![image.png](data:image/png;base64;base64,)

# Орєхова Вікторія, КМ-82

# Модифікований метод Ньютона

# **Зміст**

* Постановка задачі
* Теоретичні відомості \*\* *Модифікований метод Ньютона* \*\* *Функція Розенброка* \*\* *Критерії закінчення*
* Реалізація методу \*\* Безумовна оптимізація
* **-** Імпорт бібліотек
* **-** Оголошення констант
* **-** Оголошення функцій та виконання розрахунків, відображення результатів \*\* Умовна оптимізація
* **-** Імпорт бібліотек
* **-** Оголошення функцій та виконання розрахунків, відображення результатів
* Висновки
* Список використаної літератури

# Постановка задачі

Дослідити збіжність модифікованого методу Ньютона при мінімізації функції Розенброка 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.

Використати метод штрафних функцій (метод зовнішньої точки) для умовної оптимізації в залежності від:

1. Розташування початкової точки (всередині/поза допустимою областю).
2. Виду допустимої області (випукла/невипукла).

# Теоретичні відомості

**Модифікований метод Ньютона**

Модифікований метод Ньютона - це спрощення звичайного методу Ньютона, в якому зменшено кількість обрахунків, але збіжність цього методу стає лінійною, а не квадратичною. Пошук точки мінімума даним методом виглядає наступним чином:

![image.png](data:image/png;base64;base64,), де

![image.png](data:image/png;base64;base64,) - довжина кроку,

![image.png](data:image/png;base64;base64,) - вектор переходу в точку.

Для розрахунку мінімума даним методом, достатньо розділити завдання на 2 задачі:

1. Пошук ![image.png](data:image/png;base64;base64,), де

![image.png](data:image/png;base64;base64,) - -обернена матриця Гессе:

![image.png](data:image/png;base64;base64,) - матриця Гессе,

![image.png](data:image/png;base64;base64,) - - вектор градієнт цільової функції.

1. Пошук ![image.png](data:image/png;base64;base64,).

**Функція Розенброка**

![image.png](data:image/png;base64;base64,)

![image.png](data:image/png;base64;base64,)

**Критерії закінчення**

![image.png](data:image/png;base64;base64,)

# Реалізація методу

# Безумовна оптимізація

**Імпорт бібліотек**

from sympy import \*  
from mpl\_toolkits.mplot3d import Axes3D  
import matplotlib.pyplot as plt  
import numpy as np   
  
def function(x\_1, x\_2):  
 f = 100 \* (((x\_1 \*\* 2) - x\_2)\*\*2) + ((x\_1 - 1) \*\* 2)  
 return f

**Оголошення констант**

x\_1 = -1.2  
x\_2 = 0  
epsilon = 0.000001  
epsilon\_kr = 0.00000001

**Оголошення функцій та виконання розрахунків, відображення результатів**

*Оголошення функцій для:*

* *похідних функції;*
* *матриці Гессе;*
* *оберненої матриці Гессе;*
* *напрямку;*
* *нового значення змінної модифікованим методом Ньютона*

def grad\_f(x\_1, x\_2):  
 df\_dx\_1 = (400 \* ((x\_1 \*\* 2) - x\_2) \* x\_1) + 2 \* (x\_1 - 1)  
 df\_dx\_2 = -1\*(200 \* ((x\_1 \*\* 2) - x\_2))  
 grad = [df\_dx\_1, df\_dx\_2]  
 return grad  
  
def hessian\_matrix(x\_1, x\_2):  
 df\_dx\_11 = 400 \* ((x\_1 \*\* 2) - x\_2) + 800 \* (x\_1\*\*2) + 2  
 df\_dx\_12 = -400 \* x\_1  
 df\_dx\_21 = -400 \* x\_1  
 df\_dx\_22 = 200   
 hessian = [[df\_dx\_11, df\_dx\_12], [df\_dx\_21, df\_dx\_22]]  
 return hessian  
  
def invertible\_hessian(x\_1, x\_2):  
 h = hessian\_matrix(x\_1, x\_2)  
 delta = (h[0][0] \* h[1][1]) - (h[0][1] \* h[1][0])  
 invertible\_hessian = [[(h[1][1]) / delta, (-1 \* h[0][1]) / delta], [(-1 \* h[1][0]) / delta, (h[0][0]) / delta]]  
 return invertible\_hessian  
  
def find\_step(x\_1, x\_2):  
 i = invertible\_hessian(x\_1, x\_2)  
 g = grad\_f(x\_1, x\_2)  
 s = [(g[0] \* i[0][0] + g[1] \* i[1][0]), (g[0] \* i[0][1] + g[1] \* i[1][1])]  
 return s  
  
def next\_x(x\_1, x\_2):  
 s=find\_step(x\_1, x\_2)  
 lamda = Symbol('lamda')  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x

*Алгоритм Свена*

Пошук послідовності точок відбувається до тих пір, поки значення функції в цих точках зменшується. Коли буде знайдено точка, в якій значення функції перевищує значення функції в попередній точці, здійснюється повернення на половину останнього кроку і обчислюється значення функції в даній точці. Останні чотири точки розташовані один від одного на рівній відстані. Оцінюючи значення функції в цих точках, на основі правила виключення інтервалів визначається інтервал, що містить мінімум заданої функції.

*Алгоритм Свена з параметром 0,001*

def func\_lamda(x\_1,x\_2, value):  
 lamda = Symbol('lamda')  
 in\_function = function(next\_x(x\_1,x\_2)[0],next\_x(x\_1,x\_2)[1] )  
 f\_lamda = lambdify(lamda, in\_function, "numpy")   
 return f\_lamda(value)  
  
def lamda(x\_1,x\_2):  
 i=0  
 delta\_lamda = 0.001\*((((x\_1\*\*2)+(x\_2\*\*2))\*\*(1/2))/((((find\_step(x\_1, x\_2)[0])\*\*2)+  
 ((find\_step(x\_1,x\_2)[1])\*\*2))\*\*(1/2)))  
 lamda = 0  
 f = func\_lamda(x\_1,x\_2,lamda)  
 new\_f = f  
 if func\_lamda(x\_1, x\_2, lamda + delta\_lamda) < func\_lamda(x\_1, x\_2, lamda - delta\_lamda):  
 new\_lamda = delta\_lamda  
 while f >= new\_f:  
 lamda = new\_lamda  
 f = new\_f  
 i = i + 1  
 new\_lamda = lamda + (2\*\*i)\*delta\_lamda  
 new\_f = func\_lamda(x\_1,x\_2,new\_lamda)  
 n\_lamda = (lamda+new\_lamda) / 2  
 n\_f = func\_lamda(x\_1, x\_2, n\_lamda)  
 else:  
 new\_lamda=-delta\_lamda  
 while f>=new\_f:  
 lamda=new\_lamda  
 f=new\_f  
 i=i+1  
 new\_lamda=lamda - (2\*\*i)\*delta\_lamda  
 new\_f=func\_lamda(x\_1,x\_2,new\_lamda)  
 n\_lamda=(lamda+new\_lamda)/2  
 n\_f=func\_lamda(x\_1,x\_2,n\_lamda)  
 if (n\_f<f):  
 a=lamda  
 b=new\_lamda  
 else:  
 a=lamda-(n\_lamda-lamda)  
 b=n\_lamda   
 return a, b

*Метод золотого перетину*

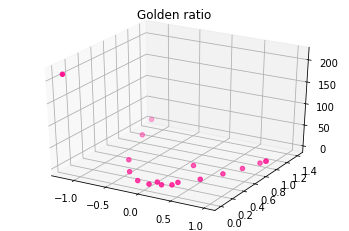
1. На першій ітерації заданий відрізок ділиться двома симетричними відносно центру точками і розраховуються значення в цих точках.
2. Після чого той з кінців відрізка, до якого серед двох знову поставлених точок ближче виявилася та, значення в якій максимальне, відкидають.
3. На наступній ітерації треба шукати лише одну нову точку, оскільки одна з двох точок послідовно одержуваних інтервалів завжди збігається з іншою точкою з пари точок попереднього інтервалу .
4. Процедура триває до тих пір, поки не буде досягнута задана точність.

def goldenratio\_mathod(x\_1, x\_2):  
 d = lamda(x\_1, x\_2)  
 a = d[0]  
 b = d[1]  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 while (b - a) > epsilon:  
 if func\_lamda(x\_1, x\_2, x) < func\_lamda(x\_1, x\_2, y):  
 a = a  
 b = y  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 if func\_lamda(x\_1, x\_2, y) <= func\_lamda(x\_1, x\_2, x):  
 a = x  
 b = b  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 lamda\_1 = (a + b) / 2  
 return lamda\_1  
def new\_x(x\_1, x\_2):  
 s = find\_step(x\_1, x\_2)  
 lamda = goldenratio\_mathod(x\_1, x\_2)  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x

*Виконання методу золотого перетину для алгоритму Свена з параметром 0,001*

def k(x\_1, x\_2):  
 x=[]  
 y=[]  
 f = []  
 old\_f = function(x\_1, x\_2)  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(old\_f)  
 print(0, '-', "%.7f" % old\_f, ':', x\_1, x\_2)  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 i=1  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', x\_1, x\_2)  
 while (abs(old\_f - new\_f)) >= epsilon:  
 i = i + 1  
 old\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 print('мінімум досягнуто за ', i ,'ітерацій з точністю 0.000001')  
 return x, y, f  
  
fige=k(x\_1, x\_2)  
x = fige[0]  
y=fige[1]  
f = fige[2]  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.scatter(x, y, f, c = 'deeppink')  
ax.set\_title('Golden ratio')  
plt.show()

0 - 212.2000000 : -1.2 0  
1 - 4.8065605 : -1.1923866240770538 1.403638675741858  
2 - 2.6611829 : -0.6293746838509133 0.3881620117286432  
3 - 2.0495976 : -0.3870681423183888 0.11437607958254892  
4 - 1.3611942 : -0.12276618121313054 -0.016644438803490563  
5 - 0.9704603 : 0.1284735107604042 -0.029418608522116238  
6 - 0.8740225 : 0.06975363740925683 0.014173764169256725  
7 - 0.6450090 : 0.23216609305930216 0.030355379204132286  
8 - 0.6008789 : 0.22483698233062285 0.05065912988470955  
9 - 0.4509243 : 0.3677908675316339 0.1126347726717753  
10 - 0.4861112 : 0.3428481978021902 0.1408392440394683  
11 - 0.2381925 : 0.5626003960808309 0.2948687783314814  
12 - 0.0866153 : 0.7257661658253564 0.5160542375876968  
13 - 0.0216858 : 0.8542213120270945 0.7276099440491612  
14 - 0.0031221 : 0.9624698500664434 0.9222086457764099  
15 - 0.0000254 : 0.9964509398579745 0.9925560989507141  
16 - 0.0000001 : 1.0001859013642824 1.000385511522132  
17 - 0.0000000 : 1.0000054982884505 1.0000110603521375  
мінімум досягнуто за 17 ітерацій з точністю 0.000001



*Алгоритм Свена з параметром 0.00001*

def lamda(x\_1,x\_2):  
 i=0  
 delta\_lamda = 0.00001\*((((x\_1\*\*2)+(x\_2\*\*2))\*\*(1/2))/((((find\_step(x\_1, x\_2)[0])\*\*2)+  
 ((find\_step(x\_1,x\_2)[1])\*\*2))\*\*(1/2)))  
 lamda = 0  
 f = func\_lamda(x\_1,x\_2,lamda)  
 new\_f = f  
 if func\_lamda(x\_1, x\_2, lamda + delta\_lamda) < func\_lamda(x\_1, x\_2, lamda - delta\_lamda):  
 new\_lamda = delta\_lamda  
 while f >= new\_f:  
 lamda = new\_lamda  
 f = new\_f  
 i = i + 1  
 new\_lamda = lamda + (2\*\*i)\*delta\_lamda  
 new\_f = func\_lamda(x\_1,x\_2,new\_lamda)  
 n\_lamda = (lamda+new\_lamda) / 2  
 n\_f = func\_lamda(x\_1, x\_2, n\_lamda)  
 else:  
 new\_lamda=-delta\_lamda  
 while f>=new\_f:  
 lamda=new\_lamda  
 f=new\_f  
 i=i+1  
 new\_lamda=lamda - (2\*\*i)\*delta\_lamda  
 new\_f=func\_lamda(x\_1,x\_2,new\_lamda)  
 n\_lamda=(lamda+new\_lamda)/2  
 n\_f=func\_lamda(x\_1,x\_2,n\_lamda)  
 if (n\_f<f):  
 a=lamda  
 b=new\_lamda  
 else:  
 a=lamda-(n\_lamda-lamda)  
 b=n\_lamda   
 return a, b

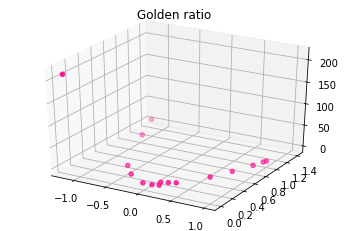
*Метод золотого перетину*

def goldenratio\_mathod(x\_1, x\_2):  
 d = lamda(x\_1, x\_2)  
 a = d[0]  
 b = d[1]  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 while (b - a) > epsilon:  
 if func\_lamda(x\_1, x\_2, x) < func\_lamda(x\_1, x\_2, y):  
 a = a  
 b = y  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 if func\_lamda(x\_1, x\_2, y) <= func\_lamda(x\_1, x\_2, x):  
 a = x  
 b = b  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 lamda\_1 = (a + b) / 2  
 return lamda\_1  
def new\_x(x\_1, x\_2):  
 s = find\_step(x\_1, x\_2)  
 lamda = goldenratio\_mathod(x\_1, x\_2)  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x

*Виконання методу золотого перетину для алгоритму Свена з параметром 0.00001*

def k(x\_1, x\_2):  
 x=[]  
 y=[]  
 f = []  
 old\_f = function(x\_1, x\_2)  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(old\_f)  
 print(0, '-', "%.7f" % old\_f, ':', x\_1, x\_2)  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 i=1  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', x\_1, x\_2)  
 while (abs(old\_f - new\_f)) >= epsilon:  
 i = i + 1  
 old\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 print('мінімум досягнуто за ', i ,'ітерацій з точністю 0.000001')  
 return x, y, f  
  
fige=k(x\_1, x\_2)  
x = fige[0]  
y=fige[1]  
f = fige[2]  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.scatter(x, y, f, c = 'deeppink')  
ax.set\_title('Golden ratio')  
plt.show()

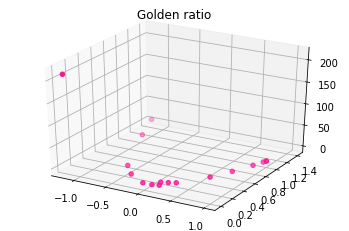
0 - 212.2000000 : -1.2 0  
1 - 4.8065605 : -1.1923866236668847 1.4036387530395689  
2 - 2.8207048 : -0.5287169614201521 0.20999097533225297  
3 - 1.8351237 : -0.3239673938472585 0.07627839665673536  
4 - 1.1763518 : -0.023539813769109463 -0.03532316895943677  
5 - 0.7818361 : 0.1696293899352948 -0.001610198016410212  
6 - 0.7943295 : 0.11277214880696207 0.021177013480460097  
7 - 0.6035240 : 0.2637043962409945 0.044762439450576744  
8 - 0.6452902 : 0.20513772697703503 0.05369363641470691  
9 - 0.4875519 : 0.35371446694282194 0.0986815819480523  
10 - 0.1324417 : 0.6366041125447259 0.40330227488773596  
11 - 0.0565311 : 0.788812198161579 0.6113018562621881  
12 - 0.0106221 : 0.9203476494352989 0.8404994418155229  
13 - 0.0005069 : 0.9818158545087413 0.9626347646319057  
14 - 0.0000008 : 1.0006190378679807 1.001303276886239  
15 - 0.0000000 : 1.000014526138671 1.0000296297034024  
мінімум досягнуто за 15 ітерацій з точністю 0.000001



*Виконання методу золотого перетину з іншою точністю*

def goldenratio\_mathod(x\_1, x\_2):  
 d = lamda(x\_1, x\_2)  
 a = d[0]  
 b = d[1]  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 while (b - a) > epsilon\_kr:  
 if func\_lamda(x\_1, x\_2, x) < func\_lamda(x\_1, x\_2, y):  
 a = a  
 b = y  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 if func\_lamda(x\_1, x\_2, y) <= func\_lamda(x\_1, x\_2, x):  
 a = x  
 b = b  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 lamda\_1 = (a + b) / 2  
 return lamda\_1  
def new\_x(x\_1, x\_2):  
 s = find\_step(x\_1, x\_2)  
 lamda = goldenratio\_mathod(x\_1, x\_2)  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x  
  
def k(x\_1, x\_2):  
 x=[]  
 y=[]  
 f = []  
 old\_f = function(x\_1, x\_2)  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(old\_f)  
 print(0, '-', "%.7f" % old\_f, ':', x\_1, x\_2)  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 i=1  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', x\_1, x\_2)  
 while (abs(old\_f - new\_f)) >= epsilon\_kr:  
 i = i + 1  
 old\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 print('мінімум досягнуто за ', i ,'ітерацій з точністю 0.00000001')  
 return x, y, f  
  
fige=k(x\_1, x\_2)  
x = fige[0]  
y=fige[1]  
f = fige[2]  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.scatter(x, y, f, c = 'deeppink')  
ax.set\_title('Golden ratio')  
plt.show()

0 - 212.2000000 : -1.2 0  
1 - 4.8065605 : -1.1923866240744398 1.4036385796958488  
2 - 2.8207056 : -0.5287169564611942 0.20999091616714993  
3 - 1.8351232 : -0.32396736629439393 0.07627845786610968  
4 - 1.1763513 : -0.023539783736181208 -0.03532311295738674  
5 - 0.7818359 : 0.1696291825986066 -0.0016101821387047056  
6 - 0.7943303 : 0.11277180900330763 0.0211770311479013  
7 - 0.6035243 : 0.2637041541778668 0.04476233067636417  
8 - 0.6452905 : 0.20513748687613692 0.053693470296584056  
9 - 0.4875522 : 0.35371418164737967 0.09868138492148974  
10 - 0.1324420 : 0.6366035781185974 0.4033019251748674  
11 - 0.0565313 : 0.788811582826475 0.6113008926070376  
12 - 0.0106222 : 0.9203469624461685 0.8404982157519357  
13 - 0.0005069 : 0.9818155893764631 0.962634226578343  
14 - 0.0000008 : 1.000619046886247 1.0013032966509599  
15 - 0.0000000 : 1.000014526105322 1.0000296296491529  
16 - 0.0000000 : 1.0000001060144226 1.0000002124440586  
мінімум досягнуто за 16 ітерацій з точністю 0.00000001



*Алгоритм Свена*

def lamd(x\_1,x\_2):  
 i=0  
 delta\_lamda = 0.00001\*((((x\_1\*\*2)+(x\_2\*\*2))\*\*(1/2))/((((find\_step(x\_1, x\_2)[0])\*\*2)+  
((find\_step(x\_1,x\_2)[1])\*\*2))\*\*(1/2)))  
 lamda = 0  
 f = func\_lamda(x\_1,x\_2,lamda)  
 new\_f = f  
 if func\_lamda(x\_1, x\_2, lamda + delta\_lamda) < func\_lamda(x\_1, x\_2, lamda - delta\_lamda):  
 new\_lamda = delta\_lamda  
 while f >= new\_f:  
 lamda = new\_lamda  
 f = new\_f  
 i = i + 1  
 new\_lamda = lamda + (2\*\*i)\*delta\_lamda  
 new\_f = func\_lamda(x\_1,x\_2,new\_lamda)  
 n\_lamda = (lamda+new\_lamda) / 2  
 n\_f = func\_lamda(x\_1, x\_2, n\_lamda)  
 else:  
 new\_lamda=-delta\_lamda  
 while f>=new\_f:  
 lamda=new\_lamda  
 f=new\_f  
 i=i+1  
 new\_lamda=lamda - (2\*\*i)\*delta\_lamda  
 new\_f=func\_lamda(x\_1,x\_2,new\_lamda)  
 n\_lamda=(lamda+new\_lamda)/2  
 n\_f=func\_lamda(x\_1,x\_2,n\_lamda)  
 if (n\_f<f):  
 a=lamda  
 b=new\_lamda  
 else:  
 a=lamda-(n\_lamda-lamda)  
 b=n\_lamda  
 return a, b

*Метод ДСК-Пауелла*

Після визначення інтервалу невизначеності методом Свена застосовується метод ДСК-Пауелла, що складається з двох етапів:

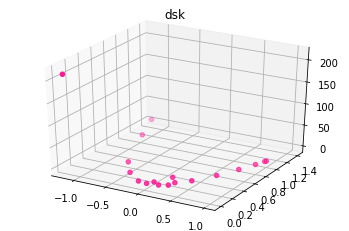
* визначення інтервалу, в якому знаходиться мінімум f (x), і одноразове оцінювання функції за допомогою квадратичної апроксимації (алгоритм ДСК);
* оцінювання функції за допомогою квадратичної апроксимації до тих пір, поки не буде досягнута необхідна точність (алгоритм Пауелла).

def dsk\_pauella(x\_1, x\_2):  
 d = lamd(x\_1,x\_2)  
 l\_1 = d[0]  
 l\_2 = (d[0] + d[1]) / 2  
 l\_3 = d[1]  
 delta\_lamda = l\_3 - l\_2  
 f\_1 = func\_lamda(x\_1, x\_2, l\_1)  
 f\_2 = func\_lamda(x\_1, x\_2, l\_2)  
 f\_3 = func\_lamda(x\_1, x\_2, l\_3)  
 l\_x = l\_2 + ((delta\_lamda \* (f\_1 - f\_3))/(2 \* (f\_1 - 2 \* f\_2 + f\_3)))  
 f\_x = func\_lamda(x\_1, x\_2, l\_x)  
 i = 0  
 while abs(l\_x - l\_2) > epsilon:  
 if l\_x < l\_2:  
 l\_3 = l\_2  
 l\_2 = l\_x  
 else:  
 l\_1 = l\_2  
 l\_2 = l\_x  
 l\_3 = l\_3  
 f\_1 = func\_lamda(x\_1, x\_2, l\_1)  
 f\_2 = func\_lamda(x\_1, x\_2, l\_2)  
 f\_3 = func\_lamda(x\_1, x\_2, l\_3)  
 a\_1 = (f\_2 - f\_1) / (l\_2 - l\_1)  
 a\_2 = (1/(l\_3 - l\_2)) \* (((f\_3-f\_1)/(l\_3 - l\_1)) - a\_1)  
 l\_x = ((l\_1 + l\_2) / 2) - (a\_1 / (2 \* a\_2))  
 i=i+1  
 lamda = l\_x  
 return lamda, i  
def new\_x(x\_1, x\_2):  
 s = find\_step(x\_1, x\_2)  
 lamda = dsk\_pauella(x\_1, x\_2)[0]  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x

*Виконання методу ДСК-Пауелла*

def k(x\_1, x\_2):  
 x=[]  
 y=[]  
 f = []  
 old\_f = function(x\_1, x\_2)  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(old\_f)  
 print(0, '-', "%.7f" % old\_f, ':', x\_1, x\_2)  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 i=1  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', x\_1, x\_2)  
 while (abs(old\_f - new\_f)) >= epsilon:  
 i = i + 1  
 old\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 print('мінімум досягнуто за ', i, 'ітерацій з точністю 0.000001')  
 return x, y, f  
  
fige=k(x\_1, x\_2)  
x = fige[0]  
y=fige[1]  
f = fige[2]  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.scatter(x, y, f, c = 'deeppink')  
ax.set\_title('dsk')  
plt.show()

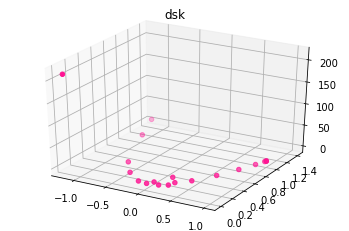
0 - 212.2000000 : -1.2 0  
1 - 4.8065605 : -1.1923866240916288 1.403638578788576  
2 - 2.6153055 : -0.5938621001416826 0.32530266528504403  
3 - 1.9750949 : -0.36909758140948246 0.10450500230501103  
4 - 1.2884997 : -0.10812205192531221 -0.012919617257261623  
5 - 0.8841180 : 0.09053544104081065 -0.015676375406516292  
6 - 0.9550435 : 0.02910837918712523 0.0119886352131411  
7 - 0.6647906 : 0.2145084845925731 0.024152128494187325  
8 - 0.6488144 : 0.19454988577984986 0.03865264962242837  
9 - 0.4978212 : 0.33851930890385146 0.09004649796321981  
10 - 0.5305104 : 0.306566160739414 0.11626731784810378  
11 - 0.3175100 : 0.4946056289999434 0.21971756207329896  
12 - 0.1107092 : 0.6824896906575316 0.4558441034636883  
13 - 0.0325747 : 0.8313701085783957 0.6847430340384679  
14 - 0.0052485 : 0.9353845676710625 0.8716680915703394  
15 - 0.0001339 : 0.9908739210610251 0.9811193918322896  
16 - 0.0000002 : 1.0002782927710874 1.0005859832265505  
17 - 0.0000000 : 1.00000613469879 1.000012406678528  
мінімум досягнуто за 17 ітерацій з точністю 0.000001



*Виконання методу ДСК-Пауелла з використанням іншого критерію закінчення*

def k(x\_1, x\_2):  
 x=[]  
 y=[]  
 f = []  
 old\_f = function(x\_1, x\_2)  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(old\_f)  
 print(0, '-', "%.7f" % old\_f, ':', x\_1, x\_2)  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 i=1  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', x\_1, x\_2)  
 while ((abs(x\_2 - x\_1))/(abs(x\_1))) >= epsilon:  
 i = i + 1  
 old\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x\_1 = new\_x(x\_1, x\_2)[0]  
 x\_2 = new\_x(x\_1, x\_2)[1]  
 new\_f = function(new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 x.append(x\_1)  
 y.append(x\_2)  
 f.append(new\_f)  
 print(i, '-', "%.7f" % new\_f, ':', new\_x(x\_1, x\_2)[0], new\_x(x\_1, x\_2)[1])  
 print('мінімум досягнуто за ', i, 'ітерацій з точністю 0.000001')  
 return x, y, f  
  
fige=k(x\_1, x\_2)  
x = fige[0]  
y=fige[1]  
f = fige[2]  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.scatter(x, y, f, c = 'deeppink')  
ax.set\_title('dsk')  
plt.show()

0 - 212.2000000 : -1.2 0  
1 - 4.8065605 : -1.1923866240916288 1.403638578788576  
2 - 2.6153055 : -0.5938621001416826 0.32530266528504403  
3 - 1.9750949 : -0.36909758140948246 0.10450500230501103  
4 - 1.2884997 : -0.10812205192531221 -0.012919617257261623  
5 - 0.8841180 : 0.09053544104081065 -0.015676375406516292  
6 - 0.9550435 : 0.02910837918712523 0.0119886352131411  
7 - 0.6647906 : 0.2145084845925731 0.024152128494187325  
8 - 0.6488144 : 0.19454988577984986 0.03865264962242837  
9 - 0.4978212 : 0.33851930890385146 0.09004649796321981  
10 - 0.5305104 : 0.306566160739414 0.11626731784810378  
11 - 0.3175100 : 0.4946056289999434 0.21971756207329896  
12 - 0.1107092 : 0.6824896906575316 0.4558441034636883  
13 - 0.0325747 : 0.8313701085783957 0.6847430340384679  
14 - 0.0052485 : 0.9353845676710625 0.8716680915703394  
15 - 0.0001339 : 0.9908739210610251 0.9811193918322896  
16 - 0.0000002 : 1.0002782927710874 1.0005859832265505  
17 - 0.0000000 : 1.00000613469879 1.000012406678528  
18 - 0.0000000 : 1.000000016349168 1.000000032772969  
19 - 0.0000000 : 1.0000000000001064 1.0000000000002134  
мінімум досягнуто за 19 ітерацій з точністю 0.000001



# Умовна оптимізація

Методи штрафних функцій використовуються для вирішення задач нелінійного програмування. Загальна вигляд задачі нелінійного програмування:

![image.png](data:image/png;base64;base64,)

Припускається, що початкова точка відома (може не належати області допустимих значень). І будується послідовність ![image.png](data:image/png;base64;base64,) - стаціонарних точок штрафних функцій. В загальному випадку, штрафна функція визначається наступним виразом:

![image.png](data:image/png;base64;base64,) , де Ф – штраф, функція від штрафного параметра R і обмежень.

**Імпорт бібліотек**

from sympy import \*  
import numpy as np  
def function(x\_1, x\_2, R):  
 f = 100 \* (((x\_1 \*\* 2) - x\_2)\*\*2) + ((x\_1 - 1) \*\* 2) + R \* (((x\_1 \*\* 2) + (x\_2 \*\* 2)- 1 ) \*\* 2)  
 return f

**Оголошення функцій та виконання розрахунків, відображення результатів**

*Оголошення функцій для:*

* *похідних функції;*
* *матриці Гессе;*
* *оберненої матриці Гессе;*
* *напрямку;*
* *нового значення змінної модифікованим методом Ньютона*

Використовуємо обмеження

![image.png](data:image/png;base64;base64,)

def grad\_f(x, y, r):  
 x\_1 = Symbol('x\_1')  
 x\_2 = Symbol('x\_2')  
 R = Symbol('R')  
 f = 100 \* (((x\_1 \*\* 2) - x\_2)\*\*2) + ((x\_1 - 1) \*\* 2) + R \* ((1 - (x\_1 \*\* 2) - (x\_2 \*\* 2)) \*\* 2)  
 d\_x1 = diff(f, x\_1)  
 d\_x2 = diff(f, x\_2)  
 d\_x1 = (lambdify((x\_1, x\_2, R), d\_x1, "numpy"))(x,y,r)  
 d\_x2 = (lambdify((x\_1, x\_2, R), d\_x2, "numpy"))(x,y,r)  
 grad = [d\_x1, d\_x2]  
 return grad  
  
def hessian\_matrix(x, y, r):  
 x\_1 = Symbol('x\_1')  
 x\_2 = Symbol('x\_2')  
 R = Symbol('R')  
 f = 100 \* (((x\_1 \*\* 2) - x\_2) \*\* 2) + ((x\_1 - 1) \*\* 2) + R \* ((1 - (x\_1 \*\* 2) - (x\_2 \*\* 2)) \*\* 2)  
 d\_x1 = diff(f, x\_1)  
 d\_x2 = diff(f, x\_2)  
 d\_x11 = diff(d\_x1, x\_1)  
 d\_x12 = diff(d\_x1, x\_2)  
 d\_x21 = diff(d\_x2, x\_1)  
 d\_x22 = diff(d\_x2, x\_2)  
 d\_x11 = (lambdify((x\_1, x\_2, R), d\_x11, "numpy"))(x, y, r)  
 d\_x12 = (lambdify((x\_1, x\_2, R), d\_x12, "numpy"))(x, y, r)  
 d\_x21 = (lambdify((x\_1, x\_2, R), d\_x21, "numpy"))(x, y, r)  
 d\_x22 = (lambdify((x\_1, x\_2, R), d\_x22, "numpy"))(x, y, r)  
 hessian = [[d\_x11, d\_x12], [d\_x21, d\_x22]]  
 return hessian  
  
def invertible\_hessian(x\_1, x\_2, R):  
 h = hessian\_matrix(x\_1, x\_2, R)  
 delta = (h[0][0] \* h[1][1]) - (h[0][1] \* h[1][0])  
 invertible\_hessian = [[(h[1][1]) / delta, (-1 \* h[0][1]) / delta], [(-1 \* h[1][0]) / delta, (h[0][0]) / delta]]  
 return invertible\_hessian  
  
def find\_step(x\_1, x\_2, R):  
 i = invertible\_hessian(x\_1, x\_2, R)  
 g = grad\_f(x\_1, x\_2, R)  
 s = [(g[0] \* i[0][0] + g[1] \* i[1][0]), (g[0] \* i[0][1] + g[1] \* i[1][1])]  
 return s  
  
def next\_x(x\_1, x\_2, R):  
 s=find\_step(x\_1, x\_2, R)  
 lamda = Symbol('lamda')  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x

*Алгоритм Свена*

def func\_lamda(x\_1,x\_2,R, value):  
 lamda = Symbol('lamda')  
 in\_function = function(next\_x(x\_1,x\_2,R)[0] ,next\_x(x\_1,x\_2,R)[1], R )  
 f\_lamda = lambdify(lamda, in\_function, "numpy")  
 return f\_lamda(value)  
  
def lamda(x\_1,x\_2, R):  
 i=0  
 delta\_lamda = 0.00001\*((((x\_1\*\*2)+(x\_2\*\*2))\*\*(1/2))/((((find\_step(x\_1, x\_2, R)[0])\*\*2)+  
((find\_step(x\_1,x\_2, R)[1])\*\*2))\*\*(1/2)))  
 lamda = 0  
 f = func\_lamda(x\_1,x\_2,R,lamda)  
 new\_f = f  
 if func\_lamda(x\_1, x\_2,R, lamda + delta\_lamda) < func\_lamda(x\_1, x\_2,R, lamda -  
delta\_lamda):  
 new\_lamda = delta\_lamda  
 while f >= new\_f:  
 lamda = new\_lamda  
 f = new\_f  
 i = i + 1  
 new\_lamda = lamda + (2\*\*i)\*delta\_lamda  
 new\_f = func\_lamda(x\_1,x\_2,R,new\_lamda)  
 n\_lamda = (lamda+new\_lamda) / 2  
 n\_f = func\_lamda(x\_1, x\_2,R, n\_lamda)  
 else:  
 new\_lamda=-delta\_lamda  
 while f>=new\_f:  
 lamda=new\_lamda  
 f=new\_f  
 i=i+1  
 new\_lamda=lamda - (2\*\*i)\*delta\_lamda  
 new\_f=func\_lamda(x\_1,x\_2,R,new\_lamda)  
 n\_lamda=(lamda+new\_lamda)/2  
 n\_f=func\_lamda(x\_1,x\_2,R,n\_lamda)  
 if (n\_f<f):  
 a=lamda  
 b=new\_lamda  
 else:  
 a=lamda-(n\_lamda-lamda)  
 b=n\_lamda  
 return a, b

*Метод золотого перетину*

Випукла область

Початкова точка всередені допустимої області

def goldenratio\_mathod(x\_1, x\_2 ,R):  
 d = lamda(x\_1, x\_2, R)  
 a = d[0]  
 b = d[1]  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 while (b - a) > 0.000001:  
 if func\_lamda(x\_1, x\_2,R, x) < func\_lamda(x\_1, x\_2,R, y):  
 a = a  
 b = y  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 if func\_lamda(x\_1, x\_2, R, y) <= func\_lamda(x\_1, x\_2, R,x):  
 a = x  
 b = b  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 lamda\_1 = (a + b) / 2  
 return lamda\_1  
  
def new\_x(x\_1, x\_2, R):  
 s = find\_step(x\_1, x\_2, R)  
 lamda = goldenratio\_mathod(x\_1, x\_2, R)  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x  
  
def k(x\_1, x\_2,R):  
 old\_f = function(x\_1, x\_2,R)  
 new\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 x\_1 = new\_x(x\_1, x\_2, R)[0]  
 x\_2 = new\_x(x\_1, x\_2, R)[1]  
 i=1  
 while (abs(old\_f - new\_f)) >= 0.00000001:  
 i = i + 1  
 old\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 x\_1 = new\_x(x\_1, x\_2, R)[0]  
 x\_2 = new\_x(x\_1, x\_2, R)[1]  
 new\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 return new\_f, i, new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1]  
   
x\_1 = -1  
x\_2 = 0  
R=1  
  
def end\_func(x\_1,x\_2,R):  
 i = 1  
 while (abs(k(x\_1,x\_2,R)[0] - k(x\_1,x\_2,R\*10)[0]))>0.01:  
 print(i, 'ітерація(R =', R,'):')  
 fige = k(x\_1, x\_2, R)  
 print('кількість підрахунків:', fige[1],', f=',fige[0], ':(', fige[2], ';', fige[3],')')  
 R = R \* 10  
 i = i+1  
end\_func(x\_1,x\_2,R)

1 ітерація(R = 1 ):  
кількість підрахунків: 14 , f= 0.042387155276155726 :( 0.8015100961579026 ; 0.6417224711478974 )  
2 ітерація(R = 10 ):  
кількість підрахунків: 4 , f= 3.1597815412988655 :( -0.7687215097002962 ; 0.5971944639988974 )

*Метод золотого перетину*

Випукла область

Початкова точка поза допустимою областю

def goldenratio\_mathod(x\_1, x\_2 ,R):  
 d = lamda(x\_1, x\_2, R)  
 a = d[0]  
 b = d[1]  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 while (b - a) > 0.000001:  
 if func\_lamda(x\_1, x\_2,R, x) < func\_lamda(x\_1, x\_2,R, y):  
 a = a  
 b = y  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 if func\_lamda(x\_1, x\_2, R, y) <= func\_lamda(x\_1, x\_2, R,x):  
 a = x  
 b = b  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 lamda\_1 = (a + b) / 2  
 return lamda\_1  
  
def new\_x(x\_1, x\_2, R):  
 s = find\_step(x\_1, x\_2, R)  
 lamda = goldenratio\_mathod(x\_1, x\_2, R)  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x  
  
def k(x\_1, x\_2,R):  
 old\_f = function(x\_1, x\_2,R)  
 new\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 x\_1 = new\_x(x\_1, x\_2, R)[0]  
 x\_2 = new\_x(x\_1, x\_2, R)[1]  
 i=1  
 while (abs(old\_f - new\_f)) >= 0.00000001:  
 i = i + 1  
 old\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 x\_1 = new\_x(x\_1, x\_2, R)[0]  
 x\_2 = new\_x(x\_1, x\_2, R)[1]  
 new\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 return new\_f, i, new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1]  
   
x\_1 = -2  
x\_2 = 0  
R=1  
  
def end\_func(x\_1,x\_2,R):  
 i = 1  
 while (abs(k(x\_1,x\_2,R)[0] - k(x\_1,x\_2,R\*10)[0]))>0.01:  
 print(i, 'ітерація(R =', R,'):')  
 fige = k(x\_1, x\_2, R)  
 print('кількість підрахунків:', fige[1],', f=',fige[0], ':(', fige[2], ';', fige[3],')')  
 R = R \* 10  
 i = i+1  
end\_func(x\_1,x\_2,R)

1 ітерація(R = 1 ):  
кількість підрахунків: 16 , f= 0.04238715527615809 :( 0.8015101080946806 ; 0.641722491401213 )  
2 ітерація(R = 10 ):  
кількість підрахунків: 5 , f= 3.159781541298866 :( -0.7687215085680306 ; 0.5971944618047148 )

Використовуємо обмеження

![image.png](data:image/png;base64;base64,)

def grad\_f(x, y, r):  
 x\_1 = Symbol('x\_1')  
 x\_2 = Symbol('x\_2')  
 R = Symbol('R')  
 f = 100 \* (((x\_1 \*\* 2) - x\_2)\*\*2) + ((x\_1 - 1) \*\* 2) + R \* (((x\_1 \*\* 2) + (x\_2 \*\* 2)- 1 ) \*\* 2)  
 d\_x1 = diff(f, x\_1)  
 d\_x2 = diff(f, x\_2)  
 d\_x1 = (lambdify((x\_1, x\_2, R), d\_x1, "numpy"))(x,y,r)  
 d\_x2 = (lambdify((x\_1, x\_2, R), d\_x2, "numpy"))(x,y,r)  
 grad = [d\_x1, d\_x2]  
 return grad  
  
def hessian\_matrix(x, y, r):  
 x\_1 = Symbol('x\_1')  
 x\_2 = Symbol('x\_2')  
 R = Symbol('R')  
 f = 100 \* (((x\_1 \*\* 2) - x\_2) \*\* 2) + ((x\_1 - 1) \*\* 2) + R \* (((x\_1 \*\* 2) + (x\_2 \*\* 2) - 1) \*\* 2)  
 d\_x1 = diff(f, x\_1)  
 d\_x2 = diff(f, x\_2)  
 d\_x11 = diff(d\_x1, x\_1)  
 d\_x12 = diff(d\_x1, x\_2)  
 d\_x21 = diff(d\_x2, x\_1)  
 d\_x22 = diff(d\_x2, x\_2)  
 d\_x11 = (lambdify((x\_1, x\_2, R), d\_x11, "numpy"))(x, y, r)  
 d\_x12 = (lambdify((x\_1, x\_2, R), d\_x12, "numpy"))(x, y, r)  
 d\_x21 = (lambdify((x\_1, x\_2, R), d\_x21, "numpy"))(x, y, r)  
 d\_x22 = (lambdify((x\_1, x\_2, R), d\_x22, "numpy"))(x, y, r)  
 hessian = [[d\_x11, d\_x12], [d\_x21, d\_x22]]  
 return hessian  
  
def invertible\_hessian(x\_1, x\_2, R):  
 h = hessian\_matrix(x\_1, x\_2, R)  
 delta = (h[0][0] \* h[1][1]) - (h[0][1] \* h[1][0])  
 invertible\_hessian = [[(h[1][1]) / delta, (-1 \* h[0][1]) / delta], [(-1 \* h[1][0]) / delta, (h[0][0]) / delta]]  
 return invertible\_hessian  
  
def find\_step(x\_1, x\_2, R):  
 i = invertible\_hessian(x\_1, x\_2, R)  
 g = grad\_f(x\_1, x\_2, R)  
 s = [(g[0] \* i[0][0] + g[1] \* i[1][0]), (g[0] \* i[0][1] + g[1] \* i[1][1])]  
 return s  
  
def next\_x(x\_1, x\_2, R):  
 s=find\_step(x\_1, x\_2, R)  
 lamda = Symbol('lamda')  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x

*Алгоритм Свена*

def func\_lamda(x\_1,x\_2,R, value):  
 lamda = Symbol('lamda')  
 in\_function = function(next\_x(x\_1,x\_2,R)[0] ,next\_x(x\_1,x\_2,R)[1], R )  
 f\_lamda = lambdify(lamda, in\_function, "numpy")  
 return f\_lamda(value)  
  
def lamda(x\_1,x\_2, R):  
 i=0  
 delta\_lamda = 0.00001\*((((x\_1\*\*2)+(x\_2\*\*2))\*\*(1/2))/((((find\_step(x\_1, x\_2, R)[0])\*\*2)+  
((find\_step(x\_1,x\_2, R)[1])\*\*2))\*\*(1/2)))  
 lamda = 0  
 f = func\_lamda(x\_1,x\_2,R,lamda)  
 new\_f = f  
 if func\_lamda(x\_1, x\_2,R, lamda + delta\_lamda) < func\_lamda(x\_1, x\_2,R, lamda -  
delta\_lamda):  
 new\_lamda = delta\_lamda  
 while f >= new\_f:  
 lamda = new\_lamda  
 f = new\_f  
 i = i + 1  
 new\_lamda = lamda + (2\*\*i)\*delta\_lamda  
 new\_f = func\_lamda(x\_1,x\_2,R,new\_lamda)  
 n\_lamda = (lamda+new\_lamda) / 2  
 n\_f = func\_lamda(x\_1, x\_2,R, n\_lamda)  
 else:  
 new\_lamda=-delta\_lamda  
 while f>=new\_f:  
 lamda=new\_lamda  
 f=new\_f  
 i=i+1  
 new\_lamda=lamda - (2\*\*i)\*delta\_lamda  
 new\_f=func\_lamda(x\_1,x\_2,R,new\_lamda)  
 n\_lamda=(lamda+new\_lamda)/2  
 n\_f=func\_lamda(x\_1,x\_2,R,n\_lamda)  
 if (n\_f<f):  
 a=lamda  
 b=new\_lamda  
 else:  
 a=lamda-(n\_lamda-lamda)  
 b=n\_lamda  
 return a, b

*Метод золотого перетину*

Невипукла область

def goldenratio\_mathod(x\_1, x\_2 ,R):  
 d = lamda(x\_1, x\_2, R)  
 a = d[0]  
 b = d[1]  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 while (b - a) > 0.000001:  
 if func\_lamda(x\_1, x\_2,R, x) < func\_lamda(x\_1, x\_2,R, y):  
 a = a  
 b = y  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 if func\_lamda(x\_1, x\_2, R, y) <= func\_lamda(x\_1, x\_2, R,x):  
 a = x  
 b = b  
 x = a + 0.382 \* (b - a)  
 y = a + 0.618 \* (b - a)  
 lamda\_1 = (a + b) / 2  
 return lamda\_1  
  
def new\_x(x\_1, x\_2, R):  
 s = find\_step(x\_1, x\_2, R)  
 lamda = goldenratio\_mathod(x\_1, x\_2, R)  
 n\_x\_1 = x\_1 - lamda \* s[0]  
 n\_x\_2 = x\_2 - lamda \* s[1]  
 n\_x = [n\_x\_1, n\_x\_2]  
 return n\_x  
  
def k(x\_1, x\_2,R):  
 old\_f = function(x\_1, x\_2,R)  
 new\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 x\_1 = new\_x(x\_1, x\_2, R)[0]  
 x\_2 = new\_x(x\_1, x\_2, R)[1]  
 i=1  
 while (abs(old\_f - new\_f)) >= 0.00000001:  
 i = i + 1  
 old\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 x\_1 = new\_x(x\_1, x\_2, R)[0]  
 x\_2 = new\_x(x\_1, x\_2, R)[1]  
 new\_f = function(new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1], R)  
 return new\_f, i, new\_x(x\_1, x\_2, R)[0], new\_x(x\_1, x\_2, R)[1]  
   
x\_1 = -1.5  
x\_2 = 0  
R=1  
  
def end\_func(x\_1,x\_2,R):  
 i = 1  
 while (abs(k(x\_1,x\_2,R)[0] - k(x\_1,x\_2,R\*10)[0]))>0.01:  
 print(i, 'ітерація(R =', R,'):')  
 fige = k(x\_1, x\_2, R)  
 print('кількість підрахунків:', fige[1],', f=',fige[0], ':(', fige[2], ';', fige[3],')')  
 R = R \* 10  
 i = i+1  
end\_func(x\_1,x\_2,R)

1 ітерація(R = 1 ):  
кількість підрахунків: 18 , f= 0.042387155276155754 :( 0.8015100958164407 ; 0.6417224711588769 )  
2 ітерація(R = 10 ):  
кількість підрахунків: 5 , f= 3.1597815412988663 :( -0.7687215084259317 ; 0.5971944617926507 )

# Висновки

У даній роботі досліджено збіжність модифікованого методу Ньютона в залежності від певних умов.

В ході дослідження було виявлено наступне:

1. За допомогою методу золотого перетину функція досягає мінімуму швидше (за 15 ітерацій), ніж за допомогою методу ДСК-Пауелла (за 17 ітерацій). Мінімальне значення функції в точці (1;1) дорівнює 0.
2. Чим менше значення E(епсилона) використовується, тобто з чим вищою точністю проводяться обчислення, тим більше ітерацій небхідно для досягнення мінімуму. Для обчислення з точністю 0.000001 знадобилося 15 ітерацій, а для 0.00000001 - 16 ітерацій.
3. Чим менше значення параметру в алгоритмі Свена, тим швидше досягається мінімум функції: для параметру 0.001 - за 17 ітерацій, для параметру 0.00001 - за 15.
4. За допомогою критерію закінчення, що визначається як модуль різниці значень нової і старої функції, мінімум досягається за 17 ітерацій, а за допомогою критерію закінчення, що визначається як норма градієнта функції, мінімум досягається за 19 ітерацій. Використання першого критерію закінчення є доцільнішим.
5. Знаходження мінімуму функції в умовній оптимізації методом зовнішньої точки при випуклій області відбувається швидше, коли початкова точка знаходиться всередині області (14 підрахунків), ніж коли вона знаходиться поза областю (16 підрахунків). Мінімальне значення функції в точці (0.8015;0.6417) дорівнює 0.04238.
6. Знаходження мінімуму функції при випуклій області відбувається швидше (14 підрахунків), ніж при невипуклій (18 підрахунків). Мінімальне значення функції в точці (0.8015;0.6417) дорівнює 0.04238.

# Список використаної літератури

1. Himmelblau D.M. “APPLIED NONLINEАR PROGRAMMIG”, McGraw-Hill Book Company, 1972;
2. Т. С. Ладогубець. Методи оптимізації - електронний конспект – Київ: НТУУ «КПІ», 2016;
3. Т. С. Ладогубець, О. Д. Фіногенов. МЕТОДИ ОПТИМІЗАЦІЇ. РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ КУРСОВОЇ РОБОТИ, КПІ ім. Ігоря Сікорського, 2019.