

# 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

Gradient descent

Newton-type methods

# Disciplined convex programming

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

# Disciplined convex program structure

- (scalar) objective can be
  - ◇ minimize convex
  - ◇ maximize concave
  - ◇ omitted (for feasibility problems)
- constraints can be
  - ◇ convex  $\leq$  concave
  - ◇ concave  $\geq$  convex
  - ◇ affine  $=$  affine
  - ◇ omitted (for unconstrained problems)
- ★ affine functions are both convex and concave

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

```
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  $x \geq 0$  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  $\hat{x}$ ,
  - ◊  $\hat{x} + \alpha x$  is feasible for all  $\alpha \geq 0$
  - ◊ objective value of  $\hat{x} + \alpha x$  improves without bound as  $\alpha \rightarrow \infty$
- to get a feasible  $\hat{x}$ , omit objective and re-solve as feasibility problem

## Some example functions

function	meaning	attributes
<code>max(x)</code>	$\max \{x_1, \dots, x_n\}$	convex nondecreasing
<code>min(x)</code>	$\min \{x_1, \dots, x_n\}$	concave nondecreasing
<code>pos(x)</code>	$\max \{0, x\}$	convex nondecreasing
<code>square_pos(x)</code>	$\max \{0, x\}^2$	convex nondecreasing
<code>inv_pos(x)</code>	$1/x$ (for $x > 0$ )	convex nonincreasing
<code>sqrt(x)</code>	$\sqrt{x}$ (for $x \geq 0$ )	concave nondecreasing
<code>norm(x,p)</code>	$\ x\ _p$	convex
<code>sum_square(x)</code>	$x_1^2 + \dots + x_n^2 = \ x\ _2^2$	convex

## Reminder: Least squares (a benchmark problem)

- choose  $x$  to minimize  $\|Ax - b\|_2^2$  given  $A \in \mathbf{R}^{m \times n}$  ( $m \geq n$ ),  $b$
- analytical solution (if  $A$  has linearly independent columns):

$$x^* =$$

- random problem instance:
  - ◊  $n = 500$ ,  $m = 1000$
  - ◊ independent standard normal  $A$  and  $b$
- computing  $A \setminus b$  takes 0.01 s on a 4.5 GHz processor

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

- solves in 0.75 s (75 times slower than  $A \setminus b$ )
- agrees with  $A \setminus b$  to nine decimal places

# Disciplined convex programming errors

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

is convex *and nondecreasing*

# Outline

Disciplined convex programming in CVX

Gradient descent

Newton-type methods

# 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  $f : \mathbf{R}^n \rightarrow \mathbf{R}$
- optimality condition is  $\nabla f(x^*) = 0$  ( $n$  equations,  $n$  unknowns)
- for example, if  $f(x) = x^\top P x + q^\top x + r$ , then

$$\nabla f(x^*) =$$

is a system of **linear** equations that can be solved efficiently  
(if  $P$  is invertible, then  $x^* =$                       is the unique solution)

- but general nonquadratic  $f$  require iterative methods

# Iterative methods

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

- 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
  - ◇
  - ◇

## 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  $\alpha(k+1) = 1.2\alpha(k)$

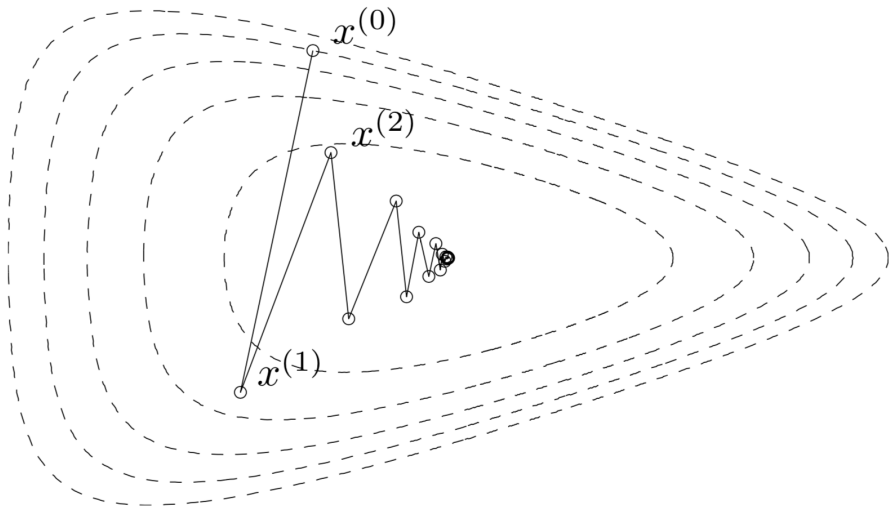
# Gradient descent

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

$$d(k) =$$

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

# Gradient descent illustration



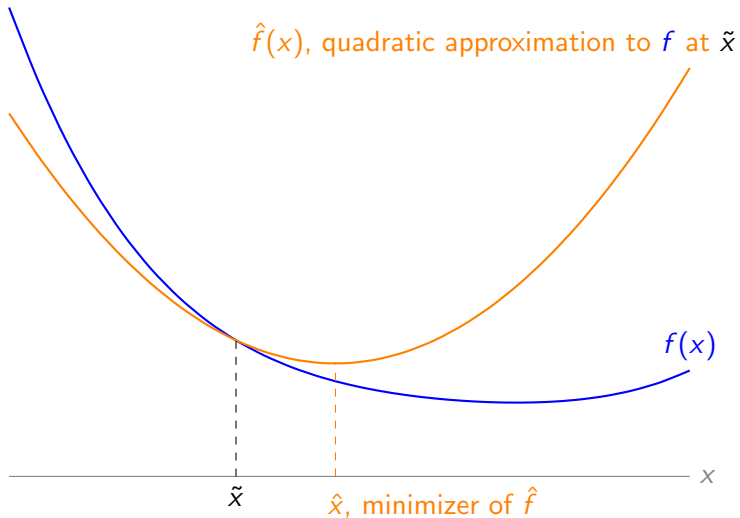
# Outline

Disciplined convex programming in CVX

Gradient descent

Newton-type methods

## Quadratic approximations



# Minimizing quadratic approximations

- Taylor's theorem: the quadratic approximation to  $f$  at  $\tilde{x}$  is
- $\nabla^2 f(\tilde{x}) \in \mathbf{R}^{n \times n}$  is the second derivative (Hessian) matrix:
- some algebra shows that if the Hessian is invertible, then

$$\hat{x} =$$

minimizes  $\hat{f}(x)$

# Newton's method

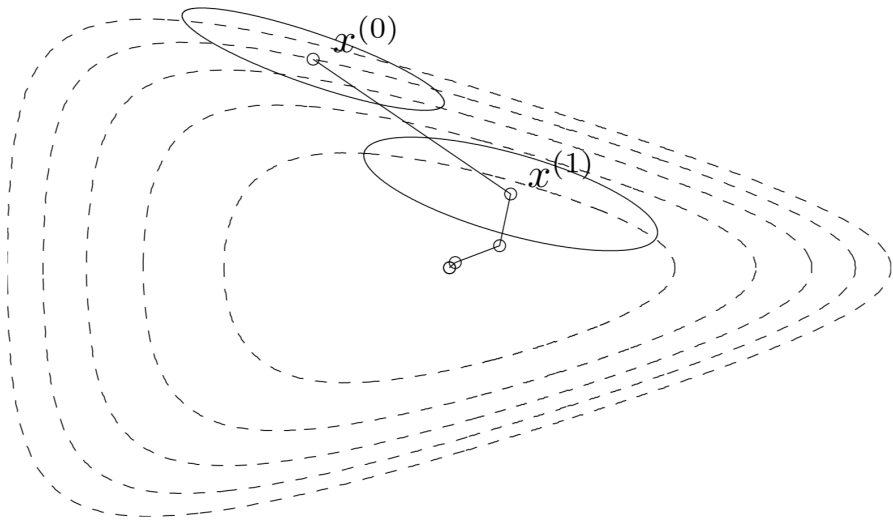
- Newton's method uses the descent direction

$$d(k) =$$

that minimizes the quadratic approximation to  $f$  at  $x(k)$

- worst case: requires  $\sim 1/\sqrt{\varepsilon}$  iterations to get  $f(x(k)) - f(x^*) \leq \varepsilon$   
(for example,  $\sim 10^2$  iterations to get  $f(x(k)) - f(x^*) \leq 10^{-4}$ )

# Newton's method illustration



# Smooth constrained convex optimization

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

# Logarithmic barrier

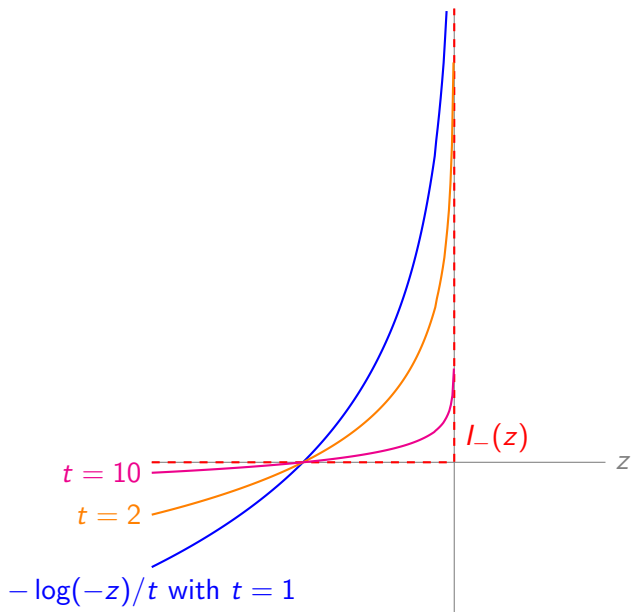
- equivalent problem: minimize  $f_0(x) + \sum_{i=1}^m l_-(f_i(x))$ , where

$$l_-(z) =$$

is the **indicator** function of  $\{z \in \mathbf{R} \mid z \leq 0\}$

- idea: for a nondecreasing sequence of  $t > 0$ , minimize
  
- logarithmic barrier function  $-\log(-z)/t$  approximates  $l_-(z)$
- approximation improves as  $t$  increases

# Logarithmic barrier approaches indicator as $t$ increases



# Barrier methods

- given  $t(0) > 0$ ,  $\gamma > 1$ , initial guess  $x(0) \in \mathbf{dom} f_0$ , 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)