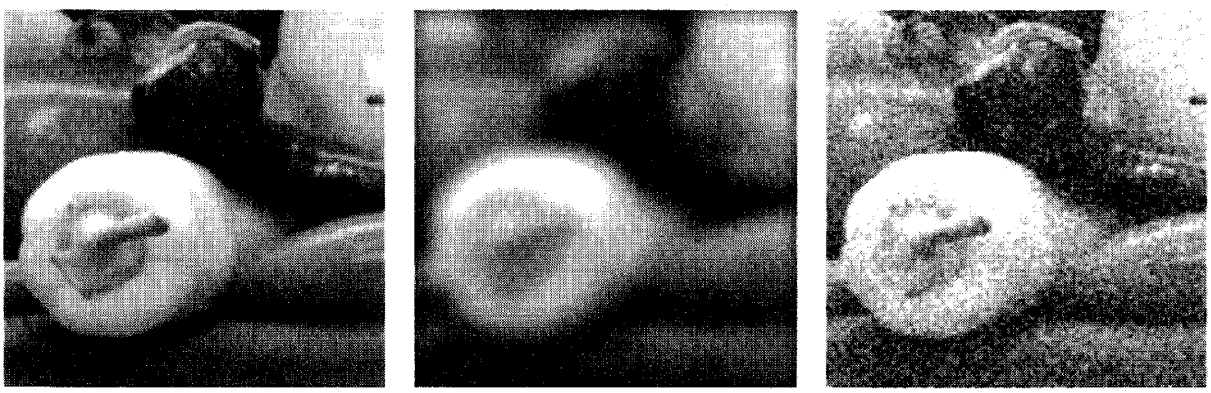
DISCOVERING THE MATHEMATICS OF IMAGE DEBLURRING

Katrina Palmer
Appalachian State University
346 Walker Hall, Boone, NC 28608
palmerk@appstate.edu

Have you ever wondered how scientists identify objects in space or how police detectives identify suspects using surveillance cameras? If so, then you've dabbled in an application image restoration. This paper includes some basic mathematics of image blurring and examples edge detection. This paper describes part of a module that was taught as part of a freshman seminar course that intended to introduce freshmen (students with little or no university background) to current research topics. For full access to the course materials, go to mathsci.appstate.edu/~kmp/EmoryCourses/FS.html.

Introduction to the Image Restoration Problem

In order to understand image reconstruction, we first need to understand what makes our images imperfect. Figure 1 demonstrates the difference between blurred images and noisy images. Blur can come from an out of focus lens or movement from the camera or the object, and can be modeled easily. Noise, randomly spaced speckles, can also appear in images, but it is more difficult to model than blur.



(a) true image (b) blurred image (c) noisy image

Suppose we start with a blurred and noisy image (see the first image in Figure 2). How can we reconstruct this image? One possibility is to use an iterative method, that is, a method that attempts to solve a problem by finding successive approximations to the solution. Figure 2 shows images at each step of an iterative method. As we can see from the images, the image quality grew sharper for a time, then at some point it became poorer at each iteration until ultimately the ``restored" image was unrecognizable. One question might be, "how do you know when to stop the iterations?"

Convolution

Convolution is a mathematical operation that models the blurring process. More precisely, in convolution, each pixel of a blurred image is a weighted sum of neighboring pixels from the original scene. The weights are defined by a *kernel*. The following

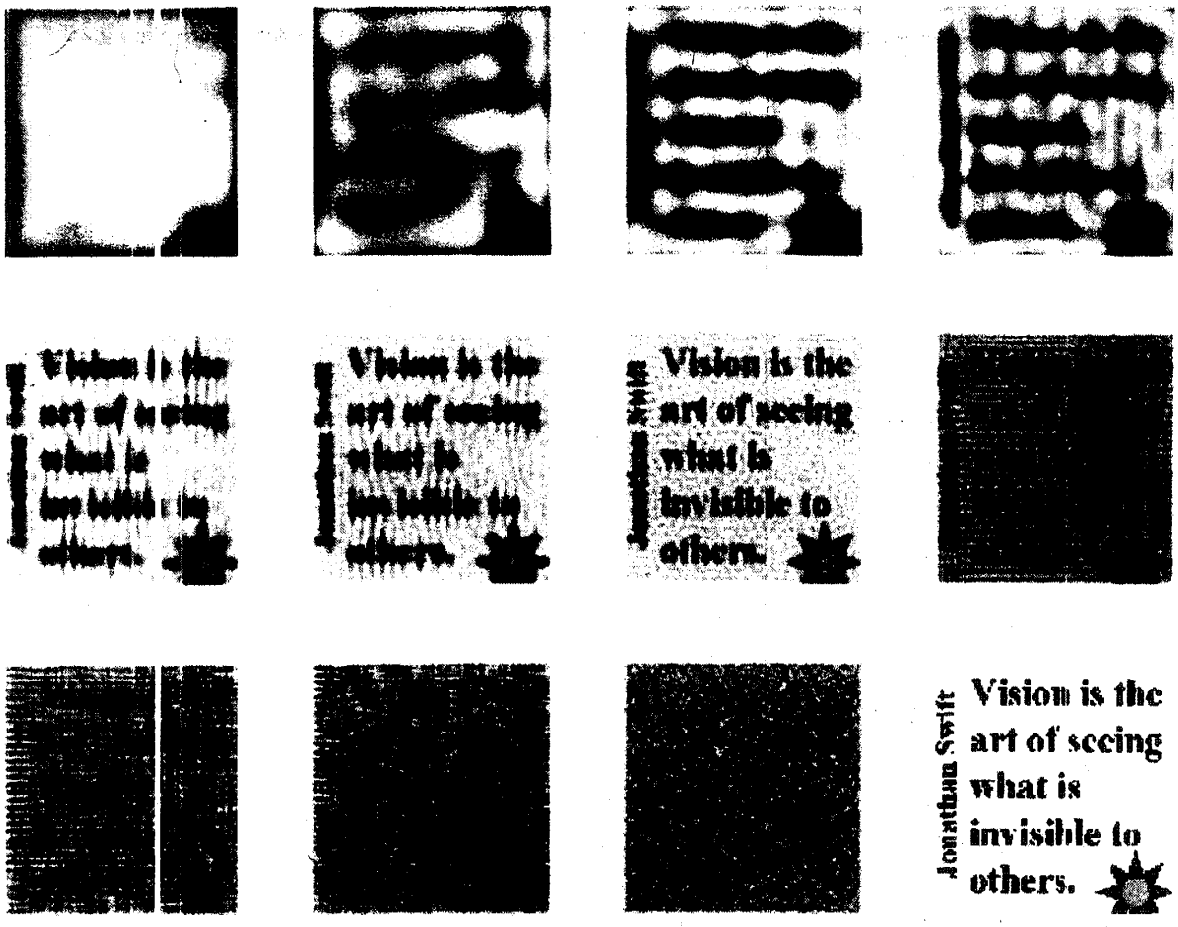


Figure 2: The first eleven images show the progression from degraded to restored to over-processed image. The last image in the series is the non-blurred, non-noisy, original scene.

example illustrates the convolution operation. Let X denote the input information, and K denote the kernel. In particular, suppose:

$$X = \begin{bmatrix} 9 & 4 & 14 & 2 \\ 15 & 1 & 10 & 7 \\ 5 & 8 & 3 & 13 \\ 16 & 12 & 6 & 11 \end{bmatrix} \text{ and } K = \begin{bmatrix} 1 & 4 & 7 \\ 6 & 9 & 2 \\ 3 & 8 & 5 \end{bmatrix}.$$

An important part of convolution is that the ``center" (or origin) of the kernel must be specified. By convention, for kernels with odd dimensions, it is usually the middle pixel value. Thus, in our example, 9 is the center of the kernel. With this notation, the convolution operation is performed as:

1. Rotate the kernel, K, 180 degrees about the center to get $\tilde{K} = \begin{bmatrix} 5 & 8 & 3 \\ 2 & 9 & 6 \\ 7 & 4 & 1 \end{bmatrix}$.

2. Place $\tilde{K}$ on top of X, so that the center of $\tilde{K}$ lies on top of a specific pixel element of X. Multiply each weight in $\tilde{K}$ by the pixel of X underneath it.
3. Sum up the individual products to get a pixel element of the output (blurred) image.

In particular, the (2,2) pixel of the output image is obtained by putting the center of $\tilde{K}$ over the (2,2) entry in X as follows:

$$\begin{bmatrix} 9^{5} & 4^{8} & 14^{3} & 2 \\ 15^{2} & 1^{9} & 10^{6} & 7 \\ 5^{7} & 8^{4} & 3^{1} & 13 \\ 16 & 12 & 6 & 11 \end{bmatrix}$$

Then, multiplying and adding, the (2,2) pixel of the output image is:
$9*5 + 4*8 + 14*3 + 15 * 2 + 1*9 + 10*6 + 5*7 + 8*4 + 3*1 = 288.$
In order to calculate the (1,1) pixel of the output image, assumptions are made about the boundary conditions. For example, with a black background (*zero boundary condition*) the image is written as $\begin{bmatrix} 0 & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & 0 \end{bmatrix}$. Now center the kernel around the (1,1) pixel, and calculate the output, 166. To learn more about other types of boundary conditions such as reflexive or periodic, see [7].

Convolution as Matrix-vector Multiplication

Convolution may be computed more efficiently as a matrix vector multiplication, $A\vec{x} = \vec{b}$, where A represents the blurring process, $\vec{x}$ is a column vector that represents the image (stack the rows of X), and $\vec{b}$ is a column vector the represents the blurred image. The goal here is to calculate the size and structure of A that would make the multiplication $A\vec{x}$ equivalent to the convolution operation of X with a given kernel. Suppose image X and kernel K are defined as

$$X = \begin{bmatrix} 9 & 4 & 14 & 2 \\ 15 & 1 & 10 & 7 \\ 5 & 8 & 3 & 13 \\ 16 & 12 & 6 & 11 \end{bmatrix} \text{ and } K = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Stacking the rows of X produces a 16 by 1 column vector. Therefore, the blurring array A needs to be a 16 by 16 array in order to have legal multiplication. To find the entries of A, recall the convolution model. The first entry of vector $\vec{b}$, corresponding to $\vec{b}_{11}$ in the blurred image, results from the convolution of K and X with the center of K lying on top of $\vec{x}_{11}$. Assuming zero boundary conditions, the arithmetic is
$0*1 + 0*2 + 0*1 + 0*2 + 9*4 + 4*2 + 0*1 + 15*2 + 1*1 = \vec{b}_{11}$
Moreover, this first entry of $\vec{b}$, from a matrix-times-vector standpoint, is the result of the first row of A times the vector $\vec{x}$. Putting these two facts together the first row of A is comprised of zeros and the appropriate elements of the kernel, in the appropriate order. That is, the first row of A looks like [4 2 0 0 2 1 0 0 0 0 0 0 0 0 0 0].
Likewise, the convolution arithmetic to produce $\vec{b}_{12}$ is
$0*1 + 0*2 + 0*1 + 9*2 + 4*4 + 14*2 + 15*1 + 1*2 + 10*1 = \vec{b}_{12}$

Thus, the second row of A is [2 4 2 0 1 2 1 0 0 0 0 0 0 0 0 0].
Using the same reasoning, A, the matrix that models the blur, is

$$\begin{bmatrix}
4 & 2 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 4 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 4 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 4 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 4 & 2 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 1 & 0 & 2 & 4 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 1 & 0 & 2 & 4 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 2 & 0 & 0 & 2 & 4 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 4 & 2 & 0 & 0 & 2 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 4 & 2 & 0 & 1 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 4 & 2 & 0 & 1 & 2 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 2 & 4 & 0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 4 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 4 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 2 & 4 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 2 & 4
\end{bmatrix}$$

Edge Detection Experiments

The problem of edge detection is to determine locations in an image where there is a sudden variation in the gray level. These sudden changes can sometimes be detected by convolving the image with certain kernels. For example, the MATLAB script below produces the images in Figure 3. Notice that K1 detects edges that go from white to black from left to right. What edges does *K2* = [0 -1; 2 0 -2; 1 0 -1] detect?

```
I = imread('logo.tif'); I=double(I);
K1 = [-1 0 1; -2 0 2; -1 0 1];
C1 = conv2(I, K1, 'same');
subplot(1,2,1), imshow(I,[]), title('Original Image')
subplot(1,2,2), imshow(O1,[]), title('Edges detected by K1')
```

Deblurring

For our work with computer-based restorations, we relied heavily on two things: one, an image restoration package called RESTORETOOLS [2], [6]; two, careful presentation and discussion of the concept of iterative methods of image restoration. In an iterative method for solving $A\vec{x} = \vec{b}$, an initial guess at the true solution is made, often $\vec{x}_0 = \vec{b}$, then updates $\vec{x}_k$ $k=1,2,\ldots$ are repeatedly computed until achieving a result that is sufficiently "good." One way to check the quality of an intermediate solution $\vec{x}_k$ is to compute the size of $A\vec{x}_k - \vec{b}$ at each step of the iterative routine. As long as this quantity, called the *residual*, gets smaller at each iteration, the updating process continues. Once it

begins to grow, the process is halted, and the previous solution is considered the best. Indeed, in theory, with the exact solution $\vec{x}$, $A\vec{x} - \vec{b}$ would be the zero vector. The

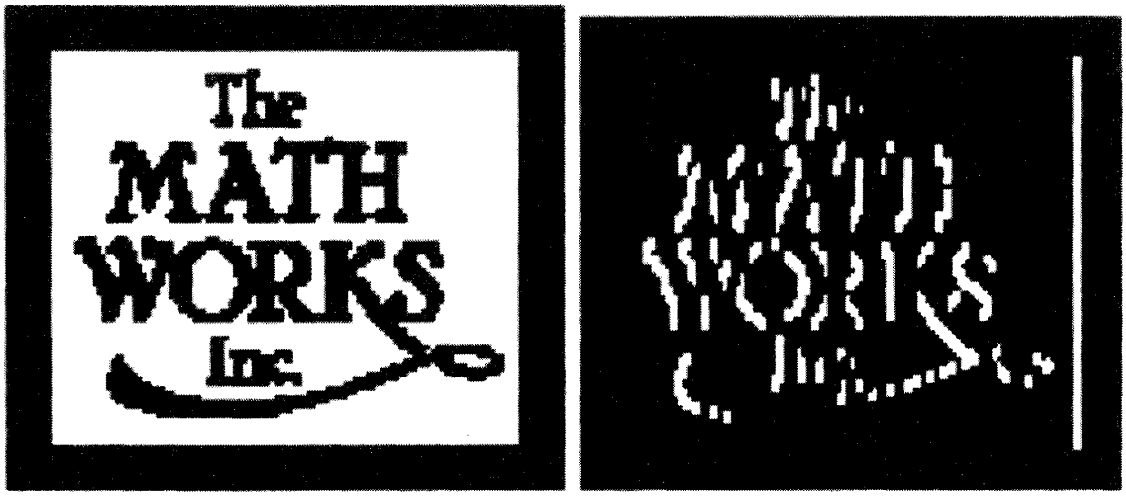


Figure 3: Shows the original image and the original convolved with kernel, K1.

problem arises because there is random noise in real data. This is why the iterations in Figure 2 get worse after some point. Many other methods exist for addressing the image restoration problem, but no method is ideal, and no method is unconditionally better than the others. Experience may provide intuition as to the ``best'' restoration method for any given image, but even then, image restoration is not an exact science. The noise in an image is truly random, a condition which our greatest computers are unable to duplicate. Nevertheless, given these limitations, minimization of the residual is at least an intuitively good place to start. For additional background reading and information, see [1], [3], [4], [5] and [8].

References:

[1] M. Bertero and P. Boccacci. *Introduction to Inverse Problems in Imaging.* IOP Publishing Ltd., London, 1998.

[2] J. Nagy, K. Palmer, and L. Perrone. *Iterative Methods in Image Restoration: An Object-Oriented Approach.* Num. Algor., v. 36, pp. 73-93, 2003.

[3] J. Nagy. *Applications of Teoplitz Systems.* SIAM News, 1995.

[4] J. Nagy. *Iterative Techniques for the Solution of Toeplitz Systems.* SIAM News, 1995.

[5] D. O'Leary and J. Nagy. *Image Deblurring: I Can See Clearly Now.* Computing in Science & Engineering, IEEE CS and AIP pub. D. O'Leary ed., pp. 2-4, 2003.

[6] RESTORETOOLS: An Object Oriented MATLAB Package for Image Restoration, 2002 http://www.mathcs.emory.edu/~nagy/RestoreTools .

[7] Efford, Nick. Digital Image Processing: A practical introduction using Java, Pearson Education Limited, pp. 137-140, 2000

[8] Using MATLAB, Version 6. The Math Works, Inc., 2000.