Report - OPT2 - TP2

Image Deblurring

Disclaimer: in the python file the functions calls are in comments write now

Tom Cuel

March 19, 2025

Contents

Introduction

Image deblurring is a fundamental problem in image processing that aims to recover a sharp image from a blurred observation. Blurring can occur due to various factors, such as camera motion, defocus, or atmospheric distortions. Mathematically, the blurring process is often modeled as a convolution between a sharp image and a blur kernel, with additional noise. Deblurring techniques can be classified into blind and non-blind methods, depending on whether the blur kernel is known or needs to be estimated. Various approaches, including deconvolution algorithms, wavelet transforms, and deep learning models, have been developed to tackle this challenging problem.

1 Problem to solve

1.1 Modelisation of the problem

Blurring an image can be mathematically represented as a linear transformation. Given an image represented as a vector $z \in \mathbb{R}^n$ (where each pixel is a single value in grayscale), blurring can be described as:

$$z' = Cz, \quad z = W^H x \tag{1}$$

Here, C is a matrix representing the blur, and W^H is a transformation that reconstructs the image from its coefficients. The matrix C has more columns than rows, meaning that only a portion of the image information is retained. Reconstructing the original image from a blurred version requires solving an inverse problem. Given the observed pixels b of the blurred image, we model the relationship as:

$$b = CW^H x \tag{2}$$

A naive approach is to minimize the least-squares error:

$$x^* \in \arg\min_{x\in\mathbb{R}^n} \frac{1}{2}\|CW^H x - b\|_2^2 \qquad z^* = W^H x^* \tag{3}$$

However, this problem is ill-conditioned, because C doesn't have enough rows. To favor special properties of the solution, we introduce regularization terms to the cost as penalties, such as for this problem solving a sparsity problem:

$$x^* \in \arg\min x \in \mathbb{R}^n \frac{1}{2}\|CW^H x - b\|_2^2 + \epsilon\|x\|_1 \qquad z^* = W^H x^* \tag{4}$$

1.2 Mathematical aspects

To proceed to any algorithm solution, we need to understand the mathematical aspects of the problem. Let's take $A = CW^H$ and $f_1(x) = \frac{1}{2}\|Ax - b\|_2^2$ and $f_2(x) = \epsilon\|x\|_1$. The problem can be written as:

$$x^* \in \arg\min_{x\in\mathbb{R}^n} (f_1(x) + f_2(x)) \tag{5}$$

The objective function is a convex combination of two convex functions, and convex combinations preserve convexity. Moreover, all the constraints in the optimization problem are linear, which implies that the feasible set of solutions is convex. Hence, the optimization problem is convex. To show that $f_1 \in S_{0,L}^{1,1}$, we need to prove that f_1 is convex and its gradient is Lipschitz continuous. The function f_1 is a quadratic function and therefore convex. Note that

$$\nabla f_1(x) = A^T(Ax - b)$$

Let $x, y \in \mathbb{R}^n$. Then,

$$\|\nabla f_1(x) - \nabla f_1(y)\|_2 = \|A^T(Ax - b) - A^T(Ay - b)\|_2 = \|A^T A(x - y)\|_2 \leq \|A^T A\|_2\|x - y\|_2 = 2\|A\|_2^2\|x - y\|_2$$

(using Cauchy-Schwarz inequality) Therefore, ∇f_1 is Lipschitz continuous with $L = \|A\|_2^2$ where $\|A\|_2$ is the largest eigenvalue of $(A^T A)^{1/2}$.

2 Proximal gradient method

2.1 Basic method

We first need to calculate the proximal operator of the function f_2 given $\alpha_k = \alpha$ and $\forall i \in \{1, \ldots, n\}$:

$$\text{prox}_{\alpha f_2}(x) = \arg\min_{y\in\mathbb{R}^n} \left(f_2(y) + \frac{1}{2\alpha}\|y - x\|_2^2\right) = \arg\min_{y\in\mathbb{R}^n} \left(\epsilon\|y\|_1 + \frac{1}{2\alpha}\|y - x\|_2^2\right) = \arg\min_{y_i\in\mathbb{R}} \left(\epsilon|y_i| + \frac{1}{2\alpha}(y_i - x_i)^2\right)$$

To find the minimum, we consider the subdifferential:

$$0 \in \partial \left(\epsilon |y_i| + \frac{1}{2\alpha}(y_i - x_i)^2 \right).$$

The derivative is:

$$\frac{1}{\alpha}(y_i - x_i) + \epsilon = 0 \quad \Rightarrow \quad y_i = x_i - \alpha\epsilon \quad \text{for } y_i > 0 \quad \text{and} \quad \frac{1}{\alpha}(y_i - x_i) - \epsilon = 0 \quad \Rightarrow \quad y_i = x_i + \alpha\epsilon \quad \text{for } y_i < 0.$$

For $y_i = 0$, we check whether x_i is close enough to zero to be mapped to zero. This happens when: $|x_i| \leq \alpha\epsilon$. Thus, the solution is given by the soft-thresholding function, that we apply to the proximal gradient step:

$$y_i = \text{sign}(x_i) \max(0, |x_i| - \alpha\epsilon) \Rightarrow v_k = x_k - \alpha \nabla f_1(x_k) \quad \text{and} \quad [x_{k+1}]_i = \text{sign}([v_k]_i) \max(0, |[v_k]_i| - \alpha\epsilon)$$

2.2 ISTA

It does correspond to when $\alpha < 2/L$ and is a first-order methods that rely on function values and gradient evaluations for optimization. While first-order methods are often the only practical option for large-scale problems, the sequence x_k can converge slowly to a solution. This property is further confirmed by the proof that ISTA behaves like: $f_1(x_k) + f_2(x_k) - f_1(x^*) - f_2(x^*) \leq O(1/k)$.

Algorithm 1 Proximal gradient method

```
Input: $f_1 \in S^{1,1}_{0,L}$, $f_2 \in \Gamma_0(\mathbb{R}^n)$, $\epsilon > 0$, $x_0 \in \mathbb{R}^n$
while $opt\_gap > tol$ and $k < max\_iter$ do
    $v_k = x_k - \alpha \nabla f_1(x_k)$,    $[x_{k+1}]_i = \text{sign}([v_k]_i) \max(0, |[v_k]_i| - \alpha\epsilon)$
    $opt\_gap = \|x_{k+1} - x_k\|_2$,    $k = k + 1$
end while
```

2.3 FISTA

. When $\alpha \leq \frac{1}{L}$, we rather use FISTA, which behaves like: $f_1(x_k) + f_2(x_k) - f_1(x^*) - f_2(x^*) \leq O(1/k^2)$. The acceleration is due to the fact that the sequence x_k converges to the solution faster than ISTA.

Algorithm 2 Nestorov's accelerated proximal gradient method

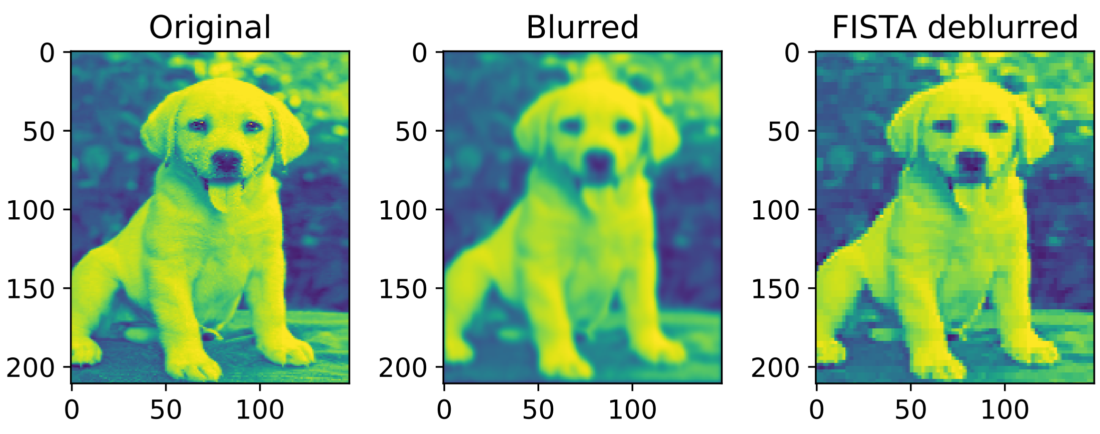
```
Input: $f_1 \in S^{1,1}_{0,L}$, $f_2 \in \Gamma_0(\mathbb{R}^n)$, $\epsilon > 0$, $x_0 \in \mathbb{R}^n, \lambda_0 = 0$
while $opt\_gap > tol$ and $k < max\_iter$ do
    $\lambda_{k+1} = \frac{1+\sqrt{1+4\lambda_k^2}}{2}$,    $\gamma_k = \frac{1-\lambda_k}{\lambda_{k+1}}$
    $v_k = x_k - \alpha \nabla f_1(x_k)$,    $[y_{k+1}]_i = \text{sign}([v_k]_i) \max(0, |[v_k]_i| - \alpha\epsilon)$,    $x_{k+1} = \gamma_k y_k + (1 - \gamma_k) y_{k+1}$
    $opt\_gap = \|x_{k+1} - x_k\|_2$,    $k = k + 1$
end while
```

2.4 Experiments

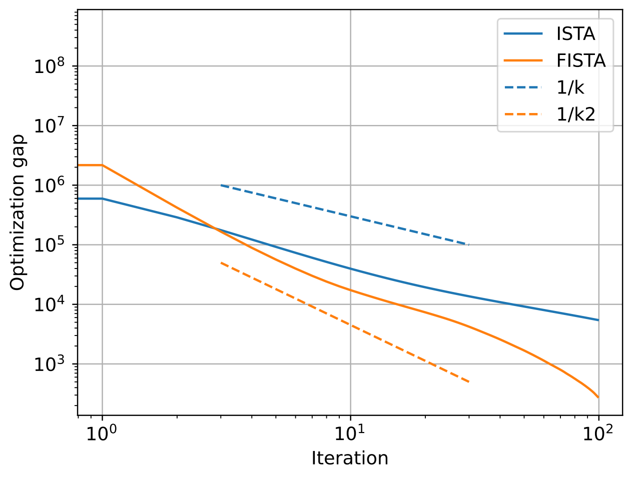
We will then take $\alpha = \frac{1}{L}$ for the programming part where L is the absolute value of largest eigenvalue of $(A^T A)$, that we can obtain by using the following function `A.eigs(neigs=1, symmetric=True)` of the `pylops` library. After coding my own ISTA and FISTA algorithm, we use the optimal solution of the already implemented `pylops.optimization.sparsity.fista` function as optimal value to run the tests of the methods

2.4.1 First results

For a first try, with $\epsilon = 0.1$ and $\alpha = \frac{1}{L}$, we can see that the FISTA method is deblurring the picture, it's not quite as detailled as the original picture but it's a good start. We can also see that the FISTA method converges faster than the ISTA method by introducing a mommentum term in the update of the x_{k+1} value.



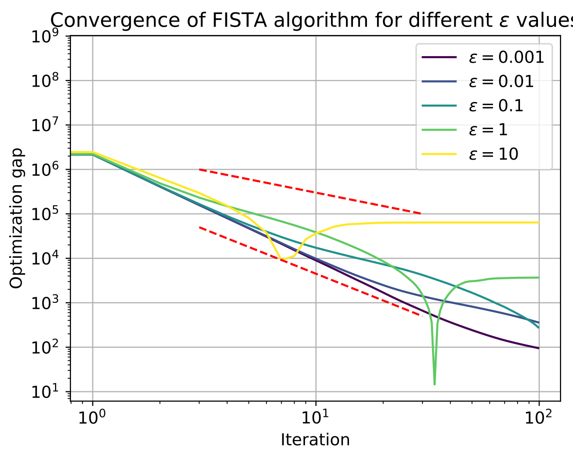
(a) Visualisation of the deblurring with FISTA



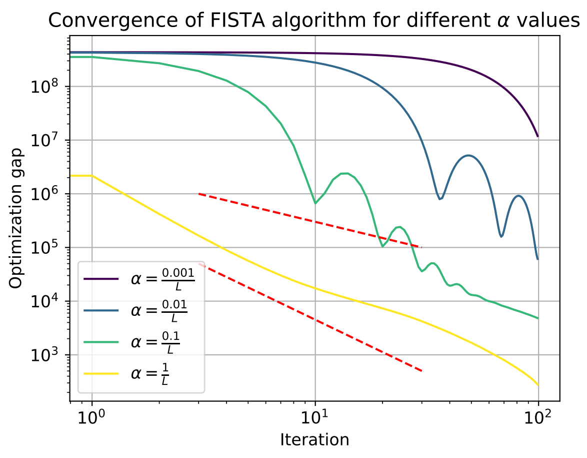
(b) Comparison of the methods convergence

2.4.2 Different values of ϵ and step sizes

We will make ϵ going from 0.001 to 10 because by default, it's at 0.1 and we will also make the step size going from $\frac{0.001}{L}$ to $\frac{1}{L}$ *(not more than 1 because it's the maximum value that can be taken for the FISTA algorithm to be valid)*.



(a) Comparison of the methods convergence for different values of ϵ

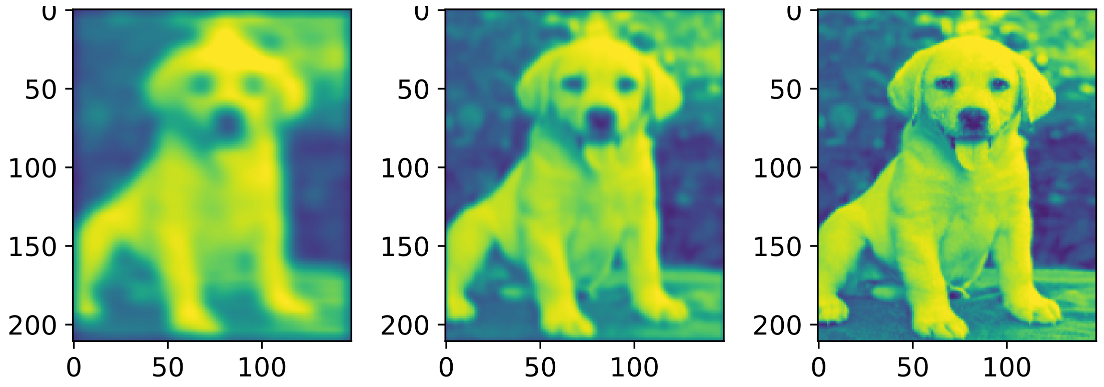


(b) Comparison of the methods convergence for different step sizes

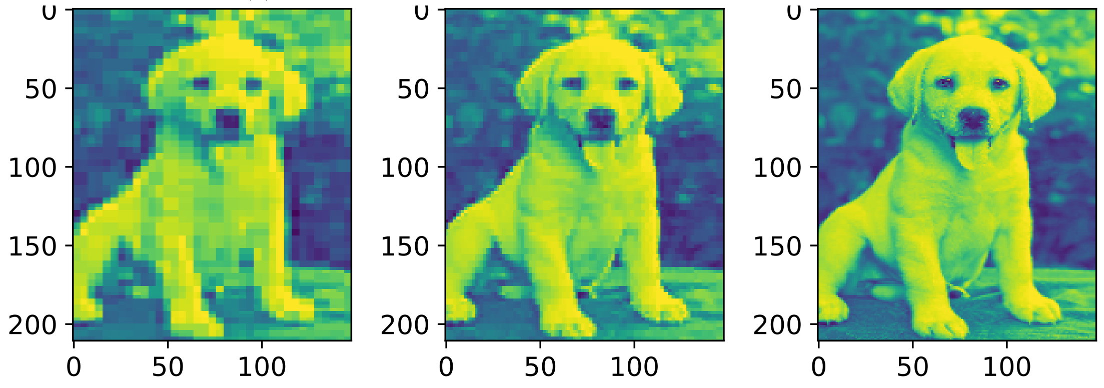
We can see that a different ϵ *which controls the trade-off between the data fitting term and the sparsity inducing term in the optimization problem* value doesn't change the convergence of the methods because the line are overlapping and all contains pretty much within the $\frac{1}{k}$ and $\frac{1}{k^2}$ bounds. On the other hand, the algorithm is very sensitive to the step size α because the convergence is not the same for all the values of α. An α near to $1/L$ provide us the fastest convergence rate out of all the values tested.

2.4.3 Different blurring

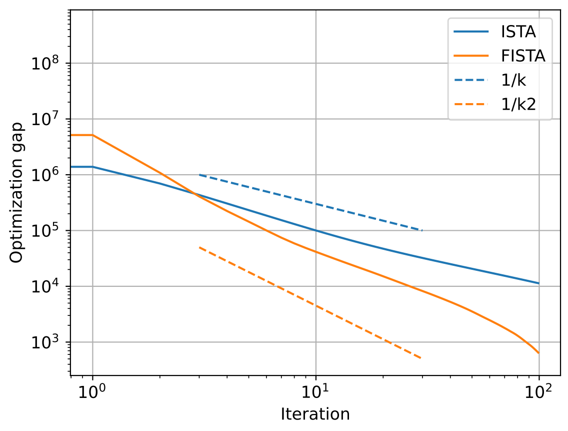
I took different blurring kernel by changing the coefficient in the exponential function of the kernel, by taking 0.1, 1 and 10. Here are first the different blurred images, and then below that are the corresponding deblurred images try,and convergence rates for the algorithms.



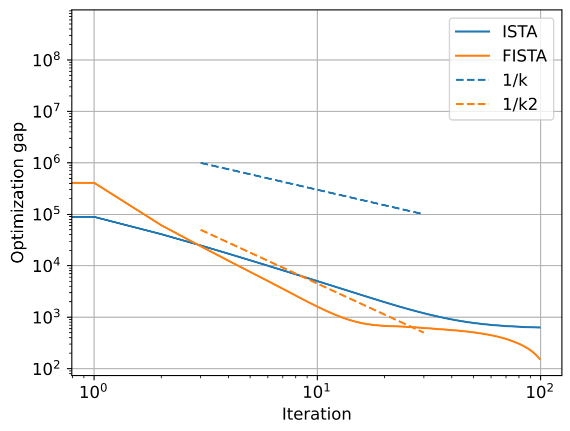
(a) Blurred images for exponential coef = 0.1, 1 and 10



(b) Deblurred images for exponential coef = 0.1, 1 and 10



(a) Convergence for exponential coef = 0.1 *(more blurring)*



(b) Convergence for exponential coef = 10 *(less blurring)*

We can clearly see, that blurring more *(smaller exponential coef)* the image makes the convergence slower and less accurate, that's normal because the problem difficulty increases with the blurring.

2.4.4 Sampling diminishing

Then I diminish the sample size, from 5 to 3 for the dog, diminishing the sampling means reducing the number of data points or samples from which the system makes its computations or decisions. This change has an impact on the dimension of the problem and the computational complexity.

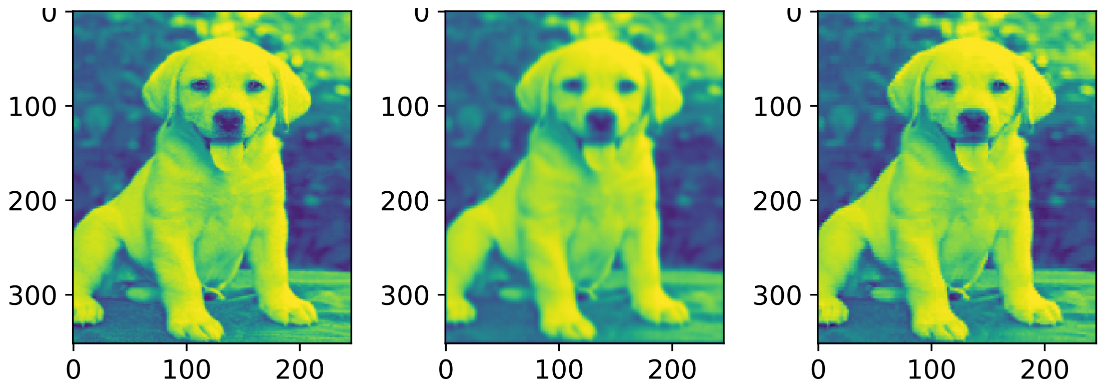


Figure 5: Deblurred images for different sampling size and the basic setup for image blurring

In this case, this reduction in sampling implies that I'm not working with the full set of data that represents the object (dog), which could lead to a loss of information. The problem dimensionality is affected because fewer samples mean fewer features to work with, the blurring isn't as intense as before. This could lead to a simplification of the problem, but it also risks making the analysis less accurate or even completely incorrect. When the sampling is increased, the number of data points grows, which increases the number of features or dimensions the model must process. As I sampled less points, the problem's dimensionality reduces. The more dimensions the problem has, the more computational resources are required. Here, it definitely was faster than before.

3 *cvxpy* library

The *cvxpy* library is a Python-embedded modeling language for convex optimization problems.

3.1 Basic method

Algorithm 3 cvxpy library algorithm

```
Input: $f_1 \in S^{1,1}_{0,L}$, $f_2 \in \Gamma_0(\mathbb{R}^n)$, $\epsilon > 0$, $x_0 \in \mathbb{R}^n$
$x = cp.Variable(n)$
objective $= cp.Minimize(f_1(x) + f_2(x))$
prob $= cp.Problem($objective$)$
prob.solve()
```

These methods have a cubic complexity in the number of variables, meaning they become infeasible for large-scale images, because it's using an interior-point method to solve the problem, which require solving large linear systems at each iteration. To mitigate this, we sample down the image to reduce the number of variables, ensuring the solver can handle the problem efficiently.

3.2 Experiments

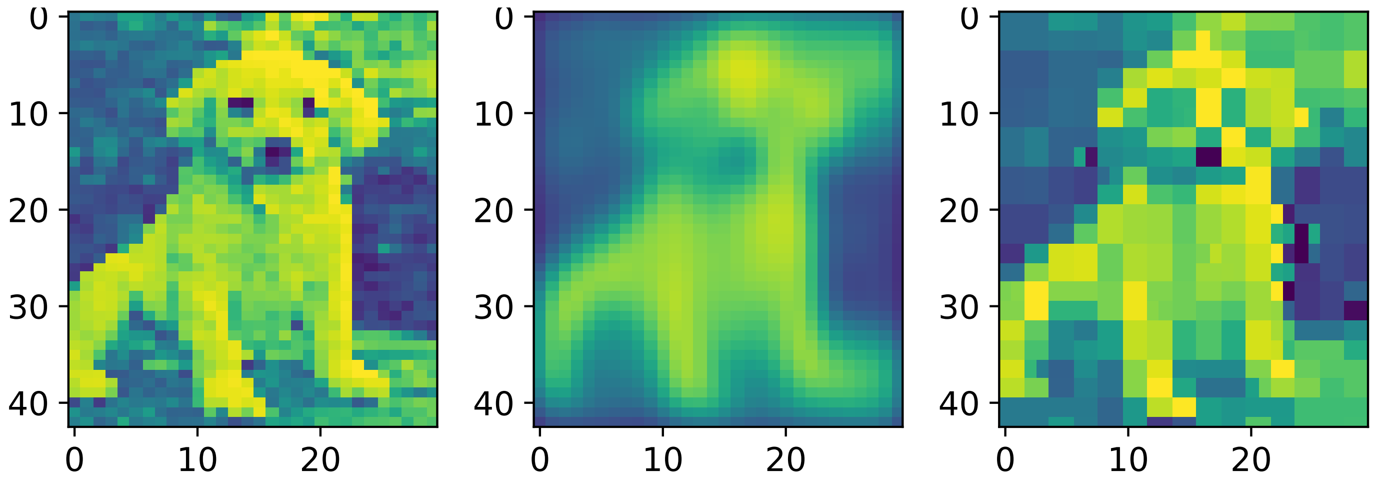


Figure 6: Original, Blurred and Deblurred images for sampling size = 25

We can see that the deblurring is not as good as the one obtained with the ISTA and FISTA methods, we don't see the dog as much on the result, that's mainly because the downsampled image loses fine details. And not only the result is worse, but the computation time is significantly higher than the ISTA and FISTA methods. While the ISTA and FISTA took around 6.7s to compute, for both of them, the deblurring for a sample downing of 5, the cvxpy method took 22s to do the same thing for a sample downing of 25.

Conclusion

In this report, we have explored the problem of image deblurring as an inverse problem, where the goal is to recover a sharp image from a blurred observation. We introduced a mathematical model for the problem and analyzed its convexity, highlighting the challenges posed by ill-conditioning.

To solve this problem, we employed optimization techniques, specifically the proximal gradient method, both classic ISTA and accelerated FISTA. We derived the soft-thresholding operator for the ℓ_1-regularized least squares problem and implemented both ISTA and FISTA, showcasing their theoretical convergence properties.

While ISTA provides a simple yet effective iterative approach, FISTA significantly accelerates convergence through momentum-based updates. These methods demonstrate the power of convex optimization in tackling real-world problems such as image restoration, but also the limitations of some pre-built libraries like *cvxpy* for large-scale problems.

Future work could focus on extending these methods to more advanced models, such as incorporating non-convex regularization techniques or leveraging deep learning-based priors for improved deblurring performance.