FrameLab: Development Guide

February 6, 2015

1 Overall Design

FrameLab 1.0 is designed to have an object-oriented, user friendly scripting interface with compute intensive routines written in compiled languages such as C and CUDA/C. The current scripting language is Matlab, using MEX as an interface mechanism to pull in compiled libraries. In the future we plan to implement Python/iPython as an alternative to Matlab, to keep the entire code open-source.

2 Matlab Object Oriented System

The current goal of Framelab is to solve approximate versions of linear inverse problems of the sort

$$\mathcal{A}u = f \tag{1}$$

Through some discretization, (1) is approximated by a finite dimensional linear system

$$A\mathbf{u} = \mathbf{f} \tag{2}$$

To handle more general and complicated problems of this type, we define an object-oriented system where A, $\mathbf{u}$ and $\mathbf{f}$ are **abstract** data types instead of simply matrices and vectors.

1. DataTypes: To allow for a flexible modeling system, for each problem of type (1),(2) we create an abstract data type for both $\mathbf{u}$ and $\mathbf{f}$.

2. Operators: To model the linear operator A, we again use abstract data types. Thus we define a class for each A, for example the ConeBeamScanner transform class

3 Compute Kernels

3.1 Computed Tomography

From [**?**]

Compiling MEX Libraries

```
mex -L"/usr/local/cuda/lib64" -lcudart -I"./" Ax_fan_mf.cpp Ax_fan_mf_cpu_siddon.cpp
 Ax_fan_mf_cpu_new.cpp Ax_fan_mf_cpu_new_fb.cpp Ax_fan_mf_gpu_siddon.cu
 Ax_fan_mf_gpu_new.cu Ax_fan_mf_gpu_new_fb.cu find_area.cpp sort_alpha.cpp
```

Possible error message about invalid conversion fron int to mxComplexity: change

```
plhs[0]=mxCreateNumericMatrix(nx*ny*nt,1,mxSINGLE_CLASS,0);
```

to

```
plhs[0]=mxCreateNumericMatrix(nx*ny*nt,1,mxSINGLE_CLASS,mxREAL);
```

in any mex interface files

Alternating Direction Method of Multipliers Recall that ADMM is designed to solve problems of the sort

$$(x^*, y^*) := \arg\min_{x,y} F(x) + G(y) \quad \text{s.t.} \quad Ax + By = b \tag{$\mathcal{P}$}$$

The approach is to consider the Augmented Lagrangian:

$$\mathscr{L}_\rho(x, y, \lambda) := F(x) + G(y) + \langle \lambda, Ax + By - b\rangle + \frac{\rho}{2}\|Ax + By - b\|_2^2$$

We then consider the saddle point problem

$$(x^*, y^*, \lambda^*)_\rho = \arg\min_{(x,y)} \arg\max_{\lambda} \mathscr{L}_\rho(x, y, \lambda) \tag{3}$$

Since we are interested in the saddle point itself and not the value of the functionals, we may complete the square in the definition of $\mathscr{L}_\rho$ to obtain

$$(3) = \arg\min_{(x,y)} \arg\max_{\lambda} F(x) + G(y) + \frac{\rho}{2}\|Ax + By - (b - \lambda/\rho)\|_2^2$$

For notational convenience, we define

$$\mathscr{L}^*_\rho(x, y, \lambda) := F(x) + G(y) + \frac{\rho}{2}\|Ax + By - (b - \lambda/\rho)\|_2^2$$

If we then perform coordinate descent/ascent, we arrive at the 3-step ADMM scheme:

$$\begin{cases} x^{(k+1)} & = \arg\min_x \mathscr{L}^*_\rho(x, y^{(k)}, \lambda^{(k)}) \\ y^{(k+1)} & = \arg\min_y \mathscr{L}^*_\rho(x^{(k+1)}, y, \lambda^{(k)}) \\ \lambda^{(k+1)} & = \arg\max_\lambda \mathscr{L}^*_\rho(x^{(k+1)}, y^{(k+1)}, \lambda) \end{cases}$$

The method can be generalized in the particular case that $F(x) + G(y)$ is further separable, e.g. $F(x) + G(y) = F_1(x_1) + F_2(x_2) + \ldots + F_n(x_n)$, with $A\mathbf{x} = \mathbf{b}$, where $\mathbf{x} = (x_1, \ldots, x_n)^T$. The augmented Lagrangian then takes the form

$$\mathscr{L}_\rho(\mathbf{x}, \Lambda) := \sum F_i(x_i) + \langle \Lambda, A\mathbf{x} - \mathbf{b} \rangle + \frac{\rho}{2} \|A\mathbf{x} - \mathbf{b}\|_2^2$$

where $\Lambda = (\lambda_1, \ldots, \lambda_n)^T$.

In FrameLab, we have implemented an ADMM object designed to solve problems of the type $\mathcal{P}$.

4 Another Section