Supplementary Material for "Point process models for presence-only analysis – a review"

March 1, 2015

1 Description of envrionmental covarites

Point process models fitted in the main text and in the Supplementary Material utilise five environmental covariates and two observer bias covariates. The environmental covariates include the number of fires since 1943, minimum and maximum annual temperature, annual rainfall, and a categorical covariate describing the type of soil. The values of the soil covariate are presented in Table 1.

Table 1: Description of soil categories used for model fitting.

Soil category	Description
2	siliceous (white) sandplains
5	low quartz sedimentary, ultramafic igneous and metamorphics, aeolian (red) sandplains, or limestone
6	felsic intrusives
7	high quartz sedimentary
8	mafic volcanics and intrusives
11	floodplain alluvium, lacustrine sediments, or estuarine sediments
12	residual alluvial sands or residual alluvial/colluvial sand and gravel
15	felsic volcanics

2 Point process model likelihood

2.1 Homogeneous Poisson point process

The simplest example of a PPM is a homogeneous Poisson point process, which is of little use in SDM but serves as a good conceptual starting point. A homogeneous Poisson point process is characterised by two conditions: first, all points are distributed independently and uniformly over $\mathcal{A}$, and second, the number of presences m is the observed realisation of a Poisson random variable M with mean $\lambda|\mathcal{A}|$ (Cressie, 1993; Diggle, 2003). As a consequence of these conditions, the intensity λ is constant throughout the region $\mathcal{A}$. More specifically, the first condition implies that given a point, the density of its location s is:

$$f(s) = \frac{1}{|\mathcal{A}|}, s \in \mathcal{A}.$$

Hence, conditional on $M = m$, the joint density of the m points $s_1, \ldots, s_m$ is:

$$f(s_1, \ldots, s_m | M = m) = \frac{1}{|\mathcal{A}|^m}. \tag{1}$$

The second condition describes the probability structure for the number of points m:

$$P(M = m) = \frac{e^{-\lambda|\mathcal{A}|}(\lambda|\mathcal{A}|)^m}{m!}, m = 0, 1, 2, \ldots. \tag{2}$$

The form of the likelihood equation can be derived from (1) and (2):

$$\begin{aligned} L(\boldsymbol{\beta}; \mathbf{s}_P) &= m! f(s_1, \ldots, s_m | M = m) P(M = m) \\ &= m! \frac{1}{|\mathcal{A}|^m} \frac{e^{-\lambda|\mathcal{A}|}(\lambda|\mathcal{A}|)^m}{m!} \\ &= \lambda^m e^{-\lambda|\mathcal{A}|}. \end{aligned} \tag{3}$$

The log-likelihood is found by taking the logarithm of (3):

$$l(\boldsymbol{\beta}; \mathbf{s}_P) = m \ln \lambda - \lambda|\mathcal{A}|.$$

Parameters of Poisson point process models are typically estimated by maximum likelihood (Cressie, 1993). In the case of a homogeneous PPM, the estimator of λ that maximises the likelihood is $|\mathcal{A}|/m$, which also happens to be the method-of-moments estimator. (The equivalence of maximum likelihood and method-of-moments estimators actually applies to all types of Poisson process.)

2.2 Inhomogeneous Poisson point process

The assumptions of an inhomogeneous Poisson point process are that (a) point events are independent of each other, which can be shown to imply that the total number of points in the study region is a Poisson random variable and (b) that the intensity $\lambda(s)$ varies spatially according to environmental conditions $\mathbf{x}(s)$ and is therefore indexed by location s.

The conditional independence of the points implies that given a point, the density of its location s is:

$$f(s) = \frac{\lambda(s)}{\int_{\mathcal{A}} \lambda(s)ds}, s \in \mathcal{A}.$$

Hence, conditional on the number of points $M = m$, the joint density of the m points $s_1, \ldots, s_m$ is:

$$f(s_1, \ldots, s_m | M = m) = \frac{\prod_{i=1}^{m} \lambda(s_i)}{(\int_{\mathcal{A}} \lambda(s)ds)^m}. \tag{4}$$

Because the number of points m is an observed realisation of a Poisson random variable M with mean $\int_{\mathcal{A}} \lambda(s)ds$, we describe the probability structure for the number of points m as follows:

$$P(M = m) = \frac{e^{-\int_{\mathcal{A}} \lambda(s)ds} (\int_{\mathcal{A}} \lambda(s)ds)^m}{m!}, \; m = 0, 1, 2, \ldots. \tag{5}$$

The form of the likelihood equation can be derived from (4) and (5):

$$\begin{aligned} L(\boldsymbol{\beta}; \mathbf{s}_P) &= m! f(s_1, \ldots, s_m | M = m) P(M = m) && (6) \\ &= m! \frac{\prod_{i=1}^{m} \lambda(s_i)}{(\int_{\mathcal{A}} \lambda(s)ds)^m} \frac{e^{-\int_{\mathcal{A}} \lambda(s)ds} (\int_{\mathcal{A}} \lambda(s)ds)^m}{m!} \\ &= e^{-\int_{\mathcal{A}} \lambda(s)ds} \prod_{i=1}^{m} \lambda(s_i). && (7) \end{aligned}$$

The $m!$ factor in (6) is included because we consider the m points of $\mathbf{s}_P$ to be unordered and thus there are $m!$ arrangements of the m points. An expression of the form of (6) is sometimes called a Janossy density (Daley & Vere-Jones, 1988). The log-likelihood is found by taking the logarithm of (7):

$$l(\boldsymbol{\beta}; \mathbf{s}_P) = \sum_{i=1}^{m} \ln \lambda(s_i) - \int_{\mathcal{A}} \lambda(s)ds.$$

3 spatstat

Perhaps the most well-known software for fitting PPMs is `spatstat` (Baddeley & Turner, 2005), an `R` package that offers a large suite of tools for analysing spatial point patterns

including plotting, model fitting, diagnostic checks, and simulation. In this Section we provide code for each step required in fitting PPMs and checking results with `spatstat`.

3.1 Setting up the study window

```
>library(spatstat)
>load("Eucalyptus sparsifolia Atlas 2012.RData") #Contains X and Y
>load("Quad1000.RData") #Contains quad
>ux = sort(unique(quad$X))
>uy = sort(unique(quad$Y))
>nx = length(ux)
>ny = length(uy)
>col.ref = match(quad$X, ux)
>row.ref = match(quad$Y, uy)
>all.vec = rep(NA, max(row.ref)*max(col.ref))
>vec.ref = (col.ref - 1)*max(row.ref) + row.ref
>all.vec[vec.ref] = 1
>Sydney.mask = matrix(all.vec, max(row.ref), max(col.ref),
 dimnames = list(uy, ux))
>Sydney.win = as.owin(im(Sydney.mask, xcol = ux, yrow = uy))
```

3.2 Make point pattern and quadrature scheme

By default, quadrature points are randomly generated in `spatstat`, although user-entered quadrature points can be used instead as below, where they are contained in the data frame `quad`.

```
>ppp.dat = ppp(X, Y, window = Sydney.win, check = FALSE)
>quads = ppp(quad$X, quad$Y, window = Sydney.win)
>Q = quadscheme(data = ppp.dat, dummy = quads, method = "grid",
 ntile = c(nx, ny), npix = c(nx, ny))
```

3.3 Setting up covariate lists

We set up a list of covariates `int.list` for fitting the model `ft.int` and a corresponding list `pred.list` with distance-based covarates set to zero for prediction.

```
>X.des = cbind(poly(quad$FC, quad$MNT, quad$MXT, quad$Rain, degree = 2,
 raw = TRUE), poly(sqrt(quad$D.Main), sqrt(quad$D.Urb), degree = 2,
 raw = TRUE), quad$soil)
>int.list = list()
>for (i in 1:dim(X.des)[2])
>{
>all.vec = rep(NA, max(row.ref)*max(col.ref))
>vec.ref = (col.ref - 1)*max(row.ref) + row.ref
>all.vec[vec.ref] = X.des[,i]
>int.list[[i]] = im(matrix(all.vec, max(row.ref), max(col.ref),
 dimnames = list(uy, ux)), xcol = ux, yrow = uy)
>}
>names(int.list) = paste("V", 1:dim(X.des)[2], sep = "")
>pred.list = int.list
>set.0 = 15:19 #Variables to set to 0
>for (v in set.0)
>{
>pred.list[[v]]$v = 0*pred.list[[v]]$v
>}
```

3.4 Fit Poisson point process model

```
>int.form = as.formula(paste("~", paste(names(int.list), collapse = "+")))
>ft.int = ppm(Q, trend = as.formula(int.form), covariates = int.list)
```

The fitted coefficients of `ft.int` are presented in Table 3.

3.5 Inference for coefficients and models

We can make inferences about coefficients in a `ppm`, or compare several nested `ppm` objects, using `summary` and `anova` tools as they are used for standard regression objects. For

example, the interaction coefficient between fire count and annual rainfall is small relative to its standard error, and an ANOVA comparing a model with and without this interaction term (`ft.int` and `ft.no.fc.Rain`) suggests it does not explain significant variation in intensity:

```
>anova(ft.no.fc.Rain, ft.int)
```

Note however that these inferential tools rely on the assumption of independence of presence points, and they are sensitive to violations of this assumption.

The `anova` function may also be used as a way to assess variable importance, by comparing the full model with models fitted without each of the covariates. Table 2 contains the differences in deviance between the full model and models fitted without each of the covariates. It suggests that the temperature variables and soil type are the most important environmental covariates in determining the distribution of *Eucalyptus sparsifolia*.

Table 2: Difference in deviance between the model with all covariates `ft.int` and models without each of the covariates, as produced by the `anova` function in `spatstat`.

Covariate	Difference in Deviance
Fire count since 1943	31.038
Minimum annual temperature	122.690
Maximum annual temperature	81.444
Annual rainfall	58.818
Soil type	85.490
Distance from the nearest main road	49.273
Distance from the nearest urban area	80.533

3.6 Fit area-interaction model

The `profilepl` function can be used to determine the appropriate interaction radius for fitting an area-interaction model.

```
>rs = data.frame(seq(2, 10, 0.2))
>names(rs) = "r"
>plike = profilepl(rs, AreaInter, Q, trend = int.form, covariates = int.list)
```

We now fit an area-interaction model with the suggested radius of 2km:

```
>ft.ai.2 = ppm(Q, trend = as.formula(int.form), covariates = int.list,
 interaction = AreaInter(2))
```

3.7 Model Diagnostics

There are a number of diagnostic tools available in `spatstat` to identify departures of model assumptions. The generic `envelope` function generates Monte Carlo simulation envelopes for some summary function of a fitted model. In the main text, we fitted a simulation envelope for the inhomogeneous K-function in Figure 3a as follows:

```
>envelope(ft.int, fun = Kinhom, nrank = 26, nsim = 1001)
```

Residual plots may be produced with a call to the `diagnose.ppm` function. Figure 3b of the main text is a plot of smoothed Pearson residuals for the model `ft.ai.2`, produced as follows:

```
>diagnose.ppm(ft.ai.2, which = "smooth", type = "Pearson", labcex = 1)
```

Figure 3b exhibits some pattern, with areas along the mid-coast and northwest showing the highest positive residuals, and the south showing the most negative residuals. Cumulative Pearson residuals for both increasing longitude (x) and latitude (y) do not show any significant departures from Monte Carlo simulation envelopes in Figure 3c-d of the main text, so the model fit may be assumed to be reasonable. The code to produce the plots in Figure 3c-d is as follows:

```
>diagnose.ppm(ft.ai.2, which = "x", type = "Pearson", compute.sd = TRUE)
>diagnose.ppm(ft.ai.2, which = "y", type = "Pearson", compute.sd = TRUE)
```

3.8 Maps of predicted intensity

```
>pred.ai.2 = predict(ft.ai.2, covariates = pred.list, ngrid = c(ny, nx))
>pred.int = predict(ft.int, covariates = pred.list, ngrid = c(ny, nx))
>plot(pred.int) #Fig. 1 (left)
>plot(pred.ai.2) #Fig. 1 (right)
```

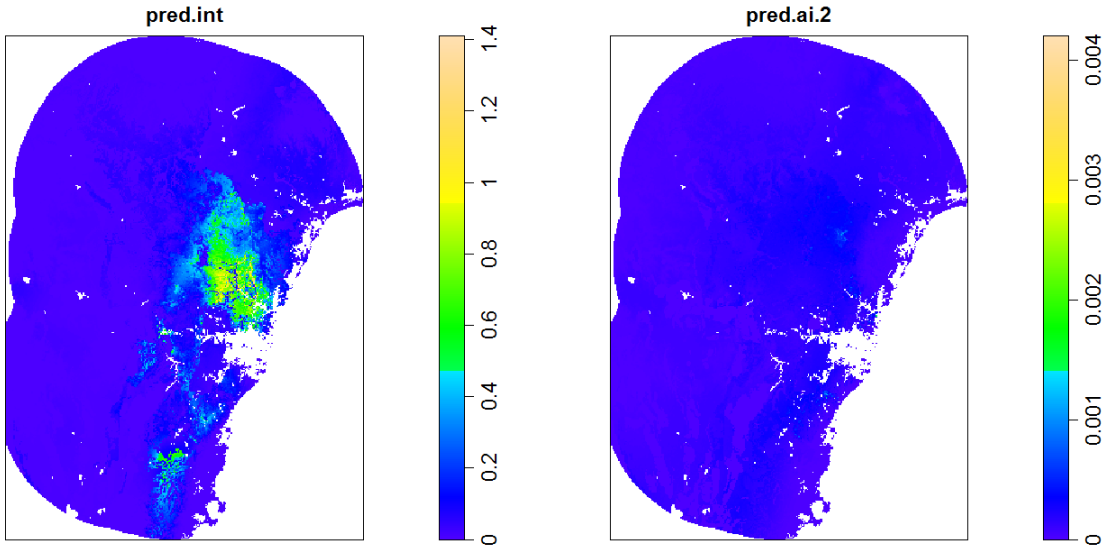


Figure 1: Maps of the spatial trend for the model `ft.int` (left) and `ft.ai.2` (right). The distance variables have been set to zero to account for observer bias.

3.9 Leverage and Influence

The `leverage` and `influence` functions provide insight into places and observed point locations in the study region that have potentially high impact on the fitted model:

```
>plot(leverage(ft.int)) #Fig. 2 (left)
>plot(influence(ft.int)) #Fig. 2 (right)
```

The interpretation is slightly different – places near the coast appear to have high leverage, suggesting that the fitted intensity would be most greatly increased if a species were to be observed there. Points with high influence suggest that if they were deleted, the change in maximum likelihood would be the greatest.

3.10 Validating the form of covariates

The `parres` function validates the parametric form of covariates by computing smoothed partial residuals, potentially suggesting a transformation if the fitted residuals deviate from the confidence bounds. Figure 3 shows the partial residual plot for maximum temperature `V3` in the fitted PPM `ft.int`, produced as follows:

```
>plot(parres(ft.int, "V3")) #Fig. 3
```

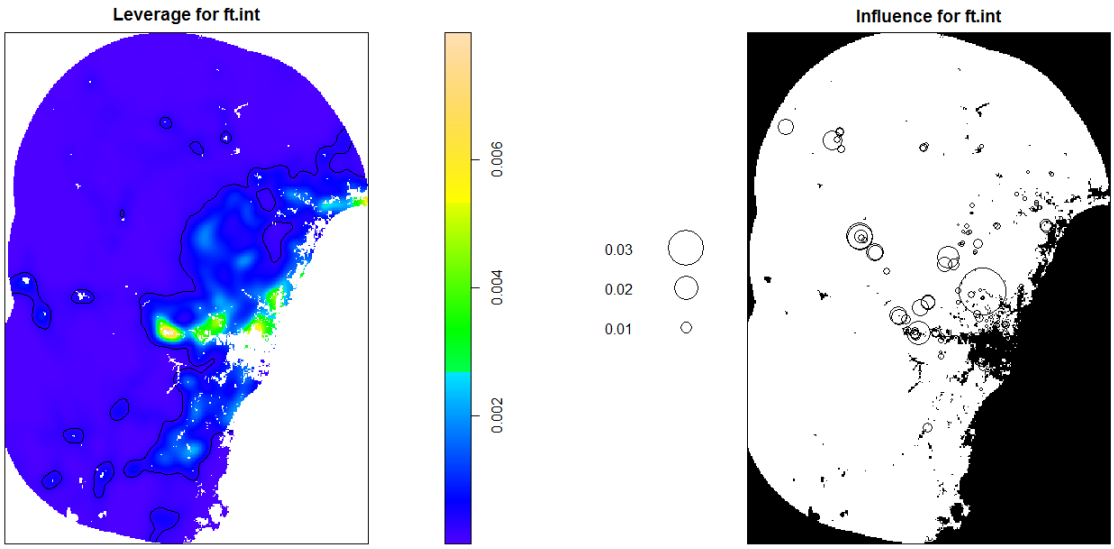


Figure 2: Maps of the leverage (left) and influence (right) of `ft.int`.

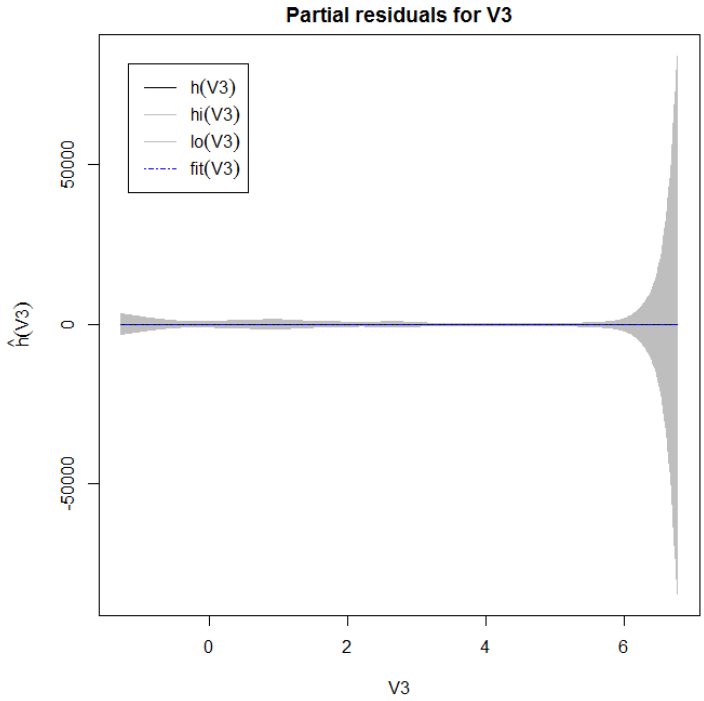


Figure 3: Smoothed partial residuals for the covariate "V3" (maximum temperature) for `ft.int`. The observed residuals fall well within the confidence bounds, suggesting that V3 does not need to be transformed.

Because the observed residuals follow the expected trend, there is no misspecification of maximum temperature. Similar plots for all 19 continuous covariates suggest that the model `ft.int` is properly specified.

4 ppmlasso

The `spatstat` package fits point process regression models by maximum likelihood, but there has been recent interest in regularised estimation of species distribution models, *e.g.* using the LASSO (Phillips *et al.*, 2006; Reineking & Schröder, 2006). The `ppmlasso` package on `R` (Renner & Warton, 2013) is a companion package to `spatstat` that permits coefficient penalisation via LASSO, adaptive LASSO (Zou, 2006), and elastic net (Zou & Hastie, 2005), while retaining much of the functionality originally available in the `spatstat` package. Various criteria are available for choosing how large the penalty should be, including AIC, BIC, and generalised cross validation (GCV). The same model is fitted as in `spatstat` – assuming intensity is a loglinear function of covariates – but `ppmlasso` fits PPMs via penalised likelihood, a method which is more numerically stable and robust to overfitting. The addition of a penalty term introduces bias but reduces variance (Hastie *et al.*, 2009) in an attempt to improve predictive performance.

4.1 Finding the appropriate spatial resolution for analysis

The `findres` function allows users to judge which spatial resolution to use for analysis based on convergence of the log-likelihood. Figure 2a of the main text was produced using the following code:

```
>library(ppmlasso) #Version 1.1
>load("Quad100m.RData") #Contains quad
>load("Eucalyptus sparsifolia Atlas 2012.RData") #Contains X and Y
>sp.xy = data.frame(X, Y)
>ppm.form = ~ poly(FC, MNT, MXT, Rain, degree = 2, raw = TRUE)
 + poly(sqrt(D.Main), sqrt(D.Urb), degree = 2, raw = TRUE)
 + soil
>scales = c(0.5, 1, 2, 4, 8, 16)
>findres(scales, formula = ppm.form, sp.xy = sp.xy, env.grid = quad)
```

This function suggests a spatial resolution of 1km is appropriate for analysis. For the categorical soil variable, a binary covariate needs to be made for each soil type. Version 1.1 of `ppmlasso` performs this step automatically.

4.2 Fitting a regularisation path of point process models

A regularisation path of Poisson point process models or area-interaction models may be fitted with a call to the `ppmlasso` function.

A Poisson PPM with a LASSO penalty that optimises non-linear GCV (Fu, 2005) can be fitted using the following line:

```
>quad.1k = sample.quad(quad, 1)
>ppm.fit = ppmlasso(ppm.form, sp.xy = sp.xy, env.grid = quad.1k,
 sp.scale = 1, criterion = "nlgcv")
```

The LASSO penalty forces uninformative covariates out of the model, so the process of model-fitting automatically does variable selection. This particular fit set three coefficients to zero, including the interaction term between fire count and annual rainfall, suggesting (as previously) that this interaction was not helpful for predicting the location of *Eucalyptus sparsifolia*, after other environmental variables.

4.3 Block cross-validation

Version 1.1 of `ppmlasso` includes block cross-validation as a method for choosing the LASSO penalty. The area-interaction model with radius 2km and LASSO penalty chosen by 5-fold block cross-validation in the main text was fitted as follows:

```
>final.fit = ppmlasso(ppm.form, sp.xy = sp.xy, env.grid = quad.1k,
 sp.scale = 1, criterion = "blockCV", n.blocks = 5, block.size = 32)
```

The fitted coefficients of this model `final.fit` are presented in Table 3.

5 Weighted GLMs

5.1 Infinitely Weighted Logistic Regression (IWLR)

Because the Poisson distribution approximates the binomial when probabilities are small, Fithian & Hastie (2013) suggested assigning large weights W to quadrature points to ensure predicted probabilities are small and approximate equivalence holds between logistic regression and a Poisson PPM, thereby validating use of logistic regression as an

estimation procedure for a Poisson PPM. To fit an IWLR model in `R` requires only two lines of code. For $W = 10^6$:

```
> up.wt = (10^6)^(1 - Pres)
> iwlr = glm(Pres ~ X.des, family = binomial(), weights = up.wt)
```

Here, `Pres` is a binary vector indicating whether a location corresponds to a species location or a quadrature point, and `X.des` is the corresponding design matrix of covariates at species locations and quadrature points. Strictly speaking, `spatstat` and `ppmlasso` are also simple to fit, requiring a single line of code, but the difference here is that a generic function (`glm`) is used to do the work which will be already familiar to most users. However, with IWLR the onus is on the user to generate a sample of quadrature points (randomly, along a regular mesh or through some other scheme) in order to supply `Pres` and `X.des` to the `glm` call. While a `glm` was used above, Fithian & Hastie (2013) argue that any regression function designed for presence-absence data could be used in its place, provided that it permits weighted observations.

A disadvantage of IWLR is that it only gives a solution proportional to a point process because the intercept term is arbitrarily reduced by $\ln W$, and hence the scale of the log-likelihood is likewise arbitrary. Thus we can no longer use the method of choosing quadrature points previously advocated, via likelihood convergence, without suitable adjustments.

5.2 Downweighted Poisson Regression (DWPR)

We propose "downweighted Poisson regression" (DWPR) as a modification of IWLR that addresses these drawbacks. For DWPR, we set the weights equal to some small value (ϵ) at presence locations, but at quadrature points we set the weights equal to the area of the study region (in our case, 86,227km^2) divided by the number of quadrature points:

```
> p.wt = rep(1.e-6, length(Pres))
> p.wt[Pres == 0] = 86227/sum(Pres == 0)
> dwpr = glm(Pres/p.wt ~ X.des, family = poisson(), weights = p.wt)
```

This has similar advantages to IWLR, but additionally estimates the intercept term of the PPM correctly, and hence also fitted values and the log-likelihood, without the problem

of scale dependence. We can then use this formulation to look at the question of how many randomly chosen quadrature points are needed to ensure likelihood convergence.

5.3 Assessing the variability in likelihood for different numbers of quadrature points

In Figure 2b of the main text, increasing numbers of randomly sampled quadrature points were used to fit PPMs using DWPR to judge the number of quadrature points for which the likelihood reasonably converged.

To generate subsets of quadrature points of increasing size:

```
>load("Quad100m.RData") #Contains bigquad
>n.quad = c(1000, 2000, 5000, 10000, 20000, 50000, 100000,
 200000, 500000)
>quad.inc = sample(1:dim(bigquad)[1], 1000)
>assign(paste("quad.", n.quad[1], sep = ""), bigquad[quad.inc[1:n.quad[1]],])
>for (i in 2:length(n.quad))
>{
>quad.inc = c(quad.inc, sample(setdiff(1:dim(bigquad)[1], quad.inc),
 (n.quad[i] - n.quad[i - 1])))
>assign(paste("quad.", n.quad[i], sep = ""), bigquad[quad.inc[1:n.quad[i]],])
>}
```

To compare the likelihood of PPMs fitted using downweighted Poisson regression:

```
>load("Eucalyptus sparsifolia Atlas 2012.RData") # Species Data
>sp.dat = data.frame(X, Y, D.Main, D.Urb, FC, MNT, MXT, Rain, soil)
>sp.dat$Pres = 1
>loglik = rep(NA, length(n.quad))
>for (i in 1:length(n.quad))
>{
>quad = get(paste("quad.", n.quad[i], sep = ""))
>quad$Pres = 0
>all.dat = data.frame(rbind(sp.dat, quad))
>X.des = as.matrix(cbind(poly(all.dat$FC, all.dat$MNT, all.dat$MXT,
 all.dat$Rain, degree = 2, raw = TRUE), poly(sqrt(all.dat$D.Main),
```

```
 sqrt(all.dat$D.Urb), degree = 2), all.dat$soil))
>p.wt = rep(1.e-8, dim(all.dat)[1])
>p.wt[all.dat$Pres == 0] = 86227/n.quad[i]
>z = all.dat$Pres/p.wt
>dwpr = glm(z ~ X.des, family = poisson(), weights = p.wt)
>mu = dwpr$fitted
>loglik[i] = sum(p.wt*(z*log(mu) - mu))
>}
>plot(n.quad, loglik, log = "x", type = "o")
```

6 MAXENT

When fitting a PPM using MAXENT, users need to adjust the default settings. Here we outline which settings best mimic the methods presented so far. Since Poisson PPMs are modelling intensity, the number of presence records is of interest. Hence if MAXENT is run with gridded environmental data, the default "Settings" option that reduces presence records to one per grid cell ("remove duplicate presence records") needs to be unchecked. Alternatively if the samples with data ("SWD") mode is used, the defaults are correct for PPMs: *i.e.* all points are retained. The raw output (not the default logistic output) can then be interpreted as a relative intensity, proportional to a Poisson PPM intensity. The maximum number of quadrature points ("background samples") can be specified in the Settings window (though MAXENT will not put more than one point per grid cell), or any number of background points can be provided in SWD mode. The sufficiency of the number of points can be evaluated using the "gain". The standard form of MAXENT's log-likelihood involves a sum, analogous to the integral of equation 4 of the main text. A constant offset to the log-likelihood turns the sum into an average, which is independent of number of quadrature points; this value is reported as the "Regularised training gain" in MAXENT's output (in the html file, the maxentResults.csv file and the maxent.log file). This gain could be compared between different numbers of quadrature points – changes of more than some small amount could be taken to indicate that the model is changing with different numbers of points. Alternatively, changes in the response curves, variable importance, or predictive performance across different numbers of quadrature points could be assessed.

The user also has the ability to adjust a number of settings, including the types of terms

to include (linear, quadratic, product, hinge, and threshold), the convergence threshold for the model fit, the degree of regularisation via the LASSO penalty and other features to fine-tune the model. In order to use the "explain tool", only linear, quadratic, and hinge terms may be included. However, there is no capacity for the model complexity (feature types and LASSO penalty) to be chosen in a data-driven manner as with `glmnet` or `ppmlasso`. Standard errors, information on residuals, capacity to explore clustering and other features of the previously described methods are also missing.

Further features of MAXENT not explored here include the ability to resample and subsample presence locations for bootstrapping and cross-validation for evaluation purposes. Whilst these cannot be implemented in spatial blocks (Wenger & Olden, 2012) from the interface, MAXENT can be run from R using the package `dismo`, so users could code their own specialised evaluation procedures. Comprehensive guides to MAXENT software are available with the program and elsewhere (Elith *et al.*, 2011; Merow *et al.*, 2013).

6.1 `dismo` package

Below we present some code for the `dismo` package for fitting a PPM.

6.1.1 Setting up to run Maxent from `R`

To run Maxent from R you need to install the package dismo and download maxent from Princeton University. Look at the help files for maxent within dismo, and here are some additional hints:

```
>library(dismo)

# find whether the maxent.jar file is in the correct place:
>jar <- paste(system.file(package="dismo"), "/java/maxent.jar", sep='')
>file.exists(jar)
```

If dismo does not think MaxEnt exists, manually put it (i.e. maxent.jar) in the right place by finding where R and dismo are installed. On many pcs it will be something like this:

```
C:\Program Files\R\R-3.0.1\library\dismo
```

On macs the R library is in the `/Library/Frameworks` directory - this is in the highest level Library folder - eg on an OSX MacBook pro it was here:
`/Library/Frameworks/R.framework/Versions/3.1/Resources/library/dismo/java`

For either PC or mac, within the dismo directory find the `java` directory and copy maxent.jar there. Then rerun this:

```
>file.exists(jar)  #the result should be TRUE
```

6.1.2 Preparing data

First set the working directory in whatever way you do that - we'll specify a name for the working directory:

```
>wd <- getwd()
```

Assuming you have gridded environmental data in a folder called "grids" in some format that can be read by the package raster. Also assuming that it is "masked" so any regions not being considered have "nodata" values.

Specify the names of the rasters – *e.g.* using variable names already introduced:

```
>vars <- c("FC", "MNT", "MXT", "Rain", "D.Main.sqrt", "D.Urb.sqrt", "soil")
```

Read these into a "stack", which is a collection of rasters with exactly the same extent, cell size etc:

```
>nsw.stack <- stack(file.path(wd, "grids/", vars))
```

To visualise the rasters:

```
>plot(nsw.stack)
```

Read in location records for the species in one of the format required for for `maxent` function - here we assume a .csv file with 2 columns of data specifying `x` (longitude) and `y` (latitude) for the sites (one row per site):

```
>locs <- read.csv(file.path(wd, "locs.csv"))
```

6.1.3 Fitting models and predicting

In `dismo`, Maxent models are fitted and predicted in 2 steps, as is common for many other modelling methods such as GLMs. The file format for inputs is described in `dismo`'s help file for the function `maxent`:

```
?maxent
```

The arguments that the `maxent` function understands are identical to those in the set of flags that Maxent (the java program) understands – if you're not familiar with them, look at the help file for Maxent from the Maxent interface.

Since maxent-in-dismo implements fitting and predicting in 2 steps, only those arguments relevant to fitting will operate in the first step, and similarly for prediction.

Below is some example code with settings correct for point process models (note: data are not supplied; the code is indicative).

Fit a first `maxent` model:

```
>max1 <- maxent(x=nsw.stack, p=locs, path=paste(wd,'/max1',sep=""),
 args=c("-P", "noautofeature", "nothreshold", "noproduct", -t soil,
 "maximumbackground=40000","noaddsamplestobackground","noremoveduplicates"))
```

The "path" as used above creates an output directory and will put the model outputs in there. If you don't specify the path, the `maxent` results will only be temporary files and you will lose them at the end of the session "args" above are, respectively (within the brackets): plot response curves, turn off autofeatures, turn off threshold features, turn off product features, toggle soil to categorical, take 40000 random background points, don't add samples to background, don't remove duplicate records from gridcells. Note the last 2 arguments are essential for the equivalence to a point process model. Any number of background points can be specified (though no more than one per grid cell will be sampled).

To see the results in a browser:

```
>max1
```

Plot showing importance of each variable:

```
>plot(max1)
```

To fit the same model but using SWD (SamplesWithData) format: first prepare some data:

```
>pres.env <- extract(nsw.stack, locs) #get the environmental data at
#presence points
>bg <- randomPoints(nsw.stack, 40000) #get 40,000 locations (or however
#many you want) in region, randomly selected from any cells with data
>bg.env <- extract(nsw.stack, bg)
>pbg.env <- rbind(pres.env, bg.env) # rowbind presence and bg
#environments
>pbg.which <- c(rep(1, nrow(locs)), rep(0, nrow(bg))) # a vector of
#1's and 0's to keep track of which rows are for presence (1) and
#which for bg (0)
```

Now for the SWD model:

```
>max1SWD <- maxent(x=pbg.env, p=pbg.which, path=paste(wd,'/max1SWD',sep=""),
 args=c("-P", "noautofeature", "nothreshold", "noproduct", -t soil,
 "maximumbackground=40000","noaddsamplestobackground"))
```

Note that this time you don't have to specify "`noremoveduplicates`" since the SWD format is ignorant of grid cells.

To predict: – *e.g.* here, to rasters:

If you are using observer bias covariates you'll need to make new rasters for the distance-to.. variables, and set these to a constant value. See help for the raster package for how to set all cells to one value. Assume we have a new stack with variables with the same name, but where the distance to.. variables are replaced with the constant-valued ones (call it `predict.stack`).

```
>max1.pred <- predict(max1, predict.stack, args="outputformat=raw",
 filename=paste(wd, '/max1/pred.asc', sep=""), format="ascii",
 progress="text")
```

Predictions will be in Maxent's raw format (correct for point process models, and different to the default logistic output), and here we are writing an .ascii file (other formats are possible).

7 INLA

7.1 Constructing the mesh and defining the SPDE

A key initial step in the R-INLA approach is to divide the study region into a series of triangles whose vertices form a "mesh" of points used in fitting. These triangular segments perform a similar role as grid cells, although there is no requirement that they be arranged in a regular rectangular grid.

We found that results were better when we defined a boundary for the study region before constructing the mesh. This can be done in R-INLA by creating a concave hull around the species locations and quadrature points (collected in a matrix `data.xy`) to serve as the boundary via the `inla.nonconvex.hull` function. The `convex` argument controls how close the hull is to its interior points and the `resolution` argument determines the number of points comprising the hull.

```
>hull = inla.nonconvex.hull(as.matrix(data.xy), convex = -0.01,
 resolution = 800)
```

We may now construct the mesh using `hull` as a boundary, with the `cutoff`, `offset` and `max.edge` arguments controlling the minimum distance between unique point locations, the extent of an inner and outer extension, and the maximum allowed edge length of triangles in the inner and outer extensions, respectively. The goal is to construct a set of triangles without small interior angles and long edges (Lindgren *et al.*, 2011).

For non-rectangular study regions, we make use of approximate solutions to stochastic partial differential equations (SPDEs, Lindgren *et al.*, 2011) for faster model-fitting. We use the constructed mesh to define the SPDE for the Gaussian field, setting initial prior parameters for the Gaussian field according to guidelines provided by R-INLA.

```
>mesh = inla.mesh.2d(boundary = hull, cutoff = 3, offset = c(20, 40),
 max.edge = c(6, 9))
>sigma0 = 0.05 ## field std.dev.
>range0 = 20
>kappa0 = sqrt(8)/range0
>tau0 = 1/(sqrt(4*pi)*kappa0*sigma0)
>spde = inla.spde2.matern(mesh, B.tau = cbind(log(tau0), -1, 1),
```

```
B.kappa = cbind(log(kappa0), 0, -1), theta.prior.mean = c(0, 0),
theta.prior.prec = c(0.1, 1))
```

The resulting mesh is illustrated in Figure 4, with the concave hull represented by the blue polygon and locations of *Eucalyptus sparsifolia* represented by red dots. We found that output was reasonably robust to prior specification, with fitted models using combinations of four values each for `sigma0` (0.01, 0.02, 0.05, 0.1) and `range0` (2, 10, 20, 40) producing largely similar results.

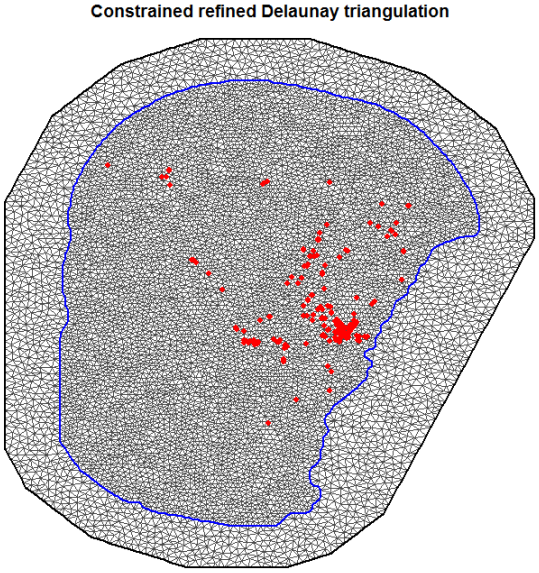


Figure 4: The mesh created in `R-INLA` for fitting the model with latent Gaussian field. The blue polygon represents the concave hull around the study region, and the red dots represent locations of *Eucalyptus sparsifolia*.

7.2 Projection matrices

Having constructed a mesh, we then create matrices that project the latent field from the mesh points to the species locations (`sp.xy`) and quadrature points (`quad.xy`):

```
>sp.mat = inla.spde.make.A(mesh, sp.xy)
>quad.mat = inla.spde.make.A(mesh, quad.xy)
```

7.3 Defining response

After defining the projection matrices `sp.mat` and `quad.mat` between the mesh and the species locations and quadrature points, respectively, we specify the covariates to be considered in the model by constructing data "stacks". Note that the `y` covariate indicates presence or absence, the `e` covariate is the expected number of presences, with 0 at species locations and the square of the spatial resolution at quadrature points, and the `dist` covariate is the nearest distance to a presence point (here defined by `pt.d` for species presences and `quad.d` for quadrature points).

```
>attach(all.dat)
>stk.sp = inla.stack(data = list(y = 1, e = 0), A = list(sp.mat, 1),
 tag = 'sp', effects = list(list(i = 1:mesh$n), data.frame(Intercept = 1,
 FC = FC[is.sp], MNT = MNT[is.sp], MXT = MXT[is.sp], Rain = Rain[is.sp],
 FC2 = FC[is.sp]^2, MNT2 = MNT[is.sp]^2, MXT2 = MXT[is.sp]^2,
 Rain2 = Rain[is.sp]^2, FC.MNT = FC[is.sp]*MNT[is.sp],
 FC.MXT = FC[is.sp]*MXT[is.sp], FC.Rain = FC[is.sp]*Rain[is.sp],
 MNT.MXT = MNT[is.sp]*MXT[is.sp], MNT.Rain = MNT[is.sp]*Rain[is.sp],
 MXT.Rain = MXT[is.sp]*Rain[is.sp], D.Main = sqrt(D.Main[is.sp]),
 D.Urb = sqrt(D.Urb[is.sp]), D.Main2 = sqrt(D.Main[is.sp])^2,
 D.Urb2 = sqrt(D.Urb[is.sp])^2,
 D.Main.D.Urb = sqrt(D.Main[is.sp])*sqrt(D.Urb[is.sp]),
 soil = soil[is.sp], dist = pt.d)))
>stk.quad = inla.stack(data = list(y = 0, e = spat.res^2),
 A = list(quad.mat, 1), tag = 'quad', effects = list(list(i = 1:mesh$n),
 data.frame(Intercept = 1, FC = FC[is.quad], MNT = MNT[is.quad],
 MXT = MXT[is.quad], Rain = Rain[is.quad], FC2 = FC[is.quad]^2,
 MNT2 = MNT[is.quad]^2, MXT2 = MXT[is.quad]^2, Rain2 = Rain[is.quad]^2,
 FC.MNT = FC[is.quad]*MNT[is.quad], FC.MXT = FC[is.quad]*MXT[is.quad],
 FC.Rain = FC[is.quad]*Rain[is.quad], MNT.MXT = MNT[is.quad]*MXT[is.quad],
 MNT.Rain = MNT[is.quad]*Rain[is.quad], MXT.Rain = MXT[is.quad]*Rain[is.quad],
 D.Main = sqrt(D.Main[is.quad]), D.Urb = sqrt(D.Urb[is.quad]),
 D.Main2 = sqrt(D.Main[is.quad])^2, D.Urb2 = sqrt(D.Urb[is.quad])^2,
 D.Main.D.Urb = sqrt(D.Main[is.quad])*sqrt(D.Urb[is.quad]),
 soil = soil[is.quad], dist = quad.d)))
>stk.all = inla.stack(stk.sp.0, stk.q.0)
```

7.4 Fitting models

Having prepared the data for fitting, we fit a model containing environmental and distance-based terms, an interaction covariate ("constructed covariate", Illian *et al.*, 2013) to capture point clustering or repulsion, and a random Gaussian field:

```
>ft.inla = inla(y ~ 0 + Intercept + FC + MNT + MXT + Rain + FC2 + MNT2 + MXT2
 + Rain2 + FC.MNT + FC.MXT + FC.Rain + MNT.MXT + MNT.Rain + MXT.Rain + D.Main
 + D.Urb + D.Main2 + D.Urb2 + D.Main.D.Urb + soil
 + f(inla.group(dist, n = 50, method = "quantile"), model = "rw1",
 scale.model = TRUE) + f(i, model = spde), family = 'poisson',
 data = inla.stack.data(stk.all),
 control.predictor = list(A = inla.stack.A(stk.all), compute = TRUE),
 E = inla.stack.data(stk.all)$e, control.compute = list(dic = TRUE),
 control.fixed = list(expand.factor.strategy = 'inla'))
```

The fitted coefficients of `ft.inla` are presented in Table 3.

We can project the mean and standard deviation of the latent field onto quadrature points using the `inla.mesh.projector` command as in Figure 4 of the main text:

```
>proj.quad = inla.mesh.projector(mesh, quad.xy)
>gf.mean = inla.mesh.project(proj.quad, ft.inla$summary.random$i$mean)
>gf.sd = inla.mesh.project(proj.quad, ft.inla$summary.random$i$sd)
```

We can produce a plot of the effect of the interaction covariate `dist` as follows:

```
>plot(ft.inla$summary.random[[2]][, 1:2], type = 'l',
 xlab = 'dist (km)', ylab = 'f(dist)'); abline(h = 0, lty = 3)
>for (i in c(4, 6)) lines(ft.inla$summary.random[[2]][,c(1, i)], lty = 2)
```

8 lgcp

First we need to define a boundary for the study region and build a point pattern object which includes the region boundary and presence points

```
>polygon <- data.frame(x=polyX,y=polyY)
>sd <- ppp(sp.dat$X,sp.dat$Y,poly=polygon)
```

where `polyX` and `polyY` are the co-ordinates of the boundary of the study region, and `sp.dat` are the data. Next we choose the cell width. To do this we can use the `minimum.contrast` function to find least squares estimates of model parameters. This gives us an idea of the extent of spatial variation via the 'scale' estimate. The `chooseCellwidth` function allows us to choose an appropriate cell width to optimize computational resource use. See (Taylor *et al.*, 2013) for further detail.

```
>minimum.contrast(sd, model = "exponential", method = "g",intens = density(sd),
 transform = log)
>chooseCellwidth(sd, cwinit=X)
>Cellwidth <- X
```

We will assume we have covariate information in a data frame `quad` which includes all relevant covariates as well as `X` and `Y` values in a grid inside the boundary of the study area. We then create the computational grid on which inference takes place. Note that the `getpolyol` is quite slow, and it is best to save the resulting object so that it can be used in multiple runs.

```
>covar=SpatialPixelsDataFrame(cbind(quad$X,quad$Y),quad)
>polyolay <- getpolyol(data = sd, pixelcovariates = covar,
 cellwidth = Cellwidth)
```

We also need to check the interpolation of covariates is accurate. Numeric variables are assigned interpolation by areal weighted mean while factor, character and other types of variable are assigned interpolation by majority vote. If any variables are incorrectly interpolated we can change the using the `assigninterp` function

```
>covar@data=guessinterp(covar@data)
>covar@data <- assigninterp(df = covar@data, vars "variable.name",
 value = "ArealWeightedSum")
```

We then need to create the Z matrix of covariates projected onto the computational grid. We do this using the `getZmat` function.

```
>Zmat <- getZmat(formula = X ~ ...,data = sd, pixelcovariates = covar,
 cellwidth = Cellwidth, overl = polyolay)
```

We can choose from a number of flexible spatial covariance functions, see Appendix C of (Taylor *et al.*, 2013) for detail, for simplicity we choose the exponential.

```
>cf <- CovFunction(exponentialCovFct)
```

The last thing we need to specify are the priors for all out parameters. To specify priors we can use the `lgcpPrior`

```
>priors <-lgcpPrior( etaprior =
 PriorSpec(LogGaussianPrior(mean = log(c(5, 5)),variance = diag(c(5, 0.5)))),
 betaprior = PriorSpec(GaussianPrior(mean = rep(0, nbeta),
 variance = diag(rep(10^3, nbeta)))))
```

where `nbeta` is the number of predictors plus 1. The running of the MCMC code can be quite sensitive to prior choice. In particular if there are warnings during the running of the code, the prior for η may have to be tightened, refer to (Taylor *et al.*, 2013) for guidance. Now all that remains is to choose the length of the MCMC chain, the desired burn in and the number of iterations to thin by and run the chain. The length of time required by the MCMC depends on the length of the chain. We suggest you first choose small values to test the code is working.

```
>lbr=c(1000000,100000,1000) #length,burn-in and thin
>ft.lgcp=lgcpPredictSpatialPlusPars(formula = X ~ ...,sd = sd,Zmat = Zmat,
 model.priors = Priors, model.inits = NULL ,spatial.covmodel = cf,
 cellwidth = Cellwidth,poisson.offset = NULL,
 mcmc.control = mcmcpars(mala.length = lbr[1], burnin = lbr[2],retain = lbr[3],
 adaptivescheme = andrieuthomsh(inith = 1,
 alpha = 0.5, C = 1, targetacceptance = 0.574)))
```

The fitted coefficients of `ft.lgcp` are presented in Table 3.

9 Fitted Coefficients

A list of fitted coefficients appears in Table 3 for models fitted in `spatstat`, `ppmlasso`, `R-INLA`, and `lgcp`. The model fitted by `spatstat` did not account for spatial dependence, unlike the models fitted by `ppmlasso`, `R-INLA` and `lgcp`. We suspect that the latent

Gaussian fields of the Cox process models fitted by `R-INLA` and `lgcp` were correlated with the environmental predictors, and hence those models identified fewer significant covariates.

References

Baddeley, A. & Turner, R. (2005). Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software*, **12**, 1–42.

Cressie, N. A. C. (1993). *Statistics for Spatial Data*. John Wiley & Sons, New York.

Daley, D. J. & Vere-Jones, D. (1988). *An Introduction to the Theory of Point Processes*. Springer, New York.

Diggle, P. (2003). *Statistical Analysis of Spatial Point Patterns*. Oxford University Press, New York.

Elith, J., Phillips, S. J., Hastie, T., Dudík, M., Chee, Y. E. & Yates, C. J. (2011). A statistical explanation of MaxEnt for ecologists. *Diversity and Distributions*, **17**, 43–57.

Fithian, W. & Hastie, T. (2013). Finite-sample equivalence in statistical models for presence-only data. *The Annals of Applied Statistics*, **7**, 1917–1939.

Fu, W. J. (2005). Nonlinear GCV and quasi-GCV for shrinkage models. *Journal of Statistical Planning and Inference*, **131**, 333–347.

Hastie, H., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York.

Illian, J. B., Martino, S., Sørbye, S. H., Gallego-Fernández, J. B., Zunzunegui, M., Esquivias, M. P. & Travis, J. M. (2013). Fitting complex ecological point process models with integrated nested Laplace approximation. *Methods in Ecology and Evolution*, **4**, 305–315.

Lindgren, F., Rue, H. & Lindström, J. (2011). An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **73**, 423–498.

Merow, C., Smith, M. J. & Silander, J. A. (2013). A practical guide to MaxEnt for modeling species' distributions: what it does, and why inputs and settings matter. *Ecography*, **36**, 1058–1069.

Phillips, S. J., Anderson, R. P. & Schapire, R. E. (2006). Maximum entropy modeling of species geographic distributions. *Ecological Modelling*, **190**, 231–259.

Reineking, B. & Schröder, B. (2006). Constrain to perform: regularization of habitat models. *Ecological Modelling*, **193**, 675–690.

Renner, I. W. & Warton, D. I. (2013). Equivalence of MAXENT and Poisson point process models for species distribution modeling in ecology. *Biometrics*, **69**, 274–281.

Taylor, B. M., Davies, T. M., Rowlingson, B. S. & Diggle, P. J. (2013). lgcp: An R Package for Inference with Spatial and Spatio-Temporal Log-Gaussian Cox Processes. *Journal of Statistical Software*, **52**.

Wenger, S. J. & Olden, J. D. (2012). Assessing transferability of ecological models: an underappreciated aspect of statistical validation. *Methods in Ecology and Evolution*, **3**, 260–267.

Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, **101**, 1418–1429.

Zou, H. & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, **67**, 301–320.

Table 3: Estimated coefficients for models fitted in `spatstat` (`ft.int`), the area-interaction model of radius 2km with LASSO penalty chosen via block cross validation in `ppmlasso` (`final.fit`), `R-INLA` (`ft.inla`) and `lgcp` (`ft.lgcp`). Coefficients in bold are significantly different from zero (available for `spatstat`, `R-INLA` and `lgcp`. Note that `spatstat` uses soil type 11 as a reference covariate, and hence it has no estimated coefficient.

Covariate	spatstat	ppmlasso	R-INLA	lgcp
Intercept	**-715.838**	-8.024	-18.851	-11.58
soil type 2	-15.663	-0.070	-21.263	-12.31
soil type 5	0.120	0.074	**34.238**	**17.31**
soil type 6	-13.550	-0.420	-19.116	-11.85
soil type 7	**1.888**	0.288	**34.795**	**18.38**
soil type 8	0.029	-0.038	**34.086**	15.66
soil type 11	NA	0	**34.314**	**17.90**
soil type 12	-0.690	0.073	**34.474**	-13.08
soil type 15	-14.178	-0.365	-17.369	-12.21
$\sqrt{\texttt{D.Main}}$	**-0.598**	0	-0.417	**-0.362**
$\sqrt{\texttt{D.Urb}}$	**-1.148**	-0.604	**-1.053**	0.017
D.Main	-0.137	-0.964	-0.131	-0.014
D.Urb	**0.093**	0.728	**0.101**	-0.003
$\sqrt{\texttt{D.Main*D.Urb}}$	0.133	0	0.130	**0.012**
FC	-3.391	0	0.010	-7.809
MNT	**-29.192**	0	-0.775	-1.845
MXT	**43.544**	0	-0.884	-0.064
Rain	**326.375**	0	-15.158	-7.581
$\texttt{FC}^2$	**-0.122**	-0.295	-0.000	0.042
$\texttt{MNT}^2$	**-0.400**	-0.217	-0.159	-0.244
$\texttt{MXT}^2$	**-0.676**	-0.228	0.000	-0.021
$\texttt{Rain}^2$	**-41.458**	0	2.703	-2.153
FC*MNT	**0.188**	-0.027	0.040	-0.114
FC*MXT	0.112	0.439	0.018	0.250
FC*Rain	0.146	-0.080	-0.607	1.180
MNT*MXT	**0.947**	0.848	0.064	0.161
MNT*Rain	**5.725**	-0.509	-0.280	-0.630
MXT*Rain	**-9.511**	0	0.439	0.517