ARIMA modelling in R

Nikolaos Kourentzes (nikolaos@kourentes.com)

Contents

1 Data and packages

We will use two example time series, one from base R and one from the *forecast* package. There are a few different alternatives for modelling ARIMA with R. The first is the `arima` function that comes built-in with R. This function provides the core for doing manual modelling with ARIMA; but requires the modeller to identify the appropriate model. More advanced alternatives exist in the *forecast* and *smooth* packages. The first extends the `arima` function to include seasonality and automatic or semi-automatic model identification. The second offers a new implementation from scratch while providing both automatic and detailed manual modelling.

Let us load the *forecast* package.

```
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

It is important to remember that the data need to be presented as time series to the functions. You can check is something is a time series using the function `class`. For example:

```
class(AirPassengers)
```

```
## [1] "ts"
```

Observe that the result is `ts`, which is a shorthand for time series. These objects contain additional information about the sampling frequency of the data and the start/end period. Observe:

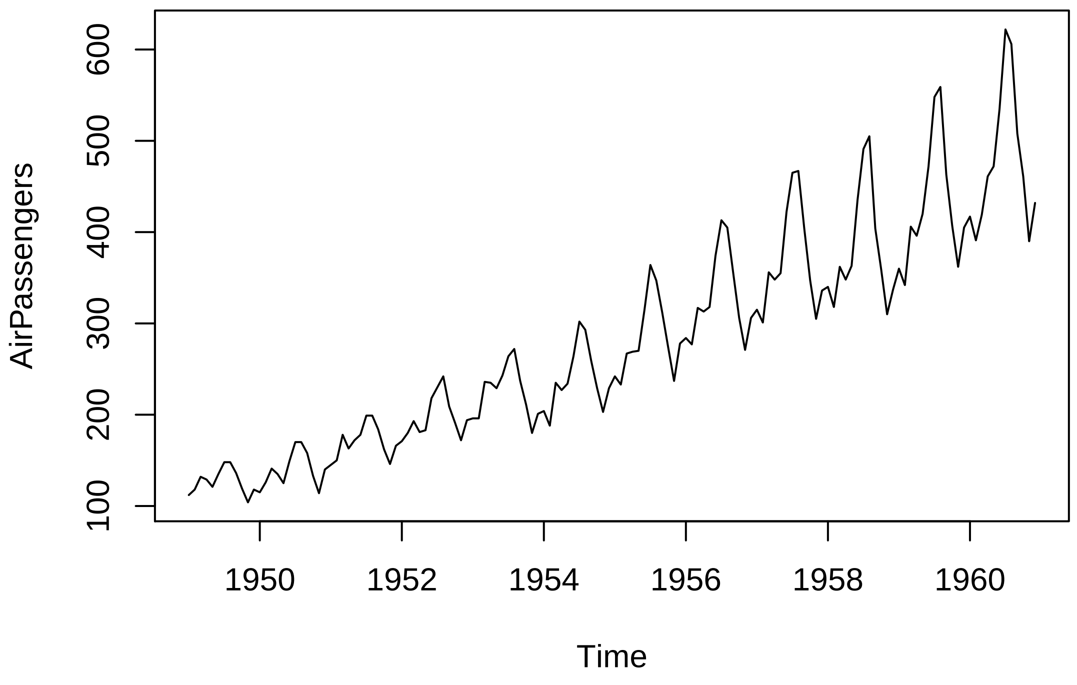
```
AirPassengers
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

2 Data exploration

We will use the `AirPassengers` time series that comes with R. It is always a good idea to first plot the time series!

```
plot(AirPassengers)
```



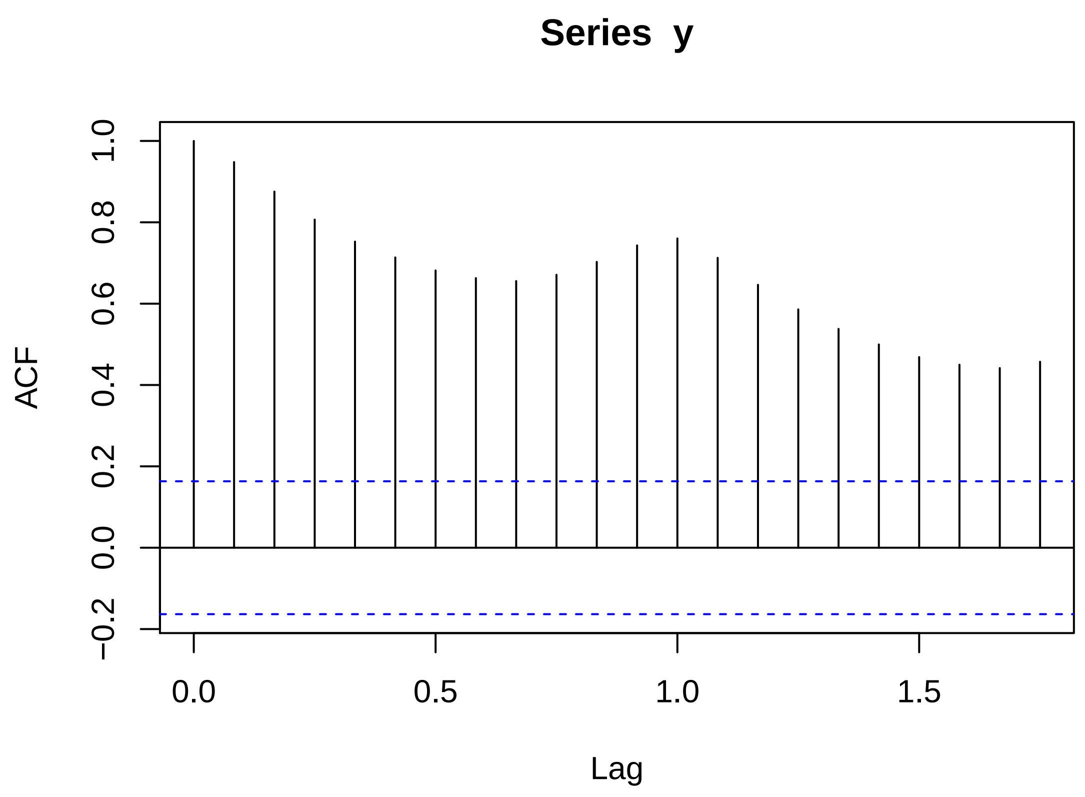
We note that it is dominated by a seasonal pattern and an increasing trend.

To avoid typing the name of the times series all the time, but also to allow you to run the same analysis with different data, we will store the series in a variable called `y`.

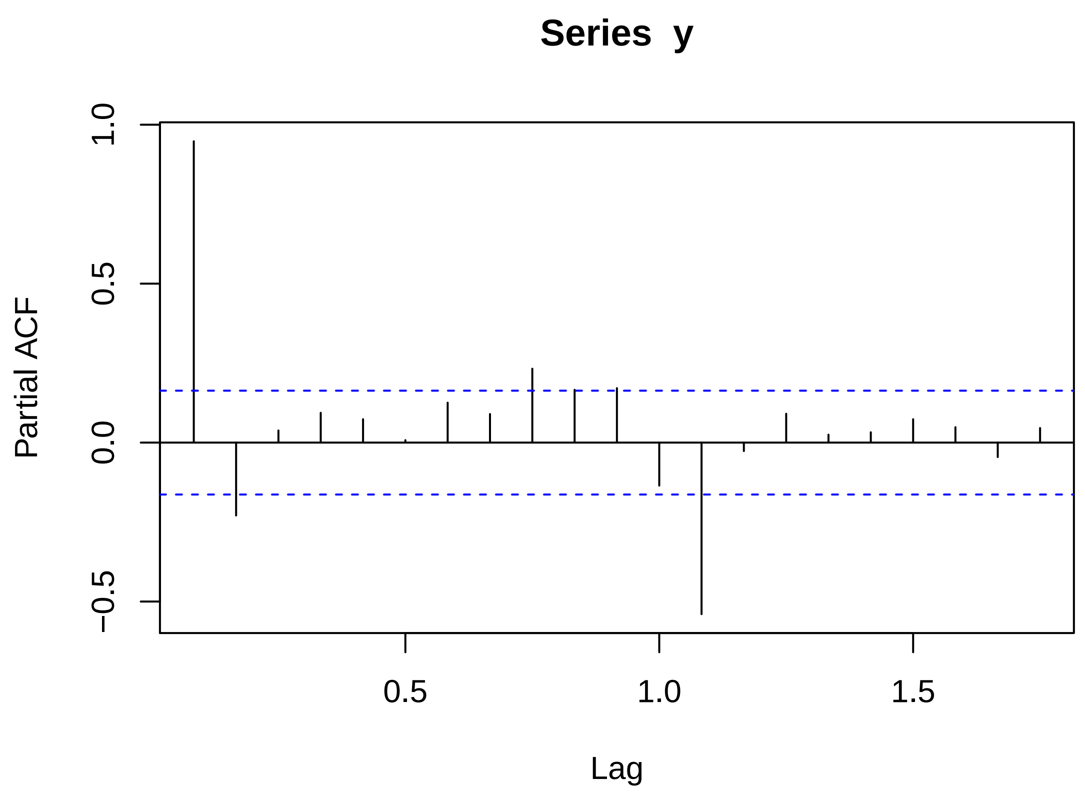
```
y <- AirPassengers
```

To get the ACF and the PACF plots we can use the functions `acf`and `pacf` respectively.

```
acf(y)
```


Series y
ACF
Lag
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.5
1.0
1.5

```r
pacf(y)
```


Series y
Partial ACF
Lag
1.0
0.5
0.0
−0.5
0.5
1.0
1.5

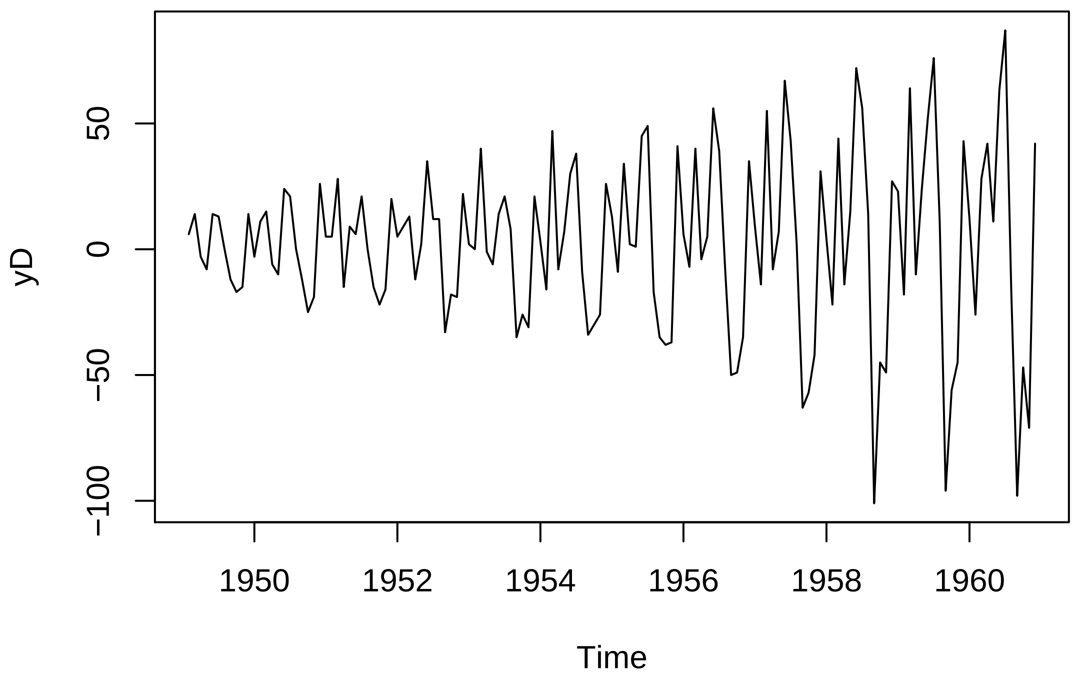
I do not want to spend much time on these plots. Even though many textbooks insist that these are somehow useful, as we demonstrated earlier in the lecture, they quickly become useless for anything more complex than clean AR or MA processes of low order, for e.g., AR(1) or AR(2). Even the original inventors of ACF and PACF have stepped away from these now that we have increased computational and statistical power!

A more useful function is `diff` that helps us introduce differencing to remove non-stationarities. These may appear as a lack of a global mean, for instance, a stochastic trend, or a stochastic seasonality. The term stochastic suggests that the process is driven by the error term of the model, which itself is a random variable. Practically, we can explain stochastic as a trend or a seasonality that is evolving over time, in contrast to deterministic that remains constant. Observe the plot of the time series before. The seasonality evolves over time.

> Note that here we avoid going into the technicalities of stationary time series, but it is important to note that we should not mix the existence of trend or seasonality with the definition of stationarity. It just happens that, for instance, the trend is also non-stationary. The inverse does not always hold! This is because the term trend is not uniquely defined across modelling classes.

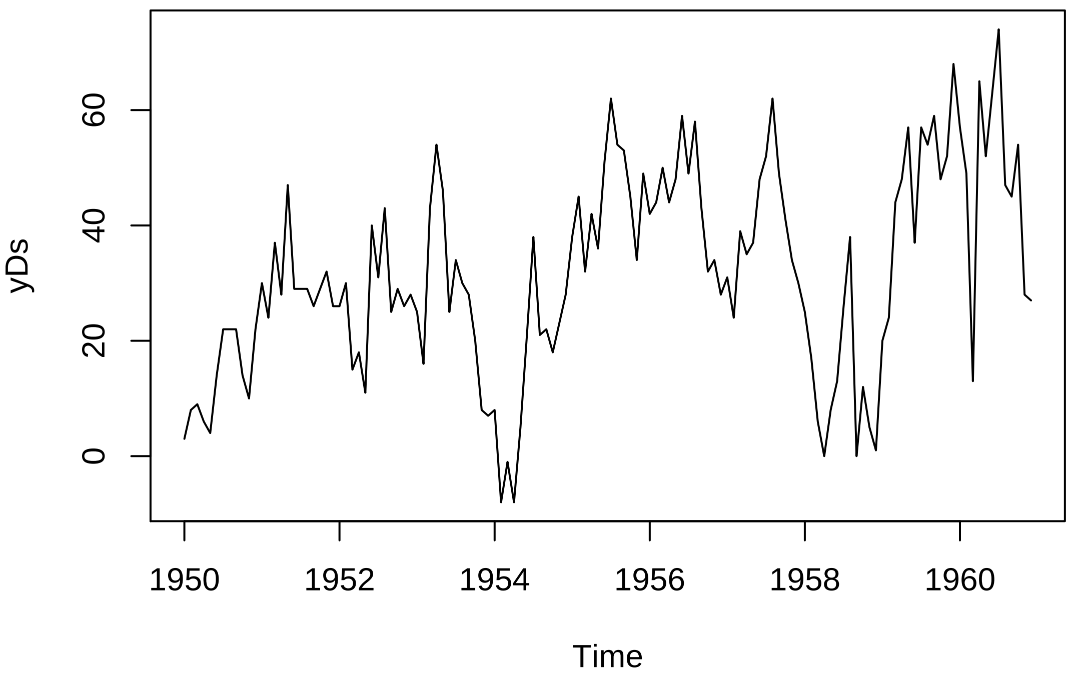
Back to our example! Differencing can be in levels or seasonal. Observe the effect:

```r
yD <- diff(y) # Differencing in levels, also called first differencing
plot(yD)
```



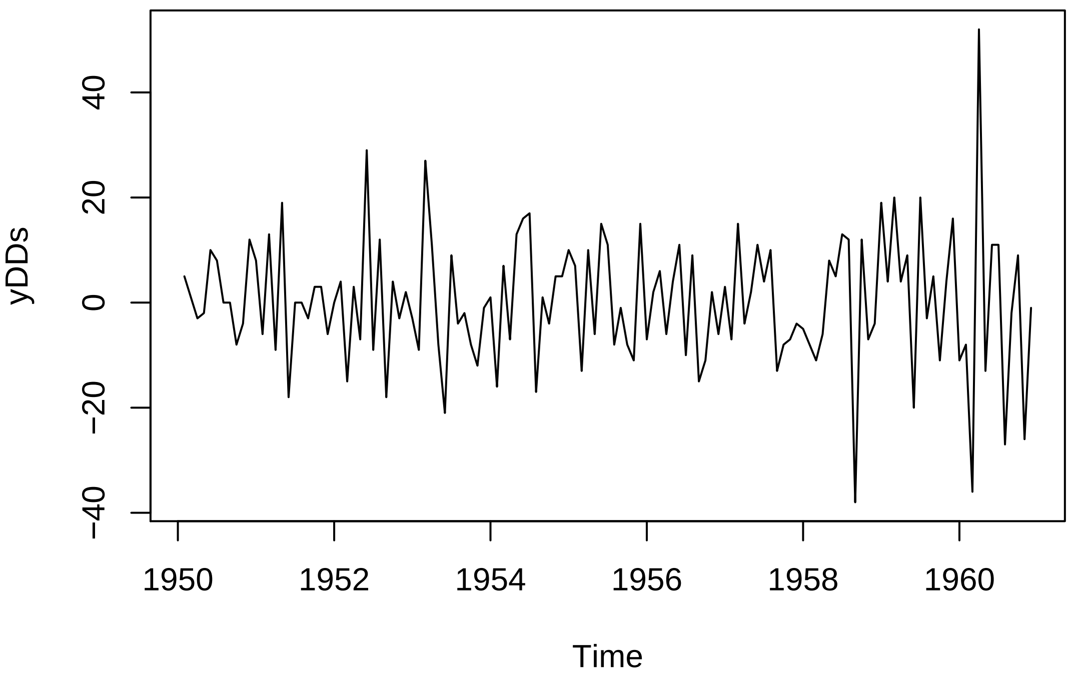
The seasonal shape is still visible, but the trend is removed, hence the term differences in level. Intuitively we are now modelling the rate of change of the time series.

```r
# Seasonal differencing. This is a monthly time series,
# and 12 corresponds to the seasonal periodicity.
yDs <- diff(y,12)
plot(yDs)
```



Now the seasonal component is removed and the remaining information appears to follow an erratic (read as stochastic!) trend. We can of course apply both sets of differencing.

```
yDDs <- diff(diff(y,12))
plot(yDDs)
```



Which seems to be free of both seasonal and trend elements. Now the series is stationary and we can look for AR or MA components relatively easy.

> Plot the ACF and PACF for different versions of differencing and observe how the plots become more informative as the time series becomes stationary. Consider the following, if a time series has a trend then the previous observation is highly correlated with the current (co-moving to the direction of the trend), so does with the observation two-periods ago and so on. This contaminates the ACF and PACF making them harder to read and use.

We can apply multiple times either level or seasonal differencing. This is to deal with higher *order* of non-stationary. This is rarely needed in practice, and even if it is we very rarely consider more than a second set of differencing. Intuitively that would be the same as modelling the rate of change of the rate of change. For instance, this may appear when there are apparent non-linearities in how the trend shifts over time, i.e., its rate of change is itself non-stationary! For 99.99% of the cases you will be okay with trying a single level and/or seasonal difference.

If you are finding that both seasonality and trend may need differencing, it is good practice to first consider seasonal differences, as these may be enough to deal with the trend as well.

Finally, note that the differencing is the integration (I) term in ARIMA.

3 Building an ARIMA model

3.1 Differencing

In most cases building an ARIMA can be seen as a two-step process. First, we identify the order of differencing and make our time series stationary, and second, we identify the appropriate order of the AR and MA terms. For the first step, we rely on time series exploration (or statistical tests if we need to automate). For the second step, we can use information criteria, such as the Akaike Information Criterion (AIC).

In R there are functions that implement many of the popular statistical tests to identify the order of differencing. For seasonal differences a helpful function is `nsdiffs`. The function implements four common statistical tests (type

?nsdiffs to see the details in the help file), and also tries for higher order of differencing. It will return a numerical value which is what should be the order of seasonal differencing, i.e., SARIMA(x,x,x)(x,this number,x).

```
nsdiffs(y)
```

```
## [1] 1
```

Here the test suggests we should apply seasonal differences of order 1.

Similarly, for level differences we can use the function `ndiffs` that works in a similar fashion. We test of level differences after applying any seasonal differences. We have already calculated `yDs`, so we will use that.

```
ndiffs(yDs)
```

```
## [1] 1
```

Again we first the first order level differences are sufficient. That would be in an ARIMA(x,this number,x) or in a SARIMA(x,this number,x)(x,x,x). As it happens we tried these options already, so the variable `yDDs` is what we need, which we had already plotted and found to appear stationary.

If any of the tests would give you a number higher than 1, then you would just apply the differencing operator again.

3.2 Modelling of the ARMA parts

Once we have made our time series stationary, we can proceed with the main part of the modelling. First, we should get familiar with the functions. We will use the `Arima` function from the *forecast* package. It is worthwhile to see all the options that it offers by typing ´?Arima´. The three first arguments are the important ones: (i) the time series; (ii) the ARIMA orders, as AR, I, MA; and (iii) the seasonal ARIMA orders, again in the same order. If the second or the third term are left blank, then they are assumed to be (0,0,0).

For example, we fit an ARMA(1,0) to our stationary series. Observe that I dropped the differencing part, as I have already made my time series stationary.

```
Arima(yDDs,order=c(1,0,0))
```

```
## Series: yDDs
## ARIMA(1,0,0) with non-zero mean
##
## Coefficients:
##           ar1    mean
##       -0.3079  0.1767
## s.e.   0.0828  0.7832
##
## sigma^2 estimated as 139.1:  log likelihood=-508.17
## AIC=1022.34   AICc=1022.53   BIC=1030.97
```

I could also ask the function to do that for me. This is better, as it greatly simplifies the creation of forecast, so that I do not have to reverse the differencing manually. Observe that the results are the same if I type:

```
Arima(y,order=c(1,1,0),seasonal=c(0,1,0))
```

```
## Series: y
## ARIMA(1,1,0)(0,1,0)[12]
##
## Coefficients:
##           ar1
##       -0.3076
## s.e.   0.0828
##
## sigma^2 estimated as 138.1:  log likelihood=-508.2
## AIC=1020.39   AICc=1020.49   BIC=1026.14
```

I used series `y` and include the differencing in both parts of the SARIMA. We will always use `y` directly from now on. At this point, I can try many different models to my data, with various alternatives for AR and MA, seasonal or not.

I need some way to quickly compare the models. To do this we can rely on information criteria, such as the AICc. The AICc is a “corrected” form of AIC for small sample sizes, a common problem in many forecasting applications. For long time series AICc and AIC are almost identical.

First, we can produce a few models that we can compare. I will rely on lists and for-loops to do this quickly. If you are unfamiliar with lists, these are very flexible memory objects in R that you can use to store any type of information.

```r
# Initialise a variable to store the results
models <- list()
# For now, I will not deal with the seasonal part.
# I will instead try to find models with AR from 0 to 3,
# and MA from 0 to 3. To do this I will use two loops,
# one for the AR and one for the MA.
AR <- MA <- 0:3
k <- 1 # A simple index to help me go through the list

# The loops start here
for (i in 1:length(AR)){ # length(AR) gives me the size of AR variable
  for (j in 1:length(MA)){
    models[[k]] <- Arima(y,order=c(i,1,j),seasonal=c(0,1,0))
    # Note that with lists we use double square brackets [[ ]]
    k <- k + 1 # This simple index makes my life easy with the list
  }
}
```

This should produce 16 models (4 AR options times 4 MA options). We can inspect any model we want by calling it from the list. For example:

```r
models[[10]]
```

```
## Series: y
## ARIMA(3,1,2)(0,1,0)[12]
##
## Coefficients:
##           ar1     ar2     ar3     ma1      ma2
##       -0.3696  0.7516  0.2595  0.0176  -0.9824
## s.e.   0.0878  0.0791  0.0888  0.0464   0.0459
##
## sigma^2 estimated as 128.2:  log likelihood=-502.87
## AIC=1017.74   AICc=1018.42   BIC=1034.99
```

As you can see we get all the coefficients (and their estimation errors), as well as various information criteria. In principle we could use the coefficients and their standard error and calculate p-values for these and test for significance, potentially simplifying the model. This has been one standard approach for model identification with ARIMA (or regression more generally), but it has quite a few caveats. It is largely considered to be superseded by using information criteria instead. Note that information criteria and p-values do not play well together. Information criteria may prefer a model that has insignificant terms, and vice versa. There is good theory in favour of information criteria, so we will do that. This is also one of the reasons that R does not output p-values readily!

The AICc values are stored in the models, so we can ask for that information:

```r
models[[10]]$aicc
```

```
## [1] 1018.416
```

If you compare with the output above, and you will verify that the value is indeed the reported AICc. In case you are wondering what else is stored in that list, you can use `names(models[[10]])`. You will see that you can extract most elements that make up the model.

We used lists so that we can get all this information with as few lines of code as possible, so to get all AICc’s we can just ask:

```
AICc <- sapply(models,function(x){x$aicc})
AICc
```

```
##  [1] 1022.583 1024.478 1019.733 1017.887 1018.165 1019.771 1020.474 1022.604
##  [9] 1019.565 1018.416 1016.491 1017.963 1024.624 1023.342 1017.872 1019.727
```

The function `sapply` goes through all elements of the list and applies the function that follows. There I just ask it to get the AICc value. We have a vector of AICc that we can compare to find the best model (smallest AICc). We can do that very easily with the `which.min` function.

```
which.min(AICc)
```

```
## [1] 11
```

It seems that model 11 is the best from all the alternatives we tried.

```
selModel <- models[[11]]
selModel
```

```
## Series: y
## ARIMA(3,1,3)(0,1,0)[12]
##
## Coefficients:
##           ar1      ar2     ar3      ma1     ma2      ma3
##       -0.1986  -0.1510  0.7142  -0.0788  0.1258  -0.9750
## s.e.   0.0759   0.0768  0.0759   0.0424  0.0435   0.0565
##
## sigma^2 estimated as 122.6:  log likelihood=-500.79
## AIC=1015.58   AICc=1016.49   BIC=1035.71
```

Naturally, this is the best from the 16 alternatives we tried and not generally. None of these 16 models explored the seasonal ARMA. You can quickly see that it can be very easy to get lost in searching across for different parameters for the non-seasonal/seasonal AR and MA. In the literature there are many smart heuristics for how to search the model space. Some rely on integer optimisation, while others are closer to the ideas of stepwise regression (iteratively expand or restrict your model). The function `auto.arima` does exactly that. It uses a smart heuristic to find reasonable orders for the model. It should be noted that it does not try all combinations! That can easily become a massive computational burden when we have to move across 4 directions of search.

```
autoModel <- auto.arima(y)
autoModel
```

```
## Series: y
## ARIMA(2,1,1)(0,1,0)[12]
##
## Coefficients:
##          ar1     ar2      ma1
##       0.5960  0.2143  -0.9819
## s.e.  0.0888  0.0880   0.0292
##
## sigma^2 estimated as 132.3:  log likelihood=-504.92
## AIC=1017.85   AICc=1018.17   BIC=1029.35
```

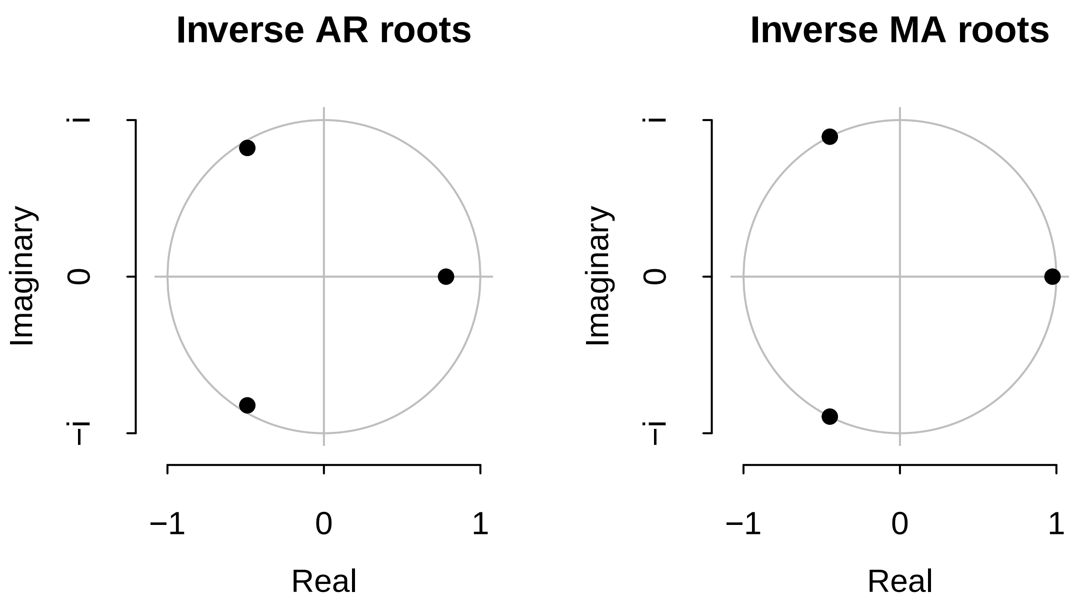
As it happens it comes up with a slightly simpler model than our selected one. It seems to be content with no seasonal ARMA terms. Also observe that its AICc is slightly larger, suggesting our model may be better. If you look at the help of `auto.arima` (using ? before the function's name, as before) you will observe the argument `ic` that instructs the heuristic what information criteria to optimise. As it happens this is the AICc. The reason that we disagree in the model is because we did an exhaustive search. The function provides a result faster and also evaluated some seasonal ARMA, so it had to cut corners somewhere! Another detail is that the orders of differencing were identified using the same functions we used. In fact this is again applied sequentially, first we test for stationarity, difference, and then deal with the ARMA terms.

Important! Note that we cannot use information criteria to compare models that do not have the same differencing!

The reason for that is in the calculation of information criteria we use the log-likelihood of the model, which is only comparable when the data are identical. Differencing will change the sample size (necessarily we lose the very first data points to construct our differences), and the scale of the data (both the mean and the variance will change with differencing). That is a limitation of conventional ARIMA modelling. If we embed ARIMA within a state-space formulation, then we can use information criteria, as differencing is handled differently. This is what the ARIMA implementation in the *smooth* package does! If you refer to my slides you will observe that some ETS models imply various orders of differencing that would normally not be comparable using information criteria. The reason we can do that is again due to the state-space formulation.

For this example, we prefer our model that is stored in `selModel`. The function in R will give you a warning if the model is invalid, so checking the coefficients manually is not necessary. Nonetheless, it is easy to plot them:

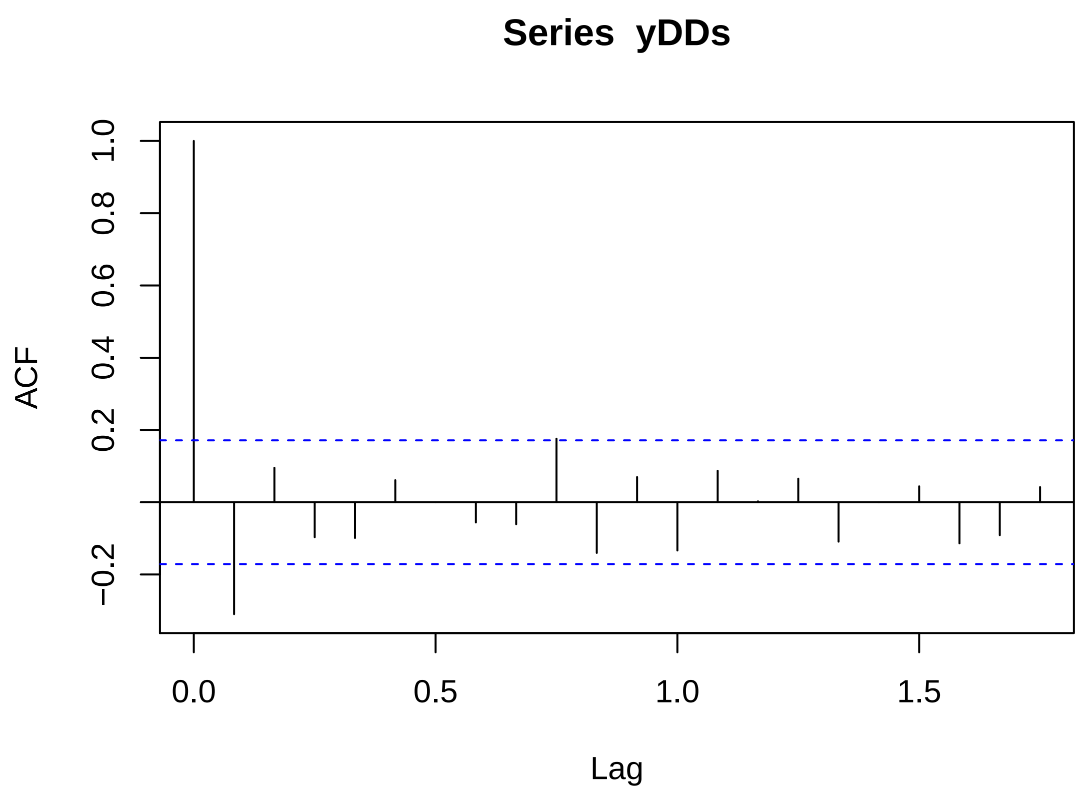
```r
plot(selModel)
```



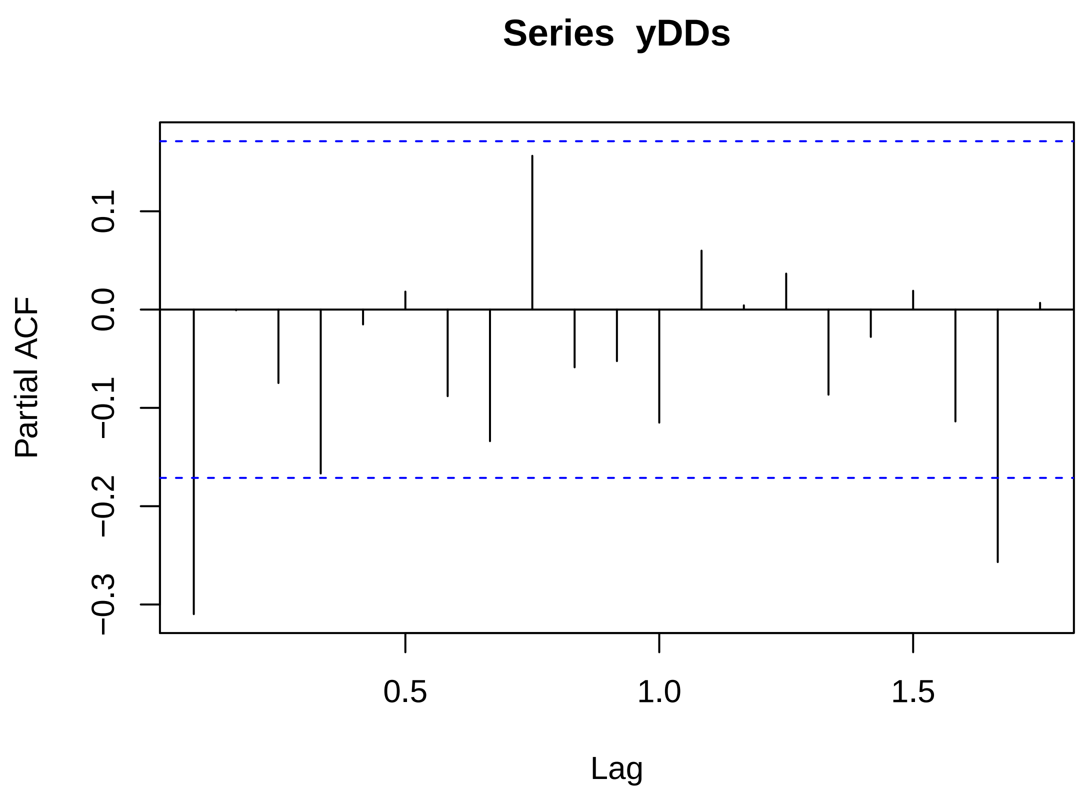
For the model to be valid all *inverse* roots must be within the unit circle. The plots show this both for the AR and MA terms, ensuring that the model behaves as it should!

To illustrate the difficulty of reading the ACF and PACF for real data, we use the stationary version of the series:

```r
acf(yDDs)
```


Series yDDs
ACF
Lag
1.0
0.8
0.6
0.4
0.2
−0.2
0.0
0.5
1.0
1.5

```r
pacf(yDDs)
```


Series yDDs
Partial ACF
Lag
0.1
0.0
−0.1
−0.2
−0.3
0.5
1.0
1.5

To me, it is not apparent how these charts suggest the selected model!

3.3 Forecasting

Now that we have identified a reasonable model we can construct forecasts. Forecasts for ARIMA are constructed iteratively, that is we forecast t+1, and then use it to forecast t+2, and so on. Thankfully this is handled by the software. So is differencing. If we did everything manually then we would have to reverse any differencing once we have constructed our forecasts. This is not necessary with the existing functions, as long as the differencing is part of the model and not done externally.

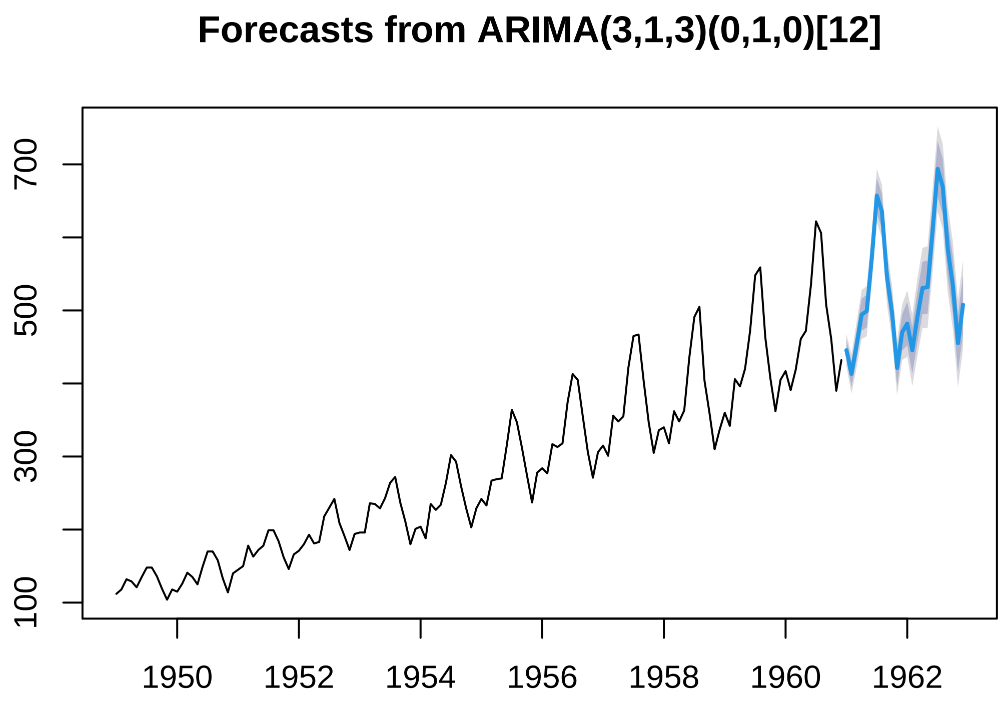
To produce forecasts we use the `forecast` function that takes two arguments, first the model and then the forecast horizon.

```
frc <- forecast(selModel,h=24)
frc
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Jan 1961       445.5745 431.2854 459.8636 423.7212 467.4278
## Feb 1961       413.1613 395.5828 430.7398 386.2773 440.0453
## Mar 1961       452.4572 431.9435 472.9709 421.0842 483.8302
## Apr 1961       494.3071 472.4367 516.1775 460.8592 527.7550
## May 1961       499.0510 476.5628 521.5391 464.6584 533.4436
## Jun 1961       571.3834 548.0045 594.7622 535.6285 607.1383
## Jul 1961       657.3677 633.5823 681.1532 620.9911 693.7444
## Aug 1961       635.6921 611.7271 659.6570 599.0408 672.3433
## Sep 1961       545.6376 521.2817 569.9936 508.3885 582.8868
## Oct 1961       497.1916 472.6701 521.7130 459.6893 534.6939
## Nov 1961       421.2254 396.6309 445.8199 383.6114 458.8394
## Dec 1961       470.1046 445.2867 494.9224 432.1490 508.0602
## Jan 1962       482.0302 452.2826 511.7779 436.5351 527.5253
## Feb 1962       445.3589 413.3951 477.3226 396.4745 494.2432
## Mar 1962       490.6623 456.0175 525.3072 437.6776 543.6471
## Apr 1962       530.7847 494.9028 566.6666 475.9081 585.6613
## May 1962       531.9233 495.4524 568.3941 476.1459 587.7006
## Jun 1962       609.5230 571.9853 647.0608 552.1141 666.9320
## Jul 1962       693.7721 655.7633 731.7809 635.6427 751.9015
## Aug 1962       669.0707 630.8324 707.3090 610.5903 727.5512
## Sep 1962       583.6411 544.8268 622.4554 524.2797 643.0024
## Oct 1962       533.4942 494.4369 572.5515 473.7612 593.2272
## Nov 1962       455.0064 415.8211 494.1918 395.0777 514.9352
## Dec 1962       507.9462 468.3741 547.5182 447.4259 568.4664
```

This gives us automatically the point forecast (forecasted mean) as well as the 80% and 95% prediction intervals. We can specify other intervals with additional arguments in the function (see help!). It is perhaps easier to understand the output if we visualise the forecasts.

```
plot(frc)
```



Not a terrible job! Note that the seasonal pattern was handled by the seasonal differencing and no additional seasonal AR or MA terms were needed.

We saved the forecasts in the variable `frc` that is a list. We use `names` to see what it contains

```
names(frc)
```

```
##  [1] "method"    "model"     "level"     "mean"      "lower"     "upper"
##  [7] "x"         "series"    "fitted"    "residuals"
```

There are quite a few variables therein. I draw your attention to `mean` that contains the point forecasts, the `lower` and `upper` that contains the lower and upper prediction intervals, and `fitted` that contains the in-sample model fit. For example, we can extract the point forecasts as:

```
frc$mean
```

```
##           Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 1961 445.5745 413.1613 452.4572 494.3071 499.0510 571.3834 657.3677 635.6921
## 1962 482.0302 445.3589 490.6623 530.7847 531.9233 609.5230 693.7721 669.0707
##           Sep      Oct      Nov      Dec
## 1961 545.6376 497.1916 421.2254 470.1046
## 1962 583.6411 533.4942 455.0064 507.9462
```

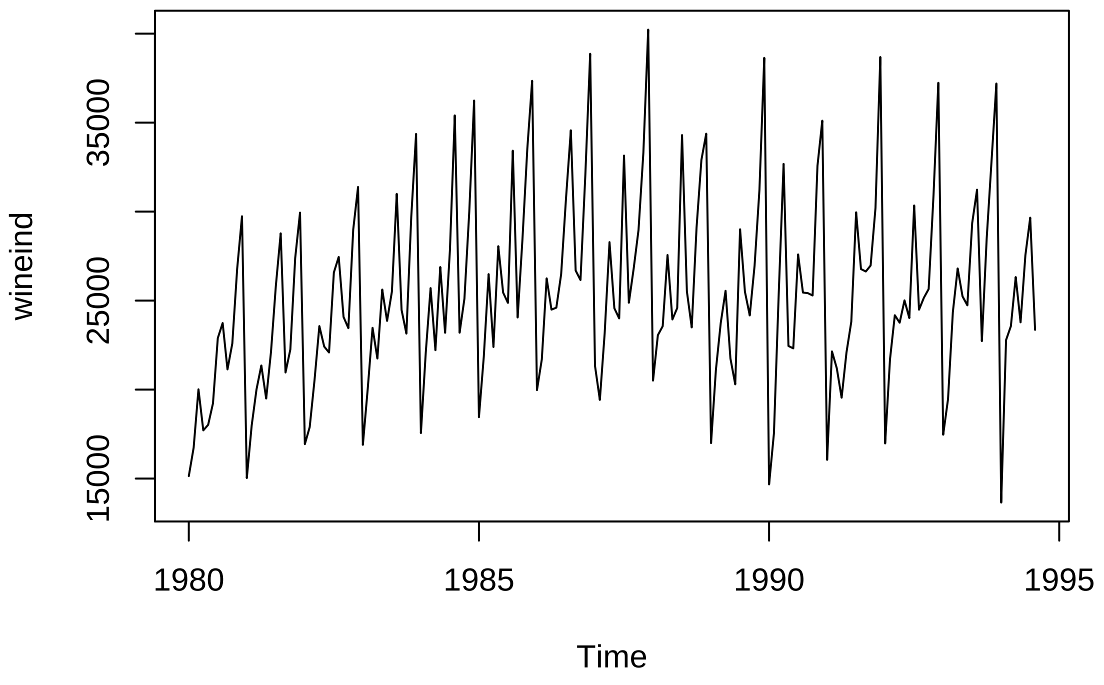
Observe that the time series characteristics (sampling frequency and dates) are carried from the original `y`. ARIMA needs `y` to be a time series so that it can know what seasonal periodicity to use, if any, and to annotate dates correctly.

> In principle there is nothing to prohibit us from introducing multiple seasonalities with ARIMA, for instance with hourly data you may observe an hour of the day pattern, a day of the week pattern, and a day in the year pattern. We would simply include additional seasonal ARIMA components. Although in principle this is very simple, in practice the identification of the correct model becomes very difficult. Currently, the functions discussed here do not support this either. However, one could use differencing as an easy approximate fix. We would preprocess the data to remove two seasonalities through respective differencing, and then model the rest with a single seasonal ARIMA.

3.4 Practice

Use the time series `wineind` to practice your arima skills! This time series is the total bottle sales of Australian wine.

```r
plot(wineind)
```



Happy forecasting!