Fitted Denoising Algorithm

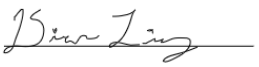
Diana Liang Biostats 615 11/20/20

Fall 2020 Biostatistics 615 Midterm Project

Due: Friday, November 20, 2020 23:59pm

Send this first page, signed, with your midterm project report (PDF or DOC file) and the `R` source code to GSI Zikai Lin's email `zikai@umich.edu`.

I hereby swear that the work turned in for this midterm project is solely mine. I did not receive any assistance, help or guidance from anyone else.

Signature: 

Denoising Algorithm

Enhancing images has been a modern problem for professional contexts such as analyzing objects in space and personal contexts such as taking the best nature picture. The denoising algorithm included with this report attempts to solve a simplified version of this modern problem. It takes a noisy square image with only one spatially contiguous object as its input and outputs a denoised square image with only two values, either part of the object or not part of the object.

The noisy square image has true values; part of the object, space R, or not part of the object, the background; to which an unknown standard deviation has been added to create the noisy values. In effect, the distribution of values in the noisy image is a mixture of two Gaussian distributions with different means but the same standard deviation, as can be deduced both logically by the set up and visually by the histogram in Figure 1. Since the algorithm is trying to separate these noisy values, there needed to be a cut off value to separate between the two designations. The best cut off value would be where the two Gaussian distributions meet with the values on either side being designated as part of R or part of the background. Seeing as many pixels in R would be wrongly designated, the final cut off point was moved slightly farther from the mean value for R since the next section would better remove wrong designations in the background.

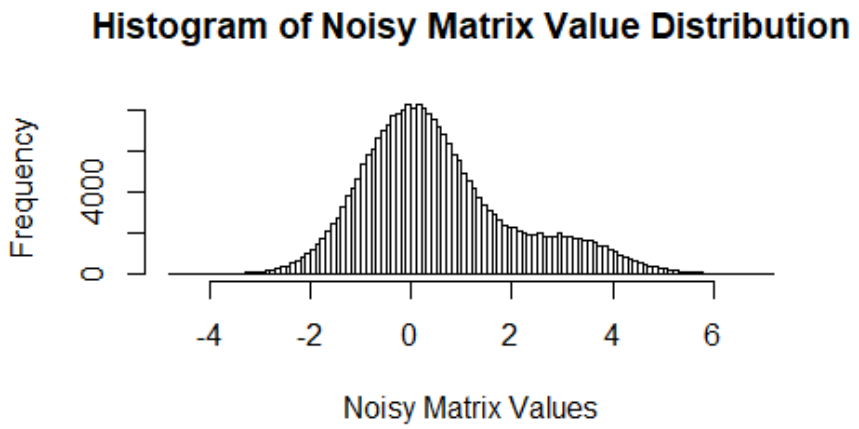


Figure 1: Distribution of Noisy Matrix Values that resembles a Mixture of two Gaussian Distributions.

The object in R had to be contiguous, but this definition leaves ambiguity allowing for objects that included background within the boundaries of R. So an iterative method was used to only include in R the pixels that were connected to one another either vertically or horizontally. Any wrongfully designated pixels in the background likely would be dismissed since the collective would have an area of less than 5% of the entire picture. The final denoised image is then outputted.

Computational and Complexity Analysis

A 500 pixel by 500 pixel noisy image translated into a matrix was denoised by the algorithm and compared to the original true image to analyze the algorithm. The inputted noisy image, outputted test

image, and original true image are shown in Figure 2. 0.72% of the pixels were wrongly designated leading to a mean absolute difference of 0.0215. The overall effectiveness of the algorithm seems decent, as demonstrated by the error measurements as well as by visually comparing the images, but could probably be improved with more restrictions on the original true image such as a lack of background within the boundaries of space R.

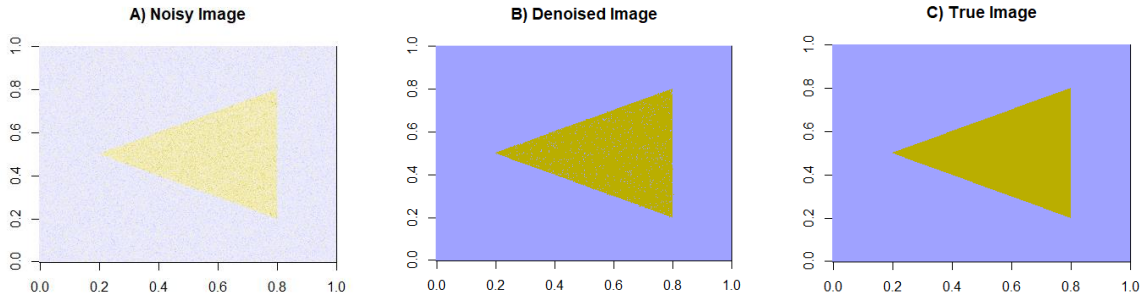


Figure 2: Visualizations of A) the Noisy Image Matrix Input, B) the Denoised Image Matrix Output, and C) the True Image Matrix that noise was added to. The yellow represents the object and the purple represents the background with other colors representing an value between those designations.

Complexity-wise, the algorithm took 375.86 seconds and an upper boundary of 183.1 MB to run. Most of the time and memory was likely taken up by the iterative method at the end to clean the background and map out R. Since almost every pixel first designated by the cut off method as part of R needed to be individually checked for fear of background being within the boundaries of R, the time complexity will increase multiplicatively with decreases in the difference between the true value for being part of R and the true value for being part of the background as well as increases in the standard deviation from the noise error. The memory used mainly came from storing the pixel locations of those designated as in R by the first cut off method and then storing the indices of the locations that were designated by the second iterative method. Overall, the memory usage does not need to be improve upon, but the time complexity could be.

The current algorithm will provide an incredibly fitted estimation of the true image, but time usage was already expensive for this case and will increase with closer true values and greater noise deviations. Decreasing time usage, though, might decrease the flexibility of the algorithm.

How to Run the Code

As detailed in the included code, shown in Figure 3, the square noisy matrix representing the image should be inputted into the denoise_matrix function, and a denoised matrix will be outputted. The function uses the Matrix package, so the Matrix package will need to be installed beforehand. Other necessary functions used by this function will also need to be defined before implementation.

```
## How to Use Algorithm ------------------------------------------------------
# Step 1: Define all the functions in Part B
# (Mid Step: Make sure the Matrix package is installed)
# Step 2: Define denoise_matrix function in Part A
# Step 3: Load or create square noisy matrix
# Step 4: denoise_matrix(your_noisy_matrix) will output a denoised matrix

load("matrix_data.RData")
output <- denoise_matrix(your_matrix_name)
```

Figure 3: Code Description and Example Code for Implementing Algorithm