Convex Optimization

Stephen Boyd

Thanks to Giray Ogut and Kasper Johansson

Master's Forum, CUHK-SZ, November 26 2024

Outline

Optimization problem

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

- $x \in \mathbf{R}^n$ is (vector) variable to be chosen (n scalar variables $x_1, \ldots, x_n$)
- f_0 is the **objective function**, to be minimized
- $f_1, \ldots, f_m$ are the **inequality constraint functions**
- $g_1, \ldots, g_p$ are the **equality constraint functions**

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

Finding good (or best) actions

- x represents some **action**, *e.g.*,
  - trades in a portfolio
  - airplane control surface deflections
  - schedule or assignment
  - resource allocation
- constraints limit actions or impose conditions on outcome
- the smaller the objective $f_0(x)$, the better
  - total cost (or negative profit)
  - deviation from desired or target outcome
  - risk
  - fuel use

Finding good models

- x represents the **parameters** in a model
- constraints impose requirements on model parameters (*e.g.*, nonnegativity)
- objective $f_0(x)$ is sum of two terms:
  - a prediction error (or loss) on some observed data
  - a (regularization) term that penalizes model complexity

Worst-case analysis

- variables are actions or parameters out of our control
  (and possibly under the control of an adversary)
- constraints limit the possible values of the parameters
- minimizing $-f_0(x)$ finds **worst possible parameter values**

- if the worst possible value of $f_0(x)$ is tolerable, you're OK
- it's good to know what the worst possible scenario can be

Optimization-based models

- model an entity as taking actions that solve an optimization problem
  - an individual makes choices that maximize expected utility
  - an organism acts to maximize its reproductive success
  - reaction rates in a cell maximize growth
  - currents in a circuit minimize total power
- (except the last) these are **very crude** models
- and yet, they often work very well

Basic use model for mathematical optimization

- instead of saying how to choose (action, model) x
- you articulate what you want (by stating the problem)
- then let an algorithm decide on (action, model) x

- **say what you want, not how to get it**

Can you solve it?

- generally, no
- but you can try to solve it approximately, and it often doesn't matter

- the exception: **convex optimization**
  - includes linear programming (LP), quadratic programming (QP), many others
  - we can solve these problems reliably and efficiently
  - comes up in many applications across many fields

Outline

Convex optimization

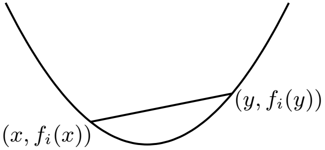
convex optimization problem:

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

- 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**



When is an optimization problem hard to solve?

- classical view:
  - linear (zero curvature) is easy
  - nonlinear (nonzero curvature) is hard
- the classical view is **wrong**
- the correct view:
  - convex (nonnegative curvature) is easy
  - nonconvex (negative curvature) is hard

Solving convex optimization problems

- many different algorithms (that run on many platforms)
  - interior-point methods for up to 10000s of variables
  - first-order methods for larger problems
  - do not require initial point, babysitting, or tuning
- can develop and deploy quickly using modeling languages such as CVXPY
- solvers are reliable, so can be embedded
- code generation yields real-time solvers that execute in milliseconds

Application areas

- machine learning, statistics
- finance
- supply chain, revenue management, advertising
- control
- signal and image processing, vision
- networking
- circuit design
- combinatorial optimization
- quantum mechanics
- flux-based analysis
- many others . . .

Modeling languages for convex optimization

- domain specific languages (DSLs) for convex optimization
  - describe problem in high level human readable language, close to the math
  - can automatically verify problem as convex
  - can automatically transform problem to standard form, then solve

- enables rapid prototyping
- it's now much easier to develop an optimization-based application
- ideal for teaching and research (can do a lot with short scripts)

- gets close to the basic idea: **say what you want, not how to get it**

Implementations

- CVXPY (Python) [Diamond and Boyd, 2014]
- Convex.jl (Julia) [Udell et al., 2014]
- CVXR (R) [Fu, Narasimhan, and Boyd, 2017]
- CVX (Matlab) [Grant and Boyd, 2006]
- YALMIP (Matlab) [Lofberg, 2004]

CVXPY example: Non-negative least squares

math:

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

- A, b given
- variable is x
- $x \geq 0$ means elementwise

CVXPY code:

```
import cvxpy as cp

A, b = ...

x = cp.Variable(n)
obj = cp.norm2(A @ x - b)**2
constr = [x >= 0]
prob = cp.Problem(cp.Minimize(obj), constr)
prob.solve()
```

CVXPY

- 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 (if not most) hedge funds
- over 27,000,000 downloads on PyPI

Outline

Mean-variance (Markowitz) optimization

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

- variable $w \in \mathbf{R}^n$ of portfolio weights
- $\mu \in \mathbf{R}^n$ and $\Sigma \in \mathbf{S}^n_{++}$ are (estimates of) asset return mean and covariance
- $\gamma > 0$ is risk aversion parameter
- basic form goes back to [Markowitz, 1952]
- can be sensitive to estimation error

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

Adding practical constraints and objective terms

- account for trading cost $\kappa^T|w - w^{\mathrm{pre}}|$;
  w^{pre} is previous holdings, $\kappa > 0$ is vector of one half bid-ask spreads
- add transaction cost, limit weights and leverage

$$\begin{array}{ll} \text{maximize} & \mu^T w - \gamma^{\mathrm{risk}} w^T \Sigma w - \gamma^{\mathrm{trade}} \kappa^T |w - w^{\mathrm{pre}}| \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w^{\min} \leq w \leq w^{\max}, \quad \|w\|_1 \leq L^{\max} \end{array}$$

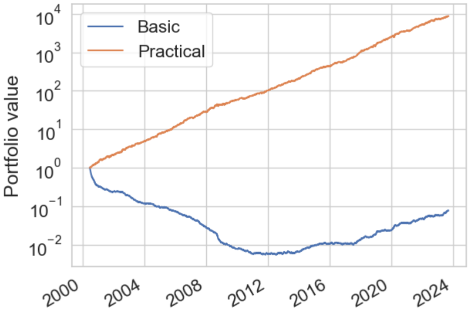
- can be implemented in a few lines in CVXPY
- this version is quite practical
- see [Boyd et al., 2024] for details, further improvements, reference implementation

Practical Markowitz in CVXPY

```python
risk = cp.quad_form(w, Sigma)
t_cost = kappa.T @ cp.abs(w - w_pre)

objective = mu.T @ w - gamma_risk * risk - gamma_trade * t_cost
constraints = [w.sum() == 1, w >= w_min, w <= w_max, cp.norm1(w) <= L_max]

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

Practical Markowitz: Example

- S&P 100, simulated but realistic μ, iterated EWMA covariance forecast
- $\gamma = 250$, $\gamma^{\text{risk}} = 35$, $\gamma^{\text{trade}} = 5$ (chosen to attain comparable risks)



Metric	Basic	Practical
Return	-9.8%	38.1%
Risk	11.6%	11.9%
Sharpe	-0.8	3.2
Drawdown	99.4%	11.2%

Sparse inverse covariance estimation

- model random vector $x \in \mathbf{R}^n$ as $x \sim \mathcal{N}(0, \Sigma)$
- log-likelihood on data $x^1, \ldots, x^N$ is

$$l(\theta) = \frac{N}{2}(\log\det\theta - \mathbf{Tr}\,\theta S) + c, \qquad S = \frac{1}{N}\sum_{i=1}^{N} x^i (x^i)^T$$

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

$$\text{maximize} \quad l(\theta) - \lambda \textstyle\sum_{i<j} |\theta_{ij}|$$

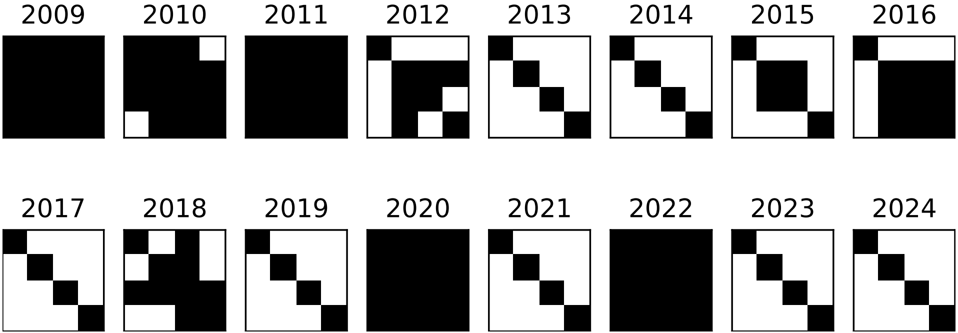
  with variable θ; $\lambda > 0$ is a (sparsity) regularization parameter
- a convex problem; yields matrix with sparse precision matrix θ
- $\theta_{ij} = 0$ means entries x_i, x_j are conditionally independent given the others

Sparse inverse covariance estimation: CVXPY

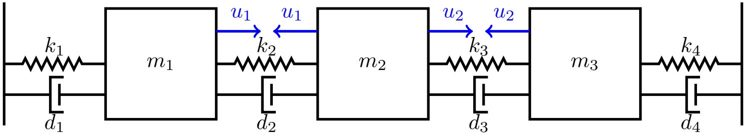
```
N, n = X.shape
Theta = cp.Variable((n, n), symmetric=True)

S = X.T @ X / N
log_likelihood = N / 2 * (cp.log_det(Theta) - cp.trace(Theta @ S))

mask = np.triu(np.ones((n, n)), k=1).astype(bool)
objective = log_likelihood - lam * cp.norm1(Theta[mask])

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

Sparse inverse covariance estimation: Example

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



Position control via tensions



- masses m_1, m_2, m_3 connected by springs and dampers
- position, velocity $p_t, v_t \in \mathbf{R}^3$
- tensions $u_t \in \mathbf{R}^2$, $0 \leq u_t \leq U^{\max}$
- initial state $p_1 = 0$, $v_1 = 0$; terminal state $p_T = p^{\text{des}}$, $v_T = 0$
- p^{des} is the target equilibrium position, with tensions u^{des}
- minimize $\sum_{t=1}^{T} \left(\|p_t - p^{\text{des}}\|_2^2 + \|v_t\|_2^2\right) + \lambda \sum_{t=1}^{T-1} \|u_t - u^{\text{des}}\|_2^2$, $\lambda > 0$

Position control via tensions: Dynamics

dynamics (discretized with sample time $h > 0$)

$$\frac{p_{t+1} - p_t}{h} = v_t, \qquad M\frac{v_{t+1} - v_t}{h} = \left(-AKA^T p_t - ADA^T v_t + Bu_t\right)$$

$M = \mathbf{diag}(m_1, m_2, m_3)$, $K = \mathbf{diag}(k_1, k_2, k_3, k_4)$, $D = \mathbf{diag}(d_1, d_2, d_3, d_4)$,

$$A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{bmatrix}$$

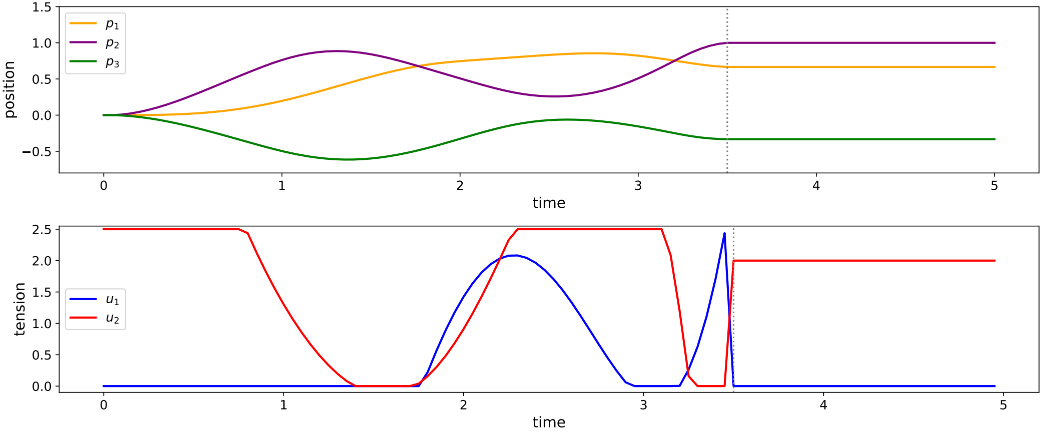
Position control via tensions: CVXPY

$$
\begin{array}{ll}
\text{minimize} & \sum_{t=1}^{T} \left(\|p_t - p^{\text{des}}\|_2^2 + \|v_t\|_2^2\right) + \lambda \sum_{t=1}^{T-1} \|u_t - u^{\text{des}}\|_2^2 \\
\text{subject to} & p_1 = 0, \quad v_1 = 0, \quad p_T = p^{\text{des}}, \quad v_T = 0, \quad 0 \le u \le U^{\max}, \\
& p_{t+1} = p_t + h v_t, \quad t = 1, \ldots, T-1 \\
& v_{t+1} = v_t + h M^{-1}\left(-AKA^T p_t - ADA^T v_t + B u_t\right), \quad t = 1, \ldots, T-1
\end{array}
$$

```python
import cvxpy as cp
A, M_inv, D, K, B, u_max, p_des, u_des, T, lam = ...
p, v, u = cp.Variable((T, 3)), cp.Variable((T, 3)), cp.Variable((T-1, 2))
obj = cp.sum_squares(p - p_des) + cp.sum_squares(v) + lam * cp.sum_squares(u-u_des)
cons = [p[0] == 0, v[0] == 0, p[-1] == p_des[-1], v[-1] == 0, 0 <= u, u <= u_max]
cons += [p[t+1] == p[t] + h * v[t] for t in range(T-1)]
cons += [v[t+1] == v[t] + h * M_inv @ (-(A @ K @ A.T) @ p[t] -
        (A @ D @ A.T) @ v[t] + B @ u[t]) for t in range(T-1)]
prob = cp.Problem(cp.Minimize(obj), cons)
prob.solve()
```

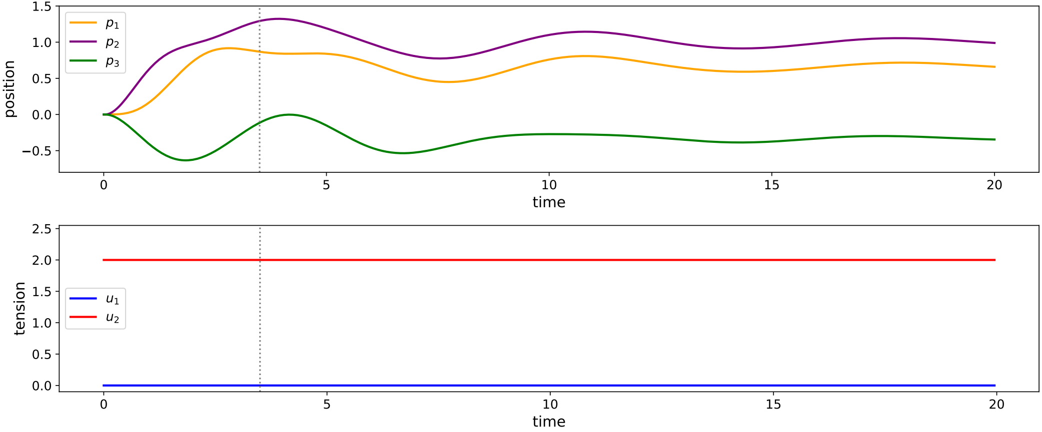
Optimal control via tensions: Example

- reaches target position $p^{\mathrm{des}} = (0.67, 1.00, -0.33)$ at $t = 3.5$



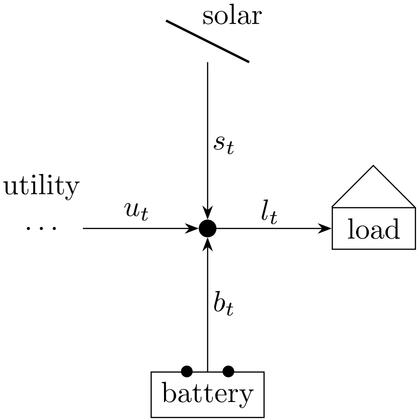
Control via constant tensions

- use tensions $u_t = u^{\text{des}}$, wait for $p_t \to p^{\text{des}}$

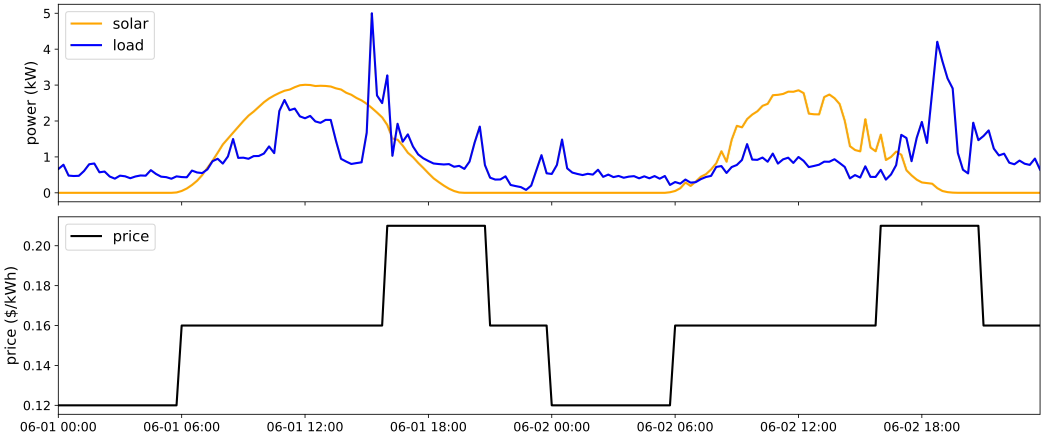


Residential energy management

- given data (in 15 minute intervals):
  - load $l_t \geq 0$
  - utility price $p_t \geq 0$
  - solar capacity $S_t \geq 0$
- variables:
  - utility power $u_t \geq 0$
  - solar power s_t, $0 \leq s_t \leq S_t$
  - battery power b_t, $|b_t| \leq B$
  - battery charge q_t, $0 \leq q_t \leq Q$
- objective and constraints:
  - minimize utility cost $\sum_{t=1}^{T} p_t u_t$
  - plus battery wear $\lambda \sum_{t=1}^{T} |b_t|$, $\lambda > 0$
  - power balance $u_t + s_t + b_t = l_t$
  - battery dynamics $q_{t+1} = q_t - 0.25 b_t$
  - $q_{T+1} = q_1$



Residential energy management: Data



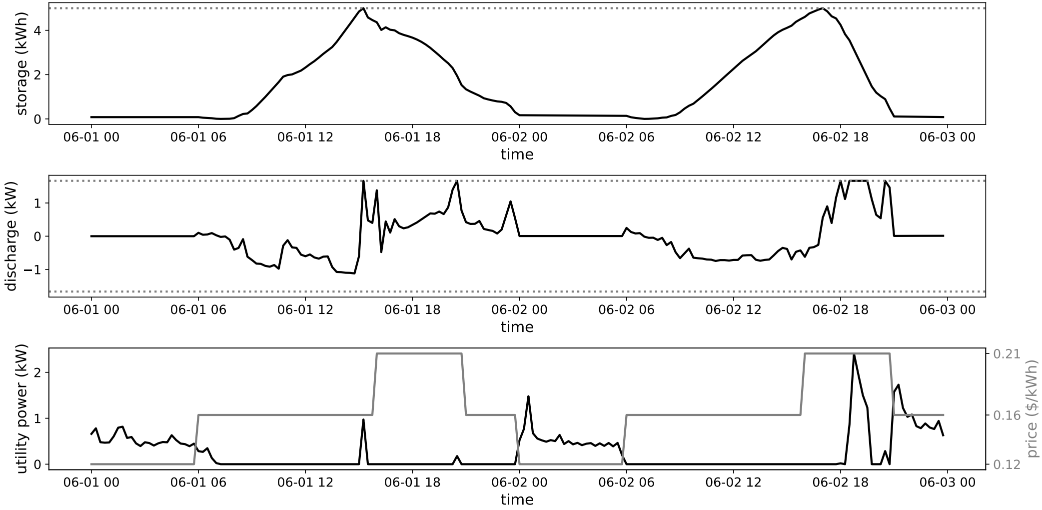
Residential energy management: CVXPY

$$
\begin{array}{ll}
\text{minimize} & \sum_{t=1}^{T} (p_t u_t + \lambda |b_t|) \\
\text{subject to} & 0 \leq q_t \leq Q, \quad 0 \leq u_t, \quad |b_t| \leq B, \quad 0 \leq s_t \leq S_t \\
& u_t + s_t + b_t = l_t, \quad t = 1, \ldots, T \\
& q_1 = q_{T+1}, \quad q_{t+1} = q_t - 0.25 b_t, \quad t = 1, \ldots, T
\end{array}
$$

```python
import cvxpy as cp
p, l, B, Q, S, T, lam = ...
q, b, u, s = cp.Variable(T+1), cp.Variable(T), cp.Variable(T), cp.Variable(T)
obj = p.T @ u + lam * cp.norm(b, 1)
cons = [0 <= q, q <= Q, 0 <= u, cp.abs(b) <= B, 0 <= s, s <= S]
cons += [u + s + b == l, q[1:] == q[:-1] - 0.25 * b, q[0] == q[-1]]
prob = cp.Problem(cp.Minimize(obj), cons)
prob.solve()
```

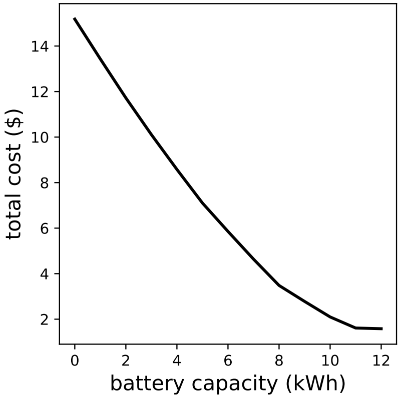
Residential energy management: Results

- battery capacity $Q = 5$ kWh, max charge/discharge $B = 1.67$ kW



Residential energy management: Storage/cost trade-off

- optimal cost vs. battery capacity Q
- max charge/discharge $B = Q/3$



Outline

Showing a function is convex

methods for establishing convexity of a function f

1. verify definition inequality: for all x, y, $\theta \in [0, 1]$,

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

2. for twice differentiable functions, show $\nabla^2 f(x) \geq 0$ (*i.e.*, is PSD)

3. construct f from simple convex functions, using operations that preserve convexity (*e.g.*, sum, maximum, positive scaling, ...)

method 3 is **by far** the most useful in practice

Composition rule

- composition of $g : \mathbf{R}^n \to \mathbf{R}^k$ and $h : \mathbf{R}^k \to \mathbf{R}$ is

$$f(x) = h(g(x)) = h(g_1(x), g_2(x), \ldots, g_k(x))$$

- f is convex if h is convex and for each i one of the following holds
  - g_i convex, h nondecreasing in its ith argument
  - g_i concave, h nonincreasing in its ith argument
  - g_i affine

- there is a similar rule for concave h

- composition rule subsumes others, *e.g.*,
  - αf is convex if f is, and $\alpha \geq 0$
  - sum of convex (concave) functions is convex (concave)
  - max (min) of convex (concave) functions is convex (concave)

Example

the function

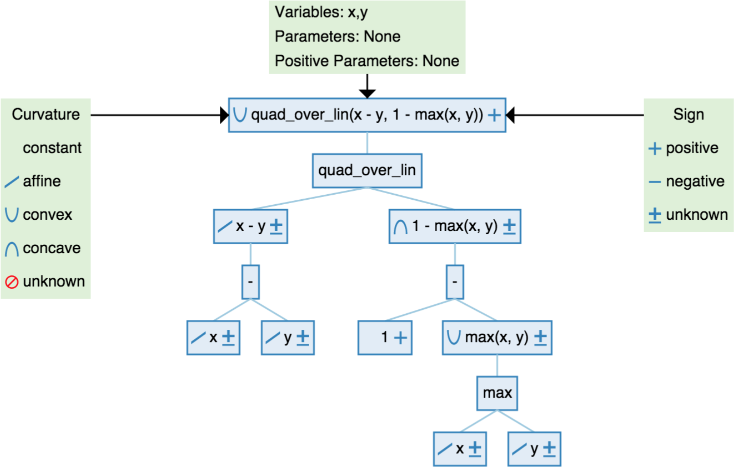
$$f(x, y) = \frac{(x-y)^2}{1 - \max(x, y)}, \qquad x < 1, \quad y < 1$$

is convex

constructive analysis:

- (leaves) x, y, and 1 are affine
- $\max(x, y)$ is convex; $x - y$ is affine
- $1 - \max(x, y)$ is concave
- function u^2/v is convex, monotone decreasing in v for $v > 0$
- f is composition of u^2/v with $u = x - y$, $v = 1 - \max(x, y)$, hence convex

Example (from dcp.stanford.edu)



Disciplined convex programming

in **disciplined convex programming** (DCP) users construct convex and concave functions as expressions using constructive convex analysis

- expressions formed from
  - **variables**,
  - **constants**,
  - and **atomic functions** from a library
- atomic functions have known convexity, monotonicity, and sign properties
- all subexpressions match general composition rule
- a valid DCP function is convex-by-construction

CVXPY example

$$\frac{(x-y)^2}{1-\max(x,y)}, \qquad x < 1, \quad y < 1$$

```python
import cvxpy as cp
x = cp.Variable()
y = cp.Variable()
expr = cp.quad_over_lin(x - y, 1 - cp.maximum(x, y))
expr.curvature # Convex
expr.sign # Positive
expr.is_dcp() # True
```

(atom `quad_over_lin(u,v)` includes domain constraint `v>0`)

Outline

Mathematical optimization

Convex optimization

Examples

Disciplined convex programming

Code generation

Conclusions

Real-time embedded optimization

- in many applications, need to solve the same problem repeatedly with different data, often with a real-time constraint
  - control: update actions as sensor signals, goals change
  - finance: rebalance portfolio as prices, predictions change
- requires extreme solver reliability, hard real-time execution
- used now when solve times are measured in minutes, hours
  - supply chain, chemical process control, trading
- (using new techniques) can be used for applications with solve times measured in **milliseconds** or **microseconds**

Code generators

CVXGEN [Mattingley and Boyd, 2010]

- generates custom QP solver in C
- used in many applications, including all of SpaceX first stage landings

CVXPYgen [Schaller et al., 2022]

- open-source code generator based on CVXPY
- easy prototype to implementation path
- generates C
- supports multiple solvers, non-QP problems

CVXGEN in action

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

Outline

Summary

convex optimization problems

- are optimization problems of a special form
- arise in many applications
- can be solved effectively
- are easy to specify using DSLs such as CVXPY
- can be used in embedded systems with hard real-time constraints

Resources

many researchers have worked on the topics covered

- Convex Optimization (book)
- EE364a (lecture slides, videos, code, homework, . . .)
- software:
  - CVXPY
  - Convex.jl
  - CVXR
  - CVX
- code and data for examples in this talk

all available online