Class 1: Modelling with seasonality and the frequency domain

Andrew Parnell
andrew.parnell@ucd.ie

Learning outcomes

- Understand how to fit seasonal models in `forecast` and JAGS
- Understand seasonal differencing and sARIMA models
- Know the difference between time and frequency domain models and be able to implement a basic Fourier model
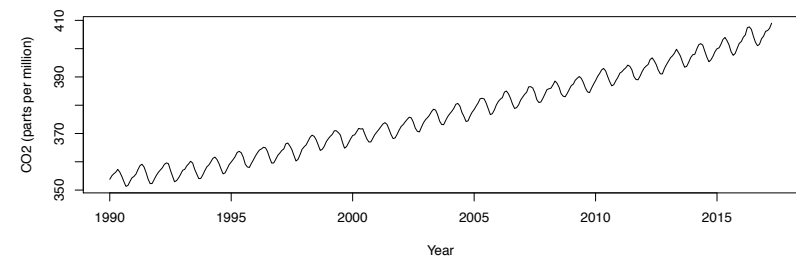
Seasonal time series

- So far we haven't covered how to deal with data that are *seasonal* in nature
- These data generally fall into two categories:
  1. Data where we know the frequency or frequencies (e.g. monthly data on a yearly cycle, frequency = 12)
  2. Data where we want to estimate the frequencies (e.g. climate time series, animal populations, etc)
- The former are easier, and there are many techniques for inducing seasonal behaviour
- The latter are much more interesting. The ACF and PACF can help, but we can usually do much better by creating a *power spectrum*
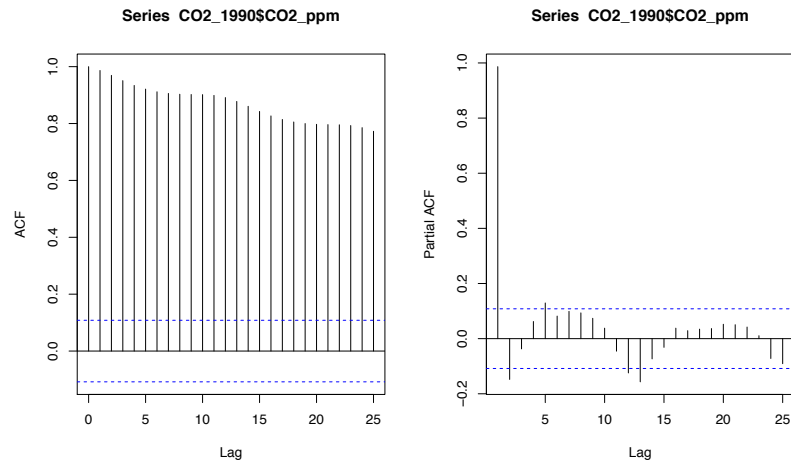
An example seasonal series

```
CO2 = read.csv(file = '../../data/CO2.csv',
               na.strings = -99.99)
CO2$date = as.Date(paste(CO2$year, CO2$month ,'01',
                   sep = '-'))
CO2_1990 = CO2[CO2$year >= 1990, ]
with(CO2_1990, plot(date, CO2_ppm, type = 'l',
               ylab = 'CO2 (parts per million)',
               xlab = 'Year'), las = 1)
```

ACF and PACF

```r
par(mfrow = c(1, 2))
acf(CO2_1990$CO2_ppm)
pacf(CO2_1990$CO2_ppm)
```


Series CO2_1990$CO2_ppm

Series CO2_1990$CO2_ppm

Seasonal time series 1: including seasonality as a covariate

- ▶ The simplest way is to include month as a covariate in a regression type model

```r
CO2_1990$mfac = model.matrix(~ as.factor(CO2_1990$month) - 1)
colnames(CO2_1990$mfac) = month.abb
lm(CO2_ppm ~ year + mfac, data = CO2_1990)
```

```
##
## Call:
## lm(formula = CO2_ppm ~ year + mfac, data = CO2_1990)
##
## Coefficients:
## (Intercept)          year        mfacJan
##  -3501.5897        1.9362        -0.6749
##      mfacFeb        mfacMar        mfacApr
##       0.1222        1.0272         2.3590
##      mfacMay        mfacJun        mfacJul
##       2.7967        2.1367         0.5126
##      mfacAug        mfacSep        mfacOct
##      -1.5704       -3.0774        -2.8770
```

Forecasts

```r
CO2_ts = ts(CO2_1990$CO2_ppm, frequency = 12,
            start = c(1990, 1))
s_model_1 = tslm(CO2_ts ~ trend + season)
plot(forecast(s_model_1, h = 24))
```


Forecasts from Linear regression model
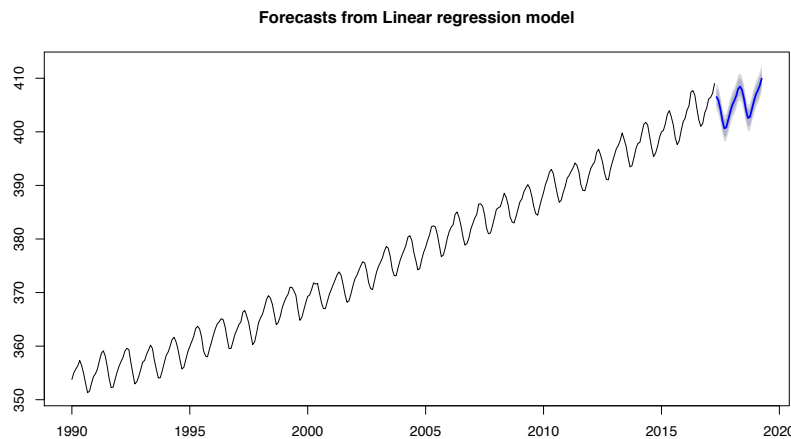
What is the time series model doing here?

- ▶ This is just a regression model, so that:

$$y_t = \alpha + \beta\text{year}_t + \gamma_1\text{Feb}_t + \gamma_2\text{Mar}_t + \ldots + \gamma_{11}\text{Dec}_t + \epsilon_t$$

- ▶ You can do this using `lm` or using `forecast`'s special function for linear regression forecasting `tslm`
- ▶ The `tslm` function is clever because it can automatically create the seasonal indicator variables
- ▶ Remember that when dealing with indicator variables you have to drop one factor level for the model to fit

Seasonal time series 2: seasonal differencing

- ▶ We have already met methods which difference the data (possibly multiple times) at lag 1
- ▶ We can alternatively create a seasonal difference by differencing every e.g. 12th observation
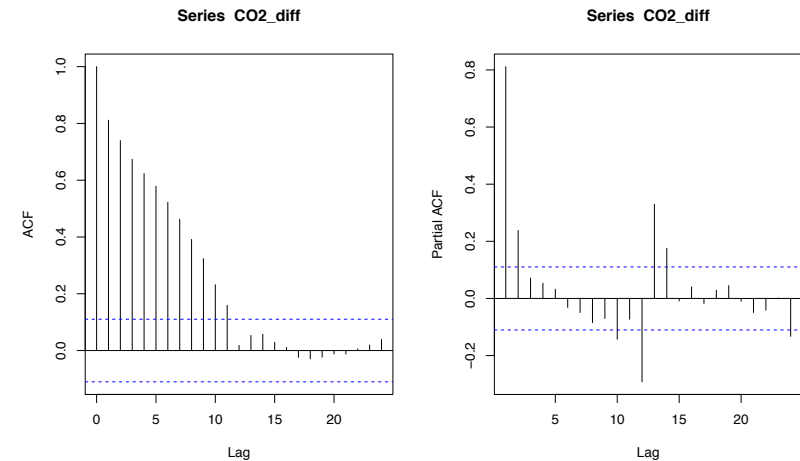
```
CO2_diff = diff(CO2_1990$CO2_ppm, lag = 12)
plot(CO2_diff, type = 'l')
```
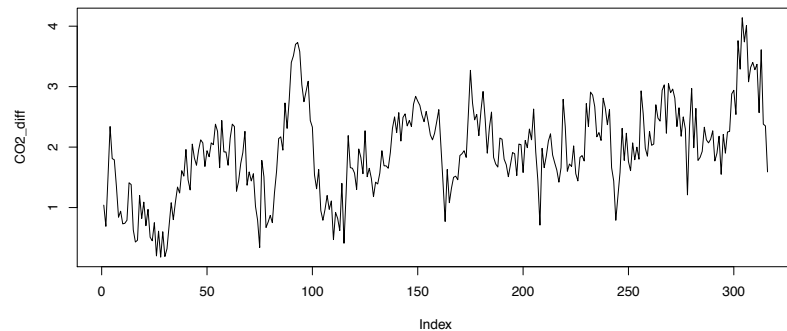
Differenced acf and pacf

```
par(mfrow = c(1, 2))
acf(CO2_diff, na.action = na.pass)
pacf(CO2_diff, na.action = na.pass)
```

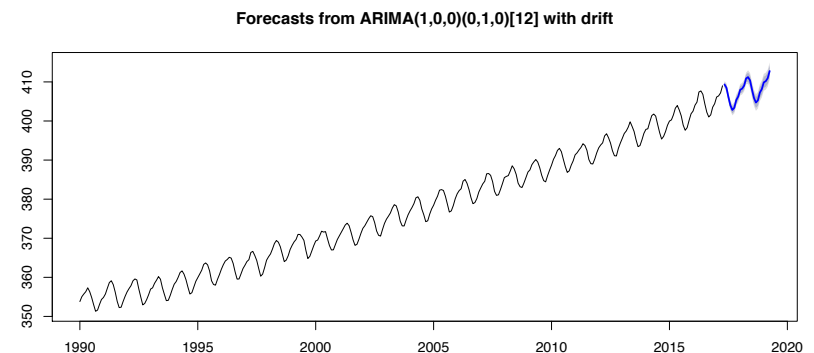## Fit an ARIMA model with a seasonal difference

```
CO2_1990_ts = ts(CO2_1990$CO2_ppm, frequency = 12,
                 start =c(1990, 1))
Arima(CO2_1990_ts, order = c(1, 0, 0),
      seasonal = c(0, 1, 0),
      include.drift = TRUE)
```

```
## Series: CO2_1990_ts
## ARIMA(1,0,0)(0,1,0)[12] with drift
##
## Coefficients:
##          ar1    drift
##       0.8129   0.1588
## s.e.  0.0325   0.0107
##
## sigma^2 estimated as 0.1943:  log likelihood=-185.12
## AIC=376.23   AICc=376.31   BIC=387.5
```

Forecasts from seasonally differenced series

```
s_model_2 = Arima(CO2_1990_ts, order = c(1, 0, 0),
                  seasonal = c(0, 1, 0), include.drift = TR
plot(forecast(s_model_2, h = 24))
```


- ▶ Pretty good. Might be able to do better with some richer models

A full seasonal arima model

- ▶ We previously met the ARIMA specification where:

$$\text{diff}^d(y_t) = \text{constant} + \text{AR terms} + \text{MA terms} + \text{error}$$

- ▶ We can extend this to include seasonal differencing and *seasonal AR and MA* terms to create a seasonal ARIMA or sARIMA model
- ▶ For example:

$$y_t - y_{t-12} = \alpha + \beta y_{t-1} + \gamma y_{t-12} + \epsilon_t$$

- ▶ This is a sARIMA$(1, 0, 0)(1, 1, 0)_{12}$ model

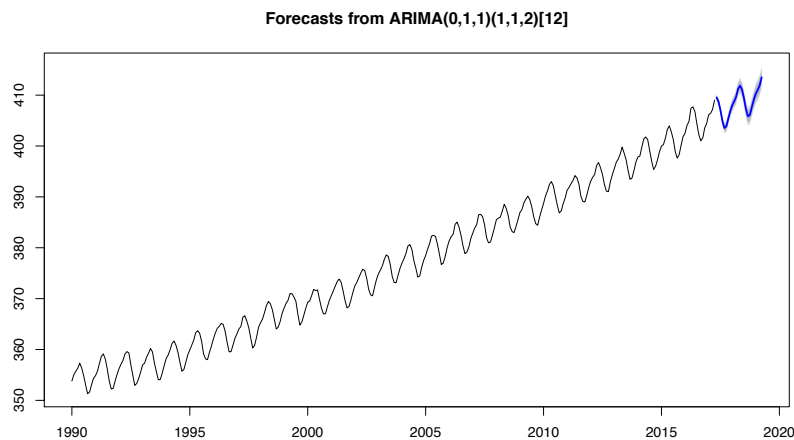## Fitting sARIMA models in `forecast`

```
auto.arima(CO2_1990_ts)
```

```
## Series: CO2_1990_ts
## ARIMA(0,1,1)(1,1,2)[12]
##
## Coefficients:
##           ma1      sar1     sma1     sma2
##       -0.3888  -0.7684  -0.1020  -0.6482
## s.e.   0.0582   0.4837   0.4891   0.4246
##
## sigma^2 estimated as 0.1137:  log likelihood=-103.27
## AIC=216.55    AICc=216.74    BIC=235.31
```

Plotting forecasts

```
s_model_3 = auto.arima(CO2_1990_ts)
plot(forecast(s_model_3, h = 24))
```


Forecasts from ARIMA(0,1,1)(1,1,2)[12]

A simple sARIMA model with JAGS

```
model_code = '
model
{
  # Likelihood
  for (t in (s+1):T) {
    y[t] ~ dnorm(mu[t], sigma^-2)
    mu[t] <- alpha + beta * y[t-1] + gamma * y[t-s]
  }

  # Priors
  alpha ~ dnorm(0, 10^-2)
  beta ~ dnorm(0, 10^-2)
  gamma ~ dnorm(0, 10^-2)
  sigma ~ dunif(0, 100)
}
'
```

Fitting a sARIMA$(1,0,0)(1,0,0)_{12}$ model in JAGS

```
s_model_4 = jags(data = list(y = CO2_ts, s = 12,
                             T = length(CO2_ts)),
                 parameters.to.save = c('alpha', 'beta',
                                        'gamma', 'sigma'),
                 model.file=textConnection(model_code))
```

```
print(s_model_4)
```

```
## Inference for Bugs model at "5", fit using jags,
##  3 chains, each with 2000 iterations (first 1000 discarded)
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%
## alpha     -6.443   0.861  -8.152  -7.038  -6.442
## beta       0.188   0.025   0.140   0.171   0.189
## gamma      0.833   0.025   0.784   0.816   0.832
## sigma      0.600   0.024   0.557   0.584   0.599
## deviance 571.698   2.815 568.192 569.609 571.054
##             75%   97.5%  Rhat n.eff
## alpha    -5.842  -4.787 1.001  3000
## beta      0.205   0.236 1.001  3000
## gamma     0.850   0.883 1.001  3000
## sigma     0.616   0.650 1.001  3000
## deviance 573.083 578.618 1.001  3000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 4.0 and DIC = 575.7
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Multiple seasonality

- Very occasionally you come across multiple seasonality models
- For example you might have hourly data over several months with both hourly and monthly seasonality
- `forecast` has a special function for creating multiple series time series: `msts`

```
x = msts(taylor, seasonal.periods=c(48,336),
         start=2000+22/52)
```

- The above is half-hourly data so has period 48 hours and 336 hours, i.e. weekly ($336/48 = 7$)
- `forecast` has some special functions (notably `tbats`) for modelling multi seasonality data

Frequency estimation

Methods for estimating frequencies

- The most common way to estimate the frequencies in a time series is to decompose it in a *Fourier Series*
- We write:

$$y_t = \alpha + \sum_{k=1}^{K} [\beta_k \sin(2\pi t f_k) + \gamma_k \cos(2\pi t f_k)] + \epsilon_t$$

- Each one of the terms inside the sum is called a *harmonic*. We decompose the series into a sum of sine and cosine waves rather than with AR and MA components
- Each sine/cosine pair has its own frequency f_k. If the corresponding coefficients β_k and γ_k are large we might believe this frequency is important

Estimating frequencies via a Fourier model

- It's certainly possible to fit the model in the previous slide in JAGS, as it's just a linear regression model with clever explanatory variables
- However, it can be quite slow to fit and, if the number of frequencies K is high, or the frequencies are close together, it can struggle to converge
- More commonly, people repeatedly fit the simpler model:

$$y_t = \alpha + \beta \sin(2\pi t f_k) + \gamma \cos(2\pi t f_k) + \epsilon_t$$

  for lots of different values of f_k. Then calculate the *power spectrum* as $P(f_k) = \frac{\beta^2 + \gamma^2}{2}$. Large values of the power spectrum indicate important frequencies
- It's much faster to do this outside of JAGS, using other methods, but we will stick to JAGS

JAGS code for a Fourier model
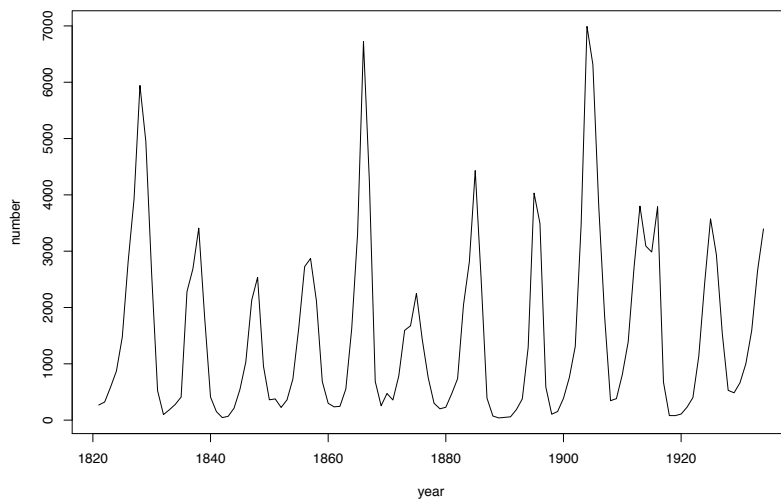
```
model_code = '
model
{
  # Likelihood
  for (t in 1:T) {
    y[t] ~ dnorm(mu[t], sigma^-2)
    mu[t] <- alpha + beta * cos(2*pi*t*f_k) +
                gamma * sin(2*pi*t*f_k )
  }
  P = (pow(beta, 2) + pow(gamma, 2)) / 2

  # Priors
  alpha ~ dnorm(0, 10^-2)
  beta ~ dnorm(0, 10^-2)
  gamma ~ dnorm(0, 10^-2)
  sigma ~ dunif(0, 100)
}
'
```

Example: the Lynx data

```
lynx = read.csv('../../data/lynx.csv')
plot(lynx, type = 'l')
```

Code to run the JAGS model repeatedly

```
periods = 5:40
K = length(periods)
f = 1/periods
Power = rep(NA,K)

for (k in 1:K) {
  curr_model_data = list(y = as.vector(lynx[,2]),
                         T = nrow(lynx),
                         f_k = f[k],
                         pi = pi)

  model_run = jags(data = curr_model_data,
                   parameters.to.save = "P",
                   model.file=textConnection(model_code))

  Power[k] = mean(model_run$BUGSoutput$sims.list$P)
}
```
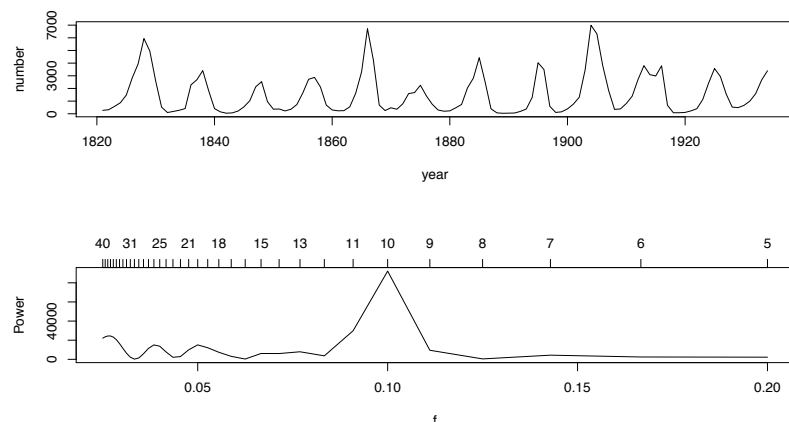
Plotting the periodogram

```r
par(mfrow = c(2, 1))
plot(lynx, type = 'l')
plot(f, Power, type='l')
axis(side = 3, at = f, labels = periods)
```

Bayesian vs traditional frequency analysis

- ▶ For quick and dirty analysis, there is no need to run the full Bayesian model, the R function `periodogram` in the TSA package will do the job, or `findfrequency` in `forecast` which is even simpler
- ▶ However, the big advantage (as always with Bayes) is that we can also plot the uncertainty in the periodogram, or combine the Fourier model with other modelling ideas (e.g. ARIMA)
- ▶ There are much fancier versions of frequency models out there (e.g. Wavelets, or frequency selection models) which can also be fitted in JAGS but require a bit more time and effort
- ▶ These Fourier models work for continuous time series too

Summary

- ▶ We now know how to fit models for seasonal data via seasonal factors, seasonal differencing, and sARIMA models
- ▶ We can fit these using `forecast` or JAGS
- ▶ We've seen a basic Fourier model for estimating frequencies via the Bayesian periodogram