Plug-and-Play ADMM using MATLAB and PyTorch

Kaibo Tang

June 18, 2024

1 Background

We typically use alternating direction method of multipliers (ADMM) to solve problems of the form:

$$\min_{x,z} f(x) + g(z) \text{ subject to } Ax + Bz = c. \tag{1}$$

In particular, we focus on problems of the form

$$\min_{x} f(x) + g(x) \iff \min_{x,z} f(x) + g(z) \text{ subject to } x = z. \tag{2}$$

A handful of tasks fall under this category, e.g., image restoration, which will be the focus of this note.

2 Problem Formulation

Consider the following image restoration problem in the form of (2):

$$\min_{x} \|Ax - b\|_2^2 + R(x) \iff \min_{x,z} \|Ax - b\|_2^2 + R(z), \tag{3}$$

where x is the desired underlying image (of arbitrary dimension) to be reconstructed, b is the potentially noisy) observation, and $R(\cdot)$ is an arbitrary regularizing term that punishes deviation from the prior knowledge about the underlying image.

The augmented Lagrangian for parameter $\rho > 0$ is given by

$$L_\rho(x, z, u) = \|Ax - b\|_2^2 + R(z) + u^T(x - z) + \frac{\rho}{2}\|x - z\|_2^2. \tag{4}$$

Completing the square and letting $w = u/\rho$ yields

$$L_\rho(x, z, u) = \|Ax - b\|_2^2 + R(z) + \frac{\rho}{2}\|x - z + w\|_2^2 - \frac{\rho}{2}\|w\|_2^2. \tag{5}$$

Following ADMM, the problem is solved by iteratively performing the following steps:

$$x^{(k)} = \arg\min_x \|Ax - b\|_2^2 + \frac{\rho}{2}\|x - z^{(k-1)} + w^{(k-1)}\|_2^2 \tag{6}$$

$$z^{(k)} = \arg\min_z R(z) + \frac{\rho}{2}\|x^{(k)} - z + w^{(k-1)}\|_2^2 \tag{7}$$

$$w^{(k)} = w^{(k-1)} + x^{(k)} - z^{(k)}. \tag{8}$$

Under the Plug-and-Play (PnP) ADMM scheme, (7) is replaced by an arbitrary denoiser that removes additive white Gaussian noise from the image. The detailed derivation of the equivalence between (7) and a Gaussian denoiser can be found here.[1] Intuitively, a denoiser of choice is assumed to "carry" some form of implicit prior about the underlying image and is capable of decreasing the cost function in (7) significantly.

Hence, given a denoiser of choice $\mathcal{D}$, the ADMM updates becomes

$$x^{(k)} = \arg\min_x \|Ax - b\|_2^2 + \frac{\rho}{2}\|x - z^{(k-1)} + w^{(k-1)}\|_2^2 \tag{9}$$

$$z^{(k)} = \mathcal{D}(x^{(k)} + w^{(k-1)}) \tag{10}$$

$$w^{(k)} = w^{(k-1)} + x^{(k)} - z^{(k)}. \tag{11}$$

3 Implementation

3.1 Preparation

Suppose we have four files at hand. `model.py` contains the model architecture;

[1]http://arxiv.org/abs/1903.08616

```python
# model.py

import torch.nn as nn

class model(nn.Module):
    def __init__(self, *args, **kwargs):
        super().__init__()
        pass
    def forward(self, x):
        pass
```

`model.pt` is the `state_dict` saved after training is finished

```python
# arbitrary_script.py

# the last kwarg is REQUIRED for it to work in MATLAB
torch.save(
    model.state_dict(),
    "./model.pt",
    _use_new_zipfile_serialization=False,
)
```

`for_matlab.py` defines some helper functions for model loading, inference, and data handling;

```python
# for_matlab.py

from model import model

import numpy as np
import torch

def to_tensor(x, device='cuda'):
    # assume x is from MATLAB of shape (Nx, Ny, Nz, C)
    # 1. load as contiguous array
    x = np.asarray(x)
    x = np.ascontiguousarray(x)
    # 2. convert x to tensor and permute to (1, C, Nx, Ny, Nz)
    x = torch.tensor(
        x[:, :, :, :, np.newaxis],
        dtype=torch.float32,
    ).permute(4, 3, 0, 1, 2).to(device)
    return x
```

```python
def to_matlab(x):
    # assume x is a tensor of shape (1, C, Nx, Ny, Nz)
    # 1. detach cpu squeeze reshape numpy
    x = x.detach().cpu().squeeze().permute(1, 2, 3, 0).numpy()
    # 2. convert x to contiguous array
    x = np.ascontiguousarray(x)
    return x

def load_model(device='cuda'):
    model_dir = "./model.pt"
    model = model()
    model.load_state_dict(torch.load(model_dir))
    model.to(device)
    model.eval()
    return model

def denoiser(x):
    model = load_model()
    inferer = load_inferer()
    x = to_tensor(x)
    with torch.no_grad():
        y = model(x)
    y = to_matlab(y)
    return y
```

`load_denoiser.m` is a MATLAB function that loads the denoiser, and return a function handle to be called in MATLAB

```matlab
% load_denoiser.m

function [D] = load_denoiser()
    model = py.importlib.import_module('model');
    py.importlib.reload(model);
    inference = py.importlib.import_module('for_matlab');
    py.importlib.reload(inference);
    D = @(x) double(inference.denoiser(x));
end
```

3.2 Initialize variables and load denoiser

Before starting, x, z, w need to be initiated as 0 and load the denoiser.

```
% main.m

% suppose image size is Nx, Ny, Nz
x = zeros(Nx,Ny,Nz,C);
z = zeros(Nx,Ny,Nz,C);
w = zeros(Nx,Ny,Nz,C);

% load denoiser
denoiser = load_denoiser();
```

Note that although the variables are initiated as matrices, during `lsqr`, the variables are treated as column vectors.

3.3 Step 1

For the ease of implementation, the update in (9) can be re-written as

$$x^{(k)} = \arg\min_x \left\| \begin{bmatrix} A \\ \sqrt{\rho/2}I \end{bmatrix} x - \begin{bmatrix} b \\ \sqrt{\rho/2}(z^{(k-1)} - w^{(k-1)}) \end{bmatrix} \right\|_2^2. \tag{12}$$

Notice that (12) has closed-form solution $\forall \rho > 0$, which is given by

$$x^{(k)} = \left(A^*A + \frac{\rho}{2}I\right)^{-1} \left(A^*b + \frac{\rho}{2}(z^{(k-1)} - w^{(k-1)})\right), \tag{13}$$

since $(A^*A+\rho/2I)$ is positive definite. Realistically, we could use `pcg` to solve (13), but using `lsqr` to solve (12) offers "favorable numeric properties."[2] Empirically, `lsqr` is twice as fast compared to solving the equivalent normal equation using `pcg`.

```
% main.m

% LHS matrix of the equation
LHS = [A; sqrt(rho/2)*eye(size(A,2)];
% RHS column vector of the equation
RHS = [b; sqrt(rho/2)*(z-w)];
% step 1: call LSQR
[a, ~, ~, ~, ~] = lsqr(LHS, RHS, 1e-6, 10, [], [], x(:));
% reshape
x = reshape(real(a), [Nx,Ny,Nz]);
```

[2] https://www.mathworks.com/help/matlab/ref/lsqr.html

For an empirical comparison of convergence under different methods solving the same equation, see Figure 1.

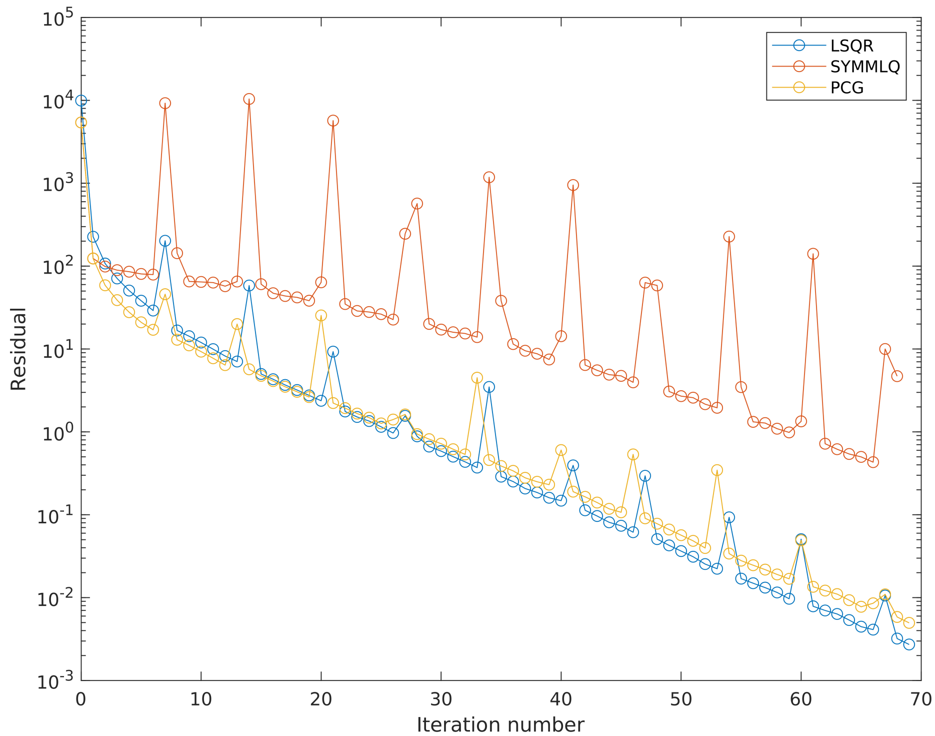

Figure 1: Solving $Ax = b$ using `LSQR` via `lsqr(A, b)` compared to solving $A^*Ax = A^*b$ using `PCG` and `SYMMLQ` via `pcg(A'*A, A'*b)` and `symmlq(A'*A, A'*b)`.

3.4 Step 2 and 3

```
% main.m

% step 2: call the denoiser on x+w
z = denoiser(x+w)

% step 3: update w
w = x+w-z
```

To get PnP ADMM started, place steps 1-3 in a for loop.