Image Deblurring
with Alternating Direction Method of Multipliers

Final Project (Option I)
DDA4310: Computational Imaging

Name: *Derong Jin*
Student ID: *120090562*
Date: *June 19, 2024*

1 Overview

This project implements an image deblurring algorithm based on the Alternating Direction Method of Multipliers (ADMM) with Total Variation (TV) regularization.

Image deblurring aims to remove blurring artifacts from images and restore the original sharp content. By formulating the deblurring problem as an inverse problem, the deblurring problem can be casted into an optimization problem. ADMM is then used to find the optimal solution by decomposing the objective function and alternating between updates. Due to the computational complexity of directly solving large images, a patch-based approach is also adopted where the image is divided into smaller patches, deblurred independently, and stitched back together via Poisson Blending.

The implementation is done in `Python` using sparse linear algebra tools from `SciPy`. Qualitative results on image patches and a full test image demonstrate the effectiveness of the ADMM deblurring method.

2 Method & Algorithm

Image de-blurring algorithms aim to remove the blurring artifacts from images or to restore the original, sharp content. In this section, we will first discuss the formulation of "deblurring problems" and convert the searching problem into an optimization problem. Then we will use ADMM to find an optimal solution. And finally, when the input image is large, we will break down the problems by deblurring the image by small patches and then stitching the patches together with Poisson Image Blending to get the final result within an acceptable time.

2.1 Deblurring as An Inverse Problem

Given a blurry image, we assume it is produced by a "forward blurring process" from an "initial clear image". One of the most popular models of the blurring process is Gaussian blurring, where the blurry image I_{blurry} is given:

$$I_{\text{blurry}} = F_{\text{gaussian}} \circ I_{\text{initial}} + N \tag{1}$$

where I_{initial} is the original image, F_{gaussian} is the Gaussian filter kernel, $\circ$ is image convolution, and N is a random noise. Since image convolution is a linear operator, Eq.1 can be rewrite in vector form:

$$b = Hx + \eta \tag{2}$$

where b, x, and η are the vector form of 2d images I_{blurry}, I_{initial}, and N, respectively, and the convolution is written as the matrix multiplication of H.

To find a satiable initial condition x, the problem can be viewed as an optimization problem:

$$\min_x \quad ||Hx - b||_2^2 \tag{3}$$

In application, the optimization problem often adopts regularization terms to improve the quality of the result. In the image deblurring problem, we often use Total Variance

(TV) regularization to encourage a smooth x:

$$\begin{aligned} &\min_x \|Hx - b\|_2^2 + \lambda \mathrm{TV}(x) \\ &\text{where} \quad \mathrm{TV}(x) = |Dx| = |D_x x + D_y x| \end{aligned} \tag{4}$$

λ is a hyperparameter, and D_x and D_y are matrices representing the convolutions by kernels $\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ respectively.

2.2 Optimization with ADMM

Through Eq.4, the image deblurring problem is formulated as an unconstrained optimization problem with sole optimizing variable x. Under TV regularization, the objective function is convex. Additionally, due to the sparsity of matrix H and D, the Alternating Direction Method of Multipliers (ADMM)[1] is practically useful in this optimization. To begin with, ADMM requires a separable objective function, so we can decompose the original objective function (Eq.4) into regularization term and objective term, and rewrite the optimization as:

$$\begin{aligned} \min_{x,z} \quad & \|Hx - b\|_2^2 + \lambda |z| \\ \text{subject to} \quad & z = Dx \end{aligned} \tag{5}$$

Subsequently, the augmented Lagrangian of Eq.5 is:

$$\begin{aligned} \mathcal{L}_\rho(x, z, y) &= f(x) + g(z) - y^\top (Dx - z) + \frac{\rho}{2}\|Dx - z\|_2^2 \\ &= \|Hx - b\|_2^2 + \lambda |z| - y^\top (Dx - z) + \frac{\rho}{2}\|Dx - z\|_2^2 \end{aligned} \tag{6}$$

In this project, we adopt the scaled form by substituting dual variable y by $u = y/\rho$:

$$L_\rho(x, z, u) = f(x) + g(z) + \frac{\rho}{2}\|r + u\|_2^2 - \frac{\rho}{2}\|u\|_2^2 \tag{7}$$

where r is the residual function

$$r = \text{residual} : (x, z) \rightsquigarrow Dx - z \tag{8}$$

In ADMM, by minimizing $\mathcal{L}_\rho$ the original problem reaches its optimum. To be specific, the algorithm requires three updates in each iteration step:

$$\begin{cases} x^{k+1} \leftarrow \underset{x}{\operatorname{argmin}}\, f(x) + (\rho/2)\|Dx^k - z^k + u^k\|_2^2 \\ z^{k+1} \leftarrow \underset{z}{\operatorname{argmin}}\, g(z) + (\rho/2)\|Dx^{k+1} - z + u^k\|_2^2 \\ u^{k+1} \leftarrow u^k + r^{k+1} = u^k Dx^{k+1} - z^{k+1} \end{cases} \tag{9}$$

In each ADMM iteration (as Eq.9), there are two sub-problems need to be solved. The first one is $\text{argmin}_x \mathcal{L}_\rho(x, z^k, u^k)$ in x-update step, and the solution is given by

$$x = (H^\top H + \rho D^\top D)^{-1}(H^\top b - D^\top(z^k - u^k)) \tag{10}$$

The second subproblem is $\text{argmin}_z \mathcal{L}_\rho(x^{k+1}, z, u^k)$ in z-update step, since g is not differentiable, we adopt an approximate algorithm to solve this sub-problem:

$$\begin{aligned} z &\approx \mathcal{S}_{\lambda/\mu}(Dx^{k+1} + u^k) \\ \text{where} \quad \mathcal{S}_k(v) &= (v-k)_+ - (-v-k)_+ \end{aligned} \tag{11}$$

Algorithm 1 illustrates the overall steps to apply ADMM in image deblurring task with TV regularization.

Algorithm 1: ADMM for Image Deblurring

Data: $b \in \mathbb{R}^n$: the blurry image
$H \in \mathbb{R}^{n\times n}$: the blurring matrix
$D \in \mathbb{R}^{k\times n}$: the regularization matrix

$x \leftarrow \mathbf{0} \in \mathbb{R}^n$;
$z \leftarrow \mathbf{0} \in \mathbb{R}^k$;
$u \leftarrow \mathbf{0} \in \mathbb{R}^k$;
while $\Delta x \geq \varepsilon$ **do** iterate until convergence
  $x = (H^\top H + \rho D^\top D)^{-1}(H^\top b - D^\top(z - u))$;
  $z = \mathcal{S}_{\lambda/\mu}(Dx + u)$;
  $u = u + Dx - z$;
end
$x = (H^\top H + \rho D^\top D)^{-1}(H^\top b - D^\top(z - u))$;
return x;

2.3 Dealing with Large Images: Patchified Deblurring & Stitching

The overall time complexity of ADMM algorithm (Algo.1) increases superlinearly with respect to the increasing number of pixels. This fact makes Algo.1 hard to deal with large images. Under large image condition, we first split the image into many smaller patches and then apply the deblurred on each image patch, then stich the patched results together with Poisson Blending[2].

3 Implementation

This project implement ADMM image deblurrer with `Python`. In the computation, calculating Eq.10 is the most time-consuming. The numerical result is given by `scipy.sparse.linalg.cg`, where $H^\top H + \rho D^\top D$ is implemented as a `LinearOperator`. In practice, this work tried 4 kinds of sparse convolution methods in computing Eq.10: `filter2D` in `OpenCV`, `convolve2d` in `SciPy`, `fft` in `NumPy`, and `simple_conv2d`, a self-implemented convolutional tool writing in `C++`. The performance measurements are shown in Tab.1.

method	average exec time (sec)
`OpenCV`	4.33×10^{-3}
`SimpleConv` (Mine)	1.09×10^{-2}
`scipy:convolve2d`	3.97×10^{-2}
`numpy:fft`	4.26×10^{-2}
`sparse`	1.40×10^{-1}

Table 1: Average execution time of different convolution implementation, conv size = 7×7, image size = 512×512

4 Evaluation

This section evaluates the performance of the proposed ADMM-based image deblurring algorithm. Tests were conducted on synthetic blurred image patches and a full test image to analyze qualitative results. Additionally, hyperparameter tuning was performed to select optimal values.

4.1 Image Deblurring Results

Adopting patched ADMM deblurring and stitching the image patches together with Poisson image blending, the deblurred image is smooth and have shape corners and edges. Fig.1 shows the comparison between blurry input and deblurred output.

(a) Blurred: PSNR = 18.636 (b) ADMM: PSNR = 19.971

Figure 1: Image Deblurring Result

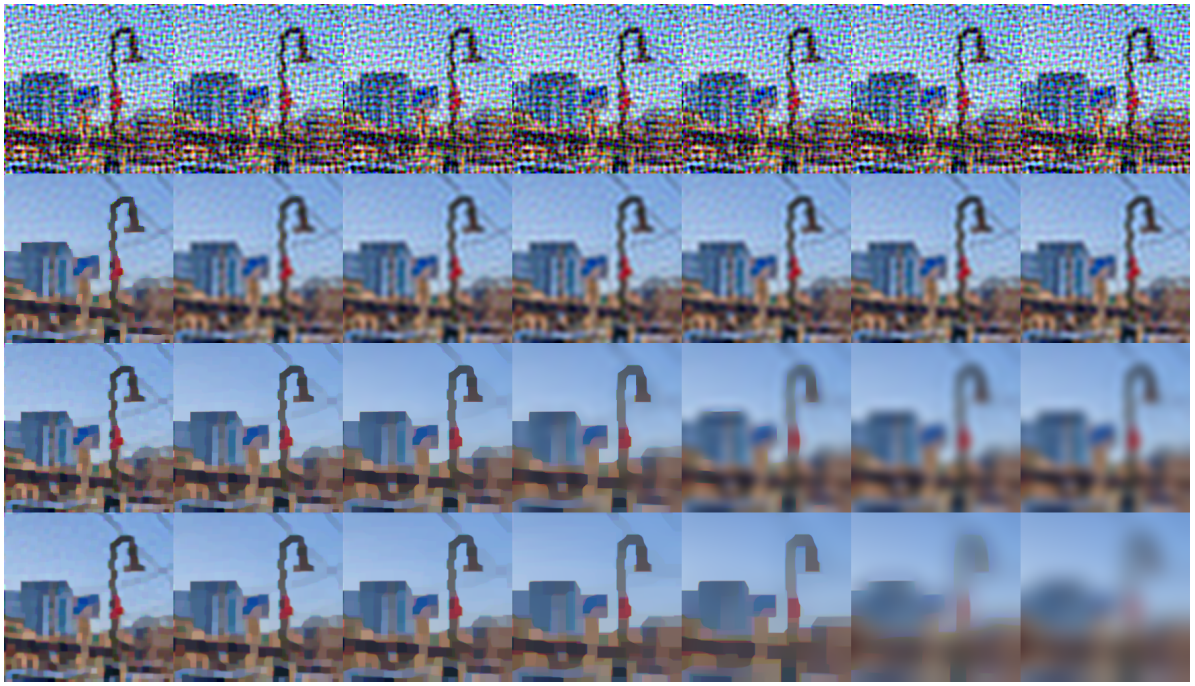
4.2 Hyperparameter Choices

Given a 256×256 image patch, this project searches the hyperparameter space and finds $\rho = 0.1, \lambda = 0.003$ yield very good results. Figure 2 gives a comprehensive illustration of the effects of different hyperparameters. From Fig.2(d), we can see that larger λ encourages smoother images and larger ρ encourages regularization.

4.3 Comparison with CLSF

n image deblurring, the constrained least squares filtering (CLSF) method offers a fast inverse filtering approach. However, as shown in Figure 3, the proposed ADMM method achieves qualitatively superior results over CLSF. The deblurred image from

(a) original (b) blurred (c) ADMM ($\rho : 0.1, \lambda : 0.003$)



(d) deblurred with different hyperparameters (row: $\rho = \{0.0001, 0.01, 0.1, 1\}$, column:$\lambda = \{0.0003, 0.001, 0.003, 0.1, 0.3, 1, 3\}$)

Figure 2: **Result of ADMM on a single image patch**: (a)-(c) shows the deblurring effect of ADMM deblurring method.

ADMM exhibits clearer details and sharper edges compared to the blurrier output of CLSF. While CLSF provides computational efficiency due to its closed-form solution, the iterative nature of ADMM allows it to produce visually more pleasing restorations by better approximating the inverse problem through alternating updates. This demonstrates the advantages of the optimization-based ADMM formulation for image deblurring applications seeking high-quality deblurring results.

5 Conclusion

This project presented the image deblurring algorithm based on ADMM optimization. ADMM decomposes the problem into subproblems with efficient solutions, enabling image restoration. Additionally, for a large image, this project split the image into patches, and stitch the patch-deblurred image with Poisson Image Blending.

(a) ADMM: PSNR = 30.890 (b) ADMM: PSNR = 32.064

(c) ADMM: P0 (d) ADMM: P1 (e) ADMM: P2 (f) ADMM: P3 (g) ADMM: P4 (h) ADMM: P5

(i) CLSF: P0 (j) CLSF: P1 (k) CLSF: P2 (l) CLSF: P3 (m) CLSF: P4 (n) CLSF: P5

Figure 3: **Comparison between CLSF and ADMM**: (a) and (b) are the whole deblurred image, (c)-(h) are six selected image patch in (a), and (i)-(n) are the image patch in (b) with the same relative position as (c)-(h) in (a)

Results show the approach effectively removes blur and recovers sharp details. However, due to its iterative nature, ADMM can be slower compared to closed-form methods like CLSF. Future work includes accelerated ADMM variants and GPU implementation to improve speed for large images. Overall, ADMM provides an effective optimization-based solution for image deblurring, though further optimization is needed for adopting to efficiency needs.

Acknowledgements

I would like to take this opportunity to thank Professor Qilin Sun and all the teaching assistants for their guidance throughout the DDA4310: Computational Photography course. I also want to express my deepest gratitude to Professor Sun for developing the wonderful course content, from which I learned a great deal of knowledge and enjoyed the learning experience immensely.

References

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends R in Machine Learning*, 3(1):1–122, 2011.

[2] Patrick P. Perez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on Graphics (TOG)*, 22(3):313–318, 2003.

Appendices

Appendix A Dependecies

The project code depends on the following packages:

- `NumPy`
- `SciPy`
- `argparse`

Appendix B Code Usage

All source code of this project is saved in `Source` directory. In `Source` directory, the ADMM deblurrer entry is `main.py`. To run the program:

```
python main.py [-h] -i IMAGE [-s SAVE_PATH] [-r RHO] [-l LAMD]
               [-ph PATCH_HEIGHT] [-pw PATCH_WIDTH]
               [-gs GAUS_SIZE] [-gstd GAUS_SIGMA]
               [--admm_mode {cv2,sparse,fft,dda4310,scipy}]
```

All image deblurring options are:

- `-i IMAGE, --image IMAGE`: The input image path
- `-s SAVE_PATH, --save_path SAVE_PATH`: Specifying the output image name.
  - default value: `result.png`
- `-r RHO, --rho RHO`: ρ value in ADMM
  - default value: 0.1
- `-l LAMD, --lamd LAMD`: λ value in regularization
  - default value: 0.003
- `-ph PATCH_HEIGHT, --patch_height PATCH_HEIGHT`: the patch height
  - default value: 256
- `-pw PATCH_WIDTH, --patch_width PATCH_WIDTH`: the patch width

  - default value: 256
- `-gs GAUS_SIZE, --gaus_size GAUS_SIZE`: Gaussian blurring kernel size
  - default value: 17
- `-gstd GAUS_SIGMA, --gaus_sigma GAUS_SIGMA`: Gaussian blurring kernel sigma
  - default value: 5
- `--admm_mode {cv2,sparse,fft,dda4310,scipy}`: admm convolution implementation
  - default value: `cv2`
  - If you want to use my self-implemented conv2d implementation, you can set `admm_mode` to `dda4310`. Additionally, please make sure you have installed `pybind11` and installed the module written in `simple_conv2d`, for more information, see Appendix. C.

Appendix C Utilizing Self-implement Conv2d Module

First, intall the module. Since the module is written in C++, please make sure you have already installed C++ build tool-kits. Additionally, the code provides `Python` API by `pybind11`, but you don't need to install `pybind11` since I have place it in the directory.

Then `cd` into the directory `simple_conv2d` and install the code:

```
cd simple_conv2d
python setup.py install
```

Now you can use the conv2d APIs, set `admm_mdoe` to `dda4310` to see the effect.