Inverse Problems Toolkit

Ferréol Soulez

Inverse problems

Solving an inverse problem involves the minimisation:

$$x = \arg\min_x \| H \cdot x - y \|^2_{\mathbf{W}} + \mu f(x)$$

- involves the computation of H, H* and H*H
- often involves the use of the proximity operator of f

Goals

- Write codes like equations in papers
- faster development / test of ideas
- reduce re-implementation of codes
- make collaboration easier
- easy to learn
- reduce bugs & regression

Structure of the toolkit

Two main classes with subclasses for each case:

- Linear operators LinOp
- Proximal function Prox

LinOp classes

Implement any linear operator H and provides method

- Apply(x) (overloading mtimes)
- Adjoint(y) (overloading transpose)
- H*H(x) and HH*(y) (if different) and H*WH
- Inverse(y) (if exist)
- Inverse*(x) (if exist)

Existing LinOp classes

- Identity H = Id
- Scale(a) H = a Id
- Diagonal(d) H = diag(d)
- Matrix(M,index) H = M
- Sum(index) Sum of the indexed dimensions
- Grad(index) Finite difference along the indexed dims
- DFT Discrete Fourier transform
- SDFT(index) DFT along the indexed dims
- Convolution(psf,index)
- Fresnel
- Diffraction

« Sliced » operators

Matlab **flawed** matrix / vector design

- vector are 2D but images can be 3D
- Matrix are 2D but operator are often 4D (for images)

Implementation of sliced operators:

- Matrix(M,index) Matrix multiplication
- Sum(index) Sum of the indexed dimensions
- Grad(index) Finite difference along the indexed dims
- SDFT(index) DFT along the indexed dims
- Convolution(psf,index)
- Sfft() and iSfft()

Operation on LinOp

- SumLinOp Linear combination of LinOps (overload plus())
- MulLinOp Composition of LinOps (overload mtimes())
- Adjoint Adjoint of a LinOp (overload ctranspose)
- OneToMany Stack LinOps that applied to the same x (WARNING specific behaviour)

Make your own LinOp

Create you class with LinOp heritage and at least implement H and H*

```
classdef YourLinOp < LinOp
  methods
      function this = YourLinOp(~) % Constructor
       this.name = 'YourLinOp'    % name of the linear operator
        this.sizein = ;          % dimension of the right hand side vector space
        this.sizeout = ;         % dimension of the left hand side vector space
        this.isinvertible =  ; % true if the operator is invertible
        this.issquare =  ; % true if the operator is square
        this.iscomplex =;       % true is the operator is complex

       |
      end
      function Apply(this,x) % Apply the operator
       |
      end
      function Adjoint(this,x) % Apply the adjoint
        |
    end
  end
```

Check you LinOp with the function CheckLinOp(…)

Proximity operator

Proximity operator :

$$\mathrm{Prox}_{\alpha\varphi}(\mathbf{x}) = \arg\min_{\mathbf{z}} \| \mathbf{x} - \mathbf{z} \|_2^2 + \alpha\varphi(\mathbf{x})$$

Implemented in class Prox with methods:

- Apply(x)
- Cost not implemented yet
- FCost not implemented yet
- Residuals not implemented yet

Prox classes

- NonNegativity — Prox.Apply(x) = max(x,0)
- Constant (cst) — Prox.Apply(x) = cst
- L2 — L2 norm prox (weighted average)
- L1 — L1 norm prox (soft thresholding)
- JointL1(index) — JointL1 (aka group sparsity) along the indexed dimensions

Make your own prox

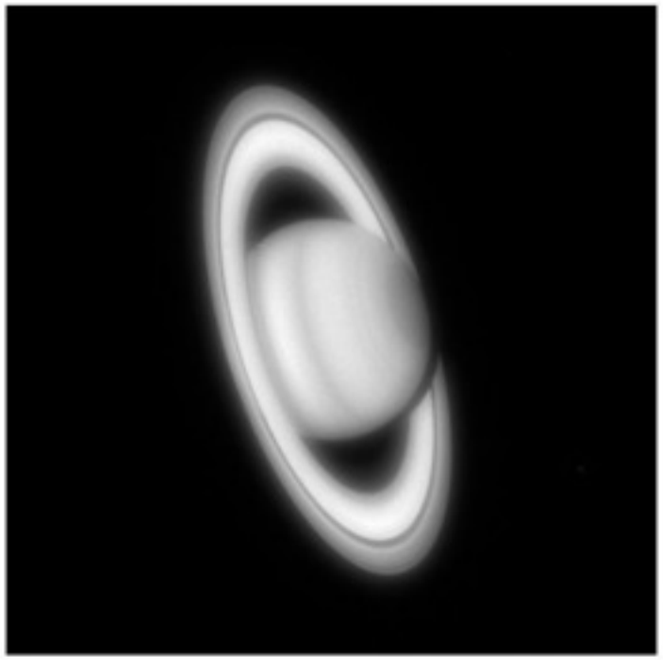
- Create your class derived of Prox class
- At least implement Apply method

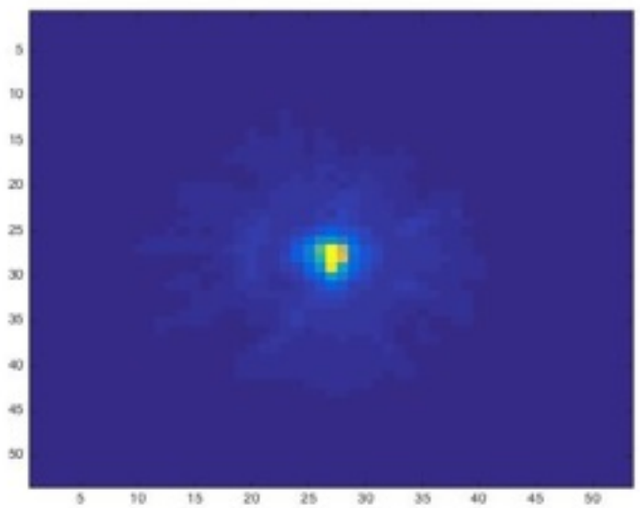
```
classdef YourProx < Prox
    %% YourProx :
    %  Matlab Inverse Problems Library
    %
    % Example
    % Obj = YourProx(x,alpha):
    % YOUR DOC
    %

    methods
        function this = YourProx() % Constructor
            this.name='YourProx';
        |
        end
        function x = Apply(this,y,alpha) % Apply the prox operator
        |
        end
    end
end
```

Example

Linear inverse problem: Deconvolution





PSF

Simple Tikhonov estimate:

$$\mathbf{x} = \arg\min_{\mathbf{x}} \| \mathbf{H} \cdot \mathbf{x} - \mathbf{y} \|_{\mathbf{W}}^2 + \mu \| \mathbf{D} \cdot \mathbf{x} \|_2^2$$

Normal equations

$$\left(\mathbf{H}^* \cdot \mathbf{W} \cdot \mathbf{H} + \mu \mathbf{D}^* \cdot \mathbf{D} \right) \cdot \mathbf{x} = \mathbf{H}^* \cdot \mathbf{W} \cdot \mathbf{y}$$

Example

Linear inverse problem: Deconvolution

```
% convolution operator
H = Convolution(fftshift(psf));
% Finite difference operator
D = Grad(size(saturn));
% Inverse covariance matrix (precision)
W = Diagonal(w); % w_k =0 if no measurement in k


mu =1e-2; %hyperparameter

b = H'* W*saturn;
A = H'*W*H + mu * D'*D;
% Solving Ax = b using conjugate gradient
maxiter = 100;
x0 =  zeros(size(saturn));
x = ConjGrad(A,b,x0,maxiter);

% Equivalent but maybe faster
A = OneToMany({H,D},[1, mu]);
x = ConjGrad(A,b, x0,maxiter,{W,1});
```

Example

Non linear inverse problem: deconvolution with TV

MAP solution:

$$\mathbf{x} = \underset{\mathbf{x}\geq\mathbf{0}}{\arg\min} \| \mathbf{H}\cdot\mathbf{x} - \mathbf{y} \|_{\mathbf{W}}^2 + \mu \mathrm{TV}(\mathbf{x})$$

Constrained formulation:

$$\mathbf{x} = \arg\min \| \mathbf{H}\cdot\mathbf{x} - \mathbf{y} \|_{\mathbf{W}}^2 + \mu \left\| \mathbf{z} \right\|_{2,1} \quad \text{s.t.} \begin{cases} \mathbf{v} \geq \mathbf{0} \\ \mathbf{x} = \mathbf{v} \\ \mathbf{z} = \mathbf{D}\cdot\mathbf{x} \end{cases}$$

Use augmented Lagrangian to enforce contraints

$$L(\mathbf{x},\mathbf{z},\mathbf{v},\mathbf{u}_1,\mathbf{u}_2) = \left\| \mathbf{H}\cdot\mathbf{x} - \mathbf{y} \right\|_{\mathbf{W}}^2 + \mathbf{u}_1\cdot(\mathbf{D}\cdot\mathbf{x} - \mathbf{z}) + \frac{\rho_1}{2}\left\| \mathbf{D}\cdot\mathbf{x} - \mathbf{z} \right\|_2^2 + \mathbf{u}_2\cdot(\mathbf{x} - \mathbf{v}) + \frac{\rho_2}{2}\left\| \mathbf{x} - \mathbf{v} \right\|_2^2 + \mu \mathrm{TV}(\mathbf{z}) \; + \mathrm{P}(\mathbf{v})$$

$$\text{With} \;\; P(\mathbf{v}) = \begin{cases} 0 & \text{if } \mathbf{v} \geq \mathbf{0} \\ +\infty & \text{otherwise} \end{cases}$$

Example

Non linear inverse problem: deconvolution with TV

ADMM framework

- Sub problem 1: simple quadratic problem

$$\mathbf{x} = \arg\min_{\mathbf{x}} \left\|\mathbf{H}\cdot\mathbf{x}-\mathbf{y}\right\|_2^2 + \frac{\rho_1}{2}\left\|\mathbf{D}\cdot\mathbf{x}-\tilde{\mathbf{z}}\right\|_2^2 + \frac{\rho_2}{2}\left\|\mathbf{x}-\tilde{\mathbf{v}}\right\|_2^2$$

$$\text{with } \tilde{\mathbf{z}} = \mathbf{z} - \frac{\mathbf{u}_1}{\rho_1} \text{ and } \tilde{\mathbf{v}} = \mathbf{v} - \frac{\mathbf{u}_2}{\rho_2}$$

- Sub problem 2:

$$\mathbf{v} = \arg\min_{\mathbf{v}} \frac{\rho_2}{2}\left\|\mathbf{v}-\tilde{\mathbf{x}}\right\|_2^2 + P(\mathbf{v}) = \max(\tilde{\mathbf{x}},0) \text{ with } \tilde{\mathbf{x}} = \mathbf{x} + \frac{\mathbf{u}_2}{\rho_2}$$

$$= \text{Prox}_{1/\rho_2 P}(\tilde{\mathbf{x}})$$

- Sub problem 3: denoising problem

$$\mathbf{z} = \arg\min_{\mathbf{z}} \frac{\rho_1}{2}\left\|\mathbf{z}-\tilde{\tilde{\mathbf{x}}}\right\|_2^2 + \mu\left\|\mathbf{z}\right\|_{1,2} \text{ with } \tilde{\tilde{\mathbf{x}}} = \mathbf{x} + \frac{\mathbf{u}_1}{\rho_1}$$

$$= \text{Prox}_{\mu/\rho_2 \|\,\|_{1,2}}(\tilde{\tilde{\mathbf{x}}})$$

Non linear inverse problem: deconvolution

```
% convolution operator
H = Convolution(fftshift(psf));
% Finite difference operator
D = Grad(size(saturn));

B = Identity(size(data));
zProx = JointL1(3); % JointL1( D*x)  == Total variation
%zProx = L2();
tProx = NonNegativity();
rho1 = 1e-3;
rho2 =1e-3;

mu =.1; %hyperparameter
x0 = zeros(size(data));
cgmaxiter = 5;
maxiter =100;
x=ADMM_Restore(H,D,B,W,data, zProx, tProx,mu, rho1, rho2,x0,maxiter,cgmaxiter);
```

```
function x=ADMM_Restore(H,D,B,W,y, zProx, tProx,mu, rho1, rho2,x0,maxiter,cgmaxiter)

A = OneToMany({H,D,B},[1, rho1, rho2]);
Wy = W*y;
for k=1:maxiter
% Sub problem 1 || Hx - y||_w^2 + rho1/2 || Dx - z + u1/rho1 ||_2^2 + rho2/2 || x - t + u2/rho2 ||_2^2
     zu1 = z - u1/rho1;
     tu2 = t - u2/rho2;
     b = A.Adjoint({Wy, zu1,tu2});%b = H'* wy + rho1*D'*zu1 + rho2 *B'*tu2;
     x = ConjGrad(A,b,  x,cgmaxiter,{W,1,1});
% Sub problem 2
     Dx = D*x;
     xu1 = Dx + u1/rho1;
     z = zProx.Apply(xu1,mu/rho1);
% Sub problem 3
     t_prev = t;
     Bx = B*x;
     xu2 = Bx + u2/rho2;
     t = tProx.Apply(xu2, 1./rho2);
% Residuals of the constraints
     res1 =  Dx - z;
     res2 = Bx - t;
% Lagrange parameters update
     u1 = u1 + rho1 * res1;
     u2 = u2 + rho2 *  res2;
end
```

How to use it & contribute

Git it!

Git repository accessible to all the lab

`git clone https://LOGIN@git.epfl.ch/repo/invpblib.git`

Feel free to fix bugs/doc, add your LinOp and Prox

```
git pull
```
I make your changes, create your files
```
git add YourFiles.m
git commit -a
git push
```

What's next

- Add Cost function classes ?
- Add optimization routines ?
- Add some compatibility layer for **`UNLocBoX`** toolbox (from LTS2)
- Any suggestion?