Fitting and Selecting ARIMA models

FISH 507 – Applied Time Series Analysis

Eli Holmes

14 Jan 2021

Today's topic: Box-Jenkins method

A. Model form selection

1. Evaluate stationarity
2. Fix stationarity problems - Selection of the differencing level (d)
3. Selection of the AR level (p)
4. Selection of the MA level (q)

B. Parameter estimation

C. Model checking

1. Test model residuals for distribution assumptions (e.g. Normality)
2. Test model residuals for temporal correlation

- For ARIMA models, much of the Box-Jenkins method will be automated with the **forecast** package functions, which we will use in the lab.
- When you work on state-space models (so ARMA with observation error), you will follow a ‘Box-Jenkins’ workflow though the process will not be so automated.

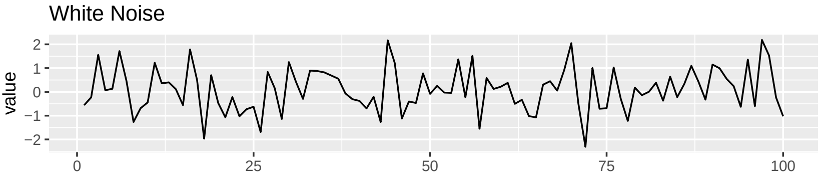
Stationarity

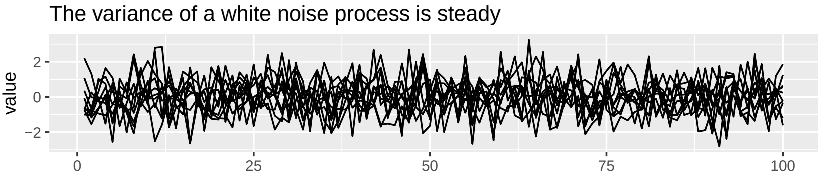
Stationarity means ‘not changing in time’ in the context of time-series models. Typically we test the trend and variance, however more generally all statistical properties of a time-series is time-constant if the time series is ‘stationary’.

Example

Many ARMA models exhibit stationarity. White noise is one type:

$$x_t = e_t, e_t \sim N(0, \sigma)$$



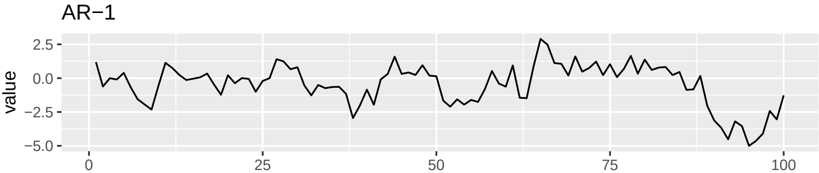


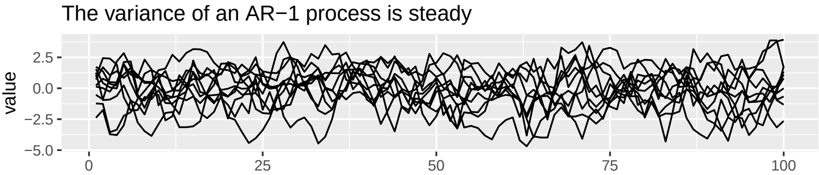
Example

An AR-1 process with $-1 < b < 1$

$$x_t = \phi x_{t-1} + e_t$$

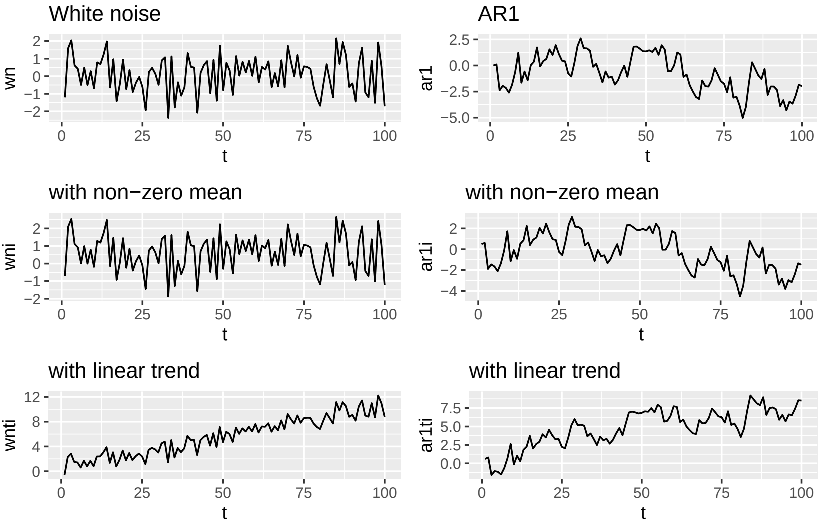
is also stationary.


AR−1


The variance of an AR−1 process is steady

Stationarity around non-zero mean

We can also have stationarity around a non-zero level or around a linear trend.



Mathematically it looks like this

AR-1

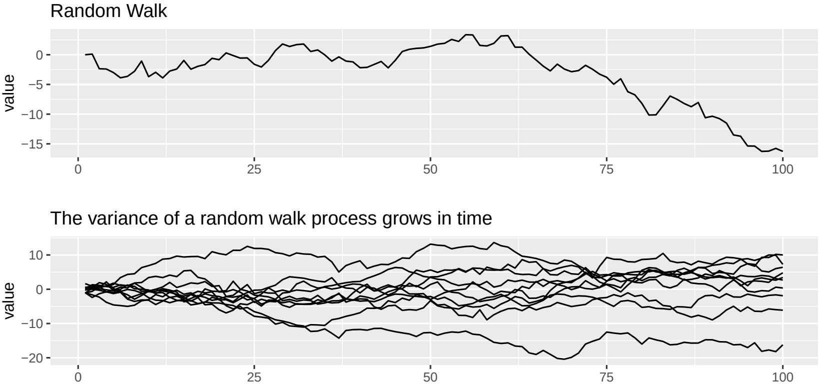
1. Non-zero mean adds μ: $x_t = \mu + \phi x_{t-1} + e_t$
2. Linear trend adds at: $x_t = \mu + at + \phi x_{t-1} + e_t$

White noise ($b = 0$)

1. Non-zero mean: $x_t = \mu + e_t$
2. Linear trend: $x_t = \mu + at + e_t$

Non-stationarity

One of the most common forms of non-stationarity that is tested for is that the process is a random walk $x_t = x_{t-1} + e_t$. A random walk is called a ‘unit root’ process in the time series literature. A test for an underlying random walk is called a ‘unit root’ test.



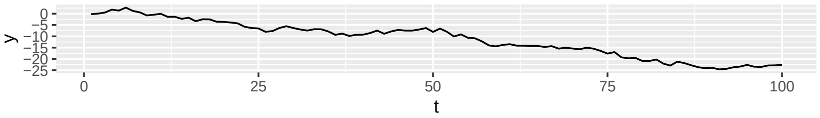
Random walk with μ and *at* added

Similar to the way we added an intecept and linear trend to the stationarity process equations, we can do the same to the random walk equation.

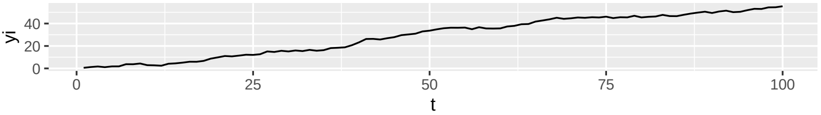
1. Non-zero mean or intercept: $x_t = \mu + x_{t-1} + e_t$
2. Linear trend: $x_t = \mu + at + x_{t-1} + e_t$

Random walk with μ and at added

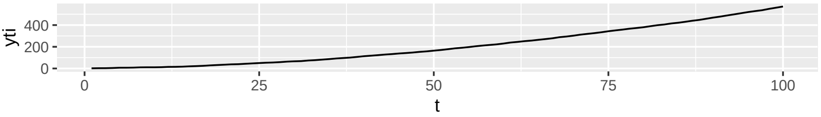
The effects are fundamentally different however. The addition of μ leads to a upward mean linear trend while the addition of at leads to exponential growth (or decline).



Random Walk



with non−zero mean added



with linear trend added

Testing for stationarity

Why is evaluating stationarity important?

- Many AR models have a flat level or trend and time-constant variance. If your data do not have those properties, you are fitting a model that is fundamentally inconsistent with your data.
- Many standard algorithms for fitting ARIMA models assume stationarity. Note, you can fit ARIMA models without making this assumption, but you need to use the appropriate algorithm.
- Many time-series models used in environmental science are random walk models, thus are non-stationary, e.g. movement, population growth, genetic drift. One must be careful to either deal with non-stationarity via transformation (differencing or covariates) OR use an algorithm which doesn't assume stationarity. We will do mainly the latter in this class.

Testing for stationarity

We will discuss three common approaches to evaluating stationarity:

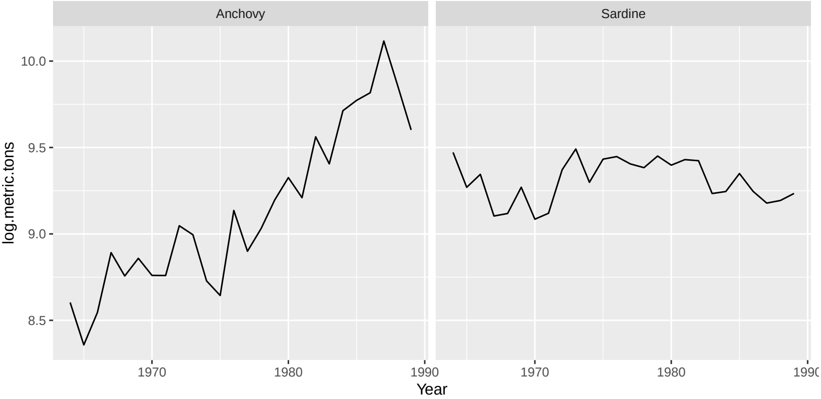
- Visual test
- (Augmented) Dickey-Fuller test
- KPSS test

Visual test

The visual test is simply looking at a plot of the data versus time. Look for

- Change in the level over time. Is the time series increasing or decreasing? Does it appear to cycle?
- Change in the variance over time. Do deviations away from the mean change over time, increase or decrease?

Anchovy and sardine catch in Greek waters



Dickey-Fuller test

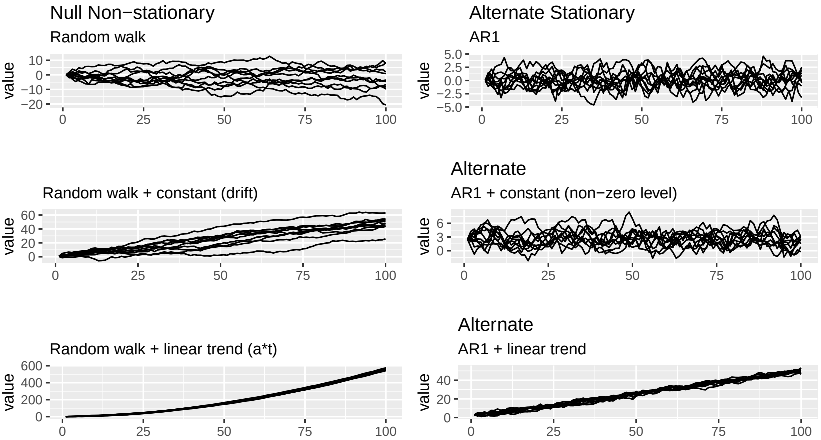
The Dickey=Fuller test (and Augmented Dickey-Fuller test) look for evidence that the time series has a unit root (a random walk process).

The **null hypothesis** is that the time series has a unit root, that is, it has a random walk component.

The **alternative hypothesis** is some variation of stationarity. The test has three main verisons.

Dickey-Fuller nulls and alternatives

It is hard to see but in the panels on the left, the variance around the trend is increasing and on the right, it is not.



Null Non−stationary

Random walk

Alternate Stationary

AR1

Random walk + constant (drift)

Alternate

AR1 + constant (non−zero level)

Random walk + linear trend (a*t)

Alternate

AR1 + linear trend

Dickey-Fuller test using `tseries::adf.test()`

`adf.test()` in the tseries package will apply the Augmented Dickey-Fuller **with a constant and trend** and report the p-value. We want to reject the Dickey=Fuller null hypothesis of non-stationarity. We will set `k=0` to apply the Dickey-Fuller test which tests for AR(1) stationarity. The Augmented Dickey-Fuller tests for more general lag-p stationarity.

```
adf.test(x, alternative = c("stationary", "explosive"),
         k = trunc((length(x)-1)^(1/3)))
```

Dickey-Fuller tests on the anchovy time series

Here is how to apply this test to the anchovy data. The null hypothesis is not rejected. That is not what we want.

```
adf.test(anchovyts, k=0)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  anchovyts
## Dickey-Fuller = -3.4558, Lag order = 0, p-value =
## 0.07003
## alternative hypothesis: stationary
```

Dickey-Fuller test with `urca::ur.df`

The `urca` R package can also be used to apply the Dickey-Fuller tests. Use `lags=0` for Dickey-Fuller which tests for AR(1) stationarity. We will set `type="trend"` to deal with the trend seen in the anchovy data. Note, `adf.test()` uses this type by default.

```
ur.df(y, type = c("none", "drift", "trend"), lags = 0)
```

- none: $x_t = \phi x_{t-1} + e_t$
- drift: $x_t = \phi x_{t-1} + \mu + e_t$
- trend: $x_t = \phi x_{t-1} + \mu + at + e_t$

Dickey-Fuller test with 'ur.df'

```r
test = urca::ur.df(anchovyts, type="trend", lags=0)
test
```

```
##
## #######################################################
## # Augmented Dickey-Fuller Test Unit Root / Cointegration
## #######################################################
##
## The value of the test statistic is: -3.4558 4.3568 5.980
```

Dickey-Fuller test with ‘ur.df’

The test statistic and the critical value at $\alpha = 0.05$ are

```r
attr(test, "teststat")
```

```
##                tau3     phi2     phi3
## statistic -3.455795 4.356764 5.980506
```

```r
attr(test,"cval")
```

```
##       1pct  5pct 10pct
## tau3 -4.15 -3.50 -3.18
## phi2  7.02  5.13  4.31
## phi3  9.31  6.73  5.61
```

The statistic is larger than the critical value and thus the null hypothesis of non-stationarity is not rejected. That’s not what we want.

The τ_3 is the one we want. This is the stationarity parameter.

$x_t = \phi x_{t-1} + \mu + at + e_t$

$x_t - x_{t-1} = \tau_3 x_{t-1} + \phi_2 + \phi_3 t + e_t$

KPSS test

The Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test has as the null hypothesis that the time series is stationary around a level trend (or a linear trend). The alternative hypothesis for the KPSS test is a random walk.

```
tseries::kpss.test(x, null = c("Level", "Trend"))
```

The stationarity assumption is general; it does not assume a specific type of stationarity such as white noise.

If both KPSS and Dickey-Fuller tests support non-stationarity, then the stationarity assumption is not supported.

KPSS test with the anchovy data

```
tseries::kpss.test(anchovyts, null="Trend")
```

```
##
##  KPSS Test for Trend Stationarity
##
## data:  anchovyts
## KPSS Trend = 0.14779, Truncation lag parameter = 2,
## p-value = 0.04851
```

Here `null="Trend"` was included to account for the increasing trend in the data. The null hypothesis of stationarity is rejected. Thus both the KPSS and Dickey-Fuller tests support the hypothesis that the anchovy time series is non-stationary. That's not what we want.

Fix stationarity problems

In this lecture we will use differencing, the I in ARIMA model refers to differencing.

Differencing the data to make the mean stationary

Differencing means to create a new time series $z_t = x_t - x_{t-1}$. First order differencing means you do this once (so z_t) and second order differencing means you do this twice (so $z_t - z_{t-1}$).

The `diff()` function takes the first difference:

```
x <- diff(c(1,2,4,7,11))
x
```

```
## [1] 1 2 3 4
```

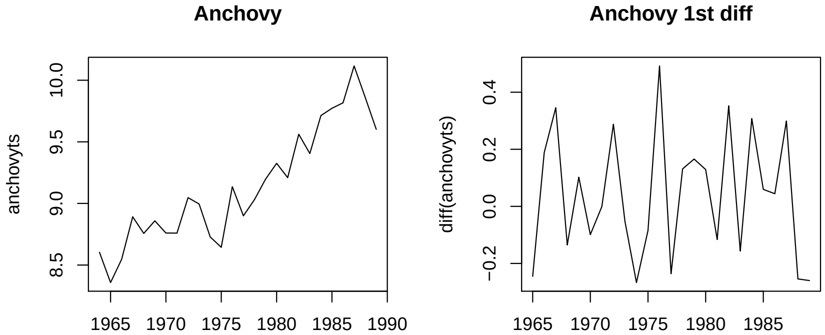
The second difference is the first difference of the first difference.

```
diff(x)
```

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

Anchovy catch first differenced

Here is a plot of the anchovy data and its first difference.



Stationarity of the first differences

Let's test the anchovy data with one difference using the KPSS test.

```r
diff.anchovy = diff(anchovyts)
kpss.test(diff.anchovy)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  diff.anchovy
## KPSS Level = 0.089671, Truncation lag parameter = 2,
## p-value = 0.1
```

The null hypothesis of stationairity is not rejected. That is good.

Stationarity of the first differences

Let's test the first difference of the anchovy data using the Augmented Dickey-Fuller test. We do the default test and allow it to chose the number of lags.

```
adf.test(diff.anchovy)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff.anchovy
## Dickey-Fuller = -3.2718, Lag order = 2, p-value =
## 0.09558
## alternative hypothesis: stationary
```

The null hypothesis of non-stationarity is not rejected. That is not what we want. However, we differenced which removed the trend thus we are testing against a more general model than we need. Let's test with an alternative hypothesis that has a non-zero mean and no trend. We can do this with `ur.df()` and `type='drift'`.

```
test <- ur.df(diff.anchovy, type="drift")
```

The test statistic and the critical values are

```r
attr(test, "teststat")
```

```
##                 tau2     phi1
## statistic -5.108275 13.15327
```

```r
attr(test,"cval")
```

```
##       1pct  5pct 10pct
## tau2 -3.75 -3.00 -2.63
## phi1  7.88  5.18  4.12
```

The test statistic for τ_2 is less than the critical at α equal 0.05. The null hypothesis of NON-stationairity IS rejected. That is good.

forecast::ndiffs() function

As an alternative to trying many different differences, you can use the `ndiffs()` function in the forecast package. This automates finding the number of differences needed.

```r
forecast::ndiffs(anchovyts, test="kpss")
```

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

```r
forecast::ndiffs(anchovyts, test="adf")
```

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

The test indicates that one difference $(x_t - x_{t-1})$ will lead to stationarity.

Summary

Test stationarity before you fit a ARMA model.

Visual test: Do the data fluctuate around a level or do they have a trend or look like a random walk?

Yes or maybe? -> Apply a “unit root” test. ADF or KPSS

No or fails the unit root test? -> Apply differencing and re-test.

Still not passing? -> Try a second difference or you may need to transform the data (if say it has an exponential trend).

Still not passing? -> ARMA model might not be the best choice. Or you may need to use an adhoc detrend.

These steps are automated by the forecast package

Box-Jenkins method

A. Model form selection

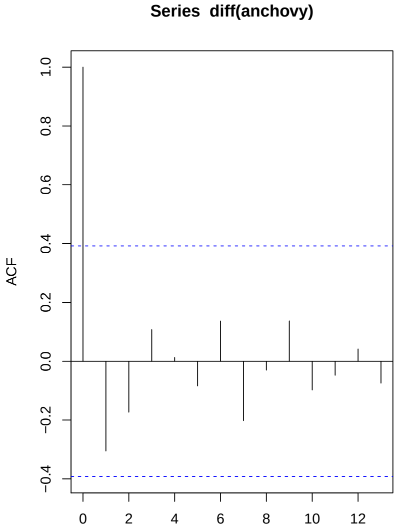
1. Evaluate stationarity
2. Selection of the differencing level (d) – to fix stationarity problems
3. **Selection of the AR level (p)**
4. **Selection of the MA level (q)**

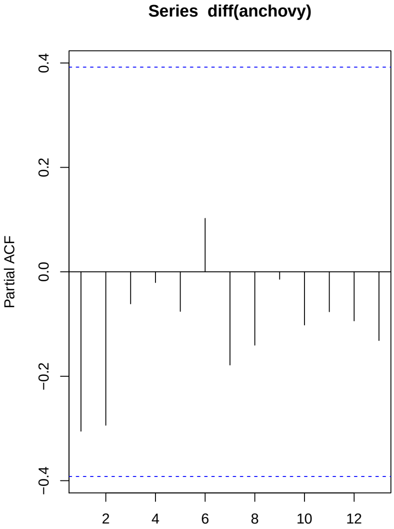
B. Parameter estimation

C. Model checking

ACF and PACF

On Tuesday, you learned how to use ACF and PACF to visually infer the AR and MA lags for a ARMA model. Here is the ACF and PACF of the differenced anchovy time series.





Formal model selection

This weighs how well the model fits against how many parameters your model has. Basic idea is to fit (many) models and use AIC, AICc or BIC to select.

The `auto.arima()` function in the forecast package in R allows you to easily do this and will also select the level of differencing (via ADF or KPSS tests).

```
forecast::auto.arima(anchovy)
```

Type `?forecast::auto.arima` to see a full description of the function.

Model selection with auto.arima()

```
forecast::auto.arima(anchovy)
```

```
## Series: anchovy
## ARIMA(0,1,1) with drift
##
## Coefficients:
##           ma1   drift
##       -0.6685  0.0542
## s.e.   0.1977  0.0142
##
## sigma^2 estimated as 0.04037:  log likelihood=5.39
## AIC=-4.79   AICc=-3.65   BIC=-1.13
```

The output indicates that the ‘best’ model is a MA(1) with first difference. “with drift” means that the mean of the anchovy first differences (the data for the model) is not zero.

Trace = TRUE

By default, step-wise selection is used for the model search. You can see what models that auto.arima() tried using trace=TRUE. The models are selected on AICc by default.

```
forecast::auto.arima(anchovy, trace=TRUE)
```

```
##
##  ARIMA(2,1,2) with drift          : 4.765453
##  ARIMA(0,1,0) with drift          : 0.01359746
##  ARIMA(1,1,0) with drift          : -0.1662165
##  ARIMA(0,1,1) with drift          : -3.647076
##  ARIMA(0,1,0)                     : -1.554413
##  ARIMA(1,1,1) with drift          : Inf
##  ARIMA(0,1,2) with drift          : Inf
##  ARIMA(1,1,2) with drift          : 1.653078
##  ARIMA(0,1,1)                     : -1.372929
##
##  Best model: ARIMA(0,1,1) with drift
```

```
## Series: anchovy
```

Selected model

First difference of the data is MA(1) with drift

$$x_t - x_{t-1} = \mu + w_t + \theta_1 w_{t-1}$$

where w_t is white noise.

Fit to simulated AR(2) data

```r
set.seed(100)
a1 = arima.sim(n=100, model=list(ar=c(.8,.1)))
forecast::auto.arima(a1, seasonal=FALSE, max.d=0)
```

```
## Series: a1
## ARIMA(1,0,0) with non-zero mean
##
## Coefficients:
##           ar1     mean
##        0.6928  -0.5343
## s.e.  0.0732   0.2774
##
## sigma^2 estimated as 0.7703:  log likelihood=-128.16
## AIC=262.33   AICc=262.58   BIC=270.14
```

The ‘best-fit’ model is AR(1) not AR(2).

How often is the ‘true’ model is chosen

Let’s run 100 simulations of a AR(2) process and record the best fits.

```
save.fits = rep(NA,100)
for(i in 1:100){
  a1 = arima.sim(n=100, model=list(ar=c(.8,.1)))
  fit = forecast::auto.arima(a1, seasonal=FALSE, max.d=0, m
  save.fits[i] = paste0(fit$arma[1], "-", fit$arma[2])
}
```

Overwhelmingly the correct type of model (AR) is selected, but usually a simpler model of AR(1) is chosen over AR(2).

Table heading is AR order - MA order.

```
table(save.fits)
```

```
## save.fits
## 1-0 2-0 3-0 4-0
##  74  20   5   1
```

stepwise=FALSE

By default, step-wise selection is used and an approximation is used for the models tried in the model selection step. For a final model selection, you should turn these off to fit a large set of models.

```r
forecast::auto.arima(anchovy, stepwise=FALSE,
                     approximation=FALSE)
```

```
## Series: anchovy
## ARIMA(0,1,1) with drift
##
## Coefficients:
##           ma1   drift
##       -0.6685  0.0542
## s.e.   0.1977  0.0142
##
## sigma^2 estimated as 0.04037:  log likelihood=5.39
## AIC=-4.79   AICc=-3.65   BIC=-1.13
```

Summary: model selection and fitting

- Once you have dealt with stationarity, you need to determine the order of the model: the AR part and the MA part.
- Although you could simply use `auto.arima()`, it is best to run `acf()` and `pacf()` on your data to understand it better. Definitely you want to plot your data and visually look for stationarity issues.
- Also evaluate if there are reasons to assume a particular structure.
  - Are you using an established model form, from say another paper?
  - Are you fitting to a process that is fundamentally AR only or AR + MA?

Box-Jenkins method

A. Model form selection

1. Evaluate stationarity
2. Selection of the differencing level (d) – to fix stationarity problems
3. Selection of the AR level (p)
4. Selection of the MA level (q)

B. Parameter estimation

C. Model checking

Check the residuals

Residuals = difference between the expected (fitted) value of x_t and the data

There is no observation error in an ARMA model. The expected value is the x_t expected from data up to $t-1$.

For example, the residual for an AR(2) model is $y_t - \hat{x}_t$.

$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + w_t$

$\hat{x}_t = \phi_1 x_{t-1} + \phi_2 x_{t-2}$

residuals() function in R

The residuals() function will return the residuals for fitted models.

```
fit <- forecast::auto.arima(anchovy)
residuals(fit)
```

```
## Time Series:
## Start = 1
## End = 26
## Frequency = 1
##  [1]  0.008549039 -0.249032308 -0.004098059  0.281393071
##  [5] -0.006015194  0.043859685 -0.123711732 -0.137125900
##  [9]  0.142098844 -0.011246624 -0.328608840 -0.358310373
## [13]  0.198311913 -0.157824727 -0.028321380  0.092732171
## [17]  0.136826748 -0.078995675  0.245238274 -0.046755189
## [21]  0.222279848  0.153983301  0.093036353  0.307250228
## [25] -0.103051063 -0.383026466
```

fitted() function in R

The fitted() function will return the expected values. Remember that for a ARMA model, these are the expected values conditioned on the data up to time $t-1$.

```r
fitted(fit)
```

```
## Time Series:
## Start = 1
## End = 26
## Frequency = 1
##  [1] 8.594675 8.606878 8.550151 8.610325 8.762619 8.8149
##  [7] 8.883569 8.896418 8.905111 9.006436 9.056857 9.0020
## [13] 8.937456 9.057378 9.059236 9.104026 9.188947 9.2884
## [19] 9.316477 9.451955 9.490634 9.618501 9.723727 9.8087
## [25] 9.964784 9.984801
```

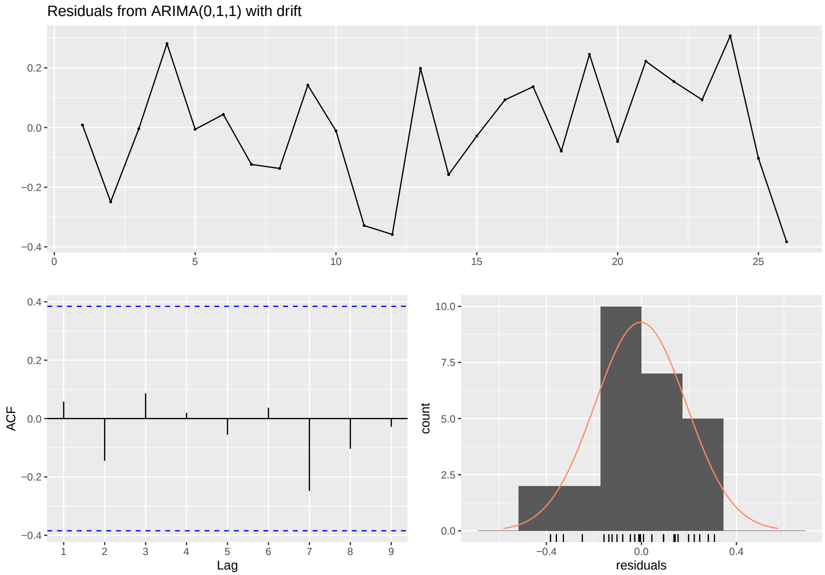
The residuals are data minus fitted.

Standard residuals tests

- Plot the residuals. They should look roughly like white noise.
- Look at the ACF of the residuals. They should be uncorrelated.
- Look at the histogram. They should be normally distributed (if that is your error assumption).

Residuals check with forecast package

```r
forecast::checkresiduals(fit)
```



Test for autocorrelation

The standard test for autocorrelated time-series residuals is the Ljung-Box test. The null hypothesis for this test is **no autocorrelation**. We do not want to reject the null.

```
forecast::checkresiduals(fit, plot=FALSE)
```

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1) with drift
## Q* = 1.0902, df = 3, p-value = 0.7794
##
## Model df: 2.   Total lags used: 5
```

$p > 0.05$ would be interpreted as not enough statistical evidence to reject the null hypothesis.

Forecasting (brief)

The basic idea of forecasting with an ARIMA model to estimate the parameters and forecast forward.

For example, let's say we want to forecast with this ARIMA(2,1,0) model:

$$x_t = \mu + \beta_1 x_{t-1} + \beta_2 x_{t-2} + e_t$$

or as `Arima()` would write this model:

$$(x_t - m) = \beta_1(x_{t-1} - m) + \beta_2(x_{t-2} - m) + e_t$$

where x_t is the first difference of our anchovy data.
$\mu = m(1 - \beta_1 - \beta_2)$.

Let's estimate the β's for this model from the anchovy data.

```
fit <- forecast::Arima(anchovyts, order=c(2,1,0), include.c
coef(fit)
```

```
##         ar1         ar2       drift 
## -0.53850433 -0.44732522  0.05367062
```

```
mu <- coef(fit)[3]*(1-coef(fit)[1]-coef(fit)[2])
mu
```

```
##     drift 
## 0.1065807
```

So we will forecast with this model:

$$x_t = 0.1065807 - 0.53850433 x_{t-1} - 0.44732522 x_{t-2} + e_t$$

To get our forecast for 1990, we do this

$$(x_{1990} - x_{1989}) = 0.106 - 0.538(x_{1989} - x_{1988}) - 0.447(x_{1988} - x_{1987})$$

Thus

$$x_{1990} = x_{1989} + 0.106 - 0.538(x_{1989} - x_{1988}) - 0.447(x_{1988} - x_{1987})$$

Here is R code to do that:

```
anchovyts[26]+mu+coef(fit)[1]*(anchovyts[26]-anchovyts[25])
  coef(fit)[2]*(anchovyts[25]-anchovyts[24])
```

```
##    drift
## 9.962083
```

Forecasting with `forecast()`

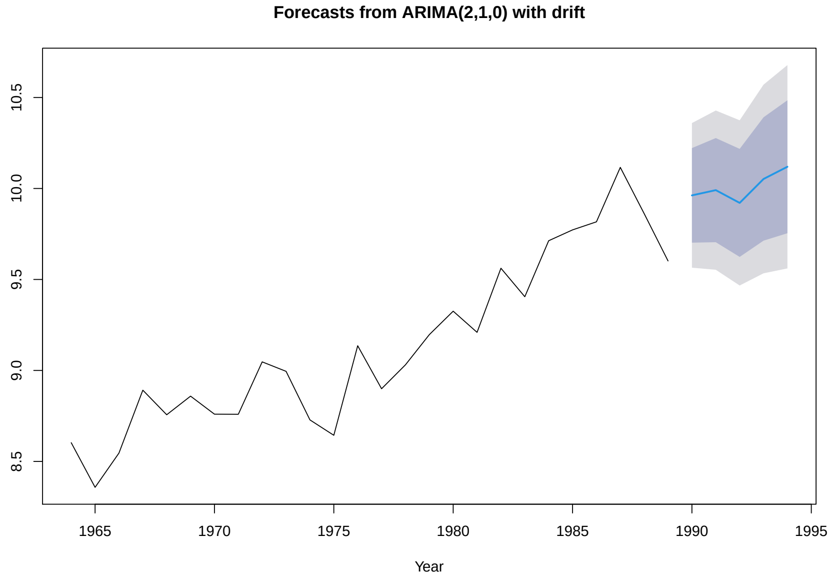
`forecast(fit, h=h)` automates the forecast calculations for us and computes the upper and lower prediction intervals. Prediction intervals include uncertainty in parameter estimates plus the process error uncertainty.

```
fr <- forecast::forecast(fit, h=5)
fr
```

```
##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 1990       9.962083 9.702309 10.22186 9.564793 10.35937
## 1991       9.990922 9.704819 10.27703 9.553365 10.42848
## 1992       9.920798 9.623984 10.21761 9.466861 10.37473
## 1993      10.052240 9.713327 10.39115 9.533917 10.57056
## 1994      10.119407 9.754101 10.48471 9.560719 10.67809
```

Plotting our forecasts

```
plot(fr, xlab="Year")
```



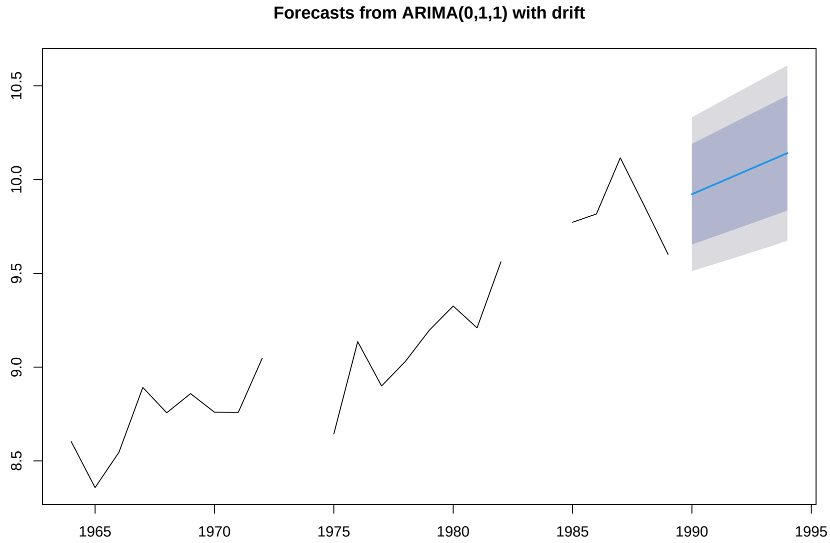
Missing values

Missing values are allowed for `Arima()` and `arima()`. We can produce forecasts with the same code.

```
anchovy.miss <- anchovyts
anchovy.miss[10:11] <- NA
anchovy.miss[20:21] <- NA
fit <- forecast::auto.arima(anchovy.miss)
fr <- forecast::forecast(fit, h=5)
fr
```

```
##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 1990       9.922074 9.654009 10.19014 9.512103 10.33205
## 1991       9.976827 9.698782 10.25487 9.551595 10.40206
## 1992      10.031579 9.743902 10.31926 9.591615 10.47154
## 1993      10.086332 9.789334 10.38333 9.632113 10.54055
## 1994      10.141084 9.835049 10.44712 9.673044 10.60912
```

```
plot(fr)
```


Forecasts from ARIMA(0,1,1) with drift
10.5
10.0
9.5
9.0
8.5
1965
1970
1975
1980
1985
1990
1995

Seasonality

Load the chinook salmon data set

```
load("chinook.RData")
head(chinook)
```

```
##   Year Month Species log.metric.tons metric.tons
## 1 1990   Jan Chinook        3.397858        29.9
## 2 1990   Feb Chinook        3.808882        45.1
## 3 1990   Mar Chinook        3.511545        33.5
## 4 1990   Apr Chinook        4.248495        70.0
## 5 1990   May Chinook        5.200705       181.4
## 6 1990   Jun Chinook        4.371976        79.2
```

The data are monthly and start in January 1990. To make this into a ts object do

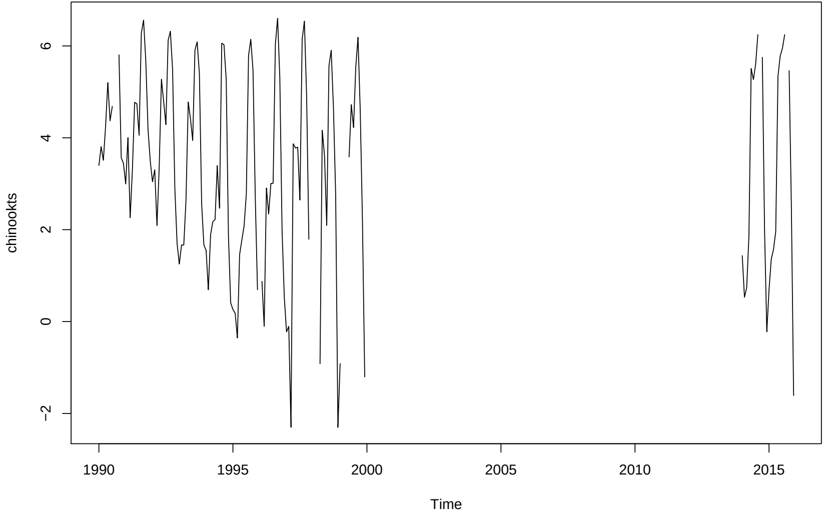
```
chinookts <- ts(chinook$log.metric.tons, start=c(1990,1),
                frequency=12)
```

`start` is the year and month and frequency is the number of months in the year.

Use `?ts` to see more examples of how to set up ts objects.

Plot seasonal data

```
plot(chinookts)
```



Seasonal ARIMA model

Seasonally differenced data:

$$z_t = x_t - x_{t+s} - m$$

Basic structure of a seasonal AR model

z_t = AR(p) + AR(season) + AR(p+season)

Example AR(1) non-seasonal part + AR(1) seasonal part

$$z_t = \phi_1 z_{t-1} + \Phi_1 z_{t-12} - \phi_1 \Phi_1 z_{t-13}$$

Notation

ARIMA (p,d,q)(ps,ds,qs)S

ARIMA (1,0,0)(1,1,0)[12]

auto.arima() for seasonal ts

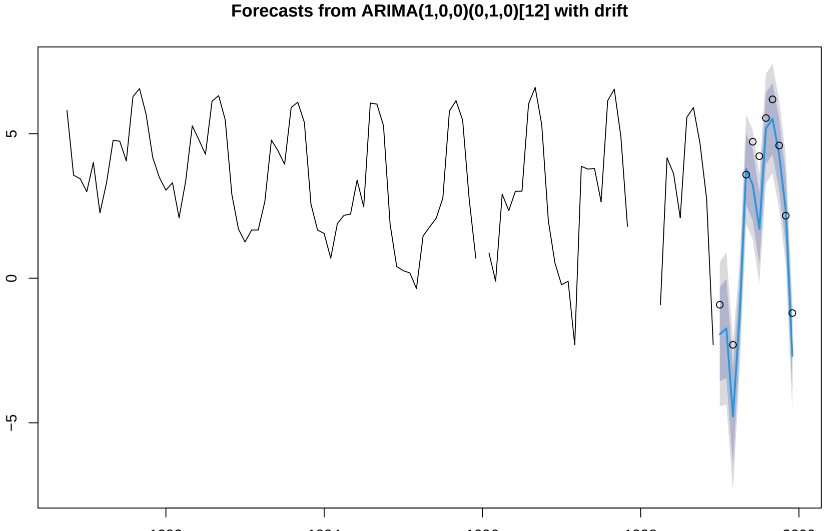
auto.arima() will recognize that our data has season and fit a seasonal ARIMA model to our data by default. We will define the training data up to 1998 and use 1999 as the test data.

```
traindat <- window(chinookts, c(1990,10), c(1998,12))
testdat <- window(chinookts, c(1999,1), c(1999,12))
fit <- forecast::auto.arima(traindat)
fit
```

```
## Series: traindat
## ARIMA(1,0,0)(0,1,0)[12] with drift
##
## Coefficients:
##          ar1    drift
##       0.3676  -0.0320
## s.e.  0.1335   0.0127
##
## sigma^2 estimated as 0.8053:  log likelihood=-107.37
## AIC=220.73   AICc=221.02   BIC=228.13
```

Forecast using seasonal model

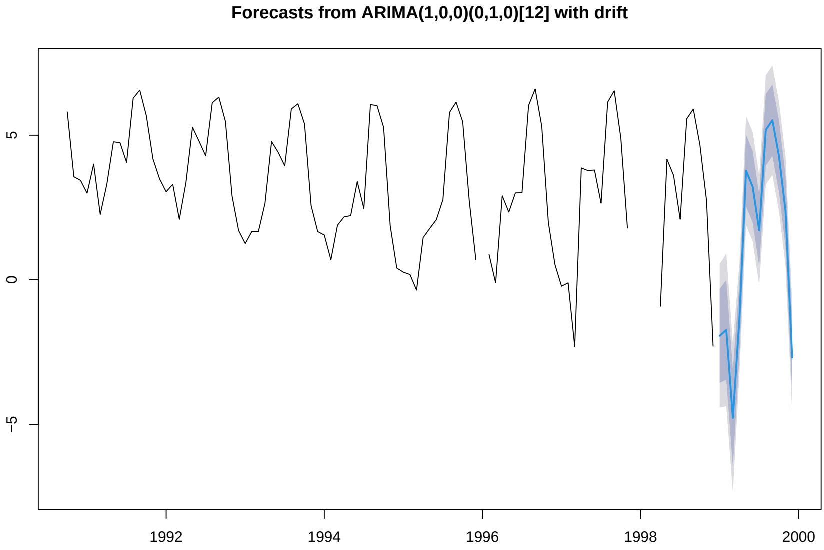
```
fr <- forecast::forecast(fit, h=12)
plot(fr)
points(testdat)
```



Forecasts from ARIMA(1,0,0)(0,1,0)[12] with drift

Missing values

Missing values are ok when fitting a seasonal ARIMA model



Summary

Basic steps for identifying a seasonal model. **forecast** automates most of this.

- Check that you have specified your season correctly in your ts object.
- Plot your data. Look for trend, seasonality and random walks.

Summary

- Use differencing to remove season and trend.
  - Season and no trend. Take a difference of lag = season
  - No seasonality but a trend. Try a first difference
  - Both. Do both types of differences
  - Neither. No differencing.
  - Random walk. First difference.
  - Parametric looking curve. Tranform.

Summary

- Examine the ACF and PACF of the differenced data.
  - Look for patterns (spikes) at seasonal lags
- Estimate likely models and compare with model selection criteria (or cross-validation). Use `TRACE=TRUE`
- Do residual checks.

Cover on your own: Intercepts and drift in `Arima()`

$d = 0$ `Arima(x, order=c(1,0,0), include.drift=FALSE, include.mean=TRUE)`

m is estimated and called `intercept`.

$$(x_t - m) = \phi_1(x_{t-1} - m) + w_t$$

`Arima(x, order=c(1,0,0), include.drift=TRUE, include.mean=FALSE)`

μ is estimated and called `drift`.

$$x_t = \mu + \phi_1 x_{t-1} + w_t$$

```
Arima(x, order=c(1,0,0), include.drift=TRUE,
include.mean=TRUE)
```

μ and m are estimated and called `drift` and `intercept`.

$$(x_t - m) = \mu + \phi_1(x_{t-1} - m) + w_t$$

If $d = 1$, then `include.mean` is ignored in `Arima()` and `include.drift` estimates an intercept like `include.mean` in the $d = 0$ case, but it is called `drift` in the output. $y_t = x_t - x_{t-1}$.

- `Arima(x, order=c(1,1,0), include.drift=TRUE)`

m is estimated and called `drift`.

$$(y_t - m) = \phi_1(y_{t-1} - m) + w_t$$

- `Arima(x, order=c(1,1,0), include.drift=FALSE)`

$$y_t = \phi_1 y_{t-1} + w_t$$

- `Arima(x, order=c(0,1,0), include.drift=TRUE)`

This is a random walk with drift.

$$(y_t - m) = w_t$$

which is

$$x_t = m + x_{t-1} + w_t$$

If $d \geq 2$, then both `include.mean` and `include.drift` are ignored. $z_t = y_t - y_{t-1} = (x_t - x_{t-1}) - (x_{t-1} - x_{t-2})$.

- `Arima(x, order=c(1,2,0))`

$$z_t = \phi_1 z_{t-1} + w_t$$

Intercepts in `arima()`

If $d = 0$,

- `arima(x, order=c(1,0,0), include.mean=TRUE)`

m is estimated and called `intercept`.

$$(x_t - m) = \phi_1(x_{t-1} - m) + w_t$$

If $d = 1$, then `include.mean` is ignored and no intercept can be estimated.

- `arima(x, order=c(1,1,0), include.mean=TRUE)`

$$y_t = \phi_1 y_{t-1} + w_t$$

- `arima(x, order=c(0,1,0))`

Because an intercept cannot be estimated, this means that a random walk with drift cannot be estimated by `arima()`.

$$y_t = w_t$$

only can be estimated which is random walk without drift.

$$x_t = x_{t-1} + w_t$$