Optimal Portfolio Weights IV: Constrained Optimization

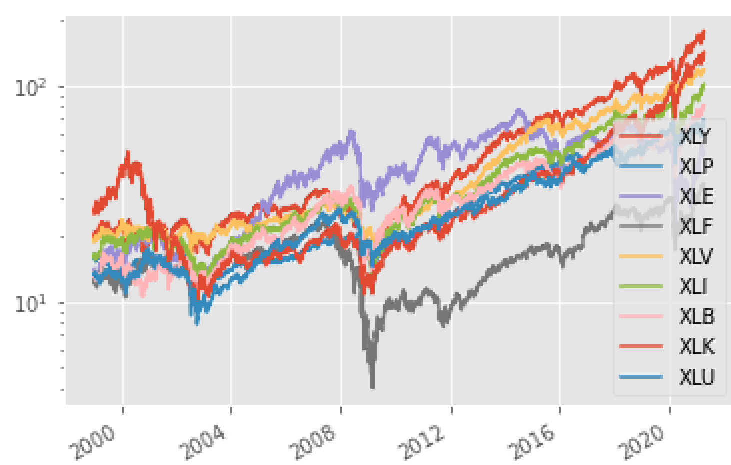
In [1]:
```
# Working with data:
import numpy  as np                                          # For scientific computing
import pandas as pd                                          # Working with tables.

# Downloading files:
import requests, zipfile, io                                       # To access websites

# Specific data providers:
from tiingo import TiingoClient                              # Stock prices.
import quandl                                                # Economic data, futures p

# API keys:
tiingo = TiingoClient({'api_key':'xxxx'})
quandl.ApiConfig.api_key = 'yyyy'

# Plotting:
import matplotlib.pyplot as plt                               # Basic plot library.
plt.style.use('ggplot')                                       # Make plots look nice
```

SPRD sector ETFs

In [2]:
```
sectors = ['XLY', 'XLP', 'XLE', 'XLF', 'XLV', 'XLI', 'XLB', 'XLK', 'XLU']

PRICE = tiingo.get_dataframe(sectors, '1999-1-1', metric_name='adjClose')
PRICE.index = pd.to_datetime(PRICE.index).tz_convert(None)
RET = PRICE.pct_change()

RATES = quandl.get(['FRED/FEDFUNDS','FRED/DGS1']) / 100
RATES.columns = ['FedFunds','Treasury_1']
```

In [3]:
```
PRICE.plot(logy=True)
```

Out[3]: <AxesSubplot:>



Get SPY as benchmark:

In [4]:
```
spy = tiingo.get_dataframe(['SPY'], '1999-1-1', metric_name='adjClose')
```

```
spy.index = pd.to_datetime(spy.index).tz_convert(None)
spy[-3:]
```

Out[4]:

	SPY
2021-04-12	411.64
2021-04-13	412.86
2021-04-14	411.45

Backtest function:

In [5]:

```
def get_rebalance_dates(frequency, start_date):
    price = PRICE[PRICE.index>start_date]
    group = getattr(price.index, frequency)
    return price[:1].index.union(price.groupby([price.index.year, group]).tail(1



def compare_performance(t):
    t.add(1).cumprod().plot()
    t.add(1).cumprod().plot(logy=True)

    annual_returns = t[:'2020'].add(1).resample('A').prod().sub(1)
    r_annual_Tbill = RATES.Treasury_1.resample('A').first()

    x = pd.DataFrame()
    x['Average_returns']   = annual_returns.mean()
    x['Geometric_average'] = annual_returns.add(1).prod().pow(1/len(annual_retur
    x['Risk_premium']      = annual_returns.sub(r_annual_Tbill, 'rows').dropna()
    x['Volatility']        = t[:'2020'].std() * 252**0.5
    x['Sharpe_ratio']      = x.Risk_premium / x.Volatility

    return x




def run_backtest(frequency, backtest_start='1900-1-1'):

    rebalance_dates = get_rebalance_dates(frequency, backtest_start)

    portfolio_value = pd.Series(1,                              index=[rebalance_dates
    weights          = pd.DataFrame(columns=PRICE.columns, index=[rebalance_dates
    trades           = pd.DataFrame(columns=PRICE.columns, index=[rebalance_dates

    previous_positions = weights.iloc[0]

    for i in range(1, len(rebalance_dates)-1):
        start_date = rebalance_dates[i]
        end_date   = rebalance_dates[i+1]

        cum_ret = RET[start_date:end_date][1:].add(1).cumprod()

        start_weights = select_weights(start_date)      # Call "select_weights()

        new_positions = portfolio_value.iloc[-1] * start_weights

        start_to_end_positions = new_positions  * cum_ret
```

```
        start_to_end_value      = start_to_end_positions.sum('columns')

        portfolio_value = portfolio_value.append(start_to_end_value)

        weights = weights.append(start_to_end_positions.div(start_to_end_value,'

        trades.loc[start_date] = new_positions - previous_positions
        previous_positions      = start_to_end_positions.iloc[-1]       # Previous

    return portfolio_value.pct_change(), weights, trades
```

Backtest strategy that equal-weights all assets:

In [6]:

```
pd.Series(1/len(sectors), index=sectors)  # put 1/n in each asset
```

Out[6]:

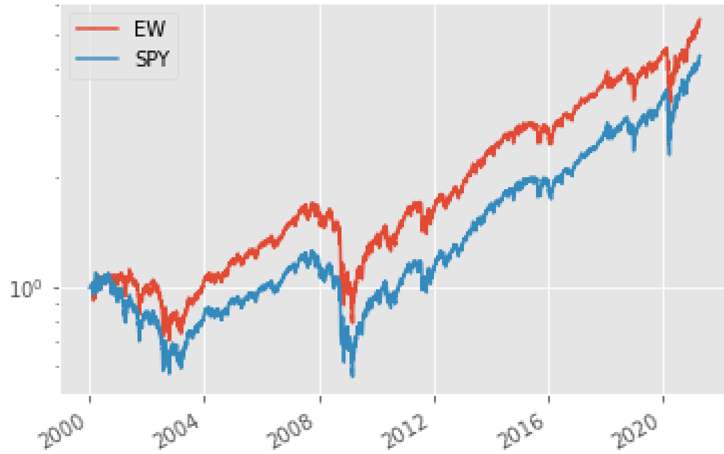
```
XLY    0.111111
XLP    0.111111
XLE    0.111111
XLF    0.111111
XLV    0.111111
XLI    0.111111
XLB    0.111111
XLK    0.111111
XLU    0.111111
dtype: float64
```

In [7]:

```
def select_weights(date):
    return pd.Series(1/len(sectors), index=sectors)   # Equal weights (1/n)


ew, weights, trades = run_backtest('month', '2000-1-1')
ew = ew.rename('EW')

t = pd.DataFrame(ew).join(spy.pct_change())
compare_performance(t)
```

Out[7]:

	Average_returns	Geometric_average	Risk_premium	Volatility	Sharpe_ratio
EW	0.091669	0.078461	0.072226	0.189818	0.380502
SPY	0.082703	0.067384	0.063260	0.198243	0.319105



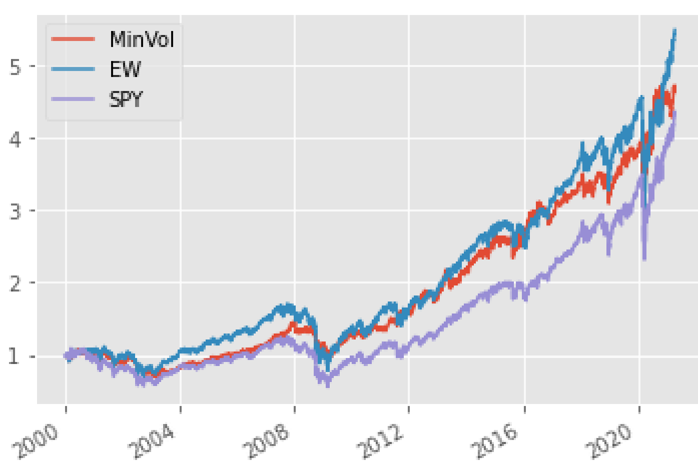
Minimum volatilty weights:

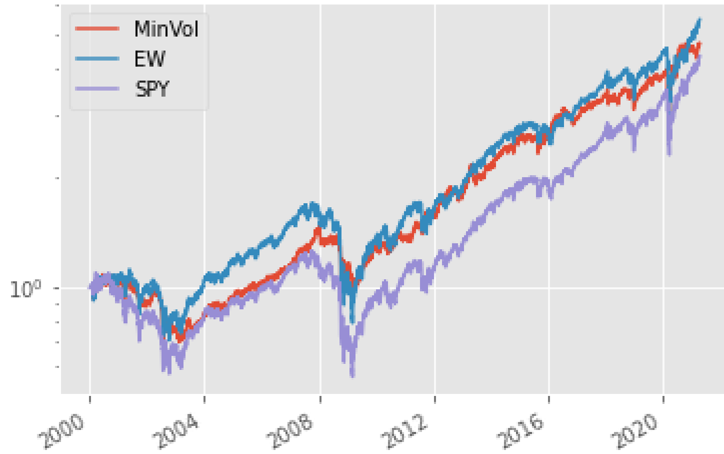
In [9]:

```
def select_weights(date):
    cov     = RET[:date][-100:].cov() * 252  # Use most recent 100 returns up to
    cov_inv = pd.DataFrame(np.linalg.inv(cov), columns=cov.columns, index=cov.in
    w = cov_inv.sum() /  cov_inv.sum().sum()         # Minimum-volatility portfo
    return w


mv, weights, trades = run_backtest('month', '2000-1-1')
mv = mv.rename('MinVol')

t = pd.DataFrame(mv).join(ew).join(spy.pct_change())
compare_performance(t)
```

Out[9]:

	Average_returns	Geometric_average	Risk_premium	Volatility	Sharpe_ratio
MinVol	0.082771	0.075506	0.063328	0.144292	0.438889
EW	0.091669	0.078461	0.072226	0.189818	0.380502
SPY	0.082703	0.067384	0.063260	0.198243	0.319105

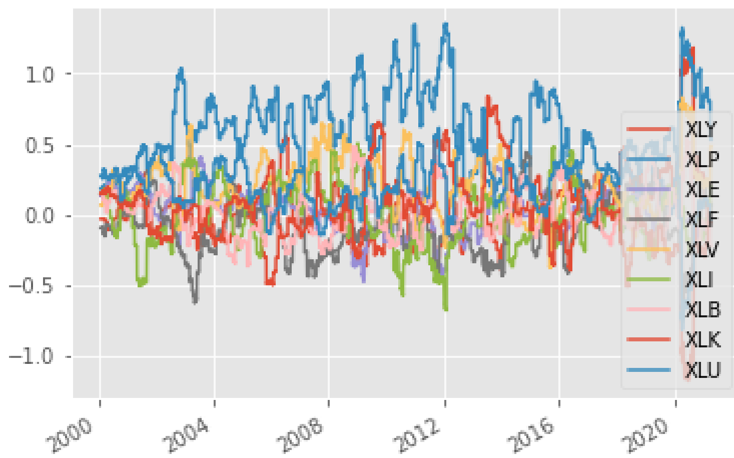




Plot the weights:

In [10]:
```
weights.plot()
```

Out[10]: <AxesSubplot:>



Note how these weights contain short positions (the negative weights).
Why short positions? All these ETFs hace relatively high positive correlation, and so the min-vol portfolio requires that we short some assets:

In [11]:
```
RET.corr()
```

Out[11]:

	XLY	XLP	XLE	XLF	XLV	XLI	XLB	XLK	XL
XLY	1.000000	0.630802	0.550253	0.757101	0.698223	0.818486	0.719942	0.721419	0.51033
XLP	0.630802	1.000000	0.477009	0.578416	0.619822	0.634423	0.564218	0.490179	0.60084
XLE	0.550253	0.477009	1.000000	0.579339	0.510408	0.661278	0.688992	0.474144	0.51416
XLF	0.757101	0.578416	0.579339	1.000000	0.632924	0.786996	0.690217	0.630460	0.50337
XLV	0.698223	0.619822	0.510408	0.632924	1.000000	0.714460	0.604808	0.678747	0.54092
XLI	0.818486	0.634423	0.661278	0.786996	0.714460	1.000000	0.809604	0.729945	0.55670
XLB	0.719942	0.564218	0.688992	0.690217	0.604808	0.809604	1.000000	0.588904	0.50931
XLK	0.721419	0.490179	0.474144	0.630460	0.678747	0.729945	0.588904	1.000000	0.46187

	XLY	XLP	XLE	XLF	XLV	XLI	XLB	XLK	XL
XLU	0.510337	0.600842	0.514161	0.503379	0.540921	0.556703	0.509319	0.461872	1.00000

Problem: shorting is costly:

- we have to pay a borrow fee when we borrow stocks
- we have to keep the (negative) value of the short position in cash (as collateral)
- we earn interest on the collateral (benchmark rate minus some spread; currently this interest is zero)
- we cannot use the collateral to finance long positions

Example:
initially you have $100 cash in your account (and nothing else),
now you short an asset and receive 50 in cash from the sale and then you buy another asset for 150.

intial account:

- cash: 100
- equity: 100 (account value)

After the transction:

- stock A: -50 (short, pay fee)
- stock B: 150 (long)
- collateral: 50 (from short sale, earns interest)
- margin loan: 50 (to pay for stock B, pay interest)
- cash total: 0
- equity: 100 (account value)

(The interest you pay on the margin loan is higher than the interest you earn on the collateral (currently you earn zero interest, and pay about 1.5%))

Most recent weights of the min-vol portfolio:

In [12]:
```
weights.iloc[-1]  # Last row of weights table
```

Out[12]:
```
XLY    0.072929
XLP    0.530791
XLE   -0.042154
XLF    0.066804
XLV    0.486284
XLI    0.200180
XLB   -0.223202
XLK   -0.053567
XLU   -0.038065
Name: 2021-04-14 00:00:00, dtype: float64
```

"Clip" weights at zero:

In [13]:
```
weights.iloc[-1].clip(0)
```

Out[13]:
```
XLY    0.072929
XLP    0.530791
XLE    0.000000
XLF    0.066804
XLV    0.486284
XLI    0.200180
XLB    0.000000
XLK    0.000000
XLU    0.000000
Name: 2021-04-14 00:00:00, dtype: float64
```

Sum of these weights:

In [14]:
```
weights.iloc[-1].clip(0).sum()
```

Out[14]:
```
1.3569883034998693
```

→ when we set negative weights to zero, the weights don't sum to one anymore!

How can we transform the weights so that they sum to one?

→ divide the clipped weights by their sum:

In [15]:
```
w = weights.iloc[-1].clip(0) / weights.iloc[-1].clip(0).sum()
w
```

Out[15]:
```
XLY    0.053744
XLP    0.391153
XLE    0.000000
XLF    0.049229
XLV    0.358355
XLI    0.147518
XLB    0.000000
XLK    0.000000
XLU    0.000000
Name: 2021-04-14 00:00:00, dtype: float64
```

Check the sum:

In [16]:
```
w.sum()
```

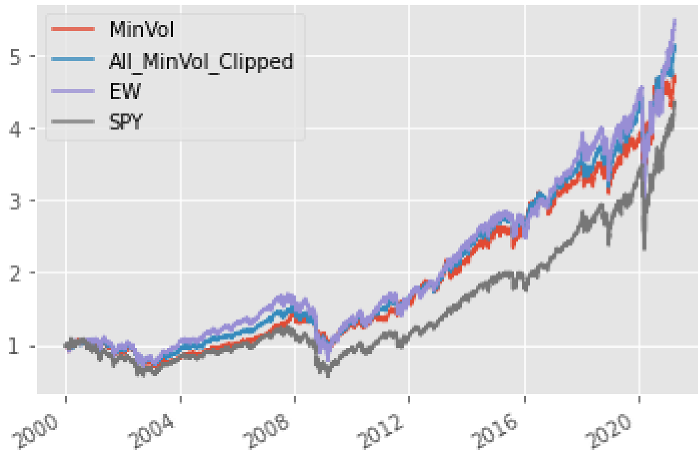
Out[16]:
```
1.0
```

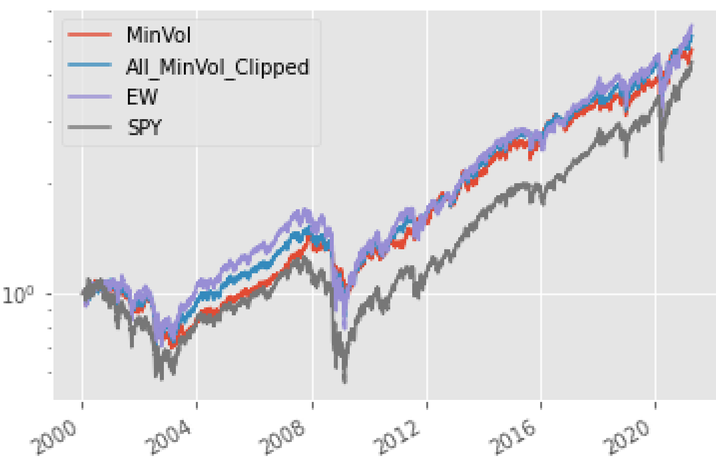
Add this to our select_weights function:

In [18]:
```
def select_weights(date):
    cov     = RET[:date][-100:].cov() * 252  # Use most recent 100 returns up to
    cov_inv = pd.DataFrame(np.linalg.inv(cov), columns=cov.columns, index=cov.in
    w = cov_inv.sum() /  cov_inv.sum().sum()         # Minimum-volatility portfo
    w = w.clip(0) / w.clip(0).sum()                  # Clip weights at zero
    return w


mv_clipped, weights, trades = run_backtest('quarter', '2000-1-1')
mv_clipped = mv_clipped.rename('All_MinVol_Clipped')

t = pd.DataFrame(mv).join(mv_clipped).join(ew).join(spy.pct_change())
compare_performance(t)
```

Out[18]:

	Average_returns	Geometric_average	Risk_premium	Volatility	Sharpe_ratio
MinVol	0.082771	0.075506	0.063328	0.144292	0.438889
All_MinVol_Clipped	0.087075	0.078038	0.067633	0.155712	0.434345
EW	0.091669	0.078461	0.072226	0.189818	0.380502
SPY	0.082703	0.067384	0.063260	0.198243	0.319105



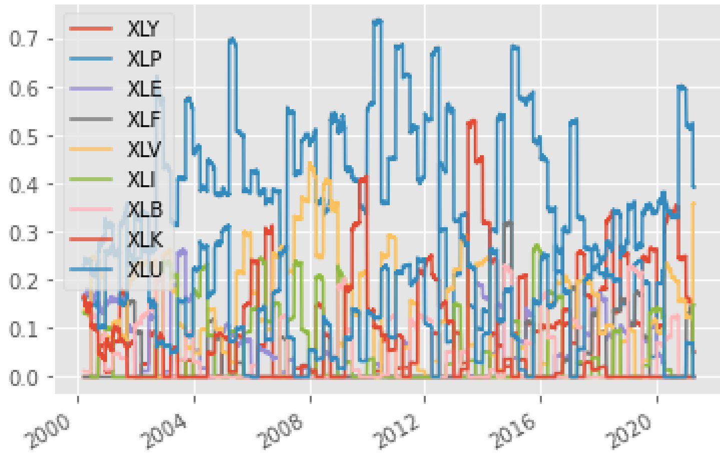


Note how the uncontrained min-vol portfolio has a lower volatility than the "clipped" portfolio (in the table above).

In [19]:

```
weights.plot()
```

Out[19]: <AxesSubplot:>



Here we chose weights like this:

1. find minimum volatility weights
2. set all negative weights to zero
3. recalibrate weights so they sum to zero: divide all weights by their sum

But this procedures only an approximation of the optimal weights.
To find the actual weights we need to:

- minimize volatility under the constraint that weights are ≥ 0

Install cvxopt library for constrained optimization:

In [20]:

```
pip install CVXOPT
```

```
Requirement already satisfied: CVXOPT in /Users/janschneider/opt/anaconda3/lib/p
ython3.7/site-packages (1.2.6)
Note: you may need to restart the kernel to use updated packages.
```

In [21]:

```
import cvxopt
from cvxopt import matrix, solvers
solvers.options['show_progress'] = False
```

Now use the cvxopt like this:

In [22]:

```
minimum_weight = 0    # or try for example 0.05 to get minimum diversification

cov = RET.cov()


n = len(cov)
P = matrix(  cov.values )
q = matrix(  np.zeros(n) )
G = matrix( -np.identity(n) )
h = matrix( -np.ones(n)*minimum_weight)
A = matrix(  np.ones(n), (1,n))              # weights sum to 1
b = matrix(1.0)

sol = solvers.qp(P, q, G, h, A, b)

pd.Series({cov.index[i]:sol['x'][i] for i in range(n)})
```

Out[22]:
```
XLY    0.000190
XLP    0.608019
XLE    0.000146
XLF    0.000010
XLV    0.224365
XLI    0.000241
XLB    0.000618
XLK    0.000346
XLU    0.166064
dtype: float64
```

Compare clipped weights to constrained minimum:

In [23]:
```
weights = pd.DataFrame()

# Min-vol with short
cov     = RET.cov()
cov_inv = pd.DataFrame(np.linalg.inv(cov), columns=cov.columns, index=cov.index)
w = cov_inv.sum() /  cov_inv.sum().sum()

weights['With_shortselling'] = w



# Min-vol, clipped at zero:
cov     = RET.cov()
cov_inv = pd.DataFrame(np.linalg.inv(cov), columns=cov.columns, index=cov.index)
w = cov_inv.sum() /  cov_inv.sum().sum()
w = w.clip(0) / w.clip(0).sum()

weights['Clipped'] = w


# Constrained minimum:
minimum_weight = 0
cov = RET.cov()

n = len(cov)
P = matrix( cov.values )
q = matrix( np.zeros(n) )
G = matrix( -np.identity(n) )
h = matrix( -np.ones(n)*minimum_weight)
A = matrix( np.ones(n), (1,n))
b = matrix(1.0)

sol = solvers.qp(P, q, G, h, A, b)

weights['Constrained'] = pd.Series({cov.index[i]:sol['x'][i] for i in range(n)})
weights
```

Out[23]:

	With_shortselling	Clipped	Constrained
XLY	0.022612	0.018936	0.000190
XLP	0.622693	0.521466	0.608019
XLE	-0.016874	0.000000	0.000146
XLF	-0.176690	0.000000	0.000010

	With_shortselling	Clipped	Constrained
XLV	0.276966	0.231941	0.224365
XLI	0.048347	0.040488	0.000241
XLB	0.040495	0.033912	0.000618
XLK	-0.000558	0.000000	0.000346
XLU	0.183008	0.153258	0.166064

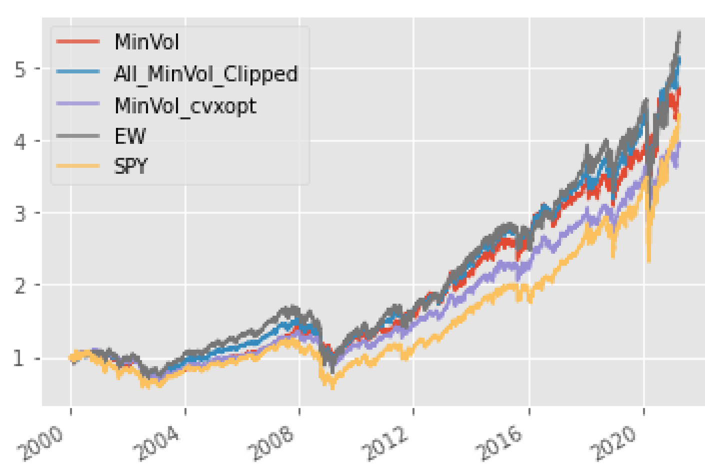
Put the contrained optimization code into the select weights function:

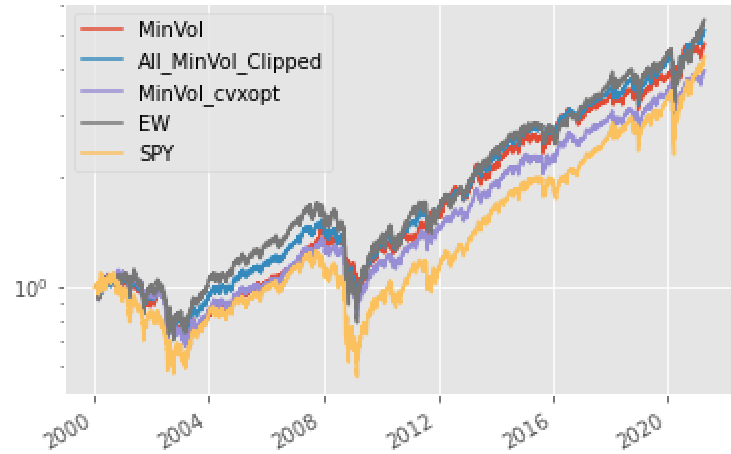
In [24]:

```
def select_weights(date):
    cov = RET[:date][-100:].cov()
    minimum_weight = 0
    n = len(cov)
    P = matrix(  cov.values )
    q = matrix(  np.zeros(n) )
    G = matrix( -np.identity(n) )
    h = matrix( -np.ones(n)*minimum_weight )
    A = matrix(  np.ones(n), (1,n))
    b = matrix(1.0)
    sol = solvers.qp(P, q, G, h, A, b)
    return pd.Series({cov.index[i]:sol['x'][i] for i in range(n)})


mv_cvxopt, weights, trades = run_backtest('month', '2000-1-1')
mv_cvxopt = mv_cvxopt.rename('MinVol_cvxopt')

t = pd.DataFrame(mv).join(mv_clipped).join(mv_cvxopt).join(ew).join(spy.pct_chan
compare_performance(t)
```

Out[24]:

	Average_returns	Geometric_average	Risk_premium	Volatility	Sharpe_ratio
MinVol	0.082771	0.075506	0.063328	0.144292	0.438889
All_MinVol_Clipped	0.087075	0.078038	0.067633	0.155712	0.434345
MinVol_cvxopt	0.074977	0.066212	0.055534	0.146711	0.378528
EW	0.091669	0.078461	0.072226	0.189818	0.380502
SPY	0.082703	0.067384	0.063260	0.198243	0.319105





Note how the coontrained minimum portfolio has a lower volatility than the unconstrained min-vol portfolio that was clipped at zero (in the table above).