Regression

1. Install Openblas on Mac Apple M1（Linux using sudo）

brew install openblas

brew list openblas

sudo apt-get install libopenblas-dev

import ctypes

import numpy as np

import pandas as pd

import os

# 加载 OpenBLAS 库

libblas = ctypes.cdll.LoadLibrary('/opt/homebrew/Cellar/openblas/0.3.27/lib/libopenblasp-r0.3.27.dylib')

# 定义 cblas\_dgemm 函数参数和返回类型

libblas.cblas\_dgemm.restype = None

libblas.cblas\_dgemm.argtypes = [

ctypes.c\_int, # Order

ctypes.c\_int, # TransA

ctypes.c\_int, # TransB

ctypes.c\_int, # M

ctypes.c\_int, # N

ctypes.c\_int, # K

ctypes.c\_double, # alpha

ctypes.POINTER(ctypes.c\_double), # A

ctypes.c\_int, # lda

ctypes.POINTER(ctypes.c\_double), # B

ctypes.c\_int, # ldb

ctypes.c\_double, # beta

ctypes.POINTER(ctypes.c\_double), # C

ctypes.c\_int # ldc

]

# 定义矩阵乘法函数

def matrix\_multiply(A, B):

m, k = A.shape

k, n = B.shape

C = np.zeros((m, n), dtype=np.double)

alpha = ctypes.c\_double(1.0)

beta = ctypes.c\_double(0.0)

lda = ctypes.c\_int(k)

ldb = ctypes.c\_int(n)

ldc = ctypes.c\_int(n)

# 将 numpy 数组转换为 ctypes 指针类型

A\_ptr = A.ctypes.data\_as(ctypes.POINTER(ctypes.c\_double))

B\_ptr = B.ctypes.data\_as(ctypes.POINTER(ctypes.c\_double))

C\_ptr = C.ctypes.data\_as(ctypes.POINTER(ctypes.c\_double))

# 调用 cblas\_dgemm 函数

libblas.cblas\_dgemm(

ctypes.c\_int(101), # Order

ctypes.c\_int(111), # TransA

ctypes.c\_int(111), # TransB

ctypes.c\_int(m), # M

ctypes.c\_int(n), # N

ctypes.c\_int(k), # K

alpha, # alpha

A\_ptr, # A

lda, # lda

B\_ptr, # B

ldb, # ldb

beta, # beta

C\_ptr, # C

ldc # ldc

)

return C

# 定义线性回归函数

def linear\_regression(X, y):

XT = X.T

XTX = matrix\_multiply(XT, X)

XTX\_inv = np.linalg.inv(XTX)

XTy = matrix\_multiply(XT, y.reshape(-1, 1))

beta = matrix\_multiply(XTX\_inv, XTy)

return beta.flatten()

# 示例数据框

data = {

'date': [20230101, 20230101, 20230102, 20230102, 20230103],

'ticker': ['AAPL', 'IBM', 'AAPL', 'IBM', 'AAPL'],

'return': [0.01, 0.02, 0.015, 0.025, 0.03],

'weight': [0.6, 0.4, 0.6, 0.4, 0.6]

}

df = pd.DataFrame(data)

def market\_return(date, df):

daily\_data = df[df['date'] == date]

weighted\_avg = np.average(daily\_data['return'], weights=daily\_data['weight'])

return weighted\_avg

# Beta 计算

def beta\_openblas(date, ticker, df):

# 过滤ticker数据

ticker\_data = df[df['ticker'] == ticker]

# 按日期排序

ticker\_data = ticker\_data.sort\_values(by='date')

# 找到当前日期的索引

current\_index = ticker\_data[ticker\_data['date'] == date].index[0]

# 确定滚动窗口的起始索引

start\_index = max(0, current\_index - 128)

# 滚动窗口的数据

window\_data = ticker\_data[start\_index:current\_index]

if len(window\_data) < 2:

return None # 数据点不足

# 计算窗口内每个日期的市场回报

window\_data['market\_return'] = window\_data['date'].apply(lambda d: market\_return(d, df))

# 获取市场回报和个股回报

X = window\_data['market\_return'].values

y = window\_data['return'].values

# 加入常数项（截距）

X = np.vstack([X, np.ones(len(X))]).T

# 使用 OpenBLAS 进行最小二乘法回归

beta = linear\_regression(X, y)

return beta[0] # 返回 beta 值

# 示例使用

print(beta\_openblas(20230103, 'AAPL', df))

上述为OpenBlas的回归算法

1. 函数调用上没有问题
2. 算法细节还有待精细化调整

优化方面：

1. 减少数据筛选和拷贝：ticker\_data = df[df['ticker'] == ticker].sort\_values(by='date')

2. 优化矩阵操作：使用更高效的矩阵乘法和求逆方法

3. 较少循环和函数调用

其余函数实现在tower.py中

1. Correlation题目

Program1: Update.py

Program2: Pear.py

使用了多进程进行优化

再考虑到以下要求和I/O等限制，最终优化版本为up.py和pe.py

1. 可以使用文件加锁和临时文件来避免异常改动

代码在update\_lock.py, pear\_py

1. 使用临时文件同时保持备份文件，同时使用原子性文件操作

代码在up1.py, pr1.py

1. 可以将时间戳信息保存在日志文件中

代码在up2.py, pr2.py

1. 使用了多进程中文件加锁，保证了这点
2. 需要确保不同机器上的程序可以进行通信，可以使用共享文件和网络时间协议同步时间戳
3. 我们已经使用了多进程来异步处理更新，其他方法还有：分块计算（将矩阵分成较小的块分块计算），增量更新（每次只更新部分矩阵，然后计算该部分的相关系数）、缓存结果避免重复计算
4. 以上方法都适用，也可以用磁盘存储将部分数据临时存储在磁盘上

上述分块等优化实现代码在up3.py, pr3.py