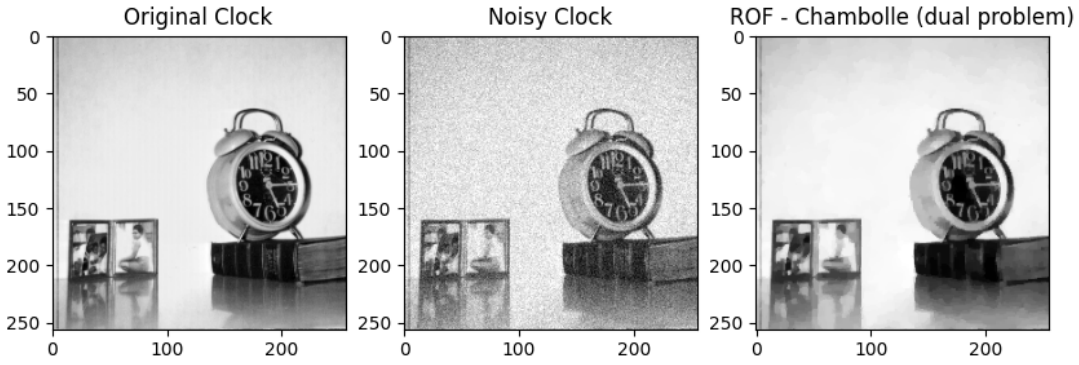
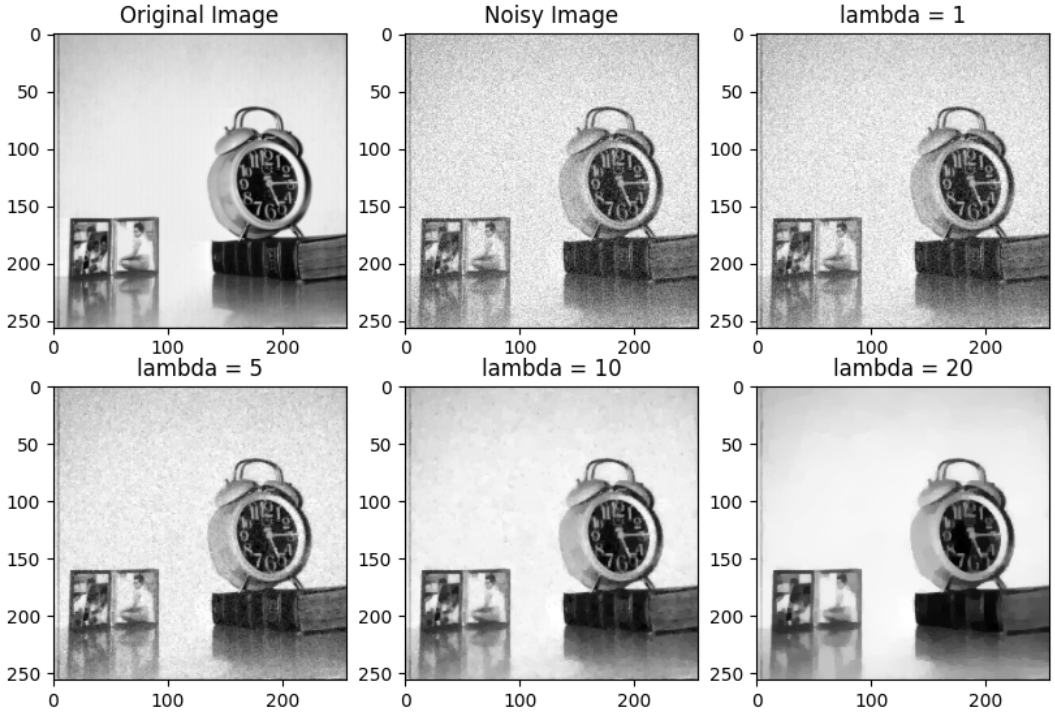
(3) (b)

The Chambolle algorithm implemented in Python, using the default parameters (lambda = 15, stopping tolerance = 1/1000, tau = 1/8) converges in 793 iterations using the clock image:



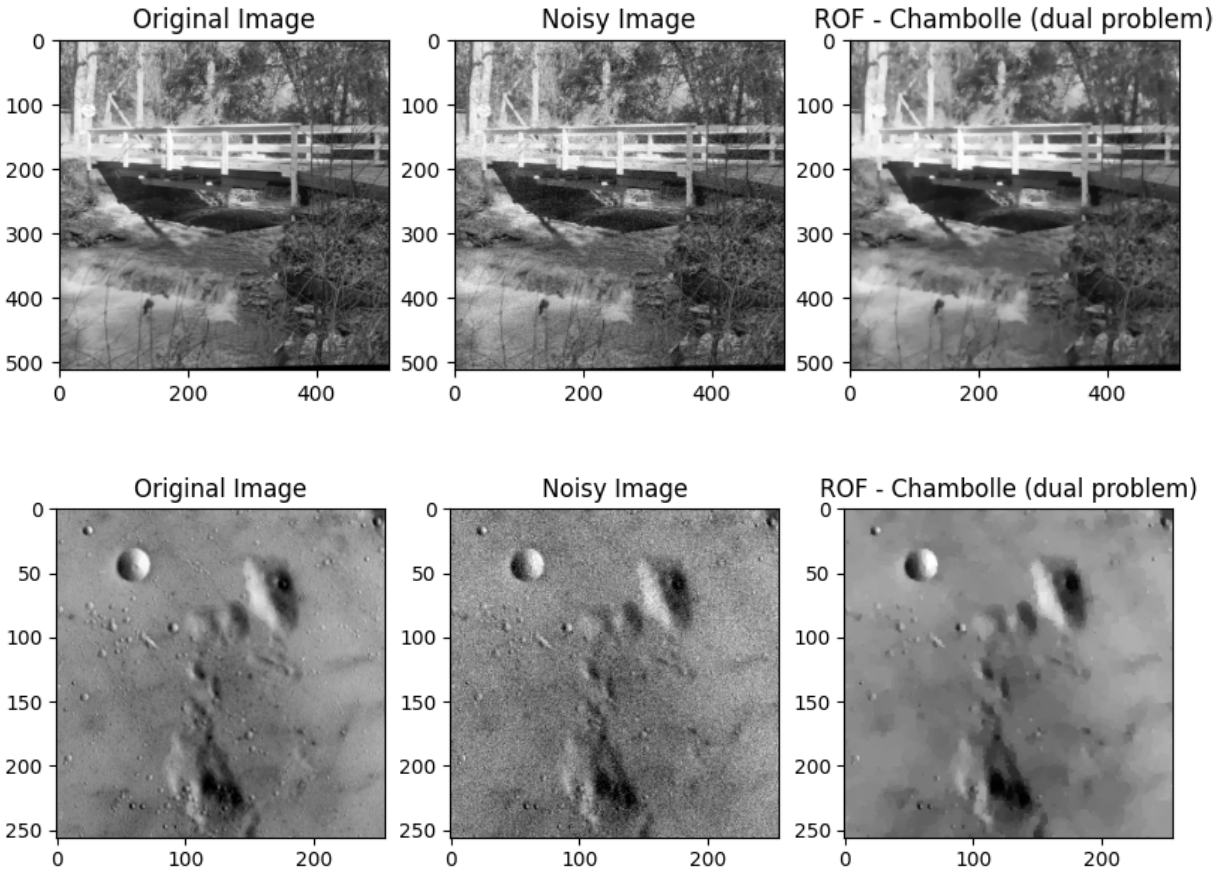
We observe that the algorithm does a good job at removing the noise and reconstructing original image, and the people in the picture frames seem to visible. However, some of the details on the book cover seems to be lost.

By varying lambda, we can examine the behavior of the algorithm as lambda changes:



It is evident that when lambda is too small, e.g. at 1, the smoothed image is still too noisy. However if lambda is too large, the algorithm smooths too much and we lose some of the detail as can be seen in the blur evident in the clock face and on the pictures in the frames.

We can also try to see how the algorithm behaves with different input images. Below, we again use the default parameters:

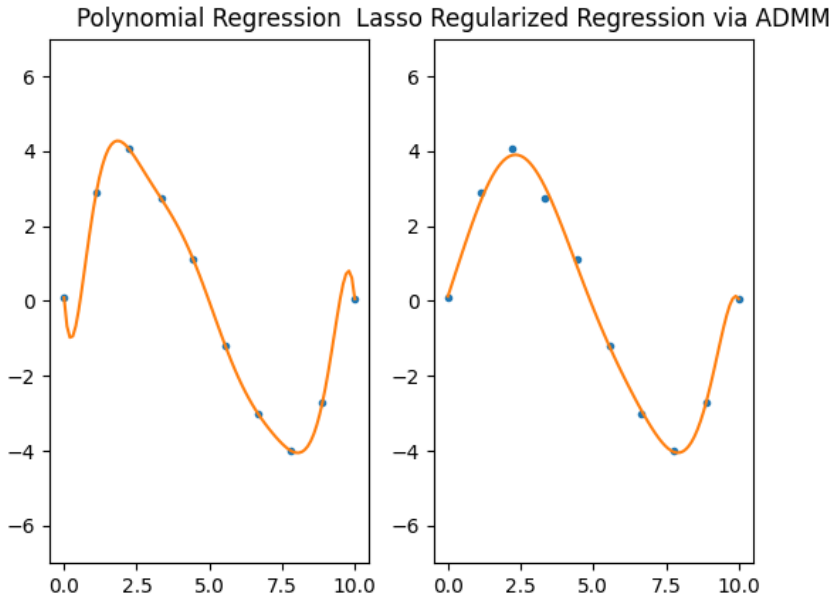


We observe that the algorithm does a fairly good job at removing noise in both images, however it also smooths “too much” and removes details of the original image. This is evident in detail being lost in the moon’s surface in the lower image, and the blurring of the water under the bridge in the upper image.

(4) (b)

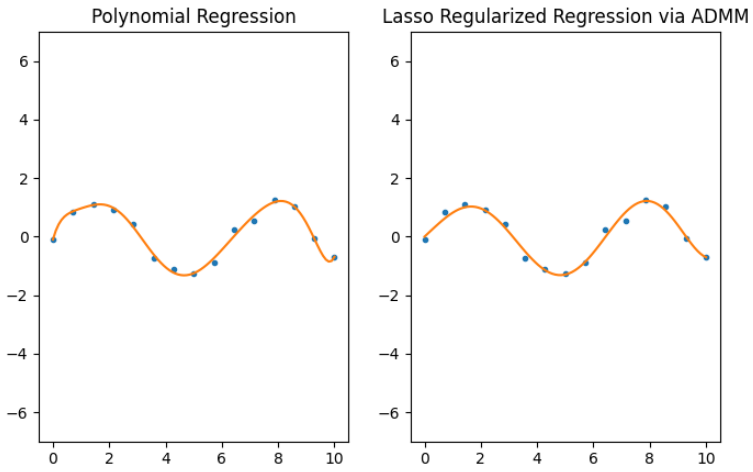
The implementation is in hw3_admm_polynomial.py and implements x, y, z updates as described in a). It should be noted that for the x update, we use the “scipy.linalg.solve” which is a scipy function to solve a system of linear equations (Mx = b).

Using the default algorithm parameters (lambda = 0.12, rho = 0.1), we get the following plot:



So we can observe that LASSO regression via ADMM appears to do a better job of approximating the true polynomial (which was a cubic) with extra noise. Whereas, simple polynomial regression seems to “overfit” to the data and the noice and tries to go through every data point, which increases the order and thus complexity of the approximating polynomial. We can also see that the 10 degree polynomial interpolation suffers from Runge’s phenomenon, where there is oscillation at the edges of the interval.

We can also change the initial data to be a sine curve with noise:

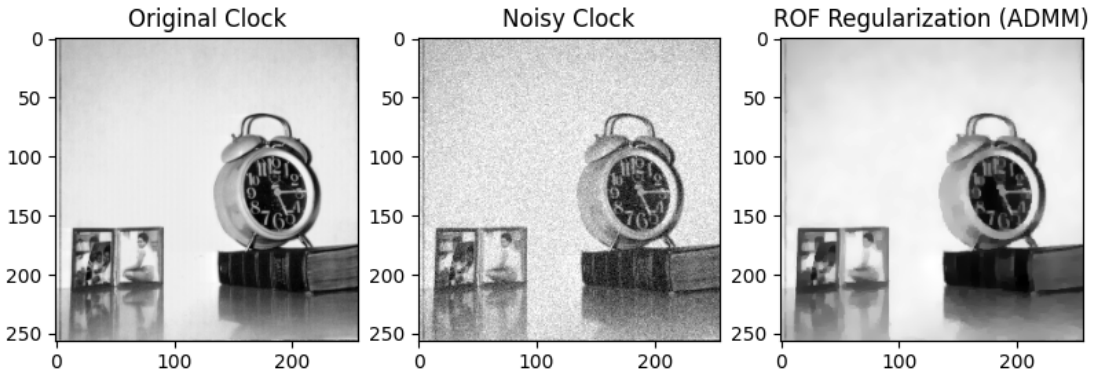


For this, lambda was set to 0.25 and the number of data points set to 15, and we can again observe how the LASSO Regularized ADMM does a better job of approximating the original sine curve than the polynomial regression. The polynomial regression still overfits to the noisy data points. However as the original data is not from a polynomial, perhaps polynomial regression was always an unsuitable choice for this task.

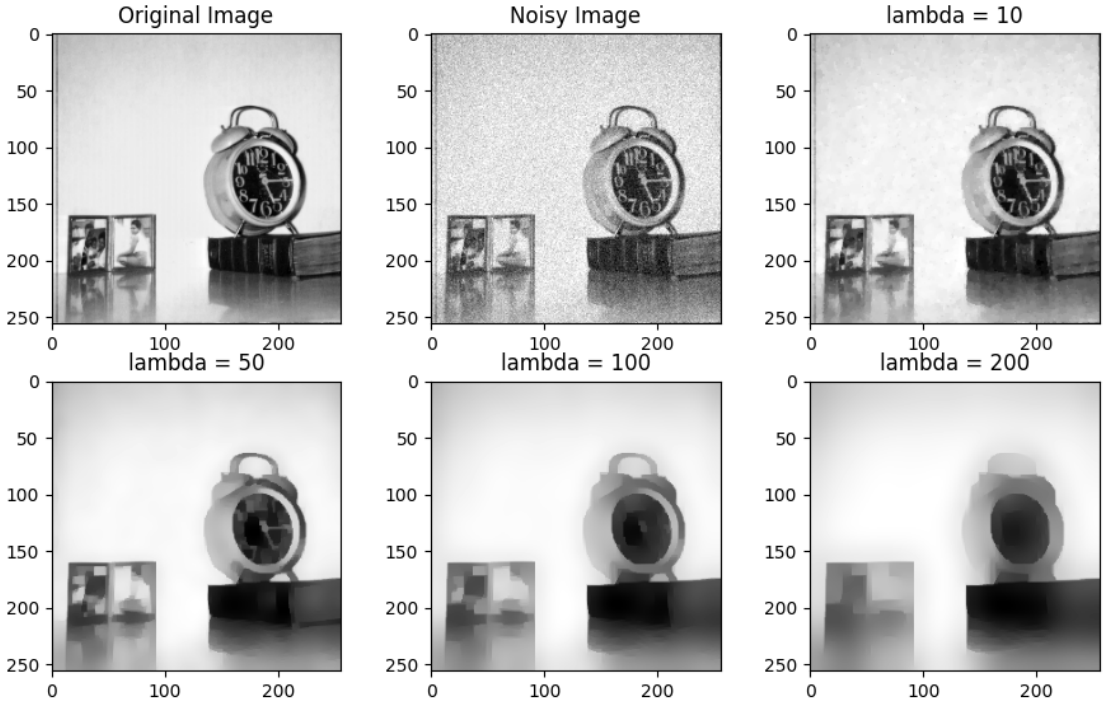
(5)

The implementation is in hw3_admm_polynomial.py and implements x, y, z updates as described in a). Throughout the implementation, sparse matrices are used via “scipy.sparse” functions to make computations more memory efficient. And in the x update, “scipy.sparse.linalg.spsolve” is used to solve a system of linear equations “Mx = b” where M is a sparse matrix.

The anisotropic ADMM that is implemented has the default parameters (lambda = 20, rho = 1). Applying this algorithm to the default clock image with noise, we get convergence in 12 iterations,



and the following plot:
We can clearly observe that the algorithm does a good job at removing noise and reconstructing the original image, albeit with some extra smoothing. To get a better understanding of the algorithm, we can examine its behavior as lambda varies:



The algorithm converges in 14, 19, 39, and 83 iterations respectively for lambda = 10, 50, 100, and 200. We observe that that smaller lambdas do a better job at keeping detail in the image such as numbers on the clock and people in the picture frames. However as lambda decreases too much, noise is also kept in the smoothed image, as is evident for lambda = 10. So a good middle ground would be our default parameter lambda = 20 which produced the regularized image at the beginning.

It is also interesting to note that for large lambdas we can observe the anisotropic behavior of the algorithm, as there is no or very little smoothing done across edges, but other aspects are smoothed too much. For example for lambda = 200, we can clearly make out the edges of the clock, book and the picture frame, but no details within them are visible.