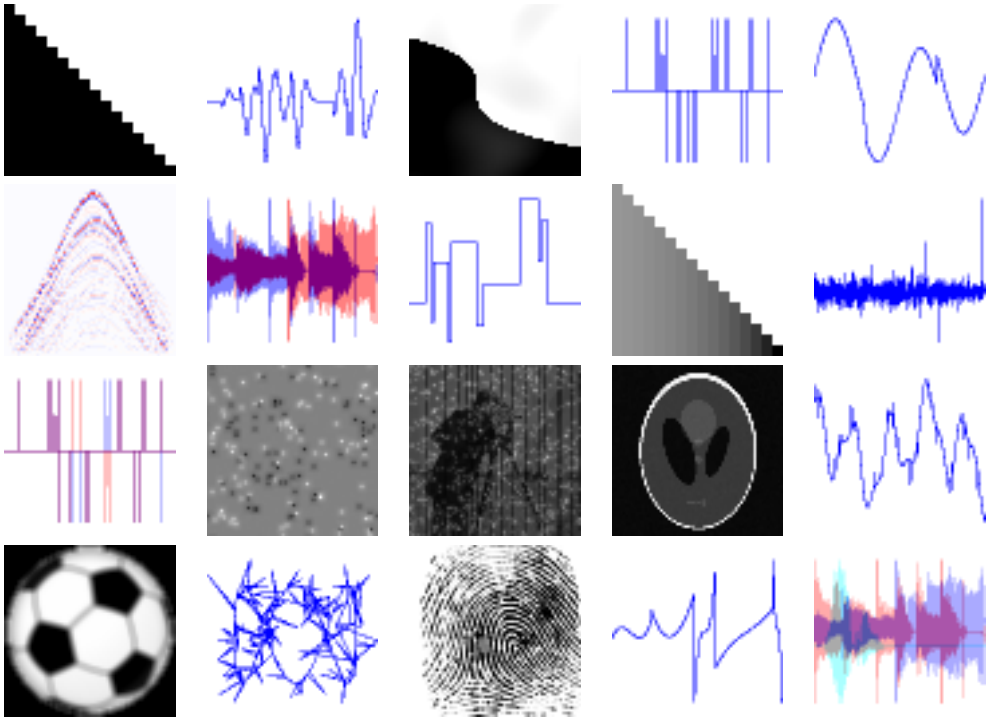
Sparco: A Testing Framework for Sparse Reconstruction

E. van den Berg, M. P. Friedlander, G. Hennenfent,
F. J. Herrmann, R. Saab, Ö. Yılmaz

University of British Columbia

October 29, 2007



Contents

Department of Computer Science Technical Report TR-2007-20
October 2007, University of British Columbia

Sparco: A Testing Framework for Sparse Reconstruction*

Ewout van den Berg Michael P. Friedlander Gilles Hennenfent
Felix J. Herrmann Rayan Saab Özgür Yılmaz

October 29, 2007

Abstract

Sparco is a framework for testing and benchmarking algorithms for sparse reconstruction. It includes a large collection of sparse reconstruction problems drawn from the imaging, compressed sensing, and geophysics literature. Sparco is also a framework for implementing new test problems and can be used as a tool for reproducible research. Sparco is implemented entirely in MATLAB, and is released as open-source software under the GNU Public License.

1 Introduction

Sparco is a suite of problems for testing and benchmarking algorithms for sparse signal reconstruction. It is also an environment for creating new test problems; a large library of standard operators is provided from which new test problems can be assembled.

Recent theoretical breakthroughs on the recovery of sparse signals from incomplete information (see, e.g., [8, 11, 16, 21]) has led to a flurry of activity in developing algorithms for solving the underlying recovery problem. Sparco originated from the need to extensively test the sparse reconstruction software package SPGL1 [2]. Because the test problems and the infrastructure needed to generate them are useful in their own right, these were gathered into a common framework that could be used to test and benchmark related algorithms.

Collections of test problem sets, such as those provided by Matrix Market [3], COPS [14], and CUTEr [25], are routinely used in the numerical linear algebra and optimization communities as a means of testing and benchmarking algorithms. These collections are invaluable because they provide a common and easily identifiable reference point. A situation in which each research group shares or contributes to a common set of test problems leads to a more transparent research process. Our aim is to provide a software package that can be used as one of the tools of reproducible research; we follow the example set by SparseLab [20].

Sparco is implemented entirely in MATLAB. It is built around a comprehensive library of linear operators that implement the main functions that are used in signal and image processing. These operators can be assembled to define new test problems. The toolbox is self contained, and external packages are only optional.

*All authors are at the University of British Columbia, Vancouver, Canada (ewout78@cs.ubc.ca, mpf@cs.ubc.ca, ghennenfent@eos.ubc.ca, fherrmann@eos.ubc.ca, rayans@ece.ubc.ca, oyilmaz@math.ubc.ca). Research supported by Collaborative Research and Development grant number 334810-05 from the Natural Sciences and Engineering Research Council of Canada as part of the DNOISE Project. (v.644)

The outline of this document is as follows. Section 2 gives background on the sparse signal recovery problem. Section 3 describes the problem instantiation mechanism of Sparco. The operators provided by Sparco and an example on how to prototype an application using these operators is given in §§4–5. Details of the operator and test-problem implementations are given in §6. Appendix A gives brief instructions on installing the software package.

2 Sparse signal recovery

A fundamental feature of digital technology is that *analog* signals—such as images and audio—are stored and processed *digitally*. This dichotomy makes it challenging to design signal processing techniques that can find efficient digital representations of analog signals. The ubiquity of digital technology makes this a vital problem of practical interest. In the classical approach, the first step is to sample the analog signal by collecting measurements. The required resolution determines the number of needed measurements, which can be huge. The second step is source coding, i.e., the quantization of the measured values followed by compression of the resulting bit-stream. Frequently, transform coding methods are employed to compress quantized measurements. These methods reduce the dimensionality of the signal by decomposing it as a linear combination of a small number of fundamental building blocks. Mathematically, the aim is to find a nice basis for the finite—but high dimensional—space in which the signal has a sparse representation, i.e., has only a few basis coefficients that are relatively large in magnitude. This sparsity is then exploited for various purposes, including compression.

The classical two-stage approach—first sample densely and then sparsify and compress—is the basis for several applications that we use in our daily lives. The JPEG image format, for example, uses the discrete cosine basis to successfully sparsify, and thus compress, natural images. Although this approach has so far proved successful in signal processing technology, there are certain restrictions it imposes. We briefly describe these and discuss how they lead to the so-called *sparse recovery problem*.

Source coding and redundant dictionaries In various settings, one can obtain much sparser representations by expanding the underlying signals with respect to a *redundant dictionary* rather than a basis. Such dictionaries can be obtained, for example, as unions of different bases—e.g., wavelet, discrete cosine transform (DCT), and Fourier—for the underlying space. In this case, signals no longer have unique representations, and finding the sparsest representation is non-trivial. In order for transform coding methods based on redundant dictionaries to be a practical alternative, one needs computationally tractable ways of obtaining sparse expansions with respect to redundant dictionaries.

Number of measurements vs. quality of the reconstruction As noted above, the number of measurements required to achieve a target resolution can be very large. A 1024-by-1024 image, for example, needs 2^{20} pixel values, which, as a raw image, would correspond to a file size in the order of megabytes. In (exploration) seismology, the situation is even more extreme since datasets are measured in terabytes.

In the compression stage, however, most of the collected data is discarded. Indeed, a typical JPEG file that stores a 1024-by-1024 image has a size in the order of hundreds of kilobytes, and the conversion to JPEG discards up to 90% of the collected data. The fact that we need to collect a huge amount of data, only to throw it away in the next stage, seems rather inefficient. Furthermore, in many settings, e.g., seismic and magnetic resonance

imaging, making a large number of measurements is very expensive and physically challenging, if not impossible, thus limiting the maximum achievable resolution by the corresponding digital representation. All these concerns lead to the question of whether one can possibly reconstruct the original signal from fewer than the "required" number of measurements, without compromising the quality of the digitized signal. Note that this would correspond to combining the sampling and compression stages into a single *compressed sensing* [15] or *compressive sampling* [4] stage.

2.1 The sparse recovery problem

The core of the sparse recovery problem can be described as follows. Consider the linear system

$$b = Ax + r, \tag{1}$$

where A is an m-by-n matrix, r is an unknown m-vector of additive noise, and the m-vector b is the set of observed measurements. The goal is to find an appropriate x that is a solution of (1). Typically, $m < n$, and this set of equations is underdetermined and has infinitely many solutions. It is well known how to find the solution of (1) that has the smallest 2-norm, provided A is full rank (e.g., see [12]). However, the aim in the sparse recovery problem is to find a (possibly approximate) solution that is in some sense sparse, so that either x or the gradient of the resulting image has few nonzero elements.

Some fundamental questions regarding the sparse recovery problem are: (i) Is there a *unique sparsest solution* of (1), particularly when $r = 0$? Under what conditions does this hold? (ii) Can one find a sparse solution in a computationally tractable way that is stable in the presence of noise, i.e., when $r \neq 0$? (iii) Is there a fast algorithm that gives a solution that is guaranteed to be sparse in some sense? (iv) How should one choose the matrix A so that the answers to (i)–(iii) are affirmative?

Next, we describe different settings that lead to the sparse recovery problem, and give a brief overview of the relevant literature and what is known about these questions.

2.2 Redundant representations and basis pursuit

As discussed above, one of the important goals in signal processing is obtaining sparse decompositions of a signal with respect to a given redundant dictionary. This problem was addressed in [11]. Using the notation in (1), the aim is to express a signal b as a linear combination of a small number of columns (atoms) from the redundant dictionary consisting of the columns of A, i.e., one aims to find a sparse vector x which satisfies (1). Note that, ideally, the solution of this problem can be found by solving the optimization problem

$$\underset{x}{\text{minimize}} \quad \|x\|_0 \quad \text{subject to} \quad \|Ax - b\|_2 \leq \sigma, \tag{2}$$

where the function $\|x\|_0$ counts the number of nonzero components of the vector x, and the parameter $\sigma \geq 0$ prescribes the desired fit in the set of equations $Ax \approx b$. This problem is combinatorial and NP hard [40]. Basis pursuit denoise [11] aims to find a sparse solution of (1) by replacing the discrete optimization problem (2) with the convex optimization problem

$$\underset{x}{\text{minimize}} \quad \|x\|_1 \quad \text{subject to} \quad \|Ax - b\|_2 \leq \sigma. \tag{3}$$

Methods for solving this (and closely related problems) include interior-point algorithms [6, 11, 34], gradient projection [2, 24], and iterative soft thresholding [13, 27]. The homotopy

[38, 41, 42] and LARS [23] approaches can be used to find solutions of (3) for all $\sigma \geq 0$. Alternatively, greedy approaches such as orthogonal matching pursuit (OMP) [43] and Stagewise OMP [22] can be used to find sparse solutions to (1) in certain cases.

Note that, from a transform coding point of view, it is not essential that the obtained sparse solution is the *unique* sparsest solution. In this context, the actual signal is b, and x is only a representation of b with respect to the given redundant dictionary.

2.3 Compressed Sensing

The sparse recovery problem is also at the heart of the recently developed theory of compressed sensing. Here, the goal is to reconstruct a signal from fewer than "necessary" number of measurements, without compromising the resolution of the end product [7, 8, 10, 15].

Suppose a signal $f \in \mathbb{R}^n$ admits a sparse representation with respect to a basis B, i.e., $f = Bx$ where x has a few nonzero entries. Now, let M be an $m \times n$ *measurement matrix*, and suppose we sample f by measuring

$$b = Mf. \tag{4}$$

Under what conditions on M and the level of sparsity of f with respect to B can one recover the coefficient vector x from the measurements b? Clearly, once one recovers x, one also has the signal $f = Bx$. Using the notation of (1), this corresponds to setting $A = MB$. That is, one wishes to solve the optimization problem (2) where b is the possibly noisy measurement vector, σ denotes the noise level, and $A = MB$. Note that in this context it is essential that we recover the "right" set of coefficients: the vector x denotes the coefficients of the signal f with respect to a basis, thus there is a one-to-one correspondence between the set of coefficients and the signal.

2.3.1 Sparse signals and exact reconstruction

Let $r = 0$ in (1), and suppose that x is sparse. For a given matrix A, if x is sufficiently sparse (i.e., $\|x\|_0$ is sufficiently small), then one can recover x exactly by solving the convex optimization problem (3) with $\sigma = 0$. For example, [17, 18, 26] derive conditions on the sparsity of x that are based on the *mutual coherence* of A, i.e., the absolute value of the largest-in-magnitude off-diagonal element of the Gram matrix of A. In contrast, [7, 8, 10] derive conditions on the sparsity of x that are based on the *restricted isometry constants* of A, which are determined by the smallest and largest singular values of certain submatrices of A.

It is important to note that both families of sufficiency conditions mentioned above are quite pessimistic and difficult to verify for a given matrix A. Fortunately, for certain types of random matrices one can prove that these, or even stronger conditions, hold with high probability; see, e.g, [9, 10, 21, 39].

Greedy algorithms such as OMP provide an alternative for finding the sparsest solution of (1) when $r = 0$. When the signal is sufficiently sparse, OMP can recover a signal exactly, and the sufficient conditions in this case depend on the (cumulative) mutual coherence of A [49]. Interestingly, many of these greedy algorithms are purely algorithmic in nature, and do not explicitly minimize a cost function.

2.3.2 Compressible signals and noisy observations

For sparse recovery techniques to be of practical interest, it is essential that they can be extended to more realistic scenarios where the signal is not exactly sparse but instead

compressible. This means that its sorted coefficients with respect to some basis decay rapidly. Moreover, the measurements in a practical setting will typically be noisy. It is established in [8, 15, 50] that the solution of (3) approximates a compressible signal surprisingly well. In fact, the approximation obtained from m random measurements turns out to be almost as good as the approximation one would obtain if one knew the largest m coefficients of the signal. Similarly, the results on OMP can be generalized to the case where the underlying signal is compressible [49].

2.3.3 Applications

Motivated by the results outlined above, compressed sensing ideas have been applied to certain problems in signal processing. Below we list some examples of such applications.

When the measurement operator M in (4) is replaced by a restriction operator, the sparse reconstruction problem can be viewed as an interpolation problem. In particular, suppose that $M = R_m M_n$, where M_n is an $n \times n$ (invertible) measurement matrix and R_m is an $m \times n$ restriction operator that removes certain rows of M_n. Equivalently, one has access only to m samples of the n-dimensional signal f, given by Mf, which one wishes to interpolate to obtain the full signal f. This problem can be phrased as a sparse approximation problem and thus the original signal can be obtained by solving (3). In exploration seismology, this approach was successfully applied in [46], where a Bayesian method exploiting sparsity in the Fourier domain was developed, and in [28, 29], where sparsity of seismic data in the curvelet domain is promoted.

Compressed sensing has also led to promising new methods for image reconstruction in medical imaging, which are largely founded on results given in [7]. MRI images are often sparse in the wavelet transform whereas angiograms are inherently sparse [36, 37] in image space. Moreover, they often have smoothly varying pixel values, which can be enforced during reconstruction by including a total variation [44] penalty term. In that case, one aims to find a balance between the variation in Bx and sparsity of x,

$$\underset{x}{\text{minimize}} \quad \|x\|_1 + \gamma \text{TV}(Bx) \quad \text{subject to} \quad \|MBx - b\|_2 \leq \sigma, \tag{5}$$

where the function $\text{TV}(\cdot)$ measures the total variation in the reconstructed signal Bx.

3 Sparco test problems

The collection of Sparco test problems is made available through a consistent interface, and a single gateway routine returns a problem structure that contains all components of a test problem, including the constituent operators. Sparco includes a suite of fast operators which are used define a set of test problems of the form (1). In order to define a consistent interface to a set of test problems that covers a broad range of applications, the operator A is always formulated as the product

$$A = MB.$$

The measurement operator M describes how the signal is sampled; the operator B describes the sparsity basis in which the signal can be sparsely represented. Examples of measurement operators are the Gaussian ensemble and the restricted Fourier operator. Examples of sparsity bases include wavelets, curvelets, and dictionaries that stitch together common bases.

Every operator in the Sparco framework is a function that takes a vector as an input, and returns the operator's action on that vector. As shown in §4, operators are instantiated

using the natural dimensions of the signal, and automatically reshape input and output vectors as needed.

3.1 Generating problems

The gateway function `generateProblem` is the single interface to all problems in the collection. Each problem can be instantiated by calling `generateProblem` with a unique problem-identifier number, and optional flags and keyword-value pairs:

```
P = generateProblem(id, [[key1,value1 | flag1],...]);
```

This command has a single output structure `P` which encodes all parts of the instantiated problem. The details of the problem structure are discussed in §3.2. The optional keyword-value pairs control various aspects of the problem formulation, such as the problem size or the noise level. In this way, a single problem definition can yield a whole family of related problem instances.

For instance, problem five generates a signal of length n from a few atoms in a DCT-Dirac dictionary. Two versions of this problem can be generated as follows:

```
P1 = generateProblem(5);          % Signal of length n = 1024 (default)
P1 = generateProblem(5,'n',2048); % Signal of length n = 2048
```

Documentation for each problem, including a list of available keyword-value pairs, the problem source, and relevant citations, can be retrieved with the call

```
generateProblem(id, 'help');
```

Other commonly supported flags are `'show'` for displaying example figures associated with the problem and `'noseed'` to suppress the initialization of the random number generators.

A full list of all available problem identifiers in the collection can be retrieved via the command

```
pList = generateProblem('list');
```

This is useful, for example, when benchmarking an algorithm on a series of test problems or for checking which problem identifiers are valid. The code fragment

```
pList = generateProblem('list');
for id = pList
    P = generateProblem(id);
    disp(P.info.title);
end
```

prints the names of every problem in the collection. The field `info.title` is common to all problem structures.

3.2 Problem structure

A single data structure defines all aspects of a problem, and is designed to give flexible access to specific components. For the sake of uniformity, there are a small number of fields that are guaranteed to exist in every problem structure. These include function handles for the operators A, B, and M, the data for the observed signal b, and the dimensions of the (perhaps unobserved) actual signal $b_0 := Bx$; these fields are summarized in Table 1.

Field	Description
`A, B, M`	Function handles for operators A, B, and M
`b`	the observed signal b
`op`	summary of all operators used
`reconstruct`	function handle to reconstruct a signal from coefficients in x
`info`	information structure (see Appendix B)
`signalSize`	dimensions of the actual signal, $b_0 := Bx$
`sizeA, sizeB, sizeM`	dimensions of the operators A, B, and M

Table 1: Basic fields in a problem structure

Seismic imaging	Hennenfent and Herrmann [30–32]
Blind source separation	Saab et al. [45]
MRI imaging	Lustig et al. [36], Candès et al. [7]
Sine and spikes	Candès and Romberg [6]
Basis pursuit	Chen et al. [11, 19]
Total variation	Takhar et al. [47]

Table 2: List of problem sources

Other useful fields include `op`, `info` and `reconstruct`. The field `op` provides a structure with all individual operators used to define `A`, `B` and `M`, as well as a string that describes each operators. The field `info` provides background information on the problem, such as citations and associated figures. The field `reconstruct` provides a method that takes a vector of coefficients x and returns a matrix (or vector) b_0 that has been reshaped to correspond to the original signal.

Many of the problems also contain the fields `signal` which holds the target test problem signal or image, `noise` which holds the noise vector r (see (1)), and, if available, `x0` which holds the true expansion of the signal in the sparsity basis B.

4 Sparco operators

At the core of the Sparco architecture is a large library of linear operators. Where possible, specialized code is used for fast evaluation of matrix-vector multiplications. Once an operator has been created, e.g.,

```
D = opDCT(128);
```

matrix-vector products with the created operator can be accessed as follows:

```
y = D(x,1);    % gives y := Dx
x = D(y,2);    % gives x := D'y
```

A full list of the basic operators available in the Sparco library is given in Tables 3 and 4.

MATLAB classes can be used to overload operations commonly used for matrices so that the objects in that class behave exactly like explicit matrices. Although this mechanism is not used for the implementation of the Sparco operators, operator overloading can provide a very convenient interface for the user. To facilitate this feature, Sparco provides the function `classOp`:

MATLAB function	Description
`opBinary`	binary (0/1) ensemble
`opBlockDiag`	compound operator with operators on the diagonal
`opBlur`	two-dimensional blurring operator
`opColumnRestrict`	restriction on matrix columns
`opConvolve1d`	one-dimensional convolution operator
`opCurvelet2d`	two-dimensional curvelet operator
`opDCT`	one-dimensional discrete cosine transform
`opDiag`	scaling operator
`opDictionary`	compound operator with operators abutted
`opDirac`	identity operator
`opFFT`	one-dimensional FFT
`opFFT2d`	two-dimensional FFT
`opFFT2C`	centralized two-dimensional FFT
`opFoG`	subsequent application of a set of operators
`opGaussian`	Gaussian ensemble
`opHaar`	one-dimensional Haar wavelet transform
`opHaar2d`	two-dimensional Haar wavelet transform
`opHeaviside`	Heaviside matrix operator
`opKron`	Kronecker product of two operators
`opMask`	vector entry selection mask
`opMatrix`	wrapper for matrices
`opPadding`	pad and unpad operators equally around each side
`opReal`	discard imaginary components
`opRestriction`	vector entry restriction
`opSign`	sign-ensemble operator
`opWavelet`	wavelet operator
`opWindowedOp`	overcomplete windowed operator

Table 3: The operators in the Sparco library

```
C = classOp(op);            % Create matrix object C from op
C = classOp(op,'nCprod');   % Additionally, create a global counter variable nCprod
```

These calls take an operator `op` and return an object from the operator class for which the main matrix-vector operations are defined. In its second form, the `classOp` function accepts an optional string argument and creates a global variable that keeps track of the number of multiplications with C and C^T. The variable can be accessed from MATLAB's base workspace. The following example illustrates the use of `classOp`:

```
F  = opFFT(128);
G  = classOp(F);
g1 = F(y,2);       % gives g1 := F^T y
g2 = G'*y;         % gives g2 := G^T y ≡ F^T y
```

Operator type	MATLAB function
Ensembles	`opBinary, opSign, opGaussian`
Selection	`opMask, opColumnRestrict, opRestriction`
Matrix	`opDiag, opDirac, opMatrix, opToMatrix`
Fast operators	`opCurvelet, opConvolve1d, opConvolve2d, opDCT, opFFT, opFFT2d, opFFT2C, opHaar, opHaar2d, opHeaviside, opWavelet`
Compound operators	`opBlockDiag, opDictionary, opFoG, opKron, opWindowedOp`
Nonlinear	`opReal, opPadding`

Table 4: Operators grouped by type.

4.1 Meta operators

Several tools are available for conveniently assembling more complex operators from the basic operators. The five meta-operators `opFoG`, `opDictionary`, `opTranspose`, `opBlockDiag`, and `opKron` take one or more of the basis operators as inputs, and assemble them into a single operator:

```
H = opFoG(A1,A2,...);          % H := A_1 · A_2 · ... · A_n
H = opDictionary(A1,A2,...);   % H := [A_1 | A_2 | ··· | A_n]
H = opTranspose(A);            % H := A^T
H = opBlockDiag(A1,A2,...);    % H := diag(A1, A2, ...)
H = opKron(A1,A2);             % H := A_1 ⊗ A_2
```

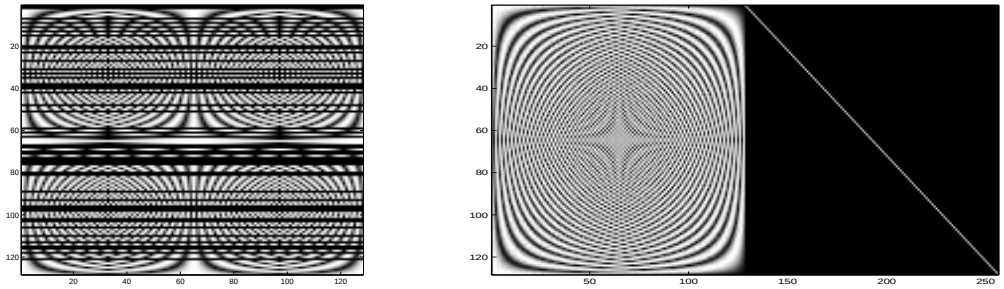
A sixth meta-operator, `opWindowedOp`, is a mixture between `opDictionary` and `opBlockDiag` in which blocks can partially overlap rather than fully (`opDictionary`), or not at all (`opBlockDiag`). A further two differences are that only a single operator is repeated and that each operator is implicitly preceded by a diagonal window operator.

4.2 Ensemble operators and general matrices

The three ensemble operators (see Table 4) can be instantiated by simply specifying their dimensions and a mode that determines the normalization of the ensembles. Unlike the other operators in the collection, the ensemble operators can be instantiated as explicit matrices (requiring $\mathcal{O}(m \cdot n)$ storage), or as implicit operators. When instantiated as implicit operators, the random number seeds are saved and rows and columns are generated on the fly during multiplication, requiring only $\mathcal{O}(n)$ storage for the normalization coefficients.

4.3 Selection operators

Two selection operators are provided: `opMask` and `opRestriction`. In forward mode, the restriction operator selects certain entries from the given vector and returns a correspondingly shortened vector. In contrast, the mask operator evaluates the dot-product with a binary vector thus zeroing out the entries instead of discarding them, and returns a vector of the same length as the input vector.



(a) (b)

Figure 1: Plots of the explicit matrices for (a) the partial Fourier measurement matrix and (b) the DCT-Dirac dictionary.

4.4 Fast operators

Sparco also provides support for operators with a special structure for which fast algorithms are available. Such operators in the library include Fourier, discrete cosine, wavelet, two-dimensional curvelet, and one-dimensional convolution of a signal with a kernel.

For example, the following code generates a partial 128×128 Fourier measurement operator (F), a masked version with 30% of the rows randomly zeroed out (M), and a dictionary consisting of an FFT and a scaled Dirac basis (B):

```
m = 128;
D = opDiag(m,0.1);   F = opFFT(m);        % D is a diagonal operator, F is an FFT
M = opFoG(opMask(rand(m,1) < 0.7),F);     % M is a masked version of F
B = opDictionary(F,D);                    % B = [F D]
```

4.5 Utilities

For general matrices there are three operators: `opDirac`, `opDiag`, and `opMatrix`. The Dirac operator coincides with the identity matrix of desired size. Diagonal matrices can be generated using `opDiag` which takes either a size and scalar, or a vector containing the diagonal entries. General matrix operators can be created using `opMatrix` with a (sparse) matrix as an argument.

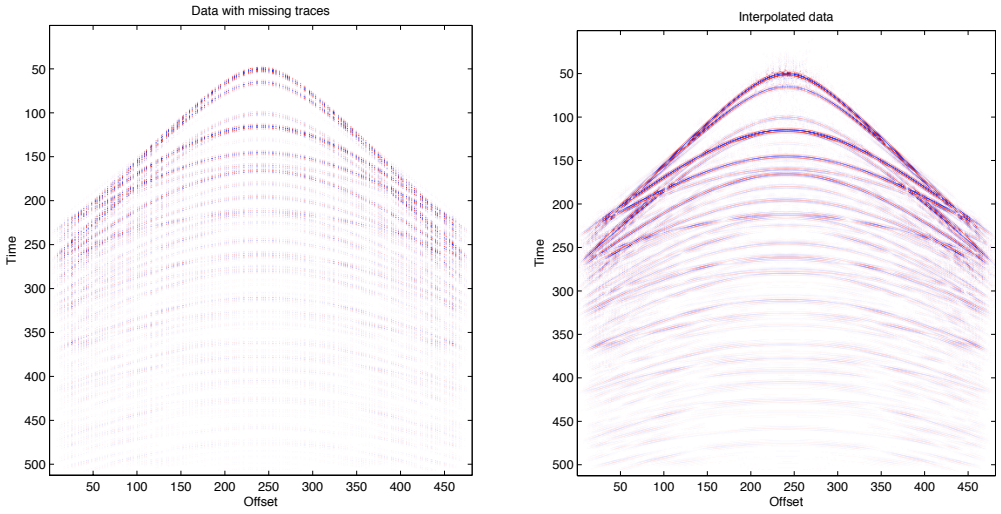
The `opToMatrix` utility function takes an implicit linear operator and forms and returns an explicit matrix. Figure 1 shows the results of using this utility function on the operators M and B defined in §4.4:

```
Mexplicit = opToMatrix(M); imagesc(Mexplicit);
Bexplicit = opToMatrix(B); imagesc(Bexplicit);
```

5 Fast prototyping using Sparco operators

In this section we give an example of how Sparco operators can be used for fast prototyping ideas and solving real-life problems. This example, whose source code can be found in `$SPARCO/examples/exSeismic.m`, inspired test problem 901.

The set of observed measurements is a spatiotemporal sampling at the Earth's surface of the reflected seismic waves caused by a seismic source. Due to practical and economical



(a) (b)

Figure 2: Seismic wavefield reconstruction. (a) Input data with missing offsets and (b) interpolated data using SPGL1 to solve a sparse recovery problem in the curvelet domain.

constraints, the spatial sampling is incomplete (Figure 2a). The data needs to be interpolated before further processing can be done to obtain an image of the Earth's subsurface. A successful approach to the seismic wavefield reconstruction problem is presented in [29, 33] where the authors form a sparse recovery problem in the curvelet domain [5] that is approximately solved using a solver for large-scale 1-norm regularized least squares. In this example, we want to form the same sparse recovery problem and solve it using SPGL1. Listing 1 shows the code to load the data, setup the operators, and solve the interpolation problem.

6 Implementation

One of the main considerations in the design of Sparco is to provide an environment that can easily expand to accommodate new test problems and operators. Indeed, the current incarnation of Sparco should only be viewed as a baseline version, and we plan to continue to add problems and operators as needed. In this section we describe some of the main components of the Sparco implementation.

6.1 Directory layout

The directory organization for Sparco is shown in Figure 3. Starting from the left, the first subtree under the root directory is `problems`, which contains all of the problem definition files. Each problem definition is a single file with the name `probNNN.m`, where `NNN` is a three digit problem identifier number. The gateway routine `generateProblem` searches this directory in order to instantiate a particular problem, or to generate a list of valid problem identifiers. This approach allows new test problems to be added without having to modify any existing code. (The structure of a problem-definition file is described in §6.2.) The

Listing 1: Prototyping a seismic imaging application using the Sparco operators.

```
% Load sesimic data (the RHS) and trace positions.
load data
load mask
[n1,n2] = size(data);

% Setup the sparse recovery problem
M = opColumnRestrict(n1,n2,find(mask),'discard');
B = opCurvelet2d(n1,n2,6,32);
A = opFoG(M,B);

rdata = data(:,find(mask));
b = rdata(:);

% Set options for SPGL1
opts = spgSetParms('iterations' , 100  , ...
                   'verbosity'  , 1    , ...
                   'bpTol'      , 1e-3 , ...
                   'optTol'     , 1e-3 );

% Solve sparse recovery problem
[x, r, g, info] = spgl1(A, b, 0, 0, [], opts);

% Construct interpolated seismic data
res = reshape(B(x,1),n1,n2);
```

subdirectory `data` contains auxiliary data, such as images and audio signals, that may be needed by the problem definitions.

The operators used to implement the test problems are stored in the `operators` directory. By convention, all operator names start with the `op` prefix; otherwise, there are no real restrictions on the operator name. The subdirectory `tools` contains functions that are called by the test problems or operators, and are also useful in their own right. A number of useful tools are discussed in more detail in §6.4. The subdirectory `examples` contains a number of scripts that illustrate the use of different Sparco features such as problem generation, creation of operators, their conversion to classes, and examples on using various solvers with the Sparco toolbox. The subdirectory `build` is used to store automatically generated figures and problem thumbnails, such as those output by `generateProblem` when the `output` flag is specified. The subdirectories `private` contain any specialized functions used by functions in the parent directory.

6.2 Implementation of a test problem

In this section we give an extended example of how a test problem is defined; we base this example on the existing Problem 5. In this problem, we generate a signal composed of a small number of cosine waves and superimpose a series of random spikes. This signal is measured using a Gaussian ensemble. For simplicity, we fix the signal length to $n = 1024$, and allow the number of observations m and the number of spikes k to be optional parameters; by

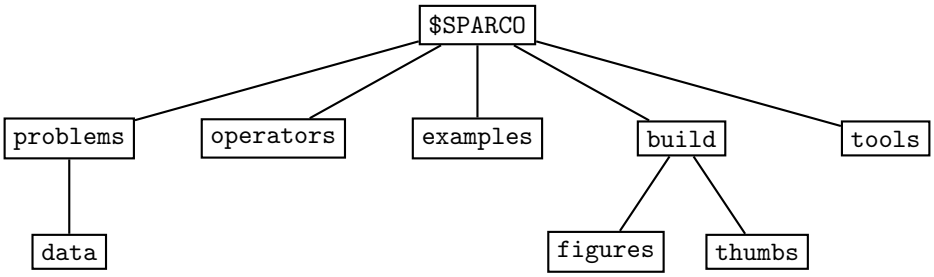


Figure 3: The Sparco directory structure

default, $m = 300$ and $k = \lfloor m/5 \rfloor$. In addition, we support the `'noseed'` flag that can be used to suppress the initialization of the random number generators.

The first step of the problem implementation is to define the function and parse the optional parameters and flags. To implement the options described above, we begin the function as follows:

```
1  function P = prob005(varargin)
2
3  % The following lines parse the input options.  These are required.
4  [opts,varg] = parseDefaultOpts(varargin{:});
5  [parm,varg] = parseOptions(varg,{'noseed'},{'k','m'});
6
7  % Retrieve optional parameter values (set to defaults if necessary).
8  m = getOption(parm,'m', 300);
9  k = getOption(parm,'k', floor(m/5));
```

The function `parseDefaultOpts` function parses the input options and searches for the generic flags `show` and `output`; these flags are stored in the output argument `opts` alongside various Sparco system settings. (Appendix B describes the default fields set by `parseDefaultOpts`.) Any processed arguments are stripped and the remaining arguments are returned in `varg`.

The function `parseOptions` parses any remaining options. In this case we scan for the `noseed` flag and optional parameters `k` and `m`. If these optional arguments are found in the argument list, the output structure `parm` will contain the fields `.k` and `.m`. Flag fields are always created, and either set to "true" if the flag was present in the argument list; otherwise, the flag is initialized as "false." The subsequent calls to `getOption` extract field information from the parameter structure. If the given field does not exist, the specified default value is returned.

For reproducibility, it is important to initialize the random number generators. However, in certain situations, e.g., computing phase-transition diagrams [21], it is desirable to manually initialize the random number generators, and disable the initialization within the problem definition. We therefore suggest the following construction:

```
10 if ~parm.noseed      % Reset RNGs by default.
11    randn('state',0);
12    rand('state',0);
13 end
```

This completes the setup of the problem definition function, and at this point, problem-specific data can be defined. In this case, the signal can be generated:

```
% Cosine part of signal
D           = classOp(opDCT(n));
cosCoef     = zeros(n,1);
cosCoef( 4) = 2*sqrt(n/2);
cosCoef(10) = 3*sqrt(n/2);
cosCoef(21) = - sqrt(n/2);
cosine      = D * cosCoef;

% Spike part of signal
p = randperm(n); p = p(1:k);
spike    = zeros(n,1);
spike(p) = randn(k,1);
```

and the fields of the problem structure initialized:

```
P.signal      = cosine + spike;
P.op.DCT      = opDCT(n);
P.op.Dirac    = opDirac(n);
P.op.Dict     = opDictionary(P.op.DCT, P.op.Dirac);
P.op.Gaussian = opGaussian(m,n,2);
P.B           = P.op.Dict;
P.M           = P.op.Gaussian;
P.b           = P.M(P.signal,1);    % b := M · signal
P             = completeOps(P);
end % function prob005
```

Note that although storing the individual operators is not strictly necessary (we could have written `P.B = opDictionary(opDCT(n),opDirac(n));`), it is recommended because it gives users easy access to the operators.

The final line of code uses the function `completeOps(P)` to finalize the problem structure `P`. In case any of the required operators A, B, or M have not been defined in `P`, `completeOps(P)` initializes the missing elements to default values: the fields B or M are set to identity operators and A is set to MB. In addition, `completeOps` creates a function `P.reconstruct` that takes an input x and outputs Bx, reshaped according to the size of `P.signal`, or to the size specified by `P.signalSize` in the case that no signal is given.

6.3 Implementation of operators

A call to one of the operator functions instantiates a particular linear operator with some specified characteristics. The operator functions are in fact little more than wrappers to lower-level routines. Each operator function simply creates an anonymous function which has certain parameters fixed, and then passes back a handle to that anonymous function.

Each Sparco operator function is a single MATLAB m-file that contains a private function that implements the lower-level routine. Listing 2 shows how the DCT operator is implemented.

As we have seen, the parameter `mode` determines whether the function returns a matrix-vector product (`mode=1`) or a matrix-vector product with the adjoint (`mode=2`). The special case `mode=0` does not perform and computation, and instead and returns a structure that

Listing 2: The MATLAB m-file `opDCT.m` implements the DCT operator

```
function op = opDCT(n)
  op = @(x,mode) opDCT_intrnl(n,x,mode);
end % function opDCT

function y = opDCT_intrnl(n,x,mode)
  if mode == 0                          % Report properties of operator
     y = {n,n,[0,1,0,1],{'DCT'}};
  elseif mode == 1                      % Return y = Dx
     y = idct(x);
  else                                  % Return y = D^T x
     y = dct(x);
  end
end % function opDCT_intrnl
```

contains the number of rows and columns of the operator, a binary vector that indicates whether the products of the forward operator with real and complex vectors are complex and similarly for the adjoint. The last field is a cell array that begins with the operator name, and contains any other operator-specific information. This information structure can be used by solvers to determine the problem size and domain. It is also used by `opDictionary` and `opFoG` when creating composite operators in order to verify that the individual operator sizes are compatible.

A useful tool for debugging linear operators is the function `dottest` in the directory `tools`. When called with an operator A, it generates random real (and complex, if appropriate) vectors x and y and checks whether $\langle Ax, y\rangle = \langle x, A^*y\rangle$.

6.4 Tools

In Sparco all problems are formulated in matrix-vector form. This means that, when the underlying signals are in two- or higher- dimensional space, they need to be reshaped before further processing can be done. This is mostly an issue for total-variation-based formulations (see (5)) because a finite-difference operator needs to be tailored to the shape of the signal. The directory `tools` contains the function `opDifference` for this purpose. This function behaves similarly to the reconstruction function `P.reconstruct`, but instead of returning the signal reconstruction b_0, it returns an array whose columns contain the differences along each dimension of the reconstructed signal; the number of columns coincides with the dimensionality of the problem. The use of this function is illustrated in `exTV` in the directory `examples`.

To reduce the dependence on external toolboxes, we reimplemented the Shepp-Logan phantom generation in `ellipses` and extended it to the modified three-dimensional model in `ellipsoids` (see also [48, p.50] and [51]). Figure 4 shows the data obtained using this function. The plots were generated using the `exSheppLogan` example script.

Many other tools are implemented in Sparco and more are being added as the number of test problems grows. For a complete list of functions, see the `tools` directory.

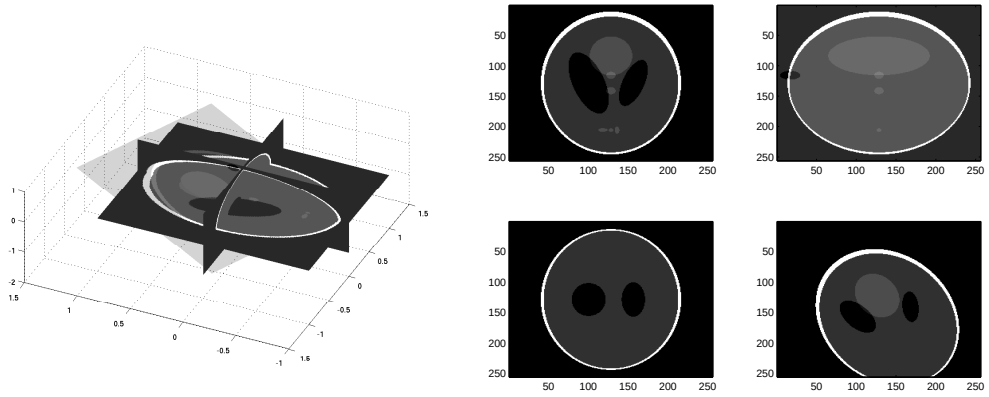


Figure 4: Four slices from the three-dimensional Shepp-Logan phantom.

7 Conclusions

Sparco is organized as a flexible framework providing test problems for sparse signal reconstruction as well as a library of operators and tools. The problem suite currently contains 25 problems and 28 operators. We will continue to add more problems in the near future. Contributions from the community are very welcome! For the latest version of Sparco, please visit `http://www.cs.ubc.ca/labs/scl/sparco/`.

Acknowledgements

The authors would like to thank to thank Michael Lustig and Stephen Wright, who kindly contributed code: Michael Lustig contributed code from the SparseMRI toolbox [35], and Stephen Wright contributed code from test problems packaged with the GPSR solver [24]. Also, our thanks to Michael Saunders and Stephen Wright for courageously testing early versions of Sparco, and for their thoughtful and patient feedback. We are delighted to acknowledge the open-source package Rice Wavelet Toolbox [1], which was invaluable in developing Sparco.

A Installation and prerequisites

In the implementation of Sparco an effort was made to reduce the number of dependencies as much as possible to simplify installation and keep the prerequisites minimal. At present there are two main dependencies: the Rice Wavelet Toolbox [1] for the wavelet operator, which is included in the distribution, and CurveLab v.2.1 [5] for the curvelet transform. Because Sparco provides general test problems, there are no dependencies on any solvers. (Several of the example scripts, however, do require external solvers.) In order to function properly, Sparco should be included in the MATLAB path. This is done by running `sparcoSetup` once after downloading the files.

Field	Description
`title`	Short description of the problem
`citations`	Relevant BibTeX entries
`fig`	figures associated with the problem (see `$SPARCO/figures/figures`)

Table 5: Basic fields in the `info` substructure

B Problem documentation

Problem documentation forms an integral part of the problem data structure returned by `generateProblem`. While most of this information has to be explicitly provided in the `P.info` field by the problem definition file (see Table Table 5), some of it can be automatically generated by the `P = completeOps(P)`. The purpose of this function is to replace missing operators M or B by the identity operator and instantiating $A := B \cdot M$. In addition, it stores the operator size in fields `P.sizeA`, `P.sizeB`, and `P.sizeM`, and initializes the operator string description fields in the problem substructure `P.op`. In the case of the example in §6.3, this gives

```
>> disp(P.op)
         DCT: @(x,mode)opDCT_intrnl(n,x,mode)
       Dirac: @(x,mode)opDirac_intrnl(n,x,mode)
        Dict: @(x,mode)opDictionary_intrnl(m,n,c,weight,opList,x,mode)
    Gaussian: @(x,mode)opMatrix_intrnl(A,opinfo,x,mode)
        strA: 'Gaussian * Dictionary(DCT, Dirac)'
        strB: 'Dictionary(DCT, Dirac)'
        strM: 'Gaussian'
```

Finally, each problem includes its own documentation as part of the leading comment string in the problem definition file.

References

[1] R. Baraniuk, H. Choi, F. Fernandes, B. Hendricks, R. Neelamani, V. Ribeiro, J. Romber, R. Gopinath, H.-T. Guo, M. Lang, J. E. Odegard, and D. Wei, *Rice Wavelet Toolbox*. http://www.dsp.rice.edu/software/rwt.shtml, 1993.

[2] E. van den Berg and M. P. Friedlander, *In pursuit of a root*, Tech. Rep. TR-2007-19, Department of Computer Science, University of British Columbia, June 2007.

[3] R. Boisvert, R. Pozo, K. Remington, R. Barrett, and J. Dongarra, *Matrix Market: A web resource for test matrix collections*, in The quality of numerical software: assesment and enhancement, R. F. Boisvert, ed., Chapman & Hall, London, 1997, pp. 125–137.

[4] E. J. Candès, *Compressive sampling*, in Proceedings of the International Congress of Mathematicians, 2006.

[5] E. J. Candès, L. Demanet, D. L. Donoho, and L.-X. Ying, *CurveLab*. http://www.curvelet.org/, 2007.

[6] E. J. Candès and J. Romberg, *ℓ_1-magic*. http://www.l1-magic.org/, 2007.

[7] E. J. Candès, J. Romberg, and T. Tao, *Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information*, IEEE Trans. Inform. Theory, 52 (2006), pp. 489–509.

[8] ——, *Stable signal recovery from incomplete and inaccurate measurements*, Comm. Pure Appl. Math., 59 (2006), pp. 1207–1223.

[9] E. J. Candès and T. Tao, *Decoding by linear programming*, IEEE Trans. Inform. Theory, 51 (2005), pp. 4203–4215.

[10] ——, *Near-optimal signal recovery from random projections: Universal encoding strategies?*, IEEE Trans. Inform. Theory, 52 (2006), pp. 5406–5425.

[11] S. S. Chen, D. L. Donoho, and M. A. Saunders, *Atomic decomposition by basis pursuit*, SIAM J. Sci. Comput., 20 (1998), pp. 33–61.

[12] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, Philadelphia, PA, 1992.

[13] I. Daubechies, M. Defrise, and C. de Mol, *An iterative thresholding algorithm for linear inverse problems with a sparsity constraint*, Comm. Pure Appl. Math., 57 (2004), pp. 1413–1457.

[14] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with COPS*, Tech. Rep. ANL/MCS-246, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 2000. Revised January 2001.

[15] D. L. Donoho, *Compressed sensing*, IEEE Trans. Inform. Theory, 52 (2006), pp. 1289 – 1306.

[16] D. L. Donoho, *For most large underdetermined systems of linear equations the minimal ℓ^1-norm solution is also the sparsest solution*, Comm. Pure Appl. Math., 59 (2006), pp. 797–829.

[17] D. L. Donoho and M. Elad, *Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization*, Proc. Natl. Acad. Sci. USA, 100 (2003), pp. 2197–2202.

[18] D. L. Donoho and X. Huo, *Uncertainty principles and ideal atomic decomposition*, IEEE Transactions on Information Theory, 47 (2001), pp. 2845–2862.

[19] D. L. Donoho and I. M. Johnstone, *Ideal spatial adaptation by wavelet shrinkage*, Biometrika, 81 (1994), pp. 425–455.

[20] D. L. Donoho, V. C. Stodden, and Y. Tsaig, *Sparselab*. http://sparselab.stanford.edu/, 2007.

[21] D. L. Donoho and J. Tanner, *Sparse nonnegative solution of underdetermined linear equations by linear programming*, Proc. Natl. Acad. Sci. USA, 102 (2005), pp. 9446–9451.

[22] D. L. Donoho and Y. Tsaig, *Sparse solution of underdetermined linear equations by stagewise orghogonal matching pursuit*. http://www.stanford.edu/~tsaig/research.html, March 2006.

[23] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, *Least angle regression*, Ann. Statist., 32 (2004), pp. 407–499.

[24] M. Figueiredo, R. Nowak, and S. J. Wright, *Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems*, February 2007. To appear in IEEE Trans. on Selected Topics in Signal Processing.

[25] N. I. M. Gould, D. Orban, and Ph. L. Toint, *CUTEr and SifDec: A constrained and unconstrained testing environment, revisited*, ACM Trans. Math. Softw., 29 (2003), pp. 373–394.

[26] R. Gribonval and M. Nielsen, *Sparse representations in unions of bases*, IEEE Transactions on Information Theory, 49 (2003), pp. 3320–3325.

[27] E. Hale, W. Yin, and Y. Zhang, *Fixed-point continuation for l1-minimization: Methodology and convergence*, Tech. Rep. TR07-07, Department of Computational and Applied Mathematics, Rice University, Houston, TX, 2007.

[28] G. Hennenfent and F. J. Herrmann, *Sparseness-constrained data continuation with frames: Applications to missing traces and aliased signals in 2/3-D*, in SEG International Exposition and 75th Annual Meeting, 2005.

[29] ——, *Application of stable signal recovery to seismic interpolation*, in SEG International Exposition and 76th Annual Meeting, 2006.

[30] ——, *Irregular sampling: from aliasing to noise*, in EAGE 69th Conference & Exhibition, 2007.

[31] ——, *Random sampling: new insights into the reconstruction of coarsely-sampled wavefields*, in SEG International Exposition and 77th Annual Meeting, 2007.

[32] ——, *Simply denoise: wavefield reconstruction via jittered undersampling*, Tech. Rep. TR-2007-5, UBC Earth & Ocean Sciences Department, September 2007.

[33] F. J. Herrmann and G. Hennenfent, *Non-parametric seismic data recovery with curvelet frames*, Tech. Rep. TR-2007-1, UBC Earth & Ocean Sciences Department, January 2007.

[34] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky, *An efficient method for ℓ_1-regularized least squares.* To appear in *IEEE Trans. on Selected Topics in Signal Processing*, August 2007.

[35] M. Lustig, D. L. Donoho, and J. M. Pauly, *Sparse MRI Matlab code.* http://www.stanford.edu/~mlustig/SparseMRI.html, 2007.

[36] ——, *Sparse MRI: The application of compressed sensing for rapid MR imaging*, 2007. Submitted to Magnetic Resonance in Medicine.

[37] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly, *Compressed sensing MRI*, 2007. Submitted to IEEE Signal Processing Magazine.

[38] D. Malioutov, M. Çetin, and A. S. Willsky, *A sparse signal reconstruction prespective for source localization with sensor arrays*, IEEE Trans. Sig. Proc., 53 (2005), pp. 3010–3022.

[39] S. Mendelson, A. Pajor, and N. Tomczak-Jaegermann, *Uniform uncertainty principle for Bernoulli and subgaussian ensembles*, 2007. arXiv:math/0608665.

[40] B. K. Natarajan, *Sparse approximate solutions to linear systems*, SIAM J. Comput., 24 (1995), pp. 227–234.

[41] M. R. Osborne, B. Presnell, and B. A. Turlach, *The Lasso and its dual*, J. of Computational and Graphical Statistics, 9 (2000), pp. 319–337.

[42] M. R. Osborne, B. Presnell, and B. A. Turlach, *A new approach to variable selection in least squares problems*, IMA J. Numer. Anal., 20 (2000), pp. 389–403.

[43] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, *Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition*, in Proceedings of the 27th Annual Asilomar Conference on Signals, Systems and Computers, vol. 1, Nov. 1993, pp. 40–44.

[44] L. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Physica D, 60 (1992), pp. 259–268.

[45] R. Saab, F. J. Herrmann, D. Wang, and Ö. Yilmaz, *Seismic signal separation via sparsity promotion*, Tech. Rep. TR-2007-6, UBC Earth & Ocean Sciences Department, October 2007.

[46] M. D. Sacchi, T. J. Ulrych, and C. J. Walker, *Interpolation and extrapolation using a high-resolution discrete Fourier transform*, IEEE Transactions on Signal Processing, 46 (1998), pp. 31–38.

[47] D. Takhar, J. N. Laska, M. Wakin, M. Duarte, D. Baron, S. Sarvotham, K. K. Kelly, and R. G. Baraniuk, *A new camera architecture based on optical-domain compression*, in Proceedings of the IS&T/SPIE Symposium on Electronic Imaging: Computational Imaging, vol. 6065, January 2006.

[48] P. Toft, *The Radon transform: Theory and implementation*, PhD thesis, Department of Mathematical Modelling, Technical University of Denmark, 1996.

[49] J. A. Tropp, *Greed is good: Algorithmic results for sparse approximation*, IEEE Trans. Inform. Theory, 50 (2004), pp. 2231–2242.

[50] ——, *Just relax: Convex programming methods for identifying sparse signals in noise*, IEEE Trans. Inform. Theory, 52 (2006), pp. 1030–1051.

[51] H.-Y. Yu, S.-Y. Zhao, and G. Wang, *A differentiable Shepp-Logan phantom and its application in exact cone-beam CT*, Phys. Med. Biol., 50 (2005), pp. 5583–5595.