Time Series Analysis

Shang Chieh

2023-05-03

Contents

Chapter 1

About

This book is a concise lecture note about *Time Series Analysis*.

The content of this book is from the course Time Series Analysis taught by Chris Bilder. You can check his YouTube channel to get full(and correct) information about this course.

Again, I do **NOT** own the content of this book. I write this book only for studying. All credits belong to Chris Bilder.

If there is any copyright concerns, I will make this book private ASAP.

Chapter 2

Introduction to R

We will go over some of the basic R operations in this chapter.

If you have questions, you should check Chris Bilder's website for full information.

2.1 Basic Operation

```
2+2
```

```
## [1] 4
```

```
2^3
```

```
## [1] 8
```

```
# calculate the cdf of std. normal
pnorm(1.96) # 1.96 is the quantile
```

```
## [1] 0.9750021
```

```
log(1)
```

```
## [1] 0
```

```
sin(pi/2)
```

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

```
3/4
```

```
## [1] 0.75
```

```
save <- 2+2
save
```

```
## [1] 4
```

```
objects()
```

```
## [1] "save"
```

```
ls()
```

```
## [1] "save"
```

```
# quit operaiton
# q()
```

2.2 Vectors

```
x <- c(1,2,3,4,5)
x
```

```
## [1] 1 2 3 4 5
```

```
sd(x)
```

```
## [1] 1.581139
```

```
mysd <- function(x){
  cat(" My data \n", x, "\n has std deviation",sqrt(var(x)))
}


mysd(x)
```

```
##  My data
##  1 2 3 4 5
##  has std deviation 1.581139
```

```
pnorm(q=1.96, mean=1.96, sd=1)
```

```
## [1] 0.5
```

The full syntax for `pnorm()` is `pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`

```
pnorm(q=c(-1.96,1.96))
```

```
## [1] 0.0249979 0.9750021
```

```
x <- c(3.68, -3.63, 0.80, 3.03, -9.86, -8.66,
    -2.38, 8.94, 0.52, 1.25)

y <- c(0.55, 1.65, 0.98, -0.07, -0.01, -0.31,
    -0.34, -1.38, -1.32, 0.53)

x+y
```

```
##  [1]  4.23 -1.98  1.78  2.96 -9.87 -8.97 -2.72  7.56 -0.80  1.78
```

```
x*y
```

```
##  [1]   2.0240  -5.9895   0.7840  -0.2121   0.0986   2.6846   0.8092 -12.3372
##  [9]  -0.6864   0.6625
```

```
mean(x)
```

```
## [1] -0.631
```

```
x-mean(x)
```

```
##  [1]  4.311 -2.999  1.431  3.661 -9.229 -8.029 -1.749  9.571  1.151  1.881
```

```
x*2
```

```
##  [1]   7.36  -7.26   1.60   6.06 -19.72 -17.32  -4.76  17.88   1.04   2.50
```

The element(elt)-wise operation makes our life easier.

2.3 Files

Click gpa.csv to download the GPA csv file.

Click gpa.txt to download the GPA txt file.

```
getwd()
```

```
## [1] "/Users/weishangjie/Documents/GitHub/Time_Series_Analysis/Book/Time_Series_Analy
```

```
gpatxt <- read.table("gpa.txt", header=TRUE, sep="")
gpacsv <- read.csv("gpa.csv")

#write.table(x = gpacsv, file = "gpa-out1.csv", quote = FALSE, row.names =
#  FALSE, sep  =",")
#write.csv(x = gpacsv, file = "gpa-out2.csv")
```

```
gpacsv$HSGPA
```

```
##  [1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39 3.65 2.85 3.83 2.22
## [16] 1.98 2.88 4.00 2.28 2.88
```

```
gpacsv$CollegeGPA
```

```
##  [1] 3.10 2.30 3.00 2.45 2.50 3.70 3.40 2.60 2.80 3.60 2.00 2.90 3.30 3.20 2.80
## [16] 2.40 2.60 3.80 2.20 2.60
```

```
gpacsv[1,1] # [row, col]
```

```
## [1] 3.04
```

```
gpacsv[,1]
```

```
##  [1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39 3.65 2.85 3.83 2.22
## [16] 1.98 2.88 4.00 2.28 2.88
```

```
gpacsv[c(1,3,5),2]
```

```
## [1] 3.1 3.0 2.5
```

```
gpacsv[,"HSGPA"]
```

```
##  [1] 3.04 2.35 2.70 2.55 2.83 4.32 3.39 2.32 2.69 2.83 2.39 3.65 2.85 3.83 2.22
## [16] 1.98 2.88 4.00 2.28 2.88
```

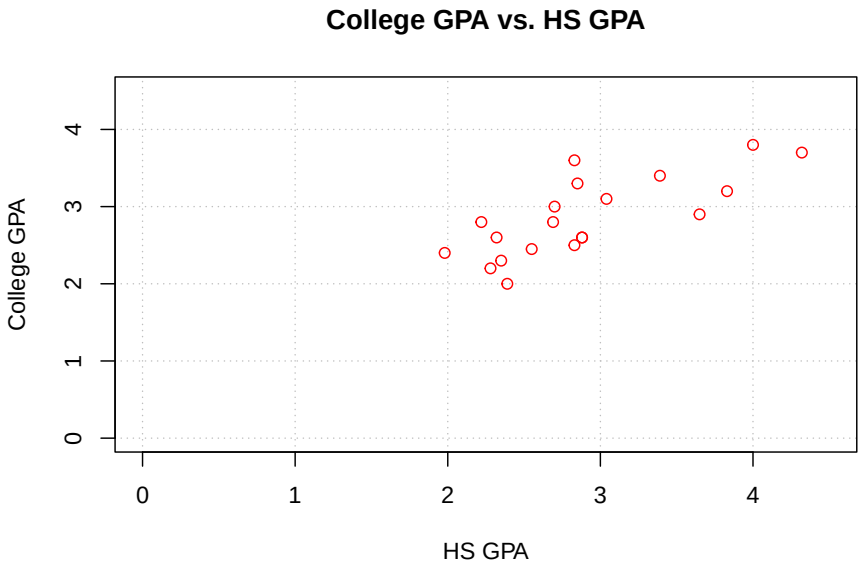
```
summary(gpacsv)
```

```
##      HSGPA         CollegeGPA
##  Min.   :1.980   Min.   :2.000
##  1st Qu.:2.380   1st Qu.:2.487
##  Median :2.830   Median :2.800
##  Mean   :2.899   Mean   :2.862
##  3rd Qu.:3.127   3rd Qu.:3.225
##  Max.   :4.320   Max.   :3.800
```

```
names(gpacsv)
```

```
## [1] "HSGPA"      "CollegeGPA"
```

```
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA,
     xlab = "HS GPA", ylab = "College GPA",
     main = "College GPA vs. HS GPA",
     xlim = c(0,4.5), ylim = c(0,4.5), col = "red",
    pch = 1, cex = 1.0, panel.first = grid(col = "gray", lty
    = "dotted"))
```



The `plot()` function creates a two dimensional plot of data.

Here are descriptions of its arguments:

- x specifies what is plotted for the x-axis.
- y specifies what is plotted for the y-axis.
- xlab and ylab specify the x-axis and y-axis labels, respectively.
- main specifies the main title of the plot.
- xlim and ylim specify the x-axis and y-axis limits, respectively.
  - Notice the use of the c() function.
- col specifies the color of the plotting points.
  - Run the `colors()` function to see what possible colors can be used.
  - Also, you can see Here for the colors from colors().
- `pch` specifies the plotting characters.
- `cex`specifies the height of the plotting characters. The value 1.0 is the default.
- `panel.first = grid()` specifies grid lines will be plotted.

- The line types can be specified as follows: `1=solid, 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash` or as one of the character strings `"blank", "solid", "dashed", "dotted", "dotdash", "longdash"`, or `"twodash"`.
  These line type specifications can be used in other functions.

- The `par()`(parameter) function's Help contains more information about the different plotting options!

2.4 Regression

Our model is:

$$CollegeGPA = \beta_0 + \beta_1 HSGPA + \epsilon$$

```
mod.fit <- lm(formula= CollegeGPA~ HSGPA, data=gpacsv)
mod.fit
```

```
##
## Call:
## lm(formula = CollegeGPA ~ HSGPA, data = gpacsv)
##
## Coefficients:
## (Intercept)        HSGPA
##      1.0869       0.6125
```

```
names(mod.fit)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
mod.fit$coefficients
```

```
## (Intercept)       HSGPA
##   1.0868795   0.6124941
```

```
round(mod.fit$residuals[1:5],2)
```

```
##     1     2     3     4     5
##  0.15 -0.23  0.26 -0.20 -0.32
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.1     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.1     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
```

```
save.fit <- data.frame(gpacsv, C.GPA.hat =
    round(mod.fit$fitted.values,2), residuals =
    round(mod.fit$residuals,2))

save.fit %>% head()
```

```
##   HSGPA CollegeGPA C.GPA.hat residuals
## 1  3.04       3.10      2.95      0.15
## 2  2.35       2.30      2.53     -0.23
## 3  2.70       3.00      2.74      0.26
## 4  2.55       2.45      2.65     -0.20
## 5  2.83       2.50      2.82     -0.32
## 6  4.32       3.70      3.73     -0.03
```

```
summary(mod.fit)
```

```
##
## Call:
## lm(formula = CollegeGPA ~ HSGPA, data = gpacsv)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.55074 -0.25086  0.01633  0.24242  0.77976
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.0869     0.3666   2.965 0.008299 **
## HSGPA         0.6125     0.1237   4.953 0.000103 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.3437 on 18 degrees of freedom
## Multiple R-squared:  0.5768, Adjusted R-squared:  0.5533
## F-statistic: 24.54 on 1 and 18 DF,  p-value: 0.0001027
```

```
class(mod.fit)
```

```
## [1] "lm"
```

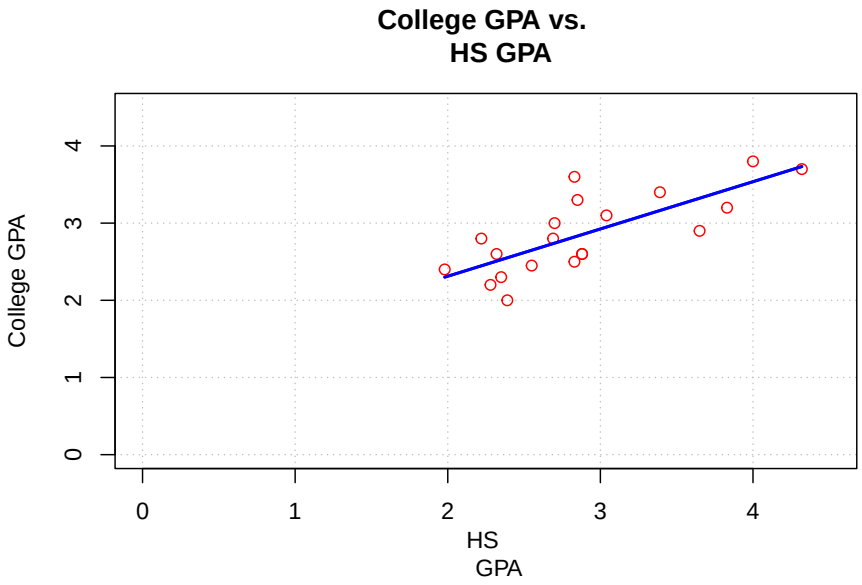
Hence, our estimated regression model is

$$\hat{collge.GPA} = \hat{\beta}_0 + \hat{\beta}_1 HS.GPA = 1.0869 + 0.6125 HS.GPA$$

```
# Open a new graphics window
# device new
dev.new(width = 8, height = 6, pointsize = 10)


# 1 row and 2 columns of plots
par(mfrow = c(1,2))
# par= graphic parameter
# mfrow= make a frame by row

# Same scatter plot as before
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS
    GPA", ylab = "College GPA", main = "College GPA vs.
    HS GPA", xlim = c(0,4.5), ylim = c(0,4.5), col =
    "red", pch = 1, cex = 1.0, panel.first = grid(col =
    "gray", lty = "dotted"))

# Puts the line y = a + bx on the plot
abline(a = mod.fit$coefficients[1], b =
    mod.fit$coefficients[2], lty = "solid", col =
    "blue", lwd = 2)
```

```
# Same scatter plot as before
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS
    GPA", ylab = "College GPA", main = "College GPA vs.
    HS GPA", xlim = c(0,4.5), ylim = c(0,4.5), col =
    "red", pch = 1, cex = 1.0, panel.first = grid(col =
    "gray", lty = "dotted"))


# Add line
```

```
# expr= math expression
curve(expr = mod.fit$coefficients[1] +
    mod.fit$coefficients[2]*x,
    xlim = c(min(gpacsv$HSGPA),max(gpacsv$HSGPA)),
    col= "blue", add = TRUE, lwd = 2)


# Draw a line from (x0, y0) to (x1, y1)
  segments(x0 = min(gpacsv$HSGPA), y0 = mod.fit$coefficients[1] + mod.fit$coefficients
           x1 = max(gpacsv$HSGPA), y1 = mod.fit$coefficients[1] + mod.fit$coefficients
           lty = 1, col = "blue", lwd = 2)
```



- The `dev.new()` function can be used to open a new plotting window.
- The `abline()` function can be used to draw straight lines on a plot. In the format used here, the line y = a + bx was drawn where a was the (intercept) and b was the (slope).
- In the second plot, the curve() function was used to draw the line on the plot. This was done to have the line within the range of the high school GPA values.

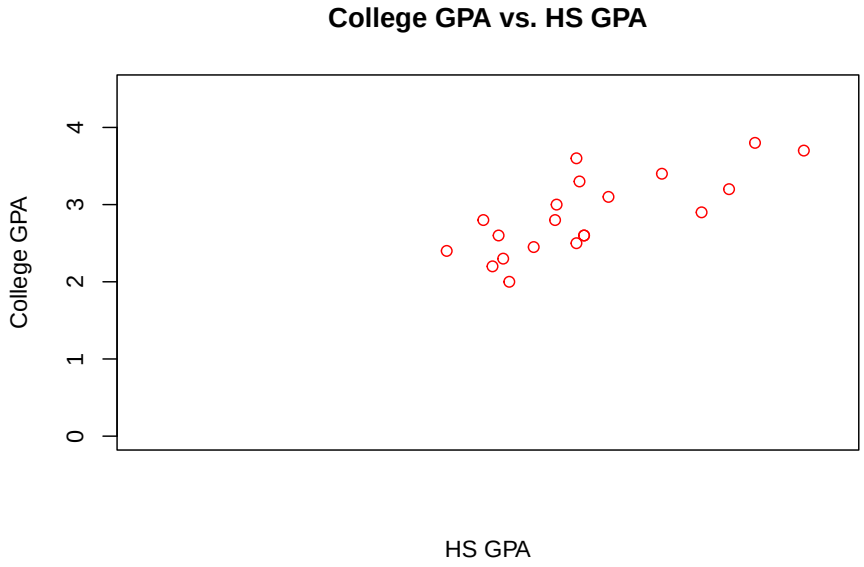
Let's use function to automate what we have done.

```
my.reg.func <- function(x, y, data) {

    # Fit the simple linear regression model and save the results in mod.fit
    mod.fit <- lm(formula = y ~ x, data = data)

    #Open a new graphics window - do not need to
    dev.new(width = 6, height = 6, pointsize = 10)

    # Same scatter plot as before
    plot(x = x, y = y, xlab = "x", ylab = "y", main = "y vs. x", panel.first=grid(col = "gray", l
      "dotted"))

    # Plot model
    curve(expr = mod.fit$coefficients[1] +
      mod.fit$coefficients[2]*x, xlim = c(min(x),max(x)),
      col = "blue", add = TRUE)

    segments(x0 = min(x), y0 = mod.fit$coefficients[1] + mod.fit$coefficients[2]*min(x),
             x1 = max(x), y1 = mod.fit$coefficients[1] + mod.fit$coefficients[2]*max(x),
             lty = 1, col = "blue", lwd = 2)

    # This is the object returned
    mod.fit
  }
```

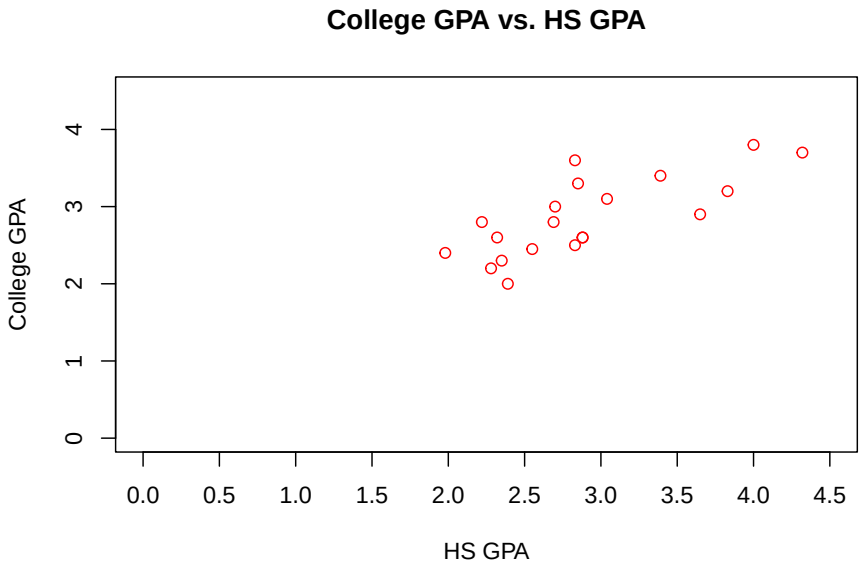
```
save.it <- my.reg.func(x = gpacsv$HSGPA, y =
    gpacsv$CollegeGPA, data = gpacsv)
```

To get specific x-axis or y-axis tick marks on a plot, use the `axis()` function. For example,

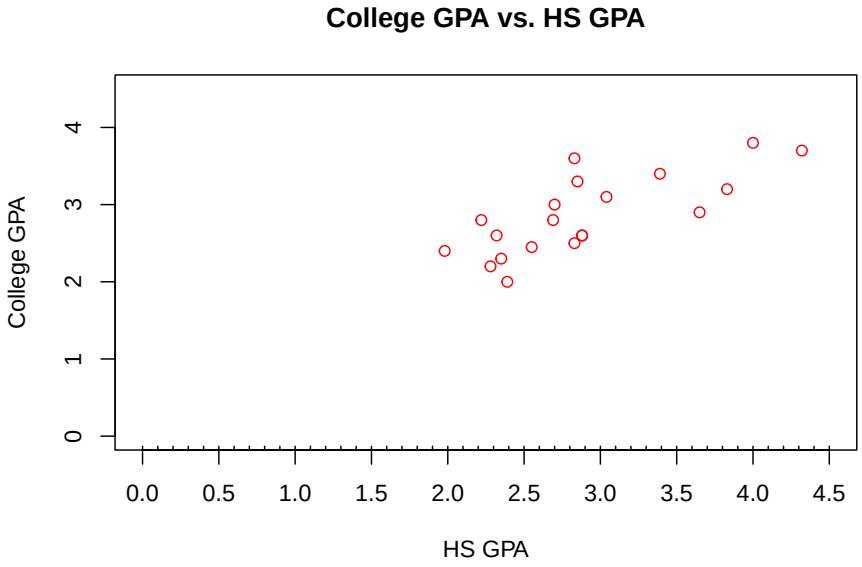
```
#Note that xaxt = "n" tells R to not give any labels on the
#  x-axis (yaxt = "n" works for y-axis)
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS GPA",
     ylab = "College GPA", main = "College GPA vs. HS GPA",
     xaxt = "n", xlim = c(0, 4.5), ylim = c(0, 4.5), col =
     "red", pch = 1)
```



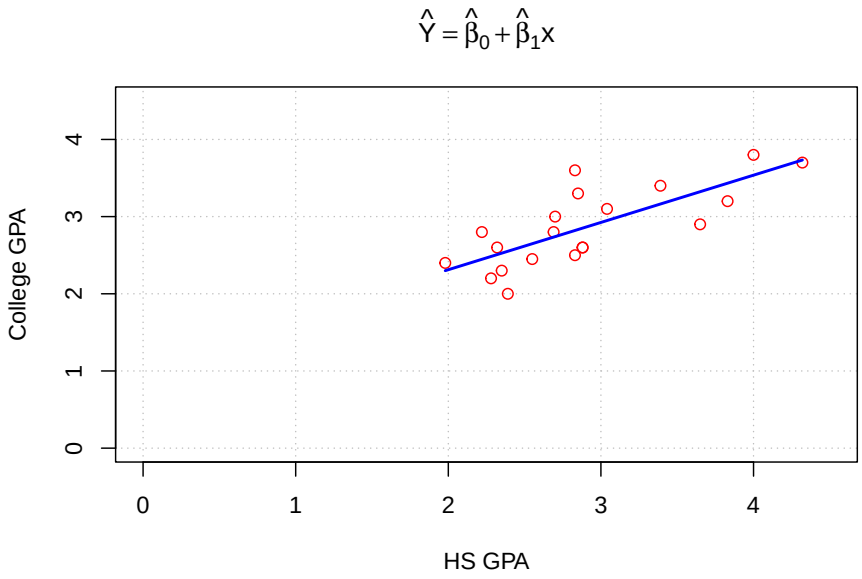

```
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS GPA",
     ylab = "College GPA", main = "College GPA vs. HS GPA",
     xaxt = "n", xlim = c(0, 4.5), ylim = c(0, 4.5), col =
     "red", pch = 1)

#Major tick marks
axis(side = 1, at = seq(from = 0, to = 4.5, by = 0.5))
```



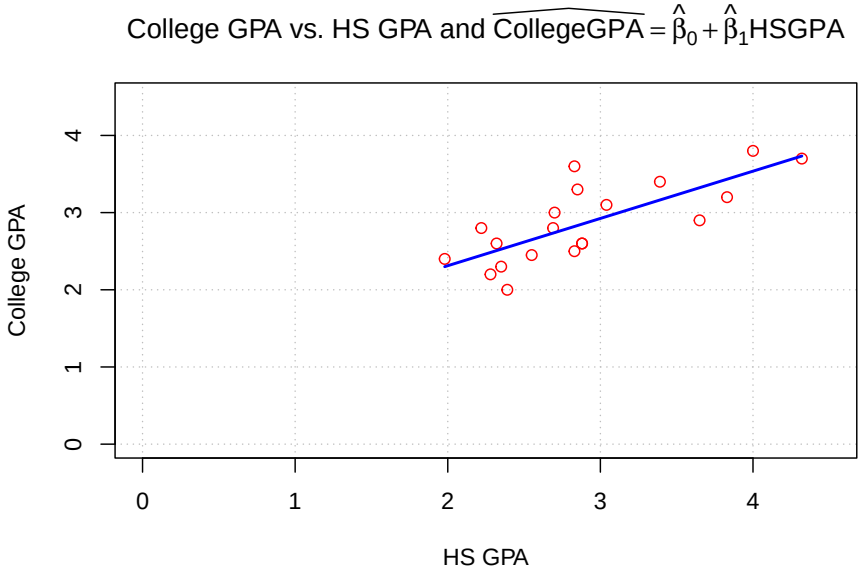

```
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS GPA",
     ylab = "College GPA", main = "College GPA vs. HS GPA",
     xaxt = "n", xlim = c(0, 4.5), ylim = c(0, 4.5), col =
     "red", pch = 1)

#Major tick marks
axis(side = 1, at = seq(from = 0, to = 4.5, by = 0.5))

#Minor tick marks
axis(side = 1, at = seq(from = 0, to = 4.5, by = 0.1), tck
      = 0.01, labels = FALSE)
```




```
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS GPA", ylab = "College GPA",
     main = expression(hat(Y) == hat(beta)[0] + hat(beta)[1]*x),
     xlim = c(0,4.5), ylim = c(0,4.5), col = "red", pch = 1, cex = 1.0, panel.first=gr

 #Draw a line from (x0, y0) to (x1, y1)
 segments(x0 = min(gpacsv$HSGPA), y0 = mod.fit$coefficients[1] + mod.fit$coefficients
          x1 = max(gpacsv$HSGPA), y1 = mod.fit$coefficients[1] + mod.fit$coefficients
          lty = 1, col = "blue", lwd = 2)
```




```r
plot(x = gpacsv$HSGPA, y = gpacsv$CollegeGPA, xlab = "HS GPA", ylab = "College GPA",
    main = expression(paste("College GPA vs. HS GPA and ", widehat(CollegeGPA) == hat(beta)[0]
    xlim = c(0,4.5), ylim = c(0,4.5), col = "red", pch = 1, cex = 1.0, panel.first=grid(col = '

#Draw a line from (x0, y0) to (x1, y1)
segments(x0 = min(gpacsv$HSGPA), y0 = mod.fit$coefficients[1] + mod.fit$coefficients[2]*min(gpa
         x1 = max(gpacsv$HSGPA), y1 = mod.fit$coefficients[1] + mod.fit$coefficients[2]*max(gpa
         lty = 1, col = "blue", lwd = 2)
```




```
demo(plotmath) #Run this to see examples
```

```
##
##
##  demo(plotmath)
##  ---- ~~~~~~~~
##
## > #  Copyright (C) 2002-2016 The R Core Team
## >
## > require(datasets)
##
## > require(grDevices); require(graphics)
##
## > ## --- "math annotation" in plots :
## >
## > ######
## > # create tables of mathematical annotation functionality
## > ######
## > make.table <- function(nr, nc) {
## +     savepar <- par(mar=rep(0, 4), pty="s")
## +     plot(c(0, nc*2 + 1), c(0, -(nr + 1)),
## +          type="n", xlab="", ylab="", axes=FALSE)
## +     savepar
## + }
```

```
##
## > get.r <- function(i, nr) {
## +     i %% nr + 1
## + }
##
## > get.c <- function(i, nr) {
## +     i %/% nr + 1
## + }
##
## > draw.title.cell <- function(title, i, nr) {
## +     r <- get.r(i, nr)
## +     c <- get.c(i, nr)
## +     text(2*c - .5, -r, title)
## +     rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5))
## + }
##
## > draw.plotmath.cell <- function(expr, i, nr, string = NULL) {
## +     r <- get.r(i, nr)
## +     c <- get.c(i, nr)
## +     if (is.null(string)) {
## +         string <- deparse(expr)
## +         string <- substr(string, 12, nchar(string) - 1)
## +     }
## +     text((2*(c - 1) + 1), -r, string, col="grey50")
## +     text((2*c), -r, expr, adj=c(.5,.5))
## +     rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5), border="grey")
## + }
##
## > nr <- 20
##
## > nc <- 2
##
## > oldpar <- make.table(nr, nc)
```

Arithmetic Operators		Lists	
x + y	$x+y$	list(x, y, z)	x, y, z
x − y	$x-y$	**Relations**	
x * y	xy	x == y	$x=y$
x/y	x/y	x != y	$x\neq y$
x %+−% y	$x\pm y$	x < y	$x<y$
x%/%y	$x\div y$	x <= y	$x\leq y$
x %*% y	$x\times y$	x > y	$x>y$
x %.% y	$x\cdot y$	x >= y	$x\geq y$
−x	$-x$	x %~~% y	$x\approx y$
+x	$+x$	x %=~% y	$x\cong y$
Sub/Superscripts		x %==% y	$x\equiv y$
x[i]	x_i	x %prop% y	$x\propto y$
x^2	x^2	x %~% y	$x\sim y$
Juxtaposition		**Typeface**	
x * y	xy	plain(x)	x
paste(x, y, z)	xyz	italic(x)	*x*
Radicals		bold(x)	**x**
sqrt(x)	$\sqrt{x}$	bolditalic(x)	***x***
sqrt(x, y)	$\sqrt[y]{x}$	underline(x)	$\underline{x}$

```
##
## > i <- 0
##
## > draw.title.cell("Arithmetic Operators", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x + y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x - y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x * y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x / y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %+-% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %/% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %*% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %.% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(-x), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(+x), i, nr); i <- i + 1
##
```

```
## > draw.title.cell("Sub/Superscripts", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x[i]), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x^2), i, nr); i <- i + 1
##
## > draw.title.cell("Juxtaposition", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x * y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(paste(x, y, z)), i, nr); i <- i + 1
##
## > draw.title.cell("Radicals", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(sqrt(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(sqrt(x, y)), i, nr); i <- i + 1
##
## > draw.title.cell("Lists", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(list(x, y, z)), i, nr); i <- i + 1
##
## > draw.title.cell("Relations", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x == y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x != y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x < y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x <= y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x > y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x >= y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %~~% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %=~% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %==% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %prop% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %~% y), i, nr); i <- i + 1
##
```

```
## > draw.title.cell("Typeface", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(plain(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(italic(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(bold(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(bolditalic(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(underline(x)), i, nr); i <- i + 1
##
## > # Need fewer, wider columns for ellipsis ...
## > nr <- 20
##
## > nc <- 2
##
## > make.table(nr, nc)
```

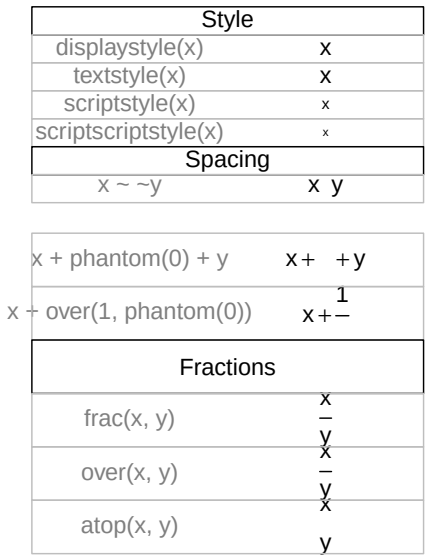
Ellipsis		**Arrows**	
list(x[1], ..., x[n])	$x_1, \ldots, x_n$	x %<–>% y	$x \leftrightarrow y$
x[1] + ... + x[n]	$x_1 + \cdots + x_n$	x %–>% y	$x \rightarrow y$
list(x[1], cdots, x[n])	$x_1, \cdots, x_n$	x %<–% y	$x \leftarrow y$
x[1] + ldots + x[n]	$x_1 + \ldots + x_n$	x %up% y	$x \uparrow y$
Set Relations		x %down% y	$x \downarrow y$
x %subset% y	$x \subset y$	x %<=>% y	$x \Leftrightarrow y$
x %subseteq% y	$x \subseteq y$	x %=>% y	$x \Rightarrow y$
x %supset% y	$x \supset y$	x %<=% y	$x \Leftarrow y$
x %supseteq% y	$x \supseteq y$	x %dblup% y	$x \Uparrow y$
x %notsubset% y	$x \not\subset y$	x %dbldown% y	$x \Downarrow y$
x %in% y	$x \in y$	**Symbolic Names**	
x %notin% y	$x \notin y$	Alpha – Omega	$A - \Omega$
Accents		alpha – omega	$\alpha - \omega$
hat(x)	$\hat{x}$	phi1 + sigma1	$\varphi + \varsigma$
tilde(x)	$\tilde{x}$	Upsilon1	Υ
ring(x)	$\mathring{x}$	infinity	∞
bar(xy)	$\overline{xy}$	32 * degree	32°
widehat(xy)	$\widehat{xy}$	60 * minute	$60'$
widetilde(xy)	$\widetilde{xy}$	30 * second	$30''$

```
## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
```

```
##
##
## > i <- 0
##
## > draw.title.cell("Ellipsis", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(list(x[1], ..., x[n])), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x[1] + ... + x[n]), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(list(x[1], cdots, x[n])), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x[1] + ldots + x[n]), i, nr); i <- i + 1
##
## > draw.title.cell("Set Relations", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %subset% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %subseteq% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %supset% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %supseteq% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %notsubset% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %in% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %notin% y), i, nr); i <- i + 1
##
## > draw.title.cell("Accents", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(hat(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(tilde(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(ring(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(bar(xy)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(widehat(xy)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(widetilde(xy)), i, nr); i <- i + 1
##
## > draw.title.cell("Arrows", i, nr); i <- i + 1
##
```

```
## > draw.plotmath.cell(expression(x %<->% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %->% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %<-% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %up% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %down% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %<=>% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %=>% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %<=% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %dblup% y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x %dbldown% y), i, nr); i <- i + 1
##
## > draw.title.cell("Symbolic Names", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(Alpha - Omega), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(alpha - omega), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(phi1 + sigma1), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(Upsilon1), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(infinity), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(32 * degree), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(60 * minute), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(30 * second), i, nr); i <- i + 1
##
## > # Need even fewer, wider columns for typeface and style ...
## > nr <- 20
##
## > nc <- 1
##
## > make.table(nr, nc)

## $mar
```

```
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 0
##
## > draw.title.cell("Style", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(displaystyle(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(textstyle(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(scriptstyle(x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(scriptscriptstyle(x)), i, nr); i <- i + 1
##
## > draw.title.cell("Spacing", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x ~~ y), i, nr); i <- i + 1
##
## > # Need fewer, taller rows for fractions ...
## > # cheat a bit to save pages
## > par(new = TRUE)
##
## > nr <- 10
##
## > nc <- 1
##
## > make.table(nr, nc)
```



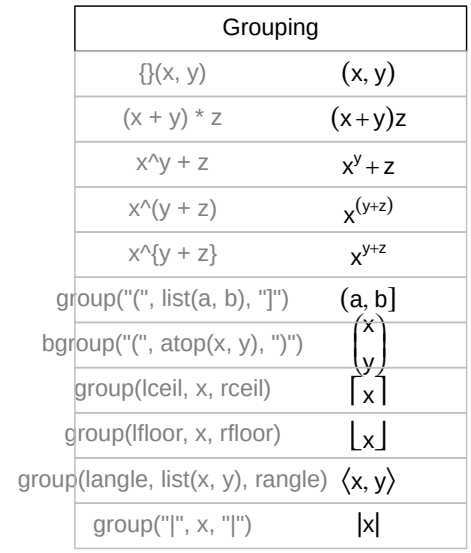

```
## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 4
##
## > draw.plotmath.cell(expression(x + phantom(0) + y), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x + over(1, phantom(0))), i, nr); i <- i + 1
##
## > draw.title.cell("Fractions", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(frac(x, y)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(over(x, y)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(atop(x, y)), i, nr); i <- i + 1
##
## > # Need fewer, taller rows and fewer, wider columns for big operators ...
## > nr <- 10
##
## > nc <- 1
```

```
##
## > make.table(nr, nc)
```

Big Operators	
sum(x[i], i = 1, n)	$\sum_{1}^{n} x_i$
prod(plain(P)(X == x), x)	$\prod_{x} P(X = x)$
integral(f(x) * dx, a, b)	$\int_{a}^{b} f(x)dx$
union(A[i], i == 1, n)	$\bigcup_{i=1}^{n} A_i$
intersect(A[i], i == 1, n)	$\bigcap_{i=1}^{n} A_i$
lim(f(x), x %–>% 0)	$\lim_{x \to 0} f(x)$
min(g(x), x >= 0)	$\min_{x \geq 0} g(x)$
inf(S)	$\inf S$
sup(S)	$\sup S$

```
## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 0
##
## > draw.title.cell("Big Operators", i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(sum(x[i], i=1, n)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(prod(plain(P)(X == x), x)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(integral(f(x) * dx, a, b)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(union(A[i], i==1, n)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(intersect(A[i], i==1, n)), i, nr); i <- i + 1
##
```

```
## > draw.plotmath.cell(expression(lim(f(x), x %->% 0)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(min(g(x), x >= 0)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(inf(S)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(sup(S)), i, nr); i <- i + 1
##
## > nr <- 12
##
## > nc <- 1
##
## > make.table(nr, nc)
```




```
## $mar
## [1] 0 0 0 0
##
## $pty
## [1] "s"
##
##
## > i <- 0
##
## > draw.title.cell("Grouping", i, nr); i <- i + 1
```

```
##
## > # Those involving '{ . }' have to be done "by hand"
## > draw.plotmath.cell(expression({}(x , y)), i, nr, string="{}(x, y)"); i <- i + 1
##
## > draw.plotmath.cell(expression((x + y)*z), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x^y + z),   i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x^(y + z)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(x^{y + z}), i, nr, string="x^{y + z}"); i <- i + 1
##
## > draw.plotmath.cell(expression(group("(", list(a, b), "]")), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(bgroup("(", atop(x, y), ")")), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(group(lceil, x, rceil)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(group(lfloor, x, rfloor)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(group(langle, list(x, y), rangle)), i, nr); i <- i + 1
##
## > draw.plotmath.cell(expression(group("|", x, "|")), i, nr); i <- i + 1
##
## > par(oldpar)
```

2.5 Object-Oriented Language

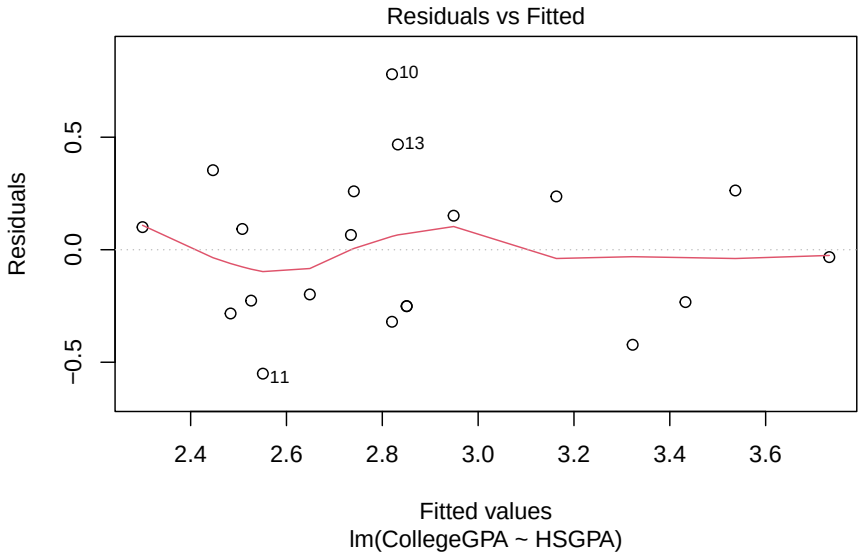
Functions are typically designed to operate on only one or very few classes of objects. However, some functions, like `summary()`, are **generic**, in the sense that essentially different versions of them have been constructed to work with different classes of objects.

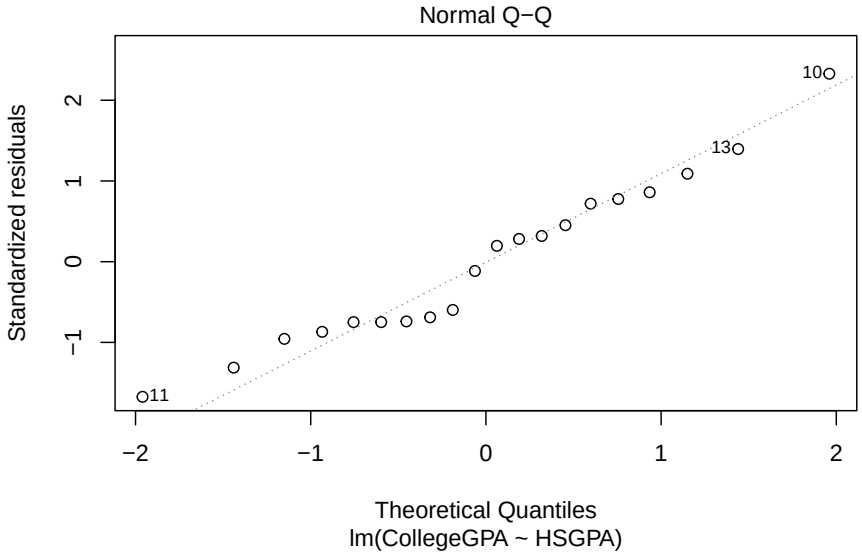
When a generic function is run with an object, R first checks the object's class type and then looks to find a method function with the name format `<generic function>.<class name>`. Below are examples for `summary()`:

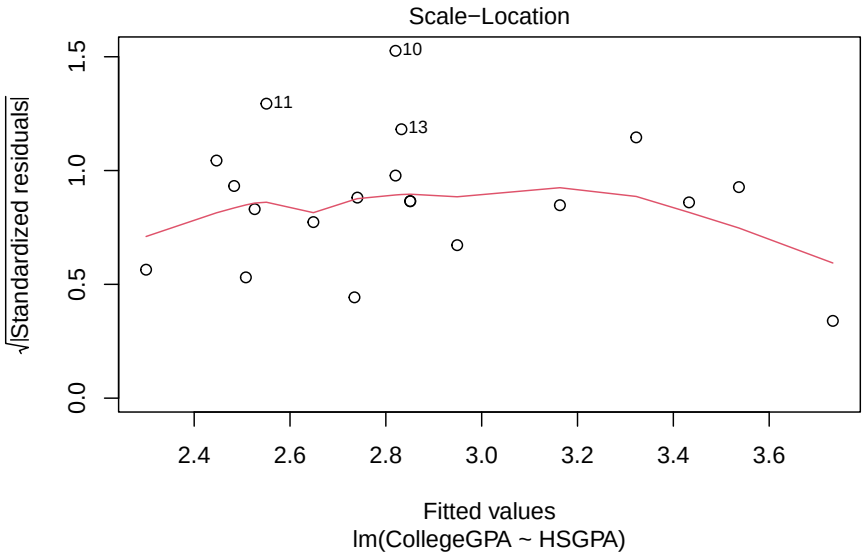
- summary(mod.fit) – The function `summary.lm()` summarizes the regression model
- summary(gpacsv) – The function `summary.data.frame()` summarizes the data frame's contents
- summary.default() – R attempts to run this function if there is no method function for a class

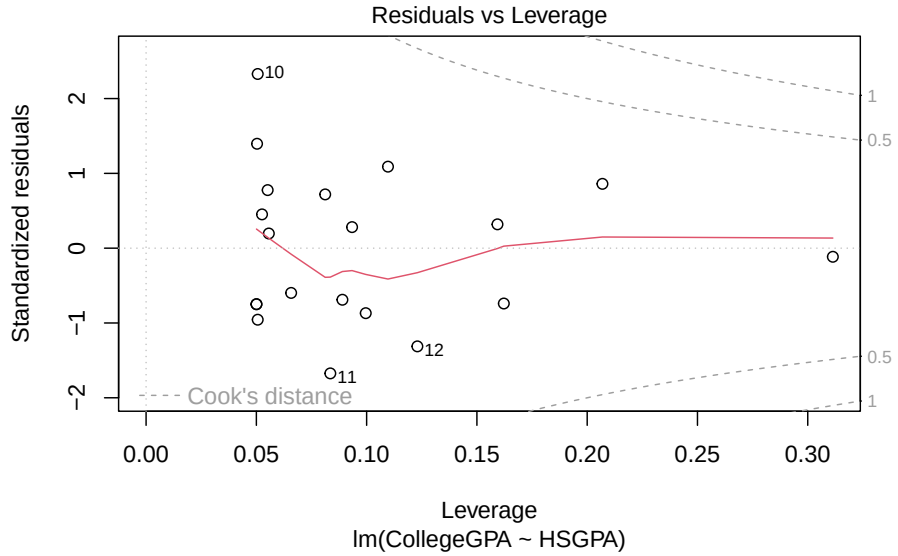
There are many generic functions! For example, `plot()` is a generic function (try`plot(mod.fit)` to see what happens!). We will also see other generic functions like `predict()` later in the notes.

```
plot(mod.fit)
```









The purpose of generic functions is to use a familiar language set with any object. So it is convenient to use the same language set no matter the application. This is why R is referred to as an object-oriented language.

To see a list of all method functions associated with a class, use `methods(class = <class name>)`. For the regression example, the method functions associated

with the `lm` class are:

```
methods(class="lm") %>% head()
```

```
## [1] "add1.lm"                    "alias.lm"
## [3] "anova.lm"                   "case.names.lm"
## [5] "coerce,oldClass,S3-method" "confint.lm"
```

To see a list of all method functions for a generic function, use `methods(generic.function = <generic function name>)`

```
methods(generic.function = "summary") %>% head()
```

```
## [1] "summary.aov"                    "summary.aovlist"
## [3] "summary.aspell"                 "summary.check_packages_in_dir"
## [5] "summary.connection"             "summary.data.frame"
```

Knowing what a name of a particular method function can be helpful to find help on it. For example, the help for `summary()` alone is not very helpful! However, the help for `summary.lm()`provides a lot of useful information about what is summarized for a regression model.

Chapter 3

Plotting

In this chapter, we will go over some *Time Series* examples. The aim of this chapter is to help you grasp some of the ideas about plotting.

3.1 Example Data

Click OSU_enroll.csv to download data.

```
osu.enroll <- read.csv(file = "OSU_enroll.csv",
    stringsAsFactors = TRUE)
```

```
head(osu.enroll)
```

```
##   t Semester Year Enrollment      date
## 1 1     Fall 1989      20110 8/31/1989
## 2 2   Spring 1990      19128  2/1/1990
## 3 3   Summer 1990       7553  6/1/1990
## 4 4     Fall 1990      19591 8/31/1990
## 5 5   Spring 1991      18361  2/1/1991
## 6 6   Summer 1991       6702  6/1/1991
```

```
tail(osu.enroll)
```

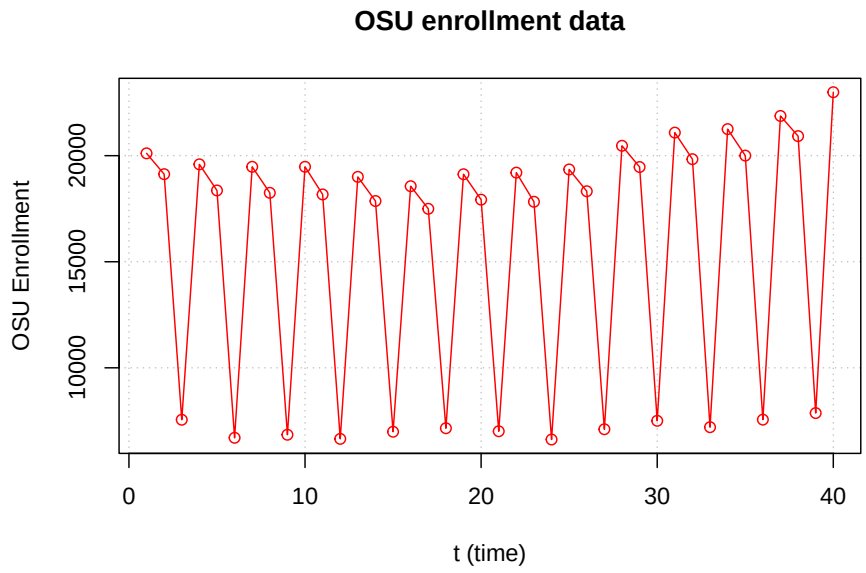
```
##     t Semester Year Enrollment      date
## 35 35   Spring 2001      20004  2/1/2001
## 36 36   Summer 2001       7558  6/1/2001
## 37 37     Fall 2001      21872 8/31/2001
```

```
## 38 38   Spring 2002      20922  2/1/2002
## 39 39   Summer 2002       7868  6/1/2002
## 40 40     Fall 2002      22992 8/31/2002
```

```
x <- osu.enroll$Enrollment
```

```
#One way to do plot
dev.new(width = 8, height = 6, pointsize = 10)

# we did not specify y-axis and R put our x in y-axis, time in  x-axis

plot(x = x, ylab = "OSU Enrollment",
       xlab = "t (time)", type="l", col = "red",
       main = "OSU Enrollment from Fall 1989 to Fall 2002",
       panel.first = grid(col = "gray", lty = "dotted"))
points(x = x, pch = 20, col = "blue")
```

```
# A little different version of the plot
  plot(x = x, ylab = "OSU Enrollment", type = "o", xlab = "t (time)", col = "red",
    main = "OSU enrollment data", panel.first = grid(col = "gray", lty = "dotted"))
```

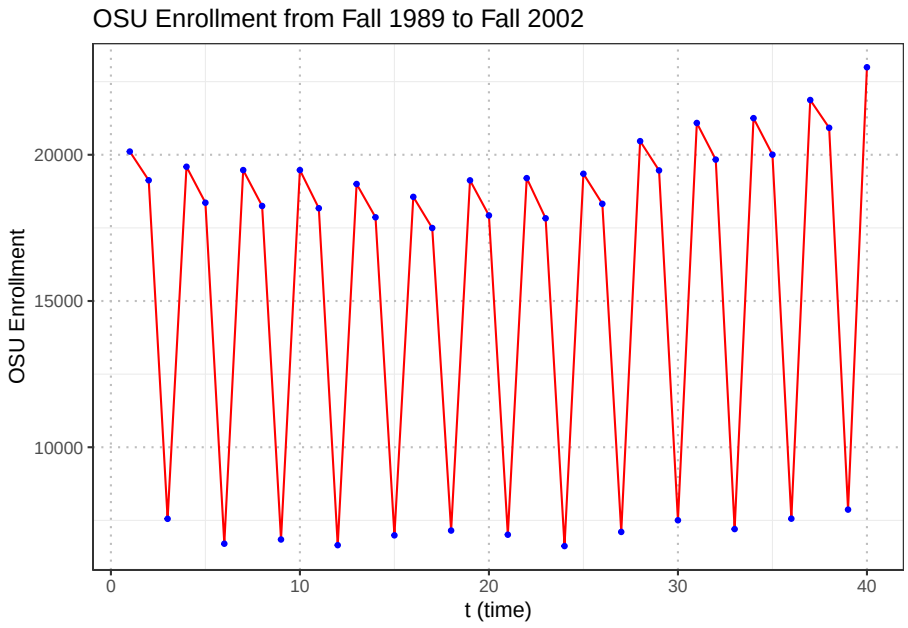



```
dev.new(width = 8, height = 6, pointsize = 10)
```

```
# we did not specify y-axis and R put our x in y-axis, time in  x-axis

plot(x = x, ylab = "OSU Enrollment",
       xlab = "t (time)", type="l", col = "red",
       main = "OSU Enrollment from Fall 1989 to Fall 2002",
       panel.first = grid(col = "gray", lty = "dotted"))

points(x = osu.enroll$Enrollment, pch = 20, col = "blue")
```

Altenatively, you can do the same thing using ggplot.

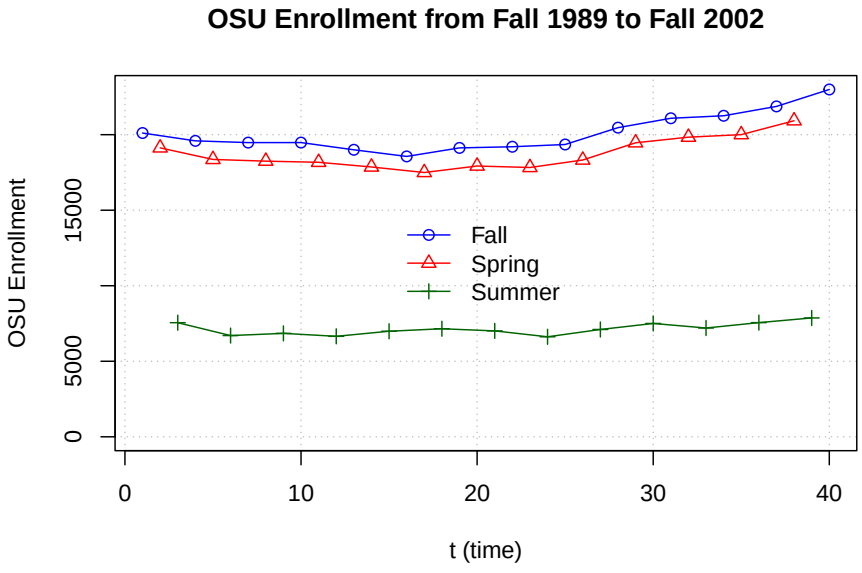
```
library(ggplot2)

# Create a data frame
df <- data.frame(osu.enroll)

# Create the plot
ggplot(df, aes(x = t, y = Enrollment)) +
  geom_line(colour = "red") +  # Line plot
  geom_point(shape = 20, colour = "blue") +  # Add points
  labs(x = "t (time)", y = "OSU Enrollment",
       title = "OSU Enrollment from Fall 1989 to Fall 2002") +  # Set axis labels and title
  theme_bw() +  # Set the theme to a white background with black lines
  theme(panel.grid.major = element_line(colour = "gray", linetype = "dotted"))  # Add gray dotte
```



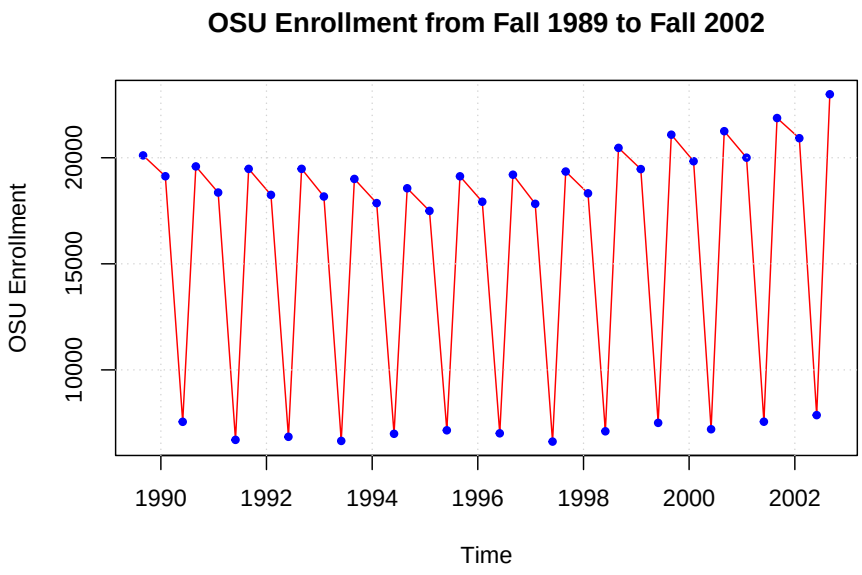
When only x is specified in the `plot()` function, R puts this on the y-axis and uses the observation number on the x-axis.

Compare this to the next plot below where both x and y arguments are specified.

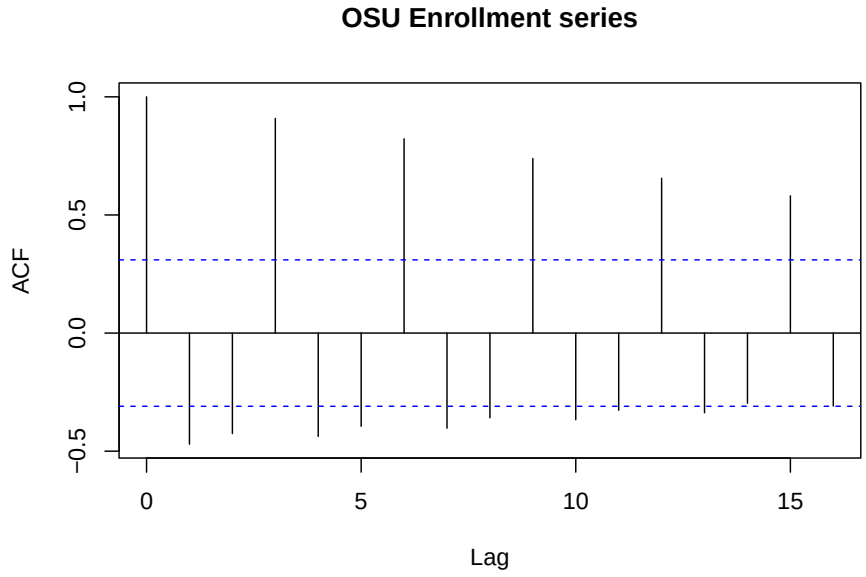
```
#More complicated plot
fall <- osu.enroll[osu.enroll$Semester == "Fall",]
spring <- osu.enroll[osu.enroll$Semester == "Spring",]
summer <- osu.enroll[osu.enroll$Semester == "Summer",]

plot(y = fall$Enrollment, x = fall$t,
    ylab = "OSU Enrollment", xlab = "t (time)",
    col = "blue",
    main = "OSU Enrollment from Fall 1989 to Fall 2002",
    panel.first = grid(col = "gray", lty = "dotted"),
    pch = 1, type = "o", ylim = c(0,max(osu.enroll$Enrollment)))

lines(y = spring$Enrollment, x = spring$t, col = "red",
    type = "o", pch = 2)

lines(y = summer$Enrollment, x = summer$t, col =
    "darkgreen", type = "o", pch = 3)

legend(x="center", legend= c("Fall","Spring","Summer"), pch=c(1,2,3), lty=c(1,1,1), co
```



```
#Another way to do plot with actual dates
plot(y = osu.enroll$Enrollment,
    x = as.Date(osu.enroll$date, format = "%m/%d/%Y"),
    xlab = "Time", type = "l", col = "red",
    main = "OSU Enrollment from Fall 1989 to Fall 2002",
    ylab = "OSU Enrollment")

points(y = osu.enroll$Enrollment,
    x = as.Date(osu.enroll$date, format = "%m/%d/%Y"), pch
    = 20, col = "blue")

#Create own gridlines
# v specifies vertical line; h specifies horizontal line
 abline(v = as.Date(c("1990/1/1", "1992/1/1", "1994/1/1",
    "1996/1/1", "1998/1/1", "2000/1/1", "2002/1/1")),
    lty = "dotted", col = "lightgray")
 abline(h = c(10000, 15000, 20000), lty = "dotted", col =
    "lightgray")
```




```
# Autocorrelation

rho.x <- acf(x = x, type = "correlation", main = "OSU Enrollment series")
```




```
rho.x
```

```
##
## Autocorrelations of series 'x', by lag
##
##      0      1      2      3      4      5      6      7      8      9     10
##  1.000 -0.470 -0.425  0.909 -0.438 -0.395  0.822 -0.403 -0.358  0.739 -0.367
##     11     12     13     14     15     16
## -0.327  0.655 -0.337 -0.297  0.581 -0.309
```

```
rho.x$acf[1:9]
```

```
## [1]  1.0000000 -0.4702315 -0.4253427  0.9087421 -0.4377336 -0.3946048  0.8224660
## [8] -0.4025871 -0.3584216
```

3.2 S&P500 Index

Click SP500weekly.csv to download data.

```
SP500 <- read.csv(file="SP500weekly.csv",stringsAsFactors = TRUE)
```

```
head(SP500)
```

```
##   WeekStart   Open   High    Low  Close AdjClose     Volume
## 1  1/1/1995 459.21 462.49 457.20 460.68   460.68 1199080000
## 2  1/8/1995 460.67 466.43 458.65 465.97   465.97 1627330000
## 3 1/15/1995 465.97 470.43 463.99 464.78   464.78 1667400000
## 4 1/22/1995 464.78 471.36 461.14 470.39   470.39 1628110000
## 5 1/29/1995 470.39 479.91 467.49 478.65   478.65 1888560000
## 6  2/5/1995 478.64 482.60 478.36 481.46   481.46 1579920000
```

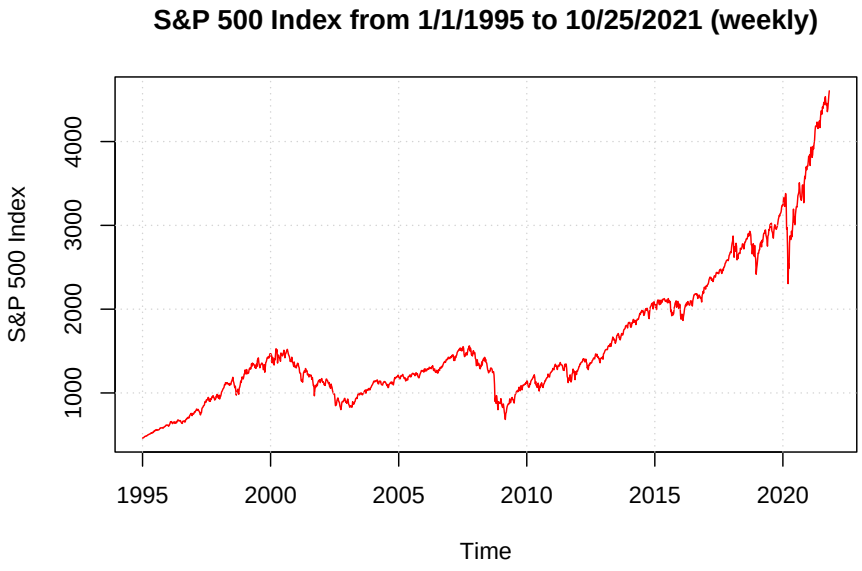
```
tail(SP500)
```

```
##       WeekStart    Open    High     Low   Close AdjClose      Volume
## 1395  9/19/2021 4402.95 4465.40 4305.91 4455.48  4455.48 15697030000
## 1396  9/26/2021 4442.12 4457.30 4288.52 4357.04  4357.04 15555390000
## 1397  10/3/2021 4348.84 4429.97 4278.94 4391.34  4391.34 14795520000
## 1398 10/10/2021 4385.44 4475.82 4329.92 4471.37  4471.37 13758090000
## 1399 10/17/2021 4463.72 4559.67 4447.47 4544.90  4544.90 13966070000
## 1400 10/24/2021 4553.69 4608.08 4537.36 4605.38  4605.38 16206040000
```

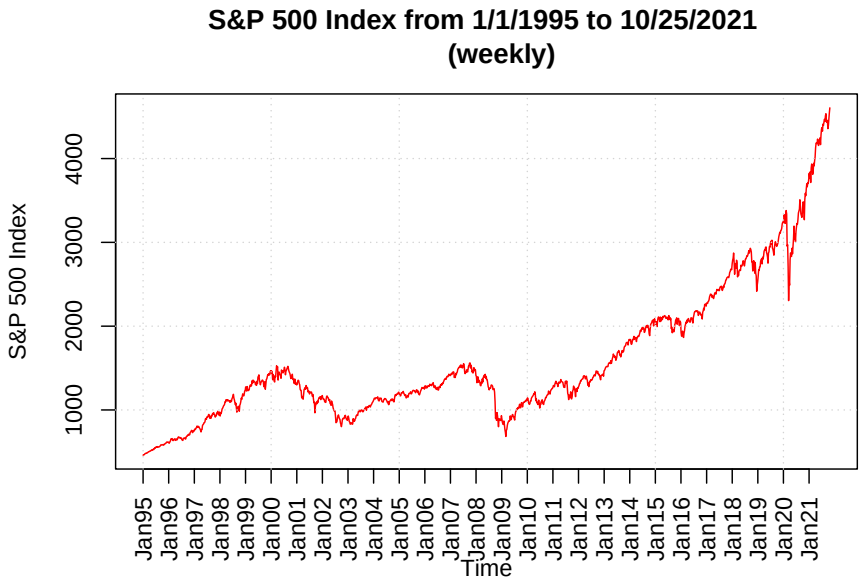
```
x <- SP500$Close
```

```
#One way to do plot
dev.new(width = 8, height = 6, pointsize = 10)
#again, we do not specify y-axis here
plot(x = x, ylab = "S&P 500 Index", xlab = "t (time)",
    type = "l", col = "red", main = "S&P 500 Index from
    1/1/1995 to 10/25/2021 (weekly)",
    panel.first = grid(col = "gray", lty = "dotted"))
```

```
#Another way to do plot with actual dates
plot(y = x, x = as.Date(SP500$WeekStart, format =
    "%m/%d/%Y"), xlab = "Time", type = "l", col = "red", main
    = "S&P 500 Index from 1/1/1995 to 10/25/2021 (weekly)",
    ylab = "S&P 500 Index")

#Create own gridlines
abline(v = as.Date(c("1995/1/1", "2000/1/1", "2005/1/1",
    "2010/1/1", "2015/1/1", "2020/1/1")), lty = "dotted",
    col = "lightgray")

abline(h = seq(from = 0, to = 5000, by = 1000), lty =
    "dotted", col = "lightgray")
```




```
# One more way with fine control of the dates
plot(y = x, x = as.Date(SP500$WeekStart, format =
    "%m/%d/%Y"), xlab = "Time", type = "l", col = "red",
    main = "S&P 500 Index from 1/1/1995 to 10/25/2021
    (weekly)", ylab = "S&P 500 Index", xaxt = "n")

axis.Date(side = 1, at = seq(from = as.Date("1995/1/1"),
    to = as.Date("2021/12/31"), by = "years"), labels =
    format(x = seq(from = as.Date("1995/1/1"), to =
    as.Date("2021/12/31"), by = "years"), format = "%b%y"),
    las = 2)  #las changes orientation of labels

#Create own gridlines
abline(v = as.Date(c("1995/1/1", "2000/1/1", "2005/1/1",
    "2010/1/1", "2015/1/1", "2020/1/1")), lty = "dotted",
    col = "lightgray")
abline(h = seq(from = 0, to = 5000, by = 1000), lty =
    "dotted", col = "lightgray")
```



3.3 Sunspots

Click SN_y_tot_V2.0.csv to download data.

```
sunspots <- read.table(file = "SN_y_tot_V2.0.csv", sep =
    ";", col.names = c("Mid.year", "Mean.total",
   "Mean.SD.total", "Numb.obs.used", "Definitive"))
```

```
head(sunspots)
```

```
##   Mid.year Mean.total Mean.SD.total Numb.obs.used Definitive
## 1   1700.5        8.3            -1            -1          1
## 2   1701.5       18.3            -1            -1          1
## 3   1702.5       26.7            -1            -1          1
## 4   1703.5       38.3            -1            -1          1
## 5   1704.5       60.0            -1            -1          1
## 6   1705.5       96.7            -1            -1          1
```

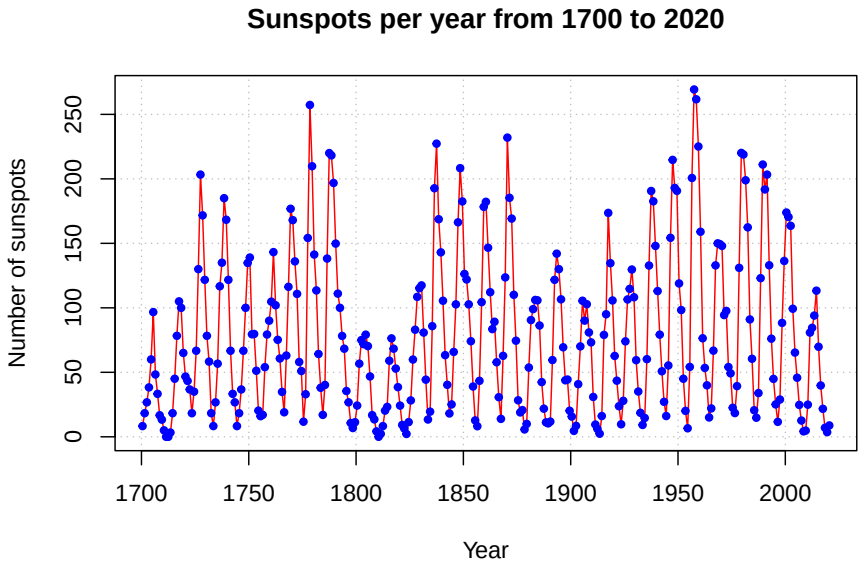
```
tail(sunspots)
```

```
##     Mid.year Mean.total Mean.SD.total Numb.obs.used Definitive
```

```
## 316   2015.5       69.8          6.4          8903           1
## 317   2016.5       39.8          3.9          9940           1
## 318   2017.5       21.7          2.5         11444           1
## 319   2018.5        7.0          1.1         12611           1
## 320   2019.5        3.6          0.5         12884           1
## 321   2020.5        8.8          4.1         14440           1
```

```
dev.new(width = 8, height = 6, pointsize = 10)

#again, we did not specify y-axis here
plot(x = sunspots$Mean.total, ylab = "Number of
    sunspots", xlab = "t (time)", type = "l", col = "red",
    main = "Sunspots per year from 1700 to 2020",
    panel.first = grid(col = "gray", lty = "dotted"))

points(x = sunspots$Mean.total, pch = 20, col = "blue")
```

```
# Include dates
plot(y = sunspots$Mean.total, x = sunspots$Mid.year, ylab
    = "Number of sunspots", xlab = "Year", type = "l", col
    = "red", main = "Sunspots per year from 1700 to 2020",
    panel.first = grid(col = "gray", lty = "dotted"))

points(y = sunspots$Mean.total, x = sunspots$Mid.year,
    pch = 20, col = "blue")
```



```
#Convert to an object of class "ts"

x <- ts(data = sunspots$Mean.total, start = 1700, frequency
    = 1)

x
```

```
## Time Series:
## Start = 1700
## End = 2020
## Frequency = 1
##   [1]   8.3  18.3  26.7  38.3  60.0  96.7  48.3  33.3  16.7  13.3   5.0   0.0
##  [13]   0.0   3.3  18.3  45.0  78.3 105.0 100.0  65.0  46.7  43.3  36.7  18.3
##  [25]  35.0  66.7 130.0 203.3 171.7 121.7  78.3  58.3  18.3   8.3  26.7  56.7
##  [37] 116.7 135.0 185.0 168.3 121.7  66.7  33.3  26.7   8.3  18.3  36.7  66.7
##  [49] 100.0 134.8 139.0  79.5  79.7  51.2  20.3  16.0  17.0  54.0  79.3  90.0
##  [61] 104.8 143.2 102.0  75.2  60.7  34.8  19.0  63.0 116.3 176.8 168.0 136.0
##  [73] 110.8  58.0  51.0  11.7  33.0 154.2 257.3 209.8 141.3 113.5  64.2  38.0
##  [85]  17.0  40.2 138.2 220.0 218.2 196.8 149.8 111.0 100.0  78.2  68.3  35.5
##  [97]  26.7  10.7   6.8  11.3  24.2  56.7  75.0  71.8  79.2  70.3  46.8  16.8
## [109]  13.5   4.2   0.0   2.3   8.3  20.3  23.2  59.0  76.3  68.3  52.9  38.5
## [121]  24.2   9.2   6.3   2.2  11.4  28.2  59.9  83.0 108.5 115.2 117.4  80.8
## [133]  44.3  13.4  19.5  85.8 192.7 227.3 168.7 143.0 105.5  63.3  40.3  18.1
## [145]  25.1  65.8 102.7 166.3 208.3 182.5 126.3 122.0 102.7  74.1  39.0  12.7
## [157]   8.2  43.4 104.4 178.3 182.2 146.6 112.1  83.5  89.2  57.8  30.7  13.9
## [169]  62.8 123.6 232.0 185.3 169.2 110.1  74.5  28.3  18.9  20.7   5.7  10.0
## [181]  53.7  90.5  99.0 106.1 105.8  86.3  42.4  21.8  11.2  10.4  11.8  59.5
## [193] 121.7 142.0 130.0 106.6  69.4  43.8  44.4  20.2  15.7   4.6   8.5  40.8
## [205]  70.1 105.5  90.1 102.8  80.9  73.2  30.9   9.5   6.0   2.4  16.1  79.0
## [217]  95.0 173.6 134.6 105.7  62.7  43.5  23.7   9.7  27.9  74.0 106.5 114.7
## [229] 129.7 108.2  59.4  35.1  18.6   9.2  14.6  60.2 132.8 190.6 182.6 148.0
## [241] 113.0  79.2  50.8  27.1  16.1  55.3 154.3 214.7 193.0 190.7 118.9  98.3
## [253]  45.0  20.1   6.6  54.2 200.7 269.3 261.7 225.1 159.0  76.4  53.4  39.9
## [265]  15.0  22.0  66.8 132.9 150.0 149.4 148.0  94.4  97.6  54.1  49.2  22.5
## [277]  18.4  39.3 131.0 220.1 218.9 198.9 162.4  91.0  60.5  20.6  14.8  33.9
## [289] 123.0 211.1 191.8 203.3 133.0  76.1  44.9  25.1  11.6  28.9  88.3 136.3
## [301] 173.9 170.4 163.6  99.3  65.3  45.8  24.7  12.6   4.2   4.8  24.9  80.8
## [313]  84.5  94.0 113.3  69.8  39.8  21.7   7.0   3.6   8.8
```

```
class(x)
```

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

```
class(sunspots$Mean.total)
```

```
## [1] "numeric"
```

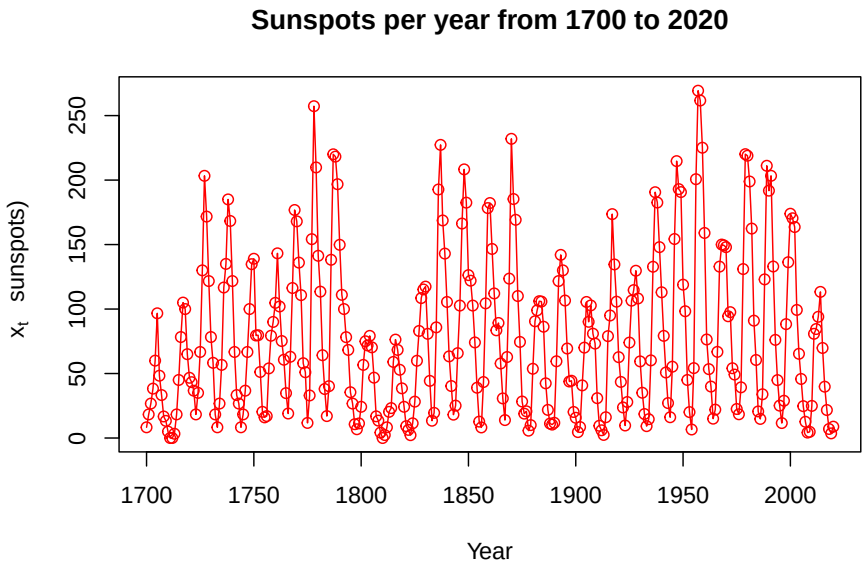
3.3.1 plot.ts()

plot() is a generic function - uses the plot.ts() method function

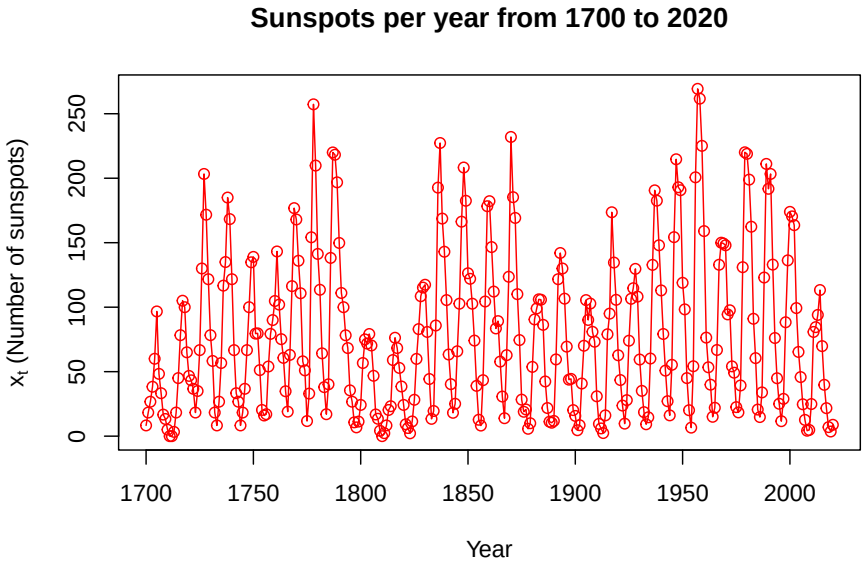
```
# we did not specify y-axis here, but x is now ts
plot(x = x, ylab = expression(paste(x[t], " (Number of
   sunspots)")), xlab = "Year", type = "o", col = "red", main
   = "Sunspots per year from 1700 to 2020")
```

```
## Warning in title(main = main, xlab = xlab, ylab = ylab, ...): font metrics
## unknown for character 0xa
```

```
## Warning in title(main = main, xlab = xlab, ylab = ylab, ...): font metrics
## unknown for character 0xa
```




```
plot.ts(x = x, ylab = expression(paste(x[t], " (Number of sunspots)")),
  xlab = "Year", type = "o", col = "red", main = "Sunspots per year from 1700 to 2020")
```




```
#type = "b" also works for "both" points and lines, but it leaves spaces between the points and l
```

Chapter 4

Basic Model

In this chapter, we will go introduce some basic *Time Series* model. Hopefully, we can discuss them in details in the following chapters.

Definition 4.1. ***Stochastic process***

Stochastic process is a collection of random variables $\{X_t\}$ indexed by t.

Time Series is a collection of random vatiables indexed according to the order they are obtained in time.

A ***realization*** of the stochastic process is the observed values.

4.1 White Noise

Example 4.1. ***White Noise***

$W_t \sim$ i.i.d.$(0, \sigma^2), \forall t = 1, ..., n$

usually, we assume normal distribution, i.e.,

$W_t \sim$ i.i.d.$N(0, \sigma^2), \forall t = 1, ..., n$

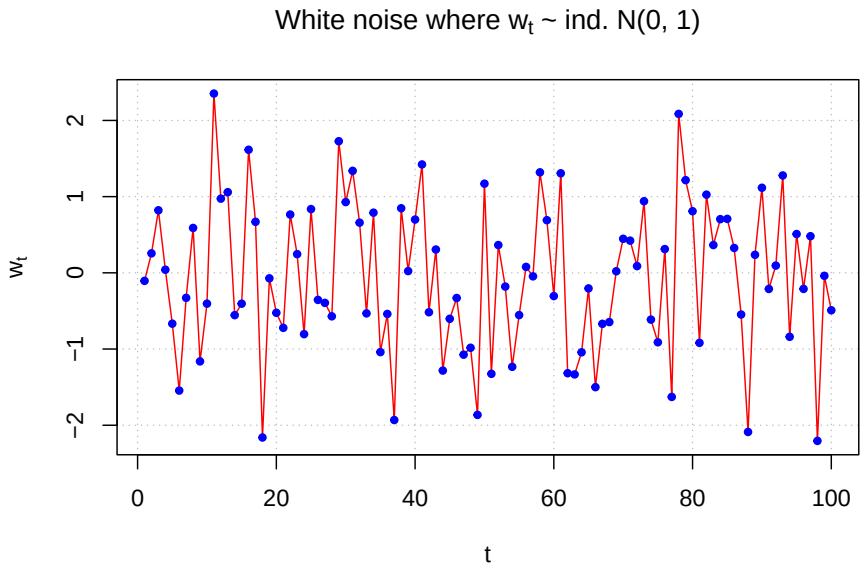
```
set.seed(8128)
w <- rnorm(n = 100, mean = 0, sd = 1)
head(w)
```

```
## [1] -0.10528941  0.25548490  0.82065388  0.04070997 -0.66722880 -1.54502793
```

```
dev.new(width = 6, height = 6, pointsize = 10)

# we did not specify y-axis here
# note that we use expression() to type math expression
plot(x = w, ylab = expression(w[t]), xlab = "t", type
    = "o", col = "red", main = expression(paste("White
    noise where ", w[t], " ~ ind. N(0, 1)")),
    panel.first = grid(col = "gray", lty = "dotted"))
```

```
## Warning in title(...): font metrics unknown for character 0xa

## Warning in title(...): font metrics unknown for character 0xa
```

```
#Advantage of second plot is separate control over color of points

plot(x = w, ylab = expression(w[t]), xlab = "t", type =
    "l", col = "red", main = expression(paste("White noise where ", w[t], " ~ ind. N(0
    panel.first = grid(col = "gray", lty = "dotted"))

points(x = w, pch = 20, col = "blue")
```

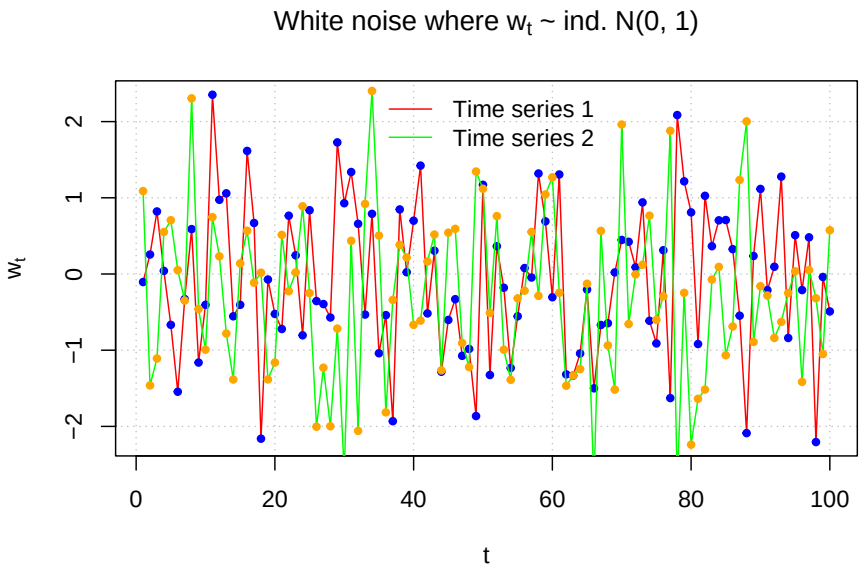


Suppose another white noise process is simulated. Below is a plot overlaying the two time series.

```
set.seed(1298)
w.new <- rnorm(n = 100, mean = 0, sd = 1)
head(w.new)
```

```
## [1]  1.08820292 -1.46217413 -1.10887422  0.55156914  0.70582813  0.05079594
```

```
par(mfrow=c(1,1))
plot(x = w,
     ylab = expression(w[t]), xlab = "t",
     type = "l", col = "red",
     main = expression(paste("White noise where ", w[t], " ~ ind. N(0, 1)")),
     panel.first = grid(col = "gray", lty = "dotted"))

points(x = w, pch = 20, col = "blue")

lines(x = w.new, col = "green")
points(x = w.new, pch = 20,col = "orange")
legend(x ="top",legend=c("Time series 1", "Time series 2"), lty=c(1,1), col=c("red", "green"),
      bty="n")
```



We could also plot the two time series separately.

```
dev.new(width = 8, height = 6, pointsize = 10) #Open a new plot window
#make frame by row 2 rows 1 cols
par(mfrow = c(2,1))
plot(x = w, ylab = expression(w[t]), xlab = "t", type =
      "l", col = "red", main = expression(paste("White noise where ", w[t], "~N(0, 1)"
      grid(col = "gray", lty = "dotted"))
points(x = w, pch = 20, col = "blue")


plot(x = w.new, ylab = expression(w.new[t]), xlab =
      "t", type = "l", col = "green", main =
      expression(paste("White noise where ", w[t], " ~ ind.N(0, 1)")), panel.first=gri
      "dotted"))
points(x = w.new, pch = 20, col = "orange")
```

```
# What if used plot.ts()?
dev.new(width = 8, height = 6, pointsize = 10) #Open a new plot window

plot.ts(x = cbind(w, w.new), ylab = expression(w[t]), xlab = "t", type = "o", col = "r

#Problem: gridlines do not extend to second plot

plot.ts(x = cbind(w, w.new), ylab = expression(w[t]), xlab = "t", type = "o", col = "r

grid(col = "gray", lty = "dotted")
#Problem: gridlines do not appear correctly on plots (could fix by specifying where to
```

4.2 Moving Average

Example 4.2. ***Moving Average of White Noise***

The previous time series had no correlation between the observations. One way to induce correlation is to create a "moving average" of the observations. This will have an effect of "smoothing" the series.

Let $m_t = \frac{w_t + w_{t-1} + w_{t-2}}{3}$. This can be done in R using the following code:

```
set.seed(8128)
w <- rnorm(n=100,mean=0, sd=1)
head(w)
```

```
## [1] -0.10528941  0.25548490  0.82065388  0.04070997 -0.66722880 -1.54502793
```

```
dev.new(width = 8, height = 6, pointsize = 10) #Open a new plot window
plot(x = w, ylab = expression(paste(m[t], " or ", w[t])), xlab = "t", type = "l", col = "red",
        panel.first = grid(col = "gray", lty = "dotted"), lty = "dotted")
  points(x = w, pch = 20, col = "blue")
```

```
# rep(1/3,3) repeats 1/3 3 times
# we can use filter() to generate moveing average
# filter() help us build linear combination of elts of w
# filter= linear combination coef
# convolution for moving average
# sides=1
m <- filter(w, filter = rep(1/3, 3), method="convolution", sides=1)

head(m)
```

```
## [1]          NA          NA  0.32361646  0.37228292  0.06471168 -0.72384892
```

```
tail(m)
```

```
## [1]  0.3158762 -0.1803096  0.2598066 -0.6450531 -0.5879723 -0.9120182
```

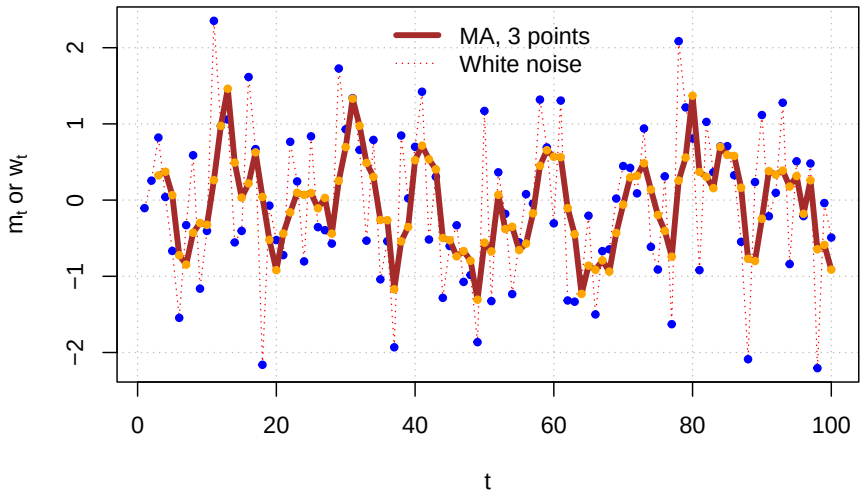
```
(w[1]+w[2]+w[3])/3
```

```
## [1] 0.3236165
```

```
(w[98]+w[99]+w[100])/3
```

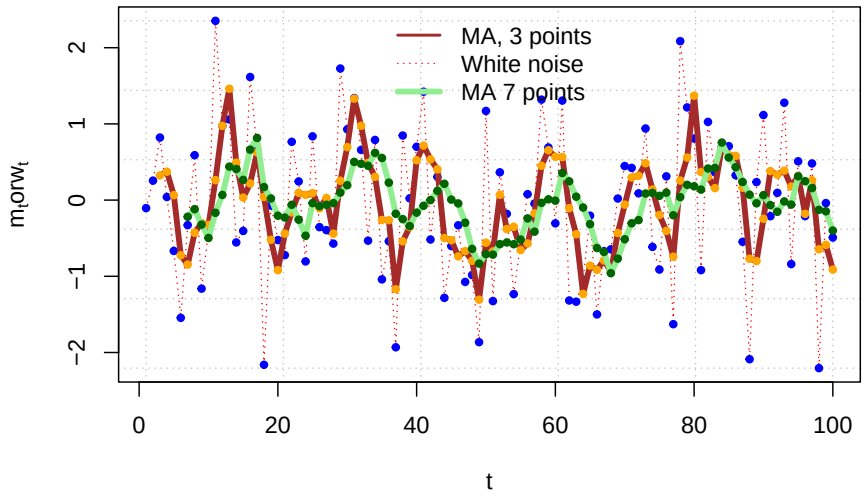
```
## [1] -0.9120182
```

```
plot(x = w, ylab = expression(paste(m[t], " or ", w[t])),
    xlab = "t", type = "l", col = "red", panel.first =
    grid(col = "gray", lty = "dotted"), lty = "dotted")
points(x = w, pch = 20, col = "blue")
lines(x = m, col = "brown", lty = "solid", lwd = 4)
points(x = m, pch = 20, col = "orange")
legend(x = "top", legend = c("MA, 3 points", "White noise"), lty = c("solid", "dotted"), col=c("b
    "red"), lwd = c(4,1), bty = "n")
```



The plot below shows a 7-point moving average.

```
m7 <- filter(x=w, filter=rep(x=1/7, times=7),method="convolution", sides=1)

plot(x=w, ylab=expression(paste(m[t],"or",w[t])), xlab="t",type="l",col="red",panel_fi

points(x=w, pch=20, col="blue")

lines(x=m, col="brown", lty="solid",lwd=4)

points(x=m, pch=20, col="orange")
lines(x = m7, col = "lightgreen", lty = "solid", lwd = 4)
points(x = m7, pch = 20, col = "darkgreen")

legend(x = "top", legend = c("MA, 3 points", "White noise", "MA 7 points"),
         lty = c("solid", "dotted", "solid"), col = c("brown", "red", "lightgreen"),
         lwd = c(2,1,4), bty="n")
```



4.3 Autoregression

Example 4.3. ***Autoregression***

An autoregression model uses past observations to predict future observations in a regression model.

Suppose the autoregression model is

$x_t = 0.7x_{t-1} + w_t, w_t \sim \text{i.i.d.} N(0,1), \forall t = 1, ..., n$

Because there is one past period on the right hand side, this is often denoted as an AR(1) model

Obviously, there will be a correlation between the random variables.

```
set.seed(6381)  #Different seed from white_noise.R
w <- rnorm(n = 200, mean = 0, sd = 1)
head(w)
```

```
## [1]  0.06737166 -0.68095839  0.78930605  0.60049855 -1.21297680 -1.14082872
```

```
#Simple way to simulate AR(1) data
x <- numeric(length = 200)
x.1 <- 0
```

```
for(i in 1:length(x)) {
    x[i] <- 0.7*x.1 + w[i]
    x.1 <- x[i]
  }
```

```
head(data.frame(x, w))
```

```
##             x           w
## 1  0.06737166  0.06737166
## 2 -0.63379823 -0.68095839
## 3  0.34564730  0.78930605
## 4  0.84245166  0.60049855
## 5 -0.62326064 -1.21297680
## 6 -1.57711117 -1.14082872
```

```
#Do not use first 100
x <- x[101:200]
```

```
dev.new(width = 8, height = 6, pointsize = 10)  #Opens up wider plot window than the d
plot(x = x, ylab = expression(x[t]), xlab = "t", type =
      "l", col = c("red"), lwd = 1 , main =
      expression(paste("AR(1): ", x[t] == 0.7*x[t-1] +
      w[t])), panel.first = grid(col = "gray", lty =
      "dotted"))
points(x = x, pch = 20, col = "blue")
```

```
 #Show first few observations after removal of first 100
  head(data.frame(c(NA, NA, x), w[99:200]))
```

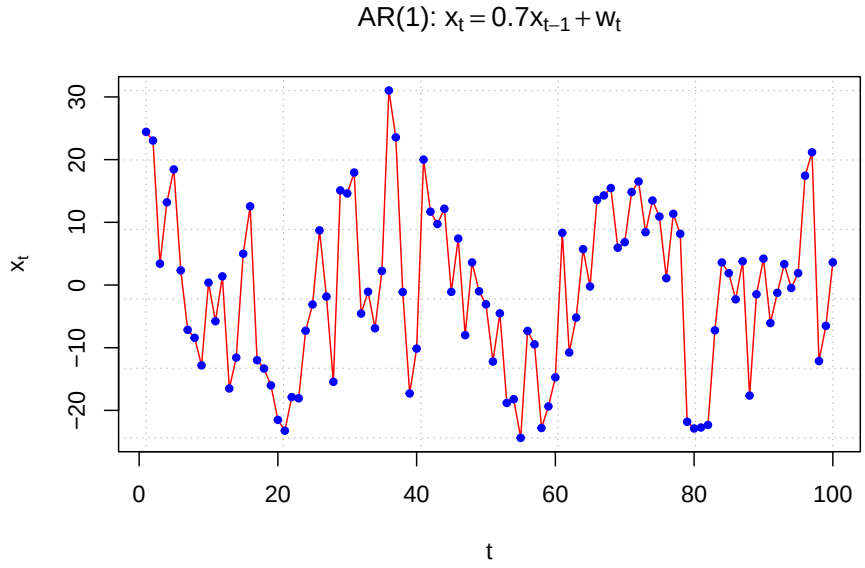
```
##   c.NA..NA..x.  w.99.200.
## 1           NA  0.3918531
## 2           NA -0.1980565
## 3     1.429572  0.7508375
## 4    -1.878239 -2.8789389
## 5    -1.470250 -0.1554829
## 6    -3.464078 -2.4349033
```

```
  #Correlation between x_t and x_t-1
  cor(x[2:100], x[1:99])
```

```
## [1] 0.7270483
```

Here is an easier way to simulate observations from an AR(1). Note that this uses an Autoregressive Integrated Moving Average (ARIMA) structure that we will discuss later in the course. In this case, I use $\sigma_w = 10$.

```
set.seed(7181)
x <- arima.sim(model = list(ar = c(0.7)), n = 100,
    rand.gen = rnorm, sd = 10)
plot(x = x, ylab = expression(x[t]), xlab = "t", type =
    "l", col = "red", lwd = 1 ,main =
    expression(paste("AR(1): ", x[t] == 0.7*x[t-1] +
    w[t])), panel.first=grid(col = "gray", lty =
    "dotted"))
points(x = x, pch = 20, col = "blue")
```



Chapter 5

Dependence

We would like to understand the relationship among all random variables in a time series. In order to do that, we would need to look at the joint distribution function.

Suppose the time series consists of the random variables . The cumulative joint distribution function for these random variables is:

$$F(c_1, c_2, ..., c_n) = P(X_1 \leq c_1, X_2 \leq c_2, ..., X_n \leq c_n)$$

This can be VERY difficult to examine over the MULTIDIMENSIONS.

The one-dimensional cumulative distributional function is denoted by $F_t(x) = P(X_t \leq x)$for a random variable X_t at time t. The corresponding probability distribution function is $f_t(x) = \frac{\partial F_t(x)}{\partial x}$

The mean value function is $\mu_t = E(X_t) = \int x f_t(x) dx$

Important: The interpretation of μ_t is that it represents the mean taken over ALL possible events that could have produced x_t. Another way to think about it is suppose that $x_1, \ldots, x_n$ is observed an infinite number of times. Then μ_1 represents the average value at time 1, μ_2 represents the average value at time 2, …

Example 5.1. Moving Average

Let $m_t = \frac{(w_t + w_{t-1} + w_{t-2})}{3}$, where $w_t \sim$ i.i.d.$N(0,1) \forall t = 1, ..., n$

Then $\mu_t = E(m_t) = E[\frac{(w_t + w_{t-1} + w_{t-2})}{3}] = \frac{1}{3}[E(w_t) + E(w_{t-1}) + E(w_{t-2})] = 0$

Example 5.2. Autoregressions

Let $x_t = 0.7x_{t-1} + w_t, w_t \sim$ i.i.d.$N(0,1), \forall t = 1, ..., n$

Then $\mu_t = E(x_t) = E[0.7x_{t-1} + w_t] = 0.7E(x_{t-1}) + E(w_t) = 0.7E(0.7x_{t-2} + w_{t-1}) + 0 = ... = 0$

5.1 Autocovariance function

To assess the dependence between two random variables, we need to examine the two-dimensional cumulative distribution function. This can be denoted as $F(c_s, c_t) = P(X_s \leq c_s, X_t \leq c_t)$for two different time points s and t.

In another course, you learned about the covariance function which measures the linear dependence between two random variables:For two random variables X and Y, the covariance between them is

$$Cov(X,Y) = E[(X–\mu_x)(Y–\mu_y)] = E(XY)–\mu_x\mu_y, \mu_x = E(X), \mu_y = E(Y)$$

Because we are interested in linear dependence between two random variables in the same time series, we will examine the autocovariance function:

$$\gamma(s,t) = Cov(X_s, X_t) = E[(X_s–\mu_s)(X_t–\mu_t)] = \int\int (x_s–\mu_s)(x_t–\mu_t)f(x_s,x_t)dx_s dx_t, \forall s,t$$

where$f(X_s, X_t) = \frac{\partial F(X_s,X_t)}{\partial X_s \partial X_t}$and assuming continuous random variables

- If the autocovariance is 0, there is no linear dependence.
- If s = t, the autocovariance is the variance:
  $\gamma(t,t) = E[(X_t - \mu_t)^2]$

Example 5.3. White Noise

Suppose $w_t \sim ind.N(0, \sigma_w^2), t = 1, ..., n.$

$$\begin{cases} \gamma(s,t) = \sigma_w^2 & \text{if } s = t \\ \gamma(s,t) = 0 & \text{if } s \neq t \end{cases}$$

Example 5.4. Moving Average

Let $m_t = \frac{w_t + w_{t-1} + w_{t-2}}{3}, w_t \sim ind.N(0,1), t = 1, ..., n$

$\gamma(s,t) = E[(m_s - \mu_s)(m_t - \mu_t)] = E[m_s m_t]$ b/c $\mu_s = \mu_t = 0$

Then $E[m_s m_t] = E[\frac{w_s + w_{s-1} + w_{s-2}}{3} \frac{w_t + w_{t-1} + w_{t-2}}{3}] = \frac{1}{9}E[(w_s + w_{s-1} + w_{s-2})(w_t + w_{t-1} + w_{t-2})]$

To find this, we need to examine a few different cases:

- s = t

$E[m_t m_t] = E[m_t^2] = Var(m_t) + [E(m_t)]^2 = \frac{1}{9} Var(w_t + w_{t-1} + w_{t-2}) + 0 = \frac{1}{9} Var(w_t) + Var(w_{t-1}) + Var(w_{t-2}) = \frac{1}{9}(1-$

- s = t - 1

$E[m_{t-1} m_t] = \frac{1}{9} E[(w_{t-1} + w_{t-2} + w_{t-3})(w_t + w_{t-1} + w_{t-2})] = \frac{1}{9}[E(w_{t-1})E(w_t) + E(w_{t-1}^2) + E(w_{t-1})E(w_{t-2}) + E(w_{t-2})E(w$

$E(w_{t-1}^2) = 1$, because $Var(w_{t-1}) = 1$

- s = t - 2

$E[m_{t-2} m_t] = \frac{1}{9} E[(w_{t-2} + w_{t-3} + w_{t-4})(w_t + w_{t-1} + w_{t-2})] = \frac{1}{9}[E(w_{t-2}w_t) + E(w_{t-2}w_{t-1}) + E(w_{t-2}w_{t-2}) + E(w_{t-3}w_t) + E$

- s = t – 3

$E[m_{t-3} m_t] = \frac{1}{9} E[(w_{t-3} + w_{t-4} + w_{t-5})(w_t + w_{t-1} + w_{t-2})] = \frac{1}{9} E[w_{t-3}w_t + w_{t-3}w_{t-1} + w_{t-3}w_{t-2} + w_{t-4}w_t + w_{t-4}w_{t-1} + w_{t-}$

Notice that s = t + 1, t + 2, … can be found in a similar manner. Also s = t - 4, t - 5,… can be found. In summary, the autocovariance function is

$$\begin{cases} \gamma(s,t) = \frac{3}{9} & \text{if } s = t \\ \gamma(s,t) = \frac{2}{9} & \text{if } |s-t| = 1 \\ \gamma(s,t) = \frac{1}{9} & \text{if } |s-t| = 2 \\ \gamma(s,t) = 0 & \text{if } |s-t| \geq 3 \end{cases}$$

Notes: - The word “lag” is used to denote the time separation between two values. For example, |s - t| = 1 denotes the lag is 1 and |s - t| = 2 denotes the lag is 2. We will use this “lag” terminology throughout this course.

- The autocovariance depends on the lag, but NOT individual times for the moving average example! This will be VERY, VERY important later!

5.2 Autocorrelation function (ACF)

In another course, the Pearson correlation coefficient was defined to be:

$$\rho = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$$

The reason the correlation coefficient is examined instead of the covariance is that it is always between –1 and 1.

Note the following:

- close to 1 means strong, positive linear dependence
- close to –1 means strong, negative linear dependence
- close to 0 means weak linear dependence.

The autocorrelation is the extension of the Pearson correlation coefficient to time series analysis. The autocorrelation function (ACF) is

$$\rho(s,t) = \frac{\gamma(s,t)}{\sqrt{\gamma(s,s)\gamma(t,t)}}$$

where s and t denote two time points. The ACF is also between –1 and 1 and has a similar interpretation as for correlation coefficient.

Notice that $\gamma(t,t)$ is the variance at time t.

Example 5.5. White Noise

Suppose $w_t \sim \text{ind}N(0,\sigma_w^2), t = 1,...,n$

$$\begin{cases} \rho(s,t) = \frac{3}{9} & \text{if } s = t \\ \rho(s,t) = \frac{2}{9} & \text{if } s \neq t \end{cases}$$

Example 5.6. Moving Average

Let $m_t = \frac{w_t + w_{t-1} + w_{t-2}}{3}, w_t \sim \text{ind}N(0,1), t = 1,...,n$

$$\begin{cases} \gamma(s,t) = \frac{3}{9} & \text{if } s = t \\ \gamma(s,t) = \frac{2}{9} & \text{if } |s-t| = 1 \\ \gamma(s,t) = \frac{1}{9} & \text{if } |s-t| = 2 \\ \gamma(s,t) = 0 & \text{if } |s-t| \geq 3 \end{cases}$$

$$\begin{cases} \rho(s,t) = 1 & \text{if } s = t \\ \rho(s,t) = \frac{2}{3} & \text{if } |s-t| = 1 \\ \rho(s,t) = \frac{1}{3} & \text{if } |s-t| = 2 \\ \rho(s,t) = 0 & \text{if } |s-t| \geq 3 \end{cases}$$

In par, if $|s-t|=1$, then $\rho(s,t)=\frac{2/9}{\sqrt{3/9}\sqrt{3/9}}=\frac{2}{3}$

5.3 Strong positive and negative linear dependence

- If there is strong positive linear dependence between x_s and x_t, the time series will appear smooth in a plot of the series versus time.
- If there is strong negative linear dependence between x_s and x_t, the time series will appear choppy in a plot of the series versus time.

Below are three plots illustrating these statements. I simulated data from different time series models. The autocorrelation for |s - t| = 1 is given for each model. The "estimated" autocorrelation, denoted by $\hat{\rho}(s,t)$, is given for that particular data set. The calculation of this estimate will be discussed later.

```
# Simulate a white noise series

set.seed(4599)
w <- rnorm(n = 100, mean = 0, sd = 1)
head(w)
```

```
## [1] 0.2822308 0.0552224 0.9433917 0.6376982 1.4084635 0.2050981
```

1. $\rho(s,t)=0.4972, \hat{\rho}(s,t)=0.4705$ for |s - t| = 1

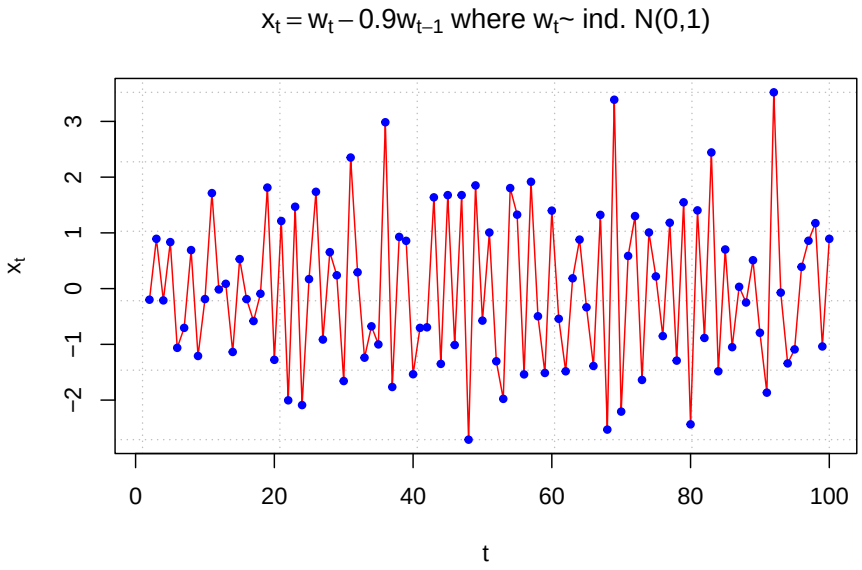
```
dev.new(width = 8, height = 6, pointsize = 10)
#Opens up wider plot window than the default (good for time series plots)

#rho = 0.4972376 we will discuss why this is the numerical value later
x <- filter(x = w, filter = c(1,0.9), method = "convolution", sides = 1)

plot(x = x, ylab = expression(x[t]), xlab = "t", type = "l", col = "red", lwd = 1 ,main = express

points(x = x, pch = 20, col = "blue")

#Pearson correlation between x_t and x_t-1
cor(x = x[3:100], y = x[2:99])
```

```
## [1] 0.4704674
```

```
par(pty = "s") # square plot

#        plotting window aspect ratio    pty
# "s"

plot(x = x[3:100], y = x[2:99], ylab = expression(x[t-1]),
    xlab = expression(x[t]), col = "red", type = "p",
    main = expression(paste(x[t-1], "vs.", x[t])),
    panel.first=grid(col = "gray", lty = "dotted"))

par(pty = "m") # Return to normal
```

2. $\rho(s,t) = -0.4972, \hat{\rho}(s,t) = -0.5725$ for |s - t| = 1

```
#rho = -0.4972376
x <- filter(x = w, filter = c(1,-0.9), method = "convolution", sides = 1)

plot(x = x, ylab = expression(x[t]), xlab = "t", type = "l", col = "red", lwd = 1 ,mai

points(x = x, pch = 20, col = "blue")
```

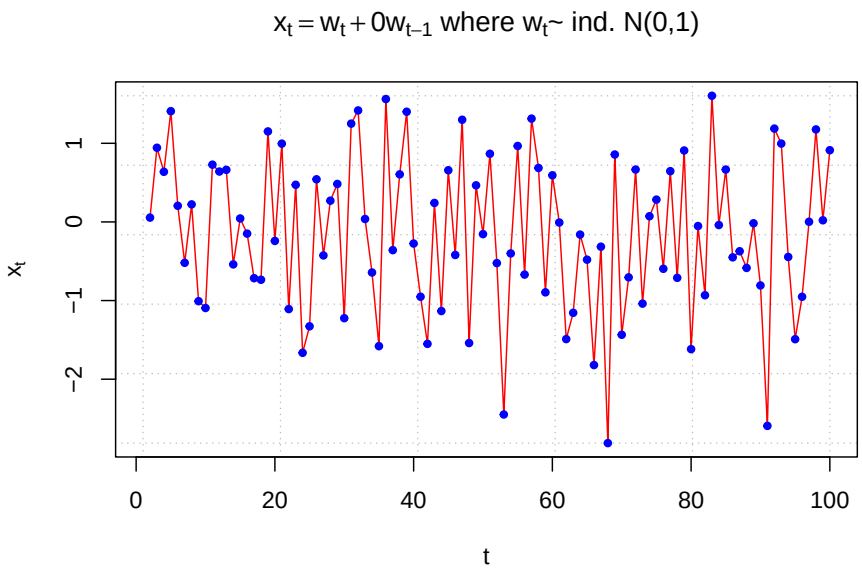



```
cor(x = x[3:100], y = x[2:99])
```

```
## [1] -0.5725393
```

3. $\rho(s,t) = 0, \hat{\rho}(s,t) = -0.1054$ for |s - t| = 1

```
#rho = 0
x <- filter(x = w, filter = c(1,0), method = "convolution", sides = 1)

plot(x = x, ylab = expression(x[t]), xlab = "t", type = "l", col = "red", lwd = 1 ,main = express

points(x = x, pch = 20, col = "blue")
```




```
cor(x = x[3:100], y = x[2:99])
```

```
## [1] -0.1053843
```

```
cor.test(x = x[3:100], y = x[2:99]) #Not significant
```

```
##
##  Pearson's product-moment correlation
##
## data:  x[3:100] and x[2:99]
## t = -1.0383, df = 96, p-value = 0.3017
```

```
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.29758249  0.09502345
## sample estimates:
##        cor
## -0.1053843
```

Plot 1. is the least choppy (jagged) and plot 2. is the most choppy. Remember what a correlation means.

A positive correlation means that "large" values tend to occur with other "large" values and "small" values tend to occur with other "small" values.

A negative correlation means that "large" values tend to occur with other "small" values and "small" values tend to occur with other "large" values.

Chapter 6

Stationarity

6.1 Stationarity

Stationarity is a VERY important concept to understand because it allow us to construct time series models.

Definition 6.1. Strictly Stationary

The probabilistic behavior of $X_{t_1}, ..., X_{t_k}$ is exactly the same as that of the shifted set $X_{t_{1+h}}, ..., X_{t_{k+h}}$ for ANY collection of time points t1,..., tk, for ANY k = 1, 2, ..., and for ANY shift h = 0, ± 1, ± 2, ...

Let $c_1, ..., c_k$ be constants. Then

$$P(X_{t_1} \leq c_1, ..., X_{t_k} \leq c_k) = P(X_{t_{1+h}} \leq c_1, ..., X_{t_{k+h}} \leq c_k)$$

i.e., the probability distribution is INVARIANT to time shifts.

e.g. $P(X_1 \leq c_1, X_2 \leq c_2) = P(X_{10} \leq c_1, X_{11} \leq c_2)$

Requiring a time series to be strictly stationary is VERY restrictive! A less restrictive requirement is weakly stationary.

Definition 6.2. Weakly Stationary

The first two moments (mean and covariance) of the time series are invariant to time shifts.

$$E(X_t) = \mu, \forall t$$

and

$$\gamma(t, t+h) = \gamma(0, h), \forall t$$

Notes:

- μ and $\gamma(0,h)$ are NOT functions of t.
- h is the lag
- $\gamma(t,t+h) = \gamma(0,h)$ for ALL time t means that the autocovariance function ONLY depends on the number of lags of the time shift. Thus,$\gamma(0,h) = \gamma(1,h+1) = \gamma(2,2+h) = \gamma(3,3+h) = ...$
- Because we will generally be dealing with a weakly stationary time series, we can make the following notational change: $\gamma(h) = \gamma(0,h) = \gamma(t,t+h)$.
- The variance of X_t is $\gamma(0)$.
- The same notational change can be made to the autocorrelation function (ACF). Thus, $\rho(h)$ denotes the ACF at lag h. Note that

$$\rho(h) = \rho(t,t+h) = \frac{\gamma(t,t+h)}{\sqrt{\gamma(t,t)}\sqrt{\gamma(t+h,t+h)}} = \frac{\gamma(h)}{\sqrt{\gamma(0)}\sqrt{\gamma(0)}} = \frac{\gamma(h)}{\gamma(0)}$$

- Strictly stationary implies weakly stationary, but the reverse is not necessarily true.
- Frequently, we will just say "stationary" to refer to weakly stationary and say the full "strictly stationary" to refer to strictly stationary.

Example 6.1. White Noise Suppose $w_t \sim ind.N(0,\sigma_w^2), t = 1,...,n$. Is this a weakly stationary time series?

Yes – mean is 0 for all wt, variance is constant for all wt, and the covariance is 0 because the random variables are independent. AND, it is also strictly stationary.

Example 6.2. Moving Average

Let $m_t = \frac{w_t + w_{t-1} + w_{t-2}}{3}, w_t \sim ind.N(0,1), t = 1,...,n$

Previously, we found that $\mu_t = 0, \forall t$ and

$$\begin{cases} \gamma(s,t) = \frac{3}{9} & \text{if } s = t \\ \gamma(s,t) = \frac{2}{9} & \text{if } |s-t| = 1 \\ \gamma(s,t) = \frac{1}{9} & \text{if } |s-t| = 2 \\ \gamma(s,t) = 0 & \text{if } |s-t| \geq 3 \end{cases}$$

Is the time series weakly stationary? Hint: let h=s-t.

Yes! You can see that $\gamma(s,t)$ only depends on h.

Comments:

- $\gamma(h) = \gamma(-h)$ for all h if the series is weakly stationary. This means that it does not matter which way the shift occurs.
- Stationarity can also be examined when two time series are of interest. We will examine this in more detail later in the course.

In summary,

- Both time series must have constant mean
- Both autocovariance functions must depend only on the lag difference
- The "cross-covariance" function, the extension of the autocovariance function for one time series to two time series, must depend only on the lag difference. The cross-covariance function is defined as

$$\gamma_{xy}(h) = E[(x_{t+h} - \mu_x)(y_t - \mu_y)]$$

Note that $\gamma_{xy}(h)$ is not necessarily equal to $\gamma_{yx}(h)$ (usually will be different).

Example 6.3. Let $x_t = w_t + w_{t-1}$ and $y_t = w_t - w_{t-1}$, where $w_t \sim \text{ind}N(0, \sigma_w^2)$ for t = 1, …, n

Claim: x_t and y_t are weakly stationary.

Proof.

- X_t

$E(X_t) = E(w_t + w_{t-1}) = E(w_t + w_{t-1}) = 0 + 0 = 0$ Thus $E(X_t) = \mu_{X_t} = 0 \forall t$

$\gamma(s,t) = E[(X_s - \mu_{X_s})(X_t - \mu_{X_t})]$
$= E[X_s X_t] = E[(w_s + w_{s+1})(w_t + w_{t+1})]$
$= E[w_s w_t + w_{s-1} w_t + w_s w_{t-1} + w_{s-1} w_{t-1}]$

if s=t, then $\gamma(t,t) = E[w_t^2 + w_{t-1} w_t + w_t w_{t-1} + w_{t-1}^2]$
$= Var(w_t) + (E[w_t])^2 + 2E[w_{t-1}]E[w_t] + Var(w_{t-1}) + (E[w_{t-1}])^2$
$= \sigma_w^2 + 0 + 0 + \sigma_w^2 + 0 = 2\sigma_w^2$

if s=t-1, then $\gamma(t-1,t) = E[w_{t-1} w_t + w_{t-2} w_t + w_{t-1}^2 + w_{t-2} w_{t-1}]$
$= E[w_{t-1}]E[w_t] + E[w_{t-2}]E[w_t] + Var(w_{t-1}) + (E[w_{t-1}])^2 + E[w_{t-2}]E[w_{t-1}]$
$= 0 + 0 + \sigma_w^2 + 0 + 0 = \sigma_w^2$

Note that $\gamma(t-1,t) = \sigma_w^2$ and $\gamma(s,t) = 0$ for |s - t|>1.

$$\begin{cases} \gamma(s,t) = 2\sigma_w^2 & \text{if } s = t \\ \gamma(s,t) = \sigma_w^2 & \text{if } |s-t| = 1 \\ \gamma(s,t) = 0 & \text{if } |s-t| > 1 \end{cases}$$

X_t is weakly stationary.

- Y_t

$E(Y_t) = E(w_t - w_{t-1}) = E(w_t - w_{t-1}) = 0 - 0 = 0$ Thus $E(Y_t) = \mu_{Y_t} = 0 \forall t$

$\gamma(s,t) = E[(Y_s - \mu_{Y_s})(Y_t - \mu_{Y_t})] = E[Y_s Y_t]$
$= E[(w_s - w_{s+1})(w_t - w_{t+1})] =$
$E[w_s w_t - w_{s-1} w_t - w_s w_{t-1} + w_{s-1} w_{t-1}]$

if s=t, then $\gamma(t,t) = E[w_t^2 - w_{t-1} w_t - w_t w_{t-1} + w_{t-1}^2]$
$= Var(w_t) + (E[w_t])^2 - 2E[w_{t-1}]E[w_t] + Var(w_{t-1}) + (E[w_{t-1}])^2$
$= \sigma_w^2 + 0 - 0 + \sigma_w^2 + 0 = 2\sigma_w^2$

if s=t-1, then $\gamma(t-1,t) = E[w_{t-1} w_t - w_{t-2} w_t - w_{t-1}^2 + w_{t-2} w_{t-1}]$
$= E[w_{t-1}]E[w_t] - E[w_{t-2}]E[w_t] - Var(w_{t-1}) - (E[w_{t-1}])^2 + E[w_{t-2}]E[w_{t-1}]$
$= 0 - 0 - \sigma_w^2 - 0 + 0 = -\sigma_w^2$

Note that $\gamma(t-1,t) = -\sigma_w^2$ and $\gamma(s,t) = 0$ for |s - t|>1.

$$\begin{cases} \gamma(s,t) = 2\sigma_w^2 & \text{if } s = t \\ \gamma(s,t) = -\sigma_w^2 & \text{if } |s-t| = 1 \\ \gamma(s,t) = 0 & \text{if } |s-t| > 1 \end{cases}$$

Y_t is weakly stationary.

□

6.2 Linear Process

The previous examples are special cases of a "linear process". In general, a linear process can be defined as

$$X_t = \mu + \sum_{j=-\infty}^{\infty} \psi_j w_{t-j}, \sum_{j=-\infty}^{\infty} |\psi_j| < \infty$$

and

$$w_t \sim \text{ind.} N(0, \sigma_w^2)$$

It can be shown that $\gamma(h) = \sigma_w^2 \sum_{j=-\infty}^{\infty} \psi_{j+h} \psi_j$ for h ≥ 0 provided the series is stationary (remember that and $\gamma(h) = \gamma(-h)$).

Proof. WLOG, let $\mu = 0$, since constants do not affect a covariance.

$E(X_t) = E(\mu + \sum_{j=-\infty}^{\infty} \psi_j w_{t-j})$
$= 0 + \sum_{j=-\infty}^{\infty} \psi_j E(w_{t-j})$
$= 0 + \sum_{j=-\infty}^{\infty} \psi_j 0 = 0$

Note that $\gamma(h) = Cov(x_t, x_{t+h}) = E(x_t x_{t+h})$–$E(x_t)E(x_{t+h}) = E(x_t x_{t+h})$ since $E(x_t) = E(x_{t+h}) = 0$

Then $E(x_t x_{t+h}) = E[(\sum_{i=-\infty}^{\infty} \psi_i w_{t-i})(\sum_{i=-\infty}^{\infty} \psi_j w_{t+h-j})]$
$= E[\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \psi_i \psi_j w_{t-i} w_{t+h-j}]$
$= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \psi_i \psi_j E(w_{t-i} w_{t+h-j})$
$= \sigma_w^2 \sum_{k=-\infty}^{\infty} \psi_k \psi_{k+h}$

Note that $E(w_{t-i} w_{t+h-j}) = 0, \forall -i \neq h-j$ and $E(w_{t-i} w_{t+h-j}) = E(w_{t-i}^2) = \sigma_w^2, \forall -i = h-j (i.e., j-i = h)$ and $\psi' s$ are constants.

Hence, $\gamma(h) = \sigma_w^2 \sum_{j=-\infty}^{\infty} \psi_{j+h} \psi_j$ for h ≥ 0. □

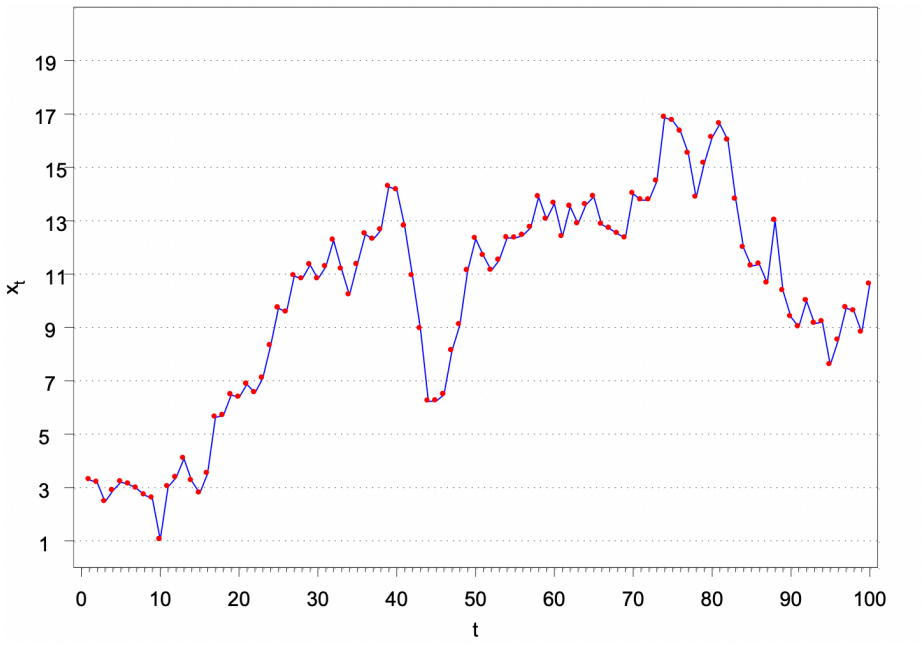
Important:

There is a very important case when weakly stationary implies strictly stationary. This occurs when the time series has a multivariate normal distribution. Remember that a univariate normal distribution is defined only by its mean and variance. The multivariate normal distribution is defined only by its mean vector and covariance matrix.

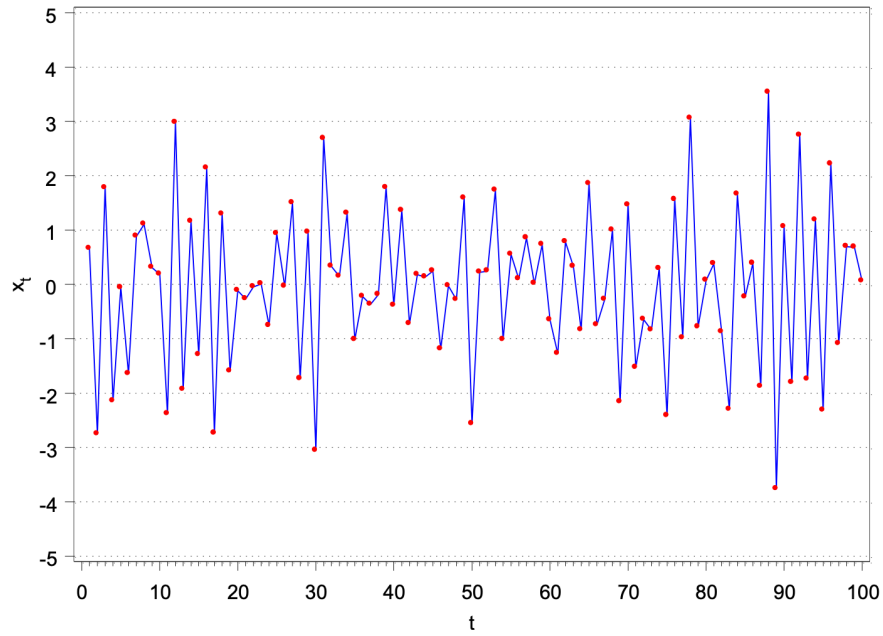
Thus, if we can assume a multivariate normal distribution, we ONLY need to check if the time series satisfies the weakly stationary requirements to say the time series is strongly stationary. Thus, notice what the word "stationary" would mean in this case.

Example 6.4. Visualizing Stationarity

Below are a few plots of the observed values of a time series. Identify which plots correspond to a weakly stationary series.

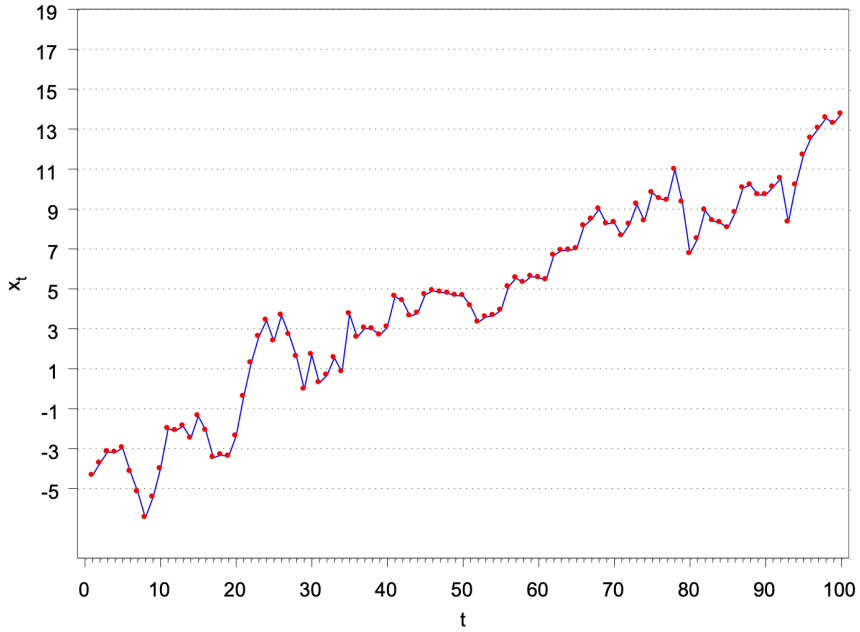


This graph violates weakly stationary. It seems that there's a structural change. You can find there are two distinct mean: high mean and low mean.

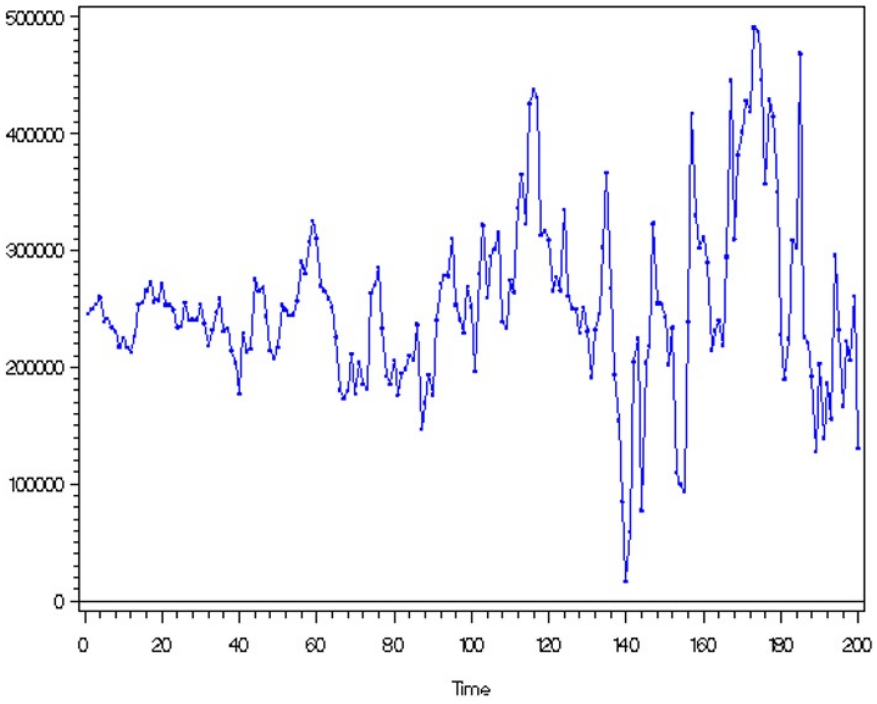


This graph doesn't seem to violate weakly stationary, as you can see the mean seems to be a constant 0 and the variablity seems to be a constant across

different time points.



Clearly, this violates weakly stationary. You can see the mean keeps changing.



Clearly, this violates weakly stationary. You can see the variance keeps changing.

It is important to have stationarity b/c it allows us to estimate the mean and variance and we need consistency to construct a model.

Chapter 7

Estimation and Inference for measures of dependence

$\mu, \gamma(h), \rho(h)$ are usually unknown so we need to estimate them. To estimate these quantities, we need to assume the time series is weakly stationary.

7.1 Sample mean function

By the weakly stationary assumption, $E(x_1) = \mu, E(x_2) = \mu, ..., E(x_n) = \mu$. Thus, a logical estimate of μ is

$$\bar{X} = \frac{1}{n}\sum_{t=1}^{n} X_t$$

Note that this would not make sense to do if the weakly stationarity assumption did not hold!

7.2 Sample autocovariance function

Again with the weakly stationarity assumption, we only need to worry about the lag difference. The estimated autocovariance function is:

$$\hat{\gamma}(h) = \frac{1}{n}\sum_{t=1}^{n-h}(X_{t+h} - \bar{X})(X_t - \bar{X})$$

- $\hat{\gamma}(h) = \hat{\gamma}(-h)$

- What is this quantity if h = 0?
  - $\hat{\gamma}(h) = \frac{1}{n}\sum_{t=1}^{n}(X_t - \bar{X})(X_t - \bar{X})$, which is essentially the sample variance.(when n is large n≈n-1)
- What is this quantity if h =1?
  - $\hat{\gamma}(h) = \frac{1}{n}\sum_{t=1}^{n-1}(X_{t+1} - \bar{X})(X_t - \bar{X})$
- This is similar to the formula often used to estimate the covariance between two random variables x and y: $\hat{Cov}(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})$.
- The sum goes up to n - h to avoid having negative subscripts in the x's.
- This is NOT an unbiased estimate of $\gamma(h)$! However, as n gets larger, the bias will go to 0.

7.3 Sample autocorrelation function (ACF)

$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}$$

Question: What does $\rho(h) = 0$ mean and why would this be important to detect?

That means there's no linear relationship between X_{t-h} and X_t for this particular lag h. This is important b/c this makes it no sense to use X_{t-h} to predict X_t.

Because this is important, we conduct hypothesis tests for $\rho(h)$ for all h $\neq$ 0! To do the hypothesis test, we need to find the sampling distribution for $\hat{\rho}(h)$ under the null hypothesis of $\rho(h) = 0$.

7.4 Sampling distribution

In summary, if $\rho(h) = 0$, x_t is stationary, and the sample size is "large", then $\hat{\rho}(h)$ has an approximate normal distribution with mean 0 and standard deviation $\sigma_{\hat{\rho}(h)} = \frac{1}{\sqrt{n}}$, i.e., $\hat{\rho}(h) \sim N(0, \frac{1}{\sqrt{n}})$

A proof is available in Shumway and Stoffer's textbook and requires an understanding asymptotics (PhD level statistics course). $(\sqrt{n}(\hat{\rho}(h) - 0) \rightarrow^d N(0,1))$

$H_0 : \rho(h) = 0$

$Z = \frac{\hat{\rho}(h) - 0}{\frac{1}{\sqrt{n}}}$

$Z > |Z_{1-\frac{\alpha}{2}}|$ reject H_0

$\hat{\rho}(h) > \pm\frac{Z_{1-\frac{\alpha}{2}}}{\sqrt{n}}$ reject H_0

For a hypothesis test, we could check if $\hat{\rho}(h)$ is within the bounds of $0 \pm \frac{Z_{1-\frac{\alpha}{2}}}{\sqrt{n}}$ or not where $P(Z < Z_{1-\frac{\alpha}{2}}) = 1 - \frac{\alpha}{2}$ for a standard normal random variable Z. If it is not, then there is sufficient evidence to conclude that $\rho(h) \neq 0$. We will be using this result a lot for the rest of this course!

Example 7.1. $x_t = 0.7x_{t-1} + w_t, w_t \sim \text{ind.}N(0,1), n = 100$

Click here to download data.

```
ar1 <- read.table(file = "AR1.0.7.txt", header = TRUE, sep = "")

head(ar1)
```

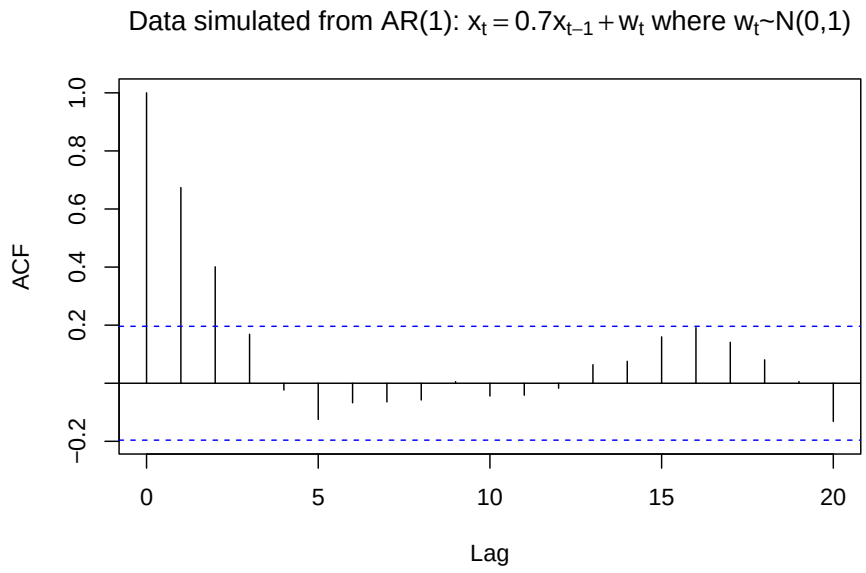
```
##   t          x
## 1 1  0.0417268
## 2 2  0.3719068
## 3 3 -0.1854518
## 4 4 -1.3829742
## 5 5 -2.8759365
## 6 6 -2.6001761
```

```
x <- ar1$x
```

```
dev.new(width = 8, height = 6, pointsize = 10)
#Opens up wider plot window than the default (good for time series plots)
plot(x = x, ylab = expression(x[t]), xlab = "t", type = "l", col = "red", lwd = 1 ,
     main = expression(paste("Data simulated from AR(1): ", x[t] == 0.7*x[t-1] + w[t], " where ",
      panel.first=grid(col = "gray", lty = "dotted"))
points(x = x, pch = 20, col = "blue")
```

The easiest way to find the autocorrelations in R is to use the `acf()` function.

```
rho.x <- acf(x = x, type = "correlation", main =
             expression(paste("Data simulated from AR(1): ", x[t] == 0.7*x[t-1] + w[t], " where "
```




```
  # ci argument can be used to change 1 - alpha for plot
  # lag.max argument can be used to change the maximum number of lags
```

In our language, the horizontal axis: lag=h, the verical axis: ACF=$\hat{\rho}(h)$

The horizontal lines on the plot are drawn at $0 \pm \frac{Z_{1-\frac{0.05}{2}}}{\sqrt{n}}$ where $Z_{1-\frac{0.05}{2}} = 1.96$. i.e., outside the blue dashed line, we reject H_0

The location of the lines can be changed by using the `ci` (confidence interval)argument. The default is ci $= 0.95$. i.e., $\alpha = 0.05$

```
rho.x
```

```
##
## Autocorrelations of series 'x', by lag
##
##      0      1      2      3      4      5      6      7      8      9     10
##  1.000  0.674  0.401  0.169 -0.023 -0.125 -0.067 -0.064 -0.058  0.005 -0.044
##     11     12     13     14     15     16     17     18     19     20
## -0.041 -0.017  0.064  0.076  0.160  0.191  0.141  0.081  0.006 -0.132
```

```
# the first one is rho_hat(0)
# the second one is rho_hat(1)
```

```
names(rho.x)
```

```
## [1] "acf"    "type"   "n.used" "lag"    "series" "snames"
```

```
rho.x$acf
```

```
## , , 1
##
##                [,1]
##  [1,]  1.000000000
##  [2,]  0.673671871
##  [3,]  0.400891188
##  [4,]  0.168552826
##  [5,] -0.023391129
##  [6,] -0.124632501
##  [7,] -0.067392830
##  [8,] -0.064248086
##  [9,] -0.057717749
## [10,]  0.005312358
## [11,] -0.044035976
## [12,] -0.041121407
## [13,] -0.017197132
## [14,]  0.063864970
## [15,]  0.075575696
## [16,]  0.159665692
## [17,]  0.191349965
## [18,]  0.140967540
## [19,]  0.080508273
## [20,]  0.005584061
## [21,] -0.131559629
```

```
# the first one is rho_hat(0)
# the second one is rho_hat(1)
```

```
rho.x$acf[1:2]
```

```
## [1] 1.0000000 0.6736719
```

Questions:

- What happens to the autocorrelations over time? Why do you think this happens?

  - From the model $x_t = 0.7x_{t-1} + w_t$, you can see that the auto correlation dies out as the lag term h increases, the main reason is the coefficient 0.7

- Is there a positive or negative correlation?

  - A positive correlation, again from our model $x_t = 0.7x_{t-1}+w_t$, 0.7>0

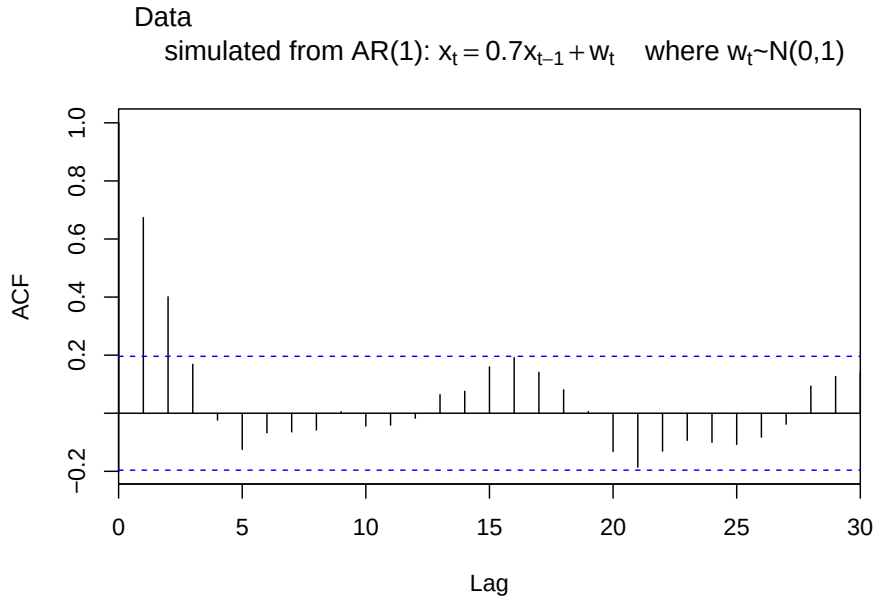
- At what lags is $ (h) $0?

  - h=0,1,2. But we don't care h=0, it's 1 just by definition.

R plots $\hat{\rho}(0) = 1$ by default. This is unnecessary because $\hat{\rho}(0)$ will be 1 for all time series data sets (again, it's just by definition)! To remove $\hat{\rho}(0)$ from the plot, one can specify the x-axis limit to start at 1. Below is one way this can be done and also illustrates how to use the `lag.max` argument.

```
par(xaxs = "i")
# Remove default 4% extra space around  min and max of x-axis

rho.x2 <- acf(x = x, type = "correlation", xlim =
    c(0,30), lag.max = 30, main = expression(paste("Data
    simulated from AR(1): ", x[t] == 0.7*x[t-1] + w[t], "
    where ", w[t], "~N(0,1)")))
```

```
## Warning in title(main %||% if (i == j) snames[i] else paste(sn.abbr[i], : font
## metrics unknown for character 0xa

## Warning in title(main %||% if (i == j) snames[i] else paste(sn.abbr[i], : font
## metrics unknown for character 0xa

## Warning in title(main %||% if (i == j) snames[i] else paste(sn.abbr[i], : font
## metrics unknown for character 0xa

## Warning in title(main %||% if (i == j) snames[i] else paste(sn.abbr[i], : font
## metrics unknown for character 0xa
```




```
par(xaxs = "r") # Return to the default: regular pattern
```

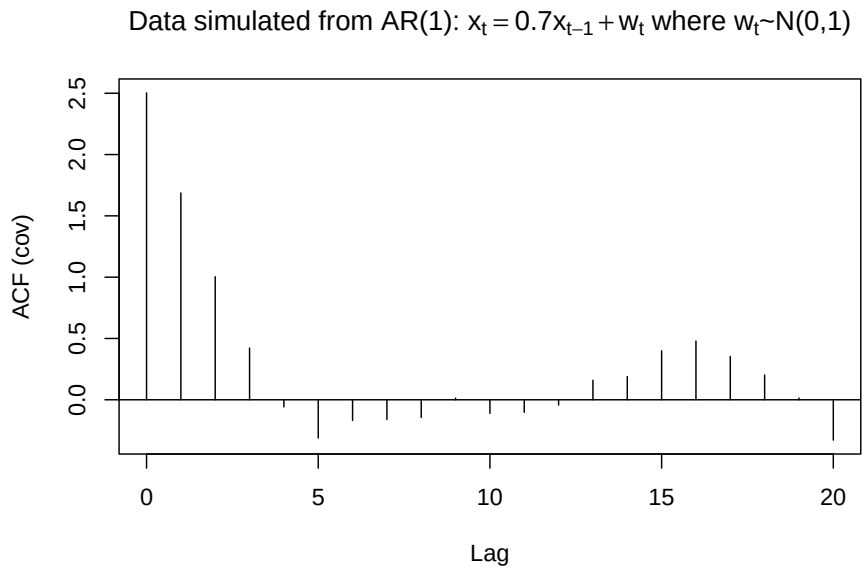
Note that $\hat{\rho}(0) = 1$ is still present but the y-axis at x = 0 hides it.

While displaying $\hat{\rho}(0) = 1$ may seem minor, we will examine these autocorrelations later in the course to determine an appropriate model for a data set. Often, one will forget to ignore the line drawn at lag = 0 and choose an incorrect model.

You should always ignore the line drawn at lag=0!!! b/c it's 1 just by definition.

The autocovariances can also be found using `acf()`.

```
acf(x = x, type = "covariance", main =
     expression(paste("Data simulated from AR(1): ", x[t]
     == 0.7*x[t-1] + w[t], " where ", w[t], "~N(0,1)")))
```



To help understand autocorrelations and their relationship with the correlation coefficient better, I decided to look at the "usual" estimated Pearson correlation coefficients between x_t, x_{t-1}, x_{t-2}, and x_{t-3}.

```
# Examine usual ways to check correlation
x.ts <- ts(x)
x.ts
```

```
## Time Series:
## Start = 1
## End = 100
## Frequency = 1
##   [1]  0.04172680  0.37190682 -0.18545185 -1.38297422 -2.87593652 -2.60017605
##   [7] -1.10401719 -0.46385116  0.80339069  2.11483585  3.39124978  3.86739194
##  [13]  2.12733595  0.67590604  0.71429367  0.38928044 -1.22681923  0.02287443
##  [19] -0.57321924 -2.68376851 -4.80850095 -2.44797633 -1.73921817 -1.48773023
##  [25]  0.68920966  0.40220308 -1.58781041 -2.07300848 -2.63408197 -3.54333559
##  [31] -2.74116328 -2.54401750 -3.19570817 -0.02623669  0.41663974 -1.74485791
##  [37] -3.04847994 -1.64692948 -1.71478199  0.11340965  1.55017869  0.47317192
##  [43] -0.18521791 -0.05759091 -1.32105323 -1.22071881 -1.53827085  0.01277076
##  [49] -3.19955388 -3.11022833 -3.30621969 -2.00546537  0.18084415  0.58776479
##  [55] -0.70238414 -0.34570939  0.94209248 -0.78176791  1.30547320 -1.04054783
##  [61] -1.06897160 -1.08850000  0.06031172 -0.05724856 -1.14731083 -0.79262221
##  [67] -0.55565451 -1.55985750 -2.17062644 -1.07776017  0.51569067  2.30660050
##  [73]  1.53530426  2.55899301  1.83836277  1.08072014  1.34125182 -0.80729300
```

```
##  [79] -1.42735924 -0.42456207 -0.11625003 -0.74807460  0.70052717  0.08557377
##  [85] -0.06039041  0.04479407 -0.12657328 -1.30097021  0.81586192 -0.13139757
##  [91]  1.84725644  1.62364752  0.33080663 -0.40824385 -1.56008530 -1.63175408
##  [97] -1.36418639 -0.37209392 -0.65833401  2.03705932
```

```
set1 <- ts.intersect(x.ts, x.ts1 = lag(x = x.ts, k = -1), x.ts2 = lag(x = x.ts, k = -2), x.ts3 =

# b/c we use ts.intersect (take intersection), we have the following
# x.ts starts at X4
# x.ts1 starts at X3
# x.ts2 starts at X2
# x.ts3 starts at X1

set1
```

```
## Time Series:
## Start = 4
## End = 100
## Frequency = 1
##            x.ts       x.ts1       x.ts2       x.ts3
##   4 -1.38297422 -0.18545185  0.37190682  0.04172680
##   5 -2.87593652 -1.38297422 -0.18545185  0.37190682
##   6 -2.60017605 -2.87593652 -1.38297422 -0.18545185
##   7 -1.10401719 -2.60017605 -2.87593652 -1.38297422
##   8 -0.46385116 -1.10401719 -2.60017605 -2.87593652
##   9  0.80339069 -0.46385116 -1.10401719 -2.60017605
##  10  2.11483585  0.80339069 -0.46385116 -1.10401719
##  11  3.39124978  2.11483585  0.80339069 -0.46385116
##  12  3.86739194  3.39124978  2.11483585  0.80339069
##  13  2.12733595  3.86739194  3.39124978  2.11483585
##  14  0.67590604  2.12733595  3.86739194  3.39124978
##  15  0.71429367  0.67590604  2.12733595  3.86739194
##  16  0.38928044  0.71429367  0.67590604  2.12733595
##  17 -1.22681923  0.38928044  0.71429367  0.67590604
##  18  0.02287443 -1.22681923  0.38928044  0.71429367
##  19 -0.57321924  0.02287443 -1.22681923  0.38928044
##  20 -2.68376851 -0.57321924  0.02287443 -1.22681923
##  21 -4.80850095 -2.68376851 -0.57321924  0.02287443
##  22 -2.44797633 -4.80850095 -2.68376851 -0.57321924
##  23 -1.73921817 -2.44797633 -4.80850095 -2.68376851
##  24 -1.48773023 -1.73921817 -2.44797633 -4.80850095
##  25  0.68920966 -1.48773023 -1.73921817 -2.44797633
##  26  0.40220308  0.68920966 -1.48773023 -1.73921817
##  27 -1.58781041  0.40220308  0.68920966 -1.48773023
##  28 -2.07300848 -1.58781041  0.40220308  0.68920966
```

```
##  29 -2.63408197 -2.07300848 -1.58781041  0.40220308
##  30 -3.54333559 -2.63408197 -2.07300848 -1.58781041
##  31 -2.74116328 -3.54333559 -2.63408197 -2.07300848
##  32 -2.54401750 -2.74116328 -3.54333559 -2.63408197
##  33 -3.19570817 -2.54401750 -2.74116328 -3.54333559
##  34 -0.02623669 -3.19570817 -2.54401750 -2.74116328
##  35  0.41663974 -0.02623669 -3.19570817 -2.54401750
##  36 -1.74485791  0.41663974 -0.02623669 -3.19570817
##  37 -3.04847994 -1.74485791  0.41663974 -0.02623669
##  38 -1.64692948 -3.04847994 -1.74485791  0.41663974
##  39 -1.71478199 -1.64692948 -3.04847994 -1.74485791
##  40  0.11340965 -1.71478199 -1.64692948 -3.04847994
##  41  1.55017869  0.11340965 -1.71478199 -1.64692948
##  42  0.47317192  1.55017869  0.11340965 -1.71478199
##  43 -0.18521791  0.47317192  1.55017869  0.11340965
##  44 -0.05759091 -0.18521791  0.47317192  1.55017869
##  45 -1.32105323 -0.05759091 -0.18521791  0.47317192
##  46 -1.22071881 -1.32105323 -0.05759091 -0.18521791
##  47 -1.53827085 -1.22071881 -1.32105323 -0.05759091
##  48  0.01277076 -1.53827085 -1.22071881 -1.32105323
##  49 -3.19955388  0.01277076 -1.53827085 -1.22071881
##  50 -3.11022833 -3.19955388  0.01277076 -1.53827085
##  51 -3.30621969 -3.11022833 -3.19955388  0.01277076
##  52 -2.00546537 -3.30621969 -3.11022833 -3.19955388
##  53  0.18084415 -2.00546537 -3.30621969 -3.11022833
##  54  0.58776479  0.18084415 -2.00546537 -3.30621969
##  55 -0.70238414  0.58776479  0.18084415 -2.00546537
##  56 -0.34570939 -0.70238414  0.58776479  0.18084415
##  57  0.94209248 -0.34570939 -0.70238414  0.58776479
##  58 -0.78176791  0.94209248 -0.34570939 -0.70238414
##  59  1.30547320 -0.78176791  0.94209248 -0.34570939
##  60 -1.04054783  1.30547320 -0.78176791  0.94209248
##  61 -1.06897160 -1.04054783  1.30547320 -0.78176791
##  62 -1.08850000 -1.06897160 -1.04054783  1.30547320
##  63  0.06031172 -1.08850000 -1.06897160 -1.04054783
##  64 -0.05724856  0.06031172 -1.08850000 -1.06897160
##  65 -1.14731083 -0.05724856  0.06031172 -1.08850000
##  66 -0.79262221 -1.14731083 -0.05724856  0.06031172
##  67 -0.55565451 -0.79262221 -1.14731083 -0.05724856
##  68 -1.55985750 -0.55565451 -0.79262221 -1.14731083
##  69 -2.17062644 -1.55985750 -0.55565451 -0.79262221
##  70 -1.07776017 -2.17062644 -1.55985750 -0.55565451
##  71  0.51569067 -1.07776017 -2.17062644 -1.55985750
##  72  2.30660050  0.51569067 -1.07776017 -2.17062644
##  73  1.53530426  2.30660050  0.51569067 -1.07776017
##  74  2.55899301  1.53530426  2.30660050  0.51569067
```

```
##  75  1.83836277  2.55899301  1.53530426  2.30660050
##  76  1.08072014  1.83836277  2.55899301  1.53530426
##  77  1.34125182  1.08072014  1.83836277  2.55899301
##  78 -0.80729300  1.34125182  1.08072014  1.83836277
##  79 -1.42735924 -0.80729300  1.34125182  1.08072014
##  80 -0.42456207 -1.42735924 -0.80729300  1.34125182
##  81 -0.11625003 -0.42456207 -1.42735924 -0.80729300
##  82 -0.74807460 -0.11625003 -0.42456207 -1.42735924
##  83  0.70052717 -0.74807460 -0.11625003 -0.42456207
##  84  0.08557377  0.70052717 -0.74807460 -0.11625003
##  85 -0.06039041  0.08557377  0.70052717 -0.74807460
##  86  0.04479407 -0.06039041  0.08557377  0.70052717
##  87 -0.12657328  0.04479407 -0.06039041  0.08557377
##  88 -1.30097021 -0.12657328  0.04479407 -0.06039041
##  89  0.81586192 -1.30097021 -0.12657328  0.04479407
##  90 -0.13139757  0.81586192 -1.30097021 -0.12657328
##  91  1.84725644 -0.13139757  0.81586192 -1.30097021
##  92  1.62364752  1.84725644 -0.13139757  0.81586192
##  93  0.33080663  1.62364752  1.84725644 -0.13139757
##  94 -0.40824385  0.33080663  1.62364752  1.84725644
##  95 -1.56008530 -0.40824385  0.33080663  1.62364752
##  96 -1.63175408 -1.56008530 -0.40824385  0.33080663
##  97 -1.36418639 -1.63175408 -1.56008530 -0.40824385
##  98 -0.37209392 -1.36418639 -1.63175408 -1.56008530
##  99 -0.65833401 -0.37209392 -1.36418639 -1.63175408
## 100  2.03705932 -0.65833401 -0.37209392 -1.36418639
```

```
cor(set1)
```

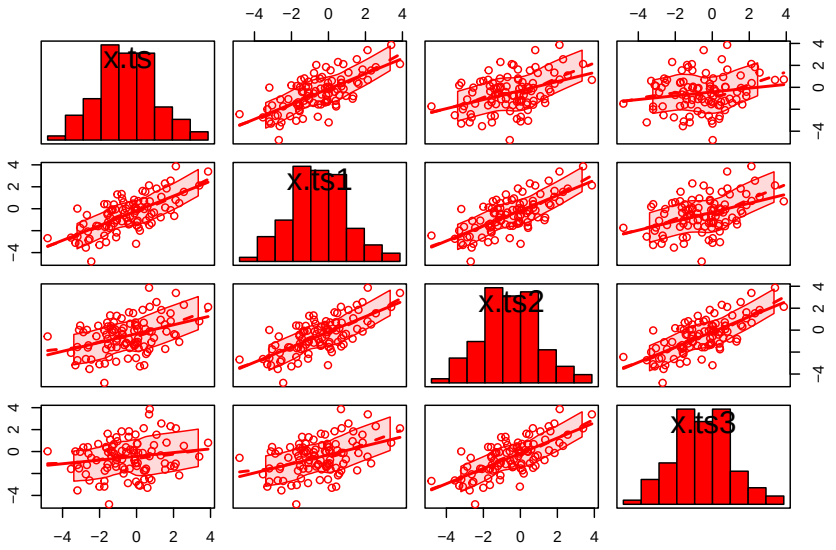
```
##            x.ts     x.ts1     x.ts2     x.ts3
## x.ts  1.0000000 0.6824913 0.4065326 0.1710145
## x.ts1 0.6824913 1.0000000 0.6929638 0.4108375
## x.ts2 0.4065326 0.6929638 1.0000000 0.6935801
## x.ts3 0.1710145 0.4108375 0.6935801 1.0000000
```

```
# corr matrix
```

```
library(car) #scatterplot.matrix is in this package
```

```
## Loading required package: carData
```

```
scatterplotMatrix(formula = ~x.ts + x.ts1 + x.ts2 + x.ts3, data = set1,
    diagonal = list(method = "histogram"), col = "red")
```




```
set2 <- ts.intersect(x.ts, x.ts1 = lag(x = x.ts, k = 1), x.ts2 = lag(x = x.ts, k = 2),

set2
```

```
## Time Series:
## Start = 1
## End = 97
## Frequency = 1
##            x.ts       x.ts1       x.ts2       x.ts3
##  1  0.04172680  0.37190682 -0.18545185 -1.38297422
##  2  0.37190682 -0.18545185 -1.38297422 -2.87593652
##  3 -0.18545185 -1.38297422 -2.87593652 -2.60017605
##  4 -1.38297422 -2.87593652 -2.60017605 -1.10401719
##  5 -2.87593652 -2.60017605 -1.10401719 -0.46385116
##  6 -2.60017605 -1.10401719 -0.46385116  0.80339069
##  7 -1.10401719 -0.46385116  0.80339069  2.11483585
##  8 -0.46385116  0.80339069  2.11483585  3.39124978
##  9  0.80339069  2.11483585  3.39124978  3.86739194
## 10  2.11483585  3.39124978  3.86739194  2.12733595
## 11  3.39124978  3.86739194  2.12733595  0.67590604
## 12  3.86739194  2.12733595  0.67590604  0.71429367
## 13  2.12733595  0.67590604  0.71429367  0.38928044
## 14  0.67590604  0.71429367  0.38928044 -1.22681923
## 15  0.71429367  0.38928044 -1.22681923  0.02287443
```

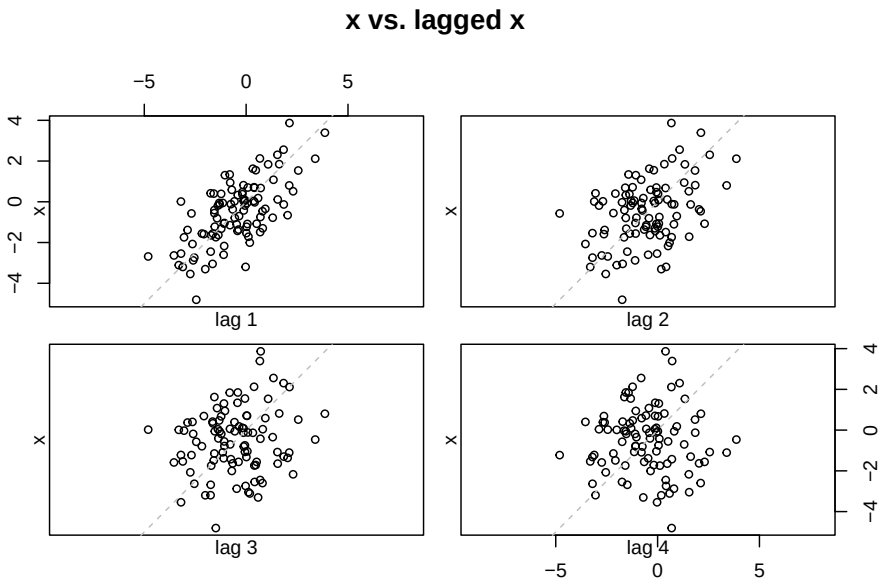
```
## 16  0.38928044 -1.22681923  0.02287443 -0.57321924
## 17 -1.22681923  0.02287443 -0.57321924 -2.68376851
## 18  0.02287443 -0.57321924 -2.68376851 -4.80850095
## 19 -0.57321924 -2.68376851 -4.80850095 -2.44797633
## 20 -2.68376851 -4.80850095 -2.44797633 -1.73921817
## 21 -4.80850095 -2.44797633 -1.73921817 -1.48773023
## 22 -2.44797633 -1.73921817 -1.48773023  0.68920966
## 23 -1.73921817 -1.48773023  0.68920966  0.40220308
## 24 -1.48773023  0.68920966  0.40220308 -1.58781041
## 25  0.68920966  0.40220308 -1.58781041 -2.07300848
## 26  0.40220308 -1.58781041 -2.07300848 -2.63408197
## 27 -1.58781041 -2.07300848 -2.63408197 -3.54333559
## 28 -2.07300848 -2.63408197 -3.54333559 -2.74116328
## 29 -2.63408197 -3.54333559 -2.74116328 -2.54401750
## 30 -3.54333559 -2.74116328 -2.54401750 -3.19570817
## 31 -2.74116328 -2.54401750 -3.19570817 -0.02623669
## 32 -2.54401750 -3.19570817 -0.02623669  0.41663974
## 33 -3.19570817 -0.02623669  0.41663974 -1.74485791
## 34 -0.02623669  0.41663974 -1.74485791 -3.04847994
## 35  0.41663974 -1.74485791 -3.04847994 -1.64692948
## 36 -1.74485791 -3.04847994 -1.64692948 -1.71478199
## 37 -3.04847994 -1.64692948 -1.71478199  0.11340965
## 38 -1.64692948 -1.71478199  0.11340965  1.55017869
## 39 -1.71478199  0.11340965  1.55017869  0.47317192
## 40  0.11340965  1.55017869  0.47317192 -0.18521791
## 41  1.55017869  0.47317192 -0.18521791 -0.05759091
## 42  0.47317192 -0.18521791 -0.05759091 -1.32105323
## 43 -0.18521791 -0.05759091 -1.32105323 -1.22071881
## 44 -0.05759091 -1.32105323 -1.22071881 -1.53827085
## 45 -1.32105323 -1.22071881 -1.53827085  0.01277076
## 46 -1.22071881 -1.53827085  0.01277076 -3.19955388
## 47 -1.53827085  0.01277076 -3.19955388 -3.11022833
## 48  0.01277076 -3.19955388 -3.11022833 -3.30621969
## 49 -3.19955388 -3.11022833 -3.30621969 -2.00546537
## 50 -3.11022833 -3.30621969 -2.00546537  0.18084415
## 51 -3.30621969 -2.00546537  0.18084415  0.58776479
## 52 -2.00546537  0.18084415  0.58776479 -0.70238414
## 53  0.18084415  0.58776479 -0.70238414 -0.34570939
## 54  0.58776479 -0.70238414 -0.34570939  0.94209248
## 55 -0.70238414 -0.34570939  0.94209248 -0.78176791
## 56 -0.34570939  0.94209248 -0.78176791  1.30547320
## 57  0.94209248 -0.78176791  1.30547320 -1.04054783
## 58 -0.78176791  1.30547320 -1.04054783 -1.06897160
## 59  1.30547320 -1.04054783 -1.06897160 -1.08850000
## 60 -1.04054783 -1.06897160 -1.08850000  0.06031172
## 61 -1.06897160 -1.08850000  0.06031172 -0.05724856
```

```
## 62 -1.08850000  0.06031172 -0.05724856 -1.14731083
## 63  0.06031172 -0.05724856 -1.14731083 -0.79262221
## 64 -0.05724856 -1.14731083 -0.79262221 -0.55565451
## 65 -1.14731083 -0.79262221 -0.55565451 -1.55985750
## 66 -0.79262221 -0.55565451 -1.55985750 -2.17062644
## 67 -0.55565451 -1.55985750 -2.17062644 -1.07776017
## 68 -1.55985750 -2.17062644 -1.07776017  0.51569067
## 69 -2.17062644 -1.07776017  0.51569067  2.30660050
## 70 -1.07776017  0.51569067  2.30660050  1.53530426
## 71  0.51569067  2.30660050  1.53530426  2.55899301
## 72  2.30660050  1.53530426  2.55899301  1.83836277
## 73  1.53530426  2.55899301  1.83836277  1.08072014
## 74  2.55899301  1.83836277  1.08072014  1.34125182
## 75  1.83836277  1.08072014  1.34125182 -0.80729300
## 76  1.08072014  1.34125182 -0.80729300 -1.42735924
## 77  1.34125182 -0.80729300 -1.42735924 -0.42456207
## 78 -0.80729300 -1.42735924 -0.42456207 -0.11625003
## 79 -1.42735924 -0.42456207 -0.11625003 -0.74807460
## 80 -0.42456207 -0.11625003 -0.74807460  0.70052717
## 81 -0.11625003 -0.74807460  0.70052717  0.08557377
## 82 -0.74807460  0.70052717  0.08557377 -0.06039041
## 83  0.70052717  0.08557377 -0.06039041  0.04479407
## 84  0.08557377 -0.06039041  0.04479407 -0.12657328
## 85 -0.06039041  0.04479407 -0.12657328 -1.30097021
## 86  0.04479407 -0.12657328 -1.30097021  0.81586192
## 87 -0.12657328 -1.30097021  0.81586192 -0.13139757
## 88 -1.30097021  0.81586192 -0.13139757  1.84725644
## 89  0.81586192 -0.13139757  1.84725644  1.62364752
## 90 -0.13139757  1.84725644  1.62364752  0.33080663
## 91  1.84725644  1.62364752  0.33080663 -0.40824385
## 92  1.62364752  0.33080663 -0.40824385 -1.56008530
## 93  0.33080663 -0.40824385 -1.56008530 -1.63175408
## 94 -0.40824385 -1.56008530 -1.63175408 -1.36418639
## 95 -1.56008530 -1.63175408 -1.36418639 -0.37209392
## 96 -1.63175408 -1.36418639 -0.37209392 -0.65833401
## 97 -1.36418639 -0.37209392 -0.65833401  2.03705932
```

```
cor(set2)
```

```
##            x.ts     x.ts1     x.ts2     x.ts3
## x.ts  1.0000000 0.6935801 0.4108375 0.1710145
## x.ts1 0.6935801 1.0000000 0.6929638 0.4065326
## x.ts2 0.4108375 0.6929638 1.0000000 0.6824913
## x.ts3 0.1710145 0.4065326 0.6824913 1.0000000
```

```
#Another way to see dependence
lag.plot(x = x, lags = 4, layout = c(2,2), main = "x vs. lagged x",
    do.lines = FALSE)
```



- The `ts()` function converts the time series data to an object that R recognizes as a time series.

- The `lag()` function is used to find xt-1, xt-2, and xt-3. The k argument specifies how many time periods to go back. Run `lag(x.ts, k = -1)` and `lag(x.ts, k = 1)`to see what happens. To get everything lined up as I wanted with `ts.intersect()`, I chose to use k = -1.

```
lag(x.ts, k = -1) #shift down one time period(forward)
```

```
## Time Series:
## Start = 2
## End = 101
## Frequency = 1
##   [1]  0.04172680  0.37190682 -0.18545185 -1.38297422 -2.87593652 -2.60017605
##   [7] -1.10401719 -0.46385116  0.80339069  2.11483585  3.39124978  3.86739194
##  [13]  2.12733595  0.67590604  0.71429367  0.38928044 -1.22681923  0.02287443
##  [19] -0.57321924 -2.68376851 -4.80850095 -2.44797633 -1.73921817 -1.48773023
##  [25]  0.68920966  0.40220308 -1.58781041 -2.07300848 -2.63408197 -3.54333559
```

```
##  [31] -2.74116328 -2.54401750 -3.19570817 -0.02623669  0.41663974 -1.74485791
##  [37] -3.04847994 -1.64692948 -1.71478199  0.11340965  1.55017869  0.47317192
##  [43] -0.18521791 -0.05759091 -1.32105323 -1.22071881 -1.53827085  0.01277076
##  [49] -3.19955388 -3.11022833 -3.30621969 -2.00546537  0.18084415  0.58776479
##  [55] -0.70238414 -0.34570939  0.94209248 -0.78176791  1.30547320 -1.04054783
##  [61] -1.06897160 -1.08850000  0.06031172 -0.05724856 -1.14731083 -0.79262221
##  [67] -0.55565451 -1.55985750 -2.17062644 -1.07776017  0.51569067  2.30660050
##  [73]  1.53530426  2.55899301  1.83836277  1.08072014  1.34125182 -0.80729300
##  [79] -1.42735924 -0.42456207 -0.11625003 -0.74807460  0.70052717  0.08557377
##  [85] -0.06039041  0.04479407 -0.12657328 -1.30097021  0.81586192 -0.13139757
##  [91]  1.84725644  1.62364752  0.33080663 -0.40824385 -1.56008530 -1.63175408
##  [97] -1.36418639 -0.37209392 -0.65833401  2.03705932
```

```
lag(x.ts, k = 0)
```

```
## Time Series:
## Start = 1
## End = 100
## Frequency = 1
##   [1]  0.04172680  0.37190682 -0.18545185 -1.38297422 -2.87593652 -2.60017605
##   [7] -1.10401719 -0.46385116  0.80339069  2.11483585  3.39124978  3.86739194
##  [13]  2.12733595  0.67590604  0.71429367  0.38928044 -1.22681923  0.02287443
##  [19] -0.57321924 -2.68376851 -4.80850095 -2.44797633 -1.73921817 -1.48773023
##  [25]  0.68920966  0.40220308 -1.58781041 -2.07300848 -2.63408197 -3.54333559
##  [31] -2.74116328 -2.54401750 -3.19570817 -0.02623669  0.41663974 -1.74485791
##  [37] -3.04847994 -1.64692948 -1.71478199  0.11340965  1.55017869  0.47317192
##  [43] -0.18521791 -0.05759091 -1.32105323 -1.22071881 -1.53827085  0.01277076
##  [49] -3.19955388 -3.11022833 -3.30621969 -2.00546537  0.18084415  0.58776479
##  [55] -0.70238414 -0.34570939  0.94209248 -0.78176791  1.30547320 -1.04054783
##  [61] -1.06897160 -1.08850000  0.06031172 -0.05724856 -1.14731083 -0.79262221
##  [67] -0.55565451 -1.55985750 -2.17062644 -1.07776017  0.51569067  2.30660050
##  [73]  1.53530426  2.55899301  1.83836277  1.08072014  1.34125182 -0.80729300
##  [79] -1.42735924 -0.42456207 -0.11625003 -0.74807460  0.70052717  0.08557377
##  [85] -0.06039041  0.04479407 -0.12657328 -1.30097021  0.81586192 -0.13139757
##  [91]  1.84725644  1.62364752  0.33080663 -0.40824385 -1.56008530 -1.63175408
##  [97] -1.36418639 -0.37209392 -0.65833401  2.03705932
```

```
lag(x.ts, k = 1)
```

```
## Time Series:
## Start = 0
## End = 99
## Frequency = 1
##   [1]  0.04172680  0.37190682 -0.18545185 -1.38297422 -2.87593652 -2.60017605
```

```
##   [7] -1.10401719 -0.46385116  0.80339069  2.11483585  3.39124978  3.86739194
##  [13]  2.12733595  0.67590604  0.71429367  0.38928044 -1.22681923  0.02287443
##  [19] -0.57321924 -2.68376851 -4.80850095 -2.44797633 -1.73921817 -1.48773023
##  [25]  0.68920966  0.40220308 -1.58781041 -2.07300848 -2.63408197 -3.54333559
##  [31] -2.74116328 -2.54401750 -3.19570817 -0.02623669  0.41663974 -1.74485791
##  [37] -3.04847994 -1.64692948 -1.71478199  0.11340965  1.55017869  0.47317192
##  [43] -0.18521791 -0.05759091 -1.32105323 -1.22071881 -1.53827085  0.01277076
##  [49] -3.19955388 -3.11022833 -3.30621969 -2.00546537  0.18084415  0.58776479
##  [55] -0.70238414 -0.34570939  0.94209248 -0.78176791  1.30547320 -1.04054783
##  [61] -1.06897160 -1.08850000  0.06031172 -0.05724856 -1.14731083 -0.79262221
##  [67] -0.55565451 -1.55985750 -2.17062644 -1.07776017  0.51569067  2.30660050
##  [73]  1.53530426  2.55899301  1.83836277  1.08072014  1.34125182 -0.80729300
##  [79] -1.42735924 -0.42456207 -0.11625003 -0.74807460  0.70052717  0.08557377
##  [85] -0.06039041  0.04479407 -0.12657328 -1.30097021  0.81586192 -0.13139757
##  [91]  1.84725644  1.62364752  0.33080663 -0.40824385 -1.56008530 -1.63175408
##  [97] -1.36418639 -0.37209392 -0.65833401  2.03705932
```

```
#      x.ts
b.ts <- ts(c(10, 20, 30, 40, 50, 60), start = c(2022, 1), frequency = 12)

#   ts.intersect()    x.ts     lag
ts.intersect(b.ts, b.ts1 = lag(x = b.ts, k = -1), b.ts2 = lag(x = b.ts, k = -2), b.ts3 = lag(x =
```

```
##          b.ts b.ts1 b.ts2 b.ts3
## Apr 2022   40    30    20    10
## May 2022   50    40    30    20
## Jun 2022   60    50    40    30
```

```
#b.ts1 b.ts2  b.ts3    b.ts     1 2 3
b.ts1 = lag(x = b.ts, k = -1)
b.ts2 = lag(x = b.ts, k = -2)
b.ts3 = lag(x = b.ts, k = -3)
b.ts
```

```
##      Jan Feb Mar Apr May Jun
## 2022  10  20  30  40  50  60
```

```
b.ts1
```

```
##      Feb Mar Apr May Jun Jul
## 2022  10  20  30  40  50  60
```

```
b.ts2
```

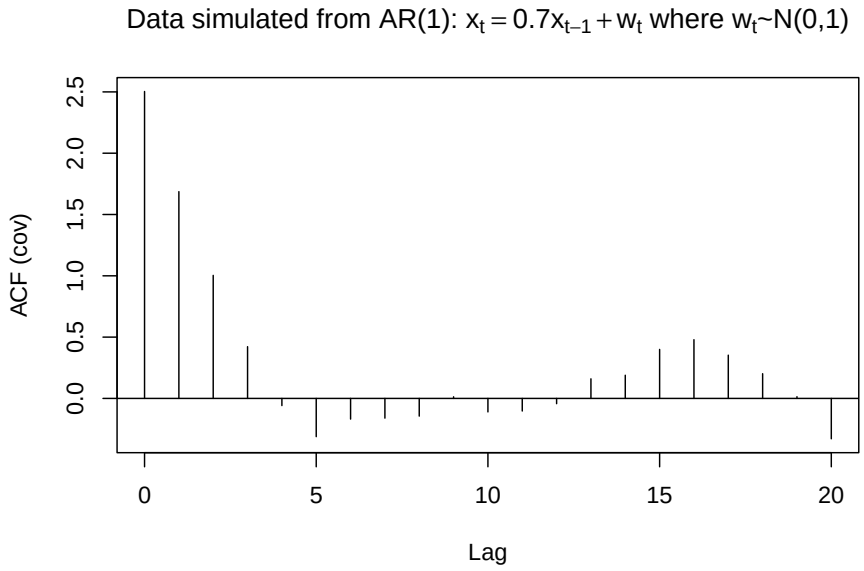
```
##      Mar Apr May Jun Jul Aug
## 2022  10  20  30  40  50  60
```

```
b.ts3
```

```
##      Apr May Jun Jul Aug Sep
## 2022  10  20  30  40  50  60
```

- The `ts.intersect()` function finds the intersection of the four different "variables".
- The`cor()`function finds the estimated Pearson correlation coefficients between all variable pairs. Notice how close these correlations are to the autocorrelations!
- The `scatterplotMatrix()` function finds a scatter plot matrix. The function is in the car package.

```
#Used for estimation later in course

  gamma.x <- acf(x = x, type = "covariance", main =
    expression(paste("Data simulated from AR(1): ", x[t] == 0.7*x[t-1] + w[t], " where
```



```
  gamma.x
```

```
##
## Autocovariances of series 'x', by lag
##
##       0       1       2       3       4       5       6       7       8       9
##  2.5033  1.6864  1.0036  0.4219 -0.0586 -0.3120 -0.1687 -0.1608 -0.1445  0.0133
##      10      11      12      13      14      15      16      17      18      19
## -0.1102 -0.1029 -0.0430  0.1599  0.1892  0.3997  0.4790  0.3529  0.2015  0.0140
##      20
## -0.3293
```

```
  mean(x)
```

```
## [1] -0.4963419
```

Example 7.2. OSU enrollment data

Click here to download data.

```
  osu.enroll <- read.csv(file = "OSU_enroll.csv", stringsAsFactors = TRUE)
  head(osu.enroll)
```

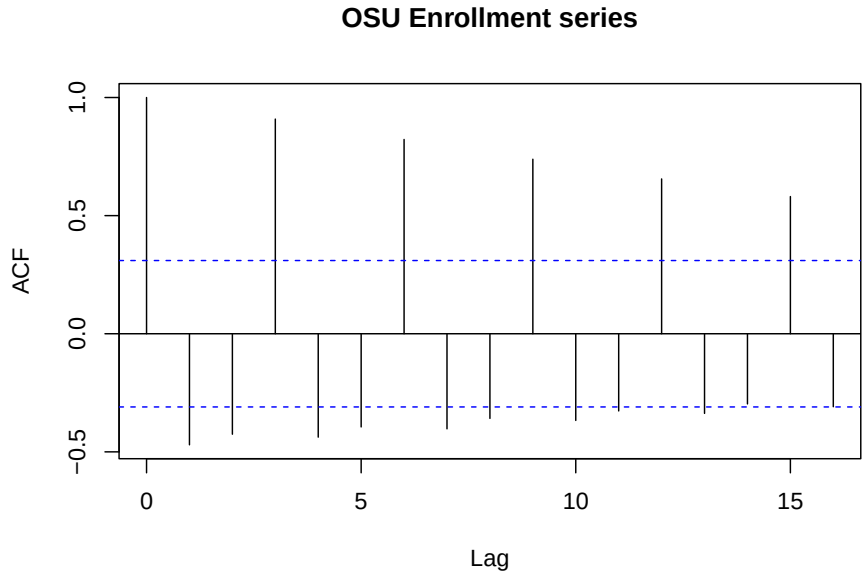
```
##   t Semester Year Enrollment      date
## 1 1     Fall 1989      20110 8/31/1989
## 2 2   Spring 1990      19128  2/1/1990
## 3 3   Summer 1990       7553  6/1/1990
## 4 4     Fall 1990      19591 8/31/1990
## 5 5   Spring 1991      18361  2/1/1991
## 6 6   Summer 1991       6702  6/1/1991
```

```
  tail(osu.enroll)
```

```
##     t Semester Year Enrollment      date
## 35 35   Spring 2001      20004  2/1/2001
## 36 36   Summer 2001       7558  6/1/2001
## 37 37     Fall 2001      21872 8/31/2001
## 38 38   Spring 2002      20922  2/1/2002
## 39 39   Summer 2002       7868  6/1/2002
## 40 40     Fall 2002      22992 8/31/2002
```

```
  x <- osu.enroll$Enrollment
```

```
rho.x <- acf(x = x, type = "correlation", main = "OSU Enrollment series")
```




```
rho.x
```

```
##
## Autocorrelations of series 'x', by lag
##
##      0      1      2      3      4      5      6      7      8      9     10
##  1.000 -0.470 -0.425  0.909 -0.438 -0.395  0.822 -0.403 -0.358  0.739 -0.367
##     11     12     13     14     15     16
## -0.327  0.655 -0.337 -0.297  0.581 -0.309
```

```
rho.x$acf[1:9]
```

```
## [1]  1.0000000 -0.4702315 -0.4253427  0.9087421 -0.4377336 -0.3946048  0.8224660
## [8] -0.4025871 -0.3584216
```

Notes:

- There are some large autocorrelations. This is a characteristic of a non-stationary series (seasonal factor). We will examine this more later.

- Because the series is not stationary, the hypothesis test for $\rho(h) = 0$ should not be done here using the methods discussed earlier.

- There is a pattern among the autocorrelations. What does this correspond to? (seasonal factor) (similar value/behavior happens during specific period of time/months across different years)

Chapter 8

Resolving Non-Stationarity Problems

8.1 Differencing

Differencing helps to create the constant mean needed for stationarity. We will use differencing a lot!

1st differences: $x_t – x_{t-1} = \nabla x_t$

2nd differences: $(x_t – x_{t-1}) – (x_{t-1} – x_{t-2}) = \nabla x_t – \nabla x_{t-1} = \nabla^2 x_t$

Taking "differences" between successive data values in the time series helps to remove trend. Specifically, 1st differences help remove linear trend and 2nd differences help remove quadratic trend.

Why does this work? Consider the linear trend model $x_t = \beta_0 + \beta_1 t$ where t = time and $ _1 $0. Then

$$x_t – x_{t-1} = \beta_0 + \beta_1 t – [\beta_0 + \beta_1 (t–1)] = \beta_1$$

which is not dependent on t.

Example 8.1. Non-stationarity in the mean

Click Here to download data.

Below is the code for a plot of xt vs. t and the ACF for xt.

```
nonstat.mean <- read.csv(file="nonstat.mean.csv")
head(nonstat.mean)
```

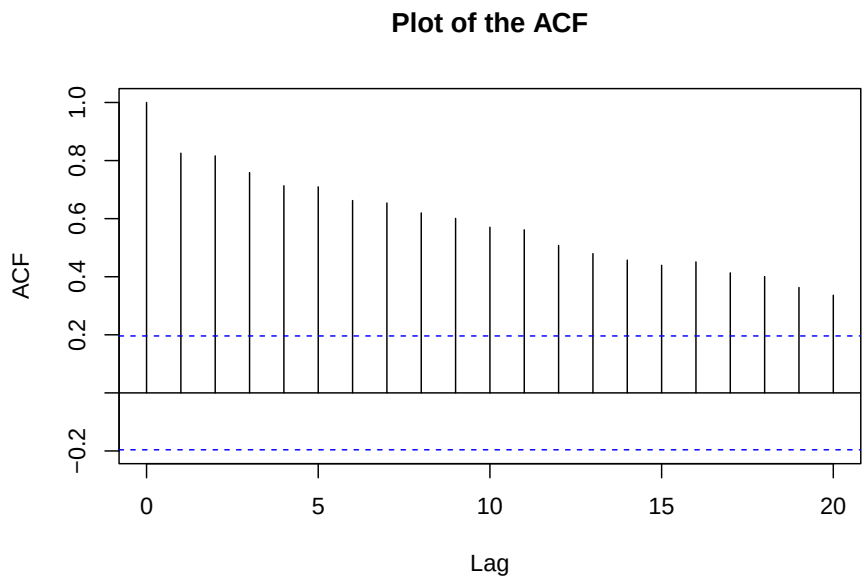
```
##   time     x
## 1    1  1.31
## 2    2 13.67
## 3    3  6.29
## 4    4 -0.95
## 5    5  9.59
## 6    6 -0.45
```

```
tail(nonstat.mean)
```

```
##     time      x
## 95    95  92.69
## 96    96  91.22
## 97    97 100.00
## 98    98 102.80
## 99    99  93.82
## 100  100 108.72
```

```
dev.new(width=8, height=6, pointsize=10)

plot(x=nonstat.mean$x, ylab=expression(x[t]), xlab="t (time)", type="l", col="red", ma

points(x=nonstat.mean$x, col="blue", pch=20)
```

```
acf(x=nonstat.mean$x, type="correlation", main="Plot of the ACF")
```



Whenever you see a ACF plot like this(high correlation), it's highly likely that there's a non-stationarity in mean.

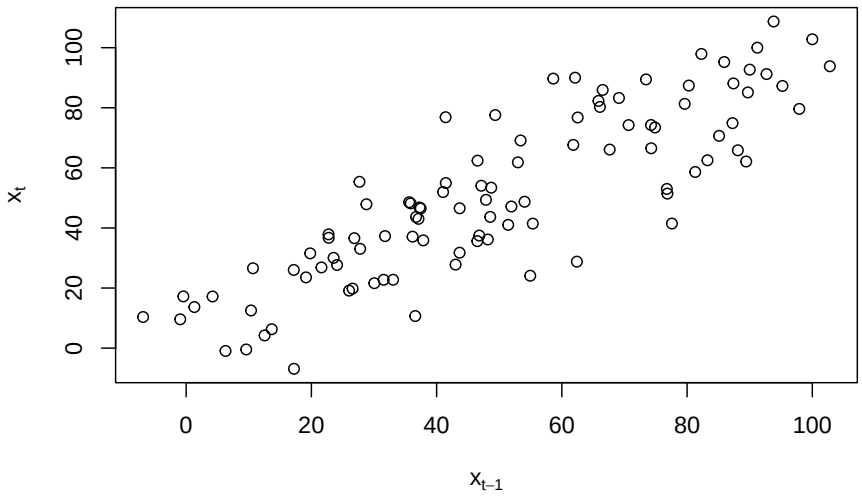
```
# scatter plot of x_t vs. x_t-1
x.ts <- ts(nonstat.mean[,2])
set1 <- ts.intersect(x.ts, x.ts1=lag(x=x.ts, k=-1))
head(set1)
```

```
##       x.ts x.ts1
## [1,] 13.67  1.31
## [2,]  6.29 13.67
## [3,] -0.95  6.29
## [4,]  9.59 -0.95
## [5,] -0.45  9.59
## [6,] 17.22 -0.45
```

```
cor(set1)
```

```
##           x.ts    x.ts1
## x.ts  1.000000 0.857779
## x.ts1 0.857779 1.000000
```

```
# Need as.numeric() so that plot.ts() is not run, want plot.default()

plot(y=as.numeric(set1[,1]), x=as.numeric(set1[,2]), ylab=expression(x[t]), type="p", xlab=expres
```



```
# you can see from the graph there's a positive linear relationship, this is why the c
```

- This data is said to have "nonstationarity in the mean" because the mean of x_t, μ_t, appears to be changing as a function of time.
- Why is there large positive autocorrelation at lag = 1, 2, … ?
  - This is b/c we have a positive linear relationship btw x_t and x_{t-1}.

Below is the code to find the first differences:

```
# Find first differences

first.diff <- diff(x=nonstat.mean$x, lag=1, differences=1)
first.diff[1:5]
```

```
## [1]  12.36  -7.38  -7.24  10.54 -10.04
```

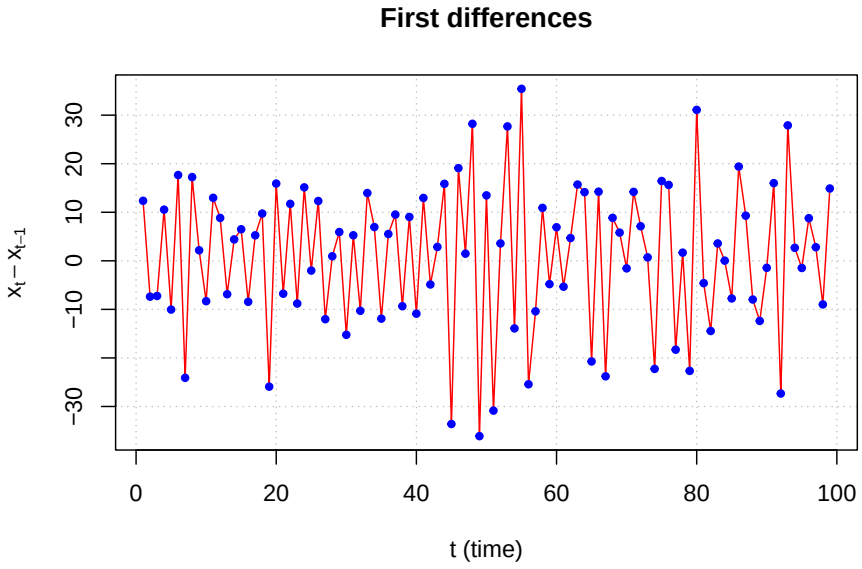
```
nonstat.mean$x[2]-nonstat.mean$x[1]
```

```
## [1] 12.36
```

```
nonstat.mean$x[3]-nonstat.mean$x[2]
```

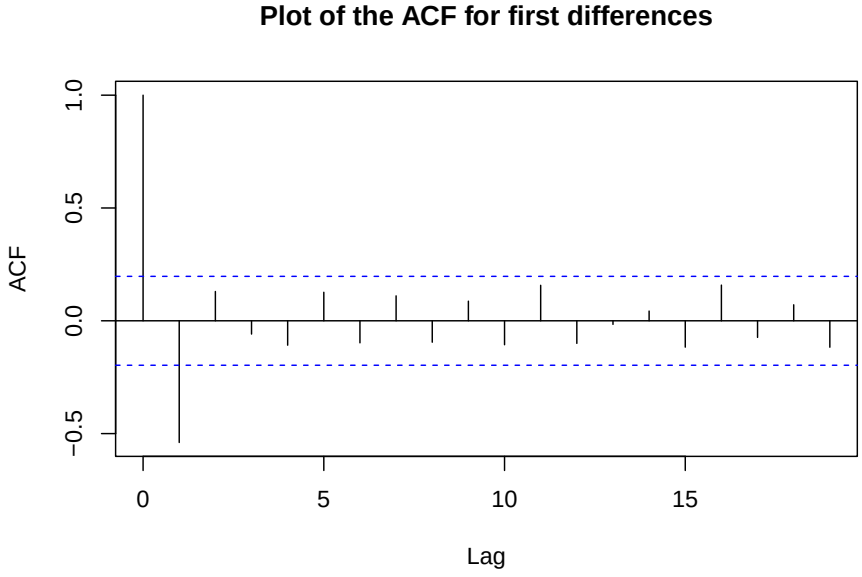
```
## [1] -7.38
```

```
plot(x= first.diff, ylab=expression(x[t]-x[t-1]), xlab="t (time)", type="l", col="red"

points(x=first.diff, col="blue", pch=20)
```



The linear relationship just disappears after first differencing!

```
acf(x=first.diff, type="correlation", main="Plot of the ACF for first differences")
```



If you want xt and xt - xt-1 in the same data frame, use the `ts.intersect()`

function:

```
x <- ts(data=nonstat.mean$x)
x.diff1 <- ts(data=first.diff, start=2)
ts.intersect(x, x.diff1)
```

```
## Time Series:
## Start = 2
## End = 100
## Frequency = 1
##          x x.diff1
##   2  13.67   12.36
##   3   6.29   -7.38
##   4  -0.95   -7.24
##   5   9.59   10.54
##   6  -0.45  -10.04
##   7  17.22   17.67
##   8  -6.88  -24.10
##   9  10.36   17.24
##  10  12.53    2.17
##  11   4.22   -8.31
##  12  17.19   12.97
##  13  26.02    8.83
##  14  19.14   -6.88
##  15  23.55    4.41
##  16  30.05    6.50
##  17  21.61   -8.44
##  18  26.86    5.25
##  19  36.59    9.73
##  20  10.67  -25.92
##  21  26.58   15.91
##  22  19.81   -6.77
##  23  31.54   11.73
##  24  22.74   -8.80
##  25  37.87   15.13
##  26  35.86   -2.01
##  27  48.18   12.32
##  28  36.16  -12.02
##  29  37.10    0.94
##  30  43.03    5.93
##  31  27.79  -15.24
##  32  33.05    5.26
##  33  22.76  -10.29
##  34  36.72   13.96
##  35  43.67    6.95
```

```
##  36  31.76  -11.91
##  37  37.28    5.52
##  38  46.81    9.53
##  39  37.46   -9.35
##  40  46.50    9.04
##  41  35.61  -10.89
##  42  48.56   12.95
##  43  43.68   -4.88
##  44  46.54    2.86
##  45  62.40   15.86
##  46  28.79  -33.61
##  47  47.88   19.09
##  48  49.36    1.48
##  49  77.58   28.22
##  50  41.47  -36.11
##  51  54.95   13.48
##  52  24.10  -30.85
##  53  27.68    3.58
##  54  55.36   27.68
##  55  41.44  -13.92
##  56  76.86   35.42
##  57  51.43  -25.43
##  58  41.03  -10.40
##  59  51.94   10.91
##  60  47.14   -4.80
##  61  54.05    6.91
##  62  48.72   -5.33
##  63  53.41    4.69
##  64  69.12   15.71
##  65  83.26   14.14
##  66  62.53  -20.73
##  67  76.77   14.24
##  68  52.98  -23.79
##  69  61.83    8.85
##  70  67.64    5.81
##  71  66.08   -1.56
##  72  80.28   14.20
##  73  87.40    7.12
##  74  88.11    0.71
##  75  65.85  -22.26
##  76  82.29   16.44
##  77  97.92   15.63
##  78  79.62  -18.30
##  79  81.30    1.68
##  80  58.63  -22.67
##  81  89.71   31.08
```

```
##  82  85.11   -4.60
##  83  70.66  -14.45
##  84  74.24    3.58
##  85  74.27    0.03
##  86  66.52   -7.75
##  87  85.93   19.41
##  88  95.23    9.30
##  89  87.26   -7.97
##  90  74.88  -12.38
##  91  73.44   -1.44
##  92  89.44   16.00
##  93  62.11  -27.33
##  94  90.01   27.90
##  95  92.69    2.68
##  96  91.22   -1.47
##  97 100.00    8.78
##  98 102.80    2.80
##  99  93.82   -8.98
## 100 108.72   14.90
```

Why does the data set start at 2?

Other types of differencing:

- 2nd differences: `diff(x, lag = 1, differences = 2)`
- xt – xt-2: `diff(x, lag = 2, differences = 1)`; this can be useful when there is a "seasonal" trend

```
# Second differences
diff(x, lag = 1, differences=2)
```

```
## Time Series:
## Start = 3
## End = 100
## Frequency = 1
##  [1] -19.74   0.14  17.78 -20.58  27.71 -41.77  41.34 -15.07 -10.48  21.28
## [11]  -4.14 -15.71  11.29   2.09 -14.94  13.69   4.48 -35.65  41.83 -22.68
## [21]  18.50 -20.53  23.93 -17.14  14.33 -24.34  12.96   4.99 -21.17  20.50
## [31] -15.55  24.25  -7.01 -18.86  17.43   4.01 -18.88  18.39 -19.93  23.84
## [41] -17.83   7.74  13.00 -49.47  52.70 -17.61  26.74 -64.33  49.59 -44.33
## [51]  34.43  24.10 -41.60  49.34 -60.85  15.03  21.31 -15.71  11.71 -12.24
## [61]  10.02  11.02  -1.57 -34.87  34.97 -38.03  32.64  -3.04  -7.37  15.76
## [71]  -7.08  -6.41 -22.97  38.70  -0.81 -33.93  19.98 -24.35  53.75 -35.68
## [81]  -9.85  18.03  -3.55  -7.78  27.16 -10.11 -17.27  -4.41  10.94  17.44
## [91] -43.33  55.23 -25.22  -4.15  10.25  -5.98 -11.78  23.88
```

```
(nonstat.mean$x[3] - nonstat.mean$x[2]) - (nonstat.mean$x[2] - nonstat.mean$x[1])
```

```
## [1] -19.74
```

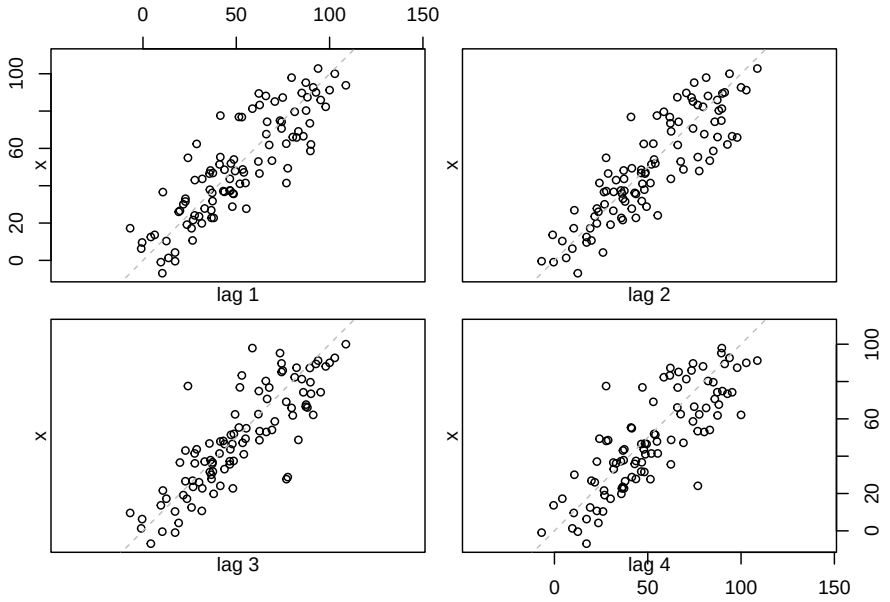
```
# x_t - x_t-2
diff(x, lag = 2, differences = 1)
```

```
## Time Series:
## Start = 3
## End = 100
## Frequency = 1
##  [1]   4.98 -14.62   3.30   0.50   7.63  -6.43  -6.86  19.41  -6.14   4.66
## [11]  21.80   1.95  -2.47  10.91  -1.94  -3.19  14.98 -16.19 -10.01   9.14
## [21]   4.96   2.93   6.33  13.12  10.31   0.30 -11.08   6.87  -9.31  -9.98
## [31]  -5.03   3.67  20.91  -4.96  -6.39  15.05   0.18  -0.31  -1.85   2.06
## [41]   8.07  -2.02  18.72 -17.75 -14.52  20.57  29.70  -7.89 -22.63 -17.37
## [51] -27.27  31.26  13.76  21.50   9.99 -35.83   0.51   6.11   2.11   1.58
## [61]  -0.64  20.40  29.85  -6.59  -6.49  -9.55 -14.94  14.66   4.25  12.64
## [71]  21.32   7.83 -21.55  -5.82  32.07  -2.67 -16.62 -20.99   8.41  26.48
## [81] -19.05 -10.87   3.61  -7.72  11.66  28.71   1.33 -20.35 -13.82  14.56
## [91] -11.33   0.57  30.58   1.21   7.31  11.58  -6.18   5.92
```

```
nonstat.mean$x[3] - nonstat.mean$x[1]
```

```
## [1] 4.98
```

```
# Plot
lag.plot(x, lags = 4, layout = c(2,2), do.lines = FALSE)
```



Note: There are formal hypothesis tests to determine if differencing is needed. This corresponds to an area of time series known as "unit root" testing. The name will be clear once we examine autoregressive models in detail.

8.2 Backshift Operator

A convenient way to represent differencing in time series models is to use the "backshift operator". It is denoted by "B" and defined as follows:

$$Bx_t = x_{t-1}$$

Notice that x_t moved back one-time period when the backshift operator was applied to it.

In general, $B^2x_t = x_{t-2}, B^3x_t = x_{t-3}, ..., and \quad B^kx_t = x_{t-k}$.

Notes:

- Let C be a constant not indexed by time. Then $BC = C$.
- $(1 - B)x_t = x_t - x_{t-1} = \nabla x_t$
- $B \times B = B^2$
- $(1-B)^2x_t = (1-2B+B^2)x_t = x_t - 2Bx_t + B^2x_t = x_t - 2x_{t-1} + x_{t-2} = x_t - x_{t-1} - x_{t-1} + x_{t-2} = (x_t - x_{t-1})$

- $(1-B)^0 x_t = x_t$
- $(1-B)x_t$ can be thought of as a "linear filter" since the linear trend is being filtered out of the time series.

Example 8.2. Moving Average

$m_t = \frac{w_t+w_{t-1}+w_{t-2}}{3}$, where $w_t \sim \text{ind}N(0,\sigma_w^2)\forall t = 1,..n$ can be represented by $(1+B+B^2)\frac{w_t}{3}$

Example 8.3. Autoregression

$x_t = 0.7x_{t-1} + w_t$, where $w_t \sim \text{ind}N(0,\sigma_w^2)\forall t = 1,..n$ can be represented by $(1-0.7B)x_t = w_t$

Example 8.4. first differencing needed example

Consider the following model:

$(1-0.7B)(1-B)x_t = w_t$, where $w_t \sim \text{ind}N(0,\sigma_w^2)\forall t = 1,..n$. This simplifies to

$$(1-0.7B)(x_t-x_{t-1}) = w_t \iff x_t-x_{t-1}-0.7Bx_t+0.7Bx_{t-1} = w_t \iff x_t = x_{t-1}+0.7x_{t-1}-0.7x_{t-2}+w_t \iff x_t = 1.7$$

Later in the course, we will identify this as a ARIMA(1,1,0) model.

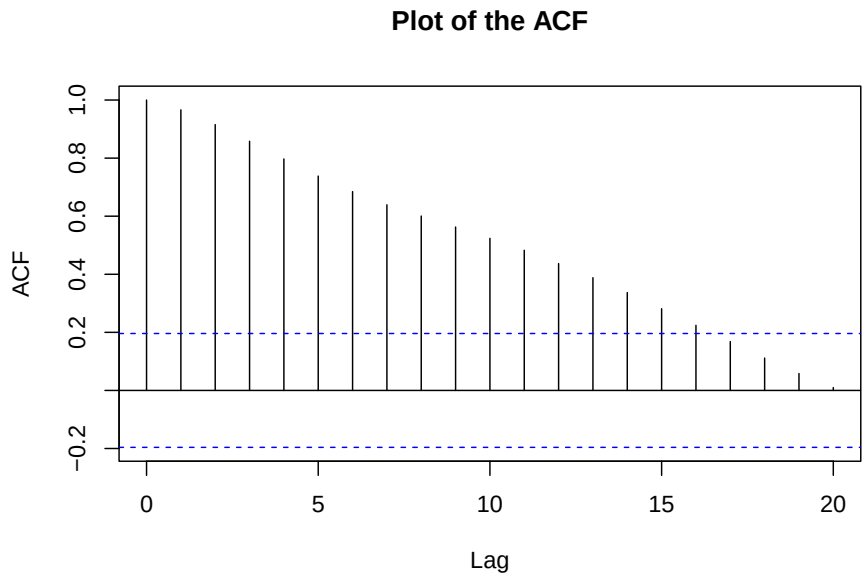
Suppose a realization of a time series is simulated from this model. Below is a plot of the data.

```
set.seed(7328)
w <- rnorm(n = 200, mean = 0, sd = 1)

x <- numeric(length = 200)
x.1 <- 0
x.2 <- 0
for (i in 1:length(x)) {
  x[i] <- 1.7*x.1 - 0.7*x.2 + w[i]
  x.2 <- x.1
  x.1 <- x[i]
}

#Do not use first 100
X <- x[101:200]
```

```
dev.new(width = 8, height = 6, pointsize = 10)
plot(x = X, ylab = expression(x[t]), xlab = "t", type =
    "l", col = "red", lwd = 1 , main =
    expression(paste("Data simulated from ", (1-0.7*B)*(1-
```

```
    B)*x[t] == w[t], " where ", w[t], "~N(0,1)")),
    panel.first=grid(col = "gray", lty = "dotted"))
points(x = X, pch = 20, col = "blue")

# from the graph there's nonstationarity in mean
```

```
acf(x=X, type="correlation", main="Plot of the ACF")
```

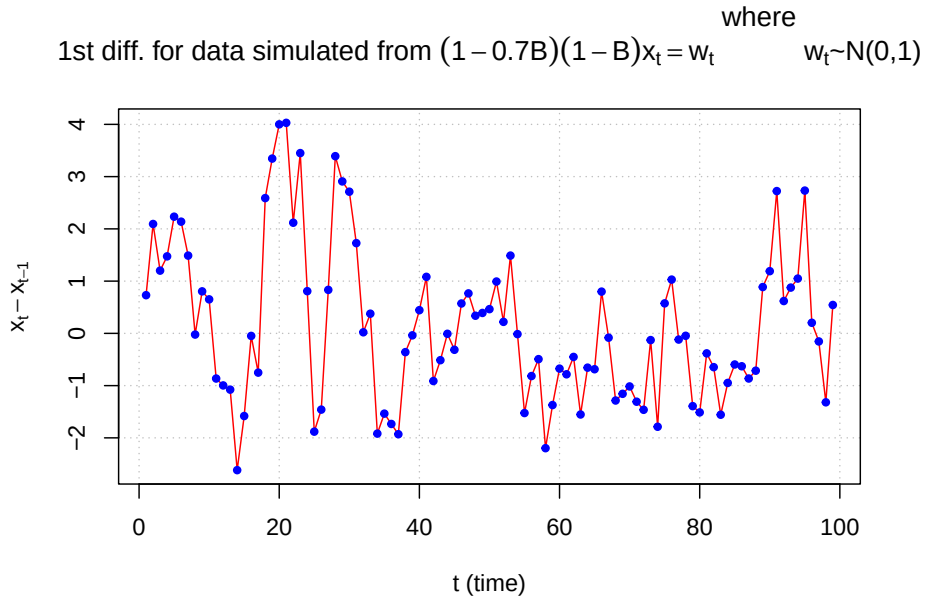


After the first differences:

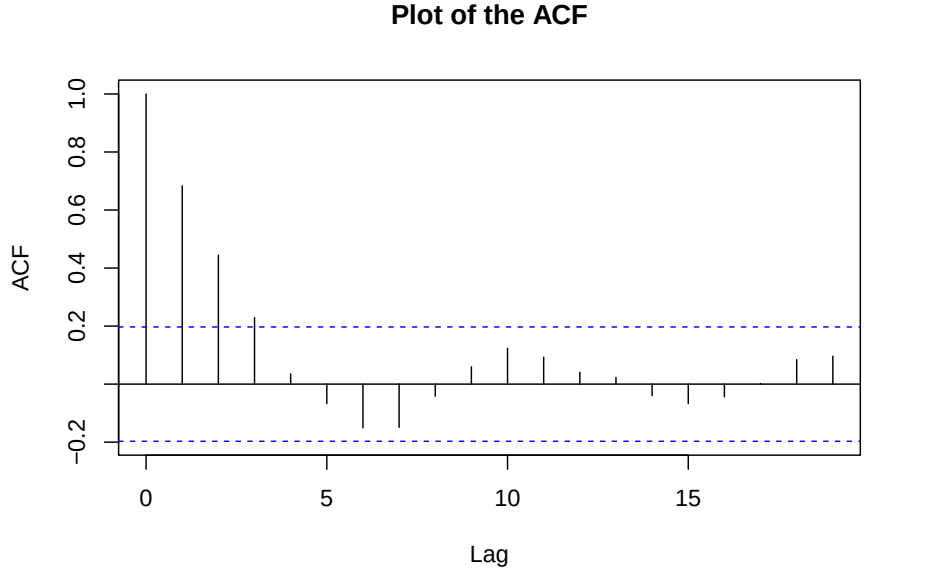
```
# Finf first differences

plot(x=diff(x=X, lag=1, differences = 1), ylab=expression(x[t]-x[t-1]), xlab="t (time)"
   ", w[t], "~N(0,1)")), panel.first=grid(col="gray",lty="dotted"))
```

```
## Warning in title(...): font metrics unknown for character 0xa

## Warning in title(...): font metrics unknown for character 0xa
```

```
points(x=diff(x=X, lag=1, differences = 1), pch=20, col="blue")
```




```
acf(x=diff(x=X, lag=1, differences = 1), type="correlation", main="Plot of the ACF")
```



An easier way is to use `arima.sim()` to simulate the data.

8.3 Fractional Differencing

Use fractional powers of B between –0.5 to 0.5 to do the differencing. This is used with long-memory time series.

Notes:

- Differencing is often used to help make a nonstationary in the mean time series stationary. Unfortunately in real applications, we do not know what exact level of differencing is needed (we can approximate it). If a too high of level of differencing is done, this can hurt a time series model. As a compromise between differencing and not differencing at all, fractional differencing can be used.

- The reason why these are called a "long memory" time series can be seen from a "short memory" time series. A short memory stationary process will have $\rho(h) \to 0$ "quickly" as $h \to \infty$. A long memory time series does not and has $\rho(h) \to 0$ "slowly". More on this later in the course.
- The fractional difference series can be represented as

$$\nabla^d x_t = \sum_{j=0}^{\infty} \pi_j x_{t-j}$$

where the π_j are found through a Taylor series expansion of $(1-B)^d$.

8.4 Transformations

In regression analysis, transformations of the response variable are taken to induce approximate constant variance. In a similar manner, we can take transformations of xt to help make a nonstationary in the variance time series be approximately stationary in the variance.

Example 8.5. Johnson & Johnson earnings per share data

This data comes from Shumway and Stoffer's book.

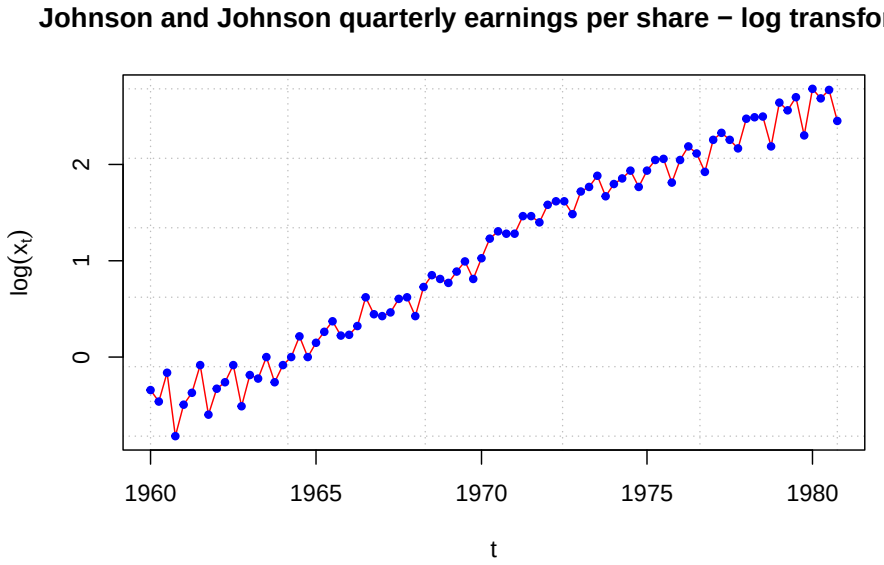
```
library(astsa)
x <- jj
x
```

```
##           Qtr1      Qtr2      Qtr3      Qtr4
## 1960  0.710000  0.630000  0.850000  0.440000
## 1961  0.610000  0.690000  0.920000  0.550000
## 1962  0.720000  0.770000  0.920000  0.600000
```

```
## 1963  0.830000  0.800000  1.000000  0.770000
## 1964  0.920000  1.000000  1.240000  1.000000
## 1965  1.160000  1.300000  1.450000  1.250000
## 1966  1.260000  1.380000  1.860000  1.560000
## 1967  1.530000  1.590000  1.830000  1.860000
## 1968  1.530000  2.070000  2.340000  2.250000
## 1969  2.160000  2.430000  2.700000  2.250000
## 1970  2.790000  3.420000  3.690000  3.600000
## 1971  3.600000  4.320000  4.320000  4.050000
## 1972  4.860000  5.040000  5.040000  4.410000
## 1973  5.580000  5.850000  6.570000  5.310000
## 1974  6.030000  6.390000  6.930000  5.850000
## 1975  6.930000  7.740000  7.830000  6.120000
## 1976  7.740000  8.910000  8.280000  6.840000
## 1977  9.540000 10.260000  9.540000  8.729999
## 1978 11.880000 12.060000 12.150000  8.910000
## 1979 14.040000 12.960000 14.850000  9.990000
## 1980 16.200000 14.670000 16.020000 11.610000
```

```
dev.new(width = 8, height = 6, pointsize = 10)  #Opens up wider plot window than the default (goo

plot(x = x, ylab = expression(x[t]), xlab = "t", type =
    "l", col = "red", lwd = 1, main = "Johnson and Johnson
    quarterly earnings per share", panel.first = grid(col =
    "gray", lty = "dotted"))

points(x = x, pch = 20, col = "blue")


# an upward trend in mean
# and the variance is increasing as well
# both non-stationarity in mean and variance
```

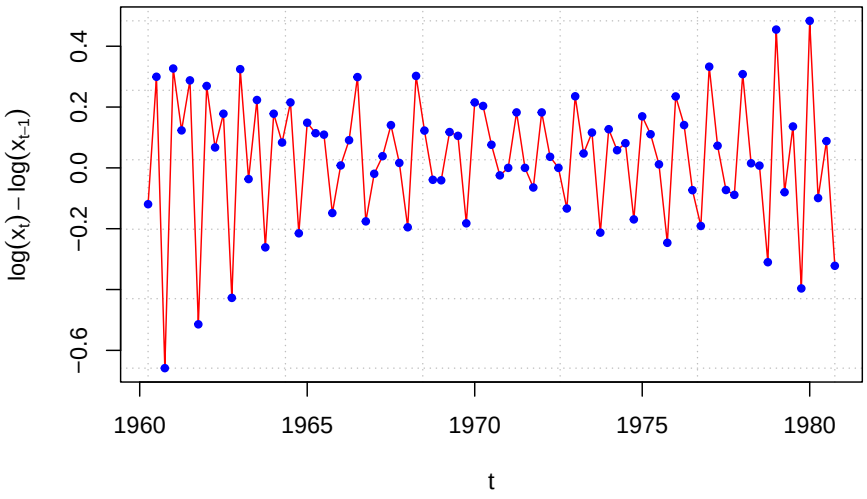
```
plot(x = log(x), ylab = expression(log(x[t])), xlab = "t", type = "l", col = "red", lwd = 1 ,
     main = "Johnson and Johnson quarterly earnings per share - log transformed",
     panel.first = grid(col = "gray", lty = "dotted"))
points(x = log(x), pch = 20, col = "blue")
```




```
#we deal with non-stationarity in variance
```

```
#Still some variance issues???
plot(x = diff(x = log(x), lag = 1, differences = 1), ylab = expression(log(x[t]) - log
     main = "Johnson and Johnson quarterly earnings per share - log transformed and 1st
     panel.first = grid(col = "gray", lty = "dotted"))
points(x = diff(x = log(x), lag = 1, differences = 1), pch = 20, col = "blue")
```

nson and Johnson quarterly earnings per share − log transformed and



Notes:

- Typical transformations include $log(x_t)$, $\sqrt{x_t}$ and x_t^{-1} There are a few different ways to deciding between the appropriate transformations. Usually, I will try all of these transformations and examine a plot of the transformed data over time to determine if the transformation worked. Constants may need to be added to x_t if x_t can be less than 0.

- Regression courses sometimes teach the Box-Cox family of transformations to determine an appropriate transformation. The process involves finding the "best" λ to transform x_t in the following manner:

$$y_t = \begin{cases} (x_t^\lambda - 1)/\lambda & \text{if } \lambda \neq 0 \\ log(x_t) & \text{if } \lambda = 0 \end{cases}$$

I have not found it used often in time series analysis. For example, Shumway and Stoffer suggest it could be used here, but do not explore it further. The `BoxCox()` and `BoxCox.lambda()` functions from the forecast package provide ways to obtain it. Please see my program.

```
#How could one do a Box-Cox transformation here?

library(package = forecast)
```

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

```
##
## Attaching package: 'forecast'
```

```
## The following object is masked from 'package:astsa':
##
##     gas
```

```
save.it <- BoxCox(x = x, lambda = "auto")
# save.it - this has the transformed data
names(save.it) # Not useful
```

```
## NULL
```

```
attributes(save.it)  # Older form of R coding shows it
```

```
## $tsp
## [1] 1960.00 1980.75    4.00
##
## $class
## [1] "ts"
##
## $lambda
## [1] 0.1540791
```

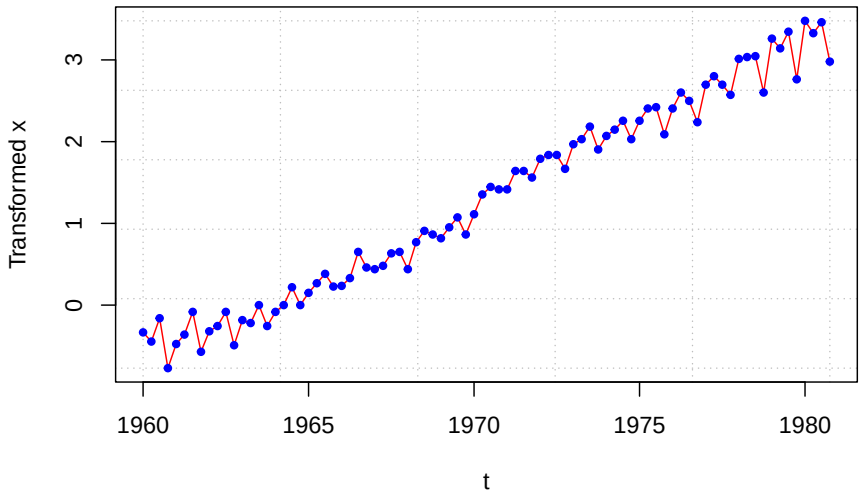
```
attributes(save.it)$lambda
```

```
## [1] 0.1540791
```

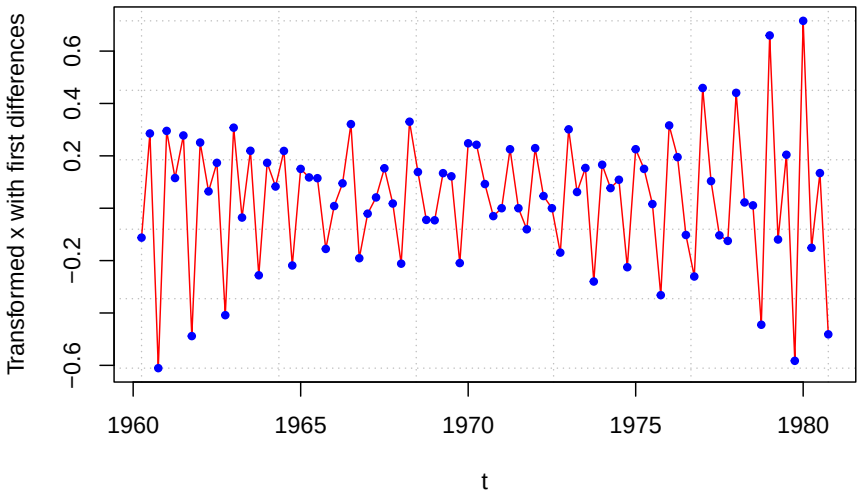
```
# Looking at the code in BoxCox() leads one to use the function below to
#   obtain lambda
BoxCox.lambda(x = x)
```

```
## [1] 0.1540752
```

```
lambda <- BoxCox.lambda(x = x)

# Don't see a benefit from using this transformation over log()
plot(x = (x^lambda - 1)/lambda, ylab = "Transformed x", xlab = "t", type = "l", col = '
points(x = (x^lambda - 1)/lambda, pch = 20, col = "blue")
```




```
plot(x = diff(x = (x^lambda - 1)/lambda, lag = 1, differences = 1),
    ylab = "Transformed x with first differences", xlab = "t", type = "l",
    col = "red", lwd = 1 ,
    panel.first = grid(col = "gray", lty = "dotted"))
points(x = diff(x = (x^lambda - 1)/lambda, lag = 1, differences = 1), pch = 20, col = "blue")
```



- If the variance stabilizing transformation is needed, do this before differencing (see Wei's time series book for a discussion). For example, suppose differencing and a natural log variance stabilizing transformation is needed. Then examine $log(x_t)$–$log(x_{t-1})$ instead of $log(x_t – x_{t-1})$.
- Variance stabilizing transformations often help with normality assumption of w_t.

Chapter 9

Autoregressive Models

9.1 AR Models

Suppose we have a time series x_t for $t = 1, ..., n$. Could we use the regression model of

$$x_t = \beta_0 + \beta_1 t + \epsilon_t,$$

where $\epsilon_t \sim \text{ind} N(0, \sigma^2)$ for it? Yes, but stated confidence levels and type I error rates will likely be incorrect. Thus, inferences can be incorrect. The reason is the likely dependence in the time series.

While one may be able to find a set of explanatory variables that help to detrend a response variable series so it appears that white noise is leftover (i.e., it looks like the error terms are independent), this is often not possible.

Instead, *autoregressive integrated moving average models (ARIMA)* are used for time series data. These models were first developed by Box and Jenkins (1970). We have already touched on all parts of this type of model:

- AR: Autoregressive
- MA: Moving average
- I: Integrated (closely related to differencing)

Often, people will refer to ARIMA models as an ARIMA process as well. This refers to how x_t comes about through a linear process. Both "model" and "process" will be used interchangeably.

9.2 Autoregressive models – AR(p)

An autoregressive model uses past observations of x_t to predict future observations. Specifically, the present value of x_t is explained by a linear function of p past values of $x_{t-1}, ..., x_{t-p}$.

Example 9.1. AR(1) from earlier

$x_t = 0.7x_{t-1} + w_t$, where $w_t \sim \text{ind}N(0,1) \forall t = 1, ..., n$

Therefore, $x_2 = 0.7x_1 + w_2$, $x_3 = 0.7x_2 + w_3, ...$

Future values may be "forecasted" by past values using

$x_{n+1} = 0.7x_n$

More on this later in the section.

An autoregressive model of order p, denoted as AR(p), is

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

where

$\phi_1, \phi_2, ..., \phi_p$ are parameters and $w_t \sim \text{ind}N(0, \sigma_w^2) \forall t = 1, ..., n$ (i.e., white noise)

Notes:

- To find parameter estimates later in this section, we will assume $w_t \sim \text{ind}N(0, \sigma_w^2)$
- WLOG, the mean of x_t, μ, will be assumed to be 0 when we write out a general model. HOWEVER, $\mu \neq 0$ in most applications! The WLOG part is here because one can simply replace x_t with $x_t - \mu$. The end result is an autoregressive model of

$$x_t - \mu = \phi_1(x_{t-1} - \mu) + \phi_2(x_{t-2} - \mu) + ... + \phi_p(x_{t-p} - \mu) + w_t \iff x_t = \mu(1 - \phi_1 - \phi_2 - ... - \phi_p) + \phi_1 x_{t-1} + \phi_2 x_{t-2}$$

where $\alpha = \mu(1 - \phi_1 - \phi_2 - ... - \phi_p)$. The α (simply a constant) does not affect the dependence structure among the x_t. This is why the common convention is to exclude the parameter when introducing these models.

When we estimate the model, we will almost always include an estimate of α.

- The AR(p) model written in vector form is

$x_t = \phi' x_{t-1} + w_t$

where

$\phi = (\phi_1, \phi_2, ..., \phi_p)'$

$x_{t-1} = (x_{t-1}, x_{t-2}, ..., x_{t-p})'$

- The AR(p) model written in backshift notation is

$(1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p B^p) x_t = w_t$

and $\phi(B) x_t = w_t$

where $\phi(B) = (1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p B^p)$ is called the autoregressive operator. We will be using this notation throughout the course.

- The model can be re-expressed as a linear combination of w_t's by "iterating backwards". For example, an AR(1) model can be represented as:

(Note: the 1 on ϕ_1 was dropped)

$x_t = \phi x_{t-1} + w_t = \phi(\phi x_{t-2} + w_{t-1}) + w_t = \phi^2 x_{t-2} + \phi w_{t-1} + w_t = \phi^2(\phi w_{t-3} + w_{t-2}) + \phi w_{t-1} + w_t = \phi^3 x_{t-3} + \phi^2 w_{t-2} + \phi w_{t-}$

,

provided that $|\phi| < 1$ and variance of x_t is bounded.

To see this, note that the model can be rewritten as

$$(1 - \phi B) x_t = w_t \implies x_t = \frac{1}{1 - \phi B} w_t \implies x_t = (1 + \phi B + \phi^2 B^2 + ...) w_t = \sum_{j=0}^{\infty} \phi^j B^j w_t = \sum_{j=0}^{\infty} \phi^j w_{t-j}$$

using the sum of an infinite series. Remember that $\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}, |a| < 1$. Writing the model as a linear combination of the w_t's is going to be VERY useful for future work!

- The mean and autocovariance function for an AR(1) stationary model can be found easily by using the above representation.

$$E(x_t) = E(\sum_{j=0}^{\infty} \phi^j w_{t-j}) = \sum_{j=0}^{\infty} \phi^j E(w_{t-j}) = 0$$

$\gamma(h) = Cov(x_t, x_{t+h}) = E(x_t x_{t+h}) - E(x_t)E(x_{t+h}) = E(x_t x_{t+h}) - 0 = E(x_t x_{t+h})$

, assuming $\mu = 0$

Then

$\gamma(h) = E[(\sum_{j=0}^{\infty} \phi^j w_{t-j})(\sum_{k=0}^{\infty} \phi^k w_{t+h-k})] = E[(w_t + \phi w_{t-1} + \phi^2 w_{t-2} + ...)(w_{t+h} + \phi w_{t+h-1} + \phi^2 w_{t+h-2} + ...)]$

if h=0,

$\gamma(0) = E[(w_t + \phi w_{t-1} + \phi^2 w_{t-2} + ...)^2] = Var(\sum_{j=0}^{\infty} \phi^j w_{t-j}) + [E(\sum_{j=0}^{\infty} \phi^j w_{t-j})]^2 = \sum_{j=0}^{\infty} \phi^{2j} Var(w_{t-j}) + 0 = \sigma_w^2 \sum_{j}$

I used these general results that are taught in a mathematical statistics course:

- $E(Y_1^2) = Var(Y_1) + E(Y_1)^2$
- $Var(a_1 Y_1 + a_2 Y_2) = a_1^2 Var(Y_1) + a_2^2 Var(Y_2)$ for independent random variables Y1 and Y2 and constants a1 and a2.

if h=1,

$\gamma(1) = E[(w_t + \phi w_{t-1} + \phi^2 w_{t-2} + ...)(w_{t+1} + \phi w_t + \phi^2 w_{t-1} + ...)] = E[w_{t+1}(w_t + \phi w_{t-1} + \phi^2 w_{t-2} + ...)] + E[\phi(w_t$

I used w_{t+1} being independent of all of the w's in $E[w_t + \phi w_{t-1} + \phi^2 w_{t-2} + ...]$ in the above result.

In general, for h$\geq$ 0, $\gamma(h) = \frac{\phi^h \sigma_w^2}{1-\phi^2}$

Also, the ACF is $\rho(h) = \frac{\gamma(h)}{\gamma(0)} = \phi^h$.

One can also go back to in the notes and use the results of a linear process there. Below is part of this page restated,

In general, a linear process can be defined as

$$x_t = \mu + \sum_{j=-\infty}^{\infty} \psi_j w_{t-j}$$

with

$$\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$$

and

$$w_t \sim \text{ind.} N(0, \sigma_w^2)$$

It can be shown that $\gamma(h) = \sigma_w^2 \sum_{j=-\infty}^{\infty} \psi_{j+h}\psi_j$ for h $\geq$ 0.

In our case, we have $\psi_0 = 1, \psi_1 = \phi, \psi_2 = \phi^2, \psi_3 = \phi^3, ...$ Therefore, $x_t = \mu + \sum_{j=-\infty}^{\infty} \psi_j w_{t-j} = 0 + \sum_{j=0}^{\infty} \phi^j B^j w_t$. This results in

$$\gamma(h) = \sigma_w^2 \sum_{j=-\infty}^{\infty} \psi_{j+h}\psi_j = \sigma_w^2 \sum_{j=0}^{\infty} \psi_{j+h}\psi_j = \sigma_w^2 \sum_{j=0}^{\infty} \phi^{j+h}\phi^j = \sigma_w^2 \sum_{j=0}^{\infty} \phi^{2j+h} = \sigma_w^2 \phi^h \sum_{j=0}^{\infty} \phi^{2j} = \sigma_w^2 \phi^h \frac{1}{1-\phi^2}$$

Note that $\psi_j = 0$ for j<0

Question: What if $\mu \neq 0$? How would this change $E(x_t)$ and $\gamma(h)$?

The answers are already given in Section 6.2.

Example 9.2. AR(1)with ϕ = 0.7 and ϕ =- 0.7

The purpose of this example is to show what observed values from an AR(1) process look like for t = 1, ..., 100 using $\sigma_w^2 = 100$. Pay close attention to the differences between $\phi = 0.7$ and $\phi = -0.7$. Questions to think about are:

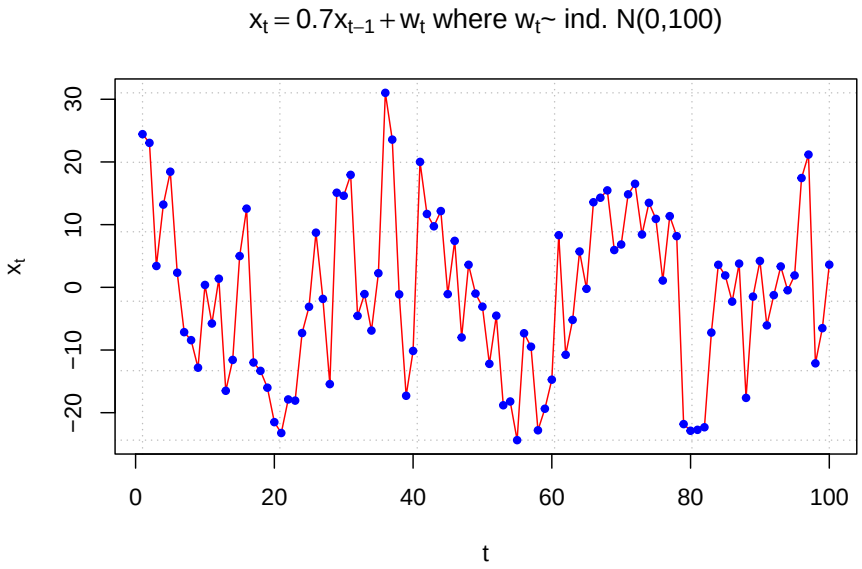
- Why are some plots more or less "choppy" ("jagged")?
- What would happen to the plots if $|\phi|$ was closer to 0 or 1?

Note that $\rho(h) = \gamma(h)/\gamma(0) = \phi^h$, which leads to autocorrelations of

h	$\phi = 0.7$	$\phi = -0.7$
1	0.7	-0.7
2	0.49	0.49
3	0.34	-0.43

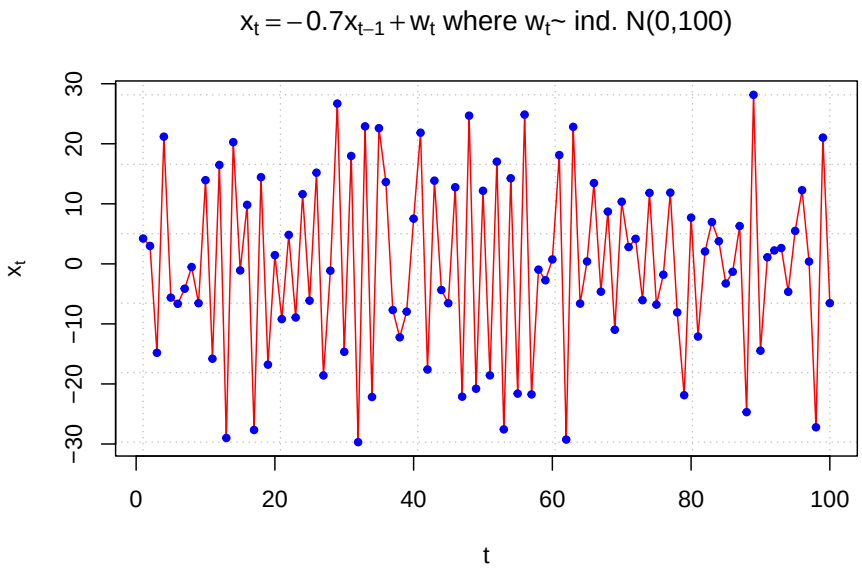
For the plot below, the model is $x_t = 0.7x_{t-1} + w_t$ and $(1–0.7B)x_t = w_t$.

```
set.seed(7181)
x <- arima.sim(model = list(ar = c(0.7)), n = 100, rand.gen
  = rnorm, sd = 10)
plot(x = x, ylab = expression(x[t]), xlab = "t", type =
  "l", col = c("red"), main =
  expression(paste(x[t] == 0.7*x[t-1] + w[t], " where ",
  w[t], "~ ind. N(0,100)")), panel.first=grid(col =
  "gray", lty = "dotted"))
points(x = x, pch = 20, col = "blue")
```



For the plot below, the model is $x_t = -0.7x_{t-1} + w_t$ and $(1–0.7B)x_t = w_t$.

```
set.seed(7181)
x <- arima.sim(model = list(ar = c(-0.7)), n = 100, rand.gen
  = rnorm, sd = 10)
plot(x = x, ylab = expression(x[t]), xlab = "t", type =
  "l", col = c("red"), main =
  expression(paste(x[t] == -0.7*x[t-1] + w[t], " where ",
  w[t], "~ ind. N(0,100)")), panel.first=grid(col =
  "gray", lty = "dotted"))
points(x = x, pch = 20, col = "blue")
```



I could just use ar = 0.7, but included `c()` because it will be needed when we have p > 1.

Be very careful in specifying these models in R!

- The model can be written as $x_t = \phi x_{t-1} + w_t$ and $(1-\phi B)x_t = w_t$. Notice that the first plot uses $\phi = +0.7$ and the second plot uses $\phi = -0.7$. The autoregressive operator can confuse matters in deciding whether the numerical value of ϕ is positive or negative.
- Software packages and textbooks are not consistent in their definitions of the positive and negative values. For example, some books may denote the AR operator to be $(1 + \phi B)$ instead of $(1-\phi B)$.

To simulate data from a higher order AR(p), use the following type of syntax in the model option: `list(ar = c(0.7, -0.4))`

This specifies $\phi_1 = 0.7$ and $\phi_2 = -0.4$.

R has a nice function that allows you to see the ACF for an AR (or ARMA) process better. The `ARMAacf()`allows one to plot the actual ACF for an AR (and ARMA) model.

Example 9.3. AR(1)with ϕ = 0.7 and ϕ =- 0.7

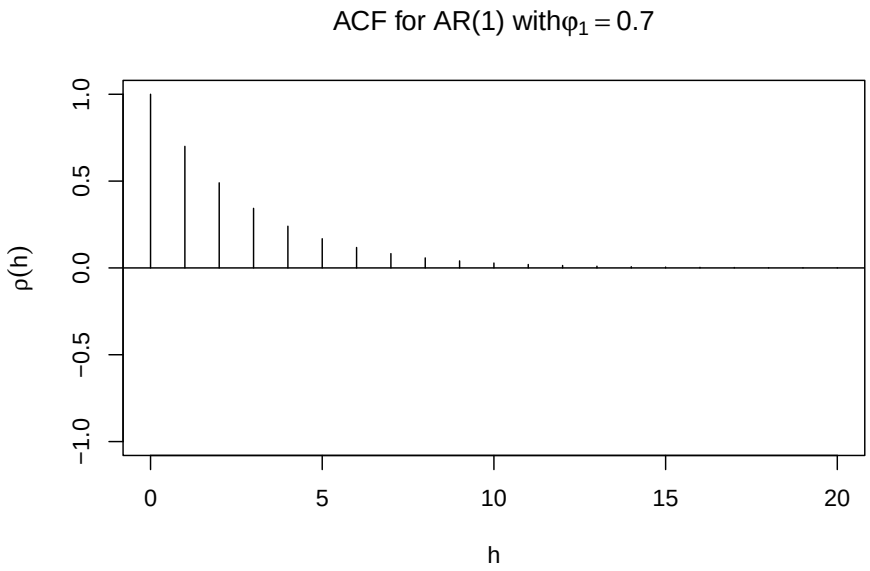
When $\phi_1 = 0.7$, here are the results of the `ARMAacf()` function. Notice the use of `phi1` instead of just `phi` in the main argument of the `plot()` function. Using

`phi` produces ϕ instead of φ. (In our context, it doesn't matter which `phi` you want to write with)

```
round(ARMAacf(ar=c(0.7), lag.max=20) ,4)
```

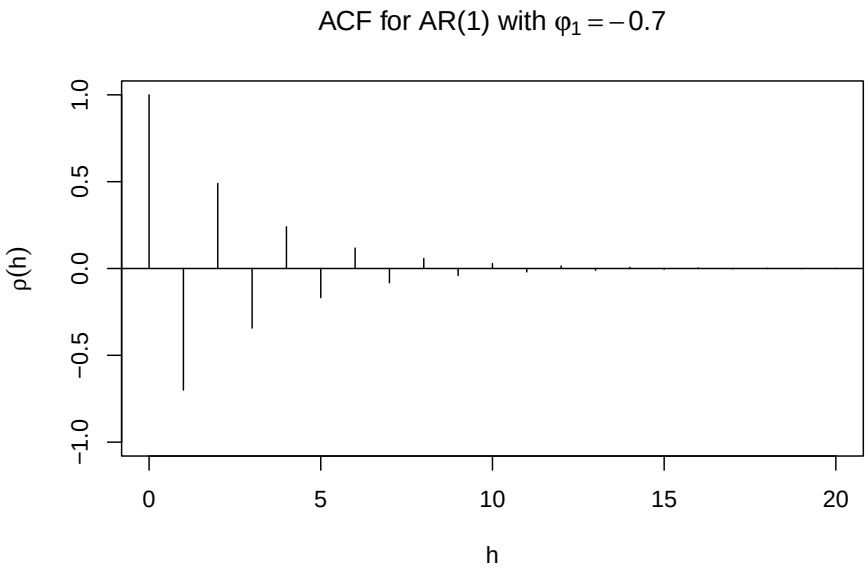
```
##      0      1      2      3      4      5      6      7      8      9     10
## 1.0000 0.7000 0.4900 0.3430 0.2401 0.1681 0.1176 0.0824 0.0576 0.0404 0.0282
##     11     12     13     14     15     16     17     18     19     20
## 0.0198 0.0138 0.0097 0.0068 0.0047 0.0033 0.0023 0.0016 0.0011 0.0008
```

```
plot(y=ARMAacf(ar=c(0.7), lag.max = 20), x=0:20, type="h", ylim=c(-1,1), xlab="h", yla

abline(h=0)
```



Using ϕ_1 = -0.7.

```
plot(y=ARMAacf(ar=c(-0.7), lag.max = 20), x=0:20, type="h", ylim=c(-1,1), xlab="h", yl

abline(h=0)
```



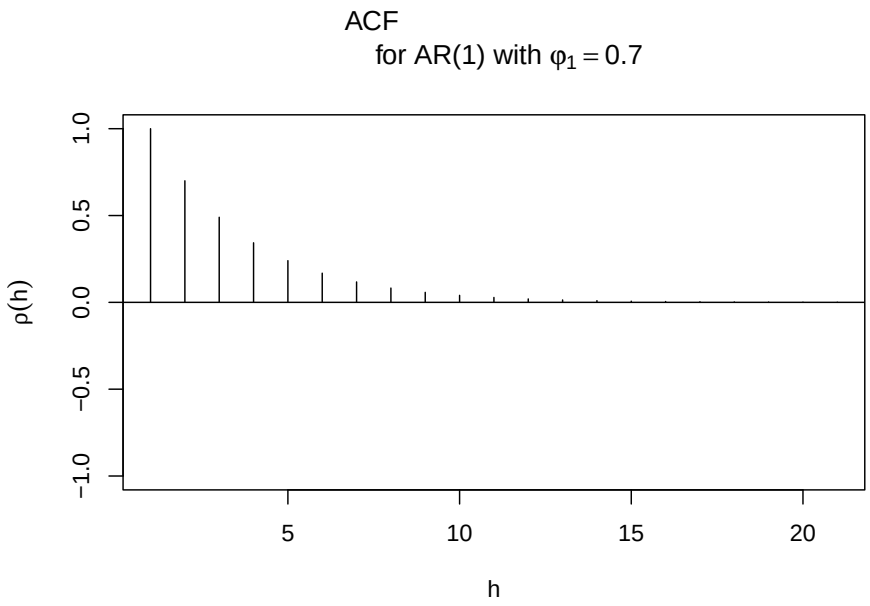
The `type = "h"` argument in the `plot()` tells R to plot vertical lines from a y-axis value of 0 value up to the autocorrelations. (Simply think of it as "height")

Be careful with the plotting of the ACF, suppose the following code was used instead:

```
# wrong graph

plot(x = ARMAacf(ar = c(0.7), lag.max = 20), type =
    "h", ylim = c(-1,1), xlab = "h", ylab =
    expression(rho(h)), main = expression(paste("ACF
    for AR(1) with ", phi1[1] == 0.7)))
```

```
## Warning in title(...): font metrics unknown for character 0xa

## Warning in title(...): font metrics unknown for character 0xa
```

```
abline(h = 0)
```



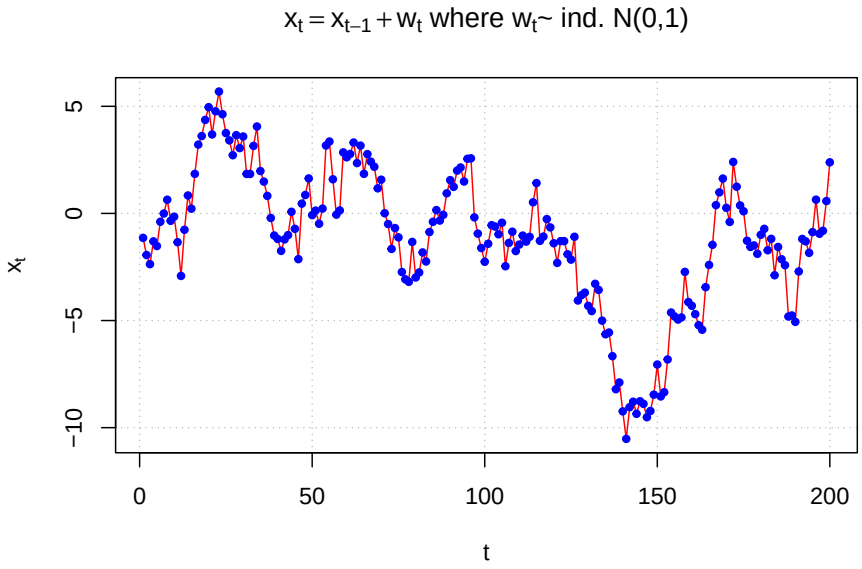
In this case, there is no `y =` part and the plot above is drawn incorrectly with $\rho(1) = 1$, $\rho(2) = 0.7$ (everything is shifted one lag, originally it should be $\rho(0) = 1$, $\rho(1) = 0.7$). The reason why this happens is that `ARMAacf()` gives output that starts at h = 0, but R does not have a way to adjust the x-axis scale to adjust for it.

Example 9.4. AR(1) with $\phi_1 = 1$

For the AR(1) process to be stationary, the following condition must be satisfied: $|\phi_1| < 1$. To see what happens when it is not satisfied, consider a sample generated from an AR(1) process with ϕ_1=1. Below is a plot.

```
set.seed(7181)
w <- rnorm(n = 200, mean = 0, sd = 1)

x <- numeric(length = 200)
x.1 <- 0
for (i in 1:length(x)) {
  x[i] <- x.1  + w[i]
  x.1 <- x[i]
}
plot(x = x, ylab = expression(x[t]), xlab = "t", type =
  "l", col = c("red"), main =
```

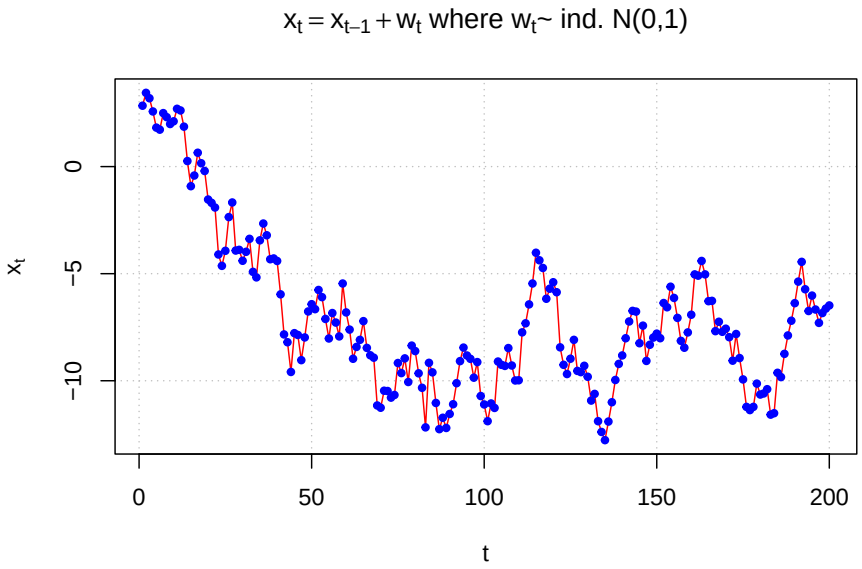
```
  expression(paste(x[t] == x[t-1] + w[t], " where ",
  w[t], "~ ind. N(0,1)" )), panel.first=grid(col =
  "gray", lty = "dotted"))
points(x = x, pch = 20, col = "blue")
```



The process no longer appears to be stationary in the mean.

Below is another set of observations generated from an AR(1) process.

```
set.seed(6234)
w <- rnorm(n = 200, mean = 0, sd = 1)

x <- numeric(length = 200)
x.1 <- 0
for (i in 1:length(x)) {
  x[i] <- x.1  + w[i]
  x.1 <- x[i]
}
plot(x = x, ylab = expression(x[t]), xlab = "t", type =
  "l", col = c("red"), main =
  expression(paste(x[t] == x[t-1] + w[t], " where ",
  w[t], "~ ind. N(0,1)" )), panel.first=grid(col =
  "gray", lty = "dotted"))
points(x = x, pch = 20, col = "blue")
```



This series also appears come from a non-stationary process.

Other examples can be constructed for $|\phi_1| \geq 1$

9.3 Causal Process

Note that the AR(1) process with $|\phi_1| \geq 1$ can be rewritten as a stationary "future-dependent" series (say $x_t = \phi x_{t+1} + w_t$ but this doesn't make any sense in real life). See Shumway and Stoffer's "Explosive AR Models and Causality" example. Because we will not know the "future" values in actual applications, this representation will not help us!

When a process is stationary AND does not depend on the future, the process is "causal". These are the type of processes that we will be interested in.

A more formal definition of a causal process will be given later.

9.4 Writing higher-order casual models as $x_t = \sum_{j=0}^{\infty} \psi_j B^j w_t$

Higher-order models will have constraints on $\phi_1, \phi_2, ..., \phi_p$ to insure a model is causal. We will examine these constraints later. For now, assume we have a causal model.

In a previous example, the AR(1) process was rewritten as

$$(1-\phi B)x_t = w_t \implies x_t = \frac{1}{1-\phi B} w_t \iff x_t = (x+\phi B+\phi^2 B^2+...)w_t = \sum_{j=0}^{\infty} \phi^j B^j w_t = \sum_{j=0}^{\infty} \phi^j w_{t-j}$$

When the process is a higher order AR(p), we can do the same thing! In general, consider the AR(p) process below:

$$(1-\phi_1 B-\phi_2 B^2-...-\phi_p B^p)x_t = w_t \iff \phi(B)x_t = w_t \iff x_t = \frac{1}{\phi(B)} w_t = \psi(B)w_t$$

where $\psi(B) = 1 + B\psi_1 + B^2\psi_2 + ...$, and $1/\phi(B) = \psi(B)$. Be careful with this definition of $\psi(B)$ because there are +, not –, values separating the terms. Also, note that $\psi(B)$ is an infinite series.

Then

$$1 = \phi(B) \times \psi(B) = (1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p B^p)(1 + \psi_1 B + \psi_2 B^2 + ...)$$

To find the values of the ψ_j's in terms of the ϕ_i's, we can equate the coefficients of the B's on both sides of the equality (note the left side has $1 \times B^0 + 0 \times B^1 + 0 \times B^2 + ...$)

$$1 = \phi(B)\times\psi(B) = (1-\phi_1 B-\phi_2 B^2-...-\phi_p B^p)(1+\psi_1 B+\psi_2 B^2+...) \iff 1 = 1-\phi_1 B-\phi_2 B^2-...+\psi_1 B-\psi_1\phi_1 B^2-\psi_1\phi_2$$

Then equating coefficients:

$B^0 : 1 = 1$

$B^1 : \psi_1 - \phi_1 = 0 \implies \phi_1 = \psi_1$

$B^2 : \psi_2 - \psi_1\phi_1 - \phi_2 = 0 \implies \psi_2 = \phi_1^2$

$B^3 : \psi_3 - \psi_2\phi_1 - \psi_1\phi_2 - \phi_3 = 0 \implies \psi_3 = (\phi_1^2 + \phi_2)\phi_1 + \phi_1\phi_2 + \phi_3 = \phi_1^3 + 2\phi_1\phi_2 + \phi_3$

In general, $\psi_j = \phi_j + \sum_{i=1}^{j-1} \phi_i \psi_{j-i}$.

Example 9.5. AR(2) process

For an AR(2) process, this means that

$$x_t = \frac{1}{1-\phi_1 B-\phi_2 B^2} w_t = (1 + \psi_1 B + \psi_2 B^2 + \psi_3 B^3 + ...)w_t$$

Then writing the ψ's in terms of the ϕ's:

$$(1+\psi_1 B+\psi_2 B^2+\psi_3 B^3+...)(1-\phi_1 B-\phi_2 B^2) = 1 \implies 1+\psi_1 B+\psi_2 B^2+\psi_3 B^3+...-\phi_1 B-\phi_1\psi_1 B^2-\phi_1\psi_2 B^3-...-\phi_2 B^2-$$

Equating powers of B implies that

$B^1 = (\psi_1 - \phi_1) = 0 \implies \psi_1 = \phi_1$

$B^2 = (\psi_2 - \phi_1\psi_1 - \phi_2) = 0 \implies \psi_2 = \phi_1\psi_1 + \phi_2 = \phi_1^2 + \phi_2$

$B^3 = (\psi_3 - \phi_1\psi_2 - \phi_2\psi_1) = 0 \implies \psi_3 = \phi_1\psi_2 + \phi_2\psi_1 = \phi_1(\phi_1^2 + \phi_2) + \phi_2\psi_1$

In general, $\psi_j = \psi_{j-1}\phi_1 + \psi_{j-2}\phi_2$ for $j \geq 2$ where $\psi_0 = 1$.

For the special case of $\phi_2 = 0$ (i.e., AR(1)), $\psi_j = \phi_1^j$ and $x_t = \frac{1}{(1-\phi_1 B)} w_t = (1 + \phi_1 B + \phi_1^2 B^2 + ...)w_t$ for $|\phi_1|<1$

R has a nice function that allows you to make this conversion a little easier. The `ARMAtoMA()` converts an ARMA model to a "MA model". For now, one can think of this like how we converted the AR(1) or AR(2) model to a model with an infinite number of w_t random variables.

```
# AR(1) with \phi_1 = 0.7
# psi_j = phi_1^j is satisfied
round(ARMAtoMA(ar = c(0.7), lag.max = 20), 4)
```

```
##  [1] 0.7000 0.4900 0.3430 0.2401 0.1681 0.1176 0.0824 0.0576 0.0404 0.0282
## [11] 0.0198 0.0138 0.0097 0.0068 0.0047 0.0033 0.0023 0.0016 0.0011 0.0008
```

These are the $\psi_1, \psi_2, ..., \psi_{20}$ values when $\phi_1 = 0.7$.

One could also use this function for higher order AR models.For example,

```
#Example for AR(2)
round(ARMAtoMA(ar = c(0.7, -0.4), lag.max = 20), 4)
```

```
##  [1]  0.7000  0.0900 -0.2170 -0.1879 -0.0447  0.0438  0.0486  0.0165 -0.0079
## [10] -0.0121 -0.0053  0.0011  0.0029  0.0016 -0.0001 -0.0007 -0.0005  0.0000
## [19]  0.0001  0.0001
```

Remember that

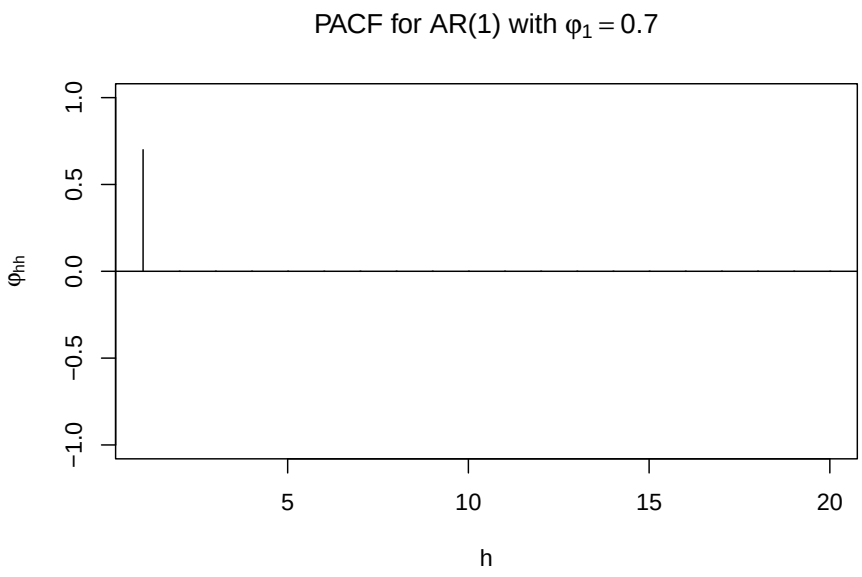
$\psi_1 = \phi_1 = 0.7$

$\psi_2 = \phi_1\psi_1 + \phi_2 = \phi_1^2 + \phi_2 = 0.7^2$–$0.4 = 0.09$.

```
# PACF

  round(ARMAacf(ar = c(0.7), lag.max = 20, pacf = TRUE),4)
```

```
##  [1] 0.7 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
## [20] 0.0
```

```
  plot(x = ARMAacf(ar = c(0.7), lag.max = 20, pacf = TRUE), type = "h", ylim = c(-1,1), xlab = "h
       main = expression(paste("PACF for AR(1) with ", phi1[1] == 0.7)))
  abline(h = 0)
```




```
# Estimated ACF and PACF

  dev.new(width = 8, height = 6, pointsize = 10)
  par(mfrow = c(1,2))

  acf(x = x, type = "correlation", lag.max = 20, ylim = c(-1,1), xlab = "h",
    main = "Estimated ACF")
  abline(h = c(-0.5, 0.5), lty = "dotted", col = "gray")
  pacf(x = x, lag.max = 20, ylim = c(-1,1), xlab = "h",
    main = "Estimated PACF")
  abline(h = c(-0.5, 0.5), lty = "dotted", col = "gray")
```

Chapter 10

Moving Average Models

10.1 Moving Average models-MA(q)

The white noise terms on the right side are linearly combined to form the model

$x_t = w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + ... + \theta_q w_{t-q} = \theta(B) w_t$,where

- $\theta_1, \theta_2, ...\theta_q$ are parameters
- $\theta(B) = (1 + \theta_1 B + \theta_2 B^2 + ... + \theta_q B^q)$
- $w_t \sim \text{ind}(0, \sigma_w^2)$ for t=1,...,n and typically assumed to have a normal distribution.

Notes:

- "+" signs are used in the moving average operator, $\theta(B)$. "-" signs were used in the autoregressive operator, $\phi(B)$. There is no particular reason why these are defined differently (other than what goes on the "right" side of the equality when writing the model out). I chose the way R and Shumway and Stoffer's textbook defines these models. **Many books use "-" signs also in $\theta(B)$, so be careful!** This will cause items like ACFs to be represented differently.
- The autocovariance of a MA(1):

$x_t = w_t + \theta_1 w_{t-1}$, where $w_t \sim \text{ind}(0, \sigma_w^2)$ for t=1,...,n

$\gamma(h) = Cov(x_t, x_{t+h}) = E(x_t x_{t+h}) - E(x_t)E(x_{t+h}) = E(x_t x_{t+h}) - 0 = E(x_t x_{t+h})$

Then

$\gamma(h) = E[(w_t + \theta_1 w_{t-1})(w_{t+h} + \theta_1 w_{t+h-1})] = E[w_t w_{t+h} + \theta_1 w_t w_{t+h-1} + \theta_1 w_{t-1} w_{t+h} + \theta_1^2 w_{t-1} w_{t+h-1}] = E(w_t w_{t+h}) + \theta_1 E($

For h=0:

$E[w_t^2]+\theta_1 E[w_t w_{t-1}]+\theta_1 E[w_{t-1}w_t]+\theta_1^2 E[w_{t-1}^2] = Var(w_t)+[E(w_t)]^2+2\theta_1 E[w_t]E[w_{t-1}]+\theta_1^2 Var(wt-1)+\theta_1^2$

For h=1:

$E[w_t w_{t+1}]+\theta_1 E[w_t^2]+\theta_1 E[w_{t-1}w_{t+1}]+\theta_1^2 E[w_{t-1}w_t] = E[w_t]E[w_{t+1}]+\theta_1[Var(w_t)+[E(w_t)]^2]+\theta_1 E[w_{t-1}]E[$

For h=2:

$$E[w_t w_{t+2}] + \theta_1 E[w_t w_{t+1}] + \theta_1 E[w_{t-1}w_{t+2}] + \theta_1^2 E[w_{t-1}w_{t+1}] = 0$$

For h>2: $\gamma(h) = 0$

Hence,

$$\gamma(h) = \begin{cases} (1+\theta_1^2)\sigma_w^2 & \text{if } h = 0 \\ \theta_1 \sigma_w^2 & \text{if } h = 1 \\ 0 & \text{if } h > 1 \end{cases}$$

- The autocorrelation function of a MA(1):

$$\rho(h) = \begin{cases} 1 & \text{if } h = 0 \\ \frac{\theta_1 \sigma_w^2}{(1+\theta_1^2)\sigma_w^2} = \frac{\theta_1}{1+\theta_1^2} & \text{if } h = 1 \\ 0 & \text{if } h > 1 \end{cases}$$

Notice the autocorrelation is 0 for h > 1!!! This is very helpful when it comes to model selection!!!

- We have already seen an "infinite order" (q = ∞) moving average process as defined in the form of a linear process. This came up in our AR(p) examples earlier. Again, a linear process can be defined as

$$x_t = \mu + \sum_{j=-\infty}^{\infty} \psi_j w_{t-j}, \ \sum_{j=-\infty}^{\infty} |\psi_j| < \infty, w_t \sim ind.N(0, \sigma_w^2)$$

where it can be shown that $\gamma(h) = \sigma_w^2 \sum_{-\infty}^{\infty} \psi_{j+h}\psi_j$ for h $\geq$ 0.

For a MA(1), $\psi_0 = 1$ and $\psi_1 = \theta_1$ and the remaining ψ_j's equal to 0. This results in

$$x_t = \mu + \sum_{j=-\infty}^{\infty} \psi_j w_{t-j} = 0 + \psi_0 w_t + \psi_1 w_{t-1} = w_t + \theta_1 w_{t-1}$$

and $\gamma(0) = \sigma_w^2 \sum_{j=-\infty}^{\infty} \psi_{j+0}\psi_j = \sigma_w^2(\psi_0^2 + \psi_1^2) = \sigma_w^2(1+\theta_1^2)$,

$\gamma(1) = \sigma_w^2 \sum_{j=-\infty}^{\infty} \psi_{j+1}\psi_j = \sigma_w^2(\psi_1 + \psi_0) = \sigma_w^2\theta_1$,

$\gamma(h) = \sigma_w^2 \sum_{j=-\infty}^{\infty} \psi_{j+h}\psi_j = \sigma_w^2(\psi_{0+h} + \psi_0) = \sigma_w^2(\psi_{0+h} + \psi_0) = 0$ for h $\geq$ 2.

Example 10.1. MA(1) with θ_1=0.7 and -0.7

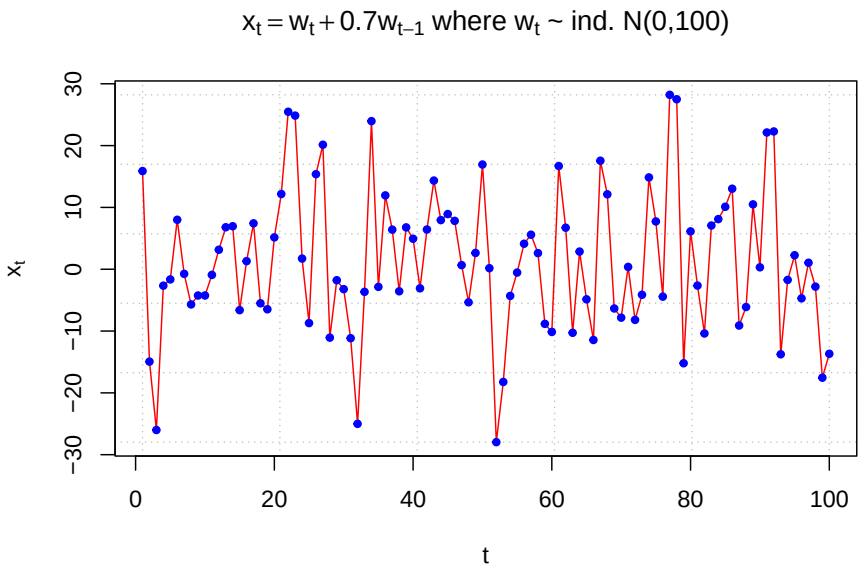
The purpose of this example is to show what observed values from a MA(1) process look like for t = 1, ..., 100. Pay close attention to the differences between $\theta_1 = 0.7$ and θ_1 = -0.7. Questions to think about are:

- Why are some plot plots more or less "choppy" ("jagged")?
- What would happen to the plots if $|\theta_1|$ was closer to 0 or 1?

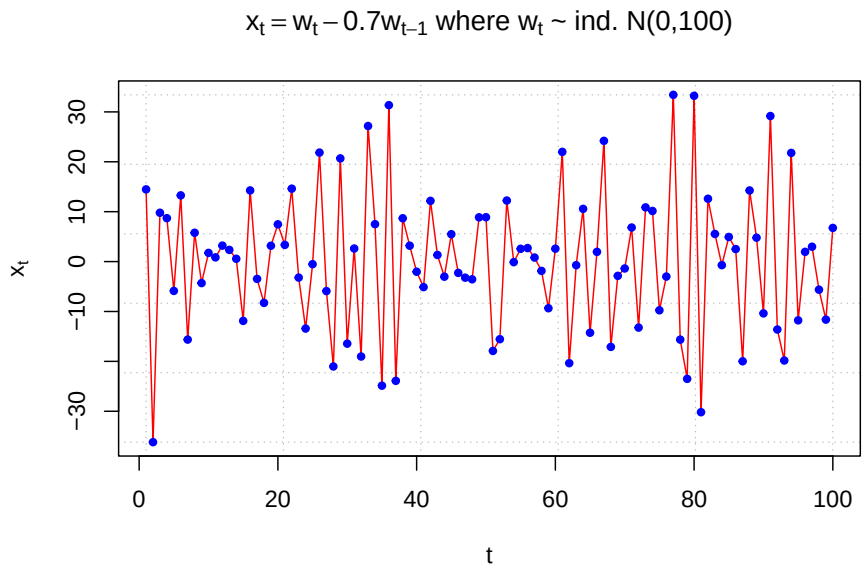
The autocorrelations are

h	$\theta_1 = 0.7$	$\theta_1 = -0.7$
1	0.47	-0.47
2	0	0
3	0	0

```
set.seed(8199)
x <- arima.sim(model=list(ma=c(0.7)), n=100, rand.gen = rnorm, sd=10)

plot(x = x, ylab = expression(x[t]), xlab = "t", type =
    "l", col = c("red"), main =
    expression(paste(x[t] == w[t] + 0.7*w[t-1], " where ",
    w[t], " ~ ind. N(0,100)")) , panel.first=grid(col =
    "gray", lty = "dotted"))

points(x = x, pch = 20, col = "blue")
```




```
set.seed(8199)
x <- arima.sim(model=list(ma=c(-0.7)), n=100, rand.gen = rnorm, sd=10)

plot(x = x, ylab = expression(x[t]), xlab = "t", type =
    "l", col = c("red"), main =
    expression(paste(x[t] == w[t] -0.7*w[t-1], " where ",
    w[t], " ~ ind. N(0,100)")) , panel.first=grid(col =
    "gray", lty = "dotted"))

points(x = x, pch = 20, col = "blue")
```




```
round(ARMAacf(ma = c(0.7), lag.max = 20),4)
```

```
##      0      1      2      3      4      5      6      7      8      9     10
## 1.0000 0.4698 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##     11     12     13     14     15     16     17     18     19     20
## 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

```
# autocorrelation is 0 for h > 1
par(mfrow=c(1,2))

plot(y = ARMAacf(ma = c(0.7), lag.max = 20), x = 0:20,
     type = "h", ylim = c(-1,1), xlab = "h", ylab =
     expression(rho(h)), main = expression(paste("ACF for
     MA(1) with ", theta[1] == 0.7)))
```

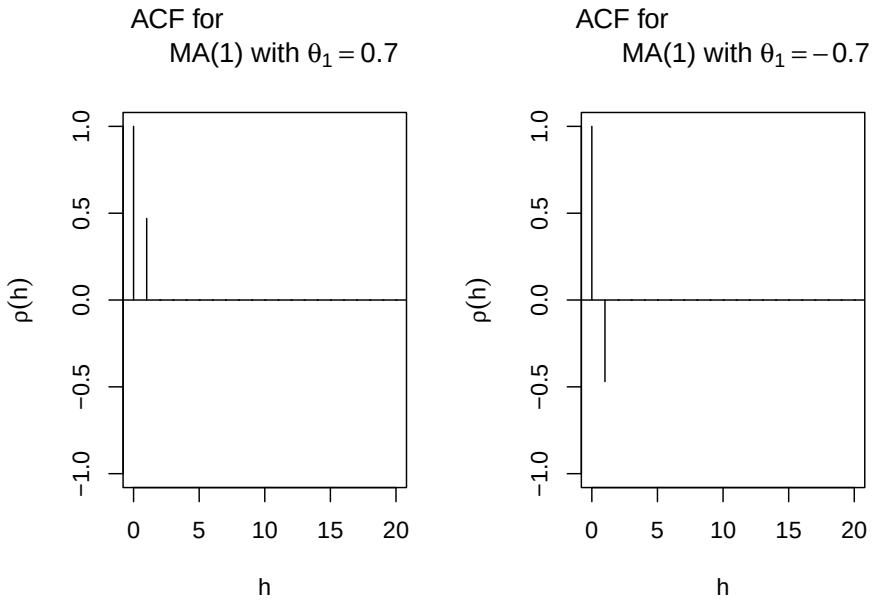
```
## Warning in title(...): font metrics unknown for character 0xa

## Warning in title(...): font metrics unknown for character 0xa
```

```
abline(h=0)

plot(y = ARMAacf(ma = c(-0.7), lag.max = 20), x =
      0:20, type = "h", ylim = c(-1,1), xlab = "h", ylab =
```

```
      expression(rho(h)), main = expression(paste("ACF for
      MA(1) with ", theta[1] == -0.7)))
```

```
## Warning in title(...): font metrics unknown for character 0xa

## Warning in title(...): font metrics unknown for character 0xa
```

```
abline(h = 0)
```



Whenever you see a plot of ACF like this, you should consider MA process for your model selection!!!

Using `ARMAtoMA()` produces the obvious result.

```
round(ARMAtoMA(ma = c(0.7), lag.max = 5),4)
```

```
## [1] 0.7 0.0 0.0 0.0 0.0
```

```
#Example for MA(2)
round(ARMAtoMA(ma = c(0.7, -0.4), lag.max = 5),4)
```

```
## [1]  0.7 -0.4  0.0  0.0  0.0
```

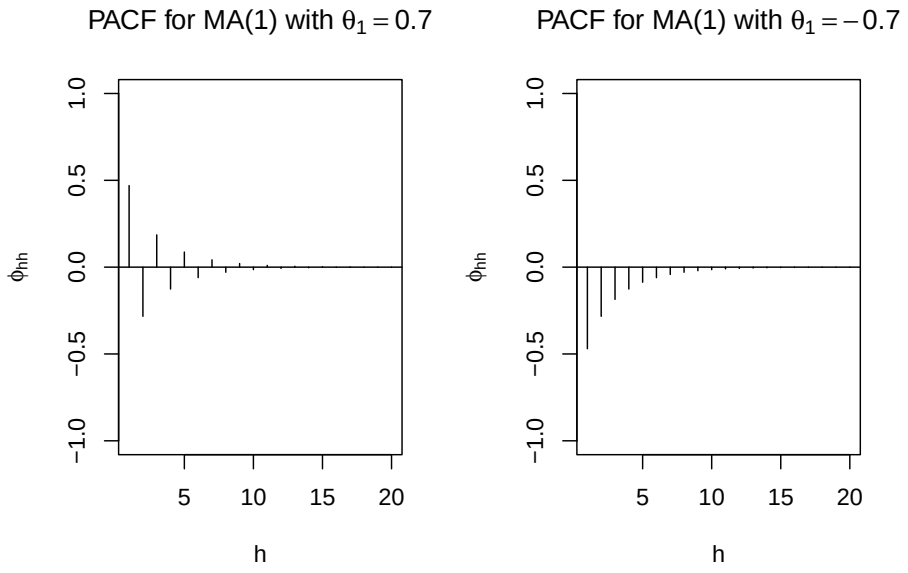
```
# PACF

par(mfrow = c(1,2))
round(ARMAacf(ma = c(0.7), lag.max = 20, pacf = TRUE),4)
```

```
##  [1]  0.4698 -0.2832  0.1856 -0.1260  0.0869 -0.0604  0.0421 -0.0294  0.0206
## [10] -0.0144  0.0101 -0.0071  0.0049 -0.0035  0.0024 -0.0017  0.0012 -0.0008
## [19]  0.0006 -0.0004
```

```
plot(x = ARMAacf(ma = c(0.7), lag.max = 20, pacf = TRUE), type = "h", ylim = c(-1,1), xlab = "h",
       main = expression(paste("PACF for MA(1) with ", theta[1] == 0.7)))
abline(h = 0)

round(ARMAacf(ma = c(-0.7), lag.max = 20, pacf = TRUE),4)
```

```
##  [1] -0.4698 -0.2832 -0.1856 -0.1260 -0.0869 -0.0604 -0.0421 -0.0294 -0.0206
## [10] -0.0144 -0.0101 -0.0071 -0.0049 -0.0035 -0.0024 -0.0017 -0.0012 -0.0008
## [19] -0.0006 -0.0004
```

```
plot(x = ARMAacf(ma = c(-0.7), lag.max = 20, pacf = TRUE), type = "h", ylim = c(-1,1), xlab = "h"
       main = expression(paste("PACF for MA(1) with ", theta[1] == -0.7)))
abline(h = 0)
```



10.2 Invertible process

A MA(q) process can be represented as an infinite order AR process. Thus,

$$\sum_{j=0}^{\infty} \pi_j B^j x_t = \pi(B) x_t = w_t$$

, where $\frac{1}{\theta(B)} = \pi(B) = 1 + \pi_1 B + \pi_2 B^2 + ...$ This is "similar" to how an AR(p) process can be represented as an infinite order MA process(Recall that to convert AR(p) process to infinite order MA process, we need causal models).

Why bothers? Well, one motivation for converting MA process to AR process is that a MA process can have the same autocorrelation for multiple values of θ and σ_w^2. For example, consider the models

$$x_t = w_t + 0.1 w_{t-1}$$

and

$$x_t = w_t + 10 w_{t-1}$$

The autocorrelation at lag 1 is $\rho(1) = 0.09 (= \frac{\theta_1}{1+\theta_1^2})$ for both models. In fact, this occurs for any MA(1) model using θ_1 and $1/\theta_1$ as the coefficient on w_{t-1}.

The model that is chosen to represent a series is the one that has an infinite order AR representation. This chosen process is called an "invertible" process.

A more formal definition of an invertible process will be given later (similar to causal process requirements).

Chapter 11

ARMA

11.1 Autoregressive Moving Average(ARMA)

Combines MA and AR models into one model. An "ARMA(p,q)" model is:

$$(1-\phi_1 B-\phi_2 B^2-...-\phi_p B^p)x_t = (1+\theta_1 B+\theta_2 B^2+...+\theta_q B^q)w_t \iff \phi(B)x_t = \theta(B)w_t$$

assuming x_t is stationary.
Notes:

- If $E(x_t) = \mu \neq 0$, then we could rewrite the model as

$$\phi(B)x_t = \alpha + \theta(B)w_t$$

where $\alpha = \mu(1 - \phi_1 - \phi_2 - ... - \phi_p)$
or let $z_t = x_t - \mu$ so that the model is $\phi(B)z_t = \theta(B)w_t$

- If the model is casual, then it can be rewritten as

$$x_t = \frac{\theta(B)}{\phi(B)}w_t = \psi(B)w_t$$

where $\psi(B) = 1 + B\psi_1 + B^2\psi_2 + ...$

- If the model is invertible, then it can be rewritten as

$$\frac{\phi(B)}{\theta(B)}x_t = \pi(B)x_t = w_t$$

, where $\pi(B) = 1 + \pi_1 B + \pi_2 B^2 + ...$

- Problem with parameter redundancy

ARMA(1,1): $(1 - 0.5B)x_t = (1 - 0.5B)w_t$

Notice that $x_t = \frac{1-0.5B}{1-0.5B}w_t = w_t$! Thus, this ARMA(1,1) can also be represented by a white noise process.

Another example illustrating this is the following ARMA(2,1):

$$(1-0.1B-0.2B^2)x_t = (1-0.5B)w_t \iff (1+0.4B)(1-0.5B)x_t = (1-0.5B)w_t \iff (1+0.4B)x_t = \frac{1-0.5}{1-0.5}$$

Thus, this ARMA(2,1) can also be represented by an ARMA(1,0).

The problem here is that both the AR and MA parts have a "common factor". To avoid this problem, we need to assume that AR and MA operators have no common factors.

To overcome the problems with stationary AR models that depend on the future and MA models that are not unique, we require ARMA models to be causal and invertible.

Let $\phi(z) = 1 - \phi_1 z - \phi_2 z^2 - ... - \phi_p z^p, \phi_p \neq 0$, and $\theta(z) = 1 + \theta_1 z + \theta_2 z^2 + ... + \theta_q z^q, \theta_q \neq 0$ be the AR and MA operators with "z" used in it to emphasize they are as polynomials and z is an ordinary algebraic variable. Note that z is possibly a complex number.

Assume that $\phi_(z)$ and $\theta(z)$ have no common factors.

11.2 Causality of an ARMA(p,q) process

An ARMA(p,q) model is causal when the roots of $\phi(z)$ lie outside the unit circle; that is, $\phi(z) = 0$ only when $|z| > 1$. The coefficients of the linear process for $x_t = \sum_{j=0}^{\infty} \psi_j B^j w_t$ can be determined from solving $\sum_{j=0}^{\infty} \psi_j z^j = \frac{\theta(z)}{\phi(z)}$.

Note that when z is a real number, |z| means absolute value of z. When z is a complex number c+di, |z| means $\sqrt{c^2 + d^2}$ where i = $\sqrt{-1}$.

Again, this means the model can be rewritten as an infinite order MA model.

Reason: Stationarity that is not future dependent

The appendix of Shumway and Stoffer's textbook includes a proof.

11.3 Invertibility of ARMA(p,q) process

An ARMA(p,q) model is invertible when the roots of $\theta(z)$ lie outside the unit circle; that is, $\theta(z) = 0$ only when |z|>1. The coefficients of the linear process for $\sum_{j=0}^{\infty} \pi_j B^j x_t = w_t$ can be determined from solving $\sum_{j=0}^{\infty} \pi_j z^j = \frac{\phi(z)}{\theta(z)}$.

Again, this means the model can be rewritten as an infinite order AR model.

Reason: Model has a unique representation

The appendix of Shumway and Stoffer's textbook includes a proof.

Example 11.1. AR(1), $(1 - \phi B)x_t = w_t$

Causal: Let $\phi(z) = (1 - \phi z)$. Note that $(1 - \phi z) = 0$ has a root of $z = \frac{1}{\phi}$. For this to be causal,

$|\frac{1}{\phi}| > 1 \iff -1 < \phi < 1$

Therefore, an AR(1) model is causal if -1 < ϕ < 1. Relate this back to earlier in the notes when we used the sum of an infinite series to find the infinite MA representation.

Invertible: Yes, because the model is already written in terms of an AR only model.

What happens if the causal conditions are violated? The model will not be causal and thus will not be "backward" stationary.

Suppose $\phi_1 = 2$ in an AR(1). Then $x_t = \phi_1 x_{t-1} + w_t = 2x_{t-1} + w_t$. Using a program similar to ar1.R in Chapter 1 (with $w_t \sim ind.N(0, 1)$ and the `for()` function), the data generated from this series produces

```
set.seed(1381)
w <- rnorm(n = 200, mean = 0, sd = 1)



x <- numeric(length = 200)
x.1 <- 0
for(i in 1:length(x)) {
    x[i] <- 2*x.1 + w[i]
    x.1 <- x[i]
}

head(data.frame(x, w))
```

```
##              x          w
## 1  0.441380065  0.4413801
```

```
## 2 -0.006146795 -0.8889069
## 3 -0.185414839 -0.1731212
## 4 -1.668871234 -1.2980416
## 5 -4.272024597 -0.9342821
## 6 -6.358892543  2.1851567
```

Notice that x_t is changing rapidly (not stationary in the mean).

Note: All AR only models are invertible!

Example 11.2. AR(2)

$(1 - \phi_1 B - \phi_2 B^2)x_t = w_t$

Invertible: Yes, because the model is already written in terms of an AR only model.

Causal: $(1 - \phi_1 z - \phi_2 z^2) = 0$ needs to have roots z_1 and z_2 outside of the unit circle.

R has a function called `polyroot()` that can find the roots of a polynomial automatically. Below are some examples of working with it.

```
#Syntax for a AR(2): 1 - phi1*z - phi2*z^2
#  phi1 = 0.5, phi2 = 0
polyroot(z = c(1, -0.5, 0))   # 1-0.5z
```

```
## [1] 2+0i
```

```
#Outside unit circle examples - causal
#phi1 = 0, phi2 = 0.5
polyroot(z=c(1,0,-0.5)) # 1-0.5z^2
```

```
## [1]  1.414214-0i -1.414214+0i
```

```
#phi1 = 0, phi2 = -0.5
polyroot(z = c(1, 0, 0.5))   # 1+0.5z^2
```

```
## [1] 0+1.414214i 0-1.414214i
```

```
#phi1 = -0.2, phi2 = -0.5
polyroot(z = c(1, 0.2, 0.5))  # 1+0.2z+0.5z^2
```

```
## [1] -0.2+1.4i -0.2-1.4i
```

```r
abs(polyroot(z = c(1, 0.2, 0.5))) #Check if outside
```

```
## [1] 1.414214 1.414214
```

```r
sqrt((-0.2)^2+1.4^2) #Verify abs() function did it correctly
```

```
## [1] 1.414214
```

```r
#phi1 = -1, phi2 = -0.5
polyroot(z = c(1, 1, 0.5))
```

```
## [1] -1+1i -1-1i
```

```r
abs(polyroot(z = c(1, 1, 0.5)))
```

```
## [1] 1.414214 1.414214
```

```r
#phi1 = -1.8, phi2 = -0.9
polyroot(z = c(1, 1.8, 0.9))
```

```
## [1] -1+0.333333i -1-0.333333i
```

```r
abs(polyroot(z = c(1, 1.8, 0.9)))
```

```
## [1] 1.054093 1.054093
```

```r
#phi1 = 0.5, phi2 = 0.25
polyroot(z = c(1, -0.5, -0.25))
```

```
## [1]  1.236068+0i -3.236068-0i
```

```r
abs(polyroot(z = c(1, -0.5, -0.25)))
```

```
## [1] 1.236068 3.236068
```

```r
# Inside unit circle examples - not causal
#phi1 = 1.8, phi2 = 0.9
polyroot(z = c(1, -1.8, -0.9))
```

```
## [1]  0.4529663+0i -2.4529663-0i
```

```
abs(polyroot(z = c(1, -1.8, -0.9)))
```

```
## [1] 0.4529663 2.4529663
```

```
#phi1 = -1.2, phi2 = 0.8
polyroot(z = c(1, 1.2, -0.8))
```

```
## [1] -0.5962912+0i  2.0962912+0i
```

```
abs(polyroot(z = c(1, 1.2, -0.8)))
```

```
## [1] 0.5962912 2.0962912
```

It can be shown that the conditions are -1 $< \phi_2 <$ 1, $\phi_2 + \phi_1 < 1$, and $\phi_2 - \phi_1 <$ 1. Shumway and Stoffer and Wei's textbooks provides proofs via the quadratic formula.

Below is a plot of the region from the Box, Jenkins, and Reinsel textbook. The ACF plots (labeled ρ_k) inside the triangle show what ACFs would look like. The other plots inside of the triangle are plots of the "partial" autocorrelation function, which will be discussed later in the course.

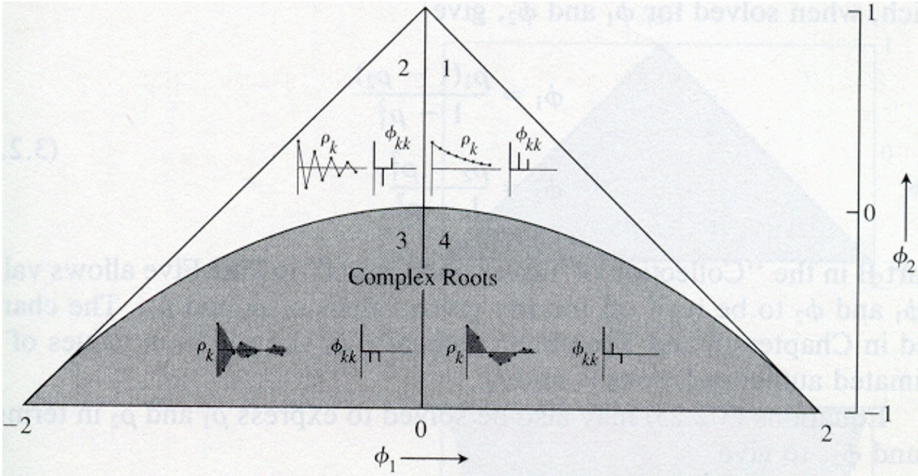
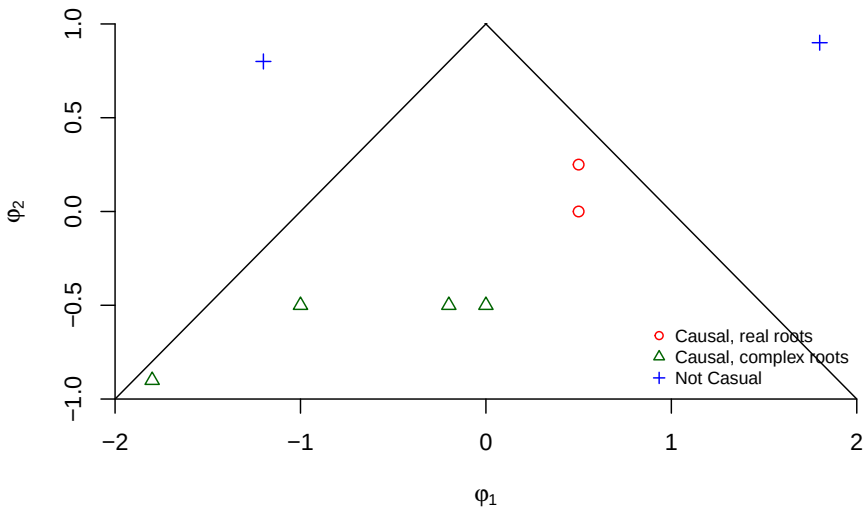


Figure 3.2 Typical autocorrelation and partial autocorrelation functions ρ_k and ϕ_{kk} for various stationary AR(2) models. (From [183].)

Figure 11.1: graph1

```
#By default, R goes 4% more on y and x-axis limits,
#  These par() options make it stop at specified limits
par(xaxs = "i", yaxs = "i")

#dummy plot
plot(x = -2, y = -1, xlim = c(-2,2), ylim = c(-1,1),
    type = "n", frame.plot = FALSE, xlab =
     expression(phi1[1]), ylab = expression(phi1[2]))

#abline() draws y = a + bx
abline(a = 1, b = -1) #Draw line of phi2 = 1 - phi1
abline(a = 1, b =  1) #Draw line of phi2 = 1 + phi1

#Plot the phi1 and phi2 values
points(x = 0.5, y = 0, pch = 1, col = "red")
points(x = 0, y = -0.5, pch = 2, col = "darkgreen")
points(x = -0.2, y = -0.5, pch = 2, col = "darkgreen")
points(x = -1, y = -0.5, pch = 2, col = "darkgreen")
points(x = -1.8, y = -0.9, pch = 2, col = "darkgreen")
points(x = 0.5, y = 0.25, pch = 1, col = "red")

points(x = 1.8, y = 0.9, pch = 3, col = "blue")
points(x = -1.2, y = 0.8, pch = 3, col = "blue")

legend("bottomright", legend = c("Causal, real roots",
      "Causal, complex roots", "Not Casual"), pch =
      c(1,2,3), col = c("red", "darkgreen", "blue"), cex =
      0.75, bty = "n")
```



Example 11.3. MA(1)

$x_t = (1 + \theta B)w_t$

Causal: Yes, because the model is already written in terms of an MA only model.

Invertible: Let $\theta(z) = (1 + \theta z)$. Note that $(1 + \theta z) = 0$ has a root of z $= -\frac{1}{\theta}$. For this to be invertible,

$|\frac{-1}{\theta}| > 1 \iff -1 < \theta < 1.$

Therefore, a MA(1) model is invertible if $-1 < \theta < 1$

Notes:

- MA only model will always be causal.
- MA(2) model has similar invertibility conditions as the AR(2) model's causal conditions.

- ARMA(1,1) model has the same causal conditions as an AR(1) and the same invertible conditions as a MA(1).

- ARMA(2,2) model has the same causal conditions as an AR(2) and the same invertible conditions as a MA(2)…

Chapter 12

ACF and PACF for ARMA models

We will examine the ACF and PACF (a new function) for a specific ARMA process. We can compare these values to their corresponding estimated values from an observed time series. Values and patterns that tend to match the best between the ARMA model and the observed time series suggest a particular model to use for the data.

12.1 ACF for MA(q)

Below is the derivation

$$x_t = \theta(B)w_t = (1 + \theta_1 B + \theta_2 B^2 + ... + \theta_q B^q) = \sum_{j=0}^{q} \theta_j B^j w_t$$

with $\theta_0 = 1$ and $w_t \sim ind(0, \sigma_w^2)$

Note that $\gamma(h) = Cov(x_t, x_{t+h}) = E(x_t x_{t+h}) - E(x_t)E(x_{t+h}) = E(x_t x_{t+h})$ since $E(x_t) = E(x_{t+h}) = 0$

When $0 \leq$ h $\leq$ q,

$$E(x_t x_{t+h}) = E[(\sum_{i=0}^{q} \theta_i B^i w_t)(\sum_{j=0}^{q} \theta_j B^j w_{t+h})] = E[\sum_{i=0}^{q}\sum_{j=0}^{q} \theta_i\theta_j w_{t-i} w_{t+h-j}] = \sum_{i=0}^{q}\sum_{j=0}^{q} \theta_i\theta_j E[w_{t-i} w_{t+h-j}] = \sigma_w^2 \sum_{i=0}^{q-h} \theta_i\theta_{i+h}$$

The last term comes about because $E(w_{t-i}w_{t+h-j}) = 0$ when -i $\neq$ h-j and $E(w_{t-i}w_{t+h-j}) = E(w_{t-i}^2) = \sigma_w^2$ when -i = h-j $\implies$ j = i+h. One can write out

the indices for the summation and then try h = 0, h = 1, … to see there are q - h pairs of indices that match up each time.

Therefore,

$$\gamma(h) = \begin{cases} \sigma_w^2 \sum_{i=0}^{q-h} \theta_i \theta_{i+h} & \text{if } 0 \leq h \leq q \\ 0 & \text{if } h > q \end{cases}$$

Note that $\gamma(0) = \sigma_w^2 \sum_{i=0}^{q} \theta_i^2$. The ACF is

$$\rho(h) = \frac{\gamma(h)}{\gamma(0)} = \begin{cases} \frac{\sum_{i=0}^{q-h} \theta_i \theta_{i+h}}{\sum_{i=0}^{q} \theta_i^2} & \text{if } 0 \leq h \leq q \\ 0 & \text{if } h > q \end{cases}$$

Notes:

- This derivation could have been directly from the linear process result examined earlier; where it can be shown that $\gamma(h) = \sigma_w^2 \sum_{-\infty}^{\infty} \psi_{j+h} \psi_j$ for h ≥ 0.
- For a MA(1): $\gamma(h) = \sigma_w^2 \sum_{i=0}^{1-h} \theta_i \theta_{i-h}$ for h = 0, 1. Then
  - $\gamma(0) = \sigma_w^2(\theta_0^2 + \theta_1^2) = \sigma_w^2(1 + \theta_1^2)$ where $\theta_0 = 1$,
  - $\gamma(1) = \sigma_w^2(\theta_0 \theta_1) = \sigma_w^2 \theta_1$, and
  - $\gamma(h) = 0$ for h > 1.
  - The ACF is $\rho(0) = 1, \rho(1) = \frac{\theta_1}{1+\theta_1^2}$, and $\rho(h) = 0$ for h > 1.
  - Plots from ma1_sim.R using MA(1):

```
set.seed(8199)
x <- arima.sim(model = list(ma = c(0.7)), n = 100, rand.gen = rnorm, sd = 10)

dev.new(width = 8, height = 6, pointsize = 10)  #Opens up wider plot window than the d
plot(x = x, ylab = expression(x[t]), xlab = "t", type = "l", col = c("red"),
     main = expression(paste(x[t] == w[t] + 0.7*w[t-1], " where ", w[t],
                             " ~ ind.N(0,100)")),
     panel.first=grid(col="gray",lty="dotted"))

points(x = x, pch = 20, col = "blue")
```

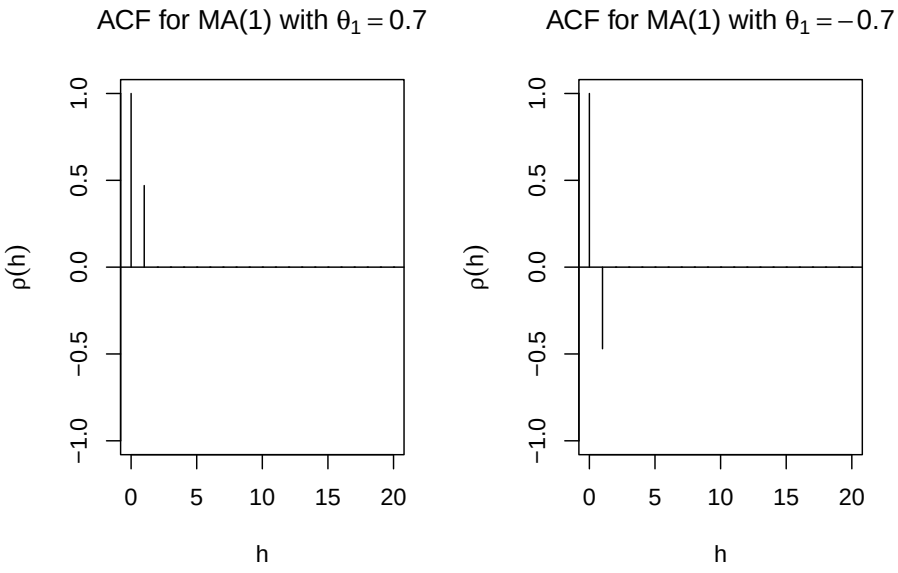
```
round(ARMAacf(ma = c(0.7), lag.max = 20),4)
```

```
##      0      1      2      3      4      5      6      7      8      9     10
## 1.0000 0.4698 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##     11     12     13     14     15     16     17     18     19     20
## 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

```
round(ARMAacf(ma = c(-0.7), lag.max = 20),4)
```

```
##       0       1       2       3       4       5       6       7       8       9
##  1.0000 -0.4698  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
##      10      11      12      13      14      15      16      17      18      19
##  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
##      20
##  0.0000
```

```
par(mfrow = c(1,2))


plot(y = ARMAacf(ma = c(0.7), lag.max = 20), x = 0:20, type = "h", ylim = c(-1,1), xlab = "h", yl
       main = expression(paste("ACF for MA(1) with ", theta[1] == 0.7)))
abline(h = 0)


plot(y = ARMAacf(ma = c(-0.7), lag.max = 20), x = 0:20, type = "h", ylim = c(-1,1), xlab = "h", y
       main = expression(paste("ACF for MA(1) with ", theta[1] == -0.7)))
abline(h = 0)
```



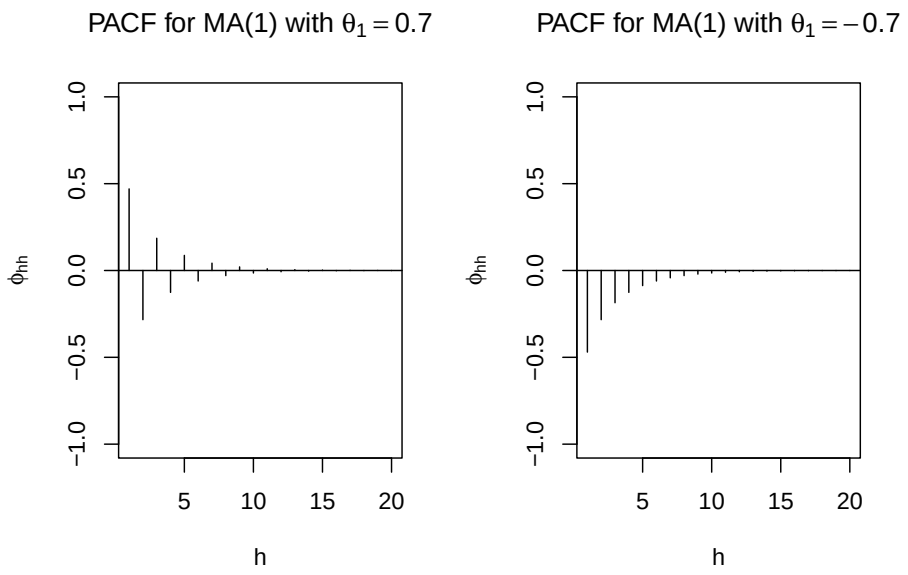
```r
# PACF
round(ARMAacf(ma = c(0.7), lag.max = 20, pacf = TRUE),4)
```

```
##  [1]  0.4698 -0.2832  0.1856 -0.1260  0.0869 -0.0604  0.0421 -0.0294  0.0206
## [10] -0.0144  0.0101 -0.0071  0.0049 -0.0035  0.0024 -0.0017  0.0012 -0.0008
## [19]  0.0006 -0.0004
```

```r
round(ARMAacf(ma = c(-0.7), lag.max = 20, pacf = TRUE),4)
```

```
##  [1] -0.4698 -0.2832 -0.1856 -0.1260 -0.0869 -0.0604 -0.0421 -0.0294 -0.0206
## [10] -0.0144 -0.0101 -0.0071 -0.0049 -0.0035 -0.0024 -0.0017 -0.0012 -0.0008
## [19] -0.0006 -0.0004
```

```r
# PACF
par(mfrow = c(1,2))

plot(x = ARMAacf(ma = c(0.7), lag.max = 20, pacf = TRUE), type = "h", ylim = c(-1,1), x
       main = expression(paste("PACF for MA(1) with ", theta[1] == 0.7)))
abline(h = 0)


plot(x = ARMAacf(ma = c(-0.7), lag.max = 20, pacf = TRUE), type = "h", ylim = c(-1,1),
       main = expression(paste("PACF for MA(1) with ", theta[1] == -0.7)))

abline(h = 0)
```



- Suppose you observed a time series. The estimated values of the ACF are $\widehat{\rho}(1) \neq 0$ and $\widehat{\rho}(h) \approx 0$ for h > 1. What may be a good model for the data? MA(1)
- For a MA(2):$\gamma(h) = \sigma_w^2 \sum_{i=0}^{2-h} \theta_i \theta_{i+h}$. Then
  - $\gamma(0) = \sigma_w^2(1 + \theta_1^2 + \theta_2^2)$,
  - $\gamma(1) = \sigma_w^2(\theta_1 + \theta_1\theta_2)$,
  - $\gamma(2) = \sigma_w^2\theta_2$, and
  - $\gamma(h) = 0$ for h>2.

  - The ACF is $\rho(0) = 1$, $\rho(1) = \frac{\theta_1+\theta_1\theta_2}{1+\theta_1^2+\theta_2^2}$, $\rho(2) = \frac{\theta_2}{1+\theta_1^2+\theta_2^2}$, and $\rho(h) = 0$ for h > 2.

- For a MA(q): $\rho(h) = 0$ for h $>$ q.

- Suppose you observed a time series. The estimated values of the ACF are $\widehat{\rho}(1) \neq 0$,$\widehat{\rho}(2) \neq 0$, and $\widehat{\rho}(h) \approx 0$ for h > 2. What may be a good model for the data? MA(2)

12.2 ACF for casual ARMA(p,q)

Below is the derivation.

First, rewrite the model as an infinite order MA!

$$\phi(B)x_t = \theta(B)w_t \implies x_t = \frac{\theta(B)}{\phi(B)}w_t \implies x_t = \psi(B)w_t$$

where $w_t \sim ind(0, \sigma_w^2)$ for t = 1, ..., n

$\psi(B) = 1 + B\psi_1 + B^2\psi_2 + ...$

We can re-write the model as

$$x_t = \sum_{i=0}^{\infty} \psi_i B^i w_t$$

with $\psi_0 = 1$

Note that $E(x_t) = 0$.

Then

$$\gamma(h) = Cov(x_t, x_{t+h}) = E(x_t x_{t+h}) = E[(\sum_{i=0}^{\infty} \psi_i B^i w_t)(\sum_{j=0}^{\infty} \psi_j B^j w_{t+h})]E[\sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \psi_i\psi_j w_{t-i}w_{t+h-j}] = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}$$

because $E(w_{t-i}w_{t+h-j}) = 0$ when -i $\neq$ h-j and $E(w_{t-i}w_{t+h-j}) = E(w_{t-i}^2) = \sigma_w^2$ when -i=h-j $\implies$ j=i+h

Look how similar this is to the proof for MA(q) and for a linear process in general!

To find the autocovariance function for any ARMA process, we can transform the model into an infinite order MA and find the ψ's!

Another interesting way to solve for the autocovariance function:

$$x_t = \sum_{j=1}^{p} \phi_j B^j x_t + \sum_{j=0}^{q} \theta_j B^j w_t = \sum_{j=1}^{p} \phi_j x_{t-j} + \sum_{j=0}^{q} \theta_j w_{t-j}$$

where $\phi_0 = 1$ and $\theta_0 = 1$

$$\gamma(h) = Cov(x_{t+h}, x_t) = E[(\sum_{j=1}^{p} \phi_j x_{t+h-j} + \sum_{j=0}^{q} \theta_j w_{t+h-j})x_t]$$

Because

$$E(x_t) = 0, \phi(B)x_t = \theta(B)w_t \iff x_t = \phi_1 x_{t-1} + ... + \phi_p x_{t-p} + w_t + \theta_1 w_{t-1} + ... + \theta_q w_{t-q} \implies x_{t+h} = \sum_{j=1}^{p} \phi_j x$$

Then

$$\gamma(h) = E(\sum_{j=1}^{p} \phi_j x_t x_{t+h-j}) + E(\sum_{j=0}^{q} \theta_j x_t w_{t+h-j}) = \sum_{j=1}^{p} \phi_j E(x_t x_{t+h-j}) + \sum_{j=0}^{q} \theta_j E(x_t w_{t+h-j}) = \sum_{j=1}^{p} \phi_j \gamma(h-j) + \sum_{j=0}^{q} \theta_j E(w_{t+}$$

because $\gamma(h-j) = E(x_t x_{t+h-j}) - E(x_t)E(x_{t+h-j}) = E(x_t x_{t+h-j}) - 0$ and $x_t = \psi(B)w_t$

So

$$\gamma(h) = \sum_{j=1}^{p} \phi_j \gamma(h-j) + \sigma_w^2 \sum_{j=h}^{q} \theta_j \psi_{j-h}$$

for h ≥ 0, because we only need the non-zero element of $E(w_{t+h-j} \sum_{i=0}^{\infty} \psi_i w_{t-i})$ occurs when -i = h-j $\iff$ i = j-h and $\psi_i = 0$ for i < 0

Thus, $\gamma(h) = \sum_{j=1}^{p} \phi_j \gamma(h-j) + \sigma_w^2 \sum_{j=h}^{q} \theta_j \psi_{j-h}$ for $0 \leq h < max(p, q+1)$ and $\gamma(h) = \sum_{j-1}^{p} \phi_j \gamma(h-j)$ for $h \geq max(p, q+1)$

Note: Dividing the above by $\gamma(0)$ produces $\rho(h)$!

Example 12.1. ACF from casual ARMA(1,1)

- $(1 - B\phi_1)x_t = (1 + \theta_1 B)w_t$ where $w_t \sim ind.(0, \sigma_w^2)$ for t = 1, ..., n
- Note that $x_t = \frac{1+\theta_1 B}{1-\phi_1 B} w_t$ can be rewritten as $x_t = \psi(B)w_t$. Then $(1 + \psi_1 B + \psi_2 B^2 + ...)(1 - \phi_1 B) = 1 + \theta_1 B$. By multiplying out, we obtain

$$1 + \psi_1 B + \psi_2 B^2 + ... - \phi_1 B - \phi_1 \psi_1 B^2 - \phi_1 \psi_2 B^3 - ... = 1 + \theta_1 B$$

Equating terms produces

$B : \theta_1 = \psi_1 - \phi_1 \implies \psi_1 = \theta_1 + \phi_1$

$B^2 : 0 = \psi_2 - \phi_1 \psi_1 \implies \psi_2 = \phi_1(\theta_1 + \phi_1)$

$B^3 : 0 = \psi_3 - \phi_1 \psi_2 \implies \psi_3 = \phi_1[\phi_1(\theta_1 + \phi_1)] = \phi_1^2(\theta_1 + \phi_1)$

We can use this result to find the autocovariances:

$\gamma(h) = \sigma_w^2 \sum_{i=0}^{\infty} \psi_i \psi_{i+h}$

$$\gamma(0) = \sigma_w^2 \sum_{i=0}^{\infty} \psi_i \psi_{i+0} = \sigma_w^2[1 + (\theta_1+\phi_1)^2 + \phi_1^2(\theta_1+\phi_1)^2 + \phi_1^4(\theta_1+\phi_1)^2 + ...] = \sigma_w^2[1 + \frac{(\theta_1+\phi_1)^2}{1-\phi_1^2}] = \frac{\sigma_w^2}{1-\phi_1^2}[1 - \phi_1^2 + \theta_1^2 + 2\theta_1$$

$$\gamma(1) = \sigma_w^2 \sum_{i=0}^{\infty} \psi_i \psi_{i+1} = \sigma_w^2(\psi_0\psi_1 + \psi_1\psi_2 + \psi_2\psi_3 + ...) = \sigma_w^2[(\theta_1+\phi_1) + \phi_1(\theta_1+\phi_1)^2 + \phi_1^3(\theta_1+\phi_1)^2 + ...] = \sigma_w^2[(\theta_1+\phi_1) + \frac{\phi_1($$

- $(1 - B\phi_1)x_t = (1 + \theta_1 B)w_t$ can be rewritten as

$x_t = \phi_1 x_{t-1} + w_t + \theta_1 w_{t-1}$
Then $\gamma(h) = \sum_{j=i}^{1} \phi_j \gamma(h-j) + \sigma_w^2 \sum_{j=h}^{1} \theta_j \psi_{j-h} = \phi_1 \gamma(h-1) + \sigma_w^2 \sum_{j=h}^{1} \theta_j \psi_{j-h}$
where $\theta_0 = 1$.
Thus,

$\gamma(0) = \phi_1\gamma(-1)+\sigma_w^2(\theta_0\psi_0+\theta_1\psi_1) = \phi_1\gamma(1)+\sigma_w^2(1+\theta_1\psi_1) = \phi_1\gamma(1)+\sigma_w^2[1+\theta_1(\theta_1+\phi_1)] = \phi_1\gamma(1)+\sigma_w^2[1+6$

and

$$\gamma(1) = \phi_1\gamma(0) + \sigma_w^2\theta_1\psi_0 = \phi_1\gamma(0) + \sigma_w^2\theta_1$$

because $\psi_0 = 1$.

- Solving for $\gamma(0)$ produces:

$\gamma(0) = \phi_1\gamma(1)+\sigma_w^2[1+\theta_1^2+\theta_1\phi_1] = \phi_1[\phi_1\gamma(0)+\sigma_w^2\theta_1]+\sigma_w^2[1+\theta_1^2+\theta_1\phi_1] \implies \gamma(0)-\phi_1^2\gamma(0) = \sigma_w^2\phi_1\theta_1+\sigma_w^2[1-$

- Solving for $\gamma(1)$ produces:

$\gamma(1) = \phi_1\gamma(0)+\sigma_w^2\theta_1 = \phi_1\frac{\sigma_w^2}{1-\phi_1^2}[\theta_1^2+2\theta_1\phi_1+1]+\sigma_w^2\theta_1 = \sigma_w^2[\frac{\phi_1(\theta_1^2+2\theta_1\phi_1+1)}{1-\phi_1^2}+\theta_1] = \sigma_w^2[\frac{\phi_1\theta_1^2+2\theta_1\phi_1^2}{1}$

- Note that $\gamma(h) = \sum_{j=1}^{1} \phi_j \gamma(h-j)$for h $\geq$ max(1, 2). Then $\gamma(2) = \phi_1\gamma(1), \gamma(3) = \phi_1\gamma(2), ...$

Thus,

$$\gamma(h) = \begin{cases} \frac{\sigma_w^2}{1-\phi_1^2}[\theta_1^2 + 2\theta_1\phi_2 + 1] & \text{if } h = 0 \\ \sigma_w^2\phi_1^{h-1}\frac{(1+\theta_1\phi_1)(\phi_1+\theta_1)}{1-\phi_1^2} & \text{if } h > 0 \end{cases}$$

- Finally,

$$\rho(h) = \frac{\sigma_w^2\phi_1^{h-1}\frac{(1+\theta_1\phi_1)(\phi_1+\theta_1)}{1-\phi_1^2}}{\frac{\sigma_w^2}{1-\phi_1^2}[\theta_1^2+2\theta_1\phi_1+1]} = \frac{\phi_1^{h-1}(1+\theta_1\phi_1)(\phi_1+\theta_1)}{\theta_1^2+2\theta_1\phi_1+1}$$

  for h ≥ 1

Notes:

- Unlike the ACF for a MA(q), this is not 0 after a particular lag.
- Why do we want to know the ACF of an ARMA(1,1)? If the estimated ACF of a time series data set exhibits similar characteristics (i.e, does not "cut off" to 0, but rather "tails off" to 0), then we may think the time series data set can be modeled with an ARMA(1,1)!

Example 12.2. ACF of an AR(p)

- Because q = 0, the autocovariance function is

$$\gamma(0) = \sum_{j=1}^{p} \psi_j \gamma(h-j) + \sigma_w^2 \sum_{j=0}^{0} \theta_j \psi_{j-0} = \sum_{j=1}^{p} \phi_k \gamma(h-j) + \sigma_w^2$$

and $\gamma(h) = \sum_{j=1}^{p} \phi_j \gamma(h-j)$ for h $\geq$ 1.

- Suppose p = 1. Then $\gamma(0) = \phi_1\gamma(-1) + \sigma_w^2 = \phi_1\gamma(1) + \sigma_w^2$, $\gamma(1) = \phi_1\gamma(0), \gamma(2) = \phi_1\gamma(1), ...$

Thus, $\gamma(0) = \phi_1\gamma(1) + \sigma_w^2 = \phi_1^2\gamma(0) + \sigma_w^2 \implies \gamma(0) = \frac{\sigma_w^2}{1-\phi_1^2}$.

Also, $\gamma(1) = \frac{\phi_1\sigma_w^2}{1-\phi_1^2}, \gamma(2) = \frac{\phi_1^2\sigma_w^2}{1-\phi_1^2}, ...$

Finally, $\gamma(h) = \frac{\phi_1^h\sigma_w^2}{1-\phi_1^2}$

The ACF is $\rho(h) = \frac{\gamma(h)}{\gamma(0)} = \frac{\frac{\phi_1^h\sigma_w^2}{1-\phi_1^2}}{\frac{\sigma_w^2}{1-\phi_1^2}} = \phi_1^h$

Remember that -1 < ϕ_1 < 1 to ensure stationarity. Notice how $\rho(h)$ will slowly "tail off" to 0. While tailing off, it can be alternating between positive and negative values.

- Suppose p=2. Then

$$\gamma(0) = \sum_{j=1}^{2} \phi_j \gamma(0-j) + \sigma_w^2 \sum_{j=0}^{0} \theta_j \psi_{j=0} = \phi_1\gamma(-1) + \phi_2\gamma(-2) + \sigma_w^2 = \phi_1\gamma(1) + \phi_2\gamma(2) + \sigma_w^2,$$

$$\gamma(1) = \phi_1\gamma(0) + \phi_2\gamma(-1) = \phi_1\gamma(0) + \phi_2\gamma(1),$$

$\gamma(2) = \phi_1\gamma(1) + \phi_2\gamma(0)$,

$\gamma(3) = \phi_1\gamma(2) + \phi_2\gamma(1)$,

Then$\rho(h) = \frac{\gamma(h)}{\gamma(0)} = \phi_1\rho(h-1) + \phi_2\rho(h-2)$ for h ge 1.

Therefore, $\rho(1) = \phi_1\rho(0) + \phi_2\rho(-1) = \phi_1 + \phi_2\rho(1) \implies \rho(1) = \frac{\phi_1}{1-\phi_2}$.

$\rho(2) = \phi_1\rho(1) + \phi_2\rho(0) = \frac{\phi_1^2}{1-\phi_2} + \phi_2 = \frac{\phi_1^2+\phi_2-\phi_2^2}{1-\phi_2}$

Remember that -1 < ϕ_2 < 1, $\phi_2+\phi_1 < 1$, and $\phi_2-\phi_1 < 1$ to ensure stationarity. Notice how $\rho(h)$ will slowly "tail off" to 0.

Notes:

- Unlike the ACF for a MA(q), the ACF is not 0 after a particular lag.
- Table of ACFs for common ARMA models.

Model	Equation	Autocorrelation
AR(1)	$x_t = \phi_1 x_{t-1} + w_t$	$\rho(h) = \phi_1^h$
AR(2)	$x_t =$ $\phi_1 x_{t-1} + \phi_2 x_{t-2} + w_t$	$\rho(h) = \phi_1\rho(h-1) + \phi_2\rho(h-2)$ for $h \geq 1$
MA(1)	$x_t = \theta_1 w_{t-1} + w_t$	$\rho(h) = \begin{cases} \frac{\theta_1}{1+\theta_1^2} & h = 1 \\ 0 & h > 1 \end{cases}$
MA(2)	$x_t =$ $\theta_1 w_{t-1} + \theta_2 w_{t-2} + w_t$	$\rho(h) = \begin{cases} \frac{\theta_1(1+\theta_2)}{1+\theta_1^2+\theta_2^2} & h = 1 \\ \frac{\theta_2}{1+\theta_1^2+\theta_2^2} & h = 2 \\ 0 & h > 2 \end{cases}$

Remember: "+" signs are used in the moving average operator, $\theta(B)$. **Many textbooks use "-" signs so be careful when you examine these book!!!!** Thus, ACFs may be written a different way.

In summary,

Model	AR(p)	MA(q)	ARMA(p,q)
ACF	Tails off to 0	Cuts off to 0 after lag q	Tails off to 0 after lag q

Example 12.3. AR(1) with $\phi_1 = 0.7, -0.7$

```
# Simulate AR(1) using arima.sim()
set.seed(7181)

x <- arima.sim(model = list(ar = c(0.7)), n = 100, rand.gen = rnorm, sd = 10)

dev.new(width = 8, height = 6, pointsize = 10)  #Opens up wider plot window than the default (goo

plot(x = x, ylab = expression(x[t]), xlab = "t", type = "l", col = "red",
       main = expression(paste(x[t] == 0.7*x[t-1] + w[t], " where ", w[t], "~ ind. N(0,100)")) ,
       panel.first=grid(col = "gray", lty = "dotted"))

points(x = x, pch = 20, col = "blue")
```

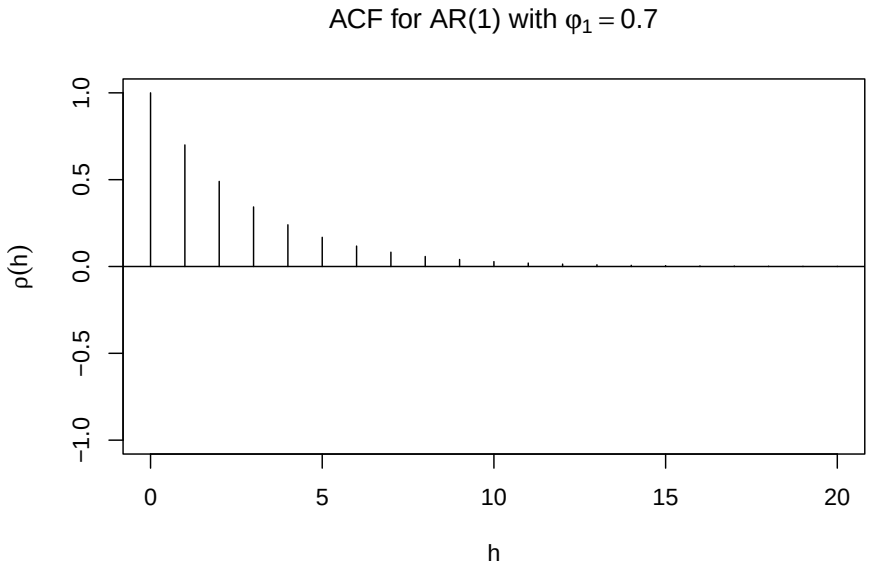
```
# Use ARMAacf() and ARMAtoMA() functions
round(ARMAacf(ar = c(0.7), lag.max = 20), 4)
```

```
##      0      1      2      3      4      5      6      7      8      9     10
## 1.0000 0.7000 0.4900 0.3430 0.2401 0.1681 0.1176 0.0824 0.0576 0.0404 0.0282
##     11     12     13     14     15     16     17     18     19     20
## 0.0198 0.0138 0.0097 0.0068 0.0047 0.0033 0.0023 0.0016 0.0011 0.0008
```

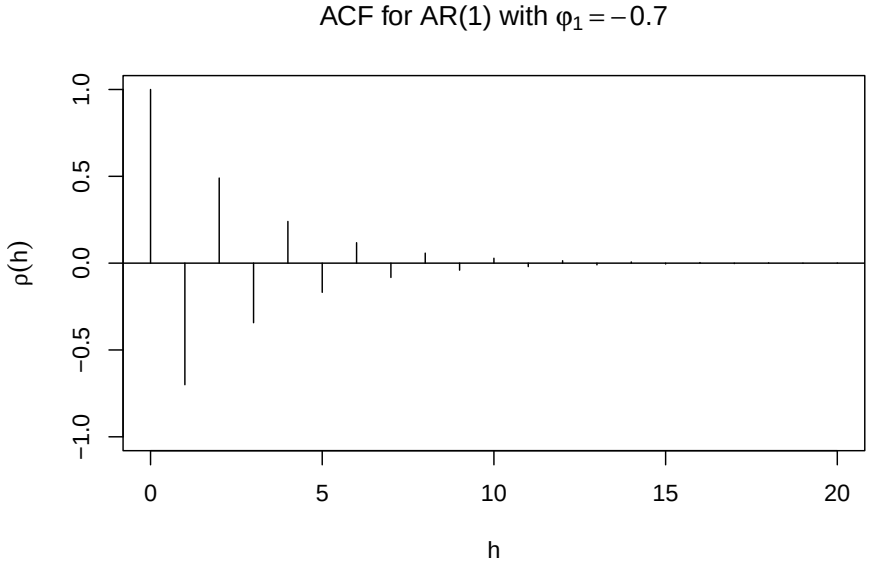
```
round(ARMAacf(ar = c(-0.7), lag.max = 20),4)
```

```
##       0       1       2       3       4       5       6       7       8       9
##  1.0000 -0.7000  0.4900 -0.3430  0.2401 -0.1681  0.1176 -0.0824  0.0576 -0.0404
##      10      11      12      13      14      15      16      17      18      19
##  0.0282 -0.0198  0.0138 -0.0097  0.0068 -0.0047  0.0033 -0.0023  0.0016 -0.0011
##      20
##  0.0008
```

```
# Use ARMAacf() and ARMAtoMA() functions

plot(y = ARMAacf(ar = c(0.7), lag.max = 20), x = 0:20, type = "h", ylim = c(-1,1), xlab = "h", yl
       main = expression(paste("ACF for AR(1) with ", phi1[1] == 0.7)))

abline(h = 0)
```




```
plot(y = ARMAacf(ar = c(-0.7), lag.max = 20), x = 0:20, type = "h", ylim = c(-1,1), xla
       main = expression(paste("ACF for AR(1) with ", phi1[1] == -0.7)))

abline(h = 0)
```



```
round(ARMAtoMA(ar = c(0.7), lag.max = 20), 4)
```

```
##  [1] 0.7000 0.4900 0.3430 0.2401 0.1681 0.1176 0.0824 0.0576 0.0404 0.0282
## [11] 0.0198 0.0138 0.0097 0.0068 0.0047 0.0033 0.0023 0.0016 0.0011 0.0008
```

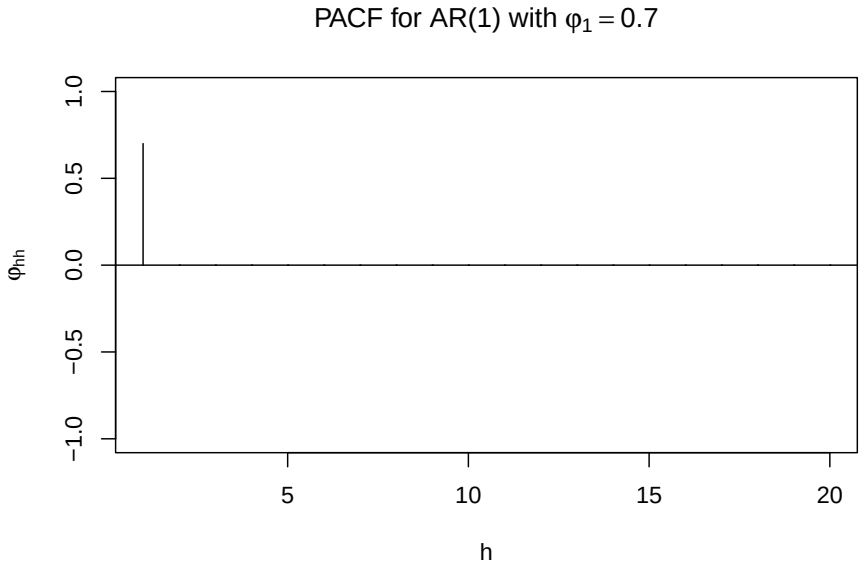
```
#Example for AR(2)
round(ARMAtoMA(ar = c(0.7, -0.4), lag.max = 20), 4)
```

```
##  [1]  0.7000  0.0900 -0.2170 -0.1879 -0.0447  0.0438  0.0486  0.0165 -0.0079
## [10] -0.0121 -0.0053  0.0011  0.0029  0.0016 -0.0001 -0.0007 -0.0005  0.0000
## [19]  0.0001  0.0001
```

```
# PACF

round(ARMAacf(ar = c(0.7), lag.max = 20, pacf = TRUE),4)
```

```
##  [1] 0.7 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
## [20] 0.0
```

```
plot(x = ARMAacf(ar = c(0.7), lag.max = 20, pacf = TRUE), type = "h", ylim = c(-1,1), xlab = "h",
       main = expression(paste("PACF for AR(1) with ", phi1[1] == 0.7)))

abline(h = 0)
```



```
# Estimated ACF and PACF

dev.new(width = 8, height = 6, pointsize = 10)

par(mfrow = c(1,2))


acf(x = x, type = "correlation", lag.max = 20, ylim = c(-1,1), xlab = "h",
    main = "Estimated ACF")

abline(h = c(-0.5, 0.5), lty = "dotted", col = "gray")

pacf(x = x, lag.max = 20, ylim = c(-1,1), xlab = "h",
    main = "Estimated PACF")

abline(h = c(-0.5, 0.5), lty = "dotted", col = "gray")
```

The `ARMAacf()` function calculated the ACF.

In an upcoming example with arma_sample.R, we will compare these plots to an estimated ACF plot. For example, we will see how to use `arima.sim()` to simulate observations from an ARMA process. When $\phi_1 = -0.7$, this is what occurs for a sample of size 1,000 and $\sigma_w^2 = 1$:

The horizontal lines for the ACF and PACF plots are drawn at $\pm 1.96 n^{-1/2} = \pm \frac{1.96}{\sqrt{1000}}$.

Try using the `ARMAacf()` function to see what other ACF plots would look like for different values of ϕ_1!

12.3 PACF for ARMA models

The ACF really helps to find the order of a MA(q) model, but does not help much for finding the order of an AR(p) (because the ACF does not "cut-off" to 0, we can not determine the order of AR model by ACF). The partial autocorrelation function (PACF) helps to find the order of an AR(p)!

Consider the following autoregressive representation:

$$x_{t+h} = \beta_{h1} x_{t+h-1} + \beta_{h2} x_{t+h-2} + ... + \beta_{hh} x_t + w_{t+h}$$

where $w_{t+h} \sim indN(0, \sigma^2)$

NOTE: The different notation is used here to help motivate you to think of this as a regression model.

Remember the interpretation of a β parameter - GIVEN all of the other variables in the model, β measures the relationship between its corresponding independent variable and the dependent variable.

Multiplying both sides by x_{t+h-j} and taking the expectation produces:

$$E(x_{t+h-j}x_{t+h}) = \beta_{h1}E(x_{t+h-j}x_{t+h-1})+\beta_{h2}E(x_{t+h-j}x_{x+h-2})+...+\beta_{hh}E(x_{t+h-j}x_t)+E(x_{t+h-j}w_{t+h})$$

Note that $E(x_{t+h-j}w_{t+h}) = E(x_{t+h-j})E(w_{t+h})$ because x_{t+h-j} and w_{t+h} are independent (x_{t+h-j} has w's with subscripts lower than t+h). Then $E(x_{t+h-j}w_{t+h}) = E(x_{t+h-j}) \times 0 = 0$.

Continuing,

$$E(x_{t+h-j}x_{t+h}) = \beta_{h1}E(x_{t+h-j}x_{t+h-1})+\beta_2E(x_{t+h-j}x_{t+h-2})+...+\beta_{hh}E(x_{t+h-j}x_t)+0 \implies \gamma(j) = \beta_{h1}\gamma(1-j)+\beta_{h2}\gamma(2-$$

Remember $\rho(j) = \frac{\gamma(j)}{\gamma(0)}$

Then for j=1,...,h:

$$\rho(1) = \beta_{h1}\rho(0)+\beta_{h2}\rho(1)+...+\beta_{hh}\rho(h-1)\rho(2) = \beta_{h1}\rho(1)+\beta_{h2}\rho(0)+...+\beta_{hh}\rho(h-2)\rho(h) = \beta_{h1}\rho(h-1)+\beta_{h2}\rho(h-2)+...+$$

Remember that $\rho(-j) = \rho(j)$

Suppose h = 1, then

$\rho(1) = \beta_{11}\rho(0) \implies \beta_{11} = \rho(1)$ because $\rho(0) = 1$

Remember if h = 1, the model is $x_{t+1} = \beta_{11}x_t + w_{t+1}$

Suppose h = 2, then

$$\rho(1) = \beta_{21}\rho(0) + \beta_{22}\rho(1)\rho(2) = \beta_{21}\rho(1) + \beta_{22}\rho(0) \implies \beta_{22} = \frac{\rho(2) - \rho(1)^2}{1 - \rho(1)^2}$$

from solving the above system of equations.

Suppose h = 3, then

$$\rho(1) = \beta_{31}\rho(0)+\beta_{32}\rho(1)+\beta_{33}\rho(2)\rho(2) = \beta_{31}\rho(1)+\beta_{32}\rho(0)+\beta_{33}\rho(1)\rho(3) = \beta_{31}\rho(2)+\beta_{32}\rho(1)+\beta_{33}\rho(0) \implies \beta_{33} = \frac{}{1-\rho}$$

This process can be continued to find β_{hh}.

These β's are called partial autocorrelations because they measure the dependence of x_t on x_{t+h} removing the effect of all the other random variables in between. Thus,

$$\beta_{11} = Corr(x_t, x_{t+1})\beta_{22} = Corr(x_t, x_{t+2}|x_{t+1})\beta_{33} = Corr(x_t, x_{t+3}|x_{t+1}, x_{t+2})$$

Also, these can be treated like "regular" correlations in terms of scale: $-1 \leq \beta_{hh} \leq 1$.

Most textbooks use ϕ_{hh} to denote the partial autocorrelations at lag h.

THIS CAN BE VERY CONFUSING SINCE MOST BOOKS USE ϕ_j TO DENOTE A PARAMETER FROM AN AUTOREGRESSIVE MODEL!!!!

I will use this notation from now on. Thus, β_{hh} in the old notation is ϕ_{hh} in the new notation!!!

Example 12.4. AR(1)

We have seen that $\rho(h) = \phi_1^h$.

$$\phi_{11} = \rho(1) = \phi_1\phi_{22} = \frac{\rho(2) - \rho(1)^2}{1 - \rho(1)^2} = \frac{\phi_1^2 - \phi_1^2}{1 - \phi_1^2} = 0\phi_{33} = 0, ...$$

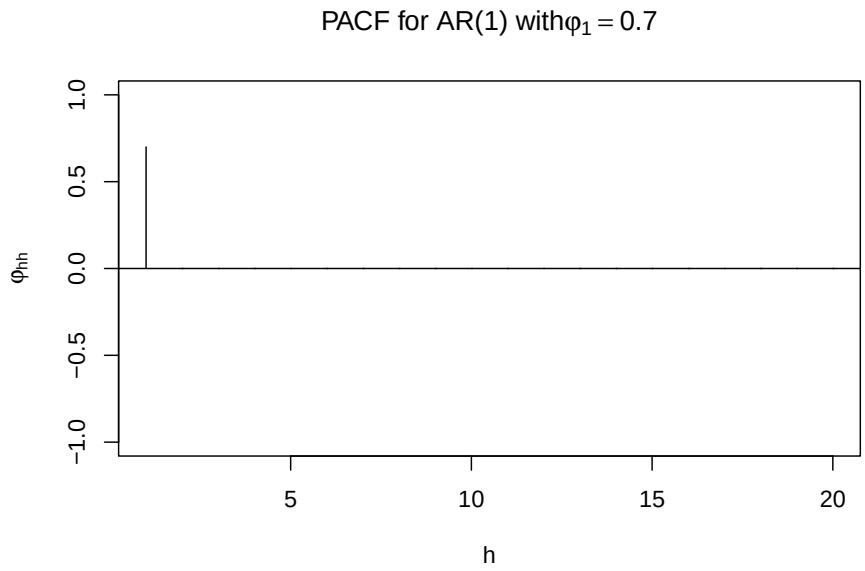
Thus, a AR(1)'s PACF cuts off to 0 after lag 1.

Suppose $\phi_1 = 0.7$. Using `ARMAacf()` with the `PACF = TRUE` option produces

```
ARMAacf(ar=c(0.7), lag.max=20, pacf=TRUE)
```

```
##  [1] 0.7 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
## [20] 0.0
```

```
plot(x=ARMAacf(ar=c(0.7), lag.max = 20, pacf=TRUE), type = "h",
     ylim=c(-1,1), xlab="h", ylab = expression(phi1[hh]), main = expression(paste("PACF

abline(h = 0)
```



Note that the first PACF value given is for h = 1 so the problems encountered earlier with the ACF and the`plot()` function do not occur here.

Example 12.5. AR(2)

From past notes, we have

$$\rho(h) = \phi_1\rho(h-1) + \phi_2\rho(h-2)\rho(1) = \frac{\phi_1}{1-\phi_2}\rho(2) = \frac{\phi_1^2 + \phi_2 - \phi_2^2}{1-\phi_2}$$

Then

$$\phi_{11} = \rho(1) = \frac{\phi_1}{1-\phi_2}\phi_{22} = \frac{\rho(2) - \rho(1)^2}{1-\rho(1)^2} = \frac{\frac{\phi_1^2+\phi_2-\phi_2^2}{1-\phi_2} - (\frac{\phi_1}{1-\phi_2})^2}{1-(\frac{\phi_1}{1-\phi_2})^2} = \frac{(\phi_1^2+\phi_2-\phi_2^2)(1-\phi_2)-\phi_1^2}{(1-\phi_2)^2-\phi_1^2} = \frac{\phi_2 - 2\phi_2^2 + \phi_2^3 - \phi_1^2}{(1-\phi_2)^2-\phi_1^2}$$

An AR(2)'s PACF cuts off to 0 after lag 2.

Notes:

- ARMA models with q > 0 do not have the partial correlations "cut-off" to 0. Instead, they behave like the ACF does for models with p > 0.
- Verify on your own that

$$\phi_{hh} = -\frac{(-\theta)^h(1-\theta^2)}{1-\theta^{2(h+1)}}$$

for an MA(1) (see Shumway and Stoffer).

From ma1_sim.R,

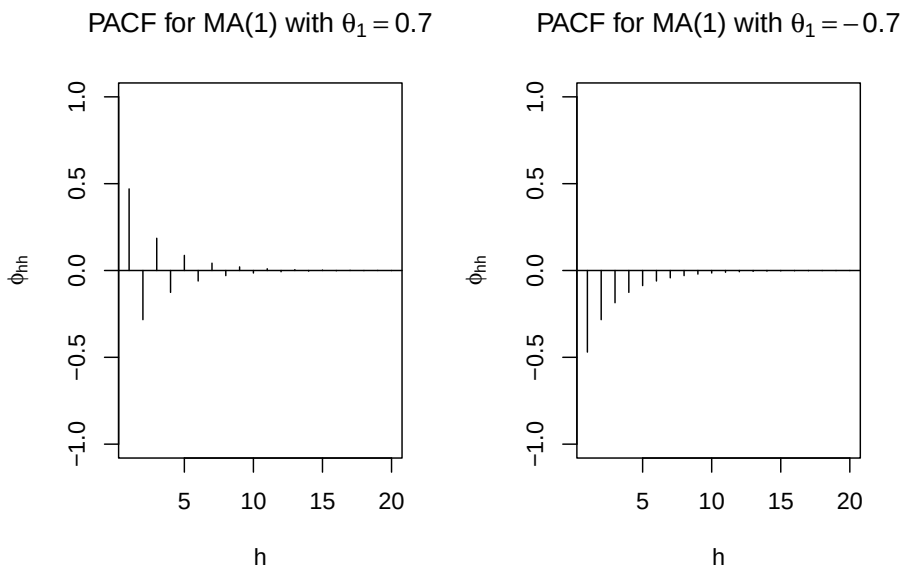
```
# PACF

par(mfrow = c(1,2))

round(ARMAacf(ma = c(0.7), lag.max = 20, pacf = TRUE),4)
```

```
##  [1]  0.4698 -0.2832  0.1856 -0.1260  0.0869 -0.0604  0.0421 -0.0294  0.0206
## [10] -0.0144  0.0101 -0.0071  0.0049 -0.0035  0.0024 -0.0017  0.0012 -0.0008
## [19]  0.0006 -0.0004
```

```
plot(x = ARMAacf(ma = c(0.7), lag.max = 20, pacf = TRUE), type = "h", ylim = c(-1,1), x
       main = expression(paste("PACF for MA(1) with ", theta[1] == 0.7)))

abline(h = 0)

round(ARMAacf(ma = c(-0.7), lag.max = 20, pacf = TRUE),4)
```

```
##  [1] -0.4698 -0.2832 -0.1856 -0.1260 -0.0869 -0.0604 -0.0421 -0.0294 -0.0206
## [10] -0.0144 -0.0101 -0.0071 -0.0049 -0.0035 -0.0024 -0.0017 -0.0012 -0.0008
## [19] -0.0006 -0.0004
```

```
plot(x = ARMAacf(ma = c(-0.7), lag.max = 20, pacf = TRUE), type = "h", ylim = c(-1,1),
       main = expression(paste("PACF for MA(1) with ", theta[1] == -0.7)))

abline(h = 0)
```



- Estimating the PACF

$\hat{\phi}_{11} = \hat{\rho}(1)$

$\hat{\phi}_{hh} = \frac{\hat{\rho} - \sum_{j=1}^{h-1} \hat{\phi}_{h-1,j}\hat{\rho}(h-j)}{1 - \sum_{j=1}^{h-1} \hat{\phi}_{h-1,j}\hat{\rho}(h-j)}$ for h=2,3,...

where $\hat{\phi}_{hj} = \hat{\phi}_{h-1,j} - \hat{\phi}_{hh}\hat{\phi}_{h-1,h-j}$ for h=3,4,...;j=1,2,...h-1. For more on this expression, see the exercises of Shumway and Stoffer. We will not calculate these by hand. Rather, we will use the `pacf()` function in R.

An approximation for the standard error is $\sigma_{\hat{\phi}_{hh}} = n^{-1/2}$. To test

$H_0 : \phi_{hh} = 0$ $Ha : \phi_{hh} \neq 0$

Reject H_0 if $|\frac{\hat{\phi}-0}{n^{-1/2}}| > Z_{1-\frac{\alpha}{2}}$

we can check if $\hat{\phi}_{hh}$ is within $\pm Z_{1-\alpha/2} \times n^{-1/2}$.

- Why would you want to do the above hypothesis test?

- Suppose you observed a time series. The estimated values of the PACF are $\hat{\phi}_{11} \neq 0$ and $\hat{\phi}_{hh} \approx 0$ for h > 1. What may be a good model for the data? AR(1)