Convex Optimization

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

June 28, 2015

Mathematical optimization

Outline

(mathy math)

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

convexity definition

- sets
- functions
- in constraints: level sets are convex
- in objective: local minimizers are global
- picture?

modeling with convexity

- sets of convex function and set *atoms*
- convexity preserving operations
  - addition, composition, partial minimization, . . .
- transformation of seemingly non-convex problems into convex
- convex approximation
- using DCP (disciplined convex programming)
  - code is very code to the math
  - convenient for humans
  - does the problem transformations for you
  - calls the solver
  - transforms back into your variables

Why convexity?

- trade off between supervision and modeling power
- nice theory
- theoretical guarantees, global optimum, interpretability: (of local minimizers)
- efficient algorithms give global solutions in polynomial time
- common language, conceptual unification
- useful subroutine for non-convex optimization (local convex approximation)
- (leaky) abstraction: once you've modeled as a convex problem, consider it solved
- with proper training, "your problem is convex" offers deep, cosmic relief
- almost a technology, (like least squares)
- prototype quickly with generic solvers
- use other methods for speed and scale if necessary

applications

- inversion
- engineering design
- optimal control
- model fitting

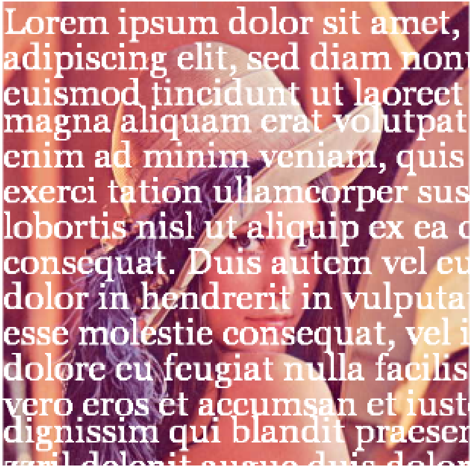
Image in-painting

Outline

Image in-painting

Original

Corrupted



Example

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

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

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

Convex model

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

- 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 different solvers/algorithms for speed or scale, if needed

code example

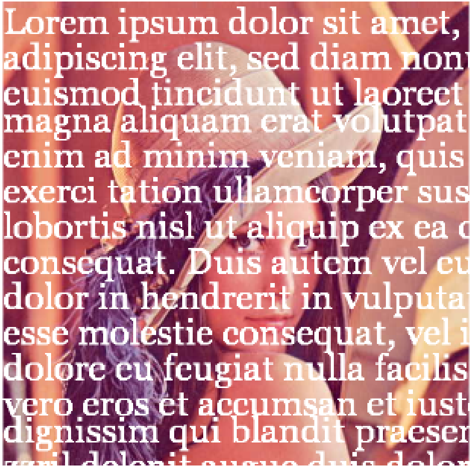
```
# K[i, j] == 1 if pixel value known, 0 if unknown
from cvxpy import *
variables = []
constr = []
for i in range(3):
    x = Variable(rows, cols)
    variables += [x]
    constr += [mul_elemwise(K, x - y[:,:,i]) == 0]

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

Example

Original

Corrupted



Example

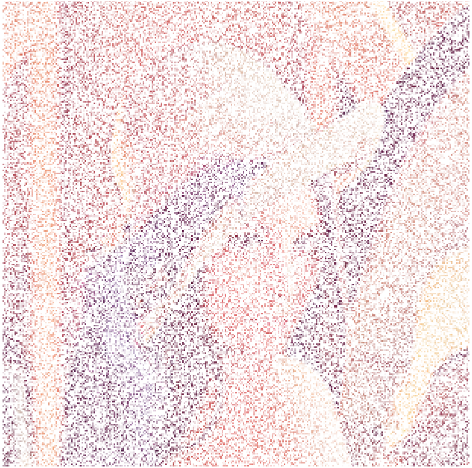
Original

Recovered

Example (80% of pixels removed)

Original

Corrupted



Example (80% of pixels removed)

Original

Recovered

What we left out

Outline

References

Outline

- Convex optimization
- CVX
- CVXPY