Forecasting Princples and Practice 2 (8-10)

Koji Mizumura

October 12, 2018(Beg) - (End)

Table of Contents

This document is html\_notebook output (not html\_document).

# Resources [Forecasting: princples and practice](https://otexts.org/fpp2/)  
[Slides](https://robjhyndman.com/seminars/uwa2017/)  
[Github](https://github.com/nealxun/ForecastingPrinciplePractices)

From the Github, we can install the package fpp2.

# 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 smooothing models are based on 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 **stationairity** and differencing time series.

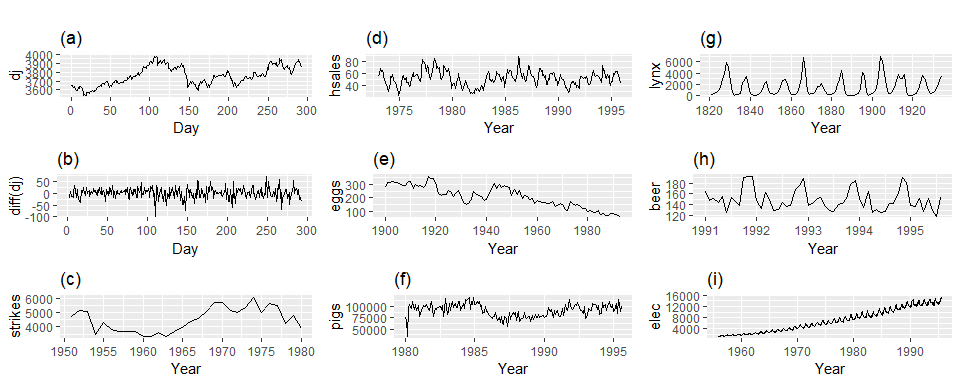
## Stationarity and differencing

A stationary time series is one whose properties do not depend on the time at which the series is observed.

More precisely, if is a *stationary* time series, then for all , the distribution of does not depend on . 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.



Which of these series are stationary? (a) Dow Jones index on 292 consecutive days; (b) Daily change in the Dow Jones index on 292 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 @ref(fig:stationary). Which of these do you think are stationary?

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

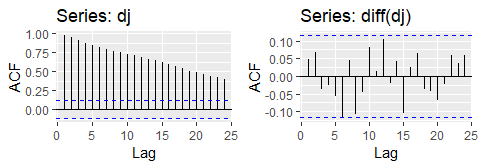
At first glance, the strong cycles in series (g) might appear to make it non-stationary. But these cycles are aperiodic — they are caused when the lynx population becomes too large for the available feed, so that they stop breeding and the population falls to very low numbers, then the regeneration of 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 @ref(fig:stationary), note that the Dow Jones index was non-stationary in panel (a), but the daily changes were stationary in panel (b). This shows one way to make a time series stationary — compute the differences between consecutive observations. This is known as **differencing**.

Transformations such as logarithms can help to stabilize the variance of a time series. Differencing can help stabilize 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 is often large and positive.



The ACF of the Dow-Jones index (left) and of the daily changes in the Dow-Jones index (right).

Box.test(diff(dj),lag=10, type="Ljung-Box")

##   
## Box-Ljung test  
##   
## data: diff(dj)  
## X-squared = 14.461, df = 10, p-value = 0.153

The ACF of the differenced Dow-Jones index looks just like that of a white noise series. There is only one autocorrelation lying just outside the 95% limits, and the Ljung-Box statistic has a *p*-value of 0.153 (for ). This suggests that the *daily change* in the Dow-Jones index 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

The differenced series will have only values, since it is not possible to calculate a difference for the first observation.

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

Random walk models are very 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.

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

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

This is the model behind the drift method discussed in Section @ref(sec-2-methods).

### 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:

In this case, will have 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

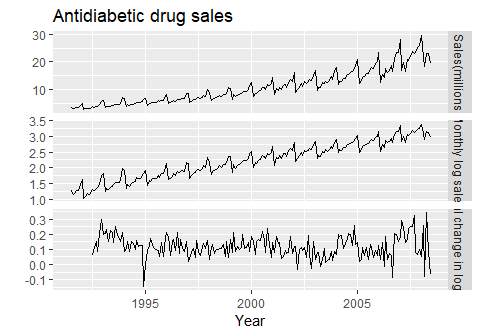
A seasonal difference is the difference between an observation and the corresponding observation from the previous year. So

where the number of seasons. These are also called “lag- differences”, as we subtract the observation after a lag of periods.

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

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

cbind("Sales(millions"=a10,  
 "Monthly log sales"=log(a10),  
 "Annual change in log sales"=diff(log(a10),12)) %>%   
 autoplot(facets=TRUE)+  
 labs(x="Year",y="")+  
 ggtitle("Antidiabetic drug sales")



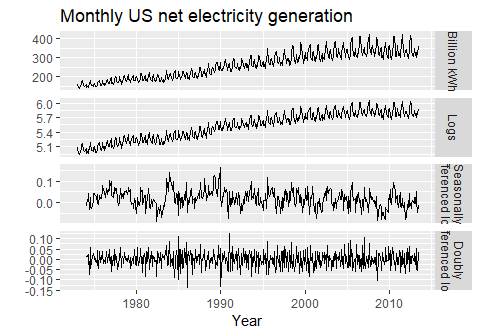
Logs and seasonal differences of the A10 (antidiabetic) sales data. The logarithms stabilize the variance, while the seasonal differences remove the sesonality and trend

Figure @ref(fig:a10diff) 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.

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 do both a **seasonal difference** and a **first difference** to obtain **stationary data**, as is shown in Figure @ref(fig:usmelec). Here, the data are first transformed using logarithms (second panel), then seasonal differences are calculated (third panel). The data still seem a little 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")



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 @ref(fig:a10diff) do not show substantially different behaviour from the seasonally differenced data in Figure @ref(fig:usmelec). 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 will be discussed later, but there are always some choices to be made in the modeling process, and different analysts may make different choices.

If denotes a seasonally differenced series, then the twice-differenced series is

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 different unit root tests are available, which are based on different assumptions and may lead to conflicting answers.

#### ADF test

One of the most popular tests is the *Augmented Dickey-Fuller (ADF) test*. For this test, the following regression model is estimated:

where denotes the first-differenced series, , and is the number of lags to include in the regression (often set to be about 3). If the original series, , needs differencing, then the coefficient should be approximately zero. If is already stationary, then . The usual hypothesis tests for regression coefficients do not work when the data are non-stationary, but the test can be carried out using the following R function from the tseries package.

library(tseries)  
adf.test(x, alternative="stationary")

In R, the default value of is set to , where is the length of the time series and means the largest integer not greater than .

The **null-hypothesis** for an ADF test is that the data are **non-stationary**. Thus, large p-values are indicative of non-stationarity, and small p-values suggest stationarity. Using the usual 5% threshold, differencing is required if the p-value is greater than 0.05.

#### KPSS test

Another popular unit root test is the *Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test*. This reverses the hypotheses, so the null hypothesis is that the data are stationary. In this case, small p-values (e.g., less than 0.05) suggest that differencing is required. The kpss.test is also from the tseries package.

library(tseries)  
kpss.test(x)

or ur.kpss function from the urca package.

library(urca)

## Warning: package 'urca' was built under R version 3.5.1

goog %>% ur.kpss() %>% summary()

##   
## #######################   
## # KPSS Unit Root Test #   
## #######################   
##   
## Test is of type: mu with 7 lags.   
##   
## Value of test-statistic is: 10.7223   
##   
## Critical value for a significance level of:   
## 10pct 5pct 2.5pct 1pct  
## critical values 0.347 0.463 0.574 0.739

he 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 , 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.

### Biblography

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. <https://doi.org/10.1016/0304-4076(92)90104-Y>

## Backshift notation

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

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

For monthly data, if we wish to consider “the same month last year,” the notation is = .

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

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

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

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

For example, a seasonal difference followed by a first difference can be written as

the same result we obtained earlier.

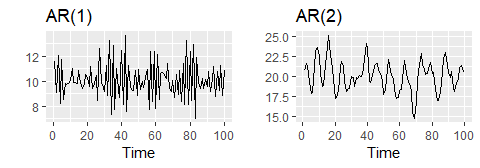
## Autoregressive models

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

Thus, an autoregressive model of order can be written as

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

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 and model. Changin the parameters results in different time series patterns. The variance of the error term will only change the scale of the series, not the patterns.



Two examples of data from autoregressive models with different parameters. Left: AR(1) with . Right: AR(2) with . In both cases, is normally distributed white noise with mean zero and variance one.

For an AR(1) moel: - when , is equivalent to white noise; - when and , is equivalent to a random walk; - when and , is equivalent to a random walk with drift; - when , tends to oscillate between positive and negative values;

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

* For an AR(1) model: .
* For an AR(2) model: , ,

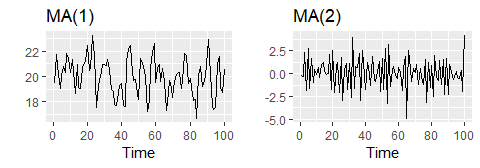
When , the restrictions are much more complicated. R takes care of these restrictions when estimating a model.

## 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.

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

Notice that each value of 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 @ref(ch-decomposition). A moving average model is used for forecasting future values, while moving average smoothing is used for estimating the trend-cycle of past values.



Two examples of data from moving average models with different parameters. Left: MA(1) with . Right: MA(2) with . In both cases, is normally distributed white noise with mean zero and variance one.

Figure @ref(fig:maq) shows some data from an MA(1) model and an MA(2) model. Changing the parameters results in different time series patterns. As with autoregressive models, the variance of the error term will only change the scale of the series, not the patterns.

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

Provided , the value of will get smaller as gets larger. So eventually we obtain

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() 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 mathematical properties that make them easier to use in practice.

The **invertibility** constraints are similar to the stationarity constraints.

More complicated conditions hold for . Again, R will take care of these constraints when estimating the models.

For example, consider the MA(1) process, . In its AR() 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 lag increases, so the more distant the observation the greater their influence on the current error. When , the weights are constant in size, and the distant observations have the same influence as the recent observations. As nether of these situations make much sense, we require , so most recent observations have higher weight han observations from the more distant past. Thus, the process is invertible when .

The invertibility constrains for other models are similar to the stationary constraints. - For an MA(1) model: () - For an MA(2) model: ()

More complicated conditions hold for q. Again, R will take care of these constrains when estimating the model.

## 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 model (in this context, “integration” is the reverse of differencing). The full model can be written as

where is the differenced series (it may have been differenced more than once). The “predictors” on the right hand side include both lagged values of and lagged errors. We call this an **ARIMA() model**, where

|  |  |
| --- | --- |
|  | order of the autoregressive part; |
|  | degree of first differencing involved; |
|  | order of the moving average part. |

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

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

|  |  |
| --- | --- |
| 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(,0,0) |
| Moving average | ARIMA(0,0,) |

Once we start combining components in this way to form more complicated models, it is much easier to work with the backshift notation. Then equation @ref(eq:8-arima) can be written as

R uses a slightly different parameterization:

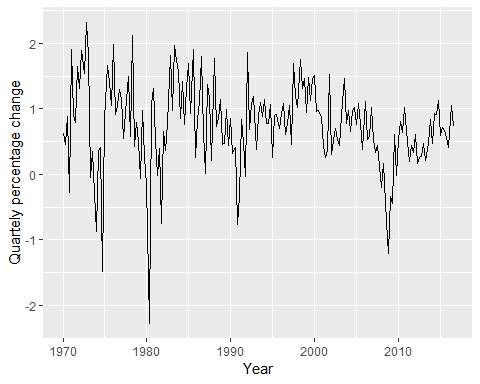
where and is the mean of . To convert to the form given by @ref(eq:arimaB), set .

Selecting appropriate values for , and can be difficult. However, the auto.arima() function in R will do it for you automatically. Later in this chapter, we will learn how the function works, and some methods for choosing these values yourself.

### US consumption expenditure

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

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



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

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

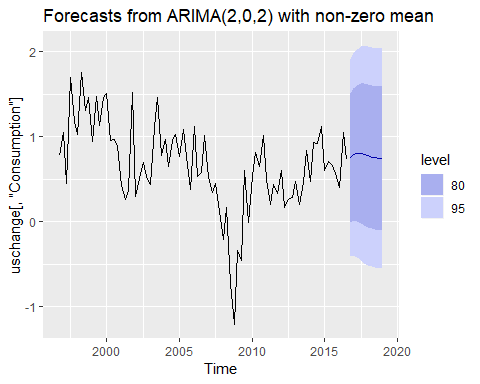
## Series: uschange[, "Consumption"]   
## ARIMA(2,0,2) with non-zero mean   
##   
## Coefficients:  
## ar1 ar2 ma1 ma2 mean  
## 1.3908 -0.5813 -1.1800 0.5584 0.7463  
## s.e. 0.2553 0.2078 0.2381 0.1403 0.0845  
##   
## sigma^2 estimated as 0.3511: log likelihood=-165.14  
## AIC=342.28 AICc=342.75 BIC=361.67

This is ARIMA(2,0,2) model:

, where c=0.746\*(1-1.391+0.581)=0.142 and is white noise with a standard deviation of .

Forecast from the model are shown in Figure 8.8.

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



### Understanding ARIMA models

The auto.arima() function is useful, but anything can be a little dangerous, and it is worth understanding something of the behaviour of the models even whem yo urely 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 and , the long-term forecasts will go to zero. - If and , the long-term forecasts will go to a non-zero constant. - If and , the long-term forecasts will follow a straight line. - If and , the long-term forecasts will go to the mean of the data. - If and , the long-term forecasts will follow a straight line. - If and , the long-term forecasts will follow a quadratic trend.

The value of also has an effect on the prediction intervals - the higher the vale of , the more rapidly the prediction intervals increase in size. For , tlhe long-term forecast standard deviation will got to the standard deviation of the historical data, so the prediction inteval will all be essentially the same.

This behaviour is seen in Figure @ref(fig:usconsumptionf) where and . 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 is important if the data show cycles. To obtain cyclic forecasts, it is necessary to have , along with some additional conditions on the parameters. For an AR(2) model, cyclic behaviour occurs if . In that case, the average period of the cycles is [[1]](#footnote-49)

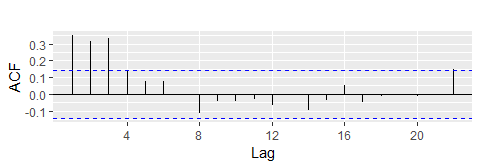
### ACF and PACF plots It is usually not possible to tell, simply from a time plot, what values of and are appropriate for the data. However, it is sometimes possible to use the ACF plot, and the closely related PACF plot, to determine appropriate values for and .

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

To overcome this problem, we can use **partial autocorrelations**. These measure the relationship between and after removing the effects of lags . 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, , the th partial autocorrelation coefficient, is equal to the estimate of in an AR() model. In practice, there are more efficient algorithms for computing than fitting all of these autoregressions, but they give the same results.

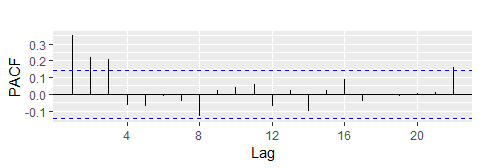
Figures @ref(fig:usconsumptionacf) and @ref(fig:usconsumptionpacf) shows the ACF and PACF plots for the US consumption data shown in Figure @ref(fig:usconsumption). The partial autocorrelations have the same critical values of as for ordinary autocorrelations, and these are typically shown on the plot as in Figure @ref(fig:usconsumptionacf).

ggAcf(uschange[,"Consumption"],main="")



ACF quarterly percentage change in US consumption. A convenient way to produce a time plot, ACF plot and PACE plot in one command is to use the ggtsdiplay function in R

ggPacf(uschange[,"Consumption"],main="")



PACF of quarterly percentage change in US consumption.

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

The data may follow an ARIMA(,, 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 in the PACF, but none beyond lag .

The data may follow an ARIMA(0,,) 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 in the ACF, but none beyond lag .

In Figure @ref(fig:usconsumptionacf), 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  
## 0.2274 0.1604 0.2027 0.7449  
## s.e. 0.0713 0.0723 0.0712 0.1029  
##   
## sigma^2 estimated as 0.3494: log likelihood=-165.17  
## AIC=340.34 AICc=340.67 BIC=356.5

This model is actually slightly better than the model identified by auto.arima() (with an AICc value of 350.67 compared with 342.75), 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  
## 0.2274 0.1604 0.2027 0.7449  
## s.e. 0.0713 0.0723 0.0712 0.1029  
##   
## sigma^2 estimated as 0.3494: log likelihood=-165.17  
## AIC=340.34 AICc=340.67 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(2,0,2) model, so we do not produce the plot here.

## Estimation and order selection

## Maximum likelihood estimation

Once the model order has been identified (i.e., the values of , and ), we need to estimate the parameters , , . When R estimates the ARIMA model, it uses *maximum likelihood estimation* (MLE). This technique finds the values of the parameters which maximize the probability of obtaining the data that we have observed. For ARIMA models, MLE is very similar to the *least squares* estimates that would be obtained by minimizing

For the regression models considered in Chapter @ref(ch-regression), 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 optimization 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 , and , R will try to maximize 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

where is the likelihood of the data, if and if . Note that the last term in parentheses is the number of parameters in the model (including , the variance of the residuals).

For ARIMA models, the corrected AIC can be written as

and the Bayesian Information Criterion can be written as

Good models are obtained by minimizing 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 selectingn the appropriate order of differencing () of a model, but only for selecting the values of and . 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 , and then we can use the AICc to select and .

## ARIMA modelling in R

### How does auto.arima work?

The auto.arima() function in R uses a variation of the Hyndman-Khandakar algorithm [@HK08], which combines unit root tests, minimization of the AICc and MLE to obtain an ARIMA model. The algorithm follows these steps.

\*\* Hyndman-Khandakar algorithm for automatic ARIMA modelling\*\*

1. The number of differences is determined using repeated KPSS tests.
2. The values of and are then chosen by minimizing the AICc after differencing the data times. Rather than considering every possible combination of and , the algorithm uses a stepwise search to traverse the model space.
   1. Four initial models are fitted:
      * ARIMA,
      * ARIMA,
      * ARIMA,
      * ARIMA. A constant is included unless . If , an additional model
      * ARIMA without a constant is also fitted.
   2. The best model (with the smallest AICc value) fitted in step (a) is set to be the “current model”.
   3. Variations on the current model are considered:
      * vary and/or from the current model by ;
      * include/exclude from the current model. The best model considered so far (either the current model or one of these variations) becomes the new current model.
   4. Repeat Step 2(c) until no lower AICc can be found.

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 argument to auto.arima() provide for many variations on the algorithm. What is described here is the default behavior.

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 argment 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 unless , 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.

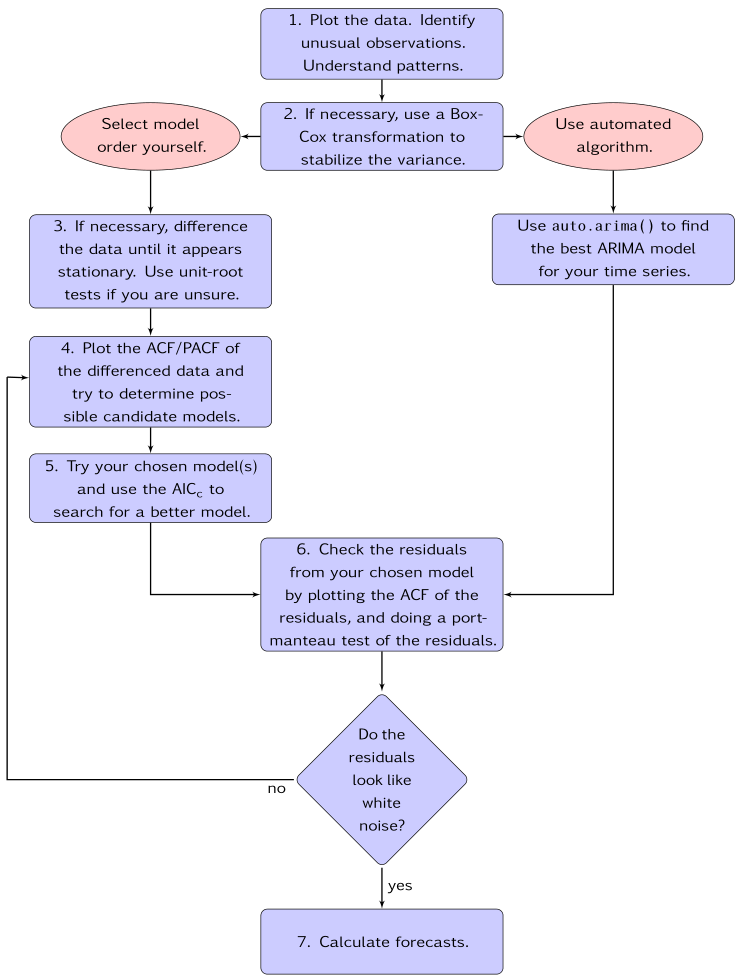
### Modeling procedure

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 usual observations
2. If necessary, transform the data (using the Box-Cox transformation) to stabilize 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() or ARIMA() 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 automated algorithm only takes care of step 3–5. So even if you use it, you will still need to take care of the other steps yourself.

The process is summarized as follows.

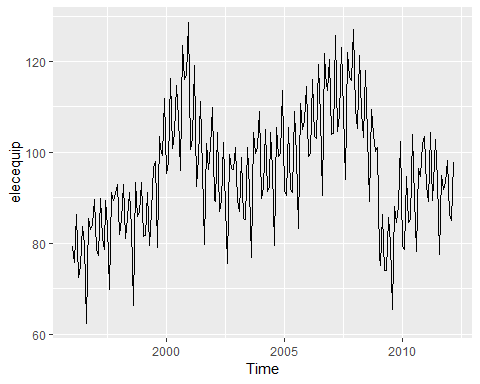


General process for forecasting using an ARIMA model.

### Example: Seasonal adjusted electrical equipment orders

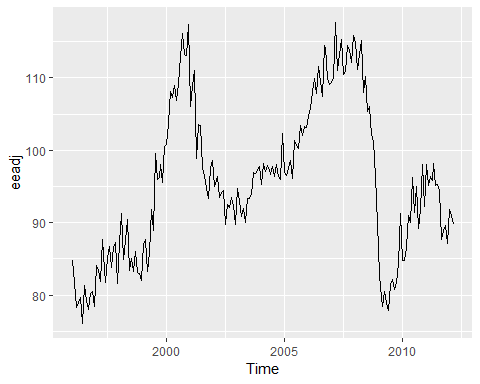
We will apply this procedure to the seasonally adjusted electrical equipment orders data shown in Figure @ref(fig:ee1).

autoplot(elecequip)



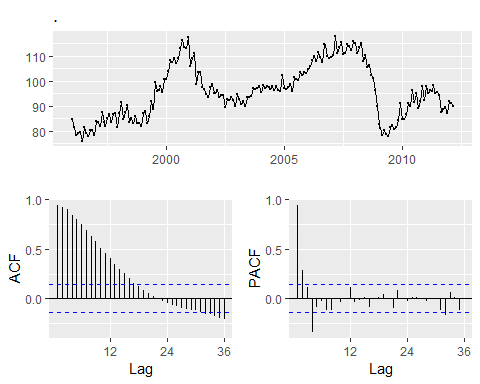
Seasonally adjusted electrical equipment orders index in the Euro area.

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



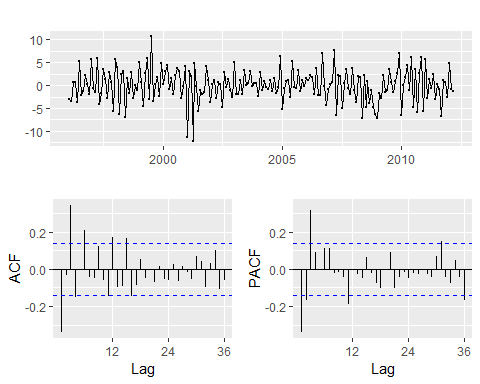
Seasonally adjusted electrical equipment orders index in the Euro area.

eeadj %>% ggtsdisplay()



Seasonally adjusted electrical equipment orders index in the Euro area.

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



Seasonally adjusted electrical equipment orders index in the Euro area.

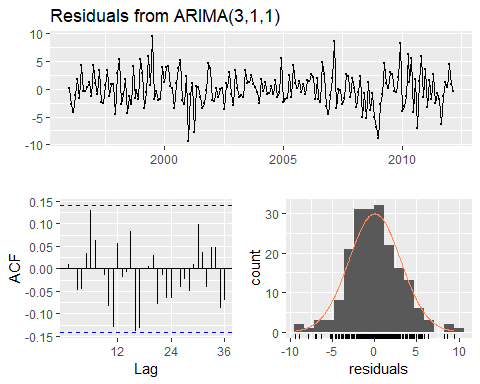
1. The time plot show some sudden changes, particularly the big drop in 2008/2009. These changes are due to the global economic environment. Otherwise, there is nothing usual about the time plot and there appear to be no need to do any data adjustments.
2. There is no evidence of **changing variance**, so we will not do a Box-Cox transformation.
3. The data are clearly non-stationary, as the sereies wanders up and fown for long periods. Consequently, we will take a first difference of the data. The differenced data are shown in Figure @ref(fig:ee2).These look stationary, and so we will not consider further differences.
4. The PACF shown in Figure @ref(fig:ee2) 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))  
summary(fit)

## Series: eeadj   
## ARIMA(3,1,1)   
##   
## Coefficients:  
## ar1 ar2 ar3 ma1  
## 0.0044 0.0916 0.3698 -0.3921  
## s.e. 0.2201 0.0984 0.0669 0.2426  
##   
## sigma^2 estimated as 9.577: log likelihood=-492.69  
## AIC=995.38 AICc=995.7 BIC=1011.72  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.03288179 3.054718 2.357169 -0.00647009 2.481603 0.28844  
## ACF1  
## Training set 0.008980578

1. The ACF plot of the residuals from the ARIMA(3.1,1) model shows that all correlations are within the threshold limits, indicating that the residuals are behaving like white noise. A portmanteu test returns a large p-value, also suggesting that the residuals are white noise.

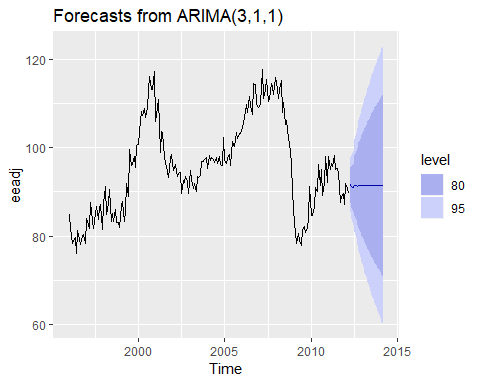
checkresiduals(fit)



##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(3,1,1)  
## Q\* = 24.034, df = 20, p-value = 0.2409  
##   
## Model df: 4. Total lags used: 24

7．The ACF plot of the residuals from the **ARIMA(3,1,1)** model shows that all correlations are within the threshold limits, indicating that the residuals are behaving like white noise. A port

autoplot(forecast(fit))



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

### Understanding constants in R

A non-seasonal ARIMA model can be written as

or equivalently as

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

By default, the Arima() function sets when and provides an estimate of when . 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 .

The argument include.mean only has an effect when and is TRUE by default. Setting include.mean=FALSE will force .

The argument include.drift allows when . For , no constant is allowed as a quadratic or higher order trend is particularly dangerous when forecasting. The parameter is called the “drift” in the R output when .

There is also an argument include.constant which, if TRUE, will set include.mean=TRUE if and include.drift=TRUE when . 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 or , a constant will be included if it improves the AICc value; for the constant is always omitted. If allowdrift=FALSE is specified, then the constant is only allowed when .

### Plotting the characteristic roots

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

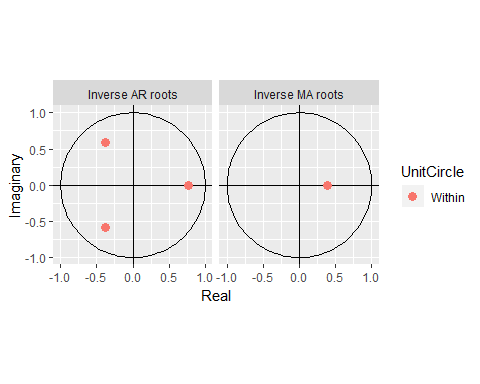
We can re-write equation @ref(eq:c) as

where is a th order polynomial in and is a th order polynomial in .

The stationarity conditions for the model are that the complex roots of lie outside the unit circle, and the invertibility conditions are that the complex roots of 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 the following plot.

autoplot(fit)



The three red dots in the left hand plot correspond to the roots of the polynomials , while the red dot in the right hand plot corresponds to the root of . 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.

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for R. Journal of Statistical Software, 27(1), 1–22. <https://doi.org/10.18637/jss.v027.i03>

## 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 is on the left hand side and all other terms are on the right
2. Rewrite the equation by replacing with
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 , these steps are repeated for until all forecasts have been caluculated. The procedure is most easily understood via an example. We will illustrate using the model fitted in the previous section. The model can be written as follows.

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:

where , , and . Then we expand the left hand side to obtain

and applying the backshift operator gives

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

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 with in @ref(eq:arima301f):

Assuming we have observations up to time , all values on the right hand side are known except for , which we replace with zero, and , which we replace with the last observed residual :

A forecast of is obtained by replacing with in @ref(eq:arima301f) . All values on the right hand side will be known at time except which we replace with , and and , both of which we replace with zero:

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 is the standard deviation of the residuals, then a 95% prediction interval is given by . This result is true for all ARIMA models regardless of their parameters and orders.

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

Then, the estimated forecast variance can be written as

and a 95% prediction interval is given by .

In Section @ref(sec:mamodels), we showed that an AR(1) model can be written as an MA() model. Using this equivalence, the above result for MA() 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(,,) model, are given in more advanced textbooks such as @BDbook16.

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 ) they will converge, so that prediction intervals for long horizons are all essentially the same. For , 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

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

## 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:

|  |  |  |
| --- | --- | --- |
| ARIMA |  |  |
|  |  |  |
|  | Non-seasonal part | Seasonal part of |
|  | of the model | of the model |

where 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 very 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) model (without a constant) is for quarterly data (), and can be written as

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, …).

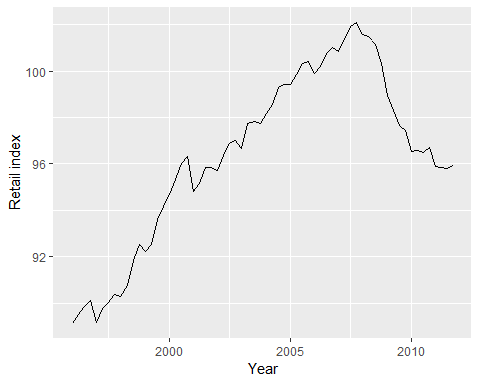
In considering the appropriate seasonal orders for a seasonal ARIMA model, restrict attention to the seasonal lags. and

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

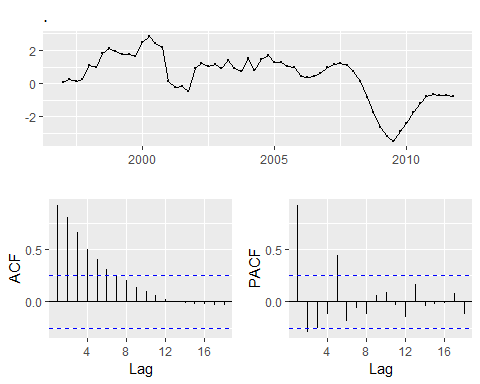
We will describe the seasonal ARIMA modelling procedure using quarterly European retail trade data from 1996 to 2011. The data are plotted in Figure 8.17.

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

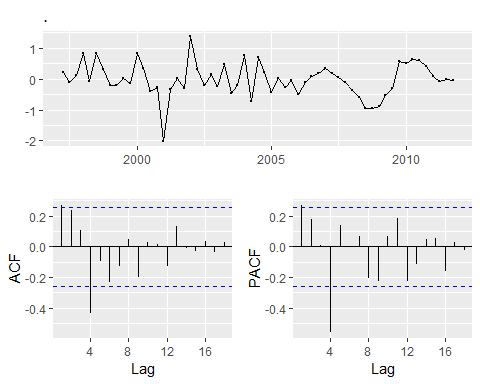


The data are clearly non-stationary, with some seasonality, so we will first take a seasonal difference. The seasonally differenced data are shown in Figure 8.18. These also appear to be non-stationary, so we take an additional first difference, shown in Figure 8.19.

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

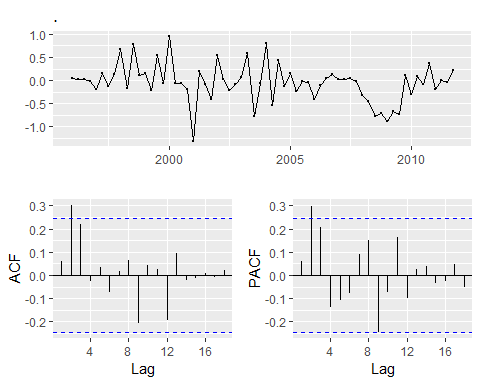


euretail %>% diff(lag=4) %>% diff() %>% ggtsdisplay()



Our aim now is to find an approproate 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 lag4 in the ACF suggests a seasonal Ma(1) component. Consequently, we begin with an ARIMA(0,1,1)(0,1,1) model, indicating a first and seasonal difference, and non-seasonal and seasonal MA(!) components. The residuals for the fitted model are shown in Figure 8.20 (By analogous logi applied to the PCAF, we could also have started with an model)

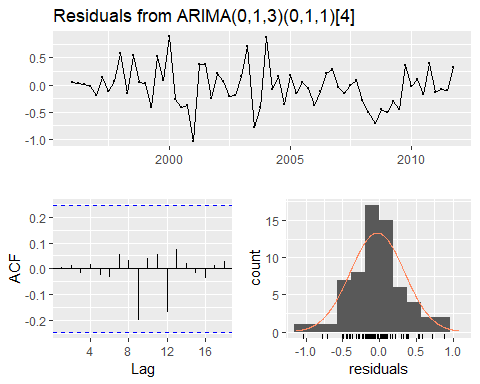
euretail %>%   
 Arima(order=c(0,1,1),seasonal=c(0,1,1)) %>%   
 residuals %>%   
 ggtsdisplay



Residuals from the fitted ARIMA(0,1,3)(0,1,1)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) model is 74.36, while that for the ARIMA(0,1,3)(0,1,1) 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) model. Its residuals are plotted in Figure @ref(fig:euretail5). 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)



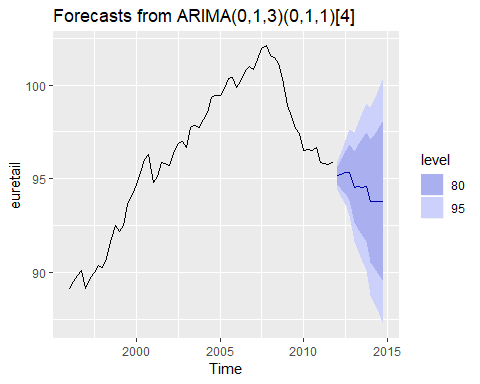
Residuals from the fitted ARIMA(0,1,3)(0,1,1) model for the European retail trade index data.

##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,3)(0,1,1)[4]  
## Q\* = 0.51128, df = 4, p-value = 0.9724  
##   
## Model df: 4. Total lags used: 8

Thus, we now have aseasonal 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 @ref(fig:euretail6).

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()



Forecasts of the European retail trade index data using the ARIMA(0,1,3)(0,1,1) 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 following result.

auto.arima(euretail)

## Series: euretail   
## ARIMA(1,1,2)(0,1,1)[4]   
##   
## Coefficients:  
## ar1 ma1 ma2 sma1  
## 0.7362 -0.4663 0.2163 -0.8433  
## s.e. 0.2243 0.1990 0.2101 0.1876  
##   
## sigma^2 estimated as 0.1587: log likelihood=-29.62  
## AIC=69.24 AICc=70.38 BIC=79.63

Notice that it has selected a different model (with a larger AICc value). auto.arima() takes some short-cuts in order to speed up the computation, and will not always give the best model. The short-cuts can be turned off, and then it will sometimes return a different model.

auto.arima(euretail, stepwise = FALSE, approximation = FALSE)

## Series: euretail   
## ARIMA(0,1,3)(0,1,1)[4]   
##   
## Coefficients:  
## ma1 ma2 ma3 sma1  
## 0.2630 0.3694 0.4200 -0.6636  
## s.e. 0.1237 0.1255 0.1294 0.1545  
##   
## sigma^2 estimated as 0.156: log likelihood=-28.63  
## AIC=67.26 AICc=68.39 BIC=77.65

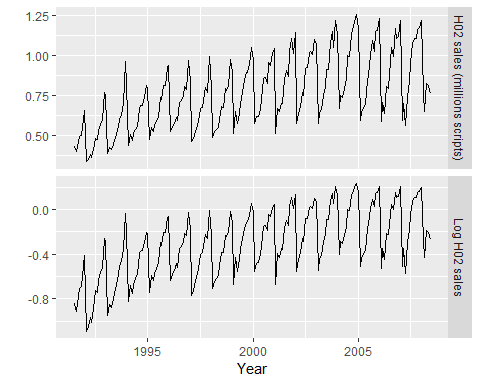
This time, it returned the same model we had identified.

The auto.arima() function uses nsdiffs() to determine (the number of seasonal differences to use) and ndiffs() (the number of ordinary differences to use). The selection of the other model parameters () 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 corticorsteriod sales in Australia. These are known as H02drugs under the Anatomical Therapeutic Chemical Classification scheme.

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



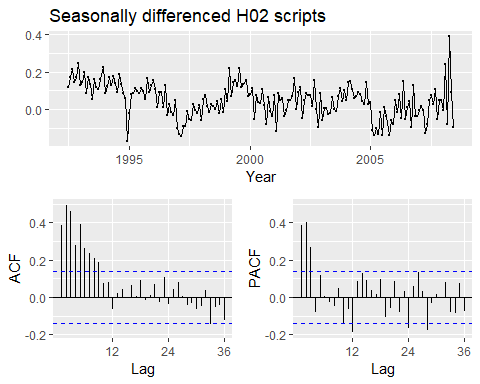
Cortecosteroid drug sales in Australia (in millions of scripts per month). Logged data shown in bottom panel.

Data from July 1991 to June 2009 are plotted in Figure @ref(fig:h02). There is a small increase in the variance with the level, so we take logarithms to stabilize 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 @ref(fig:h02b). 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")



Seasonall differenced cortecosteroid 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). 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) | -485.48 |
| ARIMA(3,0,1)(1,1,1) | -484.25 |
| ARIMA(3,0,1)(0,1,1) | -483.67 |
| ARIMA(3,0,1)(2,1,0) | -476.31 |
| ARIMA(3,0,0)(2,1,0) | -475.12 |
| ARIMA(3,0,2)(2,1,0) | -474.88 |
| ARIMA(3,0,1)(1,1,0) | -463.40 |

Of these models, the best is the ARIMA(3,0,1)(0,1,2) 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  
## -0.1603 0.5481 0.5678 0.3827 -0.5222 -0.1768  
## s.e. 0.1636 0.0878 0.0942 0.1895 0.0861 0.0872  
##   
## sigma^2 estimated as 0.004278: log likelihood=250.04  
## AIC=-486.08 AICc=-485.48 BIC=-463.28

The residuals from this model are shown in Figure @ref(fig:h02res). 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(2,1,3)(0,1,1) model. However, the model still fails the Ljung-Box test. 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 the following table.

|  |  |
| --- | --- |
| Model | RMSE |
| ARIMA(3,0,1)(0,1,2) | 0.0622 |
| ARIMA(3,0,1)(1,1,1) | 0.0630 |
| ARIMA(2,1,4)(0,1,1) | 0.0632 |
| ARIMA(2,1,3)(0,1,1) | 0.0634 |
| ARIMA(3,0,3)(0,1,1) | 0.0640 |
| ARIMA(2,1,5)(0,1,1) | 0.0640 |
| ARIMA(3,0,1)(0,1,1) | 0.0644 |
| ARIMA(3,0,2)(0,1,1) | 0.0644 |
| ARIMA(3,0,2)(2,1,0) | 0.0645 |
| ARIMA(3,0,1)(2,1,0) | 0.0646 |
| ARIMA(4,0,2)(0,1,1) | 0.0648 |
| ARIMA(4,0,3)(0,1,1) | 0.0648 |
| ARIMA(3,0,0)(2,1,0) | 0.0661 |
| ARIMA(3,0,1)(1,1,0) | 0.0679 |

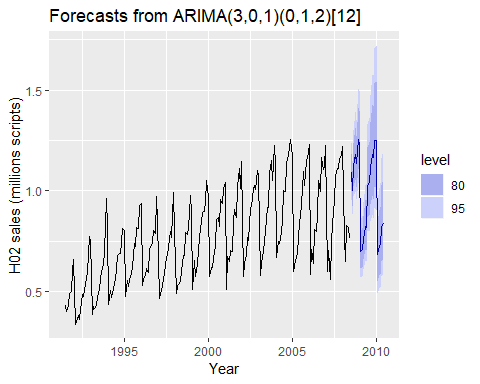
The models chosen manually and with auto.arima() are both in the top four models based on their RMSE values.

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 is 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 RIMA(3,0,1)(0,1,2) 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 @ref(fig:h02f).

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



Forecasts from the ARIMA(3,0,1)(0,1,2) model applied to the H02 monthly script sales data.

## ARIMA vs ETS

It is 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 couterparts.

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 following table gives the equivalence relationships for the two classes of models.

|  |  |  |
| --- | --- | --- |
| **ETS model** | **ARIMA model** | **Parameters** |
| ETS(A,N,N) | ARIMA(0,1,1) |  |
| ETS(A,A,N) | ARIMA(0,2,2) |  |
|  |  |  |
| ETS(A,A,N) | ARIMA(1,1,2) |  |
|  |  |  |
|  |  |  |
| ETS(A,N,A) | ARIMA(0,0,)(0,1,0) |  |
| ETS(A,A,A) | ARIMA(0,1,)(0,1,0) |  |
| ETS(A,A,A) | ARIMA(1,0,)(0,1,0) |  |

For the seasonal models, the ARIMA parameters have a large number of restrictions.

The AICc is useful for selecting between models in the same class. For example, we can use it to select an ARIMA model between candiate ARIMA models[[2]](#footnote-89) 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 @ref(sec-7-trendmethods) where, air <- window(ausair, start=1990).

head(air)

## Time Series:  
## Start = 1990   
## End = 1995   
## Frequency = 1   
## [1] 17.5534 21.8601 23.8866 26.9293 26.8885 28.8314

# 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] NaN

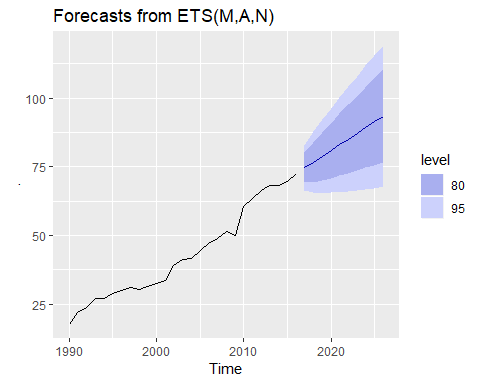
#> [1] 7.864  
mean(e2^2, na.rm=TRUE)

## [1] 9.622164

#> [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()



### 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.

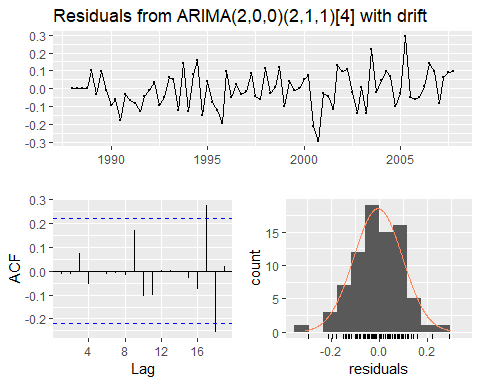
# Consider the qcement data beginning in 1988  
cement <- window(qcement, start=1988)  
  
# USe 20 years of the data as the training set  
train <- window(cement, end=c(2007,4))

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 an ARIMa model to the training data  
(fit.arima <- auto.arima(train))

## Series: train   
## ARIMA(2,0,0)(2,1,1)[4] with drift   
##   
## Coefficients:  
## ar1 ar2 sar1 sar2 sma1 drift  
## 0.6470 0.1934 0.0726 -0.2403 -0.8699 0.0105  
## s.e. 0.1165 0.1183 0.1602 0.1387 0.1624 0.0030  
##   
## sigma^2 estimated as 0.01156: log likelihood=61.45  
## AIC=-108.89 AICc=-107.25 BIC=-92.58

checkresiduals(fit.arima)



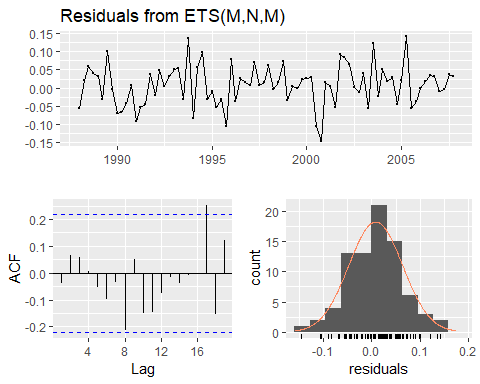
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(2,0,0)(2,1,1)[4] with drift  
## Q\* = 3.5497, df = 3, p-value = 0.3144  
##   
## Model df: 6. Total lags used: 9

The output below also shows that the ETS model selected and estimated by ets. This models also does well in capturing all the dynamics in the data as the residuals also seem to be white noise.

# Fit an ETS model to the training data  
(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 1.0439 1.0103 0.9148  
##   
## sigma: 0.0581  
##   
## AIC AICc BIC   
## -2.1967020 -0.6411464 14.4774845

checkresiduals(fit.ets)



##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,M)  
## Q\* = 6.3457, df = 3, p-value = 0.09595  
##   
## Model df: 6. Total lags used: 9

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 slighlty more accuarate model based on the test set RMSE, MAPE and MASE.

# Generate forecasts and compare accuracy over the test set  
fit.arima %>% forecast(h = 4\*(2013-2007)+1) %>% accuracy(qcement)

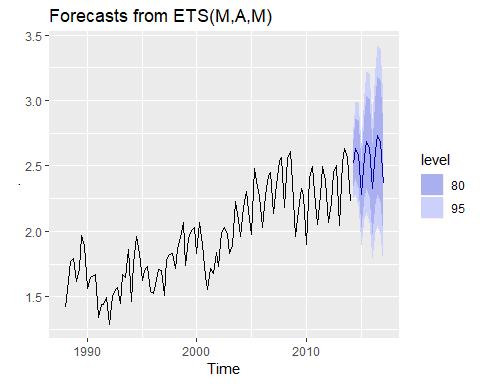
## ME RMSE MAE MPE MAPE  
## Training set -0.005847264 0.1005547 0.07954692 -0.6550065 4.347836  
## Test set -0.158355693 0.1978197 0.16749405 -7.3004639 7.653446  
## MASE ACF1 Theil's U  
## Training set 0.5434705 -0.01315073 NA  
## Test set 1.1443319 0.28848376 0.7226779

fit.ets %>% forecast(h = 4\*(2013-2007)+1) %>% accuracy(qcement)

## ME RMSE MAE MPE MAPE  
## Training set 0.01406512 0.1022079 0.07958478 0.4938163 4.371823  
## Test set -0.13495515 0.1838791 0.15395141 -6.2508975 6.986077  
## MASE ACF1 Theil's U  
## Training set 0.5437292 -0.03346295 NA  
## Test set 1.0518075 0.53438371 0.680556

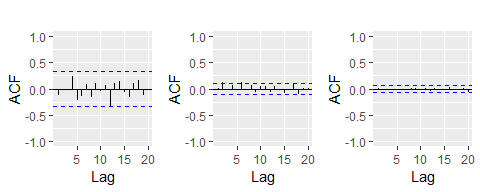
Below we generate and plot forecasts from an ets model for the next 3 years.

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



## Exercises

1. Figure @ref(fig:wnacfplus) shows the ACFs for 36 random numbers, 360 random numbers and 1,000 random numbers.
   1. Explain the differences among these figures. Do they all indicate that the data are white noise?



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.

1. 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.
4. usnetelec
5. usgdp
6. mcopper
7. enplanements
8. visitors
9. For the enplanements data, write down the differences you chose above using backshift operator notation.
10. For your retail data (from Exercise 3 in Section @ref(ex-graphics)), find the appropriate order of differencing (after transformation if necessary) to obtain stationary data.
11. Use R to simulate and plot some data from simple ARIMA models.
    1. Use the following R code to generate data from an AR(1) model with and . The process starts with .

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

1. Produce a time plot for the series. How does the plot change as you change ?
2. Write your own code to generate data from an MA(1) model with and .
3. Produce a time plot for the series. How does the plot change as you change ?
4. Generate data from an ARMA(1,1) model with , and .
5. Generate data from an AR(2) model with , and . (Note that these parameters will give a non-stationary series.)
6. Graph the latter two series and compare them.
7. Consider the number of women murdered each year (per 100,000 standard population) in the United States. (Data set wmurders).
8. By studying appropriate graphs of the series in R, find an appropriate ARIMA() model for these data.
9. Should you include a constant in the model? Explain.
10. Write this model in terms of the backshift operator.
11. Fit the model using R and examine the residuals. Is the model satisfactory?
12. Forecast three times ahead. Check your forecasts by hand to make sure that you know how they have been calculated.
13. Create a plot of the series with forecasts and prediction intervals for the next three periods shown.
14. Does auto.arima give the same model you have chosen? If not, which model do you think is better?
15. Consider the total international visitors to Australia (in millions) for the period 1980-2015. (Data set austa.)
16. 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.
17. 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.
18. Plot forecasts from an ARIMA(2,1,3) model with drift. Remove the constant and see what happens.
19. Plot forecasts from an ARIMA(0,0,1) model with a constant. Remove the MA term and plot again.
20. Plot forecasts from an ARIMA(0,2,1) model with no constant.
21. For the usgdp series:
22. if necessary, find a suitable Box-Cox transformation for the data;
23. fit a suitable ARIMA model to the transformed data using auto.arima();
24. try some other plausible models by experimenting with the orders chosen;
25. choose what you think is the best model and check the residual diagnostics;
26. produce forecasts of your fitted model. Do the forecasts look reasonable?
27. compare the results with what you would obtain using ets() (with no transformation).
28. Consider austourists, the quarterly number of international tourists to Australia for the period 1999–2010. (Data set austourists.)
29. Describe the time plot.
30. What can you learn from the ACF graph?
31. What can you learn from the PACF graph?
32. Produce plots of the seasonally differenced data . What model do these graphs suggest?
33. Does auto.arima give the same model that you chose? If not, which model do you think is better?
34. Write the model in terms of the backshift operator, then without using the backshift operator.
35. Consider the total net generation of electricity (in billion kilowatt hours) by the U.S. electric industry (monthly for the period January 1973 – June 2013). (Data set usmelec.) In general there are two peaks per year: in mid-summer and mid-winter.
36. Examine the 12-month moving average of this series to see what kind of trend is involved.
37. Do the data need transforming? If so, find a suitable transformation.
38. Are the data stationary? If not, find an appropriate differencing which yields stationary data.
39. 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?
40. 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.
41. Forecast the next 15 years of electricity generation by the U.S. electric industry. Get the latest figures from <https://goo.gl/WZIItv> to check the accuracy of your forecasts.
42. How many years of forecasts do you think are sufficiently accurate to be usable?
43. For the mcopper data:
44. if necessary, find a suitable Box-Cox transformation for the data;
45. fit a suitable ARIMA model to the transformed data using auto.arima();
46. try some other plausible models by experimenting with the orders chosen;
47. choose what you think is the best model and check the residual diagnostics;
48. produce forecasts of your fitted model. Do the forecasts look reasonable?
49. compare the results with what you would obtain using ets() (with no transformation).
50. Choose one of the following seasonal time series: hsales, auscafe, qauselec, qcement, qgas.
51. Do the data need transforming? If so, find a suitable transformation.
52. Are the data stationary? If not, find an appropriate differencing which yields stationary data.
53. 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?
54. 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.
55. Forecast the next 24 months of data using your preferred model.
56. Compare the forecasts obtained using ets().
57. 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?
58. For your retail time series (Exercise 5 above):
59. develop an appropriate seasonal ARIMA model;
60. compare the forecasts with those you obtained in earlier chapters;
61. Obtain up-to-date retail data from the [ABS website](https://goo.gl/twfCyD) (Cat 8501.0, Table 11), and compare your forecasts with the actual numbers. How good were the forecasts from the various models?
62. Produce a time plot of the sheep population of England and Wales from 1867–1939 (data set sheep).
63. Assume you decide to fit the following model:

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

1. By examining the ACF and PACF of the differenced data, explain why this model is appropriate.
2. 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 , , and . Without using the forecast function, calculate forecasts for the next three years (1940–1942).

1. Now fit the model in R and obtain the forecasts using forecast. How are they different from yours? Why?
2. Plot the annual bituminous coal production in the United States from 1920 to 1968 (data set ).
3. You decide to fit the following model to the series:
4. where is the coal production in year and is a white noise series. What sort of ARIMA model is this (i.e., what are , , and )?
   1. Explain why this model was chosen using the ACF and PACF.
   2. 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 , , , , and . Without using the forecast function, calculate forecasts for the next three years (1969–1971).

1. Now fit the model in R and obtain the forecasts from the same model. How are they different from yours? Why?
   1. Install the **rdatamarket** package in R using
   * install.packages("rdatamarket")
   1. Select a time series from <http://datamarket.com/data/list/?q=pricing:free>. Then copy its short URL and import the data using
   * x <- ts(rdatamarket::dmseries("shorturl")[,1], start=??, frequency=??)
   * (Replace ?? with the appropriate values.)
   1. Plot graphs of the data, and try to identify an appropriate ARIMA model.
   2. Do residual diagnostic checking of your ARIMA model. Are the residuals white noise?
   3. Use your chosen ARIMA model to forecast the next four years.
   4. Now try to identify an appropriate ETS model.
   5. Do residual diagnostic checking of your ETS model. Are the residuals white noise?
   6. Use your chosen ETS model to forecast the next four years.
   7. Which of the two models do you prefer?

## Further reading

The classic text which popularized ARIMA modelling was @BJ70. The most recent edition is @BJRL15, and it is still an excellent reference for all things ARIMA. @BDbook16 provides a good introduction to the mathematical background to the models, while @PTT01 describes some alternative automatic algorithms to the one used by auto.arima().

### Bibliograhy

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]

# 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 consideredregression models of the form:

where is a linear function of the predictor variables (), and 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 was 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 with in the equation. The error series is assumed to follow an ARIMA model. For example, if follows an ARIMA(1,1,1) model, we can write

where is a white noise series.

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

## Estimation

When we estimate parameters from the model, we need to minimise the sum of squared values. If we minimize the sum of squared 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 are no glonger 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 -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”.

Minimizing the sum of squared values avoids these problems. Alternatively, **maximum likelihood estimation** can be used; this will give very 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 and all of the predictors 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 and the predictors that is stationary, then the estimated coefficients will be consistent.[[3]](#footnote-105)

We therefore first difference the non-stationary variables in the model. It is often desirable to maintain the form of the relationship between 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

where , and , which is a regression model in differences with ARMA errors.

## Regression with ARIMA errors in R

The R function Arima() willfit 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.

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

will fit the model , where is an AR(1) error. This is equivalent to the model

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

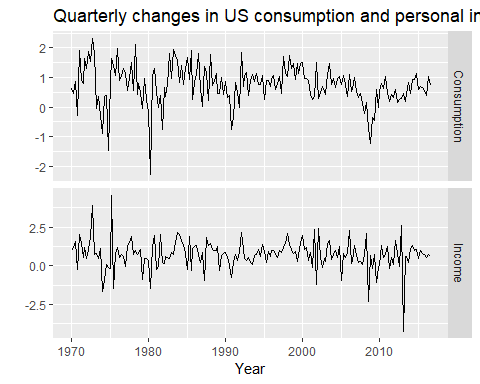
The auto.arima() function will also handle regression terms via the xreg argument. The user must specify the predictor variables to include, but auto.arima will select the best ARIMA model for the errors.

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 loweest AICc value selected.

### Example: US Personal Consumption and Income

Figure @ref(fig:usconsump) shows the quarterly changes in personal consumption expenditure and personal disposable income from 1970 to 2010. 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")



Percentage changes in quarterly personal consumption expenditure and personal disposable income for the USA, 1970 to 2010.

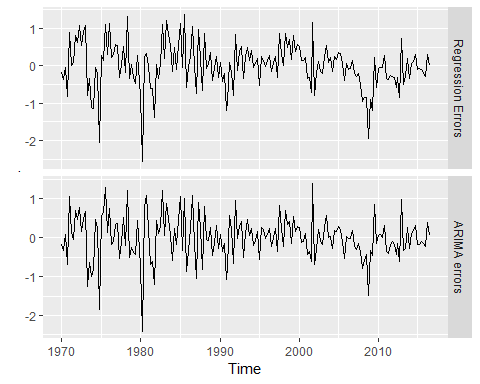
(fit <- auto.arima(uschange[,"Consumption"], xreg=uschange[,"Income"]))

## Series: uschange[, "Consumption"]   
## Regression with ARIMA(1,0,2) errors   
##   
## Coefficients:  
## ar1 ma1 ma2 intercept xreg  
## 0.6922 -0.5758 0.1984 0.5990 0.2028  
## s.e. 0.1159 0.1301 0.0756 0.0884 0.0461  
##   
## sigma^2 estimated as 0.3219: log likelihood=-156.95  
## AIC=325.91 AICc=326.37 BIC=345.29

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

We can recover both the and series using the residuals() function.

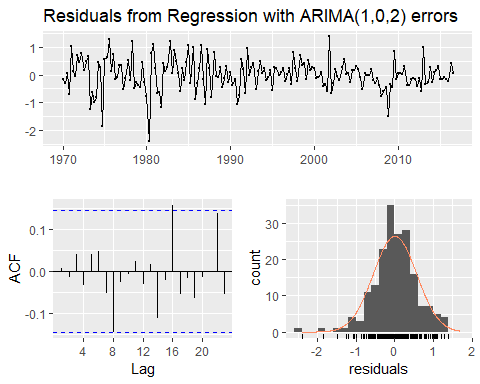
cbind("Regression Errors"=residuals(fit, type="regression"),  
 "ARIMA errors"=residuals(fit, type="innovation")) %>%   
 autoplot(facets=TRUE)



Regression errors () and ARIMA errors () from the fitted model.

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

checkresiduals(fit)



The residuals(ie the ARIMA errors) are not significantly different from white noise.

##   
## Ljung-Box test  
##   
## data: Residuals from Regression with ARIMA(1,0,2) errors  
## Q\* = 5.8916, df = 3, p-value = 0.117  
##   
## Model df: 5. Total lags used: 8

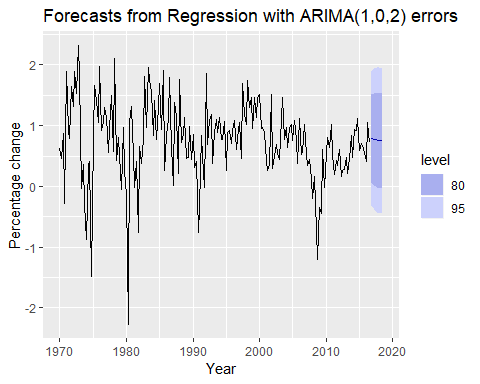
## Forecasting

To forecast 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 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")



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.

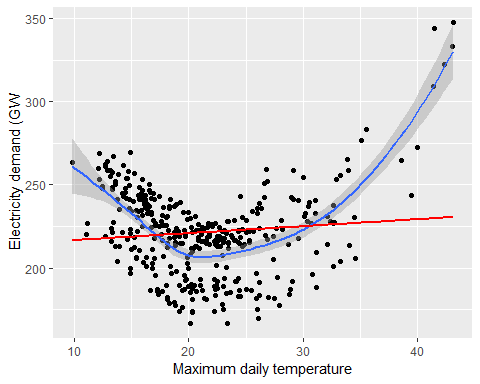
The prediction intervals for this model are narrower than those for the model developed in Section @ref(sec:nonseasonalarima) because we are now able to explain some of the variation in the data using the income predictor.

It is important to realize that the prediction intervals from regression models (with and 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

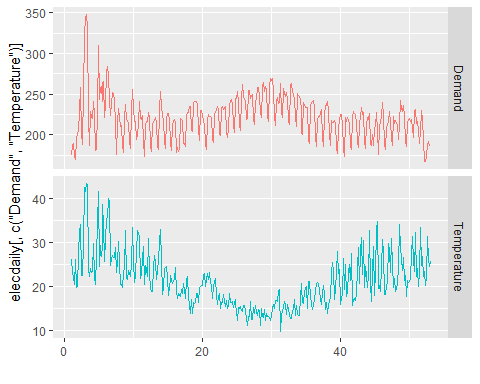
Daily electricity demand can be modelled as a function of temperature. As can be observed on an electricity bill, more electricity is used on cold days due to heating and hot days due to air conditioning. The higher demand on cold and hot days is reflected in the u-shape of Figure @ref(fig:elecscatter),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

## `geom\_smooth()` using method = 'loess' and formula 'y ~ x'



Daily electricity demand versus maximum daily temparature 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 @ref(fig:electime) show the time series of both daily demand and daily maximum temperatures. The plots highlight the need for both a non-linear and also dynamic model.



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. 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 @ref(Regr-ForeWithRegr))) where we set the temperature for the next 14 days to a constant 26 degrees.

xreg <- bind\_cols(MaxTemp=elecdaily[,"Temperature"],  
 MaxTempSq=elecdaily[,"Temperature"]^2,  
 Workday=elecdaily[,"WorkDay"])  
  
xreg %>% head(5)

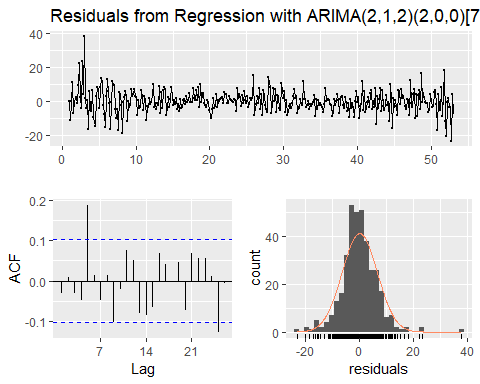
## # A tibble: 5 x 3  
## MaxTemp MaxTempSq Workday  
## <dbl> <dbl> <dbl>  
## 1 26 676 0  
## 2 23 529 1  
## 3 22.2 493. 1  
## 4 20.3 412. 0  
## 5 26.1 681. 0

fit <- auto.arima(elecdaily[,"Demand"], xreg=xreg)

summary(fit)

## Series: elecdaily[, "Demand"]   
## Regression with ARIMA(2,1,2)(2,0,0)[7] errors   
##   
## Coefficients:  
## ar1 ar2 ma1 ma2 sar1 sar2 drift MaxTemp  
## -0.0622 0.6731 -0.0234 -0.9301 0.2012 0.4021 -0.0191 -7.4996  
## s.e. 0.0714 0.0667 0.0413 0.0390 0.0533 0.0567 0.1091 0.4409  
## MaxTempSq Workday  
## 0.1789 30.5695  
## s.e. 0.0084 1.2891  
##   
## sigma^2 estimated as 43.72: log likelihood=-1200.7  
## AIC=2423.4 AICc=2424.15 BIC=2466.27  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.05300191 6.511663 4.716611 -0.07616974 2.137864 0.3238475  
## ACF1  
## Training set -0.03189628

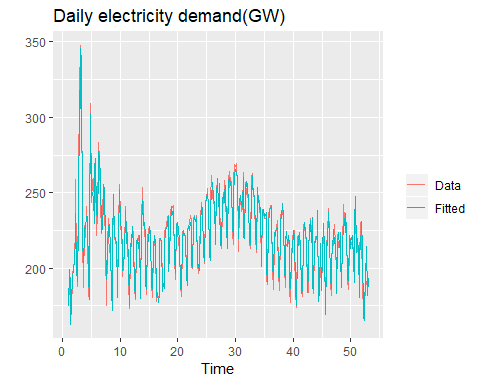
checkresiduals(fit)



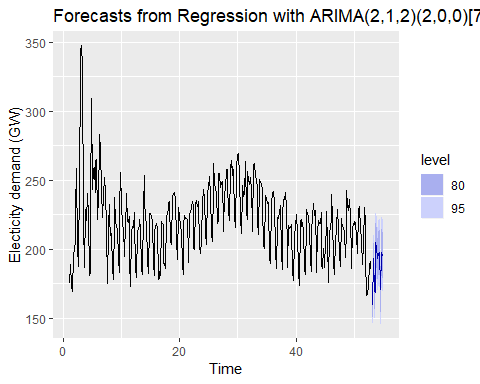
##   
## Ljung-Box test  
##   
## data: Residuals from Regression with ARIMA(2,1,2)(2,0,0)[7] errors  
## Q\* = 28.229, df = 4, p-value = 1.121e-05  
##   
## Model df: 10. Total lags used: 14

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.

autoplot(elecdaily[,"Demand"],series="Data")+  
 autolayer(fitted(fit), series="Fitted")+  
 ylab("")+  
 ggtitle("Daily electricity demand(GW)")+  
 guides(colour=guide\_legend(title=""))



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



The point forecasts look reasonable for the first two weeks of 2015. The slow down in electricity demand at the end of 2014 has caused the forecasts for the next two weeks to show similarly low demand values.

## Stochastic and deterministic trends

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

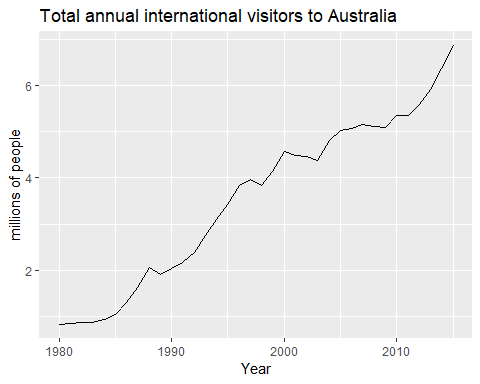
where where is an ARIMA process with . In that case, we can difference both sides so that , where is an ARMA process. In other words,

This is very similar to a random walk with drift, 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 ), 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")



Annual international visitors to Australia, 1980–2010.

Figure @ref(fig:austa) shows the total number of international visitors to Australia each year from 1980 to 2010. 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  
## 1.1127 -0.3805 0.4156 0.1710  
## s.e. 0.1600 0.1585 0.1897 0.0088  
##   
## sigma^2 estimated as 0.02979: log likelihood=13.6  
## AIC=-17.2 AICc=-15.2 BIC=-9.28

This model can be written as

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

Alternatively, the stochastic trend model can be estimated.

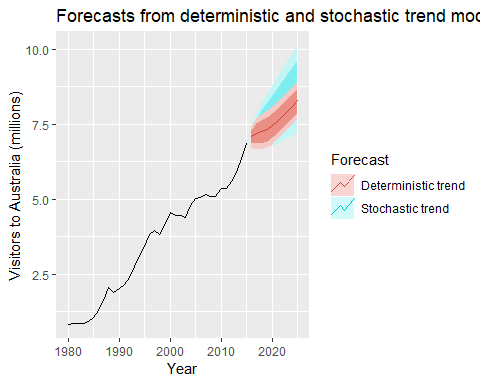
(fit2 <- auto.arima(austa,d=1))

## Series: austa   
## ARIMA(0,1,1) with drift   
##   
## Coefficients:  
## ma1 drift  
## 0.3006 0.1735  
## s.e. 0.1647 0.0390  
##   
## sigma^2 estimated as 0.03376: log likelihood=10.62  
## AIC=-15.24 AICc=-14.46 BIC=-10.57

This model can be written as , or equivalently

In this case, the estimated growth in visitor numbers is also 0.17 millions 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=data.frame(trend=length(austa)+1:10))  
fc2 <- forecast(fit2, h=10)  
autoplot(austa) +  
 forecast::autolayer(fc2, series="Stochastic trend") +  
 forecast::autolayer(fc1, series="Deterministic trend") +  
 ggtitle("Forecasts from deterministic and stochastic trend models") +  
 xlab("Year") + ylab("Visitors to Australia (millions)") +  
 guides(colour=guide\_legend(title="Forecast"))



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.

## 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.

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 parameters to be estimated for the initial seasonal 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.

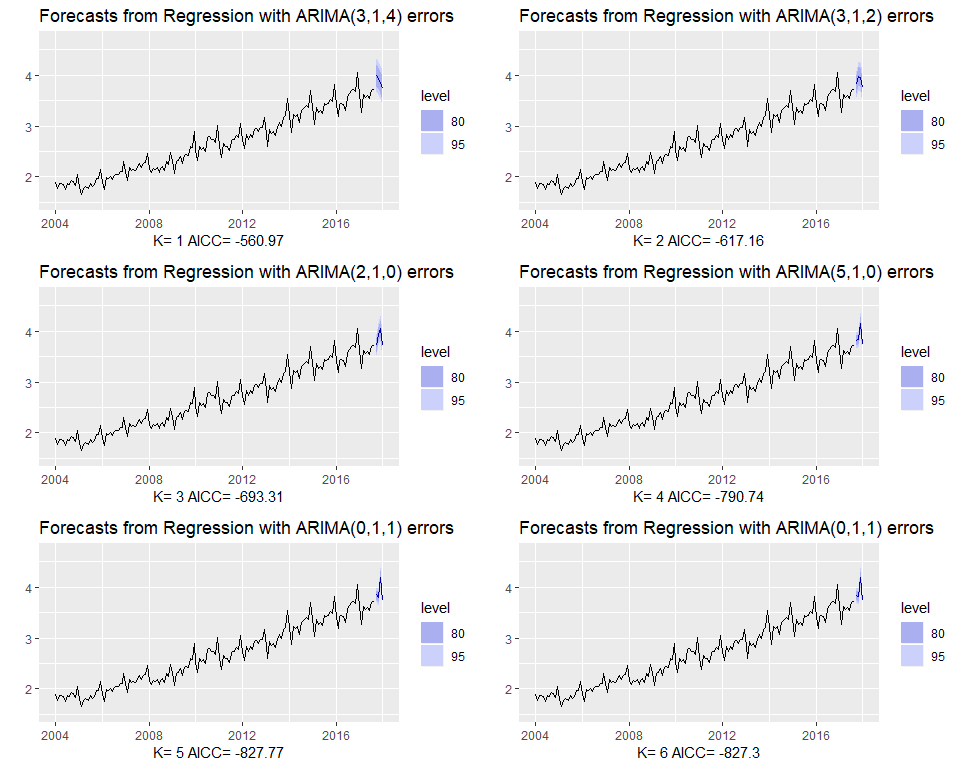
The advantages of this approach are: - it allows any length seasonality; - for data with more than one seasonal period, you can include Fourier terms of different frequencies - the seasonal pattern is smooth for small values of (but more wiggly seasonality can be handled by increasing . - the short term dynamics are easily handled with a simple ARMA error.

The only reall disadvanges (compared to a seasonal ARIMA model) is that the seasonality is assumed to be fixed — the pattern is not alloed to change over time. But, in practice, seasonality is usually remarkably constant so this is not a big disadvantage except for very 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. 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 the number of Fourier terms from 1 to 6 (which is equivalent to including seasonal dummies). Figure @ref(fig:eatout) shows the seasonal pattern projected forward as increases. Notice that as 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 maximised for , with a significant jump going from to , hence the forecasts generated from this model would be the ones used.

cafe04 <- window(auscafe, start=2004)  
  
plots <- list()  
for (i in 1:6){  
 fit <- auto.arima(cafe04, xreg=fourier(cafe04, K=i),seasonal=FALSE, lambda = 0)  
 plots[[i]] <- autoplot(forecast(fit,xreg = fourier(cafe04, K=i, h="4")))+  
 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)



Using Fourier terms and ARIMA errors for forecasting monthly expenditure on eating out in Australia

## 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 many reduce accidends immediately, but have a diminishing effect over time as employee take less care when they become faimilar iwth the bew working conditions.

In these situations, we need to allow for lagged effects of the predictor. Suppose we have only one predictor in our model. Then a model which allows for lagged effects can be written as $$ y\_t = \_0 + \_0x\_t + \_1\*x\_{t\_-1} + +*k x*{t-k} +n\_t

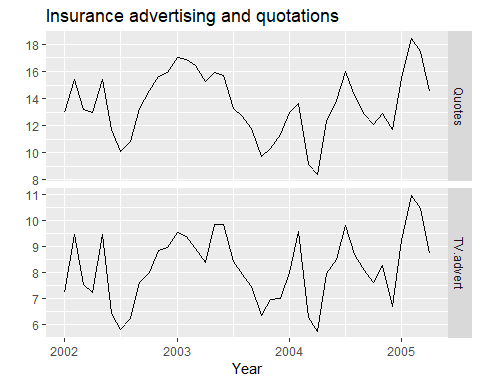
$$

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

### Example: TV advertising and insurance quotations

As 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 @ref(fig:tvadvert) shows the number of quotations the expenditure on television advertising for the company each month from January 2002 to April 2005.

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



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 curent 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. The best model is the one with the smallest AICc value.

# 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))[1:NROW(insurance),]  
  
# Choose optimal lag length for advertising based on AICc  
# Restrict data so models use same fitting period  
fit1 <- auto.arima(insurance[4:40,1], xreg=Advert[4:40,1], d=0)  
fit2 <- auto.arima(insurance[4:40,1], xreg=Advert[4:40,1:2], d=0)  
fit3 <- auto.arima(insurance[4:40,1], xreg=Advert[4:40,1:3], d=0)  
fit4 <- auto.arima(insurance[4:40,1], xreg=Advert[4:40,1:4], d=0)  
  
# Best model fitted to all data (based on AICc)  
# Refit using all data

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

c(fit1[["aicc"]],fit2[["aicc"]],fit3[["aicc"]],fit4[["aicc"]])

## [1] 68.49968 60.02357 62.83253 68.01684

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], d=0))

## Series: insurance[, 1]   
## Regression with ARIMA(3,0,0) errors   
##   
## Coefficients:  
## ar1 ar2 ar3 intercept AdLag0 AdLag1  
## 1.4117 -0.9317 0.3591 2.0393 1.2564 0.1625  
## s.e. 0.1698 0.2545 0.1592 0.9931 0.0667 0.0591  
##   
## sigma^2 estimated as 0.2165: log likelihood=-23.89  
## AIC=61.78 AICc=65.28 BIC=73.6

## ar1 ar2 ar3 intercept AdLag0 AdLag1   
## 1.4117479 -0.9317059 0.3590986 2.0392833 1.2564079 0.1624543

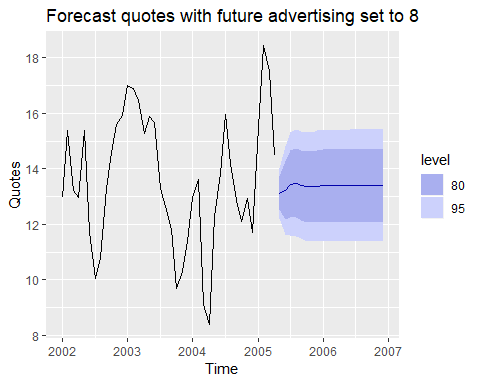
The chosen model has AR(3) errors. The model can be written as:

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

and is white noise.

We can calculate forcasts 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")



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

## Exercises

1. Consider monthly sales and advertising data for an automotive parts company (data set advert).
2. Plot the data using autoplot. Why is it useful to set facets=TRUE?
3. Fit a standard regression model where denotes sales and denotes advertising using the tslm() function.
4. Show that the residuals have significant autocorrelation.
5. What difference does it make you use the function instead:

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

1. 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?
2. Check the residuals of the fitted model.
3. 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.
4. This exercise uses data set huron giving the level of Lake Huron from 1875–1972.
5. Fit a piecewise linear trend model to the Lake Huron data with a knot at 1920 and an ARMA error structure.
6. Forecast the level for the next 30 years.
7. 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.
8. Use the data to calculate the average cost of a night’s accommodation in Victoria each month.
9. Estimate the monthly CPI.
10. Produce time series plots of both variables and explain why logarithms of both variables need to be taken before fitting any models.
11. Fit an appropriate regression model with ARIMA errors. Explain your reasoning in arriving at the final model.
12. 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.)
13. We fitted a harmonic regression model to part of the gasoline series in Exercise 6 in Section @ref(Regr-exercises). We will now revisit this model, and extend it to include more data and ARMA errors.
14. 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 minimizing the AICc or CV value.
15. Now refit the model using auto.arima to allow for correlated errors, keeping the same predictor variables as you used with tslm.
16. 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.
17. Once you have a model with white noise residuals, produce forecasts for the next year.
18. Electricity consumption is often modelled as a function of temperature. Temperature is measured by daily heating degrees and cooling degrees. Heating degrees is C minus the average daily temperature when the daily average is below 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 C when the daily average is above C; otherwise it is zero. Let denote the monthly total of kilowatt-hours of electricity used, let denote the monthly total of heating degrees, and let denote the monthly total of cooling degrees.

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

where

and , and .

1. What sort of ARIMA model is identified for ?
2. The estimated coefficients are

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Estimate | s.e. |  | -value |
|  | 0.0077 | 0.0015 | 4.98 | 0.000 |
|  | 0.0208 | 0.0023 | 9.23 | 0.000 |
|  | 0.5830 | 0.0720 | 8.10 | 0.000 |
|  | -0.5373 | 0.0856 | -6.27 | 0.000 |
|  | -0.4667 | 0.0862 | -5.41 | 0.000 |

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

1. Write the equation in a form more suitable for forecasting.
2. Describe how this model could be used to forecast electricity demand for the next 12 months.
3. Explain why the term should be modelled with an ARIMA model rather than modeling 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 model in producing forecasts.
4. For the retail time series considered in earlier chapters:
5. 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.)
6. Check the residuals of the fitted model. Does the residual series look like white noise?
7. Compare the forecasts with those you obtained earlier using alternative models.

## Further reading

A detailed discussion of dynamic regression models is provided in @Pankratz91. A generalization of dynamic regression models, known as “transfer function models”, is discussed in @BJRL15.

# Forecasting hierachical 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.

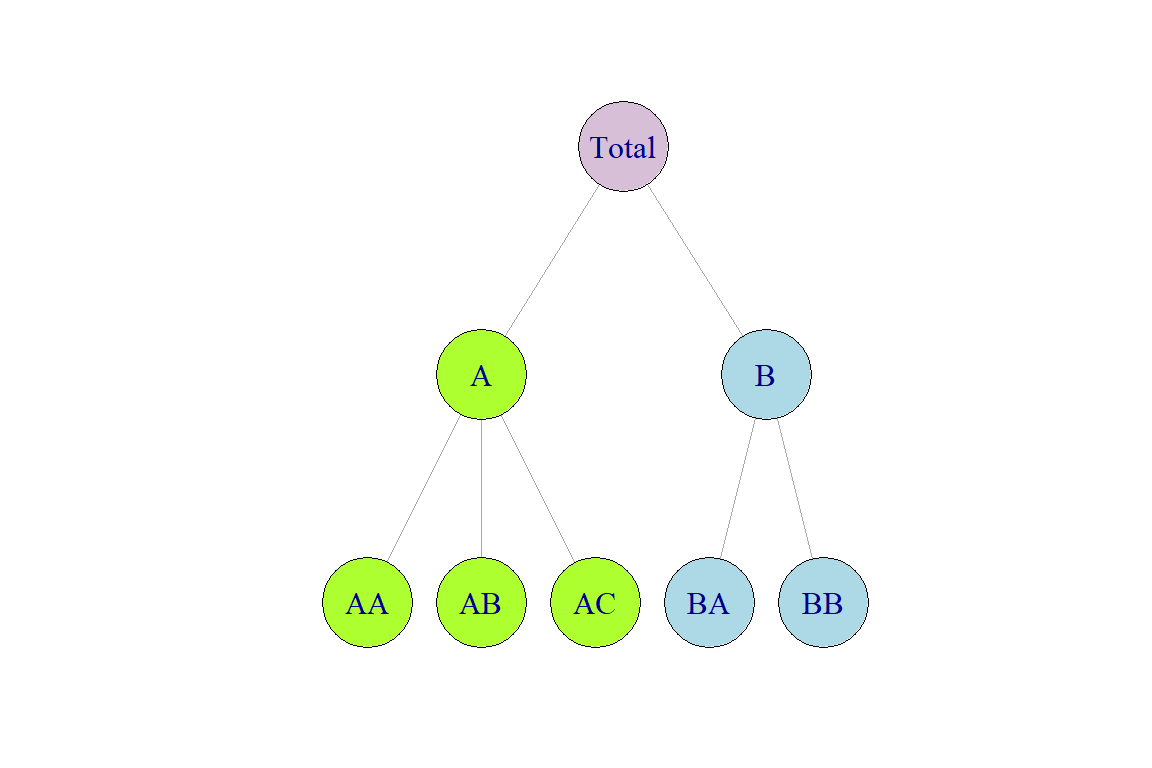
Our bicycle manufacturer may disaggregate sales by both product type and by geographic location. Then we have a more complicated aggregation structure where the product hierarchy and the geographic hierarchy can both be used together. We usually refer to these as “grouped time series”, and discuss them 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 consistent with the aggregation structure of the collection of time series. In Sections 10.3–10.7 we discuss several methods for producing coherent forecasts for both hierarchical and grouped time series.

## Hierachical time series

Figure 10.1 shows level hierachical structure. At the top of the hierarchy, at level 0, is the “Total”, the most aggregate level of the data. We denote as the th observation of the “Total” series for . The “Total” is disaggregated into two series at level 1 and each of these into three and two series respectively at the bottom-level of the hierarchy. Below the top most aggregate level, we denote as $\y{j}{t}$ the th observation of the series which corresponds to node . For example denotes the th observation of the series corresponding to node A at level 1, denotes the th observation of the series corresponding to node AB at level 2, and so on.

The total number of series in the hierarchy is . We denote as the number of series at the bottom-level, a dimenstion that is important in what follows. In this case, . Note that always . 

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

and

Substituting @ref(eq:middlelevel) into @ref(eq:toplevel) we also get $y\_{t}=\y{A}{t}+\y{B}{t}$. These equations can be thought of as aggregation constraints or summing equalities and can be more efficienlty represented using matrix notation. We construct an matrix referred to as the *summing matrix* which dictates how the bottom-level series are aggregated, consistent with the aggregation structure. For the hierarchical structure in Figure @ref(fig:HierTree) we write

$$
\begin{bmatrix}
y\_{t} \\
\y{A}{t} \\
\y{B}{t} \\
\y{AA}{t} \\
\y{AB}{t} \\
\y{AC}{t} \\
\y{BA}{t} \\
\y{BB}{t}
\end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\y{AA}{t} \\
\y{AB}{t} \\
\y{AC}{t} \\
\y{BA}{t} \\
\y{BB}{t}
\end{bmatrix}
$$

or in more compact notation

where is a -dimensional vector of all the observations in the hierarchy at time , is the summing matrix as defined above, and is an -dimensional vector of all the observations in the bottom-level of the hierarchy at time . Note that the first row in the summing represents equation @ref(eq:toplevel) above, the second and third row represent @ref(eq:middlelevel). The rows below these comprise an -dimensional identity matrix so that each bottom-level observation on the right hand side of the equation is equal to itself in the left hand side.

### Example: Australian tourism hierarchy

Australia is divided into eight geographical areas (some referred to as states and others as territories) iwth each one having its own government and some economic and administrative autonomy. Each of these can be further subdivided into smaller areas of iterest referred to as zones.

usiness planners and tourism authorities are interested in forecasts for the whole of Australia, the states and the territories, and also the regions. In this example we concentrate on quarterly domestic tourism demand, measured as the number of visitor nights Australians spend away from home, for the six states of Australia, namely: New South Wales (NSW), Queensland (QLD), South Australia (SAU), Victoria (VIC), Western Australia (WA) 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), Costal (SAUCoast), Inner (SAUInner) |
| VIC | Metro (VICMetro), West Coast (VICWstCo), East Coast (VICEstCo), Inner (VICInner) |
| WAu | Metro (WAUMetro), Costal (WAUCoast), Inner (WAUInner) |
| OTH | Metro (OTHMetro), Non-Metro (OTHNoMet) |

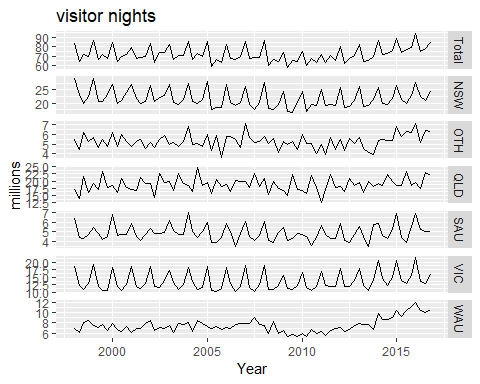
In summary, we consider five zones for NSW, four zones for VIC, and three zones for QLD, SAU and WAU. Note that Metro zones contain the capital cities and surrounding areas. For further details on these geographic areas, please refer to Appendix C in [Wickramasuriya, Athanasopoulos, & Hyndman (2018)](https://otexts.org/fpp2/hts.html#ref-Mint).

To create a hierarchical time series we use the hts function as shown in the code below. The function requires as inputs the bottom-level time series and information about the hierarchical structure. vn2 is a time series matrix containing the bottom-level series. There are alternative ways to pass to the function the structure of the hieararchy. In this case we are using the characters input. The first three characters of each column name of vn2 capture the categories at the first level of the hierarchy (States). The following five characters capture the bottom-level categories (Zones).

library(hts)

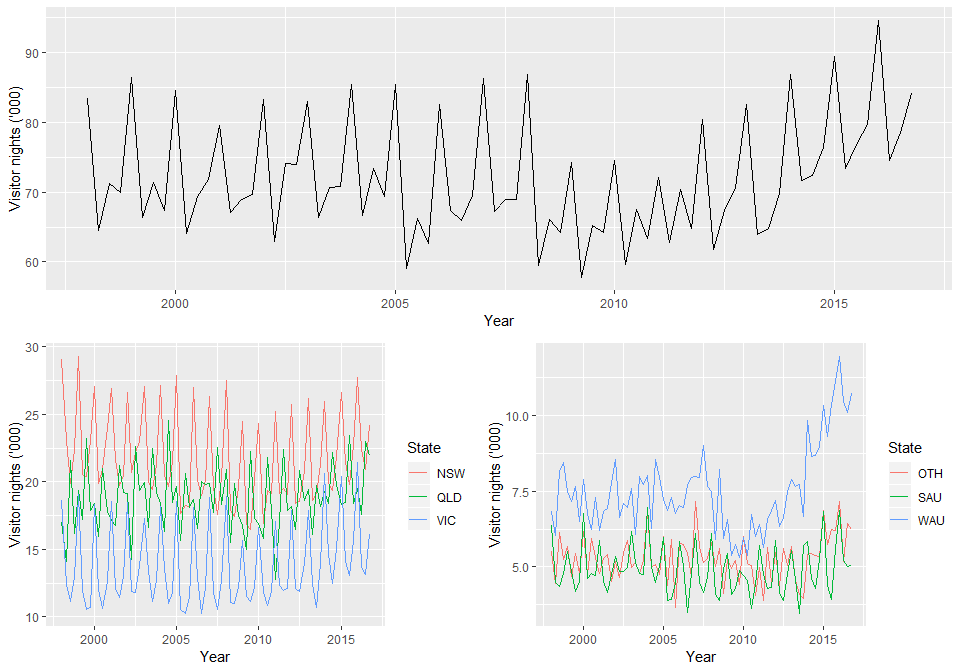
## Warning: package 'hts' was built under R version 3.5.1

tourism.hts <- hts(visnights, characters=c(3,5))  
tourism.hts %>% aggts(levels=0:1) %>%   
 autoplot(facets=T)+  
 xlab("Year")+ylab("millions")+ggtitle("visitor nights")



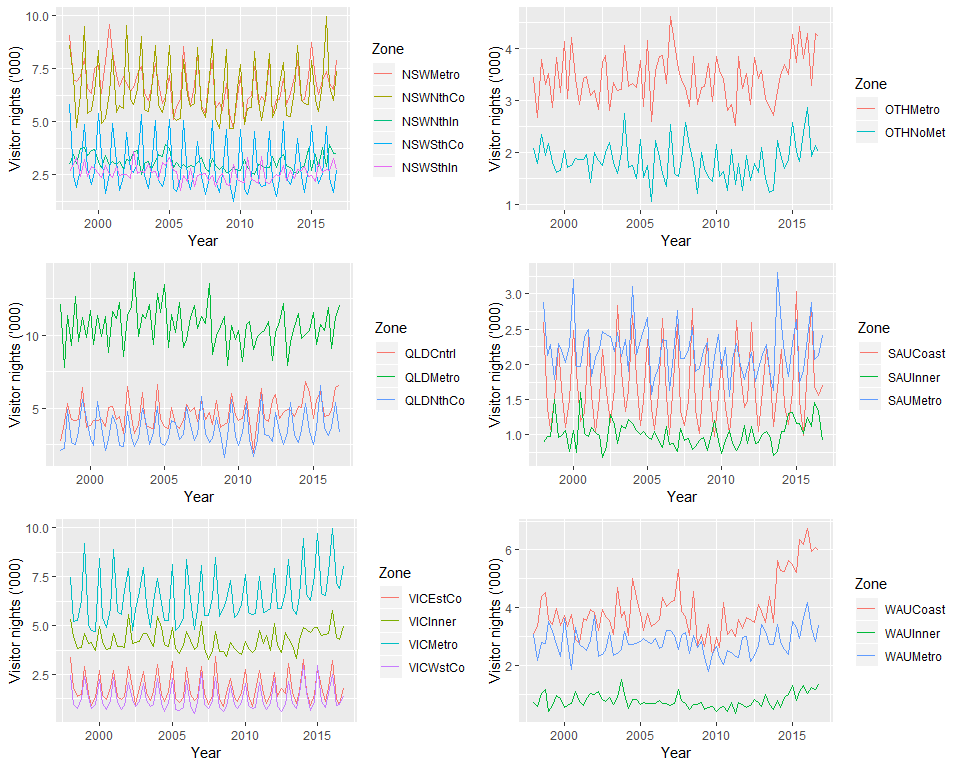
The top plot in Figure @ref(fig:tourismStates) shows the total number of visitor nights for the total of Australia while the plots bellow show the visitor nights disaggregated by state. These reveal diverse and rich dynamics at the aggregate national level and the first level of disaggregation across each state. The aggts function extracts time series from a hts object for any level of aggregation.

tourismL0 <- aggts(tourism.hts, levels=0)  
p1 <- autoplot(tourismL0)+  
 xlab("Year")+  
 ylab("Visitor nights ('000)")+  
 scale\_color\_discrete(guide=guide\_legend(title="State"))  
  
tourismL1 <- aggts(tourism.hts, levels = 1)  
p2<-autoplot(tourismL1[,c(1,3,5)]) +   
 xlab("Year") +  
 ylab("Visitor nights ('000)")+  
 scale\_colour\_discrete(guide = guide\_legend(title = "State"))  
   
p3<-autoplot(tourismL1[,c(2,4,6)]) +   
 xlab("Year") +  
 ylab("Visitor nights ('000)")+  
 scale\_colour\_discrete(guide = guide\_legend(title = "State"))  
  
lay=rbind(c(1,1),c(2,3))  
gridExtra::grid.arrange(p1, p2,p3, layout\_matrix=lay)



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

The plots in Figure @ref(fig:tourismZones) below show the bottom-level time series, i.e., the visitor nights for each zone. These help us visualise the diverse individual dynamics within each zones and assist in identifying unique and important time series. Notice for example the costal WAU zone which shows significant growth over the last few years.

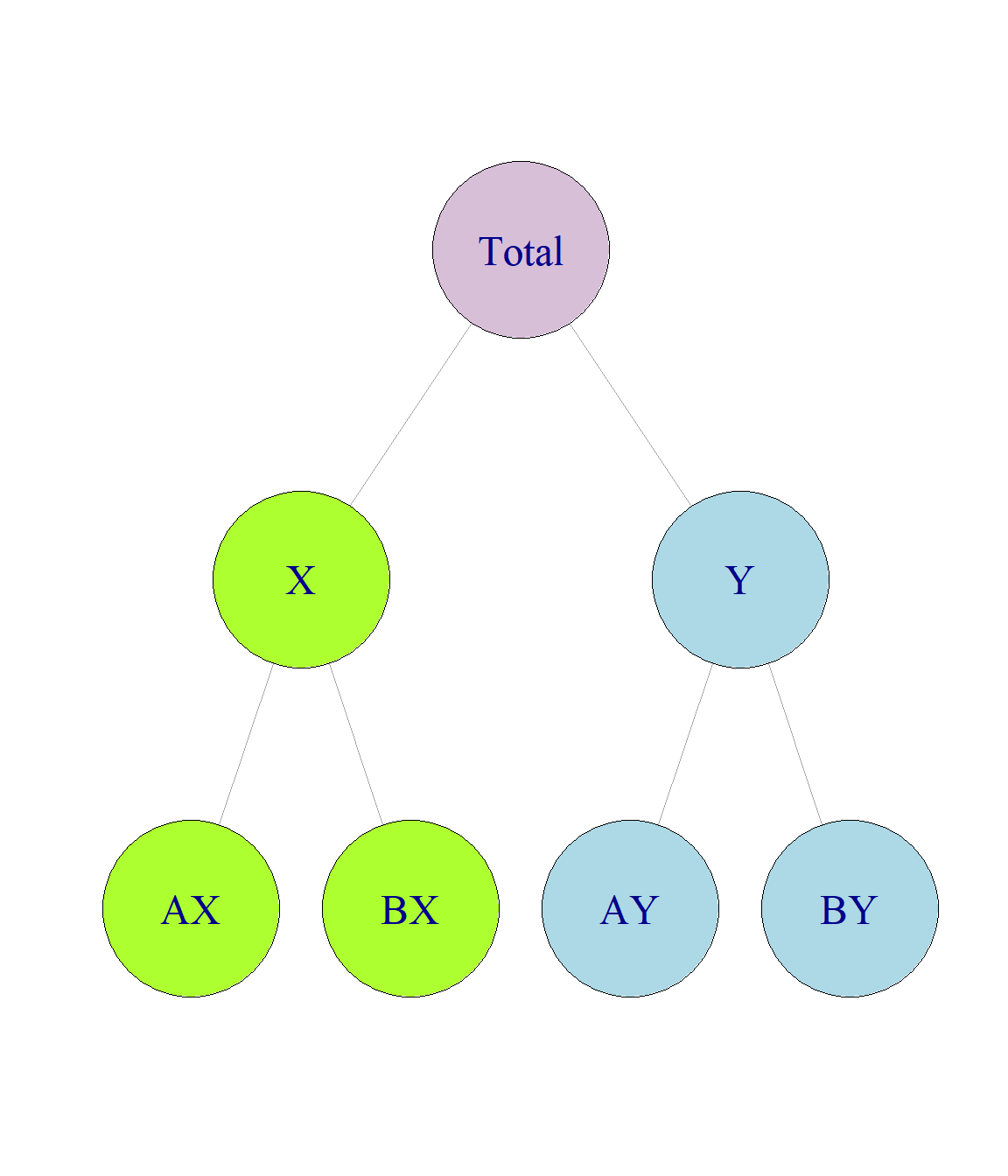
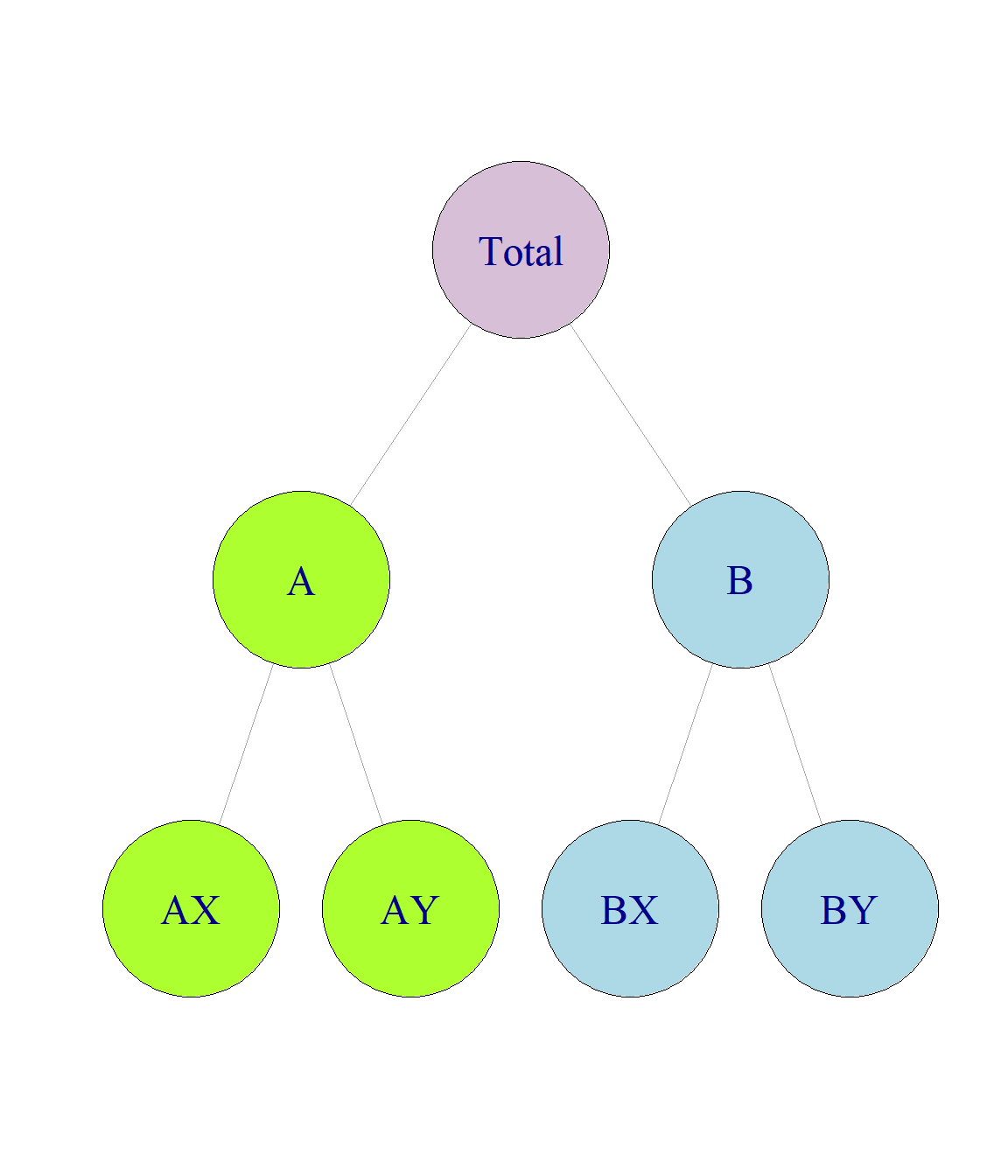


Australian domestic visitor nights over the period 1998 Q1 to 2016 Q4 disaggregated by Zones.

## Grouped time series

Another possibility is that series can be naturally grouped together based on attributes without necessarily imposing a hierarchical structure. For example the bicycles sold by the warehouse can be for males, females or unisex. Frames can be carbon, aluminium or steel. They can be single speed or have multiple gears. We can also get a similar structure when we combine two hierarchical structures. For example the bicycle manufacturer may disaggregate sales by product and also by geographical location. We refer to these as *grouped time series*. With grouped time series we still have a hierarchical structure however the structure does not naturally disaggregate in a unique way. For example we can disaggregate the bicycles by product type and then geographical location but also vice versa.

Figure @ref(fig:GroupTree) below shows a -level grouped structure. At the top of the grouped structure, is the “Total”, the most aggregate level of the data, again represented by . The “Total” can be disaggregated by attributes (A, B) forming series $\y{A}{t}$ and $\y{B}{t}$, or by attributes (X, Y) forming series $\y{X}{t}$ and $\y{Y}{t}$. At the bottom the data are disaggregated by both attributes.



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

However, for the first level of the grouped structure,

but also

These equalities can again be represented by the summing matrix which recall is of dimension . The total number of series is with series at the bottom-level. For the grouped structure in Figure @ref(fig:GroupTree) we write

$$
\begin{bmatrix}
y\_{t} \\
\y{A}{t} \\
\y{B}{t} \\
\y{X}{t} \\
\y{Y}{t} \\
\y{AX}{t} \\
\y{AY}{t} \\
\y{BX}{t} \\
\y{BY}{t}
\end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\y{AX}{t} \\
\y{AY}{t} \\
\y{BX}{t} \\
\y{BY}{t}
\end{bmatrix}
$$

where now the second and third rows of represent @ref(eq:middlelevelAB) and the fourth and fifth rows represent @ref(eq:middlelevelXY).

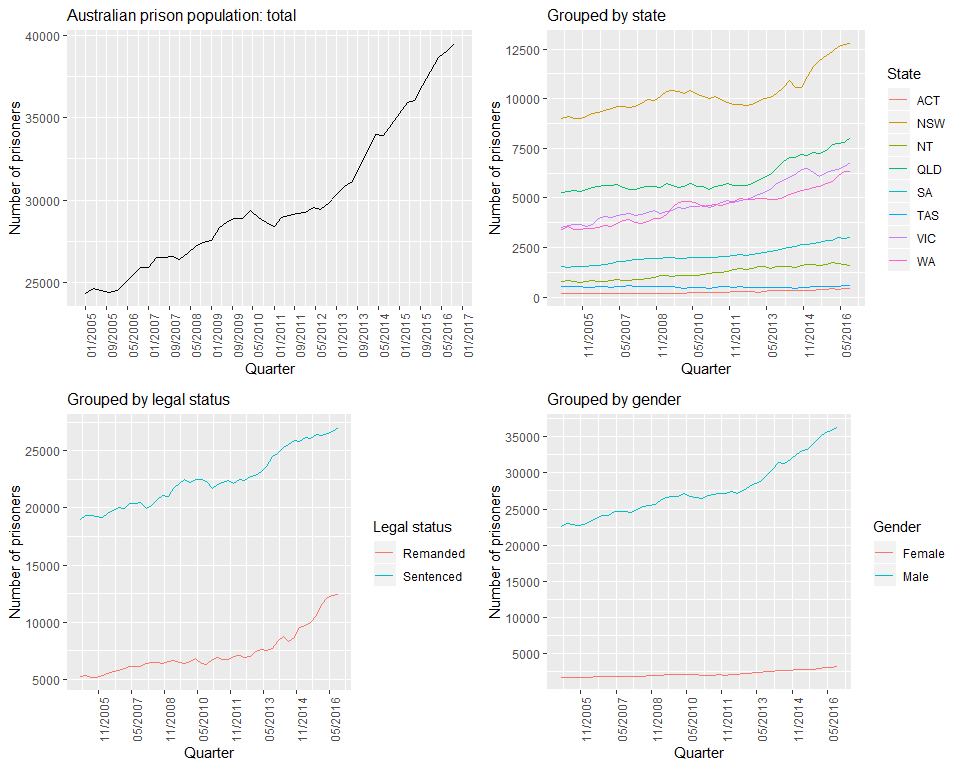
where now the second and third rows of represent @ref(eq:middlelevelAB) and the fourth and fifth rows represent @ref(eq:middlelevelXY).

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

### Example: Australian prison population

The left plot in the top row of Figure @ref(fig:prison) 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 plots show the prison population grouped by (i) state[[4]](#footnote-145) (ii) legal status, whether prisoners have already been sentenced or are in remand waiting for a sentence, and (iii) gender.

## # A tibble: 6 x 5  
## state gender legal t count  
## <fct> <fct> <fct> <date> <dbl>  
## 1 ACT Female Remanded 2005-03-01 2  
## 2 ACT Female Remanded 2005-06-01 4  
## 3 ACT Female Remanded 2005-09-01 1  
## 4 ACT Female Remanded 2005-12-01 4  
## 5 ACT Female Remanded 2006-03-01 4  
## 6 ACT Female Remanded 2006-06-01 6



To create a grouped time series, we use the gts function as shown below. Similarly to the hts function, the gts function requires as inputs the bottom-level time series and informatio about the grouping structure. prison is a time series matrix containing the bottom-level times series. Similarly to the hts function 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. The code below shows examples for both these.

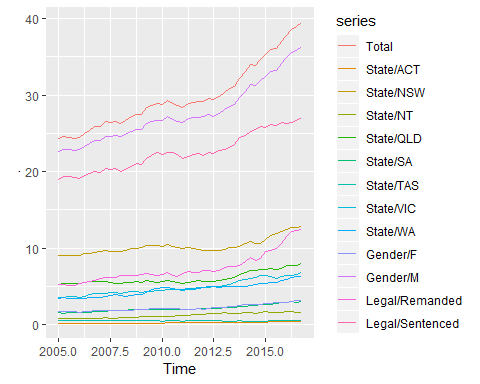
fpp2::prison %>% as.tibble() %>% head()

## # A tibble: 6 x 32  
## `ACTFRemanded ` ACTFSentenced `ACTMRemanded ` ACTMSentenced  
## <dbl> <dbl> <dbl> <dbl>  
## 1 2 5 65 106  
## 2 4 6 66 107  
## 3 1 5 63 118  
## 4 4 8 63 129  
## 5 4 9 60 117  
## 6 6 9 65 110  
## # ... with 28 more variables: `NSWFRemanded ` <dbl>, NSWFSentenced <dbl>,  
## # `NSWMRemanded ` <dbl>, NSWMSentenced <dbl>, `NT FRemanded ` <dbl>, `NT  
## # FSentenced` <dbl>, `NT MRemanded ` <dbl>, `NT MSentenced` <dbl>,  
## # `QLDFRemanded ` <dbl>, QLDFSentenced <dbl>, `QLDMRemanded ` <dbl>,  
## # QLDMSentenced <dbl>, `SA FRemanded ` <dbl>, `SA FSentenced` <dbl>, `SA  
## # MRemanded ` <dbl>, `SA MSentenced` <dbl>, `TASFRemanded ` <dbl>,  
## # TASFSentenced <dbl>, `TASMRemanded ` <dbl>, TASMSentenced <dbl>,  
## # `VICFRemanded ` <dbl>, VICFSentenced <dbl>, `VICMRemanded ` <dbl>,  
## # VICMSentenced <dbl>, `WA FRemanded ` <dbl>, `WA FSentenced` <dbl>, `WA  
## # MRemanded ` <dbl>, `WA MSentenced` <dbl>

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

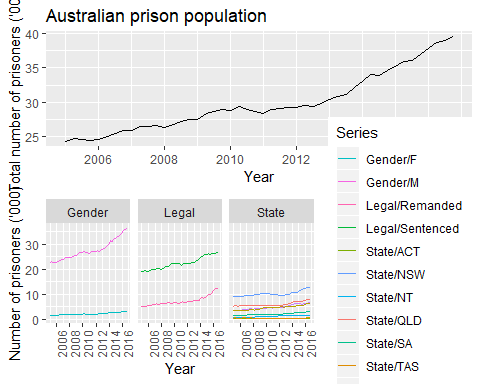
One way to plot the main groups is as follows.

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.

## The bottom-up approach

A commonly applied method for generating coherent forecasts is the bottom-up approach. This approach involves first generating base forecasts for each series at the bottom-level and then aggregating these upwards to produce forecasts for all the series in the structure.

For example, for the hierarchy of Figure @ref(fig:HierTree) we first generate -step-ahead base forecasts for each of the bottom-level series:

$$\yhat{AA}{h},~~\yhat{AB}{h},~~\yhat{AC}{h},~~ \yhat{BA}{h}~~\text{and}~~\yhat{BB}{h}.$$

Aggregating these up the hierarchy we get -step-ahead coherent forecasts for the rest of the series:

$$\tilde{y}\_{h}=\yhat{AA}{h}+\yhat{AB}{h}+\yhat{AC}{h}+\yhat{BA}{h}+\yhat{BB}{h},~~~\ytilde{A}{h}= \yhat{AA}{h}+\yhat{AB}{h}+\yhat{AC}{h}$$

and

$$\ytilde{B}{h}= \yhat{BA}{h}+\yhat{BB}{h}.$$

As in equation @ref(eq:smatrix) we can employ the summing matrix here and write

$$
\begin{bmatrix}
\tilde{y}\_{h} \\
\ytilde{A}{h} \\
\ytilde{B}{h} \\
\ytilde{AA}{h} \\
\ytilde{AB}{h} \\
\ytilde{AC}{h} \\
\ytilde{BA}{h} \\
\ytilde{BB}{h}
\end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\yhat{AA}{h} \\
\yhat{AB}{h} \\
\yhat{AC}{h} \\
\yhat{BA}{h} \\
\yhat{BB}{h}
\end{bmatrix}.
$$

In general, using more compact notation, the bottom-up approach can be represented as

where is an -dimensional vector of coherent -step-ahead forecasts of each time series within any aggregation structure and is an -dimensional vector of -step-ahead base forecasts for each of the bottom-level series of any aggregation structure. Note that for the bottom-up approach the coherent forecasts for the bottom-level series are equal to the base forecasts, i.e., .

The greatest 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 bottom-up approach is implemented in the forecast package by setting

forecast(..., method="bu",...)

## Top-down approaches

Top-down approaches involve first generating forecasts for the “Total” series on the top of the aggregation structure and then disaggregating these downwards. We let be a set of proportions which dictate how the base forecasts of the “Total” series are to be distributed to revised forecasts for each series at the bottom-level of the structure. For example for the hierarchy of Figure @ref(fig:HierTree) using proportions we get,

$$
\ytilde{AA}{t}=p\_1\hat{y}\_t,~~~\ytilde{AB}{t}=p\_2\hat{y}\_t,~~~\ytilde{AC}{t}=p\_3\hat{y}\_t,~~~\ytilde{BA}{t}=p\_4\hat{y}\_t~~~\text{and}~~~~~~\ytilde{BB}{t}=p\_5\hat{y}\_t.
$$

Using matrix notation we can stack the set of proportions in a -dimensional column vector [[5]](#footnote-151) and write

Once the bottom-level -step-ahead forecasts have been generated these can be aggregated to generate coherent forecasts for the rest of the series. In general using the summing matrix and for a specified set of proportions, top-down approaches can be represented as,

Note that for all top-down approaches the top-level coherent forecasts are equal to the top-level base forecasts, i.e., .

The most common top-down approaches specify proportions based on the historical proportions of the data. The two most common versions follow. These peformed well in the study of @GroSoh1990 hence the acronyms in the forecast package.

### Average historical proportions

for . Each proportion reflects the average of the historical proportions of the bottom-level series over the period relative to the total aggregate .

This approach is implemented in the forecast package by setting

forecast(...,method = "tdgsa",...).

### 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 . Each proportion captures the average historical value of the bottom-level series relative to the average value of the total aggregate .

This approach is implemented in the forecast package by setting

forecast(..., method = "tdgsf", ...).

The greatest attribute of such top-down approaches is their simplicity to apply. 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 very useful with low count data. On the other hand, their greatest 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

An alternative approach that improves on the historical and static nature of the proportions specified above is to use forecast proportions introduced in @AthEtAl2009.

To demonstrate the intuition of this method, consider a one level hierarchy. We first generate -step-ahead base forecasts for all the series. At level 1 we calculate the proportion of each -step-ahead base forecast to the aggregate of all the -step-ahead base forecasts at this level. We refer to these as the forecast proportions and we use these to disaggregate the top-level -step-ahead forecast and generate coherent forecasts for the whole of the hierarchy.

For a -level hierarchy this process is repeated for each node going from the top to the very bottom-level. Applying this process leads to the following general rule for obtaining the forecast proportions

for . These forecast proportions disaggregate the -step-ahead base forecast of the “Total” series to -step-ahead coherent forecasts of the bottom-level series. is the -step-ahead base forecast of the series that corresponds to the node which is levels above . is the sum of the -step-ahead base forecasts below the node that is levels above node and are directly connected to that node.

We will use the hierarchy of Figure @ref(fig:HierTree) to explain this notation and to demonstrate how this general rule is reached. Assume we have generated base forecasts for each series in the hierarchy. Recall that for the top-level “Total” series, , for any top-down approach. Here are some examples using the above notation:

* $\Shat{AA}{h}{1} = \Shat{AB}{h}{1}= \Shat{AC}{h}{1}= \yhat{AA}{h}+\yhat{AB}{h}+\yhat{AC}{h}$
* $\Shat{AA}{h}{2} = \Shat{AB}{h}{2}= \Shat{AC}{h}{2}= \Shat{A}{h}{1} = \Shat{B}{h}{1}= \hat{S}\_{h}= \yhat{A}{h}+\yhat{B}{h}$

Moving down the farthest left branch of the hierarchy coherent forecasts are given by

$$
\ytilde{A}{h} = \Bigg(\frac{\yhat{A}{h}}{\Shat{A}{h}{1}}\Bigg) \tilde{y}\_{h} =
\Bigg(\frac{\yhat{AA}{h}^{(1)}}{\Shat{AA}{h}{2}}\Bigg) \tilde{y}\_{h}
$$

and

$$
\ytilde{AA}{h} = \Bigg(\frac{\yhat{AA}{h}}{\Shat{AA}{h}{1}}\Bigg) \ytilde{A}{h}
=\Bigg(\frac{\yhat{AA}{h}}{\Shat{AA}{h}{1}}\Bigg) \Bigg(\frac{\yhat{AA}{h}^{(1)}}{\Shat{AA}{h}{2}}\Bigg)\tilde{y}\_{h}.
$$

Consequently,

$$
p\_1=\Bigg(\frac{\yhat{AA}{h}}{\Shat{AA}{h}{1}}\Bigg) \Bigg(\frac{\yhat{AA}{h}^{(1)}}{\Shat{AA}{h}{2}}\Bigg)
$$

The other proportions can be similarly obtained. The greatest disadvantage of the top-down forecast proportions approach, which is a disadvantage of any top-down approach, is that they do not produce unbiased revised forecasts even if the base forecasts are unbiased as shown by @HynEtAl2011

This approach is implemented in the forecast package by setting

forecast(..., method = "tdfp", ...).

### Bibliography

Gross, C. W., & Sohl, J. E. (1990). Disaggregation methods to expedite product line forecasting. Journal of Forecasting, 9, 233–254. <https://doi.org/10.1002/for.3980090304>

Athanasopoulos, G., Ahmed, R. A., & Hyndman, R. J. (2009). Hierarchical forecasts for Australian domestic tourism. International Journal of Forecasting, 25, 146–166. <https://robjhyndman.com/publications/hierarchical-tourism/>

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. <https://robjhyndman.com/publications/hierarchical>

## Middle-out approach

The middle-out approache 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 moddle level, coherent forecasts are generated using the bottom-up approach by aggregating the “middle-evel” 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.

## Mapping matrices

Denote as a set of -step-ahead base forecasts generated for each series in a hierarchial or grouped structure and stacked the same way as the data. For example for the hierarchy of Figure @ref(fig:HierTree)

$$
\bm{\hat{y}}\_h=\begin{bmatrix}
\hat{y}\_h \\
\yhat{A}{h} \\
\yhat{B}{h} \\
\yhat{AA}{h} \\
\yhat{AB}{h} \\
\yhat{AC}{h} \\
\yhat{BA}{h} \\
\yhat{BB}{h} \\
\end{bmatrix}.
$$

In general, all forecasting approaches for either hierarchical or grouped structures can be represented as

where reading from right to left, is the set of -step-ahead base forecasts as defined above, is a matrix that projects the base forecasts into the bottom-level, and the summing matrix sums these up using the aggregation structure to produce a set of coherent forecasts .

The matrix is defined according to the approach implemented. For example if the bottom-up approach is used to forecast the hierarchy of Figure @ref(fig:HierTree),

Notice that the comprises two partitions. The first three columns which zero out the base forecasts of the series above the bottom-level and the -dimensional identity matrix which picks only the base forecasts of the bottom-level, to then be summed up the hierarchy by the matrix.

If any of the top-down approaches were used then

The first column includes a 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 matrix. The rest of the columns zero out the base forecasts below the very top-level of aggregation.

For a middle out approach the 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 to then be summed up the hierarchy with the summing matrix, with all other base forecasts being zeroed out.

## The optimal reconciliation approach

All the approaches we have considered so far in this chapter involve choosing a particular level in the aggregation structure, generating base forecasts for that level, and then either aggregating these up or disaggregating them down to generate coherent forecasts for the rest of the series. The above examples clearly reflect this through the matrix.

In this section we introduce an approach that instead involves generating base forecasts for each series in the aggregation structure. The base forecasts are then reconciled to generate a set of coherent forecasts that are as close as possible to the base forecasts. Therefore we describe the resulting set of coherent forecasts as optimally reconciled coherent forecasts.

The general idea is derived from wanting to find a matrix that minimises the forecast error of a set of coherent forecasts from forecasting each time series in the structure. In what follows we present a simplified summary of the approach. There are a few necessary steps that need to be followed in order to get a good flavour of the approach. These unfortunately complicate the presentation. However, following through these steps we get to the specification of in equation @ref(eq:MinT) which is labelled the *MinT* estimator as it *Min*imises the *T*race of the forecast errors of the coherent forecasts across the whole structure. For further details and discussion please refer to @Mint.

Let

be the forecast errors after having produced a set of coherent forecasts across the whole structure, stacked in the same order as the data. Note that this expression is a generalisation of a forecast error as defined in Section @ref(accuracy) using matrix notation. It can be easily shown through Equation @ref(eq:SP) that for a set of base forecasts that are unbiased,[[6]](#footnote-162) defining a matrix such that , generates a set of coherent forecasts that are also unbiased. Note that this will not hold for any top-down approach.

@Mint show in Lemma 1 that

where is the variance-covariance matrix of the -step-ahead base forecast errors. This is a very important result as it shows that the forecast error variance of the coherent forecasts is a function of the error variance of base forecasts . The objective is to find a matrix that minimises the error variance of the coherent forecasts. @Mint show in Theorem 1 that the optimal matrix that minimises the such that , is given by

referred to as the *MinT* estimator.

Note that the MinT estimator involves the forecast error variance of the -step-ahead base forecasts. As this is challenging to estimate we provide below three simplifying specifications which have been shown to work well in both simulations and in practice.

1. Set where .[[7]](#footnote-163) This is the most simplifying assumption to make. Note that in this case is independent of the data and no further estimation is required. The disadvantage is however that this specification does not account for the differences in the scale between the levels of the structure which naturally exist due to aggregation. The two specifications that follow do account for this. This approach is implemented in the forecast package by setting

* forecast(..., method = "comb", weights = "ols", ...).
* The weights here are referred to as the OLS (ordinary least squares) estimator as setting in @ref(eq:MinT) gives the least squares estimator we introduced in Section @ref(Regr-MatrixEquations) with and .

1. Set where and
2. where is an -dimensional vector of residuals of the models that generated the base forecasts stacked in the same order as the data. Each element in this vector is the same as defined in Section @ref(residuals). The approach is implemented by setting

* forecast(..., method = "comb", 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*.

1. Set , where and where is a unit column vector of dimension . This specification assumes that the bottom-level base forecast errors each have variance and are uncorrelated between nodes. Hence each element of the diagonal matrix contains the number of forecast error variances contributing to that aggregation level. This estimator only depends on the structure of the hierarchy or the grouped time series. It is therefore referred to as the specification that applies *structural scaling*. Notice that the structural scaling assumes equivariant forecast errors only at the bottom-level of the structure and not across all levels which is unrealistically assumed by the first specification. Furthermore, applying the structural scaling specification is particularly useful in cases where residuals are not available and therefore variance scaling cannot be applied. For example, in cases where the base forecasts are generated by judgemental forecasting introduced in Chapter @ref(ch-judgmental). This approach is implemented by setting

* forecast(..., method = "comb", weights = "nseries", ...).

1. An alterantive to the above simplifying specifications is to direclty estimate the full covariance matrix. The most obvious and simple way would be to use the sample covariance. This is implemented by setting

* forecast(..., method = "comb", weights = "mint", covariance = "sam", ...).

However, for cases that 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

forecast(..., method = "comb", weights = "mint", covariance = "shr", ...).

For more details on MinT please refer to @Mint.

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 very 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 a hierarchical structure reflecting the geographical division of a country into states, regions, down to a very fine grid of statistical local areas. There are significant differences between the seasonal patterns in the number of tourists visiting a state or a region that is mainly seen as a summer destination versus a state or a region that caters for winter activities. These differences will be smoothed at the country level due to aggregation and on the other hand it may be extremely challenging to identify at the very bottom-level of a statistical local area due to noise. Another example for a grouped structure is the difference in the sales of clothes between genders. Such differences will be completely smoothed out at the very top-level of aggregation considering total sales, or may be very challenging to identify due to noise at the very bottom-level.

### 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 reconcilation approach with the WLS estimator using variance scaling.

library(hts)  
library(scales)  
  
fcsts = forecast(prison.gts, h = 8, method = "comb", weights = "wls", fmethod = "ets")  
  
plot(fcsts,levels = 0, color\_lab=TRUE)  
title(main = "Total")  
plot(fcsts,levels = 1, color\_lab=TRUE)  
title(main = "Grouped by state")  
plot(fcsts,levels = 2, color\_lab=TRUE)  
title(main = "Grouped by leagal status")  
plot(fcsts,levels = 3, color\_lab=TRUE)  
title(main = "Grouped by gender")

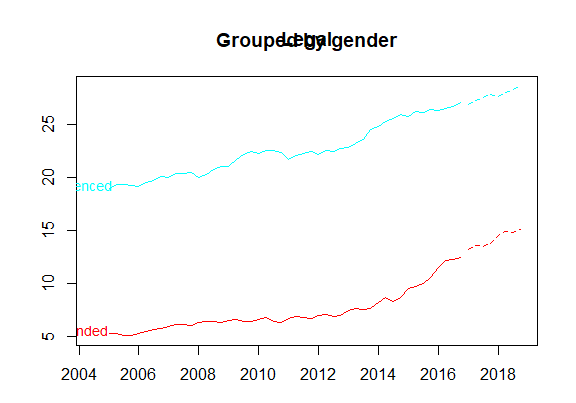
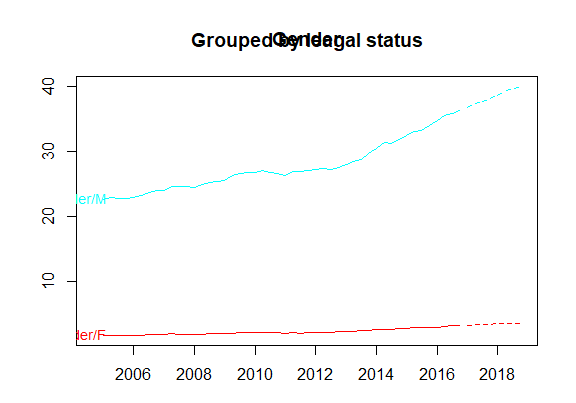
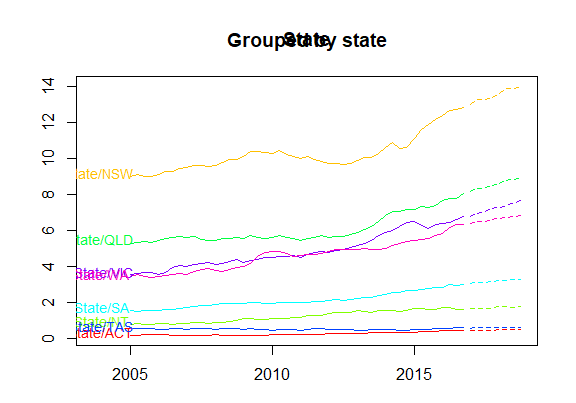
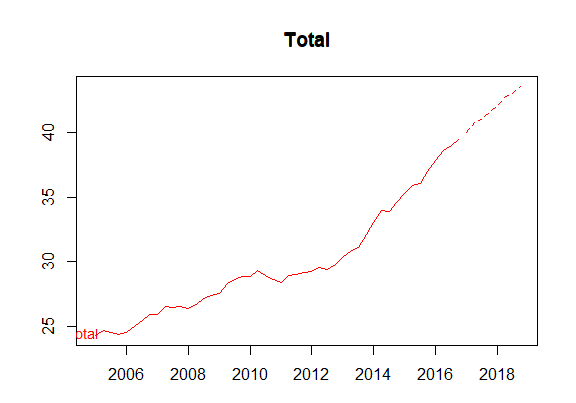
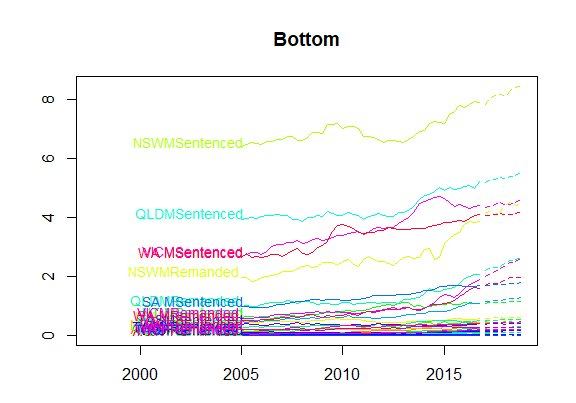
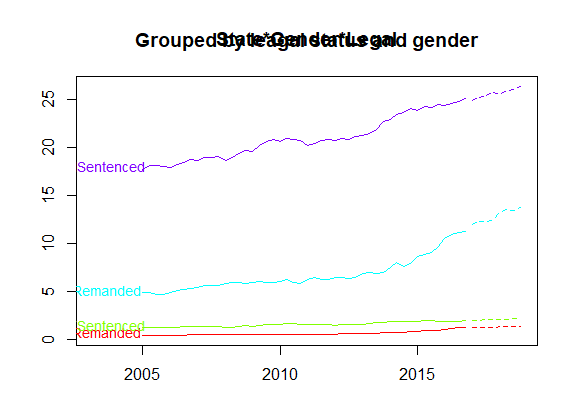
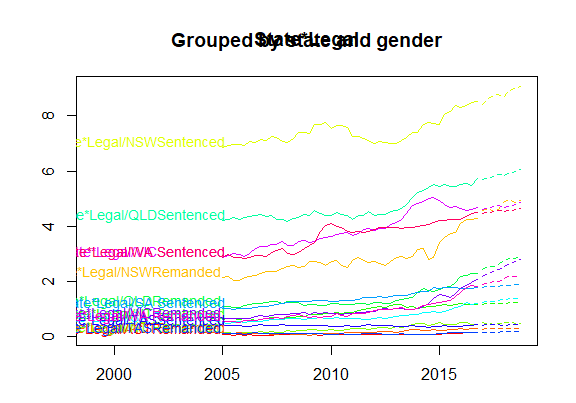
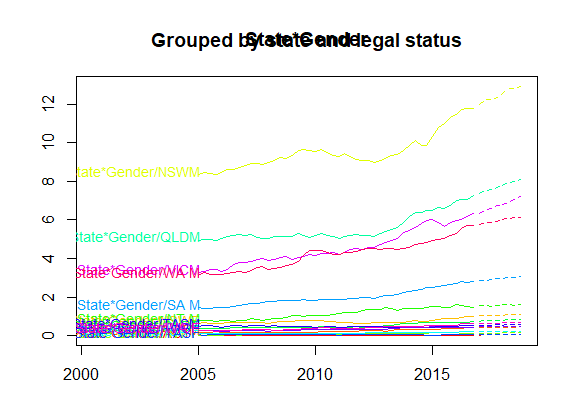


Figure @ref(fig:prisonFcasts2) plots the coherent forecasts for all the interactions of the attributes down to the bottom-level.



The accuracy.gts common is useful for evaluating the forecast accuracy accross hierachical 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 reconcilation approach uses information from all levels in the structure it geenerates more accurate coherent forecasts than the other traditional alternatives which use limited information.

# data set  
train <- window(prison.gts, end=c(2014,4))  
test <- window(prison.gts,start=2015)  
  
fcsts.opt = forecast(train, h = 8, method = "comb", weights = "wls", fmethod = "ets")  
fcsts.bu = forecast(train, h = 8, method = "bu", fmethod = "ets")  
  
tab <- matrix(NA,ncol=4,nrow=6)  
rownames(tab) <- c("Total", "State", "Legal status", "Gender","Bottom", "All series")  
colnames(tab) <- c("Bottom-up MAPE","Bottom-up MASE","Optimal MAPE","Optimal MASE")  
  
tab[1,] <- c(accuracy.gts(fcast.bu,test,levels=0)[c("MAPE","MASE"),"Total"],  
 accuracy.gts(fcsts.opt,test,levels = 0)[c("MAPE","MASE"),"Total"])  
  
j=2  
for(i in c(1:3,7)){  
 tab[j,] <-c(mean(accuracy.gts(fcsts.bu,test,levels = i)["MAPE",]),  
 mean(accuracy.gts(fcsts.bu,test,levels = i)["MASE",]),  
 mean(accuracy.gts(fcsts.opt,test,levels = i)["MAPE",]),  
 mean(accuracy.gts(fcsts.opt,test,levels = i)["MASE",]))  
j=j+1  
}  
  
tab[6,] <- c(mean(accuracy.gts(fcsts.bu,test)["MAPE",]),  
 mean(accuracy.gts(fcsts.bu,test)["MASE",]),  
 mean(accuracy.gts(fcsts.opt,test)["MAPE",]),  
 mean(accuracy.gts(fcsts.opt,test)["MASE",]))  
  
knitr::kable(tab, digits=2, booktabs=TRUE)

## Exercises

1. Write out the matrices for the Australian tourism hierarchy and the Australian prison grouped structure. Use the smatrix command to verify your answers.
2. Assume that a set of base forecasts are unbiased, i.e., . A set of coherent forecasts will also unbiased iff . In this case . Show that this is true for the bottom-up and optimal reconciliation approaches but not for any top-down or middle-out approaches.
3. Generate 8-step-ahead bottom-up forecasts using arima models for the vn2 Australian domestic tourism data. Plot the coherent forecatsts by level and comment on their nature. Are you satisfied with these forecasts?
4. Model the aggregate series for Australian domestic tourism data vn2 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 3 for the aggregate level.
5. Generate 8-step-ahead optimally reconciled coherent forecasts using arima base forecasts for the vn2 Australian domestic tourism data. Plot the coherent forecatsts by level and comment on their nature. How and why are these different to the bottom-up forecasts generated in question 3 above.
6. Define as a test-set the last two years of the vn2 Australian domestic tourism data. Generate, bottom-up, top-down and optimally reconciled forecasts for this period and compare their forecasts accuracy.

## 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. (2018). Athanasopoulos, Hyndman, Kourentzes, & Petropoulos (2017) extends the reconciliation approach to deal with temporal hierarchies.

### Bibliography

Gross, C. W., & Sohl, J. E. (1990). Disaggregation methods to expedite product line forecasting. Journal of Forecasting, 9, 233–254. <https://doi.org/10.1002/for.3980090304>

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. <https://robjhyndman.com/publications/hierarchical/>

Athanasopoulos, G., Ahmed, R. A., & Hyndman, R. J. (2009). Hierarchical forecasts for Australian domestic tourism. International Journal of Forecasting, 25, 146–166. <https://robjhyndman.com/publications/hierarchical-tourism/>

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. <https://robjhyndman.com/publications/hgts/>

Wickramasuriya, S. L., Athanasopoulos, G., & Hyndman, R. J. (2018). Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. J American Statistical Association, to appear. <https://robjhyndman.com/publications/mint/>

Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. European Journal of Operational Research, 262(1), 60–74. <https://robjhyndman.com/publications/temporal-hierarchies/>

# Advanced forecasting methods

In this chapter, we briefly discuss four more advanced forecasting methods that build on the models in ealier chapters.

## 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, dily 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 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.

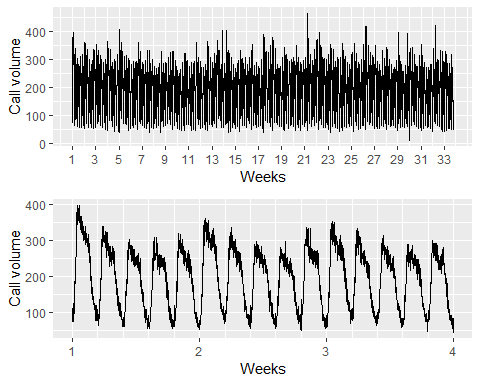
Most of the methods we have considered so far are unable to deal with these seasonal complexities. Even the ts class in R can only handle one type of seasonality, which is usually assumed to take integer values.

To deal with such series, we will use the **msts class** which handles multiple seasonality time series. Then you can specify all of the frequencies that might be relevant. It is also flexible enough to handle non-integer frequencies.

You won’t necessarily want to include all of these frequencies — just the ones that are likely to be present in the data. For example, if you have 180 days of data, you can probably ignore the annual seasonality. If the data are measurements of a natural phenomenon (e.g., temperature), you might also be able to ignore the weekly seasonality.

Figure @ref(fig:calls) 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. There is a strong daily seasonal pattern with frequency 169, and a weak weekly seasonal pattern with frequency . (Call volumes on Mondays tend to be higher than the rest of the week.) If a longer series of data were available, there may also be an annual seasonal pattern.

library(fpp2)  
library(forecast)  
  
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,nrow=2)

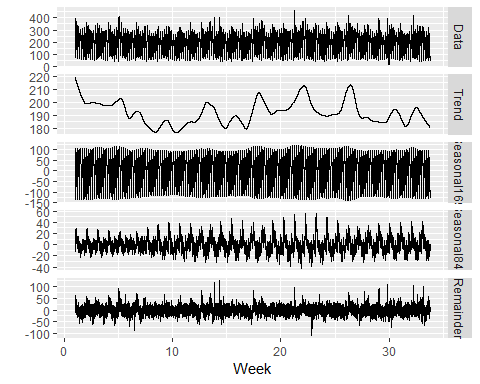


Five-minute call volume handled on weekdays between 7am and 9:05pm in a large North American commercial bank. Top panel shows data from 3 March 2003 to 23 May 2003. Bottom panel shows only the first three weeks.

### STL with multiple seasonal periods

The mstl() function is a variation of stl() designed to deal with multiple seasonality. IT will return multiple seasonal components, as well as a trend and remainder component.

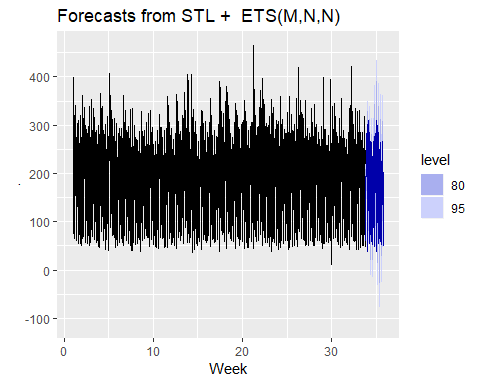
calls %>% mstl() %>%   
 autoplot()+xlab("Week")



There are two seaosonal patterns show, 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 week.

The decomposition can also be used in forecasting, with each of the seasonal components forecast using a seasonal naive 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")



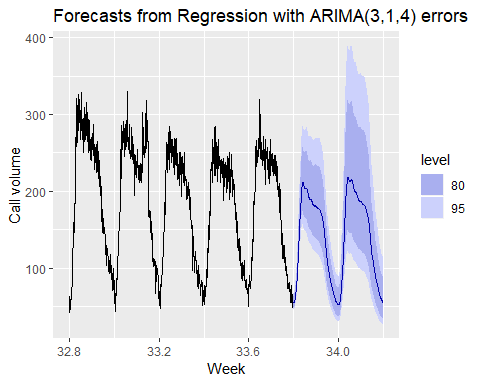
### Dynamic harmonin regression with multiple seasonal periods

With multiple seasonalities, we can use Fourier terms as we did in earlier chapters. 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

for . 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=F, 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("Week")



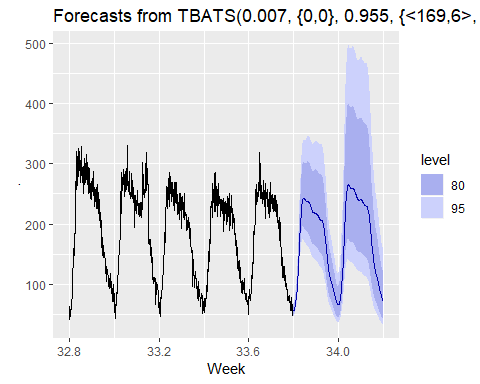
This is larger model, containing 43 parameters: 7 ARIMA coefficients, 20 Fourier coefficients for frequency 169, and 16 Fourier coefficients for frequency 845. We don’t use all the Fourier terms for frequency 845 because there is some overlap with the terms of frequency 159 (since 845=5\*169).

### 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)+  
 xlab("Call volume")+xlab("Week")



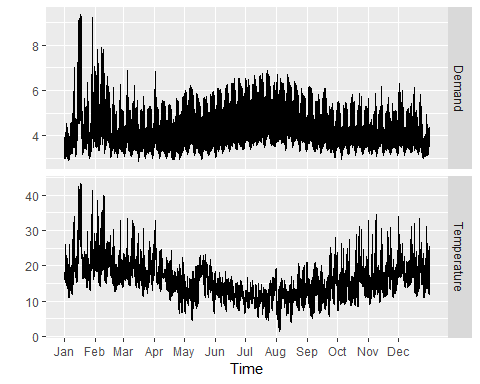
Here the prediction intervals appear to be much too wide – something that seems to happen quite often with TBATS models unfortunately.

### 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).

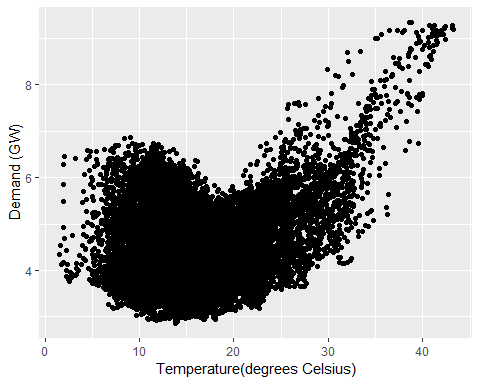
autoplot(elecdemand[,c("Demand","Temperature")],  
 facet=TRUE) +  
 scale\_x\_continuous(minor\_breaks=NULL,  
 breaks=2014+  
 cumsum(c(0,31,28,31,30,31,30,31,31,30,31,30))/365,  
 labels=month.abb) +  
 xlab("Time") + ylab("")



Half-hourly electricity demand and corresponding temperatures in 2014, Victoria, Australia.

Plotting electricity demand against temperature shows that there is 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)")

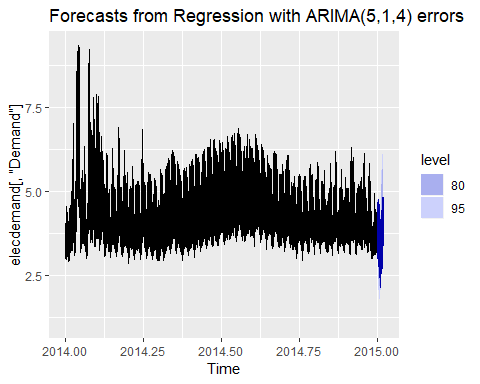


We will fit a regression model with a **piecewise linear function** of temperature (containing a knot at 18 degrees), and harmonic regression terms to allow for the daily seasonal pattern.

cooling <- pmax(elecdemand[,"Temperature"],18)  
fit <- auto.arima(elecdemand[,"Demand"],  
 xreg=cbind(fourier(elecdemand, c(10,10,0)),  
 heating=elecdemand[,"Temperature"],  
 cooling=cooling))

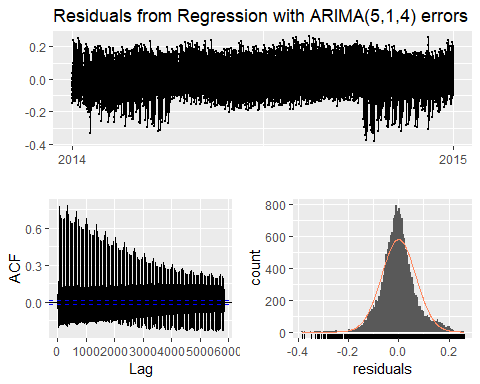
Forecasting with such models is difficult because we require future values of the predictor variables. Future values of the Fourier terms are easy to compute, but future temperatures are, of course, unknown. We could use temperature forecasts obtain from a meteorological model if we are only interested in forecasting up to a week ahead. Alternatively, we could use scenario forecasting and plug in possible temperature patterns. In the following example, we have used a repeat of the last week of temperatures to generate future possible demand values.

temps <- subset(elecdemand[,"Temperature"], start=NROW(elecdemand)-7\*48-1)  
fc <- forecast(fit, xreg=cbind(fourier(temps, c(10,10,0)),  
 heating=temps, cooling=pmax(temps,18)))  
autoplot(fc)



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)



##   
## Ljung-Box test  
##   
## data: Residuals from Regression with ARIMA(5,1,4) errors  
## Q\* = 738500, df = 3455, p-value < 2.2e-16  
##   
## Model df: 49. Total lags used: 3504

#>   
#> Ljung-Box test  
#>   
#> data: Residuals from Regression with ARIMA(5,1,4) errors  
#> Q\* = 740000, df = 3500, p-value <2e-16  
#>   
#> Model df: 49. 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. <https://robjhyndman.com/publications/complex-seasonality/>

Hyndman, R. J., & Fan, S. (2010). Density forecasting for long-term peak electricity demand. IEEE Transactions on Power Systems, 25(2), 1142–1153. <https://robjhyndman.com/publications/peak-electricity-demand/>

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. <https://robjhyndman.com/publications/stlf/>

## Vector autoregressions

Our limitation with the models we have considered so far is that they impose a unidirectiona lrelationship — the forecast variable is influenced by the predictor variables, not vice versa. However, there are many cases where the reverse should also be allowed for — where all variables affect each other. In chapter 9, the changes the changes in personal consumption expenditure () were forecast based on the changes in personal disposable income (). However, in this case a bi-directional relationship may be more suitable: an increase in will lead to an increase in 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 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 s: denotes the th observation of variable , denotes the th observation of variable , and so on.

A VAR model is a generalisation of the univariate autoregressive model for forecasting a collection of variables; that is, a vector of time series.[[8]](#footnote-200) It comprises one equation per variable considered in the system. The right hand side of each equation includes a constant and lags of all 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

where and are white noise processes that may be contemporaneously correlated. Coefficient captures the influence of the th lag of variable on itself, while coefficient captures the influence of the th lag of variable on .

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 explorer here, is that the sereies maybe **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.

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  
. Then the one-step-ahead forecasts are generated by

This is the same form as @ref(eq:var1a)–@ref(eq:var1b), except that the errors have been set to zero and parameters have been replaced with their estimates. For , the forecasts are given by

Again, this is the same form as @ref(eq:var1a)–@ref(eq:var1b), except that the errors have been set to zero, parameters have been replaced with their estimates, and the unknown values of and 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. They are, how many variables (denoted by ) and how many lags (denoted by ) should be included in the system. The number of coefficients to be estimated in a VAR is equal to (or per equation). For example, for a VAR with variables and lags, there are 16 coefficients per equation making for a total of 80 coefficients to be estimated. The more coefficients to be estimated the larger the estimation error entering the forecast.

In practice it is usual to keep small and include only variables that are correlated to each other and therefore useful in forecasting each other. Information criteria are commonly used to select the number of lags to be included.

VARs are implemented in the **vars** package in R. It contains a function VARselect to choose the number of lags 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.[[9]](#footnote-201) Care should be taken using the AIC as it tends to choose large numbers of lags. Instead, for VAR models, we prefer to use the BIC.

A criticism VARMs face is that they are **atheoretical**. They are not built on some economic theory that imposes a theoretical structure to the equations. Every variable is assumed to influence every other variable in the system, which makes direct interpretation of the estimated coefficients very difficult. Despite this, VARs are useful in several contexts.

* forecasting a collection of related variables where no explicit interpretation is required;
* testing whether one variable is useful in forecasting another (the basis of Granger causality tests);
* impulse response analysis, where the response of one variable to a sudden but temporary change in another variable is analysed;
* forecast error variance decomposition, where the proportion of the forecast variance of one variable is attributed to the effect of other variables.

### Example: A VAR model for forecasting US consumption

## Loading required package: MASS

##   
## Attaching package: 'MASS'

## The following object is masked \_by\_ '.GlobalEnv':  
##   
## cement

## The following objects are masked from 'package:fma':  
##   
## cement, housing, petrol

## The following object is masked from 'package:dplyr':  
##   
## select

## Loading required package: strucchange

## Loading required package: sandwich

##   
## Attaching package: 'strucchange'

## The following object is masked from 'package:stringr':  
##   
## boundary

## Loading required package: lmtest

VARselect(uschange[,1:2], lag.max=8, type="const")[["selection"]]

## AIC(n) HQ(n) SC(n) FPE(n)   
## 5 1 1 5

var <- VAR(uschange[,1:2], p=3, type="const")  
serial.test(var, lags.pt=10, type="PT.asymptotic")

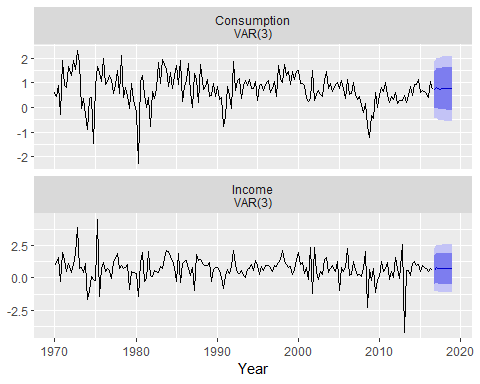
##   
## Portmanteau Test (asymptotic)  
##   
## data: Residuals of VAR object var  
## Chi-squared = 33.617, df = 28, p-value = 0.2138

summary(var)

##   
## VAR Estimation Results:  
## =========================   
## Endogenous variables: Consumption, Income   
## Deterministic variables: const   
## Sample size: 184   
## Log Likelihood: -380.155   
## Roots of the characteristic polynomial:  
## 0.7821 0.4981 0.4981 0.4441 0.2857 0.2857  
## Call:  
## VAR(y = uschange[, 1:2], p = 3, type = "const")  
##   
##   
## Estimation results for equation Consumption:   
## ============================================   
## Consumption = Consumption.l1 + Income.l1 + Consumption.l2 + Income.l2 + Consumption.l3 + Income.l3 + const   
##   
## Estimate Std. Error t value Pr(>|t|)   
## Consumption.l1 0.19100 0.07965 2.398 0.017518 \*   
## Income.l1 0.07837 0.05453 1.437 0.152445   
## Consumption.l2 0.15954 0.08252 1.933 0.054792 .   
## Income.l2 -0.02706 0.05723 -0.473 0.636831   
## Consumption.l3 0.22646 0.07970 2.841 0.005018 \*\*   
## Income.l3 -0.01454 0.05425 -0.268 0.789055   
## const 0.29081 0.08300 3.504 0.000581 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
##   
## Residual standard error: 0.5955 on 177 degrees of freedom  
## Multiple R-Squared: 0.2132, Adjusted R-squared: 0.1865   
## F-statistic: 7.994 on 6 and 177 DF, p-value: 1.216e-07   
##   
##   
## Estimation results for equation Income:   
## =======================================   
## Income = Consumption.l1 + Income.l1 + Consumption.l2 + Income.l2 + Consumption.l3 + Income.l3 + const   
##   
## Estimate Std. Error t value Pr(>|t|)   
## Consumption.l1 0.45349 0.11560 3.923 0.000125 \*\*\*  
## Income.l1 -0.27303 0.07915 -3.450 0.000702 \*\*\*  
## Consumption.l2 0.02167 0.11977 0.181 0.856663   
## Income.l2 -0.09005 0.08306 -1.084 0.279788   
## Consumption.l3 0.35377 0.11568 3.058 0.002572 \*\*   
## Income.l3 -0.05376 0.07875 -0.683 0.495699   
## const 0.38750 0.12047 3.217 0.001543 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
##   
## Residual standard error: 0.8643 on 177 degrees of freedom  
## Multiple R-Squared: 0.1755, Adjusted R-squared: 0.1475   
## F-statistic: 6.279 on 6 and 177 DF, p-value: 5.309e-06   
##   
##   
##   
## Covariance matrix of residuals:  
## Consumption Income  
## Consumption 0.3546 0.1845  
## Income 0.1845 0.7470  
##   
## Correlation matrix of residuals:  
## Consumption Income  
## Consumption 1.0000 0.3585  
## Income 0.3585 1.0000

The R output on the following page shows the lag length selected by each of the information criteria available in the **vars** package. There is a large discrepancy between a VAR(5) selected by the AIC and a VAR(1) selected by the BIC. In similar fashion to the univariate ARIMA methodology we test that the residuals are uncorrelated using a Portmanteau test [[10]](#footnote-203) The null hypothesis of no serial correlation in the residuals is rejected for both a VAR(1) and a VAR(2) and therefore we fit a VAR(3) as now the null is not rejected. The forecasts generated by the VAR(3) are plotted in Figure @ref(fig:VAR3).

forecast(var) %>%  
 autoplot() + xlab("Year")



Forecasts for US consumption and income generated from a VAR(3).

## Neural network models

Artificial neural networks are forecasint methods that are based on simple mathmatical 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” organized in layers. The predictors (or inputs) from the buttom layer, and the forecasts (or outputs) form the top later. There may be intermediate layers containing “hidden neurons”.

The very simplest networks contain no hidden layers and are equivalent to linear regression. Figure @ref(fig-10-nnet) 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 MSE. Of course, in this simple example, we can use linear regression which is a much more efficient method for training the model.

[shorten \>=1pt,-\>,draw=black!50, node distance=] =[<-,shorten \<=1pt]  
=[circle,fill=black!25,minimum size=17pt,inner sep=0pt] =[neuron,  
fill=green!50]; =[neuron, fill=red!50]; =[neuron, fill=blue!50]; = [text  
width=4em, text centered]  
  
/ in <span>1,...,4</span> (I-) at (0,-) ;  
  
\(O) ;  
  
in <span>1,...,4</span> (I-) edge (O);  
  
(input) <span>Input layer</span>; ;  
  
\@ref(fig-10-nnet)

This is known as a *multilayer feed-forward network* where each layer of nodes receives inputs from the previous layers. The outputs of 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 in Figure @ref(fig-10-nnet1) are linearly combined to give

In the hidden layer, this is then modified using a nonlinear function such as a sigmoid,

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 and are “learned” from the data. The values of the weights are often restricted to prevent them 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, threre 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 numer of hidden layers and the number of nodes in each hidden layer, must be specified in advance. We will consider how these can be chosen using cross-validation later in this chapter.

### 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 (chap [8](https://otexts.org/fpp2/arima.html#arima)). We call this a **neural network autoregression** or **NNAR** model.

In this book, we only consider feed-foward networks with hidden layer, and we use the notation NNAR(,) to indicate there are lagged inputs and nodes in the hidden layer. For example, a NNAR(9,5) model is a neural network with the alst nine observations () used as inputs to forecast the output and with five neurons in the hidden layer. A NNAR() model is equivalent to an ARIMA() 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) model has inputs , , and , and two neurons in the hidden layer. More generally, an NNAR() model has inputs and neurons in the hidden layer. A NNAR() model is equivalent to an ARIMA()(,0,0) model but without the restrictions on the parameters to ensure stationarity.

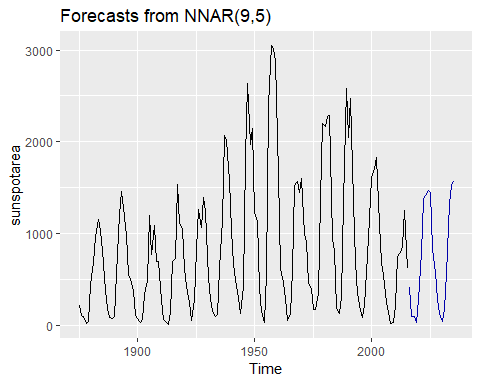
The nnetar() function fits an NNAR() model. If the values of and are not specified, they are automatically selected. For non-seasonal time series, the default is the optimal number of lags (according to the AIC) for a linear AR() model. For seasonal time series, the default values are and is chosen from the optimal linear model fitted to the seasonally adjusted data. If is not specified, it is set to (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 containing 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, forecats from NNAR(10,6) are shown for the next 30 years. We have set a Box-Cox transformation with lambda=0 to ensure the forecats stay positive.

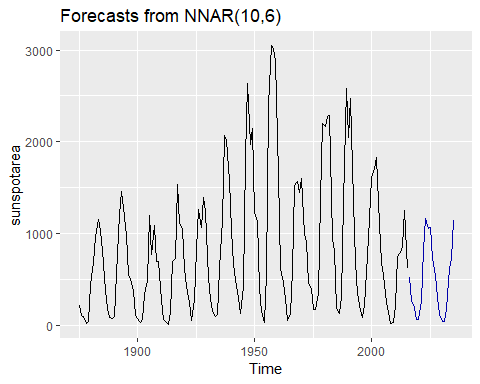
library(forecast)  
library(fpp2)  
  
fit <- nnetar(sunspotarea)  
autoplot(forecast(fit,h=20))



Forecasts from a neural network with nine lagged inputs and one hidden layer containing five neurons.

The forecasts actually go slightly negative, which is of course impossible. If we wanted to restrict the forecasts to remain positive, we could use a log transformation (specified by the Box-Cox parameter ):

fit <- nnetar(sunspotarea, lambda=0)  
autoplot(forecast(fit,h=20))



### 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 do it using simulation where future sample paths are generated using bootstrapped residuals.

Suppose we fit a NNETAR model to the famous Canadian lynx data:

(fit <- nnetar(lynx, lambda=0.5))

## Series: lynx   
## Model: NNAR(8,4)   
## Call: nnetar(y = lynx, lambda = 0.5)  
##   
## Average of 20 networks, each of which is  
## a 8-4-1 network with 41 weights  
## options were - linear output units   
##   
## sigma^2 estimated as 92.08

We have used a Box-Cox transformation with to ensure the residuals will be roughly homos The model can be written as

where is a vector containing lagged values of the series, and is a neural network with 4 hidden nodes in a single layer.

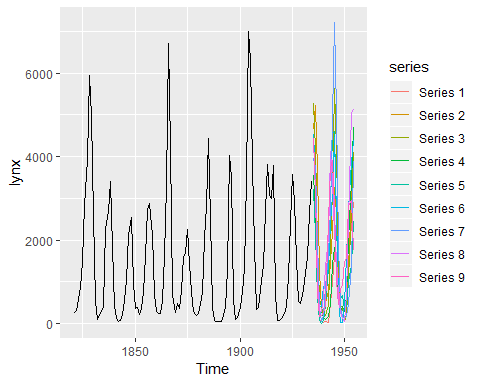
The error series is assumed to be homoscedastic (and possibly also normally distributed).

We can simulate future sample paths of this model iteratively, by randomly generating a value for , either from a normal distribution, or by resampling from the historical values. So if is a random draw from the distribution of errors at time , then

is one possible draw from the forecast distribution for . Setting , we can then repeat the process to get

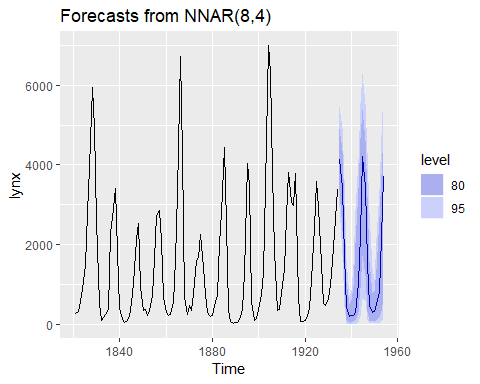
In this way, we can iteratively simulate a future sample path. By repeatedly simulating sample paths, we build up knowledge of the distribution for all future values based on the fitted neural network. Here is a simulation of 9 possible future sample paths for the lynx data. Each sample path covers the next 20 years after the observed data.

sim <- ts(matrix(0, nrow=20, ncol=9), start=end(lynx)[1]+1)  
for(i in seq(9))  
 sim[,i] <- simulate(fit, nsim=20)  
  
autoplot(lynx) + forecast::autolayer(sim)



If we do this a few hundred or thousand times, we can get a very good picture of the forecast distributions. This is how the forecast.nnetar function produces prediction intervals:

fcast <- forecast(fit, PI=TRUE, h=20)  
autoplot(fcast)



Because it is a little slow, PI=FALSE is the default, so prediction intervals are not computed unless requested. The npaths argument in forecast.nnetar 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).

## Bootstrapping and baggin

### Bootstrapping time series

In the preceeding section, and in section [3.5](https://otexts.org/fpp2/prediction-intervals.html#prediction-intervals), we bootstrap the residuals of a time series in order to simulate future values of a series using a model.

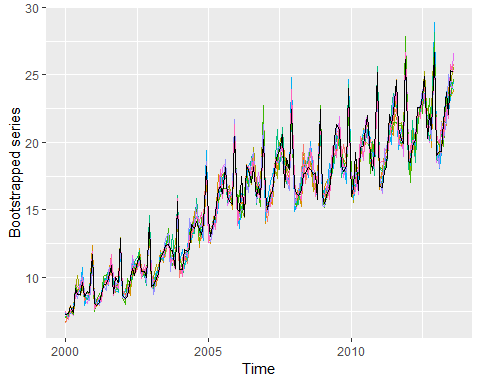
More generally, we can generate new time series that are similar to our observed series, using another type of bootstrap.

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 remiander components to get bootstrapped remainder time 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](https://otexts.org/fpp2/prediction-intervals.html#prediction-intervals). 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,col=T)+  
 autolayer(debitcards,col=F)+  
 ylab("Bootstrapped series")+guides(col="none")

## For a multivariate timeseries, specify a seriesname for each timeseries. Defaulting to column names.



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 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 generation 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 got 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 <- 100L  
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.

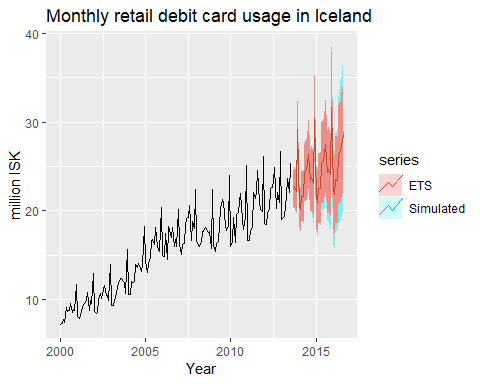
h <- 36L  
future <- matrix(0, nrow=nsim, ncol=h)  
for(i in seq(nsim))  
 future[i,] <- simulate(ets(sim[[i]]), nsim=h)

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")



### Bagged ETS forecasts

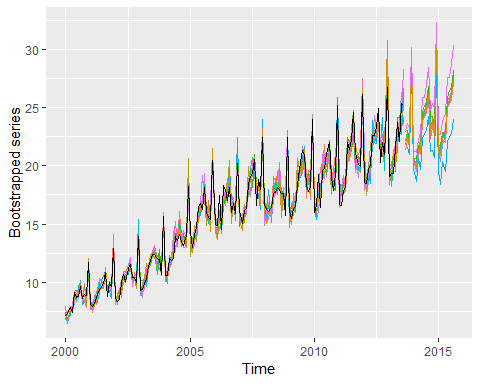
Another use for these bootstrap time series is to improve forecast accuracy. If we produce forecasts from each of the additional time series, and average the result forecasts, we get better forecasts than if we simply forecast the original time series directly. This is called **“bagging”** wich stands for “**b**ootstrap **agg**regatin**g**”.

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.

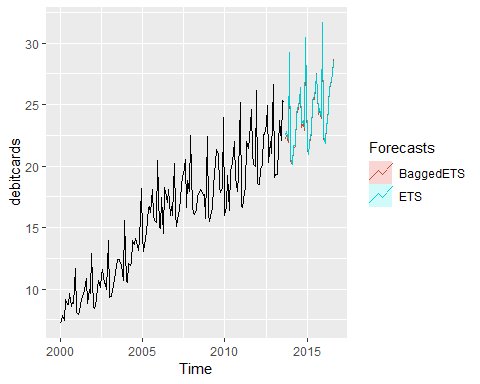
sim <- bld.mbb.bootstrap(debitcards, 10) %>%  
 as.data.frame() %>%  
 ts(frequency=12, start=2000)  
fc <- purrr::map(as.list(sim),  
 function(x){forecast(ets(x))[["mean"]]}) %>%  
 as.data.frame() %>%  
 ts(frequency=12, start=start)  
autoplot(debitcards) +  
 autolayer(sim, colour=TRUE) +  
 autolayer(fc, colour=TRUE) +  
 autolayer(debitcards, colour=FALSE) +  
 ylab("Bootstrapped series") +  
 guides(colour="none")

## For a multivariate timeseries, specify a seriesname for each timeseries. Defaulting to column names.  
## For a multivariate timeseries, specify a seriesname for each timeseries. Defaulting to column names.



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"))



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.

## Exercises

1. Use the tbats function to model your retail time series.
2. Check the residuals and produce forecasts.
3. Does this completely automated approach work for these data?
4. Have you saved any degrees of freedom by using Fourier terms rather than seasonal differencing?
5. Consider the weekly data on US finished motor gasoline products supplied (thousands of barrels per day) (series gasoline):
6. Fit a tbats model to these data.
7. Check the residuals and produce forecasts.
8. Could you model these data using any of the other methods we have considered in this book?
9. Experiment with using nnetar() on your retail data and other data we have considered in previous chapters.

## Bibliography

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. <https://robjhyndman.com/publications/hksg/>

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. <https://robjhyndman.com/publications/bagging-ets/>

## Further reading

e 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 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. <https://robjhyndman.com/publications/complex-seasonality/>

Pfaff, B. (2008). Analysis of integrated and cointegrated time series with R. New York, USA: Springer Science & Business Media. [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. <https://doi.org/10.1016/j.ijforecast.2011.04.001>

Lahiri, S. N. (2003). Resampling methods for dependent data. New York, USA: Springer Science & Business Media. [Amazon]

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. <https://robjhyndman.com/publications/bagging-ets/>

# 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](https://robjhyndman.com/hyndsight/).

## Weekly, daily and sub-daily data

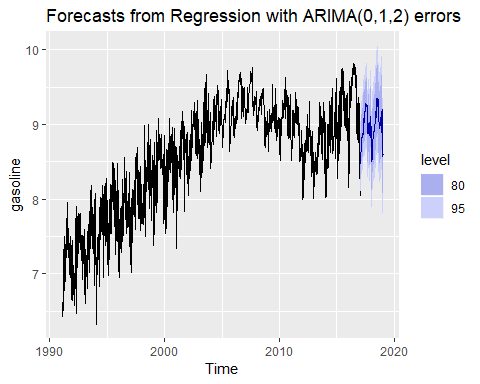
Weekly, daily and sub-daily data can be challenging for forecasting, although for different reasons.

### Weekly data

Weekly data is difficult 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 a dynamic harmonic regression model, as discussed in Section @ref(sec-dhr). Here is an example using weekly data on US finished motor gasoline products supplied (in thousands of barrels per day) from February 1991 to May 2005. The number of Fourier terms was selected by minimizing the AICc. The order of the ARIMA model is also selected by minimizing the AICc, although that is done within the auto.arima() function.

library(forecast)  
library(fpp2)  
  
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)



The fitted model has 13 pairs of Fourier terms and can be written as

where is an Regression with ARIMA(0,1,2) errors process. Because is non-stationary, the model is actually estimated on the differences of the variables on both sides of this equation. There are 26 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 29 (the other three coming from the 2 MA parameters and the drift parameter).

An alternative approach is the TBATS model introduced in Section @ref(sec:complexseasonality). This was the subject of [Exercise 11.2](sec-ex-11.html). In this example, the forecasts are almost identical and there is little to differentiate the two models. The TBATS model is preferable when the seasonality changes over time. The ARIMA 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, than it will be possible to use one of the single-seasonal methods we have discussed (e.g., ETS or seasonal ARIMA). But when the time series is long enough so that some of the longer seasonal periods become apparent, it will be necessary to use dynamic harmonic regression or TBATS, as discussed in Section @ref(sec:complexseasonality).

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 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).

## Time series of counts

All of the methods discussed in this book assume that the data have a continuous sample space. But very 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 and the discrete sample space has no perceivable effect on our forecasts. However, if our data contains small counts , 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 @Croston72. 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 be the th non-zero quantity, and let be the time between and . Croston’s method involves separate simple exponential smoothing forecasts on the two new series and . Because the method is usually applied to time series of demand for items, is often called the “demand” and the “inter-arrival time”.

If and are the one-step forecasts of the th demand and inter-arrival time respectively, based on data up to demand , then Croston’s method gives

The smoothing parameter takes values between 0 and 1 and is assumed to be the same for both equations. Let be the time for the last observed positive observation. Then the -step ahead forecast for the demand at time , is given by the ratio

There are no algebraic results allowing us to compute prediction intervals for this method, because the method does not correspond to any statistical model [@SH05].

The croston function produces forecasts using Croston’s method. In the following example, we apply the method to monthly sales of a lubricant that is rarely used. This data set was part of a consulting project that one of us did for an oil company several years ago.

The data contain small counts, with many months registering no sales at all, and only small numbers of items sold in other months.

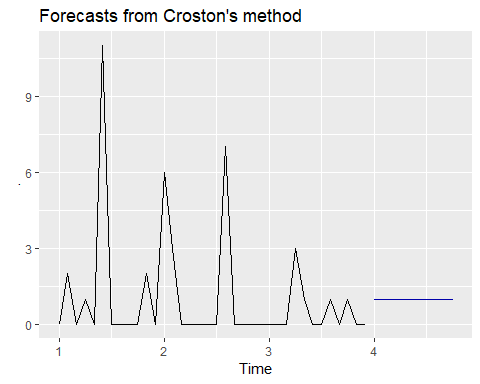
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec |
| 0 | 2 | 0 | 1 | 0 | 11 | 0 | 0 | 0 | 0 | 2 | 0 |
| 6 | 3 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 3 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

The demand and arrival series are computed from the above data.

|  |  |  |
| --- | --- | --- |
| i | q | a |
| 1 | 2 | 2 |
| 2 | 1 | 2 |
| 3 | 11 | 2 |
| 4 | 2 | 5 |
| 5 | 6 | 2 |
| 6 | 3 | 1 |
| 7 | 7 | 6 |
| 8 | 3 | 8 |
| 9 | 1 | 1 |
| 10 | 1 | 3 |
| 11 | 1 | 2 |

The croston function simply uses by default, and 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. This gives the demand forecast 2.750 and the arrival forecast 2.793. So the forecast of the original series is . 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](https://cran.r-project.org/package=tsintermittent) package for R.

Forecasting models that deal more directly with the count nature of the data are described in @christou2015count.

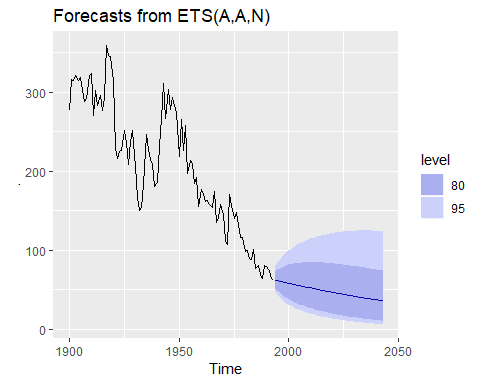
## Ensuring forecasts stay within limits

It is common to want forecasts to be positive, or to require them to be within some specified range . 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 . 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()



Because we set biasadj=TRUE, the forecasts are the means of the forecast distributions.

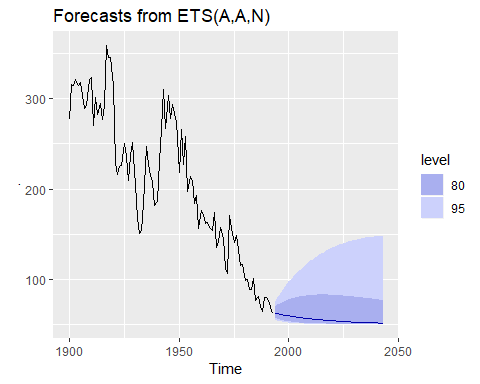
### Forecasts constrained to an interval

To see how to handle data constrained to an interval, imagine that the egg prices were constrained to line within and . Then we can transform the data using a scaled logit transform which maps to the whole real line:

where is on the original scale and is the transformed data. To reverse the transformation, we will use

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)



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.

## 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 [@BatesGranger1969], showing that combining forecasts often leads to better forecast accuracy. Twenty years later, @Clemen89 wrote

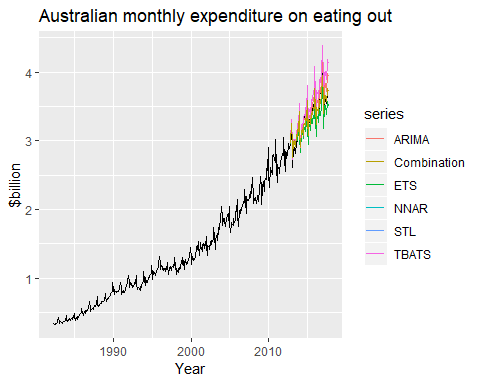
The results have been virtually unanimous: combining multiple forecasts leads to increased forecast accuracy. in many cases one can make dramatic performance improvements by simply averaging the forecasts.

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 forecast 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 = T),h=h)  
STL <- stlf(train, lambda=0,h=h, biasadj=T)  
NNAR <- forecast(nnetar(train),h=h)  
TBATS <- forecast(tbats(train),h=h)  
Combination <- (ETS$mean+ARIMA$mean+STL$mean+NNAR$mean+TBATS$mean)/5

autoplot(auscafe)+  
 forecast::autolayer(ETS$mean, series="ETS")+  
 autolayer(ARIMA$mean, series="ARIMA")+  
 autolayer(STL$mean, series="STL")+  
 autolayer(STL$mean, series = "NNAR")+  
 autolayer(STL$mean, series = "TBATS")+  
 autolayer(Combination, series="Combination")+  
 xlab("Year")+ylab("$billion")+  
 ggtitle("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 Combination   
## 0.13699696 0.12146220 0.21446157 0.30926778 0.08840889 0.07162870

TBATS does particularly well with this time series data, but the combination approach is not far behind. For other data, TBATS may be quite poor, while the combination approach is almost always close to, or better than, the best component method.

## Prediction intervals for aggregates

A common problem is to forecast the aggregate of several time per periods of data, using a model fitted to the disaggregated data. For example, you may have monthly data but wish to frecast the total for the next year. Or you may have monthly data but wish 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 forecasts the aggregate gas demaind 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 very close to the sum of the individual forecasts:

sum(fc$mean[1:6])

## [1] 281.8006

meanagg

## [1] 281.8744

Ptrfiction intervals are also easy to obtain:

# 80% interval:  
quantile(sim, prob=c(0.1, 0.9))

## 10% 90%   
## 263.2250 300.9481

# 95% interval:   
quantile(sim, prob=c(0.025, 0.975))

## 2.5% 97.5%   
## 253.6763 311.3678

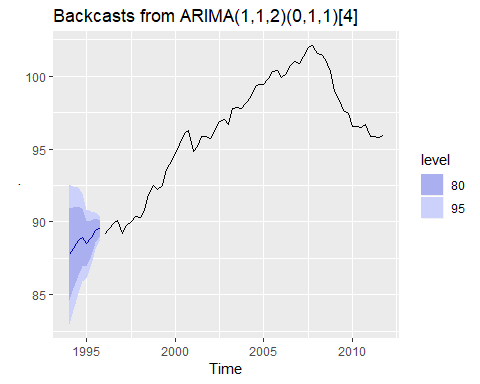
## Backcasting

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 very 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 function to backcast any time series. Here is an example applied to quarterly retail trade in the Eoro area. The data are from 1996-2011. 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))



## 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 you need more observations than there are parameters in your forecasting model. However, in practice, you 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)

## Warning: package 'Mcomp' was built under R version 3.5.1

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()

## Warning in value[[3L]](cond): The chosen test encountered an error, so no  
## seasonal differencing is selected. Check the time series data.

## .  
## 0 1 2 3   
## 54 73 15 2

### 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.

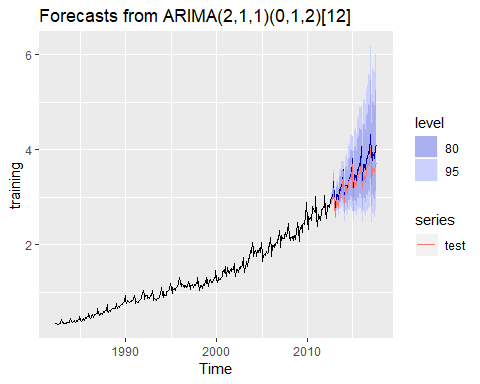
## Forecasting on training and test sets

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 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.

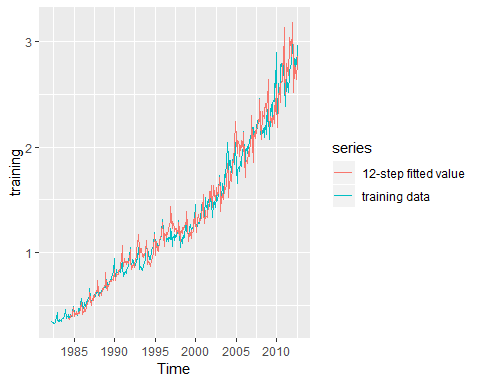
training <- subset(auscafe, end=length(auscafe)-61)  
test <- subset(auscafe, start=length(auscafe)-60)  
  
cafe.train <- Arima(training, order=c(2,1,1),  
 seasonal=c(0,1,2),  
 lambda=0)  
  
cafe.train %>%   
 forecast(h=60) %>%   
 autoplot()+autolayer(test)



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 value")

## Warning: Removed 25 rows containing missing values (geom\_path).



### One-step forecats 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 We estimate the model on these data, but then compute . 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  
## Training set -0.002622226 0.04591291 0.03412845 -0.07300775 1.002202  
## MASE ACF1  
## Training set 0.1898996 -0.05704137

Note that the second call to Arima does not involve the model being re-estimated. Instead, the model obtained in the first call is applied to the test data in the second call.

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”).

END

## 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.

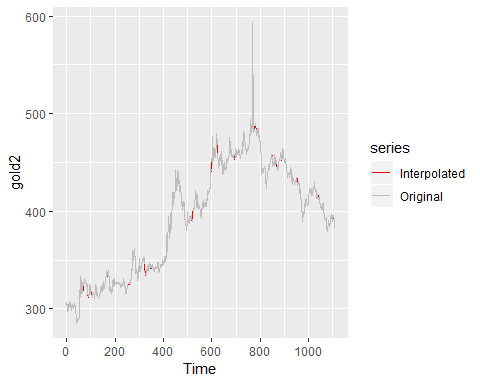
In other situations, the missingness may be essentially random. For example, someone may have forgotten to record the sales figures, or the data recording device may have malfunctioned. If the timing of the missing data is not informative for the forecasting problem, then the missing values can be handled 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"))



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 estimate the seasonally component, and the seasonally adjusted series are linear interpolated. More sophisticated missing value interpolation is provided in the [imputeTS package](https://cran.r-project.org/web/packages/imputeTS/index.html).

### 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)

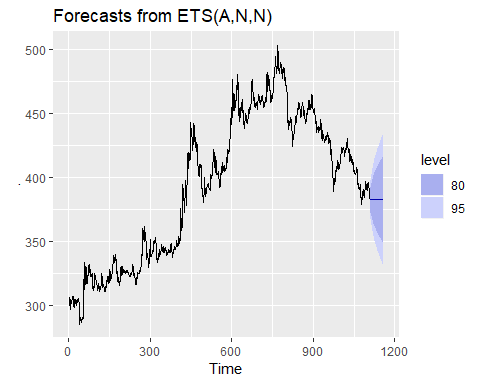
## $index  
## [1] 770  
##   
## $replacements  
## [1] 494.9

gold[768:772]

## [1] 495.00 502.75 593.70 487.05 487.75

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()



Notice that the outlier and missing values have been replaced with estimates.

## 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.

<https://otexts.org/fpp2/appendix-using-r.html>

1. arc cos is the inverse cosine function. You should be able to find it on your calculator. It may be labelled acos or cos. [↑](#footnote-ref-49)
2. As already pointed out, comparing information criteria is only valid for ARIMA models of the same orders of differencing. [↑](#footnote-ref-89)
3. Forecasting with cointegrated models is discussed by @Harris03. [↑](#footnote-ref-105)
4. Australia comprises eight geographical areas six states and two territories: Australian Capital Territory, New South Wales, Northern Terrirory, Queensland, South Australia, Tasmania, Victoria, Western Australia. In this example we consider all eight. [↑](#footnote-ref-145)
5. denotes the transpose of . [↑](#footnote-ref-151)
6. All forecasting approaches used in this textbook generate unbiased forecasts. [↑](#footnote-ref-162)
7. Note that is a proportionality constant. It does not need to be estimated or specified here as it gets cancelled out in estimating in @ref(eq:MinT). We include it here for completeness. [↑](#footnote-ref-163)
8. 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 @AthEtAl2012. [↑](#footnote-ref-200)
9. For a detailed comparison of these criteria, see Chapter 4.3 of @Lut2005. [↑](#footnote-ref-201)
10. The tests for serial correlation in the “vars” package are multivariate generalisations of the tests presented in Section @ref(residuals). [↑](#footnote-ref-203)