AMATH482 HW03

Joshua Borgman

University of Washington

Josh Borgman

Thursday, February 12, 2015

Contents

[1. Introduction and Overview 3](#_Toc411516981)

[2. Theoretical Background 3](#_Toc411516982)

[Linear Filtering for Image Denoising 3](#_Toc411516983)

[Shannon Filters 3](#_Toc411516984)

[Diffusion in Image Processing 3](#_Toc411516985)

[3. Algorithm Implementation and Development 3](#_Toc411516986)

[Linear Filtering: 3](#_Toc411516987)

[Diffusion of Image Noise 3](#_Toc411516988)

[4. Computational Results 3](#_Toc411516989)

[5. Summary and Conclusions 3](#_Toc411516990)

[Appendix A: 4](#_Toc411516991)

[Appendix B: 4](#_Toc411516992)

# 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.

# Theoretical Background

## Linear Filtering for Image Denoising

## Shannon Filters

## Diffusion in Image Processing

# Algorithm Implementation and Development

## Linear Filtering:

## Diffusion of Image Noise

# Computational Results

# Summary and Conclusions

# 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 curtosey 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

%

%Difussion 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;

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

% refrence

% 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;

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 equallity 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 centeral 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 increaseing 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

%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 predeclare 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

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