CVXPY exercises

Simple cvxpy exercise

Solve the following optimization problem using CVXPY:

$$\begin{array}{ll} \text{minimize} & |x| - 2\sqrt{y} \\ \text{subject to} & 2 \geq e^x \\ & x + y = 5, \end{array}$$

where $x, y \in \mathbb{R}$ are variables. Find the optimal values of x and y.

LINK to the documentation.

```
# Your code here
```

EXTRA: Risk budget allocation.

Suppose an amount $x_i > 0$ is invested in n assets, labeled $i = 1, \ldots, n$, with asset return covariance matrix $\Sigma \in \mathcal{S}^n_{++}$. We define the *risk* of the investments as the standard deviation of the total return, $R(x) = (x^T \Sigma x)^{1/2}$.

We define the (relative) *risk contribution* of asset i (in the portfolio x) as

$$\rho_i = \frac{\partial \log R(x)}{\partial \log x_i} = \frac{\partial R(x)}{R(x)} \frac{x_i}{\partial x_i}, \qquad i = 1, \ldots, n.$$

Why is the logarithm here?! Because it reflects fraction of relative changex (say, per 1%). Take a look at easticity definition at wiki. Thus ρ_i gives the fractional increase in risk per fractional increase in investment i. We can express the risk contributions as

$$\rho_i = \frac{x_i (\Sigma x)_i}{x^T \Sigma x}, \qquad i = 1, \ldots, n,$$

from which we see that $\sum_{i=1}^n \rho_i = 1$. For general x, we can have $\rho_i < 0$, which means that a small increase in investment i decreases the risk. Desirable investment choices have $\rho_i > 0$, in which case we can interpret ρ_i as the fraction of the total risk contributed by the investment in asset i. Note that the risk contributions are homogeneous, i.e., scaling x by a positive constant does not affect ρ_i.

Problem statement

In the *risk budget allocation problem*, we are given Σ and a set of desired risk contributions $\rho_i^{\text{des}} > 0$ with $\mathbf{1}^\mathbf{T} \rho^{\text{des}} = \mathbf{1}$; the goal is to find an investment mix $x \succ 0$, $\mathbf{1}^\mathbf{T}\mathbf{x} = \mathbf{1}$, with these risk contributions. When $\rho^{\text{des}} = (1/n)\mathbf{1}$, the problem is to find an investment mix that achieves so-called *risk parity*.

(a)

Explain how to solve the risk budget allocation problem using convex optimization.

Hint. Minimize $(1/2)x^T \Sigma x - \sum_{i=1}^n \rho_i^{\text{des}} \log x_i$.

Double-click (or enter) to edit

(b)

Find the investment mix that achieves risk parity for the return covariance matrix Σ below.

```
import numpy as np
import cvxpy as cp
Sigma = np.array(np.matrix("""6.1  2.9  -0.8  0.1;
                         2.9  4.3  -0.3  0.9;
                        -0.8 -0.3   1.2 -0.7;
                         0.1  0.9  -0.7  2.3"""))
rho = np.ones(4)/4
```

```
### YOUR CODE
```

Portfolio optimization

source

Portfolio allocation vector

In this example we show how to do portfolio optimization using CVXPY. We begin with the basic definitions. In portfolio optimization we have some amount of money to invest in any of n different assets. We choose what fraction w_i of our money to invest in each asset i, $i = 1, \ldots, n$.

We call $w \in \mathbf{R}^n$ the *portfolio allocation vector*. We of course have the constraint that $\mathbf{1}^T w = 1$. The allocation $w_i < 0$ means a *short position* in asset i, or that we borrow shares to sell now that we must replace later. The allocation $w \geq 0$ is a *long only* portfolio. The quantity

$$\|w\|_1 = \mathbf{1}^T w_+ + \mathbf{1}^T w_-$$

is known as *leverage*.

Asset returns

We will only model investments held for one period. The initial prices are $p_i > 0$. The end of period prices are $p_i^+ > 0$. The asset (fractional) returns are $r_i = (p_i^+ - p_i)/p_i$. The porfolio (fractional) return is $R = r^T w$.

A common model is that r is a random variable with mean $\mathbf{E}r = \mu$ and covariance $\mathbf{E(r - \mu)(r - \mu)^T} = \Sigma$. It follows that R is a random variable with $\mathbf{E}R = \mu^T w$ and $\mathbf{var}(R) = w^T \Sigma w$. $\mathbf{E}R$ is the (mean) *return* of the portfolio. $\mathbf{var}(R)$ is the *risk* of the portfolio. (Risk is also sometimes given as $\mathbf{std}(R) = \sqrt{\mathbf{var}(R)}$.)

Portfolio optimization has two competing objectives: high return and low risk.

Classical (Markowitz) portfolio optimization

Classical (Markowitz) portfolio optimization solves the optimization problem

$$\begin{array}{ll} \text{maximize} & \mu^T w - \gamma w^T \Sigma w \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \in \mathcal{W}, \end{array}$$

where $w \in \mathbf{R}^n$ is the optimization variable, $\mathcal{W}$ is a set of allowed portfolios (e.g., $\mathcal{W} = \mathbf{R}^n_+$ for a long only portfolio), and $\gamma > 0$ is the *risk aversion parameter*.

The objective $\mu^T w - \gamma w^T \Sigma w$ is the *risk-adjusted return*. Varying γ gives the optimal *risk-return trade-off*. We can get the same risk-return trade-off by fixing return and minimizing risk.

Example

In the following code we compute and plot the optimal risk-return trade-off for 10 assets, restricting ourselves to a long only portfolio.

In [ ]:

```
# Generate data for long only portfolio optimization.
import numpy as np
np.random.seed(1)
n = 10
mu = np.abs(np.random.randn(n, 1))
Sigma = np.random.randn(n, n)
Sigma = Sigma.T @ Sigma
```

In [ ]:

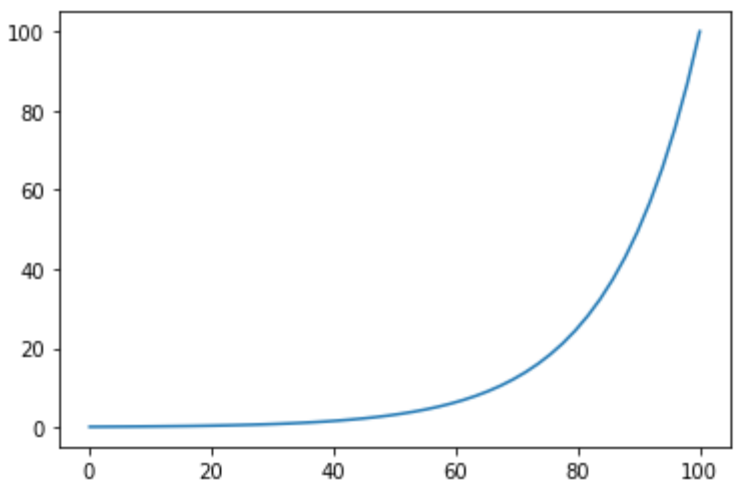
```
# Long only portfolio optimization.
import cvxpy as cp


w = cp.Variable(n)
gamma = cp.Parameter(nonneg=True)
ret = mu.T @ w
risk = cp.quad_form(w, Sigma)
prob = cp.Problem(cp.Minimize(gamma*risk - ret),
               [cp.sum(w) == 1,
                w >= 0])
```

In [ ]:

```
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f7524867690>]
```



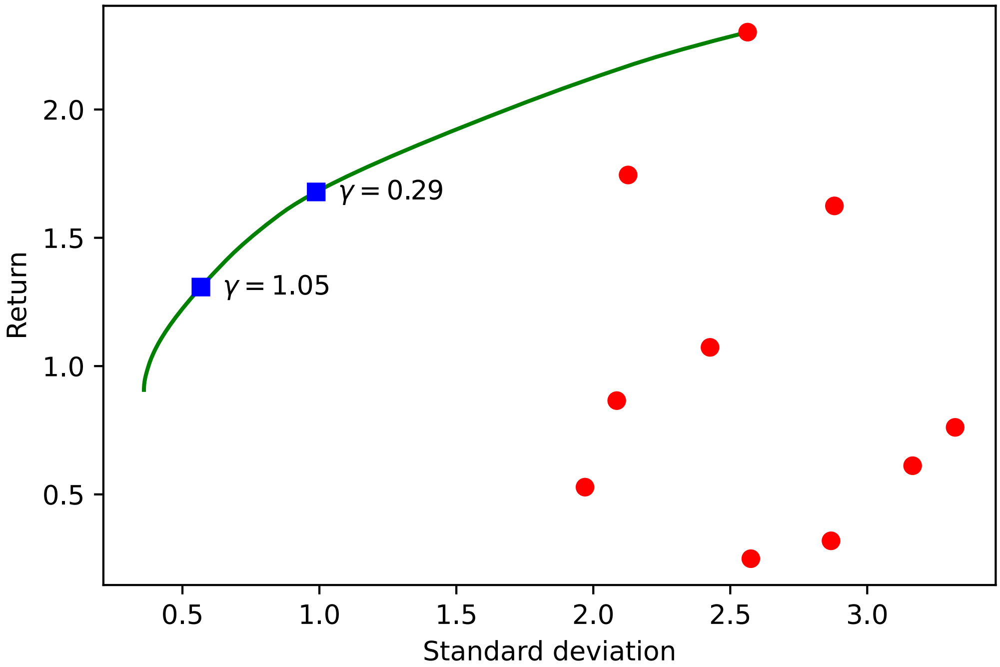
In [ ]:

```
# Compute trade-off curve.
```

```
from tqdm.auto import tqdm
SAMPLES = 100
risk_data = np.zeros(SAMPLES)
ret_data = np.zeros(SAMPLES)
gamma_vals = np.logspace(-2, 3, num=SAMPLES)
for i in tqdm(range(SAMPLES)):
    gamma.value = gamma_vals[i]
    prob.solve()
    risk_data[i] = cp.sqrt(risk).value
    ret_data[i] = ret.value
```

In [ ]:

```
# Plot long only trade-off curve.
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

markers_on = [29, 40]
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(risk_data, ret_data, 'g-')
for marker in markers_on:
    plt.plot(risk_data[marker], ret_data[marker], 'bs')
    ax.annotate(r"$\gamma = %.2f$" % gamma_vals[marker], xy=(risk_data[marker]+.08
for i in range(n):
    plt.plot(cp.sqrt(Sigma[i,i]).value, mu[i], 'ro')
plt.xlabel('Standard deviation')
plt.ylabel('Return')
plt.show()
```

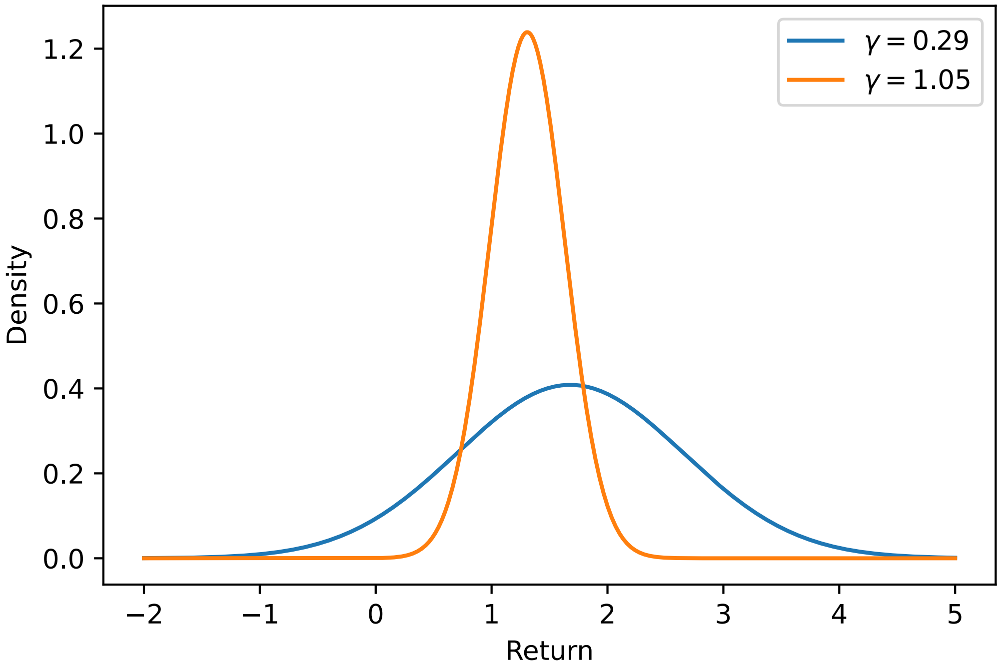


We plot below the return distributions for the two risk aversion values marked on the trade-off curve. Notice that the probability of a loss is near 0 for the low risk value and far above 0 for the high risk value.

In [ ]:

```
# Plot return distributions for two points on the trade-off curve.
import scipy.stats as spstats


plt.figure()
for midx, idx in enumerate(markers_on):
    gamma.value = gamma_vals[idx]
    prob.solve()
    x = np.linspace(-2, 5, 1000)
```

```
    plt.plot(x, spstats.norm.pdf(x, ret.value, risk.value), label=r"$\gamma = %.2f

plt.xlabel('Return')
plt.ylabel('Density')
plt.legend(loc='upper right')
plt.show()
```



Portfolio constraints

There are many other possible portfolio constraints besides the long only constraint. With no constraint ($\mathcal{W} = \mathbf{R}^n$), the optimization problem has a simple analytical solution. We will look in detail at a *leverage limit*, or the constraint that $\|w\|_1 \leq L^{\max}$.

Another interesting constraint is the *market neutral* constraint $m^T \Sigma w = 0$, where m_i is the capitalization of asset i. $M = m^T r$ is the *market return*, and $m^T \Sigma w = \mathbf{cov}(M, R)$. The market neutral constraint ensures that the portfolio return is uncorrelated with the market return.

Example

In the following code we compute and plot optimal risk-return trade-off curves for leverage limits of 1, 2, and 4. Notice that more leverage increases returns and allows greater risk.

In [ ]:

```
# Portfolio optimization with leverage limit.
Lmax = cp.Parameter()
prob = cp.Problem(cp.Maximize(ret - gamma*risk),
               [cp.sum(w) == 1,
                cp.norm(w, 1) <= Lmax])
```

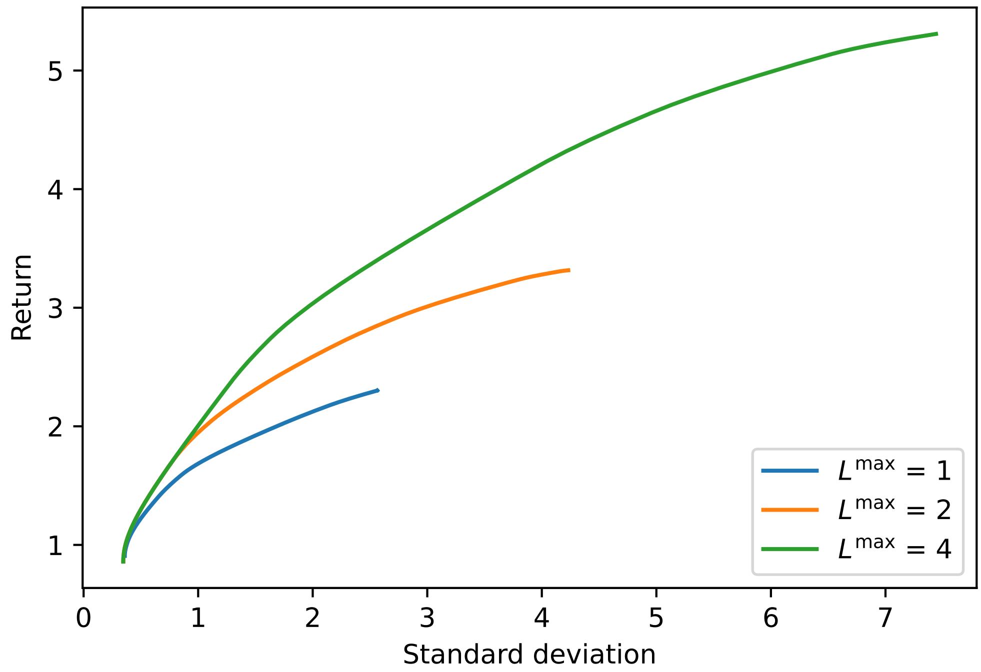
In [ ]:

```
# Compute trade-off curve for each leverage limit.
L_vals = [1, 2, 4]
SAMPLES = 100
risk_data = np.zeros((len(L_vals), SAMPLES))
ret_data = np.zeros((len(L_vals), SAMPLES))
gamma_vals = np.logspace(-2, 3, num=SAMPLES)
w_vals = []
for k, L_val in enumerate(L_vals):
    for i in range(SAMPLES):
        Lmax.value = L_val
        gamma.value = gamma_vals[i]
        prob.solve(solver=cp.SCS)
```

```
        risk_data[k, i] = cp.sqrt(risk).value
        ret_data[k, i] = ret.value
```

In [ ]:

```
# Plot trade-off curves for each leverage limit.
for idx, L_val in enumerate(L_vals):
    plt.plot(risk_data[idx,:], ret_data[idx,:], label=r"$L^{\max}$ = %d" % L_val)
for w_val in w_vals:
    w.value = w_val
    plt.plot(cp.sqrt(risk).value, ret.value, 'bs')
plt.xlabel('Standard deviation')
plt.ylabel('Return')
plt.legend(loc='lower right')
plt.show()
```



We next examine the points on each trade-off curve where $w^T \Sigma w = 2$. We plot the amount of each asset held in each portfolio as bar graphs. (Negative holdings indicate a short position.) Notice that some assets are held in a long position for the low leverage portfolio but in a short position in the higher leverage portfolios.

In [ ]:

```
# Portfolio optimization with a leverage limit and a bound on risk.
prob = cp.Problem(cp.Maximize(ret),
              [cp.sum(w) == 1,
               cp.norm(w, 1) <= Lmax,
               risk <= 2])
```

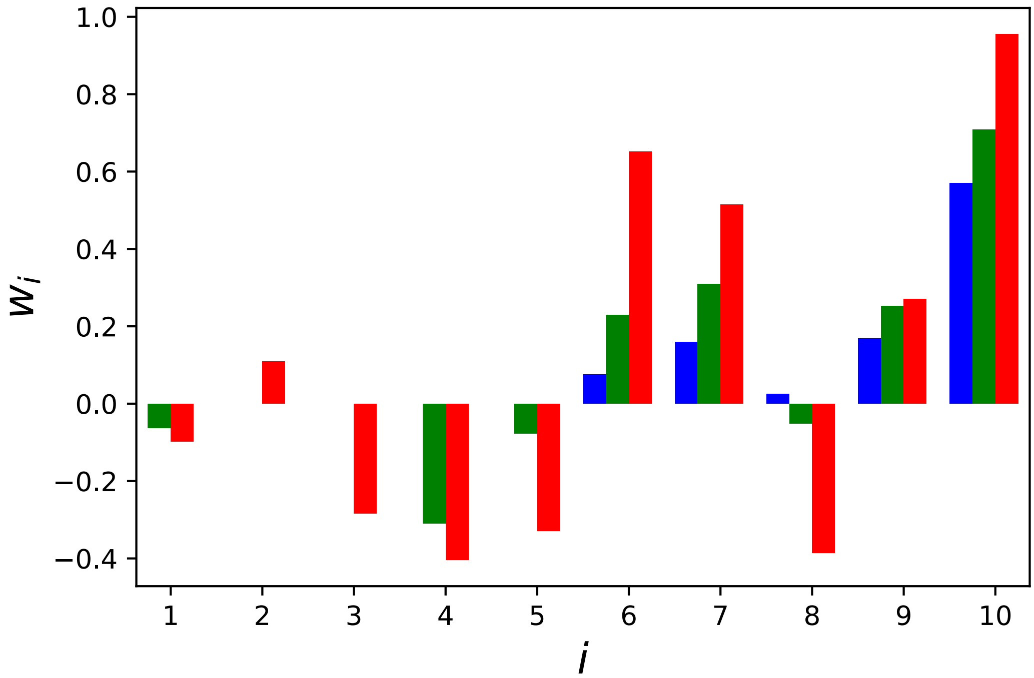
In [ ]:

```
# Compute solution for different leverage limits.
for k, L_val in enumerate(L_vals):
    Lmax.value = L_val
    prob.solve()
    w_vals.append( w.value )
```

In [ ]:

```
# Plot bar graph of holdings for different leverage limits.
colors = ['b', 'g', 'r']
indices = np.argsort(mu.flatten())
for idx, L_val in enumerate(L_vals):
     plt.bar(np.arange(1,n+1) + 0.25*idx - 0.375, w_vals[idx][indices], color=colo
             label=r"$L^{\max}$ = %d" % L_val, width = 0.25)
plt.ylabel(r"$w_i$", fontsize=16)
plt.xlabel(r"$i$", fontsize=16)
plt.xlim([1-0.375, 10+.375])
```

```
plt.xticks(np.arange(1,n+1))
plt.show()
```



Variations

There are many more variations of classical portfolio optimization. We might require that $\mu^T w \geq R^{\min}$ and minimize $w^T \Sigma w$ or $\|\Sigma^{1/2} w\|_2$. We could include the (broker) cost of short positions as the penalty $s^T(w)_-$ for some $s \geq 0$. We could include transaction costs (from a previous portfolio w^{prev}) as the penalty

$$\kappa^T |w - w^{\text{prev}}|^\eta, \quad \kappa \geq 0.$$

Common values of η are $\eta = 1,\ 3/2,\ 2.$

Factor covariance model

A particularly common and useful variation is to model the covariance matrix Σ as a factor model

$$\Sigma = F\tilde{\Sigma}F^T + D,$$

where $F \in \mathbf{R}^{n \times k}$, $k \ll n$ is the *factor loading matrix*. k is the number of factors (or sectors) (typically 10s). F_{ij} is the loading of asset i to factor j. D is a diagonal matrix; $D_{ii} > 0$ is the *idiosyncratic risk*. $\tilde{\Sigma} > 0$ is the *factor covariance matrix*.

$F^T w \in \mathbf{R}^k$ gives the portfolio *factor exposures*. A portfolio is *factor j neutral* if $(F^T w)_j = 0$.

Portfolio optimization with factor covariance model

Using the factor covariance model, we frame the portfolio optimization problem as

$$\begin{array}{ll}
\text{maximize} & \mu^T w - \gamma \left(f^T \tilde{\Sigma} f + w^T D w \right) \\
\text{subject to} & \mathbf{1}^T w = 1, \quad f = F^T w \\
& w \in \mathcal{W}, \quad f \in \mathcal{F},
\end{array}$$

where the variables are the allocations $w \in \mathbf{R}^n$ and factor exposures $f \in \mathbf{R}^k$ and $\mathcal{F}$ gives the factor exposure constraints.

Using the factor covariance model in the optimization problem has a computational advantage. The solve time is $O(nk^2)$ versus $O(n^3)$ for the standard problem.

Example

In the following code we generate and solve a portfolio optimization problem with 50 factors and 3000 assets. We set the leverage limit $= 2$ and $\gamma = 0.1$.

We solve the problem both with the covariance given as a single matrix and as a factor model. Using CVXPY with the OSQP solver running in a single thread, the solve time was 173.30 seconds for the single matrix formulation and 0.85 seconds for the factor model formulation. We collected the timings on a MacBook Air with an Intel Core i7 processor.

In [ ]:

```python
# Generate data for factor model.
n = 3000
m = 50
np.random.seed(1)
mu = np.abs(np.random.randn(n, 1))
Sigma_tilde = np.random.randn(m, m)
Sigma_tilde = Sigma_tilde.T.dot(Sigma_tilde)
D = np.diag(np.random.uniform(0, 0.9, size=n))
F = np.random.randn(n, m)
```

In [ ]:

```python
# Factor model portfolio optimization.
w = cp.Variable(n)
f = F.T*w
gamma = cp.Parameter(nonneg=True)
Lmax = cp.Parameter()
ret = mu.T*w
risk = cp.quad_form(f, Sigma_tilde) + cp.quad_form(w, D)
prob_factor = cp.Problem(cp.Maximize(ret - gamma*risk),
                         [cp.sum(w) == 1,
                          cp.norm(w, 1) <= Lmax])

# Solve the factor model problem.
Lmax.value = 2
gamma.value = 0.1
prob_factor.solve(verbose=True)
```

```
-----------------------------------------------------------------
           OSQP v0.6.2  -  Operator Splitting QP Solver
              (c) Bartolomeo Stellato,  Goran Banjac
        University of Oxford  -  Stanford University 2021
-----------------------------------------------------------------
problem:  variables n = 6050, constraints m = 6052
          nnz(P) + nnz(A) = 172325
settings: linear system solver = qdldl,
          eps_abs = 1.0e-05, eps_rel = 1.0e-05,
          eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
          rho = 1.00e-01 (adaptive),
          sigma = 1.00e-06, alpha = 1.60, max_iter = 10000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: on, time_limit: off

iter   objective    pri res    dua res    rho        time
   1  -2.1359e+03   7.63e+00   3.73e+02   1.00e-01   7.33e-02s
 200  -4.1946e+00   1.59e-03   7.86e-03   3.60e-01   4.82e-01s
 400  -4.6288e+00   3.02e-04   6.01e-04   3.60e-01   8.18e-01s
 600  -4.6444e+00   2.20e-04   7.87e-04   3.60e-01   1.20e+00s
 800  -4.6230e+00   1.09e-04   3.70e-04   3.60e-01   1.60e+00s
1000  -4.6223e+00   8.59e-05   1.04e-04   3.60e-01   1.97e+00s
1200  -4.6205e+00   8.56e-05   9.35e-06   3.60e-01   2.31e+00s
1400  -4.6123e+00   6.44e-05   1.54e-04   3.60e-01   2.67e+00s
```

```
1575  -4.6064e+00   2.97e-05   4.06e-05   3.60e-01   2.99e+00s

status:               solved
solution polish:      unsuccessful
number of iterations: 1575
optimal objective:    -4.6064
run time:             3.03e+00s
optimal rho estimate: 3.87e-01
```

Out[ ]:
```
4.606413081938858
```

In [ ]:
```python
# Standard portfolio optimization with data from factor model.
risk = cp.quad_form(w, F.dot(Sigma_tilde).dot(F.T) + D)
prob = cp.Problem(cp.Maximize(ret - gamma*risk),
               [cp.sum(w) == 1,
                cp.norm(w, 1) <= Lmax])

# Uncomment to solve the problem.
# WARNING: this will take many minutes to run.
prob.solve(verbose=True, max_iter=30000)
```

```
-----------------------------------------------------------------
           OSQP v0.6.2  -  Operator Splitting QP Solver
              (c) Bartolomeo Stellato,  Goran Banjac
        University of Oxford  -  Stanford University 2021
-----------------------------------------------------------------
problem:  variables n = 6000, constraints m = 6002
          nnz(P) + nnz(A) = 4519500
settings: linear system solver = qdldl,
          eps_abs = 1.0e-05, eps_rel = 1.0e-05,
          eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
          rho = 1.00e-01 (adaptive),
          sigma = 1.00e-06, alpha = 1.60, max_iter = 30000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: on, time_limit: off

iter   objective    pri res    dua res    rho        time
   1  -1.1774e+04   2.65e+02   1.51e+04   1.00e-01   7.97e+00s
 200  -4.1080e+02   2.42e-01   8.86e-04   1.00e-01   1.10e+01s
 400  -1.9413e+02   1.13e-01   2.51e-04   1.00e-01   1.41e+01s
 600  -1.2345e+02   6.40e-02   1.09e-04   1.00e-01   1.72e+01s
 800  -8.7560e+01   4.67e-02   5.29e-05   1.00e-01   2.03e+01s
1000  -6.5202e+01   3.49e-02   2.99e-05   1.00e-01   2.34e+01s
1200  -5.0118e+01   2.68e-02   1.91e-05   1.00e-01   2.65e+01s
1400  -3.9737e+01   2.09e-02   1.41e-05   1.00e-01   2.95e+01s
1600  -3.2445e+01   1.72e-02   1.06e-05   1.00e-01   3.26e+01s
1800  -2.6947e+01   1.42e-02   8.27e-06   1.00e-01   3.57e+01s
2000  -2.2700e+01   1.17e-02   6.57e-06   1.00e-01   3.88e+01s
2200  -1.9294e+01   9.74e-03   5.29e-06   1.00e-01   4.19e+01s
2400  -1.6616e+01   8.26e-03   4.32e-06   1.00e-01   4.50e+01s
2600  -1.4460e+01   7.01e-03   3.56e-06   1.00e-01   4.80e+01s
2800  -1.2704e+01   5.95e-03   2.93e-06   1.00e-01   5.11e+01s
3000  -1.1267e+01   5.06e-03   2.43e-06   1.00e-01   5.42e+01s
3200  -1.0092e+01   4.25e-03   2.00e-06   1.00e-01   5.72e+01s
3400  -9.1244e+00   3.58e-03   1.66e-06   1.00e-01   6.04e+01s
3600  -8.3286e+00   3.04e-03   1.38e-06   1.00e-01   6.35e+01s
3800  -7.6760e+00   2.60e-03   1.14e-06   1.00e-01   6.66e+01s
4000  -7.1409e+00   2.26e-03   9.40e-07   1.00e-01   6.96e+01s
4200  -6.7000e+00   2.04e-03   7.81e-07   1.00e-01   7.27e+01s
4400  -6.3366e+00   1.85e-03   6.50e-07   1.00e-01   7.58e+01s
4600  -6.0382e+00   1.69e-03   5.41e-07   1.00e-01   7.89e+01s
4800  -5.7969e+00   1.58e-03   4.54e-07   1.00e-01   8.22e+01s
5000  -5.5953e+00   1.46e-03   3.83e-07   1.00e-01   8.54e+01s
5200  -5.4277e+00   1.37e-03   3.24e-07   1.00e-01   8.84e+01s
5400  -5.2885e+00   1.28e-03   2.73e-07   1.00e-01   9.15e+01s
5600  -5.1729e+00   1.20e-03   2.30e-07   1.00e-01   9.47e+01s
```

5800	-5.0768e+00	1.13e-03	1.94e-07	1.00e-01	9.78e+01s
6000	-4.9968e+00	1.08e-03	1.63e-07	1.00e-01	1.01e+02s
6200	-4.9301e+00	1.02e-03	1.37e-07	1.00e-01	1.04e+02s
6400	-4.8746e+00	9.80e-04	1.18e-07	1.00e-01	1.07e+02s
6600	-4.8281e+00	9.40e-04	1.09e-07	1.00e-01	1.10e+02s
6800	-4.7893e+00	9.04e-04	1.01e-07	1.00e-01	1.13e+02s
7000	-4.7568e+00	8.72e-04	9.40e-08	1.00e-01	1.16e+02s
7200	-4.7295e+00	8.44e-04	8.75e-08	1.00e-01	1.20e+02s
7400	-4.7372e+00	8.63e-04	2.54e-07	1.00e-01	1.23e+02s
7600	-4.7339e+00	8.57e-04	1.41e-07	1.00e-01	1.26e+02s
7800	-4.7278e+00	8.25e-04	8.93e-08	1.00e-01	1.29e+02s
8000	-4.7195e+00	7.99e-04	5.47e-08	1.00e-01	1.32e+02s
8200	-4.7100e+00	7.75e-04	4.25e-08	1.00e-01	1.35e+02s
8400	-4.7002e+00	7.59e-04	3.67e-08	1.00e-01	1.38e+02s
8600	-4.6909e+00	7.51e-04	3.23e-08	1.00e-01	1.41e+02s
8800	-4.6824e+00	7.42e-04	3.05e-08	1.00e-01	1.44e+02s
9000	-4.6749e+00	7.35e-04	2.86e-08	1.00e-01	1.47e+02s
9200	-4.6684e+00	7.27e-04	2.66e-08	1.00e-01	1.51e+02s
9400	-4.6627e+00	7.21e-04	2.47e-08	1.00e-01	1.54e+02s
9600	-4.6577e+00	7.17e-04	2.29e-08	1.00e-01	1.57e+02s
9800	-4.6534e+00	7.13e-04	2.15e-08	1.00e-01	1.60e+02s
10000	-4.6496e+00	7.10e-04	2.03e-08	1.00e-01	1.63e+02s
10200	-4.6463e+00	7.06e-04	1.91e-08	1.00e-01	1.66e+02s
10400	-4.6434e+00	7.03e-04	1.81e-08	1.00e-01	1.69e+02s
10600	-4.6335e+00	6.88e-04	3.27e-07	5.04e-01	1.80e+02s
10800	-4.6280e+00	6.75e-04	2.51e-07	5.04e-01	1.83e+02s
11000	-4.6248e+00	6.64e-04	1.95e-07	5.04e-01	1.86e+02s
11200	-4.6228e+00	6.55e-04	1.54e-07	5.04e-01	1.89e+02s
11400	-4.6218e+00	6.45e-04	8.79e-08	5.04e-01	1.93e+02s
11600	-4.6207e+00	6.44e-04	9.50e-08	5.04e-01	1.96e+02s
11800	-4.6198e+00	6.43e-04	9.48e-08	5.04e-01	1.99e+02s
12000	-4.6190e+00	6.42e-04	9.17e-08	5.04e-01	2.02e+02s
12200	-4.6182e+00	6.41e-04	8.76e-08	5.04e-01	2.05e+02s
12400	-4.6175e+00	6.39e-04	8.34e-08	5.04e-01	2.08e+02s
12600	-4.6175e+00	6.20e-04	3.63e-07	5.04e-01	2.11e+02s
12800	-4.6158e+00	6.17e-04	2.08e-07	5.04e-01	2.14e+02s
13000	-4.6153e+00	6.13e-04	1.47e-07	5.04e-01	2.17e+02s
13200	-4.6148e+00	6.10e-04	1.09e-07	5.04e-01	2.21e+02s
13400	-4.6454e+00	5.54e-04	2.45e-06	5.04e-01	2.24e+02s
13600	-4.6464e+00	5.27e-04	7.63e-07	5.04e-01	2.27e+02s
13800	-4.6382e+00	5.07e-04	5.38e-07	5.04e-01	2.30e+02s
14000	-4.6332e+00	4.89e-04	4.15e-07	5.04e-01	2.33e+02s
14200	-4.6304e+00	4.65e-04	3.03e-07	5.04e-01	2.36e+02s
14400	-4.6286e+00	4.52e-04	2.31e-07	5.04e-01	2.39e+02s
14600	-4.6274e+00	4.41e-04	1.90e-07	5.04e-01	2.42e+02s
14800	-4.6263e+00	4.36e-04	1.57e-07	5.04e-01	2.46e+02s
15000	-4.6254e+00	4.31e-04	1.31e-07	5.04e-01	2.49e+02s
15200	-4.6247e+00	4.27e-04	1.10e-07	5.04e-01	2.52e+02s
15400	-4.6240e+00	4.24e-04	9.37e-08	5.04e-01	2.55e+02s
15600	-4.6234e+00	4.22e-04	8.00e-08	5.04e-01	2.58e+02s
15800	-4.6229e+00	4.21e-04	6.87e-08	5.04e-01	2.61e+02s
16000	-4.6224e+00	4.21e-04	5.93e-08	5.04e-01	2.64e+02s
16200	-4.6220e+00	4.21e-04	5.14e-08	5.04e-01	2.67e+02s
16400	-4.6217e+00	4.21e-04	4.48e-08	5.04e-01	2.70e+02s
16600	-4.6213e+00	4.20e-04	3.92e-08	5.04e-01	2.73e+02s
16800	-4.6210e+00	4.20e-04	3.44e-08	5.04e-01	2.77e+02s
17000	-4.6208e+00	4.20e-04	2.84e-08	5.04e-01	2.80e+02s
17200	-4.6207e+00	4.19e-04	2.37e-08	5.04e-01	2.83e+02s
17400	-4.6205e+00	4.18e-04	2.00e-08	5.04e-01	2.86e+02s
17600	-4.6203e+00	4.18e-04	1.82e-08	5.04e-01	2.89e+02s
17800	-4.6202e+00	4.17e-04	1.72e-08	5.04e-01	2.92e+02s
18000	-4.6200e+00	4.16e-04	1.64e-08	5.04e-01	2.95e+02s
18200	-4.6256e+00	4.14e-04	9.55e-07	5.04e-01	2.98e+02s
18400	-4.6227e+00	4.15e-04	3.35e-07	5.04e-01	3.01e+02s
18600	-4.6224e+00	4.16e-04	1.95e-07	5.04e-01	3.04e+02s
18800	-4.6226e+00	4.16e-04	1.27e-07	5.04e-01	3.07e+02s
19000	-4.6229e+00	4.15e-04	8.84e-08	5.04e-01	3.11e+02s
19200	-4.6231e+00	4.15e-04	6.51e-08	5.04e-01	3.14e+02s

```
19400  -4.6233e+00   4.14e-04   5.14e-08   5.04e-01   3.17e+02s
19600  -4.6235e+00   4.14e-04   4.14e-08   5.04e-01   3.20e+02s
19800  -4.6236e+00   4.14e-04   3.39e-08   5.04e-01   3.23e+02s
20000  -4.6236e+00   4.13e-04   2.83e-08   5.04e-01   3.26e+02s
20200  -4.6237e+00   4.13e-04   2.40e-08   5.04e-01   3.29e+02s
20400  -4.6279e+00   4.19e-04   1.10e-06   5.04e-01   3.32e+02s
20600  -4.6328e+00   4.14e-04   4.43e-07   5.04e-01   3.35e+02s
20800  -4.6348e+00   4.09e-04   3.19e-07   5.04e-01   3.38e+02s
21000  -4.6360e+00   4.06e-04   2.50e-07   5.04e-01   3.41e+02s
21200  -4.6368e+00   4.03e-04   2.00e-07   5.04e-01   3.45e+02s
21400  -4.6375e+00   4.00e-04   1.62e-07   5.04e-01   3.48e+02s
21600  -4.6380e+00   3.99e-04   1.40e-07   5.04e-01   3.51e+02s
21800  -4.6386e+00   3.98e-04   1.18e-07   5.04e-01   3.54e+02s
22000  -4.6392e+00   3.98e-04   1.00e-07   5.04e-01   3.57e+02s
22200  -4.6396e+00   3.97e-04   9.12e-08   5.04e-01   3.60e+02s
22400  -4.6402e+00   3.88e-04   1.72e-06   5.04e-01   3.63e+02s
22600  -4.6481e+00   3.69e-04   6.62e-07   5.04e-01   3.66e+02s
22800  -4.6513e+00   3.68e-04   3.75e-07   5.04e-01   3.70e+02s
23000  -4.6578e+00   3.65e-04   1.26e-06   5.04e-01   3.73e+02s
23200  -4.6606e+00   3.71e-04   3.81e-07   5.04e-01   3.76e+02s
23400  -4.6574e+00   3.69e-04   1.74e-07   5.04e-01   3.79e+02s
23600  -4.6563e+00   3.55e-04   3.46e-07   5.04e-01   3.82e+02s
23800  -4.6543e+00   3.48e-04   1.62e-07   5.04e-01   3.85e+02s
24000  -4.6534e+00   3.46e-04   9.43e-08   5.04e-01   3.89e+02s
24200  -4.6530e+00   3.45e-04   7.96e-08   5.04e-01   3.92e+02s
24400  -4.6575e+00   3.44e-04   6.80e-07   5.04e-01   3.95e+02s
24600  -4.6604e+00   3.40e-04   4.12e-07   5.04e-01   3.98e+02s
24800  -4.6596e+00   3.36e-04   2.92e-07   5.04e-01   4.02e+02s
25000  -4.6582e+00   3.30e-04   2.22e-07   5.04e-01   4.05e+02s
25200  -4.6570e+00   3.26e-04   1.74e-07   5.04e-01   4.08e+02s
25400  -4.6560e+00   3.21e-04   1.40e-07   5.04e-01   4.11e+02s
25600  -4.6552e+00   3.18e-04   1.21e-07   5.04e-01   4.14e+02s
25800  -4.6545e+00   3.15e-04   1.09e-07   5.04e-01   4.17e+02s
26000  -4.6540e+00   3.12e-04   9.83e-08   5.04e-01   4.20e+02s
26200  -4.6536e+00   3.10e-04   8.83e-08   5.04e-01   4.23e+02s
26400  -4.6644e+00   2.98e-04   8.13e-07   5.04e-01   4.26e+02s
26600  -4.6618e+00   2.91e-04   3.47e-07   5.04e-01   4.29e+02s
26800  -4.6579e+00   2.86e-04   2.42e-07   5.04e-01   4.33e+02s
27000  -4.6555e+00   2.84e-04   2.09e-07   5.04e-01   4.36e+02s
27200  -4.6539e+00   2.85e-04   1.89e-07   5.04e-01   4.39e+02s
27400  -4.6529e+00   2.85e-04   1.70e-07   5.04e-01   4.42e+02s
27600  -4.6522e+00   2.85e-04   1.53e-07   5.04e-01   4.45e+02s
27800  -4.6517e+00   2.84e-04   1.38e-07   5.04e-01   4.48e+02s
28000  -4.6513e+00   2.84e-04   1.24e-07   5.04e-01   4.51e+02s
28200  -4.6510e+00   2.84e-04   1.12e-07   5.04e-01   4.54e+02s
28400  -4.6507e+00   2.83e-04   1.01e-07   5.04e-01   4.57e+02s
28600  -4.6504e+00   2.82e-04   9.15e-08   5.04e-01   4.61e+02s
28800  -4.6502e+00   2.82e-04   8.31e-08   5.04e-01   4.64e+02s
29000  -4.6499e+00   2.81e-04   7.57e-08   5.04e-01   4.67e+02s
29200  -4.6497e+00   2.80e-04   6.92e-08   5.04e-01   4.70e+02s
29400  -4.6495e+00   2.80e-04   6.34e-08   5.04e-01   4.73e+02s
29600  -4.6493e+00   2.79e-04   5.82e-08   5.04e-01   4.77e+02s
29800  -4.6491e+00   2.79e-04   5.36e-08   5.04e-01   4.80e+02s
30000  -4.6489e+00   2.78e-04   4.94e-08   5.04e-01   4.83e+02s

status:               solved inaccurate
number of iterations: 30000
optimal objective:    -4.6489
run time:             4.83e+02s
optimal rho estimate: 9.34e-01
```

Out[ ]:
```
4.64886481797565
```

In [ ]:
```python
print('Factor model solve time = {}'.format(prob_factor.solver_stats.solve_time))
print('Single model solve time = {}'.format(prob.solver_stats.solve_time))
```

```
Factor model solve time = 3.0333686589999997
```

```
Single model solve time = 483.077024319
```

Как с деньгами обстоит вопрос

What about real data?

In [ ]:

```
!pip install yfinance
# !pip install fix_yahoo_finance
```

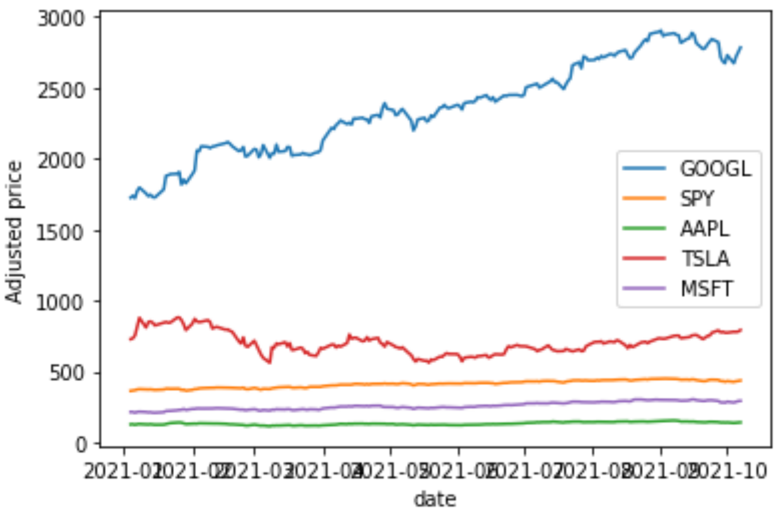
```
Collecting yfinance
  Downloading yfinance-0.1.63.tar.gz (26 kB)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packa
ges (from yfinance) (1.1.5)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packag
es (from yfinance) (1.19.5)
Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.7/dist-pac
kages (from yfinance) (2.23.0)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dis
t-packages (from yfinance) (0.0.9)
Collecting lxml>=4.5.1
  Downloading lxml-4.6.3-cp37-cp37m-manylinux2014_x86_64.whl (6.3 MB)
     |████████████████████████████████| 6.3 MB 11.7 MB/s
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packa
ges (from pandas>=0.24->yfinance) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/
dist-packages (from pandas>=0.24->yfinance) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages
(from python-dateutil>=2.7.3->pandas>=0.24->yfinance) (1.15.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist
-packages (from requests>=2.20->yfinance) (2021.5.30)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/loc
al/lib/python3.7/dist-packages (from requests>=2.20->yfinance) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
packages (from requests>=2.20->yfinance) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packa
ges (from requests>=2.20->yfinance) (2.10)
```

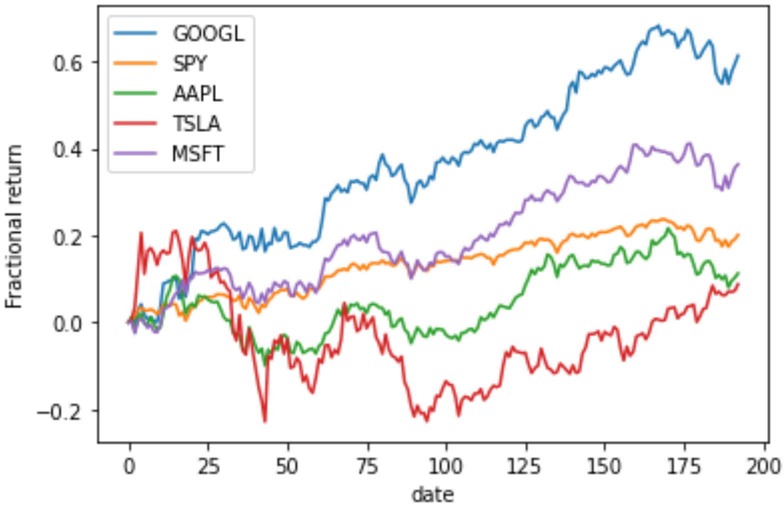
```
Building wheels for collected packages: yfinance
  Building wheel for yfinance (setup.py) ... done
  Created wheel for yfinance: filename=yfinance-0.1.63-py2.py3-none-any.whl size=2
3918 sha256=0cddfaff1dea1d7576ce522315873d91405fa976a0ec086d71156241ae5de968
  Stored in directory: /root/.cache/pip/wheels/fe/87/8b/7ec24486e001d3926537f5f780
1f57a74d181be25b11157983
Successfully built yfinance
Installing collected packages: lxml, yfinance
  Attempting uninstall: lxml
    Found existing installation: lxml 4.2.6
    Uninstalling lxml-4.2.6:
      Successfully uninstalled lxml-4.2.6
Successfully installed lxml-4.6.3 yfinance-0.1.63
```

In [ ]:

```
import datetime
import matplotlib.pyplot as plt
import pandas
from pandas_datareader import data as pdr
import yfinance as yfin
yfin.pdr_override()

stocks = ['GOOGL', 'SPY', 'AAPL', 'TSLA', 'MSFT']

df = pdr.get_data_yahoo(stocks, start="2021-01-01")

# Adjusted price
for stock in stocks:
    plt.plot(df['Adj Close'][stock], label=stock)
plt.xlabel('date')
plt.ylabel('Adjusted price')
plt.legend()
plt.show()

# Fractional return
frac_return = {}
for stock in stocks:
    frac_return[stock] = [(price - df['Adj Close'][stock][0])/df['Adj Close'][stoc
    plt.plot(frac_return[stock], label=stock)
plt.xlabel('date')
plt.ylabel('Fractional return')
plt.legend()
plt.show()
```

```
[*********************100%***********************]  5 of 5 completed
```





In [ ]:

```
# Calculating mean and covariance of fractional return
# number of stocks
N = len(frac_return)

# number of historical values per stock
M = len(frac_return[stocks[0]])

mu = np.zeros(N)
Sigma = np.zeros((N,N))
Prices = np.zeros((M,N))

for i_asset, (stock, return_array) in enumerate(frac_return.items()):
    mu[i_asset] = np.array(return_array).mean()
    Prices[:, i_asset] = return_array

Sigma = 1/N*(Prices - mu).T @ (Prices - mu)
```

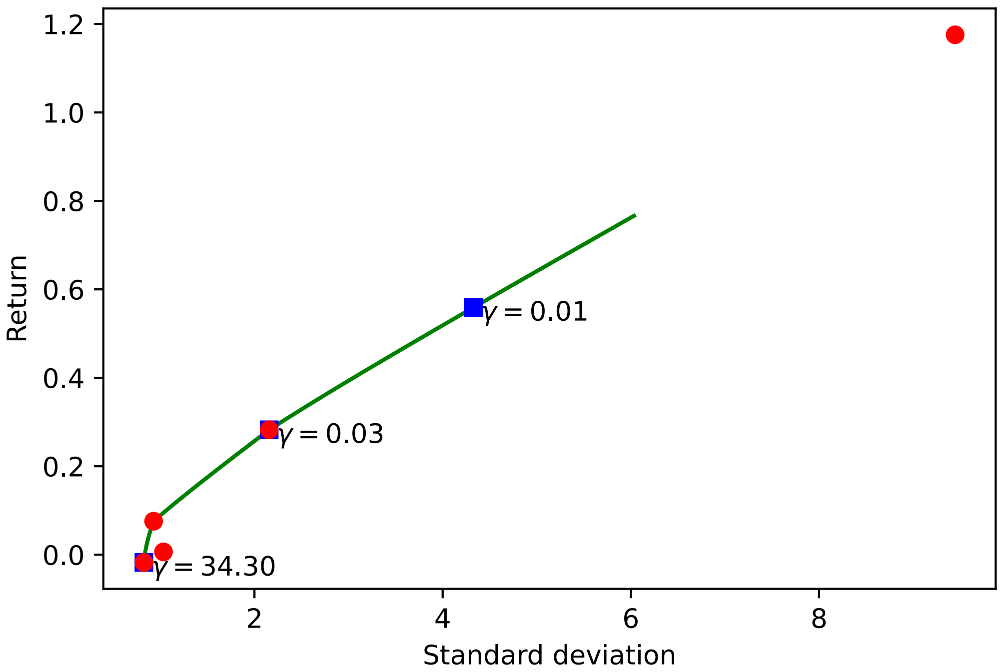
In [ ]:

```
# Long only portfolio optimization.
import cvxpy as cp


w = cp.Variable(N)
gamma = cp.Parameter(nonneg=True)
ret = mu.T@w
risk = cp.quad_form(w, Sigma)
prob = cp.Problem(cp.Maximize(ret - gamma*risk),
               [cp.sum(w) == 1,
                w >= 0])
```

In [ ]:

```
# Compute trade-off curve.
SAMPLES = 100
risk_data = np.zeros(SAMPLES)
ret_data = np.zeros(SAMPLES)
gamma_vals = np.logspace(-2, 3, num=SAMPLES)
for i in range(SAMPLES):
    gamma.value = gamma_vals[i]
    prob.solve()
    risk_data[i] = cp.sqrt(risk).value
    ret_data[i] = ret.value

# Plot long only trade-off curve.
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

markers_on = [3, 10, 70]
fig = plt.figure()
ax = fig.add_subplot(111)
```

```
plt.plot(risk_data, ret_data, 'g-')
for marker in markers_on:
    plt.plot(risk_data[marker], ret_data[marker], 'bs')
    ax.annotate(r"$\gamma = %.2f$" % gamma_vals[marker], xy=(risk_data[marker]+.08
for i in range(N):
    plt.plot(cp.sqrt(Sigma[i,i]).value, mu[i], 'ro')
plt.xlabel('Standard deviation')
plt.ylabel('Return')
plt.show()
```



In [ ]:

```
# Long only portfolio optimization.
import cvxpy as cp


w = cp.Variable(N)
gamma = 10
ret = mu.T@w
risk = cp.quad_form(w, Sigma)
prob = cp.Problem(cp.Maximize(ret - gamma*risk),
                [cp.sum(w) == 1,
                 w >= 0])
prob.solve(verbose=True)

print(f'{[(stock, x) for stock, x in zip(stocks, w.value)]}')
```

```
-----------------------------------------------------------------
           OSQP v0.6.2  -  Operator Splitting QP Solver
              (c) Bartolomeo Stellato,  Goran Banjac
        University of Oxford  -  Stanford University 2021
-----------------------------------------------------------------
problem:  variables n = 5, constraints m = 6
          nnz(P) + nnz(A) = 25
settings: linear system solver = qdldl,
          eps_abs = 1.0e-05, eps_rel = 1.0e-05,
          eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
          rho = 1.00e-01 (adaptive),
          sigma = 1.00e-06, alpha = 1.60, max_iter = 10000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: on, time_limit: off

iter   objective    pri res    dua res    rho        time
   1  -2.0945e-03   1.00e+00   1.36e+03   1.00e-01   2.08e-04s
  50   6.8270e+00   5.75e-06   1.96e-04   8.09e-01   5.78e-04s
plsh   6.8271e+00   2.14e-22   0.00e+00   --------   9.09e-04s
```

```
status:                 solved
solution polish:        successful
number of iterations: 50
optimal objective:      6.8271
run time:               9.09e-04s
optimal rho estimate: 5.42e-01

[('GOOGL', -2.140038590828118e-22), ('SPY', 0.9999999999999993), ('AAPL', 4.435850
852504018e-23), ('TSLA', 1.0105310029397988e-23), ('MSFT', -8.623781472977922e-2
3)]
```

In [ ]:

```python
[(stock, x) for (stock, x) in zip(stocks, w.value)]
```

Out[ ]:

```
[('GOOGL', -1.2310333739669261e-23),
 ('SPY', 1.0705162474990867e-22),
 ('AAPL', 0.5995001807644814),
 ('TSLA', -1.5763444272763734e-23),
 ('MSFT', 0.4004998192355186)]
```

Materials

- Portfolio Optimization Algo Trading colab notebook
- Multi objective portfolio optimization