# FINM 33210 Final Project

This project will guide you through a typical buy-side quant research project, in which you will use some of the statistical learning methods discussed earlier in the course to analyze a multi-factor model of the type discussed in module 4.

## Import Packages

In [1]:
```
import pandas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from statsmodels.formula.api import ols
from scipy.stats import gaussian_kde
import scipy
import scipy.sparse
import patsy
from statistics import median
import bz2
import math
```

## Load Data

In general, parsing text into numbers, dates, and other python data types can be done once, at the beginning of a research project, and the results can be saved to binary files. If the files are more than a few megabytes, use compression to avoid filling up your disk. In the following example, the model data has already been pre-processed and saved into pickle files, and the bz2 compression algorithm was used.

In [2]:
```
model_dir = '/Users/gordonritter/Dropbox/teaching/pickle_data/'

def sort_cols(test):
    return(test.reindex(sorted(test.columns), axis=1))

frames = {}
for year in [2004,2005,2006]:
    fil = model_dir + "pandas-frames." + str(year) + ".pickle.bz2"
    frames.update(pickle.load( bz2.open( fil, "rb" ) ))

for x in frames:
    frames[x] = sort_cols(frames[x])

covariance = {}
for year in [2003,2004,2005,2006]:
    fil = model_dir + "covariance." + str(year) + ".pickle.bz2"
    covariance.update(pickle.load( bz2.open(fil, "rb" ) ))

## the following dates array is used later for constructing time series plot

my_dates = sorted(list(map(lambda date: pd.to_datetime(date, format='%Y%m%d'
```

## Data Definitions

After the above loading operation is finished, "frames" will be a dictionary keyed by date. For example, the string "20040102" is one such key. Accessing the value at this key, with frames["20040102"], gives a data frame containing a daily cross section. Each row in the data frame corresponds to a particular stock, and there are many columns containing various attributes of the stock that have been collected. The meanings of the columns that we will use in this project are defined below; columns which may exist in the data, but not listed here, are not needed.

- ID: a unique identifier that can be used to link stocks across time
- DataDate: the date when the data would have been known, as of the close

## Factors

- 1DREVRSL: very short-term reversal, potential alpha factor but probably too fast-moving to be tradable
- STREVRSL: short-term reversal, potential alpha factor
- LTREVRSL: long-term reversal, potential risk factor
- BETA: risk factor computed from CAPM beta regression
- EARNQLTY: earnings quality, potential alpha factor
- EARNYILD: earnings yield (blend of forecasted earnings and historical earnings divided by market cap)
- GROWTH: mix of historical and forecasted earnings growth
- INDMOM: industry momentum (relative historical performance of the other stocks in the same industry)
- LEVERAGE: financial leverage of the company's balance sheet, usually a risk factor
- LIQUIDTY: factor with high loadings for very liquidly traded names; usually a risk factor
- MGMTQLTY: alpha factor which looks at quantitative measures of how well-run a company is by its management
- MOMENTUM: 12-month growth in stock price, usually a risk factor
- PROFIT: profitability, potential alpha factor
- PROSPECT: based on skewness of the return distribution, potential risk factor
- RESVOL: risk factor computed from residual volatility
- SEASON: seasonality-based alpha factor
- SENTMT: news sentiment alpha factor
- SIZE: risk factor based on log(market capitalization)
- VALUE: risk factor based on ratio of tangible book value to current price

## Non-factor Attributes

- Ret: asset's total return on the next day after the factor loadings are known, suitable as the Y vector in a regression analysis
- SpecRisk: specific risk is another name for predicted residual volatility. We called this the D matrix in our discussion of APT models.
- TotalRisk: predicted total vol, including factor and idiosyncratic contributions, annualized
- Yield: the dividend yield of the asset
- HistBeta: historically estimated CAPM beta coefficient
- PredBeta: model-predicted beta coefficient in the future
- IssuerMarketCap: aggregate market capitalization of the company (all share classes from the same issuer)
- BidAskSpread: bid-offer spread (average for the day)
- CompositeVolume: composite trading volume for the day

## Industry factors

'AERODEF', 'AIRLINES', 'ALUMSTEL', 'APPAREL', 'AUTO', 'BANKS','BEVTOB', 'BIOLIFE', 'BLDGPROD','CHEM', 'CNSTENG', 'CNSTMACH', 'CNSTMATL', 'COMMEQP', 'COMPELEC', 'COMSVCS', 'CONGLOM', 'CONTAINR', 'DISTRIB', 'DIVFIN', 'ELECEQP', 'ELECUTIL', 'FOODPROD', 'FOODRET', 'GASUTIL', 'HLTHEQP', 'HLTHSVCS', 'HOMEBLDG', 'HOUSEDUR','INDMACH', 'INSURNCE', 'INTERNET', 'LEISPROD', 'LEISSVCS', 'LIFEINS', 'MEDIA', 'MGDHLTH','MULTUTIL', 'OILGSCON', 'OILGSDRL', 'OILGSEQP', 'OILGSEXP', 'PAPER', 'PHARMA', 'PRECMTLS','PSNLPROD','REALEST', 'RESTAUR', 'ROADRAIL','SEMICOND', 'SEMIEQP','SOFTWARE', 'SPLTYRET', 'SPTYCHEM', 'SPTYSTOR', 'TELECOM', 'TRADECO', 'TRANSPRT', 'WIRELESS'

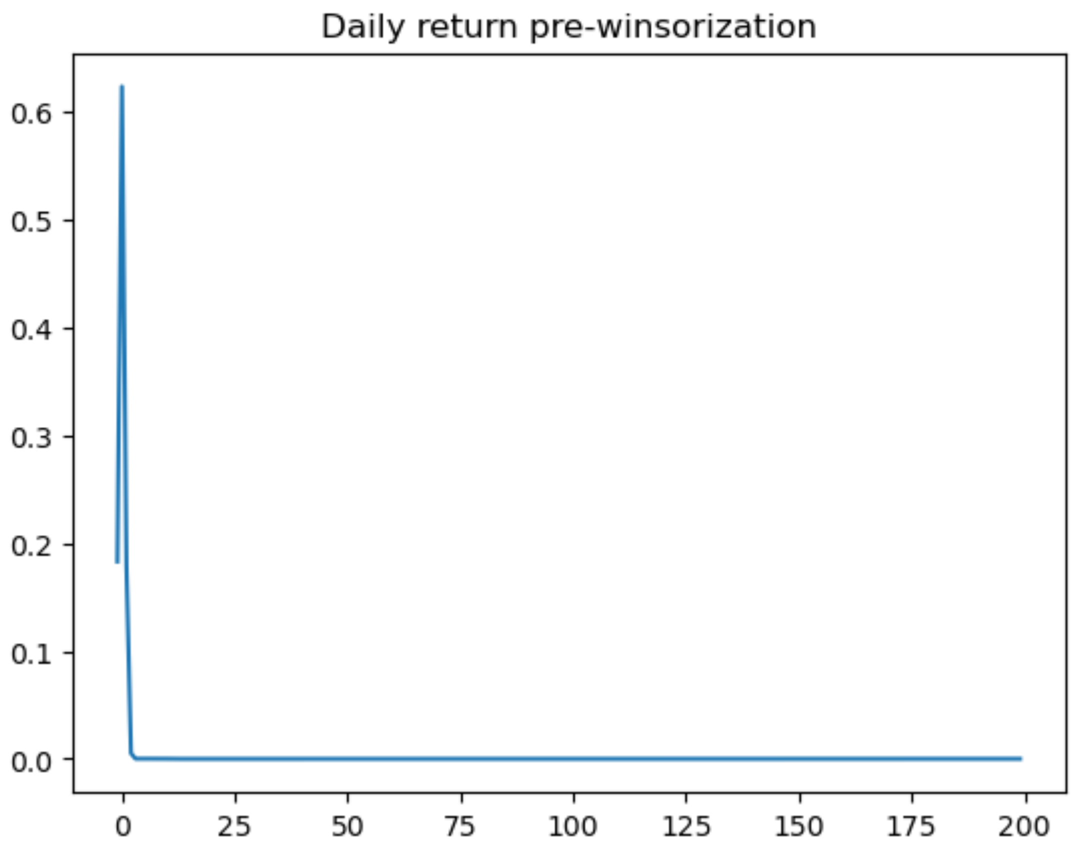
## Data Cleaning and Winsorization

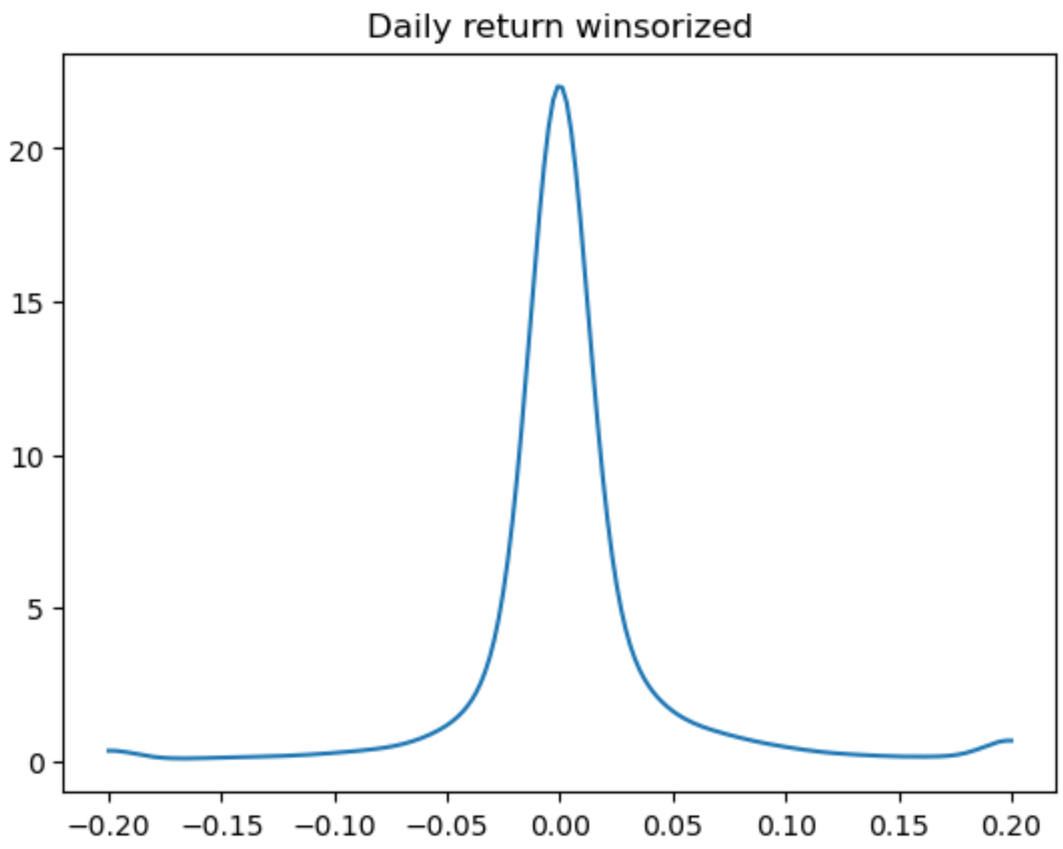
The distribution of many statistics can be heavily influenced by outliers. A simple approach to robustifying parameter estimation procedures is to set all outliers to a specified percentile of the data; for example, a 90% winsorization would see all data below the 5th percentile set to the 5th percentile, and data above the 95th percentile set to the 95th percentile. Winsorized estimators are usually more robust to outliers than their more standard forms.

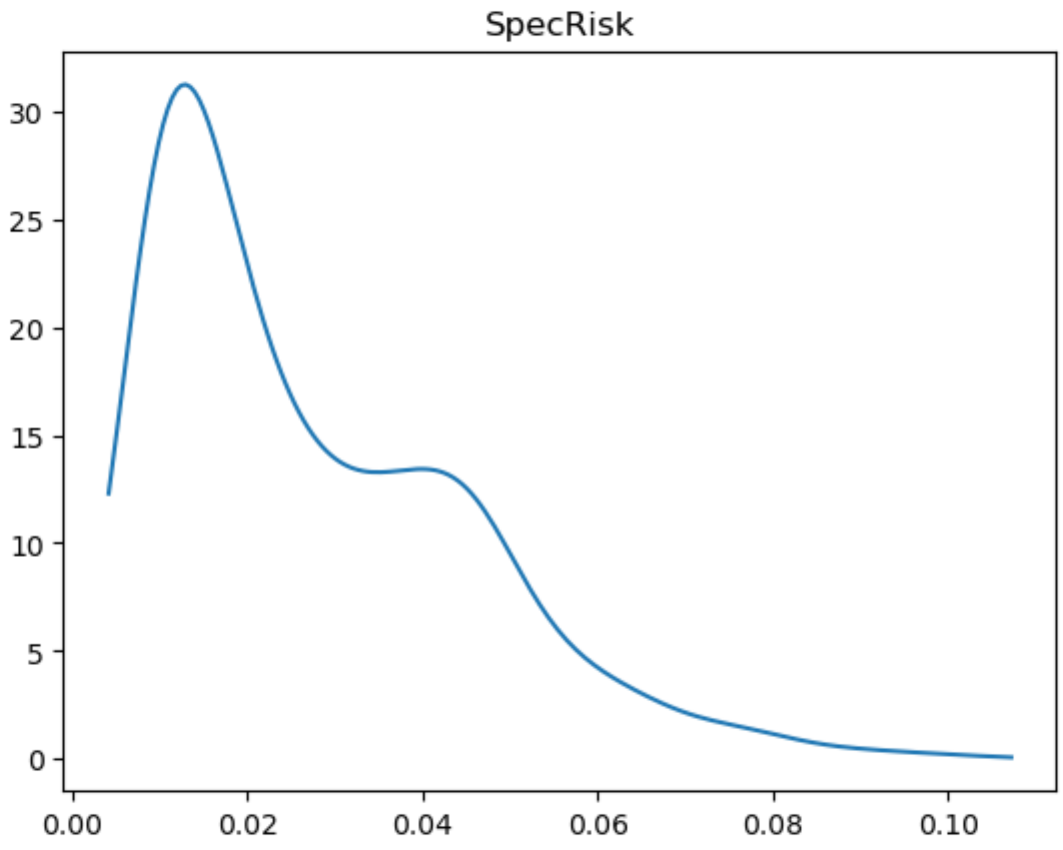
In [3]:
```python
def wins(x,a,b):
    return(np.where(x <= a,a, np.where(x >= b, b, x)))

def clean_nas(df):
    numeric_columns = df.select_dtypes(include=[np.number]).columns.tolist()

    for numeric_column in numeric_columns:
        df[numeric_column] = np.nan_to_num(df[numeric_column])

    return df
```

We can check the distribution of returns with a density plot, both before and after winsorization to observe the effect of trimming the tails.

In [4]:
```python
def density_plot(data, title):
    density = gaussian_kde(data)
    xs = np.linspace(np.min(data),np.max(data),200)
    density.covariance_factor = lambda : .25
    density._compute_covariance()
    plt.plot(xs,density(xs))
    plt.title(title)
    plt.show()

test = frames['20040102']
density_plot(test['Ret'], 'Daily return pre-winsorization')
density_plot(wins(test['Ret'],-0.2,0.2), 'Daily return winsorized')

D = (test['SpecRisk'] / (100 * math.sqrt(252))) ** 2
density_plot(np.sqrt(D), 'SpecRisk')
```




Daily return winsorized
20
15
10
5
0
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20


SpecRisk
30
25
20
15
10
5
0
0.00
0.02
0.04
0.06
0.08
0.10

## Factors

## Factor Exposures and Factor Returns

Arbitrage pricing theory relaxes several of the assumptions made in the course of deriving the CAPM. In particular, we relax the assumption that all investors do the same optimization and hence that there is a single efficient fund. This allows the possibility that a CAPM-like relation may hold, but with multiple underlying sources of risk.

Specifically, let $r_i, i = 1, \ldots, n$ denote the cross-section of asset returns over a given time period $[t, t+1]$. In a fully-general model, the multivariate distribution $p(\mathbf{r})$ could have arbitrary covariance and higher-moment structures, but remember that for $n$ large there is typically never enough data to estimate such over-parameterized models.

Instead, we assume a structural model which is the most direct generalization of the CAPM:

$$r_i = \beta_{i,1} f_1 + \beta_{i,2} f_2 + \cdots + \beta_{i,p} f_p + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma_i^2)$$

If $p = 1$, this reduces to the Capital Asset Pricing Model (CAPM) in a rather direct way.

With $p > 1$, the model starts to differ from the CAPM in several very important aspects. In the CAPM, we were able to identify the single efficient fund by arguing that its weights must equal the market-capitalization weights. Hence we were given for free a very nice proxy for the single efficient fund: a capitalization-weighted basket such as the Russell 3000. Hence in the $p = 1$ case we had a convenient proxy which could be used to impute the return $f_1$, which we called $r_M$. Also $\beta_{i,1}$ could be estimated, with no more than the usual statistical estimation error, by time-series regression.

If $p > 1$ then the underlying assumptions of that argument break down: there is no longer any simple way to identify $f_j$ nor $\beta_{i,j}$ ($j = 1, \ldots, p$). We shall return to the estimation problem in due course.

To avoid confusion with the CAPM, and its simplistic $\beta$ coefficient (which is still sometimes used in larger multi-factor models), it is conventional to make the following notation change: $\beta_{i,j}$ becomes $X_{i,j}$ and so the model equation becomes

$$r_i = X_{i,1} f_1 + X_{i,2} f_2 + \cdots + X_{i,p} f_p + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma_i^2)$$

It's difficult to simultaneously estimate both all components $X_{i,j}$ and all risk-source returns $f_j$, so one usually assumes one is known and calculates the other via regression. In what follows, we focus on the approach where $X$ is known, and the $f_j$ are assumed to be hidden (aka latent) variables.

The structural equation is more conveniently expressed in matrix form:

$$R_{t+1} = X_t f_{t+1} + \epsilon_{t+1}, \quad E[\epsilon] = 0, \;\; V[\epsilon] = D$$

where $R_{t+1}$ is an $n$-dimensional random vector containing the cross-section of returns in excess of the risk-free rate over some time interval $[t, t+1]$, and $X_t$ is a (non-random) $n \times p$ matrix that can be calculated entirely from data known before time $t$. The variable $f$ in denotes a $p$-dimensional random vector process which cannot be observed directly.

Since the variable $f$ denotes a $p$-dimensional random vector process which cannot be observed directly, information about the $f$-process must be obtained via statistical inference. We assume that the $f$-process has finite first and second moments given by

$$E[f] = \mu_f, \;\; \text{and} \;\; V[f] = F.$$

The primary outputs of a statistical inference process are the parameters $\mu_f$ and $F$, and other outputs one might be interested in include estimates of the daily realizations $\hat{f}_{t+1}$.

The simplest way of estimating historical daily realizations of $\hat{f}_{t+1}$ is by least-squares (ordinary or weighted, as appropriate), viewing the defining model equation as a regression problem.

In [5]:
```
industry_factors = ['AERODEF', 'AIRLINES', 'ALUMSTEL', 'APPAREL', 'AUTO',
       'BANKS','BEVTOB', 'BIOLIFE', 'BLDGPROD','CHEM', 'CNSTENG', 'CNSTMACH'
       'COMSVCS', 'CONGLOM', 'CONTAINR', 'DISTRIB',
       'DIVFIN', 'ELECEQP', 'ELECUTIL', 'FOODPROD', 'FOODRET', 'GASUTIL',
       'HLTHEQP', 'HLTHSVCS', 'HOMEBLDG', 'HOUSEDUR','INDMACH', 'INSURNCE',
        'LEISPROD', 'LEISSVCS', 'LIFEINS', 'MEDIA', 'MGDHLTH','MULTUTIL',
       'OILGSCON', 'OILGSDRL', 'OILGSEQP', 'OILGSEXP', 'PAPER', 'PHARMA',
       'PRECMTLS','PSNLPROD','REALEST',
       'RESTAUR', 'ROADRAIL','SEMICOND', 'SEMIEQP','SOFTWARE', 'SPLTYRET', '
       'TELECOM', 'TRADECO', 'TRANSPRT', 'WIRELESS']

style_factors = ['BETA', 'SIZE', 'MOMENTUM', 'VALUE', 'GROWTH', 'LEVERAGE',
                 'DIVYILD', 'LTREVRSL', 'EARNQLTY']

## an R-style formula which can be used to construct a cross sectional regre
def get_formula(alphas, Y):
    L = ["0"]
    L.extend(alphas)
    L.extend(style_factors)
    L.extend(industry_factors)
    return Y + " ~ " + " + ".join(L)

## The term 'estu' is short for estimation universe
def get_estu(df):
    estu = df.loc[df.IssuerMarketCap > 1e9].copy(deep=True)
    return estu

def estimate_factor_returns(df, alphas):
    ## build universe based on filters
    estu = get_estu(df)
    ## winsorize returns for fitting
    estu['Ret'] = wins(estu['Ret'], -0.25, 0.25)

    model = ols(get_formula(alphas, "Ret"), data=estu)
    return(model.fit())
```

In a real trading scenario, alpha factor construction would be the culmination of a very long research process, usually undertaken by experts in financial markets and requiring time and ingenuity. For this exercise, we consider several well-known alpha factors from the list above.

Running one OLS per day over several years, where each OLS involves several thousand observations and about 50-100 independent variables, takes a few minutes.

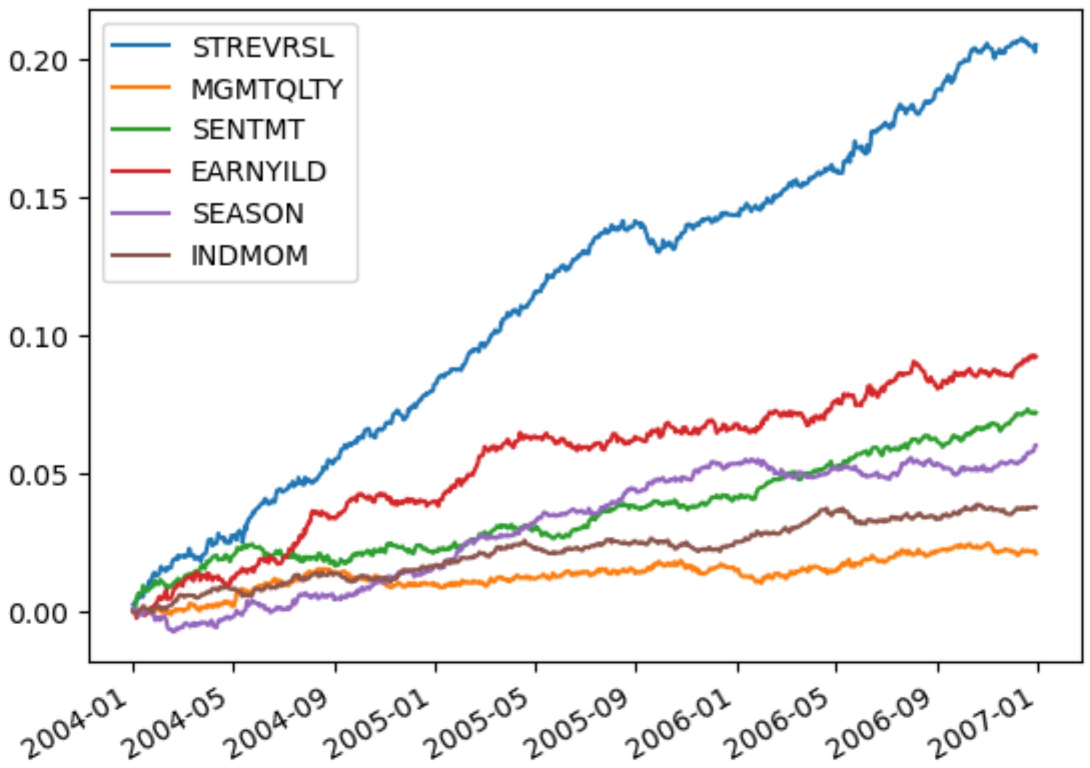
In [6]:
```
alpha_factors = ['STREVRSL', 'MGMTQLTY', 'SENTMT', 'EARNYILD', 'SEASON', 'IN
```

In [7]:
```
facret = {}

for date in frames:
    facret[date] = estimate_factor_returns(frames[date], list(alpha_factors)
```

We now have a multivariate time series of factor returns stored in the variable facret. We can plot the cumulative sum of the factor returns.

In [8]:
```
facret_df = pd.DataFrame(index = my_dates)

for dt in my_dates:
    for alp in alpha_factors:
        facret_df.at[dt, alp] = facret[dt.strftime('%Y%m%d')][alp]

facret_df.cumsum().plot()
```

Out[8]:
```
<AxesSubplot: >
```



The following table gives the vector called $\mu_f$ in lecture.

In [9]:
```
print(facret_df.mean())
```

```
STREVRSL    0.000271
MGMTQLTY    0.000028
SENTMT      0.000095
EARNYILD    0.000122
SEASON      0.000080
INDMOM      0.000050
dtype: float64
```

## Helpful code to show how to get X, F, D matrices for a particular date

In [10]:
```
def colnames(X):
    if(type(X) == patsy.design_info.DesignMatrix):
        return(X.design_info.column_names)
    if(type(X) == pandas.core.frame.DataFrame):
        return(X.columns.tolist())
    return(None)

def diagonal_factor_cov(date, X):
    cv = covariance[date]
    k = np.shape(X)[1]
    Fm = np.zeros([k,k])
    for j in range(0,k):
        fac = colnames(X)[j]
        Fm[j,j] = (0.01**2) * cv.loc[(cv.Factor1==fac) & (cv.Factor2==fac),'
    return(Fm)

def risk_exposures(estu):
    L = ["0"]
    L.extend(style_factors)
    L.extend(industry_factors)
    my_formula = " + ".join(L)
    return patsy.dmatrix(my_formula, data = estu)
```

In [13]:
```
my_date = '20040102'

# estu = estimation universe
estu = get_estu(frames[my_date])
n = estu.shape[0]

estu['Ret'] = wins(estu['Ret'], -0.25, 0.25)

rske = risk_exposures(estu)
F = diagonal_factor_cov(my_date, rske)

X = np.asarray(rske)
D = np.asarray( (estu['SpecRisk'] / (100 * math.sqrt(252))) ** 2 )

kappa = 1e-5

print(F)
```

```
[[0.00348478 0.         0.         ... 0.         0.         0.        ]
 [0.         0.00061952 0.         ... 0.         0.         0.        ]
 [0.         0.         0.00042807 ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.00521378 0.         0.        ]
 [0.         0.         0.         ... 0.         0.00726435 0.        ]
 [0.         0.         0.         ... 0.         0.         0.02185539]]
```

## Problem 1.

Consider the list of potential alpha factors whose factor returns were plotted above. Consider ways of combining all of the alpha factors into a single, composite alpha factor. The goal is to produce a composite alpha factor which does the best job of predicting the dependent variable "Ret" on out-of-sample data. Restrict your training set on each day to the estimation universe that was used above in get_estu. Take the period 2008-2010 as the ultimate test set which you will hold in a "vault" until you are ready to do a final evaluation of your composite alpha factor. Use the period 2003-2007 for training/validation. Use cross-validation to select a model from the full family of models you are considering. It could be as simple as lasso, but we encourage you to be creative and try non-linear combinations of factors as well. Use methods from the course.

## Problem 2.

Referring to formula (4.3) for the Markowitz portfolio, and the covariance matrix in (4.29), code up a function to compute the Markowitz portfolio for each date in our sample. Refer to the helpful code above to get the different pieces in (4.29). For the risk-aversion constant, use $\kappa$ = 1e-5. Restrict yourself to the estimation universe that was used above in get_estu. Use your composite alpha factor from Problem 1 as the substitute for $\mathbb{E}[\mathbf{r}]$. It is recommended to use the fast inversion formula (4.32) to speed things up. Compute the dot product of your portfolio with the return, that is compute $\mathbf{h} \cdot \mathbf{r}$ for each date in the sample, and plot the cumulative sum of the results. For $\mathbf{r}$, use the column called "Ret" in the same data frame that was used to compute the portfolio itself.

## Extra credit

Plot other interesting metrics which help understand the portfolios in problem 2. For example, plot their long/short/net in dollars, number of holdings, factor model's predicted volatility of the portfolio, percent of variance from idio, style, industry.

## Teamwork

Note that problem 2 only needs the output from Problem 1, but it does not depend on the details of how Problem 1 is being solved. So a team could split up along those lines. Part of the team can start on Problem 2, just assuming the output from problem 1 is, say, a single one of the alpha factors, until the part of the team doing problem 1 has something more interesting.

```
In [ ]:
```