How to use the MATLAB FFT2-routines

Harald E. Krogstad, NTNU, 2004

This short note describes how the MATLAB functions for two-dimensional Fast Fourier transforms may be used for filtering and signal processing. The MATLAB reference is rather sketchy and does not provide much help for the inexperienced user. We shall, however, assume that the reader knows the elementary properties of the Fourier transform.

1 The 1D FFT

Let us first of all recall that the *Fast Fourier Transform* (FFT) is not a new and strange Fourier transform, but simply an effective numerical algorithm for carrying out *discrete Fourier transforms (DFTs)*. For a vector $X = \{x_n\}_{n=1}^N$, the DFT $Y = \{y_r\}_{r=1}^N$ is defined as

$$y_r = \sum_{n=1}^{N} x_n e^{-\frac{2\pi i(n-1)(r-1)}{N}}. \tag{1}$$

This is MATLAB's definition. Other definitions are possible, so check your other sources for a +-sign in the exponent, or a factor N^{-1} or $N^{-1/2}$ in front of the sum! As long as we are consistent, all definitions are equally good.

Note that Y in general will be *complex* even if X is *real*. However, y_0 and $y_{\frac{N}{2}+1}$ will always be real if X is real (note that $y_{\frac{N}{2}+1}$ exists only when N is even). It is easy to show that the inverse DFT (bringing back X from Y) is simply

$$x_n = \frac{1}{N} \sum_{r=1}^{N} y_r e^{\frac{2\pi i(n-1)(r-1)}{N}}. \tag{2}$$

The sign in the exponent has now changed, and we need a factor N^{-1} in front of the sum. In MATLAB, the two operations are computed simply by

$$\begin{aligned} Y &= \mathbf{fft}\,(X)\,, \\ X &= \mathbf{ifft}(Y). \end{aligned} \tag{3}$$

Try this on some vectors!

Note that the result of $\mathbf{ifft}\,(\mathbf{fft}\,(X))$ will be defined complex by MATLAB, even if X is real. However, the imaginary parts should be of the order of the machine accuracy.

2 The 2D FFT

Consider a matrix

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \cdots & a_{MN} \end{bmatrix}. \tag{4}$$

The MATLAB 2-dimensional Fourier transform of A is defined as the *complex matrix*

$$Y = \mathbf{fft2}(A) = \begin{bmatrix} y_{11} & \cdots & y_{1N} \\ \vdots & \ddots & \vdots \\ y_{M1} & \cdots & y_{MN} \end{bmatrix} \tag{5}$$

where

$$y_{rs} = \sum_{m=1}^{M} \sum_{n=1}^{N} a_{mn} e^{-\frac{2\pi i(m-1)(r-1)}{M} - \frac{2\pi i(n-1)(s-1)}{N}}. \tag{6}$$

The MATLAB index convention complicates the expression somewhat. For the discrete Fourier transform it is more convenient to let arrays run from 0 to $M-1$.

In order to reproduce A from Y by an inverse Fourier transform, it is now necessary to compute

$$a_{mn} = \frac{1}{MN} \sum_{r=1}^{M} \sum_{s=1}^{N} y_{rs} e^{+\frac{2\pi i(m-1)(r-1)}{M} + \frac{2\pi i(n-1)(s-1)}{N}}, \tag{7}$$

and this is carried out by

$$A = \mathbf{ifft2}\,(Y) \tag{8}$$

This is about all the reference guide tells you.

What if we want to *do* something with the Fourier transform, for example some kind of filtering? A typical application would be that A is a digitized image, and $\{a_{mn}\}$ are the *pixel* values. The pixel values are obtained from the image, $I(x, y)$, by a *sampling*,

$$\begin{aligned} a_{mn} &= I\left(\Delta x(m-1), \Delta y(n-1)\right), \\ m &= 1, \cdots, M,\ n = 1, \cdots, N. \end{aligned} \tag{9}$$

The constants Δx and Δy are called the respective *sampling intervals*.

In general, a *linear filter operation* of a 2D function f on $\mathbb{R}^2$ is a convolution $f \to T * f$ defined as

$$T * f\,(\mathbf{x}) = \int_{R^2} f(\mathbf{x} - \mathbf{y}) T\,(\mathbf{y})\, d\mathbf{y} \tag{10}$$

By defining the continuous 2D Fourier transform as

$$\mathcal{F}(f)(\mathbf{k}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\mathbf{x}) e^{-i\mathbf{kx}} d\mathbf{x}, \tag{11}$$

the filtering of the image, $I \to h * I$ amounts to a multiplication in the Fourier domain:

$$\mathcal{F}(T * I)(\mathbf{k}) = \mathcal{F}(T)(\mathbf{k}) \times \mathcal{F}(I)(\mathbf{k}). \tag{12}$$

The result is finally obtained by an inverse Fourier transform,

$$T * I(\mathbf{x}) = \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathcal{F}(T * I)(\mathbf{k}) e^{i\mathbf{kx}} d\mathbf{k}. \tag{13}$$

The filter may be a *low pass filter*, say,

$$\mathcal{F}(T)(\mathbf{k}) = \begin{cases} 1, & |\mathbf{k}| \le k_0 \\ 0, & |\mathbf{k}| > k_0 \end{cases}, \tag{14}$$

a *high pass filter*,

$$\mathcal{F}(T)(\mathbf{k}) = \begin{cases} 0, & |\mathbf{k}| \le k_0 \\ 1, & |\mathbf{k}| > k_0 \end{cases}, \tag{15}$$

or defined by some other function of $\mathbf{k}$ (For many commonly used filters, T is actually what is known as a *generalized function).*

The purpose of the rest of this note is to teach you how to carry out this type of operation in MATLAB.

2.1 Real data

Most images or other two-dimensional data are real, that is, $I(\mathbf{x})$ consists of real numbers. However, the Fourier transform will generally be complex, although it then follows from the definition that

$$\mathcal{F}(I)(-\mathbf{k}) = \overline{\mathcal{F}(I)(\mathbf{k})}. \tag{16}$$

This is an important equation:

If we know that the image is real, it is sufficient to know $\mathcal{F}(I)$ for only "half" of the $\mathbf{k}$-values.

For example, it is sufficient to know $\mathcal{F}(I)(\mathbf{k})$ where $\mathbf{k} = (k_x, k_y)$ for $k_x \ge 0$.

The second important observation is that when the filter is real, that is, $T(\mathbf{x})$ is a real function, then *also*

$$T * I(\mathbf{x}) \tag{17}$$

should be real for all $\mathbf{x}$. This is a very important check on what we have been doing is correct.

Due to numerical rounding errors, the answer in MATLAB will have very small imaginary parts (compared to the real parts). As long as the imaginary parts are of the order of the machine accuracy (typically around 10^{-15} if the data are of order one), this is nothing to worry about.

2.2 How to locate $\mathcal{F}(I)(\mathbf{k})$ in $FFT2(A)$

Let us first look at the connection between $Y = \mathbf{fft}(A)$ and $\mathcal{F}(I)$. Unfortunately, this is rather tricky:

$$\begin{aligned} y_{rs} &= \sum_{m=1}^{M}\sum_{n=1}^{N} a_{mn} e^{-\frac{2\pi i(m-1)(r-1)}{M} - \frac{2\pi i(n-1)(s-1)}{N}} \\ &= \frac{1}{\Delta x \Delta y} \sum_{m=1}^{M}\sum_{n=1}^{N} I\left(\Delta x(m-1), \Delta y(n-1)\right) e^{-\frac{2\pi i \Delta x(m-1)(r-1)}{\Delta x M} - \frac{2\pi i \Delta y(n-1)(s-1)}{\Delta y N}} \Delta x \Delta y \\ &\approx \frac{1}{\Delta x \Delta y} \int_0^{(M-1)\Delta x} \int_0^{(N-1)\Delta y} I(x,y) e^{-\frac{2\pi i}{\Delta x M}(r-1)x - \frac{2\pi i}{\Delta y N}(s-1)y} dxdy \\ &= \mathcal{F}(I)\left(\frac{2\pi i}{\Delta x M}(r-1), \frac{2\pi i}{\Delta y N}(s-1)\right). \end{aligned} \tag{18}$$

For the approximation to be good, the sampling intervals Δx and Δy have to be sufficiently small. Moreover, the integral will only cover the region

$$\mathbf{x} \in [0, \Delta x(M-1)] \times [0, \Delta y(N-1)],$$

and hence we prefer that M and N are large.

The matrix Y gives us an approximation to the Fourier transform on a *grid* of **k**-values, namely

$$\begin{aligned} k_x &= \frac{2\pi i}{\Delta x M}(r-1),\ r = 1, \cdots, M, \\ k_y &= \frac{2\pi i}{\Delta x N}(s-1),\ s = 1, \cdots, N. \end{aligned} \tag{19}$$

There is only one little complication. If we look at the definition of Y, and assume that $r < M/2$ and $s < N/2$, then

$$\begin{aligned} y_{(M-r+1)(N-s+1)} &= \sum_{m=1}^{M}\sum_{n=1}^{N} a_{mn} e^{-\frac{2\pi i(m-1)((M-r+1)-1)}{M} - \frac{2\pi i(n-1)((N-s+1)-1)}{N}} \\ &= \sum_{m=1}^{M}\sum_{n=1}^{N} a_{mn} e^{-\frac{2\pi i(m-1)(M-r)}{M} - \frac{2\pi i(n-1)(N-s)}{N}} \\ &= \sum_{m=1}^{M}\sum_{n=1}^{N} a_{mn} e^{-\frac{2\pi i(m-1)(-r)}{M} - \frac{2\pi i(n-1)(-s)}{N}} = y_{-r,-s}, \end{aligned} \tag{20}$$

since $\exp(2\pi i m) = 1$ for all integers m. This means that for indices larger than about $M/2$ and $N/2$, we do not really obtain the Fourier transform for wavenumbers according to Eqn. 19, but instead *the Fourier transform for negative wavenumbers.*

To sum up: *The 2D FFT computed from MATLAB, contains an approximation to the Fourier transform on a discrete grid ranging from about* $-\frac{\pi}{\Delta x}$ *to* $\frac{\pi}{\Delta x}$ *in steps of* $\Delta k_x = \frac{2\pi}{M\Delta x}$*, and similarly for* k_y*.*

The following small piece of MATLAB code computes these wavenumbers in the correct locations and in the standard matrix format created by **meshgrid**:

```
kx1 = mod( 1/2 + (0:(M-1))/M , 1 ) - 1/2;

kx = kx1*(2*pi/deltax);

ky1 = mod( 1/2 + (0:(N-1))/N , 1 ) - 1/2;

ky = ky1*(2*pi/deltay);

[KX,KY] = meshgrid(kx,ky);
```

Note that the (positive) Nyquist wavenumber occurs for $M/2+1$ and $N/2+1$ whenever M or N are even. If you are building a Fourier transform of a real function, you must ensure that the Fourier transform for the indices $m = M/2+1$ and $n = N/2+1$ are REAL. In practice, it is safest simply to set these values to 0. The Nyquist wavenumbers are not on the `KX,KY`-grid when M and N are odd. The Fourier transform for $(-k_x, -k_y)$ should also always be the complex conjugate of the transform for (k_x, k_y).

With these arrays, it is quite simple to code a lowpass filter, say

$$T(\mathbf{k}) = \begin{cases} 1, & |\mathbf{k}| < k_0, \\ 0, & 0. \end{cases} \tag{21}$$

Taking full advantage of the powerful MATLAB syntax,

$$\texttt{T} = (\texttt{KX}.*\texttt{KX} + \texttt{KY}.*\texttt{KY} \; < \; \texttt{k0}\hat{}\texttt{2}) \tag{22}$$

The filtering may then be written as

$$\texttt{Afilt} \; = \; \texttt{ifft2}(\; \texttt{T}.*\texttt{fft2}(\texttt{A})\;) \tag{23}$$

Finally, if we look at the wavenumber locations in `KX` and `KY`, the picture is a bit confusing. For plots of the Fourier transform it is best to have $\mathbf{k} = \mathbf{0}$ in the center. This may be arranged by the routines `fftshift` and `ifftshift` (see the MATLAB documentation).