Chapter 3
Simple Computations Using Fourier Transforms

There are many useful computations such as correlations and convolutions that can be implemented using FTs. In fact, taking advantage of computationally efficient DFT techniques such as the FFT often executes much faster than more straightforward implementations. Subsequent chapters reuse these tools in an optical context. For example, convolution is used in Ch. 5 to simulate the effects of diffraction and aberrations on image quality, and structure functions are used in Ch. 9 to validate the statistics of turbulent phase screens.

Three of these tools, namely convolution, correlation, and structure functions, are closely related and have similar mathematical definitions. Furthermore, they are all written in terms of FTs in this chapter. However, their uses are quite different, and each common use is explained in the upcoming sections. These different uses cause the implementations of each to be quite different. For example, correlations and structure functions are usually performed on data that pass through an aperture. Consequently, their computations are modified to remove the effects of the aperture.

The last computation discussed in this chapter is the derivative. Like the other computations in this chapter, the method presented is based on FTs to allow for efficient computation. The method is then generalized to computing gradients of two-dimensional functions. While derivatives and gradients are not used again in later chapters, derivatives are discussed because some readers might want to compute derivatives for topics related to optical turbulence, like simulating the operation of wavefront sensors.

3.1 Convolution

We begin this discussion of FT-based computations with convolution for a couple of reasons. First, convolution plays a central role in linear-systems theory.[14] The output of a linear system is the convolution of the input signal with the system's impulse response. In the context of simulating optical wave propagation, the linear-systems formalism applies to coherent and incoherent imaging, analog optical image processing, and free-space propagation. The second reason we begin with

Listing 3.1 Code for performing a one-dimensional discrete convolution in MATLAB.

```
function C = myconv(A, B, delta)
% function C = myconv(A, B, delta)
    N = length(A);
    C = ift(ft(A, delta) .* ft(B, delta), 1/(N*delta));
```

Listing 3.2 MATLAB example of performing a discrete convolution with comparison to the analytic evaluation of the convolution integral.

```
% example_conv_rect_rect.m
N = 64;      % number of samples
L = 8;       % grid size [m]
delta = L / N;  % sample spacing [m]
F = 1/L;     % frequency-domain grid spacing [1/m]
x = (-N/2 : N/2-1) * delta;
w = 2;       % width of rectangle
A = rect(x/w);  B = A; % signal
C = myconv(A, B, delta); % perform digital convolution
% continuous convolution
C_cont = w*tri(x/w);
```

convolution is that its practical implementation is the simplest of all the FT-based computations discussed in this chapter.

Throughout this book, we use the symbol $\otimes$ to denote the convolution operation defined by

$$C_{fg}(x) = f(x) \otimes g(x) = \int_{-\infty}^{\infty} f(x')\, g(x - x')\, dx'. \tag{3.1}$$

Often, the two functions being convolved have very different characteristics. Particularly when convolution is used in the context of linear systems, one function is a signal and the other is an impulse response. In the time domain, the impulse response ordinarily has a short duration, while the signal usually has a much longer duration. In the spatial domain, like for optical imaging, the impulse response ordinarily has a narrow spatial extent, while the signal usually occupies a comparatively larger area. The act of convolution smears the input slightly so that the duration or extent of the output is slightly larger than that of the input signal. Often, this spreading effect requires that the inputs to numerical convolution be padded with zeros at the edges of the grid to avoid artifacts of undesired periodicity.[10] In this book, the signals involved are usually already padded with zeros.

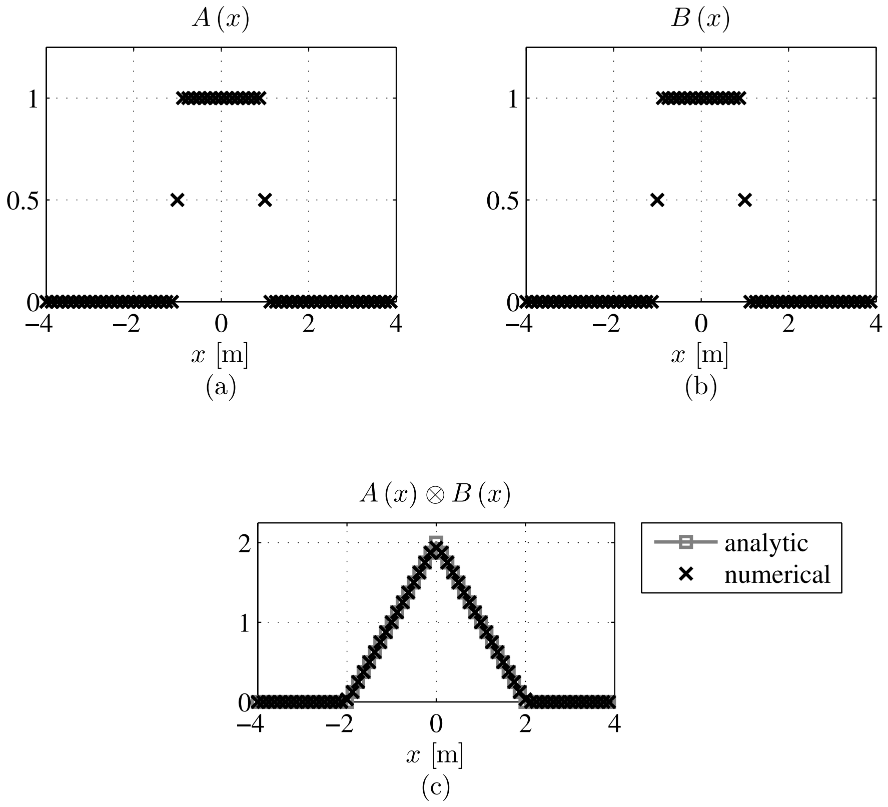

Figure 3.1 A `rect` function convolved with itself. Plots (a) and (b) show the sampled functions that are input into the convolution algorithm. Plot (c) shows the result from the DFT-based computation and the analytic result.

The implementation begins with using the convolution theorem, which is mathematically stated as[5]

$$\mathcal{F}\left[f(x) \otimes g(x)\right] = \mathcal{F}\left[f(x)\right]\mathcal{F}\left[g(x)\right]. \tag{3.2}$$

The beneficial mathematical property here is that the often-difficult-to-compute convolution integral is equivalent to simple multiplication in the frequency domain. Then, by inverse Fourier transforming both sides, Eq. (3.1) can be rewritten as

$$f(x) \otimes g(x) = \mathcal{F}^{-1}\left\{\mathcal{F}\left[f(x)\right]\mathcal{F}\left[g(x)\right]\right\}. \tag{3.3}$$

The computational benefit of the convolution theorem is that, when taking advantage of the FFT algorithm, Eq. (3.3) is typically much faster to evaluate numerically than Eq. (3.1) as a double sum. Accordingly, Listing 3.1 gives MATLAB code for the function `myconv` that takes advantage of this property.

Listing 3.2 gives example use of `myconv`, and the results are plotted in Fig. 3.1. In the example, the function $\mathrm{rect}\,(x/w)$ is convolved with itself, and the analytic

Listing 3.3 Code for performing a two-dimensional discrete convolution in MATLAB.

```
function C = myconv2(A, B, delta)
% function C = myconv2(A, B, delta)
    N = size(A, 1);
    C = ift2(ft2(A, delta) .* ft2(B, delta), 1/(N*delta));
```

Listing 3.4 MATLAB example of performing a two-dimensional discrete convolution. A rectangle function is convolved with itself.

```
% example_conv2_rect_rect.m

N = 256;      % number of samples
L = 16;       % grid size [m]
delta = L / N;  % sample spacing [m]
F = 1/L;      % frequency-domain grid spacing [1/m]
x = (-N/2 : N/2-1) * delta;
[x y] = meshgrid(x);
w = 2;        % width of rectangle
A = rect(x/w) .* rect(y/w);  % signal
B = rect(x/w) .* rect(y/w);  % signal
C = myconv2(A, B, delta); % perform digital convolution
% continuous convolution
C_cont = w^2*tri(x/w) .* tri(y/w);
```

result is $w\ \mathrm{tri}\,(x/w)$. The code uses $w = 2$, a grid size of 8 m, and 64 samples. Clearly, the close agreement between the analytic and numerical results in the figure shows that the computer code is operating properly, and `myconv` uses the proper scaling.

Two-dimensional convolution is quite important in optics. Particularly, to compute a diffraction image, one must convolve the geometric image with the imaging system's two-dimensional spatial impulse response. This optical application of two-dimensional convolution is discussed further in Sec. 5.2. Generalizing Listing 3.1 to perform convolution in two dimensions is quite straightforward. In the computer code, the calls to the functions `ft` and `ift` are replaced by `ft2` and `ift2`, respectively. The MATLAB code is given in Listing 3.3 for the function `myconv2`.

Listing 3.4 gives an example of a two-dimensional convolution. In the example, the function $A(x, y) = \mathrm{rect}\,(x/w)\,\mathrm{rect}\,(y/w)$ is convolved with itself. In this case, $w = 2.0$ m, the grid size is 16 m, and there are 256 grid points per side. Figure 3.2 shows the analytic and numerical results. Note the close agreement between them.

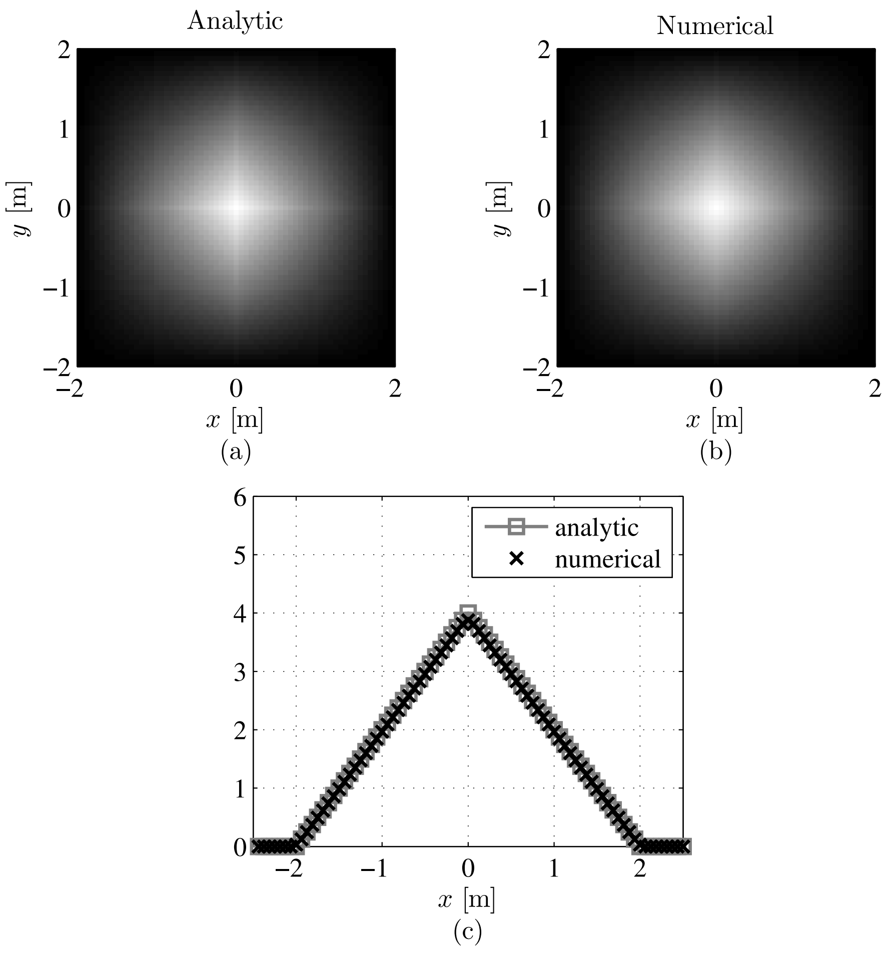

Figure 3.2 A rectangle function convolved with itself. Plot (a) shows the analytic result, while plot (b) shows the numerical result. Plot (c) shows a comparison of the $y = 0$ slices of the analytic and numerical results.

3.2 Correlation

Correlation functions are mathematically very similar to convolutions. Because of the differences in implementation though, we begin the discussion of correlation in two dimensions. Let us define the two-dimensional correlation integral as

$$\Gamma_{fg}\left(\Delta\mathbf{r}\right) = f\left(\mathbf{r}\right) \star g\left(\mathbf{r}\right) = \int_{-\infty}^{\infty} f\left(\mathbf{r}\right) g^{*}\left(\mathbf{r} - \Delta\mathbf{r}\right) \, d\mathbf{r}, \tag{3.4}$$

where the $\star$ notation has also been used to denote correlation. Comparing this to Eq. (3.1), we can see that the only mathematical differences between convolution and correlation are a the complex conjugate on $g\left(x\right)$ and a minus sign on its argument. There is even a correlation theorem similar to the convolution theorem that provides similar mathematical and computational benefits. Inverse Fourier transforming both sides, Eq. (3.4) can be rewritten as

$$f(x) \star g(x) = \mathcal{F}^{-1}\left\{\mathcal{F}[f(x)]\,\mathcal{F}[g(x)]^*\right\}. \tag{3.5}$$

Despite the mathematical similarities between convolution and correlation, their usages are often quite different. Usually, correlation is often used to determine the similarity between two signals. Accordingly, the two input signals $f(x)$ and $g(x)$ often have relatively similar characteristics, whereas the two inputs to convolution are usually quite different from each other. The separation $\Delta\mathbf{r}$ at which the correlation peaks may tell the distance between features in the two signals. When the two inputs are the same, i.e., $f(x) = g(x)$, it is an auto-correlation. The width of the auto-correlation's peak may reveal information about the signal's variations.

A particular application of auto-correlation in this book is analysis of processes and fields that fluctuate randomly. At any given time or point in space, a random quantity may be specified by a probability density function (PDF). To describe the temporal and spatial variations, often the mean auto-correlation is used. As a relevant example, sometimes optical sources themselves fluctuate randomly. This is the realm of statistical optics.[6] In Ch. 9, the optical field fluctuates randomly due to atmospheric turbulence even if the source field has no fluctuations. The correlation properties of the field contain information about the cause of the fluctuations.[15] For example, Ch. 9 presents theoretical expressions for the mean auto-correlation of optical fields that have propagated through atmospheric turbulence in terms of the turbulent coherence diameter. The theoretical expression is compared to the numerical auto-correlation of simulated random draws of turbulence-degraded fields. The favorable comparison provides a means of verifying proper operation of the turbulent simulation.

The mean correlation is the ensemble average of many independent and identically distributed realizations of Eq. (3.4). The very basic implementation of Eq. (3.4) is very similar to that of convolution. However, optical data are often collected through a circular or annular aperture, while we must represent two-dimensional data in a rectangular array of numbers. Sometimes we wish to isolate the correlation of the data within the pupil and exclude effects of the pupil when we compute quantities like auto-correlation. For example, we may be observing a field that is partially coherent. To relate observation-plane measurements to properties of the source, we need to compute the auto-correlation of the pupil-plane field, not the combined pupil-plane field and aperture. The basic approach, like the one used for convolution, would capture the combined effects of the signal and aperture. To remove the effects of the aperture, the implementation presented here is more complicated than that for convolution.

Let the optical field be $u(\mathbf{r})$ and the shape of the pupil be represented by $w(\mathbf{r})$. The function $w(\mathbf{r})$ is a "window" that is usually equal to one inside the optical aperture and zero outside, written formally as

$$w(\mathbf{r}) = \begin{cases} 1 & \mathbf{r} \text{ inside pupil} \\ 0 & \mathbf{r} \text{ outside pupil.} \end{cases} \tag{3.6}$$

This allows us to use only the region of the field that is transmitted through the aperture. The data that our sensors collect through the aperture is

$$u'(\mathbf{r}) = u(\mathbf{r})\, w(\mathbf{r}). \tag{3.7}$$

If we compute the auto-correlation of the windowed data, we get

$$\Gamma_{u'u'}(\Delta\mathbf{r}) = u'(\mathbf{r}) \star u'(\mathbf{r}) = \int_{-\infty}^{\infty} u(\mathbf{r})\, u^*(\mathbf{r} - \Delta\mathbf{r})\, w(\mathbf{r})\, w^*(\mathbf{r} - \Delta\mathbf{r})\; d\mathbf{r}. \tag{3.8}$$

The integrand is equal to $u(\mathbf{r})\, u^*(\mathbf{r} - \Delta\mathbf{r})$ wherever $w(\mathbf{r})\, w^*(\mathbf{r} - \Delta\mathbf{r})$ is nonzero. Let us denote this region as $R(\mathbf{r}, \Delta\mathbf{r})$. Then we can rewrite the integral as

$$\Gamma_{u'u'}(\Delta\mathbf{r}) = \int_{R(\mathbf{r},\Delta\mathbf{r})} u(\mathbf{r})\, u^*(\mathbf{r} - \Delta\mathbf{r})\; d\mathbf{r}. \tag{3.9}$$

Now we compute the area of $R(\mathbf{r}, \Delta\mathbf{r})$ as

$$A(\Delta\mathbf{r}) = \int R(\mathbf{r}, \Delta\mathbf{r})\; d\mathbf{r} = \Gamma_{ww}(\Delta\mathbf{r}). \tag{3.10}$$

Listing 3.5 MATLAB code for performing a two-dimensional discrete correlation removing effects of the aperture.

```
function c = corr2_ft(u1, u2, mask, delta)
% function c = corr2_ft(u1, u2, mask, delta)

    N = size(u1, 1);
    c = zeros(N);
    delta_f = 1/(N*delta);  % frequency grid spacing [m]

    U1 = ft2(u1 .* mask, delta);    % DFTs of signals
    U2 = ft2(u2 .* mask, delta);
    U12corr = ift2(conj(U1) .* U2, delta_f);

    maskcorr = ift2(abs(ft2(mask, delta)).^2, delta_f) ...
        * delta^2;
    idx = logical(maskcorr);
    c(idx) =  U12corr(idx) ./ maskcorr(idx) .* mask(idx);
```

Listing 3.6 MATLAB example of performing a two-dimensional discrete auto-correlation. A rectangle function is correlated with itself.

```
% example_corr2_rect_rect.m

N = 256;      % number of samples
L = 16;       % grid size [m]
delta = L / N;  % sample spacing [m]
F = 1/L;     % frequency-domain grid spacing [1/m]
x = (-N/2 : N/2-1) * delta;
[x y] = meshgrid(x);
w = 2;       % width of rectangle
A = rect(x/w) .* rect(y/w);  % signal
mask = ones(N);
% perform digital correlation
C = corr2_ft(A, A, mask, delta);
% analytic correlation
C_cont = w^2*tri(x/w) .* tri(y/w);
```

If we know that the average of $\Gamma_{uu}(\Delta\mathbf{r})$ is truly independent of $\mathbf{r}$, $u(\mathbf{r})$ is called wide-sense stationary, and we can write

$$\langle \Gamma_{u'u'}(\Delta\mathbf{r}) \rangle = A(\Delta\mathbf{r}) \langle \Gamma_{uu}(\Delta\mathbf{r}) \rangle . \tag{3.11}$$

To compute Eq. (3.11) efficiently, we can use FTs. Using the auto-correlation theorem, we can define

$$W(\mathbf{f}) = \mathcal{F}\{w(\mathbf{r})\} \tag{3.12}$$

$$U'(\mathbf{f}) = \mathcal{F}\{u'(\mathbf{r})\}, \tag{3.13}$$

and then write

$$\langle \Gamma_{uu}(\Delta\mathbf{r}) \rangle = \frac{\left\langle \mathcal{F}^{-1}\left\{|U'(\mathbf{f})|^2\right\}\right\rangle}{\mathcal{F}^{-1}\left\{|W(\mathbf{f})|^2\right\}} \tag{3.14}$$

Equation (3.14) can be generalized to handle cross correlations between two fields $u_1(\mathbf{r})$ and $u_2(\mathbf{r})$. MATLAB code for computing this cross correlation using a generalized version of Eq. (3.14) is given in Listing 3.5.

Listing 3.6 gives an example of a two-dimensional auto-correlation. In the example, the function $A(x,y) = \mathrm{rect}(x/w)\,\mathrm{rect}(y/w)$ is correlated with itself. In this case, $w = 2.0$ m, the grid size is 16 m, and there are 256 grid points per side. The mask value is one over the entire grid because there is no aperture. Because the function is symmetric about the x and y axes, the result is the same as the convolution example above. Figure 3.3 shows the analytic and numerical results. Once again, note the close agreement between them. An example of computing mean auto-correlation with an aperture mask is given in Sec. 9.5.5.

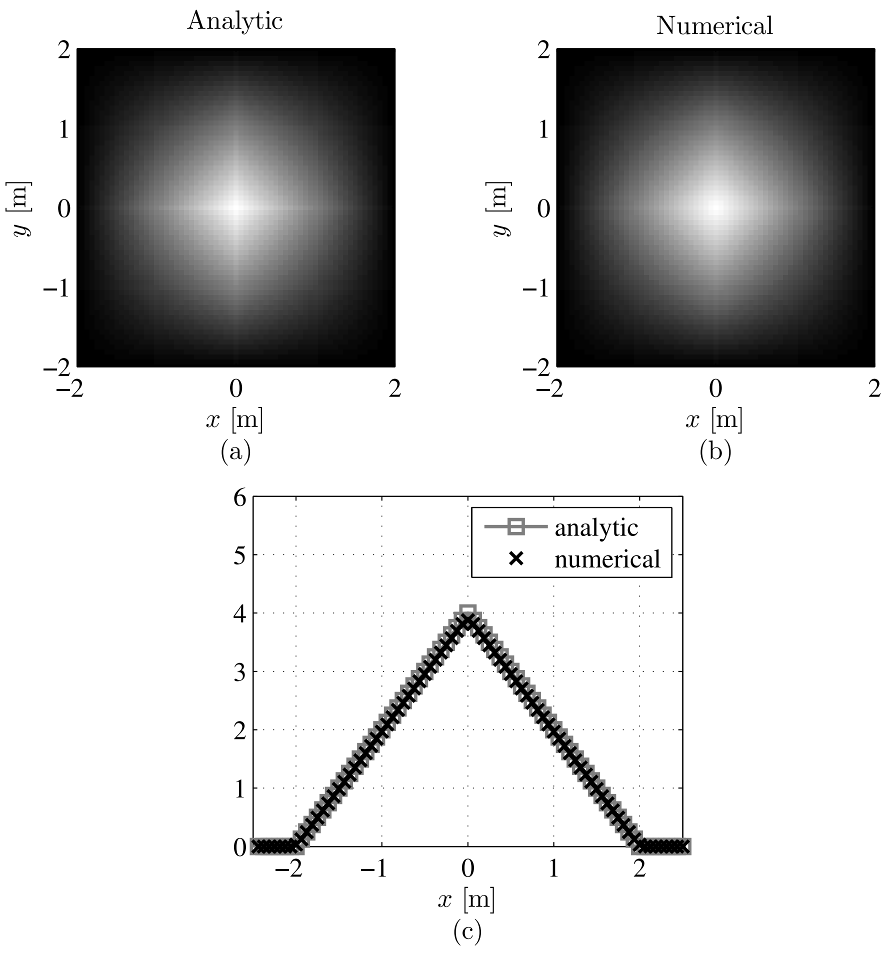

Figure 3.3 A rectangle function correlated with itself. Plot (a) shows the analytic result, while plot (b) shows the numerical result. Plot (c) shows a comparison of the $y = 0$ slices of the analytic and numerical results.

3.3 Structure Functions

Structure functions are another statistical measure of random fields, and they are closely related to auto-correlations. They are particularly appropriate for studying random fields that are not wide-sense stationary. See Ref. 6 for a detailed discussion of statistical stationarity. Structure functions are often used in optical turbulence to describe the behavior of quantities like refractive index, phase, and log-amplitude. The structure function of one realization of a random field $g(\mathbf{r})$ is defined as

$$D_g(\Delta\mathbf{r}) = \int [g(\mathbf{r}) - g(\mathbf{r} + \Delta\mathbf{r})]^2 \, d\mathbf{r}. \tag{3.15}$$

Like with correlations, a statistical structure function is an ensemble average over Eq. (3.15). It can be shown that when the random field is statistically isotropic, the

Listing 3.7 MATLAB code for performing a two-dimensional discrete structure function removing effects of the aperture.

```
function D = str_fcn2_ft(ph, mask, delta)
% function D = str_fcn2_ft(ph, mask, delta)

    N = size(ph, 1);
    ph = ph .* mask;

    P = ft2(ph, delta);
    S = ft2(ph.^2, delta);
    W = ft2(mask, delta);
    delta_f = 1/(N*delta);
    w2 = ift2(W.*conj(W), delta_f);

    D = 2 * ift2(real(S.*conj(W)) - abs(P).^2, ...
        delta_f) ./ w2 .* mask;
```

Listing 3.8 MATLAB example of performing a two-dimensional structure function of a rectangle function.

```
% example_strfcn2_rect.m

N = 256;      % number of samples
L = 16;       % grid size [m]
delta = L / N;  % sample spacing [m]
F = 1/L;     % frequency-domain grid spacing [1/m]
x = (-N/2 : N/2-1) * delta;
[x y] = meshgrid(x);
w = 2;       % width of rectangle
A = rect(x/w) .* rect(y/w);  % signal
mask = ones(N);
% perform digital structure function
C = str_fcn2_ft(A, mask, delta) / delta^2;
% continuous structure function
C_cont = 2 * w^2 * (1 - tri(x/w) .* tri(y/w));
```

mean structure function and auto-correlation are related by

$$D_g\left(\Delta\mathbf{r}\right) = 2\left[\Gamma_{gg}\left(\mathbf{0}\right) - \Gamma_{gg}\left(\Delta\mathbf{r}\right)\right]. \tag{3.16}$$

Also similar to correlations, we often must compute the structure function of windowed data. Using windowed data u' yields

$$\left\langle D_{u'}\left(\Delta\mathbf{r}\right)\right\rangle = A\left(\Delta\mathbf{r}\right)\left\langle D_u\left(\Delta\mathbf{r}\right)\right\rangle. \tag{3.17}$$

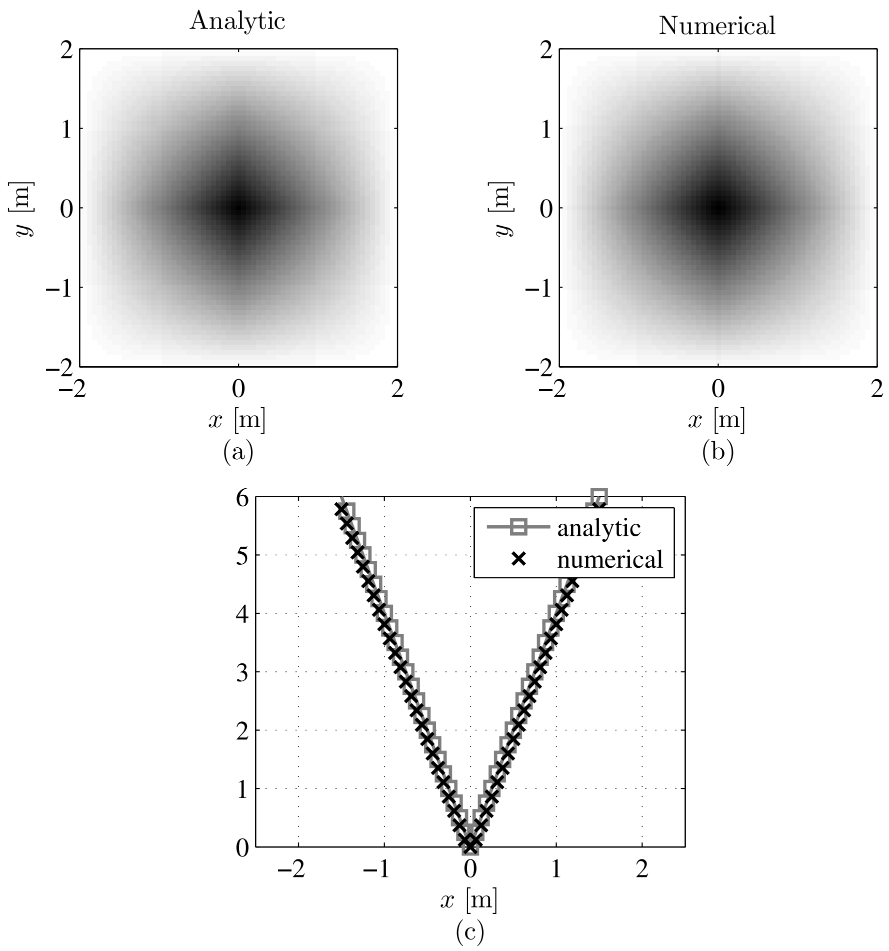

Figure 3.4 Structure function of a rectangle function. Plot (a) shows the analytic result, while plot (b) shows the numerical result. Plot (c) shows a comparison of the $y = 0$ slices of the analytic and numerical results.

Then we must focus on $D_{u'}(\Delta\mathbf{r})$. Multiplying out the terms inside the integral, we get

$$\begin{aligned} D_{u'}(\Delta\mathbf{r}) = \int \big[& u'^2(\mathbf{r})\, w(\mathbf{r}+\Delta\mathbf{r}) \\ & -2u'(\mathbf{r})\, u'(\mathbf{r}+\Delta\mathbf{r}) + u'^2(\mathbf{r}+\Delta\mathbf{r})\, w(\mathbf{r})\big]\, d\mathbf{r}. \end{aligned} \tag{3.18}$$

Now we can replace each term by its Fourier integral representation, which allows for an efficient computation when we use FFTs. To do so, first let us define

$$W(\mathbf{f}) = \mathcal{F}\{w(\mathbf{r})\} \tag{3.19}$$

$$U'(\mathbf{f}) = \mathcal{F}\{u'(\mathbf{r})\} \tag{3.20}$$

$$S(\mathbf{f}) = \mathcal{F}\left\{\left[u'(\mathbf{r})\right]^2\right\}. \tag{3.21}$$

Also, note that $W(\mathbf{f}) = W^*(\mathbf{f})$ because $w(\mathbf{r})$ is real. Then, with these definitions and properties, we can write

$$\begin{aligned} D_{u'}(\Delta\mathbf{r}) = \int \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} & \{S(\mathbf{f}_1) W^*(\mathbf{f}_2) \\ & + S^*(\mathbf{f}_2) W(\mathbf{f}_1) - 2U'(\mathbf{f}_1) \left[U'(\mathbf{f}_2)\right]^* \} \\ & \times e^{i2\pi(\mathbf{f}_1+\mathbf{f}_2)\cdot\mathbf{r}} e^{-i2\pi\mathbf{f}_2\cdot\Delta\mathbf{r}}\, d\mathbf{f}_1\, d\mathbf{f}_2\, d\mathbf{r}. \end{aligned} \tag{3.22}$$

Now, evaluating the $\mathbf{r}$ integral and then the $\mathbf{f}_2$ integral yields

$$\begin{aligned} D_{u'}(\Delta\mathbf{r}) = \int_{-\infty}^{\infty} & \{S(\mathbf{f}_1) W^*(\mathbf{f}_1) \\ & + S^*(\mathbf{f}_1) W(\mathbf{f}_1) - 2U'(\mathbf{f}_1) \left[U'(\mathbf{f}_1)\right]^* \} e^{-i2\pi\mathbf{f}_1\cdot\Delta\mathbf{r}}\, d\mathbf{f}_1 \end{aligned} \tag{3.23}$$

$$= 2 \int_{-\infty}^{\infty} \left\{ \mathrm{Re}\left[S(\mathbf{f}_1) W^*(\mathbf{f}_1)\right] - \left|U'(\mathbf{f}_1)\right|^2 \right\} e^{-i2\pi\mathbf{f}_1\cdot\Delta\mathbf{r}}\, d\mathbf{f}_1 \tag{3.24}$$

$$= 2\,\mathcal{F}\left\{ \mathrm{Re}\left[S(\mathbf{f}_1) W^*(\mathbf{f}_1)\right] - \left|U'(\mathbf{f}_1)\right|^2 \right\}. \tag{3.25}$$

Listing 3.7 implements Eqs. (3.17) and (3.25) to compute a structure function through the use of FTs.

Listing 3.8 gives an example of computing a two-dimensional structure function. The example computes the structure function of the two-dimensional signal $A(x, y) = \mathrm{rect}(x/w)\,\mathrm{rect}(y/w)$. As in the previous example, $w = 2.0$ m, the grid size is 16 m, and there are 256 grid points per side. The mask value is one over the entire grid. To compute the analytic result, we can take advantage of the relationship between structure functions and auto-correlations as given by Eq. (3.16). This example uses the same signal as the previous example of correlation, so we apply this relationship to compute the analytic structure function from the analytic auto-correlation. Figure 3.4 shows the analytic and numerical results. Once again, note the close agreement between them. Sections 9.3 and 9.5.5 give examples of computing the mean structure function of a random field.

3.4 Derivatives

This chapter concludes with one last computation based on DFTs, namely derivatives. Derivatives are not used again in this book, but readers who simulate the operation of devices such as wavefront sensors may find this section useful. Several useful devices such as the Shack-Hartmann and shearing-interferometer wavefront sensors can measure the gradient of optical phase.

Listing 3.9 MATLAB code for performing a one-dimensional discrete derivative.

```
function der = derivative_ft(g, delta, n)
% function der = derivative_ft(g, delta, n)

    N = length(g);   % number of samples in g
    % grid spacing in the frequency domain
    F = 1/(N*delta);
    f_X = (-N/2 : N/2-1) * F;   % frequency values

    der = ift((i*2*pi*f_X).^n .* ft(g, delta), F);
```

Listing 3.10 MATLAB example of performing a one-dimensional discrete derivative. The corresponding plots are shown in Fig. 3.5.

```
% example_derivative_ft.m

N = 64;      % number of samples
L = 6;       % grid size [m]
delta = L/N;    % grid spacing [m]
x = (-N/2 : N/2-1) * delta;
w = 3;       % size of window (or region of interest) [m]
window = rect(x/w); % window function [m]
g = x.^5 .* window; % function
% discrete derivatives
gp_samp = real(derivative_ft(g, delta, 1)) .* window;
gpp_samp = real(derivative_ft(g, delta, 2)) .* window;
% analytic derivatives
gp = 5*x.^4 .* window;
gpp = 20*x.^3 .* window;
```

By taking the n^{th}-order derivative of Eq. (2.1) with respect to x and moving the derivative operator inside the FT, it is easy to show that

$$\mathcal{F}\left\{\frac{d^n}{dx^n} g(x)\right\} = (i2\pi f_x)^n \, \mathcal{F}\{g(x)\}\,. \tag{3.26}$$

We can take advantage of this relationship to compute $dg(x)/dx$ by taking the inverse FT of both sides. This is the principle behind the MATLAB code shown in Listing 3.9, which gives the `derivative_ft` function.

Listing 3.10 shows example usage of the `derivative_ft` function. In this example, $g(x) = x^5$. The first two derivatives of this function are computed, and the results are shown in Fig. 3.5 along with the analytic results for comparison.

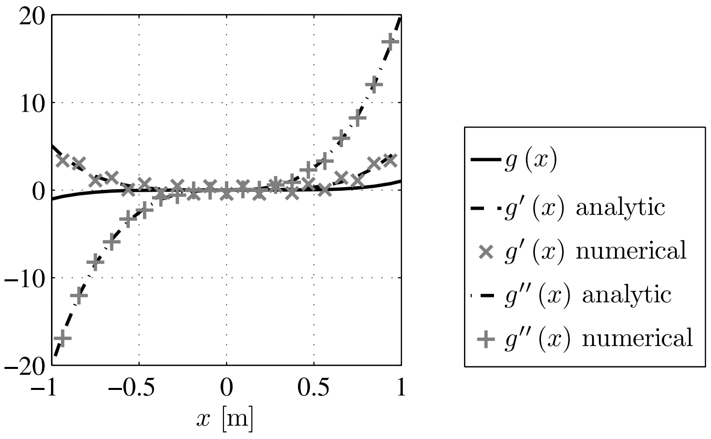

Figure 3.5 Plot of the function $g(x) = x^5$ and its first two derivatives computed numerically with the analytic expressions included for comparison.

Listing 3.11 MATLAB code for computing the discrete gradient of a function using FTs.

```
function [gx gy] = gradient_ft(g, delta)
% function [gx gy] = gradient_ft(g, delta)

    N =size(g, 1);    % number of samples per side in g
    % grid spacing in the frequency domain
    F = 1/(N*delta);
    fX = (-N/2 : N/2-1) * F;    % frequency values
    [fX fY] = meshgrid(fX);
    gx = ift2(i*2*pi*fX .* ft2(g, delta), F);
    gy = ift2(i*2*pi*fY .* ft2(g, delta), F);
```

Note that a window function is used to limit the extent of the signal and mitigate aliasing of the computed spectrum because $g(x)$ and its first few derivatives are not bandlimited functions. Using the window function improves the accuracy of the numerical derivative.

Now, generalizing Eq. (3.26) to two dimensions, we can compute the x and y partial derivatives of a two-dimensional scalar function $g(x, y)$. Using steps similar to those that produced Eq. (3.26), it is easy to show that

$$\mathcal{F}\left\{\frac{\partial^n}{\partial x^n} g(x,y)\right\} = (i2\pi f_x)^n \mathcal{F}\{g(x,y)\} \tag{3.27}$$

$$\mathcal{F}\left\{\frac{\partial^n}{\partial y^n} g(x,y)\right\} = (i2\pi f_y)^n \mathcal{F}\{g(x,y)\}. \tag{3.28}$$

Then the gradient of the function uses the $n = 1$ case so that

$$\nabla g(x,y) = \mathcal{F}^{-1}\{i2\pi f_x \mathcal{F}\{g(x,y)\}\}\,\hat{\mathbf{i}} + \mathcal{F}^{-1}\{i2\pi f_y \mathcal{F}\{g(x,y)\}\}\,\hat{\mathbf{j}}. \tag{3.29}$$

Listing 3.12 MATLAB example of performing a discrete gradient of a two-dimensional scalar function. The corresponding plots are shown in Fig. 3.6.

```
% example_gradient_ft.m
N = 64;       % number of samples
L = 6;        % grid size [m]
delta = L/N;     % grid spacing [m]
x = (-N/2 : N/2-1) * delta;
[x y] = meshgrid(x);
g = exp(-(x.^2 + y.^2));
% computed derivatives
[gx_samp gy_samp] = gradient_ft(g, delta);
gx_samp = real(gx_samp);
gy_samp = real(gy_samp);
% analytic derivatives
gx = -2*x.*exp(-(x.^2+y.^2));
gy = -2*y.*exp(-(x.^2+y.^2));
```

This is easily implemented in MATLAB code, as shown in Listing 3.11, which gives the `gradient_ft` function.

Listing 3.12 shows example usage of the `gradient_ft` function. In this example,

$$g(x, y) = \exp\left[-\left(x^2 + y^2\right)\right], \tag{3.30}$$

and the analytic gradient is given by

$$\nabla g(x, y) = -2\exp\left[-\left(x^2 + y^2\right)\right]\left(x\hat{\mathbf{i}} + y\hat{\mathbf{j}}\right). \tag{3.31}$$

The numerical gradient of this function is computed in the listing, and the results are shown in Fig. 3.6 along with the analytic results for comparison. This time, a window function is not needed because $g(x, y)$ is nearly bandlimited. The quiver plots shown in Figs. 3.6(b) and (c) show the same trends. While it is not exactly evident in the plots, the analytic and numerical gradients are in very close agreement.

3.5 Problems

1. Perform a discrete convolution of the signal function $\text{rect}(x + a) + \text{tri}(x)$ with the impulse response $\exp\left[-(\pi/3)x^2\right]$ for several values of a. At which value of a are the two features in the signal just barely resolved? You do not need to use a formal criterion for resolution, just visually inspect plots of the convolution results.

2. Perform a discrete convolution of the signal $\text{circ}\left[a\left(x^2 + y^2\right)^{1/2}\right]$ with itself for $a = 1$ and $a = 10$. Show the two-dimensional surface plot of the numer-

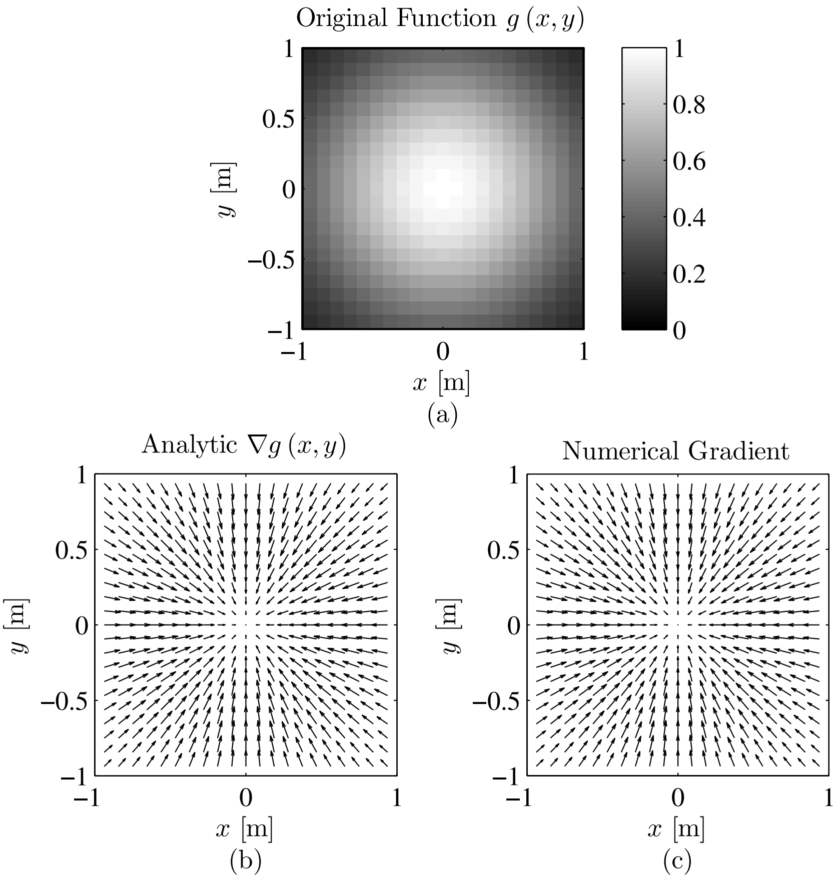

Figure 3.6 Plot of the function $g(x,y) = \exp\left[-\left(x^2+y^2\right)\right]$ and its gradient computed numerically with the analytic expressions included for comparison.

ical and analytic results and a plot of the $y = 0$ slice of the numerical and analytic results.

3. Numerically compute the first derivative of the function $g(x) = J_2(x)$, where $J_2(x)$ is a Bessel function of the first kind, order 2. Plot the result and show agreement with the analytic answer in the region $-1 \leq x \leq 1$.