9 Spatio-temporal models and data

9.1 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.

9.2 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.

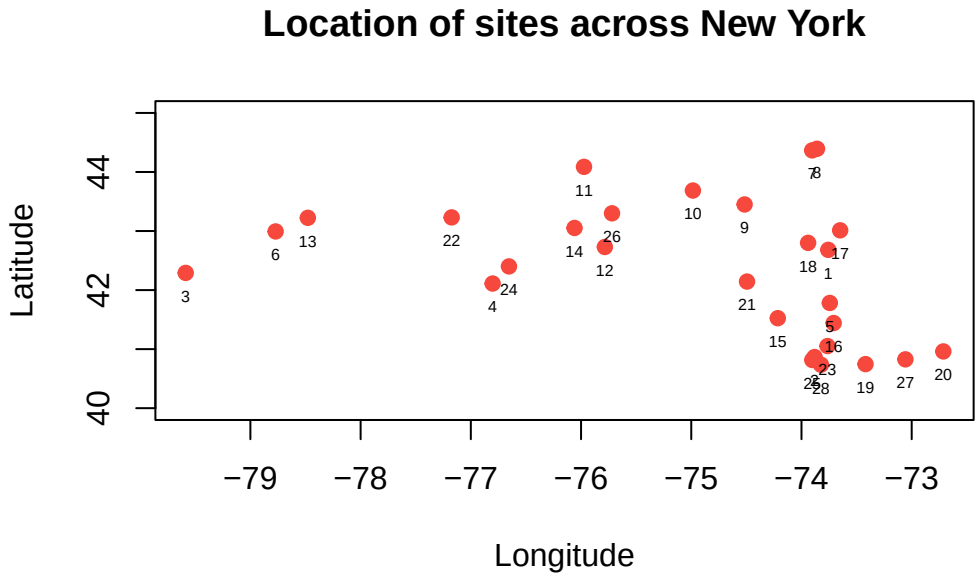
```
sites = unique(NYdata[,1:3])
dim(sites)
```

```
[1] 28  3
```

```
numSites = dim(sites)[1]
head(sites)
```

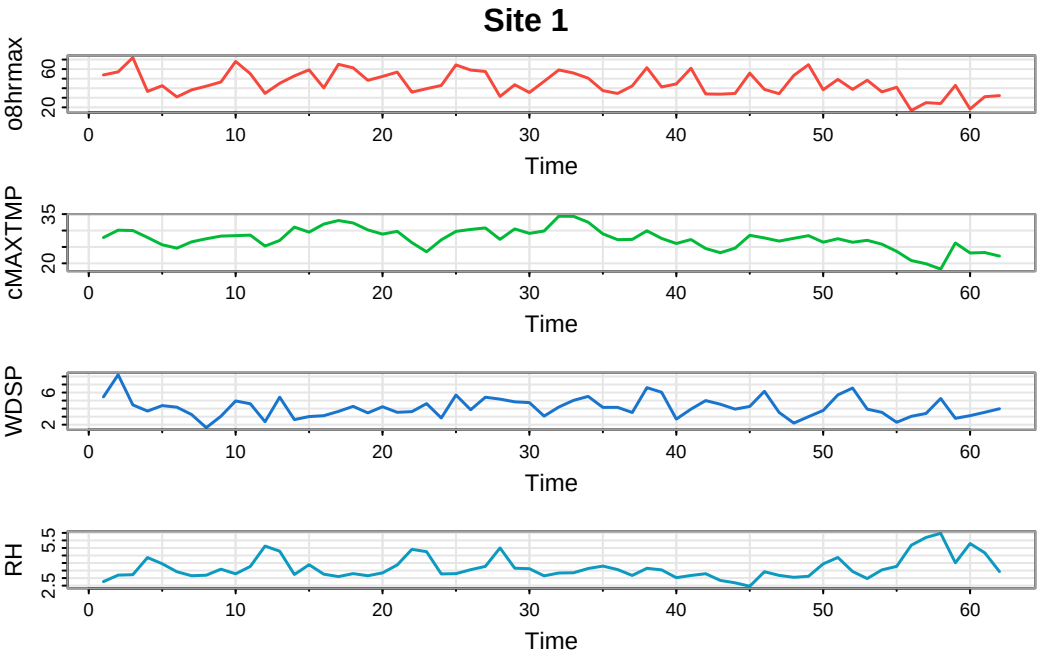
```
    s.index Longitude Latitude
1         1   -73.757   42.681
63        2   -73.881   40.866
125       3   -79.587   42.291
187       4   -76.802   42.111
249       5   -73.743   41.782
311       6   -78.771   42.993
```

```
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")
```

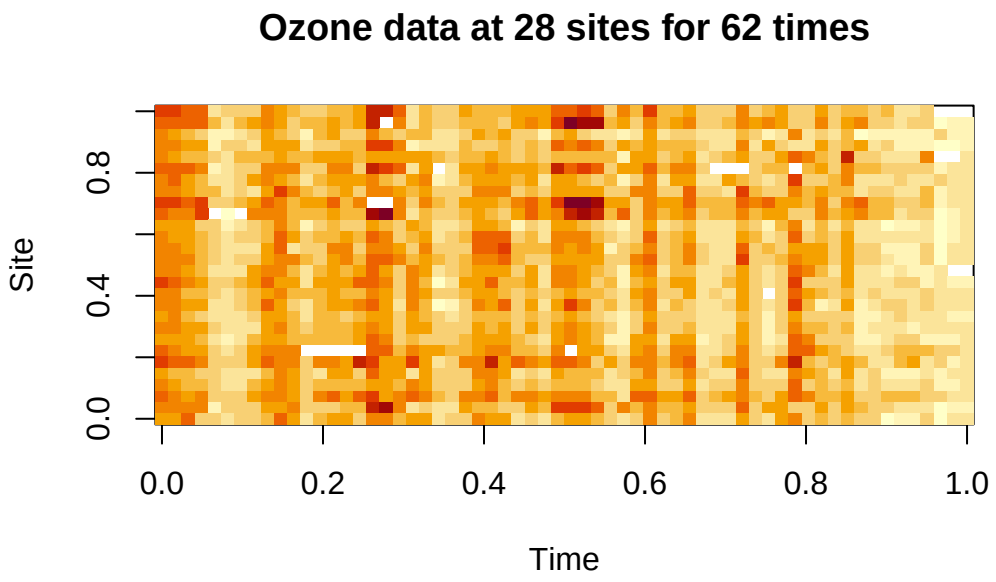


So the observations in space and time are actually multivariate, with several different variables begin measured simultaneously. To keep things simpler, we will focus on just one variable, ozone.

```
ozone = NYdata[,c("o8hrmax", "s.index")]
ozone = unstack(ozone)
dim(ozone) # "space-wide" format
```

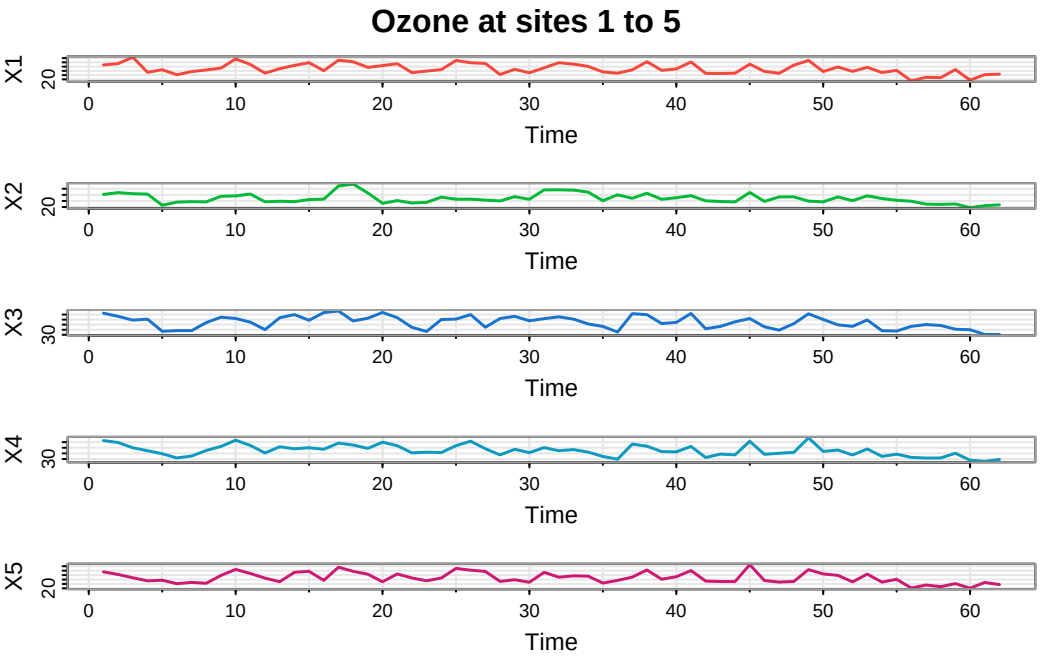
```
[1] 62 28
```

```
## "full grid" representation - "STF" in "spacetime" terminology
image(as.matrix(ozone),
  main="Ozone data at 28 sites for 62 times",
  xlab="Time", ylab="Site")
```



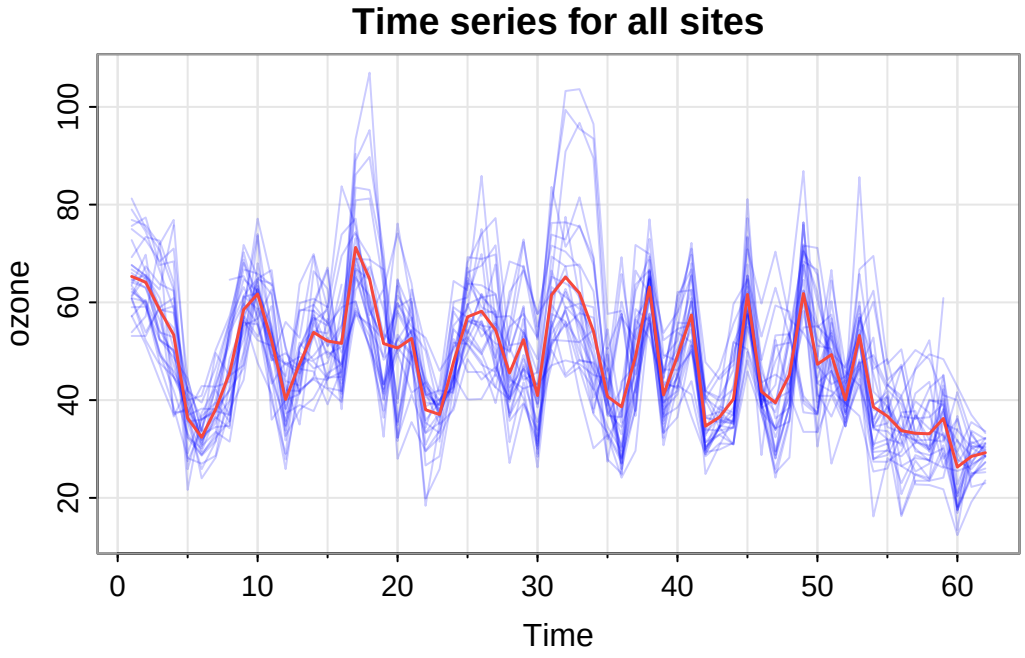
We can visualise the data matrix as an image, which shows a few missing observations (in white), but not too many. This is a fairly complete full-grid spatio-temporal dataset. We can do time series plots for a very small number of sites at once.

```
tsplot(ozone[,1:5], col=2:6, lwd=1.5,
  main="Ozone at sites 1 to 5")
```



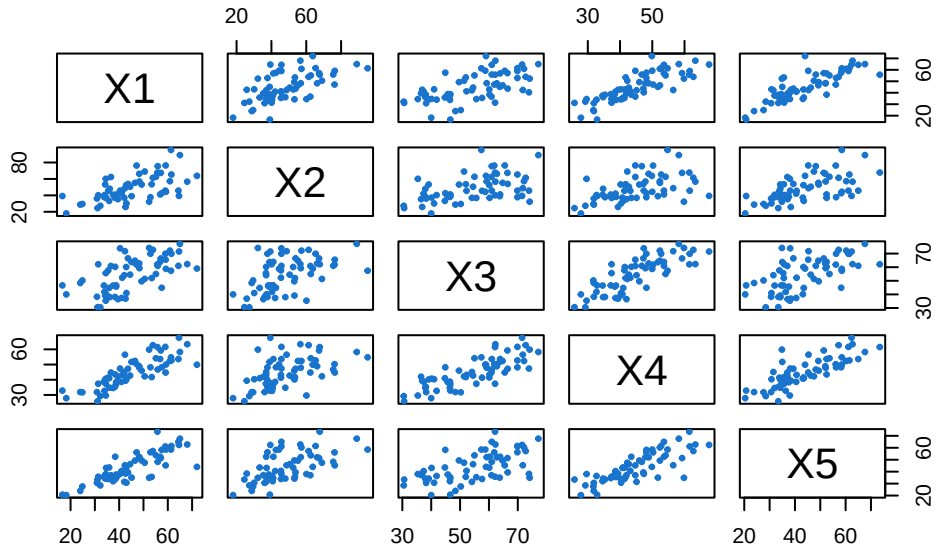
But to look at the time series for all sites simultaneously, we need to overlay, and use transparency to stop the plot from looking too messy.

```
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)
```

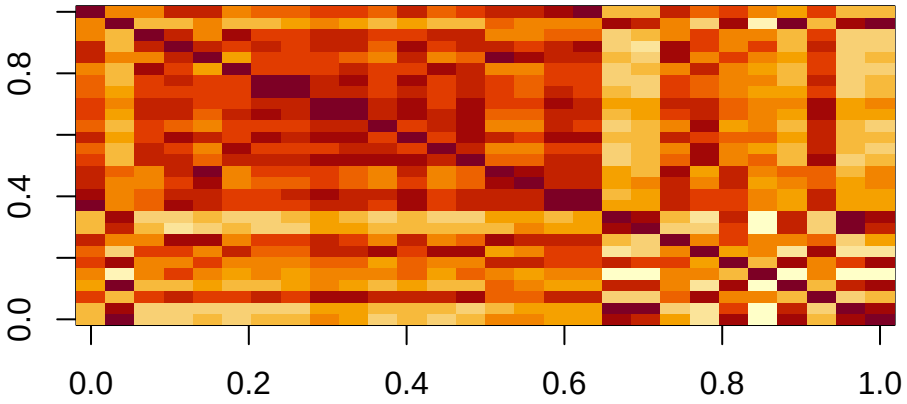


Similarly, we can look in detail at cross-correlations for a small number of time series, but need to revert to an image to see the full correlation structure.

```
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.

9.3 Spatio-temporal modelling

9.3.1 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.

9.3.1.1 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.

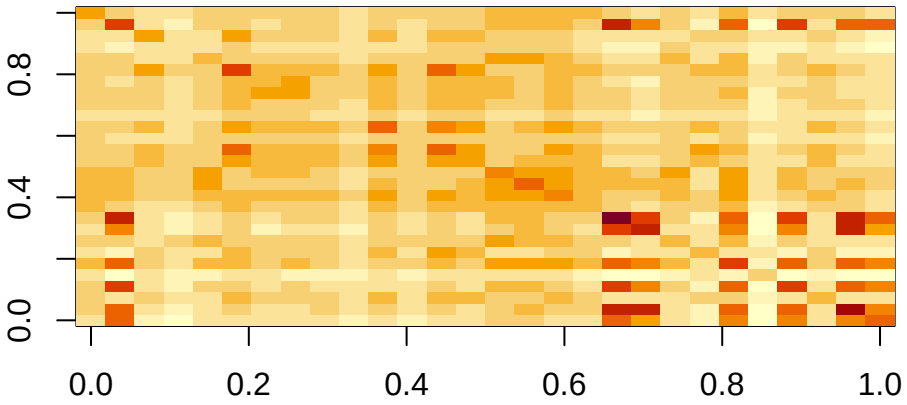
9.3.1.1.1 Unconstrained multivariate data

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])
```



9.3.1.1.2 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,
```

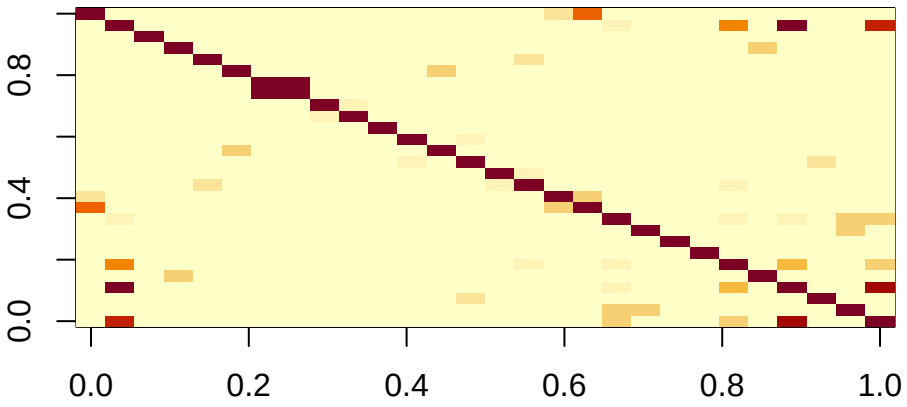
```
    function(x)
      mvtnorm::dmvnorm(x, sigma=cm(param), log=TRUE)),
        na.rm=TRUE)
```

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.

9.3.1.2 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.

9.3.2 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.

9.3.2.1 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.

9.3.2.2 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.

9.3.2.2.1 Unconstrained models

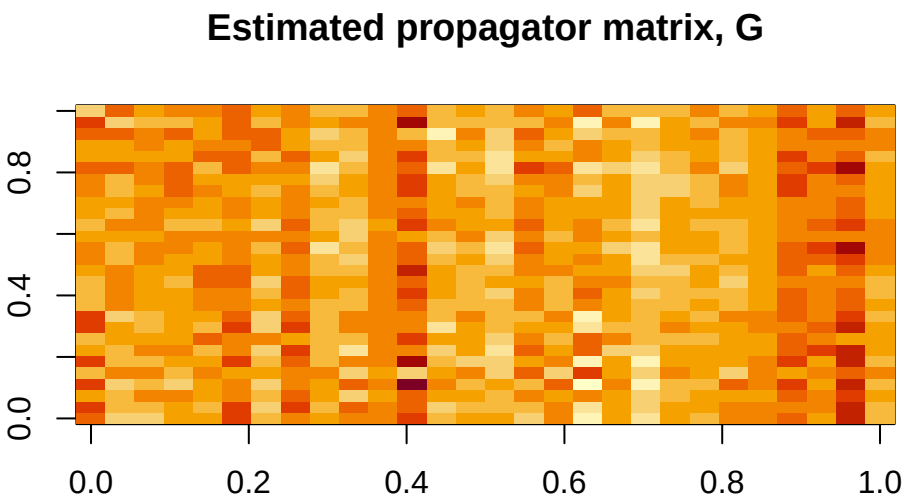
Obviously, one possibility is to consider a completely unconstrained G, with elements to be estimated by least squares (or maximum likelihood). We can use the same least squares approach that we adopted in Chapter 4 by writing our model in the form

$$\mathsf{X}_{2:n} = \mathsf{X}_{1:(n-1)}\mathsf{G}^\top + \mathsf{E},$$

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.

9.3.2.2.2 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.

9.3.2.2.3 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.

9.3.3 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.

9.3.3.1 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.

9.3.3.2 Example: NY ozone data

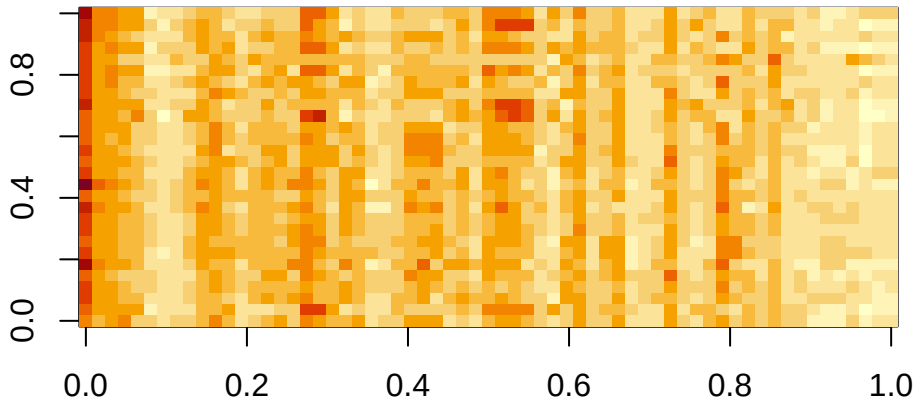
Let's see how we could implement a model like this using the `dlm` R package. To keep things simple we will just assume a propagator matrix of the form $\mathsf{G} = \alpha\mathbb{I}$. We will also start off with an evolution covariance matrix of the form $\mathsf{W} = \sigma_w^2\mathbb{I}$, but we will 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)
```

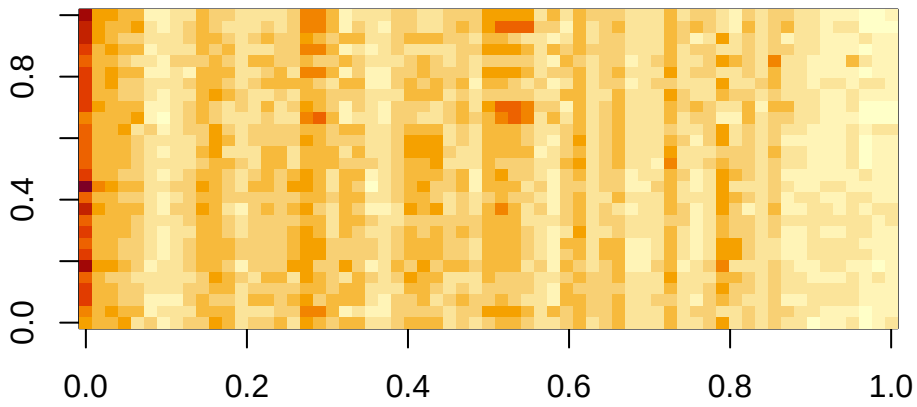


This works, and give smoothed states that are visibly smoother than the raw data. Importantly, it also sensibly smooths over the missing data. But this model isn't really explicitly spatial. Apart from assuming common parameters across sites, the DLMs at each site are independent. For spatial smoothing 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)
```



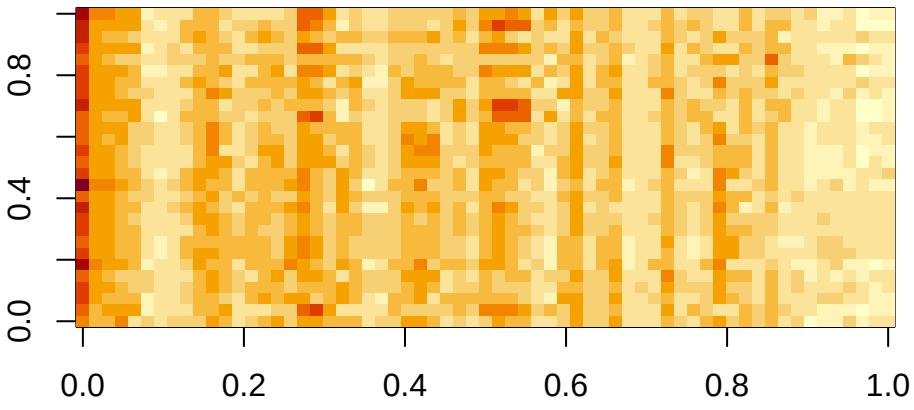
This also works, after a fashion, but note that the optimal length scale is just the upper bound I imposed on the optimiser. If an upper bound is not imposed, the length scale just keeps increasing until the model crashes. Inferring length scales is notoriously difficult in spatial statistics, and is 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.

9.3.3.3 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.

9.3.3.4 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.

9.3.4 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.

9.4 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.

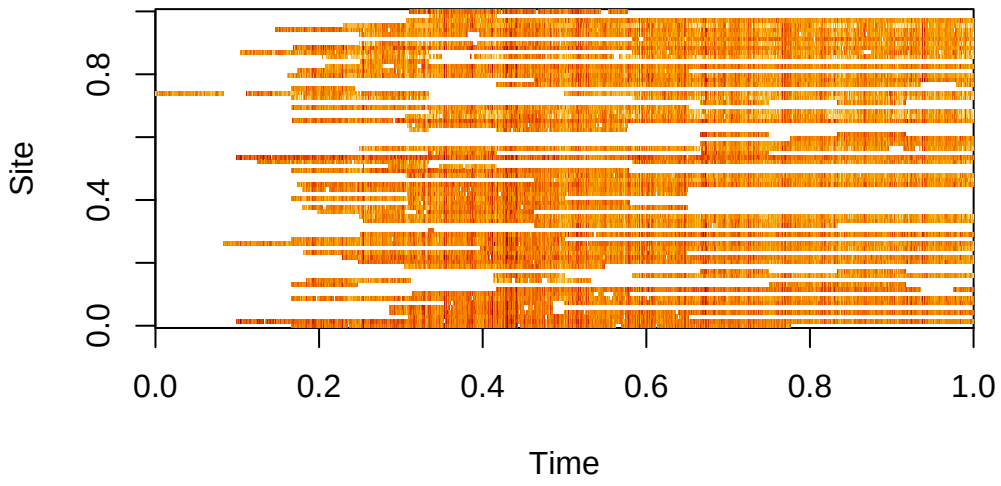
The data is air quality data (specifically, PM10 concentration), for 70 different sites in Germany, over a 10 year period. We can load and inspect the data as follows.

```
library(spacetime)
data(air)
dim(air) # "time-wide" format
```

```
space  time
   70  4383
```

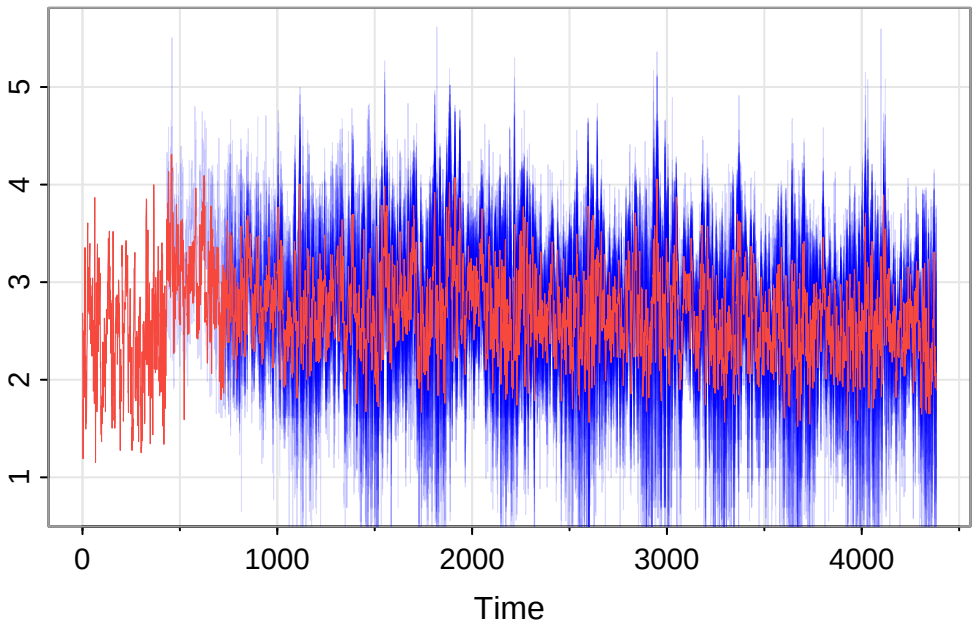
It is actually in "time-wide" format, which we don't necessarily want, and can be logged to be more Gaussian.

```
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.

9.5 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.