Convex Optimization

Stephen Boyd

Electrical Engineering
Computer Science
Management Science and Engineering
Institute for Computational Mathematics & Engineering
Stanford University

Institute for Advanced Study
City University of Hong Kong
17/2/2017

Outline

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
- 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
  - transmitted signal
- 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

Engineering design

- x represents a design (of a circuit, device, structure, ...)
- constraints come from
  - manufacturing process
  - performance requirements
- objective $f_0(x)$ is combination of cost, weight, power, ...

Finding good models

- x represents the *parameters* in a model
- constraints impose requirements on model parameters (*e.g.*, nonnegativity)
- objective $f_0(x)$ is the prediction error on some observed data (and possibly a term that penalizes model complexity)

Inversion

- x is something we want to estimate/reconstruct, given some measurement y
- constraints come from prior knowledge about x
- objective $f_0(x)$ measures deviation between predicted and actual measurements

Worst-case analysis (pessimization)

- 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

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

Summary

- **summary**: optimization arises *everywhere*

Summary

- **summary**: optimization arises *everywhere*

- **the bad news**: most optimization problems are *intractable*
  i.e., *we cannot solve them*

Summary

- **summary**: optimization arises *everywhere*

- **the bad news**: most optimization problems are *intractable*
  i.e., *we cannot solve them*

- **an exception**: *convex optimization problems are tractable*
  i.e., we (generally) *can* solve them

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

Why

- beautiful, nearly complete theory
  - duality, optimality conditions, ...

Why

- beautiful, nearly complete theory
  - duality, optimality conditions, ...
- effective algorithms, methods (in theory and practice)
  - get **global solution** (and optimality certificate)
  - polynomial complexity

Why

- beautiful, nearly complete theory
  - duality, optimality conditions, ...
- effective algorithms, methods (in theory and practice)
  - get **global solution** (and optimality certificate)
  - polynomial complexity
- conceptual unification of many methods

Why

- beautiful, nearly complete theory
  - duality, optimality conditions, ...
- effective algorithms, methods (in theory and practice)
  - get **global solution** (and optimality certificate)
  - polynomial complexity
- conceptual unification of many methods
- **lots of applications** (many more than previously thought)

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

The approach

- try to formulate your optimization problem as convex
- if you succeed, you can (usually) solve it (numerically)

The approach

- try to formulate your optimization problem as convex
- if you succeed, you can (usually) solve it (numerically)

- some tricks:
  - change of variables
  - approximation of true objective, constraints
  - *relaxation*: ignore terms or constraints you can't handle

Outline

Radiation treatment planning

- radiation beams with intensities x_j are directed at patient
- radiation dose y_i received in voxel i
- $y = Ax$
- $A \in \mathbf{R}^{m \times n}$ comes from beam geometry, physics
- goal is to choose x to deliver prescribed radiation dose d_i
  - $d_i = 0$ for non-tumor voxels
  - $d_i > 0$ for tumor voxels
- $y = d$ not possible, so we'll need to compromise
- typical problem has $n = 10^3$ beams, $m = 10^6$ voxels

Radiation treatment planning via convex optimization

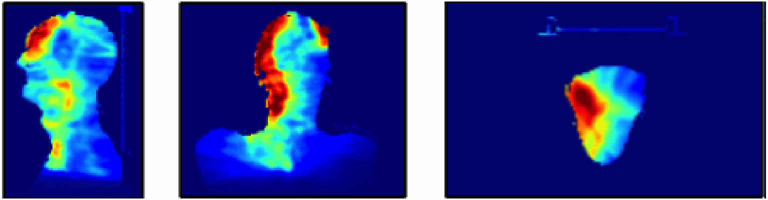
$$
\begin{array}{ll}
\text{minimize} & \sum_i f_i(y_i) \\
\text{subject to} & x \geq 0, \quad y = Ax
\end{array}
$$

- variables $x \in \mathbf{R}^n$, $y \in \mathbf{R}^m$
- objective terms are

$$f_i(y_i) = w_i^{\rm over}(y_i - d_i)_+ + w_i^{\rm under}(d_i - y_i)_+$$

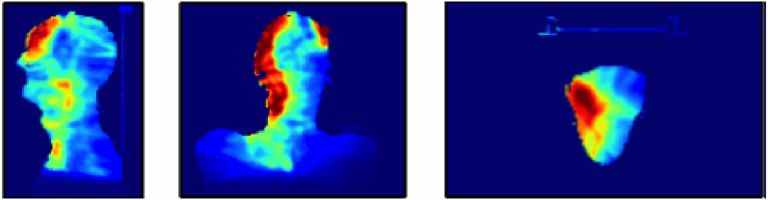
- $w_i^{\rm over}$ and $w_i^{\rm under}$ are positive weights
- *i.e.*, we charge linearly for over- and under-dosing
- a convex optimization problem

Example



- real patient case with $n = 360$ beams, $m = 360000$ voxels
- optimization-based plan essentially the same as plan used

Example



- real patient case with $n = 360$ beams, $m = 360000$ voxels
- optimization-based plan essentially the same as plan used
- (but we computed the plan in a few seconds, not many hours)

Image in-painting

- guess pixel values in obscured/corrupted parts of image
- *total variation in-painting*: choose pixel values $x_{ij} \in \mathbf{R}^3$ to minimize *total variation*

$$\mathrm{TV}(x) = \sum_{ij} \left\| \begin{bmatrix} x_{i+1,j} - x_{ij} \\ x_{i,j+1} - x_{ij} \end{bmatrix} \right\|_2$$

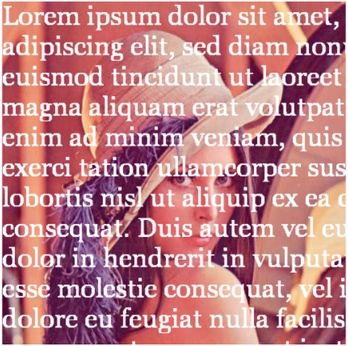
- a convex problem

Example

512 $\times$ 512 color image ($n \approx 800000$ variables)

Original

Corrupted



Example

Original

Recovered

Support vector machine

- goal: predict a Boolean outcome from a set of n features
  - *e.g.*, spam filter, fraud detection, customer purchase

Support vector machine

- goal: predict a Boolean outcome from a set of n features
  - *e.g.*, spam filter, fraud detection, customer purchase
- data (a_i, b_i), $i = 1, \ldots, m$
  - $a_i \in \mathbf{R}^n$ feature vectors; $b_i \in \{-1, 1\}$ Boolean outcomes
- linear predictor: $\hat{b} = \mathrm{sign}(w^T a - v)$
  - $w \in \mathbf{R}^n$ is weight vector; $v \in \mathbf{R}$ is threshold

Support vector machine

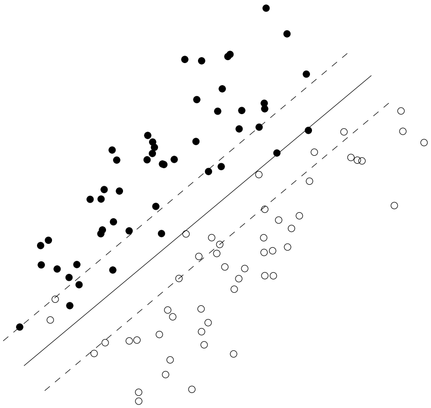
- goal: predict a Boolean outcome from a set of n features
  - *e.g.*, spam filter, fraud detection, customer purchase
- data (a_i, b_i), $i = 1, \ldots, m$
  - $a_i \in \mathbf{R}^n$ feature vectors; $b_i \in \{-1, 1\}$ Boolean outcomes
- linear predictor: $\hat{b} = \mathrm{sign}(w^T a - v)$
  - $w \in \mathbf{R}^n$ is weight vector; $v \in \mathbf{R}$ is threshold
- SVM: choose w, v to minimize (convex) objective

$$(1/m) \sum_{i=1}^{m} \Big(1 - b_i(w^T a_i - v)\Big)_+ + (\lambda/2)\|w\|_2^2$$

where $\lambda > 0$ is parameter

SVM

$$w^T z - v = 0 \text{ (solid)}; \quad |w^T z - v| = 1 \text{ (dashed)}$$



Sparsity via ℓ_1 regularization

- adding ℓ_1-norm regularization

$$\lambda\|x\|_1 = \lambda(|x_1| + |x_2| + \cdots + |x_n|)$$

  to objective results in **sparse** x

- $\lambda > 0$ controls trade-off of sparsity versus main objective
- **preserves convexity, hence tractability**
- used for many years, in many fields
  - sparse design
  - feature selection in machine learning (lasso, SVM, ...)
  - total variation reconstruction in signal processing
  - compressed sensing

Lasso

- regression problem with ℓ_1 regularization:

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1$$

with $A \in \mathbf{R}^{m \times n}$

- useful even when $n \gg m$ (!!); does **feature selection**

Lasso

- regression problem with ℓ_1 regularization:

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1$$

  with $A \in \mathbf{R}^{m \times n}$
- useful even when $n \gg m$ (!!); does **feature selection**
- *cf.* ℓ_2 regularization ('ridge regression'):

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_2^2$$

Lasso

- regression problem with ℓ_1 regularization:

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1$$

  with $A \in \mathbf{R}^{m \times n}$
- useful even when $n \gg m$ (!!); does **feature selection**
- *cf.* ℓ_2 regularization ('ridge regression'):

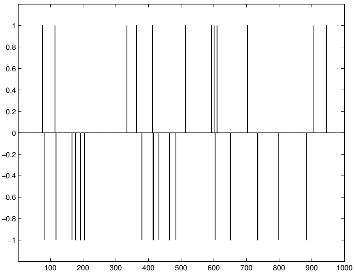
$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_2^2$$

- lasso, ridge regression have **same computational cost**

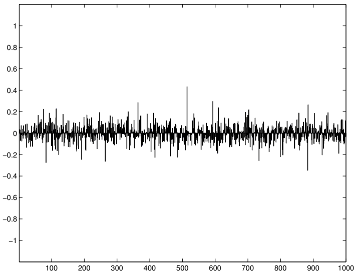
Example

- $m = 200$ examples, $n = 1000$ features
- examples are noisy linear measurements of true x
- true x is sparse (30 nonzeros)

true x

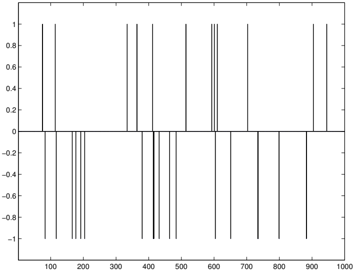


ℓ_2 reconstruction

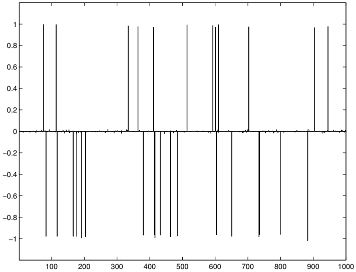


Example

true x



ℓ_1 (lasso) reconstruction



State of the art — Medium scale solvers

- 1000s–10000s variables, constraints
- reliably solved by interior-point methods on single machine
- exploit problem sparsity
- not quite a technology, but getting there

State of the art — Modeling languages

- (new) high level language support for convex optimization
  - describe problem in high level language
  - description is automatically transformed to cone problem
  - solved by standard solver, transformed back to original form

State of the art — Modeling languages

- (new) high level language support for convex optimization
  - describe problem in high level language
  - description is automatically transformed to cone problem
  - solved by standard solver, transformed back to original form

- enables rapid prototyping (for small and medium problems)
- ideal for teaching (can do a lot with short scripts)

CVX

- parser/solver written in Matlab (M. Grant, 2005)
- SVM: minimize

$$(1/m)\sum_{i=1}^{m}\left(1 - b_i(w^T a_i - v)\right)_+ + (\lambda/2)\|w\|_2^2$$

- CVX specification:

```
cvx_begin
    variables w(n) v   % weight, offset
    L=(1/m)*sum(pos(1-b.*(A*w-v)));   % avg. loss
    minimize (L+(lambda/2)*sum_square(w))
cvx_end
```

Outline

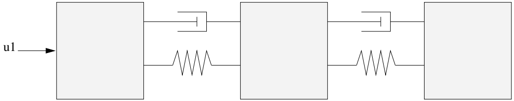
Realt-time embedded optimization

- in many applications, need to solve the same problem repeatedly with different data
  - 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

Realt-time embedded optimization

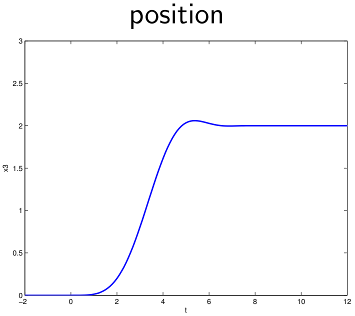
- in many applications, need to solve the same problem repeatedly with different data
  - control: update actions as sensor signals, goals change
  - finance: rebalance portfolio as prices, predictions change
- requires extreme solver reliability, hard real-time execution
- used now when solve times 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**

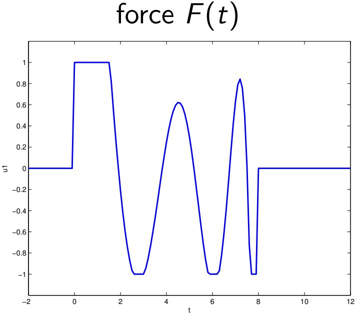
Example — Positioning



- force $F(t)$ moves object, modeled as 3 masses (2 vibration modes)
- goal: move object to commanded position as quickly as possible, with $|F(t)| \leq 1$
- reduces to a (quasi-) convex problem

Optimal force profile





CVXGEN code generator

- handles small, medium size problems transformable to QP (J. Mattingley, 2010)
- accepts high-level problem family description
- uses primal-dual interior-point method
- generates flat library-free C source

CVXGEN code generator

- handles small, medium size problems transformable to QP (J. Mattingley, 2010)
- accepts high-level problem family description
- uses primal-dual interior-point method
- generates flat library-free C source

- typical speed-up over general solver: **100–10000**$\times$

CVXGEN example specification — SVM

```
dimensions
  m = 50   % training examples
  n = 10   % dimensions
end
parameters
  a[i] (n), i = 1..m   % features
  b[i], i = 1..m   % outcomes
  lambda positive
end
variables
  w (n)   % weights
  v     % offset
end
minimize
  (1/m)*sum[i = 1..m](pos(1 - b[i]*(w'*a[i] - v))) +
    (lambda/2)*quad(w)
end
```

CVXGEN sample solve times

problem	SVM	Positioning
variables	61	590
constraints	100	742
CVX, Intel i3	270 ms	2100 ms
CVXGEN, Intel i3	230 μs	4.8 ms

Outline

Large-scale distributed optimization

- *large-scale* optimization problems arise in many applications
  - machine learning/statistics with huge datasets
  - dynamic optimization on large-scale networks
  - image, video processing

Large-scale distributed optimization

- *large-scale* optimization problems arise in many applications
  - machine learning/statistics with huge datasets
  - dynamic optimization on large-scale networks
  - image, video processing
- we'll use *distributed optimization*
  - split variables/constraints/objective terms among a set of *agents/processors/devices*
  - agents coordinate to solve large problem, by passing relatively small messages
  - can target modern large-scale computing platforms
  - long history, going back to 1950s

Consensus optimization

- want to solve problem with N objective terms

$$\text{minimize} \quad \sum_{i=1}^{N} f_i(x)$$

e.g., f_i is the loss function for ith block of training data

- consensus form:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f_i(x_i) \\ \text{subject to} & x_i - z = 0 \end{array}$$

  - x_i are **local variables**
  - z is the **global variable**
  - $x_i - z = 0$ are **consistency** or **consensus** constraints

Consensus optimization via ADMM

with $\overline{x}^k = (1/N)\sum_{i=1}^N x_i^k$ (average over local variables)

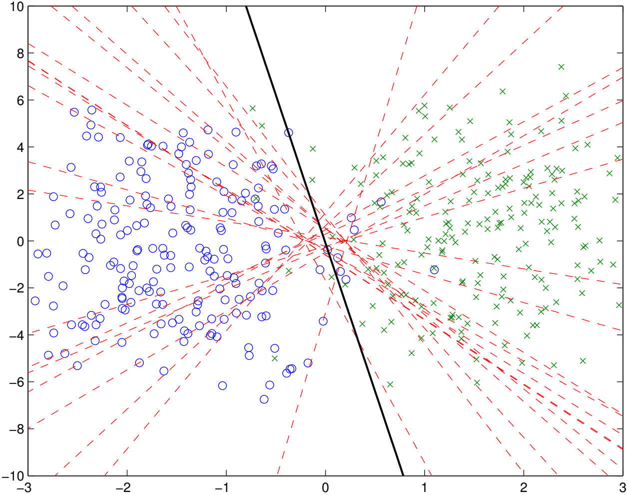
$$\begin{aligned} x_i^{k+1} &:= \operatorname*{argmin}_{x_i} \left(f_i(x_i) + (\rho/2)\|x_i - \overline{x}^k + u_i^k\|_2^2 \right) \\ u_i^{k+1} &:= u_i^k + (x_i^{k+1} - \overline{x}^{k+1}) \end{aligned}$$

- get **global** minimum, under very general conditions
- u^k is running sum of inconsistencies (PI control)
- minimizations carried out independently and in parallel
- coordination is via averaging of local variables x_i

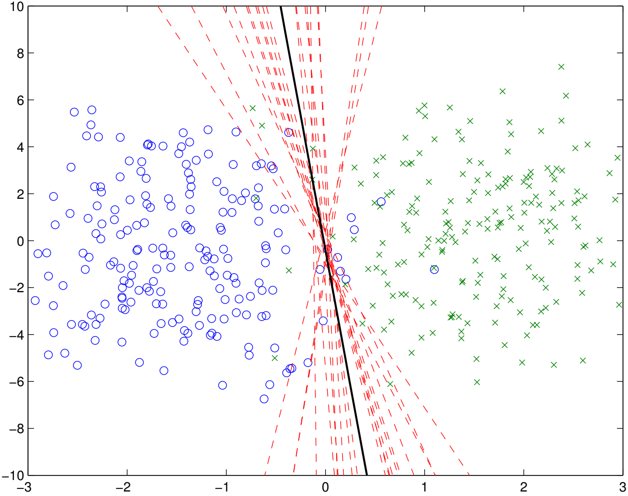
Example — Consensus SVM

- baby problem with $n = 2$, $m = 400$ to illustrate
- examples split into $N = 20$ groups, in worst possible way: each group contains only positive or negative examples

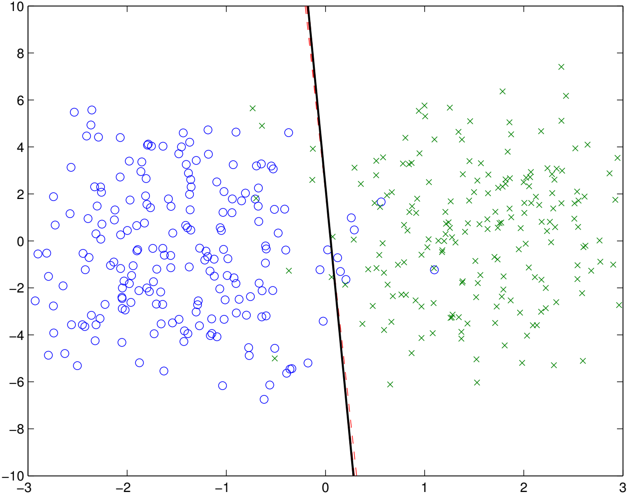
Iteration 1



Iteration 5



Iteration 40



Outline

Summary

- convex optimization problems **arise in many applications**
- convex optimization problems **can be solved effectively**
  - small problems at microsecond/millisecond time scales
  - medium-scale problems using general purpose methods
  - arbitrary-scale problems using distributed optimization
- high level language support (CVX) makes prototyping easy

References

many researchers have worked on the topics covered

- *Convex Optimization* (Boyd & Vandenberghe)
- *CVX: Matlab software for disciplined convex programming* (Grant & Boyd)
- *CVXGEN: A code generator for embedded convex optimization* (Mattingley & Boyd)
- *Distributed optimization and statistical learning via the alternating direction method of multipliers* (Boyd, Parikh, Chu, Peleato, & Eckstein)

all available (with code) from `stanford.edu/~boyd`