Appendix

魏上傑

2023-04-27

目錄

1 Time Series Primer

To create a time series object, use the command `ts`.

Use `as.ts` to coerce an object to a time series and `is.ts` to test whether an object is a time series.

```
mydata <- c(1,2,3,2,1)
mydata
```

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

```
# Make it a time series
mydata <- as.ts(mydata)
```

```
# Make it an annual time series starting in 1950
mydata <- ts(mydata, start = 1950)
mydata
```

```
## Time Series:
## Start = 1950
## End = 1954
## Frequency = 1
## [1] 1 2 3 2 1
```

```r
# Make it a quarterly time series starting in 1953Q3
mydata <- ts(mydata, start = c(1950,3), frequency = 4)
mydata
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1950                1    2
## 1951    3    2    1
```

```r
# view the sampled times
time(mydata)
```

```
##         Qtr1    Qtr2    Qtr3    Qtr4
## 1950                 1950.50 1950.75
## 1951 1951.00 1951.25 1951.50
```

To use part of a time series object, use `window()`

```r
x <- window(mydata, start=c(1951,1), end=c(1951,3))
x
```

```
##      Qtr1 Qtr2 Qtr3
## 1951    3    2    1
```

Next, we'll look at lagging and differencing. First make a simple series, x_t

```r
x <- ts(1:5)
x
```

```
## Time Series:
## Start = 1
## End = 5
## Frequency = 1
## [1] 1 2 3 4 5
```

Now, column bind (`cbind`) lagged values of x_t and notice that `lag(x)` is *forward* lag, whereas `lag(x, -1)` is *backward* lag.

```r
cbind(x, lag(x), lag(x, -1))
```

```
## Time Series:
## Start = 0
## End = 6
## Frequency = 1
##    x lag(x) lag(x, -1)
## 0 NA      1         NA
## 1  1      2         NA
## 2  2      3          1
## 3  3      4          2
## 4  4      5          3
## 5  5     NA          4
## 6 NA     NA          5
```

Notice that at 3, for example, x is 3, `lag(x)` is ahead at 4, and `lag(x, -1)` is behind at 2

Compare `cbind` and `ts.intersect`

```
ts.intersect(x, lag(x, 1), lag(x, -1))
```

```
## Time Series:
## Start = 2
## End = 4
## Frequency = 1
##   x lag(x, 1) lag(x, -1)
## 2 2         3          1
## 3 3         4          2
## 4 4         5          3
```

To difference a series, $\nabla x_t = x_t - x_{t-1}$, use

```
diff(x)
```

```
## Time Series:
## Start = 2
## End = 5
## Frequency = 1
## [1] 1 1 1 1
```

but note that

```
# lag=2, difference =1
diff(x, 2)
```

```
## Time Series:
## Start = 3
## End = 5
## Frequency = 1
## [1] 2 2 2
```

is **NOT** second order differencing, it is $x_t - x_{t-2}$. For second order differencing, that is $\nabla^2 x_t$, do this:

```
diff(diff(x))
```

```
## Time Series:
## Start = 3
## End = 5
## Frequency = 1
## [1] 0 0 0
```

or

```
diff(x, lag = 1, difference=2)
```

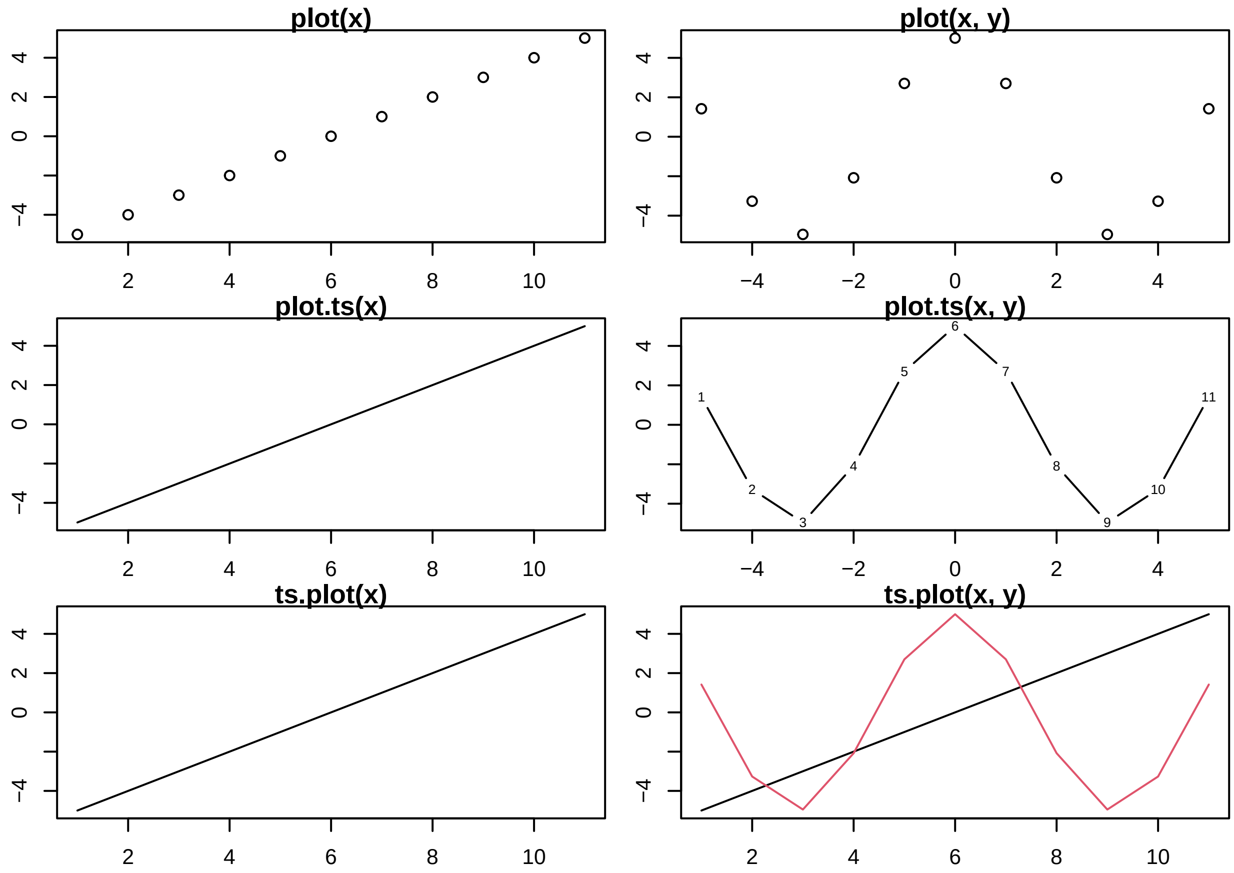
```
## Time Series:
## Start = 3
## End = 5
## Frequency = 1
## [1] 0 0 0
```

and so on for higher order differencing.

For graphing time series, if `x` is a time series, the `plot(x)` will produce a time plot. If `x` is not a time series object, then `plot.ts(x)` will coerce it into a time plot as will `ts.plot(x)`. There are differences, which we explore in the following.

```
x=-5:5 # x is not a time series object
y= 5*cos(x) # neither is y
op <- par(mfrow=c(3,2)) #multifigure setup: 3 rows, 2 cols
par(mar=c(2,2,1,1))
plot(x, main="plot(x)")
plot(x, y, main="plot(x, y)")
```

```
plot.ts(x, main="plot.ts(x)")
plot.ts(x, y, main="plot.ts(x, y)")
ts.plot(x, main="ts.plot(x)")
ts.plot(ts(x), ts(y), col=1:2, main=("ts.plot(x, y)"))
```




```
par(op) # reset the graphics parameters
```

We will also use regression via `lm()`.

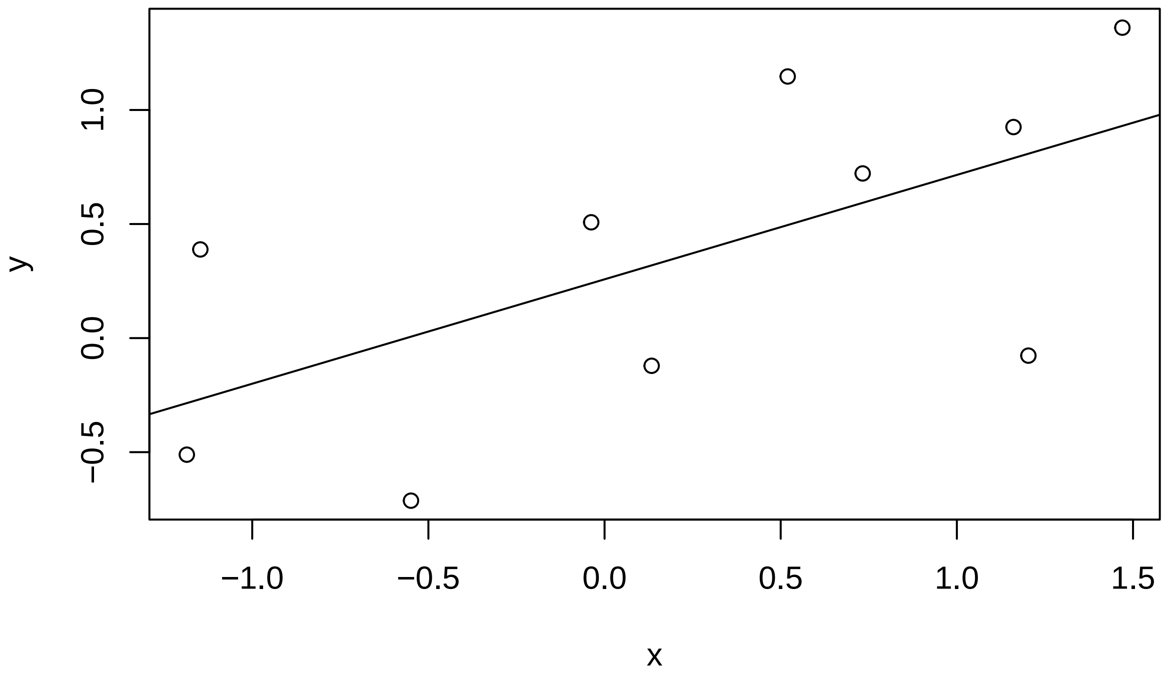
Suppose we want to fit $y = \alpha + \beta x + \epsilon$

```
set.seed(1999)
x <- rnorm(10, 0, 1)
y <- x+rnorm(10, 0, 1)
summary(fit <- lm(y~x))
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q  Median      3Q     Max
## -0.8851 -0.3867  0.1325  0.3896  0.6561
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.2576     0.1892   1.362   0.2104
## x             0.4577     0.2016   2.270   0.0529 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.58 on 8 degrees of freedom
## Multiple R-squared:  0.3918, Adjusted R-squared:  0.3157
## F-statistic: 5.153 on 1 and 8 DF,  p-value: 0.05289
```

```r
plot(x, y) # draw a scatterplot of the data
abline(fit) # add the fitted line to the plot
```




```r
resid(fit) # display the residuals
```

```
##          1          2          3          4          5          6          7
##  0.1288372  0.2672500 -0.8851025  0.4304056 -0.4402385  0.6512681 -0.7188025
##          8          9         10
## -0.2259478  0.6560924  0.1362381
```

```
fitted(fit) # display the fitted values
```

```
##           1           2           3           4           5           6
##  0.592949606  0.240309304  0.808211365  0.930316045  0.318809880  0.495535464
##           7           8           9          10
##  0.006184128 -0.285001160 -0.267459264  0.788866713
```

```
lm(y~0+x) # exclude the intercept
```

```
##
## Call:
## lm(formula = y ~ 0 + x)
##
## Coefficients:
##     x
## 0.525
```

If you use `lm()` for lagged values of a time series, then you need to 'tie' the series together using `ts.intersect`.

```
library(astsa)
ded <- ts.intersect(cmort, part, part4=lag(part,-4), dframe = TRUE)
```

```
fit <- lm(cmort~ part+part4, data=ded, na.action = NULL)
summary(fit)
```

```
##
## Call:
## lm(formula = cmort ~ part + part4, data = ded, na.action = NULL)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -22.743  -5.368  -0.414   5.269  37.854
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 69.01020    1.37498  50.190  < 2e-16 ***
## part         0.15140    0.02898   5.225 2.56e-07 ***
## part4        0.26297    0.02899   9.071  < 2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.323 on 501 degrees of freedom
## Multiple R-squared:  0.3091, Adjusted R-squared:  0.3063
## F-statistic: 112.1 on 2 and 501 DF,  p-value: < 2.2e-16
```

There is package called `dynlm` that makes it easy to fit lagged regressions. The basic advantage of `dynlm` is that it avoids having to make a data frame.

In problem 2.1, you are asked to fit a regression model

$$x_t = \beta t + \alpha_1 Q_1(t) + \alpha_2 Q_2(t) + \alpha_3 Q_3(t) + \alpha_4 Q_4(t) + w_t$$

where x_t is logged Johnson & Johnson quarterly earnings (n=84), and $Q_i(t)$ is the indicator of quarter i=1,2,3,4. The indicators can be made using `factor`

```
trend <- time(jj) -1970 # help center time
Q <- factor(rep(1:4, 21)) # make Quarter factors
reg <- lm(log(jj)~0+trend+Q, na.action=NULL) # no intercept
# model.matrix(reg) # view the model matrix
```

```
summary(reg)
```

```
##
## Call:
## lm(formula = log(jj) ~ 0 + trend + Q, na.action = NULL)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.29318 -0.09062 -0.01180  0.08460  0.27644
##
## Coefficients:
##       Estimate Std. Error t value Pr(>|t|)
## trend 0.167172   0.002259   74.00   <2e-16 ***
## Q1    1.052793   0.027359   38.48   <2e-16 ***
## Q2    1.080916   0.027365   39.50   <2e-16 ***
## Q3    1.151024   0.027383   42.03   <2e-16 ***
## Q4    0.882266   0.027412   32.19   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1254 on 79 degrees of freedom
```

```
## Multiple R-squared:  0.9935, Adjusted R-squared:  0.9931
## F-statistic:  2407 on 5 and 79 DF,  p-value: < 2.2e-16
```

The workhorse for ARIMA simulations is `arima.sim`

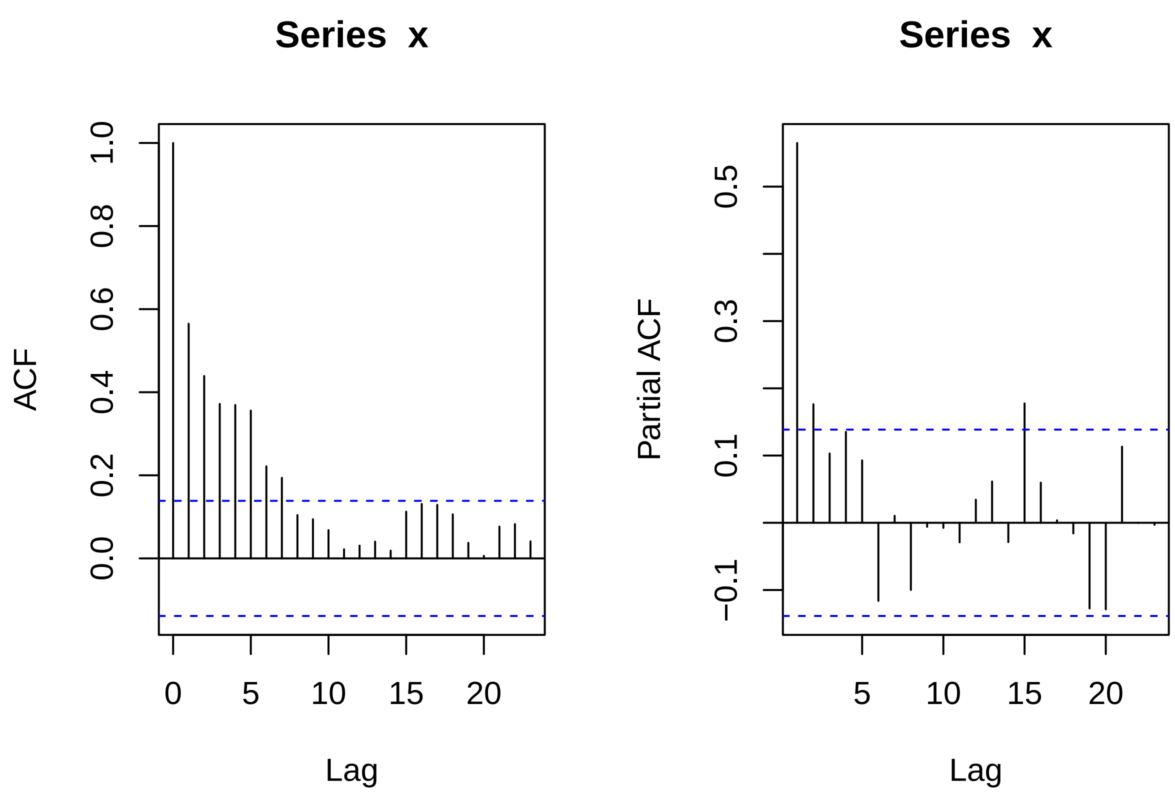
```
x <- arima.sim(list(order=c(1,0,0), ar=.9),n=100)+50 #AR(1) w/ mean 50
x = arima.sim(list(order=c(2,0,0),ar=c(1,-.9)),n=100) # AR(2)
x = arima.sim(list(order=c(1,1,1),ar=.9,ma=-.5),n=200) # ARIMA(1,1,1)
```

Next, we'll discuss ARIMA estimation.

First, we'll fit an ARMA(1,1) model to some simulated data.

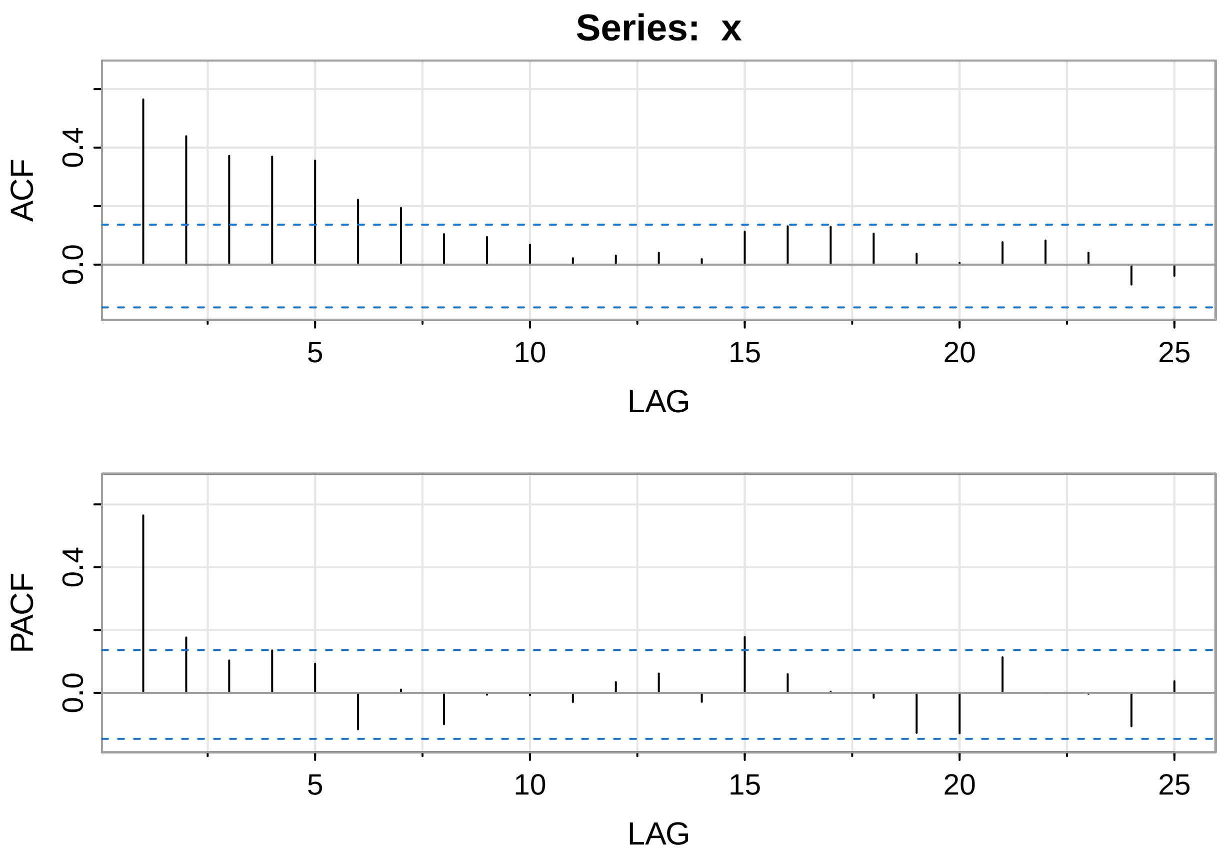
```
set.seed(666)
x = 50 + arima.sim(list(order=c(1,0,1), ar=.9, ma=-.5), n=200)

par(mfrow=c(1,2))
acf(x); pacf(x) # display sample ACF and PACF ... or ...
```




```
par(mfrow=c(1,1))
```

```r
acf2(x) # use our script
```




```
##      [,1] [,2] [,3] [,4] [,5]  [,6] [,7] [,8]  [,9] [,10] [,11] [,12] [,13]
## ACF  0.56 0.44 0.37 0.37 0.36  0.22 0.19  0.1  0.09  0.07  0.02  0.03  0.04
## PACF 0.56 0.18 0.10 0.14 0.09 -0.12 0.01 -0.1 -0.01 -0.01 -0.03  0.03  0.06
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.02  0.11  0.13  0.13  0.11  0.04  0.01  0.08  0.08  0.04 -0.07 -0.04
## PACF -0.03  0.18  0.06  0.00 -0.02 -0.13 -0.13  0.11  0.00  0.00 -0.11  0.04
```

```r
x.fit = arima(x, order = c(1, 0, 1)) # fit the model
x.fit
```

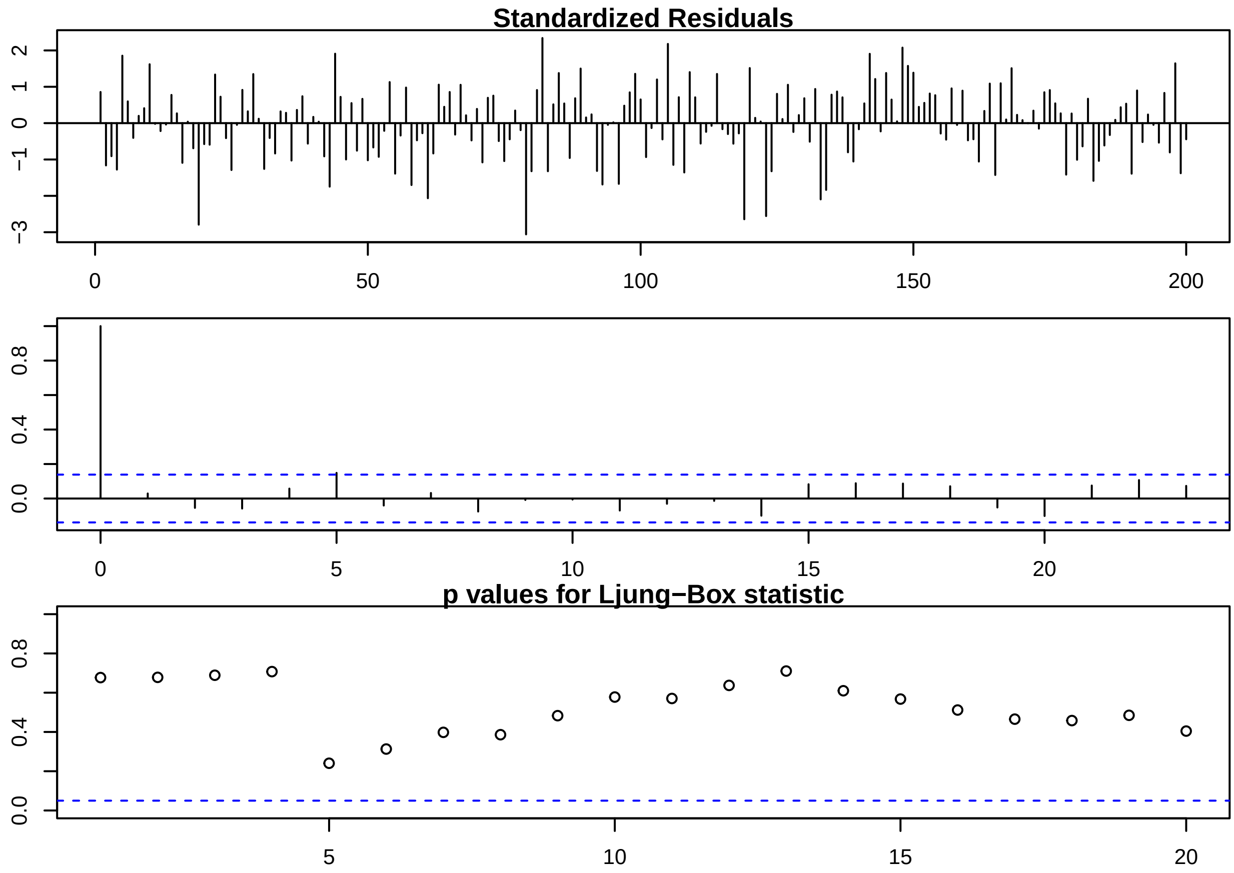
```
##
## Call:
## arima(x = x, order = c(1, 0, 1))
##
## Coefficients:
##          ar1     ma1  intercept
##       0.8340  -0.432    49.8962
## s.e.  0.0645   0.111     0.2452
##
## sigma^2 estimated as 1.07:  log likelihood = -290.79,  aic = 589.58
```

Note that the reported `intercept` estimate is an estimate of the mean and *NOT* the constant.

THe fitted model is

$$\hat{x}_t - 49.896 = .834(x_{t-1} - 49.896) + \hat{w}_t, \hat{\sigma}^2_w = 1.070$$

```
# Diagonostics
par(mar=c(2,2,1,1))
tsdiag(x.fit, gof.lag = 20) # don't use this
```

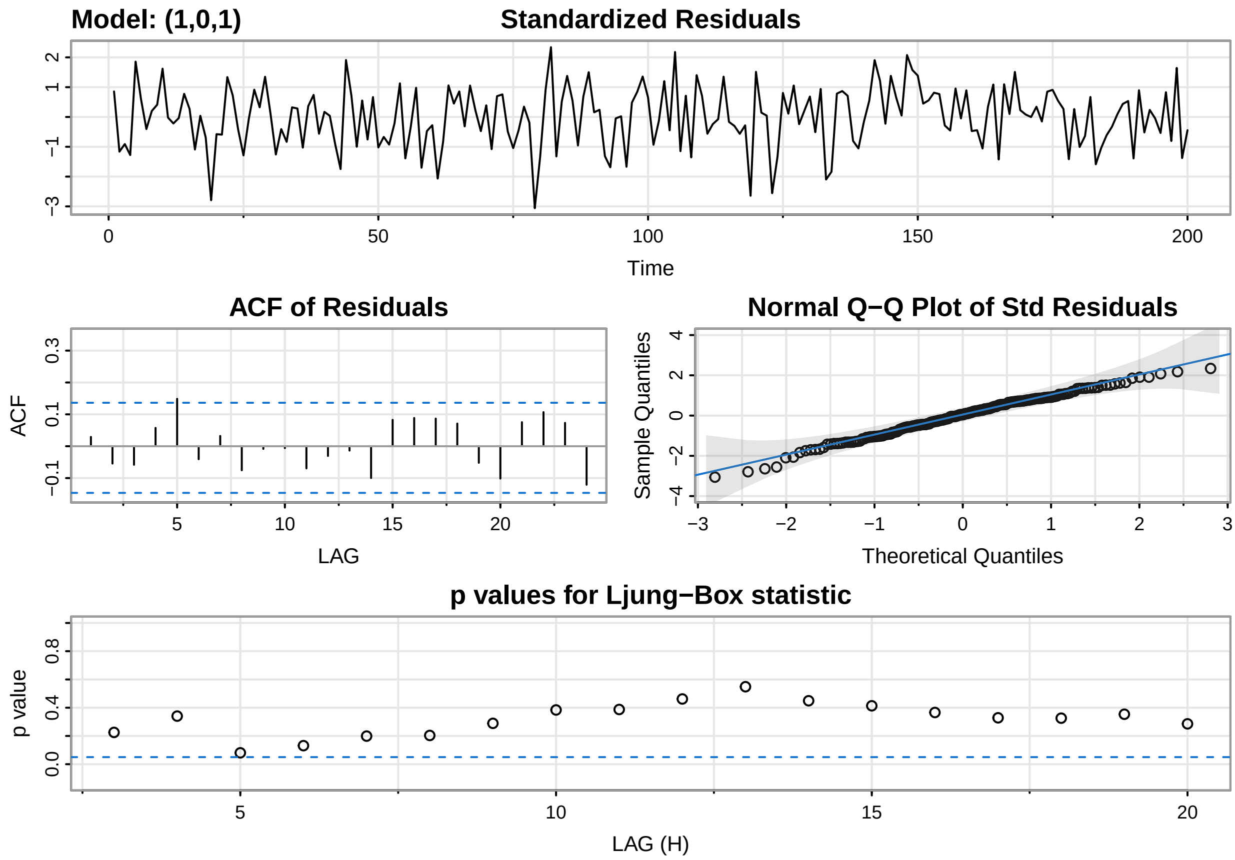


but the Ljung-Box-Pierce test is not correct because it does not take into account the fact that the residuals are from a fitted model. If the analysis is repeated using the `sarima` script, a partial output would look like the following (`sarima` will also display the correct diagnostics as a graphic; e.g., see Figure 3.17 on page 151):

```
sarima(x, 1, 0, 1)
```

```
## initial  value 0.252132
## iter   2 value 0.146060
## iter   3 value 0.086279
## iter   4 value 0.074070
## iter   5 value 0.057867
## iter   6 value 0.049119
## iter   7 value 0.041873
```

```
## iter   8 value 0.039568
## iter   9 value 0.038863
## iter  10 value 0.037761
## iter  11 value 0.037498
## iter  12 value 0.037471
## iter  13 value 0.037470
## iter  14 value 0.037470
## iter  15 value 0.037468
## iter  16 value 0.037468
## iter  17 value 0.037468
## iter  18 value 0.037468
## iter  19 value 0.037468
## iter  20 value 0.037468
## iter  21 value 0.037468
## iter  22 value 0.037468
## iter  22 value 0.037468
## final  value 0.037468
## converged
## initial  value 0.035175
## iter   2 value 0.035101
## iter   3 value 0.035079
## iter   4 value 0.035042
## iter   5 value 0.035015
## iter   6 value 0.035008
## iter   7 value 0.035007
## iter   8 value 0.035007
## iter   9 value 0.035007
## iter  10 value 0.035007
## iter  11 value 0.035007
## iter  12 value 0.035007
## iter  13 value 0.035007
## iter  14 value 0.035007
## iter  15 value 0.035007
## iter  15 value 0.035007
## final  value 0.035007
## converged
```

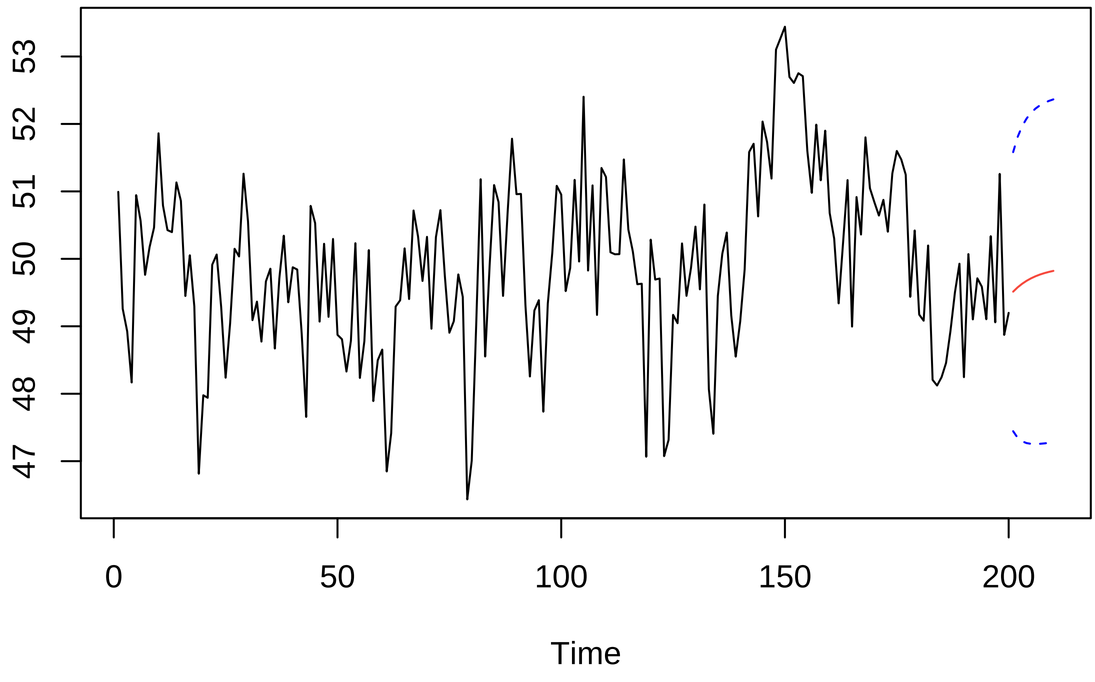



```
## $fit
##
## Call:
## arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D, Q), period = S),
##     xreg = xmean, include.mean = FALSE, transform.pars = trans, fixed = fixed,
##     optim.control = list(trace = trc, REPORT = 1, reltol = tol))
##
## Coefficients:
##          ar1      ma1    xmean
##       0.8340  -0.432  49.8962
## s.e.  0.0645   0.111   0.2452
##
## sigma^2 estimated as 1.07:  log likelihood = -290.79,  aic = 589.58
##
## $degrees_of_freedom
## [1] 197
##
## $ttable
##       Estimate     SE  t.value p.value
## ar1     0.8340 0.0645  12.9350   0e+00
## ma1    -0.4320 0.1110  -3.8904   1e-04
## xmean  49.8962 0.2452 203.5003   0e+00
```

```
##
## $AIC
## [1] 2.947891
##
## $AICc
## [1] 2.948503
##
## $BIC
## [1] 3.013857
```

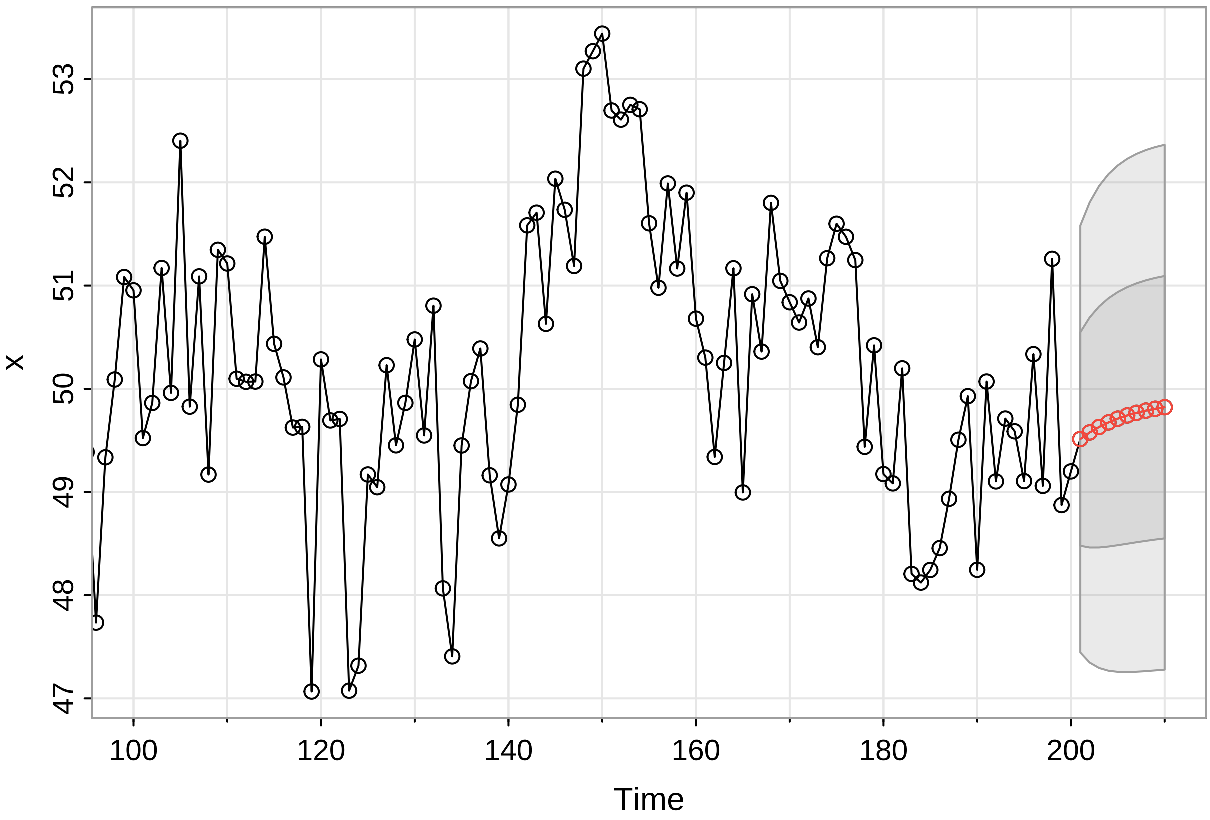
To obtain and plot the forecasts, see below

```
x.fore <- predict(x.fit, n.ahead = 10)
U <- x.fore$pred +2*x.fore$se # fore$pred for predicted values
L <- x.fore$pred -2*x.fore$se # fore$se for std. errors
miny <- min(x,L); maxy <- max(x, U)
ts.plot(x, x.fore$pred, col=1:2, ylim=c(miny, maxy))
lines(U, col="blue", lty="dashed")
lines(L, col="blue", lty="dashed")
```



Use `sarima.for` to make life easier:

```
sarima.for(x, 10, 1, 0, 1)
```




```
## $pred
## Time Series:
## Start = 201
## End = 210
## Frequency = 1
##  [1] 49.51320 49.57677 49.62978 49.67400 49.71088 49.74164 49.76729 49.78869
##  [9] 49.80653 49.82142
##
## $se
## Time Series:
## Start = 201
## End = 210
## Frequency = 1
##  [1] 1.034326 1.114795 1.167505 1.202811 1.226771 1.243167 1.254446 1.262232
##  [9] 1.267621 1.271356
```

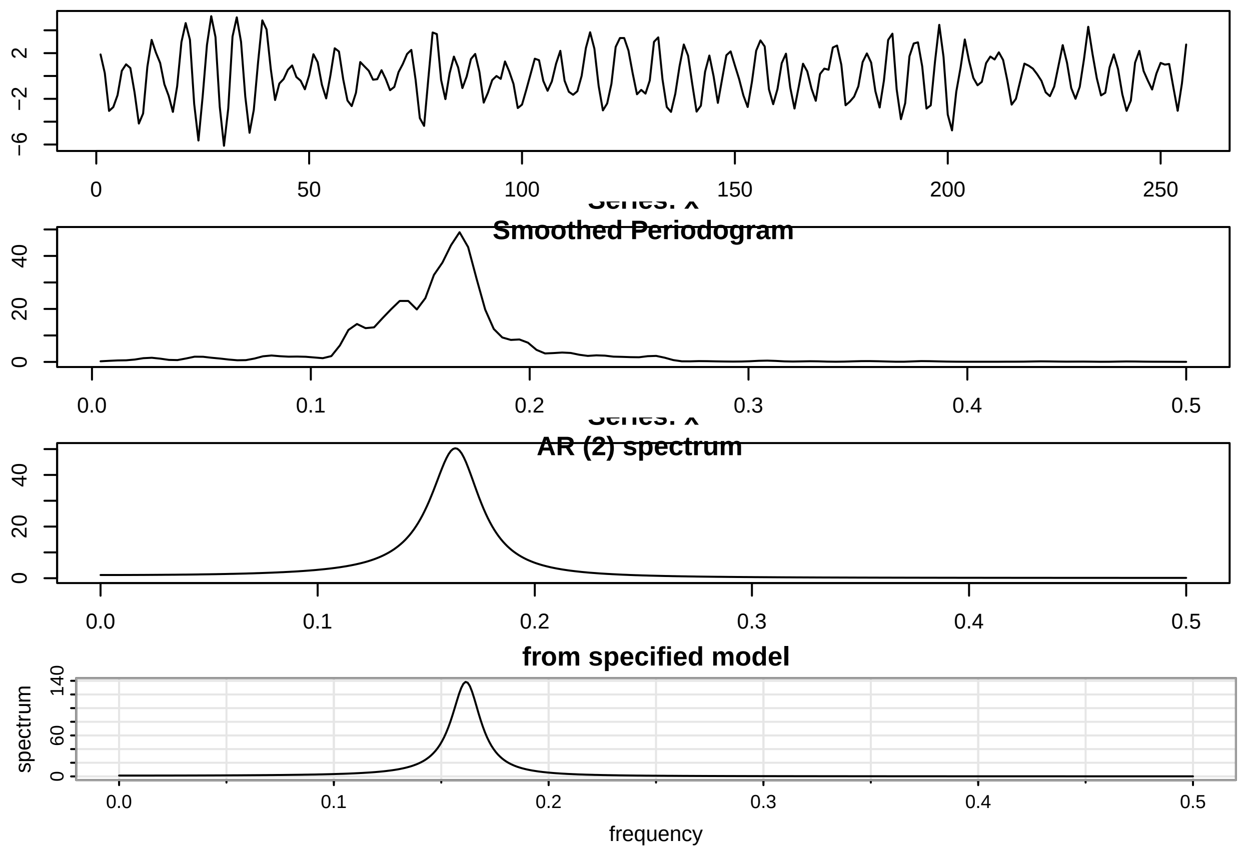
Lastly, we take a look at spectral analysis.

We will simulate AR(2) and them estimate the spectrum via nonparametic and parametic methods.

```
x <- arima.sim(list(order=c(2,0,0), ar=c(1, -.9)), n=2^8) #some data
u <- polyroot(c(1,-1,.9)) # x is AR(2) w/ cpx roots
Arg(u[1])/2*pi # dominant frequency around .16
```

```
## [1] 1.595419
```

```
par(mfcol=c(4,1))
par(mar=c(2,2,1,1))
plot.ts(x)
spec.pgram(x, spans = c(3,3), log="no") #nonparametric spectral estimate
spec.ar(x, log="no") # parametric spectral estimate
arma.spec(ar=c(1,-.9)) #true spectral density
```



See `spectrum` as an alternative to `spec.pgram`

Note that R tapers and logs by default, so if you simply want the periodogram of a series, the command is `spec.pgram(x, taper=0, fast=FALSE, detrend=FALSE, log="no")`

If you just asked for `spec.pgram(x)`, you would not get the RAW periodogram b/c the data are detrended, possibly padded, and tapered, even though the title of the resulting graphic would say *Raw Periodogram*.

An easier way to get a raw periodogram is

```
per <- abs(fft(x))^2/ length(x)
```