Time Series Analysis

Gavin Simpson

February 2017

1 Handling temporal data in R

In this section of the practical, you will learn to use some basic R code to produce temporal data objects in R.

The current date and time can be produce using the `Sys.Date()` and `Sys.time()` functions. Compute the current date and time and save these for use later

```
> today <- Sys.Date()
> today
```

```
[1] "2017-02-16"
```

```
> now <- Sys.time()
> now
```

```
[1] "2017-02-16 09:23:51 ACDT"
```

Sequences of dates and times can be produced using the `seq()` method. Produce a sequence of dates from today of length 20 incrementing two days at a time.

```
> dseq <- seq(today, by = "2 days", length = 20)
> dseq
```

```
 [1] "2017-02-16" "2017-02-18" "2017-02-20" "2017-02-22" "2017-02-24"
 [6] "2017-02-26" "2017-02-28" "2017-03-02" "2017-03-04" "2017-03-06"
[11] "2017-03-08" "2017-03-10" "2017-03-12" "2017-03-14" "2017-03-16"
[16] "2017-03-18" "2017-03-20" "2017-03-22" "2017-03-24" "2017-03-26"
```

Q: What is the last date in the sequence?

Q: Produce another series of dates of length 10, incrementing five days at a time. What is the 6th date in the sequence?

Q: Using the arithmetic operator +, how else might you generate the above sequences?

Now we will create a weekly time series of 100 observations from today

```
> dates <- seq(today, by = "1 week", length = 100)
```

Q: To what week of the year does the last observation belong? To help answer this, you will need the `format` function and to know that the place holder for week-of-the-year is `"%W"`. To extract the last date in the series you can use `dates[100]`. Refer to the lecture notes for help.

2 Mauna Loa CO_2

The first thing to do is to load the first data set we will use. This data set contains observations on the concentration of carbon dioxide (CO_2) in the atmosphere made at Mauna Loa. This is an in-built data set in R so can be loaded via the `data` function

```
> data(co2)
> class(co2)
```

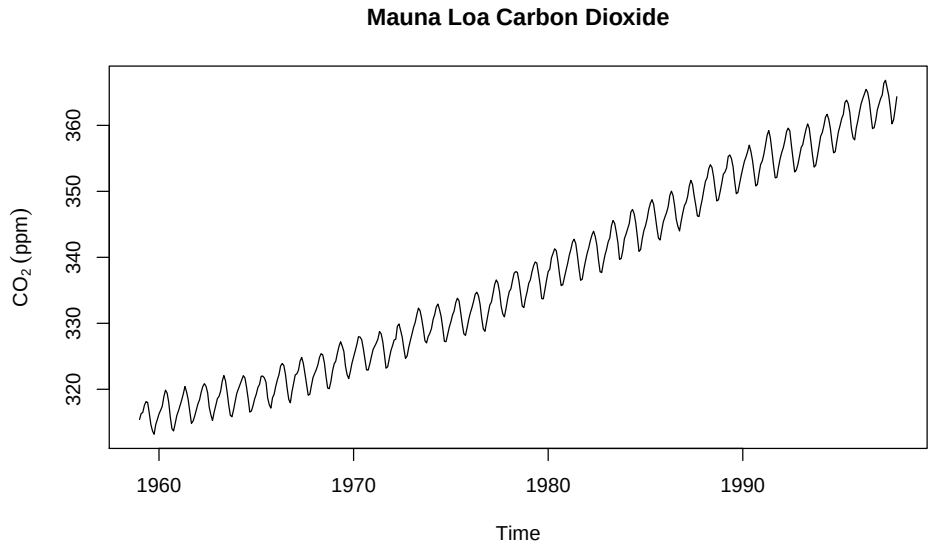

Figure 1: Mauna Loa CO_2 concentration 1959–1997

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

```
> tsp(co2)
```

```
[1] 1959.000 1997.917   12.000
```

Note the class of the `co2` object; it is a time series object of class `"ts"`. The `tsp()` function provides details on the time series properties of the data; showing the start and end dates (fractions of years) and the periodicity of the samples per time unit, which in this case is a year. We can plot the data using the `plot` method

```
> ylab <- expression(CO[2] ~ (ppm))
> plot(co2, ylab = ylab, main = "Mauna Loa Carbon Dioxide")
```

The series exhibit strong seasonality and an obvious trend.

We can use standard R functions to compute seasonal or annual averages to give gross descriptions of the series

```
> Month <- factor(rep(1:12, times = 39), labels = month.abb)
> Year <- factor(rep(1959:1997, each = 12))
> aggregate(as.numeric(co2), by = list(Month = Month), mean)
> aggregate(as.numeric(co2), by = list(Year = Year), mean)
```

Boxplots can also be draw to visualise patterns in the data

```
> layout(matrix(1:2, ncol = 2))
> boxplot(co2 ~ Month)
> boxplot(co2 ~ Year)
> layout(1)
```

Can you think why the seasonal averages and boxplots do not show the strong seasonal component exhibited in the raw data? Differencing is done using the `diff` function. To compute the first order differences for the co2 data and plot them, use

```
> co2.d <- diff(co2)
> plot(co2.d)
```

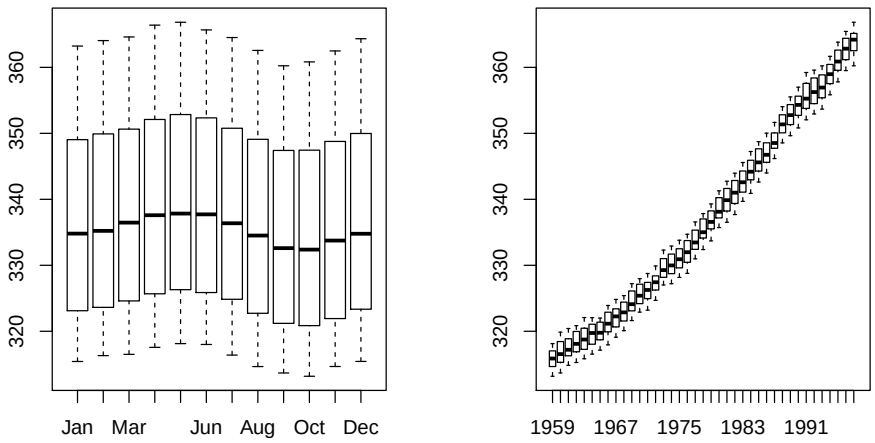

Figure 2: Boxplots of seasonal and annual averages of the Mauna Loa CO_2 concentration 1959–1997

We can decompose the time series into trend, seasonal and error components. Classical decomposition of time series is performed using the `decompose()` function

```
> co2.deco <- decompose(co2)
> str(co2.deco)
```

```
List of 6
 $ x       : Time-Series [1:468] from 1959 to 1998: 315 316 316 318 318 ...
 $ seasonal: Time-Series [1:468] from 1959 to 1998: -0.0536 0.6106 1.3756 2.5168 3.0003 ...
 $ trend   : Time-Series [1:468] from 1959 to 1998: NA NA NA NA NA ...
 $ random  : Time-Series [1:468] from 1959 to 1998: NA NA NA NA NA ...
 $ figure  : num [1:12] -0.0536 0.6106 1.3756 2.5168 3.0003 ...
 $ type    : chr "additive"
 - attr(*, "class")= chr "decomposed.ts"
```

Note that the returned object contains separate series for each of the fitted components. To visualise the decomposed time series, we build the total plot up from the individual components

```
> layout(matrix(1:4, ncol = 1))
> op <- par(oma = c(2,0,0,1) + 0.1, mar = c(2,4,2,2) + 0.1,
+           xpd = NA)
> plot(co2, xlab = "")
> plot(co2.deco$trend, xlab = "", ylab = "Trend")
> plot(co2.deco$seasonal, xlab = "", ylab = "Seasonal")
> plot(co2.deco$random, ylab = "Remainder")
> par(op)
> layout(1)
```

An alternative decomposition is possible using Loess via the STL method. In R, this technique is available in the `stl()` function. To compute the STL on the Mauna Loa data run the following

```
> co2.stl <- stl(co2, s.window = 13)
> plot(co2.stl)
```

To visualise how well the decompositions have performed we can compute the ACF of the two random components. If the random components are purely random processes then the ACF will show no significant autocorrelations

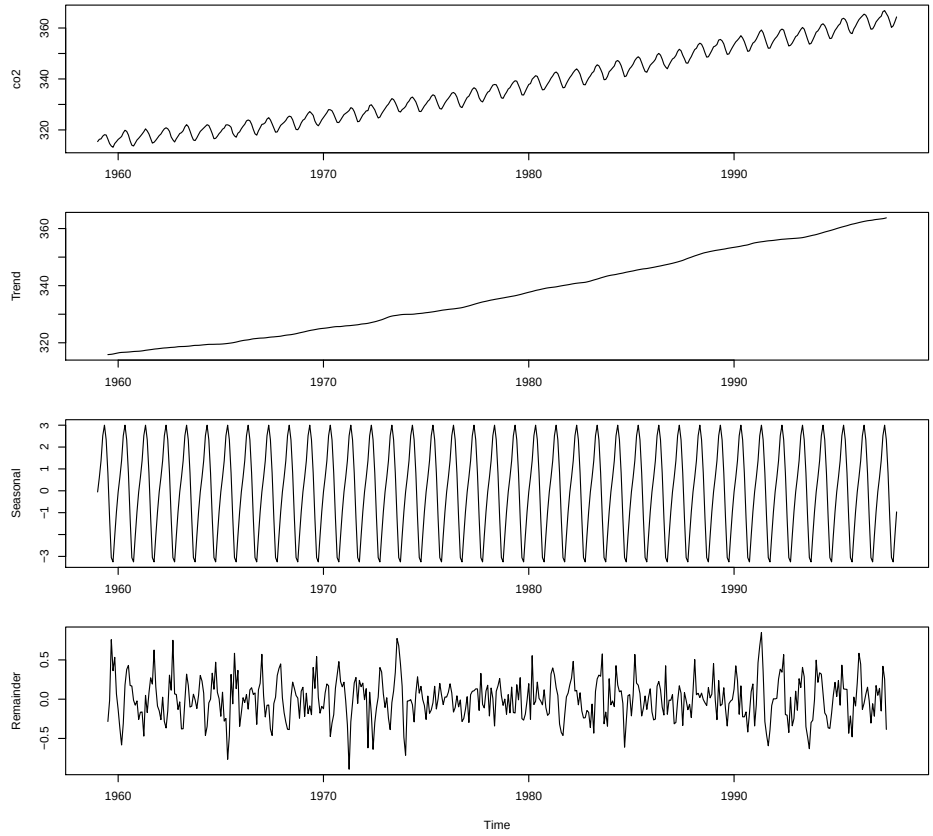

Figure 3: Decomposition of the Mauna Loa CO_2 concentration time series 1959–1997

```
> layout(matrix(1:2, ncol = 1))
> acf(co2.deco$random, na.action = na.omit, lag.max = 36)
> acf(co2.stl$time.series[,3], lag.max = 36)
> layout(1)
```

Compare the ACFs of the two decompositions. Which decomposition appears to have performed better? Can you account for this?

3 Fitting SARIMA models to data

We will now fit a SARIMA model to the Mauna Loa data; we will use the observations up to and including 1990 to build the model and subsequently predict for the remaining years. In order to select the most appropriate form for the SARIMA model, we fit several different combinations of model parameters and select the best fitting model using BIC. We will do this in several code chunks. The first selects out the observations up to the end of 1990 using the `window()` function, then computes all combinations of parameter sets (`CO2.pars`), and finally we create an object to hold the BIC values for each model (`CO2.bic`)

```
> CO2 <- window(co2, end = c(1990, 12))
> CO2.pars <- expand.grid(ar = 0:2, diff = 1, ma = 0:2,
+                         sar = 0:1, sdiff = 1, sma = 0:1)
> CO2.bic <- rep(0, nrow(CO2.pars))
```

We now use a loop to move over the parameter sets and fit a SARIMA model to them, extracting the BIC for the model. This step will take a minute or two as it is fitting 36 models.

```
> for(i in seq(along = CO2.bic)) {
+     CO2.bic[i] <- AIC(arima(CO2,
+                             unlist(CO2.pars[i, 1:3]),
+                             unlist(CO2.pars[i, 4:6])),
+                       k = log(length(CO2)))
+ }
```

To see which parameter set has lowest BIC, we find the minimum BIC value and use that to select out the parameter set for that BIC

```
> CO2.pars[which.min(CO2.bic), ]

   ar diff ma sar sdiff sma
22  0    1  1   0     1   1
```

In for abbreviated form SARIMA(p,d,q)(P,D,Q)s, write out the form of the fitted model with lowest BIC.

As we didn't save the model fits in the loop above, we need to refit the best model

```
> CO2.mod <- arima(CO2, order = c(0,1,1), seasonal = c(0,1,1))
> CO2.mod

Call:
arima(x = CO2, order = c(0, 1, 1), seasonal = c(0, 1, 1))

Coefficients:
          ma1     sma1
      -0.3605  -0.8609
s.e.   0.0545   0.0313

sigma^2 estimated as 0.08031:  log likelihood = -66.8,  aic = 139.61
```

Look at the printed output, what are the estimates for the SARIMA model parameters? Are these parameter estimates significantly different from zero?

R contains a nice diagnostics plotting function for time series, `tsdiag()`. We use this function now on the model residuals, which should be white noise.

```
> tsdiag(CO2.mod, gof.lag = 36)
```

The upper panel plots the residuals of the model fit, the middle panel shows the ACF for the residuals and the bottom panel shows p-values for the a test statistic of the autocorrelation of the residuals. Look at the two lower panels. Are there any problems in the model residuals?

If we are happy with the model, we can proceed to use the model to predict for the last seven years of observations not used in model fitting. To predict from the SARIMA model, we use the `predict()` function to compute the n-step ahead forecasts as their 95% confidence interval

```
> pred <- predict(CO2.mod, n.ahead = 7*12)
> upr <- pred$pred + (2 * pred$se)
> lwr <- pred$pred - (2 * pred$se)
```

To plot the predictions on the observed time series, we use the following code chunk

```
> ylim <- range(co2, upr, lwr)
> plot(co2, ylab = ylab, main = expression(bold(Mauna~Loa~CO[2])),
+      xlab = "Year", ylim = ylim)
> lines(pred$pred, col = "red")
> lines(upr, col = "red", lty = 2)
> lines(lwr, col = "red", lty = 2)
> legend("topleft", legend = c("Observed","Predicted","95% CI"),
+        col = c("black","red","red"), lty = c(1,1,2), bty = "n")
```

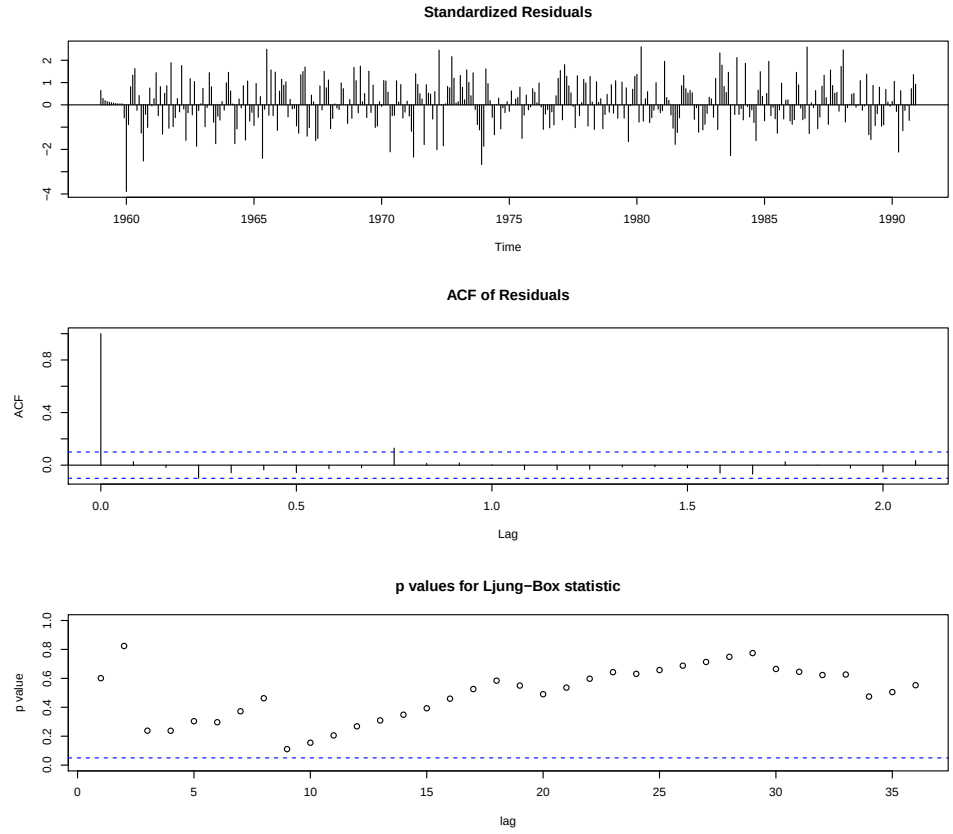

Figure 4: Diagnostics plots for the SARIMA model fit to the Mauna Loa CO_2 data

4 Regression models for time series

We will now look at fitting regression models to time series data, making use of autocorrelation functions for the residuals as well as using splines to model non-linear trends and seasonal components. We will continue to use the Mauna Loa CO_2 data as a simple example.

Fitting regression models to time series data involves setting up a few new variables within a data frame to hold information about sampling month, time since start of sampling etc. These variables will be used to describe trends and seasonal components in the regression models themselves.

```
> carbon <- data.frame(CO2 = as.numeric(co2),
+                      Month = rep(month.abb, 39),
+                      Year = rep(1959:1997, each = 12),
+                      Time = seq_len(length(co2)))
> carbon <- within(carbon,
+                  Date <- as.Date(paste("01", Month, Year),
+                                  format = "%d %b %Y"))
```

The first call above creates a new data frame with the observed data plus sampling month, year and time since sampling began. The second call creates a date variable to assist with plotting later.

We start by fitting a simple linear model including components for the trend and season

```
> mod <- lm(CO2 ~ Year + Month, data = carbon)
> summary(mod)

Call:
lm(formula = CO2 ~ Year + Month, data = carbon)
```

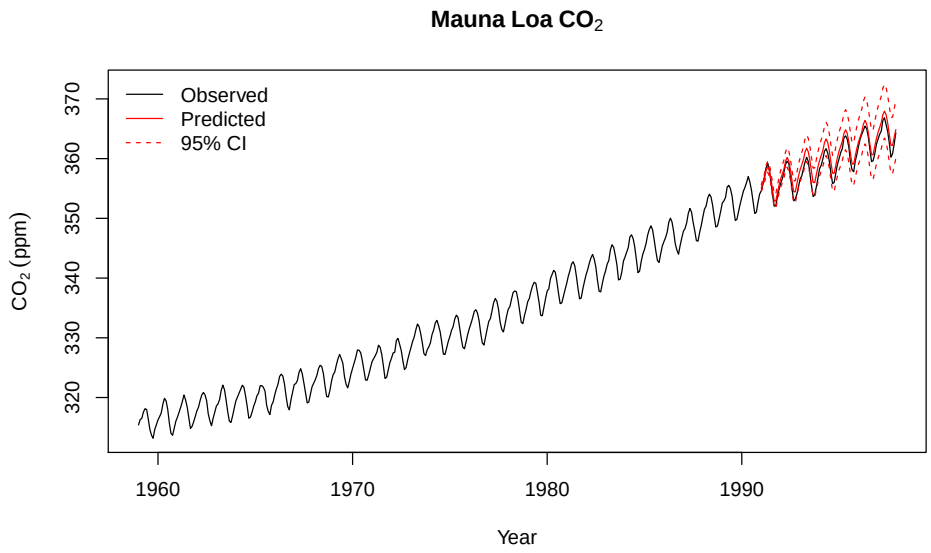

Figure 5: Observations and fitted values from the SARIMA model fitted to the Mauna Loa CO_2 data

```
Residuals:
   Min     1Q Median     3Q    Max
-2.768 -1.284 -0.405  1.261  4.337

Coefficients:
              Estimate Std. Error  t value Pr(>|t|)
(Intercept) -2.253e+03  1.330e+01 -169.446  < 2e-16 ***
Year         1.310e+00  6.720e-03  195.003  < 2e-16 ***
MonthAug    -3.336e+00  3.705e-01   -9.004  < 2e-16 ***
MonthDec    -2.600e+00  3.705e-01   -7.016 8.31e-12 ***
MonthFeb    -2.091e+00  3.705e-01   -5.643 2.94e-08 ***
MonthJan    -2.864e+00  3.705e-01   -7.728 7.04e-14 ***
MonthJul    -1.378e+00  3.705e-01   -3.719 0.000225 ***
MonthJun     3.385e-02  3.705e-01    0.091 0.927260
MonthMar    -1.240e+00  3.705e-01   -3.346 0.000888 ***
MonthMay     5.877e-01  3.705e-01    1.586 0.113421
MonthNov    -3.826e+00  3.705e-01  -10.327  < 2e-16 ***
MonthOct    -5.125e+00  3.705e-01  -13.832  < 2e-16 ***
MonthSep    -5.052e+00  3.705e-01  -13.633  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.636 on 455 degrees of freedom
Multiple R-squared:  0.9884,        Adjusted R-squared:  0.988
F-statistic:  3218 on 12 and 455 DF,  p-value: < 2.2e-16

> anova(mod)

Analysis of Variance Table

Response: CO2
           Df Sum Sq Mean Sq   F value    Pr(>F)
```

```
Year         1 101808  101808 38026.305 < 2.2e-16 ***
Month       11   1576     143    53.521 < 2.2e-16 ***
Residuals 455   1218       3
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Look at the model output. Are the terms significant? Is there anything that would give you cause for concern when interpreting the model output directly? To plot the observed data and fitted model, we use

```
> plot(CO2 ~ Date, data = carbon, type = "l")
> lines(fitted(mod) ~ Date, data = carbon, col = "red")
```

We note that the trend, being linear, fails to capture the changes in the rate of increase in atmospheric CO_2 conditions over time. We will return to this later when we fit a non-linear trend.

The linear model fitted does not account for the autocorrelation of the observations. There is still strong autocorrelation

```
> acf(resid(mod))
```

although a lot of this may be due simply to the poorly fitting trend. We can refit the model using GLS and assuming AR(1) errors. First load the nlme package which contains the functions we need

```
> require(nlme)
```

We fit a GLS model in the same way as a linear model, but we now specify an extra argument, `correlation`

```
> mod.gls <- gls(CO2 ~ Year + Month, data = carbon,
+                correlation = corAR1(form = ~ Time))
> summary(mod.gls)
> anova(mod.gls)
> plot(CO2 ~ Date, data = carbon, type = "l")
> lines(fitted(mod.gls) ~ Date, data = carbon, col = "red")
```

Look at the output and model fit. What is the value of the correlation parameter? Are all the predictors still significant? Are there problems with the fitted model?

We note that the model does not adequately fit the observed trend. We could allow non-linear trends by fitting polynomials in Time, but we will move on to fitting models that include smoothers to illustrate this more advanced technique. We being by loading the mgcv package, and add a numeric variable for the month of observation, which we need to fit a smoother to

```
> require(mgcv)
> carbon <- within(carbon,
+                  month <- as.numeric(format(Date,
+                                             format = "%m")))
```

We will fit a model that includes a smoother for the seasonal component, plus a smoother for the trend. We will also fit the model assuming AR(1) correlations in the model residuals

```
> mod.amm <- gamm(CO2 ~ s(Time, bs = "cr") + s(month, bs = "cc"),
+                 data = carbon,
+                 correlation = corAR1(form = ~ Time))
> summary(mod.amm$gam)

Family: gaussian
Link function: identity

Formula:
CO2 ~ s(Time, bs = "cr") + s(month, bs = "cc")
```

```
Parametric coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 337.05509    0.05146    6549   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
           edf Ref.df     F p-value
s(Time)  7.815  7.815 10797  <2e-16 ***
s(month) 7.877  8.000   559  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.999
  Scale est. = 0.24799   n = 468
```

```
> summary(mod.amm$lme)

Linear mixed-effects model fit by maximum likelihood
 Data: strip.offset(mf)
       AIC      BIC    logLik
  475.7256 500.6165 -231.8628

Random effects:
 Formula: ~Xr - 1 | g
 Structure: pdIdnot
              Xr1       Xr2       Xr3       Xr4       Xr5       Xr6       Xr7
StdDev: 0.3472701 0.3472701 0.3472701 0.3472701 0.3472701 0.3472701 0.3472701
              Xr8
StdDev: 0.3472701

 Formula: ~Xr.0 - 1 | g.0 %in% g
 Structure: pdIdnot
           Xr.01    Xr.02    Xr.03    Xr.04    Xr.05    Xr.06    Xr.07    Xr.08
StdDev: 0.333099 0.333099 0.333099 0.333099 0.333099 0.333099 0.333099 0.333099
         Residual
StdDev: 0.4979896

Correlation Structure: AR(1)
 Formula: ~Time | g/g.0
 Parameter estimate(s):
      Phi
0.6680634
Fixed effects: y ~ X - 1
                Value  Std.Error  DF  t-value p-value
X(Intercept) 337.0551 0.05151842 466 6542.419       0
Xs(Time)Fx1  -50.8962 0.20647850 466 -246.497       0
 Correlation:
            X(Int)
Xs(Time)Fx1 0

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-2.80098711 -0.65032451 -0.03941756  0.62016244  2.85509750

Number of Observations: 468
```

```
Number of Groups:
         g g.0 %in% g
         1          1
```

The model is similar to the GLS one specified above, but now we use cubic regression splines for the two predictor variables. Note the `"cc"` in the smoother for `month`; this specifies a special type of smoother, a cyclic smoother, which is one where the two end points join so there is no discontinuity between Jan and Dec.

How many degrees of freedom are used by each of the two smooth terms?

```
> intervals(mod.amm$lme)

Approximate 95% confidence intervals

 Fixed effects:
                 lower      est.     upper
X(Intercept) 336.95407 337.05509 337.15611
Xs(Time)Fx1  -51.30112 -50.89624 -50.49137
attr(,"label")
[1] "Fixed effects:"

 Random Effects:
  Level: g
                lower      est.     upper
sd(Xr - 1) 0.02398623 0.1205965 0.6117784
  Level: g.0
                  lower     est.     upper
sd(Xr.0 - 1) 0.04091911 0.110955 0.3019294

 Correlation structure:
        lower      est.     upper
Phi 0.5680537 0.6680634 0.7486295
attr(,"label")
[1] "Correlation structure:"

 Within-group standard error:
    lower      est.     upper
0.4356838 0.4979896 0.5692056
```

Look for the correlation parameter. What is the estimate correlation and it's confidence interval?

We can plot the estimated smooth functions

```
> plot(mod.amm$gam, pages = 1, scale = 0)
```

and the fitted values from the model

```
> plot(CO2 ~ Date, data = carbon, type = "l")
> lines(fitted(mod.amm$lme) ~ Date, data = carbon, col = "red")
```

We can formally test if the AR(1) structure is required by fitting a model without the structure and comparing the two models with a likelihood ratio test

```
> mod2.amm <- gamm(CO2 ~ s(Time, bs = "cr") + s(month, bs = "cc"),
+                  data = carbon)
> anova(mod.amm$lme, mod2.amm$lme)

             Model df      AIC      BIC    logLik   Test  L.Ratio p-value
mod.amm$lme      1  6 475.7256 500.6165 -231.8628
mod2.amm$lme     2  5 667.4271 688.1695 -328.7136 1 vs 2 193.7015  <.0001
```

Look the the information statistics and the output of the LRT. Which of the models fits the data best?

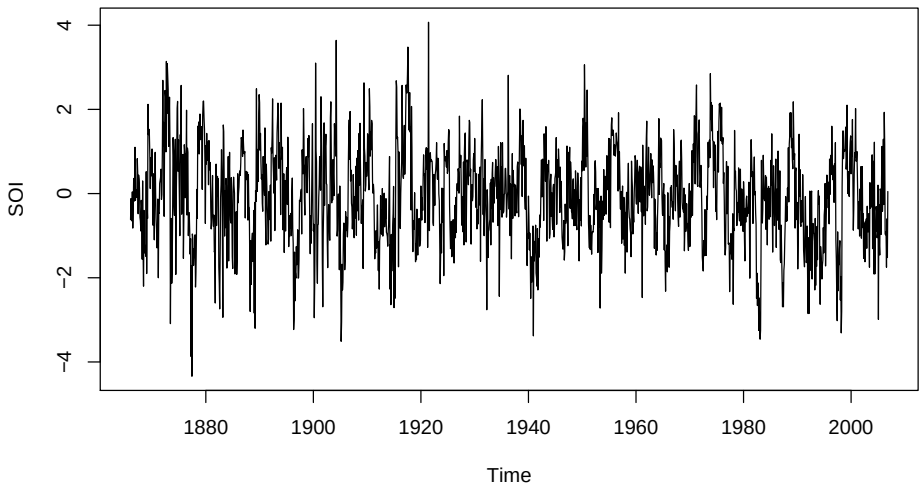

Figure 6: Southern Oscillation index 1866–2006

5 Spectral Analysis

To illustrate some aspects of spectral analysis we turn to a data set of observations on the *Southern Oscillation Index* (SOI), which is defined as the normalised pressure difference between Tahiti and Darwin. El Niño events occur when the SOI is strongly negative. Monthly SOI time series data from January 1866 to the end of November 2006 are stored in the data set `soi.txt` which can be downloaded from the web. These data can easily be converted to a `"ts"` object by specifying the appropriate time series properties.

```
> ## read in SOI data
> #baseurl <- "http://www.ucl.ac.uk/~ucfagls/teaching/ceh/"
> #soi <- read.table(paste(baseurl, "soi.txt", sep = ""), header = TRUE)
> soi <- read.table("soi.txt", header = TRUE)
> head(soi)

    SOI
1 -0.62
2 -0.12
3 -0.62
4 -0.65
5  0.04
6 -0.82

> ## convert to ts class - data start jan 1866, finish november 2006
> soi.ts <- with(soi, ts(SOI, start = c(1866, 1), end = c(2006, 11),
+                        frequency = 12))
```

A plot of the data can be produced using the `plot()` method, the resulting plot is shown in 6:

```
> plot(soi.ts, ylab = "SOI")
```

The general purpose spectral analysis function in standard R is `spectrum()`. It can generate both raw and smooth periodograms via a number of methods, the default of which produced the raw periodogram. For the SOI time series the spectrum is produced and plotted via

```
> soi.raw <- spectrum(soi.ts)
```

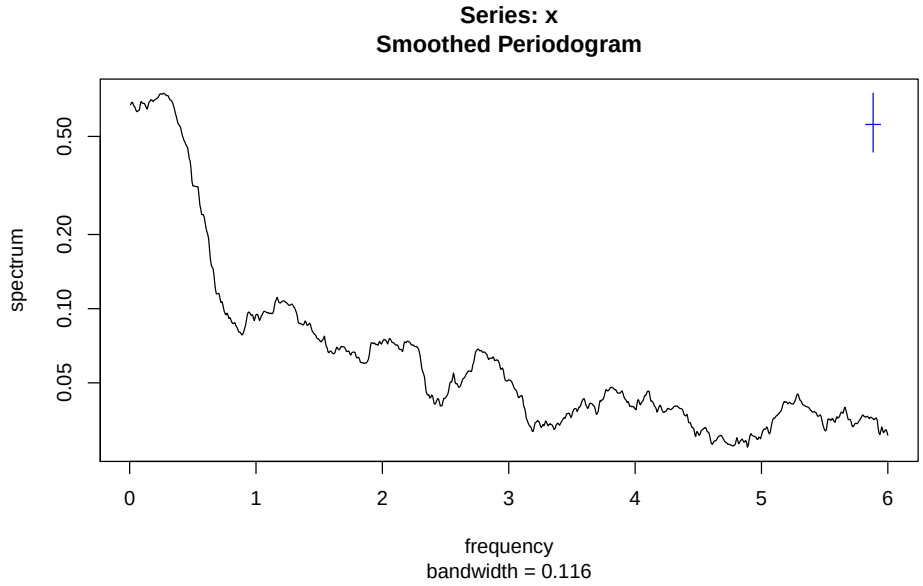

Figure 7: Smoothed periodogram of the Southern Oscillation index 1866–2006

As we might suspect, the raw periodogram is difficult to interpret and requires some amount of smoothing to discern pattern in the spikes. A useful rule of thumb for time series up to approximately 1000 observations is to use as the smoothing span the value that is $\sqrt{2 \times n}$ where n is the number of observations.

```
> soi.smo <- spectrum(soi.ts, spans = sqrt(2 * length(soi.ts)))
```

The resulting plot is shown in Figure 7. Low frequency variation dominates the periodogram, with a small peak at around 0.27 cycles per year (approximately 45 months). You can zoom into the part of the periodogram of interest using

```
> plot(soi.smo$freq[1:60], soi.smo$spec[1:60], type = "l", log = "y")
```

In both figures, the x-axis is in units of cycles per year. To convert this into the cycle length in months, we first compute the cycle length in years ($1/0.27 = 3.7$) then multiply this by 12 months, which gives 44.44 months.

To compute the spectrum using the AR(p) algorithm, all that is required is to provide the `method = "ar"` argument to `spectrum()` and there is no need for smoothing

```
> soi.smo <- spectrum(soi.ts, method = "ar")
```

Q: How does the AR(p) estimation compare with the smoother periodogram? How many AR terms were used to fit the periodogram.

5.1 Spectral Analysis of the Central England Temperature series

If you have time, load in a set of data from the Central England Temperature series, 1659–2009, and compute the power spectrum using the smoothed periodogram and the AR(p) methods. What frequencies and periods of variation appear important in the power spectrum?

To load the data use:

```
> cet <- read.csv("monthly_cet_mean_temp_1659-2009.csv",
+                 na.strings = -99.9, row.names = 1)
> head(cet, n = 4)
```

```
     JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
1659   3   4   6   7  11  13  16  16  13  10   5   2
1660   0   4   6   9  11  14  15  16  13  10   6   5
1661   5   5   6   8  11  14  15  15  13  11   8   6
1662   5   6   6   8  11  15  15  15  13  11   6   3
```

```
> tail(cet, n = 4)
```

```
     JAN FEB MAR  APR  MAY  JUN  JUL  AUG  SEP  OCT NOV DEC
2006 4.3 3.7 4.9  8.6 12.3 15.9 19.7 16.1 16.8 13.0 8.1 6.5
2007 7.0 5.8 7.2 11.2 11.9 15.1 15.2 15.4 13.8 10.9 7.3 4.9
2008 6.6 5.4 6.1  7.9 13.4 13.9 16.2 16.2 13.5  9.7 7.0 3.5
2009 3.0 4.1 7.0 10.0   NA   NA   NA   NA   NA   NA  NA  NA
```

A little manipulation is required to convert this data set to a `"ts"` object, by transposing it and dropping the `NA` values.

```
> tmp <- t(cet)
> cet.ts <- ts(tmp[!is.na(tmp)], start = c(1659, 1), frequency = 12)
> rm(tmp)
```

Plot the time series. Describe the patterns you see

```
> plot(cet.ts)
```

The smoothed periodogram can be computed using

```
> cet.smo <- spectrum(cet.ts, span = 10)
```

Look at the resulting plot, what frequencies/periods appear important? Try different values for `span` to see what effect this has on the periodogram.

The AR(p) algorithm can be used to compute the periodogram with the following code

```
> cet.ar <- spectrum(cet.ts, method = "ar")
```

6 Wavelets

There are a number of packages available now for R that implement wavelet-based analyses. Of particular relevance is the biwavelet package. It is a port for two MATLAB packages popular in the geosciences; the WTC package of Alex Grinsted, and the wavelet package of Christopher Torrence & Gilbert Compo.

Load the biwavelet package. The basic wavelet function is `wt()` which takes a two-column matrix-like object; the first column is the temporal information, with the second column cotaining the time series observations themselves. To facilitate this we do a bit of data processing of the SOI data set

```
> require(biwavelet)
> ## data start jan 1866, finish november 2006
> start <- as.Date("1866-01-01")
> end <- as.Date("2006-11-01")
> soi.dates <- seq(start, end, by = "month")
> soi2 <- soi[, 1]
```

We pass these objects as a matrix-like object to `wt()`

```
> soi.wt <- wt(cbind(as.numeric(soi.dates), soi2))
```

The `plot()` method is a little rough-n-ready at the moment — you need to create some extra space around the plot to hold the colourbar scale for the wavelet power.

```
> op <- par(oma=c(0, 0, 0, 1), mar=c(5, 4, 4, 5) + 0.1)
> plot(soi.wt, plot.cb = TRUE, plot.phase = FALSE, lwd.sig = 1)
> par(op)
```