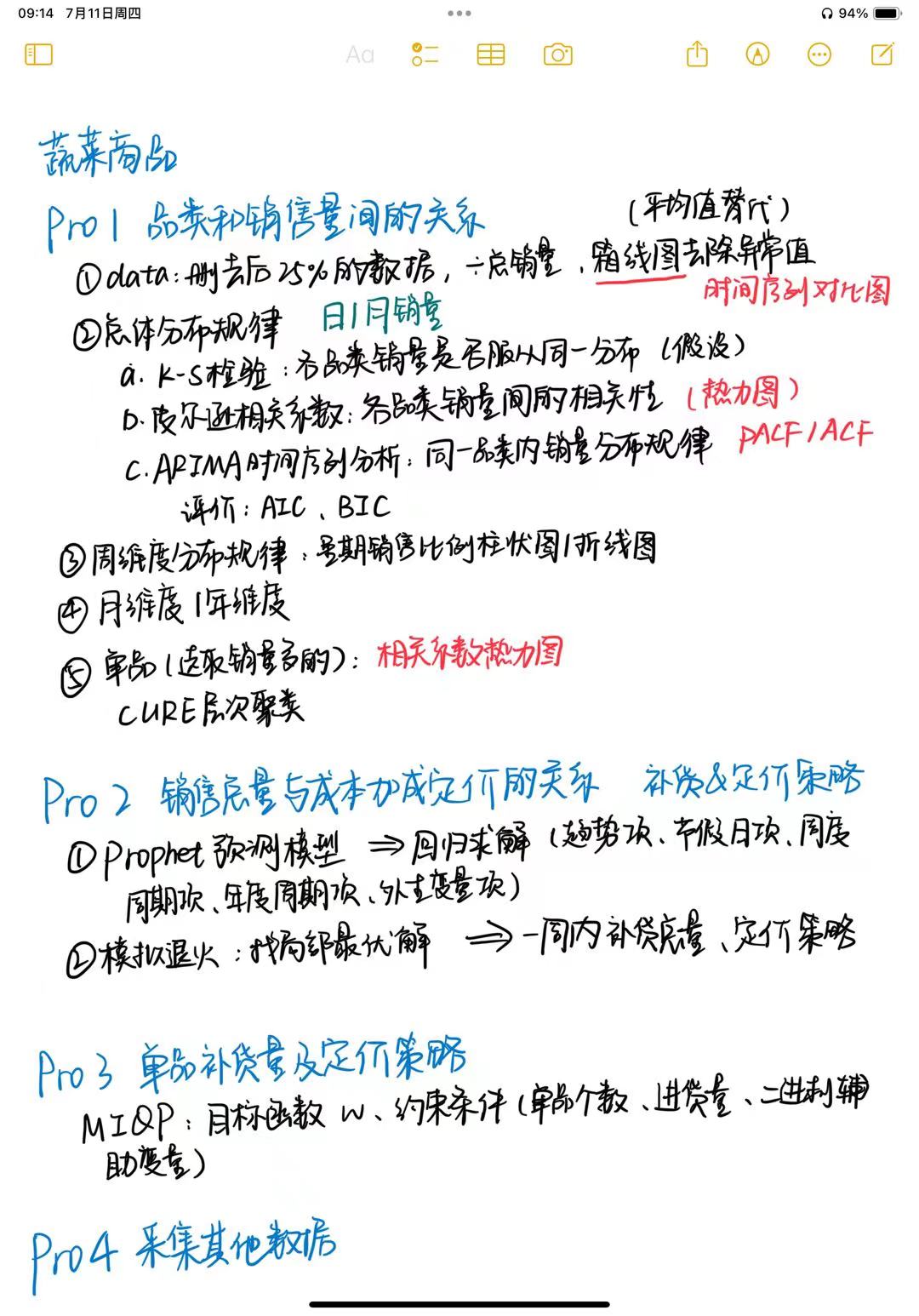
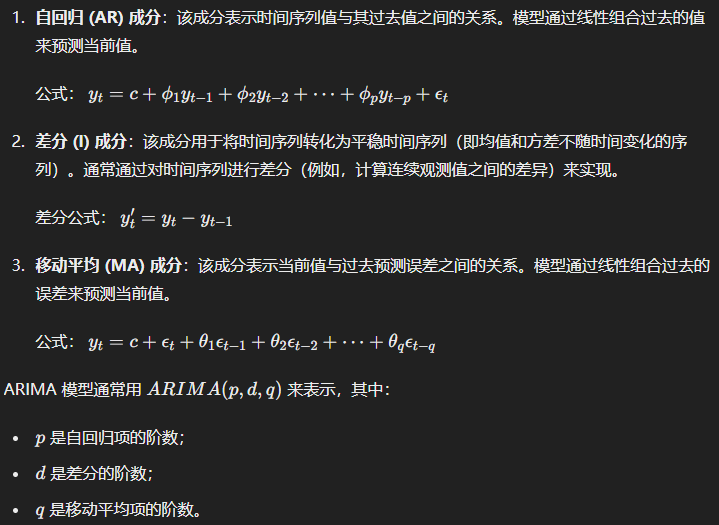
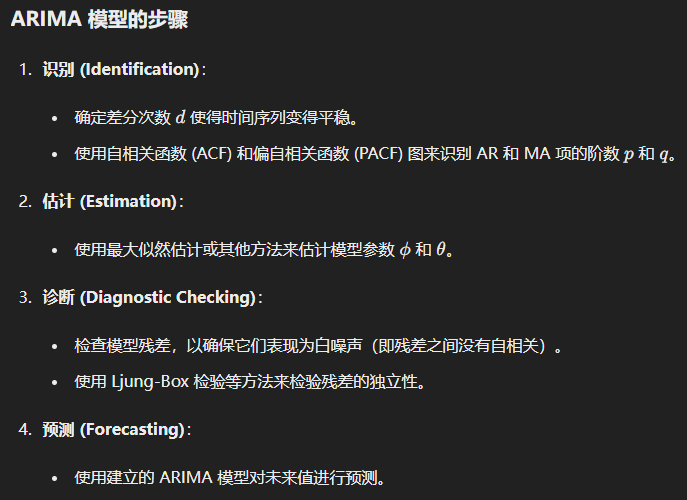
****

**ARIMA时间序列分析**

ARIMA (Autoregressive Integrated Moving Average) 是一种用于**时间序列**分析和**预测**的统计模型。它结合了**自回归 (AR)、差分 (I) 和移动平均 (MA)** 三个成分。





import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

# 加载时间序列数据

data = pd.read\_csv('your\_timeseries\_data.csv')

ts = data['column\_name']

# 画出时间序列

plt.figure(figsize=(10, 4))

plt.plot(ts)

plt.title('Time Series')

plt.show()

# 确定 ARIMA 模型参数

p = 1 # 自回归项阶数

d = 1 # 差分阶数

q = 1 # 移动平均项阶数

# 拟合 ARIMA 模型

model = ARIMA(ts, order=(p, d, q))

model\_fit = model.fit()

# 打印模型摘要

print(model\_fit.summary())

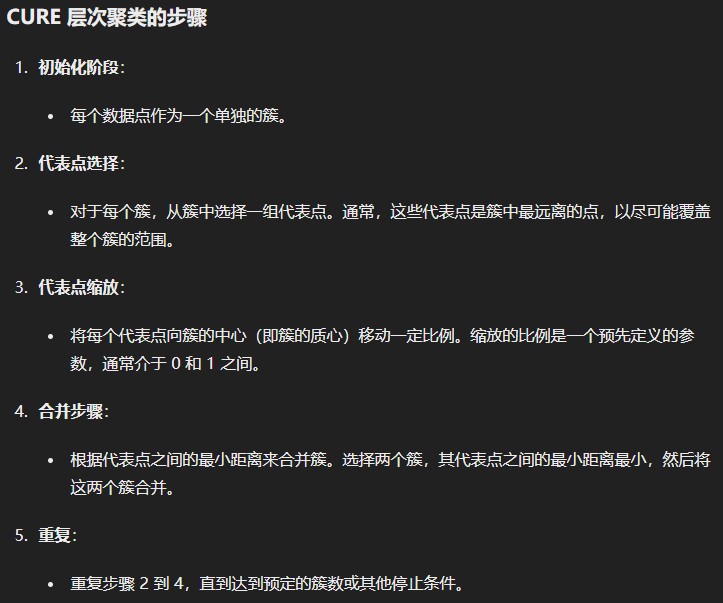
# 进行预测

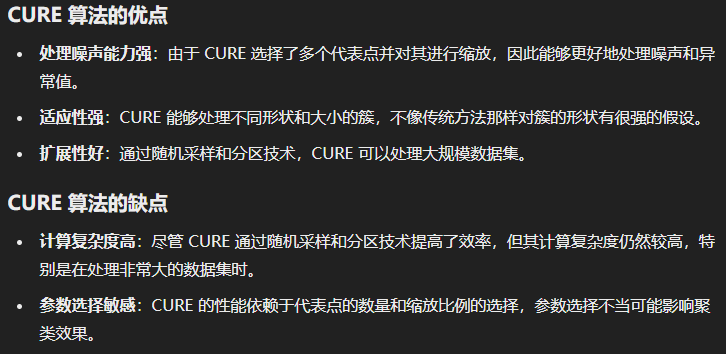
forecast = model\_fit.forecast(steps=10)

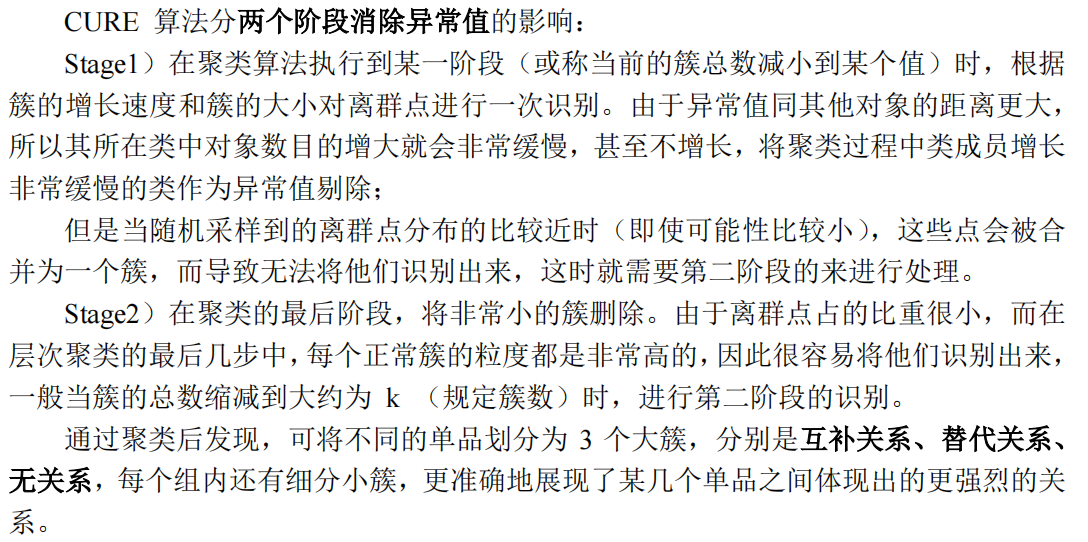
print(forecast)

**CURE 层次聚类**

CURE (Clustering Using REpresentatives) 是一种用于**大规模数据集**的层次聚类算法。与传统的层次聚类方法（如单链法、全链法）不同，CURE 通过**选择多个代表点**并对这些代表点进行缩放来**处理噪声和形状多样的数据集**。







import numpy as np

import random

def euclidean\_distance(a, b):

return np.sqrt(np.sum((a - b) \*\* 2))

class CURE:

def \_\_init\_\_(self, n\_clusters, n\_representatives=10, shrink\_factor=0.5):

self.n\_clusters = n\_clusters

self.n\_representatives = n\_representatives

self.shrink\_factor = shrink\_factor

self.clusters = []

self.representatives = []

def fit(self, X):

# 初始化每个点为一个单独的簇

self.clusters = [[x] for x in X]

self.representatives = [self.\_get\_representatives(cluster) for cluster in self.clusters]

while len(self.clusters) > self.n\_clusters:

min\_distance = float('inf')

clusters\_to\_merge = (0, 1)

# 找到最近的两个簇

for i in range(len(self.clusters)):

for j in range(i + 1, len(self.clusters)):

dist = self.\_min\_distance\_between\_clusters(self.representatives[i], self.representatives[j])

if dist < min\_distance:

min\_distance = dist

clusters\_to\_merge = (i, j)

# 合并最近的两个簇

self.\_merge\_clusters(clusters\_to\_merge[0], clusters\_to\_merge[1])

return self

def \_get\_representatives(self, cluster):

# 随机选择代表点

if len(cluster) <= self.n\_representatives:

representatives = cluster

else:

representatives = random.sample(cluster, self.n\_representatives)

# 计算簇的质心

centroid = np.mean(representatives, axis=0)

# 将代表点向质心缩放

representatives = [self.shrink\_factor \* r + (1 - self.shrink\_factor) \* centroid for r in representatives]

return representatives

def \_min\_distance\_between\_clusters(self, rep1, rep2):

return min(euclidean\_distance(r1, r2) for r1 in rep1 for r2 in rep2)

def \_merge\_clusters(self, idx1, idx2):

self.clusters[idx1].extend(self.clusters[idx2])

del self.clusters[idx2]

self.representatives[idx1] = self.\_get\_representatives(self.clusters[idx1])

del self.representatives[idx2]

# 示例数据

X = np.array([

[1, 2], [2, 3], [3, 4],

[8, 7], [8, 8], [25, 80]

])

# 初始化并运行 CURE 聚类

cure = CURE(n\_clusters=2, n\_representatives=2, shrink\_factor=0.5)

cure.fit(X)

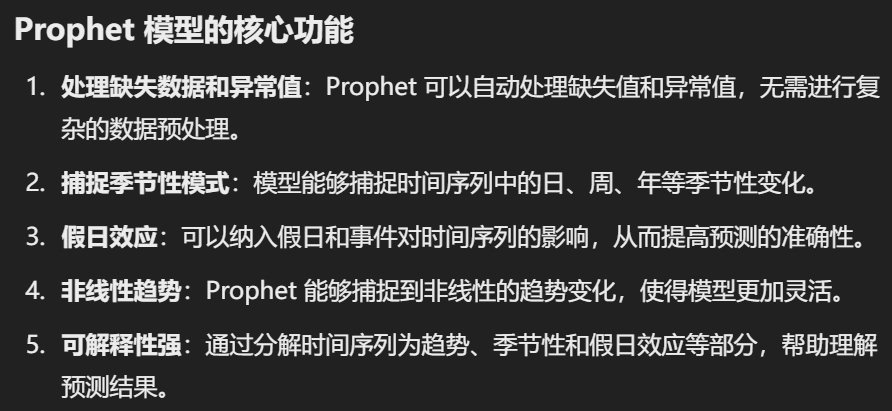
# 输出聚类结果

for i, cluster in enumerate(cure.clusters):

print(f"Cluster {i}: {cluster}")

**Prophet预测模型**

Prophet 是由 Facebook 开发的开源预测工具，旨在处理具有**强季节性**、**假日效应和缺失数据**的**时间序列**。它在建模和预测的**可解释性**方面表现优异，适用于业务分析和数据科学。



import pandas as pd

import numpy as np

from prophet import Prophet

import matplotlib.pyplot as plt

# 生成示例数据

np.random.seed(0)

data = pd.DataFrame({

'ds': pd.date\_range(start='2020-01-01', periods=365),

'y': np.random.rand(365) \* 10

})

# 初始化并训练模型

model = Prophet()

model.fit(data)

# 预测未来 30 天

future = model.make\_future\_dataframe(periods=30)

forecast = model.predict(future)

# 可视化结果

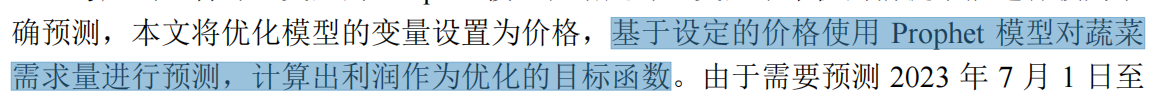
fig1 = model.plot(forecast)

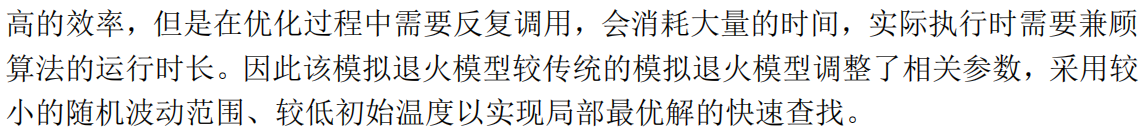
plt.show()

fig2 = model.plot\_components(forecast)

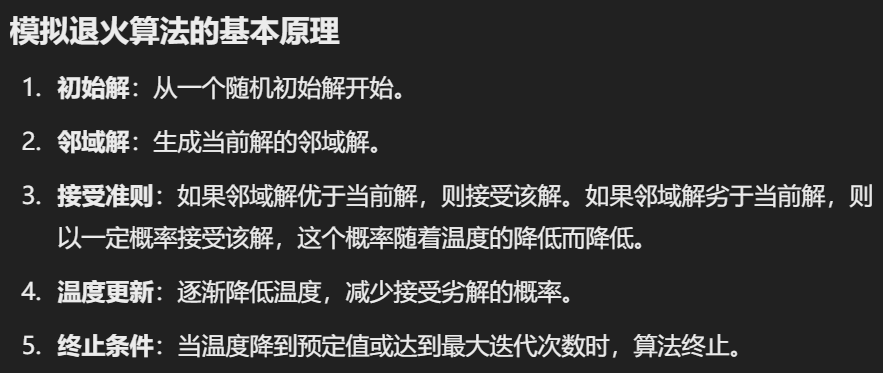
plt.show()

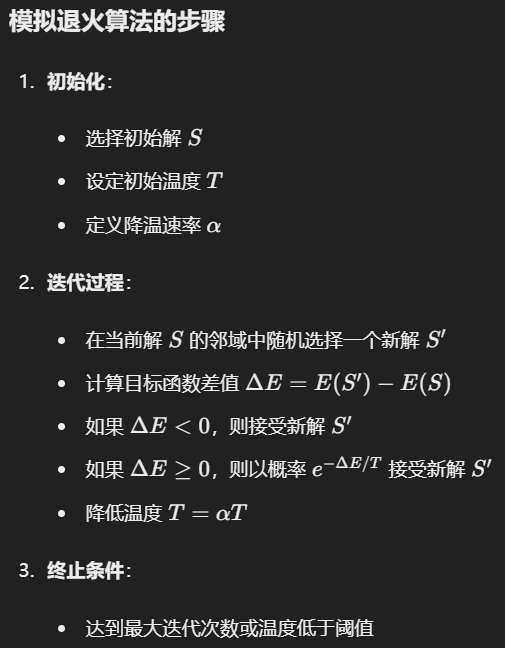
**模拟退火**





模拟退火 (Simulated Annealing, SA) 是一种**概率性**的**全局优化**算法，灵感来源于物理学中的退火过程。模拟退火算法在解决**组合优化问题**（如旅行商问题、背包问题）中表现出色。它通过允许在搜索过程中接受劣解来跳出局部最优，从而寻找到**全局最优解**。





import numpy as np

import matplotlib.pyplot as plt

# 定义目标函数

def objective\_function(x):

return x\*\*2 + 10 \* np.sin(x)

# 模拟退火算法

def simulated\_annealing(objective, bounds, n\_iterations, step\_size, temp):

# 随机初始解

best = bounds[:, 0] + (bounds[:, 1] - bounds[:, 0]) \* np.random.rand(len(bounds))

best\_eval = objective(best)

curr, curr\_eval = best, best\_eval

scores = [best\_eval]

for i in range(n\_iterations):

# 在当前解的邻域中随机选择一个新解

candidate = curr + np.random.randn(len(bounds)) \* step\_size

candidate\_eval = objective(candidate)

# 接受新解的概率

if candidate\_eval < curr\_eval or np.random.rand() < np.exp((curr\_eval - candidate\_eval) / temp):

curr, curr\_eval = candidate, candidate\_eval

if candidate\_eval < best\_eval:

best, best\_eval = candidate, candidate\_eval

# 降低温度

temp \*= 0.99

scores.append(best\_eval)

return best, best\_eval, scores

# 参数设置

bounds = np.asarray([[-10.0, 10.0]])

n\_iterations = 1000

step\_size = 0.1

temp = 10

# 运行模拟退火算法

best, best\_eval, scores = simulated\_annealing(objective\_function, bounds, n\_iterations, step\_size, temp)

print('Best Solution: %s, Objective: %.5f' % (best, best\_eval))

# 可视化结果

plt.plot(scores, label='Objective Value')

plt.xlabel('Iteration')

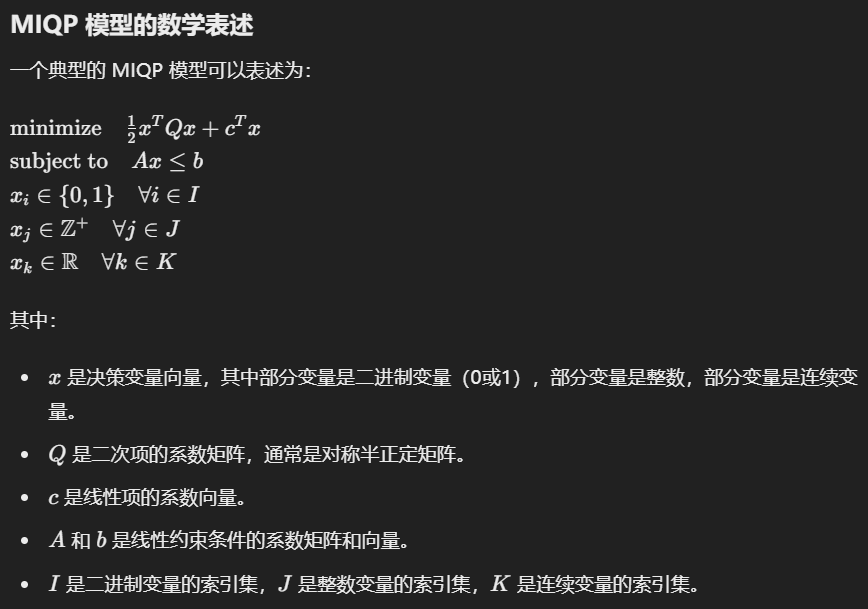
plt.ylabel('Objective Value')

plt.legend()

plt.show()

**混合整数二次规划**

混合整数二次规划（Mixed-Integer Quadratic Programming, MIQP）是一种**优化模型**，其中**目标函数是二次**的，**约束条件是线性**的，并且决策变量可以是连续的也可以是整数的。MIQP在许多应用领域有广泛应用，如金融投资组合优化、能源系统规划、生产调度等。



import gurobipy as gp

from gurobipy import GRB

import numpy as np

# 创建模型

model = gp.Model("miqp\_example")

# 添加变量

# x1 和 x2 是连续变量

x1 = model.addVar(vtype=GRB.CONTINUOUS, name="x1")

x2 = model.addVar(vtype=GRB.CONTINUOUS, name="x2")

# x3 是二进制变量

x3 = model.addVar(vtype=GRB.BINARY, name="x3")

# x4 是整数变量

x4 = model.addVar(vtype=GRB.INTEGER, name="x4")

# 设置目标函数: 0.5 \* (x1^2 + x2^2) + 2\*x1 + 3\*x2 + x3 + 4\*x4

Q = np.array([[1, 0], [0, 1]])

c = np.array([2, 3])

objective = 0.5 \* (x1 \* Q[0, 0] \* x1 + x2 \* Q[1, 1] \* x2) + c[0] \* x1 + c[1] \* x2 + x3 + 4 \* x4

model.setObjective(objective, GRB.MINIMIZE)

# 添加约束条件

model.addConstr(2 \* x1 + 3 \* x2 + x3 <= 10, "c0")

model.addConstr(x1 + x4 >= 5, "c1")

# 求解模型

model.optimize()

# 输出结果

if model.status == GRB.OPTIMAL:

print('Optimal solution found:')

print(f'x1: {x1.x}')

print(f'x2: {x2.x}')

print(f'x3: {x3.x}')

print(f'x4: {x4.x}')

else:

print('No optimal solution found')