Spatial and temporal statistics

Darren Wilkinson

12 September, 2025

Table of contents

Preface

These notes support the module **MATH4341: Spatio-temporal statistics IV**. Term 1 (Michaelmas) will concentrate on temporal modelling and the analysis of time series data. Term 2 (Epiphany) will be primarily concerned with spatial modelling and spatial statistics. We will look briefly at modelling fully spatio-temporal data at the end of the course.

- These notes: https://darrenjw.github.io/spatio-temporal/
- Also available as a PDF document (better for printing and annotating).

Michaelmas term

Part 1: Introduction to time series and linear filters

- Chapter 1 Introduction

Part 2: Linear stochastic systems

- Chapter 2 Linear systems
- Chapter 3 ARMA models
- Chapter 4 Forecasting and estimation
- Chapter 5 Spectral analysis

Part 3: State space modelling

- Chapter 6 HMMs
- Chapter 7 DLMs
- Chapter 8 State space modelling

Part 4: Introduction to spatial data

- Chapter 9 Introduction to spatial data

Epiphany term

Please note that although the lecture notes for Michaelmas term are essentially complete, the notes for Epiphany term are currently being written, and will not be complete until January 2026.

Part 5: Geostatistics

- Chapter 10 Random fields
- Chapter 11 GPs
- Chapter 12 Spectral theory for GPs
- Chapter 13 Kriging

Part 6: Lattice models

- Chapter 14 Lattice random fields
- Chapter 15 Spatial auto-regressive models
- Chapter 16 Inference for lattice models

Part 7: Spatial point processes

- Chapter 17 Spatial point processes

Part 8: Spatio-temporal modelling

- Chapter 18 Spatio-temporal models and data

Each chapter corresponds (very roughly) to one week (two lectures) of material.

Reading list and other resources

The main recommended text for Michaelmas term is Shumway and Stoffer (2017). This is an excellent reference, but has a different style to this course and covers more/different material, in a different order, and typically in more depth. After that, there are many excellent texts on time series analysis, including some classics. Chatfield and Xing (2019) is a good introductory text. Priestley (1989) is the classic text for spectral analysis. West and Harrison (2013) is the classic reference for DLMs, but Petris, Petrone, and Campagnoli (2009) and Särkkä and Svensson (2023) are interesting additions/alternatives.

The main recommended text for Epiphany term is Cressie (2015), but this classic now seems a little old-fashioned and is not to everyone's taste. Ripley (2005) is more approachable, but also quite dated. Gelfand et al. (2010) is a more modern overview of the field. Rasmussen and Williams (2005) and Rue and Held (2005) are also useful for parts of the course.

Wikle, Zammit-Mangion, and Cressie (2019) will be useful for the final part of the course and Cressie and Wikle (2015) is a more substantial reference for this part of the course.

Of course, in addition to traditional textbooks, there is a lot of good material available freely online (including wikipedia). I'll attempt to link to some of this from appropriate points in the notes.

We will use **R** for all of the computational examples. Some CRAN packages will be used. I will attempt to keep a full list of required installs here (but we will also use some of the required dependencies of these packages).

Michaelmas term:

```
install.packages(c(
  "astsa", "signal", "netcontrol", "dlm", "mvtnorm"
))
```

Additional packages required for Epiphany term:

```
install.packages(c(
  "sf", "sp", "spData", "spdep", "imager", "spatstat",
  "osmdata", "ggplot2", "spTimer"
))
```

All of the examples, figures and simulations in the notes are fully reproducible in R. The code blocks for each chapter are intended to be run sequentially from the start of the chapter. These blocks are easy to copy-and-paste from the web version of the notes. Illustrative implementations of many important algorithms from time series analysis are provided, including simulation and fitting of ARMA models, the Kalman filter, and forward-backward algorithms for HMMs. Despite this, there is not a single explicit `for`, `while` or `repeat` loop to be found anywhere in the code examples. So the code examples also serve to illustrate a more functional approach to R programming than is typically adopted.

Note that the CRAN task view for time series analysis gives an overview of a large number of R packages relevant to Michaelmas term. The spatial task view is very useful for Epiphany term. Similarly, the task view for spatio-temporal data gives an overview of packages relevant to the end of the course. Also note that the Big book of R lists several e-books relevant to the more practical aspects of the course.

About these notes

This is a Quarto book. To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction to time series

1.1 Time series data

1.1.1 What is a time series?

A time series is simply a collection of observations indexed by *time*. Since we measure time in a strictly ordered fashion, from the past, through the present, to the future, we assume that our time index is some subset of the real numbers, and in many cases, this subset will be a subset of the integers. So, for example, we could denote the observation at time t by x_t. Then our time series could be written

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)$$

if we have n observations corresponding to time indices $t = 1, 2, \ldots, n$.

When we use (consecutive) integers for the time index, we are implicitly assuming that the observations are *equispaced* in time.

Crucially, in time series analysis, the *order of the observations matters*. If we permute our observations in some way, the time series is different. Statistical models for time series are not invariant to permutations of the data. This is obviously different to many statistical models you have studied previously, where observations (within some subgroup) are iid (or exchangeable).

In the second half of this module we will study spatial statistical models and the analysis of spatial data. Spatial data is indexed by a position in space, and the spatial index is often assumed to be a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$. Spatial models also have the property that they are not invariant to a permutation of the observations. Indeed, there is a strong analogy to be drawn between time series data and 1-d spatial data, where the spatial index is a subset of $\mathbb{R}$. This analogy can sometimes be helpful, and there are situations where it makes sense to treat time series data as 1-d spatial data.

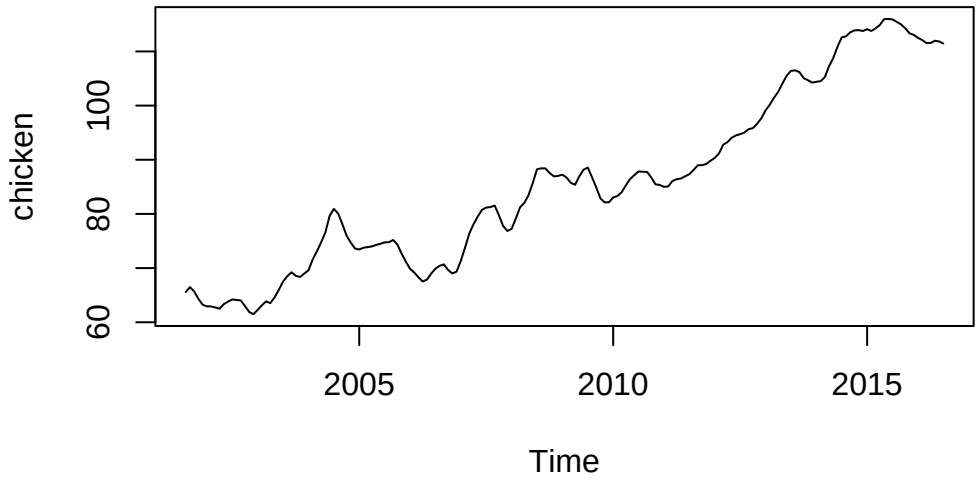
However, the very special nature of the 1-d case (in particular, the complete ordering property of the reals), and the strong emphasis in classical time series analysis on the case of equispaced observations, means that models for time series and methods for the analysis of time series data typically look very different to the models used in spatial statistics.

We will therefore temporarily forget about spatial modelling and spatial statistics, and develop models and methods explicitly focused on time series. We will better understand the relationships between time series and spatial analysis much later, towards the end of Term 2.

1.1.2 An example

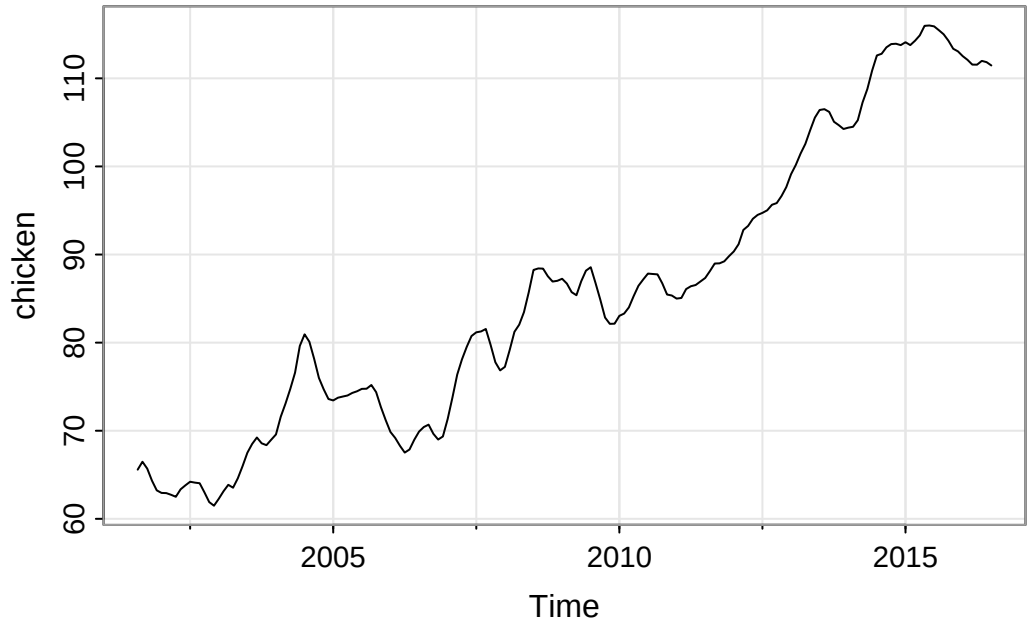
Consider the price of chicken in the US (measured in US cents per pound in weight of chicken). This is a dataset in the R CRAN package `astsa`.

```
library(astsa)
plot(chicken)
```



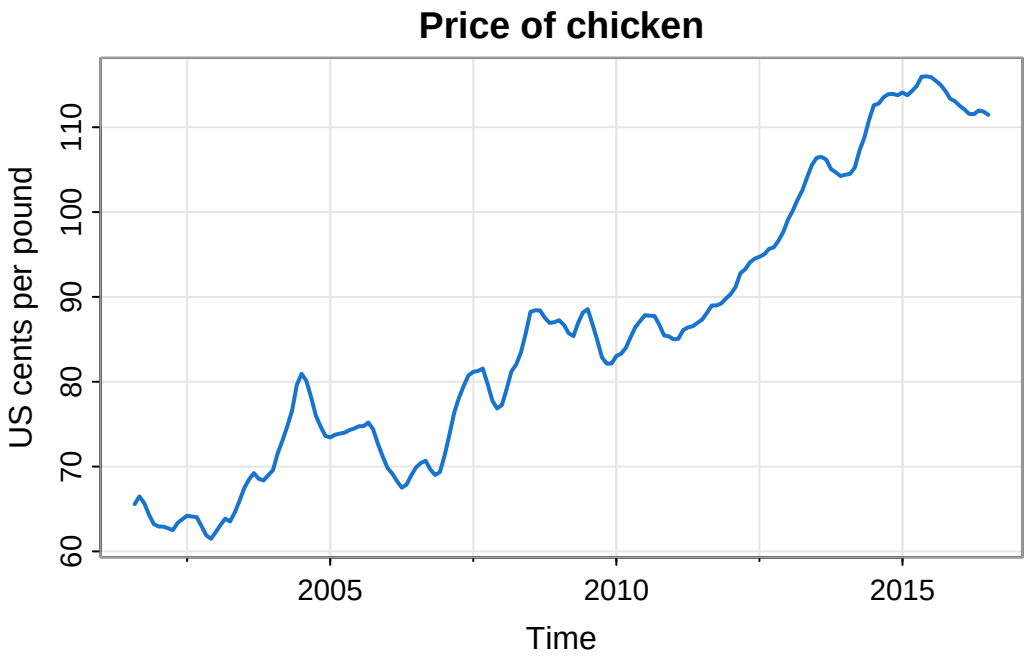
Note that observations close together in time are more highly correlated than observations at disparate times. This uses the R base function for plotting time series (`plot.ts`, since the `chicken` data is a `ts` object). We could also plot using `tsplot` from the `astsa` package.

```
tsplot(chicken)
```



If we wanted, we could make it look a bit nicer.

```
tsplot(chicken, col=4, lwd=2,
  main="Price of chicken", ylab="US cents per pound")
```



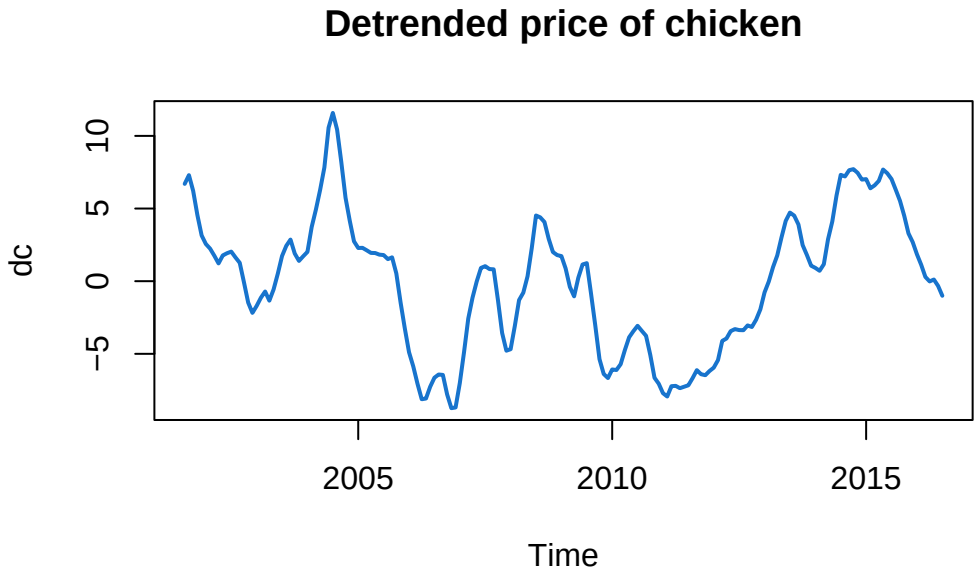
Remember that you can get help on the `astsa` package with `help(package="astsa")`, and help on particular functions with (eg.) `?tsplot`. You can get a list of datasets in the package with `data(package="astsa")`. Note that there is also an online guide to this package: fun with astsa.

1.1.3 Detrending

It is clear that observations close together in time are more highly correlated than observations far apart, but is that simply because of the slowly increasing trend? We can see by *detrending* the data in some way. There are many ways one can detrend a time series, but here, since there looks to be a roughly linear increase in price with time, we could detrend by fitting a linear regression model using time as a covariate, and subtracting off the linear fit, in order to leave us with some detrended residuals.

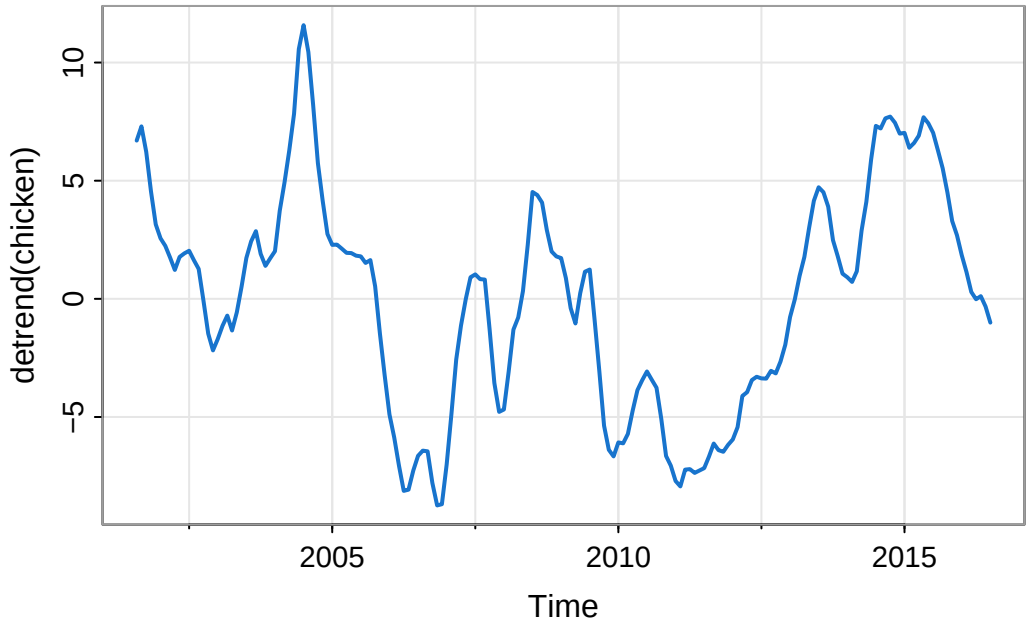
We could do this with base R functions.

```
tc = time(chicken)
res = residuals(lm(chicken ~ tc))
dc = ts(res, start=start(chicken), freq=frequency(chicken))
plot(dc, lwd=2, col=4, main="Detrended price of chicken")
```



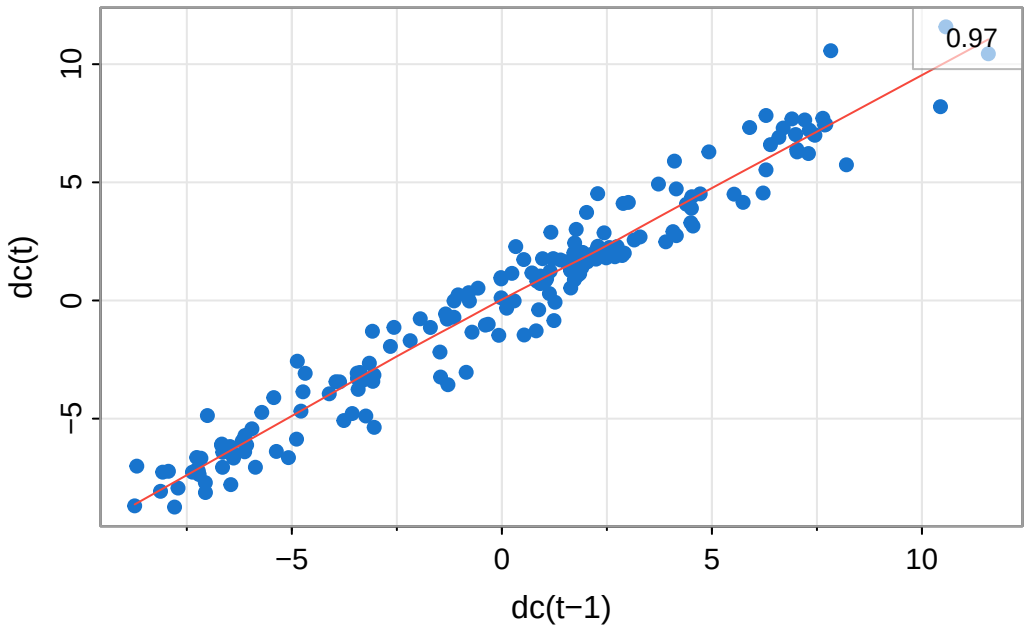
But the `astsa` package has a built-in `detrend` function which makes this much easier.

```
tsplot(detrend(chicken), lwd=2, col=4)
```



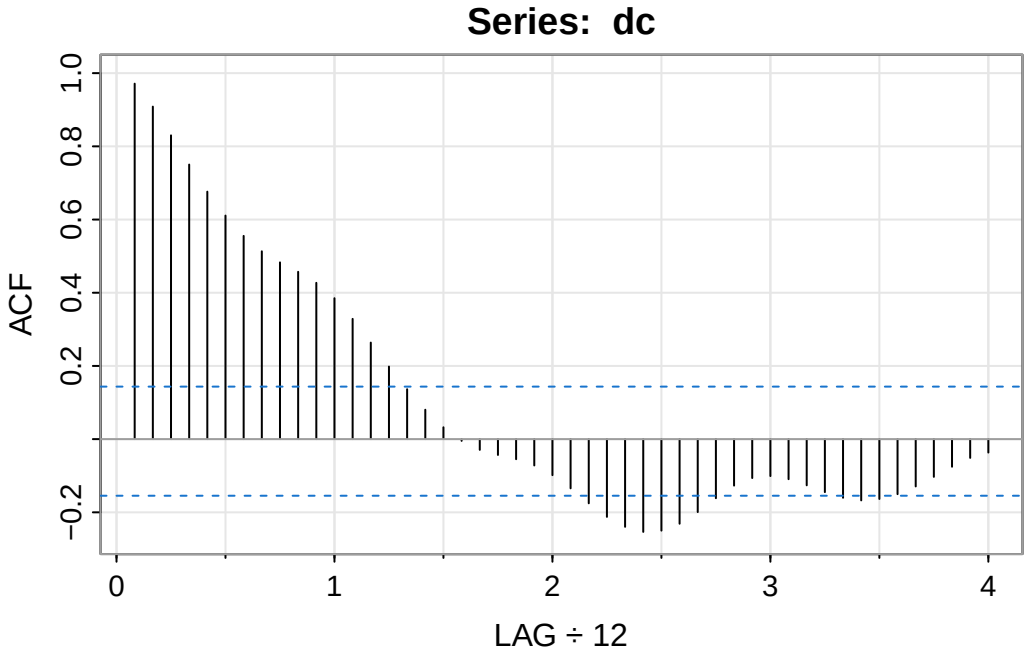
Note that despite having removed the obvious trend from the data it is still the case that observations close together in time are more highly correlated than observations far apart. We can visualise this nearby correlation more clearly by producing a scatter plot showing each observation against the previous observation.

```
lag1.plot(dc, pch=19, col=4)
```



For iid observations, there would be no correlation between adjacent values. Here we see that the correlation at lag 1 (observations one time index apart) is around 0.97. This correlation decreases as the lag increases.

```
acf1(dc)
```




```
 [1]  0.97  0.91  0.83  0.75  0.68  0.61  0.56  0.51  0.48  0.46  0.43  0.39
[13]  0.33  0.26  0.20  0.14  0.08  0.03  0.00 -0.03 -0.04 -0.05 -0.07 -0.10
[25] -0.13 -0.18 -0.21 -0.24 -0.25 -0.25 -0.23 -0.20 -0.16 -0.13 -0.11 -0.10
[37] -0.11 -0.13 -0.14 -0.16 -0.17 -0.16 -0.15 -0.13 -0.10 -0.08 -0.05 -0.04
```

This plot of the auto-correlation against lag is known as the *auto-correlation function* (ACF), or sometimes correlogram, and is an important way to understand the dependency structure of a (stationary) time series. From this plot we see that observations have to be more than one year apart for the auto-correlations to become statistically insignificant.

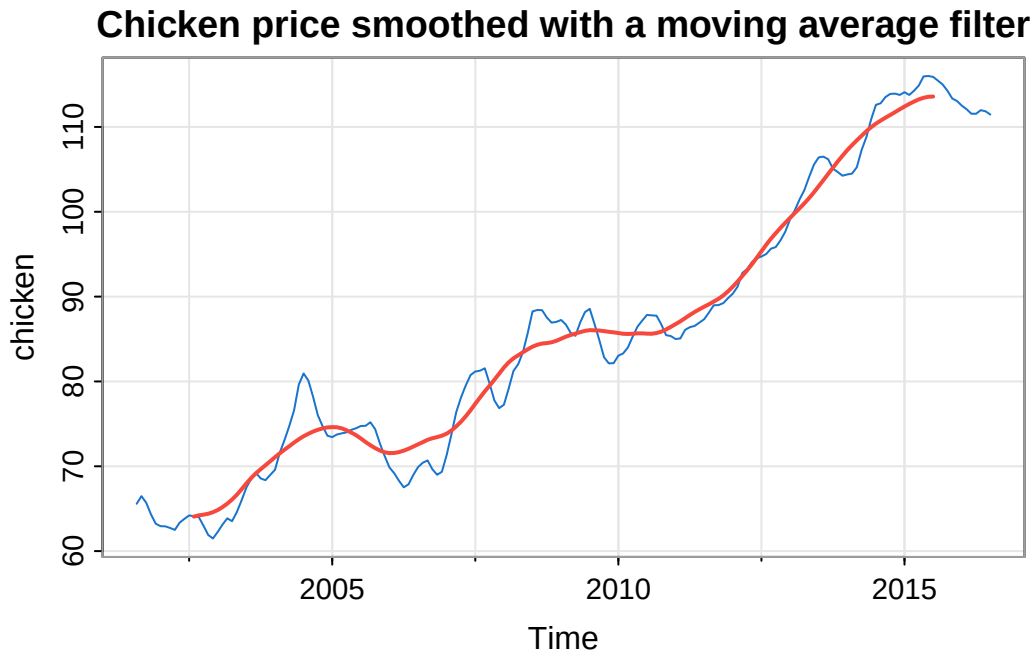
1.2 Filtering time series

1.2.1 Smoothing with convolutional linear filters

Sometimes short-term fluctuations in a time series can obscure longer-term trends. We know that the sample mean has much lower variance than an individual observation, so some kind of (weighted) average might be a good way of getting rid of noise in the data. However, for time series models, it only really makes sense to perform an average locally. This is the idea behind linear filters.

By way of example, consider the chicken price data, and consider forming a new time series where each new observation is the sample mean of the 25 observations closest to that time point.

```
tsplot(chicken, col=4,
  main="Chicken price smoothed with a moving average filter")
lines(filter(chicken, rep(1/25, 25)), col=2, lwd=2)
```



Note that the new time series is shorter than the original time series, and some convention is required for aligning the two time series. Here, R's `filter` function has done something sensible. We see that the filter has done exactly what we wanted - it has smoothed out short-term fluctuations, better revealing the long-term trend in the data.

A *convolutional linear filter* defines a new time series $\mathbf{y} = \{y_t\}$ from an input time series $\mathbf{x} = \{x_t\}$ via

$$y_t = \sum_{i=q_1}^{q_2} w_i x_{t-i},$$

for $q_1 \leq 0 \leq q_2$ and pre-specified *weights*, $w_{q_1}, \ldots, w_{q_2}$ (so $q_2 - q_1 + 1$ weights in total). If the weights are non-negative and sum to one, then the filter represents a moving average, but this is not required. Note that if the input series is defined for time indices $1, 2, \ldots, n$, then the output series will be defined for time indices $q_2 + 1, q_2 + 2, \ldots, n + q_1$, and hence will be of length $n + q_1 - q_2$.

In the context of digital signal processing (DSP), this would be called a finite impulse response (FIR) filter, and the convolution operation is sometimes summarised with the shorthand notation

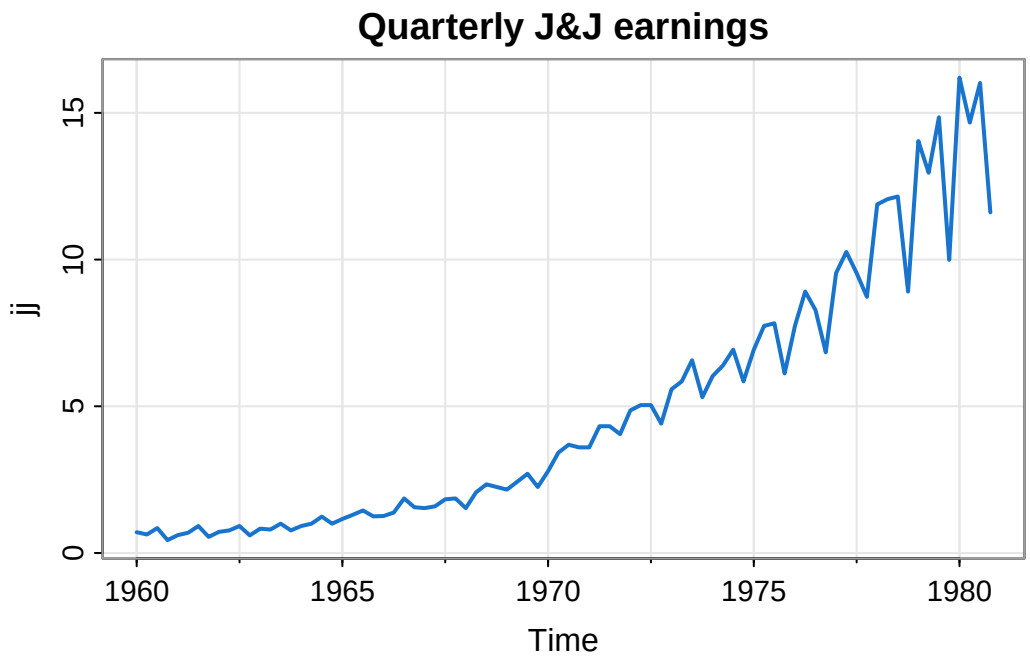
$$\mathbf{y} = \mathbf{w} * \mathbf{x}.$$

Very commonly, a *symmetric* filter will be used, where $q_1 = -q$ and $q_2 = q$, for some $q > 0$, and $w_i = w_{-i}$. Although the weight vector is of length $2q + 1$, there are only $q + 1$ free parameters to be specified. In this case, if the input time indices are $1, 2, \ldots, n$, then the output indices will be $q + 1, q + 2, \ldots, n - q$ and hence the output will be of length $n - 2q$.

1.2.2 Seasonality

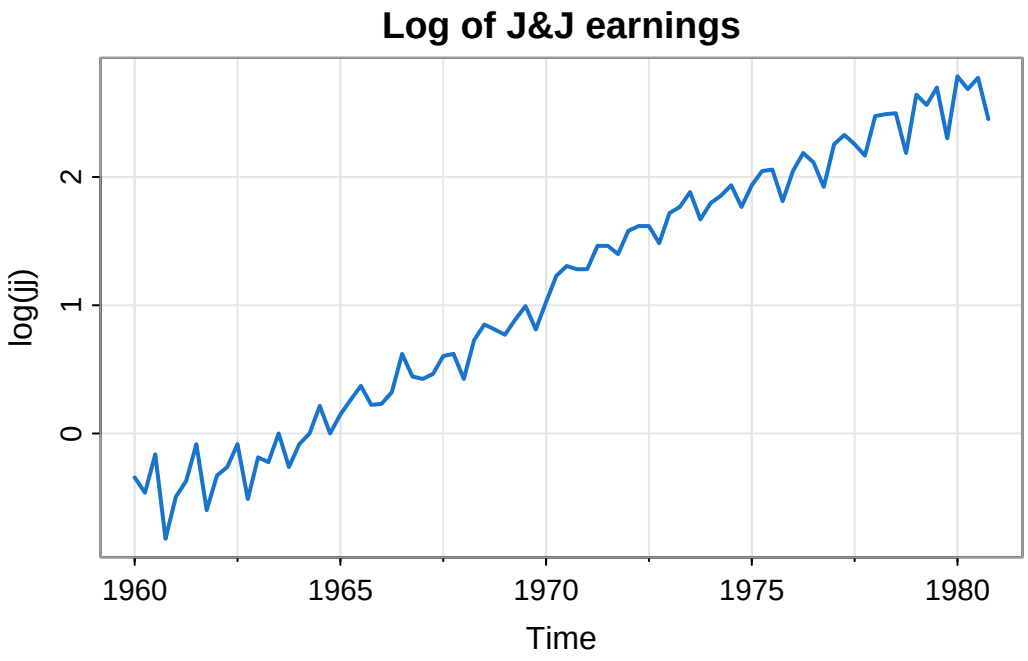
Let's look now at the quarterly share earnings for the company Johnson and Johnson.

```
tsplot(jj, col=4, lwd=2, main="Quarterly J&J earnings")
```



We can see an exponential increase in the overall trend. We can also see that the deviations from the overall trend seem to exhibit a regular pattern with a period of 4. There seem to be particular quarters where earnings are lower than the overall trend, and others where they seem to be higher. This kind of *seasonality* in time series data is very common. Seasonality can happen on many time scales, but *yearly*, *weekly* and *daily* is most common. On closer inspection, we see that the seasonal effects seem to be increasing exponentially, in line with the overall trend. We could attempt to develop a *multiplicative model* for this data, but it might be simpler to just take logs.

```
tsplot(log(jj), col=4, lwd=2, main="Log of J&J earnings")
```



To progress further, we probably want to detrend in some way. We could take out a linear fit, but we could also estimate the trend by smoothing out the seasonal effects with a moving average that averages over 4 consecutive quarters to give a yearly average. That is, we could apply a convolutional linear filter with weight vector

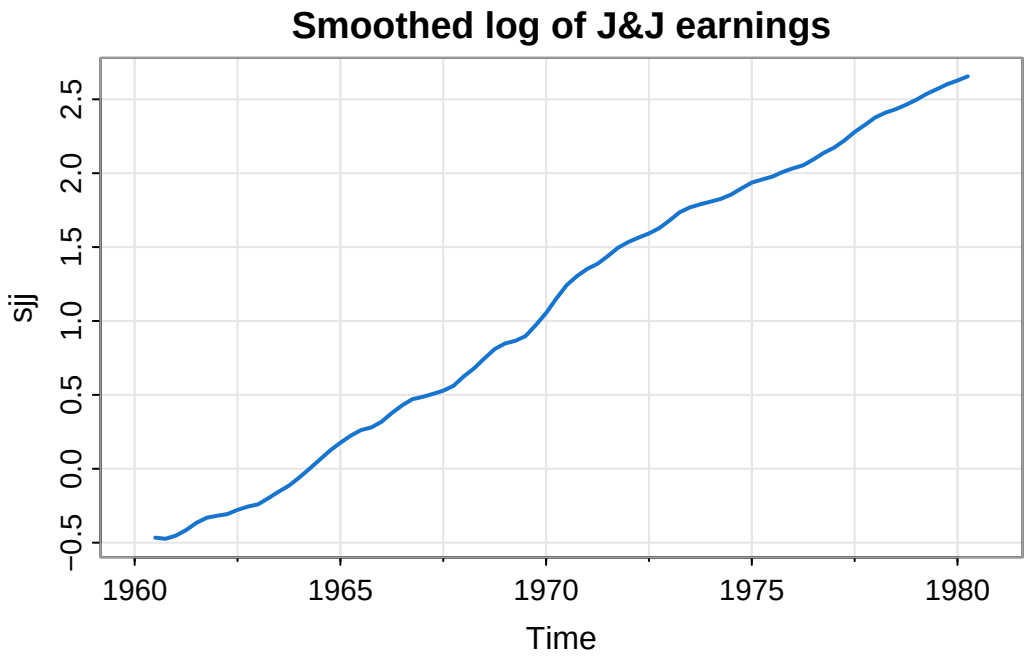
$$\mathbf{w} = (0.25, 0.25, 0.25, 0.25).$$

The problem with this is that it is of even length, so it represents an average that lies between two quarters, and so doesn't align nicely with our existing series. We can fix this by using a symmetric moving average centred on a given quarter by instead using a weight vector of length 5.

$$\mathbf{w} = (0.125, 0.25, 0.25, 0.25, 0.125).$$

Now each quarter still receives a total weight of 0.25, but the weight for the quarter furthest away is split across two different years.

```
sjj = filter(log(jj), c(0.125,0.25,0.25,0.25,0.125))
tsplot(sjj, col=4, lwd=2, main="Smoothed log of J&J earnings")
```



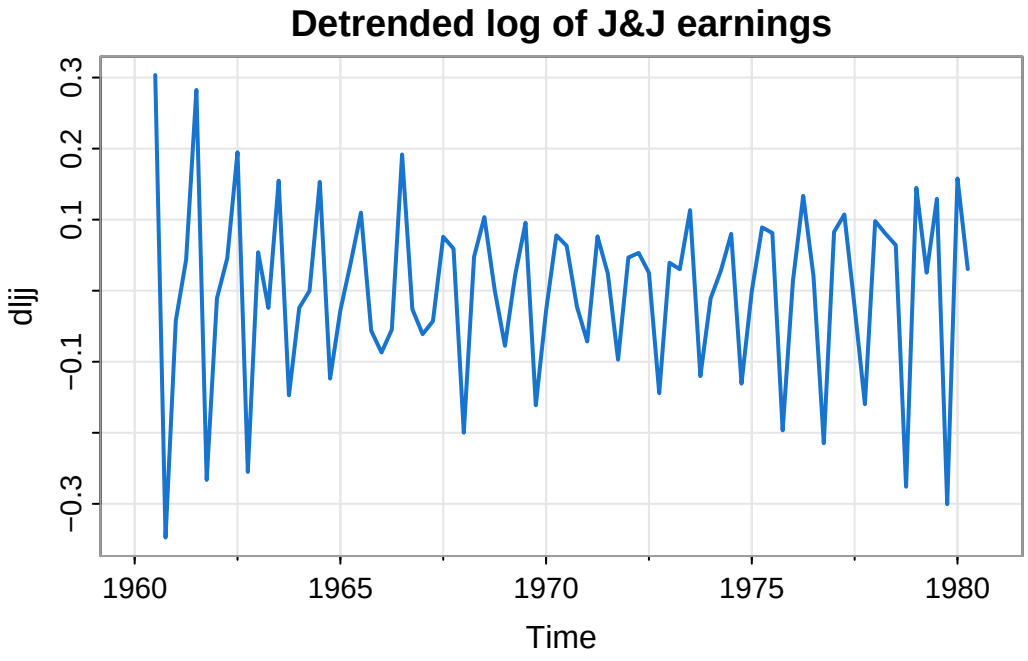
We can see that this has nicely smoothed out the seasonal fluctuations giving just the long term trend.

So, in general, when trying to deseasonalise a time series with seasonal fluctuations with period m, if m is odd (which is quite rare), then just use a filter with a weight vector of length m, with all weights set to $1/m$. In the more typical case where m is even, use a weight vector of length $m + 1$ where the end weights are set to $1/(2m)$ and all other weights are set to $1/m$.

This deseasonalisation process is an example of what DSP people would call a low-pass filter. It has filtered out the high-frequency oscillations, allowing the long term (low frequency) behaviour to pass through.

We can now detrend the log data by subtracting off this trend to leave just the seasonal effects.

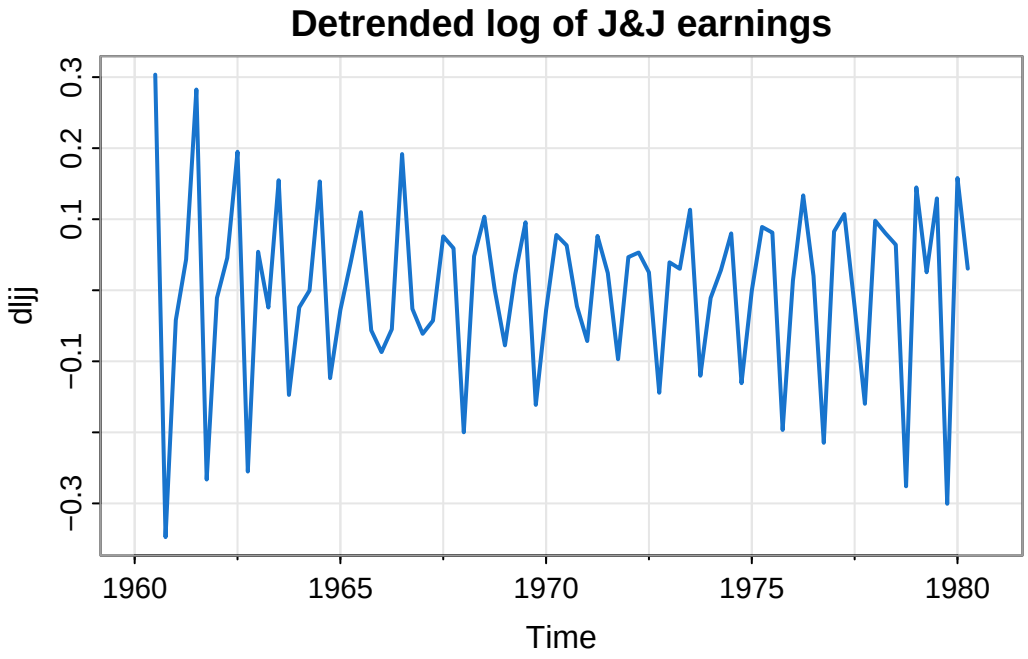
```
dljj = log(jj) - sjj
tsplot(dljj, col=4, lwd=2, main="Detrended log of J&J earnings")
```



Note that we could have directly computed the detrended data by applying a convolutional linear filter with weight vector

$$\mathbf{w} = (-0.125, -0.25, 0.75, -0.25, -0.125)$$

```
dljj = filter(log(jj), c(-0.125,-0.25,0.75,-0.25,-0.125))
tsplot(dljj, col=4, lwd=2, main="Detrended log of J&J earnings")
```



Note that this filter has weights summing to zero. This detrending process is an example of what DSP people would call a high-pass filter. It has allowed the high frequency oscillations to pass through, while removing the long term (low frequency) behaviour.

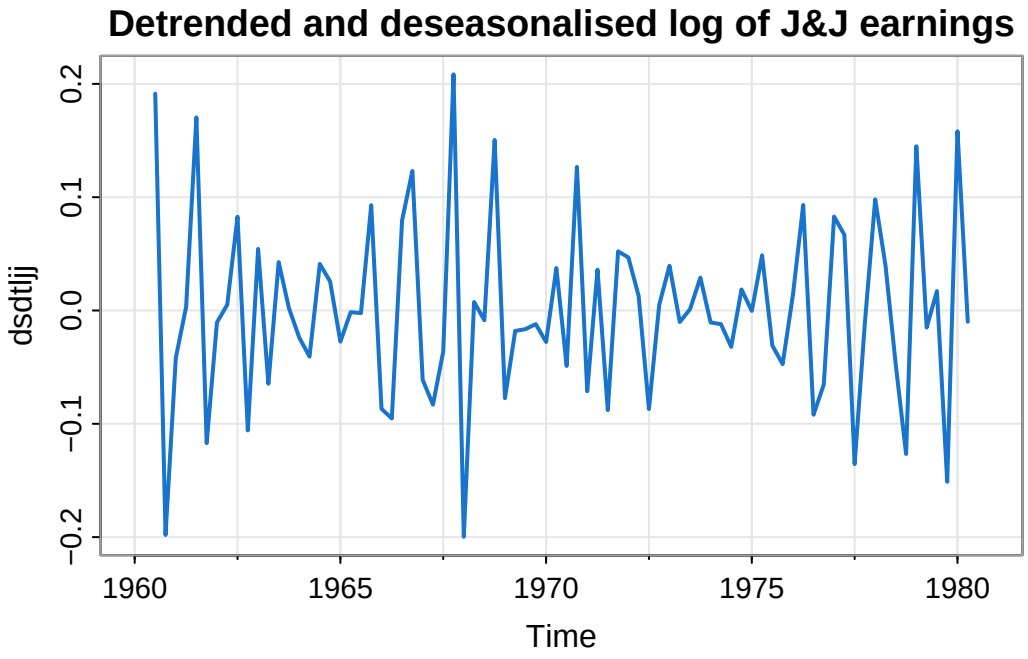
We could use this detrended data to get a simple estimate of the seasonal effects associated with each of the four quarters.

```
seff = rowMeans(matrix(dljj, nrow=4), na.rm=TRUE)
seff
```

```
[1] -0.0001916332  0.0403896607  0.1121470367 -0.1492881658
```

We can now strip out these seasonal effects.

```
dsdtljj = dljj - seff
tsplot(dsdtljj, col=4, lwd=2,
  main="Detrended and deseasonalised log of J&J earnings")
```



Not perfect (as the seasonal effects aren't perfectly additive), and definitely lots of auto-correlation left, but it's getting closer to some kind of correlated noise process.

1.2.3 Exponential smoothing and auto-regressive linear filters

We previously smoothed the chicken price data by applying a moving average filter to it. That is by no means the only way we could smooth the data. One drawback of the moving average filter is that you lose values at each end of the series. Losing values at the right-hand end is particularly problematic if you are using historic data up to the present in order to better understand the present (and possibly forecast into the future). In the on-line context it makes sense to keep a current estimate of the underlying level (smoothed value), and update this estimate with each new observation. Exponential smoothing is probably the simplest variant of this approach. We can define a smoothed time series $\mathbf{s} = \{s_t\}$ using the input time series $\mathbf{x} = \{x_t\}$ via the update equation

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1},$$

for some fixed $\alpha \in [0, 1]$. It is clear from this definition that the new smoothed value is a weighted average of the current observation and the previous smoothed value. High values of α put more weight on the current observation (and hence, less smoothing), whereas low values of α put less weight on the current observation, leading to more smoothing. It is called exponential smoothing because recursive application of the update formula makes it clear that the weights associated with observations from the past decay away exponentially with time. It is worth noting that the update relation can be re-written

$$s_t = s_{t-1} + \alpha(x_t - s_{t-1}),$$

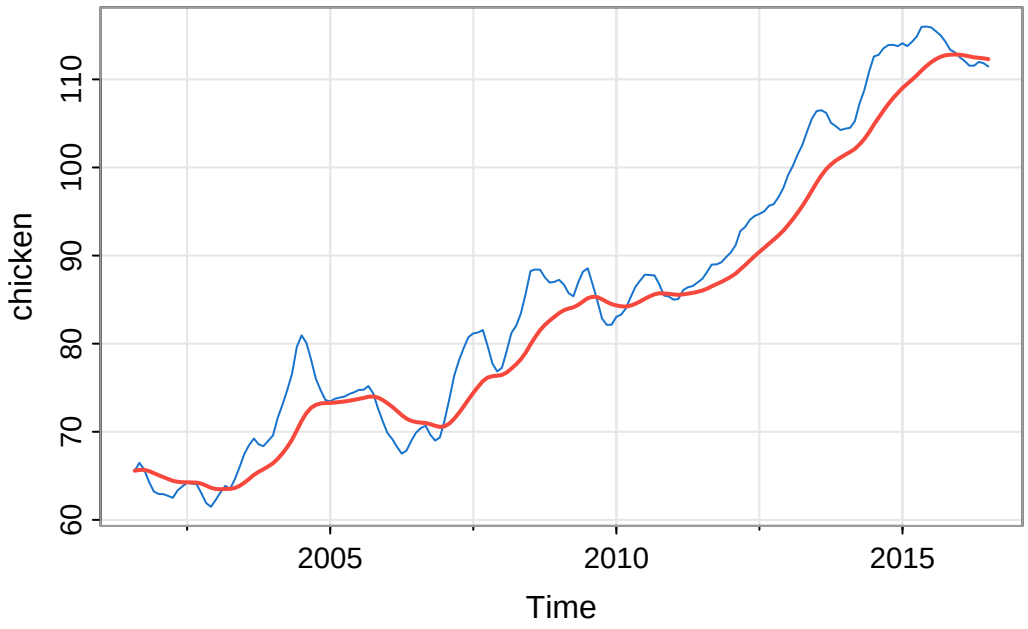
which can sometimes be convenient.

Note that some strategy is required for initialising the smoothed state. eg. if the first observation is x_1, then to compute s_1, some initial s_0 is required. Sometimes $s_0 = 0$ is used, but more often $s_0 = x_1$ is used, which obviously involves looking ahead at the data, but ensures that $s_1 = x_1$, which is often sensible.

The smoothing value, α can be specified subjectively, noting that this is no more subjective than choosing the size of the smoothing window for a moving average filter. Alternatively, some kind of cross-validation strategy can be used, based on, say, one-step ahead prediction errors.

The `stats::filter` function is not flexible enough to allow us to implement this kind of linear filter, so we instead using the `signal::filter` function.

```
alpha = 0.1
sm = signal::filter(alpha, c(1, alpha-1), chicken, init.y=chicken[1])
tsp(sm) = tsp(chicken)
tsplot(chicken, col=4)
lines(sm, col=2, lwd=2)
```



We will figure out the details of the `signal::filter` function shortly. We see that the filtered output is much smoother than the input, but also that it somewhat lags the input series. This can be tuned by adjusting the smoothing parameter. Less smoothing will lead to less lag.

Exponential smoothing is a special case of an *auto-regressive moving average* (ARMA) linear filter, which DSP people would call an infinite impulse response (IIR) filter. The ARMA filter relates an input time series $\mathbf{x} = \{x_t\}$ to an output time series $\mathbf{y} = \{y_t\}$ via the relation

$$\sum_{i=0}^{p} a_i y_{t-i} = \sum_{j=0}^{q} b_j x_{t-j},$$

for some vectors $\mathbf{a}$ and $\mathbf{b}$, of length $p+1$ and $q+1$, respectively. This can obviously be rearranged as

$$y_t = \frac{1}{a_0}\left(\sum_{j=0}^{q} b_j x_{t-j} - \sum_{i=1}^{p} a_i y_{t-i}\right).$$

These are more powerful than the convolution filters considered earlier, since they allow the propagation of persistent state information.

It is clear that if we choose $\mathbf{a} = (1, \alpha - 1)$, $\mathbf{b} = (\alpha)$ we get the exponential smoothing model previously described. The `signal::filter` function takes $\mathbf{b}$ and $\mathbf{a}$ as its first two arguments, which should explain the above code block.

1.3 Multivariate time series

The `climhyd` dataset records monthly observations on several variables relating to Lake Shasta in California (see `?climhyd` for further details). The monthly observations are stored as rows of a data frame. We can get some basic information about the dataset as follows.

```
str(climhyd)
```

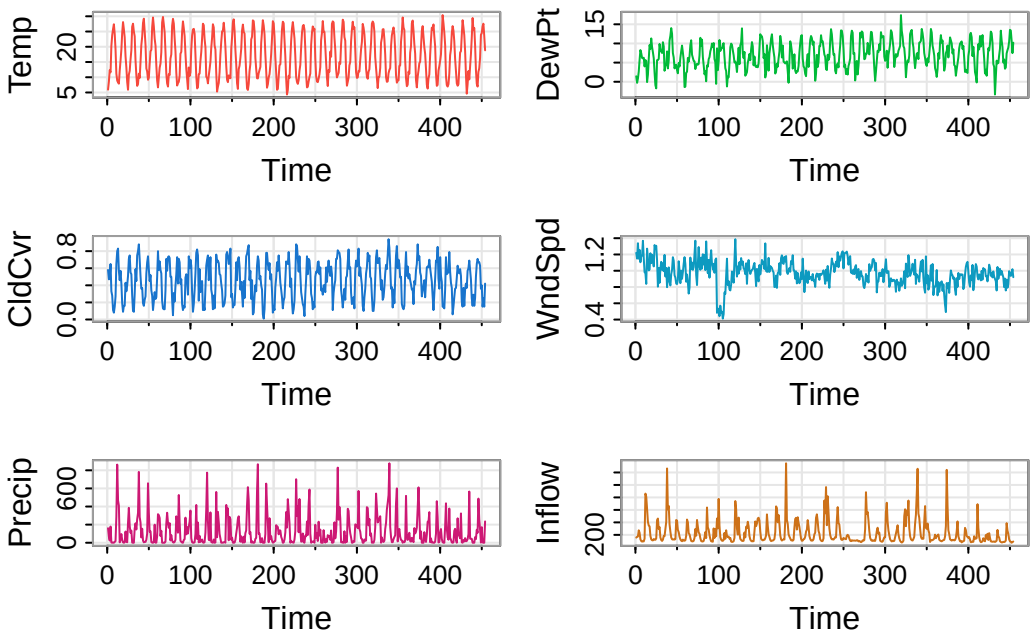
```
'data.frame':   454 obs. of  6 variables:
 $ Temp  : num  5.94 8.61 12.28 11.61 20.28 ...
 $ DewPt : num  1.436 -0.285 0.857 2.696 5.7 ...
 $ CldCvr: num  0.58 0.47 0.49 0.65 0.33 0.28 0.11 0.08 0.15 0.27 ...
 $ WndSpd: num  1.22 1.15 1.34 1.15 1.26 ...
 $ Precip: num  160.53 65.79 24.13 178.82 2.29 ...
 $ Inflow: num  156 168 173 273 233 ...
```

```
head(climhyd)
```

```
   Temp      DewPt CldCvr   WndSpd  Precip   Inflow
1  5.94  1.436366   0.58 1.219485 160.528 156.1173
2  8.61 -0.284660   0.47 1.148620  65.786 167.7455
3 12.28  0.856728   0.49 1.338430  24.130 173.1567
4 11.61  2.696482   0.65 1.147778 178.816 273.1516
5 20.28  5.699536   0.33 1.256730   2.286 233.4852
6 23.83  8.275339   0.28 1.104325   0.508 128.4859
```

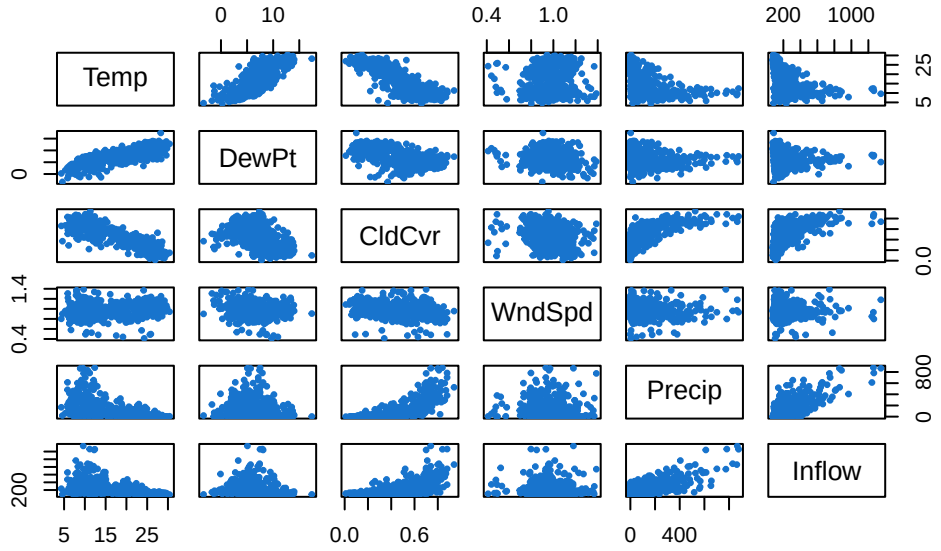
We can then plot the observations.

```
tsplot(climhyd, ncol=2, col=2:7)
```



The data can be regarded as a collection of six univariate time series. We could just treat them completely separately, applying univariate time series analysis techniques to each variable in turn. However, this wouldn't necessarily be a good idea unless the series are all *independent* of one another. But typically, the reason we study a collection of time series together is because we suspect that they are not all independent of one another. Here, we could look at a simple pairs plot of the variables.

```
pairs(climhyd, col=4, pch=19, cex=0.5)
```



We see that there do appear to be cross-correlations between the series, strongly suggesting that the component series are not independent of one another. It is then an interesting question as to the best way to analyse a collection of dependent time series, with correlations across variables as well as over time. We will come back to this question later.

1.4 Time series analysis

We've looked at some basic exploratory tools for visualising and decomposing time series into trend, seasonal and residual components. But we know from previous courses that to do any meaningful statistical inference for the underlying features, or probabilistic prediction (or forecasting), then it is very helpful to have a statistical model of the data generating process. We typically approach this by building a model with some systematic components (such as a trend and seasonal effects), and then using an appropriate probability model for the residuals. But in most simple statistical models, this residual process is assumed to be iid. However, we have seen that for time series, it is often implausible that the residuals will be iid. We have seen that it is likely that we should be able to account for systematic effects, leaving residuals that are mean zero. In many cases, it may even be plausible that the residuals have constant variance. But we have seen that typically the residuals will be auto-correlated, with correlations tailing off as the lag increases. It would therefore be useful to have a family of discrete time stochastic processes that can model this kind of behaviour. It will turn out that ARMA models are a good solution to this problem, and we will study these properly in Chapter 3. But first we will set the scene by examining a variety of (first and second order) linear Markovian dynamical systems. This will help to provide some context for ARMA models, and hopefully make them seem a bit less mysterious.

2 Linear systems

2.1 Introduction

It is likely that you will be familiar with many parts of this chapter, having seen different bits in different courses, in isolation. It might seem a bit strange to look at deterministic linear dynamical systems in a stats course, but it is arguably helpful to see them alongside their stochastic counterparts, in one place, in a consistent notation, to be able to compare and contrast. It could also be argued that this material would belong better in a course on stochastic processes. But time series models *are* stochastic processes, so we really do need to understand something about stochastic processes to be able to model time series appropriately. This chapter will hopefully make clear how everything links together in a consistent way, and how understanding deterministic dynamical systems helps to understand stochastic counterparts. In particular, it should help to motivate the study of ARMA models in Chapter 3.

2.2 Vector random quantities

Although this course will be mainly concerned with univariate time series, there are many instances in the analysis of time series where vector random quantities and the multivariate normal distribution crop up. It will be useful to recap some essential formulae that you have probably seen before.

2.2.1 Moments

For a random vector $\mathbf{X}$ with ith element X_i, the expectation is the vector $\mathbb{E}[\mathbf{X}]$, of the same length, with ith element $\mathbb{E}[X_i]$.

The covariance matrix between vectors $\mathbf{X}$ and $\mathbf{Y}$, is the matrix with (i,j)th element $\mathbb{C}\mathrm{ov}[X_i, Y_j]$, and hence given by

$$\mathbb{C}\mathrm{ov}[\mathbf{X}, \mathbf{Y}] = \mathbb{E}\left\{(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{Y} - \mathbb{E}[\mathbf{Y}])^\top\right\} = \mathbb{E}[\mathbf{X}\mathbf{Y}^\top] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{Y}]^\top.$$

It is easy to show that

$$\mathbb{C}\mathrm{ov}[\mathsf{A}\mathbf{X} + \mathbf{b}, \mathsf{C}\mathbf{Y} + \mathbf{d}] = \mathsf{A}\mathbb{C}\mathrm{ov}[\mathbf{X}, \mathbf{Y}]\mathsf{C}^\top,$$

$$\mathbb{C}\mathrm{ov}[\mathbf{X} + \mathbf{Y}, \mathbf{Z}] = \mathbb{C}\mathrm{ov}[\mathbf{X}, \mathbf{Z}] + \mathbb{C}\mathrm{ov}[\mathbf{Y}, \mathbf{Z}], \quad \text{and} \quad \mathbb{C}\mathrm{ov}[\mathbf{X}, \mathbf{Y}]^\top = \mathbb{C}\mathrm{ov}[\mathbf{Y}, \mathbf{X}].$$

We define the variance matrix

$$\mathbb{V}\mathrm{ar}[\mathbf{X}] = \mathbb{C}\mathrm{ov}[\mathbf{X}, \mathbf{X}],$$

from which it follows that

$$\mathbb{V}\mathrm{ar}[\mathsf{A}\mathbf{X} + \mathbf{b}] = \mathsf{A}\mathbb{V}\mathrm{ar}[\mathbf{X}]\mathsf{A}^\top.$$

2.2.2 The multivariate normal distribution

The p-dimensional multivariate normal distribution can be defined as an affine transformation of a p-vector of iid standard normal random scalars, and is characterised by its (p-dimensional) expectation vector, $\boldsymbol{\mu}$ and ($p \times p$) variance matrix, Σ, and written $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$. From this definition it is clear that affine transformations of multivariate normal random quantities will be multivariate normal. It is also relatively easy to deduce that the probability density function (PDF) must take the form

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = (2\pi)^{-p/2}|\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}.$$

From this, relatively routine computations show that if $\mathbf{X}$ and $\mathbf{Y}$ are jointly multivariate normal, then both $\mathbf{X}$ and $\mathbf{Y}$ are marginally multivariate normal, and further, that the conditional random quantity, $(\mathbf{X}|\mathbf{Y}=\mathbf{y})$ is multivariate normal, and characterised by its conditional mean and variance,

$$\mathbb{E}[\mathbf{X}|\mathbf{Y}=\mathbf{y}] = \mathbb{E}[\mathbf{X}] + \mathbb{C}\text{ov}[\mathbf{X},\mathbf{Y}]\mathbb{V}\text{ar}[\mathbf{Y}]^{-1}(\mathbf{y}-\mathbb{E}[\mathbf{Y}]),$$

$$\mathbb{V}\text{ar}[\mathbf{X}|\mathbf{Y}=\mathbf{y}] = \mathbb{V}\text{ar}[\mathbf{X}] - \mathbb{C}\text{ov}[\mathbf{X},\mathbf{Y}]\mathbb{V}\text{ar}[\mathbf{Y}]^{-1}\mathbb{C}\text{ov}[\mathbf{Y},\mathbf{X}].$$

It is noteworthy that the conditional variance of $\mathbf{X}$ does not depend on the observed value of $\mathbf{Y}$ (this is a special property of the multivariate normal distribution).

2.3 First order systems

2.3.1 Deterministic

2.3.1.1 Discrete time

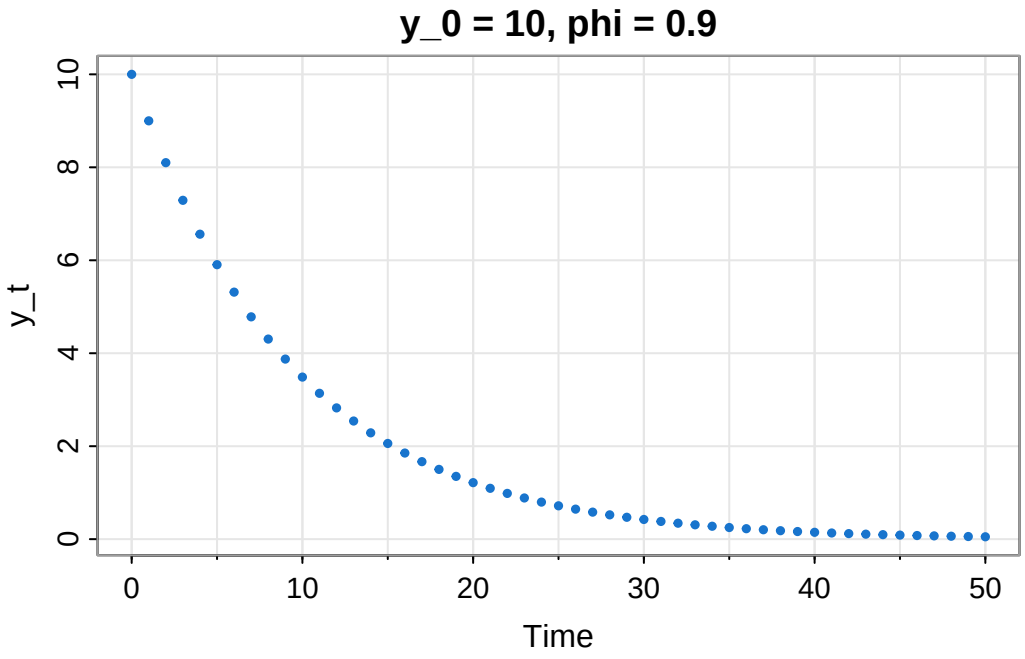
We will start with the simplest linear Markovian model: the *linear recurrence relation*. We start with the special case

$$y_t = \phi y_{t-1}, \quad t \in \{1, 2, \ldots\},$$

for some fixed $\phi \in \mathbb{R}$, initialised with some $y_0 \in \mathbb{R}$. Using substitution we get $y_1 = \phi y_0$, $y_2 = \phi y_1 = \phi^2 y_0$, *etc.*, leading to

$$y_t = \phi^t y_0, \quad t \in \{1, 2, \ldots\}.$$

```
library(astsa)
tsplot(0:50, sapply(0:50, function(t) 10*(0.9)^t),
  type="p", pch=19, cex=0.5, col=4, ylab="y_t",
  main="y_0 = 10, phi = 0.9")
```



This will be *stable* for $|\phi| < 1$, leading to $y_\infty = 0$, and will diverge geometrically for $|\phi| > 1, y_0 \neq 0$ (other cases require special consideration). Next consider the shifted system $x_t = y_t + \mu$, so that

$$x_t = \mu + \phi(x_{t-1} - \mu).$$

By construction, this system will be stable for $|\phi| < 1$ with $x_\infty = \mu$. Note that we can re-write this as

$$x_t = \phi x_{t-1} + (1 - \phi)\mu,$$

so for a system of the form

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

we know that it will be stable for $|\phi| < 1$ with $x_\infty = \mu = k/(1 - \phi)$. Further, since the general solution in terms of μ is

$$x_t = \mu + \phi^t(x_0 - \mu) = \phi^t x_0 + (1 - \phi^t)\mu,$$

the general solution in terms of k is

$$x_t = \phi^t x_0 + \frac{1 - \phi^t}{1 - \phi}k.$$

2.3.1.2 Vector discrete time

We can generalise the above system to vectors in the obvious way. Start with

$$\mathbf{y}_t = \Phi\mathbf{y}_{t-1}, \quad t \in \{1, 2, \ldots\},$$

for some fixed matrix $\Phi \in M_{n\times n}$, initialised with vector $\mathbf{y}_0 \in \mathbb{R}^n$. Exactly as above we get

$$\mathbf{y}_t = \Phi^t\mathbf{y}_0,$$

and this will be stable provided that all of the eigenvalues (λ_i) of Φ lie inside the unit circle in the complex plane ($|\lambda_i| < 1$), since then $\Phi^t \to 0$, giving $\mathbf{y}_\infty = \mathbf{0}$.

Defining $\mathbf{x}_t = \mathbf{y}_t + \boldsymbol{\mu}$ gives

$$\mathbf{x}_t = \boldsymbol{\mu} + \Phi(\mathbf{x}_{t-1} - \boldsymbol{\mu}) = \Phi\mathbf{x}_{t-1} + (\mathbb{I} - \Phi)\boldsymbol{\mu},$$

and so a system of the form

$$\mathbf{x_t} = \Phi \mathbf{x}_{t-1} + \mathbf{k},$$

in the stable case will have $\mathbf{x}_\infty = \boldsymbol{\mu} = (\mathbb{I} - \Phi)^{-1}\mathbf{k}$. Note that in the stable case, $\mathbb{I} - \Phi$ will be invertible. Further, since the general solution in terms of $\boldsymbol{\mu}$ is

$$\mathbf{x}_t = \boldsymbol{\mu} + \Phi^t(\mathbf{x}_0 - \boldsymbol{\mu}) = \Phi^t \boldsymbol{x}_0 + (\mathbb{I} - \Phi^t)\boldsymbol{\mu},$$

the general solution in terms of $\mathbf{k}$ is

$$\mathbf{x}_t = \Phi^t \boldsymbol{x}_0 + (\mathbb{I} - \Phi^t)(\mathbb{I} - \Phi)^{-1}\mathbf{k}.$$

2.3.1.3 Continuous time

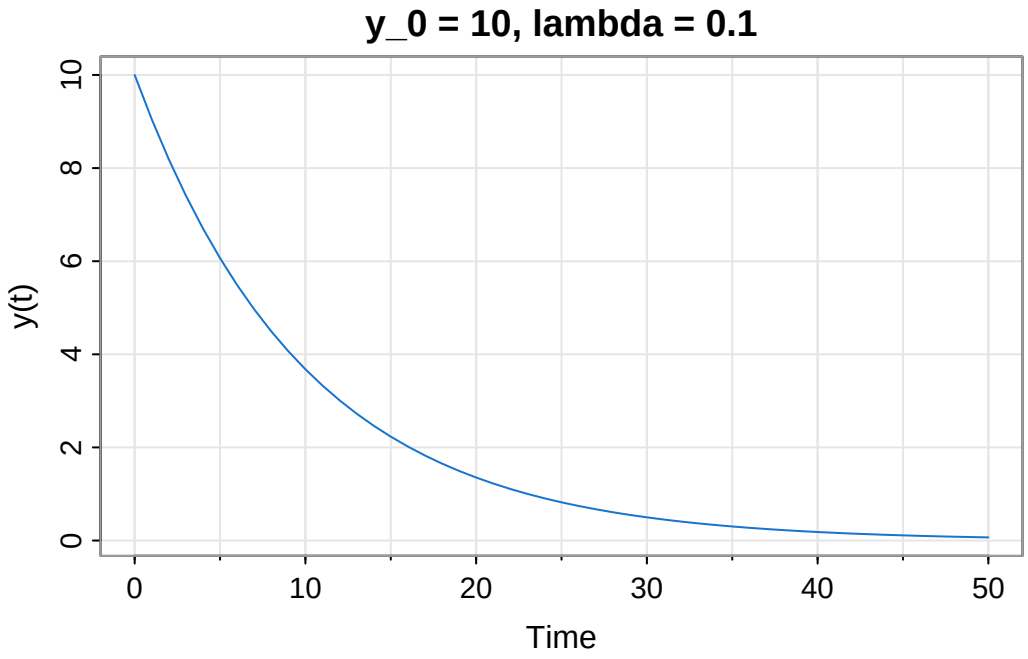
Let's revert to the scalar case, but switch from discrete time to cts time. The analogous system is the scalar linear ODE

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -\lambda y,$$

for some fixed $\lambda \in \mathbb{R}$, initialised with $y_0 \in \mathbb{R}$. By separating variables (or guessing) we deduce the solution

$$y(t) = y_0 e^{-\lambda t}.$$

```
tsplot(0:50, sapply(0:50, function(t) 10*exp(-0.1*t)),
  cex=0.5, col=4, ylab="y(t)", main="y_0 = 10, lambda = 0.1")
```



This will be *stable* for $\lambda > 0$, giving $y(\infty) = 0$, otherwise likely to diverge (exponentially, but various cases to consider). Note that $y(t) = \phi^t y_0$ for $\phi = e^{-\lambda}$, so the continuous time model evaluated at integer times is just a discrete time model.

Defining $x(t) = y(t) + \mu$ gives

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -\lambda(x - \mu),$$

and so for $\lambda > 0$ we will get $x(\infty) = \mu$. So for a linear ODE of the form

$$\frac{\mathrm{d}x}{\mathrm{d}t} + \lambda x = k,$$

we have $x(\infty) = \mu = k/\lambda$ in the stable case ($\lambda > 0$). Further, since the general solution in terms of μ is

$$x(t) = \mu + (x_0 - \mu)e^{-\lambda t} = x_0 e^{-\lambda t} + \mu(1 - e^{-\lambda t}),$$

the general solution in terms of k is

$$x(t) = x_0 e^{-\lambda t} + \frac{k}{\lambda}(1 - e^{-\lambda t}).$$

2.3.1.4 Vector continuous time

The vector equivalent to the previous model is

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = -\Lambda\mathbf{y},$$

for $\Lambda \in M_{n\times n}$, initialised with $\mathbf{y}_0 \in \mathbb{R}^n$. The solution is

$$\mathbf{y}(t) = \exp\{-\Lambda t\}\mathbf{y}_0,$$

where $\exp\{\cdot\}$ denotes the matrix exponential function. For *stability*, we need the eigenvalues of $\exp\{-\Lambda\}$ to be inside the unit circle. For that we need the real parts of the eigenvalues of $-\Lambda$ to be negative. In other words *we need the real parts of the eigenvalues of* Λ *to be positive*. Otherwise the process is likely to diverge, but again, there are various cases to consider. In the stable case we have $\mathbf{y}(\infty) = \mathbf{0}$.

Note that for integer times, t, we have $\mathbf{y}(t) = \Phi^t\mathbf{y}_0$, where $\Phi = \exp\{-\Lambda\}$, so again the solution of the continuous time model at integer times is just a discrete time recurrence relation.

Defining $\mathbf{x}(t) = \mathbf{y}(t) + \boldsymbol{\mu}$ gives

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = -\Lambda(\mathbf{x} - \boldsymbol{\mu}),$$

with stable limit $\mathbf{x}(\infty) = \boldsymbol{\mu}$. So, given a linear ODE of the form

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} + \Lambda\mathbf{x} = \mathbf{k},$$

we deduce the stable limit $\mathbf{x}(\infty) = \boldsymbol{\mu} = \Lambda^{-1}\mathbf{k}$. Note that in the stable case, Λ will be invertible. Further, since the general solution in terms of $\boldsymbol{\mu}$ is

$$\mathbf{x}(t) = \boldsymbol{\mu} + \exp\{-\Lambda t\}(\mathbf{x}_0 - \boldsymbol{\mu}) = \exp\{-\Lambda t\}\mathbf{x}_0 + (\mathbb{I} - \exp\{-\Lambda t\})\boldsymbol{\mu},$$

the general solution in terms of $\mathbf{k}$ is

$$\mathbf{x}(t) = \exp\{-\Lambda t\}\mathbf{x}_0 + (\mathbb{I} - \exp\{-\Lambda t\})\Lambda^{-1}\mathbf{k}.$$

2.3.2 Stochastic

We now turn our attention to the natural linear Gaussian stochastic counterparts of the deterministic systems we have briefly reviewed.

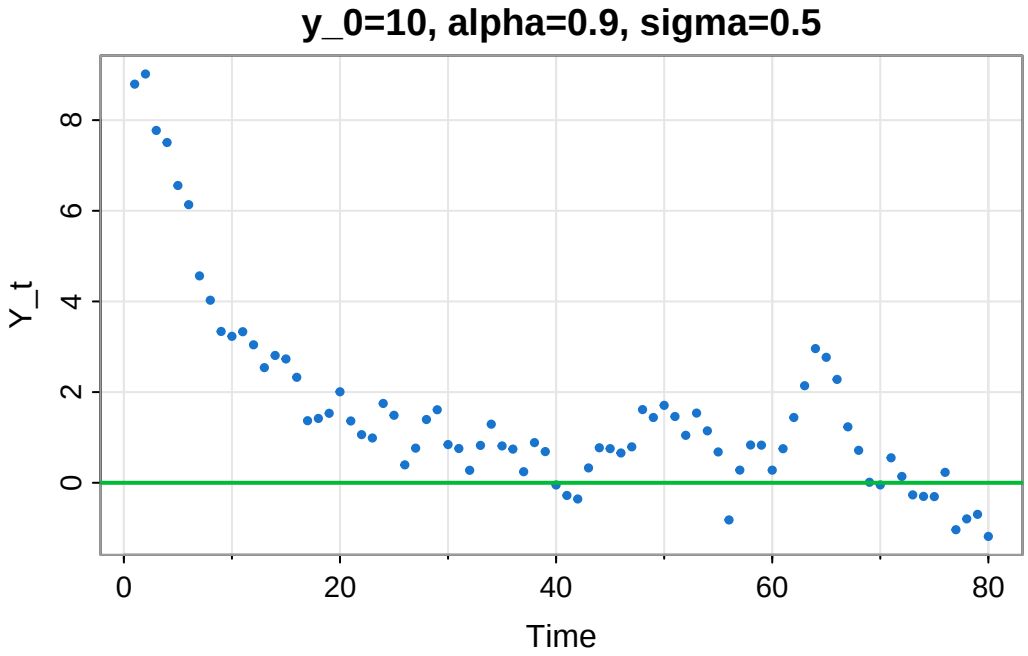
2.3.2.1 Discrete time: AR(1)

We start by looking at a first order Markovian scalar linear Gaussian auto-regressive model, commonly denoted AR(1). The basic form of the model can be written

$$Y_t = \phi Y_{t-1} + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2), \tag{2.1}$$

for some fixed $\phi \in \mathbb{R}$, noise variance $\sigma > 0$, and for now we will assume that it is initialised at some fixed $Y_0 = y_0 \in \mathbb{R}$. It is just a linear recurrence relation with some iid Gaussian noise added at each time step. It's also an appropriate auto-regressive linear filter applied to iid noise.

```r
tsplot(filter(rnorm(80, 0, 0.5), 0.9, "rec", init=10),
  type="p", col=4, pch=19, cex=0.5, ylab="Y_t",
  main="y_0=10, alpha=0.9, sigma=0.5")
abline(h=0, col=3, lwd=2)
```



Taking the expectation of Equation 2.1 gives

$$\mathbb{E}[Y_t] = \phi \mathbb{E}[Y_{t-1}]$$

In other words, the expectation satisfies a linear recurrence relation with solution

$$\mathbb{E}[Y_t] = \phi^t y_0.$$

In the stable case ($|\phi| < 1$) we will have limiting expectation 0. But this is a random process, so just knowing about its expectation isn't enough. Next take the variance of Equation 2.1

$$\mathbb{V}\text{ar}[Y_t] = \phi^2 \mathbb{V}\text{ar}[Y_{t-1}] + \sigma^2,$$

and note that the variance also satisfies a linear recurrence relation with solution

$$\mathbb{V}\text{ar}[Y_t] = \frac{1-\phi^{2t}}{1-\phi^2}\sigma^2,$$

since we are assuming here that $\mathbb{V}\text{ar}[Y_0] = 0$. So, since each update is a linear transformation of a Gaussian random variable, we know that the marginal distribution at each time must be Gaussian, and hence

$$Y_t \sim \mathcal{N}\left(\phi^t y_0, \frac{1-\phi^{2t}}{1-\phi^2}\sigma^2\right).$$

The limiting distribution (in the stable case, $|\phi| < 1$) is therefore

$$Y_\infty \sim \mathcal{N}\left(0, \frac{\sigma^2}{1-\phi^2}\right).$$

So, in the case of a stochastic model, stability implies that the process converges to some *limiting probability distribution*, and not some fixed point. But the moments of that distribution (such as the mean and variance), *are* converging to a fixed point.

If we now consider the shifted system $X_t = Y_t + \mu$, for some fixed $\mu \in \mathbb{R}$, we have

$$X_t = \mu + \phi(X_{t-1} - \mu) + \varepsilon_t = \phi X_{t-1} + (1-\phi)\mu + \varepsilon_t.$$

Now, $\mathbb{V}\mathrm{ar}[X_t] = \mathbb{V}\mathrm{ar}[Y_t]$, but

$$\mathbb{E}[X_t] = \mathbb{E}[Y_t] + \mu = \phi^t y_0 + \mu = \phi^t(x_0 - \mu) + \mu = \phi^t x_0 + (1-\phi^t)\mu,$$

so

$$X_t \sim \mathcal{N}\left(\phi^t x_0 + (1-\phi^t)\mu, \frac{1-\phi^{2t}}{1-\phi^2}\sigma^2\right).$$

Consequently, if presented with an auto-regression in the form

$$X_t = \phi X_{t-1} + k + \varepsilon_t,$$

for some fixed $k \in \mathbb{R}$, just put $k = (1-\phi)\mu$ to get

$$X_t \sim \mathcal{N}\left(\phi^t x_0 + \frac{1-\phi^t}{1-\phi}k, \frac{1-\phi^{2t}}{1-\phi^2}\sigma^2\right),$$

which obviously has limiting distribution

$$X_\infty \sim \mathcal{N}\left(\frac{k}{1-\phi}, \frac{\sigma^2}{1-\phi^2}\right).$$

2.3.2.1.1 Stationarity and further properties

Let's go back to the centred process Y_t (to keep the algebra a bit simpler) and think more carefully about what we have been doing. Everything was conditioned on the initial value of the process, $Y_0 = y_0$, so when we deduced the "marginal" distribution for Y_t, this should more correctly have been considered to be the *conditional* distribution of Y_t *given* $Y_0 = y_0$,

$$(Y_t | Y_0 = y_0) \sim \mathcal{N}\left(\phi^t y_0, \frac{1-\phi^{2t}}{1-\phi^2}\sigma^2\right).$$

This conditional distribution can be thought of as a transition kernel of the stochastic process. Being clear about this then allows us to consider other initialisation strategies, including random initialisation. In particular, suppose now that Y_0 is drawn from a normal distribution (independent of the process), $Y_0 \sim \mathcal{N}(\mu_0, \sigma_0^2)$. Since Y_0 is normal and $Y_t|Y_0$ is normal with linear dependence on Y_0, then Y_0 and Y_t are jointly normal, and so is the (true) marginal for Y_t. The mean of this marginal can be computed with the law of total expectation,

$$\mathbb{E}[Y_t] = \mathbb{E}(\mathbb{E}[Y_t|Y_0]) = \mathbb{E}(\phi^t Y_0) = \phi^t \mathbb{E}(Y_0) = \phi^t \mu_0.$$

Similarly, the variance can be computed with the law of total variance,

$$\begin{aligned}
\mathbb{V}\mathrm{ar}[Y_t] &= \mathbb{E}(\mathbb{V}\mathrm{ar}[Y_t|Y_0]) + \mathbb{V}\mathrm{ar}(\mathbb{E}[Y_t|Y_0]) \\
&= \mathbb{E}\left(\frac{1-\phi^{2t}}{1-\phi^2}\sigma^2\right) + \mathbb{V}\mathrm{ar}(\phi^t Y_0) \\
&= \frac{1-\phi^{2t}}{1-\phi^2}\sigma^2 + \phi^{2t}\sigma_0^2
\end{aligned}$$

Consequently, the marginal distribution is

$$Y_t \sim \mathcal{N}\left(\phi^t \mu_0, \frac{1-\phi^{2t}}{1-\phi^2}\sigma^2 + \phi^{2t}\sigma_0^2\right).$$

What happens if we choose the limiting distribution as the initial distribution? *ie.* if we choose $\mu_0 = 0$ and $\sigma_0^2 = \sigma^2/(1-\phi^2)$? Subbing these in to the above and simplifying gives

$$Y_t \sim \mathcal{N}\left(0, \frac{\sigma^2}{1-\phi^2}\right).$$

So, the limiting distribution is stationary in the sense that if the process is initialised with this distribution, it will keep exactly this marginal distribution for all future times. In this case, since the dynamics of the process do not depend explicitly on time and the marginal distribution is the same at each time, it is clear that the full joint distribution of the process is invariant under a time shift. There are various short-cuts one can deploy to deduce the properties of such stationary stochastic processes.

The covariance $\mathbb{C}\text{ov}(Y_s, Y_{s+t}),\ t \geq 0$ will only depend on the value of t, and will be independent of the value of s. We can call this the auto-covariance function, $\gamma_t = \mathbb{C}\text{ov}(Y_s, Y_{s+t})$. This can be exploited in order to deduce properties of the process. eg. if we start with the dynamics

$$Y_s = \phi Y_{s-1} + \varepsilon_s,$$

multiply by Y_{s-t},

$$Y_{s-t}Y_s = \phi Y_{s-t}Y_{s-1} + Y_{s-t}\varepsilon_s,$$

and take expectations

$$\mathbb{E}[Y_{s-t}Y_s] = \phi\mathbb{E}[Y_{s-t}Y_{s-1}] + \mathbb{E}[Y_{s-t}\varepsilon_s].$$

For $t > 0$ the final expectation is 0, leading to the recurrence

$$\gamma_t = \phi\gamma_{t-1},$$

with solution

$$\gamma_t = \gamma_0\phi^t = \frac{\phi^t\sigma^2}{1-\phi^2}.$$

The corresponding auto-correlation function, $\rho_t = \mathbb{C}\text{orr}(Y_s, Y_{s+t}) = \gamma_t/\gamma_0$ is

$$\rho_t = \phi^t.$$

So, the AR(1) process, in the stable case, admits a stationary distribution, but is not iid, having correlations that die away geometrically as values become further apart in time.

2.3.2.2 Vector discrete time: VAR(1)

The obvious multivariate generalisation of the AR(1) process is the first-order vector auto-regressive process, denoted VAR(1):

$$\mathbf{Y}_t = \Phi\mathbf{Y}_{t-1} + \boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(0, \Sigma), \tag{2.2}$$

for some fixed $\Phi \in M_{n\times n}$, Σ an $n \times n$ symmetric positive semi-definite (PSD) variance matrix, and again assuming a given fixed initial condition $\mathbf{Y}_0 = \mathbf{y_0} \in \mathbb{R}^n$.

Taking the expectation of Equation 2.2 we get

$$\mathbb{E}[\mathbf{Y}_t] = \Phi\mathbb{E}[\mathbf{Y}_{t-1}],$$

which is a linear recurrence with solution

$$\mathbb{E}[\mathbf{Y}_t] = \Phi^t\mathbf{y}_0.$$

In the stable case (eigenvalues of Φ inside the unit circle), this will have a limit of $\mathbf{0}$. If we now take the variance of Equation 2.2 we get

$$\mathbb{V}\text{ar}[\mathbf{Y}_t] = \Phi\mathbb{V}\text{ar}[\mathbf{Y}_{t-1}]\Phi^\top + \Sigma.$$

This is actually a kind of linear recurrence for matrices, but it's more complicated than the examples we considered earlier. However, if we let V be the solution of the discrete Lyapunov equation

$$\mathsf{V} = \mathsf{\Phi}\mathsf{V}\mathsf{\Phi}^\top + \mathsf{\Sigma},$$

we can check by direct substitution that

$$\mathbb{V}\mathrm{ar}[\mathbf{Y}_t] = \mathsf{V} - \mathsf{\Phi}^t\mathsf{V}{\mathsf{\Phi}^t}^\top$$

solves the recurrence relation. There are standard direct methods for solving Lyapunov equations (eg. `netcontrol::dlyap`). It is then clear that in the stable case the limiting variance is V. Since updates are linear transformations of multivariate normals, every marginal is normal, and so we have

$$\mathbf{Y}_t \sim \mathcal{N}\left(\mathsf{\Phi}^t\mathbf{y}_0, \mathsf{V} - \mathsf{\Phi}^t\mathsf{V}{\mathsf{\Phi}^t}^\top\right),$$

with limit

$$\mathbf{Y}_\infty \sim \mathcal{N}(\mathbf{0}, \mathsf{V}).$$

If we now consider the shifted system $\mathbf{X}_t = \mathbf{Y}_t + \boldsymbol{\mu}$, we have $\mathbb{E}[\mathbf{X}_t] = \mathsf{\Phi}^t\mathbf{x_0} + (\mathbb{I} - \mathsf{\Phi}^t)\boldsymbol{\mu}$ and $\mathbb{V}\mathrm{ar}[\mathbf{X}_t] = \mathbb{V}\mathrm{ar}[\mathbf{Y}_t]$ following very similar arguments to those we have already seen, and so

$$\mathbf{X}_t \sim \mathcal{N}\left(\mathsf{\Phi}^t\mathbf{x_0} + (\mathbb{I} - \mathsf{\Phi}^t)\boldsymbol{\mu}, \mathsf{V} - \mathsf{\Phi}^t\mathsf{V}{\mathsf{\Phi}^t}^\top\right).$$

Consequently, if presented with an auto-regressive model in the form

$$\mathbf{X}_t = \mathsf{\Phi}\mathbf{X}_{t-1} + \mathbf{k} + \boldsymbol{\varepsilon}_t,$$

just put $\mathbf{k} = (\mathbb{I} - \mathsf{\Phi})\boldsymbol{\mu}$ to get

$$\mathbf{X}_t \sim \mathcal{N}\left(\mathsf{\Phi}^t\mathbf{x_0} + (\mathbb{I} - \mathsf{\Phi}^t)(\mathbb{I} - \mathsf{\Phi})^{-1}\mathbf{k}, \mathsf{V} - \mathsf{\Phi}^t\mathsf{V}{\mathsf{\Phi}^t}^\top\right),$$

with limit

$$\mathbf{X}_\infty \sim \mathcal{N}\left((\mathbb{I} - \mathsf{\Phi})^{-1}\mathbf{k}, \mathsf{V}\right).$$

2.3.2.2.1 Stationarity and further properties

As for the scalar case, what we have deduced above is actually the transition kernel of the process conditional on an initial condition $\mathbf{Y}_0 = \mathbf{y}_0$. We can check that in the stable case if we initialise the process with the limiting distribution it is stationary. So again, in the stable case we can consider the stationary process implied by these dynamics, and use stationarity in order to deduce properties of the process.

Here we will look at the covariance function $\mathsf{\Gamma}_t = \mathbb{C}\mathrm{ov}(\mathbf{Y}_s, \mathbf{Y}_{s+t})$ (noting that $\mathsf{\Gamma}_{-t} = \mathsf{\Gamma}_t^\top$) using a similar approach as before.

$$\begin{aligned}
\mathbf{Y}_s &= \mathsf{\Phi}\mathbf{Y}_{s-1} + \boldsymbol{\varepsilon}_s \\
\Rightarrow \mathbf{Y}_s\mathbf{Y}_{s-t}^\top &= \mathsf{\Phi}\mathbf{Y}_{s-1}\mathbf{Y}_{s-t}^\top + \boldsymbol{\varepsilon}_s\mathbf{Y}_{s-t}^\top \\
\Rightarrow \mathbb{E}[\mathbf{Y}_s\mathbf{Y}_{s-t}^\top] &= \mathsf{\Phi}\mathbb{E}[\mathbf{Y}_{s-1}\mathbf{Y}_{s-t}^\top] + \mathbb{E}[\boldsymbol{\varepsilon}_s\mathbf{Y}_{s-t}^\top] \\
\Rightarrow \mathsf{\Gamma}_{-t} &= \mathsf{\Phi}\mathsf{\Gamma}_{1-t} \quad \text{for } t > 0 \\
\Rightarrow \mathsf{\Gamma}_t^\top &= \mathsf{\Phi}\mathsf{\Gamma}_{t-1}^\top \\
\Rightarrow \mathsf{\Gamma}_t &= \mathsf{\Gamma}_{t-1}\mathsf{\Phi}^\top \\
\Rightarrow \mathsf{\Gamma}_t &= \mathsf{\Gamma}_0{\mathsf{\Phi}^t}^\top \\
\Rightarrow \mathsf{\Gamma}_t &= \mathsf{V}{\mathsf{\Phi}^t}^\top
\end{aligned}$$

2.3.2.3 Continuous time: OU

Switching back to the scalar case but moving to cts time we must describe our linear Markov process using a stochastic differential equation (SDE),

$$\mathrm{d}Y(t) = -\lambda Y(t)\,\mathrm{d}t + \sigma\,\mathrm{d}W(t), \quad Y(0) = y_0, \tag{2.3}$$

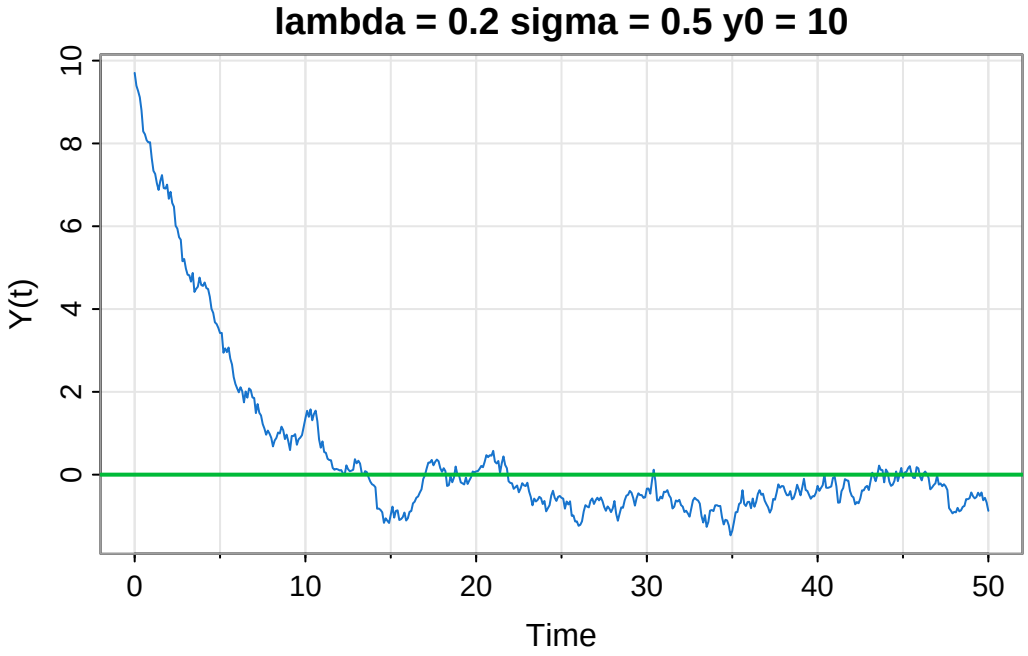
for fixed $\lambda, \sigma > 0$, known as the Ornstein–Uhlenbeck (OU) process. Here, $\mathrm{d}W(t)$ should be interpreted as increments of a Wiener process (often referred to as Brownian motion), and hence, informally, $\mathrm{d}W(t) \sim \mathcal{N}(0, \mathrm{d}t)$. We are going to be very informal in our treatment of SDEs - they are a little bit tangential to the main thrust of the course, and careful treatment gets very technical very quickly. It is helpful to write Equation 2.3 in the form

$$\begin{aligned} Y(t+\mathrm{d}t) &= Y(t) - \lambda Y(t)\,\mathrm{d}t + \sigma\,\mathrm{d}W(t) \\ &= (1-\lambda\,\mathrm{d}t)Y(t) + \sigma\,\mathrm{d}W(t), \end{aligned}$$

so then taking expectations gives

$$\mathbb{E}[Y(t+\mathrm{d}t)] = (1-\lambda\,\mathrm{d}t)\mathbb{E}[Y(t)].$$

```
T=50; dt=0.1; lambda=0.2; sigma=0.5; y0=10
times = seq(0, T, dt)
y = filter(rnorm(length(times), 0, sigma*sqrt(dt)),
      1 - lambda*dt, "rec", init=y0)
tsplot(times, y, col=4, ylab="Y(t)",
  main=paste("lambda =",lambda,"sigma =",sigma,"y0 =",y0))
abline(h=0, col=3, lwd=2)
```



Defining $m(t) = \mathbb{E}[Y(t)]$ leads to

$$m(t+\mathrm{d}t) = (1-\lambda\,\mathrm{d}t)m(t),$$

and re-arranging gives

$$\frac{m(t+\mathrm{d}t) - m(t)}{\mathrm{d}t} = -\lambda\, m(t).$$

In other words,

$$\frac{\mathrm{d}m}{\mathrm{d}t} = -\lambda m.$$

So as you might have guessed, the expectation satisfies the obvious simple linear ODE, with solution

$$m(t) = \mathbb{E}[Y(t)] = y_0 e^{-\lambda t}.$$

Now instead if we evaluate the variance we get

$$\begin{aligned}\mathbb{V}\mathrm{ar}[Y(t+\mathrm{d}t)] &= (1-\lambda\,\mathrm{d}t)^2\mathbb{V}\mathrm{ar}[Y(t)] + \sigma^2\,\mathrm{d}t \\ &= (1-2\lambda\,\mathrm{d}t)\mathbb{V}\mathrm{ar}[Y(t)] + \sigma^2\,\mathrm{d}t,\end{aligned}$$

since $\mathrm{d}t^2 \simeq 0$. Again, things might be clearer if we define $v(t) = \mathbb{V}\mathrm{ar}[Y(t)]$ to get

$$v(t+\mathrm{d}t) = (1-2\lambda\,\mathrm{d}t)v(t) + \sigma^2\,\mathrm{d}t,$$

and re-arranging gives

$$\frac{v(t+\mathrm{d}t)-v(t)}{\mathrm{d}t} = -2\lambda\,v(t) + \sigma^2.$$

In other words

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -2\lambda\,v + \sigma^2.$$

So the variance also satisfies a simple linear ODE, with solution

$$v(t) = \frac{\sigma^2}{2\lambda}(1-e^{-2\lambda t}).$$

Again, since everything is linear Gaussian, so will be the solution, so

$$Y(t) \sim \mathcal{N}\left(e^{-\lambda t}y_0, \frac{\sigma^2}{2\lambda}(1-e^{-2\lambda t})\right),$$

with limit

$$Y(\infty) \sim \mathcal{N}(0, \sigma^2/(2\lambda)).$$

It should be noted that when considered at integer times, this process is *exactly* an AR(1) with $\phi = e^{-\lambda}$, though the role of the parameter σ in the two models is slightly different.

Again, we can consider the shifted system, $X(t) = Y(t) + \mu$. We get $\mathbb{E}[X(t)] = e^{-\lambda t}x_0 + (1-e^{-\lambda_t})\mu$, $\mathbb{V}\mathrm{ar}[X(t)] = \mathbb{V}\mathrm{ar}[Y(t)]$, so

$$X(t) \sim \mathcal{N}\left(e^{-\lambda t}x_0 + (1-e^{-\lambda t})\mu, \frac{\sigma^2}{2\lambda}(1-e^{-2\lambda t})\right)$$

for the SDE

$$\mathrm{d}X(t) = -\lambda[X(t)-\mu]\,\mathrm{d}t + \sigma\,\mathrm{d}W(t).$$

2.3.2.3.1 Stationarity and further properties

Again, what we have considered above is really the transition kernel of the continuous time process. But in the stable case, we get a limiting distribution, and we can easily check that this limiting distribution is stationary. So in the stable case, the OU model determines a stationary stochastic process in continuous time. Since everything is Gaussian, it is a one-dimensional stationary Gaussian process (GP). In fact, it turns out to be the *only* stationary GP with the (first-order) Markov property. We know its mean and variance, so to fully characterise the process we just need to know its auto-covariance or auto-correlation function. We can

calculate this analogously to the discrete time case. Start by considering the dynamics of the process at time s.

$$\begin{aligned} Y(s+\mathrm{d}t) &= (1-\lambda\,\mathrm{d}t)Y(s) + \sigma\,\mathrm{d}W(s) \\ \Rightarrow Y(s-t)Y(s+\mathrm{d}t) &= (1-\lambda\,\mathrm{d}t)Y(s-t)Y(s) + \sigma Y(s-t)\,\mathrm{d}W(s) \\ \Rightarrow \gamma(t+st) &= (1-\lambda\,\mathrm{d}t)\gamma(t) \quad \text{for } t>0 \\ \Rightarrow \frac{\mathrm{d}\gamma}{\mathrm{d}t} &= -\lambda\gamma \\ \Rightarrow \gamma(t) &= \gamma(0)e^{-\lambda t} \end{aligned}$$

So $\rho(t) = e^{-\lambda t}, \forall t > 0$. Since $\rho(t) = \rho(-t)$ for scalar stationary processes, in general we have

$$\rho(t) = e^{-\lambda|t|},$$

and this correlation function completes the characterisation of an OU process as a GP.

2.3.2.4 Vector continuous time: VOU

We can now consider the obvious multivariate generalisation of the OU process, the vector OU (VOU) process,

$$\mathrm{d}\mathbf{Y}(t) = -\Lambda\mathbf{Y}(t)\,\mathrm{d}t + \Omega\,\mathrm{d}\mathbf{W}(t), \quad \mathbf{Y}(0) = \mathbf{y}_0,$$

where now fixed $\Lambda, \Omega \in M_{n\times n}$ and $\mathrm{d}\mathbf{W}(t)$ represents increments of a vector Wiener process, informally $\mathrm{d}\mathbf{W}(t) \sim \mathcal{N}(\mathbf{0}, \mathbb{I}\mathrm{d}t)$. Again, it is helpful to re-write the SDE, informally, as

$$\mathbf{Y}(t+\mathrm{d}t) = (\mathbb{I} - \Lambda\,\mathrm{d}t)\mathbf{Y}(t) + \Omega\,\mathrm{d}\mathbf{W}(t).$$

Taking expectations and defining $\mathbf{m}(t) = \mathbb{E}[\mathbf{Y}(t)]$ leads, unsurprisingly, to the ODE

$$\frac{\mathrm{d}\mathbf{m}}{\mathrm{d}t} = -\Lambda\mathbf{m},$$

with solution

$$\mathbf{m}(t) = \mathbb{E}[\mathbf{Y}(t)] = \exp\{-\Lambda t\}\mathbf{y}_0.$$

Stability of the fixed point of $\mathbf{0}$ will require the real parts of the eigenvalues of Λ to be positive.

Taking the variance gives

$$\mathbb{V}\mathrm{ar}[\mathbf{Y}(t+\mathrm{d}t)] = (\mathbb{I} - \Lambda\,\mathrm{d}t)\mathbb{V}\mathrm{ar}[\mathbf{Y}(t)](\mathbb{I} - \Lambda\,\mathrm{d}t)^\top + \Omega\Omega^\top\mathrm{d}t.$$

Putting $\mathsf{V}(t) = \mathbb{V}\mathrm{ar}[\mathbf{Y}(t)]$ and $\Sigma = \Omega\Omega^\top$ leads to the linear matrix ODE

$$\frac{\mathrm{d}\mathsf{V}}{\mathrm{d}t} = -\Lambda\mathsf{V} - \mathsf{V}\Lambda^\top + \Sigma.$$

Even without explicitly solving, it is clear that at the fixed point we will have the continuous Lyapunov equation

$$\Lambda\mathsf{V}_\infty + \mathsf{V}_\infty\Lambda^\top = \Sigma.$$

Again, there are standard direct methods for solving this for V_∞ (eg. `maotai::lyapunov`). Analogously with the discrete time case, we can verify by direct substitution that the solution to the variance ODE is given by

$$\mathsf{V}(t) = \mathsf{V}_\infty - \exp\{-\Lambda t\}\mathsf{V}_\infty\exp\{-\Lambda t\}^\top.$$

Everything is linear and multivariate normal, so the marginal is too, giving

$$\mathbf{Y}(t) \sim \mathcal{N}\left(\exp\{-\Lambda t\}\mathbf{y}_0\,, \mathsf{V}_\infty - \exp\{-\Lambda t\}\mathsf{V}_\infty\exp\{-\Lambda t\}^\top\right),$$

with limiting distribution

$$\mathbf{Y}(\infty) \sim \mathcal{N}(\mathbf{0}, \mathsf{V}_\infty)$$

in the stable case. We can analyse the shifted system $\mathbf{X}(t) = \mathbf{Y}(t) + \boldsymbol{\mu}$ as we have seen previously to obtain

$$\mathbf{X}(t) \sim \mathcal{N}\left(\exp\{-\mathsf{\Lambda} t\}\mathbf{y}_0 + (\mathbb{I} - \exp\{-\mathsf{\Lambda} t\})\boldsymbol{\mu}\,, \mathsf{V}_\infty - \exp\{-\mathsf{\Lambda} t\}\mathsf{V}_\infty \exp\{-\mathsf{\Lambda} t\}^\top\right)$$

as the marginal for the SDE

$$\mathrm{d}\mathbf{X}(t) = -\mathsf{\Lambda}[\mathbf{X}(t) - \boldsymbol{\mu}]\mathrm{d}t + \Omega\,\mathrm{d}\mathbf{W}(t).$$

2.3.2.4.1 Stationarity and further properties

Again, we have so far characterised the transition kernel of the VOU process. But in the stable case, we can again check that the limiting distribution is stationary, and so the stable VOU process admits a stationary solution. This stationary solution is a vector-valued one-dimensional GP, and since we know its mean and variance, we just need the auto-covariance function to complete its GP characterisation. Again, we start with the dynamics at time s.

$$\begin{aligned}
\mathbf{Y}(s + \mathrm{d}t) &= (\mathbb{I} - \mathsf{\Lambda}\,\mathrm{d}t)\mathbf{Y}(s) + \Omega\,\mathrm{d}\mathbf{W}(s) \\
\Rightarrow \mathbf{Y}(s + \mathrm{d}t)\mathbf{Y}(s - t)^\top &= (\mathbb{I} - \mathsf{\Lambda}\,\mathrm{d}t)\mathbf{Y}(s)\mathbf{Y}(s - t)^\top + \Omega\,\mathrm{d}\mathbf{W}(s)\mathbf{Y}(s - t)^\top \\
\Rightarrow \mathsf{\Gamma}(-t - \mathrm{d}t) &= (\mathbb{I} - \mathsf{\Lambda}\,\mathrm{d}t)\mathsf{\Gamma}(-t), \quad \text{for } t > 0 \\
\Rightarrow \mathsf{\Gamma}(t + \mathrm{d}t)^\top &= (\mathbb{I} - \mathsf{\Lambda}\,\mathrm{d}t)\mathsf{\Gamma}(t)^\top \\
\Rightarrow \mathsf{\Gamma}(t + \mathrm{d}t) &= \mathsf{\Gamma}(t)(\mathbb{I} - \mathsf{\Lambda}^\top\,\mathrm{d}t) \\
\Rightarrow \frac{\mathrm{d}\mathsf{\Gamma}}{\mathrm{d}t} &= -\mathsf{\Gamma}\mathsf{\Lambda}^\top \\
\Rightarrow \mathsf{\Gamma}(t) &= \mathsf{\Gamma}(0)\exp\{-\mathsf{\Lambda}^\top t\} = \mathsf{V}_\infty \exp\{-\mathsf{\Lambda} t\}^\top.
\end{aligned}$$

2.4 Second order systems

We will just very briefly consider second-order generalisations of the first-order models that we have analysed. As we shall see, we can directly analyse a second-order model, but we can also convert a second-order model to a first-order model with extended state space, so understanding the first-order case is in some sense sufficient.

2.4.1 Deterministic

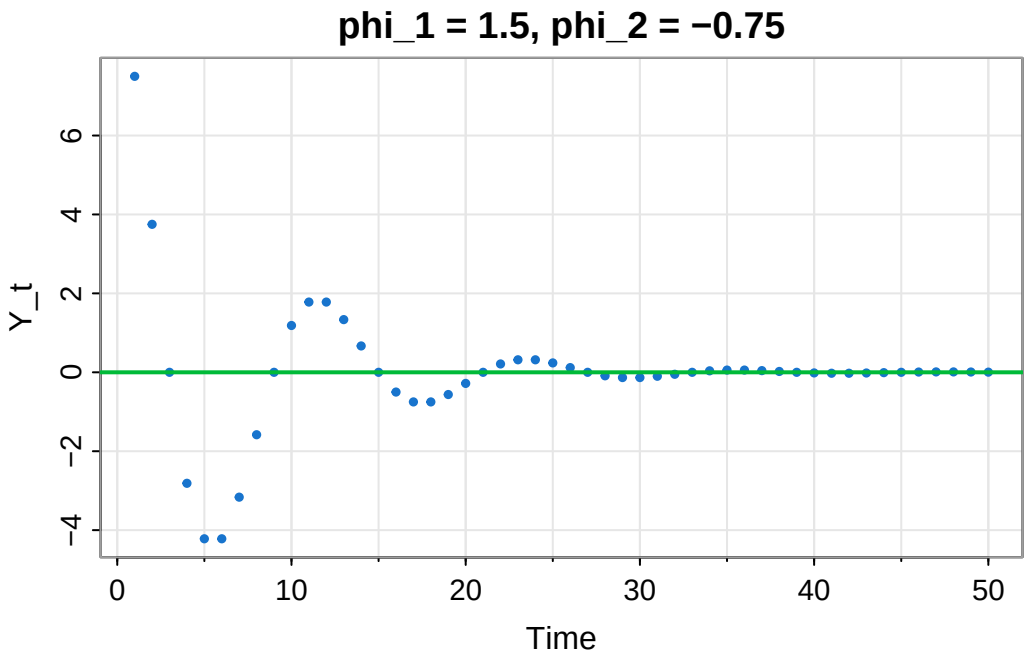
2.4.1.1 Discrete time

Consider the linear recursion

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2}, \quad t = 2, 3, \ldots$$

initialised with $y_0, y_1 \in \mathbb{R}$.

```
tsplot(filter(rep(0,50), c(1.5, -0.75), "rec", init=c(10,10)),
  type="p", pch=19, col=4, cex=0.5, ylab="Y_t",
  main="phi_1 = 1.5, phi_2 = -0.75")
abline(h=0, col=3, lwd=2)
```



We could try solving this directly by looking for solutions of the form $y_t = A\lambda^t$. Substituting in leads to the quadratic

$$\lambda^2 - \phi_1\lambda - \phi_2 = 0.$$

This will have two solutions, λ_1, λ_2, leading to a general solution of the form

$$y_t = A_1\lambda_1^t + A_2\lambda_2^t,$$

(assuming that $\lambda_1 \neq \lambda_2$), and the two initial values y_0, y_1 can be used to deduce the values of A_1, A_2. It is clear that this solution will be stable provided that $|\lambda_i| < 1$, $i = 1, 2$.

Which combinations of ϕ_1 and ϕ_2 will lead to stable systems? Since we have a condition on λ_i and the λ_i are just a function of ϕ_1 and ϕ_2, we can figure out the region of ϕ_1–ϕ_2 space corresponding to stable solutions. Careful analysis of the relevant inequalities reveals that the stable region is the triangular region defined by

$$\phi_2 < 1 - \phi_1, \quad \phi_2 < 1 + \phi_1, \quad \phi_2 > -1.$$

Within this triangular region, λ_i will be complex if $\phi_1^2 + 4\phi_2 < 0$, and this will correspond to the oscillatory stable region.

We can consider a shifted version ($x_t = y_t + \mu$), as before.

This is all fine, but an alternative is to re-write the system as a first order vector recurrence

$$\begin{pmatrix} y_t \\ y_{t-1} \end{pmatrix} = \begin{pmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} y_{t-1} \\ y_{t-2} \end{pmatrix}.$$

We know that this system will be stable provided that the eigenvalues of the update matrix lie inside the unit circle. We can check that the eigenvalues of this matrix are given by the solutions of the quadratic in λ considered above. This technique of taking a higher-order linear dynamical system and expressing it as a first order system with a higher-dimensional state space can be applied to all of the models that we have been considering: deterministic or stochastic, discrete or cts time.

2.4.1.2 Vector discrete time

The second-order vector discrete time linear recursion

$$\mathbf{y}_t = \Phi_1 \mathbf{y}_{t-1} + \Phi_2 \mathbf{y}_{t-2}$$

can be re-written as the first order system

$$\begin{pmatrix} \mathbf{y}_t \\ \mathbf{y}_{t-1} \end{pmatrix} = \begin{pmatrix} \Phi_1 & \Phi_2 \\ \mathbb{I} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{y}_{t-1} \\ \mathbf{y}_{t-2} \end{pmatrix}.$$

2.4.1.3 Continuous time

The scalar second-order linear system in cts time can be written

$$\ddot{y} + \nu_1 \dot{y} + \nu_2 y = 0,$$

and initialised with $y(0)$ and $\dot{y}(0)$. Here, when compared to the first-order equivalent, there has been a switch of parameter from λ to ν_1, ν_2 in order to avoid confusion with eigenvalues. As with the discrete time system, we could directly analyse this system by looking for solutions of the form $y = Ae^{\lambda t}$. Substituting in leads to the quadratic

$$\lambda^2 + \nu_1 \lambda + \nu_2 = 0.$$

This can be solved for the two solutions λ_1, λ_2 to get a general solution of the form

$$y(t) = A_1 e^{\lambda_1 t} + A_2 e^{\lambda_2 t}$$

(assuming that $\lambda_1 \neq \lambda_2$), and the initial conditions can be used to solve for A_1, A_2. It is clear that this system will be stable provided that the real parts of λ_1, λ_2 are both negative.

Alternatively, by defining $v \equiv \dot{y}$, this second-order system could be re-written as

$$\dot{v} + \nu_1 v + \nu_2 y = 0,$$

and hence as the first order vector system,

$$\begin{pmatrix} \dot{v} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\nu_1 & -\nu_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} v \\ y \end{pmatrix}.$$

We know that this system will be stable provided that the generator matrix has eigenvalues with negative real parts. Solving for the eigenvalues leads to the same quadratic in λ we deduced above.

We can consider the stable region in terms of ν_1 and ν_2. Again, careful analysis of the relevant inequalities reveals that the stable region corresponds to the positive quadrant $\nu_1, \nu_2 > 0$, and that the roots will be complex if $4\nu_2 > \nu_1^2$.

2.4.1.4 Vector continuous time

The second-order vector linear system

$$\ddot{\mathbf{y}} + \Lambda_1 \dot{\mathbf{y}} + \Lambda_2 \mathbf{y} = \mathbf{0},$$

can be re-written as the first-order system

$$\begin{pmatrix} \dot{\mathbf{v}} \\ \dot{\mathbf{y}} \end{pmatrix} = \begin{pmatrix} -\Lambda_1 & -\Lambda_2 \\ \mathbb{I} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{v} \\ \mathbf{y} \end{pmatrix},$$

where $\mathbf{v} \equiv \dot{\mathbf{y}}$.

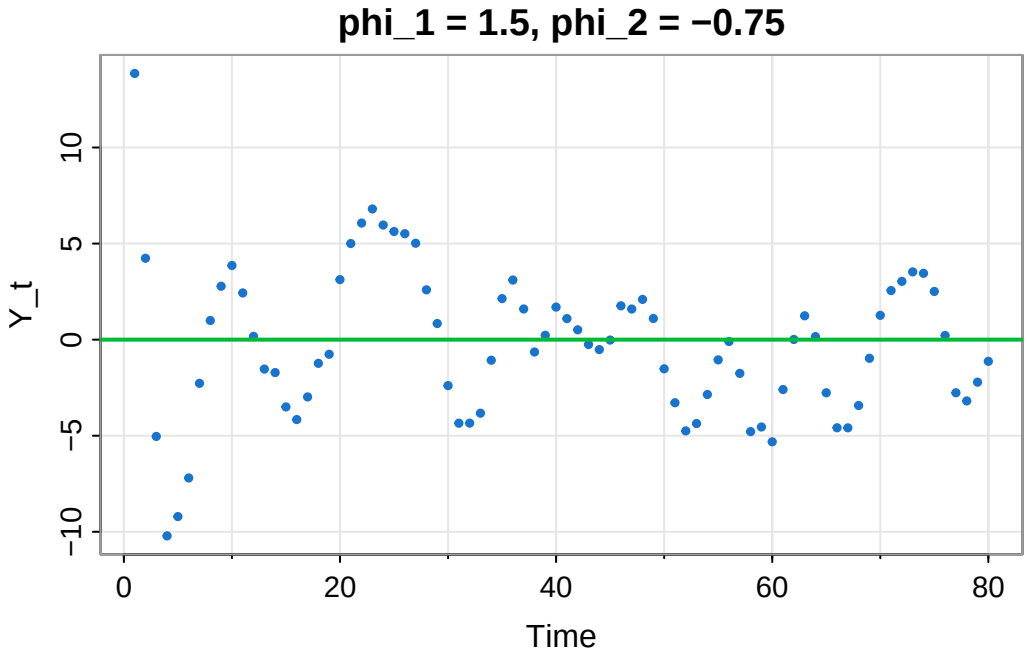
2.4.2 Stochastic

2.4.2.1 Discrete time: AR(2)

The second-order Markovian scalar linear Gaussian auto-regressive model, AR(2), takes the form

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2).$$

```
tsplot(filter(rnorm(80), c(1.5, -0.75), "rec", init=c(20,20)),
  type="p", pch=19, col=4, cex=0.5, ylab="Y_t",
  main="phi_1 = 1.5, phi_2 = -0.75")
abline(h=0, col=3, lwd=2)
```



Taking expectations gives the second-order linear recurrence

$$\mathbb{E}[Y_t] = \phi_1 \mathbb{E}[Y_{t-1}] + \phi_2 \mathbb{E}[Y_{t-2}].$$

Looking for solutions of the form $\mathbb{E}[Y_t] = A\lambda^t$ gives the quadratic

$$\lambda^2 - \phi_1 \lambda - \phi_2 = 0,$$

so the system will be stable if the solutions, λ_i, are inside the unit circle ($|\lambda_i| < 1$). We have already described this stable region in ϕ_1–ϕ_2 space, while examining the deterministic second-order linear recurrence.

Alternatively, we could re-write this second-order system as the VAR(1):

$$\begin{pmatrix} Y_t \\ Y_{t-1} \end{pmatrix} = \begin{pmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} Y_{t-1} \\ Y_{t-2} \end{pmatrix} + \begin{pmatrix} \varepsilon_t \\ 0 \end{pmatrix}.$$

We know that this system will be stable if the eigenvalues of the auto-regressive matrix are inside the unit circle. But solving for the eigenvalues leads to the same quadratic that we deduced above. Also note that here the error variance matrix is $\begin{pmatrix} \sigma^2 & 0 \\ 0 & 0 \end{pmatrix}$.

2.4.2.2 Vector discrete time: VAR(2)

The VAR(2) model

$$\mathbf{Y}_t = \Phi_1 \mathbf{Y}_{t-1} + \Phi_2 \mathbf{Y}_{t-2} + \boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim N(\mathbf{0}, \Sigma)$$

can be re-written as the VAR(1) system

$$\begin{pmatrix} \mathbf{Y}_t \\ \mathbf{Y}_{t-1} \end{pmatrix} = \begin{pmatrix} \Phi_1 & \Phi_2 \\ \mathbb{I} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{Y}_{t-1} \\ \mathbf{Y}_{t-2} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\varepsilon}_t \\ \mathbf{0} \end{pmatrix}.$$

It will be stable when the eigenvalues of the auto-regressive matrix lie inside the unit circle.

2.4.2.3 Continuous time

We wrote the first-order continuous time linear system in the form

$$\mathrm{d}Y(t) = -\lambda Y(t)\,\mathrm{d}t + \sigma\,\mathrm{d}W(t),$$

where $W(t)$ is a Wiener process, but a physicist might just write it as a Langevin equation:

$$\dot{Y}(t) + \lambda Y(t) = \sigma\eta(t),$$

where $\eta(t)$ is a "white noise" process. This requires some care in interpreting, since $Y(t)$ is not actually differentiable, and $\eta(t)$ is not a regular function. However, this way of writing it makes the second-order generalisation more obvious:

$$\ddot{Y}(t) + \nu_1\dot{Y}(t) + \nu_2 Y(t) = \sigma\eta(t),$$

where, again, we've switched from λ to ν_i to avoid confusion with eigenvalues. If we throw caution to the wind, take expectations, and put $m(t) = \mathbb{E}[Y(t)]$ we get

$$\ddot{m} + \nu_1\dot{m} + \nu_2 = 0.$$

We know that seeking solutions of the form $m(t) = Ae^{\lambda t}$ will lead to the quadratic

$$\lambda^2 + \nu_1\lambda + \nu_2 = 0$$

and that the system will be stable provided that the solutions, λ_i, both have negative real part.

If we now try to be a bit more careful, and define $V(t) = \dot{Y}(t)$, we can re-write the Langevin equation as

$$\dot{V}(t) + \nu_1 V(t) + \nu_2 Y(t) = \sigma\eta(t),$$

and then it is reasonably clear that we can write the system as a pair of coupled (S)DEs,

$$\begin{aligned} \mathrm{d}V(t) &= -(\nu_1 V(t) + \nu_2 Y(t))\mathrm{d}t + \sigma\,\mathrm{d}W(t) \\ \mathrm{d}Y(t) &= V(t)\,\mathrm{d}t, \end{aligned}$$

and hence as a first-order vector OU process

$$\begin{pmatrix} \mathrm{d}V(t) \\ \mathrm{d}Y(t) \end{pmatrix} = \begin{pmatrix} -\nu_1 & -\nu_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} V(t) \\ Y(t) \end{pmatrix} \mathrm{d}t + \begin{pmatrix} \sigma & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathrm{d}W_{1,t} \\ \mathrm{d}W_{2,t} \end{pmatrix}.$$

We know that this VOU process will be stable if the eigenvalues of the generator matrix have negative real parts, but of course, solving for the eigenvalues leads to the quadratic we deduced above.

2.4.2.4 Vector continuous time

Using "physics notation", we could informally write the second-order system as

$$\ddot{\mathbf{Y}}(t) + \Lambda_1 \dot{\mathbf{Y}}(t) + \Lambda_2 \mathbf{Y}(t) = \Omega \boldsymbol{\eta}(t),$$

understanding that what this really represents is the VOU process

$$\begin{pmatrix} \mathrm{d}\mathbf{V}(t) \\ \mathrm{d}\mathbf{Y}(t) \end{pmatrix} = \begin{pmatrix} -\Lambda_1 & -\Lambda_2 \\ \mathbb{I} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}(t) \\ \mathbf{Y}(t) \end{pmatrix} \mathrm{d}t + \begin{pmatrix} \Omega & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathrm{d}\mathbf{W}_{1,t} \\ \mathrm{d}\mathbf{W}_{2,t} \end{pmatrix},$$

where $\mathbf{V}(t) = \dot{\mathbf{Y}}(t)$. Stability can be determined by checking if the eigenvalues of the generator matrix all have negative real part.

3 ARMA models

3.1 Introduction

In this chapter we will consider a method for building a flexible class of stationary Gaussian processes for discrete-time real-valued time series. Such processes will rarely be appropriate for directly fitting to real observed time series data, but will be extremely useful for modelling the "residuals" of a time series after taking out some systematic, and/or seasonal effect, and/or detrending in some way. We have seen that such residuals can often be assumed to be mean zero and stationary, but with a non-trivial correlation structure. Autoregressive–moving-average (ARMA) models are good solution to this problem.

An ARMA model is simply the stochastic process obtained by applying an ARMA (or IIR) filter to a Gaussian white noise process. That is, the stochastic process

$$\dots, X_{-2}, X_{-1}, X_0, X_1, X_2, \dots$$

is ARMA(p, q) if it is determined by the filter

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}, \tag{3.1}$$

applied to iid Gaussian white noise $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$, for some fixed p-vector $\boldsymbol{\phi}$ and q-vector $\boldsymbol{\theta}$. Since the filter is linear, it follows that the ARMA model is a discrete-time Gaussian process. It should be immediately clear that this process generalises the AR(1) and AR(2) processes considered in Chapter 2. However, this connection with the AR(1) and AR(2) should immediately raise questions about stationarity. We have seen that even for these simple special cases, parameters must be carefully chosen in order to ensure that the process is stable and will converge to a stationary distribution. But we have also seen that even in this stable case, if the process is not initialised with draws from the stationary distribution, it will not be stationary, at least until convergence is asymptotically reached.

3.1.1 Stationarity

In the context of time series analysis, we typically want to use ARMA models as a convenient way of describing a stationary discrete-time Gaussian process. We therefore want to restrict our attention to parameter regimes where the process is stable and converges to a stationary distribution. We will need to be careful to check that this is the case. We also assume that the process has been running from $t = -\infty$, or that it is carefully initialised with draws from the (joint) stationary distribution, so that there is no *transient phase* of the process that needs to be considered.

At this point it might be helpful to more carefully define stationarity in the context of time series.

A discrete-time stochastic process is ***strictly stationary*** *iff the joint distribution of*

$$X_t, X_{t+1}, \dots X_{t+k}$$

is identical to the joint distribution of

$$X_{t+l}, X_{t+l+1}, \dots X_{t+l+k}, \quad \forall t, k, l \in \mathbb{Z}.$$

In particular, but not only, it implies that the marginal distribution of X_t is independent of t. This condition is quite strong, and hard to verify in practice, so there is also a weaker notion of stationarity that is useful.

*A discrete-time stochastic process is **weakly stationary** (AKA **second-order stationary**) if:*

- $\mathbb{E}[X_t] = \mathbb{E}[X_s] = \mu, \quad \forall s, t \in \mathbb{Z}$
- $\mathbb{V}\text{ar}[X_t] = \mathbb{V}\text{ar}[X_s] = v, \quad \forall s, t \in \mathbb{Z}$
- $\mathbb{C}\text{ov}[X_t, X_{t+k}] = \mathbb{C}\text{ov}[X_s, X_{s+k}] = \gamma_k, \quad \forall s, t, k \in \mathbb{Z}.$

In other words, the mean and variance are constant, and the covariance between two values depends only on the lag. It is clear that (in the case of a process with finite mean and variance) the property of weak stationarity follows from strict stationary, hence the name. However, a Gaussian process is determined by its first two moments, and so for Gaussian processes the concepts of strict and weak stationarity coincide, and we will refer to such processes as being **stationary**, or not, without needing to qualify.

The sequence $\gamma_k,\ k \in \mathbb{Z}$ is known as the *auto-covariance* function of a weakly stationary stochastic process. Note that $\gamma_{-k} = \gamma_k$. It is often also convenient to work with the corresponding *auto-correlation* function,

$$\rho_k \equiv \mathbb{C}\text{orr}[X_t, X_{t+k}] = \gamma_k/\gamma_0, \quad \forall k \in \mathbb{Z}.$$

Let us now get a better feel for stationary ARMA models by looking at some important special cases.

3.2 AR(p)

The Autoregressive model model of order p, denoted AR(p) or ARMA($p, 0$), is defined by

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2), \quad \forall t \in \mathbb{Z}. \tag{3.2}$$

The process is determined by fixed $\boldsymbol{\phi} \in \mathbb{R}^p, \sigma \in \mathbb{R}^+$. We briefly examined the special cases AR(1) and AR(2) in Chapter 2, but we now tackle the general case. It is crucial to notice that, by construction, ε_t will be independent of X_s for $t > s$, but not otherwise (since then ε_t will have been involved in some way in the construction of X_s). This property also holds in the more general case, and will be repeatedly exploited in the analysis of ARMA models.

The first thing to consider is whether the particular choice of $\boldsymbol{\phi}$ corresponds to a stable, and hence stationary, model. There are many ways one could try to understand this, but using the same approach as we used in Chapter 2, we can write the model as the p-dimensional VAR(1) model

$$\begin{pmatrix} X_t \\ X_{t-1} \\ \vdots \\ X_{t-p+1} \end{pmatrix} = \begin{pmatrix} \phi_1 & \cdots & \phi_{p-1} & \phi_p \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix} \begin{pmatrix} X_{t-1} \\ X_{t-2} \\ \vdots \\ X_{t-p} \end{pmatrix} + \begin{pmatrix} \varepsilon_t \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

If we call the $p \times p$ autoregressive matrix Φ, then we know from our analysis of the stability of VAR(1) models that we will get a (stable) stationary model provided that all of the eigenvalues of Φ lie inside the unit circle in the complex plane. If we attempt to find the eigenvalues by carrying out Laplace expansion on the equation $|\Phi - \lambda \mathbb{I}| = 0$, we arrive at the polynomial equation

$$\lambda^p - \sum_{k=1}^{p} \phi_k \lambda^{p-k} = 0.$$

So, we want all of the solutions in λ to have modulus *less* than one.

Aside: *Note that we have established a connection between the roots of an arbitrary polynomial and the eigenvalues of a specially constructed matrix. So, if we have a numerical eigenvalue solver at our disposal,*

we could use it to numerically determine all of the roots of an arbitrary polynomial in a very straightforward way.

If we now make the substitution $u = \lambda^{-1}$ we get the polynomial equation

$$1 - \sum_{k=1}^{p} \phi_k u^k = 0,$$

and so we want all of the solutions to this equation to have modulus *greater* than one. The polynomial on the LHS of this equation crops up frequently in the analysis of AR models, so it has a name. Our model will be *stationary* provided that all roots of the *characteristic polynomial*

$$\phi(u) = 1 - \sum_{k=1}^{p} \phi_k u^k$$

have modulus *greater* than one, and hence lie *outside* of the unit circle in the complex plane.

We will now assume that we are considering only the stationary case, and try to understand other characteristics of this model. Since we know that it is a stationary Gaussian process, we know that it is characterised by its mean and auto-covariance function. Let's start with the mean.

It should be obvious, either by symmetry, or by analogy with the AR(1) and AR(2), that the only possible stationary mean for this process is zero. However, it is instructive to explicitly compute it. Taking the expectation of Equation 3.2 gives

$$\begin{aligned} \mathbb{E}[X_t] &= \sum_{k=1}^{p} \phi_k \mathbb{E}[X_{t-k}] \\ \Rightarrow \mu &= \sum_{k=1}^{p} \phi_k \mu \\ \Rightarrow \left(1 - \sum_{k=1}^{p} \phi_k\right) \mu &= 0 \\ \Rightarrow \phi(1)\mu &= 0 \end{aligned}$$

Now, since we are assuming a stationary process, the roots of $\phi(u)$ are outside of the unit circle, and hence $\phi(1) \neq 0$, giving $\mu = 0$ as anticipated (in the stationary case).

Next we consider the auto-covariance function.

3.2.1 The Yule-Walker equations

If we multiply Equation 3.2 by X_{t-k} for $k > 0$ and take expectations we get

$$\mathbb{E}[X_{t-k}X_t] = \sum_{i=1}^{p} \phi_i \mathbb{E}[X_{t-k}X_{t-i}],$$

using the fact that $\mathbb{E}[X_{t-k}\varepsilon_t] = \mathbb{E}[X_{t-k}]\mathbb{E}[\varepsilon_t] = 0$. In other words,

$$\gamma_k = \sum_{i=1}^{p} \phi_i \gamma_{k-i}, \quad k > 0. \tag{3.3}$$

So, the auto-covariance function satisfies a linear recurrence. We can either use this linear recurrence directly in order to generate auto-covariances, or seek a general solution. But either way, we will need to know $\gamma_0, \gamma_1, \ldots, \gamma_{p-1}$ in order to initialise the recursion or fix a particular solution. If we consider the above equations for $k = 1, 2, \ldots, p$, and remember that $\gamma_{-k} = \gamma_k$, we have p equations in the $p + 1$ unknowns $\gamma_0, \gamma_1, \ldots, \gamma_p$. So, we need one more linear constraint. There are two commonly adopted ways to proceed

from here. Either way, it is useful to consider the case $k = 0$. That is, multiply Equation 3.2 by X_t and take expectations. First we need to compute

$$\mathbb{E}[X_t\varepsilon_t] = \mathbb{C}\text{ov}[X_t, \varepsilon_t] = \mathbb{C}\text{ov}\left[\sum_i \phi_i X_{t-i} + \varepsilon_t, \varepsilon_t\right] = \mathbb{C}\text{ov}[\varepsilon_t, \varepsilon_t] = \mathbb{V}\text{ar}[\varepsilon_t] = \sigma^2.$$

Then we get

$$\gamma_0 = \sum_{i=1}^{p} \phi_i\gamma_i + \sigma^2.$$

This gives us the $(p+1)$th equation that we need to solve for $\gamma_0, \gamma_1, \ldots, \gamma_p$. There is a related but slightly different approach, where we divide Equation 3.3 by γ_0 to get

$$\rho_k = \sum_{i=1}^{p} \phi_i\rho_{k-i}, \quad k = 1, 2, \ldots, p,$$

which we regard as p equations in p unknowns, since we know that $\rho_0 = 1$. These are the equations most commonly referred to as the *Yule-Walker equations*, though conventions vary. But if we adopt this approach we still need to know the stationary variance $v = \gamma_0$ in order to complete the characterisation of the process. For that we just divide the above equation for γ_0 through by γ_0 and re-arrange to get

$$v = \gamma_0 = \frac{\sigma^2}{1 - \sum_{i=1}^{p} \phi_i\rho_i}.$$

The above approach to solving for the particular solution of the auto-covariance function using the Yule-Walker equations is very convenient when solving by hand in the case of an AR(1), AR(2) or AR(3), say, but not especially convenient when solving on a computer in the general case. For this it is more convenient to revert to our formulation in terms of a VAR(1) model, with the auto-regressive matrix Φ already described, and innovation variance matrix

$$\Sigma = \begin{pmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

We then know from our analysis in Chapter 2 that the stationary variance matrix is given by the solution, V, to

$$\mathsf{V} = \Phi\mathsf{V}\Phi^\top + \Sigma,$$

and that there are standard direct methods to solve this discrete Lyapunov equation (eg. `netcontrol::dlyap`). The first row (or column) of V is $\gamma_0, \gamma_1, \ldots, \gamma_{p-1}$, as required.

Once we have the first p auto-covariances, we can use Equation 3.3 to generate more auto-covariances, as required. Alternatively, we can seek an explicit solution for γ_k in terms of k (only), using standard techniques for solving linear recursions. If we seek solution of the form $\gamma_k = Au^{-k}$ and substitute this in to Equation 3.3 we find that the u we require are roots of the characteristic polynomial $\phi(u)$. If we call the roots $u_1, \ldots, u_p$, then our general solution will be of the form

$$\gamma_k = \sum_{i=1}^{p} A_i u_i^{-k},$$

and we can use our first p auto-covariances to determine $A_1, \ldots, A_p$.

3.2.2 The backshift operator

Although it is not strictly necessary, it is often quite convenient to analyse and manipulate ARMA (and other discrete time) models using the backshift operator (AKA the *lag operator*).

*The **backshift operator**, B, has the property*

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

It follows that $B^2 x_t = BBx_t = Bx_{t-1} = x_{t-2}$, and hence $B^k x_t = x_{t-k}$. Then, for consistency, we have $B^{-1}x_t = x_{t+1}$, the forward shift operation, and $B^0 x_t = x_t$, the identity. Since it is a linear operator, it can be manipulated algebraically like other operators. It is very much analogous to the differential operator, D, often used in the analysis of (second-order) ODEs. We can use the backshift operator to simplify the description of an AR(p) model as follows.

$$\begin{aligned}
X_t &= \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t \\
\Rightarrow X_t - \sum_{i=1}^{p} \phi_i X_{t-i} &= \varepsilon_t \\
\Rightarrow X_t - \sum_{i=1}^{p} \phi_i B^i X_t &= \varepsilon_t \\
\Rightarrow \left(1 - \sum_{i=1}^{p} \phi_i B^i\right) X_t &= \varepsilon_t \\
\Rightarrow (1 - \sum_{i=1}^{p} \phi_i B^i) X_t &= \varepsilon_t \\
\Rightarrow \phi(B) X_t &= \varepsilon_t,
\end{aligned}$$

where $\phi(B)$ is the characteristic polynomial evaluated with B. The linear operator $\phi(B)$ is known as the *autoregressive operator*.

We can understand some of the algebraic properties of the backshift operator by first expanding an AR(1) with successive substitution.

$$\begin{aligned}
X_t &= \phi X_{t-1} + \varepsilon_t \\
&= \phi(\phi X_{t-2} + \varepsilon_{t-1}) + \varepsilon_t \\
&= \dots \\
&= \sum_{k=0}^{\infty} \phi^k \varepsilon_{t-k} \\
&= \sum_{k=0}^{\infty} \phi^k B^k \varepsilon_t \\
&= \left(\sum_{k=0}^{\infty} (\phi B)^k\right) \varepsilon_t.
\end{aligned}$$

In other words, the model

$$(1 - \phi B) X_t = \varepsilon_t$$

is exactly equivalent to the model

$$X_t = (1 - \phi B)^{-1} \varepsilon_t, \tag{3.4}$$

since

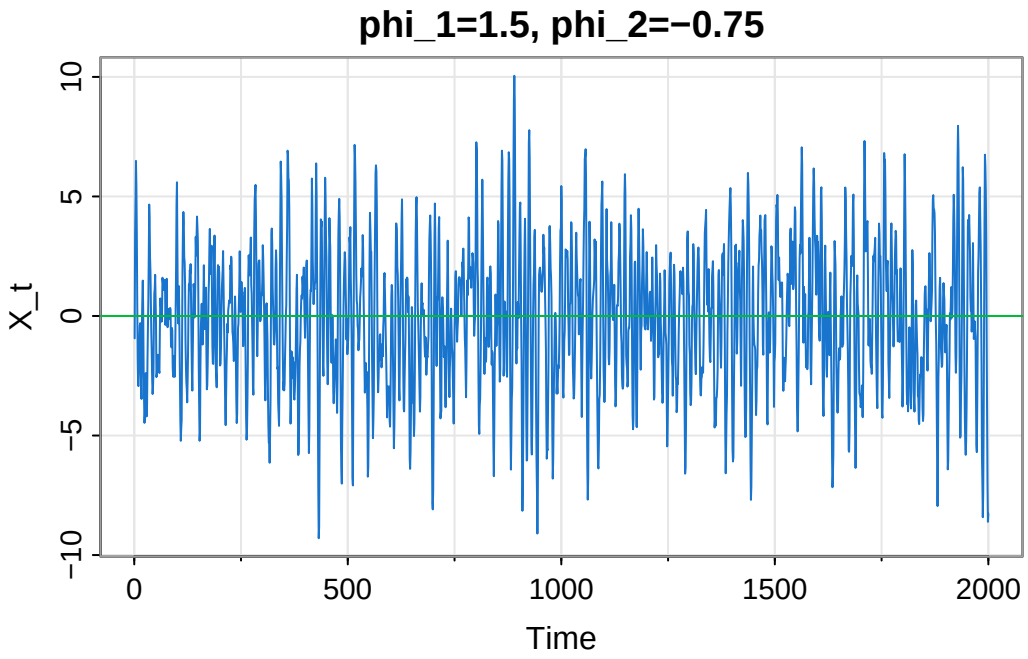
$$(1 - \phi B)^{-1} = \sum_{k=0}^{\infty} (\phi B)^k = 1 + \phi B + \phi^2 B^2 + \cdots.$$

For reasons to become clear, Equation 3.4 is known as the MA(∞) representation of the AR(1) process.

3.2.3 Example: AR(2)

Let's go back to the AR(2) example considered very briefly in Chapter 2.

```
library(astsa)
set.seed(42)
phi = c(1.5, -0.75); sigma = 1; n = 2000; burn = 100
x = filter(rnorm(n+burn, 0, sigma), phi, "rec")[-(1:burn)]
tsplot(x, col=4, cex=0.5, ylab="X_t",
  main=paste0("phi_1=", phi[1], ", phi_2=", phi[2]))
abline(h=0, col=3)
```



Although in Chapter 2 we briefly switched to plotting discrete-time series with points (to emphasise the distinction between discrete and cts time), we now revert to the more usual convention of joining successive observations with straight line segments.

This model has $\phi_1 = 3/2,\ \phi_2 = -3/4, \sigma = 1$. In principle this completely characterises the process, but on their own, these parameters are not especially illuminating. The plot *suggests* that this model is probably stationary, but isn't especially conclusive, so we should certainly check.

To check by hand, we want to find the roots of the characteristic polynomial by solving the quadratic

$$1 - \phi_1 u - \phi_2 u^2 = 0.$$

If we substitute in for ϕ_1 and ϕ_2 and use the quadratic formula we get

$$u_{1,2} = 1 \pm \mathrm{i}/\sqrt{3},$$

and these obviously have modulus greater than one, so the model is stationary.

To check with a computer, it's arguably more convenient to see if the eigenvalues of the generator of the equivalent VAR(1) model have modulus less than one.

```
Phi = matrix(c(1.5,-0.75,1,0), ncol=2, byrow=TRUE)
evals = eigen(Phi)$values
```

```
abs(evals)
```

```
[1] 0.8660254 0.8660254
```

They both have modulus less than one, so again, we are in the stationary region. We can also check that the roots that we calculated by hand are correct.

```
1/evals
```

```
[1] 1-0.5773503i 1+0.5773503i
```

In fact, R has a built-in polynomial root finder, `polyroot`, so the simplest approach is to use this function to directly find the roots of the characteristic polynomial.

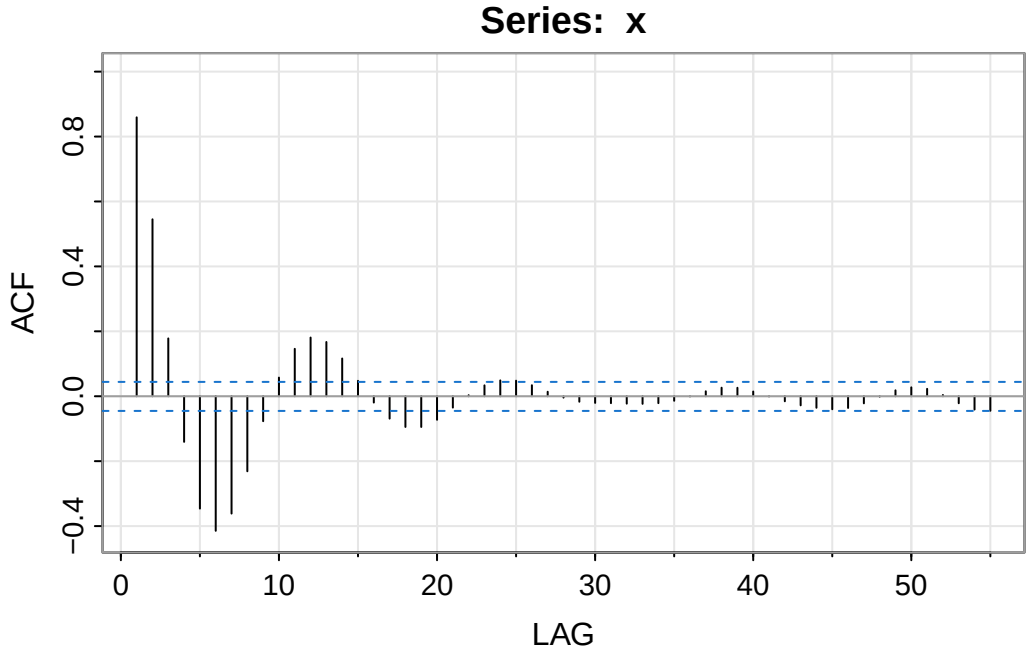
```
polyroot(c(1, -1.5, 0.75))
```

```
[1] 1+0.5773503i 1-0.5773503i
```

However, having the VAR(1) generator matrix at our disposal will turn out to be useful, anyway, as we will see shortly.

Now that we have established that the process is stationary, attention can focus on the auto-covariance structure. We can start by looking at the empirical ACF of the simulated data.

```
acf1(x)
```




```
 [1]  0.86  0.55  0.18 -0.14 -0.35 -0.41 -0.36 -0.23 -0.08  0.06  0.15  0.18
[13]  0.17  0.12  0.05 -0.02 -0.07 -0.09 -0.09 -0.07 -0.04  0.00  0.03  0.05
[25]  0.05  0.03  0.01  0.00 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.01  0.00
[37]  0.02  0.03  0.03  0.01  0.00 -0.02 -0.03 -0.04 -0.04 -0.04 -0.02  0.00
[49]  0.02  0.03  0.02  0.00 -0.02 -0.04 -0.04
```

Given the complex roots of the characteristic polynomial, this oscillatory behaviour in the ACF is to be expected. But what is the true ACF of the model? We know that however we want to compute the auto-covariances, we will need to know the first two, $\gamma_0 = v$ and γ_1, in order to initialise the recursion or find a particular solution to the explicit form.

If we want to compute these by hand, we will use some form of the Yule-Walker equations. It is usually easiest to use the auto-correlation form of the Yule-Walker equations. We will look at that approach shortly, but to begin with, it is instructive to examine the approach based on the auto-covariance form. Start with

$$\begin{aligned}\gamma_0 &= \phi_1\gamma_1 + \phi_2\gamma_2 + \sigma^2\\ \gamma_1 &= \phi_1\gamma_0 + \phi_2\gamma_1\\ \gamma_2 &= \phi_1\gamma_1 + \phi_2\gamma_0,\end{aligned}$$

to get our three equations in three unknowns. To solve these by Gaussian elimination, we can write in the form of an augmented matrix,

$$\left[\begin{array}{ccc|c} 1 & -\phi_1 & -\phi_2 & \sigma^2 \\ -\phi_1 & 1-\phi_2 & 0 & 0 \\ -\phi_2 & -\phi_1 & 1 & 0 \end{array}\right].$$

Substituting in our values gives

$$\left[\begin{array}{ccc|c} 1 & -3/2 & 3/4 & 1 \\ -3/2 & 7/4 & 0 & 0 \\ 3/4 & -3/2 & 1 & 0 \end{array}\right].$$

Row reduction will lead to the first three auto-covariances. I will cheat.

```
m = matrix(c(1,-3/2,3/4, -3/2,7/4,0, 3/4,-3/2,1), ncol=3, byrow=TRUE)
solve(m, c(1,0,0))
```

```
[1] 8.615385 7.384615 4.615385
```

So $v = \gamma_0 = 8.62$, $\gamma_1 = 7.38$, $\gamma_2 = 4.62$.

On a computer, the above approach is not the simplest. Here it is simpler to use our VAR(1) representation and solve for the stationary variance matrix.

```
Sig = matrix(c(1,0,0,0), ncol=2)
V = netcontrol::dlyap(t(Phi), Sig) # note transpose of Phi
V
```

```
         [,1]     [,2]
[1,] 8.615385 7.384615
[2,] 7.384615 8.615385
```

The first row (or column) of V gives the first two auto-covariances. Note that they are the same as we computed above.

Note: *According to the documentation for* `dlyap`*, it should not be necessary to transpose* `Phi`*, but it gives the wrong answer if I don't. I think this is a bug in either the function or its documentation.*

We can check that we do have a good solution by checking that

```
V - (Phi %*% V %*% t(Phi))
```

```
              [,1]          [,2]
[1,] 1.000000e+00 1.776357e-15
[2,] 2.664535e-15 3.552714e-15
```

is close to `Sig`.

Now that we have the initial auto-covariances we can generate more. If we are on a computer, it may be simplest to just recursively generate them.

```
initAC = V[1,]
acvf = filter(rep(0,49), c(1.5, -0.75), "rec", init=initAC)
acvf = c(initAC[1], acvf)
acvf[1:20]
```

```
 [1]  8.6153846  7.3846154  4.6153846  1.3846154 -1.3846154 -3.1153846
 [7] -3.6346154 -3.1153846 -1.9471154 -0.5841346  0.5841346  1.3143029
[13]  1.5333534  1.3143029  0.8214393  0.2464318 -0.2464318 -0.5544715
[19] -0.6468835 -0.5544715
```

We can turn these into auto-correlations and overlay them on the empirical ACF of the simulated data set.

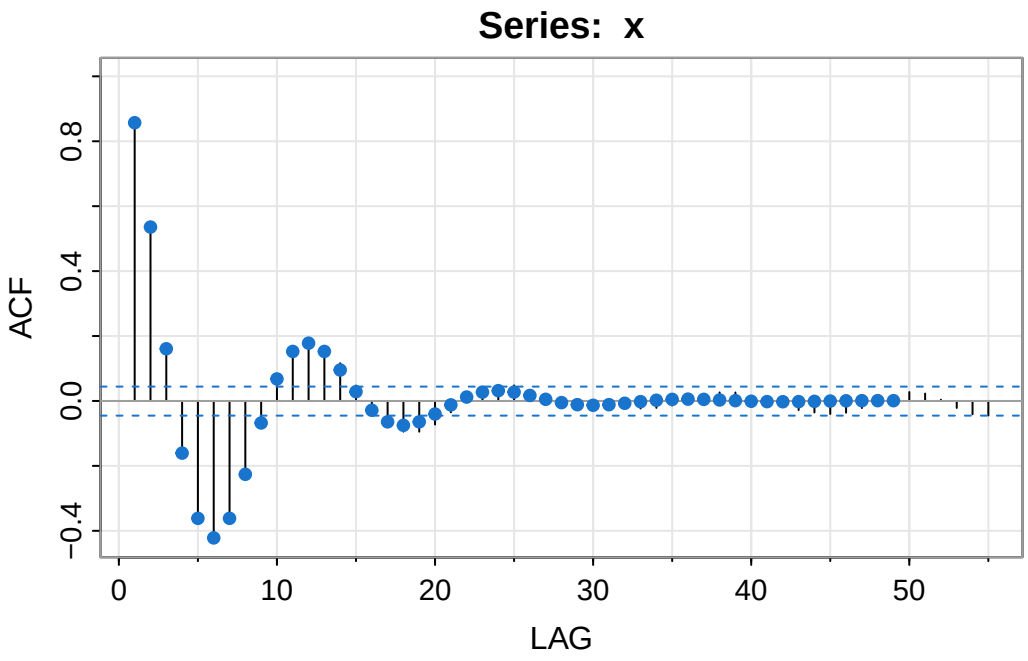
```
acrf = acvf[-1]/acvf[1]
print(acrf[1:20])
```

```
 [1]  0.85714286  0.53571429  0.16071429 -0.16071429 -0.36160714 -0.42187500
 [7] -0.36160714 -0.22600446 -0.06780134  0.06780134  0.15255301  0.17797852
[13]  0.15255301  0.09534563  0.02860369 -0.02860369 -0.06435830 -0.07508469
[19] -0.06435830 -0.04022394
```

```
acf1(x)
```

```
 [1]  0.86  0.55  0.18 -0.14 -0.35 -0.41 -0.36 -0.23 -0.08  0.06  0.15  0.18
[13]  0.17  0.12  0.05 -0.02 -0.07 -0.09 -0.09 -0.07 -0.04  0.00  0.03  0.05
[25]  0.05  0.03  0.01  0.00 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.01  0.00
[37]  0.02  0.03  0.03  0.01  0.00 -0.02 -0.03 -0.04 -0.04 -0.04 -0.02  0.00
[49]  0.02  0.03  0.02  0.00 -0.02 -0.04 -0.04
```

```
points(1:length(acrf), acrf, col=4, pch=19, cex=0.8)
```



We see that the empirical ACF closely matches the true ACF of the AR(2) model, at least for small lags. In fact, if we just want the auto-correlation function, there is a built-in function to compute it.

```
ARMAacf(ar=c(1.5, -0.75), lag.max=10)
```

```
          0           1           2           3           4           5
 1.00000000  0.85714286  0.53571429  0.16071429 -0.16071429 -0.36160714
          6           7           8           9          10
-0.42187500 -0.36160714 -0.22600446 -0.06780134  0.06780134
```

If working by hand, it may be preferable to find an explicit solution for γ_k in terms of k. We know that the solution is of the form

$$\gamma_k = A_1 u_1^{-k} + A_2 u_2^{-k},$$

where u_i are the roots of the characteristic equation. We can use γ_0 and γ_1 to fix A_1 and A_2, since

$$\begin{aligned}\gamma_0 &= A_1 + A_2 \\ \gamma_1 &= A_1 u_1^{-1} + A_2 u_2^{-1},\end{aligned}$$

and we could write this as the augmented matrix

$$\left[\begin{array}{cc|c} 1 & 1 & \gamma_0 \\ u_1^{-1} & u_2^{-1} & \gamma_1 \end{array}\right].$$

Substituting in our values give

$$\left[\begin{array}{cc|c} 1 & 1 & 8.615 \\ 1/(1-\mathrm{i}/\sqrt{3}) & 1/(1+\mathrm{i}/\sqrt{3}) & 7.385 \end{array}\right],$$

which we can row-reduce to compute A_1 and A_2. Again, I will cheat.

```
m = matrix(c(1,1, 1/complex(r=1,i=-1/sqrt(3)), 1/complex(r=1,i=1/sqrt(3))),
      ncol=2, byrow=TRUE)
A = solve(m, c(8.615, 7.385))
```

```
A
```

```
[1] 4.3075-1.066655i 4.3075+1.066655i
```

So $A_1 = 4.31 - 1.07i$, $A_2 = 4.31 + 1.07i$ are complex conjugates, just as the roots of the characteristic polynomial are. This is to be expected, since

$$\gamma_k = Au^{-k} + \bar{A}\bar{u}^{-k}$$

clearly has the property that $\gamma_k = \bar{\gamma}_k$, and so γ_k is real, as we would hope. To get rid of the imaginary parts, it is convenient to switch to modulus/argument form, putting $u = re^{\mathrm{i}\theta}$ $(r > 1)$, $A = se^{\mathrm{i}\psi}$, since then

$$\begin{aligned}\gamma_k &= sr^{-k}\left[e^{\mathrm{i}(\psi - k\theta)} + e^{-\mathrm{i}(\psi - k\theta)}\right] \\ &= 2sr^{-k}\cos(\psi - k\theta).\end{aligned}$$

Writing it this way makes it clear that the frequency of the oscillations in the ACF are determined by θ, the argument of u.

```
u = polyroot(c(1, -1.5, 0.75))[1]
f = Arg(u)/(2*pi)
print(f)
```

```
[1] 0.08333333
```

```
1/f
```

```
[1] 12
```

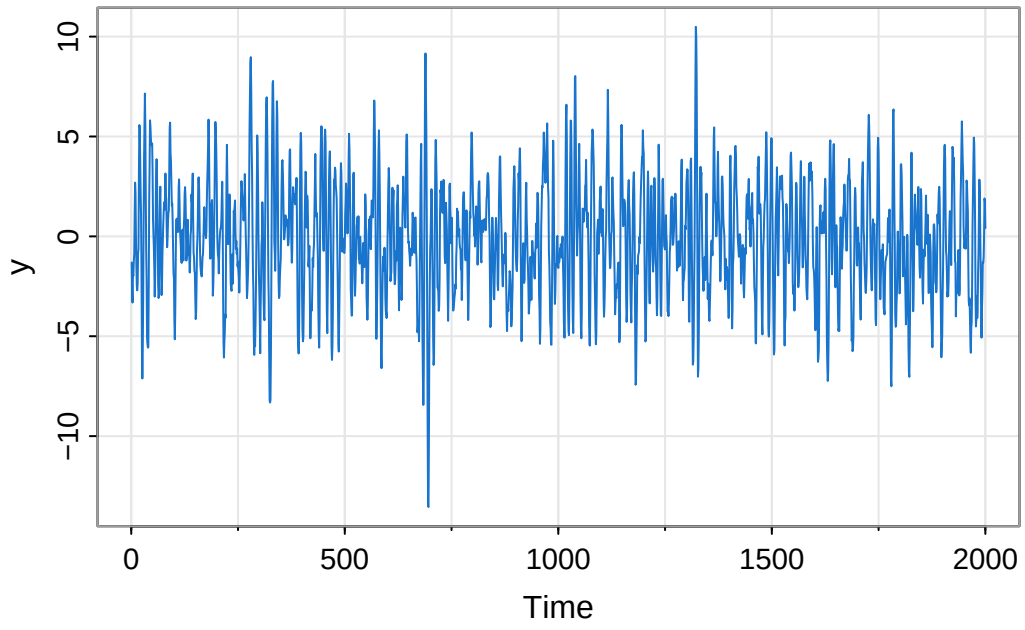
For our example, the frequency is 1/12, and hence the period of the oscillations is 12. We now have everything we need to express our auto-covariance function explicitly. Here is a quick check using R.

```
sapply(0:10, function(k)
  2*abs(A[2])*abs(u)^(-k)*cos(Arg(A[2])-k*Arg(u)))
```

```
 [1]  8.6150000  7.3850000  4.6162500  1.3856250 -1.3837500 -3.1148438
 [7] -3.6344531 -3.1155469 -1.9474805 -0.5845605  0.5837695
```

Finally, note that we have been simulating data from an AR model by applying a recursive filter to white noise. This is pedagogically instructive, but there is a function in base R, `arima.sim` that can simulate data from any ARMA model directly (and from a generalisation of ARMA, known as ARIMA).

```
y = arima.sim(n=(n+burn), list(ar=phi), sd=sigma)[-(1:burn)]
tsplot(y, col=4)
```



3.2.3.1 The auto-correlation approach

We have now completely solved for the auto-covariance function of our example process. However, when working by hand it turns out to be significantly simpler to solve directly for the auto-correlation function of the process using the auto-correlation form of the Yule-Walker equations. This isn't limiting, since if we want the auto-covariance function, we can just multiply through by the stationary variance. Let's work this through for our current example. Begin with the auto-correlation form of the first Yule-Walker equation:

$$\rho_1 = \phi_1\rho_0 + \phi_2\rho_1 = \phi_1 + \phi_2\rho_1,$$

since $\rho_0 = 1$, and so

$$\rho_1 = \frac{\phi_1}{1-\phi_2},$$

and this is true for any AR(2) model. Substituting in for our ϕ_1 and ϕ_2 gives $\rho_1 = 6/7$. But now we know ρ_0 and ρ_1 we have enough information to initialise the Yule-Walker equations, without having to solve a bunch of linear equations (we have, but the system is smaller and simpler, which is generally the case). If we want to know the stationary variance of the process, we will also need to know ρ_2, which we can calculate directly from the 2nd Yule-Walker equation

$$\rho_2 = \phi_1\rho_1 + \phi_2\rho_0 = \frac{3}{2}\times\frac{6}{7} + \frac{-3}{4}\times 1 = \frac{15}{28}.$$

We can then compute the stationary variance,

$$\gamma_0 = \frac{\sigma_2}{1-\phi_1\rho_1-\phi_2\rho_2} = 8\frac{8}{13}.$$

We can now look for a general solution to either the auto-correlation or auto-covariance function. We will look at the auto-correlation function. There are many ways to proceed here, but from our above analysis, we know that in the case of an AR(2) with complex roots our solution will end up being of the form

$$\rho_k = Ar^{-k}\cos k\theta + Br^{-k}\sin k\theta,$$

where $u = re^{\mathrm{i}\theta}$ and A, B are real constants that can be determined by ρ_0 and ρ_1. Clearly for any AR(2), $\rho_0 = 1$ implies $A = 1$. Then the equation for ρ_1,

$$\rho_1 = r^{-1}\cos\theta + Br^{-1}\sin\theta,$$

can be re-arranged for B as

$$B = \frac{\rho_1 r - \cos\theta}{\sin\theta}.$$

For our example, we have $r = 2/\sqrt{3}$, $\theta = \pi/6$ and $\rho_1 = 6/7$, so substituting in gives

$$B = \frac{\frac{6}{7}\frac{2}{\sqrt{3}} - \frac{\sqrt{3}}{2}}{\frac{1}{2}} = \frac{\sqrt{3}}{7},$$

leading to full auto-correlation function

$$\rho_k = \left(\frac{\sqrt{3}}{2}\right)^k \cos(k\pi/6) + \frac{\sqrt{3}}{7}\left(\frac{\sqrt{3}}{2}\right)^k \sin(k\pi/6)$$

A quick check with R confirms that we haven't messed up.

```
sapply(0:10, function(k)
  (sqrt(3)/2)^k*cos(k*pi/6) + (sqrt(3)/7)*(sqrt(3)/2)^k*sin(k*pi/6))
```

```
 [1]  1.00000000  0.85714286  0.53571429  0.16071429 -0.16071429 -0.36160714
 [7] -0.42187500 -0.36160714 -0.22600446 -0.06780134  0.06780134
```

3.2.4 Partial auto-covariance and correlation

The ACF is a useful characterisation of the way in which correlations decay away with increasing lag. But the ACF of an AR model does not sharply truncate in general, since observations from the distant past do have some (small) influence on the present. However, that influence is "carried" via the intermediate observations. If we *knew* the intermediate observations, there would be no way for the observations from the distant past to give us any more information about the present. So, for an AR model, it could be useful to understand the correlation between observations that is "left over" after adjusting for the effect of any intermediate observations. This is the idea behind the partial auto-correlation function (PACF).

In the context of stationary Gaussian time series, the *partial auto-covariance* at lag k is just the *conditional* covariance

$$\gamma_k^\star = \mathbb{C}\text{ov}[X_{t-k}, X_t | \mathbf{X}_{(t-k+1):(t-1)}].$$

This is well-defined, since we know from multivariate normal theory that the conditional covariance does not actually depend on the observed value of the conditioning variable. Nevertheless, it can sometimes be convenient to explicitly include it in the notation, in which case we could write

$$\gamma_k^\star = \mathbb{C}\text{ov}[X_{t-k}, X_t | \mathbf{X}_{(t-k+1):(t-1)} = \mathbf{x}_{(t-k+1):(t-1)}].$$

We can then define the *partial auto-correlation* at lag k by

$$\rho_k^\star = \frac{\mathbb{C}\text{ov}[X_{t-k}, X_t | \mathbf{X}_{(t-k+1):(t-1)}]}{\mathbb{V}\text{ar}[X_t | \mathbf{X}_{(t-k+1):(t-1)}]} = \frac{\gamma_k^\star}{\mathbb{V}\text{ar}[X_t | \mathbf{X}_{(t-k+1):(t-1)}]}.$$

Intuitively, it should be clear that for an AR(p), we will have $\gamma_k^\star = \rho_k^\star = 0,\ \forall k > p$, since then there will be no remaining influence of the observation from lag k after conditioning on the intervening observations. We

can confirm this, since for $k > p$,

$$\begin{aligned}
\gamma_k^\star &= \mathbb{C}\text{ov}[X_{t-k}, X_t | \mathbf{X}_{(t-k+1):(t-1)} = \mathbf{x}_{(t-k+1):(t-1)}] \\
&= \mathbb{C}\text{ov}\left[X_{t-k}, \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t \middle| \mathbf{X}_{(t-k+1):(t-1)} = \mathbf{x}_{(t-k+1):(t-1)} \right] \\
&= \mathbb{C}\text{ov}\left[X_{t-k}, \sum_{i=1}^{p} \phi_i x_{t-i} + \varepsilon_t \middle| \mathbf{X}_{(t-k+1):(t-1)} = \mathbf{x}_{(t-k+1):(t-1)} \right] \\
&= \mathbb{C}\text{ov}\left[X_{t-k}, \varepsilon_t | \mathbf{X}_{(t-k+1):(t-1)} = \mathbf{x}_{(t-k+1):(t-1)} \right] \\
&= 0.
\end{aligned}$$

Another interesting case is the case $k = p$, since then

$$\begin{aligned}
\gamma_p^\star &= \mathbb{C}\text{ov}[X_{t-p}, X_t | \mathbf{X}_{(t-p+1):(t-1)} = \mathbf{x}_{(t-p+1):(t-1)}] \\
&= \mathbb{C}\text{ov}\left[X_{t-p}, \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t \middle| \mathbf{X}_{(t-p+1):(t-1)} = \mathbf{x}_{(t-p+1):(t-1)} \right] \\
&= \mathbb{C}\text{ov}\left[X_{t-p}, \sum_{i=1}^{p-1} \phi_i x_{t-i} + \phi_p X_{t-p} + \varepsilon_t \middle| \mathbf{X}_{(t-p+1):(t-1)} = \mathbf{x}_{(t-p+1):(t-1)} \right] \\
&= \mathbb{C}\text{ov}\left[X_{t-p}, \phi_p X_{t-p} | \mathbf{X}_{(t-p+1):(t-1)} = \mathbf{x}_{(t-p+1):(t-1)} \right] \\
&= \phi_p \mathbb{V}\text{ar}\left[X_{t-p} | \mathbf{X}_{(t-p+1):(t-1)} = \mathbf{x}_{(t-p+1):(t-1)} \right],
\end{aligned}$$

and hence

$$\rho_p^\star = \phi_p.$$

So, importantly, $\rho_p^\star \neq 0$ for $\phi_p \neq 0$. It is also clear that $\gamma_1^\star = \gamma_1$ and $\rho_1^\star = \rho_1$, since then there are no intervening observations.

Other partial auto-correlations $(\rho_2^\star, \ldots, \rho_{p-1}^\star)$ are less obvious, but can be computed using standard multivariate normal theory:

$$\mathbb{C}\text{ov}[X, Y | \mathbf{Z}] = \mathbb{C}\text{ov}[X, Y] - \mathbb{C}\text{ov}[X, \mathbf{Z}] \mathbb{V}\text{ar}[\mathbf{Z}]^{-1} \mathbb{C}\text{ov}[\mathbf{Z}, Y].$$

However, in practice they are typically computed by a more efficient method, known as the *Durbin-Levinson Algorithm* (an application of Levinson recursion), but the details of this are not important for this course.

3.2.4.1 Example: AR(2)

For the AR(2) model that we have been studying, we can use `ARMAacf` to compute the PACF as well as the ACF.

```
ARMAacf(ar=c(1.5, -0.75), lag.max=4)
```

```
        0         1         2         3         4
1.0000000 0.8571429 0.5357143 0.1607143 -0.1607143
```

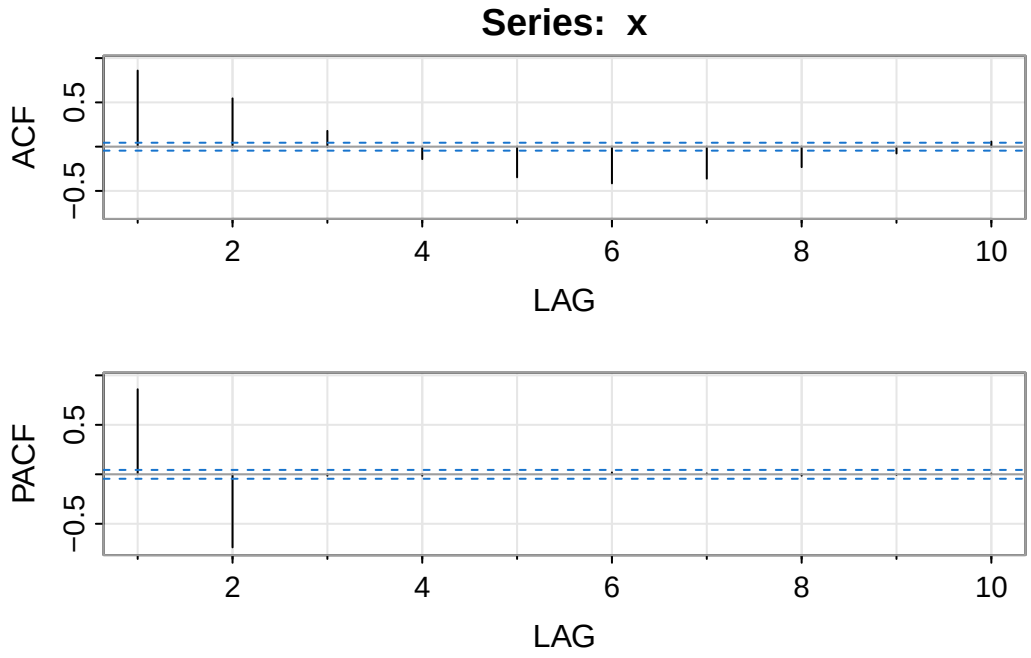
```
ARMAacf(ar=c(1.5, -0.75), lag.max=4, pacf=TRUE)
```

```
[1]  8.571429e-01 -7.500000e-01  1.913000e-15 -7.357691e-16
```

The important things to note are that $\rho_1^\star = \rho_1$, $\rho_2^\star = \phi_2$ and $\rho_k^\star = 0,\ \forall k > 2$. Crucially, the PACF truncates at the order of the AR model.

Numerous methods exist to estimate partial auto-correlations empirically, from data. The details need not concern us. We can look at both the ACF and PACF for a given dataset using `astsa::acf2`. eg. we can apply it to our simulated dataset.

```
acf2(x, max.lag=10)
```




```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
ACF   0.86  0.55  0.18 -0.14 -0.35 -0.41 -0.36 -0.23 -0.08  0.06
PACF  0.86 -0.74 -0.01 -0.01  0.00  0.02  0.01 -0.01 -0.01  0.01
```

3.3 MA(q)

An MA(q) (or ARMA$(0, q)$) model is determined by

$$X_t = \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}, \tag{3.5}$$

for white noise $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$. In other words, X_t is the result of applying a FIR convolutional linear filter to white noise. It is immediately clear that (in contrast to the AR model) since white noise is stationary, and the convolutional linear filter being applied is time invariant, that the process will always be *stationary*, irrespective of the parameters $\boldsymbol{\theta}, \sigma$.

Note that many formula relating to MA processes can be simplified by defining $\theta_0 \equiv 1$, so that

$$X_t = \sum_{j=0}^{q} \theta_j \varepsilon_{t-j}.$$

It is immediately clear via linearity of expectation that the stationary mean is $\mathbb{E}[X_t] = 0$. The stationary variance is also very straightforwardly computed as

$$\begin{aligned} v = \gamma_0 = \mathbb{V}\mathrm{ar}[X_t] &= \sum_{j=0}^{q} \mathbb{V}\mathrm{ar}[\theta_j \varepsilon_{t-j}] \\ &= \sum_{j=0}^{q} \theta_j^2 \sigma^2 \\ &= \sigma^2 \sum_{j=0}^{q} \theta_j^2. \end{aligned}$$

The auto-covariance function is not much more difficult. For $k > 0$ we have

$$\begin{aligned} \gamma_k &= \mathbb{C}\mathrm{ov}[X_{t-k}, X_t] \\ &= \mathbb{C}\mathrm{ov}\left[\sum_{i=0}^{q} \theta_i \varepsilon_{t-k-i}, \sum_{j=0}^{q} \theta_j \varepsilon_{t-j}\right] \\ &= \sum_{i=0}^{q}\sum_{j=0}^{q} \theta_i \theta_j \mathbb{C}\mathrm{ov}[\varepsilon_{t-k-i}, \varepsilon_{t-j}] \\ &= \sigma^2 \sum_{i=0}^{q}\sum_{j=0}^{q} \theta_i \theta_j \delta_{k+i,j} \\ &= \sigma^2 \sum_{j=k}^{q} \theta_{j-k}\theta_j, \quad k \leq q \\ &= \sigma^2 \sum_{j=0}^{q-k} \theta_j \theta_{j+k}. \end{aligned}$$

In other words,

$$\gamma_k = \begin{cases} \sigma^2 \displaystyle\sum_{j=0}^{q-k} \theta_j \theta_{j+k}, & 0 \leq k \leq q, \\ 0, & k > q. \end{cases}$$

Note, in particular, that $\gamma_q = \sigma^2 \theta_q \neq 0$ for $\sigma, \theta_q \neq 0$. Similarly,

$$\rho_k = \begin{cases} \displaystyle\sum_{j=0}^{q-k} \theta_j \theta_{j+k} \Big/ \sum_{j=0}^{q} \theta_j^2, & 0 \leq k \leq q, \\ 0, & k > q. \end{cases}$$

So, the ACF truncates after lag q.

3.3.1 Backshift notation

It can sometimes be useful to write the MA(q) process using backshift notation

$$X_t = \theta(B)\varepsilon_t,$$

where

$$\theta(B) = 1 + \sum_{j=1}^{q} \theta_j B^j$$

is known as the *moving average operator*.

3.3.2 Special case: the MA(1)

It will be instructive to consider in detail the special case of the MA(1) process

$$X_t = \varepsilon_t + \theta\varepsilon_{t-1} = (1 + \theta B)\varepsilon_t.$$

The ACF

$$\rho_k = \begin{cases} \frac{\theta}{1+\theta^2}, & k = 1 \\ 0, & k > 1 \end{cases}$$

has the interesting property that replacing θ by θ^{-1} leads to exactly the same ACF (try it!). So there are two different values of θ that lead to exactly the same ACF. By adjusting the noise variance, σ^2 appropriately, we can also ensure that the full auto-covariance function matches up as well. But a mean zero stationary Gaussian process is entirely determined by its auto-covariance function. So there are two different MA(1) models that define exactly the same Gaussian process. This is potentially problematic, especially when it comes to estimating an MA model from data. For reasons to become clear, we should prefer the model with $|\theta| < 1$, and only one of the two models will have that property, so we can use this to rescue us from the potential non-uniqueness problem. We will see how this generalises to general MA(q) process once we better understand why the $|\theta| < 1$ solution is preferred.

We know that the ACF truncates after lag one. Can we say anything about the PACF? Just as we could use successive substitution to turn an AR(1) into an MA(∞), we can do the same to turn an MA(1) into an AR(∞).

$$\begin{aligned} \varepsilon_t &= X_t - \theta\varepsilon_{t-1} \\ &= X_t - \theta(X_{t-1} - \theta\varepsilon_{t-2}) \\ &= \cdots \\ &= X_t + \sum_{j=1}^{\infty} (-\theta)^j X_{t-j} \end{aligned}$$

This will converge to an AR(∞) representation of an MA(1) process provided that $|\theta| < 1$. We say that the process is *invertible*, in the sense that the MA process can be inverted to an AR process. It may be helpful to write this using backshift notation as

$$\left(1 + \sum_{j=1}^{\infty} (-\theta)^j B^j\right) X_t = \varepsilon_t.$$

This makes perfect sense, since

$$X_t = (1 + \theta B)\varepsilon_t \quad \Rightarrow \quad (1 + \theta B)^{-1} X_t = \varepsilon_t,$$

and

$$(1 + \theta B)^{-1} = 1 + \sum_{j=1}^{\infty} (-\theta)^j B^j.$$

So, since (in the invertible case) the MA(1) has a representation as a AR(∞), and we know that the PACF of a AR(p) truncates after lag p, it is clear that the PACF of the MA(1) will not truncate. You should also be starting to see some duality between AR and MA processes.

We can use R to compute the ACF and PACF of an MA process. eg. For an MA(1) with $\theta = 0.8$:

```
ARMAacf(ma=c(0.8), lag.max=6)
```

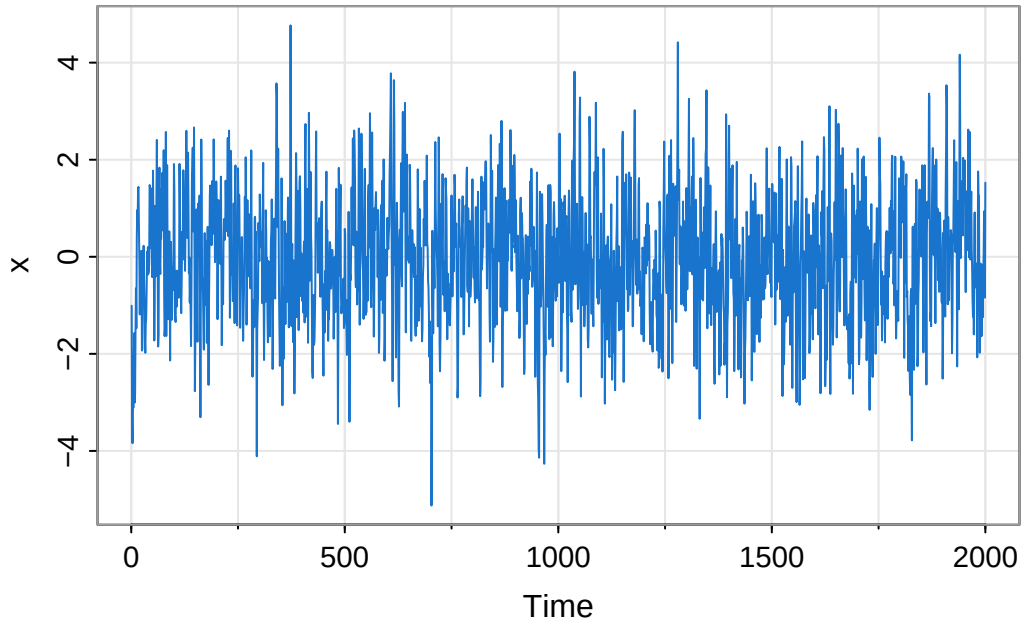
```
        0         1         2         3         4         5         6
1.0000000 0.4878049 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
```

```
ARMAacf(ma=c(0.8), lag.max=6, pacf=TRUE)
```

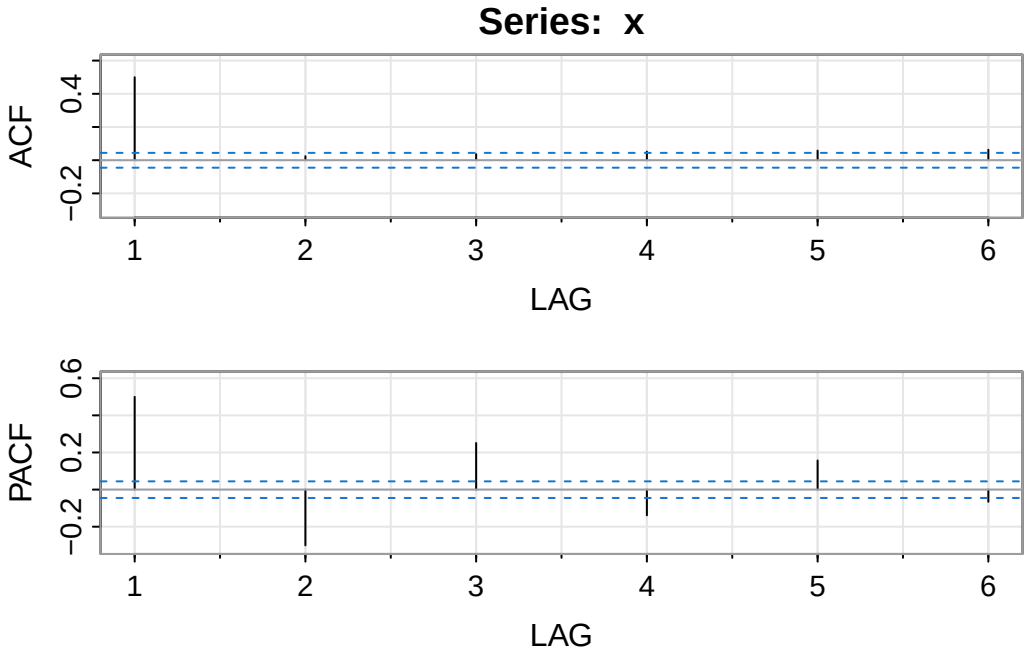
```
[1]  0.4878049 -0.3122560  0.2214778 -0.1651935  0.1266695 -0.0987133
```

We can also simulate data, either by applying a convolutional filter to noise, or by using `arima.sim`.

```
x = filter(rnorm(2000), c(1, 0.8))
x = arima.sim(n=2000, list(ma=c(0.8)), sd=1)
## either of the above approaches is fine
tsplot(x, col=4)
```




```
acf2(x, max.lag=6)
```



```
      [,1]   [,2] [,3]   [,4] [,5]   [,6]
ACF    0.5   0.02 0.04   0.05 0.06   0.06
PACF   0.5 -0.30 0.25 -0.14 0.16 -0.07
```

3.3.3 Invertibility

For an MA(1), we saw that it would be invertible if $|\theta| < 1$. That's the same as saying that the root of the moving average characteristic polynomial

$$\theta(u) = 1 + \theta u$$

satisfies $|u| > 1$. This isn't a coincidence.

For an MA(q), the characteristic polynomial $\theta(u)$ is of degree q. So, by the fundamental theorem of algebra, we can factor this polynomial in the form

$$\theta(u) = \theta_q \prod_{j=1}^{q} (u - u_j),$$

where $u_1, \ldots, u_q$ are the roots of $\theta(u)$. Consequently, we can write

$$\theta(u)^{-1} = \theta_q^{-1} \prod_{j=1}^{q} (u - u_j)^{-1},$$

so it is reasonably clear that the MA(q) will be invertible if every term of the form $(u-u_j)^{-1}$ has a convergent series expansion. But

$$\begin{aligned}(u - u_j)^{-1} &= -u_j^{-1} \left(1 - \frac{u}{u_j}\right)^{-1} \\ &= -u_j^{-1} \left(1 + \frac{u}{u_j} + \frac{u^2}{u_j^2} + \cdots\right)\end{aligned}$$

will converge for $|u_j| > 1$. Consequently, $\theta(u)^{-1}$ will have a convergent expansion when *all* roots of $\theta(u)$ are *outside* of the unit circle. This is the condition we will require for the very desirable property of invertibility.

3.4 ARMA(p,q)

We have seen that a AR(p) process has the property that its PACF will truncate after lag p, but that its ACF does not sharply truncate, but rather decays away geometrically. We have also seen that the MA(q) process has the property that its ACF will truncate after lag q, but that its PACF does not (again, it decays away geometrically). We require the roots of the auto-regressive operator to lie outside of the unit circle in order to ensure stationarity. In fact, this also ensures that the AR process is invertible to an MA process. We also know that although an MA process is always stationary, we nevertheless restrict attention to the case where the roots of the moving average operator lie outside of the unit circle in order to ensure invertibility (and uniqueness). We now turn our attention to the general ARMA process, with both AR and MA components.

We consider the model

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j},$$

for $p, q > 0$. We could write this with backshift notation as

$$\phi(B) X_t = \theta(B) \varepsilon_t,$$

for the appropriately defined auto-regressive and moving-average operators $\phi(B)$ and $\theta(B)$, respectively. We assume that the roots of both $\phi(u)$ and $\theta(u)$ lie outside of the unit circle, so that the process is stationary and invertible. This means that we can express the ARMA(p, q) model either as the MA(∞) process

$$X_t = \frac{\theta(B)}{\phi(B)}\varepsilon_t,$$

or as the AR(∞) process

$$\frac{\phi(B)}{\theta(B)}X_t = \varepsilon_t.$$

It is then clear that for $p, q > 0$, neither the ACF nor the PACF will sharply truncate.

3.4.1 Parameter redundancy

There is an additional issue that arises for mixed ARMA processes that did not arise in the pure AR or MA case. Start with a pure white noise process

$$X_t = \varepsilon_t$$

and apply $(1 - \lambda B)$ to both sides (for some fixed $|\lambda| < 1$) to get

$$X_t = \lambda X_{t-1} + \varepsilon_t - \lambda\varepsilon_{t-1}$$

So this now *looks like* an ARMA(1,1) process. But it is actually the same white noise process that we started off with, as we would find if we tried to compute its ACF, etc. But this is a problem, because there are clearly infinitely many apparently different ways to represent exactly the same stationary Gaussian process. So we are back to having a non-uniqueness/parameter redundancy problem. Again, this will be problematic, especially when we come to think about inferring ARMA models from data. This issue arises whenever $\phi(u)$ and $\theta(u)$ contain an identical root. So, in addition to assuming that the roots of $\phi(u)$ and $\theta(u)$ all lie outside of the unit circle, we further restrict attention to the case where $\phi(u)$ and $\theta(u)$ have no roots in common. If we are presented with an ARMA(p, q) model where $\phi(u)$ and $\theta(u)$ *do* have a root in common, we can just divide out the corresponding linear factor from both in order to obtain an ARMA$(p-1, q-1)$ model that is equivalent (and more parsimonious).

3.4.2 Example: ARMA(1,1)

Consider the ARMA(1,1) process

$$X_t = \phi X_{t-1} + \varepsilon_t + \theta\varepsilon_{t-1},$$

for $|\phi|, |\theta| < 1$, $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$. For $k \geq 0$ we can multiply through by X_{t-k} and take expectations to get

$$\gamma_k = \phi\gamma_{k-1} + \mathbb{C}\text{ov}[X_{t-k}, \varepsilon_t] + \theta\mathbb{C}\text{ov}[X_{t-k}, \varepsilon_{t-1}].$$

To proceed further it will be useful to know that

$$\mathbb{C}\text{ov}[X_t, \varepsilon_t] = \mathbb{C}\text{ov}[\phi X_{t-1} + \varepsilon_t + \theta\varepsilon_{t-1}, \varepsilon_t] = \sigma^2$$

and

$$\mathbb{C}\text{ov}[X_t, \varepsilon_{t-1}] = \mathbb{C}\text{ov}[\phi X_{t-1} + \varepsilon_t + \theta\varepsilon_{t-1}, \varepsilon_{t-1}] = \phi\mathbb{C}\text{ov}[X_{t-1}, \varepsilon_{t-1}] + \theta\sigma^2 = (\phi + \theta)\sigma^2.$$

So now, for $k = 0$ we have

$$\gamma_0 = \phi\gamma_1 + (1 + \theta\phi + \theta^2)\sigma^2$$

and for $k = 1$ we have

$$\gamma_1 = \phi\gamma_0 + \theta\sigma^2.$$

Solving these gives

$$v = \gamma_0 = \frac{1 + 2\theta\phi + \theta^2}{1 - \phi^2}\sigma^2$$

and

$$\gamma_1 = \frac{(1 + \theta\phi)(\phi + \theta)}{1 - \phi^2}\sigma^2.$$

For $k > 1$ we have

$$\gamma_k = \phi\gamma_{k-1},$$

and hence

$$\gamma_k = \begin{cases} \dfrac{1 + 2\theta\phi + \theta^2}{1 - \phi^2}\sigma^2 & k = 0 \\ \dfrac{(1 + \theta\phi)(\phi + \theta)}{1 - \phi^2}\phi^{k-1}\sigma^2 & k > 0. \end{cases}$$

We can also write down the auto-correlation function

$$\rho_k = \frac{(1 + \theta\phi)(\phi + \theta)}{1 + 2\theta\phi + \theta^2}\phi^{k-1}, \quad k > 0.$$

Note that the ACF degenerates to that of a white noise process when $\theta = -\phi$. This corresponds to the common root redundancy problem discussed above. Note also that the same would happen if $\phi\theta = -1$, but that our invertibility criterion rules out that possibility.

For concreteness, let's now study the particular case $\phi = 0.8$, $\theta = 0.6$ using R. We can get R to explicitly compute the ACF and PACF of this model.

```
ARMAacf(ar=c(0.8), ma=c(0.6), lag.max=6)
```

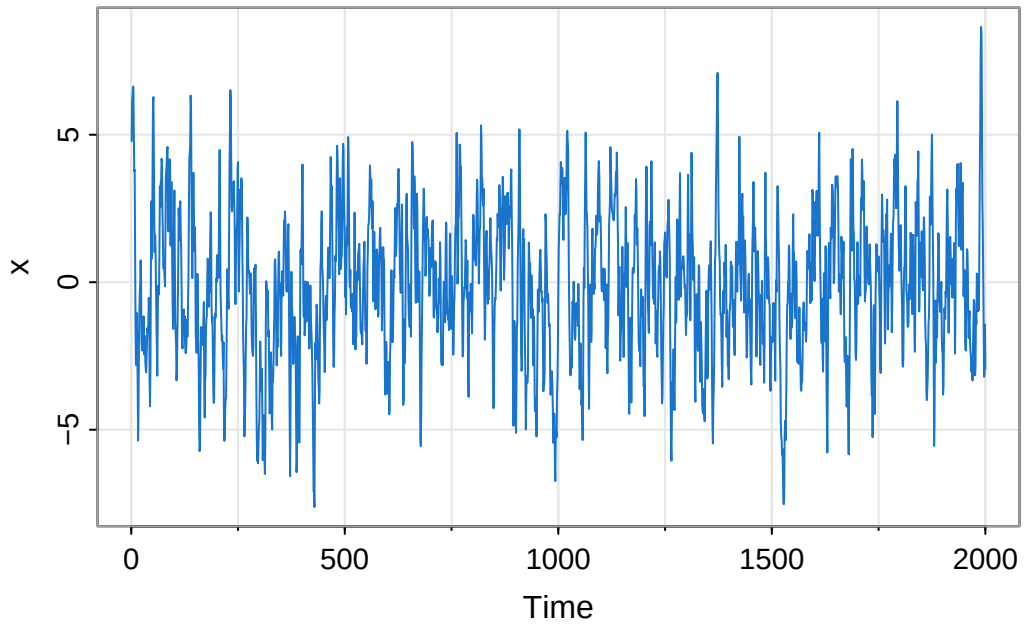
```
        0         1         2         3         4         5         6
1.0000000 0.8931034 0.7144828 0.5715862 0.4572690 0.3658152 0.2926521
```

```
ARMAacf(ar=c(0.8), ma=c(0.6), lag.max=6, pacf=TRUE)
```

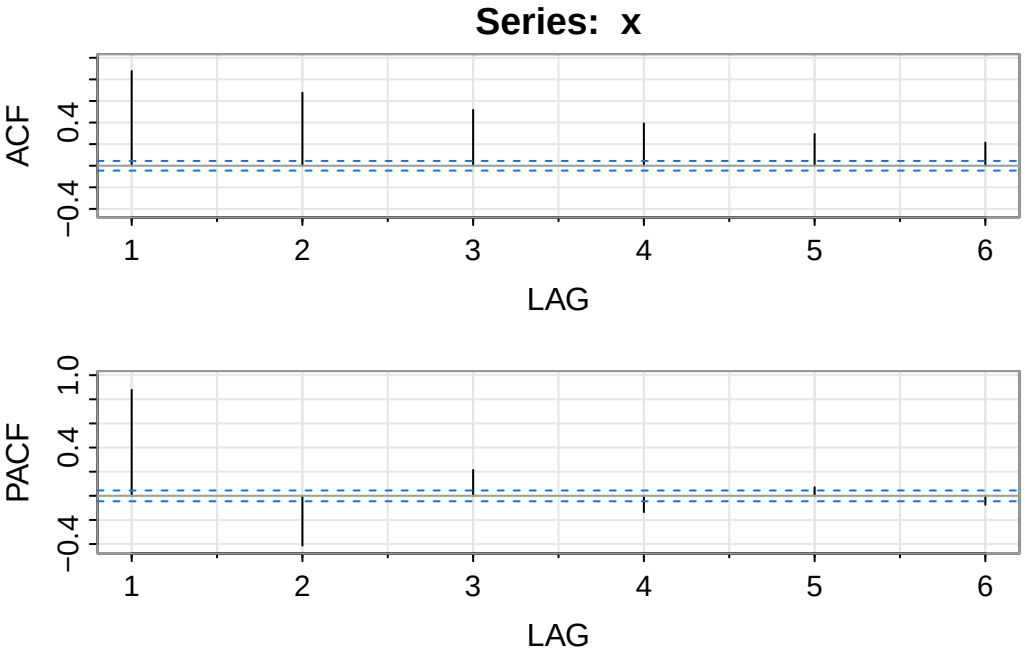
```
[1]  0.89310345 -0.41089371  0.22744126 -0.13276292  0.07888737 -0.04716820
```

We can also simulate data from this model and look at the empirical statistics.

```
x = signal::filter(c(1, 0.6), c(1, -0.8), rnorm(2000))
x = arima.sim(n=2000, list(ar=c(0.8), ma=c(0.6)), sd=1)
## either of the above approaches is fine
tsplot(x, col=4)
```




```
acf2(x, max.lag=6)
```




```
     [,1]  [,2] [,3]  [,4] [,5]  [,6]
ACF  0.88  0.68 0.52  0.39 0.29  0.21
PACF 0.88 -0.41 0.21 -0.14 0.07 -0.07
```

4 Estimation and forecasting

In this chapter we consider the problem of fitting time series models to observed time series data, and using the fitted models to forecast the unobserved future observations of the time series.

4.1 Fitting ARMA models to data

Since we mainly know about ARMA models, we concentrate here on the problem of fitting ARMA models to time series data. We assume that the time series has been detrended and has zero mean, so that fitting a mean zero stationary model to the data makes sense. We will also assume that the order of the process (the values of p and q) are known. This is unlikely to be true in practice, but the idea is that appropriate values can be identified by looking at ACF and PACF plots for the (detrended) time series.

4.1.1 Moment matching

Since we have seen how to compute the ACF of ARMA models, and we know how to compute the empirical ACF of an observed time series, it is natural to consider estimating the parameters of an ARMA model by finding parameters that match up well with the empirical ACF. Since the ACF encodes the second moments of the time series, this is *moment matching*, or the method of moments in the context of stationary time series analysis. This approach works best in the context of AR(p) models, so we focus mainly on that case.

4.1.1.1 AR(p)

Recall that the Yule-Walker equations from Section 3.2.1 could be written in either auto-covariance or auto-correlation form. We start with the auto-covariance form. If we define $\boldsymbol{\gamma} = (\gamma_1, \dots, \gamma_p)^\top$, $\boldsymbol{\phi} = (\phi_1, \dots, \phi_p)^\top$ and let Γ be the $p \times p$ symmetric matrix

$$\Gamma = \{\gamma_{i-j} | i, j = 1, \dots, p\},$$

then the Yule-Walker equations can be written in matrix form as

$$\Gamma\boldsymbol{\phi} = \boldsymbol{\gamma}, \quad \sigma^2 = \gamma_0 - \boldsymbol{\phi}^\top\boldsymbol{\gamma}.$$

Then given an empirical estimate of the auto-covariance function computed from time series data, we can construct estimates $\hat{\gamma}_0$, $\hat{\boldsymbol{\gamma}}$, and hence $\hat{\Gamma}$, and use them to estimate the parameters of the AR(p) model as

$$\hat{\boldsymbol{\phi}} = \hat{\Gamma}^{-1}\hat{\boldsymbol{\gamma}}, \quad \hat{\sigma}^2 = \hat{\gamma}_0 - \hat{\boldsymbol{\gamma}}^\top\hat{\Gamma}^{-1}\hat{\boldsymbol{\gamma}}.$$

These estimates are known as the *Yule-Walker estimators* of the AR(p) model parameters.

If we prefer, we can re-write all of the above in auto-correlation form. Define $\boldsymbol{\rho} = (\rho_1, \dots, \rho_p)^\top$ and let P be the $p \times p$ symmetric matrix

$$\mathsf{P} = \{\rho_{i-j} | i, j = 1, \dots, p\}.$$

Then the Yule-Walker equations can be written

$$\mathsf{P}\boldsymbol{\phi} = \boldsymbol{\rho}, \quad \sigma^2 = \gamma_0(1 - \boldsymbol{\phi}^\top\boldsymbol{\rho}).$$

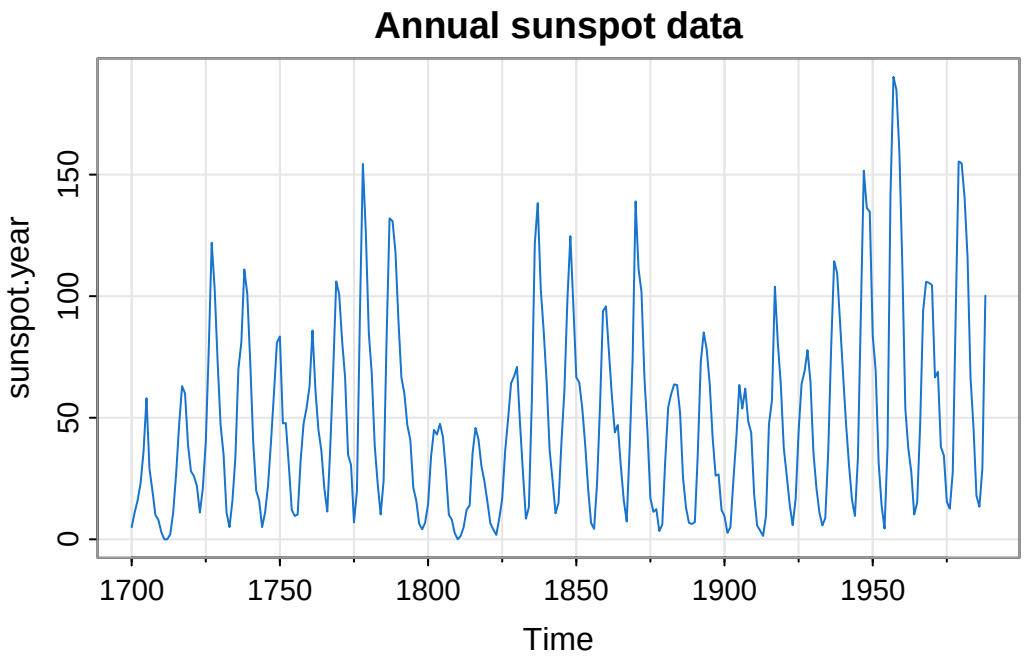
So, given empirical ACF estimates $\hat{\boldsymbol{\rho}}$, $\hat{\mathsf{P}}$, along with an estimate of the stationary variance, $\hat{v} = \hat{\gamma}_0$, we can compute parameter estimates

$$\hat{\boldsymbol{\phi}} = \hat{\mathsf{P}}^{-1}\hat{\boldsymbol{\rho}}, \quad \hat{\sigma}^2 = \hat{\gamma}_0(1 - \hat{\boldsymbol{\rho}}^\top\hat{\mathsf{P}}^{-1}\hat{\boldsymbol{\rho}}).$$

4.1.1.1.1 Example: AR(2)

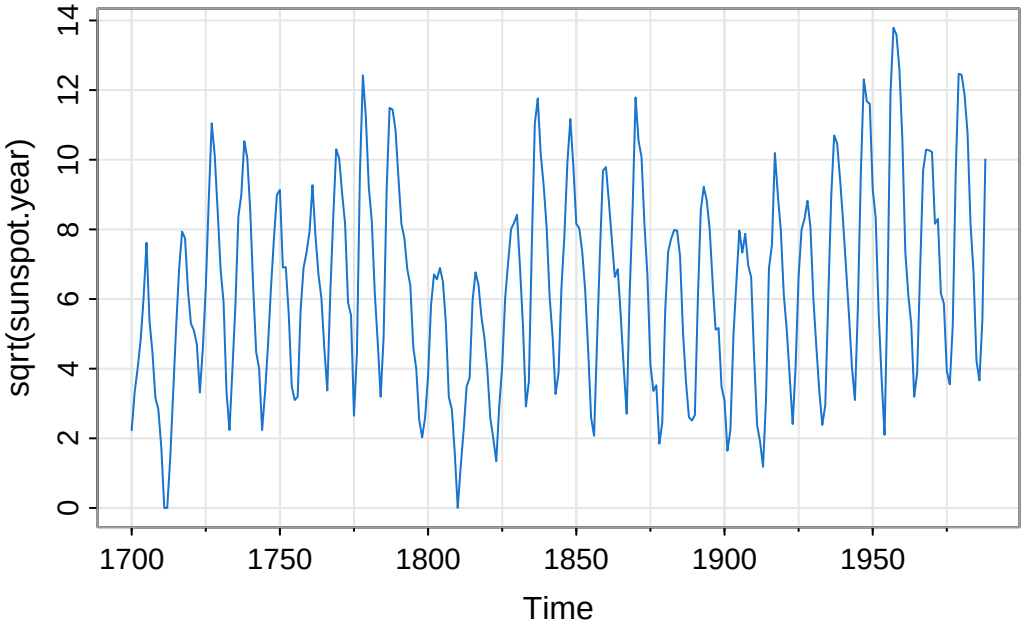
Let's look at some of the sunspot data built in to R (`?sunspot.year`).

```
library(astsa)
tsplot(sunspot.year, col=4, main="Annual sunspot data")
```



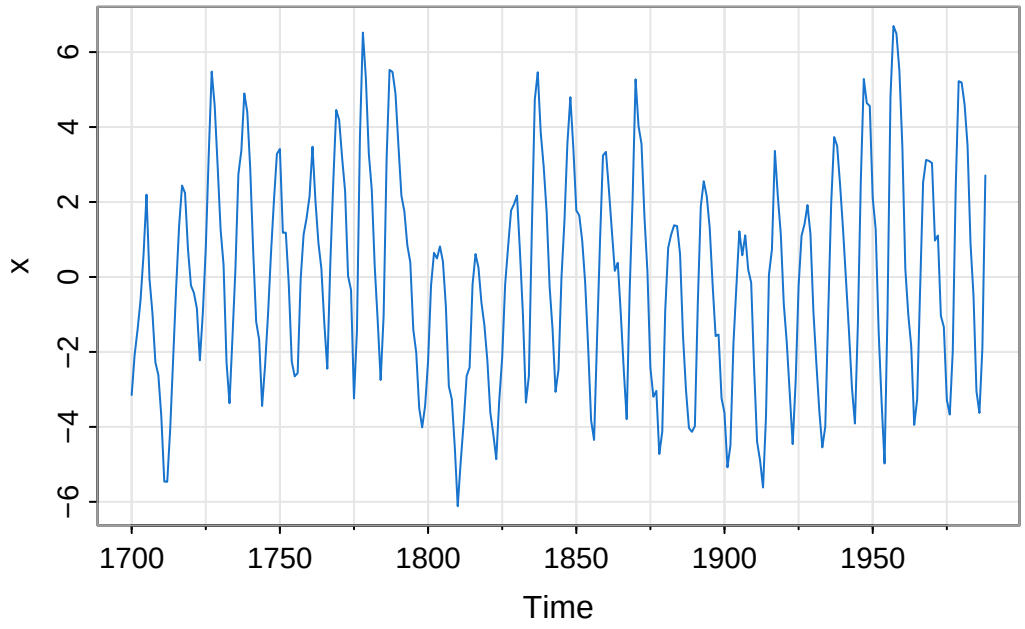
Since it is count data, it is a bit right-skewed, so the square root is a bit more Gaussian.

```
tsplot(sqrt(sunspot.year), col=4)
```



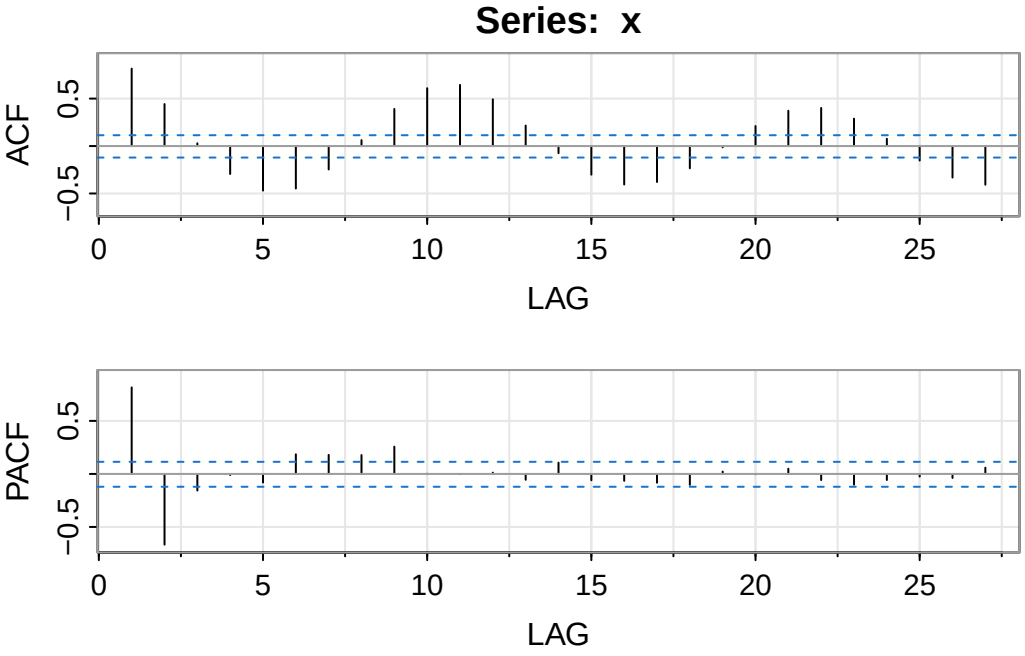
and we should detrend before fitting an ARMA model.

```
x = detrend(sqrt(sunspot.year))
tsplot(x, col=4)
```



We can see some periodicity to this data, further revealed by looking at the ACF

```
acf2(x)
```




```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7] [,8] [,9] [,10] [,11] [,12] [,13]
ACF   0.82  0.44  0.03 -0.29 -0.47 -0.45 -0.25 0.06 0.39  0.61  0.64  0.49  0.22
PACF 0.82 -0.67 -0.16 -0.01 -0.08  0.19  0.18 0.18 0.26  0.00  0.00  0.01 -0.06
      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
ACF   -0.08 -0.30 -0.41 -0.38 -0.23 -0.01  0.21  0.37  0.40  0.29  0.08 -0.15
```

```
PACF  0.11 -0.06 -0.07 -0.08 -0.10  0.02  0.00  0.05 -0.06 -0.10 -0.06 -0.02
     [,26] [,27]
ACF  -0.33 -0.41
PACF -0.04  0.06
```

The ACF suggests a period of between 10 and 11 years. The PACF suggests that something like an AR(2) may be appropriate for this data, so that is what we will fit.

```
rho = acf(x, plot=FALSE, lag.max=2)$acf[2:3]
P = matrix(c(1, rho[1], rho[1], 1), ncol=2)
phi = solve(P, rho)
phi
```

```
[1]  1.3602493 -0.6671228
```

These coefficients correspond to complex roots of the characteristic polynomial, as we might expect given the oscillatory nature of the data. We can confirm the suggested period of the oscillations.

```
u  = polyroot(c(1, -phi))[1]
f = Arg(u)/(2*pi)
1/f
```

```
[1] 10.7068
```

So the period of the oscillations is a little under 11 years.

4.1.1.2 MA and ARMA models

We can imagine doing a similar moment matching approach for an MA(q) process (or a more general ARMA model), but it doesn't work out so nicely, due to the ACF being non-linear in the parameters. We could investigate this in more detail, but moment matching isn't such a great parameter estimation strategy in general, so let's just move on.

4.1.2 Least squares

As an alternative to moment matching, we can approach parameter estimation as a least squares problem, where we try to find the parameters that minimise the sum of squares of the noise terms, ε_t. Again this is simpler in the case of an AR(p) model, so we begin with that.

4.1.2.1 AR(p)

The model

$$x_t = \sum_{k=1}^{p} \phi_k x_{t-k} + \varepsilon_t, \quad t = 1, 2, \ldots, n$$

is actually in the form of a multiple linear regression model. We can therefore treat it as a linear least squares problem, choosing parameters $\boldsymbol{\phi}$ to minimise $\mathcal{L}(\boldsymbol{\phi}) = \sum_{t=1}^{n} \varepsilon_t^2$. We can simplify the problem by ignoring the initialisation problem by conditioning on the first p observations and ignoring their contribution to the loss

function. This gives us a so-called *conditional* least squares problem. We can write the model in matrix form as

$$\begin{pmatrix} x_{p+1} \\ x_{p+2} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} x_p & x_{p-1} & \cdots & x_1 \\ x_{p+1} & x_p & \cdots & x_2 \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1} & x_{n-2} & \cdots & x_{n-p} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_p \end{pmatrix} + \begin{pmatrix} \varepsilon_{p+1} \\ \varepsilon_{p+2} \\ \vdots \\ \varepsilon_n \end{pmatrix},$$

or

$$\mathbf{x} = \mathsf{X}\boldsymbol{\phi} + \boldsymbol{\varepsilon},$$

for appropriately defined $(n-p)$-vectors $\mathbf{x}$ and $\boldsymbol{\varepsilon}$ and $(n-p) \times p$ matrix, X. We then know from basic least squares theory that

$$\mathcal{L}(\boldsymbol{\phi}) = \sum_{t=p+1}^{n} \varepsilon_t^2 = \|\boldsymbol{\varepsilon}\|^2 = \boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon}$$

is minimised by choosing $\boldsymbol{\phi}$ to be the solution of the *normal equations*,

$$\mathsf{X}^\top \mathsf{X} \boldsymbol{\phi} = \mathsf{X}^\top \mathbf{x},$$

in other words

$$\hat{\boldsymbol{\phi}} = (\mathsf{X}^\top \mathsf{X})^{-1} \mathsf{X}^\top \mathbf{x}.$$

We can then estimate σ as the empirical variance of the residuals.

4.1.2.1.1 Example: AR(2)

Let us now fit the sunspot data example again, but this time using least squares.

```
n = length(x)
X = cbind(x[2:(n-1)], x[1:(n-2)])
xp = x[3:n]
lm(xp ~ 0 + X)
```

```
Call:
lm(formula = xp ~ 0 + X)

Coefficients:
     X1        X2
 1.4032   -0.7086
```

We use R's `lm` function to solve the least squares problem. Note that the coefficients are similar, but not identical, to those obtained earlier.

4.1.2.1.2 Connection between moment matching and least squares

It is clear from the definition of the matrix X, that its associated sample covariance matrix, $\mathsf{X}^\top \mathsf{X}/(n-p)$ will be (unbiased and) consistent for Γ (recall that there are $n-p$ rows in X). Then by considering a single column of the covariance matrix, similar reasoning reveals that $\mathsf{X}^\top \mathbf{x}/(n-p)$ will be consistent for $\boldsymbol{\gamma}$. So if we wanted, we could use the finite sample estimates:

$$\hat{\Gamma} = \frac{\mathsf{X}^\top \mathsf{X}}{n-p}, \quad \hat{\boldsymbol{\gamma}} = \frac{\mathsf{X}^\top \mathbf{x}}{n-p}.$$

But if we do this, we find that our moment matching estimator,

$$\hat{\boldsymbol{\phi}} = \hat{\Gamma}^{-1} \hat{\boldsymbol{\gamma}} = (\mathsf{X}^\top \mathsf{X})^{-1} \mathsf{X}^\top \mathbf{x}$$

is just the least squares estimator. So the two approaches are asymptotically equivalent. This is the real reason why moment matching actually works quite well for AR models.

4.1.2.2 ARMA(p,q)

The approach for a general ARMA model is similar to that for an AR model, but with an extra twist. Again, the problem simplifies if we ignore the contribution associated with the first p observations. But now we also have an issue with initialising the errors. Again, we can simplify by conditioning on the first q required errors. But here we don't actually know what to condition on, so we just condition on them all being zero. This *conditional* least squares approach is again approximate, but has negligible impact when n is large. We can then use the least squares loss function

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) = \sum_{t=p+1}^{n} \varepsilon_t^2 \quad \text{where} \quad \varepsilon_t = x_t - \sum_{j=1}^{p} \phi_j x_{t-j} - \sum_{k=1}^{q} \theta_k \varepsilon_{t-k},$$

and we assume $\varepsilon_t = 0, \ \forall t \leq p$. For $q > 0$ this is a *nonlinear* least squares problem, so we need to minimise it numerically. There are efficient ways to do this, but the details are not especially relevant to this course.

4.1.2.2.1 Example: ARMA(2, 1)

Just for fun, let's fit an ARMA(2, 1) to the sunspot data.

```
loss = function(param) {
  phi = param[1:2]; theta = param[3]
  eps = signal::filter(c(1, -phi), c(1, theta),
    x[3:length(x)], init.x=x[1:2], init.y=c(0))
  sum(eps*eps)
}

optim(rep(0.1,3), loss)$par
```

```
[1]  1.4840176 -0.7749327 -0.1623121
```

The first two elements of the parameter vector correspond to $\boldsymbol{\phi}$, and the last corresponds to θ. In practice, we will typically use the built-in function `arima` to fit ARMA models to data.

```
arima(x, c(2, 0, 1), include.mean=FALSE)
```

```
Call:
arima(x = x, order = c(2, 0, 1), include.mean = FALSE)

Coefficients:
         ar1      ar2      ma1
      1.4828  -0.7733  -0.1631
s.e.  0.0516   0.0465   0.0785

sigma^2 estimated as 1.331:  log likelihood = -452.69,  aic = 913.39
```

The `arima` function fits using the method of maximum likelihood, considered next.

4.1.3 Maximum likelihood

Neither moment matching nor least squares are particularly principled estimation methods, so we next turn our attention to the method of maximum likelihood. As usual, the AR case is simpler, so we begin with that.

4.1.3.1 AR(p)

We can always write the likelihood function of any model as a recursive factorisation of the joint distribution of the data. So, for an AR model we could write

$$\begin{aligned} L(\boldsymbol{\phi}, \sigma; \mathbf{x}) &= f(\mathbf{x}) \\ &= f(x_1) f(x_2|x_1) \cdots f(x_n|x_1, \ldots, x_{n-1}) \end{aligned}$$

In other words,

$$L(\boldsymbol{\phi}, \sigma; \mathbf{x}) = f(x_1) \prod_{t=2}^{n} f(x_t|x_1, \ldots, x_{t-1}) \tag{4.1}$$

This holds for any model, but for an AR(p), X_t depends only on the previous p values, so we can write this as

$$L(\boldsymbol{\phi}, \sigma; \mathbf{x}) = f(x_1, \ldots, x_p) \prod_{t=p+1}^{n} f(x_t|x_{t-1}, \ldots, x_{t-p}),$$

where $f(x_1, \ldots, x_p)$ is the joint (MVN) distribution of p consecutive observations, and

$$f(x_t|x_{t-1}, \ldots, x_{t-p}) = \mathcal{N}\left(x_t; \sum_{k=1}^{p} \phi_k x_{t-k}, \sigma^2\right),$$

the normal density function. We could directly maximise this likelihood using iterative numerical methods (see below). However, the likelihood obviously simplifies significantly if we ignore the contribution of the first p observations, leading to the *conditional* log-likelihood

$$\begin{aligned} \ell(\boldsymbol{\phi}, \sigma; \mathbf{x}) &= -(n-p) \log \sigma - \frac{1}{2\sigma^2} \sum_{t=p+1}^{n} \left(x_t - \sum_{k=1}^{p} \phi_k x_{t-k}\right)^2 \\ &= -(n-p) \log \sigma - \frac{1}{2\sigma^2} \sum_{t=p+1}^{n} \varepsilon_t^2. \end{aligned}$$

We can immediately see that as a function of $\boldsymbol{\phi}$, maximising this leads to exactly the least squares problem that we have already considered, and so our MLE can be obtained by solving the normal equations. The effect of conditioning on the first p observations will be negligible for long time series, but if we'd rather do exact MLE fitting using the *unconditional* likelihood, we just include the joint density of the first p observations and numerically maximise (possibly using the conditional fit as an initial guess). Recall that (assuming we are in the stationary regime) the joint density of the first p observations will be $\mathcal{N}(\mathbf{0}, \mathsf{V})$, where V is the stationary variance matrix obtained by solving the relevant discrete Lyapunov equation.

4.1.3.1.1 Example: AR(2) MLE

Let us define and optimise the (exact) log likelihood for an AR(2) fitted to the sunspot data.

```
ll = function(param) {
  phi = param[1:2]; sig = param[3]
  eps = filter(x, c(1, -phi))[2:(length(x)-1)]
  V = netcontrol::dlyap(matrix(c(phi[1], phi[2], 1, 0), ncol=2),
    matrix(c(sig*sig, 0, 0, 0), ncol=2))
  (mvtnorm::dmvnorm(x[1:2], c(0, 0), V, log=TRUE)
    - (length(x)-2)*log(sig)
    - sum(eps*eps)/(2*sig*sig)
    )
  }
```

```
optim(c(1.0, -0.2, 1.0), ll, control=list(fnscale=-1))$par
```

```
[1]  1.4016403 -0.7067519  1.1619802
```

This fit is very similar to the conditional (least squares) AR(2) fit obtained earlier, but now essentially identical to that obtained using the `arima` function:

```
arima(x, c(2, 0, 0), include.mean=FALSE)
```

```
Call:
arima(x = x, order = c(2, 0, 0), include.mean = FALSE)

Coefficients:
         ar1      ar2
      1.4017  -0.7068
s.e.  0.0422   0.0422

sigma^2 estimated as 1.35:  log likelihood = -454.71,  aic = 915.41
```

4.1.3.2 ARMA(p,q)

For a general ARMA model, we can start from the general factorisation Equation 4.1, and note again that the problem would simplify enormously if we not only conditioned on the first p observations, but also on the first q required errors. In this case there is a deterministic relationship between X_t and ε_t, and the required likelihood terms are just the densities of the new error ε_t at each time. So again, in the conditional case, we get the simplified log-likelihood

$$\ell(\boldsymbol{\phi}, \boldsymbol{\theta}, \sigma; \mathbf{x}) = -(n-p)\log\sigma - \frac{1}{2\sigma^2}\sum_{t=p+1}^{n}\varepsilon_t^2,$$

where now $\varepsilon_t = x_t - \sum_{j=1}^{p}\phi_j x_{t-j} - \sum_{k=1}^{q}\theta_k\varepsilon_{t-k}$. This is a nonlinear least squares problem that we need to solve with iterative numerical methods, as we have already seen.

4.1.4 Bayesian inference

Bayesian inference arguably provides a more principled approach to parameter estimation than any we have yet considered. However, the computational details are somewhat tangential to the main focus of this course. The key ingredients are the likelihood functions that we have already considered, and a prior on the model parameters. In general we can use sampling methods such as MCMC to explore the resulting posterior distribution. Note that a Gaussian prior for $\boldsymbol{\phi}$ is (conditionally) conjugate for ARMA models, so significant simplifications arise, especially in the AR case, or in conjunction with (block) Gibbs sampling approaches. The precise details are non-examinable in the context of this course.

4.2 Forecasting an ARMA model

4.2.1 Forecasting

Forecasting is the problem of making predictions about possible future values that a time series might take. Suppose we have time series observations $x_1, x_2, \ldots, x_n$, and that we are currently at time n, and wish to make predictions about future, currently unobserved, values of the time series, $X_{n+1}, X_{n+2}, \ldots$. We will use the notation $\hat{x}_n(k)$ for the *forecast* made at time n for the observation X_{n+k} (k time points into the future). At time n, $x_n(k)$ will be deterministic (otherwise we could just choose $\hat{x}_n(k) = X_{n+k}$!), but will be informed by the observed time series up to time n.

How should we choose $\hat{x}_n(k)$? We know that we are unlikely to be able to predict X_{n+k} exactly, but we would like to be as close as possible. So we would like to try and minimise some sort of penalty for how wrong we are. So, suppose that at time n we declare $\hat{x}_n(k) = x$, and that at time $n + k$ we see the observed value of $X_{n+k} = x_{n+k}$ are are subject to the penalty

$$p(x) = c(x_{n+k} - x)^2,$$

for some constant c in units of "penalty" (eg. British pounds). If our prediction is perfect, we will not be penalised, and the penalty increases depending on how wrong we are. At time n, we don't know what penalty we will receive, so it is random,

$$P(x) = c(X_{n+k} - x)^2.$$

We might therefore like to minimise the penalty that we expect to receive, given all of the information we have at time n. That is, we want to minimise the loss function

$$\begin{aligned}\mathcal{L}(x) &= \mathbb{E}[P(x)|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}] \\ &= \mathbb{E}[c(X_{n+k} - x)^2|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}] \\ &= c\mathbb{E}[X_{n+k}^2|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}] - 2cx\mathbb{E}[X_{n+k}|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}] + cx^2.\end{aligned}$$

This is quadratic in x, so we can minimise wrt x, either by completing the square, or by differentiating and equating to zero to get

$$\hat{x}_n(k) = \mathbb{E}[X_{n+k}|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}].$$

So, somewhat unsurprisingly, our forecast for X_{n+k} made at time n should be just its conditional expectation given the observations up to time n. Similarly, our uncertainty regarding the future observation can be well summarised by the corresponding conditional variance

$$\mathbb{V}\mathrm{ar}[X_{n+k}|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}],$$

noting that c times this is the expected penalty we will receive. For stationary Gaussian processes such as ARMA models, it is straightforward, in principle, to compute these conditional distributions using standard normal theory. However, for large n this involves large matrix computations. So typically forecast distributions are computed sequentially, exploiting the causal structure of the process.

4.2.2 Forecasting an AR(p) model

As usual, the AR(p) case is slightly simpler than that of a general ARMA model, so we start with this. We assume that we know $\mathbf{x}_{1:n}$, and want to sequentially compute $\hat{x}_n(1), \hat{x}_n(2), \ldots$. First,

$$\begin{aligned}
\hat{x}_n(1) &= \mathbb{E}[X_{n+1}|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}] \\
&= \mathbb{E}\left[\sum_{j=1}^{p} \phi_j X_{n+1-j} + \varepsilon_{n+1} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}\right] \\
&= \mathbb{E}\left[\sum_{j=1}^{p} \phi_j x_{n+1-j} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}\right] \\
&= \sum_{j=1}^{p} \phi_j x_{n+1-j}.
\end{aligned}$$

Similarly,

$$\begin{aligned}
\hat{x}_n(2) &= \mathbb{E}[X_{n+2}|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}] \\
&= \mathbb{E}\left[\sum_{j=1}^{p} \phi_j X_{n+2-j} + \varepsilon_{n+2} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}\right] \\
&= \mathbb{E}\left[\phi_1 X_{n+1} + \sum_{j=2}^{p} \phi_j x_{n+2-j} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}\right] \\
&= \phi_1 \mathbb{E}\left[X_{n+1} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}\right] + \sum_{j=2}^{p} \phi_j x_{n+2-j} \\
&= \phi_1 \hat{x}_n(1) + \sum_{j=2}^{p} \phi_j x_{n+2-j}.
\end{aligned}$$

By now it should be clear that we will have

$$\hat{x}_n(3) = \sum_{j=1}^{2} \phi_j \hat{x}_n(3-j) + \sum_{j=3}^{p} \phi_j x_{n+3-j}.$$

The notation will be greatly simplified if we drop the distinction between forecasts and observations by *defining* $\hat{x}_n(-t) = x_{n-t}$, for $t = 0, 1, \ldots, p$. Then we have

$$\hat{x}_n(3) = \sum_{j=1}^{p} \phi_j \hat{x}_n(3-j),$$

and more generally,

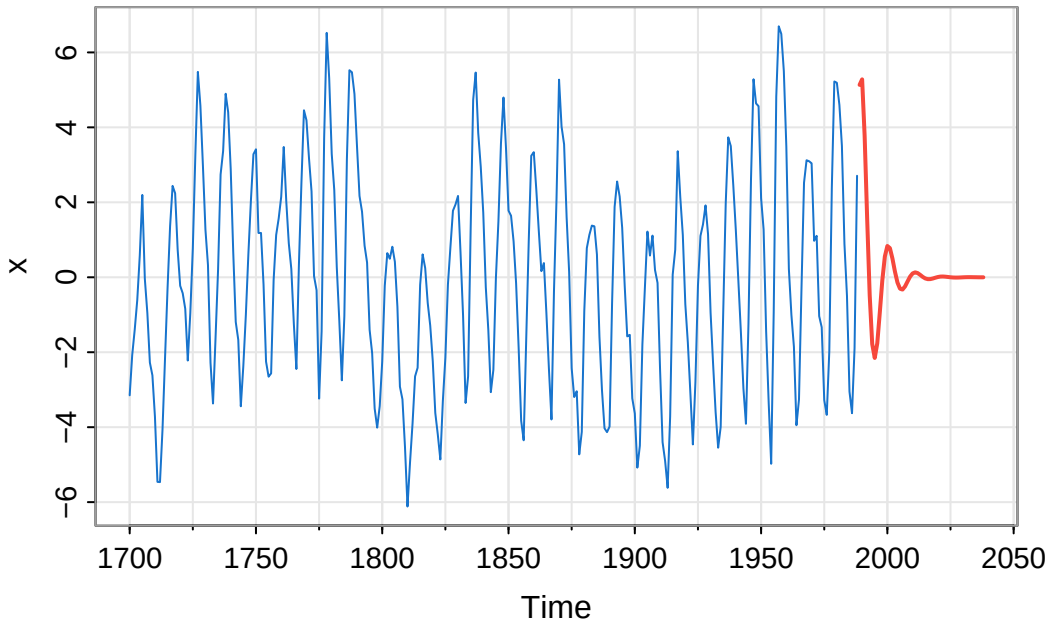
$$\hat{x}_n(k) = \sum_{j=1}^{p} \phi_j \hat{x}_n(k-j), \quad k = 1, 2, \ldots$$

That is, the forecasts satisfy the obvious pth order linear recurrence relation, initialised with the last p observed values of the time series.

4.2.2.1 Example: AR(2)

Let's compute the forecast function for an AR(2) model fit to the sunspot datata.

```
n = length(x); k=50
phi = arima(x, c(2,0,0))$coef[1:2]
fore = filter(rep(0,k), phi, "rec", init=x[n:(n-1)])
tsplot(x, xlim=c(tsp(x)[1], tsp(x)[2]+k), col=4)
lines(seq(tsp(x)[2]+1, tsp(x)[2]+k, frequency(x)), fore,
  col=2, lwd=2)
```



So, the short term forecasts oscillate in line with the recent observations, but the longer term forecasts quickly decay away to the stationary mean of zero as we become increasingly uncertain about the phase of the signal.

4.2.2.2 Forecast variance

Let us now turn attention to the forecast variances. Again, let's compute them sequentially, starting with the one-step ahead variance.

$$\begin{aligned}
\mathbb{V}\mathrm{ar}[X_{n+1}|\mathbf{X}_{1:n} = \mathbf{x}_{1:n}] &= \mathbb{V}\mathrm{ar}\left[\sum_{j=1}^{p} \phi_j X_{n+1-j} + \varepsilon_{n+1} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}\right] \\
&= \mathbb{V}\mathrm{ar}\left[\sum_{j=1}^{p} \phi_j x_{n+1-j} + \varepsilon_{n+1} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}\right] \\
&= \mathbb{V}\mathrm{ar}\left[\varepsilon_{n+1} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}\right] \\
&= \sigma^2.
\end{aligned}$$

Next, let's consider the 2-step ahead forecast variance,

$$\begin{aligned}
\mathbb{V}\text{ar}[X_{n+2}|\mathbf{X}_{1:n}=\mathbf{x}_{1:n}] &= \mathbb{V}\text{ar}\left[\sum_{j=1}^{p}\phi_j X_{n+2-j}+\varepsilon_{n+2}\middle|\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]\\
&= \mathbb{V}\text{ar}\left[\phi_1 X_{n+1}+\sum_{j=2}^{p}\phi_j x_{n+2-j}+\varepsilon_{n+2}\middle|\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]\\
&= \phi_1^2\mathbb{V}\text{ar}\left[X_{n+1}|\,\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]+\sigma^2\\
&= (1+\phi_1^2)\sigma^2.
\end{aligned}$$

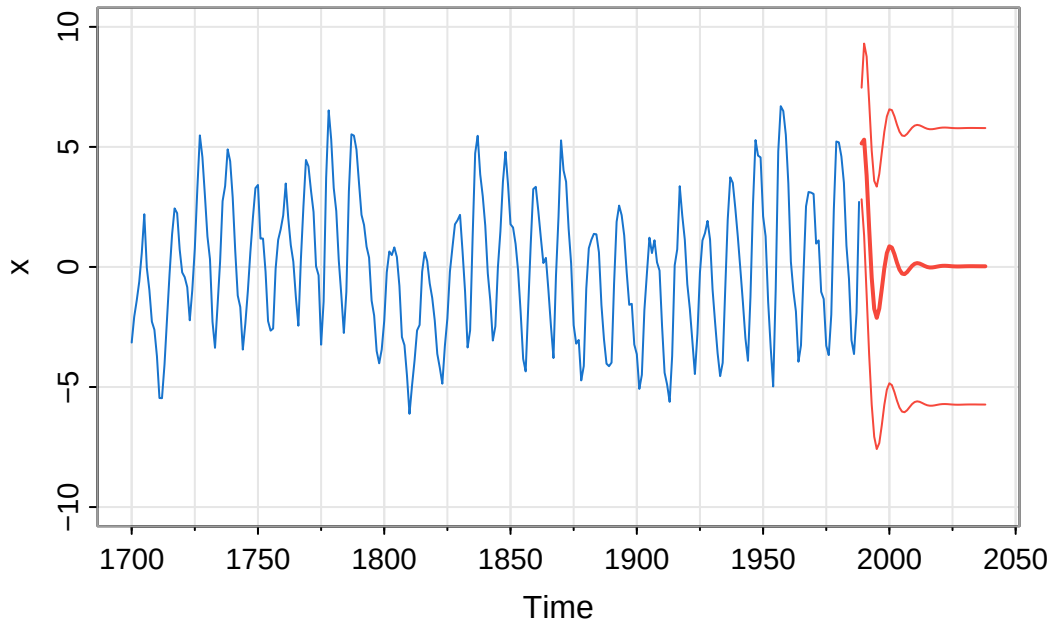
Beyond this, things start to get a bit more complicated, due to the covariances between observations. Nevertheless, we can continue to slog out the variances using successive substitution.

$$\begin{aligned}
\mathbb{V}\text{ar}[X_{n+3}|\mathbf{X}_{1:n}=\mathbf{x}_{1:n}] &= \mathbb{V}\text{ar}\left[\sum_{j=1}^{p}\phi_j X_{n+3-j}+\varepsilon_{n+3}\middle|\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]\\
&= \mathbb{V}\text{ar}\left[\phi_1 X_{n+2}+\phi_2 X_{n+1}+\sum_{j=3}^{p}\phi_j x_{n+3-j}+\varepsilon_{n+3}\middle|\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]\\
&= \mathbb{V}\text{ar}\left[\phi_1 X_{n+2}+\phi_2 X_{n+1}|\,\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]+\sigma^2\\
&= \mathbb{V}\text{ar}\left[\phi_1\left(\phi_1 X_{n+1}+\sum_{j=2}^{p}\phi_k x_{n+2-j}+\varepsilon_{n+2}\right)+\phi_2 X_{n+1}\middle|\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]+\sigma^2\\
&= \mathbb{V}\text{ar}\left[(\phi_1^2+\phi_2)X_{n+1}+\phi_1\varepsilon_{n+2}\middle|\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]+\sigma^2\\
&= (\phi_1^2+\phi_2)^2\mathbb{V}\text{ar}\left[X_{n+1}|\,\mathbf{X}_{1:n}=\mathbf{x}_{1:n}\right]+(1+\phi_1^2)\sigma^2\\
&= \left(1+\phi_1^2+[\phi_1^2+\phi_2]^2\right)\sigma^2
\end{aligned}$$

It is possible to derive recursions that give the forecast variances, but it is probably not worth it. Note that the case of the AR(1) was dealt with in Chapter 2, where explicit forecast distributions were derived. The initial condition there becomes the final observation, x_n, here. Similarly, we can write an AR(p) as a VAR(1), and use the explicit expressions for the forecast variance for the VAR(1) from Chapter 2 to deduce the forecast variance for the AR(p).

In practice, we just use the `arima` function to compute forecasts and associated uncertainties.

```
mod = arima(x, c(2,0,0))
fore = predict(mod, n.ahead=k)
pred = fore$pred
sds = fore$se
tsplot(x, xlim=c(tsp(x)[1], tsp(x)[2]+k),
  ylim=c(-10, 10), col=4)
ftimes = seq(tsp(x)[2]+1, tsp(x)[2]+k, tsp(x)[3])
lines(ftimes, pred, col=2, lwd=2)
lines(ftimes, pred+2*sds, col=2)
lines(ftimes, pred-2*sds, col=2)
```



4.2.3 Forecasting an ARMA(p,q)

As usual, the case of an ARMA model is similar to that of an AR model, but a little bit more complicated due to the extra error terms. Recall from our discussion of conditional least squares and conditional likelihood approaches to model fitting, that for long time series, there is a deterministic relationship between x_t and ε_t. Just as we can simulate an ARMA model by applying an ARMA filter to white noise, we can also recover the error process by applying an ARMA filter to the time series. In the context of forecasting, this means that if we know the observed values of the time series up to time n, we also know the observed values of the errors up to time n. We will denote these $\tilde{\varepsilon}_1, \tilde{\varepsilon}_2, \ldots, \tilde{\varepsilon}_n$ in order to emphasise that these are observed and not random. Given this, we can just proceed as before. It is clear that the one-step ahead predictive variance for any ARMA model will be σ^2, but variance forecasts further ahead get quite cumbersome quite quickly. The (mean) forecast function, $\hat{x}_n(k)$, is more straightforward.

For the ARMA model

$$X_t = \sum_{j=1}^{p} \phi_j X_{t-j} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t,$$

we have

$$\begin{aligned}
\hat{x}_n(k) &= \mathbb{E}\left[X_{n+k} \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}, \boldsymbol{\varepsilon}_{1:n} = \tilde{\boldsymbol{\varepsilon}}_{1:n}\right] \\
&= \mathbb{E}\left[\sum_{j=1}^{p} \phi_j X_{n+k-j} + \sum_{j=1}^{q} \theta_j \varepsilon_{n+k-j} + \varepsilon_t \middle| \mathbf{X}_{1:n} = \mathbf{x}_{1:n}, \boldsymbol{\varepsilon}_{1:n} = \tilde{\boldsymbol{\varepsilon}}_{1:n}\right] \\
&= \sum_{j=1}^{p} \phi_j \mathbb{E}[X_{n+k-j} | \mathbf{X}_{1:n} = \mathbf{x}_{1:n}] + \sum_{j=1}^{q} \theta_j \mathbb{E}[\varepsilon_{n+k-j} | \boldsymbol{\varepsilon}_{1:n} = \tilde{\boldsymbol{\varepsilon}}_{1:n}].
\end{aligned}$$

Now we define $\hat{x}_n(-t) = x_{n-t}$ for $t \geq 0$, as before, and also now define $\tilde{\varepsilon}_t = 0$ for $t > n$, to get

$$\hat{x}_n(k) = \sum_{j=1}^{p} \phi_j \hat{x}_n(k-j) + \sum_{j=1}^{q} \theta_j \tilde{\varepsilon}_{n+k-j}. \tag{4.2}$$

So for $k = 1, 2, \ldots, q$ we explicitly enumerate $\hat{x}_n(k)$ using Equation 4.2, and then for $k > q$ we have the linear recurrence

$$\hat{x}_n(k) = \sum_{j=1}^{p} \phi_j \hat{x}_n(k-j),$$

as for the AR(p).

5 Spectral analysis

5.1 Fourier analysis

Many time series exhibit repetitive pseudo-periodic behaviour. It is therefore tempting to try to use (mixtures of) trigonometric functions in order to capture aspects of their behaviour. The weights and frequencies associated with the trig functions will give us insight into the nature of the process. This is the idea behind spectral analysis.

In fact, we know from Fourier analysis that any reasonable function can be described as a mixture of trig functions, so we begin with a very brief recap of Fourier series before thinking about about how to use these ideas in the context of time series models.

5.1.1 Fourier series

Starting from any "reasonable" function, $f : [-\frac{1}{2}, \frac{1}{2}] \to \mathbb{R}$ (or, equivalently, any periodic function with period 1), we can write

$$f(x) = a_0 + \sum_{k=1}^{\infty} (a_k \cos 2\pi kx + b_k \sin 2\pi kx)$$

for coefficients $a_0, a_1, \ldots, \ b_1, b_2, \ldots \in \mathbb{R}$ to be determined. However, since trig is tricky, it is often more convenient to write this as

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{2\pi \mathrm{i} kx},$$

for $\ldots, c_{-1}, c_0, c_1, \ldots \in \mathbb{C}$. The sum is now doubly-infinite, since we can ensure we get back to a real-valued Fourier series by choosing $c_{-k} = \bar{c}_k \ \forall \ k \geq 0$ (and hence $c_0 \in \mathbb{R}$). Now, using the nice orthogonality property

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} e^{2\pi \mathrm{i}(j-k)x} \mathrm{d}x = \delta_{jk} = \begin{cases} 1 & j = k \\ 0 & \text{otherwise,} \end{cases}$$

we can multiply through our series by $e^{-2\pi \mathrm{i} jx}$ and integrate to determine the coefficients,

$$c_k = \int_{-\frac{1}{2}}^{\frac{1}{2}} f(x) e^{-2\pi \mathrm{i} kx} \mathrm{d}x.$$

It is a standard (but non-trivial) result of Fourier analysis that this series will converge to $f(x)$ (in, say, an L^2 sense), for reasonably well-behaved functions, f.

The main point of Fourier series is to represent a periodic (or finite range) function wrt to a countable orthonormal set of (trigonometric) basis functions. But since the mapping is invertible, it just provides us with a mapping back and forth between a function defined on $[-\frac{1}{2}, \frac{1}{2}]$ and a countable set of coefficients. So, we could alternatively view this as a way of representing a countable collection of numbers using a function defined on $[-\frac{1}{2}, \frac{1}{2}]$. This latter perspective is helpful for the analysis of time series in discrete time, and in this context, the Fourier mapping is known as the discrete time Fourier transform (DTFT).

5.1.2 Discrete time Fourier transform

An infinite time series $\ldots, x_{-1}, x_0, x_1, \ldots$ can be represented by a function $\hat{x} : [-\frac{1}{2}, \frac{1}{2}] \to \mathbb{C}$ via the relation

$$x_t = \int_{-\frac{1}{2}}^{\frac{1}{2}} \hat{x}(\nu) e^{2\pi i t \nu} \mathrm{d}\nu, \quad t \in \mathbb{Z},$$

where

$$\hat{x}(\nu) = \sum_{t=-\infty}^{\infty} x_t e^{-2\pi i t \nu}.$$

$\hat{x}(\nu)$ is the weight put on oscillations of frequency ν. More precisely, $|\hat{x}(\nu)|$ determines the weight, and $\mathrm{Arg}[\hat{x}(\nu)]$ determines the phase of the oscillation at that frequency. Note that the sign of the continuous variable (ν) has been switched. This doesn't change anything - it just maintains the usual convention that the forward transform has a minus in the complex exponential and the inverse transform does not. Also note that if the time series is real-valued, then the function $\hat{x}$ will be Hermitian. That is, it will have the property $\hat{x}(-\nu) = \overline{\hat{x}(\nu)}, \forall \nu \in [0, \frac{1}{2}]$ (and $\hat{x}(0)$ will be real). Note that no fundamentally new maths is required for this representation - it is just Fourier series turned on its head. That said, there are lots of technical convergence conditions that we are ignoring.

5.2 Spectral representation

Now we have this bijection between discrete-time time series and functions on a unit interval, we can think about what it means for random time series. Clearly a random time series X_t will induce a random continuous-time process $\hat{X}(\nu)$, and *vice-versa*. It will turn out to be very instructive to think about the class of discrete time models induced by a weighted "white noise" process. That is, $\hat{X}(\nu) = \sigma(\nu)\eta(\nu)$, where $\sigma : [-\frac{1}{2}, \frac{1}{2}] \to \mathbb{C}$ is a deterministic function specifying the weight to be placed on the frequency ν, and $\eta(\nu)$ is the stationary Gaussian "white noise" process with $\mathbb{E}[\eta(\nu)] = 0$, $\gamma(\nu) = \delta(\nu)$ (Dirac's delta). We can therefore represent our random time series as

$$X_t = \int_{-\frac{1}{2}}^{\frac{1}{2}} \sigma(\nu) e^{2\pi i t \nu} \eta(\nu) \, \mathrm{d}\nu, \quad t \in \mathbb{Z}. \tag{5.1}$$

Now, since η is not a very nice function, it is arguably better to write this as a stochastic integral wrt a Wiener process, W, as

$$X_t = \int_{-\frac{1}{2}}^{\frac{1}{2}} \sigma(\nu) e^{2\pi i t \nu} \, \mathrm{d}W(\nu), \quad t \in \mathbb{Z},$$

interpreted using Itô calculus. We will largely gloss over this technicality, but writing it this way arguably makes it easier to see that the induced time series will be a Gaussian process. Taking the expectation of Equation 5.1 gives $\mathbb{E}[X_t] = 0$, so then

$$\begin{aligned}
\mathbb{C}\mathrm{ov}[X_{t+k}, X_t] &= \mathbb{E}[X_{t+k}\bar{X}_t] \\
&= \mathbb{E}\left[\int_{-\frac{1}{2}}^{\frac{1}{2}} \sigma(\nu) e^{2\pi \mathrm{i}(t+k)\nu} \eta(\nu) \, \mathrm{d}\nu \int_{-\frac{1}{2}}^{\frac{1}{2}} \bar{\sigma}(\nu') e^{-2\pi i t \nu'} \bar{\eta}(\nu') \, \mathrm{d}\nu'\right] \\
&= \mathbb{E}\left[\int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{d}\nu \int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{d}\nu' \, \sigma(\nu)\bar{\sigma}(\nu') e^{2\pi \mathrm{i}[(t+k)\nu - t\nu']} \eta(\nu)\bar{\eta}(\nu')\right] \\
&= \int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{d}\nu \int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{d}\nu \, \sigma(\nu)\bar{\sigma}(\nu) e^{2\pi \mathrm{i}[t(\nu-\nu')+k\nu]} \mathbb{E}[\eta(\nu)\bar{\eta}(\nu')] \\
&= \int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{d}\nu \int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{d}\nu' \, \sigma(\nu)\bar{\sigma}(\nu') e^{2\pi \mathrm{i}[t(\nu-\nu')+k\nu]} \delta(\nu - \nu') \\
&= \int_{-\frac{1}{2}}^{\frac{1}{2}} \mathrm{d}\nu \, |\sigma(\nu)|^2 e^{2\pi i k \nu}.
\end{aligned}$$

Since this is independent of t, the induced time series is weakly stationary, with auto-covariance function

$$\gamma_k = \int_{-\frac{1}{2}}^{\frac{1}{2}} |\sigma(\nu)|^2 \, e^{2\pi \mathrm{i} k\nu} \, \mathrm{d}\nu.$$

If we define the function

$$S(\nu) \equiv |\sigma(\nu)|^2,$$

then we can re-write this as

$$\gamma_k = \int_{-\frac{1}{2}}^{\frac{1}{2}} S(\nu) \, e^{2\pi \mathrm{i} k\nu} \, \mathrm{d}\nu. \tag{5.2}$$

This is just a discrete time Fourier transform, so we can invert it as

$$S(\nu) = \sum_{k=-\infty}^{\infty} \gamma_k e^{-2\pi \mathrm{i} k\nu}.$$

The non-negative real-valued function $S(\nu)$ is known as the spectral density of the discrete time series process. So, glossing over (quite!) a few technical details, essentially any choice of spectral density function leads to a stationary Gaussian discrete time process, and any stationary Gaussian discrete time process has an associated spectral density function. The spectral density function and auto-covariance function are different but equivalent ways of representing the same information. Technical mathematical results relating to this equivalence include the Wiener-Khinchin theorem and Bochner's theorem. Note that if $S(\nu)$ is chosen to be an even function, and it typically will, then

$$\gamma_k = 2 \int_0^{\frac{1}{2}} S(\nu) \, \cos(2\pi k\nu) \, \mathrm{d}\nu,$$

and will be real. Similarly, if γ_k is real (and hence even), as it typically will be, then

$$S(\nu) = \gamma_0 + 2 \sum_{k=1}^{\infty} \gamma_k \cos(2\pi k\nu),$$

and hence is even.

5.2.1 Spectral densities

5.2.1.1 White noise

What discrete time process is represented by a flat spectral density, with $S(\nu) = \sigma^2, \ \forall \nu$? Substituting in to Equation 5.2 gives

$$\gamma_k = \begin{cases} \sigma^2 & k = 0 \\ 0 & k > 0. \end{cases}$$

In other words, it induces the discrete time white noise process, ε_t, with noise variance σ^2.

5.2.1.2 Some properties of the DTFT

5.2.1.2.1 Linearity

The DTFT is a linear transformation. Suppose that $z_t = ax_t + by_t$ for some time series x_t and y_t. Then,

$$\begin{aligned}\hat{z}(\nu) &= \sum_{t=-\infty}^{\infty} z_t e^{-2\pi \mathrm{i} t\nu} \\ &= \sum_{t=-\infty}^{\infty} (ax_t + by_t) e^{-2\pi \mathrm{i} t\nu} \\ &= a\sum_{t=-\infty}^{\infty} x_t e^{-2\pi \mathrm{i} t\nu} + b\sum_{t=-\infty}^{\infty} y_t e^{-2\pi \mathrm{i} t\nu} \\ &= a\hat{x}(\nu) + b\hat{y}(\nu).\end{aligned}$$

5.2.1.2.2 Time shift

Suppose that $y_t = Bx_t = x_{t-1}$, for some time series x_t. Then,

$$\begin{aligned}\hat{y}(\nu) &= \sum_{t=-\infty}^{\infty} y_t e^{-2\pi \mathrm{i} t\nu} \\ &= \sum_{t=-\infty}^{\infty} x_{t-1} e^{-2\pi \mathrm{i} t\nu} \\ &= e^{-2\pi \mathrm{i}\nu} \sum_{t=-\infty}^{\infty} x_{t-1} e^{-2\pi \mathrm{i}(t-1)\nu} \\ &= e^{-2\pi \mathrm{i}\nu} \hat{x}(\nu),\end{aligned}$$

and more generally, it easily follows that

$$\widehat{B^k x}(\nu) = e^{-2\pi \mathrm{i} k\nu} \hat{x}(\nu).$$

5.2.1.3 AR(p)

Consider the AR(p) model,

$$X_t - \sum_{k=1}^{p} \phi_k X_{t-k} = \varepsilon_t,$$

and DTFT both sides.

$$\begin{aligned}\hat{X}(\nu) - \sum_{k=1}^{p} \phi_k e^{-2\pi \mathrm{i} k\nu} \hat{X}(\nu) &= \hat{\varepsilon}(\nu) \\ \Rightarrow \left(1 - \sum_{k=1}^{p} \phi_k e^{-2\pi \mathrm{i} k\nu}\right) \hat{X}(\nu) &= \hat{\varepsilon}(\nu) \\ \Rightarrow \phi(e^{-2\pi \mathrm{i}\nu}) \hat{X}(\nu) &= \hat{\varepsilon}(\nu) \\ \Rightarrow \hat{X}(\nu) &= \frac{\hat{\varepsilon}(\nu)}{\phi(e^{-2\pi \mathrm{i}\nu})}\end{aligned}$$

Now, since ε_t is a discrete time white noise process, it has a flat spectrum, and we can write $\hat{\varepsilon}(\nu) = \sigma\eta(\nu)$, so that

$$\hat{X}(\nu) = \frac{\sigma}{\phi(e^{-2\pi \mathrm{i}\nu})} \eta(\nu),$$

and from this it is clear that the spectral density of X_t is given by

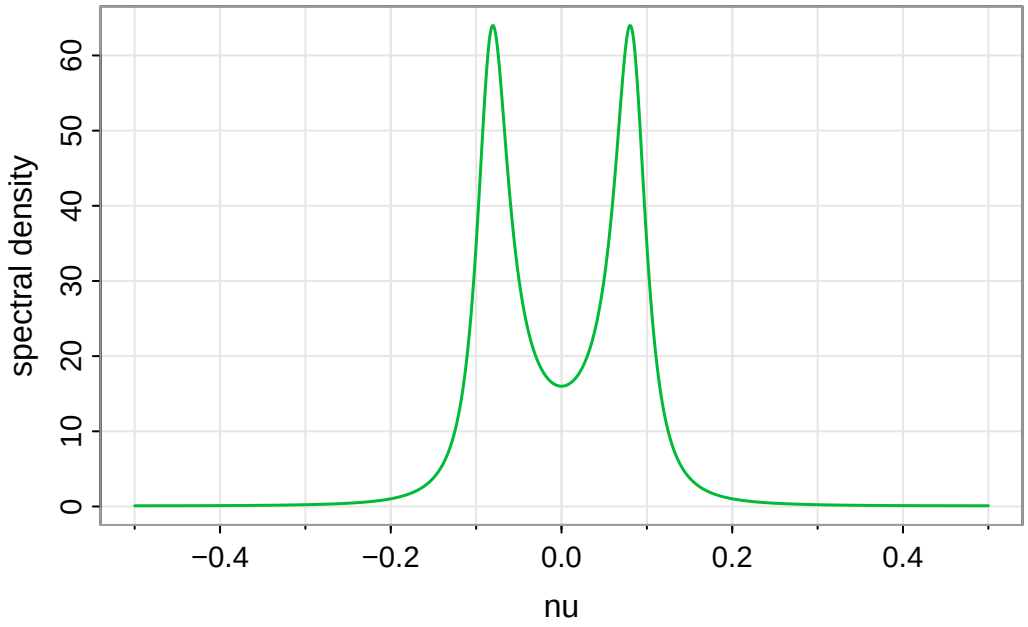
$$S(\nu) = \frac{\sigma^2}{|\phi(e^{-2\pi \mathrm{i}\nu})|^2}.$$

Recall that a stationary AR(p) model has roots of $\phi(z)$ outside of the unit circle. Here, in the denominator, we are traversing around the unit circle, so the spectral density will remain finite for stationary processes.

5.2.1.3.1 Example: AR(2)

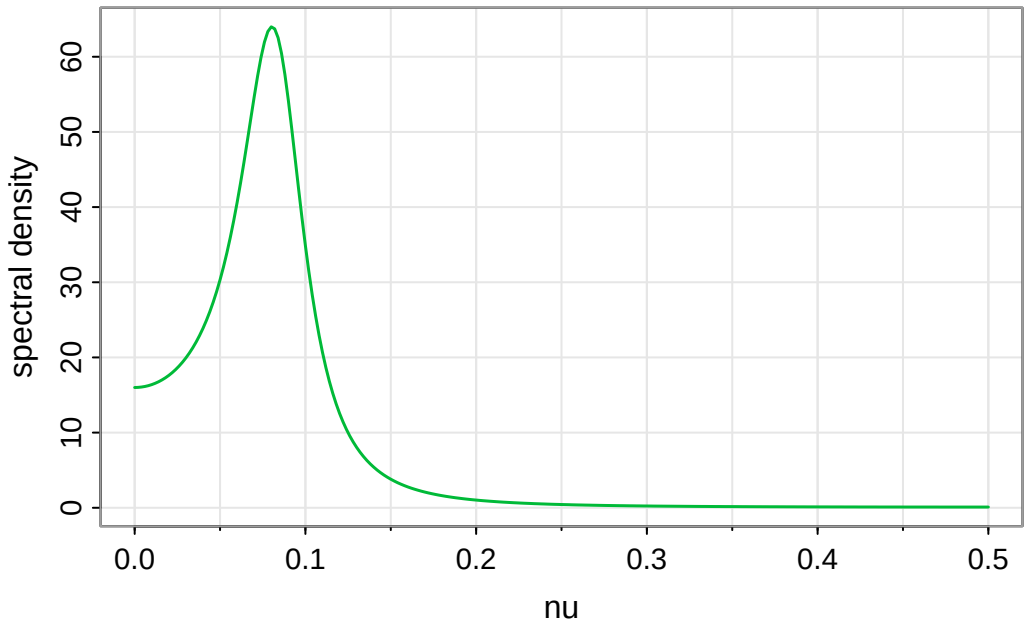
Consider an AR(2) model with $\phi_1 = 3/2,\ \phi_2 = -3/4,\ \sigma = 1$. We can plot the spectral density as follows.

```
library(astsa)
nu = seq(-0.5, 0.5, 0.001)
specd = function(nu) {
  z = complex(1, cos(2*pi*nu), -sin(2*pi*nu))
  phi = 1 - 1.5*z + 0.75*z*z
  1/abs(phi)^2
  }
spec = sapply(nu, specd)
tsplot(ts(spec, start=-0.5, deltat=0.001),
  col=3, lwd=1.5, xlab="nu", ylab="spectral density")
```



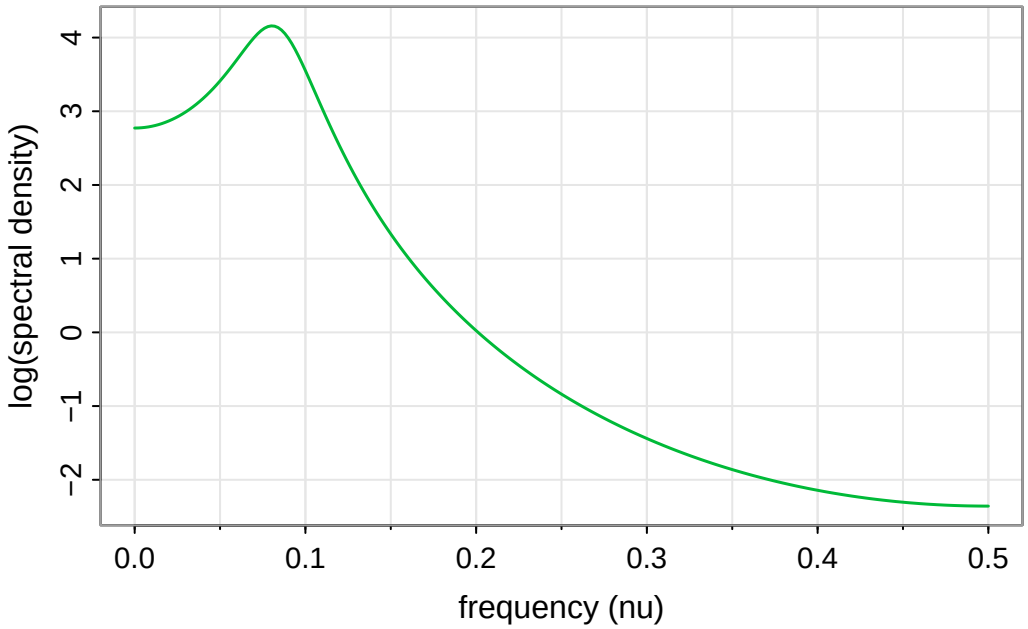
However, since the function is even, we typically only plot the function on $[0, \frac{1}{2}]$.

```
nu = seq(0, 0.5, 0.002)
spec = sapply(nu, specd)
tsplot(ts(spec, start=0, deltat=0.002),
  col=3, lwd=1.5, xlab="nu", ylab="spectral density")
```



It is quite common to see spectral densities plotted on a half-log scale.

```
tsplot(ts(log(spec), start=0, deltat=0.002),
  col=3, lwd=1.5, xlab="frequency (nu)", ylab="log(spectral density)")
```



Either way, we see that there is a single strong peak in this spectral density, occurring at around $\nu = 0.08$. In fact, we expect this, since we've looked at this AR(2) model previously.

```
peak = which.max(spec)*0.002
peak
```

```
[1] 0.082
```

```
1/peak
```

```
[1] 12.19512
```

So this model has oscillations with period around 12. This all makes sense, since the spectral density will be largest at points on the unit circle that are closest to the roots of the characteristic polynomial. But this will happen when the argument of the point of the unit circle matches that of a root.

5.2.1.4 ARMA(p,q)

A very similar argument to that used for AR(p) models confirms that the spectral density of an ARMA(p,q) process is given by

$$S(\nu) = \sigma^2 \frac{|\theta(e^{-2\pi i\nu})|^2}{|\phi(e^{-2\pi i\nu})|^2}.$$

Note that when calculating spectral densities by hand it can be useful to write this in the form

$$S(\nu) = \sigma^2 \frac{\theta(e^{2\pi i\nu})\theta(e^{-2\pi i\nu})}{\phi(e^{2\pi i\nu})\phi(e^{-2\pi i\nu})},$$

and to simplify the numerator and denominator separately, recognising the resulting real-valued trig functions that must arise, since both the numerator and denominator are real.

5.3 Finite time series

Of course, in practice we do not work with time series of infinite length. We instead work with time series of finite length, n. In the context of spectral analysis, it is more convenient to work with zero-based indexing, so we write our time series as

$$x_0, x_1, \ldots, x_{n-1}.$$

Since we only have n degrees of freedom, we do not need a continuous function $\hat{x}(\nu)$ in order to fix them. Since everything is linear, knowing $\hat{x}(\nu)$ at just n points would be sufficient, since that would give us n linear equations in n unknowns. Knowing it at n equally spaced points around the unit circle is natural, and turns out to give nice orthogonality properties that make everything very neat. This is the idea behind the discrete Fourier transform (DFT).

5.3.1 The discrete Fourier transform

The (forward) DFT takes the form

$$\hat{x}_k = \sum_{t=0}^{n-1} x_t e^{-2\pi ikt/n}, \quad k = 0, 1, \ldots, n-1.$$

It can be inverted with

$$x_t = \frac{1}{n} \sum_{k=0}^{n-1} \hat{x}_k e^{2\pi ikt/n}, \quad t = 0, 1, \ldots, n-1.$$

For the DTFT, the integral traversed the unit circle clockwise starting from minus one. The DFT moves around the unit circle clockwise starting from one, but this is just a commonly used convention, and doesn't change anything fundamental about the transform. These transforms are simple finite linear sums mapping back and forth between n values in the time and frequency domains, and hence are very amenable to computer implementation.

To understand how it works, it is helpful to define $\omega = e^{2\pi \mathrm{i}/n}$, the nth root of unity, with $w^n = 1$ and $\overline{\omega^k} = \omega^{-k} = \omega^{n-k}$. We can then write the transforms slightly more neatly as

$$x_t = \frac{1}{n}\sum_{k=0}^{n-1} \hat{x}_k \omega^{kt} \quad \text{and} \quad \hat{x}_k = \sum_{t=0}^{n-1} x_t \omega^{-kt}.$$

That the inversion works follows from the nice orthogonality property,

$$\sum_{t=0}^{n-1} \omega^{jt}\omega^{-kt} = \sum_{t=0}^{n-1} \omega^{(j-k)t} = n\,\delta_{jk},$$

which follows from the fact that the roots of unity sum to zero. If we start from the definition of x_t and sub in for the definition of $\hat{x}_k$ we find

$$\begin{aligned}
\frac{1}{n}\sum_{k=0}^{n-1} \hat{x}_k \omega^{kt} &= \frac{1}{n}\sum_{k=0}^{n-1}\sum_{s=0}^{n-1} x_s \omega^{-ks}\omega^{kt} \\
&= \frac{1}{n}\sum_{s=0}^{n-1} x_s \sum_{k=0}^{n-1} \omega^{-ks}\omega^{kt} \\
&= \frac{1}{n}\sum_{s=0}^{n-1} x_s n \delta_{st} \\
&= x_t,
\end{aligned}$$

as required. The factor of n in the inverse is a bit annoying, but has to go somewhere. The convention that we have adopted is the most common, but there are others.

The finite, discrete and linear nature of the DFT (and its inverse) make it extremely suitable for computer implementation. Naive implementation would involve $\mathcal{O}(n)$ operations to compute each coefficient, and hence a full transform would involve $\mathcal{O}(n^2)$ operations. However, it turns out that a divide-and-conquer approach can be used to reduce this to $\mathcal{O}(n \log n)$ operations. The algorithm that implements the DFT with reduced computational complexity is known as the fast Fourier transform (FFT). For large time series, this difference in complexity is transformational, and is one of the reasons that the FFT (and the *very* closely related discrete cosine transform, DCT) now underpin much of modern life.

5.3.1.1 The FFT in R

R has a built-in `fft` function that can efficiently compute a DFT. It can also invert the DFT, but it returns an unnormalised inverse (without the factor of n), so it is convenient to define our own function to invert the DFT.

```
ifft = function(x) fft(x, inverse=TRUE) / length(x)
```

With this iDFT function in place, we can test it to make sure that it correctly inverts a DFT.

```
x = 1:5
x
```

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

```
fx = fft(x)
fx
```

```
[1] 15.0+0.0000000i -2.5+3.4409548i -2.5+0.8122992i -2.5-0.8122992i
[5] -2.5-3.4409548i
```

```
ifft(fx)
```

```
[1] 1+0i 2+0i 3+0i 4+0i 5+0i
```

```
Re(ifft(fx))
```

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

5.3.1.2 Matrix formulation

It can also be instructive to consider the DFT in matrix-vector form. Starting with the inverse DFT, we can write this as

$$\mathbf{x} = \frac{1}{n}\mathsf{F}_n\hat{\mathbf{x}},$$

where F_n is the $n \times n$ Fourier matrix with (i, j)th element w^{ij}. Clearly F_n is symmetric ($\mathsf{F}_n^\top = \mathsf{F}_n$). Now notice that the forward DFT is just

$$\hat{\mathbf{x}} = \bar{\mathsf{F}}_n\mathbf{x}$$

Further, our orthogonality property implies that $\mathsf{F}_n\bar{\mathsf{F}}_n = n\mathbb{I}$ (so F_n is unitary, modulo a factor of n), and $\mathsf{F}_n^{-1} = \bar{\mathsf{F}}_n/n$. So inversion is now clear, since if we start with the definition of $\mathbf{x}$ and substitute in for $\hat{\mathbf{x}}$, we get

$$\frac{1}{n}\mathsf{F}_n\hat{\mathbf{x}} = \frac{1}{n}\mathsf{F}_n\bar{\mathsf{F}}_n\mathbf{x} = \frac{1}{n}n\mathbb{I}\mathbf{x} = \mathbf{x},$$

as required.

5.3.2 The periodogram

For a time series $x_0, x_1, \ldots, x_{n-1}$ with DFT $\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1}$, the periodogram is just the sequence of non-negative real numbers

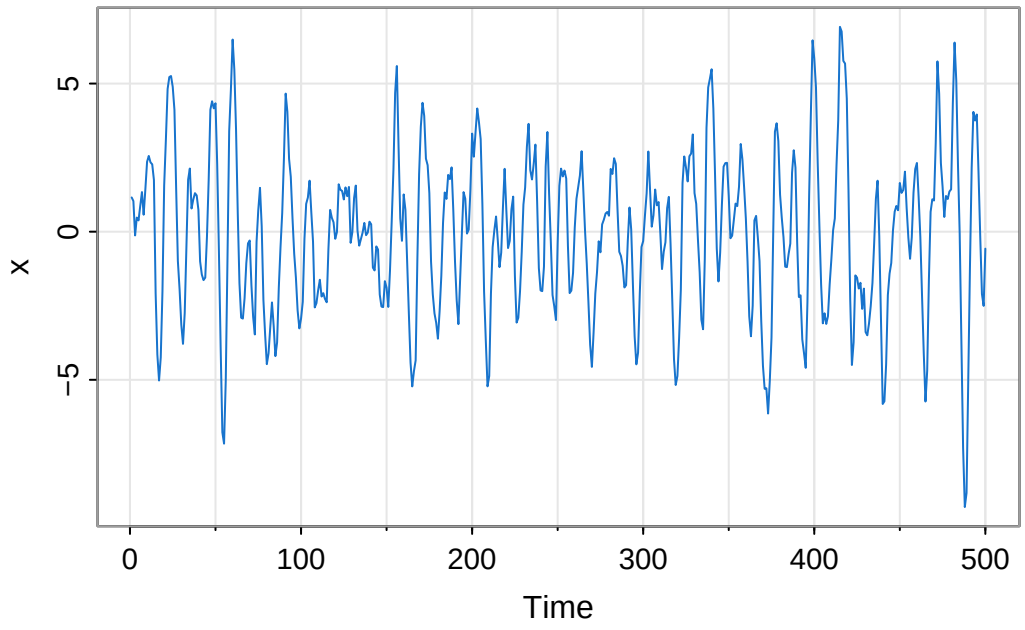
$$I_k = |\hat{x}_k|^2, \quad k = 0, 1, \ldots, n-1.$$

They represent a (very bad) estimate of the spectral density function.

5.3.2.1 Example: AR(2)

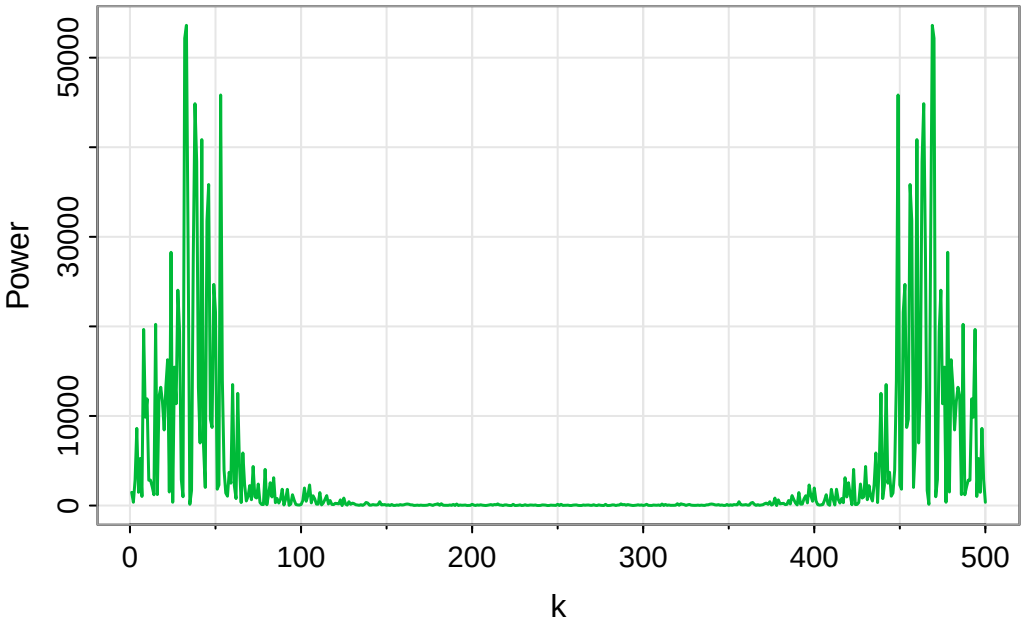
Let's see how this works in the context of our favourite AR(2) model. First, simulate some data from the model.

```
set.seed(42)
n = 500
x = arima.sim(n=n, list(ar = c(1.5, -0.75)))
tsplot(x, col=4)
```



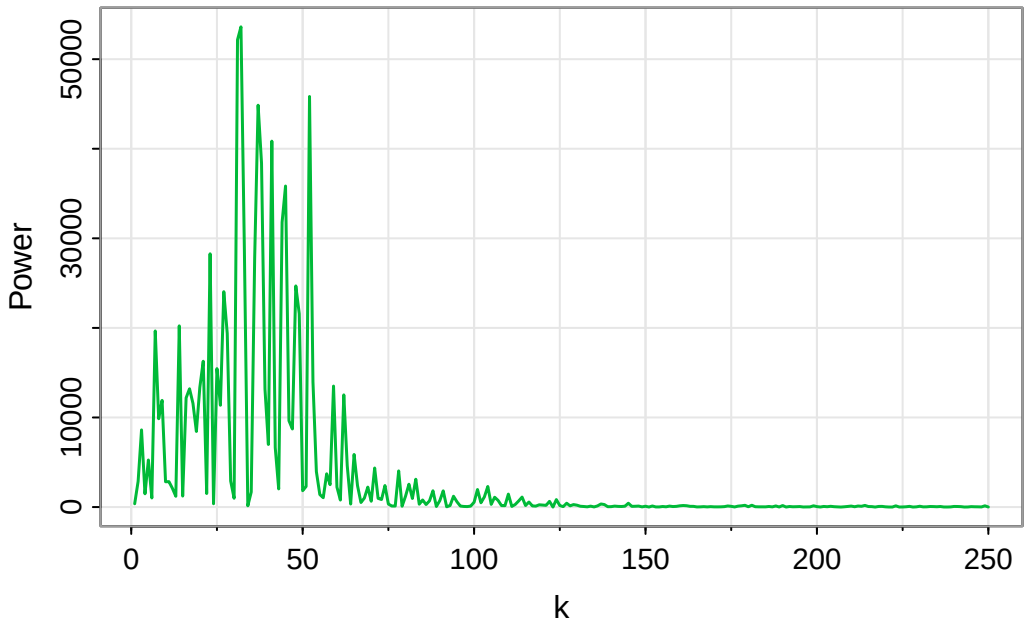
We can plot a very basic periodogram as follows.

```
tsplot(abs(fft(x))^2,
  col=3, lwd=1.5, xlab="k", ylab="Power")
```



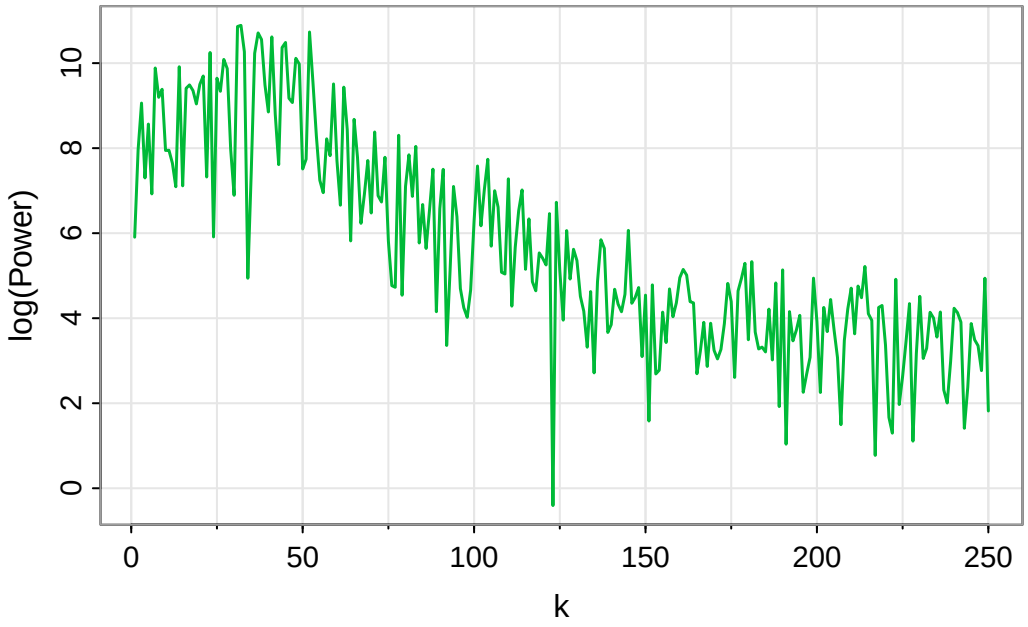
Since the second half of the periodogram is just a mirror-image of the first, typically only the first half is plotted.

```
tsplot(abs(fft(x)[2:(n/2+1)])^2,
  col=3, lwd=1.5, xlab="k", ylab="Power")
```



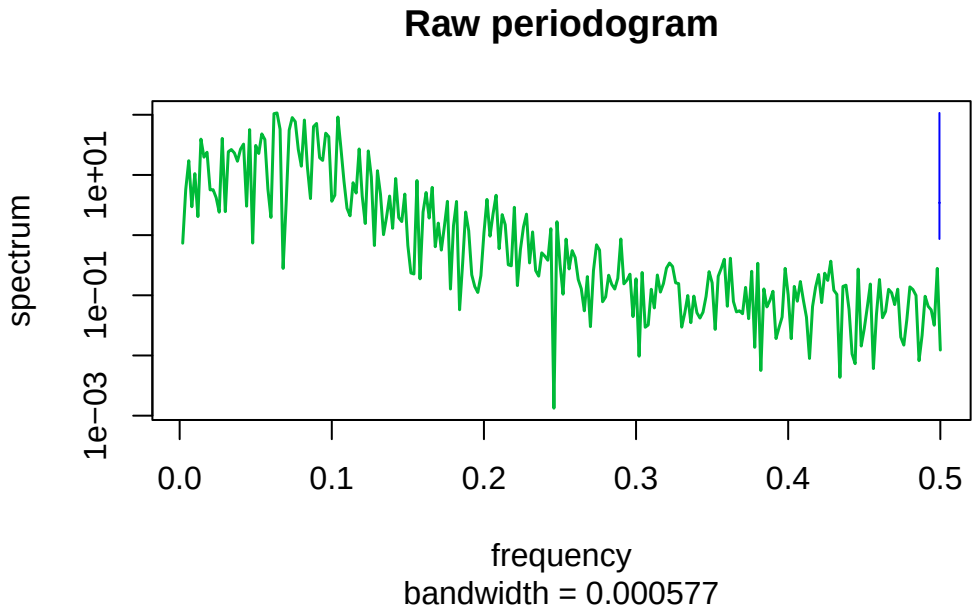
The periodogram is typically plotted on a half-log scale.

```
tsplot(log(abs(fft(x)[2:(n/2+1)])^2),
  col=3, lwd=1.5, xlab="k", ylab="log(Power)")
```

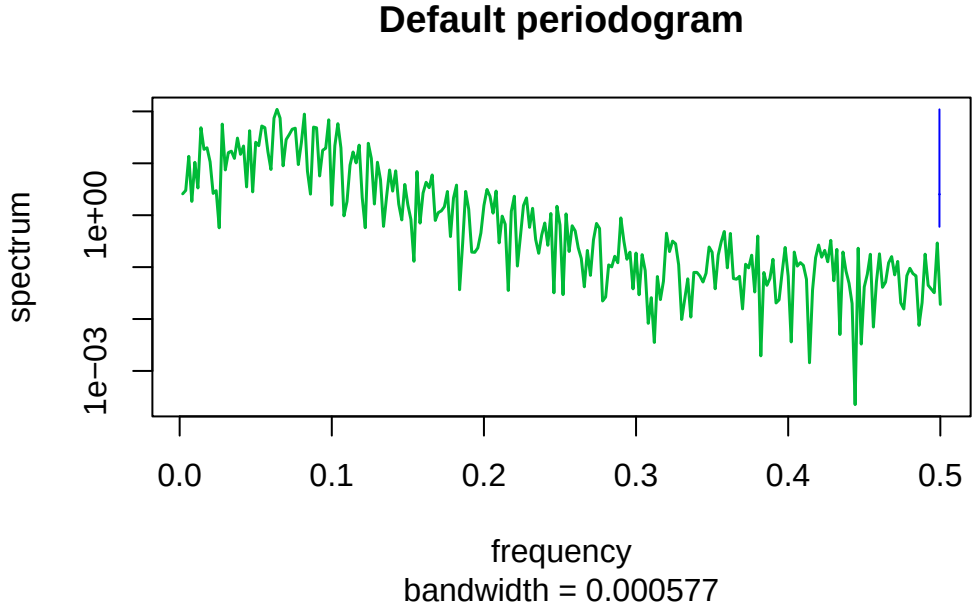


This is now exactly like R's built-in periodogram function, `spec.pgram`, with all "tweaks" disabled. But the default periodogram does a little bit of pre-processing.

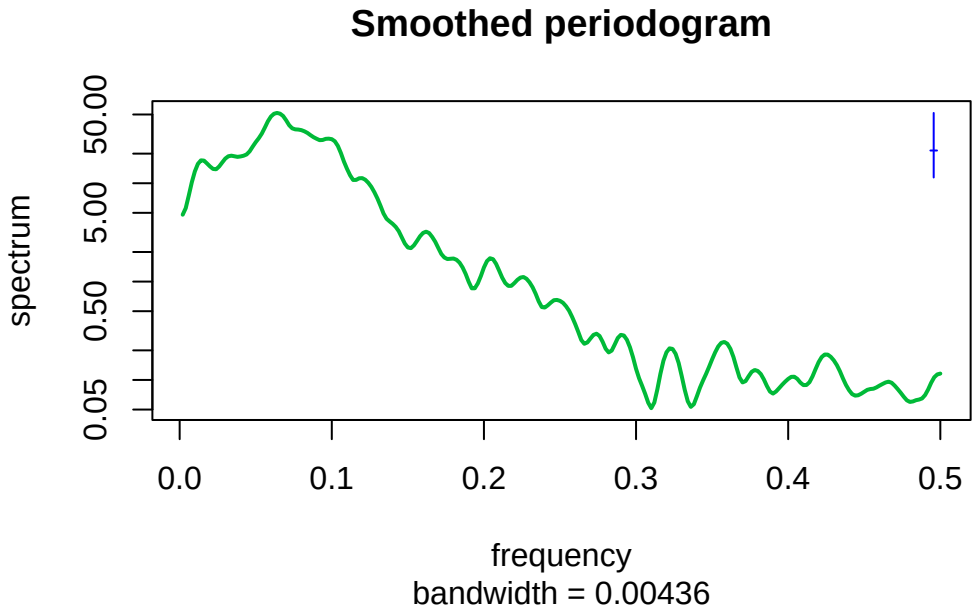
```
spec.pgram(x, taper=0, detrend=FALSE,
  col=3, lwd=1.5, main="Raw periodogram")
```




```
spec.pgram(x, col=3, lwd=1.5,
  main="Default periodogram")
```




```
spectrum(x, spans=c(5, 7),
  col=3, lwd=2, main="Smoothed periodogram")
```



In practice, various smoothing methods are used to get better a better estimate of the spectral density. This is the subject of spectral density estimation, but the details of this do not concern us. In the final plot, a peak at a frequency of around 0.08 can be seen, as we would hope.

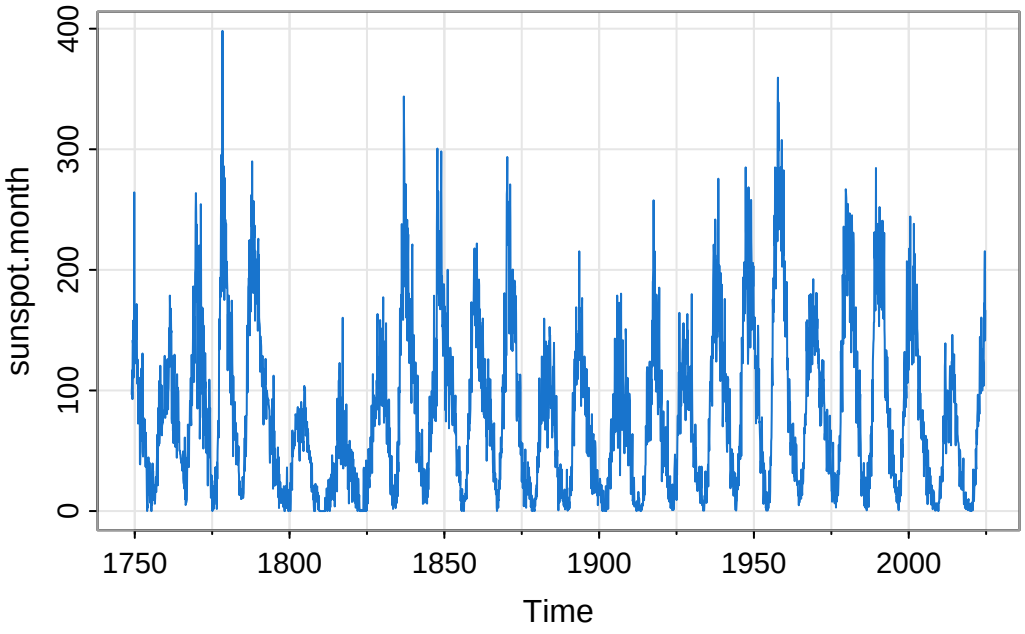
5.3.3 Smoothing with the DFT

The potential applications of the DFT (and of spectral analysis, more generally) are many and varied, but we don't have time to explore them in detail here. But essentially, the DFT decomposes your original signal into different frequency components. These different frequency components can be separated, manipulated and combined in various ways. One obvious application is smoothing, where high frequency components are simply wiped out. This is best illustrated by example.

5.3.3.1 Example: monthly sunspot data

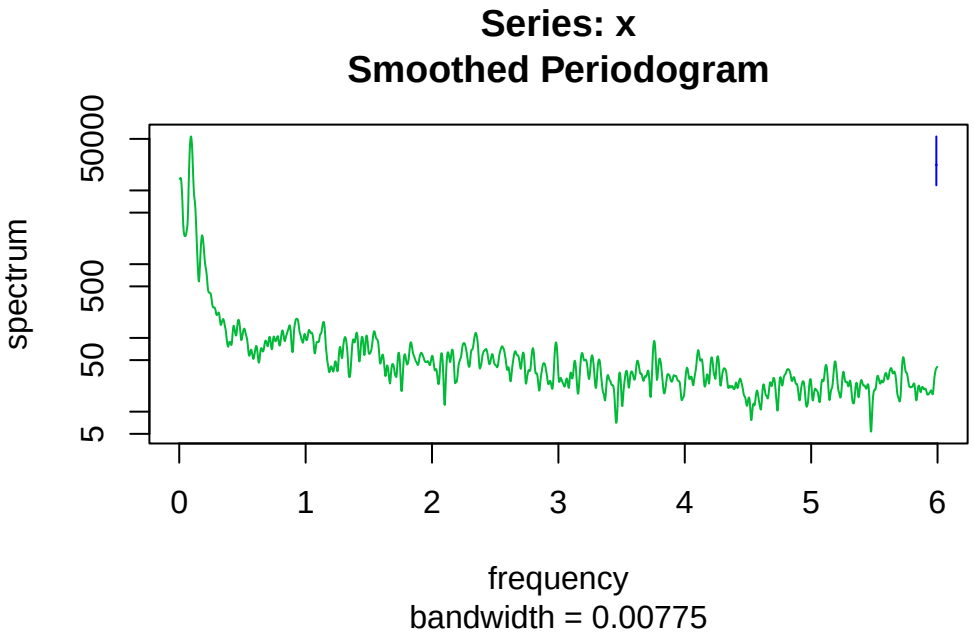
We previously looked at yearly sunspot data, and fitted an AR(2) model to it. There is also a monthly sunspot dataset that is in many ways more interesting.

```
tsplot(sunspot.month, col=4)
```



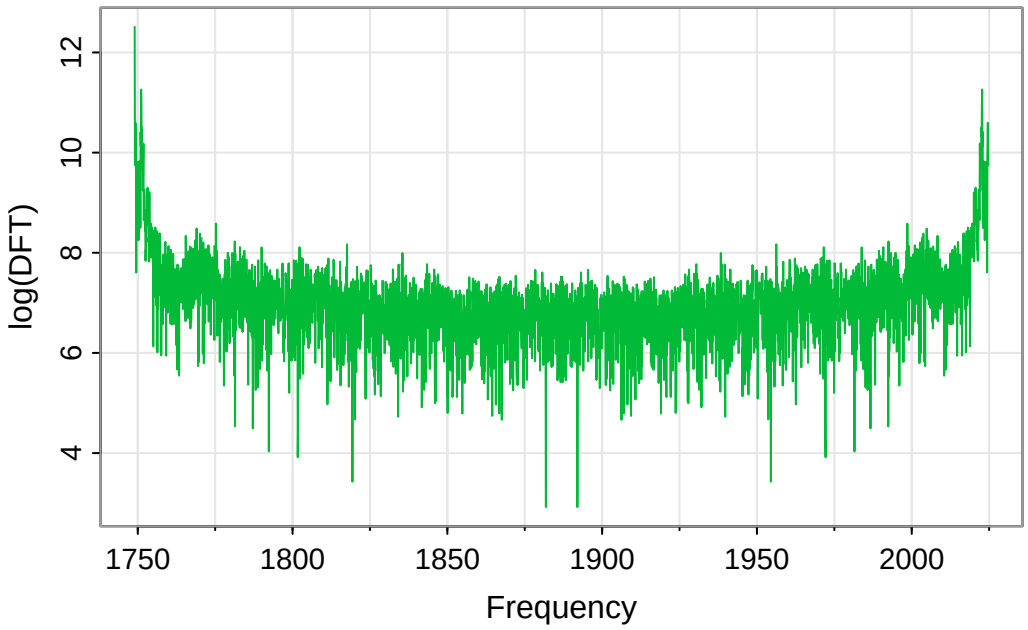
We can see strong oscillations with period just over a decade. However, fitting an AR(2) to this data fails miserably due to the presence of strong high frequency oscillations. A quick look at the periodogram

```
spectrum(sunspot.month, spans=c(5, 7), col=3)
```



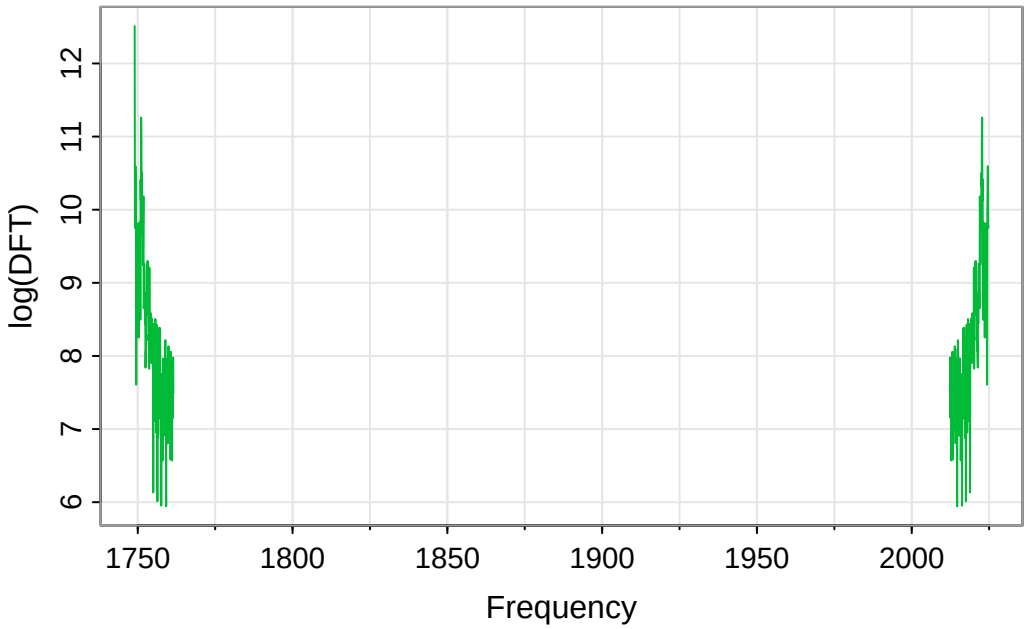
reveals that although there is a strong peak at low frequency oscillation, there is also a lot of higher frequency stuff going on. So, let's look more carefully at the DFT of the data.

```
ft = fft(sunspot.month)
tsplot(log(abs(ft)), col=3,
  xlab="Frequency", ylab="log(DFT)")
```



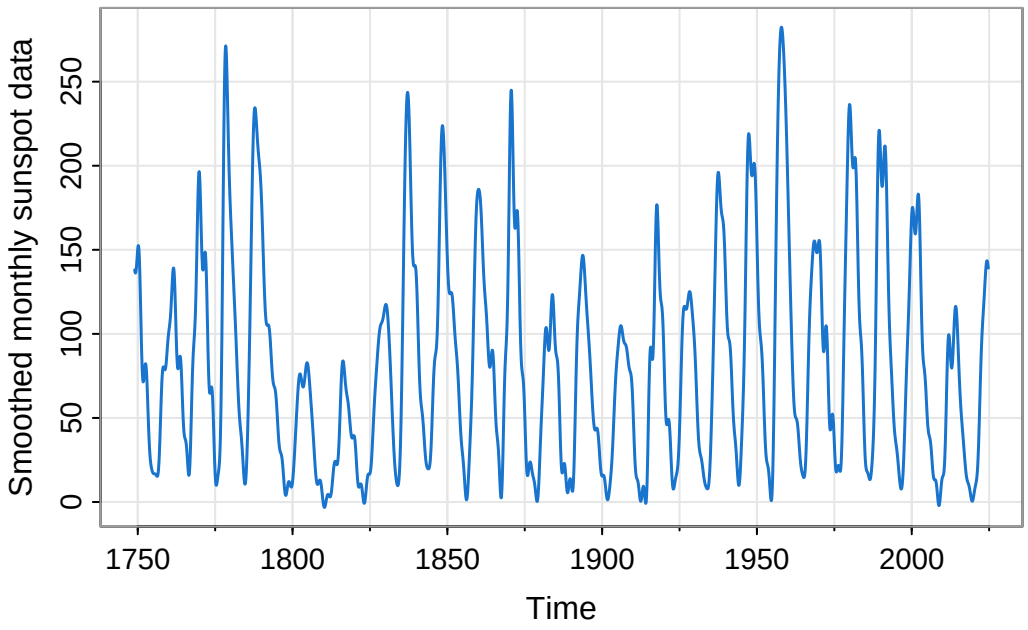
The peaks close to the two ends correspond to the low frequency oscillations that we are interested in. Everything in the middle is high frequency noise that we are not interested in, so let's just get rid of it. Suppose that we want to keep the first (and last) 150 frequency components. We can zero out the rest as follows.

```
keep = 150
ft[(keep+2):(length(ft)-keep)] = 0
tsplot(log(abs(ft)), col=3,
  xlab="Frequency", ylab="log(DFT)")
```



We can now switch back to the time domain to see the data with the high frequency components removed.

```
sm = Re(ifft(ft))
tsplot(sm, col=4, lwd=1.5,
  ylab="Smoothed monthly sunspot data")
```



This is quite a nice approach to smoothing the data.

6 Hidden Markov models (HMMs)

6.1 Introduction

So far we have been assuming that we perfectly observe our time series model. That is, if we have a model for X_t, we observe x_t. However, we are very often in a situation where we have a good model for a (Markov) process X_t, but we are only able to observe some partial or noisy aspects. We call this observation process Y_t (and to be useful, this observation must depend on X_t in some way), and hence we only ever observe y_t. The X_t process remains "hidden", but we are often able to learn a lot about it from the y_t. This is the idea behind state space modelling. Very often, our hidden process will be a linear Gaussian process like those we have mainly been considering. We will study this case in Chapter 7. It is slightly simpler to start with the case where the hidden process is a finite-state Markov chain. In this case the combined model is referred to as a hidden Markov model.

Our hidden process $X_0, X_1, \ldots, X_n$ is a Markov chain with state space $\mathcal{X} = \{1, 2, \ldots, p\}$ for some known integer $p > 1$. At time t the probability that the chain is in each state is described by a (row) vector

$$\boldsymbol{\pi}(t) = (\mathbb{P}[X_t = 1], \mathbb{P}[X_t = 2], \ldots, \mathbb{P}[X_t = p]).$$

The chain is initialised with the vector $\boldsymbol{\pi}(0)$. The dynamics of the chain are governed by a *transition matrix* P, and we use the (usual, but bad, Western) convention that the (i, j)th element corresponds to $\mathbb{P}[X_{t+1} = j | X_t = i]$. The rows of P must then sum to one, and P is known as a (right) stochastic matrix. The law of total probability leads to the update rule

$$\boldsymbol{\pi}(t+1) = \boldsymbol{\pi}(t)\mathsf{P}. \tag{6.1}$$

It is clear from Equation 6.1 that a distribution $\boldsymbol{\pi}$ satisfying

$$\boldsymbol{\pi} = \boldsymbol{\pi}\mathsf{P}$$

will be *stationary*. Choosing $\boldsymbol{\pi}(0) = \boldsymbol{\pi}$ will ensure that the Markov chain is stationary. This is a common choice, but not required. A discrete uniform distribution on the initial states is also commonly adopted, and can sometimes be simpler.

The observation process is $Y_1, Y_2, \ldots, Y_n$, and will ultimately lead to n observations $y_1, y_2, \ldots, y_n$. Y_t represents some kind of noisy or partial observation of X_t, and so, crucially, is conditionally independent of all other X_s and Y_s ($s \neq t$) *given* X_t. In other words, the distribution of Y_t will depend (directly) only on X_t. The state space, $\mathcal{Y}$, is largely irrelevant to the analysis. It could be discrete (finite or countable) or continuous. We just need to specify a probability (mass or density) function for the observation which can then be used as a likelihood. We define

$$l_i(y) = \mathbb{P}[Y_t = y | X_t = i],$$

but note that this can be replaced with a probability density in the continuous case. Then $\mathbf{l}(y)$ is a (row) p-vector of probabilities (or densities) associated with the likelihood of observing some $y \in \mathcal{Y}$.

For much of this chapter we consider the case where $\boldsymbol{\pi}(0)$, P and $\mathbf{l}(\cdot)$ are known (though we will consider parameter estimation briefly, later), and we want to use the observations $y_1, y_2, \ldots, y_n$ to tell us about the hidden Markov chain $(X_0), X_1, X_2, \ldots, X_n$.

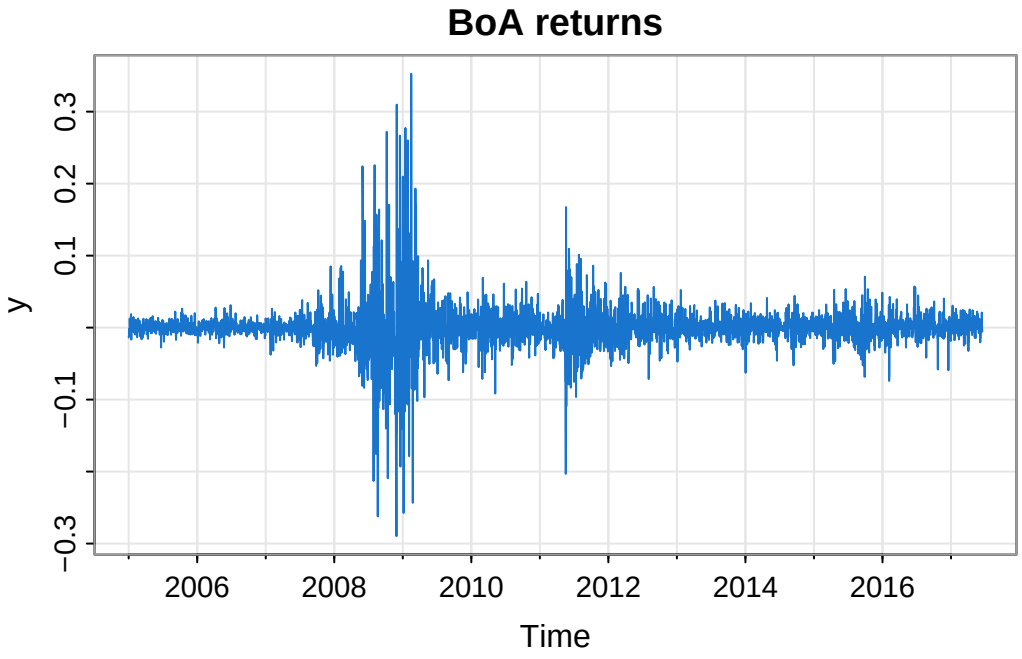
6.1.1 Example

As our running example for this chapter, we will look at a financial time series, specifically the daily returns of the Bank of America from 2005 to 2017.

```
library(astsa)
y = BCJ[,"boa"]
length(y)
```

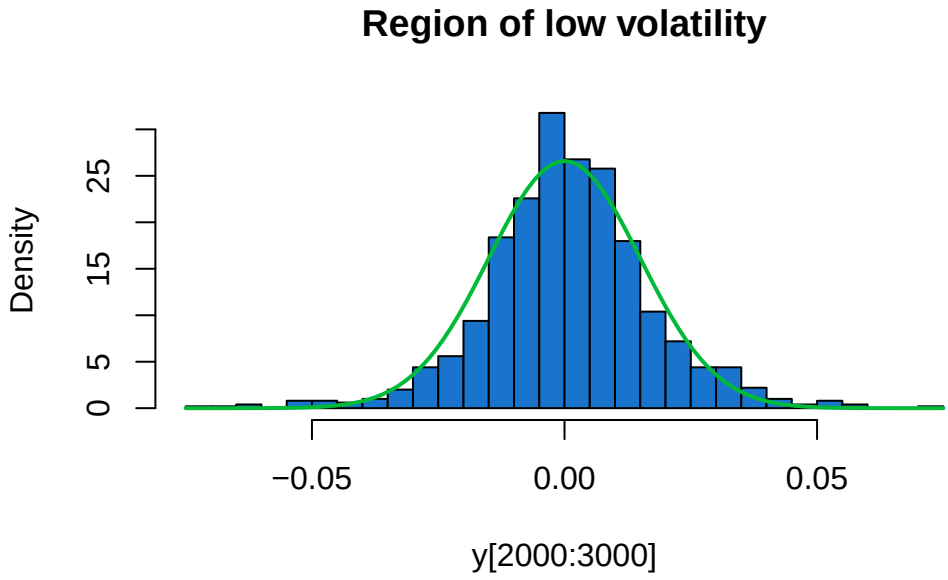
```
[1] 3243
```

```
tsplot(y, col=4, main="BoA returns")
```

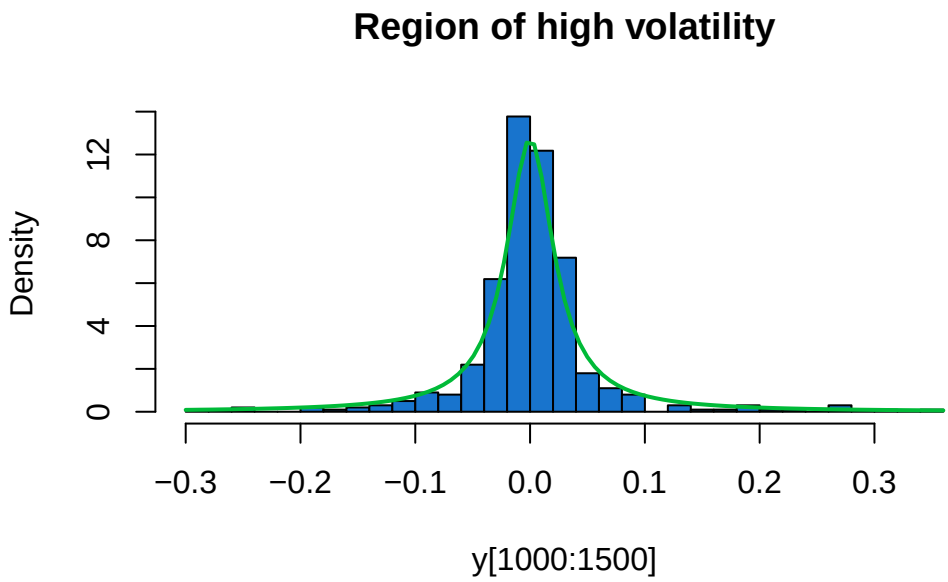


We see that the returns have mean close to zero, as we would expect, but that there are at least two periods of higher volatility - one in 2008/09, and another in 2011. We will use a HMM to automatically segment our time series into regions of high and low volatility.

```
hist(y[2000:3000], 30, col=4, freq=FALSE,
  main="Region of low volatility")
curve(dnorm(x, 0, 0.015), add=TRUE, col=3, lwd=2)
```




```
hist(y[1000:1500], 30, col=4, freq=FALSE,
  main="Region of high volatility")
curve(dcauchy(x, 0, 0.025), add=TRUE, col=3, lwd=2)
```



Very informal analysis suggests that we could reasonably model the periods of low volatility as mean zero Gaussian with standard deviation 0.015, and regions of high volatility as Cauchy with scale parameter 0.025.

For our hidden Markov chain, we will suppose that periods of low volatility ($i = 1$) last for 1,000 days on average, and that periods of high volatility ($i = 2$) last for 200 days on average, leading to the transition matrix,

$$\mathsf{P} = \begin{pmatrix} 0.999 & 0.001 \\ 0.005 & 0.995 \end{pmatrix}.$$

If we further assume $\boldsymbol{\pi}(0) = (0.5, 0.5)$, our model is fully specified.

6.2 Filtering

In the context of state space modelling, the *filtering problem* is the problem of (sequentially) computing, at each time t, the distribution of the hidden state, X_t, given all data up to time t, $\mathbf{y}_{1:t}$. That is, we want to compute

$$f_i(t) = \mathbb{P}[X_t = i | \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}], \quad \forall i \in \mathcal{X}, t = 0, 1, \ldots, n,$$

so $\mathbf{f}(t)$ is the (row) p-vector of filtered probabilities at time t. We could imagine computing this *on-line*, as each new observation becomes available. We know that $\mathbf{f}(0) = \boldsymbol{\pi}_0$, so assume that we know $\mathbf{f}(t-1)$ for some $t > 0$ and want to compute $\mathbf{f}(t)$. It is instructive to do this in two steps.

Predict step

We begin by computing

$$\tilde{f}_i(t) = \mathbb{P}[X_t = i | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}], \quad i = 1, 2, \ldots, p.$$

The law of total probability along with conditional independence gives

$$\begin{aligned}
\tilde{f}_i(t) &= \mathbb{P}[X_t = i | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}] \\
&= \sum_{j=1}^{p} \mathbb{P}[X_t = i | X_{t-1} = j, \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}] \mathbb{P}[X_{t-1} = j | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}] \\
&= \sum_{j=1}^{p} \mathbb{P}[X_t = i | X_{t-1} = j] \mathbb{P}[X_{t-1} = j | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}] \\
&= \sum_{j=1}^{p} P_{ji} f_j(t-1).
\end{aligned}$$

In other words,

$$\tilde{\mathbf{f}}(t) = \mathbf{f}(t-1)\mathsf{P}.$$

This is obviously reminiscent of our fundamental Markov chain update property, Equation 6.1, but does rely on the conditional independence of X_t and $\mathbf{Y}_{1:(t-1)}$ given X_{t-1}.

Update step

Now we have pushed our probabilities forward in time, we can condition on our new observation.

$$\begin{aligned}
f_i(t) &= \mathbb{P}[X_t = i | \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}] \\
&= \mathbb{P}[X_t = i | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}, Y_t = y_t] \\
&= \frac{\mathbb{P}[X_t = i | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}] \mathbb{P}[Y_t = y_t | X_t = i, \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}]}{\mathbb{P}[Y_t = y_t | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}]} \\
&= \frac{\mathbb{P}[X_t = i | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}] \mathbb{P}[Y_t = y_t | X_t = i]}{\mathbb{P}[Y_t = y_t | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}]} \\
&= \frac{\tilde{f}_i(t) l_i(y_t)}{\mathbb{P}[Y_t = y_t | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}]}.
\end{aligned}$$

We could therefore write

$$\mathbf{f}(t) = \frac{\tilde{\mathbf{f}}(t) \circ \mathbf{l}(y_t)}{\mathbb{P}[Y_t = y_t | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}]},$$

where $\circ$ is the Hadamard (element-wise) product, and the denominator is a scalar normalising constant, and so we could just write

$$\mathbf{f}(t) \propto \tilde{\mathbf{f}}(t) \circ \mathbf{l}(y_t),$$

which makes it clearer that we are just re-weighting the results of the predict step according to the likelihood of the observation. We can just compute the RHS and then normalise to get the LHS. Note that we may nevertheless want to keep track of the normalising constant (explained in the next section). If we prefer, we can combine the predict and update steps as

$$\mathbf{f}(t) \propto [\mathbf{f}(t-1)\mathsf{P}] \circ \mathbf{l}(y_t).$$

Starting this filtering process off at $t = 1$ and then running it forward to $t = n$ is the forward part of the forward-backward algorithm.

6.2.1 Example

We can implement the filter by creating a function that advances the algorithm by one step.

```
hmmFilter = function(P, l)
  function(f, y) {
    fNew = (f %*% P) * l(y)
    fNew / sum(fNew)
  }
```

We can use this to create the advancement function for our running example as follows.

```
advance = hmmFilter(
  matrix(c(0.999, 0.001, 0.005, 0.995), ncol=2, byrow=TRUE),
  function(y) c(dnorm(y, 0, 0.015), dcauchy(y, 0, 0.025))
  )
```

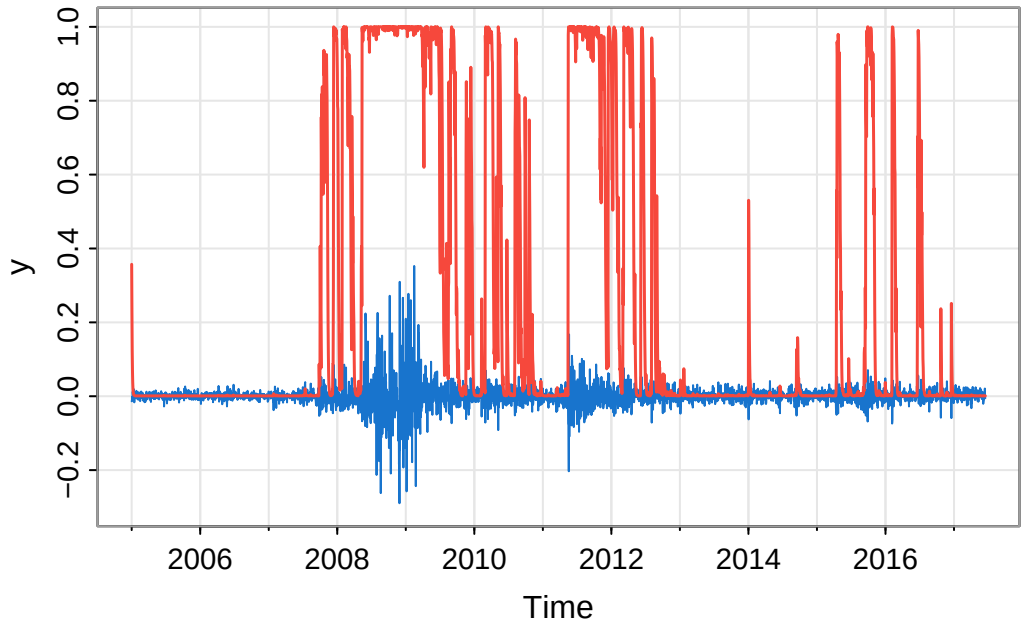
So `advance` is now a function that takes the current set of filtered probabilities and an observation and returns the next set of (normalised) filtered probabilities. We can apply this function sequentially to our time series using the `Reduce` function, which implements a functional fold. If we just want the final set of filtered probabilities, we can call it as follows.

```
Reduce(advance, y, c(0.5, 0.5))
```

```
          [,1]         [,2]
[1,] 0.9989384 0.001061576
```

This tells us that we end at a period of low volatility (with very high probability). But more likely, we will want the full set of filtered probabilities, and to plot them over the data in some way.

```
fpList = Reduce(advance, y, c(0.5, 0.5), acc=TRUE)
fpMat = sapply(fpList, cbind)
fp2Ts = ts(fpMat[2, -1], start=start(y), freq=frequency(y))
tsplot(y, col=4, ylim=c(-0.3, 1))
lines(fp2Ts, col=2, lwd=1.5)
```



Here we show the filtered probability of being in the high volatility state.

6.3 Marginal likelihood

For many reasons (including parameter estimation, to be discussed later), it is often useful to be able to compute the marginal probability (density) of the data, $\mathbb{P}[\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}]$, not conditioned on the (unknown) hidden states. If we factorise this as

$$\mathbb{P}[\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] = \prod_{t=1}^{n} \mathbb{P}[Y_t = y_t | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}],$$

we see that the required terms correspond precisely to the normalising constants calculated during filtering. So we can compute the marginal likelihood easily as a by-product of filtering with essentially no additional computation.

In practice, for reasons of numerical stability (in particular, avoiding numerical underflow), we compute the log of the marginal likelihood as

$$\log \mathbb{P}[\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] = \sum_{t=1}^{n} \log \mathbb{P}[Y_t = y_t | \mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}].$$

6.3.1 Example

We modify our previous function to update the marginal likelihood of the data so far, in addition to the filtered probabilities.

```
hmmFilterML = function(P, l)
  function(fl, y) {
    fNew = (fl$f %*% P) * l(y)
    ml = sum(fNew)
    list(f=fNew/ml, ll=(fl$ll + log(ml)))
  }
```

```
advance = hmmFilterML(
  matrix(c(0.999, 0.001, 0.005, 0.995), ncol=2, byrow=TRUE),
  function(y) c(dnorm(y, 0, 0.015), dcauchy(y, 0, 0.025))
  )

Reduce(advance, y, list(f=c(0.5, 0.5), ll=0))
```

```
$f
           [,1]         [,2]
[1,] 0.9989384 0.001061576

$ll
[1] 7971.837
```

This returns the final set of filtered probabilities, as before, but now also the marginal likelihood of the data.

6.4 Smoothing

Filtering is great in the on-line context, but off-line, with a static dataset of size n, we are probably more interested in the *smoothed* probabilities,

$$s_i(t) = \mathbb{P}[X_t = i | \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}], \quad i \in \mathcal{X},\ t = 0, 1, \ldots, n.$$

Now, from the forward filter, we already know the final

$$\mathbf{s}(n) = \mathbf{f}(n).$$

So now (for a backward recursion), suppose we already know $\mathbf{s}(t+1)$ for some $t < n$, and want to know $\mathbf{s}(t)$.

$$\begin{aligned}
s_i(t) &= \mathbb{P}[X_t = i | \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] \\
&= \sum_{j=1}^{p} \mathbb{P}[X_t = i, X_{t+1} = j | \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] \\
&= \sum_{j=1}^{p} \mathbb{P}[X_{t+1} = j | \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] \mathbb{P}[X_t = i | X_{t+1} = j, \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] \\
&= \sum_{j=1}^{p} s_j(t+1) \mathbb{P}[X_t = i | X_{t+1} = j, \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}] \\
&= \sum_{j=1}^{p} s_j(t+1) \frac{\mathbb{P}[X_t = i | \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}] \mathbb{P}[X_{t+1} = j | X_t = i, \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}]}{\mathbb{P}[X_{t+1} = j | \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}]} \\
&= \sum_{j=1}^{p} s_j(t+1) \frac{f_i(t) P_{ij}}{\tilde{f}_j(t+1)} \\
&= f_i(t) \sum_{j=1}^{p} \frac{s_j(t+1)}{\tilde{f}_j(t+1)} P_{ij}.
\end{aligned}$$

We could write this as

$$\begin{aligned}
\mathbf{s}(t) &= \mathbf{f}(t) \circ \left[\left(\mathbf{s}(t+1) \circ \tilde{\mathbf{f}}(t+1)^{-1} \right) \mathsf{P}^\top \right] \\
&= \mathbf{f}(t) \circ \left[\left(\mathbf{s}(t+1) \circ \{\mathbf{f}(t)\mathsf{P}\}^{-1} \right) \mathsf{P}^\top \right].
\end{aligned}$$

So, we can smooth by running backwards using the filtered probabilities from the forward pass.

6.4.1 Example

We can create a function to carry out one step of the backward pass as follows.

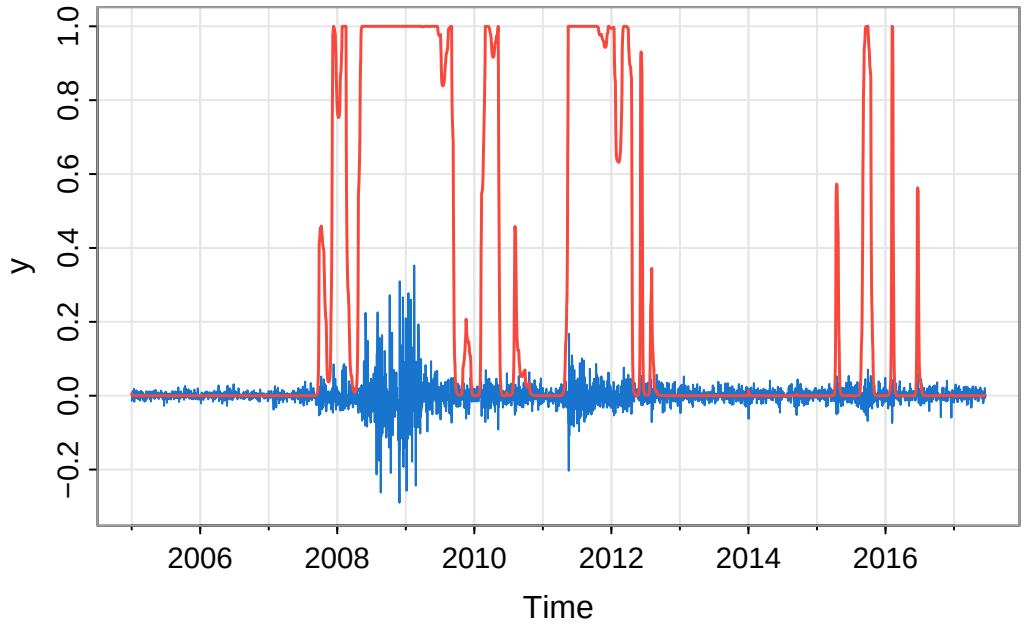
```
hmmSmoother = function(P)
  function(fp, sp) {
    fp * ((sp / (fp %*% P)) %*% t(P))
  }
```

We can then create a backward stepping function for our running example with:

```
backStep = hmmSmoother(
  matrix(c(0.999, 0.001, 0.005, 0.995), ncol=2, byrow=TRUE)
  )
```

We can then apply this to the list of filtered probabilities by requesting that `Reduce` folds from the *right* (ie. starts with the final element of the list and works backwards).

```
spList = Reduce(backStep, fpList, right=TRUE, acc=TRUE)
spMat = sapply(spList, cbind)
sp2Ts = ts(spMat[2, -1], start=start(y), freq=frequency(y))
tsplot(y, col=4, ylim=c(-0.3, 1))
lines(sp2Ts, col=2, lwd=1.5)
```



We see that the smoothed probabilities are, indeed, smoother than the filtered probabilities, since the whole time series is used for the marginal classification of each time point.

6.5 Sampling

The smoothed probabilities tell us marginally what is happening at each time point t, but don't tell us anything about the joint distribution of the hidden states. A good way to get insight into the joint distribution is to generate samples from it. This forms a useful part of many Monte Carlo algorithms for HMMs,

including MCMC methods for parameter estimation. So, we want to generate samples from the probability distribution

$$\mathbb{P}[\mathbf{X}_{1:n}|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}].$$

As with the other problems we have considered in this chapter, naive approaches won't work. Here, the state space that we are simulating on has size p^n, so we can't simply enumerate probabilities of possible trajectories. However, we can nevertheless generate exact samples from this distribution by first computing filtered probabilities with a forward sweep, as previously described, but now using a different backward pass.

It is convenient to factorise the joint distribution in a "backwards" manner as

$$\begin{aligned}\mathbb{P}[\mathbf{X}_{1:n} = \mathbf{x}_{1:n}|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] &= \prod_{t=1}^{n} \mathbb{P}[X_t = x_t|\mathbf{X}_{(t+1):n} = \mathbf{x}_{(t+1):n}, \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] \\ &= \left(\prod_{t=1}^{n-1} \mathbb{P}[X_t = x_t|X_{t+1} = x_{t+1}, \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}]\right) \mathbb{P}[X_n = x_n|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}].\end{aligned}$$

Now, we know the final term $\mathbb{P}[X_n|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] = \mathbf{f}(n)$, so we can sample from this to obtain x_n. So now suppose that we know x_{t+1} for some $t < n$ and that we want to sample from

$$\mathbb{P}[X_t|X_{t+1} = x_{t+1}, \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}].$$

We have

$$\begin{aligned}\mathbb{P}[X_t = i|X_{t+1} = x_{t+1}, \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}] &= \frac{\mathbb{P}[X_t = i|\mathbf{Y}_{1:t} = \mathbf{y}_{1:t}]\mathbb{P}[X_{t+1} = x_{t+1}|X_t = i, \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}]}{\mathbb{P}[X_{t+1} = x_{t+1}|\mathbf{Y}_{1:t} = \mathbf{y}_{1:t}]} \\ &\propto f_i(t)P_{i,x_{t+1}}.\end{aligned}$$

In other words, the probabilities are proportional to $\mathbf{f}(t) \circ \mathsf{P}^{(x_{t+1})}$, the relevant column of P. So, we proceed backwards from $t = n$ to $t = 1$, sampling x_t at each step.

6.5.1 Example

We can create a function to carry out one step of backward sampling as follows.

```
hmmSampler = function(P) {
  p = nrow(P)
  function(fp, x) {
    sample(1:p, 1, prob=fp*P[,x])
  }
}
```

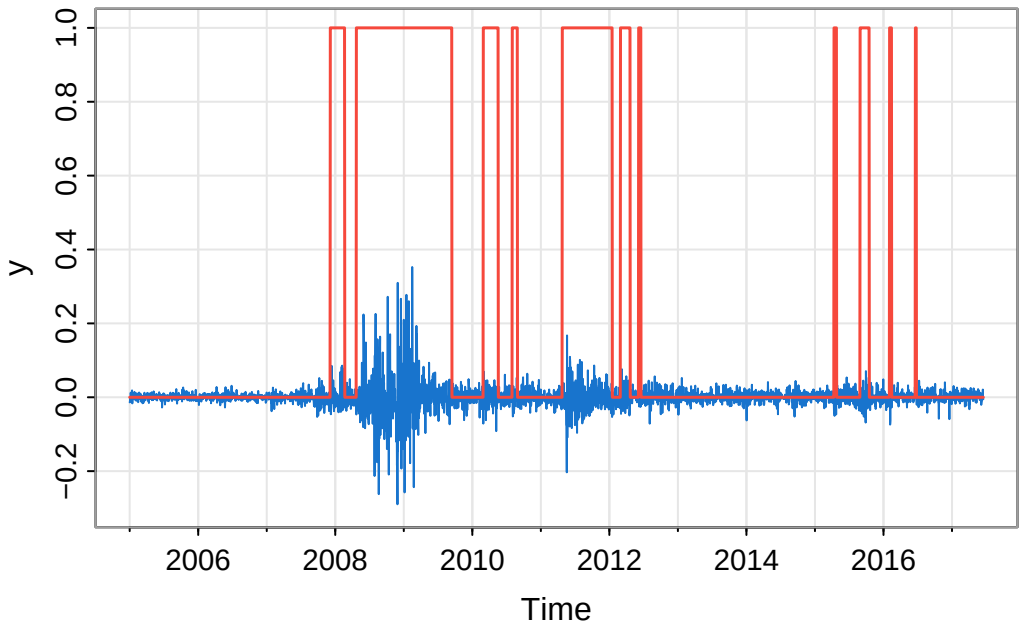
We can then create a backward stepping function for our running example with:

```
backSample = hmmSampler(
  matrix(c(0.999, 0.001, 0.005, 0.995), ncol=2, byrow=TRUE)
  )
```

We can then apply this to the list of filtered probabilities.

```
set.seed(42)
xList = Reduce(backSample, head(fpList, -1),
  init=sample(1:2, 1, prob=tail(fpList, 1)[[1]]),
  right=TRUE, acc=TRUE)
xVec = unlist(xList)
```

```
xTs = ts(xVec[-1], start=start(y), freq=frequency(y))
tsplot(y, col=4, ylim=c(-0.3, 1))
lines(xTs-1, col=2, lwd=1.5)
```



This gives us one sample from the joint conditional distribution of hidden states. Of course, every time we repeat this process we will get a different sample. Studying many such samples gives us a Monte Carlo method for understanding the joint distribution. We do not have time to explore this in detail here.

6.6 Parameter estimation

We have so far assumed that the HMM is fully specified, and that there are no unknown parameters (other than the hidden states). Of course, in practice, this is rarely the case. There could be unknown parameters in either or both of the transition matrix or observation likelihoods, for example. There are many interesting Bayesian approaches to this problem, but for this course we focus on maximum likelihood estimation.

We have seen how we can use the forward filtering algorithm to compute the marginal model likelihood. But evaluation requires a fully specified model, and therefore depends on any unknown parameters. If we call the unknown parameters $\boldsymbol{\theta}$, then the marginal likelihood is actually the likelihood function, $\ell(\boldsymbol{\theta}; \mathbf{y})$. Maximum likelihood approaches to inference seek to maximise this as a function of $\boldsymbol{\theta}$. There is a very efficient iterative algorithm for this based on the EM algorithm, which in the context of HMMs, is known as the Baum-Welch algorithm. The details are not relevant to this course, however, so we will just look at generic numerical maximisation approaches.

6.6.1 Example

For our running example, we will assume that we are happy with our specification of the transition matrix for the hidden states, but that we are not so sure about the appropriate scale parameters to use in our observation model. We will carry out ML inference for these two parameters. First, define a function that returns the marginal likelihood for a given vector of scale parameters.

```
mll = function(theta) {
  advance = hmmFilterML(
    matrix(c(0.999, 0.001, 0.005, 0.995), ncol=2, byrow=TRUE),
    function(yt)
      c(dnorm(yt, 0, theta[1]), dcauchy(yt, 0, theta[2]))
  )
  Reduce(advance, y, list(f=c(0.5, 0.5), ll=0))$ll
}

mll(c(0.015, 0.025))
```

```
[1] 7971.837
```

This works, and returns the likelihood we previously obtained when evaluated at our original guess. Can we do better than this? Let's see if `optim` can find anything better.

```
optim(c(0.015, 0.025), mll, control=list(fnscale=-1))
```

```
$par
[1] 0.01268440 0.02074005

$value
[1] 7992.119

$counts
function gradient
      51       NA

$convergence
[1] 0

$message
NULL
```

It can find parameters that are slightly better than our original guess, so we should probably re-do all of our previous analyses using these. This is left as an exercise!

6.7 R package for HMMs

We have seen some nice simple illustrative implementations of the key algorithms underpinning the analysis of HMMs. But they are not very efficient or numerically stable. Many better implementations exist, and some are available in R via packages on CRAN. The CRAN package `HiddenMarkov` is of particular note. If you want to do serious work based on HMMs, it is worth finding out how this package works.

7 Dynamic linear models (DLMs)

7.1 Introduction

The set up here is similar to the previous chapter. We have a "hidden" Markov chain model, $\mathbf{X}_t$, and a conditionally independent observation process $\mathbf{Y}_t$ leading to an observed time series $\mathbf{y}_1, \ldots, \mathbf{y}_n$. Again, we want to use the observations to learn about this hidden process. But this time, instead of a finite state Markov chain, the hidden process is a linear Gaussian model. Then, analytic tractability requires that the observation process is also linear and Gaussian. In that case, we can use standard properties of the multivariate normal distribution to do filtering, smoothing, etc., using straightforward numerical linear algebra.

For a hidden state of dimension p and an observation process of dimension m, the model can be written in conditional form as

$$\begin{aligned} \mathbf{X}_t|\mathbf{X}_{t-1} &\sim \mathcal{N}(\mathsf{G}_t\mathbf{X}_{t-1}, \mathsf{W}_t), \\ \mathbf{Y}_t|\mathbf{X}_t &\sim \mathcal{N}(\mathsf{F}_t\mathbf{X}_t, \mathsf{V}_t), \quad t = 1, 2, \ldots, n, \end{aligned}$$

for (possibly time dependent) matrices $\mathsf{G}_t, \mathsf{F}_t, \mathsf{W}_t, \mathsf{V}_t$ of appropriate dimensions, all (currently) assumed known. Note that these matrices are often chosen to be independent of time, in which case we drop the time subscript and write them $\mathsf{G}, \mathsf{F}, \mathsf{W}, \mathsf{V}$. The model is initialised with

$$\mathbf{X}_0 \sim \mathcal{N}(\mathbf{m}_0, \mathsf{C}_0),$$

for prior parameters $\mathbf{m}_0$ (a p-vector), C_0 (a $p \times p$ matrix), also assumed known. We can also write the model in state-space form as

$$\begin{aligned} \mathbf{X}_t &= \mathsf{G}_t\mathbf{X}_{t-1} + \boldsymbol{\omega}_t, \quad \boldsymbol{\omega}_t \sim \mathcal{N}(\mathbf{0}, \mathsf{W}_t), \\ \mathbf{Y}_t &= \mathsf{F}_t\mathbf{X}_t + \boldsymbol{\nu}_t, \quad \boldsymbol{\nu}_t \sim \mathcal{N}(\mathbf{0}, \mathsf{V}_t), \end{aligned}$$

from which it is possibly more clear that in the case of time independent G, W, the hidden process evolves as a VAR(1) model. The observation process is also quite flexible, allowing noisy observation of some linear transformation of the hidden state. Note that for a univariate observation process ($m = 1$), corresponding to a univariate time series, F_t will be row vector representing a linear functional of the (vector) hidden state.

7.2 Filtering

The filtering problem is the computation of $(\mathbf{X}_t|\mathbf{Y}_{1:t} = \mathbf{y}_{1:t})$, for $t = 1, 2, \ldots, n$. Since everything in the problem is linear and Gaussian, the filtering distributions will be too, so we can write

$$(\mathbf{X}_t|\mathbf{Y}_{1:t} = \mathbf{y}_{1:t}) \sim \mathcal{N}(\mathbf{m}_t, \mathsf{C}_t),$$

for some $\mathbf{m}_t$, C_t to be determined. The forward recursions we use to compute these are known as the Kalman filter.

Since we know $\mathbf{m}_0$, C_0, we assume that we know $\mathbf{m}_{t-1}$, C_{t-1} for some $t > 0$, and proceed to compute $\mathbf{m}_t$, C_t. Just as for HMMs, this is conceptually simpler to break down into two steps.

Predict step

Since $\mathbf{X}_t = \mathsf{G}_t\mathbf{X}_{t-1} + \boldsymbol{\omega}_t$ and $\mathbf{X}_t$ is conditionally independent of $\mathbf{Y}_{1:(t-1)}$ given $\mathbf{X}_{t-1}$, we will have

$$\begin{aligned}\mathbb{E}[\mathbf{X}_t|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}] &= \mathbb{E}\left\{\mathbb{E}[\mathbf{X}_t|\mathbf{X}_{t-1}]\,\middle|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right\} \\ &= \mathbb{E}\left\{\mathsf{G}_t\mathbf{X}_{t-1}|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right\} \\ &= \mathsf{G}_t\mathbb{E}\left\{\mathbf{X}_{t-1}|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right\} \\ &= \mathsf{G}_t\mathbf{m}_{t-1}.\end{aligned}$$

Similarly,

$$\begin{aligned}\mathbb{V}\mathrm{ar}[\mathbf{X}_t|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}] &= \mathbb{V}\mathrm{ar}\left\{\mathbb{E}[\mathbf{X}_t|\mathbf{X}_{t-1}]|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right\} \\ &\quad + \mathbb{E}\left\{\mathbb{V}\mathrm{ar}[\mathbf{X}_t|\mathbf{X}_{t-1}]|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right\} \\ &= \mathbb{V}\mathrm{ar}\left\{\mathsf{G}_t\mathbf{X}_{t-1}|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right\} + \mathbb{E}\left\{\mathsf{W}_t|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right\} \\ &= \mathsf{G}_t\mathsf{C}_{t-1}\mathsf{G}_t^\top + \mathsf{W}_t.\end{aligned}$$

We can therefore write

$$(\mathbf{X}_t|\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}) \sim \mathcal{N}(\tilde{\mathbf{m}}_t, \tilde{\mathsf{C}}_t),$$

where we define

$$\tilde{\mathbf{m}}_t = \mathsf{G}_t\mathbf{m}_{t-1}, \quad \tilde{\mathsf{C}}_t = \mathsf{G}_t\mathsf{C}_{t-1}\mathsf{G}_t^\top + \mathsf{W}_t.$$

Update step

Using $\mathbf{Y}_t = \mathsf{F}_t\mathbf{X}_t + \boldsymbol{\nu}_t$ and some basic linear normal properties, we deduce

$$\left(\begin{array}{c}\mathbf{X}_t \\ \mathbf{Y}_t\end{array}\,\middle|\,\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right) \sim \mathcal{N}\left(\left[\begin{array}{c}\tilde{\mathbf{m}}_t \\ \mathsf{F}_t\tilde{\mathbf{m}}_t\end{array}\right], \left[\begin{array}{cc}\tilde{\mathsf{C}}_t & \tilde{\mathsf{C}}_t\mathsf{F}_t^\top \\ \mathsf{F}_t\tilde{\mathsf{C}}_t & \mathsf{F}_t\tilde{\mathsf{C}}_t\mathsf{F}_t^\top + \mathsf{V}_t\end{array}\right]\right).$$

If we define $\mathbf{f}_t = \mathsf{F}_t\tilde{\mathbf{m}}_t$ and $\mathsf{Q}_t = \mathsf{F}_t\tilde{\mathsf{C}}_t\mathsf{F}_t^\top + \mathsf{V}_t$ this simplifies to

$$\left(\begin{array}{c}\mathbf{X}_t \\ \mathbf{Y}_t\end{array}\,\middle|\,\mathbf{Y}_{1:(t-1)} = \mathbf{y}_{1:(t-1)}\right) \sim \mathcal{N}\left(\left[\begin{array}{c}\tilde{\mathbf{m}}_t \\ \mathbf{f}_t\end{array}\right], \left[\begin{array}{cc}\tilde{\mathsf{C}}_t & \tilde{\mathsf{C}}_t\mathsf{F}_t^\top \\ \mathsf{F}_t\tilde{\mathsf{C}}_t & \mathsf{Q}_t\end{array}\right]\right).$$

We can now use the multivariate normal conditioning formula to obtain

$$\begin{aligned}\mathbf{m}_t &= \tilde{\mathbf{m}}_t + \tilde{\mathsf{C}}_t\mathsf{F}_t^\top\mathsf{Q}_t^{-1}[\mathbf{y}_t - \mathbf{f}_t] \\ \mathsf{C}_t &= \tilde{\mathsf{C}}_t - \tilde{\mathsf{C}}_t\mathsf{F}_t^\top\mathsf{Q}_t^{-1}\mathsf{F}_t\tilde{\mathsf{C}}_t.\end{aligned}$$

The weight that is applied to the discrepancy between the new observation and its forecast is often referred to as the (optimal) *Kalman gain*, and denoted K_t. So if we define

$$\mathsf{K}_t = \tilde{\mathsf{C}}_t\mathsf{F}_t^\top\mathsf{Q}_t^{-1}$$

we get

$$\begin{aligned}\mathbf{m}_t &= \tilde{\mathbf{m}}_t + \mathsf{K}_t[\mathbf{y}_t - \mathbf{f}_t] \\ \mathsf{C}_t &= \tilde{\mathsf{C}}_t - \mathsf{K}_t\mathsf{Q}_t\mathsf{K}_t^\top.\end{aligned}$$

7.2.1 The Kalman filter

Let us now summarise the steps of the Kalman filter. Our input at time t is $\mathbf{m}_{t-1}$, C_{t-1} and $\mathbf{y}_t$. We then compute:

- $\tilde{\mathbf{m}}_t = \mathsf{G}_t\mathbf{m}_{t-1}, \quad \tilde{\mathsf{C}}_t = \mathsf{G}_t\mathsf{C}_{t-1}\mathsf{G}_t^\top + \mathsf{W}_t$

- $\mathbf{f}_t = \mathsf{F}_t\tilde{\mathbf{m}}_t, \quad \mathsf{Q}_t = \mathsf{F}_t\tilde{\mathsf{C}}_t\mathsf{F}_t^\top + \mathsf{V}_t$
- $\mathsf{K}_t = \tilde{\mathsf{C}}_t\mathsf{F}_t^\top\mathsf{Q}_t^{-1}$
- $\mathbf{m}_t = \tilde{\mathbf{m}}_t + \mathsf{K}_t[\mathbf{y}_t - \mathbf{f}_t], \quad \mathsf{C}_t = \tilde{\mathsf{C}}_t - \mathsf{K}_t\mathsf{Q}_t\mathsf{K}_t^\top.$

Note that the only inversion involves the $m \times m$ matrix Q_t (there are no $p \times p$ inversions). So, in the case of a univariate time series, no matrix inversions are required.

Also note that it is easy to handle missing data within the Kalman filter by carrying out the *predict* step and skipping the *update* step. If the observation at time t is missing, simply compute $\tilde{\mathbf{m}}_t$ and $\tilde{\mathsf{C}}_t$ as usual, but then skip this update step, since there is no observation to condition on, returning $\tilde{\mathbf{m}}_t$ and $\tilde{\mathsf{C}}_t$ as $\mathbf{m}_t$ and C_t.

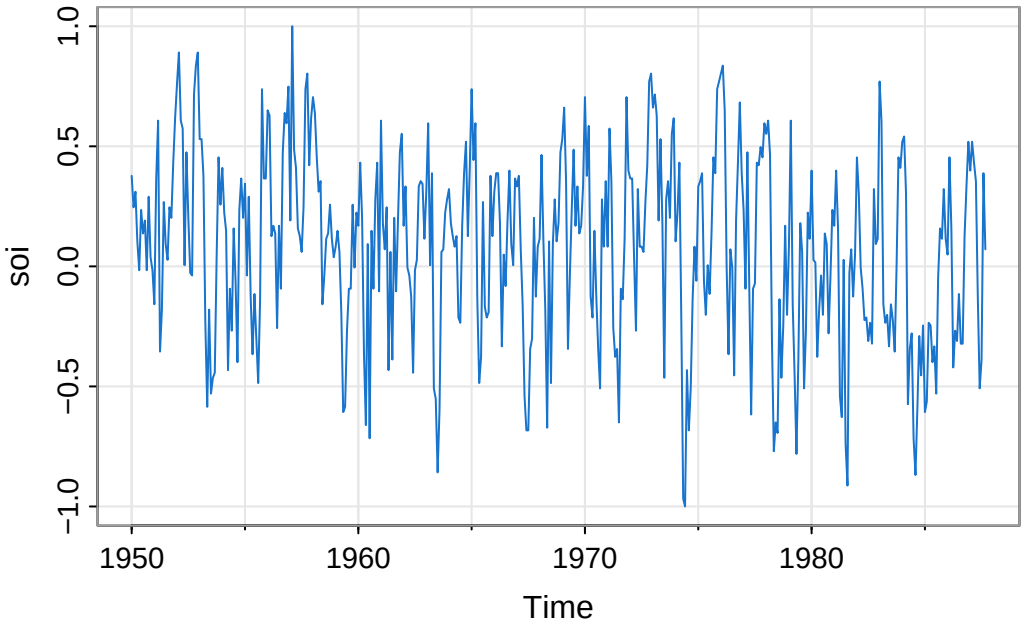
7.2.2 Example

To keep things simple, we will just implement the Kalman filter for time-invariant $\mathsf{G}, \mathsf{W}, \mathsf{F}, \mathsf{V}$.

```
kFilter = function(G, W, F, V)
  function(mC, y) {
    m = G %*% mC$m
    C = (G %*% mC$C %*% t(G)) + W
    f = F %*% m
    Q = (F %*% C %*% t(F)) + V
    K = t(solve(Q, F %*% C))
    m = m + (K %*% (y - f))
    C = C - (K %*% Q %*% t(K))
    list(m=m, C=C)
  }
```

So, if we provide $\mathsf{G}, \mathsf{W}, \mathsf{F}, \mathsf{V}$ to the `kFilter` function, we will get back a function for carrying out one step of the Kalman filter. As a simple illustration, suppose that we are interested in the long-term trend in the Southern Oscillation Index (SOI).

```
library(astsa)
tsplot(soi, col=4)
```



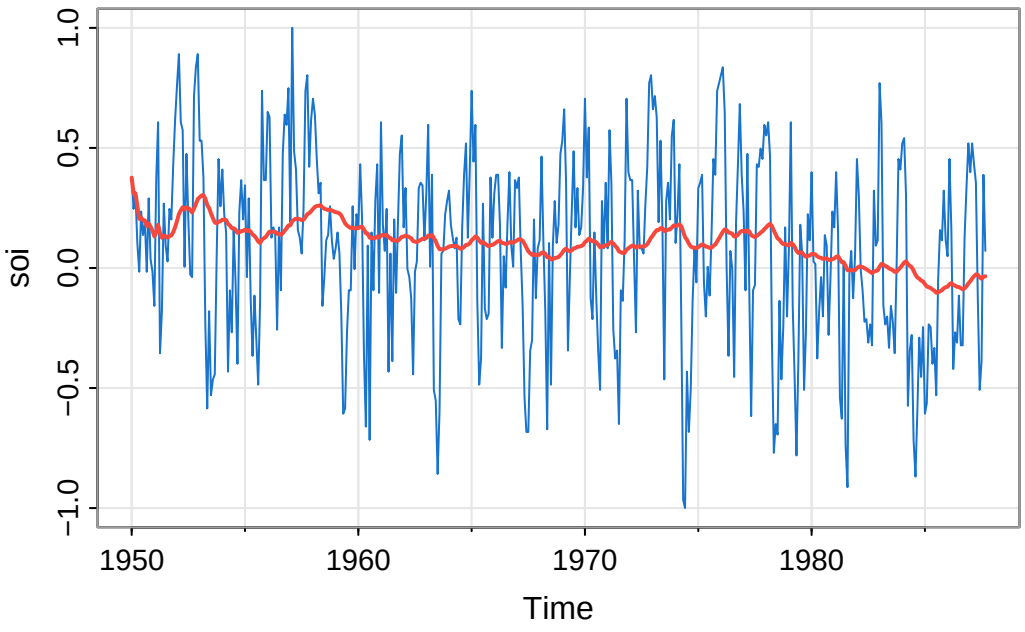
We suppose that a simple random walk model is appropriate for the (hidden) long term trend (so $\mathsf{G} = 1$, and note that the implied VAR(1) model is not stationary), and that the observations are simply noisy observations of that hidden trend (so $\mathsf{F} = 1$). To complete the model, we suppose that the monthly change in the long term trend has standard deviation 0.01, and that the noise variance associated with the observations has standard deviation 0.5. We can now create the filter and apply it, using the `Reduce` function and initialising with $\mathbf{m}_0 = 0$ and $\mathsf{C}_0 = 100$.

```
advance = kFilter(1, 0.01^2, 1, 0.5^2)
Reduce(advance, soi, list(m=0, C=100))
```

```
$m
             [,1]
[1,] -0.03453493

$C
            [,1]
[1,] 0.00495025
```

This gives us $\mathbf{m}_n$ and C_n, which might be what we want if we are simply interested in forecasting the future (see next section). However, more likely we want to keep the full set of filtered states and plot them over the data.

```
fs = Reduce(advance, soi, list(m=0, C=100), acc=TRUE)
fsm = sapply(fs, function(s) s$m)
fsTs = ts(fsm[-1], start=start(soi), freq=frequency(soi))
tsplot(soi, col=4)
lines(fsTs, col=2, lwd=2)
```



We will look at more interesting DLM models in the next chapter.

7.3 Forecasting

Making forecasts from a DLM is straightforward, since it corresponds to the computation of predictions, and we have already examined this problem in the context of the *predict* step of a Kalman filter. Suppose that we have a time series of length n, and have run a Kalman filter over the whole data set, culminating in the computation of $\mathbf{m}_n$ and C_n. We can now define our k-step ahead forecast distributions as

$$(\mathbf{X}_{n+k}|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}) \sim \mathcal{N}(\hat{\mathbf{m}}_n(k), \mathsf{C}_n(k))$$
$$(\mathbf{Y}_{n+k}|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}) \sim \mathcal{N}(\hat{\mathbf{f}}_n(k), \mathsf{Q}_n(k)),$$

for moments $\hat{\mathbf{m}}_n(k),\ \mathsf{C}_n(k),\ \hat{\mathbf{f}}_n(k),\ \mathsf{Q}_n(k)$ to be determined. Starting from $\hat{\mathbf{m}}_n(0) = \mathbf{m}_n$ and $\mathsf{C}_n(0) = \mathsf{C}_n$, we can recursively compute subsequent forecasts for the hidden states via

$$\hat{\mathbf{m}}_n(k) = \mathsf{G}_{n+k}\hat{\mathbf{m}}_n(k-1), \quad \mathsf{C}_n(k) = \mathsf{G}_{n+k}\mathsf{C}_n(k-1)\mathsf{G}_{n+k}^\top + \mathsf{W}_{n+k}.$$

Once we have a forecast for the hidden state, we can compute the moments of the forecast distribution via

$$\hat{\mathbf{f}}_n(k) = \mathsf{F}_{n+k}\hat{\mathbf{m}}_n(k), \quad \mathsf{Q}_n(k) = \mathsf{F}_{n+k}\mathsf{C}_n(k)\mathsf{F}_{n+k}^\top + \mathsf{V}_{n+k}.$$

7.4 Marginal likelihood

For many applications (including parameter estimation) it is useful to know the marginal likelihood of the data. We can factorise this as

$$L(\mathbf{y}_{1:n}) = \prod_{t=1}^{n} \pi(\mathbf{y}_t|\mathbf{y}_{1:(t-1)}),$$

but as part of the Kalman filter we compute the component distributions as

$$\pi(\mathbf{y}_t|\mathbf{y}_{1:(t-1)}) = \mathcal{N}(\mathbf{y}_t; \mathbf{f}_t, \mathsf{Q}_t),$$

so we can compute the marginal likelihood as part of the Kalman filter at little extra cost. In practice we compute the log-likelihood

$$\ell(\mathbf{y}_{1:t}) = \sum_{t=1}^{n} \log \mathcal{N}(\mathbf{y}_t; \mathbf{f}_t, \mathsf{Q}_t),$$

We might use a function (such as `mvtnorm::dmvnorm`) to evaluate the log density of a multivariate normal, but if not, we can write it explicitly in the form

$$\ell(\mathbf{y}_{1:t}) = -\frac{mn}{2}\log 2\pi - \frac{1}{2}\sum_{t=1}^{n}\left[\log|\mathsf{Q}_t| + (\mathbf{y}_t - \mathbf{f}_t)^\top \mathsf{Q}_t^{-1}(\mathbf{y}_t - \mathbf{f}_t)\right].$$

Note that this is very straightforward in the case of a univariate time series (scalar Q_t), but more generally, there is a good way to evaluate this using the Cholesky decomposition of Q_t. The details are not important for this course.

7.4.1 Example

We can easily modify our Kalman filter function to compute the marginal likelihood, and apply it to our example problem as follows.

```
kFilterML = function(G, W, F, V)
  function(mCl, y) {
    m = G %*% mCl$m
```

```
    C = (G %*% mCl$C %*% t(G)) + W
    f = F %*% m
    Q = (F %*% C %*% t(F)) + V
    ll = mvtnorm::dmvnorm(y, f, Q, log=TRUE)
    K = t(solve(Q, F %*% C))
    m = m + (K %*% (y - f))
    C = C - (K %*% Q %*% t(K))
    list(m=m, C=C, ll=mCl$ll+ll)
  }

advance = kFilterML(1, 0.01^2, 1, 0.5^2)
Reduce(advance, soi, list(m=0, C=100, ll=0))
```

```
$m
             [,1]
[1,] -0.03453493

$C
            [,1]
[1,] 0.00495025

$ll
[1] -237.2907
```

7.5 Smoothing

For the smoothing problem, we want to calculate the marginal at each time given *all* of the data,

$$\mathbf{X}_t|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}, \quad t = 1, 2, \ldots, n.$$

We will use a backward recursion to achieve this, known in this context as the Rauch-Tung-Striebel (RTS) smoother. Again, since everything is linear and Gaussian, the marginals will be, so we have

$$(\mathbf{X}_t|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}) \sim \mathcal{N}(\mathbf{s}_t, \mathsf{S}_t),$$

for $\mathbf{s}_t$, S_t to be determined. We know from the forward filter that $\mathbf{s}_n = \mathbf{m}_n$ and $\mathsf{S}_n = \mathsf{C}_n$, so we assume that we know $\mathbf{s}_{t+1}$, S_{t+1} for some $t < n$ and that we want to know $\mathbf{s}_t$, S_t.

Since $\mathbf{X}_{t+1} = \mathsf{G}_{t+1}\mathbf{X}_t + \boldsymbol{\omega}_{t+1}$, we deduce that

$$\left(\begin{array}{c} \mathbf{X}_t \\ \mathbf{X_{t+1}} \end{array} \middle| \mathbf{Y}_{1:t} = \mathbf{y}_{1:t} \right) \sim \mathcal{N}\left(\left[\begin{array}{c} \mathbf{m}_t \\ \tilde{\mathbf{m}}_{t+1} \end{array} \right], \left[\begin{array}{cc} \mathsf{C}_t & \mathsf{C}_t\mathsf{G}_{t+1}^\top \\ \mathsf{G}_{t+1}\mathsf{C}_t & \tilde{\mathsf{C}}_{t+1} \end{array} \right] \right).$$

We can use normal conditioning formula to deduce that

$$(\mathbf{X}_t|\mathbf{X}_{t+1}, \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}) \sim \mathcal{N}\left(\mathbf{m}_t + \mathsf{C}_t\mathsf{G}_{t+1}^\top\tilde{\mathsf{C}}_{t+1}^{-1}[\mathbf{X}_{t+1} - \tilde{\mathbf{m}}_{t+1}], \mathsf{C}_t - \mathsf{C}_t\mathsf{G}_{t+1}^\top\tilde{\mathsf{C}}_{t+1}^{-1}\mathsf{G}_{t+1}\mathsf{C}_t\right).$$

If we define the smoothing gain $\mathsf{L}_t = \mathsf{C}_t\mathsf{G}_{t+1}^\top\tilde{\mathsf{C}}_{t+1}^{-1}$, then this becomes

$$(\mathbf{X}_t|\mathbf{X}_{t+1}, \mathbf{Y}_{1:t} = \mathbf{y}_{1:t}) \sim \mathcal{N}\left(\mathbf{m}_t + \mathsf{L}_t[\mathbf{X}_{t+1} - \tilde{\mathbf{m}}_{t+1}], \mathsf{C}_t - \mathsf{L}_t\tilde{\mathsf{C}}_{t+1}\mathsf{L}_t^\top\right).$$

But now, since $\mathbf{Y}_{(t+1):n}$ is conditionally independent of $\mathbf{X}_t$ given $\mathbf{X}_{t+1}$, this distribution is *also* the distribution of $(\mathbf{X}_t|\mathbf{X}_{t+1}, \mathbf{Y}_{1:n} = \mathbf{y}_{1:n})$. So now we can compute $(\mathbf{X}_t|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n})$ by marginalising out $\mathbf{X}_{t+1}$. First,

$$\begin{aligned}
\mathbf{s}_t &= \mathbb{E}[\mathbf{X}_t|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] \\
&= \mathbb{E}\left[\mathbb{E}[\mathbf{X}_t|\mathbf{X}_{t+1}, \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}]\middle|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}\right] \\
&= \mathbb{E}\left[\mathbf{m}_t + \mathsf{L}_t[\mathbf{X}_{t+1} - \tilde{\mathbf{m}}_{t+1}]\middle|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}\right] \\
&= \mathbf{m}_t + \mathsf{L}_t[\mathbf{s}_{t+1} - \tilde{\mathbf{m}}_{t+1}]
\end{aligned}$$

Now,

$$\begin{aligned}
\mathsf{S}_t &= \mathbb{V}\mathrm{ar}[\mathbf{X}_t|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}] \\
&= \mathbb{E}\{\mathbb{V}\mathrm{ar}[\mathbf{X}_t|\mathbf{X}_{t+1}, \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}]|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}\} \\
&\quad + \mathbb{V}\mathrm{ar}\{\mathbb{E}[\mathbf{X}_t|\mathbf{X}_{t+1}, \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}]|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}\} \\
&= \mathbb{E}\{\mathsf{C}_t - \mathsf{L}_t\tilde{\mathsf{C}}_{t+1}\mathsf{L}_t^\top|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}\} \\
&\quad + \mathbb{V}\mathrm{ar}\{\mathbf{m}_t + \mathsf{L}_t[\mathbf{X}_{t+1} - \tilde{\mathbf{m}}_{t+1}]|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}\} \\
&= \mathsf{C}_t - \mathsf{L}_t\tilde{\mathsf{C}}_{t+1}\mathsf{L}_t^\top + \mathsf{L}_t\mathsf{S}_{t+1}\mathsf{L}_t^\top \\
&= \mathsf{C}_t + \mathsf{L}_t[\mathsf{S}_{t+1} - \tilde{\mathsf{C}}_{t+1}]\mathsf{L}_t^\top
\end{aligned}$$

7.5.1 The RTS smoother

To summarise, we receive as input $\mathbf{s}_{t+1}$ and S_{t+1}, and compute

$$\begin{aligned}
\mathsf{L}_t &= \mathsf{C}_t\mathsf{G}_{t+1}^\top\tilde{\mathsf{C}}_{t+1}^{-1} \\
\mathbf{s}_t &= \mathbf{m}_t + \mathsf{L}_t[\mathbf{s}_{t+1} - \tilde{\mathbf{m}}_{t+1}] \\
\mathsf{S}_t &= \mathsf{C}_t + \mathsf{L}_t[\mathsf{S}_{t+1} - \tilde{\mathsf{C}}_{t+1}]\mathsf{L}_t^\top,
\end{aligned}$$

re-using computations from the forward Kalman filter as appropriate. Note that the RTS smoother *does* involve the inversion of a $p \times p$ matrix.

7.6 Sampling

If we want to understand the joint distribution of the hidden states given the data, generating samples from it will be useful. So, we want to generate exact samples from

$$(\mathbf{X}_{1:n}|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}).$$

Just as for HMMs, it is convenient to do this backwards, starting from the final hidden state. We assume that we have run a Kalman filter forward, but now do sampling on the backward pass. This strategy is known in this context as *forward filtering, backward sampling* (FFBS). We know that the distribution of the final hidden state is

$$(\mathbf{X}_n|\mathbf{Y}_{1:n} = \mathbf{y}_{1:n}) \sim \mathcal{N}(\mathbf{m}_n, \mathsf{C}_n),$$

so we can sample from this to obtain a realisation of $\mathbf{x}_n$. So now assume that we are given $\mathbf{x}_{t+1}$ for some $t < n$, and want to simulate an appropriate $\mathbf{x}_t$. From the smoothing analysis, we know that

$$(\mathbf{X}_t|\mathbf{X}_{t+1} = \mathbf{x}_{t+1}, \mathbf{Y}_{1:n} = \mathbf{y}_{1:n}) \sim \mathcal{N}\left(\mathbf{m}_t + \mathsf{L}_t[\mathbf{x}_{t+1} - \tilde{\mathbf{m}}_{t+1}], \mathsf{C}_t - \mathsf{L}_t\tilde{\mathsf{C}}_{t+1}\mathsf{L}_t^\top\right).$$

But since $\mathbf{X}_t$ is conditionally independent of all future $\mathbf{X}_s,\ s > t+1$, given $\mathbf{X}_{t+1}$, this is also the distribution of $(\mathbf{X}_t|\mathbf{X}_{(t+1):n} = \mathbf{x}_{(t+1):n}, \mathbf{Y}_{1:n} = \mathbf{y}_{1:n})$ and so we just sample from this, and so on, in a backward pass.

7.7 Parameter estimation

Just as for HMMs, there are a wide variety of Bayesian and likelihood-based approaches to parameter estimation. Here, again, we will keep things simple and illustrate basic numerical optimisation of the marginal log-likelihood of the data.

7.7.1 Example

For our SOI running example, suppose that we are happy with the model structure, but uncertain about the variance parameters V and W. We can do maximum likelihood estimation for those two parameters by first creating a function to evaluate the likelihood for a given parameter combination.

```
mll = function(wv) {
  advance = kFilterML(1, wv[1], 1, wv[2])
  Reduce(advance, soi, list(m=0, C=100, ll=0))$ll
}

mll(c(0.01^2, 0.5^2))
```

```
[1] -237.2907
```

We can see that it reproduces the same log-likelihood as we got before when we evaluate it at our previous guess for V and W. But can we do better than this? Let's see what `optim` can do.

```
optim(c(0.01^2, 0.5^2), mll, control=list(fnscale=-1))
```

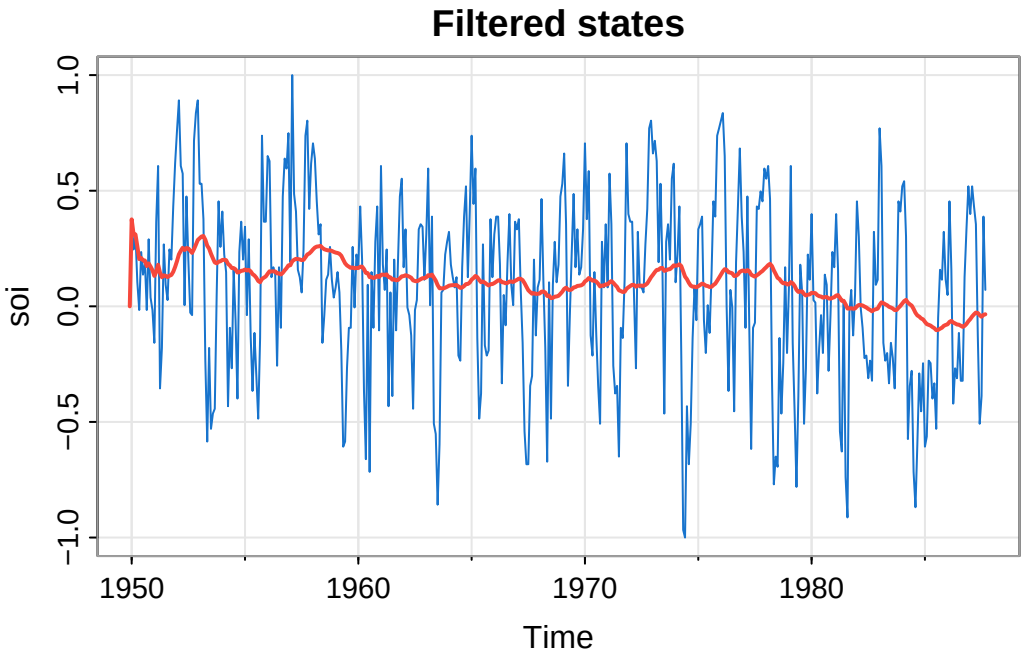
```
$par
[1] 0.05696905 0.03029240

$value
[1] -144.0333

$counts
function gradient
      91       NA

$convergence
[1] 0

$message
NULL
```

So `optim` has found parameters with a *much* higher likelihood, corresponding to a much better fitting model. Note that these optimised parameters apportion the noise more equally between the hidden states and the observation process, and so lead to smoothing the hidden state much less. This may correspond to a better fit, but perhaps not to the fit that we really want. In the context of Bayesian inference, a strong prior on V and W could help to regularise the problem.

7.8 R package for DLMs

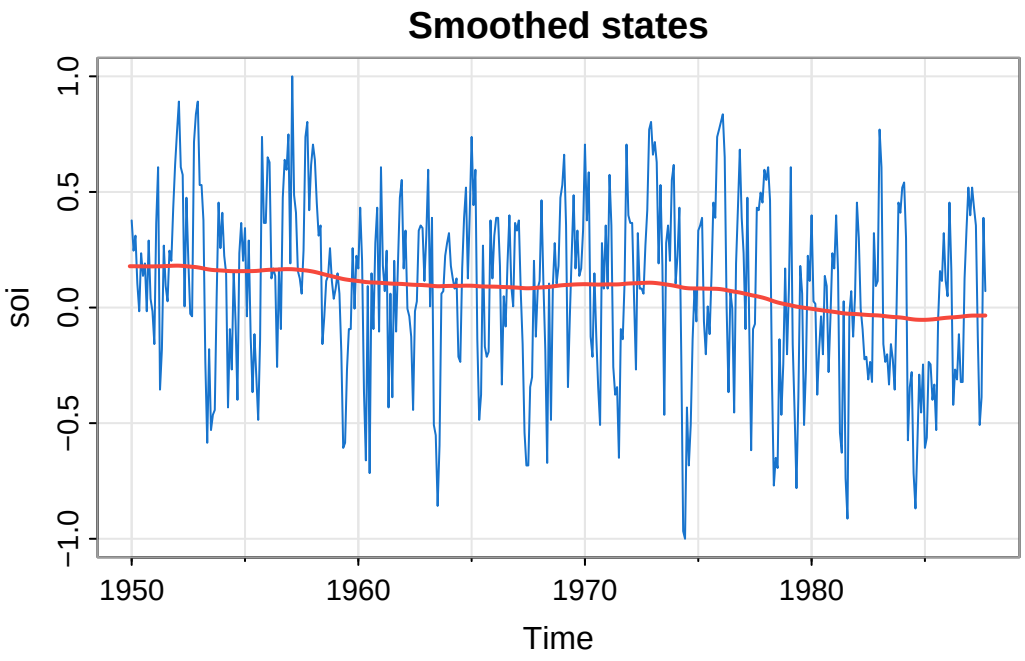
We have seen that it is quite simple to implement the Kalman filter and related computations such as the marginal log-likelihood of the data. However, a very naive implementation such as we have given may not be the most efficient, nor the most numerically stable. Many different approaches to implementing the Kalman filter have been developed over the years, which are mathematically equivalent to the basic Kalman filter recursions, but are more efficient and/or more numerically stable than a naive approach. The `dlm` R package is an implementation based on the singular value decomposition (SVD), which is significantly more numerically stable than a naive implementation. We will use this package for the study of more interesting DLMs in Chapter 8. More information about this package can be obtained with `help(package="dlm")` and `vignette("dlm", package="dlm")`. It is described more fully in Petris, Petrone, and Campagnoli (2009). We can fit our simple DLM to the SOI data as follows.

```
library(dlm)
mod = dlm(FF=1, GG=1, V=0.5^2, W=0.01^2, m0=0, C0=100)
fs = dlmFilter(soi, mod)
tsplot(soi, col=4, main="Filtered states")
lines(fs$m, col=2, lwd=2)
```



We can also compute the smoothed states.

```
ss = dlmSmooth(soi, mod)
tsplot(soi, col=4, main="Smoothed states")
lines(ss$s, col=2, lwd=2)
```



These are very smooth. We can also optimise the parameters, similar to the approach we have already seen. But note that the `dlmMLE` optimiser requires that the vector of parameters being optimised is unconstrained, so we use exp and log to map back and forth between a constrained and unconstrained space.

```
buildMod = function(lwv)
  dlm(FF=1, GG=1, V=exp(lwv[2]), W=exp(lwv[1]),
    m0=0, C0=100)

opt = dlmMLE(soi, parm=c(log(0.5^2), log(0.01^2)),
  build=buildMod)
opt
```

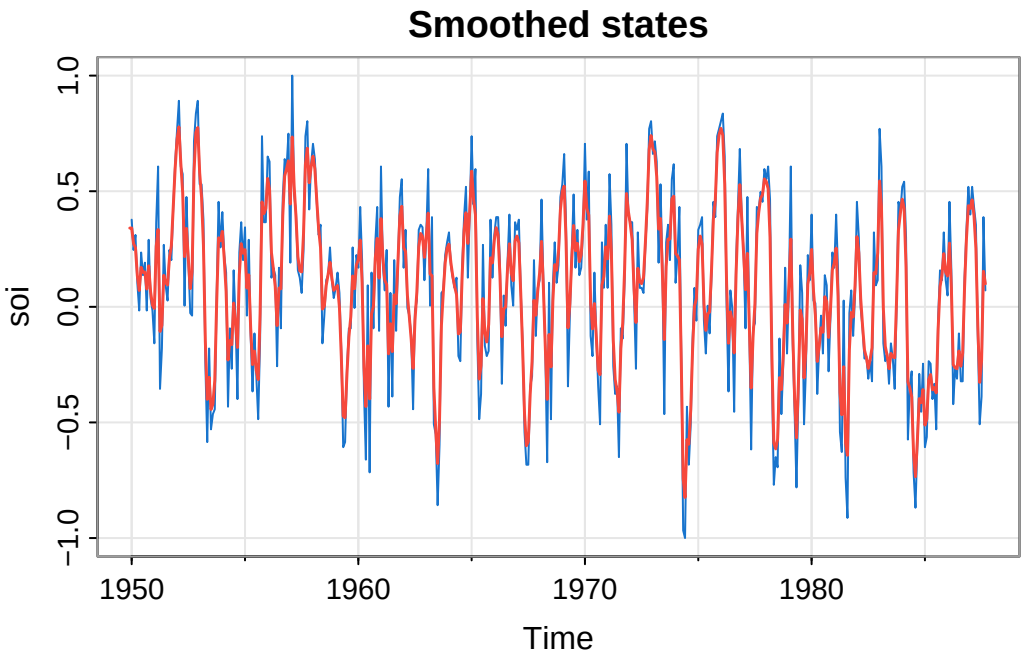
```
$par
[1] -2.865240 -3.496717

$value
[1] -272.2459

$counts
function gradient
      35       35

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
exp(opt$par)
```

```
[1] 0.05696943 0.03029668
```

After mapping back to the constrained space, we see that we get the same optimised values as before (the log-likelihood is different, but that is just due to dropping of unimportant constant terms). We can check the smoothed states for this optimised model as follows.

```
mod = buildMod(opt$par)
ss = dlmSmooth(soi, mod)
tsplot(soi, col=4, main="Smoothed states")
lines(ss$s, col=2, lwd=1.5)
```



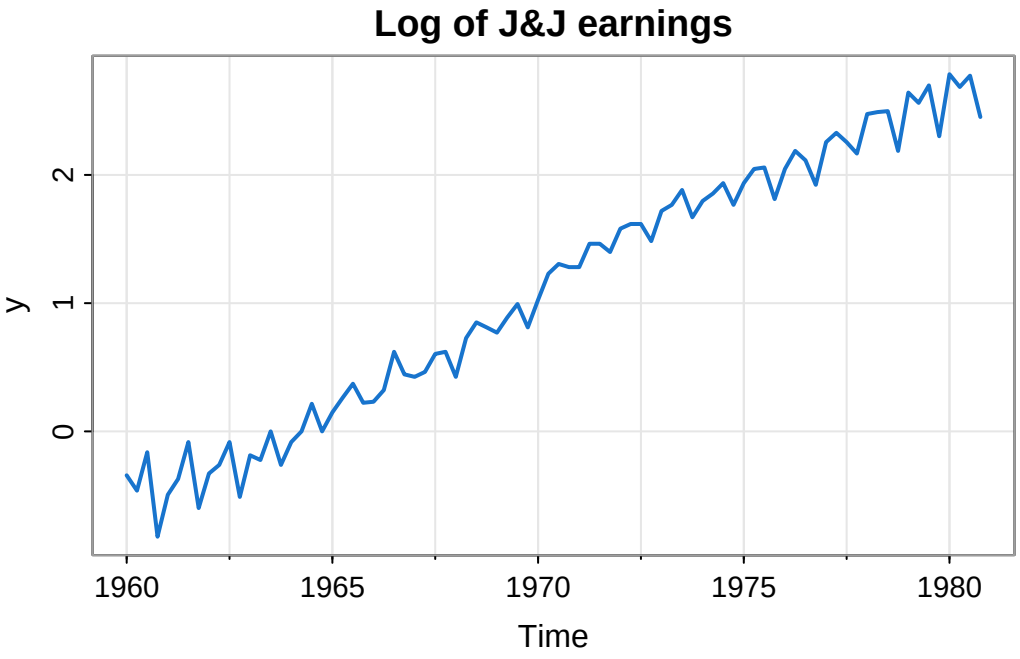
This confirms that the smoothed states for the optimised model are not very smooth. This suggests that our simple random walk model for the data is perhaps not very good.

8 State space modelling

8.1 Introduction

In Chapter 7 we introduced the DLM and showed how a model of DLM form could be used for filtering, smoothing, forecasting, etc. However, we haven't yet discussed how to go about "building" an appropriate DLM model (in particular, specifying $\mathsf{G}_t, \mathsf{F}_t, \mathsf{W}_t, \mathsf{V}_t$) for a given time series. That is the topic of this chapter. We will use the `dlm` R package to illustrate the concepts. We will start by looking at some simple models for long-term trends, then consider seasonal effects, then the problem of combining trends and seasonal effects, before going on to think about the incorporation of ARMA components. We will focus mainly on (time invariant models for) univariate time series, but will briefly examine multivariate time series at the end of the chapter. Our main running example will be (the log of) J&J earnings, considered briefly in Chapter 1.

```
library(astsa)
library(dlm)
y = log(jj)
tsplot(y, col=4, lwd=2, main="Log of J&J earnings")
```



8.2 Polynomial trend

8.2.1 Locally constant model

In the previous chapter we saw that the choice $\mathsf{G} = 1, \mathsf{F} = 1$ corresponds to noisy observation of a Gaussian random walk. This random walk model for the underlying (hidden) state is known as the *locally constant model* in the context of DLMs, and sometimes (for reasons to become clear), as the *first order polynomial*

trend model. Such a model is clearly inappropriate for the J&J example, since the trend appears to be (at least, locally) linear.

8.2.2 Locally linear model

We can model a time series with a locally linear trend by choosing $\mathsf{F} = (1, 0)$,

$$\mathsf{G} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \mathsf{W} = \begin{pmatrix} w_1 & 0 \\ 0 & w_2 \end{pmatrix}.$$

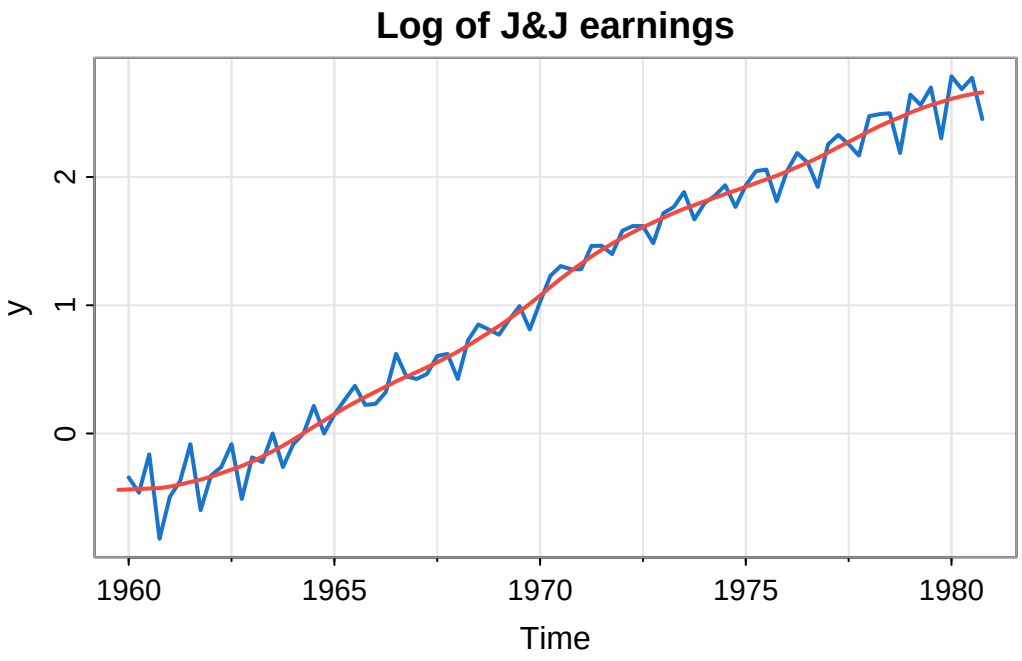
This is known as the *locally linear model*, or sometimes as the *second order polynomial trend* model. To see why this corresponds to a linear trend, it is perhaps helpful to write the hidden state vector as $\mathbf{X}_t = (\mu_t, \tau_t)^\top$. So then y_t is clearly just a noisy observation of μ_t (the current *level*), and $\mu_t = \mu_{t-1} + \tau_{t-1} + \omega_{1,t}$, so the current level increases by a systematic amount τ_{t-1} (the *trend*), and is also corrupted by noise. On the other hand, $\tau_t = \tau_{t-1} + \omega_{2,t}$ is just a random walk (typically with very small variance, so that the trend changes very slowly).

We can create such a model using the `dlm::dlmModPoly` function, by providing the order required (2 for locally linear), and the diagonal of V and W (off-diagonal elements are assumed to be zero).

```
mod = dlmModPoly(2, dV=0.1^2, dW=c(0.01^2, 0.01^2))
mod
```

```
$FF
     [,1] [,2]
[1,]    1    0

$V
     [,1]
[1,] 0.01

$GG
     [,1] [,2]
[1,]    1    1
[2,]    0    1

$W
      [,1]  [,2]
[1,] 1e-04 0e+00
[2,] 0e+00 1e-04

$m0
[1] 0 0

$C0
      [,1]  [,2]
[1,] 1e+07 0e+00
[2,] 0e+00 1e+07
```

```
ss = dlmSmooth(y, mod)
tsplot(y, col=4, lwd=2, main="Log of J&J earnings")
lines(ss$s[,1], col=2, lwd=2)
```



This captures the trend quite well, but not the strong seasonal effect. We will return to this issue shortly.

8.2.3 Higher-order polynomial models

This strategy extends straightforwardly to higher order polynomials. For example, a locally quadratic (order 3) model can be constructed using $\mathsf{F} = (1, 0, 0)$ and

$$\mathsf{G} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

A diagonal W is typically adopted, sometimes with some of the diagonal elements set to zero. Models beyond order 3 are rarely used in practice.

8.3 Seasonal models

Many time series exhibit periodic behaviour with known period. There are two commonly used approaches used to modelling such seasonal behaviour using DLMs. Both involve using a cyclic matrix, G, with period s (eg. $s = 12$ for monthly data), so that $\mathsf{G}^s = \mathbb{I}$. We will look briefly at both.

8.3.1 Seasonal effects

Arguably the most straightforward way to model seasonality is to explicitly model s separate seasonal effects, and rotate and evolve them, as required. So, for a purely seasonal model, call the s seasonal effects $\alpha_1, \alpha_2, \ldots, \alpha_s$, and make these the components of the state vector, $\mathbf{X}_t$. Then choosing the $1 \times s$ matrix $\mathsf{F} = (1, 0, \ldots, 0)$ and G the $s \times s$ cyclic permutation matrix

$$\mathsf{G} = \begin{pmatrix} \mathbf{0}^\top & 1 \\ \mathbb{I} & \mathbf{0} \end{pmatrix}$$

gives the basic evolution structure. For example, in the case $s = 4$ (quarterly data), we have

$$\mathsf{G} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

and so

$$\mathsf{G} \begin{pmatrix} \alpha_4 \\ \alpha_3 \\ \alpha_2 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_4 \\ \alpha_3 \\ \alpha_2 \end{pmatrix},$$

and the seasonal effects get rotated through the state vector each time, repeating after s time points. If we want the seasonal effects to be able to evolve gradually over time, we can allow this by choosing (say)

$$\mathsf{W} = \mathrm{diag}\{w, 0, \ldots, 0\},$$

for some $w > 0$.

This structure would be fine for a purely seasonal model, but in practice a seasonal component is often used as part of a larger model that also models the overall mean of the process. In this case there is an identifiability issue, exactly analogous to the problem that arises in linear models with a categorical covariate. Adding a constant value to each of the seasonal effects and subtracting the same value from the overall mean leads to exactly the same model. So we need to constrain the effects to maintain identifiability of the overall mean. This is typically done by ensuring that the effects sum to zero. There are many ways that such a constraint could be encoded with a DLM, but one common approach is to notice that in this case there are only $s - 1$ degrees of freedom for the effects (eg. the final effect is just minus the sum of the others), and so we can model the effects with an $s - 1$ dimensional state vector. We can then update this with the $(s - 1) \times (s - 1)$ evolution matrix

$$\mathsf{G} = \begin{pmatrix} -\mathbf{1}^\top & -1 \\ \mathbb{I} & \mathbf{0} \end{pmatrix}.$$

We will illustrate for $s = 4$. Suppose that our hidden state is $\mathbf{X}_t = (\alpha_4, \alpha_3, \alpha_2)^\top$. We know that we can reconstruct the final effect, $\alpha_1 = -(\alpha_4 + \alpha_3 + \alpha_2)$, so this vector captures all of our seasonal effects. But now

$$\begin{pmatrix} -1 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_4 \\ \alpha_3 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_4 \\ \alpha_3 \end{pmatrix},$$

and again, the final effect (now α_2) can be reconstructed from the rest. This approach also rotates through our s seasonal effects, but now constrains the sum of the effects to be fixed at zero. This is how the function `dlm::dlmModSeas` constructs a seasonal model.

8.3.2 Fourier components

An alternative approach to modelling seasonal effects, which can be more parsimonious, is to use Fourier components. Start by assuming that we want to model our seasonal effects with a single sine wave of period s. Putting $\omega = 2\pi/s$, we can use the rotation matrix

$$\mathsf{G} = \begin{pmatrix} \cos\omega & \sin\omega \\ -\sin\omega & \cos\omega \end{pmatrix}$$

and the observation matrix $\mathsf{F} = (1, 0)$. Applying G repeatedly to a 2-vector $\mathbf{X}_t$ will give an oscillation of period s, with phase and amplitude determined by the components of $\mathbf{X}_t$. This gives a very parsimonious way of modelling seasonal effects, but may be too simple. We can add in other Fourier components by defining

$$\mathsf{H}_k = \begin{pmatrix} \cos k\omega & \sin k\omega \\ -\sin k\omega & \cos k\omega \end{pmatrix}, \quad k = 1, 2, \ldots,$$

(so $\mathsf{H}_k = \mathsf{G}^k$) and setting $\mathsf{F} = (1, 0, 1, 0, \ldots)$,

$$\mathsf{G} = \text{blockdiag}\{\mathsf{H}_1, \mathsf{H}_2, \ldots\}.$$

In principle we can go up to $k = s/2$. Then we will have s degrees of freedom, and we will essentially be just representing our seasonal effects by their discrete Fourier transform. Typically we will choose k somewhat smaller than this, omitting higher frequencies in order to obtain a smoother, more parsimonious representation. We can use a diagonal W matrix in order to allow the phase and amplitude of each harmonic to slowly evolve. This is how the `dlm::dlmModTrig` function constructs a seasonal model.

8.4 Model superposition

We can combine DLM models to make more complex DLMs in various ways. One useful approach is typically known as *model superposition*. Suppose that we have two DLM models with common observation dimension, m, and state dimensions $p_1,\ p_2$, respectively. If model i is $\{\mathsf{G}_{it}, \mathsf{F}_{it}, \mathsf{W}_{it}, \mathsf{V}_{it}\}$, then the "sum" of the two models can be defined by

$$\Big\{ \text{blockdiag}\{\mathsf{G}_{1t}, \mathsf{G}_{2t}\}, (\mathsf{F}_{1t}, \mathsf{F}_{2t}), \text{blockdiag}\{\mathsf{W}_{1t}, \mathsf{W}_{2t}\}, \mathsf{V}_{1t} + \mathsf{V}_{2t} \Big\},$$

having observation dimension m and state dimension $p_1 + p_2$. It is clear that this "sum" operation can be applied to more than two models with observation dimension m, and that the "sum" operation is associative, so DLM models of this form comprise a semigroup wrt this operation. Combining models in this way provides a convenient way of building models including both overall trends as well as seasonal effects. This sum operation can be used to combine DLM models in the `dlm` package by using the + operator.

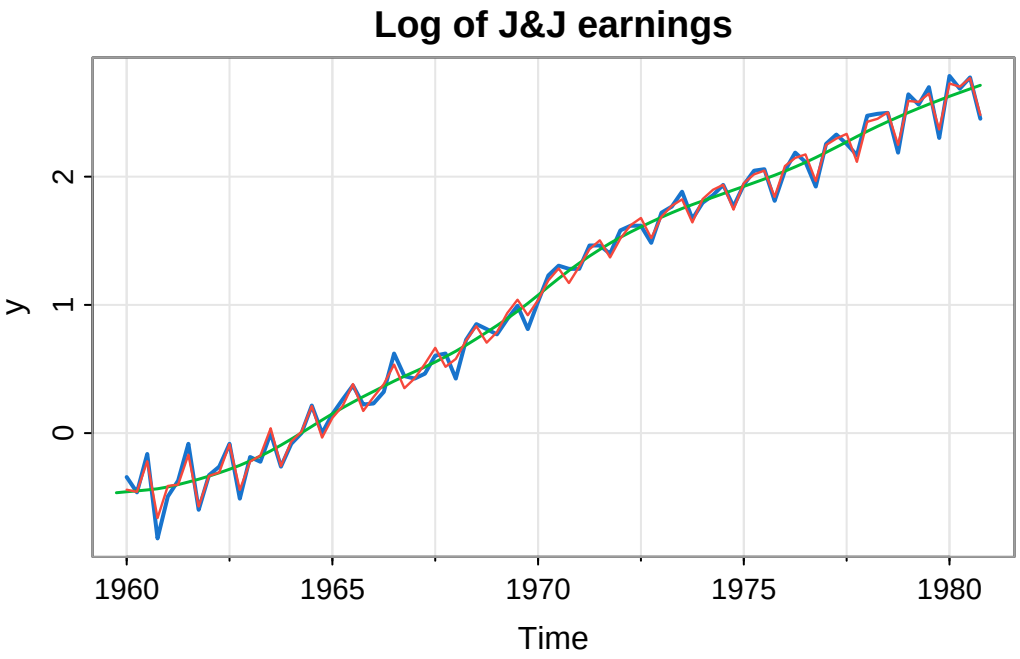
For our running example, we can fit a model including a locally linear trend and a quarterly seasonal effect as follows.

```
mod = dlmModPoly(2, dV=0.1^2, dW=c(0.01^2, 0.01^2)) +
       dlmModSeas(4, dV=0, dW=c(0.02^2, 0, 0))
mod
```

```
$FF
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    1    0    0

$V
     [,1]
[1,] 0.01

$GG
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0   -1   -1   -1
[4,]    0    0    1    0    0
[5,]    0    0    0    1    0

$W
      [,1]   [,2]   [,3] [,4] [,5]
[1,] 1e-04 0e+00 0e+00    0    0
[2,] 0e+00 1e-04 0e+00    0    0
[3,] 0e+00 0e+00 4e-04    0    0
```

```
[4,] 0e+00 0e+00 0e+00    0    0
[5,] 0e+00 0e+00 0e+00    0    0

$m0
[1] 0 0 0 0 0

$C0
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 1e+07 0e+00 0e+00 0e+00 0e+00
[2,] 0e+00 1e+07 0e+00 0e+00 0e+00
[3,] 0e+00 0e+00 1e+07 0e+00 0e+00
[4,] 0e+00 0e+00 0e+00 1e+07 0e+00
[5,] 0e+00 0e+00 0e+00 0e+00 1e+07
```

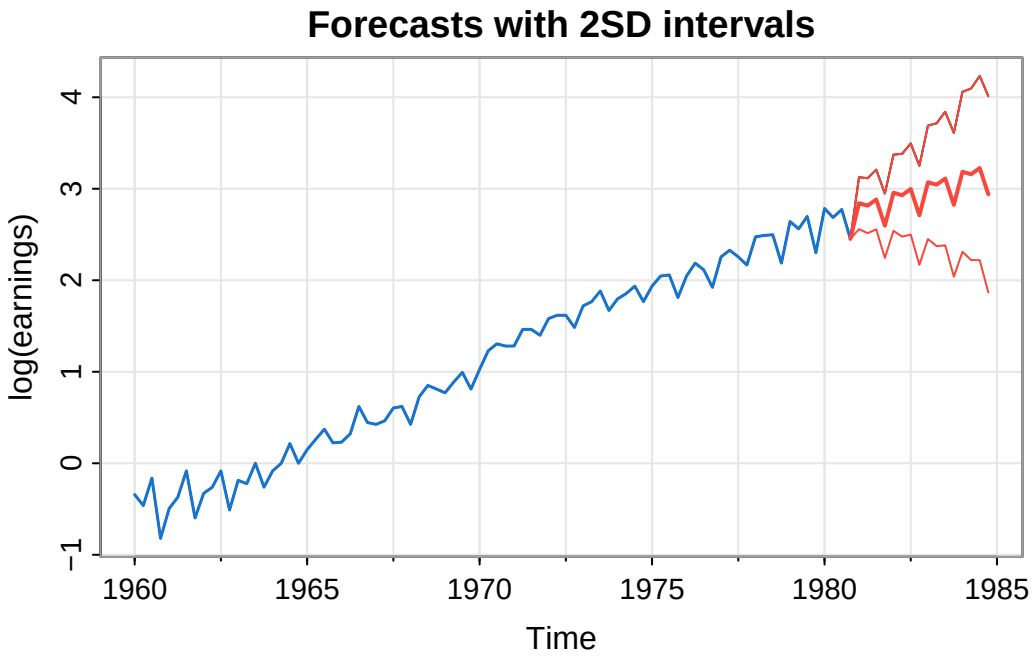
```
ss = dlmSmooth(y, mod) # smoothed states
so = ss$s[-1,] %*% t(mod$FF) # smoothed observations
so = ts(so, start=start(y), freq=frequency(y))
tsplot(y, col=4, lwd=2, main="Log of J&J earnings")
lines(ss$s[,1], col=3, lwd=1.5)
lines(so, col=2, lwd=1.2)
```



Once we are happy that the model is a good fit to the data, we can use it for forecasting.

```
fit = dlmFilter(y, mod)
fore = dlmForecast(fit, 16) # forecast 16 time points
pred = ts(c(tail(y, 1), fore$f),
  start=end(y), frequency=frequency(y))
upper = ts(c(tail(y, 1), fore$f + 2*sqrt(unlist(fore$Q))),
  start=end(y), frequency=frequency(y))
lower = ts(c(tail(y, 1), fore$f - 2*sqrt(unlist(fore$Q))),
  start=end(y), frequency=frequency(y))
all = ts(c(y, upper[-1]), start=start(y),
```

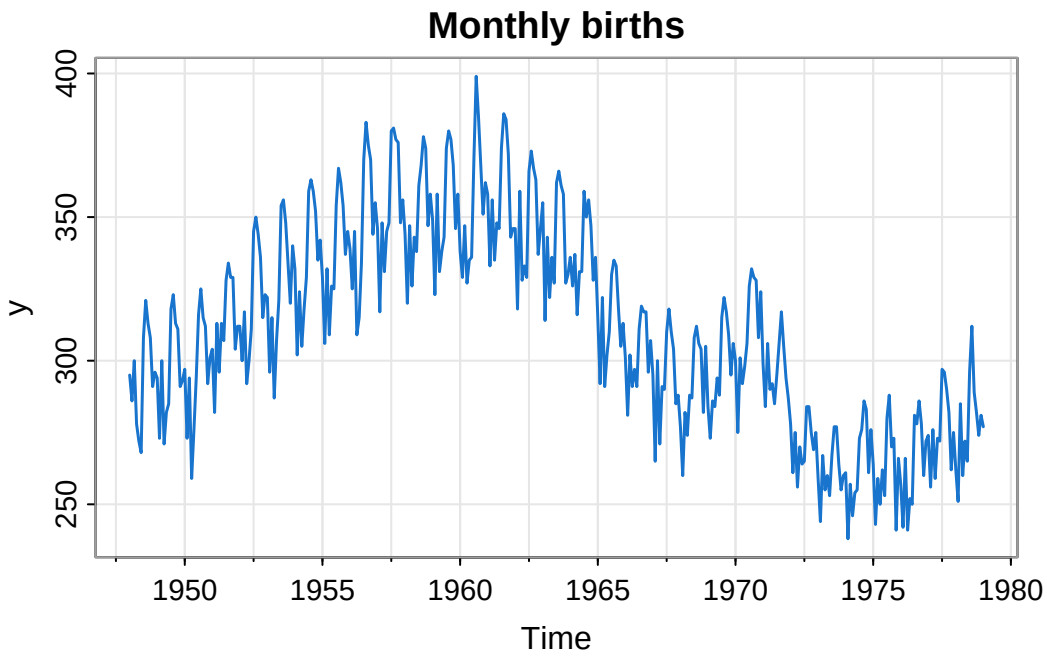
```
  frequency = frequency(y))
tsplot(all, ylab="log(earnings)",
  main="Forecasts with 2SD intervals")
lines(y, col=4, lwd=1.5)
lines(pred, col=2, lwd=2)
lines(upper, col=2)
lines(lower, col=2)
```



8.4.1 Example: monthly births

For an additional example, we will look at monthly live birth data for the US.

```
y = birth
tsplot(y, col=4, lwd=1.5, main="Monthly births")
```



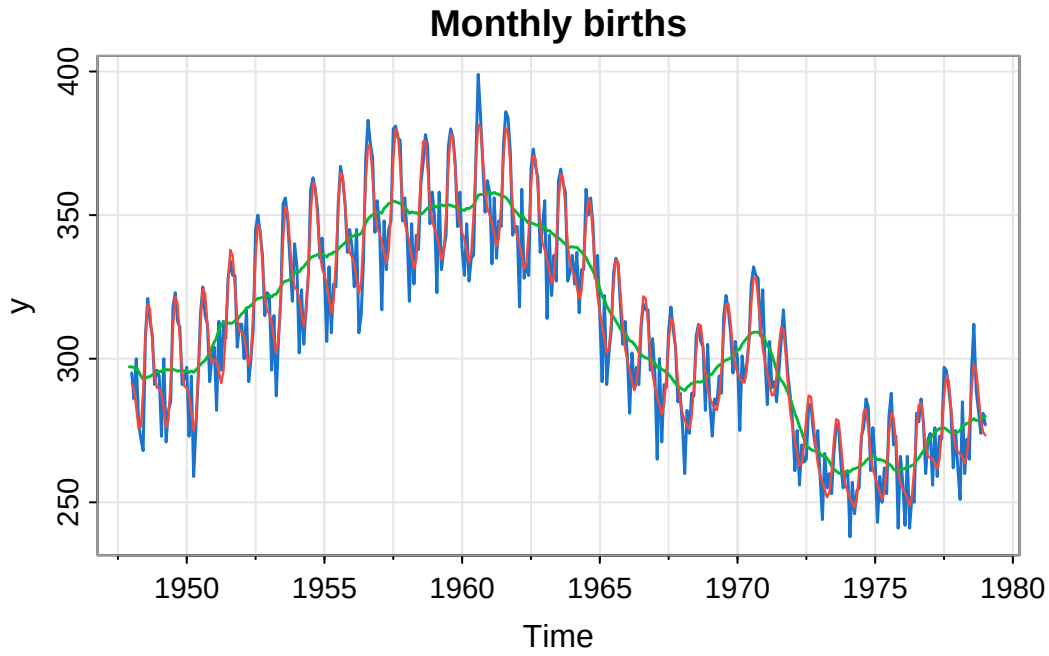
This seems to have a random walk nature, but with a strong seasonal component. We will assume that we do not want to use a full seasonal model, and instead use a seasonal model based on Fourier components with two harmonics. We may be unsure as to what variances to assume, and so want to estimate those using maximum likelihood. We could fit this using the `dlm` package as follows.

```
buildMod = function(lpar)
  dlmModPoly(1, dV=exp(lpar[1]), dW=exp(lpar[2])) +
    dlmModTrig(12, 2, dV=0, dW=rep(exp(lpar[3]), 4))
opt = dlmMLE(y, parm=log(c(100, 1, 1)),
        build=buildMod)
opt
```

```
$par
[1]  4.482990  1.925763 -3.228793

$value
[1] 1116.91

$counts
function gradient
      14       14

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
mod = buildMod(opt$par)
ss = dlmSmooth(y, mod)
so = ss$s[-1,] %*% t(mod$FF) # smoothed observations
so = ts(so, start(y), frequency = frequency(y))
```

```
tsplot(y, col=4, lwd=1.5, main="Monthly births")
lines(ss$s[,1], col=3, lwd=1.5)
lines(so, col=2, lwd=1.2)
```



The optimised parameters seem to correspond to a good fit to the data.

8.5 ARMA models in state space form

We have seen in previous chapters how ARMA models define a useful class of discrete time Gaussian processes that can effectively capture the kinds of auto-correlation structure that we often see in time series. They can be useful building blocks in DLM models. To use them in this context, we need to know how to represent them in state space form.

8.5.1 AR models

In fact, we have already seen a popular way to represent an AR(p) model as a VAR(1), and this is exactly what we need for state space modelling. So, to build a DLM representing noisy observation of a hidden AR(p) model we specify $\mathsf{F} = (1, 0, \ldots, 0)$,

$$\mathsf{G} = \begin{pmatrix} \phi_1 & \cdots & \phi_{p-1} & \phi_p \\ 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix},$$

$\mathsf{W} = \mathrm{diag}(\sigma^2, 0, \ldots, 0)$, $\mathsf{V} = (v)$. As a reminder of why this works, we see that if we define $\mathbf{X}_t = (X_t, X_{t-1}, \ldots, X_{t-p})^\top$, where X_t is our AR(p) process, then

$$\mathsf{G}\mathbf{X}_{t-1} + \begin{pmatrix} \varepsilon_t \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \mathbf{X}_t,$$

and so the VAR(1) update preserves the stationary distribution of our AR(p) model. However, this VAR(1) representation of an AR(p) model is by no means unique. In fact, it turns out that replacing the above G with $\mathsf{G}^\top$ gives a VAR(1) model which also has the required AR(p) model as the marginal distribution of the first component. This is not so intuitive, but generalises more easily to the ARMA case, so it is worth understanding. For this case we instead define the state vector to be

$$\mathbf{X}_t^\star = \begin{pmatrix} X_t \\ \phi_2 X_{t-1} + \phi_3 X_{t-2} + \ldots + \phi_p X_{t-p+1} \\ \vdots \\ \phi_{p-1} X_{t-1} + \phi_p X_{t-2} \\ \phi_p X_{t-1} \end{pmatrix},$$

where X_t is our AR(p) process of interest, and note that only the first component of this state vector has the distribution we require. But with this definition we have

$$\mathsf{G}^\top \mathbf{X}_{t-1}^\star + \begin{pmatrix} \varepsilon_t \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \mathbf{X}_t^\star,$$

and so this VAR(1) update preserves the distribution of this state vector. Consequently, the first component of the VAR(1) model defined this way will be marginally our AR(p) model of interest. This representation is commonly encountered in practice, since the same approach works for ARMA models, where it is necessary to carry information about "old" errors in the state vector.

8.5.2 ARMA models

Representing an arbitrary ARMA model as a VAR(1) is also possible, and again the representation is not unique. However, the second approach that we used for AR(p) models can be adapted very easily to the ARMA case. Consider first an ARMA($p, p-1$) model, so that we have one fewer MA coefficients than AR coefficients. Using the $p \times p$ auto-regressive matrix

$$\mathsf{G} = \begin{pmatrix} \phi_1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{p-1} & 0 & \ddots & 1 \\ \phi_p & 0 & \cdots & 0 \end{pmatrix},$$

together with the error vector

$$\boldsymbol{\varepsilon}_t = \begin{pmatrix} 1 \\ \theta_1 \\ \vdots \\ \theta_{p-1} \end{pmatrix} \varepsilon_t,$$

leads to a hidden state vector with first component marginally the ARMA($p, p-1$) process of interest. Note that the error covariance matrix is

$$\mathsf{W} = \begin{pmatrix} 1 \\ \theta_1 \\ \vdots \\ \theta_{p-1} \end{pmatrix} \begin{pmatrix} 1 \\ \theta_1 \\ \vdots \\ \theta_{p-1} \end{pmatrix}^\top \sigma^2.$$

To understand why this works, define our state vector to be

$$\mathbf{X}_t = \begin{pmatrix} X_t \\ \phi_2 X_{t-1} + \phi_3 X_{t-2} + \ldots + \phi_p X_{t-p+1} + \theta_1 \varepsilon_t + \cdots + \theta_{p-1} \varepsilon_{t-p+2} \\ \vdots \\ \phi_{p-1} X_{t-1} + \phi_p X_{t-2} + \theta_{p-2} \varepsilon_t + \theta_{p-1} \varepsilon_{t-1} \\ \phi_p X_{t-1} + \theta_{p-1} \varepsilon_t \end{pmatrix}.$$

We can then directly verify that

$$\mathsf{G}\mathbf{X}_{t-1} + \boldsymbol{\varepsilon}_t = \mathbf{X}_t,$$

and so this VAR(1) update preserves the distribution of our state vector. Consequently, the first component is marginally ARMA, and so using $\mathsf{F} = (1, 0, \cdots, 0)$ will pick out this component for incorporation into the model of observation process.

We have considered the case of ARMA$(p, p-1)$, but note that any ARMA(p, q) model can be represented as an ARMA$(p^\star, p^\star - 1)$ model by choosing $p^\star = \max\{p, q+1\}$ and padding coefficient vectors with zeros as required. This is exactly the approach used by the function `dlm::dlmModARMA`.

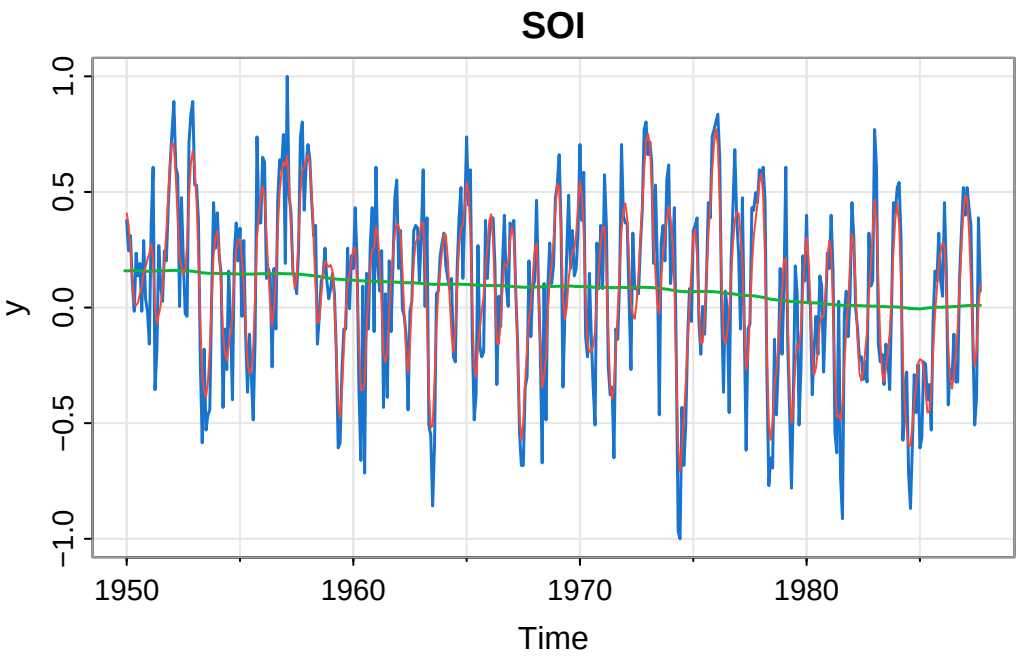
8.5.3 Example: SOI

Let us return to the SOI example we looked at at the end of Chapter 7. We can model this using a locally constant model, but also with a seasonal effect. Further, we can investigate any residual correlation structure by including an AR(2) component. We can fit this as follows.

```
y = soi
buildMod = function(param)
  dlmModPoly(1, dV=exp(param[1]), dW=exp(param[2])) +
    dlmModARMA(ar=c(param[3], param[4]),
     sigma2=exp(param[5]), dV=0) +
    dlmModTrig(12, 2, dV=0, dW=rep(exp(param[6]), 4))
opt = dlmMLE(y, parm=c(log(0.1^2), log(0.01^2),
  0.2, 0.1, log(0.1^2), log(0.01^2)), build=buildMod)
opt
```

```
$par
[1] -3.100868e+00 -9.242014e+00  8.792923e-01 -7.119263e-06 -4.572246e+00
[6] -1.010190e+01

$value
[1] -310.9818

$counts
function gradient
      51       51

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
mod = buildMod(opt$par)
ss = dlmSmooth(y, mod)
so = ss$s[-1,] %*% t(mod$FF) # smoothed observations
so = ts(so, start(y), freq=frequency(y))
tsplot(y, col=4, lwd=1.5, main="SOI")
lines(ss$s[,1], col=3, lwd=1.5)
lines(so, col=2)
```



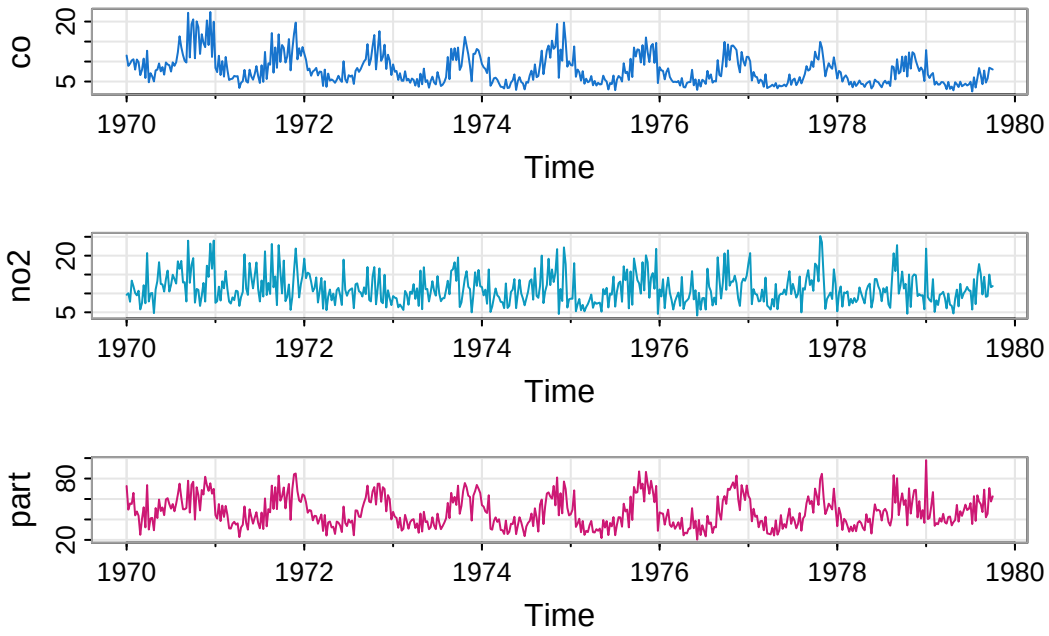
This looks to be a good fit, suggesting a gradual slow decreasing trend, but inspection of the optimised parameters (in particular `param[5]`, the log of σ^2 in the ARMA component), suggests that the AR(2) component is not necessary.

8.6 Multivariate models

The DLM formalism introduced in Chapter 7 covers multivariate observations of dimension m. Each of the m rows of F determines how the state vector is transformed into a component of the observation vector. However, the process of building an appropriate model for multivariate data is considerably more challenging than for univariate time series. Here we will give a brief introduction to some of the most important concepts. We will then revisit the topic later in the context of spatio-temporal data, which can often be sensibly regarded as a multivariate time series with special (spatial) structure.

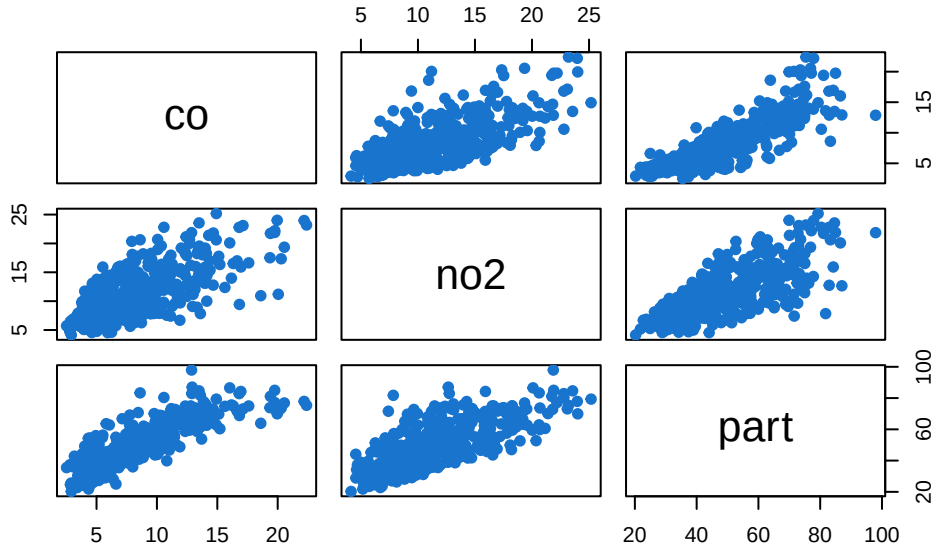
For our running example we will use part of the `astsa::lap` dataset. This contains weekly data on various pollutant levels, in addition to some mortality data. To keep things simple, we will just focus on three (correlated) pollution variables, `co` (Carbon monoxide), `no2` (Nitrogen dioxide) and `part` (Particulate matter).

```
pol = lap[, c(6, 8, 11)]
tsplot(pol, col=4:6)
```



We see that each component is seasonal, with peaks occurring at similar times of the year. This will certainly lead to associate between the component series, as can be confirmed with a pairs plot.

```
pairs(pol, col=4, pch=19)
```



What is much less obvious at this stage is whether the association is entirely due to the seasonality, or whether there is residual correlation between the series even after accounting for other factors. We will need to carefully model the data in order to obtain a satisfactory understanding of this issue.

We will begin our modelling process by developing appropriate univariate models for each component, and then think about how to combine them into a multivariate model in an appropriate way. From a basic eye-balling of the data, it appears that each series can probably be reasonably well described by a locally constant model with a seasonal component that isn't too complicated, so may be reasonably described with

two Fourier components. However, the time series all look different, so each component will need to be parameterised separately. A basic model building function is given below.

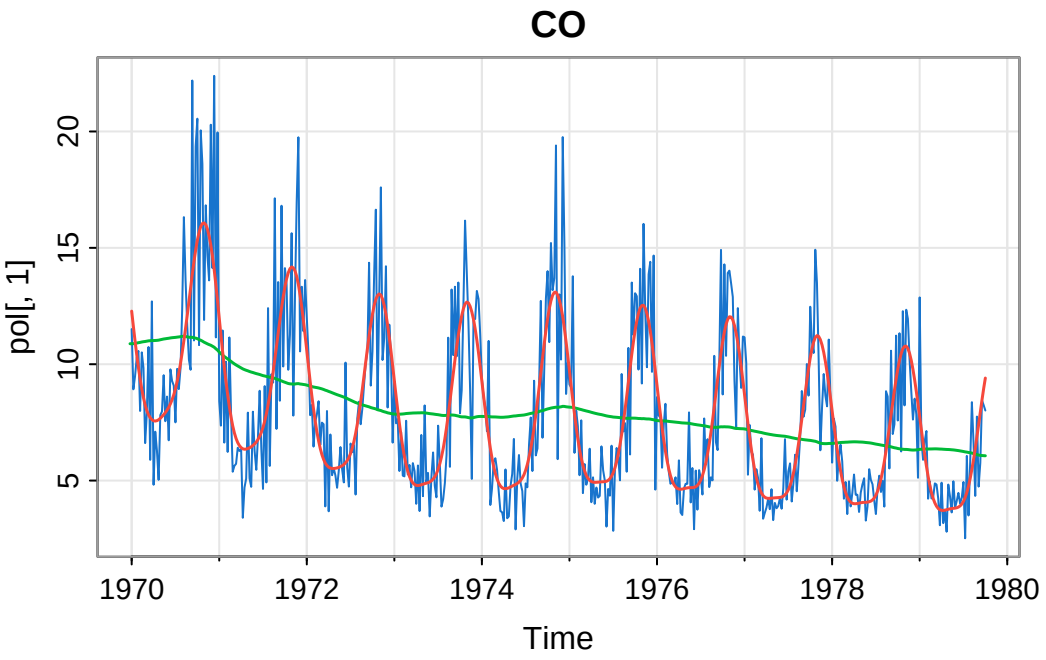
```
buildMod = function(lpar) {
    par = exp(lpar)
    dlmModPoly(1, dV=par[1], dW=par[2]) +
        dlmModTrig(52, 2, dV=0, dW=rep(par[3], 4))
}
```

We can use this function to parameterise each component in turn. We will start with carbon monoxide.

```
opt = dlmMLE(pol[,1], parm=log(c(10, 0.1, 0.01)),
             build=buildMod)
print(exp(opt$par))
```

```
[1] 4.4767675057 0.0135413012 0.0004923974
```

```
p1Mod = buildMod(opt$par)
ss = dlmSmooth(pol[,1], p1Mod) # smoothed states
so = ss$s[-1,] %*% t(p1Mod$FF) # smoothed obs
so = ts(so, start(pol[,1]), freq=frequency(pol[,1]))
tsplot(pol[,1], col=4, main="CO")
lines(ss$s[,1], col=3, lwd=1.5) # current level
lines(so, col=2, lwd=1.5)
```




```
print(dlmLL(pol[,1], p1Mod)) # Negative log-likelihood for this fit
```

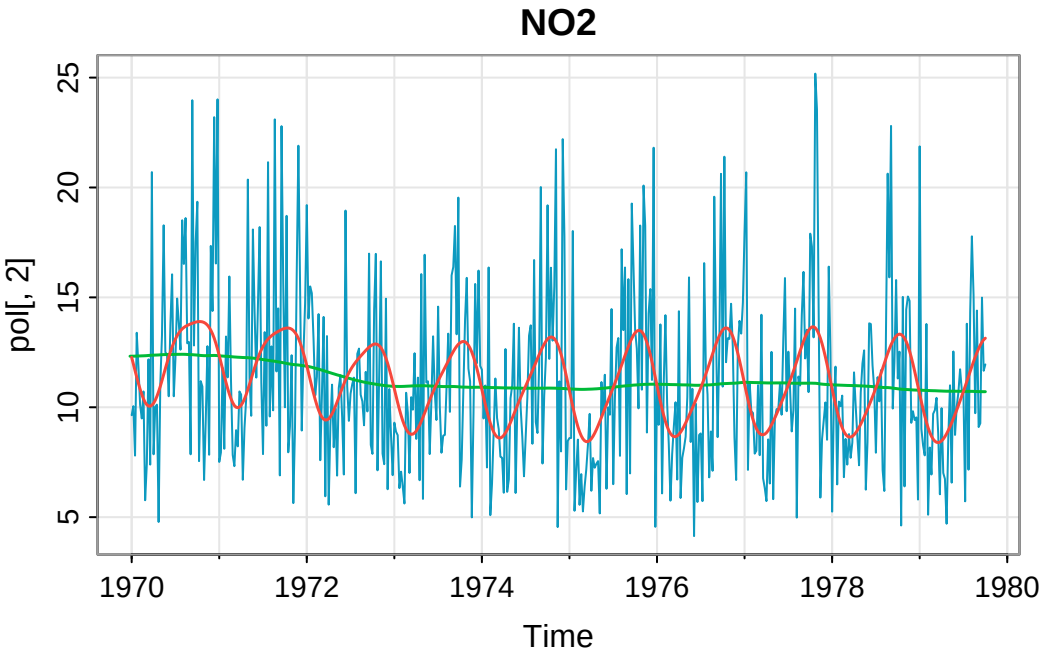
```
[1] 697.7364
```

The `dlmMLE` function seems to have found a good fit for this component. We can do the same for nitrogen dioxide.

```
opt = dlmMLE(pol[,2], parm=log(c(10, 0.1, 0.01)),
             build=buildMod)
print(exp(opt$par))
```

```
[1] 1.387212e+01 6.254836e-03 8.642782e-04
```

```
p2Mod = buildMod(opt$par)
ss = dlmSmooth(pol[,2], p2Mod) # smoothed states
so = ss$s[-1,] %*% t(p2Mod$FF) # smoothed obs
so = ts(so, start(pol[,2]), freq=frequency(pol[,2]))
tsplot(pol[,2], col=5, main="NO2")
lines(ss$s[,1], col=3, lwd=1.5)
lines(so, col=2, lwd=1.5)
```




```
print(dlmLL(pol[,2], p2Mod))
```

```
[1] 973.3261
```

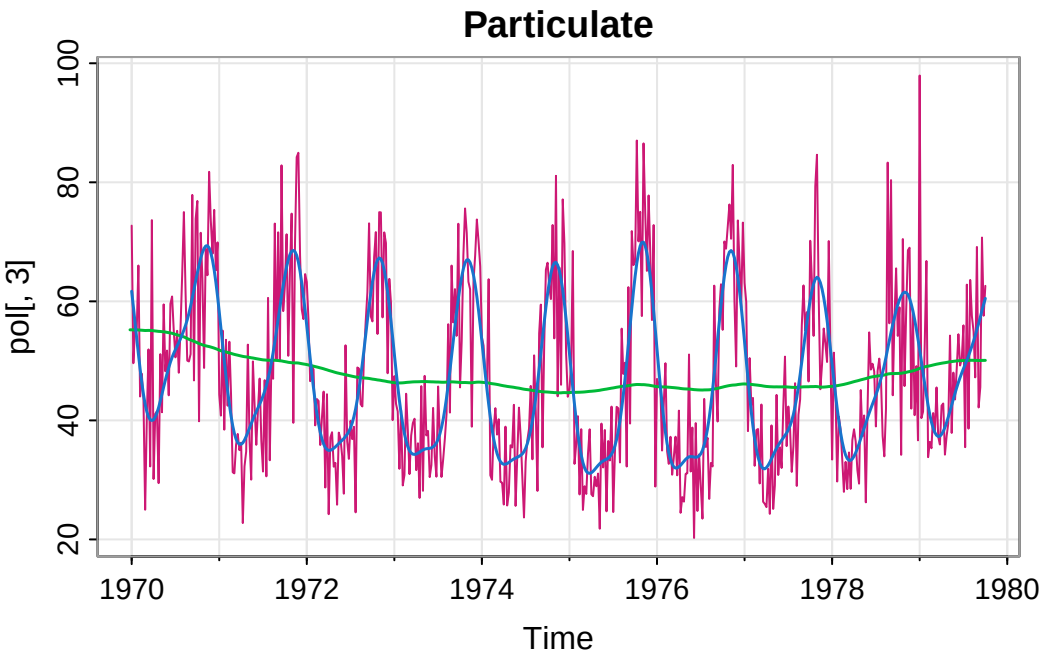
This fit also seems fine. Finally we repeat for particulate matter.

```
opt = dlmMLE(pol[,3], parm=log(c(10, 0.1, 0.01)),
             build=buildMod)
print(exp(opt$par))
```

```
[1] 90.65737051  0.14812352  0.07025641
```

```
p3Mod = buildMod(opt$par)
ss = dlmSmooth(pol[,3], p3Mod) # smoothed states
so = ss$s[-1,] %*% t(p3Mod$FF) # smoothed obs
```

```
so = ts(so, start(pol[,3]), freq=frequency(pol[,3]))
tsplot(pol[,3], col=6, main="Particulate")
lines(ss$s[,1], col=3, lwd=1.5)
lines(so, col=4, lwd=1.5)
```




```
print(dlmLL(pol[,3], p3Mod))
```

```
[1] 1460.519
```

This also seems fine. So, we now have three perfectly reasonable univariate models. If we believe that all of the series are truly independent of one another then this is fine. However, if we think that there may be dependencies and correlations between the components that are not accounted for by fitting univariate models independently, then we need a proper multivariate DLM for the 3-dimensional observation vector.

8.6.1 Model concatenation

There is an obvious way to glue together independent DLMs to form a combined DLM that is entirely equivalent. The operation is quite similar to the model superposition operation considered in Section 8.4, but importantly different. Here we suppose that we have two models. Model i (for $i = 1, 2$) has state dimension p_i, observation dimension m_i and is defined by $\{\mathsf{G}_{it}, \mathsf{F}_{it}, \mathsf{W}_{it}, \mathsf{V}_{it}\}$. Their *concatenation* or *outer sum* is a model with state dimension $p_1 + p_2$ and observation dimension $m_1 + m_2$ defined by

$$\Big\{\text{blockdiag}\{\mathsf{G}_{1t}, \mathsf{G}_{2t}\}, \text{blockdiag}\{\mathsf{F}_{1t}, \mathsf{F}_{2t}\}, \text{blockdiag}\{\mathsf{W}_{1t}, \mathsf{W}_{2t}\}, \text{blockdiag}\{\mathsf{V}_{1t}, \mathsf{V}_{2t}\}\Big\}$$

Note that the component models are entirely independent of one another, so this model is entirely equivalent to the separate models (and hence not intrinsically better). Again, this operation can clearly be repeated to combine multiple models, and the operation is clearly associative.

This outer sum is implemented in the `dlm` package using the `%+%` operator.

```
## Combined multivariate model
polMod = p1Mod %+% p2Mod %+% p3Mod
print(polMod$F)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    1    1    0    1    0    0    0    0    0     0     0     0     0     0
[2,]    0    0    0    0    0    1    1    0    1     0     0     0     0     0
[3,]    0    0    0    0    0    0    0    0    0     0     1     1     0     1
     [,15]
[1,]     0
[2,]     0
[3,]     0
```

```
print(polMod$V)
```

```
          [,1]     [,2]     [,3]
[1,] 4.476768  0.00000  0.00000
[2,] 0.000000 13.87212  0.00000
[3,] 0.000000  0.00000 90.65737
```

```
print(dlmLL(pol, polMod))
```

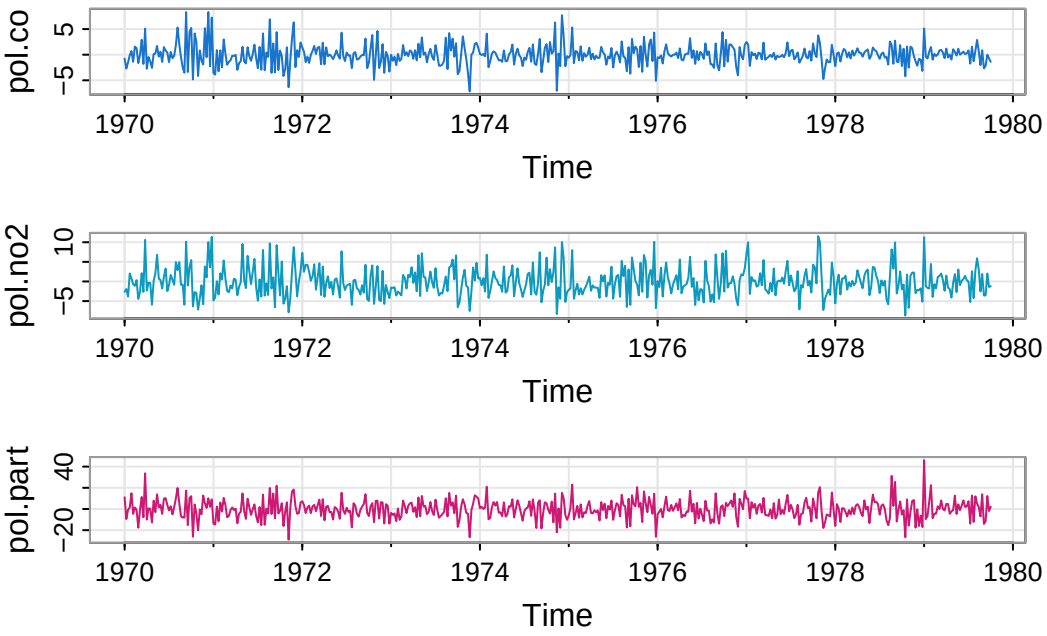
```
[1] 3131.581
```

Computing the (negative) log-likelihood of the combined model gives us the sum of the log-likelihoods of the component models, as we would expect. As discussed, this independent component model is equivalent to the separate univariate models, so not yet any better. However, this is a good "straw man" multivariate model that we can use as a starting point for building a better, more fundamentally multivariate model.

8.6.2 Multivariate dependence

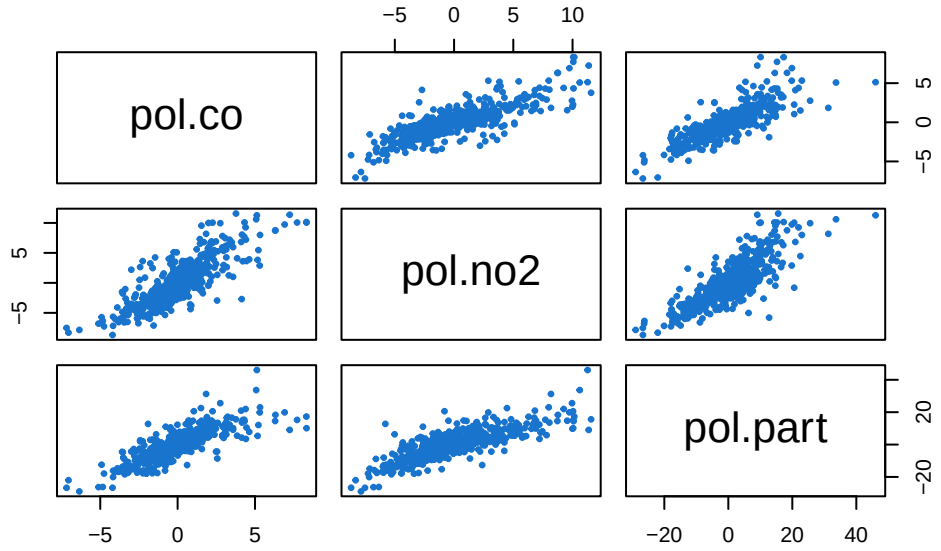
Now that we have a reasonable model based on an assumption of independence between the components, we can investigate whether there appear to be any dependencies between the components that are not just indirect associations due to similar patterns of seasonality. We begin by looking at the "residuals" for the base model. We can form residuals by subtracting the fitted smoothed observations from the actual observations.

```
ss = dlmSmooth(pol, polMod)
so = ss$s[-1,] %*% t(polMod$FF)
so = ts(so, start(pol), freq=frequency(pol))
resid = pol - so
tsplot(resid, col=4:6)
```



We see that each component residual time series looks fairly random, suggesting that our univariate models are all reasonable fits. But are there any cross-correlations between the residual components?

```
pairs(resid, col=4, pch=19, cex=0.5)
```



Clearly there are. This is strongly suggestive that our base model could be improved by building some kind of dependence between the components. There are many ways that this could be done. The simplest approach may be to allow some correlation in the V or W matrices. But in some cases in can also make sense to couple the components by introducing additional non-zero off-diagonal elements into the G matrix. Exploring these things in detail is beyond the scope of this course. For now we will just see how allowing correlations in the V matrix can greatly improve the model fit. Comparing the covariance matrix of the residuals with the current fitted V matrix

```
  print(cov(resid))
```

```
            pol.co   pol.no2 pol.part
pol.co    4.288101  6.225905 15.40535
pol.no2   6.225905 13.535552 27.90618
pol.part 15.405352 27.906175 85.62305
```

```
  print(polMod$V)
```

```
         [,1]     [,2]     [,3]
[1,] 4.476768  0.00000  0.00000
[2,] 0.000000 13.87212  0.00000
[3,] 0.000000  0.00000 90.65737
```

shows that while the diagonal is similar, the residuals have strong cross-correlations that are currently absent from our model. We could directly optimise a model with a parameterised covariance matrix, but this requires a little care due to the positive-definiteness constraint. For illustrative purposes, we can just replace the existing V matrix with the empirical covariance matrix of the residuals.

```
  polMod$V = cov(resid)
  print(dlmLL(pol, polMod))
```

```
[1] 2523.091
```

We see that this leads to a massive improvement in the model likelihood (the negative log-likelihood is much smaller). So this is now a much better fitting model, and should therefore lead to improved smoothing and forecast distributions. Of course, other model refinements could also help, but we leave investigation of this to the interested reader.

As previously discussed, multivariate time series modelling is a big topic, and good models usually rely on very specific features of the multivariate data under consideration. Spatio-temporal data is a good example of multivariate time series with special structure, and we will see how this can be exploited at the end of the course, once we know a bit about spatial modelling.

9 Introduction to spatial data

9.1 Introduction

We now turn attention away from time series to spatial data. We will see that there are many similarities, but also some important differences. There a many different kinds of spatial data. Each kind may require a different modelling approach, and may be subject to different inferential questions. We will concentrate on the three or four most commonly encountered types of spatial data.

9.2 Point referenced data and geostatistics

Just as time series consist of observations indexed by time, spatial datasets typically consist of observations indexed by a point in space. So we can write $z(\mathbf{s})$ for the value of an observation at location $\mathbf{s}$. We will mainly consider univariate real-valued data, so then $z(\mathbf{s}) \in \mathcal{Z} \subset \mathbb{R}$, but multivariate extensions are possible. The spatial location belongs to the spatial domain, $\mathcal{D}$, so $\mathbf{s} \in \mathcal{D}$. This can typically be thought of as a point in 3-dimensional space, $\mathcal{D} \subset \mathbb{R}^3$, but in practice most spatial data sets live in a two-dimensional sub-space, so we will typically have $\mathcal{D} \subset \mathbb{R}^2$. We will also consider 1-d spatial data for illustrative purposes (and for linking back to time series models), so then $\mathcal{D} \subset \mathbb{R}$. In this course we will assume that $\mathcal{D}$ is a subset of a Euclidean vector space. However, this is a big assumption, since many spatial data sets are geospatial, and indexed by points on the surface of the Earth, which is most definitely not a Euclidean space! However, we will assume that the data sets that we are dealing with are over a sufficiently small area that projection onto $\mathbb{R}^2$ is reasonable. A dataset consisting of n observations can typically be written

$$\{(\mathbf{s}_i, z(\mathbf{s}_i)) \in \mathcal{D} \times \mathcal{Z} | i = 1, 2, \ldots, n\},$$

and the observed value $z(\mathbf{s}_i)$ will often be written z_i. In the context of geostatistics, the spatial locations are not considered to be random. They are typically irregularly distributed, but essentially arbitrary, and perhaps determined by the experimenter, either informally, or by some experimental design (which may be random, but the randomness in the design is not of inferential interest). It is the observations themselves that are typically considered to arise from some kind of random process. That is, the measurements $z(\mathbf{s})$ are considered observations of the random variable $Z(\mathbf{s})$, where $\{Z(\mathbf{s}) | \mathbf{s} \in \mathcal{D}\})$ is considered to be a *random function* (typically referred to as a random field or stochastic process) $\mathcal{D} \to \mathcal{Z}$. The observations at different locations will typically not be iid (or exchangeable), since observations at nearby locations will typically be more highly correlated than observations at distant locations. In the context of geostatistics, this fact is often referred to as Tobler's first law of geography: "*everything is related to everything else, but near things are more related than distant things*".

The typical problem of inferential interest is interpolation (sometimes combined with smoothing), where data at the sites for which observations are available are used to make a prediction for the values that would be observed at sites for which measurements have not been made. That is, a prediction is wanted for $Z(\mathbf{s})$, the unobserved value at an arbitrarily chosen site $\mathbf{s} \in \mathcal{D}$, $\mathbf{s} \notin \{\mathbf{s}_i | i = 1, 2, \ldots, n\}$.

9.2.1 The `meuse` dataset

There are many packages for working with spatial data in R. In the past, `sp` was the package used to define spatial datasets, and many packages built on top of this to add additional functionality. These days, `sf`

(simple features) is the preferred base package for spatial data, and new packages for spatial analysis typically build on top of this. However, many useful packages still exist which rely on `sp`, so it is not possible to completely abandon this package yet. Recall that the CRAN spatial task view gives an invaluable summary of spatial packages in R. We will start by looking at a typical geostatistical dataset that is included in the `sp` package.

```
library(sp)

data(meuse)
class(meuse)
```

```
[1] "data.frame"
```

```
dim(meuse)
```

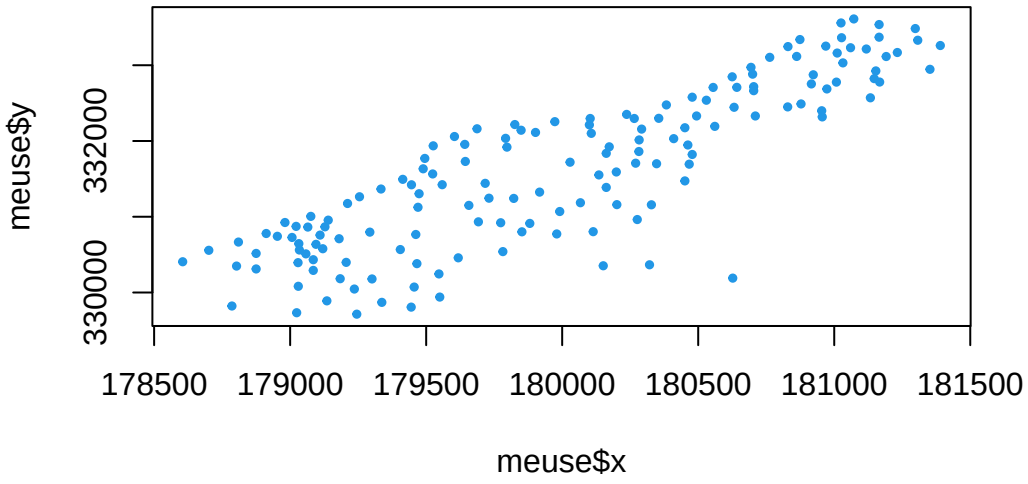
```
[1] 155  14
```

```
head(meuse[,1:10])
```

```
       x      y cadmium copper lead zinc  elev       dist   om ffreq
1 181072 333611    11.7     85  299 1022 7.909 0.00135803 13.6     1
2 181025 333558     8.6     81  277 1141 6.983 0.01222430 14.0     1
3 181165 333537     6.5     68  199  640 7.800 0.10302900 13.0     1
4 181298 333484     2.6     81  116  257 7.655 0.19009400  8.0     1
5 181307 333330     2.8     48  117  269 7.480 0.27709000  8.7     1
6 181390 333260     3.0     61  137  281 7.791 0.36406700  7.8     1
```

Use `?meuse` to find out more about this data. We see that the first two columns of this data set give x and y coordinates of the spatial locations of the observervations, with respect to some coordinate system (discussed further below). There are then some columns representing some measurements, and some additional covariate information that we will ignore. We will focus mainly on the `zinc` column - a measurement of zinc concentration in the ground at each location. First, let's just plot the spatial locations of the observations.

```
plot(meuse$x, meuse$y, pch=19, cex=0.5, col=4)
```

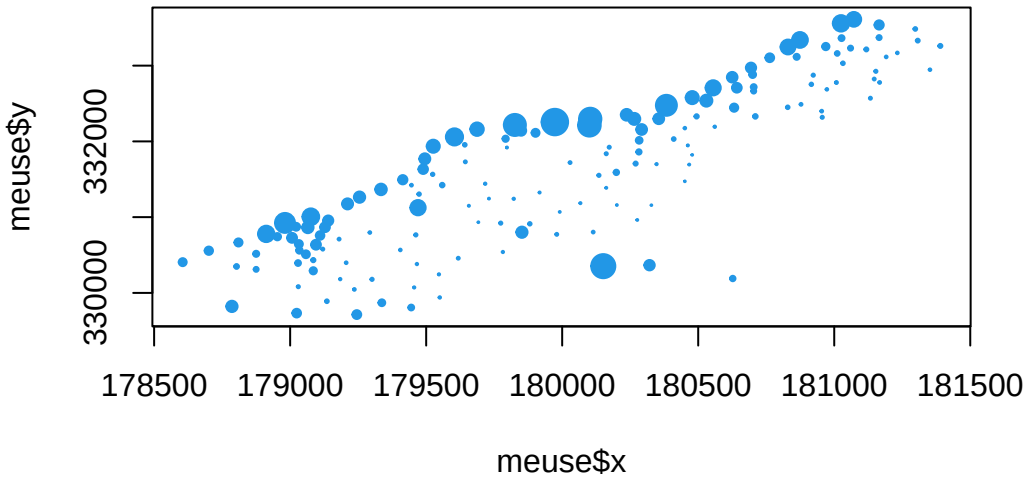


The measurements all lie to the east of the Meuse river (the Dutch side) next to the Dutch town of Stein, which is why there are no measurements in the top left of the plot. By looking at the range of zinc values, we can modify the plot to show levels of zinc concentration by the size of the plotted point.

```
summary(meuse$zinc)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  113.0   198.0   326.0   469.7   674.5  1839.0
```

```
plot(meuse$x, meuse$y, pch=19, cex=meuse$zinc/1000, col=4)
```



We can now clearly see that zinc concentrations are typically higher close to the river, and lower further away, but with some variability.

Using the `sf` package, we can convert this data frame into an `sf` object. To do this, we will use the function `st_as_sf`. We will need to tell this function which columns contain the x and y coordinates, and also, what the coordinates *mean* (ie. what units they are in, and what they are measured with respect to). The help information (`?meuse`) confirms that the coordinates are RDH (a Dutch coordinate system). These have a coordinate reference system (CRS) code of EPSG:28992.

```
library(sf)
```

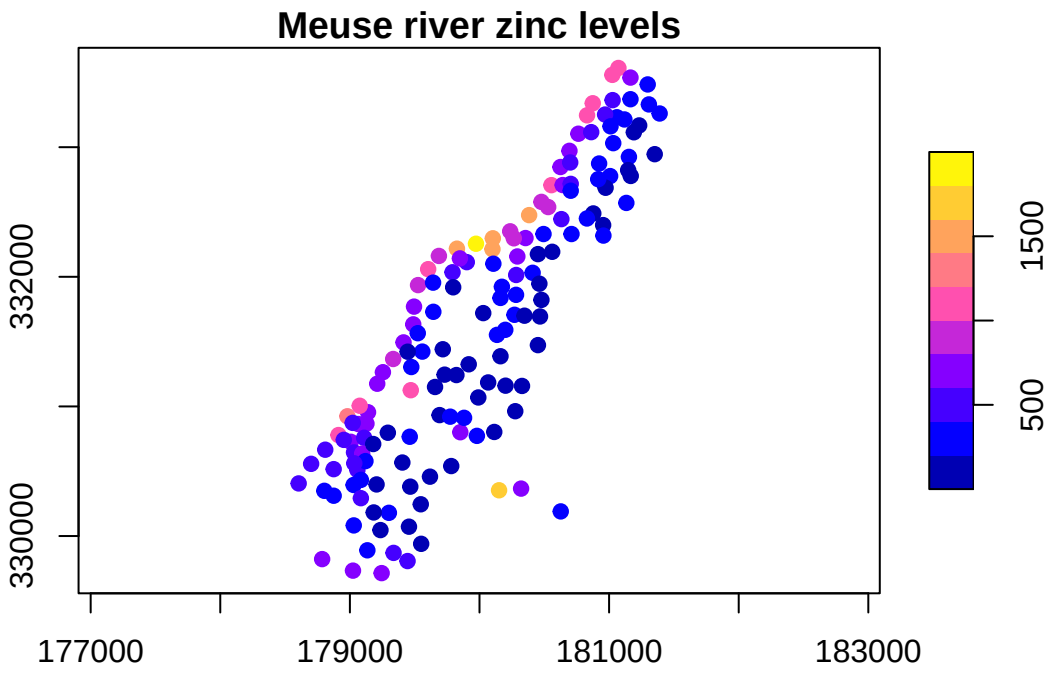
```
Linking to GEOS 3.12.1, GDAL 3.8.4, PROJ 9.4.0; sf_use_s2() is TRUE
```

```
meuse_sf = st_as_sf(meuse, coords=c("x", "y"), crs=28992)
class(meuse_sf)
```

```
[1] "sf"         "data.frame"
```

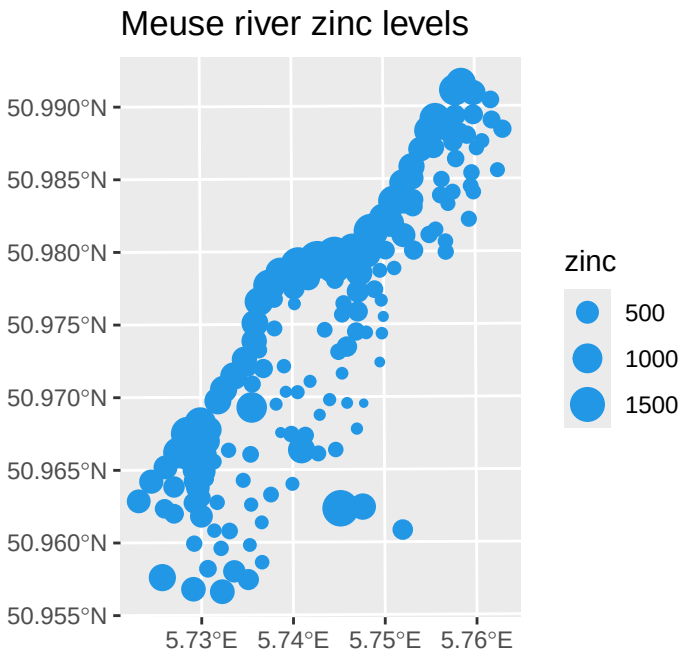
So `meuse_sf` is an object of class `sf` in addition to `data.frame`. We can plot objects of class `sf`.

```
plot(meuse_sf[,"zinc"], pch=19,
     main="Meuse river zinc levels", axes=TRUE)
```

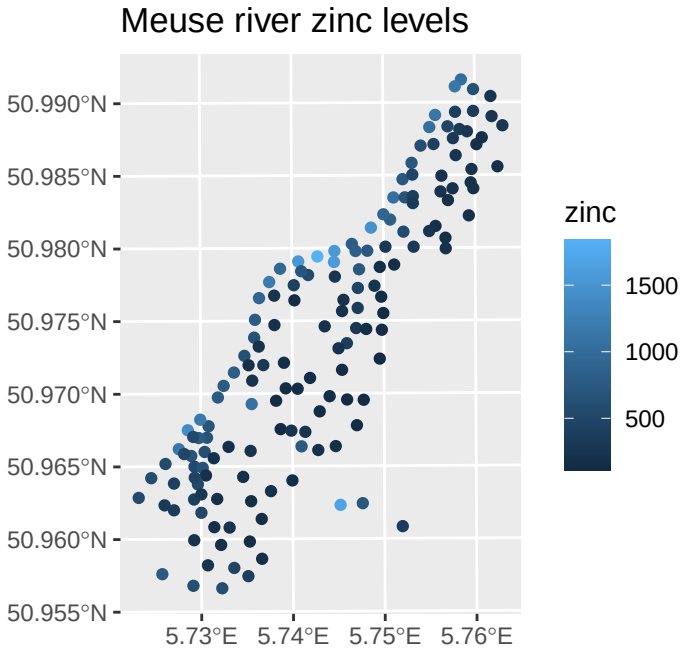


Here, a colour gradient is used to indicate zinc levels at each spatial location. In this course we will mainly use base R graphics, but `sf` objects work well with `ggplot2` (and the `tidyverse` more generally).

```
library(ggplot2)

ggplot(meuse_sf) +
    geom_sf(aes(cex=zinc), col=4) +
    ggtitle("Meuse river zinc levels")
```

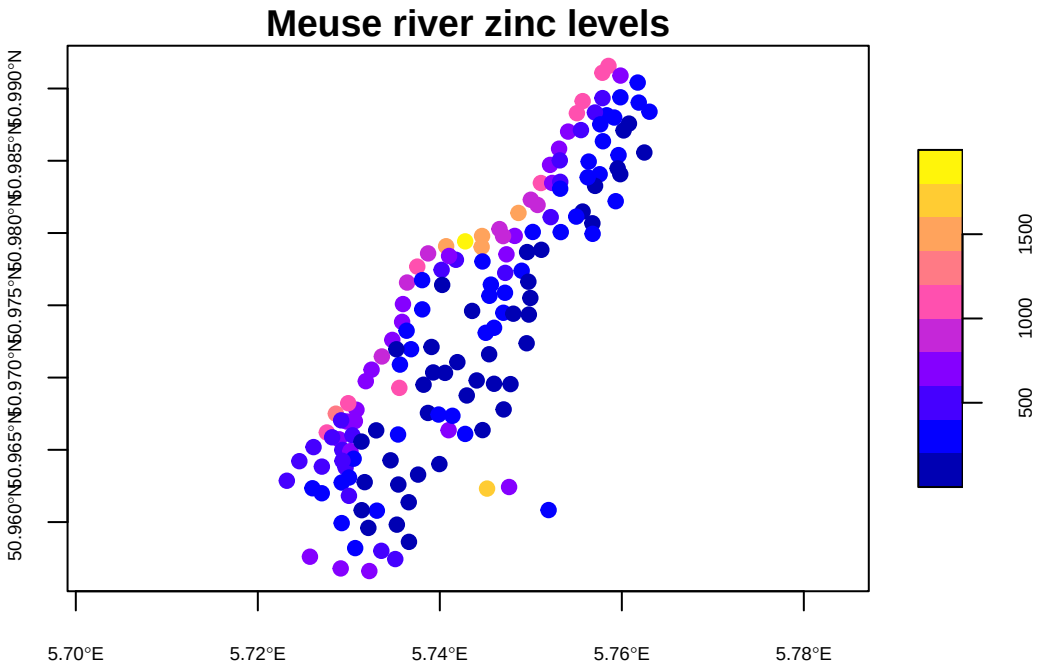



```
ggplot(meuse_sf) +
    geom_sf(aes(colour=zinc)) +
    ggtitle("Meuse river zinc levels")
```



Note that the axes on the `ggplot` graphics show the spatial location in longitude and latitude. This is possible because the CRS allows mapping of coordinates to other coordinate systems. The function `st_transform` can be used to remap the coordinates in an `sf` object.

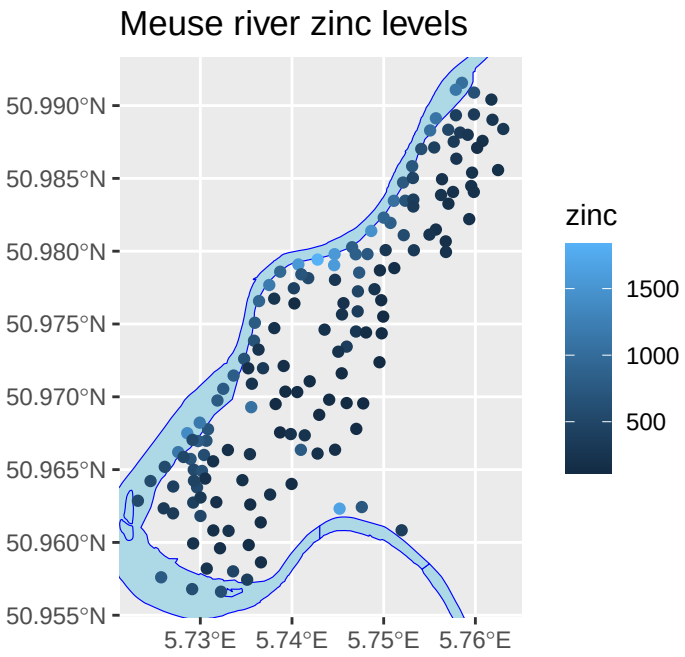
```
meuse_sf2 = st_transform(meuse_sf, "EPSG:4326")
plot(meuse_sf2[,"zinc"], pch=19,
     main="Meuse river zinc levels", axes=TRUE, cex.axis=0.6)
```



Note that 4326 is the EPSG code for WGS84, the current standard reference coordinate system for Earth (based on a 2-d longitude and latitude projection).

It would be helpful to see the data we have overlaid on a plot showing the course of the Meuse river. We can do that using the `osmdata` package to query OpenStreetMap data. Don't worry about how this code works.

```
library(osmdata)
meuse_bb = st_bbox(meuse_sf2) # bounding box in long/lat
meuse_river = meuse_bb %>% opq() %>%
    add_osm_feature(key="water", value="river") %>%
    osmdata_sf()

ggplot() +
    geom_sf(data = meuse_river$osm_polygons,
            colour = "blue",
            fill = "lightblue") +
    geom_sf(data = meuse_sf2, mapping=aes(colour=zinc)) +
    coord_sf(xlim=c(meuse_bb[1], meuse_bb[3]),
             ylim=c(meuse_bb[2], meuse_bb[4])) +
    ggtitle("Meuse river zinc levels")
```



The standard statistical problem associated with data of this sort is to produce a smooth image containing a prediction of how the zinc levels vary across the entire spatial domain (possibly with some measure of uncertainty). The standard procedure for doing this is known as kriging.

9.3 Raster data and images

Raster data consist of data on a regular square grid, with each data location known as a pixel. Image data is of the same form. A basic (greyscale) image has a single real valued measurement at each pixel location. We could write this as

$$\{z(\mathbf{s}_{i,j})|i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n\}$$

Where now the sites are labelled by a row and column index. But in this case we will typically define $z_{i,j} \equiv z(\mathbf{s}_{i,j})$, and will often consider these to be elements of the $m \times n$ matrix Z.

9.3.1 Image of cell nuclei

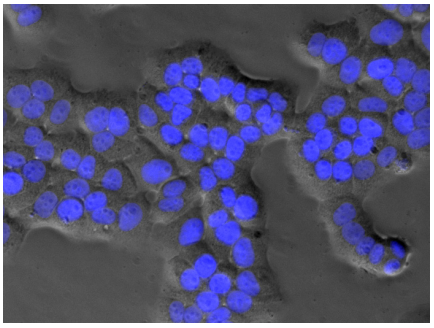
We will look at a freely available image of biological cells with stained nuclei. The image consists of 696 columns and 520 rows of pixels.

```
library(imager)
```

```
img = load.image("https://upload.wikimedia.org/wikipedia/commons/6/6d/H4IIE_cell
img
```

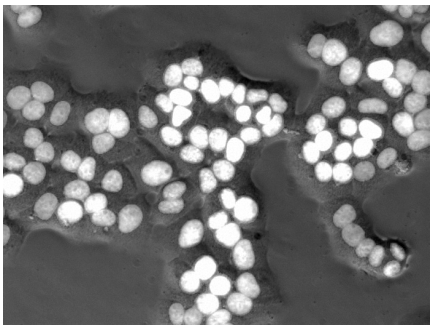
```
Image. Width: 696 pix Height: 520 pix Depth: 1 Colour channels: 3
```

```
plot(img, axes=FALSE)
```



This is a colour image with each pixel having red, green and blue intensities. Let's just pull out the blue channel.

```
bimg = channels(img, 3)[[1]]
plot(bimg, axes=FALSE)
```



This gray image has a single real-valued intensity at each pixel location (the brighter the pixel, the higher the intensity). Here, the inferential problem is not interpolation, but typically smoothing/denoising, and sometimes segmentation.

9.3.2 Images as data on a regular square lattice

Image data can be thought of as data on a regular square lattice. Each node of the lattice represents a pixel, and the intensity of that pixel is associated with the corresponding node of the lattice. Nodes in the lattice have a natural neighbourhood structure, and can be joined by edges according to some simple criterion (eg. each node is joined to its four nearest neighbours). This neighbourhood structure can be used to define statistical models for images, induced by a corresponding model on the associated lattice/graph.

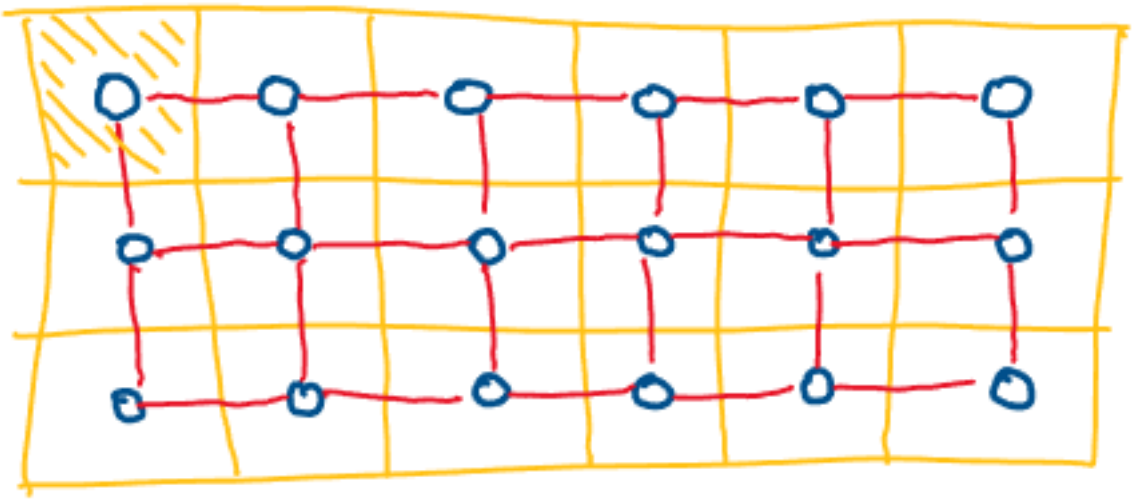


Figure 9.1: Image data as data on a square lattice

In the above figure, the yellow squares represent image pixels. The blue circles represent nodes of a graph, and the red lines denote edges joining the nodes of the lattice according to some neighbourhood structure. Here a first-order neighbourhood is used (4 nearest neighbours), but other neighbourhood structures are possible. Second order neighbourhood structures (8 nearest neighbours) are sometimes used, and these lead to statistical models with different properties.

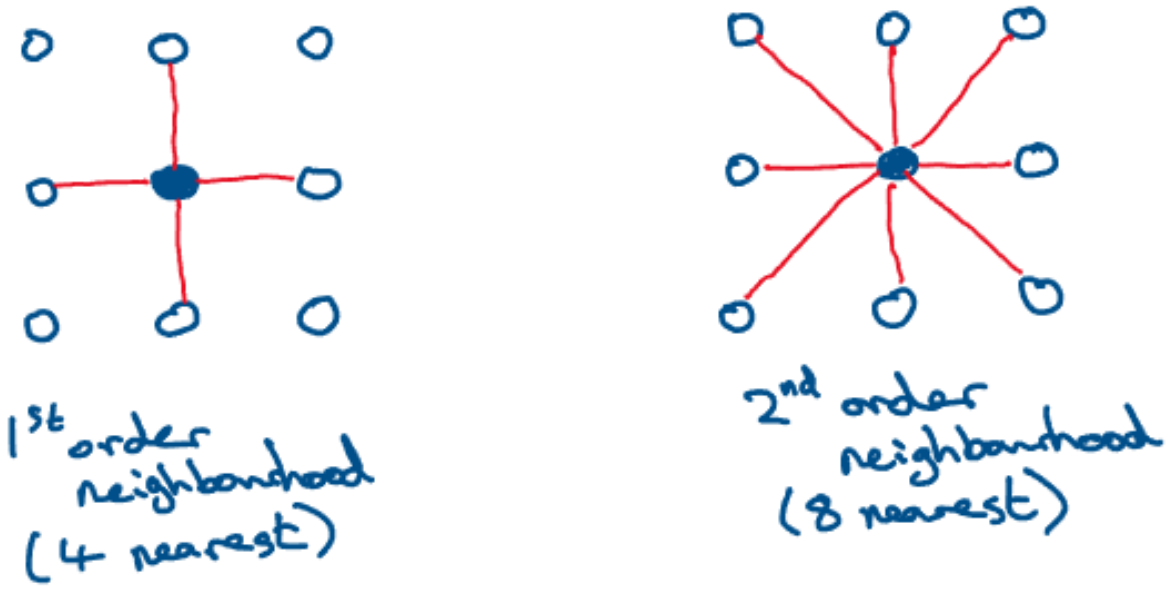


Figure 9.2: First and second order neighbourhoods of a node on a 2d square lattice

9.4 Areal data and data on irregular lattices

Images are a very simple example of *areal data*. In an image, space is divided into squares called pixels, and the measurement or value associated with each pixel represents some kind of total or average or integral over that square region. Spatial data of this form is very common, but often space is not divided into a collection of identical squares.

When thinking about spatial data at large geographic scales, there are natural ways of segmenting space, corresponding to some kind of administrative unit. The Earth is divided into continents and continents are divided into countries. Countries are often divided into states. States are often divided into counties and counties are often divided into districts or parishes, etc. These units correspond to administrative areas, and it is natural to collect data aggregated at the level of these regions. However, these regions typically have complicated and irregular shapes.

Dealing with complex geographic data of this sort is the subject of geographic information systems (GIS). This is a huge topic which is not the main focus of this course. We will just learn about the bare minimum needed to do some interesting spatial analysis.

9.4.1 North Carolina SIDS dataset

For illustration, consider the US state of North Carolina (NC). The state is divided into 100 counties, and these are the main administrative units within the state. We can therefore think of NC as being the disjoint union of 100 areal regions. Administrative data is often collected at the county level. Here we will look at some health data, relating to the incidence of SIDS in NC. The data is in the `spData` package, and the `spdep` package is useful for building dependency structures between spatial regions.

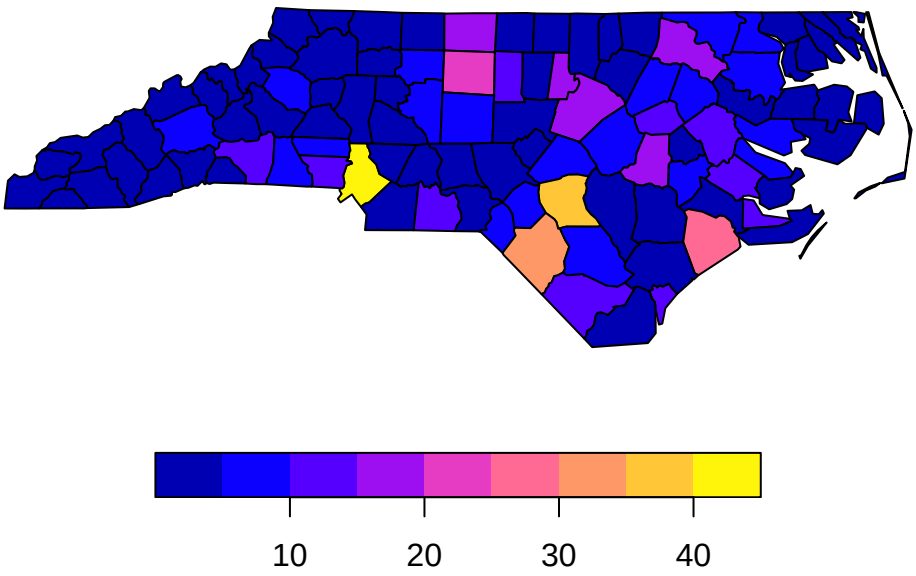
```
library(spData)
library(spdep)
```

```
nc <- st_read(system.file("shapes/sids.gpkg", package="spData")[1])
```

```
Reading layer `sids' from data source
  `/home/darren/R/x86_64-pc-linux-gnu-library/4.5/spData/shapes/sids.gpkg'
  using driver `GPKG'
Simple feature collection with 100 features and 22 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
Geodetic CRS:  NAD27
```

Here, we read the data from a GeoPackage file. This format is often used for storing complex geographic data in a standard way. This dataset contains many fields, but we will focus on the `SID74` column.

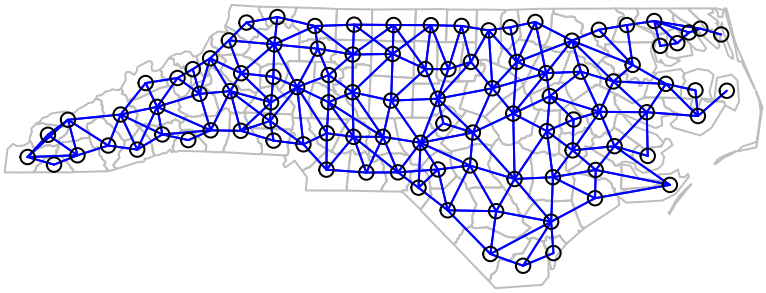
```
plot(nc[,"SID74"])
```



This gives a nice visual representation of the data in its natural spatial context. We can see from the plot that nearby areas tend to be correlated, and so we would want any statistical models we use to take account the spatial context. In principle there are many ways that this could be done, but in practice, it is often done by mapping the data onto an irregular lattice, and defining appropriate statistical models on the lattice.

To map the areal problem to a lattice, we first need to define the nodes of the lattice. This is typically done by placing a node at the centroid of each region, but there are other possibilities. Then, some kind of neighbourhood structure is required to give the edges of the lattice. There are many possibilities here. One natural option is to join two regions by an edge if they share a common boundary. This is illustrated below (but don't worry about what the code is doing).

```
plot(st_geometry(nc), border="grey")
ncCR85_nb <- read.gal(
    system.file("weights/ncCR85.gal",
                package="spData")[1],
    region.id=as.character(nc$FIPS))
plot(ncCR85_nb, sf::st_geometry(nc), add=TRUE, col="blue")
```



Other options include joining regions to k-nearest neighbours, or joining regions if they are within a certain distance of one another. These different options will lead to models with different properties, and which is most appropriate will depend very much on the context. Some sort of *weight* will typically be associated with each edge, indicating the degree of relatedness of the two connected nodes. This could be a function of the distance between the nodes, but there are other possibilities.

9.5 Spatial point data

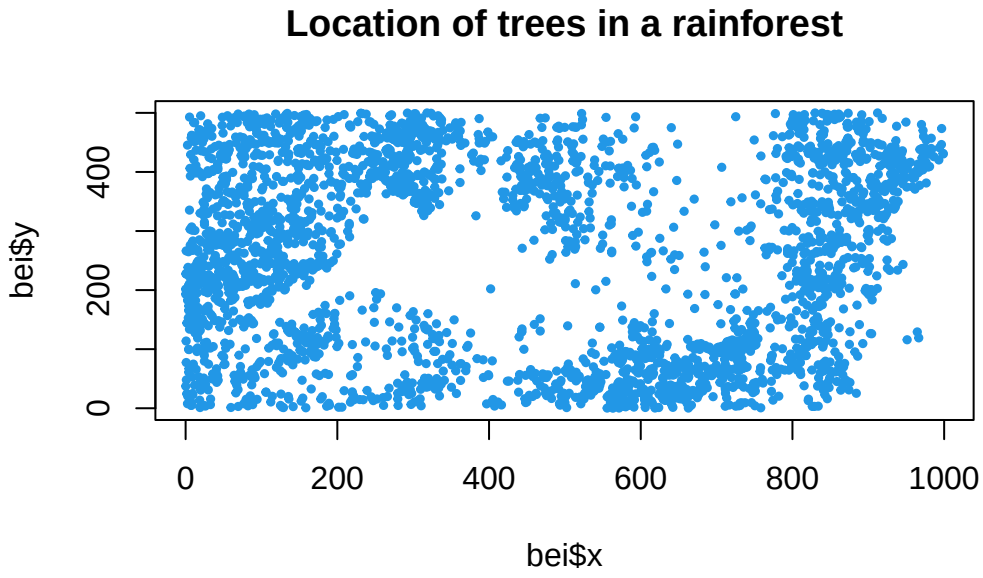
The final kind of spatial data that we will consider in this course is spatial point pattern data, for data arising from some kind of spatial point process. Here it is the *locations* of the data points that are of interest. It is the spatial locations that are measured, and considered random. They are not chosen by the experimenter. The location of trees in a natural rainforest would be a good example of spatial point data. The `spatstat` package is useful for the analysis of spatial point data, and contains some example datasets.

```
library(spatstat) # package for spatial point patterns
```

```
bei
```

```
Planar point pattern: 3604 points
window: rectangle = [0, 1000] x [0, 500] metres
```

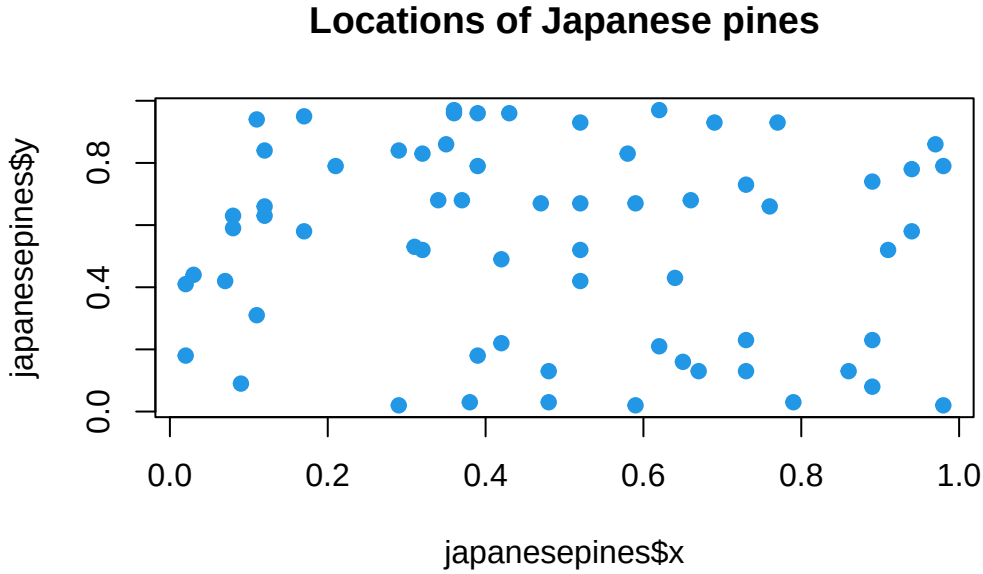
```
plot(bei$x, bei$y, pch=19, cex=0.5, col=4,
     main="Location of trees in a rainforest")
```



We can clearly see that the tree locations are not uniformly distributed within the sampling window. There are distinct areas where there are very few trees, and other areas where trees are densely packed. We would say that the trees are clustered in a non-uniform way. Assessing whether points are uniformly distributed, and if not, how they deviate from uniformity, is a typical inferential question in spatial point data analysis.

In contrast, below are the locations of some Japanese Pines.

```
plot(japanesepines$x, japanesepines$y, pch=19, col=4,
     main="Locations of Japanese pines")
```



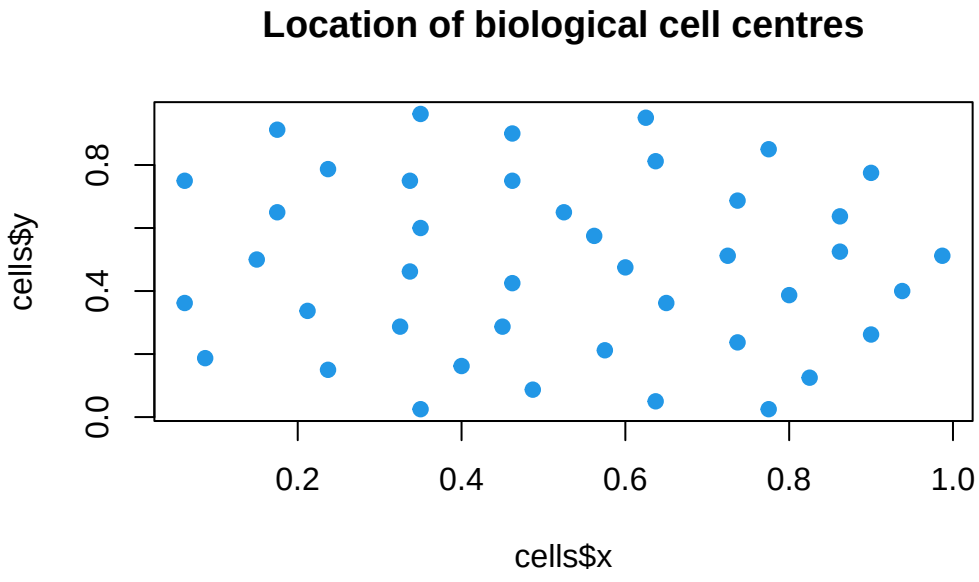
This distribution looks as though it could be *uniformly* randomly distributed. It is not that the trees are perfectly *evenly* spread. There are examples of pairs of trees that are very close together, and trees that are far from other trees. But these are things that one would expect to arise by chance, even if the true data generating mechanism is uniformly random.

Below is another example of tree locations that are clustered.

```
plot(redwood$x, redwood$y, pch=19, col=4,
     main="Location of redwood trees")
```

But not all deviations from uniformity are clusterings. Below is some data showing the locations of the centres of some biological cells under a microscope.

```
plot(cells$x, cells$y, pch=19, col=4,
     main="Location of biological cell centres")
```



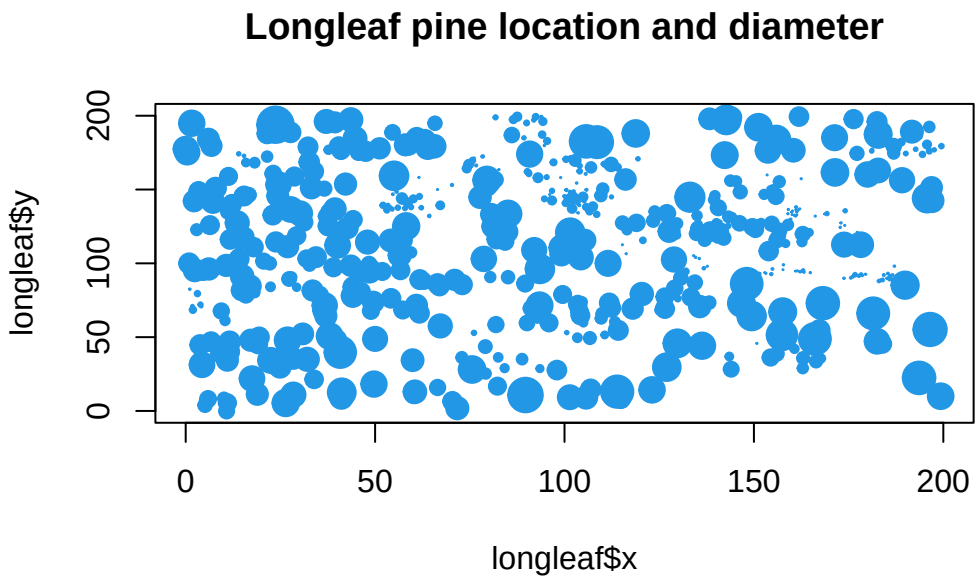
We see that these points are very *evenly* spread - *too* evenly to have arisen by chance from a uniformly random process. There are no pairs of points that are very close (and no points that are exceptionally far away from

all other points). Here there is a *repulsive* mechanism that is keeping the cell centres from getting too close to one another. This deviation from randomness is in some ways the opposite of the clustering we have observed in other point pattern examples. Detecting this kind of repulsive mechanism is also of inferential interest.

9.5.1 Marked point patterns

Sometimes there are measurements associated with each point, in addition to its location. Such data is called a *marked* point pattern. A random mechanism generating such data is called a *marked point process*. For example, in the context of trees, in addition to measuring the location of each tree, one could also measure the tree diameter.

```
plot(longleaf$x, longleaf$y, cex=longleaf$marks/30, pch=19, col=4,
     main="Longleaf pine location and diameter")
```



Understanding whether there is any relationship between the marking and the (relative) position of the point is a typical inferential question.

9.6 Fully spatio-temporal data

At the end of the course we will put together what we know about time series with what we know about spatial statistics in order to try and make sense of fully spatio-temporal data - that is, data associated with both a point in space and a specific time. Again, there are many different kinds of spatio-temporal data, each requiring a different modelling approach. We will concentrate on classical geostatistical spatio-temporal data, consisting of a collection of time series from some fixed but irregularly located sites. We can then write our observations as $z(\mathbf{s}_i, t)$. A typical example might be recordings of daily maximum temperature at a fixed collection of spatially distributed weather stations over a common time window. We will defer further discussion of spatio-temporal data and modelling until the final chapter.

10 Continuously varying random fields

10.1 Introduction

As discussed in the previous chapter, we often think of a spatial dataset as being a partial observation of a latent random field that varies continuously in space. It will be helpful to think a bit more carefully about what we mean by random fields and how they should be characterised. Essentially, a random field is a random function whose domain is the spatial domain. That is, for each $\mathbf{s} \in \mathcal{D} \subset \mathbb{R}^d$ (for some $d \in \mathbb{N}$, and typically $d = 1, 2$ or 3), $Z(\mathbf{s})$ is a random variable over the sample space $\mathcal{Z} \subset \mathbb{R}$. A *realisation* of the random field will be the collection of realisations of the random variables, $\{z(\mathbf{s})|\mathbf{s} \in \mathcal{D}\}$, and hence a (deterministic) function $\mathcal{D} \to \mathcal{Z}$. The variable $z(\cdot)$ is sometimes said to be *regionalised*, in order to emphasise its dependence on a continuously varying spatial parameter.

We typically characterise a random variable by its probability distribution, but there is a technical difficulty here, in that we have a distinct (though typically not independent) random quantity at each point in a continuously varying space, and so our random variable is infinite dimensional. Defining probability distributions over such variables is not completely straightforward. Nevertheless, it seems intuitively reasonable that if we are able to specify the joint distribution for any arbitrary finite collection of sites, then this should completely determine the full distribution of the random field. This turns out to be true provided that some reasonable consistency conditions are satisfied. We can define the cumulative distribution function for a finite number of sites as

$$F(z_1, z_2, \ldots, z_n; \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n) = \mathbb{P}[Z(\mathbf{s}_1) \leq z_1, Z(\mathbf{s}_2) \leq z_2, \ldots, Z(\mathbf{s}_n) \leq z_n].$$

In order to be consistent, $F(\cdot)$ should be symmetric under a permutation of the site indices (it shouldn't matter what order the sites are listed), and further, $F(\cdot)$ should be consistent under marginalisation. That is,

$$F(\infty, z_2, \ldots, z_n; \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n) = F(z_2, \ldots, z_n; \mathbf{s}_2, \ldots, \mathbf{s}_n).$$

Provided an $F(\cdot)$ can be defined for all $n \in \mathbb{N}$ in this consistent way, then $F(\cdot)$ determines the probability distribution of a random field whose finite dimensional distributions are consistent with $F(\cdot)$. This result is known as Kolmogorov's extension theorem (AKA the Kolmogorov *existence* theorem and sometimes as the Kolmogorov *consistency* theorem).

Not all random fields have finite moments, but when the first moments exist, it is useful to define $\mu : \mathcal{D} \to \mathbb{R}$ by

$$\mu(\mathbf{s}) = \mathbb{E}[Z(\mathbf{s})], \quad \forall \mathbf{s} \in \mathcal{D}.$$

Similarly, when second moments exist, it is useful to define the covariance function $C : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ by

$$C(\mathbf{s}, \mathbf{s}') = \mathbb{C}\text{ov}[Z(\mathbf{s}), Z(\mathbf{s}')], \quad \forall \mathbf{s}, \mathbf{s}' \in \mathcal{D}.$$

The covariance function must define a positive-definite kernel. This means that for any finite collection of sites, $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n \in \mathcal{D}$, and any n-vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^\top \in \mathbb{R}^n$ we must have

$$\sum_{j=1}^{n} \sum_{k=1}^{n} \alpha_j \alpha_k C(\mathbf{s}_j, \mathbf{s}_k) \geq 0.$$

This is clearly required since

$$\sum_{j=1}^{n} \sum_{k=1}^{n} \alpha_j \alpha_k C(\mathbf{s}_j, \mathbf{s}_k) = \mathbb{V}\text{ar}\left[\sum_{j=1}^{n} \alpha_j Z(\mathbf{s}_j)\right] \geq 0.$$

When first and second moments exist, the field is called a **second-order random field**.

10.2 Stationarity

In the context of time series analysis, we found stationary time series models to be useful building blocks for creating more realistic time series models. The same is true in spatial statistics. For most real, interesting spatial datasets, a stationarity assumption is not reasonable, but a stationary component is often invaluable in the construction of a more appropriate model. A time series model is stationary if its distribution is invariant under a shift in time. A random field is stationary if its distribution is invariant under a shift in space.

*A random field is **strictly stationary** iff*

$$F(z_1, z_2, \ldots, z_n; \mathbf{s}_1 + \mathbf{h}, \mathbf{s}_2 + \mathbf{h}, \ldots, \mathbf{s}_n + \mathbf{h}) = F(z_1, z_2, \ldots, z_n; \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n)$$

for all finite collections of sites and all shift vectors $\mathbf{h} \in \mathcal{D}$.

10.3 Weak stationarity

Strict stationarity is an extremely powerful property for a process to have, but just as for time series, a weaker notion of stationarity is useful in practice. A second order random field is **weakly stationary** (AKA **second-order stationary**) iff

- $\mu(\mathbf{s}) = \mu(\mathbf{s} + \mathbf{h}) = \mu, \quad \forall \mathbf{s}, \mathbf{h} \in \mathcal{D}$,
- $C(\mathbf{s}, \mathbf{s} + \mathbf{h}) = C(\mathbf{s}', \mathbf{s}' + \mathbf{h}) = C(\mathbf{0}, \mathbf{h}), \quad \forall \mathbf{s}, \mathbf{s}', \mathbf{h} \in \mathcal{D}$.

It is clear that a strictly stationary second-order random field will also be weakly stationary. For weakly stationary random fields we define the function $C : \mathcal{D} \to \mathbb{R}$ by

$$C(\mathbf{h}) \equiv C(\mathbf{0}, \mathbf{h}).$$

This function is the equivalent of the auto-covariance function from time series analysis. In the context of spatial statistics, this function is known as the **covariogram**. Although we are using C for two (slightly) different functions, the different domains allow disambiguation. It is clear that the covariogram determines the covariance function, since $C(\mathbf{s}, \mathbf{s}') = C(\mathbf{s}' - \mathbf{s})$. Some basic properties of the covariogram derive directly from properties of covariance.

- $\mathbb{V}\mathrm{ar}[Z(\mathbf{h})] = C(\mathbf{0}), \ \forall \mathbf{h} \in \mathcal{D}$
- $C(\mathbf{h}) = C(-\mathbf{h}), \ \forall \mathbf{h} \in \mathcal{D}$
- $|C(\mathbf{h})| \leq C(\mathbf{0}), \ \forall \mathbf{h} \in \mathcal{D}$
- C is a positive semi-definite function. That is, for any finite collection of sites, $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n \in \mathcal{D}$, and any n-vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^\top \in \mathbb{R}^n$ we have

$$\sum_{j=1}^{n} \sum_{k=1}^{n} \alpha_j \alpha_k C(\mathbf{s}_j - \mathbf{s}_k) \geq 0.$$

10.4 Intrinsic stationarity

Sometimes even weak stationarity is too strong a condition. Some processes of interest in geostatistics are not stationary, but *do* have stationary (and zero mean) *increments*. A second order process is **intrinsically stationary** iff

- $\mathbb{E}[Z(\mathbf{s} + \mathbf{h}) - Z(\mathbf{s})] = 0, \ \forall \mathbf{s}, \mathbf{h} \in \mathcal{D}$,
- $\mathbb{V}\mathrm{ar}[Z(\mathbf{s} + \mathbf{h}) - Z(\mathbf{s})] = \mathbb{V}\mathrm{ar}[Z(\mathbf{s}' + \mathbf{h}) - Z(\mathbf{s}')] = \mathbb{V}\mathrm{ar}[Z(\mathbf{h}) - Z(\mathbf{0})], \ \forall \mathbf{s}, \mathbf{s}', \mathbf{h} \in \mathcal{D}$.

It is clear that weakly stationary processes are intrinsically stationary, but the converse is not true in general. Note that the Wiener process is a simple 1-d example of a random field that is intrinsically stationary but not weakly stationary.

We define the **semi-variogram** function, $\gamma : \mathcal{D} \to \mathbb{R}$ by

$$\gamma(\mathbf{h}) \equiv \frac{1}{2}\mathbb{V}\text{ar}[Z(\mathbf{h}) - Z(\mathbf{0})],$$

and $2\gamma(\mathbf{h})$ is the **variogram**. Note that this use of $\gamma(\cdot)$ for the semi-variogram is ubiquitous in geostatistics, but somewhat conflicts with its use as an auto-covariance function in time series analysis. Fortunately, it is usually clear from the context which is intended. In the case of an *intrinsically stationary* process, we have

$$\gamma(\mathbf{h}) = \frac{1}{2}\mathbb{V}\text{ar}[Z(\mathbf{s}+\mathbf{h}) - Z(\mathbf{s})],$$

where the RHS is independent of $\mathbf{s}$, and $\mathbb{V}\text{ar}[Z(\mathbf{s}+\mathbf{h}) - Z(\mathbf{s})]$ is the variogram. Note that intrinsically stationary processes have constant mean (say, $\mathbb{E}[Z(\mathbf{s})] = \mu$). Unfortunately, the mean and variogram do not completely characterise an intrinsically stationary process, since $Z(\mathbf{s})$ and $Z(\mathbf{s}) + U$ (where U is a mean zero random quantity independent of $Z(\cdot)$) both have exactly the same increments, the same mean, and the same variogram, but different variances. This problem is usually addressed by fixing (or *registering*) the value of the field at a single point in space, $\mathbf{s}_0 \in \mathcal{D}$, ie. $Z(\mathbf{s}_0) \equiv \mu$. This is exactly analogous to fixing the value of a Wiener process to be zero at time zero.

The variogram is a useful concept because it is well-defined for all intrinsically stationary processes, whereas the covariogram is only well-defined for weakly stationary processes. For a process that *is* weakly stationary, it is clear that the covariogram and variogram are related by

$$\gamma(\mathbf{h}) = C(\mathbf{0}) - C(\mathbf{h}), \quad \forall \mathbf{h} \in \mathbf{D}, \tag{10.1}$$

since $2\gamma(\mathbf{h}) = \mathbb{V}\text{ar}[Z(\mathbf{h}) - Z(\mathbf{0})] = C(\mathbf{0}) + C(\mathbf{0}) - 2C(\mathbf{h})$. This allows direct computation of $\gamma(\cdot)$ from $C(\cdot)$. To compute $C(\cdot)$ from $\gamma(\cdot)$, the stationary variance of the process, $C(\mathbf{0})$, must also be known. It is worth noting that since $|C(\mathbf{h})| < C(\mathbf{0})$, $\gamma(\mathbf{h})$ is bounded above by $2C(\mathbf{0})$. Consequently, *weakly stationary processes cannot have unbounded variograms*, and therefore any intrinsically stationary process with an unbounded covariogram cannot be weakly stationary.

In the case of a stationary process, it is relatively straightforward to map back and forth between the (semi-)variogram and the covariogram. For an *intrinsically* stationary process, there is no covariogram, but mapping back and forth between the variogram and the covariance function is often of interest. We will start by computing the variogram from the covariance function, since that is simpler.

$$\begin{aligned} 2\gamma(\mathbf{h}) &= \mathbb{V}\text{ar}[Z(\mathbf{s}+\mathbf{h}) - Z(\mathbf{s})] \\ &= \mathbb{V}\text{ar}[Z(\mathbf{s}+\mathbf{h})] + \mathbb{V}\text{ar}[Z(\mathbf{s})] - 2\mathbb{C}\text{ov}[Z(\mathbf{s}+\mathbf{h}), Z(\mathbf{s})] \end{aligned}$$

and hence

$$2\gamma(\mathbf{h}) = C(\mathbf{s}+\mathbf{h}, \mathbf{s}+\mathbf{h}) + C(\mathbf{s}, \mathbf{s}) - 2C(\mathbf{s}+\mathbf{h}, \mathbf{s}). \tag{10.2}$$

We can use this to compute the variogram from the covariance function. Since the RHS is independent of $\mathbf{s}$, it will often be convenient to evaluate this at $\mathbf{s} = \mathbf{0}$,

$$2\gamma(\mathbf{h}) = C(\mathbf{h}, \mathbf{h}) + C(\mathbf{0}, \mathbf{0}) - 2C(\mathbf{h}, \mathbf{0}).$$

In general the variogram is insufficient to determine the covariance function, due to the registration problem. However, if we know the value of the field at some location, then it does become possible to determine the covariance function. It is most convenient if the field is known at the origin, ie. fix $Z(\mathbf{0}) = \mu$. The above equation then simplifies to

$$2\gamma(\mathbf{h}) = C(\mathbf{h}, \mathbf{h}),$$

which we can substitute back into Equation 10.2 and rearrange to get

$$C(\mathbf{s}+\mathbf{h}, \mathbf{s}) = \gamma(\mathbf{s}+\mathbf{h}) + \gamma(\mathbf{s}) - \gamma(\mathbf{h}),$$

or alternatively,

$$C(\mathbf{s}', \mathbf{s}) = \gamma(\mathbf{s}') + \gamma(\mathbf{s}) - \gamma(\mathbf{s}' - \mathbf{s}). \tag{10.3}$$

So, provided that the field is fixed at the origin, it is actually very straightforward to compute the covariance function from the variogram. It is in this sense that the variogram almost completely characterises an intrinsically stationary random field.

TODO: some more properties of the variogram

10.4.1 Examples in 1-d

Examples in 1-d can be considered as either having a single spatial dimension, or as being a model for a process in continuous time. We mentioned a few such processes briefly during the time series part of the course.

10.4.1.1 Wiener process (Brownian motion)

The Wiener process (mentioned briefly in Chapter 2) is typically defined to have $W(0) = 0$ and $\mathbb{V}\mathrm{ar}[W(s) - W(t)] = |s - t|,\ s, t \in \mathbb{R}$ (and independent mean zero Gaussian increments). It is therefore intrinsically stationary with semi-variogram

$$\gamma(h) = \frac{1}{2}\mathbb{V}\mathrm{ar}[W(h) - W(0)] = \frac{1}{2}|h|.$$

Since it is fixed at zero, we can easily compute its covariance function as

$$\begin{aligned} C(s,t) &= \gamma(s) + \gamma(t) - \gamma(s-t) \\ &= \frac{1}{2}\left(|s| + |t| - |s-t|\right) \\ &= \min\left\{|s|, |t|\right\}. \end{aligned}$$

10.4.1.2 Fractional Brownian motion (fBM)

The Wiener process has a variogram that increases linearly. It is interesting to wonder about processes with a variogram that increases non-linearly, as a fractional power. That is, a process for which

$$\mathbb{V}\mathrm{ar}[Z(t+h) - Z(t)] = |h|^{2H},$$

where the parameter $H \in (0, 1)$ is known as the Hurst index. Clearly the choice $H = 1/2$ gives the Wiener process, but other choices are associated with an intrinsically stationary process known as fractional Brownian motion. Since the variogram is unbounded, we know that the process can not be stationary, but using

$$\gamma(h) = \frac{1}{2}|h|^{2H},$$

if we fix $Z(0) = 0$ we can compute the covariance function as

$$C(s,t) = \frac{1}{2}\left(|s|^{2H} + |t|^{2H} - |s-t|^{2H}\right).$$

It is worth noting that for $H \neq 1/2$, the increments of this process are stationary but not independent. We will revisit this process in Chapter 11.

10.4.1.3 The OU process

We introduced the OU process in Section 2.3.2.3. By construction this is a Markov process with independent increments, and we showed that for reversion parameter $\lambda > 0$, the process is stationary, with covariogram

$$C(t) = \frac{\sigma^2}{2\lambda} e^{-\lambda|t|},$$

known as the *exponential covariance function*. We can use Equation 10.1 to compute the semi-variogram as

$$\gamma(t) = \frac{\sigma^2}{2\lambda}(1 - e^{-\lambda|t|}).$$

So, for $t > 0$, the semi-variogram increases from zero, asymptoting at the stationary variance of the process.

10.5 Isotropy

TODO

10.6 The nugget

TODO

11 Gaussian processes (GPs)

11.1 Introduction

12 Spectral theory for GPs

12.1 Introduction

13 Kriging

13.1 Introduction

14 Lattice random fields

14.1 Introduction

15 Spatial auto-regressive models

15.1 Introduction

16 Inference for lattice models

16.1 Introduction

17 Spatial point processes

17.1 Introduction

18 Spatio-temporal models and data

18.1 Introduction

A spatio-temporal data set is just a collection of observations labelled in both time and space. So $x(\mathbf{s}; t)$ is an observation at location $\mathbf{s} \in \mathcal{D}$ at time $t \in \mathbb{R}$. The spatial domain, $\mathcal{D}$, is usually a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$. You have seen in the first term that there are a huge range of different kinds of spatial data, and that different models and methods are appropriate for different situations. The range of different kinds of spatio-temporal models and data is even greater. We do not have time to explore these in detail now. Here we need to concentrate on the most commonly encountered form of spatio-temporal data. That is, data consisting of scalar-valued time series on a *regular* time grid of length n being observed at a fixed collection of *irregularly* distributed sites, of which there are m. We could write $\mathbf{x}(\mathbf{s})$ for the time series at site $\mathbf{s} \in \mathcal{D}$, $\mathcal{D} = \{\mathbf{s}_1, \ldots, \mathbf{s}_m\}$. We assume for now that we have temporally aligned time series across the sites, leading to a "full grid" of spatio-temporal data. That is, we have nm scalar observations,

$$\{x(\mathbf{s}_i; t) \mid i = 1, \ldots, m,\ t = 1, \ldots, n\}.$$

For data of this form on a regular time grid, we often use the notation $x_t(\mathbf{s})$ for $x(\mathbf{s}; t)$, and sometimes write $\mathbf{x}_t = (x_t(\mathbf{s}_1), \ldots, x_t(\mathbf{s}_m))^\top$ for the realisation of the spatial process at time t. We could also write X for the $n \times m$ matrix with (i, j)th element $x_i(\mathbf{s}_j)$. When the data matrix is arranged this way it is said to be in "space-wide" format. $\mathsf{X}^\top$ is said to be in "time-wide" format. In practice, it is rare to actually have all nm observations of this form, but we can often represent our data in this form provided that we are allowed to have missing data. In general, strategies are needed to deal with missing spatio-temporal observations.

We immediately see that there are two different ways of "slicing" data of this form, and these correspond to different modelling perspectives. If we adopt the *spatial perspective*, we regard the data as spatial, but with a multivariate observation at each site that happens to be a time series. We can then adopt spatial approaches to model the cross-correlation between the time series at different sites. This spatial perspective underpins many classical approaches to spatio-temporal modelling, but has limitations that we don't have time to fully explore in this module. The alternative *dynamic* or *temporal perspective*, views the data as a time series, where the multivariate observation at each time happens to be the realisation of a spatial process. This latter approach is in many ways more satisfactory, and underpins many modern approaches to spatio-temporal modelling.

18.2 Exploring spatio-temporal data

Before proceeding further, it will be useful to familiarise ourselves with some spatio-temporal data. Our main running example for this chapter will be the dataset `spTimer::NYdata`, some air quality data, measured over time, at a collection of locations across New York. You can find out more about the `spTimer` package with `help(package="spTimer")`. Let's start by trying to understand the basic structure of the data.

```
library(astsa)
library(spTimer)
dim(NYdata)
```

```
[1] 1736   10
```

```
head(NYdata)
```

```
  s.index Longitude Latitude Year Month Day o8hrmax  cMAXTMP     WDSP       RH
1       1   -73.757   42.681 2006     7   1   53.88 27.85772 5.459953 2.766221
2       1   -73.757   42.681 2006     7   2   57.13 30.11563 8.211767 3.197750
3       1   -73.757   42.681 2006     7   3   72.00 30.00001 4.459581 3.225186
4       1   -73.757   42.681 2006     7   4   36.63 27.89656 3.692225 4.362334
5       1   -73.757   42.681 2006     7   5   42.63 25.65698 4.374314 3.950320
6       1   -73.757   42.681 2006     7   6   30.88 24.61968 4.178086 3.420533
```

We can see straight away that the data is currently in "long format", where observations (of several different variables) are recorded with both a position in space and time. You can find out more about the data with `?NYdata`. Let us proceed by finding out more about the sites.

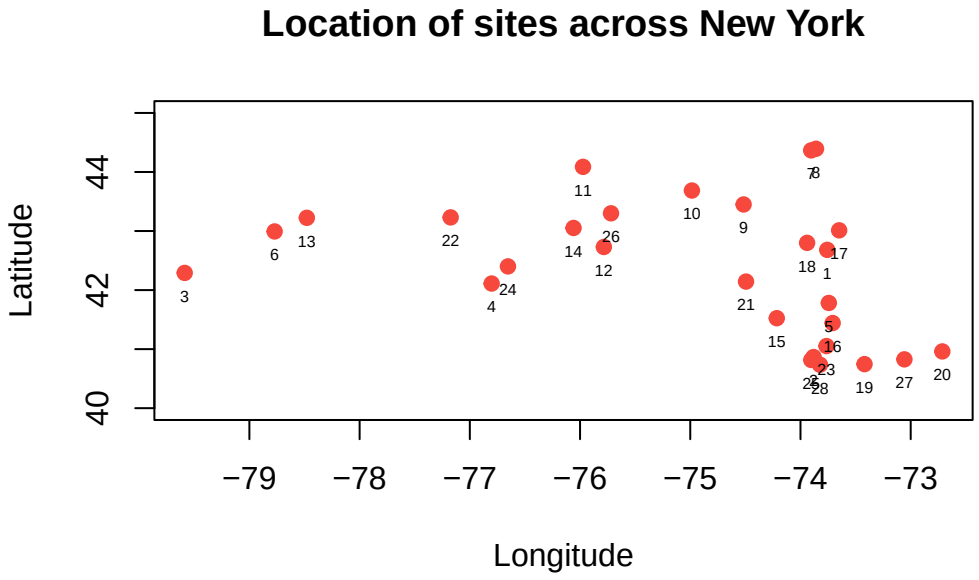
```
sites = unique(NYdata[,1:3])
dim(sites)
```

```
[1] 28  3
```

```
numSites = dim(sites)[1]
head(sites)
```

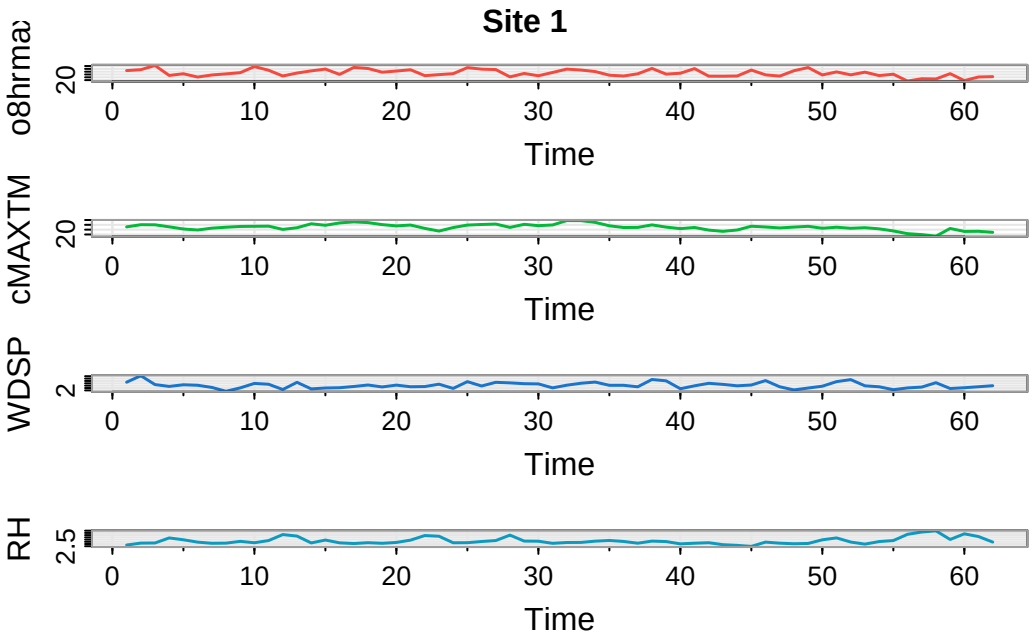
```
    s.index Longitude Latitude
1         1   -73.757   42.681
63        2   -73.881   40.866
125       3   -79.587   42.291
187       4   -76.802   42.111
249       5   -73.743   41.782
311       6   -78.771   42.993
```

```
plot(sites[,2:3], pch=19, col=2, ylim=c(40, 45),
  main="Location of sites across New York")
text(sites[,2:3], labels=sites[,1], pos=1, cex=0.5)
```



So we see that there are 28 sites, scattered irregularly across New York. Let's just look at the first site.

```
site1 = NYdata[NYdata$s.index == 1, 7:10]
tsplot(site1, col=2:5, lwd=1.5, main="Site 1")
```

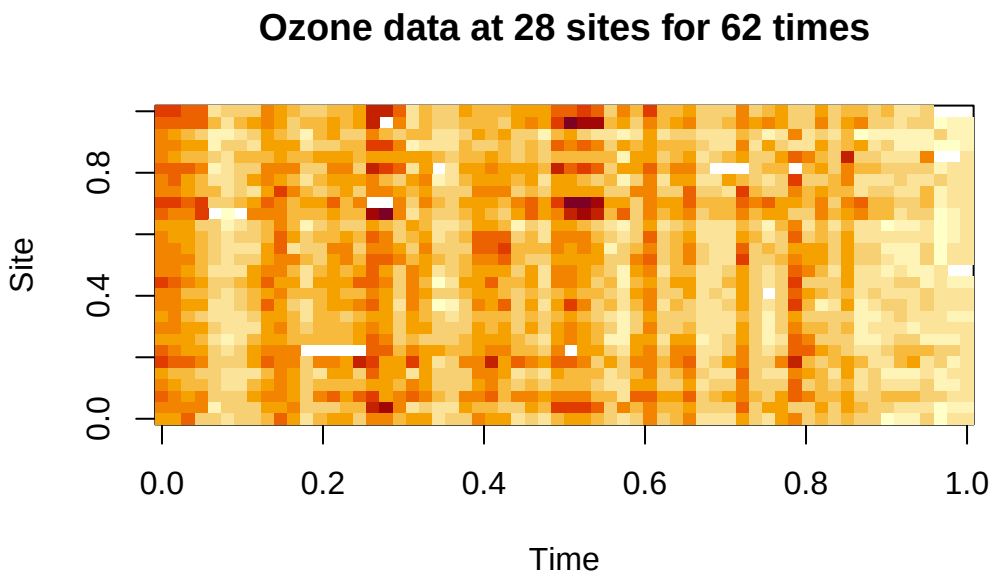


So the observations in space and time are actually multivariate, with several different variables begin measured simultaneously. To keep things simpler, we will focus on just one variable, ozone.

```
ozone = NYdata[,c("o8hrmax", "s.index")]
ozone = unstack(ozone)
dim(ozone) # "space-wide" format
```

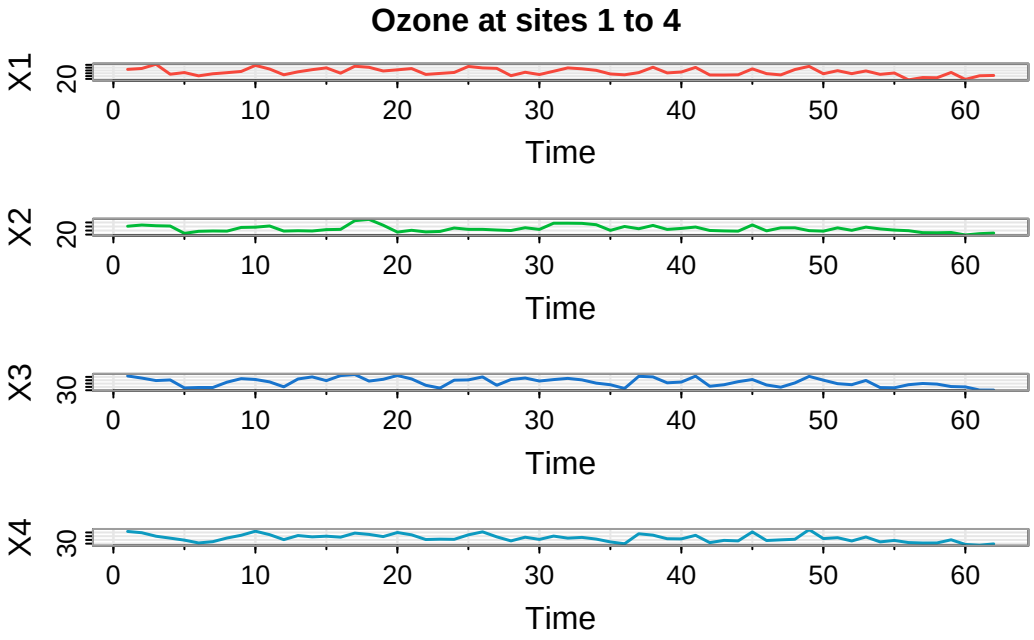
```
[1] 62 28
```

```
## "full grid" representation - "STF" in "spacetime" terminology
image(as.matrix(ozone),
  main="Ozone data at 28 sites for 62 times",
  xlab="Time", ylab="Site")
```



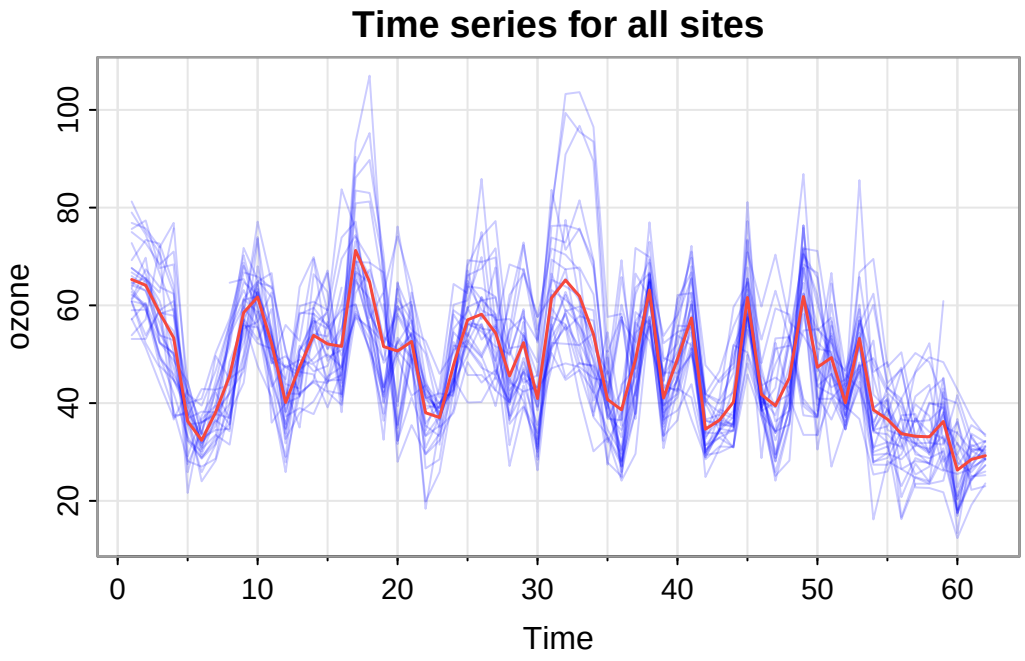
We can visualise the data matrix as an image, which shows a few missing observations (in white), but not too many. This is a fairly complete full-grid spatio-temporal dataset. We can do time series plots for a very small number of sites at once.

```
tsplot(ozone[,1:4], col=2:5, lwd=1.5,
  main="Ozone at sites 1 to 4")
```



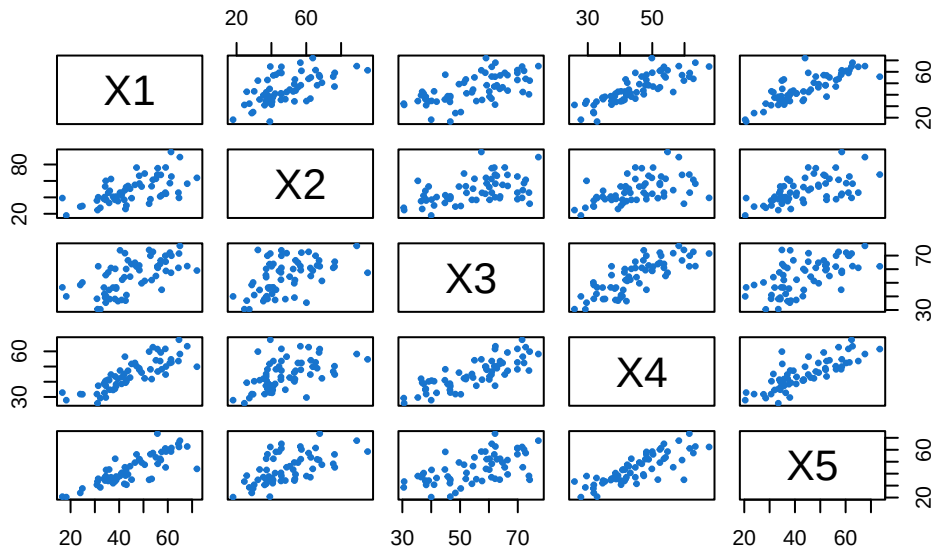
But to look at the time series for all sites simultaneously, we need to overlay, and use transparency to stop the plot from looking too messy.

```
tsplot(ozone, spaghetti=TRUE, col=rgb(0, 0, 1, 0.2),
  ylab="ozone", main="Time series for all sites")
lines(rowMeans(ozone, na.rm=TRUE), col=2, lwd=1.5)
```

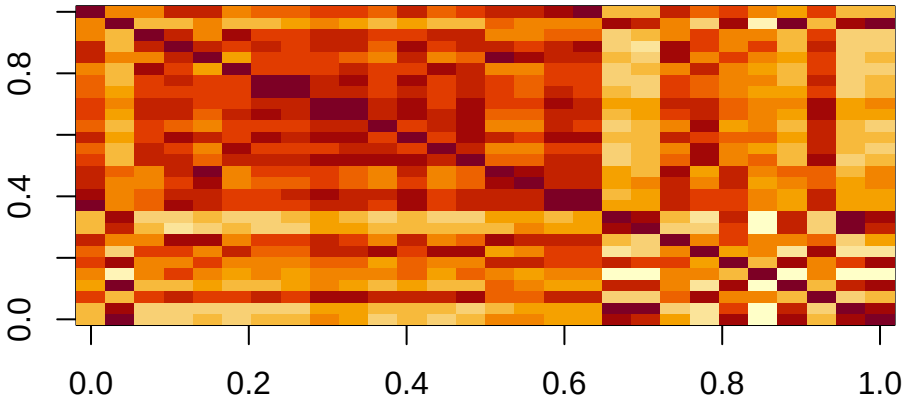


Similarly, we can look in detail at cross-correlations for a small number of time series, but need to revert to an image to see the full correlation structure.

```
pairs(ozone[,1:5], pch=19, col=4, cex=0.5)
```



```
image(cor(ozone, use="pair")[,numSites:1])
```



It is clear from this that there is a lot of interesting correlation structure present, but we are not currently doing anything with the highly relevant context of spatial proximity.

18.3 Spatio-temporal modelling

18.3.1 Spatial models

We begin by taking a more spatial view of spatio-temporal data. So, now we have a time series at a collection of sites. We could completely ignore the temporal dependence and regard the observations at each time as being iid realisations of a spatial process, or we could regard time as an additional dimension. We begin with the former view.

18.3.1.1 Purely spatial models

We begin by ignoring temporal dependence completely, so that observations at each time are independent realisations of some multivariate distribution representing the spatial process.

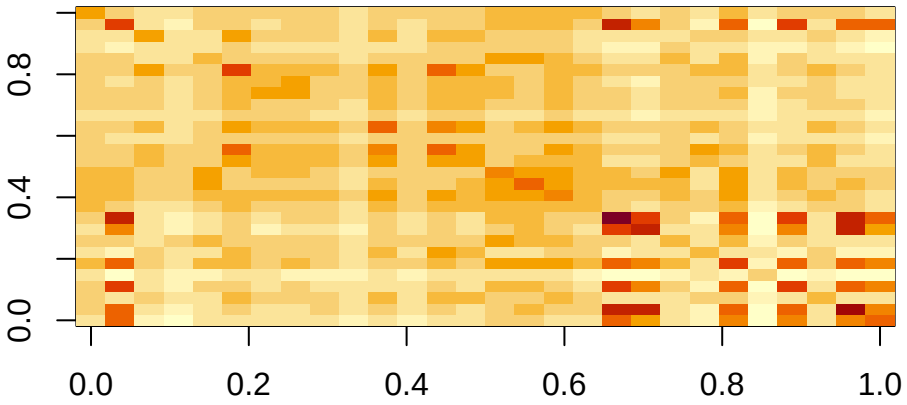
18.3.1.1.1 Unconstrained multivariate data

If we know nothing about the spatial context, we can just regard the observations at each time point as being multivariate normal,

$$\mathbf{X}_t \sim \mathcal{N}(\mathbf{0}, \Sigma),$$

for unconstrained Σ, after stripping out any mean. Then the sample covariance matrix of the data estimates Σ.

```
Sigma = cov(ozone, use="pair")
image(Sigma[,numSites:1])
```



18.3.1.1.2 Spatial covariance

Unconstrained estimation of the covariance matrix is potential problematic, since it has a very large number of degrees of freedom, and we are not exploiting our known spatial context. So we probably prefer to regard the observations as being iid from a Gaussian process (GP) with a parameterised covariance function. Here, for simplicity, we will use an *isotropic* covariance function, depending only on the geographical distance between the sites. Here, for illustrative purposes, we will use the *squared exponential* covariance function

$$C(d) = \sigma^2 \exp\{-d^2/a^2\}, \quad \sigma, a > 0,$$

with σ^2 representing the stationary variance of the GP, and a the length scale. We can encode this in R with

```
cf = function(param) {
  sig = param[1]; a = param[2]
  function(d)
    sig^2 * exp(-(d/a)^2)
}
```

The following R command computes the full matrix of geographical distances between the sites, in km, using WGS84 (an ellipsoidal refinement of the Haversine formula).

```
distMat = sp::spDists(as.matrix(sites[,2:3]), longlat=TRUE)
```

We can use this to construct a covariance matrix over the observations with

```
cm = function(param)
  cf(param)(distMat)
```

Then we can define a log-likelihood function for the data with

```
## centre the data
cOzone = sweep(ozone, 2, colMeans(ozone, na.rm=TRUE))

ll = function(param)
  sum(apply(cOzone, 1,
```

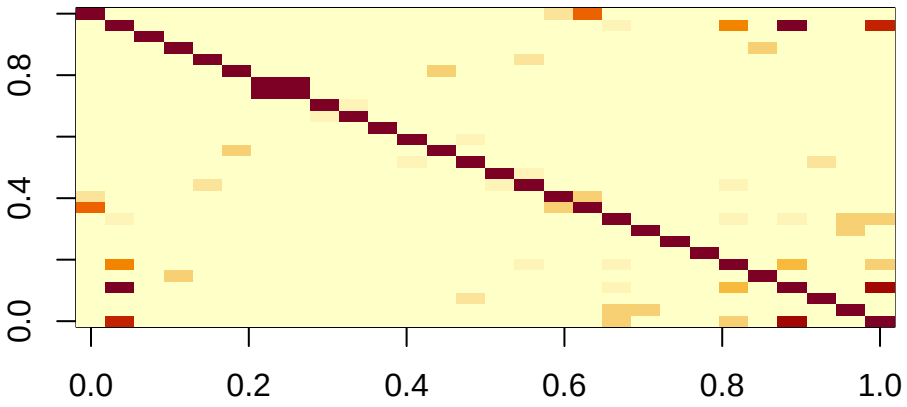
```
    function(x)
      mvtnorm::dmvnorm(x, sigma=cm(param), log=TRUE)),
        na.rm=TRUE)
```

and find the MLE for the covariance function parameters with

```
opt = optim(c(20, 50), ll, control=list(fnscale=-1))
opt$par
```

```
[1] 12.06888 30.16920
```

```
image(cm(opt$par)[,numSites:1])
```



suggesting a length scale of around 30km for ozone. We also see that the optimal covariance matrix looks quite different to the sample covariance matrix, and that there is very little correlation between sites that are not very close together (such as sites 7 and 8). This is good and bad. It is good that it properly spatially smooths, but the assumption of a common variance across sites is probably not realistic.

18.3.1.2 Space-time Gaussian process models

Ignoring the temporal dependence structure in the data is obviously unsatisfactory for various reasons. If we persist with our spatial-first perspective, we consider the presence of a time series at each spatial location. This time series could potentially also be modelled as a GP. If we think that a GP is appropriate for both spatial and temporal variation, and that the same GP model is appropriate for every time series irrespective of spatial location, then we are led to modelling the data in space and time as arising jointly from a single GP with a *separable* space-time covariance function of the form

$$C(\mathbf{s}, t) = C_s(\mathbf{s})C_t(t),$$

for given spatial covariance function $C_s(\cdot)$ and temporal covariance function $C_t(\cdot)$. Adopting a separable covariance function is convenient for multiple reasons. First, if C_s is a valid spatial covariance function and

C_t is a valid temporal covariance function, then their product is guaranteed to be a valid space-time covariance function. Thus, the imposition of separability greatly simplifies the specification of a valid spatio-temporal covariance function. Second, the assumption greatly simplifies computation, since then the joint covariance matrix over all observations can be represented as a Kronecker product of the spatial and temporal covariance matrices, allowing the avoidance of the construction or inversion of very large matrices.

Much of classical spatio-temporal modelling was built on separable GP models. However, it turns out that the assumption of separability is very strong, and quite unrealistic for most spatio-temporal data sets. So, given our limited time, we will abandon this approach, and adopt a more dynamical perspective.

18.3.2 Dynamic models for spatio-temporal data

This term we have studied models for time series, and in particular, ARMA models, and DLMs. Both of these families of models lead to linear Gaussian systems. They therefore determine Gaussian process models, and implicitly determine a (not necessarily stationary) covariance structure. However, we don't specify these models via their covariance structure. We specify their *dynamics*, and the dynamics *implicitly* determines the covariance structure. There are many advantages to this more dynamical perspective.

18.3.2.1 Random walk model

We will begin with a model that is typically over-simplistic, but we will gradually refine and improve it. Rather than assuming that our observations are iid, we will assume that they form a random walk

$$\mathbf{X}_t = \mathbf{X}_{t-1} + \boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \Sigma).$$

This non-stationary model will sometimes be appropriate when there is a high degree of persistence in changes to the levels observed. This model is very simple to fit, since the one-step differences are iid, so we can put

$$\boldsymbol{\varepsilon}_t = \mathbf{X}_t - \mathbf{X}_{t-1},$$

and estimate the parameters of iid $\boldsymbol{\varepsilon}_t$ as for the iid model. Again, we could have an unconstrained Σ, which we can estimate using the sample covariance matrix of the one-step differences, or a constrained matrix, with an explicitly spatial covariance structure, which we can estimate via maximum likelihood, as we have already seen.

18.3.2.2 VAR(1) models

We can greatly improve on the simple random walk model by allowing VAR(1) dynamics, assuming temporal evolution of the form

$$\mathbf{X}_t = \mathsf{G}\mathbf{X}_{t-1} + \boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \Sigma),$$

perhaps following some mean-centring. This model is specified by $m \times m$ matrices G and Σ. This model is very flexible, allowing a range of different stationary and non-stationary dynamics. But the utility of the model depends crucially on having an appropriate structure for the *propagator matrix*, G. Clearly, choosing $\mathsf{G} = \mathbb{I}$ gives the random walk model we have already considered, so this model class includes the random walk model as a special case.

18.3.2.2.1 Unconstrained models

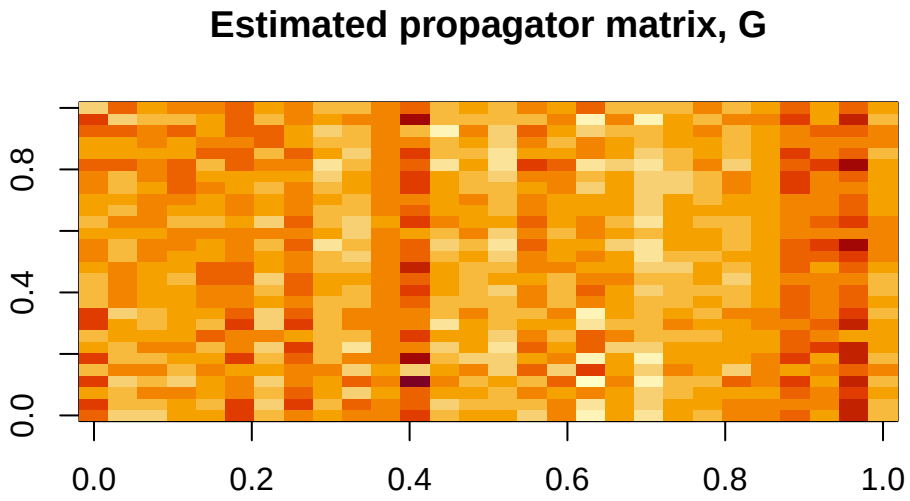
Obviously, one possibility is to consider a completely unconstrained G, with elements to be estimated by least squares (or maximum likelihood). We can use the same least squares approach that we adopted in Chapter 4 by writing our model in the form

$$\mathsf{X}_{2:n} = \mathsf{X}_{1:(n-1)}\mathsf{G}^\top + \mathsf{E},$$

to get the least squares solution

$$\hat{\mathsf{G}}^\top = (\mathsf{X}_{1:(n-1)}^\top \mathsf{X}_{1:(n-1)})^{-1}\mathsf{X}_{1:(n-1)}^\top \mathsf{X}_{2:n}.$$

```
ozone2 = as.matrix(cOzone[2:62,])
ozone1 = as.matrix(cOzone[1:61,])
mod = lm(ozone2 ~ 0 + ozone1)
G = t(mod$coeff)
image(t(G)[,numSites:1], main="Estimated propagator matrix, G")
```



Once we know G we can estimate Σ conditional on G as discussed for the random walk model. Note that there is no reason to expect (or want) G to be symmetric, and there is also no reason why all of the elements should be non-negative. Clearly this approach could be used for any multivariate time series, so we are not properly exploiting the spatial context. Also, for a large number of sites, it can be problematic to estimate all m^2 elements of G given at most nm data points.

18.3.2.2.2 Diagonal propagation

As a compromise between a simple random walk model and a completely unconstrained propagation matrix, a diagonal G can be assumed. This also ignores the spatial context, but nevertheless fixes a number of issues with the simple random walk model. Since G is diagonal, from a least squares perspective the problem reduces to m independent AR(1) models that can be fit separately.

```
apply(cOzone, 2, function(x)
  arima(x, c(1,0,0), include.mean=FALSE)$coef)
```

```
       X1        X2        X3        X4        X5        X6        X7        X8
0.3179119 0.5368396 0.4338366 0.4206251 0.3077022 0.4162096 0.4779711 0.4828871
       X9       X10       X11       X12       X13       X14       X15       X16
0.4572942 0.4952505 0.3255831 0.3944543 0.4224548 0.3950086 0.4009679 0.3972285
      X17       X18       X19       X20       X21       X22       X23       X24
0.3785257 0.4059217 0.4876205 0.5324465 0.3282226 0.2242859 0.4144865 0.1876176
      X25       X26       X27       X28
0.5343495 0.3515989 0.5994477 0.4730342
```

Alternatively, the problem can be set up as a least squares problem for the m-vector $\mathbf{g}$, where $\mathsf{G} = \text{diag}\{\mathbf{g}\}$. This is tractable, with solution

$$\hat{\mathbf{g}} = \left(\sum_{t=2}^{n} \mathbf{x}_{t-1} \circ \mathbf{x}_t\right) \circ \left(\sum_{t=2}^{n} \mathbf{x}_{t-1} \circ \mathbf{x}_{t-1}\right)^{-1}.$$

```
colSums(ozone1*ozone2, na.rm=TRUE)/colSums(ozone1*ozone1, na.rm=TRUE)
```

```
       X1        X2        X3        X4        X5        X6        X7        X8
0.3204579 0.5396044 0.4251620 0.4041003 0.3066428 0.4025162 0.4253797 0.4802306
       X9       X10       X11       X12       X13       X14       X15       X16
0.4468311 0.4804979 0.3204789 0.3849037 0.3993497 0.3830393 0.3997480 0.3953805
      X17       X18       X19       X20       X21       X22       X23       X24
0.3769048 0.4037909 0.4834982 0.5340675 0.3212627 0.2200922 0.4058193 0.1763395
      X25       X26       X27       X28
0.5320828 0.3368777 0.5524844 0.4517766
```

Here, the solution using `arima` is preferred, since it has more intelligent handling of missing data.

18.3.2.2.3 Spatial mixing kernels

Ideally, we would like to use a propagator matrix, G, which takes into account the spatial context. This is relatively simple when the sites lie on a regular lattice (see the later discussion of STAR models), but more challenging for irregularly distributed spatial locations. Note that the ith row of G corresponds to the weights applied to the sites at the previous time point when making a prediction for site i at the current time point. The diagonal model puts all weight on just site i. It might be better to distribute the weights across the k nearest neighbours of site i for some reasonably small $k > 1$. Choosing a small k will ensure that most of the elements of G are zero, and hence G will be a sparse matrix, which has significant computational advantages. However, for fairly small m we could assume a dense G, with weights varying as a function of distance. It is common to assume some sort of kernel distance function, and there are many possible choices with various advantages and disadvantages. For example, we could use the *squared exponential* kernel (irrespective of any covariance kernel assumed for Σ),

$$g_{ij} = \sigma_i \exp\{-\|\mathbf{s}_j - \mathbf{s}_i\|^2 / a_i^2\}.$$

It is quite common (but not required) to assume that the length scale is common across all sites, $a_i = a$. However, there are often very good reasons to allow σ_i to vary across sites, and in this case G will not be symmetric (which is fine).

Given this parameterisation of G, and most likely also a low-dimensional parameterisation of Σ, it is straightforward to evaluate the Gaussian likelihood, and hence optimise the log-likelihood to find the optimal parameters, similar to what we have seen many times previously.

18.3.3 Dynamic latent process models

The spatio-temporal models that we have examined so far have all been special cases of the VAR(1) model. However, when we studied DLMs in Chapter 7, we saw that it can often make sense to assume a *latent process* of auto-regressive form, but to then model our observations as some noisy linear transformation of this hidden latent process. This remains the case in the spatio-temporal context. So it is natural to consider models of DLM form for spatio-temporal processes,

$$\begin{aligned}\mathbf{Y}_t &= \mathsf{F}\mathbf{X}_t + \boldsymbol{\nu}_t, \quad \boldsymbol{\nu}_t \sim \mathcal{N}(\mathbf{0}, \mathsf{V}) \\ \mathbf{X}_t &= \mathsf{G}\mathbf{X}_{t-1} + \boldsymbol{\omega}_t. \quad \boldsymbol{\omega}_t \sim \mathcal{N}(\mathbf{0}, \mathsf{W}),\end{aligned}$$

where $\mathbf{Y}_t$ is our spatial observation at time t, and $\mathbf{X}_t$ is some kind of latent process representation of the underlying system state at time t. The precise nature of the latent process can vary according to the modelling approach adopted. We briefly consider three possibilities.

18.3.3.1 Spatial mixing kernels

If we choose $\mathsf{F} = \mathbb{I}$, then our observations are just random corruptions of the hidden latent state. So the latent state has a very similar interpretation as the models we have been considering so far. In particular, there is a one-to-one correspondence between sites and elements of the latent process state vector, and G has an interpretation as a spatial mixing kernel for the hidden latent process. It can be parameterised in the manner previously discussed. A spatial covariance kernel is often used to parameterise W, and V is often assumed to be diagonal. We have seen how to evaluate the log-likelihood of a DLM, so we can optimise the parameters of the kernel functions using maximum likelihood in the usual way.

18.3.3.2 Example: NY ozone data

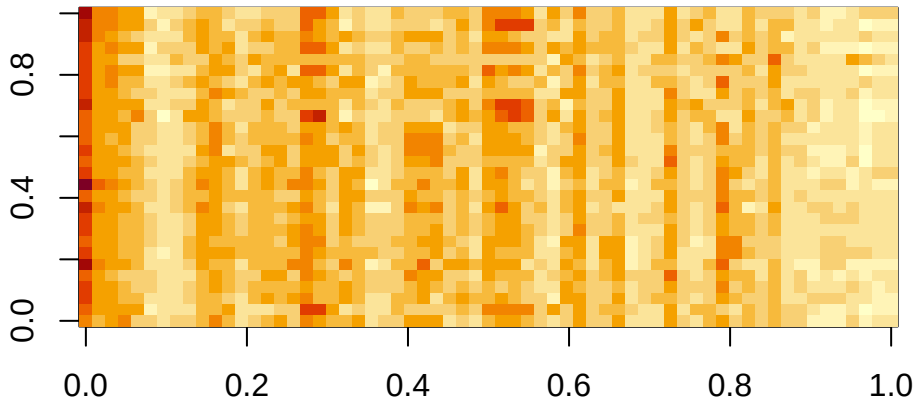
Let's see how we could implement a model like this using the `dlm` R package. To keep things simple we will just assume a propagator matrix of the form $\mathsf{G} = \alpha\mathbb{I}$. We will also start off with an evolution covariance matrix of the form $\mathsf{W} = \sigma_w^2\mathbb{I}$, but we will relax this assumption soon. We can fit this as follows.

```
library(dlm)

buildMod = function(param) {
    alpha = exp(param[1]); sigW = exp(param[2])
    sigV = exp(param[3])
    dlm(FF=diag(numSites), GG=alpha*diag(numSites),
          V = (sigV^2)*diag(numSites), W = (sigW^2)*diag(numSites),
        m0 = rep(0, numSites), C0 = (1e07)*diag(numSites))
}

opt = dlmMLE(as.matrix(cOzone), parm=log(c(0.8, 1, 3.0)),
             build=buildMod)
opt
```

```
$par
[1] -0.9494696  2.4860296 -4.7633070

$value
[1] 5228.299

$counts
```

```
function gradient
      47       47

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
mod = buildMod(opt$par)

ss = dlmSmooth(cOzone, mod)
image(ss$s)
```

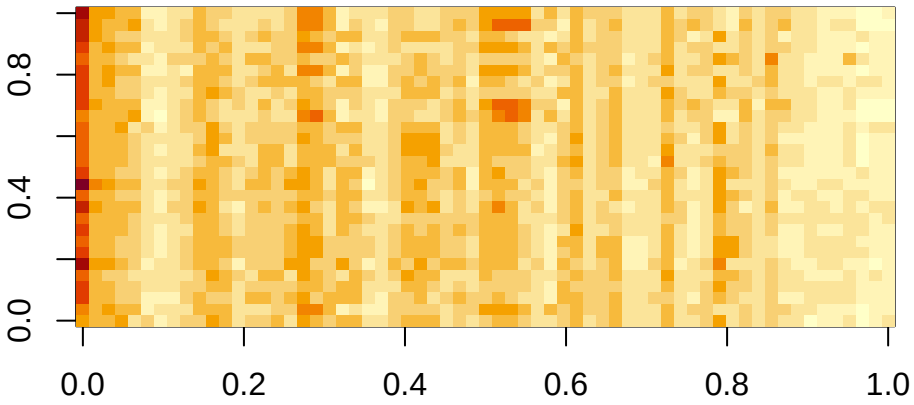


This works, and give smoothed states that are visibly smoother than the raw data. Importantly, it also sensibly smooths over the missing data. But this model isn't really explicitly spatial. Apart from assuming common parameters across sites, the DLMs at each site are independent. For spatial smoothing we really want W to be a spatial covariance matrix. We can fit a model using a spatial variance matrix of the form previously discussed (using our function `cm`) as follows.

```
buildMod = function(param) {
    alpha = exp(param[1]); sigW = exp(param[2])
    a = exp(param[3]); sigV = exp(param[4])
    dlm(FF=diag(numSites), GG=alpha*diag(numSites),
          V = (sigV^2)*diag(numSites), W = cm(c(sigW, a)),
        m0 = rep(0, numSites), C0 = (1e07)*diag(numSites))
}

opt = dlmMLE(as.matrix(cOzone), parm=c(log(0.8), log(1), log(10), log(3.0)),
             build=buildMod, lower=-4, upper=4)
opt
```

```
$par
[1] -1.1417530  2.2887486  4.0000000  0.8926005

$value
[1] 4537.371

$counts
function gradient
      28       28

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
mod = buildMod(opt$par)

ss = dlmSmooth(cOzone, mod)
image(ss$s)
```



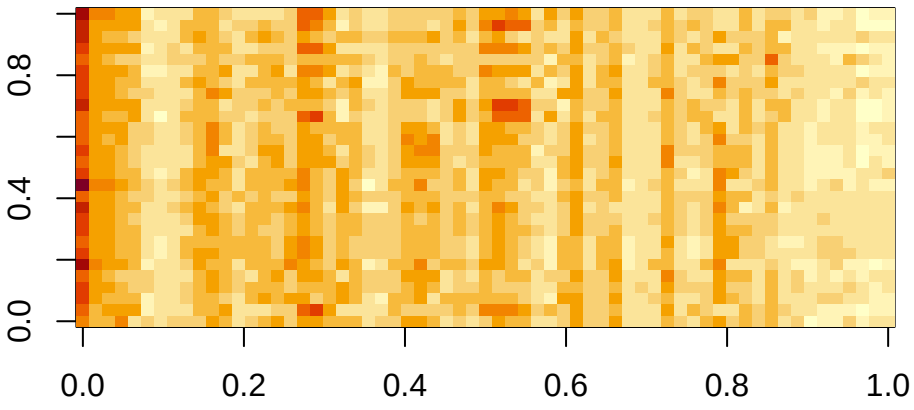
This also works, after a fashion, but note that the optimal length scale is just the upper bound I imposed on the optimiser. If an upper bound is not imposed, the length scale just keeps increasing until the model crashes. Inferring length scales is notoriously difficult in spatial statistics, and is even more challenging in the spatio-temporal setting. So, given that in the context of purely spatial modelling we previously inferred a length scale of around 30km, we could just use that length scale in the context of the dynamic model and not try to optimise it.

```
buildMod = function(param) {
    alpha = exp(param[1]); sigW = exp(param[2])
    sigV = exp(param[3])
    dlm(FF=diag(numSites), GG=alpha*diag(numSites),
```

```
          V = (sigV^2)*diag(numSites), W = cm(c(sigW, 30)),
        m0 = rep(0, numSites), C0 = (1e07)*diag(numSites))
}

opt = dlmMLE(as.matrix(cOzone), parm=c(log(0.8), log(1), log(3.0)),
             build=buildMod)
opt
```

```
$par
[1] -1.0119162  2.3656191  0.6758664

$value
[1] 4848.624

$counts
function gradient
      25       25

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
mod = buildMod(opt$par)

ss = dlmSmooth(cOzone, mod)
image(ss$s)
```



This is better. This is now giving us a dynamic model that can smooth in both time and space jointly.

18.3.3.3 STAR models

It has previously been mentioned that choosing the form of the propagator matrix, G, would be more straightforward if the spatial locations all lay on a regular lattice (typically, in 2d or 3d). Even if our actual observations $\mathbf{Y}_t$ do not, we could nevertheless model the latent state, $\mathbf{X}_t$, as a lattice process. eg., in the 2d case, we could then let $X_{t,i,j}$ be the value of the latent state at time t at position (i, j) on the lattice. A nearest-neighbour model for the time evolution of the latent state might then take the form

$$X_{t,i,j} = \alpha X_{t-1,i,j} + \beta(X_{t-1,i-1,j} + X_{t-1,i+1,j} + X_{t-1,i,j-1} + X_{t-1,i,j+1}) + \omega_{t,i,j},$$

for some fixed $\alpha, \beta > 0$. For stability, we might require $\alpha + 4\beta < 1$. This then determines the *sparse* structure of G. There are many possible variations on this approach. W could be diagonal or parameterised via a spatial covariance kernel. V is often assumed to be diagonal. Models of this form are known as space-time auto-regressive models of order one, or STAR(1), and there are generalisations to higher order, STAR(p).

The matrix F maps the actual observation sites onto the lattice. It will have m rows, and the number of columns will match the total number of sites on the lattice. The ith row of F will have a 1 in the position corresponding to the lattice site closest to $\mathbf{s}_i$, and zeros elsewhere.

DLM smoothing with this model allows interpolation of the observed data onto a regular space-time lattice. However, if the size of the lattice is large, this is computationally very demanding (notwithstanding the sparsity of G). There are other possible approaches to spatio-temporal smoothing and interpolation which may be less computationally demanding.

18.3.3.4 Basis models (spectral approaches)

STAR models provide one approach to interpolate the hidden spatio-temporal process to spatial locations other than those that have been directly observed. STAR models are typically used in order to interpolate onto a regular lattice. However, there is no reason why we can't construct a DLM that allows interpolation onto a continuous space. We just need some basis functions, for example, 2d or 3d Fourier basis functions. So, let

$$\phi_j : \mathbb{R}^2 \to \mathbb{R}, \quad j = 1, 2, \ldots$$

be basis functions defined on the whole of $\mathbb{R}^2$ (or $\mathbb{R}^3$ for 3d data). The idea is that we can represent an arbitrary function $\theta : \mathbb{R}^2 \to \mathbb{R}$ with an appropriate linear combination of basis functions

$$\theta(\mathbf{s}) = \sum_{j=1}^{\infty} \phi_j(\mathbf{s}) x_j,$$

for some collection of coefficients $x_1, x_2, \ldots$. We will seek a reduced dimension representation of a function by using just p basis functions. We will typically choose $p < m$, the number of sites with observations. Then, every p-vector $\mathbf{x} = (x_1, \ldots, x_p)^\top$ determines a function

$$\theta(\mathbf{s}) = \sum_{j=1}^{p} \phi_j(\mathbf{s}) x_j,$$

If we allow $\mathbf{x}$ to evolve in time, then the function $\theta(\cdot)$ that it represents will also evolve in time. So, we can let the coefficient vector evolve according to

$$\mathbf{X}_t = \mathsf{G}\mathbf{X}_{t-1} + \boldsymbol{\omega}_t. \quad \boldsymbol{\omega}_t \sim \mathcal{N}(0, \mathsf{W}),$$

for some very simple propagator matrix such as $\mathsf{G} = \mathbb{I}$ or $\mathsf{G} = \alpha\mathbb{I}$ for $\alpha \in (0, 1)$. Since we know from Chapter 5 that Fourier transforms decorrelate GPs, we can reasonably assume diagonal W.

We probably want an observation model of the form

$$\begin{aligned} Y_{t,i} &= \theta_t(\mathbf{s}_i) + \nu_{t,i} \\ &= \sum_{j=1}^{p} \phi_j(\mathbf{s}_i) X_{t,j} + \nu_{t,i}, \end{aligned}$$

and so F is the $m \times p$ matrix with (i,j)th element $\phi_j(\mathbf{s}_i)$. Again, V is often taken to be diagonal, but doesn't have to be. This is now just a DLM with a small number of parameters that can be fit in the usual ways. If we compute the smoothed coefficients of the latent coefficient vectors, these can be used in conjunction with the basis functions to interpolate the hidden spatial process over continuous space.

There are many possible choices of basis functions that can be used. 2d or 3d Fourier basis functions are the most obvious choice, but cosine basis functions, or wavelet basis functions, or some kind of empirical eigenfunctions can all be used.

18.3.4 R software

Fitting dynamic spatio-temporal models to data gets quite complicated and computationally intensive quite quickly. Relevant R software packages are summarised in the spatio-temporal task view. The IDE package will fit an integro-difference equation model, which we have not explicitly discussed, but is closely related to the STAR and spectral approaches. This package, in addition to a number of other approaches to the analysis of spatio-temporal data using R, is discussed in Wikle, Zammit-Mangion, and Cressie (2019). Otherwise, packages such as spBayes and spTimer will fit spatio-temporal models from a Bayesian perspective, using MCMC. Unfortunately we do not have time to explore these packages properly in this course.

18.4 Example: German air quality data

For further spatio-temporal investigation, it will be useful to have a different dataset to explore. We will now look briefly at a larger dataset than the one we have considered so far, but detailed analysis is left as an exercise.

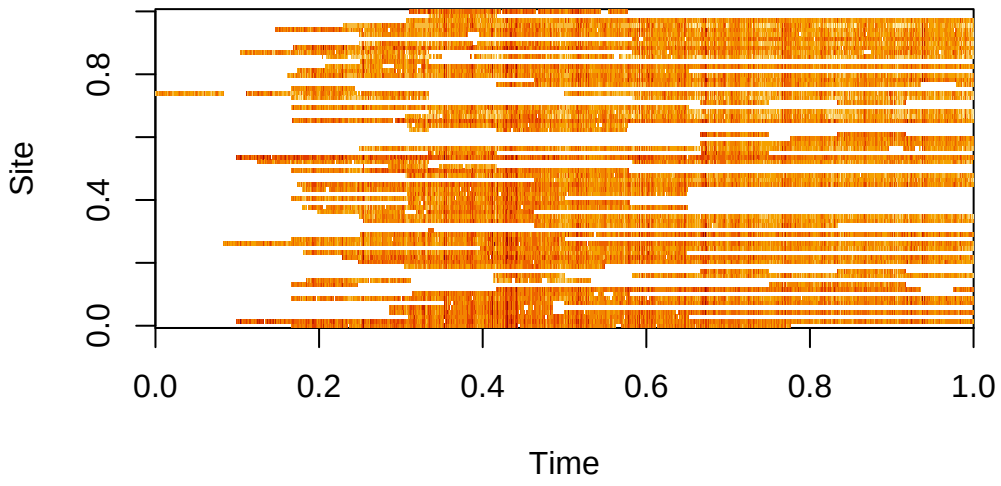
The data is air quality data (specifically, PM10 concentration), for 70 different sites in Germany, over a 10 year period. We can load and inspect the data as follows.

```
library(spacetime)
data(air)
dim(air) # "time-wide" format
```

```
space  time
   70  4383
```

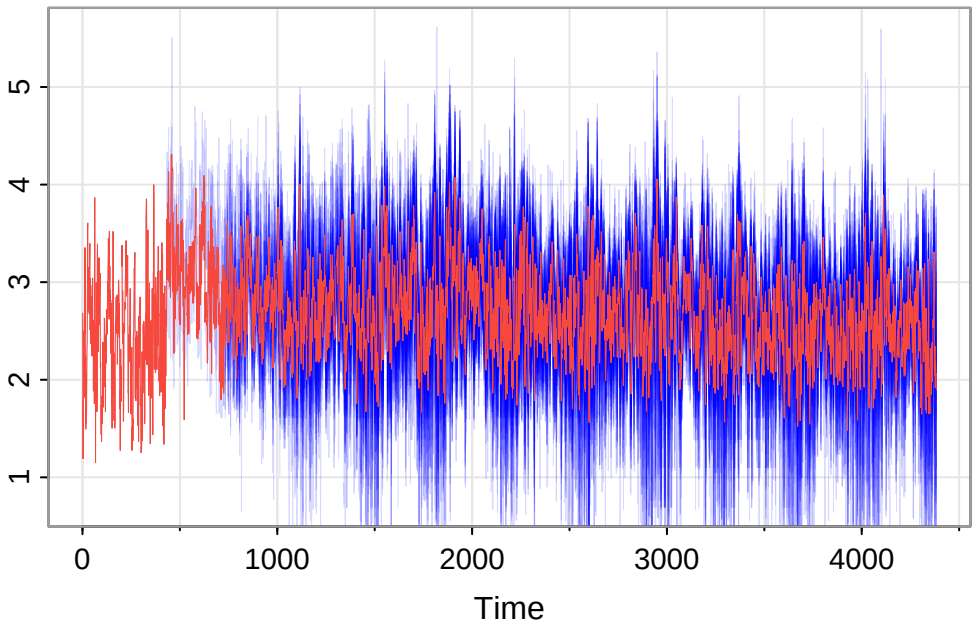
It is actually in "time-wide" format, which we don't necessarily want, and can be logged to be more Gaussian.

```
lair = t(log(air)) # space-wide format (and logged)
image(lair, xlab="Time", ylab="Site") # lots of missing data
```



We see that there is a lot of missing data in this dataset, so a proper analysis will need to carefully handle missing data issues. We can attempt to overlay the time series at the different sites.

```
tsplot(lair, spaghetti=TRUE, col=rgb(0,0,1,0.15), lwd=0.5)
lines(rowMeans(lair, na.rm=TRUE), col=2, lwd=0.5)
```



This is less satisfactory than for the New York data, but does give a reasonable overview of the dataset. The `spacetime` package has a special data structure for "full grid" spatio-temporal data like this, and we can create such a `STF` object as follows.

```
head(stations)
```

```
SpatialPoints:
```

```
        coords.x1 coords.x2
DESH001  9.585911  53.67057
DENI063  9.685030  53.52418
DEUB038  9.791584  54.07312
DEBE056 13.647013  52.44775
DEBE062 13.296353  52.65315
DEBE032 13.225856  52.47309
Coordinate Reference System (CRS) arguments: +proj=longlat +datum=WGS84
+no_defs
```

```
head(dates)
```

```
[1] "1998-01-01" "1998-01-02" "1998-01-03" "1998-01-04" "1998-01-05"
[6] "1998-01-06"
```

```
rural = STFDF(stations, dates, data.frame(PM10 = as.vector(air)))
```

See `help(package="spacetime")` for further details about these kinds of data structures.

We are now in a position to think about how to apply our newly acquired spatio-temporal modelling skills to this dataset. Doing so is left as an exercise.

18.5 Wrap-up

This has been the briefest of introductions to spatio-temporal modelling. However, many of the most important concepts and issues have been touched upon, so this material will hopefully form a useful starting point for further study.

More generally, in this half of the module we have concentrated mainly on a dynamical approach to the modelling and analysis of temporal data, using model families such as ARMA, HMM and DLM. This dynamical view has many advantages over some more classical approaches to describing temporal data. Further, we have seen how this dynamical view often has computational advantages, typically leading to algorithms that have complexity that is linear in the number of time points. We have skimmed over many technical issues and details, and there is obviously a lot more to know. But again, I hope to have provided an intuitive introduction to many of the most important problems and concepts, and that you now have a better appreciation for random processes and data that evolve in (space and) time.

References

Chatfield, C., and H. Xing. 2019. *The Analysis of Time Series: An Introduction with R*. CRC Press.

Cressie, N. 2015. *Statistics for Spatial Data*. Wiley.

Cressie, N., and C. K. Wikle. 2015. *Statistics for Spatio-Temporal Data*. Wiley.

Gelfand, A. E., P. Diggle, P. Guttorp, and M. Fuentes. 2010. *Handbook of Spatial Statistics*. CRC Press.

Petris, G., S. Petrone, and P. Campagnoli. 2009. *Dynamic Linear Models with R*. Use R! New York: Springer.

Priestley, M. B. 1989. *Spectral Analysis and Time Series*. Academic Press.

Rasmussen, C. E., and C. K. I. Williams. 2005. *Gaussian Processes for Machine Learning*. MIT Press.

Ripley, B. D. 2005. *Spatial Statistics*. Wiley.

Rue, H., and L. Held. 2005. *Gaussian Markov Random Fields: Theory and Applications*. CRC Press.

Särkkä, S., and L. Svensson. 2023. *Bayesian Filtering and Smoothing*. Cambridge University Press.

Shumway, R. H., and D. S. Stoffer. 2017. *Time Series Analysis and Its Applications, with R Examples, Fourth Edition*. Springer.

West, M., and J. Harrison. 2013. *Bayesian Forecasting and Dynamic Models*. Springer.

Wikle, C. K., A. Zammit-Mangion, and N. Cressie. 2019. *Spatio-Temporal Statistics with R*. CRC Press.