Solving convex optimization problems

Purdue ME 597, Distributed Energy Resources

Kevin J. Kircher

these slides draw on materials by Stephen Boyd at Stanford

Outline

Disciplined convex programming in CVX

Examples

Optimization algorithms

Disciplined convex programming

- is a framework for describing convex optimization problems
- uses a library of functions with curvature, monotonicity tags
- imposes rules for compositions of functions
- is sufficient but not necessary for certifying convexity

Disciplined convex program structure

- (scalar) objective can be
  - minimize convex
  - maximize concave
  - omitted (for feasibility problems)
- constraints can be
  - convex <= concave
  - concave >= convex
  - affine == affine
  - omitted (for unconstrained problems)

⋆ affine functions are both convex and concave

CVX

- implements disciplined convex programming in Matlab
- transforms user-specified convex programs into standard form
- passes standard-form problems to solvers
- interprets solver status (solved, infeasible, unbounded, . . .)
- if solved, transforms solutions back to user-specified forms

CVX syntax

```
cvx_begin
  variable x(n,1)
  minimize( norm(x,Inf) )
  subject to
    A*x == b
cvx_end
```

- constants $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$ are defined above CVX scope
- within CVX scope, x is a variable
- after `cvx_end`, CVX populates
  - `cvx_status` with solver's exit status
  - x with solution (if `cvx_status` is `Solved`)

CVX syntax (continued)

- indentation doesn't matter
- '`subject to`' is unnecessary, but can improve readability
- equality constraints use ==, not = (assignment)
- CVX interprets inequalities like `x >= 0` elementwise
- CVX does not require an initial guess or function derivatives

Infeasible problems

if problem instance is infeasible, CVX populates

- `cvx_status` with `Infeasible`
- each element of x with `NaN`

Unbounded problems

- if problem instance is unbounded, CVX populates
  - ◇ `cvx_status` with `Unbounded`
  - ◇ x with a **direction** in which problem is unbounded
- the direction x is likely not feasible, but for any feasible $\tilde{x}$,
  - ◇ $\tilde{x} + \alpha x$ is feasible for all $\alpha \geq 0$
  - ◇ objective value of $\tilde{x} + \alpha x$ improves without bound as $\alpha \to \infty$
- to get a feasible $\tilde{x}$, omit objective and re-solve as feasibility problem

Some example functions

function	meaning	attributes
`max(x)`	$\max\{x_1, \ldots, x_n\}$	convex nondecreasing
`min(x)`	$\min\{x_1, \ldots, x_n\}$	concave nondecreasing
`pos(x)`	$\max\{0, x\}$	convex nondecreasing
`square_pos(x)`	$\max\{0, x\}^2$	convex nondecreasing
`inv_pos(x)`	$1/x$ (for $x > 0$)	convex nonincreasing
`sqrt(x)`	$\sqrt{x}$ (for $x \geq 0$)	concave nondecreasing
`norm(x,p)`	$\|x\|_p$	convex
`sum_square(x)`	$x_1^2 + \cdots + x_n^2$	convex

Quadratic forms

- for $P \in \mathbf{R}^{n \times n}$, $x^\top P x$ is a **quadratic form** in $x \in \mathbf{R}^n$
- can assume P is symmetric; if it's not, replace P by $(P + P^\top)/2$:

$$\begin{aligned} x^\top (P + P^\top) x/2 &= (x^\top P x + x^\top P^\top x)/2 \\ &= (x^\top P x + (x^\top P^\top x)^\top)/2 \\ &= (x^\top P x + x^\top P x)/2 \\ &= x^\top P x \end{aligned}$$

- in CVX, $x^\top P x$ is `quad_form(x,P)`

Convexity and quadratic forms

- a symmetric $P \in \mathbf{R}^{n\times n}$ is **positive semidefinite** ($P \succeq 0$) if

$$x^\top Px \geq 0 \text{ for all } x$$

$(\iff \mathbf{det}\, P \geq 0 \iff \lambda_i \geq 0$ for all eigenvalues λ_i of $P)$

- a symmetric $P \in \mathbf{R}^{n\times n}$ is **positive definite** ($P \succ 0$) if

$$x^\top Px > 0 \text{ for all } x \neq 0$$

$(\iff \mathbf{det}\, P > 0 \iff \lambda_i > 0$ for all eigenvalues λ_i of $P)$

- the quadratic form $x^\top Px$ is
  - ◇ convex if $P \succeq 0$
  - ◇ strictly convex (so has a **unique** global minimum) if $P \succ 0$

Quadratic forms in CVX

- `quad_form` and `sum_square` tend to be slow
- using `norm` instead can improve speed and accuracy
- for example, minimizing the least squares objective

$$\texttt{sum\_square(A*x - b)} = (Ax-b)^\top(Ax-b) = \|Ax-b\|_2^2$$

  can typically be done faster by minimizing

$$\texttt{norm(A*x - b)} = \|Ax-b\|_2 = \sqrt{\|Ax-b\|_2^2}$$

- these problems are equivalent since
  - ◇ if g is increasing, minimize $g(f(x)) \iff$ minimize $f(x)$
  - ◇ $\sqrt{\cdot}$ with nonnegative arguments is increasing
  - ◇ $\|\cdot\|_2^2$ is nonnegative

Quadratic forms in CVX (continued)

- another example: (convex) constraint $x^\top Px \le c$ with $x \in \mathbf{R}^n$
- if $P \succ 0$, it has a square root $R \in \mathbf{R}^{n\times n}$ with $R^\top R = P$
  (in Matlab, `R = chol(P)` computes an upper triangular R)
- since $\|y\|_2 = \sqrt{y^\top y}$,

$$\begin{aligned} & x^\top Px \le c \\ \iff & x^\top R^\top Rx \le c \\ \iff & \|Rx\|_2^2 \le c \\ \iff & \|Rx\|_2 \le \sqrt{c} \end{aligned}$$

- in CVX, `quad_form(x,P) <= c` usually works
- but `norm(chol(P)*x) <= sqrt(c)` is usually faster

Outline

Disciplined convex programming in CVX

Examples

Optimization algorithms

Least squares

- choose x to minimize $\|Ax - b\|_2^2$ given $A \in \mathbf{R}^{m \times n}$, b
- random problem instance:
  - $n = 500$, $m = 1000$
  - independent standard normal A and b
- computing the least squares solution

$$x^\star = (A^\top A)^{-1} A^\top b = \texttt{A\textbackslash b}$$

takes 0.0145 s on a 2.7 GHz processor

Least squares: CVX `sum_square` solution

```
cvx_begin
  variable x(n,1)
  minimize( sum_square(A*x - b) )
cvx_end
```

- solves in 2.32 s
- agrees with `A\b` to nine decimal places

Least squares: CVX `norm` solution

```
cvx_begin
  variable x(n,1)
  minimize( norm(A*x - b) )
cvx_end
```

- solves in 1.35 s (42% less than `sum_square`)
- also agrees with `A\b` to nine decimal places

Least squares: disciplined convex programming error

```
cvx_begin
  variable x(n,1)
  minimize( norm(A*x - b)^2 )
cvx_end
```

```
Disciplined convex programming error:
  Illegal operation:  {convex} .^ {2}
  (Consider POW_P, POW_POS, or POW_ABS instead.)
```

- square of norm matches no composition rule
  (a convex function of a convex function may not be convex)
- but CVX would allow `square_pos(norm(A*x - b))` since

$$\texttt{square\_pos(z)} = \max\{0, z\}^2$$

  is convex *and nondecreasing*

Outline

Disciplined convex programming in CVX

Examples

Optimization algorithms

Why learn about optimization algorithms?

- tools like CVX require no knowledge of how solvers work
- but knowing a bit can help with debugging, interpreting results
- also, optimization algorithms can be clever and beautiful
- we'll just scratch the surface; other classes go much deeper

Smooth unconstrained convex optimization

- choose $x \in \mathbf{R}^n$
- to minimize $f(x)$
- given smooth convex $f : \mathbf{R}^n \to \mathbf{R}$

- optimality condition is $\nabla f(x^\star) = 0$ (n equations, n unknowns)
- for example, if $f(x) = x^\top P x + q^\top x + r$, then

$$\nabla f(x^\star) = 2Px^\star + q = 0$$

  is a system of linear equations that can be solved efficiently
  (if P is invertible, then $x^\star = -P^{-1}q/2$ is the unique solution)
- but general nonquadratic f require iterative methods

Iterative methods

iterative methods

- typically require an **initial guess** $x(0) \in \textbf{dom}\, f$
- produce a sequence of **iterates** $x(1), x(2), \ldots \in \textbf{dom}\, f$
- **converge** if $f(x(k)) \to f(x^\star)$ and $\nabla f(x(k)) \to 0$ as $k \to \infty$

Descent methods

- given initial guess $x(0) \in \mathbf{dom}\, f$, repeat:
  1. find a **descent direction** $d(k)$
  2. find a **step size** $\alpha(k)$
  3. update $x(k+1) = x(k) + \alpha(k)d(k)$
  4. increment k

  until a stopping condition (such as $\|\nabla f(x(k))\|$ small) holds
- descent direction and step size should satisfy
  - $x(k) + \alpha(k)d(k) \in \mathbf{dom}\, f$
  - $f(x(k) + \alpha(k)d(k)) < f(x(k))$

Finding a good step size $\alpha(k)$

- finding a good step size is called a **line search**
- if $\alpha(k)$ is too small, $f(x(k+1)) < f(x(k))$ but progress is slow
- if $\alpha(k)$ is too big, we risk $f(x(k+1)) > f(x(k))$
- one simple line search method:
  - while $f(x(k) + \alpha(k)d(k)) \geq f(x(k))$,
    - set $\alpha(k) \leftarrow \alpha(k)/2$
  - set $x(k+1) = x(k) + \alpha(k)d(k)$
  - set $\alpha(k+1) = 1.2\alpha(k)$

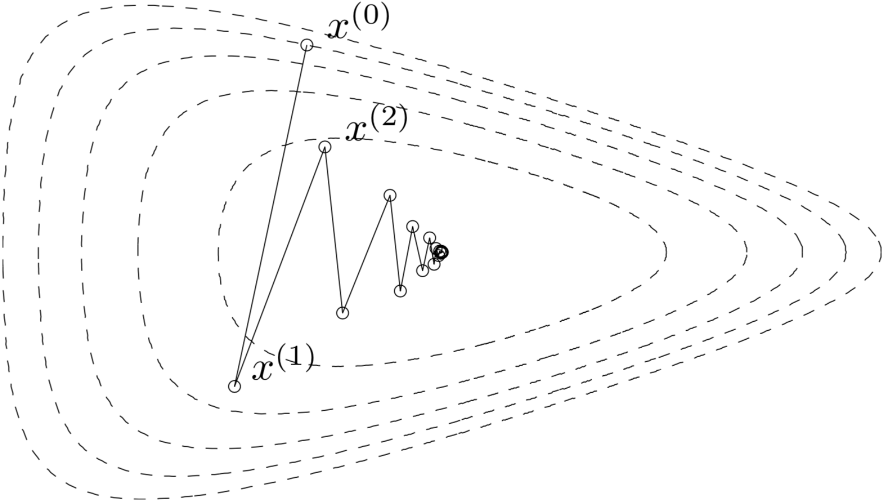
Gradient descent

- $-\nabla f(x)$ points in the direction of steepest descent of f at x
- so gradient descent uses descent direction

$$d(k) = -\nabla f(x(k))$$

- worst case: requires $\sim 1/\varepsilon$ iterations to get $f(x(k)) - f(x^\star) \leq \varepsilon$ (for example, $\sim 10^4$ iterations to get $f(x(k)) - f(x^\star) \leq 10^{-4}$)

Gradient descent illustration



Boyd and Vandenberghe (2004), Convex Optimization

Minimizing quadratic approximations

- Taylor's theorem: the quadratic approximation to f at $\tilde{x}$ is

$$\hat{f}(x) = f(\tilde{x}) + \nabla f(\tilde{x})^\top (x - \tilde{x}) + \frac{1}{2}(x - \tilde{x})^\top \nabla^2 f(\tilde{x})(x - \tilde{x})$$

- $\nabla^2 f(\tilde{x}) \in \mathbf{R}^{n \times n}$ is the second derivative (Hessian) matrix:

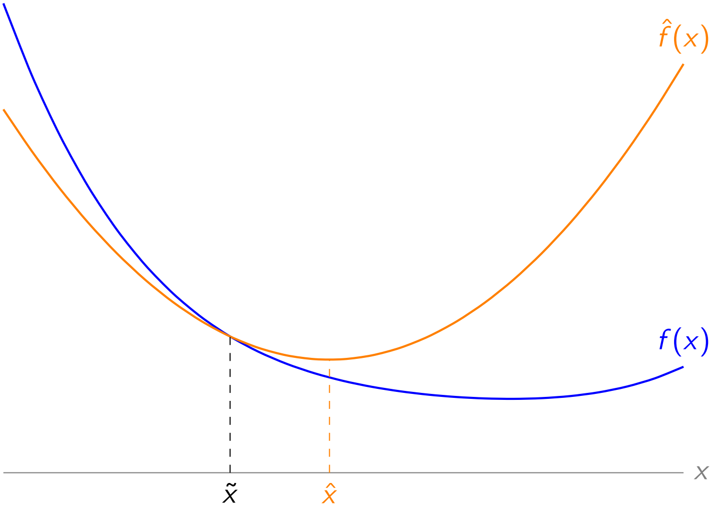
$$\nabla^2 f(\tilde{x})_{ij} = \left. \frac{\partial^2 f}{\partial x_i \partial x_j} \right|_{\tilde{x}}$$

- some algebra shows that if the Hessian is invertible, then

$$\hat{x} = \tilde{x} - \nabla^2 f(\tilde{x})^{-1} \nabla f(\tilde{x})$$

minimizes $\hat{f}(x)$

Quadratic approximation illustration



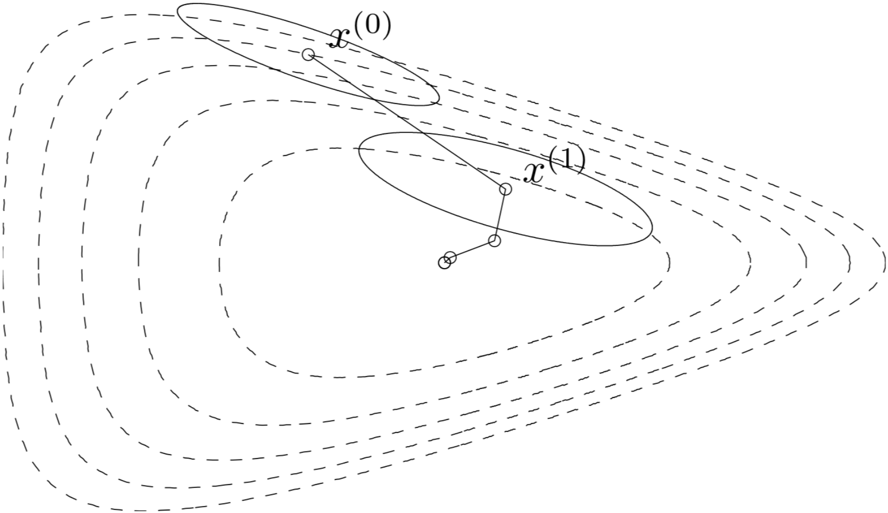
Newton's method

- Newton's method uses the descent direction

$$d(k) = -\nabla^2 f(x(k))^{-1} \nabla f(x(k))$$

  that minimizes the quadratic approximation to f at $x(k)$
- worst case: requires $\sim 1/\sqrt{\varepsilon}$ iterations to get $f(x(k)) - f(x^\star) \leq \varepsilon$ (for example, $\sim 10^2$ iterations to get $f(x(k)) - f(x^\star) \leq 10^{-4}$)

Newton's method illustration



Boyd and Vandenberghe (2004), Convex Optimization

Smooth constrained convex optimization

- choose $x \in \mathbf{R}^n$
- to minimize $f_0(x)$
- subject to $f_1(x) \leq 0, \ldots, f_m(x) \leq 0$
- given smooth convex $f_0, \ldots, f_m : \mathbf{R}^n \to \mathbf{R}$

Logarithmic barrier

- equivalent problem: minimize $f_0(x) + \sum_{i=1}^{m} I_-(f_i(x))$, where

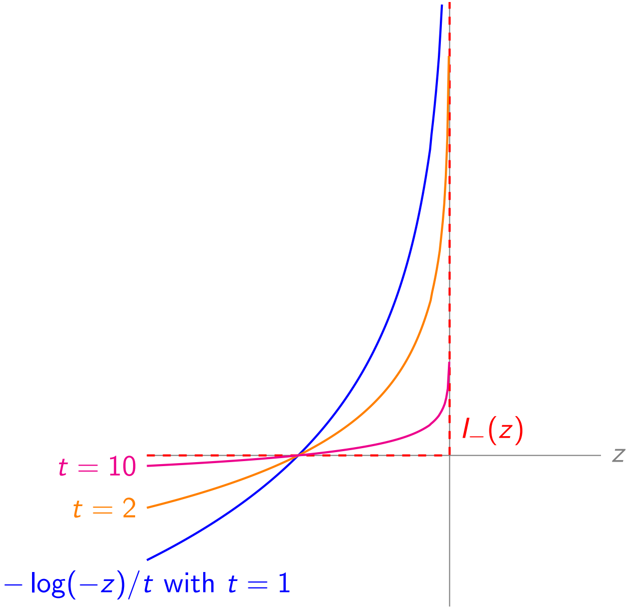
$$I_-(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ \infty & \text{otherwise} \end{cases}$$

  is the **indicator** function of $\{z \in \mathbf{R} \mid z \leq 0\}$
- idea: for a nondecreasing sequence of $t > 0$, minimize

$$f_0(x) - \frac{1}{t} \sum_{i=1}^{m} \log(-f_i(x))$$

- logarithmic barrier function $-\log(-z)/t$ approximates $I_-(z)$
- approximation improves as t increases

Logarithmic barrier approaches indicator as t increases



Barrier methods

- given $t(0) > 0$, $\gamma > 1$, initial guess $x(0) \in \textbf{dom}\, f_0$, repeat:
    1. set $x(k+1)$ by minimizing $f_0(x) - \frac{1}{t(k)} \sum_{i=1}^{m} \log(-f_i(x))$
    2. set $t(k+1) = \gamma t(k)$

  until a stopping condition (such as t large) holds
- step 1 typically uses Newton's method, initialized at $x(k)$
- trade-off: $\gamma \uparrow \implies$ outer iterations $\downarrow$ but Newton iterations $\uparrow$
- barrier methods converge at a rate similar to Newton's method

Interior-point methods

- are used by most solvers that CVX calls
- are conceptually similar to barrier methods
- do not need user-specified initial guesses or function derivatives
- have polynomial-time guarantees on worst-case complexity
- are often very fast in practice
- are typically faster for narrower problem classes:
    - ◇ linear programming (easiest)
    - ◇ quadratic programming
    - ◇ second-order cone programming
    - ◇ semidefinite programming (hardest)