Image Processing
and
Fourier Analysis

A.G. de Wijn

November 14, 2010

Contents

Chapter 1

Image Processing

1.1 Flatfields and Darkfields

CCD cameras used to record images or spectra always have pixel-dependent intensity registration. There are always dust particles on the detector, some pixels might be more sensitive than others, etc. Furthermore, there is always some noise and background intensity. Observers always make *flatfields* and *darkfields* to be able to correct for these errors. For now, we'll focus on correcting images. Spectra can also be corrected, but it is more difficult.

A flatfield is recorded with the CCD illuminated by a source that does not have an intensity variation over the image. Typical methods are defocussing the telescope and/or moving the telescope around quickly while exposing. A darkfield is recorded with the CCD not illuminated by the source. Some telescopes interrupt the beam well before the CCDs, or point the telescope away from the source.

If we have raw data R, a flatfield F, and a darkfield D, we can find the corrected data C,

$$C = \frac{R - D}{F - D}\,. \tag{1.1}$$

Exercise 1.1. Explain equation 1.1.

Exercise 1.2. Get the raw, flat, and dark images. These data were recorded with the Dutch Open Telescope (DOT). Display the raw image on your screen with the IDL procedure `tvscl`. Now correct the raw image, and display the result. Use the IDL procedures `bytscl` and `flick` to blink between the raw and the corrected image. Make a nice PostScript figure of the image, with annotated axes (the size of a pixel is $0.071''$ square).

1.2 Histogram Analysis

A *histogram* is a plot in which the number of data points with some property is plotted against that property.

Exercise 1.3. Use the IDL procedure `histogram` to produce a histogram of intensity of a G-band image. Study the IDL help to understand what parameters of the histogram you can tune, and choose appropriate values. The `plot` routine can be used in histogram mode by setting the `psym` keyword to the correct value.

In order to increase the contrast of the image, we perform a *histogram equalization*. The procedure attempts to produce a flat histogram, so that all intensities appear in equal amounts.

Exercise 1.4. IDL has two histogram equalization functions. Use the simple equalization to display an equalized image. Compare the histogram of the equalized image with the original histogram. How does the more sophisticated equalization function differ from the simple one? Compare the results.

1.3 Scatter Plots

A good way to compare images pixel-by-pixel is by making a so-called *scatter plot*.

Exercise 1.5. Plot the intensity of the first G-band image against the intensity of the Ca II H image. Use the `psym` keyword to plot a dot for each pixel pair. What does this figure tell you about the images?

1.4 Making Dopplergrams from Line Scans

If we have a narrow filter, we can "scan" through a line to produce a series of images that sample the profile of the line. From these images we can, e.g., determine Doppler information. Take two images d_1 and d_2 from the scan on either side of the line center. If the line is Dopplershifted, the observed intensity on one side will be higher than on the other. In the most general case, we have

$$v_{\mathrm{Doppler}} = \mathcal{C}\left(\frac{d_1 - d_2}{d_1 + d_2}\right) , \tag{1.2}$$

where $\mathcal{C}$ is a calibration function that relates the relative intensity difference to the Doppler velocity. We analyze two DOT Hα images here. These images were recorded at 350 mÅ from the line center, one in the blue wing, the other in the red. The filter width is 250 mÅ.

Exercise 1.6. Restore the Hα line profile from file and make a plot. Determine the center of the line λ_{cen}. Next, make two Gaussian profiles with a FWHM of 250 mÅ at $\lambda_{\mathrm{cen}} + 350$ mÅ and -350 mÅ. The measured intensity is just the integral over the product of the line profile and the Gaussian. Compute the relative intensity difference. To which Doppler speed does it correspond?

Exercise 1.7. Determine the calibration function $\mathcal{C}$ by repeatedly shifting the Hα atlas line profile and sampling the result by the Gaussians. You could use the IDL function `convol` to do it efficiently. Select only the central region $-15 \leq v_{\mathrm{Doppler}} \leq 20$ km/s.

Exercise 1.8. For each pixel in the DOT data, determine the relative intensity difference. Apply the calibration to determine a map of the Doppler velocities from the two images. You can do this in one statment with the IDL `interpol` function! Make a nice figure from the result with a color bar. Think carefully about how you scale the intensity of the image!

Chapter 2

Fourier Theory

This chapter is a brief overview of the theory behind Fourier transforms. Some places to find more background information are Brault and White (Astron. Astrophys. 13, 169–189) and Mathworld (`http://mathworld.wolfram.com/`).

We use the mathematical variable x, which is commonly associated with physical lengths. The theory obviously also applies to other physical variables such as time (commonly t). In this case, wave number k is often replaced by frequency f.

2.1 Fourier Series and Transforms

The functions $\mathrm{e}^{\mathrm{i}nx}$ with $n, m \in \mathbb{Z}$ form an orthonormal base on $[-\pi, \pi]$,

$$\int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{i}nx}\, \mathrm{e}^{-\mathrm{i}mx}\, \mathrm{d}x = 2\pi\delta_{mn}\,.$$

If $f(x)$ is a complex-valued function periodic on $[-\pi, \pi]$, we can write

$$f(x) = \sum_{n=-\infty}^{\infty} A_n\, \mathrm{e}^{\mathrm{i}nx}\,, \tag{2.1}$$

with

$$A_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\, \mathrm{e}^{-\mathrm{i}nx}\, \mathrm{d}x\,. \tag{2.2}$$

Equation 2.1 is called the *Fourier series expansion* of the function $f(x)$.

We can extend equations 2.1 and 2.2 to a complex-valued function periodic on any interval $[-L/2, L/2]$ by rescaling,

$$f(x) = \sum_{n=-\infty}^{\infty} A'_n\, \mathrm{e}^{2\pi\mathrm{i}\, nx/L}\,,$$

$$A'_n = \frac{1}{L} \int_{-L/2}^{L/2} f(x)\, \mathrm{e}^{-2\pi\mathrm{i}\, nx/L}\, \mathrm{d}x\,.$$

We now take the limit $L \to \infty$, substitute the discrete A'_n by the continuous $F(k)\,\mathrm{d}k$ while letting $n/L \to k$, and replace the sum with an integral. We get the *forward Fourier transform*

$$F(k) = \mathcal{F}_x[f(x)](k) = \int_{-\infty}^{\infty} f(x)\,\mathrm{e}^{-2\pi\mathrm{i}\,kx}\,\mathrm{d}x\,, \tag{2.3}$$

and the *backward Fourier transform*

$$f(x) = \mathcal{F}_k^{-1}[F(k)](x) = \int_{-\infty}^{\infty} F(k)\,\mathrm{e}^{2\pi\mathrm{i}\,kx}\,\mathrm{d}k\,. \tag{2.4}$$

The variable k has the inverse dimension of L. In the case that L measures distance, k is often called the *wave number*.

A complex number c can be written as $c = l\,\mathrm{e}^{\mathrm{i}\phi}$, where $l = |c|$ is the absolute value of c, and ϕ is the angle the vector c makes with the real axis. If we now look at a single k, the contribution to the integral in equation 2.4 is

$$F(k)\,\mathrm{e}^{2\pi\mathrm{i}\,kx} = |F(k)|\,\mathrm{e}^{\mathrm{i}(2\pi\,kx+\phi(k))}\,.$$

This describes a circle of radius $|F(k)|$ in complex space, where the position on the circle is determined by the value of x. The value of $\phi(k)$ determines the starting angle.

The value $|F(k)|$ is called the Fourier *amplitude* at k. The square of the amplitude, $|F(k)|^2 = F^*(k)\,F(k)$, is called the *power* at k. Finally, the angle $\phi(k)$ is called the *phase* at k. If we plot a property as a function of k, we speak of a *spectrum*.

2.2 Conversion

The Fourier series of $f(x)$ converges to the function $\bar{f}(x) = \frac{1}{2}\left(\lim_{\xi\uparrow x} f(\xi) + \lim_{\xi\downarrow x} f(\xi)\right)$ (equal to $f(x)$ if $f(x)$ is continous at x and to the average of the two limit points if not) if the function satisfies the *Dirichlet conditions*:

1. it must be piecewise analytic and single-valued;
2. it can only have a finite number of discontinuities;
3. it can only have a finite number of extrema.

The forward and backward Fourier transform of a function $f(x)$ exist if

1. $\int_{-\infty}^{\infty} |f(x)|\,\mathrm{d}x$ exists;
2. $f(x)$ has a finite number of discontinuities;
3. $f(x)$ has a bounded variation.

2.3 Real-valued vs. Compex-valued Functions

Up to now, we have covered the most general case of a complex-valued $f(x)$. In most physics problems, we measure *real* data. If we then compute the forward Fourier transform, we still get a complex-valued function, that can contain twice as much information. We thus suspect

that there is some symmetry in the Fourier series and transform when the input data is real. If $f(x)$ is real, we observe that

$$A_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\, \mathrm{e}^{-\mathrm{i}nx}\, \mathrm{d}x = A_{-n}^* \,.$$

Similarly for the Fourier transform, we find $F(k) = F^*(-k)$ if $f(x)$ is real-valued. This, in turn, also ensures that the backward transform again yields a real-valued function. Note that these symmetry conditions also demand that A_0 and $F(0)$ are both real. These conditions are indeed satisfied, because we have

$$A_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\, \mathrm{d}x = \langle f(x) \rangle \,,$$

and similarly

$$F(0) = \int_{-\infty}^{\infty} f(x)\, \mathrm{d}x \,.$$

2.4 Discretization

Experimental data is almost never recorded continuously, but rather at discrete intervals. We therefore review some theory of the *discrete Fourier transform* or DFT. Instead of a continuous function $f(x)$, we now have a discrete function $f_n \equiv f(x_n)$, where $x_n = n\,\Delta_x$ and $n = 0, \ldots, N-1$. Replacing the integral in equations 2.3 and 2.4 with a sum, we get the *forward discrete Fourier transform*

$$F_n = \sum_{m=0}^{N-1} f_m\, \mathrm{e}^{-2\pi \mathrm{i}\, nm/N} \,, \tag{2.5}$$

and the *backward discrete Fourier transform*

$$f_n = \frac{1}{N} \sum_{m=0}^{N-1} F_m\, \mathrm{e}^{2\pi \mathrm{i}\, nm/N} \,. \tag{2.6}$$

$F_n \equiv F(k_n)$ now also samples discrete wave numbers $k_n = n/(2N\,\Delta_x)$.

2.4.1 The Nyquist frequency and aliasing

The natural consequence of the discrete sampling of the data is that there is a limit on detectable wavelengths. The lowest frequency (or wave number) associated with the longest wavelength that cannot be detected is called the *critical* or *Nyquist frequency*,

$$f_{\mathrm{Nyq}} = 1/(2\,\Delta_x) \,. \tag{2.7}$$

A wave component with a frequency of exactly the Nyquist frequency can, in principle, be detected, but it is impossible to determine the phase unless the amplitude is assumed and vice versa. If the frequency f is above the Nyquist frequency, it causes *aliasing*. Due to the oscillatory behavior, the sampling produces exactly the same results as in the case of a wave with a lower frequency $f' = f + 2lf_{\mathrm{Nyq}}$, where $l \in \mathbb{Z}$ such that $-f_{\mathrm{Nyq}} \leq f \leq f_{\mathrm{Nyq}}$.

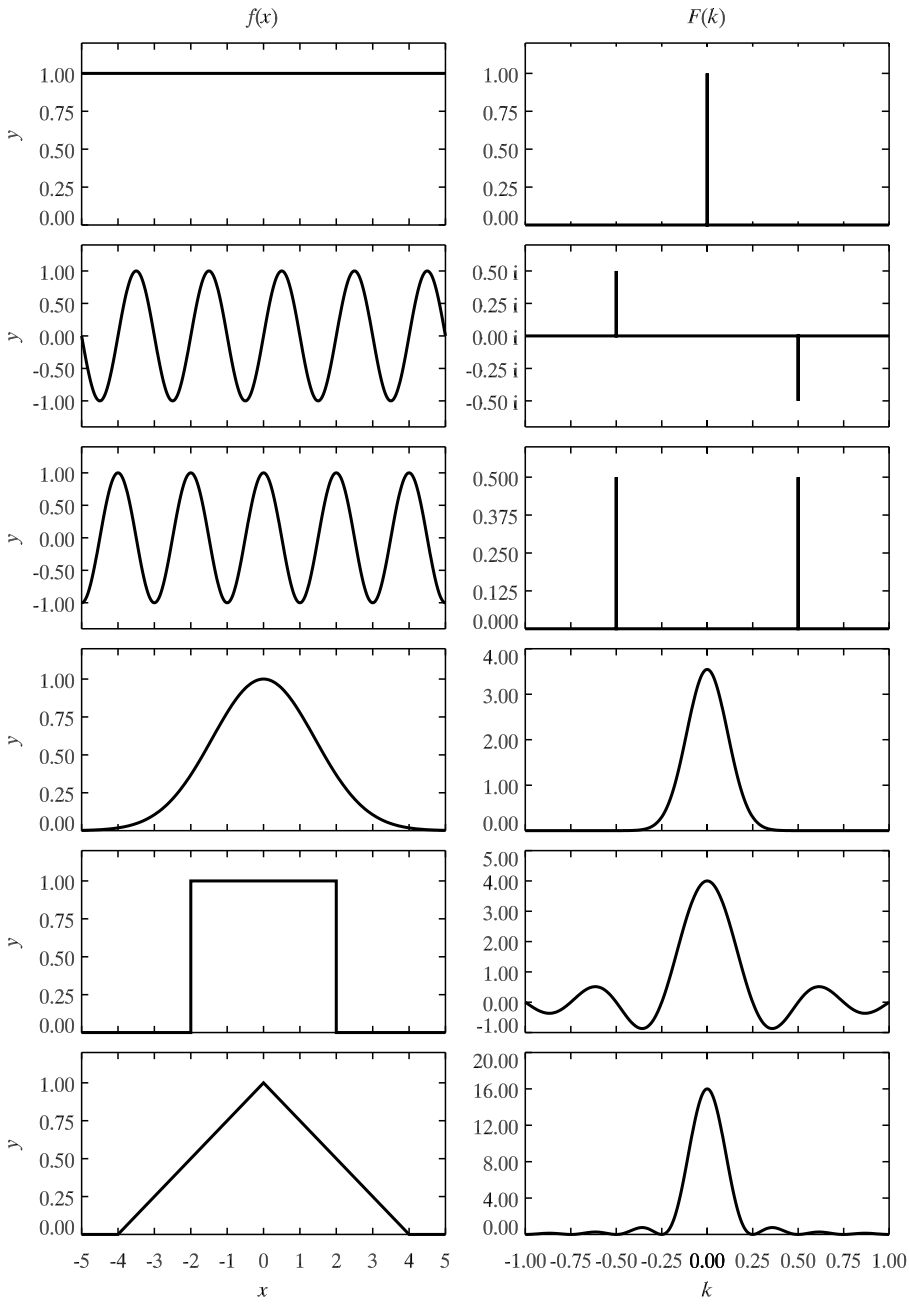

Figure 2.1: Some examples of functions and their Fourier transforms. Left column from top to bottom: constant 1, $\sin \pi x$, $\cos \pi x$, Gaussian $\mathrm{e}^{-x^2/4}$, the rectangle $\Pi(x/4)$, and the triangle $\Lambda(x/4) = \Pi(x/4) \otimes \Pi(x/4)$. Their Fourier transforms in the right column are from top to bottom: $\delta(k)$, $\frac{1}{2}\mathrm{i}\,(\delta(k+0.5) - \delta(k-0.5))$, $\frac{1}{2}\,(\delta(k+0.5) + \delta(k-0.5))$, $\sqrt{4\pi}\,\mathrm{e}^{-4\pi^2\,k^2}$, $4\,\mathrm{sinc}(4k)$, and $16\,\mathrm{sinc}^2(4k)$.

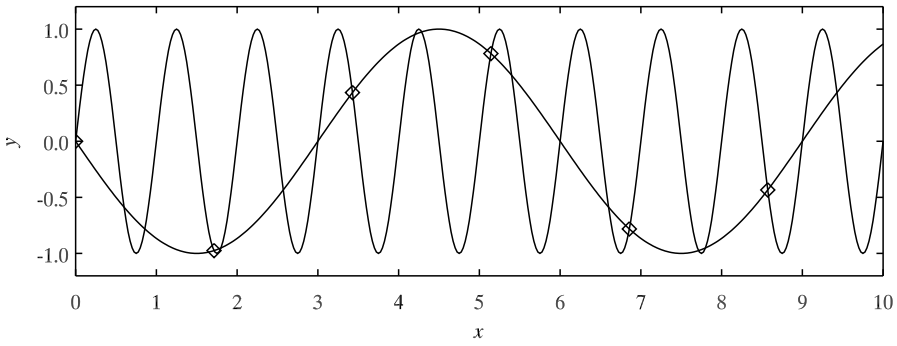

Figure 2.2: Aliasing of a signal due to undersampling. The signal has a frequency of 1 and is sampled at intervals of 12/7. The resulting aliased signal thus has a frequency of $1 - 14/12 = -1/6$.

2.5 The Fast Fourier Transform

Simple evaluation of the sums in equations 2.5 and 2.6 require $\mathcal{O}(N^2)$ operations. A *Fast Fourier Transform* or FFT is an efficient algorithm to compute the same result in $\mathcal{O}(N \log N)$ operations. Though there are many FFT algorithms, the most common is the *Cooley-Tukey* algorithm. It recursively breaks down a DFT of size N into two smaller DFTs of size N_1 and N_2 with $N = N_1 N_2$ and $\mathcal{O}(N)$ multiplications by complex roots of unity (i.e., solutions to the equation $x^N = 1$). For the interested reader, we explain the simplest form here. It is the so-called radix-2 decimation-in-time. Assuming N is divisible by 2 and writing $N' = N/2$, let the DFT of the even-indexed data points $f_0 = f_{1,0}, f_2 = f_{1,1}, \ldots, f_{N-2} = f_{1,N'-1}$ be denoted as $F_{1,0}, F_{1,1}, \ldots, F_{1,N'-1}$, and the DFT of the odd-indexed data points $f_1 = f_{2,0}, f_3 = f_{2,1}, \ldots, f_{N-1} = f_{2,N'-1}$ as $F_{2,0}, F_{2,1}, \ldots, F_{2,N'-1}$. It then follows that

$$
\begin{aligned}
F_n &= \sum_{m=0}^{N-1} f_m \, \mathrm{e}^{-2\pi \mathrm{i}\, nm/N} \\
&= \sum_{m=0}^{N/2-1} f_{2m} \, \mathrm{e}^{-2\pi \mathrm{i}\, n(2m)/N} + \sum_{m=0}^{N/2-1} f_{2m+1} \, \mathrm{e}^{-2\pi \mathrm{i}\, n(2m+1)/N} \\
&= \sum_{m=0}^{N'-1} f_{1,m} \, \mathrm{e}^{-2\pi \mathrm{i}\, nm/N'} + \mathrm{e}^{-2\pi \mathrm{i} n/N} \sum_{m=0}^{N'-1} f_{2,m} \, \mathrm{e}^{-2\pi \mathrm{i}\, nm/N'} \\
&= \begin{cases} F_{1,n} + \mathrm{e}^{-2\pi \mathrm{i} n/N} F_{2,n} & \text{if } n < N' \\ F_{1,n-N'} - \mathrm{e}^{-2\pi \mathrm{i}(n-N')/N} F_{2,n-N'} & \text{if } n \geq N' \end{cases}
\end{aligned}
$$

More complicated versions of the same principle can be applied to other decimations. In principle, a dataset of arbitrary length can be decomposed recursively in this manner to give a computation time of $\mathcal{O}(N \log N)$. In practice, however, most FFT impementations only provide fast, optimized routines for factorizations in small primes.

Chapter 3

Working with 1D Fourier Transforms

In this chapter we use the 1D Fourier transform to study intensity oscillations in the solar chromosphere. We use observations from the *Transition Region And Coronal Explorer* (TRACE) of two UV passbands: 1600 Å and 1700 Å. These two passbands sample the low chromosphere and the high photosphere, respectively. The data that we use here is a time seqeunce. The original data consisted of images, but in this case we only use a single pixel. You can find the data in `TRACE_1d.save`.

3.1 Power Spectrum

The square of the absolute value of a complex Fourier coefficient is related to the amplitude of the contribution of the associated oscillation in the data. Remember that this value is commonly called *power*.

Exercise 3.1. Compute and plot the power spectrum of the 1600 Å data sequence. Annotate the x-axis in mHz units. Do you see the 5-minute p-mode oscillations?

3.2 Apodization

Most observational data aren't perfectly periodic. The discontinuity between the end and the start of the sequence introduces undesired artifacts in the transform. These artifacts are often termed *edge effects*. We can suppress them by *apodizing* the data sequences. The data is made partially or fully periodic by multiplying it with an *apodization window* that decreases at the edges of the data. There is considerable choice in the exact shape of the window to use.

Exercise 3.2. Apodize the data with a Hanning window, and compute the power spectrum of the result. Use the IDL function `hanning`. Explain the differences with the power spectrum of the data that wasn't apodized. Can you explain what went wrong? *Hint: plot the original and apodized data.*

Exercise 3.3. To avoid introducing Fourier components with the apodization, we should at least subtract the mean of the sequence before we apply the window. Subtract the mean value

from the data, apply the Hanning window, and compute the power spectrum. Compare the result to the power spectra computed in the previous exercises and explain the differences.

3.3 Convolutions, Cross-power, Filtering, and Phase Difference

The convolution of two datasets d_1 and d_2 of equal length N is defined as

$$(d_1 \otimes d_2)(n) = \sum_{m=0}^{N-1} d_1(m)\, d_2(n-m)\,, \tag{3.1}$$

where the datasets are treated as being cyclic, i.e., $d_1(n + lN) = d_1(n)$ with $l \in \mathbb{Z}$. A simple algorithm would require $\mathcal{O}(N^2)$ operations. As we'll see, it is possible to perform a convolution (or deconvolution) with FFTs in only $\mathcal{O}(N \log N)$ operations.

Starting from equation 3.1, write $D_1(k) = \mathcal{F}_x[d_1(x)](k)$ and $D_2(k) = \mathcal{F}_x[d_2(x)](k)$, and substitute the backward transform of D_2 for d_2,

$$\begin{aligned}
(d_1 \otimes d_2)(n) &= \sum_{m=0}^{N-1} d_1(m) \sum_{l=0}^{N-1} D_2(l)\, \mathrm{e}^{2\pi \mathrm{i}(n-m)l/N} \\
&= \sum_{l=0}^{N-1} D_2(l) \left(\sum_{m=0}^{N-1} d_1(m)\, \mathrm{e}^{-2\pi \mathrm{i}\, ml/N} \right) \mathrm{e}^{2\pi \mathrm{i}\, nl/N} \\
&= \sum_{l=0}^{N-1} D_1(l)\, D_2(l)\, \mathrm{e}^{2\pi \mathrm{i}\, nl/N} = \mathcal{F}_l^{-1}[D_1(l)\, D_2(l)](n)\,.
\end{aligned}$$

We see now that in order compute a convolution with Fourier transforms, we can compute the DFT of the two datasets, multiply the result pointwise, and transform back.

In many cases it won't be practical to use a Fourier transform to compute a convolution. The kernel is often smaller than the dataset itself, thus requiring much less computations in the direct approach. A Fourier transform can still be used by padding the kernel with zeroes, but the computational advantage is lost quickly. We discuss three cases in which a Fourier-based approach is beneficial.

The cross-power of two sequences is defined as

$$P_{\text{cross}}(k) = \mathcal{F}_x^*[d_1(x)](k)\, \mathcal{F}_x[d_2(x)](k)\,. \tag{3.2}$$

The regular Fourier power is just the special case of the cross-power where $d_1 = d_2$. Contrary to the regular power, the cross-power is complex.

Exercise 3.4. Compute the cross-power spectrum of the 1600 and 1700 Å sequences. Plot the absolute value of the spectrum. What does the cross-power describe?

3.3.1 Cross-correlation

The *cross-correlation* of two datasets is defined as

$$d_1(n) \star d_2(n) = d_1^*(-n) \otimes d_2(n)\,. \tag{3.3}$$

Since we work with real data, and $\mathcal{F}_n[d_1(-x)](k) = \mathcal{F}_n^*[d_1(n)](k)$, we have $\mathcal{F}_n[d_1(n) \star d_2(n)](k) = \mathcal{F}_n^*[d_1(n)](k)\, \mathcal{F}_n[d_2(n)](k)$, which is just the cross-power of d_1 and d_2.

Exercise 3.5. $d_1 \star d_2$ is obviously real-valued. Show that the inverse Fourier transform of the cross-power indeed gives a real-valued result.

Exercise 3.6. Compute the cross-correlation of 1600 and the 1700 Å TRACE sequences, and explain what you see. Can you explain where the name "cross-correlation" comes from?

3.3.2 Autocorrelation

A special case of the cross-correlation is the *autocorrelation*. As the name already suggests, it's the cross-correlation of a function with itself.

Exercise 3.7. Compute the autocorrelations of the 1600 and the 1700 Å sequences.

3.3.3 Convolutions

Exercise 3.8. Compute a convolution with Fourier transforms of the 1600 Å sequence with a Gaussian with a FWHM of 10 timesteps and make a plot of the result. Verify the result by comparing with a direct convolution using the IDL `convol` function. Watch out for normalization in the Fourier transform.

With a *deconvolution* we attempt to solve the equation $d_c = d_1 \otimes d_2$ for d_1 given d_c and d_2. In a direct approach, this would require solving a system of N linear equations, but we can do it much easier with FFTs. Though the deconvolution works in principle, in practice noise will spoil the results.

Exercise 3.9. Explain how to use Fourier transforms to compute a deconvolution. Convolute the data with the Gaussian as in exercise 3.8, and then deconvolute. Explain what happened.

3.3.4 Filtering

The Fourier transform can be used to filter certain frequencies out of data. In principle, filtering in Fourier space is simply adjusting the power spectrum. We can apply very simple filters, such as selecting only a certain band, but there are many powerful applications of filtering in Fourier space. Here, we'll work with an *optimum filter* to overcome the problems with noise in a deconvolution.

What we observe is a sum of the smeared data and noise, $o(x) = d(x) \otimes a(x) + n(x)$, where $d(x)$ is the data, $a(x)$ is the smearing function, and $n(x)$ is noise. A simple deconvolution with $a(x)$ will result in $r(x) = \mathcal{F}_k^{-1}[\mathcal{F}_x[o(x)](k)/\mathcal{F}_x[a(x)](k)](x)$. Because $\mathcal{F}_x[a(x)](k)$ tends to be small for large k, this increase the power of high frequency noise. What we really want, is to correct for $a(x)$ as best as possible while supressing the noise $n(x)$. The *optimum* filter is a filter $\Phi(k)$ that gives the closest smooth restored data $r(x)$ to the original data $d(x)$ (typically in the root-mean-square error sense),

$$r(x) = \mathcal{F}_k^{-1}[\mathcal{F}_x[o(x)](k)\,\Phi(k)/\mathcal{F}_x[a(x)](k)](x)\,. \tag{3.4}$$

We assume the noise is both random and uncorrelated to the signal. These assumptions are not essential to the derivation, but they reduce the complexity of the problem. If we do the math, we'll find that

$$\Phi(k) = \frac{P_s(k)}{P_s(k) + P_n(k)}\,, \tag{3.5}$$

where $P_s(k)$ is the power spectrum of the smeared signal, and $P_n(k)$ is the power spectrum of the noise. Of course, we don't know $P_s(k)$ or $P_n(k)$. However, because this is the best possible filter, in most cases we can approximate the actual smeared signal and noise with simple models that depend only on a few parameters.

Exercise 3.10. We will attempt to reconstruct data that we have degraded with both a smearing function and with noise. Convolute the data with a smearing function and add white noise with an amplitude of 0.1 (use the IDL function `randomn`). Examine the data in Fourier space and estimate the noise level. Approximate the smeared signal with a power law. Now compute the optimum filter from the approximated smeared data and the noise level. Correct the data with the filter and compare the result with the original data.

3.3.5 Phase Difference

Multiplication of two vectors in complex space can be visualized as summing the angles of the vectors with the real axis and multiplying their lengths. Complex conjugation can be visualized as reversing the angle of the vector with the real axis. So the angle the cross-power makes with the real axis is really the angle between the two Fourier coefficients. This angle is the *phase difference* between the two datasets. There is, of course, a 2π-ambiguity in the result.

Exercise 3.11. Compute and plot the phase difference spectrum of the 1600 and 1700 Å sequences. If you have more than one sequence, how would you compute the average phase difference spectrum?

There is a noticable drift in the phase difference. This is because the observations weren't made simultaneously. A fixed time delay Δt between the two sequences causes a linear phase delay in Fourier space of $\phi(f) = 2\pi\Delta t\, f$.

Exercise 3.12. Correct the phase difference spectrum for the time delay between the sequences.

3.3.6 Shifting

In essence, by correcting the phase difference for the non-simultaneous exposure, we are *shifting* one of the arrays to match the other. In this case, we simply adjust the resulting phase difference, but we can also adjust the phases of the Fourier components. If we then transform back, we will get the shifted array. The advantage of this method over an interpolation-based routine is that in this case, we don't change the power spectrum of the data.

Exercise 3.13. Shift the 1600 Å data by 0.67 pixel to the right both using the IDL procedure `interpolate` with the keyword `cubic=-0.5` and using the Fourier transform. Compare the results.

Chapter 4

Fourier Transforms in Multiple Dimensions

Up to now we have only covered Fourier transforms in one dimension. We will now extend this to multiple dimensions. We'll only cover the forward transform here, because the backward transform is extended in much the same way. Let's start with the analytical forward Fourier Transform in two dimensions,

$$F(k_x, k_y) = \mathcal{F}_{x,y}[f(x,y)](k_x, k_y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, \mathrm{e}^{-2\pi \mathrm{i}(k_x x + k_y y)}\, \mathrm{d}x\mathrm{d}y\,. \tag{4.1}$$

From this equation, we can see that the multi-dimensional Fourier transform is nothing more than sequential application of a one-dimensional Fourier transform in all dimensions,

$$\begin{aligned} F(k_x, k_y) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, \mathrm{e}^{-2\pi \mathrm{i}(k_x x + k_y y)}\, \mathrm{d}x\mathrm{d}y \\ &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(x,y)\, \mathrm{e}^{-2\pi \mathrm{i}\, k_x x}\, \mathrm{d}x\right) \mathrm{e}^{-2\pi \mathrm{i}\, k_y y}\, \mathrm{d}y \\ &= \mathcal{F}_y[\mathcal{F}_x[f(x,y)](k_x, y)](k_x, k_y)\,. \end{aligned}$$

The same holds for the discrete Fourier transform of a N_x-by-N_y array $d(x,y)$,

$$F(k_x, k_y) = \sum_{x=0}^{N_x-1}\sum_{y=0}^{N_y-1} d(x,y)\, \mathrm{e}^{-2\pi \mathrm{i}(k_x x + k_y y)}\,. \tag{4.2}$$

This transform now has two Nyquist frequencies, one in each dimension. Since it is again a composition of two one-dimensional Fourier transforms, an FFT algorithm can compute the result in $\mathcal{O}(N \log N)$ operations, where $N = N_x\, N_y$.

It is now simple to extend the Fourier transforms to more than two dimensions. It should also be noted that in order to compute a multi-dimensional Fourier transform, you don't have to compute it at once, but you can sequentially process each dimension seperately.

Chapter 5

Working with 2D and 3D Fourier Transforms

In this chapter, we'll perform some analysis with 2D and 3D Fourier transforms. We'll use images from the DOT in the G band and in Ca II H for 2D transforms, and a sequence of TRACE images for 3D transforms.

5.1 Power

Exercise 5.1. Compute the 2D spatial power spectrum of a G-band and Ca II H image.

5.1.1 Annular Integration

In most cases, we can assume that the properties of the image recorded by the camera are independent of the orientation of the camera. In this case, the horizontal and vertical axes of the image aren't even determined by something on the sun. Therefore, it's nonsense to have a k_x and a k_y. We should combine the two into a single spatial wavenumber k, given by $k^2 = k_x^2 + k_y^2$. The power can be averaged directly over annuli of constant k.

Exercise 5.2. Compute the 1D spatial power spectrum of a DOT image by annular integration of the 2D power spectrum.

Exercise 5.3. Compute the 2D k-f power spectrum from the 3D Fourier transform of the TRACE sequence. Of course, the data is real-valued. Can you avoid unnecessary computations?

5.2 Cross-power

Exercise 5.4. Compute the 2D spatial cross-power spectrum of the DOT images and plot the absolute value.

5.3 Phase Difference

Exercise 5.5. Compute the phase difference between the first G-band image and the Ca II H image. Don't forget to correct for the delay between the images.

5.3.1 Computing Displacement

If we have two images that are similar except for an unknown shift, we can use Fourier transforms to accurately determine the shift. This is a common problem in data reduction. Atmospheric seeing, solar rotation, telescope jitter, etc., all cause whole-field motions that must be compensated for in post-processing. In the ideal case, the images have the same power spectrum, but the phases on the Fourier components are different. We can find the offset by analyzing the phase difference between the images. It's usually easier to simply use a convolution. We then expect to have a peak at the location corresponding to the displacement that best matches the shift between the images.

Exercise 5.6. Compute the displacement between the two DOT G-band images with a convolution. Verify that the result is correct by shifting one and blinking it against the other. What happens if you exchange the images? Which image is the reference? How accurate do you think you can determine the displacement?

Appendix A

IDL Quick Reference

A.1 Getting Help

IDL comes with a built-in help system. You can access from the command line:

```
> ?
```

You can also request help on a specific subject (e.g., a routine or procedure) directly:

```
> ? subject
```

A.2 Graphics Output

IDL can output graphics on a number of different devices. The three most common are `win` for Microsoft Windows, `x` for *nix XWindows, and `ps` for PostScript. You can use the procedure `set_plot` to change the output device.

A.3 Plotting Results

You can use the following code to make a nice plot of your results.

```
> plot, xaxis, yaxis, xtitle='xtitle', ytitle='ytitle'
```

To overplot additional curves, use `oplot`.

```
> oplot, xaxis, yaxis2
```

These routines work the same for any display.

A.4 Displaying Images

Making nice plots of 2D data is more difficult than simple plots of curves. On your screen, pixels have a natural size, so the size of the plot is determined by the size of the image you're displaying. On a postscript device (e.g., a printer), there is no natural pixel size, so the image is scaled to a desired size. Here is some code you can use to produce postscript output.

```
> set_plot, 'ps'
> device, filename='output.eps', /encapsulated, xsize=10, ysize=6.2
> tv, bytscl(image), 0.15, 0.15, xsize=0.8, ysize=0.8, /normal
> plot, xaxis, yaxis, /nodata, /noerase, /xstyle, /ystyle, $
>     position=[0.15,0.15,0.95,0.95], xticklen=-0.05, yticklen=-0.05, $
>     xtitle='xtitle', ytitle='ytitle'
> device, /close
```

The image displayed by `tv` is scaled to $0.8 \times 10 = 8$ by $0.8 \times 6.2 = 4.96$ centimeter. Of course, if your image needs to have square pixels, you must choose the image aspect ratio correctly. This means that you also have to adjust the size of the postscript output. Furthermore, you must take care that your axis tickmarks are in the correct location. Always ask yourself where the tickmarks need to be: between pixels, or at pixel centers?

Appendix B

Alfred's Guide to Making Beautiful and Complicated Figures with IDL

In Appendix A we've discussed how to make a simple figure with IDL. While this suffices for many situations, sometimes you just want to do a little better. Here, we'll discuss how to make complicated yet beautiful figures with IDL. The first rule of making nice figures in IDL is: don't let IDL do anything by itself. We'll control all parameters of our figure ourselves. I should add as a disclaimer that a number of things described here are on the basis of what I find aesthetical. Your personal preferences might be different. The methods, however, are universially applicable.

I will refer to the *figure* as the entire product. The *plots* are only those areas that contain the objects that are to be displayed.

B.1 Setting Up

There's really no point in making nice figures unless you're going to use them in some sort of document. Also, the pixel-oriented screen output is different in nature than the resolution-independent PostScript output that we'll want eventually. We therefore set the output to PostScript immediately,

```
> set_plot, 'ps'
```

By default, IDL will use an ugly, built-in font. The device fonts look a lot better, but they may change from one device to another. Even two printers may not have the same PostScript fonts. Therefore, we choose to use the TrueType fonts:

```
> !p.font = 1
```

The lines IDL draws are typically very thin. We make them a bit thicker by setting some system variables:

```
> !p.thick = 2
> !x.thick = 2
> !y.thick = 2
> !z.thick = 2
```

We should decide on how our figure will be layed out. How many plots will it have, how large are they, and how will they be arranged? As an example, I'll cover the case of two plots of equal size, with an aspect ratio equal to the golden ratio. I'll assume the plots have the same x-axis, so we'll position them one above the other. In many cases where the figure is complicated, I prefer to make a sketch on paper. At this point, you should decide on the width of the figure. In this case we'll take 8.8 centimeter, because it exactly fits a column in most journals.

```
> xsize = 8.8
```

B.2 Computing Figure Measurements

I like to have the same distance from the bottom of my figure to the bottom horizontal axis as from the left edge to the left vertical axis. Trial and error have shown that 12% of a 8.8-centimeter wide figure is a good value.

```
> margin = 0.12
```

I also have a fixed distance between two panels, the top of the figure to the top horizontal axis, and the right of the figure to the right vertical axis.

```
> wall = 0.03
```

We must compute the final size of the figure, and of the plots in the figure. We can now compute the width of the plots in units normalized to an 8.8-centimeter wide figure,

```
> a = xsize/8.8 - (margin + wall)
```

Since we settled on the golden ratio for the aspect ratio of the plots, we can compute the height of each plot,

```
> b = a * 2d / (1 + sqrt(5))
```

Next, we compute the vertical size of the figure,

```
> ysize = (margin + b + wall + b + wall)*8.8
```

Now, we must compute the corners of the plots so we can instruct IDL to place them exactly where we want them. So far we've worked with units normalized to an 8.8-centimeter wide figure, but IDL can't do that. We will calculate the horizontal and vertical coordinates in units normalized to the figure width and height, respectively.

```
> x1 = margin*8.8/xsize
> x2 = x1 + a*8.8/xsize
> xc = x2 + wall*8.8/xsize
> y1 = margin*8.8/ysize
> y2 = y1 + b*8.8/ysize
> y3 = y2 + wall*8.8/ysize
> y4 = y3 + b*8.8/ysize
> yc = y4 + wall*8.8/ysize
```

If you've done everything right, the values of `xc` and `yc` should now be equal to 1 (or, because of limited accuracy, very close).

We also want the horizontal and vertical tick marks of the plots to have equal length. The manual is rather cryptic, but it turns out IDL scales the length of a vertical tick mark to the height of the plot, and the horizontal to the width. I keep my ticks the same length, irrespective of the size of the figure.

```
> ticklen = 0.01
> xticklen = ticklen/b
> yticklen = ticklen/a
```

B.3 Plotting Data

We should start by opening the output file. We want encapsulated PostScript output of a specific size. We also want to use the TrueType Times font, with a font size of 10 points.

```
> device, filename='output.eps', /encapsulated, xsize=xsize, ysize=ysize, $
> /tt_font, set_font='Times', fontsize=10
```

If we were plotting an image, we would add the keywords `bits_per_pixel=8` to make sure we get a sufficient number of shades of gray.

Next, we'll create the frame of the first plot, using the `position` keyword to control where it will end up in the figure,

```
> plot, xrange, yrange1, /nodata, /noerase, position=[x1,x2,y1,y2], $
> xtitle='xtitle', ytitle='ytitle', $
> xminor=0, yminor=0, xticklen=xticklen, yticklen=yticklen, $
> xticks=nxticks, xtickv=xtickvalues, yticks=nyticks, ytickv=ytickvalues
```

Now, we plot the data,

```
> oplot, xaxis, yaxis1
```

Of course, we can add many more `oplot` or other statements here, as long as they use the axis drawn by the `plot` call.

Once we're done with the first plot, we proceed with the second plot. In this case, we don't want annotations on the x-axis.

```
> plot, xrange, yrange2, /nodata, /noerase, position=[x1,x2,y3,y4], $
> ytitle='ytitle', $
> xminor=0, yminor=0, xticklen=xticklen, yticklen=yticklen, $
> xticks=nxticks, xtickv=xtickvalues, yticks=nyticks, ytickv=ytickvalues, $
> xtickname=replicate(' ',nxticks+1)
```

The only way to suppress the tick names is by setting them to spaces with the `{x|y|z}tickname` keyword. Now plot the data,

```
> oplot, xaxis, yaxis2
```

Finally, we want to close the output file. Optionally, we will set the plot device back to the screen (XWindows output, in this case).

```
> device, /close
> set_plot, 'x'
```

B.4 Notes

IDL has a built-in tool `!p.multi` for making multi-panel plots. I've used this in the past, and though it does what it's supposed to, it's far from pleasent to work with. It's limited in its possibilities (e.g., it only allows for plots of equal size), and it changes things I don't want it to touch, such as the font size. The `plot` keyword `noerase` comes in handy. By setting it, `plot` does not erase the figure before it starts. You must take care yourself that your plots don't overlap, etc., but in return you can make your figure as complicated as you want.

If you're making a figure of an image, there are a few things you should take care of. First of all, the aspect ratio of the plot area is set by the image. Secondly, you should probably add `bits_per_pixel=8` to the `device` call to ensure sufficient color depth. Finally, you should be careful with the order in which you draw the figure. The figure will be built up in layers, so if you want to plot something over the image (e.g., contours, etc.) you must `tv` the image first, then plot the contours. In most cases, I will even `tv` the image before I draw the axes with the `plot` call. In some extreme cases, I have even drawn the axes last, by first using invisble axes.

Obviously, you don't want to type all this every time you make a small adjustment to this particular figure. I always write the code to produce a figure in an IDL procedure. The added advantage is that IDL will clean up after the procedure ends, removing all non-global variables (i.e., not `!p.font` etc.).