Convex Optimization in Quantitative Finance

Stephen Boyd Kasper Johansson Philipp Schiele

Stanford University

June 11, 2024

Overview

convex optimization problems

- are a special type of mathematical optimization problem
- can be efficiently solved
- are easily specified using domain specific languages such as CVXPY
- can be used to solve a wide variety of problems arising in finance

these slides give many examples in finance

- our examples are simplified, but readily extended
- we give code snippets for all of them
- full code is available at `https://github.com/cvxgrp/cvx-finance-examples`

Outline

Convex optimization

Markowitz portfolio construction

Maximum expected utility portfolio construction

Sparse inverse covariance estimation

Worst-case risk analysis

Option pricing

Currency exchange

Optimal execution

Optimal consumption

Alternative investment planning

Blending forecasts

Bond pricing

Model predictive control

Outline

Convex optimization

Markowitz portfolio construction

Maximum expected utility portfolio construction

Sparse inverse covariance estimation

Worst-case risk analysis

Option pricing

Currency exchange

Optimal execution

Optimal consumption

Alternative investment planning

Blending forecasts

Bond pricing

Model predictive control

Optimization problem

$$
\begin{array}{lll}
\text{minimize} & f_0(x) & \\
\text{subject to} & f_i(x) \leq 0, & i = 1, \dots, m \\
 & g_i(x) = 0, & i = 1, \dots, p
\end{array}
$$

- $x \in \mathbf{R}^n$ is (vector) variable to be chosen
- f_0 is the *objective function*, to be minimized
- $f_1, \dots, f_m$ are the inequality constraint functions
- $g_1, \dots, g_p$ are the equality constraint functions

- variations: maximize objective, multiple objectives, ...

Convex optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{array}$$

- variable $x \in \mathbf{R}^n$
- equality constraints are **linear**
- $f_0, \ldots, f_m$ are **convex**: for $\theta \in [0, 1]$,

$$f_i(\theta x + (1 - \theta)y) \leq \theta f_i(x) + (1 - \theta)f_i(y)$$

  i.e., f_i have nonnegative (upward) curvature

- variations: maximize concave objective, multiple convex objectives, ...

Why

for convex optimization problems there are

- effective algorithms
  - get **global solution** (and optimality certificate)
  - theory: polynomial complexity
  - practice: fast and reliable (no need to tune parameters)
  - many open source and commercial implementations
- many applications in machine learning, signal processing, statistics, control, engineering design, and **finance**

Modeling languages

- high level language support for convex optimization
  - describe problem in high level language
  - simple syntax rules to certify problem convexity
  - description automatically transformed to a standard form
  - solved by standard solver, transformed back to original form
- implementations:
  - CVXPY (Python)
  - YALMIP, CVX (Matlab)
  - Convex.jl (Julia)
  - CVXR (R)
- can be coupled with open source or commercial solvers
- work well for problems up to around 100k variables

CVXPY

a modeling language in Python for convex optimization

- developed since 2014
- open source all the way to the solvers
- syntax very similar to NumPy
- used in many research projects, courses, companies
- tens of thousands of users, including many in finance
- over 27,000,000 downloads on PyPI
- many extensions available

Example

regularized least squares problem with bounds:

$$\begin{array}{ll} \text{minimize} & \|Ax-b\|_2^2 + \gamma\|x\|_1 \\ \text{subject to} & \|x\|_\infty \leq 1 \end{array}$$

CVXPY specification:

```
import cvxpy as cp
x = cp.Variable(n)
cost = cp.sum_squares(A@x-b) + gamma*cp.norm(x,1)
prob = cp.Problem(cp.Minimize(cost),[cp.norm(x,"inf")<=1])
opt_val = prob.solve()
solution = x.value
```

- `A`, `b`, `gamma` are constants, `gamma` nonnegative
- `solve` method converts problem to standard form, solves, assigns `value` attributes

Outline

Convex optimization

Markowitz portfolio construction

Maximum expected utility portfolio construction

Sparse inverse covariance estimation

Worst-case risk analysis

Option pricing

Currency exchange

Optimal execution

Optimal consumption

Alternative investment planning

Blending forecasts

Bond pricing

Model predictive control

Mean-variance (Markowitz) optimization

$$\begin{array}{ll} \text{maximize} & \mu^T w \\ \text{subject to} & w^T \Sigma w \leq (\sigma^{\text{tar}})^2, \quad \mathbf{1}^T w = 1 \end{array}$$

- variable $w \in \mathbf{R}^n$ of portfolio weights
- $\mu \in \mathbf{R}^n$ and $\Sigma \in \mathbf{S}^n_{++}$ are asset return mean and covariance
- σ^{tar} is target (per period) volatility
- basic form goes back to [Markowitz, 1952]

```
w = cp.Variable(n)
objective = mu.T @ w
constraints = [cp.quad_form(w, Sigma) <= sigma**2, cp.sum(w) == 1]
prob = cp.Problem(cp.Maximize(objective), constraints)
prob.solve()
```

Adding practical constraints and objective terms

- include cash holdings c, previous holdings w^{pre}, trades $z = w - w^{\text{pre}}$
- account for (convex) holding costs ϕ^{hold} and trading costs ϕ^{trade}
- limit weights, cash, trades, turnover $T = \|z\|_1$, and leverage $L = \|w\|_1$

$$
\begin{array}{ll}
\text{maximize} & \mu^T w - \gamma^{\text{hold}} \phi^{\text{hold}}(w, c) - \gamma^{\text{trade}} \phi^{\text{trade}}(z) \\
\text{subject to} & \mathbf{1}^T w + c = 1, \quad z = w - w^{\text{pre}}, \\
& w^{\text{min}} \leq w \leq w^{\text{max}}, \quad c^{\text{min}} \leq c \leq c^{\text{max}}, \quad L \leq L^{\text{tar}}, \\
& z^{\text{min}} \leq z \leq z^{\text{max}}, \quad T \leq T^{\text{tar}}, \\
& \|\Sigma^{1/2} w\|_2 \leq \sigma^{\text{tar}}
\end{array}
$$

- variation: soften constraints, *i.e.*, penalize violations
- can be implemented in around ten lines in CVXPY
- see [Boyd et al., 2024] for details and reference implementation

Factor covariance model

$$\Sigma = F\Sigma^{\mathrm{f}}F^T + D$$

- $F \in \mathbf{R}^{n \times k}$ is matrix of factor loadings
- k is number of factors, typically with $k \ll n$
- Σ^{f} is $k \times k$ factor covariance matrix
- D is diagonal matrix of unexplained (idiosyncratic) variances
- a strong regularizer which can give better return covariance estimates

Exploiting a factor model

- with factor model, cost of portfolio optimization reduced from $O(n^3)$ to $O(nk^2)$ flops [Boyd and Vandenberghe, 2004]
- easily exploited in CVXPY
- timings for Clarabel open source solver:

		solve time (s)	
assets n	factors k	factor model	full covariance
100	10	0.002	0.040
300	20	0.010	0.700
1000	30	0.080	25.600
3000	50	0.600	460.000

Backtesting

- fast solve time enables backtesting of strategy variations
  - what-if analysis
  - sensitivity analysis
  - hyperparameter tuning
- with 1000 assets and 30 factors, we can backtest 3 years of daily trading in a minute
- in one hour, we can carry out 2000 3 year backtests on a 32-core machine

Robustifying Markowitz

- basic mean-variance optimization can be sensitive to estimation errors in μ, Σ
- replace mean return $\mu^T w$ with worst-case return

$$R^{\text{wc}} = \min\{(\mu + \delta)^T w \mid |\delta| \leq \rho\} = \mu^T w - \rho^T |w|$$

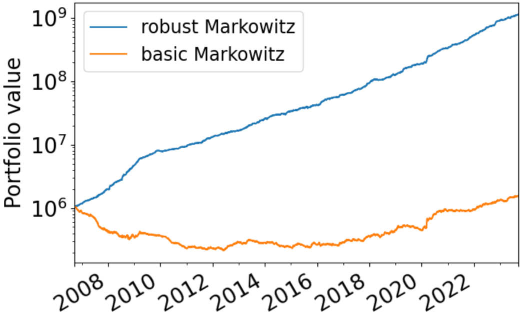
  where $\rho \geq 0$ is vector of mean return uncertainties
- replace risk $w^T \Sigma w$ with worst-case risk

$$\begin{aligned} \left(\sigma^{\text{wc}}\right)^2 &= \max\{w^T(\Sigma + \Delta)w \mid |\Delta_{ij}| \leq \varrho(\Sigma_{ii}\Sigma_{jj})^{1/2}\} \\ &= \sigma^2 + \varrho\left(\sum_{i=1}^n \Sigma_{ii}^{1/2}|w_i|\right)^2 \end{aligned}$$

  where $\varrho \geq 0$ gives covariance uncertainty
- easily handled by CVXPY

Example

- S&P 100, simulated but realistic μ, target annualized risk 10%
- hyper-parameters tuned each year based on previous two years
- out-of-sample portfolio performance for basic Markowitz and robust Markowitz
- Sharpe ratios 0.2 and 4.6 (using the same mean and covariance)



Outline

Convex optimization

Markowitz portfolio construction

Maximum expected utility portfolio construction

Sparse inverse covariance estimation

Worst-case risk analysis

Option pricing

Currency exchange

Optimal execution

Optimal consumption

Alternative investment planning

Blending forecasts

Bond pricing

Model predictive control

Expected utility maximization

- asset returns $r \in \mathbf{R}^n$; portfolio weights $w \in \mathbf{R}^n$
- portfolio return $r^T w$; wealth grows by factor $1 + r^T w$
- expected utility is $\mathbf{E}\, U(1 + r^T w)$, where U is concave increasing utility function
- choose portfolio weights $w \in \mathcal{W}$ (a convex set) to maximize expected utility
- a convex optimization problem [Von Neumann and Morgenstern, 1947]
- reduces to mean-variance in some cases (*e.g.*, exponential utility, Gaussian returns) [Markowitz and Blay, 2014; Luxenberg and Boyd, 2024]
- allows handling of options, nonlinear payoffs, ...
- with $U(x) = \log x$ we get Kelly gambling [Kelly, 1956]; maximizes wealth growth rate

Sample based approximation

- when $\mathbf{E}\, U(1 + r^T w)$ can't be expressed analytically, use sample based approximation
- generate N samples $r_1, \ldots, r_N$, with probabilities $\pi_1, \ldots, \pi_N$
- approximate expected utility as $\mathbf{E}\, U(1 + r^T w) \approx \sum_{i=1}^N \pi_i U(1 + r_i^T w)$
- sample based approximate expected utility maximization:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^N \pi_i U(1 + r_i^T w) \\ \text{subject to} & w \in \mathcal{W} \end{array}$$

- easily handled by CVXPY

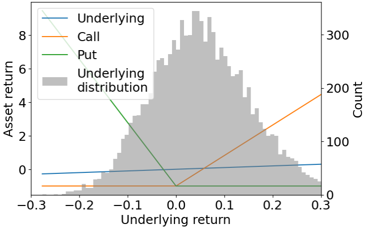
Sample based approximation in CVXPY

- returns r_i are columns of $N \times n$ array `returns`
- probabilities π are in array `probabilities`
- CRRA utility with relative risk aversion $\rho \geq 0$, $U(x) = (x^{1-\rho} - 1)/(1 - \rho)$

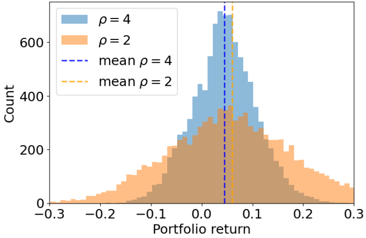
```
def U(x):
    return (x**(1-rho) - 1)/(1-rho)

w = cp.Variable(n)

objective = probabilities @ U(1 + returns @ w)
constraints = [cp.sum(w) == 1]

prob = cp.Problem(cp.Maximize(objective), constraints)
prob.solve()
```

Example

- optimize portfolio of one underlying, one call, and one put, both at-the-money
- underlying with $1 + r$ log-normal
- CRRA utility with relative risk aversion ρ, $\mathcal{W} = \{w \mid \mathbf{1}^T w = 1\}$
- sample approximation with $N = 10^5$ samples



Asset returns



Portfolio return distributions

Outline

Convex optimization

Markowitz portfolio construction

Maximum expected utility portfolio construction

Sparse inverse covariance estimation

Worst-case risk analysis

Option pricing

Currency exchange

Optimal execution

Optimal consumption

Alternative investment planning

Blending forecasts

Bond pricing

Model predictive control

Sparse inverse covariance estimation

- model return in period t as $r_t \sim \mathcal{N}(0, \Sigma)$
- log-likelihood

$$l_t(\theta) = \frac{1}{2}\left(-n\log(2\pi) + \log\det\theta - r_t^T\theta r_t\right),$$

  where $\theta = \Sigma^{-1}$ is the precision matrix
- sparse inverse covariance estimation problem [Friedman et al., 2007]

$$\begin{array}{ll} \text{maximize} & \sum_{t=1}^T l_t(\theta) - \lambda\sum_{i<j}|\theta_{ij}| \\ \text{subject to} & \theta \succeq 0 \end{array}$$

  with variable θ; $\lambda > 0$ is a (sparsity) regularization parameter
- a convex problem; yields matrix with sparse precision matrix θ
- $\theta_{ij} = 0$ means returns $(r_t)_i$, $(r_t)_j$ are conditionally independent given the others

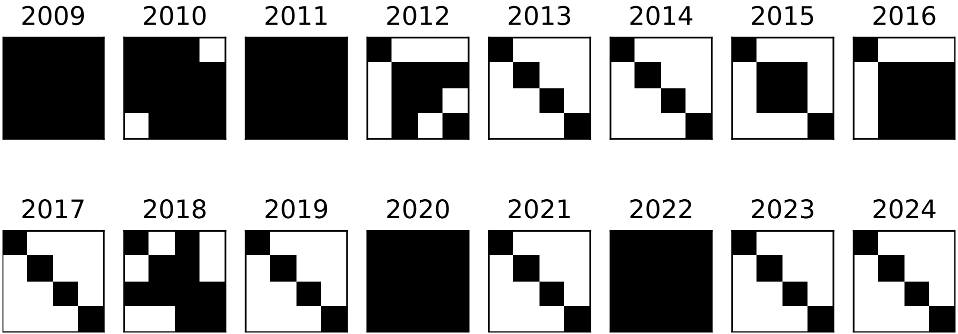
Sparse inverse covariance estimation in CVXPY

- `log_likelihood` is sum of log-likelihoods up to positive scaling and additive constant

```
Theta = cp.Variable((n, n), PSD=True)

log_likelihood = cp.sum(cp.hstack(
    [cp.log_det(Theta) - cp.quad_form(r, Theta) for r in returns]
))
mask = np.triu(np.ones((n, n)), k=1).astype(bool)
objective = log_likelihood - alpha * cp.norm1(Theta[mask])

prob = cp.Problem(cp.Maximize(objective))
prob.solve()
```

Example

- daily returns of US, Europe, Asia, and Africa stock indices from 2009 to 2024
- figure shows yearly sparsity pattern of inverse covariance; white boxes denote zero entries



Outline

Worst-case portfolio risk

- we hold n assets with weights $w \in \mathbf{R}^n$, $\mathbf{1}^T w = 1$
- variance of portfolio return is $w^T \Sigma w$, where $\Sigma \in \mathbf{R}^{n \times n}$ is the return covariance
- now suppose $\Sigma \in \mathcal{S}$ but otherwise uncertain
- set of possible covariances $\mathcal{S}$ is a convex set, *e.g.*,

$$\mathcal{S} = \left\{ \Sigma \geq 0 \mid L_{ij} \leq \Sigma_{ij} \leq U_{ij}, \quad i,j = 1, \ldots, n \right\}$$

  where L and U are lower and upper bounds on entries
- the **worst-case variance** consistent with our belief $\Sigma \in \mathcal{S}$ is

$$\sigma^2_{\text{wc}} = \sup \left\{ w^T \Sigma w \mid \Sigma \geq 0,\ \Sigma \in \mathcal{S} \right\}$$

- evaluating σ^2_{wc} is a convex optimization problem

Worst-case portfolio risk in CVXPY

- `weights` denote portfolio weights
- `L` and `U` are matrices of lower and upper bounds on covariances

```
Sigma = cp.Variable((n, n), PSD=True)

objective = cp.Maximize(cp.quad_form(weights, Sigma))
constraints = []
for i in range(n):
    for j in range(i):
        constraints += [L[i, j] <= Sigma[i, j], Sigma[i, j] <= U[i, j]]

prob = cp.Problem(objective, constraints)
prob.solve()
```

Example

- portfolio weights and uncertain covariance

$$w = \begin{bmatrix} 0.5 \\ 0.25 \\ -0.05 \\ 0.3 \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} 0.2 & + & + & \pm \\ + & 0.1 & - & - \\ + & - & 0.3 & + \\ \pm & - & + & 0.1 \end{bmatrix},$$

- $+$ means nonnegative, $-$ means nonpositive, and $\pm$ means unknown sign
- worst-case risk is 0.18 (volatility 42%)
- risk with diagonal covariance matrix is 0.07 (volatility 26%)
- worst-case covariance is

$$\begin{bmatrix} 0.20 & 0.14 & -0.24 & 0.14 \\ 0.14 & 0.10 & -0.17 & 0.10 \\ -0.24 & -0.17 & 0.30 & -0.17 \\ 0.14 & 0.10 & -0.17 & 0.10 \end{bmatrix}$$

Outline

Convex optimization

Markowitz portfolio construction

Maximum expected utility portfolio construction

Sparse inverse covariance estimation

Worst-case risk analysis

Option pricing

Currency exchange

Optimal execution

Optimal consumption

Alternative investment planning

Blending forecasts

Bond pricing

Model predictive control

Investment arbitrage

- invest x_j in asset j, with prices $p_1, \ldots, p_n$; initial cost is $p^T x$
- at the end of the investment period there are only m possible outcomes
- V_{ij} is the payoff of asset j in outcome i
- first investment is risk-free (cash): $p_1 = 1$ and $V_{i1} = 1$ for all i
- **arbitrage**: there is an x with $p^T x < 0$, $Vx \geq 0$
- *i.e.*, we receive money up front, and cannot lose

- standard assumption: the prices are such that **there is no arbitrage**

Fundamental theorem of asset pricing

- by Farkas' lemma, there is no arbitrage $\Longleftrightarrow$ there exists $\pi \in \mathbf{R}_+^m$ with $V^T \pi = p$
- first column of V is $\mathbf{1}$, so we have $\mathbf{1}^T \pi = 1$
- π is interpreted as a **risk-neutral probability** on the outcomes $1, \ldots, m$
- $V^T \pi$ are the expected values of the payoffs under the risk-neutral probability
- $V^T \pi = p$ means asset prices equal their expected payoff under the risk-neutral probability

- fundamental theorem of asset pricing:

  there is no arbitrage $\Longleftrightarrow$ *there exists a risk-neutral probability distribution under which each asset price is its expected payoff*

Check for arbitrage in CVXPY

```
pi = cp.Variable(m, nonneg=True)
prob = cp.Problem(cp.Minimize(0), [V.T @ pi == p])
prob.solve()

if prob.status == 'optimal':
    print('No arbitrage exists')
elif prob.status == 'infeasible':
    print('Arbitrage exists')
```

Option price bounds

- suppose $p_1, \ldots, p_{n-1}$ are known, but p_n is unknown
- arbitrage-free range for p_n is found by solving

$$\begin{array}{ll} \text{minimize/maximize} & p_n \\ \text{subject to} & V^T\pi = p, \quad \pi \geq 0, \quad \mathbf{1}^T\pi = 1 \end{array}$$

  with variables $p_n \in \mathbf{R}$ and $\pi \in \mathbf{R}^m$
- can be solved in CVXPY
- if the minimum and maximum are equal, the market is **complete**

Option price bounds in CVXPY

- p_known is vector of known prices, of length $n - 1$

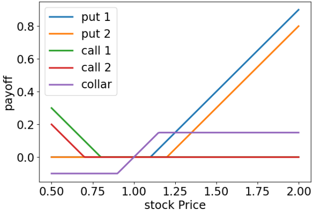
```
pi = cp.Variable(m, nonneg=True)
p_n = cp.Variable()
p = cp.hstack([p_known, p_n])

prob = cp.Problem(cp.Minimize(p_n), [V.T @ pi == p])
prob.solve()
print(f'Minimum arbitrage-free price: {p_n.value}')

prob = cp.Problem(cp.Maximize(p_n), [V.T @ pi == p])
prob.solve()
print(f'Maximum arbitrage-free price: {p_n.value}')
```

Example

- $n = 7$ assets:
  - a risk-free asset with price 1 and payoff 1
  - an underlying asset with price 1 and uncertain payoff
  - four vanilla options on the underlying with known (market) prices

Type	Strike	Price
Call	1.1	0.06
Call	1.2	0.03
Put	0.8	0.02
Put	0.7	0.01

option types and prices



option payoff diagram

- $m = 200$ possible outcomes for the underlying asset, uniformly between 0.5 and 2
- we seek price bounds on a collar option with floor 0.9 and cap 1.15
- solving optimization problem gives the arbitrage-free collar price range $[-0.015, 0.033]$

Outline

Currency exchange problem

- we hold $c^{\text{init}} = (c_1^{\text{init}}, \ldots, c_n^{\text{init}})$ of n currencies, in USD under nominal exchange rates
- want to exchange them to obtain (at least) $c^{\text{req}} = (c_1^{\text{req}}, \ldots, c_n^{\text{req}})$, valued in USD
- $X \in \mathbf{R}^{n \times n}$ is currency exchange matrix; $X_{ij} \geq 0$ the amount of j we exchange for i, in USD
- $\Delta_{ij} \geq 0$ is cost of exchanging one USD of currency j for currency i, expressed as a fraction
- exchange X_{ij} costs us $X_{ij}\Delta_{ij}$ USD

- optimal currency exchange: find X that minimizes cost

$$
\begin{array}{ll}
\text{minimize} & \sum_{i,j=1}^{n} X_{ij}\Delta_{ij} \\
\text{subject to} & X_{ij} \geq 0, \quad \textbf{diag}(X) = 0, \\
& c_i^{\text{init}} + \sum_j X_{ij} - \sum_j X_{ji} \geq c_i^{\text{req}}, \quad i = 1, \ldots, n
\end{array}
$$

Currency exchange in CVXPY

```
X = cp.Variable((n, n), nonneg=True)

objective = cp.sum(cp.multiply(X, Delta))
constraints = [
    cp.diag(X) == 0,
    c_init + cp.sum(X, axis=1) - cp.sum(X, axis=0) >= c_req
]

prob = cp.Problem(cp.Minimize(objective), constraints)
prob.solve()
```

Example

- USD, EUR, CAD, SEK, with initial and required holdings (in $\$10^6$)

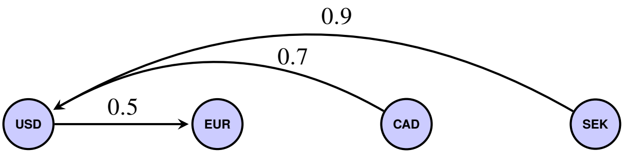
$$c^{\text{init}} = (1, 1, 1, 1), \qquad c^{\text{req}} = (2.1, 1.5, 0.3, 0.1)$$

- exchange rates in basis points (bps) (10^{-4})

$\Delta =$

	USD	EUR	CAD	SEK
USD	0.0	0.1	4.4	4.8
EUR	0.1	0.0	5.0	5.7
CAD	2.8	6.9	0.0	8.5
SEK	1.1	7.9	7.6	0.0

- cheap to trade USD and EUR, expensive to trade CAD and SEK
- optimal exchanges (in $\$10^6$)



Outline

Purchase execution and risk

- want to purchase Q shares over T periods $t = 1, \ldots, T$
- $q \in \mathbf{R}^T$ is purchase schedule; $q \geq 0$, $\mathbf{1}^T q = Q$
- (bid-ask midpoint) price dynamics:

$$p_t = p_{t-1} + \xi_t, \quad t = 2, \ldots, T,$$

  with p_1 known, ξ_t IID $\mathcal{N}(0, \sigma^2)$
- nominal cost is random variable $p^T q$, with

$$\mathbf{E}(p^T q) = p_1 Q, \qquad \mathbf{var}(p^T q) = q^T \Sigma q$$

  where $\Sigma_{kl} = \sigma^2 \min(k-1, l-1)$
- $q^T \Sigma q$ is the **risk**

Market impact

- transaction (market impact) cost, in USD ('squareroot model'):

$$\sum_{t=1}^{T} \sigma \pi_t^{1/2} q_t = \sigma \sum_{t=1}^{T} q_t^{3/2} / v_t^{1/2}$$

- v_t is market volume, $\pi_t = q_t / v_t$ is participation rate in period t
- actual cost of execution is nominal mean cost $p_1 Q$ plus transaction cost

Optimal execution

- trade off risk and transaction cost, with participation rate limit

$$
\begin{array}{ll}
\text{minimize} & \sigma \sum_{t=1}^{T} \left(q_t^{3/2} / v_t^{1/2} \right) + \gamma q^T \Sigma q \\
\text{subject to} & q \geq 0, \quad \mathbf{1}^T q = Q, \quad q_t / v_t \leq \pi^{\max}, \quad t = 1, \ldots, T,
\end{array}
$$

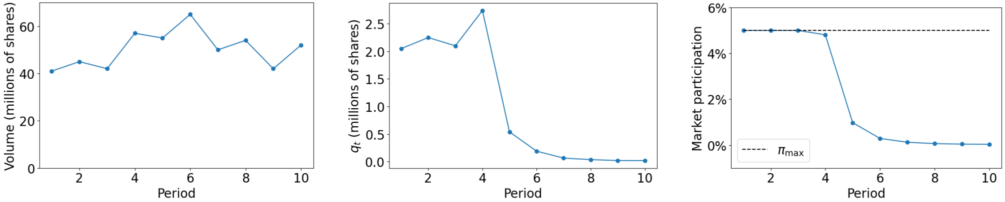
- $\gamma > 0$ is a risk aversion parameter
- $\pi^{\max}$ participation rate limit
- a convex problem [Almgren and Chriss, 2001]
- an alternate formulation reduces computational complexity from $O(T^3)$ to $O(T)$
- without risk term and participation contraint, constant participation is optimal

Optimal execution in CVXPY

```
q = cp.Variable(T, nonneg=True)
pi = q / v

risk = cp.quad_form(q, Sigma)
transaction_cost = sigma * cp.power(q, 3 / 2) @ cp.power(v, -1 / 2)

objective = cp.Minimize(transaction_cost + gamma * risk)
constraints = [cp.sum(q) == Q, pi <= pi_max]

prob = cp.Problem(objective, constraints)
prob.solve()
```

Example

- purchase 10 million Apple shares over 10 trading days (Feb 8–22, 2024)
- participation rate limit $\pi^{\max} = 5\%$



volume　　optimal purchase schedule　　participation rate

Outline

Merton consumption-investment dynamics

- plan consumption and investment at times $t = 0, \ldots, T$
- in period t, wealth is k_t, consumption is c_t, labor income is y_t (all in inflation adjusted USD)
- remaining wealth invested in n assets with return mean μ and covariance Σ
- $r_t \in \mathbf{R}^n$ is asset return in period t
- $h_t \in \mathbf{R}^n$ denotes amounts invested in period t in USD
- $r_t^T h_t$ is the portfolio return in USD
- wealth dynamics given by
$$k_{t+1} = k_t - c_t + y_t + r_t^T h_t$$

Merton consumption-investment problem

- maximize expected utility of consumption and bequest

$$\mathbf{E}\left(\frac{\beta}{\rho}k_T^\rho + \frac{1}{\rho}\sum_{t=0}^{T-t} c_t^\rho\right)$$

  $\beta > 0$ sets relative importance of bequest; $\rho < 1$ sets risk aversion

- a stochastic control problem, solved in [Merton, 1975]

Deterministic wealth dynamics

- replace stochastic wealth dynamics $k_{t+1} = k_t - c_t + y_t + r_t^T h_t$ with deterministic dynamics

$$k_{t+1} = k_t - c_t + y_t + \mu^T h_t - \frac{(1-\rho)}{2} \frac{h_t^T \Sigma h_t}{k_t + v_t}$$

- v_t is the present value of future labor income discounted at the risk-free rate μ^{rf}

$$v_t = \sum_{\tau=t}^{T-1} y_\tau \exp(-\mu^{\mathrm{rf}}(\tau - t))$$

- we require $k_t + v_t > 0$, *i.e.*, wealth plus future labor income is positive
- last term is a pessimistic adjustment for risk derived in [Moehle and Boyd, 2021]

Certainty equivalent convex optimization formulation

- yields deterministic convex optimization problem

$$
\begin{array}{ll}
\text{maximize} & \frac{\beta}{\rho} k_T^\rho + \frac{1}{\rho} \sum_{t=0}^{T-1} c_t^\rho \\
\text{subject to} & k_{t+1} \leq k_t - c_t + y_t + \mu^T h_t - \frac{(1-\rho)}{2} \frac{h_t^T \Sigma h_t}{k_t + v_t} \\
& k_t = \mathbf{1}^T h_t, \quad c_t \geq 0
\end{array}
$$

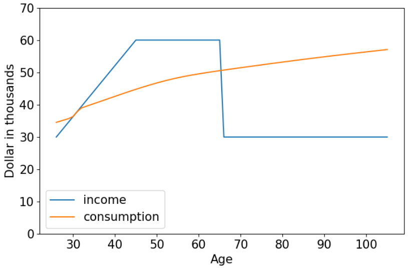
  (dynamic equality is replaced by inequality constraint, which is tight at solution)
- this certainty equivalent problem also solves stochastic problem
- can be extended to include mortality, liabilities, taxes, portfolio constraints, ...

Certainty equivalent Merton problem in CVXPY

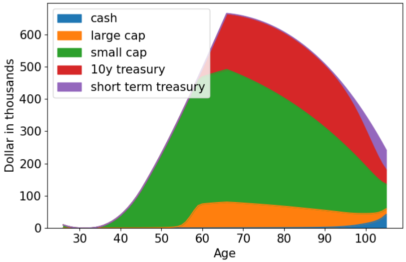
```
k = cp.Variable(T + 1)
h = cp.Variable((n, T))
c = cp.Variable(T, nonneg=True)

Sigma_half = np.linalg.cholesky(Sigma)

objective = beta / rho * k[T] ** rho + 1 / rho * cp.sum(c**rho)
constraints = [k[0] == k0, k[:-1] == cp.sum(h, axis=0)]
constraints += [
    k[t + 1] <= k[t] - c[t] + y[t] + mu.T @ h[:, t]
    - (1 - rho) / 2 * cp.quad_over_lin(Sigma_half.T @ h[:, t], k[t] + v[t])
    for t in range(T)
]

prob = cp.Problem(cp.Maximize(objective), constraints)
prob.solve()
```

Example

- plan over 80 years (age 25–105), with initial wealth $k_0 = 10,000$ USD
- $n = 5$ assets, utility parameter $\rho = -4$, bequest parameter $\beta = 10$
- five asset classes, with long only portfolio constraint $h_t \geq 0$
- salary grows until age 50, then is constant, then drops to 50% at age 65



optimal consumption



optimal investments

Outline

Alternative investments

- investor makes **commitments** to an alternative investment in each quarter $t = 1, \ldots, T$
- over time she puts committed money into the investment in response to **capital calls**
- she receives money through **distributions**
- examples: private equity, venture capital, infrastructure projects, ...

Alternative investment dynamics

- $c_t, p_t, d_t \geq 0$ are commitments, capital calls, and distributions
- $n_t \geq 0$ is net asset value (NAV), r_t is investment return
- $u_t \geq 0$ is total uncalled previous commitments
- dynamics:

$$n_{t+1} = n_t(1 + r_t) + p_t - d_t, \qquad u_{t+1} = u_t - p_t + c_t$$

  with $n_0 = 0$, $u_0 = 0$
- simple model of calls and distributions:

$$p_t = \gamma^{\text{call}} u_t, \qquad d_t = \gamma^{\text{dist}} n_t$$

- $\gamma^{\text{call}}, \gamma^{\text{dist}} \in (0, 1)$ are call and distribution intensities or rates

Alternative investment planning

- choose commitments $c_1, \ldots, c_T$ to minimize

$$\frac{1}{T+1}\sum_{t=1}^{T+1}(n_t - n^{\text{des}})^2 + \lambda \frac{1}{T-1}\sum_{t=1}^{T-1}(c_{t+1} - c_t)^2,$$

  where n^{des} is the desired NAV and $\lambda > 0$ is a smoothing parameter
- penalizes deviation from desired NAV, and encourages smooth commitment schedule
- can add constraints such as

$$c_t \le c^{\text{max}}, \qquad u_t \le u^{\text{max}}$$

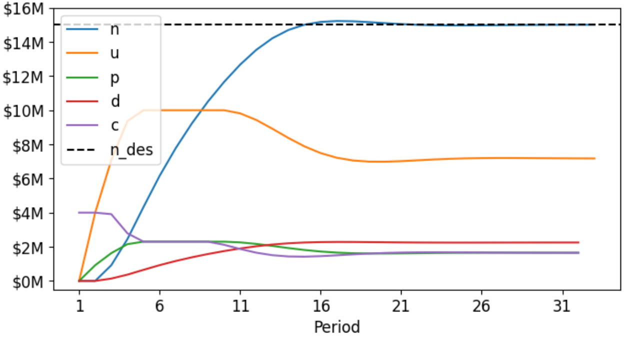
- yields convex problem
- can be extended to uncertain parameters, multiple illiquid investments, and mixed with liquid investments [Luxenberg et al., 2022]

Alternative investment planning in CVXPY

```
n = cp.Variable(T+1, nonneg=True); u = cp.Variable(T+1, nonneg=True)
p = cp.Variable(T, nonneg=True); d = cp.Variable(T, nonneg=True)
c = cp.Variable(T, nonneg=True)

tracking =  cp.mean((n-n_des)**2)
smoothing = lmbda * cp.mean(cp.diff(c)**2)

constraints = [c <= c_max, u <= u_max, n[0] == 0, u[0] == 0]
for t in range(T):
    constraints += [n[t+1] == (1+r)*n[t]+p[t]-d[t]]
    constraints += [u[t+1] == u[t]-p[t]+c[t]]
    constraints += [p[t] == gamma_call*u[t], d[t] == gamma_dist*n[t]]

prob = cp.Problem(cp.Minimize(tracking+smoothing), constraints)
prob.solve()
```

Example

- $T = 32$ (eight years), $r_t = 0.04$ (4% quarterly return), $\gamma^{\text{call}} = .23$, $\gamma^{\text{dist}} = .15$
- planning parameters: $c^{\max} = 4$, $u^{\max} = 10$, $n^{\text{des}} = 15$, and $\lambda = 5$



Outline

Blending forecasts

- we observe $x_1, \ldots, x_t \in \mathbf{R}^d$ and seek a forecast $\hat{x}_{t+1}$ of x_{t+1}
- we have K forecasts $\hat{x}_{t+1}^1, \ldots, \hat{x}_{t+1}^K$
- called K experts [Hastie et al., 2009]
- we blend them using weights $\pi_t^1, \ldots, \pi_t^K$ with $\pi_t \geq 0$, $\mathbf{1}^T \pi_t = 1$

$$\hat{x}_{t+1} = \sum_{k=1}^{K} \pi_t^k \hat{x}_{t+1}^k$$

- the weights can vary over time
- we may want them smoothly varying, *i.e.*, $\pi_{t+1} \approx \pi_t$
- we may want them close to some prior or baseline weights π^{pri}
- examples: return mean, return covariance, ...

Blending forecasts using convex optimization

- find weights π_t as solution of convex optimization problem

$$
\begin{array}{ll}
\text{minimize} & \frac{1}{M}\sum_{\tau=t-M+1}^{t} \ell(\hat{x}_\tau, x_\tau) + r^{\text{sm}}(\pi, \pi_{t-1}) + r^{\text{pri}}(\pi, \pi^{\text{pri}}) \\
\text{subject to} & \hat{x}_\tau = \sum_{k=1}^{K} \pi_k \hat{x}_\tau^k, \quad \pi \geq 0, \quad \mathbf{1}^T \pi = 1
\end{array}
$$

  with variable $\pi \in \mathbf{R}^K$
- ℓ is prediction loss, r^{sm} penalizes weight change, r^{pri} penalizes deviation from prior
- we assume $\ell, r^{\text{sm}}, r^{\text{pri}}$ are convex in π
- idea: use blending weights that would have worked well over the last M periods

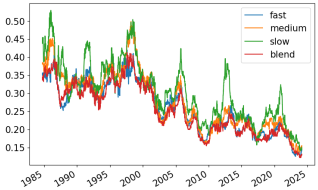
Blending forecasts in CVXPY

- `X_hat` is an $M \times K$ matrix of expert forecasts over the last M periods $\tau = t - M + 1, \ldots, t$
- `x` is an M-vector of observed quantities over the same period

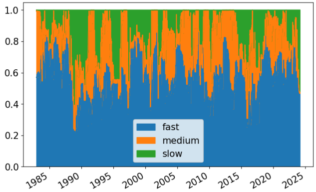
```
pi = cp.Variable(K, nonneg=True)
x_hat = X_hat @ pi

objective = cp.Minimize(cp.mean(loss(x_hat, x)))
constraints = [cp.sum(pi) == 1]

prob = cp.Problem(objective, constraints)
prob.solve()
```

Example

- predict log of daily trading volume of Apple, 1982–2024
- $K = 3$ predictors: 5-day (fast), 21-day (medium), and 63-day (slow) moving medians
- absolute loss $\ell(\hat{x}_\tau, x_\tau) = |\hat{x}_\tau - x_\tau|$; $M = 250$ trading days



250-day rolling median absolute error



weights (π)

error	fast	median	slow	blend
median	0.25	0.27	0.29	0.24
90th percentile	0.72	0.74	0.82	0.68
10th percentile	0.05	0.05	0.05	0.04

Outline

Bond price model

- bond pays holder c_t in periods $t = 1, \ldots, T$ (coupons and principal)
- price of bond is discounted present value of payments

$$p = \sum_{t=1}^{T} c_t \exp(-t(y_t + s)),$$

where $y = (y_1, \ldots, y_T) \in \mathbf{R}^T$ is the yield curve, and $s \geq 0$ the spread
- we assume $y = Ya$ where Y are basis functions and a are coefficients
- can use principal component analysis to fit Y to historical data
- spread depends on bond rating (readily extended to depend on other attributes)

Matrix pricing problem

- we are given market prices p_i and ratings $r_i \in \{1, \ldots, K\}$ of n bonds, $i = 1, \ldots, n$
- we want to fit a yield curve $y \in \mathbf{R}^T$ and spreads $s \in \mathbf{R}^K$ to this data
- we add constraint $0 \leq s_1 \leq \cdots \leq s_K$ (higher ratings have lower spreads)
- using square error we fit y and s by solving problem

$$
\begin{array}{ll}
\text{minimize} & \sum_{i=1}^n \left(p_i - \sum_{t=1}^T c_{i,t} \exp(-t(y_t + s_{r_i}))\right)^2 \\
\text{subject to} & 0 \leq s_1 \leq \cdots \leq s_K
\end{array}
$$

  with variables y and s
- not convex, but can be solved (approximately) as a sequence of convex problems
- linearize exponential term and iteratively fit yields and spreads

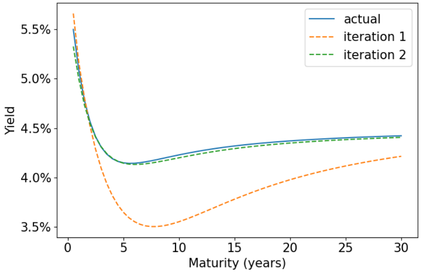
Matrix pricing in CVXPY

- use CVXPY to automatically linearize the exponential term as `p_current + Delta_hat`
- would add trust penalty to the iterates in practice
- code below computes first iteration

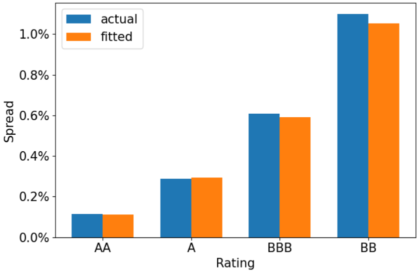
```
y = cp.Variable(T, value=y_init)
s = cp.Variable(K, value=S_init, nonneg=True)
a = cp.Variable(a.size, values=a_init)

discount = cp.exp(cp.multiply(-t, y.reshape((1, -1)) + s[ratings].reshape((-1, 1))))
p_current = cp.sum(cp.multiply(C, discount), axis=1)

Delta_hat = p_current.grad[y].T @ (y-y.value) + p_current.grad[s].T @ (s - s.value)
objective = cp.norm2(p - (p_current.value + Delta_hat))
constraints = [cp.diff(s) >= 0, y == Y @ a]

problem = cp.Problem(cp.Minimize(objective), constraints)
problem.solve()
```

Example

- consider $n = 1000$ bonds, with a maturity of up to 30 years
- bonds are rated AAA, AA, A, BBB, BB
- used data from 1990 to 2024 to fit basis functions, latest yields, and spread to price bonds
- fit yields and spreads to bond prices gives \$0.03 RMSE (2.9 bps)



iteratively fitted yields for rating AAA



rating spreads (vs. AAA)

Outline

Stochastic control

- dynamics $x_{t+1} = f(x_t, u_t, w_t)$, $t = 0, 1, \ldots, T-1$
- $x_t \in \mathcal{X}$ is the state, $u_t \in \mathcal{U}$ is the input or action, $w_t \in \mathcal{W}$ is the disturbance
- $x_0, w_0, \ldots, w_{T-1}$ are independent random variables
- state feedback **policy** $u_t = \pi_t(x_t)$, $t = 0, 1, \ldots, T-1$
- stochastic control problem: choose policy to minimize

$$J = \mathbf{E}\left(\sum_{t=0}^{T-1} g_t(x_t, u_t) + g_T(x_T)\right)$$

- stage cost $g_t(x_t, u_t)$; terminal cost $g_T(x_T)$
- examples: investing, execution, consumption, ...

Solution via dynamic programming

- exact solution from Bellman [1954]
- only practical in special cases
  - $\mathcal{X}, \mathcal{U}$ finite
  - linear dynamics and quadratic cost
  - $x_t \in \mathbf{R}^n$ with n very small, like 2 or 3
  - a few other special cases (*e.g.*, Merton problem)
- but several heuristics and approximations work very well

Model predictive control

to evaluate $\pi_t^{\text{mpc}}(x_t)$:

- **forecast**: predict stochastic future values $w_t, \ldots, w_{T-1}$ as $\hat{w}_{t|t}, \ldots, \hat{w}_{T-1|t}$
- **plan**: solve certainty equivalent problem assuming forecasts are correct

$$\begin{array}{ll} \text{minimize} & \sum_{\tau=t}^{T-1} g_t(\hat{x}_{\tau|t}, u_t) + g_T(\hat{x}_{T|t}) \\ \text{subject to} & \hat{x}_{\tau+1|t} = f(x_{\tau|t}, u_{\tau|t}, \hat{w}_{\tau|t}), \quad \tau = t, \ldots, T-1, \quad \hat{x}_{t|t} = x_t \end{array}$$

  with variables $u_{\tau|t}, \ldots, u_{T-1|t}, \hat{x}_{t|t}, \ldots, \hat{x}_{T|t}$
- **execute**: $\pi_t^{\text{mpc}}(x_t) = u_{t|t}$ (*i.e.*, take first action in plan)

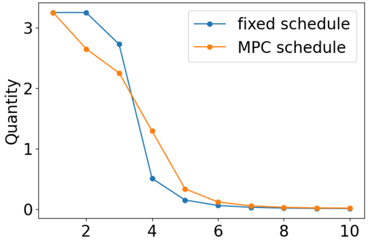
Model predictive control

- when f is linear in x, u and g_t are convex, planning problem is convex, hence tractable
- MPC is optimal in a few special cases, but often performs extremely well
- used in many industries, *e.g.*, guiding SpaceX's Falcon first stages to their landings [Blackmore, 2016]

- **receding horizon MPC**
  - a variation for when there is no terminal time T
  - solve planning problem over H-period horizon $\tau = t$ to $\tau = t + H$
  - can include terminal cost or constraint

Example: Order execution via MPC

- purchase 10 million Apple shares over 10 trading days (Feb 8–22, 2024)
- participation rate limit $\pi^{\max} = 5\%$
- forecasts:
  - $\hat{v}_{\tau|t}$ is 5-day trailing median of volumes, $\tau = t, \ldots, T$
  - $\sigma_{\tau|t}$ is 21-day trailing standard deviation, $\tau = t, \ldots, T$
- transaction cost is 1.881B USD for fixed schedule and 1.877B USD for MPC
- MPC saves us 4M USD, about 20 bps



Example: Order execution via MPC