COMP0114 - Coursework 2

Due: Wednesday February 22nd, 2023 at 16:00

Week 1:

1. We will work with the cameraman image, of dimensions 256x256. The blurring function we use is MATLAB's `imgaussfilt`. Here is the effect of this function on our image for $\sigma = 0.5$.

Figure 1: Our original image (left), and the blurred version using imgaussfilt (right)

We will now apply blur AND noise to the image, and try to deconvolve it. A first method is suggested in c), using normal equations. Here is the equation we are looking to solve (for f_α, which would be the estimate of the true image):

$$(A^T A + \alpha I) f_\alpha = A^T g$$

We identify the normal equation for a ridge regression i.e using the norm-2 squared of the estimated image as the penalty term. A represents the blurring matrix. But here, we have not stored the blurring effect as a matrix, we are using a function instead. Therefore, we cannot invert the term on the left.

As recommended, we will apply the left transforms (blurring and addition of a term with alpha) and the right transform (blurring, because the convolution is self-adjoint) respectively to our estimate and our noisy and blurry data, and we will optimize over f_{alpha} using different methods each involving systems of linear equations: preconditioned conjugate gradients (`pcg`), generalized minimum residual method (`gmres`) and least-squares (`lsqr`). Let's note that for `lsqr`, we will use an augmented equation that should perform better than the other two.

In terms of numerical implementation, for `lsqr`, we have to provide the transpose of the augmented matrix. How do we manage the dimensions? The vector in input is $515 * 256$ (because we have g on the top part, and 0s to fill the bottom part). For the non-transpose, we only take the first half of the input vector and reshape it to $256 * 256$. The output of the non-transpose will be $512 * 256$ because of the augmented matrix. For the transpose case, we take the first part of the vector and blur it, and we take the second part and we multiply it by the $\sqrt{\alpha}$ term, and we

finally concatenate the first output with the second one to still have a $512 * 256$ output.

The results obtained can be found in Figure 2:

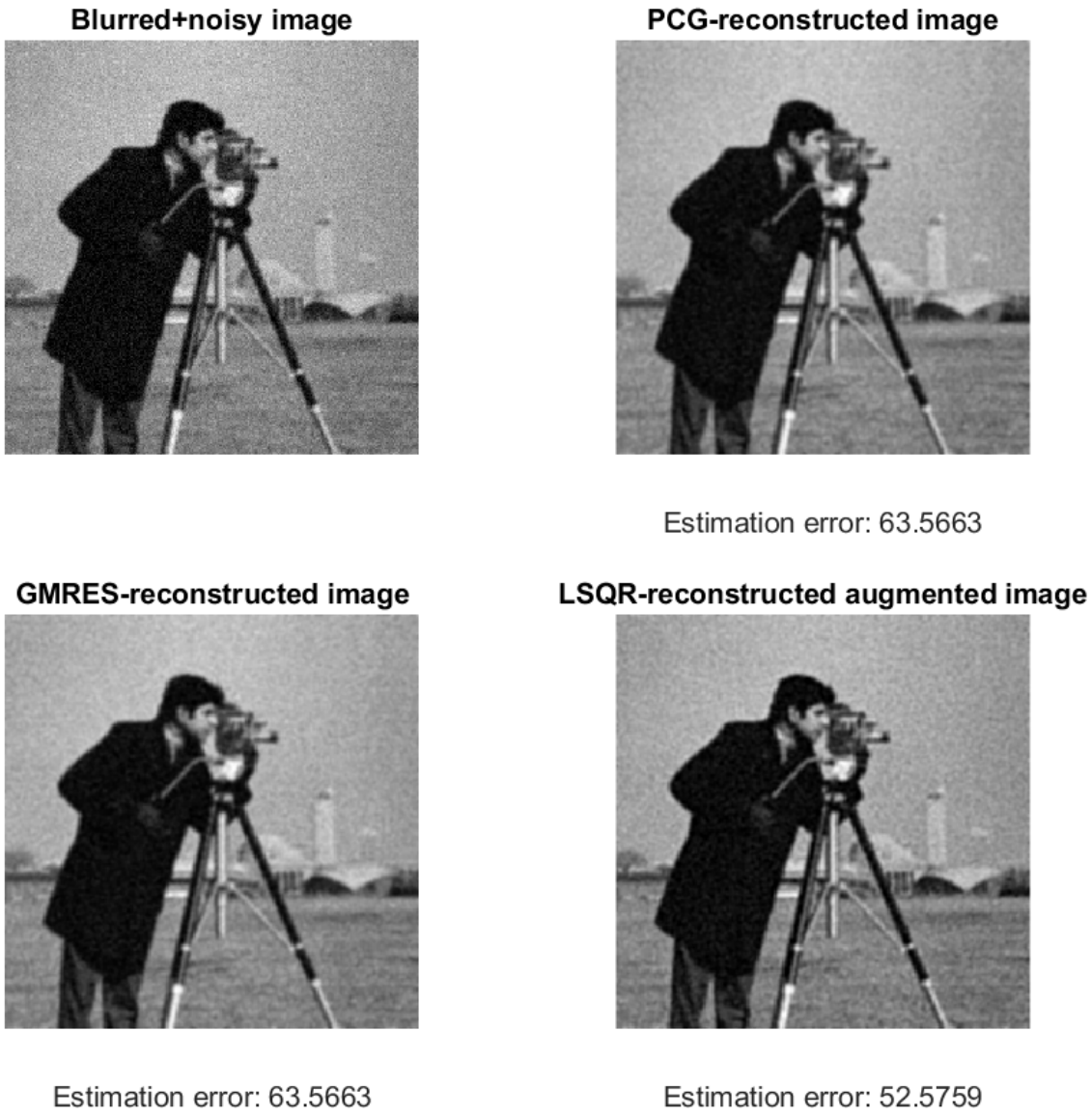

Figure 2: Blurred+noisy image (top left), result with PCG (top right), with GMRES (bottom left), and result with the augmented equation using LSQR (bottom right). Parameters used: $\sigma = 1$, $\theta = 0.05$, $\alpha = 1$, regularized with the L2-norm.

We can see that the noise level has been reduced. It is hard to tell if a certain amount of blur has been removed yet. What is sure, is that regularization takes care of the noise, and that the augmented equation does a better job than the `pcg` and `gmres` alone. Here are the performances for each algorithm:

- `pcg converged at iteration 12 to a solution with relative residual 6.9e-11.`
- `gmres converged at iteration 12 to a solution with relative residual 6.8e-11.`
- `lsqr stopped at iteration 20 without converging to the desired tolerance`

```
1e-10 because the maximum number of iterations was reached. The iterate
returned (number 20)has relative residual 0.38.
```

We can see that `pcg` and `gmres` are the fastest, and that for a same tolerance and number of approximation, `lsqr` does not achieve the same objective value. The reconstruction for `lsqr` looks better than the one from the others. Let's note that `1e-10` is a really high value for the objective score, and that the score obtained by `lsqr` can be considered as good, in the following parts, the scores will be around `10-1`/`10-2` too.

In this part, we have seen that regularization can be used to reduce the noise in the image. It can nevertheless vary depending on the equation we optimize for, the value of the parameter, and the method of optimization. We will focus on the value of α in the next part.

Week 2:

2. In this part, we have kept $\sigma = 1$ and $\theta = 0.05$.

For the discrepency principle, we have started with the following equation:

$$\frac{1}{n}\|\boldsymbol{r}_\alpha\|^2 - \sigma^2 = 0 \qquad \text{where} \quad \boldsymbol{r}_\alpha = g - A\boldsymbol{f}_\alpha$$

We have first computed $\boldsymbol{f}_\alpha$ with one solver from the last part (here, PCG). We do that for every α in a linspace array to obtain a $DP(\alpha)$ curve, and then we use MATLAB's `fzero` to find the zero of this $DP(\alpha)$ function. This root α is the optimal in the sense of the DP criterion.

The α value found by our function is: 0.061241.

We are also asked to plot an L-curve, consisting of the points $\{\|\boldsymbol{r}_\alpha\|^2, \|\boldsymbol{f}_\alpha\|^2\}$ (because here, our regularizer is Ridge i.e a squared L2-norm). Using once again our previous PCG solver to compute the $\boldsymbol{f}_\alpha$ values for different α, we obtain the graph shown in Figure 3.

We know that the optimal α value is found at the point of highest curvature of the curve (i.e usually the only corner of the curve). Here, looking at the coordinates, we can estimate an α of: $0,034825$.

The two values are coherent, although the α value computed by the DP principle is almost twice as high as the one found on the L-curve. Still, these values stay low. Given that the DP equation uses directly the noise θ value, we may want to give more importance to this result than the L-curve result.

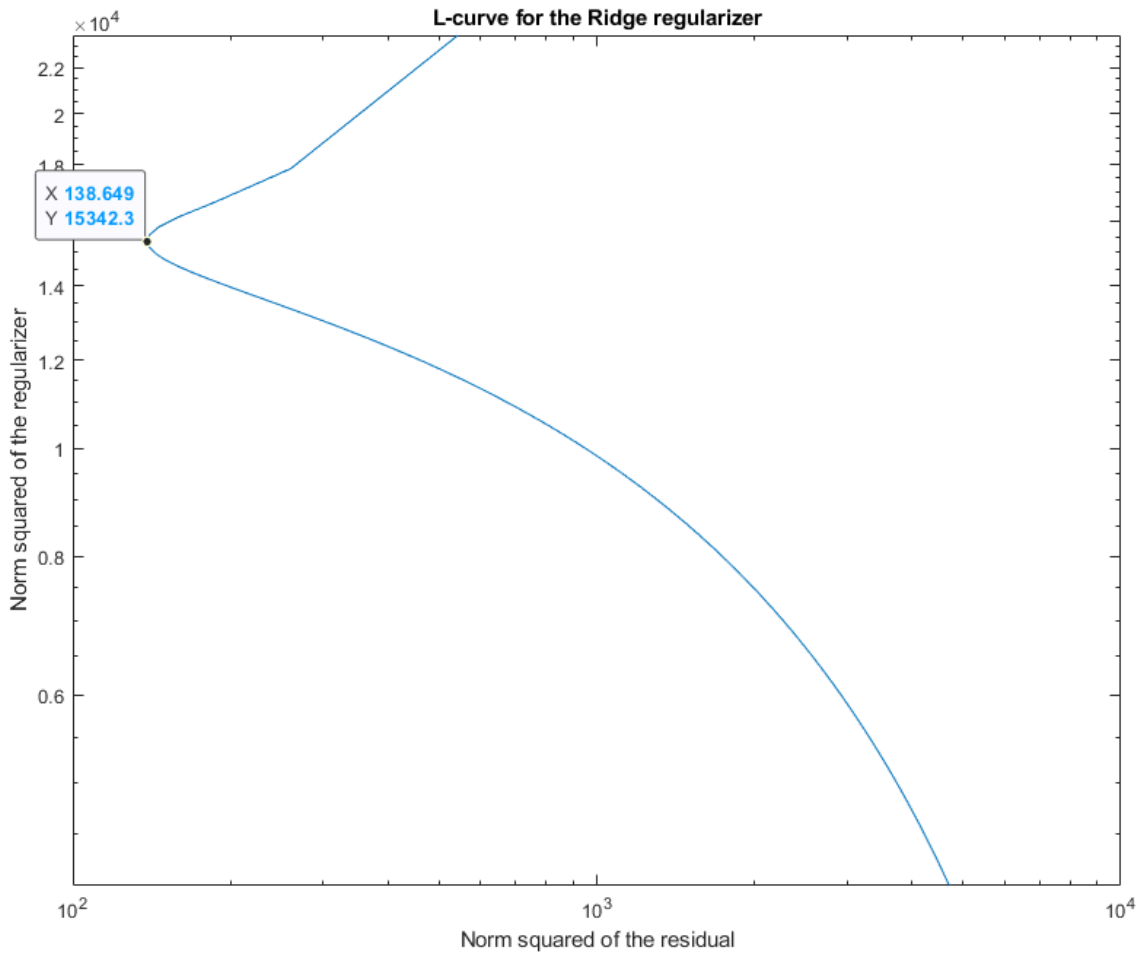

Figure 3: L-curve for the Ridge regularizer for parameters $\sigma = 1$ and $\theta = 0.05$

3. Now onto a new regularizer: the L2-norm of the image gradient. In the same manner as the blurring operator, we build the gradient operator using MATLAB's `diff`. We have to define a new function to get the full gradient, because `diff` only computes the difference along one axis. Moreover, it removes one dimension (either horizontally for the x axis, or vertically for the y axis), so we have to fill this column/row to be able to add the two gradient maps. What we do is therefore computing the x and y gradients separately using `diff`, we create a $256 * 256$ array where we fill respectively the rows 2 to 256 and the columns 2 to 256 with these gradients, and we copy the 2nd row (resp. column) as the 1st row (resp. column) of the new array. We perform twice this operation per axis. Finally, we add these two maps to obtain de Laplacian. The results we have obtained are shown in Figure 4.

   The estimation error has dropped, we obtain better noise removal, and we get to see more blur, which should be the case without noise. Adding the gradient as a regularisation thus enhances the performance. But, now we see that the edges are getting more blurred too, and this is not really what we want, we would need to fine-tune every decision depending on the gradient value (edge/not edge, direction etc..).

   To choose an appropriate α, we run the same routine as in question 2. The DP procedure gives us the following result: $\alpha = 0.017463$.

   The L-curve computed for this new regularizer can be found in Figure 5 below.

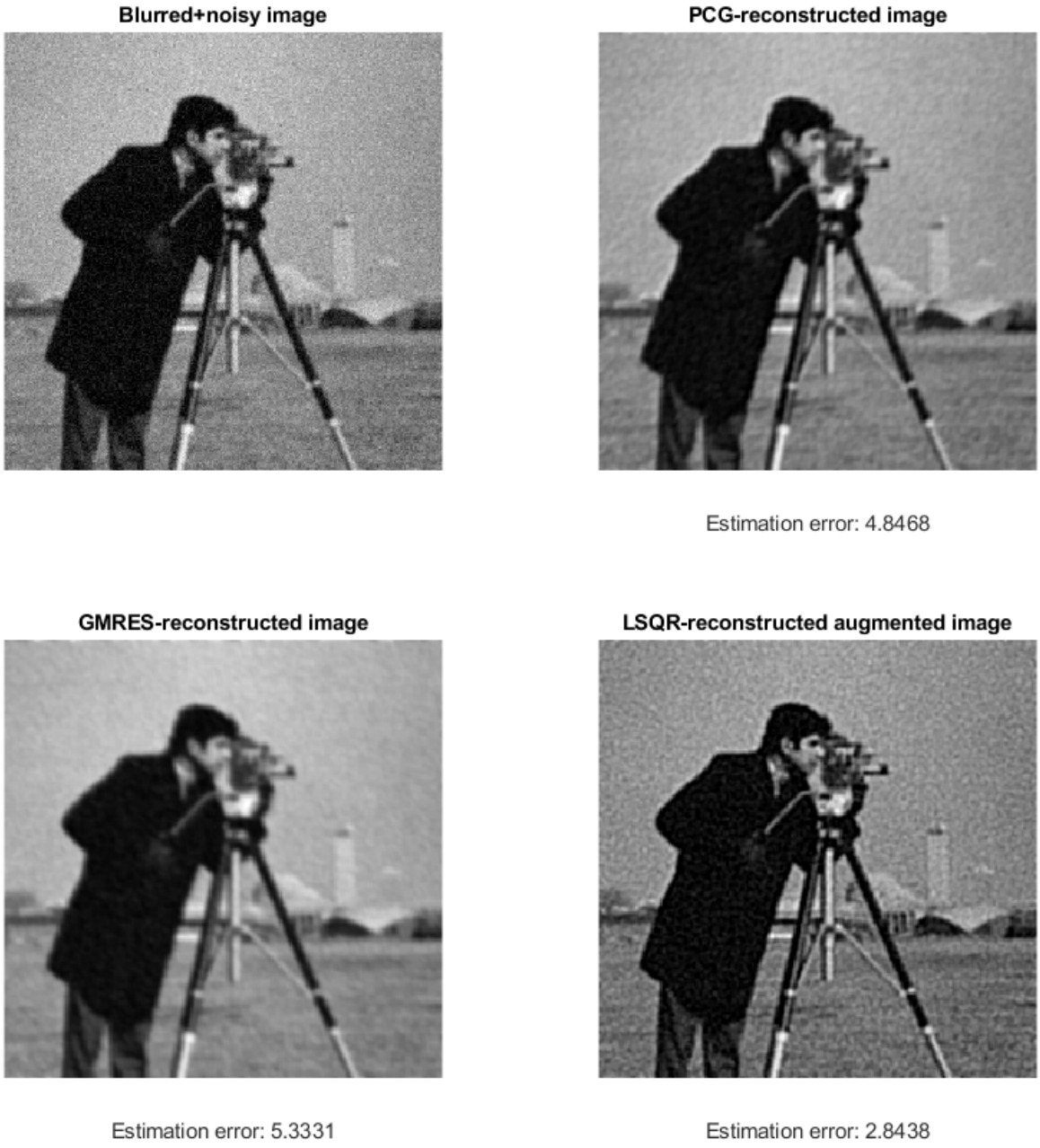

Figure 4: Blurred+noisy image (top left), result with PCG (top right), with GMRES (bottom left), and result with the augmented equation using LSQR (bottom right). Parameters used: $\sigma = 1$, $\theta = 0.05$, $\alpha = 1$, regularized with the L2-norm of the gradient.

Here, we can compute: $\alpha = 0.074625$. This value is higher than the DP estimation, but the values remain low. One reason that could explain this is found in looking at the graph. The curve is not as smooth as previously, and there seem to be many instabilities when taking higher values of α, and therefore `fzero` may find different root solutions (it actually does! but when trying the other values found by it, the estimation score goes higher, so we stick to the lowest α value). It might be due to the fact that the values of the gradient map are much lower than the original image, and then the regularization often makes the DP criterion close to zero.

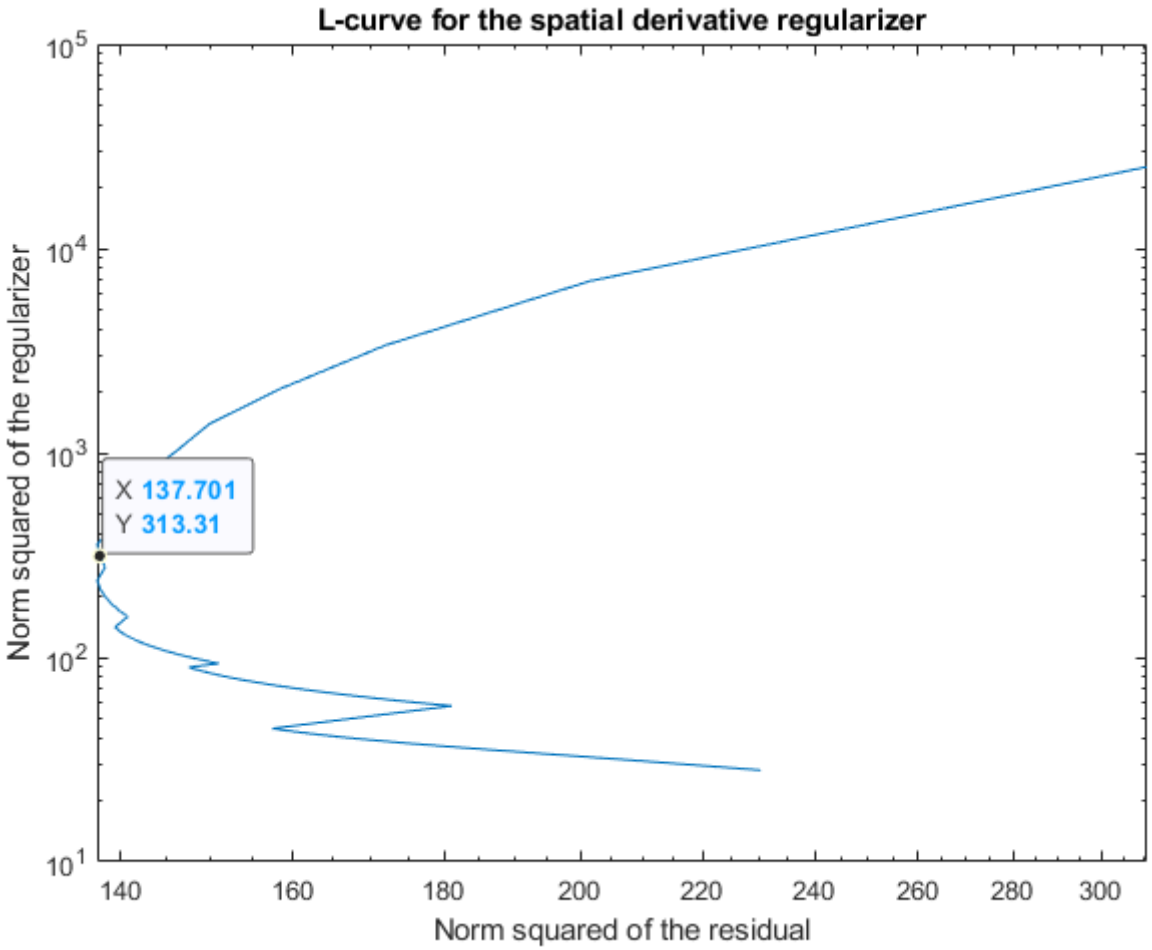

Figure 5: L-curve for the gradient regularizer for parameters $\sigma = 1$ and $\theta = 0.05$

Week 3:

4. We have identified the problem in the last part: do not process the edge in the noise removal task. The solution suggested here is to use a diffusivity term, in particular the Perona-Malik function. Let's look at the output of such function in Figure 6.

   We can easily see that the edges have the lowest values, and so when multiplying term-by-term with the laplacian, we would only have values on the non-edge terms, thus processing only the low-frequency parts (background and simple textures) which is what we want. We identify a new hyperparameter: the percentage of normalization of the output. The higher it is, the more edge-selective it is. We will stick to 20% as it gives a good balance between edges and non-edges.

   Let's see this new function in action: in Figure 7, we show the comparison between the Laplacian operation (simple blurring) and the Laplacian operation with the Perona-Malik weighting.

   We can see that on the left part, when using Perona-Malik, the edges are not blurred. We also observe that our choice of hyperparameter is ok, and that there is a sufficient number of edges staying sharp.

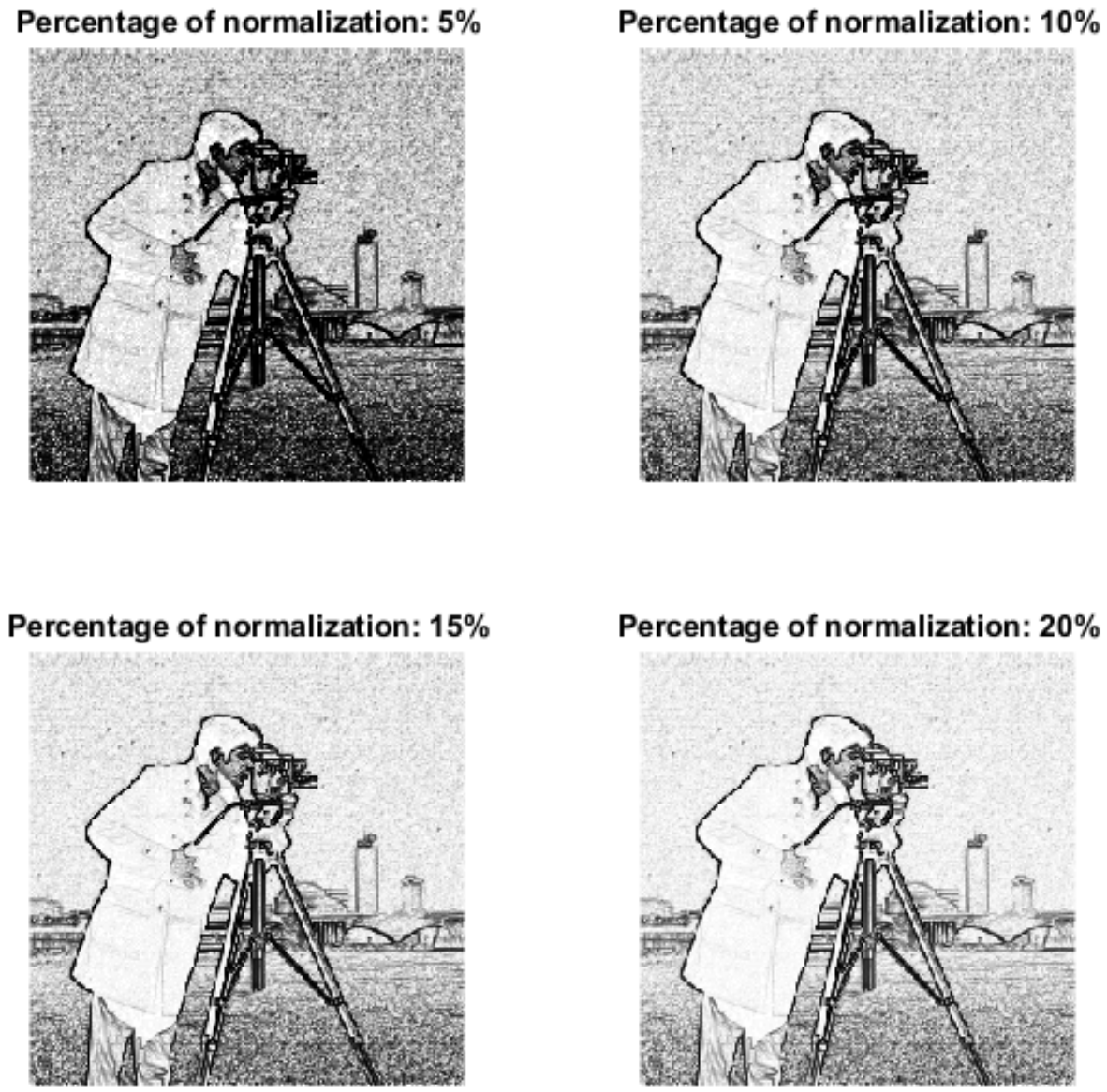

Figure 6: Perona-Malik function output for different percentages of normalization

Now that we have presented our new tool, let's apply it to our solvers, and choice of the best regularization parameter. The output results of PCG, GMRES and augmented LSQR are presented in Figure 8.

We achieve once again a better estimation error, and when comparing with the results from the previous part, we can see that the edges stick out more in our new results, which is exactly what we wanted to achieve. Let's note that the augmented LSQR solver is particularly sensitive to the weighting, we can clearly see some artifacts a bit everywhere representing the weightings in the sky which are not correct: that is where using the edges of the data compared to the edges of the ground truth impacts our result. We might think about an algorithm that could remove weightings when the predicted edges are not close enough to each other, to remove the abnormalities in the sky for example.

We have run into numerical problems when computing these results. Because we weight the Laplacian output from the start, we may obtain negative values, and so we have to provide the solvers with a starting point (in our case, the PCG/GM-

RES/LSQR estimates from part 1 run on 1 iteration), so it does not start with the null vector and ends up with a singular matrix.

Let's take a look at the L-curve in Figure 9 (our DP routine does not converge here due to numerical errors) to determine the best α value.

We identify on the curve $\alpha = 1.4975$, which is a much higher value than what we had found previously. It might be due to the fact that now that the edges are preserved intact, we can accentuate the denoising of the outer parts.

Let's finish testing our method by iterating it. What we expect: by modifying the diffusivity at each iteration, we should get at each time a better estimation of the edges (because the output is less blurred), and therefore the unblurring operation should focus at each iteration on new areas that were previously considered as edges (false negatives in the sense of the Perona-Malik function). The only change we make that instead of providing $A^T g$ as the PCG ground-truth (the blurred+noisy estimate), we provide directly the ground-truth to indicate to the algorithm that we are looking to completely remove the blur. Also, we provide f_{i-1} as a starting point when computing f_i. We have tried using both the isotropic and the anisotropic routines. The results can be found in Figures 10-12.

What we observe is that the procedure converges in 1 iteration, the gradient-only routine gives a cleaner result than the gradient with diffusivity, and the latter is "blocked" by the diffusivity map (as in it cannot get it cleaner than it already is). We tried using the gradient-only method first to see if the gradient with diffusivity method could continue and improve our result with a cleaner diffusivity map, but the gradient-only method seems to already reach its maximum estimation in 1 iteration, and the gradient with diffusivity method just cannot improve it.

The gradient with diffusivity has trouble improving the diffusivity map, because there are still so many artifacts in the sky, and the grass.. we would need to threshold some values or as suggested earlier, remove the black spots that are not connected to other edges. The reconstruction still looks ok.

The reconstruction from the gradient-only method is very good, the score is only bigger because the intensity of the sky is higher compared to the original.

By looking at the diffusivity map, we can understand why the gradient with diffusivity method struggled at the start of our previous tests: the noise has propagated all over the diffusivity map. It shows once again that we need a better estimate for the edges at the start, or we could try a regularizer without gradient at the start.

Otherwise, if we had to choose a criterion to stop the iterations, we could have used a condition on the score of the reconstructed image or the diffusivity map (if we can have access to a denoised estimate), or stop when the score does not improve after n iterations. We can also take a look at the value of the regularizer and see its evolution.

Conclusion:

To conclude, in this coursework, we have investigated different regularizers that perform denoising and deblurring. We have seen how important it is to take a look at our tools (Laplacian, Perona-Malik) on separate examples and functions, to

ensure that they will work well once implemented in our routines. We have also seen that even if we have found the best regularization method, we still have to run more tests to determine the best regularization parameter. We might also need to alternate between different regularization methods depending on our starting data, and the evolution of the estimates. Finally, we have seen that by using functions (here `diff` and `imgaussfilt`), we can still use the matrix/vector-based methods, and therefore save storage and even perform operations that could not have been possible on high-resolution images.

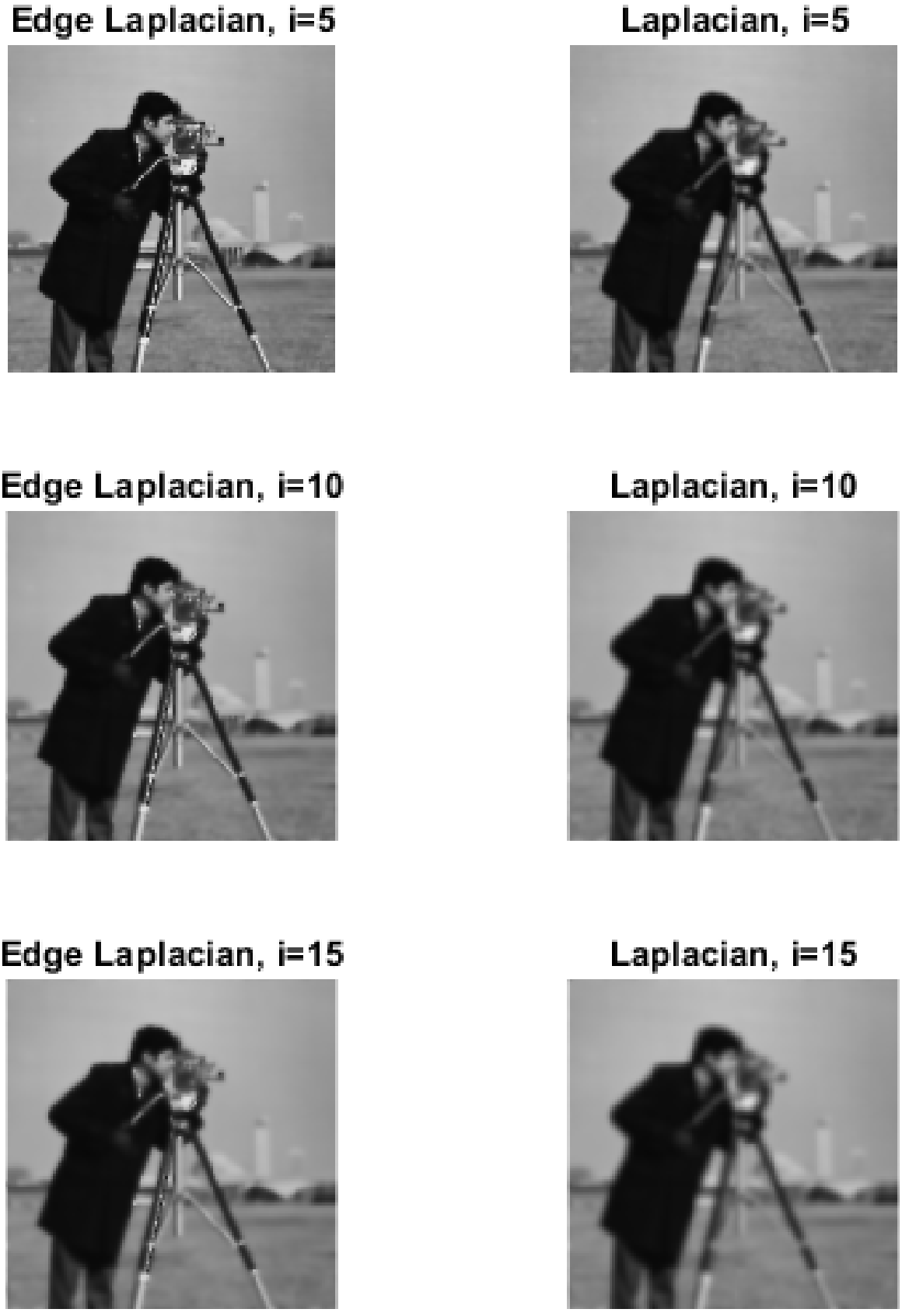

Figure 7: Laplacian blurring+Perona-Malik weighting (left, percentage of normalization: 20%) and Laplacian blurring (right) for a different number of iterations on our source image

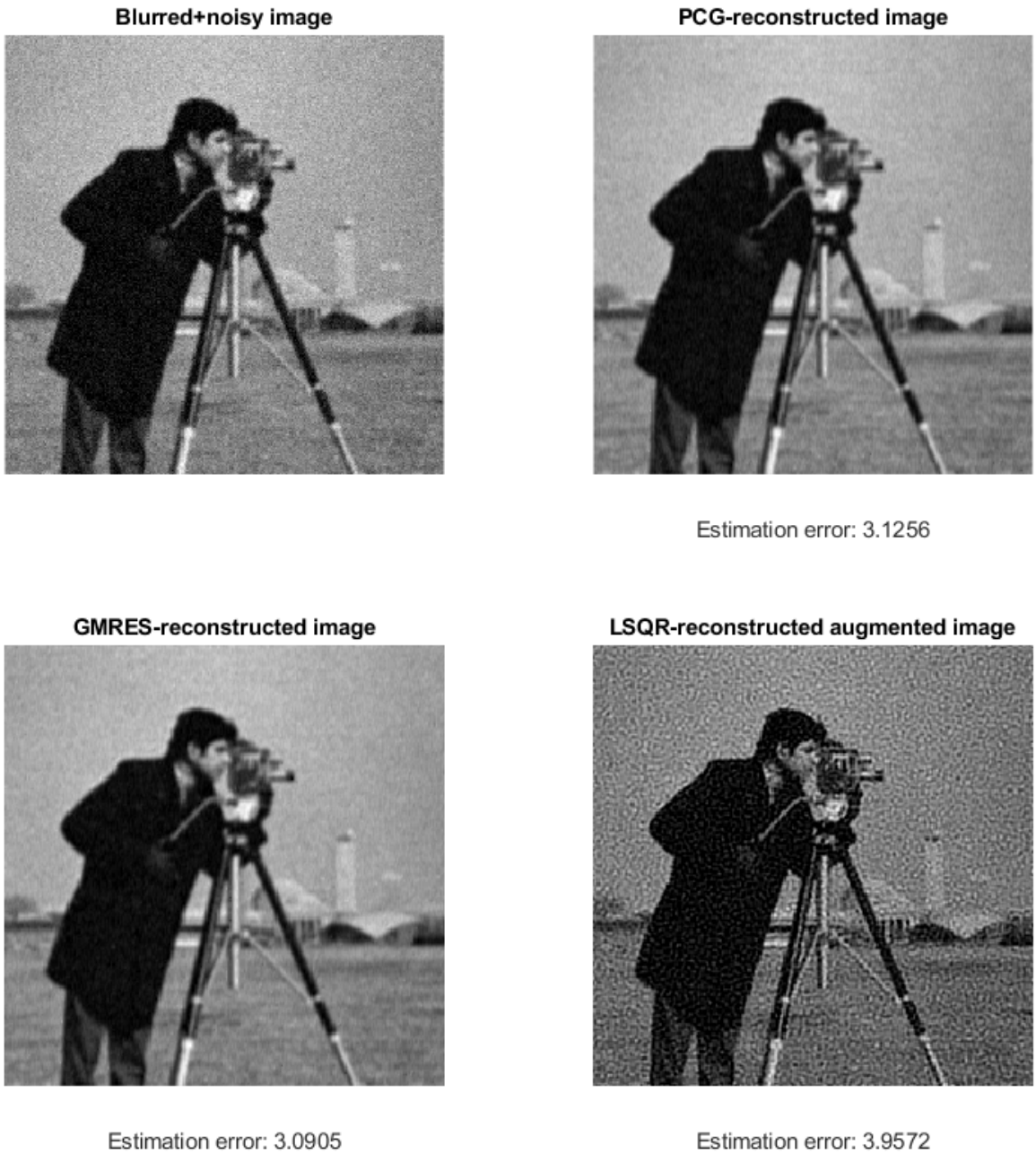

Figure 8: Blurred+noisy image (top left), result with PCG (top right), with GMRES (bottom left), and result with the augmented equation using LSQR (bottom right). Parameters used: $\sigma = 1$, $\theta = 0.05$, $\alpha = 1$, regularized with the L2-norm of the gradient with Perona-Malik weighting.

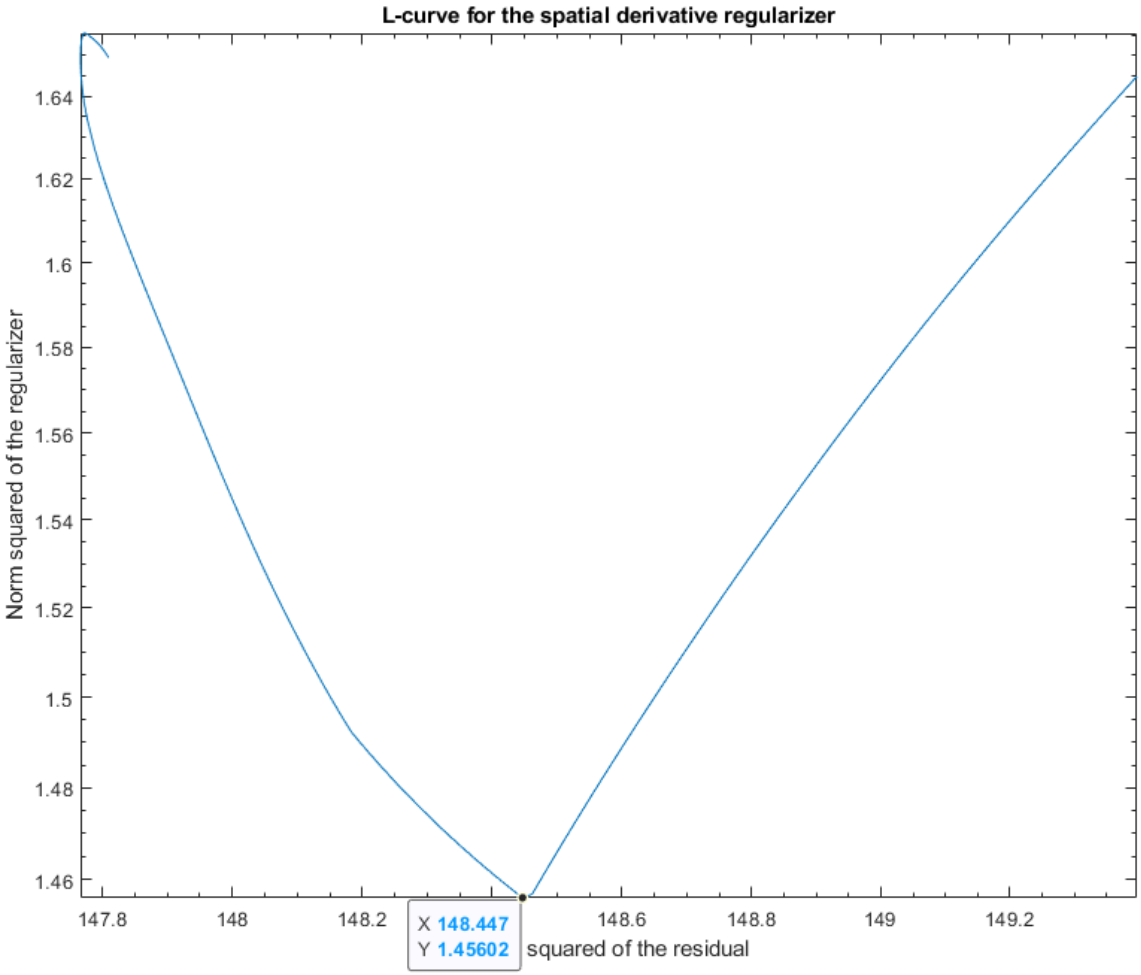

Figure 9: L-curve for the gradient with diffusivity regularizer for parameters $\sigma = 1$ and $\theta = 0.05$

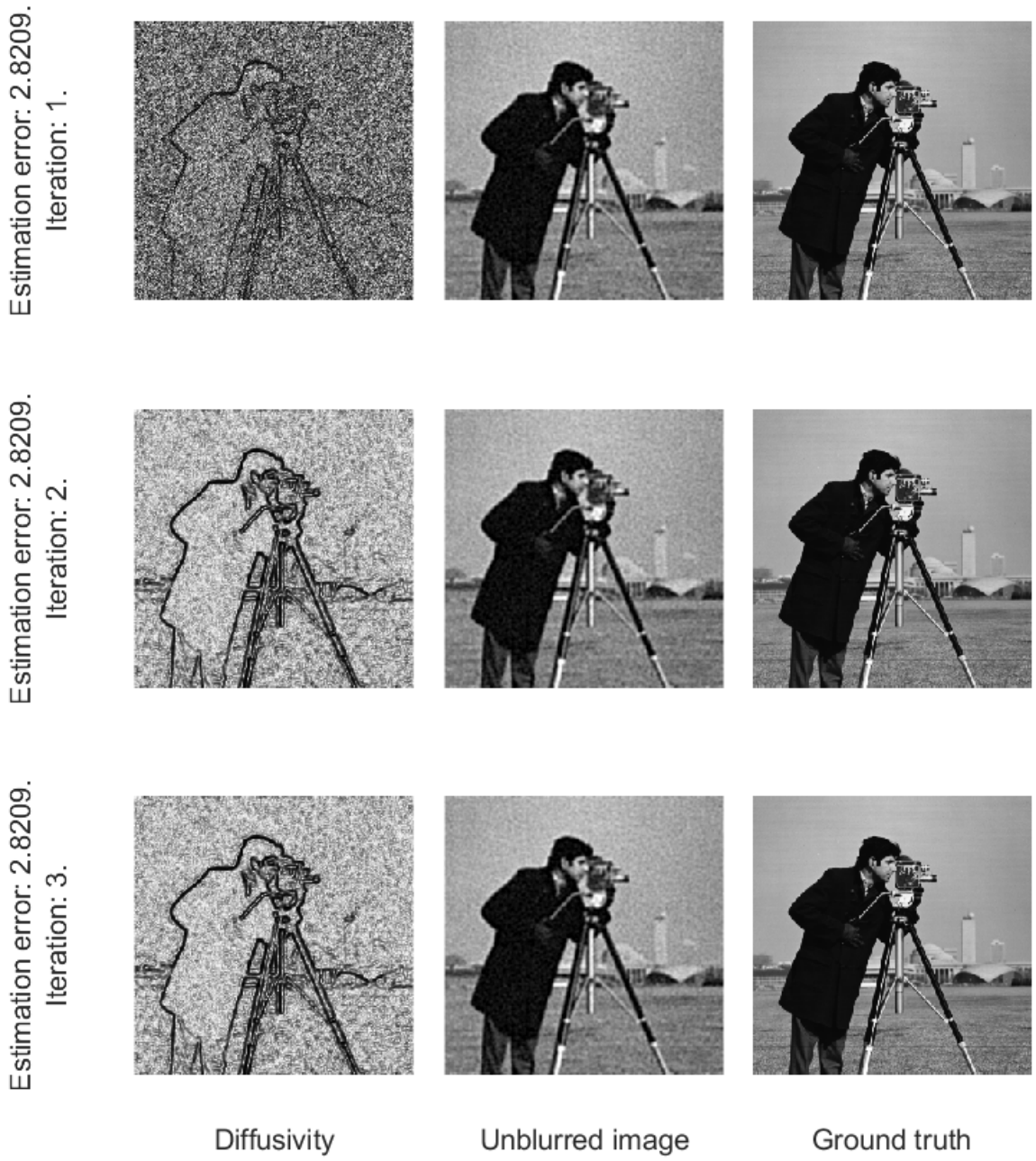

Figure 10: Iterating our deblurring method with the gradient with diffusivity regularizer for parameters $\sigma = 1$, $\theta = 0.05$, and $\alpha = 1.4759$.

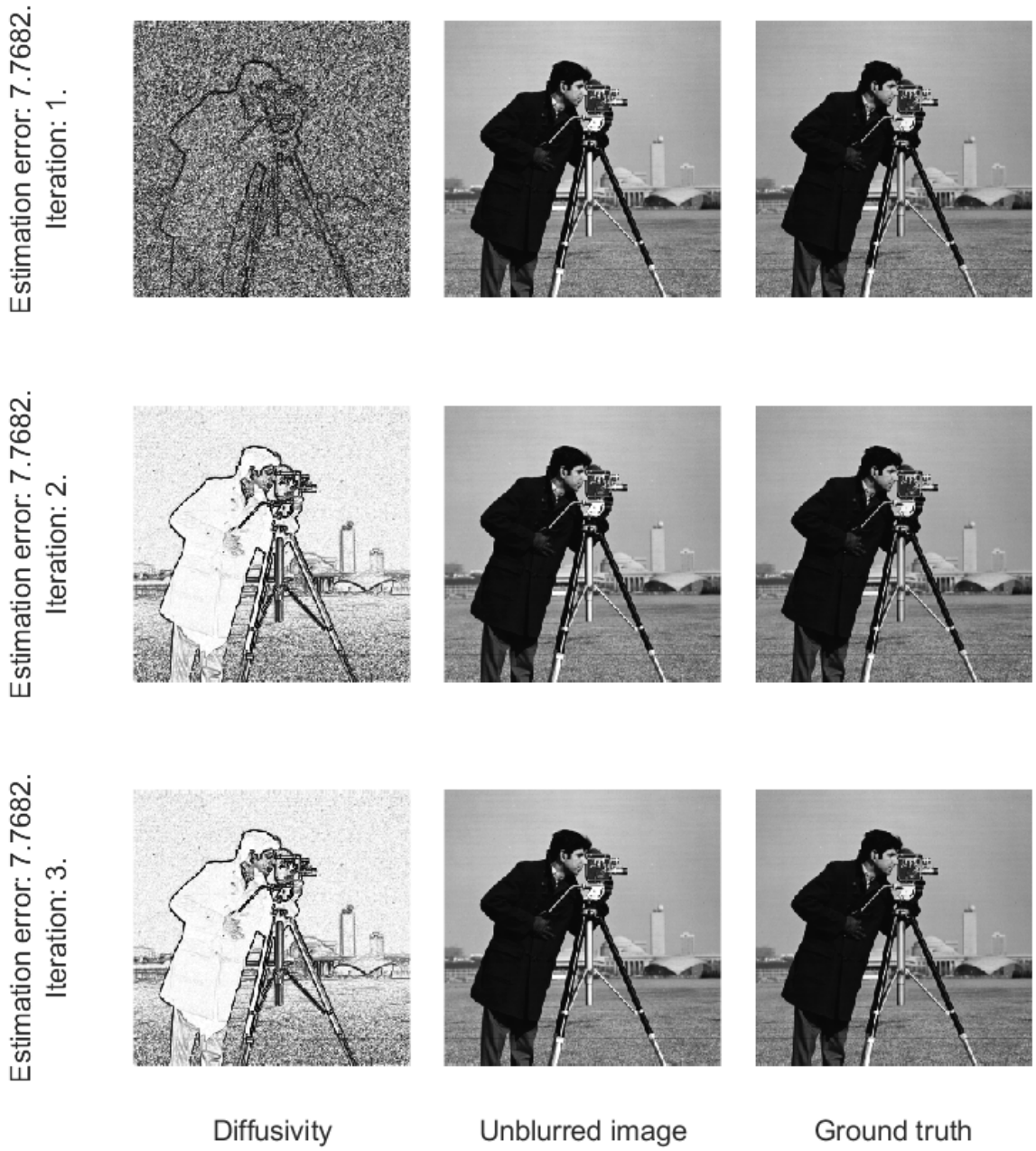

Figure 11: Iterating our deblurring method with the gradient regularizer for parameters $\sigma = 1$, $\theta = 0.05$, and $\alpha = 1.4759$.

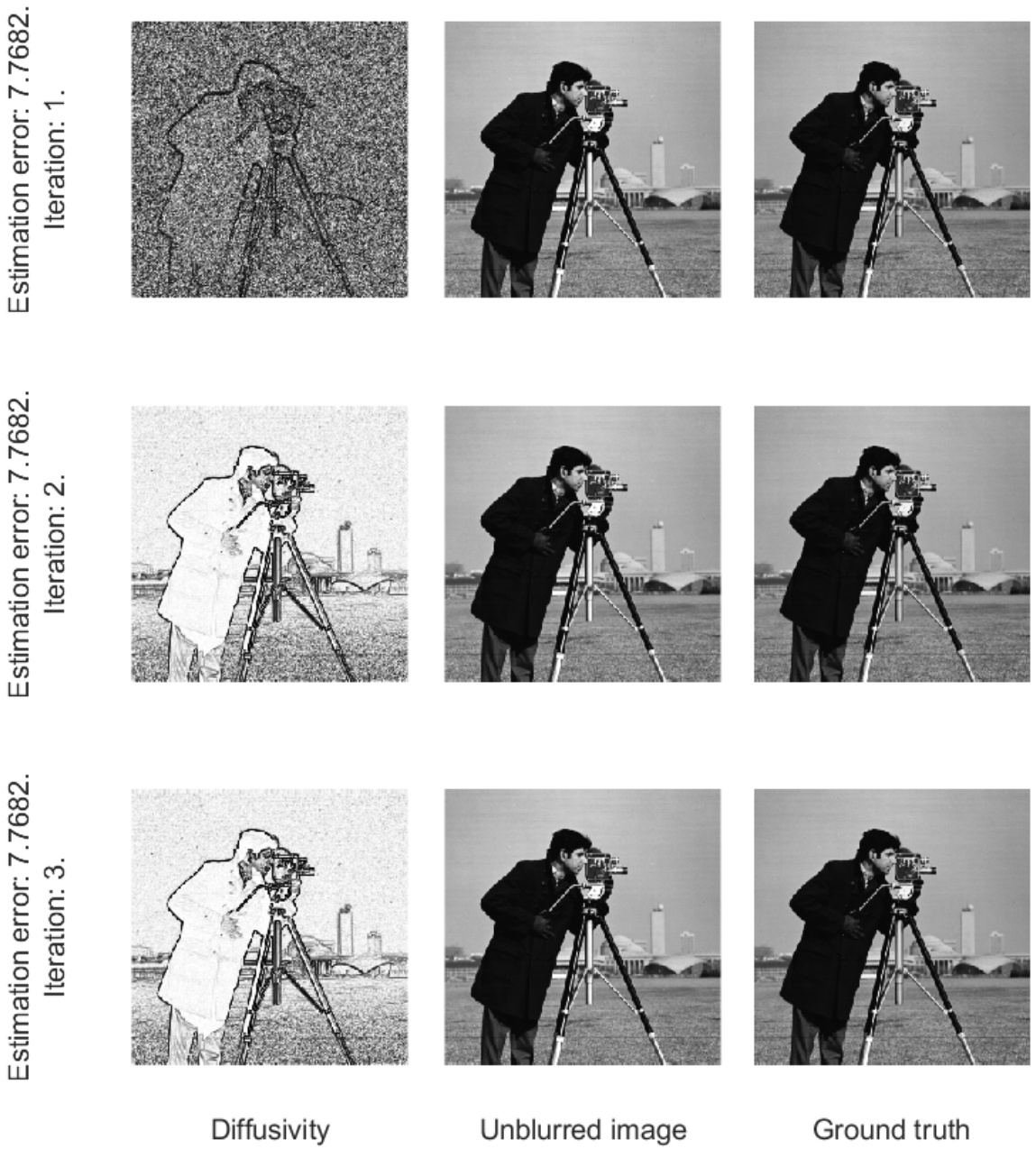

Figure 12: Iterating our deblurring method with the gradient regularizer ONLY for the first iteration, and then the gradient with diffusivity regularizer for parameters $\sigma = 1$, $\theta = 0.05$, and $\alpha = 1.4759$.