4.4 The FFT and MATLAB

4.4.1 The FFT and MATLAB

MATLAB implements the Fourier transform with the following functions: `fft`, `ifft`, `fftshift`, `ifftshift`, `fft2`, `ifft2`. We describe them briefly and then illustrate them with examples.

1. `fft`. This is the one-dimensional Fourier transform. Assuming a signal is saved as an array in the variable \( X \), then \( \text{fft}(X) \) returns the Fourier transform of \( X \) as a vector of the same size as \( X \). Recall that the values returned by \( \text{fft}(X) \) are frequencies.

2. `ifft`. This is the one-dimensional inverse Fourier transform. Given a vector or frequencies \( Y \), \( \text{ifft}(Y) \) will return the signal \( X \) corresponding to these frequencies. In theory, if \( Y = \text{fft}(X) \) then \( \text{ifft}(Y) = X \).

3. `fft2` and `ifft2` are the two dimensional versions of `fft` and `ifft`.

4. `fftshift` rearranges the output of `fft`, `fft2` so the zero-frequency component of the array is at the center. This can be useful to visualize the result of a Fourier transform.

5. `ifftshift` is the inverse of `fftshift`.

We give more explanations about how `fft` works. Using it is not completely straightforward.

- As we saw above, the DFT is evaluated for the fundamental frequency of a signal and its multiples. Let us see what this really is.

- Recall that the fundamental period of a function \( f \) is the smallest positive real number \( T_0 \) such that \( f(t + T_0) = f(t) \). For example, if a function is periodic over an interval \([a, b]\) then its period will be \( b - a \) (the length of the interval over which the function repeats itself. If this is the smallest interval over which the function repeats itself, then \( b - a \) is the fundamental period, which we often call the period.

- The frequency of a signal is the number of periods per second. So, if \( T \) is the period of a signal, then its frequency \( f \) is: \( f = \frac{1}{T} \), it is expressed in Hz (Hertz). So, the fundamental frequency \( f_0 \) is \( f_0 = \frac{1}{T_0} \) where \( T_0 \) is the fundamental period.

- Sometimes, when talking about frequency, we also mean the angular frequency: \( \omega \) defined by \( \omega = 2\pi f \) where \( f \) is the frequency. So, the fundamental angular frequency, \( \omega_0 \) is defined to be \( \omega_0 = 2\pi f_0 = \frac{2\pi}{T_0} \). It is expressed in rad/s. In what follows, when we say frequency, we really mean angular frequency.
From the above, it follows that the fundamental frequency of a periodic function over an interval \([a, b]\) is \(\omega = \frac{2\pi}{b - a} \text{rad/s} \) (or \(\frac{1}{b - a} \text{Hz}\)). Hence, the multiples, denoted \(\omega_n\), are \(\omega_n = \frac{2\pi n}{b - a}\).

Over an interval \([-L, L]\) as it is the case in the examples below, we see that \(\omega_n = \frac{2\pi n}{2L} = \frac{\pi n}{L}\).

The integer values \(n\) are called the wavenumbers.

To compute the DFT, we sample a signal into a finite number of points, we call \(N\). The DFT, in turn, gives the contribution of the first \(N\) multiples of the fundamental frequency to the signal. These fundamental frequencies will also be over an interval centered at 0.

So, to summarize, the wavenumbers \(n\) will run from \(-\frac{N}{2}\) to \(\frac{N}{2} - 1\). However, the output of MATLAB’s \texttt{fft}\ correspond to wavenumbers ordered as \(\{0, 1, 2, ..., \frac{N}{2}, -\frac{N}{2} + 1, ..., -2, -1\}\). This should explain line 8 of the next two examples.

For example, in the next two examples, with \texttt{numpt} = \(2^9 = 512\), \(L = 30\), then \(x\) (the time) will run from \(-30\) to \(30\), the wavenumber \(n\) will run from \(-255\) to \(256\) hence the frequencies \(\frac{2\pi n}{60} = \frac{\pi n}{30}\) will run from \(\frac{(-255) (\pi)}{30} = -26.704\) to \(\frac{(256) (\pi)}{30} = 26.808\).

For example, in the next two examples, with \texttt{numpt} = \(2^9 = 512\), \(L = 5\), then \(x\) (the time) will run from \(-5\) to \(5\), the wavenumber \(n\) will run from \(-256\) to \(255\) hence the frequencies \(\frac{2\pi n}{10} = \frac{\pi n}{5}\) will run from \(\frac{(-255) (\pi)}{5} = -160.22\) to \(\frac{(256) (\pi)}{5} = 160.85\).

A last remark. As noted above, the FFT assumes that the function is periodic over the interval of study. If it is not, the FFT will contain an error at the extremities of the interval of study. This can be seen when starting with a signal, using \texttt{fft}\ on it, then recovering it with \texttt{i fft}.

### 4.4.2 Examples with the One Dimensional FFT

**Example 4.4.1** This first example simply illustrates how the Fourier transform find the various frequencies which make up a signal see figure 4.5.

1. clear all;
2. \% Define the time domain
3. `numpt=2^8;` % Number of Fourier modes (always a power of 2)
4. `L=30;` % time slot to transform, of the form \([-L,L]\]
5. `x2=linspace(-L,L,numpt+1);` % Discretize the time domain. Break the interval \([-L,L]\) into numpt+1 points (+1 in order to include 0).
6. `x=x2(1:numpt);` % time discretization, we want a power of 2 number of points
7. % Define the frequency domain. Remember, we are plotting multiples of the fundamental frequency, that is \(2\pi n/(2L)\), since there are \(N\) points, the wavenumber, \(n\), will run from \(-numpt/2+1\) to numpt/2. Finally, the output of MATLAB's FFT corresponds with wavenumbers ordered as \(\{0,1,\ldots,n/2,-n/2+1,\ldots,-2,1\}\)
8. `k=(2*pi/(2*L))*[0:numpt/2 -numpt/2+1:-1];`
9. % Define the signal
10. `% signal=2*sin(4*x);` % the clean signal. You can change it
11. `% signal=2*sin(4*x)+4*sin(5*x);` % the clean signal. You can change it
12. `% signal=2*sin(4*x)+4*cos(5*x)+3*sin(12*x);` % the clean signal. You can change it
13. % Use fft
14. `utn=fft(signal);` %FFT the noisy signal
15. % Plot the clean signal, the noise and the noisy signal
16. `figure(1),subplot(2,1,1),plot(x,signal,'b'),xlabel('time'),ylabel('signal'),hold on
17. xlabel('time')
18. ylabel('signal')
19. `figure(1),subplot(2,1,2),plot(k,abs(utn)/max(abs(utn)),'r')
20. xlabel('frequency')
21. ylabel('normalized signal')
22. hold off

We now make some remarks about the code.

- Line 3 defines the number of points, it has to be a power of 2.
- Line 4 defines the time slot to transform. In theory, the Fourier transform is valid on the entire real line \((-\infty, \infty)\). In practice, since we are discretizing the time interval, it has to be a bounded interval, that is of the form \([-L,L]\).
4.4. THE FFT AND MATLAB

Figure 4.9: A signal in both the time and frequency domains

- Line 5 discretizes the interval \([-L, L]\). It creates numpt + 1 points (0 is included).

- Line 6 keeps the first numpt points (a power of 2).

- Line 8, we have to rescale the frequency domain by a factor of \(2\pi\) so the frequency numbers FFT returns match the ones in our signal. Recall that \(\sin 2\pi f\) has a period of \(f\).

- Lines 10-12 define various signals to try. You can also make up your own. Only one line should be uncommented.

- Line 14 computes the FFT of the signal.

- Lines 15-22 plot the signal and its FFT.
  - Line 16 does several things. First, the \(\text{subplot}\) command is used to arrange several plots into rows and columns. Then the \(\text{plot}\) command does the actual plotting.
  - Lines 17 and 18 put the labels on the x and y axes.
  - Line 19 is similar to 16. Note that we rescaled the frequency so that the maximum value is 1.

**Example 4.4.2** This next example defines a signal, adds noise to it, takes the Fourier transform of the noisy signal, removes the frequencies with little significance, recovers the signals from these altered frequencies and compares the two signals see figure 4.10 and 4.11.
1. clear all
2. % Define the time domain
3. numpt=2^8; % Number of Fourier modes
4. L=30; % interval is [-L,L]
5. x2=linspace(-L,L,numpt+1); % Discretize the time domain. Break the interval [-L,L] into numpt+1 points (+1 in order to include 0).
6. x=x2(1:numpt); % time discretization, we want a power of 2 number of points
7. % Define the frequency domain. Remember, we are plotting multiples of the fundamental frequency, that is 2*pi*n/(2*L), since there are N points, the wavenumber, n, will run from -numpt/2+1 to numpt/2. Finally, the output of MATLAB’s FFT corresponds with wavenumbers ordered as {0,1,...,n/2,-n/2+1,...,-1}
8. k=(2*pi/(2*L))*[0:numpt/2 -numpt/2+1:-1];
9. % Define the signal
10. % signal=2*sin(4*x); % the clean signal. You can change it
11. % signal=2*sin(4*x)+4*sin(5*x); % the clean signal. You can change it
12. signal=2*sin(4*x)+4*cos(5*x)+3*sin(10*x); % the clean signal. You can change it
13. % Define the noise
14. a=-1; % lower bound of the noise value
15. b=1; % upper bound of the noise value
16. noise=a + (b-a).*rand(1,numpt); % define the noise
17. noisysig=signal+noise; %noisy signal
18. % FFT the noisy signal, remove frequencies which do not pay a big role,
19. % then recover the signal
20. utn=fft(noisysig); %FFT the noisy signal
21. ytn=(abs(utn)>2).*utn; % Keep frequencies with large contribution
22. un=ifft(ytn); % Recover the signal without frequencies without low contributions
23. % Plot the clean signal, the noise and the noisy signal
4.4. THE FFT AND MATLAB

24. \%figure('time', 'signal or noise') \%forces each plot in a different window
25. hold on
26. figure(1), subplot(2,1,1), plot(x,signal,'b',x,noisysig,'g'), hold on \% signal and noisy signal
27. xlabel('time')
28. ylabel('signal')
29. figure(1), subplot(2,1,2), plot(x,noise,'r'), axis([min(x) max(x) min(min(noisysig),min(signal)) max(max(noisysig),max(signal))]), hold on
30. xlabel('time')
31. ylabel('noise')
32. figure(2), subplot(2,1,1), plot(fftsignalk, abs(fftsignalutn)/max(abs(fftsignalutn)),'r'), hold on
33. xlabel('frequency')
34. ylabel('normalized signal')
35. figure(2), subplot(2,1,2), plot(x,signal,'b',x,un,'r'), hold on
36. xlabel('time')
37. ylabel('recovered signal')
38. hold off

We now make some remarks about the code.

- Part of this code is similar to the previous code. I will only comment on what is different.
- Line 14-17 define the noisy signal by creating noise added to the clean signal.
- Lines 20-22 attempt to remove the noise as follow:
  - First, they take the Fourier transform to go to the frequency domain (line 20).
  - Next, only frequencies larger than a threshold (2 in our case, but you can experiment by changing this value) are kept (line 21).
  - Finally, the signal is recovered from these altered frequencies (line 22).
- Lines 25-31 plot the clean signal, the added noise and the resulting noisy signal.
Chapter 4. Time-Frequency Analysis: Fourier Transforms and Wavelets

Figure 4.10: A clean signal, some noise and a noisy signal

- Lines 32-38 plot the FFT of the noisy signal so we can see that the frequencies reported are indeed those of the signal. Then, we plot the original signal together with the recovered signal so we can see how different they are.

4.4.3 The Two Dimensional FFT

The one-dimensional DFT and FFT can be generalized to higher dimensions. We will focus on dimension two here, and apply the FFT to images. In this section, we just do a very brief overview of the main features of the two-dimensional DFT and FFT with MATLAB. We could spend a whole semester studying the various techniques which can be developed with the FFT to clean images, enhance images, and many other activities on images.

Gray images are saved as a two-dimensional array, that is a matrix. If the image has \( m \times n \) pixels, then the matrix will be \( m \times n \). Each entry in the matrix corresponds to a pixel. The value in that entry is the color (gray) intensity. For 8-bit images, it is an integer between 0 and 255.

Color images are saved as a three-dimensional array. The third dimension indicates how many base colors are used. For example, an \( m \times n \) image saved in the RGB color scheme will be saved as an \( m \times n \times 3 \) matrix which is in fact \( 3m \times n \) matrices, one for each color.

An image is also a signal, but it is a 2-dimensional signal. Like a one-dimensional signal, it can be represented as a sum of sine and cosine waves. The difference here is that we have sine and cosine waves in each dimension. Assuming we have an image \( f(x, y) \) of size \( M \times N \), it can be represented in the
4.4. THE FFT AND MATLAB

Figure 4.11: A noisy signal in the time and frequency domain as well as the same signal cleaned using the FFT

frequency domain by its two-dimensional DFT which is

\[ Df(m,n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} f(k,l) e^{-i2\pi \left( \frac{mk}{M} + \frac{nl}{N} \right)} \] (4.24)

You will note that the exponential is in fact a product of two exponentials, \( e^{-i2\pi \frac{mk}{M}} e^{-i2\pi \frac{nl}{N}} \), one sine and cosine wave for each dimension. As before, \( Df(m,n) \) represents the contribution the sine and cosine waves of frequency \( e^{-i2\pi \left( \frac{mk}{M} + \frac{nl}{N} \right)} \) make to the image. Since for images frequency represents color, we can determine which color (or combination of colors) each pixel has.

The inverse Fourier transform is

\[ f(k,l) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} Df(m,n) e^{i2\pi \left( \frac{mk}{M} + \frac{nl}{N} \right)} \]

It allows to recover an image from the frequencies which make it.

4.4.4 The Two Dimensional FFT and MATLAB

Many image manipulations happen in the frequency domain, so the outline to perform these manipulations is as follows:

1. Start with an original image (a signal in the time or spatial domain).
2. Take its Fourier transform, \texttt{fft2}, we are now in the frequency domain.

3. Manipulate the image in the frequency domain. This is often referred to as 'apply a filter'. There are filters for pretty much everything one wants to do such as enhancement, blurring, removing noise, detect edges, ...

4. Take the inverse Fourier transform, \texttt{ifft2}, of the modified image in the previous step. We now have our modified image back in spatial domain.

In the following example, we will:

1. load an image,

2. add noise to it,

3. take the Fourier transform of the noisy image

4. only keep the frequencies with high contribution, thus eliminating most of the noise.

5. take the inverse Fourier transform of the filtered image, this will give us a cleaned image.

We will discuss each section of the code. In the process, we will learn the following MATLAB functions (we assume our images are $m \times n$ pixels):

1. **imread.** Reads an image from its file. The format is \texttt{img=imread(filename)}; where \texttt{filename} is a string containing the filename of the image, \texttt{img} is the matrix containing the image. Note that if the image is grayscale, \texttt{img} will be an $m \times n$ matrix, if the image is a color image (RGB), \texttt{img} will be an $m \times n \times 3$ matrix. Both matrices will have integer entries. We can convert it to a grayscale matrix with the next command.

2. **rgb2gray.** Converts a color image to a grayscale image. The format is \texttt{newimg=rgb2gray(img)}; where \texttt{img} is a color image to convert and \texttt{newimg} is the converted image, an $m \times n$ matrix with integer entries.

3. **double.** Converts integers to doubles (real numbers). The format is \texttt{mat2=double(mat1)}; where \texttt{mat1} is a matrix with integer entries, \texttt{mat2} is a matrix with real entries. Real entries are needed for the FFT.

4. **imshow.** Displays an image. The format is \texttt{imshow(img,[ ])}; where \texttt{img} is a matrix containing the image.

5. **fft2.** Computes the 2D Fourier transform. The format is \texttt{fimg=fft2(img)} where \texttt{img} is a matrix containing the image and \texttt{fimg} is a matrix containing the Fourier transform.

6. **fftshift.** Shift the zero-frequency component to the center of the spectrum. This makes visualizing the Fourier Transform easier. The format is \texttt{img1=fftshift(img)} where \texttt{img} is a matrix and \texttt{img1} is the shifted matrix.
7. \texttt{ifft2}. Computes the inverse of the 2D Fourier transform. The format is \texttt{img=ifft2(fimg)} where \texttt{img} is a matrix containing the image and \texttt{fimg} is a matrix containing the Fourier transform.

\textbf{Example 4.4.3} Removing noise from an image.

1. \% Load the image
   \[ A=imread('head.png'); \]

2. \% Convert to grayscale
   \[ Abw=rgb2gray(A); \]

3. \% Convert to double
   \[ Abw=double(Abw); \]

4. \% Display the original image (see figure 4.12)
   \[ figure(1),imshow(Abw,[]); \]

5. \% Adding noise
   \% Define the noise
   \( a=0; \% lower bound of the noise value \)
   \( b=100; \% upper bound of the noise value \)
   \( noise=a+(b-a).*rand(nx,ny); \% define the noise \)
   \( Abwn=Abw+double(noise); \% add noise to the image \)

6. \% Display the image with noise
   \[ figure(2),imshow(Abwn,[]); \]

7. \% Take the Fourier Transform
   \[ fftA=fft2(Abwn); \]

8. \% Display the Fourier transform (see figure 4.13)
   \[ figure(3),imshow(abs(fftA),[]); \]

9. \% Try again. In the FFT, high peaks can be so high that they hide the details. We can reduce the scale with the log function (see figure 4.14).
   \[ figure(4),imshow(log(1+abs(fftA)),[]); \]

10. \% Try again. We use \texttt{fftshift} to put the coefficients with 0 frequency at the center (see figure 4.15).
    \[ figure(5),imshow(log(1+abs(fftshift(fftA))),[]); \]

11. \% Remove the noise with a high pass filter
    \[ Threshold=median(2*median(abs(fftA))); \% define what to keep \]
    \[ cfftA=(abs(fftA)>Threshold).*fftA; \% keep frequencies with contribution > Threshold \]

12. \% Recover the cleaned image using the inverse Fourier transform
    \[ ifftA=ifft2(cfftA); \% recover the 'cleaned' image \]
Figure 4.12: Original Image
Figure 4.13: Fourier Transform
Figure 4.14: Better FFT
Figure 4.15: Even Better FFT
13. % Display the cleaned image (see figure)
   figure(6),imshow(fftA, []);

You will note that we did not recover the original image. Most of the noise is gone, but the image is also blurred.

4.4.5 Exercises

1. Experiment with example 4.4.1 by changing the signal as well as the parameters nump and L. Comment on what is happening.

2. Experiment with example 4.4.2 by changing the signal as well as the parameters nump and L and also the parameter related to the noise. Comment on what is happening.
3. In example 4.4.2 we started with a clean signal (call it $S_1$), we added noise to it (call the noisy signal $S_2$), then we took the Fourier transform (call the signal in the frequency domain $S_4$), we then removed the frequencies which did not participate a lot in the signal (call the filtered signal $S_5$), finally we recovered the cleaned signal by taking the inverse Fourier transform (call the cleaned signal $S_6$). Could we also recover the noise (without using $S_1$)? How? Write the MATLAB code to implement it (this is just a slight modification of the code you already have).

4. Still in example 4.4.2. Describe another method for recovering the noise (which also does not involve $S_1$). Write the MATLAB code to implement it (this is just a slight modification of the code you already have).

5. Experiment with the 2D Fourier transform. Create your own images, add noise to them, remove the noise with the Fourier transform.

6. There are additional image processing tools in MATLAB. Research the MATLAB functions `fspecial` and `imfilter` and experiment with them.

7. Is this material relevant with the SETI (Search for ExtraTerrestrial Intelligence) project? How?