Table 7.2: Applying Holt's linear method with $\alpha = 0.8321$ and $\beta^* = 0.0001$ to Australian air passenger data (millions of passengers).

Year	Time	Observation	Level	Slope	Forecast
	t	y_t	ℓ_t	b_t	$\hat{y}_{t\|t-1}$
1989	0		15.57	2.102	
1990	1	17.55	17.57	2.102	17.67
1991	2	21.86	21.49	2.102	19.68
1992	3	23.89	23.84	2.102	23.59
1993	4	26.93	26.76	2.102	25.94
1994	5	26.89	27.22	2.102	28.86
1995	6	28.83	28.92	2.102	29.33
1996	7	30.08	30.24	2.102	31.02
1997	8	30.95	31.19	2.102	32.34
1998	9	30.19	30.71	2.101	33.29
1999	10	31.58	31.79	2.101	32.81
2000	11	32.58	32.80	2.101	33.89
2001	12	33.48	33.72	2.101	34.90
2002	13	39.02	38.48	2.101	35.82
2003	14	41.39	41.25	2.101	40.58
2004	15	41.60	41.89	2.101	43.35
2005	16	44.66	44.54	2.101	44.00
2006	17	46.95	46.90	2.101	46.65
2007	18	48.73	48.78	2.101	49.00
2008	19	51.49	51.38	2.101	50.88
2009	20	50.03	50.61	2.101	53.49
2010	21	60.64	59.30	2.102	52.72
2011	22	63.36	63.03	2.102	61.40
2012	23	66.36	66.15	2.102	65.13
2013	24	68.20	68.21	2.102	68.25
2014	25	68.12	68.49	2.102	70.31
2015	26	69.78	69.92	2.102	70.60
2016	27	72.60	72.50	2.102	72.02
	h				$\hat{y}_{t+h\|t}$
	1				74.60
	2				76.70
	3				78.80
	4				80.91
	5				83.01

The very small value of β^* means that the slope hardly changes over time.

Damped trend methods

The forecasts generated by Holt's linear method display a constant trend (increasing or decreasing) indefinitely into the future. Empirical evidence indicates that these methods tend to over-forecast, especially for longer forecast horizons. Motivated by this observation, Gardner & McKenzie (1985) introduced a parameter that "dampens" the trend to a flat line some time in the future. Methods that include a damped trend have proven to be very successful, and are arguably the most popular individual methods when forecasts are required automatically for many series.

In conjunction with the smoothing parameters α and β^* (with values between 0 and 1 as in Holt's method), this method also includes a damping parameter $0 < \phi < 1$:

$$\begin{aligned}
\hat{y}_{t+h|t} &= \ell_t + (\phi + \phi^2 + \cdots + \phi^h)b_t \\
\ell_t &= \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}.
\end{aligned}$$

If $\phi = 1$, the method is identical to Holt's linear method. For values between 0 and 1, ϕ dampens the trend so that it approaches a constant some time in the future. In fact, the forecasts converge to $\ell_T + \phi b_T/(1-\phi)$ as $h \to \infty$ for any value $0 < \phi < 1$. This means that short-run forecasts are trended while long-run forecasts are constant.

In practice, ϕ is rarely less than 0.8 as the damping has a very strong effect for smaller values. Values of ϕ close to 1 will mean that a damped model is not able to be distinguished from a non-damped model. For these reasons, we usually restrict ϕ to a minimum of 0.8 and a maximum of 0.98.

Example: Air Passengers (continued)

Figure 7.3 shows the forecasts for years 2017–2031 generated from Holt's linear trend method and the damped trend method.

```
fc <- holt(air, h=15)
fc2 <- holt(air, damped=TRUE, phi = 0.9, h=15)
autoplot(air) +
  autolayer(fc, series="Holt's method", PI=FALSE) +
  autolayer(fc2, series="Damped Holt's method", PI=FALSE) +
  ggtitle("Forecasts from Holt's method") + xlab("Year") +
  ylab("Air passengers in Australia (millions)") +
  guides(colour=guide_legend(title="Forecast"))
```

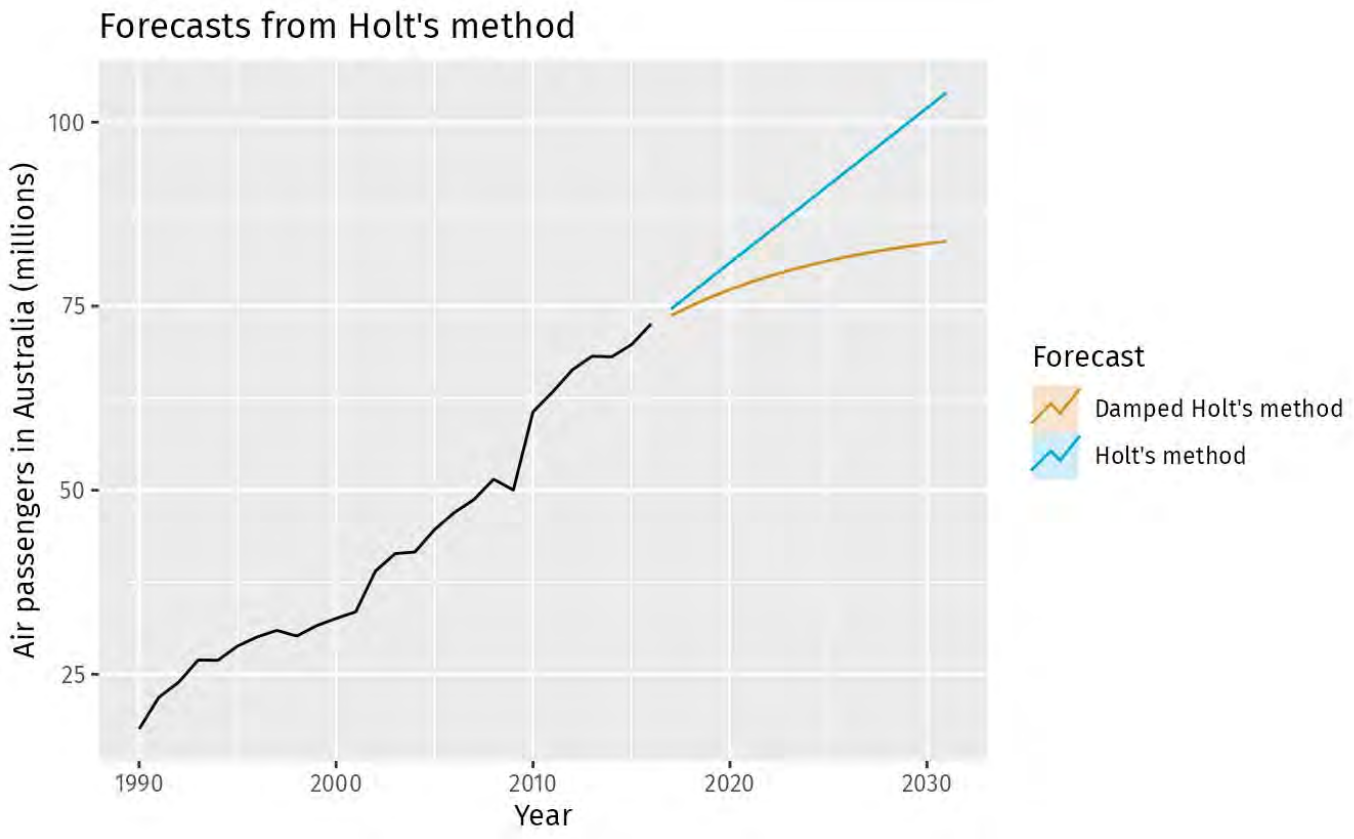


Figure 7.3: Forecasting total annual passengers of air carriers registered in Australia (millions of passengers, 1990–2016). For the damped trend method, $\phi = 0.90$.

We have set the damping parameter to a relatively low number ($\phi = 0.90$) to exaggerate the effect of damping for comparison. Usually, we would estimate ϕ along with the other parameters. We have also used a rather large forecast horizon ($h = 15$) to highlight the difference between a damped trend and a linear trend. In practice, we would not normally want to forecast so many years ahead with only 27 years of data.

Example: Sheep in Asia

In this example, we compare the forecasting performance of the three exponential smoothing methods that we have considered so far in forecasting the sheep livestock population in Asia. The data spans the period 1961–2007 and is shown in Figure 7.4.

```
autoplot(livestock) +
  xlab("Year") + ylab("Livestock, sheep in Asia (millions)")
```

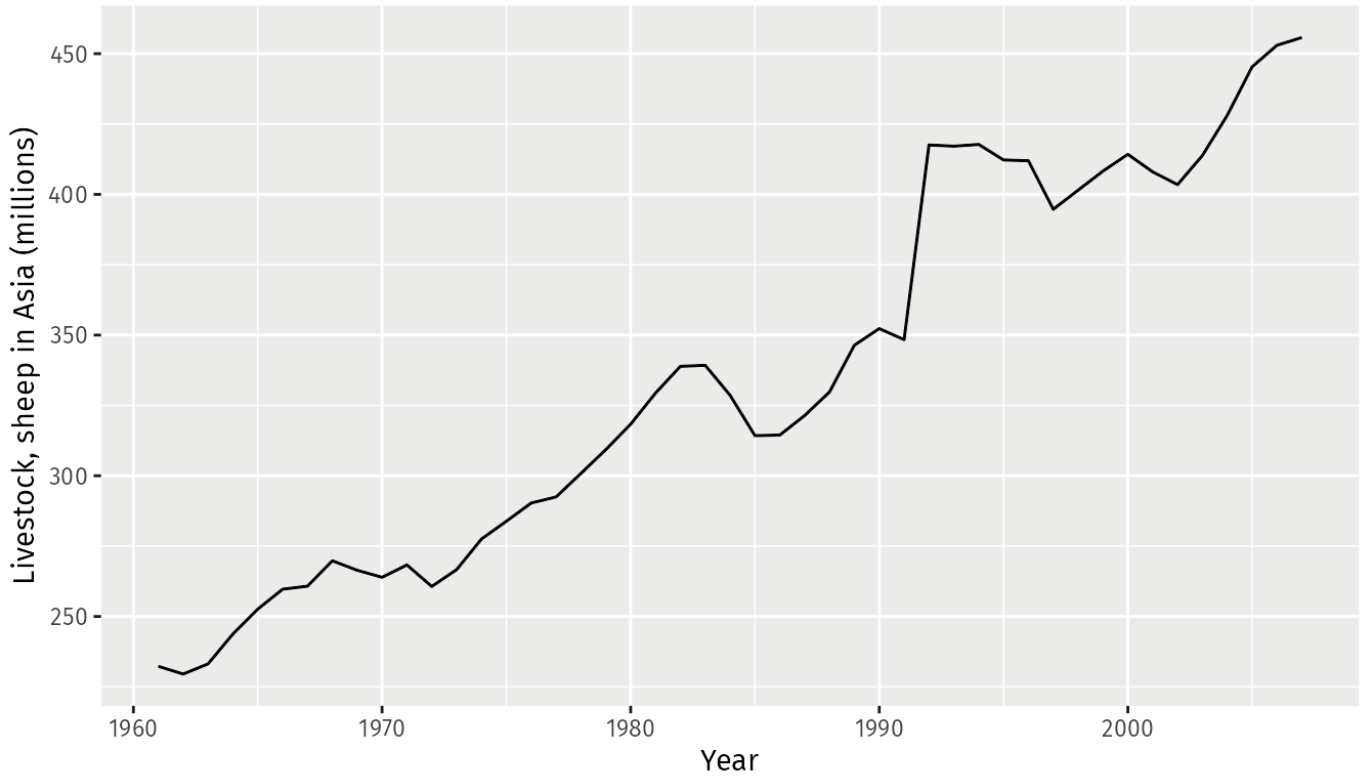


Figure 7.4: Annual sheep livestock numbers in Asia (in million head)

We will use time series cross-validation to compare the one-step forecast accuracy of the three methods.

```
e1 <- tsCV(livestock, ses, h=1)
e2 <- tsCV(livestock, holt, h=1)
e3 <- tsCV(livestock, holt, damped=TRUE, h=1)
# Compare MSE:
mean(e1^2, na.rm=TRUE)
#> [1] 178.3
mean(e2^2, na.rm=TRUE)
#> [1] 173.4
mean(e3^2, na.rm=TRUE)
#> [1] 162.6
# Compare MAE:
mean(abs(e1), na.rm=TRUE)
#> [1] 8.532
mean(abs(e2), na.rm=TRUE)
#> [1] 8.803
mean(abs(e3), na.rm=TRUE)
#> [1] 8.024
```

Damped Holt's method is best whether you compare MAE or MSE values. So we will proceed with using the damped Holt's method and apply it to the whole data set to get forecasts for future years.

```
fc <- holt(livestock, damped=TRUE)
# Estimated parameters:
fc[["model"]]
#> Damped Holt's method
#>
#> Call:
#>  holt(y = livestock, damped = TRUE)
#>
#>   Smoothing parameters:
#>     alpha = 0.9999
#>     beta  = 3e-04
#>     phi   = 0.9798
#>
#>   Initial states:
#>     l = 223.35
#>     b = 6.9046
#>
#>   sigma:  12.84
#>
#>   AIC  AICc   BIC
#> 427.6 429.7 438.7
```

The smoothing parameter for the slope is estimated to be essentially zero, indicating that the trend is not changing over time. The value of α is very close to one, showing that the level reacts strongly to each new observation.

```
autoplot(fc) +
  xlab("Year") + ylab("Livestock, sheep in Asia (millions)")
```

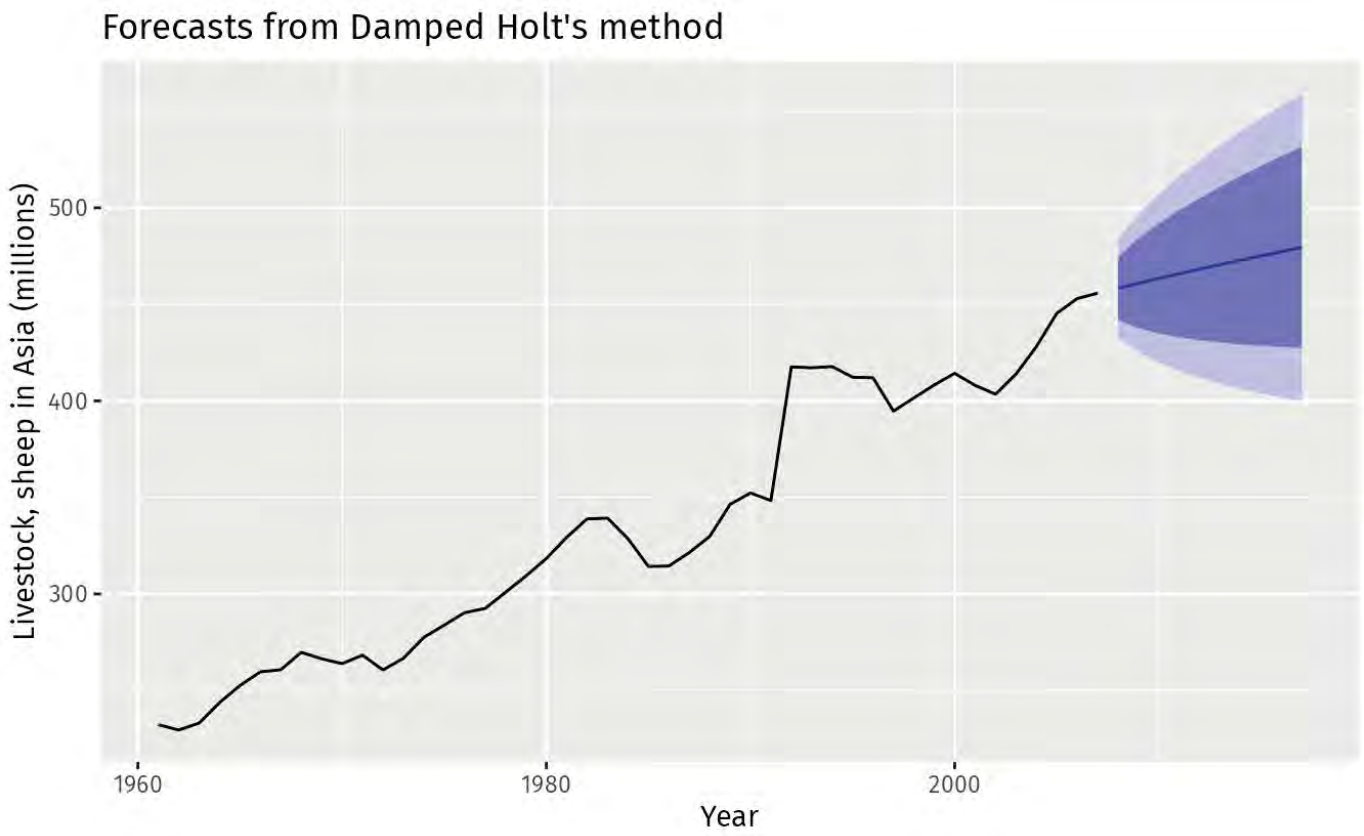


Figure 7.5: Forecasting livestock, sheep in Asia: comparing forecasting performance of non-seasonal method.

The resulting forecasts look sensible with increasing trend, and relatively wide prediction intervals reflecting the variation in the historical data. The prediction intervals are calculated using the methods described in Section 7.7.

In this example, the process of selecting a method was relatively easy as both MSE and MAE comparisons suggested the same method (damped Holt's). However, sometimes different accuracy measures will suggest different forecasting methods, and then a decision is required as to which forecasting method we prefer to use. As forecasting tasks can vary by many dimensions (length of forecast horizon, size of test set, forecast error measures, frequency of data, etc.), it is unlikely that one method will be better than all others for all forecasting scenarios. What we require from a forecasting method are consistently sensible forecasts, and these should be frequently evaluated against the task at hand.

Bibliography

Gardner, E. S., & McKenzie, E. (1985). Forecasting trends in time series. *Management Science*, *31*(10), 1237–1246. [DOI]

Holt, C. C. (1957). *Forecasting seasonals and trends by exponentially weighted averages* (O.N.R. Memorandum No. 52). Carnegie Institute of Technology,

Pittsburgh USA. [DOI]

7.3 Holt-Winters' seasonal method

Holt (1957) and Winters (1960) extended Holt's method to capture seasonality. The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations — one for the level ℓ_t, one for the trend b_t, and one for the seasonal component s_t, with corresponding smoothing parameters α, β^* and γ. We use m to denote the frequency of the seasonality, i.e., the number of seasons in a year. For example, for quarterly data $m = 4$, and for monthly data $m = 12$.

There are two variations to this method that differ in the nature of the seasonal component. The additive method is preferred when the seasonal variations are roughly constant through the series, while the multiplicative method is preferred when the seasonal variations are changing proportional to the level of the series. With the additive method, the seasonal component is expressed in absolute terms in the scale of the observed series, and in the level equation the series is seasonally adjusted by subtracting the seasonal component. Within each year, the seasonal component will add up to approximately zero. With the multiplicative method, the seasonal component is expressed in relative terms (percentages), and the series is seasonally adjusted by dividing through by the seasonal component. Within each year, the seasonal component will sum up to approximately m.

Holt-Winters' additive method

The component form for the additive method is:

$$\begin{aligned}
\hat{y}_{t+h|t} &= \ell_t + hb_t + s_{t+h-m(k+1)} \\
\ell_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\
s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},
\end{aligned}$$

where k is the integer part of $(h-1)/m$, which ensures that the estimates of the seasonal indices used for forecasting come from the final year of the sample. The level equation shows a weighted average between the seasonally adjusted observation $(y_t - s_{t-m})$ and the non-seasonal forecast $(\ell_{t-1} + b_{t-1})$ for time t. The

trend equation is identical to Holt's linear method. The seasonal equation shows a weighted average between the current seasonal index, $(y_t - \ell_{t-1} - b_{t-1})$, and the seasonal index of the same season last year (i.e., m time periods ago).

The equation for the seasonal component is often expressed as

$$s_t = \gamma^*(y_t - \ell_t) + (1-\gamma^*)s_{t-m}.$$

If we substitute ℓ_t from the smoothing equation for the level of the component form above, we get

$$s_t = \gamma^*(1-\alpha)(y_t - \ell_{t-1} - b_{t-1}) + [1-\gamma^*(1-\alpha)]s_{t-m},$$

which is identical to the smoothing equation for the seasonal component we specify here, with $\gamma = \gamma^*(1-\alpha)$. The usual parameter restriction is $0 \leq \gamma^* \leq 1$, which translates to $0 \leq \gamma \leq 1-\alpha$.

Holt-Winters' multiplicative method

The component form for the multiplicative method is:

$$\begin{aligned}
\hat{y}_{t+h|t} &= (\ell_t + hb_t)s_{t+h-m(k+1)} \\
\ell_t &= \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\
s_t &= \gamma\frac{y_t}{(\ell_{t-1} + b_{t-1})} + (1-\gamma)s_{t-m}
\end{aligned}$$

Example: International tourist visitor nights in Australia

We apply Holt-Winters' method with both additive and multiplicative seasonality to forecast quarterly visitor nights in Australia spent by international tourists. Figure 7.6 shows the data from 2005, and the forecasts for 2016–2017. The data show an obvious seasonal pattern, with peaks observed in the March quarter of each year, corresponding to the Australian summer.

```
aust <- window(austourists,start=2005)
fit1 <- hw(aust,seasonal="additive")
fit2 <- hw(aust,seasonal="multiplicative")
autoplot(aust) +
  autolayer(fit1, series="HW additive forecasts", PI=FALSE) +
  autolayer(fit2, series="HW multiplicative forecasts",
    PI=FALSE) +
  xlab("Year") +
  ylab("Visitor nights (millions)") +
  ggtitle("International visitors nights in Australia") +
  guides(colour=guide_legend(title="Forecast"))
```

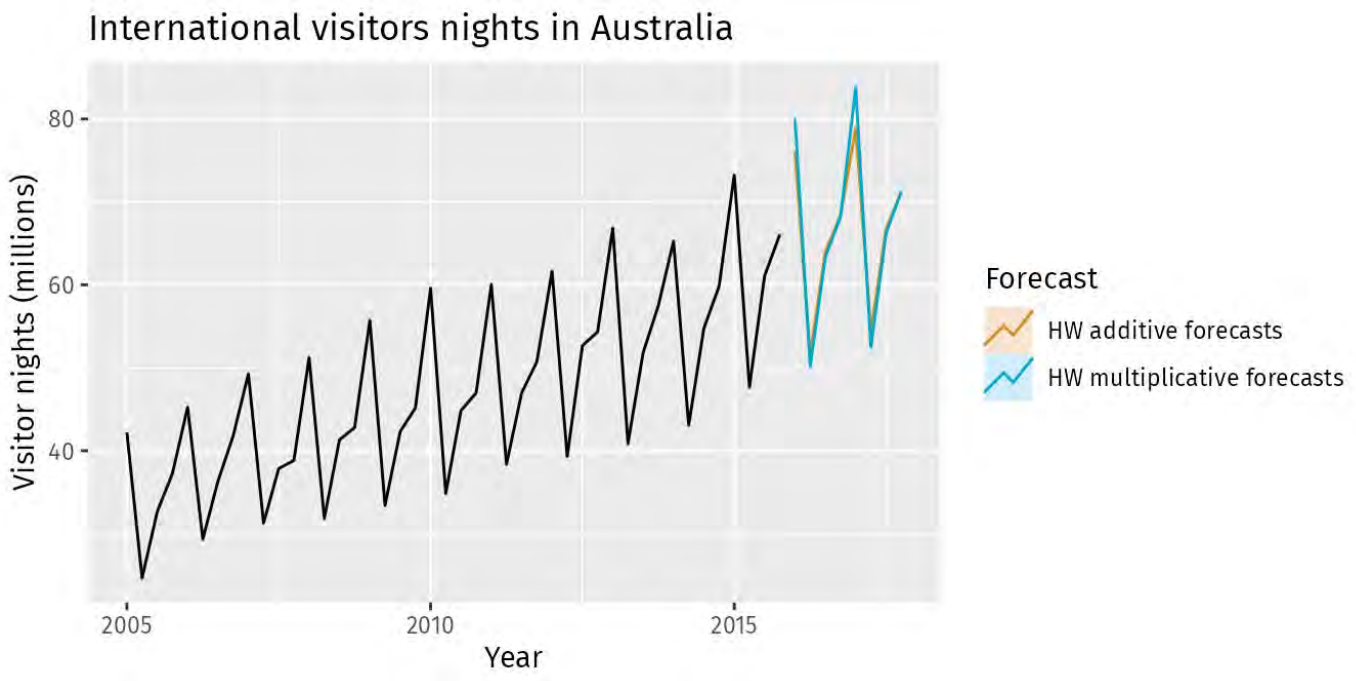


Figure 7.6: Forecasting international visitor nights in Australia using the Holt-Winters method with both additive and multiplicative seasonality.

Table 7.3: Applying Holt-Winters' method with additive seasonality for forecasting international visitor nights in Australia. Notice that the additive seasonal component sums to approximately zero. The smoothing parameters and initial estimates for the components have been estimated by minimising RMSE ($\alpha = 0.306$, $\beta^* = 0.0003$, $\gamma = 0.426$ and RMSE$= 1.763$).

	t	y_t	ℓ_t	b_t	s_t	$\hat{y}_t$	
2004 Q1	-3				9.70		
2004 Q2	-2				-9.31		
2004 Q3	-1				-1.69		
2004 Q4	0		32.26	0.70	1.31		
2005 Q1	1	42.21	32.82	0.70	9.50		42.66
2005 Q2	2	24.65	33.66	0.70	-9.13		24.21
2005 Q3	3	32.67	34.36	0.70	-1.69		32.67
2005 Q4	4	37.26	35.33	0.70	1.69		36.37
	⋮	⋮	⋮	⋮	⋮		⋮
2015 Q1	41	73.26	59.96	0.70	12.18		69.05
2015 Q2	42	47.70	60.69	0.70	-13.02		47.59
2015 Q3	43	61.10	61.96	0.70	-1.35		59.24
2015 Q4	44	66.06	63.22	0.70	2.35		64.22
	h						$\hat{y}_{T+h\|T}$
2016 Q1	1						76.10
2016 Q2	2						51.60
2016 Q3	3						63.97
2016 Q4	4						68.37
2017 Q1	5						78.90
2017 Q2	6						54.41
2017 Q3	7						66.77
2017 Q4	8						71.18

Table 7.4: Applying Holt-Winters' method with multiplicative seasonality for forecasting international visitor nights in Australia. Notice that the multiplicative seasonal component sums to approximately $m = 4$. The smoothing parameters and initial estimates for the components have been estimated by minimising RMSE ($\alpha = 0.441$, $\beta^* = 0.030$, $\gamma = 0.002$ and RMSE$= 1.576$).

	t	y_t	ℓ_t	b_t	s_t	$\hat{y}_t$	
2004 Q1	-3				1.24		
2004 Q2	-2				0.77		
2004 Q3	-1				0.96		
2004 Q4	0		32.49	0.70	1.02		
2005 Q1	1	42.21	33.51	0.71	1.24		41.29
2005 Q2	2	24.65	33.24	0.68	0.77		26.36
2005 Q3	3	32.67	33.94	0.68	0.96		32.62
2005 Q4	4	37.26	35.40	0.70	1.02		35.44
	⋮	⋮	⋮	⋮	⋮		⋮
2015 Q1	41	73.26	58.57	0.66	1.24		72.59
2015 Q2	42	47.70	60.42	0.69	0.77		45.62
2015 Q3	43	61.10	62.17	0.72	0.96		58.77
2015 Q4	44	66.06	63.62	0.75	1.02		64.38
	h						$\hat{y}_{T+h\|T}$
2016 Q1	1						80.09
2016 Q2	2						50.15
2016 Q3	3						63.34
2016 Q4	4						68.18
2017 Q1	5						83.80
2017 Q2	6						52.45
2017 Q3	7						66.21
2017 Q4	8						71.23

The applications of both methods (with additive and multiplicative seasonality) are presented in Tables 7.3 and 7.4 respectively. Because both methods have exactly the same number of parameters to estimate, we can compare the training RMSE from both models. In this case, the method with multiplicative seasonality fits the data best. This was to be expected, as the time plot shows that the seasonal variation in the data increases as the level of the series increases. This is also reflected in the two sets of forecasts; the forecasts generated by the method with the multiplicative seasonality display larger and increasing seasonal variation as the level of the forecasts increases compared to the forecasts generated by the method with additive seasonality.

The estimated states for both models are plotted in Figure 7.7. The small value of γ for the multiplicative model means that the seasonal component hardly changes over time. The small value of β^* for the additive model means the slope component hardly changes over time (check the vertical scale). The increasing size of the seasonal component for the additive model suggests that the model is less appropriate than the multiplicative model.

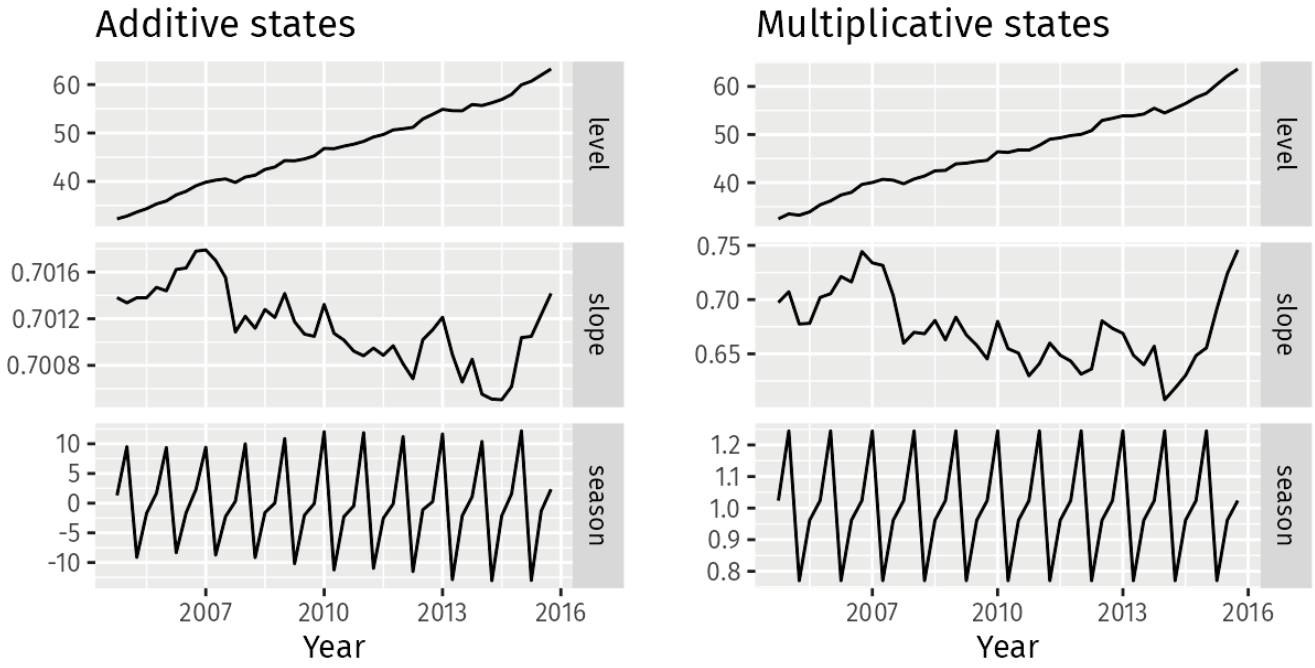


Figure 7.7: Estimated components for the Holt-Winters method with additive and multiplicative seasonal components.

Holt-Winters' damped method

Damping is possible with both additive and multiplicative Holt-Winters' methods. A method that often provides accurate and robust forecasts for seasonal data is the Holt-Winters method with a damped trend and multiplicative seasonality:

$$\begin{aligned}
\hat{y}_{t+h|t} &= \left[\ell_t + (\phi + \phi^2 + \cdots + \phi^h)b_t\right] s_{t+h-m(k+1)}. \\
\ell_t &= \alpha(y_t / s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1} \\
s_t &= \gamma \frac{y_t}{(\ell_{t-1} + \phi b_{t-1})} + (1-\gamma)s_{t-m}.
\end{aligned}$$

```
hw(y, damped=TRUE, seasonal="multiplicative")
```

Example: Holt-Winters method with daily data

The Holt-Winters method can also be used for daily type of data, where the seasonal period is $m = 7$, and the appropriate unit of time for h is in days. Here, we generate daily forecasts for the last five weeks for the `hyndsight` data, which contains the daily pageviews on the Hyndsight blog for one year starting April 30, 2014.

```
fc <- hw(subset(hyndsight,end=length(hyndsight)-35),
         damped = TRUE, seasonal="multiplicative", h=35)
autoplot(hyndsight) +
  autolayer(fc, series="HW multi damped", PI=FALSE)+
  guides(colour=guide_legend(title="Daily forecasts"))
```

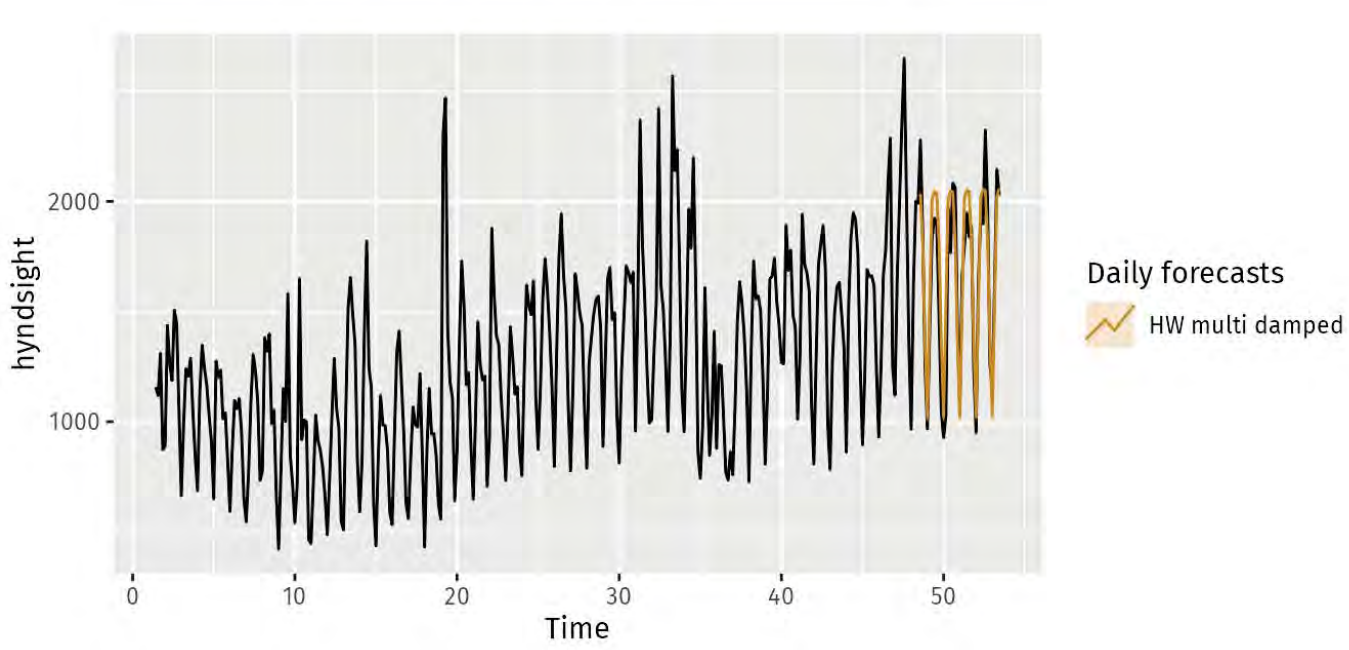


Figure 7.8: Forecasts of daily pageviews on the Hyndsight blog.

Clearly the model has identified the weekly seasonal pattern and the increasing trend at the end of the data, and the forecasts are a close match to the test data.

Bibliography

Holt, C. C. (1957). *Forecasting seasonals and trends by exponentially weighted averages* (O.N.R. Memorandum No. 52). Carnegie Institute of Technology, Pittsburgh USA. [DOI]

Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management Science*, *6*(3), 324–342. [DOI]

7.4 A taxonomy of exponential smoothing methods

Exponential smoothing methods are not restricted to those we have presented so far. By considering variations in the combinations of the trend and seasonal components, nine exponential smoothing methods are possible, listed in Table 7.5. Each method is labelled by a pair of letters (T,S) defining the type of 'Trend' and 'Seasonal' components. For example, (A,M) is the method with an additive trend and multiplicative seasonality; (A_d,N) is the method with damped trend and no seasonality; and so on.

Table 7.5: A two-way classification of exponential smoothing methods.

Trend Component	**Seasonal Component**		
	N	**A**	**M**
	(None)	**(Additive)**	**(Multiplicative)**
N (None)	(N,N)	(N,A)	(N,M)
A (Additive)	(A,N)	(A,A)	(A,M)
A_d (Additive damped)	(A_d,N)	(A_d,A)	(A_d,M)

Some of these methods we have already seen using other names:

Short hand	**Method**
(N,N)	Simple exponential smoothing
(A,N)	Holt's linear method
(A_d,N)	Additive damped trend method
(A,A)	Additive Holt-Winters' method
(A,M)	Multiplicative Holt-Winters' method
(A_d,M)	Holt-Winters' damped method

This type of classification was first proposed by Pegels (1969), who also included a method with a multiplicative trend. It was later extended by Gardner (1985) to include methods with an additive damped trend and by Taylor (2003) to include methods with a multiplicative damped trend. We do not consider the multiplicative trend methods in this book as they tend to produce poor forecasts. See Hyndman, Koehler, Ord, & Snyder (2008) for a more thorough discussion of all exponential smoothing methods.

Table 7.6 gives the recursive formulas for applying the nine exponential smoothing methods in Table 7.5. Each cell includes the forecast equation for generating h-step-ahead forecasts, and the smoothing equations for applying the method.

Table 7.6: Formulas for recursive calculations and point forecasts. In each case, ℓ_t denotes the series level at time t, b_t denotes the slope at time t, s_t denotes the seasonal component of the series at time t, and m denotes the number of seasons in a year; α, β^*, γ and ϕ are smoothing parameters, $\phi_h = \phi + \phi^2 + \cdots + \phi^h$, and k is the integer part of $(h-1)/m$.

Trend	Seasonal		
	N	**A**	**M**
N	$\hat{y}_{t+h\|t} = \ell_t$ $\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1}) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = \ell_t s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t/\ell_{t-1}) + (1-\gamma)s_{t-m}$
A	$\hat{y}_{t+h\|t} = \ell_t + hb_t$ $\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + hb_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = (\ell_t + hb_t)s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$ $s_t = \gamma(y_t/(\ell_{t-1} + b_{t-1})) + (1-\gamma)s_{t-m}$
A_d	$\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t$ $\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + \phi_h b_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $s_t = \gamma(y_t - \ell_{t-1} - \phi b_{t-1}) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = (\ell_t + \phi_h b_t)s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1}$ $s_t = \gamma(y_t/(\ell_{t-1} + \phi b_{t-1})) + (1-\gamma)s_{t-m}$

Bibliography

Gardner, E. S. (1985). Exponential smoothing: The state of the art. *Journal of Forecasting*, *4*(1), 1–28. [DOI]

Hyndman, R. J., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). *Forecasting with exponential smoothing: The state space approach*. Berlin: Springer-Verlag. http://www.exponentialsmoothing.net

Pegels, C. C. (1969). Exponential forecasting: Some new variations. *Management Science*, *15*(5), 311–315. [DOI]

Taylor, J. W. (2003). Exponential smoothing with a damped multiplicative trend. *International Journal of Forecasting*, *19*(4), 715–725. [DOI]

7.5 Innovations state space models for exponential smoothing

In the rest of this chapter, we study the statistical models that underlie the exponential smoothing methods we have considered so far. The exponential smoothing methods presented in Table 7.6 are algorithms which generate point forecasts. The statistical models in this section generate the same point forecasts, but can also generate prediction (or forecast) intervals. A statistical model is a stochastic (or random) data generating process that can produce an entire forecast distribution. We will also describe how to use the model selection criteria introduced in Chapter 5 to choose the model in an objective manner.

Each model consists of a measurement equation that describes the observed data, and some state equations that describe how the unobserved components or states (level, trend, seasonal) change over time. Hence, these are referred to as **state space models**.

For each method there exist two models: one with additive errors and one with multiplicative errors. The point forecasts produced by the models are identical if they use the same smoothing parameter values. They will, however, generate different prediction intervals.

To distinguish between a model with additive errors and one with multiplicative errors (and also to distinguish the models from the methods), we add a third letter to the classification of Table 7.5. We label each state space model as ETS($\cdot$, $\cdot$, $\cdot$) for (Error, Trend, Seasonal). This label can also be thought of as ExponenTial Smoothing. Using the same notation as in Table 7.5, the possibilities for each component are: Error $=$ $\{$A,M$\}$, Trend $=$ $\{$N,A,A$_d$$\}$ and Seasonal $=$ $\{$N,A,M$\}$.

ETS(A,N,N): simple exponential smoothing with additive errors

Recall the component form of simple exponential smoothing:

$$\begin{aligned} \text{Forecast equation} \quad & \hat{y}_{t+1|t} = \ell_t \\ \text{Smoothing equation} \quad & \ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}, \end{aligned}$$

If we re-arrange the smoothing equation for the level, we get the "error correction" form:

$$\begin{aligned} \ell_t &= \ell_{t-1} + \alpha(y_t - \ell_{t-1}) \\ &= \ell_{t-1} + \alpha e_t \end{aligned}$$

where $e_t = y_t - \ell_{t-1} = y_t - \hat{y}_{t|t-1}$ is the residual at time t.

The training data errors lead to the adjustment of the estimated level throughout the smoothing process for $t = 1, \dots, T$. For example, if the error at time t is negative, then $y_t < \hat{y}_{t|t-1}$ and so the level at time $t-1$ has been over-estimated. The new level ℓ_t is then the previous level ℓ_{t-1} adjusted downwards. The closer α is to one, the "rougher" the estimate of the level (large adjustments take place). The smaller the α , the "smoother" the level (small adjustments take place).

We can also write $y_t = \ell_{t-1} + e_t$, so that each observation can be represented by the previous level plus an error. To make this into an innovations state space model, all we need to do is specify the probability distribution for e_t. For a model with additive errors, we assume that residuals (the one-step training errors) e_t are normally distributed white noise with mean 0 and variance σ^2. A short-hand notation for this is $e_t = \varepsilon_t \sim \text{NID}(0, \sigma^2)$; NID stands for "normally and independently distributed".

Then the equations of the model can be written as

$$y_t = \ell_{t-1} + \varepsilon_t \tag{7.3}$$
$$\ell_t = \ell_{t-1} + \alpha\varepsilon_t. \tag{7.4}$$

We refer to (7.3) as the *measurement* (or observation) equation and (7.4) as the *state* (or transition) equation. These two equations, together with the statistical distribution of the errors, form a fully specified statistical model. Specifically, these constitute an innovations state space model underlying simple exponential smoothing.

The term "innovations" comes from the fact that all equations use the same random error process, ε_t. For the same reason, this formulation is also referred to as a "single source of error" model. There are alternative multiple source of error formulations which we do not present here.

The measurement equation shows the relationship between the observations and the unobserved states. In this case, observation y_t is a linear function of the level ℓ_{t-1}, the predictable part of y_t, and the error ε_t, the unpredictable part of y_t. For other innovations state space models, this relationship may be nonlinear.

The state equation shows the evolution of the state through time. The influence of the smoothing parameter α is the same as for the methods discussed earlier. For example, α governs the amount of change in successive levels: high values of α allow rapid changes in the level; low values of α lead to smooth changes. If $\alpha = 0$, the level of the series does not change over time; if $\alpha = 1$, the model reduces to a random walk model, $y_t = y_{t-1} + \varepsilon_t$. (See Section 8.1 for a discussion of this model.)

ETS(M,N,N): simple exponential smoothing with multiplicative errors

In a similar fashion, we can specify models with multiplicative errors by writing the one-step-ahead training errors as relative errors:

$$\varepsilon_t = \frac{y_t - \hat{y}_{t|t-1}}{\hat{y}_{t|t-1}}$$

where $\varepsilon_t \sim \text{NID}(0, \sigma^2)$. Substituting $\hat{y}_{t|t-1} = \ell_{t-1}$ gives $y_t = \ell_{t-1} + \ell_{t-1}\varepsilon_t$ and $e_t = y_t - \hat{y}_{t|t-1} = \ell_{t-1}\varepsilon_t$.

Then we can write the multiplicative form of the state space model as

$$\begin{aligned} y_t &= \ell_{t-1}(1 + \varepsilon_t) \\ \ell_t &= \ell_{t-1}(1 + \alpha\varepsilon_t). \end{aligned}$$

ETS(A,A,N): Holt's linear method with additive errors

For this model, we assume that the one-step-ahead training errors are given by $\varepsilon_t = y_t - \ell_{t-1} - b_{t-1} \sim \text{NID}(0, \sigma^2)$. Substituting this into the error correction equations for Holt's linear method we obtain

$$
\begin{aligned}
y_t &= \ell_{t-1} + b_{t-1} + \varepsilon_t \\
\ell_t &= \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t \\
b_t &= b_{t-1} + \beta\varepsilon_t,
\end{aligned}
$$

where, for simplicity, we have set $\beta = \alpha\beta^*$.

ETS(M,A,N): Holt's linear method with multiplicative errors

Specifying one-step-ahead training errors as relative errors such that

$$
\varepsilon_t = \frac{y_t - (\ell_{t-1} + b_{t-1})}{(\ell_{t-1} + b_{t-1})}
$$

and following an approach similar to that used above, the innovations state space model underlying Holt's linear method with multiplicative errors is specified as

$$
\begin{aligned}
y_t &= (\ell_{t-1} + b_{t-1})(1 + \varepsilon_t) \\
\ell_t &= (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t) \\
b_t &= b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t
\end{aligned}
$$

where again $\beta = \alpha\beta^*$ and $\varepsilon_t \sim \text{NID}(0, \sigma^2)$.

Other ETS models

In a similar fashion, we can write an innovations state space model for each of the exponential smoothing methods of Table 7.6. Table 7.7 presents the equations for all of the models in the ETS framework.

Table 7.7: State space equations for each of the models in the ETS framework.

ADDITIVE ERROR MODELS

Trend	Seasonal: N	Seasonal: A	Seasonal: M
N	$y_t = \ell_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$	$y_t = \ell_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = \ell_{t-1}s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/\ell_{t-1}$
A	$y_t = \ell_{t-1} + b_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$	$y_t = \ell_{t-1} + b_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$ $b_t = b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1})s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + b_{t-1})$
A_d	$y_t = \ell_{t-1} + \phi b_{t-1} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$	$y_t = \ell_{t-1} + \phi b_{t-1} + s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$ $b_t = \phi b_{t-1} + \beta\varepsilon_t$ $s_t = s_{t-m} + \gamma\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m} + \varepsilon_t$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t/s_{t-m}$ $b_t = \phi b_{t-1} + \beta\varepsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\varepsilon_t/(\ell_{t-1} + \phi b_{t-1})$

MULTIPLICATIVE ERROR MODELS

Trend	Seasonal: N	Seasonal: A	Seasonal: M
N	$y_t = \ell_{t-1}(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$	$y_t = (\ell_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + \alpha(\ell_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + s_{t-m})\varepsilon_t$	$y_t = \ell_{t-1}s_{t-m}(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1}(1 + \alpha\varepsilon_t)$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A	$y_t = (\ell_{t-1} + b_{t-1})(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + b_{t-1} + \alpha(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (\ell_{t-1} + b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$
A_d	$y_t = (\ell_{t-1} + \phi b_{t-1})(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $\ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(\ell_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (\ell_{t-1} + \phi b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $\ell_t = (\ell_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(\ell_{t-1} + \phi b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$

7.6 Estimation and model selection

Estimating ETS models

An alternative to estimating the parameters by minimising the sum of squared errors is to maximise the "likelihood". The likelihood is the probability of the data arising from the specified model. Thus, a large likelihood is associated with a good model. For an additive error model, maximising the likelihood (assuming normally distributed errors) gives the same results as minimising the sum of squared errors. However, different results will be obtained for multiplicative error models. In this section, we will estimate the smoothing parameters α, β, γ and ϕ, and the initial states ℓ_0, b_0, $s_0, s_{-1}, \dots, s_{-m+1}$, by maximising the likelihood.

The possible values that the smoothing parameters can take are restricted. Traditionally, the parameters have been constrained to lie between 0 and 1 so that the equations can be interpreted as weighted averages. That is, $0 < \alpha, \beta^*, \gamma^*, \phi < 1$. For the state space models, we have set $\beta = \alpha\beta^*$ and $\gamma = (1-\alpha)\gamma^*$. Therefore, the traditional restrictions translate to $0 < \alpha < 1$, $0 < \beta < \alpha$ and $0 < \gamma < 1-\alpha$. In practice, the damping parameter ϕ is usually constrained further to prevent numerical difficulties in estimating the model. In R, it is restricted so that $0.8 < \phi < 0.98$.

Another way to view the parameters is through a consideration of the mathematical properties of the state space models. The parameters are constrained in order to prevent observations in the distant past having a continuing effect on current forecasts. This leads to some *admissibility* constraints on the parameters, which are usually (but not always) less restrictive than the traditional constraints region (Hyndman et al., 2008, p. Ch10). For example, for the ETS(A,N,N) model, the traditional parameter region is $0 < \alpha < 1$ but the admissible region is $0 < \alpha < 2$. For the ETS(A,A,N) model, the traditional parameter region is $0 < \alpha < 1$ and $0 < \beta < \alpha$ but the admissible region is $0 < \alpha < 2$ and $0 < \beta < 4 - 2\alpha$.

Model selection

A great advantage of the ETS statistical framework is that information criteria can be used for model selection. The AIC, AIC_c and BIC, introduced in Section 5.5, can be used here to determine which of the ETS models is most appropriate for a given time series.

For ETS models, Akaike's Information Criterion (AIC) is defined as

$$\text{AIC} = -2\log(L) + 2k,$$

where L is the likelihood of the model and k is the total number of parameters and initial states that have been estimated (including the residual variance).

The AIC corrected for small sample bias (AIC_c) is defined as

$$\text{AIC}_\text{c} = \text{AIC} + \frac{2k(k+1)}{T-k-1},$$

and the Bayesian Information Criterion (BIC) is

$$\text{BIC} = \text{AIC} + k[\log(T) - 2].$$

Three of the combinations of (Error, Trend, Seasonal) can lead to numerical difficulties. Specifically, the models that can cause such instabilities are ETS(A,N,M), ETS(A,A,M), and ETS(A,A_d,M), due to division by values potentially close to zero in the state equations. We normally do not consider these particular combinations when selecting a model.

Models with multiplicative errors are useful when the data are strictly positive, but are not numerically stable when the data contain zeros or negative values. Therefore, multiplicative error models will not be considered if the time series is not strictly positive. In that case, only the six fully additive models will be applied.

The `ets()` function in R

The models can be estimated in R using the `ets()` function in the **forecast** package. Unlike the `ses()`, `holt()` and `hw()` functions, the `ets()` function does not produce forecasts. Rather, it estimates the model parameters and returns information about the fitted model. By default it uses the AICc to select an appropriate model, although other information criteria can be selected.

The R code below shows the most important arguments that this function can take, and their default values. If only the time series is specified, and all other arguments are left at their default values, then an appropriate model will be selected automatically. We explain the arguments below. See the help file for a complete description.

```
ets(y, model="ZZZ", damped=NULL, alpha=NULL, beta=NULL,
    gamma=NULL, phi=NULL, lambda=NULL, biasadj=FALSE,
    additive.only=FALSE, restrict=TRUE,
    allow.multiplicative.trend=FALSE)
```

y

The time series to be forecast.

model

A three-letter code indicating the model to be estimated using the ETS classification and notation. The possible inputs are "N" for none, "A" for additive, "M" for multiplicative, or "Z" for automatic selection. If any of the inputs is left as "Z", then this component is selected according to the information criterion. The default value of `ZZZ` ensures that all components are selected using the information criterion.

damped

If `damped=TRUE`, then a damped trend will be used (either A or M). If `damped=FALSE`, then a non-damped trend will used. If `damped=NULL` (the default), then either a damped or a non-damped trend will be selected, depending on which model has the smallest value for the information criterion.

alpha, beta, gamma, phi

The values of the smoothing parameters can be specified using these arguments. If they are set to `NULL` (the default setting for each of them), the parameters are estimated.

lambda

Box-Cox transformation parameter. It will be ignored if `lambda=NULL` (the default value). Otherwise, the time series will be transformed before the model is estimated. When `lambda` is not `NULL`, `additive.only` is set to `TRUE`.

biasadj

If `TRUE` and `lambda` is not `NULL`, then the back-transformed fitted values and forecasts will be bias-adjusted.

additive.only

Only models with additive components will be considered if `additive.only=TRUE`. Otherwise, all models will be considered.

restrict

If `restrict=TRUE` (the default), the models that cause numerical difficulties are not considered in model selection.

allow.multiplicative.trend

Multiplicative trend models are also available, but not covered in this book. Set this argument to `TRUE` to allow these models to be considered.

Working with `ets` objects

The `ets()` function will return an object of class `ets`. There are many R functions designed to make working with `ets` objects easy. A few of them are described below.

coef()

returns all fitted parameters.

accuracy()

returns accuracy measures computed on the training data.

summary()

prints some summary information about the fitted model.

autoplot() and plot()

produce time plots of the components.

residuals()

returns residuals from the estimated model.

fitted()

returns one-step forecasts for the training data.

simulate()

will simulate future sample paths from the fitted model.

forecast()

computes point forecasts and prediction intervals, as described in the next section.

Example: International tourist visitor nights in Australia

We now employ the ETS statistical framework to forecast tourist visitor nights in Australia by international arrivals over the period 2016–2019. We let the `ets()` function select the model by minimising the AICc.

```
aust <- window(austourists, start=2005)
fit <- ets(aust)
summary(fit)
#> ETS(M,A,M)
#>
#> Call:
#>  ets(y = aust)
#>
#>   Smoothing parameters:
#>     alpha = 0.1908
#>     beta  = 0.0392
#>     gamma = 2e-04
#>
#>   Initial states:
#>     l = 32.3679
#>     b = 0.9281
#>     s = 1.022 0.9628 0.7683 1.247
#>
#>   sigma:  0.0383
#>
#>   AIC  AICc   BIC
#> 224.9 230.2 240.9
#>
#> Training set error measures:
#>                     ME  RMSE  MAE     MPE  MAPE   MASE   ACF1
#> Training set 0.04837 1.671 1.25 -0.1846 2.693 0.4095 0.2006
```

The model selected is ETS(M,A,M):

$$
\begin{aligned}
y_t &= (\ell_{t-1} + b_{t-1})s_{t-m}(1 + \varepsilon_t) \\
\ell_t &= (\ell_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t) \\
b_t &= b_{t-1} + \beta(\ell_{t-1} + b_{t-1})\varepsilon_t \\
s_t &= s_{t-m}(1 + \gamma\varepsilon_t).
\end{aligned}
$$

The parameter estimates are $\hat{\alpha} = 0.1908$, $\hat{\beta} = 0.0392$, and $\hat{\gamma} = 0.0002$. The output also returns the estimates for the initial states ℓ_0, b_0, s_0, s_{-1}, s_{-2} and s_{-3}. Compare these with the values obtained for the equivalent Holt-Winters method with multiplicative seasonality presented in Table 7.4. The ETS(M,A,M) model will give different point forecasts to the multiplicative Holt-Winters' method, because the parameters have been estimated differently. With the `ets()` function, the default estimation method is maximum likelihood rather than minimum sum of squares.

Figure 7.9 shows the states over time, while Figure 7.11 shows point forecasts and prediction intervals generated from the model. The small values of β and γ mean that the slope and seasonal components change very little over time. The narrow prediction intervals indicate that the series is relatively easy to forecast due to the strong trend and seasonality.

```
autoplot(fit)
```

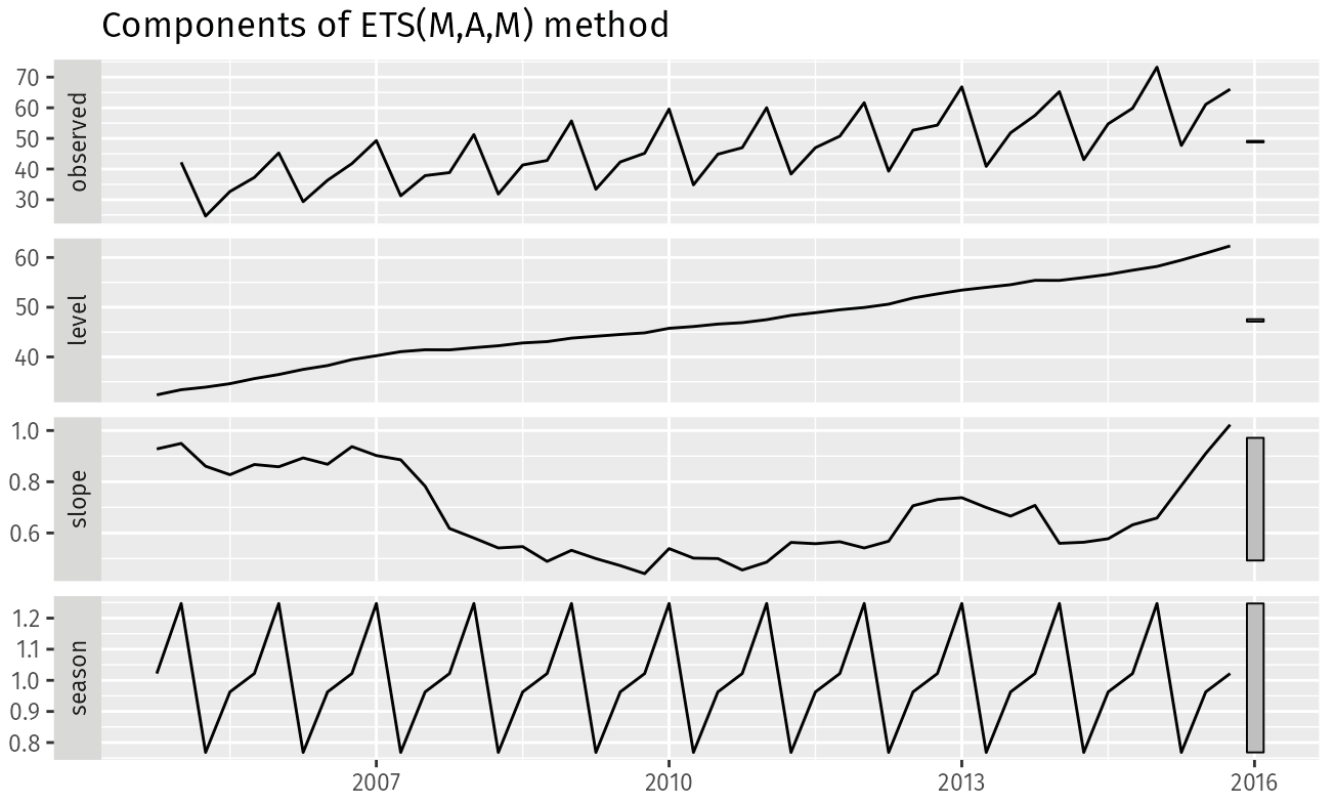


Figure 7.9: Graphical representation of the estimated states over time.

Because this model has multiplicative errors, the residuals are not equivalent to the one-step training errors. The residuals are given by $\hat{\varepsilon}_t$, while the one-step training errors are defined as $y_t - \hat{y}_{t|t-1}$. We can obtain both using the `residuals()` function.

```
cbind('Residuals' = residuals(fit),
      'Forecast errors' = residuals(fit,type='response')) %>%
  autoplot(facet=TRUE) + xlab("Year") + ylab("")
```

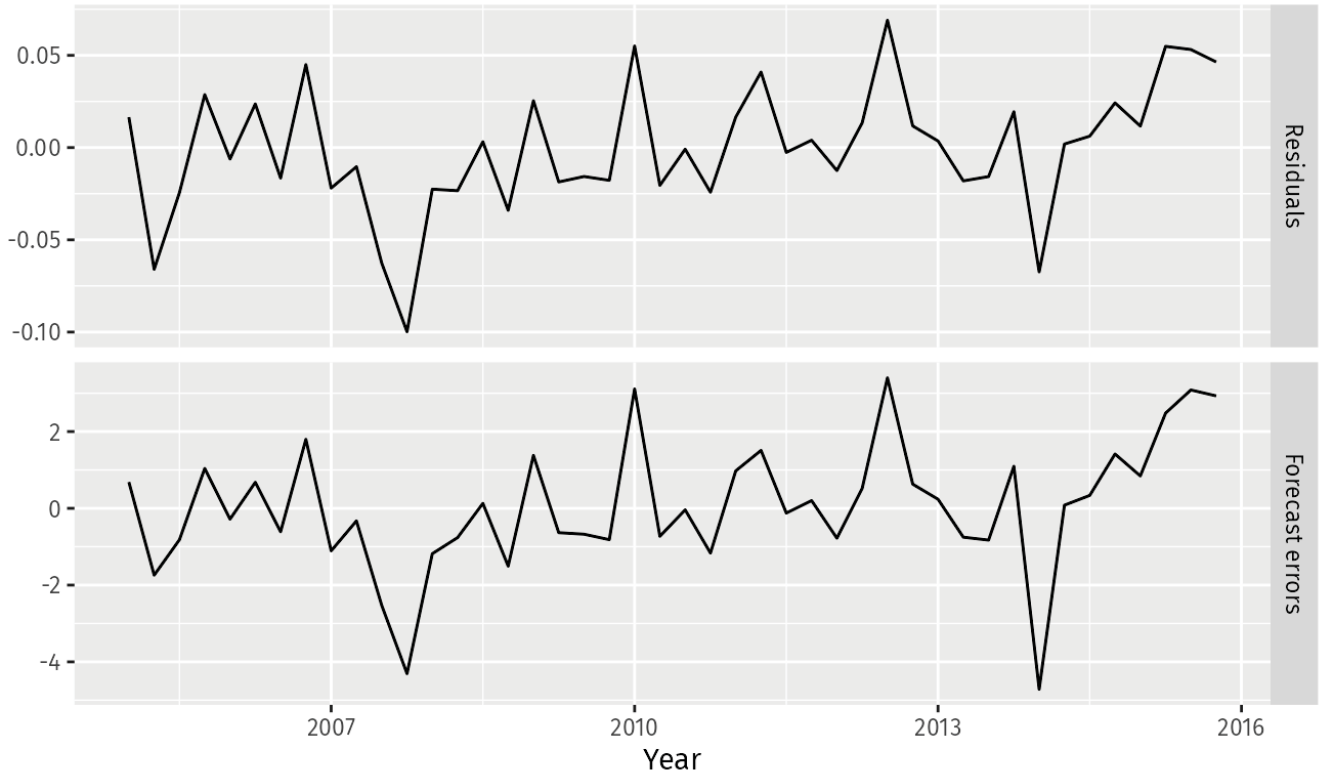


Figure 7.10: Residuals and one-step forecast errors from the ETS(M,A,M) model.

The `type` argument is used in the `residuals()` function to distinguish between residuals and forecast errors. The default is `type='innovation'` which gives regular residuals.

Bibliography

Hyndman, R. J., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). *Forecasting with exponential smoothing: The state space approach*. Berlin: Springer-Verlag. http://www.exponentialsmoothing.net

7.7 Forecasting with ETS models

Point forecasts are obtained from the models by iterating the equations for $t = T+1, \dots, T+h$ and setting all $\varepsilon_t = 0$ for $t > T$.

For example, for model ETS(M,A,N), $y_{T+1} = (\ell_T + b_T)(1 + \varepsilon_{T+1})$. Therefore $\hat{y}_{T+1|T} = \ell_T + b_T$. Similarly,

$$\begin{aligned} y_{T+2} &= (\ell_{T+1} + b_{T+1})(1 + \varepsilon_{T+2}) \\ &= \left[(\ell_T + b_T)(1 + \alpha\varepsilon_{T+1}) + b_T + \beta(\ell_T + b_T)\varepsilon_{T+1}\right](1 + \varepsilon_{T+2}). \end{aligned}$$

Therefore, $\hat{y}_{T+2|T} = \ell_T + 2b_T$, and so on. These forecasts are identical to the forecasts from Holt's linear method, and also to those from model ETS(A,A,N). Thus, the point forecasts obtained from the method and from the two models that underlie the method are identical (assuming that the same parameter values are used).

ETS point forecasts are equal to the medians of the forecast distributions. For models with only additive components, the forecast distributions are normal, so the medians and means are equal. For ETS models with multiplicative errors, or with multiplicative seasonality, the point forecasts will not be equal to the means of the forecast distributions.

To obtain forecasts from an ETS model, we use the `forecast()` function.

```
fit %>% forecast(h=8) %>%
  autoplot() +
  ylab("International visitor night in Australia (millions)")
```

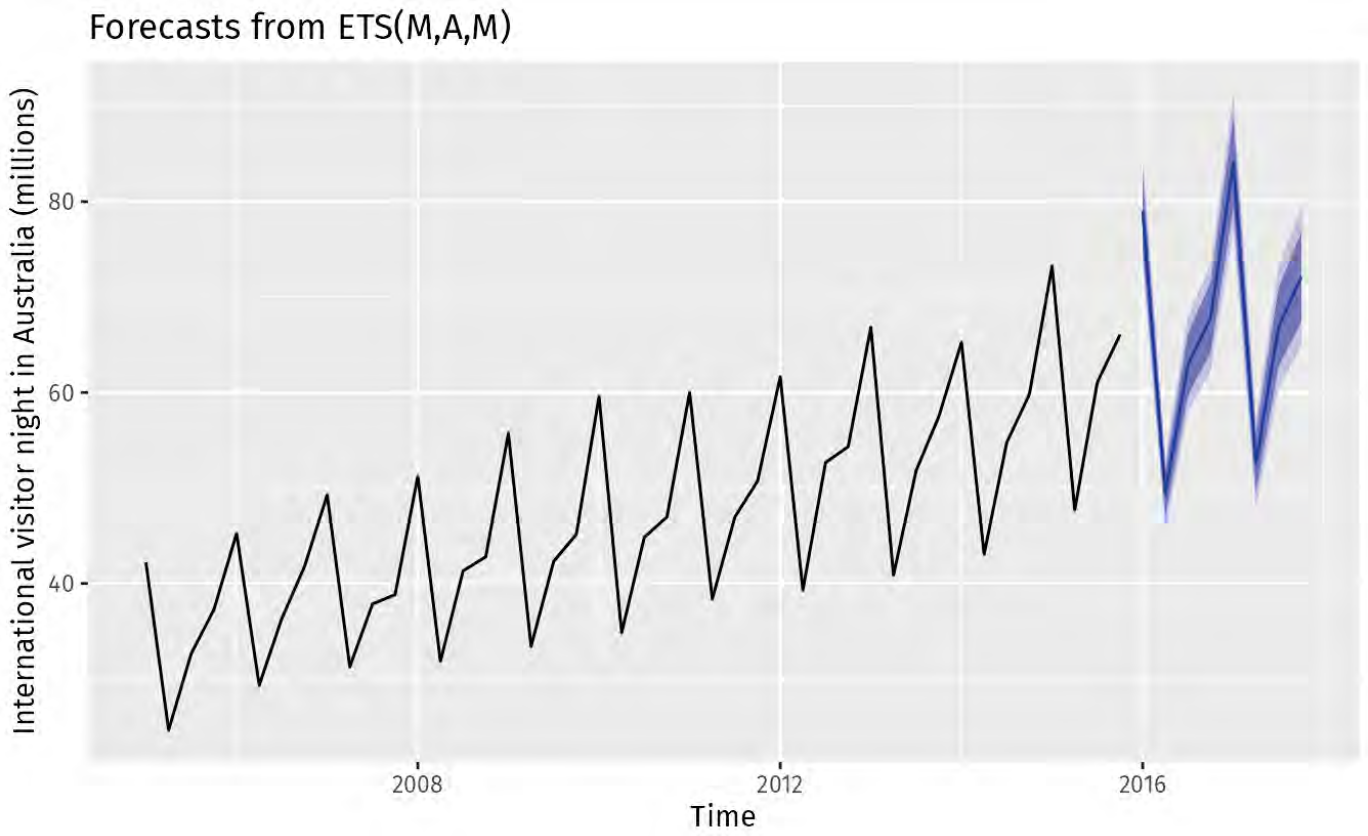


Figure 7.11: Forecasting international visitor nights in Australia using an ETS(M,A,M) model.

Prediction intervals

A big advantage of the models is that prediction intervals can also be generated — something that cannot be done using the methods. The prediction intervals will differ between models with additive and multiplicative methods.

For most ETS models, a prediction interval can be written as

$$\hat{y}_{T+h|T} \pm c\sigma_h$$

where c depends on the coverage probability, and σ_h^2 is the forecast variance. Values for c were given in Table 3.1. For ETS models, formulas for σ_h^2 can be complicated; the details are given in Chapter 6 of Hyndman et al. (2008). In Table 7.8 we give the formulas for the additive ETS models, which are the simplest.

Table 7.8: Forecast variance expressions for each additive state space model, where σ^2 is the residual variance, m is the seasonal period, and k is the integer part of $(h-1)/m$ (i.e., the number of complete years in the forecast period prior to time $T+h$).

Model	Forecast variance: σ_h^2
(A,N,N)	$\sigma_h^2 = \sigma^2\big[1+\alpha^2(h-1)\big]$
(A,A,N)	$\sigma_h^2 = \sigma^2\Big[1+(h-1)\big\{\alpha^2+\alpha\beta h+\frac{1}{6}\beta^2 h(2h-1)\big\}\Big]$
(A,A$_d$,N)	$\sigma_h^2 = \sigma^2\Big[1+\alpha^2(h-1)+\frac{\beta\phi h}{(1-\phi)^2}\{2\alpha(1-\phi)+\beta\phi\}$ $-\frac{\beta\phi(1-\phi^h)}{(1-\phi)^2(1-\phi^2)}\big\{2\alpha(1-\phi^2)+\beta\phi(1+2\phi-\phi^h)\big\}\Big]$
(A,N,A)	$\sigma_h^2 = \sigma^2\Big[1+\alpha^2(h-1)+\gamma k(2\alpha+\gamma)\Big]$
(A,A,A)	$\sigma_h^2 = \sigma^2\Big[1+(h-1)\big\{\alpha^2+\alpha\beta h+\frac{1}{6}\beta^2 h(2h-1)\big\}$ $+\gamma k\big\{2\alpha+\gamma+\beta m(k+1)\big\}\Big]$
(A,A$_d$,A)	$\sigma_h^2 = \sigma^2\Big[1+\alpha^2(h-1)+\gamma k(2\alpha+\gamma)$ $+\frac{\beta\phi h}{(1-\phi)^2}\{2\alpha(1-\phi)+\beta\phi\}$ $-\frac{\beta\phi(1-\phi^h)}{(1-\phi)^2(1-\phi^2)}\big\{2\alpha(1-\phi^2)+\beta\phi(1+2\phi-\phi^h)\big\}$ $+\frac{2\beta\gamma\phi}{(1-\phi)(1-\phi^m)}\big\{k(1-\phi^m)-\phi^m(1-\phi^{mk})\big\}\Big]$

For a few ETS models, there are no known formulas for prediction intervals. In these cases, the `forecast()` function uses simulated future sample paths and computes prediction intervals from the percentiles of these simulated future paths.

Using `forecast()`

The R code below shows the possible arguments that this function takes when applied to an ETS model. We explain each of the arguments in what follows.

```
forecast(object, h=ifelse(object$m>1, 2*object$m, 10),
level=c(80,95), fan=FALSE, simulate=FALSE, bootstrap=FALSE,
npaths=5000, PI=TRUE, lambda=object$lambda, biasadj=NULL, ...)
```

object
The object returned by the `ets()` function.

h

The forecast horizon — the number of periods to be forecast.

level

The confidence level for the prediction intervals.

fan

If `fan=TRUE`, `level=seq(50,99,by=1)`. This is suitable for fan plots.

simulate

If `simulate=TRUE`, prediction intervals are produced by simulation rather than using algebraic formulas. Simulation will also be used (even if `simulate=FALSE`) where there are no algebraic formulas available for the particular model.

bootstrap

If `bootstrap=TRUE` and `simulate=TRUE`, then the simulated prediction intervals use re-sampled errors rather than normally distributed errors.

npaths

The number of sample paths used in computing simulated prediction intervals.

PI

If `PI=TRUE`, then prediction intervals are produced; otherwise only point forecasts are calculated.

lambda

The Box-Cox transformation parameter. This is ignored if `lambda=NULL`. Otherwise, the forecasts are back-transformed via an inverse Box-Cox transformation.

biasadj

If `lambda` is not `NULL`, the back-transformed forecasts (and prediction intervals) are bias-adjusted.

Bibliography

Hyndman, R. J., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). *Forecasting with exponential smoothing: The state space approach*. Berlin: Springer-Verlag. http://www.exponentialsmoothing.net

7.8 Exercises

1. Consider the `pigs` series — the number of pigs slaughtered in Victoria each month.

    a. Use the `ses()` function in R to find the optimal values of α and ℓ_0, and generate forecasts for the next four months.
    b. Compute a 95% prediction interval for the first forecast using $\hat{y} \pm 1.96s$ where s is the standard deviation of the residuals. Compare your interval with the interval produced by R.
2. Write your own function to implement simple exponential smoothing. The function should take arguments `y` (the time series), `alpha` (the smoothing parameter α) and `level` (the initial level ℓ_0). It should return the forecast of the next observation in the series. Does it give the same forecast as `ses()` ?

3. Modify your function from the previous exercise to return the sum of squared errors rather than the forecast of the next observation. Then use the `optim()` function to find the optimal values of α and ℓ_0. Do you get the same values as the `ses()` function?

4. Combine your previous two functions to produce a function which both finds the optimal values of α and ℓ_0, and produces a forecast of the next observation in the series.

5. Data set `books` contains the daily sales of paperback and hardcover books at the same store. The task is to forecast the next four days' sales for paperback and hardcover books.

    a. Plot the series and discuss the main features of the data.
    b. Use the `ses()` function to forecast each series, and plot the forecasts.
    c. Compute the RMSE values for the training data in each case.
6. We will continue with the daily sales of paperback and hardcover books in data set `books` .

    a. Apply Holt's linear method to the `paperback` and `hardback` series and compute four-day forecasts in each case.
    b. Compare the RMSE measures of Holt's method for the two series to those of simple exponential smoothing in the previous question. (Remember that

Holt's method is using one more parameter than SES.) Discuss the merits of the two forecasting methods for these data sets.

c. Compare the forecasts for the two series using both methods. Which do you think is best?

d. Calculate a 95% prediction interval for the first forecast for each series, using the RMSE values and assuming normal errors. Compare your intervals with those produced using `ses` and `holt`.

7. For this exercise use data set `eggs`, the price of a dozen eggs in the United States from 1900–1993. Experiment with the various options in the `holt()` function to see how much the forecasts change with damped trend, or with a Box-Cox transformation. Try to develop an intuition of what each argument is doing to the forecasts.

   [Hint: use `h=100` when calling `holt()` so you can clearly see the differences between the various options when plotting the forecasts.]

   Which model gives the best RMSE?

8. Recall your retail time series data (from Exercise 3 in Section 2.10).

   a. Why is multiplicative seasonality necessary for this series?

   b. Apply Holt-Winters' multiplicative method to the data. Experiment with making the trend damped.

   c. Compare the RMSE of the one-step forecasts from the two methods. Which do you prefer?

   d. Check that the residuals from the best method look like white noise.

   e. Now find the test set RMSE, while training the model to the end of 2010. Can you beat the seasonal naïve approach from Exercise 8 in Section 3.7?

9. For the same retail data, try an STL decomposition applied to the Box-Cox transformed series, followed by ETS on the seasonally adjusted data. How does that compare with your best previous forecasts on the test set?

10. For this exercise use data set `ukcars`, the quarterly UK passenger vehicle production data from 1977Q1–2005Q1.

    a. Plot the data and describe the main features of the series.

    b. Decompose the series using STL and obtain the seasonally adjusted data.

    c. Forecast the next two years of the series using an additive damped trend method applied to the seasonally adjusted data. (This can be done in one step using `stlf()` with arguments `etsmodel="AAN", damped=TRUE`.)

d. Forecast the next two years of the series using Holt's linear method applied to the seasonally adjusted data (as before but with `damped=FALSE`).
e. Now use `ets()` to choose a seasonal model for the data.
f. Compare the RMSE of the ETS model with the RMSE of the models you obtained using STL decompositions. Which gives the better in-sample fits?
g. Compare the forecasts from the three approaches? Which seems most reasonable?
h. Check the residuals of your preferred model.

11. For this exercise use data set `visitors`, the monthly Australian short-term overseas visitors data, May 1985–April 2005.

a. Make a time plot of your data and describe the main features of the series.
b. Split your data into a training set and a test set comprising the last two years of available data. Forecast the test set using Holt-Winters' multiplicative method.
c. Why is multiplicative seasonality necessary here?
d. Forecast the two-year test set using each of the following methods:
   i. an ETS model;
   ii. an additive ETS model applied to a Box-Cox transformed series;
   iii. a seasonal naïve method;
   iv. an STL decomposition applied to the Box-Cox transformed data followed by an ETS model applied to the seasonally adjusted (transformed) data.
e. Which method gives the best forecasts? Does it pass the residual tests?
f. Compare the same four methods using time series cross-validation with the `tsCV()` function instead of using a training and test set. Do you come to the same conclusions?

12. The `fets()` function below returns ETS forecasts.

```
fets <- function(y, h) {
  forecast(ets(y), h = h)
}
```

a. Apply `tsCV()` for a forecast horizon of $h = 4$, for both ETS and seasonal naïve methods to the `qcement` data, (Hint: use the newly created `fets()` and the existing `snaive()` functions as your forecast function arguments.)
b. Compute the MSE of the resulting 4-step-ahead errors. (Hint: make sure you remove missing values.) Why are there missing values? Comment on which forecasts are more accurate. Is this what you expected?

13. Compare `ets()`, `snaive()` and `stlf()` on the following six time series. For `stlf()`, you might need to use a Box-Cox transformation. Use a test set of three years to decide what gives the best forecasts. `ausbeer`, `bricksq`, `dole`, `a10`, `h02`, `usmelec`.

14. a. Use `ets()` on the following series: `bicoal`, `chicken`, `dole`, `usdeaths`, `lynx`, `ibmclose`, `eggs`. Does it always give good forecasts?
    b. Find an example where it does not work well. Can you figure out why?
15. Show that the point forecasts from an ETS(M,A,M) model are the same as those obtained using Holt-Winters' multiplicative method.

16. Show that the forecast variance for an ETS(A,N,N) model is given by

$$\sigma^2 \left[1 + \alpha^2(h-1)\right].$$

17. Write down 95% prediction intervals for an ETS(A,N,N) model as a function of ℓ_T, α, h and σ, assuming normally distributed errors.

7.9 Further reading

- Two articles by Ev Gardner (Gardner, 1985, 2006) provide a great overview of the history of exponential smoothing, and its many variations.
- A full book treatment of the subject providing the mathematical details is given by Hyndman et al. (2008).

Bibliography

Gardner, E. S. (1985). Exponential smoothing: The state of the art. *Journal of Forecasting*, *4*(1), 1–28. [DOI]

Gardner, E. S. (2006). Exponential smoothing: The state of the art — Part II. *International Journal of Forecasting*, *22*, 637–666. [DOI]

Hyndman, R. J., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). *Forecasting with exponential smoothing: The state space approach*. Berlin: Springer-Verlag. http://www.exponentialsmoothing.net

Chapter 8 ARIMA models

ARIMA models provide another approach to time series forecasting. Exponential smoothing and ARIMA models are the two most widely used approaches to time series forecasting, and provide complementary approaches to the problem. While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data.

Before we introduce ARIMA models, we must first discuss the concept of stationarity and the technique of differencing time series.

8.1 Stationarity and differencing

A stationary time series is one whose properties do not depend on the time at which the series is observed.[15] Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times. On the other hand, a white noise series is stationary — it does not matter when you observe it, it should look much the same at any point in time.

Some cases can be confusing — a time series with cyclic behaviour (but with no trend or seasonality) is stationary. This is because the cycles are not of a fixed length, so before we observe the series we cannot be sure where the peaks and troughs of the cycles will be.

In general, a stationary time series will have no predictable patterns in the long-term. Time plots will show the series to be roughly horizontal (although some cyclic behaviour is possible), with constant variance.

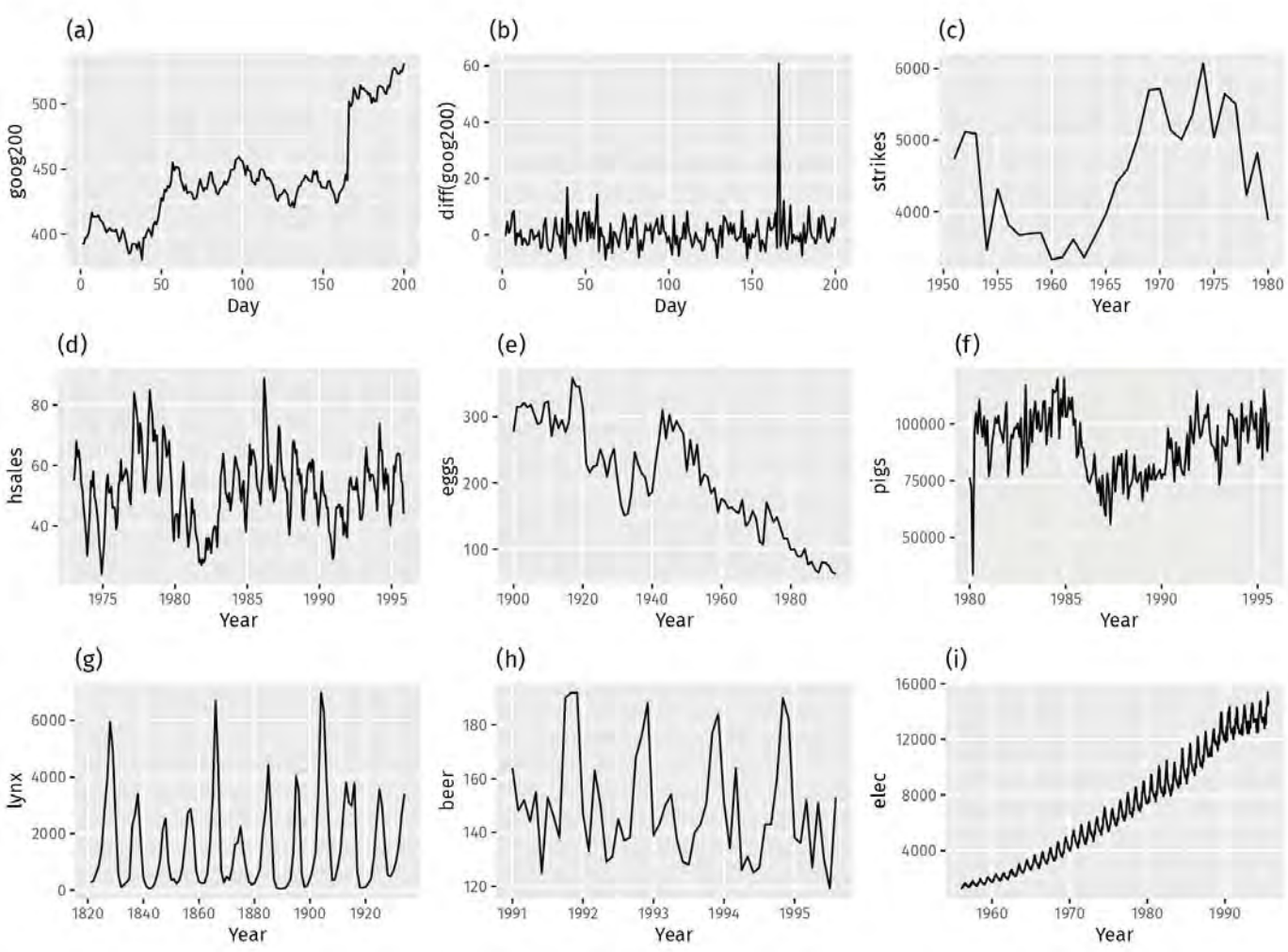


Figure 8.1: Which of these series are stationary? (a) Google stock price for 200 consecutive days; (b) Daily change in the Google stock price for 200 consecutive days; (c) Annual number of strikes in the US; (d) Monthly sales of new one-family houses sold in the US; (e) Annual price of a dozen eggs in the US (constant dollars); (f) Monthly total of pigs slaughtered in Victoria, Australia; (g) Annual total of lynx trapped in the McKenzie River district of north-west Canada; (h) Monthly Australian beer production; (i) Monthly Australian electricity production.

Consider the nine series plotted in Figure 8.1. Which of these do you think are stationary?

Obvious seasonality rules out series (d), (h) and (i). Trends and changing levels rules out series (a), (c), (e), (f) and (i). Increasing variance also rules out (i). That leaves only (b) and (g) as stationary series.

At first glance, the strong cycles in series (g) might appear to make it non-stationary. But these cycles are aperiodic — they are caused when the lynx population becomes too large for the available feed, so that they stop breeding and the population falls to low numbers, then the regeneration of their food sources allows the population to grow again, and so on. In the long-term, the timing of these cycles is not predictable. Hence the series is stationary.

Differencing

In Figure 8.1, note that the Google stock price was non-stationary in panel (a), but the daily changes were stationary in panel (b). This shows one way to make a non-stationary time series stationary — compute the differences between consecutive observations. This is known as **differencing**.

Transformations such as logarithms can help to stabilise the variance of a time series. Differencing can help stabilise the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality.

As well as looking at the time plot of the data, the ACF plot is also useful for identifying non-stationary time series. For a stationary time series, the ACF will drop to zero relatively quickly, while the ACF of non-stationary data decreases slowly. Also, for non-stationary data, the value of r_1 is often large and positive.

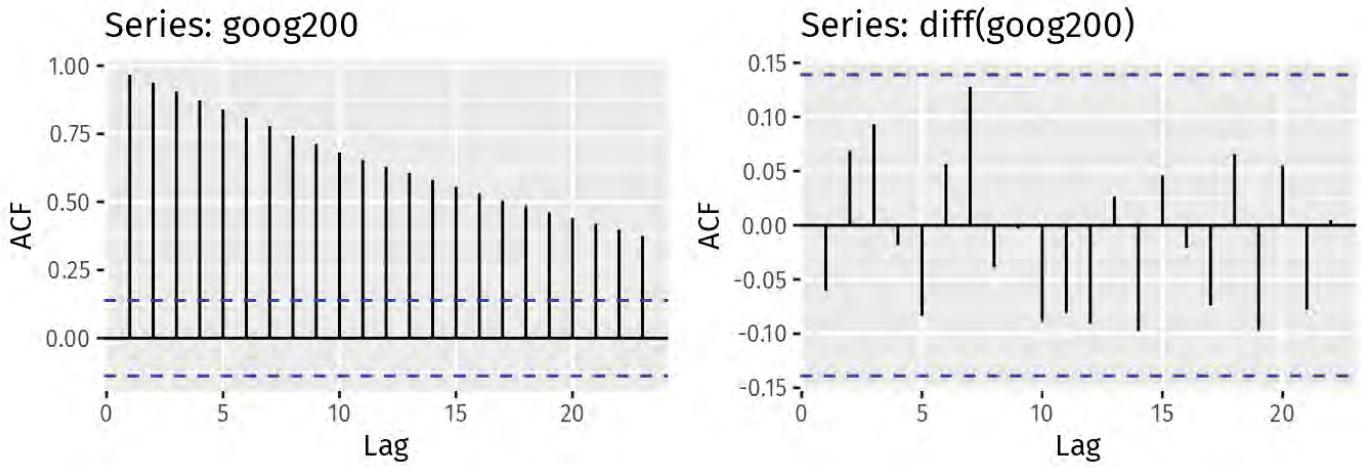


Figure 8.2: The ACF of the Google stock price (left) and of the daily changes in Google stock price (right).

```
Box.test(diff(goog200), lag=10, type="Ljung-Box")
#>
#>  Box-Ljung test
#>
#> data:  diff(goog200)
#> X-squared = 11, df = 10, p-value = 0.4
```

The ACF of the differenced Google stock price looks just like that of a white noise series. There are no autocorrelations lying outside the 95% limits, and the Ljung-Box Q^* statistic has a p-value of 0.355 (for $\ell = 10$). This suggests that the *daily change* in the Google stock price is essentially a random amount which is uncorrelated with that of previous days.

Random walk model

The differenced series is the *change* between consecutive observations in the original series, and can be written as

$$y'_t = y_t - y_{t-1}.$$

The differenced series will have only $T - 1$ values, since it is not possible to calculate a difference y'_1 for the first observation.

When the differenced series is white noise, the model for the original series can be written as

$$y_t - y_{t-1} = \varepsilon_t,$$

where ε_t denotes white noise. Rearranging this leads to the "random walk" model

$$y_t = y_{t-1} + \varepsilon_t.$$

Random walk models are widely used for non-stationary data, particularly financial and economic data. Random walks typically have:

- long periods of apparent trends up or down
- sudden and unpredictable changes in direction.

The forecasts from a random walk model are equal to the last observation, as future movements are unpredictable, and are equally likely to be up or down. Thus, the random walk model underpins naïve forecasts, first introduced in Section 3.1.

A closely related model allows the differences to have a non-zero mean. Then

$$y_t - y_{t-1} = c + \varepsilon_t \quad \text{or} \quad y_t = c + y_{t-1} + \varepsilon_t\ .$$

The value of c is the average of the changes between consecutive observations. If c is positive, then the average change is an increase in the value of y_t. Thus, y_t will tend to drift upwards. However, if c is negative, y_t will tend to drift downwards.

This is the model behind the drift method, also discussed in Section 3.1.

Second-order differencing

Occasionally the differenced data will not appear to be stationary and it may be necessary to difference the data a second time to obtain a stationary series:

$$\begin{aligned} y''_t &= y'_t - y'_{t-1} \\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2}. \end{aligned}$$

In this case, y''_t will have $T-2$ values. Then, we would model the "change in the changes" of the original data. In practice, it is almost never necessary to go beyond second-order differences.

Seasonal differencing

A seasonal difference is the difference between an observation and the previous observation from the same season. So

$$y'_t = y_t - y_{t-m},$$

where $m =$ the number of seasons. These are also called "lag-m differences", as we subtract the observation after a lag of m periods.

If seasonally differenced data appear to be white noise, then an appropriate model for the original data is

$$y_t = y_{t-m} + \varepsilon_t.$$

Forecasts from this model are equal to the last observation from the relevant season. That is, this model gives seasonal naïve forecasts, introduced in Section 3.1.

The bottom panel in Figure 8.3 shows the seasonal differences of the logarithm of the monthly scripts for A10 (antidiabetic) drugs sold in Australia. The transformation and differencing have made the series look relatively stationary.

```
cbind("Sales ($million)" = a10,
      "Monthly log sales" = log(a10),
      "Annual change in log sales" = diff(log(a10),12)) %>%
  autoplot(facets=TRUE) +
    xlab("Year") + ylab("") +
    ggtitle("Antidiabetic drug sales")
```

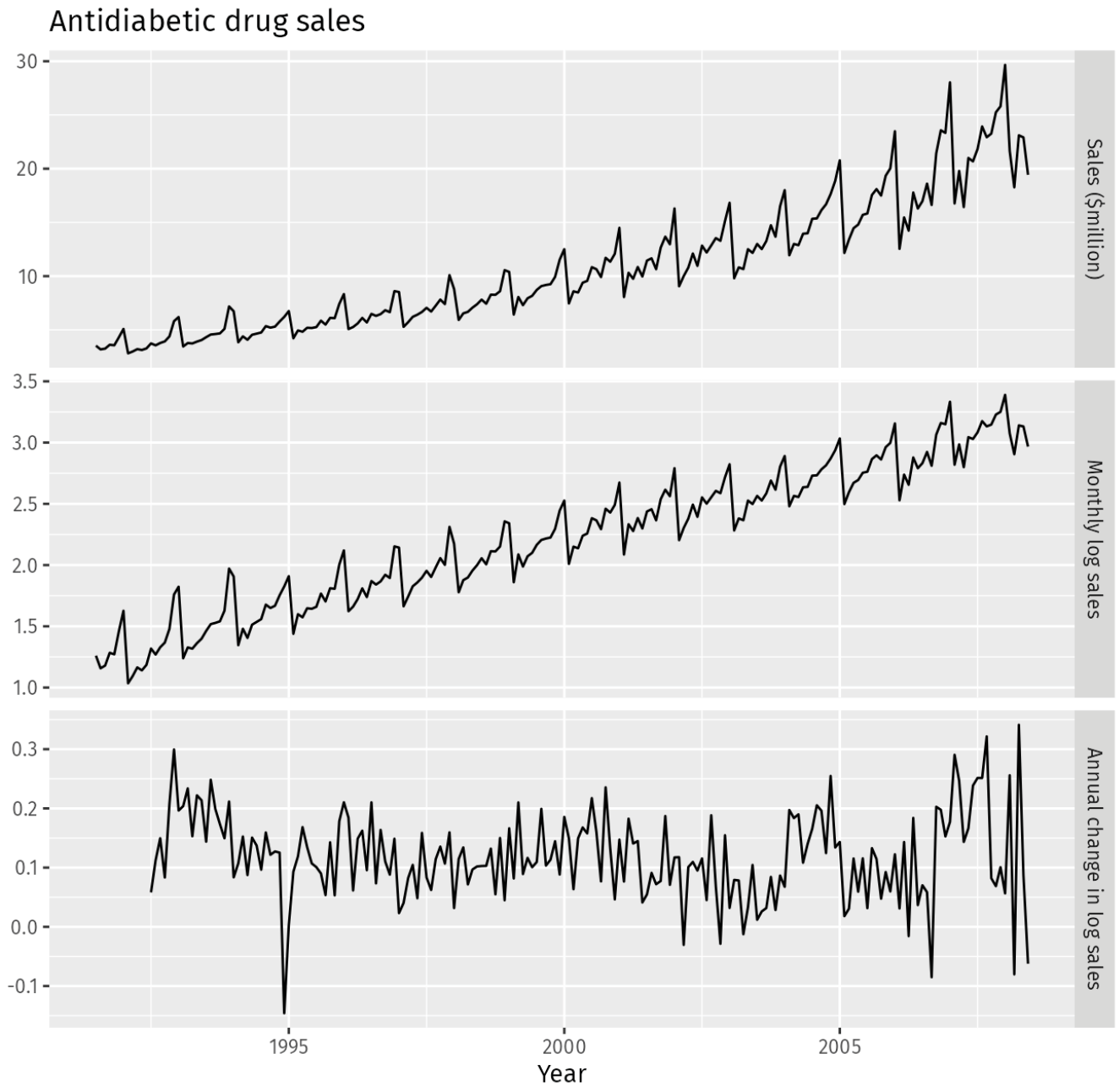


Figure 8.3: Logs and seasonal differences of the A10 (antidiabetic) sales data. The logarithms stabilise the variance, while the seasonal differences remove the seasonality and trend.

To distinguish seasonal differences from ordinary differences, we sometimes refer to ordinary differences as "first differences", meaning differences at lag 1.

Sometimes it is necessary to take both a seasonal difference and a first difference to obtain stationary data, as is shown in Figure 8.4. Here, the data are first transformed using logarithms (second panel), then seasonal differences are calculated (third panel). The data still seem somewhat non-stationary, and so a further lot of first differences are computed (bottom panel).

```
cbind("Billion kWh" = usmelec,
      "Logs" = log(usmelec),
      "Seasonally\n differenced logs" =
        diff(log(usmelec),12),
      "Doubly\n differenced logs" =
        diff(diff(log(usmelec),12),1)) %>%
  autoplot(facets=TRUE) +
    xlab("Year") + ylab("") +
    ggtitle("Monthly US net electricity generation")
```

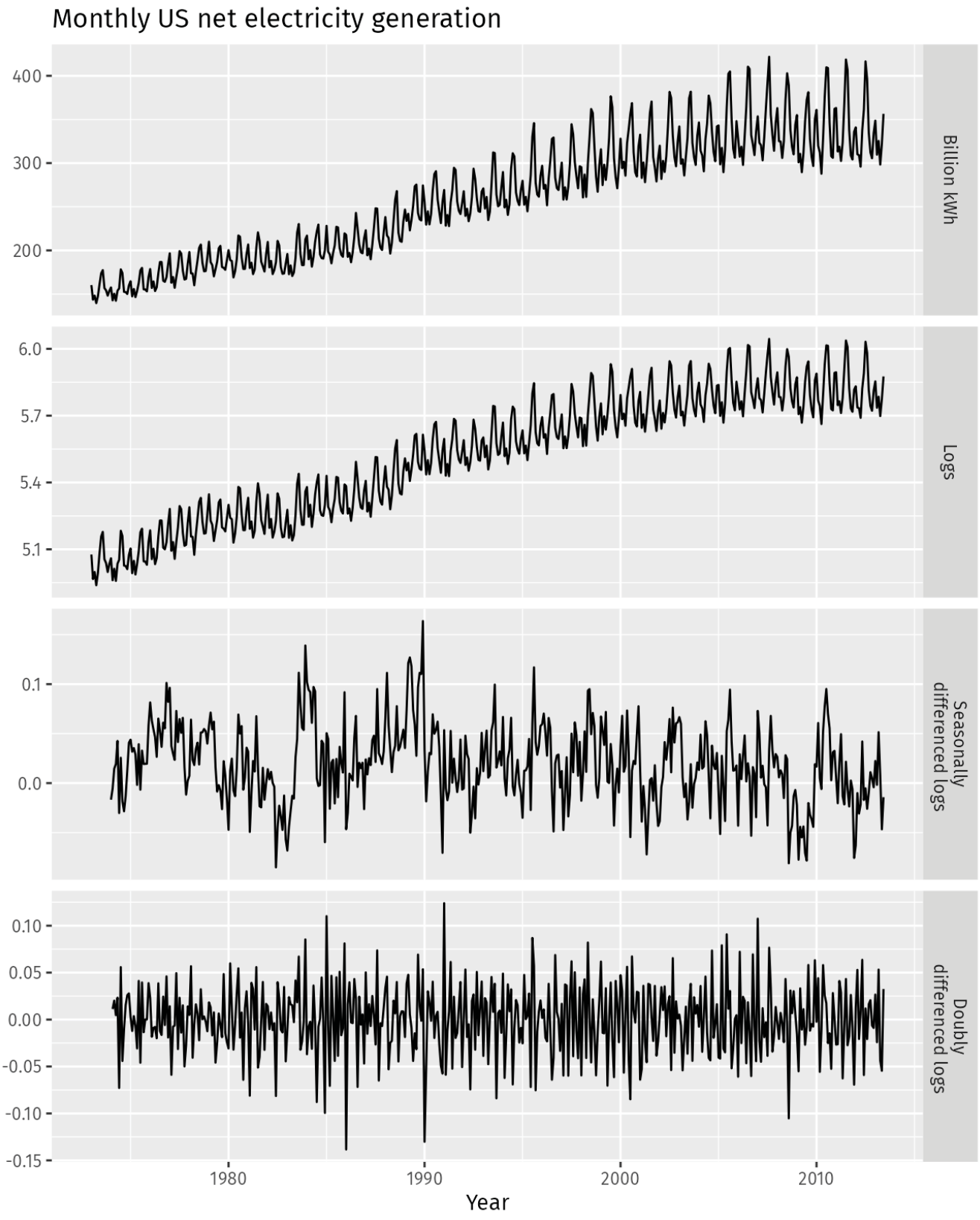


Figure 8.4: Top panel: US net electricity generation (billion kWh). Other panels show the same data after transforming and differencing.

There is a degree of subjectivity in selecting which differences to apply. The seasonally differenced data in Figure 8.3 do not show substantially different behaviour from the seasonally differenced data in Figure 8.4. In the latter case, we could have decided to stop with the seasonally differenced data, and not done an extra round of differencing. In the former case, we could have decided that the data

were not sufficiently stationary and taken an extra round of differencing. Some formal tests for differencing are discussed below, but there are always some choices to be made in the modelling process, and different analysts may make different choices.

If $y'_t = y_t - y_{t-m}$ denotes a seasonally differenced series, then the twice-differenced series is

$$\begin{aligned} y''_t &= y'_t - y'_{t-1} \\ &= (y_t - y_{t-m}) - (y_{t-1} - y_{t-m-1}) \\ &= y_t - y_{t-1} - y_{t-m} + y_{t-m-1} \end{aligned}$$

When both seasonal and first differences are applied, it makes no difference which is done first—the result will be the same. However, if the data have a strong seasonal pattern, we recommend that seasonal differencing be done first, because the resulting series will sometimes be stationary and there will be no need for a further first difference. If first differencing is done first, there will still be seasonality present.

It is important that if differencing is used, the differences are interpretable. First differences are the change between one observation and the next. Seasonal differences are the change between one year to the next. Other lags are unlikely to make much interpretable sense and should be avoided.

Unit root tests

One way to determine more objectively whether differencing is required is to use a *unit root test*. These are statistical hypothesis tests of stationarity that are designed for determining whether differencing is required.

A number of unit root tests are available, which are based on different assumptions and may lead to conflicting answers. In our analysis, we use the *Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test* (Kwiatkowski, Phillips, Schmidt, & Shin, 1992). In this test, the null hypothesis is that the data are stationary, and we look for evidence that the null hypothesis is false. Consequently, small p-values (e.g., less than 0.05) suggest that differencing is required. The test can be computed using the `ur.kpss()` function from the **urca** package.

For example, let us apply it to the Google stock price data.

```
library(urca)
goog %>% ur.kpss() %>% summary()
#>
#> ######################
#> # KPSS Unit Root Test #
#> ######################
#>
#> Test is of type: mu with 7 lags.
#>
#> Value of test-statistic is: 10.72
#>
#> Critical value for a significance level of:
#>                 10pct  5pct 2.5pct  1pct
#> critical values 0.347 0.463  0.574 0.739
```

The test statistic is much bigger than the 1% critical value, indicating that the null hypothesis is rejected. That is, the data are not stationary. We can difference the data, and apply the test again.

```
goog %>% diff() %>% ur.kpss() %>% summary()
#>
#> ######################
#> # KPSS Unit Root Test #
#> ######################
#>
#> Test is of type: mu with 7 lags.
#>
#> Value of test-statistic is: 0.0324
#>
#> Critical value for a significance level of:
#>                 10pct  5pct 2.5pct  1pct
#> critical values 0.347 0.463  0.574 0.739
```

This time, the test statistic is tiny, and well within the range we would expect for stationary data. So we can conclude that the differenced data are stationary.

This process of using a sequence of KPSS tests to determine the appropriate number of first differences is carried out by the function `ndiffs()`.

```
ndiffs(goog)
#> [1] 1
```

As we saw from the KPSS tests above, one difference is required to make the `goog` data stationary.

A similar function for determining whether seasonal differencing is required is `nsdiffs()`, which uses the measure of seasonal strength introduced in Section 6.7 to determine the appropriate number of seasonal differences required. No seasonal differences are suggested if $F_S < 0.64$, otherwise one seasonal difference is suggested.

We can apply `nsdiffs()` to the logged US monthly electricity data.

```
usmelec %>% log() %>% nsdiffs()
#> [1] 1
usmelec %>% log() %>% diff(lag=12) %>% ndiffs()
#> [1] 1
```

Because `nsdiffs()` returns 1 (indicating one seasonal difference is required), we apply the `ndiffs()` function to the seasonally differenced data. These functions suggest we should do both a seasonal difference and a first difference.

Bibliography

Kwiatkowski, D., Phillips, P. C. B., Schmidt, P., & Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, *54*(1-3), 159–178. [DOI]

15. More precisely, if $\{y_t\}$ is a **stationary** time series, then for all s, the distribution of $(y_t, \dots, y_{t+s})$ does not depend on t.↩

8.2 Backshift notation

The backward shift operator B is a useful notational device when working with time series lags:

$$By_t = y_{t-1} \ .$$

(Some references use L for "lag" instead of B for "backshift".) In other words, B, operating on y_t, has the effect of shifting the data back one period. Two applications of B to y_t shifts the data back two periods:

$$B(By_t) = B^2y_t = y_{t-2} \ .$$

For monthly data, if we wish to consider "the same month last year," the notation is $B^{12}y_t = y_{t-12}$.

The backward shift operator is convenient for describing the process of *differencing*. A first difference can be written as

$$y'_t = y_t - y_{t-1} = y_t - By_t = (1-B)y_t \ .$$

Note that a first difference is represented by $(1-B)$. Similarly, if second-order differences have to be computed, then:

$$y''_t = y_t - 2y_{t-1} + y_{t-2} = (1-2B+B^2)y_t = (1-B)^2y_t \ .$$

In general, a dth-order difference can be written as

$$(1-B)^d y_t.$$

Backshift notation is particularly useful when combining differences, as the operator can be treated using ordinary algebraic rules. In particular, terms involving B can be multiplied together.

8.3 Autoregressive models

In a multiple regression model, we forecast the variable of interest using a linear combination of predictors. In an autoregression model, we forecast the variable of interest using a linear combination of *past values of the variable*. The term *auto*regression indicates that it is a regression of the variable against itself.

Thus, an autoregressive model of order p can be written as

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t,$$

where ε_t is white noise. This is like a multiple regression but with *lagged values* of y_t as predictors. We refer to this as an **AR(p) model**, an autoregressive model of order p.

Autoregressive models are remarkably flexible at handling a wide range of different time series patterns. The two series in Figure 8.5 show series from an AR(1) model and an AR(2) model. Changing the parameters $\phi_1, \ldots, \phi_p$ results in different time series patterns. The variance of the error term ε_t will only change the scale of the series, not the patterns.

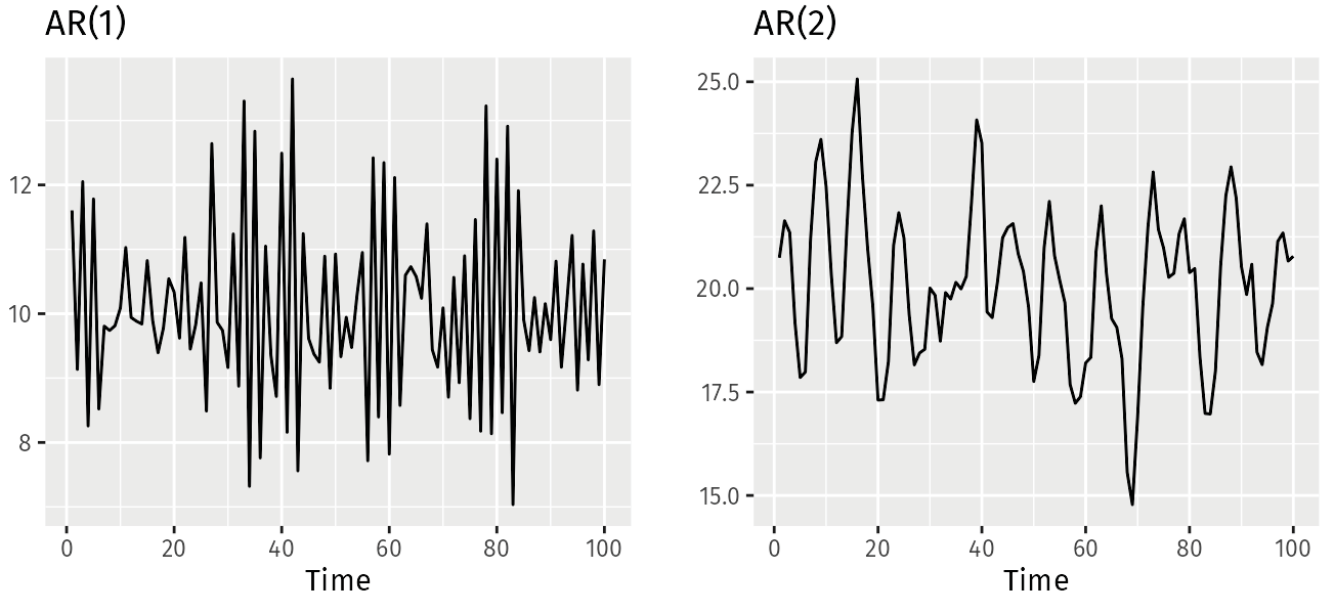


Figure 8.5: Two examples of data from autoregressive models with different parameters. Left: AR(1) with $y_t = 18 - 0.8y_{t-1} + \varepsilon_t$. Right: AR(2) with $y_t = 8 + 1.3y_{t-1} - 0.7y_{t-2} + \varepsilon_t$. In both cases, ε_t is normally distributed white noise with mean zero and variance one.

For an AR(1) model:

- when $\phi_1 = 1$ and $c = 0$, y_t is equivalent to a random walk;
- when $\phi_1 = 1$ and $c \neq 0$, y_t is equivalent to a random walk with drift;
- when $\phi_1 < 0$, y_t tends to oscillate around the mean.

We normally restrict autoregressive models to stationary data, in which case some constraints on the values of the parameters are required.

- For an AR(1) model: $-1 < \phi_1 < 1$.
- For an AR(2) model: $-1 < \phi_2 < 1$, $\phi_1 + \phi_2 < 1$, $\phi_2 - \phi_1 < 1$.

When $p \geq 3$, the restrictions are much more complicated. R takes care of these restrictions when estimating a model.

8.4 Moving average models

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model.

$$y_t = c + \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \cdots + \theta_q\varepsilon_{t-q},$$

where ε_t is white noise. We refer to this as an **MA(q) model**, a moving average model of order q. Of course, we do not *observe* the values of ε_t, so it is not really a regression in the usual sense.

Notice that each value of y_t can be thought of as a weighted moving average of the past few forecast errors. However, moving average *models* should not be confused with the moving average *smoothing* we discussed in Chapter 6. A moving average model is used for forecasting future values, while moving average smoothing is used for estimating the trend-cycle of past values.

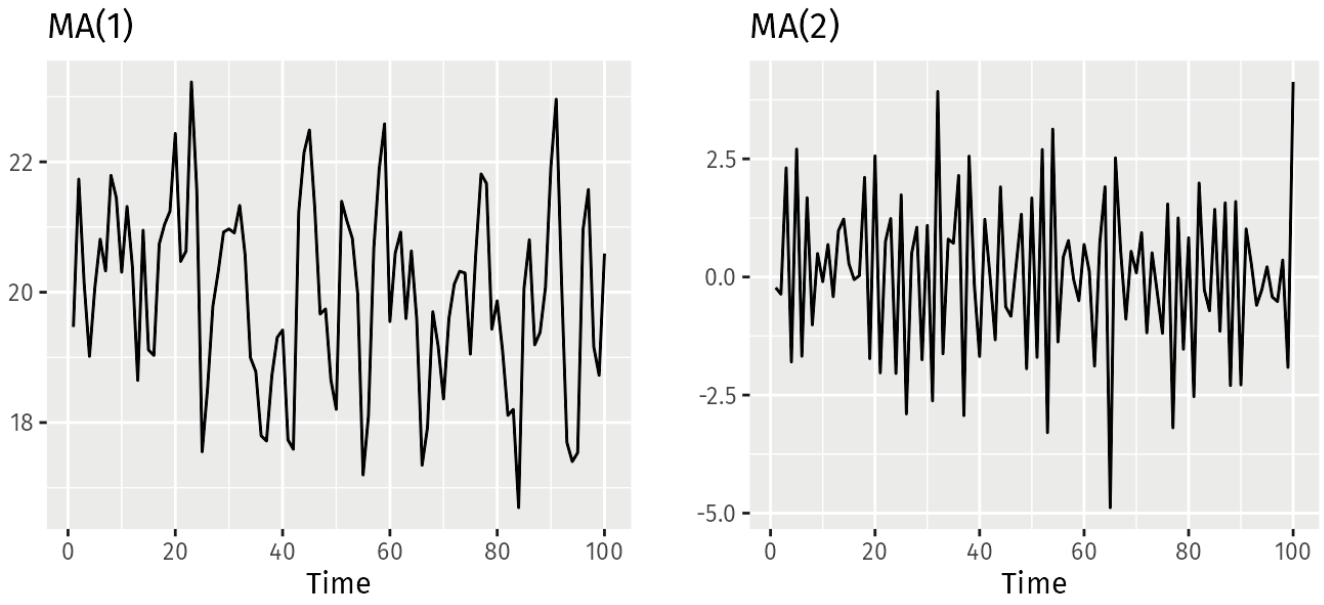


Figure 8.6: Two examples of data from moving average models with different parameters. Left: MA(1) with $y_t = 20 + \varepsilon_t + 0.8\varepsilon_{t-1}$. Right: MA(2) with $y_t = \varepsilon_t - \varepsilon_{t-1} + 0.8\varepsilon_{t-2}$. In both cases, ε_t is normally distributed white noise with mean zero and variance one.

Figure 8.6 shows some data from an MA(1) model and an MA(2) model. Changing the parameters $\theta_1, \ldots, \theta_q$ results in different time series patterns. As with autoregressive models, the variance of the error term ε_t will only change the scale of the series, not the patterns.

It is possible to write any stationary AR(p) model as an MA(∞) model. For example, using repeated substitution, we can demonstrate this for an AR(1) model:

$$\begin{aligned} y_t &= \phi_1 y_{t-1} + \varepsilon_t \\ &= \phi_1(\phi_1 y_{t-2} + \varepsilon_{t-1}) + \varepsilon_t \\ &= \phi_1^2 y_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \\ &= \phi_1^3 y_{t-3} + \phi_1^2 \varepsilon_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \\ &\text{etc.} \end{aligned}$$

Provided $-1 < \phi_1 < 1$, the value of ϕ_1^k will get smaller as k gets larger. So eventually we obtain

$$y_t = \varepsilon_t + \phi_1 \varepsilon_{t-1} + \phi_1^2 \varepsilon_{t-2} + \phi_1^3 \varepsilon_{t-3} + \cdots,$$

an MA(∞) process.

The reverse result holds if we impose some constraints on the MA parameters. Then the MA model is called **invertible**. That is, we can write any invertible MA(q) process as an AR(∞) process. Invertible models are not simply introduced to enable us to convert from MA models to AR models. They also have some desirable mathematical properties.

For example, consider the MA(1) process, $y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1}$. In its AR($\infty$) representation, the most recent error can be written as a linear function of current and past observations:

$$\varepsilon_t = \sum_{j=0}^{\infty} (-\theta)^j y_{t-j}.$$

When $|\theta| > 1$, the weights increase as lags increase, so the more distant the observations the greater their influence on the current error. When $|\theta| = 1$, the weights are constant in size, and the distant observations have the same influence as the recent observations. As neither of these situations make much sense, we require $|\theta| < 1$, so the most recent observations have higher weight than observations from the more distant past. Thus, the process is invertible when $|\theta| < 1$.

The invertibility constraints for other models are similar to the stationarity constraints.

- For an MA(1) model: $-1 < \theta_1 < 1$.
- For an MA(2) model: $-1 < \theta_2 < 1,\ \ \theta_2 + \theta_1 > -1,\ \ \theta_1 - \theta_2 < 1$.

8.5 Non-seasonal ARIMA models

If we combine differencing with autoregression and a moving average model, we obtain a non-seasonal ARIMA model. ARIMA is an acronym for AutoRegressive Integrated Moving Average (in this context, "integration" is the reverse of differencing). The full model can be written as

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t, \tag{8.1}$$

where y'_t is the differenced series (it may have been differenced more than once). The "predictors" on the right hand side include both lagged values of y_t and lagged errors. We call this an **ARIMA(p, d, q) model**, where

$p =$	order of the autoregressive part;
$d =$	degree of first differencing involved;
$q =$	order of the moving average part.

The same stationarity and invertibility conditions that are used for autoregressive and moving average models also apply to an ARIMA model.

Many of the models we have already discussed are special cases of the ARIMA model, as shown in Table 8.1.

Table 8.1: Special cases of ARIMA models.

White noise	ARIMA(0,0,0)
Random walk	ARIMA(0,1,0) with no constant
Random walk with drift	ARIMA(0,1,0) with a constant
Autoregression	ARIMA(p,0,0)
Moving average	ARIMA(0,0,q)

Once we start combining components in this way to form more complicated models, it is much easier to work with the backshift notation. For example, Equation (8.1) can be written in backshift notation as

$$\underset{\uparrow}{(1 - \phi_1 B - \cdots - \phi_p B^p)} \quad \underset{\uparrow}{(1 - B)^d y_t} \quad = \quad \underset{\uparrow}{c + (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t} \tag{8.2}$$
$$\text{AR}(p) \qquad d \text{ differences} \qquad \text{MA}(q)$$

R uses a slightly different parameterisation:

$$(1-\phi_1 B-\cdots-\phi_p B^p)(y_t'-\mu)=(1+\theta_1 B+\cdots+\theta_q B^q)\varepsilon_t, \tag{8.3}$$

where $y_t'=(1-B)^d y_t$ and μ is the mean of y_t'. To convert to the form given by (8.2), set $c=\mu(1-\phi_1-\cdots-\phi_p)$.

Selecting appropriate values for p, d and q can be difficult. However, the `auto.arima()` function in R will do it for you automatically. In Section 8.7, we will learn how this function works, along with some methods for choosing these values yourself.

US consumption expenditure

Figure 8.7 shows quarterly percentage changes in US consumption expenditure. Although it is a quarterly series, there does not appear to be a seasonal pattern, so we will fit a non-seasonal ARIMA model.

```
autoplot(uschange[,"Consumption"]) +
  xlab("Year") + ylab("Quarterly percentage change")
```

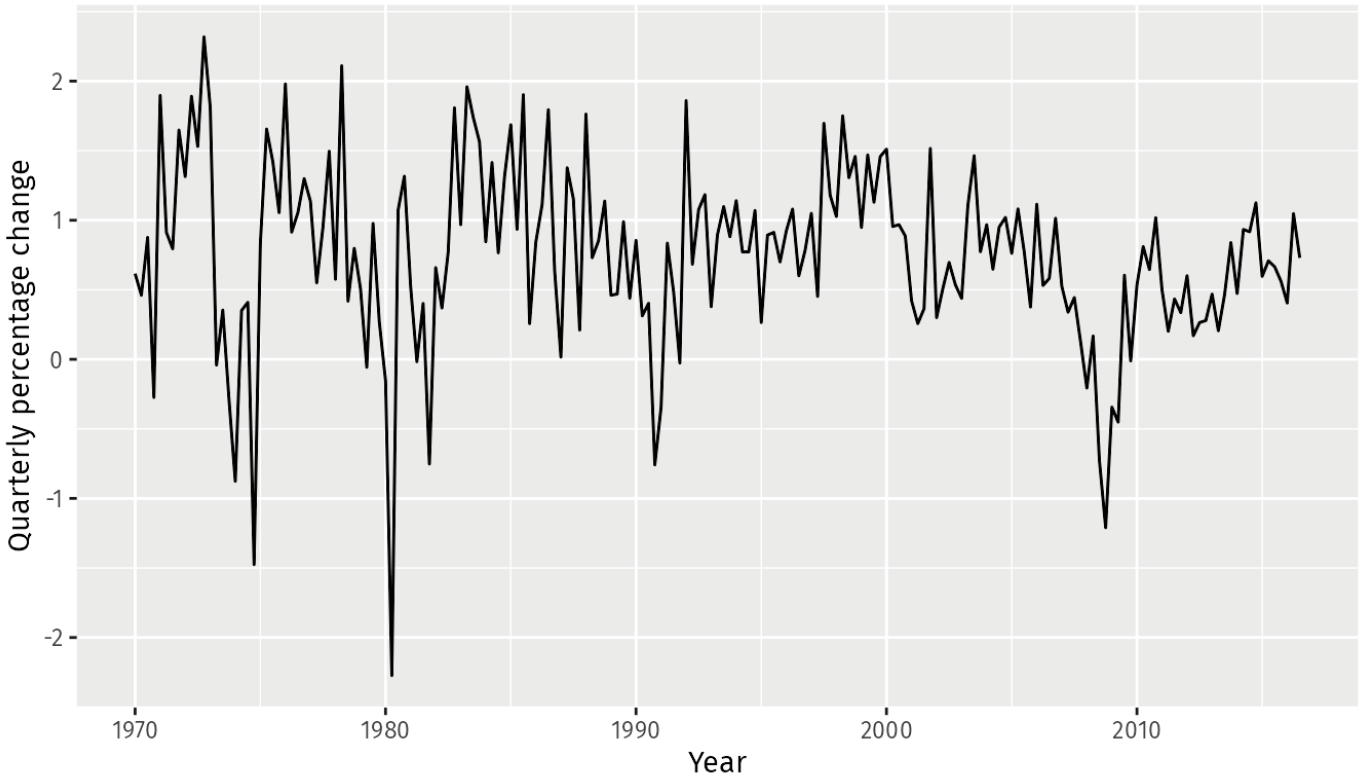


Figure 8.7: Quarterly percentage change in US consumption expenditure.

The following R code was used to select a model automatically.

```
fit <- auto.arima(uschange[,"Consumption"], seasonal=FALSE)
```

```
#> Series: uschange[, "Consumption"]
#> ARIMA(1,0,3) with non-zero mean
#>
#> Coefficients:
#>          ar1     ma1    ma2    ma3   mean
#>       0.589  -0.353  0.085  0.174  0.745
#> s.e.  0.154   0.166  0.082  0.084  0.093
#>
#> sigma^2 = 0.35:  log likelihood = -164.8
#> AIC=341.6   AICc=342.1   BIC=361
```

This is an ARIMA(1,0,3) model:

$$y_t = c + 0.589y_{t-1} - 0.353\varepsilon_{t-1} + 0.0846\varepsilon_{t-2} + 0.174\varepsilon_{t-3} + \varepsilon_t,$$

where $c = 0.745 \times (1 - 0.589) = 0.307$ and ε_t is white noise with a standard deviation of $0.592 = \sqrt{0.350}$. Forecasts from the model are shown in Figure 8.8.

```
fit %>% forecast(h=10) %>% autoplot(include=80)
```

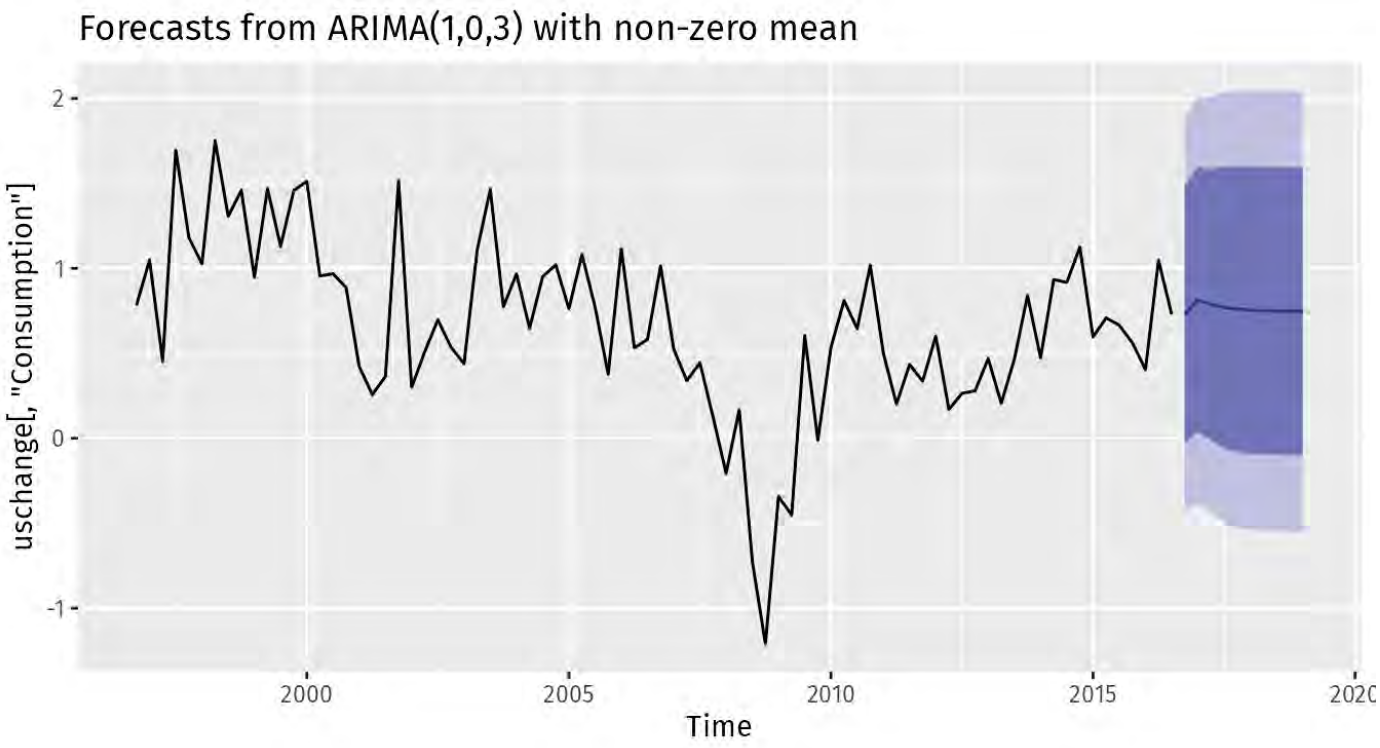


Figure 8.8: Forecasts of quarterly percentage changes in US consumption expenditure.

Understanding ARIMA models

The `auto.arima()` function is useful, but anything automated can be a little dangerous, and it is worth understanding something of the behaviour of the models even when you rely on an automatic procedure to choose the model for you.

The constant c has an important effect on the long-term forecasts obtained from these models.

- If $c = 0$ and $d = 0$, the long-term forecasts will go to zero.
- If $c = 0$ and $d = 1$, the long-term forecasts will go to a non-zero constant.
- If $c = 0$ and $d = 2$, the long-term forecasts will follow a straight line.
- If $c \neq 0$ and $d = 0$, the long-term forecasts will go to the mean of the data.
- If $c \neq 0$ and $d = 1$, the long-term forecasts will follow a straight line.
- If $c \neq 0$ and $d = 2$, the long-term forecasts will follow a quadratic trend.

The value of d also has an effect on the prediction intervals — the higher the value of d, the more rapidly the prediction intervals increase in size. For $d = 0$, the long-term forecast standard deviation will go to the standard deviation of the historical data, so the prediction intervals will all be essentially the same.

This behaviour is seen in Figure 8.8 where $d = 0$ and $c \neq 0$. In this figure, the prediction intervals are almost the same for the last few forecast horizons, and the point forecasts are equal to the mean of the data.

The value of p is important if the data show cycles. To obtain cyclic forecasts, it is necessary to have $p \geq 2$, along with some additional conditions on the parameters. For an AR(2) model, cyclic behaviour occurs if $\phi_1^2 + 4\phi_2 < 0$. In that case, the average period of the cycles is [16]

$$\frac{2\pi}{\arccos(-\phi_1(1-\phi_2)/(4\phi_2))}.$$

ACF and PACF plots

It is usually not possible to tell, simply from a time plot, what values of p and q are appropriate for the data. However, it is sometimes possible to use the ACF plot, and the closely related PACF plot, to determine appropriate values for p and q.

Recall that an ACF plot shows the autocorrelations which measure the relationship between y_t and y_{t-k} for different values of k. Now if y_t and y_{t-1} are correlated, then y_{t-1} and y_{t-2} must also be correlated. However, then y_t and y_{t-2} might be correlated, simply because they are both connected to y_{t-1}, rather than because of any new information contained in y_{t-2} that could be used in forecasting y_t.

To overcome this problem, we can use **partial autocorrelations**. These measure the relationship between y_t and y_{t-k} after removing the effects of lags $1, 2, 3, \ldots, k-1$. So the first partial autocorrelation is identical to the first autocorrelation, because there is nothing between them to remove. Each partial autocorrelation can be estimated as the last

coefficient in an autoregressive model. Specifically, α_k, the kth partial autocorrelation coefficient, is equal to the estimate of ϕ_k in an AR(k) model. In practice, there are more efficient algorithms for computing α_k than fitting all of these autoregressions, but they give the same results.

Figures 8.9 and 8.10 shows the ACF and PACF plots for the US consumption data shown in Figure 8.7. The partial autocorrelations have the same critical values of $\pm 1.96/\sqrt{T}$ as for ordinary autocorrelations, and these are typically shown on the plot as in Figure 8.9.

```
ggAcf(uschange[,"Consumption"])
```

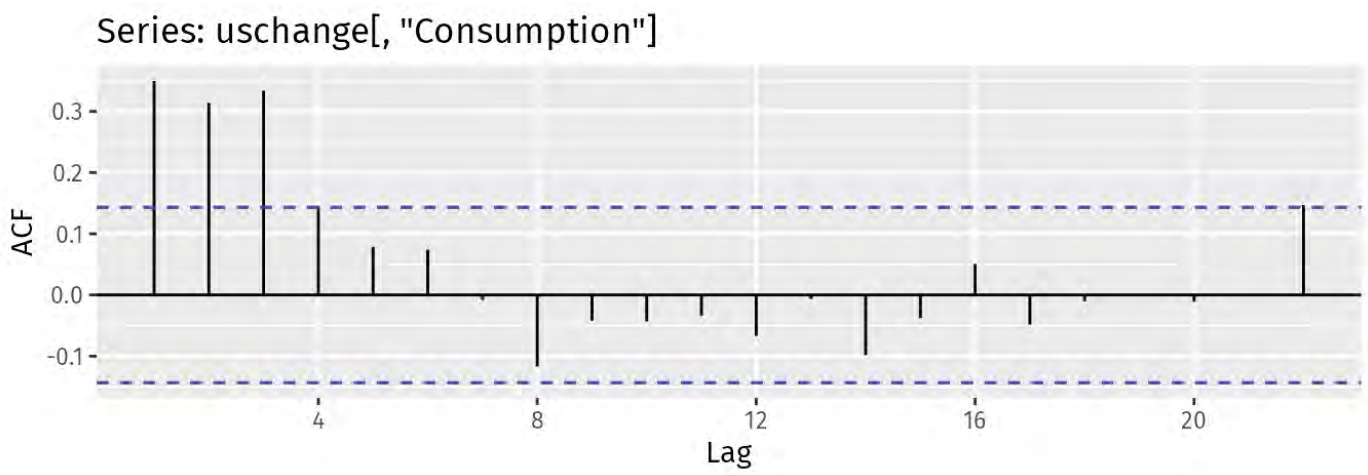


Figure 8.9: ACF of quarterly percentage change in US consumption.

```
ggPacf(uschange[,"Consumption"])
```

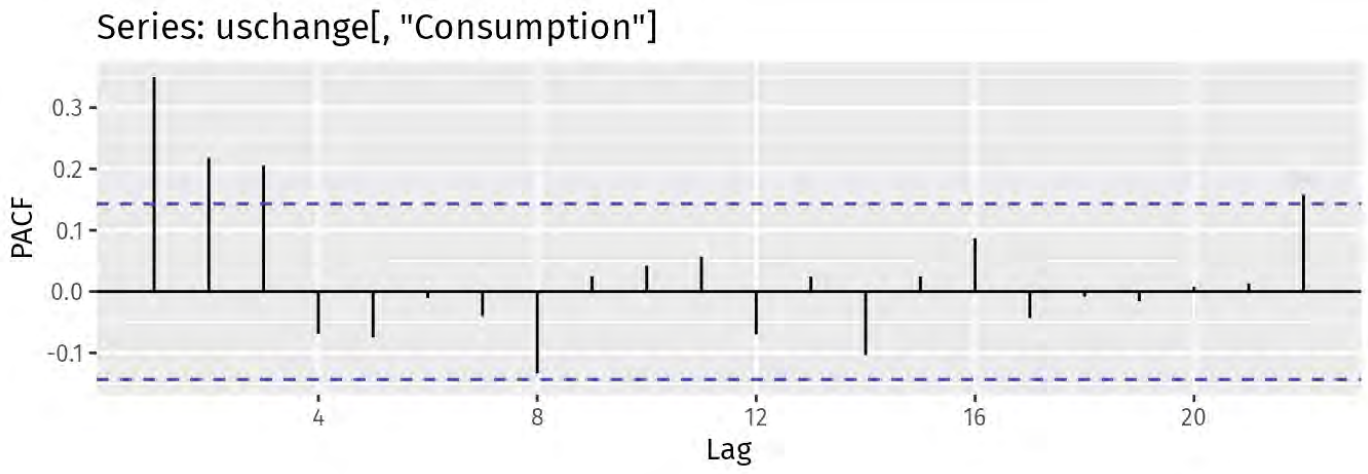


Figure 8.10: PACF of quarterly percentage change in US consumption.

If the data are from an ARIMA(p,d,0) or ARIMA(0,d,q) model, then the ACF and PACF plots can be helpful in determining the value of p or q.[17] If p and q are both positive, then the plots do not help in finding suitable values of p and q.

The data may follow an ARIMA(p,d,0) model if the ACF and PACF plots of the differenced data show the following patterns:

- the ACF is exponentially decaying or sinusoidal;

- there is a significant spike at lag p in the PACF, but none beyond lag p.

The data may follow an ARIMA(0,d,q) model if the ACF and PACF plots of the differenced data show the following patterns:

- the PACF is exponentially decaying or sinusoidal;
- there is a significant spike at lag q in the ACF, but none beyond lag q.

In Figure 8.9, we see that there are three spikes in the ACF, followed by an almost significant spike at lag 4. In the PACF, there are three significant spikes, and then no significant spikes thereafter (apart from one just outside the bounds at lag 22). We can ignore one significant spike in each plot if it is just outside the limits, and not in the first few lags. After all, the probability of a spike being significant by chance is about one in twenty, and we are plotting 22 spikes in each plot. The pattern in the first three spikes is what we would expect from an ARIMA(3,0,0), as the PACF tends to decrease. So in this case, the ACF and PACF lead us to think an ARIMA(3,0,0) model might be appropriate.

```
(fit2 <- Arima(uschange[,"Consumption"], order=c(3,0,0)))
#> Series: uschange[, "Consumption"]
#> ARIMA(3,0,0) with non-zero mean
#>
#> Coefficients:
#>         ar1    ar2    ar3   mean
#>       0.227  0.160  0.203  0.745
#> s.e.  0.071  0.072  0.071  0.103
#>
#> sigma^2 = 0.349:  log likelihood = -165.2
#> AIC=340.3   AICc=340.7   BIC=356.5
```

This model is actually slightly better than the model identified by `auto.arima()` (with an AICc value of 340.67 compared to 342.08). The `auto.arima()` function did not find this model because it does not consider all possible models in its search. You can make it work harder by using the arguments `stepwise=FALSE` and `approximation=FALSE` :

```
(fit3 <- auto.arima(uschange[,"Consumption"], seasonal=FALSE,
  stepwise=FALSE, approximation=FALSE))
#> Series: uschange[, "Consumption"]
#> ARIMA(3,0,0) with non-zero mean
#>
#> Coefficients:
#>         ar1    ar2    ar3   mean
#>       0.227  0.160  0.203  0.745
#> s.e.  0.071  0.072  0.071  0.103
#>
#> sigma^2 = 0.349:  log likelihood = -165.2
#> AIC=340.3   AICc=340.7   BIC=356.5
```

We also use the argument `seasonal=FALSE` to prevent it searching for seasonal ARIMA models; we will consider these models in Section 8.9.

This time, `auto.arima()` has found the same model that we guessed from the ACF and PACF plots. The forecasts from this ARIMA(3,0,0) model are almost identical to those shown in Figure 8.8 for the ARIMA(1,0,3) model, so we do not produce the plot here.

16. arc cos is the inverse cosine function. You should be able to find it on your calculator. It may be labelled acos or $\cos^{-1}$.↩

17. A convenient way to produce a time plot, ACF plot and PACF plot in one command is to use the `ggtsdisplay()` function.↩

8.6 Estimation and order selection

Maximum likelihood estimation

Once the model order has been identified (i.e., the values of p, d and q), we need to estimate the parameters c, $\phi_1, \dots, \phi_p$, $\theta_1, \dots, \theta_q$. When R estimates the ARIMA model, it uses *maximum likelihood estimation* (MLE). This technique finds the values of the parameters which maximise the probability of obtaining the data that we have observed. For ARIMA models, MLE is similar to the *least squares* estimates that would be obtained by minimising

$$\sum_{t=1}^{T} \varepsilon_t^2.$$

(For the regression models considered in Chapter 5, MLE gives exactly the same parameter estimates as least squares estimation.) Note that ARIMA models are much more complicated to estimate than regression models, and different software will give slightly different answers as they use different methods of estimation, and different optimisation algorithms.

In practice, R will report the value of the *log likelihood* of the data; that is, the logarithm of the probability of the observed data coming from the estimated model. For given values of p, d and q, R will try to maximise the log likelihood when finding parameter estimates.

Information Criteria

Akaike's Information Criterion (AIC), which was useful in selecting predictors for regression, is also useful for determining the order of an ARIMA model. It can be written as

$$\text{AIC} = -2\log(L) + 2(p + q + k + 1),$$

where L is the likelihood of the data, $k = 1$ if $c \neq 0$ and $k = 0$ if $c = 0$. Note that the last term in parentheses is the number of parameters in the model (including σ^2, the variance of the residuals).

For ARIMA models, the corrected AIC can be written as

$$\text{AICc} = \text{AIC} + \frac{2(p+q+k+1)(p+q+k+2)}{T-p-q-k-2},$$

and the Bayesian Information Criterion can be written as

$$\text{BIC} = \text{AIC} + [\log(T) - 2](p+q+k+1).$$

Good models are obtained by minimising the AIC, AICc or BIC. Our preference is to use the AICc.

It is important to note that these information criteria tend not to be good guides to selecting the appropriate order of differencing (d) of a model, but only for selecting the values of p and q. This is because the differencing changes the data on which the likelihood is computed, making the AIC values between models with different orders of differencing not comparable. So we need to use some other approach to choose d, and then we can use the AICc to select p and q.

8.7 ARIMA modelling in R

How does `auto.arima()` work?

The `auto.arima()` function in R uses a variation of the Hyndman-Khandakar algorithm (Hyndman & Khandakar, 2008), which combines unit root tests, minimisation of the AICc and MLE to obtain an ARIMA model. The arguments to `auto.arima()` provide for many variations on the algorithm. What is described here is the default behaviour.

Hyndman-Khandakar algorithm for automatic ARIMA modelling

1. The number of differences $0 \leq d \leq 2$ is determined using repeated KPSS tests.

2. The values of p and q are then chosen by minimising the AICc after differencing the data d times. Rather than considering every possible combination of p and q, the algorithm uses a stepwise search to traverse the model space.

   a. Four initial models are fitted:
   - ARIMA$(0,d,0)$,
   - ARIMA$(2,d,2)$,
   - ARIMA$(1,d,0)$,
   - ARIMA$(0,d,1)$.

   A constant is included unless $d=2$. If $d \leq 1$, an additional model is also fitted:
   - ARIMA$(0,d,0)$ without a constant.

   b. The best model (with the smallest AICc value) fitted in step (a) is set to be the "current model".

   c. Variations on the current model are considered:
   - vary p and/or q from the current model by ± 1;
   - include/exclude c from the current model.

   The best model considered so far (either the current model or one of these variations) becomes the new current model.

   d. Repeat Step 2(c) until no lower AICc can be found.

The default procedure uses some approximations to speed up the search. These approximations can be avoided with the argument `approximation=FALSE`. It is possible that the minimum AICc model will not be found due to these approximations, or because of the use of a stepwise procedure. A much larger set of models will be searched if the argument `stepwise=FALSE` is used. See the help file for a full description of the arguments.

Choosing your own model

If you want to choose the model yourself, use the `Arima()` function in R. There is another function `arima()` in R which also fits an ARIMA model. However, it does not allow for the constant c unless $d = 0$, and it does not return everything required for other functions in the **forecast** package to work. Finally, it does not allow the estimated model to be applied to new data (which is useful for checking forecast accuracy). Consequently, it is recommended that `Arima()` be used instead.

Modelling procedure

When fitting an ARIMA model to a set of (non-seasonal) time series data, the following procedure provides a useful general approach.

1. Plot the data and identify any unusual observations.
2. If necessary, transform the data (using a Box-Cox transformation) to stabilise the variance.
3. If the data are non-stationary, take first differences of the data until the data are stationary.
4. Examine the ACF/PACF: Is an ARIMA($p, d, 0$) or ARIMA($0, d, q$) model appropriate?
5. Try your chosen model(s), and use the AICc to search for a better model.
6. Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.
7. Once the residuals look like white noise, calculate forecasts.

The Hyndman-Khandakar algorithm only takes care of steps 3–5. So even if you use it, you will still need to take care of the other steps yourself.

The process is summarised in Figure 8.11.

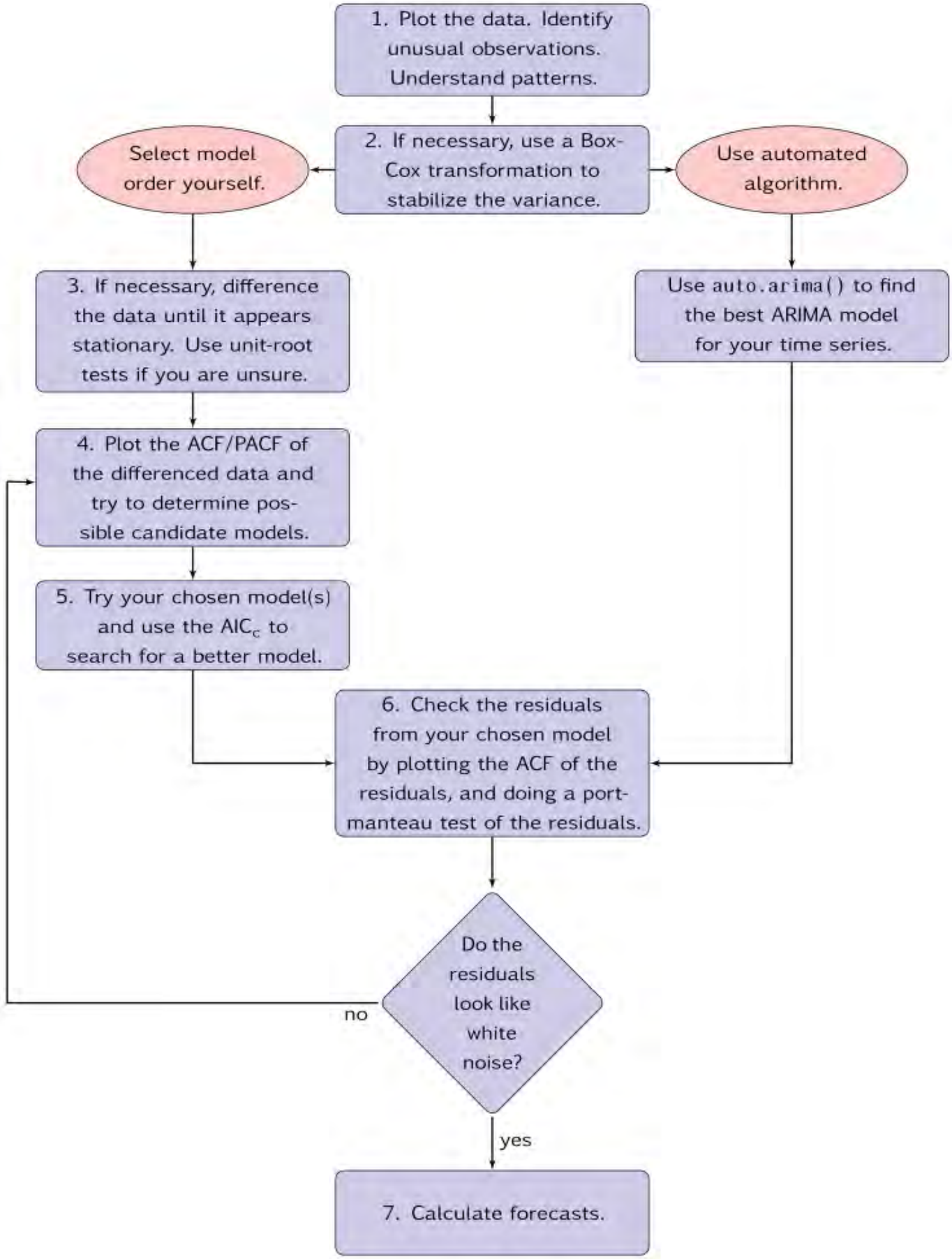


Figure 8.11: General process for forecasting using an ARIMA model.

Portmanteau tests of residuals for ARIMA models

With ARIMA models, more accurate portmanteau tests are obtained if the degrees of freedom of the test statistic are adjusted to take account of the number of parameters in the model. Specifically, we use $\ell - K$ degrees of freedom in the test, where K is the number of AR and MA parameters in the model. So for the non-seasonal models, we have considered so far, $K = p + q$. The correct value of K is automatically determined in the `checkresiduals()` function.

Example: Seasonally adjusted electrical equipment orders

We will apply this procedure to the seasonally adjusted electrical equipment orders data shown in Figure 8.12.

```
elecequip %>% stl(s.window='periodic') %>% seasadj() -> eeadj
autoplot(eeadj)
```

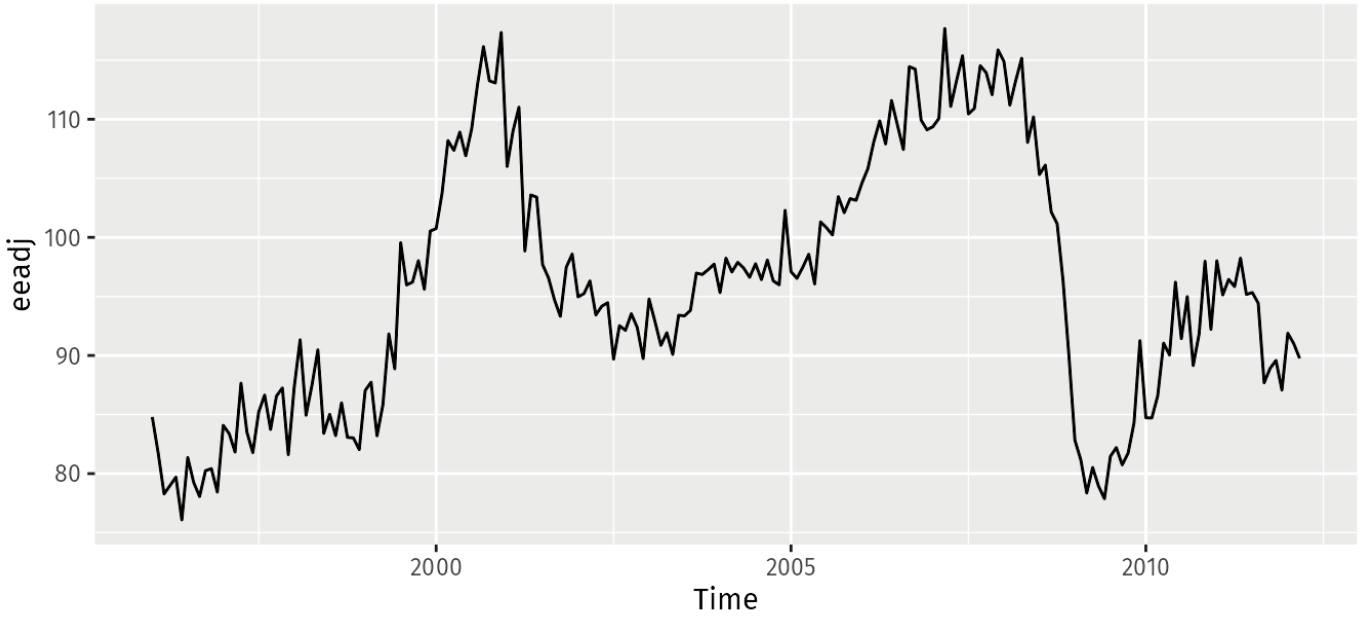


Figure 8.12: Seasonally adjusted electrical equipment orders index in the Euro area.

1. The time plot shows some sudden changes, particularly the big drop in 2008/2009. These changes are due to the global economic environment. Otherwise there is nothing unusual about the time plot and there appears to be no need to do any data adjustments.

2. There is no evidence of changing variance, so we will not do a Box-Cox transformation.

3. The data are clearly non-stationary, as the series wanders up and down for long periods. Consequently, we will take a first difference of the data. The differenced data are shown in Figure 8.13. These look stationary, and so we will not consider further differences.

```
eeadj %>% diff() %>% ggtsdisplay(main="")
```

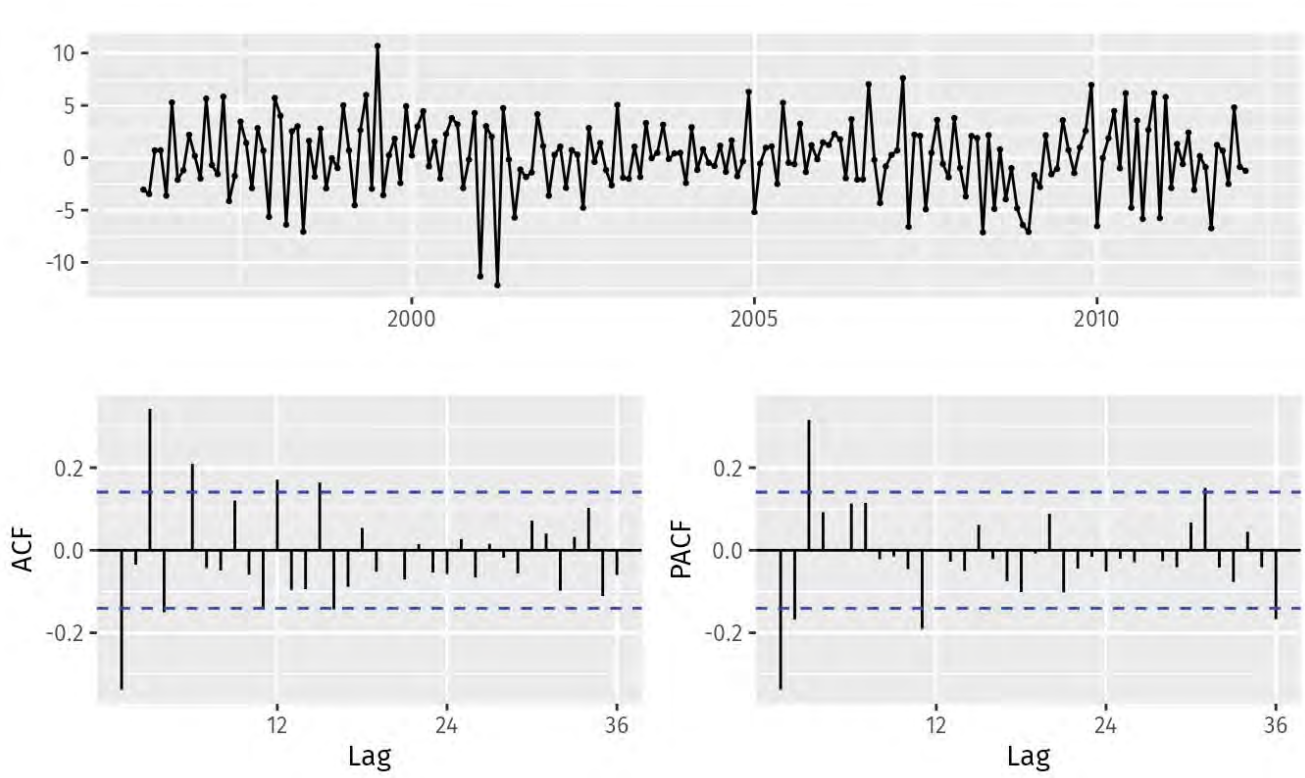


Figure 8.13: Time plot and ACF and PACF plots for the differenced seasonally adjusted electrical equipment data.

4. The PACF shown in Figure 8.13 is suggestive of an AR(3) model. So an initial candidate model is an ARIMA(3,1,0). There are no other obvious candidate models.

5. We fit an ARIMA(3,1,0) model along with variations including ARIMA(4,1,0), ARIMA(2,1,0), ARIMA(3,1,1), etc. Of these, the ARIMA(3,1,1) has a slightly smaller AICc value.

```
(fit <- Arima(eeadj, order=c(3,1,1)))
#> Series: eeadj
#> ARIMA(3,1,1)
#>
#> Coefficients:
#>          ar1    ar2    ar3     ma1
#>       0.004  0.092  0.370  -0.392
#> s.e.  0.220  0.098  0.067   0.243
#>
#> sigma^2 = 9.58:  log likelihood = -492.7
#> AIC=995.4   AICc=995.7   BIC=1012
```

6. The ACF plot of the residuals from the ARIMA(3,1,1) model shows that all autocorrelations are within the threshold limits, indicating that the residuals are behaving like white noise. A portmanteau test returns a large p-value, also suggesting that the residuals are white noise.

```
checkresiduals(fit)
```

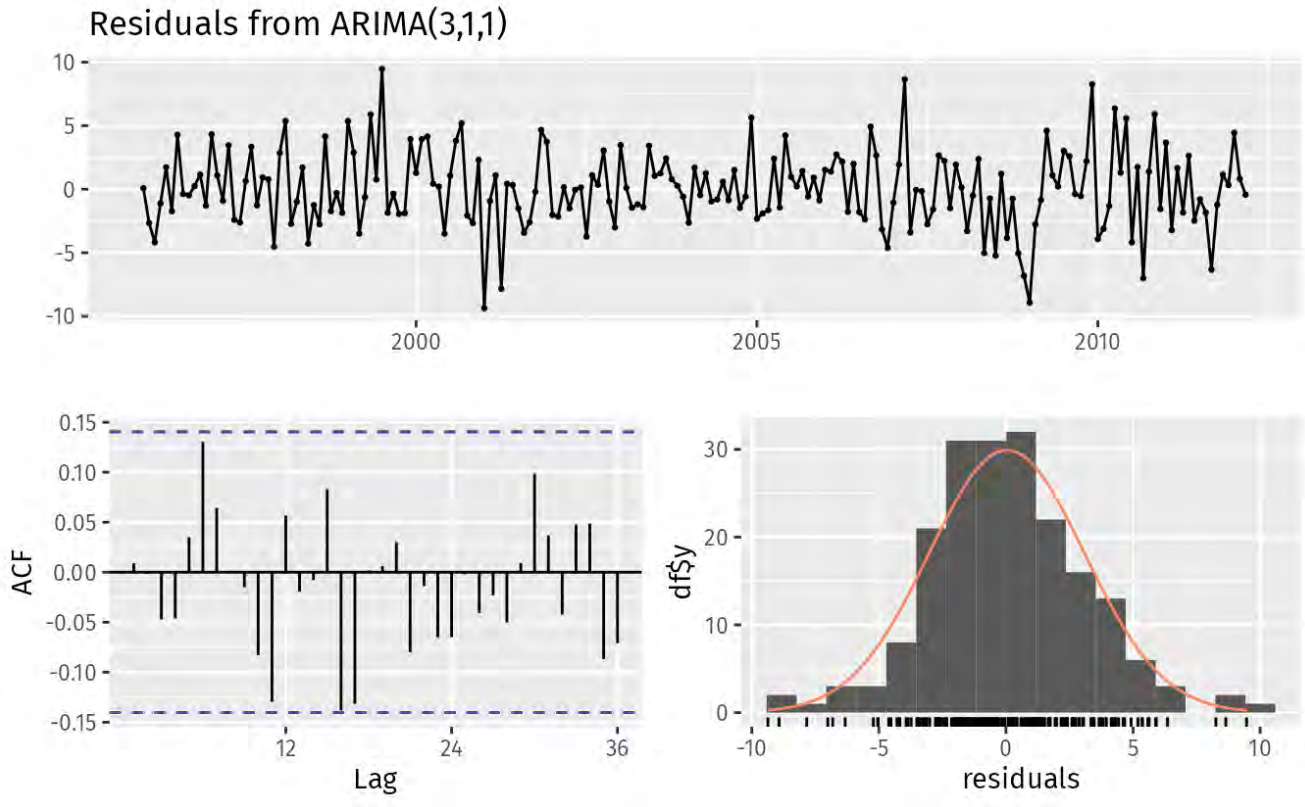


Figure 8.14: Residual plots for the ARIMA(3,1,1) model.

```
#>
#>  Ljung-Box test
#>
#> data:  Residuals from ARIMA(3,1,1)
#> Q* = 24, df = 20, p-value = 0.2
#>
#> Model df: 4.   Total lags used: 24
```

7. Forecasts from the chosen model are shown in Figure 8.15.

```
autoplot(forecast(fit))
```

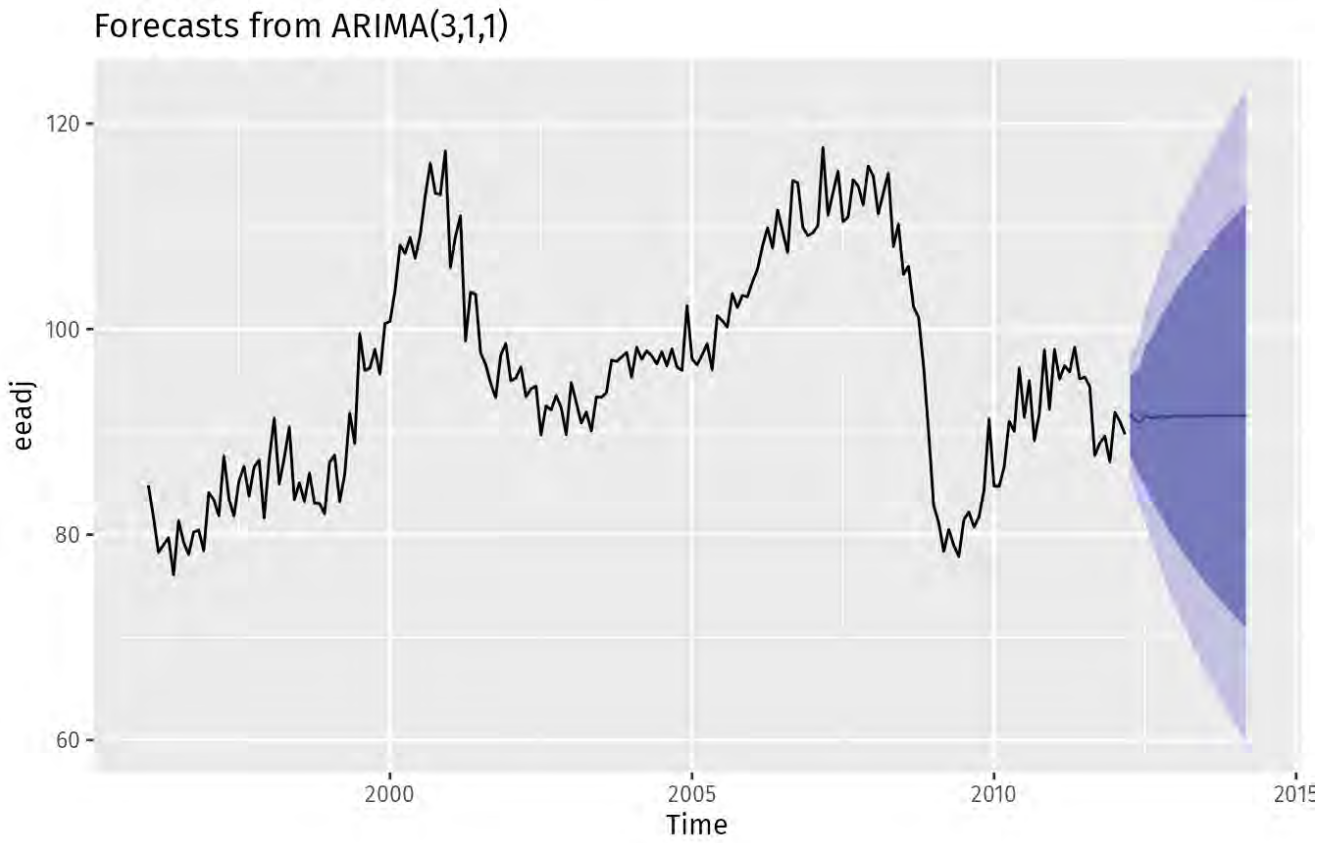


Figure 8.15: Forecasts for the seasonally adjusted electrical orders index.

If we had used the automated algorithm instead, we would have obtained an ARIMA(3,1,0) model using the default settings, but the ARIMA(3,1,1) model if we had set `approximation=FALSE` .

Understanding constants in R

A non-seasonal ARIMA model can be written as

$$(1-\phi_1 B - \cdots - \phi_p B^p)(1-B)^d y_t = c + (1+\theta_1 B + \cdots + \theta_q B^q)\varepsilon_t, \qquad (8.4)$$

or equivalently as

$$(1-\phi_1 B - \cdots - \phi_p B^p)(1-B)^d (y_t - \mu t^d/d!) = (1+\theta_1 B + \cdots + \theta_q B^q)\varepsilon_t, \quad (8.5)$$

where $c = \mu(1-\phi_1 - \cdots - \phi_p)$ and μ is the mean of $(1-B)^d y_t$. R uses the parameterisation of Equation (8.5).

Thus, the inclusion of a constant in a non-stationary ARIMA model is equivalent to inducing a polynomial trend of order d in the forecast function. (If the constant is omitted, the forecast function includes a polynomial trend of order $d-1$.) When $d=0$, we have the special case that μ is the mean of y_t.

By default, the `Arima()` function sets $c=\mu=0$ when $d>0$ and provides an estimate of μ when $d=0$. It will be close to the sample mean of the time series, but usually not identical to it as the sample mean is not the maximum likelihood estimate when $p+q>0$.

The argument `include.mean` only has an effect when $d=0$ and is `TRUE` by default. Setting `include.mean=FALSE` will force $\mu=c=0$.

The argument `include.drift` allows $\mu \neq 0$ when $d=1$. For $d>1$, no constant is allowed as a quadratic or higher order trend is particularly dangerous when forecasting. The parameter μ is called the "drift" in the R output when $d=1$.

There is also an argument `include.constant` which, if `TRUE`, will set `include.mean=TRUE` if $d=0$ and `include.drift=TRUE` when $d=1$. If `include.constant=FALSE`, both `include.mean` and `include.drift` will be set to `FALSE`. If `include.constant` is used, the values of `include.mean=TRUE` and `include.drift=TRUE` are ignored.

The `auto.arima()` function automates the inclusion of a constant. By default, for $d=0$ or $d=1$, a constant will be included if it improves the AICc value; for $d>1$ the constant is always omitted. If `allowdrift=FALSE` is specified, then the constant is only allowed when $d=0$.

Plotting the characteristic roots

(This is a more advanced section and can be skipped if desired.)

We can re-write Equation (8.4) as

$$\phi(B)(1-B)^d y_t = c + \theta(B)\varepsilon_t$$

where $\phi(B) = (1 - \phi_1 B - \cdots - \phi_p B^p)$ is a pth order polynomial in B and $\theta(B) = (1 + \theta_1 B + \cdots + \theta_q B^q)$ is a qth order polynomial in B.

The stationarity conditions for the model are that the p complex roots of $\phi(B)$ lie outside the unit circle, and the invertibility conditions are that the q complex roots of $\theta(B)$ lie outside the unit circle. So we can see whether the model is close to invertibility or stationarity by a plot of the roots in relation to the complex unit circle.

It is easier to plot the inverse roots instead, as they should all lie *within* the unit circle. This is easily done in R. For the ARIMA(3,1,1) model fitted to the seasonally adjusted electrical equipment index, we obtain Figure 8.16.

```
autoplot(fit)
```

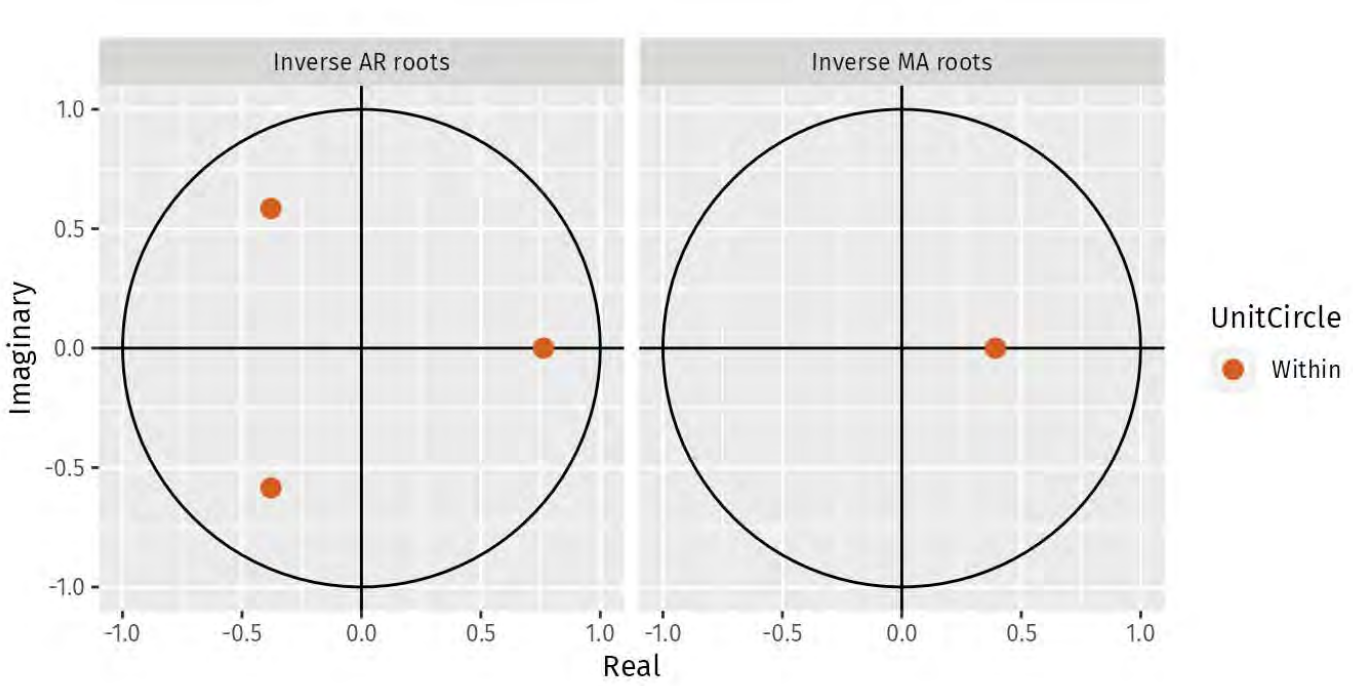


Figure 8.16: Inverse characteristic roots for the ARIMA(3,1,1) model fitted to the seasonally adjusted electrical equipment index.

The three red dots in the left hand plot correspond to the roots of the polynomials $\phi(B)$, while the red dot in the right hand plot corresponds to the root of $\theta(B)$. They are all inside the unit circle, as we would expect because R ensures the fitted model is both stationary and invertible. Any roots close to the unit circle may be numerically unstable, and the corresponding model will not be good for forecasting.

The `Arima()` function will never return a model with inverse roots outside the unit circle. The `auto.arima()` function is even stricter, and will not select a model with roots close to the unit circle either.

Bibliography

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, *27*(1), 1–22. [DOI]

8.8 Forecasting

Point forecasts

Although we have calculated forecasts from the ARIMA models in our examples, we have not yet explained how they are obtained. Point forecasts can be calculated using the following three steps.

1. Expand the ARIMA equation so that y_t is on the left hand side and all other terms are on the right.
2. Rewrite the equation by replacing t with $T+h$.
3. On the right hand side of the equation, replace future observations with their forecasts, future errors with zero, and past errors with the corresponding residuals.

Beginning with $h=1$, these steps are then repeated for $h=2,3,\ldots$ until all forecasts have been calculated.

The procedure is most easily understood via an example. We will illustrate it using the ARIMA(3,1,1) model fitted in the previous section. The model can be written as follows:

$$(1-\hat{\phi}_1B-\hat{\phi}_2B^2-\hat{\phi}_3B^3)(1-B)y_t=(1+\hat{\theta}_1B)\varepsilon_t,$$

where $\hat{\phi}_1=0.0044$, $\hat{\phi}_2=0.0916$, $\hat{\phi}_3=0.3698$ and $\hat{\theta}_1=-0.3921$. Then we expand the left hand side to obtain

$$\left[1-(1+\hat{\phi}_1)B+(\hat{\phi}_1-\hat{\phi}_2)B^2+(\hat{\phi}_2-\hat{\phi}_3)B^3+\hat{\phi}_3B^4\right]y_t=(1+\hat{\theta}_1B)\varepsilon_t,$$

and applying the backshift operator gives

$$y_t-(1+\hat{\phi}_1)y_{t-1}+(\hat{\phi}_1-\hat{\phi}_2)y_{t-2}+(\hat{\phi}_2-\hat{\phi}_3)y_{t-3}+\hat{\phi}_3y_{t-4}=\varepsilon_t+\hat{\theta}_1\varepsilon_{t-1}.$$

Finally, we move all terms other than y_t to the right hand side:

$$y_t=(1+\hat{\phi}_1)y_{t-1}-(\hat{\phi}_1-\hat{\phi}_2)y_{t-2}-(\hat{\phi}_2-\hat{\phi}_3)y_{t-3}-\hat{\phi}_3y_{t-4}+\varepsilon_t+\hat{\theta}_1\varepsilon_{t-1}. \quad (8.6)$$

This completes the first step. While the equation now looks like an ARIMA(4,0,1), it is still the same ARIMA(3,1,1) model we started with. It cannot be considered an ARIMA(4,0,1) because the coefficients do not satisfy the stationarity conditions.

For the second step, we replace t with $T+1$ in (8.6):

$$y_{T+1} = (1+\hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \varepsilon_{T+1} + \hat{\theta}_1\varepsilon_T.$$

Assuming we have observations up to time T, all values on the right hand side are known except for ε_{T+1}, which we replace with zero, and ε_T, which we replace with the last observed residual e_T:

$$\hat{y}_{T+1|T} = (1+\hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \hat{\theta}_1 e_T.$$

A forecast of y_{T+2} is obtained by replacing t with $T+2$ in (8.6) . All values on the right hand side will be known at time T except y_{T+1} which we replace with $\hat{y}_{T+1|T}$, and ε_{T+2} and ε_{T+1}, both of which we replace with zero:

$$\hat{y}_{T+2|T} = (1+\hat{\phi}_1)\hat{y}_{T+1|T} - (\hat{\phi}_1 - \hat{\phi}_2)y_T - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-1} - \hat{\phi}_3 y_{T-2}.$$

The process continues in this manner for all future time periods. In this way, any number of point forecasts can be obtained.

Prediction intervals

The calculation of ARIMA prediction intervals is more difficult, and the details are largely beyond the scope of this book. We will only give some simple examples.

The first prediction interval is easy to calculate. If $\hat{\sigma}$ is the standard deviation of the residuals, then a 95% prediction interval is given by $\hat{y}_{T+1|T} \pm 1.96\hat{\sigma}$. This result is true for all ARIMA models regardless of their parameters and orders.

Multi-step prediction intervals for ARIMA(0,0,q) models are relatively easy to calculate. We can write the model as

$$y_t = \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}.$$

Then, the estimated forecast variance can be written as

$$\hat{\sigma}_h = \hat{\sigma}^2 \left[1 + \sum_{i=1}^{h-1} \hat{\theta}_i^2\right], \qquad \text{for } h = 2, 3, \ldots,$$

where $\hat{\theta}_i = 0$ for $i > q$, and a 95% prediction interval is given by $\hat{y}_{T+h|T} \pm 1.96\sqrt{\hat{\sigma}_h}$.

In Section 8.4, we showed that an AR(1) model can be written as an MA(∞) model. Using this equivalence, the above result for MA(q) models can also be used to obtain prediction intervals for AR(1) models.

More general results, and other special cases of multi-step prediction intervals for an ARIMA(p,d,q) model, are given in more advanced textbooks such as Brockwell & Davis (2016).

The prediction intervals for ARIMA models are based on assumptions that the residuals are uncorrelated and normally distributed. If either of these assumptions does not hold, then the prediction intervals may be incorrect. For this reason, always plot the ACF and histogram of the residuals to check the assumptions before producing prediction intervals.

In general, prediction intervals from ARIMA models increase as the forecast horizon increases. For stationary models (i.e., with $d = 0$) they will converge, so that prediction intervals for long horizons are all essentially the same. For $d \geq 1$, the prediction intervals will continue to grow into the future.

As with most prediction interval calculations, ARIMA-based intervals tend to be too narrow. This occurs because only the variation in the errors has been accounted for. There is also variation in the parameter estimates, and in the model order, that has not been included in the calculation. In addition, the calculation assumes that the historical patterns that have been modelled will continue into the forecast period.

Bibliography

Brockwell, P. J., & Davis, R. A. (2016). *Introduction to time series and forecasting* (3rd ed). New York, USA: Springer. [Amazon]

8.9 Seasonal ARIMA models

So far, we have restricted our attention to non-seasonal data and non-seasonal ARIMA models. However, ARIMA models are also capable of modelling a wide range of seasonal data.

A seasonal ARIMA model is formed by including additional seasonal terms in the ARIMA models we have seen so far. It is written as follows:

$$\text{ARIMA} \quad \underbrace{(p,d,q)}_{\uparrow} \quad \underbrace{(P,D,Q)_m}_{\uparrow}$$

Non-seasonal part of the model — Seasonal part of of the model

where $m =$ number of observations per year. We use uppercase notation for the seasonal parts of the model, and lowercase notation for the non-seasonal parts of the model.

The seasonal part of the model consists of terms that are similar to the non-seasonal components of the model, but involve backshifts of the seasonal period. For example, an ARIMA(1,1,1)(1,1,1)$_4$ model (without a constant) is for quarterly data ($m = 4$), and can be written as

$$(1-\phi_1 B)\,(1-\Phi_1 B^4)(1-B)(1-B^4)y_t = (1+\theta_1 B)\,(1+\Theta_1 B^4)\varepsilon_t.$$

The additional seasonal terms are simply multiplied by the non-seasonal terms.

ACF/PACF

The seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF and ACF. For example, an ARIMA(0,0,0)(0,0,1)$_{12}$ model will show:

- a spike at lag 12 in the ACF but no other significant spikes;
- exponential decay in the seasonal lags of the PACF (i.e., at lags 12, 24, 36, ...).

Similarly, an ARIMA(0,0,0)(1,0,0)$_{12}$ model will show:

- exponential decay in the seasonal lags of the ACF;
- a single significant spike at lag 12 in the PACF.

In considering the appropriate seasonal orders for a seasonal ARIMA model, restrict attention to the seasonal lags.

The modelling procedure is almost the same as for non-seasonal data, except that we need to select seasonal AR and MA terms as well as the non-seasonal components of the model. The process is best illustrated via examples.

Example: European quarterly retail trade

We will describe the seasonal ARIMA modelling procedure using quarterly European retail trade data from 1996 to 2011. The data are plotted in Figure 8.17.

```
autoplot(euretail) + ylab("Retail index") + xlab("Year")
```

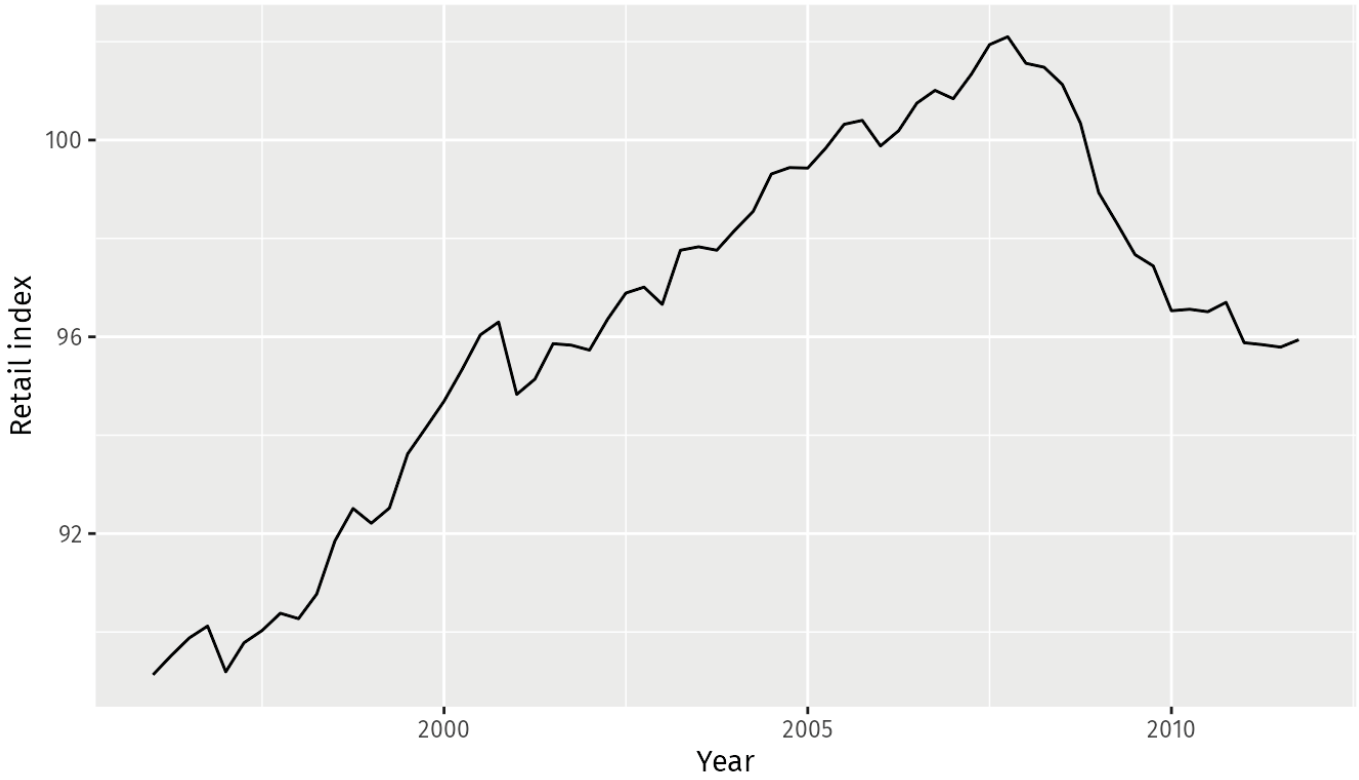


Figure 8.17: Quarterly retail trade index in the Euro area (17 countries), 1996–2011, covering wholesale and retail trade, and the repair of motor vehicles and motorcycles. (Index: 2005 = 100).

The data are clearly non-stationary, with some seasonality, so we will first take a seasonal difference. The seasonally differenced data are shown in Figure 8.18. These also appear to be non-stationary, so we take an additional first difference, shown in Figure 8.19.

```
euretail %>% diff(lag=4) %>% ggtsdisplay()
```

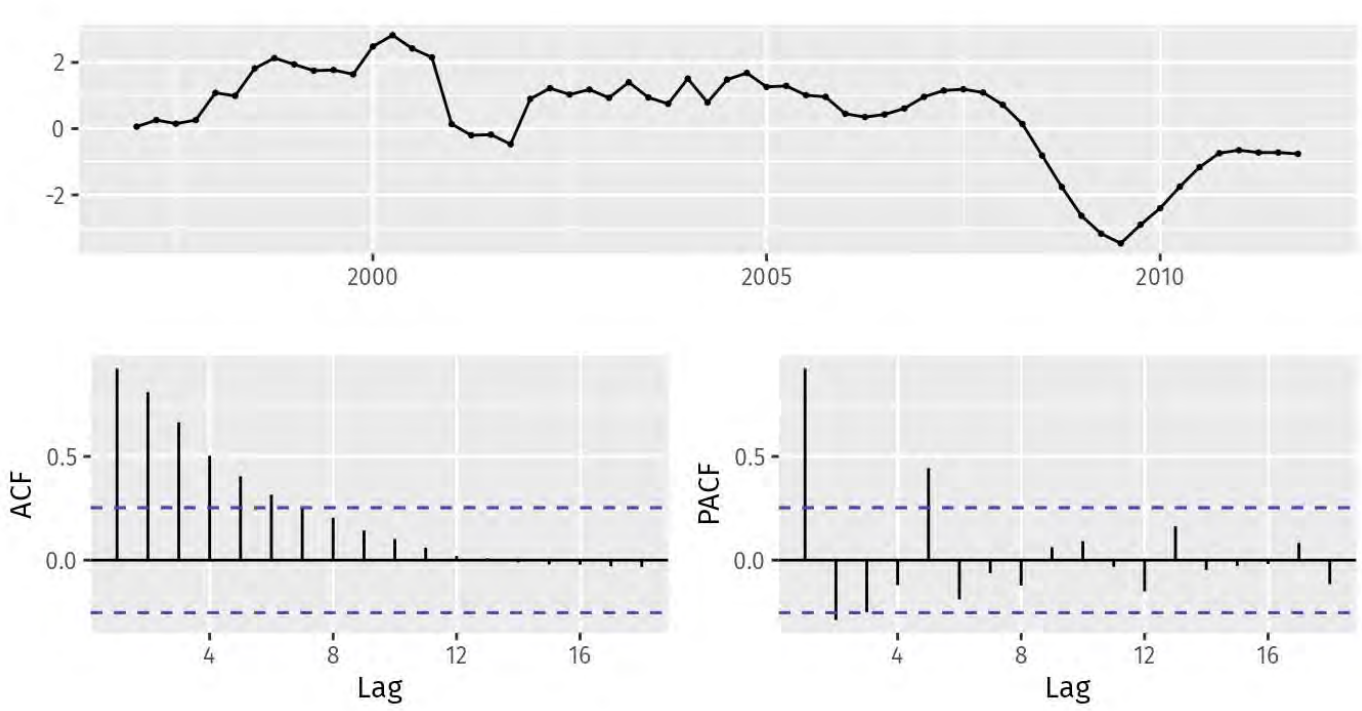


Figure 8.18: Seasonally differenced European retail trade index.

```
euretail %>% diff(lag=4) %>% diff() %>% ggtsdisplay()
```

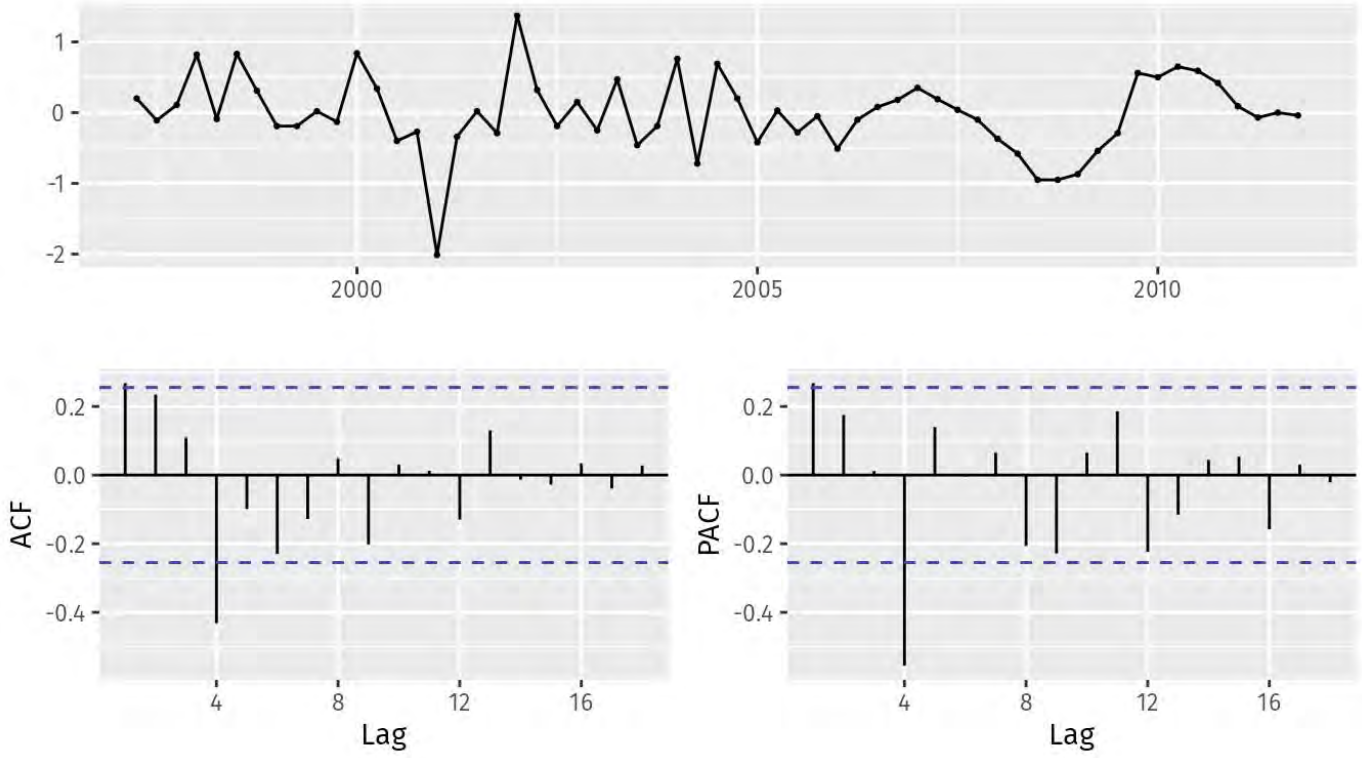


Figure 8.19: Double differenced European retail trade index.

Our aim now is to find an appropriate ARIMA model based on the ACF and PACF shown in Figure 8.19. The significant spike at lag 1 in the ACF suggests a non-seasonal MA(1) component, and the significant spike at lag 4 in the ACF suggests a seasonal MA(1) component. Consequently, we begin with an ARIMA(0,1,1)$(0,1,1)_4$ model, indicating a first and seasonal difference, and non-seasonal and seasonal MA(1) components. The residuals for the fitted model are shown in Figure 8.20. (By analogous logic applied to the PACF, we could also have started with an ARIMA(1,1,0)$(1,1,0)_4$ model.)

```
euretail %>%
  Arima(order=c(0,1,1), seasonal=c(0,1,1)) %>%
  residuals() %>% ggtsdisplay()
```

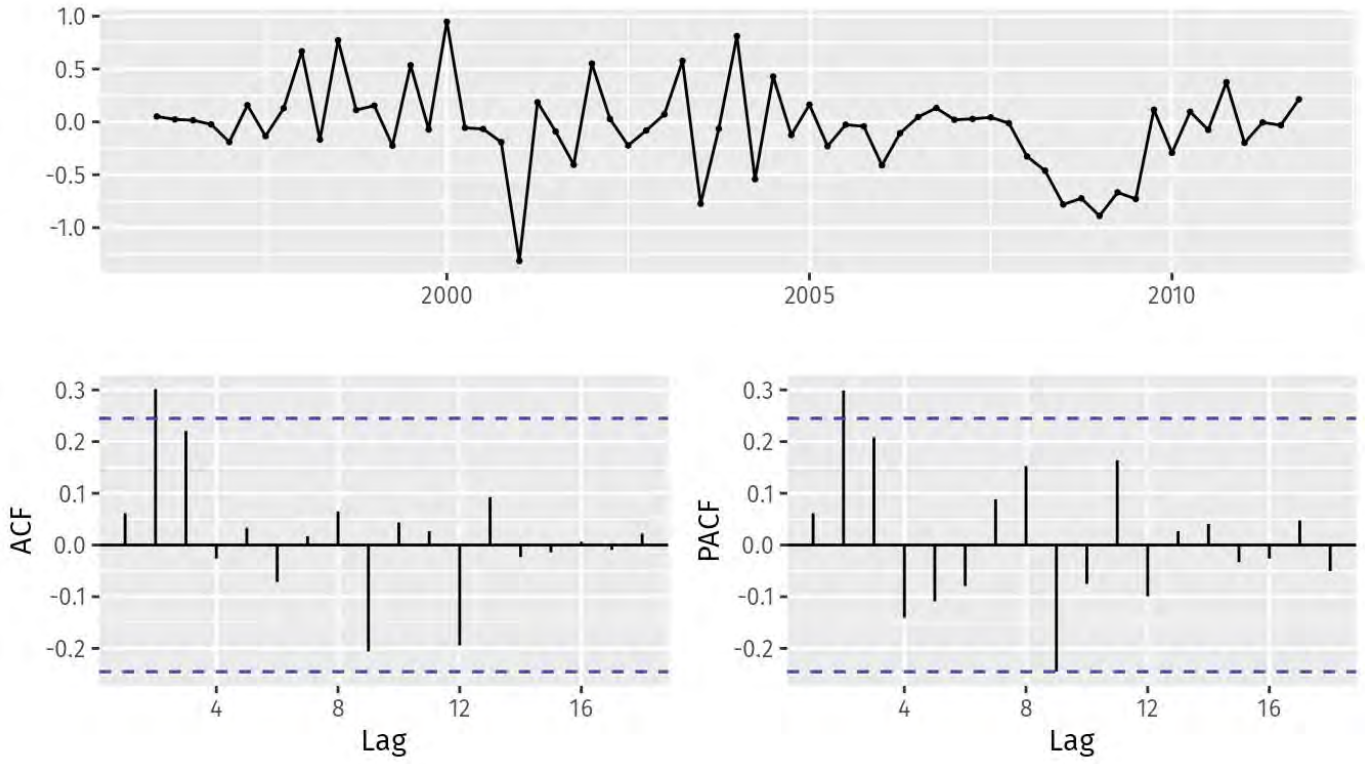


Figure 8.20: Residuals from the fitted ARIMA(0,1,1)$(0,1,1)_4$ model for the European retail trade index data.

Both the ACF and PACF show significant spikes at lag 2, and almost significant spikes at lag 3, indicating that some additional non-seasonal terms need to be included in the model. The AICc of the ARIMA(0,1,2)$(0,1,1)_4$ model is 74.36, while that for the ARIMA(0,1,3)$(0,1,1)_4$ model is 68.53. We tried other models with AR terms as well, but none that gave a smaller AICc value. Consequently, we choose the ARIMA(0,1,3)$(0,1,1)_4$ model. Its residuals are plotted in Figure 8.21. All the spikes are now within the significance limits, so the residuals appear to be white noise. The Ljung-Box test also shows that the residuals have no remaining autocorrelations.

```
fit3 <- Arima(euretail, order=c(0,1,3), seasonal=c(0,1,1))
checkresiduals(fit3)
```

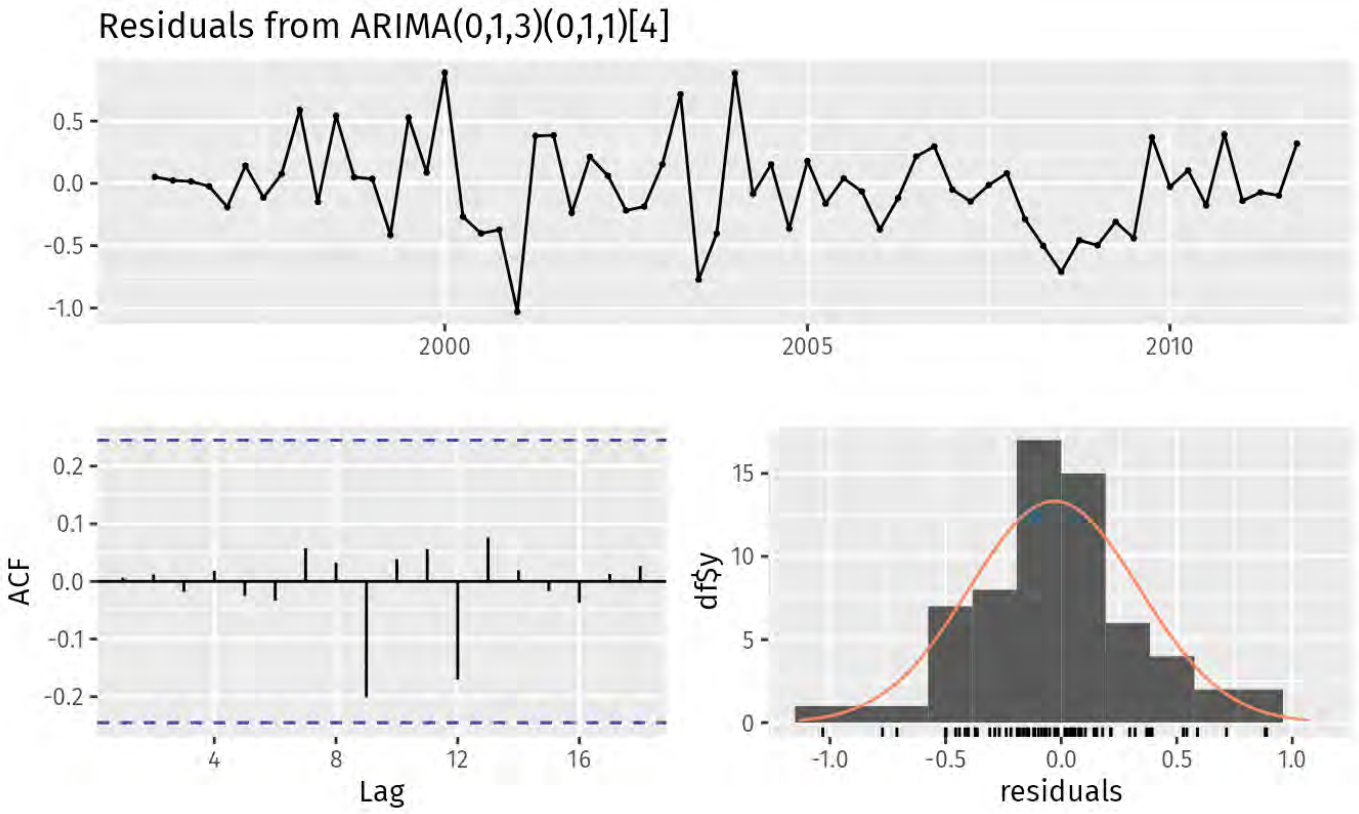


Figure 8.21: Residuals from the fitted ARIMA(0,1,3)(0,1,1)$_4$ model for the European retail trade index data.

```
#>
#>  Ljung-Box test
#>
#> data:  Residuals from ARIMA(0,1,3)(0,1,1)[4]
#> Q* = 0.51, df = 4, p-value = 1
#>
#> Model df: 4.   Total lags used: 8
```

Thus, we now have a seasonal ARIMA model that passes the required checks and is ready for forecasting. Forecasts from the model for the next three years are shown in Figure 8.22. The forecasts follow the recent trend in the data, because of the double differencing. The large and rapidly increasing prediction intervals show that the retail trade index could start increasing or decreasing at any time — while the point forecasts trend downwards, the prediction intervals allow for the data to trend upwards during the forecast period.

```
fit3 %>% forecast(h=12) %>% autoplot()
```

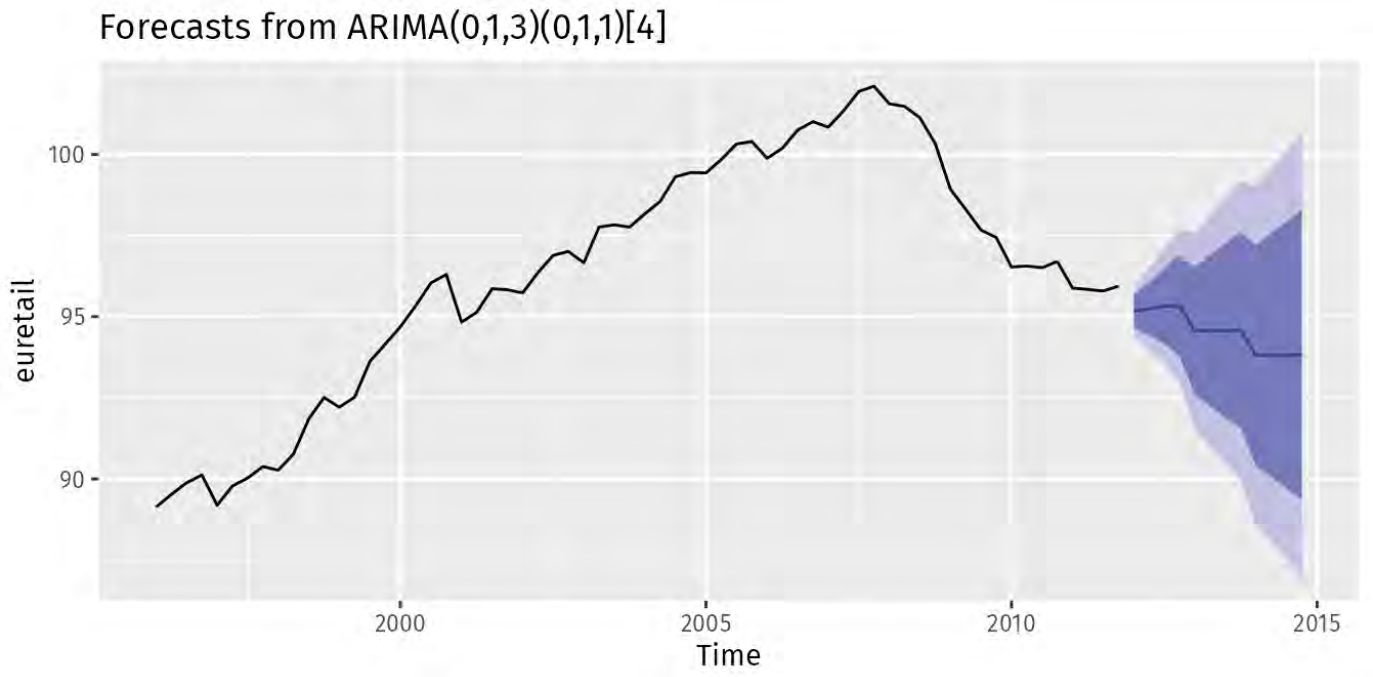


Figure 8.22: Forecasts of the European retail trade index data using the ARIMA(0,1,3)(0,1,1)$_4$ model. 80% and 95% prediction intervals are shown.

We could have used `auto.arima()` to do most of this work for us. It would have given the same result.

```
auto.arima(euretail)
#> Series: euretail
#> ARIMA(0,1,3)(0,1,1)[4]
#>
#> Coefficients:
#>          ma1    ma2    ma3    sma1
#>       0.263  0.369  0.420  -0.664
#> s.e.  0.124  0.126  0.129   0.155
#>
#> sigma^2 = 0.156:  log likelihood = -28.63
#> AIC=67.26   AICc=68.39   BIC=77.65
```

The `auto.arima()` function uses `nsdiffs()` to determine D (the number of seasonal differences to use), and `ndiffs()` to determine d (the number of ordinary differences to use). The selection of the other model parameters (p, q, P and Q) are all determined by minimizing the AICc, as with non-seasonal ARIMA models.

Example: Corticosteroid drug sales in Australia

Our second example is more difficult. We will try to forecast monthly corticosteroid drug sales in Australia. These are known as H02 drugs under the Anatomical Therapeutic Chemical classification scheme.

```
lh02 <- log(h02)
cbind("H02 sales (million scripts)" = h02,
      "Log H02 sales"=lh02) %>%
  autoplot(facets=TRUE) + xlab("Year") + ylab("")
```

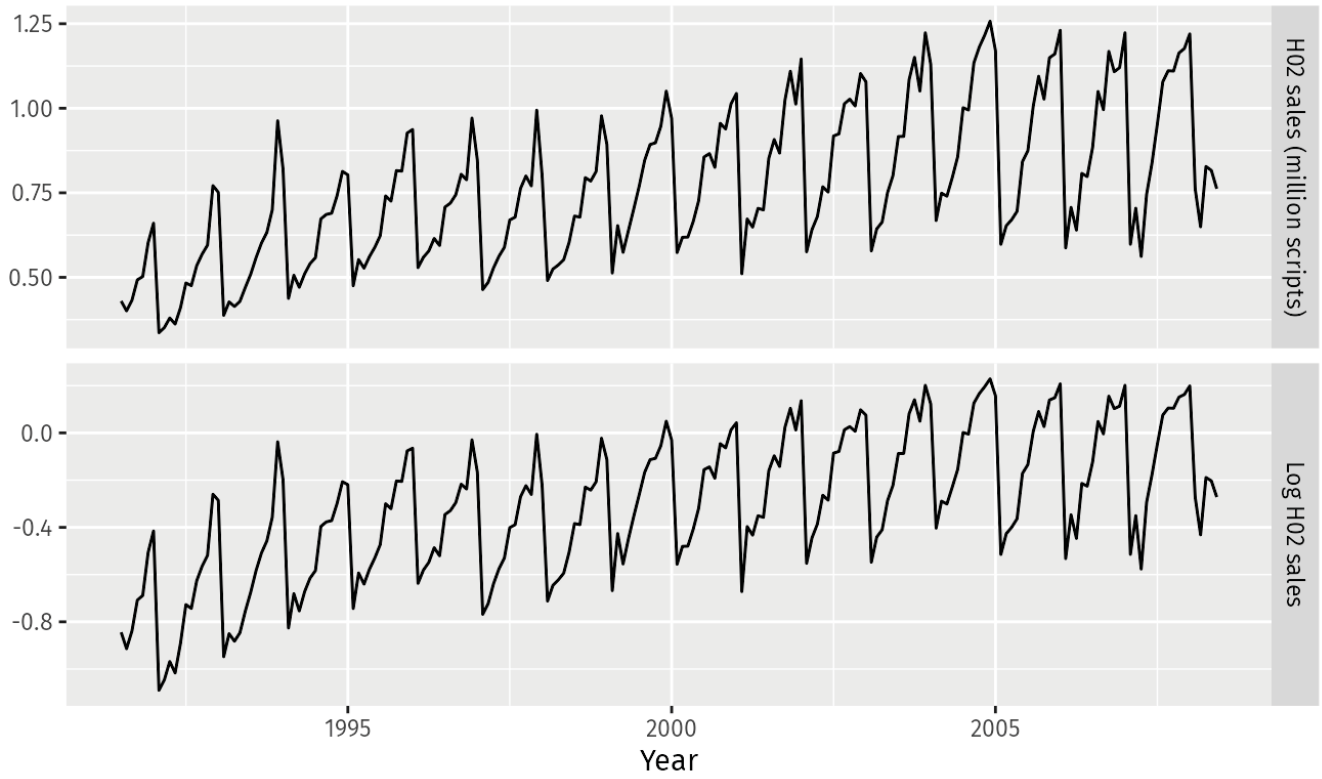


Figure 8.23: Corticosteroid drug sales in Australia (in millions of scripts per month). Logged data shown in bottom panel.

Data from July 1991 to June 2008 are plotted in Figure 8.23. There is a small increase in the variance with the level, so we take logarithms to stabilise the variance.

The data are strongly seasonal and obviously non-stationary, so seasonal differencing will be used. The seasonally differenced data are shown in Figure 8.24. It is not clear at this point whether we should do another difference or not. We decide not to, but the choice is not obvious.

The last few observations appear to be different (more variable) from the earlier data. This may be due to the fact that data are sometimes revised when earlier sales are reported late.

```
lh02 %>% diff(lag=12) %>%
  ggtsdisplay(xlab="Year",
    main="Seasonally differenced H02 scripts")
```


Figure 8.24: Seasonally differenced corticosteroid drug sales in Australia (in millions of scripts per month).

In the plots of the seasonally differenced data, there are spikes in the PACF at lags 12 and 24, but nothing at seasonal lags in the ACF. This may be suggestive of a seasonal AR(2) term. In the non-seasonal lags, there are three significant spikes in the PACF, suggesting a possible AR(3) term. The pattern in the ACF is not indicative of any simple model.

Consequently, this initial analysis suggests that a possible model for these data is an ARIMA(3,0,0)(2,1,0)$_{12}$. We fit this model, along with some variations on it, and compute the AICc values shown in the following table.

Model	AICc
ARIMA(3,0,1)(0,1,2)$_{12}$	-485.5
ARIMA(3,0,1)(1,1,1)$_{12}$	-484.2
ARIMA(3,0,1)(0,1,1)$_{12}$	-483.7
ARIMA(3,0,1)(2,1,0)$_{12}$	-476.3
ARIMA(3,0,0)(2,1,0)$_{12}$	-475.1
ARIMA(3,0,2)(2,1,0)$_{12}$	-474.9
ARIMA(3,0,1)(1,1,0)$_{12}$	-463.4

Of these models, the best is the ARIMA(3,0,1)(0,1,2)$_{12}$ model (i.e., it has the smallest AICc value).

```
(fit <- Arima(h02, order=c(3,0,1), seasonal=c(0,1,2),
  lambda=0))
#> Series: h02
#> ARIMA(3,0,1)(0,1,2)[12]
#> Box Cox transformation: lambda= 0
#>
#> Coefficients:
#>          ar1    ar2    ar3    ma1    sma1    sma2
#>       -0.160  0.548  0.568  0.383  -0.522  -0.177
#> s.e.   0.164  0.088  0.094  0.190   0.086   0.087
#>
#> sigma^2 = 0.00428:  log likelihood = 250
#> AIC=-486.1   AICc=-485.5   BIC=-463.3

checkresiduals(fit, lag=36)
```

Figure 8.25: Residuals from the ARIMA(3,0,1)(0,1,2)$_{12}$ model applied to the H02 monthly script sales data.

```
#>
#>  Ljung-Box test
#>
#> data:  Residuals from ARIMA(3,0,1)(0,1,2)[12]
#> Q* = 51, df = 30, p-value = 0.01
#>
#> Model df: 6.   Total lags used: 36
```

The residuals from this model are shown in Figure 8.25. There are a few significant spikes in the ACF, and the model fails the Ljung-Box test. The model can still be used for forecasting, but the prediction intervals may not be accurate due to the correlated residuals.

Next we will try using the automatic ARIMA algorithm. Running `auto.arima()` with all arguments left at their default values led to an ARIMA(2,1,1)(0,1,2)$_{12}$ model. However, the model still fails the Ljung-Box test for 36 lags. Sometimes it is just not possible to find a model that passes all of the tests.

Test set evaluation:

We will compare some of the models fitted so far using a test set consisting of the last two years of data. Thus, we fit the models using data from July 1991 to June 2006, and forecast the script sales for July 2006 – June 2008. The results are summarised in Table 8.2.

Table 8.2: RMSE values for various ARIMA models applied to the H02 monthly script sales data.

Model	RMSE
ARIMA(3,0,1)(0,1,2)$_{12}$	0.0622
ARIMA(3,0,1)(1,1,1)$_{12}$	0.0630
ARIMA(2,1,3)(0,1,1)$_{12}$	0.0634
ARIMA(2,1,1)(0,1,2)$_{12}$	0.0634
ARIMA(2,1,2)(0,1,2)$_{12}$	0.0635
ARIMA(3,0,3)(0,1,1)$_{12}$	0.0637
ARIMA(3,0,1)(0,1,1)$_{12}$	0.0644
ARIMA(3,0,2)(0,1,1)$_{12}$	0.0644
ARIMA(3,0,2)(2,1,0)$_{12}$	0.0645
ARIMA(3,0,1)(2,1,0)$_{12}$	0.0646
ARIMA(4,0,2)(0,1,1)$_{12}$	0.0648
ARIMA(4,0,3)(0,1,1)$_{12}$	0.0648
ARIMA(3,0,0)(2,1,0)$_{12}$	0.0661
ARIMA(3,0,1)(1,1,0)$_{12}$	0.0679

The models chosen manually and with `auto.arima()` are both in the top four models based on their RMSE values.

When models are compared using AICc values, it is important that all models have the same orders of differencing. However, when comparing models using a test set, it does not matter how the forecasts were produced — the comparisons are always valid. Consequently, in the table above, we can include some models with only seasonal differencing and some models with both first and seasonal differencing, while in the earlier table containing AICc values, we only compared models with seasonal differencing but no first differencing.

None of the models considered here pass all of the residual tests. In practice, we would normally use the best model we could find, even if it did not pass all of the tests.

Forecasts from the ARIMA(3,0,1)(0,1,2)$_{12}$ model (which has the lowest RMSE value on the test set, and the best AICc value amongst models with only seasonal differencing) are shown in Figure 8.26.

```
h02 %>%
  Arima(order=c(3,0,1), seasonal=c(0,1,2), lambda=0) %>%
  forecast() %>%
  autoplot() +
    ylab("H02 sales (million scripts)") + xlab("Year")
```

Figure 8.26: Forecasts from the ARIMA(3,0,1)(0,1,2)$_{12}$ model applied to the H02 monthly script sales data.

8.10 ARIMA vs ETS

It is a commonly held myth that ARIMA models are more general than exponential smoothing. While linear exponential smoothing models are all special cases of ARIMA models, the non-linear exponential smoothing models have no equivalent ARIMA counterparts. On the other hand, there are also many ARIMA models that have no exponential smoothing counterparts. In particular, all ETS models are non-stationary, while some ARIMA models are stationary.

The ETS models with seasonality or non-damped trend or both have two unit roots (i.e., they need two levels of differencing to make them stationary). All other ETS models have one unit root (they need one level of differencing to make them stationary).

Table 8.3 gives the equivalence relationships for the two classes of models. For the seasonal models, the ARIMA parameters have a large number of restrictions.

Table 8.3: Equivalence relationships between ETS and ARIMA models.

ETS model	ARIMA model	Parameters
ETS(A,N,N)	ARIMA(0,1,1)	$\theta_1 = \alpha - 1$
ETS(A,A,N)	ARIMA(0,2,2)	$\theta_1 = \alpha + \beta - 2$ $\theta_2 = 1 - \alpha$
ETS(A,A_d,N)	ARIMA(1,1,2)	$\phi_1 = \phi$ $\theta_1 = \alpha + \phi\beta - 1 - \phi$ $\theta_2 = (1 - \alpha)\phi$
ETS(A,N,A)	ARIMA(0,1,m)(0,1,0)$_m$	
ETS(A,A,A)	ARIMA(0,1,$m+1$)(0,1,0)$_m$	
ETS(A,A_d,A)	ARIMA(0,1,$m+1$)(0,1,0)$_m$	

The AICc is useful for selecting between models in the same class. For example, we can use it to select an ARIMA model between candidate ARIMA models[18] or an ETS model between candidate ETS models. However, it cannot be used to compare between ETS and ARIMA models because they are in different model classes, and the likelihood is computed in different ways. The examples below demonstrate selecting between these classes of models.

Example: Comparing `auto.arima()` and `ets()` on non-seasonal data

We can use time series cross-validation to compare an ARIMA model and an ETS model. The code below provides functions that return forecast objects from `auto.arima()` and `ets()` respectively.

```
fets <- function(x, h) {
  forecast(ets(x), h = h)
}
farima <- function(x, h) {
  forecast(auto.arima(x), h=h)
}
```

The returned objects can then be passed into `tsCV()`. Let's consider ARIMA models and ETS models for the `air` data as introduced in Section 7.2 where, `air <- window(ausair, start=1990)`.

```
# Compute CV errors for ETS as e1
e1 <- tsCV(air, fets, h=1)
# Compute CV errors for ARIMA as e2
e2 <- tsCV(air, farima, h=1)
# Find MSE of each model class
mean(e1^2, na.rm=TRUE)
#> [1] 7.864
mean(e2^2, na.rm=TRUE)
#> [1] 9.622
```

In this case the ets model has a lower tsCV statistic based on MSEs. Below we generate and plot forecasts for the next 5 years generated from an ETS model.

```
air %>% ets() %>% forecast() %>% autoplot()
```

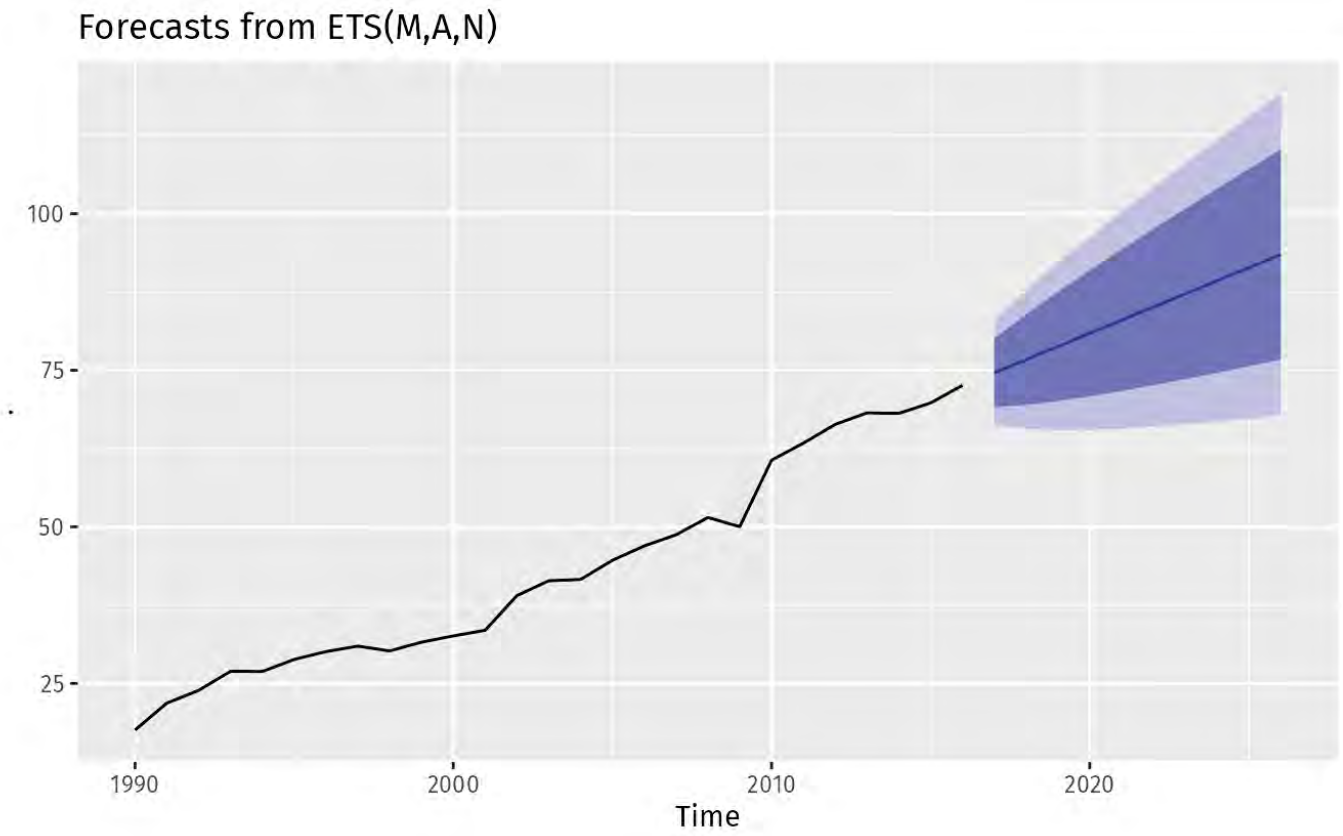


Figure 8.27: Forecasts from an ETS model fitted to monthly totals of air transport passengers in Australia.

Example: Comparing `auto.arima()` and `ets()` on seasonal data

In this case we want to compare seasonal ARIMA and ETS models applied to the quarterly cement production data `qcement`. Because the series is relatively long, we can afford to use a training and a test set rather than time series cross-validation. The advantage is that this is much faster. We create a training set from the beginning of 1988 to the end of 2007 and select an ARIMA and an ETS model using the `auto.arima()` and `ets()` functions.

```
# Consider the qcement data beginning in 1988
cement <- window(qcement, start=1988)
# Use 20 years of the data as the training set
train <- window(cement, end=c(2007,4))
```

The output below shows the ARIMA model selected and estimated by `auto.arima()`. The ARIMA model does well in capturing all the dynamics in the data as the residuals seem to be white noise.

```
(fit.arima <- auto.arima(train))
#> Series: train
#> ARIMA(1,0,1)(2,1,1)[4] with drift
#>
#> Coefficients:
#>          ar1     ma1   sar1    sar2    sma1  drift
#>       0.889  -0.237  0.081  -0.235  -0.898  0.010
#> s.e.  0.084   0.133  0.157   0.139   0.178  0.003
#>
#> sigma^2 = 0.0115:  log likelihood = 61.47
#> AIC=-109   AICc=-107.3   BIC=-92.63
checkresiduals(fit.arima)
```

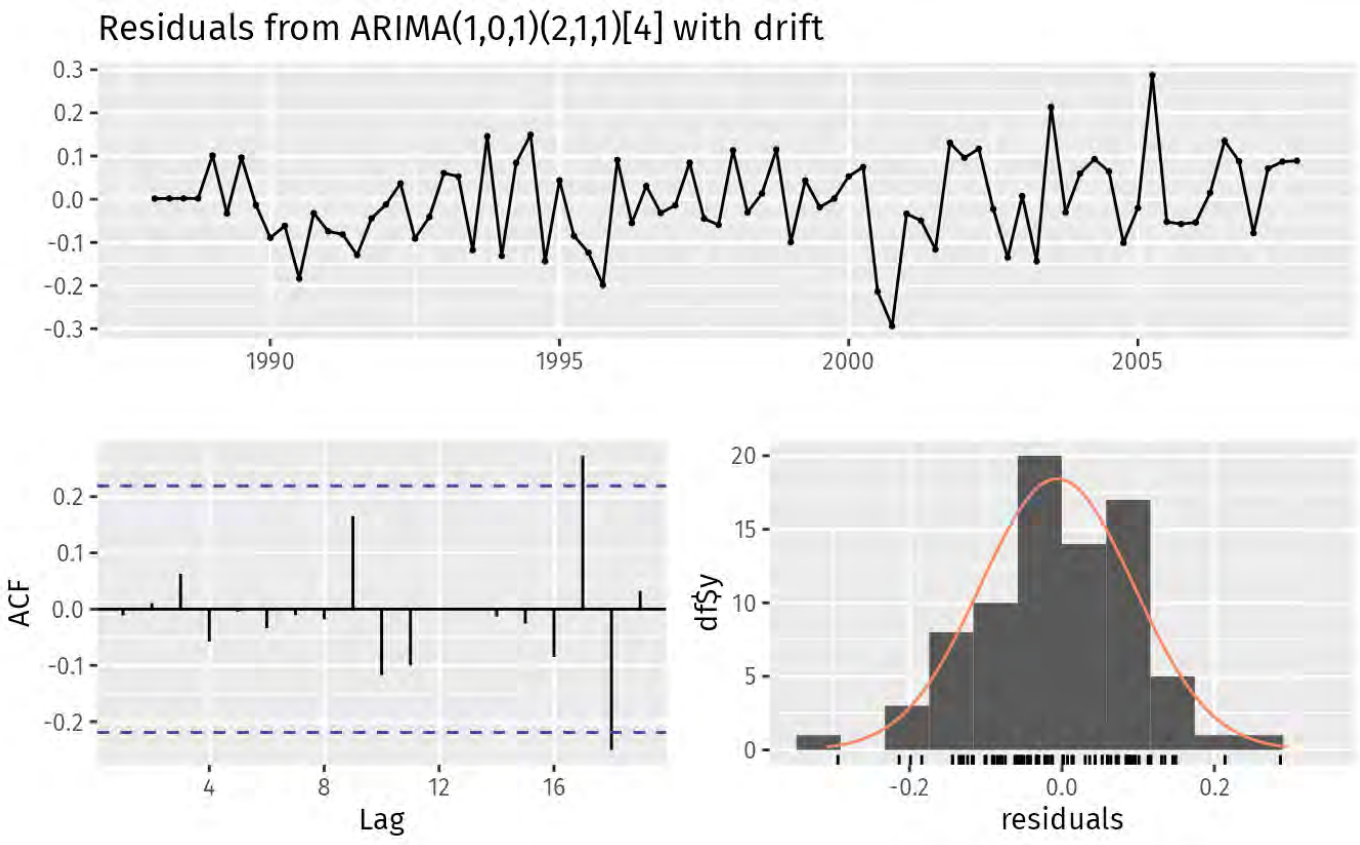


Figure 8.28: Residual diagnostic plots for the ARIMA model fitted to the quarterly cement production training data.

```
#>
#>  Ljung-Box test
#>
#> data:  Residuals from ARIMA(1,0,1)(2,1,1)[4] with drift
#> Q* = 0.78, df = 3, p-value = 0.9
#>
#> Model df: 5.   Total lags used: 8
```

The output below also shows the ETS model selected and estimated by `ets()`. This model also does well in capturing all the dynamics in the data, as the residuals similarly appear to be white noise.

```
(fit.ets <- ets(train))
#> ETS(M,N,M)
#>
#> Call:
#>  ets(y = train)
#>
#>   Smoothing parameters:
#>     alpha = 0.7341
#>     gamma = 1e-04
#>
#>   Initial states:
#>     l = 1.6439
#>     s = 1.031 1.044 1.01 0.9148
#>
#>   sigma:  0.0581
#>
#>     AIC    AICc     BIC
#> -2.1967 -0.6411 14.4775
checkresiduals(fit.ets)
```

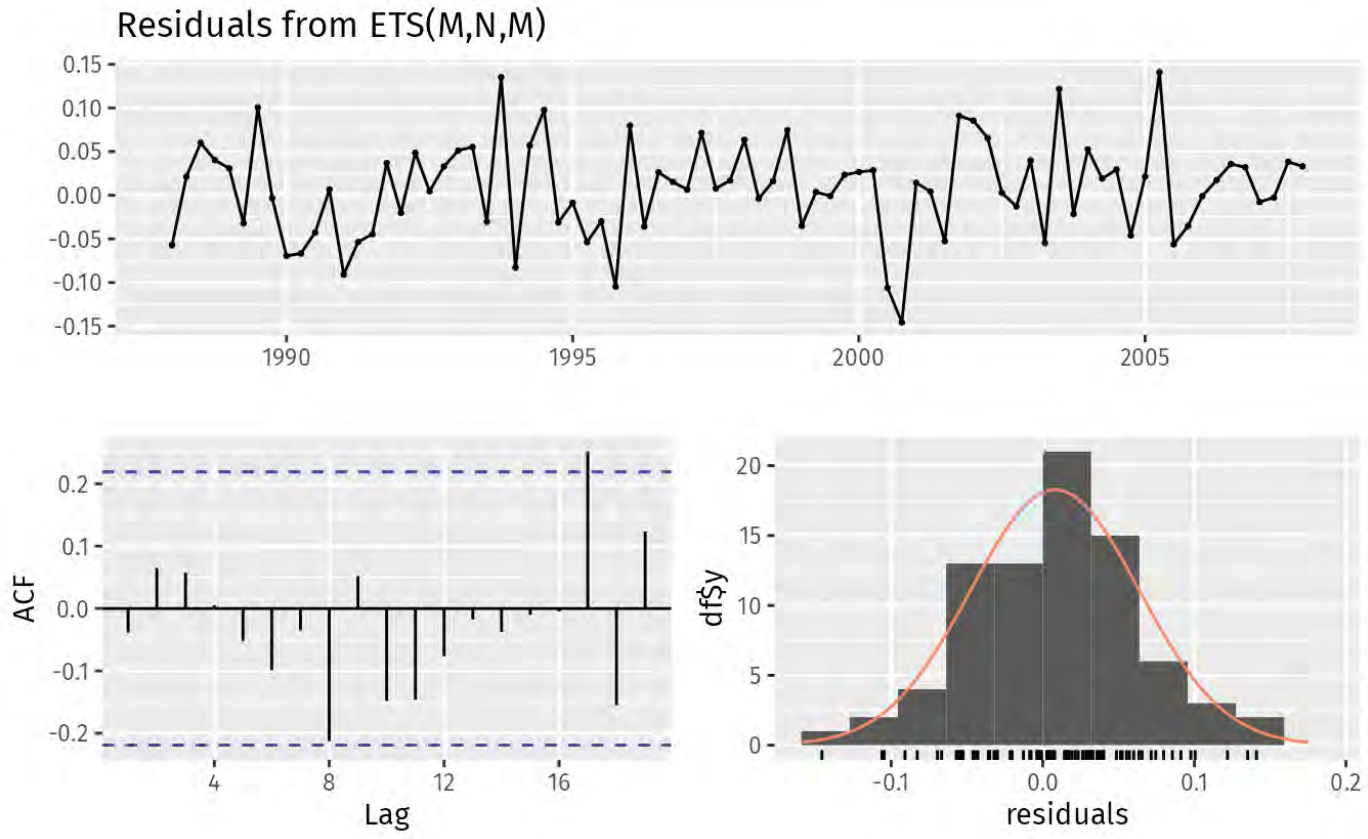


Figure 8.29: Residual diagnostic plots for the ETS model fitted to the quarterly cement production training data.

```
#>
#>  Ljung-Box test
#>
#> data:  Residuals from ETS(M,N,M)
#> Q* = 6.1, df = 8, p-value = 0.6
#>
#> Model df: 0.   Total lags used: 8
```

The output below evaluates the forecasting performance of the two competing models over the test set. In this case the ETS model seems to be the slightly more accurate model based on the test set RMSE, MAPE and MASE.

```
# Generate forecasts and compare accuracy over the test set
a1 <- fit.arima %>% forecast(h = 4*(2013-2007)+1) %>%
  accuracy(qcement)
a1[,c("RMSE","MAE","MAPE","MASE")]
#>                 RMSE     MAE  MAPE   MASE
#> Training set 0.1001 0.07989 4.372 0.5458
#> Test set     0.1996 0.16882 7.719 1.1534
a2 <- fit.ets %>% forecast(h = 4*(2013-2007)+1) %>%
  accuracy(qcement)
a2[,c("RMSE","MAE","MAPE","MASE")]
#>                 RMSE     MAE  MAPE   MASE
#> Training set 0.1022 0.07958 4.372 0.5437
#> Test set     0.1839 0.15395 6.986 1.0518
```

Notice that the ARIMA model fits the training data slightly better than the ETS model, but that the ETS model provides more accurate forecasts on the test set. A good fit to training data is never an indication that the model will forecast well.

Below we generate and plot forecasts from an ETS model for the next 3 years.

```
# Generate forecasts from an ETS model
cement %>% ets() %>% forecast(h=12) %>% autoplot()
```

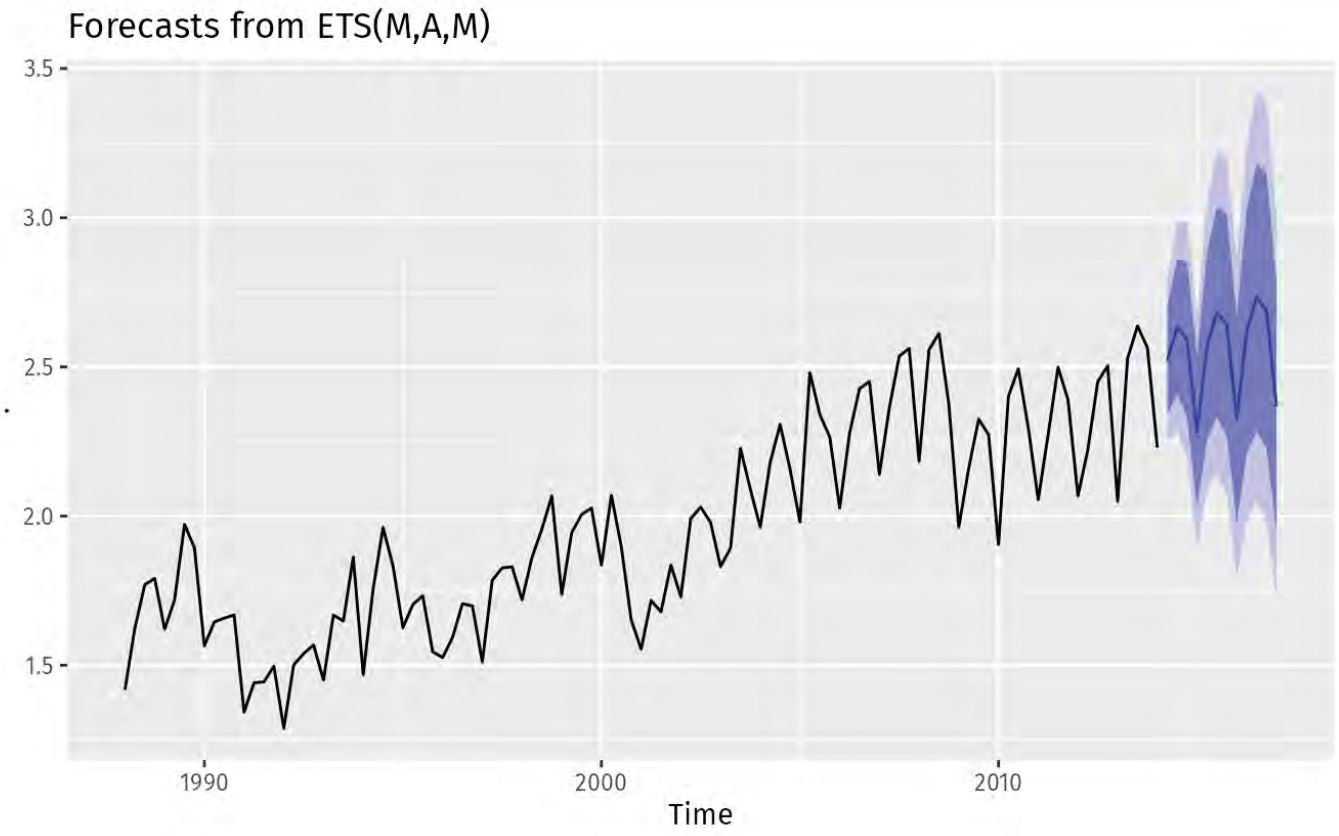


Figure 8.30: Forecasts from an ETS model fitted to all of the available quarterly cement production data.

18. As already noted, comparing information criteria is only valid for ARIMA models of the same orders of differencing.↩

8.11 Exercises

1. Figure 8.31 shows the ACFs for 36 random numbers, 360 random numbers and 1,000 random numbers.

   a. Explain the differences among these figures. Do they all indicate that the data are white noise?

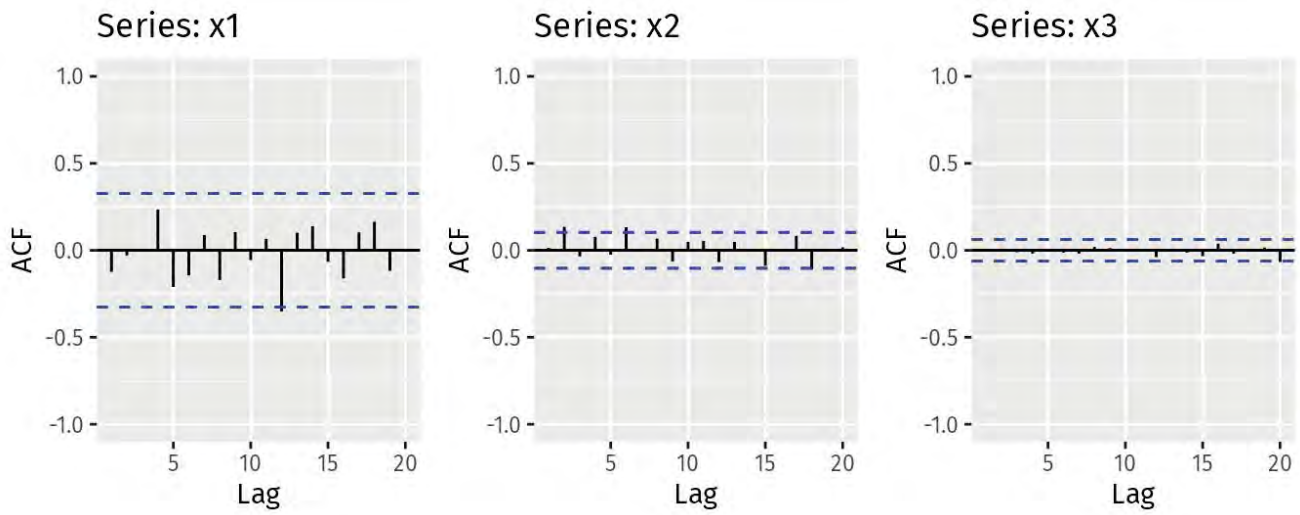


Figure 8.31: Left: ACF for a white noise series of 36 numbers. Middle: ACF for a white noise series of 360 numbers. Right: ACF for a white noise series of 1,000 numbers.

   b. Why are the critical values at different distances from the mean of zero? Why are the autocorrelations different in each figure when they each refer to white noise?

2. A classic example of a non-stationary series is the daily closing IBM stock price series (data set `ibmclose`). Use R to plot the daily closing prices for IBM stock and the ACF and PACF. Explain how each plot shows that the series is non-stationary and should be differenced.

3. For the following series, find an appropriate Box-Cox transformation and order of differencing in order to obtain stationary data.

   a. `usnetelec`
   b. `usgdp`
   c. `mcopper`
   d. `enplanements`
   e. `visitors`

4. For the `enplanements` data, write down the differences you chose above using backshift operator notation.

5. For your retail data (from Exercise 3 in Section 2.10), find the appropriate order of differencing (after transformation if necessary) to obtain stationary data.

6. Use R to simulate and plot some data from simple ARIMA models.

   a. Use the following R code to generate data from an AR(1) model with $\phi_1 = 0.6$ and $\sigma^2 = 1$. The process starts with $y_1 = 0$.

```
y <- ts(numeric(100))
e <- rnorm(100)
for(i in 2:100)
  y[i] <- 0.6*y[i-1] + e[i]
```

   b. Produce a time plot for the series. How does the plot change as you change ϕ_1 ?

   c. Write your own code to generate data from an MA(1) model with $\theta_1 = 0.6$ and $\sigma^2 = 1$.

   d. Produce a time plot for the series. How does the plot change as you change θ_1 ?

   e. Generate data from an ARMA(1,1) model with $\phi_1 = 0.6$, $\theta_1 = 0.6$ and $\sigma^2 = 1$.

   f. Generate data from an AR(2) model with $\phi_1 = -0.8$, $\phi_2 = 0.3$ and $\sigma^2 = 1$. (Note that these parameters will give a non-stationary series.)

   g. Graph the latter two series and compare them.

7. Consider `wmurders` , the number of women murdered each year (per 100,000 standard population) in the United States.

   a. By studying appropriate graphs of the series in R, find an appropriate ARIMA(p, d, q) model for these data.
   b. Should you include a constant in the model? Explain.
   c. Write this model in terms of the backshift operator.
   d. Fit the model using R and examine the residuals. Is the model satisfactory?
   e. Forecast three times ahead. Check your forecasts by hand to make sure that you know how they have been calculated.
   f. Create a plot of the series with forecasts and prediction intervals for the next three periods shown.

g. Does `auto.arima()` give the same model you have chosen? If not, which model do you think is better?

8. Consider `austa`, the total international visitors to Australia (in millions) for the period 1980-2015.

   a. Use `auto.arima()` to find an appropriate ARIMA model. What model was selected. Check that the residuals look like white noise. Plot forecasts for the next 10 periods.
   b. Plot forecasts from an ARIMA(0,1,1) model with no drift and compare these to part a. Remove the MA term and plot again.
   c. Plot forecasts from an ARIMA(2,1,3) model with drift. Remove the constant and see what happens.
   d. Plot forecasts from an ARIMA(0,0,1) model with a constant. Remove the MA term and plot again.
   e. Plot forecasts from an ARIMA(0,2,1) model with no constant.

9. For the `usgdp` series:

   a. if necessary, find a suitable Box-Cox transformation for the data;
   b. fit a suitable ARIMA model to the transformed data using `auto.arima()`;
   c. try some other plausible models by experimenting with the orders chosen;
   d. choose what you think is the best model and check the residual diagnostics;
   e. produce forecasts of your fitted model. Do the forecasts look reasonable?
   f. compare the results with what you would obtain using `ets()` (with no transformation).

10. Consider `austourists`, the quarterly visitor nights (in millions) spent by international tourists to Australia for the period 1999–2015.

    a. Describe the time plot.
    b. What can you learn from the ACF graph?
    c. What can you learn from the PACF graph?
    d. Produce plots of the seasonally differenced data $(1 - B^4)Y_t$. What model do these graphs suggest?
    e. Does `auto.arima()` give the same model that you chose? If not, which model do you think is better?
    f. Write the model in terms of the backshift operator, then without using the backshift operator.

11. Consider `usmelec`, the total net generation of electricity (in billion kilowatt hours) by the U.S. electric industry (monthly for the period January 1973 – June 2013). In general there are two peaks per year: in mid-summer and mid-winter.

a. Examine the 12-month moving average of this series to see what kind of trend is involved.
b. Do the data need transforming? If so, find a suitable transformation.
c. Are the data stationary? If not, find an appropriate differencing which yields stationary data.
d. Identify a couple of ARIMA models that might be useful in describing the time series. Which of your models is the best according to their AIC values?
e. Estimate the parameters of your best model and do diagnostic testing on the residuals. Do the residuals resemble white noise? If not, try to find another ARIMA model which fits better.
f. Forecast the next 15 years of electricity generation by the U.S. electric industry. Get the latest figures from the EIA to check the accuracy of your forecasts.
g. Eventually, the prediction intervals are so wide that the forecasts are not particularly useful. How many years of forecasts do you think are sufficiently accurate to be usable?

12. For the `mcopper` data:

a. if necessary, find a suitable Box-Cox transformation for the data;
b. fit a suitable ARIMA model to the transformed data using `auto.arima()`;
c. try some other plausible models by experimenting with the orders chosen;
d. choose what you think is the best model and check the residual diagnostics;
e. produce forecasts of your fitted model. Do the forecasts look reasonable?
f. compare the results with what you would obtain using `ets()` (with no transformation).

13. Choose one of the following seasonal time series: `hsales`, `auscafe`, `qauselec`, `qcement`, `qgas`.

a. Do the data need transforming? If so, find a suitable transformation.
b. Are the data stationary? If not, find an appropriate differencing which yields stationary data.
c. Identify a couple of ARIMA models that might be useful in describing the time series. Which of your models is the best according to their AIC values?
d. Estimate the parameters of your best model and do diagnostic testing on the residuals. Do the residuals resemble white noise? If not, try to find another ARIMA model which fits better.
e. Forecast the next 24 months of data using your preferred model.
f. Compare the forecasts obtained using `ets()`.

14. For the same time series you used in the previous exercise, try using a non-seasonal model applied to the seasonally adjusted data obtained from STL. The `stlf()` function will make the calculations easy (with `method="arima"`). Compare the forecasts with those obtained in the previous exercise. Which do you think is the best approach?

15. For your retail time series (Exercise 5 above):

    a. develop an appropriate seasonal ARIMA model;
    b. compare the forecasts with those you obtained in earlier chapters;
    c. Obtain up-to-date retail data from the ABS website (Cat 8501.0, Table 11), and compare your forecasts with the actual numbers. How good were the forecasts from the various models?

16. Consider `sheep`, the sheep population of England and Wales from 1867–1939.

    a. Produce a time plot of the time series.

    b. Assume you decide to fit the following model:

$$y_t = y_{t-1} + \phi_1(y_{t-1} - y_{t-2}) + \phi_2(y_{t-2} - y_{t-3}) + \phi_3(y_{t-3} - y_{t-4}) + \varepsilon_t,$$

    where ε_t is a white noise series. What sort of ARIMA model is this (i.e., what are p, d, and q)?

    c. By examining the ACF and PACF of the differenced data, explain why this model is appropriate.

    d. The last five values of the series are given below:

Year	1935	1936	1937	1938	1939
Millions of sheep	1648	1665	1627	1791	1797

    The estimated parameters are $\phi_1 = 0.42$, $\phi_2 = -0.20$, and $\phi_3 = -0.30$. Without using the `forecast` function, calculate forecasts for the next three years (1940–1942).

    e. Now fit the model in R and obtain the forecasts using `forecast`. How are they different from yours? Why?

17. The annual bituminous coal production in the United States from 1920 to 1968 is in data set `bicoal`.

    a. Produce a time plot of the data.

b. You decide to fit the following model to the series:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \phi_3 y_{t-3} + \phi_4 y_{t-4} + \varepsilon_t$$

where y_t is the coal production in year t and ε_t is a white noise series. What sort of ARIMA model is this (i.e., what are p, d, and q)?

c. Explain why this model was chosen using the ACF and PACF.

d. The last five values of the series are given below.

Year	1964	1965	1966	1967	1968
Millions of tons	467	512	534	552	545

The estimated parameters are $c = 162.00$, $\phi_1 = 0.83$, $\phi_2 = -0.34$, $\phi_3 = 0.55$, and $\phi_4 = -0.38$. Without using the `forecast` function, calculate forecasts for the next three years (1969–1971).

e. Now fit the model in R and obtain the forecasts from the same model. How are they different from yours? Why?

18. Before doing this exercise, you will need to install the **Quandl** package in R using

```
install.packages("Quandl")
```

a. Select a time series from Quandl. Then copy its short URL and import the data using

```
y <- Quandl("?????", api_key="?????", type="ts")
```

(Replace each `?????` with the appropriate values.)

b. Plot graphs of the data, and try to identify an appropriate ARIMA model.

c. Do residual diagnostic checking of your ARIMA model. Are the residuals white noise?

d. Use your chosen ARIMA model to forecast the next four years.

e. Now try to identify an appropriate ETS model.

f. Do residual diagnostic checking of your ETS model. Are the residuals white noise?

g. Use your chosen ETS model to forecast the next four years.

h. Which of the two models do you prefer?

8.12 Further reading

- The classic text which popularised ARIMA modelling was Box & Jenkins (1970). The most recent edition is Box, Jenkins, Reinsel, & Ljung (2015), and it is still an excellent reference for all things ARIMA.
- Brockwell & Davis (2016) provides a good introduction to the mathematical background to the models.
- Peña, Tiao, & Tsay (2001) describes some alternative automatic algorithms to the one used by `auto.arima()`.

Bibliography

Box, G. E. P., & Jenkins, G. M. (1970). *Time series analysis: Forecasting and control*. San Francisco: Holden-Day.

Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: Forecasting and control* (5th ed). Hoboken, New Jersey: John Wiley & Sons. [Amazon]

Brockwell, P. J., & Davis, R. A. (2016). *Introduction to time series and forecasting* (3rd ed). New York, USA: Springer. [Amazon]

Peña, D., Tiao, G. C., & Tsay, R. S. (Eds.). (2001). *A course in time series analysis*. New York, USA: John Wiley & Sons. [Amazon]

Chapter 9 Dynamic regression models

The time series models in the previous two chapters allow for the inclusion of information from past observations of a series, but not for the inclusion of other information that may also be relevant. For example, the effects of holidays, competitor activity, changes in the law, the wider economy, or other external variables, may explain some of the historical variation and may lead to more accurate forecasts. On the other hand, the regression models in Chapter 5 allow for the inclusion of a lot of relevant information from predictor variables, but do not allow for the subtle time series dynamics that can be handled with ARIMA models. In this chapter, we consider how to extend ARIMA models in order to allow other information to be included in the models.

In Chapter 5 we considered regression models of the form

$$y_t = \beta_0 + \beta_1 x_{1,t} + \cdots + \beta_k x_{k,t} + \varepsilon_t,$$

where y_t is a linear function of the k predictor variables $(x_{1,t}, \ldots, x_{k,t})$, and ε_t is usually assumed to be an uncorrelated error term (i.e., it is white noise). We considered tests such as the Breusch–Godfrey test for assessing whether the resulting residuals were significantly correlated.

In this chapter, we will allow the errors from a regression to contain autocorrelation. To emphasise this change in perspective, we will replace ε_t with η_t in the equation. The error series η_t is assumed to follow an ARIMA model. For example, if η_t follows an ARIMA(1,1,1) model, we can write

$$\begin{aligned} y_t &= \beta_0 + \beta_1 x_{1,t} + \cdots + \beta_k x_{k,t} + \eta_t, \\ (1-\phi_1 B)(1-B)\eta_t &= (1+\theta_1 B)\varepsilon_t, \end{aligned}$$

where ε_t is a white noise series.

Notice that the model has two error terms here — the error from the regression model, which we denote by η_t, and the error from the ARIMA model, which we denote by ε_t. Only the ARIMA model errors are assumed to be white noise.

9.1 Estimation

When we estimate the parameters from the model, we need to minimise the sum of squared ε_t values. If we minimise the sum of squared η_t values instead (which is what would happen if we estimated the regression model ignoring the autocorrelations in the errors), then several problems arise.

1. The estimated coefficients $\hat{\beta}_0, \dots, \hat{\beta}_k$ are no longer the best estimates, as some information has been ignored in the calculation;
2. Any statistical tests associated with the model (e.g., t-tests on the coefficients) will be incorrect.
3. The AICc values of the fitted models are no longer a good guide as to which is the best model for forecasting.
4. In most cases, the p-values associated with the coefficients will be too small, and so some predictor variables will appear to be important when they are not. This is known as "spurious regression".

Minimising the sum of squared ε_t values avoids these problems. Alternatively, maximum likelihood estimation can be used; this will give similar estimates of the coefficients.

An important consideration when estimating a regression with ARMA errors is that all of the variables in the model must first be stationary. Thus, we first have to check that y_t and all of the predictors $(x_{1,t}, \dots, x_{k,t})$ appear to be stationary. If we estimate the model when any of these are non-stationary, the estimated coefficients will not be consistent estimates (and therefore may not be meaningful). One exception to this is the case where non-stationary variables are co-integrated. If there exists a linear combination of the non-stationary y_t and the predictors that is stationary, then the estimated coefficients will be consistent.[19]

We therefore first difference the non-stationary variables in the model. It is often desirable to maintain the form of the relationship between y_t and the predictors, and consequently it is common to difference all of the variables if any of them need differencing. The resulting model is then called a "model in differences", as distinct from a "model in levels", which is what is obtained when the original data are used without differencing.

If all of the variables in the model are stationary, then we only need to consider ARMA errors for the residuals. It is easy to see that a regression model with ARIMA errors is equivalent to a regression model in differences with ARMA errors. For example, if the above regression model with ARIMA(1,1,1) errors is differenced we obtain the model

$$\begin{aligned} y'_t &= \beta_1 x'_{1,t} + \cdots + \beta_k x'_{k,t} + \eta'_t, \\ (1-\phi_1 B)\eta'_t &= (1+\theta_1 B)\varepsilon_t, \end{aligned}$$

where $y'_t = y_t - y_{t-1}$, $x'_{t,i} = x_{t,i} - x_{t-1,i}$ and $\eta'_t = \eta_t - \eta_{t-1}$, which is a regression model in differences with ARMA errors.

Bibliography

Harris, R., & Sollis, R. (2003). *Applied time series modelling and forecasting*. Chichester, UK: John Wiley & Sons. [Amazon]

19. Forecasting with cointegrated models is discussed by Harris & Sollis (2003).↩

9.2 Regression with ARIMA errors in R

The R function `Arima()` will fit a regression model with ARIMA errors if the argument `xreg` is used. The `order` argument specifies the order of the ARIMA error model. If differencing is specified, then the differencing is applied to all variables in the regression model before the model is estimated. For example, the R command

```
fit <- Arima(y, xreg=x, order=c(1,1,0))
```

will fit the model $y'_t = \beta_1 x'_t + \eta'_t$, where $\eta'_t = \phi_1 \eta'_{t-1} + \varepsilon_t$ is an AR(1) error. This is equivalent to the model

$$y_t = \beta_0 + \beta_1 x_t + \eta_t,$$

where η_t is an ARIMA(1,1,0) error. Notice that the constant term disappears due to the differencing. To include a constant in the differenced model, specify `include.drift=TRUE` .

The `auto.arima()` function will also handle regression terms via the `xreg` argument. The user must specify the predictor variables to include, but `auto.arima()` will select the best ARIMA model for the errors. If differencing is required, then all variables are differenced during the estimation process, although the final model will be expressed in terms of the original variables.

The AICc is calculated for the final model, and this value can be used to determine the best predictors. That is, the procedure should be repeated for all subsets of predictors to be considered, and the model with the lowest AICc value selected.

Example: US Personal Consumption and Income

Figure 9.1 shows the quarterly changes in personal consumption expenditure and personal disposable income from 1970 to 2016 Q3. We would like to forecast changes in expenditure based on changes in income. A change in income does not necessarily translate to an instant change in consumption (e.g., after the loss of a job, it may take a few months for expenses to be reduced to allow for the new circumstances).

However, we will ignore this complexity in this example and try to measure the instantaneous effect of the average change of income on the average change of consumption expenditure.

```
autoplot(uschange[,1:2], facets=TRUE) +
  xlab("Year") + ylab("") +
  ggtitle("Quarterly changes in US consumption
    and personal income")
```

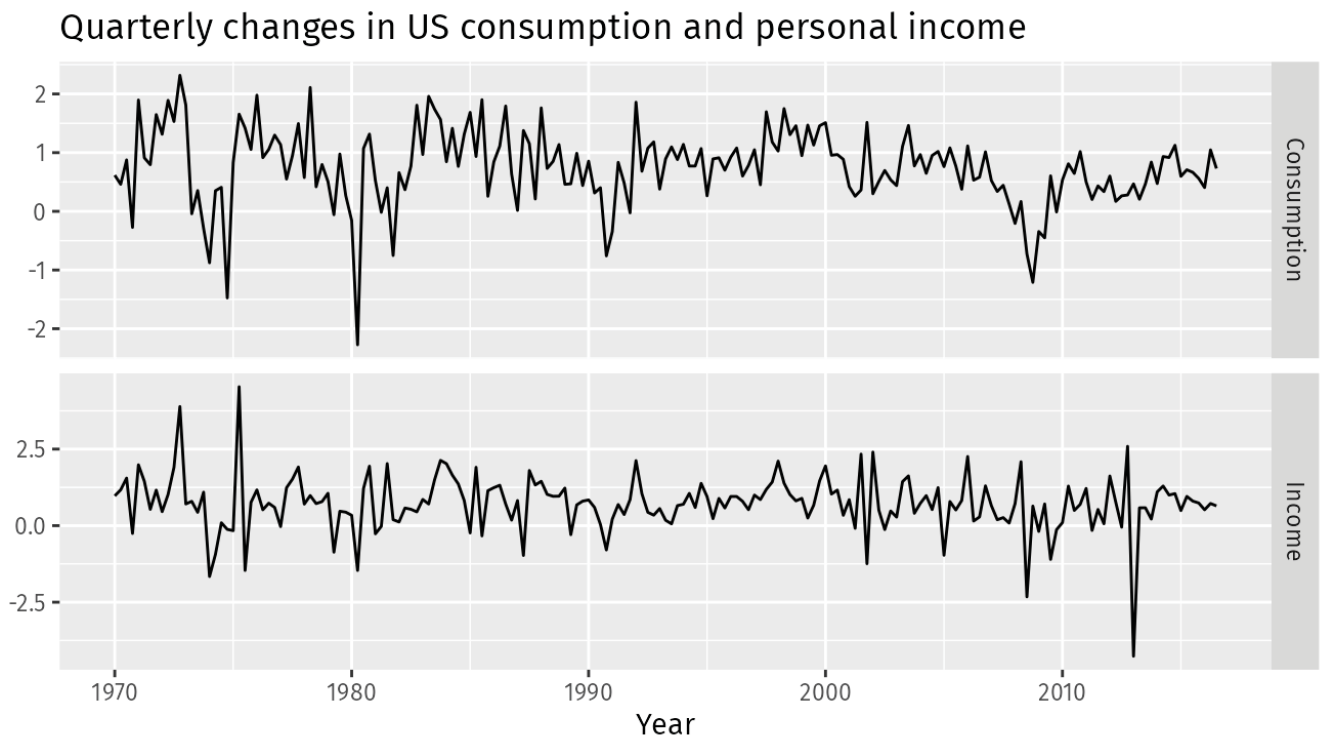


Figure 9.1: Percentage changes in quarterly personal consumption expenditure and personal disposable income for the USA, 1970 to 2016 Q3.

```
(fit <- auto.arima(uschange[,"Consumption"],
  xreg=uschange[,"Income"]))
#> Series: uschange[, "Consumption"]
#> Regression with ARIMA(1,0,2) errors
#>
#> Coefficients:
#>         ar1     ma1    ma2  intercept   xreg
#>       0.692  -0.576  0.198      0.599  0.203
#> s.e.  0.116   0.130  0.076      0.088  0.046
#>
#> sigma^2 = 0.322:  log likelihood = -156.9
#> AIC=325.9   AICc=326.4   BIC=345.3
```

The data are clearly already stationary (as we are considering percentage changes rather than raw expenditure and income), so there is no need for any differencing. The fitted model is

$$\begin{aligned} y_t &= 0.599 + 0.203x_t + \eta_t, \\ \eta_t &= 0.692\eta_{t-1} + \varepsilon_t - 0.576\varepsilon_{t-1} + 0.198\varepsilon_{t-2}, \\ \varepsilon_t &\sim \text{NID}(0, 0.322). \end{aligned}$$

We can recover estimates of both the η_t and ε_t series using the `residuals()` function.

```
cbind("Regression Errors" = residuals(fit, type="regression"),
      "ARIMA errors" = residuals(fit, type="innovation")) %>%
  autoplot(facets=TRUE)
```

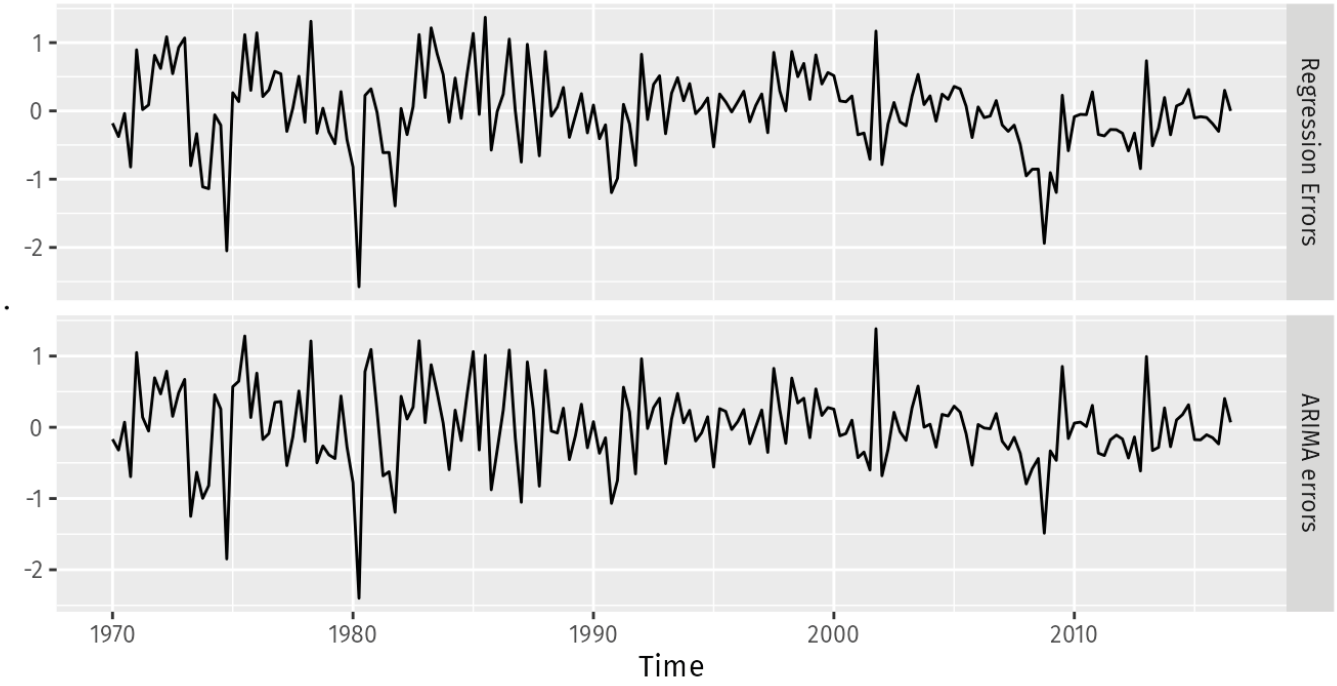


Figure 9.2: Regression errors (η_t) and ARIMA errors (ε_t) from the fitted model.

It is the ARIMA errors that should resemble a white noise series.

```
checkresiduals(fit)
```

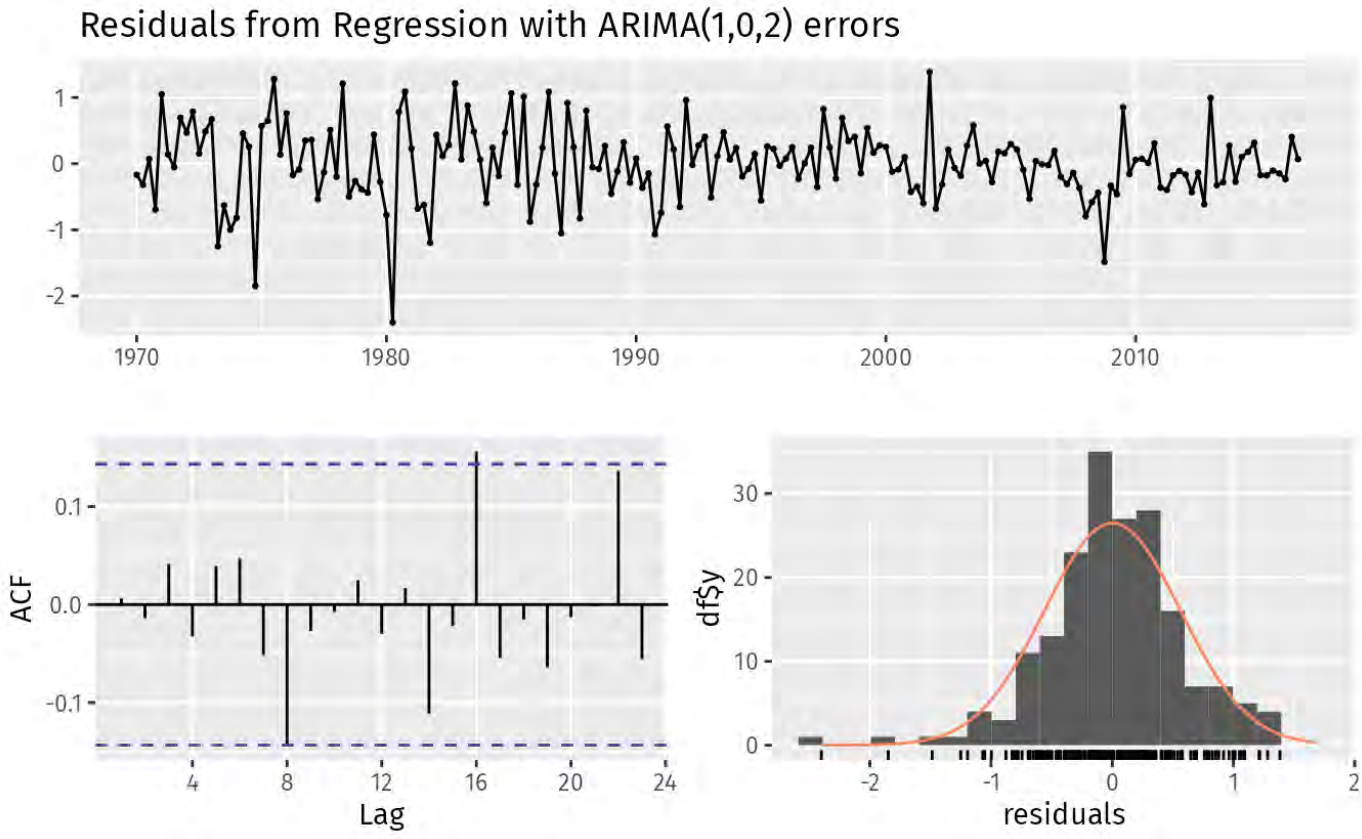


Figure 9.3: The residuals (i.e., the ARIMA errors) are not significantly different from white noise.

```
#>
#>  Ljung-Box test
#>
#> data:  Residuals from Regression with ARIMA(1,0,2) errors
#> Q* = 5.89, df = 5, p-value = 0.32
#>
#> Model df: 3.   Total lags used: 8
```

9.3 Forecasting

To forecast using a regression model with ARIMA errors, we need to forecast the regression part of the model and the ARIMA part of the model, and combine the results. As with ordinary regression models, in order to obtain forecasts we first need to forecast the predictors. When the predictors are known into the future (e.g., calendar-related variables such as time, day-of-week, etc.), this is straightforward. But when the predictors are themselves unknown, we must either model them separately, or use assumed future values for each predictor.

Example: US Personal Consumption and Income

We will calculate forecasts for the next eight quarters assuming that the future percentage changes in personal disposable income will be equal to the mean percentage change from the last forty years.

```
fcast <- forecast(fit, xreg=rep(mean(uschange[,2]),8))
autoplot(fcast) + xlab("Year") +
  ylab("Percentage change")
```

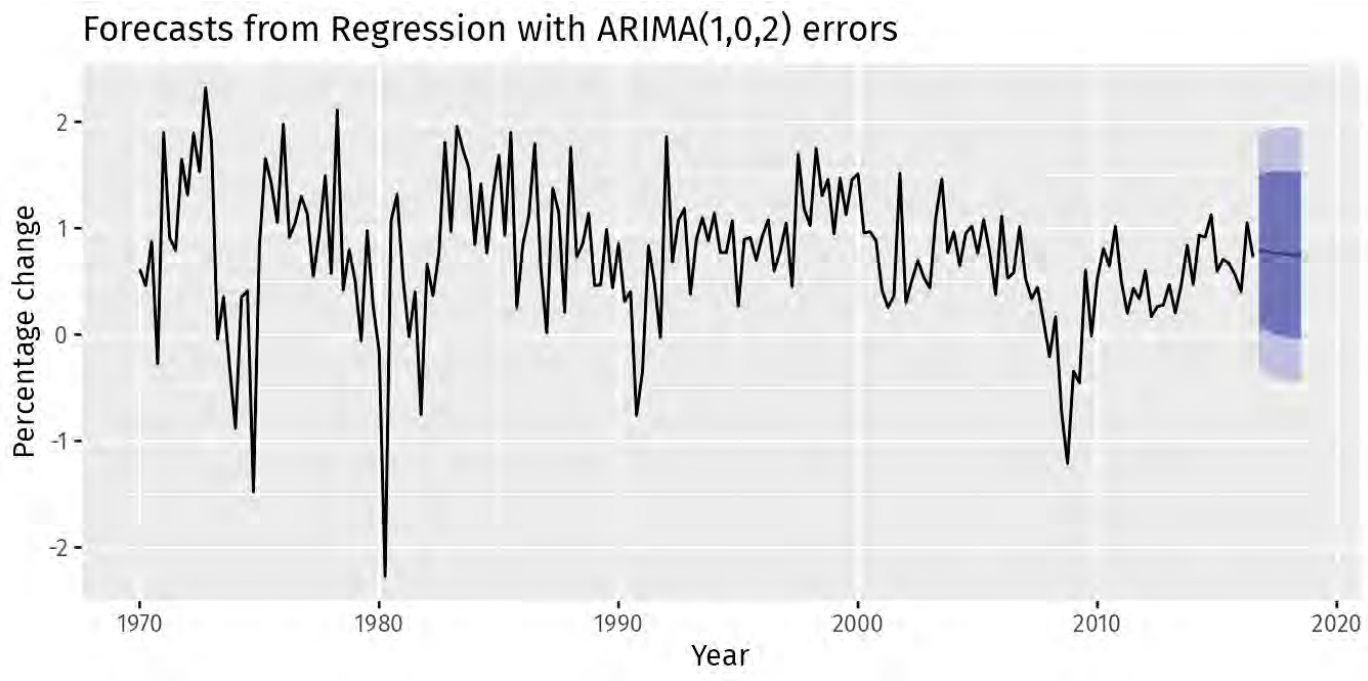


Figure 9.4: Forecasts obtained from regressing the percentage change in consumption expenditure on the percentage change in disposable income, with an ARIMA(1,0,2) error model.

The prediction intervals for this model are narrower than those for the model developed in Section 8.5 because we are now able to explain some of the variation in the data using the income predictor.

It is important to realise that the prediction intervals from regression models (with or without ARIMA errors) do not take into account the uncertainty in the forecasts of the predictors. So they should be interpreted as being conditional on the assumed (or estimated) future values of the predictor variables.

Example: Forecasting electricity demand

Daily electricity demand can be modelled as a function of temperature. As can be observed on an electricity bill, more electricity is used on cold days due to heating and hot days due to air conditioning. The higher demand on cold and hot days is reflected in the u-shape of Figure 9.5, where daily demand is plotted versus daily maximum temperature.

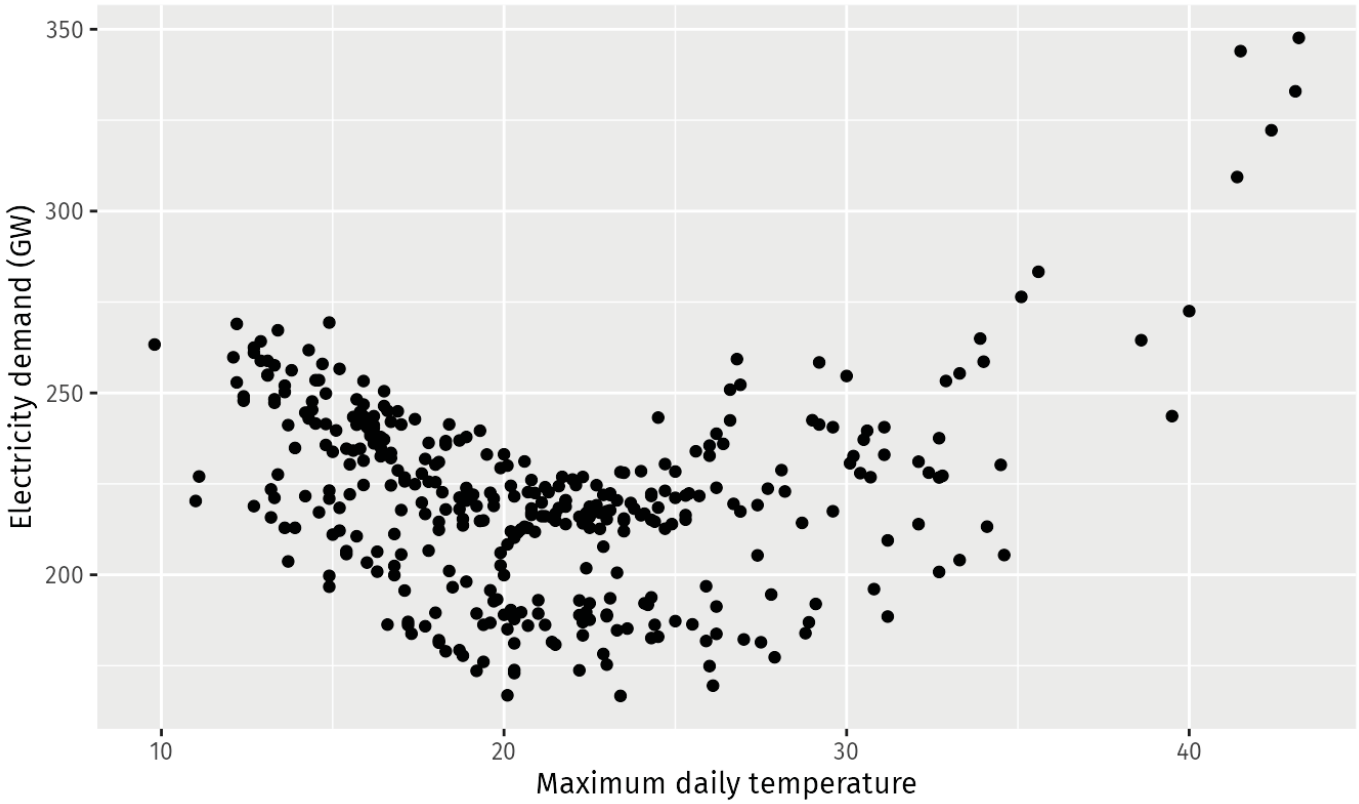


Figure 9.5: Daily electricity demand versus maximum daily temperature for the state of Victoria in Australia for 2014.

The data are stored as `elecdaily` including total daily demand, an indicator variable for workdays (a workday is represented with 1, and a non-workday is represented with 0), and daily maximum temperatures. Because there is weekly seasonality, the

frequency has been set to 7. Figure 9.6 shows the time series of both daily demand and daily maximum temperatures. The plots highlight the need for both a non-linear and a dynamic model.

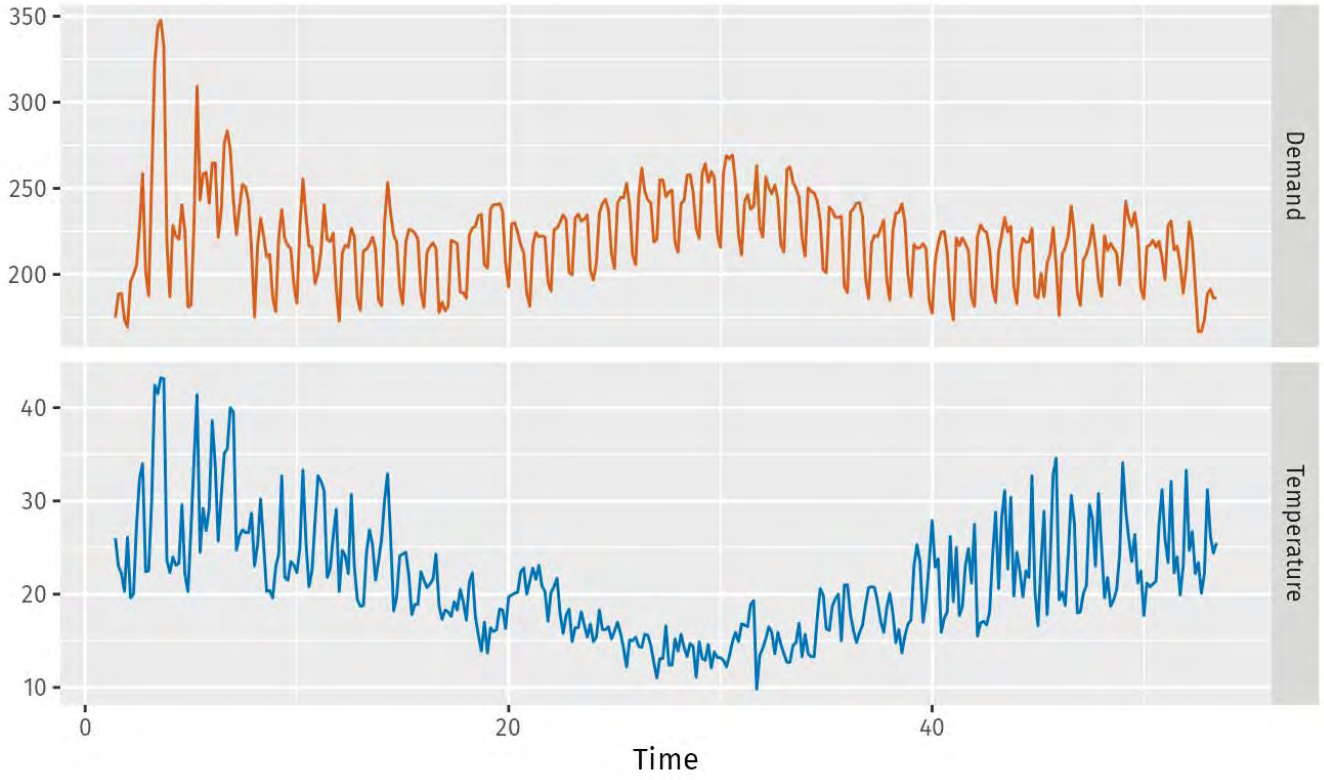


Figure 9.6: Daily electricity demand and maximum daily temperature for the state of Victoria in Australia for 2014.

In this example, we fit a quadratic regression model with ARMA errors using the `auto.arima()` function.

```
xreg <- cbind(MaxTemp = elecdaily[, "Temperature"],
              MaxTempSq = elecdaily[, "Temperature"]^2,
              Workday = elecdaily[, "WorkDay"])
fit <- auto.arima(elecdaily[, "Demand"], xreg = xreg)
checkresiduals(fit)
```

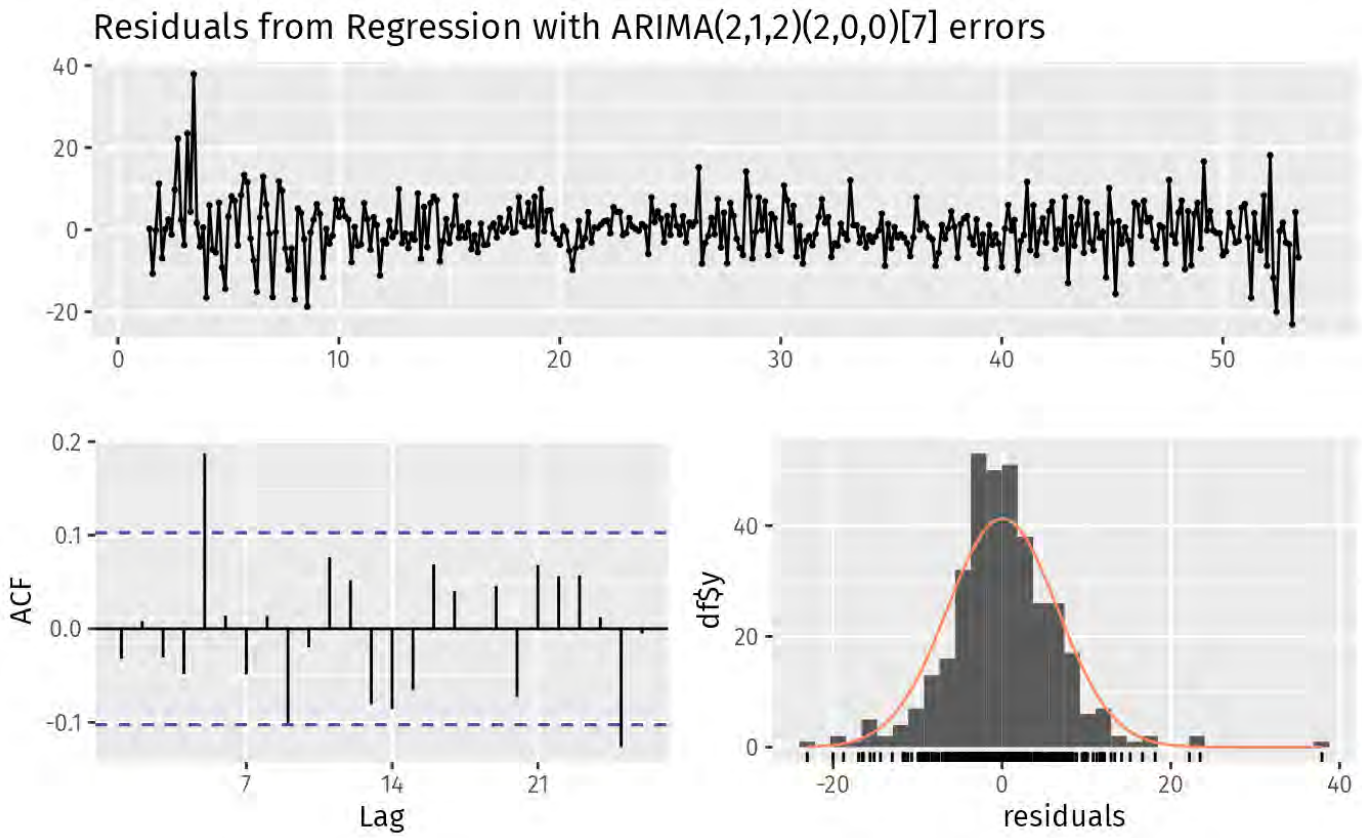


Figure 9.7: Residuals diagnostics for a dynamic regression model for daily electricity demand with workday and quadratic temperature effects.

```
#>
#>  Ljung-Box test
#>
#> data:  Residuals from Regression with ARIMA(2,1,2)(2,0,0)[7] errors
#> Q* = 28.2, df = 8, p-value = 0.00043
#>
#> Model df: 6.   Total lags used: 14
```

The model has some significant autocorrelation in the residuals, which means the prediction intervals may not provide accurate coverage. Also, the histogram of the residuals shows one positive outlier, which will also affect the coverage of the prediction intervals.

Using the estimated model we forecast 14 days ahead starting from Thursday 1 January 2015 (a non-work-day being a public holiday for New Years Day). In this case, we could obtain weather forecasts from the weather bureau for the next 14 days. But for the sake of illustration, we will use scenario based forecasting (as introduced in Section 5.6) where we set the temperature for the next 14 days to a constant 26 degrees.

```
fcast <- forecast(fit,
  xreg = cbind(MaxTemp=rep(26,14), MaxTempSq=rep(26^2,14),
    Workday=c(0,1,0,0,1,1,1,1,1,0,0,1,1,1)))
autoplot(fcast) + ylab("Electricity demand (GW)")
```

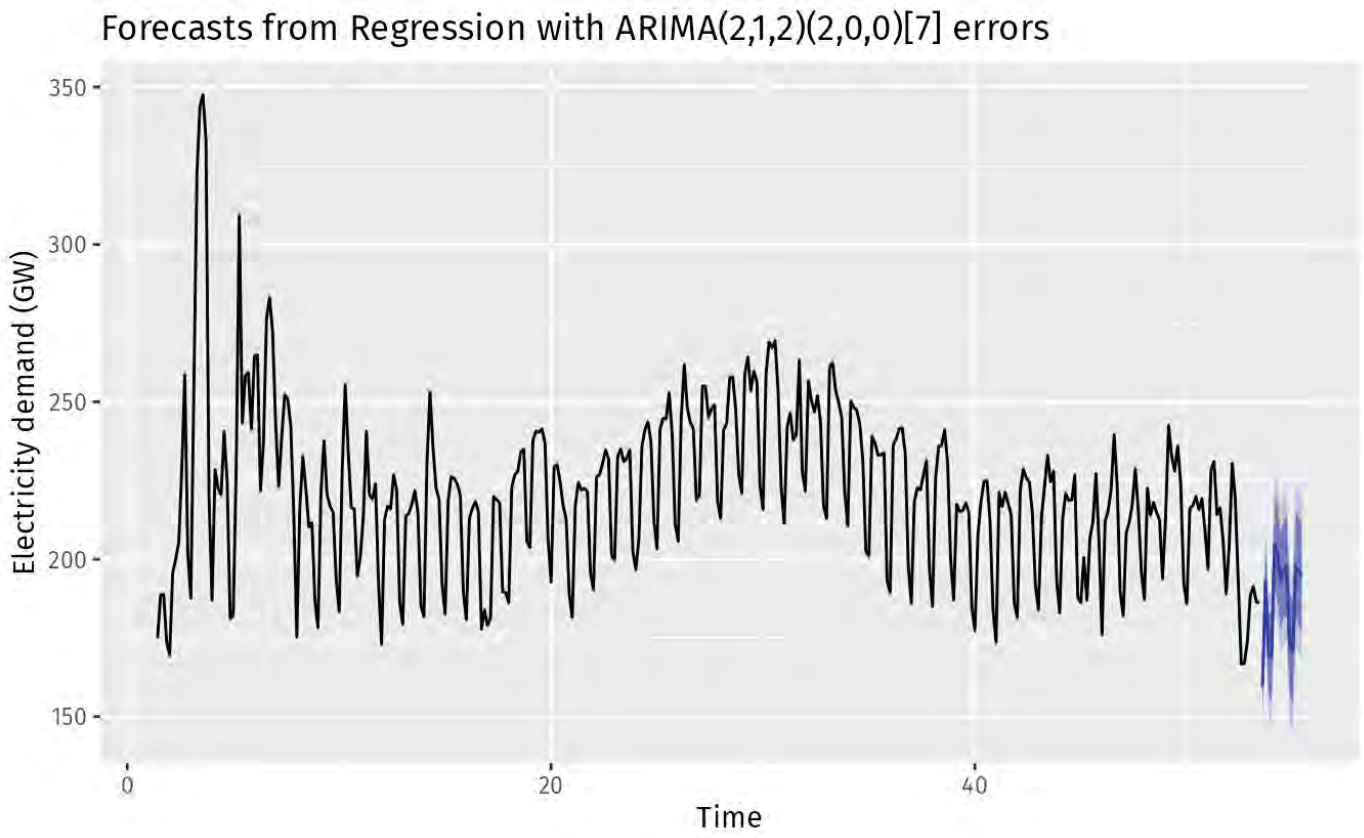


Figure 9.8: Forecasts from the dynamic regression model for daily electricity demand. All future temperatures have been set to 26 degrees, and the working day dummy variable has been set to known future values.

The point forecasts look reasonable for the first two weeks of 2015. The slow down in electricity demand at the end of 2014 (due to many people taking summer vacations) has caused the forecasts for the next two weeks to show similarly low demand values.

9.4 Stochastic and deterministic trends

There are two different ways of modelling a linear trend. A *deterministic trend* is obtained using the regression model

$$y_t = \beta_0 + \beta_1 t + \eta_t,$$

where η_t is an ARMA process. A *stochastic trend* is obtained using the model

$$y_t = \beta_0 + \beta_1 t + \eta_t,$$

where η_t is an ARIMA process with $d = 1$. In the latter case, we can difference both sides so that $y'_t = \beta_1 + \eta'_t$, where η'_t is an ARMA process. In other words,

$$y_t = y_{t-1} + \beta_1 + \eta'_t.$$

This is similar to a random walk with drift (introduced in Section 8.1), but here the error term is an ARMA process rather than simply white noise.

Although these models appear quite similar (they only differ in the number of differences that need to be applied to η_t), their forecasting characteristics are quite different.

Example: International visitors to Australia

```
autoplot(austa) + xlab("Year") +
  ylab("millions of people") +
  ggtitle("Total annual international visitors to Australia")
```

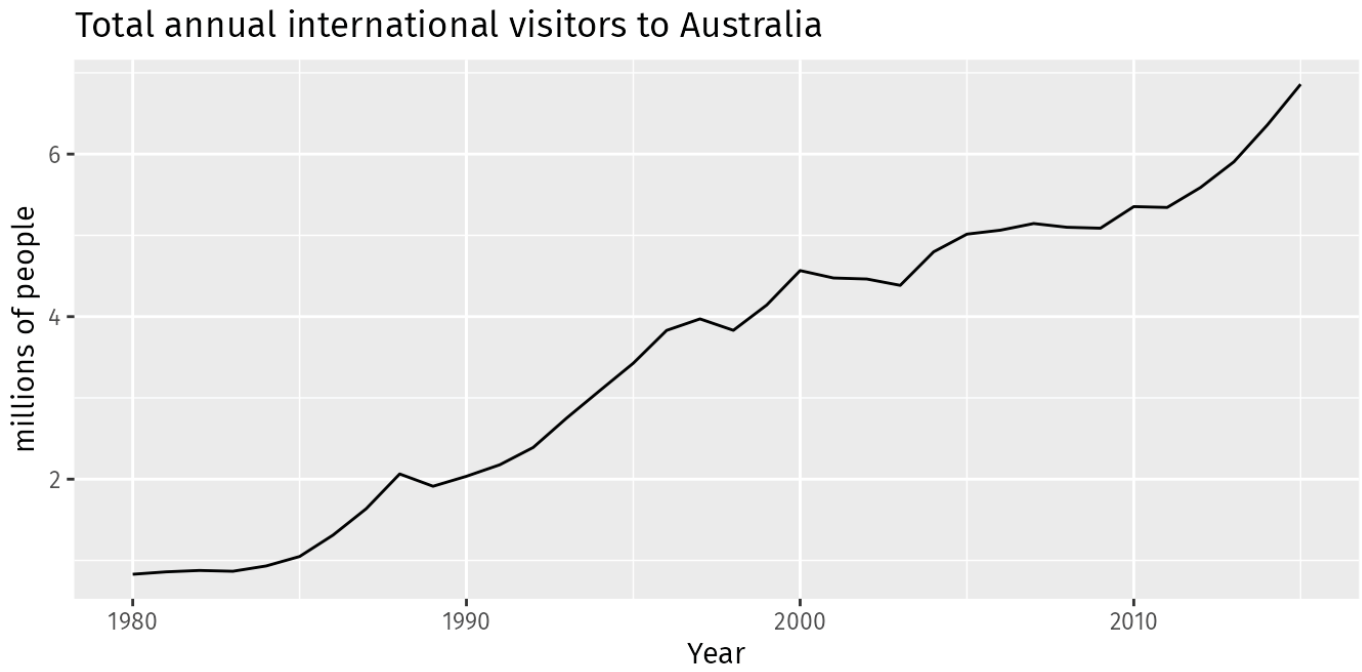


Figure 9.9: Annual international visitors to Australia, 1980–2015.

Figure 9.9 shows the total number of international visitors to Australia each year from 1980 to 2015. We will fit both a deterministic and a stochastic trend model to these data.

The deterministic trend model is obtained as follows:

```
trend <- seq_along(austa)
(fit1 <- auto.arima(austa, d=0, xreg=trend))
#> Series: austa
#> Regression with ARIMA(2,0,0) errors
#>
#> Coefficients:
#>          ar1     ar2  intercept   xreg
#>       1.113  -0.380      0.416  0.171
#> s.e.  0.160   0.158      0.190  0.009
#>
#> sigma^2 = 0.0298:  log likelihood = 13.6
#> AIC=-17.2   AICc=-15.2   BIC=-9.28
```

This model can be written as

$$
\begin{aligned}
y_t &= 0.416 + 0.171t + \eta_t \\
\eta_t &= 1.113\eta_{t-1} - 0.380\eta_{t-2} + \varepsilon_t \\
\varepsilon_t &\sim \text{NID}(0, 0.030).
\end{aligned}
$$

The estimated growth in visitor numbers is 0.17 million people per year.

Alternatively, the stochastic trend model can be estimated.

```
(fit2 <- auto.arima(austa, d=1))
#> Series: austa
#> ARIMA(0,1,1) with drift
#>
#> Coefficients:
#>         ma1  drift
#>       0.301  0.173
#> s.e.  0.165  0.039
#>
#> sigma^2 = 0.0338:  log likelihood = 10.62
#> AIC=-15.24   AICc=-14.46   BIC=-10.57
```

This model can be written as $y_t - y_{t-1} = 0.173 + \eta_t'$, or equivalently

$$\begin{aligned} y_t &= y_0 + 0.173t + \eta_t \\ \eta_t &= \eta_{t-1} + 0.301\varepsilon_{t-1} + \varepsilon_t \\ \varepsilon_t &\sim \text{NID}(0, 0.034). \end{aligned}$$

In this case, the estimated growth in visitor numbers is also 0.17 million people per year. Although the growth estimates are similar, the prediction intervals are not, as Figure 9.10 shows. In particular, stochastic trends have much wider prediction intervals because the errors are non-stationary.

```
fc1 <- forecast(fit1,
  xreg = length(austa) + 1:10)
fc2 <- forecast(fit2, h=10)
autoplot(austa) +
  autolayer(fc2, series="Stochastic trend") +
  autolayer(fc1, series="Deterministic trend") +
  ggtitle("Forecasts from trend models") +
  xlab("Year") + ylab("Visitors to Australia (millions)") +
  guides(colour=guide_legend(title="Forecast"))
```

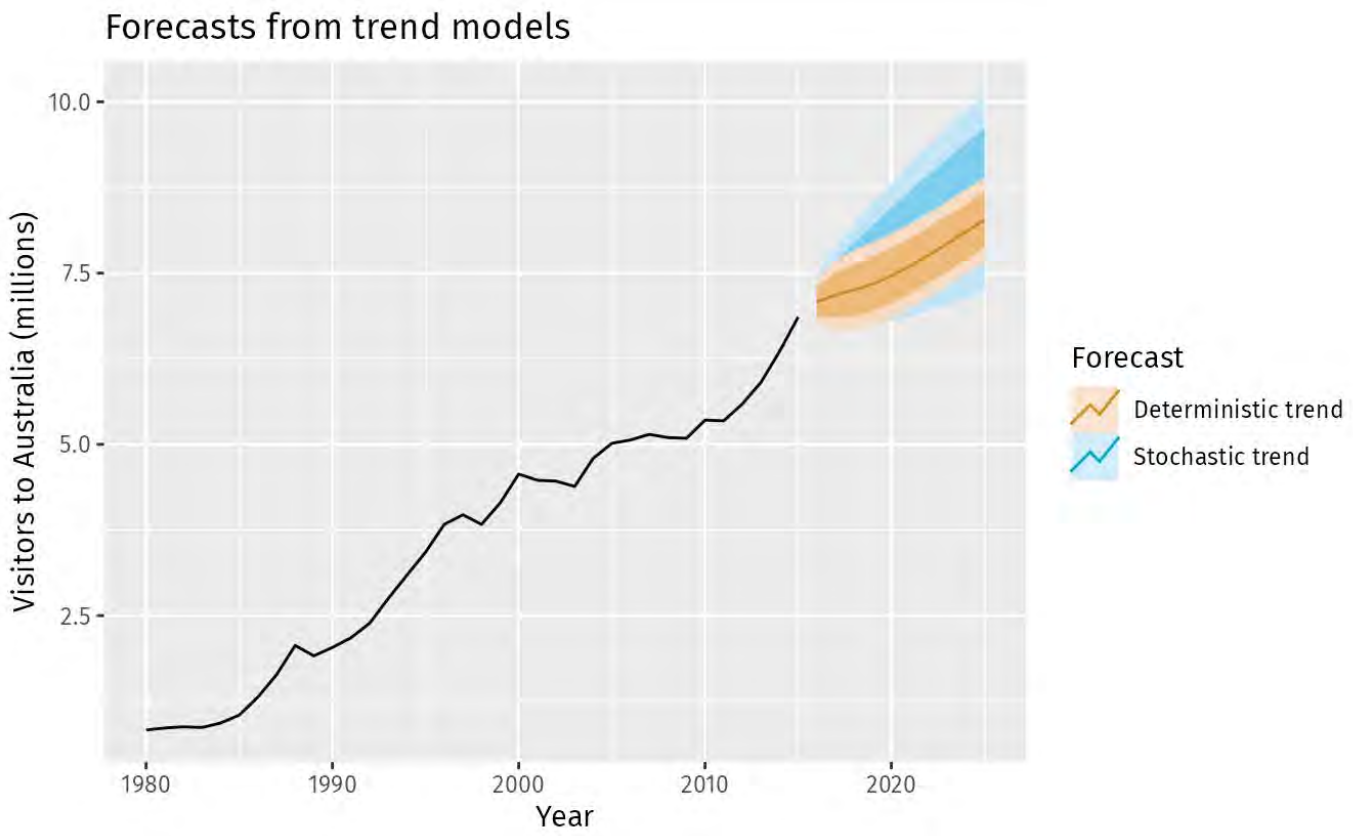


Figure 9.10: Forecasts of annual international visitors to Australia using a deterministic trend model and a stochastic trend model.

There is an implicit assumption with deterministic trends that the slope of the trend is not going to change over time. On the other hand, stochastic trends can change, and the estimated growth is only assumed to be the average growth over the historical period, not necessarily the rate of growth that will be observed into the future. Consequently, it is safer to forecast with stochastic trends, especially for longer forecast horizons, as the prediction intervals allow for greater uncertainty in future growth.

9.5 Dynamic harmonic regression

When there are long seasonal periods, a dynamic regression with Fourier terms is often better than other models we have considered in this book.[20]

For example, daily data can have annual seasonality of length 365, weekly data has seasonal period of approximately 52, while half-hourly data can have several seasonal periods, the shortest of which is the daily pattern of period 48.

Seasonal versions of ARIMA and ETS models are designed for shorter periods such as 12 for monthly data or 4 for quarterly data. The `ets()` function restricts seasonality to be a maximum period of 24 to allow hourly data but not data with a larger seasonal frequency. The problem is that there are $m-1$ parameters to be estimated for the initial seasonal states where m is the seasonal period. So for large m, the estimation becomes almost impossible.

The `Arima()` and `auto.arima()` functions will allow a seasonal period up to $m=350$, but in practice will usually run out of memory whenever the seasonal period is more than about 200. In any case, seasonal differencing of high order does not make a lot of sense — for daily data it involves comparing what happened today with what happened exactly a year ago and there is no constraint that the seasonal pattern is smooth.

So for such time series, we prefer a harmonic regression approach where the seasonal pattern is modelled using Fourier terms with short-term time series dynamics handled by an ARMA error.

The advantages of this approach are:

- it allows any length seasonality;
- for data with more than one seasonal period, Fourier terms of different frequencies can be included;
- the smoothness of the seasonal pattern can be controlled by K, the number of Fourier sin and cos pairs – the seasonal pattern is smoother for smaller values of K;
- the short-term dynamics are easily handled with a simple ARMA error.

The only real disadvantage (compared to a seasonal ARIMA model) is that the seasonality is assumed to be fixed — the seasonal pattern is not allowed to change over time. But in practice, seasonality is usually remarkably constant so this is not a big disadvantage except for long time series.

Example: Australian eating out expenditure

In this example we demonstrate combining Fourier terms for capturing seasonality with ARIMA errors capturing other dynamics in the data. For simplicity, we will use an example with monthly data. The same modelling approach using weekly data is discussed in Section 12.1.

We use `auscafe`, the total monthly expenditure on cafes, restaurants and takeaway food services in Australia ($billion), starting in 2004 up to November 2016 and we forecast 24 months ahead. We vary K, the number of Fourier sin and cos pairs, from $K = 1$ to $K = 6$ (which is equivalent to including seasonal dummies). Figure 9.11 shows the seasonal pattern projected forward as K increases. Notice that as K increases the Fourier terms capture and project a more "wiggly" seasonal pattern and simpler ARIMA models are required to capture other dynamics. The AICc value is minimised for $K = 5$, with a significant jump going from $K = 4$ to $K = 5$, hence the forecasts generated from this model would be the ones used.

```
cafe04 <- window(auscafe, start=2004)
plots <- list()
for (i in seq(6)) {
  fit <- auto.arima(cafe04, xreg = fourier(cafe04, K = i),
    seasonal = FALSE, lambda = 0)
  plots[[i]] <- autoplot(forecast(fit,
      xreg=fourier(cafe04, K=i, h=24))) +
    xlab(paste("K=",i,"   AICC=",round(fit[["aicc"]],2))) +
    ylab("") + ylim(1.5,4.7)
}
gridExtra::grid.arrange(
  plots[[1]],plots[[2]],plots[[3]],
  plots[[4]],plots[[5]],plots[[6]], nrow=3)
```

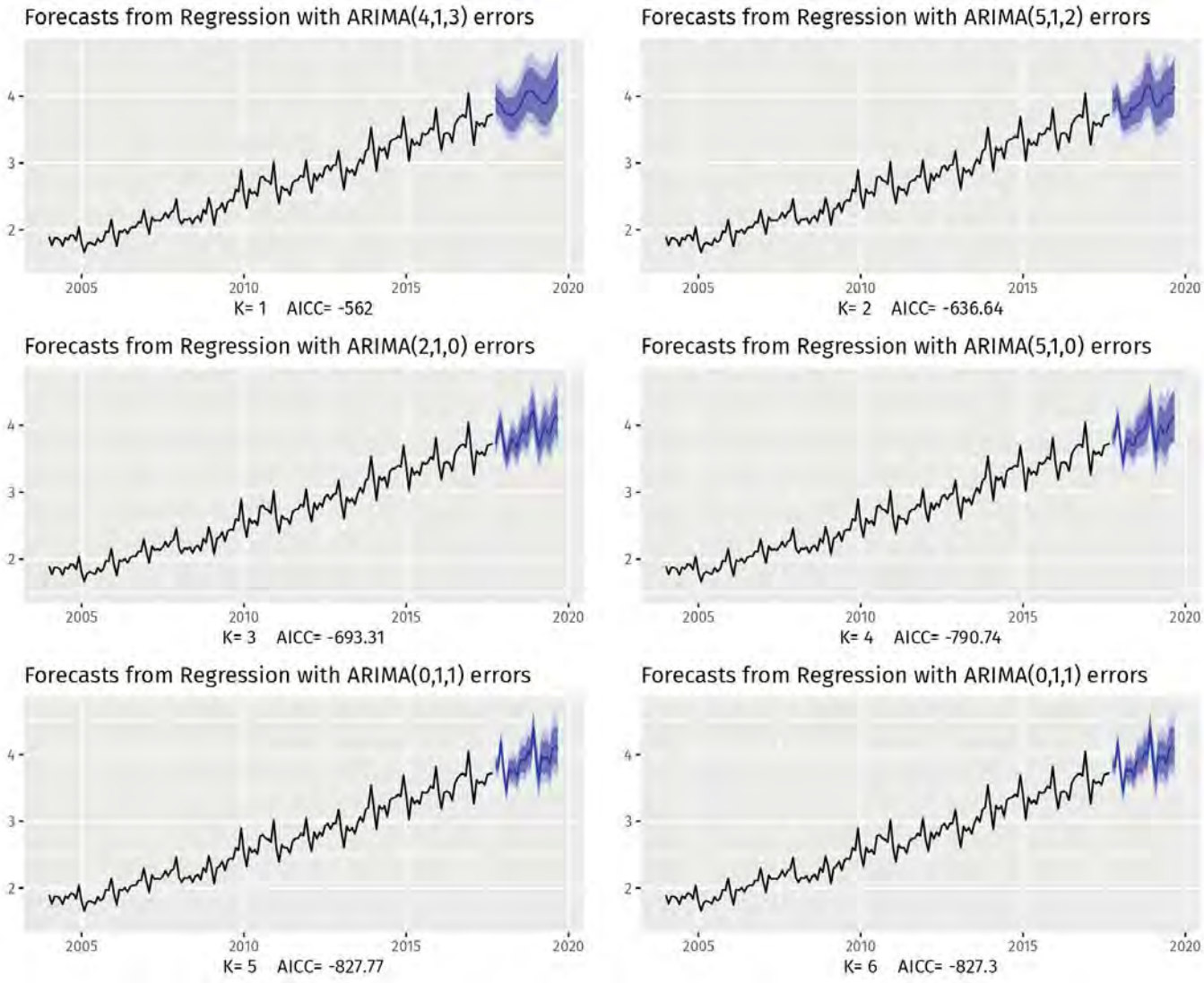


Figure 9.11: Using Fourier terms and ARIMA errors for forecasting monthly expenditure on eating out in Australia.

Bibliography

Young, P. C., Pedregal, D. J., & Tych, W. (1999). Dynamic harmonic regression. *Journal of Forecasting*, *18*, 369–394. [DOI]

20. The term "dynamic harmonic regression" is also used for a harmonic regression with time-varying parameters (Young, Pedregal, & Tych, 1999).↩

9.6 Lagged predictors

Sometimes, the impact of a predictor which is included in a regression model will not be simple and immediate. For example, an advertising campaign may impact sales for some time beyond the end of the campaign, and sales in one month will depend on the advertising expenditure in each of the past few months. Similarly, a change in a company's safety policy may reduce accidents immediately, but have a diminishing effect over time as employees take less care when they become familiar with the new working conditions.

In these situations, we need to allow for lagged effects of the predictor. Suppose that we have only one predictor in our model. Then a model which allows for lagged effects can be written as

$$y_t = \beta_0 + \gamma_0 x_t + \gamma_1 x_{t-1} + \cdots + \gamma_k x_{t-k} + \eta_t,$$

where η_t is an ARIMA process. The value of k can be selected using the AICc, along with the values of p and q for the ARIMA error.

Example: TV advertising and insurance quotations

A US insurance company advertises on national television in an attempt to increase the number of insurance quotations provided (and consequently the number of new policies). Figure 9.12 shows the number of quotations and the expenditure on television advertising for the company each month from January 2002 to April 2005.

```
autoplot(insurance, facets=TRUE) +
  xlab("Year") + ylab("") +
  ggtitle("Insurance advertising and quotations")
```

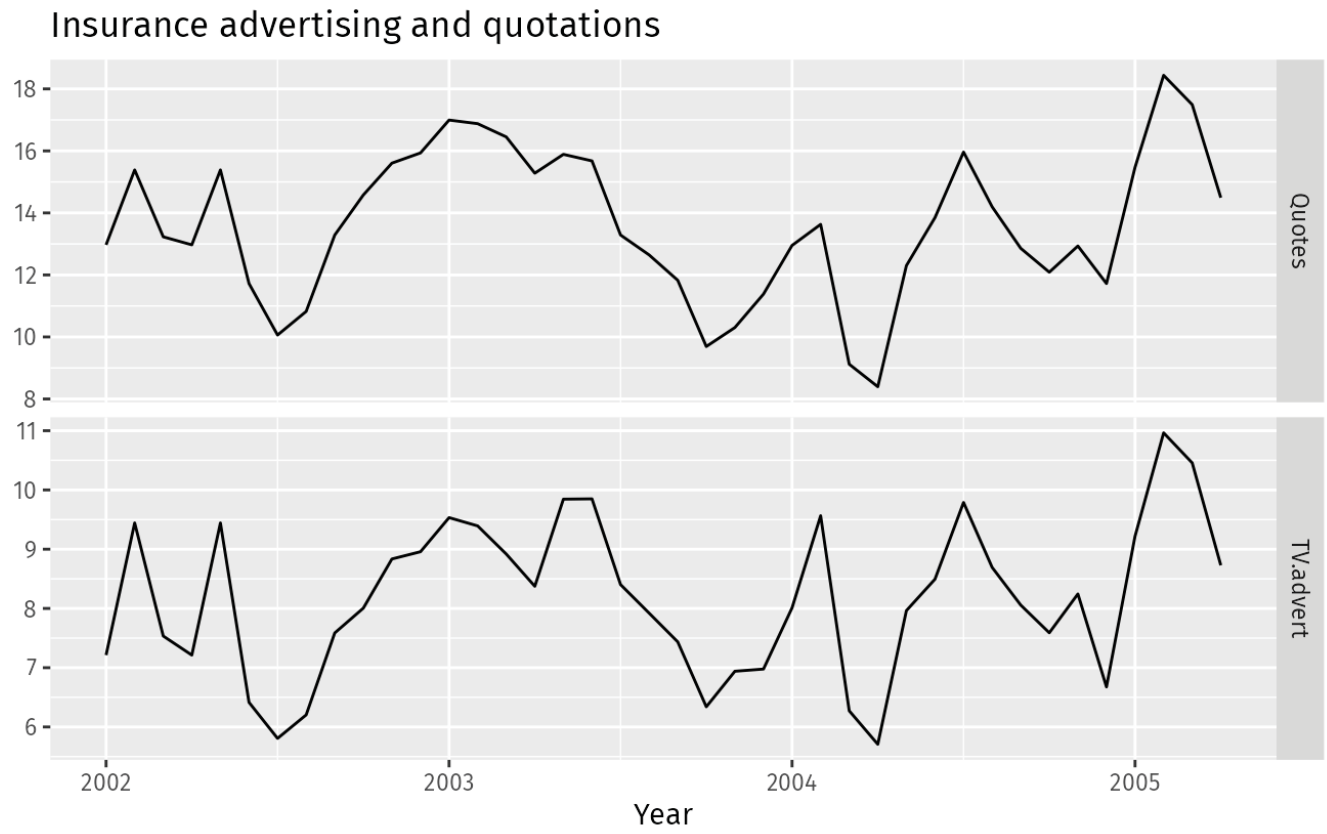


Figure 9.12: Numbers of insurance quotations provided per month and the expenditure on advertising per month.

We will consider including advertising expenditure for up to four months; that is, the model may include advertising expenditure in the current month, and the three months before that. When comparing models, it is important that they all use the same training set. In the following code, we exclude the first three months in order to make fair comparisons.

```
# Lagged predictors. Test 0, 1, 2 or 3 lags.
Advert <- cbind(
    AdLag0 = insurance[,"TV.advert"],
    AdLag1 = stats::lag(insurance[,"TV.advert"],-1),
    AdLag2 = stats::lag(insurance[,"TV.advert"],-2),
    AdLag3 = stats::lag(insurance[,"TV.advert"],-3)) %>%
  head(NROW(insurance))

# Restrict data so models use same fitting period
fit1 <- auto.arima(insurance[4:40,1], xreg=Advert[4:40,1],
  stationary=TRUE)
fit2 <- auto.arima(insurance[4:40,1], xreg=Advert[4:40,1:2],
  stationary=TRUE)
fit3 <- auto.arima(insurance[4:40,1], xreg=Advert[4:40,1:3],
  stationary=TRUE)
fit4 <- auto.arima(insurance[4:40,1], xreg=Advert[4:40,1:4],
  stationary=TRUE)
```

Next we choose the optimal lag length for advertising based on the AICc.

```
c(fit1[["aicc"]],fit2[["aicc"]],fit3[["aicc"]],fit4[["aicc"]])
#> [1] 68.500 60.024 62.833 65.457
```

The best model (with the smallest AICc value) has two lagged predictors; that is, it includes advertising only in the current month and the previous month. So we now re-estimate that model, but using all the available data.

```
(fit <- auto.arima(insurance[,1], xreg=Advert[,1:2],
  stationary=TRUE))
#> Series: insurance[, 1]
#> Regression with ARIMA(3,0,0) errors
#>
#> Coefficients:
#>         ar1     ar2    ar3  intercept  AdLag0  AdLag1
#>       1.412  -0.932  0.359      2.039   1.256   0.162
#> s.e.  0.170   0.255  0.159      0.993   0.067   0.059
#>
#> sigma^2 = 0.217:  log likelihood = -23.89
#> AIC=61.78   AICc=65.4   BIC=73.43
```

The chosen model has AR(3) errors. The model can be written as

$$y_t = 2.039 + 1.256x_t + 0.162x_{t-1} + \eta_t,$$

where y_t is the number of quotations provided in month t, x_t is the advertising expenditure in month t,

$$\eta_t = 1.412\eta_{t-1} - 0.932\eta_{t-2} + 0.359\eta_{t-3} + \varepsilon_t,$$

and ε_t is white noise.

We can calculate forecasts using this model if we assume future values for the advertising variable. If we set the future monthly advertising to 8 units, we get the forecasts in Figure 9.13.

```
fc8 <- forecast(fit, h=20,
  xreg=cbind(AdLag0 = rep(8,20),
             AdLag1 = c(Advert[40,1], rep(8,19))))
autoplot(fc8) + ylab("Quotes") +
  ggtitle("Forecast quotes with future advertising set to 8")
```

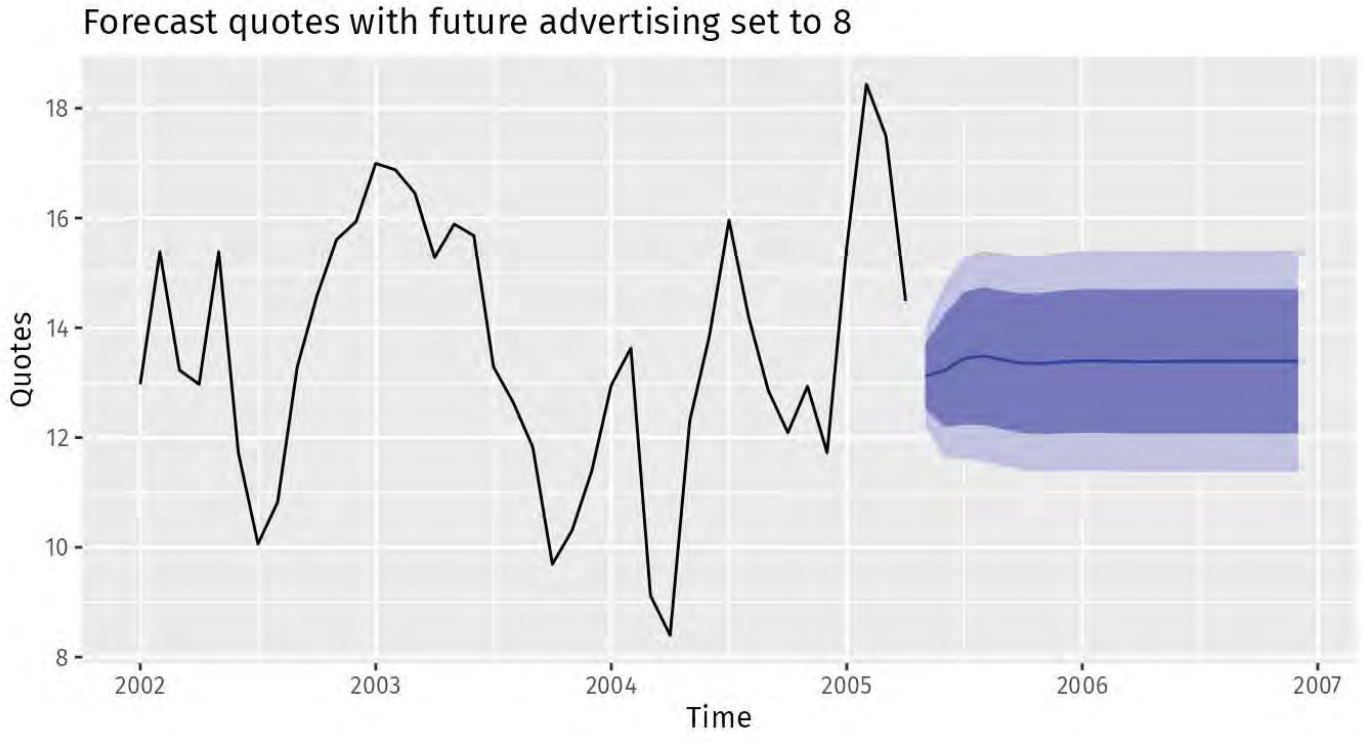


Figure 9.13: Forecasts of monthly insurance quotes, assuming that the future advertising expenditure is 8 units in each future month.

9.7 Exercises

1. Consider monthly sales and advertising data for an automotive parts company (data set `advert`).

   a. Plot the data using `autoplot`. Why is it useful to set `facets=TRUE`?

   b. Fit a standard regression model $y_t = a + bx_t + \eta_t$ where y_t denotes sales and x_t denotes advertising using the `tslm()` function.

   c. Show that the residuals have significant autocorrelation.

   d. What difference does it make you use the `Arima` function instead:

   ```
   Arima(advert[,"sales"], xreg=advert[,"advert"],
     order=c(0,0,0))
   ```

   e. Refit the model using `auto.arima()`. How much difference does the error model make to the estimated parameters? What ARIMA model for the errors is selected?

   f. Check the residuals of the fitted model.

   g. Assuming the advertising budget for the next six months is exactly 10 units per month, produce and plot sales forecasts with prediction intervals for the next six months.

2. This exercise uses data set `huron` giving the level of Lake Huron from 1875–1972.

   a. Fit a piecewise linear trend model to the Lake Huron data with a knot at 1920 and an ARMA error structure.

   b. Forecast the level for the next 30 years.

3. This exercise concerns `motel`: the total monthly takings from accommodation and the total room nights occupied at hotels, motels, and guest houses in Victoria, Australia, between January 1980 and June 1995. Total monthly takings are in thousands of Australian dollars; total room nights occupied are in thousands.

a. Use the data to calculate the average cost of a night's accommodation in Victoria each month.
b. Use `cpimel` to estimate the monthly CPI.
c. Produce time series plots of both variables and explain why logarithms of both variables need to be taken before fitting any models.
d. Fit an appropriate regression model with ARIMA errors. Explain your reasoning in arriving at the final model.
e. Forecast the average price per room for the next twelve months using your fitted model. (Hint: You will need to produce forecasts of the CPI figures first.)

4. We fitted a harmonic regression model to part of the `gasoline` series in Exercise 6 in Section 5.10. We will now revisit this model, and extend it to include more data and ARMA errors.

a. Using `tslm()`, fit a harmonic regression with a piecewise linear time trend to the full `gasoline` series. Select the position of the knots in the trend and the appropriate number of Fourier terms to include by minimising the AICc or CV value.
b. Now refit the model using `auto.arima()` to allow for correlated errors, keeping the same predictor variables as you used with `tslm()`.
c. Check the residuals of the final model using the `checkresiduals()` function. Do they look sufficiently like white noise to continue? If not, try modifying your model, or removing the first few years of data.
d. Once you have a model with white noise residuals, produce forecasts for the next year.

5. Electricity consumption is often modelled as a function of temperature. Temperature is measured by daily heating degrees and cooling degrees. Heating degrees is 18°C minus the average daily temperature when the daily average is below 18°C; otherwise it is zero. This provides a measure of our need to heat ourselves as temperature falls. Cooling degrees measures our need to cool ourselves as the temperature rises. It is defined as the average daily temperature minus 18°C when the daily average is above 18°C; otherwise it is zero. Let y_t denote the monthly total of kilowatt-hours of electricity used, let $x_{1,t}$ denote the monthly total of heating degrees, and let $x_{2,t}$ denote the monthly total of cooling degrees.

An analyst fits the following model to a set of such data:

$$y_t^* = \beta_1 x_{1,t}^* + \beta_2 x_{2,t}^* + \eta_t,$$

where

$$(1-B)(1-B^{12})\eta_t = \frac{1-\theta_1 B}{1-\phi_{12}B^{12}-\phi_{24}B^{24}}\varepsilon_t$$

and $y_t^* = \log(y_t)$, $x_{1,t}^* = \sqrt{x_{1,t}}$ and $x_{2,t}^* = \sqrt{x_{2,t}}$.

a. What sort of ARIMA model is identified for η_t?

b. The estimated coefficients are

Parameter	Estimate	s.e.	Z	P-value
β_1	0.0077	0.0015	4.98	0.000
β_2	0.0208	0.0023	9.23	0.000
θ_1	0.5830	0.0720	8.10	0.000
ϕ_{12}	-0.5373	0.0856	-6.27	0.000
ϕ_{24}	-0.4667	0.0862	-5.41	0.000

Explain what the estimates of β_1 and β_2 tell us about electricity consumption.

c. Write the equation in a form more suitable for forecasting.

d. Describe how this model could be used to forecast electricity demand for the next 12 months.

e. Explain why the η_t term should be modelled with an ARIMA model rather than modelling the data using a standard regression package. In your discussion, comment on the properties of the estimates, the validity of the standard regression results, and the importance of the η_t model in producing forecasts.

6. For the retail time series considered in earlier chapters:

a. Develop an appropriate dynamic regression model with Fourier terms for the seasonality. Use the AIC to select the number of Fourier terms to include in the model. (You will probably need to use the same Box-Cox transformation you identified previously.)

b. Check the residuals of the fitted model. Does the residual series look like white noise?

c. Compare the forecasts with those you obtained earlier using alternative models.

9.8 Further reading

- A detailed discussion of dynamic regression models is provided in Pankratz (1991).
- A generalisation of dynamic regression models, known as "transfer function models", is discussed in Box et al. (2015).

Bibliography

Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: Forecasting and control* (5th ed). Hoboken, New Jersey: John Wiley & Sons. [Amazon]

Pankratz, A. E. (1991). *Forecasting with dynamic regression models*. New York, USA: John Wiley & Sons. [Amazon]

Chapter 10 Forecasting hierarchical or grouped time series

Warning: this is a more advanced chapter and assumes a knowledge of some basic matrix algebra.

Time series can often be naturally disaggregated by various attributes of interest. For example, the total number of bicycles sold by a cycling manufacturer can be disaggregated by product type such as road bikes, mountain bikes, children's bikes and hybrids. Each of these can be disaggregated into finer categories. For example hybrid bikes can be divided into city, commuting, comfort, and trekking bikes; and so on. These categories are nested within the larger group categories, and so the collection of time series follow a hierarchical aggregation structure. Therefore we refer to these as "hierarchical time series", the topic of Section 10.1.

Hierarchical time series often arise due to geographic divisions. For example, the total bicycle sales can be disaggregated by country, then within each country by state, within each state by region, and so on down to the outlet level.

Our bicycle manufacturer may disaggregate sales by both product type and by geographic location. Then we have a more complicated aggregation structure where the product hierarchy and the geographic hierarchy can both be used together. We usually refer to these as "grouped time series", and discuss them in Section 10.2.

It is common to produce disaggregated forecasts based on disaggregated time series, and we usually require the forecasts to add up in the same way as the data. For example, forecasts of regional sales should add up to give forecasts of state sales, which should in turn add up to give a forecast for the national sales.

In this chapter we discuss forecasting large collections of time series that must add up in some way. The challenge is that we require forecasts that are **coherent** across the aggregation structure. That is, we require forecasts to add up in a manner that is

10.1 Hierarchical time series

Figure 10.1 shows a $K = 2$-level hierarchical structure. At the top of the hierarchy (which we call level 0) is the "Total", the most aggregate level of the data. The tth observation of the Total series is denoted by y_t for $t = 1, \dots, T$. The Total is disaggregated into two series at level 1, which in turn are divided into three and two series respectively at the bottom-level of the hierarchy. Below the top level, we use $y_{\mathrm{j},t}$ to denote the tth observation of the series corresponding to node j. For example, $y_{A,t}$ denotes the tth observation of the series corresponding to node A at level 1, $y_{AB,t}$ denotes the tth observation of the series corresponding to node AB at level 2, and so on.

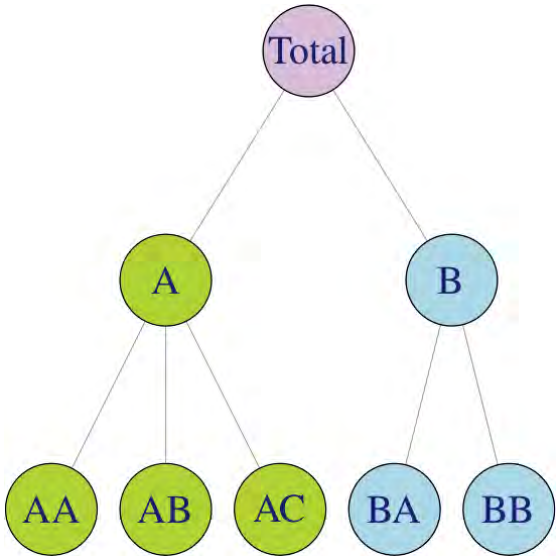


Figure 10.1: A two level hierarchical tree diagram.

In this small example, the total number of series in the hierarchy is $n = 1 + 2 + 5 = 8$, while the number of series at the bottom-level is $m = 5$. Note that $n > m$ in all hierarchies.

For any time t, the observations at the bottom-level of the hierarchy will sum to the observations of the series above. For example,

$$y_t = y_{\mathrm{AA},t} + y_{\mathrm{AB},t} + y_{\mathrm{AC},t} + y_{\mathrm{BA},t} + y_{\mathrm{BB},t} \tag{10.1}$$

and

$$y_{\mathrm{A},t} = y_{\mathrm{AA},t} + y_{\mathrm{AB},t} + y_{\mathrm{AC},t} \quad \text{and} \quad y_{\mathrm{B},t} = y_{\mathrm{BA},t} + y_{\mathrm{BB},t}. \tag{10.2}$$

Substituting (10.2) into (10.1), we also get $y_t = y_{\mathrm{A},t} + y_{\mathrm{B},t}$. These equations can be

thought of as aggregation constraints or summing equalities, and can be more efficiently represented using matrix notation. We construct an $n \times m$ matrix $\boldsymbol{S}$ (referred to as the "summing matrix") which dictates the way in which the bottom-level series are aggregated.

For the hierarchical structure in Figure 10.1, we can write

$$\begin{bmatrix} y_t \\ y_{\text{A},t} \\ y_{\text{B},t} \\ y_{\text{AA},t} \\ y_{\text{AB},t} \\ y_{\text{AC},t} \\ y_{\text{BA},t} \\ y_{\text{BB},t} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{\text{AA},t} \\ y_{\text{AB},t} \\ y_{\text{AC},t} \\ y_{\text{BA},t} \\ y_{\text{BB},t} \end{bmatrix}$$

or in more compact notation

$$\boldsymbol{y}_t = \boldsymbol{S}\boldsymbol{b}_t, \tag{10.3}$$

where $\boldsymbol{y}_t$ is an n-dimensional vector of all the observations in the hierarchy at time t, $\boldsymbol{S}$ is the summing matrix, and $\boldsymbol{b}_t$ is an m-dimensional vector of all the observations in the bottom-level of the hierarchy at time t. Note that the first row in the summing matrix $\boldsymbol{S}$ represents Equation (10.1) above, the second and third rows represent (10.2). The rows below these comprise an m-dimensional identity matrix $\boldsymbol{I}_m$ so that each bottom-level observation on the right hand side of the equation is equal to itself on the left hand side.

Example: Australian tourism hierarchy

Australia is divided into eight geographic areas (some called states and others called territories) with each one having its own government and some economic and administrative autonomy. Each of these can be further subdivided into smaller areas of interest, referred to as zones. Business planners and tourism authorities are interested in forecasts for the whole of Australia, for the states and the territories, and also for the zones. In this example we concentrate on quarterly domestic tourism demand, measured as the number of visitor nights Australians spend away from home. To simplify our analysis, we combine the two territories and Tasmania

into an "Other" state. So we have six states: New South Wales (NSW), Queensland (QLD), South Australia (SAU), Victoria (VIC), Western Australia (WAU) and Other (OTH). For each of these we consider visitor nights within the following zones.

State	Zones
NSW	Metro (NSWMetro), North Coast (NSWNthCo), South Coast (NSWSthCo), South Inner (NSWSthIn), North Inner (NSWNthIn)
QLD	Metro (QLDMetro), Central (QLDCntrl), North Coast (QLDNthCo)
SAU	Metro (SAUMetro), Coastal (SAUCoast), Inner (SAUInner)
VIC	Metro (VICMetro), West Coast (VICWstCo), East Coast (VICEstCo), Inner (VICInner)
WAU	Metro (WAUMetro), Coastal (WAUCoast), Inner (WAUInner)
OTH	Metro (OTHMetro), Non-Metro (OTHNoMet)

We consider five zones for NSW, four zones for VIC, and three zones each for QLD, SAU and WAU. Note that Metro zones contain the capital cities and surrounding areas. For further details on these geographic areas, please refer to Appendix C in Wickramasuriya, Athanasopoulos, & Hyndman (2019).

To create a hierarchical time series, we use the `hts()` function as shown in the code below. The function requires two inputs: the bottom-level time series and information about the hierarchical structure. `visnights` is a time series matrix containing the bottom-level series. There are several ways to input the structure of the hierarchy. In this case we are using the `characters` argument. The first three characters of each column name of `visnights` capture the categories at the first level of the hierarchy (States). The following five characters capture the bottom-level categories (Zones).

```
library(hts)
tourism.hts <- hts(visnights, characters = c(3, 5))
tourism.hts %>% aggts(levels=0:1) %>%
  autoplot(facet=TRUE) +
  xlab("Year") + ylab("millions") + ggtitle("Visitor nights")
```

Figure 10.2: Australian domestic visitor nights over the period 1998 Q1 to 2016 Q4 disaggregated by State.

The top plot in Figure 10.2 shows the total number of visitor nights for the whole of Australia, while the plots below show the data disaggregated by state. These reveal diverse and rich dynamics at the aggregate national level, and the first level of disaggregation for each state. The `aggts()` function extracts time series from an `hts` object for any level of aggregation.

The plots in Figure 10.3 show the bottom-level time series, namely the visitor nights for each zone. These help us visualise the diverse individual dynamics within each zone, and assist in identifying unique and important time series. Notice, for example, the coastal WAU zone which shows significant growth over the last few years.

```
library(tidyverse)
cols <- sample(scales::hue_pal(h=c(15,375),
  c=100,l=65,h.start=0,direction = 1)(NCOL(visnights)))
as_tibble(visnights) %>%
  gather(Zone) %>%
  mutate(Date = rep(time(visnights), NCOL(visnights)),
         State = str_sub(Zone,1,3)) %>%
  ggplot(aes(x=Date, y=value, group=Zone, colour=Zone)) +
    geom_line() +
    facet_grid(State~., scales="free_y") +
    xlab("Year") + ylab("millions") +
    ggtitle("Visitor nights by Zone") +
    scale_colour_manual(values = cols)
```

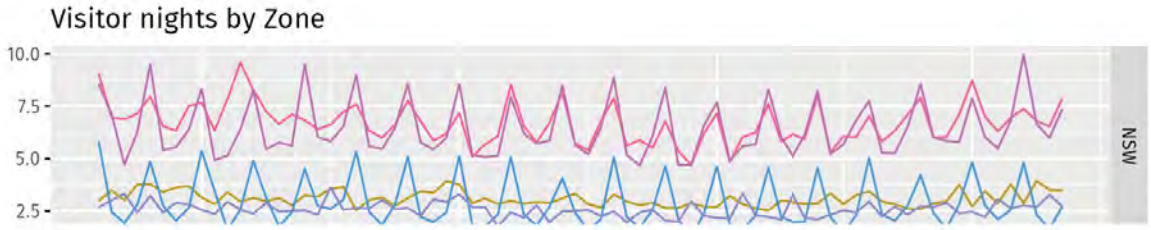


Figure 10.3: Australian domestic visitor nights over the period 1998 Q1 to 2016 Q4 disaggregated by Zones.

To produce this figure, we are using various functions from the **tidyverse** collection of packages. The details are beyond the scope of this book, but there are many good online resources available to learn how to use these packages.

Bibliography

Wickramasuriya, S. L., Athanasopoulos, G., & Hyndman, R. J. (2019). Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. *Journal of the American Statistical Association*, *114*(526), 804–819. [DOI]

10.2 Grouped time series

Grouped time series involve more general aggregation structures than hierarchical time series. With grouped time series, the structure does not naturally disaggregate in a unique hierarchical manner, and often the disaggregating factors are both nested and crossed. For example, we could further disaggregate all geographic levels of the Australian tourism data by purpose of travel (such as holidays, business, etc.). So we could consider visitors nights split by purpose of travel for the whole of Australia, and for each state, and for each zone. Then we describe the structure as involving the purpose of travel "crossed" with the geographic hierarchy.

Figure 10.4 shows a $K = 2$-level grouped structure. At the top of the grouped structure is the Total, the most aggregate level of the data, again represented by y_t. The Total can be disaggregated by attributes (A, B) forming series $y_{\mathrm{A},t}$ and $y_{\mathrm{B},t}$, or by attributes (X, Y) forming series $y_{\mathrm{X},t}$ and $y_{\mathrm{Y},t}$. At the bottom level, the data are disaggregated by both attributes.

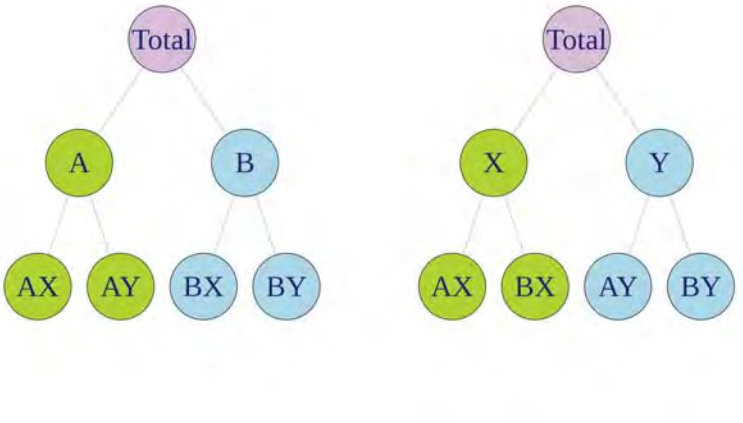


Figure 10.4: Alternative representations of a two level grouped structure.

This example shows that there are alternative aggregation paths for grouped structures. For any time t, as with the hierarchical structure,

$$y_t = y_{\mathrm{AX},t} + y_{\mathrm{AY},t} + y_{\mathrm{BX},t} + y_{\mathrm{BY},t}.$$

However, for the first level of the grouped structure,

$$y_{\text{A},t} = y_{\text{AX},t} + y_{\text{AY},t} \qquad y_{\text{B},t} = y_{\text{BX},t} + y_{\text{BY},t} \tag{10.4}$$

but also

$$y_{\text{X},t} = y_{\text{AX},t} + y_{\text{BX},t} \qquad y_{\text{Y},t} = y_{\text{AY},t} + y_{\text{BY},t}. \tag{10.5}$$

These equalities can again be represented by the $n \times m$ summing matrix $\boldsymbol{S}$. The total number of series is $n = 9$ with $m = 4$ series at the bottom-level. For the grouped structure in Figure 10.4 we write

$$\begin{bmatrix} y_t \\ y_{\text{A},t} \\ y_{\text{B},t} \\ y_{\text{X},t} \\ y_{\text{Y},t} \\ y_{\text{AX},t} \\ y_{\text{AY},t} \\ y_{\text{BX},t} \\ y_{\text{BY},t} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{\text{AX},t} \\ y_{\text{AY},t} \\ y_{\text{BX},t} \\ y_{\text{BY},t} \end{bmatrix},$$

or

$$\boldsymbol{y}_t = \boldsymbol{S}\boldsymbol{b}_t,$$

where the second and third rows of $\boldsymbol{S}$ represent (10.4) and the fourth and fifth rows represent (10.5).

Grouped time series can sometimes be thought of as hierarchical time series that do not impose a unique hierarchical structure, in the sense that the order by which the series can be grouped is not unique.

Example: Australian prison population

The top row of Figure 10.5 shows the total number of prisoners in Australia over the period 2005 Q1 to 2016 Q4. This represents the top-level series in the grouping structure. The rest of the panels show the prison population disaggregated by (i) state[21] (ii) legal status, whether prisoners have already been sentenced or are in remand waiting for a sentence, and (iii) gender. In this example, the three factors are crossed, but none are nested within the others.

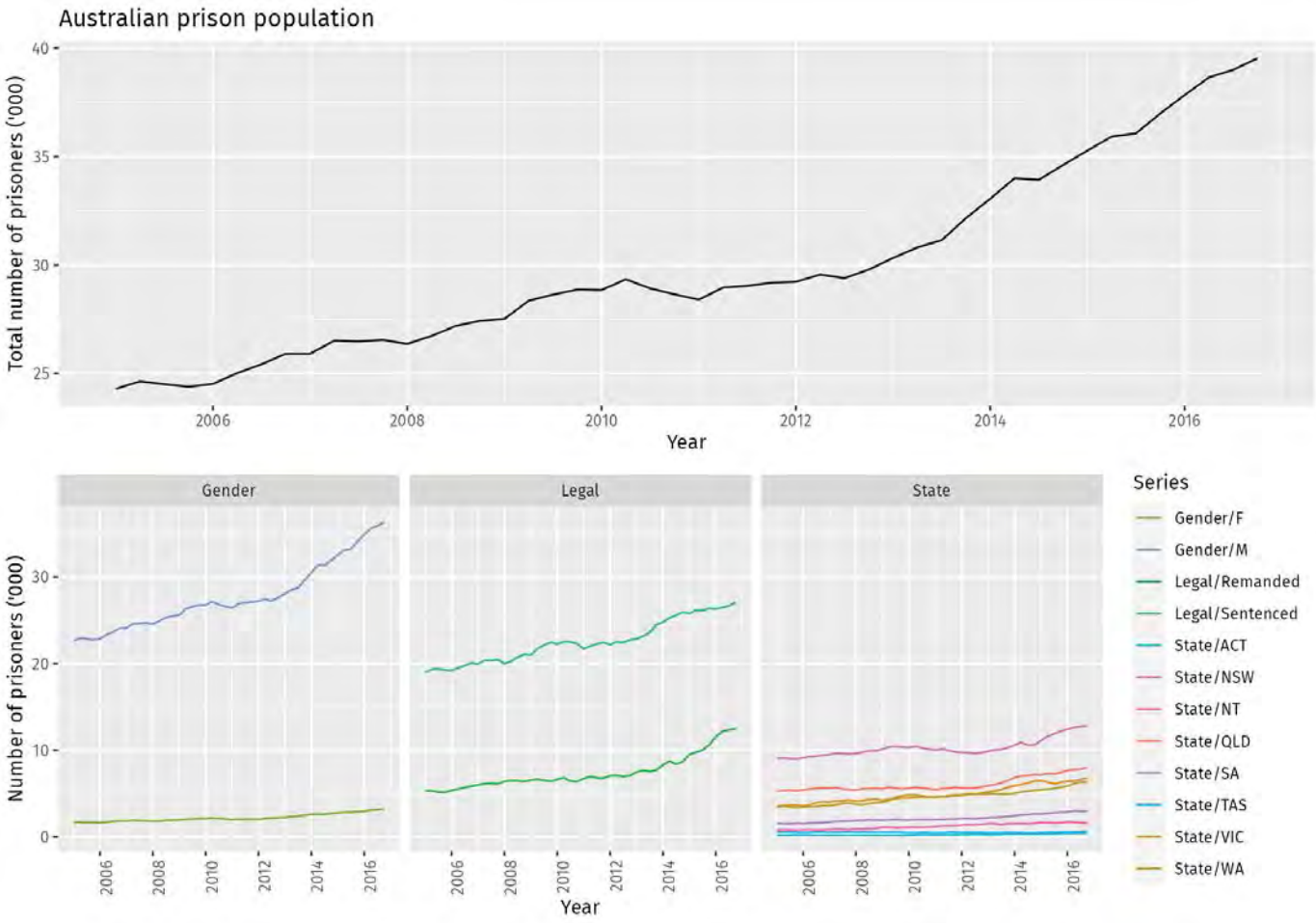


Figure 10.5: Total Australian quarterly adult prison population, disaggregated by state, by legal status and by gender.

To create a grouped time series, we use the `gts()` function. Similar to the `hts()` function, inputs to the `gts()` function are the bottom-level time series and information about the grouping structure. `prison` is a time series matrix containing the bottom-level time series. The information about the grouping structure can be passed in using the `characters` input. (An alternative is to be more explicit about the labelling of the series and use the `groups` input.)

```
prison.gts <- gts(prison/1e3, characters = c(3,1,9),
  gnames = c("State", "Gender", "Legal",
             "State*Gender", "State*Legal",
             "Gender*Legal"))
```

One way to plot the main groups is as follows.

```
prison.gts %>% aggts(level=0:3) %>% autoplot()
```

But with a little more work, we can construct Figure 10.5 using the following code.

```
p1 <- prison.gts %>% aggts(level=0) %>%
  autoplot() + ggtitle("Australian prison population") +
    xlab("Year") + ylab("Total number of prisoners ('000)")
groups <- aggts(prison.gts, level=1:3)
cols <- sample(scales::hue_pal(h=c(15,375),
          c=100,l=65,h.start=0,direction = 1)(NCOL(groups)))
p2 <- as_tibble(groups) %>%
  gather(Series) %>%
  mutate(Date = rep(time(groups), NCOL(groups)),
         Group = str_extract(Series, "([A-Za-z ]*)")) %>%
  ggplot(aes(x=Date, y=value, group=Series, colour=Series)) +
    geom_line() +
    xlab("Year") + ylab("Number of prisoners ('000)") +
    scale_colour_manual(values = cols) +
    facet_grid(.~Group, scales="free_y") +
    scale_x_continuous(breaks=seq(2006,2016,by=2)) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
gridExtra::grid.arrange(p1, p2, ncol=1)
```

Plots of other group combinations can be obtained similarly. Figure 10.6 shows the Australian prison population disaggregated by all possible combinations of two attributes at a time. The top plot shows the prison population disaggregated by state and legal status, the middle panel shows the disaggregation by state and gender and the bottom panel shows the disaggregation by legal status and gender.

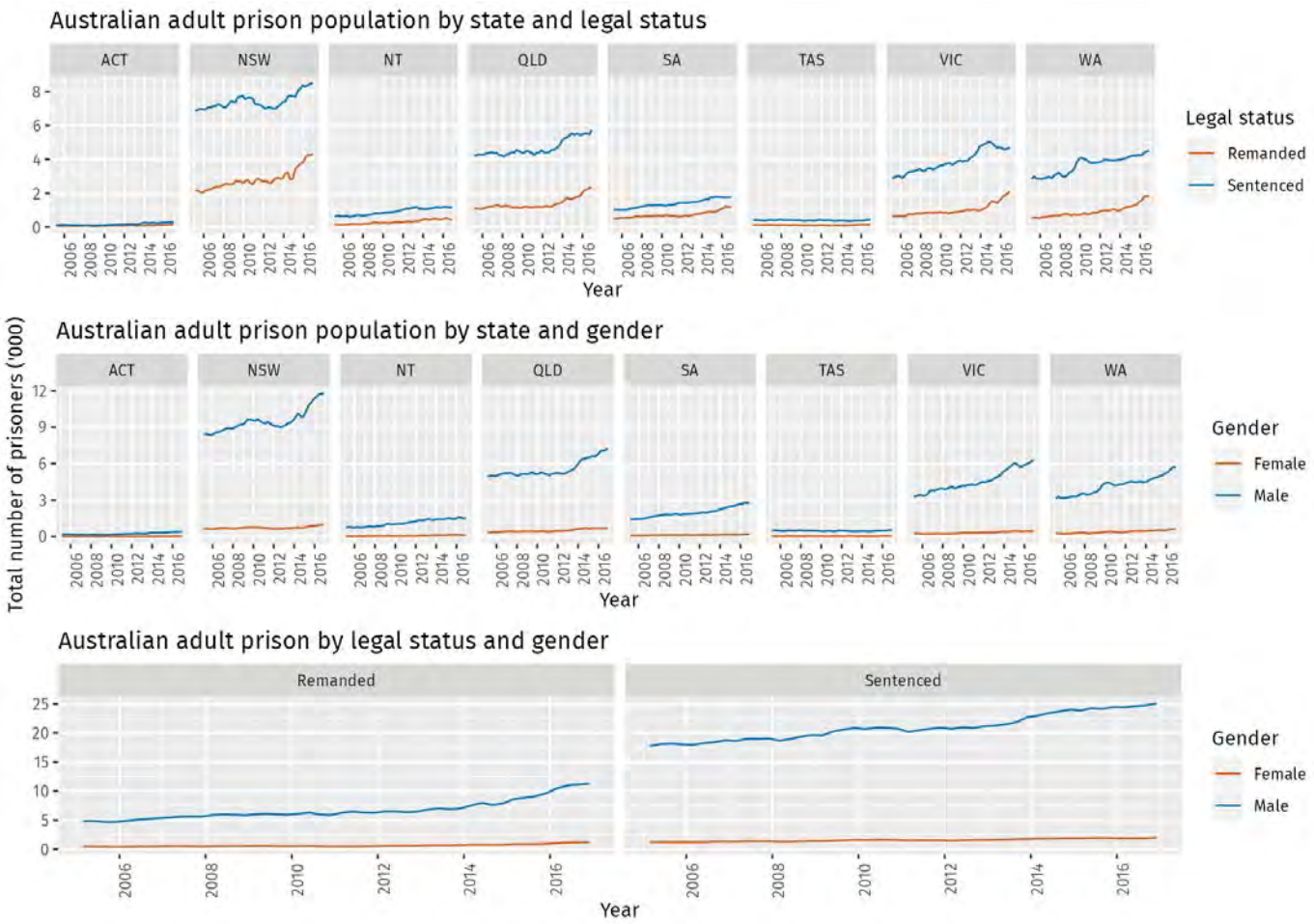


Figure 10.6: Australian adult prison population disaggregated by pairs of attributes.

Figure 10.7 shows the Australian adult prison population disaggregated by all three attributes: state, legal status and gender. These form the bottom-level series of the grouped structure.

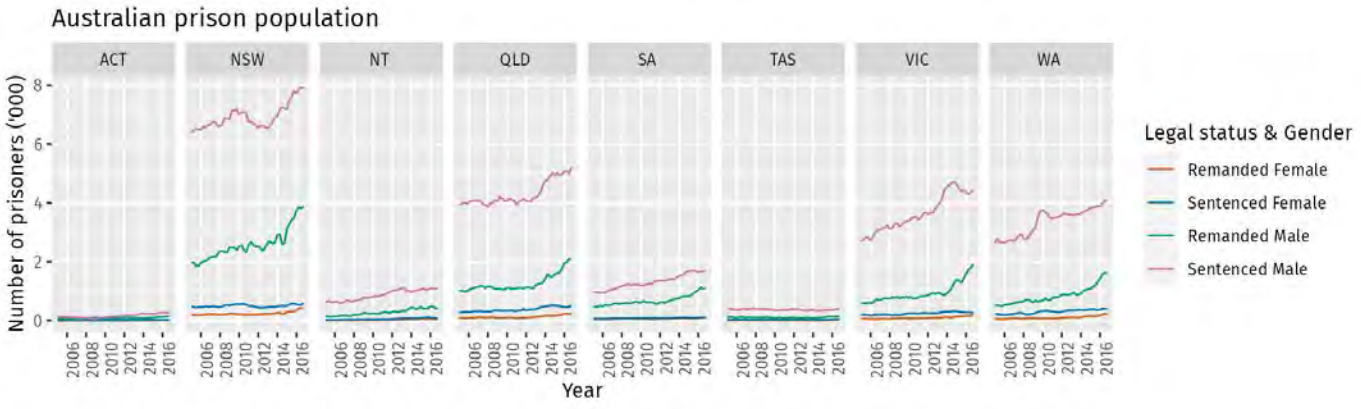


Figure 10.7: Bottom-level time series for the Australian adult prison population, grouped by state, legal status and gender.

21. Australia comprises eight geographic areas six states and two territories: Australian Capital Territory, New South Wales, Northern Territory, Queensland, South Australia, Tasmania, Victoria, Western Australia. In this example we

10.3 The bottom-up approach

A simple method for generating coherent forecasts is the bottom-up approach. This approach involves first generating forecasts for each series at the bottom-level, and then summing these to produce forecasts for all the series in the structure.

For example, for the hierarchy of Figure 10.1, we first generate h-step-ahead forecasts for each of the bottom-level series:

$$\hat{y}_{\text{AA},h}, \quad \hat{y}_{\text{AB},h}, \quad \hat{y}_{\text{AC},h}, \quad \hat{y}_{\text{BA},h} \quad \text{and} \quad \hat{y}_{\text{BB},h}.$$

(We have simplified the previously used notation of $\hat{y}_{T+h|T}$ for brevity.) Summing these, we get h-step-ahead coherent forecasts for the rest of the series:

$$\begin{aligned} \tilde{y}_h &= \hat{y}_{\text{AA},h} + \hat{y}_{\text{AB},h} + \hat{y}_{\text{AC},h} + \hat{y}_{\text{BA},h} + \hat{y}_{\text{BB},h}, \\ \tilde{y}_{\text{A},h} &= \hat{y}_{\text{AA},h} + \hat{y}_{\text{AB},h} + \hat{y}_{\text{AC},h}, \\ \text{and} \quad \tilde{y}_{\text{B},h} &= \hat{y}_{\text{BA},h} + \hat{y}_{\text{BB},h}. \end{aligned}$$

(In this chapter, we will use the "tilde" notation to indicate coherent forecasts.) As in Equation (10.3), we can employ the summing matrix here and write

$$\begin{bmatrix} \tilde{y}_h \\ \tilde{y}_{\text{A},h} \\ \tilde{y}_{\text{B},h} \\ \tilde{y}_{\text{AA},h} \\ \tilde{y}_{\text{AB},h} \\ \tilde{y}_{\text{AC},h} \\ \tilde{y}_{\text{BA},h} \\ \tilde{y}_{\text{BB},h} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_{\text{AA},h} \\ \hat{y}_{\text{AB},h} \\ \hat{y}_{\text{AC},h} \\ \hat{y}_{\text{BA},h} \\ \hat{y}_{\text{BB},h} \end{bmatrix}.$$

Using more compact notation, the bottom-up approach can be represented as

$$\tilde{\boldsymbol{y}}_h = \boldsymbol{S}\hat{\boldsymbol{b}}_h,$$

where $\tilde{\boldsymbol{y}}_h$ is an n-dimensional vector of coherent h-step-ahead forecasts, and $\hat{\boldsymbol{b}}_h$ is an m-dimensional vector of h-step-ahead forecasts for each of the bottom-level series.

An advantage of this approach is that we are forecasting at the bottom-level of a structure, and therefore no information is lost due to aggregation. On the other hand, bottom-level data can be quite noisy and more challenging to model and forecast.

The hts package for R

Forecasts can be produced using the `forecast()` function applied to objects created by `hts()` or `gts()`. The **hts** package has three in-built options to produce forecasts: ETS models, ARIMA models or random walks; these are controlled by the `fmethod` argument. It also use several methods for producing coherent forecasts, controlled by the `method` argument.

For example, suppose we wanted bottom-up forecasts using ARIMA models applied to the prison data. Then we would use

```
forecast(prison.gts, method="bu", fmethod="arima")
```

which will apply the `auto.arima()` function to every bottom-level series in our collection of time series. Similarly, ETS models would be used if `fmethod="ets"` was used.

10.4 Top-down approaches

Top-down approaches only work with strictly hierarchical aggregation structures, and not with grouped structures. They involve first generating forecasts for the Total series y_t, and then disaggregating these down the hierarchy.

We let $p_1, \dots, p_m$ be a set of disaggregation proportions which dictate how the forecasts of the Total series are to be distributed to obtain forecasts for each series at the bottom-level of the structure. For example, for the hierarchy of Figure 10.1 using proportions $p_1, \dots, p_5$ we get,

$$\tilde{y}_{\text{AA},t} = p_1\hat{y}_t, \quad \tilde{y}_{\text{AB},t} = p_2\hat{y}_t, \quad \tilde{y}_{\text{AC},t} = p_3\hat{y}_t, \quad \tilde{y}_{\text{BA},t} = p_4\hat{y}_t \quad \text{and} \quad \tilde{y}_{\text{BB},t} = p_5\hat{y}_t.$$

Using matrix notation we can stack the set of proportions in a m-dimensional vector $\boldsymbol{p} = (p_1, \dots, p_m)'$ and write

$$\tilde{\boldsymbol{b}}_t = \boldsymbol{p}\hat{y}_t.$$

Once the bottom-level h-step-ahead forecasts have been generated, these are aggregated to generate coherent forecasts for the rest of the series. In general, for a specified set of proportions, top-down approaches can be represented as

$$\tilde{\boldsymbol{y}}_h = \boldsymbol{S}\boldsymbol{p}\hat{y}_t.$$

The two most common top-down approaches specify disaggregation proportions based on the historical proportions of the data. These performed well in the study of Gross & Sohl (1990).

Average historical proportions

$$p_j = \frac{1}{T}\sum_{t=1}^{T}\frac{y_{j,t}}{y_t}$$

for $j = 1, \dots, m$. Each proportion p_j reflects the average of the historical proportions of the bottom-level series $y_{j,t}$ over the period $t = 1, \dots, T$ relative to the total aggregate y_t.

This approach is implemented in the `forecast()` function by setting `method="tdgsa"`, where `tdgsa` stands for "top-down Gross-Sohl method A".

Proportions of the historical averages

$$p_j = \sum_{t=1}^{T} \frac{y_{j,t}}{T} \Big/ \sum_{t=1}^{T} \frac{y_t}{T}$$

for $j = 1, \ldots, m$. Each proportion p_j captures the average historical value of the bottom-level series $y_{j,t}$ relative to the average value of the total aggregate y_t.

This approach is implemented in the `forecast()` function by setting `method="tdgsf"`, where `tdgsf` stands for "top-down Gross-Sohl method F".

A convenient attribute of such top-down approaches is their simplicity. One only needs to model and generate forecasts for the most aggregated top-level series. In general, these approaches seem to produce quite reliable forecasts for the aggregate levels and they are useful with low count data. On the other hand, one disadvantage is the loss of information due to aggregation. Using such top-down approaches, we are unable to capture and take advantage of individual series characteristics such as time dynamics, special events, etc.

Forecast proportions

Because historical proportions used for disaggregation do not take account of how those proportions may change over time, top-down approaches based on historical proportions tend to produce less accurate forecasts at lower levels of the hierarchy than bottom-up approaches. To address this issue, proportions based on forecasts rather than historical data can be used (Athanasopoulos, Ahmed, & Hyndman, 2009).

Consider a one level hierarchy. We first generate h-step-ahead forecasts for all of the series. We don't use these forecasts directly, and they are not coherent (they don't add up correctly). Let's call these "initial" forecasts. We calculate the proportion of each h-step-ahead initial forecast at the bottom level, to the aggregate of all the h-step-ahead initial forecasts at this level. We refer to these as the forecast proportions, and we use them to disaggregate the top-level h-step-ahead initial forecast in order to generate coherent forecasts for the whole of the hierarchy.

For a K-level hierarchy, this process is repeated for each node, going from the top to the bottom level. Applying this process leads to the following general rule for obtaining the forecast proportions:

$$p_j = \prod_{\ell=0}^{K-1} \frac{\hat{y}_{j,h}^{(\ell)}}{\hat{S}_{j,h}^{(\ell+1)}}$$

where $j = 1, 2, \ldots, m$, $\hat{y}_{j,h}^{(\ell)}$ is the h-step-ahead initial forecast of the series that corresponds to the node which is ℓ levels above j, and $\hat{S}_{j,h}^{(\ell)}$ is the sum of the h-step-ahead initial forecasts below the node that is ℓ levels above node j and are directly connected to that node. These forecast proportions disaggregate the h-step-ahead initial forecast of the Total series to get h-step-ahead coherent forecasts of the bottom-level series.

We will use the hierarchy of Figure 10.1 to explain this notation and to demonstrate how this general rule is reached. Assume we have generated initial forecasts for each series in the hierarchy. Recall that for the top-level "Total" series, $\tilde{y}_h = \hat{y}_h$, for any top-down approach. Here are some examples using the above notation:

- $\hat{y}_{\mathrm{A},h}^{(1)} = \hat{y}_{\mathrm{B},h}^{(1)} = \hat{y}_h = \tilde{y}_h$;
- $\hat{y}_{\mathrm{AA},h}^{(1)} = \hat{y}_{\mathrm{AB},h}^{(1)} = \hat{y}_{\mathrm{AC},h}^{(1)} = \hat{y}_{\mathrm{A},h}$;
- $\hat{y}_{\mathrm{AA},h}^{(2)} = \hat{y}_{\mathrm{AB},h}^{(2)} = \hat{y}_{\mathrm{AC},h}^{(2)} = \hat{y}_{\mathrm{BA},h}^{(2)} = \hat{y}_{\mathrm{BB},h}^{(2)} = \hat{y}_h = \tilde{y}_h$;
- $\hat{S}_{\mathrm{AA},h}^{(1)} = \hat{S}_{\mathrm{AB},h}^{(1)} = \hat{S}_{\mathrm{AC},h}^{(1)} = \hat{y}_{\mathrm{AA},h} + \hat{y}_{\mathrm{AB},h} + \hat{y}_{\mathrm{AC},h}$;
- $\hat{S}_{\mathrm{AA},h}^{(2)} = \hat{S}_{\mathrm{AB},h}^{(2)} = \hat{S}_{\mathrm{AC},h}^{(2)} = \hat{S}_{\mathrm{A},h}^{(1)} = \hat{S}_{\mathrm{B},h}^{(1)} = \hat{S}_h = \hat{y}_{\mathrm{A},h} + \hat{y}_{\mathrm{B},h}$.

Moving down the farthest left branch of the hierarchy, coherent forecasts are given by

$$\tilde{y}_{\mathrm{A},h} = \left(\frac{\hat{y}_{\mathrm{A},h}}{\hat{S}_{\mathrm{A},h}^{(1)}}\right)\tilde{y}_h = \left(\frac{\hat{y}_{\mathrm{AA},h}^{(1)}}{\hat{S}_{\mathrm{AA},h}^{(2)}}\right)\tilde{y}_h$$

and

$$\tilde{y}_{\mathrm{AA},h} = \left(\frac{\hat{y}_{\mathrm{AA},h}}{\hat{S}_{\mathrm{AA},h}^{(1)}}\right)\tilde{y}_{\mathrm{A},h} = \left(\frac{\hat{y}_{\mathrm{AA},h}}{\hat{S}_{\mathrm{AA},h}^{(1)}}\right)\left(\frac{\hat{y}_{\mathrm{AA},h}^{(1)}}{\hat{S}_{\mathrm{AA},h}^{(2)}}\right)\tilde{y}_h.$$

Consequently,

$$p_1 = \left(\frac{\hat{y}_{\text{AA},h}}{\hat{S}^{(1)}_{\text{AA},h}}\right)\left(\frac{\hat{y}^{(1)}_{\text{AA},h}}{\hat{S}^{(2)}_{\text{AA},h}}\right).$$

The other proportions can be obtained similarly.

One disadvantage of all top-down approaches, including this one, is that it does not produce unbiased coherent forecasts (Hyndman, Ahmed, Athanasopoulos, & Shang, 2011).

This approach is implemented in the `forecast()` function by setting `method="tdfp"`, where `tdfp` stands for "top-down forecast proportions".

Bibliography

Athanasopoulos, G., Ahmed, R. A., & Hyndman, R. J. (2009). Hierarchical forecasts for Australian domestic tourism. *International Journal of Forecasting*, *25*, 146–166. [DOI]

Gross, C. W., & Sohl, J. E. (1990). Disaggregation methods to expedite product line forecasting. *Journal of Forecasting*, *9*, 233–254. [DOI]

Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G., & Shang, H. L. (2011). Optimal combination forecasts for hierarchical time series. *Computational Statistics and Data Analysis*, *55*(9), 2579–2589. [DOI]

10.5 Middle-out approach

The middle-out approach combines bottom-up and top-down approaches. First, a "middle level" is chosen and forecasts are generated for all the series at this level. For the series above the middle level, coherent forecasts are generated using the bottom-up approach by aggregating the "middle-level" forecasts upwards. For the series below the "middle level", coherent forecasts are generated using a top-down approach by disaggregating the "middle level" forecasts downwards.

This approach is implemented in the `forecast()` function by setting `method="mo"` and by specifying the appropriate middle level via the `level` argument. For the top-down disaggregation below the middle level, the top-down forecast proportions method is used.

10.6 Mapping matrices

All of the methods considered so far can be expressed using a common notation.

Suppose we forecast all series independently, ignoring the aggregation constraints. We call these the **base forecasts** and denote them by $\hat{\boldsymbol{y}}_h$ where h is the forecast horizon. They are stacked in the same order as the data $\boldsymbol{y}_t$.

Then all forecasting approaches for either hierarchical or grouped structures can be represented as

$$\tilde{\boldsymbol{y}}_h = \boldsymbol{S}\boldsymbol{G}\hat{\boldsymbol{y}}_h, \tag{10.6}$$

where $\boldsymbol{G}$ is a matrix that maps the base forecasts into the bottom-level, and the summing matrix $\boldsymbol{S}$ sums these up using the aggregation structure to produce a set of coherent forecasts $\tilde{\boldsymbol{y}}_h$.

The $\boldsymbol{G}$ matrix is defined according to the approach implemented. For example if the bottom-up approach is used to forecast the hierarchy of Figure 10.1, then

$$\boldsymbol{G} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Notice that $\boldsymbol{G}$ contains two partitions. The first three columns zero out the base forecasts of the series above the bottom-level, while the m-dimensional identity matrix picks only the base forecasts of the bottom-level. These are then summed by the $\boldsymbol{S}$ matrix.

If any of the top-down approaches were used then

$$\boldsymbol{G} = \begin{bmatrix} p_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ p_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ p_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ p_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ p_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The first column includes the set of proportions that distribute the base forecasts of

the top-level to the bottom-level. These are then summed up the hierarchy by the $\boldsymbol{S}$ matrix. The rest of the columns zero out the base forecasts below the highest level of aggregation.

For a middle out approach, the $\boldsymbol{G}$ matrix will be a combination of the above two. Using a set of proportions, the base forecasts of some pre-chosen level will be disaggregated to the bottom-level, all other base forecasts will be zeroed out, and the bottom-level forecasts will then summed up the hierarchy via the summing matrix.

Forecast reconciliation

We can rewrite Equation (10.6) as

$$\tilde{\boldsymbol{y}}_h = \boldsymbol{P}\hat{\boldsymbol{y}}_h, \tag{10.7}$$

where $\boldsymbol{P} = \boldsymbol{SG}$ is a "projection" or a "reconciliation matrix". That is, it takes the incoherent base forecasts $\hat{\boldsymbol{y}}_h$, and reconciles them to produce coherent forecasts $\tilde{\boldsymbol{y}}_h$.

In the methods discussed so far, no real reconciliation has been done because the methods have been based on forecasts from a single level of the aggregation structure, which have either been aggregated or disaggregated to obtain forecasts at all other levels. However, in general, we could use other $\boldsymbol{G}$ matrices, and then $\boldsymbol{P}$ will be combining and reconciling all the base forecasts in order to produce coherent forecasts.

In fact, we can find the optimal $\boldsymbol{G}$ matrix to give the most accurate reconciled forecasts.

10.7 The optimal reconciliation approach

Optimal forecast reconciliation will occur if we can find the $\boldsymbol{G}$ matrix which minimises the forecast error of the set of coherent forecasts. We present here a simplified summary of the approach. More details are provided in Wickramasuriya et al. (2019).

Suppose we generate coherent forecasts using Equation (10.6), repeated here for convenience:

$$\tilde{\boldsymbol{y}}_h = \boldsymbol{SG}\hat{\boldsymbol{y}}_h.$$

First we want to make sure we have unbiased forecasts. If the base forecasts $\hat{\boldsymbol{y}}_h$ are unbiased, then the coherent forecasts $\tilde{\boldsymbol{y}}_h$ will be unbiased provided $\boldsymbol{SGS} = \boldsymbol{S}$ (Hyndman et al., 2011). This provides a constraint on the matrix $\boldsymbol{G}$. Interestingly, no top-down method satisfies this constraint, so all top-down methods are biased.

Next we need to find the error in our forecasts. Wickramasuriya et al. (2019) show that the variance-covariance matrix of the h-step-ahead coherent forecast errors is given by

$$\boldsymbol{V}_h = \text{Var}[\boldsymbol{y}_{T+h} - \tilde{\boldsymbol{y}}_h] = \boldsymbol{SGW}_h\boldsymbol{G}'\boldsymbol{S}'$$

where $\boldsymbol{W}_h = \text{Var}[(\boldsymbol{y}_{T+h} - \hat{\boldsymbol{y}}_h)]$ is the variance-covariance matrix of the corresponding base forecast errors.

The objective is to find a matrix $\boldsymbol{G}$ that minimises the error variances of the coherent forecasts. These error variances are on the diagonal of the matrix $\boldsymbol{V}_h$, and so the sum of all the error variances is given by the trace of the matrix $\boldsymbol{V}_h$. Wickramasuriya et al. (2019) show that the matrix $\boldsymbol{G}$ which minimises the trace of $\boldsymbol{V}_h$ such that $\boldsymbol{SGS} = \boldsymbol{S}$, is given by

$$\boldsymbol{G} = (\boldsymbol{S}'\boldsymbol{W}_h^{-1}\boldsymbol{S})^{-1}\boldsymbol{S}'\boldsymbol{W}_h^{-1}.$$

Therefore, the optimal reconciled forecasts are given by

$$\tilde{\boldsymbol{y}}_h = \boldsymbol{S}(\boldsymbol{S}'\boldsymbol{W}_h^{-1}\boldsymbol{S})^{-1}\boldsymbol{S}'\boldsymbol{W}_h^{-1}\hat{\boldsymbol{y}}_h. \tag{10.8}$$

We refer to this as the "MinT" (or Minimum Trace) estimator.

To use this in practice, we need to estimate $\boldsymbol{W}_h$, the forecast error variance of the h-step-ahead base forecasts. This can be difficult, and so we provide four simplifying approximations which have been shown to work well in both simulations and in practice.

1. Set $\boldsymbol{W}_h = k_h \boldsymbol{I}$ for all h, where $k_h > 0$.[22] This is the most simplifying assumption to make, and means that $\boldsymbol{G}$ is independent of the data, providing substantial computational savings. The disadvantage, however, is that this specification does not account for the differences in scale between the levels of the structure, or for relationships between series. This approach is implemented in the `forecast()` function by setting `method = "comb"` and `weights = "ols"`.

   The weights here are referred to as OLS (ordinary least squares) because setting $\boldsymbol{W}_h = k_h \boldsymbol{I}$ in (10.8) gives the least squares estimator we introduced in Section 5.7 with $\boldsymbol{X} = \boldsymbol{S}$ and $\boldsymbol{y} = \hat{\boldsymbol{y}}$.

2. Set $\boldsymbol{W}_h = k_h \text{diag}(\hat{\boldsymbol{W}}_1)$ for all h, where $k_h > 0$,

$$\hat{\boldsymbol{W}}_1 = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{e}_t \boldsymbol{e}_t',$$

   and $\boldsymbol{e}_t$ is an n-dimensional vector of residuals of the models that generated the base forecasts stacked in the same order as the data. The approach is implemented by setting `method = "comb"` and `weights = "wls"`.

   This specification scales the base forecasts using the variance of the residuals and it is therefore referred to as the WLS (weighted least squares) estimator using *variance scaling*.

3. Set $\boldsymbol{W}_h = k_h \boldsymbol{\Lambda}$ for all h, where $k_h > 0$, $\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{S}\boldsymbol{1})$, and $\boldsymbol{1}$ is a unit vector of dimension m (the number of bottom-level series). This specification assumes that the bottom-level base forecast errors each have variance k_h and are uncorrelated between nodes. Hence each element of the diagonal $\boldsymbol{\Lambda}$ matrix contains the number of forecast error variances contributing to each node. This estimator only depends on the structure of the aggregations, and not on the actual data. It is therefore referred to as *structural scaling*. Applying the structural scaling specification is particularly useful in cases where residuals are not available, and so variance scaling cannot be applied; for example, in cases where the base forecasts are generated by judgmental forecasting (Chapter 4). This approach is implemented by setting `method="comb"` and `weights = "nseries"`.

4. Set $\boldsymbol{W}_h = k_h \boldsymbol{W}_1$ for all h, where $k_h > 0$. Here we only assume that the error covariance matrices are proportional to each other, and we directly estimate the full one-step covariance matrix $\boldsymbol{W}_1$. The most obvious and simple way would be to use the sample covariance. This is implemented by setting `method = "comb"`, `weights = "mint"`, and `covariance = "sam"`.

   However, for cases where the number of bottom-level series m is large compared to the length of the series T, this is not a good estimator. Instead we use a shrinkage estimator which shrinks the sample covariance to a diagonal matrix. This is implemented by setting `method = "comb"`, `weights = "mint"`, and `covariance = "shr"`.

In summary, unlike any other existing approach, the optimal reconciliation forecasts are generated using all the information available within a hierarchical or a grouped structure. This is important, as particular aggregation levels or groupings may reveal features of the data that are of interest to the user and are important to be modelled. These features may be completely hidden or not easily identifiable at other levels.

For example, consider the Australian tourism data introduced in Section 10.1, where the hierarchical structure followed the geographic division of a country into states and zones. Some coastal areas will be largely summer destinations, while some mountain regions may be winter destinations. These differences will be smoothed at the country level due to aggregation.

Example: Forecasting Australian prison population

We compute the forecasts for the Australian prison population, described in Section 10.2. Using the default arguments for the `forecast()` function, we compute coherent forecasts by the optimal reconciliation approach with the WLS estimator using variance scaling.

```
prisonfc <- forecast(prison.gts)
```

To obtain forecasts for each level of aggregation, we can use the `aggts()` function. For example, to calculate forecasts for the overall total prison population, and for the one-factor groupings (State, Gender and Legal Status), we use:

```
fcsts <- aggts(prisonfc, levels=0:3)
```

A simple plot is obtained using

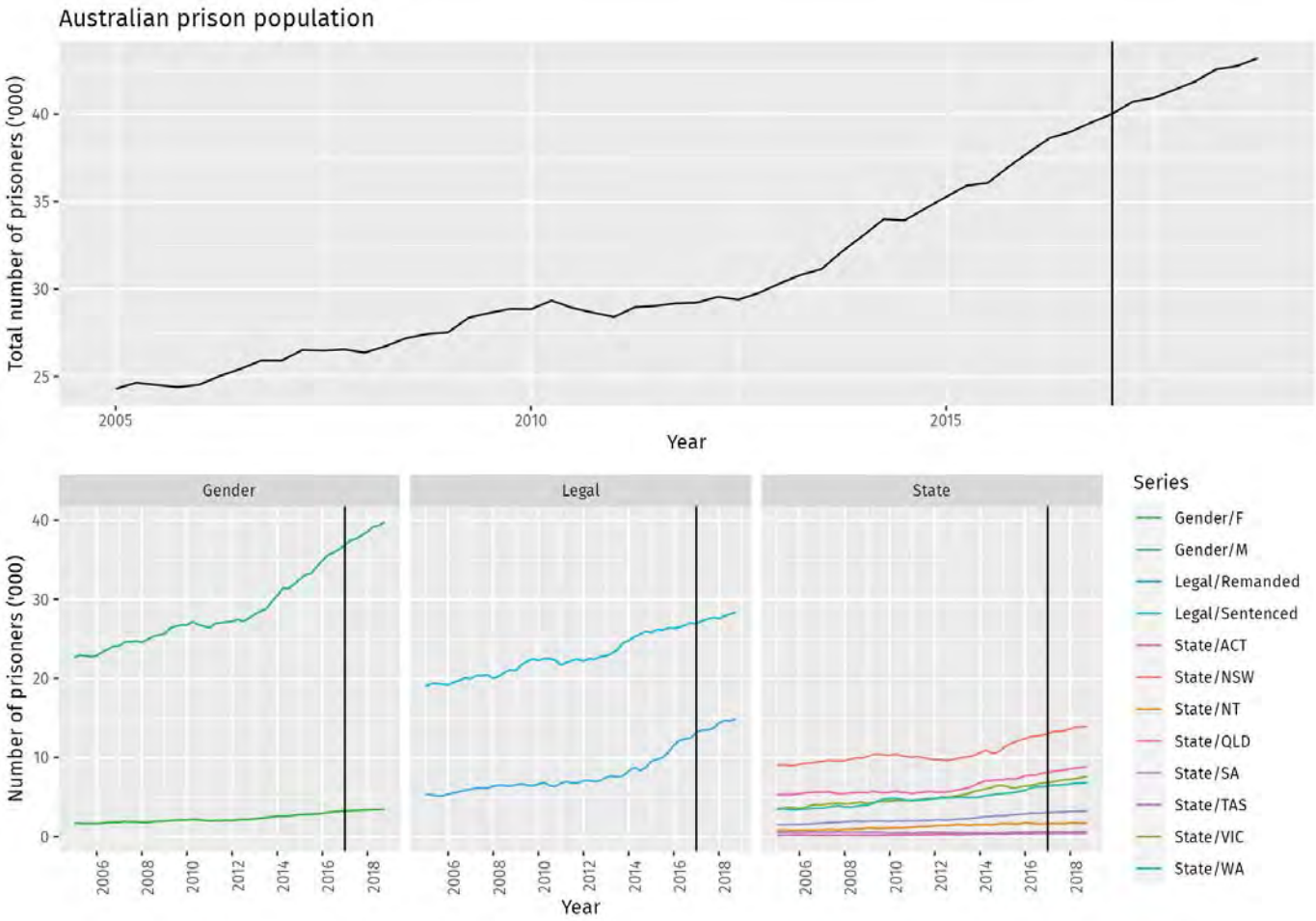


Figure 10.8: Coherent forecasts for the total Australian adult prison population and for the population grouped by state, by legal status and by gender.

Similar code was used to produce Figure 10.9. The left panel plots the coherent forecasts for interactions between states and gender. The right panel shows forecasts for the bottom-level series.

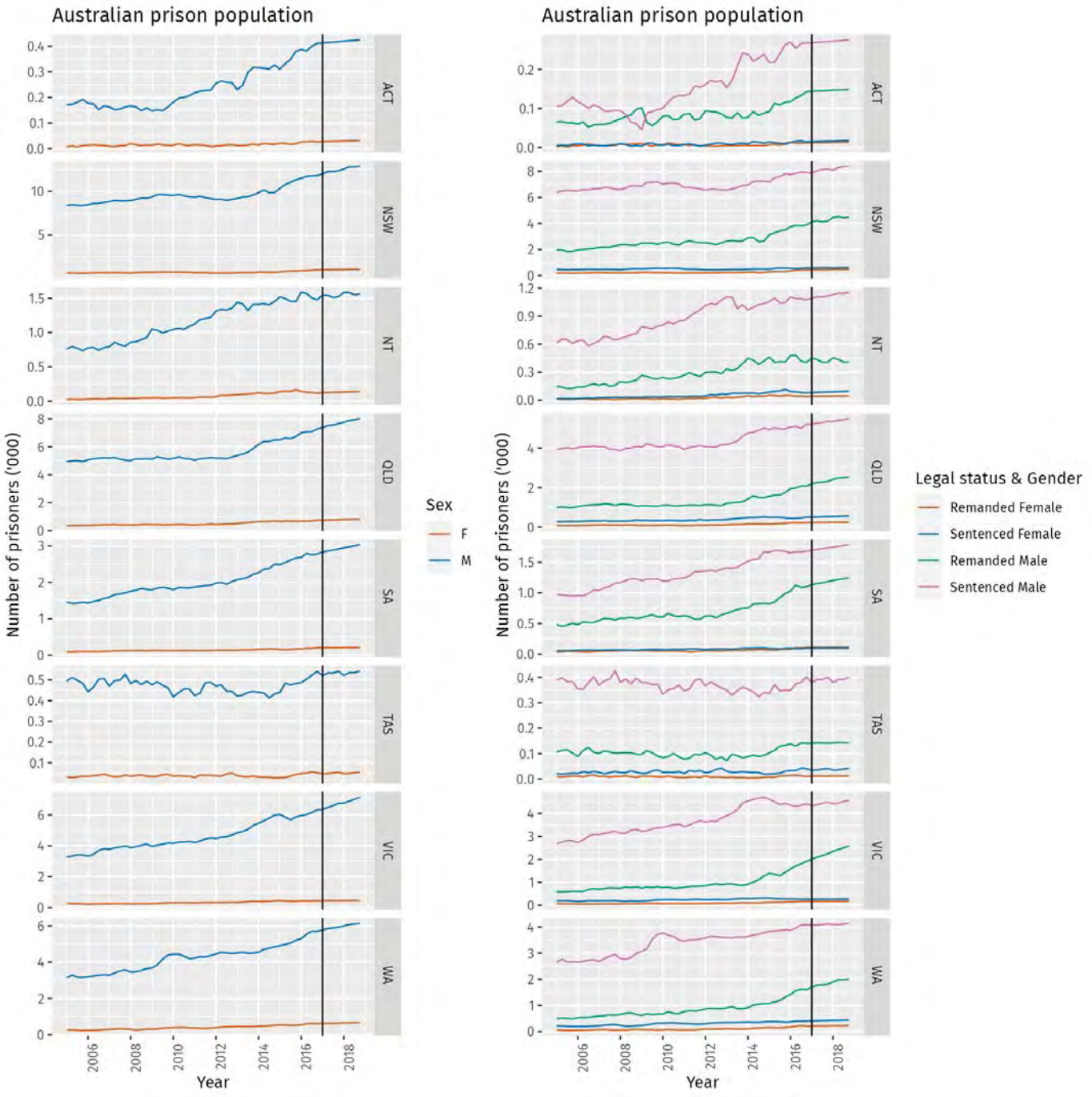


Figure 10.9: Coherent forecasts for the Australian adult prison population grouped by all interactions of attributes.

The `accuracy()` command is useful for evaluating the forecast accuracy across hierarchical or grouped structures. The following table summarises the accuracy of the bottom-up and the optimal reconciliation approaches, forecasting 2015 Q1 to 2016 Q4 as a test period.

The results show that the optimal reconciliation approach generates more accurate forecasts especially for the top level. In general, we find that as the optimal reconciliation approach uses information from all levels in the structure, it generates more accurate coherent forecasts than the other traditional alternatives which use limited information.

Table 10.1: Accuracy of Australian prison population forecasts for different groups of series.

	Bottom-up		Optimal	
	MAPE	MASE	MAPE	MASE
Total	5.32	1.84	3.08	1.06
State	7.59	1.88	7.62	1.85
Legal status	6.40	1.76	4.32	1.14
Gender	8.62	2.68	8.72	2.74
Bottom	15.82	2.23	15.25	2.16
All series	12.41	2.16	12.02	2.08

Bibliography

Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G., & Shang, H. L. (2011). Optimal combination forecasts for hierarchical time series. *Computational Statistics and Data Analysis*, *55*(9), 2579–2589. [DOI]

Wickramasuriya, S. L., Athanasopoulos, G., & Hyndman, R. J. (2019). Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. *Journal of the American Statistical Association*, *114*(526), 804–819. [DOI]

22. Note that k_h is a proportionality constant. It does not need to be estimated or specified here as it gets cancelled out in (10.8).↩

10.8 Exercises

1. Write out the $\boldsymbol{S}$ matrices for the Australian tourism hierarchy and the Australian prison grouped structure. Use the `smatrix` command to verify your answers.
2. Generate 8-step-ahead bottom-up forecasts using ARIMA models for the `visnights` Australian domestic tourism data. Plot the coherent forecasts by level and comment on their nature. Are you satisfied with these forecasts?
3. Model the aggregate series for Australian domestic tourism data `visnights` using an ARIMA model. Comment on the model. Generate and plot 8-step-ahead forecasts from the ARIMA model and compare these with the bottom-up forecasts generated in question 2 for the aggregate level.
4. Generate 8-step-ahead optimally reconciled coherent forecasts using ARIMA base forecasts for the `visnights` Australian domestic tourism data. Plot the coherent forecasts by level and comment on their nature. How and why are these different to the bottom-up forecasts generated in question 2 above.
5. Using the last two years of the `visnights` Australian domestic tourism data as a test set, generate bottom-up, top-down and optimally reconciled forecasts for this period and compare their accuracy.

10.9 Further reading

There are no other textbooks which cover hierarchical forecasting in any depth, so interested readers will need to tackle the original research papers for further information.

- Gross & Sohl (1990) provide a good introduction to the top-down approaches.
- The reconciliation methods were developed in a series of papers, which are best read in the following order: Hyndman et al. (2011), Athanasopoulos et al. (2009), Hyndman, Lee, & Wang (2016), Wickramasuriya et al. (2019).
- Athanasopoulos, Hyndman, Kourentzes, & Petropoulos (2017) extends the reconciliation approach to deal with temporal hierarchies.

Bibliography

Athanasopoulos, G., Ahmed, R. A., & Hyndman, R. J. (2009). Hierarchical forecasts for Australian domestic tourism. *International Journal of Forecasting*, *25*, 146–166. [DOI]

Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. *European Journal of Operational Research*, *262*(1), 60–74. [DOI]

Gross, C. W., & Sohl, J. E. (1990). Disaggregation methods to expedite product line forecasting. *Journal of Forecasting*, *9*, 233–254. [DOI]

Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G., & Shang, H. L. (2011). Optimal combination forecasts for hierarchical time series. *Computational Statistics and Data Analysis*, *55*(9), 2579–2589. [DOI]

Hyndman, R. J., Lee, A., & Wang, E. (2016). Fast computation of reconciled forecasts for hierarchical and grouped time series. *Computational Statistics and Data Analysis*, *97*, 16–32. [DOI]

Wickramasuriya, S. L., Athanasopoulos, G., & Hyndman, R. J. (2019). Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. *Journal of the American Statistical Association*, *114*(526), 804–819. [DOI]

11.1 Complex seasonality

So far, we have considered relatively simple seasonal patterns such as quarterly and monthly data. However, higher frequency time series often exhibit more complicated seasonal patterns. For example, daily data may have a weekly pattern as well as an annual pattern. Hourly data usually has three types of seasonality: a daily pattern, a weekly pattern, and an annual pattern. Even weekly data can be challenging to forecast as it typically has an annual pattern with seasonal period of $365.25/7 \approx 52.179$ on average.

Such multiple seasonal patterns are becoming more common with high frequency data recording. Further examples where multiple seasonal patterns can occur include call volume in call centres, daily hospital admissions, requests for cash at ATMs, electricity and water usage, and access to computer web sites.

Most of the methods we have considered so far are unable to deal with these seasonal complexities. Even the `ts` class in R can only handle one type of seasonality, which is usually assumed to take integer values.

To deal with such series, we will use the `msts` class which handles multiple seasonality time series. This allows you to specify all of the frequencies that might be relevant. It is also flexible enough to handle non-integer frequencies.

Despite this flexibility, we don't necessarily want to include all of these frequencies — just the ones that are likely to be present in the data. For example, if we have only 180 days of data, we may ignore the annual seasonality. If the data are measurements of a natural phenomenon (e.g., temperature), we can probably safely ignore any weekly seasonality.

The top panel of Figure 11.1 shows the number of retail banking call arrivals per 5-minute interval between 7:00am and 9:05pm each weekday over a 33 week period. The bottom panel shows the first three weeks of the same time series. There is a strong daily seasonal pattern with frequency 169 (there are 169 5-minute intervals per day), and a weak weekly seasonal pattern with frequency $169 \times 5 = 845$. (Call volumes on Mondays tend to be higher than the rest of the week.) If a longer series of data were available, we may also have observed an annual seasonal pattern.

```
p1 <- autoplot(calls) +
  ylab("Call volume") + xlab("Weeks") +
  scale_x_continuous(breaks=seq(1,33,by=2))
p2 <- autoplot(window(calls, end=4)) +
  ylab("Call volume") + xlab("Weeks") +
  scale_x_continuous(minor_breaks = seq(1,4,by=0.2))
gridExtra::grid.arrange(p1,p2)
```

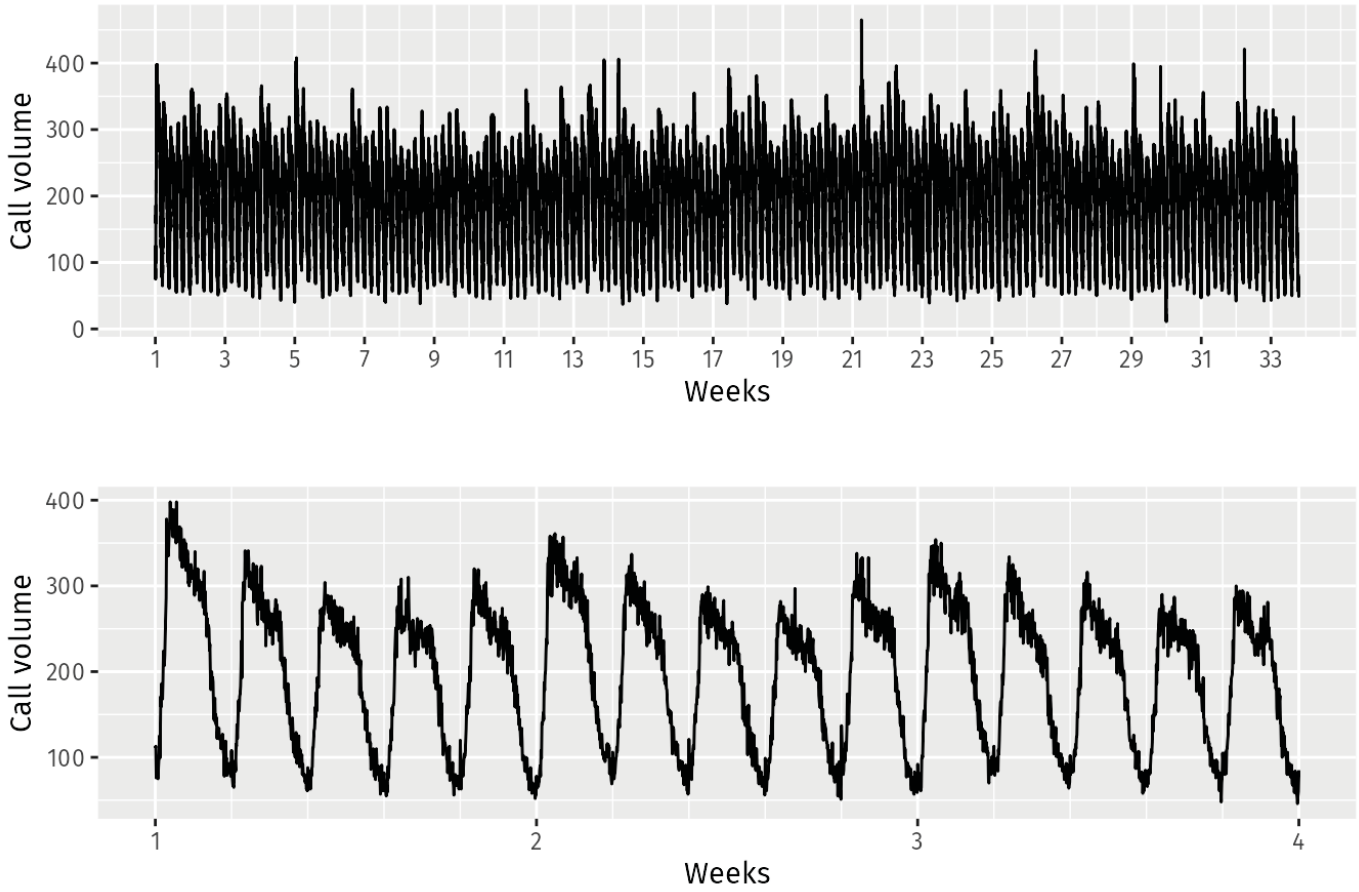


Figure 11.1: Five-minute call volume handled on weekdays between 7:00am and 9:05pm in a large North American commercial bank. Top panel shows data from 3 March 2003 for 164 days.

STL with multiple seasonal periods

The `mstl()` function is a variation on `stl()` designed to deal with multiple seasonality. It will return multiple seasonal components, as well as a trend and remainder component.

```
calls %>% mstl() %>%
  autoplot() + xlab("Week")
```

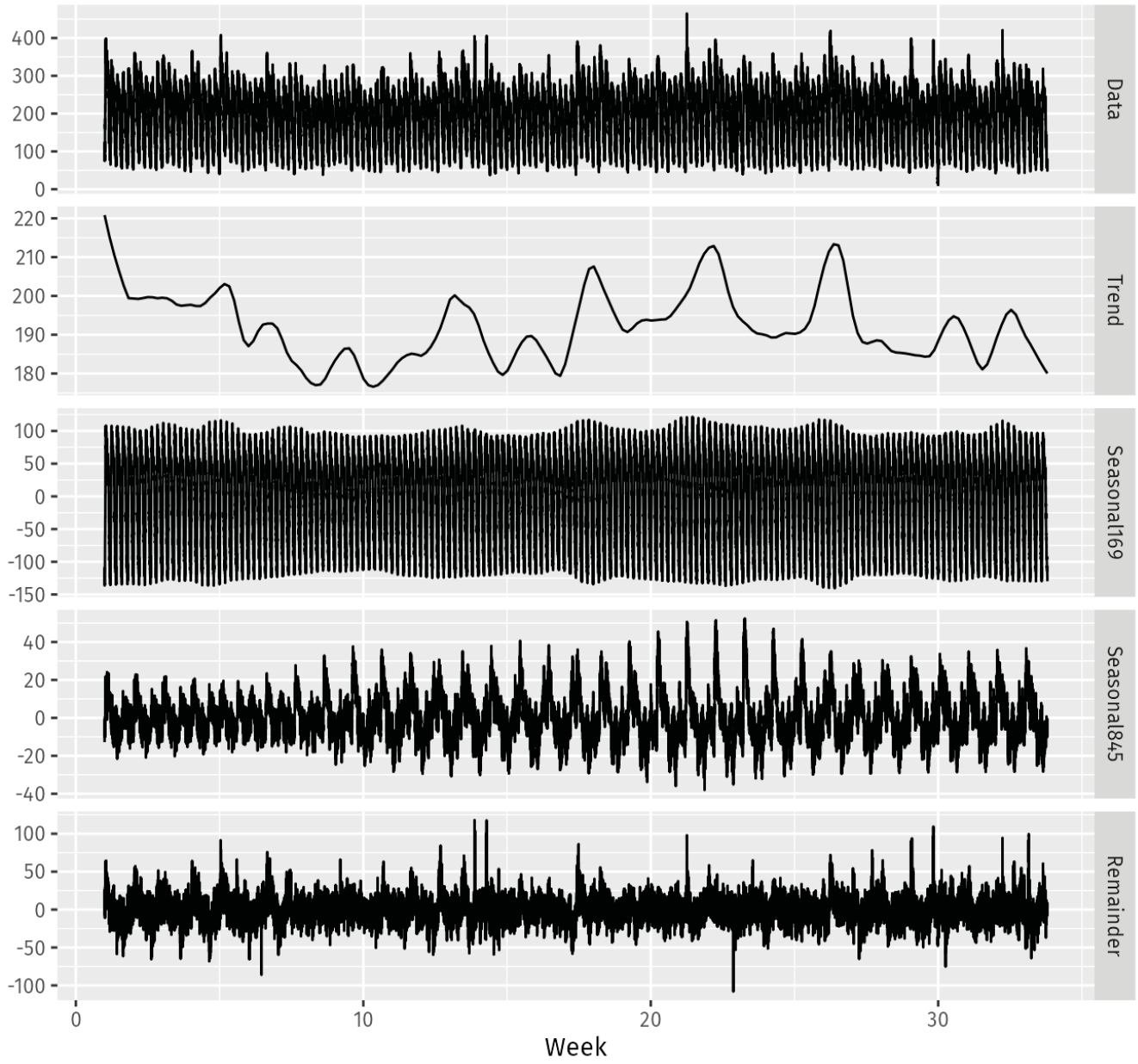


Figure 11.2: Multiple STL for the call volume data.

There are two seasonal patterns shown, one for the time of day (the third panel), and one for the time of week (the fourth panel). To properly interpret this graph, it is important to notice the vertical scales. In this case, the trend and the weekly seasonality have relatively narrow ranges compared to the other components, because there is little trend seen in the data, and the weekly seasonality is weak.

The decomposition can also be used in forecasting, with each of the seasonal components forecast using a seasonal naïve method, and the seasonally adjusted data forecasting using ETS (or some other user-specified method). The `stlf()` function will do this automatically.

```
calls %>%  stlf() %>%
  autoplot() + xlab("Week")
```

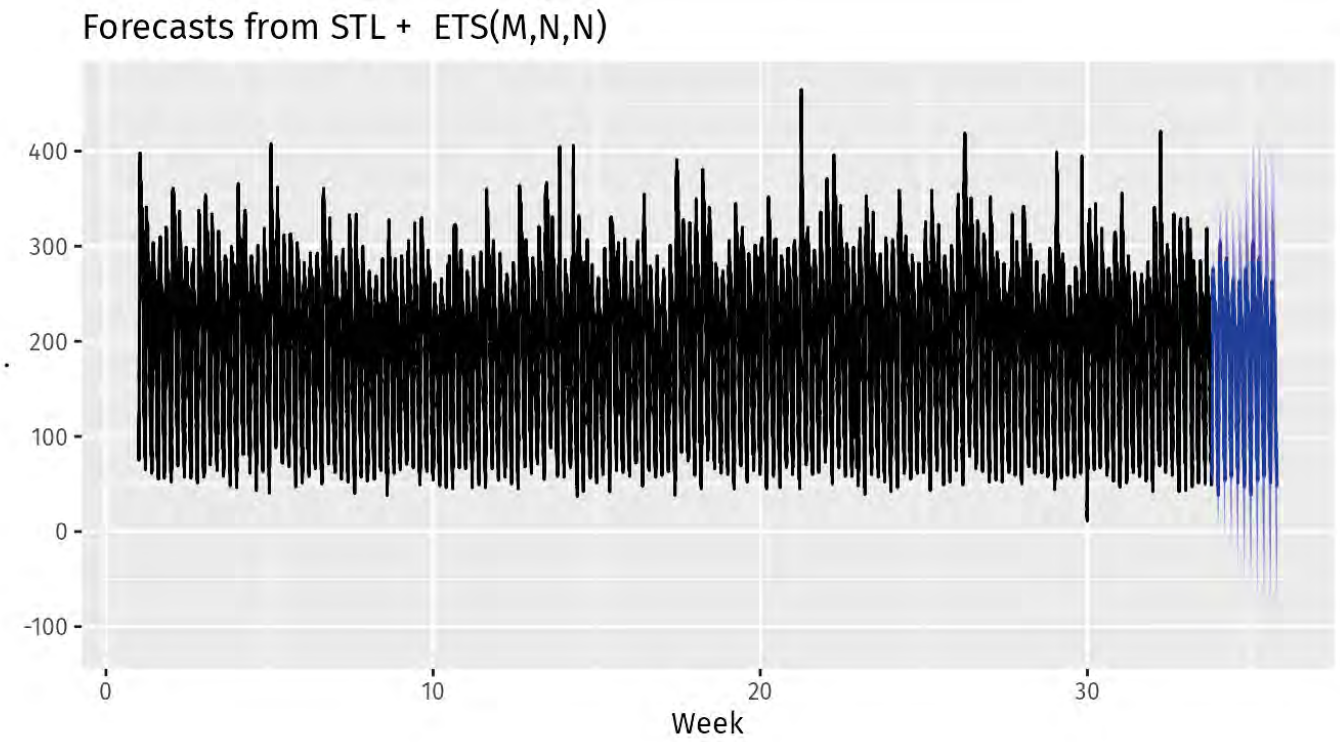


Figure 11.3: Multiple STL for the call volume data.

Dynamic harmonic regression with multiple seasonal periods

With multiple seasonalities, we can use Fourier terms as we did in earlier chapters (see Sections 5.4 and 9.5). Because there are multiple seasonalities, we need to add Fourier terms for each seasonal period. In this case, the seasonal periods are 169 and 845, so the Fourier terms are of the form

$$\sin\left(\frac{2\pi kt}{169}\right), \quad \cos\left(\frac{2\pi kt}{169}\right), \quad \sin\left(\frac{2\pi kt}{845}\right), \quad \text{and} \quad \cos\left(\frac{2\pi kt}{845}\right),$$

for $k = 1, 2, \ldots$. The `fourier()` function can generate these for you.

We will fit a dynamic harmonic regression model with an ARMA error structure. The total number of Fourier terms for each seasonal period have been chosen to minimise the AICc. We will use a log transformation (`lambda=0`) to ensure the forecasts and prediction intervals remain positive.

```
fit <- auto.arima(calls, seasonal=FALSE, lambda=0,
          xreg=fourier(calls, K=c(10,10)))
fit %>%
  forecast(xreg=fourier(calls, K=c(10,10), h=2*169)) %>%
  autoplot(include=5*169) +
    ylab("Call volume") + xlab("Weeks")
```

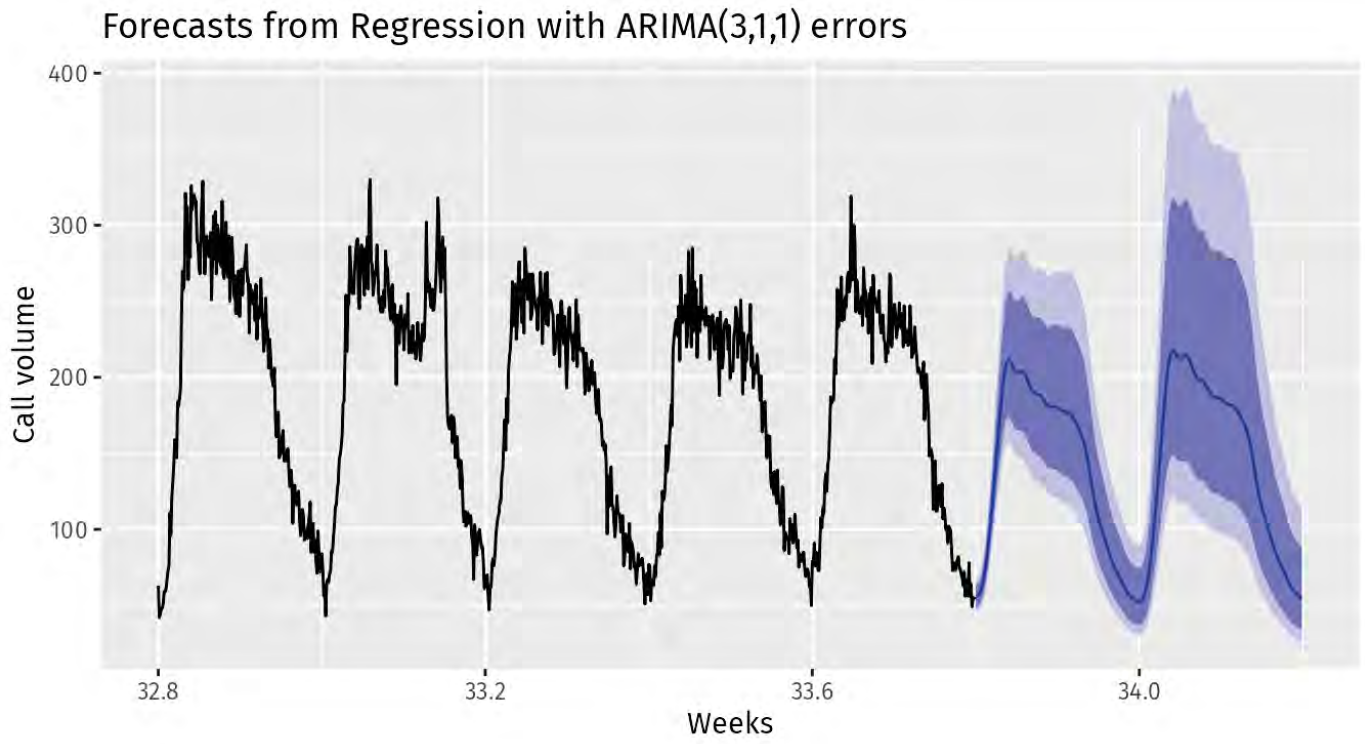


Figure 11.4: Forecasts from a dynamic harmonic regression applied to the call volume data.

This is a large model, containing 40 parameters: 4 ARMA coefficients, 20 Fourier coefficients for frequency 169, and 16 Fourier coefficients for frequency 845. We don't use all the Fourier terms for frequency 845 because there is some overlap with the terms of frequency 169 (since $845 = 5 \times 169$).

TBATS models

An alternative approach developed by De Livera, Hyndman, & Snyder (2011) uses a combination of Fourier terms with an exponential smoothing state space model and a Box-Cox transformation, in a completely automated manner. As with any automated modelling framework, there may be cases where it gives poor results, but it can be a useful approach in some circumstances.

A TBATS model differs from dynamic harmonic regression in that the seasonality is allowed to change slowly over time in a TBATS model, while harmonic regression terms force the seasonal patterns to repeat periodically without changing. One drawback of TBATS models, however, is that they can be slow to estimate, especially with long time series. Hence, we will consider a subset of the `calls` data to save time.

```
calls %>%
  subset(start=length(calls)-2000) %>%
  tbats() -> fit2
fc2 <- forecast(fit2, h=2*169)
autoplot(fc2, include=5*169) +
  ylab("Call volume") + xlab("Weeks")
```

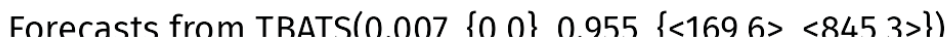


Figure 11.5: Forecasts from a TBATS model applied to the call volume data.

Here the prediction intervals appear to be much too wide – something that seems to happen quite often with TBATS models unfortunately.

Complex seasonality with covariates

TBATS models do not allow for covariates, although they can be included in dynamic harmonic regression models. One common application of such models is electricity demand modelling.

Figure 11.6 shows half-hourly electricity demand in Victoria, Australia, during 2014, along with temperatures for the same period for Melbourne (the largest city in Victoria).

```
autoplot(elecdemand[,c("Demand","Temperature")],
    facet=TRUE) +
  scale_x_continuous(minor_breaks=NULL,
    breaks=2014+
      cumsum(c(0,31,28,31,30,31,30,31,31,30,31,30))/365,
    labels=month.abb) +
  xlab("Time") + ylab("")
```

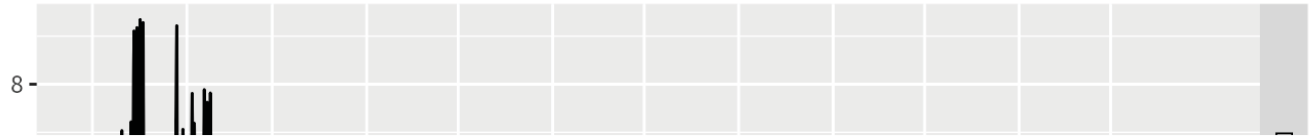


Figure 11.6: Half-hourly electricity demand and corresponding temperatures in 2014, Victoria, Australia.

Plotting electricity demand against temperature (Figure 11.7) shows that there is a nonlinear relationship between the two, with demand increasing for low temperatures (due to heating) and increasing for high temperatures (due to cooling).

```
elecdemand %>%
  as.data.frame() %>%
  ggplot(aes(x=Temperature, y=Demand)) + geom_point() +
    xlab("Temperature (degrees Celsius)") +
    ylab("Demand (GW)")
```

Figure 11.7: Half-hourly electricity demand for Victoria, plotted against temperatures for the same times in Melbourne, the largest city in Victoria.

We will fit a regression model with a piecewise linear function of temperature (containing a knot at 18 degrees), and harmonic regression terms to allow for the daily seasonal pattern.

```
cooling <- pmax(elecdemand[,"Temperature"], 18)
fit <- auto.arima(elecdemand[,"Demand"],
          xreg = cbind(fourier(elecdemand, c(10,10,0)),
                heating=elecdemand[,"Temperature"],
                cooling=cooling))
```

Forecasting with such models is difficult because we require future values of the predictor variables. Future values of the Fourier terms are easy to compute, but future temperatures are, of course, unknown. If we are only interested in forecasting up to a week ahead, we could use temperature forecasts obtain from a meteorological

model. Alternatively, we could use scenario forecasting (Section 4.5) and plug in possible temperature patterns. In the following example, we have used a repeat of the last two days of temperatures to generate future possible demand values.

```
temps <- subset(elecdemand[,"Temperature"],
          start=NROW(elecdemand)-2*48+1)
fc <- forecast(fit,
        xreg=cbind(fourier(temps, c(10,10,0)),
          heating=temps, cooling=pmax(temps,18)))
autoplot(fc, include=14*48)
```

Figure 11.8: Forecasts from a dynamic harmonic regression model applied to half-hourly electricity demand data.

Although the short-term forecasts look reasonable, this is a crude model for a complicated process. The residuals demonstrate that there is a lot of information that has not been captured with this model.

```
checkresiduals(fc)
```

Figure 11.9: Residual diagnostics for the dynamic harmonic regression model.

```
#>
#>  Ljung-Box test
#>
#> data:  Residuals from Regression with ARIMA(5,1,4) errors
#> Q* = 738424, df = 3495, p-value <2e-16
#>
#> Model df: 9.   Total lags used: 3504
```

More sophisticated versions of this model which provide much better forecasts are described in Hyndman & Fan (2010) and Fan & Hyndman (2012).

Bibliography

De Livera, A. M., Hyndman, R. J., & Snyder, R. D. (2011). Forecasting time series with complex seasonal patterns using exponential smoothing. *J American Statistical Association*, *106*(496), 1513–1527. [DOI]

Fan, S., & Hyndman, R. J. (2012). Short-term load forecasting based on a semi-parametric additive model. *IEEE Transactions on Power Systems*, *27*(1), 134–141. [DOI]

Hyndman, R. J., & Fan, S. (2010). Density forecasting for long-term peak electricity demand. *IEEE Transactions on Power Systems*, *25*(2), 1142–1153. [DOI]

11.2 Vector autoregressions

One limitation of the models that we have considered so far is that they impose a unidirectional relationship — the forecast variable is influenced by the predictor variables, but not vice versa. However, there are many cases where the reverse should also be allowed for — where all variables affect each other. In Section 9.2, the changes in personal consumption expenditure (C_t) were forecast based on the changes in personal disposable income (I_t). However, in this case a bi-directional relationship may be more suitable: an increase in I_t will lead to an increase in C_t and vice versa.

An example of such a situation occurred in Australia during the Global Financial Crisis of 2008–2009. The Australian government issued stimulus packages that included cash payments in December 2008, just in time for Christmas spending. As a result, retailers reported strong sales and the economy was stimulated. Consequently, incomes increased.

Such feedback relationships are allowed for in the vector autoregressive (VAR) framework. In this framework, all variables are treated symmetrically. They are all modelled as if they all influence each other equally. In more formal terminology, all variables are now treated as "endogenous". To signify this, we now change the notation and write all variables as ys: $y_{1,t}$ denotes the tth observation of variable y_1, $y_{2,t}$ denotes the tth observation of variable y_2, and so on.

A VAR model is a generalisation of the univariate autoregressive model for forecasting a vector of time series.[23] It comprises one equation per variable in the system. The right hand side of each equation includes a constant and lags of all of the variables in the system. To keep it simple, we will consider a two variable VAR with one lag. We write a 2-dimensional VAR(1) as

$$y_{1,t} = c_1 + \phi_{11,1}y_{1,t-1} + \phi_{12,1}y_{2,t-1} + e_{1,t} \tag{11.1}$$
$$y_{2,t} = c_2 + \phi_{21,1}y_{1,t-1} + \phi_{22,1}y_{2,t-1} + e_{2,t}, \tag{11.2}$$

where $e_{1,t}$ and $e_{2,t}$ are white noise processes that may be contemporaneously correlated. The coefficient $\phi_{ii,\ell}$ captures the influence of the ℓth lag of variable y_i on itself, while the coefficient $\phi_{ij,\ell}$ captures the influence of the ℓth lag of variable y_j on y_i.

If the series are stationary, we forecast them by fitting a VAR to the data directly (known as a "VAR in levels"). If the series are non-stationary, we take differences of the data in order to make them stationary, then fit a VAR model (known as a "VAR in differences"). In both cases, the models are estimated equation by equation using the principle of least squares. For each equation, the parameters are estimated by minimising the sum of squared $e_{i,t}$ values.

The other possibility, which is beyond the scope of this book and therefore we do not explore here, is that the series may be non-stationary but cointegrated, which means that there exists a linear combination of them that is stationary. In this case, a VAR specification that includes an error correction mechanism (usually referred to as a vector error correction model) should be included, and alternative estimation methods to least squares estimation should be used.[24]

Forecasts are generated from a VAR in a recursive manner. The VAR generates forecasts for *each* variable included in the system. To illustrate the process, assume that we have fitted the 2-dimensional VAR(1) described in Equations (11.1)–(11.2), for all observations up to time T. Then the one-step-ahead forecasts are generated by

$$\begin{aligned}\hat{y}_{1,T+1|T} &= \hat{c}_1 + \hat{\phi}_{11,1}y_{1,T} + \hat{\phi}_{12,1}y_{2,T}\\ \hat{y}_{2,T+1|T} &= \hat{c}_2 + \hat{\phi}_{21,1}y_{1,T} + \hat{\phi}_{22,1}y_{2,T}.\end{aligned}$$

This is the same form as (11.1)–(11.2), except that the errors have been set to zero and parameters have been replaced with their estimates. For $h = 2$, the forecasts are given by

$$\begin{aligned}\hat{y}_{1,T+2|T} &= \hat{c}_1 + \hat{\phi}_{11,1}\hat{y}_{1,T+1} + \hat{\phi}_{12,1}\hat{y}_{2,T+1}\\ \hat{y}_{2,T+2|T} &= \hat{c}_2 + \hat{\phi}_{21,1}\hat{y}_{1,T+1} + \hat{\phi}_{22,1}\hat{y}_{2,T+1}.\end{aligned}$$

Again, this is the same form as (11.1)–(11.2), except that the errors have been set to zero, the parameters have been replaced with their estimates, and the unknown values of y_1 and y_2 have been replaced with their forecasts. The process can be iterated in this manner for all future time periods.

There are two decisions one has to make when using a VAR to forecast, namely how many variables (denoted by K) and how many lags (denoted by p) should be included in the system. The number of coefficients to be estimated in a VAR is equal to $K + pK^2$ (or $1 + pK$ per equation). For example, for a VAR with $K = 5$ variables

and $p = 3$ lags, there are 16 coefficients per equation, giving a total of 80 coefficients to be estimated. The more coefficients that need to be estimated, the larger the estimation error entering the forecast.

In practice, it is usual to keep K small and include only variables that are correlated with each other, and therefore useful in forecasting each other. Information criteria are commonly used to select the number of lags to be included.

VAR models are implemented in the **vars** package in R. It contains a function `VARselect()` for selecting the number of lags p using four different information criteria: AIC, HQ, SC and FPE. We have met the AIC before, and SC is simply another name for the BIC (SC stands for Schwarz Criterion, after Gideon Schwarz who proposed it). HQ is the Hannan-Quinn criterion, and FPE is the "Final Prediction Error" criterion.[25] Care should be taken when using the AIC as it tends to choose large numbers of lags. Instead, for VAR models, we prefer to use the BIC.

A criticism that VARs face is that they are atheoretical; that is, they are not built on some economic theory that imposes a theoretical structure on the equations. Every variable is assumed to influence every other variable in the system, which makes a direct interpretation of the estimated coefficients difficult. Despite this, VARs are useful in several contexts:

1. forecasting a collection of related variables where no explicit interpretation is required;
2. testing whether one variable is useful in forecasting another (the basis of Granger causality tests);
3. impulse response analysis, where the response of one variable to a sudden but temporary change in another variable is analysed;
4. forecast error variance decomposition, where the proportion of the forecast variance of each variable is attributed to the effects of the other variables.

Example: A VAR model for forecasting US consumption

```
library(vars)

VARselect(uschange[,1:2], lag.max=8,
  type="const")[["selection"]]
#> AIC(n)  HQ(n)  SC(n) FPE(n)
#>      5      1      1      5
```

The R output shows the lag length selected by each of the information criteria available in the **vars** package. There is a large discrepancy between the VAR(5) selected by the AIC and the VAR(1) selected by the BIC. This is not unusual. As a result we first fit a VAR(1), as selected by the BIC.

```
var1 <- VAR(uschange[,1:2], p=1, type="const")
serial.test(var1, lags.pt=10, type="PT.asymptotic")
var2 <- VAR(uschange[,1:2], p=2, type="const")
serial.test(var2, lags.pt=10, type="PT.asymptotic")
```

In similar fashion to the univariate ARIMA methodology, we test that the residuals are uncorrelated using a Portmanteau test[26]. Both a VAR(1) and a VAR(2) have some residual serial correlation, and therefore we fit a VAR(3).

```
var3 <- VAR(uschange[,1:2], p=3, type="const")
serial.test(var3, lags.pt=10, type="PT.asymptotic")
#>
#>  Portmanteau Test (asymptotic)
#>
#> data:  Residuals of VAR object var3
#> Chi-squared = 34, df = 28, p-value = 0.2
```

The residuals for this model pass the test for serial correlation. The forecasts generated by the VAR(3) are plotted in Figure 11.10.

```
forecast(var3) %>%
  autoplot() + xlab("Year")
```

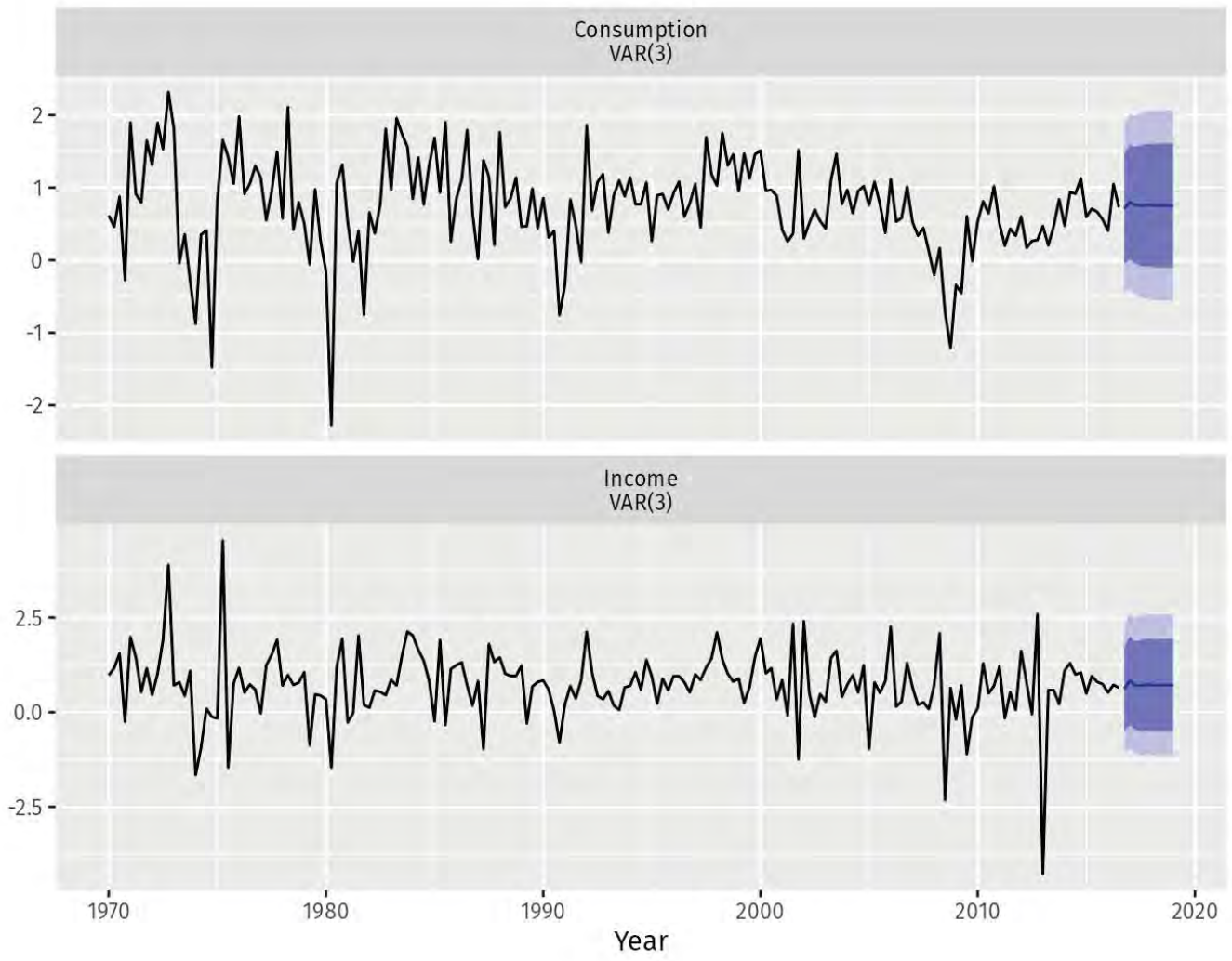


Figure 11.10: Forecasts for US consumption and income generated from a VAR(3).

Bibliography

Athanasopoulos, G., Poskitt, D. S., & Vahid, F. (2012). Two canonical VARMA forms: Scalar component models vis-à-vis the echelon form. *Econometric Reviews*, *31*(1), 60–83. [DOI]

Hamilton, J. D. (1994). *Time series analysis*. Princeton University Press, Princeton. [Amazon]

Lütkepohl, H. (2005). *New introduction to multiple time series analysis*. Berlin: Springer-Verlag. [Amazon]

Lütkepohl, H. (2007). General-to-specific or specific-to-general modelling? An opinion on current econometric terminology. *Journal of Econometrics*, *136*(1), 234–319. [DOI]

23. A more flexible generalisation would be a Vector ARMA process. However, the relative simplicity of VARs has led to their dominance in forecasting. Interested readers may refer to Athanasopoulos, Poskitt, & Vahid (2012).↩

24. Interested readers should refer to Hamilton (1994) and Lütkepohl (2007).↩

25. For a detailed comparison of these criteria, see Chapter 4.3 of Lütkepohl (2005).↩

26. The tests for serial correlation in the "vars" package are multivariate generalisations of the tests presented in Section 3.3.↩

11.3 Neural network models

Artificial neural networks are forecasting methods that are based on simple mathematical models of the brain. They allow complex nonlinear relationships between the response variable and its predictors.

Neural network architecture

A neural network can be thought of as a network of "neurons" which are organised in layers. The predictors (or inputs) form the bottom layer, and the forecasts (or outputs) form the top layer. There may also be intermediate layers containing "hidden neurons".

The simplest networks contain no hidden layers and are equivalent to linear regressions. Figure 11.11 shows the neural network version of a linear regression with four predictors. The coefficients attached to these predictors are called "weights". The forecasts are obtained by a linear combination of the inputs. The weights are selected in the neural network framework using a "learning algorithm" that minimises a "cost function" such as the MSE. Of course, in this simple example, we can use linear regression which is a much more efficient method of training the model.

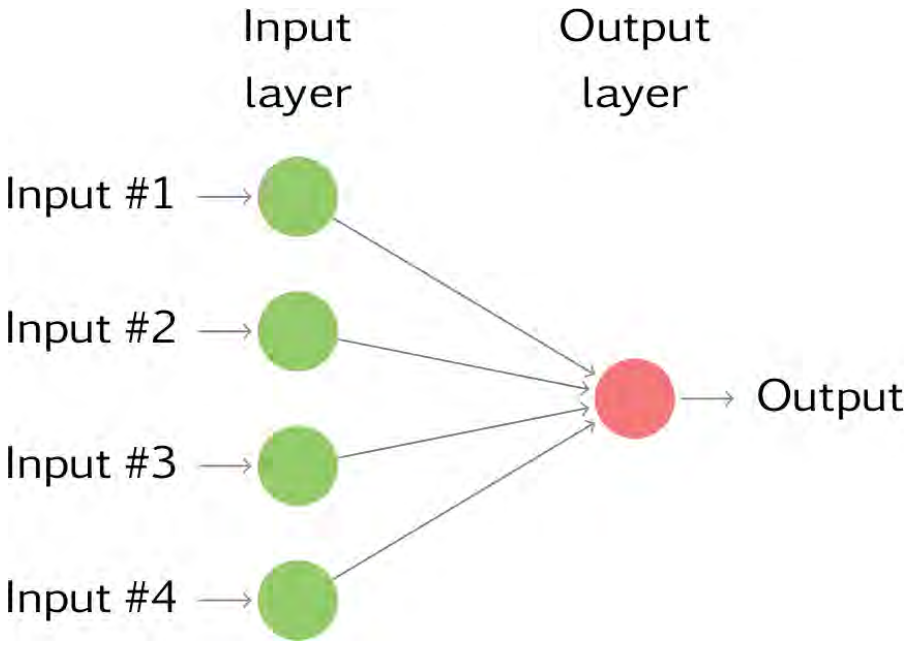


Figure 11.11: A simple neural network equivalent to a linear regression.

Once we add an intermediate layer with hidden neurons, the neural network becomes non-linear. A simple example is shown in Figure 11.12.

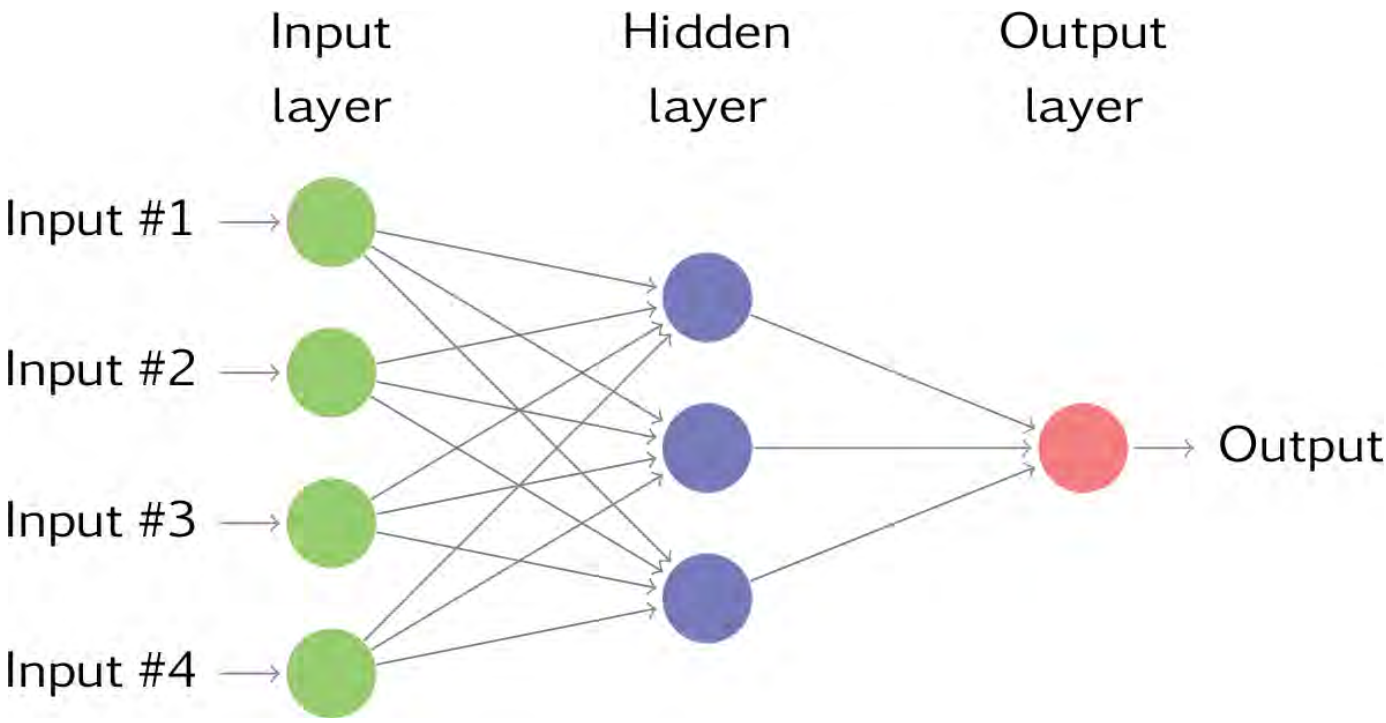


Figure 11.12: A neural network with four inputs and one hidden layer with three hidden neurons.

This is known as a *multilayer feed-forward network*, where each layer of nodes receives inputs from the previous layers. The outputs of the nodes in one layer are inputs to the next layer. The inputs to each node are combined using a weighted linear combination. The result is then modified by a nonlinear function before being output. For example, the inputs into hidden neuron j in Figure 11.12 are combined linearly to give

$$z_j = b_j + \sum_{i=1}^{4} w_{i,j} x_i.$$

In the hidden layer, this is then modified using a nonlinear function such as a sigmoid,

$$s(z) = \frac{1}{1+e^{-z}},$$

to give the input for the next layer. This tends to reduce the effect of extreme input values, thus making the network somewhat robust to outliers.

The parameters b_1, b_2, b_3 and $w_{1,1}, \dots, w_{4,3}$ are "learned" from the data. The values of the weights are often restricted to prevent them from becoming too large. The parameter that restricts the weights is known as the "decay parameter", and is often

set to be equal to 0.1.

The weights take random values to begin with, and these are then updated using the observed data. Consequently, there is an element of randomness in the predictions produced by a neural network. Therefore, the network is usually trained several times using different random starting points, and the results are averaged.

The number of hidden layers, and the number of nodes in each hidden layer, must be specified in advance. Usually, these would be selected using cross-validation.

Neural network autoregression

With time series data, lagged values of the time series can be used as inputs to a neural network, just as we used lagged values in a linear autoregression model (Chapter 8). We call this a neural network autoregression or NNAR model.

In this book, we only consider feed-forward networks with one hidden layer, and we use the notation NNAR(p, k) to indicate there are p lagged inputs and k nodes in the hidden layer. For example, a NNAR(9,5) model is a neural network with the last nine observations $(y_{t-1}, y_{t-2}, \dots, y_{t-9})$ used as inputs for forecasting the output y_t, and with five neurons in the hidden layer. A NNAR(p, 0) model is equivalent to an ARIMA(p, 0, 0) model, but without the restrictions on the parameters to ensure stationarity.

With seasonal data, it is useful to also add the last observed values from the same season as inputs. For example, an NNAR(3,1,2)$_{12}$ model has inputs y_{t-1}, y_{t-2}, y_{t-3} and y_{t-12}, and two neurons in the hidden layer. More generally, an NNAR(p, P, k)$_m$ model has inputs $(y_{t-1}, y_{t-2}, \dots, y_{t-p}, y_{t-m}, y_{t-2m}, \dots, y_{t-Pm})$ and k neurons in the hidden layer. A NNAR(p, P, 0)$_m$ model is equivalent to an ARIMA(p, 0, 0)(P,0,0)$_m$ model but without the restrictions on the parameters that ensure stationarity.

The `nnetar()` function fits an NNAR(p, P, k)$_m$ model. If the values of p and P are not specified, they are selected automatically. For non-seasonal time series, the default is the optimal number of lags (according to the AIC) for a linear AR(p) model. For seasonal time series, the default values are $P = 1$ and p is chosen from the optimal linear model fitted to the seasonally adjusted data. If k is not specified, it is set to $k = (p + P + 1)/2$ (rounded to the nearest integer).

When it comes to forecasting, the network is applied iteratively. For forecasting one step ahead, we simply use the available historical inputs. For forecasting two steps ahead, we use the one-step forecast as an input, along with the historical data. This process proceeds until we have computed all the required forecasts.

Example: sunspots

The surface of the sun contains magnetic regions that appear as dark spots. These affect the propagation of radio waves, and so telecommunication companies like to predict sunspot activity in order to plan for any future difficulties. Sunspots follow a cycle of length between 9 and 14 years. In Figure 11.13, forecasts from an NNAR(10,6) are shown for the next 30 years. We have set a Box-Cox transformation with `lambda=0` to ensure the forecasts stay positive.

```
fit <- nnetar(sunspotarea, lambda=0)
autoplot(forecast(fit,h=30))
```

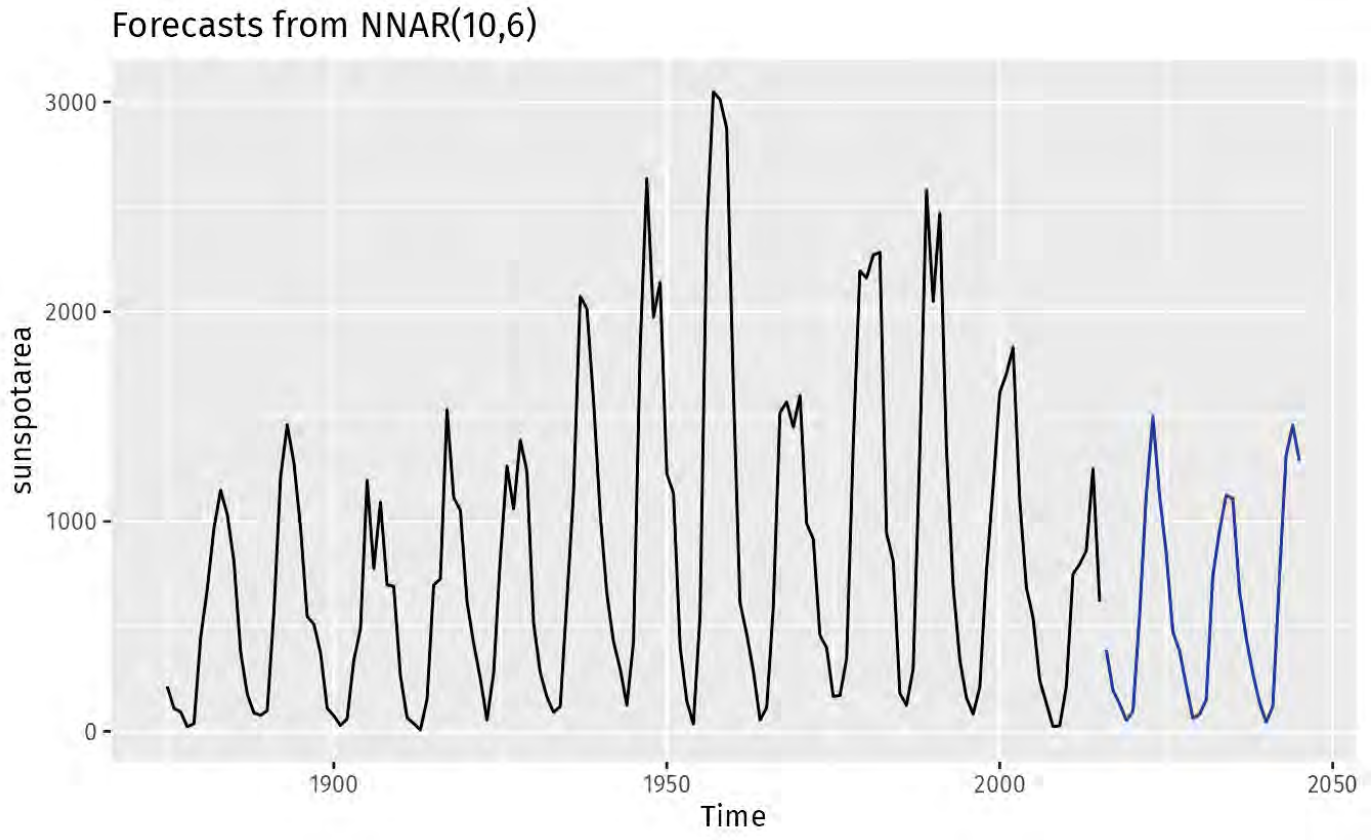


Figure 11.13: Forecasts from a neural network with ten lagged inputs and one hidden layer containing six neurons.

Here, the last 10 observations are used as predictors, and there are 6 neurons in the hidden layer. The cyclicity in the data has been modelled well. We can also see the asymmetry of the cycles has been captured by the model, where the increasing part of the cycle is steeper than the decreasing part of the cycle. This is one difference between a NNAR model and a linear AR model — while linear AR models can model cyclicity, the modelled cycles are always symmetric.

Prediction intervals

Unlike most of the methods considered in this book, neural networks are not based on a well-defined stochastic model, and so it is not straightforward to derive prediction intervals for the resultant forecasts. However, we can still compute prediction intervals using simulation where future sample paths are generated using bootstrapped residuals (as described in Section 3.5).

The neural network fitted to the sunspot data can be written as

$$y_t = f(\boldsymbol{y}_{t-1}) + \varepsilon_t$$

where $\boldsymbol{y}_{t-1} = (y_{t-1}, y_{t-2}, \dots, y_{t-10})'$ is a vector containing lagged values of the series, and f is a neural network with 6 hidden nodes in a single layer. The error series $\{\varepsilon_t\}$ is assumed to be homoscedastic (and possibly also normally distributed).

We can simulate future sample paths of this model iteratively, by randomly generating a value for ε_t, either from a normal distribution, or by resampling from the historical values. So if ε^*_{T+1} is a random draw from the distribution of errors at time $T+1$, then

$$y^*_{T+1} = f(\boldsymbol{y}_T) + \varepsilon^*_{T+1}$$

is one possible draw from the forecast distribution for y_{T+1}. Setting $\boldsymbol{y}^*_{T+1} = (y^*_{T+1}, y_T, \dots, y_{T-8})'$, we can then repeat the process to get

$$y^*_{T+2} = f(\boldsymbol{y}^*_{T+1}) + \varepsilon^*_{T+2}.$$

In this way, we can iteratively simulate a future sample path. By repeatedly simulating sample paths, we build up knowledge of the distribution for all future values based on the fitted neural network.

Here is a simulation of 9 possible future sample paths for the sunspot data. Each sample path covers the next 30 years after the observed data.

```
sim <- ts(matrix(0, nrow=30L, ncol=9L),
  start=end(sunspotarea)[1L]+1L)
for(i in seq(9))
  sim[,i] <- simulate(fit, nsim=30L)
autoplot(sunspotarea) + autolayer(sim)
```

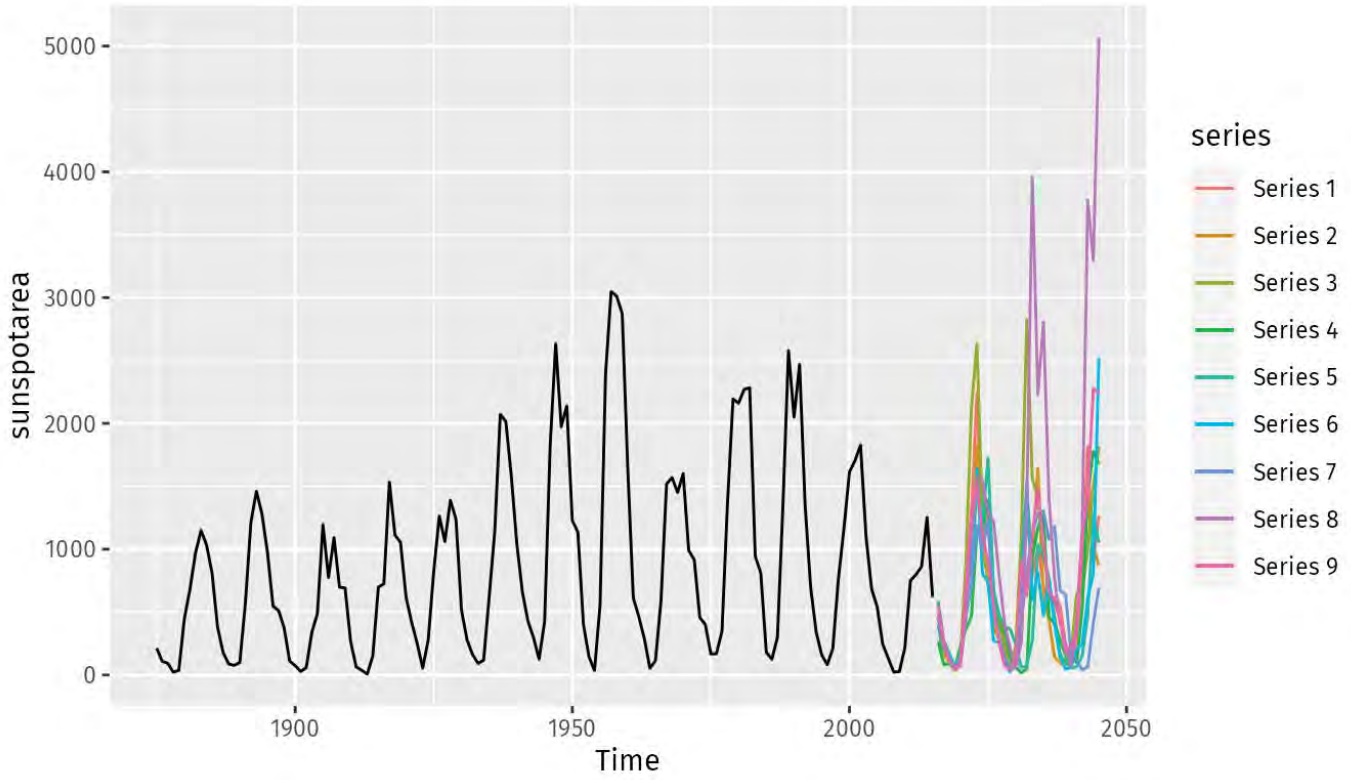


Figure 11.14: Future sample paths for the annual sunspot data.

If we do this a few hundred or thousand times, we can get a good picture of the forecast distributions. This is how the `forecast()` function produces prediction intervals for NNAR models:

```
fcast <- forecast(fit, PI=TRUE, h=30)
autoplot(fcast)
```

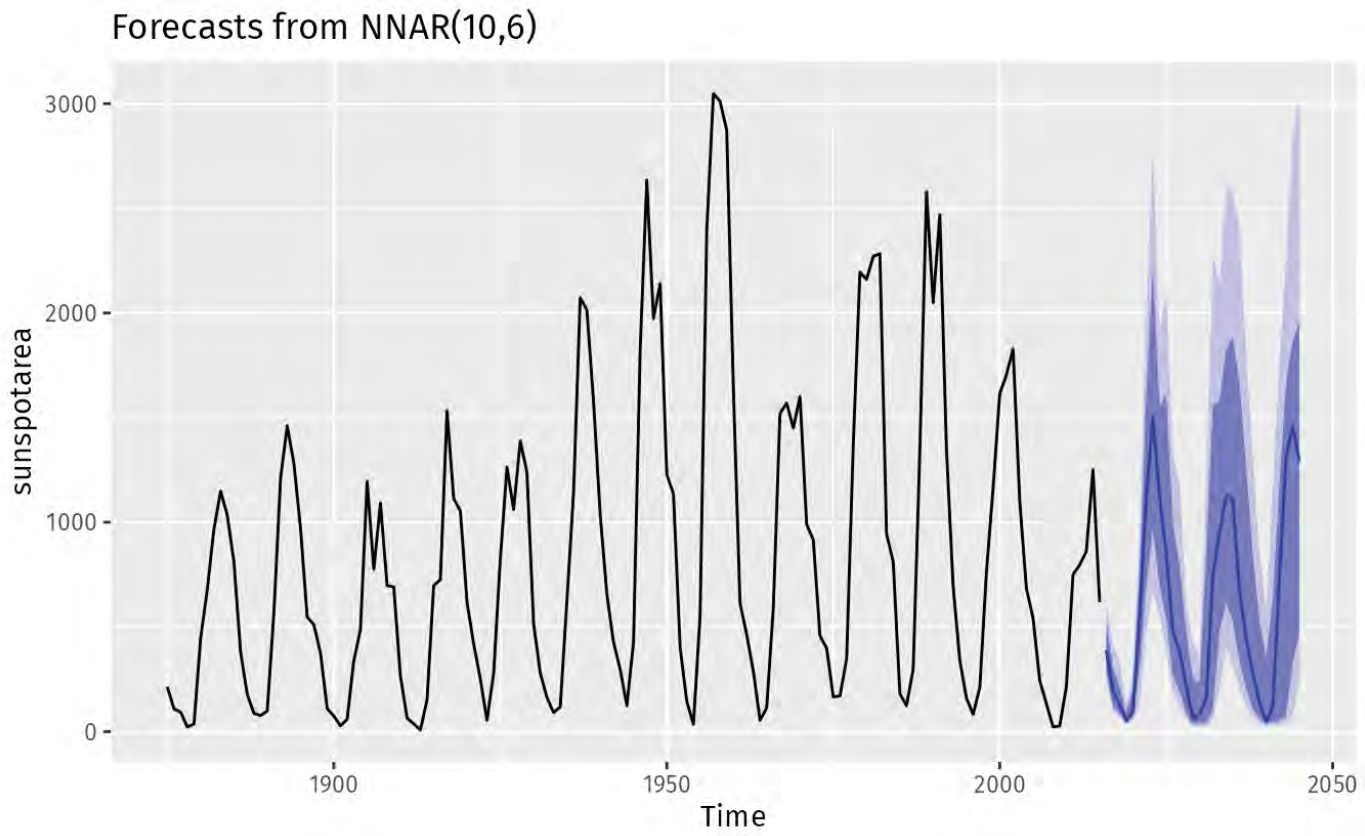


Figure 11.15: Forecasts with prediction intervals for the annual sunspot data. Prediction intervals are computed using simulated future sample paths.

Because it is a little slow, `PI=FALSE` is the default, so prediction intervals are not computed unless requested. The `npaths` argument in `forecast()` controls how many simulations are done (default 1000). By default, the errors are drawn from a normal distribution. The `bootstrap` argument allows the errors to be "bootstrapped" (i.e., randomly drawn from the historical errors).

11.4 Bootstrapping and bagging

Bootstrapping time series

In the preceding section, and in Section 3.5, we bootstrap the residuals of a time series in order to simulate future values of a series using a model.

More generally, we can generate new time series that are similar to our observed series, using another type of bootstrap.

First, the time series is Box-Cox-transformed, and then decomposed into trend, seasonal and remainder components using STL. Then we obtain shuffled versions of the remainder component to get bootstrapped remainder series. Because there may be autocorrelation present in an STL remainder series, we cannot simply use the re-draw procedure that was described in Section 3.5. Instead, we use a "blocked bootstrap", where contiguous sections of the time series are selected at random and joined together. These bootstrapped remainder series are added to the trend and seasonal components, and the Box-Cox transformation is reversed to give variations on the original time series.

Some examples are shown in Figure 11.16 for the monthly expenditure on retail debit cards in Iceland, from January 2000 to August 2013.

```
bootseries <- bld.mbb.bootstrap(debitcards, 10) %>%
  as.data.frame() %>% ts(start=2000, frequency=12)
autoplot(debitcards) +
  autolayer(bootseries, colour=TRUE) +
  autolayer(debitcards, colour=FALSE) +
  ylab("Bootstrapped series") + guides(colour="none")
```

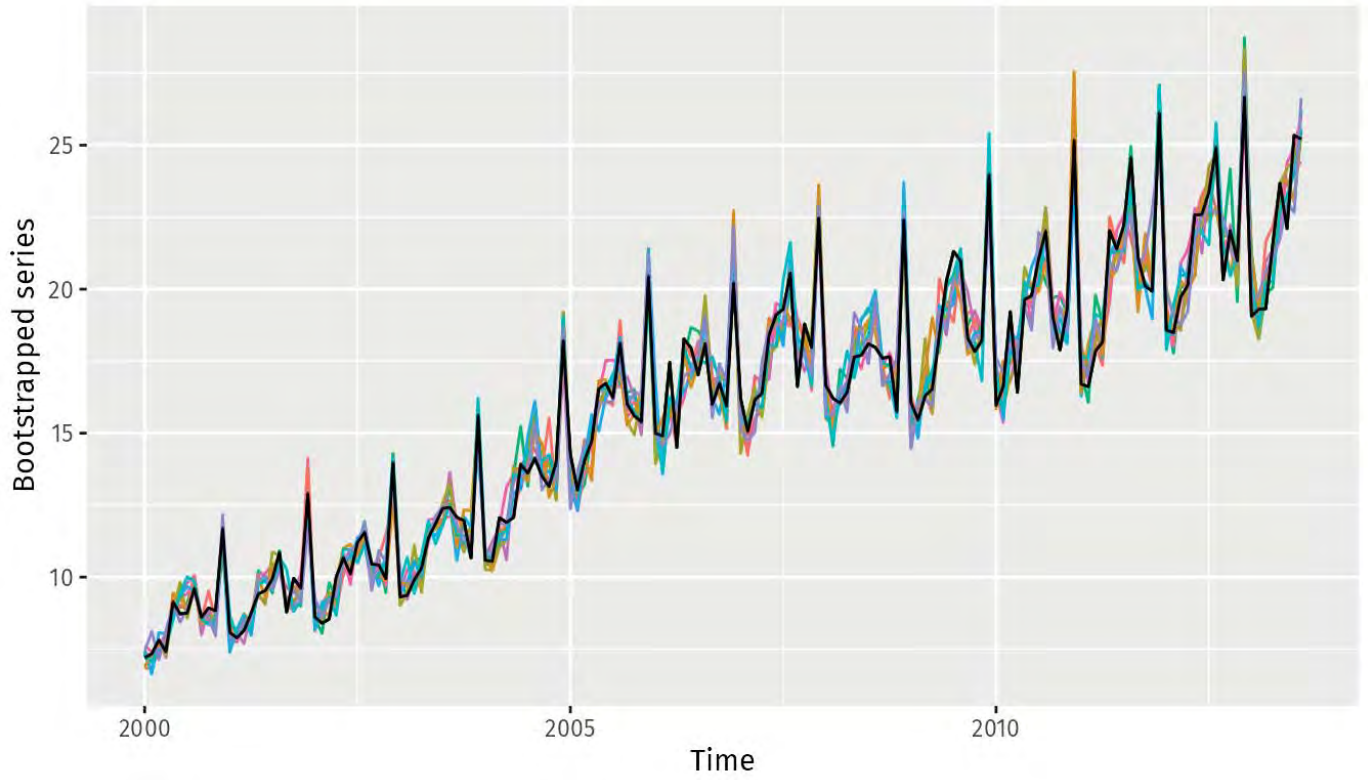


Figure 11.16: Ten bootstrapped versions of monthly expenditure on retail debit cards in Iceland.

This type of bootstrapping can be useful in two ways. First it helps us to get a better measure of forecast uncertainty, and second it provides a way of improving our point forecasts using "bagging".

Prediction intervals from bootstrapped series

Almost all prediction intervals from time series models are too narrow. This is a well-known phenomenon and arises because they do not account for all sources of uncertainty. Hyndman, Koehler, Snyder, & Grose (2002) measured the size of the problem by computing the actual coverage percentage of the prediction intervals on test data, and found that for ETS models, nominal 95% intervals may only provide coverage between 71% and 87%. The difference is due to missing sources of uncertainty.

There are at least four sources of uncertainty in forecasting using time series models:

1. The random error term;
2. The parameter estimates;
3. The choice of model for the historical data;
4. The continuation of the historical data generating process into the future.

When we produce prediction intervals for time series models, we generally only take into account the first of these sources of uncertainty. Even if we ignore the model uncertainty and the uncertainty due to changing data generating processes (sources 3 and 4), and we just try to allow for parameter uncertainty as well as the random error term (sources 1 and 2), there are no algebraic solutions apart from some simple special cases.

We can use bootstrapped time series to go some way towards overcoming this problem. We demonstrate the idea using the `debitcards` data. First, we simulate many time series that are similar to the original data, using the block-bootstrap described above.

```
nsim <- 1000L
sim <- bld.mbb.bootstrap(debitcards, nsim)
```

For each of these series, we fit an ETS model and simulate one sample path from that model. A different ETS model may be selected in each case, although it will most likely select the same model because the series are similar. However, the estimated parameters will be different. Therefore the simulated sample paths will allow for model uncertainty and parameter uncertainty, as well as the uncertainty associated with the random error term. This is a time-consuming process as there are a large number of time series to model.

```
h <- 36L
future <- matrix(0, nrow=nsim, ncol=h)
for(i in seq(nsim))
  future[i,] <- simulate(ets(sim[[i]]), nsim=h)
```

Finally, we take the means and quantiles of these simulated sample paths to form point forecasts and prediction intervals.

```
start <- tsp(debitcards)[2]+1/12
simfc <- structure(list(
    mean = ts(colMeans(future), start=start, frequency=12),
    lower = ts(apply(future, 2, quantile, prob=0.025),
               start=start, frequency=12),
    upper = ts(apply(future, 2, quantile, prob=0.975),
               start=start, frequency=12),
    level=95),
  class="forecast")
```

These prediction intervals will be larger than those obtained from an ETS model applied directly to the original data.

```
etsfc <- forecast(ets(debitcards), h=h, level=95)
autoplot(debitcards) +
  ggtitle("Monthly retail debit card usage in Iceland") +
  xlab("Year") + ylab("million ISK") +
  autolayer(simfc, series="Simulated") +
  autolayer(etsfc, series="ETS")
```

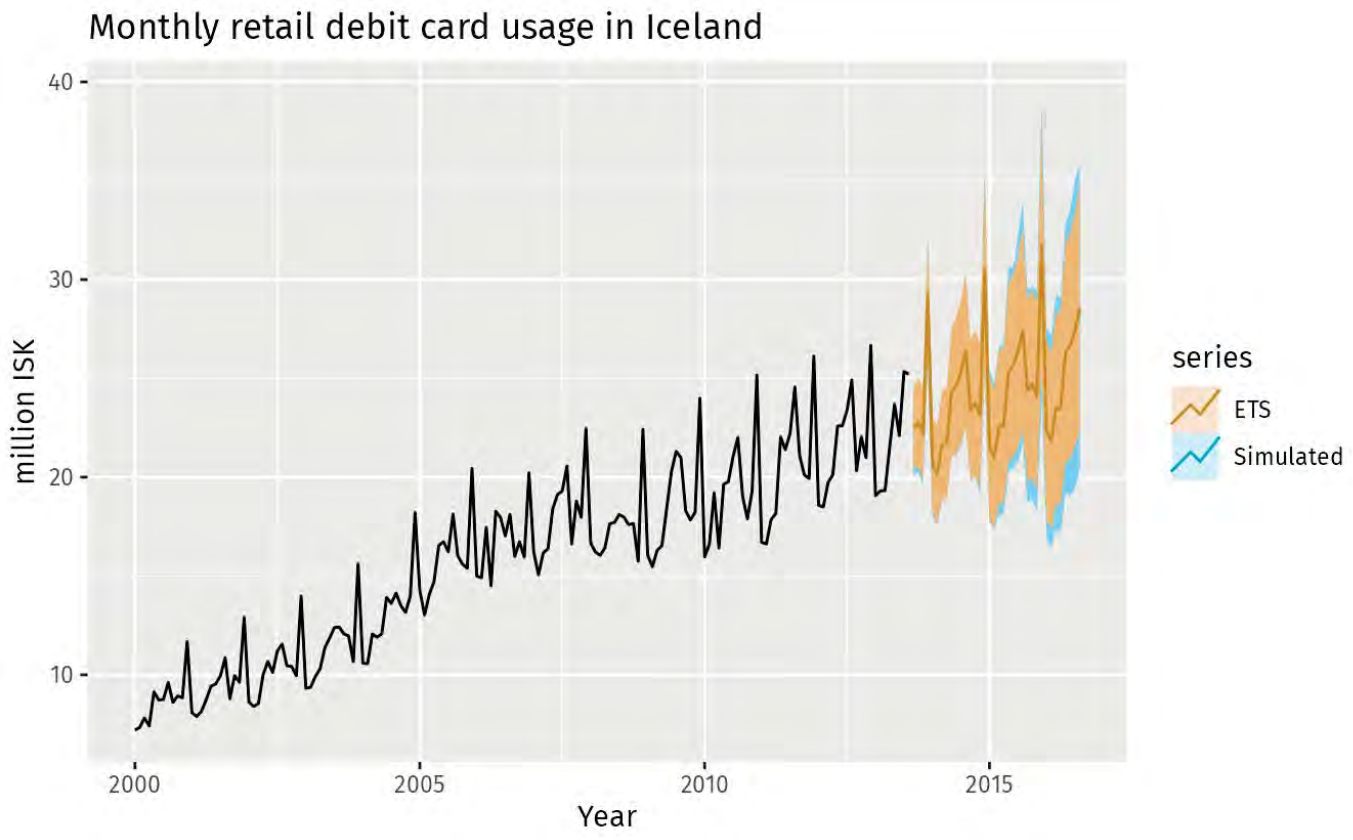


Figure 11.17: Forecasts of Iceland debit card usage using an ETS model with regular prediction intervals along with prediction intervals computed using simulations that allow for model and parameter uncertainty.

Bagged ETS forecasts

Another use for these bootstrapped time series is to improve forecast accuracy. If we produce forecasts from each of the additional time series, and average the resulting forecasts, we get better forecasts than if we simply forecast the original time series directly. This is called "bagging" which stands for "**b**ootstrap **agg**regating".

We could simply average the simulated future sample paths computed earlier. However, if our interest is only in improving point forecast accuracy, and not in also obtaining improved prediction intervals, then it is quicker to average the point forecasts from each series. The speed improvement comes about because we do not need to produce so many simulated series.

We will use `ets()` to forecast each of these series. Figure 11.18 shows ten forecasts obtained in this way.

```
sim <- bld.mbb.bootstrap(debitcards, 10) %>%
  as.data.frame() %>%
  ts(frequency=12, start=2000)
fc <- purrr::map(as.list(sim),
            function(x){forecast(ets(x))[["mean"]]}) %>%
      as.data.frame() %>%
      ts(frequency=12, start=start)
autoplot(debitcards) +
  autolayer(sim, colour=TRUE) +
  autolayer(fc, colour=TRUE) +
  autolayer(debitcards, colour=FALSE) +
  ylab("Bootstrapped series") +
  guides(colour="none")
```

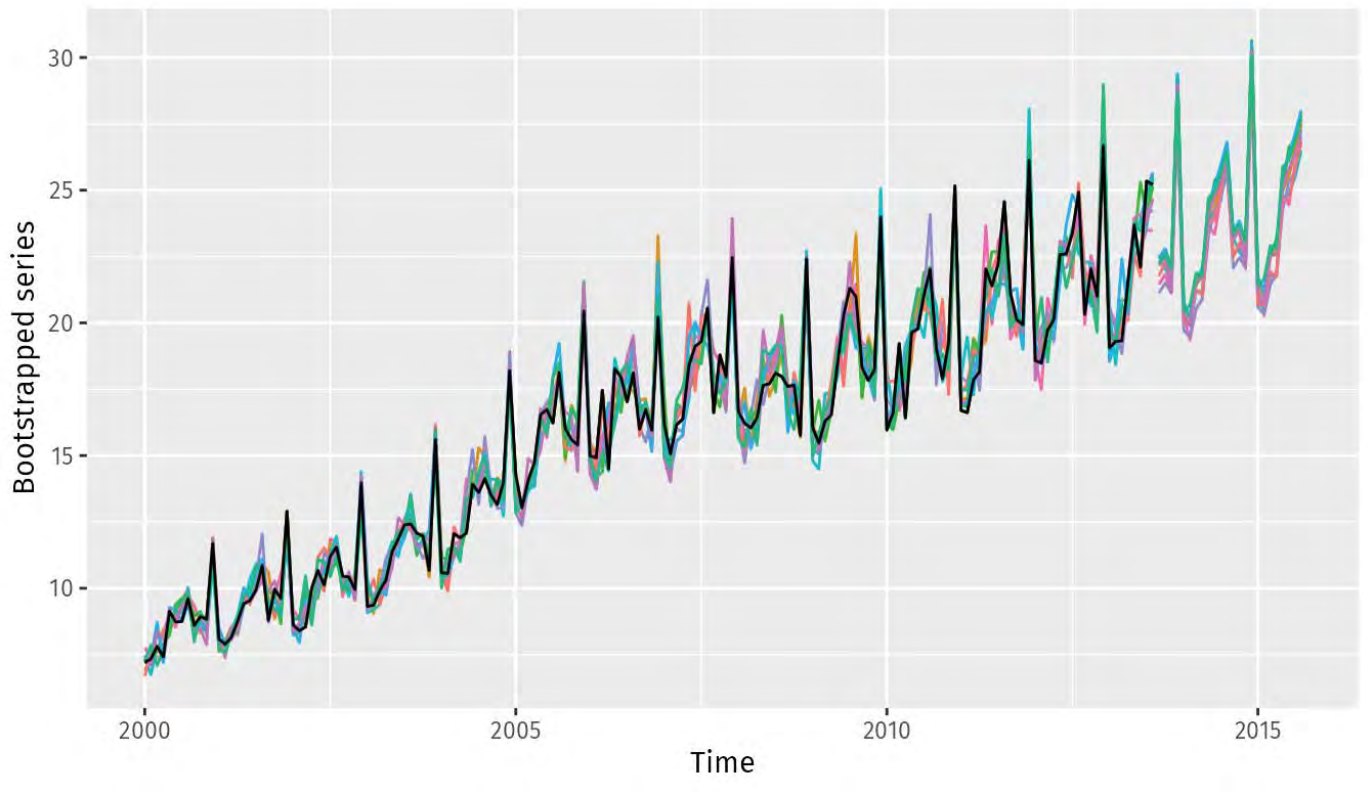


Figure 11.18: Forecasts of the ten bootstrapped series obtained using ETS models.

The average of these forecasts gives the bagged forecasts of the original data. The whole procedure can be handled with the `baggedETS()` function. By default, 100 bootstrapped series are used, and the length of the blocks used for obtaining bootstrapped residuals is set to 24 for monthly data. The resulting forecasts are shown in Figure 11.19.

```
etsfc <- debitcards %>% ets() %>% forecast(h=36)
baggedfc <- debitcards %>% baggedETS() %>% forecast(h=36)
autoplot(debitcards) +
  autolayer(baggedfc, series="BaggedETS", PI=FALSE) +
  autolayer(etsfc, series="ETS", PI=FALSE) +
  guides(colour=guide_legend(title="Forecasts"))
```

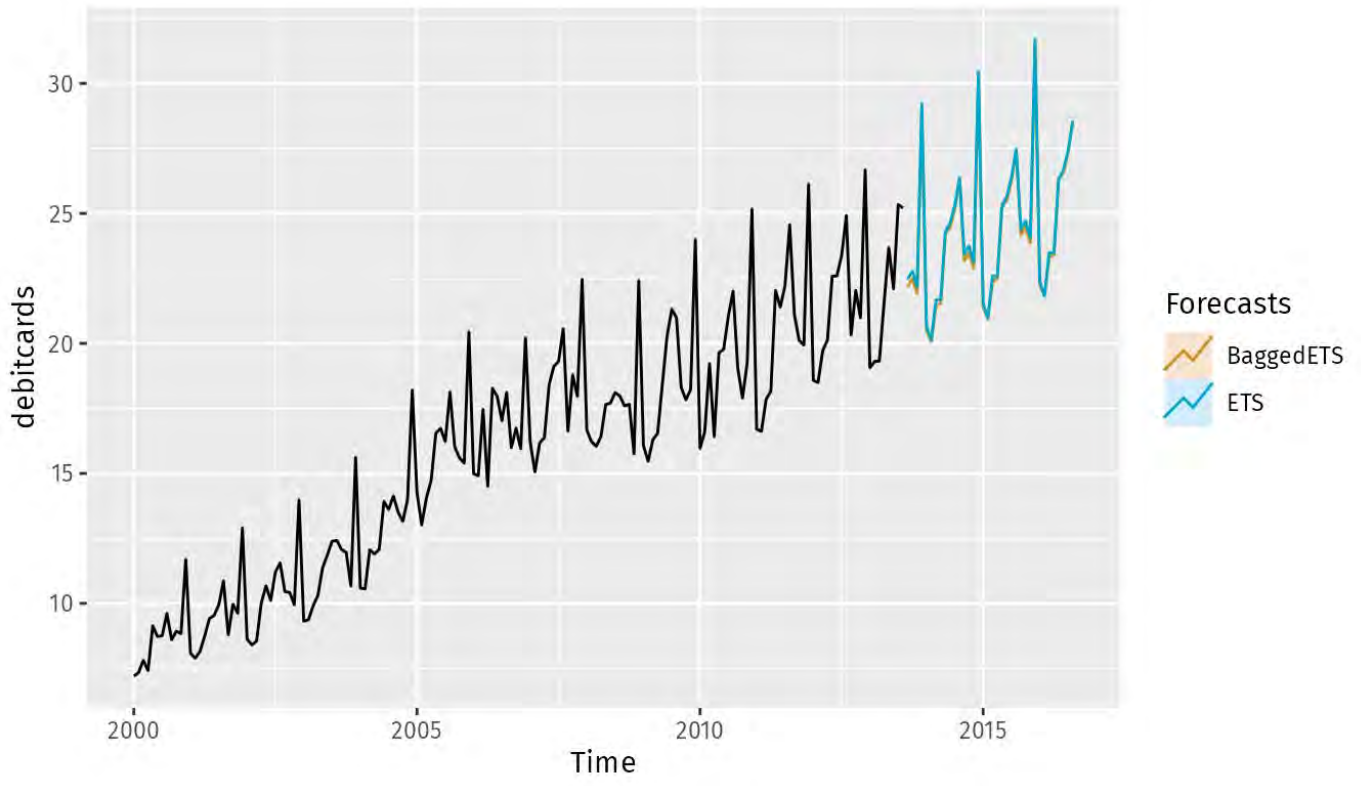


Figure 11.19: Comparing bagged ETS forecasts (the average of 100 bootstrapped forecast) and ETS applied directly to the data.

In this case, it makes little difference. Bergmeir, Hyndman, & Benítez (2016) show that, on average, bagging gives better forecasts than just applying `ets()` directly. Of course, it is slower because a lot more computation is required.

Bibliography

Bergmeir, C., Hyndman, R. J., & Benítez, J. M. (2016). Bagging exponential smoothing methods using STL decomposition and Box-Cox transformation. *International Journal of Forecasting*, *32*(2), 303–312. [DOI]

Hyndman, R. J., Koehler, A. B., Snyder, R. D., & Grose, S. (2002). A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*, *18*(3), 439–454. [DOI]

11.5 Exercises

1. Use the `tbats()` function to model your retail time series.

    a. Check the residuals and produce forecasts.
    b. Does this completely automated approach work for these data?
    c. Have you saved any degrees of freedom by using Fourier terms rather than seasonal differencing?

2. Consider the weekly data on US finished motor gasoline products supplied (millions of barrels per day) (series `gasoline`):

    a. Fit a TBATS model to these data.
    b. Check the residuals and produce forecasts.
    c. Could you model these data using any of the other methods we have considered in this book?

3. Experiment with using `nnetar()` on your retail data and other data we have considered in previous chapters.

11.6 Further reading

- De Livera et al. (2011) introduced the TBATS model and discuss the problem of complex seasonality in general.
- Pfaff (2008) provides a book-length overview of VAR modelling and other multivariate time series models.
- Neural networks for individual time series have not tended to produce good forecasts. Crone, Hibon, & Nikolopoulos (2011) discuss this issue in the context of a forecasting competition.
- Bootstrapping for time series is discussed in Lahiri (2003).
- Bagging for time series forecasting is relatively new. Bergmeir et al. (2016) is one of the few papers which addresses this topic.

Bibliography

Bergmeir, C., Hyndman, R. J., & Benítez, J. M. (2016). Bagging exponential smoothing methods using STL decomposition and Box-Cox transformation. *International Journal of Forecasting*, *32*(2), 303–312. [DOI]

Crone, S. F., Hibon, M., & Nikolopoulos, K. (2011). Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting*, *27*(3), 635–660. [DOI]

De Livera, A. M., Hyndman, R. J., & Snyder, R. D. (2011). Forecasting time series with complex seasonal patterns using exponential smoothing. *J American Statistical Association*, *106*(496), 1513–1527. [DOI]

Lahiri, S. N. (2003). *Resampling methods for dependent data*. New York, USA: Springer Science & Business Media. [Amazon]

Pfaff, B. (2008). *Analysis of integrated and cointegrated time series with R*. New York, USA: Springer Science & Business Media. [Amazon]

Chapter 12 Some practical forecasting issues

In this final chapter, we address many practical issues that arise in forecasting, and discuss some possible solutions. Several of these sections are adapted from Hyndsight blog posts.

12.1 Weekly, daily and sub-daily data

Weekly, daily and sub-daily data can be challenging for forecasting, although for different reasons.

Weekly data

Weekly data is difficult to work with because the seasonal period (the number of weeks in a year) is both large and non-integer. The average number of weeks in a year is 52.18. Most of the methods we have considered require the seasonal period to be an integer. Even if we approximate it by 52, most of the methods will not handle such a large seasonal period efficiently.

The simplest approach is to use an STL decomposition along with a non-seasonal method applied to the seasonally adjusted data (as discussed in Chapter 6). Here is an example using weekly data on US finished motor gasoline products supplied (in millions of barrels per day) from February 1991 to May 2005.

```
gasoline %>% stlf() %>% autoplot()
```

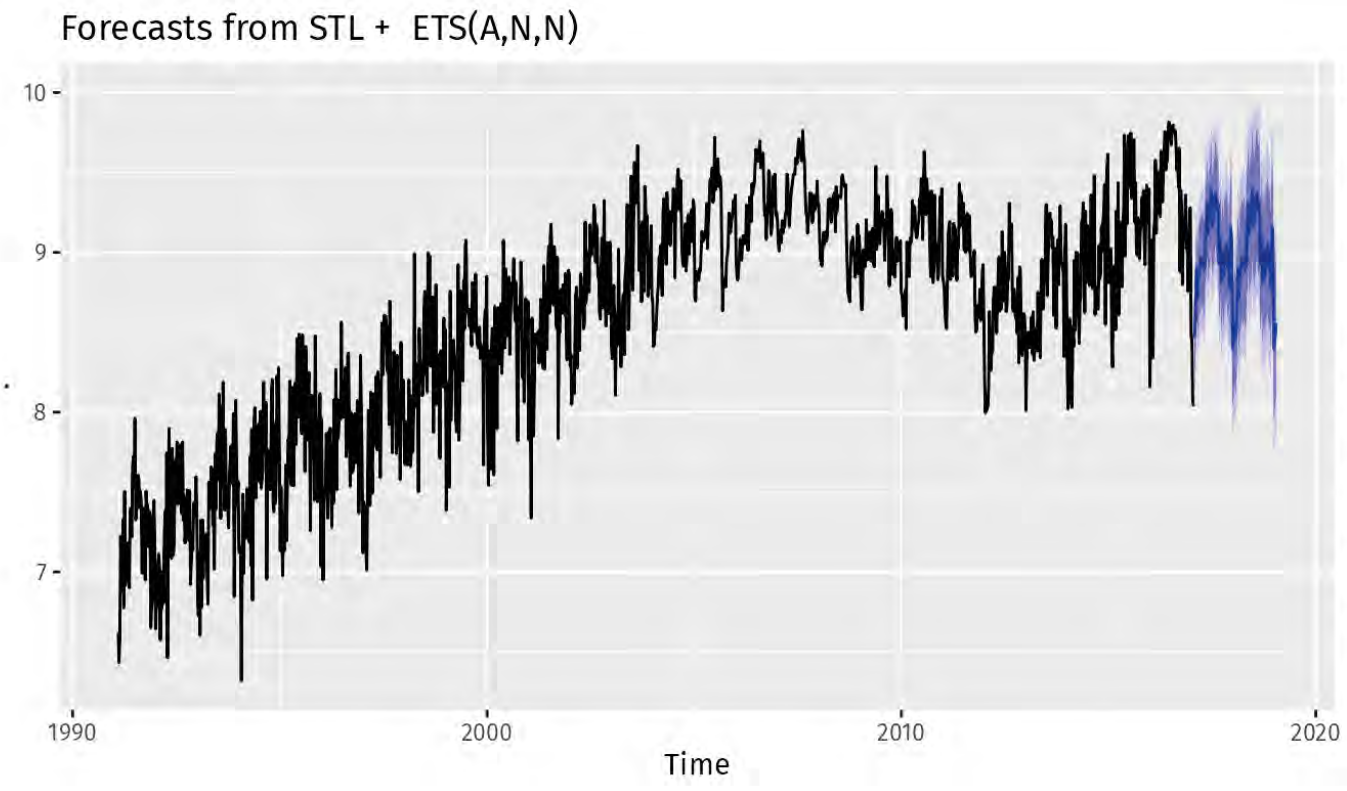


Figure 12.1: Forecasts for weekly US gasoline production using an STL decomposition with an ETS model for the seasonally adjusted data.

An alternative approach is to use a dynamic harmonic regression model, as discussed in Section 9.5. In the following example, the number of Fourier terms was selected by minimising the AICc. The order of the ARIMA model is also selected by minimising the AICc, although that is done within the `auto.arima()` function.

```
bestfit <- list(aicc=Inf)
for(K in seq(25)) {
  fit <- auto.arima(gasoline, xreg=fourier(gasoline, K=K),
    seasonal=FALSE)
  if(fit[["aicc"]] < bestfit[["aicc"]]) {
    bestfit <- fit
    bestK <- K
  }
}

fc <- forecast(bestfit,
  xreg=fourier(gasoline, K=bestK, h=104))
autoplot(fc)
```

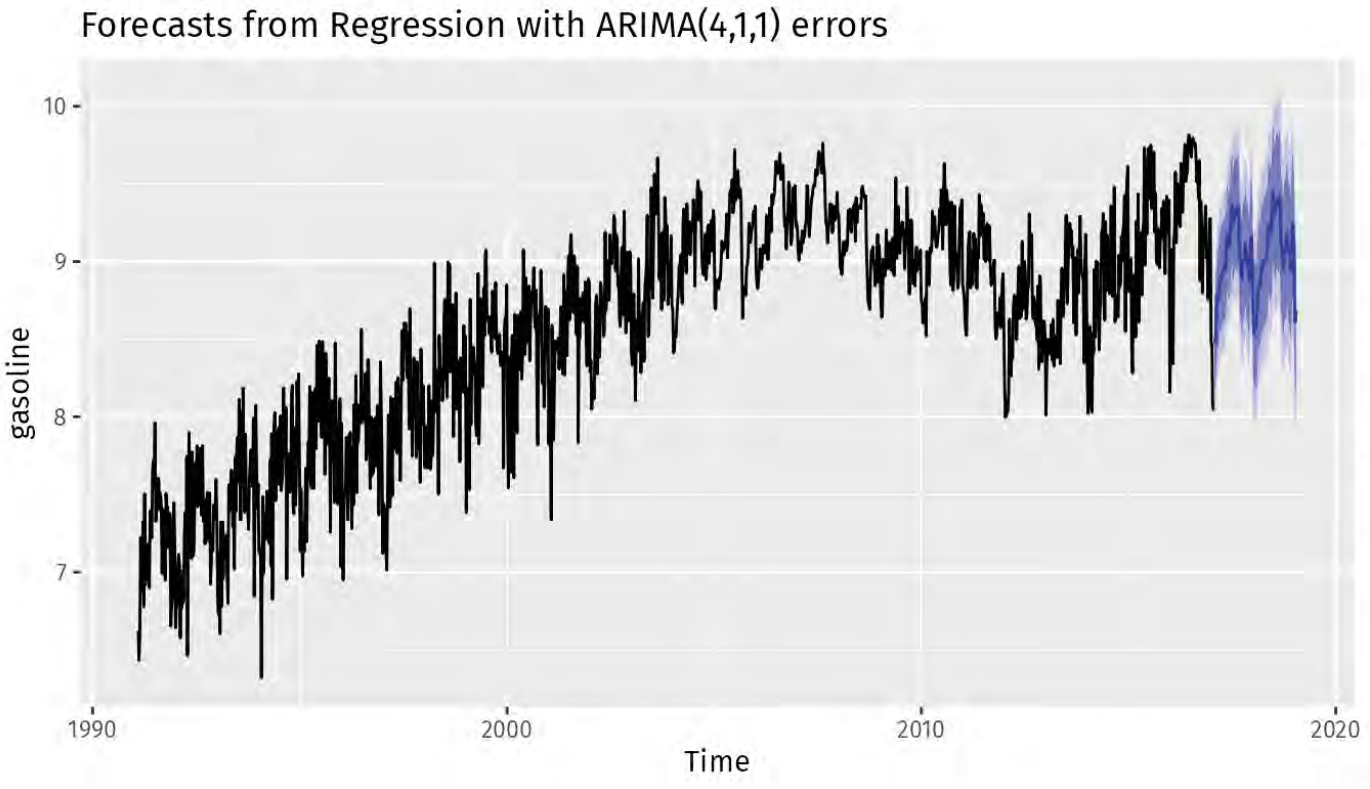


Figure 12.2: Forecasts for weekly US gasoline production using a dynamic harmonic regression model.

The fitted model has 18 pairs of Fourier terms and can be written as

$$y_t = bt + \sum_{j=1}^{18} \left[\alpha_j \sin\left(\frac{2\pi jt}{52.18}\right) + \beta_j \cos\left(\frac{2\pi jt}{52.18}\right) \right] + \eta_t$$

where η_t is an ARIMA(4,1,1) process. Because n_t is non-stationary, the model is actually estimated on the differences of the variables on both sides of this equation. There are 36 parameters to capture the seasonality which is rather a lot, but apparently required according to the AICc selection. The total number of degrees of freedom is 42 (the other six coming from the 4 AR parameters, 1 MA parameter, and the drift parameter).

A third approach is the TBATS model introduced in Section 11.1. This was the subject of Exercise 2 in Section 11.5. In this example, the forecasts are almost identical to the previous two methods.

The STL approach or TBATS model is preferable when the seasonality changes over time. The dynamic harmonic regression approach is preferable if there are covariates that are useful predictors as these can be added as additional regressors.

Daily and sub-daily data

Daily and sub-daily data are challenging for a different reason — they often involve multiple seasonal patterns, and so we need to use a method that handles such complex seasonality.

Of course, if the time series is relatively short so that only one type of seasonality is present, then it will be possible to use one of the single-seasonal methods we have discussed in previous chapters (e.g., ETS or a seasonal ARIMA model). But when the time series is long enough so that some of the longer seasonal periods become apparent, it will be necessary to use STL, dynamic harmonic regression or TBATS, as discussed in Section 11.1.

However, note that even these models only allow for regular seasonality. Capturing seasonality associated with moving events such as Easter, Id, or the Chinese New Year is more difficult. Even with monthly data, this can be tricky as the festivals can fall in either March or April (for Easter), in January or February (for the Chinese New Year), or at any time of the year (for Id).

The best way to deal with moving holiday effects is to use dummy variables. However, neither STL, ETS nor TBATS models allow for covariates. Amongst the models discussed in this book (and implemented in the **forecast** package for R), the only choice is a dynamic regression model, where the predictors include any dummy holiday effects (and possibly also the seasonality using Fourier terms).

12.2 Time series of counts

All of the methods discussed in this book assume that the data have a continuous sample space. But often data comes in the form of counts. For example, we may wish to forecast the number of customers who enter a store each day. We could have 0, 1, 2, , customers, but we cannot have 3.45693 customers.

In practice, this rarely matters provided our counts are sufficiently large. If the minimum number of customers is at least 100, then the difference between a continuous sample space $[100, \infty)$ and the discrete sample space $\{100, 101, 102, \ldots\}$ has no perceivable effect on our forecasts. However, if our data contains small counts $(0, 1, 2, \ldots)$, then we need to use forecasting methods that are more appropriate for a sample space of non-negative integers.

Such models are beyond the scope of this book. However, there is one simple method which gets used in this context, that we would like to mention. It is "Croston's method", named after its British inventor, John Croston, and first described in Croston (1972). Actually, this method does not properly deal with the count nature of the data either, but it is used so often, that it is worth knowing about it.

With Croston's method, we construct two new series from our original time series by noting which time periods contain zero values, and which periods contain non-zero values. Let q_i be the ith non-zero quantity, and let a_i be the time between q_{i-1} and q_i. Croston's method involves separate simple exponential smoothing forecasts on the two new series a and q. Because the method is usually applied to time series of demand for items, q is often called the "demand" and a the "inter-arrival time".

If $\hat{q}_{i+1|i}$ and $\hat{a}_{i+1|i}$ are the one-step forecasts of the $(i+1)$th demand and inter-arrival time respectively, based on data up to demand i, then Croston's method gives

$$\hat{q}_{i+1|i} = (1-\alpha)\hat{q}_{i|i-1} + \alpha q_i, \tag{12.1}$$

$$\hat{a}_{i+1|i} = (1-\alpha)\hat{a}_{i|i-1} + \alpha a_i. \tag{12.2}$$

The smoothing parameter α takes values between 0 and 1 and is assumed to be the same for both equations. Let j be the time for the last observed positive observation. Then the h-step ahead forecast for the demand at time $T+h$, is given by the ratio

$$\hat{y}_{T+h|T} = q_{j+1|j}/a_{j+1|j}. \tag{1}$$

There are no algebraic results allowing us to compute prediction intervals for this method, because the method does not correspond to any statistical model (Shenstone & Hyndman, 2005).

The `croston()` function produces forecasts using Croston's method. It simply uses $\alpha = 0.1$ by default, and ℓ_0 is set to be equal to the first observation in each of the series. This is consistent with the way Croston envisaged the method being used.

Example: lubricant sales

Several years ago, we assisted an oil company with forecasts of monthly lubricant sales. One of the time series is shown in the table below. The data contain small counts, with many months registering no sales at all, and only small numbers of items sold in other months.

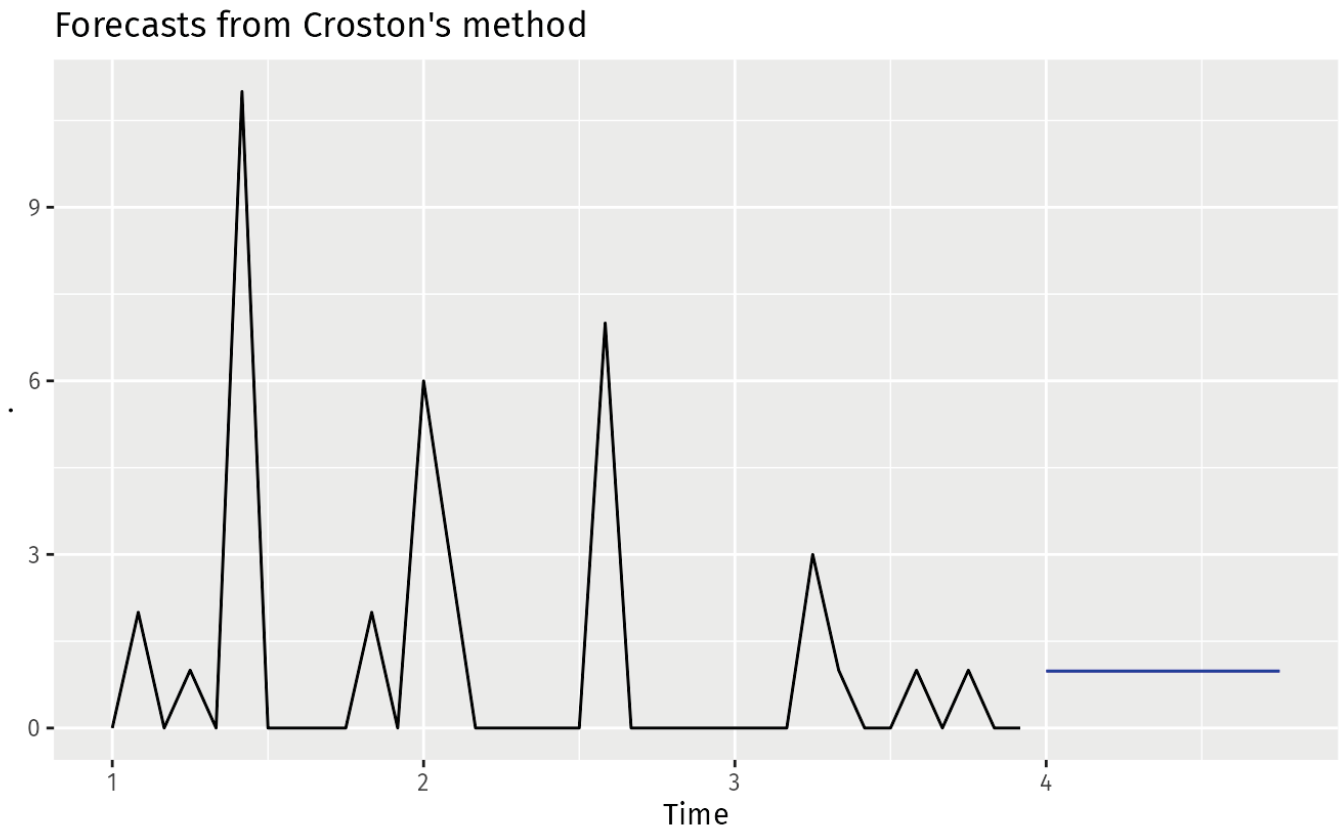
Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	0	2	0	1	0	11	0	0	0	0	2	0
2	6	3	0	0	0	0	0	7	0	0	0	0
3	0	0	0	3	1	0	0	1	0	1	0	0

There are 11 non-zero demand values in the series, denoted by q. The corresponding arrival series a are also shown in the following table.

i	1	2	3	4	5	6	7	8	9	10	11
q	2	1	11	2	6	3	7	3	1	1	1
a	2	2	2	5	2	1	6	8	1	3	2

Applying Croston's method gives the demand forecast 2.750 and the arrival forecast 2.793. So the forecast of the original series is $\hat{y}_{T+h|T} = 2.750/2.793 = 0.985$. In practice, R does these calculations for you:

```
productC %>% croston() %>% autoplot()
```



An implementation of Croston's method with more facilities (including parameter estimation) is available in the **tsintermittent** package for R.

Forecasting models that deal more directly with the count nature of the data are described in Christou & Fokianos (2015).

Bibliography

Christou, V., & Fokianos, K. (2015). On count time series prediction. *Journal of Statistical Computation and Simulation*, *85*(2), 357–373. [DOI]

Croston, J. D. (1972). Forecasting and stock control for intermittent demands. *Operational Research Quarterly*, *23*(3), 289–303. [DOI]

Shenstone, L., & Hyndman, R. J. (2005). Stochastic models underlying Croston's method for intermittent demand forecasting. *Journal of Forecasting*, *24*(6), 389–402. [DOI]

12.3 Ensuring forecasts stay within limits

It is common to want forecasts to be positive, or to require them to be within some specified range $[a, b]$. Both of these situations are relatively easy to handle using transformations.

Positive forecasts

To impose a positivity constraint, simply work on the log scale, by specifying the Box-Cox parameter $\lambda = 0$. For example, consider the real price of a dozen eggs (1900-1993; in cents):

```
eggs %>%
  ets(model="AAN", damped=FALSE, lambda=0) %>%
  forecast(h=50, biasadj=TRUE) %>%
  autoplot()
```

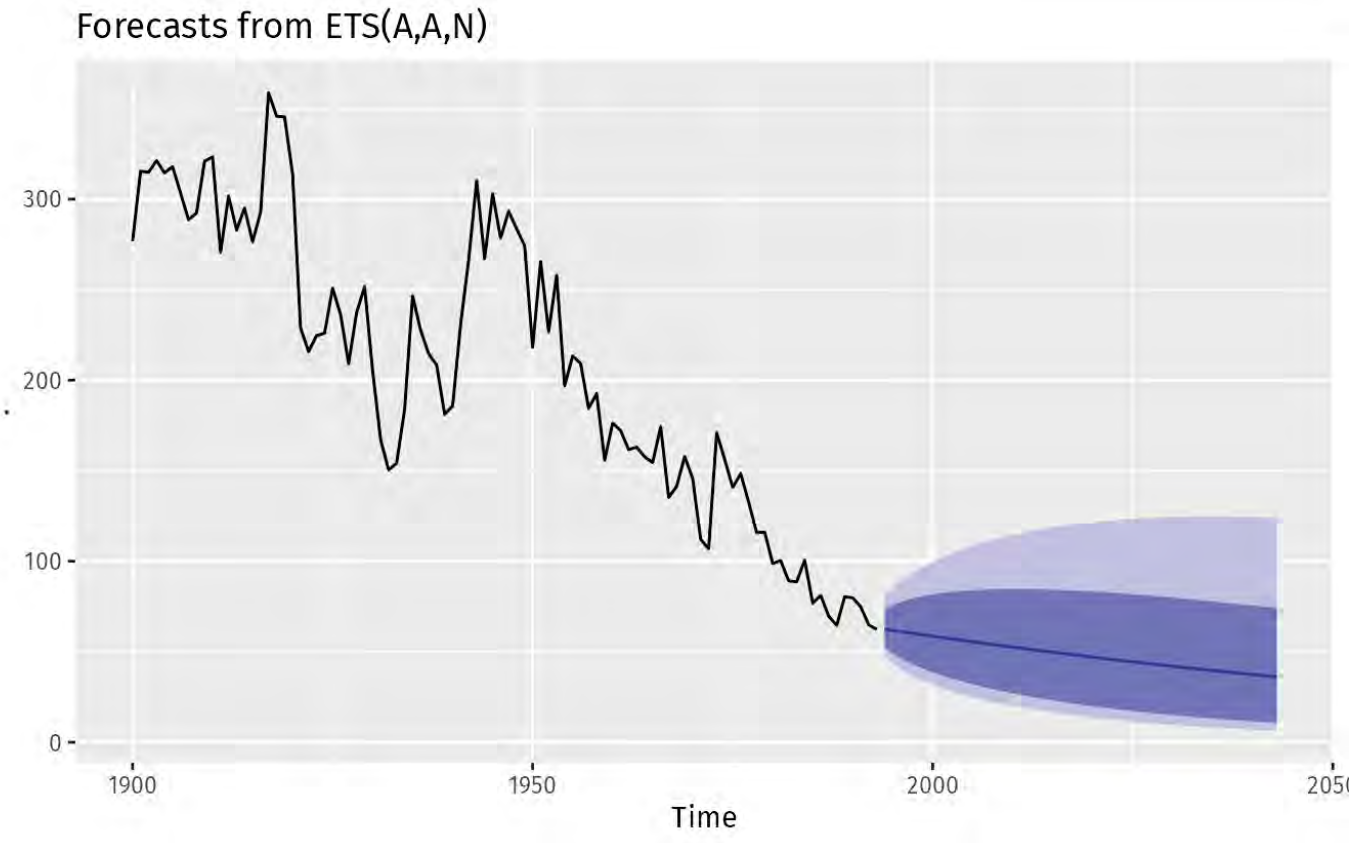


Figure 12.3: Forecasts for the price of a dozen eggs, constrained to be positive using a Box-Cox transformation.

Because we set `biasadj=TRUE`, the forecasts are the means of the forecast distributions.

Forecasts constrained to an interval

To see how to handle data constrained to an interval, imagine that the egg prices were constrained to lie within $a = 50$ and $b = 400$. Then we can transform the data using a scaled logit transform which maps (a, b) to the whole real line:

$$y = \log\left(\frac{x-a}{b-x}\right),$$

where x is on the original scale and y is the transformed data. To reverse the transformation, we will use

$$x = \frac{(b-a)e^y}{1+e^y} + a.$$

This is not a built-in transformation, so we will need to do more work.

```
# Bounds
a <- 50
b <- 400
# Transform data and fit model
fit <- log((eggs-a)/(b-eggs)) %>%
  ets(model="AAN", damped=FALSE)
fc <- forecast(fit, h=50)
# Back-transform forecasts
fc[["mean"]] <- (b-a)*exp(fc[["mean"]]) /
  (1+exp(fc[["mean"]])) + a
fc[["lower"]] <- (b-a)*exp(fc[["lower"]]) /
 (1+exp(fc[["lower"]])) + a
fc[["upper"]] <- (b-a)*exp(fc[["upper"]]) /
 (1+exp(fc[["upper"]])) + a
fc[["x"]] <- eggs
# Plot result on original scale
autoplot(fc)
```

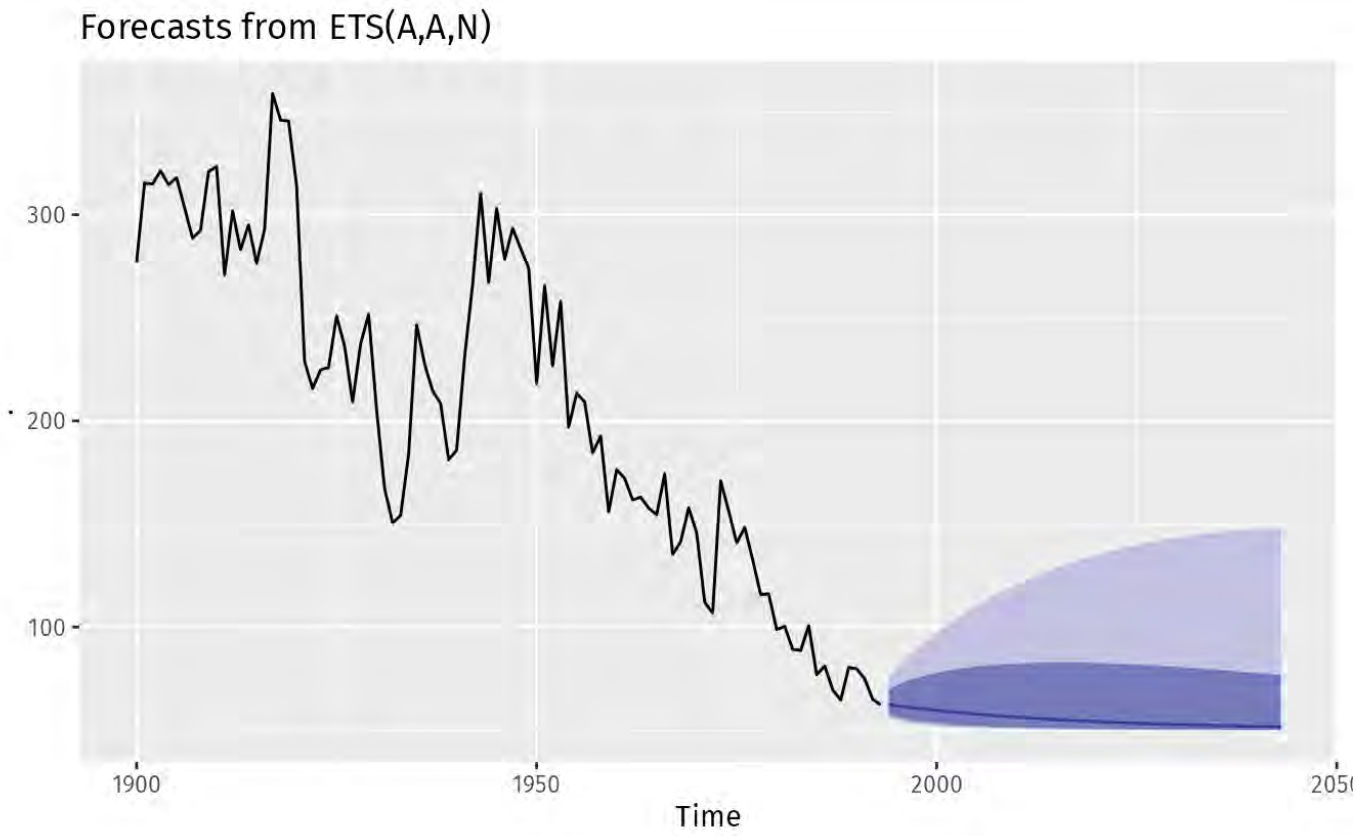


Figure 12.4: Forecasts for the price of a dozen eggs, constrained to be lie between 50 and 400.

No bias-adjustment has been used here, so the forecasts are the medians of the future distributions. The prediction intervals from these transformations have the same coverage probability as on the transformed scale, because quantiles are preserved under monotonically increasing transformations.

The prediction intervals lie above 50 due to the transformation. As a result of this artificial (and unrealistic) constraint, the forecast distributions have become extremely skewed.

12.4 Forecast combinations

An easy way to improve forecast accuracy is to use several different methods on the same time series, and to average the resulting forecasts. Nearly 50 years ago, John Bates and Clive Granger wrote a famous paper (Bates & Granger, 1969), showing that combining forecasts often leads to better forecast accuracy. Twenty years later, Clemen (1989) wrote

> The results have been virtually unanimous: combining multiple forecasts leads to increased forecast accuracy. In many cases one can make dramatic performance improvements by simply averaging the forecasts.

While there has been considerable research on using weighted averages, or some other more complicated combination approach, using a simple average has proven hard to beat.

Here is an example using monthly expenditure on eating out in Australia, from April 1982 to September 2017. We use forecasts from the following models: ETS, ARIMA, STL-ETS, NNAR, and TBATS; and we compare the results using the last 5 years (60 months) of observations.

```
train <- window(auscafe, end=c(2012,9))
h <- length(auscafe) - length(train)
ETS <- forecast(ets(train), h=h)
ARIMA <- forecast(auto.arima(train, lambda=0, biasadj=TRUE),
  h=h)
STL <- stlf(train, lambda=0, h=h, biasadj=TRUE)
NNAR <- forecast(nnetar(train), h=h)
TBATS <- forecast(tbats(train, biasadj=TRUE), h=h)
Combination <- (ETS[["mean"]] + ARIMA[["mean"]] +
  STL[["mean"]] + NNAR[["mean"]] + TBATS[["mean"]])/5
```

```
autoplot(auscafe) +
  autolayer(ETS, series="ETS", PI=FALSE) +
  autolayer(ARIMA, series="ARIMA", PI=FALSE) +
  autolayer(STL, series="STL", PI=FALSE) +
  autolayer(NNAR, series="NNAR", PI=FALSE) +
  autolayer(TBATS, series="TBATS", PI=FALSE) +
  autolayer(Combination, series="Combination") +
  xlab("Year") + ylab("$ billion") +
  ggtitle("Australian monthly expenditure on eating out")
```

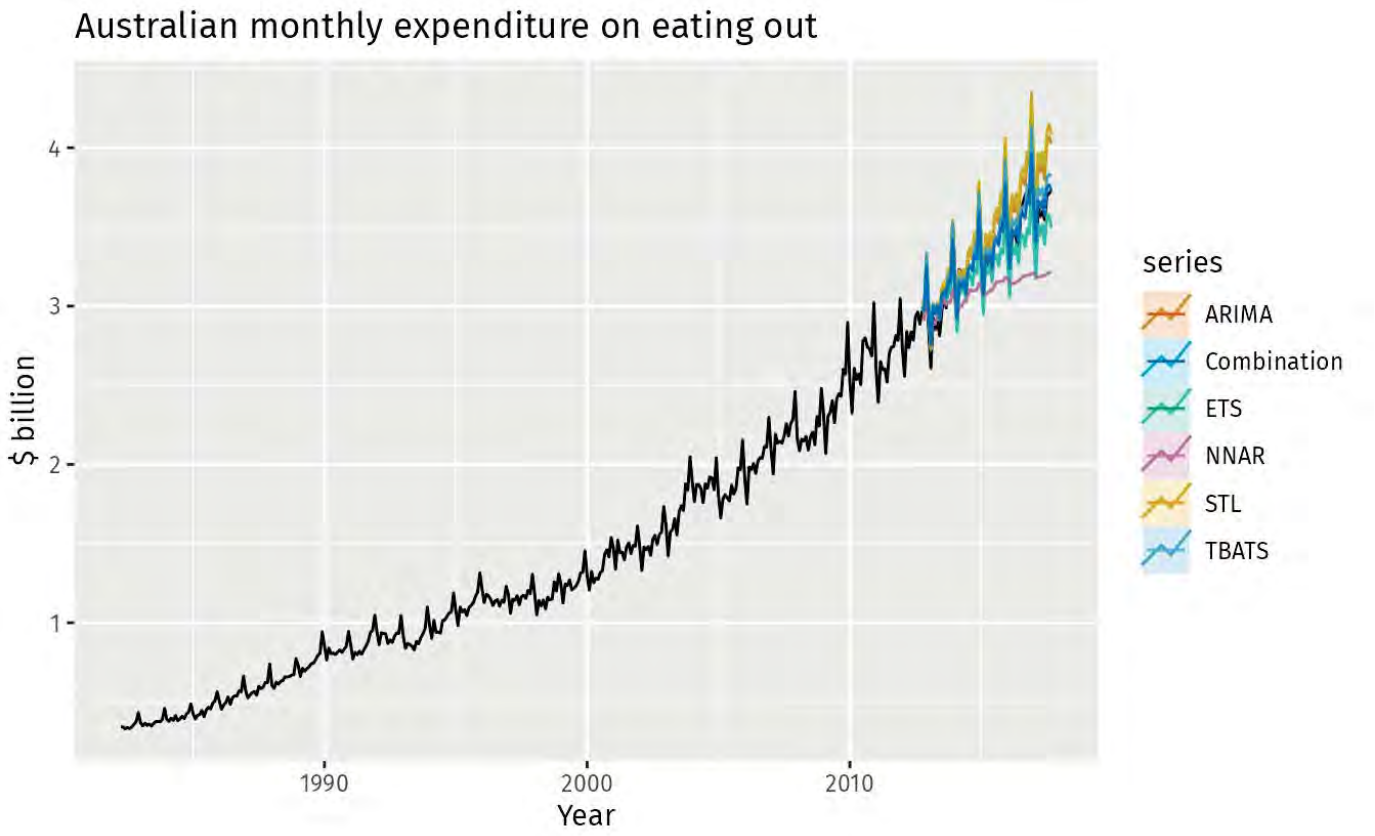


Figure 12.5: Point forecasts from various methods applied to Australian monthly expenditure on eating out.

```
c(ETS = accuracy(ETS, auscafe)["Test set","RMSE"],
  ARIMA = accuracy(ARIMA, auscafe)["Test set","RMSE"],
  `STL-ETS` = accuracy(STL, auscafe)["Test set","RMSE"],
  NNAR = accuracy(NNAR, auscafe)["Test set","RMSE"],
  TBATS = accuracy(TBATS, auscafe)["Test set","RMSE"],
  Combination =
    accuracy(Combination, auscafe)["Test set","RMSE"])
#>         ETS       ARIMA     STL-ETS        NNAR       TBATS Combinatic
#>     0.13700     0.15920     0.19310     0.31769     0.09406     0.0716
```

TBATS does particularly well with this series, but the combination approach is even better. For other data, TBATS may be quite poor, while the combination approach is almost always close to, or better than, the best component method.

Bibliography

Bates, J. M., & Granger, C. W. J. (1969). The combination of forecasts. *Operational Research Quarterly*, *20*(4), 451–468. [DOI]

Clemen, R. (1989). Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, *5*(4), 559–583. [DOI]

12.5 Prediction intervals for aggregates

A common problem is to forecast the aggregate of several time periods of data, using a model fitted to the disaggregated data. For example, we may have monthly data but wish to forecast the total for the next year. Or we may have weekly data, and want to forecast the total for the next four weeks.

If the point forecasts are means, then adding them up will give a good estimate of the total. But prediction intervals are more tricky due to the correlations between forecast errors.

A general solution is to use simulations. Here is an example using ETS models applied to Australian monthly gas production data, assuming we wish to forecast the aggregate gas demand in the next six months.

```
# First fit a model to the data
fit <- ets(gas/1000)
# Forecast six months ahead
fc <- forecast(fit, h=6)
# Simulate 10000 future sample paths
nsim <- 10000
h <- 6
sim <- numeric(nsim)
for(i in seq_len(nsim))
  sim[i] <- sum(simulate(fit, future=TRUE, nsim=h))
meanagg <- mean(sim)
```

The mean of the simulations is close to the sum of the individual forecasts:

```
sum(fc[["mean"]][1:6])
#> [1] 281.8
meanagg
#> [1] 281.7
```

Prediction intervals are also easy to obtain:

```
#80% interval:
quantile(sim, prob=c(0.1, 0.9))
#> 10% 90%
#> 263 301
#95% interval:
quantile(sim, prob=c(0.025, 0.975))
#>  2.5% 97.5%
#> 254.1 311.4
```

12.6 Backcasting

Sometimes it is useful to "backcast" a time series — that is, forecast in reverse time. Although there are no in-built R functions to do this, it is easy to implement. The following functions reverse a `ts` object and a `forecast` object.

```
# Function to reverse time
reverse_ts <- function(y)
{
  ts(rev(y), start=tsp(y)[1L], frequency=frequency(y))
}
# Function to reverse a forecast
reverse_forecast <- function(object)
{
  h <- length(object[["mean"]])
  f <- frequency(object[["mean"]])
  object[["x"]] <- reverse_ts(object[["x"]])
  object[["mean"]] <- ts(rev(object[["mean"]]),
    end=tsp(object[["x"]])[1L]-1/f, frequency=f)
  object[["lower"]] <- object[["lower"]][h:1L,]
  object[["upper"]] <- object[["upper"]][h:1L,]
  return(object)
}
```

Then we can apply these functions to backcast any time series. Here is an example applied to quarterly retail trade in the Euro area. The data are from 1996 to 2011. We backcast to predict the years 1994-1995.

```
# Backcast example
euretail %>%
  reverse_ts() %>%
  auto.arima() %>%
  forecast() %>%
  reverse_forecast() -> bc
autoplot(bc) +
  ggtitle(paste("Backcasts from",bc[["method"]]))
```

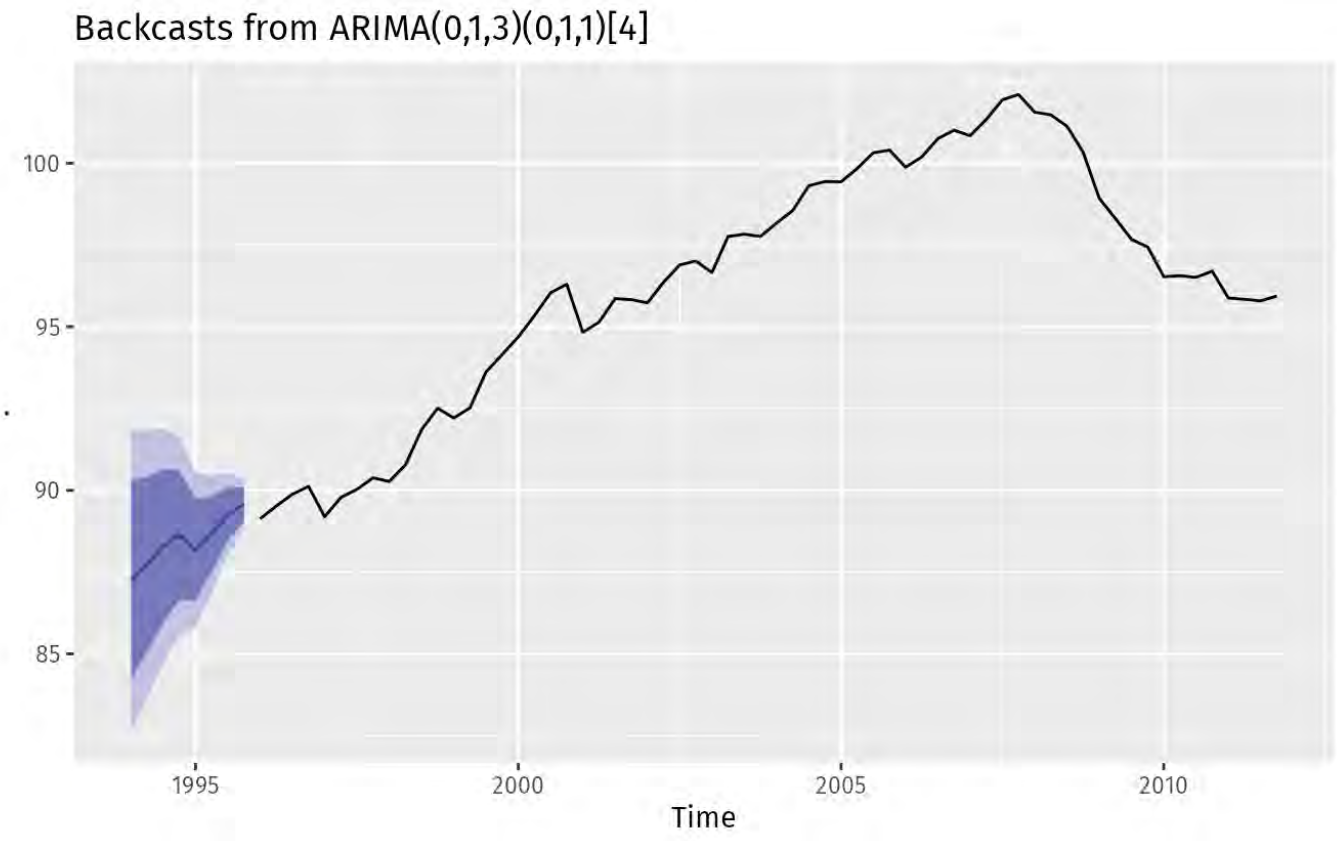


Figure 12.6: Backcasts for quarterly retail trade in the Euro area using an ARIMA model.

12.7 Very long and very short time series

Forecasting very short time series

We often get asked how *few* data points can be used to fit a time series model. As with almost all sample size questions, there is no easy answer. It depends on the *number of model parameters to be estimated and the amount of randomness in the data*. The sample size required increases with the number of parameters to be estimated, and the amount of noise in the data.

Some textbooks provide rules-of-thumb giving minimum sample sizes for various time series models. These are misleading and unsubstantiated in theory or practice. Further, they ignore the underlying variability of the data and often overlook the number of parameters to be estimated as well. There is, for example, no justification whatever for the magic number of 30 often given as a minimum for ARIMA modelling. The only theoretical limit is that we need more observations than there are parameters in our forecasting model. However, in practice, we usually need substantially more observations than that.

Ideally, we would test if our chosen model performs well out-of-sample compared to some simpler approaches. However, with short series, there is not enough data to allow some observations to be withheld for testing purposes, and even time series cross validation can be difficult to apply. The AICc is particularly useful here, because it is a proxy for the one-step forecast out-of-sample MSE. Choosing the model with the minimum AICc value allows both the number of parameters and the amount of noise to be taken into account.

What tends to happen with short series is that the AIC suggests simple models because anything with more than one or two parameters will produce poor forecasts due to the estimation error. We applied the `auto.arima()` function to all the series from the M-competition with fewer than 20 observations. There were a total of 144 series, of which 54 had models with zero parameters (white noise and random walks), 73 had models with one parameter, 15 had models with two parameters, and 2 series had models with three parameters. Interested readers can carry out the same exercise using the following code.

```
library(Mcomp)
library(purrr)
n <- map_int(M1, function(x) {length(x[["x"]])})
M1[n < 20] %>%
  map_int(function(u) {
    u[["x"]] %>%
      auto.arima() %>%
      coefficients() %>%
      length()
  }) %>%
  table()
```

Forecasting very long time series

Most time series models do not work well for very long time series. The problem is that real data do not come from the models we use. When the number of observations is not large (say up to about 200) the models often work well as an approximation to whatever process generated the data. But eventually we will have enough data that the difference between the true process and the model starts to become more obvious. An additional problem is that the optimisation of the parameters becomes more time consuming because of the number of observations involved.

What to do about these issues depends on the purpose of the model. A more flexible and complicated model could be used, but this still assumes that the model structure will work over the whole period of the data. A better approach is usually to allow the model itself to change over time. ETS models are designed to handle this situation by allowing the trend and seasonal terms to evolve over time. ARIMA models with differencing have a similar property. But dynamic regression models do not allow any evolution of model components.

If we are only interested in forecasting the next few observations, one simple approach is to throw away the earliest observations and only fit a model to the most recent observations. Then an inflexible model can work well because there is not enough time for the relationships to change substantially.

For example, we fitted a dynamic harmonic regression model to 26 years of weekly gasoline production in Section 12.1. It is, perhaps, unrealistic to assume that the seasonal pattern remains the same over nearly three decades. So we could simply fit

a model to the most recent years instead.

12.8 Forecasting on training and test sets

Typically, we compute one-step forecasts on the training data (the "fitted values") and multi-step forecasts on the test data. However, occasionally we may wish to compute multi-step forecasts on the training data, or one-step forecasts on the test data.

Multi-step forecasts on training data

We normally define fitted values to be one-step forecasts on the training set (see Section 3.3), but a similar idea can be used for multi-step forecasts. We will illustrate the method using an ARIMA(2,1,1)(0,1,2)$_{12}$ model for the Australian eating-out expenditure. The last five years are used for a test set, and the forecasts are plotted in Figure 12.7.

```
training <- subset(auscafe, end=length(auscafe)-61)
test <- subset(auscafe, start=length(auscafe)-60)
cafe.train <- Arima(training, order=c(2,1,1),
  seasonal=c(0,1,2), lambda=0)
cafe.train %>%
  forecast(h=60) %>%
  autoplot() + autolayer(test)
```

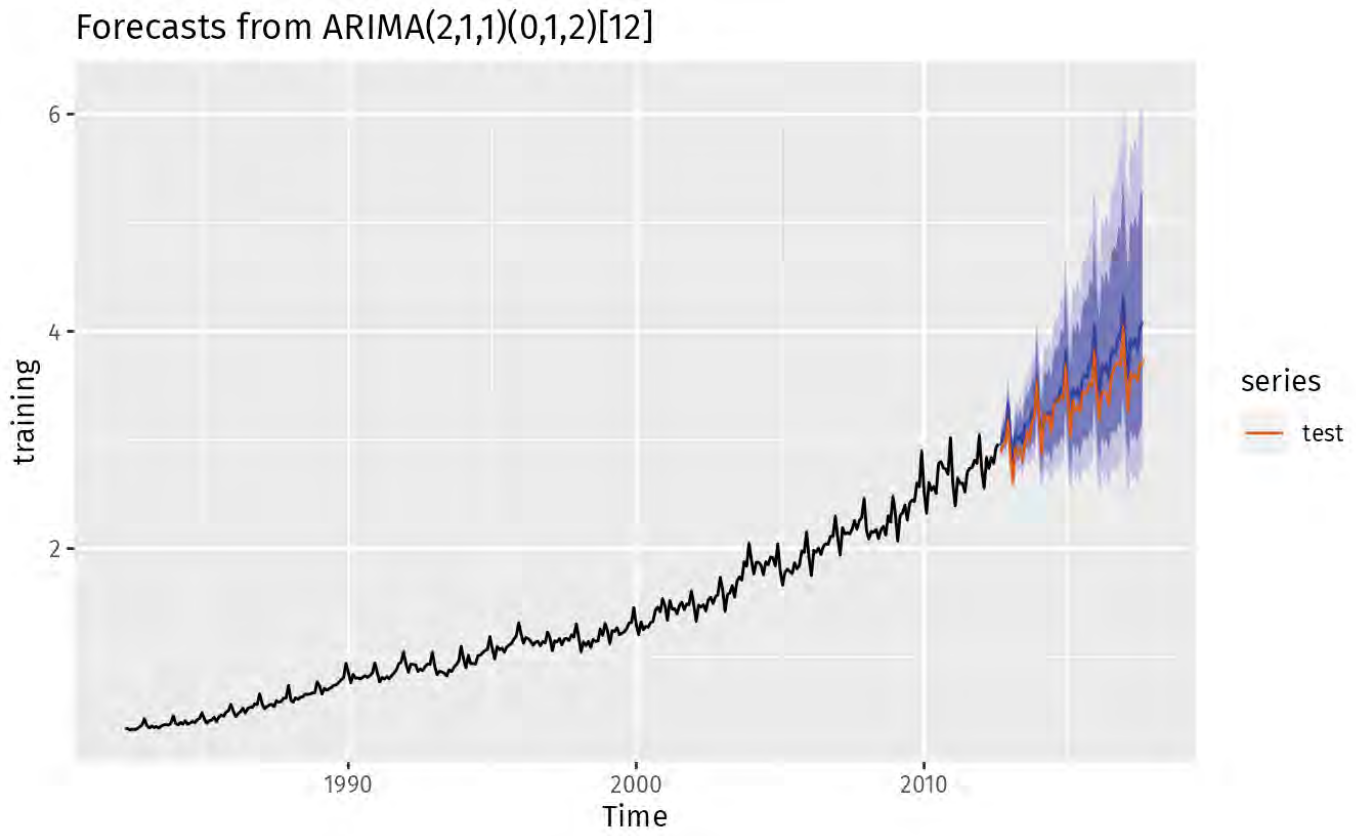


Figure 12.7: Forecasts from an ARIMA model fitted to the Australian café training data.

The `fitted()` function has an `h` argument to allow for h-step "fitted values" on the training set. Figure 12.8 is a plot of 12-step (one year) forecasts on the training set. Because the model involves both seasonal (lag 12) and first (lag 1) differencing, it is not possible to compute these forecasts for the first few observations.

```
autoplot(training, series="Training data") +
  autolayer(fitted(cafe.train, h=12),
    series="12-step fitted values")
```

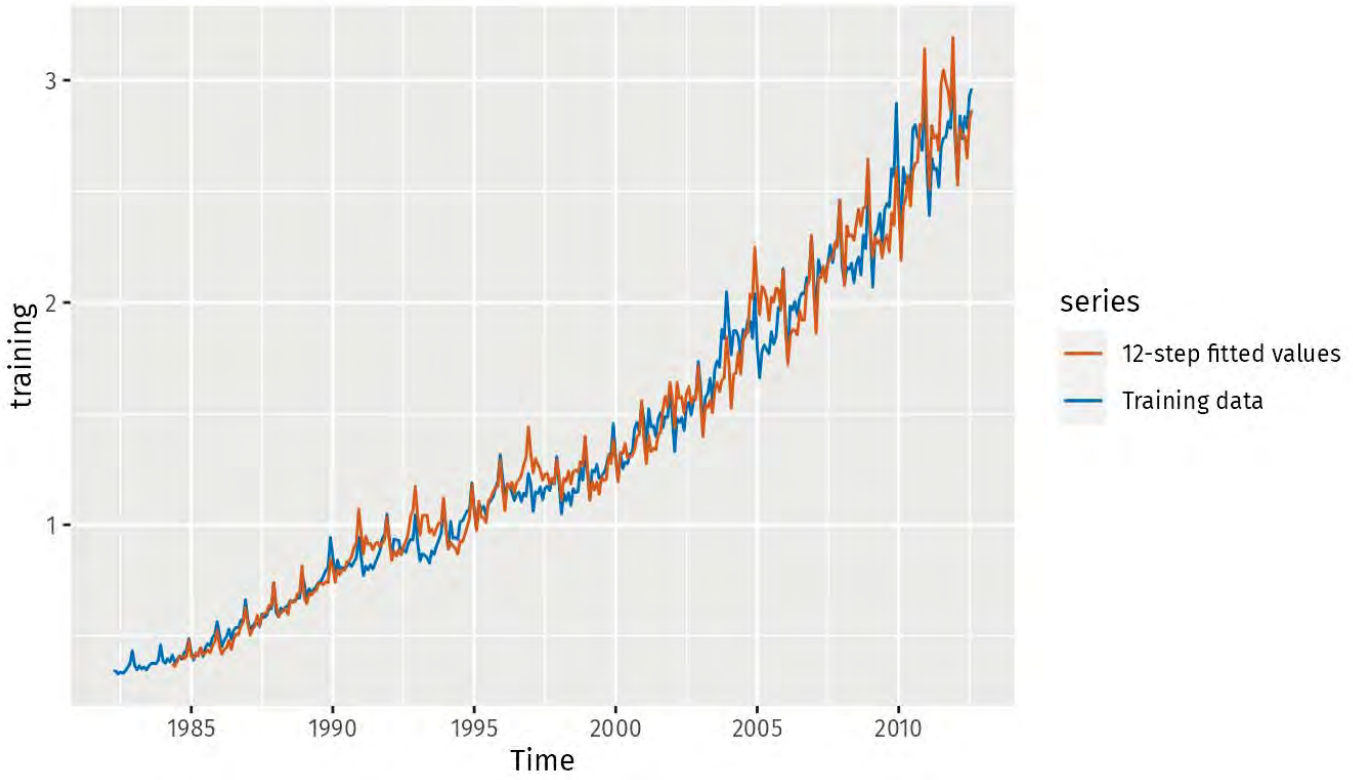


Figure 12.8: Twelve-step fitted values from an ARIMA model fitted to the Australian café training data.

One-step forecasts on test data

It is common practice to fit a model using training data, and then to evaluate its performance on a test data set. The way this is usually done means the comparisons on the test data use different forecast horizons. In the above example, we have used the last sixty observations for the test data, and estimated our forecasting model on the training data. Then the forecast errors will be for 1-step, 2-steps, ..., 60-steps ahead. The forecast variance usually increases with the forecast horizon, so if we are simply averaging the absolute or squared errors from the test set, we are combining results with different variances.

One solution to this issue is to obtain 1-step errors on the test data. That is, we still use the training data to estimate any parameters, but when we compute forecasts on the test data, we use all of the data preceding each observation (both training and test data). So our training data are for times $1, 2, \dots, T-60$. We estimate the model on these data, but then compute $\hat{y}_{T-60+h|T-61+h}$, for $h = 1, \dots, T-1$. Because the test data are not used to estimate the parameters, this still gives us a "fair" forecast. For the `ets()`, `Arima()`, `tbats()` and `nnetar()` functions, these calculations are easily carried out using the `model` argument.

Using the same ARIMA model used above, we now apply the model to the test data.

```
cafe.test <- Arima(test, model=cafe.train)
accuracy(cafe.test)
#>                       ME    RMSE     MAE      MPE  MAPE   MASE     ACF1
#> Training set -0.002622 0.04591 0.03413 -0.07301 1.002 0.1899 -0.05704
```

Note that `Arima()` does not re-estimate in this case. Instead, the model obtained previously (and stored as `cafe.train`) is applied to the test data. Because the model was not re-estimated, the "residuals" obtained here are actually one-step forecast errors. Consequently, the results produced from the `accuracy()` command are actually on the test set (despite the output saying "Training set").

12.9 Dealing with missing values and outliers

Real data often contains missing values, outlying observations, and other messy features. Dealing with them can sometimes be troublesome.

Missing values

Missing data can arise for many reasons, and it is worth considering whether the missingness will induce bias in the forecasting model. For example, suppose we are studying sales data for a store, and missing values occur on public holidays when the store is closed. The following day may have increased sales as a result. If we fail to allow for this in our forecasting model, we will most likely under-estimate sales on the first day after the public holiday, but over-estimate sales on the days after that. One way to deal with this kind of situation is to use a dynamic regression model, with dummy variables indicating if the day is a public holiday or the day after a public holiday. No automated method can handle such effects as they depend on the specific forecasting context.

In other situations, the missingness may be essentially random. For example, someone may have forgotten to record the sales figures, or the data recording device may have malfunctioned. If the timing of the missing data is not informative for the forecasting problem, then the missing values can be handled more easily.

Some methods allow for missing values without any problems. For example, the naïve forecasting method continues to work, with the most recent non-missing value providing the forecast for the future time periods. Similarly, the other benchmark methods introduced in Section 3.1 will all produce forecasts when there are missing values present in the historical data. The R functions for ARIMA models, dynamic regression models and NNAR models will also work correctly without causing errors. However, other modelling functions do not handle missing values including `ets()`, `stlf()`, and `tbats()`.

When missing values cause errors, there are at least two ways to handle the problem. First, we could just take the section of data after the last missing value, assuming there is a long enough series of observations to produce meaningful forecasts.

Alternatively, we could replace the missing values with estimates. The `na.interp()` function is designed for this purpose.

The `gold` data contains daily morning gold prices from 1 January 1985 to 31 March 1989. This series was provided to us as part of a consulting project; it contains 34 missing values as well as one apparently incorrect value. Figure 12.9 shows estimates of the missing observations in red.

```
gold2 <- na.interp(gold)
autoplot(gold2, series="Interpolated") +
  autolayer(gold, series="Original") +
  scale_colour_manual(
    values=c(`Interpolated`="red",`Original`="gray"))
```

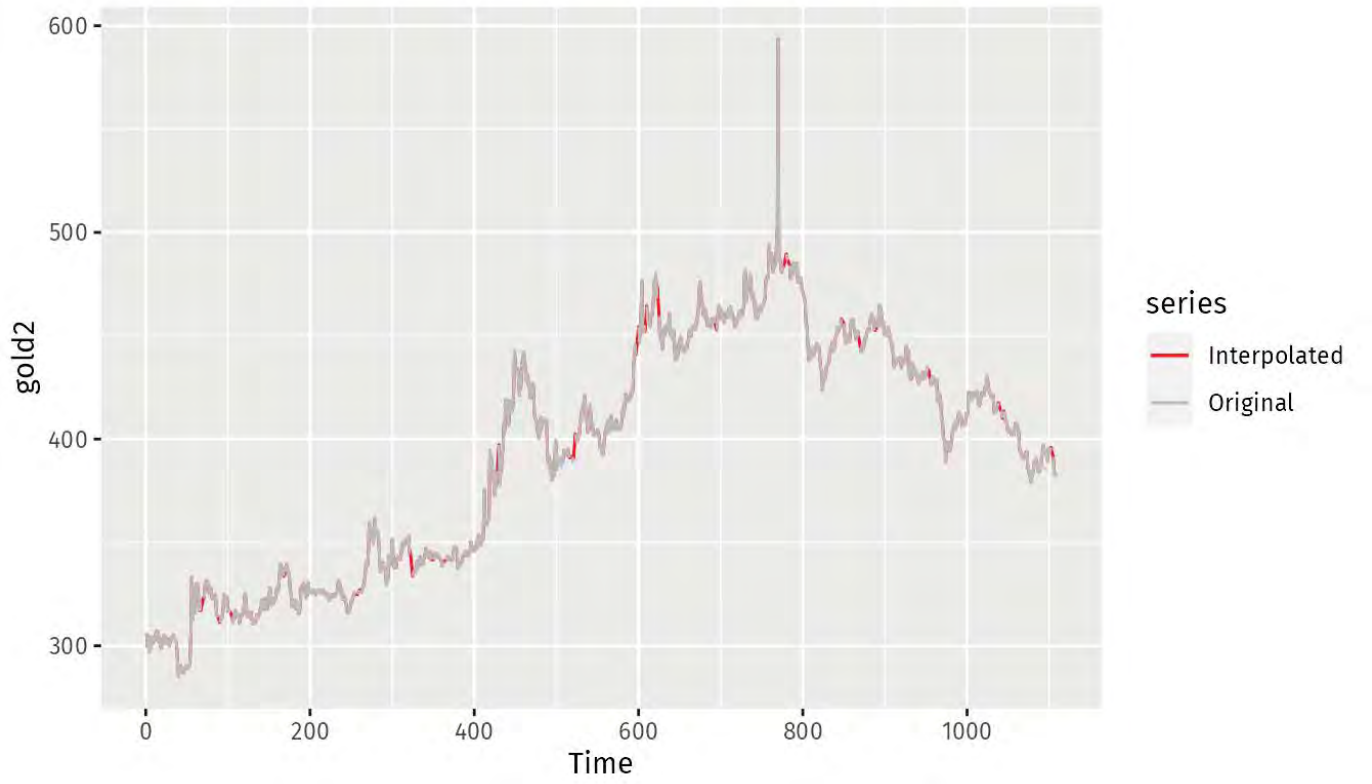


Figure 12.9: Daily morning gold prices for 1108 consecutive trading days beginning on 1 January 1985 and ending on 31 March 1989.

For non-seasonal data like this, simple linear interpolation is used to fill in the missing sections. For seasonal data, an STL decomposition is used to estimate the seasonal component, and the seasonally adjusted series are linear interpolated. More sophisticated missing value interpolation is provided in the **imputeTS** package.

Outliers

Outliers are observations that are very different from the majority of the observations in the time series. They may be errors, or they may simply be unusual. (See Section 5.3 for a discussion of outliers in a regression context.) All of the methods we have considered in this book will not work well if there are extreme outliers in the data. In this case, we may wish to replace them with missing values, or with an estimate that is more consistent with the majority of the data.

Simply replacing outliers without thinking about why they have occurred is a dangerous practice. They may provide useful information about the process that produced the data, and which should be taken into account when forecasting.

However, if we are willing to assume that the outliers are genuinely errors, or that they won't occur in the forecasting period, then replacing them can make the forecasting task easier.

The `tsoutliers()` function is designed to identify outliers, and to suggest potential replacement values. In the `gold` data shown in Figure 12.9, there is an apparently outlier on day 770:

```
tsoutliers(gold)
#> $index
#> [1] 770
#>
#> $replacements
#> [1] 494.9
```

Closer inspection reveals that the neighbouring observations are close to $100 less than the apparent outlier.

```
gold[768:772]
#> [1] 495.00 502.75 593.70 487.05 487.75
```

Most likely, this was a transcription error, and the correct value should have been $493.70.

Another useful function is `tsclean()` which identifies and replaces outliers, and also replaces missing values. Obviously this should be used with some caution, but it does allow us to use forecasting models that are sensitive to outliers, or which do not handle missing values. For example, we could use the `ets()` function on the `gold` series, after applying `tsclean()`.

```
gold %>%
  tsclean() %>%
  ets() %>%
  forecast(h=50) %>%
  autoplot()
```

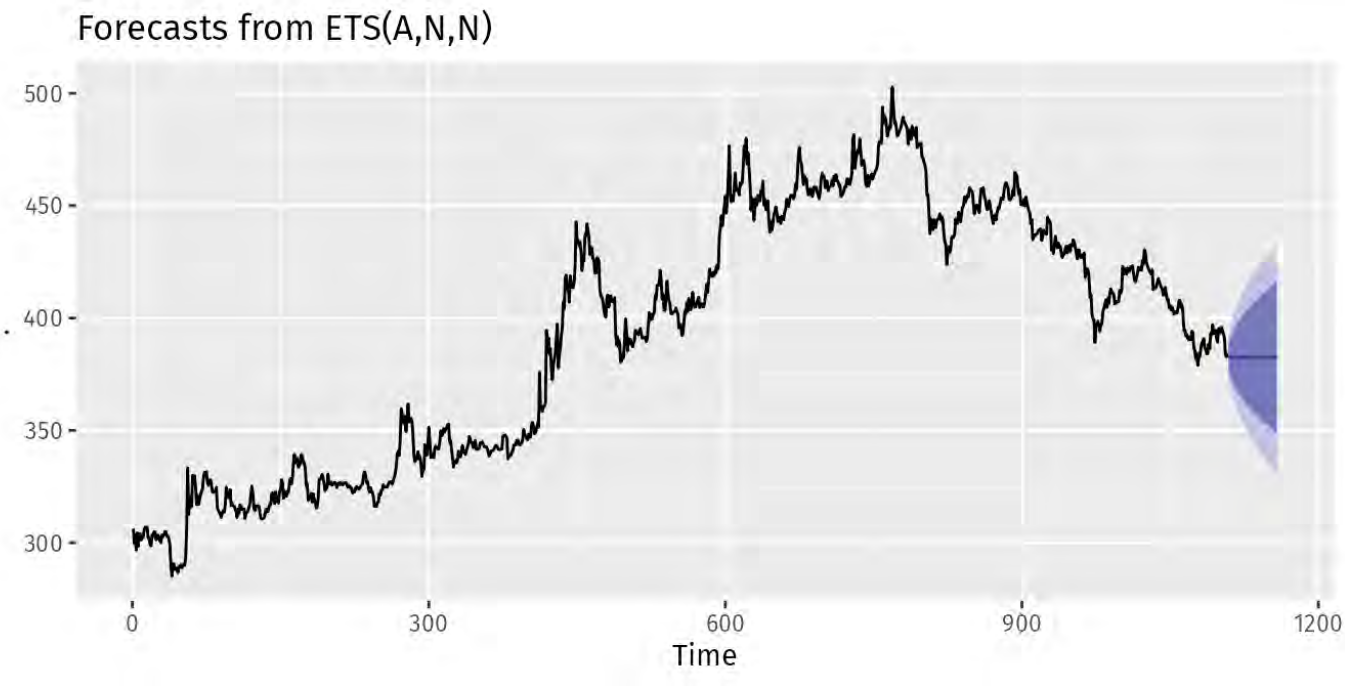


Figure 12.10: Forecasts from an ETS model for the gold price data after removing an outlier.

Notice that the outlier and missing values have been replaced with estimates.

12.10 Further reading

So many diverse topics are discussed in this chapter, that it is not possible to point to specific references on all of them. The last chapter in Ord et al. (2017) also covers "Forecasting in practice" and discusses other issues that might be of interest to readers.

Bibliography

Ord, J. K., Fildes, R., & Kourentzes, N. (2017). *Principles of business forecasting* (2nd ed.). Wessex Press Publishing Co. [Amazon]

Appendix: Using R

This book uses R and is designed to be used with R. R is free, available on almost every operating system, and there are thousands of add-on packages to do almost anything you could ever want to do. We recommend you use R with RStudio.

Installing R and RStudio

1. Download and install R.
2. Download and install RStudio.
3. Run RStudio. On the "Packages" tab, click on "Install packages" and install the package **fpp2** (make sure "install dependencies" is checked).

That's it! You should now be ready to go.

R examples in this book

We provide R code for most examples in shaded boxes like this:

```
autoplot(a10)
h02 %>% ets() %>% forecast() %>% summary()
```

These examples assume that you have the **fpp2** package loaded (and that you are using at least v2.3 of the package). So you should use the command `library(fpp2)` before you try any examples provided here. (This needs to be done at the start of every R session.) Sometimes we also assume that the R code that appears earlier in the same section of the book has also been run; so it is best to work through the R code in the order provided within each section.

Getting started with R

If you have never previously used R, please work through the first section (chapters 1-8) of "R for Data Science" by Garrett Grolemund and Hadley Wickham. While this does not cover time series or forecasting, it will get you used to the basics of the R language. The Coursera R Programming course is also highly recommended.

You will learn how to use R for forecasting using the exercises in this book.

Bibliography

Armstrong, J. S. (1978). *Long-range forecasting: From crystal ball to computer*. John Wiley & Sons. [Amazon]

Armstrong, J. S. (Ed.). (2001). *Principles of forecasting: A handbook for researchers and practitioners*. Kluwer Academic Publishers. [Amazon]

Athanasopoulos, G., Ahmed, R. A., & Hyndman, R. J. (2009). Hierarchical forecasts for Australian domestic tourism. *International Journal of Forecasting*, *25*, 146–166. [DOI]

Athanasopoulos, G., & Hyndman, R. J. (2008). Modelling and forecasting Australian domestic tourism. *Tourism Management*, *29*(1), 19–31. [DOI]

Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. *European Journal of Operational Research*, *262*(1), 60–74. [DOI]

Athanasopoulos, G., Poskitt, D. S., & Vahid, F. (2012). Two canonical VARMA forms: Scalar component models vis-à-vis the echelon form. *Econometric Reviews*, *31*(1), 60–83. [DOI]

Bates, J. M., & Granger, C. W. J. (1969). The combination of forecasts. *Operational Research Quarterly*, *20*(4), 451–468. [DOI]

Bergmeir, C., Hyndman, R. J., & Benítez, J. M. (2016). Bagging exponential smoothing methods using STL decomposition and Box-Cox transformation. *International Journal of Forecasting*, *32*(2), 303–312. [DOI]

Bergmeir, C., Hyndman, R. J., & Koo, B. (2018). A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics and Data Analysis*, *120*, 70–83. [DOI]

Bickel, P. J., & Doksum, K. A. (1981). An analysis of transformations revisited. *Journal of the American Statistical Association*, *76*(374), 296–311.

Box, G. E. P., & Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society. Series B, Statistical Methodology*, *26*(2), 211–252. [DOI]

Box, G. E. P., & Jenkins, G. M. (1970). *Time series analysis: Forecasting and control*. San Francisco: Holden-Day.

Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: Forecasting and control* (5th ed). Hoboken, New Jersey: John Wiley &

Sons. [Amazon]
Brockwell, P. J., & Davis, R. A. (2016). *Introduction to time series and forecasting* (3rd ed). New York, USA: Springer. [Amazon]
Brown, R. G. (1959). *Statistical forecasting for inventory control*. McGraw/Hill.
Buehler, R., Messervey, D., & Griffin, D. (2005). Collaborative planning and prediction: Does group discussion affect optimistic biases in time estimation? *Organizational Behavior and Human Decision Processes*, *97*(1), 47–63. [DOI]
Christou, V., & Fokianos, K. (2015). On count time series prediction. *Journal of Statistical Computation and Simulation*, *85*(2), 357–373. [DOI]
Clemen, R. (1989). Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, *5*(4), 559–583. [DOI]
Cleveland, R. B., Cleveland, W. S., McRae, J. E., & Terpenning, I. J. (1990). STL: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, *6*(1), 3–33. http://bit.ly/stl1990
Cleveland, W. S. (1993). *Visualizing data*. Hobart Press. [Amazon]
Crone, S. F., Hibon, M., & Nikolopoulos, K. (2011). Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting*, *27*(3), 635–660. [DOI]
Croston, J. D. (1972). Forecasting and stock control for intermittent demands. *Operational Research Quarterly*, *23*(3), 289–303. [DOI]
Dagum, E. B., & Bianconcini, S. (2016). *Seasonal adjustment methods and real time trend-cycle estimation*. Springer. [Amazon]
De Livera, A. M., Hyndman, R. J., & Snyder, R. D. (2011). Forecasting time series with complex seasonal patterns using exponential smoothing. *J American Statistical Association*, *106*(496), 1513–1527. [DOI]
Eroglu, C., & Croxton, K. L. (2010). Biases in judgmental adjustments of statistical forecasts: The role of individual differences. *International Journal of Forecasting*, *26*(1), 116–133. [DOI]
Fan, S., & Hyndman, R. J. (2012). Short-term load forecasting based on a semi-parametric additive model. *IEEE Transactions on Power Systems*, *27*(1), 134–141. [DOI]
Fildes, R., & Goodwin, P. (2007a). Against your better judgment? How organizations can improve their use of management judgment in forecasting. *Interfaces*, *37*(6), 570–576. [DOI]
Fildes, R., & Goodwin, P. (2007b). Good and bad judgment in forecasting: Lessons from four companies. *Foresight: The International Journal of Applied Forecasting*, (8), 5–10.

Franses, P. H., & Legerstee, R. (2013). Do statistical forecasting models for SKU-level data benefit from including past expert knowledge? *International Journal of Forecasting*, *29*(1), 80–87. [DOI]

Gardner, E. S. (1985). Exponential smoothing: The state of the art. *Journal of Forecasting*, *4*(1), 1–28. [DOI]

Gardner, E. S. (2006). Exponential smoothing: The state of the art — Part II. *International Journal of Forecasting*, *22*, 637–666. [DOI]

Gardner, E. S., & McKenzie, E. (1985). Forecasting trends in time series. *Management Science*, *31*(10), 1237–1246. [DOI]

Goodwin, P., & Wright, G. (2009). *Decision analysis for management judgment* (4th ed). Chichester: John Wiley & Sons. [Amazon]

Green, K. C., & Armstrong, J. S. (2007). Structured analogies for forecasting. *International Journal of Forecasting*, *23*(3), 365–376. [DOI]

Gross, C. W., & Sohl, J. E. (1990). Disaggregation methods to expedite product line forecasting. *Journal of Forecasting*, *9*, 233–254. [DOI]

Groves, R. M., Fowler, F. J., Couper, M. P., Lepkowski, J. M., Singer, E., & Tourangeau, R. (2009). *Survey methodology* (2nd ed). John Wiley & Sons. [Amazon]

Hamilton, J. D. (1994). *Time series analysis*. Princeton University Press, Princeton. [Amazon]

Harrell, F. E. (2015). *Regression modeling strategies: With applications to linear models, logistic and ordinal regression, and survival analysis* (2nd ed). New York, USA: Springer. [Amazon]

Harris, R., & Sollis, R. (2003). *Applied time series modelling and forecasting*. Chichester, UK: John Wiley & Sons. [Amazon]

Harvey, N. (2001). Improving judgment in forecasting. In J. S. Armstrong (Ed.), *Principles of forecasting: A handbook for researchers and practitioners* (pp. 59–80). Boston, MA: Kluwer Academic Publishers. [DOI]

Holt, C. C. (1957). *Forecasting seasonals and trends by exponentially weighted averages* (O.N.R. Memorandum No. 52). Carnegie Institute of Technology, Pittsburgh USA. [DOI]

Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G., & Shang, H. L. (2011). Optimal combination forecasts for hierarchical time series. *Computational Statistics and Data Analysis*, *55*(9), 2579–2589. [DOI]

Hyndman, R. J., & Fan, S. (2010). Density forecasting for long-term peak electricity demand. *IEEE Transactions on Power Systems*, *25*(2), 1142–1153.

Hyndman, R. J. & Athanasopoulos, G. (2021). Forecasting: Principles and Practice (3rd ed)

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, *27*(1), 1–22. [DOI]

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679–688. [DOI]

Hyndman, R. J., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). *Forecasting with exponential smoothing: The state space approach*. Berlin: Springer-Verlag. http://www.exponentialsmoothing.net

Hyndman, R. J., Koehler, A. B., Snyder, R. D., & Grose, S. (2002). A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*, *18*(3), 439–454. [DOI]

Hyndman, R. J., Lee, A., & Wang, E. (2016). Fast computation of reconciled forecasts for hierarchical and grouped time series. *Computational Statistics and Data Analysis*, *97*, 16–32. [DOI]

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2014). *An introduction to statistical learning: With applications in R*. New York: Springer. [Amazon]

Kahn, K. B. (2006). *New product forecasting: An applied approach*. M.E. Sharp. [Amazon]

Kahneman, D., & Lovallo, D. (1993). Timid choices and bold forecasts: A cognitive perspective on risk taking. *Management Science*, *39*(1), 17–31. [DOI]

Kwiatkowski, D., Phillips, P. C. B., Schmidt, P., & Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, *54*(1-3), 159–178. [DOI]

Lahiri, S. N. (2003). *Resampling methods for dependent data*. New York, USA: Springer Science & Business Media. [Amazon]

Lawrence, M., Goodwin, P., O'Connor, M., & Önkal, D. (2006). Judgmental forecasting: A review of progress over the last 25 years. *International Journal of Forecasting*, *22*(3), 493–518. [DOI]

Lütkepohl, H. (2005). *New introduction to multiple time series analysis*. Berlin: Springer-Verlag. [Amazon]

Lütkepohl, H. (2007). General-to-specific or specific-to-general modelling? An opinion on current econometric terminology. *Journal of Econometrics*, *136*(1), 234–319. [DOI]

Morwitz, V. G., Steckel, J. H., & Gupta, A. (2007). When do purchase intentions predict sales? *International Journal of Forecasting*, *23*(3), 347–364. [DOI]

Önkal, D., Sayım, K. Z., & Gönül, M. S. (2013). Scenarios as channels of forecast advice. *Technological Forecasting and Social Change*, *80*(4), 772–788. [DOI]

Ord, J. K., Fildes, R., & Kourentzes, N. (2017). *Principles of business forecasting* (2nd ed.). Wessex Press Publishing Co. [Amazon]

Pankratz, A. E. (1991). *Forecasting with dynamic regression models*. New York, USA: John Wiley & Sons. [Amazon]

Pegels, C. C. (1969). Exponential forecasting: Some new variations. *Management Science*, *15*(5), 311–315. [DOI]

Peña, D., Tiao, G. C., & Tsay, R. S. (Eds.). (2001). *A course in time series analysis*. New York, USA: John Wiley & Sons. [Amazon]

Pfaff, B. (2008). *Analysis of integrated and cointegrated time series with R*. New York, USA: Springer Science & Business Media. [Amazon]

Randall, D. M., & Wolff, J. A. (1994). The time interval in the intention-behaviour relationship: Meta-analysis. *British Journal of Social Psychology*, *33*(4), 405–418. [DOI]

Rowe, G. (2007). A guide to Delphi. *Foresight: The International Journal of Applied Forecasting*, (8), 11–16.

Rowe, G., & Wright, G. (1999). The Delphi technique as a forecasting tool: Issues and analysis. *International Journal of Forecasting*, *15*(4), 353–375. [DOI]

Sanders, N., Goodwin, P., Önkal, D., Gönül, M. S., Harvey, N., Lee, A., & Kjolso, L. (2005). When and how should statistical forecasts be judgmentally adjusted? *Foresight: The International Journal of Applied Forecasting*, *1*(1), 5–23. http://www.forecastpro.com/Trends/pdf/Nada%20Sanders%20Judgmental%20Adjustments%20to%20Statistical%20Forecasts%20July%202008.pdf

Sheather, S. J. (2009). *A modern approach to regression with R*. New York, USA: Springer. [Amazon]

Shenstone, L., & Hyndman, R. J. (2005). Stochastic models underlying Croston's method for intermittent demand forecasting. *Journal of Forecasting*, *24*(6), 389–402. [DOI]

Taylor, J. W. (2003). Exponential smoothing with a damped multiplicative trend. *International Journal of Forecasting*, *19*(4), 715–725. [DOI]

Theodosiou, M. (2011). Forecasting monthly and quarterly time series using STL decomposition. *International Journal of Forecasting*, *27*(4), 1178–1195. [DOI]

Unwin, A. (2015). *Graphical data analysis with R*. Chapman; Hall/CRC. [Amazon]

Wang, X., Smith, K. A., & Hyndman, R. J. (2006). Characteristic-based clustering for time series data. *Data Mining and Knowledge Discovery*, *13*(3), 335–364. [DOI]

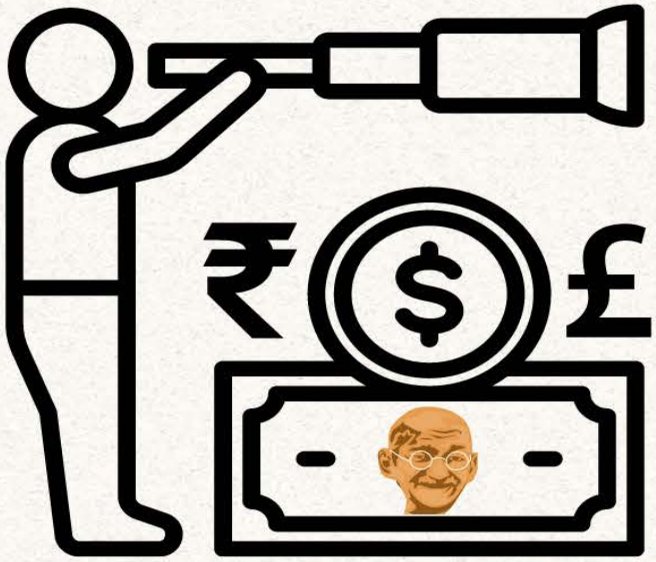
Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis* (2nd ed). Springer. [Amazon]

Wickramasuriya, S. L., Athanasopoulos, G., & Hyndman, R. J. (2019). Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. *Journal of the American Statistical Association*, *114*(526), 804–819. [DOI]

Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management Science*, *6*(3), 324–342. [DOI]

Young, P. C., Pedregal, D. J., & Tych, W. (1999). Dynamic harmonic regression. *Journal of Forecasting*, *18*, 369–394. [DOI]

FORECASTING WITH R LANGUAGE AND MICROSOFT EXCEL



★★★★★

"THE FUTURE DEPENDS UPON THE PRESENT"

DR. MAYUR DOSHI
AND
SUKETU SANGHVI