# solveCAPMExercise.py

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def computeCAPMStats(r, rIdx):

# This function computes CAPM betas ("beta"), estimated systematic (i.e. market)

# risk ("sysRisk") and company-specific risk ("specRisk")

#

# INPUTS: r = TxN array of stock returns (in columns; T = number of periods, N = number of stocks)

# rIdx = Tx1 vector of market returns

#

# OUTPUTS: beta = 1xN vector of estimated beta coefficients

# sysRisk = 1xN vector of stocks' systematic risk estimates

# specRisk = 1xN vector of stocks' company-specific risk estimates

[T,N] = r.shape

# initialise variables

beta = np.full(N, np.nan)

specRisk = np.full(N, np.nan)

for i in np.arange(0,N):

X = np.hstack((np.ones((T,1)), rIdx))

Y = r[:,i]

# run time series regressions for each stock

betas = np.linalg.lstsq(X, Y, rcond=None)[0]

beta[i] = betas[1]

specRisk[i] = np.nanstd(Y - X.dot(betas))

sysRisk = beta \* np.nanstd(rIdx)

return beta, sysRisk, specRisk

datadir = 'XXX'

r = pd.read\_excel(datadir+'stockReturns.xlsx',sheet\_name='Data',usecols=[1,2,3,4])

rIdx = pd.read\_excel(datadir+'stockReturns.xlsx',sheet\_name='Data',usecols=[5])

names = r.columns.tolist()

beta, sysRisk, specRisk = computeCAPMStats(r.values, rIdx.values)

plt.figure(figsize=(10,6))

plt.subplot(3,1,1)

plt.bar(names,beta)

plt.title('Betas')

plt.subplot(3,1,2)

plt.bar(names,sysRisk)

plt.title('Systematic Risk')

plt.subplot(3,1,3)

plt.bar(names,specRisk)

plt.title('Specific Risk')

plt.show()

# solveFamaMacBethExercise.py

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def famaMacBeth(factor, returns):

[T, N] = factor.shape

gamma = np.full(T, np.nan)

for t in np.arange(1,T):

Y = returns[t,np.newaxis].T

X = np.hstack((np.ones((N,1)), factor[t-1,np.newaxis].T))

gamma[t] = np.linalg.lstsq(X, Y, rcond=None)[0][1]

tstat = np.nanmean(gamma) / (np.nanstd(gamma) / np.sqrt(T))

return gamma, tstat

datadir = 'XXX'

dates = pd.read\_excel(datadir+'uk\_data.xlsx',sheet\_name='returns',usecols=[0])

r = pd.read\_excel(datadir+'uk\_data.xlsx',sheet\_name='returns',usecols=[1,2,3,4,5])

betas = pd.read\_excel(datadir+'uk\_data.xlsx',sheet\_name='beta',usecols=[1,2,3,4,5])

names = r.columns.tolist()

[gamma, tstat] = famaMacBeth(betas.values, r.values)

plt.figure(figsize=(10,6))

plt.subplot(2,1,1)

plt.plot(gamma)

plt.title('Factor Returns')

plt.subplot(2,1,2)

plt.bar(['Mean Factor Return','T-Stat'],[np.nanmean(gamma),tstat])

plt.title('Mean Factor Return & T-Stat')

plt.show()

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def standardiseFactor(factorRaw):

# Standardises a factor cross-sectionally by first subtracting

# the cross-sectional mean from the raw factor exposures at each

# point in time and then dividing this difference by the cross-sectional

# standard deviation. Standardised factor exposures have a cross-sectional

# mean of zero and a cross-sectional standard deviation of one.

#

# INPUTS: factorRaw = TxN array of raw factor exposures, e.g. B/P ratios (where

# T is the number of time periods and N is the number of assets

#

# OUTPUTS: factorStd = TxN array of standardised factor exposures

N = factorRaw.shape[1]

# standardise factor (subtract mean from raw factor exposure and divide this difference

# by std. dev. cross-sectionally)

avg = np.tile(np.nanmean(factorRaw,axis=1,keepdims=True),(1,N))

stdev = np.tile(np.nanstd(factorRaw,axis=1,ddof=1,keepdims=True),(1,N))

return (factorRaw - avg) / stdev

# solveFamaMacBethMultiExercise.py

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from standardiseFactor import standardiseFactor

def famaMacBethMulti(returns, factors):

[T, N, K] = factors.shape

gamma = np.full((T,K), np.nan)

resid = np.full((T,N), np.nan)

for t in np.arange(1,T):

Y = returns[t,np.newaxis].T

X = np.hstack((np.ones((N,1)), factors[t-1,:,:]))

coefs = np.linalg.lstsq(X, Y, rcond=None)[0].T

gamma[t, :] = coefs[0,1:]

resid[t,:] = Y.T - X.dot(coefs.T).T

return gamma, resid

datadir = '/Users/berndhanke/Dropbox (Personal)/Cass Course (Quant Trading)/'

dates = pd.read\_excel(datadir+'uk\_factordata.xlsx',sheet\_name='Returns',usecols=[0]).values

returns = pd.read\_excel(datadir+'uk\_factordata.xlsx',sheet\_name='Returns',usecols=[1,2,3,4,5,6,7,8,9])

names = returns.columns.tolist()

returns = returns.values

factors = np.full(returns.shape+(3,), np.nan)

# Place factors into 3-dimensional array

factors[:,:,0] = pd.read\_excel(datadir+'uk\_factordata.xlsx',sheet\_name='Book2Price',usecols=[1,2,3,4,5,6,7,8,9]).values

factors[:,:,1] = pd.read\_excel(datadir+'uk\_factordata.xlsx',sheet\_name='Earnings2Price',usecols=[1,2,3,4,5,6,7,8,9]).values

factors[:,:,2] = pd.read\_excel(datadir+'uk\_factordata.xlsx',sheet\_name='Momentum',usecols=[1,2,3,4,5,6,7,8,9]).values

# obtain the number of dates (rows), number of stocks (columns) and the

# number of factors (layers in the array)

T,N,K = factors.shape

for i in np.arange(0, K):

factors[:, :, i] = standardiseFactor(factors[:, :, i])

# run Fama-MacBeth repeated cross-sectional regressions and collect

# regression coefficients (factor returns) and regression residuals

# (stock-specific returns)

gamma, residuals = famaMacBethMulti(returns, factors)

# graph cumulative factor return series

plt.plot(dates[1:],np.cumsum(gamma[1:,:],axis=0))

plt.title('Cumulative Factor Returns')

plt.legend(('Book/Price','Earnings/Price','Momentum'))

plt.show()

# extract a two-dimensional array of the latest factor exposures of each of the stocks

fac = factors[-1,:,:].reshape(N,K)

# add the systematic component of the covariance matrix estimate and the

# stock specific component to obtain the final covariance estimate

sigma\_factor = fac.dot(np.cov(gamma[1:,:],rowvar=False)).dot(fac.T) + np.diag(np.nanvar(residuals,axis=0))

# compute the covariance from historical returns directly (note: this does

# not provide a robust covariance matrix estimate whenever the number of

# stock is large relative to the number of historical return periods used)

sigma\_simple = np.cov(returns,rowvar=False)