Unit 7. Frequency Domain Processing

7.1 Frequency Content of One-Dimensional Signals

Grayscale Variation across a Row. Consider the top row {f(0,n)} of an image {f(m,n)} where m = 0. Upon suppressing the index m = 0 we can write this as a 1-dimensional array (discrete function) {f(n)}. Figure 7.1a presents an example of {f(n)}. Figure 7.1b shows the intensity values as gray levels. On the other hand, we could take the first column {f(m,0)} as a 1-dimensional function, or any other single row or column.

Figure 7.1. Row and Column Functions

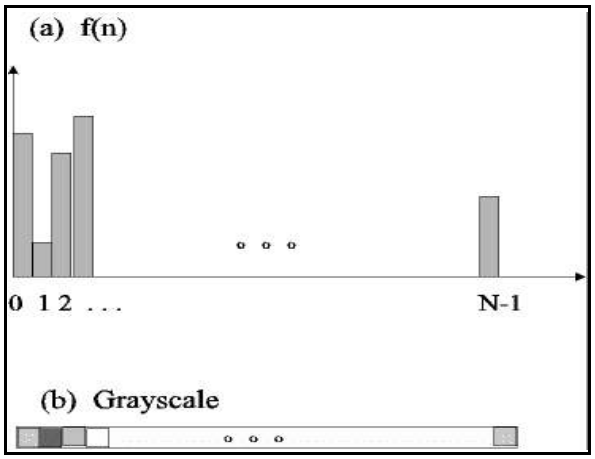
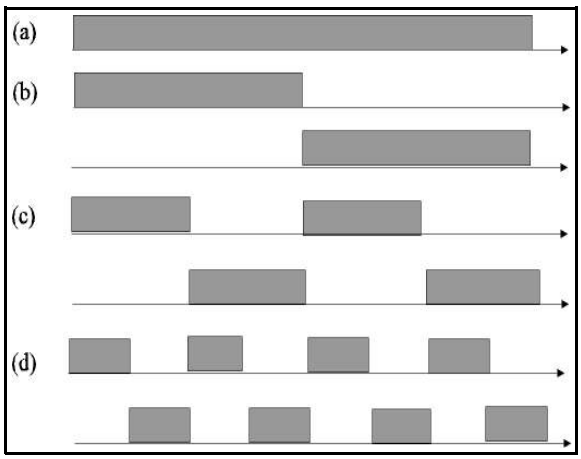


Figure 7.2. 1-Dimensional Unit Changes

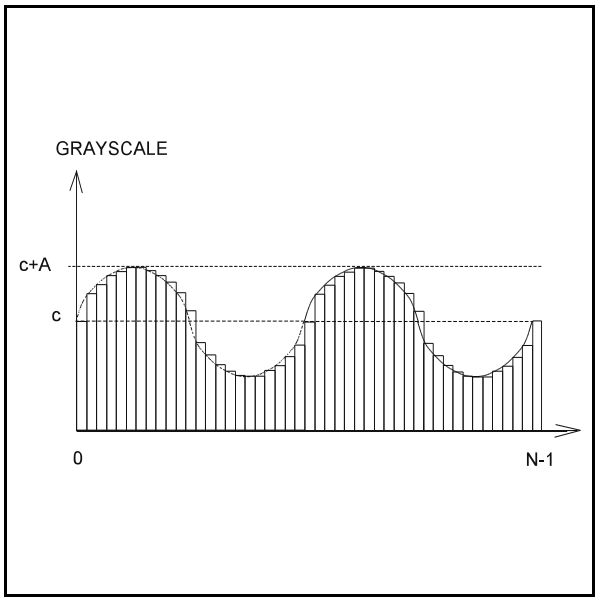


Such 1-dimensional functions may be constant (a single shade of gray across all row or column pixels), may change one time, or may change two or more times. Figure 7.2 presents some possibilities of the variation in grayscale over the domain row of pixels. These are unit functions but can be multiplied by a constant between 0 and 255 for grayscale (this changes the intensity, or amplitude of these functions). Figure 7.2a shows the unit constant over the *period* (domain) of N row pixels. Figure 7.2b shows a single unit change from one gray level to another and Figure 7.2c and 7.2d show more unit changes across one spatial cycle (e.g., row).

In Figure 7.2a there are 0 cycles over the period of N pixels (denoted by $f_0(n)$, a constant function), while in Figure 7.2b, there is a single cycle ($f_1(n)$) over the period (the *fundamental period*) and there are two of these offset by ½ cycle. Figure 7.2c shows two cycles ($f_2(n)$) over the fundamental period, while Figure 7.2d shows 4 complete cycles ($f_4(n)$). Note that the maximum number of these cycles over the period of N pixels is N/2 (it takes at least two pixels per cycle). When the domain is in the time dimension we designate the number of cycles per second by the unit *Herz* (Hz). Similarly, we use Herz to refer to the number of cycles across the fundamental period of the spatial domain.

Linear combinations of the functions $f_0(n)$, $f_1(n)$,...,$f_{N-1}(n)$ may be used to create discrete functions such as that shown in Figure 7.1. Positive and negative coefficients can convert the unit changes into changes of any magnitude, and the summing allows the construction of discrete functions from these *independent* functions (independent in the sense that any one can not be equal to a linear combination of the other ones).

Figure 7.3. Sinusoidal Variation



Sinusoids are the natural, or canonical, cyclical functions. The sinusoids $s(n) = \sin(2\pi n/N)$ and $c(n) = \cos(2\pi n/N)$ go through a single cycle each as n goes through the fundamental period from $n = 0$ to $n = N-1$. The sinusoids $s(n;k) = \sin(2\pi kn/N)$ and $c(n;k) = \cos(2\pi kn/N)$ go through k cycles for any fixed k as n goes from $n = 0$ to $n = N-1$. The parameter k determines the rate of change over the fundamental period N. It is well known that the functions $\{\sin(2\pi kn/N), \cos(2\pi kn/N): k = 0,...,N-1\}$ are *linearly independent* (no single function can be represented as a linear combination of the remaining ones).

An important principle that we use in the next subsection is: *all terms* of the forms

$\cos(2\pi kt/T)$, $\sin(2\pi \ell t/T)\cos(2\pi kt/T)$
$\cos(2\pi \ell t/T)\cos(2\pi kt/T)$, $\sin(2\pi \ell t/T)\sin(2\pi kt/T)$

sum (integrate) to zero (for $\ell \neq k$) over the fundamental period T on which $\cos(2\pi t/T)$ and $\sin(2\pi t/T)$ complete one cycle. Figure 7.3 presents two cycles of a *raised* sinusoidal variation (a constant c is added) with amplitude A in the grayscale across a single row of an image. The discrete values are shown under the continuous function.

The Discrete Fourier Series in a Single Dimension. A *signal* is a function of time or spatial dimensions (or both). Intensity (gray level) in an image is a 2-dimensional signal over space, while along a single row or column it is a 1-dimensional signal. Let $\{f(n)\}$ be a grayscale (intensity) signal, $n = 0,...,N-1$. We want to decompose $\{f(n)\}$ into its frequency components, expressed in natural cyclical independent functions, as the linear combination (where "≈" designates *approximately equal to*)

$$f(n) \approx \sum_{(k=0,N-1)}[a_k\cos(2\pi kn/N) + b_k\sin(2\pi kn/N)], \quad n = 1,...,N-1 \tag{7.1}$$

When $k = 0$, we have the term

$$a_0\cos(2\pi 0n/N) + b_k\sin(2\pi 0n/N) = a_0(1) + b_0(0) = a_0 \tag{7.2a}$$

so that we can write Equation (7.1) as

$$f(n) \approx a_0 + \sum_{(k=1,N-1)} [a_k\cos(2\pi kn/N) + b_k\sin(2\pi kn/N)] \tag{7.2b}$$

We do not know yet whether or not a set of coefficients $\{a_k,b_k: k = 0,...,N-1\}$ exists such that equality of the lefthand and righthand sides of Equation (7.2b) holds for every n in the domain of f(n). We consider now the *mean-square error* between the lefthand and righthand sides of Equation 7.2b to be

$$\epsilon = (1/N)\sum_{(n=0,N-1)}\{f(n) - [a_0 + \sum_{(k=1,N-1)} [a_k\cos(2\pi kn/N) + b_k\sin(2\pi kn/N)]]\}^2 \tag{7.3}$$

It is convenient in the derivation that follows to use the continuous form of the mean square error with t = n and T = N (the fundamental period), where integrals replace summations, to obtain

$$\epsilon = (1/T)\int_0^T \{f(t) - [a_0 + \sum_{(k=1,N-1)} [a_k\cos(2\pi kt/T) + b_k\sin(2\pi kt/T)]]\}^2 dt \tag{7.4}$$

We first solve Equation (7.4) for a_0 by setting the partial derivative with respect to a_0 to zero.

$$\partial\epsilon/\partial a_0 = 0 \tag{7.5}$$

$$2(1/T)\int_0^T \{f(t) - [a_0 + \Sigma_{(k=1,N-1)} [a_k\cos(2\pi kt/T) + b_k\sin(2\pi kt/T)]]\}(-1)dt$$

Therefore

$$(1/T)\int_0^T f(t)dt = \tag{7.6}$$

$$(1/T)\int_0^T a_0 dt + \Sigma_{(k=1,N-1)}\{(1/T)\int_0^T a_k\cos(2\pi kt/T)dt + (1/T)\int_0^T b_k\sin(2\pi kt/T)]dt\}$$

All except the first term in the last line integrate to zero (see the last subsection) so

$$a_0 = (1/T)\int_0^T f(t)dt \tag{7.7}$$

We next solve for a_k, $0 < k \leq N-1$. Setting the appropriate derivative of the mean-square error to 0, we obtain

$$\partial\epsilon/\partial a_k = 0 = \tag{7.8}$$

$$2(1/T)\int_0^T \{f(t) - [a_0 + \Sigma_{(s=1,N-1)} [a_s\cos(2\pi st/T) + b_s\sin(2\pi st/T)]]\}(-\cos(2\pi kt/T))dt$$

Thus

$$(1/T)\int_0^T f(t)\cos(2\pi kt/T)dt = \tag{7.9}$$

$$(1/T)\int_0^T a_0\cos(2\pi kt/T)dt + \Sigma_{(s=1,N-1)}(1/T)\left\{\int_0^T a_s\cos(2\pi st/T)\cos(2\pi kt/T)dt + \int_0^T b_s\sin(2\pi st/T)\cos(2\pi kt/T)dt\right\}$$

The integral with integrand $a_0\cos(2\pi kt/T)$ equals zero. All integrals with the integrand $b_s\sin(2\pi st/T)\cos(2\pi kt/T)$ also evaluate to 0. Of the remaining terms of products of cosines, all except where s = k integrate to zero. We evaluate the integral of the term $\cos^2(2\pi kt/T)$ where s = k via the trigonometric identity

$$\cos^2(2\pi kt/T) = (1/2)[1 - \cos(4\pi kt/T)] \tag{7.10}$$

Thus we have

$$(1/T)\int_0^T f(t)\cos(2\pi kt/T)dt = a_k(1/T)\int_0^T [1/2 - \cos(4\pi kt/T)/2]dt = a_k/2 \tag{7.11}$$

This yields

$$a_k = (2/T)\int_0^T f(t)\cos(2\pi kt/T)dt, \quad k = 1,...,N-1 \tag{7.12a}$$

Similarly, the derivation follows for

$$b_k = (2/T)\int_0^T f(t)\sin(2\pi kt/T)dt, \quad k = 1,...,N-1 \tag{7.12b}$$

The above results can be summed up in discrete form as

$$f(n) \approx a_0 + \sum_{(k=1,N-1)}[a_k\cos(2\pi kn/N) + b_k\sin(2\pi kn/N)], \quad n = 0,1,...,N-1 \tag{7.13a}$$

$$a_0 = (1/N)\sum_{(n=0,N-1)}f(n) \tag{7.13b}$$

$$a_k = (2/N)\sum_{(n=0,N-1)}f(n)\cos(2\pi kn/N), \quad k = 1,...,N-1 \tag{7.13c}$$

$$b_k = (2/N)\sum_{(n=0,N-1)}f(n)\sin(2\pi kn/N), \quad k = 1,...,N-1 \tag{7.13d}$$

The righthand side of Equation (7.13a) is called the *discrete Fourier series* (DFS) for {f(n)}. Note that f(n) is the sum of a constant a_0 (the average value of f(n) over the fundamental period) and a sequence of periodic functions that all complete their kth cycles at n = N. Thus f(n) is a periodic function that repeats itself on [N,2N-1], [2N,3N-1], and also on [-N+1,0], [-2N+1,-N+1], and so forth. We are interested, however,

in f(n) only for n = 0,...,N-1, but we should remember that f(n) exists outside of [0,N-1] and is not zero there.

But why are we interested in expressing f(n) as a sum of sinusoids of different frequencies? The answer is that their coefficients a_k and b_k give the relative importance of each frequency at k Hz (that is, k cycles per fundamental period, k = 0,...,N-1) in f(n). Actually, the two independent functions $\cos(2\pi kn/N)$ and $\sin(2\pi kn/N)$ have the frequency k Hz, so we combine their coefficients to obtain the total magnitude of the kth frequency component by

$$|c_k| = (a_k^2 + b_k^2)^{1/2} \tag{7.14}$$

Recalling that v = voltage = (current amperage)x(ohms resistance) = iR, so that i = v/R, we can think of the signal f(n) as being a voltage across a one ohm resistor. From p = power = (voltage)x(current) we obtain

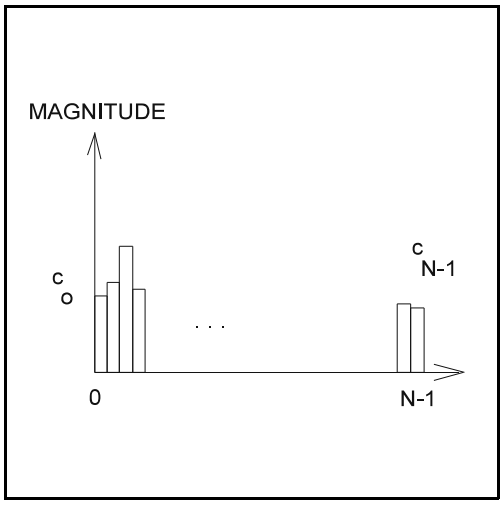
$$p = vi = v(v/R) = v^2 = f(n)^2 \tag{7.15}$$

It can be shown (*Parseval's theorem*) by squaring and integrating that the average power over the fundamental period is (continuous form for f(n) as f(t))

$$\sum_{(k=0,N-1)} c_k^2 = (1/T)\int_0^T f(t)^2 dt \tag{7.16}$$

Note that c_0^2 is the dc (direct current) power component, while c_k^2 is the power in the k-Hz component. For this reason, $\{c_k^2: k = 0,...,N-1\}$ is called the *power spectrum*, while $\{c_k: k = 0,...,N-1\}$ is called the *magnitude spectrum* (or *spectrum*) of f(n). Figure 7.4 shows an example of a magnitude spectrum for an image row f(n).

Figure 7.4. A Magnitude Spectrum of a Row



Frequency Processing via Filtering. Suppose that we want to sharpen a 1-dimensional signal f(n).We can first convert {f(n)} to the magnitude spectrum $\{|c_k|: k = 0, ...,N-1\}$ via Equations (7.13 b,c,d) and (7.14) and then to change the magnitudes $|c_k|$ to $|C_k|$ by making them smaller for low k and making them larger for large k (high pass). The proportions of the new C_k are then taken of the original a_k and b_k to obtain new coefficients A_k and B_k. Finally, these new coefficients are used in Equation (7.13a) to compute new processed values g(n) of the function. Such processing is called *high pass filtering*. But if we reduce (*attenuate*) the coefficients a_k and b_k in magnitude for $k \geq k_0$ for some frequency k_0, but do not change them for $k < k_0$ (or even boost them) then this is a *lowpass filter*. If the f(n) come from a row in an image, then the new function g(n) is a processed image row. The processing of a single row or column at a time, however, does

not use information in the other rows and columns, so we need to process in two dimensions, and especially in neighborhoods of each pixel.

7.2 Frequency Content of Two-Dimensional Signals

The Discrete Fourier Series of an Image. The 2-dimensional forms of Equations (7.13) are

$$f(m,n) \approx a_{0,0} + \sum\sum_{(v=0,N-1;u=0,M-1)}[a_{u,v}\cos(2\pi(um/M + vn/N)) + b_{u,v}\sin(2\pi(um/M + vn/N))] \tag{7.17a}$$

$$a_{0,0} = [1/(MN)] \sum\sum_{(n=0,N-1;m=0,M-1)} f(m,n) \tag{7.17b}$$

$$a_{u,v} = [2/(MN)] \sum\sum_{(n=0,N-1;m=0,M-1)} f(m,n)\cos(2\pi(um/M + vn/N)) \tag{7.17c}$$

$$b_{u,v} = [2/(MN)] \sum\sum_{(n=0,N-1;m=0,M-1)} f(m,n)\sin(2\pi(um/M + vn/N)) \tag{7.17d}$$

The righthand side of Equation (7.17a) is the 2-dimensional *discrete Fourier series* (DFS) for the image {f(m,n)}. The respective spectral magnitude and spectrral power of {f(m,n)} are given by

$$|c_{u,v}| = (a_{u,v}^2 + b_{u,v}^2)^{1/2} \tag{7.18a}$$

$$c_{u,v}^2 = a_{u,v}^2 + b_{u,v}^2 \tag{7.18b}$$

There is a spectrum value $|c_{u,v}|$ for each pair of frequencies (u,v), $0 \le u \le M-1$, $0 \le v \le N-1$. Thus the spectrum is also a 2-dimensional signal over a rectangular region of frequencies and so we sometimes call the spectrum $\{|c_{u,v}|\}$ a *frequency image*. We prefer to use the notation {F(u,v)}, where

$$|F(u,v)| = |c_{u,v}| = (a_{u,v}^2 + b_{u,v}^2)^{1/2} \tag{7.18c}$$

This notation associates the frequency image {F(u,v)} with the spatial image {f(m,n)}.

Given an image {f(m,n)} we can use Equations (7.17b,c,d) and (7.18a) to compute the spectrum. Filtering consists of changing the magnitude (or power) level at the various frequency pairs (u,v) by adjusting |F(u,v)| to obtain |G(u,v)|. We then distribute the new level G(u,v) proportionally over the coefficients $a_{u,v}$ and $b_{u,v}$ to obtain new coeffients $A_{u,v}$ and $B_{u,v}$. These are then used in Equation (7.17a) to compute the new processed image {g(m,n)}. These proportions are

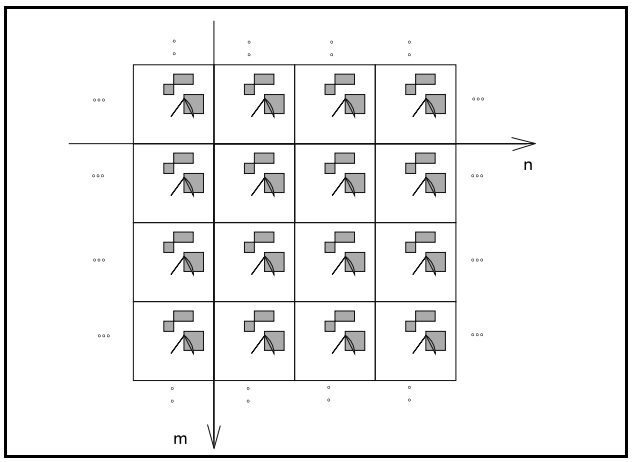
$$A_{u,v} = [|G(u,v)|/|F(u,v)|]a_{u,v}, \quad B_{u,v} = [|G(u,v)|/|F(u,v)|]b_{u,v} \tag{7.19a}$$

where a filter function H(u,v) = |G(u,v)|/|F(u,v)| is determined. We almost always design H(u,v) first according to what kind of filtering we want to do. Thus we obtain

$$A_{u,v} = H(u,v)a_{u,v}, \quad B_{u,v} = H(u,v)b_{u,v} \tag{7.19b}$$

Equation (7.17a) is periodic in both m and n dimensions with respective fundamental periods of M and N, and so it represents an image repeated indefinitely in these dimensions as shown in Figure 7.5. To prevent such periodicity, we use discrete Fourier transform defined in Section 7.3.

Figure 7.5. DFS Periodicity



The Exponential Form of the DFS. It is convenient to use the standard complex exponential in the discrete Fourier series of Equations (7.13) and (7.17). Letting $j = \sqrt{(-1)}$, we put

$$c_k = (a_k - jb_k)/2 \tag{7.20a}$$

$$c_{-k} = (a_k + jb_k)/2 \tag{7.20b}$$

we see that

$$a_0 = c_0, \quad a_k = c_k + c_{-k} \tag{7.21a}$$

$$b_k = j(c_k - c_{-k}) \quad (k \neq 0) \tag{7.21b}$$

Using the Euler-DeMoivre equations

$$e^{\pm jx} = \cos(x) \pm j\sin(x), \quad \cos(x) = [e^{jx} + e^{-jx}]/2, \quad \sin(x) = [e^{jx} - e^{-jx}]/(2j) \tag{7.22}$$

we obtain

$$a_k\cos(2\pi kn/N) + b_k\sin(2\pi kn/N) = c_k e^{j2\pi kn/N} + c_{-k}e^{-j2\pi kn/N} \tag{7.23}$$

The exponential form of the DFS is then

$$f(n) = \sum_{(k=-N+1,N-1)} c_k \exp[j2\pi kn/N] \tag{7.24a}$$

$$c_k = (1/N)\sum_{(n=0,N-1)} f(n)\exp[-j2\pi kn/N] \tag{7.24b}$$

This model has the advantage of compact notation, but also the disadvantages of having imaginary parts to the spectrum $\{c_k\}$ (corresponding to the sine-cosine terms) and worse, the spectrum has indices of negative k as well as positive, so that half of the power at frequency k Hz is shown on the negative axis of the frequency dimension (Hz). Of course there is no such thing as negative frequency (the unpleasantness of the complex model). Nonetheless, the model is useful and in common usage.

The 2-dimensional exponential form of the discrete Fourier series (DFS) of an image $\{f(m,n)\}$ is

$$f(m,n) = \sum\sum_{(u=-M+1,M-1;\ v=-N+1,N-1)} F(u,v)\exp[j2\pi(um/M + vn/N)] \tag{7.25a}$$

where

$$F(u,v) = [1/(MN)]\sum\sum_{(m=-M+1,M-1;\ n=-N+1,N-1)} f(m,n)\exp[-j2\pi(um/M + vn/N)] \tag{7.25b}$$

The magnitude spectrum {|F(u,v)|} is obtained from the magnitude of a complex number, for example, |x + jy| is computed from the real and imaginary parts by

$$|x + jy| = (|x + jy|^2)^{1/2}, \quad |x + jy| = [(x + jy)(x - jy)]^{1/2} = [x^2 - jxy + jxy - j^2y^2]^{1/2} = [x^2 + y^2]^{1/2} \tag{7.26}$$

7.3 The Discrete Fourier and Hartley-Bracewell Transforms

The Discrete Fourier Transform. The use of Equations (7.25a), (7.25b) and (7.26) yields a resulting processed image {g(m,n)} that is periodic and repeats itself over the fundamental periods of M and N (for example, going from N to 2N-1, M to 2M-1, going from -N+1 to 0, -M+1 to 0, and going from 2N to 3N-1, etc.). Figure 7.5 shows the situation. The entire plane is filled up with repeated images. To avoid this infinite repetition, we use the *discrete Fourier transform* (DFT) {F(u,v)}. The DFT is the limit where f(m,n) = 0 for all (m,n) outside of the image, obtained by taking larger and larger blocks in the plane with the image in the center. The frequencies become smaller and smaller until we have continuous u and v in the limit. The coefficients become more numerous until in the end there is one for every continuous pair (u,v). This eliminates periodicity in the plane. The *DFT* and *inverse DFT* (IDFT) *pair* is

$$f(m,n) = \sum\sum_{(u=0,M-1;v=0,N-1)} F(u,v)\exp[j2\pi(um/M + vn/N)] \tag{7.28a}$$

$$F(u,v) = [1/(MN)]\sum\sum_{(m=0,M-1;n=0,N-1)} f(m,n)\exp[-j2\pi(um/M + vn/N)] \tag{7.28b}$$

These are discrete approximations (m = s, n = t) to the continuous Fourier and inverse Fourier transforms

$$f(s,t) = (1/4\pi^2)\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\exp[j2\pi(us/M + vt/N)]dudv \tag{7.29a}$$

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(m,n)\exp[-j2\pi(us/M + vt/N)]dsdt \tag{7.29b}$$

Equations (7.28b and (7.29b) are the DFTs while (7.28a) and (7.29a) are the unique IDFTs. We will use the properties of the continuous Fourier transform in the design of filters. Images {g(m,n)} that have been processed with the DFT do not repeat over extended fundamental periods. This will be important when we perform convolution in the spatial plane with masks derived from the frequency domain filters.

The Hartley-Bracewell Transform. An equivalent formulation that avoids the use of complex numbers is the Hartley transform for the continuous case. It was extended to the discrete case by Bracewell. The transformations for going to the frequency domain from the spatial domain, and going to the spatial domain rom the frequency domain, respectively, are

$$F(u,v) = [1/(MN)]\sum\sum_{(m=0,M-1;n=0,N-1)} f(m,n)\text{cas}[2\pi(um/M + vn/N)] \tag{7.30a}$$

$$f(m,n) = \sum\sum_{(u=0,M-1;v=0,N-1)} F(u,v)\text{cas}[2\pi(um/M + vn/N)] \tag{7.30b}$$

where

$$\text{cas}(\Theta) = \cos(\Theta) + \sin(\Theta) = (\sqrt{2})\cos(\Theta - \pi/4) \tag{7.30c}$$

The advantages of the discrete Hartley-Bracewell transform (DHBT) over the DFT are that there are no complex numbers and that MN real numbers {f(m,n)} are transformed to MN real numbers {F(u,v}. The filtering is done by use of the *even* $H_e(u,v)$ and *odd* $H_o(u,v)$ parts of the filter H(u,v) by

$$G(u,v) = H_e(u,v)F(u,v) + H_o(u,v)F(-u,-v) \tag{7.31}$$

$$H_e(u,v) = (1/2)[H(u,v) + H(-u,-v)], \quad H_o(u,v) = (1/2)[H(u,v) - H(-u,-v)] \tag{7.32a}$$

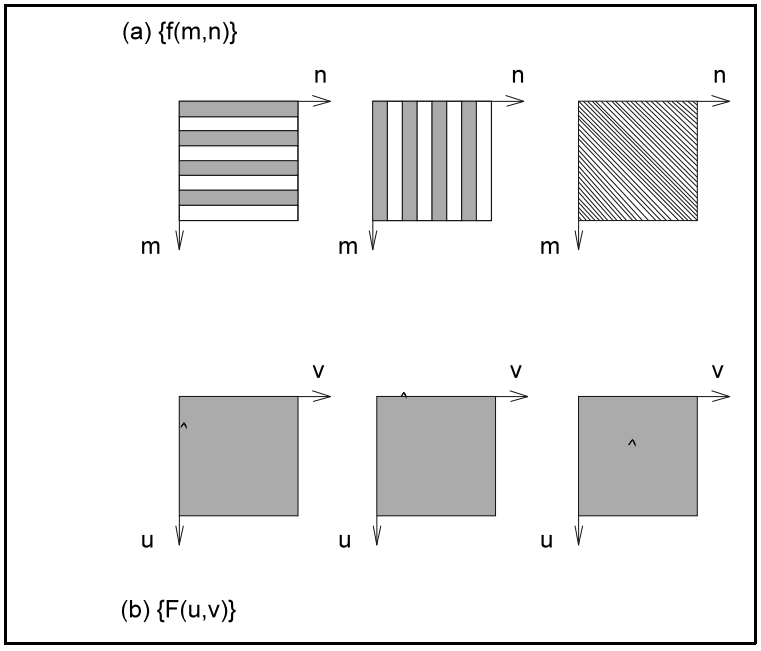
$$H(u,v) = H_e(u,v) + H_o(u,v) \tag{7.32b}$$

Virtually all of the filters we use in image processing are even, in which case the odd part $H_o(u,v)$ is zero. In that case, we use

$$G(u,v) = H(u,v)F(u,v) \tag{7.33}$$

Examples of Simple Spectral Images. Figures 7.6a present three simple images and Figures 7.6b present their spectra (frequency images) in the row underneath the spatial images. The first has spatial variation only in the m-direction (x-axis), while the second has spatial variation of grayscale only in the n-direction (y-axis). The third image is interesting because it has nonzero frequency in each of the m-direction and the n-direction.

Figure 7.6. Examples of Images and Their Spectra



The spectrum {F(u,v)} of an image {f(m,n)} is obtained by a linear transformation that is additive. This implies that if all of the images of Figure 7.6a were added into a single image, the spectrum would be the sum of all of the sprectra of Figure 7.6b. An image of a complex scene has many different frequencies (rates of change in gray scale) across the rows and columns. Thus the spectrum has various shades of gray when displayed as an image.

The Frequency Filtering Process. The process of filtering a 2-dimensional image requires first that the image {f(m,n)} be transformed via Equation (7.28b) to obtain the spectrum {F(u,v)} which has a real and imaginary part at each (u,v) because of the complex exponential model. Thus the result is

$$F(u,v) = F_{Re}(u,v) + jF_{Im}(u,v) \tag{7.27a}$$

A real valued filter H(u,v) is then applied to obtain the filtered frequency image parts

$$G_{Re}(u,v) = H(u,v)F_{Re}(u,v),\ \ G_{Im}(u,v) = H(u,v)F_{Im}(u,v) \tag{7.27b}$$

This filtered spectrum $G(u,v) = G_{Re}(u,v) + jG_{Im}(u,v)$ is then inverted via Equation (7.28a) with G(u,v) in place of F(u,v) to obtain the filtered image {g(m,n)}, where $g(m,n) = g_{Re}(m,n) + jg_{Im}(m,n)$. Here we throw away the imaginary part and keep the real part as the new image.

Computational Complexity. This process of computing the 2-dimensional spectrum of an image takes MN multiplications to compute F(u,v) for every frequency pair (u,v), and there are MN such pairs. Thus M^2N^2 multiplications are required (and the same number of additions) to obtain {F(u,v)}. After {F(u,v)} has been processed to obtain {G(u,v)}, the new values of {g(m,n)} must be computed, which takes another M^2N^2 multiplications.

For M = N = 512, M^4 is more than 68 billion (multiplications). At one million multiplications per second, this requires 68,000 seconds, or more than 19 hours. Another 19 hours of multiplications are required to invert the filtered frequency image. We obviously do not process images using Equations (7.28a,b) directly. Techniques are available for reducing the computational complexity.

7.4 A Fast Fourier Transform

Row-Column Decomposition. A standard method is to decompose the DFT of Equation (7.28b) in two dimensions into a sequence of 1-dimensional DFTs which is considerably faster than the full process of M^2N^2 multiplications and the same number of additions. Recall that the discrete Fourier transform (DFT) pair (DFT[f(m,n)] = F(u,v), IDFT[F(u,v)] = f(m,n)) is

$$F(u,v) = \sum_{(n=0,N-1)}\sum_{(m=0,M-1)} f(m,n)\exp[-j2\pi(mu/M + nv/N)] \tag{7.34a}$$

$$f(m,n) = [1/(MN)]\sum_{(v=0,N-1)}\sum_{(u=0,M-1)} F(u,v)\exp[j2\pi(mu/M + nv/N)] \tag{7.34b}$$

Equation (7.34a) gives the 2-dimensional DFT while Equation (7.34b) gives the *inverse discrete Fourier transform* (IDFT).

As described above, the computation of each value F(u,v) takes MN multiplications and as many additions. There are MN such values, so the entire process takes M^2N^2 multiplications (and about the same number of additions). It is clear that the IDFT also takes this number of mathematical operations. Further, any algorithm that computes the DFT can also be used to compute the IDFT by dividing by MN and substituting the imaginary j in for -j in the exponent. Because F(u,v) is a complex value for each u and v, the algorithm must be done on each of the real and imaginary parts to obtain the IDFT.

The homomorphic property of exponentials makes the DFT *separable*, that is

$$\exp[-j2\pi(mu/M + nv/N)] = \exp[-j2\pi(mu/M]\exp[-j2\pi(nv/N)] \tag{7.35a}$$

permits us to write Equation (7.34a) successively as

$$F(u,v) = \sum_{(n=0,N-1)}\sum_{(m=0,M-1)} f(m,n)\exp[-j2\pi mu/M]\exp[-j2\pi nv/N)] \tag{7.35b}$$

$$F(u,v) = \sum_{(n=0,N-1)}\{\sum_{(m=0,M-1)} f(m,n)\exp[-j2\pi mu/M]\}\exp[-j2\pi nv/N)] \tag{7.35c}$$

$$F(u,v) = \sum_{(n=0,N-1)}\{F_u(n)\}\exp[-j2\pi nv/N)] \tag{7.36}$$

$F_u(n)$ is a 1-dimensional DFT for each fixed u and each fixed n and there are MN of these. Each of them has used M multiplications. In total, then, the array of all values $\{F_u(n)\}$ requires $M(MN) = M^2N$ multiplications and about the same number of additions.

The DFT $F(u,v)$ is a 1-dimensional DFT on $F_u(n)$ that takes N multiplications according to Equation (7.36). There are MN such values: one for each (u,n) and each requires N multiplications, so the total number of multiplications is $N(MN) = MN^2$ (and similarly for the number of additions). Thus the total number of multiplications (the number of additions is similar) is

$$M^2N + MN^2 = MN(M+N) \tag{7.37}$$

This is the *row-column decomposition* form of *fast Fourier transform* (FFT) in two dimensions. Figure 7.7 shows the situation for row-column decomposition where we sum over m for each n and u and then sum over n for each v.

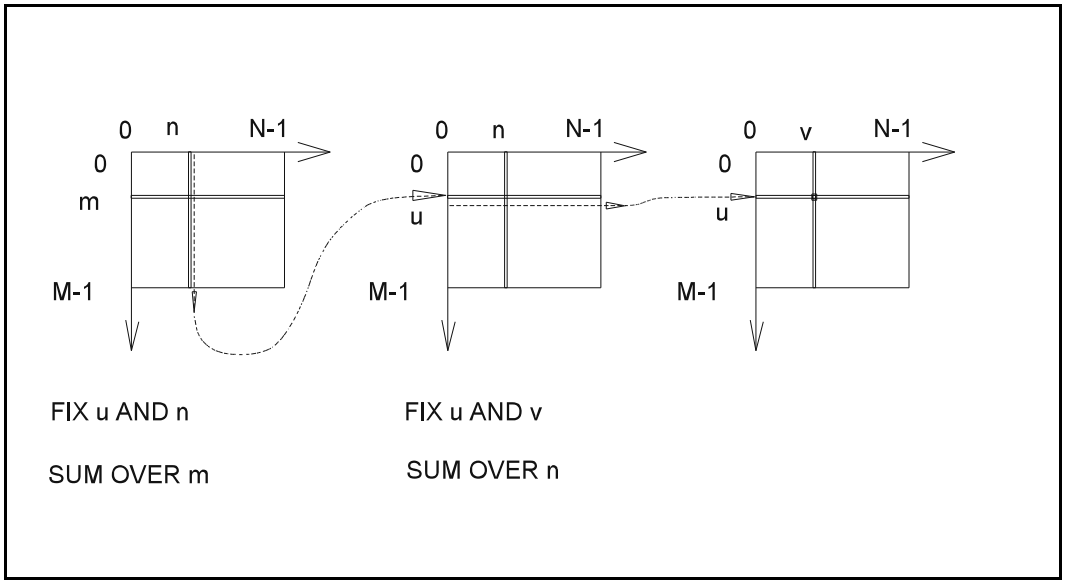


Figure 7.7. The row-column FFT process

For M = N = 512, $MN(M+N) = 2M^3 = 2(512^3) = 134,177,200$ multiplications. At a rate of one million multiplications per second, this would take 134.177 seconds, or about 2.233 minutes. For the full set of computations (100%) there are $M^2N^2 = M^4 = (512)^4 = 68,719,472,000$ multiplications. At 1,000,000 multiplications per second, this takes 19.09 hours to compute. Thus the row-column decomposition takes 0.4% of the time that the full set of computations takes.

This is the simplest form of FFT for 2-dimensional data. Faster algorithms exist, but they are more difficult to analyze and program, and they increase the number of additions. There is another way to increase the speed. The row-column decomposition uses 1-dimensional DFTs, for which there are 1-dimensional FFTs such as the Cooley-Tukey algorithm, the Winograd algorithm, and the prime factor algorithms. Appendix 4a lists an algorithm for a 1-dimensional FFT that rearranges the order of the functional values via the *bit reversal technique*. We recommend that every professional image processor have one available in software. It can reduce the amount of computation for M = N = 512 to 0.0035% of that required for the full process.

An Algorithm for the Row-Column Decomposition DFT. An MxN image array {f(m,n)} is to be mapped linearly to an MxN frequency image complex array {F(u,v)} via the MxN intermediate image $\{F_u(n)\}$. The algorithm must also go in reverse so it must transform an array of complex numbers. For the image {f(m,n)} the imaginary parts are zero. We modify the form of Equation (7.35c) to obtain

$$F(u,v) = \tag{7.38a}$$

$$\sum_{(n=0,N-1)}\{\sum_{(m=0,M-1)} f(m,n)[\cos(2\pi mu/M) - j\sin(2\pi mu/M)]\}[\cos(2\pi nv/N) - j\sin(2\pi nv/N)]$$

This can be written as

$$\sum_{(n=0,N-1)}\{F_u(n)\}[\cos(2\pi nv/N) - j\sin(2\pi nv/N)] \tag{7.38b}$$

for row-column decomposition using the Gauss-DeMoivre equations. Recall that to multiply the complex numbers (a + jb) and (x + jy) we obtain

$$(a + jb)(x + jy) = (ax - by) + j(ay + bx) \tag{7.38c}$$

The imaginary part consists of the sum of the cross products and the real part is the difference of the real and imaginary parts. Before starting the computations of the FFT, it is efficient to compute the exponential values as cosines and sines via

$$C_u(u,m) = \cos(2\pi mu/M), \quad S_u(u,m) = -\sin(2\pi mu/M), \; u,m = 0,...,M-1 \tag{7.39a}$$

$$C_v(v,n) = \cos(2\pi nv/N), \quad S_v(v,n) = -\sin(2\pi nv/N), \quad v,n = 0,...,N-1 \tag{7.39b}$$

These real numbers are stored in arrays of variables {CU[u,m]}, {SU[u,m]}, {CV[v,n]} and {SV[v,n]} for storage in memory so they will not need to be computed each time a term is summed.

The *intermediate DFTs* of the form $\{F_u(n)\}$ of Equation (7.38a,b) have real and imaginary parts for

$$F_u(u,n) = F_{uRe}(u,n) + jF_{uIm}(u,n) \tag{7.39c}$$

that are stored in computer program variables as

FURe[u,n], FUIm[u,n] (7.39d)

Upon computing the *final DFTs* for the array variable F[u,v], both the real and imaginary parts of Equation (7.38b) must be used to compute the real and imaginary parts of F[u,v], which we denote by

FRe[u,v], FIm[u,v]

Thus the program must compute

FRe[u,v] = FURe[u,n]*CV[v,n] - FUIm[u,n]*SV[v,n] (7.39e)

FIm[u,v] = FUIm[u,n]*CV[v,n] + FURe[u,v]*SV[v,n] (7.39f)

The FFT Algorithm. Given the image as input, we compute the output spectral image.

Given: array {f[m,n]}, m = 0,...,M-1, n = 0,...,N-1

Compute: FURe[u,n], FUIm[u,n], u = 0,...,M-1, n = 0,...,N-1 (Intermediate DFT)

FRe[u,v], Fim[u,v], u = 0,...,M-1, n = 0,...,N-1 (Final DFT)

```
/---------------------------------Step 1-------------------------------------/
for u = 0 to M-1 do                          //For computing intermediate/
   for  m = 0 to M-1 do                      //DFTs FURe[u,v], FUIm[u,v]./
      CU[u,m] ← cos(2πum/M);          //Real & Imaginary parts of/
      SU[u,m] ← -sin(2πum/M);                //the complex exponentials./
for v = 0 to N-1 do                          //For computing the final/
   for  n = 0 to N-1 do                      //DFTs F[u,v]./
      CV[v,n] ← cos(2πvn/N);                 //Real & Imaginary parts of/
      SV[v,n] ← -sin(2πvn/N);                //the complex exponentials./
/---------------------------------Step 2-------------------------------------/
for u = 0 to M-1 do                          //Compute intermediate DFT/
   for n = 0 to N-1 do                       //FU[u,n], but first, initialize the/
      FURe[u,n] ← 0; FUIm[u,n] ← 0;          //real and imaginary parts./
      for m = 0 to M-1 do                    //Sum real & imaginary parts./
        FURe[u,n] ← FURe[u,n] + f[m,n]*CU[u,m]; //Modification needed for IDFT./
        FUIm[u,n] ← FUIm[u,n] + f[m,n]*SU[u,n]; //Modification needed for IDFT./
/---------------------------------Setp 3-------------------------------------/
for v = 0 to N-1 do                          //Compute the final DFTs F[u,v]/
   for u = 0 to M-1 do                       //real & imaginary parts/
      FRe[u,v] ← 0;                          //Initialize the real & imaginary sums/
      FIm[u,v] ← 0;
      for n = 0 to N-1 do                                //Sum over n to get F[u,v]/
         FRe[u,v] ← FRe[u,v] + FURe[u,n]*CV[v,n] - FUIm[u,n]*SV[v,n];
         FIm[u,v] ← FIm[u,v] + FUIm[u,n]*CV[v,n] + FURe[u,n]*SV[v,n];
```

The last two computations come from Equations (7.39e,f) above. This algorithm takes the row-column decomposition DFT of a real valued image {f(m,n)}, but it does not divide the final result by MN, nor does it use *j* in place of *-j*. The inverse discrete Fourier transform must substitute *j* for *-j*, take the DFT of a complex valued image, and then divide all results by MN.

A Row-Column Decomposition Algorithm for the IDFT. This above algorithm can also be used for the IDFT if we put in a selection key so the user can select the features of the IDFT. The general algorithm is the same except that the user is asked to select "f" (for *forward*, or DFT) or "b" (for *backward*, or IDFT). Then the following is inserted

```
if key = 'b'  then c = -1.0;
else               c =  1.0;
```

The general algorithm then multiplies each SU[] and SV[] by c in Step 1 above. At the end of the algorithm, another double loop is inserted to divide by MN whenever *key = 'b'*.

```
for u = 0 to M-1 do
   for v = 0 to N-1 do
    FRe[u,v] ← FRe[u,v]/(MN);
    FIm[u,v] ← FIm[u,v]/(MN);
```

In Step 2, the DFT loop

```
for m = 0 to M-1 do                                  //Sum real & imaginary parts/
   FURe[u,n] ← FURe[u,n] + f[m,n]*CU[u,m];
   FUIm[u,n] ← FUIm[u,n] + f[m,n]*SU[u,n];
```

is replaced by

```
for m = 0 to M-1 do                                  //Sum real & imaginary parts/
   FURe[u,n] ← FURe[u,n] + fRe[m,n]*CU[u,m] - fIm[m,n]*SU[u,m];
   FUIm[u,n] ← FUIm[u,n] + fIm[m,n]*CU[u,n] + fRe[m,n]*SU[u,m];
```

where fRe[m,n] and fIm[m,n] are the respective real and imaginary parts of the image f(m,n) that is replaced by F[u,v] to go backwards.

After an image {f(m,n)} has been transformed into $\{F(u,v)\} = \{F_{Re}(u,v) + jF_{Im}(u,v)\}$, then the frequency image {F(u,v)} is filtered with a real valued filter H(u,v) via

$$G_{Re}(u,v) = H(u,v)F_{Re}(u,v) \tag{7.40a}$$

$$G_{Im}(u,v) = H(u,v)F_{Im}(u,v) \tag{7.40b}$$

The real and imaginary parts of this filtered frequency image $G(u,v) = G_{Re}(u,v) + jG_{Im}(u,v)$ go into the respective real and imaginary parts of $f(m,n) = f_{Re}(m,n) + jf_{Im}(m,n)$ in the algorithm, "b" is selected for *key*, and the output $F_{Re}(u,v)$ will contain the new filtered spatial image to be written to a file as {g(m,n)}. Thus this modified algorithm computes both DFTs and IDFTs.

7.5 Important and Necessary Tips and Tricks to Make FFT Filtering Work

Translation of the Frequency Axes. To effectively filter the FFT |F(u,v)|, we shift the origin in (u,v) to the center of the spectral image. This is done by multiplying f(m,n) by a translation factor to obtain

$$f(m,n)\exp[2\pi j\{(M/2)m + (N/2)n\}] \Rightarrow F(u - M/2, v - N/2)$$

This centers the origin on (u,v) = (M/2, N/2) in the frequency plane. Thus a filter H(u,v) is then symmetric in all directions from the origin and so the 1-dimensional filter described in the previous section works. In transforming back to the spatial domain, we must multiply G(u - M/2, v - M/2) by the term

$$\exp[-2\pi j\{(M/2)m + (N/2)n\}]$$

to account for the original translation.

Note that

$$\exp[2\pi j\{(M/2)m + (N/2)n\}] = \exp[\pi j\{(M)m + (N)n\}] = \exp[\pi j\{m + n\}] = (-1)^{(m+n)}$$

so we multiply each term by 1 or -1.

Scaling. Often the range of the FFT |F(u,v)| is quite large and so we must compress it. This can be done by taking

$$D(u,v) = c \log\{1 + |F(u,v)|\}$$

Upon taking the IFFT of the filtered spectral image, we may need to rescale to expand the range of gray levels (with *Matlab* we use the *imshow(Im, [ ])* where the *[ ]* requests the scaling)..

7.6 Spectral Filtering

Lowpass and Highpass Filters. *Filtering* a 2-dimensional signal is a process of suppressing the magnitude (and thus power) of one or more intervals (*bands*) of frequencies while passing undisturbed, or even boosting, others. For example, a smoothing filter H(u,v) passes all frequencies (u,v) such that $u \le u_c$ and $v \le v_c$, where u_c and v_c are the *cutoff* frequencies of the filter in the respective horizontal and vertical directions. Thus H(u,v) = 1 $u \le u_c$ and $v \le v_c$, but H(u,v) = 0 elsewhere. Upon inversion of the new spectrum G(u,v) = H(u,v)F(u,v) back to an image {g(m,n)}, the new image does not have sharp changes that require higher frequencies, but edges would be broadened and all changes would be more gradual. The image would therefore be smoothed by such *lowpass* filtering. On the other hand, if we passed the high frequencies and suppressed the low frequencies, the edges and differences between neighborhing pixels would be sharper. This is called *highpass* filtering.

Figure 7.8 displays lowpass, highpass, bandpass, and notch filters in a single dimension. Filters are denoted by H(u) for a single frequency dimension, or by H(u,v) for 2-dimensional frequency. In two dimensions a lowpass filter can be specified by a distance cutoff parameter

$$d_0 = (u_0^2 + v_0^2)^{1/2} \tag{7.41a}$$

from the frequency origin. Figure 7.9 presents 2-dimensional lowpass and highpass filters with cutoff frequency d_0, where H(u,v) = H(d) = 1, respectively, for

$$d = (u^2 + v^2)^{1/2} \le d_0, \quad d = (u^2 + v^2)^{1/2} \ge d_0 \tag{7.41b}$$

For a *passive filter*, no power boost is given to any band of frequencies, so that either a frequency (u,v) is passed by multiplying it by 1 or it is attenuated (suppressed) by multiplying it by a positive real number less than unity. An *ideal* filter has infinitely sharp cutoffs. Figures 7.8 and 7.9 show ideal filters. The new filtered frequency images with infinitely sharp cutoffs require infinitely high frequencies to represent such sharp cutoffs, which are not possible with $0 \le u \le M-1$ and $0 \le v \le N-1$.

Figure 7.8. Some Common Filters H(u).

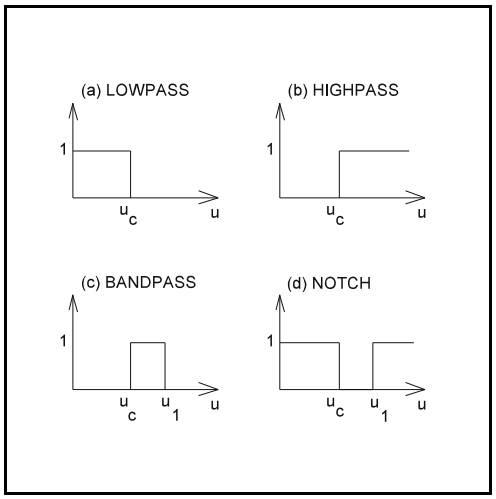
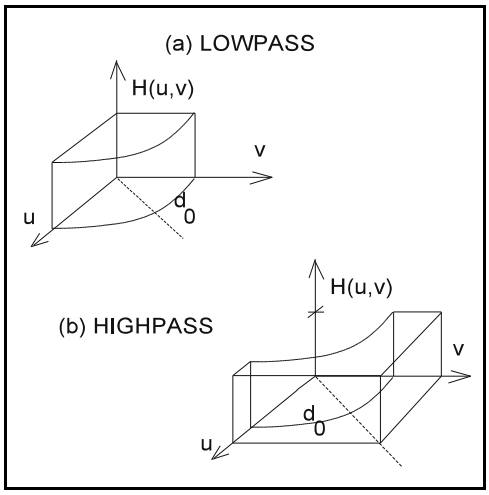


Figure 7.9. Filters Specified by d_0.



Ideal lowpass filters have the 2-dimensional form

$$H(u,v) = \begin{cases} 1, & (u^2 + v^2)^{1/2} \le d_0 \\ 0, & \text{otherwise} \end{cases} \tag{7.41a}$$

and ideal highpass filters have the 2-dimensional form

$$H(u,v) = \begin{cases} 1, & (u^2 + v^2)^{1/2} \ge d_0 \\ 0, & \text{otherwise} \end{cases} \tag{7.41b}$$

For $M \neq N$ we can use the *elliptical* filters with additional design parameters *a* and *b*. We show only the lowpass one here, but the highpass filter is obvious from it.

$$H(u,v) = \begin{cases} 1, & [(au)^2 + (bv)^2]^{1/2} \ge d_0 \\ 0, & \text{otherwise} \end{cases} \tag{7.41c}$$

In general, ideal filters are not suitable real world filters. The requirement of ideal filters for infinitely sharp cutoff causes a problem when inverted back into a new image: the new image has lower frequencies only and the power at these frequencies is distorted by the inversion of the infinitely sharp cutoff. This problem is well known in signal processing as *aliasing*.

Butterworth and Gaussian Filters. Both Butterworth and Gaussian type filters avoid the sharp cutoff, and thus avoid significant aliasing of the processed image. Butterworth lowpass and highpass filters have the respective forms

$$H_{lo}(d) = 1 \,/\, \{1 + [d/d_0]^{2n}\}, \quad H_{hi}(d) = 1 \,/\, \{1 + [d_0/d]^{2n}\} \tag{7.42}$$

where $d = (u^2 + v^2)^{1/2}$ and the design parameter d_0 is a cutoff distance in the frequency plane. Figure 7.10a presents a lowpass and 7.10b displays a highpass Butterworth filter. Another design parameter is n and n = 1 provides a more gradual roll-off while n = 2 gives a sharper, but still rounded roll-off.

Figure 7.11 shows a lowpass and a highpass Gaussian filter. These filters have the respective forms

$$H_{lo}(d) = \exp[-d^2/(2\sigma^2)], \; H_{hi}(d) = 1 - \exp[-d^2/(2\sigma^2)] \tag{7.43}$$

The design parameter σ gives the *spread* of the filter. For example, at $d_0 = \sigma = (M+N)/4$, the filters effectively pass the frequencies on one side of $d_0 = (M+N)/4$ while attenuating those on the other side. Note that $H(\sigma) = \exp[-\sigma^2/(2\sigma^2)] = \exp[-1/2] = 0.607$.

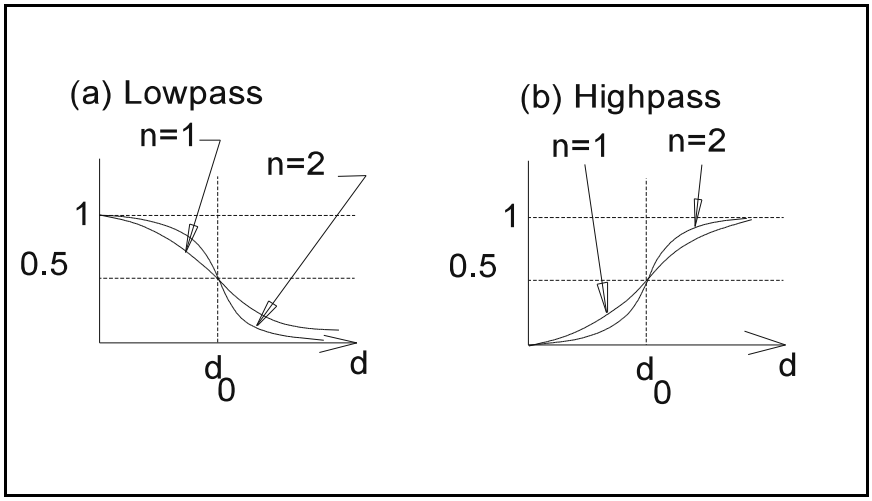


Figure 7.10. Lowpass and Highpass Butterworth Filters.

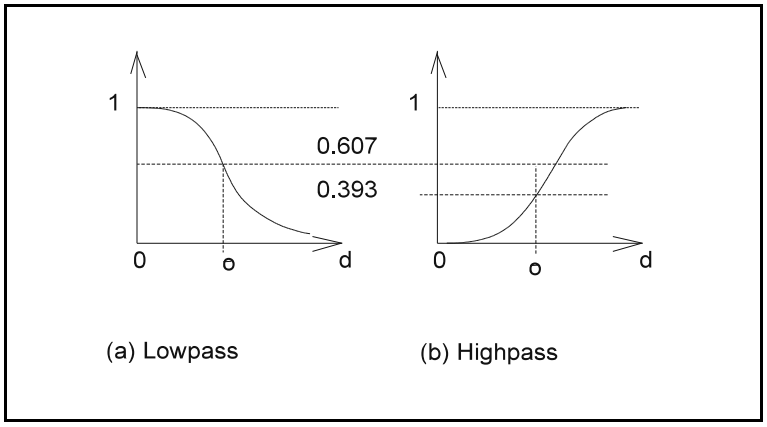
The DFT F(u,v) of an image f(m,n) can be expressed as a *magnitude* and a *phase* spectrum respectively defined via

$$|F(u,v)| = \{F_{Re}(u,v)^2 + F_{Im}(u,v)^2\}^{1/2} \tag{7.44a}$$

$$\Theta(u,v) = \text{atan}(F_{Im}(u,v) / F_{Re}(u,v)) \qquad (7.44b)$$

If we transform only the real part back, then we lose the phase part so that the gray levels will be changed as we filtered them but they will be located in different places relative to the image coordinates. Thus the image will be strongly degraded.

Figure 7.11. Gaussian lowpass and highpass filters.



7.7 Frequency Filtering with *Matlab*

Computing and Displaying the FFT of an Image. The function *fft2(Im1)* is used to compute the FFT of an image *Im1*. The FFT converts a spatial image {f(m,n)} into a spectral, or frequency, image {F(u,v)} where each spectral pixel F(u,v) is complex valued. The origin (u,v) = (0,0) of {F(u,v)} is at the upper left corner of the complex image {F(u,v)} and must be shifted to the center so that a filter H(u,v) can be applied (see the next section). To do this we shift the origin with the function *fftshift(F)*, where *F* is the FFT of *Im1*.

In order to display an FFT we must convert the complex valued pixels to real magnitudes, and additionally, we need to take its logarithm so it will not have a range that is too wide. For these we use the functions *abs()* and *log()*. When we take the inverse FFT of a spectral image {F(u,v)} we obtain a complex valued spatial image in the spatial domain. We can only display the real values of this, which is done by the function *real()*; The following codes at the command line take the image I1 into an FFT and shift it, then do an inverse shift and take the inverse FFT, which is then followed by taking the real part for display.

```
>> I2 = real(ifft2(ifftshift(fftshift(fft2(I1)))));   %take IFFT of  FFT of image I1
>> imshow(I2, [ ]);                                    %show IFFT of FFT result ([ ] scales image)
```

Now we break this up and show the parts that will be useful in filtering in the spectral domain to be used separately. First we take the FFT, then take the logarithm of the magnitude (absolute value) of the FFT for display.

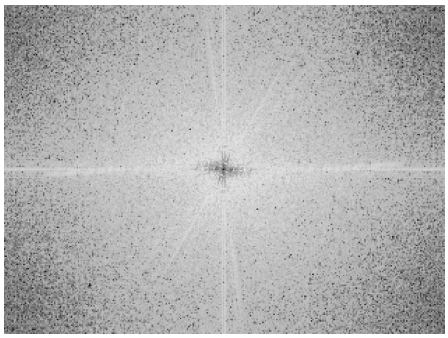
```
>> F = fftshift(fft2(I));          %take FFT of I and shift the origin to the center
>> F1 = log(abs(F));               %get the log of the absolute value for display
>> imshow(F1, [ ]);                %display the result with scaling (we must include [ ])
```

Now we convert these back into a real valued spatial image by inversely shifting the origin back to the upper left corner and then taking the IFFT, followed by getting the real part and displaying it. Figure 7.12 shows the recovered original image and Figure 7.13 displays the FFT (the logarithm of the absolute value).

```
>> I2 = real(ifft2(ifftshift(F)));      %shift back, take the IFFT and get real part
>> figure, imshow(I2, [ ]);             %show the result with scaling (with [ ]).
```

Figure 7.12. Recovered original building image.

Figure 7.13. FFT of the building image.



Designing Filters. Filters can be designed in the spectral (frequency) domain on a mesh grid. We first make a mesh grid in the two dimensions u and v by means of the function *freqspace(size, string)*, where we take *size* to be 256 and *string* to be *'meshgrid'* to specify the array. We then define a filter function H is the spectral domain by means of the function *fspecial(string, size, sigma)*, where string here is 'gaussian' and the size is 256x256. The width is designated by *sigma*, which we take to be 64 here.

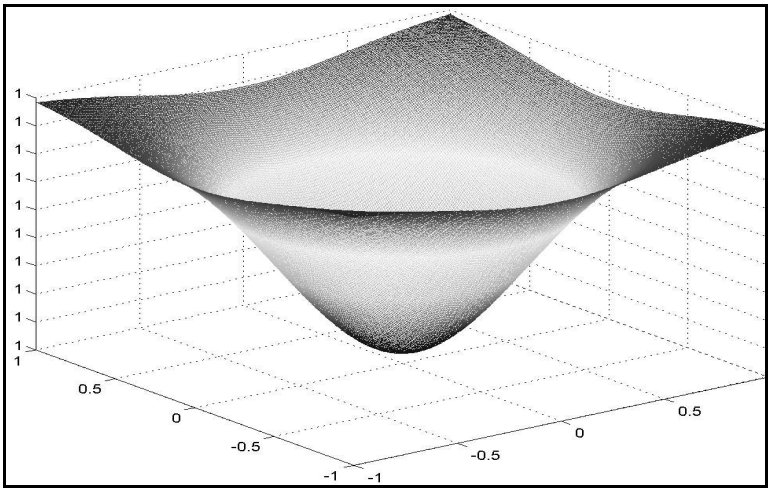
```
>> [u, v] = freqspace(256, 'meshgrid');   %set up 256x256 mesh grid in spectral domain
>> H = fspecial('gaussian', 256, 64);     %make lowpass Gaussian, 256x256 with sigma = 64
```

We next subtract every value in the array H from 1 to obtain an inverted Gaussian that is a highpass filter. We display the mesh grid in *Matlab* with the function *mesh(x, y, A),* where the dimensions are *x* and *y* and *A* is the array. Thus we have

```
>> H1 = 1 - H;                          %1 - H is the highpass inverted Gaussian
>> mesh(u, v, H1);                      %show 3-D inverted Gaussian on mesh grid
```

We now have a high pass filter centered on the origin (u,v) = (0,0) in the spectral domain that will attenuate (suppress) the low frequencies in the center and pass the high frequencies at the higher frequencies at the outer parts of the mesh grid. Now we will read in a 256x256 image and take the FFT of it to get its spectral image. We will actually be able to display the spectral image as a 3-dimensional intensity image (the magnitude versus frequency pairs (u,v)) as well as in a 2-dimensional image display. The commands are given below.

Figure 7.14. The inverted Gaussian.




```
>> I = imread('camera256.tif');        %read in cameraman image
>> figure, imshow(I);                  %show cameraman image
>> F = fftshift(fft2(I));              %take FFT and shift origin to center (u,v) = (0,0)
>> F1 = log(abs(F));                   %take log-magnitude (absolute value) for display
>> figure, mesh(u, v, F1);             %display FFT magnitude over mesh display
>> figure, imshow(F1, []);             %scale and show log-magnitude of the FFT
```

Figure 7.15. 3-D mesh grid of FFT.

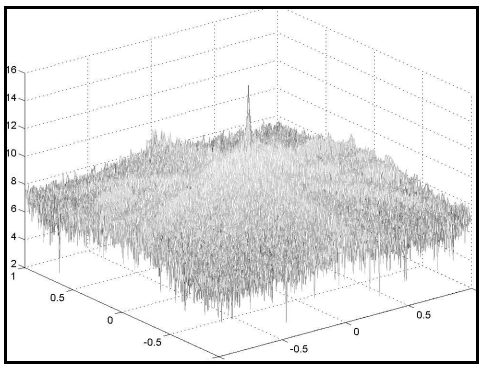
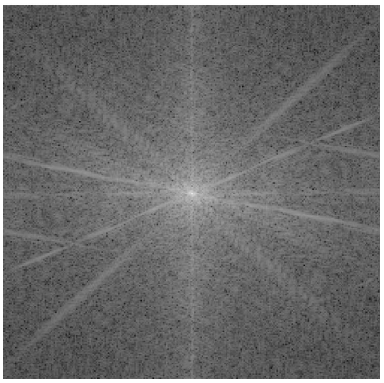


Figure 7.16. Log magnitude of FFT of image.



Now we filter the FFT F of the cameraman image by multiplying F by the inverted Gaussian filter H1. Afterward, we display the log-magnitude of the FFT.

```
>> G = H1 .* F;                        %filter FFT by inverted Gaussian filter H1
>> g = real(ifft2(ifftshift(G)));      %get real part of IFFT of filtered image
```

Figure 7.17. Original camerman image.

Figure 7.18. Highpass Gaussian filtered image.

Now we experiment with both sharpening and smoothing via the following commands.

```
>> [u, v] = freqspace(256,'meshgrid');
>> H = fspecial('gaussian', 256, 64);
>> H1 = 1 - H; figure, mesh(u, v, H1);
>> F = fftshift(fft2(I));
>> G = H .* F;
>> Ig = real(ifft2(ifftshift(G)));
>> figure, imshow(Ig, []);
```

The results are shown in Figures 7.19 (smoothed with a lowpass Gaussian filter) and 7.20 (sharpened with a highpass Gaussian filter).

Figure 7.19. Lowpass Lena image.

Figure 7.20. Highpass Lena image.

7.8 The Convolution Theorem

Linear Systems. Because of the computational complexity of the 2-dimensional DFT (discrete Fourier transform), or of the DHBT (discrete Hartley-Bracewell transform), conversion and inversion, and of the problems of scaling the inverted processed image properly, we rarely use this method directly. Instead, we design a filter H(u,v) in the frequency domain and use its inversion h(m,n) to the spatial domain, and

then process the image directly in the spatial domain by use of convolution. The convolution theorem makes this possible. We consider the 1-dimensional case first.

A linear electronic or optical system can be represented, or modeled, by a linear function L[·] that has the basic linear properties

$$L[a_1 f_1(x) + a_2 f_2(x)] = a_1 L[f_1(x)] + a_2 L[f_2(x)] \tag{7.45}$$

An input f(x) *excites* a system that then yields an output *response* g(x) = L[f(x)]. The linear system affects the input in two ways: (i) it multiplies it by a constant *gain* c (g(x) = cf(x) = L[f(x)]); and ii) it is *shift-invariant* (g(x) = L[f(x - x_0)] = L[f(x)]). A physical linear system has both a gain and a delay

$$g(x) = L[f(x)] = cf(x - x_0) \tag{7.46}$$

It has been observed that if we put a sinusoid signal at a single frequency k (on a fundamental period T) with amplitude 1 through the linear system L with the above properties we obtain

$$g(x) = L[f_k(x)] = L[\cos(2\pi kx/T)] = c_k \cos(2\pi k(x - x_0^{(k)})/T) \tag{7.47a}$$

Upon putting another frequency r ≠ k through, we obtain

$$g(x) = L[f_r(x)] = L[\cos(2\pi rx/T)] = c_k \cos(2\pi r(x - x_0^{(r)})/T) \tag{7.47b}$$

Thus by the linearity property, if we put a function f(x) with multiple frequencies through a linear system, we can write

$$g(x) = L[f(x)] = L[a_0 + \Sigma_{(k=1,N-1)} [a_k \cos(2\pi kn/N) + b_k \sin(2\pi kn/N)]] = \tag{7.47c}$$

$$L[a_0] + \Sigma_{(k=1,N-1)} a_k L[\cos(2\pi kn/N)] + b_k L[\sin(2\pi kn/N)]$$

Each of the terms in the summation is an output with the same frequency as the input component, by Equation (7.47a) above. Thus the output has the same frequencies, but it has a gain and a shift.

It has been observed that the gain and shift are both dependent upon the frequency. Thus we conclude that in general, a function f(x) that contains multiple frequencies is transformed by a linear system according to

$$g(x) = L[f(x)] = ß(u) \cdot f(x - x_0(u)) \tag{7.47d}$$

where the gain ß(u) is a function of the frequency u, which could be continuous, and $x_0(u)$ is a shift that depends upon the frequency u.

The Convolution Integral and Sum. It is common to model a linear electronic or optical system by an integral that sums up the effects of the system on the input signal over time or space. The most general model for f(x) → g(x) is

$$g(x) = \int_{-\infty}^{\infty} h(x,s)f(s)ds \tag{7.48a}$$

This model must also satisfy the shift invariant property, so that

$$g(x-\tau) = \int_{-\infty}^{\infty} h(x-\tau,s)f(s)ds = \int_{-\infty}^{\infty} h(x,s)f(s)ds = g(x) \tag{7.48b}$$

Upon adding τ to both s and x, we obtain the new formulation

$$g(x) = \int_{-\infty}^{\infty} h(x,s+\tau)f(s+\tau)ds \tag{7.48c}$$

Equation (7.48c) is equal to Equation (7.48a), and so we conclude that

$$h(x+\tau,s+\tau) = h(x,s) \tag{7.49a}$$

is true for all τ. Thus the difference between s and x is all that matters here. We define a new function

$$h(x-s) = h(x,s) \tag{7.49b}$$

(put $\tau \leftarrow -s$ in Equation (7.49a) to obtain h(x-s, 0) for any s and x and drop the 0). Thus

$$g(x) = \int_{-\infty}^{\infty} h(x-s)f(s)ds, \quad \text{for each } x, -\infty < x < \infty \tag{7.49c}$$

Now we define the *convolution operation* on two functions f(x) and g(x) via

$$h(x)*f(x) = \int_{-\infty}^{\infty} f(s)h(x-s)ds = \int_{-\infty}^{\infty} f(x-s)h(s)ds \tag{7.50}$$

The second integral follows from the first by making the proper substitutions.

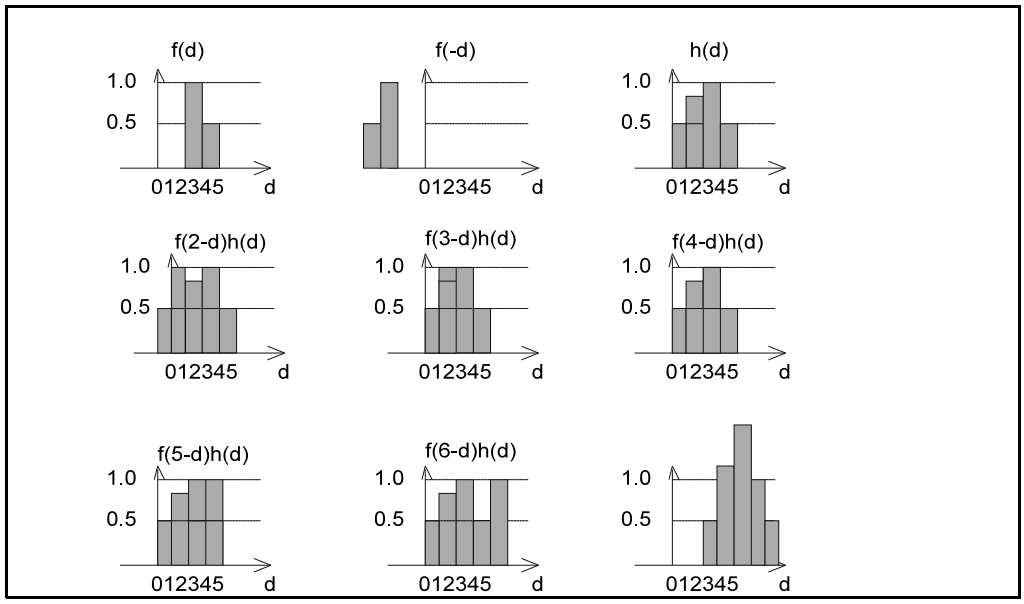
The *discrete convolution operation* between discrete functions {h(n)} and {f(n)} is

$$g(n) = f(n)*h(n) = \sum_{(k=-\infty,\infty)} f(k)h(n-k) = \sum_{(k=-\infty,\infty)} f(n-k)h(k) \tag{7.51}$$

for every n = 0,...,N-1. If the length of {h(n)} and {f(n)} are respectively M and N, then the output function {g(n)} has length of M + N - 1.

Figure 7.19 shows the step-by-step convolution of two discrete functions of finite respective lengths of 2 and 4. The result of the convolution is a function of finite length 5. Convolution spreads functions. For each d, the *folded* function f(-d) = f(0-d) is centered on a point and the products are summed. This yields a function for g(d) for which at each point d, all of the effects the system {h(r)} are contributed.

Figure 7.19. Convolution of two discrete functions



Convolution in Two Dimensions. For an image {f(m,n)} and another function {h(m,n)}, their discrete convolution in two dimensions is

$$g(m,n) = f(m,n)*h(m,n) = \sum\sum_{(k=0,N-1;\ell=0,M-1)} f(\ell,k)h(m-\ell,n-k) \tag{7.52}$$

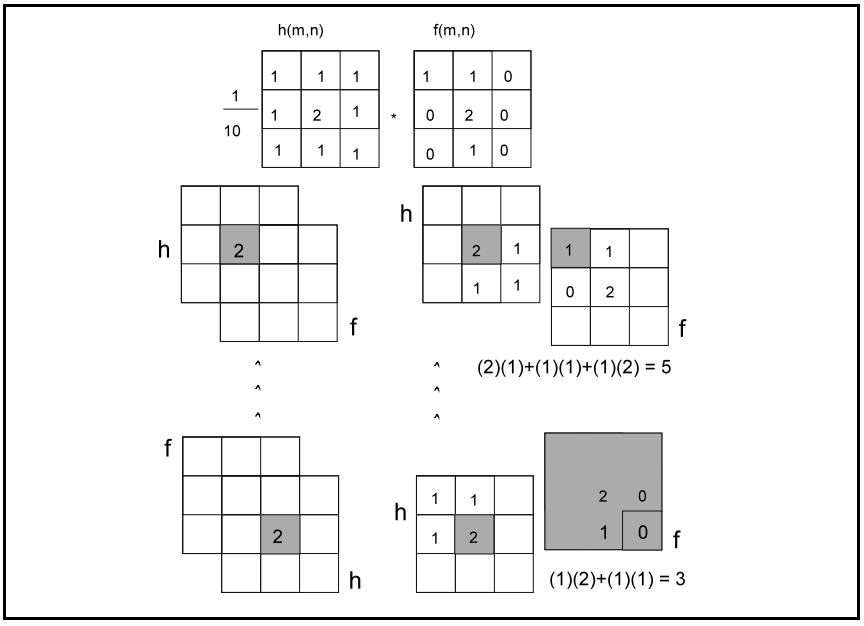
at each pixel location (m,n). A new image is formed by means of some linear system {h(m,n)} that operates on the true image {f(m,n)} to form the actual image {g(m,n)} via

$$g(m,n) = f(m,n)*h(m,n) \tag{7.53}$$

One way to remove distortion caused by the effects of a linear system {h(m,n)} is to estimate h(m,n) and design a filter H(u,v) that represents the system and to use the inverse Fourier transform $\tilde{h}$(m,n) to partially remove the distortion. For most image processing, we can design a filter H(u,v), take the IDFT h(m,n), and perform convolution on the image to filter it.

Figure 7.20 shows the convolution of two discrete functions in two dimensions. The origin is at the center of the mask {h(m,n)}. Because h(m,n) is symmetrical about that origin, h(-m,-n) = h(m,n). For each point (m,n), we center {h(k,ℓ)} at (m,n) and perform the convolution to obtain all effects of {h(k,ℓ)} on f(m,n). It is clear that {h(k,ℓ)} is a convolution mask and that {f(m,n)} could be an image of any size.

Figure 7.20. Example of 2-dimensional discrete convolution



The Convolution Theorem. Let f(x) have discrete Fourier transform F(u) and h(x) have DFT H(u). Then the convolution theorem states that f(x)*h(x) and F(u)H(u) are transform pairs, that is

$$f(x)*h(x) \overset{FT}{\underset{IFT}{<==>}} F(u)H(u)$$

In other words, the product of F(u) and H(u) is a Fourier transform that inversely transforms back into the function given by f(x)*h(x), and f(x)*h(x) Fourier transforms to F(u)H(u).

To see that this is so, consider that

$$FT[f(x)*h(x)] = \int_{-\infty}^{\infty} f(x)*h(x)\exp(-j2\pi ux)dx = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(s)h(x-s)\exp(-j2\pi ux)dsdx = \tag{7.54}$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(s)h(x)\exp(-j2\pi u(x+s))dsdx = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\{f(s)\exp(-j2\pi us)ds\}\{h(x)\exp(-j2\pi ux)dx\} =$$

$$\int_{-\infty}^{\infty}\{f(s)\exp(-j2\pi us)ds\}\int_{-\infty}^{\infty}\{h(x)\exp(-j2\pi ux)dx\} = F(u)H(u)$$

Therefore, FT[f(x)*g(x)] = F(u)G(u) (q.e.d.).

Figure 7.21. The filtering process.

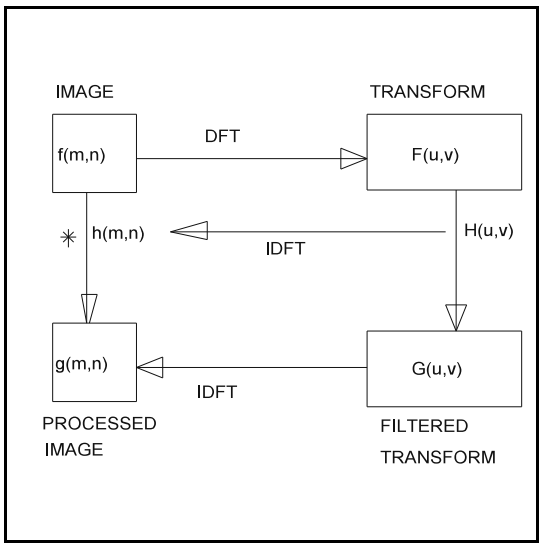
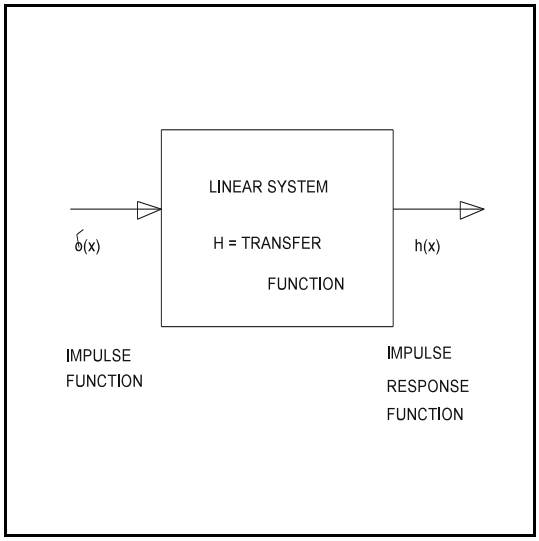


Figure 7.22 System response to δ(x).



In two dimensions, the same situation holds, so that

$$FT[f(m,n)*h(m,n)] = F(u,v)H(u,v), \quad f(m,n)*h(m,n) = IDFT[F(u,v)H(u,v)] \tag{7.56}$$

This allows us to perform convolution in the spatial domain that is equivalent to filtering in the frequency domain. Figure 7.21 shows the filtering processes and Figure 7.22 displays a block diagram for it.

Filtering Images via Convolution Using FT Properties. We now have all of the tools that we need to filter an image {f(m,n)} directly in the spatial domain. Suppose that we view an image {f(m,n)} and design a filter H(u,v) in the frequency domain to filter the image a particular way. Rather than taking the FT of {f(m,n)} and using H(u,v) on the Fourier transform (frequency image) {F(u,v)} of {f(m,n)} via G(u,v) = H(u,v)F(u,v) and then taking the inverse Fourier transform {g(m,n)} of G(u,v), we instead take the inverse Fourier transform {h(m,n)} of H(u,v). The filter H(u,v) is known as the *transfer function* for the linear system that filters input functions {f(m,n)}. Figure 7.15 shows the situation.

The IDFT h(m,n) of the transfer function H(u,v) is called the *impulse response function* of the linear system that H(u,v) represents. The *delta impulse function* (or impulse function), is defined by

$$\delta(x) = \begin{cases} \infty, & x = 0 \\ 0, & \text{otherwise} \end{cases} \tag{7.57}$$

$$\int_{-\infty}^{\infty} \delta(x)dx = 1 \tag{7.58}$$

It follows that

$$\delta(x - x_0) = \begin{cases} \infty, & x = x_0 \\ 0, & \text{otherwise} \end{cases} \tag{7.59}$$

$$\int_{-\infty}^{\infty} \delta(x)dx = \int_{x_0-\epsilon}^{x_0+\epsilon} \delta(x)dx = 1 \tag{7.60}$$

Knowing what a linear system does to the impulse function describes the system behavior. We first observe that the convolution of any function f(x) with $\delta(x-x_0)$ returns the value $f(x_0)$. Let h(x) be the output of a linear system when $\delta(x)$ is the input.

$$\int_{-\infty}^{\infty} \delta(x_0-s)f(s)ds = \tag{7.61}$$

$$f(x_0) \int_{-\infty}^{\infty} \delta(x_0-s)ds = f(x_0)(1) = f(x_0)$$

In the discrete case

$$\delta(k) = \begin{cases} 1, & k = 0 \\ 0, & \text{otherwise} \end{cases} \tag{7.62}$$

$$\sum_{(k=-\infty,\infty)}\delta(k) = 1, \ \sum_{(k=-\infty,\infty)}\delta(n-k) = 1 \tag{7.63}$$

It follows that for each n, a disrete function {f(n)} can be expressed as

$$f(n) = \sum_{(k=-\infty,\infty)}\delta(n-k)f(k) = \sum_{(k=-\infty,\infty)}\delta(k)f(n-k) \tag{7.64}$$

From Equation (7.64) we can see the effects of a linear system on the input function f(n).

$$L[f(n)] = \sum_{(k=-\infty,\infty)} L[\delta(n-k)f(k)] =$$

$$\sum_{(k=-\infty,\infty)} f(k)L[\delta(n-k)] = \sum_{(k=-\infty,\infty)} f(k)h(n-k) = f(n)*h(n) \tag{7.65}$$

for each n. This is due to {f(k)} being a sequence of real values. Equation (7.54) establishes that the impulse response {h(k)} determines the system behavior on any input signal f(x) (q.e.d.).

Having designed an appropriate filter H(u,v) and obtained h(m,n) via the IDFT, we apply it to {f(m,n)} via convolution to obtain the filtered image {g(m,n)} via

$$g(m,n) = h(m,n)*f(m,n) \tag{7.66}$$

According to Equations (7.55), {g(m,n)} is the same image that we would obtain by taking the inverse Fourier transform of

$$G(u,v) = H(u,v)F(u,v) \tag{7.67}$$

In practice, {h(m,n)} is truncated to be all zeros except for a small mask. We will see in the next chapter that truncation can be dangerous unless we can prevent aliasing.

7.9 Fourier Transforms and Spatial Filtering Examples

Some Fourier Transform Pairs and Properties. The FT pairs listed in Table 7.1 are the most useful for what follows in the next chapter. Upon integrating the function f(x) in the Fourier transform integral, the FT F(u) can be obtained, or in some cases, integrating F(u) in the inverse Fourier transform integral yields the resultant f(x). By combining the pairs with properties, we can obtain new filters.

Table 7.1. Fourier Transform Pairs and Properties

Function	Fourier Transform
1. $f(x) = 1$, all x (constant) $f(x) = c$, all x (constant)	$F(u) = \delta(u)$ (impulse function) $F(u) = c\delta(u)$ (impulse function)
2. $\delta(x)$ (impulse function) $c\delta(x)$ (impulse function)	$F(u) = 1$, all u (constant) $F(u) = c$, all u (constant)
3. $f(x) = A\text{sinc}(2u_0x) = A\sin(2\pi u_0x)/(2\pi u_0x)$	$F(u) = A/(2u_0)$, $\lvert u\rvert \le u_0$ 0, elsewhere
4. $f(x) = A$, $\lvert x\rvert \le x_0$ 0, elsewhere	$F(u) = 2Ax_0\text{sinc}(2ux_0) = 2Ax_0\sin(2\pi x_0u)/(2\pi x_0u)$ all u
5. $f(x) = \exp[-\pi x^2]$, all x	$F(u) = \exp[-\pi u^2]$, all u
6. $f(x) = 1 - \lvert x\rvert/x_0$, $\lvert x\rvert \le x_0$	$F(u) = x_0\text{sinc}^2(x_0u)$, all u
7. $f(x) = \exp[-a\lvert x\rvert]$, all x	$F(u) = (2a)/[a^2 + (2\pi u)^2]$, all u

8. *Scaling*: f(x) <=> F(u) if and only if f(ax) <=> (1/|a|)F(u/a)

9. *Duality*: f(x) <=> F(u) if and only if F(x) <=> f(-u)

10. *Multiplication*:
if ($f_1(x)$ <=> $F_1(u)$ and $f_2(x)$ <=> $F_2(u)$) then ($f_1(x)*f_2(x)$ <=> $F_1(u)F_2(u)$)

11. *Linearity*: $af_1(x) + bf_2(x)$ if and only if $aF_1(u) + bF_2(u)$

Items 8, 9, 10 and 11 in Table 7.1 are properties that can be used to create other FT pairs. For example, Items 1 and 2 are duals of each other. To put Item 5 in Gaussian form, we scale x by

$$\alpha = 1/(2\pi\sigma^2)^{1/2} \tag{7.68}$$

and thus scaling x^2 by

$$\alpha^2 = 1/(2\pi\sigma^2) \tag{7.69}$$

we obtain

$$\exp[-\pi(\alpha x)^2] \leftrightarrow (1/|\alpha|)\exp[-\pi(u/\alpha)^2] \tag{7.70}$$

$$\exp[-x^2/(2\sigma^2)] \leftrightarrow (2\pi\sigma^2)^{1/2} \exp[-(2\pi^2\sigma^2)u^2] \tag{7.71}$$

By the duality property (Item 9 of Table 7.1) and the fact that a Gaussian filter is symmetrical about the origin, we have that

$$(2\pi\sigma^2)^{1/2} \exp[-(2\pi^2\sigma^2)x^2] \leftrightarrow \exp[-u^2/(2\sigma^2)] \tag{7.72}$$

Thus Gaussians transform to Gaussians. For large σ, the Gaussian impulse response function

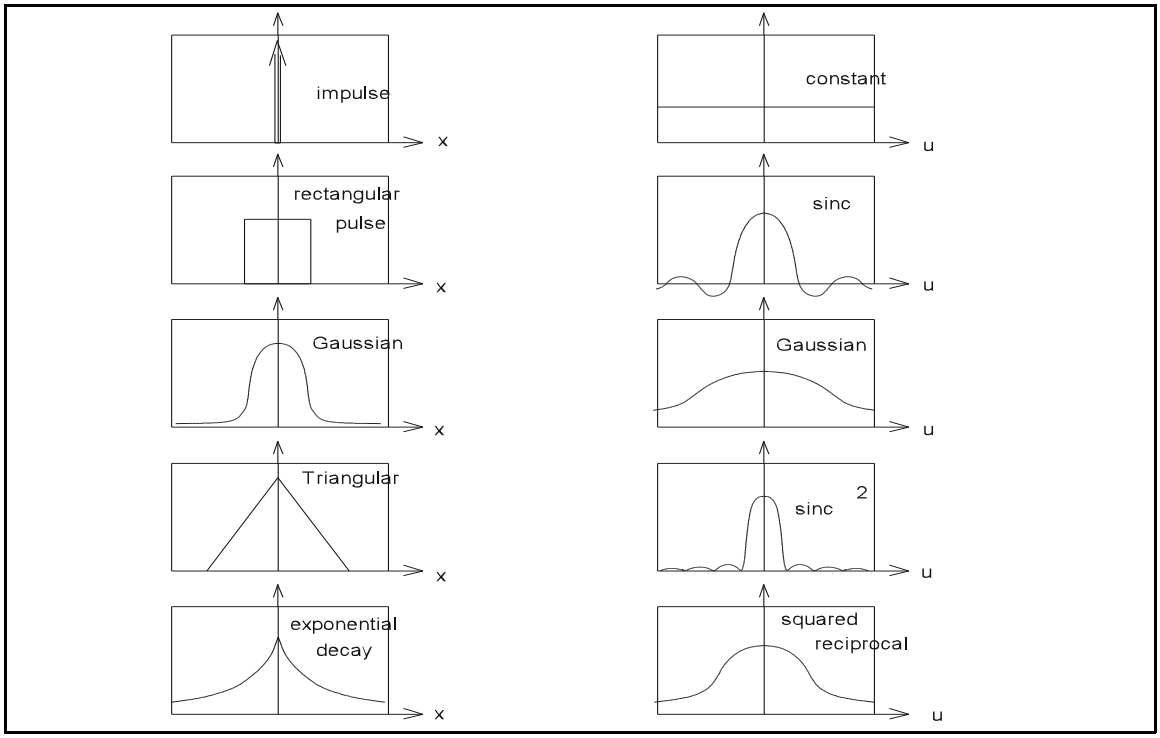
$$h(d) = \exp[-d^2/(2\sigma^2)] \tag{7.73}$$

is a wide rounded pulse and maps to

$$F(u) = (2\pi\sigma^2)^{1/2} \exp[-u^2/(1/(2\pi^2\sigma^2))] \tag{7.74}$$

Upon letting $\sigma^2 \to \infty$, we get wider and wider pulses (with rounded corners) of unit height that transform to higher and higher pulses that are narrower and narrower. In the limit, the unit constant function Fourier transforms to the impulse (infinite spike) function $\delta(u)$. Figure 7.23 shows the shape of these transform pair functions. In general, higher narrow pulses transform to lower wide pulses and vice versa. The shape of the pulses is important (for example, rectangular versus triangular versus Gaussian versus sinc). By duality, each pair can be reversed in order with the x and u interchanged.

Figure 7.23. Graphs of transform pairs



Gaussian Lowpass and Highpass Filtering. We present here two examples of filtering with Gaussian filters. Let us design a Gaussian lowpass filter for "Shuttle" (a 320x240 image) similar to that of Figure 7.11a.

We standardize the spatial domain by letting [-M/2,M/2] map to [-1/2,1/2] upon dividing by M. We standardize the frequency domain as well by mapping [-M/2,M/2] to [-1/2,1/2] (we use the minimum M of M = 240 and N = 320). We now design a Gaussian filter with $\sigma_u = 1/2$ (which we use for the cutoff frequency). The standardized Gaussian filter (as a function of frequency distance d) is

$$H(d) = \exp[-d^2/(2\sigma_u^2)] = \exp[-2d^2] \tag{7.75}$$

The impulse response function is

$$h(d) = (2\pi\sigma_u^2)^{1/2} \exp[-2\pi^2\sigma_u^2 d^2] = (\pi/2)^{1/2} \exp[-\pi^2 d^2/2] = 1.2533\exp[-4.9348d^2] \tag{7.76}$$

The filter has the values

$$H(0) = 1,\ H(\sigma_u/2) = \exp[-1/8] = 0.8825,\ H(\sigma_u) = \exp[-1/2] = 0.607$$

Thus the filter is a reasonable lowpass filter. In the spatial domain we have

$h(0) = 1.2533\exp[0] = 1.2533, \quad h(1/2) = 1.2533\exp[-4.9348/4] = 0.3650$
$h(1) = 1.2533\exp[-4.9348] = 0.0090$

The convolution mask that results from {h(d)} has effective size (M/2)x(M/2). It takes the average of a large portion of the image {f(m,n)} at each point (m,n), which yields strong oversmoothing. We want to reduce the size to, say, a 3x3 mask. The standard deviation of the spatial domain Gaussian must satisfy

$$-2\pi^2\sigma_u^2 d^2 = -d^2/(2\sigma_m^2) \tag{7.77}$$

$$\sigma_m^2 = 1/(4\pi^2\sigma_u^2) = 0.10132, \ \sigma_m = 0.31831 \tag{7.78}$$

In the original spatial domain of pixels, this is (0.31831)*240 = 76.39 pixels. A convolution mask of 38 pixels on each side of the mask origin is too large. We sometimes want to restrict our mask to 3x3. We take the convolution function values according to their distance from the mask origin and so these Euclidean distances are 0, 1 and $2^{1/2}$ times 2/M = 1/(M/2). Thus we arrive at the following mask.

$$\{h(d)\} = (r) \begin{matrix} | h(2^{1/2}/(M/2)) & h(1/(M/2)) & h(h(2^{1/2}/(M/2)) | \\ |h(1/(M/2)) & h(0) & h(1/(M/2)) | \\ |h(2^{1/2}/(M/2)) & h(1/(M/2)) & h(h(2^{1/2}/(M/2)) | \end{matrix} = \tag{7.79}$$

$$(0.08870) \begin{matrix} | 1.2524 & 1.2529 & 1.2524| \\ | 1.2529 & 1.2533 & 1.2529| \\ | 1.2524 & 1.2529 & 1.2524| \end{matrix}$$

The multiplier r is the reciprocal of the sum of all of the entries. This is a smoothing filter as we know from the Gaussian filter that yielded it.

What we desire now is to obtain the convolution mask for a sharpening filter that does not accentuate the noise as does the hedged Laplacian. Consider the filter $H^{\sim}(d)$ in the frequency domain

$$H^{\sim}(d) = 1 - H(d) \tag{7.80}$$

where H(d) is the Gaussian lowpass filter described above. By the linearity of the IDFT

$$h^{\sim}(d) = IDFT[H^{\sim}(d)] = IDFT[1 - H(d)] = IDFT[1] - IDFT[H(d)] = \delta(d) - h(d) \tag{7.81}$$

But {δ(d)} is the discrete impulse function and is therefore the identity mask, while {h(d)} is the mask of Equation 7.64e). We scale the identity mask by the factor 2 so as not to obtain a darkened image. Thus

$$h^{\sim}(d) = \begin{matrix} |0 & 0 & 0| \\ |0 & 2 & 0| \\ |0 & 0 & 0| \end{matrix} - (0.08870) \begin{matrix} |1.2524 & 1.2529 & 1.2524| \\ |1.2529 & 1.2533 & 1.2529| \\ |1.2524 & 1.2529 & 1.2524| \end{matrix} = \tag{7.82}$$

$$\begin{matrix} |-0.1111 & -0.1111 & -0.1111| \\ |-0.1111 & 1.8888 & -0.1111| \\ |-0.1111 & -0.1111 & -0.1111| \end{matrix}$$

Figure 7.24 displays the Gaussian smoothed image due to Equation (7.79). Figure 7.25 shows the result of filtering with the highpass convolution mask of Equation (7.82). It is sharpened without the accentuation of noise, similar to unsharp masking. Compare it with Figure 3.7. We will see in the next chapter that we can exercise more control over the filtering by boosting the higher frequencies, strongly attenuating the midrange frequencies and slightly attenuating the lower frequencies.

Figure 7.24. Gaussian smoothed shuttle.

Figure 7.25. Gaussian sharpened shuttle.

7.10 Exercises

7.1 Compute the (continuous) Fourier transform for f(x), where

$$f(x) = 1, \quad -2 \le x \le 2$$
$$= 0, \quad \text{elsewhere}$$

Use the DeMoivre-Euler equations to obtain the sinc function.

7.2 Repeat Exercise 7.1 with "$-T/2 \le x \le T/2$" in place of "$-2 \le x \le 2$."

7.3 Compute the (continuous) inverse Fourier transform for

$$F(u) = 1, \quad -2 \le u \le 2$$
$$= 0, \quad \text{elsewhere}$$

7.4 Show that the Fourier transform of a Gaussian function $f(x) = \exp(-x^2/2\sigma^2)$ is also a Gaussian type of function. Find σ_f for F(u).

7.5 Explain the difference between the discrete Fourier series and the discrete Fourier transform.

7.6 The convolution (or impulse response) function h(x,y) that represents a Gaussian filter H(u,v) is also a Gaussian function. Each Gaussian is 2-dimensional here. For the Gaussian convolution function

$$h(x,y) = \exp\{-[(x-p_x)^2 + (y-p_y)^2]/(2\sigma^2)\}$$

where (p_x,p_y) is the center pixel (origin) of the mask, find a convolution mask (discrete impulse response function) with which to perform the equivalent of lowpass filtering via mask convolution.

7.7 Find a convolution mask to perform a highpass Gaussian filtering of an image, where the highpass filter has the form

$$H(u,v) = 1 - \exp\{-[u^2 + v^2]/(2\sigma^2)\}$$

In other words, find the inverse Fourier transform of H(u,v) to obtain h(x,y) and then determine a convolution mask from h(x,y).

7.8 Draw an image by overlaying all three of the simple images in Figure 7.5a on a single image. Now find the spectrum of this image. Use the linearity property of the Fourier transform.

7.9 Would there be any problem with a Butterworth lowpass or highpass filter where n was taken to be very large, say n = 10?

7.10 Draw 2-dimensional Butterworth and Gaussian lowpass filters.

7.11 Draw 2-dimensional Butterworth and Gaussian highpass filters.

7.12 Write an algorithm for a general row-column decomposition DFT that can be used for both DFT and IDFT.

7.13 Write a computer program in C for the algorithm of Exercise 7.12.

7.14 Write a computer program in C that implements the Danielson-Lanczos FFT in 1-dimension that is given in Appendix 7a.

7.15 Design an ellipsoidal highpass filter that attenuates u for $u \leq 120$ and v for $v \leq 160$. Determine a convolution mask in the spatial domain that implements it approximately. Filter the shuttle image and compare the results with Figure 7.18.

7.16 Use Matlab to apply a Gaussian lowpass filter to the shuttle image.

7.17 Use Matlab to apply a type of Gaussian highpass filter to the shuttle image.

Appendix 7a. A Fast Fourier Transform in One dimension

The following pseudo-code gives an algorithm for performing an FFT on each intermediate DFT and each final DFT in the row-column decomposition DFT. The real and imaginary data parts FR and FI are each real valued 1-dimensional arrays and are passed to the function **FFTBR**() for it to perform a 1-dimensional DFT on them. The function **FFTBR**() first calls the routine **Order**() to arrange the data by bit reversal so that the fast method of summing the DFT can be carried out. When **FFTBR**() has completed the FFT, it calls **WriteBack**() to write the DFT values into global variables for the main program (that does the row-column decomposition) to use. We do not specify **WriteBack**() here. The integer *key1* indicates DFT or IDFT and what stage of intermediate or final transform is desired.

```
/*-----------------------------------------------------------------*/
/* PSEUDO-CODE FOR THE RADIX-2 FFT                                 */
/*-----------------------------------------------------------------*/
/* Fast Fourier Transform with Bit Reversal                        */
/*  F[r] = FR[r] + jFI[r] is the function to be transformed with FFT*/
/*  FR[r] contains the real part of the data                       */
/*  FI[r] contains the imaginary part of the data                  */
/*  The total number of data points {F[r]} is 2^m = 1024           */
/*-----------------------------------------------------------------*/
void FFTBR(float FR[1024], float FI[1024], int m, int key1)
{ float theta, tR, tI, Pi = 3.141593;
  int rep, disp, n;
  int i, j, j2, k;
  /*----call function to rearrange data points via bit reversal------*/
  order(FR,FI,m);
  n = pow(2,m);
  /*-----------------------------------------------------------------*/
  for i = 1 to n do
     rep = pow(2,i);
     disp = rep/2;
     arg = 2.0*Pi/rep;
     for j = 1 to disp do
       theta = (j-1)*arg;
       C = cos(theta);
       S = sin(theta);
       k = j;
       while (k <= j)
            j2 = k + disp;
            tR =  C*FR[j2] + S*FI[j2];
            tI = -S*FR[j2] + C*FI[j2];
            FR[j2] = FR[k] - tR;
            FI[j2] = FI[k] - tI;
            FR[k] = FR[k] + tR;
            FI[k] = FI[k] + tI;
            k = k + rep;
```

```
  /*-------------------------------------------------------------------*/
  WriteBack();
  return;
}
/*-------------------------------------------------------------------*/
/*  Bit Reversal Function to Order the Data Points for Radix 2    */
/*-------------------------------------------------------------------*/
void order(float R, I, m)
{  int n, ndiv2, nminus1, j;
   float tR, tI;
   n = pow(2,n);
   ndiv2 = n/2;
   nminus1 = n - 1;
   j = 1;
   for i = 1 to nminus1 do
      if (i < j) then
          tR = R[j];
          R[j] = R[i];
          R[i] = tR;
          tI = I[j];
          I[j] = I[i];
          I[i] = tI;
      k = ndiv2;
      while (k<j) do
          j = j - k;
          k = k/2;
          j = j + k;
  return;
}
```