# Sparco: A testing framework for sparse reconstruction

Ewout van den Berg
Department of Computer Science, UBC

with
M.P. Friedlander, G. Hennenfent, F.J. Herrmann, R. Saab, and Ö. Yılmaz

February 18, 2008

This presentation was carried out as part of the SINBAD project with financial support, secured through ITF, from the following organizations: BG, BP, Chevron, ExxonMobil, and Shell. SINBAD is part of the collaborative research & development (CRD) grant number 334810-05 funded by the Natural Science and Engineering Research Council (NSERC).

# Background

- Central theme: sparse recovery
- Numerous solvers
  - Linear Programming, Iterative Soft Thresholding
  - PDCO, GPSR, L1LS, SPGL1, ...
- Ad hoc problem evaluation
  - Problems often easy to solve
  - Comparison between results is difficult
  - Lacks objectivity
- Standard set of test problems to provide a common ground
- Sparco (after common denominator: sparse coefficients)

# Sparco

- Software toolbox written in Matlab
- Prototyping compressed sensing scenarios
- Provides small/medium scale test problems
- Not intended as replacement for SlimPy

# Design criteria

- General problem format
  - Independent of solver
  - Independent of formulation (BP, BPDN, QP, TV)
- Easy to use and understand
  - Getting more people to use it
  - Simplifies comparing results
- Easy to extend
  - Community can contribute problems
  - We don't have to do all the work ourselves
- Easy to maintain
  - Self documenting (webpage is automatically generated)
  - Minimize dependencies between files

# Outline

1. Problem formulation
2. Operators
3. Sparco internals
4. Application

# Problem Formulation

- All problems are given in general form:

$$b = Ax_0 + n$$

- $x_0$ is a sparse coefficient vector
- $n$ is additive noise
- $b$ is observed signal
- $A = MB$
  - $M$ measurement matrix
  - $B$ sparsity 'basis'

# Problem Formulation: Example

- Example from MRI (Shepp-Logan phantom)
  - M: restricted Fourier,
  - B: wavelet transform

- Independent of sparsity formulation:

- Basis pursuit denoise: $\underset{x}{\text{minimize}} \ \|x\|_1 \ \text{ s.t. } \ \|Ax - b\|_2 \le \sigma$
- Total variation: $\underset{x}{\text{minimize}} \ \|x\|_1 + \gamma \mathrm{TV}(Bx) + \lambda \|Ax - b\|_2$

# OPERATORS

# Overview

- Operators provide routines for
  - Multiplication; $v = Ax$, $w = A^T y$ `v=A(x,1), w=A(y,2)`
  - Querying its properties `info = A([],0)`
  - Conversion to class object `C = opClass(A); C*x, C'*y`
- Most operators are implemented based on fast routines
- Different types of operators
  - Basic operators
  - Meta operators
  - Other operators

# Basic operators

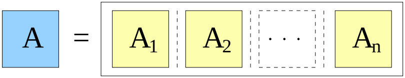
- Identity `A = opDirac(...);`
- General matrix `A = opMatrix(...);`
- Diagonal matrix `A = opDiag(...);`

- DCT/FFT `A = opDCT(...), opFFT(...);`
- Curvelet `A = opCurvelet2d(...);`
- Wavelet `A = opWavelet(...);`
- Hadamard `A = opHadamard(...);`
- Heaviside `A = opHeaviside(...);`

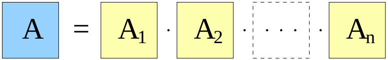
- Gaussian ensemble `A = opGaussian(...);`
- Sign ensemble `A = opSign(...);`
- Binary ensemble `A = opBinary(...);`

# Meta operators

- Transpose `AT = opTranspose(A);`
- Dictionary `A = opDictionary(A1,A2,...,An);`

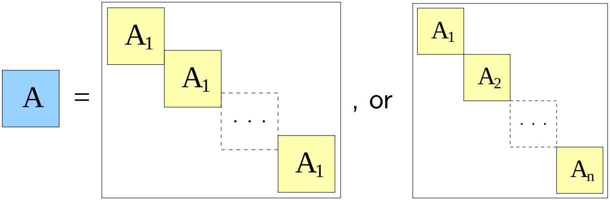



- Compound functions `A = opFoG(A1,A2,...,An);`

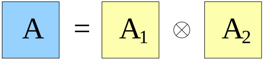


# Meta operators *(continued)*

- Block diagonal `A = opBlockDiag(A1,A2,...,An)`
- Windowed with overlap `A = opWindowedOp(...)`




- Kronecker `A = opKron(A1,A2);`

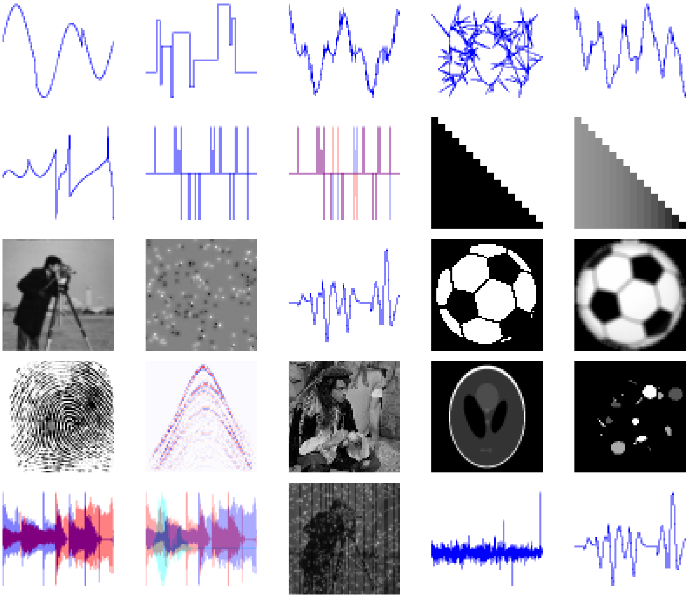



— Demo time —

# Other operators

- Restriction `A = opRestriction(...)`
- Binary mask `A = opMask(...)`
- Column restriction `A = opColumnRestrict(...)`
- 1D Convolution `A = opConvolve1d(...)`
- Non-linear operators
  - Split complex `A = opSplitComplex(...)`
  - Complex to real `A = opReal(...)`

# SPARCO INTERNALS



# Sparco problems

- Currently 27 test problems implemented
- Collected from dozens of papers
- First comprehensive set of test problems

# Instantiating problems

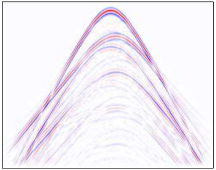
- Problems are identified by their unique ID or name
- `P = generateProblem(5)`, or equivalently
  `P = generateProblem('gcosspike')`.
- `l = generateProblem('list')`

# Problem structure

| P. | **Description** | |
|---|---|---|
| `A` | Operator $A = MB$ | |
| `B` | Sparsity 'basis' $B$ | |
| `M` | Measurement operator $M$ | |
| `x0` | Desired solution | (optional) |
| `b` | Observed signal | |
| `op` | Operators used to construct $M$ and $B$ | |
| `info` | Documentation | |
| ............. | ...................................... | |
| `reconstruct` | Routine to reconstruct signal/image from $x$ | |
| `signal` | Original signal | (optional) |
| `signalSize` | Size of desired signal | (optional) |

# Illustration

Problem 901 – Recovery of missing traces



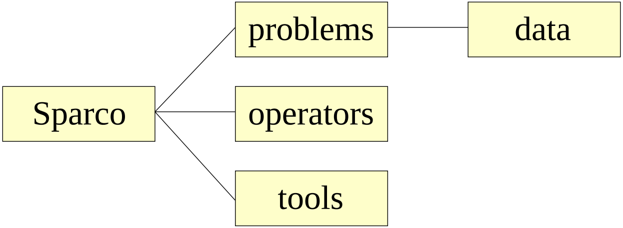
- `P.B` - Curvelet adjoint
- `P.M` - Column restriction operator
- `P.signal` - All traces
- `P.b` - Column restriction of traces (vectorized)

- `[x,r,g,info] = spgl1(P.A,P.b,0,0); % Solve BP`

- `result = P.reconstruct(x); % Resize(B * x)`

— Demo time —

# Directory structure




- Problems are identified by name `prob###.m`
- Adding a file adds a problem
- Operators need additions for documentation only

# Applications of Sparco

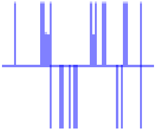
- Testing robustness of solver
- Comparing performance between solvers
- Rapid prototyping

# Comparing performance

Table: *Basis pursuit comparisons*

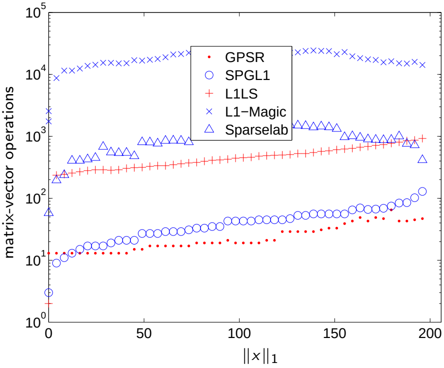
| Problem | PDCO $\|r\|_2$ | PDCO $\|x\|_1$ | Homotopy $\|r\|_2$ | Homotopy $\|x\|_1$ | SPGL1 $\|r\|_2$ | SPGL1 $\|x\|_1$ |
|---|---|---|---|---|---|---|
| blocksig | 3.3e−04 | 4.5e+02 | 1.0e−04 | 4.5e+02 | 2.0e−14 | 4.5e+02 |
| blurrycam | *t* | *t* | *t* | *t* | 9.9e−05 | 1.0e+04 |
| blurspike | 9.1e−03 | 3.4e+02 | *t* | *t* | 9.9e−05 | 3.5e+02 |
| cosspike | 1.6e−04 | 2.2e+02 | 1.0e−04 | 2.2e+02 | 8.6e−05 | 2.2e+02 |
| finger | *t* | *t* | *t* | *t* | 8.2e−05 | 5.5e+03 |
| gcosspike | 1.9e−05 | 1.8e+02 | 1.0e−04 | 1.8e+02 | 9.9e−05 | 1.8e+02 |
| jitter | 1.3e−05 | 1.8e+00 | 1.0e−04 | 1.7e+00 | 5.3e−05 | 1.7e+00 |
| p3poly | 4.6e−02 | 1.7e+03 | 8.5e+01[f] | 1.8e+03 | 9.5e−05 | 1.7e+03 |
| **seismic** | *t* | *t* | *t* | *t* | **8.6e−05** | **3.9e+03** |
| sgnspike | 9.3e−06 | 2.0e+01 | 1.0e−04 | 2.0e+01 | 8.0e−05 | 2.0e+01 |
| soccer1 | *t* | *t* | *t* | *t* | 1.0e−04 | 4.2e+02 |
| spiketrn | 3.6e−03 | 1.3e+01 | 1.0e−04 | 1.3e+01 | 9.9e−05 | 1.3e+01 |
| srcsep1 | 8.2e−03 | 1.1e+03 | *t* | *t* | 8.6e−05 | 1.1e+03 |
| srcsep2 | 5.5e−03 | 1.1e+03 | *t* | *t* | 1.0e−04 | 1.1e+03 |
| yinyang | 1.4e−03 | 2.6e+02 | 2.8e−03[f] | 4.7e+02 | 9.6e−05 | 2.6e+02 |

# Gaussian ensemble, spikes signal

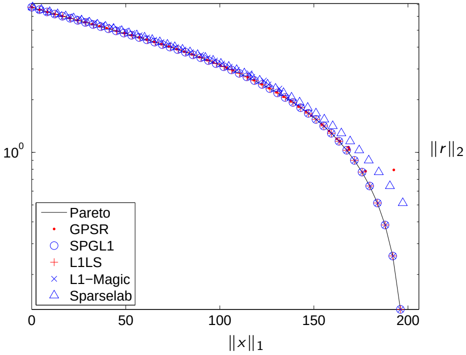
$A =$ Gaussian $\quad x_0 =$ 

$A$: 1200×5120
Candès, Romberg, Tao '05

## Matrix-vector products




## Pareto curve



# Rapid prototyping – Example 1

- Given existing problem with complex $A$, $b$
- People have claimed that complex

$$\underset{z \in \mathbb{C}^n}{\text{minimize}} \|z\|_1 \quad \text{subject to} \quad Az = b$$

can be solved with

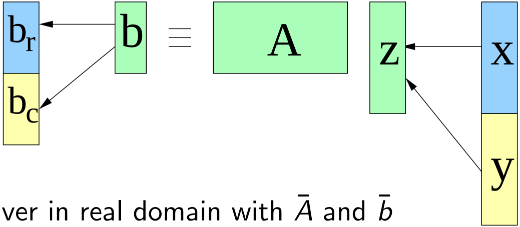
$$\underset{x,y \in \mathbb{R}^n}{\text{minimize}} \|x\|_1 + \|y\|_1 \quad \text{subject to} \quad A(x + iy) = b$$

- Let's find out!

# Rapid prototyping – Example 1

- `opSplitComplex(...);`
- Reformulate operator $A$ and vector $b$

```
S1   = opSplitComplex(m);
S2   = opTranspose(opSplitComplex(n));
Abar = opFoG(S1,A,S2);
bbar = S1(b,1);
```




- Run solver in real domain with $\bar{A}$ and $\bar{b}$

— Demo time —

# Rapid prototyping – Example 2

- Signal sparse in DCT-Dirac dictionary
- Measure with Gaussian ensemble

```
B = opDictionary(opDCT(256),opDirac(256));
M = opGaussian(192,256);
A = opFoG(M,B)
x = sparse(512,32);
n = randn(192,1);
b = A(x,1) + n;
```

- Recover coefficients by Basis Pursuit Denoise

```
xs = spgl1(A,b,0,norm(n,2));
```

http://www.cs.ubc.ca/labs/scl/sparco/