梯度下降法的实践

源代码

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

def f(x, y):
    return 3 * x**3 + y**4 + np.exp(y) + 5 # 定义函数

x = np.linspace(-2, 2, 1000)
y = np.linspace(-2, 2, 1000) # 均匀取点
X, Y = np.meshgrid(x, y) # 结合x，y，生成坐标矩阵
Z = f(X, Y) # 计算函数值

# print(X,Y)

fig = plt.figure(figsize=(8, 6)) # 弹出画面的尺寸
ax = fig.add_subplot(111, projection='3d') # 创建一个三维子图
surf = ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none') # 绘制一个三维表面图
fig.colorbar(surf, shrink=0.5, aspect=5) # 添加一个颜色条

ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')
ax.set_title('Surface plot of f(x, y) = 3x^3 + y^4 + e^y + 5') # 给图表加上特征

# plt.show() # 展示绘制的图表，为后面题，先注释掉

# 准备数据
data = np.column_stack((x,y,Z))
x_data = data[:, 0]  # x 坐标
y_data = data[:, 1]  # y 坐标
z_data = data[:, 2]  # z 坐标（函数值）

# 定义函数模型
def f(x, y, a, b, c, d):
    return a * x ** 3 + b * y ** 4 + c * np.exp(y) + d

# 定义损失函数
def loss(params, x, y, z):
    a, b, c, d = params
    predictions = f(x, y, a, b, c, d)
    mse = np.mean((predictions - z) ** 2)
    return mse

# 初始化a,b,c,d
params = np.array([1.0, 1.0, 1.0, 1.0])

learning_rate = 0.01    # 学习率
num_iterations = 1000   # 迭代次数
```

```
# 记录损失值的列表
loss_history = []

# 梯度下降
for i in range(num_iterations):
    # 计算预测值
    predictions = f(x_data, y_data, *params)

    # 计算损失
    mse = loss(params, x_data, y_data, z_data)

    # 记录损失值
    loss_history.append(mse)

    # 计算梯度
    grad_a = 2 * np.mean((predictions - z_data) * x_data ** 3)
    grad_b = 2 * np.mean((predictions - z_data) * y_data ** 4)
    grad_c = 2 * np.mean((predictions - z_data) * np.exp(y_data))
    grad_d = 2 * np.mean(predictions - z_data)

    # 更新参数
    params -= learning_rate * np.array([grad_a, grad_b, grad_c, grad_d])

    if i % 20 == 0:
        print(f"Iteration {i}: Loss = {mse:.4f} ")

print("Final parameters:", params)

iterations = range(len(loss_history)) # 制作一个同样1000长度的迭代次数的数组

# 绘制损失函数
plt.figure(figsize=(10,6))
plt.plot(iterations,loss_history,label='Loss')
plt.title('损失函数')
plt.xlabel('Iterations')
plt.ylabel('MSE')
plt.legend()
plt.grid(True)
plt.show()
```

截图

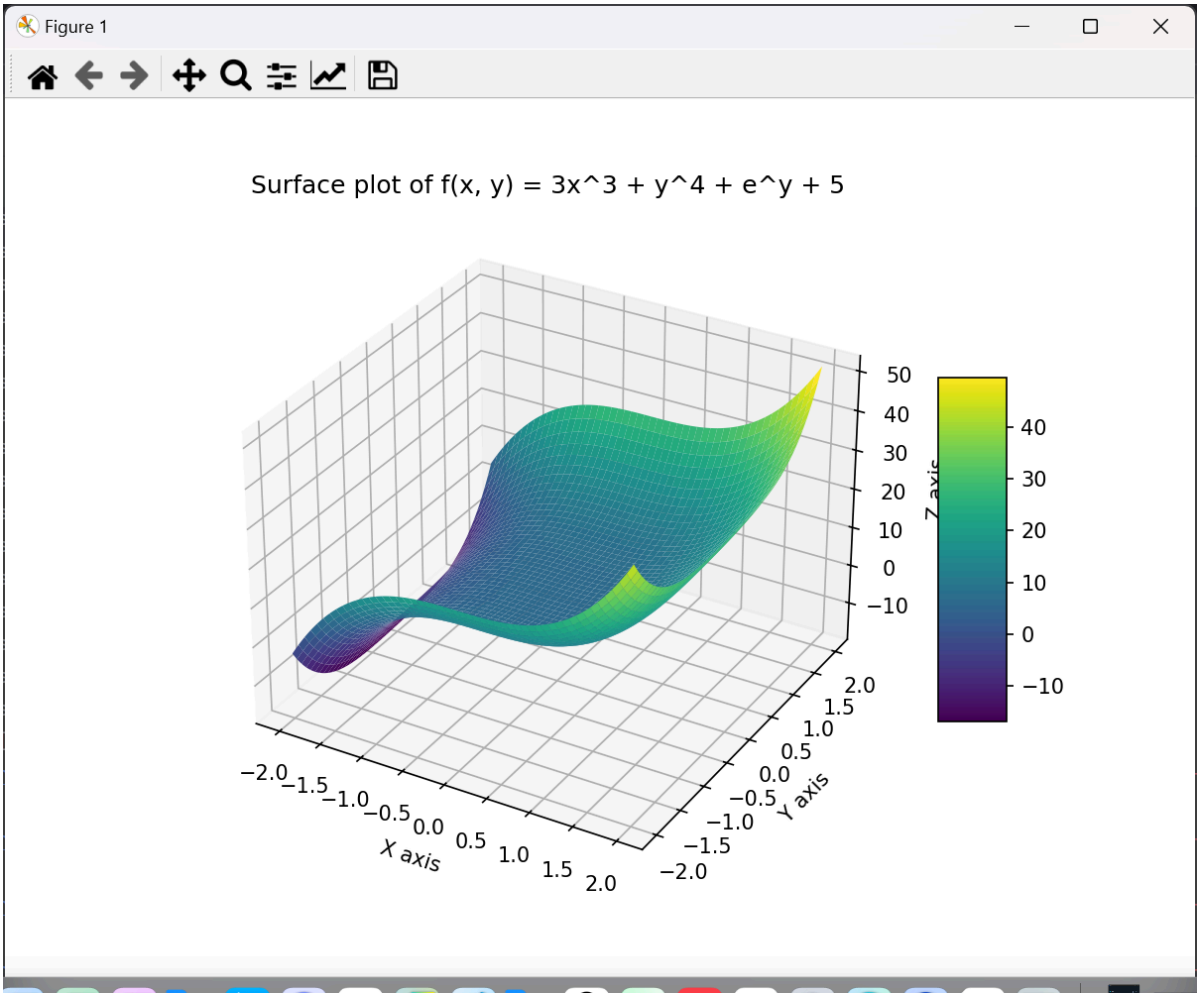
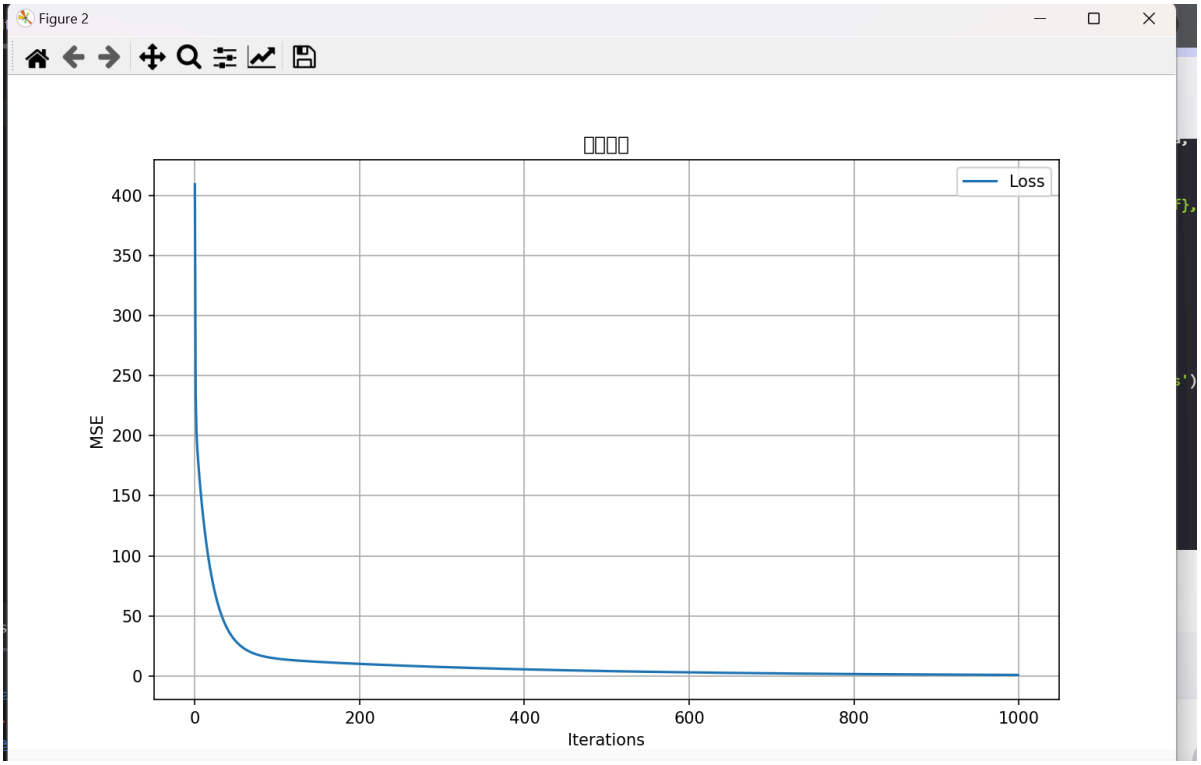


Figure 1
Surface plot of f(x, y) = 3x^3 + y^4 + e^y + 5
50
40
30
20
10
0
−10
40
30
20
10
0
−10
Z axis
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Y axis
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X axis

```
Iteration 580: Loss = 3.2739
Iteration 600: Loss = 3.0849
Iteration 620: Loss = 2.9068
Iteration 640: Loss = 2.7390
Iteration 660: Loss = 2.5809
Iteration 680: Loss = 2.4319
Iteration 700: Loss = 2.2915
Iteration 720: Loss = 2.1592
Iteration 740: Loss = 2.0345
Iteration 760: Loss = 1.9171
Iteration 780: Loss = 1.8064
Iteration 800: Loss = 1.7021
Iteration 820: Loss = 1.6039
Iteration 840: Loss = 1.5113
Iteration 860: Loss = 1.4240
Iteration 880: Loss = 1.3418
Iteration 900: Loss = 1.2643
Iteration 920: Loss = 1.1914
Iteration 940: Loss = 1.1226
Iteration 960: Loss = 1.0578
Iteration 980: Loss = 0.9967
Final parameters: [  1.07792428   1.32041565  -0.96521388 -16.25817442]
```



总结

用梯度下降拟合函数的过程其实就是前向传播和反向传播的过程，首先从输入层输入数据，经过模型后输出，计算误差，在通过反向传播传回给模型，帮助其更新参数，然后不断地重复这个过程，模型地拟合效果越来越好

我遇到的问题

在绘制**损失函数**随着迭代次数的的变化时，我直接使用 iterations 作为X轴的数据，但实际上 iterations 只是一个数，而输入的X轴需要一个数组，然后就想到了用一个新变量，利用记录的MSE的值，来给定X的个数，这样就避免了最开始出现的变量1对1000的问题

学习收获

学习了如何使用 matplotlib 来画函数，用 numpy 存储和使用数据，如何用梯度下降法去拟合一个函数