Example

Currently, we are in the process of registering the python package with Pypi. So, for right now, to test the ZCAPM the code below will need to be present in the testing file.

For the most convenient testing experience use the ZCAPM Juptyer Notebook.ipynb file

Included on the python ZCAPM github repository is data used for testing the model. The data is as follows

1. ff_factors.csv is a file containing the Fama French factors
2. ff25_day.csv is a file containing returns for 25 size - Book-to-Market sorter portfolios
3. ind47.csv is a file containing returns for 47 industry portfolios
4. mu_sigma.csv contains returns for the equal weight market return and market sigma as discussed in the ZCAPM book

- estLinearModel(), rollapplyLM(), and LMRegression() are all used for constructing and estimating time series factor loadings for linear factor models such as the Fama French 3 factor model

- estZCAPM(), rollapplyEM(), EMRegression(), and EM_loop() are all methods used for estimating time series factor loadings for our proposed ZCAPM model with the Expectation Maxization Algorithm

- FamaMacBeth() is used for running the Fama-MacBeth test

for more information on each method use `__doc__`

In [91]:

```
'''ZCAPM Author: James. W Kolari, Wei Liu, and Jianhua Z. Huang
    Code Author : Jacob Atnip <jatnip64@gmail.com>'''
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.weightstats import ttest_ind

class Testing():
    '''Class used for testing factor models including the ZCAPM'''
    def __init__(self,print_progress = True):
        #set to True to print progress of the code when running tests
        self.print_progress = print_progress

    def EMRegression(self,dates,port_exc_ret, mkt_exc_ret, mkt_sigma, tol, MaxIter, criterion):
        '''
        Runs the EM_loop

        Parameters
        ----------
```

```
        dates : pandas series
            series containing dates for the current window of the rolling window estimation
        port_exc_ret : pandas series
            series containing excess returns for the current portfolio
        mkt_exc_ret : pandas series
            series containing exccess returns for an equal weight market index
        mkt_sigma : pandas series
            series containing cross sectional standard deviation of returns
        tol : float
            Used in the EM Algorithm for desired difference (relative or absolute) between current and prior param
            estimates
        MaxIter: int
            Max number of times the EM Algorithm should be repeated
        criterion: int
            1 : use absolute difference when comparing current and prior parameter estimates in EM Algorithm
            2 : use relative difference when comparing current and prior parameter estimates in EM Algorithm

    Returns
    -------
        1
        This method does not return anything important. Rather this method calls EM_loop which adds estimation
        results to the self.results dataframe

    '''
    #declare the portfolio name and names of beta/zeta columns. slices the portfolio returns, market returns, and
    #market sigma for the estimation period
    port_name = port_exc_ret.name
    beta_zeta_cols = [port_name+' beta',port_name+' zeta']
    date = dates.iloc[-1]
    port_exc_ret = np.array(port_exc_ret.loc[port_exc_ret.index.isin(dates)])
    mkt_exc_ret = np.array(mkt_exc_ret.loc[mkt_exc_ret.index.isin(dates)])
    mkt_sigma = np.array(mkt_sigma.loc[mkt_sigma.index.isin(dates)])

    #Initialize D and p
    resid_d = sm.OLS(port_exc_ret,mkt_exc_ret.reshape((-1,1))).fit().resid
    D = np.zeros(len(resid_d))
    D[resid_d >= 0] = 1
    D[resid_d < 0] = -1
    p_hat = np.mean(D==1)

    #Initialize beta, zeta, and sigma squared
    init_lm = sm.OLS(port_exc_ret,np.append(mkt_exc_ret.reshape((-1,1)),(D*mkt_sigma).reshape((-1,1)),axis=1)).fit
    beta_hat = init_lm.params[0]
    Z_hat = init_lm.params[1]
    sigma_sq_hat = np.mean(init_lm.resid**2)

    #Calls the EM_loop method
    self.EM_loop(beta_hat,Z_hat,sigma_sq_hat,p_hat,port_exc_ret,mkt_exc_ret,mkt_sigma,tol,MaxIter,criterion,beta_z
    return 1

def EM_loop(self, beta_hat, Z_hat, sigma_sq_hat, p_hat, port_exc_ret, mkt_exc_ret, mkt_sigma, tol, MaxIter, criter
```

```
        '''
        Fills in the self.results dataframe with beta and zeta estimates

        Parameters
        ----------
            beta_hat : float
                initial estimate of beta
            Z_hat : float
                initial estimate of zeta
            sigma_sq_hat : float
                initial estimate of sigma squared
            p_hat : float
                initial estimate of p
            port_exc_ret : numpy array
                array containing excess returns for the current portfolio during the estimation period
            mkt_exc_ret : numpy array
                array containing exccess returns for an equal weight market index for the estimation period
            mkt_sigma : numpy array
                array containing cross sectional standard deviation of returns for the estimation period
            tol : float
                Used in the EM Algorithm for desired difference (relative or absolute) between current and prior param
                estimates
            MaxIter: int
                Max number of times the EM Algorithm should be repeated
            criterion: int
                1 : use absolute difference when comparing current and prior parameter estimates in EM Algorithm
                2 : use relative difference when comparing current and prior parameter estimates in EM Algorithm
            beta_zeta_cols : list
                list of the beta and zeta column names of self.results for the current portfolio
            date : int
                date representing the current month for which the model is being fit

        Returns
        -------
            This method does not return anything important. Rather this method fills in the self.results dataframe wit
            with estimates of beta and zeta

        '''
        #Heart of the EM Algorithm. See book for additional details on steps involved in this algorithm
        delta = 1
        cnt = 0
        flag = True

        while flag:
            cnt += 1

            eta_pos = np.exp((-(port_exc_ret - beta_hat*mkt_exc_ret - Z_hat*mkt_sigma)**2)/(2*sigma_sq_hat))
            eta_neg = np.exp((-(port_exc_ret - beta_hat*mkt_exc_ret + Z_hat*mkt_sigma)**2)/(2*sigma_sq_hat))
            p_hat_t = eta_pos*p_hat/(eta_pos * p_hat + eta_neg*(1-p_hat))

            D_hat_t = 2*p_hat_t - 1
```

```
            LHS11 = np.sum(mkt_exc_ret**2)
            LHS21 = np.sum(D_hat_t * mkt_exc_ret * mkt_sigma)
            LHS22 = np.sum(mkt_sigma**2)
            RHS1 = np.sum(port_exc_ret*mkt_exc_ret)
            RHS2 = np.sum(D_hat_t*port_exc_ret*mkt_sigma)
            beta_hat_new = (LHS22*RHS1 - LHS21*RHS2)/(LHS11*LHS22 - LHS21**2)
            Z_hat_new = (LHS11*RHS2 - LHS21*RHS1)/(LHS11*LHS22 - LHS21**2)

            if Z_hat_new <0:
                Z_hat_new = -Z_hat_new

            sigma_sq_hat_new = np.mean((port_exc_ret - beta_hat*mkt_exc_ret - Z_hat_new*D_hat_t*mkt_sigma)**2 + (Z_hat_

            p_hat_new = np.mean(p_hat_t)

            self.diff = np.zeros(4)

            if criterion == 1:
                self.diff[0] = (beta_hat_new - beta_hat)/abs(beta_hat)
                self.diff[1] = (Z_hat_new - Z_hat)/abs(Z_hat)
                self.diff[2] = (sigma_sq_hat_new - sigma_sq_hat)/abs(sigma_sq_hat)
                self.diff[3] = (p_hat_new - p_hat)/abs(p_hat)

            if criterion == 2:
                self.diff[0] = (beta_hat_new - beta_hat)/(abs(beta_hat) + 1)
                self.diff[1] = (Z_hat_new - Z_hat)/(abs(Z_hat) + 1)
                self.diff[2] = (sigma_sq_hat_new - sigma_sq_hat)/(abs(sigma_sq_hat) + 1)
                self.diff[3] = (p_hat_new - p_hat)/(abs(p_hat) + 1)

            delta = max(abs(self.diff))
            if (delta < tol) or (cnt > MaxIter):
                flag = False

            beta_hat = beta_hat_new
            Z_hat = Z_hat_new
            sigma_sq_hat = sigma_sq_hat_new
            p_hat = p_hat_new

        #collect final estimates of beta and zeta and then add them to the correct row and column of self.results
        Z_star = Z_hat * (2*p_hat - 1)
        out = [beta_hat,Z_star]
        self.results.loc[date,beta_zeta_cols] = out

    def LMRegression(self,dates,port_exc_ret,factors):
        '''
        Fills in the self.results dataframe with estimates (using OLS) for the factor loadings of a linear model

        Parameters
        ----------
            dates : pandas series
```

```
                series containing dates for the current window of the rolling window estimation
            port_exc_ret : pandas series
                series containing excess returns for the current portfolio
            factors : pandas dataframe
                dataframe containing factor returns

        Returns
        -------
            1
            This method does not return anything important. Rather this method fills in the self.results dataframe wit
            estimates for the factor loadings

        '''
        #slice the portfolio returns and factor returns for the estimation period
        date = dates.iloc[-1]
        port_name = port_exc_ret.name
        port = np.array(port_exc_ret.loc[port_exc_ret.index.isin(dates)])
        facs = np.array(factors.loc[factors.index.isin(dates),:])

        #fit the linear model using OLS and collect parameters from the regression results
        res = sm.OLS(port,facs).fit().params
        self.results.loc[date,[j for j in self.results.columns if port_name in j]] = list(res)
        return 1

    def rollapplyLM(self,port_exc_ret,factors,width):
        '''
        Creates a rolling window for the current portfolio and calls LMRegression

        Parameters
        ----------
            port_exc_ret : pandas series
                series containing excess returns for the current portfolio
            factors : pandas dataframe
                dataframe containing factor returns
            width : int
                specifies the width of the rolling window in months

        Returns
        -------
            1
            This method does not return anything important. Rather this method calls the LM Regression method.

        '''
        #create a dates series to roll on and apply the LMRegression function
        dates = pd.Series(port_exc_ret.index.unique()).iloc[:-1]
        if self.print_progress:
            print('Fitting linear model for '+port_exc_ret.name,end='\r')
        dates.rolling(width).apply(self.LMRegression,args = (port_exc_ret,factors))
        return 1

    def estLinearModel(self,port_exc_ret,factors,width):
```

```
    '''
    Calls the rollapplyLM function for applying the LMRegression function

    Parameters
    ----------
        port_exc_ret : pandas dataframe
            dataframe containing excess returns for the portfolios
        factors : pandas dataframe
            dataframe containing factor returns
        width : int
            specifies the width of the rolling window in months

    Returns
    -------
        self.results : pandas dataframe
            contains estimates of factor loadings for each portfolio

    '''
    #create a dataframe that will store results of the factor loading estimation. Columns of dataframe are combina
    #of each portfolio name and for each factor name
    columns = []
    for port_name in port_exc_ret.columns:
        for fac_name in factors.columns:
            columns.append(port_name+' '+fac_name)
    self.results = pd.DataFrame([[0 for i in range(len(columns))] for i in range(len(port_exc_ret.index.unique()[w

    #apply the rollapplyLM function to each column of the portfolio returns
    port_exc_ret.apply(self.rollapplyLM,axis = 0,args = (factors,width))

    return self.results

def rollapplyEM(self,port_exc_ret,mkt_exc_ret,mkt_sigma,width,tol,MaxIter,criterion):
    '''
    Creates a rolling window for the current portfolio and calls EMRegression

    Parameters
    ----------
        port_exc_ret : pandas datafrane
            dataframe containing excess returns for the portfolios
        mkt_exc_ret : pandas series
            series containing exccess returns for an equal weight market index
        mkt_sigma : pandas series
            series containing cross sectional standard deviation of returns
        tol : float
            Used in the EM Algorithm for desired difference (relative or absolute) between current and prior param
            estimates
        MaxIter: int
            Max number of times the EM Algorithm should be repeated
        criterion: int
            1 : use absolute difference when comparing current and prior parameter estimates in EM Algorithm
            2 : use relative difference when comparing current and prior parameter estimates in EM Algorithm
```

```
        Returns
        -------
            This method does not return anything important. Rather this method calls the EM Regression method.

        '''
        #create a data pandas series to roll on and apply the EMRegression method
        dates = pd.Series(port_exc_ret.index.unique()).iloc[:-1]
        if self.print_progress:
            print('Fitting ZCAPM model for '+port_exc_ret.name,end='\r')
        dates.rolling(width).apply(self.EMRegression,args = (port_exc_ret, mkt_exc_ret,mkt_sigma,tol,MaxIter,criterion

    def estZCAPM(self,port_exc_ret,mkt_exc_ret,mkt_sigma,tol,MaxIter,criterion,width):
        '''
        Calls the rollapplyEM function for applying the EMRegression function

        Parameters
        ----------
            port_exc_ret : pandas datafrane
                dataframe containing excess returns for the portfolios
            mkt_exc_ret : pandas series
                series containing exccess returns for an equal weight market index
            mkt_sigma : pandas series
                series containing cross sectional standard deviation of returns
            tol : float
                Used in the EM Algorithm for desired difference (relative or absolute) between current and prior param
                estimates
            MaxIter: int
                Max number of times the EM Algorithm should be repeated
            criterion: int
                1 : use absolute difference when comparing current and prior parameter estimates in EM Algorithm
                2 : use relative difference when comparing current and prior parameter estimates in EM Algorithm

        Returns
        -------
            self.results : pandas dataframe
                contains estimates of factor loadings for each portfolio

        '''
        #creates a dataframe to store the results of estimating the ZCAPM model. Columns are combinations of each port
        #name and each factor (beta and zeta)
        columns = []
        for port_name in port_exc_ret.columns:
            betap = port_name +' beta'
            zetap = port_name +' zeta'
            columns.append(betap)
            columns.append(zetap)

        self.results = pd.DataFrame([[0 for i in range(len(port_exc_ret.columns)*2)] for i in range(len(port_exc_ret.i
        #apply the rollapplyEM function to each column of portfolio returns
        port_exc_ret.apply(self.rollapplyEM,axis = 0,args = (mkt_exc_ret,mkt_sigma,width,tol,MaxIter,criterion))
```

```
        return self.results

    def FamaMacBeth(self,port_exc_ret_mon,factor_loadings,factor_list,model_name):
        '''
        Runs the Fama-Macbeth Regression

        Parameters
        ----------
            port_exc_ret_mon : pandas datafrane
                dataframe containing excess monthly returns for the portfolios
            factor_loadings : pandas dataframe
                dataframe containing factor loadings for each portfolio
            factor_list : list
                names of each factor in the model
            model_name : string
                name of the model

        Returns
        -------
            results : pandas dataframe
                contains regression results of the Fama MacBeth test

        '''
        #create dataframe to hold the regression results of the Fama-MacBeth test
        results = pd.DataFrame([[0 for j in range(len(factor_list)+1)] for i in range(len(port_exc_ret_mon.index))],in
        cols_list = []
        for factor in factor_list:
            cols = factor_loadings.columns[factor_loadings.columns.str.contains(factor)]
            cols_list.append(cols)

        #Perform the Fama-MacBeth test. Use simple for loop rather than rolling apply is a neglible difference in spee
        for i,date in enumerate(port_exc_ret_mon.index):
            loadings = factor_loadings.iloc[i,:]
            loadings_list = []
            for cols in cols_list:
                loadings_list.append(list(loadings.loc[cols]))
            facts = np.array(loadings_list).transpose()
            facts = sm.add_constant(facts)
            rets = np.array(port_exc_ret_mon.iloc[i,:])
            coef = list(sm.OLS(rets,facts).fit().params)
            results.loc[date,:] = coef

        #calculate t test results
        means = results.mean()
        t_test_results = []
        for col in results.columns:
            test_result = ttest_ind(results.loc[:,col],[0 for i in results.loc[:,col]])[0]
            t_test_results.append(test_result)
        t_test_results.append('')
```

```
        #calculate mean factor loadings and perform the single regression approach to calculate R2 for Fama MacBeth te
        mean_factor_loadings = factor_loadings.mean()
        mean_factor_loadings_list = []
        for factor in factor_list:
            loading = list(mean_factor_loadings.loc[mean_factor_loadings.index.str.contains(factor)])
            mean_factor_loadings_list.append(loading)
        mean_factor_loadings = np.array(mean_factor_loadings_list).transpose()
        mean_factor_loadings = sm.add_constant(mean_factor_loadings)

        mean_returns = np.array(port_exc_ret_mon.mean())

        r2 = sm.OLS(mean_returns,mean_factor_loadings).fit().rsquared

        means.loc['Single Regression Approach R-squared'] = r2

        #create dataframe to hold final results of Fama MacBeth test
        results = pd.DataFrame(np.array([means,t_test_results]).transpose(),index = means.index,columns = ['coefficien
        results.index.name = model_name

        return results
```

Load and prepare data for testing

In [92]:

```
#load data
ports = pd.read_csv(r"your path to ff25_day.csv",index_col = 'Date')
factors = pd.read_csv(r"your path to mu_sigma.csv",index_col = 'Date')
ff_factors = pd.read_csv(r"your path to ff_factors.csv",index_col = 'Date')

#convert indices to datetime objects
ports.index = pd.to_datetime(ports.index)
factors.index = pd.to_datetime(factors.index)
ff_factors.index = pd.to_datetime(ff_factors.index)

#align dates of each dataframe
ports = ports.loc['1964-01-02':'2015-12-31',:]
factors = factors.loc['1964-01-02':,:]
ff_factors = ff_factors.loc['1964-01-02':'2015-12-31',:]

#add the Fama French Factors to one dataframe
factors.loc[:,['SMB','HML','MOM']] = ff_factors.loc[:,['SMB','HML','MOM']]
del ff_factors

#calculate excess returns
ports = ports.sub(factors.R_f,axis = 'rows')

#create column identifying the current year and month
ports.insert(0,'YearMonth',ports.index.strftime('%Y')+ports.index.strftime('%m'))
factors.insert(0,'YearMonth',factors.index.strftime('%Y')+factors.index.strftime('%m'))

#convert the yearmonth column to int rather than string
```

```
ports.YearMonth = ports.YearMonth.astype(int)
factors.YearMonth = factors.YearMonth.astype(int)

#calculate monthly portfolio returns.
monthports = (ports.iloc[:,1:]/100)+1
monthports.insert(0,'YearMonth',ports.YearMonth)
monthports = monthports.groupby('YearMonth').prod()
monthports = (monthports -1)*100

######## IMPORTANT########
#trims off all of the 1964 monthly returns to ensure that the Fama-MacBeth test is performed OUT OF SAMPLE
monthports = monthports.iloc[12:,:]

YearMonth = pd.Series(ports.YearMonth.unique(),index = ports.YearMonth.unique())

#create pandas series for mkt ret, mkt sigma, and factors. Convert indices of these series and portfolio return datafr
#to be the YearMonth list. Useful for indexing purposes while testing
mu = (factors.loc[:,"R_a.R_f"])
sigma = (factors.loc[:,"sigma_a"])
facs = factors.loc[:,['YearMonth','R_a.R_f','SMB','HML']]

mu.index = factors.YearMonth
sigma.index = factors.YearMonth
ports.set_index('YearMonth',inplace = True)
facs.set_index('YearMonth',inplace =True)
```

Create class instance

In [93]:
```
#Use test = Testing(False) if you do not want progress updates while the code is running
test = Testing()
```

Time series estimations of ZCAPM and linear factor models

In [94]:
```
#calculates time series factor loadings for each portfolio. See Testing class for information on the arguments of each
#method
zcapm_results = test.estZCAPM(ports,mu,sigma,.001,1000,1,12)
ff3_results = test.estLinearModel(ports,facs,12)
capm_results = test.estLinearModel(ports,facs.loc[:,['R_a.R_f']],12)

#adjust the zeta estimates for each portfolio to monthly estimates
zeta_cols = zcapm_results.columns[zcapm_results.columns.str.contains('zeta')]
zcapm_results.loc[:,zeta_cols] = zcapm_results.loc[:,zeta_cols]*21
```

```
Fitting linear model for BIG.HiBMBM
```

Out of sample cross-sectional Fama MacBeth test

In [95]:
```
#Runs the Fama-MacBeth Test for each portfolio
ZCAPM = test.FamaMacBeth(monthports,zcapm_results,['beta','zeta'],'ZCAPM')
```

```
FF3 = test.FamaMacBeth(monthports,ff3_results,['R_a.R_f','SMB','HML'],'Fama-French 3 Factor')
CAPM = test.FamaMacBeth(monthports,capm_results,['R_a.R_f',],'CAPM')
```

In [96]:

```
ZCAPM
```

Out[96]:

ZCAPM	coefficients	t-values
intercept	0.7593262880472027	3.0875681939793744
beta	-0.17833179614591696	-0.7265999715044811
zeta	0.4885823411522055	4.299951332644644
Single Regression Approach R-squared	0.9690609647879033	

In [102…

```
FF3
```

Out[102…

Fama-French 3 Factor	coefficients	t-values
intercept	0.8899926349195385	4.625249967909042
R_a.R_f	-0.3742676286214678	-1.7778283203036678
SMB	0.18226260438326772	1.3752332078248815
HML	0.3025136631944702	2.544038141241701
Single Regression Approach R-squared	0.6470600399973218	

In [103…

```
CAPM
```

Out[103…

CAPM	coefficients	t-values
intercept	0.9215175378932576	3.7385395290180075
R_a.R_f	-0.29867579468565364	-1.1877174962510977
Single Regression Approach R-squared	0.5198257918234421	

In [ ]: