A Detailed Guide to spmodel

Michael Dumelle, Matt Higham, and Jay M. Ver Hoef

1 Introduction

`spmodel` is an **R** package used to fit, summarize, and predict for a variety of spatial statistical models. In Section 2, we give a brief theoretical introduction to spatial linear models. In Section 3, we outline the variety of methods used to estimate the parameters of spatial linear models. In Section 4, we explain how to obtain predictions at unobserved locations. In Section 5, we detail some advanced modeling features, including random effects, partition factors, anisotropy, and big data approaches. In Section 6, we end with a short discussion. Before proceeding, we load `spmodel` by running

```
library(spmodel)
```

If you use `spmodel` in a formal publication or report, please cite it. Citing `spmodel` lets us devote more resources to it in the future. We view the `spmodel` citation by running

```
citation(package = "spmodel")
```

```
#>
#> To cite spmodel in publications use:
#>
#>   Michael Dumelle, Matt Higham, and Jay M. Ver Hoef (2022). spmodel:
#>   Spatial Statistical Modeling and Prediction. R package version 0.1.1.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {spmodel: Spatial Statistical Modeling and Prediction},
#>     author = {Michael Dumelle and Matt Higham and Jay M. {Ver Hoef}},
#>     year = {2022},
#>     note = {R package version 0.1.1},
#>   }
```

- An overview of basic features in `spmodel`: `vignette("basics", "spmodel")`
- Technical details regarding many functions: `vignette("technical", "spmodel")`

We will create visualizations using ggplot2 (Wickham 2016), which we load by running

```
library(ggplot2)
```

ggplot2 is only installed alongside `spmodel` when `dependencies = TRUE` in `install.packages()`, so check that it is installed before reproducing any visualizations in this vignette.

2 The Spatial Linear Model

Statistical linear models are often parameterized as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{1}$$

where for a sample size n, $\mathbf{y}$ is an $n \times 1$ vector of response variables, $\mathbf{X}$ is an $n \times p$ design (model) matrix of explanatory variables, $\boldsymbol{\beta}$ is an $p \times 1$ vector of fixed effects controlling the impact of $\mathbf{X}$ on $\mathbf{y}$, and $\boldsymbol{\epsilon}$ is an $n \times 1$ vector of random errors. Typically, it is assumed that $\mathrm{E}(\boldsymbol{\epsilon}) = \mathbf{0}$ and $\mathrm{Cov}(\boldsymbol{\epsilon}) = \sigma^2_\epsilon \mathbf{I}$, where $\mathrm{E}(\cdot)$ denotes expectation, $\mathrm{Cov}(\cdot)$ denotes covariance, σ^2_ϵ denotes a variance parameter, and $\mathbf{I}$ denotes the identity matrix.

The model in Equation 1 assumes the elements of $\mathbf{y}$ are uncorrelated. Typically for spatial data, elements of $\mathbf{y}$ are correlated, as observations close together in space tend to be more similar than observations far apart (Tobler 1970). Failing to properly accommodate the spatial dependence in $\mathbf{y}$ can cause researchers to draw incorrect conclusions about their data. To accommodate spatial dependence in $\mathbf{y}$, an $n \times 1$ spatial random effect, $\boldsymbol{\tau}$, is added to Equation 1, yielding the model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\tau} + \boldsymbol{\epsilon}, \tag{2}$$

where $\mathrm{E}(\boldsymbol{\tau}) = \mathbf{0}$, $\mathrm{Cov}(\boldsymbol{\tau}) = \sigma^2_\tau \mathbf{R}$, and $\mathbf{R}$ is a matrix that determines the spatial dependence structure in $\mathbf{y}$ and depends on a range parameter, ϕ. The parameter σ^2_τ is called the spatially dependent random error variance or partial sill. The parameter σ^2_ϵ is called the spatially independent random error variance or nugget. These two variance parameters are henceforth more intuitively written as σ^2_{de} and σ^2_{ie}, respectively. The covariance of $\mathbf{y}$ is denoted $\boldsymbol{\Sigma}$ and given by $\sigma^2_{de}\mathbf{R} + \sigma^2_{ie}\mathbf{I}$. The parameters that compose this covariance are typically referenced by the vector $\boldsymbol{\theta}$, which is called the covariance parameter vector.

Equation 2 is called the spatial linear model. The spatial linear model applies to both point-referenced and areal data. Data are point-referenced when the elements in $\mathbf{y}$ are observed at point-locations indexed by x-coordinates and y-coordinates on a spatially continuous surface with an infinite number of locations. The `splm()` function is used to fit spatial linear models for point-referenced data (these are often called geostatistical models). One spatial covariance function available in `splm()` is the exponential spatial covariance function, which has an $\mathbf{R}$ matrix given by

$$\mathbf{R} = \exp(-\mathbf{h}/\phi),$$

where $\mathbf{h}$ is a matrix of Euclidean distances among observations. Recall ϕ is the range parameter, controlling the behavior of $\mathbf{R}$ as a function of distance. Parameterizations for `splm()` spatial covariance types and their $\mathbf{R}$ matrices can be seen by running `help("splm", "spmodel")` or `vignette("technical", "spmodel")`.

Data are areal when the elements in $\mathbf{y}$ are observed as part of a finite network of polygons whose connections are indexed by a neighborhood structure. For example, the polygons may represent counties in a state who are neighbors if they share at least one boundary. The `spautor()` function is used to fit spatial linear models for areal data (these are often called spatial autoregressive models). One spatial autoregressive covariance function available in `spautor()` is the simultaneous autoregressive spatial covariance function, which has an $\mathbf{R}$ matrix given by

$$\mathbf{R} = [(\mathbf{I} - \phi\mathbf{W})(\mathbf{I} - \phi\mathbf{W})^\top]^{-1},$$

where $\mathbf{W}$ is a weight matrix describing the neighborhood structure in $\mathbf{y}$. Paremertizations for `spautor()` spatial covariance types and their $\mathbf{R}$ matrices can be seen by running `help("spautor", "spmodel")` or `vignette("technical", "spmodel")`.

3 Model Fitting

In this section, we show how to use the `splm()` and `spautor()` functions to estimate parameters of the spatial linear model. We also explore diagnostic tools in `spmodel` that evaluate model fit. The `splm()` and `spautor()` functions share similar syntactic structure with the `lm()` function used to fit non-spatial linear models (linear models without spatial dependence) from Equation 1. `splm()` and `spautor()` generally require at least three arguments:

- `formula`: a formula that describes the relationship between the response variable ($\mathbf{y}$) and explanatory variables ($\mathbf{X}$)
  - `formula` in `splm()` is the same as `formula` in `lm()`

- `data`: a `data.frame` or `sf` object that contains the response variable, explanatory variables, and spatial information
- `spcov_type`: the spatial covariance type (`"exponential"`, `"matern"`, `"car"`, etc)

If `data` is an `sf` (Pebesma 2018) object, spatial information is stored in the object's geometry. If `data` is a `data.frame`, then for point-referenced data the x-coordinates and y-coordinates must be provided via the `xcoord` and `ycoord` arguments, and for areal data, the weight matrix must be provided via the `W` argument. Appendix A uses the `caribou` data, a `tibble` (a special `data.frame`), to show how to explicitly provide spatial information via `xcoord` and `ycoord` (in `splm()`) or `W` (in `spautor()`).

In the following subsections, we use the point-referenced `moss` data, an `sf` object that contains data on heavy metals in mosses near a mining road in Alaska. We view the first few rows of `moss` by running

```
moss
```

```
#> Simple feature collection with 365 features and 7 fields
#> Geometry type: POINT
#> Dimension:     XY
#> Bounding box:  xmin: -445884.1 ymin: 1929616 xmax: -383656.8 ymax: 2061414
#> Projected CRS: NAD83 / Alaska Albers
#> # A tibble: 365 x 8
#>    sample field_dup lab_rep year  sideroad log_dist2road log_Zn
#>    <fct>  <fct>     <fct>   <fct> <fct>            <dbl>  <dbl>
#>  1 001PR  1         1       2001  N                 2.68   7.33
#>  2 001PR  1         2       2001  N                 2.68   7.38
#>  3 002PR  1         1       2001  N                 2.54   7.58
#>  4 003PR  1         1       2001  N                 2.97   7.63
#>  5 004PR  1         1       2001  N                 2.72   7.26
#>  6 005PR  1         1       2001  N                 2.76   7.65
#>  7 006PR  1         1       2001  S                 2.30   7.59
#>  8 007PR  1         1       2001  N                 2.78   7.16
#>  9 008PR  1         1       2001  N                 2.93   7.19
#> 10 009PR  1         1       2001  N                 2.79   8.07
#> # ... with 355 more rows, and 1 more variable: geometry <POINT [m]>
```

We can learn more about `moss` by running `help("moss", "spmodel")`.

3.1 Estimation

Generally the covariance parameters ($\boldsymbol{\theta}$) and fixed effects ($\boldsymbol{\beta}$) of the spatial linear model require estimation. The default estimation method in `spmodel` is restricted maximum likelihood (Patterson and Thompson 1971; Harville 1977; Wolfinger, Tobias, and Sall 1994). Maximum likelihood estimation is also available. For point-referenced data, semivariogram weighted least squares (Cressie 1985) and semivariogram composite likelihood (Curriero and Lele 1999) are additional estimation methods. The estimation method is controlled using the `estmethod` argument.

To visualize the distribution of log zinc concentration in the `moss` data (Figure 1), run

```
ggplot(moss, aes(color = log_Zn)) +
  geom_sf(size = 2) +
  scale_color_viridis_c() +
  theme_gray(base_size = 14)
```

We estimate parameters of a spatial linear model regressing log zinc concentration (`log_Zn`) on log distance to a haul road (`log_dist2road`) using an exponential spatial covariance function by running

```
spmod <- splm(log_Zn ~ log_dist2road, moss, spcov_type = "exponential")
```

We summarize the model fit by running

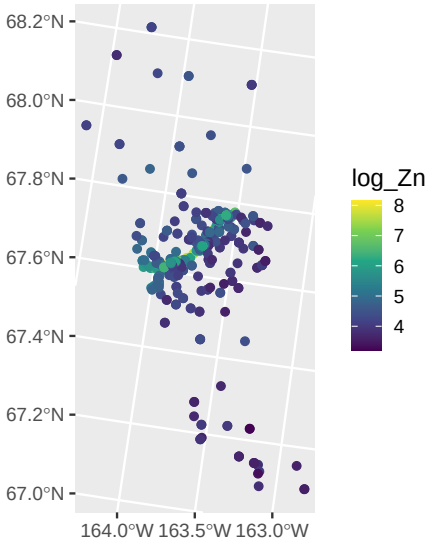

Figure 1: Distribution of log zinc concentration in the moss data.

```
summary(spmod)
```

```
#>
#> Call:
#> splm(formula = log_Zn ~ log_dist2road, data = moss, spcov_type = "exponential")
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -2.6801 -1.3606 -0.8103 -0.2485  1.1298
#>
#> Coefficients (fixed):
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    9.76825    0.25216   38.74   <2e-16 ***
#> log_dist2road -0.56287    0.02013  -27.96   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Pseudo R-squared: 0.683
#>
#> Coefficients ( spatial covariance):
#>        de        ie     range
#> 3.595e-01 7.897e-02 8.237e+03
```

The fixed effects coefficient table contains estimates, standard errors, z-statistics, and asymptotic p-values for each fixed effect. From this table, we notice there is evidence that log zinc concentration significantly decreases with distance from the haul road (p-value $<$ 2e-16). We see the fixed effect estimates by running

```
coef(spmod)
```

```
#>   (Intercept) log_dist2road
#>     9.7682525    -0.5628713
```

The model summary also contains the exponential spatial covariance parameter estimates, which we can view by running

```
coef(spmod, type = "spcov")
```

```
#>           de           ie        range       rotate        scale
#> 3.595316e-01 7.896824e-02 8.236712e+03 0.000000e+00 1.000000e+00
#> attr(,"class")
#> [1] "exponential"
```

The dependent random error variance (σ^2_{de}) is estimated to be approximately 0.36 and the independent random error variance (σ^2_{ie}) is estimated to be approximately 0.079. The range (ϕ) is estimated to be approximately 8,237. The effective range is the distance at which the spatial covariance is approximately zero. For the exponential covariance, the effective range is 3ϕ. This means that observations whose distance is greater than 24,711 meters are approximately uncorrelated. The `rotate` and `scale` parameters affect the modeling of anisotropy (Section 5.4). By default, they are assumed to be zero and one, respectively, which means that anisotropy is not modeled (i.e., the spatial covariance is assumed isotropic, or independent of direction). We plot the fitted spatial covariance function (Figure 2) by running

```
plot(spmod, which = 7)
```

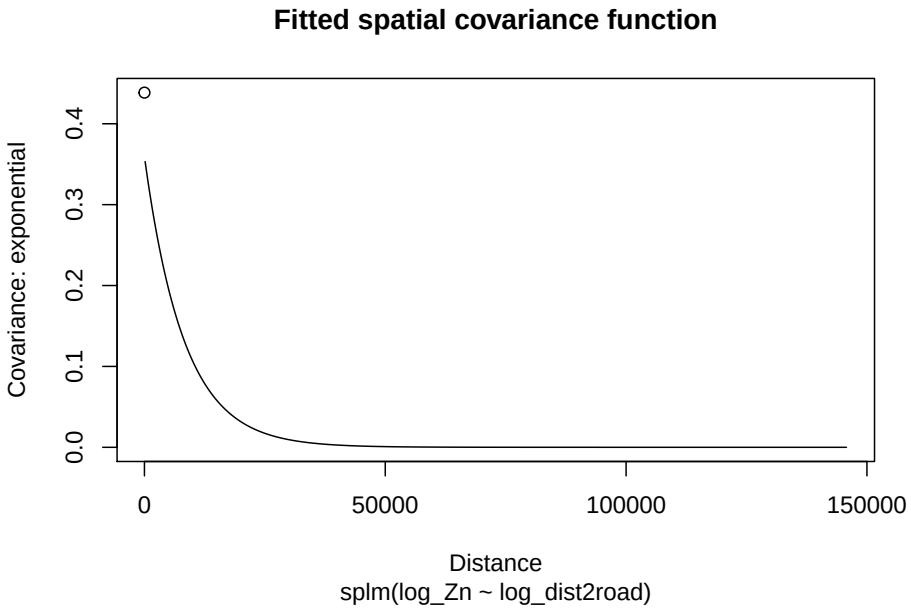

Figure 2: Empirical spatial covariance of fitted model.

We can learn more about the plots available for fitted models by running `help("plot.spmod", "spmodel")`.

3.2 Model-Fit Statistics

The quality of model fit can be assessed using a variety of statistics readily available in `spmodel`. The first model-fit statistic we consider is the pseudo R-squared. The pseudo R-squared is a generalization of the classical R-squared from non-spatial linear models that quantifies the proportion of variability in the data explained by the fixed effects. The pseudo R-squared is defined as

$$PR2 = 1 - \frac{\mathcal{D}(\hat{\boldsymbol{\Theta}})}{\mathcal{D}(\hat{\boldsymbol{\Theta}}_0)},$$

where $\mathcal{D}(\hat{\boldsymbol{\Theta}})$ is the deviance of the fitted model indexed by parameter vector $\hat{\boldsymbol{\Theta}}$ and $\mathcal{D}(\hat{\boldsymbol{\Theta}}_0)$ is the deviance of an intercept-only model indexed by parameter vector $\hat{\boldsymbol{\Theta}}_0$. We compute the pseudo R-squared by running

```
pseudoR2(spmod)
```

```
#> [1] 0.6829687
```

Roughly 68% of the variability in log zinc is explained by log distance from the road. The pseudo R-squared can be adjusted to account for the number of explanatory variables using the `adjust` argument. Pseudo R-squared (and the adjusted version) is most helpful for comparing models that have the same covariance structure.

The next two model-fit statistics we consider are the spatial AIC and AICc (Hoeting et al. 2006). The AIC and AICc evaluate the fit of a model with a penalty for the number of parameters estimated. This penalty balances model fit and model parsimony. The AICc is a correction to AIC for small sample sizes. As the sample size increases, the AIC and AICc become closer to one another. The lower the AIC and AICc, the better the balance of model fit and parsimony.

The spatial AIC and AICc are given by

$$\begin{aligned} \text{AIC} &= -2\ell(\hat{\boldsymbol{\Theta}}) + 2(|\hat{\boldsymbol{\Theta}}|) \\ \text{AICc} &= -2\ell(\hat{\boldsymbol{\Theta}}) + 2n(|\hat{\boldsymbol{\Theta}}|)/(n - |\hat{\boldsymbol{\Theta}}| - 1), \end{aligned}$$

where $\ell(\hat{\boldsymbol{\Theta}})$ is the log-likelihood of the data evaluated at the estimated parameter vector $\hat{\boldsymbol{\Theta}}$ maximizing $\ell(\boldsymbol{\Theta})$, and n is the sample size. For maximum likelihood, $\hat{\boldsymbol{\Theta}} = \{\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\beta}}\}$. For restricted maximum likelihood $\hat{\boldsymbol{\Theta}} = \{\hat{\boldsymbol{\theta}}\}$. AIC comparisons between a model fit using restricted maximum likelihood and a model fit using maximum likelihood are meaningless, as the models are fit with different likelihoods. AIC comparisons between models fit using restricted maximum likelihood are only valid when the models have the same fixed effect structure. In contrast, AIC comparisons between models fit using maximum likelihood are valid when the models have different fixed effect structures.

Suppose we want to quantify the difference in model quality between the spatial model and a non-spatial model using the AIC and AICc criteria. We fit a non-spatial model (Equation 1) in `spmodel` by running

```
lmod <- splm(log_Zn ~ log_dist2road, moss, spcov_type = "none")
```

We compute the spatial AIC and AICc of the spatial model and non-spatial model by running

```
AIC(spmod, lmod)
```

```
#>       df      AIC
#> spmod  3 373.2089
#> lmod   1 636.0635
```

```
AICc(spmod, lmod)
```

```
#>       df     AICc
#> spmod  3 373.2754
#> lmod   1 636.0745
```

The noticeably lower AIC and AICc of of the spatial model indicate that it is a better fit to the data than the non-spatial model.

Another approach to comparing the model fits is to perform leave-one-out cross validation. In leave-one-out cross validation, a single observation is removed from the data, the model is re-fit, and a prediction is made for the held-out observation. Then a loss metric like mean-squared-prediction error is computed and used to evaluate model fit. The lower the mean-squared-prediction error, the better the model fit. For computational efficiency, leave-one-out cross validation in `spmodel` is performed by first estimating $\boldsymbol{\theta}$ using all the data and then re-estimating $\boldsymbol{\beta}$ for each observation. We perform leave-one-out cross validation for the spatial and non-spatial model by running

```
loocv(spmod)
```

```
#> [1] 0.1110895
```

```
loocv(lmod)
```

```
#> [1] 0.3237897
```

The noticeably lower mean-squared-prediction error of the spatial model indicates that it is a better fit to the data than the non-spatial model.

3.3 Diagnostics

An observation is said to have high leverage if its combination of explanatory variable values is far from the mean vector of the explanatory variables. For a non-spatial model, the leverage of the ith observation is the ith diagonal element of the hat matrix given by

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top.$$

For a spatial model, the leverage of the ith observation is the ith diagonal element of the spatial hat matrix given by

$$\mathbf{H}^* = (\mathbf{X}^*(\mathbf{X}^{*\top}\mathbf{X})^{-1}\mathbf{X}^{*\top}),$$

where $\mathbf{X}^* = \boldsymbol{\Sigma}^{-1/2}\mathbf{X}$ and $\boldsymbol{\Sigma}^{-1/2}$ is the matrix square root of the covariance matrix, $\boldsymbol{\Sigma}$ (Montgomery, Peck, and Vining 2021). The spatial hat matrix can be viewed as the non-spatial hat matrix applied to the "whitened" $\mathbf{X}$ matrix, $\mathbf{X}^*$. We compute the hat values (leverage) by running

```
hatvalues(spmod)
```

The larger the hat value, the larger the leverage.

The fitted value of an observation is the estimated mean response given the observation's explanatory variable values and the model fit:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}.$$

We compute the fitted values by running

```
fitted(spmod)
```

Fitted values for the spatially dependent random errors ($\boldsymbol{\tau}$), spatially independent random errors ($\boldsymbol{\epsilon}$), and random effects can also be obtained via `fitted()` by changing the `type` argument.

The residuals act as estimates of errors, measuring each response's deviation from its fitted value. The raw residuals are given by

$$\mathbf{e}_r = \mathbf{y} - \hat{\mathbf{y}}.$$

We compute the raw residuals of the spatial model by running

```
residuals(spmod)
```

The raw residuals are typically not directly checked for linear model assumptions, as they have covariance closely resembling the covariance of $\mathbf{y}$. These residuals can be "whitened" by pre-multiplying by $\hat{\boldsymbol{\Sigma}}^{-1/2}$, yielding the Pearson residuals:

$$\mathbf{e}_p = \hat{\boldsymbol{\Sigma}}^{-1/2}\mathbf{e}_r.$$

When the model is correct, the Pearson residuals have mean zero, variance approximately one, and are uncorrelated. We compute the Pearson residuals of the spatial model by running

```
residuals(spmod, type = "pearson")
```

The covariance of $\mathbf{e}_p$ is $(\mathbf{I} - \mathbf{H}^*)$, which is approximately $\mathbf{I}$ for large sample sizes. Explicitly dividing $\mathbf{e}_p$ by the respective diagonal element of $(\mathbf{I} - \mathbf{H}^*)$ yields the standardized residuals:

$$\mathbf{e}_s = \frac{\mathbf{e}_p}{\sqrt{(1 - diag(\mathbf{H}^*))}},$$

where $diag(\mathbf{H}^*)$ denotes the diagonal of $\mathbf{H}^*$.

We compute the standardized residuals of the spatial model by running

```
residuals(spmod, type = "standardized")
```

or

```
rstandard(spmod)
```

When the model is correct, the standardized residuals have mean zero, variance one, and are uncorrelated. It is common to check linear model assumptions through visualizations. We can plot the standardized residuals vs fitted values by running

```
plot(spmod, which = 1) # figure omitted
```

When the model is correct, the standardized residuals should be evenly spread around zero with no discernible pattern. We can plot a normal QQ-plot of the standardized residuals by running

```
plot(spmod, which = 2) # figure omitted
```

When the standardized residuals are normally distributed, they should closely follow the normal QQ-line.

An observation is said to be influential if its omission has a large impact on model fit. Typically this is measured using Cook's distance (Cook and Weisberg 1982). For the non-spatial model, the Cook's distance of the ith observation is denoted $\mathbf{D}$ and given by

$$\mathbf{D} = \mathbf{e}_s^2 \frac{diag(\mathbf{H})}{p(1 - diag(\mathbf{H}))}.$$

For a spatial model, the Cook's distance of the ith observation is denoted $\mathbf{D}^*$ and given by

$$\mathbf{D}^* = \mathbf{e}_s^2 \frac{diag(\mathbf{H}^*)}{p(1 - diag(\mathbf{H}^*))}.$$

The larger the Cook's distance, the larger the influence. We compute Cook's distance by running

```
cooks.distance(spmod)
```

We visualize the Cook's distance versus leverage (hat values) by running

```
plot(spmod, which = 6) # figure omitted
```

3.4 The broom functions: `tidy()`, `glance()`, and `augment()`

The `tidy()`, `glance()`, and `augment()` functions from the broom **R** package (Robinson, Hayes, and Couch 2021) are defined for `spmodel` objects. The `tidy()` function returns a tidy tibble of the coefficient table from `summary()`:

```
tidy(spmod)
```

```
#> # A tibble: 2 x 5
#>   term          estimate std.error statistic p.value
#>   <chr>            <dbl>     <dbl>     <dbl>   <dbl>
#> 1 (Intercept)      9.77     0.252       38.7        0
#> 2 log_dist2road   -0.563    0.0201     -28.0        0
```

This tibble format makes it easy to pull out the coefficient names, estimates, standard errors, z-statistics, and p-values from the `summary()` output. The `glance()` function returns a tidy tibble of model-fit statistics:

```
glance(spmod)
```

```
#> # A tibble: 1 x 9
#>       n     p  npar value   AIC  AICc logLik deviance pseudo.r.squared
#>   <int> <dbl> <int> <dbl> <dbl> <dbl>  <dbl>    <dbl>            <dbl>
#> 1   365     2     3  367.  373.  373.  -184.      363            0.683
```

The `glances()` function can be used to look at many models simultaneously:

```
glances(spmod, lmod)
```

```
#> # A tibble: 2 x 10
#>   model     n     p  npar value   AIC  AICc logLik deviance pseudo.r.squared
#>   <chr> <int> <dbl> <int> <dbl> <dbl> <dbl>  <dbl>    <dbl>            <dbl>
#> 1 spmod   365     2     3  367.  373.  373.  -184.     363             0.683
#> 2 lmod    365     2     1  634.  636.  636.  -317.     363.            0.671
```

The `augment()` function augments the original data with model diagnostics:

```
augment(spmod)
```

```
#> Simple feature collection with 365 features and 7 fields
#> Geometry type: POINT
#> Dimension:     XY
#> Bounding box:  xmin: -445884.1 ymin: 1929616 xmax: -383656.8 ymax: 2061414
#> Projected CRS: NAD83 / Alaska Albers
#> # A tibble: 365 x 8
#>    log_Zn log_dist2road .fitted .resid    .hat  .cooksd .std.resid
#>  * <dbl>         <dbl>   <dbl>  <dbl>   <dbl>    <dbl>      <dbl>
#>  1   7.33          2.68    8.26 -0.928 0.102   0.112        -1.48
#>  2   7.38          2.68    8.26 -0.880 0.0101  0.000507     -0.316
#>  3   7.58          2.54    8.34 -0.755 0.0170  0.000475     -0.236
#>  4   7.63          2.97    8.09 -0.464 0.0137  0.000219      0.178
#>  5   7.26          2.72    8.24 -0.977 0.0177  0.00515      -0.762
#>  6   7.65          2.76    8.21 -0.568 0.0147  0.000929     -0.355
#>  7   7.59          2.30    8.47 -0.886 0.0170  0.00802      -0.971
#>  8   7.16          2.78    8.20 -1.05  0.0593  0.0492       -1.29
#>  9   7.19          2.93    8.12 -0.926 0.00793 0.000451     -0.337
#> 10   8.07          2.79    8.20 -0.123 0.0265  0.00396       0.547
#> # ... with 355 more rows, and 1 more variable: geometry <POINT [m]>
```

By default, only the columns of `data` used to fit the model are returned alongside the diagnostics. All original columns of `data` are returned by setting `drop` to `FALSE`.

3.5 An Areal Data Example

Next we use the areal `seal` data, an `sf` object that contains the log of the estimated harbor-seal trends from abundance data across polygons in Alaska. We view the first few rows of `seal` by running

```
seal
```

```
#> Simple feature collection with 62 features and 1 field
#> Geometry type: POLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 913618.8 ymin: 1007542 xmax: 1116002 ymax: 1145054
#> Projected CRS: NAD83 / Alaska Albers
```

```
#> # A tibble: 62 x 2
#>    log_trend                                                      geometry
#>        <dbl>                                                  <POLYGON [m]>
#>  1  NA       ((1035002 1054710, 1035002 1054542, 1035002 1053542, 1035002 10525~
#>  2  -0.282   ((1037002 1039492, 1037006 1039490, 1037017 1039492, 1037035 10394~
#>  3  -0.00121 ((1070158 1030216, 1070185 1030207, 1070187 1030207, 1070211 10302~
#>  4   0.0354  ((1054906 1034826, 1054931 1034821, 1054936 1034822, 1055001 10348~
#>  5  -0.0160  ((1025142 1056940, 1025184 1056889, 1025222 1056836, 1025256 10567~
#>  6   0.0872  ((1026035 1044623, 1026037 1044605, 1026072 1044610, 1026083 10446~
#>  7  -0.266   ((1100345 1060709, 1100287 1060706, 1100228 1060706, 1100170 10607~
#>  8   0.0743  ((1030247 1029637, 1030248 1029637, 1030265 1029642, 1030328 10296~
#>  9  NA       ((1043093 1020553, 1043097 1020550, 1043101 1020550, 1043166 10205~
#> 10  -0.00961 ((1116002 1024542, 1116002 1023542, 1116002 1022542, 1116002 10215~
#> # ... with 52 more rows
```

We can learn more about the data by running `help("seal", "spmodel")`.

To visualize the distribution of log seal trends in the `seal` data (Figure 3), run

```
ggplot(seal, aes(fill = log_trend)) +
  geom_sf(size = 0.75) +
  scale_fill_viridis_c() +
  theme_gray(base_size = 14)
```

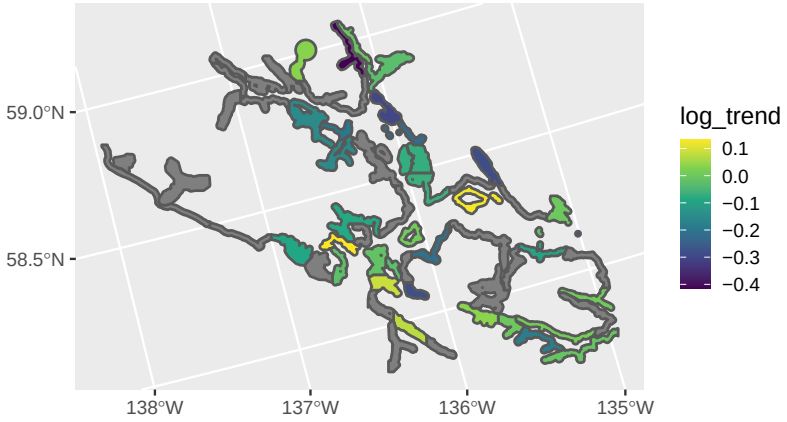

Figure 3: Distribution of log seal trends in the seal data. Polygons are gray where seal trends were not available.

Polygons are considered neighbors if they share at least one boundary. Polygons are gray if the log trend is missing. It is important to keep these missing observations in the data while fitting the model to preserve the neighborhood structure.

We estimate parameters of a spatial autoregressive model regressing log seal trends (`log_trend`) on an intercept using a conditional autoregressive (CAR) spatial covariance by running

```
sealmod <- spautor(log_trend ~ 1, seal, spcov_type = "car")
```

By default, `spautor()` calculates the weight matrix internally by defining observations as neighbors if they share at least one border (observations are not neighbors with themselves), uses row standardization, and

assumes `ie` equals zero.

We tidy and glance at the fitted model and augment the data by running

```
tidy(sealmod)
```

```
#> # A tibble: 1 x 5
#>   term        estimate std.error statistic p.value
#>   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
#> 1 (Intercept)  -0.0710    0.0250     -2.85 0.00443
```

```
glance(sealmod)
```

```
#> # A tibble: 1 x 9
#>       n     p  npar value   AIC  AICc logLik deviance pseudo.r.squared
#>   <int> <dbl> <int> <dbl> <dbl> <dbl>  <dbl>    <dbl>             <dbl>
#> 1    34     1     3 -36.9 -30.9 -30.1   18.4     32.9                 0
```

```
augment(sealmod)
```

```
#> Simple feature collection with 34 features and 6 fields
#> Geometry type: POLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 980001.5 ymin: 1010815 xmax: 1116002 ymax: 1145054
#> Projected CRS: NAD83 / Alaska Albers
#> # A tibble: 34 x 7
#>    log_trend .fitted  .resid   .hat .cooksd .std.resid                 geometry
#>  *     <dbl>   <dbl>   <dbl>  <dbl>   <dbl>      <dbl>             <POLYGON [m]>
#>  1  -0.282   -0.0710 -0.211  0.0179 0.0233      -1.14  ((1037002 1039492, 10370~
#>  2  -0.00121 -0.0710  0.0698 0.0699 0.0412       0.767 ((1070158 1030216, 10701~
#>  3   0.0354  -0.0710  0.106  0.0218 0.0109       0.705 ((1054906 1034826, 10549~
#>  4  -0.0160  -0.0710  0.0550 0.0343 0.00633      0.430 ((1025142 1056940, 10251~
#>  5   0.0872  -0.0710  0.158  0.0229 0.0299       1.14  ((1026035 1044623, 10260~
#>  6  -0.266   -0.0710 -0.195  0.0280 0.0493      -1.33  ((1100345 1060709, 11002~
#>  7   0.0743  -0.0710  0.145  0.0480 0.0818       1.30  ((1030247 1029637, 10302~
#>  8  -0.00961 -0.0710  0.0614 0.0143 0.00123      0.293 ((1116002 1024542, 11160~
#>  9  -0.182   -0.0710 -0.111  0.0131 0.0155      -1.09  ((1079864 1025088, 10798~
#> 10   0.00351 -0.0710  0.0745 0.0340 0.0107       0.561 ((1110363 1037056, 11103~
#> # ... with 24 more rows
```

4 Prediction

In this section, we show how to use `predict()` to perform spatial prediction (also called Kriging) in `spmodel`. We fit a model using the point-referenced `sulfate` data, an `sf` object that contains sulfate measurements in the conterminous United States. We make predictions for each location in the point-referenced `sulfate_preds` data, an `sf` object that contains locations in the conterminous United States at which to predict sulfate. We visualize the distribution of the sulfate data (Figure 4, left) by running

```
ggplot(sulfate, aes(color = sulfate)) +
  geom_sf(size = 2.5) +
  scale_color_viridis_c(limits = c(0, 45)) +
  theme_gray(base_size = 18)
```

We fit a spatial linear model regressing sulfate on an intercept using a spherical spatial covariance function by running

```
sulfmod <- splm(sulfate ~ 1, sulfate, spcov_type = "spherical")
```

Then we obtain best linear unbiased predictions (Kriging predictions) using `predict()`, where the `newdata` argument contains the locations at which to predict:

```
predict(sulfmod, newdata = sulfate_preds)
```

We make and predictions at the locations in `sulfate_preds` and store them as a new variable called `preds` in the `sulfate_preds` data set and then visualize them (Figure 4, right) by running

```
sulfate_preds$preds <- predict(sulfmod, newdata = sulfate_preds)
ggplot(sulfate_preds, aes(color = preds)) +
  geom_sf(size = 2.5) +
  scale_color_viridis_c(limits = c(0, 45)) +
  theme_gray(base_size = 18)
```

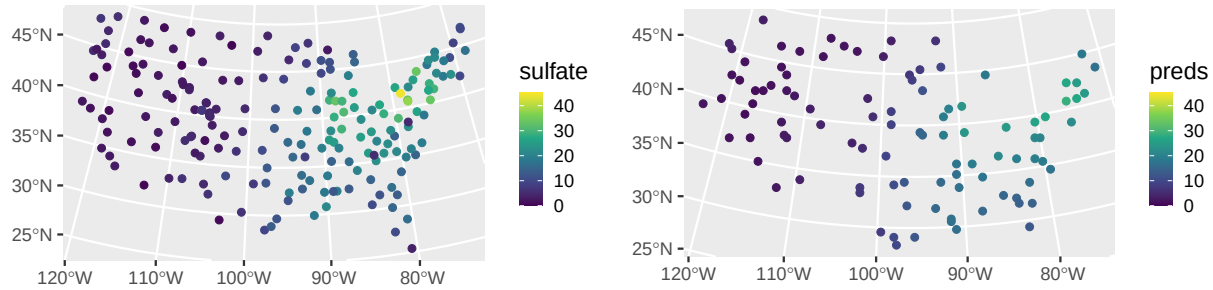

Figure 4: Distribution of observed sulfate (left) and sulfate predictions (right) in the conterminous United States.

If explanatory variables were used to fit the model, the same explanatory variables must be included in `newdata` with the same names they have in `data`. If `data` is a `data.frame`, coordinates must be included in `newdata` with the same names as they have in `data`. If `data` is an `sf` object, coordinates must be included in `newdata` with the same geometry name as they have in `data`. When using projected coordinates, the projection for `newdata` should be the same as the projection for `data`.

Prediction standard errors are returned by setting the `se.fit` argument to `TRUE`:

```
predict(sulfmod, newdata = sulfate_preds, se.fit = TRUE)
```

The `interval` argument determines the type of interval returned. If `interval` is `"none"` (the default), no interval is returned. If `interval` is `"prediction"`, a `100 * level`% prediction interval is returned (the default is a 95% prediction interval):

```
predict(sulfmod, newdata = sulfate_preds, interval = "prediction")
```

If `interval` is `"confidence"`, the predictions are instead the estimated mean given each observation's explanatory variable values. The corresponding `100 * level`% confidence interval is returned:

```
predict(sulfmod, newdata = sulfate_preds, interval = "confidence")
```

Previously we used the `augment()` function to augment `data` with model diagnostics. We can also use `augment()` to augment `newdata` with predictions, standard errors, and intervals. We remove the model predictions from `sulfate_preds` before showing how `augment()` is used to obtain the same predictions by running

```
sulfate_preds$preds <- NULL
```

We then view the first few rows of `sulfate_preds` augmented with a 90% prediction interval by running

```
augment(sulfmod, newdata = sulfate_preds, interval = "prediction", level = 0.90)
```

```
#> Simple feature collection with 100 features and 3 fields
#> Geometry type: POINT
#> Dimension:     XY
#> Bounding box:  xmin: -2283774 ymin: 582930.5 xmax: 1985906 ymax: 3037173
#> Projected CRS: NAD83 / Conus Albers
#> # A tibble: 100 x 4
#>    .fitted .lower .upper            geometry
#>  *   <dbl>  <dbl>  <dbl>         <POINT [m]>
#>  1    1.40  -5.33   8.14  (-1771413 1752976)
#>  2   24.5   18.2   30.8    (1018112 1867127)
#>  3    8.99   2.36  15.6  (-291256.8 1553212)
#>  4   16.4    9.92  23.0    (1274293 1267835)
#>  5    4.91  -1.56  11.4  (-547437.6 1638825)
#>  6   26.7   20.4   33.0    (1445080 1981278)
#>  7    3.00  -3.65   9.66  (-1629090 3037173)
#>  8   14.3    7.97  20.6    (1302757 1039534)
#>  9    1.49  -5.08   8.06  (-1429838 2523494)
#> 10   14.4    7.97  20.8    (1131970 1096609)
#> # ... with 90 more rows
```

Here `.fitted` represents the predictions.

Alternatively, we can include missing responses (`NA` values) in the data used for model fitting. The missing responses will be ignored for model fitting but stored in the fitted model object. We can add a column of missing responses to `sulfate_preds`, bind it together with `sulfate`, and re-fit the model by running

```
sulfate_preds$sulfate <- NA
sulfate_with_NA <- rbind(sulfate, sulfate_preds)
sulfmod_with_NA <- splm(sulfate ~ 1, sulfate_with_NA, "spherical")
```

We can then make predictions at the observations with missing responses by running

```
predict(sulfmod_with_NA)
```

This call to `predict()` finds in `sulfmod_with_NA` the `newdata` object, which is a subset of `data` that contains only the observations with missing responses:

```
sulfmod_with_NA$newdata
```

```
#> Simple feature collection with 100 features and 1 field
#> Geometry type: POINT
#> Dimension:     XY
#> Bounding box:  xmin: -2283774 ymin: 582930.5 xmax: 1985906 ymax: 3037173
#> Projected CRS: NAD83 / Conus Albers
#> First 10 features:
#>     sulfate                  geometry
#> 198      NA  POINT (-1771413 1752976)
#> 199      NA   POINT (1018112 1867127)
#> 200      NA POINT (-291256.8 1553212)
#> 201      NA   POINT (1274293 1267835)
#> 202      NA POINT (-547437.6 1638825)
#> 203      NA   POINT (1445080 1981278)
```

```
#> 204      NA  POINT (-1629090 3037173)
#> 205      NA   POINT (1302757 1039534)
#> 206      NA  POINT (-1429838 2523494)
#> 207      NA   POINT (1131970 1096609)
```

We can also use `augment()` to make the predictions by running

```
augment(sulfmod_with_NA, newdata = sulfmod_with_NA$newdata)
```

```
#> Simple feature collection with 100 features and 2 fields
#> Geometry type: POINT
#> Dimension:     XY
#> Bounding box:  xmin: -2283774 ymin: 582930.5 xmax: 1985906 ymax: 3037173
#> Projected CRS: NAD83 / Conus Albers
#> # A tibble: 100 x 3
#>    sulfate .fitted            geometry
#>  *   <dbl>   <dbl>         <POINT [m]>
#>  1      NA    1.40  (-1771413 1752976)
#>  2      NA   24.5    (1018112 1867127)
#>  3      NA    8.99 (-291256.8 1553212)
#>  4      NA   16.4    (1274293 1267835)
#>  5      NA    4.91 (-547437.6 1638825)
#>  6      NA   26.7    (1445080 1981278)
#>  7      NA    3.00  (-1629090 3037173)
#>  8      NA   14.3    (1302757 1039534)
#>  9      NA    1.49  (-1429838 2523494)
#> 10      NA   14.4    (1131970 1096609)
#> # ... with 90 more rows
```

Recall that omitting the `newdata` argument and running `augment(sulfmod_with_NA)` returns model diagnostics, not predictions.

For areal data, predictions cannot be computed at locations that were not incorporated in the neighborhood structure used to fit the model. Thus predictions are only possible for observations in `data` whose response values are missing (`NA`), as their locations are incorporated into the neighborhood structure. For example, we make predictions of log seal trends at the missing polygons from Figure 3 by running

```
predict(sealmod)
```

We can also use `augment()` to make the predictions:

```
augment(sealmod, newdata = sealmod$newdata)
```

```
#> Simple feature collection with 28 features and 2 fields
#> Geometry type: POLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 913618.8 ymin: 1007542 xmax: 1115097 ymax: 1132682
#> Projected CRS: NAD83 / Alaska Albers
#> # A tibble: 28 x 3
#>    log_trend .fitted                                                geometry
#>  *     <dbl>   <dbl>                                             <POLYGON [m]>
#>  1        NA -0.113  ((1035002 1054710, 1035002 1054542, 1035002 1053542, 10350~
#>  2        NA -0.0108 ((1043093 1020553, 1043097 1020550, 1043101 1020550, 10431~
#>  3        NA -0.0608 ((1099737 1054310, 1099752 1054262, 1099788 1054278, 10998~
#>  4        NA -0.0383 ((1099002 1036542, 1099134 1036462, 1099139 1036431, 10991~
#>  5        NA -0.0730 ((1076902 1053189, 1076912 1053179, 1076931 1053179, 10769~
#>  6        NA -0.0556 ((1070501 1046969, 1070317 1046598, 1070308 1046542, 10703~
#>  7        NA -0.0968 ((1072995 1054942, 1072996 1054910, 1072997 1054878, 10729~
```

```
#>  8        NA -0.0716 ((960001.5 1127667, 960110.8 1127542, 960144.1 1127495, 96~
#>  9        NA -0.0776 ((1031308 1079817, 1031293 1079754, 1031289 1079741, 10312~
#> 10        NA -0.0647 ((998923.7 1053647, 998922.5 1053609, 998950 1053631, 9990~
#> # ... with 18 more rows
```

5 Advanced Features

`spmodel` offers several advanced features for fitting spatial linear models. We briefly discuss each next using the `moss` data and simulated data, though technical details for each advanced feature can be seen by running `vignette("technical", "spmodel")`.

5.1 Fixing Spatial Covariance Parameters

We may desire to fix specific spatial covariance parameters at a particular value. Perhaps some parameter value is known, for example. Or perhaps we want to compare nested models where a reduced model uses a fixed parameter value while the full model estimates the parameter. Fixing spatial covariance parameters while fitting a model is possible using the `spcov_initial` argument to `spmod()` and `spautor()`. The `spcov_initial` argument takes an `spcov_initial` object (run `help("spcov_initial", "spmodel")` for more). `spcov_initial` objects can also be used to specify initial values used during optimization, even if they are not assumed to be fixed. By default, `spmodel` uses a grid search to find suitable initial values to use during optimization.

Suppose the goal is to compare the full model to a model that assumes the independent random error variance (nugget) is zero. First, the `spcov_initial` object is specified:

```
init <- spcov_initial("exponential", ie = 0, known = "ie")
print(init)
```

```
#> $initial
#> ie
#>  0
#>
#> $is_known
#>   ie
#> TRUE
#>
#> attr(,"class")
#> [1] "exponential"
```

The `init` output shows that the `ie` parameter has an initial value of zero that is assumed to be known. Next the model is fit:

```
spmod_red <- splm(log_Zn ~ log_dist2road, moss, spcov_initial = init)
```

Notice that because the `spcov_initial` object contains information about the spatial covariance type, the `spcov_type` argument is not required when `spcov_initial` is provided. We can use `glances()` to glance at both models:

```
glances(spmod, spmod_red)
```

```
#> # A tibble: 2 x 10
#>   model         n     p  npar value   AIC  AICc logLik deviance pseudo.r.squared
#>   <chr>     <int> <dbl> <int> <dbl> <dbl> <dbl>  <dbl>    <dbl>            <dbl>
#> 1 spmod       365     2     3  367.  373.  373.  -184.     363             0.683
#> 2 spmod_red   365     2     2  378.  382.  382.  -189.     374.            0.703
```

The lower AIC and AICc of the full model compared to the reduced model indicates that the independent random error variance is important to the model. A likelihood ratio test comparing the full and reduced models is also possible using `anova()`.

5.2 Random Effects

Random effects incorporate additional sources of variability into model fitting. They are accommodated in `spmodel` using similar syntax as for random effects in the nlme (Pinheiro and Bates 2006) and lme4 (Bates et al. 2015) **R** packages. Random effects are specified via a formula passed to the `random` argument. Next we show two examples incorporating random effects into the spatial linear model using the `moss` data.

The first example explores random intercepts for the `sample` variable. The `sample` variable indexes each unique location, which can have replicate observations due to field duplicates (`field_dup`) and lab replicates (`lab_rep`). We create extra correlation for repeated observations from a sample by creating a random intercept for each level of `sample`. We incorporate the random intercepts for `sample` by running

```
rand1 <- splm(
  log_Zn ~ log_dist2road,
  moss,
  spcov_type = "exponential",
  random = ~ sample
)
```

Note that `sample` is shorthand for `(1 | sample)`, which is more explicit notation that indicates random intercepts for each level of `sample`.

The second example adds a random intercept for `year`, which creates extra correlation for observations within a year. It also adds a random slope for `log_dist2road` within `year`, which lets the effect of `log_dist2road` vary between years. We fit this model by running

```
rand2 <- splm(
  log_Zn ~ log_dist2road,
  moss,
  spcov_type = "exponential",
  random = ~ sample + (log_dist2road | year)
)
```

Note that `sample + (log_dist2road | year)` is shorthand for `(1 | sample) + (log_dist2road | year)`. If only random slopes within year are desired (and no random intercepts), a `- 1` is given to the relevant portion of the formula: `(log_dist2road - 1 | year)`. When there is more than one term in `random`, each term must be surrounded by parentheses (recall that the random intercept shorthand automatically includes relevant parentheses). More examples of random effect syntax are provided in Appendix B.

We glance at all three models by running

```
glances(spmod, rand1, rand2)
```

```
#> # A tibble: 3 x 10
#>   model     n     p  npar value   AIC  AICc logLik deviance pseudo.r.squared
#>   <chr> <int> <dbl> <int> <dbl> <dbl> <dbl>  <dbl>    <dbl>            <dbl>
#> 1 rand2   365     2     6  190.  202.  202.  -94.9     363.            0.215
#> 2 rand1   365     2     4  335.  343.  343. -168.      363             0.661
#> 3 spmod   365     2     3  367.  373.  373. -184.      363             0.683
```

`rand2` has the lowest AIC and AICc.

It is possible to fix random effect variances using the `randcov_initial` argument. `randcov_initial` can also be used to set initial values for optimization.

5.3 Partition Factors

A partition factor is a variable that assumes observations are uncorrelated when they are from different levels of the partition factor. Partition factors are specified in `spmodel` by providing a formula with a single variable to the `partition_factor` argument. Suppose that for the `moss` data, it is appropriate to assume observations in different years (`year`) are uncorrelated. We fit a model that treats year as a partition factor by running

```
part <- splm(
  log_Zn ~ log_dist2road,
  moss,
  spcov_type = "exponential",
  partition_factor = ~ year
)
```

5.4 Anisotropy

A spatial covariance function for point-referenced data is isotropic if it behaves similarly in all directions (i.e., is independent of direction) as a function of distance. An anisotropic covariance function does not behave similarly in all directions as a function of distance. Consider the spatial covariance imposed by an eastward-moving wind pattern. A one-unit distance in the x-direction likely means something different than a one-unit distance in the y-direction. Figure 5 shows ellipses for an isotropic and anisotropic covariance function centered at the origin (a distance of zero).

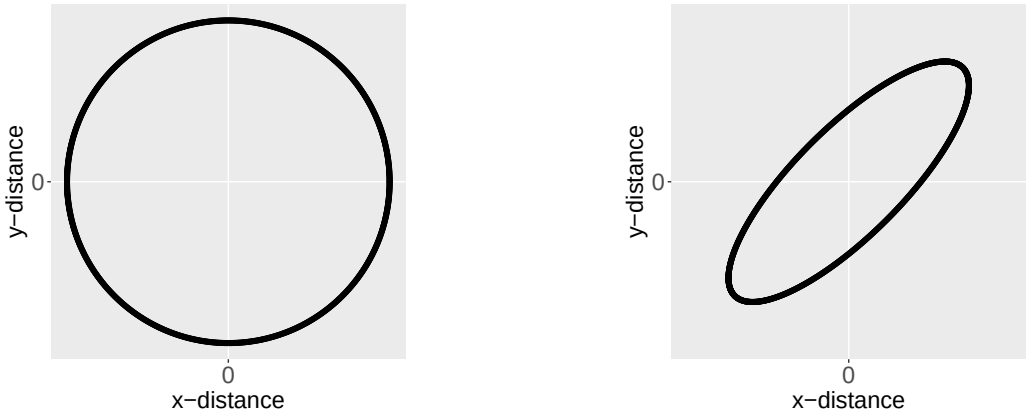

Figure 5: Ellipses for an isotropic (left) and anisotropic (right) covariance function centered at the origin. The black outline of each ellipse is a level curve of equal correlation.

The black outline of each ellipse is a level curve of equal correlation. The left ellipse (a circle) represents an isotropic covariance function. The distance at which the correlation between two observations lays on the level curve is the same in all directions. The right ellipse represents an anisotropic covariance function. The distance at which the correlation between two observations lays on the level curve is different in different directions.

Accounting for anisotropy involves a rotation and scaling of the x-coordinates and y-coordinates such that the spatial covariance function that uses these transformed distances is isotropic. We fit a model with anisotropy by running

```
spmod_anis <- splm(
  log_Zn ~ log_dist2road,
  moss,
  spcov_type = "exponential",
  anisotropy = TRUE
```

```
)
summary(spmod_anis)
```

```
#>
#> Call:
#> splm(formula = log_Zn ~ log_dist2road, data = moss, spcov_type = "exponential",
#>     anisotropy = TRUE)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -2.5279 -1.2239 -0.7202 -0.1921  1.1659
#>
#> Coefficients (fixed):
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    9.54798    0.22291   42.83   <2e-16 ***
#> log_dist2road -0.54601    0.01855  -29.44   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Pseudo R-squared: 0.7048
#>
#> Coefficients ( spatial covariance):
#>        de        ie     range    rotate     scale
#> 3.561e-01 6.812e-02 8.732e+03 2.435e+00 4.753e-01
#> attr(,"class")
#> [1] "exponential"
```

The `rotate` parameter is between zero and π radians and represents the angle of a clockwise rotation of the ellipse. The `scale` parameter is between zero and one and represents the ratio of the distance between the origin and the edge of the ellipse along the minor axis to the distance between the origin and the edge of the ellipse along the major axis. Figure 6 shows the transformation that turns an anisotropic ellipse into an isotropic one (i.e., a circle). This transformation requires rotating the the coordinates clockwise by `rotate` and then scaling them the reciprocal of `scale`. The transformed coordinates are then used instead of the original coordinates to compute distances and spatial covariances.

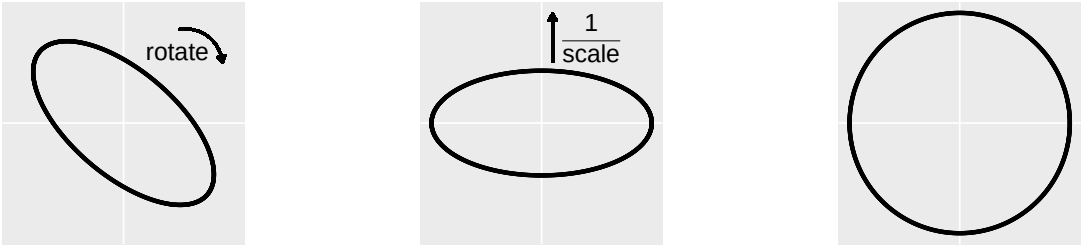

Figure 6: A visual representation of the anisotropy transformation. In the left figure, the first step is to rotate the anisotropic ellipse clockwise by the `rotate` parameter. In the middle figure, the second step is to scale the minor axis by the reciprocal of the `scale` parameter. In the right figure, the anisotropic ellipse has been transformed into an isotropic one (i.e., a circle). The transformed coordinates are then used instead of the original coordinates to compute distances and spatial covariances.

Note that specifying an initial value for `rotate` that is different from zero, specifying an initial value for `scale` that is different from one, or assuming either `rotate` or `scale` are unknown in `spcov_initial` will cause `splm()` to fit a model with anisotropy (and will override `anisotropy = FALSE`). Estimating anisotropy

parameters is only possible for maximum likelihood and restricted maximum likelihood estimation, but fixed anisotropy parameters can be accommodated for semivariogram weighted least squares or semivariogram composite likelihood estimation. Also note that anisotropy is not relevant for areal data because the spatial covariance function depends on a neighborhood structure instead of distances between points.

5.5 Simulating Spatial Data

The `sprnorm()` function is used to simulate normal (Gaussian) spatial data. To use `sprnorm()`, the `spcov_params()` function is used to create an `spcov_params` object. The `spcov_params()` function requires the spatial covariance type and and parameter values. We create an `spcov_params` object by running

```
sim_params <- spcov_params("exponential", de = 5, ie = 1, range = 0.5)
```

We set a reproducible seed and then simulate data at 3000 random locations in the unit square using the spatial covariance parameters in `sim_params` by running

```
set.seed(0)
n <- 3000
x <- runif(n)
y <- runif(n)
sim_coords <- tibble::tibble(x, y)
sim_resp <- sprnorm(sim_params, data = sim_coords, xcoord = x, ycoord = y)
sim_data <- tibble::tibble(sim_coords, sim_resp)
```

We can visualize the simulated data (Figure 7, left) by running

```
ggplot(sim_data, aes(x = x, y = y, color = sim_resp)) +
  geom_point(size = 1.5) +
  scale_color_viridis_c(limits = c(-7, 7)) +
  theme_gray(base_size = 18)
```

There is noticeable spatial patterning in the response variable (`sim_resp`). The default mean in `sprnorm()` is zero for all observations, though a mean vector can be provided using the `mean` argument. The default number of samples generated in `sprnorm()` is one, though this is changed using the `samples` argument. Because `dat` is a `tibble` (`data.frame`) and not an `sf` object, the columns in `dat` representing the x-coordinates and y-coordinates must be provided to `sprnorm()`.

Note that the output from `coef(object, type = "spcov")` is a `spcov_params` object. This is useful if one wants to simulate data given the estimated spatial covariance parameters from a fitted model. Random effects are incorporated into simulation via the `randcov_params` argument.

5.6 Big Data

The computational cost associated with model fitting is exponential in the sample size for all estimation methods. For maximum likelihood and restricted maximum likelihood, the computational cost of estimating $\boldsymbol{\theta}$ is cubic. For semivariogram weighted least squares and semivariogram composite likelihood, the computational cost of estimating $\boldsymbol{\theta}$ is quadratic. The computational cost associated with estimating $\boldsymbol{\beta}$ and prediction is cubic in the model-fitting sample size, regardless of estimation method. Typically samples sizes approaching 10,000 make the computational cost of model fitting and prediction infeasible, which necessitates the use of big data methods. `spmodel` offers big data methods for model fitting of point-referenced data via the `local` argument to `splm()`, capable of fitting models with hundreds of thousands to millions of observations rather quickly. Because of the neighborhood structure of areal data, the big data methods used for point-referenced data do not apply to areal data. Thus there is no big data method for areal data or `local` argument to `spautor()` and model fitting sample sizes cannot be too large.

`spmodel` offers big data methods for prediction of point-referenced data or areal data via the `local` argument to `predict()`, capable of making predictions at hundreds of thousands to millions of locations rather quickly.

To show how to use `spmodel` for big data estimation and prediction, we use the `sim_data` data from Section 5.5. Because `sim_data` is a `tibble` (`data.frame`) and not an `sf` object, the column in `data` representing the x-coordinates and y-coordinates must be explicitly provided to `splm()`. Though the sample sizes used next for model fitting and prediction are relatively small, the purpose of the following examples is simply to illustrate the big data approaches.

5.6.1 Model-fitting

`spmodel` uses a "local indexing" approximation for big data model fitting of point-referenced data. Observations are first assigned an index. Then for the purposes of model fitting, observations with different indexes are assumed uncorrelated. This approach has some connections to composite likelihood and to the partition factors discussed earlier.

The `local` argument to `splm()` controls the big data options. `local` is a list with several arguments. The arguments to the `local` list control the method used to assign the indexes, the number of observations with the same index, the number of unique indexes, variance adjustments to the covariance matrix of $\hat{\boldsymbol{\beta}}$, whether or not to use parallel processing, and if parallel processing is used, the number of cores.

The simplest way to accommodate big data is to set `local` to `TRUE`. This is shorthand for `local = list(method = "random", size = 50, var_adjust = "theoretical", parallel = FALSE)`, which creates groups with approximately 50 observations each using a random assignment of groups to indexes, uses the theoretically-correct variance adjustment, and does not use parallel processing.

```
local1 <- splm(sim_resp ~ 1, sim_data, spcov_type = "exponential",
               xcoord = x, ycoord = y, local = TRUE)
summary(local1)
```

```
#>
#> Call:
#> splm(formula = sim_resp ~ 1, data = sim_data, spcov_type = "exponential",
#>     xcoord = x, ycoord = y, local = TRUE)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -5.0356 -1.3514 -0.1468  1.2842  6.5381
#>
#> Coefficients (fixed):
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   -1.021      0.699   -1.46    0.144
#>
#> Coefficients ( spatial covariance):
#>     de     ie  range
#> 2.8724 0.9735 0.2644
```

Instead of using `local = TRUE`, we can explicitly set `local`. For example, we fit a model using using k-means clustering (MacQueen and others 1967) on the x-coordinates and y-coordinates to create 60 groups (clusters), the pooled variance adjustment, and parallel processing with two cores by running

```
local2_list <- list(method = "kmeans", groups = 60, var_adjust = "pooled",
                    parallel = TRUE, ncores = 2)
local2 <- splm(sim_resp ~ 1, sim_data, spcov_type = "exponential",
               xcoord = x, ycoord = y, local = local2_list)
summary(local2)
```

```
#>
#> Call:
#> splm(formula = sim_resp ~ 1, data = sim_data, spcov_type = "exponential",
```

```
#>     xcoord = x, ycoord = y, local = local2_list)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -4.98801 -1.30386 -0.09927  1.33176  6.58567
#>
#> Coefficients (fixed):
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  -1.0683     0.1759  -6.073 1.25e-09 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Coefficients ( spatial covariance):
#>     de     ie  range
#> 2.5434 0.9907 0.2312
```

We can use `loocv()` to evaluate model fit. Note that `loocv()` has big data options that are passed to `predict()`, which we discuss in Section 5.6.2.

```
loocv(local1, local = list(method = "distance", parallel = TRUE, ncores = 2))
```

```
#> [1] 1.243062
```

```
loocv(local2, local = list(method = "distance", parallel = TRUE, ncores = 2))
```

```
#> [1] 1.242925
```

Likelihood-based statistics like `AIC()`, `AICc()`, `logLik()`, and `deviance()` should not be used to compare a model fit with a big data approximation to a model fit without a big data approximation, as the two approaches maximize different likelihoods.

5.6.2 Prediction

For point-referenced data, `spmodel` uses a "local neighborhood" approximation for big data prediction. Each prediction is computed using a subset of the observed data instead of all the observed data. Before further discussing big data prediction, we simulate 1000 locations in the unit square requiring prediction:

```
n_pred <- 1000
x <- runif(n_pred)
y <- runif(n_pred)
sim_preds <- tibble::tibble(x = x, y = y)
```

The `local` argument to `predict()` controls the big data options. `local` is a list with several arguments. The arguments to the `local` list control the method used to subset the observed data, the number of observations in each subset, whether or not to use parallel processing, and if parallel processing is used, the number of cores.

The simplest way to accommodate big data prediction is to set `local` to `TRUE`. This is shorthand for `local = list(method = "covariance", size = 50, parallel = FALSE)`, which implies that uniquely for each location requiring prediction, only the 50 observations in the data most correlated with it are used in the computation and parallel processing is not used. Using the `local1` fitted model, we store these predictions as a variable called `preds` in the `sim_preds` data by running

```
sim_preds$preds <- predict(local1, newdata = sim_preds, local = TRUE)
```

The predictions are visualized (Figure 7, right) by running

```
ggplot(sim_preds, aes(x = x, y = y, color = preds)) +
  geom_point(size = 1.5) +
```

```
  scale_color_viridis_c(limits = c(-7, 7)) +
  theme_gray(base_size = 18)
```

They display a similar pattern as the observed data.

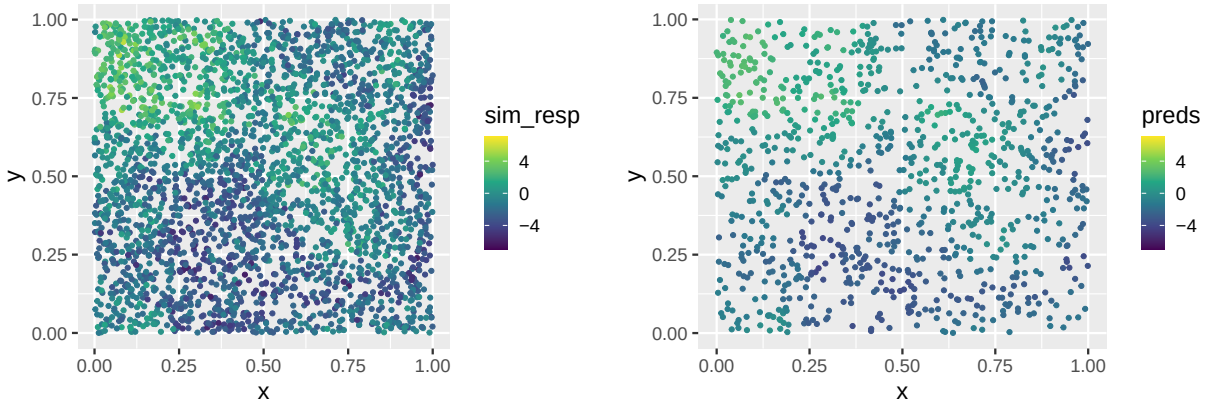

Figure 7: Observed data and big data predictions at unobserved locations. In the left figure, spatial data are simulated in the unit square. A spatial linear model is fit using the default big data approximation for model-fitting. In the right figure, predictions are made using the fitted model and the default big data approximation for prediction.

Instead of using `local = TRUE`, we can explicitly set `local`. For example, we make predictions using the distance method, subsets of 30 observations, and parallel processing with 2 cores by running

```
pred_list <- list(method = "distance", size = 30, parallel = TRUE, ncores = 2)
predict(local1, newdata = sim_preds, local = pred_list)
```

For areal data, no local neighborhood approximation exists because of the data's underlying neighborhood structure. Thus all of the data must be used to compute predictions and by consequence, `method` and `distance` are not components of the `local` list. The only components of the `local` list for areal data are `parallel` and `ncores`:

```
predict(sealmod, local = list(parallel = TRUE, ncores = 2))
```

6 Discussion

Throughout this vignette, we have shown how to use `spmodel` to fit, summarize, and predict for a variety of spatial statistical models. Spatial linear models for point-referenced data (i.e., geostatistical models) are fit using the `splm()` function. Spatial linear models for areal data (i.e., spatial autoregressive models) are fit using the `spautor()` function. Several model-fit statistics and diagnostics are available. The broom functions `tidy()` and `glance()` are used to tidy and glance at a fitted model. The broom function `augment()` is used to augment `data` with model diagnostics and augment `newdata` with predictions. Several advanced features are available to accommodate fixed covariance parameter values, random effects, partition factors, anisotropy, simulated data, and big data approximations for model fitting and prediction.

We appreciate feedback from users regarding `spmodel`. To learn more about how to provide feedback or contribute to `spmodel`, please visit our GitHub repository at https://github.com/USEPA/spmodel.

References

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software* 67 (1): 1–48. https://doi.org/10.18637/jss.v067.i01.

Cook, R Dennis, and Sanford Weisberg. 1982. *Residuals and Influence in Regression*. New York: Chapman; Hall.

Cressie, Noel. 1985. "Fitting Variogram Models by Weighted Least Squares." *Journal of the International Association for Mathematical Geology* 17 (5): 563–86.

Curriero, Frank C, and Subhash Lele. 1999. "A Composite Likelihood Approach to Semivariogram Estimation." *Journal of Agricultural, Biological, and Environmental Statistics*, 9–28.

Harville, David A. 1977. "Maximum Likelihood Approaches to Variance Component Estimation and to Related Problems." *Journal of the American Statistical Association* 72 (358): 320–38.

Hoeting, Jennifer A, Richard A Davis, Andrew A Merton, and Sandra E Thompson. 2006. "Model Selection for Geostatistical Models." *Ecological Applications* 16 (1): 87–98.

MacQueen, James, and others. 1967. "Some Methods for Classification and Analysis of Multivariate Observations." In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1:281–97. 14. Oakland, CA, USA.

Montgomery, Douglas C, Elizabeth A Peck, and G Geoffrey Vining. 2021. *Introduction to Linear Regression Analysis*. John Wiley & Sons.

Patterson, H Desmond, and Robin Thompson. 1971. "Recovery of Inter-Block Information When Block Sizes Are Unequal." *Biometrika* 58 (3): 545–54.

Pebesma, Edzer. 2018. "Simple Features for R: Standardized Support for Spatial Vector Data." *The R Journal* 10 (1): 439–46. https://doi.org/10.32614/RJ-2018-009.

Pinheiro, José, and Douglas Bates. 2006. *Mixed-Effects Models in S and S-Plus*. Springer science & business media.

Robinson, David, Alex Hayes, and Simon Couch. 2021. *Broom: Convert Statistical Objects into Tidy Tibbles*. https://CRAN.R-project.org/package=broom.

Tobler, Waldo R. 1970. "A Computer Movie Simulating Urban Growth in the Detroit Region." *Economic Geography* 46 (sup1): 234–40.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wolfinger, Russ, Randy Tobias, and John Sall. 1994. "Computing Gaussian Likelihoods and Their Derivatives for General Linear Mixed Models." *SIAM Journal on Scientific Computing* 15 (6): 1294–1310.

Appendices

A An Additional Example Using `caribou`

The `caribou` data are observed on an equally spaced lattice and can be analyzed as point-referenced or areal data. We view the first few rows of `caribou` by running

```
caribou
```

```
#> # A tibble: 30 x 5
#>    water tarp      z     x     y
#>    <fct> <fct> <dbl> <dbl> <dbl>
#>  1 Y     clear  2.42     1     6
#>  2 Y     shade  2.44     2     6
#>  3 Y     none   1.81     3     6
#>  4 N     clear  1.97     4     6
#>  5 N     shade  2.38     5     6
#>  6 Y     none   2.22     1     5
#>  7 N     clear  2.10     2     5
#>  8 Y     clear  1.80     3     5
#>  9 Y     shade  1.96     4     5
#> 10 Y     none   2.10     5     5
#> # ... with 20 more rows
```

First we analyze `caribou` as point-referenced data. Because `caribou` is not an `sf` object, we must provide the columns in `caribou` that represent the x-coordinates and y-coordinates. We fit a spatial linear model regressing nitrogen percentage (`z`) on water presence (`water`) and tarp cover (`tarp`) by running

```
cariboumod <- splm(z ~ water + tarp, data = caribou,
                   spcov_type = "exponential", xcoord = x, ycoord = y)
```

An analysis of variance can be conducted to assess the overall impact of the `tarp` variable, which has three levels (clear, shade, and none), and the `water` variable, which has two levels (water and no water). We perform an analysis of variance and tidy the results by running

```
tidy(anova(cariboumod))
```

```
#> # A tibble: 3 x 4
#>   effects        df statistic  p.value
#>   <chr>       <int>     <dbl>    <dbl>
#> 1 (Intercept)     1     43.5  4.33e-11
#> 2 water           1      1.66 1.98e- 1
#> 3 tarp            2     15.4  4.51e- 4
```

There is significant evidence that at least one tarp cover impacts nitrogen. Note that, like in `summary()`, these p-values are associated with an asymptotic hypothesis test (here, an asymptotic Chi-squared test).

Next we analyze `caribou` as areal data. Because `caribou` is not an `sf` object, we must create a weights matrix. We define two observations as neighbors if they are adjacent (directly east, west, north, or south) to one another. Two observations in `caribou` are adjacent if the distance between them equals one (recall that observations are not neighbors with themselves):

```
coords <- cbind(caribou$x, caribou$y)
dists <- as.matrix(dist(coords))
W <- dists == 1
```

Currently, `W` is a logical matrix with `TRUE`s and `FALSE`s. We coerce it to a numeric matrix by running

```
W <- W * 1
```

The ijth value in `W` is `1` if the observation in the ith row is neighbors with the observation in the jth row and `0` otherwise. We fit a spatial autoregressive model regressing the nitrogen percentage (`z`) on water presence (`water`) and tarp cover (`tarp`) by running

```
cariboumod <- spautor(z ~ water + tarp, data = caribou,
                      spcov_type = "car", W = W)
```

We perform an analysis of variance and tidy the results by running

```
tidy(anova(cariboumod))
```

```
#> # A tibble: 3 x 4
#>   effects        df statistic   p.value
#>   <chr>       <int>     <dbl>     <dbl>
#> 1 (Intercept)     1    714.   2.36e-157
#> 2 water           1      1.82 1.77e-  1
#> 3 tarp            2     15.1  5.17e-  4
```

There is significant evidence that at least one tarp cover impacts nitrogen. Note that, like in `summary()`, these p-values are associated with an asymptotic hypothesis test (here, an asymptotic Chi-squared test).

B Random Effect Syntax

A couple of common ways to specify random effects in the `random` argument to `splm()` or `spautor()` include:

- `~ (1 | group)` : Random intercepts for each level of `group`. `~ group` is shorthand for `~ (1 | group)`.
- `~ (var | group)`: Random intercepts for each level of `group` and random slopes that depend on the variable `var` for each level of `group`.

Some additional syntax for more complicated random effects structures include:

- `~ (var - 1 | group)`: Random slopes (without intercepts) that depend on the variable `var` for each level of `group`.
- `~ (1 | group:subgroup)`: Random intercepts for each combination of levels in `group` and levels in `subgroup`. `~ group:subgroup` is shorthand for `~ (1 | group:subgroup)`.
- `~ (var | group:subgroup)`: Random intercepts for each combination of levels in `group` and levels in `subgroup` and random slopes that depend on the variable `var` for each combination of levels in `group` and levels in `subgroup`.
- `~ (var - 1 | group:subgroup)`: Random slopes (without intercepts) that depend on the variable `var` for each combination of levels in `group` and levels in `subgroup`.
- `~ (1 | group/subgroup)`: Shorthand for `~ (1 | group) + (1 | group:subgroup)`. Commonly, the `group/subgroup` notation implies `subgroup` is nested within `group`.
- `~ (var | group/subgroup)`: Shorthand for `~ (var | group) + (var | group:subgroup)`. Commonly, the `group/subgroup` notation implies `subgroup` is nested within `group`.
- `~ (var - 1 | group/subgroup)`: Shorthand for `~ (var - 1 | group) + (var - 1 | group:subgroup)`. Commonly, the `group/subgroup` notation implies `subgroup` is nested within `group`.

Distinct random effects terms are separated in `random` by `+`. Each term must be wrapped in parentheses. For example, to incorporate random intercepts for `group` and `subgroup`, `random` looks like `~ (1 | group) + (1 | subgroup)`. For random intercepts, recall that `~ group` is shorthand for `~ (1 | group)`. Thus, an equivalent representation of `~ (1 | group) + (1 | subgroup)` is `~ group + subgroup`. Note that for both random intercepts and random slopes, the variable on the right-hand side of `|` (i.e., `group`, `subgroup`, `group:subgroup`) must be a factor (or character) variable.