Chapter 2

Describing Univariate Time Series

As briefly discussed in Section 1.3, aims of time series analysis can be roughly grouped into four categories: descriptive, inferential, predictive, and control. In this chapter, we will consider a number of graphs that can be used for trying to classify a time series into long, short, or no memory. Later in the text, we will see that these ideas of "memory" fall in line with more mathematical concepts that are assumed to hold throughout this chapter. (In Chapter 3, we will discuss at methods for transforming time series so that the series satisfies assumptions. We will discuss exactly what are those assumptions in Chapter 4.)

2.1 Some Basic Notation

In Section 1.1, the index set $\mathcal{T}$ of the time series was introduced. In this book, our attention will be restricted a time index set with evenly spaced, discrete values. Time series notation is similar in spirit to random variable notation. Most often the random series is denoted by capital letters at the end of the alphabet. In accordance with this convention, we will denote the random time series by $X(t)$ if $\{t \in \mathcal{T}\}$ is continuous, or by X_t if the index set is discrete (see Section 1.1). In Chapter 4, we will provide a formal definition of a time series based on a probabilistic (random) perspective. In this chapter, we will focus on observed values of the series, denoted by a lower-case x_t, using the subscript notation, since the time index in all our examples is discrete. The length of the time series will be denoted n.

2.2 Basic Descriptive Statistics

The first aim of any statistical procedure is to give a succinct description of the data being analyzed, both graphically and numerically. For example, in elementary statistical methods, commonly considered descriptive statistics for a univariate random sample are the sample mean or sample standard deviation. For bivariate data, one might consider the sample correlation coefficient. Each of these measures are single numbers. In time series analysis, there are four basic graphical techniques for describing a time series: the time plot, the

correlogram, the partial correlogram, and the periodogram. In Figure 1.1, we saw time plots of 10 different series, most of which were described in some way in Chapter 1.

In this chapter, we will introduce the other three graphs and illustrate how they are used as descriptive statistics for the time series data $x_1, \ldots, x_n$. The first two of these, are the sample correlogram and the sample partial correlogram. Because these two graphs explain features of the series across time, they are often placed into a group of time series analysis methods referred to as **time domain methods**. The third graph, called the periodogram, examines the behavior of the series via harmonic analysis and is in a group of methods referred to as **frequency domain methods**. These three graphs are displays of numerical descriptive statistics of a time series. As we define the descriptive statistics for a time series, it is essential to your understanding that you recognize that the descriptive statistics are themselves *series of numbers*. This is in direct contrast to the *single-valued* descriptive statistics in elementary methods.

2.2.1 The Sample Correlogram

The distinguishing characteristic of a time series is that it can exhibit serial correlation; that is, correlation over time. For example, Figure 2.1 contains two scatterplots of x_t versus x_{t-1} for Series I (top plot) and Series II (bottom plot). Note that in Series I, there appears to be little correlation, while in Series II, there appears to be high positive correlation between points that are one unit apart in time. For these two series – indeed, for any time series – we can also construct plots of x_t versus x_{t-2}, of x_t versus x_{t-3}, an so on. Each of these plots enable us to visually examine if there is a relationship between the series at with time-lagged values of itself.

There is also a numerical means for examining correlation of a series with time-lagged values of itself. To begin this discussion, consider how correlation is measured between two variables in elementary statistics. The sample correlation coefficient, given by

$$\hat{\rho} = \frac{\sum_{i=1}^{n}(w_i - \bar{w})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(w_i - \bar{w})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}. \tag{2.1}$$

is a numerical measure the strength of the linear association, or correlation, between the n independent observations of a set of bivariate data $(w_1, y_1), \ldots, (w_n, y_n)$. On the other hand, for time series data set $x_1, \ldots, x_n$, we want to measure the correlation of the data with lagged values of themselves. Thus for a lag v, we have $n - v$ pairs of xs that are separated by v time units, namely the pairs

$$(x_1, x_{1+v}),\ (x_2, x_{2+v}),\ \ldots,\ (x_{n-v}, x_n).$$

One way to measure the correlation between these pairs is to apply the formula in equation (2.1), letting abscissa in each ordered pair take the role of the ws in equation (2.1), letting the ordinates play the role of the ys, and modifying the limits on the sum appropriately. In fact, the traditional measure of serial correlation in a time series is a slight modification of this and is called the **sample autocorrelation coefficient**, defined below in equation (2.2). In Section 6.1.2, we will discuss why this definition is used.

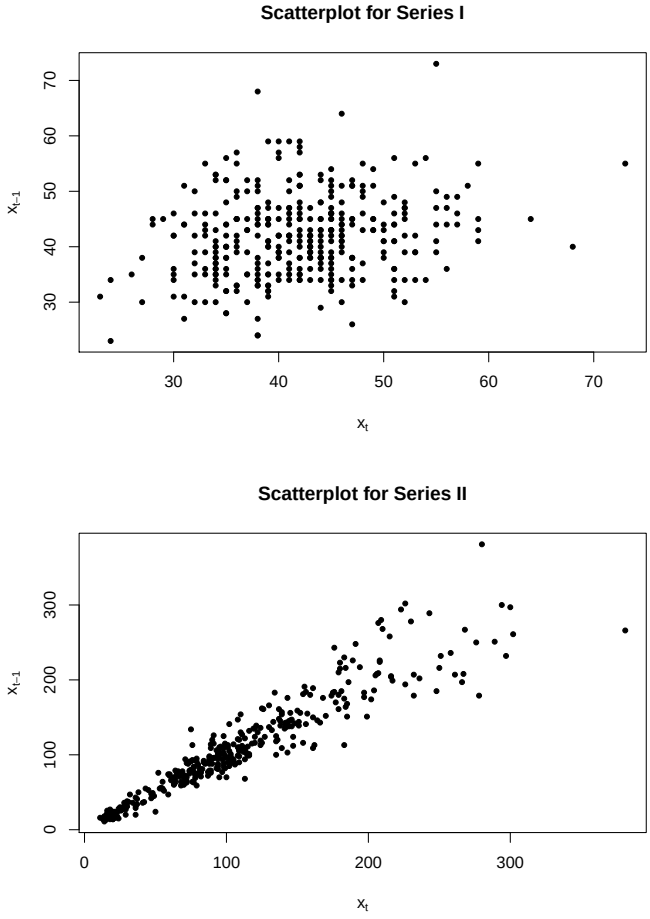


Figure 2.1: Scatterplots of x_t versus x_{t-1} for Series I (top) and Series II (bottom).

Definition 2.1 Sample Autocorrelation Function.
Let $x_1, \ldots, x_n$ be a univariate time series data set. The **sample autocorrelation coefficient at lag v** *is given by*

$$\hat{\rho}_v = \frac{\sum_{t=1}^{n-v}(x_t - \bar{x})(x_{t+v} - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2}, \quad v < n, \tag{2.2}$$

where $\bar{x} = n^{-1}\sum_{t=1}^{n} x_t$ is the sample mean of $x_1, \ldots, x_n$. A plot of $\hat{\rho}_v$ versus v for $v = 0, 1, \ldots, M$ for some maximum lag M is called the **correlogram** *of the data.*

There are a few properties of the sample autocorrelation coefficient that need to be noted. First $\hat{\rho}_0 = 1$. Secondly the largest value of v that can be considered is $n - 1$, since if $v \geq n$, there is no pair of x's separated by v time units. Furthermore, we can define $\hat{\rho}_{-v} = \hat{\rho}_v$, and

so we will sometimes write

$$\hat{\rho}_v = \frac{\sum_{t=1}^{n-|v|}(x_t - \bar{x})(x_{t+|v|} - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2}, \quad |v| < n.$$

Most importantly, notice the sample autocorrelation coefficient is not a single number, but is a *function of a lag* $v = 0, 1, \ldots, M$, having $M + 1$ values. Contrast this with the *single value* correlation coefficient in elementary statistics.

Now suppose we are considering a bivariate data set; that is, at each time we observe

$$\mathbf{z}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix}, \quad t = 1, \ldots, n.$$

Then we have the following definition.

Definition 2.2 Cross-correlation Coefficient.
The **cross-correlation coefficient at lag v** *is given by*

$$\hat{\rho}_{xy,v} = \begin{cases} \dfrac{\sum_{t=1}^{n-v}(x_t - \bar{x})(y_{t+v} - \bar{y})}{\sqrt{\sum_{t=1}^{n}(x_t - \bar{x})^2 \sum_{t=1}^{n}(y_t - \bar{y})^2}}, & v = 0, 1, \ldots, n-1 \\ \hat{\rho}_{yx,-v} & v = -(n-1), -(n-2), \ldots, 0. \end{cases}$$

Note that $\hat{\rho}_{xy}$ is not symmetric about $v = 0$. A plot of $\hat{\rho}_{xy,v}$ versus v for $v = -M, \ldots, M$ is called the **cross-correlogram of x and y**.

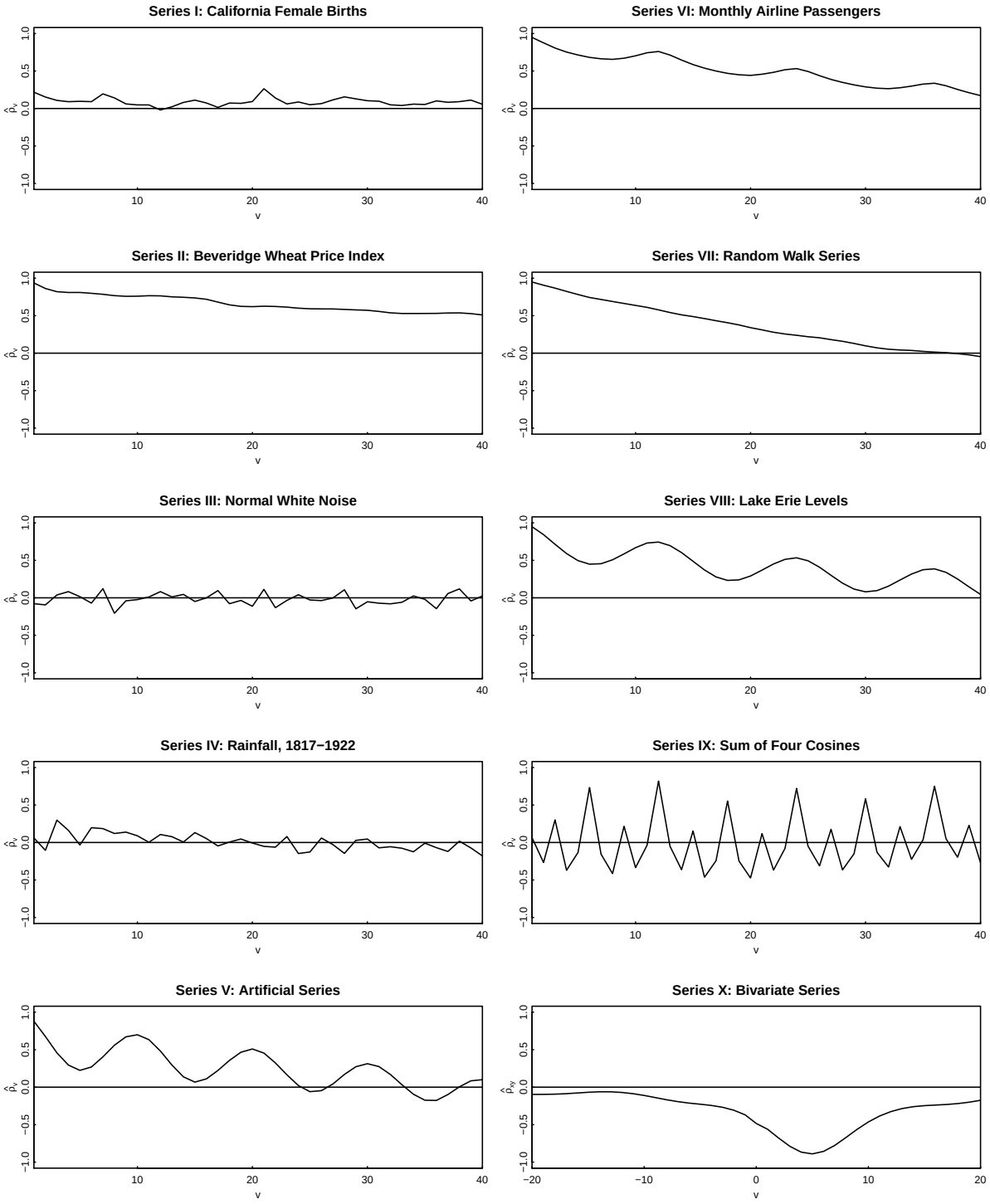


Figure 2.2: The correlograms of Series I through X.

Correlograms are used for three basic purposes. To illustrate these purposes, consider Figure 2.2, where we have given the correlograms for the ten series produced earlier. A large value of $\hat{\rho}_v$ is indicative of a possible periodicity in the data of v time units. For example, the correlogram of a pure cosine curve is also a sinusoid (see Exercise T2.2). Notice that Series V and VI exhibit this behavior.

A correlogram that does (does not) decay rapidly to zero and then stay there as v increases means that the series is short (long) memory. Note that the wheat price index, the artificial series, the airline data, and the random walk series appear to be long memory by inspection of the correlogram.

For the bivariate series, the cross-correlogram is provided. A large cross-correlation of lag v indicates that x and y may be very similar except that they are out of phase; that is, one of the series is leading the other. This would appear to be the case in the bivariate series that we have been considering.

2.2.2 The Partial Correlogram

As in elementary statistics, high correlation does not imply causality. Partial autocorrelation and multiple correlation are two additional measures of correlation encountered in elementary statistics. These two measures are used in an attempt to describe further the correlation in certain sets of variables. In the next paragraphs, we will provide a brief reminder of their description, as well as their analog in time series analysis, called the partial autocorrelation coefficients and residuals variances. As with the sample autocorrelation coefficient, when the partial correlation and multiple correlation are generalized to a time series context, they will become *series of numbers*, and not single-valued (as they are in elementary statistics).

In elementary statistics, if we have random samples from populations Y, Z, and W_1, ..., W_p, we can find the residuals e_y and e_z of regressing the observed y on the ws and the observed z on the ws. We then find the usual sample correlation coefficient between $e_{y,1}, \ldots, e_{y,n}$ and $e_{z,1}, \ldots, e_{z,n}$. The result is what is called the sample partial correlation coefficient between y and z given $w_1, \ldots, w_p$, and is often denoted $\hat{\rho}_{y\,z|w_1,\ldots,w_p}$. The **sample partial autocorrelation coefficient of lag v** is the correlation between x_t and x_{t+v} after having removed the common linear effect of the data in between. We denote the lag v partial autocorrelation by $\hat{\theta}_v$. In Section 6.4, we will provide a more exact definition of $\hat{\theta}_v$.

Going back to elementary statistics, the sample multiple correlation coefficient $R^2_{y|w_1,\ldots,w_p}$ is the proportion of variability in $y_1, \ldots, y_n$ that is "explained" by its linear relationship with the observed $w_{1,1}, \ldots, w_{1,n}, \ldots, w_{p,1}, \ldots, w_{p,n}$. The analog in time series analysis to the sample multiple correlation coefficient is the **standardized residual variance at lag v**. The standardized residual variance at lag v is computed in the following manner. First find the sample residual variance $\hat{\sigma}^2_v$ of the residuals of having regressed x_t on $x_{t-1}, \ldots, x_{t-v}$. Then divide $\hat{\sigma}^2_v$ by the $\hat{\sigma}^2$ (the sample variance of the xs) and subtract the result from one. The result is the standardized residual variance, which is the time series analog of the multiple correlation coefficient R^2. These values will always be between zero and one, and a small (large) value indicates that x_t is (is not) very predictable from its past, thus indicating that the data are long (short) memory. A useful rule of thumb is that the data are long memory

if their standardized residual variances sequence becomes less than $8/n$ for some lags.

In Figure 2.3, we display the partial correlogram and standardized residual variances for each of the univariate series that we have been considering. We can use the partial autocorrelations in the same way that the usual partial correlations are used. In Series VII for example, the correlogram takes some time to decay to zero, while the partial correlogram is large for lag one and then small for all other lags. This indicates that the correlation between x_t and x_{t+2} is due only to the common relationship of x_t and x_{t+2} to x_{t+1}.

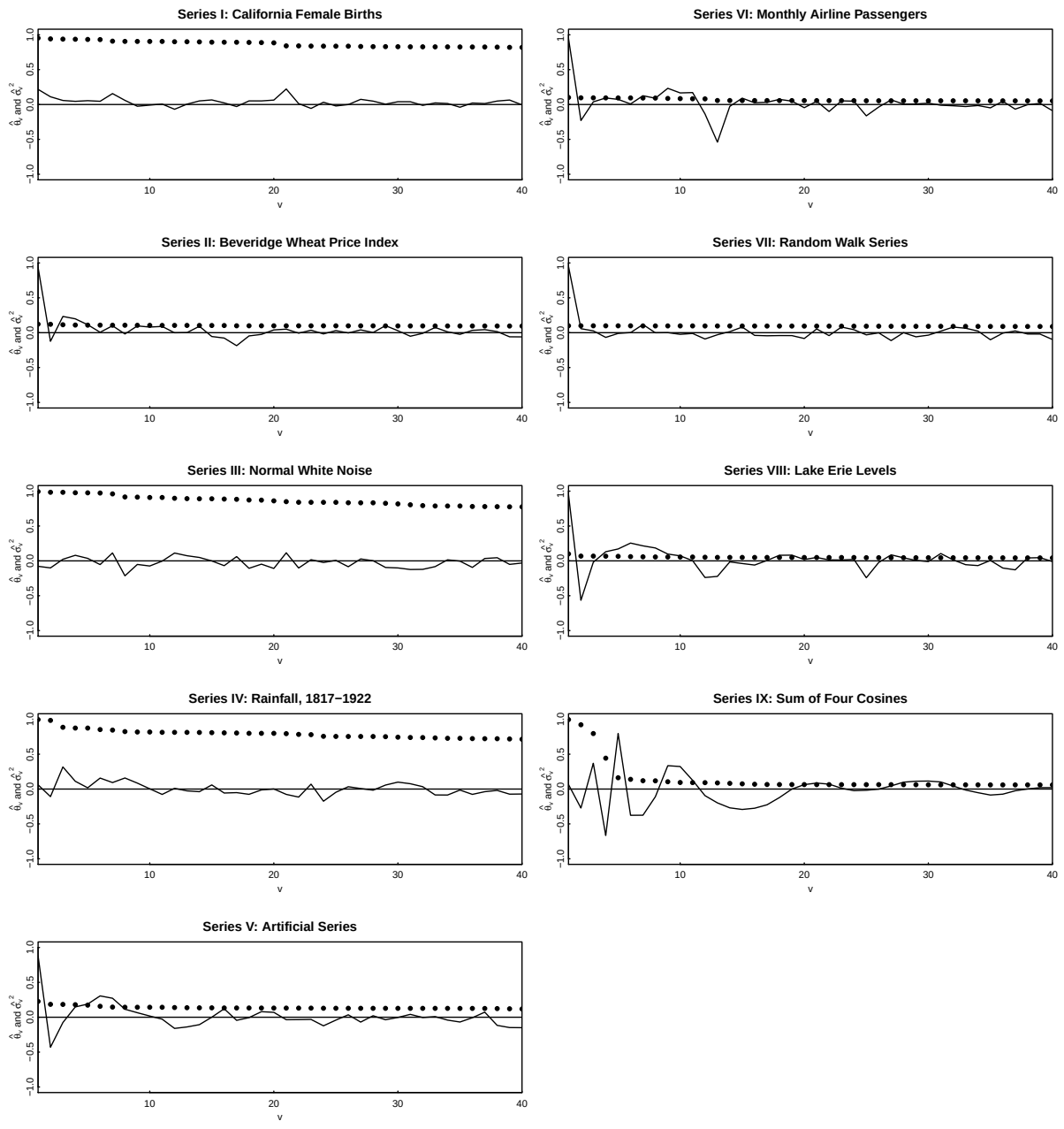


Figure 2.3: The partial autocorrelations (solid curve) and standardized residual variances (points) of Series I through IX.

2.2.3 The Periodogram and Sample Spectral Density

A recurring theme in science is the study of deterministic phenomena that have been observed over time (music, speech, radio signals, etc.) via harmonic analysis; that is, by decomposing a time record into the sum of sinusoids of various frequencies and amplitudes. Harmonic analysis is also important in the study of random phenomena observed over time. We will study the theoretical aspects of this analysis in Chapter 4 (and Appendix A.2). However, in this introductory section, we will simply motivate the theory by looking at what is called the periodogram of a data set $x_1, \ldots, x_n$. For those who are a bit rusty in trigonometry, we have provided a review of basic trigonometry in Appendix A.1.

The basic idea of harmonic analysis is that is it possible to find a unique set of sinusoids (a cosine plus a sine of the same frequency) that when added together reconstruct the mathematical object being studied. In other words, a complicated object can be studied by breaking it down into these simple "frequency components" and studying them separately. Furthermore, in a certain sense, conclusions made about one sinusoid are independent of conclusions made about others (see, for example, Exercise 2.1). A similar idea is found in analysis of variance in traditional statistics. Orthogonal contrasts decompose the effects of different treatments into the sum of linear, quadratic, and higher-order effects that are orthogonal, or statistically independent.

The mathematical objects that can be studied by harmonic analysis vary widely. Examples include sequences of finitely many numbers, infinitely long sequences, functions that are periodic, and general continuous or even discontinuous functions. No matter what object is being studied, the results are always of the form of a decomposition into independent sinusoids that can be studied separately.

Definition 2.3 SINUSOID.
A **sinusoid $g_k(x)$ of period k** *(or* **frequency k^{-1}***), is the function*

$$g_k(x) = a\cos\frac{2\pi x}{k} + b\sin\frac{2\pi x}{k}, \quad x \in (-\infty, \infty).$$

Since the functions cosine and sine have period 2π, we have that $g_k(x + k\ell) = g_k(x)$, for any integer ℓ. We can also write

$$g_k(x) = C\cos\left(\frac{2\pi x}{k} - \phi\right),$$

where $C = \sqrt{a^2 + b^2}$ and $\phi = \arctan(b/a)$ are called the amplitude and phase of g_k (see Exercise T1.2).

The four panels of Figure 2.4 illustrate the roles the period, amplitude, and phase play in the behavior of a sinusoid.

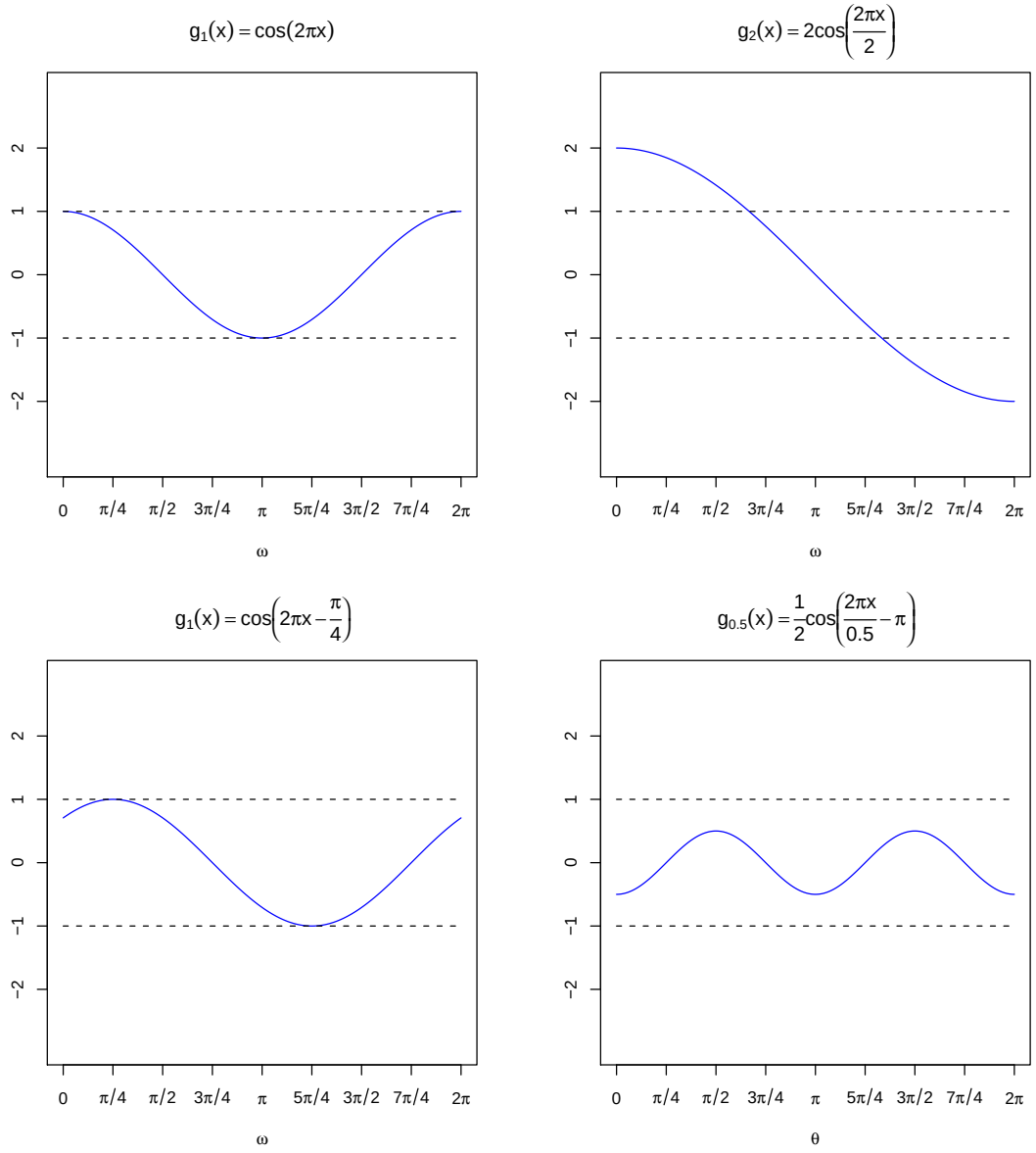


Figure 2.4: Illustration of the roles of amplitude (C), period (k), and phase (ϕ) for four sinusoids. The sinusoid in the top left panel has $C = 1,\ k = 1$, and $\phi = 0$. The sinusoid in the top right panel has $C = 2,\ k = 2$, and $\phi = 0$. The bottom left panel contains a graph of a sinusoid with $C = 1,\ k = 1$, and $\phi = \pi/4$. The bottom right graph is a sinusoid having $C = 0.5,\ k = 0.5$ and $\phi = \pi$. On all plots, the two horizontal reference lines are at -1 and 1, corresponding to an amplitude of one.

which contains four graphs, each of which are a graph of a sinusoid for $x \in (0,\, 1)$, which corresponds to $\omega = 2\pi x \in (0,\, 2\pi)$. As you read the explanation of the graphs, keep in mind that an angle of ω from $(0,\, 2\pi)$ corresponds to a (time) interval of length one.

$$\begin{aligned} g_1(x) &= \cos(2\pi x) \\ g_2(x) &= 2\cos\left(\frac{2\pi x}{2}\right) \\ g_1(x) &= \cos\left(2\pi x - \frac{\pi}{4}\right) \\ g_{0.5}(x) &= \frac{1}{2}\cos\left(\frac{2\pi x}{0.5} - \pi\right) \end{aligned}$$

The graph in the top left panel is simply a standard cosine curve. It ranges (on the vertical axis) between -1 and 1. Since the period (and frequency) is one, the curve completes exactly one cycle across the interval $(0,\, 2\pi)$; that is, there is one completed cycle across one unit of *time*.

The second curve in the top right panel illustrates the effect on the curve of amplitude and period. The amplitude of this curve is $C = 2$. So, instead of ranging (vertically) from -1 to 1, the curve ranges from -2 to 2. Additionally, this curve completes only $1/2$ of one cycle from $(0,\, 2\pi)$. That is because the period of this sinusoid is $k = 2$. In other words, it takes $k = 2$ lengths of the interval $(0,\, 2\pi)$, or two *time periods*, for the curve to complete one cycle. The frequency (number of repetitions of the cosine over the interval $(0,\, 2\pi)$) is the reciprocal of the period; exactly $1/2$ of the curve occurs over $(0,\, 2\pi)$. The curve will achieve the maximum of 2 at $\omega = 0$ and it's minimum at 2π.

The third graph, in the bottom left panel, has an amplitude of one, so the curve ranges from -1 to 1, and a period of $k = 1$, so it completes one full cycle across an interval of length 2π. Notice that the curve in the bottom left is exactly the same as the curve in the top left, with the exception of a horizontal shift. This shift is equal to $k\phi = \pi/4$, illustrating the effect of a non-zero phase of the curve. The result is that the curve is equal to the its maximum (of one) at $\pi/4$ and is equal to it minimum at $5\pi/4$.

The fourth graph has an amplitude of $1/2$, so the curve will range from $-1/2$ to $1/2$. The period of the curve is also $1/2$, so that the curve will complete a cycle in $1/2$ the length of the interval $(0,\, 2\pi)$, or in $1/2$ a time unit. This translates into the curve completing two cycles (having a frequency of two) in the interval $(0,\, 2\pi)$, or in one unit of time. Finally the phase is equal to $\phi = \pi$. The curve is shifted horizontally by the amount $k\phi = \pi/2$. Thus the curve reaches its maximum of $C = 1/2$ at $\pi/2$ and $3\pi/2$, and its minimum of $-1/2$ at $0, \pi$, and 2π.

It is traditional to discuss angles in radians, as we did in the discussion above. However, to make the connection between frequency domain analysis easier to interpret in terms of the time series, from this point on, we will by-pass the mental arithmetic required in the previous discussion of sinusoids, and define frequencies ω to be on the interval $(0,\, 1)$. We now present the next theorem, which describes the orthogonality of sinusoids. This theorem is easily proved using trigonometric identities (see Appendix A.1 for a review).

Theorem 2.1 ORTHOGONALITY OF SINUSOIDS.
Let $\omega_k = (k-1)/n$, for $k = 1, \ldots, [n/2]+1$. Then

(a) $\sum_{t=1}^{n} \cos 2\pi(t-1)\omega_j \cos 2\pi(t-1)\omega_k = \begin{cases} 0 & \omega_j \neq \omega_k \\ n & \omega_j = \omega_k = 0,\ 0.5 \\ n/2 & \omega_j = \omega_k \neq 0,\ 0.5 \end{cases}$

(b) $\sum_{t=1}^{n} \sin 2\pi(t-1)\omega_j \sin 2\pi(t-1)\omega_k = \begin{cases} n/2 & \omega_j = \omega_k \neq 0, \frac{1}{2} \\ 0 & otherwise \end{cases}$

(c) $\sum_{t=1}^{n} \cos 2\pi(t-1)\omega_j \sin 2\pi(t-1)\omega_k = 0$, *for all* ω_j, ω_k

(d) $\sum_{t=1}^{n} \cos 2\pi(t-1)\omega_j = \begin{cases} n & \omega_j = 0 \\ 0 & otherwise \end{cases}$

(e) $\sum_{t=1}^{n} \sin 2\pi(t-1)\omega_j = 0$ *for all* ω_j.

For the interested reader, the proof of this theorem is found in Appendix B.1.

The final piece of the puzzle in the harmonic analysis of a series of numbers is an important mathematical quantity called the discrete Fourier transform of a set of numbers, which we now define.

Definition 2.4 DISCRETE FOURIER TRANSFORM.
The **discrete Fourier transform (DFT)** *of the (possibly complex) numbers $x_1, \ldots, x_n$ is the set of complex numbers $z_1, \ldots, z_n$ given by*

$$\begin{aligned} z_k &= \sum_{t=1}^{n} x_t e^{2\pi i(t-1)\omega_k} \\ &= \sum_{t=1}^{n} x_t \cos 2\pi(t-1)\omega_k + i \sum_{t=1}^{n} x_t \sin 2\pi(t-1)\omega_k, \end{aligned} \tag{2.3}$$

where $\omega_k = (k-1)/n,\ k = 1, \ldots, n$.

If a minus sign is inserted into the exponent in equation (2.3), the result is called the **inverse discrete Fourier transform (IDFT)**.

If one was to find the IDFT of the DFT of x, the result would be nx (see Exercise T2.3).

A theorem known as the "Sinusoidal Decomposition of Data," (Theorem 2.3) states that a time series data set, $x_1, x_2, \ldots, x_n$ can be decomposed into a set of orthogonal sinusoids for frequencies $1/n, 2/n, \ldots, [n/2]/n$. Moreover, there is a decomposition of variability in the xs (as measured by $\hat{\sigma}^2$) in terms of the sum of squared amplitudes of those sinusoids. In terms of understanding why the xs vary then, these squared amplitudes play an important role. The theorem is somewhat complex, so the statement is postponed until the more mathematically advanced Section 2.6. The end result lends itself to the definition of the periodogram of x.

Definition 2.5 PERIODOGRAM.
For a time series data set $x_1, \ldots, x_n$, let $\omega_k = (k-1)/n,\ k = 1, \ldots, [n/2]+1$, and define

$$C_k^2 = \frac{1}{n^2}\left|\sum_{t=1}^{n} x_t e^{2\pi i(t-1)\omega_k}\right|^2, \quad k = 1, \ldots, [n/2]+1.$$

A plot of nC_k^2 versus ω_k is called the **periodogram of $\boldsymbol{x}$**. *The function*

$$\hat{f}(\omega) = \begin{cases} \frac{1}{n}\left|\sum_{t=1}^{n} x_t e^{2\pi i(t-1)\omega}\right|^2, & \omega \in [0,\ 0.5] \\ \hat{f}(1-\omega), & \omega \in [0.5,\ 1] \end{cases}$$

is called the **sample spectral density function of $\boldsymbol{x}$**.

Note that the periodogram is the sample spectral density function evaluated at the so-called "natural frequencies" $\omega_1, \omega_2, \ldots, \omega_{[n/2]+1}$. Note that $C_k^2 = |z_k|^2/n^2$.

Interpreting the Periodogram

Table 2.1: Interpreting the periodogram.

Appearance of Data	Nature of the Periodogram
Smooth	Excess of low frequency; that is, the amplitudes of sinusoids of low frequency (long period) are large relative to other frequencies
Wiggly	Excess of high frequency
Random (no pattern)	No frequencies dominate
Basically sinusoidal of period p time units	A peak at frequency $1/p$
Periodic of period p but not sinusoidal	A peak at fundamental frequency $1/p$ and peaks at some multiples of $1/p$ (harmonics) (See Exercise C2.6 for an example)

In Figure 2.5 we give a plot of three data sets of length 200 and the natural logarithm of their periodograms. The data sets were constructed by letting

$$x_t = \alpha \cos\frac{2\pi(t-1)}{100} + \beta \cos\frac{2\pi(t-1)}{10} + \delta \cos\frac{2\pi(t-1)}{4}; \tag{2.4}$$

that is, x_t is the sum of pure cosines of frequencies 1/100, 1/10, and 1/4. The three series were obtained by varying the coefficients α, β, and δ. Specifically, Series 1 has (10,3,1), Series 2 has (3,3,3), and Series 3 has (1,3,10).

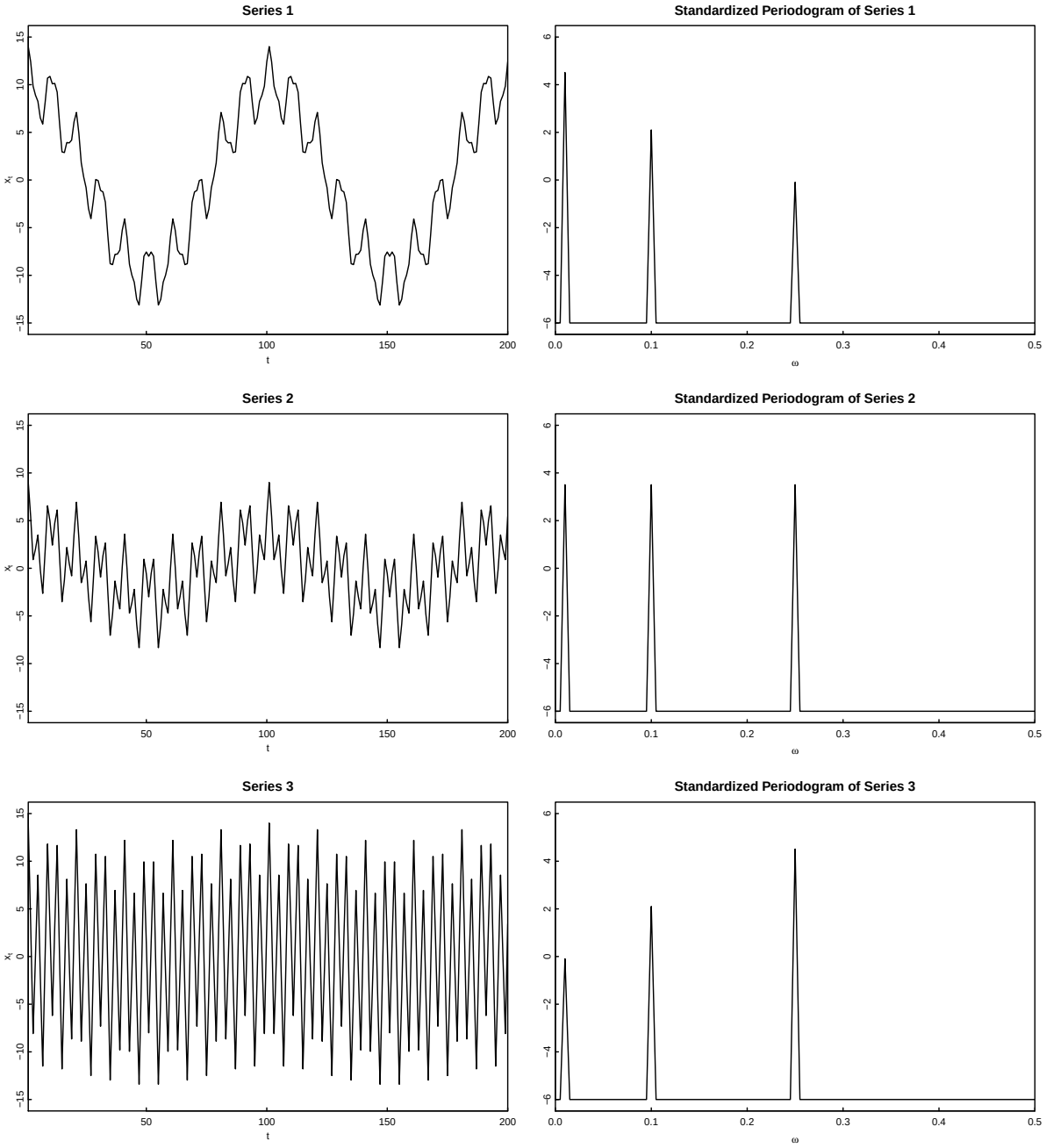


Figure 2.5: Sums of pure cosines and the natural logarithm of their periodograms.

Recall that a sinusoid of frequency ω is the sum of a sine and a cosine of that frequency, but for simplicity, we are considering sinusoids that have no sine component. A sinusoid of long period (low frequency) is very smooth in appearance relative to one of short period (high frequency). Thus when α is large relative to β and δ (such as in Series 1), we would expect x to be relatively smooth in appearance; that is, the long-term rise and fall of the data should be large relative to the short term oscillations. On the other hand, Series 3 appears very wiggly since δ is large relative to α and β. In Series 2, the values of α, β, and δ are all equal to three, and the resulting data are in between Series 1 and 3 in terms of wiggliness.

Now note how the periodograms pick out the amplitudes of the components of the sinusoids. The horizontal axis of the periodogram represents the frequencies $\omega \in (0, 1/2)$. The three frequencies that were used to construct the series were 1/100, 1/10, and 1/4. Over each of these frequencies, the periodogram has a peak with a height equal to $\log(nC^2/\hat{\sigma}^2)$. So if we have the value of the series sample variance $\hat{\sigma}^2$, we could take the values of the heights of these peaks, and their corresponding frequencies, and reconstruct the sinusoid that resulted in the series. Of course, "real" time series data sets are not made up of pure sums of only a few sinusoids. There is noise in the observations. (See Computational Exercises C2.14 and C2.17.) However, with this example, as motivation, we can make the qualitative statements given in Table 2.1.

In Figure 2.6 we give the plots of the natural logarithm of a standardized form of the periodograms of the nine univariate series that we have been discussing throughout the chapter. (Frequency domain analysis of bivariate series will be discussed in more detail in Chapter 8.)

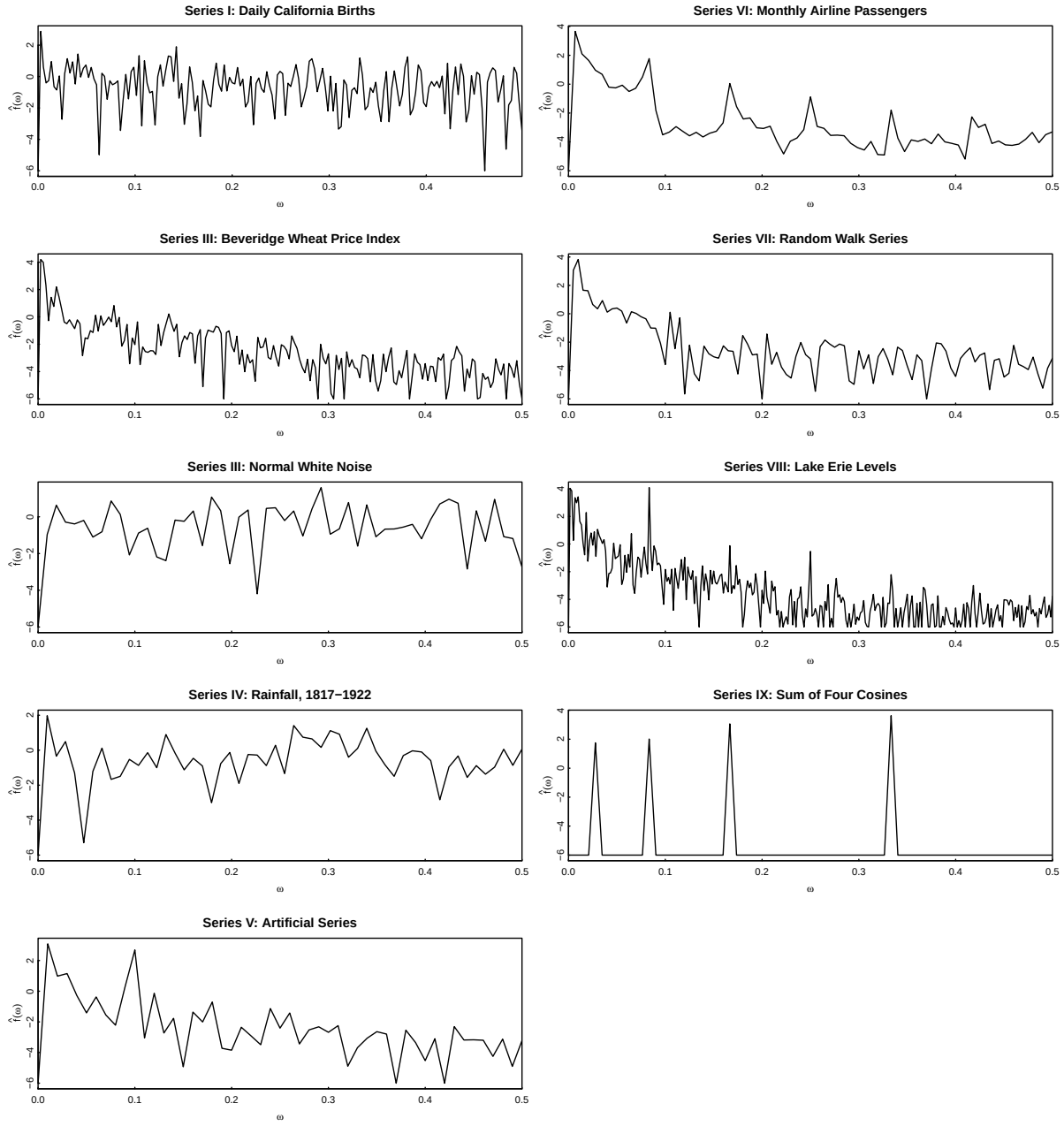


Figure 2.6: The natural logarithm of the standardized periodograms of Series I-IX.

Note how the graphs confirm the discussion above. In particular, note the harmonics in the periodogram of the airline data, the random pattern in those for Series III and IV, the excess of low frequency (long period) in Series VIII, the periodicity in the birth data, and how the periodogram is able to tell that Series IX is, in fact, the sum of four pure cosines.

Displaying the Periodogram

It is often the case that the periodogram (as given in Definition 2.5) has a few very large values that dwarf the rest of the values in the plot. For this reason, the natural logarithm of the values is plotted rather than the values themselves. This allows us to see other values of interest. In Chapter 6 we will see other reasons for plotting the log periodogram (Theorem 6.1). We would also like to plot the log of the periodogram on some standard scale so that several such plots can be compared in a meaningful way. Part (c) of Theorem 2.3 gives that the sum of the second through the n^{th} squared amplitudes gives the sample variance. This provides a convenient scaling factor, and yields the following theorem.

Theorem 2.2 Standardizing the Periodogram.
Let nC_k^2 for $k = 1, 2, \ldots, n$ be the periodogram ordinate of a time series $x_1, \ldots, x_n$, and let $\hat{\sigma}^2 = n^{-1}\sum_{t=1}^{n}(x_t - \bar{x})^2$. Then

$$\frac{1}{n}\sum_{k=2}^{n}\frac{nC_k^2}{\hat{\sigma}^2} = 1;$$

that is, the average value of $nC_k^2/\hat{\sigma}^2$ is one.

Because of this theorem, we will routinely display

$$\log\left(\frac{nC_k^2}{\hat{\sigma}^2}\right) \text{ versus } \frac{k-1}{n}, \quad k = 1, 2, \ldots, \left[\frac{n}{2}\right] + 1,$$

with the graph truncated at -6 and 6 on the vertical axis. It is rare to encounter a value of $nC_k^2/\hat{\sigma}^2$ that is greater than e^6 or less than e^{-6}.

The Sample Spectral Distribution Function

We have seen that data that have no obvious trends, cycles, or serial correlation should have a periodogram that oscillates roughly around a constant. On the other hand, smooth (wiggly) data have an excess of low (high) frequency, while cyclic data will have peaks in their periodogram. One useful way to display these ideas graphically is via another statistic known as the sample spectral distribution function.

Definition 2.6 Sample Spectral Distribution Function.
Let $\hat{f}(\omega_1), \ldots, \hat{f}(\omega_Q)$ be the sample spectral density function of data $x_1, \ldots x_n$ at the frequencies $\omega_j = (j-1)/Q,\ j = 1, \ldots, Q$. Let $q = [Q/2] + 1$ and

$$\hat{F}(\omega_k) = \frac{\sum_{j=1}^{k}\hat{f}(\omega_j)}{\sum_{j=1}^{q}\hat{f}(\omega_j)}, \quad k = 1, \ldots, q.$$

Then $\hat{F}(\omega_1), \ldots, \hat{F}(\omega_q)$ *is called the* **sample spectral distribution function of** $\boldsymbol{x}$.

If $Q = n$, then the $\hat{f}$'s are the periodogram ordinates and the $\hat{F}$'s could be called the cumulative periodogram of x. Note that $\hat{F}(\omega_q) = 1$, while if $\bar{x} = 0$, we have $\hat{f}(\omega_1) = 0$ so that $\hat{F}(\omega_1) = 0$. Therefore,

$$0 \leq \hat{F}(\omega_j) \leq 1, \quad k = 1, \ldots, q,$$

and the spectral distribution function of random data will fluctuate about the straight line that connects the points $(0, 0)$ and $(0.5, 1)$, that is, the line $y = 2x$.

In Figure 2.7 we display the cumulative spectral density of our example data sets. Note that these graphs are much less variable than their corresponding log periodograms. Also an excess of low (high) frequency means that the cumulative periodogram starts out above (below) the line $y = 2x$, before catching up to it before $\omega = 0.5$. Note too that peaks in the periodogram translate into jumps in the cumulative periodogram (such as at frequency 1/12 in the airline data) and the cumulative periodogram of the white noise series (Series III and IV) vary little from the line $y = 2x$.

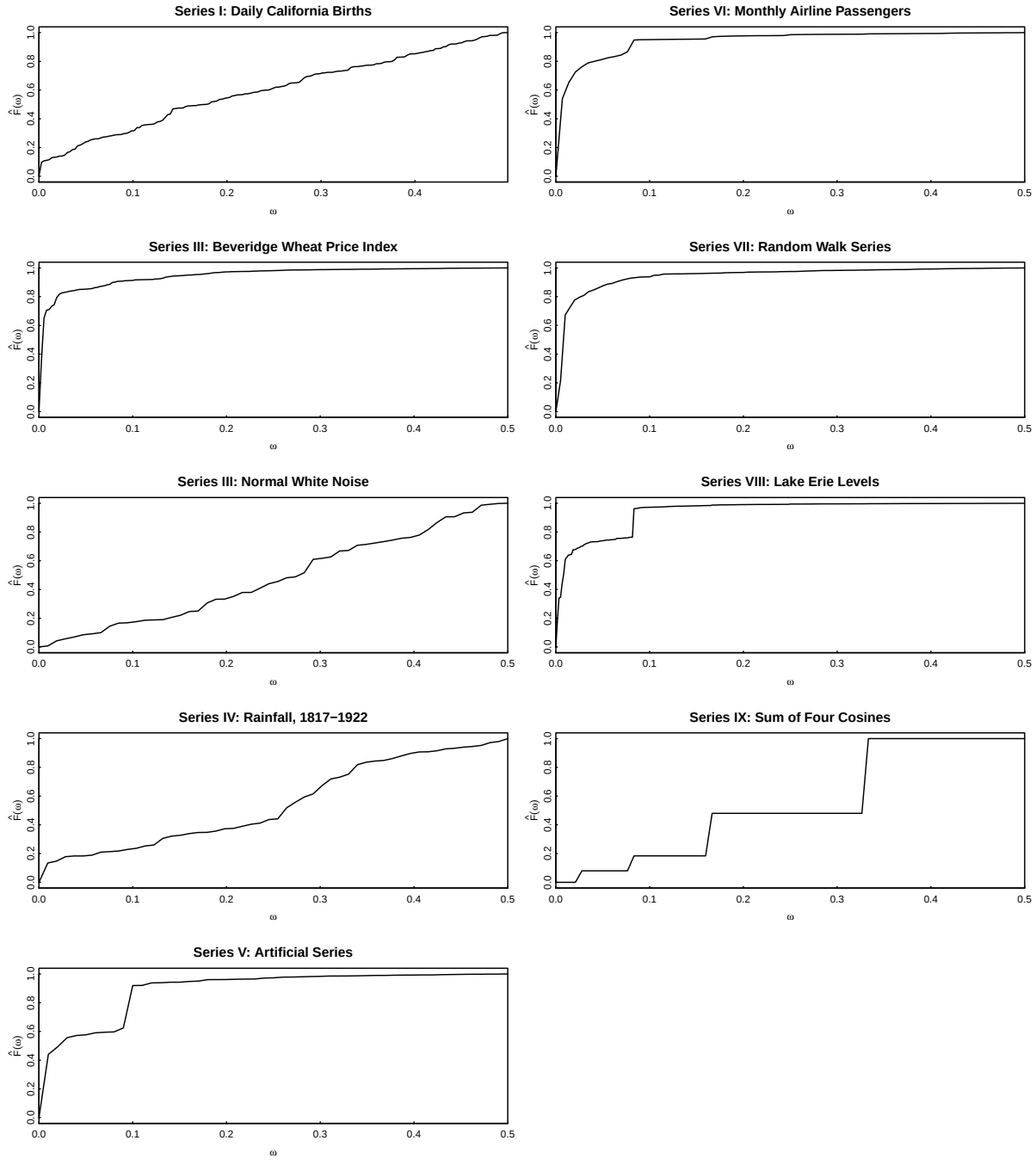


Figure 2.7: Cumulative periodograms for Series I-IX.

2.2.4 The Relationship Between the Correlogram and the Periodogram

Recall that the sample autocorrelation coefficient of lag v for data x is given for

$$\hat{\rho}_v = \frac{\sum_{t=1}^{n-|v|}(x_t - \bar{x})(x_{t+v} - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2}, \quad |v| < n.$$

Then $\hat{\rho}_v$ can be written as

$$\hat{\rho}_v = \frac{\hat{R}_v}{\hat{R}_0}, \quad |v| < n,$$

if we define the **sample autocovariance function** $\hat{R}$ by

$$\hat{R}_v = \frac{1}{n}\sum_{t=1}^{n-|v|}(x_t - \bar{x})(x_{t+v} - \bar{x}), \quad |v| < n.$$

There are very specific mathematical relationships among $\hat{\rho}, \hat{R}$, and the sample spectral density $\hat{f}$. For the more advanced reader, the details of those relationships are provided in Theorem 2.4 of Section 2.6. In Section 2.2.4, we make use of those relationships to describe how an algorithm called the fast Fourier transform (FFT) can be used to calculate the correlogram and the periodogram much more rapidly than a straightforward implementation of their defining formulas.

The Fast Fourier Transform (FFT)

To calculate the discrete Fourier transform

$$w_k = \sum_{t=1}^{n} x_t e^{2\pi i(t-1)(k-1)/n}, \quad k = 1, \ldots, n,$$

of data $x_1, \ldots, x_n$ appears to require n^2 multiplications and additions (n for each of the values of k) and many evaluations of complex exponentials. In the mid-1960s, researchers made use of a variety of trigonometric identities to obtain algorithms called fast Fourier transform (FFT) algorithms (see Brigham (1988)) that essentially require only $n(p_1 + \cdots + p_k)$ multiplications and additions, where $p_1, \ldots, p_k$ are the prime factors of n, and a greatly reduced number of evaluations of complex exponentials. For example, if $n = 1024 = 2^{10}$, then the number of operations is $1024(2 + \cdots + 2) = 2 \times 1024 \times \log_2 1024 \approx 20,000$ as opposed to $1024^2 \approx 1,048,576$, a savings of a factor of about 50. In Appendix A.3, we describe some of the details of how the FFT works.

Note that if n is not very composite, that is, it has some large prime factors, then the FFT saves very little time over a straight forward DFT. The documentation regarding using the FFT in *R* says, "The FFT is fastest when the length of the series being transformed is highly composite (i.e., has many factors). If this is not the case, the transform may take a long time to compute and will use a large amount of memory." Modern day computing

power make this statement ambiguous at best. Consider, for example a series of length 12,637, which is a prime number. Using the `fft` function to compute the discrete Fourier transform of a series of this length takes less than one second.

The FFT and Convolutions

Let $z_1, \ldots, z_n$ be a data set such that $\bar{z} = 0$. The FFT is also useful in calculating the autocovariance function $\hat{R}_z$, which is defined to be the convolution

$$\hat{R}_{z,v} = \frac{1}{n} \sum_{t=1}^{n-|v|} z_t z_{t+|v|}, \quad |v| < n.$$

If we use this formula to find $\hat{R}_{z,0}, \ldots, \hat{R}_{z,M}$, we need

$$n + (n-1) + \cdots + (n-M) = (M+1)n - \frac{M(M+1)}{2}$$

multiplications and additions. If, on the other hand, we pad z with $r \geq M$ zeros such that $Q = n + r$ is the next power of 2 greater than or equal to $n + M$, then we can use part (c) of Theorem 2.4 to get the desired quantities by doing tow FFTs of length Q, that is, in $2Q\log_2(Q)$ multiplications and additions plus some evaluations of complex exponentials. Consider again $n = 12,637$, which is prime, and $M = 500$. $Q = 2^{14} = 16,384$ is the power of 2 that is greater than $12,637 + 500 = 13,137$. We would require $2(16,384)(14) = 458,752$ operations to do the two FFTs versus $6,331,137 - 125,250 = 6,205,887$ operations using the convolution formula. In Example 2.12, we provide a function for experimenting with which method is quicker for various values of n and M.

There are two functions in *Timeslab in R* that can be used for computing the sample autocorrelation function. The function `corr` uses the FFT method just described. On the other hand, `corr1` uses the convolution method. With the exception of the functions' names, the syntax for using both functions, and the values they return, are identical. Section 2.4 contains an explanation for using the `corr` function.

2.3 Describing the Distribution of Data

There are a variety of methods for providing a graphical description of the distribution of a data set in elementary statistics. We will discuss three in this section. The first is the most popular, known as the histogram, the second is what Parzen (1983a) calls an informative quantile plot, and the last is a kernel probability density estimator.

A word of caution is in order. The methods below are well-defined for independent, identically distributed data – which come from the same population, and so have the same underlying distributional structure. Thus, when applying them to a time series data set, they are most appropriate when the time series is white noise. If the time series is not white noise, the use and interpretation of these methods is not necessarily proper, and must be done advisedly. The reasons for this will be addressed briefly in Chapter 3, and more specifically in Chapter 4.

2.3.1 Histograms

Simply put, a histogram is a bar graph of the data. The range of the data is divided into a set of non-overlapping intervals, usually of equal width. The bars are centered on the midpoints of the intervals, and the heights of the bars are the proportion of values in the sample that fall within the interval. There are many opinions on how one should best choosing the width of the intervals, or equivalently the number of intervals. The *R* function `hist` gives the user three choices. The first of these, which is also the default, is the "Sturges" method (Sturges 1926), which computes the number of intervals based on the range of the data $r_x = \max(x_1, \ldots, x_n) - \min(x_1, \ldots, x_n)$. The approach by Sturges computes the width of the intervals as approximately

$$\frac{r_x}{[1 + \log_2(n)]},$$

where $[\cdot]$ is the greatest integer value. A second method for finding the number of intervals for a histogram is due to Scott (1979), who proposes to computes the number of intervals based on the estimate of the standard deviation s_x of the data. Specifically, if s_x is not zero, then the approximate number of intervals is computed using

$$\left[\frac{r_x}{3.5 s_x} n^1/3 + 1\right].$$

If s_x is zero, then the number of intervals is one. The third available method is the "Freedman-Diaconis" method (Freedman & Diaconis 1981). This choice uses the interquartile range (IQR), unless that is zero. In that case, the approximate number of intervals is

$$\left[\frac{2IQR}{n^{1/3}}\right].$$

If the IQR is zero, then the Freedman-Diaconis rule replaces IQR with the mean absolute deviation. If that is zero too, then the number of bins is one.

2.3.2 Informative Quantile Plot

An informative quantile plot is a plot of

$$y_{(i)} = \frac{x_{(i)} - \tilde{m}}{2IQR} \quad \text{versus} \quad u_{(i)} = \frac{i - 0.5}{n}, \quad i = 1, 2, \ldots, n,$$

where $\tilde{m}$ is the median of the data, and $x_{(i)}$ is the i^{th} smallest x, and the values plotted on the vertical axis are truncated to fall between -1 and 1. The values $y_{(i)}$ are one means of computing the standardized quantiles of the $x_{(i)}$. There are many others. The interested reader is referred to the *R* help for the `quantile` function, and the references therein. When an informative quantile plot is produced, a reference line is placed on the plot. This line is the true quantile plot of a U(0,1) random variable. One can get an idea of what the informative quantile plot should look like for various distributions by generating a sample via the `wn` function and then calling `infqnt`. See Example 2.13. The *Timeslab in R* function `infqnt(x)` creates the informative quantile plot of `x` which is a vector containing the data $x_1, \ldots, x_n$.

2.3.3 Kernel Density Estimation

As computing technology has advanced and become common place, highly computational statistical methods have become more mainstream. One such method is kernel density estimation. Kernel density estimation is a nonparametric method, since it makes no assumptions about the particular functional form of the density. The histogram of a data set is an extremely elementary form of a kernel density estimator. It divides the range of the data into non-overlapping intervals and counts how many data values are in each of the intervals. Then a bar graph of the relative frequencies (counts divided by sample size) is formed. Thus we can think of the height of the bar for a particular interval as the average number (or proportion) of the total data that are in the interval. In a ore general kernel density estimator, the range is also divided into intervals, but these intervals are allowed to overlap, and the aim is to estimate the density at the midpoint of each interval. Instead of just counting the number of points in the interval, the method assigns a number between zero and one to each point in the interval with a data point at the center of the interval getting the highest possible value. The farther from the center that a point is, the less weight it is given. The function that is used to determine these "weights" is called a kernel, and thus the method is called the kernel method. The intervals are determined by specifying their center point and their width; that is, the i^{th} interval is given by $[y_{(i)} - h,\, y_{(i)} + h]$, where the number h is called the **bandwidth**. Notice that the width of an interval is not h, but rather $2h$.

Formally then, the density estimator at a point $y_{(i)}$ is given by

$$\hat{f}(y_{(i)}) = \frac{1}{nh} \sum_{j=1}^{n} W\left(\frac{y_{(i)} - x_{(j)}}{h}\right).$$

where the kernel W is a function that is nonnegative and symmetric about zero. Thus to form a kernel density estimate, one must specify: (1) the number of points at which to find the estimator, and (2) the kernel to be used, and (3) the bandwidth which is to be used. Table 2.2 contains a list of the kernels that are available in *R* and *Timeslab in R*. Note that each of the kernels in Table 2.2 is defined for $|t| \leq 1$ and is zero for $|t| > 1$. In fact, each of them is a probability density function on $[-1, 1]$ as they are positive and integrate to one. Thus the resulting density estimators are comparable.

To use the `tsdensity` function, the call is

```
tsdensity(x, rbins, npts = 501, kernel = "parzen")
```

A detailed description of using `tsdensity` is given in Section 2.4. Of interest in our current discussion is the argument `rbins`, which is a real scalar that determines the bandwidth to be used. If the value entered is positive, then the bandwidth that is used is given by

$$h = \frac{r_x}{2\texttt{rbins}}. \tag{2.5}$$

Specifying the bandwidth in this indirect way means that the user does not need to know the range of the data prior to issuing the `tsdensity` function. A large value of `rbins` will

Table 2.2: Kernels available in *R* and *Timeslab in R*. To compute a kernel density estimate using *Timeslab in R*, the function call is `tsdensity(x, kernel = argument-name)`; In *R*, the function call is `density(x, kernel = argument-name)`. In both instances, "`argument-name`" is replaced by the name or number below in the "Kernel argument in function call" column of the table.

Software	Kernel argument in function call	Kernel name	Kernel function $W(t)$
Timeslab	`"rectangular"` or `1`	Rectangular	$\frac{1}{2}$
R	`"rectangular"`		
Timeslab	`"cosine"` or `2`	Cosine	$\frac{1}{2}\{1+\cos(\pi t)\}$
R	`"optcosine"`		
Timeslab	`"epanechnikov"` or `3`	Epanechnikov	$\frac{3}{4}\left(1-t^2\right)$
R	`"epanechnikov"`		
Timeslab	`"biweight"` or `4`	Biweight	$\frac{15}{16}\left(1-t^2\right)^2$
R	`"biweight"`		
Timeslab	`"triangular"` or `5`	Triangular	$1-\lvert t\rvert$
R	`"triangular"`		
Timeslab	`"gaussian"` or `6`	Gaussian	$1.012339e^{-\pi t^2}$
R	Default		
Timeslab	Default	Parzen	$\begin{cases} \frac{4}{3}-8t^2+8\lvert t\rvert^3, & \lvert t\rvert \le 0.5 \\ \frac{8}{3}\{1-\lvert t\rvert\}^3 & \lvert t\rvert > 0.5 \end{cases}$
R	Not available.		

lead to a small value of h, and the resulting estimate will appear wiggly. If a negative value, or a value of zero, is provided for `rbins`, then the `tsdensity` function will use

$$\texttt{rbins} = \frac{r_x n^q}{3.5 s_x}, \tag{2.6}$$

where q is 1/3 for the rectangular kernel, and 1/5 for the others. This translates to a bandwidth that is proportional to $n^{-1/3}$ for the rectangular window, or $n^{-1/5}$ for the others.

The `density` function in *R* uses the Gaussian kernel as the default kernel. The choice of bandwidth is also performed in a different manner than in `tsdensity`. For example, for the Gaussian kernel, the bandwidth h is to taken to be

$$h = 4 \times 1.06 \min\left(\hat{\sigma},\ \frac{\text{IQR}}{1.34}\right) n^{-1/5},$$

in accordance with equation (5.5) on page 130 of Venables & Ripley (2002). We will examine the relationships between the bandwidth and kernel function in a bit more detail when we discuss another use of kernel estimation in Section 6.3.

Most users of kernel density estimators agree that the choice of the kernel function is not as important as the choice of the bandwidth h. Choosing an optimal bandwidth is a topic of

much discussion in the literature. Just a smattering of important work in this area is found in Parzen (1962a), Tapia & Thompson (1978), Silverman (1986), Sheather & Jones (1991), and Scott (1992). There is much more literature about the choice of bandwidth and kernels.

Figure 2.8 contains a plot of a transformed version of the Beveridge wheat price index, its histogram with the number of intervals chosen by the `hist` function, its informative quantile plot, and three density estimates using the Parzen kernel for values of `rbin`, or accordingly, three bandwidths. The specific transformation applied to the data is the "first difference". This type of transformation is effective for removing linear trends. We will discuss differencing in more detail in Section 3.1.2. The informative quantile plot of a normally distributed sample should intersect the lower left and upper right corners of the plot (see Computational Exercise C2.18). For the data that we are considering, the plot shows clearly the presence of too many large and small values for the data to be consistent with normality, even though the density estimates are clearly symmetric. See Example D.1 for another illustration of the use of the `density` function.

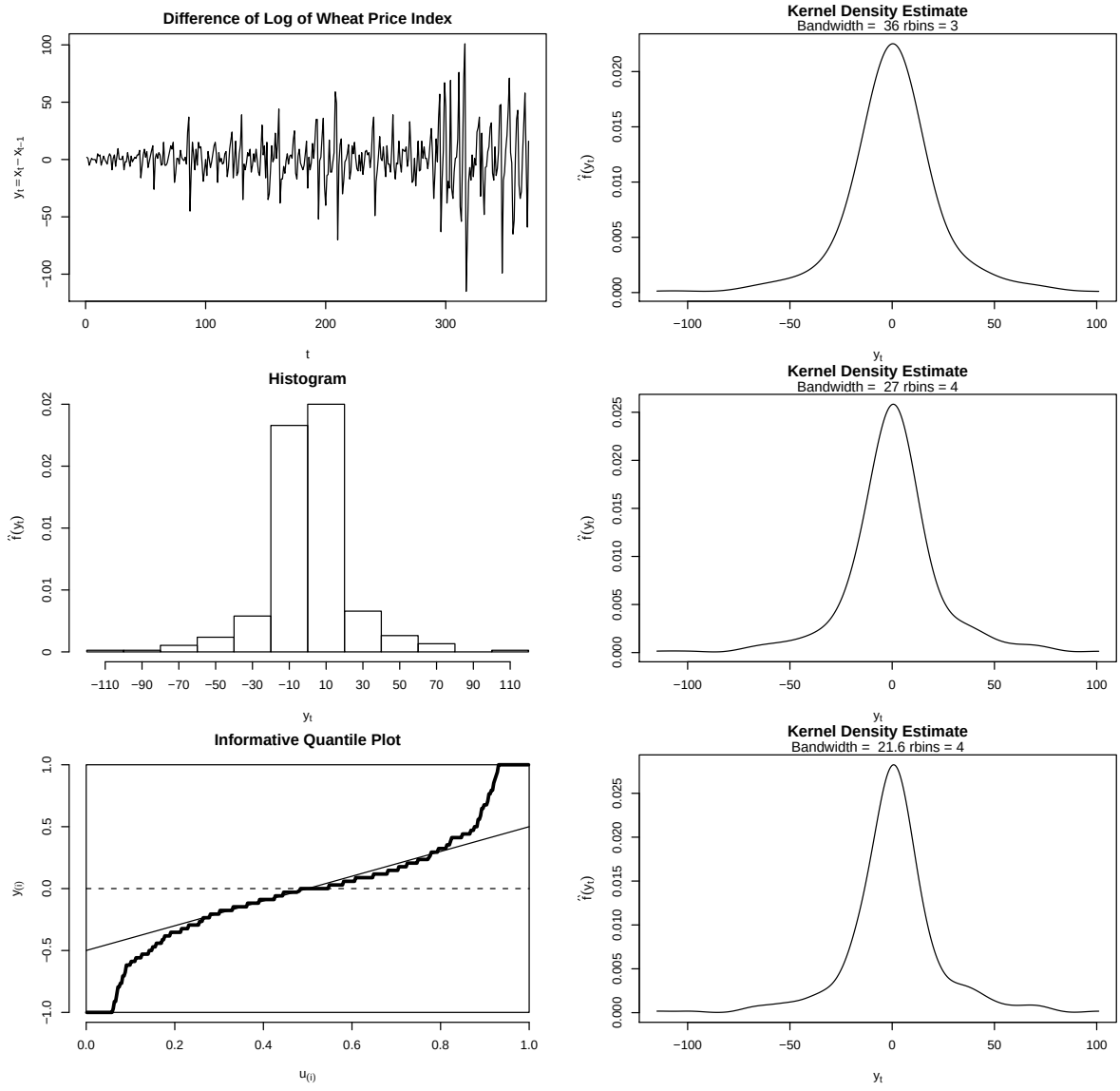


Figure 2.8: Studying the distribution of the first difference of the Beveridge Wheat Price Index.

2.4 Using *Timeslab in R* and *R*

2.4.1 *Timeslab in R* Functions

There are several functions in *Timeslab in R* for computing and plotting the descriptive statistics described in this Section. Below we explain how to use each of them, and provide some examples.

The `corr` Function

The call to the *Timeslab* function `corr(x, m = 0)` returns the variance of the time series. If `m` is a positive integer, then `corr` returns a list containing two items.

`var`: a scalar containing the variance of the time series and,

`corr`: a vector of length `m` containing the sample autocovariances for lags $1, \ldots, $`m`.

Example 2.1 Computing the Autocorrelation of the Airline Data.
Refer back to Example 1.1. The variable `x.air` *created in that example contains the airline data. To compute the variance of the airline data, simply type*

```
corr(x.air)
```

To compute and plot the sample autocorrelation function of the airline data, issue the following commands.

```
air.corr <- corr(x.air, 40)$corr
plot(air.corr, type = "l", ylim = c(-1, 1), xlab = "Lag",
  ylab = "Sample Autocorrelation",
  main = "Correlogram of Airline Data")
```

Equivalently, you could type

```
plot(corr(x.air, 40)$corr, type = "l", ylim = c(-1,1),
  xlab = "Lag",ylab = "Sample Autocorrelation",
  main = "Correlogram of Airline Data")
```

To see a listing of the values of the autocorrelation you need only type

```
corr(x.air, 40)
```

The `corr` function calculates the sample autocorrelations using the "two FFT method" described in Section 2.2.4 and does the padding with zeros automatically. The input data is not affected in any way. If `m = 0`, then the value of the object `corr` in the list is a `NULL` value.

Finally, as mentioned at the end of Section 2.2.4, *Timeslab in R* contains a second function called `corr1` that also computes the sample autocorrelation function for a time series contained in `x` for up to lag `m`. The primary difference in the two functions is that `corr` uses a two-FFT method, while `corr1` uses a convolution method. With the exception of the functions' names, the syntax and the objects in the list returned by `corr1` are identical to those of `corr`.

The corr2 Function

The function corr2 computes the sample cross-correlation between two series. The function has three required arguments. The first two, x and y, are real vectors of equal lengths containing the observed time series. The third argument is the integer m that indicates the maximum number of lags. The function returns a list of the following objects.

var.x: a real scalar containing the sample variance of the series x,

var.y: a real scalar containing the sample variance of the series y,

lags: an integer vector of length $2m + 1$ containing the lags $-m$, $-(m-1)$, ..., -1, 0, 1, ..., $m-1$, m, for which the cross-correlation was computed,

cross.corr: a real vector of length $2m + 1$ real scalar containing the sample cross-correlation between the series x and y.

Example 2.2 Computing the Cross-Correlation for the Bivariate Gas Furnace Series.
Below is a listing of a set of commands that will compute and plot the cross-correlation between the two series in the gas furnace data.

```
x <- datasets()$gasfurnin
y <- datasets()$gasfurnout
c2.furn <- corr2(x, y, 20)
lags <- c2.furn$lags
plot(lags, c2.furn$cross.corr, type = "l", ylim = c(-1, 1),
  xlab = "v", ylab = "Cross-correlation",
  main = "Gas Furnace Input and Output")
lines(lags, rep(0, length(lags)))
```

The parcorr Function

The function parcorr has two required arguments, x and m, and returns a vector of sample partial autocorrelations for lags $1, \ldots, m$. The example below shows how to compute and plot the sample partial autocorrelation for lags $1, \ldots, 30$.

Example 2.3 Computing Sample Partial Autocorrelations of the Airline Data.

```
air.part <- parcorr(x.air, 30) plot(air.part, type = "l",
   ylim = c(-1, 1), xlab = "Lag", ylab = expression(hat(theta)[v]),
   main = "Partial Correlogram of Airline Data")
```

or equivalently

```
plot(parcorr(x.air, 30), type = "l", ylim = c(-1,1),
  xlab ="Lag", ylab = expression(hat(theta)[v]),
  main = "Partial Correlogram of Airline Data")
```

The `perdgm` and `plotsp` Functions

The *Timeslab* functions `perdgm` and `plotsp` compute the periodogram and plot the natural logarithm of the standardized periodogram, respectively.

The call to the function to compute the periodogram is

```
perdgm(x)
```

where `x` is the name of the variable containing the time series. The function returns the periodogram at frequencies $(k-1)/n,\ k = 1, 2, \ldots, [n/2]+1$.

The call to the function to plot the periodogram is

```
plotsp(f, n, div, main = "Log Std Spectra")
```

The arguments to this function are

`f`: a vector containing the periodogram.

`n`: an integer containing the length of the time series.

`div`: a real scalar containing the divisor to use in standardizing the periodogram (usually the time series variance).

`main`: a character string containing the title for the plot. The default title is `Log Std Spectra`.

Example 2.4 Computing and Plotting the Airline Data Periodogram.
The code below illustrates how to use the two functions to compute and plot these for the airline data.

```
f.air <- perdgm(x.air)
plotsp(f.air, length(x.air), tsvar(x.air))
```

or equivalently

```
plotsp(perdgm(x.air), length(x.air), tsvar(x.air))
```

The `cumsp` Function

The *Timeslab* function `cumsp(f)` returns the cumulative periodogram when the argument `f` is the periodogram of the sample. The example below illustrates how to compute and plot the cumulative periodogram of the airline data.

Example 2.5 Cumulative Spectra of the Airline Data.
The code for computing and plotting the cumulative periodogram of the airline data is below. As with the other examples, you can separate the one line of code into multiple lines.

```
plot(freqs(length(x.air)), cumsp(f.air), type = "l",
  ylim = c(0, 1), xlab = expression(omega),
  ylab = expression(hat(F)(omega)), main = "Cumulative Periodogram")
```

The example also illustrates the Timeslab *function* `freqs(n)`*, which returns a vector containing the frequencies* $(k-1)/n,\ k = 0, 1, \ldots, [n/2]+1$.

The `tsdensity` Function

The *Timeslab* function `tsdensity` computes a kernel density estimate for a data set `x`. The call to the function is

```
tsdensity(x, rbins = 0, npts = 501, kernel = "parzen")
```

The arguments are

`x`: a real vector containing the data for which the density is to be estimated.

`rbins`: a scalar that determines the bandwidth. If `rbins` is zero or negative, then the value for `rbins` is computed using equation (2.6). The bandwidth is computed using equation (2.5).

`npts`: an integer scalar giving the number of points within the range of `x` at which to compute the kernel density estimate. The default is `npts = 501`.

`kernel`: is a character or integer-valued argument indicating the kernel to be used in the estimate. Acceptable values of `kernel` are provided in Table 2.2. The default kernel function is the Parzen kernel.

The function `tsdensity` returns a list containing the following objects.

`bw`: a real scalar containing the bandwidth.

`y`: a vector of length `npts` containing the values within the range of `x` at which the density estimate was computed.

`fy`: a vector of length `npts` containing the kernel density estimates at the corresponding values in the vector `y`.

Unlike the `hist` and `infqnt` functions, `tsdensity` does not produce a plot of the density that it calculates. To plot it, one can use the *R* function `plot`. Several densities can be superimposed on the same axes using the *R* `lines` function. This allows the user to create informative labels and scales for the plots.

Example 2.6 Estimating the Density of Random Normal Variates.
To see how to use the `tsdensity` *function, try the following series of commands, designed to generate a random sample of size 1000 from a standard normal distribution and save it in the vector* `x`*. Then three kernel density estimates (called* `fhat`*,* `fhat1`*, and* `fhat2`*) are computed. The next four lines find "nice" values for the x- and y-axes. The three lines following (the call to the* `plot` *function and the following two calls to the* `lines` *function) plot the three kernel density estimates on the same set of axes. The final call to the* `lines` *function superimposes the true standard normal density on the axes for the values of* `x`*.*

```
x <- rnorm(1000)

fhat <- tsdensity(x, rbins = 3)
fhat2 <- tsdensity(x, rbins = 4)
fhat3 <- tsdensity(x, rbins = 5)

xmin <- min(-3.5, x)
xmax <- max(x, 3.5)
ymin <- min(fhat$fy, fhat2$fy, fhat3$fy)
ymax <- max(fhat$fy, fhat2$fy, fhat3$fy, dnorm(0))

plot(fhat$y, fhat$fy, type = "l", xlim = c(xmin, xmax),
     ylim = c(ymin, ymax), lty = 2, lwd = 1.5, col = "black",
     xlab = "x", ylab = "Density", main = "Kernel Density Estimates")
lines(fhat2$y, fhat2$fy, lty = 3, lwd = 1.5, col = "blue")
lines(fhat3$y, fhat3$fy, lty = 4, lwd = .5, col = "darkgreen")
lines(sort(x), dnorm(sort(x)), lwd = 2.24, col = "red")
```

2.4.2 *R* Functions

The functions below are not part of *Timeslab in R*. They are part of the standard *R* release. They were discussed in the previous sections, and so we explain their usage here in a bit more detail. For more information, the interested reader is referred to help that comes with *R*.

The `fft` Function

The *R* function `fft` performs the fast Fourier transform of an array. The call to the function is

```
fft(z, inverse = FALSE)
```

The argument `z` is a real or complex array containing the values to be transformed. If `inverse = TRUE`, then the inverse FFT is returned.

The `hist` Function

The *R* function `hist` computes and plots a histogram of the given data values. By default, the histogram plotted by *R* has (1) non-overlapping intervals of equal widths, with (2) the bars' heights equal to the frequencies (counts) of the observations in each interval. A call to the function, with a partial listing of available arguments, `hist` is below. Following that is a description of some commonly used arguments. We conclude with three brief examples.

```
hist(x, breaks = "Sturges", freq = NULL, probability = !freq,
     ..., plot = TRUE, nclass = NULL, ...)
```

There is a long list of arguments available for use in the call to the `hist` function. Many of them are passed to the `plot.histogram` function, and so are graphical in nature. Below, we describe only those arguments that affect the computation of the histogram, and not those that affect the aesthetic appearance of the plot. For more information, the interested reader should consult the *R* documentation by typing `help(hist)` or `help(plot.histogram)` at the *R* prompt.

`x`: a vector of values for which the histogram is desired.

`breaks`: one of

- a vector giving the breakpoints for the intervals of the histogram,
- a single number giving the number of intervals for the histogram,
- a character string naming an algorithm to compute the number of cells. This can be one of `"Sturges"` (the default), `"Scott"`, or `"FD"`. For more information on these algorithms, see Section 2.3.1.
- a function to compute the number of intervals.

In the last three cases, the number computed will be used as an approximation to the actual number of intervals used.

`freq`: a logical argument. If `freq = TRUE`, then the heights of the histogram bars will be the frequency (or count) of the number of data values in each interval. If `freq = FALSE`, then the estimated probability densities are plotted. The value of `freq` will default to true only if the intervals are of equal width, and if `probability` (below) is not specified.

`probability`: a logical acting as an alias for `!freq`. If `probability = TRUE`, then the heights of the histogram bars will be the proportion of data values in the respective intervals.

`plot`: a logical argument, with default `plot = TRUE`. If `plot = FALSE`, the a call to the `hist` function returns an object of class `histogram`, which is a list containing objects as described below, and the plot is not created.

`nclass`: an augment that is equivalent th `breaks` for a scalar or character argument.

If `plot = FALSE`, or the call to the histogram is `varname <- hist(x, ...)`, then function returns an object of class `histogram` which is a list with the following components. (Fore more on classes in *R*, see Appendix D.1.)

`breaks`: a vector of the interval boundaries.

`counts`: for each interval, the number of values of `x` in the interval.

`density`: estimated density values for each interval. If the intervals are all equal width, then the values are the relative frequencies.

`intensities`: the same as `density`.

`mids`: the midpoints of the intervals.

`xname`: a character string with the name of the `x` argument.

`equidist`: a logical, indicating if the distances between the values in `breaks` are all equal.

We now provide three brief examples to illustrate use of the `hist` function.

```
x <- rnorm(200)
hist(x)
```

will create a random sample of size 200, stored in the vector `x`. The call `hist(x)` creates a frequency histogram of `x`. In the next lines of code, we assume the variable `x` contains for which the histogram is desired.

```
hist(x, probability = TRUE)
hist(x, probability = TRUE, plot = FALSE)
xhist <- hist(x, probability = TRUE)
xhist
```

The first line will create a probability (relative frequency histogram) of the data in `x`. The second line returns the values as previously described. The last two lines will create the probability histogram, and save the numerical information in the object `xhist`. Typing `xhist` provides the numerical list.

The `density` Function

The *R* function `density` computes kernel density estimates. Its default method does so with the given kernel and bandwidth for univariate data. The call to the function is

```
density(x, bw = "nrd0", adjust = 1,
        kernel = c("gaussian", "epanechnikov", "rectangular",
                   "triangular", "biweight", "cosine",
                   "optcosine"),
        weights = NULL, window = kernel, width,
        give.Rkern = FALSE, n = 512, from, to, cut = 3,
        na.rm = FALSE, ...)
```

A brief description of the arguments for the `density` function is below. For more details, the reader should consult the *R* documentation.

`x`: the data from which the estimate is to be computed.

`bw`: the smoothing bandwidth to be used. `bw` can also be a character string giving a rule to choose the bandwidth. For more information, see help on the *R* function `bw.nrd`.

`adjust`: the bandwidth actually used in computing the density estimate is actually `adjust*bw`.

`kernel, window`: a character string giving the smoothing kernel to be used. This must be one of `"gaussian"`, `"epanechnikov"`, `"rectangular"`, `"triangular"`, `"biweight"`, `"cosine"`, or `"optcosine"`, and may be abbreviated to a (unique) single letter prefix. The default is `kernel = "gaussian"`.

`weights`: an (optional) numeric vector, the same length as `x`, of non-negative observation weights. The default is `NULL`, and is equivalent to `weights = rep(1/length(x), length = length(x))`.

`width`: an optional argument, equivalent to `bw`.

`give.Rkern`: a logical; if `give.Rkern = TRUE`, then no density is estimated, and the "canonical bandwidth" of the chosen `kernel` is returned instead.

`n`: the number of equally spaced points at which the density is to be estimated. When `n > 512`, it is rounded up to a power of 2 (as in the *R* function `fft`) during the calculations and the final result is interpolated by the *R* function `approx`.

`from, to`: the left and right-most points of the grid at which the density is to be estimated; the defaults are `cut*bw` outside the range of `x`.

`cut`: by default, the values of `from` and `to` are `cut` bandwidths beyond the extremes of the data.

`na.rm`: a logical argument. If `na.rm = TRUE`, then missing values are removed from `x`. If `na.rm = FALSE`, any missing value in `x` will cause an error. The default value is `na.rm = FALSE`.

The value returned by the `density` function is an object of class `density`, whose underlying structure is a list containing the following components.

`x`: the `n` coordinates of the points where the density is estimated.

`y`: the estimated density values.

`bw`: the value of the bandwidth.

`n`: the sample size after eliminating missing values.

`call`: the call which produced the result.

`data.name`: the deparsed name of the `x` argument.

`has.na`: a logical, which is always `FALSE`.

If `give.Rkern = TRUE`, then the function `density` returns the "canonical bandwidth" instead of the list above.

The `plot.density` Function

Unlike the `hist` function, the `density` function does not produce a plot of the estimated density. The `plot.density` function will create the plot. The call to the function is

```
plot(x, ...)
```

where the argument `x` is of class `density`, as returned by the call to the `density` function. Notice that (1) the `.density` is not typed into the call, and that (2) the *class* of the argument `x` to the `plot` argument will dictate what is plotted. For more on classes in *R*, see Appendix D.1.

To illustrate the use of the `density` and `plot` functions together, the reader is encouraged to perform the following sets of commands.

```
x <- rnorm(200)
plot(x, type = "l", xlab = "t", ylab = expression(x[t]),
     main = "Time Plot of Normal White Noise")
density(x)
fhat <- density(x)
plot(fhat, main = "Density Estimate")
```

2.5 Pulling It All Together

Most chapters in this book will end with a section similar in spirit to this one, where we pull together using *Timeslab in R* with concepts discussed in the chapter. The examples in this section will illustrate how to write *R* functions to plot a set of descriptive statistics, generate data that is the sum of four cosines and describe that data, and more. Some of the examples include *R* programming features that are not explained in this text. In those cases, the reader is referred to the *R* documentation.

Example 2.7 Descriptive Statistics.
The function `descplot` *calculates and displays the basic descriptive statistics introduced in Chapter 2 for a univariate time series. If the function were executed on a series generated from calling* `wn(n = 100, seed = 123456)` *for* `m = 20`*, the result would be the graph in Figure 2.9.*

```
descplot <- function(x, m)
#-------------------------------------------------------------
#  R function to compute and plot descriptive statistics of a
#  univariate time series.
#
#  REQUIRED INPUT:
#  x = a real vector vector containing the time series.
#  m = an integer scalar indicating the number of lags for
#      computing the sample autocorrelation and sample
#      partial autocorrelation.
#
#  descplot(x,m) returns a list containing objects:
#
#  mean = a real scalar containing the sample mean.
#  var = a real scalar containing the sample variance.
#  corr = a vector of length m containing the sample
#         autocorrelations.
#  part = a vector of length m containing the sample partial
#         autocorrelations.
#  srvar = a vector of length m containing the standardized
#          residual variances.
#  perd = a vector of length [n/2]+1 containing the periodogram
#  cperd = a vector of length [n/2]+1 containing the cumulative
#          periodogram
#
#  The function also creates the following plots in a single
#  graphics window.
#  1.  A time plot of the data
#  2.  The correlogram
#  3.  The sample partial autocorrelations versus lags up to m
#  4.  The standardized residual variances versus lags up to m
#  5.  The natural logarithm of the standardized periodogram
#  6.  The cumulative periodogram with normal white noise
#      test lower and upper boundary lines.
#-------------------------------------------------------------
{
   corr  <- corr(x, m)$corr
   part  <- parcorr(x, m)
   srvar <- srvars(x, m)
   perd  <- perdgm(x)
   cperd <- cumsum(perd) / sum(perd)
```

```
    sq1 <- 1.36/sqrt(length(cperd))

    par(mfrow = c(3, 2))

    plot(x, type = "l", main = "Time Plot of the Series",
         xlab = "Time (t)", ylab = expression(x[t]))
    plot(corr, type = "l", ylim = c(-1, 1), main = "Correlogram",
         xlab="Lag (v)",ylab = expression(hat(rho)[v]))
    lines(c(0, m), c(0, 0))
    plot(part, type = "l", ylim = c(-1, 1), main="Partial Correlogram",
         xlab="Lag (v)",ylab = expression(hat(theta)[v]))
    lines(c(0, m), c(0, 0))
    plot(srvar, ylim = c(0, 1), main = "Standardized Residual Variances",
         xlab = "Lag (v)", ylab = expression(hat(sigma^2)[v]))
    plotsp(perd, length(x), tsvar(x), main = "Periodogram")
    plot(freqs(length(x)), cperd, main = "Cumulative Periodogram",
         xlab = expression(omega), ylab = expression(hat(F)(omega)))
    lines(c(0, 0.5), c(0, 1))
    lines(c(0, (0.5*(1-sq1)), c(sq1, 1)))
    lines(c((0.5*sq1), 0.5),c(0, (1-sq1)))

    return(list(mean = mean(x), var = tsvar(x), corr = corr,
                part = part, srvar = srvar, perd = perd, cperd = cperd))
}
```

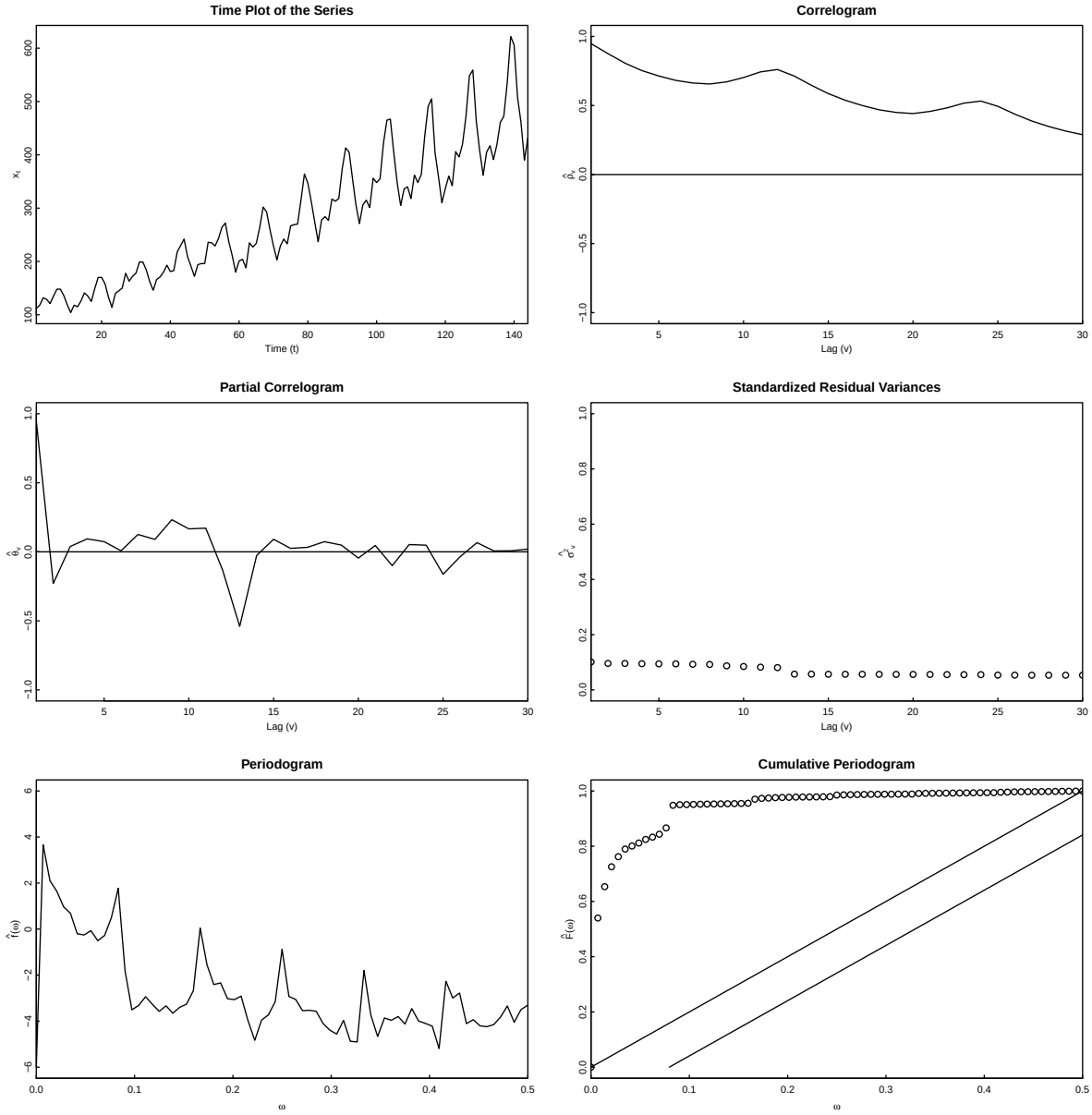


Figure 2.9: Example graph from `descplot` function.

In the listing of the `descplot` function, there are two *R* functions used for controlling the appearance of the graphics window. The first is the function `par` which can be used for setting or querying graphical parameters. More information about using `par` is available through *R*'s on-line help by typing `help(par)` at the *R* prompt. The other function used in this example is `expression`. In short, `expression` is a one of several *R* functions that allow the user to include mathematical expressions in graphics text. For more information on all these functions, see the documentation in *R* for `plotmath`.

The *Timeslab* function `describe(x, m)` computes and returns, but does not plot the descriptive statistics plotted and returned by the `descplot` function. Therefore the lines beginning at `corr <- corr(x,m)$corr` and ending with `cperd <- cumsum(perd)/sum(perd)` could be replaced by (something like) `z <- describe(x,m)`. This would also affect the calls to plot for the correlogram through the cumulative periodogram. For example, call to the `plot` function that results in the correlogram would change to

```
plot(z$corr, type = "l", ylim = c(-1, 1), xlab = "Lag (v)",
     ylab = expression(hat(rho)[v]), main = "Correlogram")
```

The lines in `descplot` that create the plots of the sample partial autocorrelations, standardized residual variances, periodogram, and cumulative periodogram would change in a similar manner.

Example 2.8 SCATTERPLOTS.
The function `phaseplot` *was used twice, in conjunction with* `par(mfrow=c(2,1))`*, to produce the scatterplots in Figure 2.1 in Section 2.2.1.*

```
phaseplot <- function(x, k, main = "Phase Plot")
#-------------------------------------------------------------------
#  R function to create a scatterplot of x_{t-k} versus x_t.
#
#  INPUT: x = a real vector containing the time series.
#         k = an integer scalar containing the lag.
#         main = a character string containing the title for
#                the plot.
#
#  VALUE: The function phaseplot creates a scatterplot of
#         x_{t-k} verssu x_t.
#-------------------------------------------------------------------
{
  plot(x[1:(length(x)-k)], x[(k+1):length(x)], pch = 20, main = main,
       xlab = expression(x[t]), ylab = bquote(x[t-.(k)]))

  return(invisible())
}
```

The sequence of function calls to create Figure 2.1 went something like

```
    par(mfrow = c(2, 1))
    phaseplot(datasets()$calfem, 1, main = "Scatterplot for Series I")
    phaseplot(datasets()$bev, 1, main = "Scatterplot for Series II")
```

The `bquote` function is an example of a function for including mathematical expressions in graphics text. For more information, see the *R* documentation on `plotmath`.

Example 2.9 Sum of Four Cosines.
The function `four.cosines` *generates four cosine curves, plots each of them, and their sum, and then calculates, plots, and lists the periodogram of the sum. The periodogram is zero except for the values at the frequencies of the individual cosine curves.*

```
four.cosines <- function()
#-------------------------------------------------------------
#
#  Function to illustrate how the periodogram can decompose
#  a series into a sum of sinusoids.
#
#-------------------------------------------------------------
{
  t <- 2*pi*(0:143)/144

  x1 <- 2*cos(t*12) + 3*sin(t*12)
  x2 <- cos(t*36) + 3*sin(t*36)
  x3 <- 6*cos(t*6) + sin(t*6)
  x4 <- 7*cos(t*3) + 4*sin(t*3)
  x  <- x1 + x2 + x3 + x4

  y.min <- min(x1, x2, x3, x4, x)
  y.max <- max(x1, x2, x3, x4, x)

  par(mfcol = c(5,2), mar = c(3,3,2,1), oma = c(0,0,0,0),
      tck = 0.01, cex.axis = 0.75, xaxs = "i",
      cex.main = 1, mgp = c(1,0.1,0), cex.lab = 0.75)

  plot(x1, type = "l", ylim = c(y.min, y.max),
       xlab = "t", ylab = expression(x[1]),
       main = "Cosine of Period 12 & Squared Amplitude 13")
  plot(x2, type = "l", ylim = c(y.min, y.max),
       xlab = "t", ylab = expression(x[2]),
       main = "Cosine of Period 36 & Squared Amplitude 10")
  plot(x3, type = "l", ylim = c(y.min, y.max),
       xlab = "t", ylab = expression(x[3]), ylim = c(y.min, y.max),
       main = "Cosine of Period 6 & Squared Amplitude 37")
```

```
  plot(x4, type = "l", ylim = c(y.min, y.max),
       xlab = "t", ylab = expression(x[4]),
       main = "Cosine of Period 3 & Squared Amplitude 65")
  plot(x, type = "l", ylim = c(y.min, y.max),
       xlab = "t", ylab = expression(sum(x[i], i == 1,4)),
       main = "Sum of Four Cosines")

  plotsp(perdgm(x1), 144, tsvar(x1),
         main = expression(paste("Periodogram of ",x[1])))
  plotsp(perdgm(x2), 144, tsvar(x2),
         main = expression(paste("Periodogram of ",x[2])))
  plotsp(perdgm(x3), 144, tsvar(x3),
         main = expression(paste("Periodogram of ",x[3])))
  plotsp(perdgm(x4), 144, tsvar(x4),
         main = expression(paste("Periodogram of ",x[4])))
  plotsp(perdgm(x), 144, tsvar(x),
         main = "Periodogram of Sum of Four Cosines")

  return(invisible())
}
```

Example 2.10 The Sawtooth Function.
A classic example of a periodic but non-sinusoidal function is the sawtooth function (see Figure 2.10 for an example). The function `sawtooth` *will generate a data set consisting of* `ncycles` *cycles of the sawtooth function where a cycle is defined to be 0 through* `amp` *followed by* `amp` -1 *through* $-$`amp`*, followed by* $-$`amp` $+1$ *through 0. Thus each cycle consists of* 4`amp` *points. The variable* `amp` *represents the amplitude of the sawtooth. In Figure 2.10, we have used 5 and 10 for* `ncycles` *and* `amp`*, respectively.*

```
sawtooth <- function(ncycles = 5, amp = 10, plot.it = as.logical(T))
#-------------------------------------------------------------------------
#  R function to generate and plot a data set consisting of ncycles
#  cycles of the sawtooth function with amplitude amp.
#
#  The function sawtooth has three optional input arguments:
#   ncycles = an integer scalar indicating the number of cycles
#             to generate. The default is ncycles = 5.
#   amp = an integer scalar containing the amplitude of the cycles.
#   plot.it = a logical indicating whether or not to generate the
#             plot.  The default is plot.it = TRUE.
#
#  The function sawtooth function returns a list containing two objects
#    t = an integer vector containing the abscissa of the points
```

```
#    x = an real vector containing the ordinates of the points from
#        the sawtooth function having abscissa t.
#  If plot.it = TRUE, then the sawtooth function will also create a
#  graph of the sawtooth function having ncycles cycles and amplitude
#  as specified in amp.
#-----------------------------------------------------------------------
{
  n <- 4*amp*ncycles
  t <- 1:n
  x <- rep(0, n)
  a2 <- 2*amp
#
  x[1:amp]       <- t[1:amp]
  x[(n-amp+1):n] <- -amp + t[1:amp]
#
  for(cycle in 1:(2*ncycles-1)) {
      cc <- 2*cycle - 1
      dd <- 2*cycle + 1
      x[(cc*amp + 1):(dd*amp)] <- ((-1)^(cycle+1))*amp
                                      + ((-1)^cycle)*t[1:a2]
  }
#
  if(plot.it)
    plot(x, type = "l", xlab = "t", ylab = expression(x[t]),
         main = "Sawtooth Function")
#
  return(list(t = t, x = x))
}
```

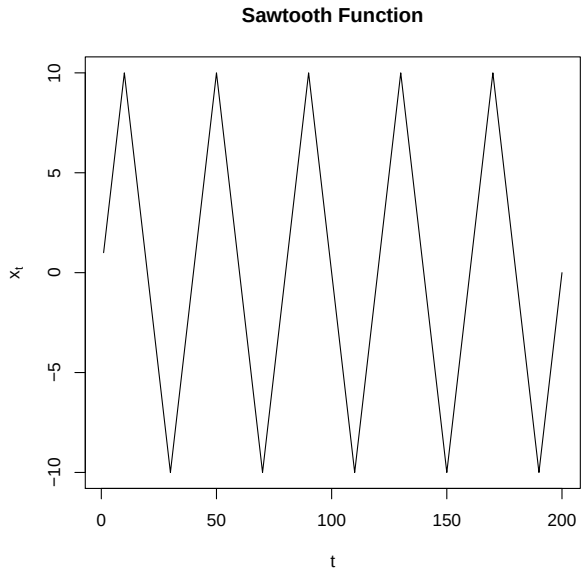


Figure 2.10: Graph of the sawtooth function with five cycles of amplitude ten.

Example 2.11 Testing for White Noise.
In this example, we construct two graphs that are useful for determining whether a data set could be white noise. We will see in Sections 6.1 and 6.2 that if $x_1, \ldots, x_n$ is a random sample from a population, then for large n:

1. *The sample autocorrelations $\hat{\rho}_1, \ldots, \hat{\rho}_M$ are independent and identically distributed as N(0,1/n) variables. Thus there is approximately a 5% chance that an individual $\hat{\rho}_v$ will be outside of $\pm 1.96/\sqrt{n}$. To produce simultaneous confidence band having a 95% confidence level, we must construct individual confidence intervals having level $0.95^{1/M}$. This is what is done in the first part of the* `wntest` *function.*

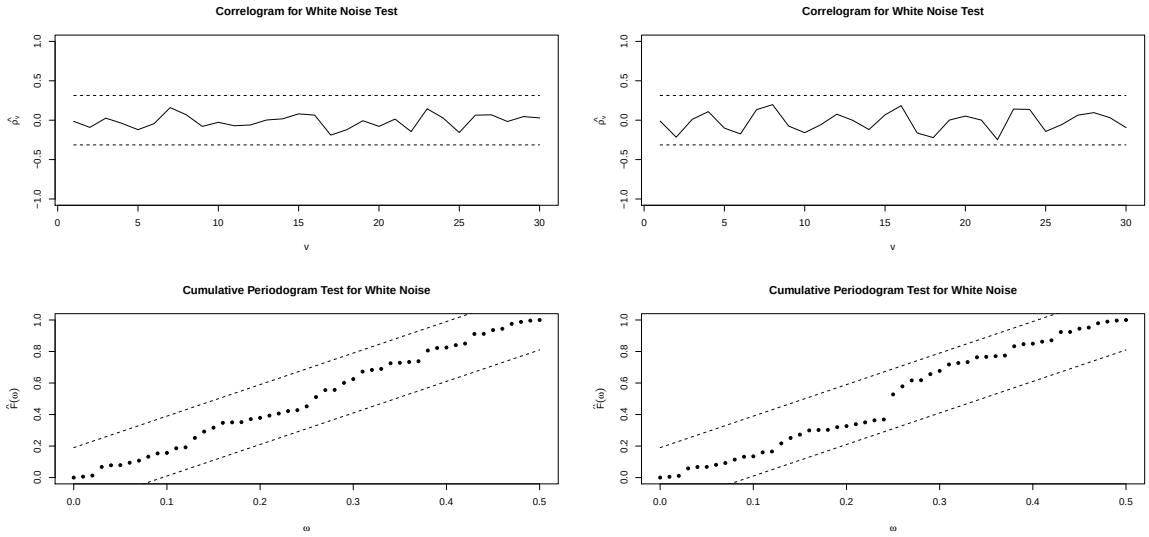
2. *The cumulative periodogram has a 95% chance of falling entirely within the lines $y = 2x \pm 1.36/\sqrt{Q}$, where $Q = [n/2]+1$. The second part of the macro constructs the graph of the cumulative periodogram and superimpose these two lines together with the line $y = 2x$.*

In Figures 2.11(a) and 2.11(b), we give the results of the function, the first series being normal white noise of length 100, and the second being the same series with a cosine of length 100, amplitude 0.5, and period 4 added to it. Notice that none of the boundary lines are crossed for either series. But the cumulative periodogram seems to increase rapidly at $\omega = 0.25$, something that is unusual and should be a hint that this data set is, in fact, not white noise.

```
wntest <- function(x, M = floor(10*log10(length(x))), conf = 0.95)
#-------------------------------------------------------------------
#  R function to find approximate (1-conf)100% simultaneous
#  confidence bands for the correlogram and cumulative periodogram
#  under the null hypothesis of white noise.
#
#  Input: x = a vector containing the series.
#         M = an integer containing the maximum lag for the
#             sample autocorrelation function.  The default is
#             M = [10 log_{10}(n)], where n = length(x).
#         conf = a real scalar containing the desired level of
#                confidence for the simultaneous confidence bands.
#                The default is conf = 0.95.
#
#  The function wntest creates a set of two graphs, in a 2x1 array.
#  The top graph contains the correlogram with the 95% simultaneous
#  confidence bands superimposed.  The bottom graph contains the
#  cumulative periodogram with 95% simultaneous confidence bands
#  superimposed.
#-------------------------------------------------------------------
{
```

```
   if(conf < 0.5 || conf > 1)
     stop("\nThe value of conf must be between 0.5 and 1.\n")
   if(M >= length(x) || M < 1)
     stop("\nThe value of M must be greater than 0 and less than n-1.\n")

   par(mfrow = c(2, 1), oma = rep(0, 4), mar = c(5, 4, 4, 2) + 0.1)
#
   simul.alpha <- (1 + conf^(1/M))/2
   rho.hat     <- corr(x, M)$corr
   cum.perd    <- cumsp(perdgm(x))
   upper.rho   <- qnorm(simul.alpha)/sqrt(length(x))
   lower.rho   <- -upper.rho
   simul.alpha <- round(simul.alpha, 6)
#
   fb <- rep(0, 501)
   b  <- seq(0.4, 2, length = 501)
   for(i in 1:501) fb[i] <- 1 - barttest(b[i], length(x))$pval
   inds <- which(round(fb, 3) == round(conf, 3))
   if(length(inds) == 1) cc <- b[inds] else cc <- median(b[inds])
   xx <- seq(0, 0.5, length = length(x))
   yy.upper <- 2*xx + cc/sqrt(length(cum.perd))
   yy.lower <- 2*xx - cc/sqrt(length(cum.perd))
#
   plot(1:M, rho.hat, type = "l", ylim = c(-1, 1), xlab = "v",
        ylab = expression(hat(rho[v])),
        main = "Correlogram for White Noise Test")
   lines(c(1, M), rep(upper.rho, 2), lty = 2)
   lines(c(1, M), rep(lower.rho, 2), lty = 2)
   plot(freqs(length(x)), cum.perd, ylim = c(0, 1), pch = 20,
        xlab = expression(omega), ylab = expression(hat(F)*(omega)),
        main = "Cumulative Periodogram Test for White Noise")
   lines(xx, yy.upper, lty = 2)
   lines(xx, yy.lower, lty = 2)
#
   return(invisible())
}
```



(a) White noise series.

(b) White noise plus cosine.

Figure 2.11: Result of `wntest` function on two datasets.

Example 2.12 Two Methods of Finding Correlations.
In Section 2.2.4, there are two computational methods for computing the sample autocorrelation coefficient: the two-FFT method, and the convolution method. The function `timecov` *allows a user to experiment with the two methods and obtain a precise timing for various sample sizes and numbers of correlations.*

```
timecov <- function(n = 500, M = 40)
#-----------------------------------------------------------------------
#  R function to compare system times for computing the sample
#  autocovariance function for a series of length n for up to M lags.
#  The function has two optional arguments.
#
#  OPTIONAL INPUT:
#   n = an integer scalar containing the desired length of the time
#       series.
#   M = an integer scalar containing the maximum number of sample
#       sample autocovariances to be computed.
#
#  The function timecov returns a list containing the following two
#  objects:
#  time.2FFTs = a real scalar indicating the elapsed system time for
#           computing the sample autocorrelation function using the
#           corr function (two-FFT method),
#  time.conv = a real scalar indicating the elapsed system time for
#            computing the sample autocorrelation function using the
#            corr1 function (convolution method).
#-----------------------------------------------------------------------
{
  x       <- rnorm(n)
  t.corr  <- system.time(corr(x, M))[3]
  t.corr1 <- system.time(corr1(x, M))[3]
#
  return(list(time.2FFTs = t.corr, time.conv = t.corr1))
}
```

For example, if $n = 12,637$ *and* $M = 1000$*, the function is called using*

```
timecov(12367, 1000)
```

and returns

```
$time.2FFTs
elapsed
   0.01
```

```
$time.conv
elapsed
   0.34
```

The method using 2 FFTs, which took approximately one one-hundredth of a second was approximately 34 times faster than the convolution method, which took approximately 34 one-hundredths of a second.

Example 2.13 Examining Samples via an Informative Quantile Plot.
The informative quantile plot, discussed in Section 2.3 is a plot of

$$y_{(i)} = \frac{x_{(i)} - \tilde{m}}{2IQR} \quad versus \quad u_{(i)} = \frac{i - 0.5}{n}, \quad i = 1, 2, \ldots, n,$$

where $\tilde{m}$ is the median of the data, and $x_{(i)}$ is the i^{th} smallest x, and the values plotted on the vertical axis are truncated to fall between -1 and 1. When an informative quantile plot is produced, a reference line is placed on the plot. This line is the true quantile plot of a U(0,1) random variable. One can get an idea of what the informative quantile plot should look like for various distributions by generating a sample via the `wn` *function and then calling* `infqnt`*. Try the following series of commands.*

```
x <- wn(100, "uniform")  #Generate a U(0,1) sample of length 100
infqnt(x)
```

The points generated by the call to the `wn` *function are a simple random sample of size $n = 100$ from a U(0,1) distribution. (For more information on the* `wn` *function, see Section 1.4.1.) Describe the relationship of the points to the reference line. Now try*

```
x <- wn(500, 2)  #Generate a U(0,1) sample of length 500
infqnt(x)
```

Compare the relationship of the points to the reference line for this larger sample to the previous one. More examinations of distributions are provided in the Computational Exercises C2.18 through C2.23.

2.6 A More Advanced Approach

In Section 2.2.3, we introduced the definition of the periodogram of x with the idea that a series x can be decomposed into a set of orthogonal sinusoids, and that the variability of the series (as measured by $\hat{\sigma}^2$) could be written as a sum of the squared amplitudes of those sinusoids. The theorem that proves this decomposition is state below. It provides the basic facts about the harmonic analysis of $x_1, \ldots, x_n$. It is relatively easily proved using the results in Theorem 2.1 along with additional trigonometric identities (see Appendix A.1).

Theorem 2.3 Sinusoidal Decomposition of Data.
Let $x_1, \ldots, x_n$ be a finite sequence of numbers, and let $z_1, \ldots, z_n$ be the DFT of x. For $k = 1, \ldots, n$, let $\omega_k = (k-1)/n$ and define

$$a_k = \frac{1}{n}\Re(z_k), \quad b_k = \frac{1}{n}\Im(z_k).$$

Let $\hat{\sigma}^2 = n^{-1}\sum_{t=1}^{n}(x_t - \bar{x})^2$, and for $t = 1, \ldots, n$, define

$$\begin{aligned} g_{k,t} &= a_k \cos 2\pi(t-1)\omega_k + b_k \sin 2\pi(t-1)\omega_k \\ &= C_k \cos\left[2\pi(t-1)\omega_k - \arctan(b_k/a_k)\right]. \end{aligned}$$

Then

(a) $a_1 = \bar{x}, \quad b_1 = 0, \quad C_1^2 = \bar{x}^2, \quad g_{1,t} = \bar{x}.$

(b) $x_t - \bar{x} = \sum_{k=2}^{n} g_{k,t}$.

(c) $\hat{\sigma}^2 = \sum_{k=2}^{n} C_k^2$.

(d) For $k = 2, \ldots, [n/2]+1$, we have

$$\begin{aligned} a_k = a_{n-k+2}, \qquad & b_k = -b_{n-k+2} \\ \cos 2\pi(t-1)\omega_k &= \cos 2\pi(t-1)\omega_{n-k+2} \\ \sin 2\pi(t-1)\omega_k &= -\sin 2\pi(t-1)\omega_{n-k+2} \end{aligned}$$

and thus

$$g_{k,t} = g_{n-k+2,t}, \quad C_k^2 = C_{n-k+2}^2,$$

which gives

$$\begin{aligned} x_t - \bar{x} &= \begin{cases} 2\sum_{k=2}^{[n/2]+1} g_{k,t}, & n \text{ odd} \\ 2\sum_{k=2}^{n/2} g_{k,t} + g_{(n/2)+1,t} & n \text{ even} \end{cases} \\ \hat{\sigma}^2 &= \begin{cases} 2\sum_{k=2}^{[n/2]+1} C_k^2, & n \text{ odd} \\ 2\sum_{k=2}^{n/2} C_k^2 + C_{(n/2)+1}^2, & n \text{ even.} \end{cases} \end{aligned}$$

(e) For $k = 1, \ldots, [n/2]+1$, the vectors $\boldsymbol{g}_k = (g_{k,1}, \ldots, g_{k,n})^T$ are orthogonal; that is, $\boldsymbol{g}_j^T \boldsymbol{g}_k = 0$, $j \neq k$.

Parts (d and (e) of this theorem gives a decomposition of a time series data set into a set of orthogonal sinusoids of frequencies $1/n, 2/n, \ldots, [n/2]/n$. Part (c) provides a decomposition of the variability of the series, as measured by $\hat{\sigma}^2$ as a sum of the squared amplitudes of those sinusoids. Thus, in terms of understanding why the xs vary, these squared amplitudes play a vital role. Note that $|C_k^2| = |z_k|^2/n^2$.

There are mathematical relationships between the correlogram as defined Section 2.2.1 and the periodogram in 2.2.3. The following theorem summarizes those relationships.

Theorem 2.4 RELATIONS AMONG $\hat{\rho}, \hat{R}$, AND $\hat{f}$.
Let $x_1, \ldots, x_n$ be a time series data set, and for $t = 1, \ldots, n$, let $z_t = x_t - \bar{x}$. For a positive integer M, let

$$y_t = \begin{cases} z_t & t = 1, \ldots, n \\ 0 & t = n+1, \ldots, n+M \end{cases}$$

that is, the series y is just the series z "padded" with M zeros. Let $\hat{\rho}_z, \hat{R}_z$ and $\hat{f}_z$ be the autocorrelation, autocovariance, and sample spectral density of z, respectively, and let $\hat{f}_y$ be the sample spectral density of y. Then

(a) $\hat{f}_z(\omega) = \sum_{|v|<n} \hat{R}_{z,v} e^{-2\pi i v \omega}, \quad \omega \in [0, 1]$

(b) $\hat{R}_{z,v} = \int_0^1 \hat{f}_z(\omega) e^{2\pi i v \omega} \, d\omega, \quad |v| < n$

(c) $\hat{R}_{z,v} = Q^{-1} \sum_{k=1}^{Q} \hat{f}_y(\omega) e^{2\pi i v \omega_k}, \quad v = 0, 1, \ldots, M,$

where $\omega_k = (k-1)/Q, \ k = 1, \ldots, Q$.

Parts (a) and (b) of this theorem state that $\hat{f}_z$ and $\hat{R}_z$ are related in a very specific way – they are "Fourier transform pairs." Part (c) states that the sample autocovariances can be calculated as the DFT of the sample spectral density function (which is itself the square modulus of a DFT) of the padded series y. This is important because there are circumstances when computing two DFTS (on to get $\hat{f}_y$ and one to get $\hat{R}_z$) can be must faster than computing $\hat{R}_z$ directly from its defining formula. This is discussed in more detail in Section 2.2.4 and Appendix A.3. For the interested reader, a proof of this theorem is provided in Appendix B.2.

2.7 Exercises

2.7.1 Theoretical Exercises

T2.1 Use the results of Theoretical Exercise T1.1 to find an expression for the correlogram of $x_t = a + bt, \ t = 1, 2, \ldots, n$.

T2.2 Let $x_t = a \cos 2\pi(t-1)/p, \ t = 1, \ldots, n$, where $n = mp$; that is, the x's consist of m cycles of a pure cosine with amplitude a and integer period $p > 2$.

(a) Show $\bar{x} = 0$.

(b) Show that for $0 \le v < n$,

$$\hat{R}_v = a^2 \left[\frac{1}{2} \left(1 - \frac{v}{n} \right) \cos \frac{2\pi v}{p} - \frac{c}{2n} \sin \frac{2\pi v}{p} \right],$$

where $c = \cot 2\pi/p$, and thus show that

$$\hat{\rho}_v = \left(1 - \frac{v}{n} \right) \cos \frac{2\pi v}{p} - \frac{c}{n} \sin \frac{2\pi v}{p}.$$

Thus note that the correlogram of a cosine is itself a damped (because of the factor $1-(v/n)$) cosine having the same period minus a sine having amplitude going to zero as n gets large. (*Hints:* If z is a complex number, then $1/z = \bar{z}/|z|^2$. Use real part and geometric series arguments as in Theorem 2.1. Note that $\cos 2\theta = \cos^2\theta - \sin^2\theta$.)

T2.3 Show that if we calculate the IDFT of the DFT of a set of n numbers, then the result is n times the original set of numbers.

T2.4 The informative quantile function of a continuous random variable X (as opposed to that for a data set) is defined by

$$\mathrm{IQ}(u) = \frac{Q(u) - Q(0.5)}{2\left[Q(0.75) - Q(0.25)\right]}, \quad u \in (0,1),$$

where Q is the quantile function of X. Assuming the cumulative distribution function $F_X(x)$ of X is strictly increasing, the quantile function is defined by

$$Q(u) = x \quad \Longleftrightarrow \quad F_X(x) = \Pr\left(X \le x\right) = u.$$

Show the true informative quantile plot of any uniformly distributed random variable is the same. Show that the reference line placed on the plot produced by the `infqnt` function is this informative quantile plot.

T2.5 Show that the Parzen kernel, given in Table 2.2, is the probability density function of the average of four independent random variables, each having the uniform distribution on the interval $[-1, 1]$. (Hint: Find the moment generating function of the average.)

2.7.2 Computational Exercises

C2.1 Using a text editor like *Notepad* or *emacs*, create a data set containing 50 observations from your area of interest. Use the `descplot` function from Example 2.7 to perform an initial analysis of the data. Do there appear to be any trends or cycles? Do the data appear to be long or short memory? If there are any obvious transformations that might be used on your data, try them and run the `descplot` function on the transformed data, and describe those results.

C2.2 In the `wn` function, *Timeslab* generates Cauchy variables by $X = \tan\{\pi(U - 0.5)\}$, where U is a uniform random variable over the interval $(0, 1)$. It can also be shown that the ratio of two independent standard normal variables has the Cauchy distribution. To illustrate this, generate samples of size 200 using both methods and use th `infqnt` function to produce informative quantile plots for the two samples. Do they look (almost) the same?

C2.3 Repeat Exercise C1.12. Plot the correlogram of S_t, the random walk. Since a random walk is considered long memory, what type of behavior should you expect from the correlogram? Is this, in fact, the behavior you observed?

C2.4 Use the `wn` function to generate a Gaussian white noise series of length 100 and the `corr` command to find the periodogram of the series. Use the `plotsp` command with third argument `div = 0.001`, `0.01`, `0.1`, `1`, `10`, `100`, and `1000`, and note how this moves the graph up and down. If we use the sample variance of the process as the third argument `div = tsvar(x)`, then the curve will be centered around zero on the vertical axis. For more information, see Theorem 2.2.

C2.5 Create a new function called `rw1` by modifying the *Timeslab in R* function `rw` to use U(−1, 1) random variables for the white noise process (by replacing the call to `rnorm` with a call to `runif`). Note that a realization from a time series process is also called a "sample path" (see the discussion of Figure 3.3 in Section 3.1.2). Do the sample paths for random walks using uniform white noise look much different from those using Gaussian white noise? Generate a plot of five sample paths of length 200 from each function, plotting the five paths from the uniform distribution on one set of axes. For each distribution, superimpose the five paths on the same set of axes, making one plot for each white noise distribution. Compare the two plots.

C2.6 Use the function `sawtooth` (see Example 2.10) to generate an array containing the sawtooth function of length 96 points having amplitude 2. Use the `perdgm` and `plotsp` functions to produce a plot of the standardized periodogram of the data. Are there harmonics in the plot? What is the fundamental frequency? Why?

C2.7 Write a function that will generate a cosine curve `x` of length 100 points having period 5 and amplitude 1 and will plot the correlogram of `x` for $M = 40$. What is $\hat{R}(0)$? Does this agree with what Exercise T2.2 says it should be? Note that this problem illustrates that the correlogram of a sinusoid is also a sinusoid.

C2.8 Write an `R` function that will do the following:

(a) Generate and superimpose the plots of cosines of length `n`, amplitude 1, and periods p_1 and p_2 for user-specified values `p1` and `p2` of p_1 and p_2.

(b) Use the function `dot` to find the sum of the crossproducts `SS` of the two arrays computed in part 1.

Use the function for `n = 100`, and `p1 = 10, p2 = 20` and also for `p1 = 10, p2 = 10`. Do the results agree with part 1 of Theorem 2.1?

C2.9 Using a hand-held calculator, find the DFT of the series of numbers

$$x_1 = 1, \quad x_2 = 3, \quad x_3 = 5, \quad x_4 = 7, \quad x_5 = 9.$$

Now using a hand-held calculator, find the IDFT of the $z_1, \ldots, z_5$ you computed as the DFT of $x_1, \ldots, x_5$. What is the numerical relationship between the IDFT of the DFT and the original values of $x_1, \ldots, x_n$?

C2.10 Use the `seq` function in *R* to form an array `x` of length 100 containing `x[i] = i`. Use the *R* function `FFT` to find the discrete Fourier transform of `x` and then to find the inverse discrete Fourier transform of the discrete Fourier transform of `x`. Verify that the imaginary part of this second transform is zero and the real part is 100 times the original array `x`.

C2.11 Use the `timecov` function (in Example 2.12) to find values of `n` and `M` for which the convolution method is faster, and another set for which the two-FFT method is faster.

C2.12 Find three prime numbers that are greater than 15,000. For each of these prime numbers, use the `timecov` function in Example 2.12 to determine how much faster the 2-FFT method is for computing the sample autocorrelation function compared to the convolution function for $M = 1000, 1500$, and 2000. Using the expressions in Section 2.2.4, find the numbers of computations required for both methods. Discuss your findings. When might using the 2-FFT method be truly beneficial?

C2.13 Create a vector `x <- 1:100`. Use the *R* function `fft` to find the discrete Fourier transform of `x` and then find the inverse discrete Fourier transform of the transform of `x`. Explain what you notice about the imaginary part of the second transform. What is its value? What is the value of the real part of the second transform? What is the relationship of this value to the length of the vector and the value of the original vector?

C2.14 Write an *R* function that accomplish the following:

(a) Compute and graph the sum of a three sinusoids with differing amplitudes and frequencies, as in equation (2.4).

(b) Compute and graph the standardized periodogram.

(c) Return the series, frequencies, and periodogram values in a list.

Use the results of the function to reconstruct the sinusoids that generated the data.

C2.15 Use the `wntest` function (from Example 2.11) on the first 23 and the last 22 years of the Lake Erie Levels data (named `eriel`). Does there appear to be any difference in the two analyses?

C2.16 Write a function that will successively generate white noise data of length 100, 200, 400, 800, 1600, and 3200, and use the `hist` function with `breaks = "Scott"` to create the histograms. Note how the number of intervals chosen by the algorithm proposed by Scott (1979) increases as the sample size increases. The rule of thumb that is used is that the number of intervals is proportional to $n^{1/3}$, where n is the sample size.

C2.17 Repeat Exercise C2.14, with the following modification. To each series that is the sum of three sinusoids, add normal white noise with user-specified variance. Run the function four times times, once with no white noise added to the sinusoids, and then

once with white noise variance 0.01, once with white noise variance 0.25, and once with white noise variance 4. Compare the appearance of the sinusoids with no white noise to those with white noise; do likewise for the periodograms. How does the appearance of the periodograms change with increasing white noise variance? Keep in mind, the objective is to identify a peaks in the periodogram.

C2.18 Refer to Example 2.13. Modify that example to create an informative quantile plot of normal white noise for sample sizes $n = 100$ and $n = 500$. Comment on the (1) shape of the points, (2) fit of the points to the U(0,1) reference line, and (3) the effect of increasing the sample size on the plot. (For a review of the `wn` function, see Section 1.4.1).

C2.19 Refer to Example 2.13. Modify that example to create an informative quantile plot of logistic white noise for sample sizes $n = 100$ and $n = 500$. Comment on the (1) shape of the points, (2) fit of the points to the U(0,1) reference line, and (3) the effect of increasing the sample size on the plot. (For a review of the `wn` function, see Section 1.4.1).

C2.20 Refer to Example 2.13. Modify that example to create an informative quantile plot of Cauchy white noise for sample sizes $n = 100$ and $n = 500$. Comment on the (1) shape of the points, (2) fit of the points to the U(0,1) reference line, and (3) the effect of increasing the sample size on the plot. (For a review of the `wn` function, see Section 1.4.1).

C2.21 Refer to Example 2.13. Modify that example to create an informative quantile plot of extreme value white noise for sample sizes $n = 100$ and $n = 500$. Comment on the (1) shape of the points, (2) fit of the points to the U(0,1) reference line, and (3) the effect of increasing the sample size on the plot. (For a review of the `wn` function, see Section 1.4.1).

C2.22 Refer to Example 2.13. Modify that example to create an informative quantile plot of lognormal white noise for sample sizes $n = 100$ and $n = 500$. Comment on the (1) shape of the points, (2) fit of the points to the U(0,1) reference line, and (3) the effect of increasing the sample size on the plot. (For a review of the `wn` function, see Section 1.4.1).

C2.23 Refer to Example 2.13. Modify that example to create an informative quantile plot of double exponential white noise for sample sizes $n = 100$ and $n = 500$. Comment on the (1) shape of the points, (2) fit of the points to the U(0,1) reference line, and (3) the effect of increasing the sample size on the plot. (For a review of the `wn` function, see Section 1.4.1).

C2.24 If you completed more than one of Example 2.13 and Computational Exercises C2.18 through C2.23, compare the shapes of the points for $n = 500$. How can having this

information help you determine the distribution of white noise if you are observing it from real data and not from simulated data?

C2.25 To examine the effect of sample size on a kernel density estimate, replicate Example 2.6 but change the size of the standard normal sample from `1000` to each of `30`, `100`, `500`, and `5000`. How does increasing sample size affect the kernel density estimate?

C2.26 Generate a normal white noise series of length 200 and use the `tsdensity` function with `rbins = 4` and the Epanechnikov kernel to find a probability density estimate at the 51 equally spaced points between -4 and 4, inclusive. Use the *R* function `dnorm` to evaluate the true density at the same 51 points. Then use the `plot` and `lines` function to superimpose the plots of the estimated and true densities. Use points for the density estimate, and lines for the true density.

C2.27 Generate a pair of independent Gaussian white noise series of length 100 (each). Consider the elements of these series as the real and imaginary parts of a complex time series of length 100. Use the *Timeslab* function `polar` to find the amplitude (modulus) and phase (the arctangent of the imaginary part divided by the real part) of these complex numbers. Use the `infqnt` command to find the informative quantile plot of the phases. Does it closely follow the uniform line on the plot? Relate this to Exercise T1.4.

C2.28 There has been some controversy as to whether the distribution of the Buffalo Snowfall data (named `buffsnow`) is unimodal or trimodal. Use the `tsdensity` function with the Gaussian kernel and all combinations of 16, 32, and 48 for `npts` with `rbins` = 3, 4, and 5 to estimate the density of the data. Do you have an opinion as to how many modes the distribution has? What is the bandwidth of the kernel for the three values of `rbins`?