НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Курсова робота

із дисципліни «Дослідження операцій»

на тему: Модифікований Партан МНС

Студента групи КМ-11

Опанасюка Руслана

Керівник:

старший викладач кафедри прикладної математики Ладогубець Т.С.

Кількість балів:_____

Оцінка:_____

Київ — 2024

ЗМІСТ

ВСТУП

Тема курсової роботи :

Модифікований партан-метод найшвидшого спуску.

Актуальність теми:

Методи оптимізації широко використовуються у багатьох наукових дослідженнях і практичних застосуваннях. Одним з найпоширеніших методів є метод найшвидшого спуску, який використовується для знаходження локального мінімуму або максимуму функції. Проте, класичний метод найшвидшого спуску має свої обмеження і не завжди є ефективним у складних задачах оптимізації.

Мета роботи:

Метою цієї курсової роботи є дослідження модифікованого партан-методу найшвидшого спуску на кореневій функції. Вивчити і відобразити переваги та можливості цього модифікованого методу та порівняти його результати з класичним методом найшвидшого спуску з оптимальним кроком.

Завдання дослідно-пошукової роботи:

Для досягнення поставленої мети роботи необхідно виконати наступні завдання:

Вивчити теоретичні аспекти класичного методу найшвидшого спуску.

Аналізувати його обмеження та недоліки у контексті складних задач оптимізації.

Розглянути модифікацію партан-методу найшвидшого спуску з оптимальним кроком.

Зробити програму для проведення досліджень.

Провести числові експерименти з використанням модифікованого партан-методу найшвидшого спуску та порівняти його результати з класичним методом.

Об'єкт і предмет дослідження:

Об'єктом дослідження є метод оптимізації, метод найшвидшого спуску,а саме його модифікація.

Стислий аналіз одержаних результатів:

Модифікований партан-метод найшвидшого спуску за меншу к-сть обчислень цільової функції знаходить мінімум на кореневій функції порівняно з класичним методом .

ОСНОВНА ЧАСТИНА

Постановка задачі:

Дослідити збіжність модифікованого партан-методу найшвидшого спуску при мінімізації кореневої функції в залежності від:

Впливу величини кроку при обчисленні похідних

Впливу різницевих схем при обчисленні похідних

Впливу виду критерію закінчення

1. $\frac{\left||x^{k+1}-x^{k}|\right|}{\left||x^{k}|\right|} \leq \varepsilon$

$$\frac{\left|f(x^{k+1})-f(x^{k})\right|}{|f(x^{k})|} \leq \varepsilon$$

2. $\left|\left|\nabla f\left(x^{(k)}\right)\right|\right| \leq \varepsilon$

Впливу точності одновимірного методу (Золотий перетин)

Впливу точності одновимірного методу (ДСК-Пауелла)

Впливу виду методу одновимірного пошуку(Золотий перетин і ДСК-Пауелла)

Впливу значення параметра в алгоритмі Свена

Впливу параметрів методу

Дослідження збіжності методу

1. Для дослідження збіжності модифікованого партан-методу від величини кроку було зафіксовано такі дані:

Таблиця 1. Початкові умови

Схеми обчислення похідних	Центр
Точності МОП	0,01
Значення параметру в Свена	0,01
МОП	Золотий переріз
Критерій закінчення	Система
Похибка в критерію	0,1

Таблиця 2. Вплив величини кроку при обчисленні похідних

Величини кроку h	Точка мінімуму	Мінімум	Кількість обчислень
0,1	[[0.9996277274113663], [0.9996640429485049]]	0.019737861862187817	243
0,01	[[1.0097715181954516], [0.9994780178361147]]	0.184352757546307	153
0,001	[[1.005428258923739], [0.98701959220851]]	0.24179713421356508	152
0,0001	[[1.0051975063562861], [0.9866950382436029]]	0.2423640613338387	152

З отриманих результатів фіксуємо значення 0.01

Таблиця 3. Вплив різницевих схем при обчисленні похідних

Схема	Точка мінімуму	Мінімум	Кількість

обчислення			обчислень
Ліва	[[1.0061990896753528], [0.997248127541562]]	0.17022426294420953	151
Центр	[[1.0097715181954516], [0.9994780178361147]]	0.184352757546307	153
Права	-	-	-

З отриманих результатів фіксуємо Ліва схему обчислення

Таблиця 4. Вплив критерію закінчення

Критерій	Точка мінімуму	Мінімум	Кількість обчислень
Система	[[1.0061990896753528], [0.997248127541562]]	0.17022426294420953	151
Градієнт	-	-	-

З отриманих результатів фіксуємо критерій - Систему

Таблиця 5. Вплив точності одновимірного методу (Золотий перетин)

Точність МОП	Точка мінімуму	Мінімум	Кількість обчислень
0,1	[[1.0061990896753528], [0.997248127541562]]	0.17022426294420953	151
0,01	[[1.0105702950076165], [1.0103625440220547]]	0.10291108422050758	328
0,001	[[1.0105702950076165],	0.10291108422050758	328

	[1.0103625440220547]]		
0,0001	-	-	-

З отриманих результатів фіксуємо значення 0.1

Таблиця 6. Вплив виду методу одновимірного пошуку (Золотий перетин та ДСК-Пауелла)

МОП	Точка мінімуму	Мінімум	Кількість обчислень
Золотого перерізу	[[1.0061990896753528], [0.997248127541562]]	0.17022426294420953	151
ДСК-Пауелла	-	-	-

З отриманих результатів фіксуємо МОП Золотого перерізу.

Таблиця 7. Вплив значення параметра в алгоритмі Свена

Значення параметру Свена	Точка мінімуму	Мінімум	Кількість обчислень
0,1	[[1.0106547432420554], [0.9887367402231341]]	0.2648112063093413	121
0,01	[[1.0061990896753528], [0.997248127541562]]	0.17022426294420953	151
0,001	[[1.000601909822866], [1.0007950546615314]]	0.029283483950812397	197
0,0001	[[1.0008627761874087],	0.07275735737309483	135

	[0.9992111658175339]]		

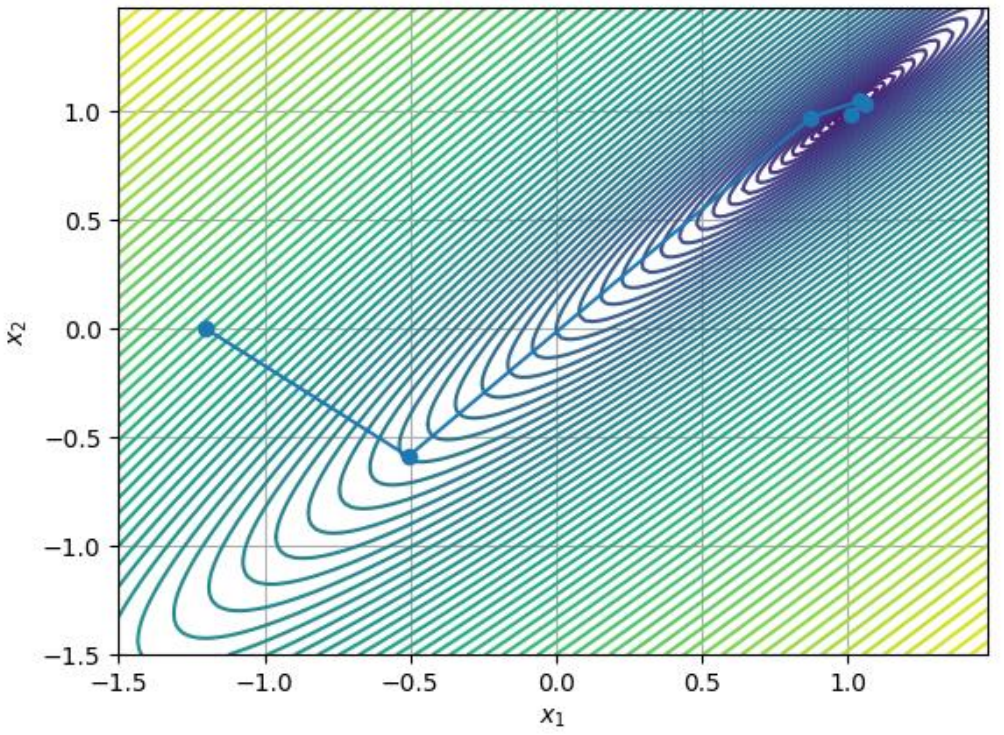
З отриманих результатів фіксуємо значення 0.1

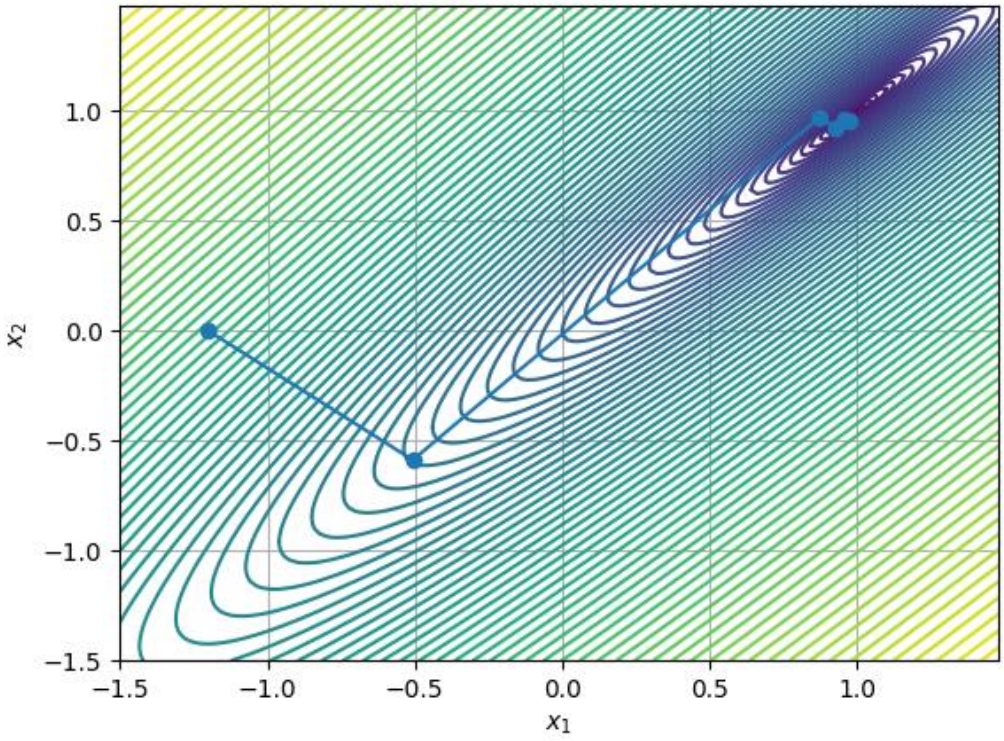
Таблиця 8. Дослідження збіжності методу

Величина похибки	Точка мінімуму	Мінімум	Кількість обчислень
0,1	-	-	-
0,01	[[1.0106547432420554], [0.9887367402231341]]	0.2648112063093413	121
0,001	[[1.0031736769405493], [1.0037427942788921]]	0.06040238174727504	240
0,0001	[[1.001597074383885], [1.0024509234262295]]	0.06040238174727504	311

З отриманих результатів фіксуємо значення 0.01

Порівняння методів: траєкторії пошуку модифікованого Партан МНС та МНС з оптимальним кроком





Таблиця 9. Вплив критерію закінчення на МНС з оптимальним кроком

Критерій	Точка мінімуму	Мінімум	Кількість обчислень
Система	[[0.9737589375014025], [0.957666841606696]]	0.23928042304599334	928
Градієнт	-	-	-

Як бачимо з таблиці 9 метод найшвидшого спуску в порівнянні з модифікованим партан МНС відчутно більше обчислює значення цільової функції(що може призводити до накопичення похибки), оскільки модифікований партан МНС відрізняється використанням додаткових напрямків, окрім напрямку антиградієнту, що забезпечує швидшу збіжність цьому методу

ВИСНОВКИ

З проведеного дослідження можна зробити наступні висновки:

Модифікований партан-метод найшвидшого спуску проявляє вищу швидкість збіжності порівняно з класичним методом найшвидшого спуску. Це досягається завдяки використанню додаткових напрямків, окрім напрямку антиградієнту, що допомагає уникнути локальних мінімумів та швидше знаходити оптимальні рішення

Порівняння модифікованого партан-методу найшвидшого спуску з методом найскорішого спуску може показати переваги та недоліки кожного з методів у різних умовах задачі.

Отже, дослідження модифікованого партан-методу найшвидшого спуску на кореневій функції показало його переваги у порівнянні з класичним методом найшвидшого спуску, а також виявило вплив різних параметрів та методів на його ефективність.

ДОДАТКИ

Код програми на мові python:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def norma(arr):
    return np.linalg.norm(arr)
```

```
f_calls = 0

def f(x):
    global f_calls
    f_calls += 1
    return (10 * (x[0] - x[1]) ** 2 + (x[0] - 1) ** 2) ** (1/4)
```

```
def gradient(x, h, diff_scheme):
    if diff_scheme == "center":
        return np.array([(f(np.array([x[0] + h, x[1]])) - f(np.array([x[0] - h,
x[1]]))) / (2 * h), (f(np.array([x[0], x[1] + h])) - f(np.array([x[0], x[1] - h])))
/ (2 * h)])
    elif diff_scheme == "right":
        return np.array([(f(np.array([x[0] + h, x[1]])) - f(x)) / h,
(f(np.array([x[0], x[1] + h])) - f(x)) / h])
    elif diff_scheme == "left":
        return np.array([(f(x) - f(np.array([x[0] - h, x[1]]))) / h, (f(x) -
f(np.array([x[0], x[1] - h]))) / h])
```

```
def first_termination_criterion(x1, x2, e):
    return ((norma(x1-x2)/norma(x2)) <= e) & (abs(f(x1)-f(x2))/abs(f(x2)) <= e)

def second_termination_criterion(x, h, diff_scheme, e):
    return norma(gradient(x, h, diff_scheme)) <= e
```

```
def DSK_Powell_method(f, a, b, e):
    xs = np.array([a, (a + b) / 2, b])
    dx = (b - a) / 2
    fx = [f(x) for x in xs]

    x_star = xs[1] + dx * (fx[0] - fx[2]) / (2 * (fx[0] - 2 * fx[1] + fx[2]))

    def end_check(x2, x_star):
        df, dx = f(x2) - f(x_star), x2 - x_star

        first_check = np.abs(df) <= e
        second_check = np.abs(dx) <= e

        return first_check and second_check

    while not end_check(xs[1], x_star):
        xs = np.append(xs, x_star)
        xs.sort()
        fxs = np.array([f(x) for x in xs])
        id_min = np.argmin(fxs)

        if id_min == 0:
            xs = xs[:3]
        elif id_min == len(xs) - 1:
            xs = xs[-3:]
        else:
            xs = xs[id_min - 1: id_min + 2]

        xs = np.unique(xs)
        if len(xs) < 3:
            print("Помилка: масив xs не містить трьох унікальних елементів")
            break



        x1, x2, x3 = xs
        a1 = (f(x2) - f(x1)) / (x2 - x1)
        a2 = 1 / (x3 - x2) * ((f(x3) - f(x1)) / (x3 - x1) - (f(x2) - f(x1)) / (x2 -
x1))
        x_star = (x1 + x2) / 2 - a1 / 2 / a2

    return x_star

def DSK_Powell_method_n(a, b, x0, s, e, f):
  f_new = lambda l: f(x0 + l * s)
  return DSK_Powell_method(f_new, a, b, e)
```

```
def golden_ration(x_a, x_b, x_k, s_k, e):
    L = x_b - x_a

    while L >= e:
        lambda_1 = x_a + 0.382 * L
        lambda_2 = x_a + 0.618 * L

        f_lambda_1 = f(x_k + lambda_1 * s_k)[0]
        f_lambda_2 = f(x_k + lambda_2 * s_k)[0]

        if f_lambda_1 < f_lambda_2:
            x_b = lambda_2
        else:
            x_a = lambda_1

        L = x_b - x_a

    return (x_a + x_b) / 2
```

```
def sven_algorithm(x, S, sven_par):
    lambd = 0
    lambd_array = np.array([lambd])
    delta_lambd = sven_par*norma(x)/norma(S)
    lambd+=delta_lambd
    f_x1 = f(x+lambd*S)
    if f_x1>f(x):
        lambd-=2*delta_lambd
        f_x1 = f(x+lambd*S)
    i = 0
    lambd_array = np.append(lambd_array, lambd)
    while True:
        if lambd>0: lambd+=delta_lambd*(2**(i+1))
        else: lambd -=delta_lambd*(2**(i+1))
        lambd_array = np.append(lambd_array, lambd)
        f_x2 = f(x+lambd*S)
        if f_x2>f_x1:
            if lambd<0: return
np.array([(lambd_array[i+2]+lambd_array[i+1])/2,lambd_array[i]])
            else: return
np.array([lambd_array[i],(lambd_array[i+2]+lambd_array[i+1])/2])
        else :
            f_x1 = f_x2
            i+=1
```

```
def optimal_mns(x, h, diff_scheme, sven_par, e, ek, opt_method=1, criterion=1):
    x_lst = [x]

    k = 0
    while True:
        S = gradient(x, h, diff_scheme)
        a, b = sven_algorithm(x, S, sven_par)
        opt_lambd = golden_ration(a, b, x, S, e)
        if opt_method == 1: opt_lambd = golden_ration(a, b, x, S, e)
        else: opt_lambd = DSK_Powell_method_n(a, b, x, S, e, f)
        x = x + opt_lambd * S
        x_lst.append(x)

        if (criterion == 1) & (first_termination_criterion(x_lst[-1], x_lst[-2],
ek)): break
        elif (criterion == 2) & (second_termination_criterion(x_lst[-1], h,
diff_scheme, ek)): break
        elif f_calls > 100000: break
        k += 1

    return x_lst
```

```
def modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek, opt_method=1,
criterion=1):
    x_lst = [x0]
    S = gradient(x0, h, diff_scheme)
    a, b = sven_algorithm(x0, S, sven_par)
    if opt_method == 1: opt_lambd = golden_ration(a, b, x0, S, e)
    else: opt_lambd = DSK_Powell_method_n(a, b, x0, S, e, f)
    x1 = x0 + opt_lambd * S
    x_lst.append(x1)

    S = gradient(x1, h, diff_scheme)
    a, b = sven_algorithm(x1, S, sven_par)
    if opt_method == 1: opt_lambd = golden_ration(a, b, x1, S, e)
    else: opt_lambd = DSK_Powell_method_n(a, b, x1, S, e, f)
    x2 = x1 + opt_lambd * S
    x_lst.append(x2)

    S = x2 - x0
    a, b = sven_algorithm(x2, S, sven_par)
    if opt_method == 1: opt_lambd = golden_ration(a, b, x2, S, e)
```

```
    else: opt_lambd = DSK_Powell_method_n(a, b, x2, S, e, f)
    x3 = x2 + opt_lambd * S
    x_lst.append(x3)

    k = 2
    while True:
        S = gradient(x_lst[-1], h, diff_scheme)
        a, b = sven_algorithm(x_lst[-1], S, sven_par)
        if opt_method == 1: opt_lambd = golden_ration(a, b, x_lst[-1], S, e)
        else: opt_lambd = DSK_Powell_method_n(a, b, x_lst[-1], S, e, f)
        x = x_lst[-1] + opt_lambd * S
        x_lst.append(x)

        S = x_lst[-1] - x_lst[k*2-3]
        a, b = sven_algorithm(x_lst[-1], S, sven_par)
        if opt_method == 1: opt_lambd = golden_ration(a, b, x_lst[-1], S, e)
        else: opt_lambd = DSK_Powell_method_n(a, b, x_lst[-1], S, e, f)
        x = x_lst[-1] + opt_lambd * S
        x_lst.append(x)

        if (criterion == 1) & (first_termination_criterion(x_lst[-1], x_lst[-2],
ek)): break
        elif (criterion == 2) & (second_termination_criterion(x_lst[-1], h,
diff_scheme, ek)): break
        elif f_calls > 100000: break
        k += 1

    return x_lst
```

```
# Початкові умови
x0 = np.array([[-1.2],
               [0]])

e  = 0.01
ek = 0.1
diff_scheme = "center"
sven_par = 0.01
```

```
# Вплив величини кроку при обчисленні похідних
for h in [0.1, 0.01, 0.001, 0.0001]:
    x = modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek)[-1]
    print(f"point: {x.tolist()}\nfunc: {f(x).item()}")
    print(f"Функція обраховувалась таку кількість разів: {str(f_calls)}\n")
    f_calls = 0
```

```
h = 0.01

# Вплив різницевих схем при обчисленні похідних
for diff_scheme in ["left", "center", "right"]:
    x = modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek)[-1]
    print(f"point: {x.tolist()}\nfunc: {f(x).item()}")
    print(f"Функція обраховувалась таку кількість разів: {str(f_calls)}\n")

    f_calls = 0
diff_scheme = "left"

# Вплив виду критерію закінчення
for criterion in [1, 2]:
    x = modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek,
criterion=criterion)[-1]
    print(f"point: {x.tolist()}\nfunc: {f(x).item()}")
    print(f"Функція обраховувалась таку кількість разів: {str(f_calls)}\n")
    f_calls = 0
```

```
# Вплив точності одновимірного методу (Золотий перетин)
for ek in [0.1, 0.01, 0.001, 0.0001]:
    x = modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek)[-1]
    print(f"point: {x.tolist()}\nfunc: {f(x).item()}")
    print(f"Функція обраховувалась таку кількість разів: {str(f_calls)}\n")
    f_calls = 0
```

```
ek = 0.1

# Вплив виду методу одновимірного пошуку (Золотий перетин та ДСК-Пауелла)
x = modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek)[-1]
print(f"point: {x.tolist()}\nfunc: {f(x).item()}")
print(f"Функція обраховувалась таку кількість разів: {str(f_calls)}\n")
f_calls = 0
```

```
# Вплив значення параметра в алгоритмі Свена
for sven_par in [0.1, 0.01, 0.001, 0.0001]:
    x = modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek)[-1]
    print(f"point: {x.tolist()}\nfunc: {f(x).item()}")
    print(f"Функція обраховувалась таку кількість разів: {str(f_calls)}\n")
    f_calls = 0
```

```
sven_par = 0.1

for e in [0.1, 0.01, 0.001, 0.0001]:
    x = modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek)[-1]
    print(f"point: {x.tolist()}\nfunc: {f(x).item()}")
    print(f"Функція обраховувалась таку кількість разів: {str(f_calls)}\n")
    f_calls = 0
```

```
e  = 0.01

x = modified_partan_mns(x0, h, diff_scheme, sven_par, e, ek)
x1 = [el[0].item() for el in x]
x2 = [el[1].item() for el in x]
fig = plt.figure()
x = np.arange(-1.5, 1.5, 0.02)
y = np.arange(-1.5, 1.5, 0.02)
X, Y = np.meshgrid(x, y)
Z = (10 * (X - Y) ** 2 + (X - 1) ** 2) ** (1/4)
plt.contour(X, Y, Z, 100)
plt.plot(x1, x2, 'o-')

plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.grid(True)
```

```
for criterion in [1, 2]:
    x = optimal_mns(x0, h, diff_scheme, sven_par, e, ek, criterion=criterion)[-1]
    print(f"point: {x.tolist()}\nfunc: {f(x).item()}")
    print(f"Функція обраховувалась таку кількість разів: {str(f_calls)}\n")
    f_calls = 0
```

```
x = optimal_mns(x0, h, diff_scheme, sven_par, e, ek)

x1 = [el[0].item() for el in x]
x2 = [el[1].item() for el in x]
fig = plt.figure()
x = np.arange(-1.5, 1.5, 0.02)
y = np.arange(-1.5, 1.5, 0.02)
X, Y = np.meshgrid(x, y)
Z = (10 * (X - Y) ** 2 + (X - 1) ** 2) ** (1/4)
```

```
plt.contour(X, Y, Z, 100)
plt.plot(x1, x2, 'o-')


plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.grid(True)
```