SPOT, distributed data, and you

Curt Da Silva

SLIM

University of British Columbia

Model problem

Suppose that you want to solve a sparsity-promoting interpolation problem

$$\min_{x} \|x\|_1$$
$$\text{such that } RMx = b$$

Model problem

Suppose that you want to solve a sparsity-promoting interpolation problem

$$\min_x \|x\|_1$$
$$\text{such that } RMx = b$$

Sampling operator - restricts vector to sampled locations

Model problem

Suppose that you want to solve a sparsity-promoting interpolation problem

$$\min_x \|x\|_1$$

$$\text{such that } RMx = b$$

Sparsity basis - maps (Curvelet, Fourier) coefficients to physical domain

Model problem

Suppose that you want to solve a sparsity-promoting interpolation problem

Coefficient vector

$$\min_x \|\overset{\swarrow}{x}\|_1$$

$$\text{such that } RMx = b$$

Model problem

Suppose that you want to solve a sparsity-promoting interpolation problem

$$\min_x \|x\|_1$$

$$\text{such that } RMx = b$$

Acquired data

Algorithm - linearized Bregman

$$z_{k+1} = z_k - t_k A^T (Ax_k - b)$$
$$x_{k+1} = S_\lambda(z_{k+1})$$

$A = RM$ - sampling + measurement operator

t_k - step size

$S_\lambda(x)$ - soft thresholding operator

$$S_\lambda(x) = \mathrm{sign}(x) \cdot \max(|x| - \lambda, 0)$$

Operations we need to perform

We need to repeatedly apply the *forward transform*

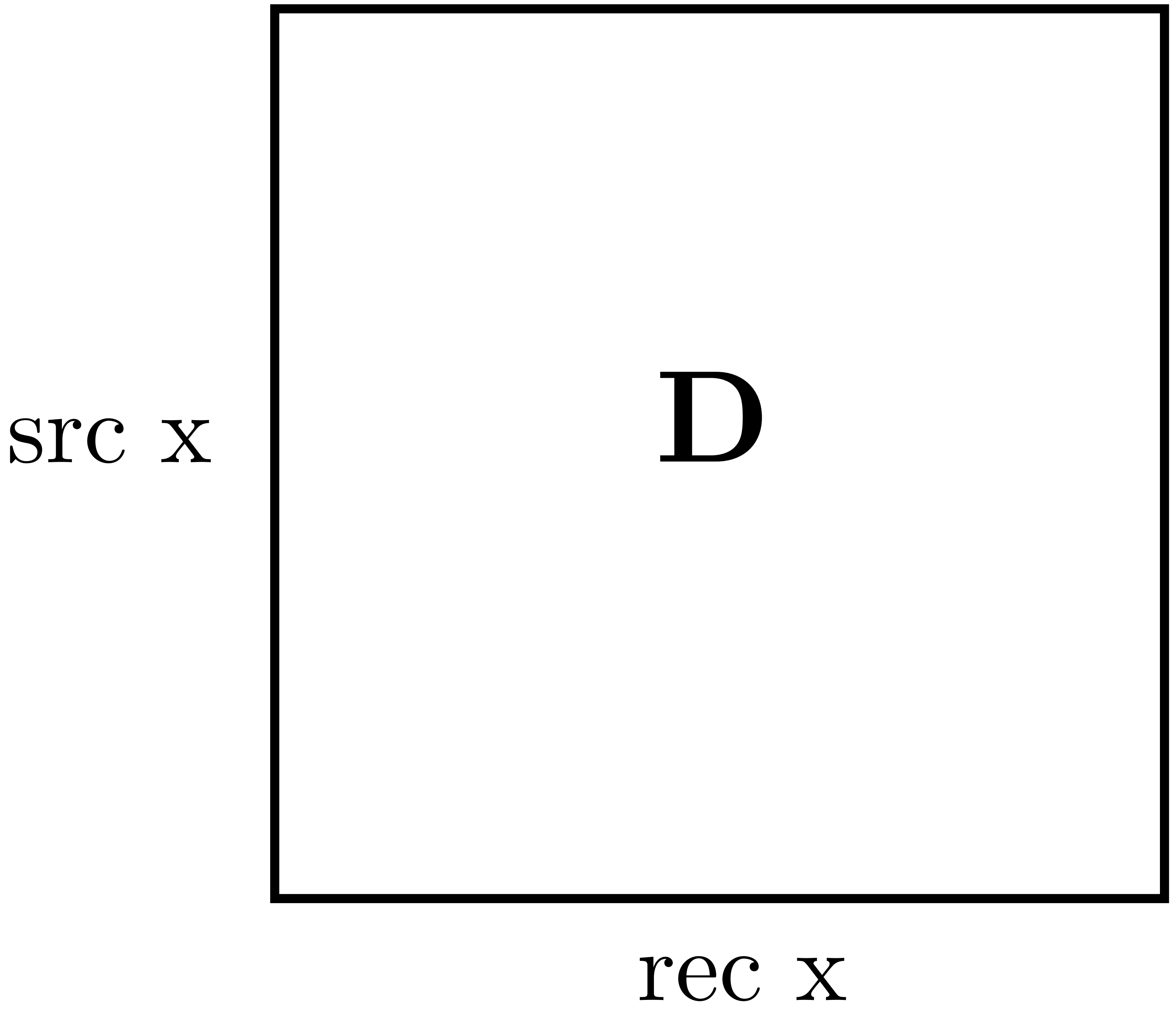
$$\text{physical space} \mapsto \text{coefficient space}$$

and apply the *adjoint transform*

$$\text{coefficient space} \mapsto \text{physical space}$$

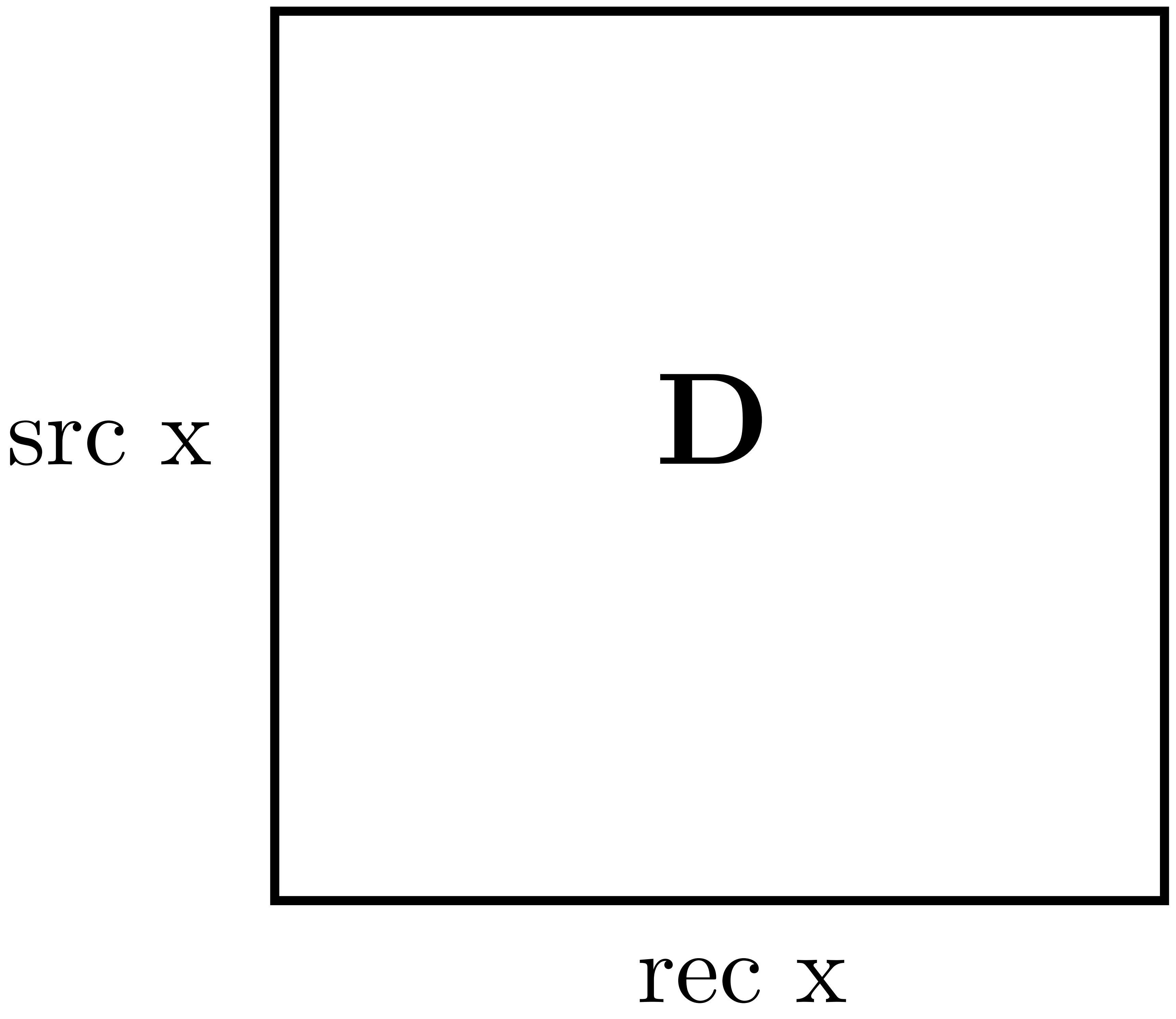
Details

Suppose we are working on two dimensional frequency slices



Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension



Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension

$$\mathbf{D}$$

src x

-1.0 + 0.0i	0.5 + 1.4i	0.5 - 1.4i
1.0 + 1.2i	0.5 - 0.9i	-1.5 - 0.3i
1.0 - 1.2i	-1.5 + 0.3i	0.5 + 0.9i

rec x

Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension

Applying the 1D Fourier transform to each column of D

$$\mathbf{FD}$$

$k_{\mathrm{src\ x}}$

0.6 + 0.0i	-0.3 + 0.5i	-0.3 + 0.5i
0.0 + 0.0i	0.0 + 0.0i	0.0 + 0.0i
2.3 + 0.0i	1.2 + 2.0i	1.2 - 2.0i

$\mathrm{rec\ x}$

Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension

Transposing the source and receiver dimensions

$$(\mathbf{FD})^T$$

rec x

0.6 + 0.0i	0.0 + 0.0i	2.3 + 0.0i
-0.3 + 0.5i	0.0 + 0.0i	1.2 + 2.0i
-0.3 + 0.5i	0.0 + 0.0i	1.2 - 2.0i

$k_{\text{src x}}$

Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension

Applying the 1D Fourier transform to the columns of this new array

$$\mathbf{F}(\mathbf{F}\mathbf{D})^T$$

$k_{\text{rec x}}$

0.0 + 0.0i	0.0 + 0.0i	0.0 + 0.0i
1.0 + 0.0i	0.0 + 0.0i	0.0 + 0.0i
0.0 + 0.0i	0.0+0.0i	4.0 + 0.0i

$k_{\text{src x}}$

Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension

Transpose the resulting array

This is our final result

$$(\mathbf{F}(\mathbf{F}\mathbf{D})^T)^T$$

$k_{\text{src x}}$

0.0 + 0.0i	1.0 + 0.0i	0.0 + 0.0i
0.0 + 0.0i	0.0 + 0.0i	0.0 + 0.0i
0.0 + 0.0i	0.0+0.0i	4.0 + 0.0i

$k_{\text{rec x}}$

Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension

We can also write this as

$$\mathbf{F}\mathbf{D}\mathbf{F}^T$$

$$\begin{array}{c|c|c|c|}
\cline{2-4}
 & 0.0 + 0.0i & 1.0 + 0.0i & 0.0 + 0.0i \\
\cline{2-4}
k_{\text{src x}} & 0.0 + 0.0i & 0.0 + 0.0i & 0.0 + 0.0i \\
\cline{2-4}
 & 0.0 + 0.0i & 0.0{+}0.0i & 4.0 + 0.0i \\
\cline{2-4}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{k_{\text{rec x}}} & \multicolumn{1}{c}{}
\end{array}$$

Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension

We can also write this as

$$\mathbf{FDF}^T$$

Apply $\mathbf{F}$ to the columns of $\mathbf{D}$

$k_{\text{src x}}$

0.0 + 0.0i	1.0 + 0.0i	0.0 + 0.0i
0.0 + 0.0i	0.0 + 0.0i	0.0 + 0.0i
0.0 + 0.0i	0.0+0.0i	4.0 + 0.0i

$k_{\text{rec x}}$

Details

If $\mathbf{D}$ our sparsity basis of choice is the 1D Fourier basis along each dimension

We can also write this as

$$\mathbf{FDF}^T$$

Apply $\mathbf{F}$ to the rows of $\mathbf{D}$

$k_{\text{src x}}$

0.0 + 0.0i	1.0 + 0.0i	0.0 + 0.0i
0.0 + 0.0i	0.0 + 0.0i	0.0 + 0.0i
0.0 + 0.0i	0.0+0.0i	4.0 + 0.0i

$k_{\text{rec x}}$

Standard Matlab

```
op_fftsrc = @(x) fft(x)/sqrt(nsrc);
op_fftrec = @(x) fft(x)/sqrt(nrec);
op_transp = @(x) x.’;
op_m = @(x) op_transp(op_fftrec(op_transp(opfftsrc(x))));
% transformed data
op_m(D);
```

Standard Matlab

That doesn’t look too bad

- it’s not intuitive to look at - hard to tell what’s going on

What if our sparsity basis changes in one dimension?

- hard to experiment

What if our data is distributed?

- not clear what to do here

Standard Matlab

How do we get adjoints/inverses?

How can I deal with more than two dimensions?

Kronecker Product

Mathematically, we can express $\mathbf{FDF}^T$ as

$$(\mathbf{F} \otimes \mathbf{F})\mathrm{vec}(\mathbf{D})$$

Kronecker Product

Mathematically, we can express $\mathbf{F}\mathbf{D}\mathbf{F}^T$ as

$$(\mathbf{F} \otimes \mathbf{F})\mathrm{vec}(\mathbf{D})$$

$\mathbf{F} \otimes \mathbf{F}$ - kronecker product of $\mathbf{F}$ and $\mathbf{F}$

$\mathrm{vec}(\mathbf{D})$ - reshape $\mathbf{D}$ in to a vector

Kronecker Product

How you read this

$$(\mathbf{F} \otimes \mathbf{F})\mathrm{vec}(\mathbf{D})$$

Kronecker Product

How you read this

$$(\mathbf{F} \otimes \underset{\uparrow}{\mathbf{F}})\mathrm{vec}(\mathbf{D})$$

Apply $\mathbf{F}$ to the first dimension of $\mathbf{D}$

Kronecker Product

How you read this

$$(\mathbf{F} \otimes \mathbf{F})\mathrm{vec}(\mathbf{D})$$

Apply $\mathbf{F}$ to the second dimension of $\mathbf{D}$

Kronecker Product

More generally

$$(\mathbf{B} \otimes \mathbf{A})\mathrm{vec}(\mathbf{D})$$

Apply $\mathbf{A}$ to the first dimension of $\mathbf{D}$

Kronecker Product

More generally

$$(\mathbf{B} \otimes \mathbf{A})\mathrm{vec}(\mathbf{D})$$

Apply $\mathbf{B}$ to the second dimension of $\mathbf{D}$

Using the SPOT toolbox

```
A = opDFT(nsrc);
B = opDFT(nrec);
F = opKron(B,A);
% transformed data
F*vec(D);
```

Advantages

Code now looks like the math

- if you understand the underlying math, you understand what’s happening
- adjoints, inverses automatically

Easy to change the operators in both dimensions

- easier to experiment with different transforms

Handling distributed data is nearly identical to serial data

Using the SPOT toolbox - serial version

```
% D resides on the current node
A = opDFT(nsrc);
B = opDFT(nrec);
F = opKron(B,A);
% transformed data
F*vec(D);
```

Using the SPOT toolbox - serial version

```
% D is distributed along columns
A = opDFT(nsrc);
B = opDFT(nrec);
F = oppKron2Lo(B,A);
% transformed data - distributed
F*vec(D);
```

Actual Matlab code

```
% Construct sampling + measurement operators
Rsrc = opRestriction(nsrc,sampled_indices);
Rrec = opDirac(nrec);
R = opKron(Rrec,Rsrc);
Msrc = opDFT(nsrc); Mrec = opDFT(nrec);
M = opKron(Mrec,Msrc);

% Construct composite operators, subsampled data
A = R*M; b = R*vec(D);

threshold = @(x) sign(x) .* max(abs(x)-lambda,0);
```

Actual Matlab code

```
x = zeros(nsrc*nrec,1); z = zeros(nsrc*nrec,1);
for itr=1:nitr
    z = z - t*A’*(A*x-b);
    x = threshold(x);
end
```

Actual Matlab code

```
x = zeros(nsrc*nrec,1); z = zeros(nsrc*nrec,1);
for itr=1:nitr
    z = z - t*A’*(A*x-b);
    x = threshold(x);
end
```

Previous algorithm

$$
\begin{aligned}
z_{k+1} &= z_k - t_k A^T (A x_k - b) \\
x_{k+1} &= S_\lambda(z_{k+1})
\end{aligned}
$$

SPOT toolbox

Allows us to implement multidimensional operations easily and consistently

- don’t need to worry about data shuffling, parallelization, etc

Code matches the math

- easier to understand, debug

All *matrix-free* - explicit matrices are never constructed, only matrix-vector products

SPOT Toolbox

Operations such as

A*B

A\B

A+B

c*A

are wrappers to functions you implement

- matrices never formed explicitly, but Matlab treats them as regular matrices

SPOT Toolbox

Lots of existing functionality

- Sums, products, inverses, diagonal operators, random matrices
- Fourier, Curvelet transform
- Parallel multilinear (Kronecker) products
- Demigration, migration, GN Hessian, Full Hessian operators in FWI
- Jacobian, GN Hessian for Hierarchical Tucker

Acknowledgements

https://www.slim.eos.ubc.ca/consortiumsoftware

SINBAD

NSERC CRSNG

This work was financially supported by SINBAD Consortium members BG Group, BGP, CGG, Chevron, ConocoPhillips, DownUnder GeoSolutions, Hess, Petrobras, PGS, Schlumberger, Statoil, Sub Salt Solutions and Woodside; and by the Natural Sciences and Engineering Research Council of Canada via NSERC Collaborative Research and Development Grant DNOISEII (CRDPJ 375142--08).