### 15. 蒙特卡洛模拟 + 贝叶斯优化案例：投资组合配置

**问题背景**：某投资公司需优化股票、债券、基金的配置比例（总资金 1000 万元），目标是年收益率最高且风险（波动率）<10%。市场存在不确定性（如股票收益率波动大），需平衡收益与风险。

**数据**：

* 过去 5 年的资产收益率数据：股票（年均收益 15%，波动率 20%）、债券（年均收益 5%，波动率 5%）、基金（年均收益 10%，波动率 12%）。

**要求**：用贝叶斯优化搜索最优配置比例，用蒙特卡洛模拟 10000 次市场情景，输出最优比例（如股票 30%、债券 50%、基金 20%）及对应的收益分布（均值、95% 置信区间），说明如何控制风险。

### 15. 蒙特卡洛模拟 + 贝叶斯优化代码：投资组合配置

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from scipy.optimize import minimize  from skopt import gp\_minimize # 需安装：pip install scikit-optimize  from skopt.space import Real  from skopt.plots import plot\_convergence  # 1. 数据准备（资产历史收益率）  np.random.seed(42)  # 三种资产：股票、债券、基金  n\_years = 5  n\_samples = 10000 # 蒙特卡洛模拟次数  # 历史收益率特征（均值、标准差、相关性）  mean\_returns = np.array([0.15, 0.05, 0.10]) # 年均收益率  std\_dev = np.array([0.20, 0.05, 0.12]) # 波动率  corr\_matrix = np.array([  [1.0, 0.3, 0.7], # 股票与其他资产的相关性  [0.3, 1.0, 0.4], # 债券与其他资产的相关性  [0.7, 0.4, 1.0] # 基金与其他资产的相关性  ])  cov\_matrix = np.outer(std\_dev, std\_dev) \* corr\_matrix # 协方差矩阵  # 2. 蒙特卡洛模拟函数  def monte\_carlo\_simulation(weights, mean\_returns, cov\_matrix, n\_samples=10000, n\_years=1):  """模拟投资组合的年收益率分布"""  weights = np.array(weights)  # 计算组合的预期收益率和波动率  port\_mean = np.dot(weights, mean\_returns)  port\_std = np.sqrt(np.dot(weights.T, np.dot(cov\_matrix, weights)))    # 生成模拟收益率（假设服从正态分布）  yearly\_returns = np.random.multivariate\_normal(  mean\_returns, cov\_matrix, size=(n\_samples, n\_years)  )  # 计算累积收益率  port\_returns = np.prod(1 + np.dot(yearly\_returns, weights), axis=1) - 1  return port\_returns  # 3. 贝叶斯优化目标函数（最小化风险调整后的损失）  def objective\_function(weights):  """目标：最大化夏普比率（收益率/波动率），转化为最小化负夏普比率"""  weights = np.array(weights)  weights = weights / np.sum(weights) # 归一化权重    # 计算组合的预期收益率和波动率  port\_mean = np.dot(weights, mean\_returns)  port\_std = np.sqrt(np.dot(weights.T, np.dot(cov\_matrix, weights)))    # 风险约束：波动率必须<10%  if port\_std > 0.10:  return 1000 # 惩罚项    # 最大化夏普比率（假设无风险利率为0）  sharpe\_ratio = port\_mean / port\_std if port\_std !=0 else -1000  return -sharpe\_ratio # 转为最小化问题  # 4. 运行贝叶斯优化  space = [  Real(0.01, 0.98, name='股票权重'), # 股票权重范围  Real(0.01, 0.98, name='债券权重') # 债券权重范围（基金权重=1-股票-债券）  ]  def constraint\_wrapper(weights):  """包装器：将二维权重转为三维（股票、债券、基金）"""  stock\_w, bond\_w = weights  fund\_w = 1 - stock\_w - bond\_w  if fund\_w < 0.01: # 基金权重下限  return 1000  return objective\_function([stock\_w, bond\_w, fund\_w])  # 优化搜索  result = gp\_minimize(  constraint\_wrapper, space, n\_calls=50, random\_state=42  )  # 提取最优权重  stock\_w, bond\_w = result.x  fund\_w = 1 - stock\_w - bond\_w  best\_weights = np.array([stock\_w, bond\_w, fund\_w])  print(f"最优投资组合权重：")  print(f"股票：{stock\_w:.2%}")  print(f"债券：{bond\_w:.2%}")  print(f"基金：{fund\_w:.2%}")  # 5. 蒙特卡洛模拟评估最优组合  port\_returns = monte\_carlo\_simulation(best\_weights, mean\_returns, cov\_matrix, n\_samples, n\_years=1)  expected\_return = np.mean(port\_returns)  risk = np.std(port\_returns)  print(f"\n预期年收益率：{expected\_return:.2%}")  print(f"收益率波动率：{risk:.2%}")  print(f"夏普比率：{expected\_return/risk:.2f}")  print(f"亏损概率（收益率<0）：{sum(port\_returns < 0)/n\_samples:.2%}")  # 6. 可视化结果  # 优化过程收敛图  plot\_convergence(result)  plt.title('贝叶斯优化收敛曲线')  plt.show()  # 收益率分布直方图  plt.figure(figsize=(10, 6))  plt.hist(port\_returns, bins=50, alpha=0.7, color='blue')  plt.axvline(expected\_return, color='red', linestyle='--', label=f'预期收益率：{expected\_return:.2%}')  plt.xlabel('年收益率')  plt.ylabel('频率')  plt.title('最优投资组合的收益率分布（蒙特卡洛模拟）')  plt.legend()  plt.grid(axis='y', alpha=0.3)  plt.show() |