МIНIСТЕРСТВО ОСВIТИ I НАУКИ УКРАЇНИ

НАЦIОНАЛЬНИЙ ТЕХНIЧНИЙ УНIВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛIТЕХНIЧНИЙ IНСТИТУТ»

Кафедра прикладної математики

Звіт

із лабораторної роботи №4

з дисципліни «Алгоритми і системи комп’ютерної математики.

Математичні алгоритми»

на тему:

«Метод Нелдера-Міда»

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студент групи КМ-63 | *Старший викладач Бай Ю.П.* |
| *Артеменко Я.К.* |  |

Київ — 2019

# **ЗМІСТ**

[**ВСТУП** 3](#_Toc26639881)

[**2 ОСНОВНА ЧАСТИНА** 4](#_Toc26639882)

[**2.1 Завдання** 4](#_Toc26639883)

[**2.2 Описання методу** 4](#_Toc26639884)

[**2.3 Використані тест-кейси** 10](#_Toc26639885)

[**3 ВИСНВОКИ** 11](#_Toc26639886)

[**ПЕРЕЛІК ПОСИЛАНЬ** 12](#_Toc26639887)

[**ДОДАТКИ** 13](#_Toc26639888)

[**Додаток А (код програми для поставленої задачі)** 13](#_Toc26639889)

[**Додаток Б (код програми для тестів)** 15](#_Toc26639890)

[**Додаток Б (результати виконання програми)** 17](#_Toc26639891)

[**Додаток Г (підтвердження правильності роботи алгоритму)** 18](#_Toc26639892)

# **ВСТУП**

Мета лабораторної роботи:

Розробити програмне забезпечення для методу Нелдера-Міда вирішення задачі безумовної оптимізації та провести аналіз задачі, що розв’язується, та методу її розв’язання на предмет виключних ситуацій, які можуть виникати під час застосування заданого методу до розв’язання поставленої задачі.

Метод Нелдера-Міда (деформованого багатогранника) призначений для наближеного рішення безумовної n-мірної задачі нелінійного програмування виду: f (x1, x2, ......., хn). Цей метод відноситься до групи методів 0-го порядку (прямого пошуку), так як в ньому не використовуються похідні. Метод є розвитком симплексного методу Спендлі та інших.

# **2 ОСНОВНА ЧАСТИНА**

# **2.1 Завдання**

Реалізувати метод для функції:

В результаті має вийти точка мінімуму «(1;1)» і значення функції має бути рівним «-1».

# **2.2 Описання методу**

Метод Нелдера-Міда – метод оптимізації (пошуку мінімуму) функції від декількох змінних. Простий і в той же час ефективний метод, що дозволяє оптимізувати функції без використання градієнтів.

Алгоритм полягає в формуванні симплекса і подальшого його деформування в напрямку мінімуму, за допомогою трьох операцій:

1. Відображення;
2. Розтягування;
3. Стиснення;

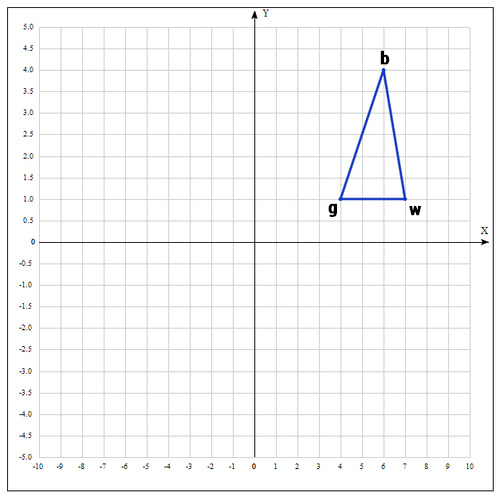
Симплекс вдає із себе геометричну фігуру, яка є n - мірним узагальненням трикутника. Для одновимірного простору - це відрізок, для двовимірного - трикутник. Таким чином n - мірний симплекс має n + 1 вершину.

Алгоритм:

1. Нехай функція, яку необхідно оптимізувати. На першому кроці вибираємо три випадкові точки і формуємо симплекс (трикутник). Обчислюємо значення функції в кожній точці: ,,. Сортуємо точки за значеннями функції в цих точках, таким чином отримуємо подвійну нерівність: .

Оскільки виконується пошук мінімуму функції, то на даному кроці буде найкращою та точка, в якій значення функції мінімальне. Для спрощення точки позначаються наступним чином:

b = V2, g = V1, w = V3, де b – best (найкраща точка), g – good (добра точка), w – wrong (погана точка) відповідно.



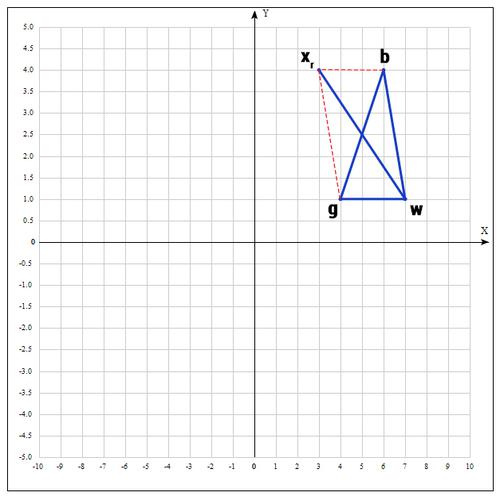
*Рис. 1*. Початкове положення точок

1. На наступному кроці знаходимо середину відрізка, точками якого є g і b. Оскільки координати середини відрізка рівні напівсумі координат його кінців, отримуємо:
2. Застосовуємо операцію відображення:

Знаходимо точку xr наступним чином:

Тобто фактично відображуємо точку w відносно mid. Коефіцієнт α зазвичай приймають рівним 1. Після цього перевіряється наша точка:

Якщо , то це добра точка.



*Рис. 2.* Нова знайдена точка Xr при відображенні

1. Використовується операція розтягування:

Знаходиться точка хе наступним чином:

В якості приймаємо , тобто відстань збільшуємо у 2 рази. Після цього перевіряємо точку хе:

Якщо , то це означає, що знайдена краща точка, ніж та, яка є на даний момент. Якщо б цього не відбулось, то кращою залишалась би точка хr. Далі точка w замінюється на xe и в результаті отримується:

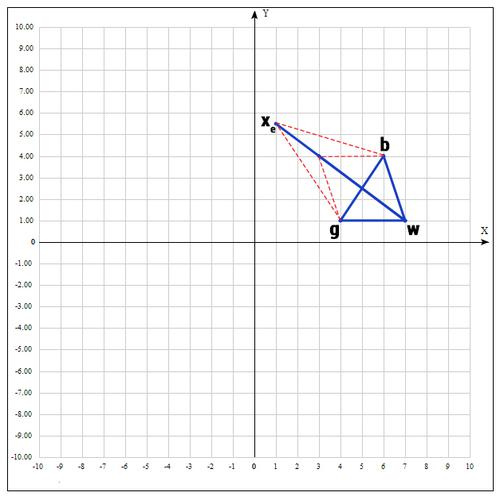


Рис.3. Нова знайдена точка хе при розтягуванні

1. Якщо впродовж цих кроків не було знайдено хороших точок, то потрібно пробувати операцію стискання. Під час цієї операції зменшується відрізок і точки знаходяться всередині трикутника.

Знаходиться хороша точка хс:

Хс =

Коефіцієнт β приймається рівним 0.5, тобто точка знаходиться на середині відрізка w… mid:

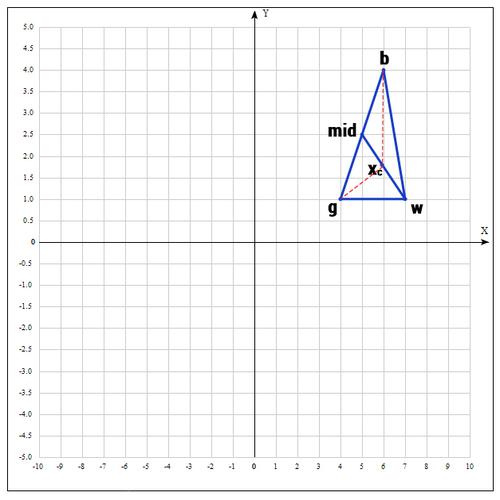


Рис. 4. Нова знайдена точка хс при стисканні

Існує ще одна операція – скорочення. В даному випадку, ми перевизначаємо весь симплекс. Ми залишаємо тільки «кращу» точку, інші визначаємо наступним чином:

, де коефіцієнт = 0.5

Загалом, існуючі точки переміщуються по направленню до поточної «кращої» точки. Перетворення виглядає наступним чином:

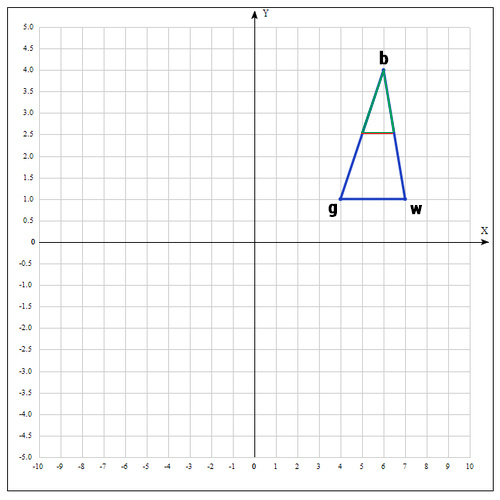


Рис. 5. Зміщення до поточної «кращої» точки

Необхідно відмітити, що дана операція дорого обходиться, оскільки необхідно заміняти точки в симплексі, тому ця операція дуже рідко зустрічається на практиці.

Алгоритм закінчується, коли:

1. Було виконано необхідну кількість ітерацій;
2. Площа симплекса досягла певної величини;
3. Поточне краще рішення досягло необхідної точності.

Перевагами даного методу є те, що він являється дуже ефективним алгоритмом пошуку екстремуму функції багатьох змінних, не накладуючим обмеження на гладкість функції. На кожній ітерації алгоритму проводиться як правило одне-два обчислення значень функції, що надзвичайно ефективно якщо ці обчислення дуже повільні. Крім того, алгоритму дуже простий в реалізації. Крім того даний алгоритм простий в реалізації.

Недоліками даного методу являється неможливість відрізнити локальний максимум від глобального, а також відсутність теорії збіжності і наявність прикладів, коли метод розходиться навіть на гладких функціях.

# **2.3 Використані тест-кейси**

Для даної лабораторної роботи було створено 5 тест-кейси, які перевіряють правильність виконання алгоритму. Функції було взято з методички, тому правильність виконання можна відразу перевіряти, оскільки там є відповіді для кожної функції. Перевірки відбувається для чотирьох різних функцій:

1. Функція, яка задана в умові лабораторної роботи;
2. ;
3. ;
4. .
5. .

# **3 ВИСНВОКИ**

В даній лабораторній роботі було розроблено програмне забезпечення для розв’язання задачі безумовної оптимізації за допомогою методу Нелдера-Міда. Також було розроблено ряд тестів, які перевіряють наявність різних помилок і виконують певні дії з помилками, які можуть виникати при розв’язанні поставленої задачі.

# **ПЕРЕЛІК ПОСИЛАНЬ**

1. Дослідження операцій. Рекомендації до виконання курсової роботи [Електронний ресурс]: навч. посіб. для студ. спеціальності 113 «Прикладна математика», спеціалізації «Наука про дані та математичне моделювання» / Т. С. Ладогубець, О. Д. Фіногенов; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 180 Кбайт). – Київ: КПІ ім. Ігоря Сікорського, 2019. – 34 с.
2. Bürmen, A., Puhan, J., and Tuma, T. (2006), “Grid Restrained Nelder-Mead Algorithm”, *Comput. Optim. Appl.* **34**, pp. 359–375.
3. А.Г.Трифонов. "Постановка задачи оптимизации и численные методы ее решения" [електронний ресурс] URL: <http://matlab.exponenta.ru/optimiz/book_2/2_1.php>
4. Химмельблау Д. Прикладное нелинейное программирование. М.: Мир, 1983г.

# **ДОДАТКИ**

# **Додаток А (код програми для поставленої задачі)**

from math import exp, log, sin, cos, tan  
from add import \*  
  
class Diapason(Exception):  
 def \_\_init\_\_(self, text):  
 self.text = text  
  
  
def f(point, funf):  
 x, y = point  
 return eval(funf)  
  
  
def nelder\_mead(fu, x1, x2, x3, maxiter=70, alpha=1, beta=0.5, gamma=2,):  
 # initialization  
 v1 = Vector(x1[0], x1[1])  
 v2 = Vector(x2[0], x2[1])  
 v3 = Vector(x3[0], x3[1])  
  
 for i in range(maxiter):  
 adict = {v1: f(v1.c(), fu), v2: f(v2.c(), fu), v3: f(v3.c(), fu)}  
 points = sorted(adict.items(), key=lambda x: x[1])  
  
 b = points[0][0]  
 g = points[1][0]  
 w = points[2][0]  
  
 mid = (g + b) / 2  
  
 # reflection  
 xr = mid + alpha \* (mid - w)  
 if f(xr.c(), fu) < f(g.c(), fu):  
 w = xr  
 else:  
 if f(xr.c(), fu) < f(w.c(), fu):  
 w = xr  
 c = (w + mid) / 2  
 if f(c.c(), fu) < f(w.c(), fu):  
 w = c  
 if f(xr.c(), fu) < f(b.c(), fu):  
  
 # expansion  
 xe = mid + gamma \* (xr - mid)  
 if f(xe.c(), fu) < f(xr.c(), fu):  
 w = xe  
 else:  
 w = xr  
 if f(xr.c(), fu) > f(g.c(), fu):  
  
 # contraction  
 xc = mid + beta \* (w - mid)  
 if f(xc.c(), fu) < f(w.c(), fu):  
 w = xc  
  
 # update points  
 v1 = w  
 v2 = g  
 v3 = b  
 return b  
  
  
def out(fu, x1, x2, x3, count):  
 xk = nelder\_mead(fu, x1, x2, x3, count)  
 print("Function value = %s" % f(xk.c(), fu))  
 print("Best poits is: %s" % (xk))  
 return [xk.c(), f(xk.c(), fu)]

# **Додаток Б (код програми для тестів)**

from Nelder\_Mead import out  
import unittest  
from math import exp, sqrt  
  
e = 0.00001  
  
  
class TestCalc(unittest.TestCase):  
 def test\_0(self):  
 print('Function 1: x\*\*3 + y\*\*3 - 3\*x\*y\n')  
 res = out("x\*\*3 + y\*\*3 - 3\*x\*y", [0, 0], [0, 1], [1, 0], 34)  
 self.assertTrue(abs(res[0][0] - 1) < e and abs(res[0][1] - 1) < e and abs(res[1] + 1) < e)  
  
 def test\_1(self):  
 print('\n\nFunction 2: 5\*(x - 3)\*\*2 + (y - 5)\*\*2\n')  
 res = out("5\*(x - 3)\*\*2 + (y - 5)\*\*2", [2, 6], [4, 4], [4, 3], 19)  
 self.assertTrue(abs(res[0][0] - 3) < e and abs(res[0][1] - 5) < e and abs(res[1]) < e)  
  
 def test\_2(self):  
 print('\n\nFunction 3: x\*\*2 + y\*\*2 + x\*y\n')  
 res = out("x\*\*2 + y\*\*2 + x\*y", [1, 1], [0, 1], [1, 0], 2)  
 self.assertTrue(abs(res[0][0]) < e and abs(res[0][1]) < e and abs(res[1]) < e)  
  
 def test\_3(self):  
 print('\n\nFunction 4: (x-5)\*\*2 + (y+3)\*\*2 - 7\n')  
 res = out("(x-5)\*\*2 + (y+3)\*\*2 - 7", [4, -2], [5, -2], [5, -4], 42)  
 self.assertTrue(abs(res[0][0] - 5) < e and abs(res[0][1] + 3) < e and abs(res[1] + 7) < e)  
  
 def test\_4(self):  
 print('\n\nFunction 5: x\*\*2 + 4\*y\*\*2 + 1\n')  
 res = out("x\*\*2 + 4\*y\*\*2 + 1", [1, 1], [0, 1], [1, 0], 2)  
 self.assertTrue(abs(res[0][0]) < e and abs(res[0][1]) < e and abs(res[1]-1) < e)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

# **Додаток Б (результати виконання програми)**

C:\Users\Yaros\AppData\Local\Programs\Python\Python36\python.exe "E:/учеба/4 курс/Автоматизация/4лаб/Nelder\_Mead\_test.py"

Function 1: x\*\*3 + y\*\*3 - 3\*x\*y

.Function value = -0.9999999998278093

Best poits is: (1.0000070632592704, 0.9999990617661199)

Function 2: 5\*(x - 3)\*\*2 + (y - 5)\*\*2

..Function value = 3.7833071757093184e-10

Best poits is: (2.9999913088411887, 5.000000805921928)

Function 3: x\*\*2 + y\*\*2 + x\*y

Function value = 0.0

Best poits is: (0.0, 0.0)

Function 4: (x-5)\*\*2 + (y+3)\*\*2 - 7

Function value = -6.999999999936116

Best poits is: (5.000006233222969, -3.0000050030574226)

Function 5: x\*\*2 + 4\*y\*\*2 + 1

Function value = 1.0

Best poits is: (0.0, 0.0)

..

----------------------------------------------------------------------

Ran 5 tests in 0.033s

OK

Process finished with exit code 0

# **Додаток Г (підтвердження правильності роботи алгоритму)**

Можна перевірити правильність за допомогою методички, оскільки звідти бралися функції, які були використані в тест-кейсах і там представлені значення, які мають отриматись. Програма видає наближені значення до тих, які мають бути в відповіді.