Convex Optimization

AJ Friend, Nick Henderson
(w/ material from Stephen Boyd and Steven Diamond)
Stanford University

June 28, 2015

Mathematical optimization

Outline

This talk

- high-level (biased) overview of convex optimization
- (fancy?) examples
- why *restrict* yourself to using convex optimization?

Mathematical optimization

optimization problem has form

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

- $x \in \mathbf{R}^n$ is **decision variable** (to be found)
- f_0 is objective function; f_i are constraint functions
- problem data are hid inside $f_0, \ldots, f_m$
- variations: add equality constraints, maximize a utility function, satisfaction (feasibility), optimal trade off

Convex optimization

convex optimization problem has form

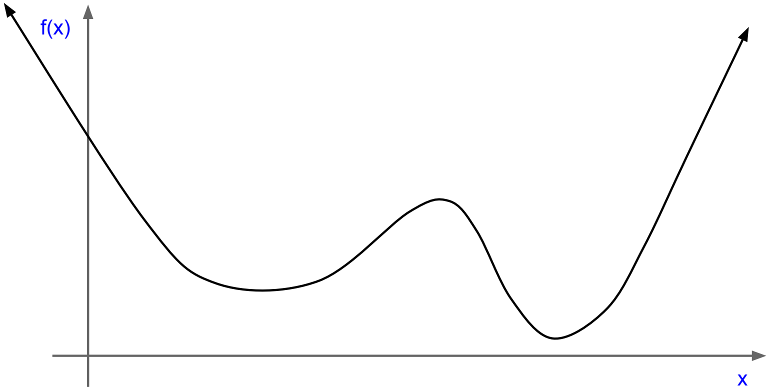
$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \end{array}$$

- $f_0, \ldots, f_m$ **convex**: for $\theta \in [0, 1]$,

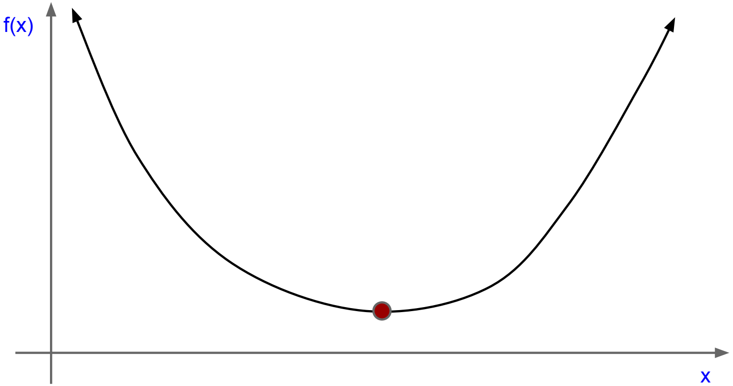
  $$f_i(\theta x + (1-\theta)y) \leq \theta f_i(x) + (1-\theta) f_i(y)$$

  i.e., f_i have nonnegative (upward) curvature

Non-convex function



Convex function



Why convexity?

- local minimizers are global
- most optimization problems often cannot be solved; **convex** problems (usually) can be
- useful theory of convexity
- effective algorithms and available software
  - global solutions
  - polynomial complexity
  - algorithms that scale
- convenient **language** to discuss problems
- unifies many methods; subroutine for non-convex problems
- expressive: **lots of applications**

Applications

- 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

Modeling with convexity

Outline

Approach

- try to formulate your problem as convex
- **build** model guaranteed to be convex from convex atoms and composition rules
- if you succeed, you can (usually) solve it (numerically)
- if non-convex, approximations can work surprisingly well
- an **interface**: say what you want, not how to get it

CVXPY

- we'll use CVXPY (`www.cvxpy.org`), convex modeling and solving tool in Python
- write code very close to the math
- CVXPY and similar tools allow for rapid prototyping
- does the work for you:
  - checks convexity
  - transforms problem (no matrix stuffing by hand!)
  - interface to several solvers

CVXPY example

math: (constrained LASSO)

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2^2 + \rho\|x\|_1 \\ \text{subject to} & \mathbf{1}^T x = 0, \quad \|x\|_\infty \leq 1 \end{array}$$

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

code:

```
from cvxpy import *
x = Variable(n)
obj = sum_squares(A*x-b) + rho*norm(x,1)
constr = [sum_entries(x)==1, norm(x,'inf')<=1]
Problem(obj,constr).solve()
```

Summary

- good trade-off between algorithmic supervision and modeling power
- lots of theoretical, algorithmic, and software tools
- with proper training, "your problem is convex" offers deep, cosmic relief

Image in-painting

Outline

Image in-painting

Original

Corrupted

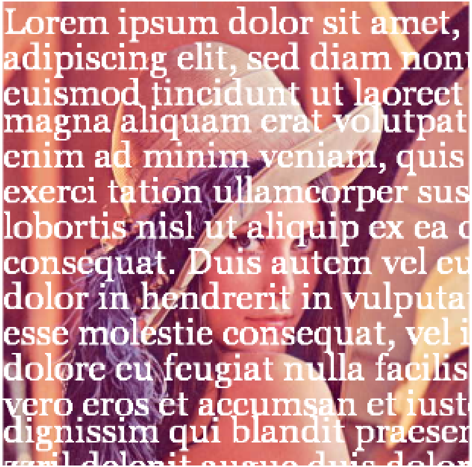


Image in-painting

guess pixel values in obscured/corrupted parts of image

- **decision variable** $x \in \mathbf{R}^{m \times n \times 3}$
- $x_{i,j} \in [0,1]^3$ gives RGB values of pixel (i,j)
- many pixels missing
- known pixel IDs given by set K, values given by **data** $y \in \mathbf{R}^{m \times n \times 3}$

total variation in-painting: choose pixel values $x_{i,j} \in \mathbf{R}^3$ to minimize

$$\mathsf{TV}(x) = \sum_{i,j} \left\| \left[\begin{array}{c} x_{i+1,j} - x_{i,j} \\ x_{i,j+1} - x_{i,j} \end{array} \right] \right\|_2$$

that is, for each pixel, minimize distance to neighbors below and to the right, subject to known pixel values

Convex model

$$\begin{array}{ll} \text{minimize} & \mathrm{TV}(x) \\ \text{subject to} & x_{i,j} = y_{i,j} \text{ if } (i,j) \in K \end{array}$$

- write what you want, not how to get it
- problem concisely and elegantly expressed
  - easily communicated
  - low overhead to tweaking model (rapid prototyping)
- we're done! (well, sort of)
  - express in code
  - invoke convex solver
  - made easier with model-and-solve tool, e.g., `CVXPY`
  - use other solvers/algorithms for speed or scale, if needed

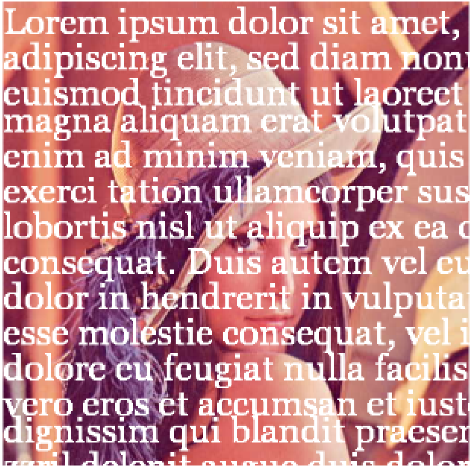
Code example

```
# K[i, j] == 1 if pixel value known, 0 if unknown
from cvxpy import *
variables = []
constr = []
for i in range(3):
    x = Variable(rows, cols)
    variables += [x]
    constr += [mul_elemwise(K, x - y[:,:,i]) == 0]

prob = Problem(Minimize(tv(*variables)), constr)
prob.solve(solver=SCS)
```

Example: 512×512 color image; about 800k variables

Original

Corrupted



Example

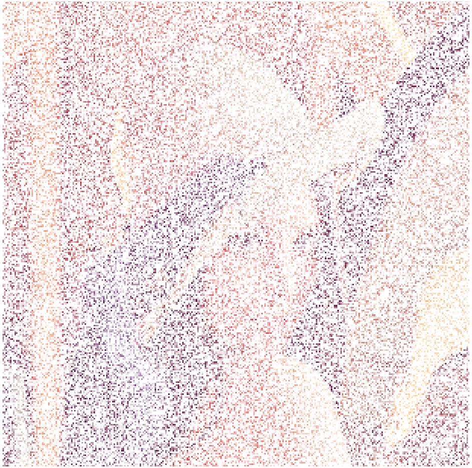
Original

Recovered

Example (80% of pixels removed)

Original

Corrupted



Example (80% of pixels removed)

Original

Recovered

Fault detection

Outline

Mathematical optimization

Modeling with convexity

Image in-painting

Fault detection

Vehicle tracking

Conclusions

Statistical model fitting

- fitting parameters in statistical models is often convex
  - least-squares
  - general maximum-likelihood estimation
- when not convex, approximations can work surprisingly well

Fault detection

- each of n possible faults occurs independently with probability p
- encode as $x_i \in \{0, 1\}$
- m sensors measure system performance
- sensor output is $y = Ax + v = \sum_{i=1}^{n} x_i a_i + v$
- v is Gaussian noise with variance σ^2
- $a_i \in \mathbf{R}^m$ is *fault signature* for fault i
- signal-to-noise ratio

$$\mathrm{SNR} = \frac{\mathbf{E}\,\|Ax\|^2}{\mathbf{E}\,\|v\|^2}$$

- **goal**: guess x (which faults have occurred) given y (sensor measurements)

Maximum likelihood estimation

- choose $x \in \{0,1\}^n$ to minimize negative log likelihood function

$$\ell(x) = \frac{1}{2\sigma^2}\|Ax - y\|_2^2 + \log(1/p - 1)\mathbf{1}^T x + c,$$

- nonconvex, NP-hard
- instead solve convex (relaxed) problem

$$\begin{array}{ll} \text{minimize} & \|Ax - y\|_2^2 + 2\sigma^2 \log(1/p - 1)\mathbf{1}^T x \\ \text{subject to} & 0 \le x_i \le 1, \quad i = 1, \ldots n \end{array}$$

  and round
- called **relaxed ML** estimate

Relaxed ML CVXPY code

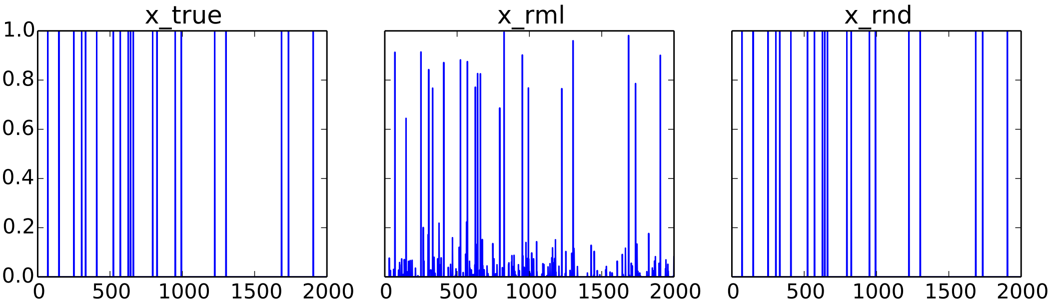
$$\begin{array}{ll} \text{minimize} & \|Ax - y\|_2^2 + 2\sigma^2 \log(1/p - 1)\mathbf{1}^T x \\ \text{subject to} & 0 \le x_i \le 1, \quad i = 1, \ldots n \end{array}$$

```
from cvxpy import *
x = Variable(n)
tau = 2*log(1/p - 1)*sigma**2
obj = Minimize(sum_squares(A*x-y) + tau*sum_entries(x))
constr = [0<=x, x<=1]
Problem(obj,constr).solve()

x_rml = np.array(x.value).flatten()
x_rnd = (x_rml>=.5).astype(int)
```

Example

$n = 2000$ possible faults, $m = 200$ measurements

$p = 0.01$, SNR $= 5$

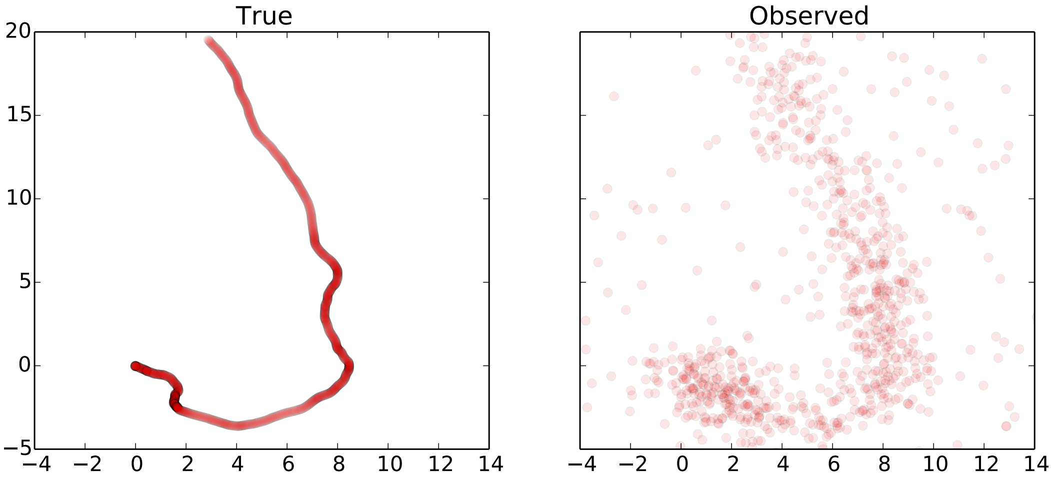


- perfect fault recovery!

Vehicle tracking

Outline

Vehicle tracking



Kalman filtering

- estimate vehicle path from noisy position measurements (with outliers)
- dynamic model of vehicle state x_t:

$$x_{t+1} = Ax_t + Bw_t, \quad y_t = Cx_t + v_t$$

  - x_t is vehicle state (position, velocity)
  - w_t is unknown drive force on vehicle
  - y_t is position measurement; v_t is noise
- Kalman filter: estimate x_t by minimizing $\sum_t \left(\|w_t\|_2^2 + \gamma\|v_t\|_2^2\right)$
- a least-squares problem; assumes w_t, v_t Gaussian

Robust Kalman filter

- to handle outliers in v_t, replace square cost with Huber cost
- **robust** Kalman filter:

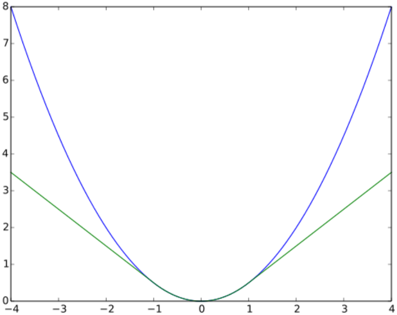
$$\begin{array}{ll} \text{minimize} & \sum_t \left(\|w_t\|_2^2 + \gamma\phi(v_t)\right) \\ \text{subject to} & x_{t+1} = Ax_t + Bw_t, \quad y_t = Cx_t + v_t \end{array}$$

where ϕ is Huber function

$$\phi(a) = \begin{cases} \|a\|_2^2 & \|a\|_2 \leq 1 \\ 2\|a\| - 1 & \|a\|_2 > 1 \end{cases}$$

- a convex problem

Huber loss function



- outside $[-1, 1]$ interval, penalizes linearly, not quadratically
- large errors more easily "forgiven", allowing fit to discount outliers

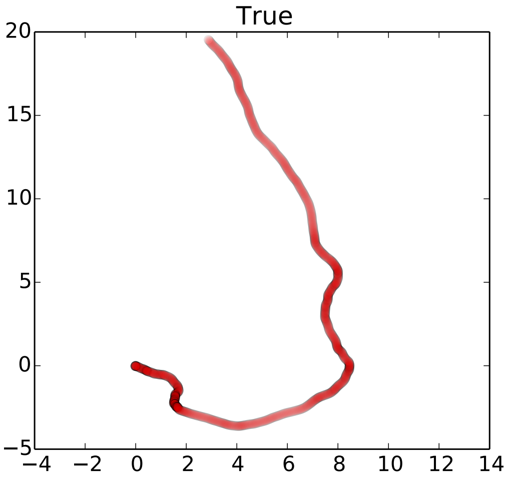
Robust KF CVXPY code

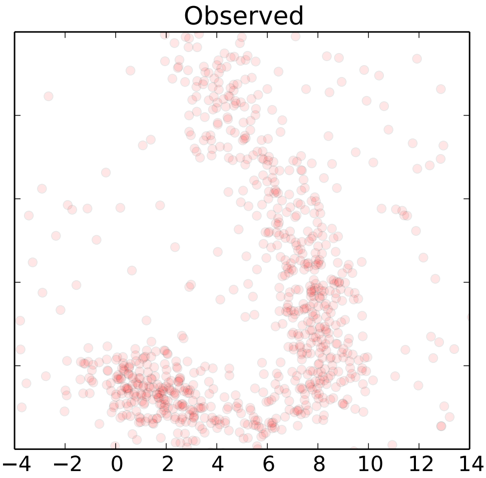
```
from cvxpy import *
x = Variable(4,n+1)
w = Variable(2,n)
v = Variable(2,n)

obj = sum_squares(w)
obj += sum(huber(norm(v[:,t])) for t in range(n))
obj = Minimize(obj)
constr = []
for t in range(n):
    constr += [ x[:,t+1] == A*x[:,t] + B*w[:,t] ,
                y[:,t]   == C*x[:,t] + v[:,t]   ]

Problem(obj, constr).solve()
```

Example

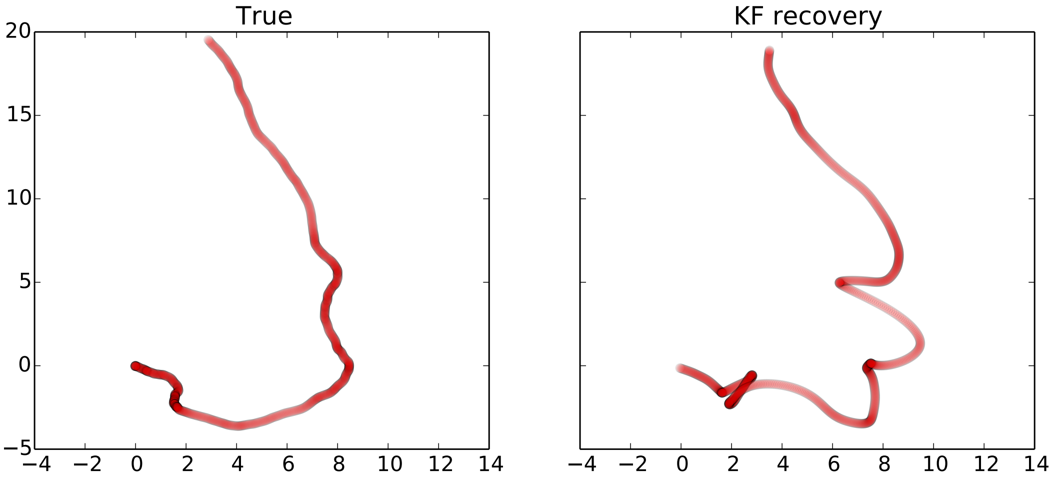
- 1000 time steps
- w_t standard Gaussian
- v_t standard Gaussian, except 30% are outliers with $\sigma = 10$

Example

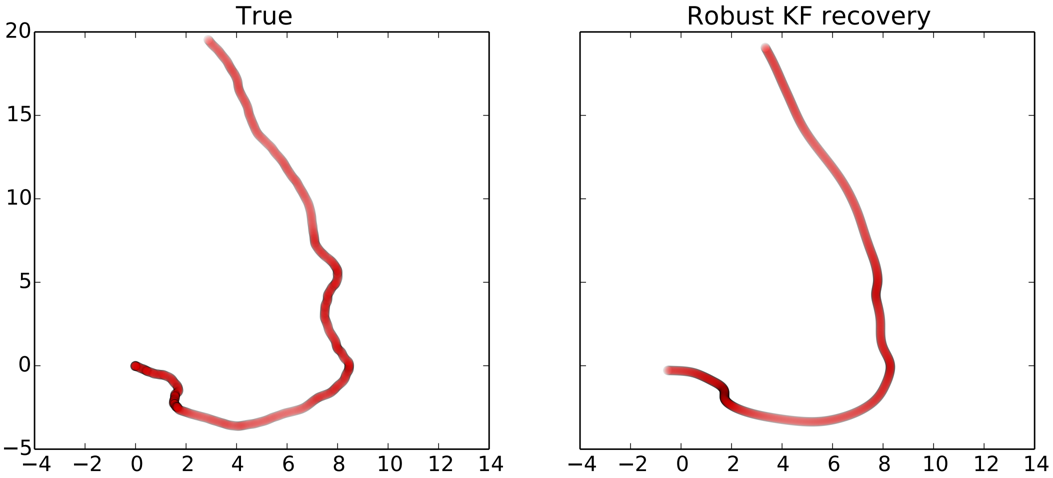




Example



Example



Conclusions

Outline

Conclusions

- convex optimization problems arise in many applications
- can be solved effectively
- high level languages (CVX, CVXPY) make prototyping easy

but lots we couldn't cover in this presentation:

- theory of convexity
- duality/Lagrange multipliers
- algorithms
  - (stochastic) gradient descent
  - interior point algorithms
  - optimal first order algorithms
  - problem splitting for large scale and distributed optimization

References

- *Convex Optimization* (Boyd & Vandenberghe)
- *CVX: Matlab software for disciplined convex programming* (Grant & Boyd)
- *CVXPY* (`www.cvxpy.org`)