\_grad)

print("{:15s} {:7d} {:3.4f} {:3.4f}".format("No partan", f\_call\_counter, x\_arr[-1], y\_arr[-1]))

plt.grid(True)

search, = plt.plot(x\_arr, y\_arr, color="orange")

dot = plt.scatter(1, 1)

plt.legend([dot, search], [ "Точка мінімуму", "Прямі пошуку"])

plt.show()

def restriction\_linear(x1):

return 0.5 - 0.8\*x1

r1\_arr = np.linspace(-1.3, 1.25, 20)

r2\_arr = [restriction\_linear(r1) for r1 in r1\_arr]

R = 0

def restriction(x1, x2):

return 0.8\*x1 + x2 - 0.5

def outer\_point(x1, x2, R = 1):

return f(x1, x2) + R\*(0.8\*x1 + x2 - 0.5)

def set\_R(r):

def R\_set(x1, x2):

return outer\_point(x1, x2, r)

return R\_set

def create\_dir\_func(x, direction = [1, 0], func = set\_R(R)):

func = set\_R(R)

def dir\_func(h):

return func(x[0] + direction[0]\*h, x[1] + direction[1]\*h)

return dir\_func

def Gradient\_descent\_restricted(x\_arr, y\_arr, partan = True, diff\_func = RRR\_diff, R\_var = 100):

eps = 0.0001

delta\_lambd = 0.5

h = 0.001

search\_method = dsc\_Powell

finish\_criteria = finish\_criteria\_grad

diff\_func = central\_diff

global R

x = [x\_arr[-1], y\_arr[-1]]

if (restriction(x[0],x[1])>0):

R = R\_var

else:

R = 0

dirs = get\_grad(x, h, diff\_func)

while finish\_criteria(dirs, 1, x\_arr, y\_arr)|(restriction(x[0],x[1])>0):

if search\_method == dsc\_Powell:

dirs = [i/module(dirs[0], dirs[1]) for i in dirs]

interval = Sven(0, delta\_lambd, create\_dir\_func(x, dirs))

lambd = search\_method(interval, eps, x, dirs)

x = [x[0] + lambd\*dirs[0], x[1] + lambd\*dirs[1]]

x\_arr.append(x[0])

y\_arr.append(x[1])

if ((len(x\_arr)%3) == 0)&partan:

dirs = [x[0] - x\_arr[-3], x[1] - y\_arr[-3]]

else:

dirs = get\_grad(x, h, diff\_func)

if (restriction(x[0],x[1])>0):

R = R\_var

else:

R = 0

return x\_arr, y\_arr

R\_list = [1, 10, 100, 1000]

print("\n{:5s} {:7s} {:6s} {:6s}".format("R", "counter", "x1", "x2"))

for R\_var in R\_list:

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent\_restricted(x\_arr, y\_arr, R\_var = R\_var)

print("{:5s} {:7d} {:3.4f} {:3.4f}".format(str(R\_var), f\_call\_counter, x\_arr[-1], y\_arr[-1]))

x\_arr, y\_arr = reset(x\_arr, y\_arr)

x\_arr, y\_arr = Gradient\_descent\_restricted(x\_arr, y\_arr, R\_var = 10)

plt.grid(True)

search, = plt.plot(x\_arr, y\_arr, color="orange")

restriction\_plot, = plt.plot(r1\_arr, r2\_arr, color="red")

dot = plt.scatter(1, 1)

plt.legend([dot, restriction\_plot, search], ["Точка мінімуму", "Обмеження", "Прямі пошуку"])

plt.show()