HW 3: Saving Derek Zoolander

Anamol Pundle

February 12, 2015

Abstract

Corrupted images of the 3-time male model of the year Derek Zoolander have been partially restored in this work. The first set of images appear to have been corrupted with the addition of white noise, while Derek has an unseemly rash near his nose on the second. Linear filtering as well as diffusion techniques are used to restore Derek's pictures. A Gaussian filter is used on the images in frequency domain, which removes unwanted high frequency content. Another technique used is the removal of all frequency content whose amplitude falls below a certain threshold. Removal of the rash from the second set of images is accomplished by local diffusion of the rash. Different values of the parameters for diffusion, Gaussian filtering and thresholding are considered and the best parameters chosen.

1 Introduction and Overview

As with time-frequency analysis and de-noising of time signals, many applications of image processing deal with cleaning up images from imperfections, pixelation, and graininess, i.e. processing of noisy images. The objective in any image processing application is to enhance or improve the quality of a given image. In this section, the filtering of noisy images will be considered with the aim of providing a higher quality, maximally de-noised image.

In this work, grayscale and colored images are cleaned up using linear filtering, through the use of a Gaussian filter. Different filter widths are explored in order to get the best result. Removal of a rash, which is a localized noisy patch in the image, is undertaken using diffusion. The effect of the parameters of diffusion such as the time span and diffusion coefficient on the solution is also explored.

2 Theoretical Background

Linear filtering can be applied in the Fourier domain of the image in order to remove the high-frequency scale fluctuations induced by the noise. A simple filter to consider is a Gaussian that takes the form:

$$F(k_x, k_y) = exp(-\sigma_x(k_x - a)^2 - \sigma_y(k_y - b)^2)$$

where σ_x and σ_y are the filter widths in the x and y directions respectively and a and b are the center-frequency values for the corresponding filtering.

Filtering is not the only way to de-noise an image. Intimately related to filtering is the use of diffusion for image enhancement. Consider for the moment the simplest spatial diffusion process in two dimensions, i.e. the heat equation:

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

where u(x,y) will represent a given image, D is a diffusion coefficient, and some boundary conditions must be imposed. If for the moment we consider periodic boundary conditions, then the solution to the heat equation can be found from, for instance, the Fourier transform. The solution of the heat equation illustrates a key and critical concept: the wave numbers (spatial frequencies) decay according to a Gaussian function. Thus linear filtering with a Gaussian is equivalent to a linear diffusion of the image for periodic boundary conditions. To solve the heat equation numerically, we discretize the spatial derivative with a second-order scheme. This approximation reduces the partial differential equation to a system of ordinary differential equations. Once this is accomplished, then a variety of standard time-stepping schemes for differential equations can be applied to the resulting system.

3 Algorithm Implementation and Development

Different algorithms are developed for de-noising the first set of photographs and removing the rash from the second set. Both are listed in this section. All images are 253 pixels by 361 pixels in size.

3.1 Algorithm for de-noising first set of images

- Both the images (color and grayscale) are imported into MATLAB using the `imread` function.
  - The color image is stored as a $235 \times 361 \times 3$ matrix, i.e., a 235×365 matrix each for R, G and B data. The grayscale image is stored as a 235×361 matrix.
- Both images are converted into floating point numbers using the `double` function.
- Grid vectors are defined using the `meshgrid` function.
- The Fourier transform of each 235×361 matrix of the color and grayscale matrix is taken using the `fft2` function, and shifted using the `fftshift` function.

- A Gaussian function in x and y centered at centre of the frequency spectrum is defined. The spread of the function is calculated heuristically.
  - The Gaussian function used is $g(K_x, K_y) = e^{-D((K_x-126.5)^2+(K_y-180.5)^2)}$. The parameter D is varied to get the best result.
- Each frequency matrix is multiplied by the Gaussian function to form a new matrix.
- Separate from the previous step, all elements in each frequency matrix below a certain frequency threshold are set to zero.
- The inverse Fourier transform of each matrix is taken using the `ifft2` function.
- Images are plotted and parameters such as the frequency threshold and spread of the Gaussian function are varied to obtain the best result.
  - The `image` function is used to plot the color image, while the `imshow` function is used to plot the grayscale image.

3.2 Algorithm for removing the rash

- Both the images (color and grayscale) are imported into MATLAB using the `imread` function.
  - The color image is stored as a $235 \times 361 \times 3$ matrix, i.e., a 235×365 matrix each for R, G and B data. The grayscale image is stored as a 235×361 matrix.
- Both images are converted into floating point numbers using the `double` function.
- The rash portion of the image is identified and stored in a different matrix for both images.
- The sparse diagonal matrix for the x and y directions Ax and Ay is constructed using the spdiags function.
- A diffusion coefficient D is also defined. This is determined heuristically.
- The Laplacian operator L is constructed from Ax and Ay using the kron function.
- A time span vector for the diffusion of the image is defined.
- The image of the rash is reshaped into a $n_x \times n_y$ matrix, where n_x and $n_y$are defined by the size of the image.
  - This is done because the ODE solver function does not take a two dimensional input.

- The image is passed to the ODE solver function, ode113, along with the time span vector, the Laplacian vector, the diffusion coefficient and the function for the right hand side of the finite difference equation.
  - The right hand side function is given by $D(L\times\text{u})$ where u is the reshaped image matrix.
- The resulting matrices are reshaped into $n_x \times n_y$ matrices using the `reshape` function.
  - This is tricky for the color image, since three different matrices need to be reshaped and combined to get the final image.
- The rash images are re-inserted into the main image and displayed using the `image` function for the colored image and the `imshow` function for the grayscale image.

4 Computational Results

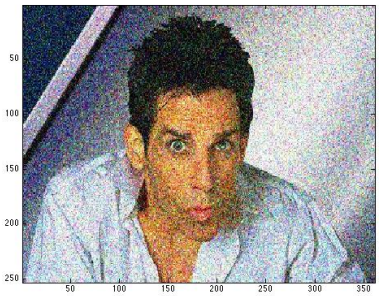
4.1 Task 1

Figure 1 shows the effect of Gaussian filtering on the colored image. Figure 1(a) is the unfiltered image, and is seen to have a lot of distortion due to noise. With a filter width factor D (as defined in the Gaussian function in Section 3) of 4×10^{-4}, some of the high frequency noise is taken out, though a blurring effect on the picture is noted. As D is increased to 10^{-3}, further noise reduction is noted, along with more blurring. Further increase of D does not have good results; the final image, with a D of 10^{-2}, is exceedingly blurred. Therefore, the image with $D = 4 \times 10^{-4}$ is arguably the best image. A similar effect is noted for the grayscale image. The image gets more blurred as D is increased (corresponding to a decrease in filter width). At a D of 10^{-2}, the image is virtually unrecognizable, whereas a D of 1×10^{-4} arguably gives the best result.

Figure 3 shows the result of the removing all frequencies with an amplitude below a certain threshold. Figure 3(a) shows the result of removing all frequencies with an amplitude less than $(1/800)^{th}$ of the maximum amplitude. Reasonably good filtering is achieved. Figure 3(b) shows the result of removing all frequencies with an amplitude less than $(1/500)^{th}$ of the maximum amplitude. The image is quite blurred. Therefore, lowering the threshold to a particular value may result in some filtering, but lowering it further will excessively blur the image.

4.2 Task 2

Figure 4 shows the result of performing diffusion on the rash portion colored image. It is found that a short time span and a relatively small diffusion constant gives good results. Figure 4(a) is the original image, where the rash is clearly



(a) Original Image

(b) Filter width factor D = 4×10^{-4}

(c) Filter width factor D =10^{-3}

(d) Filter width factor D =10^{-2}

Figure 1: Comparison of picture quality for three filter widths and the original colored image.

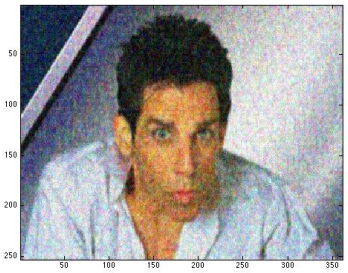
(a) Original image

(b) Filter width factor D = 1×10^{-4}

(c) Filter width factor D = 1×10^{-3}

(d) Filter width factor D = 1×10^{-2}

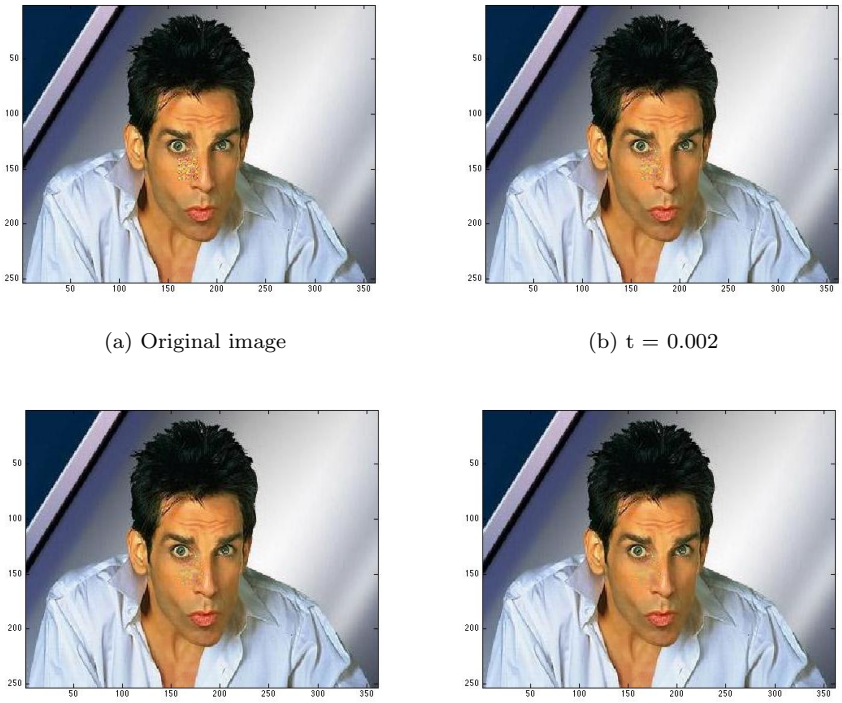
Figure 2: Comparison of picture quality for three filter widths and the original grayscale image.



(a) Threshold = $\frac{MaxAmplitude}{800}$

(b) Threshold = $\frac{MaxAmplitude}{500}$

Figure 3: Comparison of picture quality obtained by removing frequencies below different thresholds of amplitude



(a) Original image

(b) t = 0.002

(c) t = 0.004

(d) t = 0.01

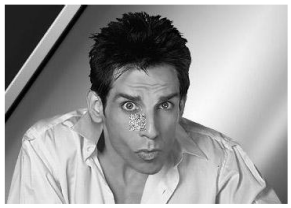
Figure 4: Image de-noising process for colored image for a diffusion constant of D = 0.1 as a function of time for t = 0, 0.002, 0.004 and 0.01.

visible. As more diffusion is applied, the rash starts to disappear. Figure 4(d) shows the final result of the diffusion, with a time of 0.01 and a diffusion constant of 0.1. The rash is almost invisible. The same steps are repeated for the black and white image, with results shown in Figure 5. The diffusion constant D is reduced to 0.08 for the black and white image, and the best result is found at t = 0.01.

Further, Gaussian filtering is applied to the image in order to smooth out the rash. Figure 6 shows the black and white image after a Gaussian filter with filter width constant $D = 10^{-4}$. The boundaries of the rash and the rest of the image get a little blurred, which leads makes the image look more realistic.

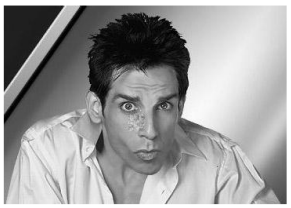
5 Summary and Conclusion

Corrupted colored and grayscale images of Derek Zoolander were filtered using a Gaussian filter with varying filter width. An appropriately sized Gaussian



(a) Original image

(b) t = 0.002

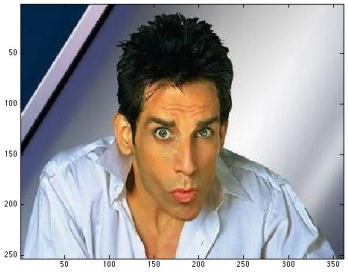


(c) t = 0.004

(d) t = 0.016

Figure 5: Image de-noising process for a grayscale image for diffusion constant of D = 0.08 as a function of time for t = 0, 0.002, 0.004 and 0.016.

(a) Image after diffusion filtered with Gaussian filter, $D = 10^{-4}$



(b) Image after diffusion filtered with Gaussian filter, $D = 4 \times 10^{-4}$

Figure 6: Images after diffusion filtered with Gaussian filter

filter was shown to be somewhat effective in cleaning up both images.

A rash was removed from Zoolander's images by applying diffusion locally to the images. It was seen that a relatively small diffusion constant and a short period of time is enough for the rash to be removed. A greater amount of time results in over-filtering the image. A Gaussian filter was applied to the images after removing the rash, in order to blur the boundary of the rash and the rest of the image.

References

1. Kutz, J. Nathan, Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data, September 2013

2. Matlab help

Appendix I: MATLAB functions used

`imread`: reads images into MATLAB

- `A = imread(FILENAME,FMT)` reads a grayscale or color image from the file specified by the string FILENAME. If the file is not in the current directory, or in a directory on the MATLAB path, specify the full pathname.

`double`: converts to double precision

- `double(X)` returns the double precision value for X. If X is already a double precision array, `double` has no effect.

`uint8`: converts to unsigned 8-bit integer

- `I = uint8(X)` converts the elements of the array X into unsigned 8-bit integers. X can be any numeric object, such as a DOUBLE.

`spdiags`: sparse matrix formed from diagonals

- `[B,d] = spdiags(A)` extracts all nonzero diagonals from the m-by-n matrix A. B is a min(m,n)-by-p matrix whose columns are the p nonzero diagonals of A. d is a vector of length p whose integer components specify the diagonals in A.

`reshape`: reshapes array

- `reshape(X,M,N)` or `reshape(X,[M,N])` returns the M-by-N matrix whose elements are taken columnwise from X. An error results if X does not have M*N elements.

`imshow`: displays image

- `imshow(I)` displays the image I in a Handle Graphics figure, where I is a grayscale, RGB (truecolor), or binary image. For binary images, `imshow` displays pixels with the value 0 (zero) as black and 1 as white.

`ode113`: solve non-stiff differential equations, variable order method.

- `[TOUT,YOUT] = ode113(ODEFUN,TSPAN,Y0)` with `TSPAN = [T0 TFINAL]` integrates the system of differential equations y' = f(t,y) from time T0 to TFINAL with initial conditions Y0. `ODEFUN` is a function handle. For a scalar T and a vector Y, `ODEFUN(T,Y)` must return a column vector corresponding to f(t,y). Each row in the solution array `YOUT` corresponds to a time returned in the column vector `TOUT`. To obtain solutions at specific times `T0,T1,...,TFINAL` (all increasing or all decreasing), use `TSPAN = [T0 T1 ... TFINAL]`.

Appendix II: MATLAB code

MATLAB code for task 1

```
clear all; close all; clc

% read images
Img1 = imread('derek1','jpg');
Img2 = imread('derek2', 'jpg');

%convert images to double precision
Img1 = double(Img1);
Img2 = double(Img2);
[nx ny nz] = size(Img1);

%define parameters for Gaussian function
gaussmult = 0.0001; gaussmult2 = 0.0001;

%define grid vectors
x = 1:nx; y = 1:ny; [Kx, Ky] = meshgrid(x, y);

for ii = 1:3
    Img1f = fftshift(fft2(Img1(:,:,ii))); %fft of image matrix
    %define Gaussian function
    gauss = exp(-gaussmult*(Kx-nx/2).^2 - gaussmult*(Ky-ny/2).^2);

    Img1f = Img1f.*gauss';
    Imax = max(max(abs(Img1f)));
    %Set threshold and set everything below threshold to 0
    Img1cleara(:,:,ii) = ifft2(Img1f);
    Img1f = fftshift(fft2(Img1(:,:,ii)));
```

```
        for jj = 1:nx
            for kk = 1:ny
                if abs(Img1f(jj, kk)) < (Imax/800)
                    Img1f(jj, kk) = 0;
                end
            end
        end
        %Inverse fft of image
        Img1clearb(:,:,ii) = ifft2(Img1f);
    end

    Img1cleara = uint8(abs(Img1cleara));
    Img1clearb = uint8(abs(Img1clearb));

    %Same steps for grayscale image
    Img2f = fftshift(fft2(Img2));
    gauss = exp(-gaussmult*(Kx-126.5).^2 - gaussmult*(Ky-180.5).^2);

    Img2f = Img2f.*gauss';
    Img2clear = (abs((ifft2(Img2f))));
```

MATLAB code for task 2

```
    clear all; close all; clc;

    %read images
    Img1 = imread('derek3', 'jpg');
    Img2 = imread('derek4', 'jpg');

    gaussmult = 0.0001;

    %save rash in different matrix
    Rash1 = Img1(135:165, 155:185, :);
    Rash2 = Img2(135:165, 155:185);
    Rash1 = double(Rash1);
    Rash2 = double(Rash2);

    [nx, ny, nz] = size(Rash1);
    x = linspace(0,1,nx); y = linspace(0,1,ny);
    dx = x(2)-x(1); dy = y(2)-y(1);

    %define Laplacian matrix
    el = ones(nx,1); Ax = spdiags([el -2*el el],[-1 0 1],nx,nx)/(dx^2);
Ix = eye(nx);
```

```
Ay = spdiags([el -2*el el],[-1 0 1],ny,ny)/(dy^2); Iy = eye(ny);

L = kron(Iy, Ax) + kron(Ay, Ix);

tspan = [0 0.002 0.004 0.01]; D = 0.1;
Rash1r = reshape(Rash1, nx*ny, 3); %reshape for ode113

for ii = 1:3
    %solve for diffusion
    [t usol1] = ode113('RHSfunc', tspan, Rash1r(:, ii),[],L, D);

    u1(:,:,ii) = usol1;
end

for ii = 1:3
    %reshape to display
    U1(:,:,ii) = reshape(u1(4,:,ii), nx, ny);
end

%add diffused rash back to original image
Img1(140:160, 160:180, :)  = U1(6:26, 6:26, :);

%repeat for grayscale image
Rash2r = reshape(Rash2, nx*ny, 1);
[t usol2] = ode113('RHSfunc', tspan, Rash2r,[],L, D);
U2 = reshape(usol2(4,:),nx, ny);
Img2(140:160, 160:180, :)  = U2(6:26, 6:26, :);

%Gaussian filtering on image
x = 1:253; y = 1:361; [Kx, Ky] = meshgrid(x, y);
gauss = exp(-gaussmult*(Kx-126.5).^2 - gaussmult*(Ky-180.5).^2);

Img2f = fftshift(fft2(Img2));
Img2f = Img2f.*gauss';
Img2clear = (abs((ifft2(Img2f))));
```