复习题

- PageRank的设计思想是通过网页之间的链接关系来衡量每个网页的重要性，并将这种重要性分配给其他网页。它是一种基于随机游走的算法，用于计算网页在互联网中的排名。
- 贝叶斯定理的内容是，在已知某个事件发生的条件下，另一个事件发生的概率可以通过该事件与前者的联合概率以及前者发生的概率来计算。它在机器学习、自然语言处理等领域有着广泛的应用，例如垃圾邮件过滤、文本分类等。
- 蒙特卡罗方法的基本原理是利用随机抽样或统计试验的方法来解决数学问题。这种方法通常用于求解复杂的数值问题，如积分、微分方程等，其优点是可以快速得到近似解，缺点是需要大量的样本才能保证结果的准确性。
- 梯度下降法的主要思想是通过沿着目标函数的负梯度方向更新参数，以最小化损失函数。通俗来说，就是不断地朝着使损失函数减小的方向调整参数，直到达到局部最优或者全局最优。这种方法常用于优化神经网络的权重和偏差，使其能够更好地拟合训练数据。
  梯度下降法是一种优化算法，它帮助我们找到一个函数的最小值。想象站在一座山上，想要下到山谷去。但是山上有雾，你看不到整个山的样子，只能感觉到脚下地势的变化。
  你可以用脚试探四周，感受哪个方向是向下倾斜最陡峭的。这就像计算梯度——梯度告诉你损失函数在给定参数点处增长最快的方向。因为我们想要减少损失，所以我们会朝相反方向走，也就是沿着负梯度的方向前进。
  梯度下降就像是你在浓雾中摸索着下山的过程，通过不断尝试和调整，最终找到一个较低的位置。在这个过程中，梯度指导了你的方向，而学习率控制了你行进的速度。

践习题

使用numpy生成服从标准正态分布的100个样本

```
np.random.seed(0)
normal_samples = np.random.randn(100)
print(normal_samples)
```

为抽样出的样本绘图展示

```
plt.hist(normal_samples, bins=20, density=True)
plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.title('Histogram of Standard Normal Distribution Samples')
plt.show()
```

计算矩阵特征值和特征向量

```
matrix_A = np.array([[2, 1], [4, 5]])
eigenvalues, eigenvectors = np.linalg.eig(matrix_A)
for i in range(len(eigenvalues)):
    print(f"Eigenvalue {i+1}: {eigenvalues[i]}")
    print(f"Eigenvector {i+1}: {eigenvectors[:, i]}\n")
```

计算协方差

```
import numpy as np

data = np.array([[1, -1, 4],
                 [2, 1, 3],
                 [1, 3, -1]])
mean = data.mean(axis=0)
normalized_data = data - mean
# 计算协方差矩阵
covariance_matrix = np.cov(normalized_data.T)
print(covariance_matrix)
```

梯度下降法

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return 0.25 * (x - 0.5)**2 + 1
def df(x):
    return x - 0.5
def gradient_descent(f, df, x0, learning_rate=0.01, max_iter=1000, tol=1e-8):
    x_history = [x0]
    for _ in range(max_iter):
        grad = df(x_history[-1])
        if abs(grad) < tol:
```

```
            break
        x_history.append(x_history[-1] - learning_rate * df(x_history[-1]))
    return x_history
x = np.linspace(-1, 2, 100)
y = f(x)
x0 = 1.5  # 初始位置
x_history = gradient_descent(f, df, x0)
plt.plot(x, y)
plt.scatter(x_history[:-1], [f(x) for x in x_history[:-1]], c='r', label='iterative points')
plt.scatter(x_history[-1], f(x_history[-1]), c='g', marker='x', s=100, label='minimum point')
plt.legend()
plt.grid(True)
plt.show()
```