Tools and techniques for sparse optimization and beyond

Ewout van den Berg

Human Language Technologies

IBM T.J. Watson Research Center, Yorktown Heights, NY

TCMM 2014, Leuven, Belgium

September 8, 2014

Overview

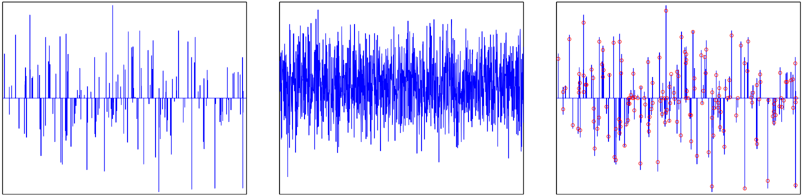
1. SPGL1 – A solver for sparse optimization
2. Spot – A linear operator toolbox for Matlab
3. DMC – A framework for disciplined massive computing

SPGL1 – A solver for sparse optimization

Joint work with Michael Friedlander at the
Department of Computer Science, UBC, Canada

Sparse recovery and compressed sensing

Compressed sensing



x $\qquad$ $b = Ax + n$ $\qquad$ $\hat{x}$

- x: 1% sparse vector
- A: randomly subsampled $1,278 \times 16,384$ DCT
- n: 10% additive noise
- Solve basis pursuit denoise formulation

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

Sparse recovery (deblurring)

$v = W^T x$

$b = Bv$

$\hat{v} = W^T \hat{x}$

- W: two-dimensional Haar wavelet transform
- B: blurring operator
- $A = BW^T$

$$\underset{x}{\text{minimize}} \quad \|x\|_1 \quad \text{subject to} \quad \|Ax - b\|_2 \le \sigma$$

Generalized sparse recovery

- Basis pursuit denoise (sparsity)

$$\underset{x}{\text{minimize}} \quad \|x\|_1 \quad \text{subject to} \quad \|Ax - b\|_2 \le \sigma$$

Generalized sparse recovery

- Basis pursuit denoise (sparsity)

$$\underset{x}{\text{minimize}} \quad \|x\|_1 \quad \text{subject to} \quad \|Ax - b\|_2 \le \sigma$$

- Multiple measurement vectors (row/column sparsity)

$$\underset{X}{\text{minimize}} \quad \|X\|_{1,2} \quad \text{subject to} \quad \|AX - B\|_F \le \sigma$$

Generalized sparse recovery

- Basis pursuit denoise (sparsity)

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

- Multiple measurement vectors (row/column sparsity)

$$\underset{X}{\text{minimize}} \quad \|X\|_{1,2} \quad \text{subject to} \quad \|AX - B\|_F \leq \sigma$$

- Nuclear-norm minimization (low rank)

$$\underset{X}{\text{minimize}} \quad \|X\|_* \quad \text{subject to} \quad \|\mathcal{P}_\mathcal{I} X - B\|_2 \leq \sigma$$

Generalized sparse recovery

- Basis pursuit denoise (sparsity)

$$\underset{x}{\text{minimize}} \quad \|x\|_1 \quad \text{subject to} \quad \|Ax - b\|_2 \le \sigma$$

- Multiple measurement vectors (row/column sparsity)

$$\underset{X}{\text{minimize}} \quad \|X\|_{1,2} \quad \text{subject to} \quad \|AX - B\|_F \le \sigma$$

- Nuclear-norm minimization (low rank)

$$\underset{X}{\text{minimize}} \quad \|X\|_* \quad \text{subject to} \quad \|\mathcal{P}_{\mathcal{I}}X - B\|_2 \le \sigma$$

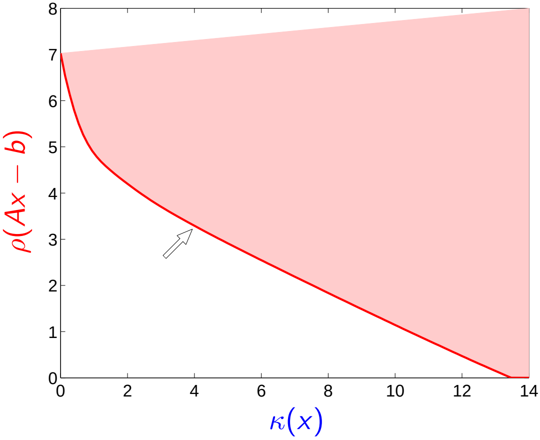
Generalized formulation

$$\underset{x}{\text{minimize}} \quad \kappa(x) \quad \text{subject to} \quad \rho(Ax - b) \le \sigma$$

Typically: ρ smooth, κ non-smooth

Pareto curve

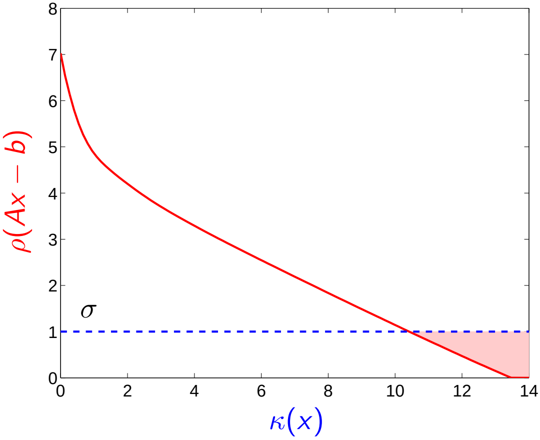
$$\underset{x}{\text{minimize}} \quad \kappa(x) \quad \text{subject to} \quad \rho(Ax - b) \leq \sigma$$



- Pareto curve: Trade-off between $\kappa(x)$ and $\rho(Ax - b)$
- Feasibility: $\rho(Ax - b) \leq \sigma$

Pareto curve

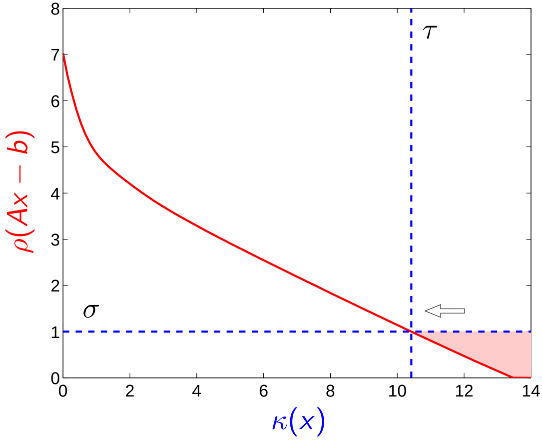
$$\underset{x}{\text{minimize}} \quad \kappa(x) \quad \text{subject to} \quad \rho(Ax-b) \le \sigma$$



- Pareto curve: Trade-off between $\kappa(x)$ and $\rho(Ax-b)$
- Feasibility: $\rho(Ax-b) \le \sigma$
- Minimize $\kappa(x)$

Pareto curve

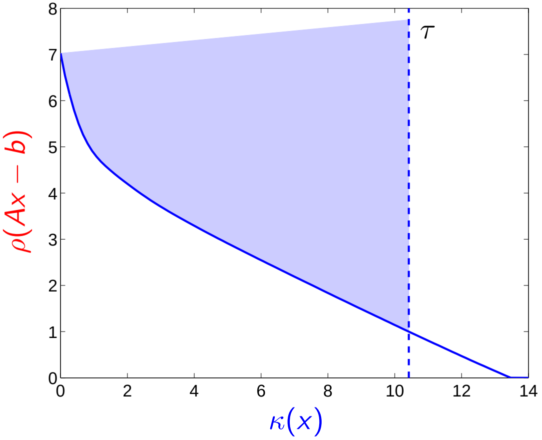
$$\underset{x}{\text{minimize}} \quad \kappa(x) \quad \text{subject to} \quad \rho(Ax - b) \le \sigma$$



- Pareto curve: Trade-off between $\kappa(x)$ and $\rho(Ax - b)$
- Feasibility: $\rho(Ax - b) \le \sigma$
- Minimize $\kappa(x)$
- Difficulty: κ is nonsmooth, constraints are not simple
- Solution: change formulation

Pareto curve

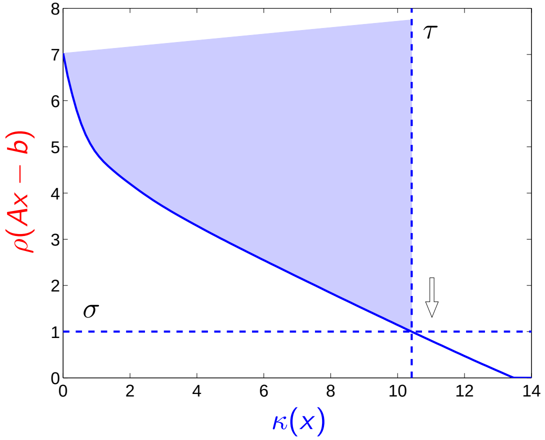
$$\underset{x}{\text{minimize}} \quad \rho(Ax - b) \quad \text{subject to} \quad \kappa(x) \leq \tau$$



- Smooth objective, simple constraints
- Minimize $\rho(Ax - b)$

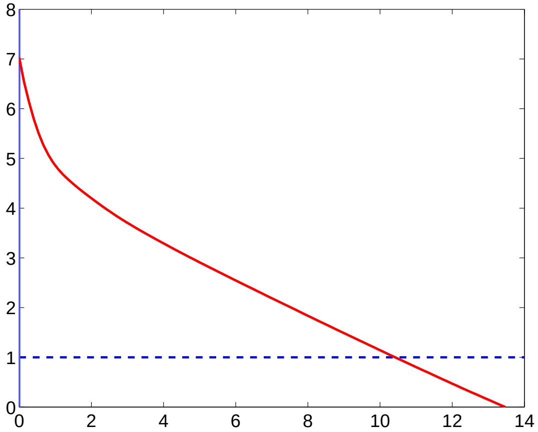
Pareto curve

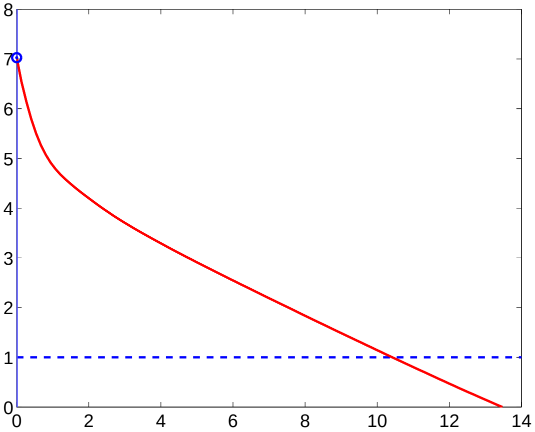
$$\underset{x}{\text{minimize}} \quad \rho(Ax - b) \quad \text{subject to} \quad \kappa(x) \leq \tau$$



- Smooth objective, simple constraints
- Minimize $\rho(Ax - b)$
- Main difficulty: Given σ, how to find τ?

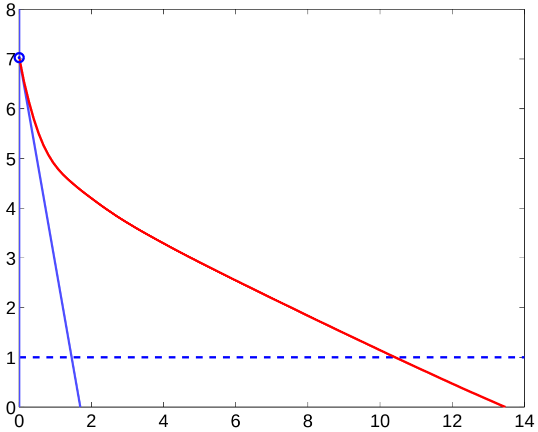
Root-finding





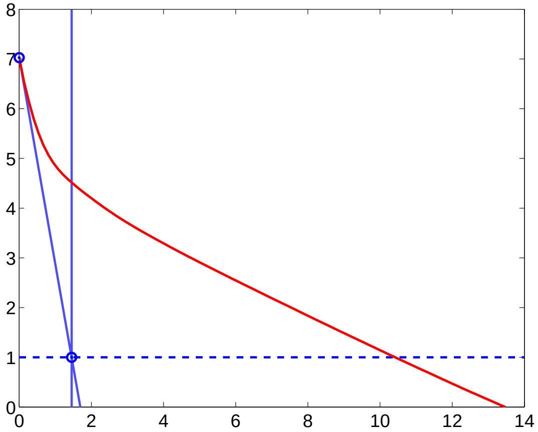
1. $\phi(\tau)$: minimize $\rho(Ax - b)$ subj to $\kappa(x) \leq \tau$ (approx.)

Root-finding



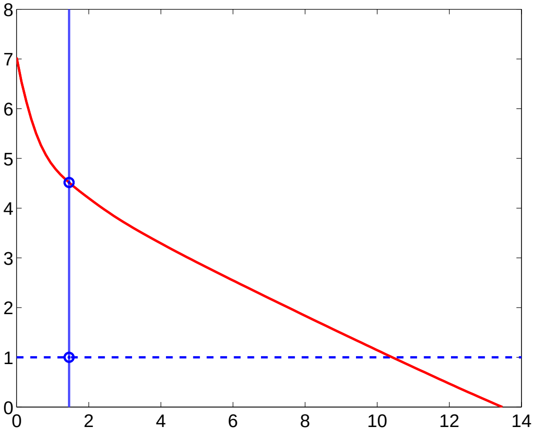
1. $\phi(\tau)$: minimize $\rho(Ax - b)$ subj to $\kappa(x) \leq \tau$ (approx.)
2. Gradient $\phi'(\tau)$: duality

Root-finding



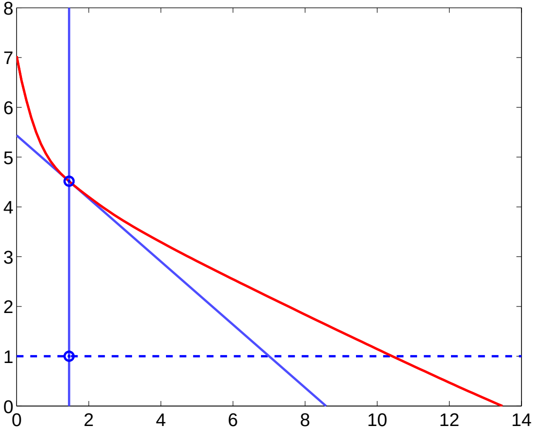
1. $\phi(\tau)$: minimize $\rho(Ax - b)$ subj to $\kappa(x) \leq \tau$ (approx.)
2. Gradient $\phi'(\tau)$: duality
3. Update τ

Root-finding



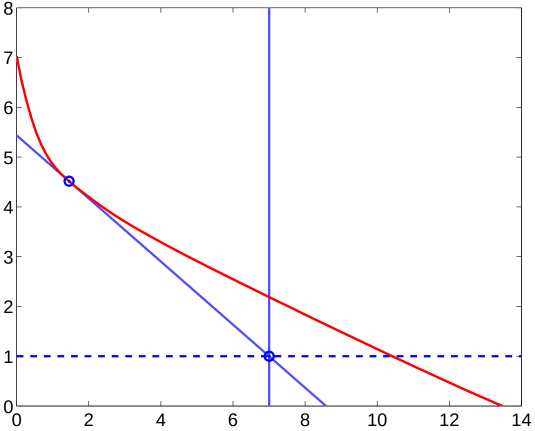
1. $\phi(\tau)$: minimize $\quad \rho(Ax - b) \quad$ subj to $\quad \kappa(x) \leq \tau$ (approx.)
2. Gradient $\phi'(\tau)$: duality
3. Update τ

Root-finding



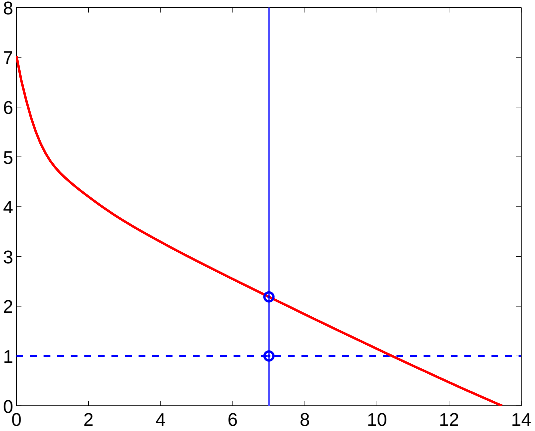
1. $\phi(\tau)$: minimize $\quad \rho(Ax - b) \quad$ subj to $\quad \kappa(x) \leq \tau \qquad$ (approx.)
2. Gradient $\phi'(\tau)$: duality
3. Update τ

Root-finding



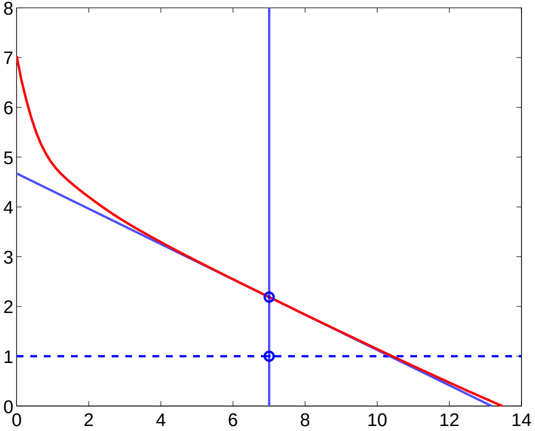
1. $\phi(\tau)$: minimize $\rho(Ax - b)$ subj to $\kappa(x) \leq \tau$ (approx.)
2. Gradient $\phi'(\tau)$: duality
3. Update τ

Root-finding



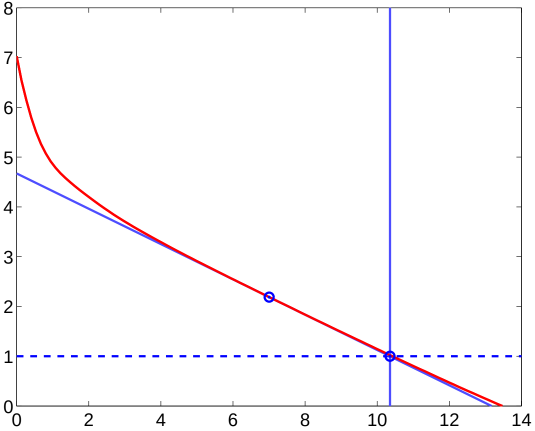
1. $\phi(\tau)$: minimize $\rho(Ax - b)$ subj to $\kappa(x) \leq \tau$ (approx.)
2. Gradient $\phi'(\tau)$: duality
3. Update τ

Root-finding



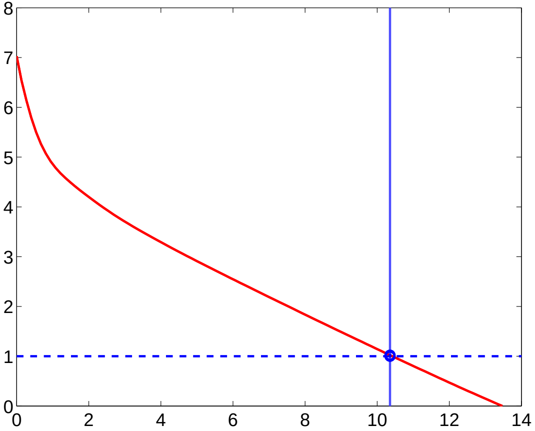
1. $\phi(\tau)$: minimize $\quad \rho(Ax - b) \quad$ subj to $\quad \kappa(x) \leq \tau \qquad$ (approx.)
2. Gradient $\phi'(\tau)$: duality
3. Update τ

Root-finding



1. $\phi(\tau)$: minimize $\rho(Ax - b)$ subj to $\kappa(x) \leq \tau$ (approx.)
2. Gradient $\phi'(\tau)$: duality
3. Update τ

Root-finding



1. $\phi(\tau)$: minimize $\rho(Ax - b)$ subj to $\kappa(x) \leq \tau$ (approx.)
2. Gradient $\phi'(\tau)$: duality
3. Update τ

Original problem:

$$\underset{x}{\text{minimize}} \quad \kappa(x) \quad \text{subject to} \quad \rho(Ax - b) \le \sigma$$

Ingredients:

1. Solver for generalized Lasso problem:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \rho(Ax - b) \quad \text{subject to} \quad \kappa(x) \le \tau.$$

2. Differentiability of Pareto curve $\phi(\tau)$
3. Practical way of evaluating gradient $\phi'(\tau)$

Original problem:

$$\underset{x}{\text{minimize}} \quad \kappa(x) \quad \text{subject to} \quad \rho(Ax - b) \leq \sigma$$

Ingredients:

1. Solver for generalized Lasso problem:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \rho(Ax - b) \quad \text{subject to} \quad \kappa(x) \leq \tau.$$

2. Differentiability of Pareto curve $\phi(\tau)$
3. Practical way of evaluating gradient $\phi'(\tau)$

Differentiability of $\phi(\tau)$ with $\rho = \|\cdot\|_2$

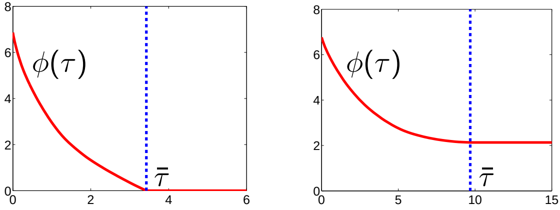
Theorem

Let κ be a gauge function and $\rho(r) = \|r\|_2$. Then $\phi(\tau)$ is convex and differentiable for $0 < \tau < \bar{\tau}$ with

$$\phi'(\tau) = -\kappa^\circ(A^T r_\tau)/\|r_\tau\|_2,$$

where $r_\tau = b - Ax_\tau$ and $x_\tau = \text{argmin}\ \phi(\tau)$.

Definition $\bar{\tau}$:



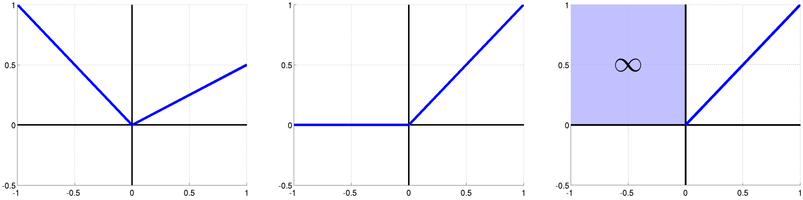
Theorem

Let κ be a gauge function and $\rho(r) = \|r\|_2$. Then $\phi(\tau)$ is convex and differentiable for $0 < \tau < \bar{\tau}$ with

$$\phi'(\tau) = -\kappa^\circ(A^T r_\tau)/\|r_\tau\|_2,$$

where $r_\tau = b - Ax_\tau$ and $x_\tau = \text{argmin}\ \phi(\tau)$.

Gauge functions:



nonnegative convex, $f(0) = 0$, $f(\alpha x) = \alpha f(x)$, $\alpha > 0$

Differentiability of $\phi(\tau)$ with $\rho = \|\cdot\|_2$

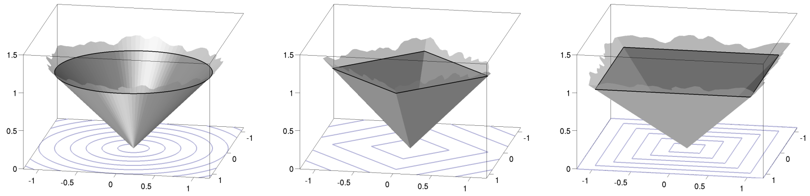
Theorem

Let κ be a gauge function and $\rho(r) = \|r\|_2$. Then $\phi(\tau)$ is convex and differentiable for $0 < \tau < \bar{\tau}$ with

$$\phi'(\tau) = -\kappa^\circ(A^T r_\tau)/\|r_\tau\|_2,$$

where $r_\tau = b - Ax_\tau$ and $x_\tau = \text{argmin } \phi(\tau)$.

Gauge functions: includes norms (polar: dual norm)



ℓ_2 ℓ_1 ℓ_∞

Differentiability of $\phi(\tau)$

Theorem

Let κ and ρ be gauge functions such that $\rho(r)$ is differentiable whenever $r \neq 0$.

Then $\phi(\tau)$ is continuously differentiable for $0 < \tau < \bar{\tau}$, with

$$\phi'(\tau) = -\kappa^\circ(A^T y_\tau),$$

and

$$y_\tau := \operatorname*{argmax}_y \quad b^T y - \tau\kappa^\circ(A^T y) \quad \text{subject to} \quad \rho^\circ(y) \leq 1.$$

Proof ingredients:

- Gradients, subgradients [Rockafellar '70]
- Subgradients of $\phi(\tau)$ and Lagrange multipliers [Bertsekas et al. '03]
- Derivation of dual using conjugate functions (cond'n gauge)
- Uniqueness of y_τ or $\lambda_\lambda := \kappa^\circ(A^T y_\tau)$ (cond'n ρ)

Framework ingredients

1. Solver for generalized Lasso problem:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \leq \tau$$

2. Differentiability of Pareto curve $\phi(\tau)$ ✓
3. Practical way of evaluating gradient $\phi'(\tau)$ ✓

Solving generalized Lasso for $\rho = \|\cdot\|_2$

Different methods for solving the subproblem for $\phi(x)$:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \leq \tau$$

Solving generalized Lasso for $\rho = \|\cdot\|_2$

Different methods for solving the subproblem for $\phi(x)$:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \leq \tau$$

1. **Non-monotone spectrally projected gradients**

   Initialize x, scaling factor γ
   While not converged
   $\quad \nabla f = A^T(Ax - b)$
   $\quad$ Linesearch in α:
   $\quad\quad x^{new} \leftarrow \mathcal{P}(x - \gamma\alpha\nabla f(x))$ **-or-**
   $\quad\quad x^{new} \leftarrow x + \alpha(\mathcal{P}(x - \gamma\nabla f(x)) - x)$
   $\quad$ Update γ, x
   End while

Solving generalized Lasso for $\rho = \|\cdot\|_2$

Different methods for solving the subproblem for $\phi(x)$:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \le \tau$$

1. Non-monotone spectrally projected gradients
2. **Projected quasi-Newton**

   Initialize x, quadratic model Q

   While not converged

       Minimize $Q(x)$ subject to $\|x\|_1 \le \tau$

       Evaluate $f(x) = \frac{1}{2}\|Ax - b\|$, and $\nabla f(x)$

       Update quadratic model Q

   End while

Solving generalized Lasso for $\rho = \|\cdot\|_2$

Different methods for solving the subproblem for $\phi(x)$:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \le \tau$$

1. Non-monotone spectrally projected gradients
2. Projected quasi-Newton
3. Nesterov's algorithm PARNES (Gu, Lim, & Wu)

While not converged

$$y_k = \underset{y \in \mathcal{Q}}{\text{argmin}} \ \nabla f(x_k)^T(y - x_k) + \frac{L}{2}\|y - x_k\|_2^2$$

$$z_k = \underset{z \in \mathcal{Q}}{\text{argmin}} \ \sum_{i=0}^{k} \frac{i+1}{2}[f(x_i) + \nabla f(x_i)^T(z - x_i)] + \frac{L}{2}\|z - c\|_2^2$$

$$x_k = \tfrac{2}{k+3} z_k + \tfrac{k+1}{k+3} y_k$$

End while

Solving generalized Lasso for $\rho = \|\cdot\|_2$

Different methods for solving the subproblem for $\phi(x)$:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \le \tau$$

1. Non-monotone spectrally projected gradients
2. Projected quasi-Newton
3. Nesterov's algorithm (Gu, Lim, & Wu)
4. Hybrid quasi-Newton method (ℓ_1)

Solving generalized Lasso for $\rho = \|\cdot\|_2$

Different methods for solving the subproblem for $\phi(x)$:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \le \tau$$

1. Non-monotone spectrally projected gradients
2. Projected quasi-Newton
3. Nesterov's algorithm (Gu, Lim, & Wu)
4. Hybrid quasi-Newton method (ℓ_1)

Solver ingredients:

- Products with A and A^T (implicit)
- Evaluation of κ and κ°
- Projection onto $\mathcal{B}_\tau := \{x \mid \kappa(x) \le \tau\}$

Solving generalized Lasso for $\rho = \|\cdot\|_2$

Different methods for solving the subproblem for $\phi(x)$:

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \le \tau$$

1. Non-monotone spectrally projected gradients
2. Projected quasi-Newton
3. Nesterov's algorithm (Gu, Lim, & Wu)
4. Hybrid quasi-Newton method (ℓ_1)

Solver ingredients:

- Products with A and A^T (implicit)
- Evaluation of κ and κ°
- Projection onto $\mathcal{B}_\tau := \{x \mid \kappa(x) \le \tau\}$

Projection onto feasible set, $\mathcal{B}_\tau$

Setting:

- Basis pursuit denoise: $\kappa(x) = \|x\|_1$

ℓ_1 projection

$\mathcal{O}(n \log n)$ or randomized $\mathcal{O}(n)$ [Duchi et al. '08, van den Berg et al. '08]

Projection onto feasible set, $\mathcal{B}_\tau$

Setting:

- Basis pursuit denoise: $\kappa(x) = \|x\|_1$
- Group sparsity/MMV: $\kappa(x) = \sum_i \|x_{(i)}\|_2$

$\ell_{1,2}$ projection

1. Set $v_i := \|x_{(i)}\|_2$
2. $u \leftarrow$ projection of v onto $\mathcal{B}_\tau(\ell_1)$
3. Result p: $p_{(i)} := (u_i/v_i) \cdot x_{(i)}$.

Projection onto feasible set, $\mathcal{B}_\tau$

Setting:

- Basis pursuit denoise: $\kappa(x) = \|x\|_1$
- Group sparsity/MMV: $\kappa(x) = \sum_i \|x_{(i)}\|_2$
- Nuclear norm minimization: $\kappa(X) = \|X\|_*$

Nuclear norm projection

1. Factorize $X = U \cdot diag(s) V^T$.
2. $z \leftarrow$ projection of s onto $\mathcal{B}_\tau(\ell_1)$
3. Result $P = U \cdot diag(z) V^T$.

Follows directly from [Cai et al., Ma et al.]

Framework ingredients

1. Solver for generalized Lasso problem: ✓

$$\phi(\tau) := \underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \kappa(x) \leq \tau$$

2. Differentiability of Pareto curve $\phi(\tau)$ ✓
3. Practical way of evaluating gradient $\phi'(\tau)$ ✓

Software – SPGL1

Framework implemented in Matlab [van den Berg, Friedlander '08]

- Subproblem solvers
  - Nonmonotone spectrally projected gradient
  - Projected quasi-Newton (PQN)
  - Hybrid quasi-Newton method (work in progress)
- Formulations
  - Basis pursuit denoise
  - Joint-sparsity and MMV
  - Sign-restricted versions
  - Nuclear norm minimization (preliminary)

SPGL1 freely available on-line

http://www.cs.ubc.ca/labs/scl/spgl1/

Spot – A linear-operator toolbox for Matlab

Joint work with Michael Friedlander, Gilles Hennenfent,
Felix Herrmann, Rayan Saab, and Özgur Yılmaz, UBC, Canada

Introduction

- Many structured linear operators A in compressed sensing
- Fast, implicit representation often available
- Requires function calls and we lose flexibility of matrices:

```
>                        > 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;
```

Introduction

- Many structured linear operators A in compressed sensing
- Fast, implicit representation often available
- Requires function calls and we lose flexibility of matrices:

```
>                    > 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

- Many structured linear operators A in compressed sensing
- Fast, implicit representation often available
- Requires function calls and we lose flexibility of matrices:

```
>                       > 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

Introduction

- Many structured linear operators A in compressed sensing
- Fast, implicit representation often available
- Requires function calls and we lose flexibility of matrices:

```
>
> y = dct(x);
> z = idct(y);
```

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

```
> D = dct(eye(16));
> y = D  * x;
> 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. Implementing operators

Outline

1. Instantiate elementary Spot operators
2. Manipulate and combine operators
3. Apply operators
4. Implementing 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` $\rightarrow$ `(D' * y)'`

Operator-matrix products by repeated application

`Y = D * X;`

- Querying operator information: `size`, `disp`, `isempty`
- Operator counter keeps track of number of applications

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;
```

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;
```

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];
```

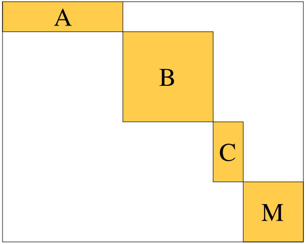
Meta operators (2/5)

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]);
```



Meta operators (3/5)

Kronecker products

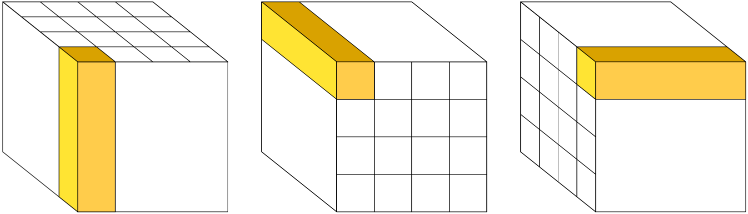
```
A = kron(A,B,C);   % A := A ⊗ B ⊗ C
```

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
```

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

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
```

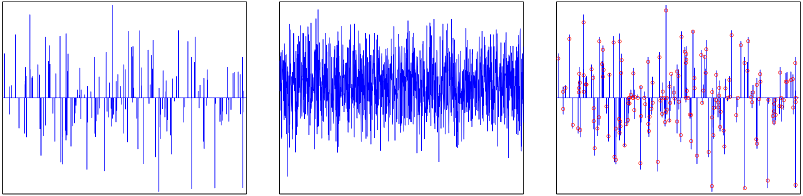
Class wrapper

```
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

Compressed sensing #1

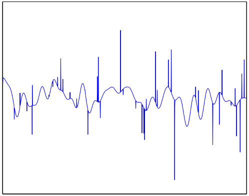


x $\qquad$ $b = Ax + n$ $\qquad$ $\hat{x}$

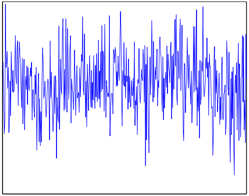
```
D = opDCT(n)
A = D(idx,:)
xhat = spgl1(A,b,[],sigma,[],options)
```

Examples

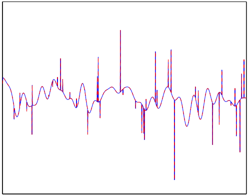
Compressed sensing #2



$v = Bx,\ B = [D, I]$



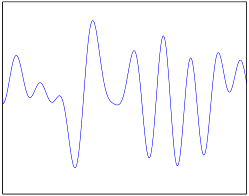
$b = Mv,$



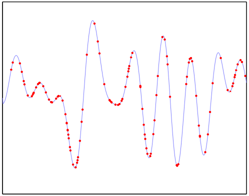
$A = MB,\ \hat{v} = B\hat{x}$

```
D = opDCT(1024)
I = opEye(1024)
B = [D, I]
M = opGaussian(480,1024)   or   M = opMatrix(...)
A = M*B
```

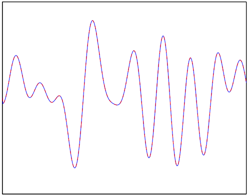
Sparse recovery (inpainting)



$v = D^T x$



$b = Rv,$



$A = RD^T,\ \hat{v} = D^T \hat{x}$

```
D = opDCT(n)
R = opRestriction(n,idx)
A = R*D'    or    A = D(:,idx)'
vhat = D'*xhat
```

Examples

Sparse recovery (deblurring)

$v = W^T x$

$b = Bv,\ A = BW^T$

$\hat{v} = W^T \hat{x}$

```
W = opWavelet(n,n,'Haar')
B = opConvolution(n,n,mask,[16,16],'cyclic')
A = B * W'
vhat = reshape(W' * xhat,n,n)
```

Software – Spot

Spot freely available on-line

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

DMC – A prototype framework for disciplined massive computing

Joint work with David Donoho at the Department of Statistics, Stanford, CA

DMC – Disciplined massive computation framework

- System for setting up/managing structured batches of jobs
- Database containing jobs and results
- Easy-to-use interface designed for expressiveness
- Generation of formatted output based on job batch structure
- A prototype: raises more design questions than answers

Motivation

Current approach:

```
function generateFigure(exportFlag)
prefix_fig = getPrefix('functions')

for param = ...
   data = runExperiment(param)
   ...
   plot(...)
end

if exportFlag
   exportFigurePDF(prefix_fig,'filename.pdf')
end
```

Motivation

Current approach:

```
function data = runExperiment(param)
prefix_cache = getPrefix('cache')

filename = [prefix_cache, generateFilename(param)]
if exist(filename,'filename')
   info = load(filename)
   data = info.data
else
   // Run experiment
   data = ...
   save(filename,'data')
end
```

Motivation

Current approach:

Advantages:

- Reproducible
- Uses precomputed results whenever possible
- Can delete cached files to force recompute

Disadvantages:

- Repetitive pattern for all figures/tables/experiments:
  Tedious to rewrite every time
- No consistency guarantees with respect to code changes
- Parallel computing: submission to platform non-automated
- Intention/goal of experiment is not clear from code

Motivation

Current approach:

```
function generateFigure(exportFlag)
prefix_fig = getPrefix('functions')

for param = ...
   data = runExperiment(param)
   ...
   plot(...)
end

if exportFlag
   exportFigurePDF(prefix_fig,'filename.pdf')
end
```

Ideal system

- High-level description language that is self documenting
- Automatic generation and execution of workflow
- Easy access to results and meta-data in shared database
- Fully reproducible/traceable (provenance)

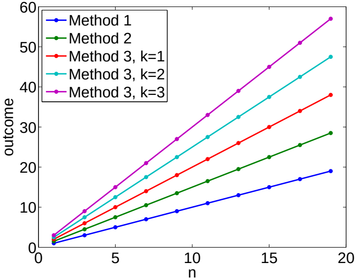
Ideal system

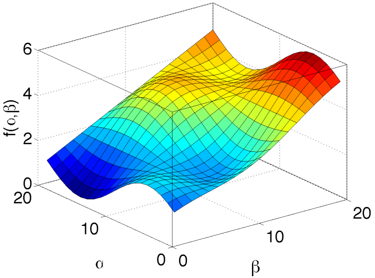
- High-level description language that is self documenting
- Automatic generation and execution of workflow
- Easy access to results and meta-data in shared database
- Fully reproducible/traceable (provenance)

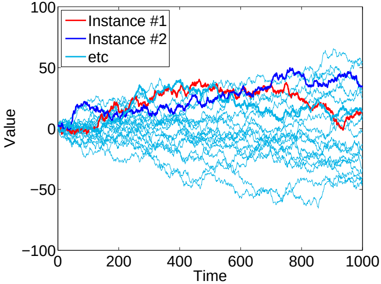
High-level language challenges:

- Need to find the appropriate scope
- Simple yet expressive enough to be useful
- Avoid: One-trick pony, fully-fledged new language
- Example of good balance: CVX

Scope







		Method #1			Method #2		
		a	b	c	a	b	c
1	A	3.1	2.4	9.1	0.1	2.2	3.5
	B	3.4	4.5	3.2	7.4	1.9	2.0
	C	3.3	3.4	6.1	3.6	2.1	2.8
2	A	3.1	2.4	9.1	0.1	2.2	3.5
	B	3.4	4.5	3.2	7.4	1.9	2.0
	C	3.3	3.4	6.1	3.6	2.1	2.8
3	A	3.1	2.4	9.1	0.1	2.2	3.5
	B	3.4	4.5	3.2	7.4	1.9	2.0
	C	3.3	3.4	6.1	3.6	2.1	2.8

Example

- Compare performance of three algorithms on several problems:

```
optionsFun     = dmcOptions(fun1,fun2,fun3)
optionsProblem = dmcOptions('problem',1,2,3)
options = optionsFun * optionsProblem
tickets = project.evaluate(options)
dmcTabulate(tickets,'iterations as iter','runtime');
```

Example

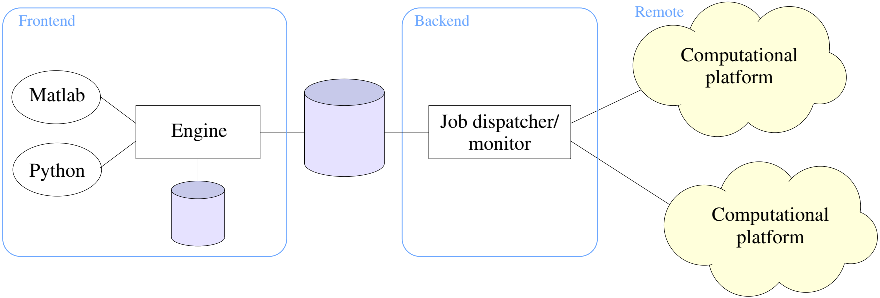
- Compare performance of three algorithms on several problems:

```
optionsFun     = dmcOptions(fun1,fun2,fun3)
optionsProblem = dmcOptions('problem',1,2,3)
options = optionsFun * optionsProblem
tickets = project.evaluate(options)
dmcTabulate(tickets,'iterations as iter','runtime');
```

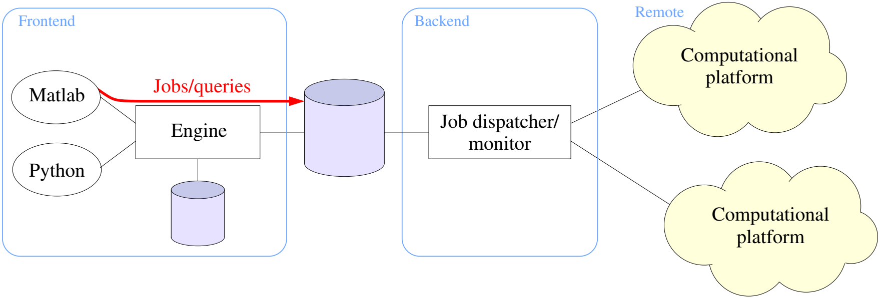
```
-----------------------------------------------------------
function     |                   problem
             |        1        |        2         |       3
             |                 |                  |
             | iter | runtime  | iter | runtime  | iter | runtime
-----------------------------------------------------------
Function #1 |  124 | 7.39374 |   89 | 5.12939 |   92 | 6.98314
Function #2 |  203 | 10.5812 |  170 | 8.26123 |  163 | 8.06182
Function #3 |  157 | 9.69676 |  168 | 10.1924 |  165 | 9.72862
-----------------------------------------------------------
```

System overview

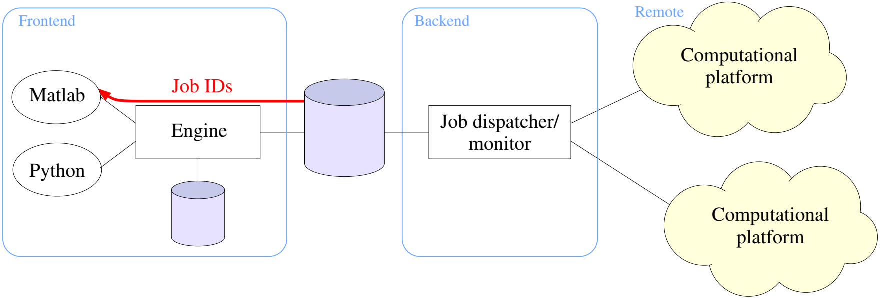
Architecture



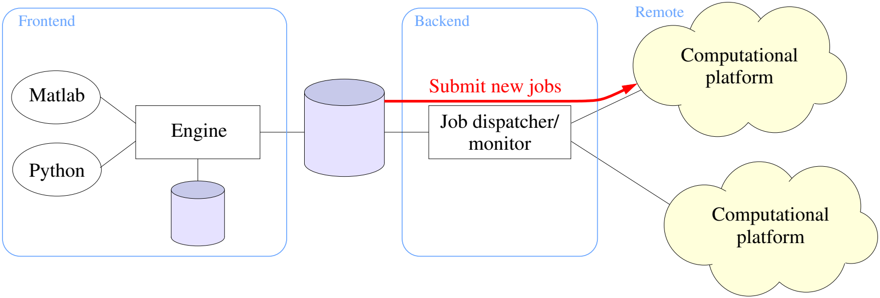
Architecture



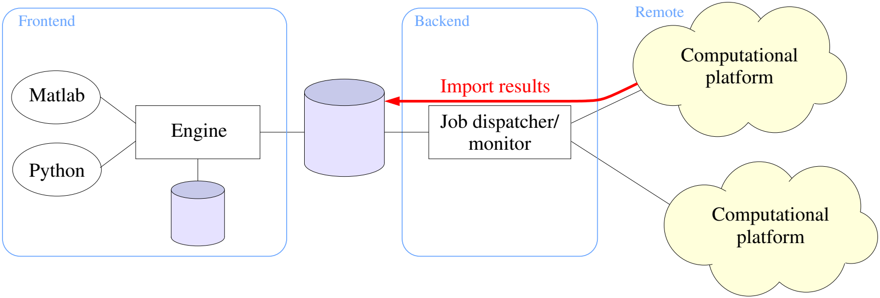
Architecture



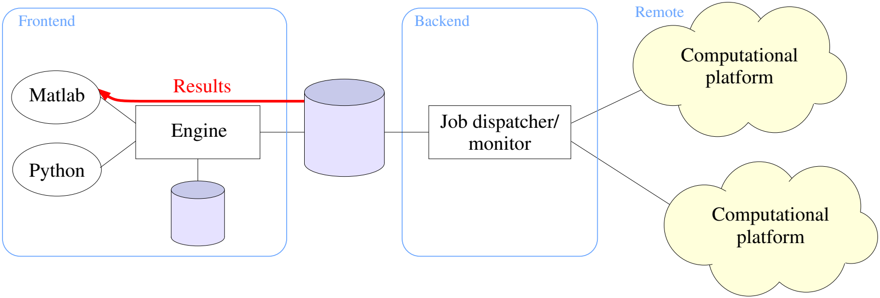
Architecture



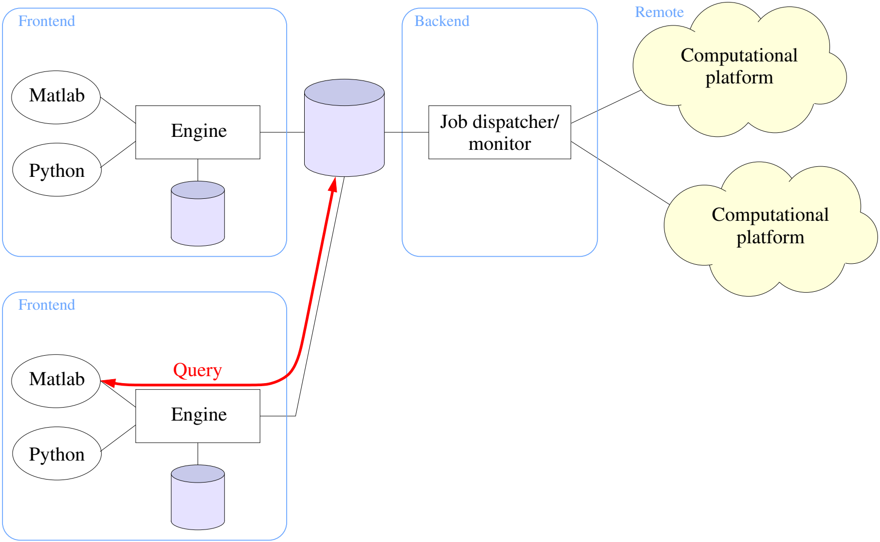
Architecture



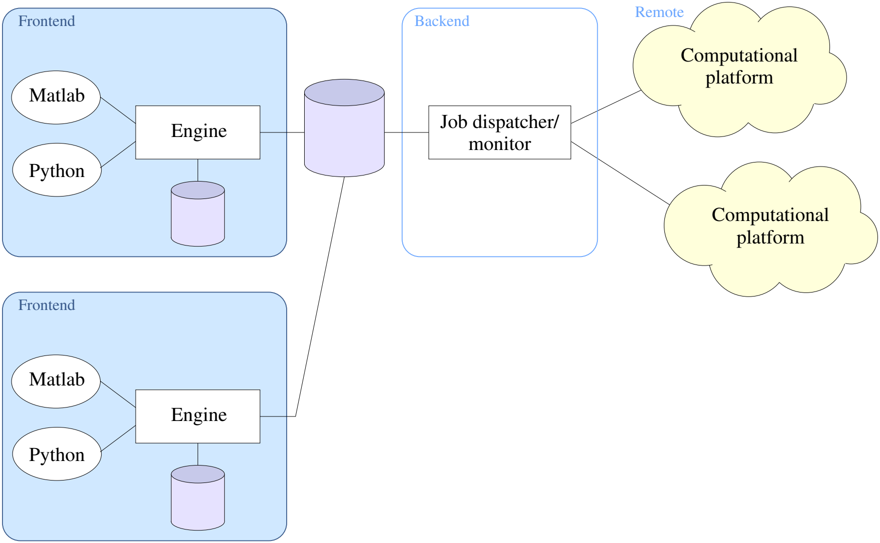
Architecture



Architecture



Architecture



Frontend – Projects

- Frontend resides on local machine
- Provides routines for job specification and processing of results
- Job specification is done in context of project (database):

Projects

- Instantiation:

```
project = dmcProject('demo')
```

- Local name 'demo' used to look up database information
- Adding a new project or joining an existing one:

```
project = dmcCreateProject('demo', hostname, ...)
```

- Hostname, database name, user credentials (prototype)
- Issues: User management, authentication, permissions

What is a job?

- Three components:
  1. Function
  2. Parameters
  3. Computational platform (optional)

Frontend – Jobs

What is a job?

- Three components:
  1. Function
  2. Parameters
  3. Computational platform (optional)

Frontend – Functions

- Creating a function object:

```
fun = project.importFunction('myfun.m')
```

- Numerous challenging issues:
  - Version control / identification of a function
  - How can other users identify / know about this function?
  - Dependencies / compiled libraries
  - Easier for established software packages

Frontend – Functions

- Creating a function object:

```
fun = project.importFunction('myfun.m')
```

- Numerous challenging issues:
  - Version control / identification of a function
  - How can other users identify / know about this function?
  - Dependencies / compiled libraries
  - Easier for established software packages

Version control and function identification

Version control

- Results are associated with a particular function
- Different version IDs or new functions names
- Changing a function 'invalidates' results in database
- Keep existing results for each version
- Version control is beyond scope of the database

Implementation

- Functions are identified by their SHA code
- Copy of the source code is stored in the database
- Minor changes: declare as equal to another function
- Users can use repository (Git, Dropbox, etc)

Dependencies

General issues

- Function may only be top of dependency graph
- How to determine all (indirect) dependencies?
- How to ensure correct versions?
- Compilation on computational platform
- Possibility: machine images on cloud
- High level of reproducibility but not suitable for prototyping

Implementation

- Allow only stand-alone functions (not very satisfactory)
- Any dependencies must be on computational platform
- Better solution possible in Python (piCloud)
- Compiled libraries remain difficult (must be pre-installed)

What is a job?

- Three components:
  1. Function
  2. Parameters
  3. Computational platform (optional)

Frontend – Parameters

Several ways of using parameters:

```
% Direct specification
fun('method',3,'scale',true)

% Create a parameter object
params = dmcParameters('method',3,'scale',true);
fun(params)

% Specialize the function
fun.setParameters('method',3,'scale',true) –or–
fun.setParameters(params)
fun()
```

Frontend – Computational platform

Need to specify where the code is executed:

```
% Direct specification
platform = project.getPlatform('cluster')
fun(platform,params)

% Set default platform
project.setDefaultPlatform(platform)
fun(params)
```

Frontend – Computational platform

Need to specify where the code is executed:

```
% Direct specification
platform = project.getPlatform('cluster')
fun(platform,params)
```

Many unresolved issues

- Should platforms be project specific?
- How to specify/configure a new platform?
- Access control and platform maintenance
- How to specify / deal with dynamic platforms (cloud)?
- Starting and stopping nodes / accounting of computational time
- Ideally done by online server

Frontend – Jobs

- **Three components of a job**: ✓
  1. Function
  2. Parameters
  3. Computational platform (optional)
- **Putting everything together**

Function evaluation

- Submission of a function evaluation:

```
ticket = fun(platform,param)

ticket = project.evaluate(fun,param,platform)
```

Function evaluation

- Submission of a function evaluation:

```
ticket = fun(platform,param)

ticket = project.evaluate(fun,param,platform)
```

- Parameters and computational platform are optional
- Combination identifies a unique job
- Dynamic versus static results

```
ticket = project.evaluateAny(fun,param,platform)
```

- Parallel job submissions are often structured;
  want to avoid submitting jobs individually

Example

```
options1 = dmcOptions('dimM',10,20,30)
options2 = dmcOptions('dimN',10,100,1000)
options = options1 * options2
tickets = project.evaluate(fun,options,'verbose',true)
```

Options

Example

```
options1 = dmcOptions('dimM',10,20,30)
options2 = dmcOptions('dimN',10,100,1000)
options = options1 * options2
tickets = project.evaluate(fun,options,'verbose',true)
```

Features

- List of functions, parameters (`dmcParameters`), platforms
- Outer product and summation of options
- At most one function and platform per combination
- No replication of parameters
- Evalute gives array of job IDs (both new and existing)

Job management

```
>> tickets.summary
======================================
Job summary
======================================
Pending    :   2 / 100
Submitted :  25 / 100
Running    :  30 / 100
Completed :  43 / 100

>> size(tickets)
5 20

>> tickets.summary(1,3)
======================================
Summary for job 10
======================================
Status : Submitted
```

- Check individual ticket status or wait for completion
- Not implemented yet: cancel, delete, restart, etc.

Function evaluation results

```
>> summary(1,2)
===================================================
Submitted :  04/17/2013 14:38:18
Started   :  04/17/2013 14:39:08
Completed :  04/17/2013 14:40:26
--- Standard output -------------------------------
                   < M A T L A B (R) >
         Copyright 1984-2012 The MathWorks, Inc.
           R2012b (8.0.0.783) 64-bit (glnxa64)
                     August 22, 2012

To get started, type one of these:  helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

Iteration 1
Iteration 2
...
Done
--------------------------------------------------------------
```

Function evaluation results

Data access

- Results are accessed by field name:

```
>> [value,mask] = tickets.getValue('name')
>> [value,mask] = tickets.getValue('name',defaultValue)
>> [value,mask] = tickets.name
```

- Can also access individual results (structure)

Formatting

- Structure of options provides information about intent
- Can use this in formatting the output (plots and tables)

Function evaluation results

Data access

- Results are accessed by field name:

```
>> [value,mask] = tickets.getValue('name')
>> [value,mask] = tickets.getValue('name',defaultValue)
>> [value,mask] = tickets.name
```

- Can also access individual results (structure)

Formatting

```
dmcTabulate(tickets,'iterations as iter','runtime');
```

function	problem 1 iter	problem 1 runtime	problem 2 iter	problem 2 runtime	problem 3 iter	problem 3 runtime
Function #1	124	7.39374	89	5.12939	92	6.98314
Function #2	203	10.5812	170	8.26123	163	8.06182
Function #3	157	9.69676	168	10.1924	165	9.72862

Data access

- Results are accessed by field name:

```
>> [value,mask] = tickets.getValue('name')
>> [value,mask] = tickets.getValue('name',defaultValue)
>> [value,mask] = tickets.name
```

- Can also access individual results (structure)

Formatting

- Structure of options provides information about intent
- Can use this in formatting the output (plots and tables)
- Options, functions, parameters, and platforms can be labeled
- More work needed here

Front-End Implementation

Implementation

Three components:

- Front-end interface (Matlab, Python)
- DMC Core (C)
- Intermediate layer (Mex, Python extension)

Interface language implements classes that interact with the core through the intermediate layer:

- Project, platform, function classes
- Parameters, options (manipulations)
- Ticket array

Preparing for job submission

Project and platform simply query the database:

```
project = dmcProject('demo');
platform = dmcPlatform('cluster');
project.setDefaultPlatform(platform);
```

Functions

- Specify source file
- Compute SHA and import if needed (engine)
- Return function ID

Options

- Combine options
- Preliminary check for uniqueness of fields

Submit jobs

Job submission

Interface language (Matlab, Python)

- Checks if platform and function are given, set default
- Converts objects into structs for easy access

Intermediate layer

- Converts data types to DMC types

Matlab	*Python*	*DMC Engine*
`int32`	`int`, `numpy.int32`	`int32`
`double`	`float`, `numpy.double`	`double`
...	...	...
`string`	`string`	`string`
`struct`, `dmcStruct`	`dictionary`	`struct`
`cell array`	`numpy object array` `list`, `tuple`, `set`	`cell`

Job submission

Interface language (Matlab, Python)

- Checks if platform and function are given, set default
- Converts objects into structs for easy access

Intermediate layer

- Converts data types to DMC types

Core engine

- Generates all required parameter combinations
- Serializes each of the combinations
- Checks database for existing jobs or submit new one
- Returns array of job IDs

Job submission

Interface language (Matlab, Python)

- Checks if platform and function are given, set default
- Converts objects into structs for easy access

Intermediate layer

- Converts data types to DMC types

Core engine

- Generates all required parameter combinations
- Serializes each of the combinations
- Checks database for existing jobs or submit new one
- Returns array of job IDs

Serialization

- Lasso problem

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2 \quad \text{subject to} \quad \|x\|_1 \le \tau$$

- Evaluate for series of τ, b:

```
optsTau = dmcOptions('tau',1:10)
optsB   = dmcOptions('b',b1,b2,b3,b4,b5)
opts    = optsB * optsTau
tickets = project.evaluate(fun,opts,'A',BIG_Matrix)
```

- Generates 50 tuples (A, b, τ), each with the same A

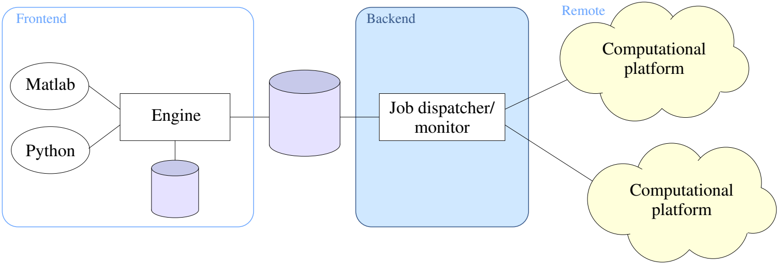
Chunks and pre-serialization

Solution

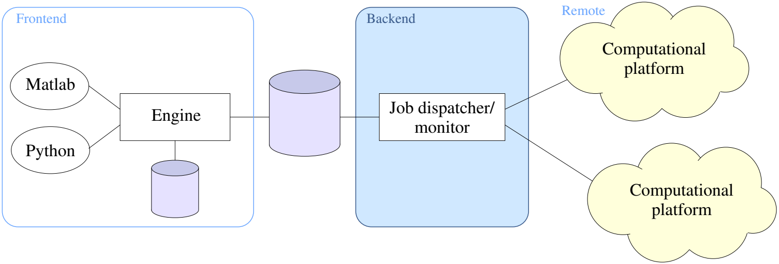
- Store large serialized variables/blocks as chunks
- Refer to chunk IDs instead of replicating data
- Reduces storage requirements
- Pre-serialize individual options before expanding/combining

Back-End Implementation

Backend


Frontend
Matlab
Python
Engine
Backend
Job dispatcher/
monitor
Remote
Computational
platform
Computational
platform

Backend



Tasks

- Check database for new jobs to submit
- Check status of existing jobs
- Import results and clean up completed jobs

Backend

Computational platform

Provides small set of primitives (shell scripts):

- Generate platform-wide unique job ID
- Provide a path name for the job
- Submit job to queue
- Get job status
- Clean up files for a given job

Backend

Computational platform

Provides small set of primitives (shell scripts)

Framework routines:

- Thin wrapper to Python and Matlab functions
- Read parameter files
- Write output files

Backend

Computational platform

Provides small set of primitives (shell scripts)

Framework routines:

- Thin wrapper to Python and Matlab functions
- Read parameter files
- Write output files

Job dispatcher

- Queries the database for pending and submitted jobs
- Calls primitives on computational platform
- Copies and outputs files required for each job
- Monitors and updates status information
- Imports results into the database

Related Work

PiCloud

- Platform for execution of functions on PiCloud:

```
>>> import cloud
>>> def add(x, y):
      return x+y
>>> add(1, 2)
3
>>> jid = cloud.call(add, 1, 2)
>>> cloud.result(jid)
3
```

+ Very easy to use
– Packaged with PiCloud computing time
± No result database (could be added as stand-alone module)
– Shutting down – acquired by Dropbox in 2013

Workflows and experiment databases

Workflows

- Extensive existing work on workflows
- Suitable for well-established process pipelines
- Less suitable for rapid prototypic and fine-granularity jobs
- Execution platform could be used

Experiment databases

- Submission of results to shared database
- Can be mined for patterns and used for meta-learning
- Structured databases design for certain problem classes

Summary

- Presented a prototype framework for structured computation
- Introduced sparse recovery solver SPGL1
- Discussed linear-operator toolbox Spot

Thank you!