Understanding Portfolio Optimization by Simulation

Jan Philipp Wöltjen

October 22, 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

$$\mathrm{Cov}\left(\boldsymbol{f}_t\right) = \boldsymbol{\Sigma}_f,$$

where

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

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

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

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

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

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

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

That is,

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

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

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

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

where

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

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

Thus the variance of asset i is

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

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

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

```
In [3]: 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
```

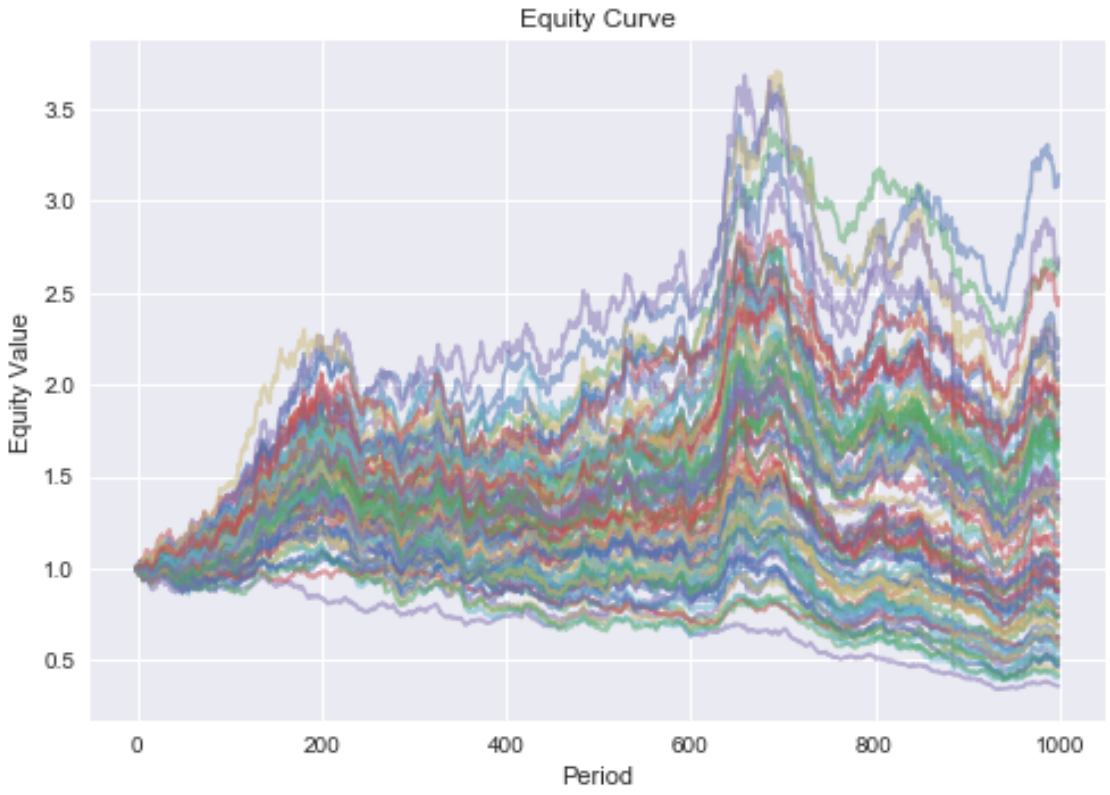
```
def gen_industry_stocks(n_stocks, market, industry, industry_name, sigma_noise,
                        p_year=250):
    mu = pd.Series(np.random.uniform(low=-0.2, high=0.2, size=n_stocks))/p_year
    beta_market = pd.Series(np.random.uniform(low=0.5, high=2.0, size=n_stocks))
    beta_industry =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


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]
```

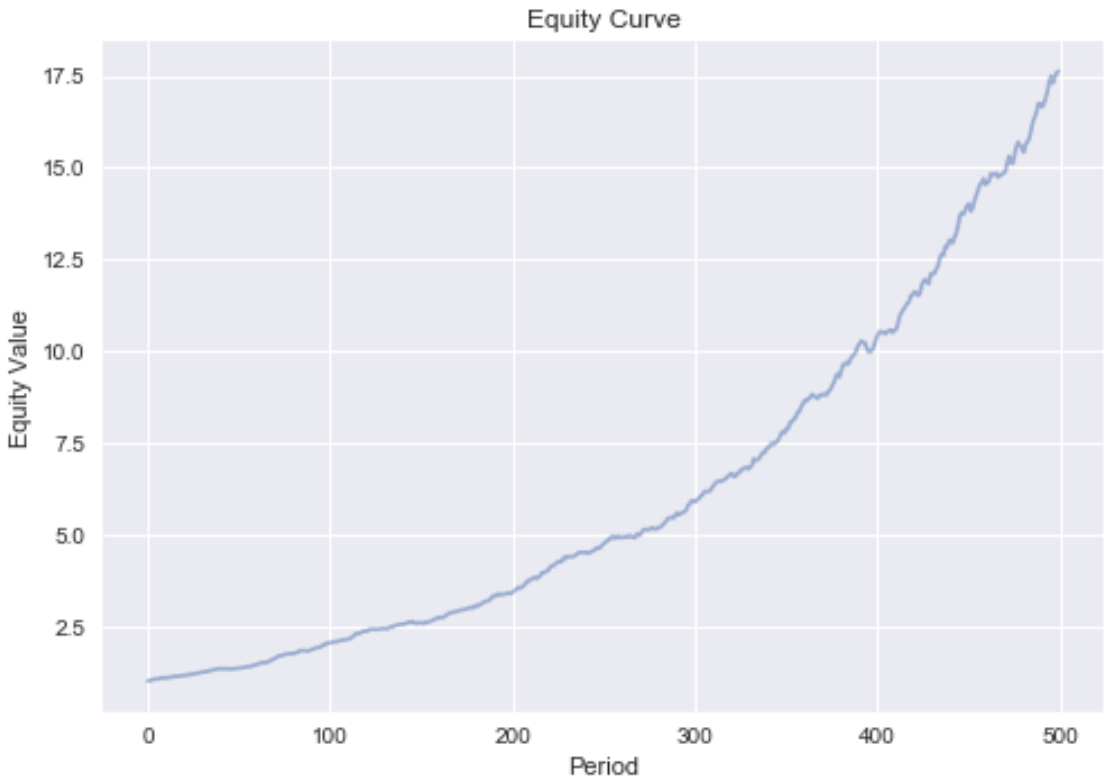
```
    IR_ann = information_ratio(portfolio_equal_oos)
    print('IR equal_oos: ', IR_ann)
    plot_equity_curve(portfolio_equal_oos)

    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 = True
```

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 and there is only one industry. It definitely beats equal weighting (equal).

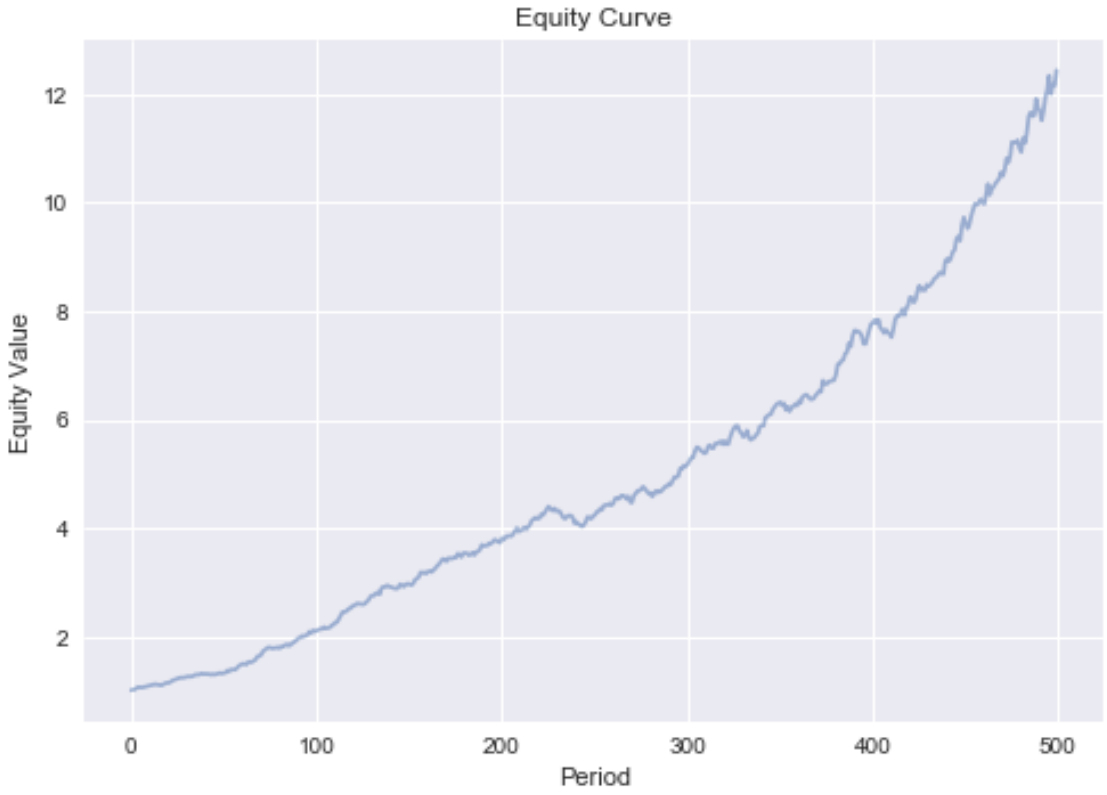
```
In [65]: industry = {}
         industry['manufacturing'] = norm.rvs(size=(1, num_samples))[0]*sigma['manufacturing']
         _ = show_standard_optimized_equity(n_stocks=100, industry=industry)
```



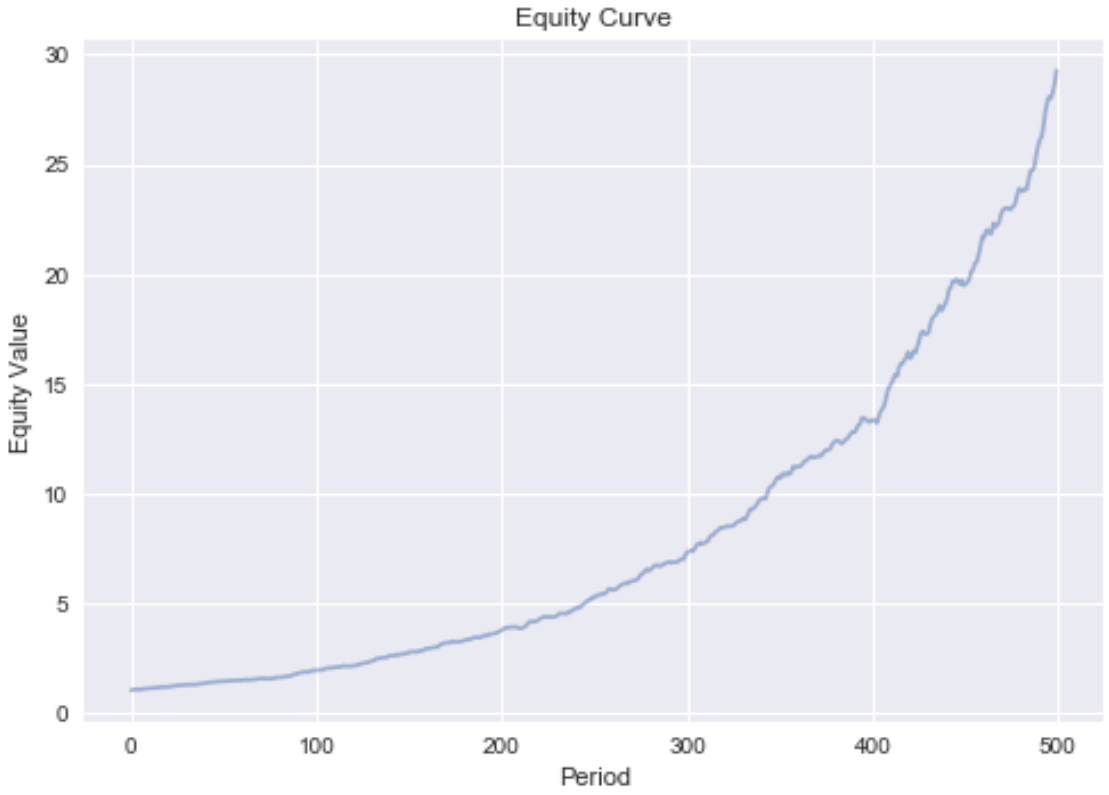

```
IR sm_is:  11.781615280191222
```



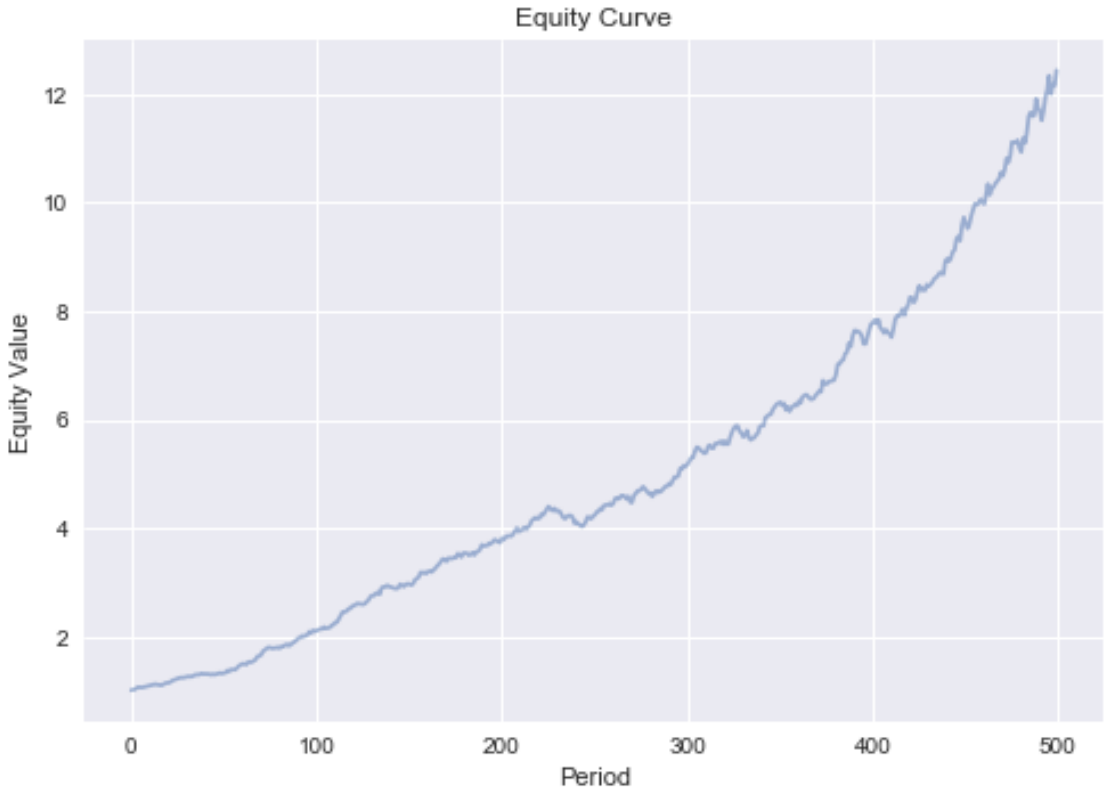

```
IR equal_is:  7.308332949721227
```




```
IR sm_oos:  12.294645503589331
```

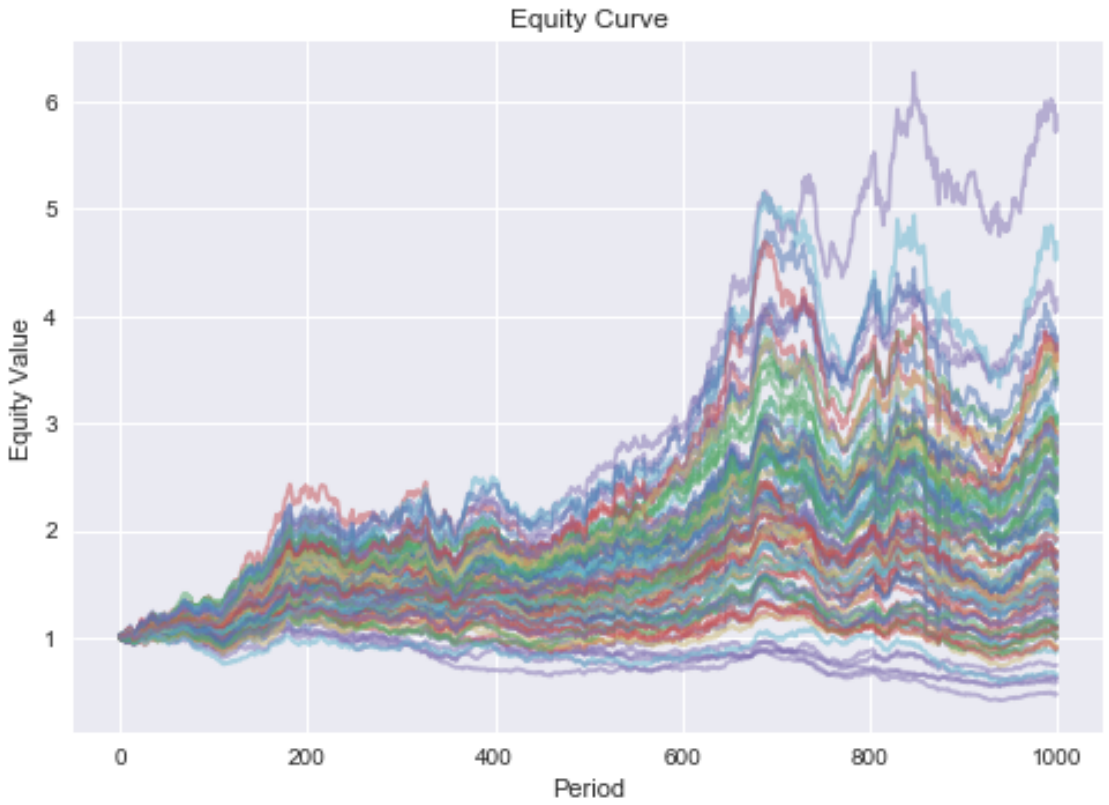



```
IR equal_oos:  7.308332949721227
```

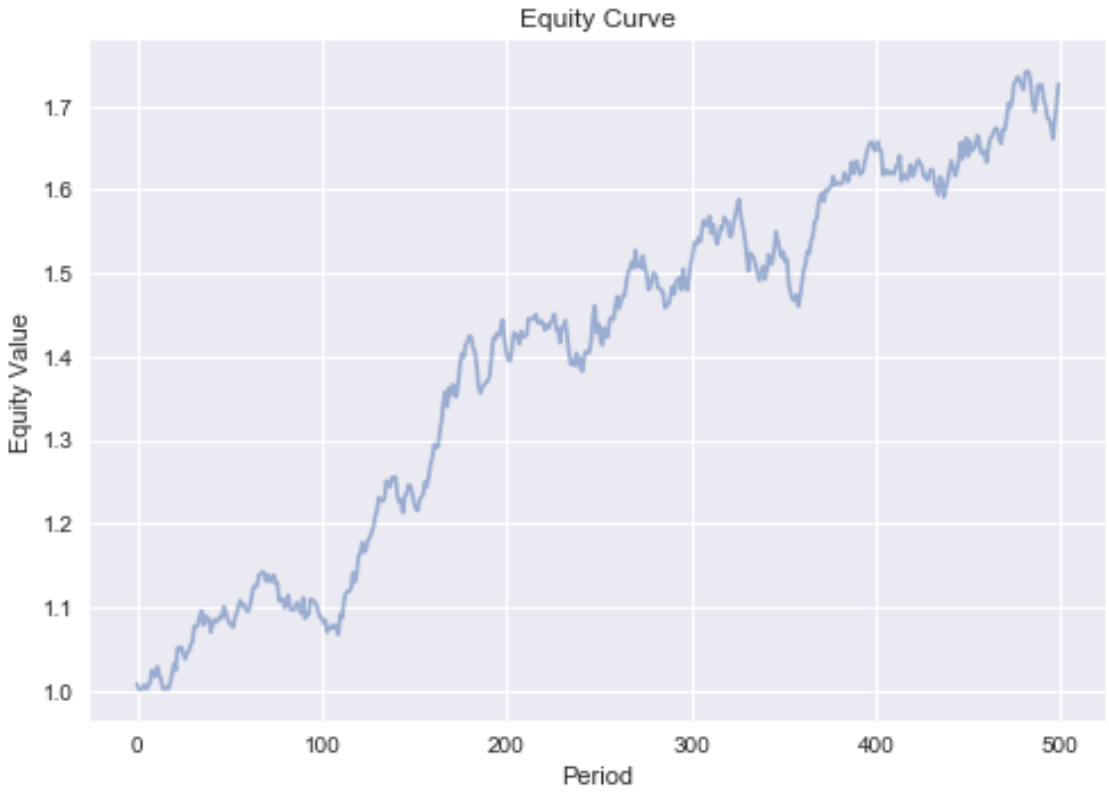


As shown below, long-only performance, of course, is horrendous. This is due to the fact that we cannot diversify the systematic risk. Since we cannot short, hedging out that risk is also not possible. Hence, our bets are not independent and the Law of Large Numbers does not apply.

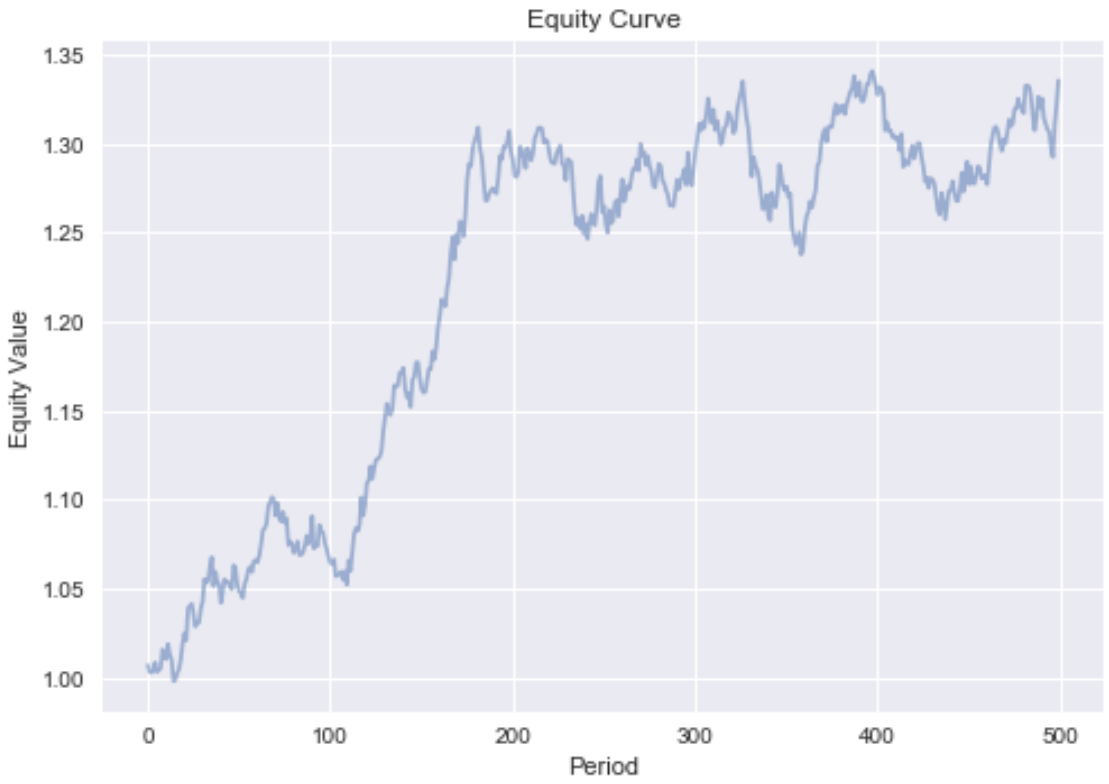
```
In [67]: _ = show_standard_optimized_equity(n_stocks=100, industry=industry, long_only=True)
```



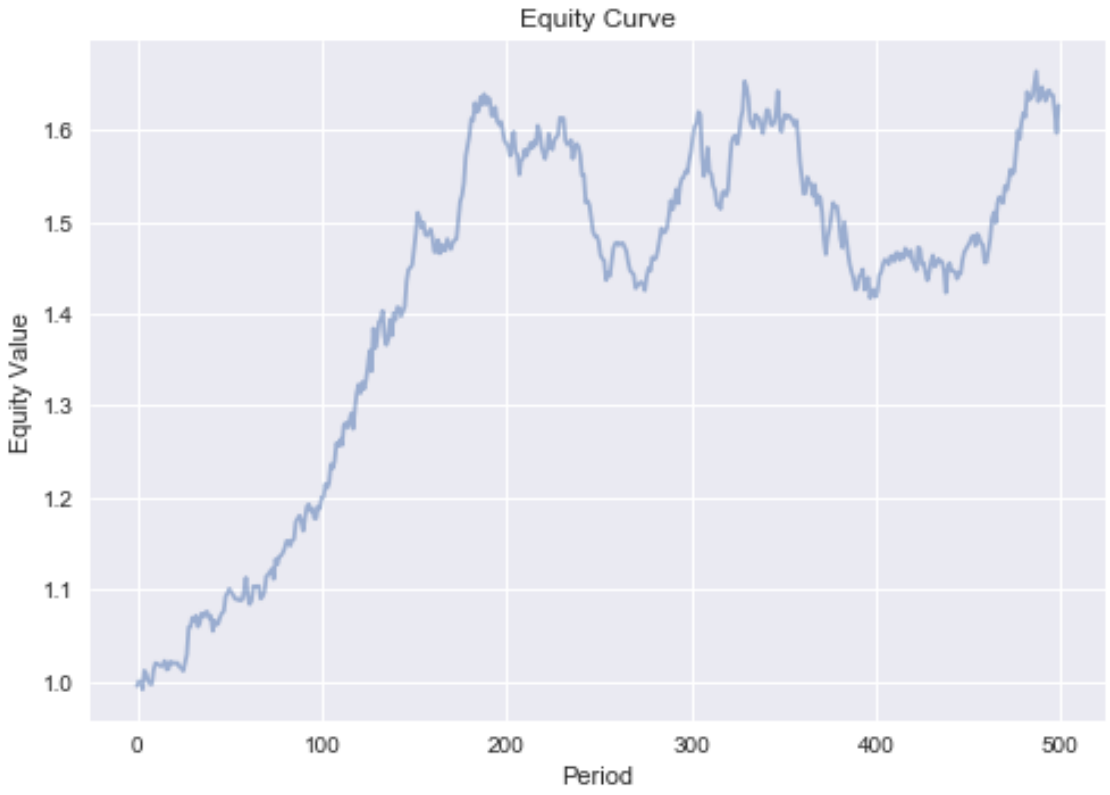

```
IR sm_is:  2.159732583044145
```



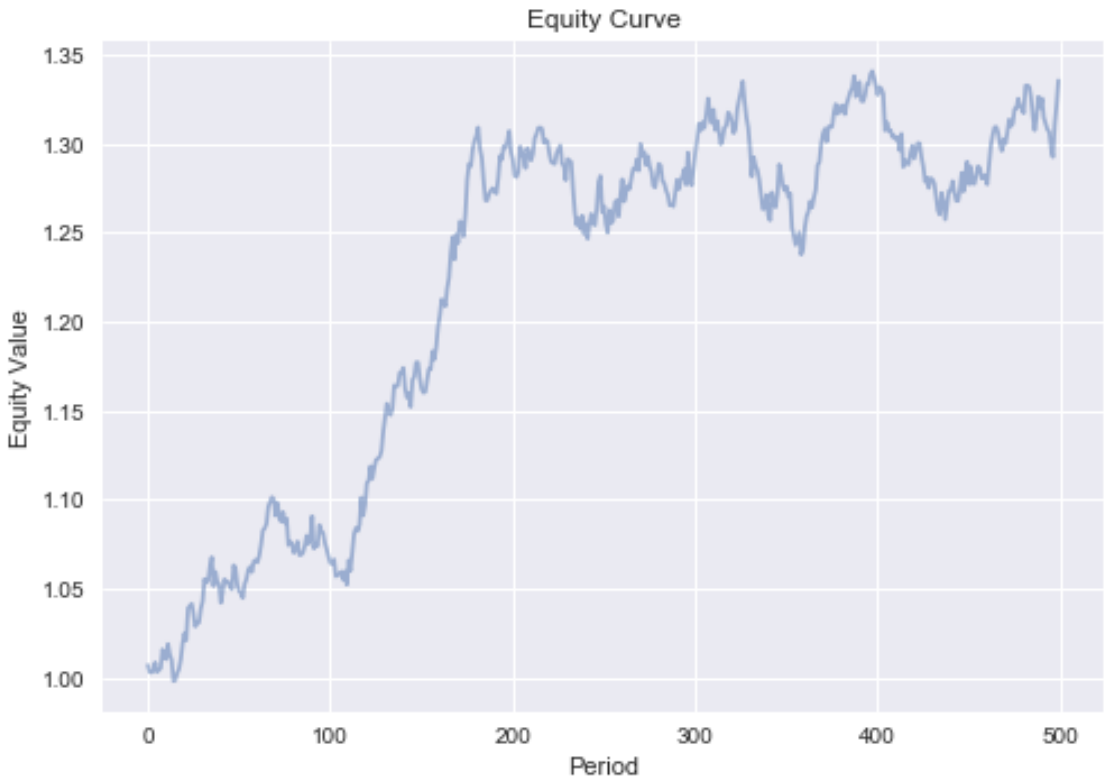

```
IR equal_is:  1.570613035352226
```




```
IR sm_oos:  1.856914588723952
```

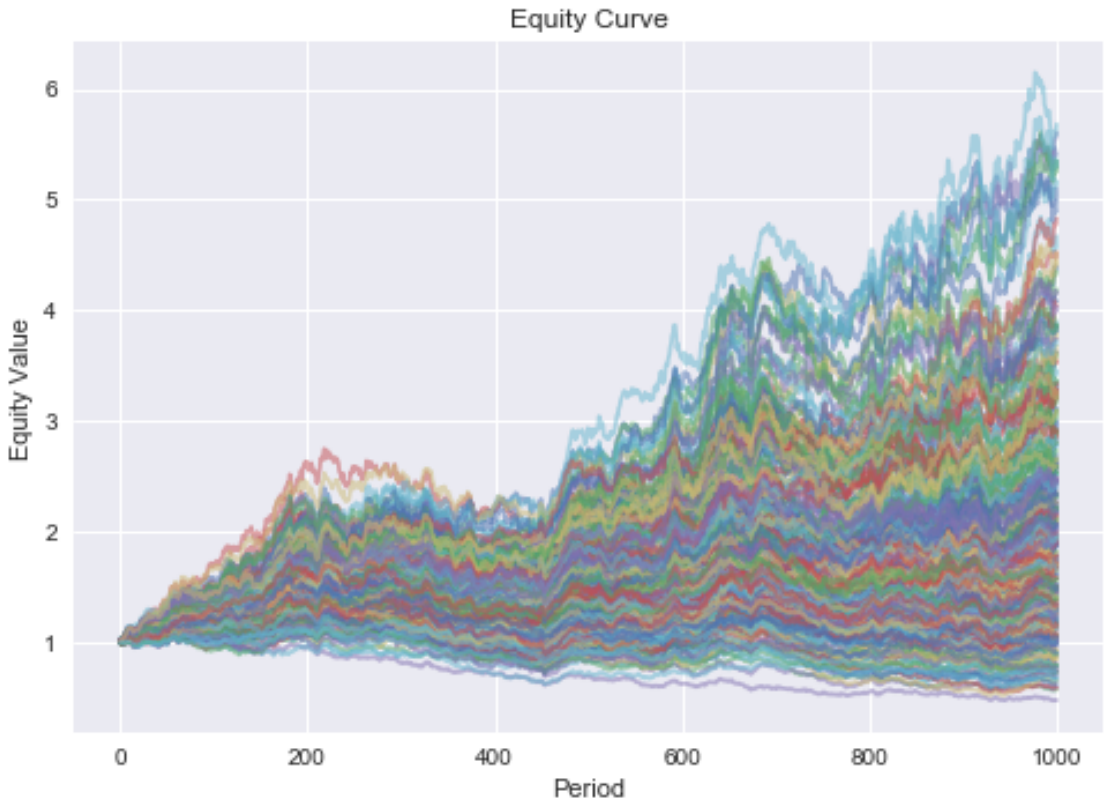



```
IR equal_oos:  1.570613035352226
```

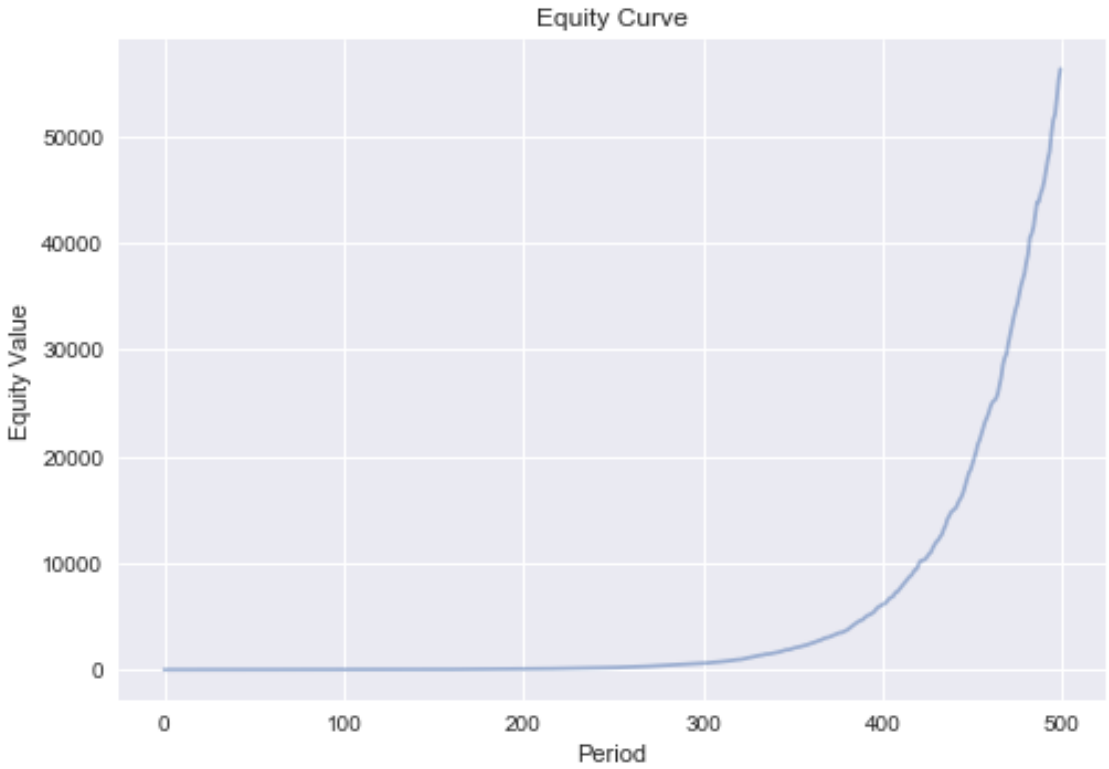


Standard mean-variance optimization works well if the number of assets is relatively small. When the number of assets is relatively large, however, the out-of-sample performance is significantly worse than the in-sample performance as demonstrated by the information ratio (IR). Even the very naive approach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

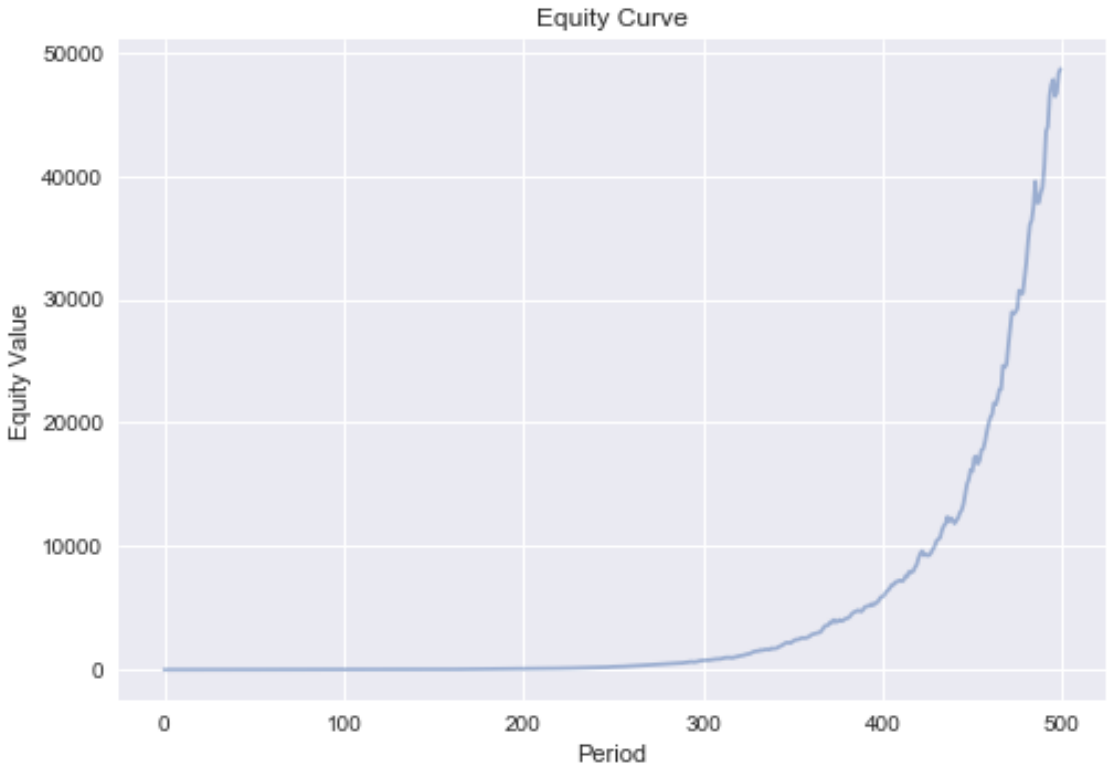
```
In [68]: weights = show_standard_optimized_equity(n_stocks=300, industry=industry)
```



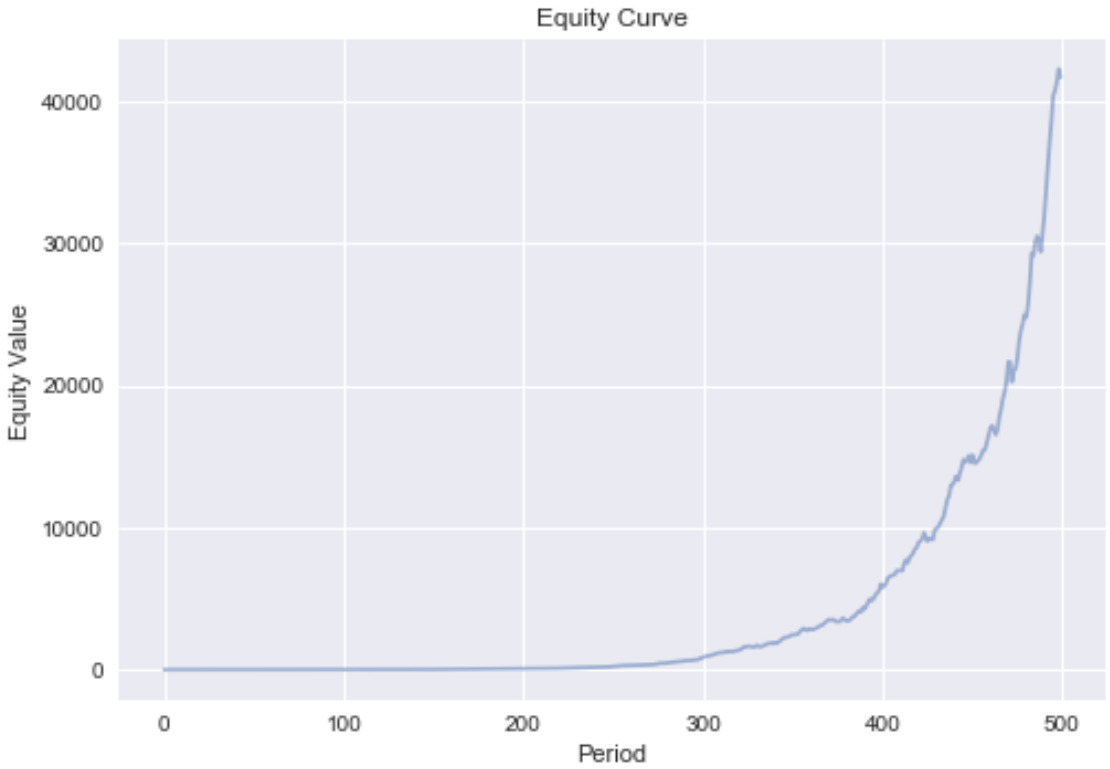

```
IR sm_is:  35.54196901119124
```



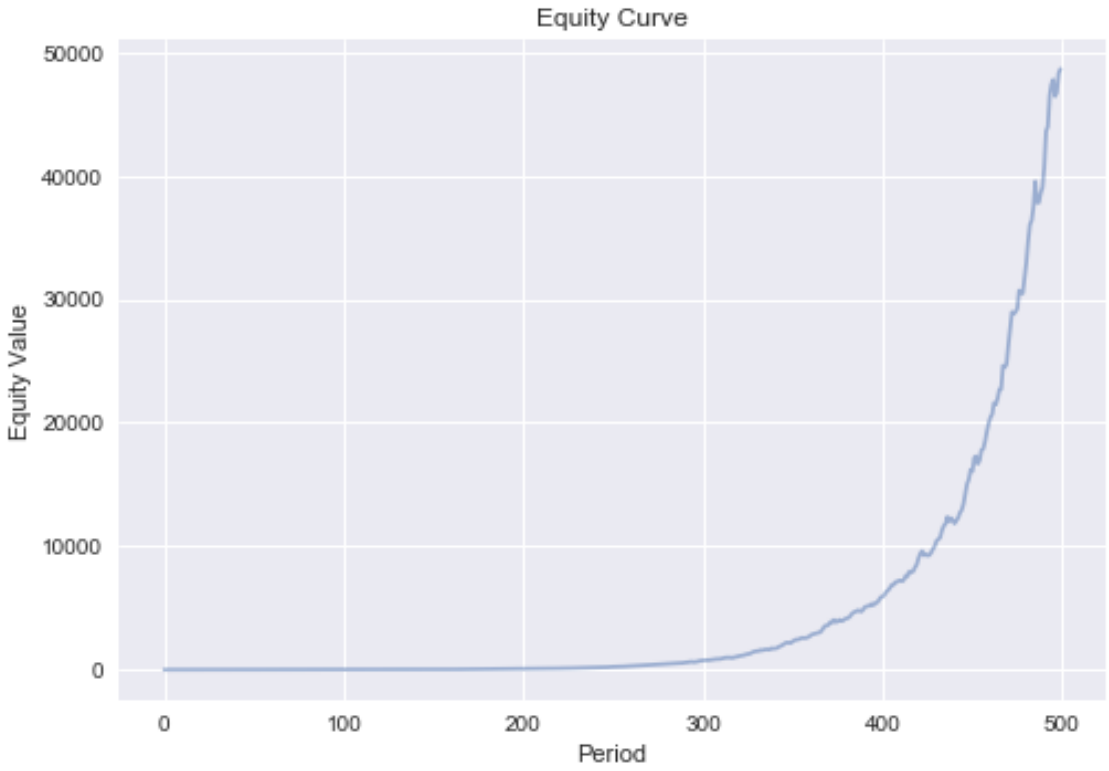

```
IR equal_is:  13.894520897023506
```




```
IR sm_oos:  12.450661665788113
```




```
IR equal_oos:  13.894520897023506
```




```
In [69]: # 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, $28.37 Long, $28.37 Short
largest long: manufacturing_stock_24 0.7335872080882955
largest short: manufacturing_stock_221 -0.675822795422206
```

```
In [70]: # Showing the first 5 positions in portfolio
         pf.portfolio_df.head()
```

Out[70]:

	position	price	factor_value	sector_id	\
manufacturing_stock_24	0.733587	1	1.0	0.0	
manufacturing_stock_139	0.661467	1	1.0	0.0	
manufacturing_stock_210	0.487374	1	1.0	0.0	
manufacturing_stock_281	0.474867	1	1.0	0.0	
manufacturing_stock_40	0.473963	1	1.0	0.0	

	position_value
manufacturing_stock_24	0.733587
manufacturing_stock_139	0.661467

```
manufacturing_stock_210        0.487374
manufacturing_stock_281        0.474867
manufacturing_stock_40         0.473963
```

```
In [71]: # Plotting the weight distribution
         pf.portfolio_df.position_value.plot(style='.')
         plt.title('Position Allocation')
         plt.xlabel('Ordered Positions')
         plt.ylabel('Weight')
         plt.show()
```

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.

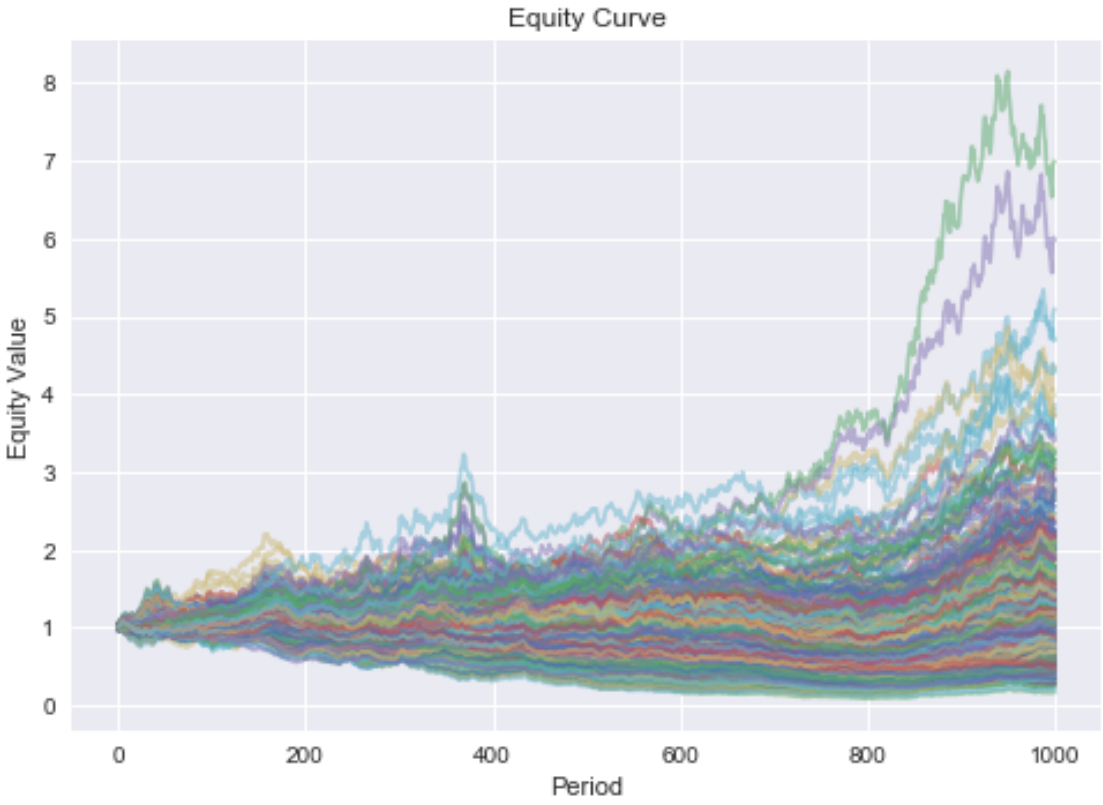
```
In [72]: trainig_pct = 0.5
         n_train = int(trainig_pct*num_samples)
         ir_sm_is = []
         ir_sm_oos = []
         ir_hier_is = []
         ir_hier_oos = []
         ir_gt_is = []
         ir_gt_oos = []
         ir_equal_is = []
         ir_equal_oos = []
```

```
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)
```

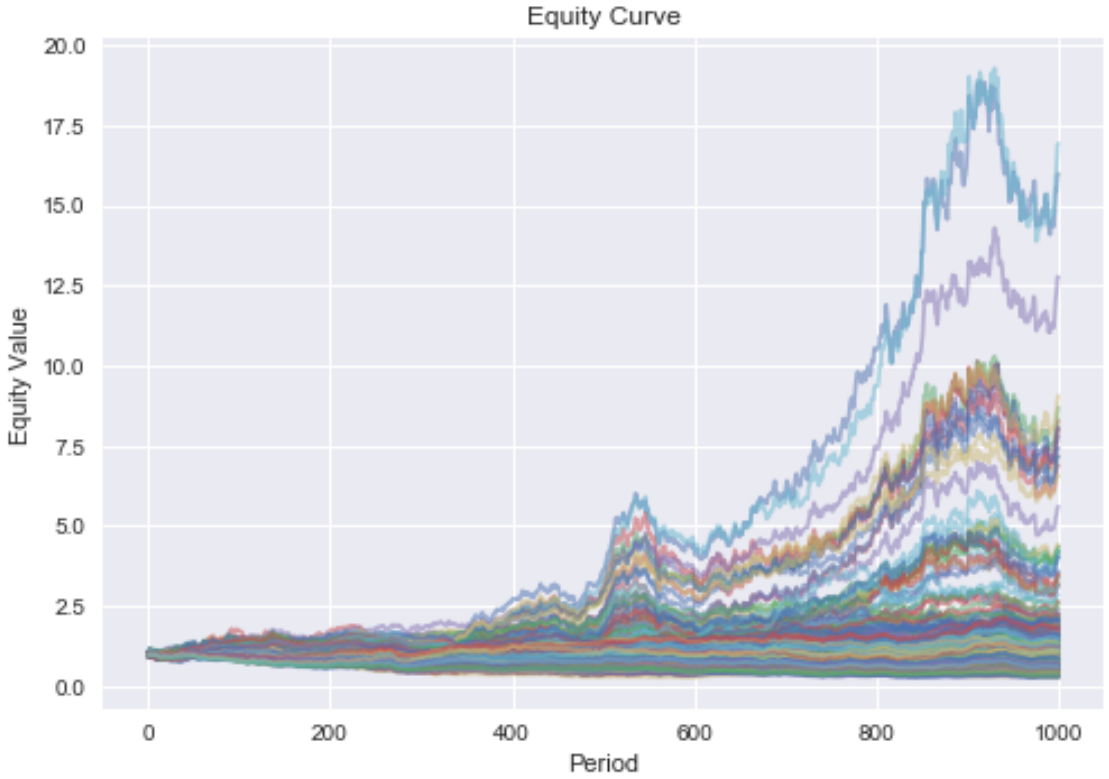
```
MC run:  0
```



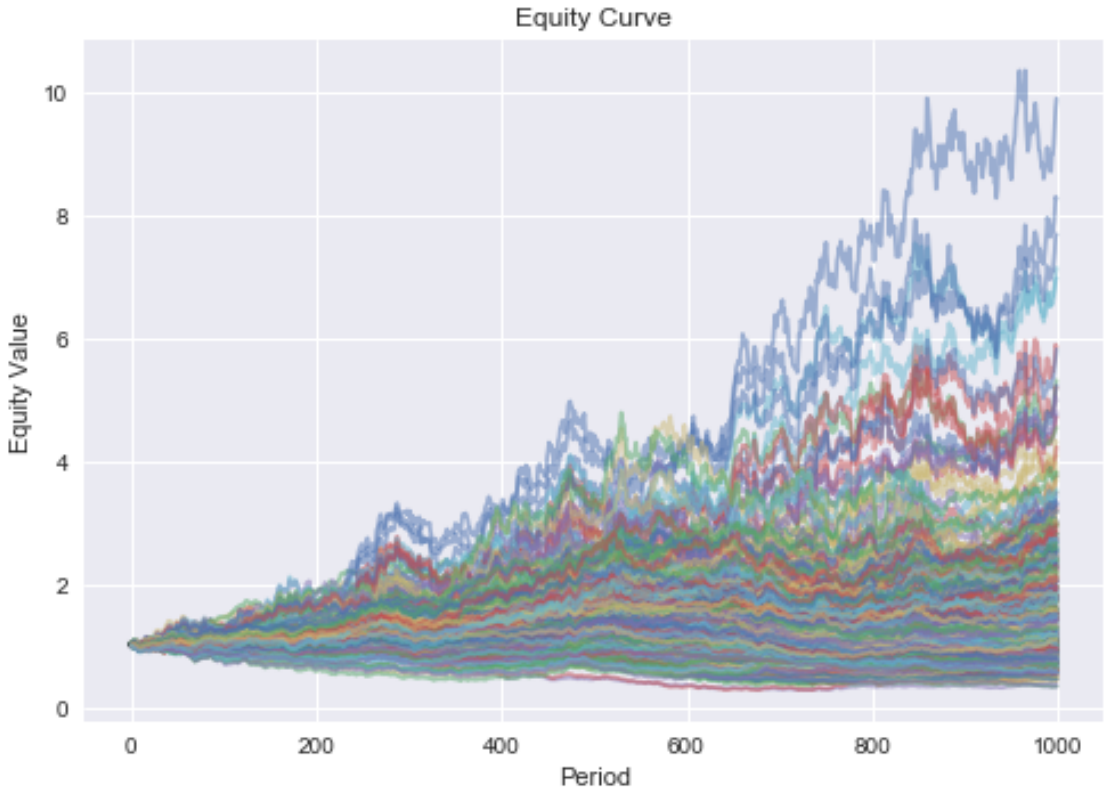

```
IR gt_is:  23.340094968228197
IR gt_oos:  24.84885845132815
```

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

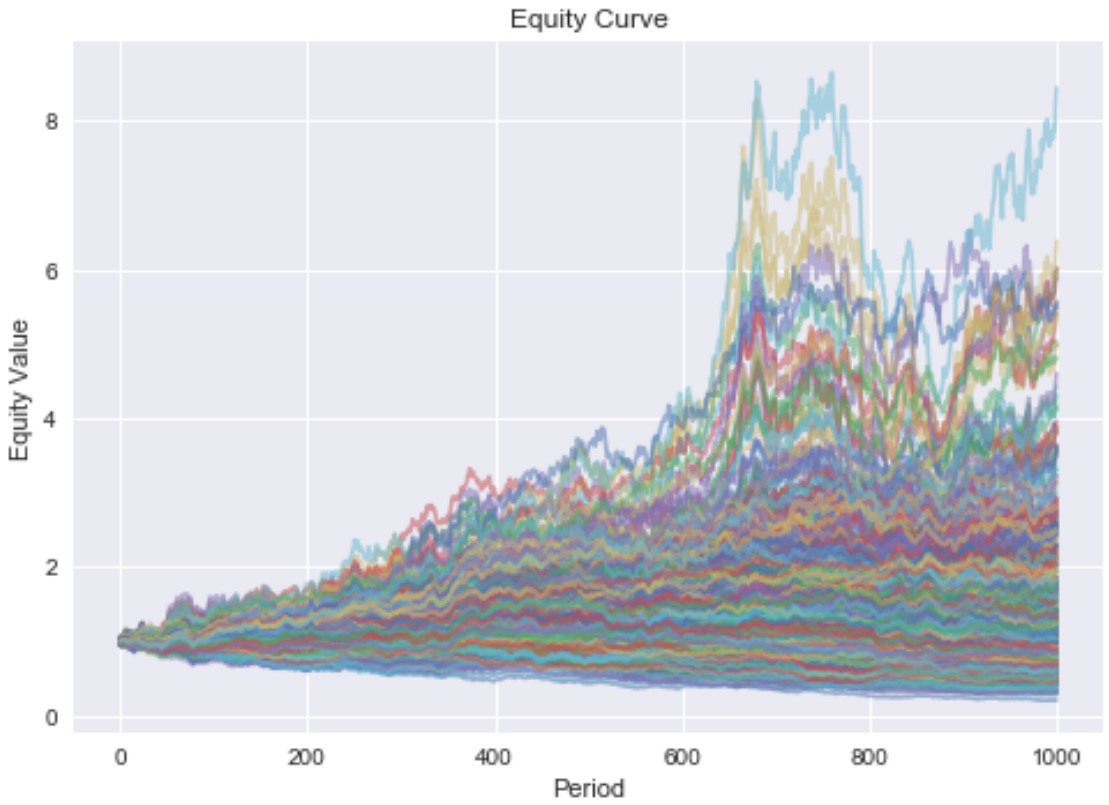
```
IR sm_is:  76.99169498172355
IR sm_oos:  8.955731023658574
{'banks': 0.013780615233506703, 'oil': 0.14201227018473386, 'insurance':
0.009676804266535381, 'tech': 0.1345772540739004, 'bio': 0.07485637514027982,
'pharma': 0.2882964146265688, 'auto': 0.09892821969011271, 'retail': 0.21679805579151,
'manufacturing': 0.021073990992852218}
IR hier_is:  22.549160734242435
IR hier_oos:  21.884894568370957
IR equal_is:  11.650715442912684
IR equal_oos:  11.084233419387685
MC run:  1
```



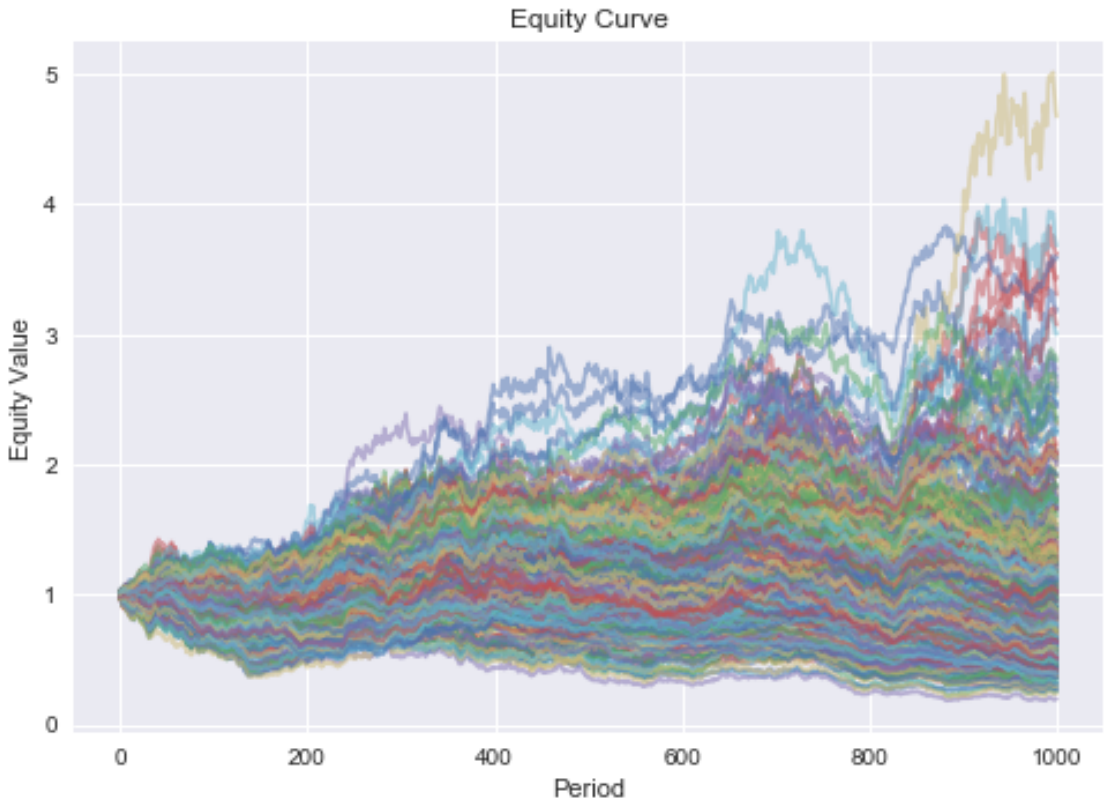

```
IR gt_is:  22.92867190408647
IR gt_oos:  23.170266833792837
IR sm_is:  74.99752490483553
IR sm_oos:  7.932286299455993
{'banks': 0.008673865012932838, 'oil': 0.13696938530822314, 'insurance':
0.03413376151438785, 'tech': 0.10079481907854868, 'bio': 0.09617349977702308,
'pharma': 0.4578338345314104, 'auto': 0.042503734567486295, 'retail':
0.11744146362216233, 'manufacturing': 0.005475636587825415}
IR hier_is:  19.45479460440231
IR hier_oos:  20.96110798031804
IR equal_is:  8.76685594886774
IR equal_oos:  8.884783011184641
MC run:  2
```



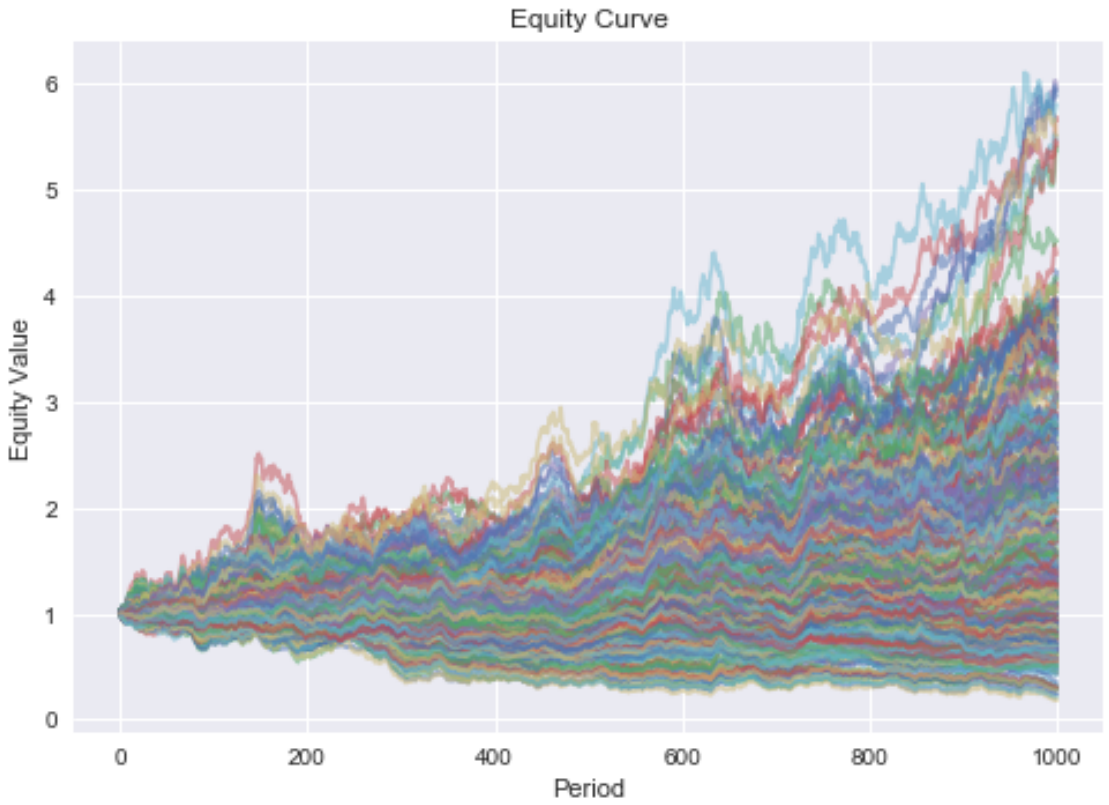

```
IR gt_is:  24.431673309601646
IR gt_oos:  22.601825803758434
IR sm_is:  98.96400578208231
IR sm_oos:  6.001669648530468
{'banks': 0.13370146687337, 'oil': 0.13296288908238935, 'insurance':
0.014591912505559062, 'tech': 0.44063875012285514, 'bio': 0.0450204282887834,
'pharma': 0.04637352884084802, 'auto': 0.04375305187888432, 'retail':
0.03653236287277768, 'manufacturing': 0.10642560953453305}
IR hier_is:  23.034196892139907
IR hier_oos:  20.18637458453616
IR equal_is:  12.365793533596324
IR equal_oos:  10.507026348853593
MC run:  3
```



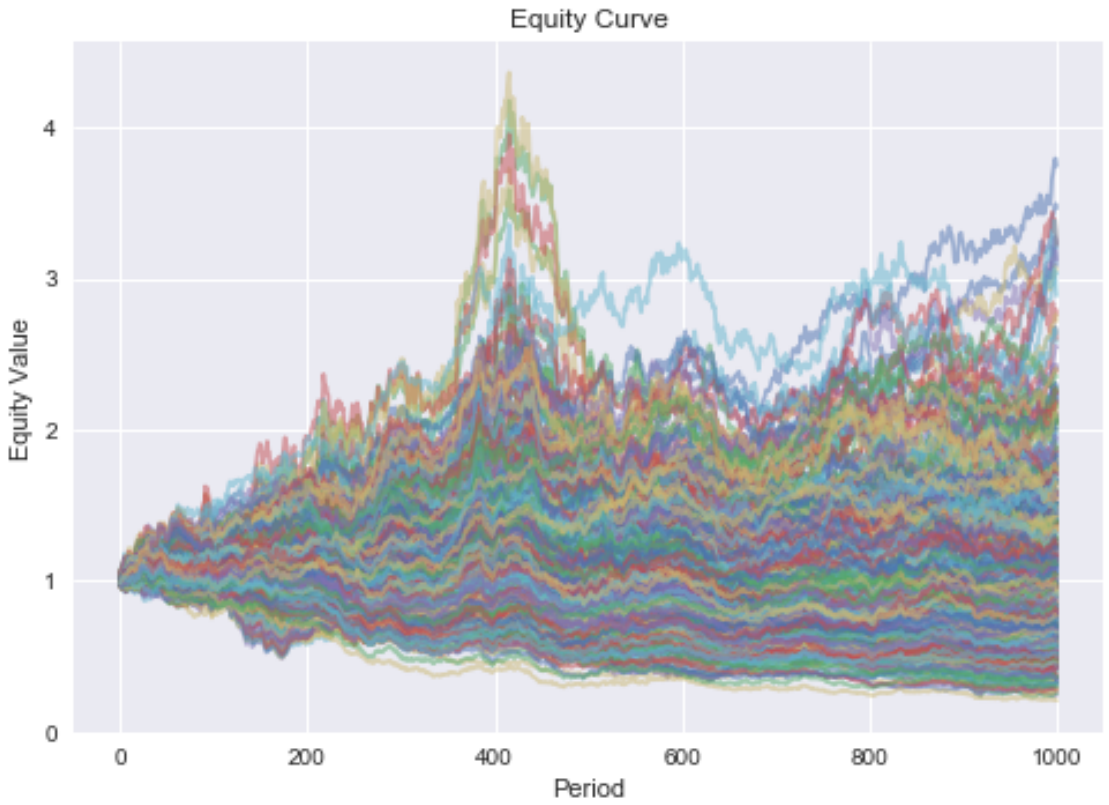

```
IR gt_is:  26.042757016758102
IR gt_oos:  25.796282333431822
IR sm_is:  72.36143003438261
IR sm_oos:  8.953162895837922
{'banks': 0.21968478727204713, 'oil': 0.0856898170145167, 'insurance':
0.060503625526665994, 'tech': 0.22168916693319649, 'bio': 0.019822730861306873,
'pharma': 0.04252899736061094, 'auto': 0.0326994418276498, 'retail':
0.3060512450953407, 'manufacturing': 0.011330188108665467}
IR hier_is:  24.45578292152887
IR hier_oos:  21.87804835274034
IR equal_is:  11.44780915728
IR equal_oos:  11.678691585772652
MC run:  4
```



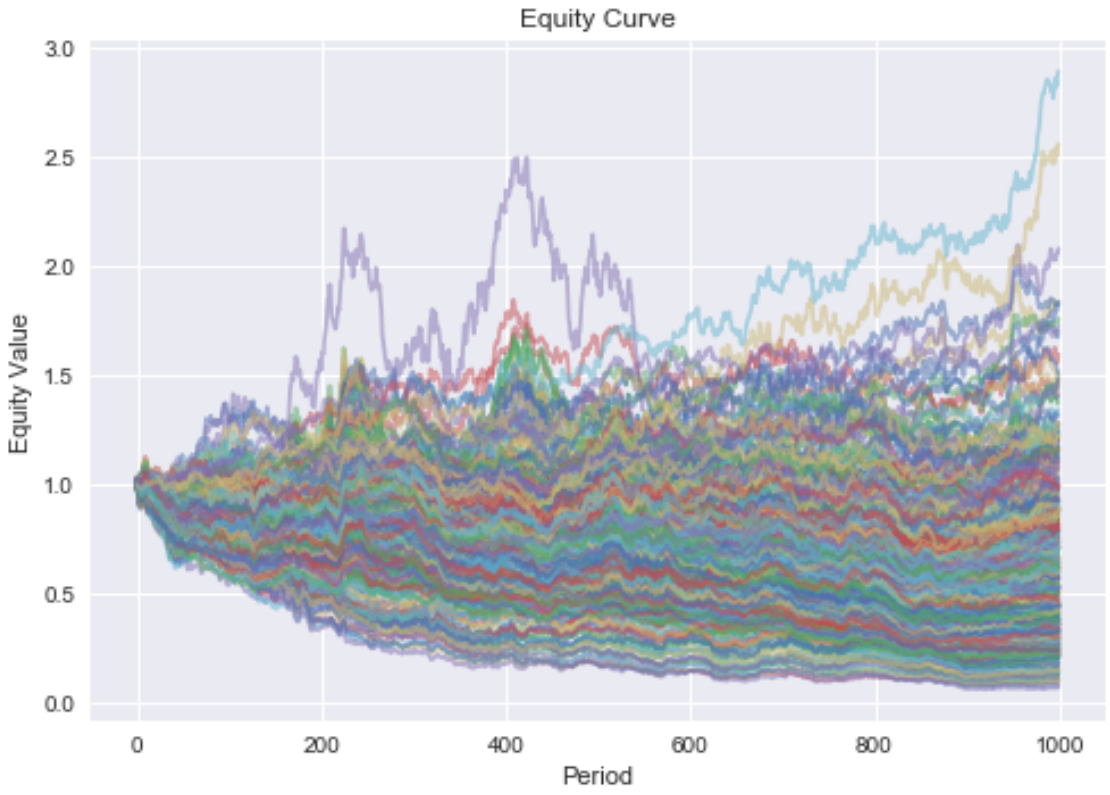

```
IR gt_is:  22.866150834756066
IR gt_oos:  25.937192783897746
IR sm_is:  76.28198586700957
IR sm_oos:  8.284362128742558
{'banks': 0.19567336654918158, 'oil': 0.22524463256030877, 'insurance':
0.03956982323159401, 'tech': 0.0867524592445317, 'bio': 0.0224379455994546, 'pharma':
0.0525771683543378, 'auto': 0.0617386106474663, 'retail': 0.04824462808140622,
'manufacturing': 0.26776136573171905}
IR hier_is:  21.76476202899824
IR hier_oos:  22.237800637760397
IR equal_is:  8.272626714482918
IR equal_oos:  8.281488422707096
MC run:  5
```



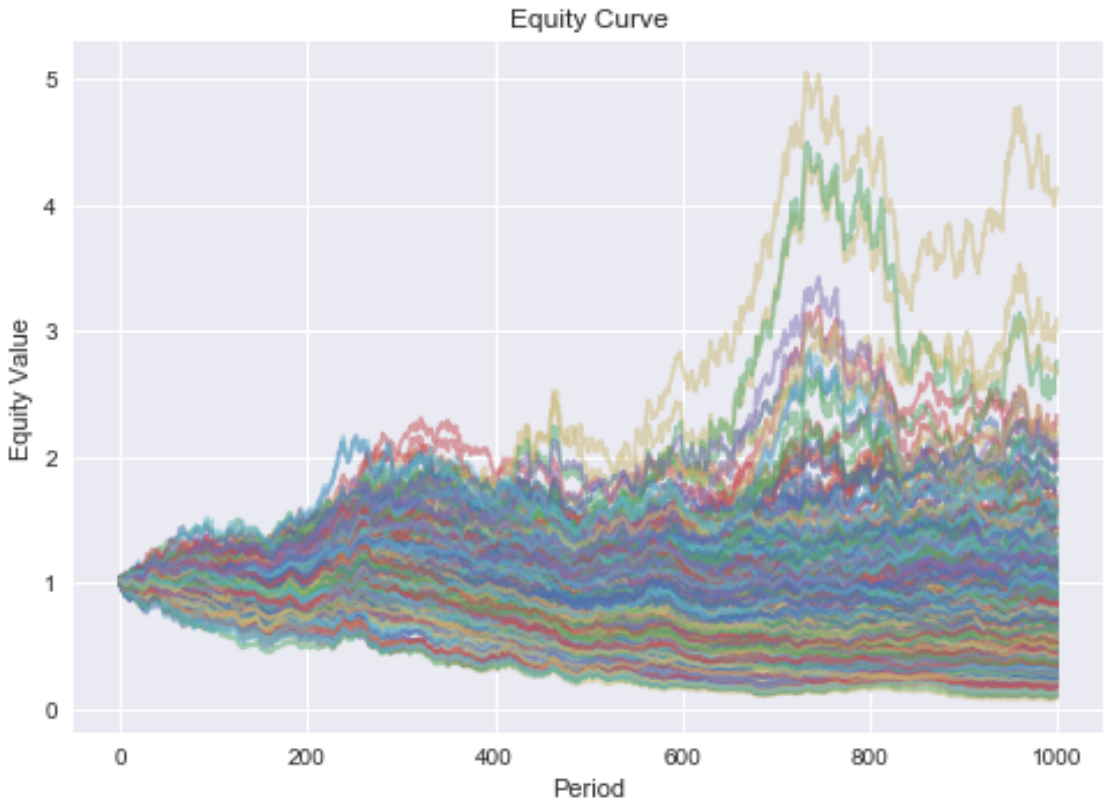

```
IR gt_is:  24.991006174755448
IR gt_oos:  25.53447541003423
IR sm_is:  67.6229032044295
IR sm_oos:  7.29574772298956
{'banks': 0.05481829529386708, 'oil': 0.08105231581549927, 'insurance':
0.035631920500844294, 'tech': 0.3783940384567301, 'bio': 0.028344282323862265,
'pharma': 0.10240989584046177, 'auto': 0.028203657317868797, 'retail':
0.21418086814093187, 'manufacturing': 0.07696472630993459}
IR hier_is:  23.122655767649963
IR hier_oos:  21.78049267143243
IR equal_is:  11.894338115022991
IR equal_oos:  11.844717129179656
MC run:  6
```



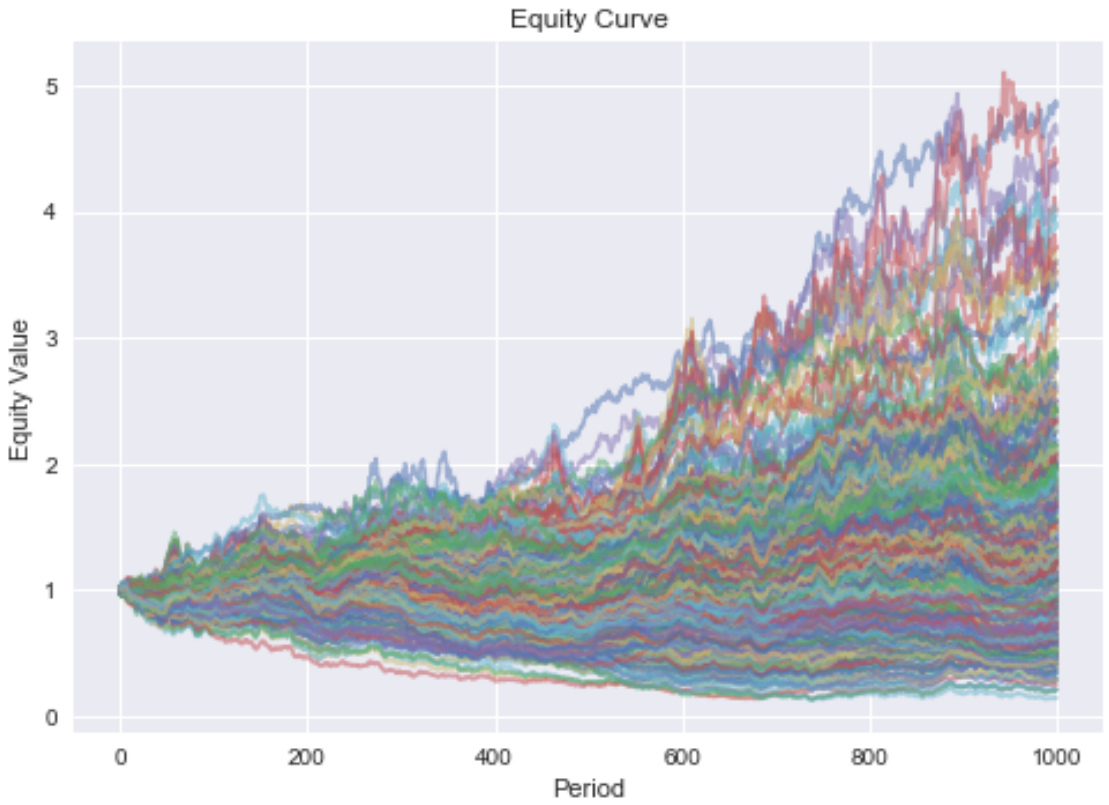

```
IR gt_is:  23.255816789796604
IR gt_oos:  25.13550051224693
IR sm_is:  83.32910299237359
IR sm_oos:  7.4890011847000615
{'banks': 0.034170248653794984, 'oil': 0.36012327783126435, 'insurance':
0.06770286709489834, 'tech': 0.3000905606467104, 'bio': 0.0452897191198764, 'pharma':
0.018139468349847455, 'auto': 0.015012963229877815, 'retail': 0.12327740542470181,
'manufacturing': 0.03619348964902843}
IR hier_is:  19.884053516907798
IR hier_oos:  20.721497297495137
IR equal_is:  6.760463456459012
IR equal_oos:  6.814176552114026
MC run:  7
```




```
IR gt_is:  22.582103773382933
IR gt_oos:  21.88105832277189
IR sm_is:  69.25628576129408
IR sm_oos:  8.062956735413001
{'banks': 0.09266494670331853, 'oil': 0.1621139284516062, 'insurance':
0.08895461994154263, 'tech': 0.14170959801178715, 'bio': 0.012026586822840859,
'pharma': 0.4619409256072568, 'auto': 0.014028723257752932, 'retail':
0.023002064246770756, 'manufacturing': 0.0035586069571242637}
IR hier_is:  20.694818600695832
IR hier_oos:  16.495707392664173
IR equal_is:  8.341255803239815
IR equal_oos:  8.996928291255331
MC run:  8
```



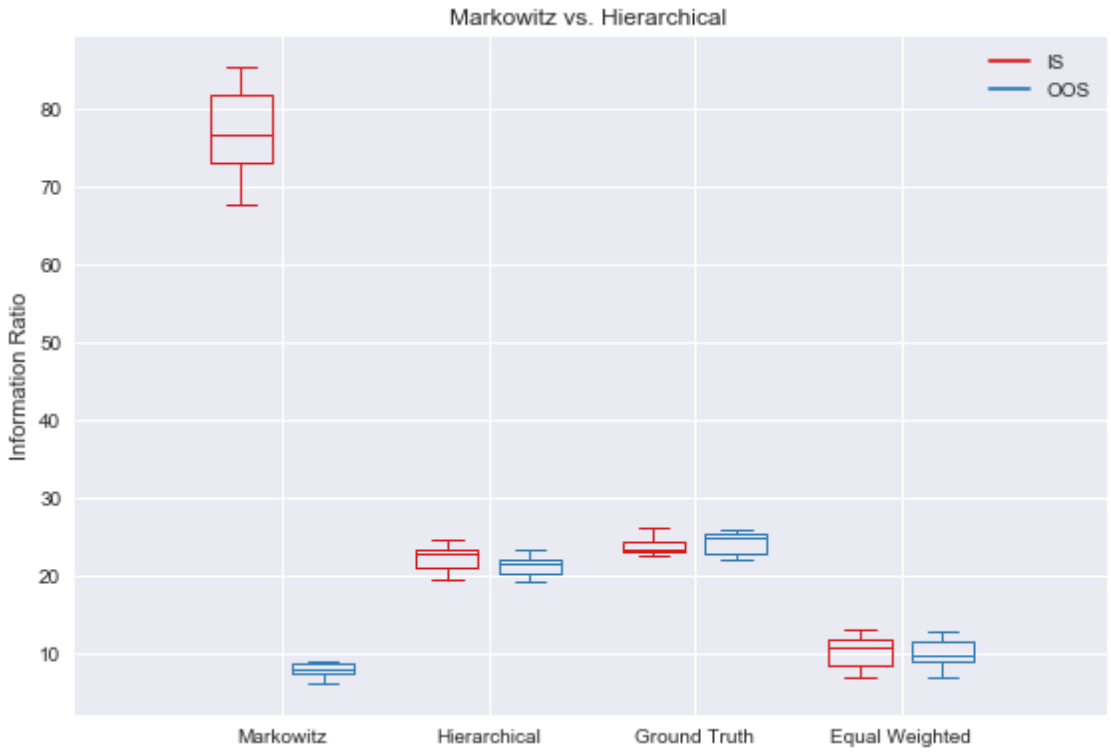

```
IR gt_is:  23.378802235039917
IR gt_oos:  21.97716459036835
IR sm_is:  77.42238517121754
IR sm_oos:  5.307559694586809
{'banks': 0.015108423035486864, 'oil': 0.06385799400944106, 'insurance':
0.02487492623885332, 'tech': 0.2770196833495348, 'bio': 0.03346345613193343, 'pharma':
0.1926252390129665, 'auto': 0.12273120378951531, 'retail': 0.15587867684891427,
'manufacturing': 0.1144403975833544}
IR hier_is:  23.326987410488957
IR hier_oos:  19.247278429260938
IR equal_is:  9.975721579837979
IR equal_oos:  9.0835738625144
MC run:  9
```




```
IR gt_is:  24.006165426224307
IR gt_oos:  25.052633802340562
IR sm_is:  85.38174228622609
IR sm_oos:  8.836540543543606
{'banks': 0.03625950480191997, 'oil': 0.08474190126356902, 'insurance':
0.13844759463987194, 'tech': 0.09310826023592658, 'bio': 0.20266693659819002,
'pharma': 0.2123862893186201, 'auto': 0.13561773966784968, 'retail':
0.027566316834781366, 'manufacturing': 0.06920545663927119}
IR hier_is:  24.28458907187489
IR hier_oos:  23.219045552279002
IR equal_is:  13.047933307889943
IR equal_oos:  12.682292527145286
```

```
In [73]: 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 [74]:
```
performance_factor = np.mean(ir_equal_oos)/np.mean(ir_sm_oos)
print('OOS Equal / Markowitz: ', performance_factor)
```

```
OOS Equal / Markowitz:  1.2948545494807482
```

In [75]:
```
performance_factor = np.mean(ir_hier_oos)/np.mean(ir_sm_oos)
print('OOS Hierarchical / Markowitz: ', performance_factor)
```

```
OOS Hierarchical / Markowitz:  2.7050687782142218
```

```
In [76]: performance_factor = np.mean(ir_hier_oos)/np.mean(ir_equal_oos)
         print('OOS Hierarchical / Equal: ', performance_factor)
```

```
OOS Hierarchical / Equal:  2.089090839815936
```

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 [77]: # 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())
```

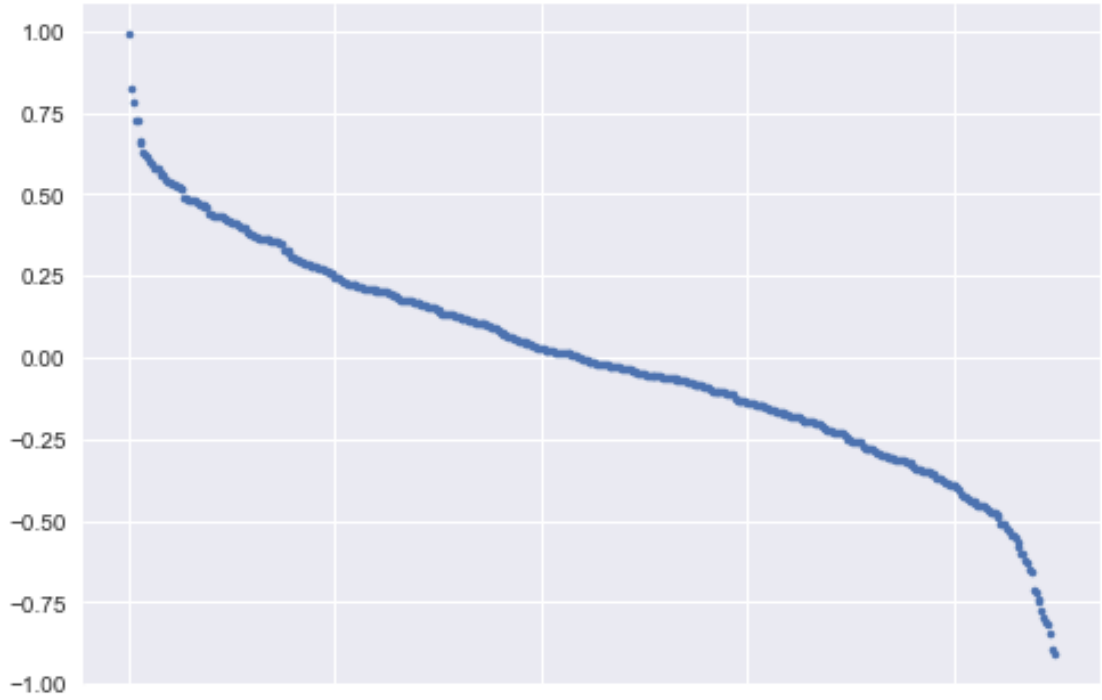
```
Portfolio: 450 Assets, $58.35 Long, $58.35 Short
largest long: manufacturing_stock_4 0.9911231850379247
largest short: manufacturing_stock_28 -0.9068529604953223
sector_net_exposures:
            position_value
sector_id
aut              -0.039674
ban              -0.130852
bio               0.162993
ins               0.509072
man               0.469503
oil               0.011216
pha              -1.259297
ret               0.124053
tec               0.152986
```

```
In [78]: print(pf.portfolio_df.head())
```

```
                      position  price  factor_value sector_id  position_value
manufacturing_stock_4  0.991123      1           1.0       man        0.991123
bio_stock_39           0.824076      1           1.0       bio        0.824076
bio_stock_20           0.782829      1           1.0       bio        0.782829
pharma_stock_14        0.727880      1           1.0       pha        0.727880
pharma_stock_38        0.727615      1           1.0       pha        0.727615
```

```
In [79]: pf.portfolio_df.position_value.plot(style='.')
```

```
Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x1a17fc79e8>
```



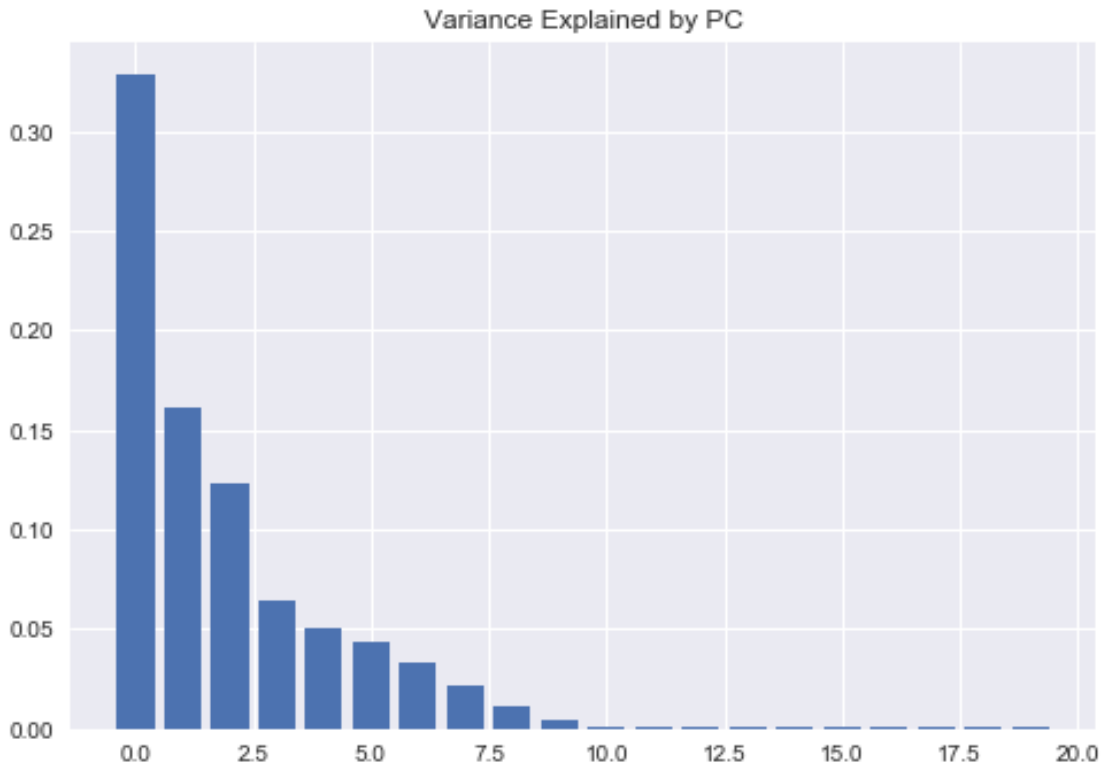
1.2 Principal Component Analysis

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 [80]: 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 [81]:
```
# denoise covariance matrix
cov = stocks_all.cov()
cov_mean = cov.values.mean()
print(cov_mean)
n = len(industries)+1

evalues, Q = np.linalg.eig(cov-cov_mean)
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)+cov_mean
```

```
7.283488201566099e-05
```

In [82]:
```
def cov_dist(corr0, corr1):
    num = np.trace(np.dot(corr0, corr1))
    den = np.linalg.norm(corr0, ord='fro')
    den *= np.linalg.norm(corr1, ord='fro')
    cmd = 1 - num / den
    return cmd
```

In [83]:
```
cov_dist(cov, cov_truth)
```

Out[83]: 0.0028991081404758923

In [84]:
```
cov_dist(pc_cov, cov_truth)
```

Out[84]: 0.01918395707307663

In [85]:
```
# pc_cov.shape
# np.linalg.matrix_rank(pc_cov)
# np.linalg.cond(pc_cov)
# np.linalg.cond(cov)
```

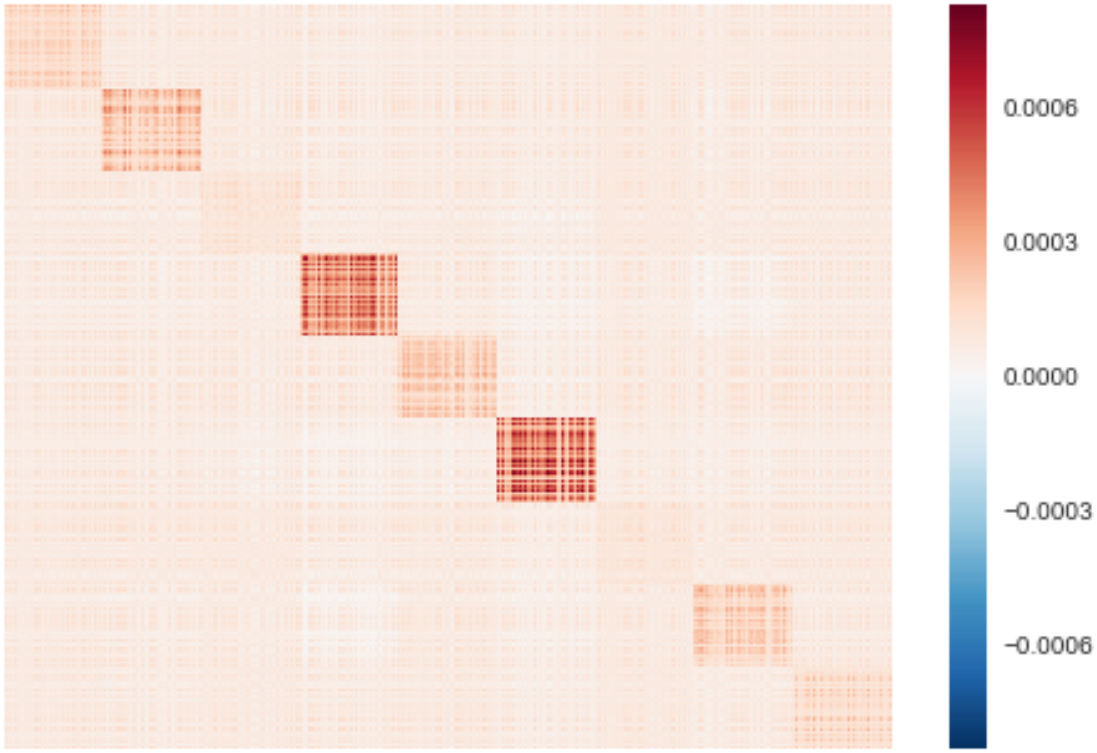
In [86]:
```
# weights_closed_form = (np.linalg.inv(pc_cov).dot(expected_returns_all)/
#
np.sum(np.abs(np.linalg.inv(pc_cov).dot(expected_returns_all))))
```

In [87]:
```
# pf = Portfolio(assets=weights.keys(),
#                position=weights_closed_form,
#                price=[1]*len(weights.keys()),
#                sector_id=pd.Series(list(weights.keys())).str[:3].values)
```

In [88]:
```
# pf.portfolio_df.position_value.plot(style='.')
```

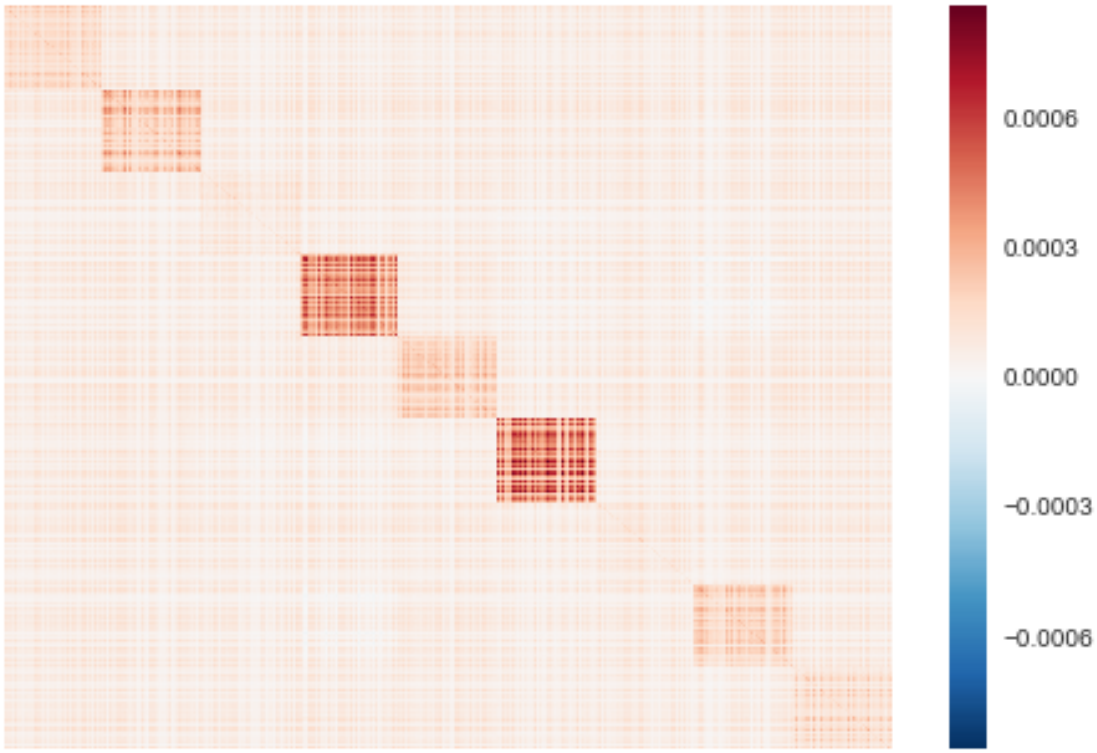
In [89]:
```
import seaborn as sns
sns.heatmap(pc_cov, xticklabels=False, yticklabels=False)
```

Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c445e80>



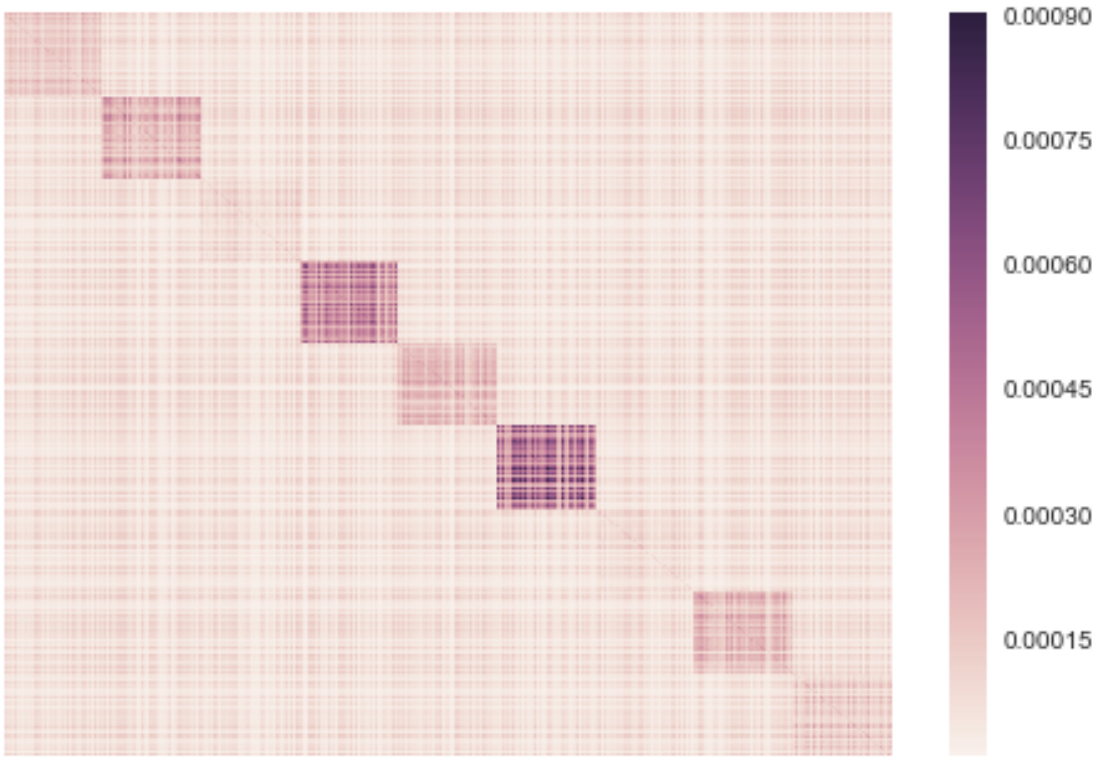
In [90]:
```
# plot sample covariance matrix to compare
sns.heatmap(cov, xticklabels=False, yticklabels=False)
```

Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c61ac88>




```
In [91]: # 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[91]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c5c76a0>
```



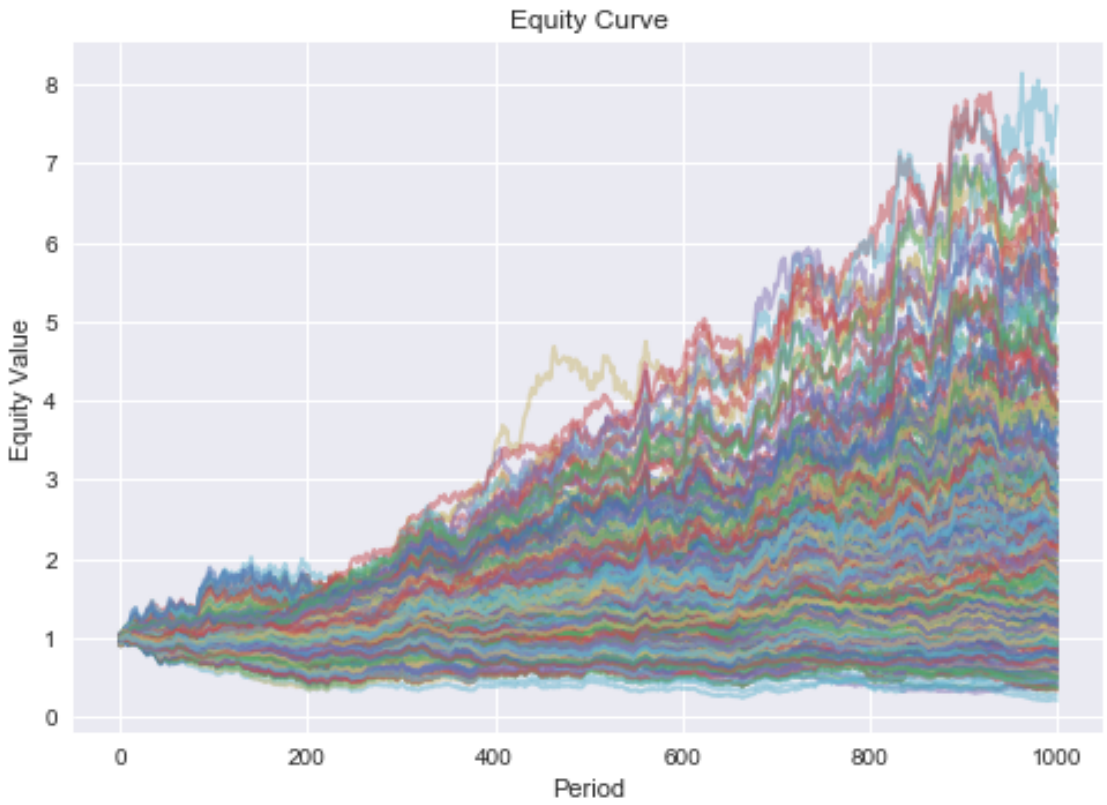
The heatmaps above look almost indistinguishable after eliminating all other components.

```
In [59]: 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)
```

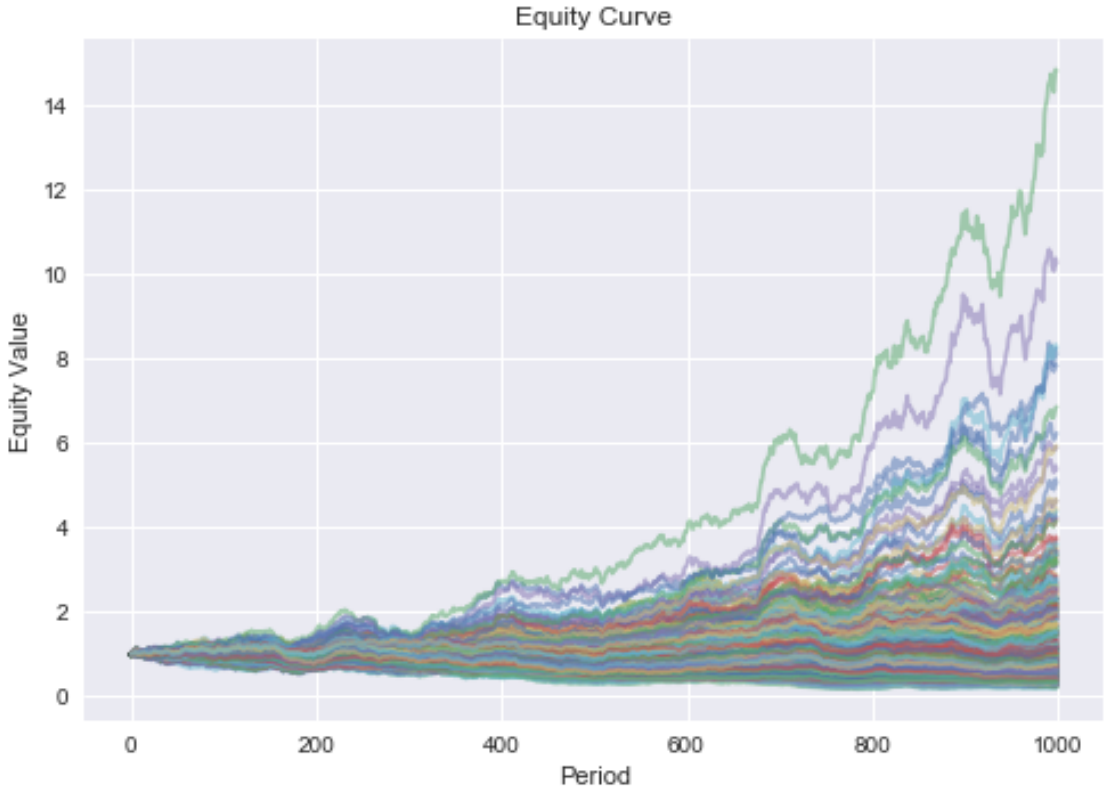
```
# PCA
n = len(industries)+1
cov_mean = cov.values.mean()
pc_cov = optimize.pc_cov(cov-cov_mean, n) + cov_mean

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
```

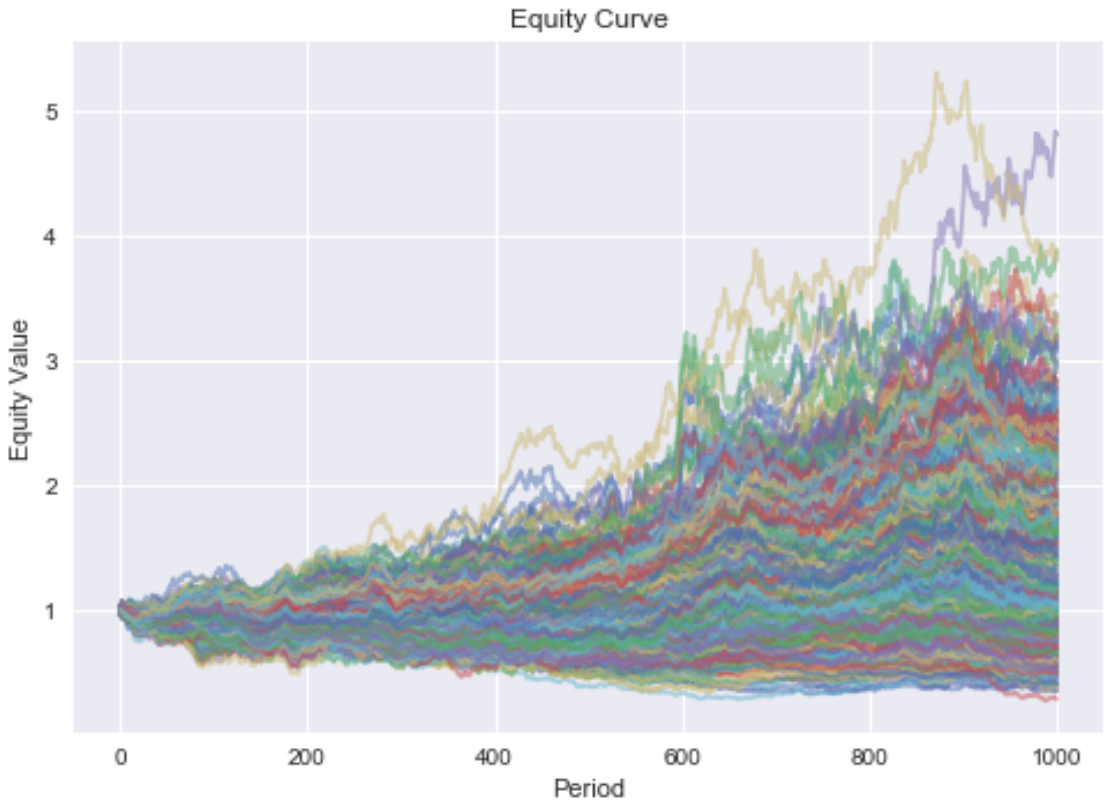



```
IR gt_is:  22.327779670589262
IR gt_oos:  23.47178430290095
```

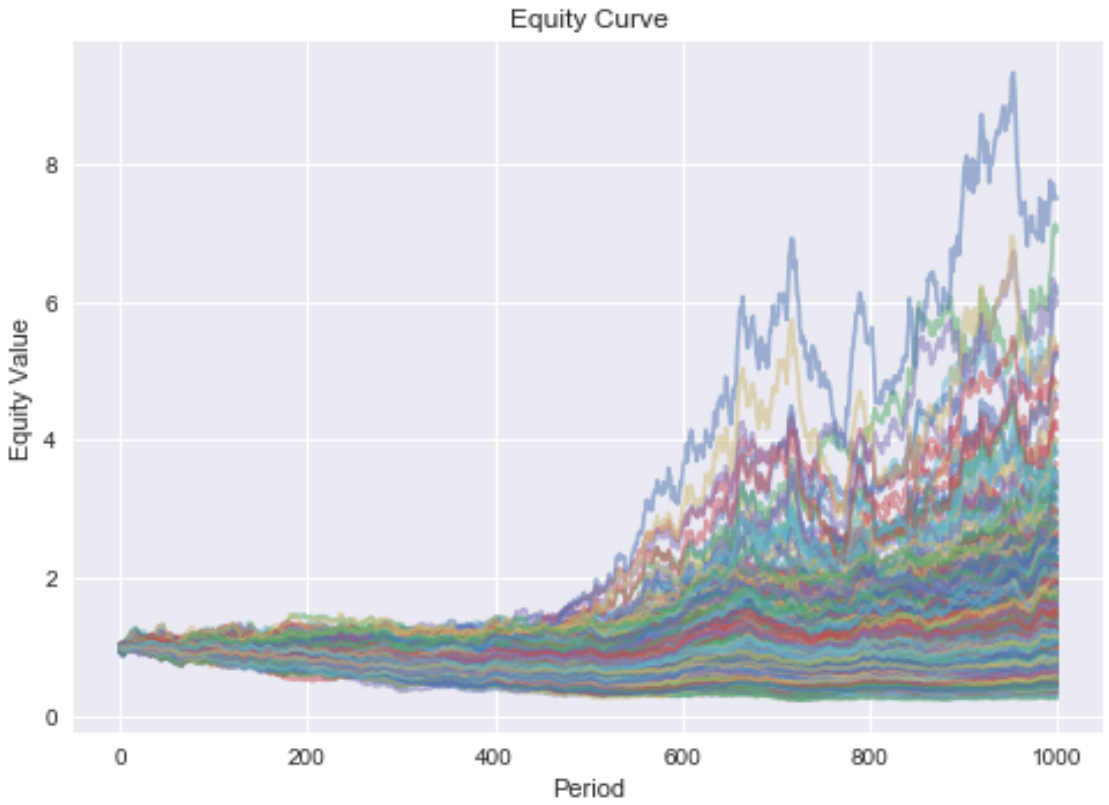
```
/Users/jan/Documents/PoCon/src/optimize.py:30: UserWarning: Optimizer did not
converge.
  warnings.warn("Optimizer did not converge.")


IR sm_is:  53.802340063882774
IR sm_oos:  7.055504320705678
IR pca_is:  21.347357187366764
IR pca_oos:  22.13523148734668
MC run:  1
```



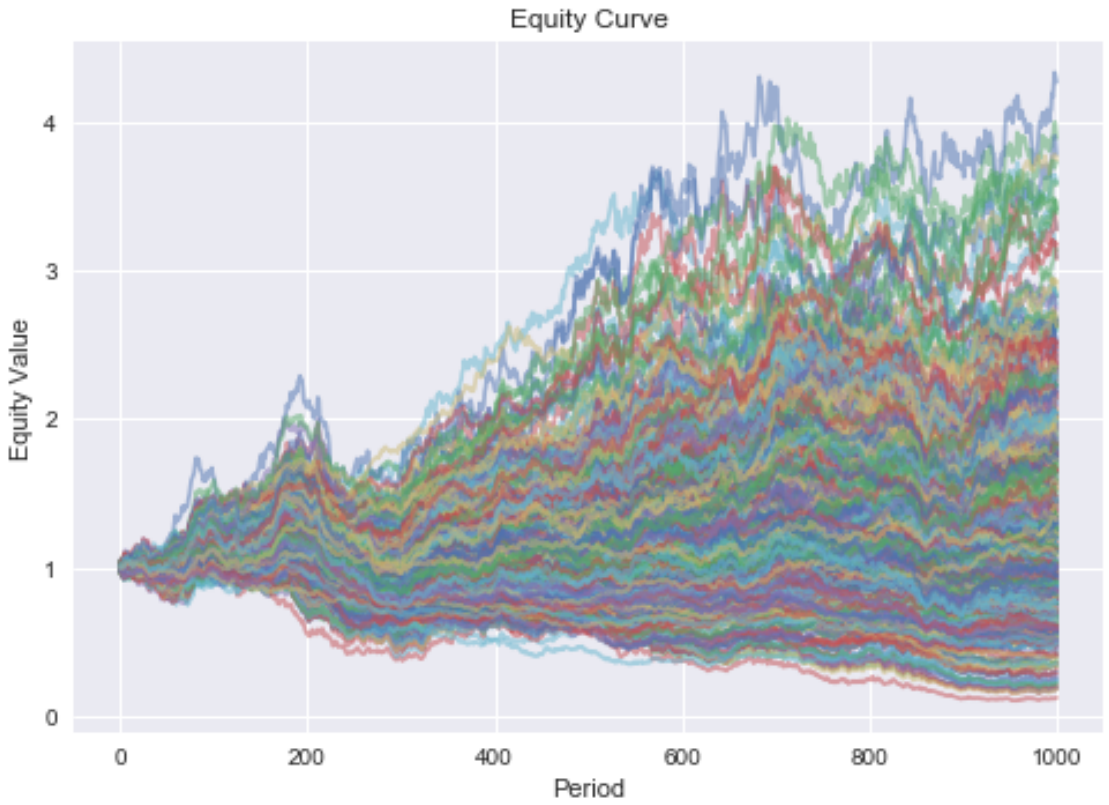

```
IR gt_is:  25.101144809904547
IR gt_oos:  24.853227124511694
IR sm_is:  70.82443442413046
IR sm_oos:  8.239623585708864
IR pca_is:  23.49806542609442
IR pca_oos:  23.60890179698081
MC run:  2
```



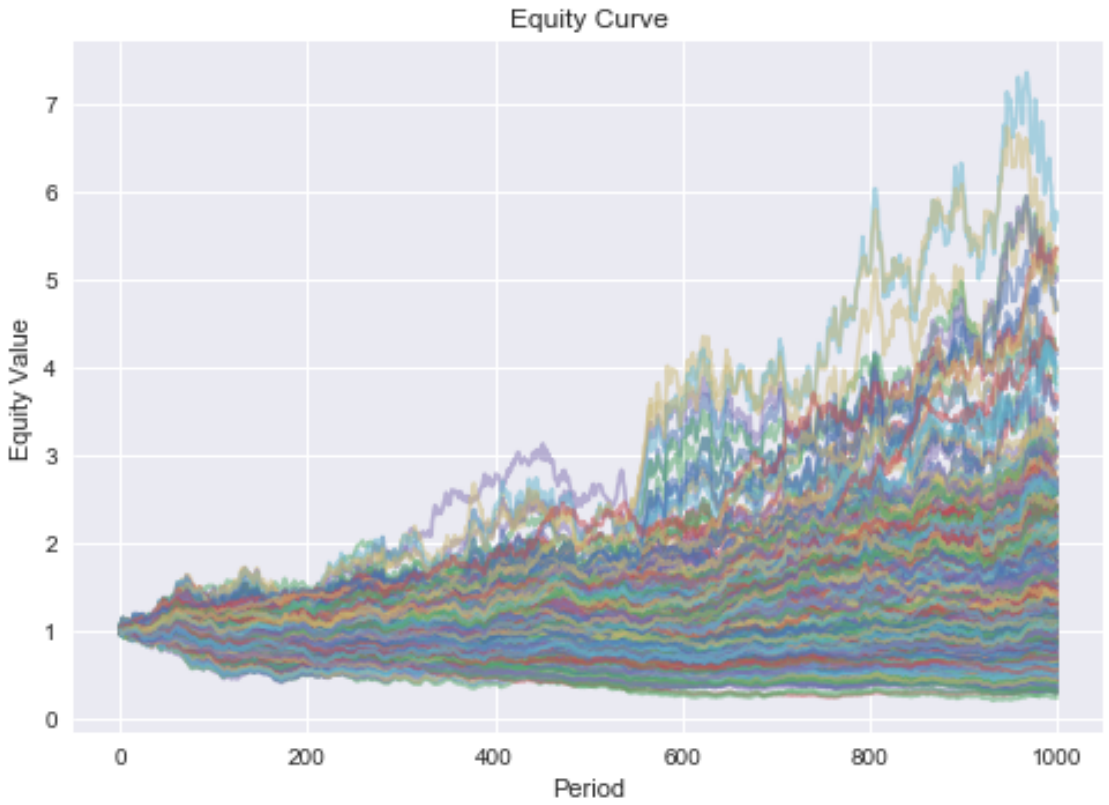

```
IR gt_is:  23.42049879805346
IR gt_oos:  24.661784287006622
IR sm_is:  78.4440836534776
IR sm_oos:  9.070363489612662
IR pca_is:  22.359102803798066
IR pca_oos:  22.732557900888644
MC run:  3
```



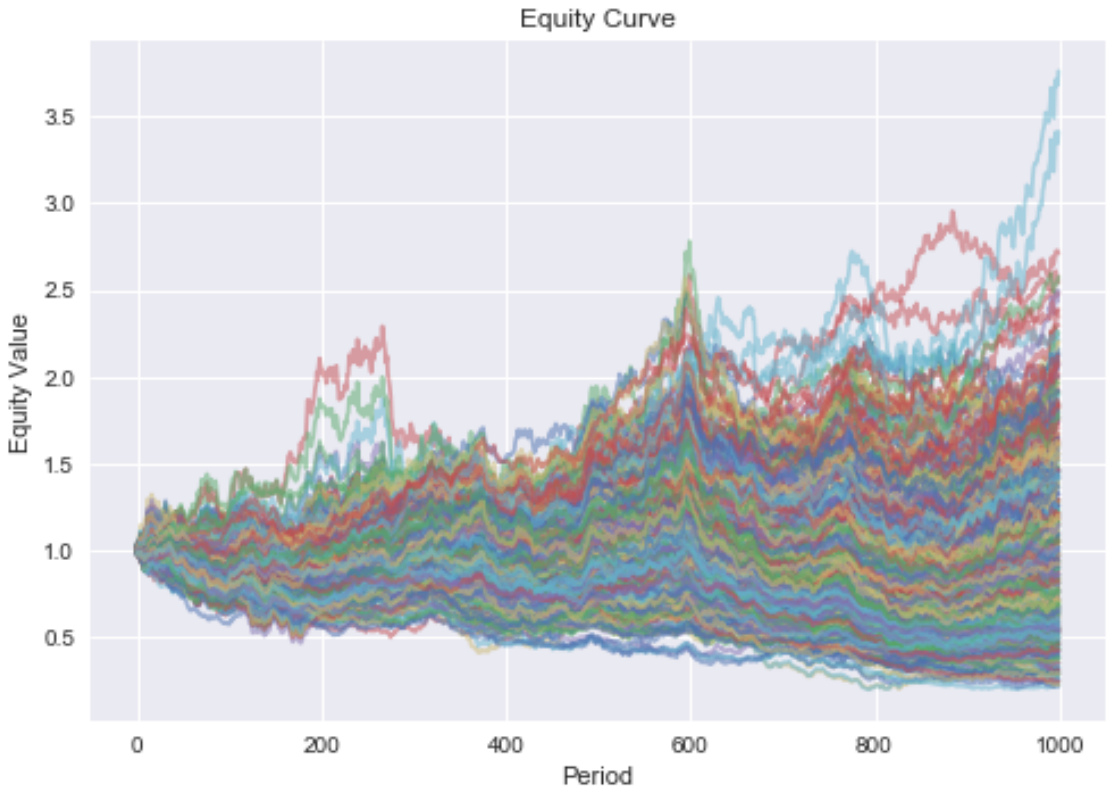

```
IR gt_is:  24.29064031726839
IR gt_oos:  23.527850344912522
IR sm_is:  99.12060509317666
IR sm_oos:  6.274097276225852
IR pca_is:  23.41472636652444
IR pca_oos:  21.816446100957062
MC run:  4
```



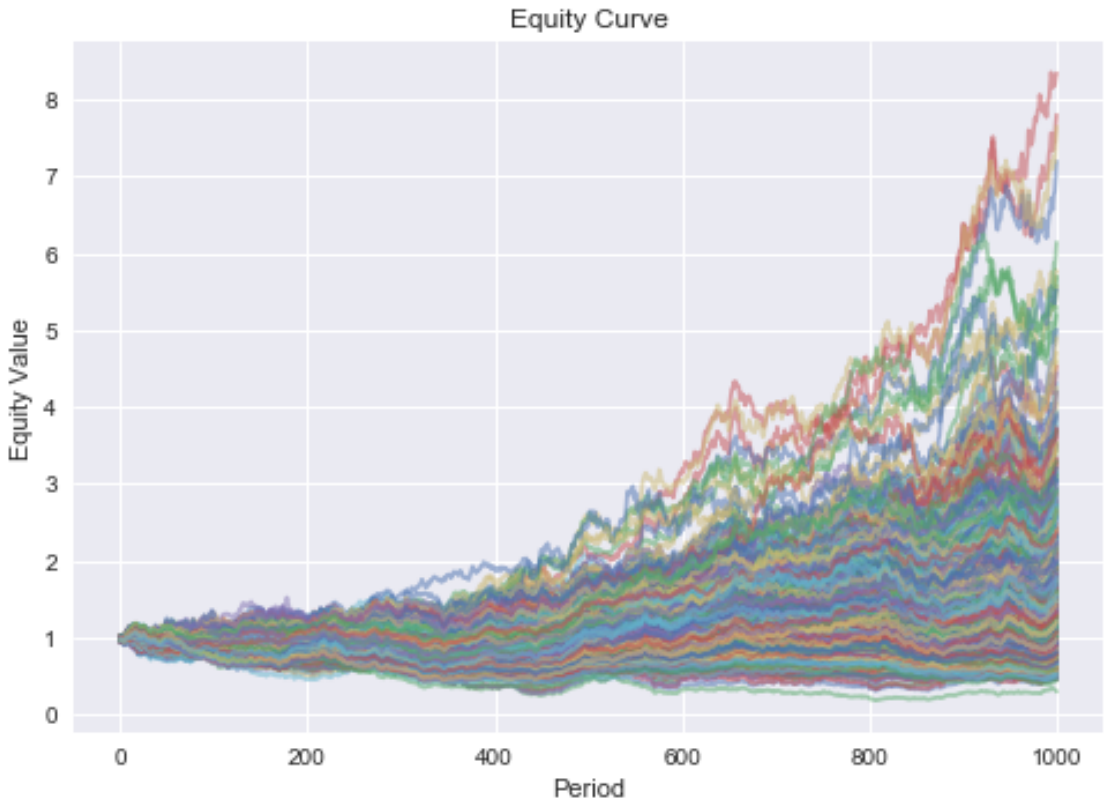

```
IR gt_is:  23.937900532009124
IR gt_oos:  26.454560476837617
IR sm_is:  84.48418297695724
IR sm_oos:  6.850278747764034
IR pca_is:  22.408453458941437
IR pca_oos:  24.964997963748075
MC run:  5
```



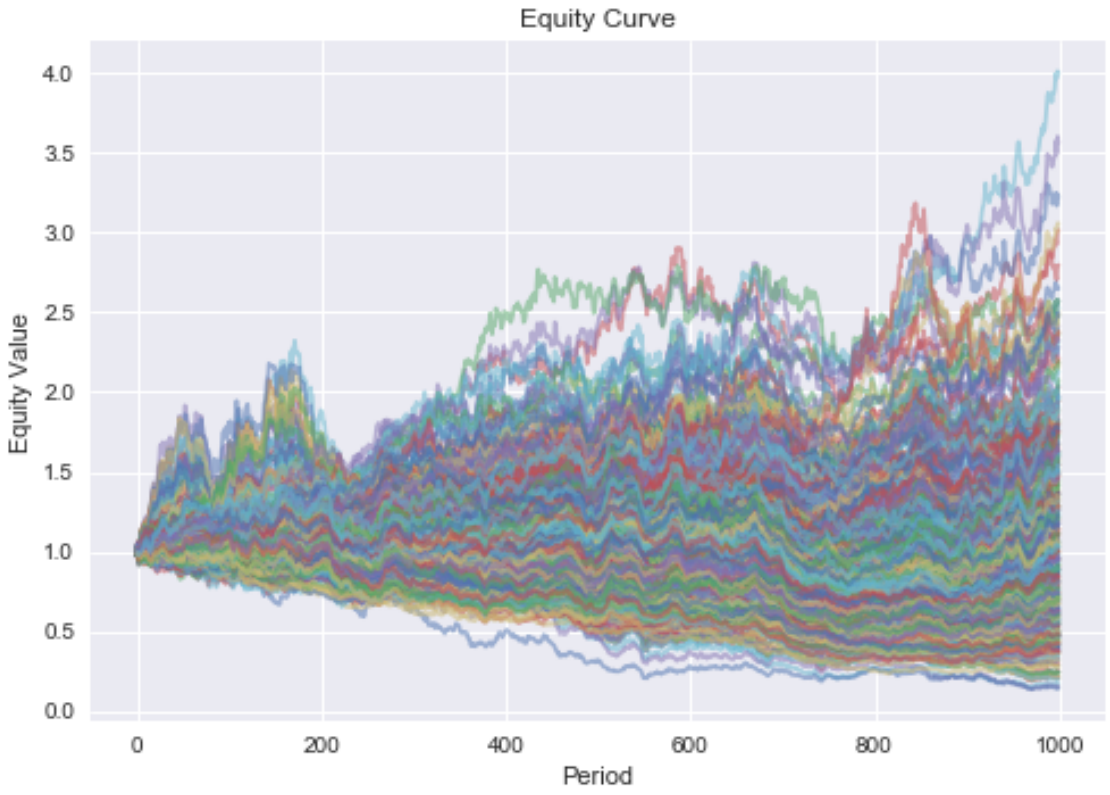

```
IR gt_is:  23.92122605927581
IR gt_oos:  22.43413617839009
IR sm_is:  86.6049788733028
IR sm_oos:  6.929143230389251
IR pca_is:  22.123143020121717
IR pca_oos:  20.20262501817184
MC run:  6
```



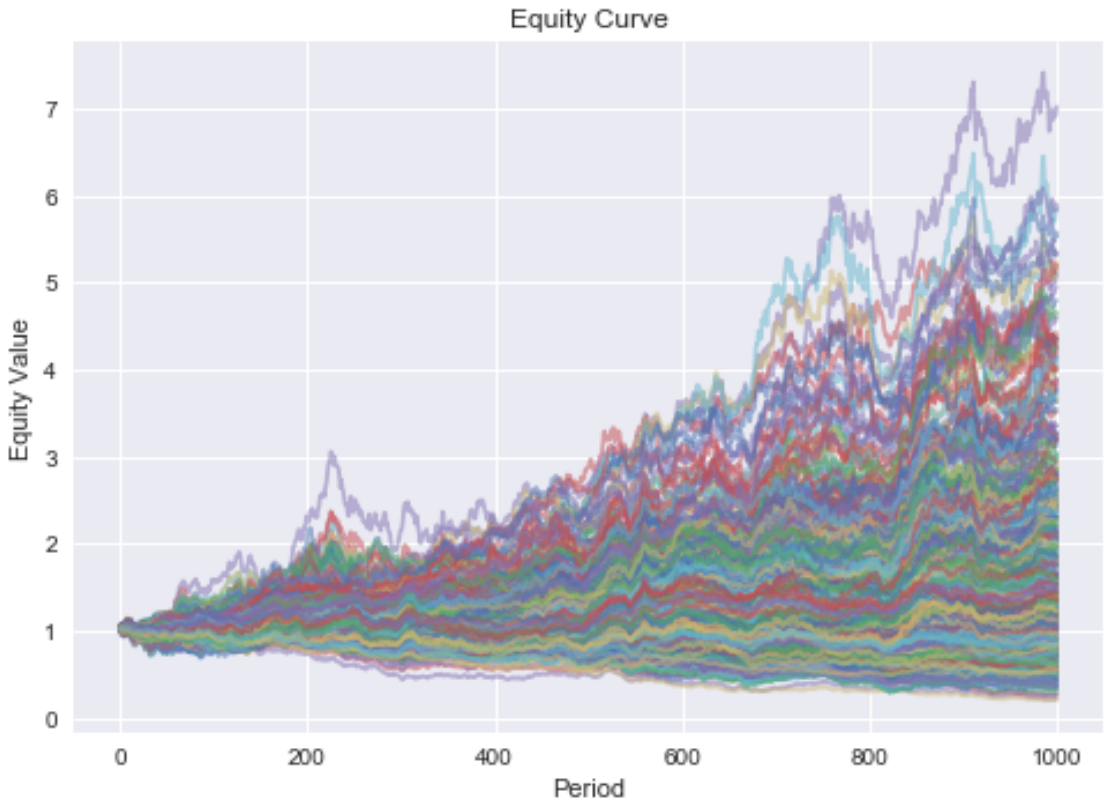

```
IR gt_is:  22.983269270061818
IR gt_oos:  23.82994484706046
IR sm_is:  60.60154130511508
IR sm_oos:  7.508591171194288
IR pca_is:  20.766612623104535
IR pca_oos:  21.861266650928304
MC run:  7
```




```
IR gt_is:  22.611953528427886
IR gt_oos:  23.331746728303457
IR sm_is:  69.96084806741126
IR sm_oos:  8.863754443027098
IR pca_is:  21.036102862175238
IR pca_oos:  21.872112196223508
MC run:  8
```



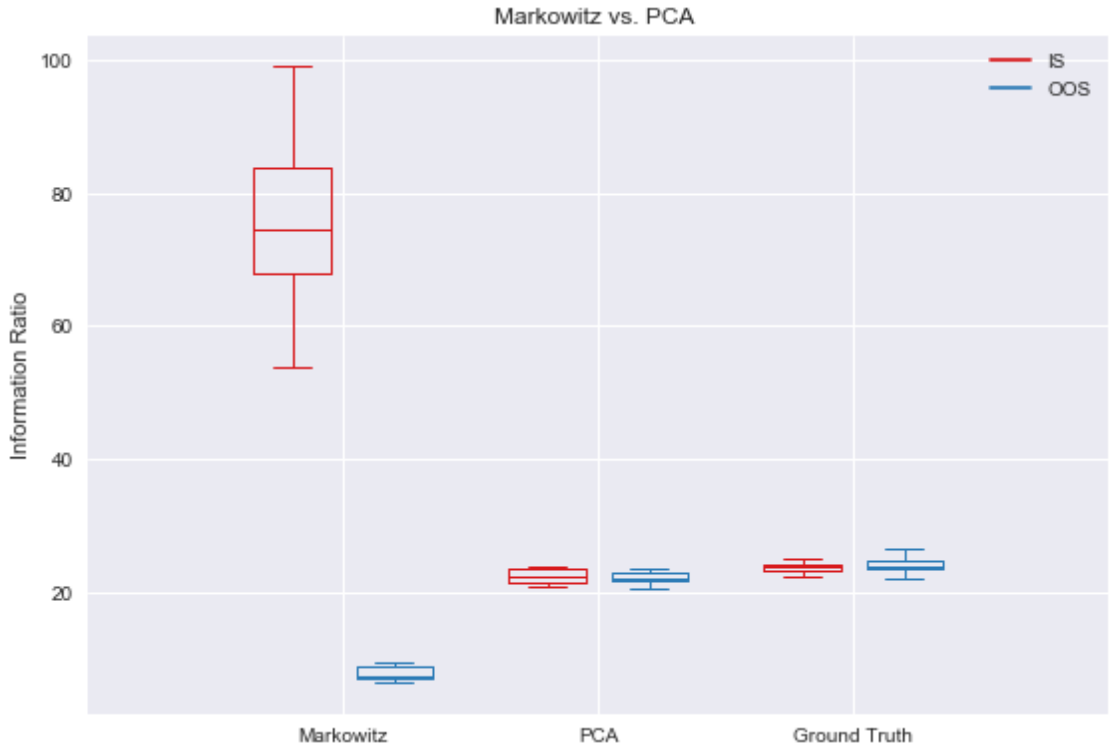

```
IR gt_is:  24.12227104667446
IR gt_oos:  24.79061749922558
IR sm_is:  67.26436300927361
IR sm_oos:  9.38671204728488
IR pca_is:  23.304190501672466
IR pca_oos:  22.913317626654727
MC run:  9
```




```
IR gt_is:  24.18494912078142
IR gt_oos:  21.894938802777837
IR sm_is:  82.53623112925658
IR sm_oos:  7.006296765579034
IR pca_is:  23.680358744003545
IR pca_oos:  20.42342839489186
```

```
In [60]: 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 [61]:
```
performance_factor = np.mean(ir_pca_oos)/np.mean(ir_sm_oos)
print('OOS PCA / Markowitz: ', performance_factor)
```

```
OOS PCA / Markowitz:  2.883108320103103
```

In [62]:
```
print(np.linalg.matrix_rank(cov))
print(np.linalg.matrix_rank(pc_cov))
```

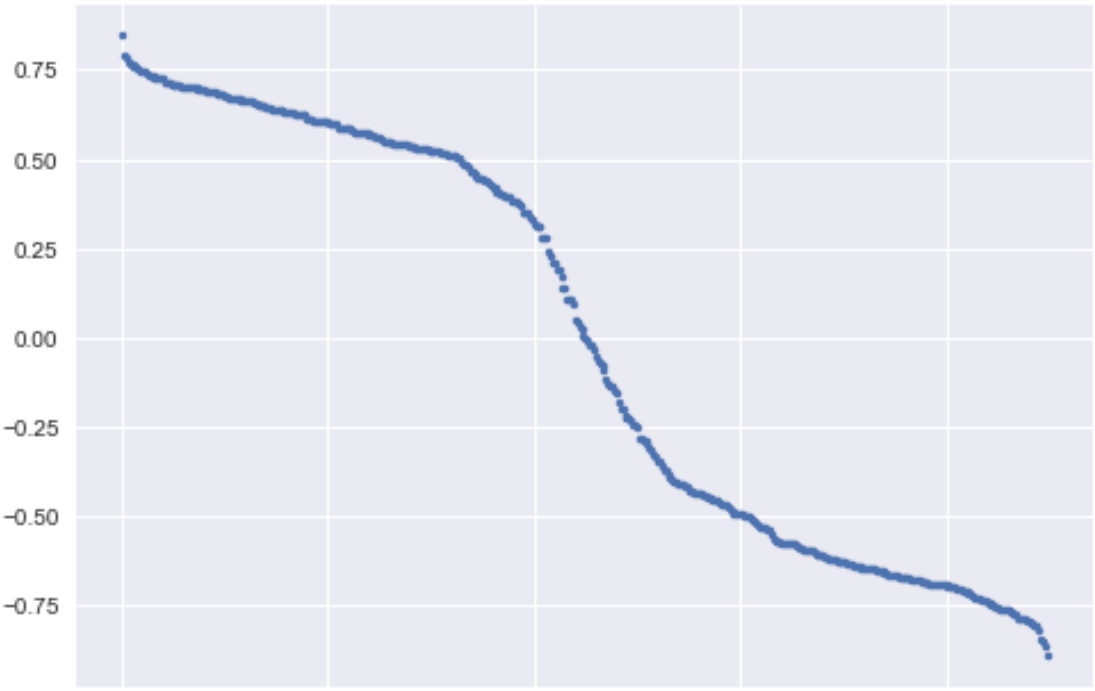
```
450
11
```

In [63]:
```
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, $122.79 Long, $122.79 Short
largest long: insurance_stock_36 0.8462782815485712
largest short: insurance_stock_42 -0.8892452760063864
sector_net_exposures:
            position_value
sector_id
aut              -1.652076
ban               1.216688
bio              -0.806726
ins              -1.310240
man              -1.492269
oil               1.145051
pha               0.029748
ret               1.761821
tec               1.108004
```

```
In [64]: pf.portfolio_df.position_value.plot(style='.')
```

```
Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x1a19daaa58>
```



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:

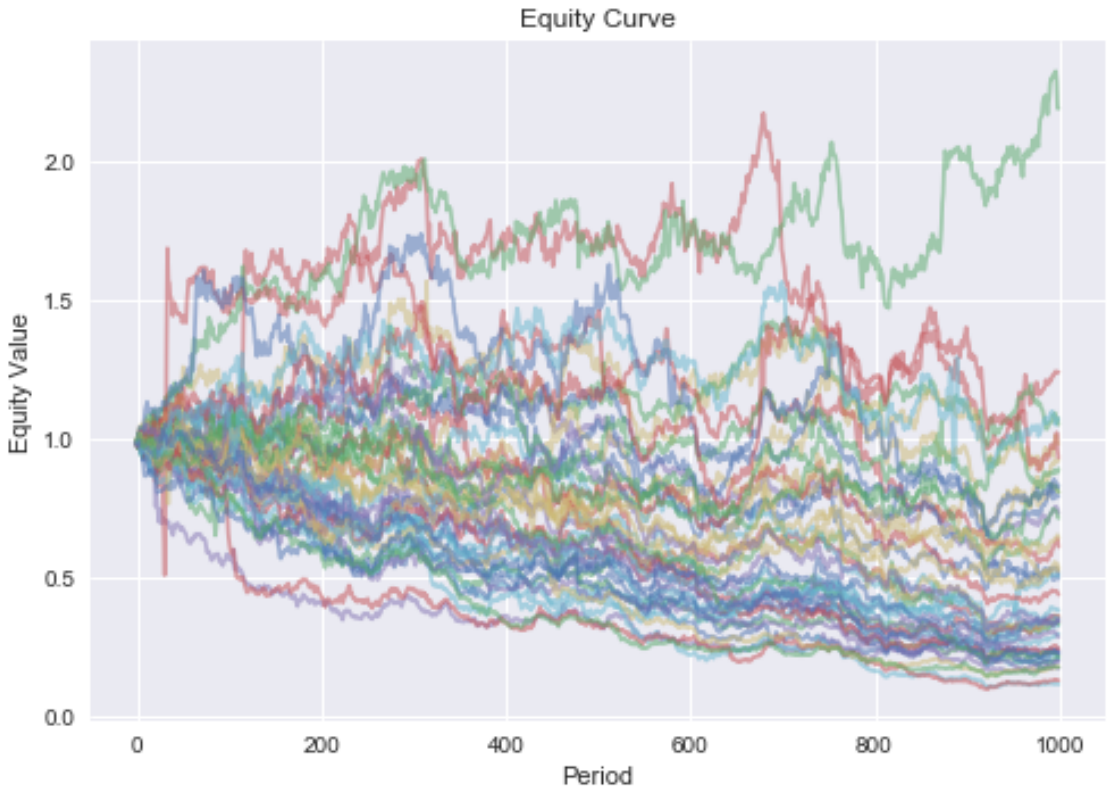
$$\begin{aligned}
\mathrm{E}\left(\sigma_t^2\right) &= \omega + \sum_{i=1}^{q} \alpha_i \mathrm{E}\left(\epsilon_{t-i}^2\right) + \sum_{i=1}^{p} \beta_i \mathrm{E}\left(\sigma_{t-i}^2\right) \\
&= \omega + \sum_{i=1}^{q} \alpha_i \mathrm{E}\left(\eta_{t-i}^2 \sigma_{t-i}^2\right) + \sum_{i=1}^{p} \beta_i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

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 [92]:
```
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.

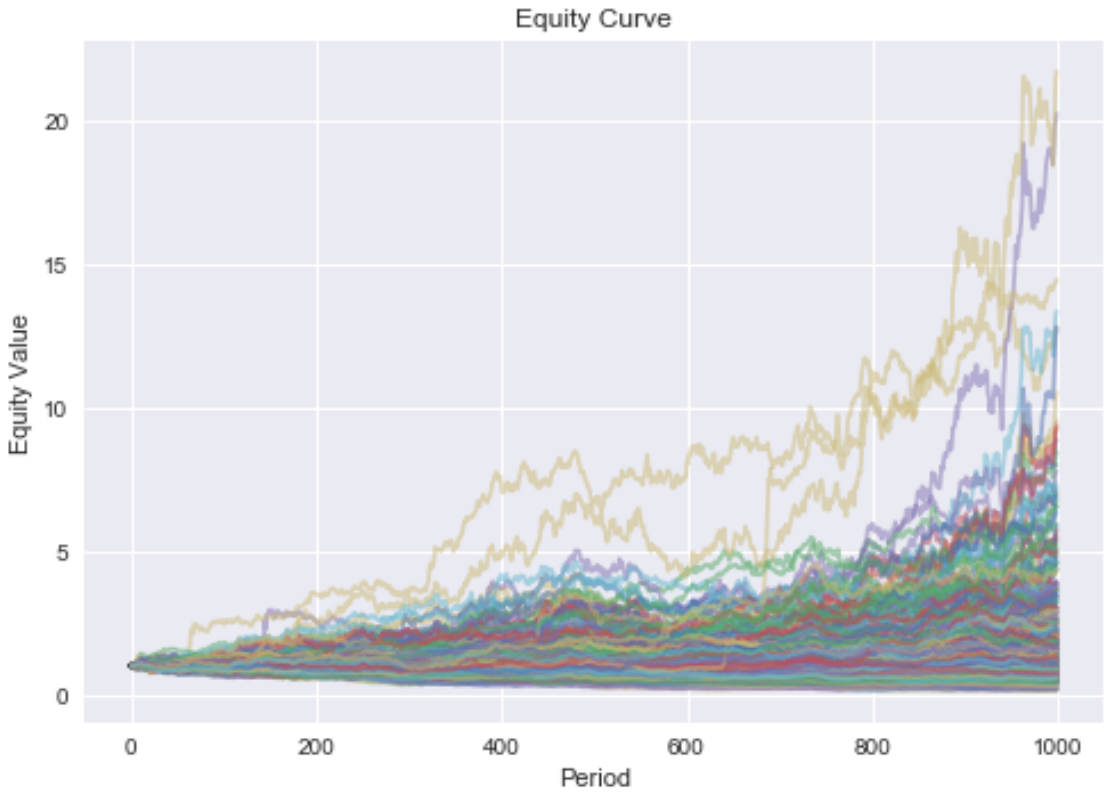
```
In [93]: trainig_pct = 0.5
         n_train = int(trainig_pct*num_samples)
         ir_sm_is = []
         ir_sm_oos = []
         ir_hier_is = []
         ir_hier_oos = []
         ir_gt_is = []
         ir_gt_oos = []
         ir_equal_is = []
         ir_equal_oos = []
         n_stocks = 50

         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]
```

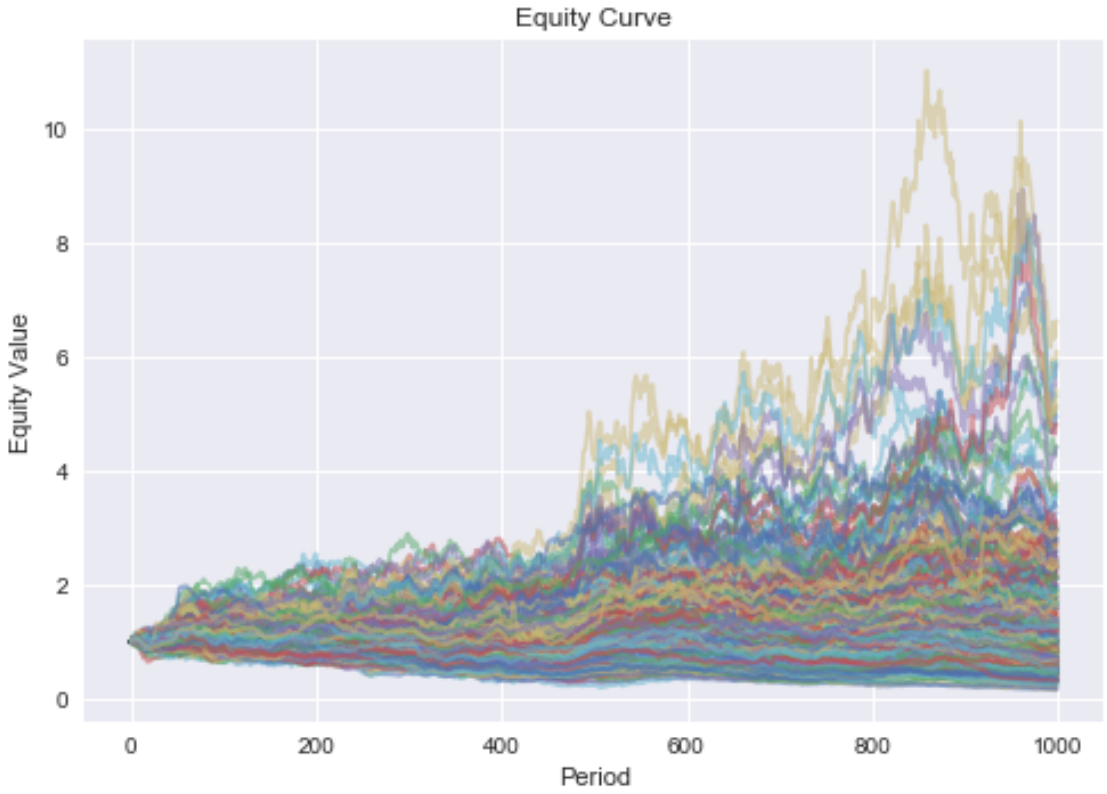
```
            portfolio_equal_oos = portfolio_equal[n_train:]

            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
```

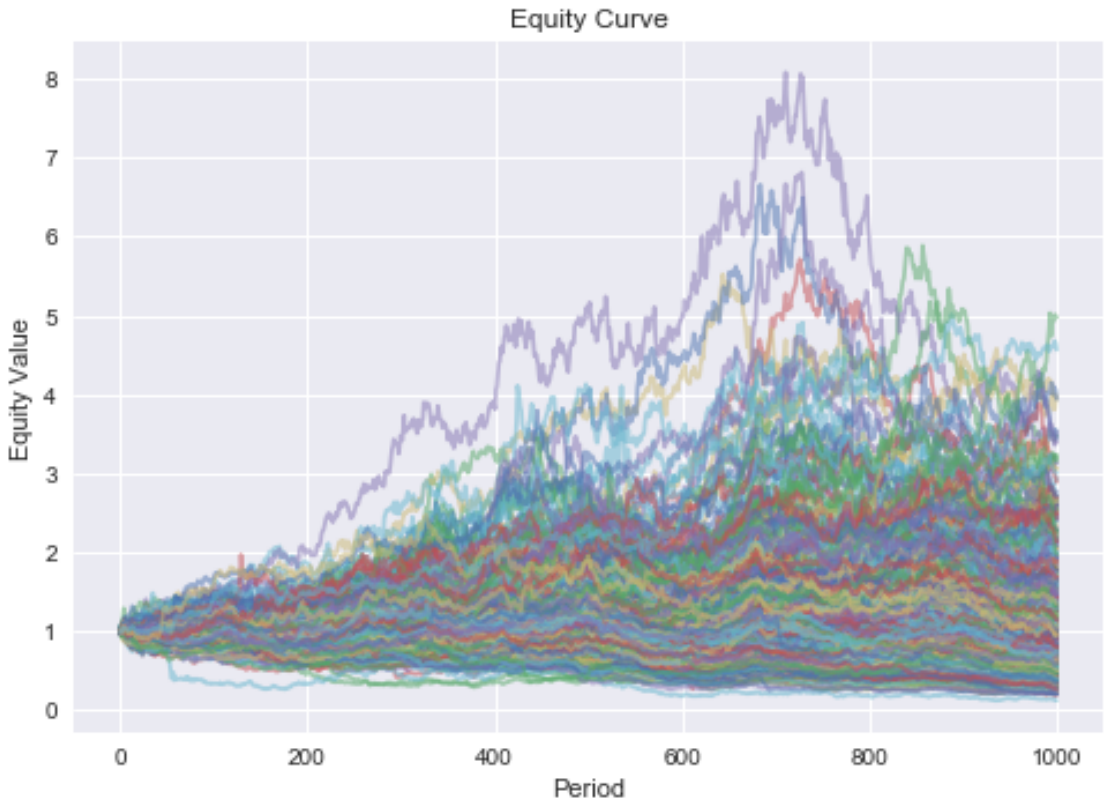



```
IR gt_is:  11.466015963253206
IR gt_oos:  10.545295300471533


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


IR sm_is:  24.56732172672373
IR sm_oos:  2.777713774203299
{'banks': 0.12883088365284823, 'oil': 0.11917174656339015, 'insurance':
0.033652923282206954, 'tech': 0.0698561496377195, 'bio': 0.049429979240176614,
'pharma': 0.2902182633232347, 'auto': 0.1030725603868126, 'retail':
```

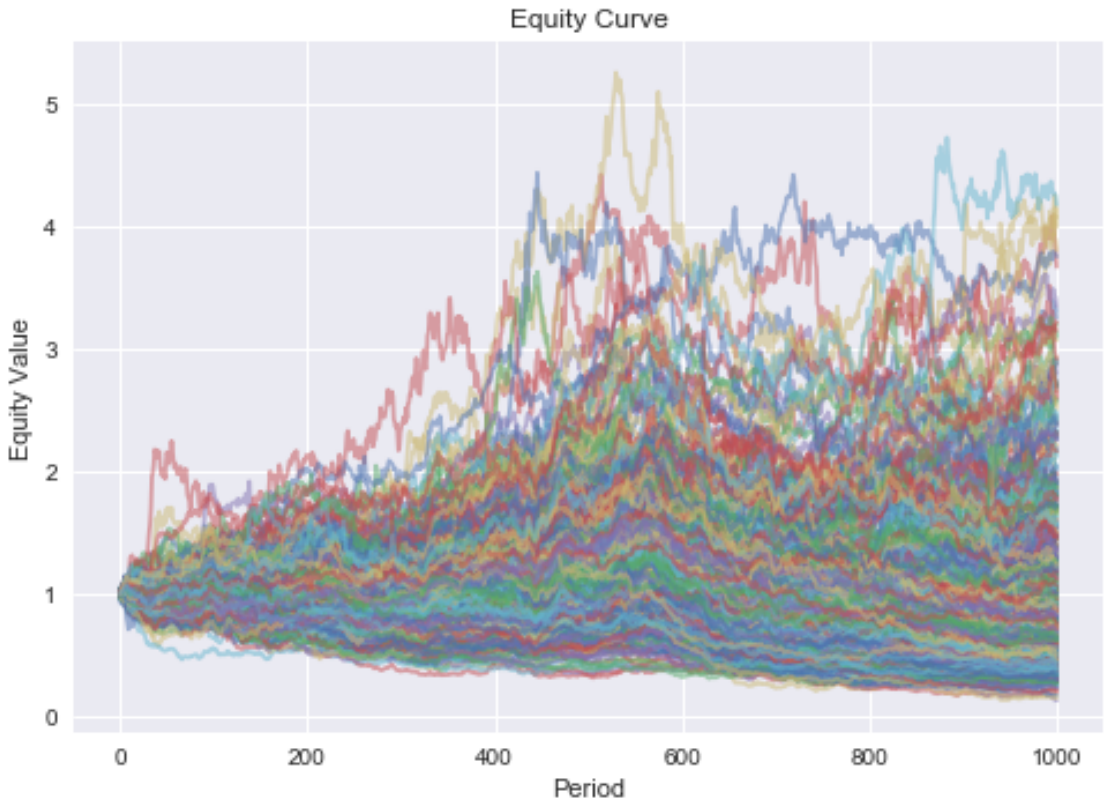
```
0.16527133112132164, 'manufacturing': 0.040496162792289465}
IR hier_is:  11.340827518178601
IR hier_oos:  9.815197947097037
IR equal_is:  7.294826260330567
IR equal_oos:  6.759005076451435
MC run:  1
```



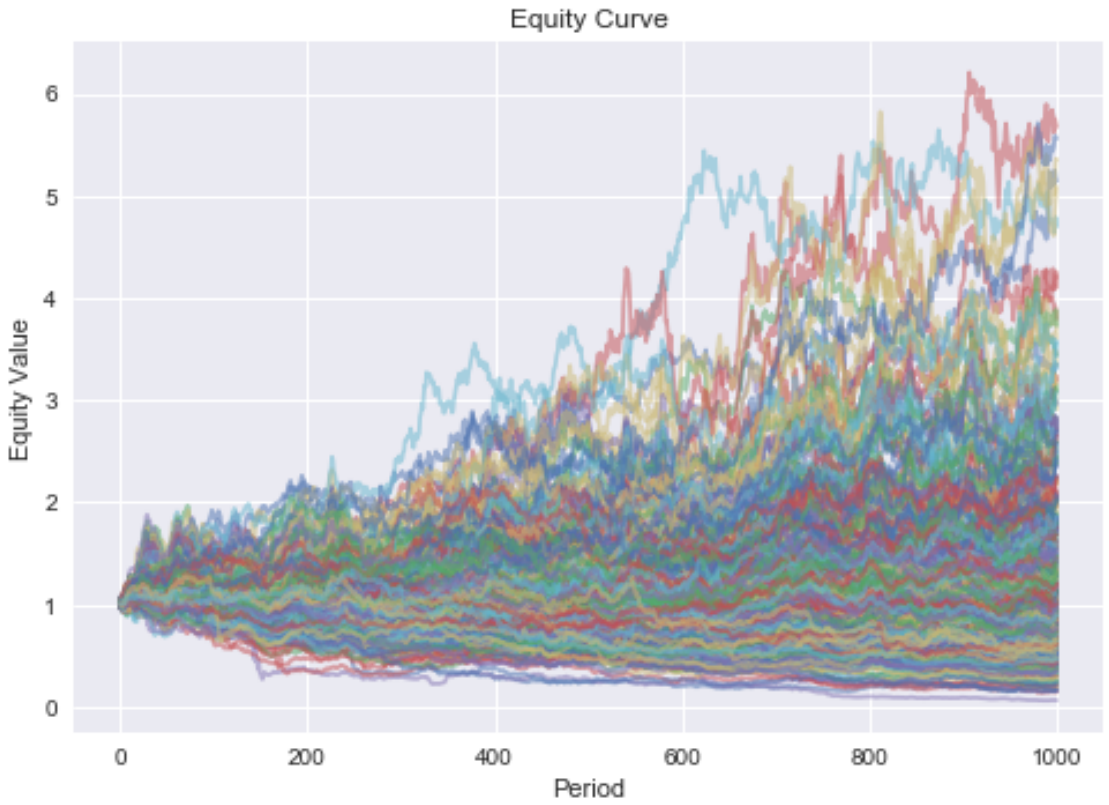

```
IR gt_is:  11.252591875374707
IR gt_oos:  11.669427244007899
IR sm_is:  36.72283174655992
IR sm_oos:  4.89030429598662
{'banks': 0.0752838506060012, 'oil': 0.11585004633765635, 'insurance':
0.04797055434153544, 'tech': 0.03225225048094753, 'bio': 0.17473729230272128,
'pharma': 0.30948456399931046, 'auto': 0.0947240793977233, 'retail':
0.08054945871449579, 'manufacturing': 0.0691479038196087}
IR hier_is:  11.583914936413338
IR hier_oos:  9.845786909714171
IR equal_is:  6.084729740614245
IR equal_oos:  6.218050407789197
MC run:  2
```



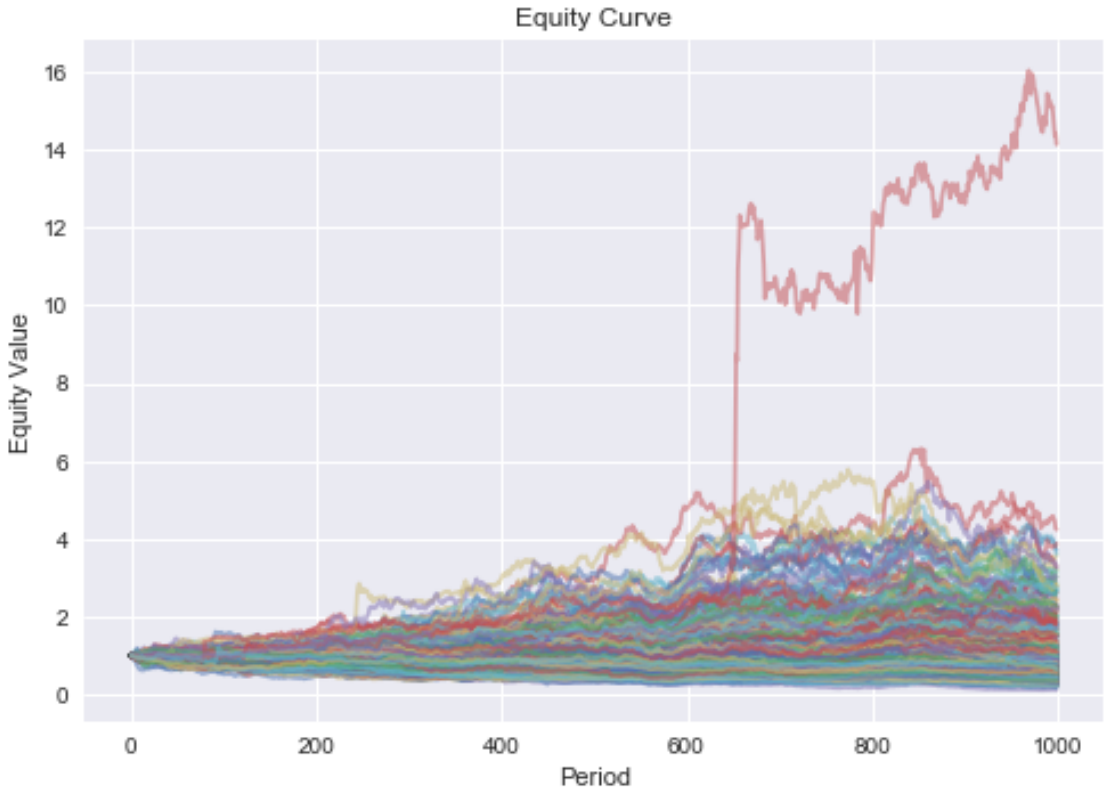

```
IR gt_is:  10.40325557232364
IR gt_oos:  12.472948957872351
IR sm_is:  45.41754500831745
IR sm_oos:  3.800358842959666
{'banks': 0.11815631590124819, 'oil': 0.08370554925019712, 'insurance':
0.11308255686500722, 'tech': 0.15167825368747573, 'bio': 0.12659824513747908,
'pharma': 0.17863859519430603, 'auto': 0.03299228716172744, 'retail':
0.10605798486946974, 'manufacturing': 0.08909021193308952}
IR hier_is:  11.586992489408901
IR hier_oos:  11.118146041996376
IR equal_is:  6.863073296689059
IR equal_oos:  7.22453563239465
MC run:  3
```



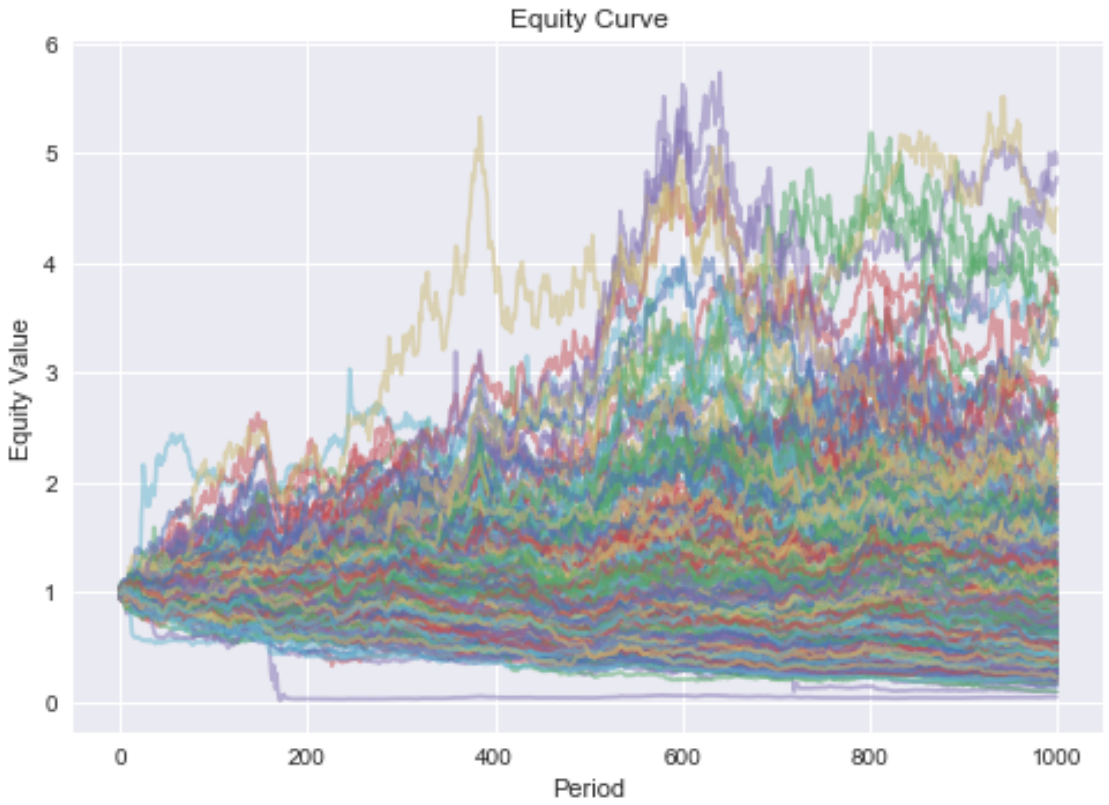

```
IR gt_is:  11.914482249249051
IR gt_oos:  10.279260770415764
IR sm_is:  18.577489241533414
IR sm_oos:  2.660319253576504
{'banks': 0.13369031920961644, 'oil': 0.1535017321466058, 'insurance':
0.1314658042729921, 'tech': 0.08370134017040906, 'bio': 0.1254636404813916, 'pharma':
0.17633965875469454, 'auto': 0.078462877653937, 'retail': 0.0380861198697822,
'manufacturing': 0.07928850744057132}
IR hier_is:  12.743153352923105
IR hier_oos:  9.534777747024371
IR equal_is:  8.5598487004829
IR equal_oos:  8.020518135915918
MC run:  4
```



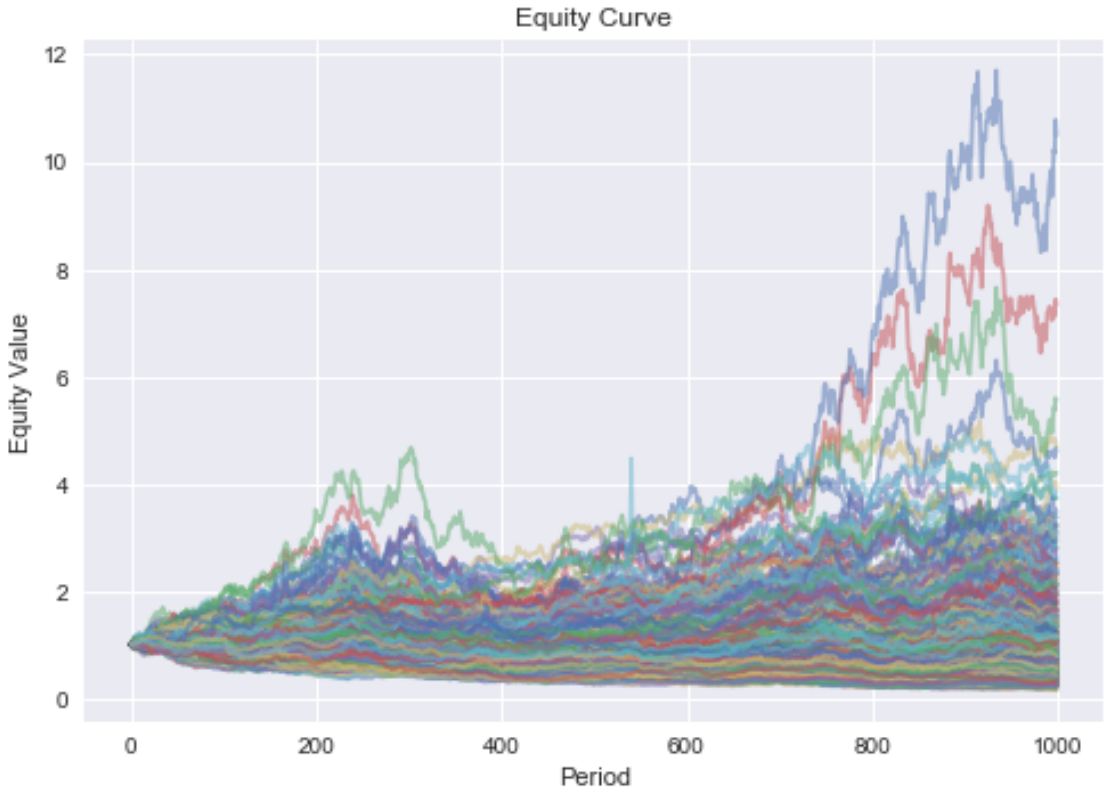

```
IR gt_is:  11.339117885115076
IR gt_oos:  11.311709060622647
IR sm_is:  45.0588656927026
IR sm_oos:  5.270713018481207
{'banks': 0.06588094713182258, 'oil': 0.22018934385961464, 'insurance':
0.12923029737156186, 'tech': 0.10509550875517538, 'bio': 0.09784767227896114,
'pharma': 0.11853222991816907, 'auto': 0.0959249811217277, 'retail':
0.06718957701379533, 'manufacturing': 0.10010944254917235}
IR hier_is:  12.85640042054815
IR hier_oos:  11.175390305398258
IR equal_is:  6.919100827280759
IR equal_oos:  6.852161826727482
MC run:  5
```



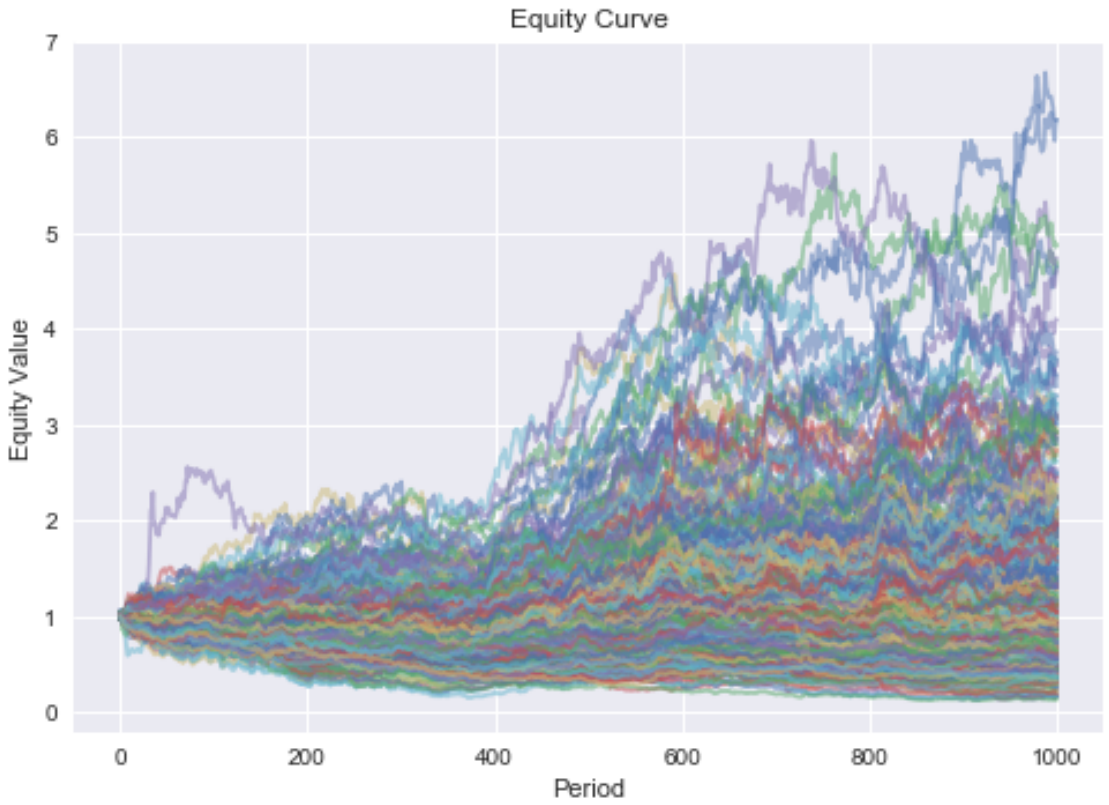

```
IR gt_is:  11.405258732751918
IR gt_oos:  9.562670584881868
IR sm_is:  34.28347292503075
IR sm_oos:  1.6352374198549882
{'banks': 0.1592642123670098, 'oil': 0.04348239450529035, 'insurance':
0.07778101358597794, 'tech': 0.04230872044357792, 'bio': 0.11530466386977967,
'pharma': 0.09004074351012842, 'auto': 0.12504580373783022, 'retail':
0.2633283854318897, 'manufacturing': 0.08344406254851593}
IR hier_is:  11.79144120798872
IR hier_oos:  8.245909993706013
IR equal_is:  8.23171016886631
IR equal_oos:  7.377670865175384
MC run:  6
```



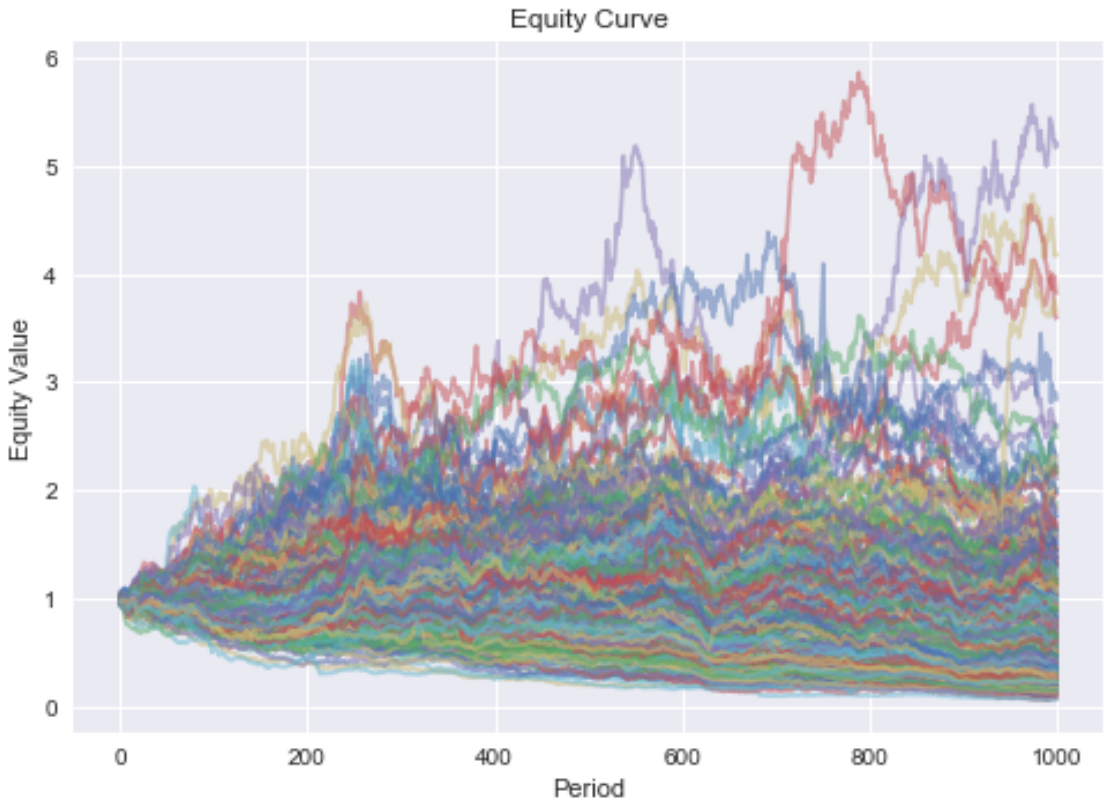

```
IR gt_is:  11.584822309011468
IR gt_oos:  9.138935714298743
IR sm_is:  46.22759397906164
IR sm_oos:  2.744565118961362
{'banks': 0.04695412567921421, 'oil': 0.1461744624836076, 'insurance':
0.010303704022059027, 'tech': 0.23318794452846045, 'bio': 0.07246069640823397,
'pharma': 0.27296406042569415, 'auto': 0.047367577978163725, 'retail':
0.08936213524059948, 'manufacturing': 0.0812252932339673}
IR hier_is:  10.571616994327336
IR hier_oos:  7.9060383816128
IR equal_is:  6.868882816870544
IR equal_oos:  5.829300434216682
MC run:  7
```




```
IR gt_is:  10.942575180774726
IR gt_oos:  11.391487454355012
IR sm_is:  34.323311152491726
IR sm_oos:  3.589991320093684
{'banks': 0.08095657956564971, 'oil': 0.07470414646485106, 'insurance':
0.060873775585545326, 'tech': 0.3155043418134584, 'bio': 0.04315509257101283,
'pharma': 0.1904437079254446, 'auto': 0.08752030916424638, 'retail':
0.04627844699220553, 'manufacturing': 0.10056359991758629}
IR hier_is:  11.639362634571908
IR hier_oos:  10.7566443867366
IR equal_is:  7.338450173152181
IR equal_oos:  6.024026816101417
MC run:  8
```



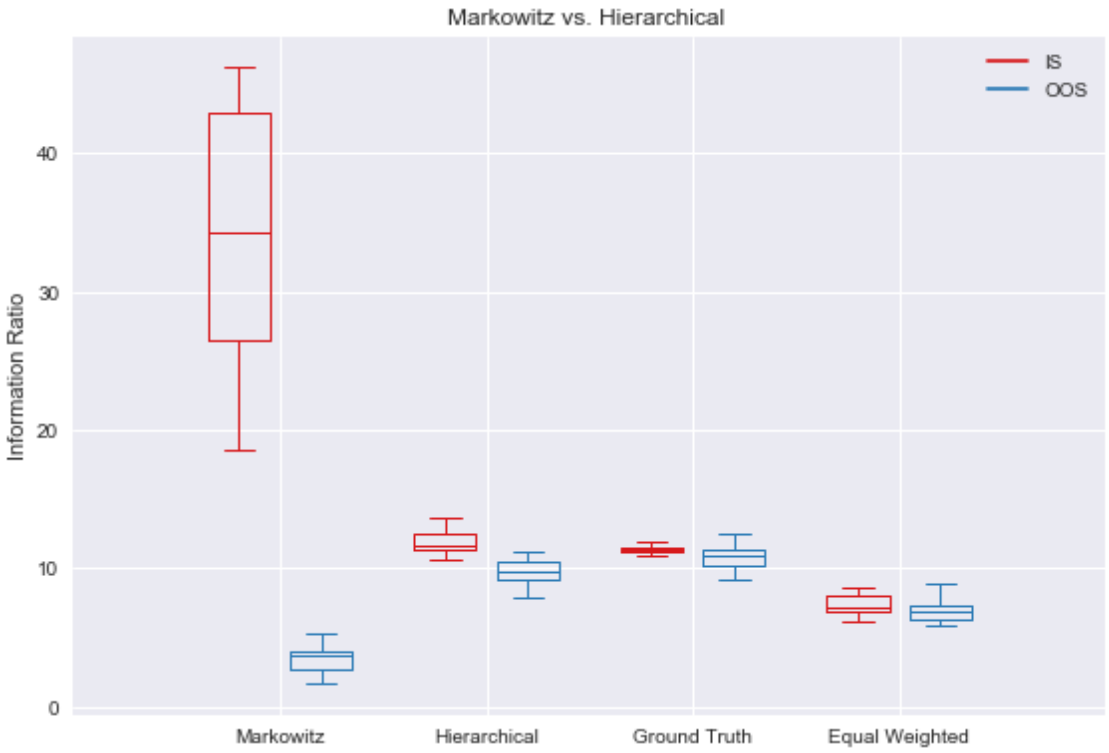

```
IR gt_is:  11.314131864753245
IR gt_oos:  10.189807481406175
IR sm_is:  24.29436479484996
IR sm_oos:  3.6976978228797575
{'banks': 0.13694293962451007, 'oil': 0.0713312181131816, 'insurance':
0.05052037549290519, 'tech': 0.15803648924317365, 'bio': 0.08670867764640053,
'pharma': 0.235242559499063, 'auto': 0.08409028726512993, 'retail':
0.045007198845270865, 'manufacturing': 0.13212025427036522}
IR hier_is:  10.786283035243066
IR hier_oos:  9.101562960138448
IR equal_is:  6.954412369834394
IR equal_oos:  6.366962592902686
MC run:  9
```




```
IR gt_is:  12.475883380288082
IR gt_oos:  11.153592087178119
IR sm_is:  32.553145414913295
IR sm_oos:  4.135221122554849
{'banks': 0.14436097328915878, 'oil': 0.08998102839366623, 'insurance':
0.11070505058822581, 'tech': 0.10947743077812186, 'bio': 0.16593405498032845,
'pharma': 0.12202713357598813, 'auto': 0.10436272535851322, 'retail':
0.09278710628805323, 'manufacturing': 0.06036449674794429}
IR hier_is:  13.708256384265477
IR hier_oos:  9.941779680136328
IR equal_is:  9.880530425251155
IR equal_oos:  8.959783982836568
```

```
In [94]: 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 [95]: performance_factor = np.mean(ir_hier_oos)/np.mean(ir_sm_oos)
         print('OOS Hierarchical / Markowitz: ', performance_factor)
```

```
OOS Hierarchical / Markowitz:  2.7680500164871074
```

```
In [96]: performance_factor = np.mean(ir_equal_oos)/np.mean(ir_sm_oos)
         print('OOS Equal / Markowitz: ', performance_factor)
```

```
OOS Equal / Markowitz:  1.9780630210638535
```

```
In [97]: performance_factor = np.mean(ir_hier_oos)/np.mean(ir_equal_oos)
         print('OOS Hierarchical / Equal: ', performance_factor)
```

```
OOS Hierarchical / Equal:  1.399374027526372
```

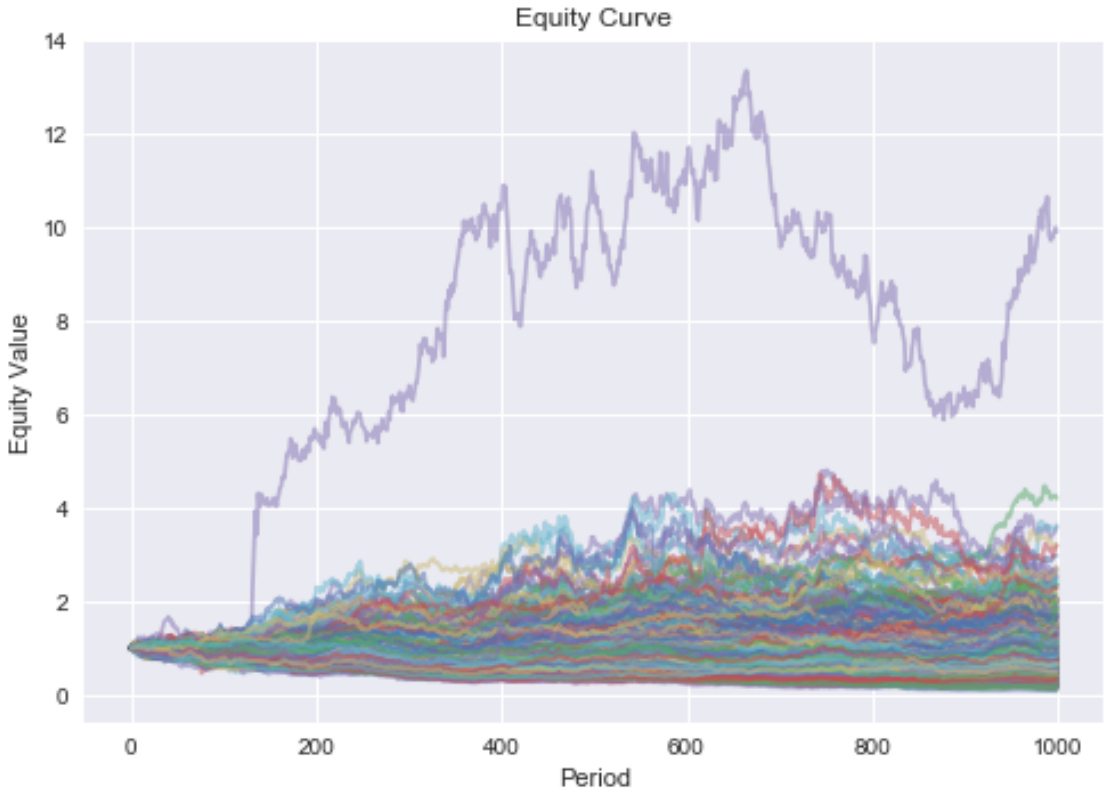
Under fat-tails Markowitz performs even worse since extreme positions expose the portfolio to asset specific tail risk. In particular, Markowitz underperforms equal weighting significantly. The hierarchical method still outperforms equal weighting, though.

```
In [98]: 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]))
```

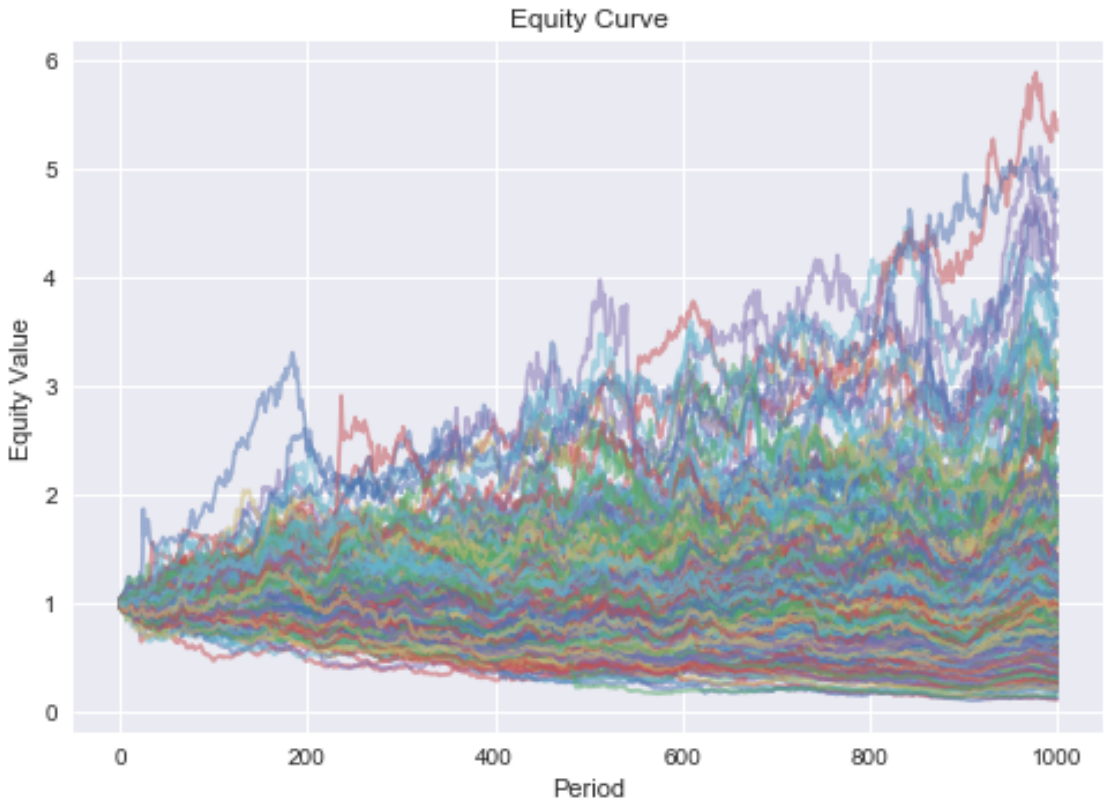
```
        B[n_stocks*n:n_stocks*(n+1), n+1] =
np.array(list(industries_beta_industries[i]))

        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
    cov_mean = cov.values.mean()
    pc_cov = optimize.pc_cov(cov-cov_mean, n) + cov_mean

    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)
```

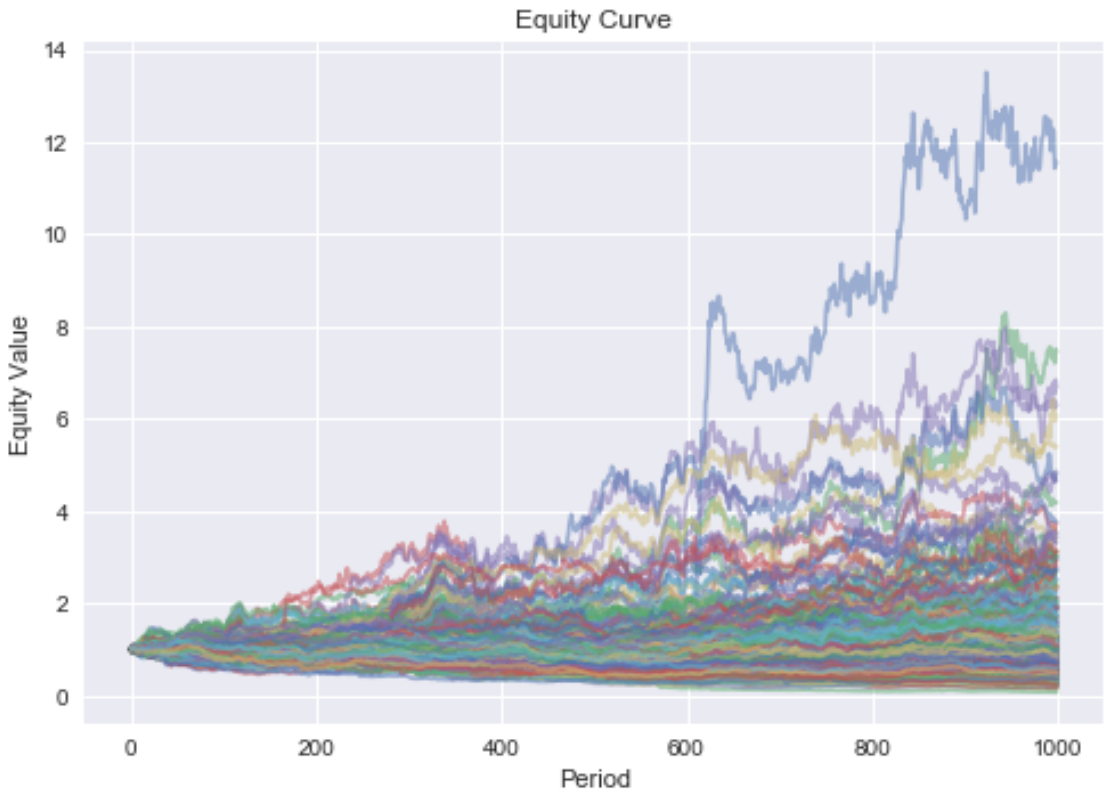
```
MC run:  0
```



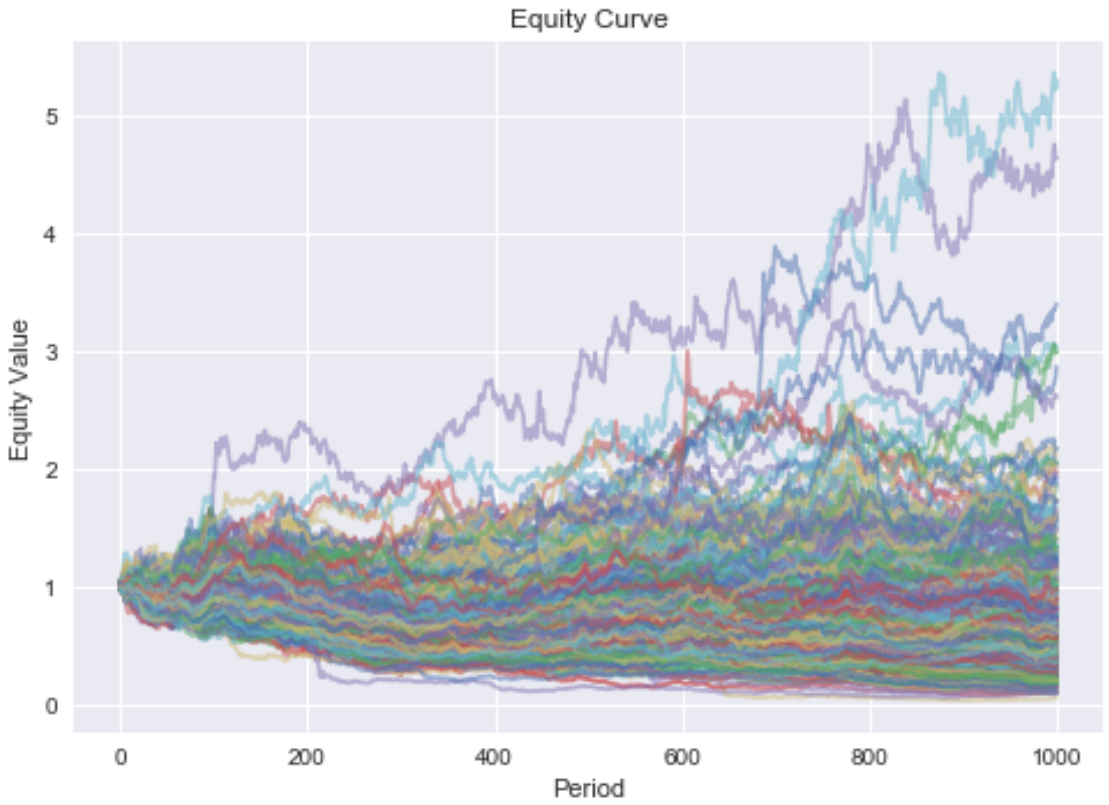

```
IR gt_is 12.651988159357627
IR gt_oos 12.396596738687132


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


IR sm_is:  35.67554748930204
IR sm_oos:  3.5886144601105574
IR pca_is:  11.723523668600246
IR pca_oos:  11.619604467670277
MC run:  1
```



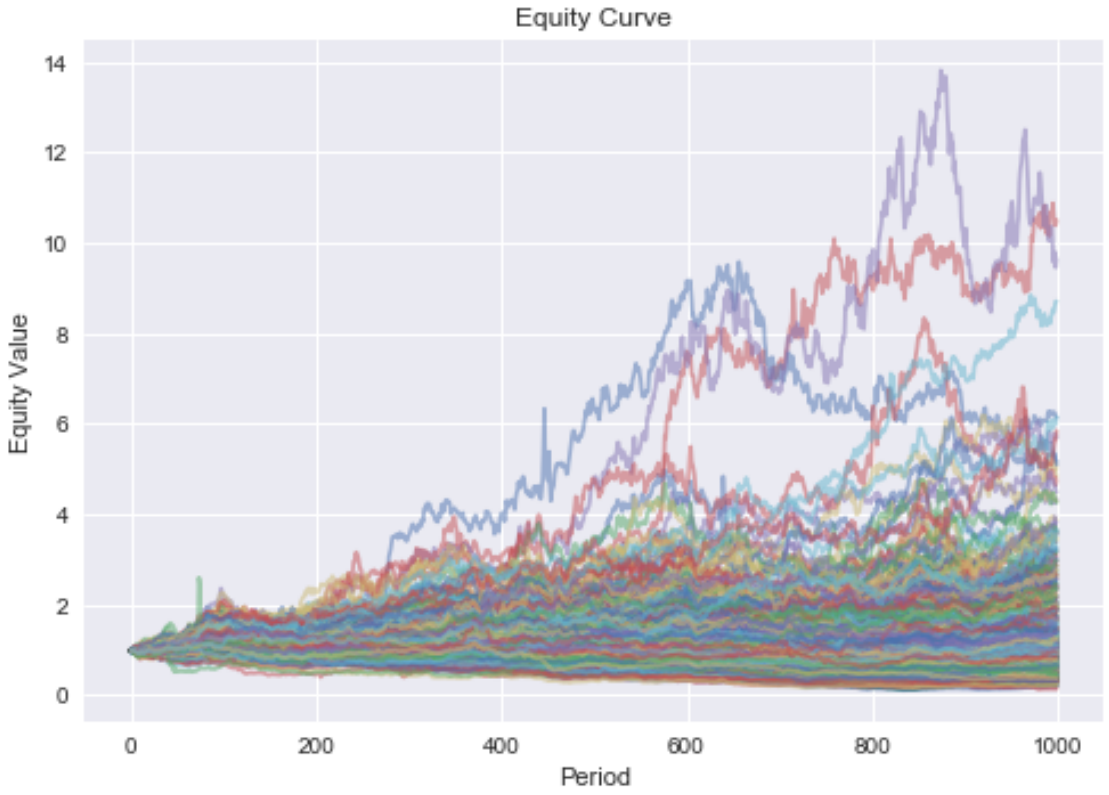

```
IR gt_is 12.359781766605892
IR gt_oos 11.539901022290252
IR sm_is:  39.99592617366191
IR sm_oos:  2.618285541761696
IR pca_is:  11.406285104524494
IR pca_oos:  10.921434933769946
MC run:  2
```



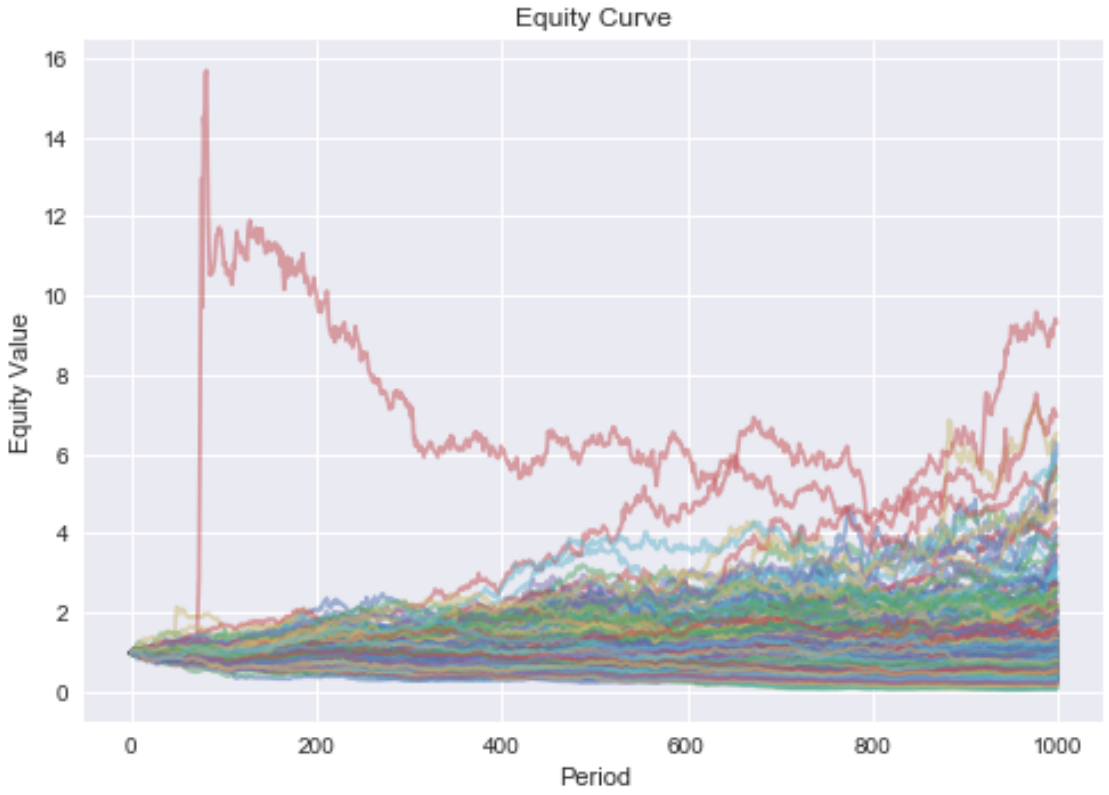

```
IR gt_is 10.586439918995474
IR gt_oos 11.567416233500039
IR sm_is:  36.17296935279856
IR sm_oos:  3.5985367123193392
IR pca_is:  9.672011861710288
IR pca_oos:  10.995837038898319
MC run:  3
```



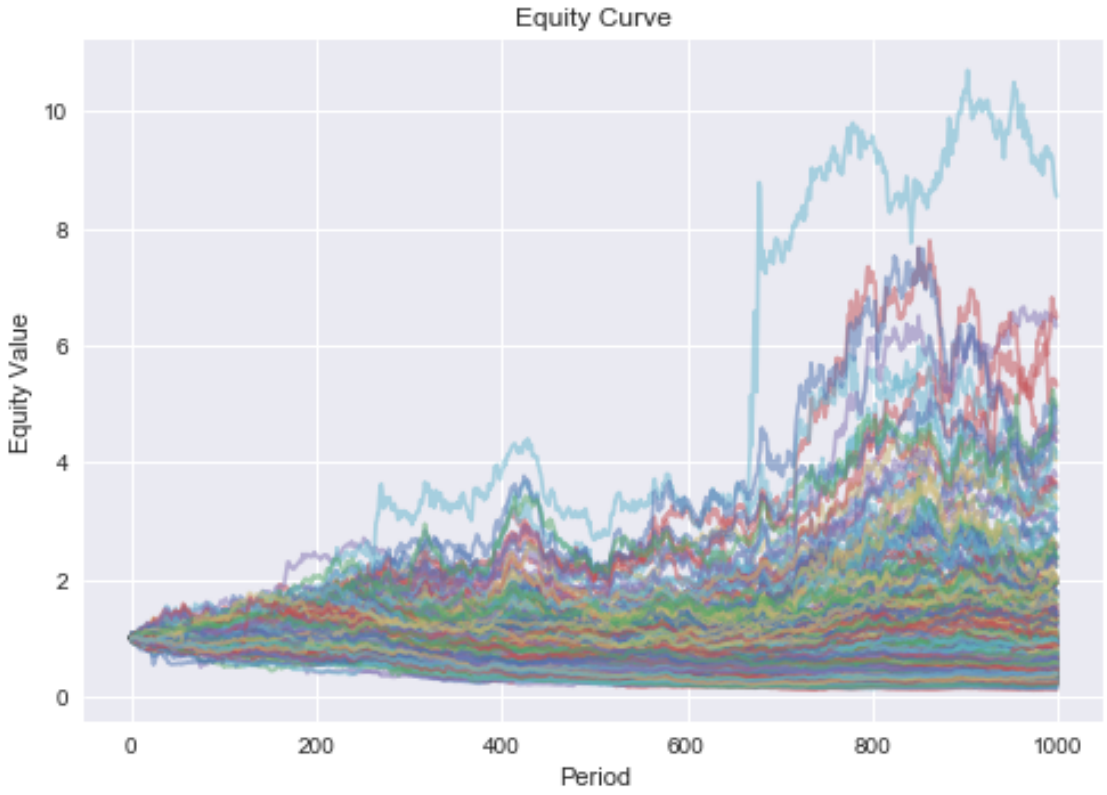

```
IR gt_is 10.609920064427657
IR gt_oos 11.707503443739451
IR sm_is:  47.74253734787293
IR sm_oos:  4.032537794362301
IR pca_is:  10.069199973702043
IR pca_oos:  10.511123507571028
MC run:  4
```



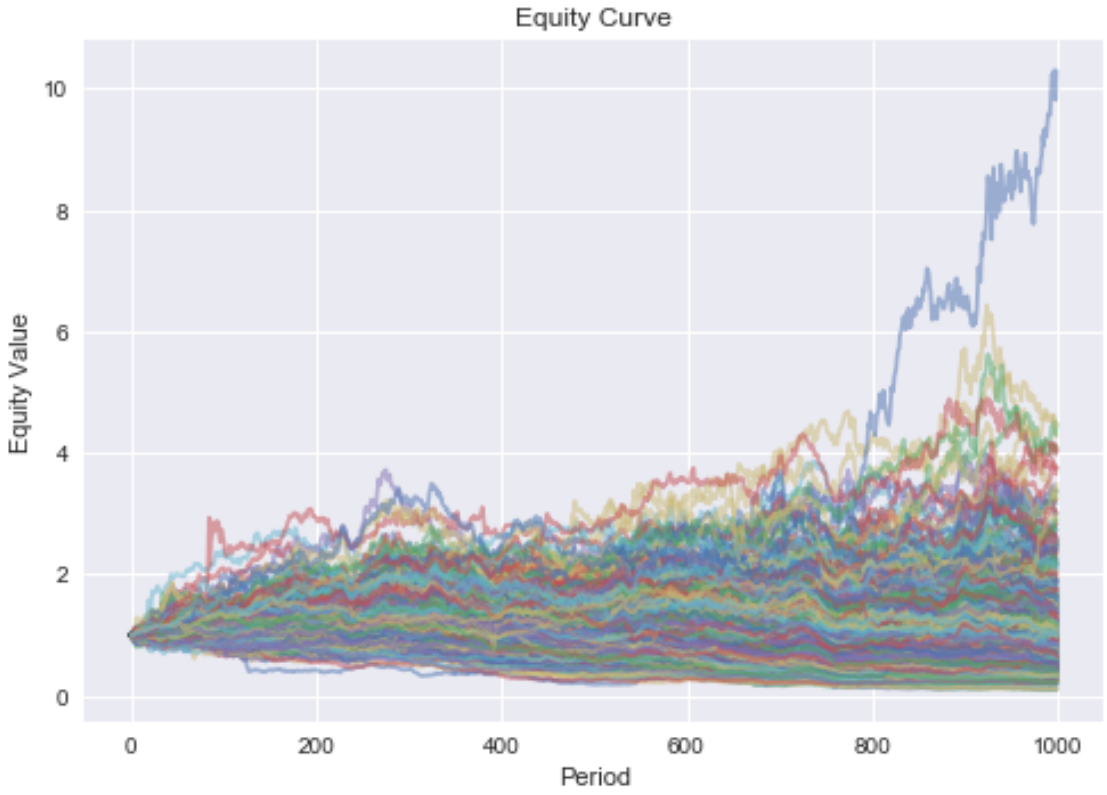

```
IR gt_is 10.398774741129852
IR gt_oos 11.531257941357945
IR sm_is:  45.421764637718624
IR sm_oos:  4.2532889380036885
IR pca_is:  10.076656304588647
IR pca_oos:  10.734208907931896
MC run:  5
```



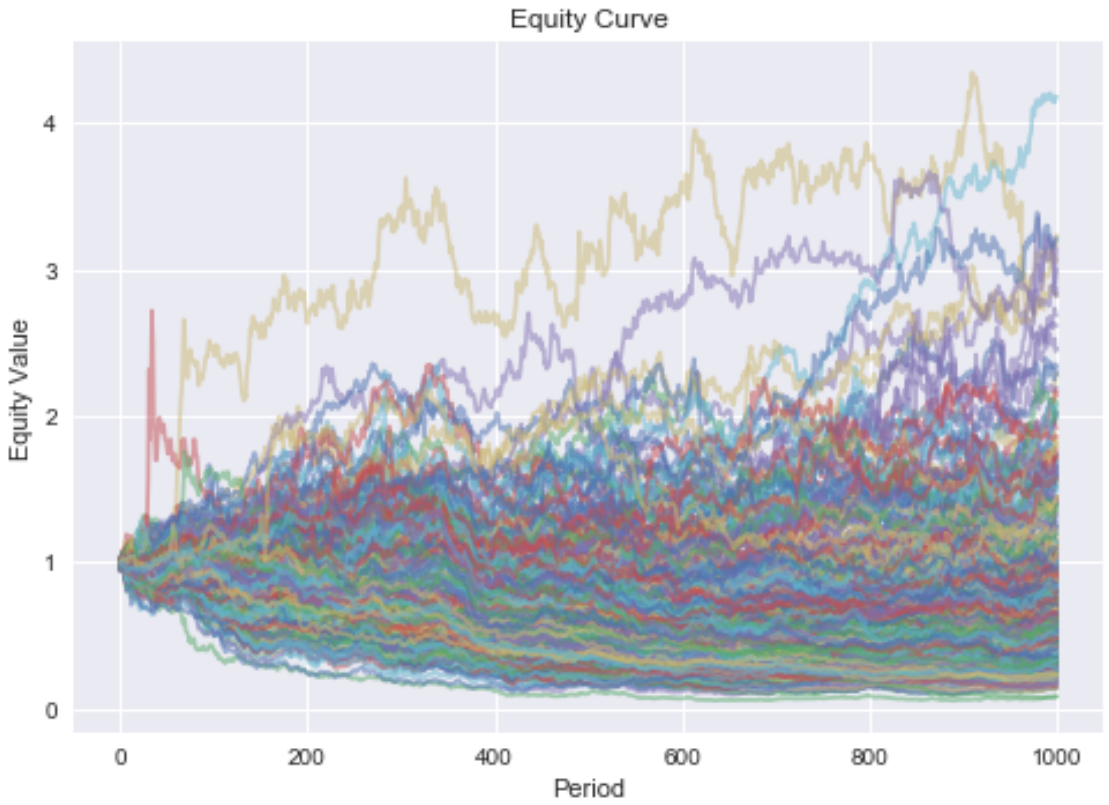

```
IR gt_is 11.646293995119173
IR gt_oos 12.008589728886246
IR sm_is:  37.1261381646745
IR sm_oos:  4.3910400441536845
IR pca_is:  11.375628364869787
IR pca_oos:  10.77994298981338
MC run:  6
```



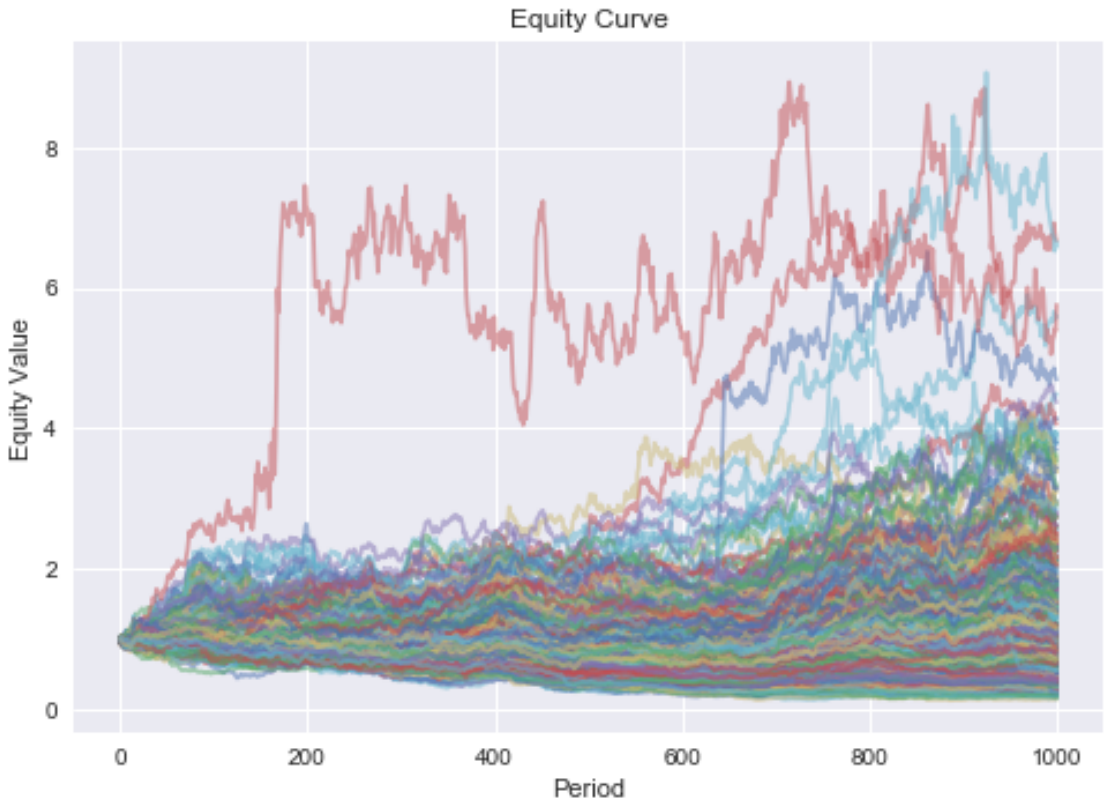

```
IR gt_is 10.306919528506828
IR gt_oos 10.908550451908418
IR sm_is:  26.412179974896116
IR sm_oos:  3.932847673268968
IR pca_is:  10.406844183840237
IR pca_oos:  9.706317195660008
MC run:  7
```




```
IR gt_is 9.789833457127461
IR gt_oos 11.561464658764566
IR sm_is:  21.362116738082474
IR sm_oos:  2.321733985153855
IR pca_is:  8.721256659884057
IR pca_oos:  10.793683529364433
MC run:  8
```




```
IR gt_is 10.683380092702254
IR gt_oos 9.942365334902894
IR sm_is:  46.12747982188347
IR sm_oos:  3.2973332580190013
IR pca_is:  10.197769771627044
IR pca_oos:  9.133148466556378
MC run:  9
```



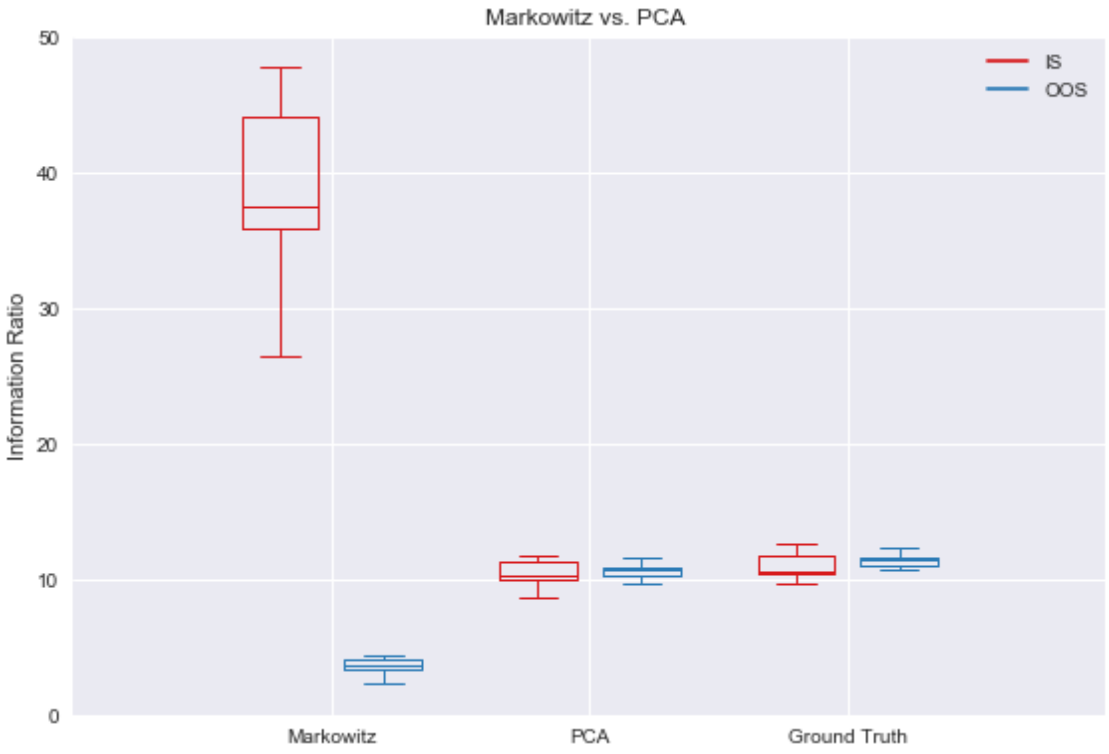

```
IR gt_is 11.737286822940108
IR gt_oos 10.803523139246135
IR sm_is:  37.98478357996682
IR sm_oos:  4.372384011487485
IR pca_is:  11.343939628756928
IR pca_oos:  10.277394838985526
```

```
In [99]: performance_factor = np.mean(ir_pca_oos)/np.mean(ir_sm_oos)
         print('OOS PCA / Markowitz: ', performance_factor)
```

```
OOS PCA / Markowitz:  2.897076048552612
```

```
In [100]: 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 [ ]: