AMATH 341 / CS 371 Fall 2004: Assignment 4

Instructor: Hans De Sterck Office: MC 5016 e-mail: *hdesterck@uwaterloo.ca*
TA: Jenny Lee Office: MC 5133 e-mail: *j39lee@math.uwaterloo.ca*

Web Site: http://www.math.uwaterloo.ca/~hdesterc/websiteW/courses/amath341.html
Newsgroup: uw.cs.cs371

Due Date: Friday November 19, beginning of class

1. (10 marks)

   (a) Find the Fourier Transform $F(q)$ of

   $$f(t) = \frac{A\ t}{L} \quad \text{for} \quad t \in [-L, L]; \qquad f(t) = 0 \quad \text{elsewhere.}$$

   Sketch $f(t)$ and $F(q)$.

   (b) Find the Fourier Series Expansion of

   $$f_T(t) = \frac{t}{\pi} \quad \text{for} \quad t \in [0, 2\pi] \quad (T = 2\pi).$$

   Give the result both in complex form and in real form.

2. (10 marks)
   Calculate the Discrete Fourier Transform of the following periodic time sequences by hand, both using the direct DFT formula, and the FFT algorithm. For the FFT calculations, organize your work in a butterfly diagram.

   (a)

   $$f[n] = (1, 2, 2, 1) \quad (n = 0, \ldots, 3;\ N = 4)$$

   (b)

   $$f[n] = (1, 2, 3, 2) \quad (n = 0, \ldots, 3;\ N = 4)$$

   Why is the resulting transform real in this case?

3. (10 marks)

   (a) Write a Matlab function that implements the DFT using the direct formula. The first line of your Matlab m-file `slowFT.m` should read
   `function DFT=slowFT(f,N)`.
   Here f is a complex vector of length N that contains the input time series, and DFT is the resulting complex DFT.

(b) Write a Matlab function that implements the DFT using the FFT algorithm. You can base your implementation on the pseudocode given in the course notes. The first line of your Matlab m-file `fastFT.m` should read
`function DFT=fastFT(f,N)`.
Again, f is a complex vector of length $N = 2^m$ that contains the input time series, and DFT is the resulting complex DFT.

(c) Compare the performance of your slowFT and fastFT implementations. Generate random time series $f = rand(N, 1)$ with $N = 2^k$, for suitable values of k. For each of the methods, see how large you can make k and still get an acceptable turnaround time. Tabulate execution times (you can use the `clock` and `etime` commands) and verify that the complexities are approximately $O(N^2)$ and $O(N \log(N))$ as theoretically expected. (Note: execution times will depend on machine load, memory structure and availability, etc., so don't expect that the theoretical complexity predictions will hold exactly in practice! Performance optimization of your code is optional. For example, assigning the complex roots of unity W_N takes a lot of time in the algorithm, if implemented naively. A lot of time can be saved by calculating these values once and saving the results in a table. Vectorization is also essential in order to get good performance.)

Place your versions of slowFT.m and fastFT.m in a single directory (nothing else in this directory) with the name `your_email _your_student_id`. Then zip up this directory (using zip -r `your_email_your_student_id your_email_your_student_id` if on a unix machine). Mail the file `your_email_your_student_id.zip` to j39lee@math.uwaterloo.ca by 1:30 pm on the due date. Your code will be subject to black box testing.

4. (5 marks)
Prove Parseval's theorem for the DFT,

$$\frac{1}{N}\sum_{n=0}^{N-1} f[n]^2 = \sum_{k=0}^{N-1} |F[k]|^2.$$

(You can assume that $f[n]$ is a real time sequence).

5. (15 marks) Image Compression.

In this problem, we study the compression of a colour image. A colour image `X` in Matlab is represented by an indexed 2D array, according to the RGB standard. More precisely,

```
X(:,:,1) = red component of the image X
X(:,:,2) = green component of the image X
X(:,:,3) = blue component of the image X
```

The compression is to be applied to 8×8 pixel subblocks for each of the components separately, and compression is obtained by dropping small Fourier coefficients on these subblocks.

Copying the files `myImage.m` and `image1.jpg` from the course homepage and executing `myImage.m` in Matlab will produce a colour image stored in `X`.

a) **Compression**
Create a Matlab function, `compress.m`, that has the following prototype:

```
[Y, drop] = compress(X, tol)
```

It takes as inputs the original image, `X`, and the drop tolerance parameter, `tol`, and outputs a compressed image `Y`. It also returns the drop ratio, `drop`, which is defined to be:

$$\text{drop ratio} = \frac{\text{total number of Fourier coefficients dropped}}{\text{total number of pixels in the image}}.$$

'Total number' here means that the numbers from all 3 colour components are added together. The drop ratio is a number between 0 and 1. If drop ratio = 0, then no Fourier coefficient is dropped; if drop ratio = 1, then all Fourier coefficients are dropped.

Specifically, your Matlab function should:

- Compute the 2D Fourier coefficients (use `fft2(single(Xblock))`) for every 8×8 subblock for each colour.
- For each subblock, set those Fourier coefficients having modulus less than `tol` to 0.
- Record the number of coefficients dropped.
- Reconstruct the compressed 8×8 image array by using the inverse 2D Fourier transform (`ifft2`). **Note**: the reconstructed image array must be set to the real part of the inverse transform.
- After all the 8×8 subblocks for all the colour components have been processed, return the entire compressed image as `Y` and the drop ratio as `drop`.

b) **Compression Levels**

Write a Matlab script that will:

- Specify four experimentally determined values of `tol` that result in drop ratios of 0, 0.4, 0.85, and 0.95 (approximately).
- Execute `compress.m` with this set of `tol` values.
- Display the four compressed images using `subplot` with the following Matlab commands for each compressed image `Y`:

```
image(uint8(Y));
axis image;
axis off;
```

The title of each image should be the `tol` value used and the resulting drop ratio.

What to hand in.

Please submit

(a) A listing of `compress.m`.

(b) A listing of the Matlab script.

(c) A figure with 4 plots of the compressed images.

(d) Comment briefly on the compressed images.