demo

August 20, 2019

1 Portfolio Optimization

In the following I will investigate several portfolio optimization methods by means of Monte Carlo simulation. In particular, I will compare Markowitz mean-variance optimization based on sample moments with industry-neutral equal weighting as a baseline. Further, I will explore a hierarchical method and principal component analysis as a means to reduce estimation error of the covariance matrix. These methods can improve upon Markowitz optimization and equal weighting both under Gaussian and fat-tailed noise distributions.

Let's simulate a bunch of stocks belonging to different industries. Each stock is composed of a deterministic trend μ, a loading on the market related stochastic trend β_{market} and a loading on the industry specific stochastic trend $\beta_{industry}$. For each stock, μ, β_{market}, and $\beta_{industry}$ are sampled from a uniform distribution. In the first simulation, I compare a mean-variance optimization with a naive diversification approach where each asset is equally weighted with the sign of the expected value. At this point I should emphasize that I assume perfect knowledge of the expected value. Traditionally, the expected value is estimated by the in-sample first moment of the asset. In my example, this would actually make sense since the mean is a consistent estimator of the deterministic trend. In practice, however, there is probably not a constant deterministic trend. This is where the alpha model steps in, which is not the topic of this study and thus assumed given.

The return of each stock is thus given by

$$\begin{aligned} R_{it} &= \mu_i + \beta_{i1} f_{1t} + \beta_{i2} f_{2t} + \cdots + \beta_{ik} f_{kt} + \epsilon_{it} \\ &= \mu_i + \sum_{\ell=1}^{k} \beta_{i\ell} f_{\ell t} + \epsilon_{it}, \quad i = 1, \ldots, N, \quad t = 1, \ldots, T \end{aligned}$$

R_{it} is the return of asset i at time $t, i = 1, \ldots, N$
$f_{\ell t}$ is the ℓ th common factor at time $t, \ell = 1, \ldots, k$
$\beta_{i\ell}$ is the factor loading or factor beta of asset i with respect to factor $\ell, i = 1, \ldots, N, \ell = 1, \ldots, k$
ϵ_{it} is the asset-specific factor or asset-specific risk.

The $k \times k$ covariance matrix of the factors is

$\operatorname{Cov}(\boldsymbol{f}_t) = \boldsymbol{\Sigma}_f$

where

$\boldsymbol{f}_t = [f_{1t}, f_{2t}, \ldots, f_{kt}]'$

Asset-specific noise is uncorrelated with the factors $\operatorname{Cov}(f_{\ell t}, \epsilon_{it}) = 0, \quad \ell = 1, \ldots, k, \quad i = 1, \ldots, N, \quad t = 1, \ldots, T,$

$$\boldsymbol{\Sigma}_\epsilon = \text{Cov}\left(\boldsymbol{\epsilon}_t\right) = \begin{bmatrix} \sigma^2_{\epsilon_1} & 0 & \cdots & 0 \\ 0 & \sigma^2_{\epsilon_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2_{\epsilon_N} \end{bmatrix} = \text{diag}\left(\sigma^2_{\epsilon_1}, \sigma^2_{\epsilon_2}, \ldots, \sigma^2_{\epsilon_N}\right),$$

The asset-specific risks are likewise uncorrelated across assets, and for each asset they are serially uncorrelated,

$$\text{Cov}\left(\epsilon_{it}, \epsilon_{jt'}\right) = \begin{cases} \sigma_i^2 & \text{if } i = j \text{ and } t = t' \\ 0 & \text{otherwise} \end{cases}$$

Thus a diagonal covariance structure reflects the assumption that all correlation between assets is due to the factors.

If we write the factor model as

$$\boldsymbol{R}_t = \boldsymbol{\alpha} + \boldsymbol{B}\boldsymbol{f}_t + \boldsymbol{\epsilon}_t, \quad t = 1, \ldots, T$$

where in the $N \times k$ matrix

$$B = \begin{bmatrix} \boldsymbol{\beta}_1' \\ \boldsymbol{\beta}_2' \\ \vdots \\ \boldsymbol{\beta}_N' \end{bmatrix} = \begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1k} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{N1} & \beta_{N2} & \cdots & \beta_{Nk} \end{bmatrix}$$

the ℓth column contains the beta coefficients associated with factor ℓ

The covariance matrix of the returns $R_t = \left[R_{1t}, R_{2t}, \ldots, R_{Nt}\right]$ implied by the factor model is

$$\Sigma = \text{Cov}\left(R_t\right) = B\Sigma_f B' + \Sigma_\epsilon$$

That is,

$$\text{Var}\left(R_i\right) = \beta_i'\Sigma_f\beta_i + \sigma^2_{\epsilon_i}$$

$$\text{Cov}\left(R_i, R_j\right) = \beta_i'\Sigma_f\beta_j$$

Since the factors are uncorrelated in this simulation, the covariance matrix simplifies to

$$\Sigma = B\Sigma_f B' + \Sigma_\epsilon = \sum_{\ell=1}^k \beta_\ell\beta_\ell' e^2_{f_\ell} + \Sigma_\epsilon$$

where

β_ℓ is the vector of loadings with respect to factor ℓ, i.e. the ℓ th column of matrix $\boldsymbol{B}$,

$\sigma^2_{f_\ell}$ is the variance of factor ℓ

Thus the variance of asset i is

$$\sigma_i^2 = \sum_{\ell=1}^k \beta_{i\ell}^2\sigma^2_{f_\ell} + \sigma^2_{\epsilon_i}$$

and the covariance between the returns of assets i and j is

$$\sigma_{ij} = \sum_{\ell=1}^k \beta_{i\ell}\beta_{j\ell}\sigma^2_{f_\ell}$$

In [1]:
```
import numpy as np
import pandas as pd
from scipy.linalg import eigh, cholesky
from scipy.stats import norm
from matplotlib import pyplot as plt
plt.style.use('seaborn')
# %matplotlib notebook
from src import optimize
from src.portfolio import Portfolio
```

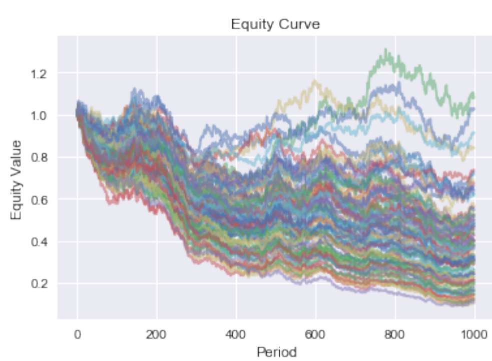
```python
def gen_industry_stocks(n_stocks, market, industry, industry_name, sigma_noise,
                        p_year=250):
    mu = pd.Series(np.random.uniform(low=-0.2, high=0.2, size=n_stocks))/p_year
    beta_market = pd.Series(np.random.uniform(low=0.5, high=2.0, size=n_stocks))
    beta_industry = pd.Series(np.random.uniform(low=0.5, high=2.0, size=n_stocks))
    stocks = pd.DataFrame()

    for i in range(n_stocks):
        normal_noise = norm.rvs(size=(1, market.shape[0]))[0]*sigma_noise
        stocks[f'{industry_name}_stock_{i}'] = (mu[i] +
                                                beta_market[i]*market +
                                                beta_industry[i]*industry +
                                                normal_noise)
    mu.index = stocks.columns
    return stocks, mu, beta_market, beta_industry


def generate_garch_11_ts(n, sigma_sq_0, mu, alpha, beta, omega):
    """ generate GARCH log returns """
    nu = np.random.normal(0, 1, n)
    r = np.zeros(n)
    epsilon = np.zeros(n)
    sigma_sq = np.zeros(n)
    sigma_sq[0] = sigma_sq_0

    if min(alpha, beta) < 0:
        raise ValueError('alpha, beta need to be non-negative')
    if omega <= 0:
        raise ValueError('omega needs to be positive')

    if alpha+beta >= 1:
        print('alpha+beta>=1, variance not defined -->\
              time series will not be weakly stationary')

    for i in range(n):

        if i > 0:
            sigma_sq[i] = (omega +
                           alpha * epsilon[i-1]**2 +
                           beta * sigma_sq[i-1])

        epsilon[i] = (sigma_sq[i]**0.5) * nu[i]

        r[i] = mu + epsilon[i]
    return r


def gen_industry_stocks_garch(n_stocks, market, industry, industry_name,
                              sigma_noise, p_year=250, garch_mu=0,
                              garch_alpha=0.4, garch_beta=0.4):
    """ Generate stocks of given industry with fat-tailed noise. """
    mu = pd.Series(np.random.uniform(low=-0.2, high=0.2, size=n_stocks))/p_year
    beta_market = pd.Series(np.random.uniform(low=0.5, high=2.0, size=n_stocks))
    beta_industry = pd.Series(np.random.uniform(low=0.5, high=2.0, size=n_stocks))
    stocks = pd.DataFrame()
    garch_noise = generate_garch_11_ts(market.shape[0],
                                       sigma_noise**2,
                                       garch_mu, garch_alpha, garch_beta,
                                       sigma_noise**2)
    for i in range(n_stocks):
        garch_noise = generate_garch_11_ts(market.shape[0],
                                           sigma_noise**2,
                                           garch_mu, garch_alpha, garch_beta,
                                           sigma_noise**2)

        stocks[f'{industry_name}_stock_{i}'] = (mu[i] +
                                                beta_market[i]*market +
                                                beta_industry[i]*industry +
```

```
                                                   garch_noise)
    mu.index = stocks.columns
    return stocks, mu, beta_market, beta_industry


def information_ratio(equity_curve, p_year=250):
    return equity_curve.mean()/equity_curve.std()*p_year**0.5


def plot_equity_curve(log_returns):
    plt.plot(np.exp(log_returns.cumsum()), alpha=0.5)
    plt.title('Equity Curve')
    plt.xlabel('Period')
    plt.ylabel('Equity Value')
    plt.show()


def show_standard_optimized_equity(n_stocks, industry, long_only=False):
    trainig_pct = 0.5
    n_train = int(trainig_pct*num_samples)
    market = norm.rvs(size=(1, num_samples))[0]*sigma['market']

    stocks, mu, beta_market, beta_industry = gen_industry_stocks(
        n_stocks,
        market,
        list(industry.values())[0],
        list(industry.keys())[0],
        sigma_noise,
        p_year)

    plot_equity_curve(stocks)

    if long_only:
        lower_bound = 0
        equal_weights = mu.apply(lambda x: 0 if x <= 0 else 1)/mu.shape[0]
        market_neutral = False
    else:
        lower_bound = -1
        equal_weights = mu.apply(np.sign)/mu.shape[0]
        market_neutral = True

    weights = optimize.minimize_objective(mu.index,
                                          optimize.negative_sharpe,
                                          market_neutral,
                                          (lower_bound, 1),
                                          mu, stocks[:n_train].cov(),
                                          0.0, 0.0)
    # Make same gross leverage
    equal_weights = equal_weights*np.abs(np.array(list(weights.values()))).sum()

    portfolio_sm_is = np.dot(stocks.values,
                             np.array(list(weights.values())))[:n_train]
    IR_ann = information_ratio(portfolio_sm_is)
    print('IR sm_is: ', IR_ann)
    plot_equity_curve(portfolio_sm_is)

    portfolio_equal_is = np.dot(stocks.values, equal_weights)[:n_train]
    IR_ann = information_ratio(portfolio_equal_is)
    print('IR equal_is: ', IR_ann)
    plot_equity_curve(portfolio_equal_is)

    portfolio_sm_oos = np.dot(stocks.values,
                              np.array(list(weights.values())))[n_train:]
    IR_ann = information_ratio(portfolio_sm_oos)
    print('IR sm_oos: ', IR_ann)
    plot_equity_curve(portfolio_sm_oos)

    portfolio_equal_oos = np.dot(stocks.values, equal_weights)[:n_train]
```

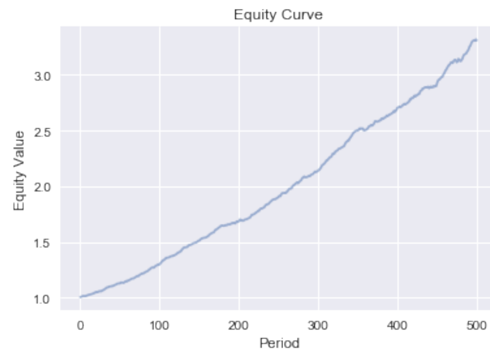
```
    IR_ann = information_ratio(portfolio_equal_oos)
    print('IR equal_oos: ', IR_ann)
    plot_equity_curve(portfolio_equal_oos)

    return weights


p_year = 250
num_samples = p_year*4

sigma = {}
sigma['market'] = 0.1/p_year**0.5
sigma['banks'] = 0.1/p_year**0.5
sigma['oil'] = 0.14/p_year**0.5
sigma['insurance'] = 0.07/p_year**0.5
sigma['tech'] = 0.2/p_year**0.5
sigma['bio'] = 0.12/p_year**0.5
sigma['pharma'] = 0.22/p_year**0.5
sigma['auto'] = 0.05/p_year**0.5
sigma['retail'] = 0.125/p_year**0.5
sigma['manufacturing'] = 0.1/p_year**0.5

sigma_noise = 0.1/p_year**0.5

MC_RUNS = 10
PLOT_STOCKS = False
```

The results below show that out-of-sample (oos) performance of the optimized portfolio using sample moments (sm) is decent if the number of assets is relatively small. It definitely beats equal weighting (equal).

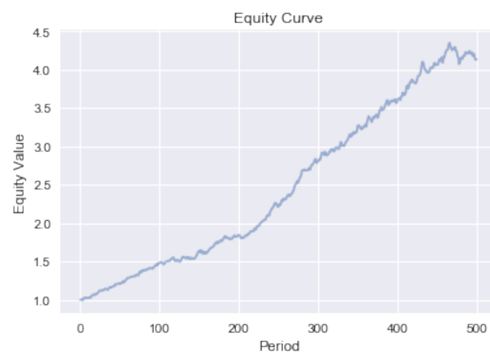
```
In [2]: industry = {}
        industry['manufacturing'] = norm.rvs(size=(1, num_samples))[0]*sigma['manufacturing']
        _ = show_standard_optimized_equity(n_stocks=100, industry=industry)
```



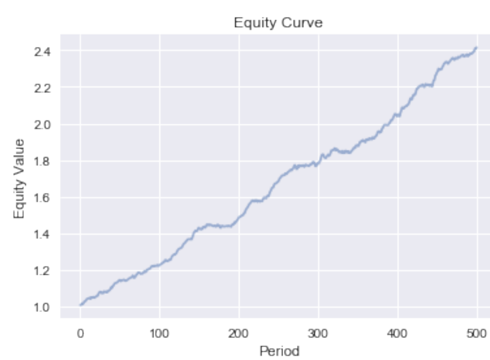

```
IR sm_is:  14.274032129538938
```



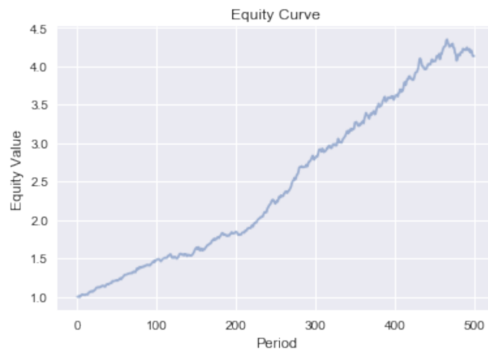

```
IR equal_is:  5.721563189877289
```




```
IR sm_oos:  7.637887456929092
```

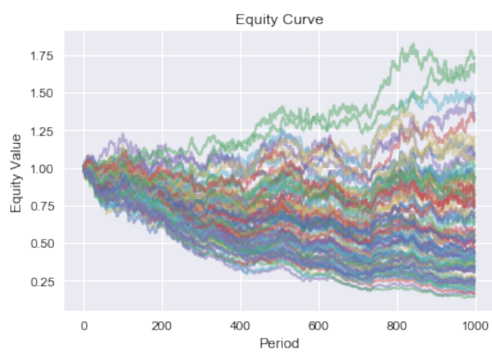



```
IR equal_oos:  5.721563189877289
```

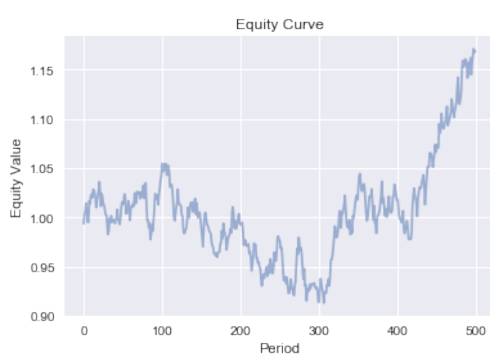


As shown below, long-only performance, of course, is horrendous. This is due to the fact that we cannot diversify the systematic risk. Since we cannot short, hedging out that risk is also not possible. Hence, our bets are not independent and the Law of Large Numbers does not apply.

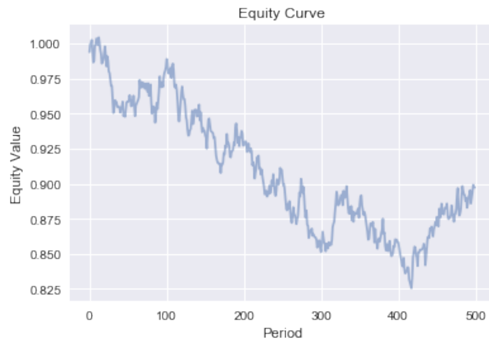
```
In [3]: _ = show_standard_optimized_equity(n_stocks=100, industry=industry, long_only=True)
```



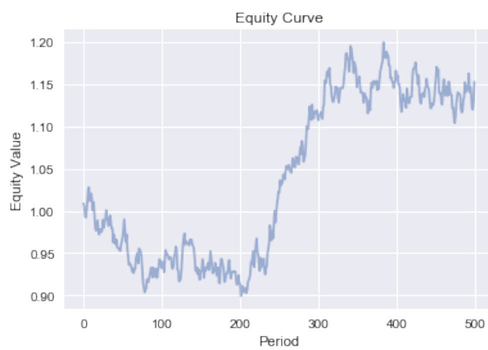

```
IR sm_is:  0.5853880251555333
```



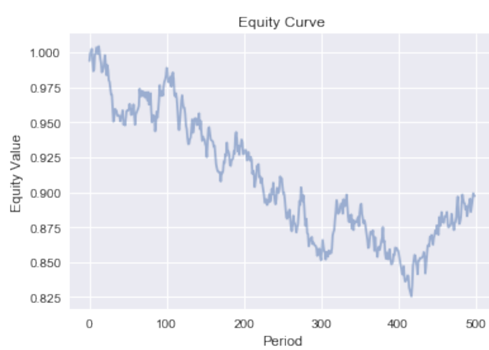
```
IR equal_is:  -0.6123757070640578
```




```
IR sm_oos:  0.5164220402938328
```

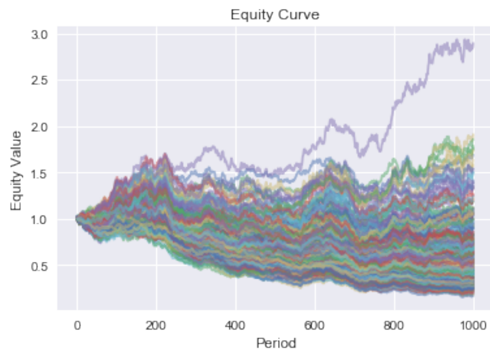



```
IR equal_oos:  -0.6123757070640578
```

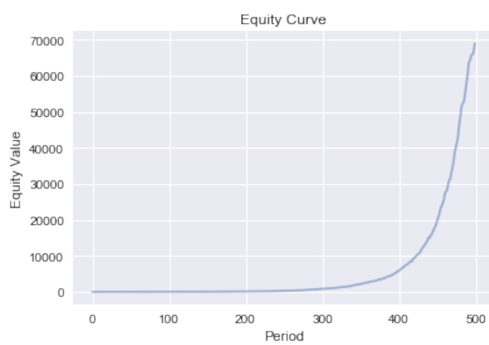


Standard mean-variance optimization works well if the number of assets is relatively small. When the number of assets is relatively large, however, the out-of-sample performance is significantly worse than the in-sample performance as demonstrated by the information ratio (IR). Even the very naive approach of equal weighting beats mean-variance optimization out-of-sample embarrassingly often.

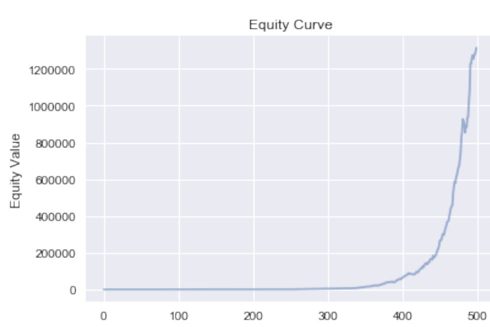
```
In [4]: weights = show_standard_optimized_equity(n_stocks=300, industry=industry)
```



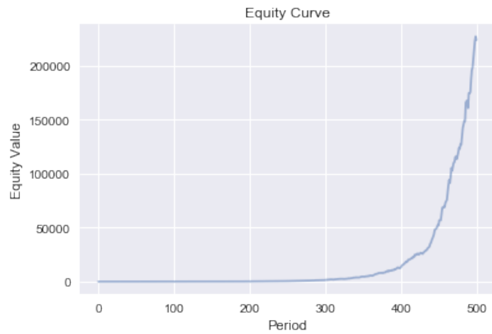

```
IR sm_is:  26.949248378797808
```



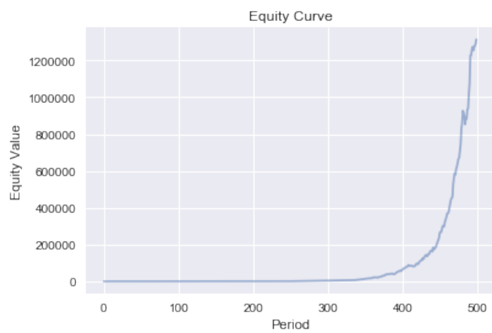

```
IR equal_is:  10.425697654396167
```



IR sm_oos: 11.796260598862835



IR equal_oos: 10.425697654396167



In [5]:
```
# creating portfolio object
pf = Portfolio(assets=weights.keys(),
               position=pd.Series(weights),
               price=[1]*len(weights.keys()))
print(pf)
print('largest long:', pf[0], pf.position(pf[0]))
print('largest short:', pf[-1], pf.position(pf[-1]))
```

```
Portfolio: 300 Assets, $35.69 Long, $35.69 Short
largest long: manufacturing_stock_35 0.7470542974614564
largest short: manufacturing_stock_171 -0.686910165001864
```

In [6]:
```
# Showing the first 5 positions in portfolio
pf.portfolio_df.head()
```

```
Out[6]:                           position  price  factor_value  sector_id  \
        manufacturing_stock_35    0.747054      1           1.0        0.0
        manufacturing_stock_227   0.618434      1           1.0        0.0
        manufacturing_stock_12    0.587384      1           1.0        0.0
        manufacturing_stock_256   0.579406      1           1.0        0.0
        manufacturing_stock_125   0.555688      1           1.0        0.0

                                  position_value
        manufacturing_stock_35          0.747054
        manufacturing_stock_227         0.618434
        manufacturing_stock_12          0.587384
        manufacturing_stock_256         0.579406
        manufacturing_stock_125         0.555688
```

```
In [7]: # Plotting the weight distribution
        pf.portfolio_df.position_value.plot(style='.')
        plt.title('Position Allocation')
        plt.xlabel('Ordered Positions')
        plt.ylabel('Weight')
        plt.show()
```

Here you can see a typical result of mean-variance optimization. There are some very large positions. These are most likely due to estimation errors of the covariance matrix and thus undesirable. This problem is even greater when μ, too, is estimated with error. As the number of assets grows, there is a higher likelihood of observing an extreme covariance value by chance and thus this problem actually grows with the number of assets. This is so common that it has a name. In the literature it goes by 'Error Maximization'.

1.1 Imposing Structure

To reduce Error Maximization we need to impose structure. One way to do this is by using our knowledge that stocks usually cluster (e.g., into certain industries). It seems like a good idea to encode this prior knowledge by mean-variance optimize stocks within clusters and then optimize allocation to these clusters. This is pursued next and compared to the standard optimization.

```
In [8]: trainig_pct = 0.5
        n_train = int(trainig_pct*num_samples)
        ir_sm_is = []
        ir_sm_oos = []
```

```
ir_hier_is = []
ir_hier_oos = []
ir_gt_is = []
ir_gt_oos = []
ir_equal_is = []
ir_equal_oos = []
n_stocks = 50

for j in range(MC_RUNS):
    print('MC run: ', j)

    market = norm.rvs(size=(1, num_samples))[0]*sigma['market']
    industries = {}
    industries['banks'] =  norm.rvs(size=(1, num_samples))[0]*sigma['banks']
    industries['oil'] = norm.rvs(size=(1, num_samples))[0]*sigma['oil']
    industries['insurance'] = norm.rvs(size=(1, num_samples))[0]*sigma['insurance']
    industries['tech'] = norm.rvs(size=(1, num_samples))[0]*sigma['tech']
    industries['bio'] = norm.rvs(size=(1, num_samples))[0]*sigma['bio']
    industries['pharma'] = norm.rvs(size=(1, num_samples))[0]*sigma['pharma']
    industries['auto'] = norm.rvs(size=(1, num_samples))[0]*sigma['auto']
    industries['retail'] = norm.rvs(size=(1, num_samples))[0]*sigma['retail']
    industries['manufacturing'] = norm.rvs(size=(1,
num_samples))[0]*sigma['manufacturing']

    industries_stocks = {}
    industries_mu = {}
    industries_beta_market = {}
    industries_beta_industries = {}
    industries_weights = {}
    industries_portfolio = {}

    stocks_all = pd.DataFrame()
    expected_returns_all = pd.Series()

    B = np.zeros((n_stocks*len(industries.keys()), len(sigma)))
    for n, i in enumerate(industries.keys()):
        industries_stocks[i],\
        industries_mu[i],\
        industries_beta_market[i],\
        industries_beta_industries[i] = gen_industry_stocks(n_stocks,
                                                             market,
                                                             industries[i],
                                                             i,
                                                             sigma_noise,
                                                             p_year)

        B[n_stocks*n:n_stocks*(n+1), 0] = np.array(list(industries_beta_market[i]))
        B[n_stocks*n:n_stocks*(n+1), n+1] =
np.array(list(industries_beta_industries[i]))

        stocks_all = pd.concat([stocks_all, industries_stocks[i]], axis=1)
        expected_returns_all = expected_returns_all.append(industries_mu[i])

        industries_weights[i] = optimize.minimize_objective(
             industries_mu[i].index,
             optimize.negative_sharpe,
             True,
             (-1, 1),
             industries_mu[i], industries_stocks[i][:n_train].cov(),
             0.0, 0.0)
        industries_portfolio[i] = np.dot(
            industries_stocks[i].values,
            np.array(list(industries_weights[i].values())))

    if PLOT_STOCKS:
        plot_equity_curve(stocks_all)

    # ground thruth
```

```
    Sigma_f = np.diag([sigma[i]**2 for i in sigma.keys()])
    Sigma_e = np.diag([sigma_noise**2]*B.shape[0])
    cov_truth = pd.DataFrame(B.dot(Sigma_f).dot(np.transpose(B))) + Sigma_e
    weights = optimize.minimize_objective(expected_returns_all.index,
                                          optimize.negative_sharpe,
                                          True,
                                          (-1, 1),
                                          expected_returns_all, cov_truth,
                                          0.0, 0.0,)

    portfolio_gt_is = np.dot(stocks_all.values,
                             np.array(list(weights.values())))[:n_train]
    IR_ann = information_ratio(portfolio_gt_is)
    ir_gt_is.append(IR_ann)
    print('IR gt_is: ', IR_ann)

    portfolio_gt_oos = np.dot(stocks_all.values,
                              np.array(list(weights.values())))[n_train:]
    IR_ann = information_ratio(portfolio_gt_oos)
    ir_gt_oos.append(IR_ann)
    print('IR gt_oos: ', IR_ann)

    # standard sample moments
    weights = optimize.minimize_objective(expected_returns_all.index,
                                          optimize.negative_sharpe,
                                          True,
                                          (-1, 1),
                                          expected_returns_all,
                                          stocks_all[:n_train].cov(),
                                          0.0, 0.0,)

    portfolio_sm_is = np.dot(stocks_all.values,
                             np.array(list(weights.values())))[:n_train]
    IR_ann = information_ratio(portfolio_sm_is)
    ir_sm_is.append(IR_ann)
    print('IR sm_is: ', IR_ann)

    portfolio_sm_oos = np.dot(stocks_all.values,
                              np.array(list(weights.values())))[n_train:]
    IR_ann = information_ratio(portfolio_sm_oos)
    ir_sm_oos.append(IR_ann)
    print('IR sm_oos: ', IR_ann)

    # hierarchical
    # optimize allocation to industry
    industry_weights = optimize.minimize_objective(
        industries.keys(),
        optimize.negative_sharpe,
        False,
        (-1, 1),
        pd.Series(index=industries.keys(),
        data=[0.1]*len(industries.keys())),
        pd.DataFrame(industries_portfolio).cov()[:n_train],
        0.0, 0.0)

#     # equal weighting industries
#     for i in industry_weights.keys():
#         industry_weights[i] = 1/len(industries)

    print(industry_weights)
    portfolio_hier_is = np.dot(pd.DataFrame(industries_portfolio).values,
                        np.array(list(industry_weights.values())))[:n_train]

    IR_ann = information_ratio(portfolio_hier_is)
    print('IR hier_is: ', IR_ann)
    ir_hier_is.append(IR_ann)

    portfolio_hier_oos = np.dot(pd.DataFrame(industries_portfolio).values,
```

```python
                          np.array(list(industry_weights.values()))[n_train:]

        IR_ann = information_ratio(portfolio_hier_oos)
        print('IR hier_oos: ', IR_ann)
        ir_hier_oos.append(IR_ann)

        equal_weights = expected_returns_all.apply(np.sign)/expected_returns_all.shape[0]

        # Make same gross leverage
        equal_weights = equal_weights*np.abs(np.array(list(weights.values()))).sum()

        portfolio_equal = np.dot(stocks_all.values,
                          equal_weights.values)
        portfolio_equal_is = portfolio_equal[:n_train]
        portfolio_equal_oos = portfolio_equal[n_train:]

        IR_ann = information_ratio(portfolio_equal_is)
        print('IR equal_is: ', IR_ann)
        ir_equal_is.append(IR_ann)

        IR_ann = information_ratio(portfolio_equal_oos)
        print('IR equal_oos: ', IR_ann)
        ir_equal_oos.append(IR_ann)
```

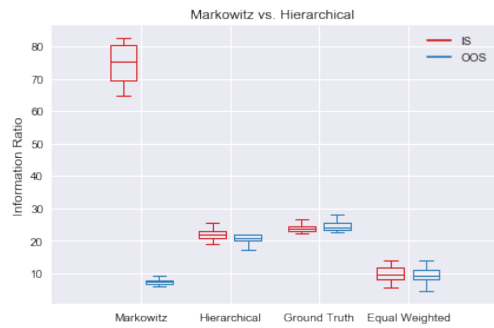
```
MC run:  0
IR gt_is:  22.97727534341568
IR gt_oos:  25.601650253512382


/Users/jan/Documents/PoCon/src/optimize.py:30: UserWarning: Optimizer did not
converge.
  warnings.warn("Optimizer did not converge.")


IR sm_is:  68.86642974717553
IR sm_oos:  6.699566592572767
{'banks': 0.08525765766039736, 'oil': 0.026799533838017413, 'insurance':
0.020444231421016416, 'tech': 0.117966834969583, 'bio': 0.30235562609587247, 'pharma':
0.06960329827570759, 'auto': 0.14206749885277828, 'retail': 0.19377667069590016,
'manufacturing': 0.041728648190727496}
IR hier_is:  20.8116940244627
IR hier_oos:  21.865542855743392
IR equal_is:  7.95182566091046
IR equal_oos:  7.326006935456171
MC run:  1
IR gt_is:  26.771132066373774
IR gt_oos:  28.159064152018402
IR sm_is:  110.09206477273513
IR sm_oos:  7.5479680514651175
{'banks': 0.044190128115900604, 'oil': 0.1276822372866994, 'insurance':
0.038706296568006365, 'tech': 0.06337915844064392, 'bio': 0.13878794406853137,
'pharma': 0.07518201151804653, 'auto': 0.07711013777617716, 'retail':
0.25935387414437844, 'manufacturing': 0.1756082120816163}
IR hier_is:  25.662726830273943
IR hier_oos:  25.388872828147313
IR equal_is:  13.832612014560612
IR equal_oos:  13.870769920429083
MC run:  2
IR gt_is:  22.12498465423602
IR gt_oos:  24.994628015545
IR sm_is:  73.95084565620881
IR sm_oos:  7.313627885190998
```

```
{'banks': 0.09533003742104401, 'oil': 0.13281804748440268, 'insurance':
0.010599084773217502, 'tech': 0.1870967422369644, 'bio': 0.0430765017518334, 'pharma':
0.019647395250293832, 'auto': 0.06477985552847948, 'retail': 0.21783365245635233,
'manufacturing': 0.22881868309741218}
IR hier_is:  21.445295159373668
IR hier_oos:  21.74898654991655
IR equal_is:  13.022774599594436
IR equal_oos:  13.431445532802027
MC run:  3
IR gt_is:  24.783685587866106
IR gt_oos:  23.964302612640793
IR sm_is:  76.90436023202933
IR sm_oos:  9.289079575452817
{'banks': 0.10178730748467713, 'oil': 0.12063615385680393, 'insurance':
0.007842505585361351, 'tech': 0.13095328977379267, 'bio': 0.15555816144859244,
'pharma': 0.2647333159484891, 'auto': 0.04527157255757351, 'retail':
0.15598235608121774, 'manufacturing': 0.01723533726349217}
IR hier_is:  23.11903322407622
IR hier_oos:  20.709967158211672
IR equal_is:  12.245319437972624
IR equal_oos:  11.768506071460667
MC run:  4
IR gt_is:  25.405717149410922
IR gt_oos:  23.83151324844424
IR sm_is:  71.91830187636323
IR sm_oos:  7.893799804413511
{'banks': 0.008331797624099297, 'oil': 0.1422152742418836, 'insurance':
0.07117306958746743, 'tech': 0.05683536741120898, 'bio': 0.03093410182989959,
'pharma': 0.30839849778761186, 'auto': 0.029121474729654418, 'retail':
0.16156160489816657, 'manufacturing': 0.1914288118900083}
IR hier_is:  22.047307244967097
IR hier_oos:  19.74034703920951
IR equal_is:  10.24930695194757
IR equal_oos:  9.26763811882215
MC run:  5
IR gt_is:  23.774046760263385
IR gt_oos:  24.416799220816827
IR sm_is:  82.51159270276858
IR sm_oos:  7.073926437980247
{'banks': 0.019026711412274785, 'oil': 0.10069088719525802, 'insurance':
0.01738849710027885, 'tech': 0.3363406418462935, 'bio': 0.11936447738914333, 'pharma':
0.1903819517161351, 'auto': 0.05639672497940893, 'retail': 0.12447721248140026,
'manufacturing': 0.03593289587980716}
IR hier_is:  21.524876253053215
IR hier_oos:  20.608853356276203
IR equal_is:  10.847118607900063
IR equal_oos:  9.382224029908766
MC run:  6
IR gt_is:  22.871741160100328
IR gt_oos:  22.487076322170182
IR sm_is:  64.75344379654321
IR sm_oos:  6.323567513587528
{'banks': 0.17029698533312862, 'oil': 0.21106271552184303, 'insurance':
0.042080354837382324, 'tech': 0.03008387395067035, 'bio': 0.007005001847849584,
'pharma': 0.3297258523256762, 'auto': 0.08767800263458783, 'retail':
0.01150600681945518, 'manufacturing': 0.11056120672940697}
IR hier_is:  20.82868495138105
IR hier_oos:  18.00323300546353
IR equal_is:  8.029881843476653
```

```
IR equal_oos:  7.893597017400724
MC run:  7
IR gt_is:  22.9079517433364
IR gt_oos:  23.2071757725262
IR sm_is:  76.91780583251605
IR sm_oos:  6.088767171800019
{'banks': 0.007013241716007736, 'oil': 0.18128528150435333, 'insurance':
0.011100832403692869, 'tech': 0.008538933940442348, 'bio': 0.12330362371058244,
'pharma': 0.35716281397695776, 'auto': 0.09910541317827144, 'retail':
0.19539604560933388, 'manufacturing': 0.01709381396035819}
IR hier_is:  18.953515732558117
IR hier_oos:  17.30131100375008
IR equal_is:  5.550279675144221
IR equal_oos:  4.445961826692587
MC run:  8
IR gt_is:  23.605123461477735
IR gt_oos:  23.264781570255654
IR sm_is:  66.61738057882724
IR sm_oos:  6.43578242236054
{'banks': 0.06607187976300144, 'oil': 0.04501823053835708, 'insurance':
0.11938848460520698, 'tech': 0.023104508754752045, 'bio': 0.09762295910249473,
'pharma': 0.18800636988316546, 'auto': 0.02543435582262118, 'retail':
0.21784042489955452, 'manufacturing': 0.21751278663084656}
IR hier_is:  23.804200363766427
IR hier_oos:  20.762433881352706
IR equal_is:  7.8827421616603885
IR equal_oos:  8.720606082070747
MC run:  9
IR gt_is:  23.85084213207428
IR gt_oos:  26.2639814301371
IR sm_is:  81.30892790285837
IR sm_oos:  7.701145910218168
{'banks': 0.03027463504519667, 'oil': 0.1856023565331236, 'insurance':
0.019317517946394445, 'tech': 0.14784769811725806, 'bio': 0.2838690145717053,
'pharma': 0.07212580567407506, 'auto': 0.033317365127124576, 'retail':
0.196812283035799, 'manufacturing': 0.030833323949323293}
IR hier_is:  22.17972270765723
IR hier_oos:  22.014619742221665
IR equal_is:  8.934768780668415
IR equal_oos:  8.982252859122386
```

In [9]:
```python
data_a = [ir_sm_is, ir_hier_is, ir_gt_is, ir_equal_is]
data_b = [ir_sm_oos, ir_hier_oos, ir_gt_oos, ir_equal_oos]

ticks = ['Markowitz', 'Hierarchical', 'Ground Truth', 'Equal Weighted']

def set_box_color(bp, color):
    plt.setp(bp['boxes'], color=color)
    plt.setp(bp['whiskers'], color=color)
    plt.setp(bp['caps'], color=color)
    plt.setp(bp['medians'], color=color)

plt.figure()

bp1 = plt.boxplot(data_a, positions=np.array(range(len(data_a)))*2.0-0.4, sym='',
widths=0.6)
bp2 = plt.boxplot(data_b, positions=np.array(range(len(data_b)))*2.0+0.4, sym='',
widths=0.6)

set_box_color(bp1, '#D7191C') # colors are from http://colorbrewer2.org/
set_box_color(bp2, '#2C7BB6')
```

```
# draw temporary red and blue lines and use them to create a legend
plt.plot([], c='#D7191C', label='IS')
plt.plot([], c='#2C7BB6', label='OOS')
plt.legend()
plt.title('Markowitz vs. Hierarchical ')
plt.ylabel('Information Ratio')
plt.xticks(range(0, len(ticks) * 2, 2), ticks)
plt.xlim(-2, len(ticks)*2)
# plt.ylim(0, 8)
plt.tight_layout()
# plt.savefig('boxcompare.png')
```




```
In [10]: performance_factor = np.mean(ir_equal_oos)/np.mean(ir_sm_oos)
         print('OOS Equal / Markowitz: ', performance_factor)
```

```
OOS Equal / Markowitz:  1.3139788078185306
```

```
In [11]: performance_factor = np.mean(ir_hier_oos)/np.mean(ir_sm_oos)
         print('OOS Hierarchical / Markowitz: ', performance_factor)
```

```
OOS Hierarchical / Markowitz:  2.8762212329272607
```

```
In [12]: performance_factor = np.mean(ir_hier_oos)/np.mean(ir_equal_oos)
         print('OOS Hierarchical / Equal: ', performance_factor)
```

```
OOS Hierarchical / Equal:  2.1889403511022882
```

The in-sample Markowitz IR is way above the ground truth IR. This may seem curious, but is explained by the overfitting to in-sample data. Whenever we are overfitting to in-sample data we can expect the out-of-sample performance to suffer. This phenomenon is nicely illustrated by the abysmal out-of-sample IR. The structure imposed by the hierarchical method causes both the in-sample and out-of-sample performance to be much closer to the ground truth. When the number of assets is sufficiently high, equal weighting outperforms Markowitz. The hierarchical method can improve upon equal weighting almost always. Note that in this context equal weighting does not imply blindly allocating equal weights to longs and short *regardless of the industry*. Instead, it assumes that in a prior step the alpha model picked roughly equal numbers of longs and shorts of a certain industry. This follows since the deterministic trend μ is sampled uniformly with mean zero. It is more appropriately thought of as a industry-matched equal weighting scheme.

```
In [13]: # create a portfolio object and print some method outputs
         pf = Portfolio(assets=weights.keys(),
                        position=pd.Series(weights),
                        price=[1]*len(weights.keys()),
                        sector_id=pd.Series(list(weights.keys())).str[:3].values
                       )
         print(pf)
         print('largest long:', pf[0], pf.position(pf[0]))
         print('largest short:', pf[-1], pf.position(pf[-1]))
         print('sector_net_exposures: \n', pf.sector_net_exposures())
```

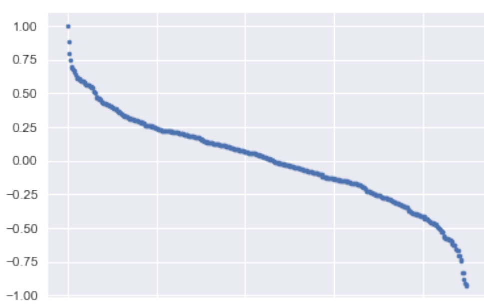
```
Portfolio: 450 Assets, $59.96 Long, $59.96 Short
largest long: banks_stock_45 0.9999573360418466
largest short: insurance_stock_40 -0.9249530915414368
sector_net_exposures:
            position_value
sector_id
aut             -0.398064
ban              0.166075
bio             -0.423796
ins             -0.594435
man              0.690644
oil             -1.106129
pha              1.326872
ret              0.459944
tec             -0.121112
```

```
In [14]: print(pf.portfolio_df.head())
```

```
                       position  price  factor_value sector_id  position_value
banks_stock_45         0.999957      1           1.0       ban        0.999957
insurance_stock_14     0.879602      1           1.0       ins        0.879602
manufacturing_stock_2  0.791352      1           1.0       man        0.791352
pharma_stock_44        0.750419      1           1.0       pha        0.750419
insurance_stock_4      0.696782      1           1.0       ins        0.696782
```

```
In [15]: pf.portfolio_df.position_value.plot(style='.')
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1bc1e438>
```



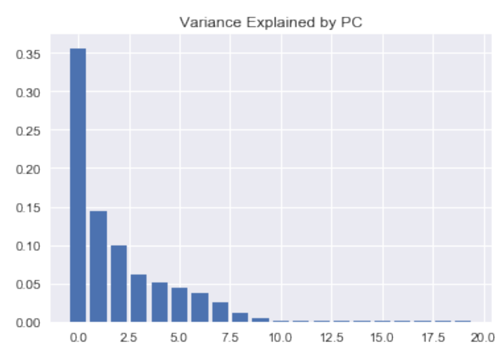
1.2 Principal Component Analysis

Principal Component Analysis (PCA) allows us to reduce the rank of the covariance matrix. Since the covariance matrix is symmetric we can decompose the matrix into its real eigenvalues and orthonormal eigenvectors. This follows from the spectral theorem.

$$S = Q\Lambda Q^{-1} = Q\Lambda Q^{\mathrm{T}} \quad \text{with} \quad Q^{-1} = Q^{\mathrm{T}}$$

In the simulation we have exposure to the market and several industries. Thus it makes sense to reduce the rank to the number of these variables. Plotting the proportion of variance explained by the first n principal components confirms this hypothesis.

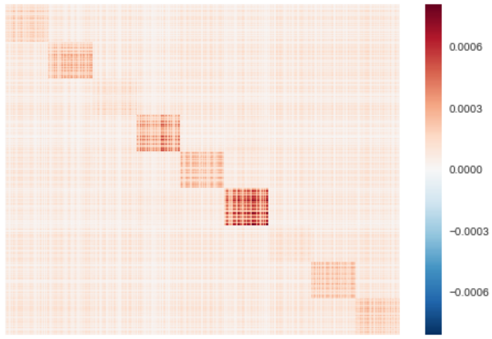
```
In [16]: from sklearn.decomposition import PCA as sklearnPCA

         n_components = 20
         sklearn_pca = sklearnPCA(n_components=n_components)
         pc = sklearn_pca.fit_transform(stocks_all)

         plt.bar(range(n_components),sklearn_pca.explained_variance_ratio_)
         plt.title('Variance Explained by PC')
         plt.show()
```




```
In [17]: # denoise covariance matrix
         cov = stocks_all.cov()
         n = len(industries)+1

         evalues, Q = np.linalg.eig(cov)
         evalues[evalues < evalues[n_components]] = 0
         Q_T = np.matrix.transpose(Q)
         L = np.diag(evalues)
         pc_cov = pd.DataFrame(Q.dot(L).dot(Q_T)).apply(np.real)
```

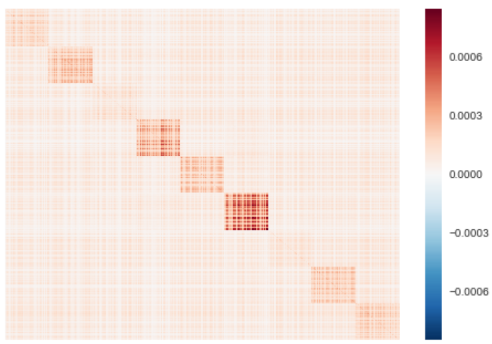
```
In [18]: import seaborn as sns
         sns.heatmap(pc_cov, xticklabels=False, yticklabels=False)
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c868ba8>
```



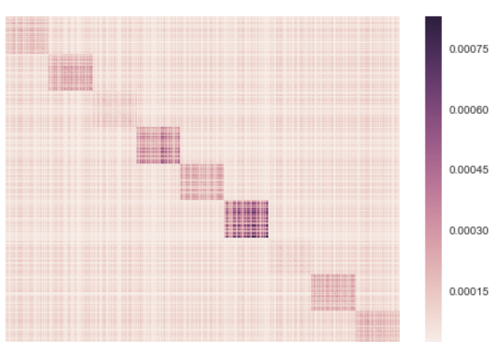

```
In [19]: # plot sample covariance matrix to compare
         sns.heatmap(cov, xticklabels=False, yticklabels=False)
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c0dc898>




```
In [20]: # compute ground truth covariance matrix to compare
         Sigma_f = np.diag([sigma[i]**2 for i in sigma.keys()])
         Sigma_e = np.diag([sigma_noise**2]*B.shape[0])
         cov_truth = pd.DataFrame(B.dot(Sigma_f).dot(np.transpose(B))) + Sigma_e
         sns.heatmap(cov_truth, xticklabels=False, yticklabels=False)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1d5a6390>



The heatmaps above look almost indistinguishable after eliminating all other components.

```
In [21]: trainig_pct = 0.5
         n_train = int(trainig_pct*num_samples)
         ir_sm_is = []
         ir_sm_oos = []
         ir_pca_is = []
         ir_pca_oos = []
         ir_gt_is = []
         ir_gt_oos = []
         n_stocks = 50

         for j in range(MC_RUNS):
             print('MC run: ', j)

             market = norm.rvs(size=(1, num_samples))[0]*sigma['market']
             industries = {}
             industries['banks'] =  norm.rvs(size=(1, num_samples))[0]*sigma['banks']
             industries['oil'] = norm.rvs(size=(1, num_samples))[0]*sigma['oil']
             industries['insurance'] = norm.rvs(size=(1, num_samples))[0]*sigma['insurance']
             industries['tech'] = norm.rvs(size=(1, num_samples))[0]*sigma['tech']
             industries['bio'] = norm.rvs(size=(1, num_samples))[0]*sigma['bio']
             industries['pharma'] = norm.rvs(size=(1, num_samples))[0]*sigma['pharma']
             industries['auto'] = norm.rvs(size=(1, num_samples))[0]*sigma['auto']
             industries['retail'] = norm.rvs(size=(1, num_samples))[0]*sigma['retail']
             industries['manufacturing'] = norm.rvs(size=(1,
         num_samples))[0]*sigma['manufacturing']

             industries_stocks = {}
             industries_mu = {}
             industries_beta_market = {}
             industries_beta_industry = {}
             industries_weights = {}
             industries_portfolio = {}

             stocks_all = pd.DataFrame()
             expected_returns_all = pd.Series()

             B = np.zeros((n_stocks*len(industries.keys()), len(sigma)))
             for n, i in enumerate(industries.keys()):
                 industries_stocks[i],\
                 industries_mu[i],\
                 industries_beta_market[i],\
                 industries_beta_industry[i] = gen_industry_stocks(n_stocks,
                                                                   market,
                                                                   industries[i],
                                                                   i,
                                                                   sigma_noise,
                                                                   p_year)
                 B[n_stocks*n:n_stocks*(n+1), 0] = np.array(list(industries_beta_market[i]))
                 B[n_stocks*n:n_stocks*(n+1), n+1] = np.array(list(industries_beta_industry[i]))

                 stocks_all = pd.concat([stocks_all,industries_stocks[i]], axis=1)
                 expected_returns_all = expected_returns_all.append(industries_mu[i])

             if PLOT_STOCKS:
                 plot_equity_curve(stocks_all)

             # ground thruth
             Sigma_f = np.diag([sigma[i]**2 for i in sigma.keys()])
             Sigma_e = np.diag([sigma_noise**2]*B.shape[0])
             cov_truth = pd.DataFrame(B.dot(Sigma_f).dot(np.transpose(B))) + Sigma_e
             weights = optimize.minimize_objective(expected_returns_all.index,
                                                   optimize.negative_sharpe,
                                                   True,
```

```
                                         (-1, 1),
                                         expected_returns_all, cov_truth,
                                         0.0, 0.0,)

        portfolio_gt_is = np.dot(stocks_all.values,
                                 np.array(list(weights.values())))[:n_train]
        IR_ann = information_ratio(portfolio_gt_is)
        ir_gt_is.append(IR_ann)
        print('IR gt_is: ', IR_ann)

        portfolio_gt_oos = np.dot(stocks_all.values,
                                  np.array(list(weights.values())))[n_train:]
        IR_ann = information_ratio(portfolio_gt_oos)
        ir_gt_oos.append(IR_ann)
        print('IR gt_oos: ', IR_ann)

        # standard sample moments
        cov = stocks_all[:n_train].cov()
        weights = optimize.minimize_objective(expected_returns_all.index,
                                         optimize.negative_sharpe,
                                         True,
                                         (-1, 1),
                                         expected_returns_all, cov,
                                         0.0, 0.0,)

        portfolio_sm_is = np.dot(stocks_all.values,
                                 np.array(list(weights.values())))[:n_train]
        IR_ann = information_ratio(portfolio_sm_is)
        ir_sm_is.append(IR_ann)
        print('IR sm_is: ', IR_ann)

        portfolio_sm_oos = np.dot(stocks_all.values,
                                  np.array(list(weights.values())))[n_train:]
        IR_ann = information_ratio(portfolio_sm_oos)
        ir_sm_oos.append(IR_ann)
        print('IR sm_oos: ', IR_ann)

        # PCA
        n = len(industries)+1
        pc_cov = optimize.pc_cov(cov, n)
        weights = optimize.minimize_objective(expected_returns_all.index,
                                         optimize.negative_sharpe,
                                         True,
                                         (-1, 1),
                                         expected_returns_all, pc_cov,
                                         0.0, 0.0,)

        portfolio_pca_is = np.dot(stocks_all.values,
                                  np.array(list(weights.values())))[:n_train]
        IR_ann = information_ratio(portfolio_pca_is)
        print('IR pca_is: ', IR_ann)
        ir_pca_is.append(IR_ann)
        portfolio_pca_oos = np.dot(stocks_all.values,
                                   np.array(list(weights.values())))[n_train:]
        IR_ann = information_ratio(portfolio_pca_oos)
        print('IR pca_oos: ', IR_ann)
        ir_pca_oos.append(IR_ann)
```

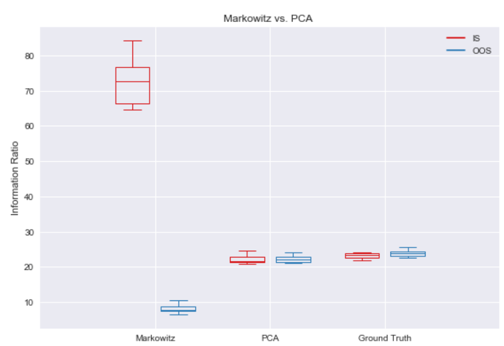
```
MC run:  0
IR gt_is:  23.928091331371824
IR gt_oos:  25.690924746070923
```

```
/Users/jan/Documents/PoCon/src/optimize.py:30: UserWarning: Optimizer did not
converge.
  warnings.warn("Optimizer did not converge.")
```

```
IR sm_is:  70.31358258914639
IR sm_oos:  8.820658092799544
IR pca_is:  23.02183181140094
IR pca_oos:  24.185608362682228
MC run:  1
IR gt_is:  23.483157047813364
IR gt_oos:  23.201994695294516
IR sm_is:  84.1651737323713
IR sm_oos:  7.706546516065167
IR pca_is:  21.41364071779635
IR pca_oos:  21.43571944493367
MC run:  2
IR gt_is:  22.28053644927649
IR gt_oos:  23.381364215926908
IR sm_is:  77.93833732468957
IR sm_oos:  7.749301682353445
IR pca_is:  21.432762259498002
IR pca_oos:  21.81894849118108
MC run:  3
IR gt_is:  23.455531871051292
IR gt_oos:  24.01300349901305
IR sm_is:  65.42997462402077
IR sm_oos:  7.6775763122535405
IR pca_is:  23.021110673153178
IR pca_oos:  22.849885724873147
MC run:  4
IR gt_is:  24.184545118553274
IR gt_oos:  24.30887580550335
IR sm_is:  69.11615485823607
IR sm_oos:  10.487583844591857
IR pca_is:  22.93057948612109
IR pca_oos:  23.17622141063133
MC run:  5
IR gt_is:  26.414787971437303
IR gt_oos:  22.95311175767001
IR sm_is:  76.88479408716702
IR sm_oos:  8.81001811360005
IR pca_is:  24.65964285823324
IR pca_oos:  21.08724797685001
MC run:  6
IR gt_is:  23.117734276665452
IR gt_oos:  24.539536043691076
IR sm_is:  65.50914915015383
IR sm_oos:  7.497741821965262
IR pca_is:  21.859366534100225
IR pca_oos:  22.482987575194088
MC run:  7
IR gt_is:  22.488948436840026
IR gt_oos:  22.576062710794744
IR sm_is:  75.00567964723727
IR sm_oos:  6.4545595153664825
IR pca_is:  21.597449811693807
IR pca_oos:  21.12812007140757
MC run:  8
IR gt_is:  21.87509337251639
IR gt_oos:  24.458493756190276
IR sm_is:  64.5012049016925
IR sm_oos:  8.950179482518198
IR pca_is:  20.792584629494716
```

```
IR pca_oos:  22.675317968867915
MC run:  9
IR gt_is:  22.947254544337085
IR gt_oos:  23.545244952815107
IR sm_is:  76.48684538500122
IR sm_oos:  6.737335116130717
IR pca_is:  21.517691968484655
IR pca_oos:  21.664761442031843
```

In [22]:
```python
data_a = [ir_sm_is, ir_pca_is, ir_gt_is]
data_b = [ir_sm_oos, ir_pca_oos, ir_gt_oos]

ticks = ['Markowitz', 'PCA', 'Ground Truth']

def set_box_color(bp, color):
    plt.setp(bp['boxes'], color=color)
    plt.setp(bp['whiskers'], color=color)
    plt.setp(bp['caps'], color=color)
    plt.setp(bp['medians'], color=color)

plt.figure()

bp1 = plt.boxplot(data_a, positions=np.array(range(len(data_a)))*2.0-0.4, sym='',
widths=0.6)
bp2 = plt.boxplot(data_b, positions=np.array(range(len(data_b)))*2.0+0.4, sym='',
widths=0.6)

set_box_color(bp1, '#D7191C') # colors are from http://colorbrewer2.org/
set_box_color(bp2, '#2C7BB6')

# draw temporary red and blue lines and use them to create a legend
plt.plot([], c='#D7191C', label='IS')
plt.plot([], c='#2C7BB6', label='OOS')
plt.legend()
plt.title('Markowitz vs. PCA ')
plt.ylabel('Information Ratio')
plt.xticks(range(0, len(ticks) * 2, 2), ticks)
plt.xlim(-2, len(ticks)*2)
# plt.ylim(0, 8)
plt.tight_layout()
# plt.savefig('boxcompare.png')
```



In [23]:
```python
performance_factor = np.mean(ir_pca_oos)/np.mean(ir_sm_oos)
print('OOS PCA / Markowitz: ', performance_factor)
```

```
OOS PCA / Markowitz:  2.750657573413819
```

```
In [24]: print(np.linalg.matrix_rank(cov))
         print(np.linalg.matrix_rank(pc_cov))
```

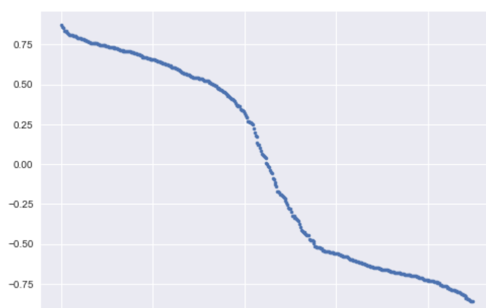
```
450
11
```

```
In [25]: pf = Portfolio(assets=weights.keys(),
                        position=pd.Series(weights),
                        price=[1]*len(weights.keys()),
                        sector_id=pd.Series(list(weights.keys())).str[:3].values)
         print(pf)
         print('largest long:', pf[0], pf.position(pf[0]))
         print('largest short:', pf[-1], pf.position(pf[-1]))
         print('sector_net_exposures:\n', pf.sector_net_exposures())
```

```
Portfolio: 450 Assets, $131.33 Long, $131.33 Short
largest long: bio_stock_18 0.8707197141374607
largest short: bio_stock_10 -0.8608912496009911
sector_net_exposures:
             position_value
sector_id
aut               0.409164
ban              -1.761326
bio               2.450520
ins               0.550541
man              -1.168924
oil              -1.701275
pha               0.933427
ret               2.341407
tec              -2.053537
```

```
In [26]: pf.portfolio_df.position_value.plot(style='.')
```

```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b76bbe0>
```



PCA reduced extreme positions and set a lot fewer weights close to zero.

1.3 Robustness under fat-tailed noise distribution

The Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model is defined by:

$$r_t = \mu_t + \epsilon_t$$
$$\epsilon_t = \sigma_t \eta_t, \quad \eta_t \overset{iid}{\sim} (0,1)$$
$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2$$
$$\omega > 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, q, \quad \beta_i \geq 0, \quad i = 1, \ldots, p$$

The unconditional variance becomes:

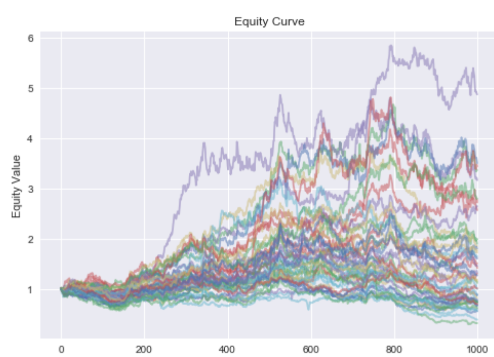
$$\begin{aligned}
\mathrm{E}\left(\sigma_t^2\right) &= \omega + \sum_{i=1}^{q} \alpha_i \mathrm{E}\left(\epsilon_{t-i}^2\right) + \sum_{i=1}^{p} \beta_i \mathrm{E}\left(\sigma_{t-i}^2\right) \\
&= \omega + \sum_{i=1}^{q} \alpha_i \mathrm{E}\left(\eta_{t-i}^2 \sigma_{t-i}^2\right) + \sum_{i=1}^{p} \beta_i \mathrm{E}\left(\sigma_{t-i}^2\right) \\
&= \omega + \sum_{i=1}^{q} \alpha_i \underbrace{\mathrm{E}\left(\eta_{t-i}^2\right)}_{=1} \mathrm{E}\left(\sigma_{t-i}^2\right) + \sum_{i=1}^{p} \beta_i \mathrm{E}\left(\sigma_{t-i}^2\right) \\
&= \omega + \left(\sum_{i=1}^{q} \alpha_i + \sum_{i=1}^{p} \beta_i\right) \mathrm{E}\left(\sigma_t^2\right)
\end{aligned}$$

Solving for $\mathrm{E}\left(\sigma_t^2\right)$, we have the unconditional variance

$$\begin{aligned}
\mathrm{E}\left(\epsilon_t^2\right) &= \mathrm{E}\left(\sigma_t^2\right) \\
&= \frac{\omega}{1 - \sum_{i=1}^{q} \alpha_i - \sum_{i=1}^{p} \beta_i}
\end{aligned}$$

Equity curves under this noise distribution look like this.

```
In [27]: n_stocks = 50
         garch_mu = 0
         garch_alpha = 0.5
         garch_beta = 0.3
         market = norm.rvs(size=(1, num_samples))[0]*sigma['market']
         banks = norm.rvs(size=(1, num_samples))[0]*sigma['banks']
         stocks, *_ = gen_industry_stocks_garch(n_stocks, market, banks, 'banks',
                                  sigma_noise, p_year, garch_mu,
                                  garch_alpha, garch_beta)
         plot_equity_curve(stocks)
```



As you can see, returns are much more extreme than under Gaussian noise.

First let's compare Markowitz vs. hierachrichal vs. equal weighting under a GARCH(1,1) noise distribution.

```
In [28]: trainig_pct = 0.5
         n_train = int(trainig_pct*num_samples)
         ir_sm_is = []
         ir_sm_oos = []
         ir_hier_is = []
         ir_hier_oos = []
         ir_gt_is = []
         ir_gt_oos = []
         ir_equal_is = []
         ir_equal_oos = []
         n_stocks = 50

         garch_mu = 0
         garch_alpha = 0.5
         garch_beta = 0.3


         for j in range(MC_RUNS):
             print('MC run: ', j)

             market = norm.rvs(size=(1, num_samples))[0]*sigma['market']
             industries = {}
             industries['banks'] =  norm.rvs(size=(1, num_samples))[0]*sigma['banks']
             industries['oil'] = norm.rvs(size=(1, num_samples))[0]*sigma['oil']
             industries['insurance'] = norm.rvs(size=(1, num_samples))[0]*sigma['insurance']
             industries['tech'] = norm.rvs(size=(1, num_samples))[0]*sigma['tech']
             industries['bio'] = norm.rvs(size=(1, num_samples))[0]*sigma['bio']
             industries['pharma'] = norm.rvs(size=(1, num_samples))[0]*sigma['pharma']
             industries['auto'] = norm.rvs(size=(1, num_samples))[0]*sigma['auto']
             industries['retail'] = norm.rvs(size=(1, num_samples))[0]*sigma['retail']
             industries['manufacturing'] = norm.rvs(size=(1,
         num_samples))[0]*sigma['manufacturing']

             industries_stocks = {}
             industries_mu = {}
             industries_beta_market = {}
             industries_beta_industries = {}
             industries_weights = {}
             industries_portfolio = {}

             stocks_all = pd.DataFrame()
             expected_returns_all = pd.Series()

             B = np.zeros((n_stocks*len(industries.keys()), len(sigma)))
             for n, i in enumerate(industries.keys()):
                 industries_stocks[i],\
                 industries_mu[i],\
                 industries_beta_market[i],\
                 industries_beta_industries[i] = gen_industry_stocks_garch(n_stocks,
                                                                           market,
                                                                           industries[i],
                                                                           i,
                                                                           sigma_noise,
                                                                           p_year,
                                                                           garch_mu,
                                                                           garch_alpha,
                                                                           garch_beta)

                 B[n_stocks*n:n_stocks*(n+1), 0] = np.array(list(industries_beta_market[i]))
                 B[n_stocks*n:n_stocks*(n+1), n+1] =
         np.array(list(industries_beta_industries[i]))

                 stocks_all = pd.concat([stocks_all,industries_stocks[i]], axis=1)
                 expected_returns_all = expected_returns_all.append(industries_mu[i])

                 industries_weights[i] = optimize.minimize_objective(industries_mu[i].index,
                                              optimize.negative_sharpe,
                                              True,
```

```
                                     (-1, 1),
                                     industries_mu[i],
industries_stocks[i][:n_train].cov(),
                                     0.0, 0.0,)
        industries_portfolio[i] = np.dot(industries_stocks[i].values,
                                         np.array(list(industries_weights[i].values())))
    if PLOT_STOCKS:
        plot_equity_curve(stocks_all)

    # ground truth
    Sigma_f = np.diag([sigma[i]**2 for i in sigma.keys()])
    # GARCH uncond Var
    Sigma_e = np.diag([(sigma_noise**2)/(1-garch_alpha-garch_beta)]*B.shape[0])
    cov_truth = pd.DataFrame(B.dot(Sigma_f).dot(np.transpose(B))) + Sigma_e
    weights = optimize.minimize_objective(expected_returns_all.index,
                                         optimize.negative_sharpe,
                                         True,
                                         (-1, 1),
                                         expected_returns_all, cov_truth,
                                         0.0, 0.0)


    portfolio_gt_is = np.dot(stocks_all.values,
                             np.array(list(weights.values())))[:n_train]
    IR_ann = information_ratio(portfolio_gt_is)
    ir_gt_is.append(IR_ann)
    print('IR gt_is: ', IR_ann)

    portfolio_gt_oos = np.dot(stocks_all.values,
                              np.array(list(weights.values())))[n_train:]
    IR_ann = information_ratio(portfolio_gt_oos)
    ir_gt_oos.append(IR_ann)
    print('IR gt_oos: ', IR_ann)

    #standard sample moments
    weights = optimize.minimize_objective(expected_returns_all.index,
                                         optimize.negative_sharpe,
                                         True,
                                         (-1, 1),
                                         expected_returns_all,
                                         stocks_all[:n_train].cov(),
                                         0.0, 0.0)

    portfolio_sm_is = np.dot(stocks_all.values,
                             np.array(list(weights.values())))[:n_train]
    IR_ann = information_ratio(portfolio_sm_is)
    ir_sm_is.append(IR_ann)
    print('IR sm_is: ', IR_ann)

    portfolio_sm_oos = np.dot(stocks_all.values,
                              np.array(list(weights.values())))[n_train:]
    IR_ann = information_ratio(portfolio_sm_oos)
    ir_sm_oos.append(IR_ann)
    print('IR sm_oos: ', IR_ann)

    # hierarchical
    # optimize allocation to industry
    industry_weights = optimize.minimize_objective(industries.keys(),
                                                  optimize.negative_sharpe,
                                                  False,
                                                  (-1, 1),
                                                  pd.Series(index=industries.keys(),
                                                  data=[0.1]*len(industries.keys())),
pd.DataFrame(industries_portfolio).cov()[:n_train],
                                                  0.0, 0.0)
#     # equal weighting industries
#     for i in industry_weights.keys():
#         industry_weights[i] = 1/len(industries)
```

```python
            print(industry_weights)
            portfolio_hier_is = np.dot(pd.DataFrame(industries_portfolio).values,
                                       np.array(list(industry_weights.values())))[:n_train]

            IR_ann = information_ratio(portfolio_hier_is)
            print('IR hier_is: ', IR_ann)
            ir_hier_is.append(IR_ann)

            portfolio_hier_oos = np.dot(pd.DataFrame(industries_portfolio).values,
                                        np.array(list(industry_weights.values())))[n_train:]

            IR_ann = information_ratio(portfolio_hier_oos)
            print('IR hier_oos: ', IR_ann)
            ir_hier_oos.append(IR_ann)

            equal_weights = expected_returns_all.apply(np.sign)/expected_returns_all.shape[0]

            # Make same gross leverage
            equal_weights = equal_weights*np.abs(np.array(list(weights.values()))).sum()

            # Make same gross leverage
            equal_weights = equal_weights*np.abs(np.array(list(weights.values()))).sum()

            portfolio_equal = np.dot(stocks_all.values,
                                     equal_weights.values)
            portfolio_equal_is = portfolio_equal[:n_train]
            portfolio_equal_oos = portfolio_equal[n_train:]

            IR_ann = information_ratio(portfolio_equal_is)
            print('IR equal_is: ', IR_ann)
            ir_equal_is.append(IR_ann)

            IR_ann = information_ratio(portfolio_equal_oos)
            print('IR equal_oos: ', IR_ann)
            ir_equal_oos.append(IR_ann)
```

```
MC run:  0
IR gt_is:  11.381839784897373
IR gt_oos:  10.25005811476404
```

```
/Users/jan/Documents/PoCon/src/optimize.py:30: UserWarning: Optimizer did not
converge.
  warnings.warn("Optimizer did not converge.")
```

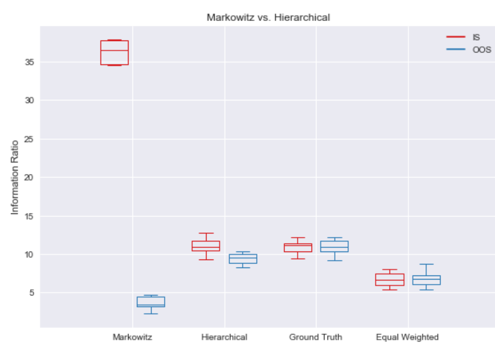
```
IR sm_is:  36.63963061230803
IR sm_oos:  4.447973650722289
{'banks': 0.05778225231788303, 'oil': 0.10428342897861984, 'insurance':
0.07543521741571599, 'tech': 0.31587014916029615, 'bio': 0.050455000852612675,
'pharma': 0.15935180704660465, 'auto': 0.04333117964647641, 'retail':
0.18252817857797649, 'manufacturing': 0.010962786003814857}
IR hier_is:  11.198458124852193
IR hier_oos:  8.235602345763404
IR equal_is:  7.580915620590376
IR equal_oos:  6.703342353268531
MC run:  1
IR gt_is:  9.427480157674227
IR gt_oos:  12.04492945891945
IR sm_is:  27.499891349533453
IR sm_oos:  4.447499004304111
{'banks': 0.06433997473727043, 'oil': 0.05918855736535658, 'insurance':
0.053130060258645535, 'tech': 0.32265337949361256, 'bio': 0.07564121246094861,
```

```
'pharma': 0.19243461205111537, 'auto': 0.0835758149828067, 'retail':
0.10935979291778583, 'manufacturing': 0.0396765957324584l}
IR hier_is:  9.347236155451956
IR hier_oos:  10.332460710116338
IR equal_is:  6.045709074937399
IR equal_oos:  6.950190038983944
MC run:  2
IR gt_is:  11.495035744085095
IR gt_oos:  10.326510454283552
IR sm_is:  37.81650733172764
IR sm_oos:  4.6807904980900155
{'banks': 0.12907389111307793, 'oil': 0.11710363764998055, 'insurance':
0.09084238467002748, 'tech': 0.1749535621009709, 'bio': 0.06683672435545432, 'pharma':
0.24160239479144172, 'auto': 0.05556876003657864, 'retail': 0.04552629225782874,
'manufacturing': 0.07849235302463957}
IR hier_is:  12.783302981964487
IR hier_oos:  9.905665791267666
IR equal_is:  6.3807942867290315
IR equal_oos:  5.414102910618966
MC run:  3
IR gt_is:  11.17564352096621
IR gt_oos:  9.213171446595831
IR sm_is:  43.32939425983168
IR sm_oos:  3.3037538831891653
{'banks': 0.02680541279426938, 'oil': 0.06408584330920583, 'insurance':
0.07067009006568603, 'tech': 0.08334107671773831, 'bio': 0.01666844778347558,
'pharma': 0.3354506367659482, 'auto': 0.0719435939908445, 'retail':
0.2408071285866011, 'manufacturing': 0.09022776998623104}
IR hier_is:  10.495916818712914
IR hier_oos:  8.243207986145178
IR equal_is:  7.139216341552791
IR equal_oos:  6.162957550168578
MC run:  4
IR gt_is:  11.01799937180968
IR gt_oos:  11.822311610672777
IR sm_is:  43.929621234522926
IR sm_oos:  4.374620503352312
{'banks': 0.08166488979222801, 'oil': 0.021013901560859975, 'insurance':
0.03401438499587243, 'tech': 0.07556887845240953, 'bio': 0.1018213957840598, 'pharma':
0.4733388205338499, 'auto': 0.06013865550315474, 'retail': 0.11102671202606172,
'manufacturing': 0.0414123613515039}
IR hier_is:  10.472858271545643
IR hier_oos:  10.073861982124532
IR equal_is:  5.99842921746639
IR equal_oos:  7.3294468896598834
MC run:  5
IR gt_is:  10.103824220304778
IR gt_oos:  11.421185324211141
IR sm_is:  34.55518539676023
IR sm_oos:  3.1879457577902666
{'banks': 0.08970815066948956, 'oil': 0.1307340776803956, 'insurance':
0.12469407189883207, 'tech': 0.24073645540833558, 'bio': 0.056126806518994134,
'pharma': 0.2154506599583787, 'auto': 0.04356796923998072, 'retail':
0.03811312527453183, 'manufacturing': 0.06086868335106177}
IR hier_is:  10.960718037258717
IR hier_oos:  9.97486228127692
IR equal_is:  8.092429722249209
IR equal_oos:  8.679285237881217
MC run:  6
```

```
IR gt_is:  10.29488944780034
IR gt_oos:  12.17669296205728
IR sm_is:  23.096866138190084
IR sm_oos:  3.5244099446923105
{'banks': 0.04458821663588455, 'oil': 0.1452822330941247, 'insurance':
0.054715831023717125, 'tech': 0.34155364901948704, 'bio': 0.09827721046719719,
'pharma': 0.11610843429365972, 'auto': 0.09071148619784236, 'retail':
0.04016316521994309, 'manufacturing': 0.06859977404814413}
IR hier_is:  10.497003057044049
IR hier_oos:  9.692562797952181
IR equal_is:  5.647853461730689
IR equal_oos:  7.372422037737031
MC run:  7
IR gt_is:  11.35421453018477
IR gt_oos:  11.050013757617139
IR sm_is:  34.609865179971735
IR sm_oos:  3.2509384370362224
{'banks': 0.05269538627612803, 'oil': 0.042157948864162915, 'insurance':
0.10730678314860663, 'tech': 0.1836807118111258, 'bio': 0.15576599512650935, 'pharma':
0.22809336008536024, 'auto': 0.07670383721407253, 'retail': 0.06291820784434829,
'manufacturing': 0.09067776962968625}
IR hier_is:  12.545979633101311
IR hier_oos:  9.466487040454792
IR equal_is:  6.949506583531387
IR equal_oos:  6.114972061598138
MC run:  8
IR gt_is:  10.353999316140763
IR gt_oos:  10.61824714848229
IR sm_is:  37.48072170328032
IR sm_oos:  2.252356765823739
{'banks': 0.07809976100716372, 'oil': 0.05241842869191247, 'insurance':
0.12488063623068338, 'tech': 0.08794854357363897, 'bio': 0.04402867182081638,
'pharma': 0.43712944433692213, 'auto': 0.06101065552288361, 'retail':
0.07928380223493628, 'manufacturing': 0.0352000565810431}
IR hier_is:  10.767989311510496
IR hier_oos:  8.897603816122166
IR equal_is:  5.409423410205528
IR equal_oos:  5.950879525394032
MC run:  9
IR gt_is:  12.14637426022587
IR gt_oos:  10.761432374181734
IR sm_is:  36.26266064003784
IR sm_oos:  2.8696040212229548
{'banks': 0.04234865653226834, 'oil': 0.045858654229409344, 'insurance':
0.05138479316066373, 'tech': 0.40415831548117837, 'bio': 0.01833294734327381,
'pharma': 0.07967726146882839, 'auto': 0.08538397693076907, 'retail':
0.1305072636491138, 'manufacturing': 0.1423481312044952}
IR hier_is:  11.846574846619799
IR hier_oos:  8.819680650273495
IR equal_is:  7.919171428328404
IR equal_oos:  6.9260644655664425
```

In [29]:
```python
data_a = [ir_sm_is, ir_hier_is, ir_gt_is, ir_equal_is]
data_b = [ir_sm_oos, ir_hier_oos, ir_gt_oos, ir_equal_oos]

ticks = ['Markowitz', 'Hierarchical', 'Ground Truth', 'Equal Weighted']

def set_box_color(bp, color):
    plt.setp(bp['boxes'], color=color)
```

```
        plt.setp(bp['whiskers'], color=color)
        plt.setp(bp['caps'], color=color)
        plt.setp(bp['medians'], color=color)

    plt.figure()

    bp1 = plt.boxplot(data_a, positions=np.array(range(len(data_a)))*2.0-0.4, sym='',
    widths=0.6)
    bp2 = plt.boxplot(data_b, positions=np.array(range(len(data_b)))*2.0+0.4, sym='',
    widths=0.6)

    set_box_color(bp1, '#D7191C') # colors are from http://colorbrewer2.org/
    set_box_color(bp2, '#2C7BB6')

    # draw temporary red and blue lines and use them to create a legend
    plt.plot([], c='#D7191C', label='IS')
    plt.plot([], c='#2C7BB6', label='OOS')
    plt.legend()
    plt.title('Markowitz vs. Hierarchical ')
    plt.ylabel('Information Ratio')
    plt.xticks(range(0, len(ticks) * 2, 2), ticks)
    plt.xlim(-2, len(ticks)*2)
    # plt.ylim(0, 8)
    plt.tight_layout()
    # plt.savefig('boxcompare.png')
```




```
In [30]: performance_factor = np.mean(ir_hier_oos)/np.mean(ir_sm_oos)
         print('OOS Hierarchical / Markowitz: ', performance_factor)
```

```
OOS Hierarchical / Markowitz:  2.576837438045077
```

```
In [31]: performance_factor = np.mean(ir_equal_oos)/np.mean(ir_sm_oos)
         print('OOS Equal / Markowitz: ', performance_factor)
```

```
OOS Equal / Markowitz:  1.8603154407711007
```

```
In [32]: performance_factor = np.mean(ir_hier_oos)/np.mean(ir_equal_oos)
         print('OOS Hierarchical / Equal: ', performance_factor)
```

```
OOS Hierarchical / Equal:  1.38516156000779
```

Under fat-tails Markowitz performs even worse since extreme positions expose the portfolio to asset specific tail risk. In particular, Markowitz underperforms equal weighting significantly. The hierarchical method still outperforms equal weighting, though.

```
In [33]: trainig_pct = 0.5
         n_train = int(trainig_pct*num_samples)
         ir_sm_is = []
         ir_sm_oos = []
         ir_pca_is = []
         ir_pca_oos = []
         ir_gt_is = []
         ir_gt_oos = []
         n_stocks = 50

         garch_mu = 0
         garch_alpha = 0.5
         garch_beta = 0.3

         for j in range(MC_RUNS):
             print('MC run: ', j)

             market = norm.rvs(size=(1, num_samples))[0]*sigma['market']
             industries = {}
             industries['banks'] =  norm.rvs(size=(1, num_samples))[0]*sigma['banks']
             industries['oil'] = norm.rvs(size=(1, num_samples))[0]*sigma['oil']
             industries['insurance'] = norm.rvs(size=(1, num_samples))[0]*sigma['insurance']
             industries['tech'] = norm.rvs(size=(1, num_samples))[0]*sigma['tech']
             industries['bio'] = norm.rvs(size=(1, num_samples))[0]*sigma['bio']
             industries['pharma'] = norm.rvs(size=(1, num_samples))[0]*sigma['pharma']
             industries['auto'] = norm.rvs(size=(1, num_samples))[0]*sigma['auto']
             industries['retail'] = norm.rvs(size=(1, num_samples))[0]*sigma['retail']
             industries['manufacturing'] = norm.rvs(size=(1,
         num_samples))[0]*sigma['manufacturing']

             industries_stocks = {}
             industries_mu = {}
             industries_beta_market = {}
             industries_beta_industries = {}
             industries_weights = {}
             industries_portfolio = {}

             stocks_all = pd.DataFrame()
             expected_returns_all = pd.Series()

             B = np.zeros((n_stocks*len(industries.keys()), len(sigma)))
             for n, i in enumerate(industries.keys()):
                 industries_stocks[i],\
                 industries_mu[i],\
                 industries_beta_market[i],\
                 industries_beta_industries[i] = gen_industry_stocks_garch(n_stocks,
                                                                           market,
                                                                           industries[i],
                                                                           i,
                                                                           sigma_noise,
                                                                           p_year,
                                                                           garch_mu,
                                                                           garch_alpha,
                                                                           garch_beta
                                                                           )
                 B[n_stocks*n:n_stocks*(n+1), 0] = np.array(list(industries_beta_market[i]))
                 B[n_stocks*n:n_stocks*(n+1), n+1] =
         np.array(list(industries_beta_industries[i]))

                 stocks_all = pd.concat([stocks_all,industries_stocks[i]], axis=1)
                 expected_returns_all = expected_returns_all.append(industries_mu[i])

             if PLOT_STOCKS:
                 plot_equity_curve(stocks_all)

             # ground truth
             Sigma_f = np.diag([sigma[i]**2 for i in sigma.keys()])
             # GARCH uncond Var
```

```python
            Sigma_e = np.diag([(sigma_noise**2)/(1-garch_alpha-garch_beta)]*B.shape[0])
            cov_truth = pd.DataFrame(B.dot(Sigma_f).dot(np.transpose(B))) + Sigma_e
            weights = optimize.minimize_objective(expected_returns_all.index,
                                                  optimize.negative_sharpe,
                                                  True,
                                                  (-1, 1),
                                                  expected_returns_all, cov_truth,
                                                  0.0, 0.0,)

            portfolio_gt_is = np.dot(stocks_all.values,
                                     np.array(list(weights.values())))[:n_train]
            IR_ann = information_ratio(portfolio_gt_is)
            ir_gt_is.append(IR_ann)
            print('IR gt_is', IR_ann)

            portfolio_gt_oos = np.dot(stocks_all.values,
                                      np.array(list(weights.values())))[n_train:]
            IR_ann = information_ratio(portfolio_gt_oos)
            ir_gt_oos.append(IR_ann)
            print('IR gt_oos', IR_ann)

            # standard sample moments
            cov = stocks_all[:n_train].cov()
            weights = optimize.minimize_objective(expected_returns_all.index,
                                                  optimize.negative_sharpe,
                                                  True,
                                                  (-1, 1),
                                                  expected_returns_all, cov,
                                                  0.0, 0.0,)

            portfolio_sm_is = np.dot(stocks_all.values,
                                     np.array(list(weights.values())))[:n_train]
            IR_ann = information_ratio(portfolio_sm_is)
            ir_sm_is.append(IR_ann)
            print('IR sm_is: ', IR_ann)

            portfolio_sm_oos = np.dot(stocks_all.values,
                                      np.array(list(weights.values())))[n_train:]
            IR_ann = information_ratio(portfolio_sm_oos)
            ir_sm_oos.append(IR_ann)
            print('IR sm_oos: ', IR_ann)

            # PCA
            n = len(industries)+1
            pc_cov = optimize.pc_cov(cov, n)

            weights = optimize.minimize_objective(expected_returns_all.index,
                                                   optimize.negative_sharpe,
                                                   True,
                                                   (-1, 1),
                                                   expected_returns_all, pc_cov,
                                                   0.0, 0.0,)

            portfolio_pca_is = np.dot(stocks_all.values,
                                      np.array(list(weights.values())))[:n_train]
            IR_ann = information_ratio(portfolio_pca_is)
            print('IR pca_is: ', IR_ann)
            ir_pca_is.append(IR_ann)
            portfolio_pca_oos = np.dot(stocks_all.values,
                                       np.array(list(weights.values())))[n_train:]
            IR_ann = information_ratio(portfolio_pca_oos)
            print('IR pca_oos: ', IR_ann)
            ir_pca_oos.append(IR_ann)
```

```
MC run:  0
IR gt_is 10.261467730615779
IR gt_oos 12.205307170065185
```

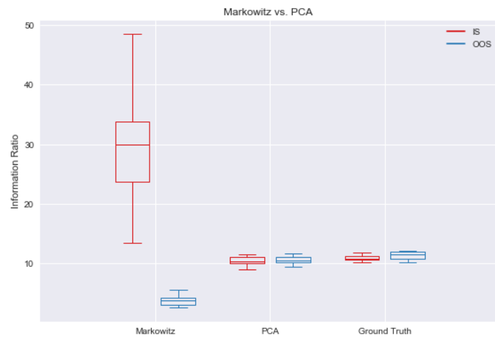
```
/Users/jan/Documents/PoCon/src/optimize.py:30: UserWarning: Optimizer did not
converge.
  warnings.warn("Optimizer did not converge.")


IR sm_is:  31.20512771358549
IR sm_oos:  3.8123236510782195
IR pca_is:  10.01501451645455
IR pca_oos:  11.31193298615081
MC run:  1
IR gt_is 10.834829073114502
IR gt_oos 10.199363654522566
IR sm_is:  48.5033767861998
IR sm_oos:  2.587504099906182
IR pca_is:  11.221432505644566
IR pca_oos:  9.415137484914569
MC run:  2
IR gt_is 11.350257499414049
IR gt_oos 11.37098932637587
IR sm_is:  34.63852875817028
IR sm_oos:  4.376452464663717
IR pca_is:  11.529198106561122
IR pca_oos:  10.268443128966922
MC run:  3
IR gt_is 10.656830170999497
IR gt_oos 10.167979939837414
IR sm_is:  28.915355118058606
IR sm_oos:  3.8330860405686034
IR pca_is:  10.166715341968379
IR pca_oos:  9.737725994879051
MC run:  4
IR gt_is 11.346637358483017
IR gt_oos 11.631193482210865
IR sm_is:  13.434900789513023
IR sm_oos:  2.824394471191307
IR pca_is:  11.2701557116966
IR pca_oos:  11.051427591477806
MC run:  5
IR gt_is 10.972648077846118
IR gt_oos 12.07679547454538
IR sm_is:  45.755145658179934
IR sm_oos:  3.663931866736201
IR pca_is:  10.436667076113618
IR pca_oos:  11.144498371271428
MC run:  6
IR gt_is 10.68026508950257
IR gt_oos 11.684912698906743
IR sm_is:  21.14543558335082
IR sm_oos:  2.737664908310083
IR pca_is:  10.33723606638786
IR pca_oos:  10.585595086766487
MC run:  7
IR gt_is 10.805406472099559
IR gt_oos 10.74162797359147
IR sm_is:  31.122845181605665
IR sm_oos:  4.150300717750667
IR pca_is:  9.91937434019101
IR pca_oos:  10.415509985218758
MC run:  8
```

```
IR gt_is 8.839536302610709
IR gt_oos 12.194076158622874
IR sm_is:  23.73105172327393
IR sm_oos:  5.663203864980169
IR pca_is:  9.035793751477094
IR pca_oos:  11.744754327636812
MC run:  9
IR gt_is 11.779771362516906
IR gt_oos 11.270879177584115
IR sm_is:  23.736621133458673
IR sm_oos:  4.873290937938001
IR pca_is:  10.523662012346414
IR pca_oos:  10.206854599638687
```

In [34]:
```
performance_factor = np.mean(ir_pca_oos)/np.mean(ir_sm_oos)
print('OOS PCA / Markowitz: ', performance_factor)
```

```
OOS PCA / Markowitz:  2.7485971382068155
```

In [35]:
```
data_a = [ir_sm_is, ir_pca_is, ir_gt_is]
data_b = [ir_sm_oos, ir_pca_oos, ir_gt_oos]

ticks = ['Markowitz', 'PCA', 'Ground Truth']

def set_box_color(bp, color):
    plt.setp(bp['boxes'], color=color)
    plt.setp(bp['whiskers'], color=color)
    plt.setp(bp['caps'], color=color)
    plt.setp(bp['medians'], color=color)

plt.figure()

bp1 = plt.boxplot(data_a, positions=np.array(range(len(data_a)))*2.0-0.4, sym='',
widths=0.6)
bp2 = plt.boxplot(data_b, positions=np.array(range(len(data_b)))*2.0+0.4, sym='',
widths=0.6)

set_box_color(bp1, '#D7191C') # colors are from http://colorbrewer2.org/
set_box_color(bp2, '#2C7BB6')

# draw temporary red and blue lines and use them to create a legend
plt.plot([], c='#D7191C', label='IS')
plt.plot([], c='#2C7BB6', label='OOS')
plt.legend()
plt.title('Markowitz vs. PCA ')
plt.ylabel('Information Ratio')
plt.xticks(range(0, len(ticks) * 2, 2), ticks)
plt.xlim(-2, len(ticks)*2)
# plt.ylim(0, 8)
plt.tight_layout()
# plt.savefig('boxcompare.png')
```



The PCA method, too, outperforms Markowitz by a large margin under fat-tails.

1.4 Conclusion

Industry neutral equal weighting can go a long way. If the number of assets is large, it tends to outperform mean-variance optimization based on sample moments. It certainly is better than long-only portfolio construction. If one is willing to invest the effort to go beyond naive approaches, imposing structure by hierarchical methods, shrinkage estimation or noise reduction via PCA seem to be good approaches based on monte carlo evidence. In addition, I showed that these methods are robust to asset-specific fat-tailed noise by having fewer extreme and instead more numerous moderate position weights.

```
In [ ]:
```