Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»  
Кафедра прикладної математики

КУРСОВА РОБОТА  
з дисципліни «Дослідження операцій»  
на тему:  
«Модифікований метод Ньютона»

Виконав: Керівник:   
студент групи КМ-83 старший викладач   
Касьяновський О.Е. Норкін Б.В.

КИЇВ 2021

Зміст

[1 ПОСТАНОВКА ЗАДАЧІ 3](#_Toc73612851)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 4](#_Toc73612852)

[3 РОЗРАХУНКОВА ЧАСТРИНА 8](#_Toc73612853)

[3.1 безумовна оптимізація 8](#_Toc73612854)

[3.1.1 величина і спосіб обчислення кроку h 8](#_Toc73612855)

[3.1.2 схема обчислення І та ІІ похідної 8](#_Toc73612856)

[3.1.3 Вид одновимірного пошуку (ДСК-Пауелла, Золотого перетину) 9](#_Toc73612857)

[3.1.4 Точність метода одновимірного пошуку 10](#_Toc73612858)

[3.1.5 Значення параметру в алгоритмі Свена 10](#_Toc73612859)

[3.1.6 вид критерія закінчення 11](#_Toc73612860)

[3.2 умовна оптимізація 12](#_Toc73612861)

[3.2.1 Метод внутрішньої точки (бар’єрних точок) 12](#_Toc73612862)

[ВИСНОВКИ 13](#_Toc73612863)

[КОД ПРОГРАМИ 16](#_Toc73612864)

# 1 ПОСТАНОВКА ЗАДАЧІ

Дослідити збіжність модифікованого методу Ньютона при мінімізації кореневої функції в залежності від:

*f(x)=(10(x1-x2)2+(x1-1)2)1/4, x(0)=(-1,2; 0,0)* в залежності від:

1. Величини кроку h при обчисленні першої та другої похідних.
2. Схеми обчислення першої та другої 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.

# 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

**Метод Ньютона.**

Розкладемо цільову функцію в ряд Тейлора, відкинемо всі члени розкладання третього порядку і вище - отримаємо квадратичну апроксимацію *f(x)*



де *f(x)* - апроксимуюча функція змінної x в точці .



Якщо - напрямок пошуку в методі Ньютона, тоді



Мінімум функції *f(x)* в напрямку  визначається диференціюванням *f(x)* по кожній з компонент  і прирівнюванням отриманих виразів нулю

.

Звідки

,

де  - матриця зворотна матриці Гессе  в точці . Перехід з точки  в точку  визначається за методом Ньютона наступним чином:

.

Напрямки і величина кроку тут точно визначено. У задачі пошуку мінімуму довільної квадратичної функції з позитивно певної матрицею Гессе метод Ньютона дає рішення за одну ітерацію, причому незалежно від вибору початкової точки. У разі загальної нелінійної цільової функції метод Ньютона зійдеться до шуканої точці  квадратично при наступних умовах: матриця Гессе мінімізуємої в точці  повинна бути додатьо визначеною, початкова точка повинна знаходитися досить близько до . Так як метод Ньютона заснований на квадратичної апроксимації, він володіє квадратичної швидкістю збіжності, тобто виконується нерівність

,

де постійна С пов'язана з обумовленістю матриці Гессе. Алгоритм не має властивість зменшення значень цільової функції від ітерації до ітерації. Для того, щоб напрямок пошуку було напрямом спуску повинна виконуватись нерівність

.

Припустимо, що поточне наближення  не є стаціонарною точкою (т.е ), і знайдемо проекцію напрямки пошуку по методу Ньютона на напрям, що задається градієнтом в точці 

.

У разі, коли матриця Гессе  додатньо визначена в точці , ця умова виконується, отже, напрямок пошуку за методом Ньютона виявляється напрямком спуску. Якщо в деякій точці  від’ємно визначена, то даний напрямок є напрямком підйому. У разі невизначеності матриці Гессе не можна зробити однозначний висновок. При мінімізації, коли всі власні значення позитивні  (Матриця додатньо визначена), локальна квадратична апроксимація відповідає кругової або еліптичної западині, що має мінімум. Якщо пара власних значень має протилежні знаки (матриця невизначена), квадратична апроксимація є сідло, яке не має локального мінімуму. У цьому випадку рух у напрямку пошуку по методу Ньютона призведе до седлової точці. Критерій, що гарантує збіжність методу Ньютона в припущенні, що функція *f(x)* двічі диференційовна, полягає в тому, що матриця, зворотна матриці Гессе цільової функції повинна бути додатньо визначеною.



**Модифікований метод Ньютона.**

Для неквадратічних функцій загального вигляду метод Ньютона не відрізняється надійністю. Збіжність методу дуже залежить від вибору початкової точки . Якщо  знаходиться на значній відстані від шуканого мінімуму, крок за методом Ньютона може виявитися досить великим, і це може привести до відсутності збіжності даного методу. Щоб забезпечити зменшення значення цільової функції від ітерації до ітерації вводиться наступний параметр довжини кроку λ:

,

де відношення  - деякий скаляр , тому

.

Напрямок пошуку задається вектором :

.

Вибір  здійснюється таким чином, щоб

.

Такий вибір  гарантує виконання нерівності

.

# 3 РОЗРАХУНКОВА ЧАСТРИНА

## 3.1 безумовна оптимізація

### 3.1.1 величина і спосіб обчислення кроку h

Схема: центральна  
Параметр: 2  
Точність = 0.01  
Одномірний пошук: Дихотомія  
Параметр Свена: 2  
Критерій закинченні: 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Крок | X | Y | F(x) | Ітерації |
| 0.1 | 1.000338670423 | 1.0003393753858 | 0.018403199731567 | 40 |
| 0.01 | 1.000342876417 | 1.0003698468931 | 0.018796931739909 | 27 |
| 0.001 | 0.999741979428 | 0.9997125044548 | 0.016563207871032 | 17 |

Як можна побачити за результатами таблиці чим менше крок тім точніше буде результат обчислень. Обираємо крок що = 0.001.

### 3.1.2 схема обчислення І та ІІ похідної

Крок = 0.001  
Параметр: 2  
Точність = 0.01  
Одномірний пошук: Дихотомія  
Параметр Свена: 2  
Критерій закінчення: 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Схема | X | Y | F(x) | Ітерації |
| Ліва | 1.000025847853 | 1.0000952809017 | 0.014868852547236 | 6 |
| Центр | 0.999741979428 | 0.9997125044548 | 0.016563207871032 | 17 |
| Прав | 0.998865656232 | 0.9984495437712 | 0.041681000964557 | 12 |

Як можна побачити за результатами таблиці хоча середня схема найбільш точна вона дає найгірший результат. Обираємо ліву різницеву схему.

Для даної роботи було використано різницеві схеми.

Для знаходження І похідної використовуємо наступні формули

Ліва схема

\lambda_{x}^+ u=\frac{u_{j+1}-u_{j}}{h}

Права схема

\lambda_{x}^- u=\frac{u_{j}-u_{j-1}}{h}

Центральна схема

\lambda_{x}^0 u=\frac{u_{j+1}-u_{j-1}}{2h}

Як можна побачити за даними таблиці найточніше похідну нам дає центральна схема.

Для знаходження ІІ похідної використовуємо наступні формули

На прикладі першої похідної ми бачимо що центральна схема є найточнішою

Центральна схема

### 3.1.3 Вид одновимірного пошуку (ДСК-Пауелла, Золотого перетину)

В якості методу одномірного пошуку було вибрано метод ДСК-Пауелла так як він більше точній тому що використовує квадратичну апроксимацію

Крок = 0.001  
Параметр: 2  
Точність = 0.01  
Схема: ліва  
Параметр Свена: 2  
Критерій закінчення: 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | X | Y | F(x) | Ітерації |
| Дих | 1.000025847853 | 1.0000952809017 | 0.014868852547236 | 6 |
| ДСК | 1.000450584095 | 1.0007094164503 | 0.030566771412195 | 12 |

Як можна побачити за результатами таблиці метод дихотомії є кращім хоча він і менш точній. Обираємо в якості одномірного пошуку метод Дихотомії.

### 3.1.4 Точність метода одновимірного пошуку

Крок = 0.001  
Параметр: 2  
Схема: ліва  
Одномірний пошук: Дихотомія  
Параметр Свена: 2  
Критерій закінчення: 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Точн | X | Y | F(x) | Ітерації |
| 0.1 | 0.999278571714 | 0.9989333266825 | 0.036174397829070 | 4 |
| 0.01 | 1.000058478531 | 1.0000952809017 | 0.014868852547236 | 6 |
| 0.001 | 1.002533402370 | 1.0020427625969 | 0.054504657801617 | 9 |

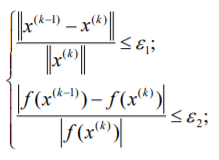
### 3.1.5 Значення параметру в алгоритмі Свена

Крок = 0.001  
Схема: ліва  
Одномірний пошук: Дихотомія  
Параметр Свена: 2  
Критерій закінчення: 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Точн | X | Y | F(x) | Ітерації |
| 1.7 | 1.000046980555 | 0.9999734725134 | 0.015399769255401 | 7 |
| 2 | 1.002533402370 | 1.0020427625969 | 0.054504657801617 | 9 |
| 2.2 | 1.002345999581 | 1.0019034144368 | 0.052266272034081 | 8 |

Як можна побачити за результатами таблиці стандартний коефіцієнт не є найкращім. Обираємо ліву різницеву схему.

### 3.1.6 вид критерія закінчення

 1 критерій

 2 критерій

Крок = 0.001  
Параметр: 2  
Схема: ліва  
Одномірний пошук: Дихотомія  
Параметр Свена: 1.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вид | X | Y | F(x) | Ітерації |
| 1 | 0.999278571714 | 0.9989333266825 | 0.036174397829070 | 7 |
| 2 | 1.000058478531 | 1.0000952809017 | 0.014868852547236 | 14 |

Як можна побачити за результатами таблиці перший критерій закінчення є більш кращім. Обираємо його.

## 3.2 умовна оптимізація

Методи штрафних функцій це методи можуть використовуватися як при лінійних, так і при нелінійних обмеженнях типу рівностей і нерівностей. Зміст методів полягає в тому, що вихідна цільова функція *f0(u)* розширюється за рахунок введення додаткових коефіцієнтів і функцій, що враховують вплив обмежень таким чином, що при порушенні обмежень різко зростає (спадає) значення розширеної цільової функції Ip(u). Тим самим рішення задачі умовної оптимізації f0(u) зводиться до розв’язку задачі безумовної оптимізації Ip(u).

### 3.2.1 Метод внутрішньої точки (бар’єрних точок)

Таким чином, задача з обмеженнями перетворюється у послідовність підзадача безумовної оптимізації. При розв’язуванні задачі необхідно обрати початкове значення штрафного параметра r та змінювати його після кожної підзадачі безумовної оптимізації так, щоб забезпечити збіжність послідовності стаціонарних точок u(k) до оптимального значення

Недоліки метода:

1. Необхідність пошуку допустимої точки, щоб використати її у якості початкової.

2. Штрафна функція F(ū) має чітко виражену «овражної» структуру при малих rk, що ускладнює рішення задач безумовної оптимізації Fk (ū).

3. Метод призначений для рішення задач нелінійного програмування з обмеженнями типу нерівності.

3.2.2 Метод зовнішньої точки

Метод передбачає побудову штрафних таким чином, щоб значення перетвореної цільової функції F(u, r) у допустимій області точно або наближено дорівнювали початковій цільовій функції f(u) а ззовні значно перевищувала значення f(u).

# ВИСНОВКИ

У даній роботі був реалізований алгоритм модифікованого методу Ньютона пошуку мінімуму функції за допомогою мови програмування Python. Була досліджена збіжність даного методу для кореневої функції із заданої початкової точки, критерієм даної збіжності слугувала кількість обчислень функції та точність результатів.

У даній програмі мінімум задачі знаходився з точністю , що є достатнім для візуалізації збіжності методу та водночас дозволяло обчислювати значення невідомої функції відносно невелику кількість разів.

Була досліджена збіжність модифікованого методу Ньютона для кореневої функції в залежності від: величини кроку h, схеми обчислення, виду одномірного пошуку, точності методу, Параметра Свена, та критерія закінчення

Нижче представлені результати дослідження.

Величина кроку h впливає но точність обчислення похідних а також на кількість ітерацій за який цей метод зійдеться, в данному віпадку біло обратн крок що = 0.001.

В якості схеми обчислення похідних будо лбрано Ліву, хоча вона і не є більш точною для даного методу вона хабезпепує найбільшу швідкість сходимості.

В якості методу одномірного пошуку було обрано метод Дихотомії тому що він сходиться набагото швідше ніж ДСК.

Для данного віпадку біло обрано точність що = 0.001, але якщо задача потребує більшої точность то це число можна зміниті що в свою чергу збільшить кількцість інерацій.

В якості коіфіціента в прарметрі Свена було обрано число 1.7, що зменшело кількість ітерацій на 2.

Отримано такі результати:

xmin = (1.0000469805552397; 0.9999734725134045)   
f(xmin) = 0.015399769255401662.

Результат досягнений за 7 обчислень цільової функції з точністю ε = 0.001.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Д. Хіммельблау. «Прикладне нелінійне програмування». М.; «Мир», 1975.

2. Г.В. Реклейтіс «Оптимизация в технике» частина 1, М.; «Мир», 1986.

# КОД ПРОГРАМИ

from sympy import diff, symbols  
import math  
x\_1 = symbols('x1')  
x\_2 = symbols('x2')  
  
  
x0 = [[-1.2, 0.0]]  
h = 0.1  
E = 0.000001  
  
matrix\_h = [[], []]  
matrix\_h\_obr = [[], []]  
matrix = []  
res\_matrix = []  
lambdf = []  
napravlenie = []  
  
  
def fx\_add(x1, x2):  
 u = diff("(10\*(x1-x2)\*\*2+(x1-1)\*\*2)\*\*(1/4)", x\_1, 1)  
 return eval(str(u))  
  
  
def fx\_func(x1, x2):  
 return (10\*(x1-x2)\*\*2+(x1-1)\*\*2)\*\*(1/4)  
  
  
def left\_1(x, y, h):  
 return (fx\_func(x, y) - fx\_func(x - h, y)) / h  
  
  
def right\_1(x, y, h):  
 return (fx\_func(x + h, y) - fx\_func(x, y)) / h  
  
  
def center\_1\_x(x, y, h):  
 return (fx\_func(x + h, y) - fx\_func(x - h, y)) / (2 \* h)  
  
  
def center\_1\_y(x, y, h):  
 return (fx\_func(x, y + h) - fx\_func(x, y - h)) / (2 \* h)  
  
  
def center\_2\_xy(x, y, h):  
 return (fx\_func(x + h, y + h) - fx\_func(x + h, y - h) - fx\_func(x - h, y + h) + fx\_func(x - h, y - h)) / (4 \* h \*\* 2)  
  
  
def center\_2\_x(x, y, h):  
 return (fx\_func(x + h, y) - fx\_func(x, y) - fx\_func(x, y) + fx\_func(x - h, y)) / (h \*\* 2)  
  
  
def center\_2\_y(x, y, h):  
 return (fx\_func(x, y + h) - fx\_func(x, y) - fx\_func(x, y) + fx\_func(x, y - h)) / (h \*\* 2)  
  
  
def det(matrix\_h):  
 matrix\_h[0] = [round(center\_2\_x(x0[-1][0], x0[-1][1], h), 5), round(center\_2\_xy(x0[-1][0], x0[-1][1], h), 5)]  
 matrix\_h[1] = [round(center\_2\_xy(x0[-1][0], x0[-1][1], h), 5), round(center\_2\_y(x0[-1][0], x0[-1][1], h), 5)]  
 return matrix\_h[0][0] \* matrix\_h[1][1] - matrix\_h[0][1] \* matrix\_h[1][0]  
  
  
def obr(matrix\_h, matrix\_h\_obr):  
 detet = det(matrix\_h)  
 matrix\_h[0][1] = -matrix\_h[0][1]  
 matrix\_h[1][0] = -matrix\_h[1][0]  
  
 matrix\_h[0][0], matrix\_h[1][1] = matrix\_h[1][1], matrix\_h[0][0]  
  
 matrix\_h\_obr[0] = [round(matrix\_h[0][0] / detet, 5), round(matrix\_h[0][1] / detet, 5)]  
 matrix\_h\_obr[1] = [round(matrix\_h[1][0] / detet, 5), round(matrix\_h[1][1] / detet, 5)]  
 return matrix\_h\_obr  
  
  
def matrix\_1(matrix):  
 matrix = [round(center\_1\_x(x0[-1][0], x0[-1][1], h), 5), round(center\_1\_y(x0[-1][0], x0[-1][1], h), 5)]  
 return matrix  
  
  
def mnoj(matrix\_h, matrix, res\_matrix, matrix\_h\_obr):  
 matrix\_h\_obr = obr(matrix\_h, matrix\_h\_obr)  
 matrix = matrix\_1(matrix)  
 res\_matrix = [round(matrix\_h\_obr[0][0] \* matrix[0] + matrix\_h\_obr[0][1] \* matrix[1], 5), round(matrix\_h\_obr[1][0] \* matrix[0] + matrix\_h\_obr[1][1] \* matrix[1], 5)]  
 return res\_matrix  
  
  
def norm(array):  
 return math.sqrt(sum([value \*\* 2 for value in array]))  
  
  
def l(x, s):  
 return 0.001 \* (norm(x) / norm(s))  
  
  
def naprav():  
 return mnoj(matrix\_h, matrix, res\_matrix, matrix\_h\_obr)  
  
  
def get\_new\_x(x, l, s):  
 return [x[0] + l \* s[0], x[1] + l \* s[1]]  
  
  
def sven(x0, s):  
 is\_end = False  
 lambd = l(x0, s)  
 power = 1  
 lambdf.append(lambd)  
 values = []  
 x\_plus = get\_new\_x(x0, lambd, s)  
 x\_minus = get\_new\_x(x0, - lambd, s)  
 a = fx\_func(\*x\_plus)  
 b = fx\_func(\*x\_minus)  
 c = fx\_func(\*x0)  
 if fx\_func(\*x\_plus) < fx\_func(\*x\_minus):  
 values.append({"l": lambd, "x": x\_plus, "f(x)": fx\_func(\*x\_plus)})  
 else:  
 values.append({"l": -lambd, "x": x\_minus, "f(x)": fx\_func(\*x\_minus)})  
  
 while not is\_end:  
 new\_l = values[-1]["l"] + 2 \*\* power \* values[0]["l"]  
 x = get\_new\_x(x0, new\_l, s)  
 fx = fx\_func(\*x)  
  
 values.append({"l": new\_l, "x": x, "f(x)": fx})  
 power += 1  
  
 if values[-2]["f(x)"] < values[-1]["f(x)"]:  
 middle\_l = (values[-1]["l"] + values[-2]["l"]) / 2  
 middle\_x = get\_new\_x(x0, middle\_l, s)  
 values.append({"l": middle\_l, "x": middle\_x, "f(x)": fx\_func(\*middle\_x)})  
 is\_end = True  
  
 if len(values) == 3:  
 # return values[0], {"l": (values[1]["l"] + values[2]["l"])/2, "x": numpy.ndarray([(values[1]["x"][0] + values[2]["x"][0])/2, (values[1]["x"][1] + values[2]["x"][1])/2]), "f(x)": (values[1]["f(x)"] + values[2]["f(x)"])/2}  
 return values[0], values[1]  
 else:  
 return values[-4], values[-1]  
  
  
def dsk\_powell(x0, interval, accuracy, s):  
 def get\_middle\_value(inter):  
 l = (inter[0]["l"] + inter[-1]["l"]) / 2  
 x = get\_new\_x(x0, l, s),  
 return {"l": l,  
 "x": x[0],  
 "f(x)": fx\_func(\*x[0])  
 }  
  
 def get\_dsk\_approximate\_min(inter, s):  
 middle\_value = get\_middle\_value(inter)  
 l\_min = middle\_value["l"] + (  
 ((inter[0]["f(x)"] - inter[1]["f(x)"]) \* abs(middle\_value["l"] - inter[0]["l"])) /  
 (2 \* (inter[0]["f(x)"] - 2 \* middle\_value["f(x)"] + inter[1]["f(x)"])))  
  
 x\_min = get\_new\_x(inter[0]["x"], l\_min - inter[0]["l"], s)  
 fx\_min = fx\_func(\*x\_min)  
  
 return {  
 "l": l\_min,  
 "x": x\_min,  
 "f(x)": fx\_min  
 }  
  
 def full\_dsk\_interval(inter):  
 return {  
 "left": inter[0],  
 "middle": get\_middle\_value(inter),  
 "approximate\_min": get\_dsk\_approximate\_min(inter, s),  
 "right": inter[-1]  
 }  
  
 def get\_powell\_approximate\_min(inter):  
 a1 = (inter[1]["f(x)"] - inter[0]["f(x)"]) / (inter[1]["l"] - inter[0]["l"])  
 a2 = (((inter[2]["f(x)"] - inter[0]["f(x)"]) / (inter[2]["l"] - inter[0]["l"])) - a1) / (  
 inter[2]["l"] - inter[1]["l"])  
 l\_min = (inter[0]["l"] + inter[1]["l"]) / 2 - (a1 / (2 \* a2))  
  
 x\_min = get\_new\_x(x0, l\_min, s)  
 fx\_min = fx\_func(\*x\_min)  
 return {  
 "l": l\_min,  
 "x": x\_min,  
 "f(x)": fx\_min  
 }  
  
 def full\_powell\_interval(inter):  
 sorted\_interval = sorted(inter, key=lambda point: point["l"])  
 return {  
 "left": sorted\_interval[0],  
 "middle": sorted\_interval[1],  
 "approximate\_min": get\_powell\_approximate\_min(sorted\_interval),  
 "right": sorted\_interval[2]  
 }  
  
 values = [full\_dsk\_interval(interval)]  
 points = sorted([value for value in values[-1].values()], key=lambda value: value["l"])  
 values.append(full\_powell\_interval([points[1], points[2], points[3]]))  
 return values  
  
  
  
napravka = [-naprav()[0], -naprav()[1]]  
sven\_interval\_1 = sven(x0[-1], napravka)  
  
precise\_interval\_1 = dsk\_powell(x0[-1], sven\_interval\_1, E, napravka)  
  
lamda = precise\_interval\_1[1]['approximate\_min']["l"]  
  
x0.append([x0[-1][0] - lamda \* mnoj(matrix\_h, matrix, res\_matrix, matrix\_h\_obr)[0],  
 x0[-1][1] - lamda \* mnoj(matrix\_h, matrix, res\_matrix, matrix\_h\_obr)[1]])  
  
  
while (abs(x0[-2][0] - x0[-1][0]) > 0.01 and abs(x0[-2][1] - x0[-1][1]) > 0.01):  
 napravka = [-naprav()[0], -naprav()[1]]  
 sven\_interval\_1 = sven(x0[-1], napravka)  
 precise\_interval\_1 = dsk\_powell(x0[-1], sven\_interval\_1, E, napravka)  
  
 lamda = precise\_interval\_1[1]['approximate\_min']["l"]  
  
 x0.append([x0[-1][0] - lamda \* mnoj(matrix\_h, matrix, res\_matrix, matrix\_h\_obr)[0],  
 x0[-1][1] - lamda \* mnoj(matrix\_h, matrix, res\_matrix, matrix\_h\_obr)[1]])  
  
print(x0)