demo

August 19, 2019

1 Portfolio Optimization

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 $\operatorname{Cov}(f_{\ell t}, \epsilon_{it}) = 0, \quad \ell = 1, \ldots, k, \quad i = 1, \ldots, N, \quad t = 1, \ldots, T,$

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

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

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

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 $R_t = [R_{1t}, R_{2t}, \ldots, R_{Nt}]$ implied by the factor model is

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

That is,

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

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

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

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

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


        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


        p_year = 250
        num_samples = p_year*4
```

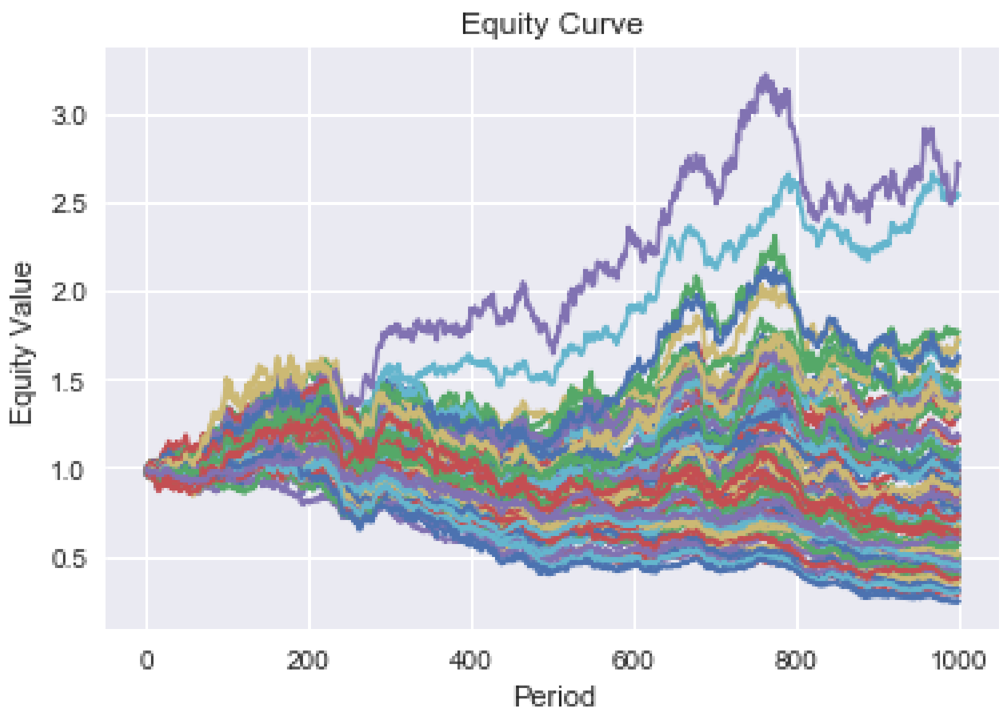
```
sigma = {}
sigma['market'] = 0.1/p_year**0.5
sigma['banks'] = 0.1/p_year**0.5
sigma['oil'] = 0.14/p_year**0.5
sigma['insurance'] = 0.07/p_year**0.5
sigma['tech'] = 0.2/p_year**0.5
sigma['bio'] = 0.12/p_year**0.5
sigma['pharma'] = 0.22/p_year**0.5
sigma['auto'] = 0.05/p_year**0.5
sigma['retail'] =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

sigma_noise = 0.1/p_year**0.5
MC_RUNS = 5
PLOT_STOCKS = False
```

In [2]:
```
def plot_equity_curve(log_returns):
    plt.plot(np.exp(log_returns.cumsum()))
    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()

#     print('|| In-Sample Performance || \n')
    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)
```

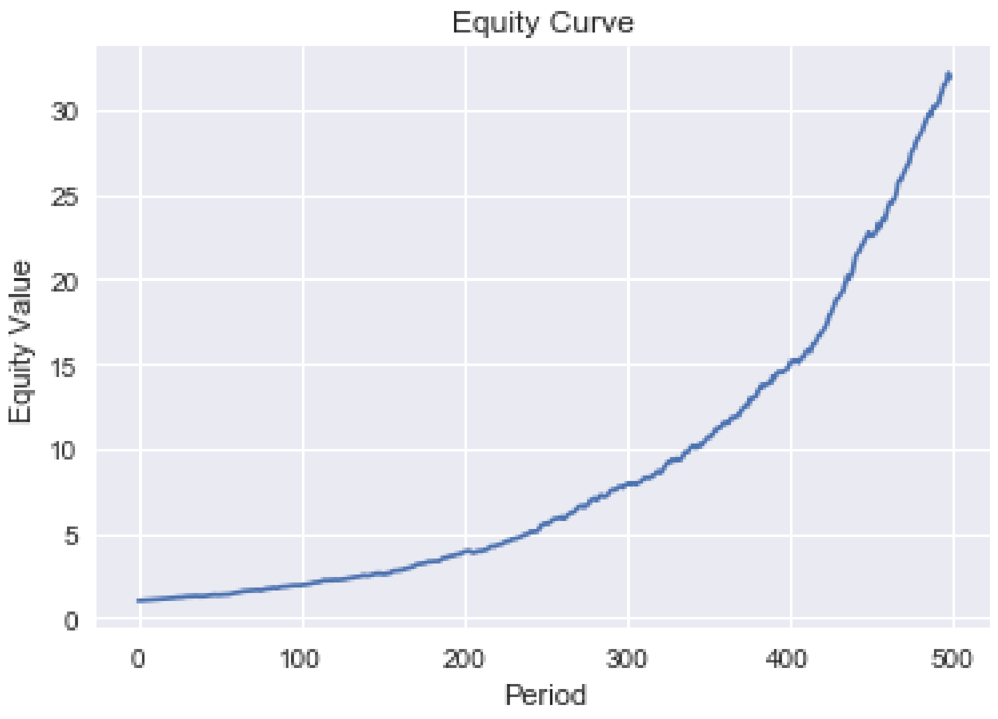
```
#      print('|| Out-of-Sample Performance || \n')
    portfolio_sm_oos = np.dot(stocks.values,
                              np.array(list(weights.values())))[n_train:]
    IR_ann = information_ratio(portfolio_sm_oos)
    print('IR sm_oos: ', IR_ann)
    plot_equity_curve(portfolio_sm_oos)

    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
```

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).

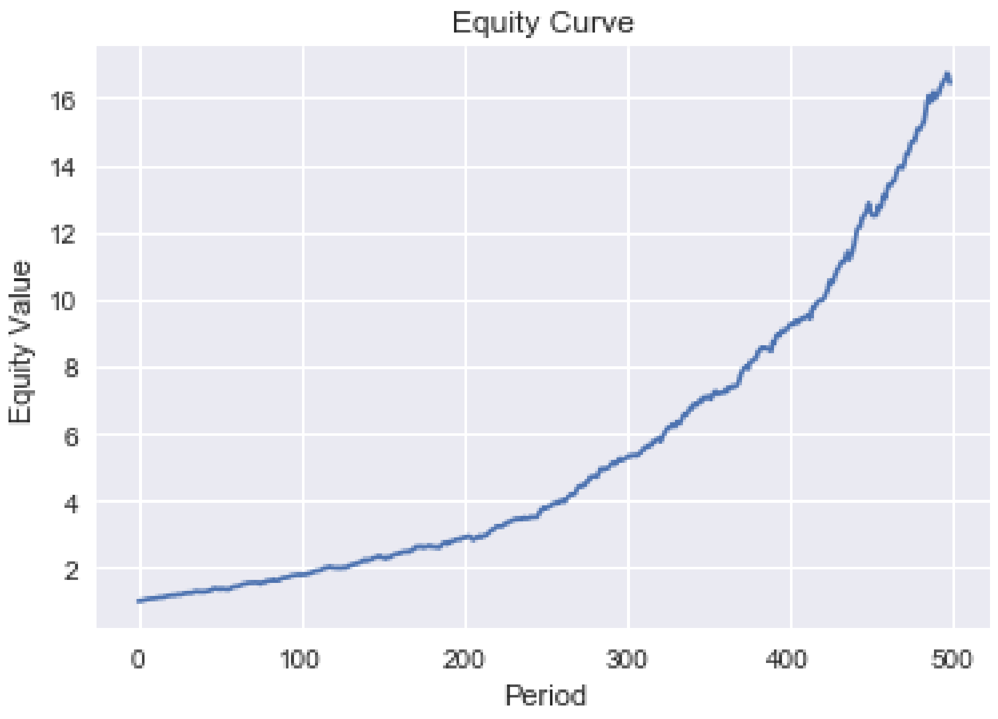
```
In [3]: industry = {}
        industry['manufacturing'] = norm.rvs(size=(1, num_samples))[0]*sigma['manufacturing']
        _ = show_standard_optimized_equity(n_stocks=100, industry=industry)
```



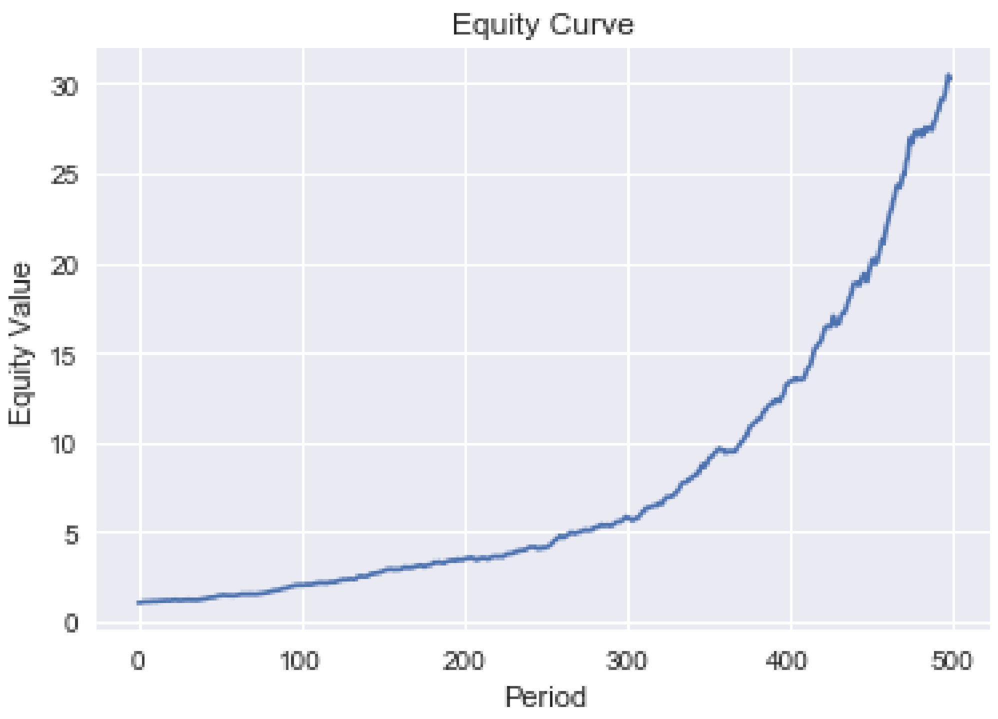
```
IR sm_is:  12.651404211591572
```



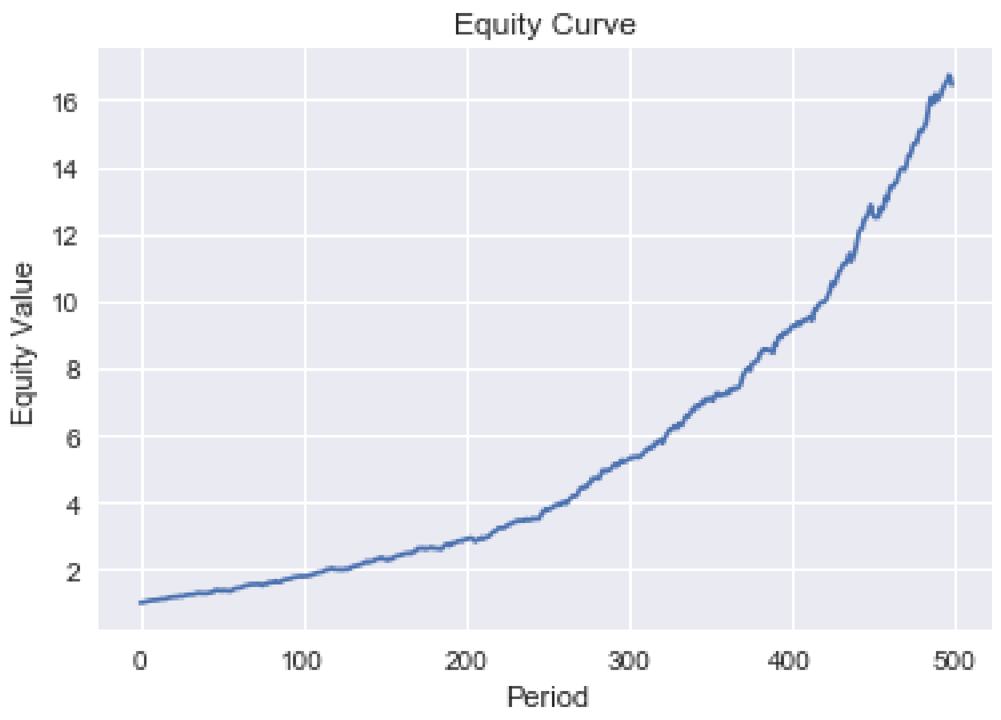
```
IR equal_is:  9.236222264204967
```



```
IR sm_oos:  9.910268820307552
```

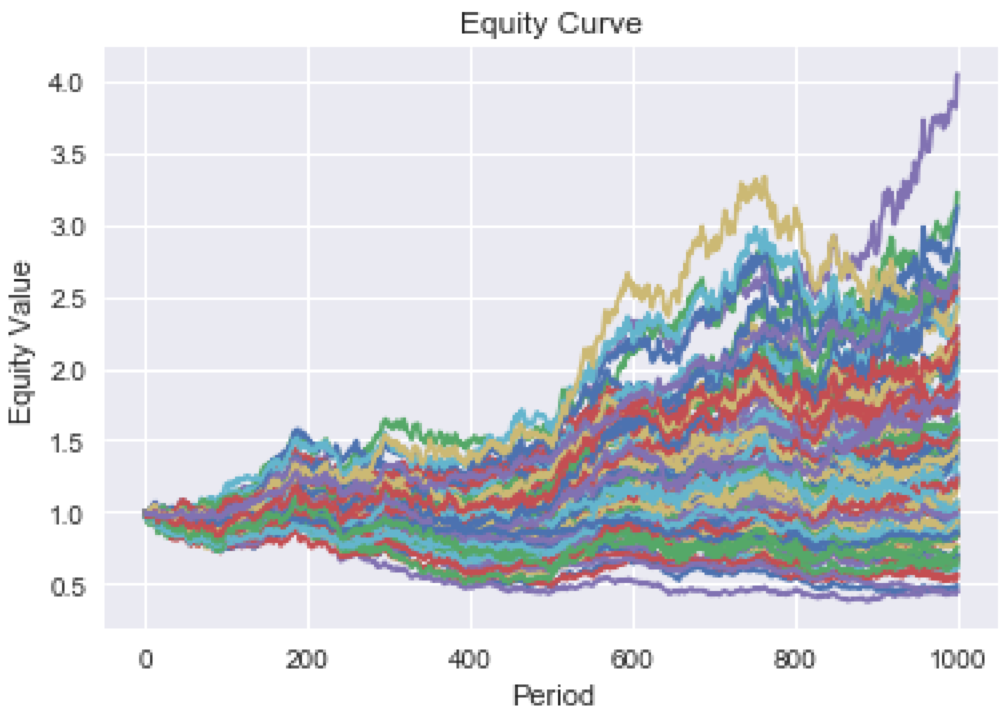


```
IR equal_oos:  9.236222264204967
```

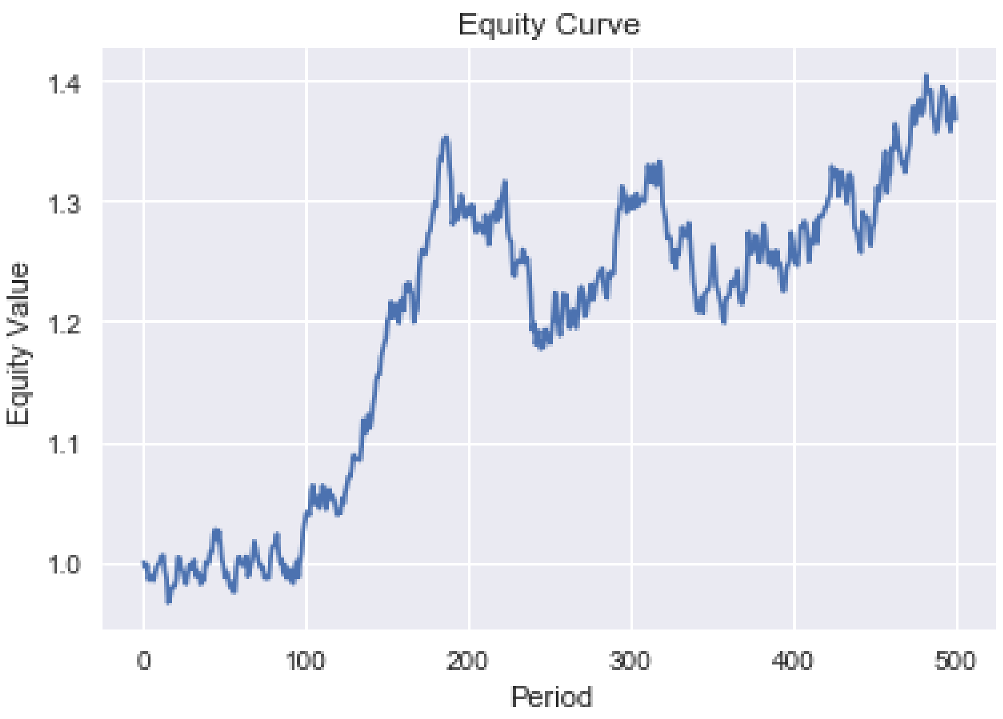


As shown below, long-only performance, of course, is horrendous. This is due to the fact that we cannot diversify the systematic risk. Since we cannot short, hedging out that risk is also not possible. Hence, our bets are not independent and the Law of Large Numbers does not apply.

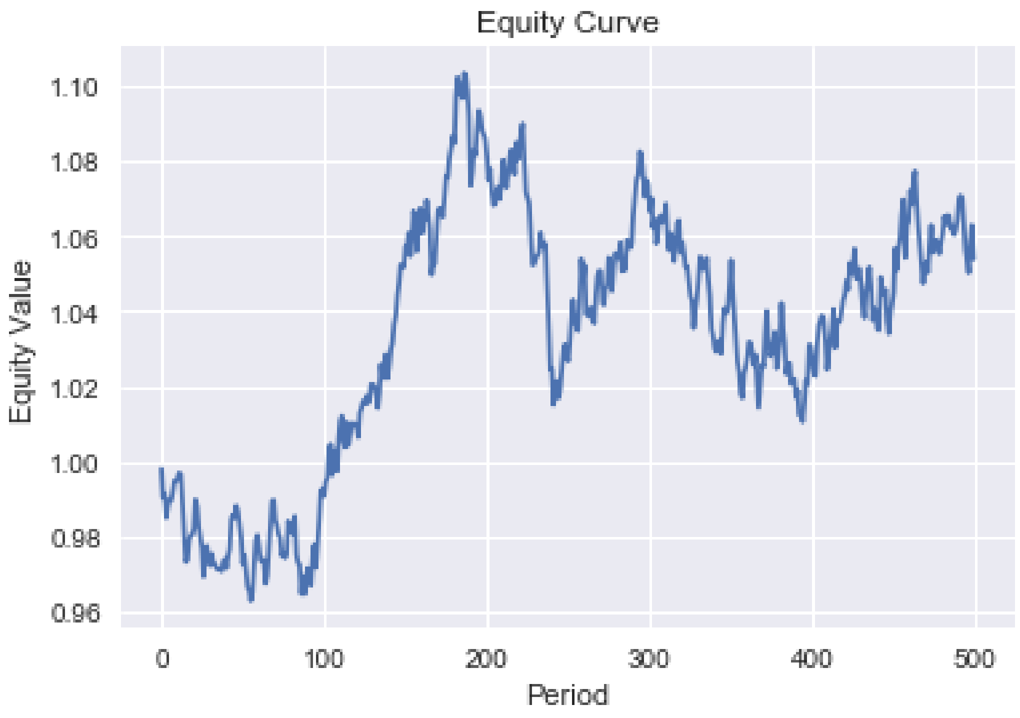
```
In [4]: _ = show_standard_optimized_equity(n_stocks=100, industry=industry, long_only=True)
```



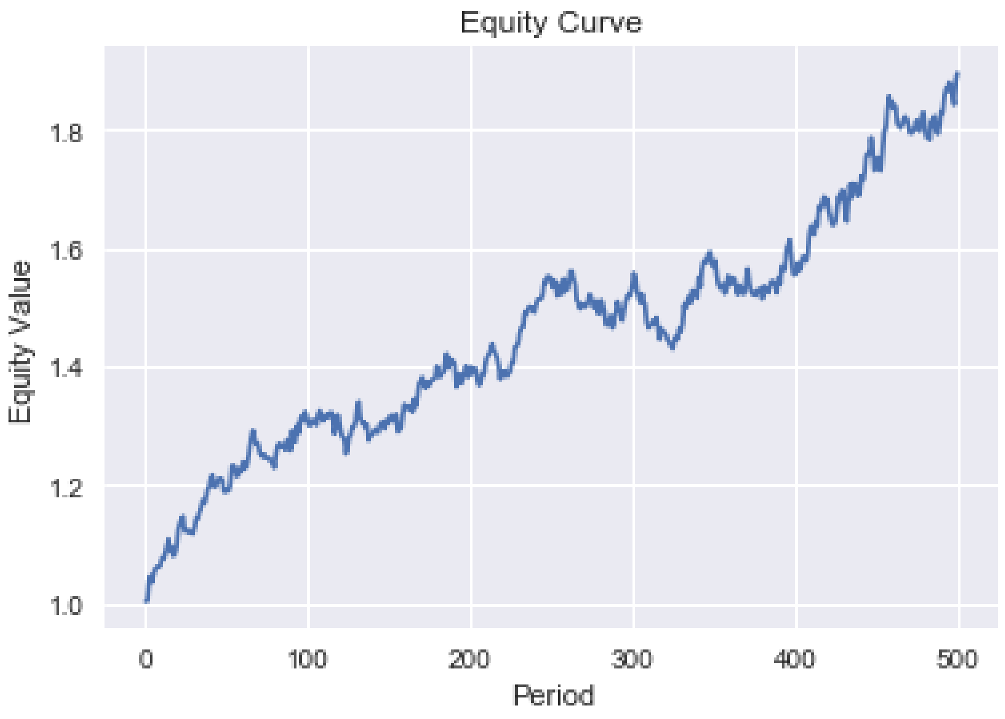
```
IR sm_is:  1.1077591138055525
```



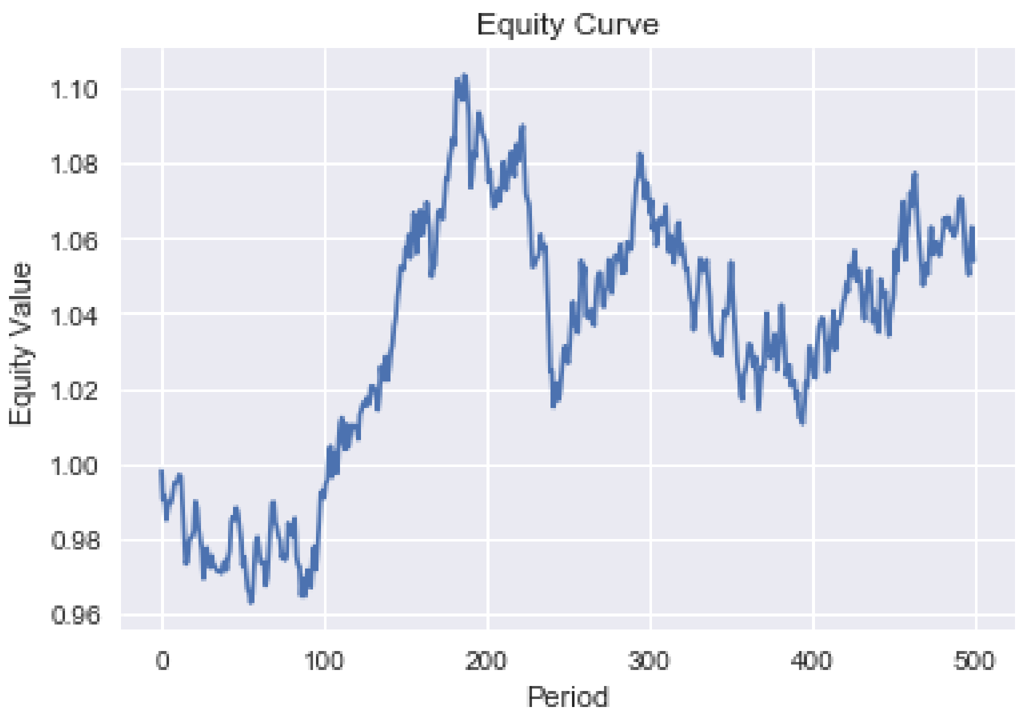
```
IR equal_is:  0.3523053803647488
```



```
IR sm_oos:  2.1694236324044276
```

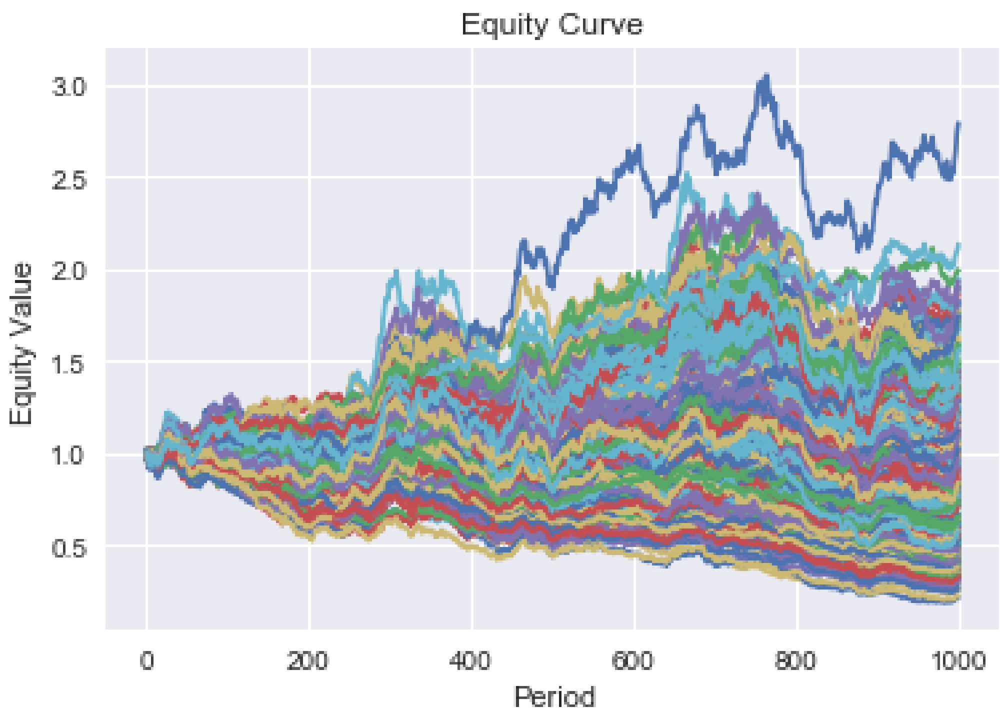


```
IR equal_oos:  0.3523053803647488
```

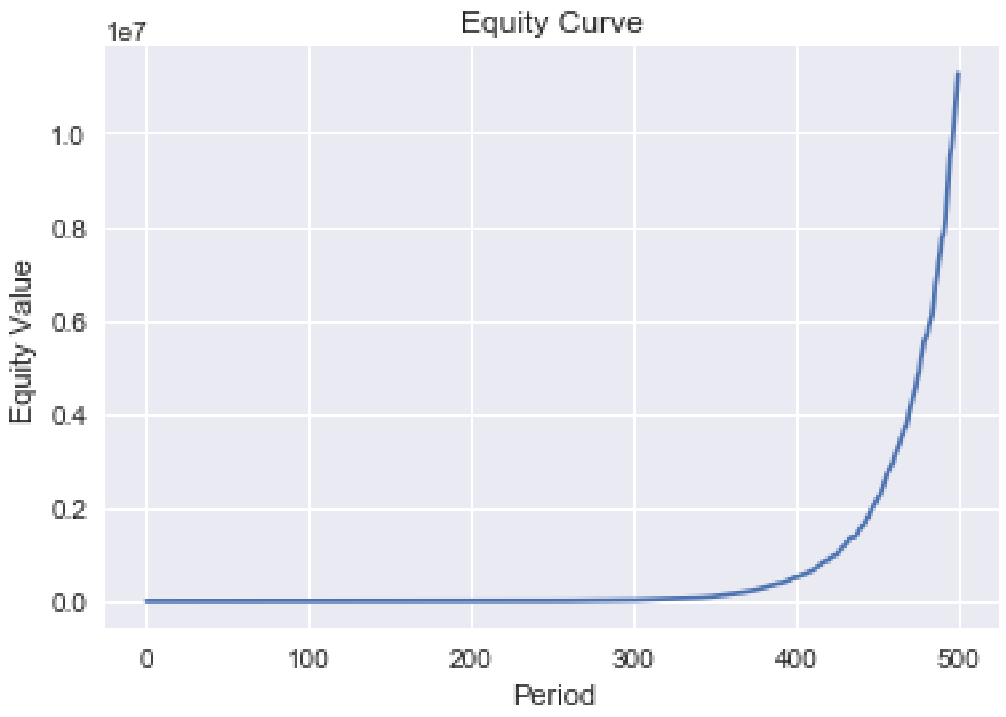


Standard mean-variance optimization works well if the number of assets is relatively small. When the number of assets is relatively large, however, the out-of-sample performance is significantly worse than the in-sample performance as demonstrated by the information ratio (IR). Even the very naive approach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

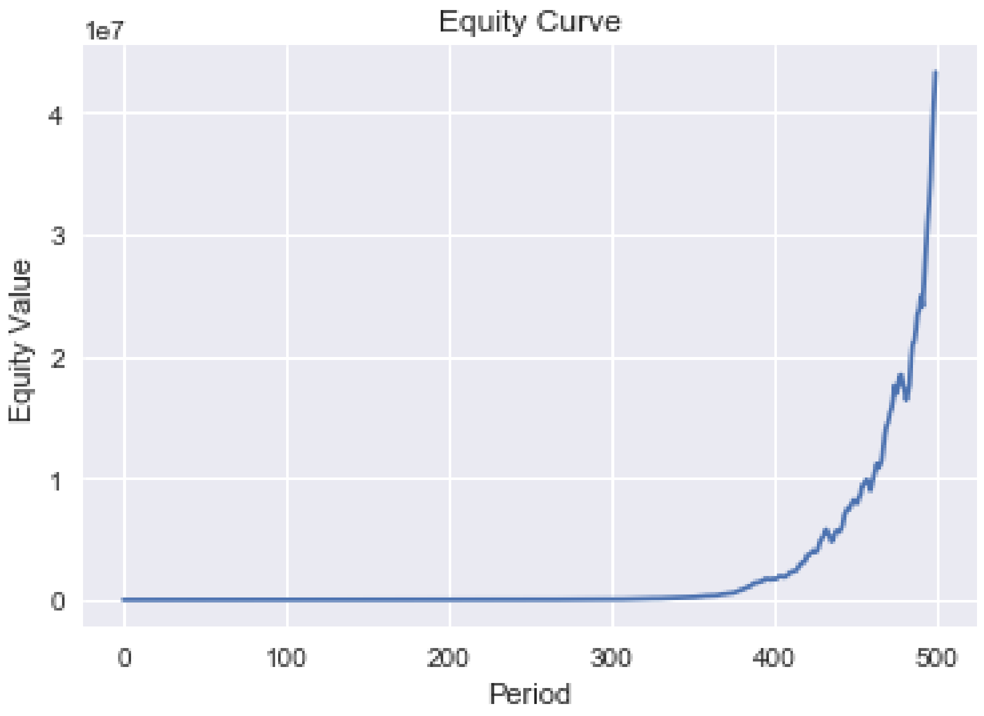
```
In [5]: weights = show_standard_optimized_equity(n_stocks=300, industry=industry)
```



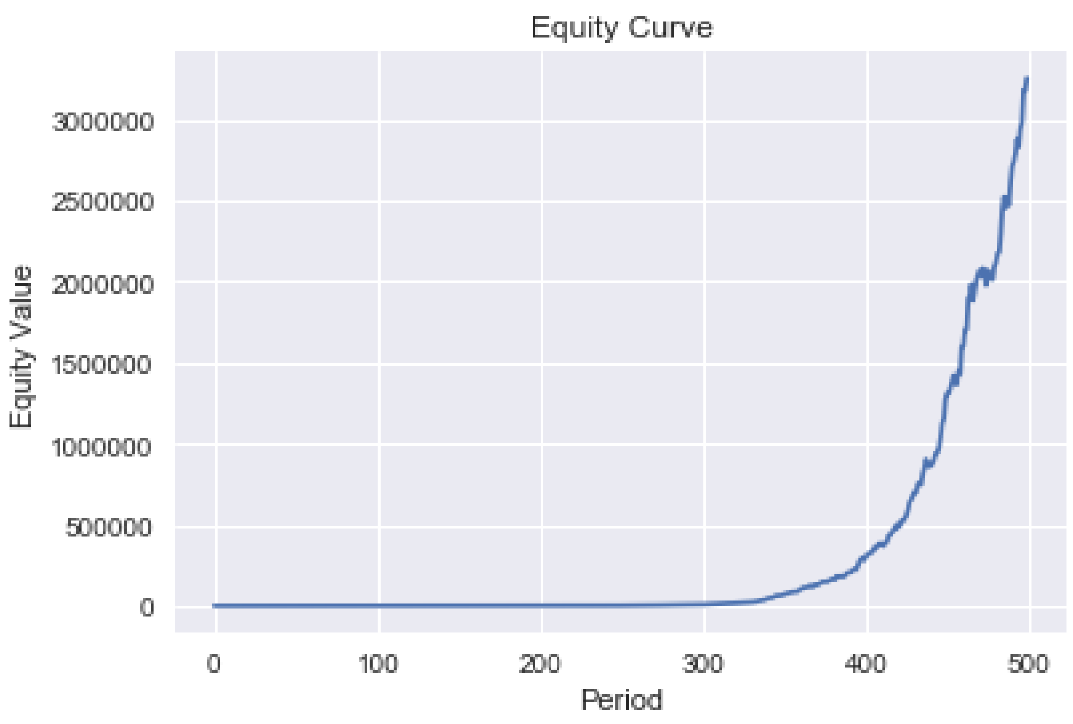
```
IR sm_is:  34.85381862182288
```



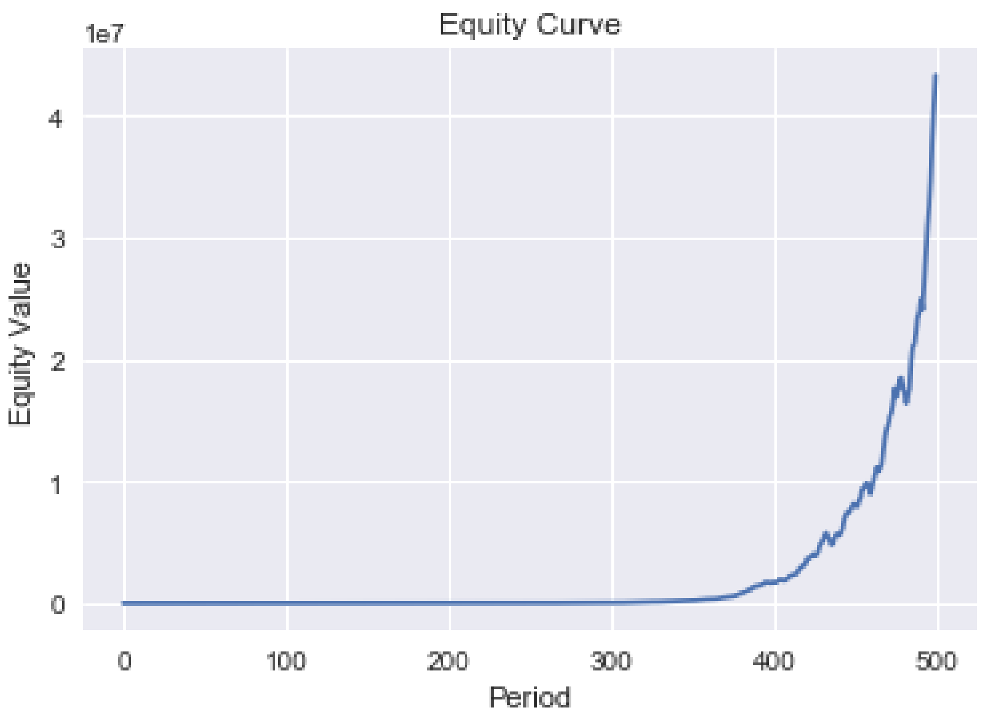
```
IR equal_is:  12.958840166422652
```



```
IR sm_oos:  12.452707833288388
```



```
IR equal_oos:  12.958840166422652
```



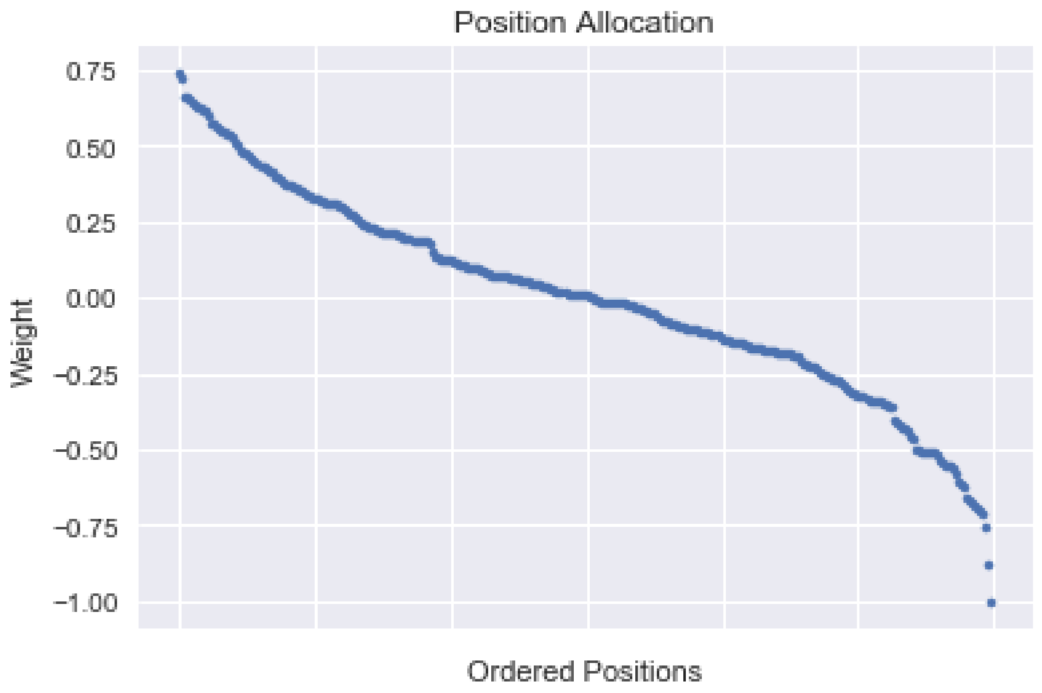
```
In [6]: # creating portfolio object
        pf = Portfolio(assets=weights.keys(),
                       position=pd.Series(weights),
                       price=[1]*len(weights.keys()))
        print(pf)
        print('largest long:', pf[0], pf.position(pf[0]))
        print('largest short:', pf[-1], pf.position(pf[-1]))
```

```
Portfolio: 300 Assets, $38.91 Long, $38.91 Short
largest long: manufacturing_stock_212 0.7374611476838714
largest short: manufacturing_stock_93 -1.0
```

```
In [7]: # Showing the first 5 positions in portfolio
        pf.portfolio_df.head()
```

```
Out[7]:                          position  price  factor_value  sector_id  \
        manufacturing_stock_212  0.737461      1           1.0        0.0
        manufacturing_stock_299  0.721744      1           1.0        0.0
        manufacturing_stock_278  0.658798      1           1.0        0.0
        manufacturing_stock_91   0.657564      1           1.0        0.0
        manufacturing_stock_33   0.651608      1           1.0        0.0

                                 position_value
        manufacturing_stock_212        0.737461
        manufacturing_stock_299        0.721744
        manufacturing_stock_278        0.658798
        manufacturing_stock_91         0.657564
        manufacturing_stock_33         0.651608
```

```
In [8]: # 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

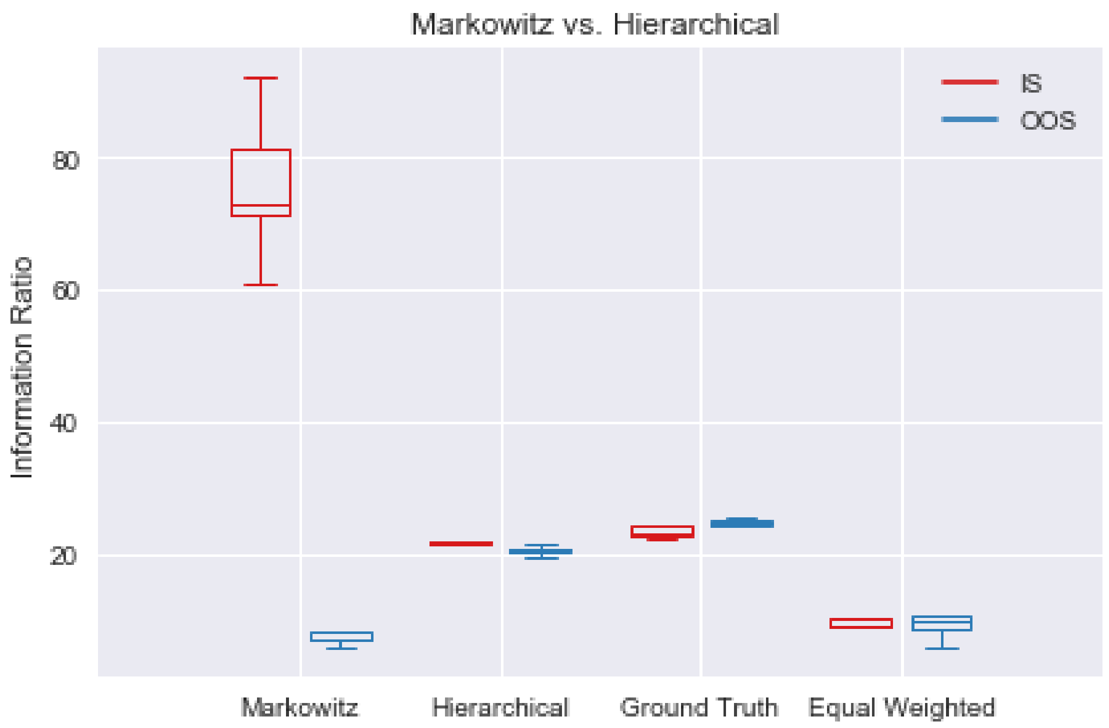
To reduce Error Maximization we need to impose structure. One way to do this is by using our knowledge that stocks usually cluster (e.g., into certain industries). It seems like a good idea to encode this prior knowledge by mean-variance optimize stocks within clusters and then optimize allocation to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

```
In [9]: 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,
                        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:  22.764742896098532
IR gt_oos:  25.423096737395447


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


IR sm_is:  81.48841850109251
IR sm_oos:  8.080572298936241
{'banks': 0.1129615293128751, 'oil': 0.14377718823878444, 'insurance':
0.014058226197895225, 'tech': 0.25574763628860203, 'bio': 0.11006700677931346,
'pharma': 0.2816138500871653, 'auto': 0.033685980441805465, 'retail':
0.031187363281997244, 'manufacturing': 0.016901219371561752}
IR hier_is:  21.75333684206342
IR hier_oos:  21.47607243677229
IR equal_is:  8.934242690940316
IR equal_oos:  10.597192555243918
MC run:  1
IR gt_is:  24.24065757939983
IR gt_oos:  24.084090939842696
IR sm_is:  61.00200926194065
IR sm_oos:  6.886453408141832
{'banks': 0.12417886489545167, 'oil': 0.13952485969362854, 'insurance':
0.06895271343658817, 'tech': 0.31437383592800566, 'bio': 0.05407952882456184,
'pharma': 0.13957806647364773, 'auto': 0.10251699902607415, 'retail':
0.03122666292627042, 'manufacturing': 0.025568468795771876}
IR hier_is:  21.72164768576419
IR hier_oos:  20.585302751567717
IR equal_is:  16.334608097834664
IR equal_oos:  16.14347365872683
MC run:  2
IR gt_is:  22.898593588631904
IR gt_oos:  24.81793868209896
IR sm_is:  92.13530611917427
IR sm_oos:  7.023607721552295
{'banks': 0.012727478982841565, 'oil': 0.1298031410912971, 'insurance':
0.08402321976453021, 'tech': 0.12197948913655515, 'bio': 0.19117486045719584,
'pharma': 0.2449844422880232, 'auto': 0.10318305557951167, 'retail':
0.10458233928488639, 'manufacturing': 0.007541973415158882}
IR hier_is:  22.614677127228017
IR hier_oos:  20.832201639311666
IR equal_is:  10.152899371981803
IR equal_oos:  8.557038995491695
MC run:  3
IR gt_is:  24.226075720624188
IR gt_oos:  25.17534926287912
IR sm_is:  72.71040048334821
IR sm_oos:  10.80162959604957
{'banks': 0.04446499461729821, 'oil': 0.10251008614085398, 'insurance':
0.01228147959832786, 'tech': 0.4463644803362729, 'bio': 0.049745698303080677,
'pharma': 0.2437541747818386, 'auto': 0.020232478546664837, 'retail':
0.06916138316382736, 'manufacturing': 0.011485224511835599}
IR hier_is:  21.273039781724037
IR hier_oos:  20.174332249361626
IR equal_is:  6.897695450146542
IR equal_oos:  5.948558102245953
MC run:  4
IR gt_is:  22.47176220106334
IR gt_oos:  22.076205524532817
IR sm_is:  71.31430082387298
IR sm_oos:  5.77684577884029
{'banks': 0.13882612416254275, 'oil': 0.13751535739762402, 'insurance':
0.047784372997667725, 'tech': 0.3160644145571731, 'bio': 0.03827014948041616,
'pharma': 0.19184647320369733, 'auto': 0.044945218143515354, 'retail':
0.004103569090263107, 'manufacturing': 0.08064432096710052}
IR hier_is:  21.4003668898193
IR hier_oos:  19.618852819668398
IR equal_is:  9.203178199131479
IR equal_oos:  9.636011379780278
```

```
In [10]: 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 [11]: performance_factor = np.mean(ir_equal_oos)/np.mean(ir_sm_oos)
         print('OOS Equal / Markowitz: ', performance_factor)

OOS Equal / Markowitz:  1.3192494270659583
```

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

OOS Hierarchical / Markowitz:  2.662409505487703
```

```
In [13]: performance_factor = np.mean(ir_hier_oos)/np.mean(ir_equal_oos)
         print('OOS Hierarchical / Equal: ', performance_factor)

OOS Hierarchical / Equal:  2.0181244356565418
```

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. It is more appropriately thought of as a industry-matched equal weighting scheme.

```
In [14]: # 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())
```

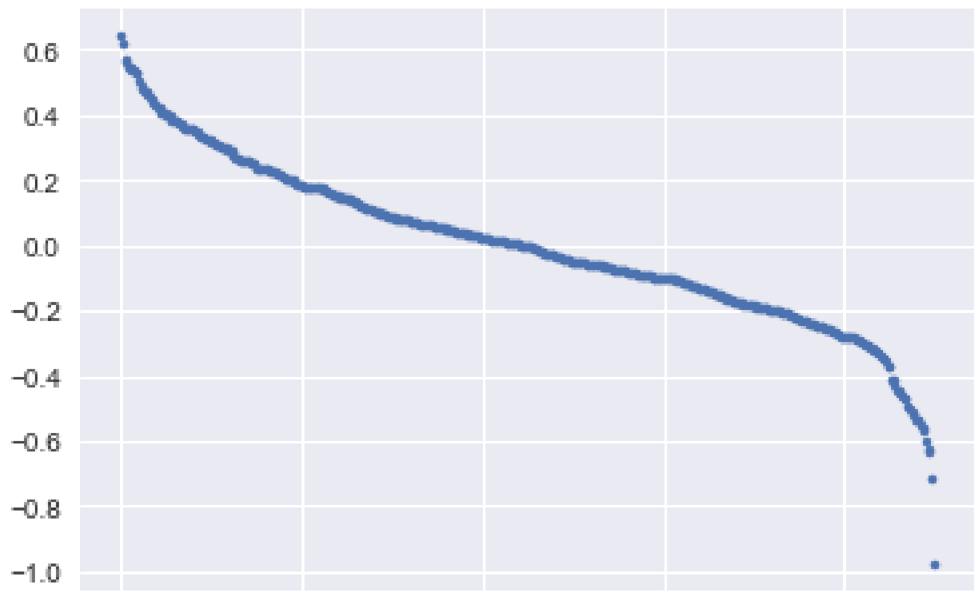
```
Portfolio: 450 Assets, $43.76 Long, $43.76 Short
largest long: oil_stock_13 0.6417725939631443
largest short: auto_stock_20 -0.9752070715541381
sector_net_exposures:
            position_value
sector_id
aut              -0.295522
ban              -0.405407
bio              -0.373189
ins               1.011198
man              -1.089961
oil              -0.484716
pha               0.945934
ret              -0.201969
tec               0.893630
```

```
In [15]: print(pf.portfolio_df.head())
```

```
                    position  price  factor_value sector_id  position_value
oil_stock_13        0.641773      1           1.0       oil        0.641773
insurance_stock_17  0.622193      1           1.0       ins        0.622193
tech_stock_14       0.569854      1           1.0       tec        0.569854
retail_stock_23     0.563937      1           1.0       ret        0.563937
tech_stock_21       0.546087      1           1.0       tec        0.546087
```

```
In [16]: pf.portfolio_df.position_value.plot(style='.')
```

```
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1a176fab38>
```



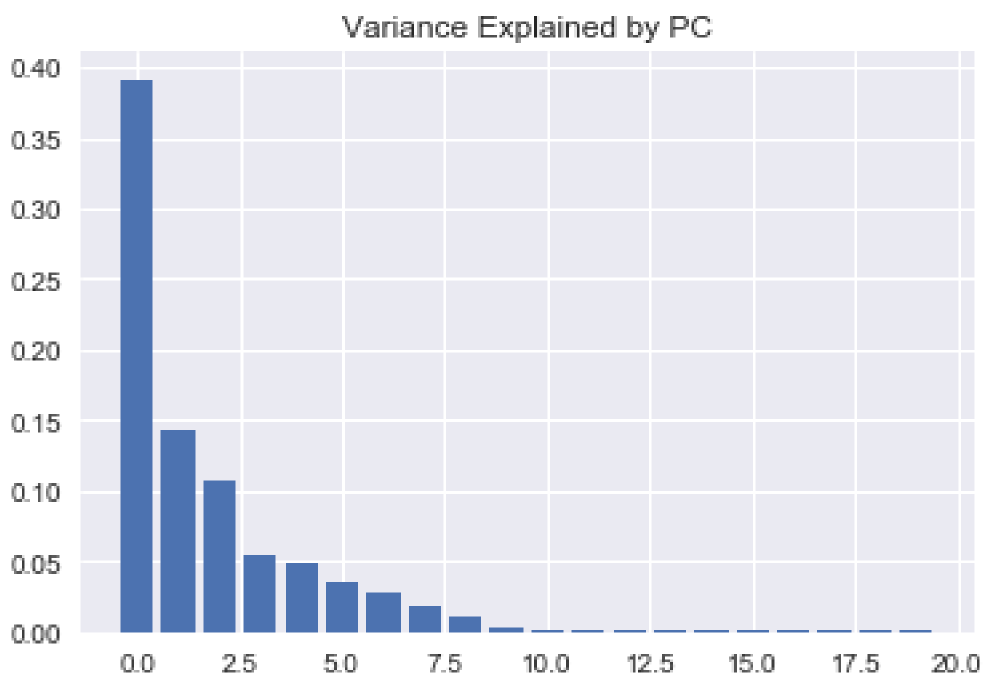
1.2 Principal Component Analysis

Principal Component Analysis (PCA) allows us to reduce the dimension 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.

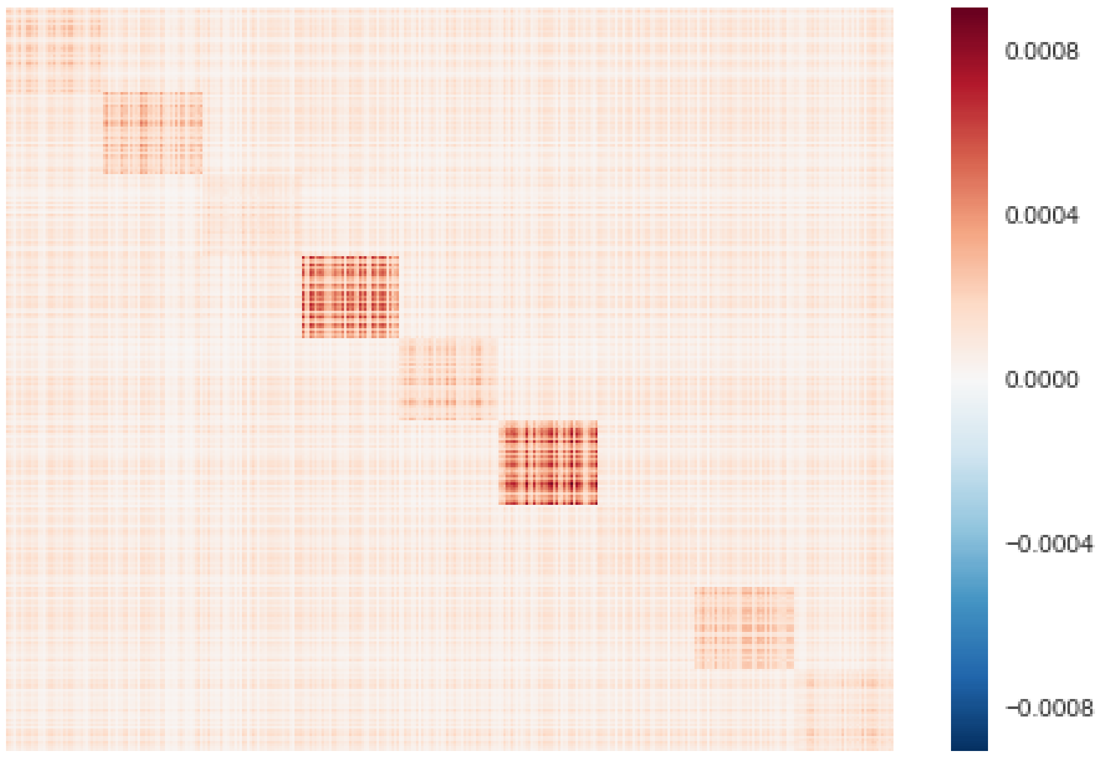
$$S = Q\Lambda Q^{-1} = Q\Lambda Q^{\mathrm{T}} \quad \text{with} \quad Q^{-1} = Q^{\mathrm{T}}$$

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

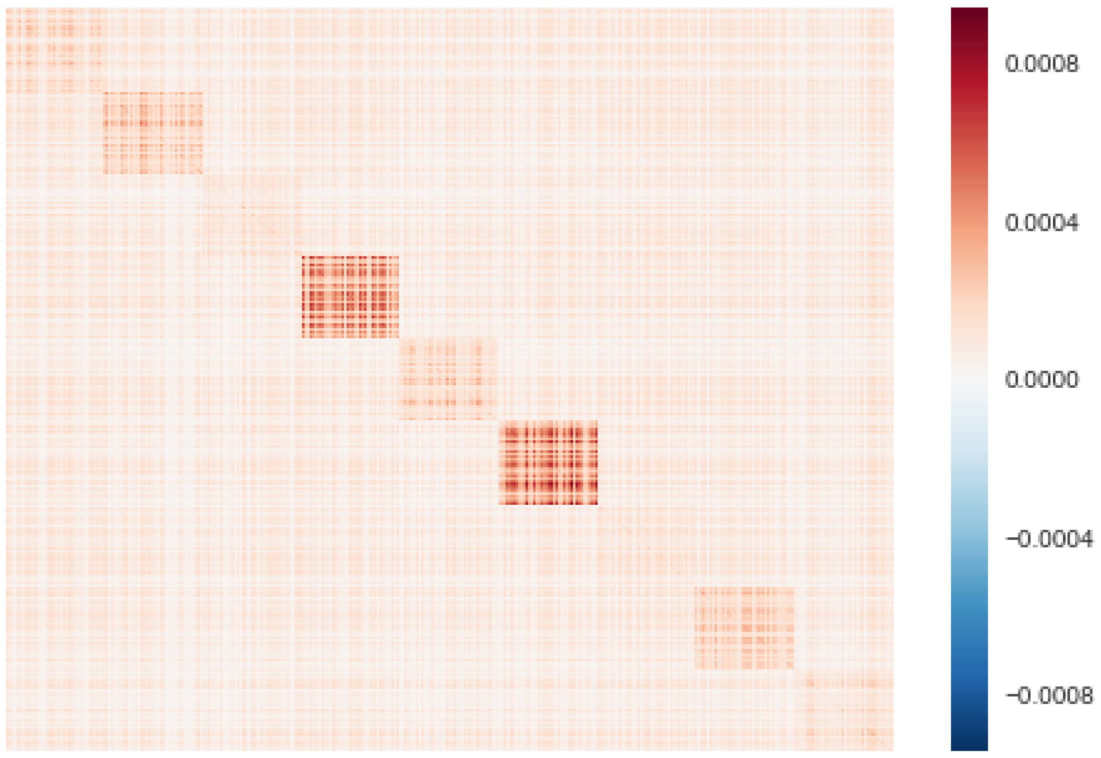
```
In [17]: from sklearn.decomposition import PCA as sklearnPCA

         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()
```



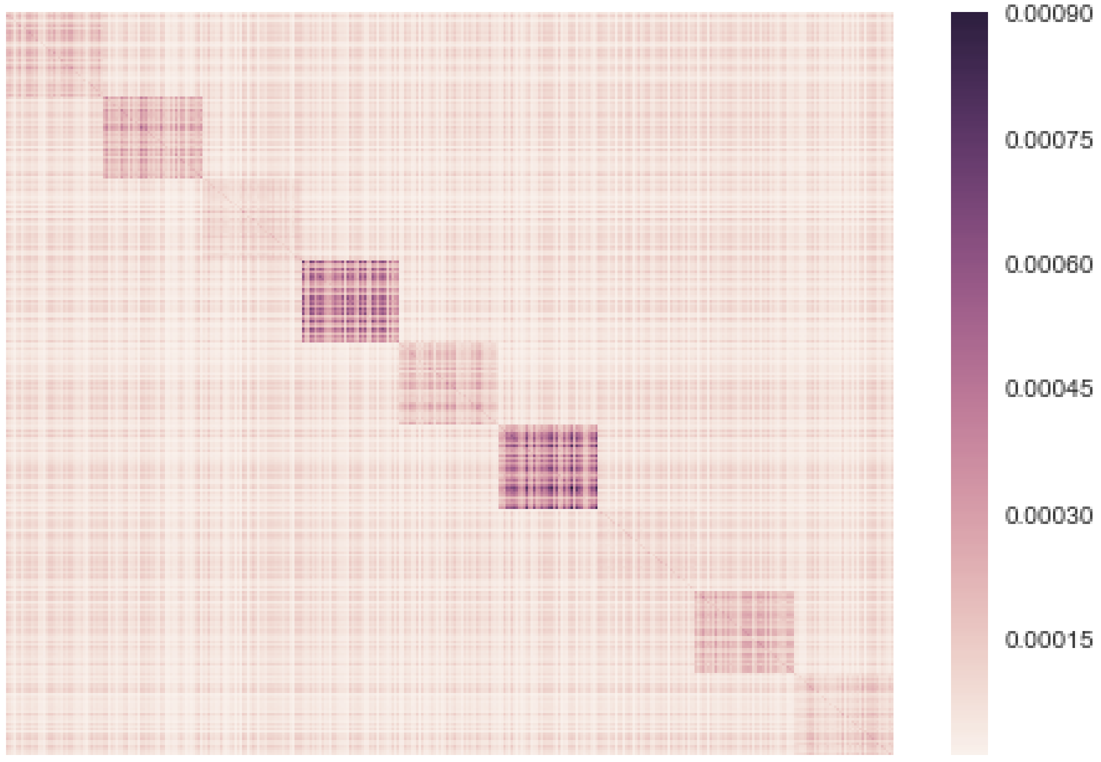
```
In [18]: # denoise covariance matrix
         cov = stocks_all.cov()
         n = len(industries)+1

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

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1a17c6beb8>
```



```
In [20]: # plot sample covariance matrix to compare
         sns.heatmap(cov, xticklabels=False, yticklabels=False)

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a13afd0>
```



```
In [21]: # 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[21]: <matplotlib.axes._subplots.AxesSubplot at 0x181551ee48>
```



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

```
In [22]: 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)
```

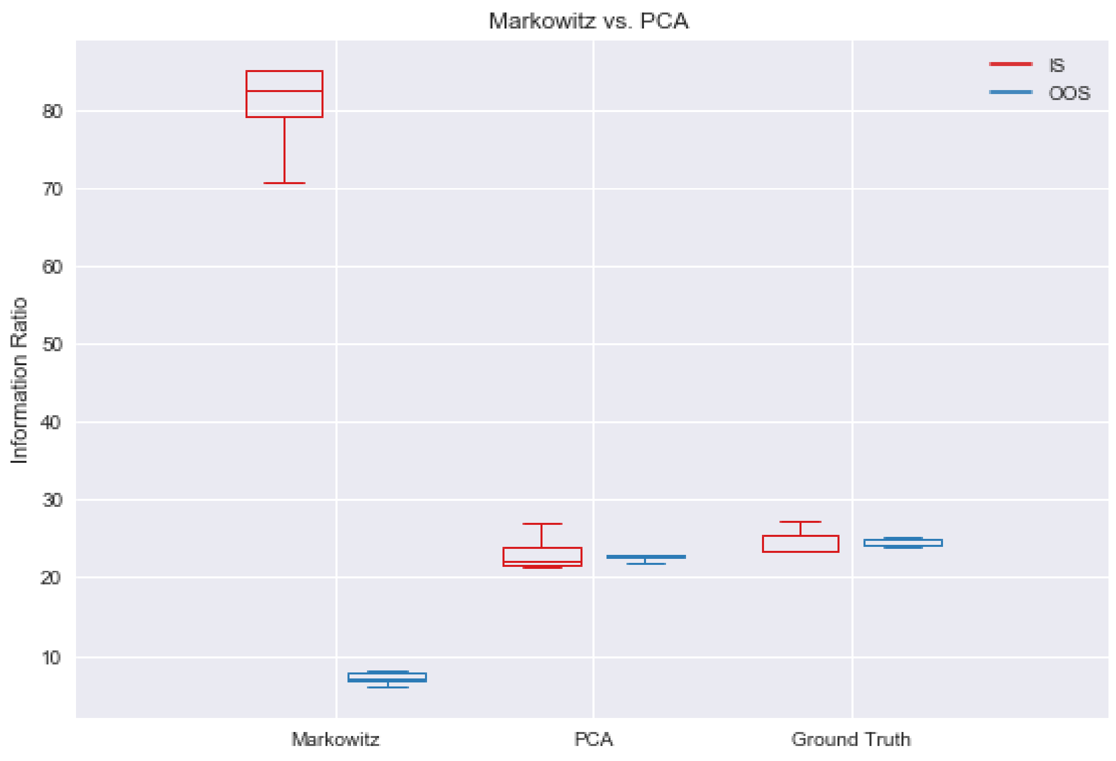
```
            # PCA
            n = len(industries)+1
            pc_cov = optimize.pc_cov(cov, n)
            weights = optimize.minimize_objective(expected_returns_all.index,
                                                  optimize.negative_sharpe,
                                                  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:  23.374207157554075
IR gt_oos:  24.22487984962384
```

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

```
IR sm_is:  82.50759694936612
IR sm_oos:  7.0057612696101375
IR pca_is:  22.216859381000006
IR pca_oos:  22.595827859949612
MC run:  1
IR gt_is:  25.418134789859575
IR gt_oos:  24.240600658640563
IR sm_is:  95.76372494651923
IR sm_oos:  5.958420678047078
IR pca_is:  23.91078700478066
IR pca_oos:  22.990440378515917
MC run:  2
IR gt_is:  23.56364989642909
IR gt_oos:  23.990022007140244
IR sm_is:  79.0767690999176
IR sm_oos:  6.901858131089847
IR pca_is:  21.420330187631972
IR pca_oos:  22.030064199104288
MC run:  3
IR gt_is:  23.3874383590086
IR gt_oos:  25.244127982411683
IR sm_is:  70.68916364298254
IR sm_oos:  7.902184405991386
IR pca_is:  21.75082366746542
IR pca_oos:  24.393423066434213
MC run:  4
IR gt_is:  27.290745199005972
IR gt_oos:  25.02578210149913
IR sm_is:  85.04175309422818
IR sm_oos:  8.154456369030104
IR pca_is:  26.925445403425606
IR pca_oos:  22.825455393641207
```

```
In [23]: 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 [24]: performance_factor = np.mean(ir_pca_oos)/np.mean(ir_sm_oos)
         print('OOS PCA / Markowitz: ', performance_factor)
```

```
OOS PCA / Markowitz:  3.1967327651604864
```

```
In [25]: print(np.linalg.matrix_rank(cov))
         print(np.linalg.matrix_rank(pc_cov))
```

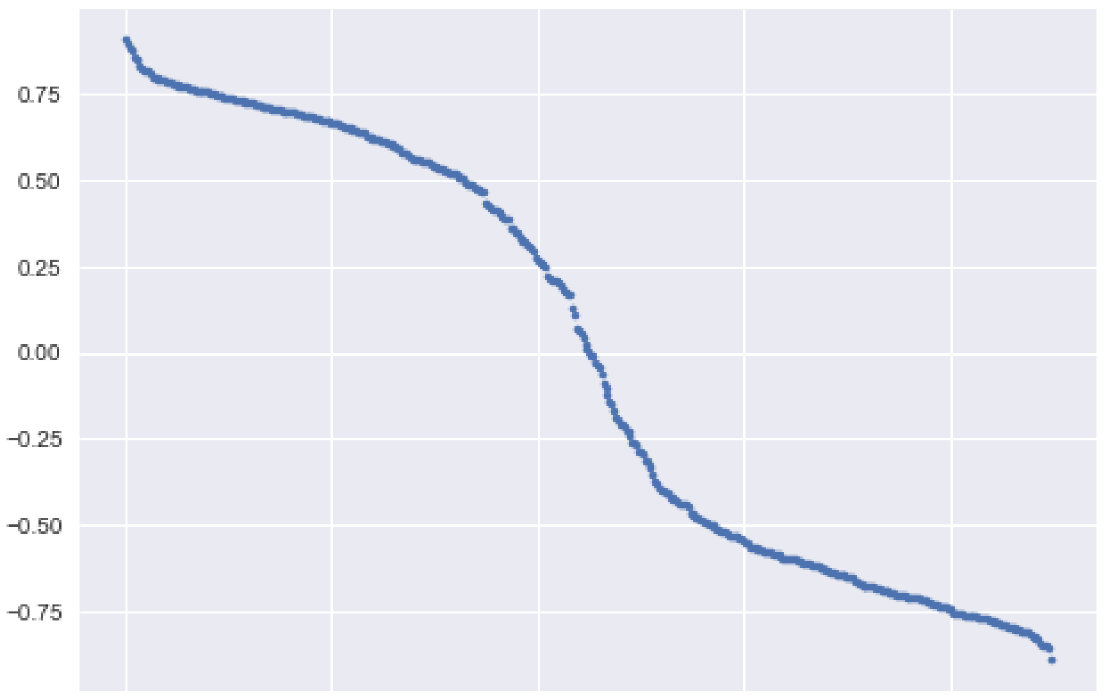
```
450
11
```

```
In [26]: 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, $130.53 Long, $130.53 Short
largest long: manufacturing_stock_40 0.9044735707106419
largest short: insurance_stock_31 -0.8883571288891736
sector_net_exposures:
            position_value
sector_id
aut              -0.595581
ban              -0.018102
bio               2.715188
ins              -1.293434
man               0.030597
oil               2.145395
pha              -2.739619
ret               1.563692
tec              -1.808135
```

```
In [27]: pf.portfolio_df.position_value.plot(style='.')
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a506f28>
```



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:

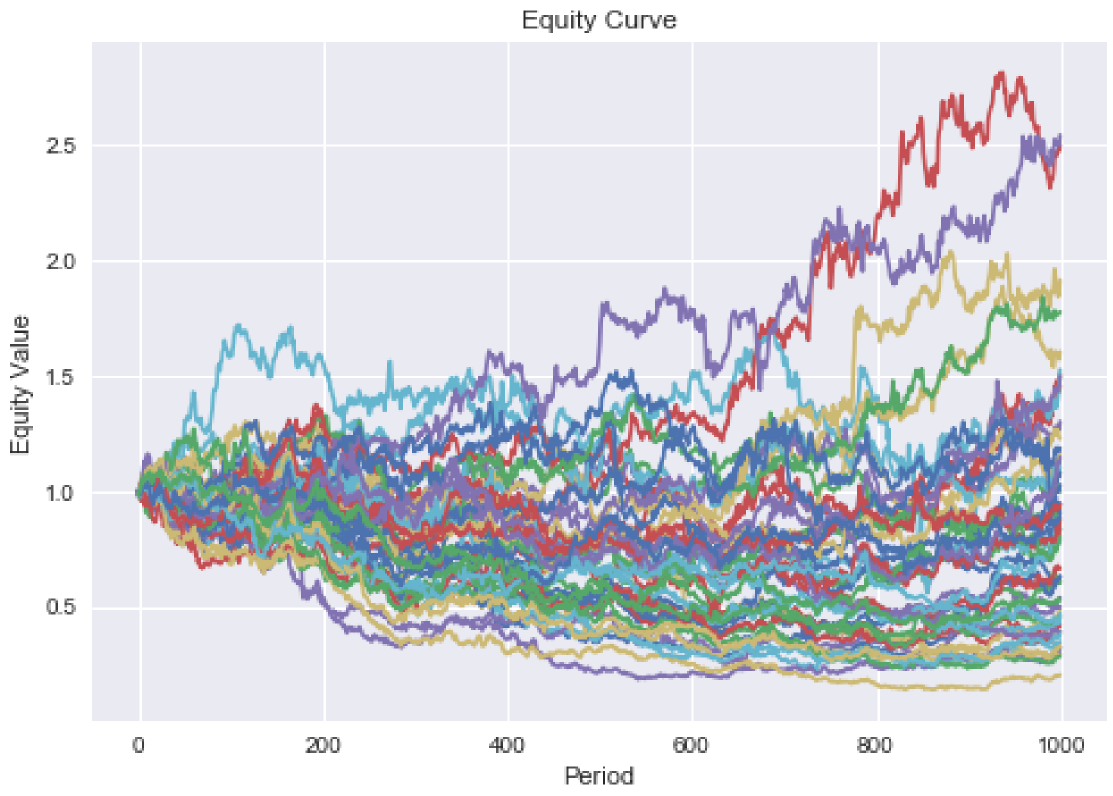
$$\begin{aligned}
\mathrm{E}\left(\sigma_t^2\right) &= \omega + \sum_{i=1}^{q} \alpha_i \mathrm{E}\left(\epsilon_{t-i}^2\right) + \sum_{i=1}^{p} \beta_i \mathrm{E}\left(\sigma_{t-i}^2\right) \\
&= \omega + \sum_{i=1}^{q} \alpha_i \mathrm{E}\left(\eta_{t-i}^2 \sigma_{t-i}^2\right) + \sum_{i=1}^{p} \beta_i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

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 nise distribution look like this.

```
In [36]: 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)

    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)
```

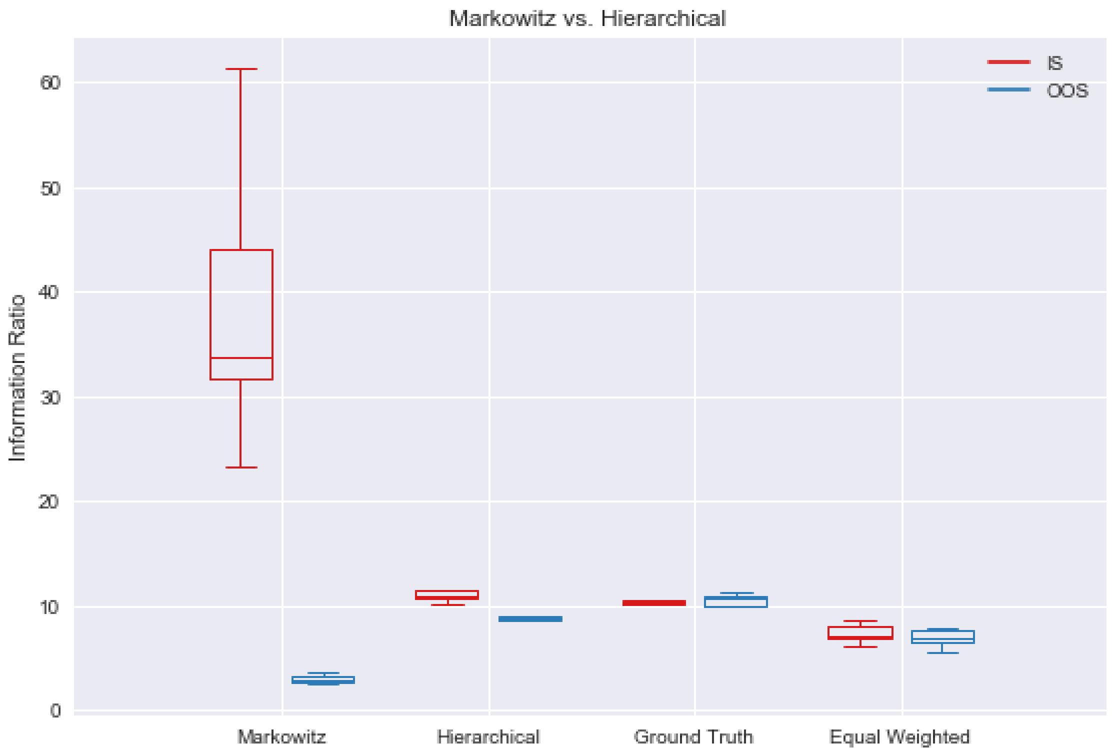
```
            ir_hier_is.append(IR_ann)

            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:  10.314924137730154
IR gt_oos:  11.219773222633183


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


IR sm_is:  44.07406918703761
IR sm_oos:  3.299182310696336
{'banks': 0.15967829378891055, 'oil': 0.1636020654087193, 'insurance':
0.04078337564155619, 'tech': 0.2508566420893103, 'bio': 0.09343134145129015, 'pharma':
0.05204879547856919, 'auto': 0.053656763591487855, 'retail': 0.15069582320238875,
'manufacturing': 0.03524689934776772}
IR hier_is:  10.687466946383722
IR hier_oos:  10.355871865738731
IR equal_is:  8.646609676920752
IR equal_oos:  7.736393224746492
MC run:  1
IR gt_is:  10.174839063328545
IR gt_oos:  9.964522319055435
IR sm_is:  33.77882119655665
IR sm_oos:  2.4542785693304356
{'banks': 0.0795117291238569, 'oil': 0.14235509568143997, 'insurance':
0.14007480362066402, 'tech': 0.06853274182949477, 'bio': 0.08809651316811047,
'pharma': 0.20047352348159514, 'auto': 0.07015826481167935, 'retail':
0.16301195251625608, 'manufacturing': 0.04778537576690336}
IR hier_is:  10.98426987771284
IR hier_oos:  8.69865353488074
IR equal_is:  8.060779706831138
IR equal_oos:  7.9023600364309345
```

```
MC run:  2
IR gt_is:  11.584320797566154
IR gt_oos:  10.713938421161808
IR sm_is:  61.391905413003606
IR sm_oos:  2.77196109978414
{'banks': 0.11744003832461196, 'oil': 0.0833591642101323, 'insurance':
0.04207044533458641, 'tech': 0.15800493451150344, 'bio': 0.19528268451668537,
'pharma': 0.18413031367391863, 'auto': 0.055619872788576386, 'retail':
0.1033814383975889, 'manufacturing': 0.06071110824239664}
IR hier_is:  12.567557047983506
IR hier_oos:  8.935635490111618
IR equal_is:  7.163851823883613
IR equal_oos:  6.89399462412708
MC run:  3
IR gt_is:  10.510696822965606
IR gt_oos:  9.917822616448193
IR sm_is:  23.24424801054156
IR sm_oos:  2.85689345810808
{'banks': 0.057071875629943064, 'oil': 0.13811942257648732, 'insurance':
0.06719395752458439, 'tech': 0.10928283339768104, 'bio': 0.1724985366005242, 'pharma':
0.14528957407974658, 'auto': 0.09567592998414085, 'retail': 0.05167713025092805,
'manufacturing': 0.16319073995596442}
IR hier_is:  11.392469285226964
IR hier_oos:  8.57303516322006
IR equal_is:  6.9072710495518
IR equal_oos:  5.615288496477106
MC run:  4
IR gt_is:  9.412876464600133
IR gt_oos:  10.85642244057754
IR sm_is:  31.691928838527556
IR sm_oos:  3.6671934307537186
{'banks': 0.07644752817679888, 'oil': 0.17235966474892608, 'insurance':
0.05912214359740948, 'tech': 0.06594169325594036, 'bio': 0.06702133246179037,
'pharma': 0.3639364249129581, 'auto': 0.07309227555950167, 'retail':
0.032910189667125986, 'manufacturing': 0.08916874761954892}
IR hier_is:  10.165432999344953
IR hier_oos:  8.796881195995462
IR equal_is:  6.183074202060472
IR equal_oos:  6.44793905253365
```

```
In [29]: 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:  3.014056979917056
```

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

```
OOS Equal / Markowitz:  2.298810927068244
```

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

```
OOS Hierarchical / Equal:  1.3111373990904904
```

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
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 10.758594652817013
IR gt_oos 11.113050819241867
```

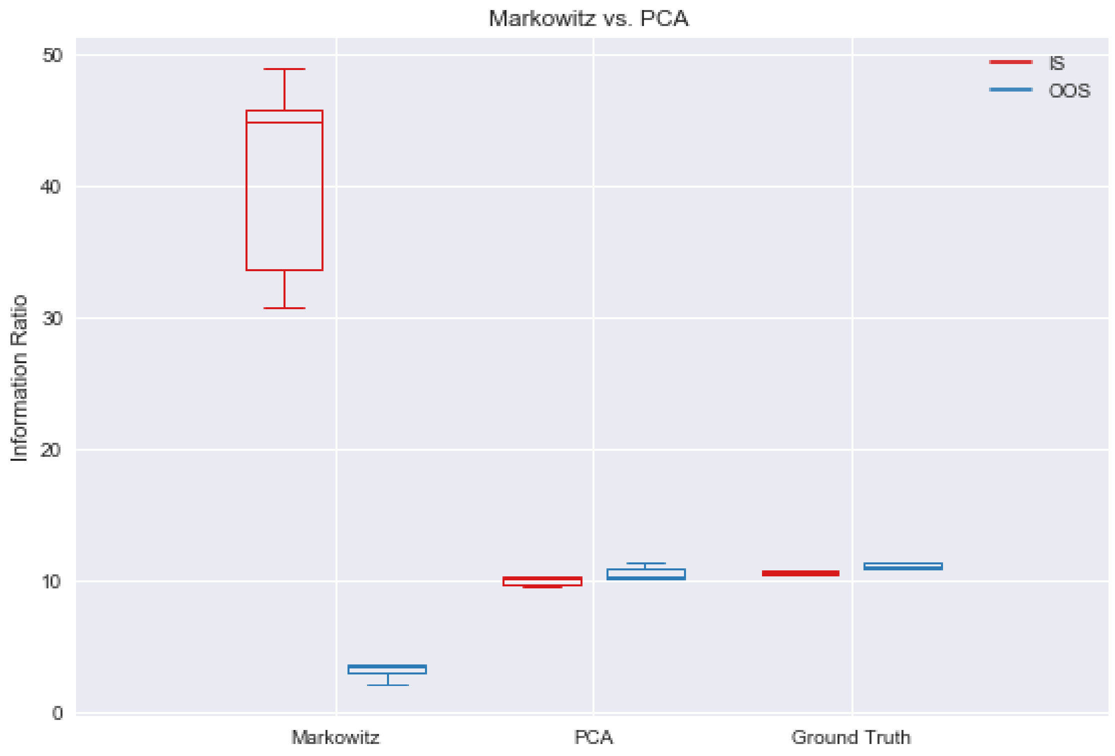
```
/Users/jan/Documents/PoCon/src/optimize.py:30: UserWarning: Optimizer did not
converge.
  warnings.warn("Optimizer did not converge.")
```

```
IR sm_is:  45.786748204016575
IR sm_oos:  2.1310671782905524
IR pca_is:  10.318541498885843
IR pca_oos:  10.98982646744523
MC run:  1
IR gt_is 10.67688681381962
IR gt_oos 11.421483144077598
IR sm_is:  48.90253528100965
IR sm_oos:  3.051585524228475
IR pca_is:  10.247278634275052
IR pca_oos:  10.41375102506033
MC run:  2
IR gt_is 12.176829836445028
IR gt_oos 12.06736269017553
IR sm_is:  30.738471183724553
IR sm_oos:  3.7438989461220653
IR pca_is:  11.642604559997816
IR pca_oos:  11.422175067044499
MC run:  3
IR gt_is 10.53346113430525
IR gt_oos 9.854214602957956
IR sm_is:  33.7132282314814
IR sm_oos:  3.7362309737404167
IR pca_is:  9.563417908861911
IR pca_oos:  8.58128351620866
MC run:  4
IR gt_is 10.431839695937438
IR gt_oos 10.994779704295947
IR sm_is:  44.819303024148425
IR sm_oos:  3.5385970189220033
IR pca_is:  9.753478592732034
IR pca_oos:  10.152142979890636
```

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

```
OOS PCA / Markowitz:  3.1823943514171646
```

```
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 [ ]:
```