# AMATH482 HW03

Joshua Borgman

University of Washington

Josh Borgman

Thursday, February 12, 2015

*See AMATH 482 Notes J Nathan Kutz p. 323

## Contents

*See AMATH 482 Notes J Nathan Kutz p. 323

## 1. Introduction and Overview

The influence of imaging science in a variety of technical fields from technology to medicine has grown in bounds. The implementation of imaging science to biomedical instrumentation and graphic generation is a small glimpse of this impact. These particulars are the results of the more profound underpinning mathematical framework with which more general data analysis can be performed. In this report with will consider a few basic concepts of image analysis and manipulation, mainly considered with the correction of corrupted or excessively noisy data. In a greater generality image science can be used with various purposes, including image contrast enhancement, image denoising, image deblurring, inpainting (filling in missing data over certain portions of an image or data set), and segmentation (edge detection). The main mathematical concepts used to these ends are Fourier analysis, Wavelets, stochastic modeling, which utilizes statistical nature of a data set, partial differential equations and diffusion.

## 2. Theoretical Background

### Linear Filtering for Image Denoising (Gaussian Filters)

Similar to the denoising of time-frequency analysis signals, a fair number of applications of image processing deal with the elimination of image noise in the form of grainy noise. The main goal, as always, being to produce a maximally denoised representation of the true signal without distorting the image. Noise in terms of an image is present has high frequency components of the image's Fourier spectrum, which would otherwise tend to be very sparse in high frequency components. The goal of filtering an image is thus to remove unwanted noise fluctuations or graininess present as these high frequency components.

A two dimensional image (black and white) can be subjected to a simple Gaussian filter of the generic form

$$F(k_x, k_y) = e^{\left(-\sigma_x(k_x-a)^2-\sigma_y\left(k_y-b\right)^2\right)}$$

For which $\sigma_x$ and $\sigma_y$ are the filter widths in their respective directions, and centered at frequency values *a* and *b* in either direction.

Over filtering of the image can cause loss of significant amounts of information concerning the figure. Narrow filters often result in the loss of significant amounts of image information in addition to the noise. Strong filters with large widths cause smooth and blurry images, causing the loss of fine scale features. Moderate levels of filtering produce more reasonable results with reduction of the noise and conservation of most valuable information.

### Shannon Filters

Similar to the filtering done with Gaussian filters, filtering can be done with a variety of other filters. One such filter is the Shannon filter; a simple step function with a value of unity transmitting within a banded region and zero outside of it. Similar to filtering done with the Gaussian filter, strong filtering produces a blurred image while moderate filtering yields an enhanced image quality. It is worth considering the effect that the Fourier transform has on a step function; producing a sharp sinc-like behavior.

*See AMATH 482 Notes J Nathan Kutz p. 323

### Diffusion in Image Processing

Spatial diffusion in two dimensions in its simplest form is represented by the second order differential equation

$$u_t = D\nabla^2 u$$

In which $u(x, y)$ represents a given image. $\nabla^2 = \partial_x^2 + \partial_y^2$ is a second order spatial derivative. D is a diffusion coefficient that. The equation requires some sort of boundary conditions that must be imposed, most often a periodic boundary equation that allows for a solution via the Fourier transform

$$\widehat{u_t} = -D\left(k_x^2 + k_y^2\right)\hat{u} \rightarrow \hat{u} = \hat{u}_0 e^{-D\left(k_x^2+k_y^2\right)t}$$

This solution highlights the fact that the wavenumbers (spatial frequencies) decay according to a Gaussian function, and thus linear filtering with a Gaussian is equivalent to linear diffusion of the image for periodic boundary conditions.

While this solution establishes a relation between the two, diffusion permits a more general workspace through which to consider image cleanup by modifying

$$u_t = \nabla \cdot (D(x, y)\nabla u)$$

In which *D(x,y)* is a spatial diffusion coefficient. This allows for the flexible localization of the diffusion over targeted trouble spots, while neglecting the rest of the image.

## 3. Algorithm Implementation and Development

### Linear Filtering:

Linear filtering with any of the various filters is implemented in a way very similar to the filtering of a time-variant signal as we have previously seen. The main uniqueness comes from the development of the filter.

To implement linear filtering we take in the image data, transform the data using a fast Fourier algorithm in two dimensions ($fft2(\,)$) and multiplying our data by our filter and then reforming our data through and inverse Fourier transform ($ifft2(\,)$) in two dimensions.

When designing the filter we use meshgrid to generate the two-dimensional wavenumbers in x and y directions.

Fourier shifts need not be used unless visualization of the data is needed.

### Diffusion of Image Noise

The general algorithm for diffusion of an image utilizes the discretized form of the heat equation

$$\frac{d\boldsymbol{u}}{dt} = \frac{\kappa}{\Delta x^2}\boldsymbol{Au}$$

in which A is a sparse matrix of form (13.3.64)*. This discretization follows directly from the discretization of the spatial derivative with a second-order scheme for numerical solutions, such that

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{\Delta x^2}[u(x + \Delta x, y\,) - 2u(x, y) + u(x - \Delta x, y)]$$

*See AMATH 482 Notes J Nathan Kutz p. 323

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\Delta y}[u(x, y + \Delta y) - 2u(x,y) + u(x, y - \Delta y)].$$

The discretized version of the second dimensional heat equation assumes that $\Delta x = \Delta y = \delta$ are the same. We have designed **u** by stacking slices of the data in the second dimension y, such that

$$u_{nm} = u(x_n, y_m)$$

With these modifications to the theoretical base, our procedure progresses as follows:

i. Build the sparse matrix **A**
ii. Generate the desired initial condition vector $u = u_0$
iii. Call an ODE solver from the MATLAB suite, passing the diffusion constant, spatial step, and function of the governing form define above.
iv. Plot the results as a function of time and space.

## 4. Computational Results

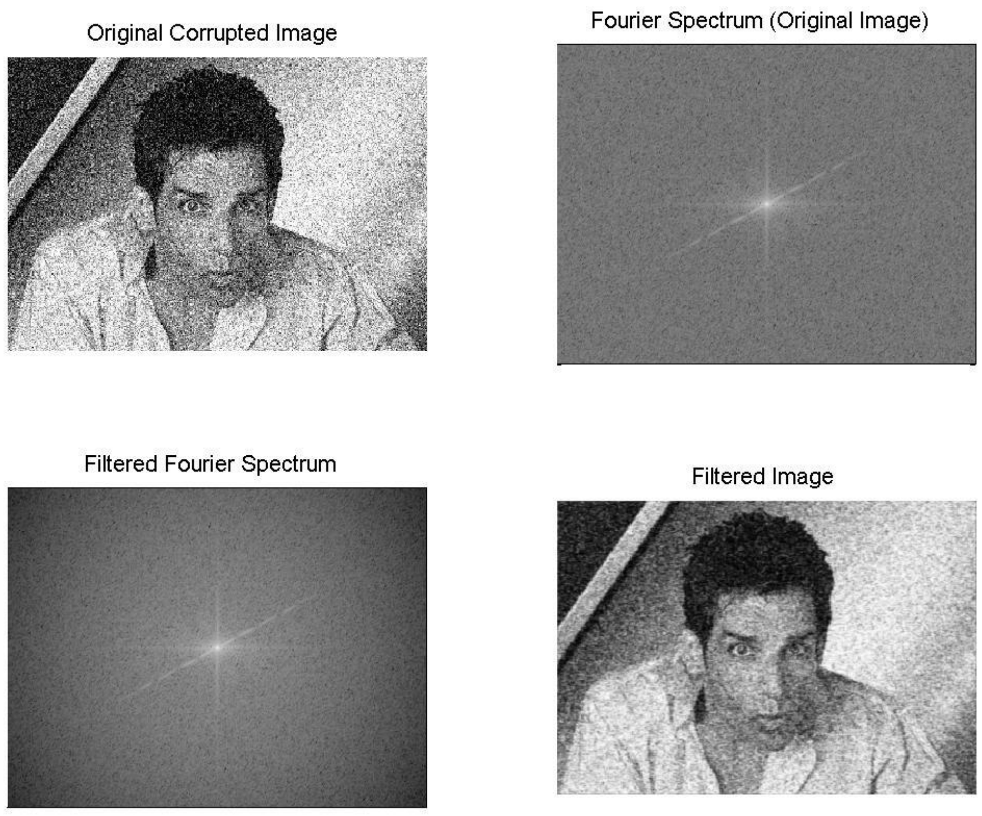

*Figure 1: This image shows the results of section one of the code in Appendix B. The original black and white image and its associated Fourier spectrum are displayed, along with an initial display of a filtering with Gaussian filters*

*See AMATH 482 Notes J Nathan Kutz p. 323

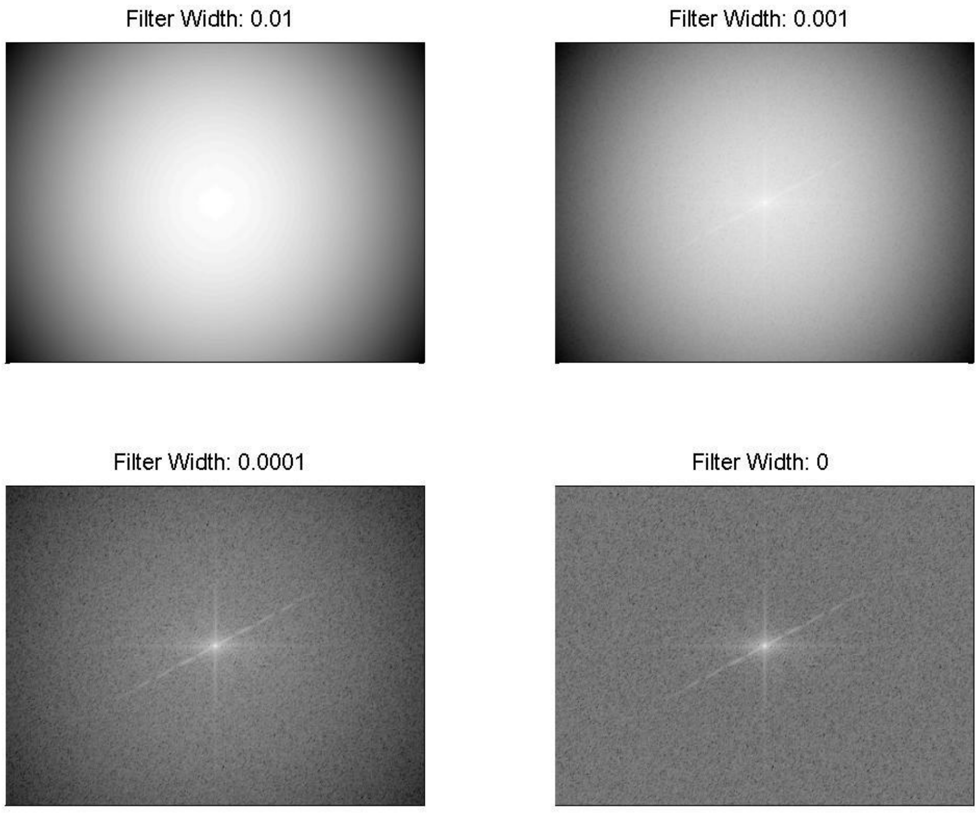

*Figure 2: This image is the result of section 2 of the code. We see the effect of increasing the filter width of a Gaussian filter in the spatial domain.*

From the above figure we are able to see the strength of the various widths. The stronger filters have more defined edges that throw out more information, as opposed to the more neutral filter sizes that permit most data and degrade only the more fringing high frequency elements.

*See AMATH 482 Notes J Nathan Kutz p. 323

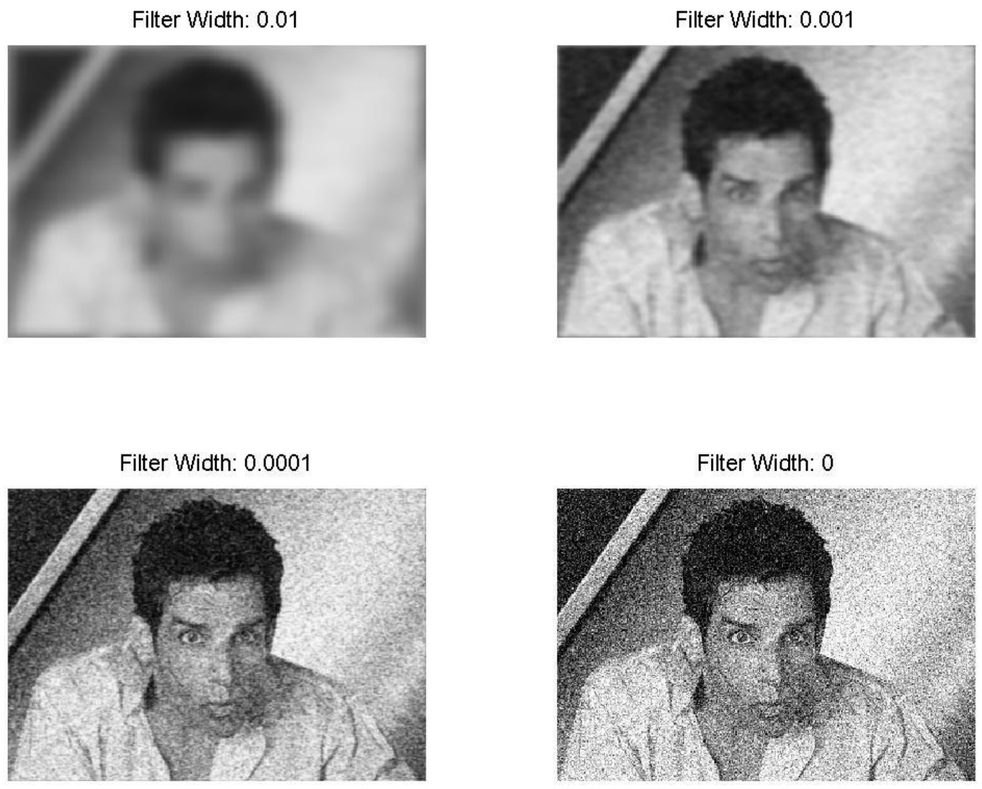

*Figure 3: These images are the results of applying the previous Gaussian filters.*

The narrow filter widths of the upper left view and the upper right view cause significant blurring of the image, representing substantial loss of image information.

The bottom left image does a nice job of quieting some of the noise that is observed in the lower right original image.

*See AMATH 482 Notes J Nathan Kutz p. 323

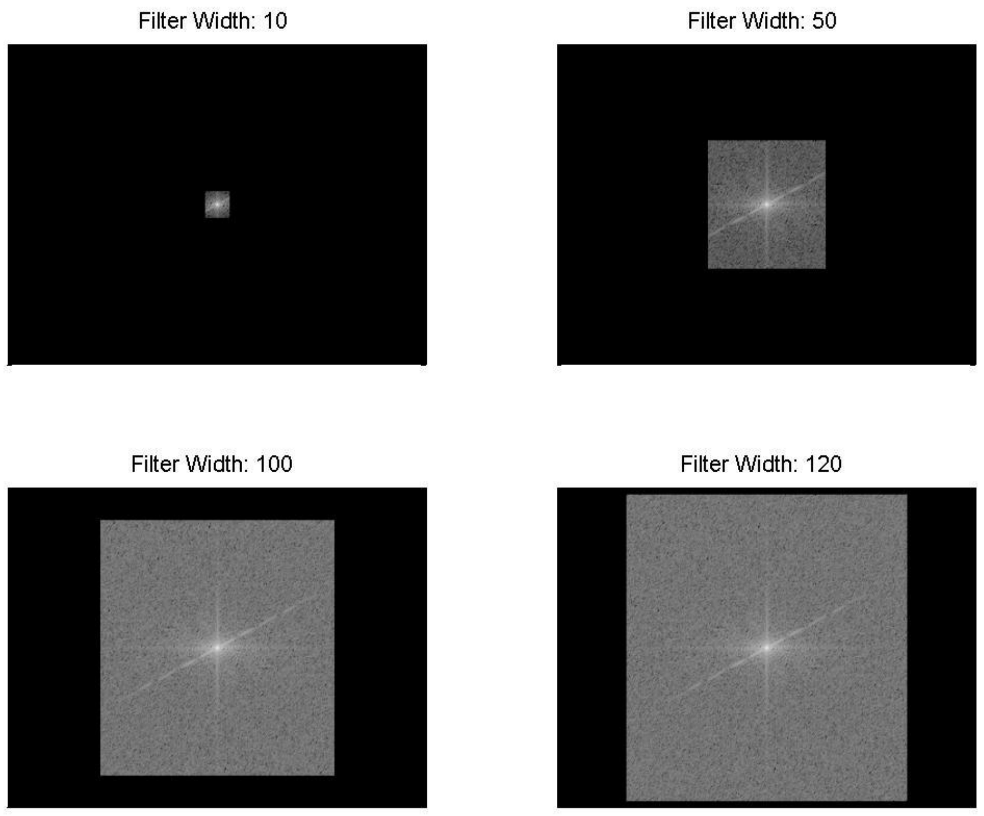

*Figure 4: As an alternative to Gaussian filters we have generated several Shannon filters of variable width. These are applied to the black and white image to yield the following image.*

The above Shannon filters are created in a similar style as the Gaussians before them and when applied yield variation in the filtering of the image.

*See AMATH 482 Notes J Nathan Kutz p. 323

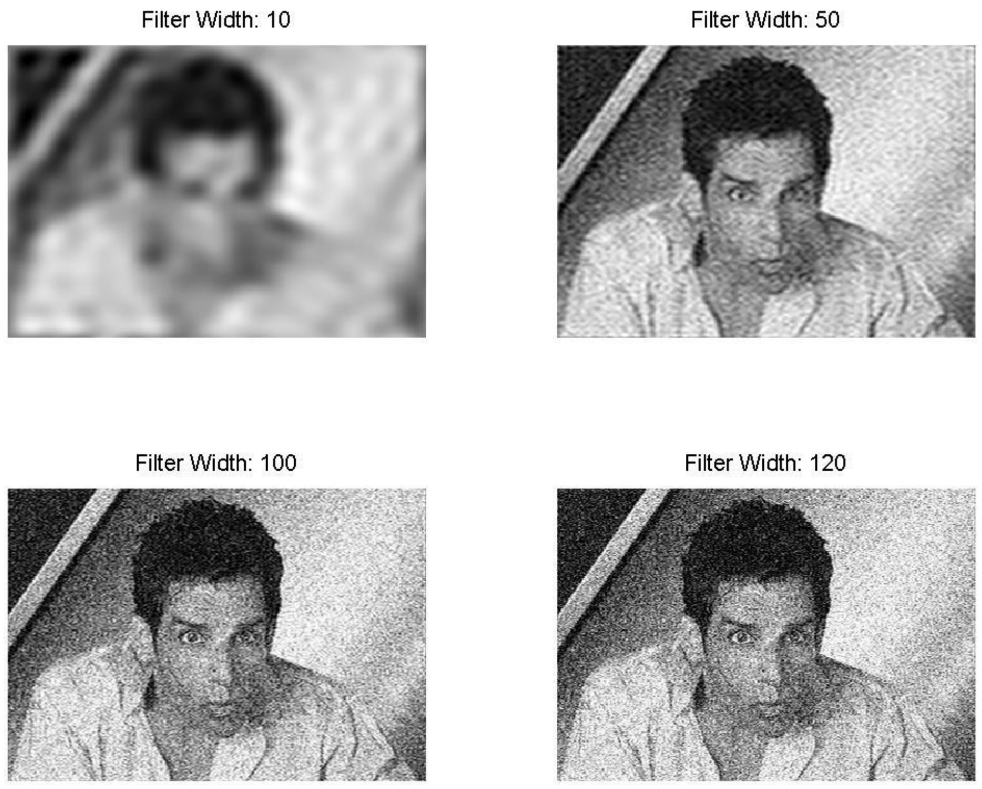

*Figure 5: These images are the results of applying the Shannon filters to the image of interest.*

The above filters show as expected that the harsher narrow filters blur the image where the maximally large filters do little to edit the noise of the original unfiltered image.

*See AMATH 482 Notes J Nathan Kutz p. 323

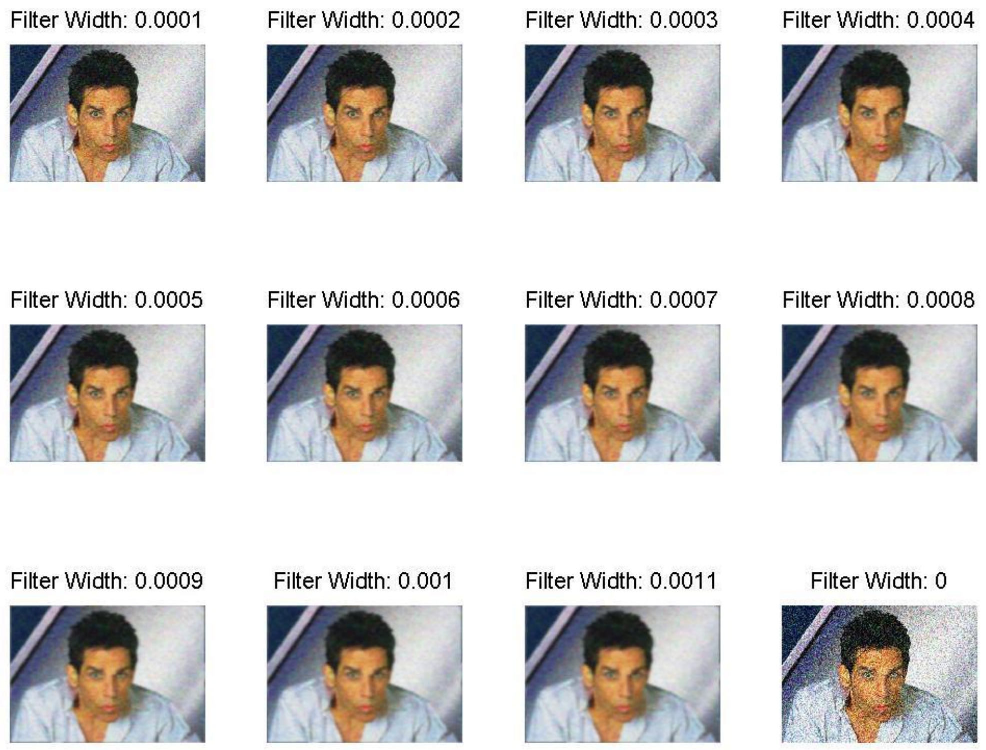

*Figure 6: The many numbered images show the effects of Gaussian filtering on the RGB image.*

Optimal levels of filtering seem to be with narrower widths than were preferential for the black and white image. Widths of around 0.0002-0.0004 seem to be equally beneficial and almost indistinguishable.

*See AMATH 482 Notes J Nathan Kutz p. 323

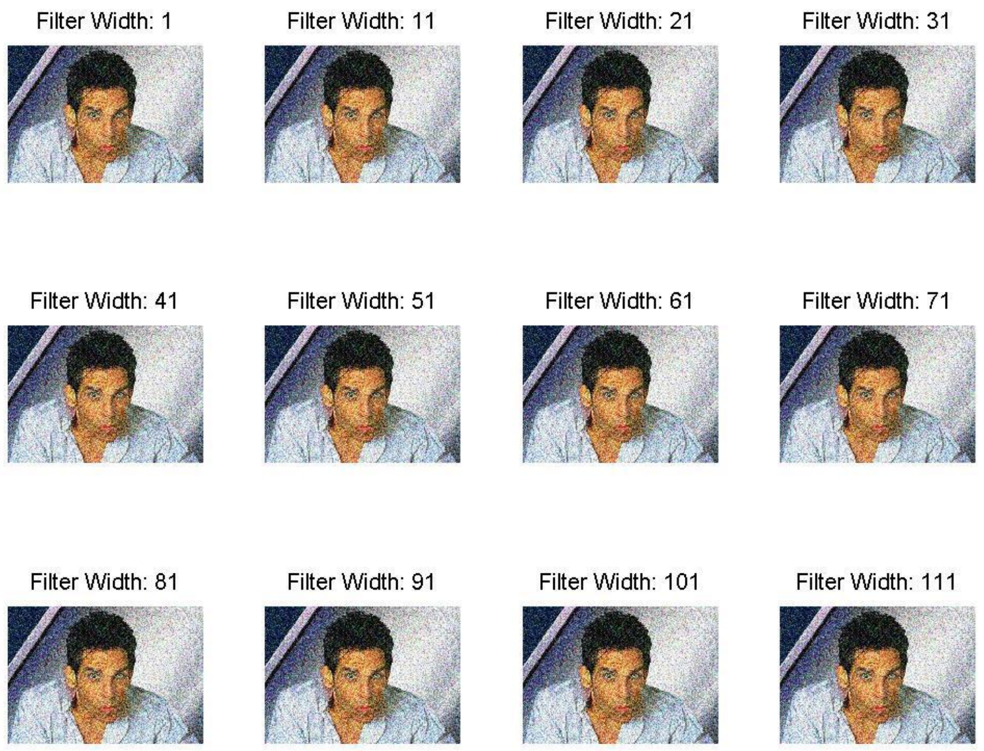

*Figure 7: This image shows the results of various Shannon filter widths.*

In contrary to the Gaussian filtering , there is very little noticeable use of the filters when using the Shannon filter. There is very little difference in the amount of noise at the various levels.

*See AMATH 482 Notes J Nathan Kutz p. 323

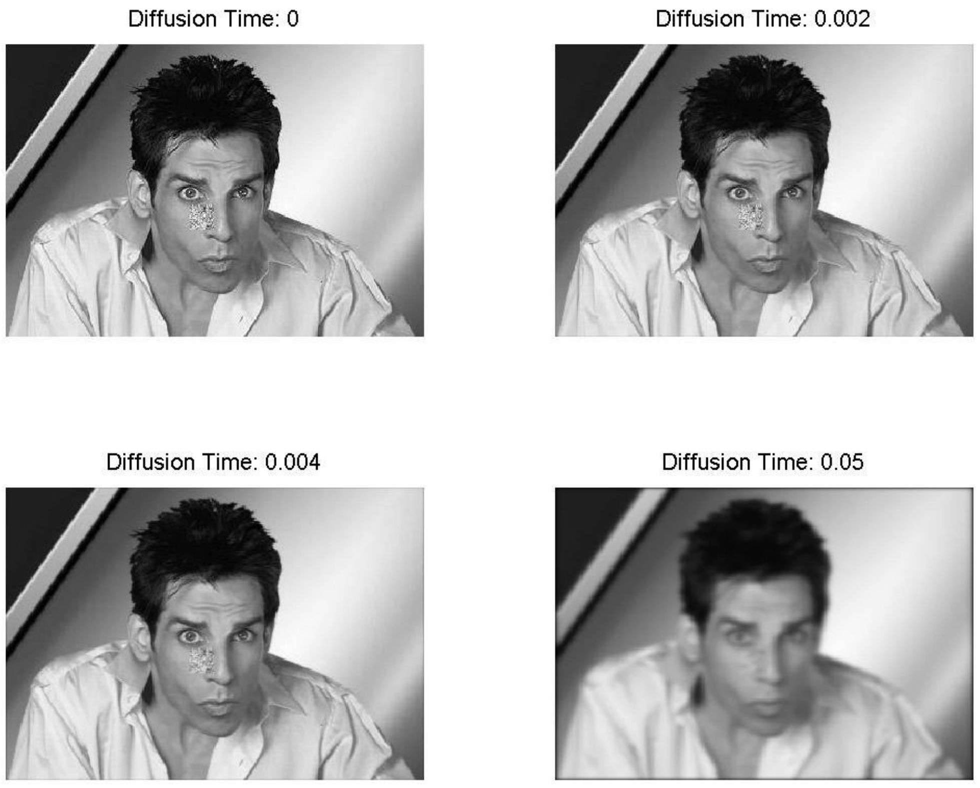

*Figure 8: This image shows the relation of diffusion and linear filtering. By applying the diffusion to the entire image, the results are similar to using a linear filtering scheme.*

While diffusion can be useful in denoising the whole image, it does us little good to be used on the whole image as we are only concerned with a small portion of the image. These images have a constant diffusion constant that is applied everywhere.

The following images were created by localizing the diffusion process through the use of a diffusion coefficient scheme that limited the window of application. It was similar in design to a Shannon filter, but instead was used to limit the spatial area (pixels) which were affected by the diffusion. The remainder of the image was left unchanged.

*See AMATH 482 Notes J Nathan Kutz p. 323

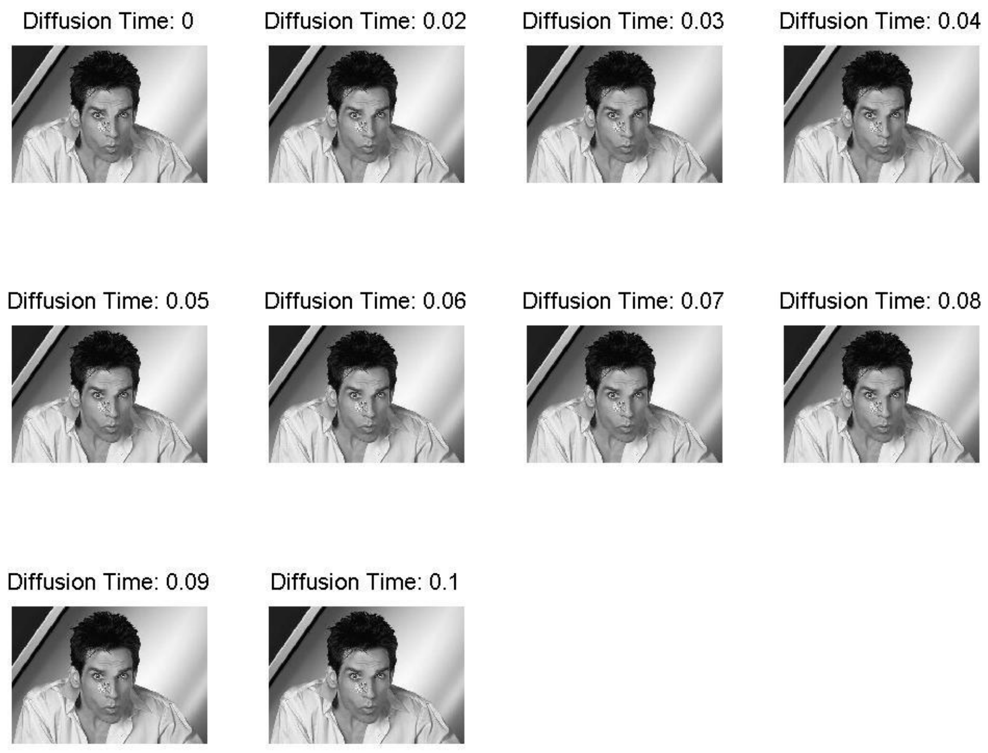

*Figure 9: This image shows various time lengths of diffusion for a localized window of width 10.*

The image above showed improvement towards the mid ranges and caused too little or too much blurring on the earlier and later images. It is pretty apparent that adjustment of the center of the filter needs to be done after this image as well.

*See AMATH 482 Notes J Nathan Kutz p. 323

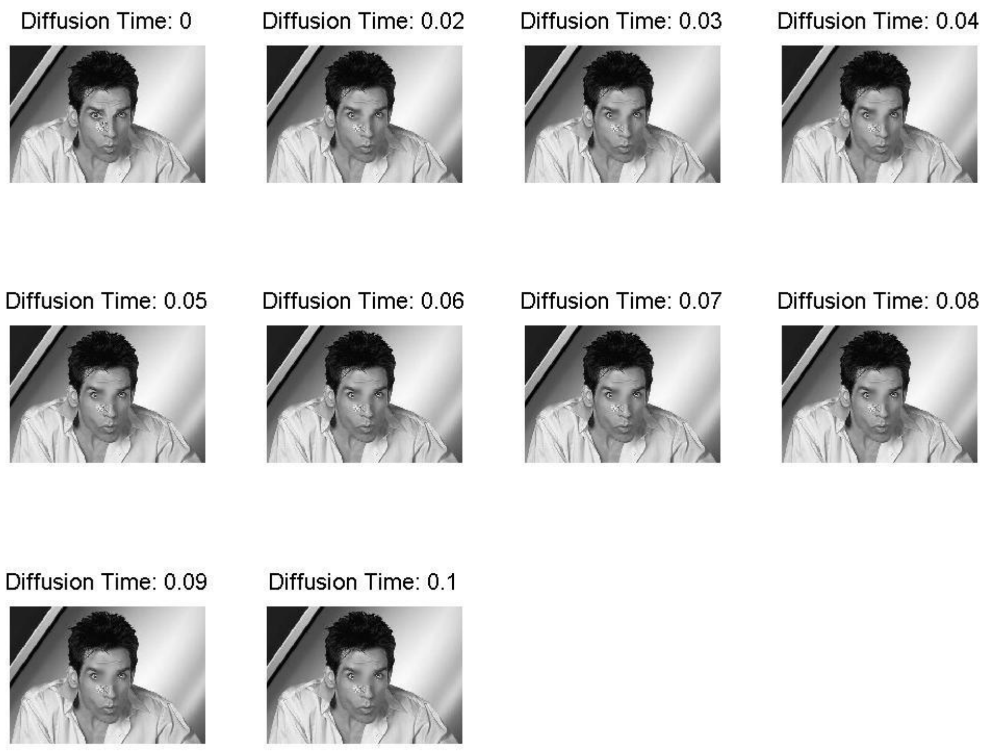

*Figure 10: This image shows the same center as above but a wider filter width. Again it is even more obvious that we need to adjust the center.*

*See AMATH 482 Notes J Nathan Kutz p. 323

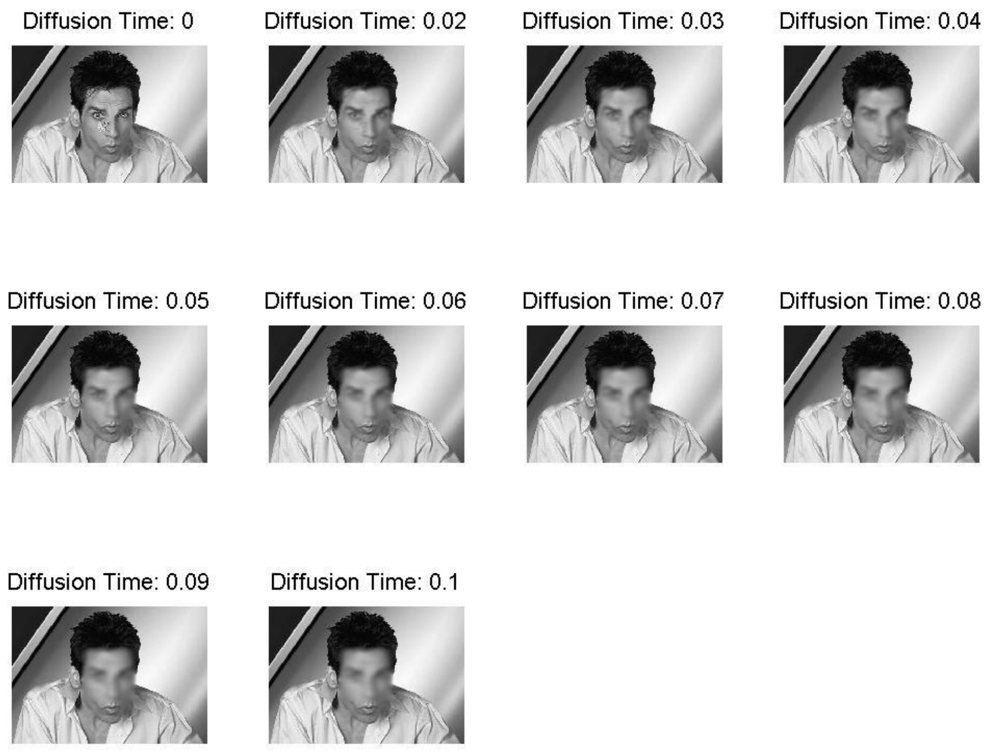

*Figure 11: The above image shows the extreme of a very large filter width that blurs the entire face, while still leaving the other image areas untouched.*

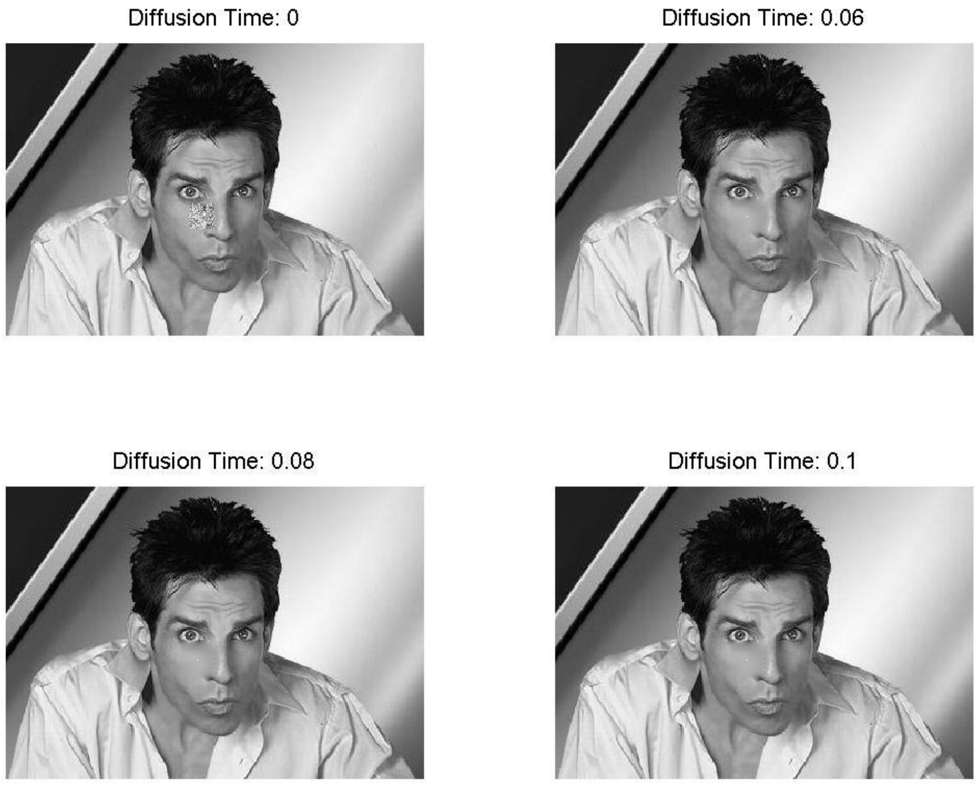

*Figure 12: This image had a window of 15 for the localization area, centered at a pixel location of 19/32 xc, 59/128 yc.*

The above image shows the use of a localized diffusion process with a small window that best localizes the diffusion to the target area. This amount of blurring and center for the filtering area was achieved via trial and error after an initial guess based on the proportional placement of the flaw. It is for this previous reason that the center location is in proportional changes of the true pixel center *(xc, yc).*

*See AMATH 482 Notes J Nathan Kutz p. 323

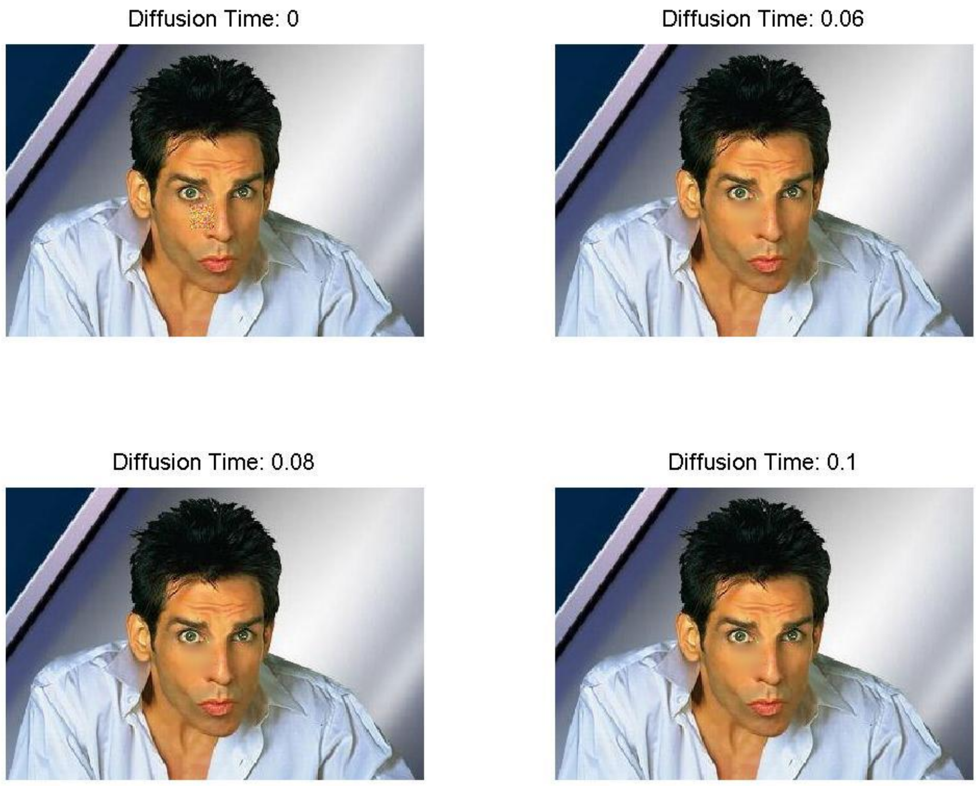

*Figure 13: This image shows the application of the previously explained process on all three layers of and RGB matrix. The width of the window is 15 and the diffusion constant is 0.001. The window is centered at the previously mentioned location.*

*See AMATH 482 Notes J Nathan Kutz p. 323

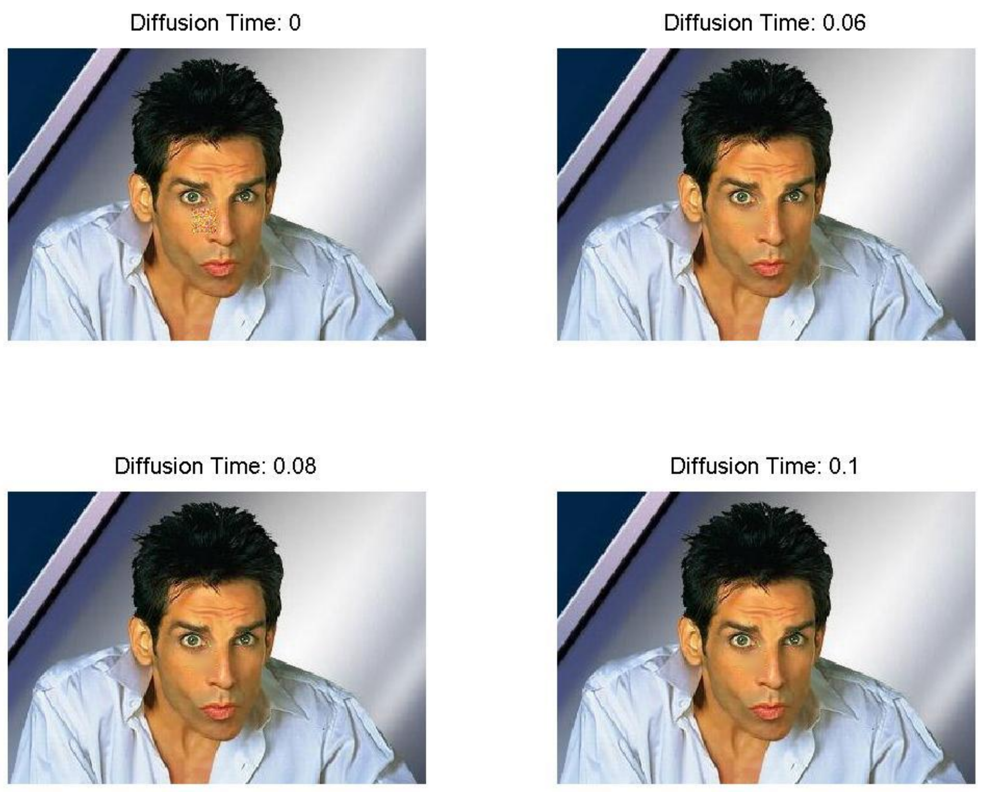

*Figure 14: This final image is identical to the last, except that the window with has been changed to 10. Same center and diffusion constant as above.*

The main variation in the last two images was caused by a modification in the width of the filter designed to avoid blurring of the right eye.

## 5. Summary and Conclusions

The applications of these various processes are far ranging in application. We have only scratched the surface of image processing as we examined the filtering of images by the use of *linear filters* of both Gaussian and Shannon-type forms. We followed up by applying localized diffusion to a problem area that enhanced the photo. Much of what was accomplished, as always, was through trial and error following an initial guess, but the theory and application of the underlying mathematics has obviously provided a systematic frame work that has far reaching applications.

*See AMATH 482 Notes J Nathan Kutz p. 323

## Appendix A:

```
%This function is used in conjunction with an ode function to
%perform the linear algebra diffusion computation on an image.
%This works for a BW image or a single layer of an RGB matrix
%
%limited to constant diffusion constant D
function rhs = image_rhs(t, u, dum, L, D)
    rhs = D*L*u;
end



%Joshua Borgman 12 February 2015
% Adapted from code courtesy of J Nathan Kutz @ University of WA
%This function is used in conjunction with an ode function to
%perform the linear algebra diffusion computation on an image.
%This works for a BW image or a single layer of an RGB matrix
%
%Diffusion constant D has been made variable so as to localize diffusion
%operation.

function rhs = image_rhs_var(t, u, dum, L, D)
    rhs = (L*u).*D;
end
```

## Appendix B:

```
%% Joshua Borgman  12 February 2015   STUDENT ID:1241267
% AMATH 482 Homework 3: Image Denoising with Filtering and Diffusion
Techniques
%{
This section loads a (preEstablished) black and white image and plots
the image side-by-side with its Fourier spectrum
%}

clear all; close all; clc;

Abw = imread('derek2', 'jpeg');
Abw = double(Abw);

[nx,ny] = size(Abw);

B = Abw;
Bt = fft2(B); Bts = fftshift(Bt);



xc = floor(nx/2)+1;
```

*See AMATH 482 Notes J Nathan Kutz p. 323

```
yc = floor(ny/2)+1;

subplot(2,2,1), imshow(uint8(B)), colormap(gray)
subplot(2,2,2), pcolor((log(abs(Bts)))); shading interp
colormap(gray), set(gca,'Xtick',[],'Ytick',[])
title(subplot(2,2,1), 'Original Corrupted Image'), title(subplot(2,2,2), ...
    'Fourier Spectrum (Original Image)');


%% This section shows the basic steps to building a Gaussian Filter
% width fixed at 0.0001, centered at half the pixel dimension space, (xc,yc)

kx=1:nx; ky=1:ny; [Kx,Ky]=meshgrid(kx,ky);
F=exp(-0.0001*(Kx-xc).^2-0.0001*(Ky-yc).^2);
Btsf=Bts.*F';

figure(1)
subplot(2,2,3), pcolor(log(abs(Btsf))); shading interp %plots log of shifted
Fourier spectrum
colormap(gray), set(gca,'Xtick',[],'Ytick',[])
Btf=ifftshift(Btsf); Bf=ifft2(Btf);
subplot(2,2,4), imshow(uint8(abs(Bf))), colormap(gray), %plots original image
title(subplot(2,2,3), 'Filtered Fourier Spectrum'), title(subplot(2,2,4),
'Filtered Image')


%% This section tries a variety of Gaussian Filter widths to view the impact
% on the image denoising. A filter width of 0 is the original image for
% reference
% Gaussian filters of varied widths

fs=[0.01 0.001 0.0001 0]; % List of desired filter widths

for j=1:length(fs)
    F=exp(-fs(j)*(Kx-xc).^2-fs(j)*(Ky-yc).^2); %Gaussian Filter
    Btsf=Bts.*F'; Btf=ifftshift(Btsf); Bf=ifft2(Btf);
    figure(4), subplot(2,2,j), pcolor(log(abs(Btsf)))
    shading interp,colormap(gray),set(gca,'Xtick',[],'Ytick',[])
    title(['Filter Width: ', num2str(fs(j))]),
    figure(5), subplot(2,2,j), imshow(uint8(Bf)), colormap(gray)
    title(['Filter Width: ', num2str(fs(j))]);
end

%% This section shows how to construct and apply a Shannon Filter of Width 50
% This is similar to the previous section but simply uses a different
% filter (i.e. step function)it is assumed that the filter has equal width
% in either direction

figure()
width=50;
Fs=zeros(nx,ny);
Fs((xc-width):1:(xc+width),(yc-width):1:(yc+width)) ...
    =ones(2*width+1,2*width+1);
Btsf=Bts.*Fs;
```

*See AMATH 482 Notes J Nathan Kutz p. 323

```
Btf=ifftshift(Btsf); Bf=ifft2(Btf);
subplot(1,2,1), pcolor(log(abs(Btsf))); shading interp
colormap(gray), set(gca,'Xtick',[],'Ytick',[])
subplot(1,2,2), imshow(uint8(Bf)), colormap(gray)

%% Multiple Window Widths for Shannon Filter
% This section allows for the viewing of several values of filter width,
% the same equality assumption as the previous section is made.

width=[10 50 100 120]; %varied filter widths
for j=1:4
    Fs=zeros(nx,ny);
    Fs((xc-width(j)):1:(xc+width(j)),(yc-width(j)):1:(yc+width(j))) ...
        =ones(2*width(j)+1,2*width(j)+1);
    Btsf=Bts.*Fs;

    Btf=ifftshift(Btsf); Bf=ifft2(Btf);
    figure(6), subplot(2,2,j), pcolor(log(abs(Btsf))); shading interp
    colormap(gray), set(gca,'Xtick',[],'Ytick',[])
    title(['Filter Width: ', num2str(width(j))]);
    figure(7), subplot(2,2,j), imshow(uint8(Bf)), colormap(gray)
    title(['Filter Width: ', num2str(width(j))]);
end

%% Processing of RGB Images
% This section applies the linear filters developed in the previous section
% to RGB images rather than simple grayscale images. The main principles
% are the same, we simply apply the transformations to all 3 sets of pixel
% strengths.

clear all, close all, clc
Aim = imread('derek1', 'jpeg');
A = double(Aim);

[nx,ny,nz] = size(A);

At = zeros(size(A));
Ats = zeros(size(A));
for j = 1:nz
    At(:,:,j) = fft2(A(:,:,j)); Ats(:,:,j) = fftshift(At(:,:,j));
end

xc = floor(nx/2)+1;
yc = floor(ny/2)+1;

%%
%Creates several Gaussian filters for RGB image.
% As is the case with all the Gaussian filters here, they are each centered
% at the central wavenumbers.
kx=1:nx; ky=1:ny; [Kx,Ky]=meshgrid(kx,ky);
w=[0.0001:0.0001:0.0011 0];

for jj=1:length(w)
F=exp(-w(jj)*(Kx-xc).^2-w(jj)*(Ky-yc).^2);
```

```
Atsf = zeros(size(Ats));
    for j=1:3
        Atsf(:,:,j)=Ats(:,:,j).*F';
        Atf(:,:,j)=ifftshift(Atsf(:,:,j)); Af(:,:,j)=ifft2(Atf(:,:,j));
    end
    figure(2), subplot(3,ceil(length(w)/3), jj),imshow(uint8(abs(Af)));
        title(['Filter Width: ', num2str(w(jj))]);
end

figure(1), imshow(uint8(abs(Af)));

%% Shannon Filter for RGB
% creates and applies various Shannon Filters to RGB image

width=[1:10:120];
for jj=1:length(width)
Fs=zeros(nx,ny);
Fs((xc-width(jj)):1:(xc+width(jj)),(yc-width(jj)):1:(yc+width(jj))) ...
    = ones(2*width(jj)+1,2*width(jj)+1);
    Atsf = zeros(size(Ats));
    for j=1:3
        Atsf(:,:,j)=Ats(:,:,j).*F';
        Atf(:,:,j)=ifftshift(Atsf(:,:,j)); Af(:,:,j)=ifft2(Atf(:,:,j));
    end
    figure(2), subplot(3,ceil(length(width)/3), jj),imshow(uint8(abs(Af)));
        title(['Filter Width: ', num2str(width(jj))]);
end

%% 2D Diffusion of image
%{
We can use a constant diffusion constant if we wish to mimic the linear
filtering
of the previous sections. However to edit D to be specific to certain
pixel values, we can note the relation of u matrix in notes. u =
u(1,1)....u(m,n)
where u(1,1) is pixel 1,1

For the exact position of the filter, I have proceeded via trial and error
noting the direction of increasing x and y.
%}
% This is the basic procedure to perform diffusion of a bw image

clear all, close all, clc
Bim = imread('derek4', 'jpeg');
B = double(Bim);

[nx,ny,nz] = size(B); %nz = 3 for RGB and 1 for BW

x=linspace(0,1,nx); y=linspace(0,1,ny); dx=x(2)-x(1); dy=y(2)-y(1);
onex=ones(nx,1); oney=ones(ny,1);
Dx=(spdiags([onex -2*onex onex],[-1 0 1],nx,nx))/dx^2; Ix=eye(nx);
Dy=(spdiags([oney -2*oney oney],[-1 0 1],ny,ny))/dy^2; Iy=eye(ny);
L=kron(Iy,Dx)+kron(Dy,Ix);

%kron function converts operator to higher dimensional
```

*See AMATH 482 Notes J Nathan Kutz p. 323

```
%space, similar to the use of meshgrid.

%This part construct the variable diffusion constant
Dvar = zeros(size(B));
spotcx = floor(nx*19/32);
spotcy = floor(ny*59/128);
width = 15;      % width of area to which we have applied diffusion
dConst = 0.001; %strength of diffusion constant when applied.
Dvar((spotcx-width):1:(spotcx+width),(spotcy-width):1:(spotcy+width)) ...
    =dConst*ones(2*width+1,2*width+1);
Dvar = reshape(Dvar, nx*ny, 1); % reshape for proper use in ode function

%Performs actual diffusion computation
tspan=[0 0.06:0.02:0.1]; u=reshape(B,nx*ny,1);
[t,usol]=ode113('image_rhs_var',tspan,u,[],L,Dvar);

%Here we have examined several diffusion times (tspan)
for j=1:length(t)
    A_clean=uint8(reshape(usol(j,:),nx,ny));
    A_clean(spotcx, spotcy) = 255;
    subplot(2,ceil(length(t)/2), j), imshow(A_clean);
    title(['Diffusion Time: ', num2str(t(j))]);
end

%% RGB Diffusion
%This extends the previous BW algorithm to an RGB image through the
% successive application of the same diffusion algorithm

clear all, close all, clc
Aim = imread('derek3', 'jpeg');
A = double(Aim);

[nx,ny,nz] = size(A);

x=linspace(0,1,nx); y=linspace(0,1,ny); dx=x(2)-x(1); dy=y(2)-y(1);
onex=ones(nx,1); oney=ones(ny,1);
Dx=(spdiags([onex -2*onex onex],[-1 0 1],nx,nx))/dx^2; Ix=eye(nx);
Dy=(spdiags([oney -2*oney oney],[-1 0 1],ny,ny))/dy^2; Iy=eye(ny);
L=kron(Iy,Dx)+kron(Dy,Ix);

Dvar = zeros(size(A(:,:,1)));
spotcx = floor(nx*19/32); spotcy = floor(ny*121/2^8);
width = 10; dConst = 0.001;
Dvar((spotcx-width):1:(spotcx+width),(spotcy-width):1:(spotcy+width)) ...
    =dConst*ones(2*width+1,2*width+1);
Dvar = reshape(Dvar, nx*ny, 1);

tspan=[0 0.06:0.02:0.1]; u = zeros(nx*ny,1,3);
%need to pre-declare to avoid possible scope issues
t = zeros(length(tspan),1,3); usol = zeros(length(t), nx*ny,3);

for k = 1:3
    u(:,:,k)=reshape(A(:,:,k),nx*ny,1);
    [t(:,:,k),usol(:,:,k)]=ode113('image_rhs_var',tspan,u(:,:,k),[],L,Dvar);
end
```

*See AMATH 482 Notes J Nathan Kutz p. 323

```
for j=1:length(t)
        A_clean=uint8(reshape(usol(j,:),nx,ny,3));
        subplot(2,ceil(length(t)/2), j), imshow(A_clean);
        title(['Diffusion Time: ', num2str(t(j))]);
end
```

*See AMATH 482 Notes J Nathan Kutz p. 323