Chapter 3

Transforming and Forecasting 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 discuss at methods for transforming time series so that the series satisfies assumptions for time series analysis techniques. In Section 3.2, we will present some simple methods for time series forecasting.

3.1 Transforming Time Series

Some of the time series analysis techniques that we will introduce in later chapters (beginning in Chapter 4) assume (1) that the data being analyzed have no deterministic trends or cycles, and (2) that the variability in the data is constant over time. The traditional method of analyzing data that fail to meet these requirements is to do the analysis in three steps.

1. Try various transformations until the result appears to meet the requirements,

2. analyze the result of step 1, and

3. do the inverse operation of what was done in step 1.

This strategy is very similar to that used in regression analysis. In this section, we will describe some of the transformations that are often used in time series analysis.

3.1.1 Stabilizing Variance

Suppose that the variability in a data set x appears to be increasing as time increases, like in the airline data set. If the mean level of the data is constant with time, then the variability in $y_t = \log x_t$ should appear fairly constant over time. This is a fairly common occurrence in

real data. In general, one might try various power transformations to obtain a series having constant variability; that is,

$$y_t = [x_t]^r .$$

3.1.2 Removing Trends

Another common phenomenon in time series data is that the values appear to be growing in some polynomial fashion with time, particularly linearly; that is, the data appear to follow the *linear trend*

$$x_t = a + bt + \varepsilon_t,$$

where ε_t is white noise. Such polynomial trends are often removed by using a method called differencing.

Definition 3.1 Difference of a Time Series.
The $\boldsymbol{d^{\text{th}}}$ **difference** *of a time series data set x having n elements is a data set y having $n-d$ elements obtained by*

$$y_t = x_{t-d} - x_t, \quad t = 1, \ldots, n-d.$$

To illustrate differencing, suppose the time series $x_t = a + bt + \varepsilon_t$. Then

$$\begin{aligned} y_t &= x_{t+1} - x_t \\ &= [a + b(t+1) + \varepsilon_{t+1}] - [a + bt + \varepsilon_t] \\ &= b + \varepsilon_{t+1} - \varepsilon_t, \end{aligned}$$

which no longer contains a linear trend. It is not difficult to show that if a time series x contains a d^{th} degree polynomial trend, then applying the first difference d times will remove the trend (see Exercise T3.2). Similarly, if a time series x has a cycle of length s time units, then taking the s^{th} difference will remove the cycle (see Exercise T3.3).

Perhaps the most well-known example of the effect use of differencing is with the airline data that we have been analyzing. The variability in the data is increasing with time (as noted in Section 3.1.1), so a logarithmic transformation is usually applied to the series. The result of this transformation has an obvious 12-month cycle and also a linear trend. Thus the traditional advice on a series such as this is to take the first difference of the 12^{th} difference of the natural logarithm of the original data. The graphs in Figure 3.1 show the progression of the transformations.

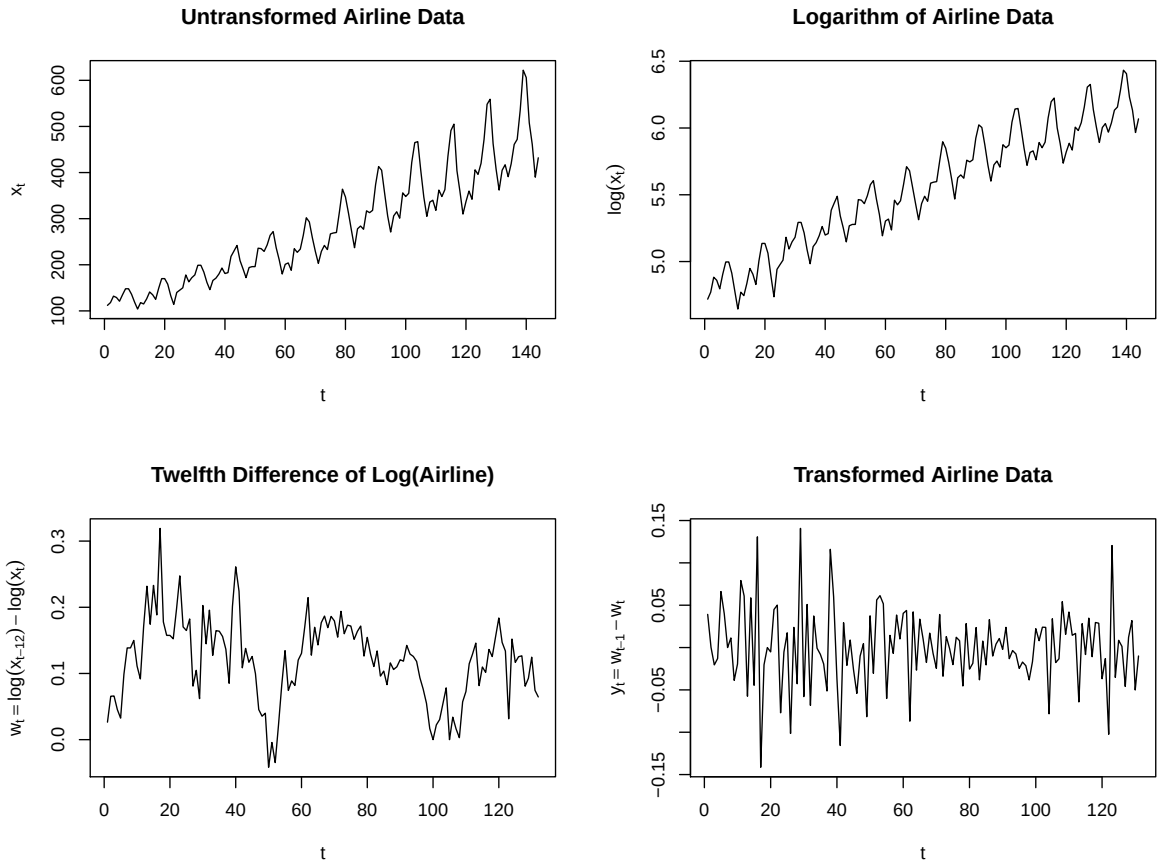

Figure 3.1: Successive transformations of the airline data: (1) "Raw" (untransformed) series (top left); (2) Natural logarithm transformation (top right); (3) Twelfth difference of the natural logarithm; (4) First difference of twelfth difference of the natural logarithm.

Note that the final series is only of length $(n-12)-1 = 131$. Thirteen observations were lost due to differencing.

Applying differencing is very simple, and often leads to good results, but some care must be taken. For example, in our linear trend example, the result of taking the first difference was

$$y_t = b + \varepsilon_{t+1} - \varepsilon_t,$$

which is a special kind of time series called a moving average time series (see Section 5.3) of order one with coefficient -1. Some standard analysis techniques cannot be applied to such a series.

Regression Analysis

A traditional method of removing trends and cycles from time series data is to do ordinary regression of the data on the deterministic functions of time. For example, Series V of the univariate series that we have been analyzing was formed by

$$x_t = 10 + 0.10t + 3\cos\frac{2\pi(t-1)}{10} + \varepsilon_t, \quad t = 1, \ldots, 100,$$

where ε_t is a series having serial correlation (actually, ε_t is a realization from an AR(1) process with coefficient -0.75; see Section 5.4). If we did ordinary least squares regression based on a model of the form

$$y_t = \beta_0 + \beta_1 t + \beta_2 \cos\frac{2\pi(t-1)}{10} + \varepsilon_t,$$

we would obtain the residuals, e_t say, in the plot in Figure 3.2, which can then be further analyzed by time series analysis. [1]

Accounting for Seasonal Variability

Often it is of interest in seasonal time series to analyze how a data set differs from regular seasonal variation. For example, suppose that x consists of m years of monthly data, that is $n = 12m$. We can define the monthly means and variances to the be means and variances of each of the 12 data sets consisting of like months; the Januaries, the Februaries, and so on. Thus

$$\begin{aligned} \bar{x}_k &= \frac{1}{m}\sum_{t=1}^{m} x_{k+12(t-1)} \\ s_k^2 &= \frac{1}{m}\sum_{t=1}^{m}\left(x_{k+12(t-1)} - \bar{x}_k\right)^2, \end{aligned}$$

$k = 1, \ldots, 12$. For other types of seasonality (for example, quarterly or hourly), the seasonal means and variances can be defined in a similar manner. See Exercise T3.1.

[1] The estimated least squares coefficients are $\hat{\beta}_0 = 10.145, \hat{\beta}_1 = 0.095$, and $\hat{\beta}_2 = 2.846$.

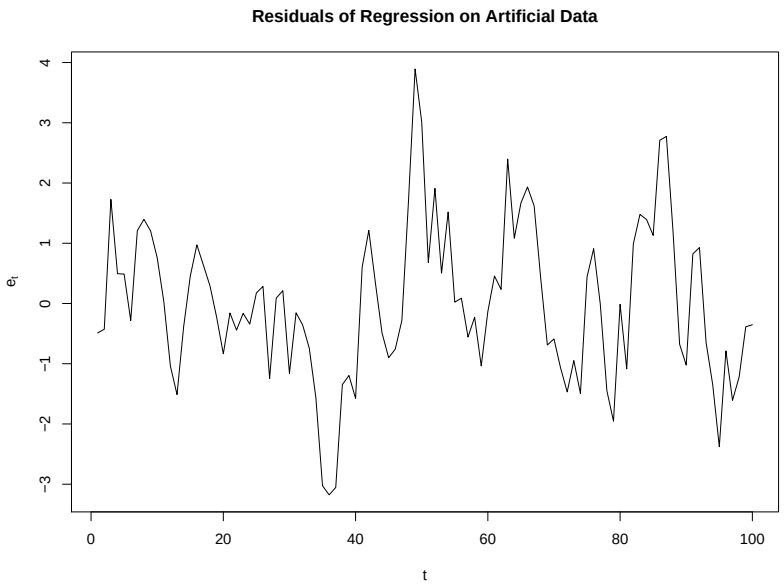

Figure 3.2: Residuals from a least squares regression on Series V.

Trend or Variance?

Certain time series models can yield a realization that appears to have a trend. A realization from a time series process is also called a **sample path**. Consider the series in the top left panel of Figure 3.3. The top right panel of this figure contains a time plot of a realization of length 100 from a model called a "random walk." It *appears* that the data exhibits an increasing linear trend. Certainly there is no clear evidence of nonconstant variance *within* the path of the data. So based on previous discussions, we might consider fitting a least squares regression line to the series to remove what we believe to be a linear trend. The top right panel contains the residuals from such a fit. These residuals are far from random, and *seem* to exhibit some type of cyclic behavior. So the least squares regression line may remove what appears to be a linear trend, but unfortunately, there is to be a fairly complicated structure that still remains. The next panel, on the bottom left, contains the first difference of the original series. These points do appear to be random. In fact, they are. It can be shown that the first difference of a random walk is white noise.

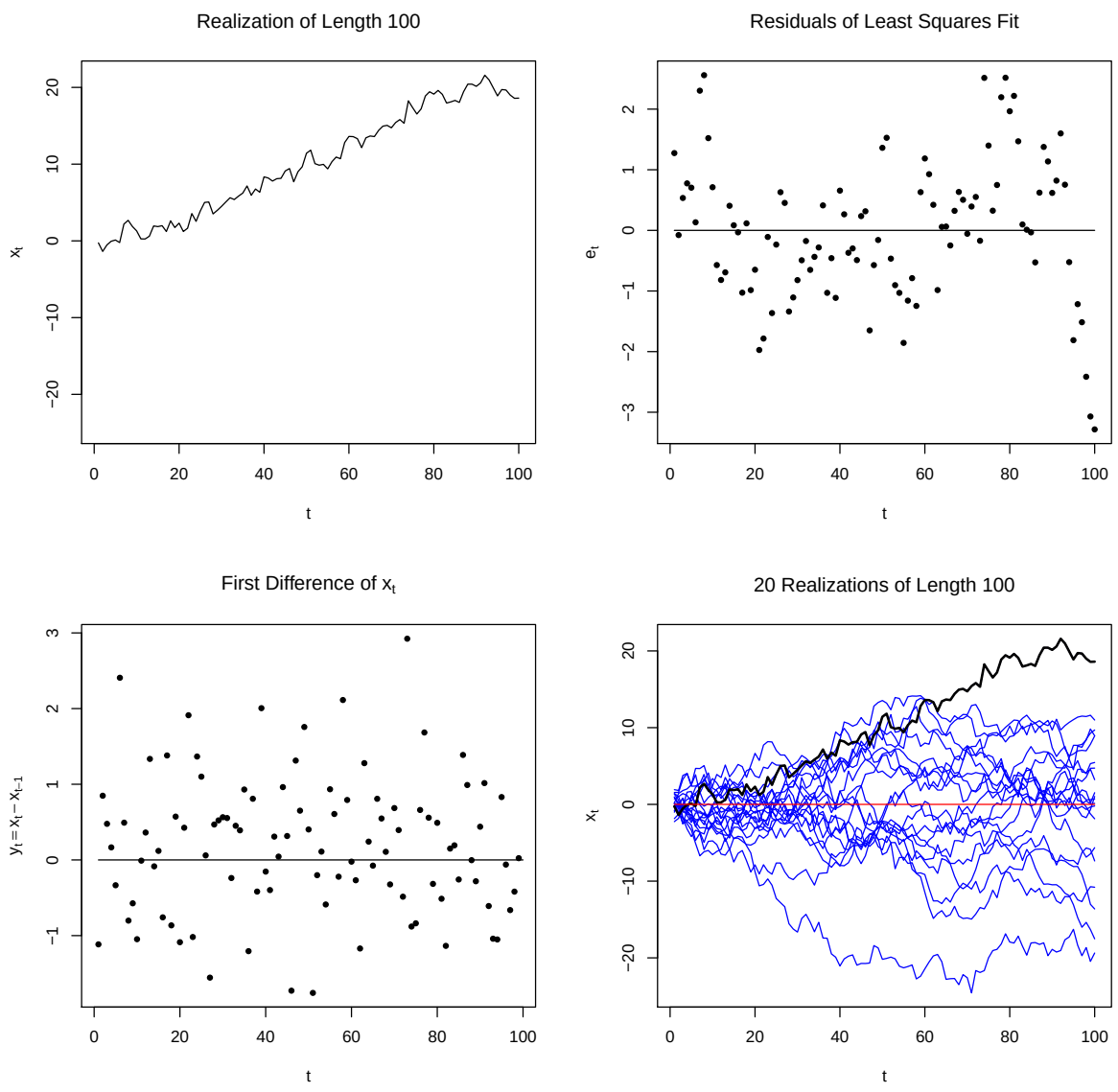

Figure 3.3: Illustration of difficulty in determining if behavior is the result of a linear trend or a different type of nonconstant variance. The top left panel is a realization of length 100 from a specific type of model. The top right panel are the residuals from a least squares regression fit. The bottom left panel are the first difference of the series. The bottom right contains that same realization, with an additional 19 realizations of length 100 from the same model superimposed.

The first difference resulting in white noise is not a contradiction to our earlier discussion. However, there is more here than meets the eye. The rest of the story is seen in the bottom right panel of the figure. The graph in that panel contains the same realization from the top left (in black), along with 19 other realizations (all in blue) of length 100 from the same model. Notice that, as time increases, the variability *between* the paths increases. This is a second type of nonconstant variance that can occur in some time series models, and it violates an assumption that we will require for further analysis. We'll discuss these ideas further in Chapter 4. The second idea that is illustrated in this bottom right panel is that a realization of length n of a model can be thought of as a *single* path from an infinite number of possible paths – in essence a "sample of size one" – from the infinite number of possible paths. In fact, most of the 19 paths don't have an increasing linear trend. About half of the paths wander above (the center) of zero, and the other half of the paths wander below zero. The trend in the chosen path is an artifact. The set of all possible paths is called the **ensemble**. This concept is one of a few that brings forth the idea that the length of a time series is not the same thing as the sample size for a random sample in traditional statistics. It is also one of the reasons why the methods discussed in Section 2.3 do not necessarily make sense when applied to a series that is not white noise. We will discuss these concepts further in Chapter 4 and Section 5.1.

3.1.3 General Smoothing Operations

Another approach to removing deterministic-looking parts of a time series is to use general methods of smoothing data and them analyze the deviations from the smooth version. For example, one way to smooth wiggly data is to use a moving average smoother, for example, the one of length three is

$$\begin{aligned} y_2 &= \frac{x_1 + x_2 + x_3}{3} \\ y_3 &= \frac{x_2 + x_3 + x_4}{3} \\ &\vdots \\ y_{n-1} &= \frac{x_{n-2} + x_{n-1} + x_n}{3}. \end{aligned}$$

In this example, we chose to index y using the index of the center value of the averaged xs. Our choice of indexing is intentional and meant to emphasize a point – there are fewer values of y than of x. There should be no ambiguity in reindexing y_2 above as y_1; reindexing y_3 as y_2, and so forth until y_{n-1} is reindexed to y_{n-2}. However, when superimposing the smoothed values with the original data, the proper placing of the smoothed values is important. More important that the choice of beginning index, notice that consecutive ys have two of the same xs in common in their average. Therefore, we expect that the ys won't vary as much as the original xs; that is, y will be much smoother in appearance than x. Figure 3.4 displays a series of length 100 that was generated from a pure cosine of period 40 with N(0,1) white noise added to it. The original data is plotted using points (as opposed to a line). The line

superimposed on the plot is a moving average smoother of length 11. As alluded to in our short discussion on the choice of indexing, and proper plotting, notice that the line begins over point $t = 6$ on the horizontal axis, and ends at $t = 95$. While the cosine curve is not evident in the original data, the smoothed data clearly exhibit a cyclic behavior.

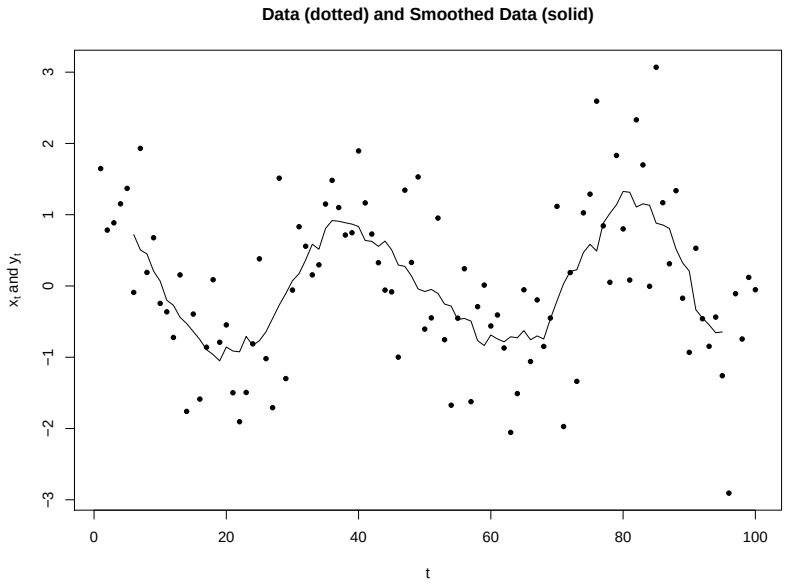

Figure 3.4: An Example of the Moving Average Smoother.

3.2 Some Simple Forecasting Methods

An important objective of time series analysis in many scientific areas is to forecast (predict) future values of a time series $x_1, \ldots, x_n$. In this section we begin our discussion of forecasting methods by looking at some simple methods, many of which are natural "extensions" of the transformation methods just discussed in Section 3.1. In this chapter, we will discuss only finding single point predictions at times $n + 1, n + 2, \ldots, n + h$. But there is much more to prediction than this. In Section 4.4, we will study the theory of prediction, and in Chapter 6, we will discuss some of the more elaborate inferential methods that are practiced.

Each of the methods in this section can be viewed as having two steps: (1) a model fitting step, and (2) a forecasting step. Note that many statistical procedures are of this form.

3.2.1 The Inverse Differencing Method

A simple but effective method for extrapolating trends and cycles in data that can be handled by differencing is to extend the data set with M new values in such a way that if the extended

series of length $n + M$ were then differenced, the last M values (the ones corresponding to the new values) would all equal the mean of the differences of the original data. For example, if a data set appears to have a linear trend, we might detrend it by applying first differences. We could then find a new data point x_{n+1} so that $x_{n+1} - x_n = \bar{z}$, where $\bar{z}$ is the mean of the differences of $x_1, \ldots, x_n$. Thus $x_{n+1} = x_n + \bar{z}$, and if we continue the process, we obtain

$$x_{n+h} = x_{n+h+1} + \bar{z}, \quad h \geq 1.$$

If we have data that appear to contain both a linear trend and a seasonal cycle, we might apply both first and d^{th} differences, where d is the length of the seasonal cycle. For monthly data having an annual cycle, we would solve

$$[x_{n+1} - x_n] - [x_{n-11} - x_{n-12}] = \bar{z},$$

where $\bar{z}$ is the mean of the first and twelfth differences of $x_1, \ldots, x_n$. This results in the predictor

$$x_{n+h} = x_{n+h+1} + x_{n+h-12} - x_{n+h-13} + \bar{z}, \quad h \geq 1.$$

Figure 3.5 contains a plot of the original time airline data, plotted with both lines and points, and two years (24 months) of predicted values obtained by the inverse differencing method using first and twelfth differences. The function to obtain the numerical values of the extended values is described in Example 3.1. Inverse differencing works well on the airline

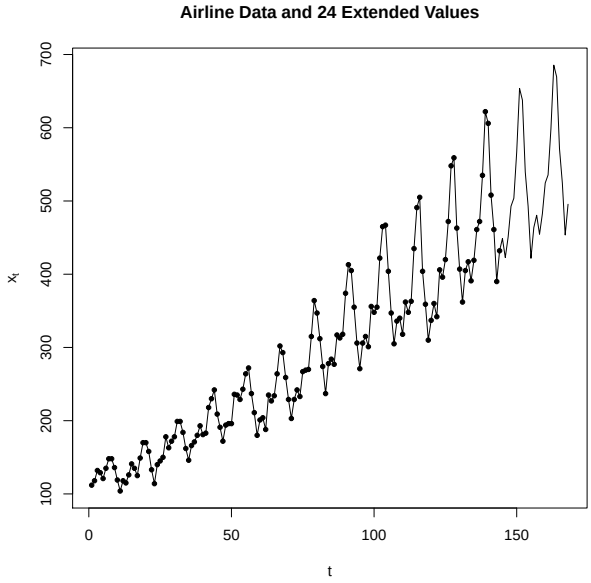

Figure 3.5: Airline Data and Extended Values.

data because the behavior in the data is so "predictable," that is, it is almost deterministic.

This naive method works because the airline data is a long memory series, and on such series, almost any method will work well, including just drawing the next two cycles by hand. For series that do not follow such a deterministic pattern, we will have to use more sophisticated forecasting methods.

3.2.2 The Regression Method

If the data appear to follow a deterministic function of time that can be expressed as a simple or multiple linear regression model, we can predict future values of the series by substituting future values of t into the least squares regression function (see Appendix A.5 for a discussion of multiple linear regression). For example, if

$$x_t = \beta_0 + \beta_1 t + \beta_2 \cos \frac{2\pi(t-1)}{p} + \varepsilon_t, \quad t = 1, \ldots, n,$$

where the period p is known, we can estimate β_0, β_1, and β_2 by their least squares estimates $\hat{\beta}_0, \hat{\beta}_1$, and $\hat{\beta}_2$, and forecast a value h steps into the future by

$$\hat{x}_{n+h} = \hat{\beta}_0 + \hat{\beta}_1(n+h) + \hat{\beta}_2 \cos \frac{2\pi(n+h-1)}{p}.$$

3.2.3 Simple Moving Average

This method models an observation as a simple average of the previous m observations, where m is to be chosen. For a given moving average of length k, we can calculate

$$S_k = \frac{1}{n-k} \sum_{t=k+1}^{n} \left(x_t - \frac{1}{k} \sum_{j=1}^{k} x_{t-j} \right)^2$$

as a measure of how well the simple moving average model of length k fits the observed data. We chose m as the value of k that minimizes S_k. Then we forecast future values recursively. For example, if $m = 3$, we calculate

$$\begin{aligned} \hat{x}_{n+1} &= \frac{1}{3}\left[x_n + x_{n-1} + x_{n-2}\right] \\ \hat{x}_{n+2} &= \frac{1}{3}\left[\hat{x}_{n+1} + x_n + x_{n-1}\right] \\ \hat{x}_{n+3} &= \frac{1}{3}\left[\hat{x}_{n+2} + \hat{x}_{n+1} + x_n\right] \end{aligned}$$

and so on.

3.2.4 Simple Exponential Smoothing

Instead of modeling an observation as a simple average of the previous m observations, exponential smoothing methods model x_t as a weighted average of all the previous values.

The type of weights that are used depends on the appearance of the data and leads to methods having a variety of names. We will discuss only the simple exponential smoothing technique which is most suitable for data that appear to have no linear or seasonal trends.

The simple exponential smoothing method consists of modeling x_t as a weighted average,

$$\hat{x}_{t+1} = \sum_{j=1}^{t} \beta_j x_{t+1-j}, \quad t \geq 1.$$

If we let $\beta_j = \alpha(1-\alpha)^{j-1}$, where $0 \leq \alpha \leq 1$ – that is, if we let the weights decay exponentially to zero – then $\sum_{j=1}^{\infty} \beta_j = 1$, and thus for large t, the weights will approximately sum to one. We can choose α as the value of a minimizing

$$S(a) = \sum_{t=2}^{n} \left\{ x_t - \sum_{j=1}^{t-1} a(1-a)^{j-1} x_{t-j} \right\}^2,$$

and then forecast the value at time $n+1$ by

$$\hat{x}_{n+1} = \sum_{j=1}^{n} \alpha(1-\alpha)^{j-1} x_{n+1-j}.$$

The calculations involved in this process can be greatly reduced by noting that

$$\hat{x}_{t+1} = \alpha x_t + (1-\alpha)\hat{x}_t, \tag{3.1}$$

where $\hat{x}_1$ is defined to be 0. We can also write this as $\hat{x}_{t+1} - (1-\alpha)\hat{x}_t = \alpha x_t$, which is called a *difference equation* of order one. Difference equations are very important in the study of time series analysis.

3.2.5 Difference Equations and the Behavior of Forecasts

Throughout this text, the more mathematical material will typically be saved for the "More Advanced Approach" section of the chapter. However, when that material is fundamental, and many other sections of the text depend upon it, it will be presented as a main part of the chapter. Such is the case for the material that appears in this section. It may be more difficult than the material in the previous sections, but it is foundational. Therefore, we encourage the reader to take a deep breath, and dive in. Take the time and mental energy it will require to master this material. It will prove highly beneficial in later chapters! We begin with the definition of a difference equation.

Definition 3.2 Difference Equation.
Let $z_1, z_2, \ldots$, and $w_1, w_2, \ldots$ be sequences of real numbers. Then

$$z_t + \alpha_1 z_{t-1} + \cdots + \alpha_p z_{t-p} = w_t \tag{3.2}$$

is called a **difference equation of order p, coefficients $\alpha_1, \ldots, \alpha_p$, and forcing term w.**

It is helpful here to relate the two series in the definition of a difference equation of order one to the fitted values and original series in equation (3.1) from our previous discussion on simple exponential smoothing. Note that the product of α with the original series – αx_t – acts as the forcing terms w_t, the fitted values $\hat{x}_{t+h}$ take the role of the z_t series, and the coefficient α_1 in equation (3.2) is $\alpha_1 = -(1-\alpha)$. At first, it may seem contrary to intuition that fitted values $\hat{x}_t$ act as z_t in the definition, and that αx_t act as the forcing term. But it should not since, in practice, we know the values of the forcing term – the original series – and we would like to find the values of the series z_t – the fitted values – using the difference equation.

For a difference equation of order p, to calculate the values of z for all values of t, it is sufficient to know all the values of w and any p consecutive values of z. These p values are called the starting values or **initial conditions**. If we know the initial conditions $z_1, \ldots, z_p$ and the values $w_{p+1}, \ldots, w_n$, for a difference equation, then we can find $z_{p+1}, \ldots, z_n$ recursively by

$$z_{p+j} = w_{p+j} - \sum_{k=1}^{p} \alpha_k z_{p+j-k}, \quad j = 1, \ldots, n-p.$$

Note that w_{p+1} and $z_1, \ldots, z_p$ are used to find z_{p+1}, which is in turn used to find z_{p+2}, and so on. The `diffeq` function, explained in Section 3.2.6, will perform this recursion.[2]

In Example 3.5, we provide a function which will choose α and superimpose the plots of the data and the resulting fitted values. Note that we can also write the simple exponential smoothing recursion as

$$\hat{x}_{t+1} = \hat{x}_t + \alpha(x_t - \hat{x}_t);$$

that is, the predicted value at time $t+1$ is that for time t plus an adjustment for the error in using $\hat{x}_t$ to model x_t. To forecast x_{n+h} for $h > 1$, some authors suggest using

$$\hat{x}_{n+h} = \alpha x_n + (1-\alpha)\hat{x}_{n+h+1}.$$

Many of the forecasting methods used in the time series analysis produce forecasts that are future values of a difference equation. In Theorem 3.1, we present results that can be used to describe analytically the behavior of these forecasts. The results of this theorem will be useful in other contexts as well.

Definition 3.3 Characteristic and Indicial Polynomials.
Let

$$\sum_{j=0}^{p} \alpha_j z_{t-j} = w_t$$

be a p^{th} order difference equation. If $w_t = 0$ for all t, then the difference equation is said to be **homogeneous**. *The polynomials*

$$g(z) = \sum_{j=0}^{p} \alpha_j z^j \quad \textit{and} \quad h(z) = \sum_{j=0}^{p} \alpha_j z^{p-j}$$

[2]The `diffeq` function replaces one of the operations performed by the `ardt` function in the original version of *Timeslab*.

are called the **characteristic** *and* **indicial polynomials**, *respectively, of the difference equation.*

Let $z_1, \ldots, z_p$ be the zeros of h; that is, $h(z_i) = 0$. Note that the zeros of g are the reciprocals of the zeros of h. These zeros play an important role in solving difference equations.

Theorem 3.1 Solution of Difference Equations.
Suppose that z_t satisfies a p^{th} order homogeneous difference equation, and let $z_1, \ldots, z_p$ be the zeros of the indicial polynomial of the equation. Then

(a) The zeroes are either real or occur in complex conjugate pairs; that is, if $z_i = a + bi$ is a complex valued zero, then $\bar{z}_i = a - bi$ is also a zero.

(b) z_t is equal to the sum of terms of two types. If a real root z occurs m times, then a term of the form

$$(\beta_1 + \beta_2 t + \ldots + \beta_m t^{m-1}) z^t$$

is included. For a distinct real root this term is $\beta_1 z^t$. If a conjugate pair ($z = a + bi, \bar{z} = a - bi$) occurs m times, then a term of the form

$$\left[\gamma_1 \cos(t\theta + \delta_1) + \gamma_2 t \cos(t\theta + \delta_2) + \cdots + \gamma_m t^m \cos(t\theta + \delta_m)\right] r^t$$

is included, where t and θ are determined by expressing z and $\bar{z}$ as $r(\cos\theta \pm i \sin\theta)$; that is, $r = \sqrt{a^2 + b^2}$ and $\theta = \arctan(b/a)$. In either case, the β's, γ's, and δ's are determined by solving the system of equations resulting from p initial conditions, that is, by setting any p known values of z equal to the sum of the terms described above.

(c) If the zeros are all real and distinct, then as $t \to \infty$, the values of z_t either converge to zero, oscillate between $-\infty$ and ∞, or diverge to ∞.

Note that part (c) is just one possible application of part (b). (See Exercise T3.5 for another.) To illustrate solving a difference equation analytically, consider

$$z_t - z_{t-1} + z_{t-2} - z_{t-3} = 0, \quad t \geq 4,$$

with initial conditions $z_1 = 1, z_2 = -1$, and $z_3 = 1$. Then the indicial polynomial is

$$h(z) = z^3 - z^2 + z - 1 = (z^2 + 1)(z - 1),$$

which has zeroes $z_1 = i, z_2 = -i$, and $z_3 = 1$. For z_3 a term of the form $\beta 1^t = \beta$ is included, while for the z_1 and z_2, we have $r = 1$ and $\theta = \pi/2$, and thus a term of the form $\gamma \cos(t\pi/2 + \delta)$ is included. This yields

$$z_t = \beta + \gamma \cos\left(\frac{t\pi}{2} + \delta\right),$$

where β, γ, and δ are found by solving the system of equations

$$\begin{aligned} z_1 &= 1 = \beta + \gamma \cos\left(\frac{\pi}{2} + \delta\right) \\ z_2 &= -1 = \beta + \gamma \cos\left(\pi + \delta\right) \\ z_3 &= 1 = \beta + \gamma \cos\left(\frac{3\pi}{2} + \delta\right). \end{aligned}$$

It is not difficult to show that these three equations are satisfied when $\beta = 1, \gamma = -2$ and $\delta = \pi$. Finally, we have

$$z_t = 1 - 2\cos\left(\frac{t\pi}{2} + \pi\right),$$

and thus z repeats its values every four points with z_4 through z_7 given by 3, 1, -1, and 1.

If we plot the real and imaginary parts of a complex number $z = a + bi$, on an X-Y plane, then $|z| = \sqrt{a^2 + b^2}$ is greater than one if and only if the plotted point falls outside of a circle of radius one that is centered at the origin. This circle is called the unit circle, and we will henceforth refer to zeros being inside, outside, or on the unit circle. In Chapter 7, we will describe methods for determining whether the zeros of a polynomial are outside the unit circle without actually finding them.

3.2.6 Using *Timeslab in R*

The set of *Timeslab* functions for simple prediction methods are described below.

The `extend` Function

The `extend` function can be used to predict `h` steps ahead by using the inverse differencing method described in Section 3.2.1. The call to the function is

```
extend(x, h, d1, d2 = 0)
```

The required argument `x` is a vector containing the time series; `h` is the number of point that the time series is to be extended. The required argument `d1` is an integer containing the order of the difference of the data. If another differencing operation is also necessary (as in the case of the airline data), then the value of `d2` specifies the order of that differencing operation. The default value of `d2 = 0`. The `extend` function returns a vector of length `n + h` (where `n` is the length of the original time series) containing the original time series in the first `n` elements and the extended time series in the last `h`.

Example 3.1 Extending the Airline Data Using Inverse Differencing.
The code below extends the natural logarithm of the airline data by two years (`h = 24` *months), when the first difference and the twelfth difference of the log of the data is analyzed.*

```
extend(air, 24, 1, 12)
```

The `diffeq` Function

Recall the definition of a difference equation of order p, with coefficients $\alpha_1, \ldots, \alpha_p$ and forcing term w given in Section 3.2.5. The function `diffeq` can be used to construct values of $z_{p+1}, \ldots, z_n$, given the values of the coefficients, and the initial values of the forcing term $w_1, \ldots, w_p$. The call to the function `diffeq` is

```
diffeq(alpha, n, e, p = length(alpha))
```

The argument `alpha` is a vector of length `p` containing the coefficients of the difference equation, `p` is an integer containing the order of the difference equation, `n` is an integer indicating the length of the desired realization, and `e` is a vector of length at least `p` containing the initial value in the forcing term. The output of `diffeq` is the vector of length `n`. The first `p` values are the values in `e`, and the `p + 1` through $n^{\texttt{th}}$ values are the constructed z values. If `p` is given to be zero, then the value of `alpha` is set to zero, and a white noise series is returned.

The `ardt` Function

Recall again the definition of a difference equation of order p, with coefficients $\alpha_1, \ldots, \alpha_p$, and forcing term w. In the function

```
ardt(alpha, n, rvar = 1, seed = 0)
```

the argument `alpha` contains coefficients $\alpha_1, \ldots, \alpha_p$. Unlike the `diffeq` function, the `ardt` function generates forcing terms from a normal white noise series with variance supplied by the user in the real scalar argument `rvar`. The integer argument `n` determines the length of the values generated from the difference equation. The `seed` argument is used in the `wn` function. If the values of `alpha` are properly chosen, then the output of `ardt` is a list containing a vector `x` that is slightly different in the description of what `diffeq` returns, in that it is a realization from an "autoregressive model" (see Section 5.4). Also in the list is an integer `error`. If `error = 0` then values of `alpha` were appropriately chosen. In Chapter 4, we will discuss specifically what we mean when we say "appropriate values" for `alpha`.

The `divpoly` Functions

A call to the function `divpoly` returns a vector containing the first `n` coefficients of $g(z)/h(z)$, where $g(z)$ and $h(z)$ have coefficients specified in the vectors `num` and `den` respectively. The zero degree coefficient of both $g(z)$ and $h(z)$ must equal one. The call to the function is

```
divpoly(num, den, n)
```

The `multpoly` Function

The function `multpoly` returns a vector containing the coefficients of the product $g(z)h(z)$ of two polynomials, where $g(z)$ and $h(z)$ have coefficients specified in the vectors `alpha` and `beta`, respectively. The zero degree coefficient in both $g(z)$ and $h(z)$ must have value one. The call to the function is

```
multpoly(alpha, beta)
```

The `poly` Function

The function `poly` returns a vector of length `n` that contains the values of the polynomial $g(z)$ evaluated at the values in the vector `x`. The coefficients of $g(z)$ are specified in the vector `coeffs`. The call to the function is

```
poly(coeffs, x)
```

The `polyrt` Function

The function `polyrt` finds the roots of a polynomial with coefficients specified in the vector `coeffs`. The zero degree coefficient must be one. The function has two arguments that are optional. The integer `m` specifies the maximum number of iterations of the procedure. The default is `m = 100`. The real number `eps` is the convergence criteria, having default 1×10^{-6}. The call to the function is

```
polyrt(coeffs, m = 100, eps = 1.e-6)
```

The function `polyrt` returns a list containing two items. The first item is the complex vector `roots` containing the roots of the polynomial. The second item is an integer error indicator `error`. If `error = 0`, then the algorithm converged; if not, then `error = 1`.

The `rtpoly` Function

The function `rtpoly` returns a vector containing the non-zero order coefficients of the polynomial having roots specified in the vector `roots`. (The zero order coefficient is one.) The call to the function is

```
rtpoly(roots)
```

3.3 Using *Timeslab in R* and *R*

A number of *R* and *Timeslab* functions are available for transforming time series. They are defined and explained below.

3.3.1 *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.

3.3.2 *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*.

Power Transformations

If a transformation of the time series in `x` is desired, in *R* it is easy to raise each element of an array to a power `r` by using `y <- x^r`. There are four functions in *R* for computing logarithms.

`log(x, base = exp(1))`: computes the natural logarithm of `x`.

`logb(x, base = exp(1))`: computes the natural logarithm of `x`.

- For both `log` and `logb`, if the value of `base` is something other than `exp(1)`, then the function computes the logarithm having base `num` as specified by `base = num`.

`log10(x)`: computes the base 10 logarithm of `x`.

`log2(x)`: computes the base 2 (binary) logarithm of `x`.

- `log10` and `log2` are only special cases, but are usually more efficient computationally than `log` and `logb`.

The `diff` Function

The *R* function `diff` returns lagged differences. There are a number of forms this function can take. We will focus on the form

```
diff(x, lag = 1, differences = 1)
```

where the required argument `x` is the variable containing the time series. The optional argument `lag` is an integer indicating which lag to use. The default value is `1`. The optional argument `differences` is an integer indicating the order of the difference. The default value is `1`. If `x` is a vector of length n, and `differences = 1`, then the computed result is $x_{t+\texttt{lag}} - x_t$ for each value of $t = 1, \ldots, n - \texttt{lag}$. If `differences` is larger than one, then the algorithm is applied recursively to `x`.

Example 3.2 `Taking Differences of the Airline Data`
The variable `x.air` *was created in Example 1.1, and contains 144 observations of the monthly number of airline passengers from 1949 to 1960. Because the variance appears to increase in time, as suggested in Section 3.1.2, we first take the natural log of the series, and assign it to the variable* `air`*. The next two lines take the 12th difference, and then the first difference. Figure 3.1 contains plots of the four series. The final (two) line(s) plots the final result, which corresponds to the bottom left graph in Figure 3.1.*

```
air <- log(x.air)
air.12.diff <- diff(air, lag = 12)
air.diff <- diff(air.12.diff)
plot(air.diff, type = "l", xlab = "t",
     ylab = "Transformed Airline Data",
     main = "Time Plot of Transformed Airline Data")
```

The same result obtained by the first three lines can be obtained in one line in the following manner.

```
diff(diff(log(x.air), lag = 12))
```

The `submns` and `divsds` Functions

There are two functions in *Timeslab in R* for calculating seasonal means and variances of a data set. They can also be used to subtract seasonal means (`submns`) or to divide by seasonal standard deviations (`divsds`). The call to the function `submns` takes the following form.

```
submns(x, d = 1, iopt = 1, xbar = rep(0,d))
```

The only required argument `x` is a vector containing the time series. The argument `d` indicates the period; so for monthly data, `d = 12`. The default value of `d = 1`. In this case, the usual sample mean $\bar{x} = n^{-1} \sum_{t=1}^{n} x_t$ is computed. The argument `iopt` indicates what the function is to compute and return. Values of `iopt` are

`iopt = 0`: the `d` periodic means are computed, but are not subtracted from the data. In this case, the function returns the sample mean of the series if `d = 1`, and if `d` is greater than one, `submns` returns a list containing two items: (1) a vector `x` containing the original time series, and (2) a vector `xbar` containing the `d` seasonal means.

`iopt = 1`: the `d` periodic means are computed and are subtracted from the data. `iopt = 1` is the default. In this case, the function returns a list containing two items: (1) a vector `x` containing the time series with the seasonal means subtracted, and (2) a vector `xbar` containing the `d` seasonal means.

`iopt = 2`: the seasonal means provided in the argument `xbar` are added to the data. In this case, the function returns a list containing (1) a vector `x` containing the original time series with the user-provided seasonal means added to the data, and (2) a vector `xbar` containing the seasonal means.

The final optional argument `xbar` is a vector of length `d` containing the seasonal means.

The call to the function `divsds` is

```
divsds(x, d = 1, iopt = 1, sds = rep(0,d))
```

The arguments of `divsds` are defined in a manner similar to those in `submns`. However, `divsds` divides (or multiplies) by the seasonal standard deviations, depending upon the value of `iopt`. If `d = 1`, the operations use the overall standard deviation $\hat{\sigma}^2 = n^{-1}\sum_{t=1}^{n}(x_t - \bar{x})^2$.

Example 3.3 Seasonal Means of Airline Data.
The airline data again serves as a good illustration. Refer back to the natural logarithm of the original series, created in Example 3.2. Instead of differencing, the linear trend can also be explained through a least squares regression analysis using the model

$$y_t = \log[x_t] = \beta_0 + \beta_1 t + \varepsilon_t, \quad t = 1, \ldots, 144.$$

For each $t = 1, \ldots, 144$, the residual that remains after fitting this model is a measure of the deviation of (the natural logarithm of) the number of airline passengers from the expected number of passengers at time t. The seasonal mean of the residuals provides an estimate of the mean deviation of the (natural logarithm of the) monthly number of passengers from the overall mean. The R *code, including the call to the* `submns` *function is something like what is given below. Recall that the variable* `air` *is the natural logarithm of the airline data (see Example 3.2).*

```
air.res <- lsfit(1:length(air), air)$residuals
submns
```

Figure 3.6 contains two graphs. The top graph is a plot of the natural logarithm of the airline data (versus time) with the least squares fitted regression line superimposed. The bottom panel is a plot of the residuals (versus time).[3] *The additional symbols on the plot are of the seasonal mean corresponding to the month for that residual. For example, the first residual corresponds to the month of January in 1949, the thirteenth corresponds to the month of January in 1950, and so on. If you average the residuals of all the Januaries, the result is -0.0855. This value is superimposed at every January residual with a small red circle. The mean residual for all Februaries is plotted with a small green triangle, etc. The legend at the top of the plot provides the plotting symbol for each month.*

The `filt` Function

The moving average smoother from Section 3.1.3 is a special case of what is called a "linear filter,"

$$y_t = \beta_0 x_{t+m} + \beta_1 x_{t+m-1} + \cdots + \beta_m x_t, \quad t = 1, \ldots, n - m.$$

Linear filters will be discussed in greater detail in Section 4.3. For the meantime, the *Timeslab in R* function

[3] Note that these residuals do *not* have the desirable properties required for a traditional regression analysis using t or F tests!

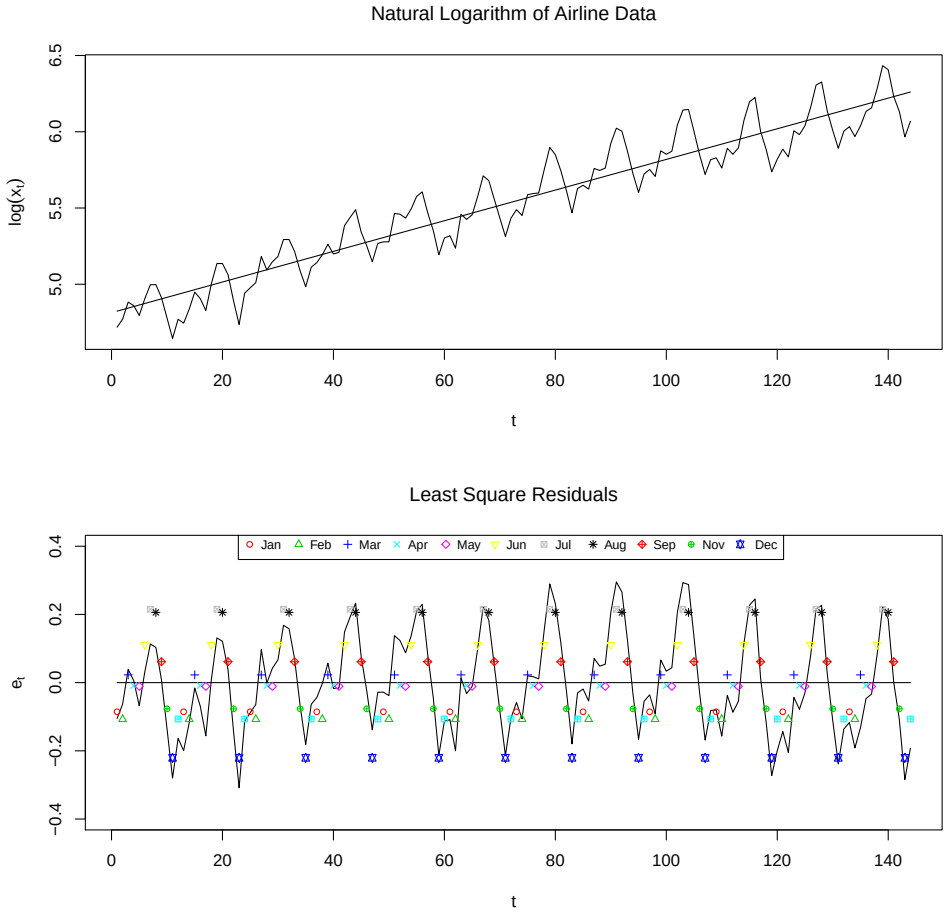

Figure 3.6: Plots of the natural logarithm of the airline data with a fitted least squares regression line superimposed (top) and a residual plot with the seasonal means plotted at each month.

```
filt(beta, beta0, x)
```

will perform such a filtering operation. The argument `beta` is a vector of length m containing the coefficients $\beta_1, \ldots, \beta_m$. The argument `beta0` is a scalar containing the coefficient β_0, and the argument `x` is a vector containing the time series x. The function returns the smoothed version of x.

Example 3.4 Using `filt` to Compute a Moving Average Smoother.
For a change of pace, let's consider the data set called `eriel`*, containing the levels of Lake Erie from 1921 to 1970. To access this data set, use the function* `datasets`*, as described in Section 1.4.1 on page 21. For ease of discussion, save the data set in a variable called* `eriel`.

```
eriel <- datasets()$eriel
```

To get a moving average smoothing of length six, say, the six coefficients $\beta_0 = \beta_5$ *are all* $\frac{1}{6}$. *To compute the smoothed series, the call to the* `filt` *function is*

```
beta <- rep(1/6, 5)
beta0 <- 1/6
filt(beta, beta0, eriel)
```

or equivalently, simply `filt(rep(1/6, 5), 1/6, eriel)`. *A plot of the original data with the moving average smoother superimposed is given in Figure 3.7.*

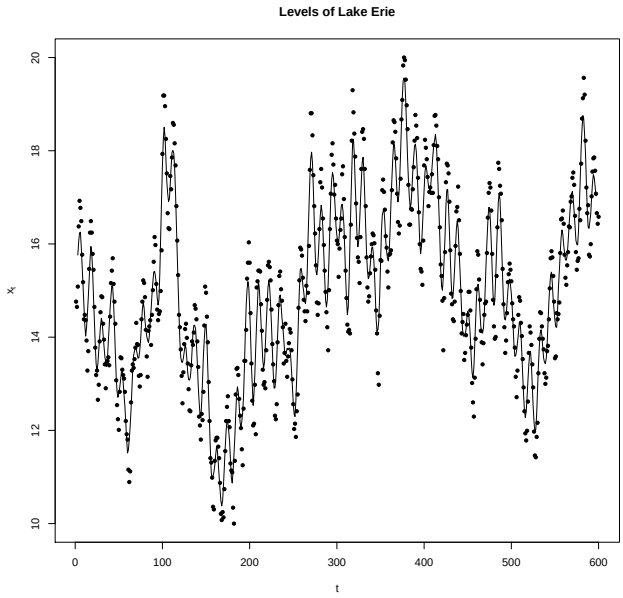

Figure 3.7: Levels of Lake Erie (dots) with moving average smoother superimposed (line).

3.4 Pulling It All Together

In this section, we will take the concepts we presented on transformation and forecasting. You may recall in our discussion of exponential for forecasting, we mentioned that we would provide a function that will choose the value of α and superimpose the plots of the data and the resulting fitted values. We do that now.

Example 3.5 Exponential Smoothing.
The function `expsm` *determines the value of* α *to be used in simple exponential smoothing and uses it to smooth the data by making use of the* `diffeq` *function. Following the listing of the function, we give two examples of using simple exponential smoothing.*

```
expsm <- function(x, first = 0, last = 1, k = 101)
#-------------------------------------------------------------------------
# R function to find value of alpha in exponential smoothing, and then
# use the exponential smoother to find fitted values, and superimpose
# those as the same plot as the original data.
#
# INPUT: x = a real vector containing the data to be smoothed.
#        first = a real scalar containing the minimum value of
#                alpha to be considered.  The default is first = 0.
#        last = a real scalar containing the maximum value of alpha
#               to be considered.  The default is last = 1.
#        k = an integer scalar containing the number of values of alpha
#            to consider.  The default is 101.
#
#  VALUE: The function expsm creates a set of two plots.  The first
#         is a plot of values of the sum of square errors for varying
#         values of the smoothing constant (a) versus a.  The second
#         plot is a plot of the original series (line) with the
#         fitted values using exponential smoothing with value alpha
#         = min_a (SS(a)) superimposed as points.  The function also
#         returns a list containing the following objects.
#         alpha = a real scalar between 0 and 1 containing the value
#                 of the smoothing constant the minimizes the sume of
#                 squared errors.
#         min.SS = a real scalar containing the minimum value of the
#                  sum of squared errors for all considered values
#                  of smoothing constant.
#         xhat = a real vector of length n containing the fitted
#                values using exponential smoothing, based on the value
#                alpha.
#         avalues = a vector of length k containing all considered
#                   values of smoothing constant.
```

```
#         all.SS = a vector of length k containing the error sum of
#                  squares for all a values.
#
#  FORTRAN subroutine: diffeq (through diffeq function)
#
#  Written for R: 10/29/2010 JLH
#-------------------------------------------------------------------------
{
  if(last < first) {
    cc <- last
    last <- first
    first <- cc
  }
  if(first < 0 || last > 1)
    stop("\n The value of alpha must be between 0 and 1.\n")

  n     <- length(x)
  SS    <- rep(0, length = k)
  avals <- seq(first, last, length = k)

  for(i in 1:k) {
    xhat  <- diffeq((avals[i]-1), n = n, e = c(0, avals[i]*x))
    res   <- x - xhat
    SS[i] <- dot(res, res)
  }

  alpha <- avals[which(SS == min(SS))]
  SSmin <- SS[which(SS == min(SS))]

  xhat <- diffeq(alpha-1, n = n, e = c(0, alpha*x))

  del.x <- 0.05*(last - first)
  ylim <- c(min(x, xhat), max(x, xhat))
  par(mfrow = c(2, 1),  oma = rep(0, length = 4), mar = c(5, 4, 2, 2) + 0.1)

  plot(avals, SS, xlab = "a", ylab = "S(a)", pch = 20)
  mtext("Sum of Squares: Exponential Smoothing", line = 0.25, cex = 1.25)
  points(alpha, SSmin, col = "red", pch = 19)
  text(last-del.x, max(SS), labels = bquote(alpha == .(alpha)),
       col = "red")
  plot(xhat, ylim = ylim, type = "p", pch = 20, col="blue",
       xlab = "t", ylab = expression(x[t]))
```

```
  mtext("Series and Forecasts", line = 1.25, cex = 1.25)
  mtext(bquote(alpha == .(alpha)), line = 0.25, cex = 1)
  lines(1:n, x, lwd = 2)

  return(list(alpha = alpha, min.SS = SSmin, xhat = xhat,
              avalues = avals, all.SS = SS))
}
```

1. *First we will generate data from an autoregressive process, mentioned earlier in Section 3.2.6, and then use the* `expsm` *function to find a value of the smoothing constant* α*. The code will look something like what is below. The graph generated by the call to the* `expsm` *function is seen in Figure 3.8.*

```
x <- ardt(c(-0.7, 0.1), 250)$x  #Get data from an AR process
alpha <- expsm(x)$alpha
```

For this series, the value of the smoothing constant is $\alpha = 0.8$*. That means, for this data set, the value of* a *that minimized*

$$S(a) = \sum_{t=2}^{250} \left\{ x_t - \sum_{j=1}^{t-1} a(1-a)^{j-1} x_{t-j} \right\}^2,$$

was $a = 0.8$*. So we set* $\alpha = 0.8$ *to find the smoothed values represented by the blue dots in the bottom graph in Figure 3.8.*

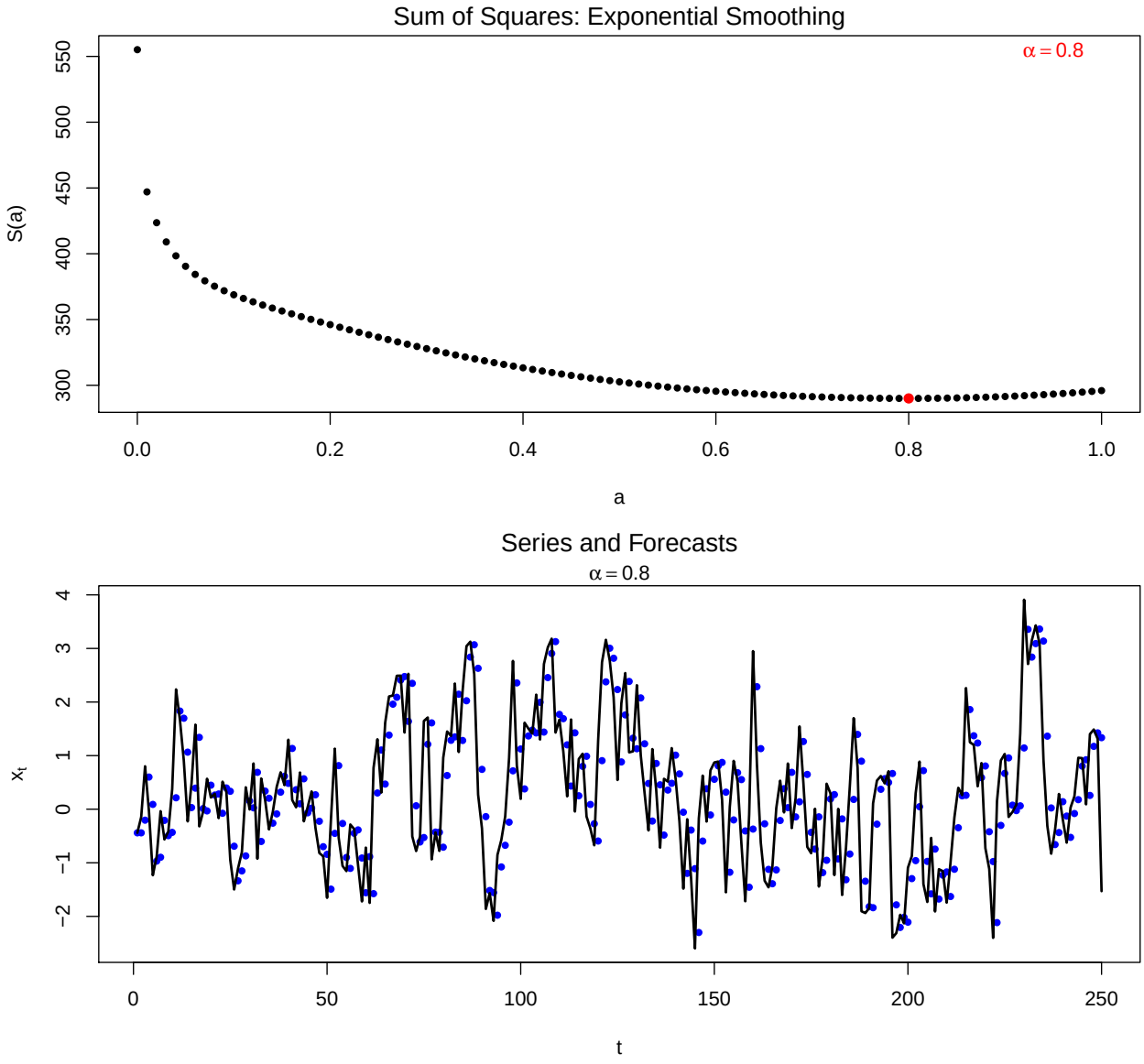

Figure 3.8: Graphs generated by the `expsm` function using data from an autoregressive process. The top graph is of $S(a)$ versus a; the bottom graph is of the original series (line) with the fitted values via exponential smoothing with $\alpha = 0.8$ superimposed (blue dots).

2. *In this example, we apply exponential smoothing to the first difference of the natural logarithm of the Beveridge wheat price index. We examined the first difference of that data set in Section 2.3.3. The natural logarithm of the first difference is taken to remove what appears to be nonconstant variance in the series after differencing. Applying the following lines of* R *code*

```
x <- diff(log(datasets()$bev))  #Tranform data.
exp.bev <- expsm(x)
exp.bev$alpha
exp.bev$min.SS
```

provides us with a smoothing coefficient of $\alpha = 0.02$*, having sum of squared error equal to 11.88034. If we want to further refine the value of* α*, we can do so by reapplying the* `expsm` *function, and placing more restrictive* `first` *and* `last` *values on those considered for* a*, as well as increasing the number of values considered from* `k = 101` *to, say,* `k = 501`*.*

```
exp.bev <- expsm(x, first = 0.015, last = 0.025, k = 501)
exp.bev$alpha
exp.bev$min.SS
```

The result is seen in Figure 3.9. The sum of squared errors for the refined value of α *is 11.87941.*

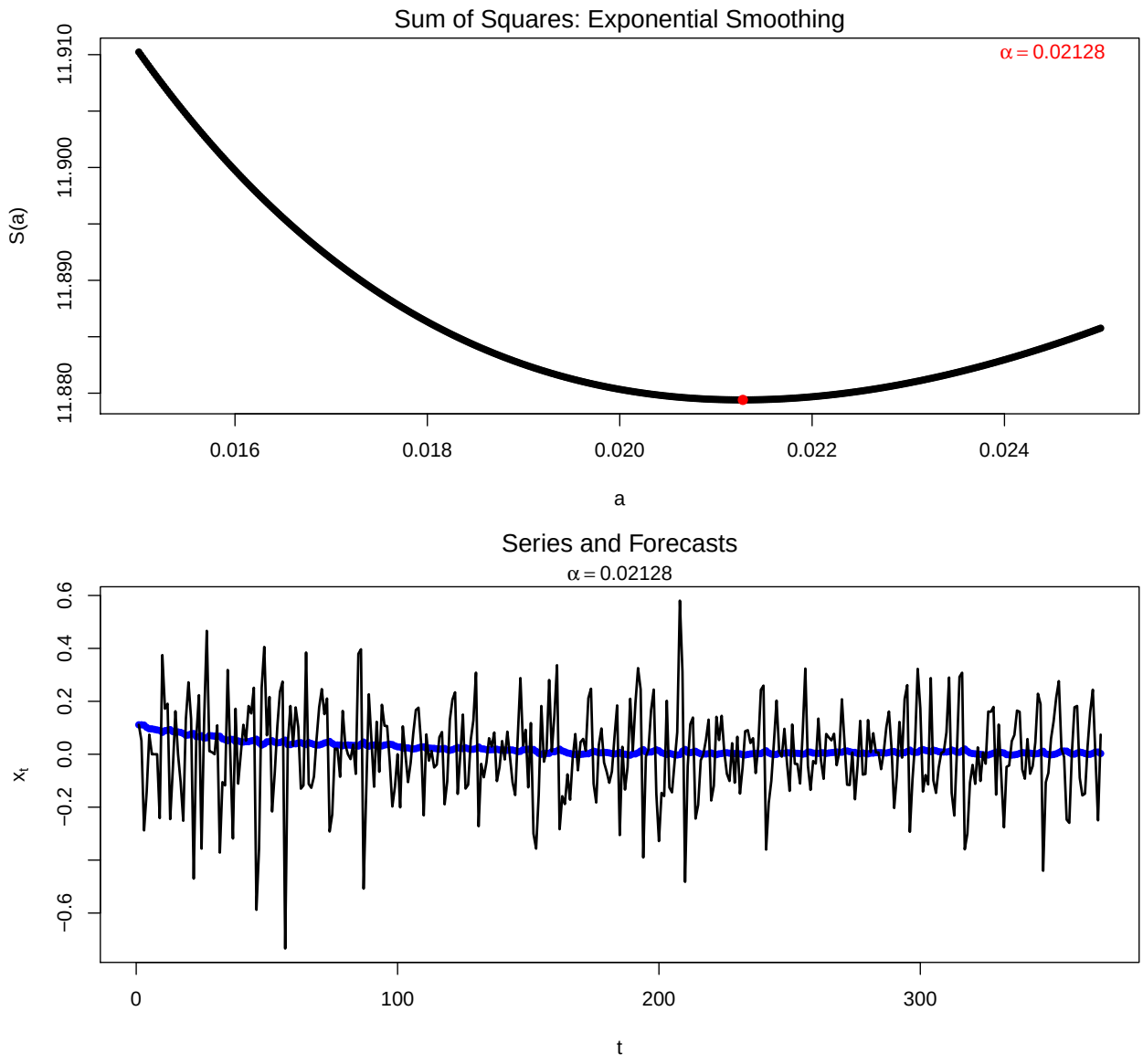

Figure 3.9: Graphs generated by the `expsm` function using the first difference of the natural logarithm of the Beveridge wheat price index for refining values of α. The top graph gives a value for $\alpha = 0.02128$. The bottom graph is of the original series (line) with the fitted values found via exponential smoothing with $\alpha = 0.02128$ superimposed (blue dots).

3.5 Exercises

3.5.1 Theoretical Exercises

T3.1 Suppose that

$$y_t = \mu_{j(t)} + \varepsilon_t, \quad t = 1, \ldots, n = kd,$$

where the ε's are *iid* N(0,σ^2), and $j(t) = \text{mod}(t-1, d) + 1$, and

$$\text{mod}(l, m) = l - m[l/m],$$

and $[x]$ denotes the greatest integer less than or equal to x.

(a) Write this model as $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\beta} = (\mu_1, \ldots, \mu_d)^T$ and $\boldsymbol{X}$ is $(n \times d)$.

(b) Show that the least squares estimate $\hat{\boldsymbol{\beta}}$ of $\boldsymbol{\beta}$ are the seasonal means described in Section 3.1, and that $\hat{\boldsymbol{\beta}} \sim N_d(\boldsymbol{\beta}, k^{-1}\sigma^2 \boldsymbol{I}_d)$.

(c) Suppose $d = 7$ and

$$y_t = \mu + \sum_{k=1}^{3} \left[a_k \cos \frac{2\pi(t-1)k}{7} + b_k \sin \frac{2\pi(t-1)k}{7} \right] + \varepsilon_t,$$

for $t = 1, \ldots, n = 7m$. Let $\boldsymbol{\gamma} = (\mu, a_1, a_2, a_3, b_1, b_2, b_3)^T$. Write this model as $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$, find the least squares estimator of $\boldsymbol{\gamma}$, and thus show that

$$\frac{n}{4}(\hat{a}_k^2 + \hat{b}_k^2) = \hat{f}(\omega_{km+1}),$$

where $\hat{f}$ denotes the periodogram of the y's.

T3.2 Show that differencing a d^{th} degree polynomial d time will remove the polynomial trend.

T3.3 Show that the sth difference of a set of data that is periodic with period s is no longer periodic.

T3.4 Show that in the exponential smoothing model,

$$\hat{x}(t+1) = \alpha x_t + (1-\alpha)\hat{x}_t.$$

T3.5 Show that if the zeroes of the characteristic polynomial for a homogeneous difference equation for z are all greater than one in modulus, then $\lim_{t\to\infty} z_t = 0$.

T3.6 Use Theorem 3.1 to show that the difference equation

$$z_t - 2\cos\omega z(t-1) + z(t-2) = 0, \quad t \geq 2,$$

with $z_0 = 1$ and $z_1 = \cos\omega$ has solution $z_t = \cos t\omega$.

T3.7 The informative quantile function of a continuous random variable X (as opposed to that for a data set) is defined by

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

where Q is the quantile function of X. The quantile function is defined by

$$Q(u) = x \quad \Longleftrightarrow \quad F(x) = P(X \leq x) = u,$$

assuming the cumulative distribution function (CDF) F of X is strictly increasing. Show that the true informative quantile plot of any uniformly distributed random variable is the same.

T3.8 Show that the Parzen kernel 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.)

3.5.2 Computational Exercises

C3.1 Write a function that will use the `filt` function to form

$$y_t = \sum_{j=-k}^{k} \alpha_j x_{t-j}, \quad t = 1, \ldots, n$$

where $\alpha_j = \rho^{|j|}$ and ρ, n, k, and x are specified by the user of the function. Notice that `filt` is designed to have only coefficients with nonnegative indices and so you will have to be clever in how you form the `beta` and `beta0` to be used by `filt`. (*Hint:* `beta0` will be α_{-k}.) Note also that the input array will have to be of length $n + 2k$. Try out the function for white noise input and for $\rho = 0.8$ and $\rho = 0.5$ with $k = 1, 2, 3$ for each ρ. You will need to divide y_t by the sum of the weights. Be sure when you superimpose the plots of x_t and y_t that you pay attention to the indexing so that the two series line up properly.

C3.2 Use the `poly` function to form a time series consisting of 100 values of a quadratic polynomial over the interval [0, 1] (you can choose the coefficients). Use the `diff` function twice in order to apply first differencing twice. Is the result a constant?

C3.3 Form a series

$$x_t = 1 + 0.01t + \cos\left(\frac{2\pi(t-1)}{10}\right), \quad t = 1, \ldots, 100,$$

and use the `extend` function with `d1 = 1` and `d2 = 10` to get the next 30 values of the series. Plot the original data and the extended series on the same graph.

C3.4 Verify that the following function gives the same result for `n = 100`, `amp = 2.0`, and `per = 5.0` as does the call

```
x <- 2*cos(2*pi*((1:100) - 1)/5).
```

Notice that this new function uses the result of Exercise T3.6 to generate a cosine curve.

```
cosde <- function(n = 100, amp = 2, per = 5)
#-----------------------------------------------------------------------
# R function to generate a cosine curve of length n with amplitude
# amp and period per by using the difference equation method.
#
# INPUT: n = an integer scalar indicating the number of points to
#            generate from the cosine curve.
#        amp = a real scalar containing the amplitude of the cosine.
#        per = a real scalar containing the period of the cosine.
#
# VALUE: The function cosde returns a vector of length n with the
#        values of a cosine curve with amplitude amp and period per.
#-----------------------------------------------------------------------
{
  theta  <- 2 * pi / per
  f      <- rep(0, length = n)
  f[1:2] <- cos(theta*(0:1))
  ct     <- -2*f[2]
  alpha  <- c(ct, 1)
  x      <- diffeq(alpha, n, f)

  return(amp*x)
}
```

C3.5 Suppose that X is a random variable having the binomial distribution with parameters n and p; that is

$$f(x) = \Pr(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, 1, \ldots, n.$$

(a) Show that $g(x) = \log f(x)$ satisfies the difference equation

$$g(x) - g(x-1) = h(x) = \log\left(\frac{n-x+1}{x}\,\frac{p}{1-p}\right), \quad x \geq 1.$$

(*Hint:* Find $f(x)/f(x-1)$. Notice that the subscript in terms of t in the definition of our difference equation has been replaced by (x).)

(b) Find $g(0)$ and thus use the `diffeq` function to write a function that will find $g(1), \ldots, g(n)$, and thus $f(0), f(1), \ldots f(n)$ for user-specified n and p. Note that you will need to add 1 to all of the indices of the arrays.

(c) Check your new function by comparing its result with that of using the R function `dbinom(x, n, p)` for `n = 100` and `p = 0.4`.

Note that other discrete probability distributions can be found in a similar fashion.

C3.6 Use the R function `lsfit` to find the residuals from a linear trend for the natural logarithm of the airline data. Does this regression remove the seasonal cycle in the data? From these residuals, construct two data sets: (1) the 12^{th} differences, and (2) the result of subtracting monthly means. Use the `wntest` function to analyze each of these two series. Which of the two series looks more like white noise?

C3.7 Write a function that will do the following:

(a) Generate a data set of length `n` made up of the sum of a linear trend having intercept `a` and slope `b`, a pure cosine having amplitude `amp` and period `per`, and a white noise series having variance `sig2`.

(b) Use the R function `lsfit` to find the residuals from regressing the data on a linear trend and cosine.

(c) Use the `wntest` function to investigate the whiteness of the residuals.

Try the function out for a variety of coefficients. Find a set of coefficients so that the linear part predominates, another set of coefficients so that the sinusoid does, and a third so that the error series predominates.

C3.8 Extend the Beveridge wheat price data by 120 points using the inverse differencing method with $d = 1$. Plot the results with the vertical axes ranging from 0 to 500. Use the `lsfit` function to fit a linear trend to the data and then extend this line by 120 points and plot the data and the line. Use the same scaling factors as in the inverse differencing method. How do the two plots compare?

C3.9 Write a function that will calculate and plot the exponential smoothing weights

$$\beta_j = \alpha(1-\alpha)^{j-1}m \quad j = 1, \ldots, n,$$

for user specified values of $\alpha \in (0, 1)$ and n. The easy way to calculate the β values is to find the logarithms and then use the `exp` function to get their actual values. For what values of α do the weights decay to zero the slowest?

C3.10 The matrix operations described in Appendix A.4 and calculated by the *Timeslab* functions `mchol`, `gram.schmidt`, and `swp` are related in a variety of ways.

(a) Generate an matrix `X` of dimension 10×4 containing randomly generated standard normal variates.

(b) Use the `gram.schmidt` function to find the factors $\boldsymbol{Q}$ and $\boldsymbol{R}$ in the Gram-Schmidt decomposition of $\boldsymbol{X}$. Use the resulting matrices `Q` and `R` to represent the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. Verify that $\boldsymbol{X} = \boldsymbol{QR}$. Also check that `Q` is orthogonal by finding $\boldsymbol{Q}^T\boldsymbol{Q}$.

(c) Compute $\boldsymbol{Y} = \boldsymbol{X}^T\boldsymbol{X}$, and assign it to a matrix `Y`. Using `mchol`, find the modified Cholesky factors $\boldsymbol{L}$ and $\boldsymbol{D}$. Verify that $\boldsymbol{L} = \boldsymbol{R}^T$ and $\boldsymbol{D} = \boldsymbol{Q}^T\boldsymbol{Q}$.

(d) Now use the `swp` function to sweep $\boldsymbol{Y}$ on its first diagonal using `swp(y, 1, 1)`. Note that the second through fourth elements of the first row of the result are the same as the corresponding elements of $\boldsymbol{R}$, thus the below diagonal elements of the first column of $\boldsymbol{L}$. Next sweep the resulting matrix on its second diagonal by `swp(swp(y, 1, 1)$A, 2, 2)`. How is that result related to $\boldsymbol{L}$? To $\boldsymbol{R}$? Continue this process until the diagonals have all been used. Verify that the result is the inverse of $\boldsymbol{Y} = \boldsymbol{X}^T\boldsymbol{X}$ using the R function `solve` and by multiplying the final result by $\boldsymbol{X}^T\boldsymbol{X}$.