IMAGE DEBLURRING - COMPUTATION OF CONFIDENCE INTERVALS

VIKTORIA TAROUDAKI
tarvic@math.umd.edu
AMSC Program, University of Maryland, College Park

Advisor: Prof.Dianne P. O'Leary
oleary@cs.umd.edu
Professor, Computer Science Department
and Institute for Advanced Computer Studies
University of Maryland

Spring Semester 2011

Abstract

Cameras often record blurred images of the original object. Restoration of the image is not a trivial procedure, and it can be very expensive for large images. In this project we are trying to efficiently compute confidence intervals for the digital values that represent the image and visualize them so that the viewer can distinguish truth from uncertainty.

1 Introduction

An image is divided into pixels that have values denoting the color of that pixel. A grayscale image, which we will use for simplicity, has one value for each pixel, an integer in the interval $[0, 255]$. 0 is the black color, and 255 is the white color.

Blurring occurs when a pixel value is affected by its neighbors. In this project, we will assume that this is caused by a linear transformation arising from the camera.

We will use the following notation:

Symbol	**Size**	**Explanation**
K	$m \times n$	Matrix defined through the Point Spread function (PSF) in the case of a linear problem
X		Original Clear Image
x	$n \times 1$	Vector containing the values corresponding to the pixels of the image X
B		The blurred image we measure
b	$m \times 1$	Vector which contains the values of the pixels of the blurred image B
e	$m \times 1$	Noise Vector

With the above notation, the model of the blurred image is described by the equation $b = Kx + e$, and we know that $0 \leq x_j \leq 255$, $i = 1, \ldots, n$.

In general, the goal is, given the vector b and the matrix K and also given a distribution for the noise such that the mean value is 0 and the variance is a nonsingular matrix S^2, to compute confidence intervals (i.e. intervals in which the true pixel values of the object fall with a certain statistical confidence) for the quantities $\varphi_k^* = w_k^T x$ for $k = 1, 2, \ldots, p$, where w_k are given vectors. If w_k is a column from the identity matrix, then we obtain a confidence interval for a single pixel.

The confidence intervals we will use are the simultaneous confidence intervals: we determine l_k and u_k, $k = 1, 2, \ldots p$, so that

$$Pr\{l_k \leq \varphi_k^* \leq u_k, k = 1, 2, \ldots, p\} \geq \alpha$$

where Pr denotes probability and $\alpha \in (0, 1)$ is a confidence level.

The following theorem is a slight generalization of one by O'Leary and Rust [4].

Theorem 1 *Suppose that the noise is normally distributed. Then, given α in $(0, 1)$, there is a $100\alpha\%$ probability that the true value of $w_k^T x$ is contained in the interval $[l_k, u_k]$ where*

$$l_k = \min\{w_k^T x : \|Kx - b\|_S \leq \mu, 0 \leq x_j \leq 255, j = 1, \cdots, n\}$$

and

$$u_k = \max\{w_k^T x : \|Kx - b\|_S \leq \mu, 0 \leq x_j \leq 255, j = 1, \cdots, n\},$$

$rank(K) = q$, $\int_0^{\gamma^2} \chi_q^2(\rho)d\rho = \alpha$, $r_0 = \min_{0 \leq x_j \leq 255} \|Kx - b\|_S^2$,$\mu^2 = r_0 + \gamma^2$ *and* χ_q^2 *is the probability density function for the chi-squared distribution with* q *degrees of freedom.*

2 Approach

2.1 Point-Spread Function and Blurring Matrix K

In general, the matrix K can be experimentally measured using point spread functions for each pixel of the original image. An easy way to do this is by constructing an artificial image which contains only one white pixel (of value 255) as the target pixel, say the (i, j) pixel of the image X (or the $(j-1) \cdot m + i$ element of the vector x) and black anywhere else (value 0).

We consider this as a clear image and we blur it the same way as we would blur the original image (or the vector corresponding to the original). Then, we measure the resulting blurred image B, the point spread function. The corresponding vector b is the $(j-1) \cdot m + i$ column of the matrix K.

If we know that the blur is spatially invariant, then measuring only one column of the blurring matrix K is enough to determine the whole matrix, as the rest of the columns of K are simply going to be some displacement of that one column.

For the purposes of this project, the blurring matrix K was constructed using spatially invariant blur and Gaussian Point Spread Functions. Usually these Point Spread Functions are of much smaller size than the original image. Let p be the size of the PSF. Then, for $k, l = 1 \ldots p$, define

$$PSF(k, l) = exp\Big(-\frac{1}{2}\frac{(k - c_1)^2}{s_1^2} - \frac{1}{2}\frac{(l - c_2)^2}{s_2^2}\Big)$$

where c_1 and c_2 are the coordinates of the center of the Point Spread Function which for a Point Spread Function which corresponds to the pixel (i, j) are equal to i and j respectively. In our experiments we set $p = 3$ and $p = 5$ and $s_1 = s_2 = 3$. We also set $m = n$, making K square.

2.2 Blurring an Image and Constructing Noise

After the blurring matrix K has been computed, we blur the image by Kx. But in order to simulate the real case of blurred images, we need to add random noise. For this, we construct a random vector, e, with elements with mean 0 and standard deviation a specified number sdv. The variance matrix in this case is the identity matrix multiplied by the number sdv^2, so it apparently symmetric and invertible. The noisy blurred image thus corresponds to the sum $b = Kx + e$. To simulate truth even better, the noise differs in every run.

2.3 Computing μ^2

By Theorem 1, we need to compute μ^2 in order to define the ends of the confidence intervals. The rank of the blurring matrix K ($q = rank(K)$) should equal m. This can be easily verified using the Singular Value Decomposition (SVD) of K. We can find γ^2 from $\int_0^{\gamma^2} \chi_q^2(\rho)d\rho = \alpha$, with α being the desired probability that defines the confidence intervals. In the examples that follow, we use $\alpha = 0.95$ which is a common confidence level. In addition, we can compute the minimum of a norm: $r_0 = min_{0\leq x_j\leq 255}\|Kx-b\|_S^2$ and finally get $\mu^2 = r_0 + \gamma^2$.

The γ^2 is computed in MatLab using the command *chi2inv(1-a,q)*. For the minimization of the norm, we use the *lsqlin* command of MatLab which performs linear least squares estimations with the constraints or the *quadprog* which uses quadratic programming with the same constraints. To do this, the first thing we need to do is to tranform the S-norm to the 2-norm that MatLab can handle. Thus,

$$\|Kx-b\|_S^2 = (Kx-b)^T S^{-2}(Kx-b) = (S^{-1}(Kx-b)^T)(S^{-1}(Kx-b)) = \|S^{-1}Kx-S^{-1}b\|_2^2$$

These matrices and vectors manipulated by *lsqlin*. For *quadprog*, we need to modify the matrices and vectors to get the appropriate H and f that it takes as input.

2.4 Lower and Upper Bounds of Confidence Intervals

For the simultaneous confidence intervals, the ends of the confidence intervals, or else the bounds are given by the following equations:

$$l_k = min\{w_k^T x : \|Kx - b\|_S \leq \mu, 0 \leq x_j \leq 255, j = 1, \cdots, n\}$$

$$u_k = max\{w_k^T x : \|Kx - b\|_S \leq \mu, 0 \leq x_j \leq 255, j = 1, \cdots, n\}.$$

O'Leary and Rust ([4]) have proven the following theorem:

Theorem 2 *The values l_k and u_k are defined by the two extreme roots of $L(\varphi) - \mu^2 = 0$ where $L(\varphi) = min_x\{\|Kx - b\|_S^2 : 0 \leq x_j \leq 255, j = 1, \cdots, n, w_k^T x = \varphi\}$*

The proof makes use of the convexity of the function. Also, the computation of the parameter μ assures us that the function $L(\varphi) - \mu^2$ can be negative. These two facts together and the continuity of the function gives us that there are two roots.

The function $L(\varphi)$ can be evaluated in MatLab by the *lsqlin* or the *quadprog* functions and transforming the data each time as described in the previous section. The main idea here was to compute a function $L(\varphi)$ that finds the minimum of the norm for all x with the constraints of the image, i.e., $0 \leq x_i \leq 255$ for all i. The output of this, the minimum norm, is used in computing $L(\varphi) - \mu^2$ and we find the zeros. This would give us values for φ where $\varphi = w^T x$. If, as in this project, w are the columns of the identity matrix, then for each one of these vectors w, we get the value of one pixel as φ. The lower and upper bounds of the confidence intervals will then give us the lower and the upper limits of the value of each one of the pixels with the probability that defines the confidence intervals and which we used to compute the μ.

2.5 Sub-images and Sub-matrices

The blurring matrix K is defined by the point spread function. It is square, block diagonal and symmetric and has a size which is the square of the size of the original image. This matrix can be very large depending on the size of the image. For example an image of size 128×128 will have a blurring matrix of size 16384×16384. This matrix is used in the iterations and so the number of operations in the algorithm is extremely high

To reduce the expense, we partition the blurring matrix into sub-matrices and partition the whole problem into p smaller problems which can be solved individually. These sub-problems are much easier to solve in the sense that they require fewer operations and as a result, less time. These sub-problems are independent from each other and so they can be solved simultaneously using parallel computing (for more details, see the implementation section).

Mathematically, let's consider again the problem: $b = Kx$ where K is the $n \times n$ PSF matrix, x is the vector corresponding to the original image and b the vector corresponding to the blurred image. If we want to make a sub-problem of size $r \times c$ (r rows and c columns of the sub-image defining the sub-problem), we proceed as follows. Using a matrix E with n rows and rc columns, with columns that are unit vectors

corresponding to the pixels in the sub-image, and a matrix $\bar{E}$ that corresponds to the other $n - rc$ unit vectors of a matrix such that $I = [E\bar{E}]\Big(\begin{bmatrix} E^T \\ \bar{E}^T \end{bmatrix}\Big)$ we have that:

$$E^T b = E^T K x = (E^T K[E\bar{E}])\Big(\begin{bmatrix} E^T \\ \bar{E}^T \end{bmatrix} x\Big) = E^T[\hat{K}_s \hat{K}_t]\begin{bmatrix} x_s \\ x_t \end{bmatrix} = [K_s K_t]\begin{bmatrix} x_s \\ x_t \end{bmatrix} = K_s x_s + K_t x_t$$

where x_s is the vector corresponding to the sub-image. Bringing the second term of the right hand side to the left we have that $b_{st} = b_s - K_t x_t = K_s x_s$. The x_t that we need to use is a first approximation of the values of the pixels of the image, $\hat{x}$, that we can compute by minimizing the S-norm: $\|b - Kx\|_S^2$ under the constraint that $0 \leq x \leq 255$. Mathematically, $K_t x_t$ includes all the pixels of the image that are not contained in the sub-image. Practically though, because the matrix K is sparse, having zeros in a lot of places, only the neighboring pixels have an impact in the result. From now on, these neighboring pixels will be called the boundary of the sub-image. The role of the boundary will also be discussed later under the validation section.

The sub-problem we need to solve is $b_{st} = K_s x_s$.

E is a matrix of unit vectors that we choose, but in this project we will use the matrix which is part of the identity matrix corresponding to the sub-problem, i.e., if we have $x_s = \binom{x_1}{x_2}$, then we will take E to be the first two columns of the identity matrix with size $n \times n$ so that the matrix K_s is going to be 2×2.

The methods presented in the introduction can now be applied to these smaller problems. We compute the confidence intervals for each one of these so that we have confidence intervals for the whole image.

To display the computed confidence intervals, we can choose a random value in each of the confidence intervals. We make many different samples. Then we can display these deblurred images in frames that change quickly and produce an effect which is called twinkle (Nagy and O'Leary [**?**]). If the confidence intervals are short, then the twinkle effect will show a somehow robust image. If the intervals are long, then we get information.

3 Implementation

The implementation of the algorithms was done in MatLab.

We divide the problem into smaller problems which can be solved more easily. If

the blur is spatially invariant, these sub-problems involve the same matrix. These sub-problems can be solved using parallel computing. Parallel computing is useful to handle bigger images in about the same time that MatLab needs for a smaller image.

The idea that was followed in this project for subimages was to check if the image was sufficiently small and, if so, deal with it as a whole. If it was classified as a big image, then it was divided into subimages that do not overlap. That means than no two subimages have common pixels. This procedure has some restrictions though. First the original image should be small or of size $2^{pow_1} \times 2^{pow_2}$. Whether an image is small or not is defined by a parameter that the user inserts. More precisely it means that the image is smaller or equal to the size of the subimage we want to use. The subimage should also be square of size $2^{pow} \times 2^{pow}$. Each subimage could be treated individually using as boundary values the values computed by the initial minimization problem, $\hat{x}$ as stated in the previous section on Subimages. At the end of the code, all the results are combined together to give results of the same type and size of the original image.

One may think that according to the above, that for the parallelization, we send each subimage into a different worker. That could be a good idea if we had a lot of subimages relative to the number of workers. But imagine an image that has one large subimage. Then this subimage would be worked by one of the workers and the others wouldn't help at all. Thus, the time would be similar to using the non-parallel code. This problem was overcome by parallelizing the minimizations inside each subimage. Thus the maximum number of workers would be used and the running time would be minimized. Doing that, we can even parallelize the code when the image is small and we do not separate it into smaller ones.

The implementation and the runs of the codes were performed on a computer with only two cores, so our excpected speed up is just a factor of two.

4 Databases

The images that have been used for the purpose of the project are grayscale images of various sizes. The maximum size of the images that can be used by the code are determined by the memory that MatLab can handle in each computer. In our case, images up to 64×64 could be used.

Constructing specific images, cropping images or resizing already existing ones, created an image database.

5 Validation

In order to validate the code, we need to run the program using data which should give us a known or expected result as an output. By data, we mean blurred images for which we know the clear image. In detail, we take some images which are considered as the clear, original images. The confidence intervals that we compute should contain the values of the pixels of the original image with the probability that we used to compute the intervals. We blur the clear images to obtain the input for our code. Noise is also added to the blurred images to simulate reality and in order to have the symmetric and positive definite matrix S that the theory requires. We then compute the confidence intervals using our code and we count how many samples fall within the intervals.

Validation was done for the appropriate relation of image size and subimage size but using for example a 64×64 image with 16×16 or 32×32 size of subimage was forbidden by the running time. (Details on the running time can be found in the Testing section.)

We run the code N times and we count how many times all of the confidence intervals include the true values of the pixels. Let this number be M. If $M/N \geq a$ or equivalently if $M \geq 100aN\%$, then the code of computing the simultaneous confidence intervals is validated.

In order to construct the blurring matrix we used zero boundary conditions. That means that we assume that the neighborhood around the clear image is all black. When we deal with sub-images, a boundary region around each subimage has an effect on the subimage, and this is why we use a first approximation of the values of the pixels by the solution of a minimization problem. The uncertainty that this introduces may ruin the method. For this reason, we should be careful and use subimages relatively big with respect to the size of the boundary. This issue appeared during the validation process: with small subimages, the results were not validated whereas for larger subimages, the confidence intervals were correctly computed under the specific probability.

The formula to compute the number of pixels in the boundary (BP) given the size of the subimage ($n \times n$) and the size of the PSF ($p \times p$) is the following

$$BP = 4\left(n + \frac{p-1}{2}\right)\frac{p-1}{2} = 2(p-1)\left(n + \frac{p-1}{2}\right)$$

6 Testing

The goal of this project was to implement a code for finding the confidence intervals for the values of the pixels of an image. But simply writing a correct, validated code is not enough. A good code should give the right results in a relatively short time without much cost in memory, in the frame of the problem. Several tests have been run and comparisons have been made.

Running time and storage, different Point Spread Functions and different types of error, i.e., different standard deviation matrices, were the first things that were tested. The reason for this was that the running time may depend on the size of the image, its format and the way that the values of the pixels are stored. The same things affect the memory. Also, for various types of blurring (Point Spread Function) the running time for the same image may be different due to the different blurring matrix and the minimization techniques.

All of the above were studied and compared using images constructed exclusively for this purpose. Figures 1-3 present some output examples and the comparison between the methods.

Figure 1: full 64×64 image

Figure 2: full 32×32 image

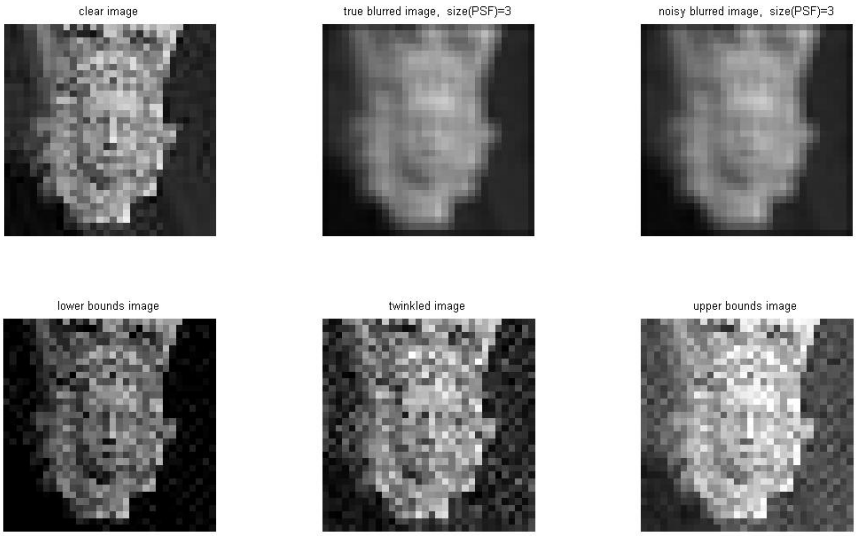

Figure 3: full 32×32 image

Finally, the running time of the codes was examined. This is done in two ways. First, we increase the size of the image to see how the time changes and second, for a

particular image size, we change the number of the subimages used. The codes that are compared are the serial code that uses the image as a whole, the parallel code that uses the image as a whole, the serial code that separates the image into subimages, and the parallel code that separates the image into subimages.

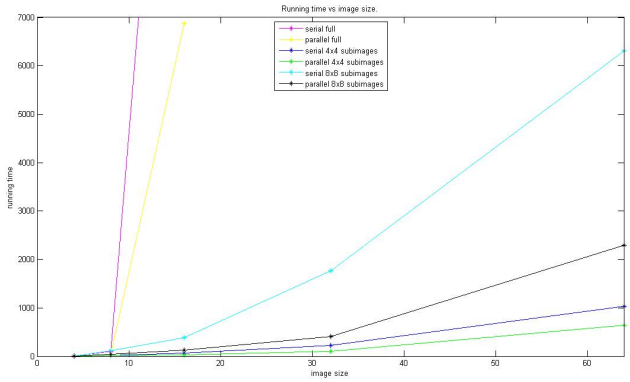

Figure 4: Running Time - Size of Image

The first plot (Figure 4) is time as a function of the size of the image. We include two sub-images of size 4×4 whenever that is validated, and 8×8. The time was the average of several runs that were made with the same parameters.

It is clear that with every method shown here, the time increases as the size of the image increases. This increase does not seem to be linear. The slowest methods seem to be those who deal with the image as a whole whereas the fastest are those with small subimages. In each pair of methods, the parallel one is faster as expected.

7 Summary

In this project 4 different codes strongly related to each other were developed. These codes compute simultaneous confidence intervals following the theory in [4] and [3]. Two of them are parallel versions of the corresponding serial codes computing the intervals from a whole image or an image separated into subimages. These codes were validated and comparison with respect to time was performed.

References

[1] Tony F. Chan and Jianhong (Jackie) Shen, *"Image Processing and Analysis"*,SIAM, Philadelphia, 2005

[2] Per Christian Hansen, James G. Nagy and Dianne P. O'Leary, *"Deblurring Images Matrices, Spectra, and Filtering"*, SIAM, Philadelphia, 2006

[3] James G. Nagy and Dianne P. O'Leary, *"Image Restoration through Subimages and Confidence Images"*, Electronic Transactions on Numerical Analysis, 13, 2002, p. 22-37

[4] Dianne P. O'Leary and Bert W. Rust, *"Confidence Intervals for inequality constrained least squares problems, with applications to ill-posed problems"*, SIAM Journal on Scientific and Statistical Computing, 7, 1986, p. 473-489

[5] Bert W. Rust and Dianne P. O'Leary, *"Confidence intervals for discrete approximations to ill-posed problems"*, The Journal of Computational and Graphical Statistics, 3, 1994, p. 67-96