Spot — A linear operator toolbox for Matlab

Ewout van den Berg and Michael P. Friedlander

Department of Computer Science
University of British Columbia

SINBAD Sponsor Meeting, November 23–24, 2009

Sparco operator toolbox (SPOT)

- Sparco: test problem suite for compressed sensing
- Combinations of linear operators; $A = MB$, $B = [D, I]$
- Developed operator toolbox

```
D = opDCT(256);
I = opEye(256);
B = opDictionary(D,I);
M = ...;
A = opFoG(M,B);
```

Sparco operator toolbox (SPOT)

- Sparco: test problem suite for compressed sensing
- Combinations of linear operators; $A = MB$, $B = [D, I]$
- Developed operator toolbox

```
D = opDCT(256);
I = opEye(256);
B = opDictionary(D,I);
M = ...;
A = opFoG(M,B);
```

- Application of operator (function)

```
y = A(x,1);
x = A(y,2);
C = opClass(A);  % Rudimentary class wrapper
y = C * x;
x = C' * y;
```

Spot

- Leverages classes in Matlab
- Factors out the operator toolbox part from Sparco
- Largely improved in usability, flexibility
- Main purpose: rapid prototyping

Aim of Spot

- Combine the advantages of functions and matrices
- Keep the matrix feel in operators

```
>                    > D = opDCT(16);    > D = dct(eye(16));
> y = dct(x);        > y = D  * x;       > y = D  * x;
> z = idct(y);       > z = D' * y;       > z = D' * y;
```

Aim of Spot

- Combine the advantages of functions and matrices
- Keep the matrix feel in operators

```
>                          > D = opDCT(16);    > D = dct(eye(16));
> y = dct(x);              > y = D  * x;       > y = D  * x;
> z = idct(y);             > z = D' * y;       > z = D' * y;
```

Advantage

- Fast operation
- Low memory requirements

Disadvantage

- Cannot pass to functions
- Difficult to manipulate

Aim of Spot

- Combine the advantages of functions and matrices
- Keep the matrix feel in operators

```
>                    > D = opDCT(16);    > D = dct(eye(16));
> y = dct(x);        > y = D  * x;       > y = D  * x;
> z = idct(y);       > z = D' * y;       > z = D' * y;
```

Disadvantage

- Does not scale well

Advantage

- Can pass to function
- Easy to manipulate

Aim of Spot

- Combine the advantages of functions and matrices
- Keep the matrix feel in operators

```
>                       > D = opDCT(16);      > D = dct(eye(16));
> y = dct(x);           > y = D   * x;        > y = D   * x;
> z = idct(y);          > z = D'  * y;        > z = D'  * y;
```

Advantage

- Fast operation
- Low memory requirements

Advantage

- Can pass to function
- Easy to manipulate

Outline

1. Instantiate elementary Spot operators
2. Manipulate and combine operators
3. Apply operators
4. Adding operators

Outline

1. Instantiate elementary Spot operators
2. Manipulate and combine operators
3. Apply operators
4. Adding operators

Applying operators

- Spot operators designed to work on vectors
- All data is vectorized
- Operators internally reshape data if needed (e.g., 2D-DFT)

Applying operators

- Spot operators designed to work on vectors
- All data is vectorized
- Operators internally reshape data if needed (e.g., 2D-DFT)

- Application is similar to matrix-vector products

```
y = D * x;
```

`y'*D` $\longrightarrow$ `(D' * y)'`

- Operator-matrix products by repeated application

```
Y = D * X;
```

Applying operators

- Spot operators designed to work on vectors
- All data is vectorized
- Operators internally reshape data if needed (e.g., 2D-DFT)

- Application is similar to matrix-vector products*

```
y = D * x;
```

`y'*D` $\longrightarrow$ `(D' * y)'`

- Operator-matrix products by repeated application

```
Y = D * X;
```

Non-linear operators are possible but not recommended!

Basic operators

Elementary

- opEye, opZeros, opOnes, opDiag
- opMatrix
- opGaussian, opBinary

Basic operators

Elementary

- opEye, opZeros, opOnes, opDiag
- opMatrix
- opGaussian, opBinary

Fast transforms

- opCurvelet, opSurfacelet
- opDCT, opDCT2, opDFT, opDFT2, opWavelet

```
D = opDFT2(m,n); % 2D discrete Fourier transform
```

Basic operators

Elementary

- opEye, opZeros, opOnes, opDiag
- opMatrix
- opGaussian, opBinary

Fast transforms

- opCurvelet, opSurfacelet
- opDCT, opDCT2, opDFT, opDFT2, opWavelet

```
D = opDFT2(m,n); % 2D discrete Fourier transform
```

- opHeaviside, opHadamard, opToeplitz
- opConvolve (regular, truncated, cyclic)

```
C = opConvolve(m,n,kernel,offset,type);
```

Meta operators (1/5)

Multiplication

```
C = B * A;      % Overloaded *:  C = opFoG(B,A);
C = 3 * A;
C = -B;
```

Meta operators (1/5)

Multiplication

```
C = B * A;      % Overloaded *:  C = opFoG(B,A);
C = 3 * A;
C = -B;
```

Addition

```
x = (B + C + D) * y;
A = B + C + D;
x = A * y;      % Equivalent to first statement
C = M + A;
```

Meta operators (1/5)

Multiplication

```
C = B * A;      % Overloaded *:  C = opFoG(B,A);
C = 3 * A;
C = -B;
```

Addition

```
x = (B + C + D) * y;
A = B + C + D;
x = A * y;      % Equivalent to first statement
C = M + A;
```

Transposition and conjugation

```
A = B';
A = B.';
A = conj(B);
```

Meta operators (2/5)

Dictionaries

```
A = [B, C, M];   % Horizontal concatenation
A = [B; C; M];   % Vertical concatenation
A = [B, C; C', D];
```

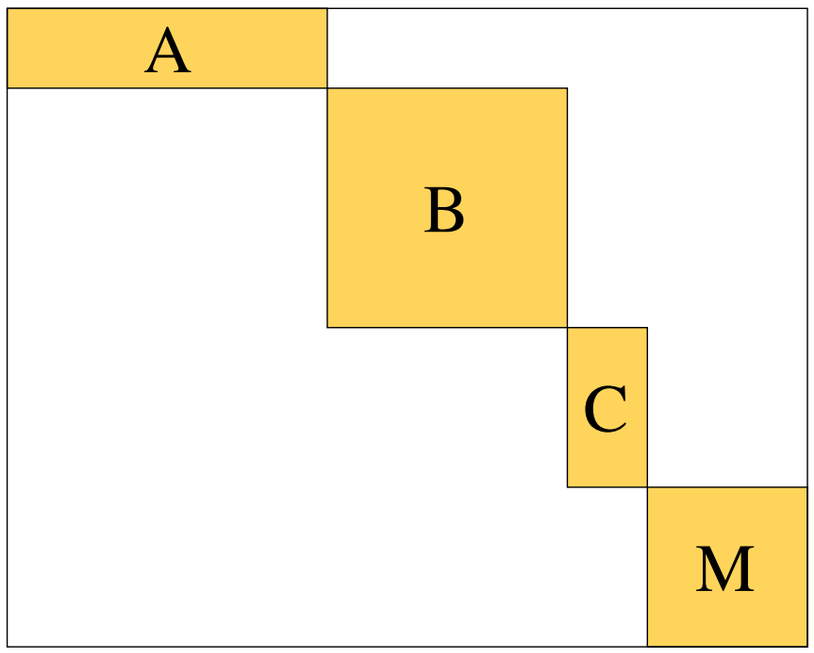
Meta operators (2/5)

Dictionaries

```
A = [B, C, M];   % Horizontal concatenation
A = [B; C; M];   % Vertical concatenation
A = [B, C; C', D];
```

Block diagonal

```
D = blkdiag(A,B,C,M);
D = opBlockDiag([weight],op1,...,opn,[overlap]);
```



blkdiag

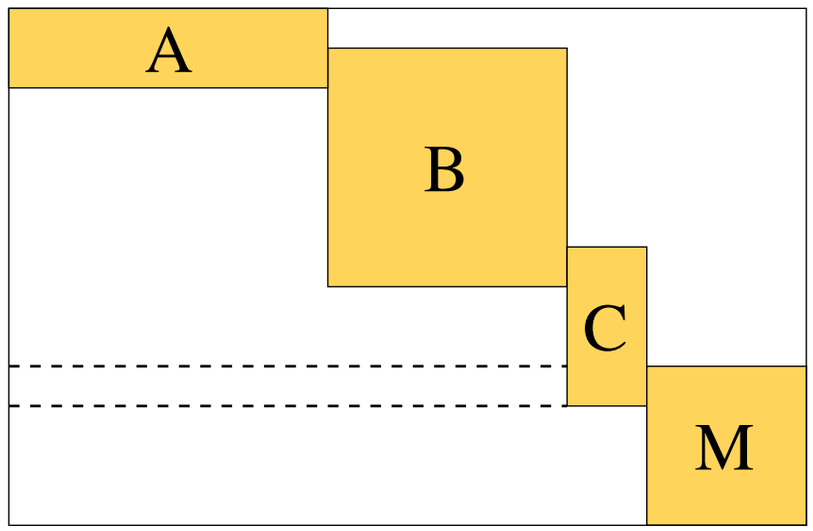
Meta operators (2/5)

Dictionaries

```
A = [B, C, M];   % Horizontal concatenation
A = [B; C; M];   % Vertical concatenation
A = [B, C; C', D];
```

Block diagonal

```
D = blkdiag(A,B,C,M);
D = opBlockDiag([weight],op1,...,opn,[overlap]);
```



`overlap > 0`

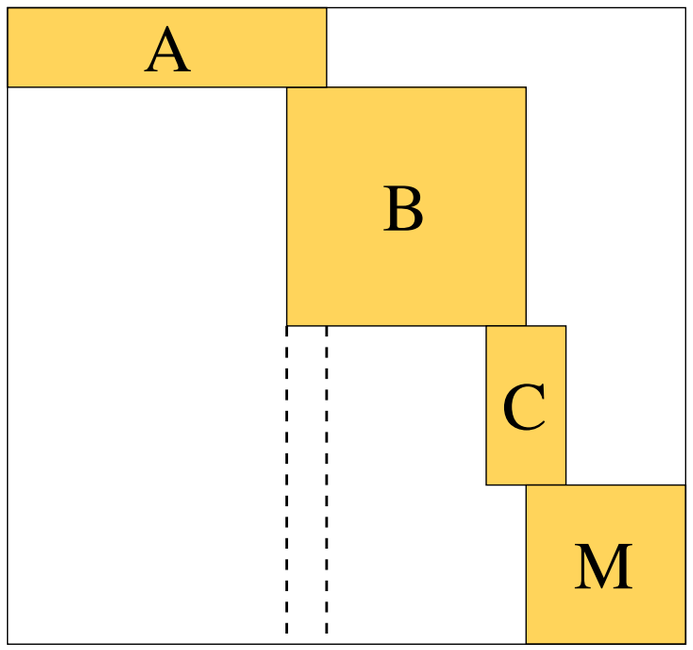
Meta operators (2/5)

Dictionaries

```
A = [B, C, M];   % Horizontal concatenation
A = [B; C; M];   % Vertical concatenation
A = [B, C; C', D];
```

Block diagonal

```
D = blkdiag(A,B,C,M);
D = opBlockDiag([weight],op1,...,opn,[overlap]);
```



`overlap < 0`

Meta operators (3/5)

Kronecker products

```
A = kron(A,B,C);   % A := A ⊗ B ⊗ C
```

Meta operators (3/5)

Kronecker products

```
A = kron(A,B,C);   % A := A ⊗ B ⊗ C
```

Example: n^3 data volume

```
F  = opDFT(n);
I  = opEye(n);
A1 = kron(I,I,F); % DFT along first dimension
A2 = kron(I,F,I); % DFT along second dimension
A3 = kron(F,I,I); % DFT along third dimension
```

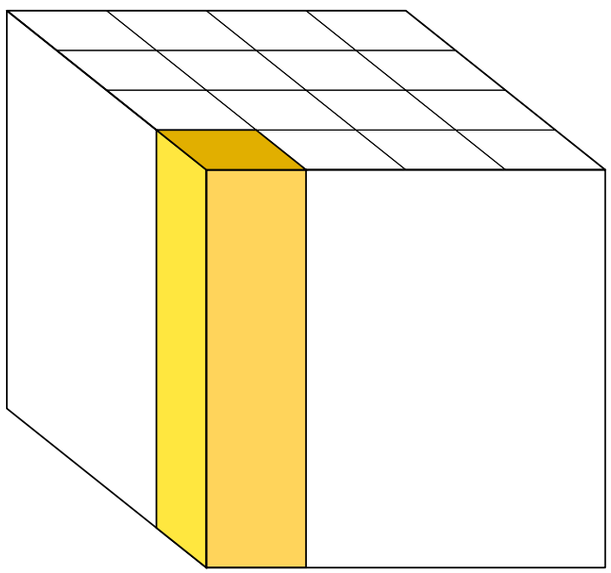
Meta operators (3/5)

Kronecker products

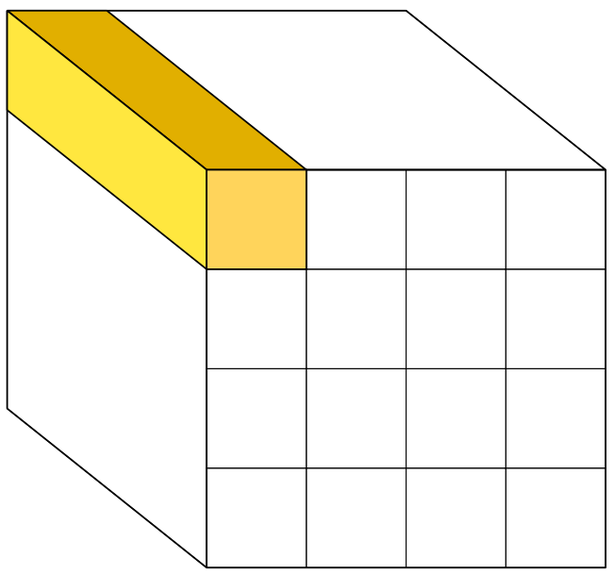
```
A = kron(A,B,C);   % A := A ⊗ B ⊗ C
```

Example: n^3 data volume

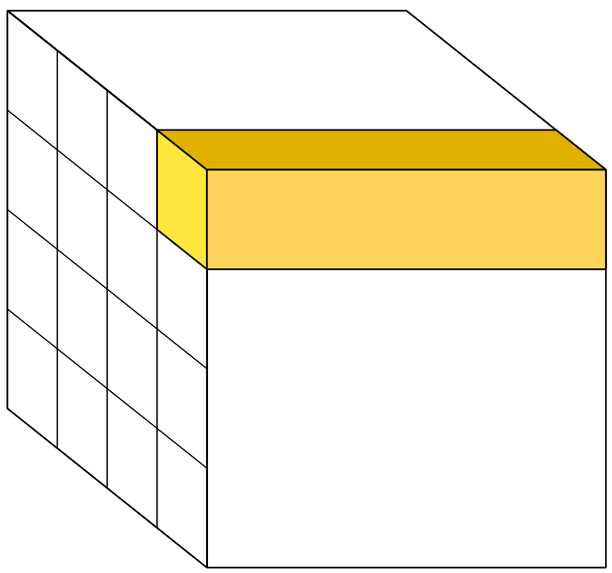
```
F  = opDFT(n);
I  = opEye(n);
A1 = kron(I,I,F); % DFT along first dimension
A2 = kron(I,F,I); % DFT along second dimension
A3 = kron(F,I,I); % DFT along third dimension
```



A1



A2



A3

Meta operators (4/5)

Slicing

```
A = B(:,idx);
A = B(idx,:);
A = B(idx1,idx2);
```

Meta operators (4/5)

Slicing

```
A = B(:,idx);
A = B(idx,:);
A = B(idx1,idx2);
```

Example: Randomly restricted Fourier operator

```
F = opDFT(128);
p = randperm(128);
A = F(p(1:50),:);
```

Meta operators (4/5)

Slicing

```
A = B(:,idx);
A = B(idx,:);
A = B(idx1,idx2);
```

Example: Randomly restricted Fourier operator

```
F = opDFT(128);
p = randperm(128);
A = F(p(1:50),:);
```

Assignment

```
A(idx1,idx2) = B;
A(5,:) = []; % Cut the fifth row
A(:,5) = []; % Cut the fifth column
```

Meta operators (5/5)

Pseudo-inverse and backslash

```
x = A \ b;
P = pinv(A);
x = P * b;
```

Overdetermined system

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2$$

Underdetermined system

$$\underset{x}{\text{minimize}} \quad \|x\|_2 \quad \text{s.t.} \quad Ax = b$$

Linear systems are solved using LSQR

Tools

Querying operator information

- `size`
- `disp`
- `isempty`

Counter facilities

- Counters associated with each operator
- Keeps track of number of applications

Adding operators

Function wrapper

```
y = fun(x,1);
x = fun(y,2);
F = opFunction(m,n,fun,cflag);
y = F * x;
x = F'* y
```

Adding operators

Function wrapper

```
y = fun(x,1);
x = fun(y,2);
F = opFunction(m,n,fun,cflag);
y = F * x;
x = F'* y
```

Class wrapper

```
C = opClass(m,n,obj,cflag,linflag);
```

Adding operators

Function wrapper

```
y = fun(x,1);
x = fun(y,2);
F = opFunction(m,n,fun,cflag);
y = F * x;
x = F'* y
```

Class wrapper

```
C = opClass(m,n,obj,cflag,linflag);
```

Deriving a child class

- Inherit from base class (`opSpot`)
- Write constructor
- Implement `multiply(op,x,mode)` function
- Optionally: overload other operations

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