Math 521 Homework 5

Due on Tuesday, May 8, 2018

Kristin Holmbeck

Contents

List of Figures

Theory

1. Fourier series coefficient shift

Suppose $f(w)$ has the Fourier series representation $f(w) = \sum_{k\in\mathbb{Z}} c_k e^{ikw}$ and suppose that $g(w) = e^{imw} f(w)$. For some $m \in \mathbb{Z}$, show that

$$g(w) = \sum_{k=-\infty}^{\infty} d_k e^{ikw}, \quad \text{where} \quad d_k = c_{k-m}$$

$$\begin{aligned} g(w) = e^{imw} f(w) &= e^{imw} \sum_{k\in\mathbb{Z}} c_k e^{ikw} \\ &= \sum_{k\in\mathbb{Z}} c_k e^{iw(k+m)} \\ &= \sum_{h\in\mathbb{Z}} c_{h-m} e^{iwh} \qquad \text{by setting } h = k+m \end{aligned}$$

2. Fourier series computation (continuous case)

Find the Fourier series for the 2π-periodic square wave function

$$f(\omega) = \begin{cases} -k & \text{if} \quad -\pi < \omega < 0 \\ k & \text{if} \quad 0 < \omega < \pi \end{cases} \quad \text{and} \quad f(\omega + 2\pi) = f(\omega)$$

Since f is 2π-periodic, we can represent its Fourier series by $f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx}$, where (as derived in class) the coefficients c_k are given by

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-ikx} dx.$$

Using this, let's solve for the coefficients (when $k \neq 0$):

$$\begin{aligned} c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-ikx} dx &= \frac{1}{2\pi} \left[\int_{-\pi}^{0} -k e^{-ikx} dx + \int_{0}^{\pi} k e^{-ikx} dx \right] \\ &= \frac{k}{2\pi} \left[\int_{0}^{\pi} e^{-ikx} dx - \int_{-\pi}^{0} e^{-ikx} dx \right] \qquad (\text{note that } c_0 = 0) \\ &= \frac{k}{2\pi} \left[\frac{-1}{ik} \, e^{-ikx} \Big|_{x=0}^{\pi} + \frac{1}{ik} \, e^{-ikx} \Big|_{x=-\pi}^{0} \right] \\ &= \frac{-1}{2\pi i} \left[e^{-ik\pi} - 1 - \left(1 - e^{ik\pi} \right) \right] = \frac{i}{2\pi} \left[e^{ik\pi} + e^{-ik\pi} \right] \\ &= \frac{e^{2ik\pi} + 1}{2\pi e^{ik\pi}} \\ \implies c_k &= \begin{cases} 0 & k = 0 \\ \frac{e^{2ik\pi}+1}{2\pi e^{ik\pi}} & k \neq 0 \end{cases} \end{aligned}$$

Thus, the Fourier series for $f(\omega)$ is given by

$$f(x) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \frac{e^{2ik\pi} + 1}{e^{ik(\pi - x)}}$$

3. Wavelet decomposition

Compute by hand the Haar wavelet decomposition (Pyramidal decomposition) of the vector $x^T = [1, 7, 3, 2]$ by viewing it as

$$f(x) = \begin{cases} 1 & \text{if} \quad x \in [0, \frac{1}{4}) \\ 7 & \text{if} \quad x \in [\frac{1}{4}, \frac{1}{2}) \\ -3 & \text{if} \quad x \in [\frac{1}{2}, \frac{3}{4}) \\ 2 & \text{if} \quad x \in [\frac{3}{4}, 1) \end{cases} \quad \text{and} \quad f(\omega + 2\pi) = f(\omega)$$

Graphically show the projections onto the scaling and wavelet subspaces

We need to arbitrarily specify the size of the smallest scale to start the pyramidal decomposition algorithm. We will choose $\dim f = 4 = 2^{-j}$, thus the finest resolution required is at the V_{-2} level. For the Haar wavelet, we can represent levels as

$$\phi^{j+1}(x) = \sqrt{(2)} \sum_n h_n \phi^j(x)$$

This will involve the subspaces

$$V_{-2} = \text{span}\{2\phi(4x - k)\}$$
$$V_{-1} = \text{span}\{\sqrt{2}\phi(2x - k)\}$$
$$V_0 = \text{span}\{\phi(x - k)\}$$

where $k \in \mathbb{Z}$ for each subspace.

Computing

1. Median Filter for Noise Reduction

Implement a 3×3 *median filter* and apply the filtering process on a corrupted image of app-ndt-Chip5.JPG located via the course website. Specifically, corrupt app-ndt-Chip-5.JPG with *salt-and-pepper* noise, where the corrupted pixels are either set to the maximum value (which looks like snow in the image) or have single bits flipped over. In some cases, single pixels can be set alternatively to zero or to the maximum value (i.e., 255 on a 8-bit machine). Then apply the median filter to de-noise the corrupted image. Compare your result with the original.

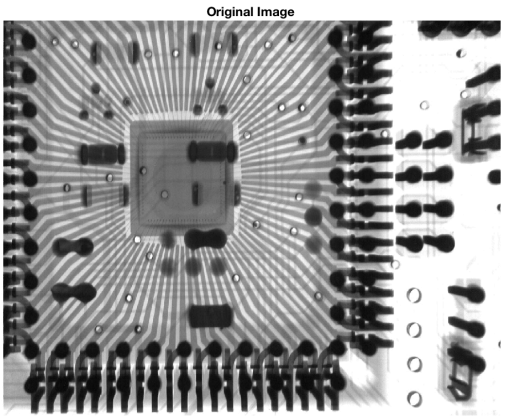

Figure 1: Original Image

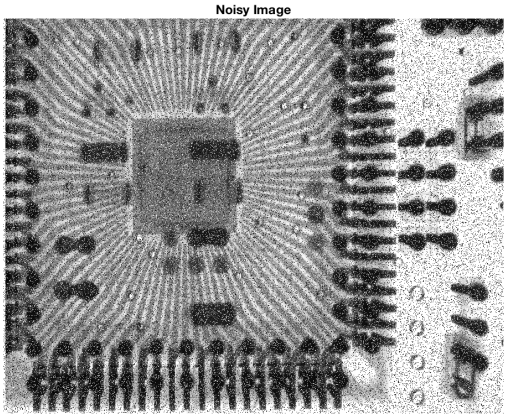

Figure 2: Salt-and-Pepper Image

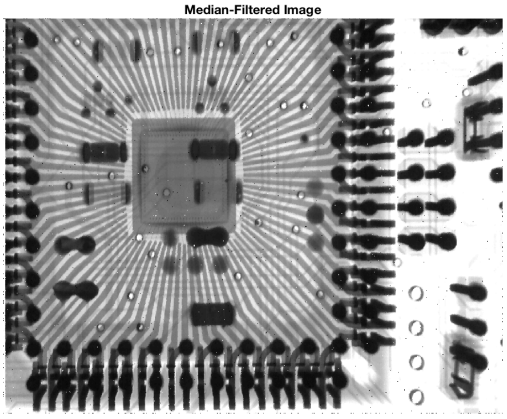

Figure 3: 3×3 Median-Filtered Image

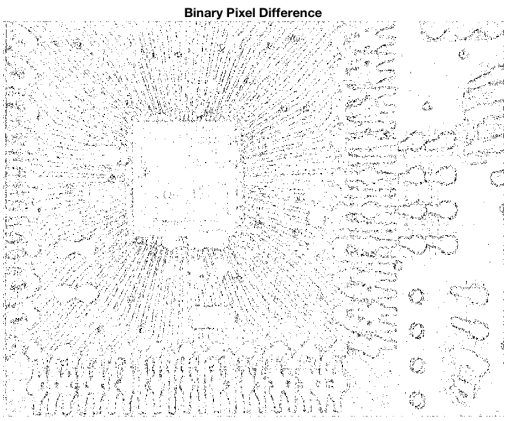

Figure 4: Pixel difference between original and filtered image

In visualizing absolute difference between the true image (in Figure 1) and the image that comes from applying a 3×3 median filter to Figure 2, it is unclear to see where the images differ. The plot shown in Figure 4 comes from the *locations* where the images differ, as opposed to the intensity values.

2. Laplacian Filter for Image Sharpening

Given an image CTimage.JPG on the course website, perform the following operations:

a. Construct a 3×3 average filter to smooth the image.

b. Then use a 2D Laplacian filter mask to extract the edges of the smoothed image.

c. Finally, enhance the smoothed image with the result from part (b). How does this image compare to the original?

We begin with the original image (Figure 5) and apply a 3×3 averaging filter to obtain the image in Figure 6. We then apply a Laplacian filter, which approximates the 2^{nd} derivative at each pixel. Since the 2^{nd} derivative is acceleration, the Laplacian of an image is akin to extracting information on *edges* in the image – sharp changes between background and foreground. By adding the edge information back into an image, we can *sharpen* features and thereby enhance the original, given image. An example is shown below with some CT scan data.

To perform this process, we need masks representing the averge and the Laplacian of a small region. If both masks are 3×3,

$$m_{\text{avg}} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \text{and} \qquad m_{\text{Laplace}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Convolving m_{avg} with each pixel in the original image, we end up generating, for each pixel, an average of its neighbors in an 3×3 neighborhood, as shown in Figure 6. We will call this our "given" image, i.e. given a blurred image, sharpen it. Now, apply the Laplace mask to this averaged image to obtain edge information (Figure 7). To obtain the final sharpened image, we can add or subtract the Laplacian image.

According to [1], if we use the Laplacian filter m_{Laplace} as defined above, then the sharpened image is given by $I_{\text{sharp}} = I_{\text{orig}} - I_{\text{Laplacian}}$. We could have also used the negative of the Laplacian filter (with positive 4 in the center) to generate the sharpened image via the *addition* of the original and Laplacian-filtered image.

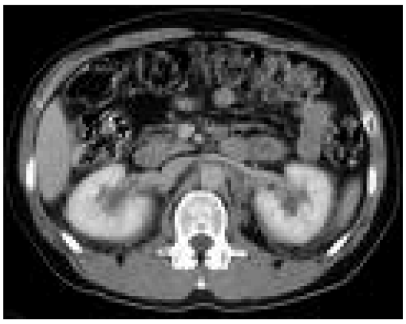

Figure 5: Original CT Scan ("True" Image)

Figure 6: Average-Filtered CT Scan

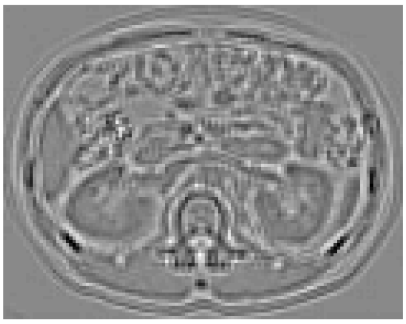

Figure 7: Laplacian-filtered CT Scan from Figure 6

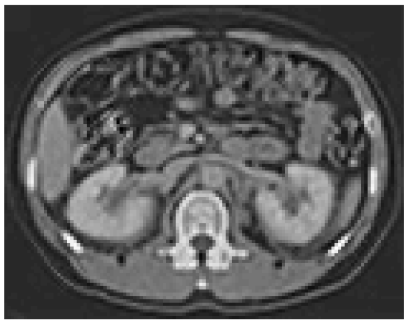

Figure 8: CT Scan sharpened from Figure 6

Furthermore, let's look at the difference between the "true", original image (Figure 5) and the sharpened image (Figure 8), which is shown in Figure 9. This difference looks very similar to the edge information from Figure 7. Indeed, we show in Figure 10 that (up to machine precision) $I_{\text{true}} - I_{\text{sharp}} = I_{\text{Laplacian}}$.

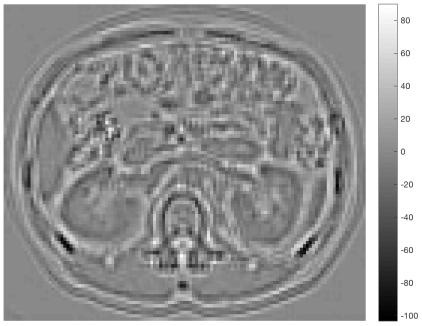

Figure 9: Difference between "true" and sharpened image

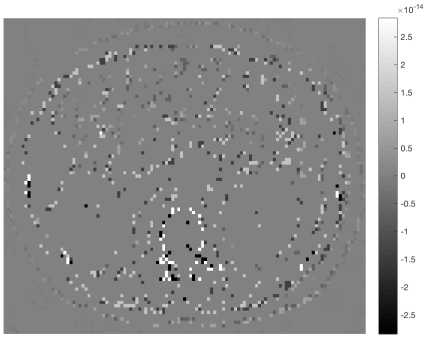

Figure 10: $I_{\text{true}} - I_{\text{sharp}} = I_{\text{Laplacian}}$

3.

Test your codes with your favorite image for this problem. Make sure your codes are as general as possible and be sure to plot the results.

a. Write a function in MATLAB to compute the approximation and each of the three detail components of an image. (i.e., you will produce 4 *extremely* short codes here) Notice that the resolution of LL, HL, LH, and HH will be $M/2 \times N/2$, where (M, N) is the resolution of the original image.

b. Write a subroutine to reconstruct an image from **only** the *approximation* component as a function of level. Notice that the resolution of the reconstructed image will be of size $M \times N$.

c. Compress the image up to level 3 using **only the approximation component**. Compute the compression ratio as a function of each compression level. Plot the compressed image for each level along with their compression ratio. Note that the compression ratio in this case can be defined as

$$CR = \frac{\#\text{ of nonzero entries in the original}}{\#\text{ of nonzero entries in the compressed}}$$

Code

Image Filtering

This code was created to hande several methods, including median, average, and Laplacian filters.

```
function im_out = imfilt(im, filtType, filtSiz)
% IMFILT
% Image filtering.
%
% Syntax:
%   im_out = IMFILT(im_in, filtType)
%   im_out = IMFILT(im_in, filtType, filtSiz)
%
% where filterSiz is a 2-element vector specifying the size of the median
% filter, and filtType is a string that can have the following values:
%   'median'
%   'average'
%   'laplacian' (uses the 3x3 matrix with -4 at the center)
%

% Author: Kristin Holmbeck, April 2018

if nargin == 2
    filtSiz = [3,3];
end

filtType = lower(filtType);
switch filtType
    case 'median'
        ffun = @median;
    case 'average'
%         ffun = @mean;
        AvgFilt = ones(filtSiz) / prod(filtSiz);
        ffun = @(x) sum(AvgFilt(:).*x);
    case 'laplacian'
        LapFilt = [0,  1, 0;
                   1, -4, 1;
                   0,  1, 0];
        filtSiz = size(LapFilt);
        ffun = @(x) sum(LapFilt(:).*x);
    otherwise
        error('Filter type not defined for this method');
end

% Created a padded image
im_siz      = size(im);
im_pad      = zeros(im_siz + 2*filtSiz);
im_new      = im_pad;
sub_rows    = (filtSiz(1)-1)+[1:im_siz(1)];
sub_cols    = (filtSiz(2)-1)+[1:im_siz(2)];
im_pad(sub_rows, sub_cols) = im;

fs = floor(filtSiz/2);      % not sure if this line is correct in general
for ii = sub_rows
    for jj = sub_cols
        % get neighbors of current pixel
        % apply the filter to that neighborhood
        dat = im_pad(ii-fs(1):ii+fs(1), jj-fs(2):jj+fs(2));
        im_new(ii,jj) = ffun( dat(:) );
    end
end
```

```
57
58  im_out = im_new(sub_rows, sub_cols);
59
60  end
```

References

[1] Chang, Jen-Mei. *Matrix Methods for Geometric Data Analysis and Recognition*. 2014.