CVXPY x NASA Course 2024

Philipp Schiele Steven Diamond Parth Nobel Akshay Agrawal

July 8, 2024

Code Generation for Quadcopter Control

Outline

Homework review

Parameters

Converting to standard form

Code generation

Solution

https://marimo.app/l/b3bjrd

This lecture

- explain the role of parameters in CVXPY
- show examples of transforming problems into standard form
- introduce CVXPYgen, a CVXPY extension for code generation
- cover more convex optimization examples

Resources

- Embedded Code Generation with CVXPY [Schaller et al.]
- `https://github.com/cvxgrp/cvxpygen`
- Differentiable Convex Optimization Layers [Agrawal et al.]
- `https://github.com/cvxgrp/cvxpylayers`

Outline

Homework review

Parameters

Converting to standard form

Code generation

Parameters in CVXPY

- symbolic representations of constants
- can specify sign (for use in DCP analysis)
- change value of constant without re-parsing problem

```python
# scalar parameter
alpha = cp.Parameter()

# scalar, non-negative
gamma = cp.Parameter(nonneg=True)

# matrix with shape m x n
A = cp.Parameter((m, n))
```

Parameters in CVXPY

- useful for computing trade-off curves
  - fuel use versus trajectory smoothness
  - portfolio risk versus return
  - model sparsity versus data fit
- code below updates the parameter gamma in a for loop

```
x_values = []
for val in numpy.logspace(-4, 2, 100):
    gamma.value = val
    prob.solve()
    x_values.append(x.value)
```

Parallel style trade-off curve

```python
# Use tools for parallelism in standard library.
from multiprocessing import Pool

# Function maps gamma value to optimal x.
def get_x(gamma_value):
    gamma.value = gamma_value
    result = prob.solve()
    return x.value

# Parallel computation with N processes.
pool = Pool(processes = N)
x_values = pool.map(get_x, numpy.logspace(-4, 2, 100))
```

Performance

(LASSO)

$$\text{minimize} \quad \|Ax - b\|_2^2 + \gamma\|x\|_1$$

with variable $x \in \mathbf{R}^n$

- $A \in \mathbf{R}^{1000 \times 500}$, 100 values γ
- single thread time for one LASSO: 1.6 seconds (OSQP)

	for-loop	4 proc.	32 proc.	warm-start
4 core MacBook Pro	180 sec	70 sec	81 sec	31 sec
32 cores, Intel Xeon	285 sec	89 sec	31 sec	48 sec

Parametrized convex optimization

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

- $x \in \mathbf{R}^n$ is the optimization variable
- f_0 is the convex objective function, to be minimized
- $f_1,\dots,f_m$ are convex inequality constraint functions
- $g_1 \dots, g_p$ are affine equality constraint functions
- $\theta \in \mathbf{R}^d$ is the parameter

Exercise

`https://marimo.app/l/629s3k`

Outline

Example: Quadratic program

- original (nonnegative least squares) problem

$$\begin{array}{ll} \text{minimize} & \|Gx - h\|_2^2 \\ \text{subject to} & x \geq 0, \end{array}$$

  with variable $x \in \mathbf{R}^n$, parameters $\theta = (G, h)$

- canonicalize to form accepted by QP solver OSQP (Stellato 2020),

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\tilde{x}^T P \tilde{x} + q^T \tilde{x} \\ \text{subject to} & l \leq A\tilde{x} \leq u \end{array}$$

  with variable $\tilde{x} \in \mathbf{R}^{\tilde{n}}$, canonical parameters $\tilde{\theta} = (P, q, A, l, u)$

Example: Canonicalization and retrieval

- canonicalize original problem using $\tilde{x} = x$ and

$$P = 2G^T G, \quad q = -2G^T h, \quad A = I, \quad l = 0, \quad u = \infty$$

- retrieve solution of original problem as $x^\star = \tilde{x}^\star$

- this example was simple and could easily be done by hand
- more complex examples much less so

Example: CVXPY code

```
import cvxpy as cp

x = cp.Variable(n, name='x')

# declare parameters
G = cp.Parameter((m, n), name='G')
h = cp.Parameter(m, name='h')

problem = cp.Problem(cp.Minimize(cp.sum_squares(G@x-h)), [x>=0])

# specify parameter values
G.value = numpy.random.randn(m, n)
h.value = numpy.random.randn(m)

problem.solve(solver='OSQP')
```

Expanding nonlinearities

- adding an L1-norm

$$\begin{array}{ll} \text{minimize} & \|Gx - h\|_2^2 + \gamma \|x\|_1 \\ \text{subject to} & x \geq 0, \end{array}$$

  with variable $x \in \mathbf{R}^n$, parameters $\theta = (G, h, \gamma)$
- how do we convert to a QP?
- the trick is introducing a new variable

Adding a new variable

- original problem

$$\begin{array}{ll} \text{minimize} & \|Gx - h\|_2^2 + \gamma \|x\|_1 \\ \text{subject to} & x \geq 0, \end{array}$$

- equivalent problem with added variable $y \in \mathbf{R}^n$

$$\begin{array}{ll} \text{minimize} & \|Gx - h\|_2^2 + \gamma \sum_{i=1}^n y_i \\ \text{subject to} & x \geq 0 \\ & -y_i \leq x_i \leq y_i, \quad i = 1, \ldots, n, \end{array}$$

- implied conditions
  - $y_i \geq 0$
  - if $x_i \geq 0$, $y_i = x_i$
  - if $x_i < 0$, $y_i = -x_i$

Exercise

`https://marimo.app/l/1eg9ov`

Solution

`https://marimo.app/l/gat8zt`

Outline

Homework review

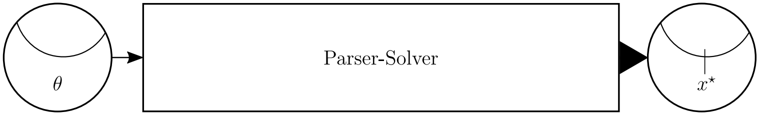
Parameters

Converting to standard form

Code generation

Parser-solvers

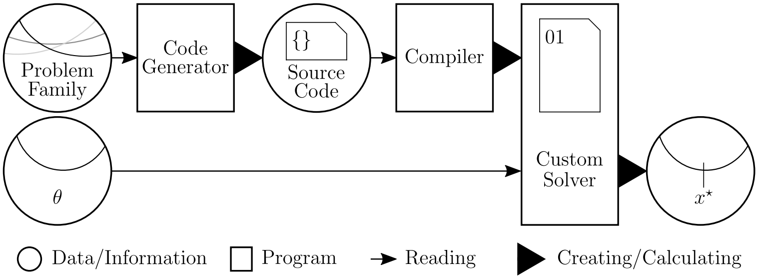
- parser-solvers canonicalize each time the problem is solved



- parser-solvers compile a *problem instance* into a *canonicalized problem instance*, then solve it
- most DSLs are parser-solvers

Code generators

- code generators compile a *problem family* into source code for a *custom solver*



- useful for
  - embedded applications, possibly with hard real-time deadlines
  - speeding up the solution of many different problem instances

CVXGEN code generator

- developed by Mattingley and Boyd in 2010
- handles problems transformable to QPs
- generates custom interior-point solver in flat, explicit C
- handles problem families with up to a few thousand parameters
- generated code suitable for real-time control systems
- used for autonomous driving, dynamic energy management, real-time trading, precision landing (*e.g.*, all SpaceX Falcon 9 and Falcon Heavy landings)

CVXGEN in action

https://blogs.nasa.gov/spacex/2019/06/25/side-boosters-have-landed/

CVXPYgen

- a new open-source code generator built on CVXPY
- developed by Schaller, Banjac, Diamond, Agrawal, Stellato, and Boyd in 2022
- generates custom canonicalizer and retrieval in flat C
- can be used with multiple solvers: OSQP, SCS (O'Donoghue 2016), ECOS (Domahidi 2013)
- first generic code generator that supports SOCPs
- supports warm-starting, which can give significant speedup
- handles high-dimensional parameters with user-defined sparsity patterns
- compiled CVXPYgen solver can be used as a custom solver for CVXPY (!)

Disciplined parametrized programming (DPP)

- restricts how parameters enter problem description, in addition to DCP rules
- for DPP-compliant problems, canonicalization and retrieval can be affine mappings (Agrawal 2019)

$$\tilde{\theta} = C \begin{bmatrix} \theta \\ 1 \end{bmatrix}, \qquad x^\star = R \begin{bmatrix} \tilde{x}^\star \\ 1 \end{bmatrix}$$

- C and R are (very) sparse matrices
- CVXPYgen generates flat C code to implement canonicalization and retrieval
- sparse-matrix-vector multiplies, using pointers or avoiding updates when possible

DPP examples

```python
x = cp.Variable()
gamma = cp.Parameter()

gamma * x # DPP
gamma**2 * x # not DPP
x / gamma # not DPP
x + gamma # DPP
gamma * x + 2 * gamma # DPP

# choose your parameters so they enter the problem
# in an affine way:
gamma2 = cp.Parameter()
gamma2.value = gamma.value ** 2
gamma2 * x # DPP
```

Example (again)

- canonicalize original nonnegative least squares problem

$$\begin{array}{ll} \text{minimize} & \|Gx - h\|_2^2 \\ \text{subject to} & x \geq 0 \end{array}$$

  to OSQP standard form

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\tilde{x}^T P \tilde{x} + q^T \tilde{x} \\ \text{subject to} & l \leq A\tilde{x} \leq u \end{array}$$

- canonicalization shown before is not an affine mapping from θ to $\tilde{\theta}$

Example: Affine canonicalization and retrieval

- first transform to problem

$$\begin{array}{ll} \text{minimize} & \|\tilde{x}_2\|_2^2 \\ \text{subject to} & \tilde{x}_2 = G\tilde{x}_1 - h, \quad \tilde{x}_1 \geq 0, \end{array}$$

  with variable $\tilde{x} = (\tilde{x}_1, \tilde{x}_2)$, $\tilde{x}_1 = x$

- canonicalization is affine:

$$P = \begin{bmatrix} 0 & 0 \\ 0 & 2I \end{bmatrix}, \quad q = 0, \quad A = \begin{bmatrix} G & -I \\ I & 0 \end{bmatrix}, \quad l = \begin{bmatrix} h \\ 0 \end{bmatrix}, \quad u = \begin{bmatrix} h \\ \infty \end{bmatrix}$$

- retrieval is affine: $x^\star = [I\ 0]\tilde{x}^\star$

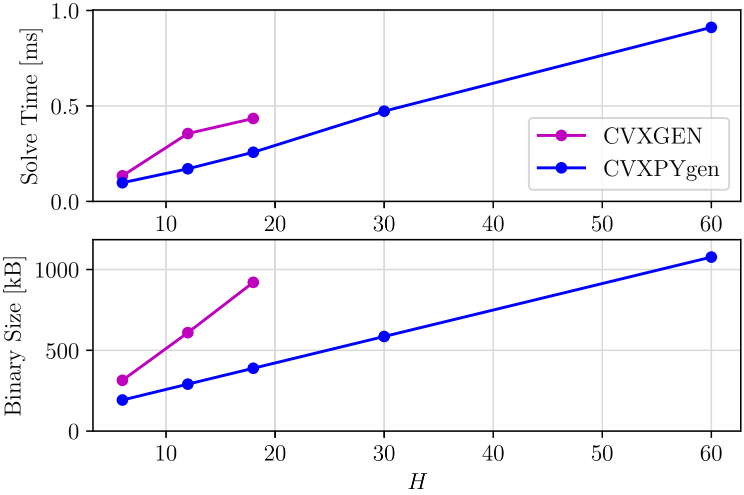
Example: CVXPY/CVXPYgen code

```python
import cvxpy as cp
from cvxpygen import cpg

# model problem
x = cp.Variable(n, name='x')
G = cp.Parameter((m, n), name='G')
h = cp.Parameter(m, name='h')
problem = cp.Problem(cp.Minimize(cp.sum_squares(G@x-h)), [x>=0])

# generate code
cpg.generate_code(problem)
```

Example: Model predictive control (MPC)

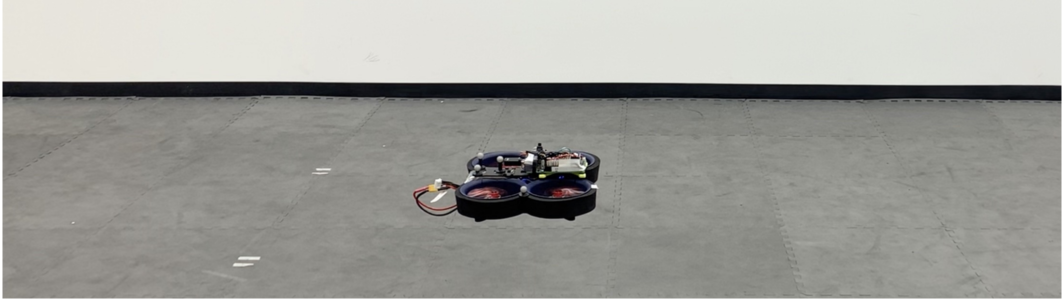
- family of MPC problems for control of a drone
- parametrized by horizon length $H \in \{6, 12, 18, 30, 60\}$
- number of variables around $10H$
- binary sizes and solve times on MacBook Pro 2.3GHz Intel i5, using gcc -O3

Comparison with CVXGEN



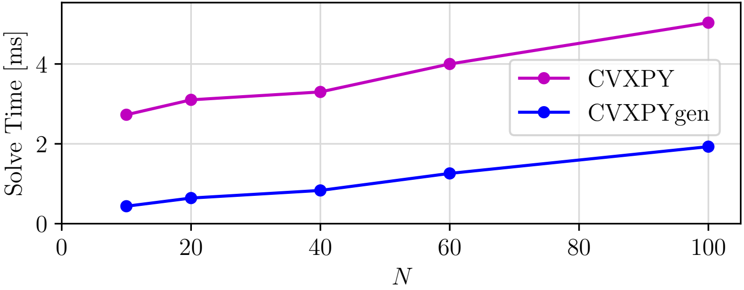
Deployment in embedded system

- use generated solver to control 14×14 cm quadcopter
- generated code compiled in a robot operating system (ROS) node
- run on drone's Intel Atom x5-Z8350 processor at 30Hz



Example: Portfolio trading

- family of portfolio optimization problems
- parametrized by number of assets $N \in \{10, 20, 40, 60, 100\}$
- number of variables around $2N$
- solve times with CVXPY and CVXPY interface to CVXPYgen



Break-even point

- *break-even point*: number of instances that need to be solved before CVXPYgen is faster than CVXPY, when we include the code generation and compilation time
- around 5000, and not too dependent on N
- typical portfolio optimization back-test involves daily trading over multiple years, with hundreds of different hyper-parameter values
- gives order 100k or more solves, well above the break-even point

Summary

CVXPYgen

- gives seemless path from prototyping in Python/CVXPY to implementation in C
- handles wider variety of problems than CVXGEN (*e.g.*, SOCPs)
- outperforms CVXGEN in terms of
  - allowable problem size
  - compiled code size
  - solve times
- gives significant speedup on general-purpose machines with many solves (compared with CVXPY)

Examples

Outline

Portfolio Optimization

Worst-Case Risk Analysis

Optimal Advertising

Portfolio allocation vector

- invest fraction w_i in asset i, $i = 1, \ldots, n$
- $w \in \mathbf{R}^n$ is *portfolio allocation vector*
- $\mathbf{1}^T w = 1$
- $w_i < 0$ means a *short position* in asset i
  (borrow shares and sell now; must replace later)
- $w \geq 0$ is a *long only* portfolio
- $\|w\|_1 = \mathbf{1}^T w_+ + \mathbf{1}^T w_-$ is *leverage*
  (many other definitions used ...)

Asset returns

- investments held for one period
- initial prices $p_i > 0$; end of period prices $p_i^+ > 0$
- asset (fractional) returns $r_i = (p_i^+ - p_i)/p_i$
- portfolio (fractional) return $R = r^T w$
- common model: r is a random variable, with mean $\mathbf{E}\, r = \mu$, covariance $\mathbf{E}(r-\mu)(r-\mu)^T = \Sigma$
- so R is a RV with $\mathbf{E}\, R = \mu^T w$, $\mathbf{var}(R) = w^T \Sigma w$
- $\mathbf{E}\, R$ is (mean) *return* of portfolio
- $\mathbf{var}(R)$ is *risk* of portfolio
  (risk also sometimes given as $\mathbf{std}(R) = \sqrt{\mathbf{var}(R)}$)

- two objectives: high return, low risk

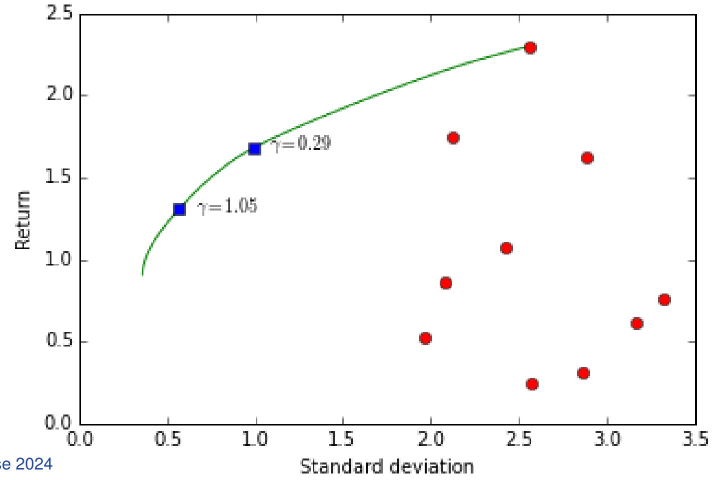
Classical (Markowitz) portfolio optimization

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

- variable $w \in \mathbf{R}^n$
- $\mathcal{W}$ is set of allowed portfolios
- common case: $\mathcal{W} = \mathbf{R}^n_+$ (long only portfolio)
- $\gamma > 0$ is the *risk aversion parameter*
- $\mu^T w - \gamma w^T \Sigma w$ is *risk-adjusted return*
- varying γ gives optimal *risk-return trade-off*
- can also fix return and minimize risk, etc

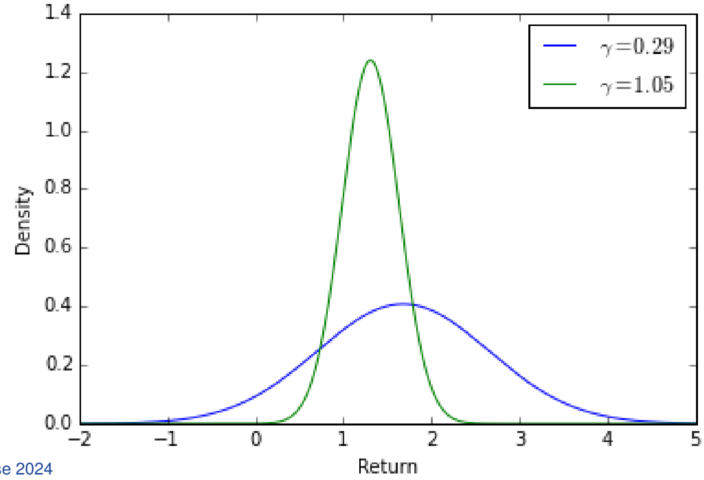
Example

optimal risk-return trade-off for 10 assets, long only portfolio



Example

return distributions for two risk aversion values



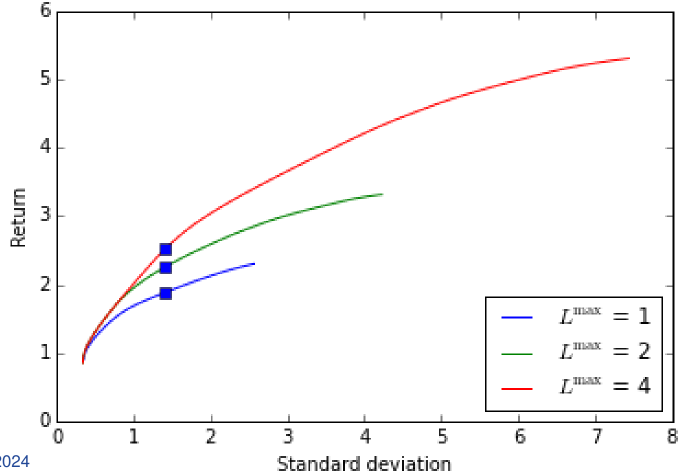
Portfolio constraints

- $\mathcal{W} = \mathbf{R}^n$ (simple analytical solution)
- leverage limit: $\|w\|_1 \leq L^{\max}$
- *market neutral*: $m^T \Sigma w = 0$
  - m_i is capitalization of asset i
  - $M = m^T r$ is *market return*
  - $m^T \Sigma w = \mathbf{cov}(M, R)$

  i.e., market neutral portfolio return is uncorrelated with market return

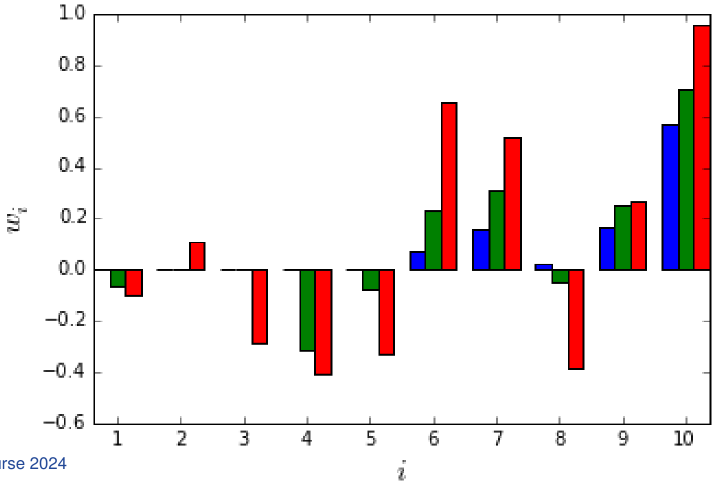
Example

optimal risk-return trade-off curves for leverage limits $1, 2, 4$



Example

three portfolios with $w^T \Sigma w = 2$, leverage limits $L = 1, 2, 4$



Variations

- require $\mu^T w \geq R^{\min}$, minimize $w^T \Sigma w$ or $\|\Sigma^{1/2} w\|_2$
- include (broker) cost of short positions,

$$s^T (w)_-, \quad s \geq 0$$

- include transaction cost (from previous portfolio w^{prev}),

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

common models: $\eta = 1,\ 3/2,\ 2$

Factor covariance model

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

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

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

Portfolio optimization with factor covariance model

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

- variables $w \in \mathbf{R}^n$ (allocations), $f \in \mathbf{R}^k$ (factor exposures)
- $\mathcal{F}$ gives factor exposure constraints

- computational advantage: $O(nk^2)$ vs. $O(n^3)$

Example

- 50 factors, 3000 assets
- leverage limit $= 2$
- solve with covariance given as
  - single matrix
  - factor model
- CVXPY/OSQP single thread time

covariance	solve time
single matrix	173.30 sec
factor model	0.85 sec

Outline

Portfolio Optimization

Worst-Case Risk Analysis

Optimal Advertising

Covariance uncertainty

- single period Markowitz portfolio allocation problem
- we have fixed portfolio allocation $w \in \mathbf{R}^n$
- return covariance Σ not known, but we believe $\Sigma \in \mathcal{S}$
- $\mathcal{S}$ is convex set of possible covariance matrices
- risk is $w^T \Sigma w$, a *linear function of* Σ

Worst-case risk analysis

- what is the worst (maximum) risk, over all possible covariance matrices?
- worst-case risk analysis problem:

$$\begin{array}{ll} \text{maximize} & w^T \Sigma w \\ \text{subject to} & \Sigma \in \mathcal{S}, \quad \Sigma \succeq 0 \end{array}$$

  with variable Σ
- ...a convex problem with variable Σ

- if the worst-case risk is not too bad, you can worry less
- if not, you'll confront your worst nightmare

Example

- $w = (-0.01, 0.13, 0.18, 0.88, -0.18)$
- optimized for Σ^{nom}, return 0.1, leverage limit 2
- $\mathcal{S} = \left\{\Sigma^{\text{nom}} + \Delta \ : \ |\Delta_{ii}| = 0,\ |\Delta_{ij}| \leq 0.2\right\}$,

$$\Sigma^{\text{nom}} = \begin{bmatrix} 0.58 & 0.2 & 0.57 & -0.02 & 0.43 \\ 0.2 & 0.36 & 0.24 & 0 & 0.38 \\ 0.57 & 0.24 & 0.57 & -0.01 & 0.47 \\ -0.02 & 0 & -0.01 & 0.05 & 0.08 \\ 0.43 & 0.38 & 0.47 & 0.08 & 0.92 \end{bmatrix}$$

Example

- nominal risk $= 0.168$
- worst case risk $= 0.422$

$$\text{worst case } \Delta = \begin{bmatrix} 0 & 0.04 & -0.2 & -0. & 0.2 \\ 0.04 & 0 & 0.2 & 0.09 & -0.2 \\ -0.2 & 0.2 & 0 & 0.12 & -0.2 \\ -0. & 0.09 & 0.12 & 0 & -0.18 \\ 0.2 & -0.2 & -0.2 & -0.18 & 0 \end{bmatrix}$$

Outline

Portfolio Optimization

Worst-Case Risk Analysis

Optimal Advertising

Ad display

- m advertisers/ads, $i = 1, \ldots, m$
- n time slots, $t = 1, \ldots, n$
- T_t is total traffic in time slot t
- $D_{it} \geq 0$ is number of ad i displayed in period t
- $\sum_i D_{it} \leq T_t$
- contracted minimum total displays: $\sum_t D_{it} \geq c_i$
- goal: choose D_{it}

Clicks and revenue

- C_{it} is number of clicks on ad i in period t
- click model: $C_{it} = P_{it}D_{it}$, $P_{it} \in [0, 1]$
- payment: $R_i > 0$ per click for ad i, up to budget B_i
- ad revenue

$$S_i = \min \left\{ R_i \sum_t C_{it}, B_i \right\}$$

…a concave function of D

Ad optimization

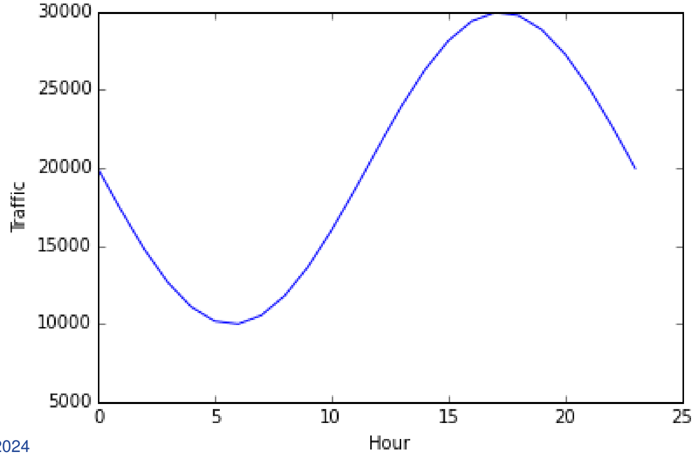
- choose displays to maximize revenue:

$$\begin{array}{ll} \text{maximize} & \sum_i S_i \\ \text{subject to} & D \geq 0, \quad D^T \mathbf{1} \leq T, \quad D\mathbf{1} \geq c \end{array}$$

- variable is $D \in \mathbf{R}^{m \times n}$
- data are T, c, R, B, P

Example

- 24 hourly periods, 5 ads (A through E)
- total traffic:



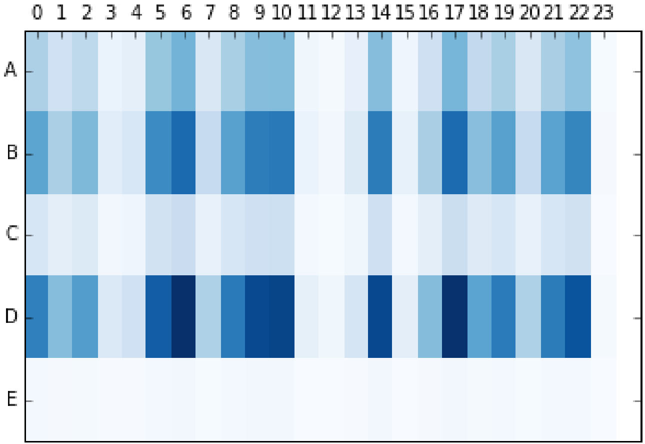
Example

- ad data:

Ad	A	B	C	D	E
c_i	61000	80000	61000	23000	64000
R_i	0.15	1.18	0.57	2.08	2.43
B_i	25000	12000	12000	11000	17000

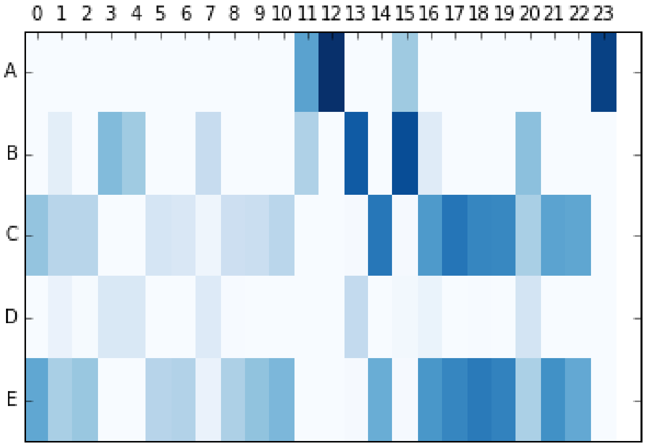
Example

P_{it}



Example

optimal D_{it}



Example

ad revenue

Ad	A	B	C	D	E
c_i	61000	80000	61000	23000	64000
R_i	0.15	1.18	0.57	2.08	2.43
B_i	25000	12000	12000	11000	17000
$\sum_t D_{it}$	61000	80000	148116	23000	167323
S_i	182	12000	12000	11000	7760