## Homework #2 (Temporary)

**FFT packages:** http://www.netlib.org/fftpack, vfftpack, fishpack;
**Matlab Demo:** Toolboxes, Image Processing, Wavelet Toolbox (wvdtool)

**(1)** Explore the efficiency in FFT

Find the number of operations, $c(n)$, for `fft` in Matlab for vectors of length $n = 1, 2, \cdots$ (Use the `flops` command). Plot, on a log-log scalar, $n^2/c(n)$ and $(n \log n)/c(n)$. For which $n$ does `fft` behave like a "slow" Fourier transformation? For which values of $n$ besides powers of 2 does `fft` behavior like "fast transformation"?

**(2)** Explore fast Poisson solver.

We start with 1-d Poisson equation:

$$u''(x) = f(x), \quad \text{for } 0 < x < 1 \tag{1}$$

for each of the following possible boundary value specifications:

$$\text{Dirichlet-Dirichlet:} \qquad u(0) = u(1) = 0 \tag{2}$$

$$\text{Neumann-Neumann:} \qquad u'(0) = u'(1) = 0 \tag{3}$$

$$\text{Dirichlet-Neumann:} \qquad u(0) = u'(1) = 0 \tag{4}$$

$$\text{Periodic:} \qquad u(0) = u(1) \tag{5}$$

We take the following computational grids:

$$x_j = jh, \qquad \text{for } j = 0, 1, \cdots, N$$

where $h = 1/N$ is the grid-size and $N = 2^k$ for some integer $k$. The second order finite difference approximation of (1) is given by:

$$\frac{u(x_{j+1}) - 2u(x_j) + u(x_{j-1})}{h^2} = f(x_j) \tag{6}$$

and the boundary conditions are given by:

$$\text{Dirichlet-Dirichlet:} \qquad u(x_0) = u(x_N) = 0 \tag{7}$$

$$\text{Neumann-Neumann:} \qquad \frac{u(x_0 + h) - u(x_0 - h)}{2h} = \frac{u(x_N + h) - u(x_N - h)}{2h} = 0 \tag{8}$$

$$\text{Dirichlet-Neumann:} \qquad u(x_0) = \frac{u(x_N + h) - u(x_N - h)}{2h} = 0 \tag{9}$$

$$\text{Periodic:} \qquad u(x_0) = u(x_N) \tag{10}$$

for (2)-(5) respectively.

Note that for the case of Neumann-Neumann or Periodic, there is a free constant in the solution. We can normalize the solution by setting

$$\int_0^1 u(x)\, dx = 0\,. \tag{11}$$

**(a)** The choice of Fourier expansion is based on the boundary conditions. Corresponding to the case (2)-(5), we represent the solution in following trigonometric series:

$$u_j = \sum_{k=1}^{N-1} \hat{u}_k \sin\left(\frac{jk\pi}{N}\right), \quad j = 1, 2, \cdots, N-1 \tag{12}$$

$$u_j = \frac{1}{2}\hat{u}_0 + \frac{(-1)^j}{2}\hat{u}_N + \sum_{k=1}^{N-1} \hat{u}_k \cos\left(\frac{jk\pi}{N}\right), \quad j = 0, 1, \cdots, N \tag{13}$$

$$u_j = \sum_{k=1}^{N} \hat{u}_k \sin\left(\frac{j(2k-1)\pi}{2N}\right), \quad j = 1, 2, \cdots, N \tag{14}$$

$$u_j = \frac{1}{2}a_0 + \frac{(-1)^j}{2}a_{N/2} + \sum_{k=1}^{N/2-1}\left(a_k \cos\left(\frac{2jk\pi}{N}\right) + b_k \sin\left(\frac{2jk\pi}{N}\right)\right) \tag{15}$$

respectively. The Fourier coefficients in (12)-(15) can be expressed in the following inverse transformation:

$$\hat{u}_k = \frac{2}{N}\sum_{j=1}^{N-1} u_j \sin\left(\frac{jk\pi}{N}\right), \quad k = 1, 2, \cdots, N-1 \tag{16}$$

$$\hat{u}_k = \frac{2}{N}\left(\frac{1}{2}u_0 + \frac{(-1)^k}{2}u_N + \sum_{j=1}^{N-1} u_j \cos\left(\frac{jk\pi}{N}\right)\right), \quad k = 0, 1, \cdots, N \tag{17}$$

$$\hat{u}_k = \frac{2}{N}\left(\sum_{j=1}^{N-1} u_j \sin\left(\frac{j(2k-1)\pi}{2N}\right) + \frac{(-1)^{k-1}}{2}u_N\right), \quad k = 1, 2, \cdots N \tag{18}$$

$$a_k = \frac{2}{N}\sum_{j=0}^{N-1} u_j \cos\left(\frac{2jk\pi}{N}\right), \quad k = 0, 1, \cdots, N/2$$

and

$$b_k = \frac{2}{N}\sum_{j=0}^{N-1} u_j \sin\left(\frac{2jk\pi}{N}\right), \quad k = 1, \cdots, N/2-1 \tag{19}$$

respectively. Be aware that the exact formulation in (12)-(19), especially the normalizing constants, may be different in FFT packages. You shall read the documentations of the package first.

(i) Verify that the solution in the form of (12)-(15) satisfies the boundary conditions (7)-(10), respectively.

(ii) Express the right hand side $f$ in (6) in a similar transformation and find the Fourier multiplier $\lambda_k$ such that $\lambda_k \hat{u}_k = \hat{f}_k$. Point out when $\lambda_k = 0$.

**(b)** For each of the possible boundary conditions (2)-(5), derive a fast Poisson solver and implement them with the corresponding transformations in (12)-(19).

**(c)** Extend **(b)** to general domain $[a, b]$ and inhomogeneous boundary condition. Write a subroute with input: $a$, $b$, $N$, $f$, and the boundary data, and output: $u$, the solution satisfying the boundary conditions.

**(d)** Perform a truncation error estimate for (6) with Dirichlet boundary condition or - Neumann boundary condition. Use the code in **(c)** to check the accuracy with $N = 16, 32, 64, \cdots$.

**(e)** (optional) Extend **(c)-(d)** to 2D case and write some generic fast Poisson solvers. Compare the numerical performance (CPU time) in a set of $128 \times 128$ or $256 \times 256$ to any of numerical methods you have used such as Gauss-Seidel, Gaussian elimination, multi-grid methods, find out how much efficiency you have gained with the fast Poisson solver.

**(3)** Explore image compression and Wiener Filter

An "image" is a function defined on $(a, b) \times (c, d)$ which gives the brightness at each point. A discretized image consists of "pixels" obtained by dividing $(a, b) \times (c, d)$ in $N$ by $M$ subsquares, with a brightness value given in the center of each subsquare. The two dimensional version of the discrete Cosine transform (`dct2` and `idct2` in Matlab) is frequently used in image processing.

**(a)** (import an image into matlab). Take an image (it should be sharp and have strong contrast) in some standard image formates (GIF, BMP, TIFF, etc). However, you shall not use JPEG format since JPEG images have been modified in a way such that almost surely the higher frequency DCT coefficients were zeroed out. You can for example download an image from web, or scan in an image and then save it either in GIF or TIFF formate (TIFF can store more image information for each pixel).

Select a part of your image which is of size 256 by 256. You can use `xv` in Unix system to cut the image, or select a smaller rectangle later from Matlab (say `im=im(1:256, 1:256)`).

(Type `xv &`. Click the right mouse button to bring up the controls and load your image. Select "Image Info" from the "Windows" menu. While holding the left mouse button drag the mouse from one corner to the diagonally opposite corner while the current size is displayed in the Info window. When you are satisfied with the selected area release the mouse button, then click "Crop". Click the Save button, choose as format "TIFF" and as colors "Greyscale". Select "Compression: None")

Import the image into Matlab as a two dimensional array `im` using the command

```
im = double(imread('filename.tif','tif'))
```

and display your image using the command

```
imshow(im,[])
```

**(b)** (image compression)

The image is denoted as $u_{i,j}$, the discrete cosine transform of this image is denoted as $\hat{u}_{k,\ell}$. The quality of the image can be reasonably retained even if only few terms of discrete cosine coefficients in $\hat{u}_{k,\ell}$ are used to reconstruct the image. This is known as image compression. The rate of the compression is the percentage, $x\%$, of the discrete cosine coefficients used to reconstruct the image. In other words, we can set $(1 - x\%)$ of the less significant amplitude discrete cosine coefficients to zero. (you may use matlab commands: `dcim=dct2(im);`, (suppose the size of dcim is $N \times N$), use `dcim1=reshape(dcim, N*N,1)` to get one dimensional array; `[s, index]=sort(abs(dcim1)); cutoff=(1-x%)*N*N, dcim1(index(1:cutoff))=0`, dcim= reshape(dcim1, N, N)). Find out the largest possible compression such that the compressed image still resembles original image well. (you can also use the Matlab demo, Toolboxes, image processing, discrete cosine transformation to see the effects in the data compression)

**(c)** (recover a "blurred" image)

A blurred image, denoted as $u_b$, can be constructed by replacing the value of each pixel with the average value of all pixels in a 15 by 15 rectangle. This kind of filter can be defined using the matrix:

$$f = \frac{1}{225}\left[\begin{array}{cc} I_{15,15} & 0 \\ 0 & 0 \end{array}\right]_{\text{size of } u}, \qquad I_{15,15} = \left[\begin{array}{ccc} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{array}\right]_{15,15}$$

(In matlab, $I_{15,15}$= `ones(15)`). The "blurred" image is then constructed by convoluting the original image $u$ by the filter $f$.

It's easiest to do it in the discrete cosine domain because convolution becomes simple multiplication:

$$\hat{u}_b = \hat{u}\hat{f}$$

(in matlab, use command: $\hat{u}_b$ =$\hat{u}$.*$\hat{f}$) and $u_b$ is the inverse transformation from $\hat{u}_b$.

Test that you can recover the original image by deconvolution:

$$\hat{u} = \hat{u}_b / \hat{f}$$

(in matlab use command: $\hat{u} = \hat{u}_b$ ./ $\hat{f}$) and $u$ is the inverse transformation of $\hat{u}$.

**(d)** (Wiener filter for noise) We can now add a random noise (i.i.d.) at each pixel of the blurred image $u_b$. The distribution of the noise can be uniform distribution Unif[-c,c] where $c$ is equal to 4% of the maximal pixel value in the blurred picture, or normal distribution with mean zero and standard deviation $\sqrt{c/6}$.

The original picture can be recovered by deconvolution and Wiener filter (also known as optimal filter) as follows:

$$\hat{v} = \frac{\hat{u}_n \hat{f}}{\hat{f}^2 + \alpha^2}$$

(in matlab use command: $\hat{v}$ = ($\hat{u}_n$ .* $\hat{f}$) ./ ($\hat{f}$ .* $\hat{f}$ + $\alpha^2$)) where $\hat{u}_n$ is the transformed blurred and noisy image, $\hat{f}$ is the transformed filter defined in the previous question, $\alpha$ is a free parameter that should be optimized for each case.

Try to recover the original image and report the best possible parameter $\alpha$ you have used. (you can also use `wiener2` in matlab, the Matlab demo, Toolboxes, image processing, noise reduction filtering to see the effects of the noise and filtering)

**(4)** Explore the cross and auto-covariance

Let $f_j$ and $g_j$ be values of two 1-periodic functions at the points $j/N$ for $j \in Z$ (i.e., $f_j = f_{j+N}$, $g_j = g_{j+N}$). Define the cross covariance of $f$ and $g$ as

$$h_j = \frac{1}{N}\sum_{\ell=0}^{N-1} (f_\ell - \text{mean }(f))(g_{\ell+j} - \text{mean }(g)), \quad j \in Z$$

where mean $(f)$ and mean $(g)$ are the means of $f$ and $g$. When $f = g$, the cross covariance is called auto-covariance. This is an important statistical tool in study complicated physical system such as turbulence.

**(a)** Let $\hat{f}_k$, $\hat{g}_k$, $\hat{h}_k$ be the discrete cosine transforms of $f$, $g$ and $h$. Derive a formula for $\hat{h}_k$ in terms of $\hat{f}_k$ and $\hat{g}_k$. ($\hat{h}_k$ is called cross-spectral density)

**(b)** Write a code to compute the cross covariance of two vectors $f$ and $g$. You should first subtract the means form the vectors, then pad them with sufficiently many zeros to avoid "end effects" and use the FFT.

**(c)** (you can skip this part) Import data from some of experiments or numerical simulations you or your group have conducted and use the cross covariance to discuss your physical system.

**(5)** Explore the ability of wavelets of being able to reveal features of a signal in frequency or time domain but on both domains together.

Consider a signal $f(t) = sin(5t)$. Obtain the FFT and the continuous wavelet transform (CWT) (using 'db4' for example) of this signal. Plot the signal as well as its FFT and CWT. Next, change the signal $f(t)$ by making a very small section of it equal to zero. Get again the same three plots and compare them with the first ones. Comment on the differences. What information about the signal alteration can we get from the FFT and the CWT?.

**(6)** Explore image compression using wavelets.

In this exercise we will do image compression using wavelets. Decompose an image using discrete wavelet transform and show the resulting wavelet coefficients. We will assume that the coefficients will be stored as integers. Therefore the next step is to calculate the minimum number of bits necessary to represent the coefficients in each component of the wavelet decomposition. This is easily done by doing $n = 1 + log_2 C_m$, where $C_m$ is the coefficient with the biggest absolute value in the component being considered. You may want to check if you are already achieving some compression at this stage by comparing the total number of bits necessary to represent the image to the original one (which used $n = 512\text{rows} \times 512\text{columns} \times 8\text{bits per pixels} = 2097152$ bits.).

Next you can achieve higher compression by representing some of the coeffients more coarsely (i.e. use less bits, not storing the least significant ones) or even removing some of them. For example, the human visual system is less sensible to quantization noise in the higher frequency components of the wavelet decomposition so you should be able represent them more coarsely without significantly affecting the perceptible quality of the image.

Turn in your results as a table which details how many bits you use for each component of the wavelet decomposition and the total number of bits you needed. Also calculate the PSNR (Peak Signal-to-Noise Ratio) which you can calculate by doing $PSNR = 10log_{10}(255^2/MSE)$, $MSE$ is the mean square error between the original and the final image. Feel free to try different wavelets and different depth of wavelet decomposition.