Convex Optimization

Stephen Boyd Steven Diamond AJ Friend
Stanford University

XDATA PI Meeting, July 2014

Outline

Convex optimization

Image in-painting

Fault detection

Robust Kalman filtering

Summary

(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

The good news

everything is an optimization problem

- *choose parameters* in model to fit data
  (minimize misfit or error on observed data)
- *optimize actions* (minimize cost or maximize profit)
- *allocate resources* over time
  (minimize cost, power; maximize utility)
- *engineering design*
  (trade off weight, power, speed, performance, lifetime)

The bad news

you can't solve most optimization problems

- generally NP-hard
- heuristics often require tuning, luck, babysitting

Some (limited) good news

we can solve **convex optimization problems**
(reliably, using algorithms that scale)

- objective and constraint functions $f_0, \ldots, f_m$ must have **nonnegative curvature**
- not so easy to detect unless you're trained
- new DSLs (domain-specific languages) for convex optimization make it fairly easy to use

CVXPY example

math: (constrained LASSO)

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

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

---

code:

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

Using optimization

- say what you want, not how to get it
- if what you want is not convex, then approximate it as convex
  (even terrible approximations can yield excellent results)
- if you care about something, put it in the objective or constraints
- tweak objective/constraints, not policy/actions/parameter values

Convex optimization applications

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

Outline

Image in-painting

- guess pixel values in obscured/corrupted parts of image
- *total variation in-painting*: choose pixel values $x_{i,j} \in \mathbf{R}^3$ to minimize
$$\mathrm{TV}(x) = \sum_{i,j} \left\| \begin{bmatrix} x_{i+1,j} - x_{i,j} \\ x_{i,j+1} - x_{i,j} \end{bmatrix} \right\|_2$$
- a convex problem

Example

- 512×512 color image
- denote corrupted pixels with $K \in \{0, 1\}^{512 \times 512}$
  - $K_{ij} = 1$ if pixel value is known
  - $K_{ij} = 0$ if unknown
- $X_{\rm corr} \in \mathbf{R}^{512 \times 512 \times 3}$ is corrupted image
- 60 seconds to solve with CVXPY and SCS on a laptop

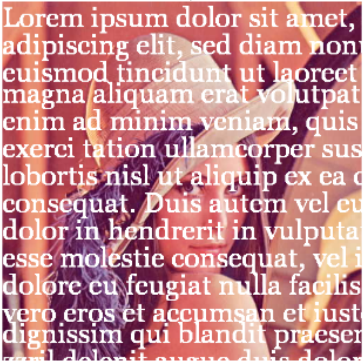
Image in-painting CVXPY code

```
from cvxpy import *
variables = []
constr = []
for i in range(3):
    X = Variable(rows, cols)
    variables += [X]
    constr += [mul_elemwise(K, X - X_corr[:,:,i]) == 0]

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

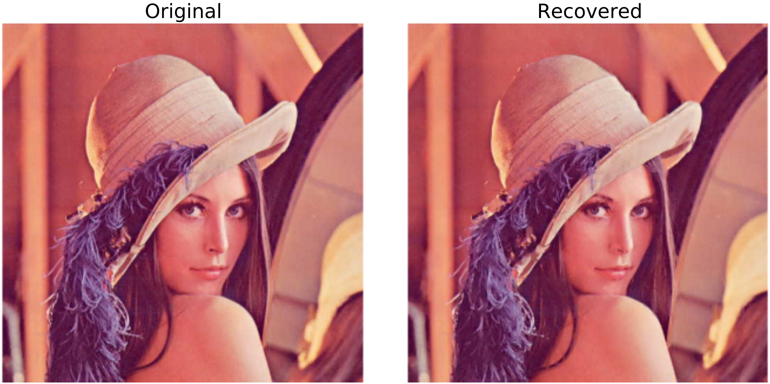
Example

Original

Corrupted



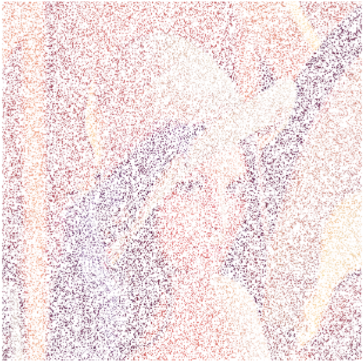
Example



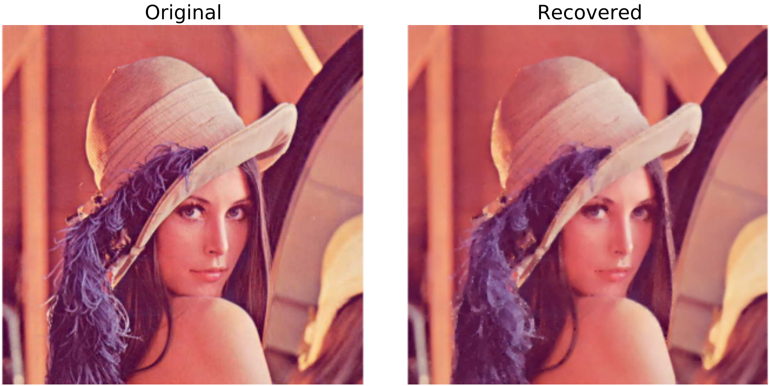
Example (80% of pixels removed)

Original

Corrupted



Example (80% of pixels removed)



Outline

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
- **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

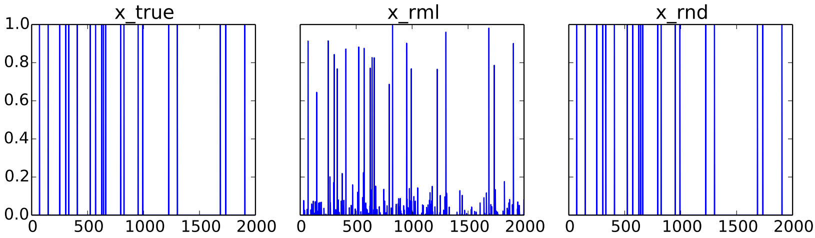
- called **relaxed ML** estimate

Relaxed ML CVXPY code

```
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
- 4 seconds to solve with CVXPY and ECOS on a laptop

Outline

Kalman filter

- estimate vehicle track from noisy position measurements
- 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

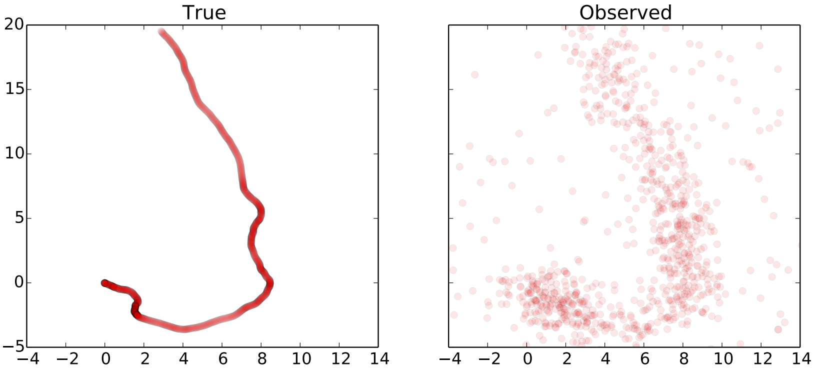
Robust KF CVXPY code

```
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()
```

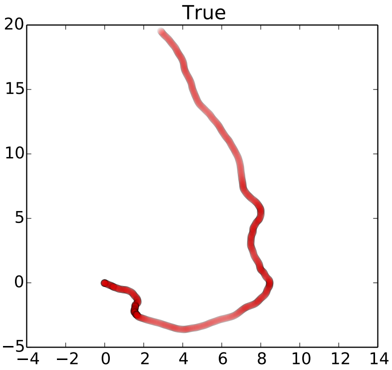
Example

- 1000 time steps
- w_t standard Gaussian
- v_t standard Gaussian, except 30% are outliers with $\sigma = 10$
- 30 seconds to solve with CVXPY and ECOS on a laptop

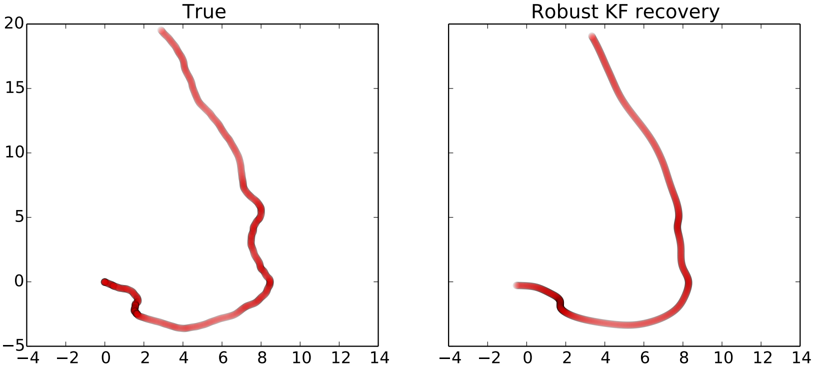
Example



Example



Example


True
Robust KF recovery
20
15
10
5
0
−5
−4 −2 0 2 4 6 8 10 12 14
−4 −2 0 2 4 6 8 10 12 14

Outline

Summary

- you can use convex optimization to solve cool problems
- with new DSLs and solvers it's easy to rapidly prototype convex optimization methods
- and it will scale
- not all parts are in place, but we're getting there